Alcohol Beverage Services (ABS)

2025-12-03

1. Libraries & Global Options

This analysis integrates three primary data sources to evaluate retail alcohol sales performance at the store level. The core dataset consists of line-level transaction records, capturing individual purchase details including product classification, quantity, net sales amount, store location, and transaction date. This transactional data is supplemented with store-level attributes, such as physical characteristics and operational designations, and with trade-area demographic data describing the population surrounding each store.

Combining these sources enables analysis at multiple levels, including product, transaction, store, and temporal dimensions, and provides the foundation for both descriptive analysis and subsequent modeling.

```
# Core data manipulation, dates, strings, plotting, reading
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.5.1
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 4.5.1
```

```
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 4.5.1
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.5.1
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.5.1
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 4.5.1
```

```
library(purrr)
```

```
## Warning: package 'purrr' was built under R version 4.5.1
```

```
# Modeling, bootstrapping, formatting, labels
library(broom)
```

```
## Warning: package 'broom' was built under R version 4.5.1
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.5.1
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-10
```

```
library(boot)
library(scales)
```

```
## Warning: package 'scales' was built under R version 4.5.1
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
library(ggrepel)
```

```
## Warning: package 'ggrepel' was built under R version 4.5.1
```

```
library(arules)
```

```
## Warning: package 'arules' was built under R version 4.5.1
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.5.2
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

2. Data and Methods

Substantial data cleaning and standardization were required prior to analysis. Transaction-level text fields were trimmed, standardized for casing, and normalized to eliminate inconsistencies in product descriptions, store names, unit sizes, and pack formats. Legacy and inconsistent labeling conventions were reconciled to ensure that equivalent products and units were treated consistently across the dataset.

Non-retail, non-inventory, donation, and invalid records were removed to restrict the analysis to legitimate retail sales activity. Additional validity checks were applied to numeric fields, including quantities and net amounts, and refund transactions were explicitly identified to prevent distortion of unit price calculations. A standardized unit type was defined to support consistent aggregation across beer and non-beer categories.

These steps ensure that the resulting dataset accurately reflects true retail behavior and is suitable for reliable aggregation, comparison, and modeling.

2.1 Data Sources

```
# Line-level transactions
transactions_raw <- read.csv(
  "TransactionData.csv",
  header   = TRUE,
  sep      = ",",
  encoding = "UTF-8",
  quote    = "\""
)

# Store-level attributes (e.g., square footage)
designation  <- read_csv("Designation.csv",
                         show_col_types = FALSE)

# Trade-area demographic attributes
demographics <- read_csv("Demographics.csv",
                         show_col_types = FALSE)
```

2.2 Core Cleaning & Standardization

```r
# Basic trimming, casing, and date conversion
transactions <- transactions_raw %>%
  mutate(
    CLASSIFICATIONDEPARTMENT = trimws(CLASSIFICATIONDEPARTMENT),
    CLASSIFICATIONTYPE       = trimws(CLASSIFICATIONTYPE),
    STORENAME                = trimws(STORENAME),
    SIZE                     = trimws(SIZE),
    PACKUNIT                 = trimws(PACKUNIT),
    TAGDESC                  = trimws(TAGDESC),
    DESCRIPTION = DESCRIPTION %>%
      str_trim() %>%
      str_squish() %>%
      str_to_upper(),
    TRANSDATE = as.Date(TRANSDATE)
  )

# Normalize store names (known rename)
transactions <- transactions %>%
  mutate(
    STORENAME = if_else(
      STORENAME == "White Oak",
      "White Oak Town Center",
      STORENAME
    )
  )

# Normalize legacy size labels
transactions <- transactions %>%
  mutate(
    SIZE = case_when(
      SIZE == "5LTR" ~ "5L",
      SIZE == "3LTR" ~ "3L",
      SIZE == "1LTR" ~ "1L",
      TRUE           ~ SIZE
    )
  )

# Normalize pack units and convert numeric codes to *pk / bottle
transactions <- transactions %>%
  mutate(
    PACKUNIT = case_when(
      str_detect(PACKUNIT, "^[0-9]+$") & PACKUNIT == "1" ~ "Btl",
      str_detect(PACKUNIT, "^[0-9]+$")                  ~ paste0(PACKUNIT, "pk"),
      PACKUNIT == "08pk"                                ~ "8pk",
      PACKUNIT == "06pk"                                ~ "6pk",
      PACKUNIT == "05pk"                                ~ "5pk",
      PACKUNIT == "04pk"                                ~ "4pk",
      PACKUNIT == "03pk"                                ~ "3pk",
      PACKUNIT == "02pk"                                ~ "2pk",
      PACKUNIT == "01pk"                                ~ "Btl",
      TRUE                                              ~ PACKUNIT
    )
  )

# Back fill missing CLASSIFICATIONTYPE for BEER rows
transactions <- transactions %>%
  mutate(
    CLASSIFICATIONTYPE = if_else(
      (CLASSIFICATIONTYPE %in% c("", "NULL") | is.na(CLASSIFICATIONTYPE)) &
        CLASSIFICATIONDEPARTMENT == "BEER",
      CLASSIFICATIONDEPARTMENT,
      CLASSIFICATIONTYPE
    )
  )

# Drop non-retail / dirty / non-stock rows
transactions <- transactions %>%
  filter(
    STORENAME != "Westwood",
    CLASSIFICATIONDEPARTMENT != "DONATIONS",
    CLASSIFICATIONDEPARTMENT != "NON INVENTORY",
    SIZE != "UNIT",
    SIZE != "" & !is.na(SIZE),
    TAGDESC != "NULL" & !is.na(TAGDESC),
    TAGDESC == "STOCK"
  )

# Define UnitType: pack unit for BEER, bottle size otherwise
transactions <- transactions %>%
  mutate(
    UnitType = if_else(
      CLASSIFICATIONDEPARTMENT == "BEER",
      PACKUNIT,
      SIZE
    )
  )

# Define refunds and enforce basic numeric validity
transactions <- transactions %>%
  mutate(
    IsRefund = NETAMOUNT < 0
  ) %>%
  filter(
    is.finite(NETAMOUNT),
    is.finite(LINEQTY),
    LINEQTY > 0
  ) %>%
  filter(
    !is.na(TRANSACTIONID),
    !is.na(STORENAME),
    TRANSACTIONID != "",
    STORENAME     != "",
    !is.na(CLASSIFICATIONDEPARTMENT),
    CLASSIFICATIONDEPARTMENT != ""
  )

# Final description cleaning + unit price (FEE) calculation
transactions <- transactions %>%
  mutate(
    DESCRIPTION = DESCRIPTION %>%
      str_to_upper() %>%
      str_replace_all("[^A-Z0-9\\s]", " ") %>%
      str_squish(),
    FEE = if_else(!IsRefund, NETAMOUNT / LINEQTY, NA_real_)
  )
```

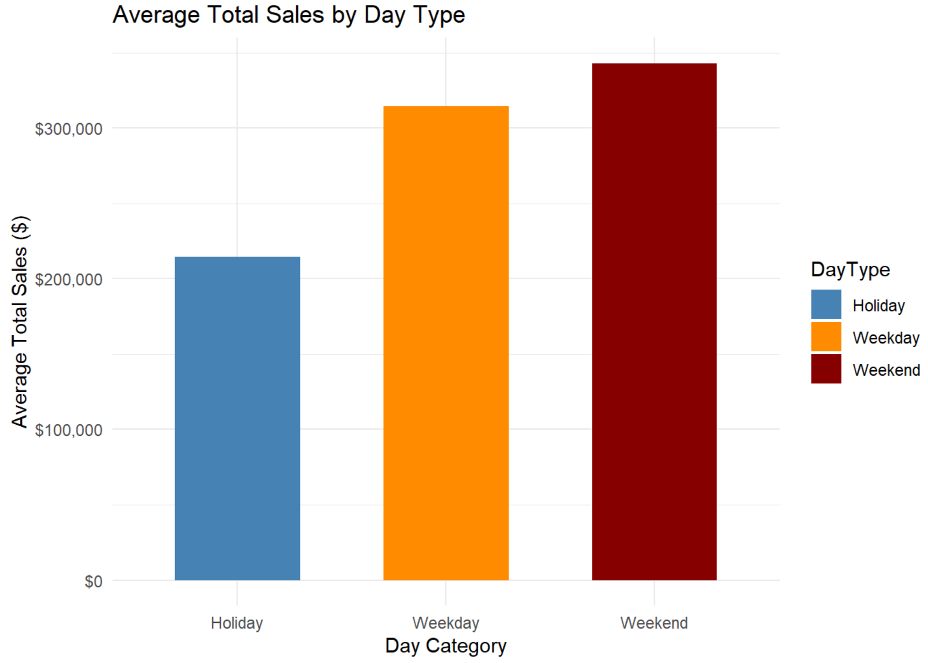
2.3 Years, Holidays, and Day Types

To support time-based analysis, transaction dates were converted to standard date formats and enriched with calendar attributes. Each transaction day was classified as a weekday, weekend, or holiday using a defined U.S. holiday calendar that includes both fixed holidays and extended seasonal periods around year-end.

Daily sales totals were then aggregated and compared across day types to evaluate systematic differences in purchasing behavior. This classification enables assessment of temporal demand patterns and provides insight into how holidays and weekends affect overall sales performance.

The resulting summaries reveal meaningful variation in average daily sales across day categories, highlighting the importance of accounting for calendar effects when evaluating store performance and sales trends.

```r
# Clone for time-series / calendar analysis
transactionsFS <- transactions
transactionsFS$TRANSDATE <- as.Date(transactionsFS$TRANSDATE)

# Define holiday / New Year ranges
ny_range  <- seq(as.Date("2023-12-21"), as.Date("2023-12-24"), by = "day")
ny_range1 <- seq(as.Date("2024-12-21"), as.Date("2024-12-24"), by = "day")
ny_range2 <- seq(as.Date("2023-12-29"), as.Date("2024-01-02"), by = "day")
ny_range3 <- seq(as.Date("2024-12-29"), as.Date("2025-01-02"), by = "day")

# Holiday calendar (fixed + ranges)
us_holidays <- as.Date(c(
  "2023-07-04","2023-09-04","2023-11-23","2023-12-25",
  ny_range, ny_range1,
  "2024-05-27","2024-07-04","2024-09-02","2024-11-28","2024-12-25",
  ny_range2, ny_range3,
  "2025-05-26"
))

# Tag each day as Holiday / Weekend / Weekday
transactionsFS <- transactionsFS %>%
  mutate(
    DayOfWeek = wday(TRANSDATE, label = TRUE),
    DayType   = case_when(
      TRANSDATE %in% us_holidays    ~ "Holiday",
      DayOfWeek %in% c("Sat","Sun") ~ "Weekend",
      TRUE                          ~ "Weekday"
    )
  )

daily_sales <- transactionsFS %>%
  group_by(TRANSDATE, DayType) %>%
  summarise(DailySales = sum(NETAMOUNT, na.rm = TRUE), .groups = "drop")

# Mean per DayType
avg_totals <- daily_sales %>%
  group_by(DayType) %>%
  summarise(AverageDailySales = mean(DailySales), .groups = "drop")
ggplot(avg_totals, aes(x = DayType, y = AverageDailySales, fill = DayType)) +
  geom_col(width = 0.6) +
  labs(
    title = "Average Total Sales by Day Type",
    x = "Day Category",
    y = "Average Total Sales ($)"
  ) +
  scale_y_continuous(labels = scales::dollar) +
  theme_minimal() +
  scale_fill_manual(values = c("steelblue", "darkorange", "darkred"))
```



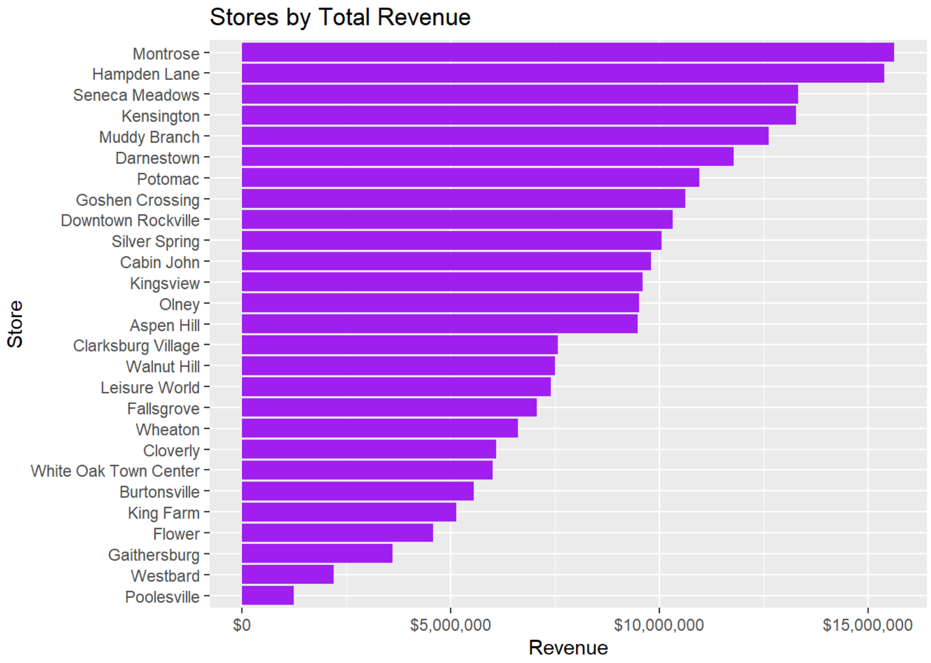
# Interpretation: Sales by Day Type

Average daily sales differ markedly by calendar classification. Holidays generate the highest mean daily revenue, followed by weekends, with weekdays consistently lowest. The magnitude of these differences indicates that calendar effects are not marginal noise but a structural component of demand. This supports explicitly controlling for day-type composition in subsequent store-level modeling, as stores with higher holiday or weekend exposure may appear to outperform on raw sales metrics absent adjustment.

3. Store Performance – Revenue & Ticket Size

This section examines store-level performance using two complementary metrics: total revenue and average ticket size. Total revenue captures overall sales volume and reflects store scale and traffic, while average ticket size (sales per transaction) provides insight into customer purchasing behavior and basket value. Analyzing both measures allows comparison of stores that generate high revenue through high traffic versus those that rely on higher per-transaction spending.

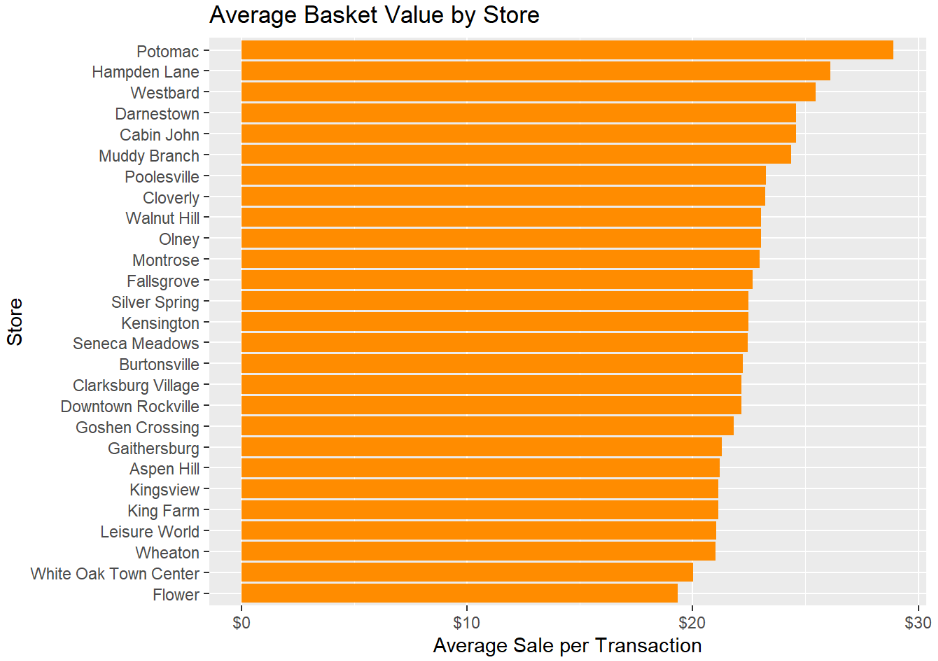
```r
# Total sales per store
store_sales <- transactions %>%
  filter(!IsRefund) %>%
  group_by(STORENAME) %>%
  summarise(TotalSales = sum(NETAMOUNT, na.rm = TRUE)) %>%
  arrange(desc(TotalSales))

# Rank stores by revenue
ggplot(store_sales, aes(x = reorder(STORENAME, TotalSales), y = TotalSales)) +
  geom_col(fill = "purple") +
  coord_flip() +
  labs(
    title = "Stores by Total Revenue",
    x     = "Store",
    y     = "Revenue"
  ) +
  scale_y_continuous(labels = dollar)
```



# Interpretation: Store Revenue

Total revenue is highly concentrated among a subset of stores, with a clear drop-off beyond the top-ranked locations. This dispersion suggests that system-wide performance is disproportionately influenced by a small number of high-volume stores, while the median store operates at a substantially lower revenue level. These differences motivate normalization by square footage and traffic in later sections to distinguish scale effects from true productivity differences.

```
# Average ticket per store (sales per transaction)
store_eff <- transactions %>%
  filter(!IsRefund) %>%
  group_by(STORENAME) %>%
  summarise(
    TotalSales    = sum(NETAMOUNT, na.rm = TRUE),
    Transactions  = n(),
    AvgTicket     = TotalSales / Transactions
  ) %>%
  arrange(desc(AvgTicket))

ggplot(store_eff, aes(x = reorder(STORENAME, AvgTicket), y = AvgTicket)) +
  geom_col(fill = "darkorange") +
  coord_flip() +
  labs(
    title = "Average Basket Value by Store",
    x     = "Store",
    y     = "Average Sale per Transaction"
  ) +
  scale_y_continuous(labels = dollar)
```



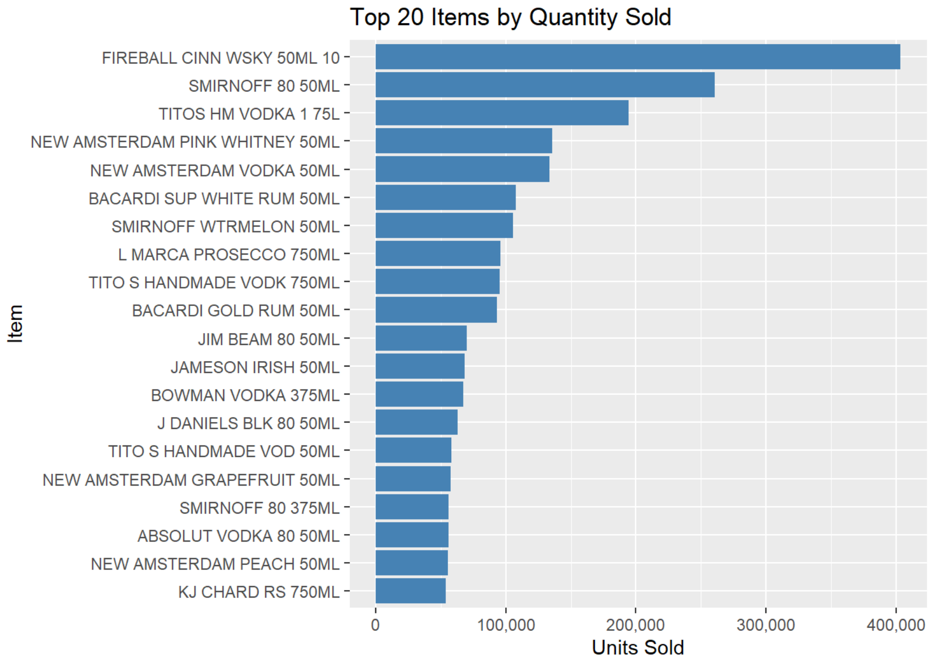
# Interpretation: Average Ticket Size

Average basket value varies meaningfully across stores and does not closely mirror the revenue ranking. Several high-revenue stores do not exhibit the highest average ticket sizes, implying that revenue leadership is often driven by transaction volume rather than per-transaction spend. Conversely, some lower-revenue stores achieve relatively high basket values but lack sufficient traffic to translate this into high total sales.

4. Top Items (Quantity & Revenue)

To identify key products driving sales, item-level performance is evaluated using both quantity sold and total revenue. Ranking items by quantity highlights high-volume products that contribute to traffic and repeat purchases, while ranking by revenue emphasizes higher-priced or premium products that disproportionately affect total sales. Together, these views provide a balanced understanding of product performance across volume and value dimensions.

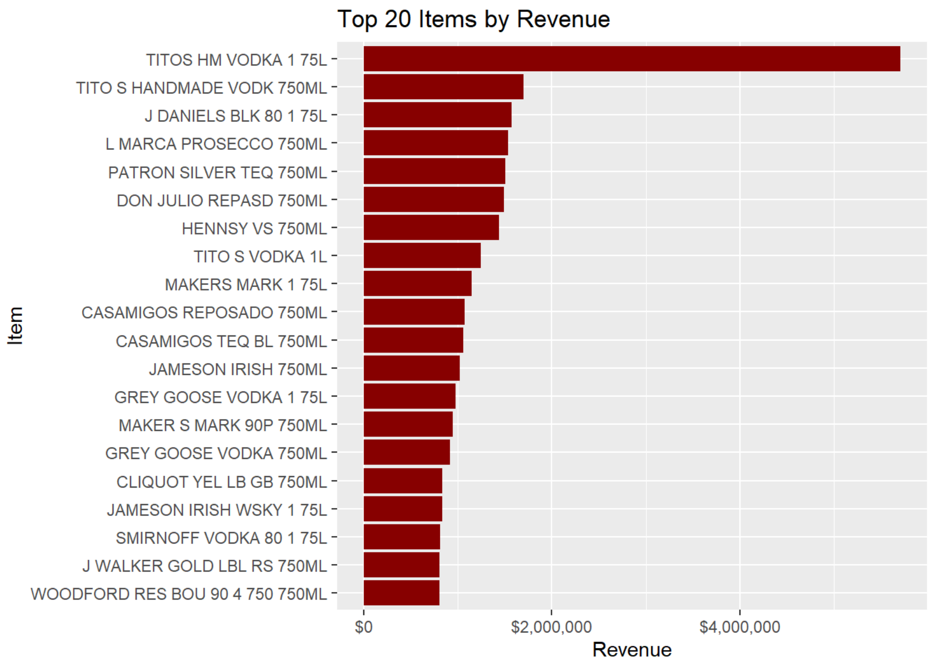
```
# Top items by units sold
top_items_qty <- transactions %>%
  filter(!IsRefund) %>%
  group_by(DESCRIPTION) %>%
  summarise(UnitsSold = sum(LINEQTY, na.rm = TRUE)) %>%
  arrange(desc(UnitsSold)) %>%
  head(20)

ggplot(top_items_qty,
       aes(x = reorder(DESCRIPTION, UnitsSold), y = UnitsSold)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Top 20 Items by Quantity Sold",
    x     = "Item",
    y     = "Units Sold"
  ) +
  scale_y_continuous(labels = comma)
```



# Interpretation: Top Items by Quantity

Unit sales are highly concentrated among a small number of products, consistent with a classic “core SKU” structure. These high-volume items account for a disproportionate share of transaction units despite representing a small fraction of total assortment. Their dominance implies that availability and replenishment reliability for these items are likely to have an outsized effect on customer satisfaction and repeat visits.

```
# Top items by revenue
top_items_rev <- transactions %>%
  filter(!IsRefund) %>%
  group_by(DESCRIPTION) %>%
  summarise(Revenue = sum(NETAMOUNT, na.rm = TRUE)) %>%
  arrange(desc(Revenue)) %>%
  head(20)

ggplot(top_items_rev,
       aes(x = reorder(DESCRIPTION, Revenue), y = Revenue)) +
  geom_col(fill = "darkred") +
  coord_flip() +
  labs(
    title = "Top 20 Items by Revenue",
    x     = "Item",
    y     = "Revenue"
  ) +
  scale_y_continuous(labels = dollar)
```



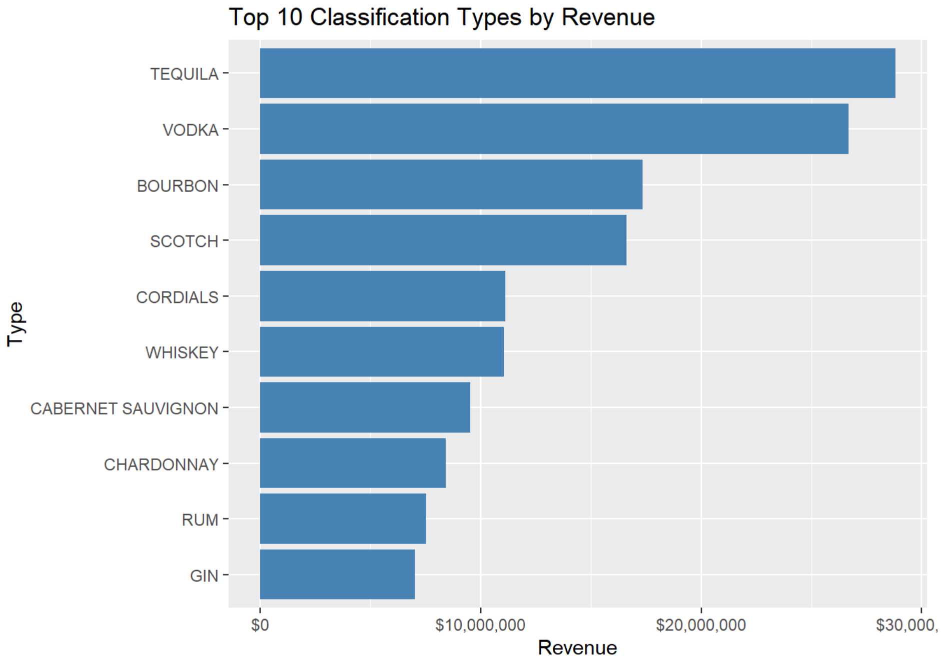
Interpretation: Top Items by Revenue

The set of highest-revenue items differs noticeably from the highest-volume items, reflecting the role of price and product tier in revenue generation. This divergence indicates that revenue performance depends jointly on volume and price, reinforcing the need to analyze both quantity-driven and value-driven products when evaluating assortment effectiveness.

5. Category-Level EDA (Department & Type)

This section aggregates sales at the category level to assess performance across major product groupings. Revenue is summarized by classification type and department to identify which segments contribute most to overall sales. Category-level analysis helps contextualize store and item-level results by revealing structural differences in demand across product classes.

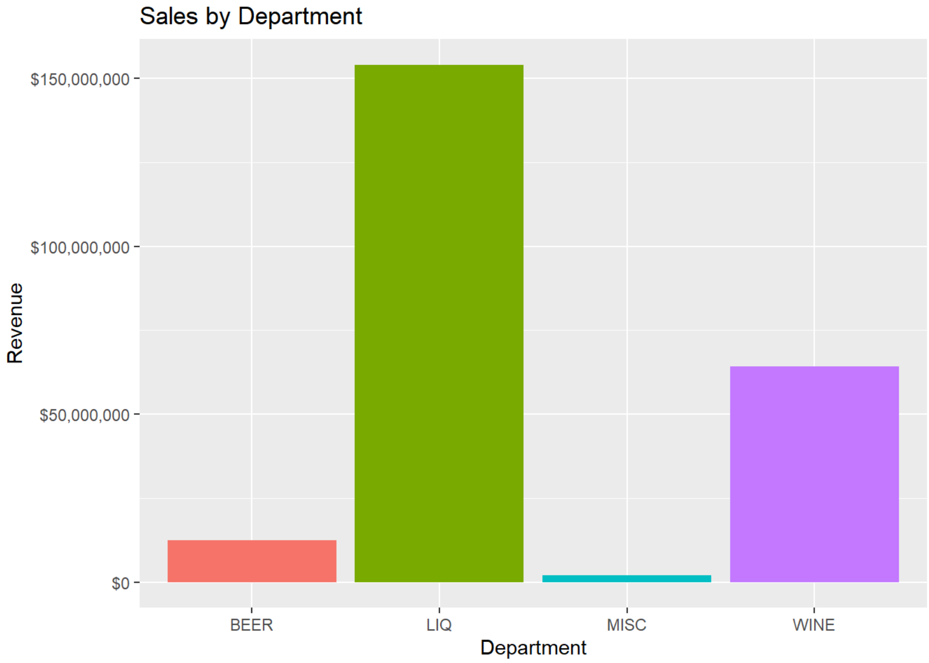
```
# Top classification types by revenue
transactions %>%
  filter(!IsRefund) %>%
  group_by(CLASSIFICATIONTYPE) %>%
  summarise(TotalSales = sum(NETAMOUNT, na.rm = TRUE)) %>%
  arrange(desc(TotalSales)) %>%
  head(10) %>%
  ggplot(aes(x = reorder(CLASSIFICATIONTYPE, TotalSales), y = TotalSales)) +
  geom_col(fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Top 10 Classification Types by Revenue",
    x     = "Type",
    y     = "Revenue"
  ) +
  scale_y_continuous(labels = dollar)
```



# Interpretation: Top Classification Types

Sales concentration in the top classification types suggests that a limited number of product types drive a large portion of revenue. This pattern helps prioritize category focus when analyzing mix differences across stores or designing store-level assortment strategies.

```
# Revenue split by major department
transactions %>%
  filter(!IsRefund) %>%
  group_by(CLASSIFICATIONDEPARTMENT) %>%
  summarise(TotalSales = sum(NETAMOUNT, na.rm = TRUE)) %>%
  ggplot(aes(x = CLASSIFICATIONDEPARTMENT, y = TotalSales, fill = CLASSIFICATIONDEPARTMENT)) +
  geom_col() +
  labs(
    title = "Sales by Department",
    x     = "Department",
    y     = "Revenue"
  ) +
  scale_y_continuous(labels = dollar) +
  theme(legend.position = "none")
```



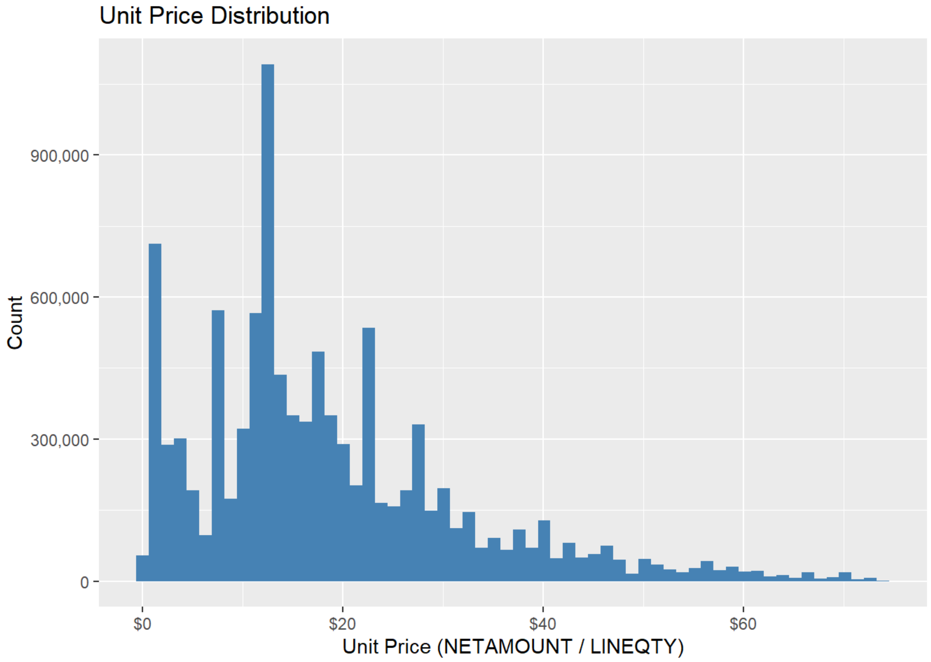
# Interpretation: Department Sales Mix

Revenue is unevenly distributed across major departments, with liquor accounting for the dominant share of total sales, followed by wine and beer, and a minimal contribution from miscellaneous items. Because department shares vary across stores, this structural mix difference has direct implications for sales per square foot and must be controlled for in store-level comparisons. Treating stores as homogeneous without accounting for mix would confound productivity with category composition.

6. Price Distribution (FEE = Price per Line Unit)

Unit-level pricing behavior is examined by analyzing the distribution of transaction prices per unit. Prices are calculated as net sales divided by quantity and are capped at the 99th percentile to reduce the influence of extreme outliers. Both overall price distributions and department-level comparisons are used to evaluate pricing variability and differences in typical unit prices across product categories.

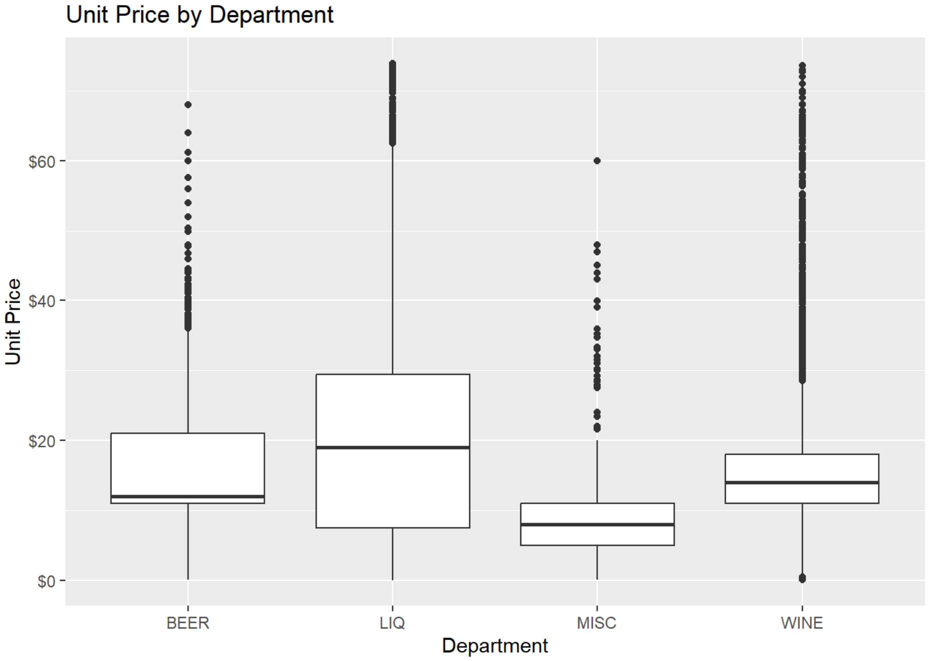
```
# Overall unit price distribution (capped at 99th percentile)
transactions %>%
  filter(!IsRefund, !is.na(FEE), FEE > 0, FEE < quantile(FEE, 0.99, na.rm = TRUE)) %>%
  ggplot(aes(FEE)) +
  geom_histogram(bins = 60, fill = "steelblue") +
  labs(
    title = "Unit Price Distribution",
    x     = "Unit Price (NETAMOUNT / LINEQTY)",
    y     = "Count"
  )+
  scale_y_continuous(labels = comma) +
  scale_x_continuous(labels = scales::dollar)
```



# Interpretation: Unit Price Distribution

Unit prices show a skewed distribution, with most purchases concentrated at lower price points and a smaller number of high-priced transactions in the tail. Capping at the 99th percentile reduces the influence of extreme values and provides a clearer view of typical pricing behavior.

```
# Price distribution by department
transactions %>%
  filter(!IsRefund, !is.na(FEE), FEE > 0,
         FEE < quantile(FEE, 0.99, na.rm = TRUE)) %>%
  ggplot(aes(x = CLASSIFICATIONDEPARTMENT, y = FEE)) +
  geom_boxplot() +
  labs(
    title = "Unit Price by Department",
    x     = "Department",
    y     = "Unit Price"
  ) +
  scale_y_continuous(labels = scales::dollar)
```



# Interpretation: Price by Department

Unit prices vary by department, indicating that departments differ in typical price levels and price dispersion. This matters for interpreting revenue patterns, since higher department revenue may reflect higher prices, higher volume, or both.

7. Year Subsets (FY23–24 and FY24–25)

To enable consistent year-over-year comparison, transaction data are segmented into two fiscal-year subsets: FY23–24 and FY24–25. Fiscal years are defined using July 1 as the start date and June 30 as the end date, aligning the analysis with standard public-sector reporting conventions. In addition, the enriched transaction dataset containing day-type classifications is similarly filtered to the FY24–25 period to support subsequent temporal and store-level analysis within a consistent time frame.

```
# Filter transactions into fiscal years
y24_25 <- transactions %>%
  filter(
    TRANSDATE >= as.Date("2024-07-01"),
    TRANSDATE <= as.Date("2025-06-30")
  )

y23_24 <- transactions %>%
  filter(
    TRANSDATE >= as.Date("2023-07-01"),
    TRANSDATE <= as.Date("2024-06-30")
  )

# Holiday/daytype subset for FY24–25
transactionsFS_fy <- transactionsFS %>%
  filter(
    TRANSDATE >= as.Date("2024-07-01"),
    TRANSDATE <= as.Date("2025-06-30")
  )

transactionsFS %>%
  count(DayType)
```

```
##   DayType       n
## 1 Holiday   59073
## 2 Weekday 7111358
## 3 Weekend 3019959
```

8. Store-Level KPIs (Sales, Traffic, Baskets, per SqFt)

This section constructs a comprehensive set of store-level performance indicators for FY24–25. Core metrics include total sales, transaction counts, and basket counts, which capture overall revenue and customer activity. These measures are further normalized by store square footage to account for differences in physical store size and to enable fair performance comparisons across locations. Basket-level metrics, including average basket value and average items per basket, are also calculated to characterize purchasing behavior and transaction composition at each store.

```
# Core store-level KPIs for Y24–25
store_summary <- transactions %>%
  filter(!IsRefund) %>%
  group_by(STORENAME) %>%
  summarise(
    TotalSales        = sum(NETAMOUNT, na.rm = TRUE),
    TotalTransactions = n(),
    TotalBaskets      = n_distinct(TRANSACTIONID),
    .groups           = "drop"
  )

# Add store square footage
store_sqft <- designation %>%
  mutate(STORENAME = trimws(STORENAME)) %>%
  select(STORENAME, SquareFootage)

store_summary <- store_summary %>%
  left_join(store_sqft, by = "STORENAME") %>%
  mutate(
    TotalSalesPerSqFt        = TotalSales / SquareFootage,
    TotalTransactionsPerSqFt = TotalTransactions / SquareFootage,
    TotalBasketsPerSqFt      = TotalBaskets / SquareFootage,
    Avg_BasketValue          = TotalSales / TotalBaskets
  )

# Basket-level metrics at store level
basket_metrics <- transactions %>%
  filter(!IsRefund) %>%
  group_by(STORENAME, TRANSACTIONID) %>%
  summarise(
    BasketTotal = sum(NETAMOUNT, na.rm = TRUE),
    Items       = sum(LINEQTY,   na.rm = TRUE),
    .groups     = "drop"
  )

basket_summary <- basket_metrics %>%
  group_by(STORENAME) %>%
  summarise(
    AvgBasketValue    = mean(BasketTotal, na.rm = TRUE),
    AvgItemsPerBasket = mean(Items,       na.rm = TRUE),
    .groups           = "drop"
  )

# Attach basket metrics back to store KPIs
store_summary <- store_summary %>%
  left_join(basket_summary, by = "STORENAME")
```

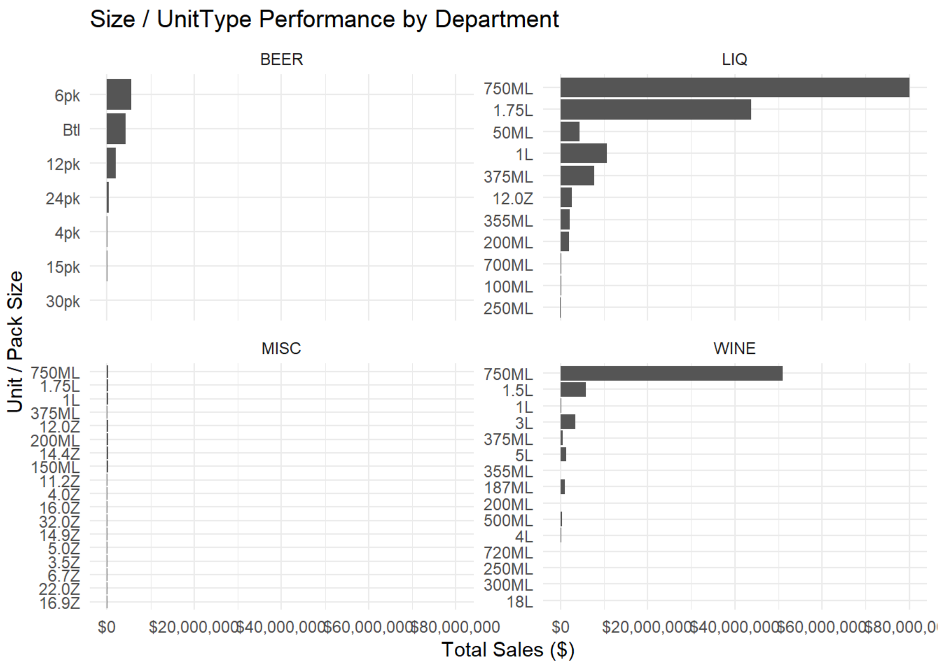
9. Department & Size Mix (Y24–25)

To assess differences in product mix across departments, sales performance is aggregated by department and unit type (bottle size or pack size). Both total sales and quantity sold are calculated, and department-level shares are derived to quantify the relative importance of each unit type within a department. This analysis highlights how product size and packaging contribute to revenue and volume differently across categories, providing insight into assortment structure and customer preferences.

```
# Size/pack performance within each department
size_perf <- transactions %>%
  filter(!IsRefund) %>%
  group_by(CLASSIFICATIONDEPARTMENT, UnitType) %>%
  summarise(
    TotalSales = sum(NETAMOUNT, na.rm = TRUE),
    TotalQty   = sum(LINEQTY,   na.rm = TRUE),
    Baskets    = n_distinct(TRANSACTIONID),
    .groups    = "drop"
  ) %>%
  group_by(CLASSIFICATIONDEPARTMENT) %>%
  mutate(
    SalesShareInDept = TotalSales / sum(TotalSales, na.rm = TRUE),
    QtyShareInDept   = TotalQty   / sum(TotalQty,   na.rm = TRUE)
  ) %>%
  ungroup() %>%
  arrange(CLASSIFICATIONDEPARTMENT, desc(TotalSales))

size_perf
```

```
## # A tibble: 51 × 7
##    CLASSIFICATIONDEPARTM…¹ UnitType TotalSales TotalQty Baskets SalesShareInDept
##    <chr>                   <chr>         <dbl>    <dbl>   <int>            <dbl>
##  1 BEER                    6pk       5548109.    477796  356458       0.446
##  2 BEER                    Btl       4333680.    133443   99249       0.348
##  3 BEER                    12pk      1999619.     94486   83243       0.161
##  4 BEER                    24pk       367660.      8457    5545       0.0295
##  5 BEER                    4pk        124601.     12131    9852       0.0100
##  6 BEER                    15pk        76256.      6947    6146       0.00613
##  7 BEER                    30pk           22.0        1       1       0.00000177
##  8 LIQ                     750ML    80073982.   2715388 1955814       0.520
##  9 LIQ                     1.75L    43804699.   1425996 1096192       0.284
## 10 LIQ                     1L       10625322.    431716  272643       0.0689
## # ℹ 41 more rows
## # ℹ abbreviated name: ¹CLASSIFICATIONDEPARTMENT
## # ℹ 1 more variable: QtyShareInDept <dbl>
```

```
# Faceted view of UnitType sales by department
ggplot(size_perf,
       aes(x = reorder(UnitType, TotalSales), y = TotalSales)) +
  geom_col() +
  coord_flip() +
  facet_wrap(~ CLASSIFICATIONDEPARTMENT, scales = "free_y") +
  labs(
    title = "Size / UnitType Performance by Department",
    x     = "Unit / Pack Size",
    y     = "Total Sales ($)"
  ) +
  scale_y_continuous(labels = dollar_format()) +
  theme_minimal()
```



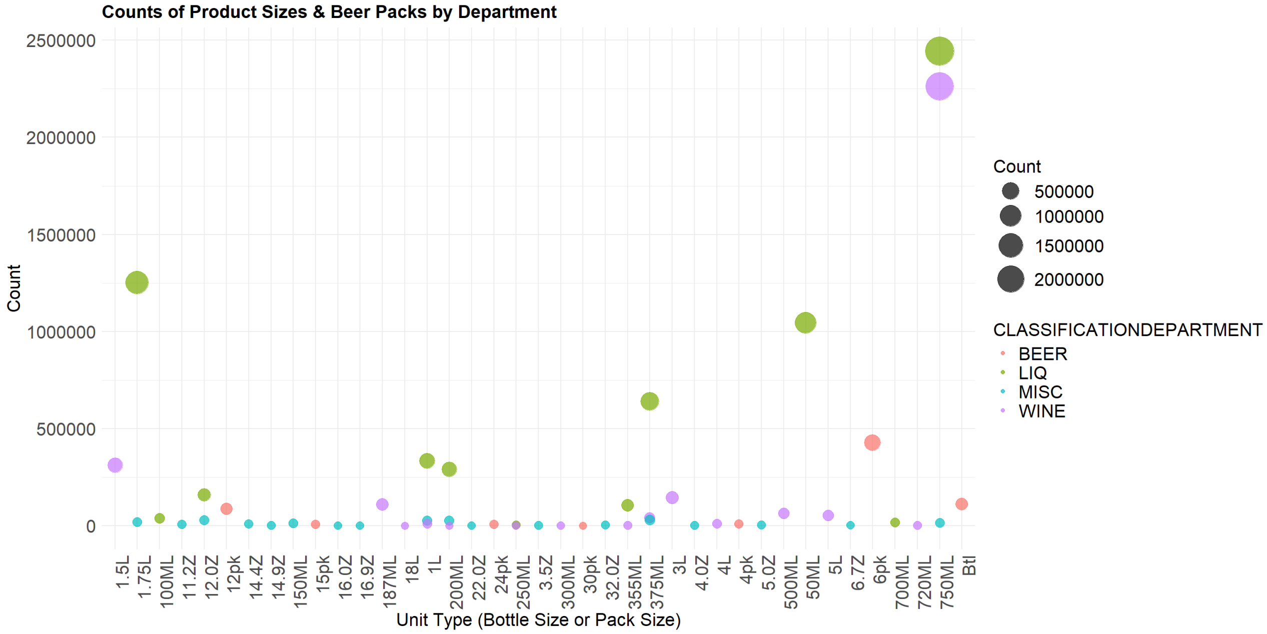
# Interpretation: Size and Unit-Type Mix by Department

Within each department, sales are concentrated in a small number of dominant unit sizes or pack formats. For example, beer sales are heavily weighted toward multi-pack formats, while liquor sales are dominated by standard 750ml and 1.75L bottles. The presence of a long tail of low-share unit types suggests potential assortment complexity that may contribute limited incremental revenue relative to operational cost.

10. Counts of Sizes / Packs by Department

This section examines the frequency distribution of product sizes and pack configurations across departments. Counts of each unit type are computed to assess assortment breadth and operational complexity within categories. A bubble plot visualization is used to simultaneously display frequency, relative magnitude, and departmental grouping, enabling comparison of how product variety differs across major departments.

```
# Frequency of size/pack combinations within departments
size_counts <- transactions %>%
  group_by(CLASSIFICATIONDEPARTMENT, UnitType) %>%
  summarise(Count = n(), .groups = "drop") %>%
  arrange(desc(Count))

# Bubble plot of size/pack frequencies
ggplot(size_counts,
       aes(x = UnitType, y = Count, size = Count, color = CLASSIFICATIONDEPARTMENT)) +
  geom_point(alpha = 0.7) +
  scale_size(range = c(3, 12)) +
  labs(
    title = "Counts of Product Sizes & Beer Packs by Department",
    x     = "Unit Type (Bottle Size or Pack Size)",
    y     = "Count"
  ) +
  theme_minimal() +
  theme(
    axis.text.x  = element_text(angle = 90, hjust = 1, size = 16),   # x-axis labels
    axis.text.y  = element_text(size = 16),                          # y-axis labels
    axis.title.x = element_text(size = 16),                          # x-axis title
    axis.title.y = element_text(size = 16),                          # y-axis title
    plot.title   = element_text(size = 16, face = "bold"),           # plot title
    legend.text  = element_text(size = 16),                          # legend labels
    legend.title = element_text(size = 16)                           # legend title
  )
```



# Interpretation: Product Size and Pack Frequency

The frequency distribution of unit types is highly skewed, with a few size and pack configurations appearing in a large proportion of transaction records and many others occurring rarely. This imbalance indicates that assortment breadth is not evenly utilized and that a substantial portion of SKU variety contributes marginally to observed transaction activity. Such patterns are relevant for evaluating replenishment efficiency and shelf-space allocation.

11. Category Mix by Store (Wide Pivot)

Store-level category mix is analyzed by calculating department-specific sales shares for each store. These shares are reshaped into a wide, store-by-department matrix to facilitate comparison across locations and to support downstream modeling. In parallel, sales shares by day type (holiday, weekend, weekday) are computed for each store to capture temporal demand patterns. Together, these features provide a structured representation of each store's sales composition across both product categories and time dimensions.

```
# Department-level sales and shares by store
store_dept_sales <- transactions %>%
  filter(!IsRefund) %>%
  group_by(STORENAME, CLASSIFICATIONDEPARTMENT) %>%
  summarise(
    TotalSales = sum(NETAMOUNT, na.rm = TRUE),
    TotalQty   = sum(LINEQTY,   na.rm = TRUE),
    .groups    = "drop"
  )

store_dept_mix <- store_dept_sales %>%
  group_by(STORENAME) %>%
  mutate(
    StoreTotalSales = sum(TotalSales, na.rm = TRUE),
    DeptShare       = TotalSales / StoreTotalSales
  ) %>%
  ungroup()

# Store-by-department share matrix
store_category_mix <- store_dept_mix %>%
  select(STORENAME, CLASSIFICATIONDEPARTMENT, DeptShare) %>%
  pivot_wider(
    names_from  = CLASSIFICATIONDEPARTMENT,
    values_from = DeptShare,
    values_fill = 0,
    names_prefix = "Share_"
  )

store_category_mix
```

```
## # A tibble: 27 × 5
##    STORENAME          Share_BEER Share_LIQ Share_MISC Share_WINE
##    <chr>                   <dbl>     <dbl>      <dbl>      <dbl>
##  1 Aspen Hill             0.0919     0.698    0.00700      0.203
##  2 Burtonsville           0.0725     0.632    0.00763      0.288
##  3 Cabin John             0.0651     0.554    0.0109       0.370
##  4 Clarksburg Village     0.0773     0.618    0.00753      0.297
##  5 Cloverly               0.0570     0.670    0.00749      0.266
##  6 Darnestown             0.0548     0.558    0.00918      0.378
##  7 Downtown Rockville     0.0490     0.663    0.00872      0.279
##  8 Fallsgrove             0.0546     0.637    0.00812      0.300
##  9 Flower                 0.0712     0.670    0.00760      0.251
## 10 Gaithersburg           0.0437     0.765    0.0105       0.181
## # ℹ 17 more rows
```

```
# Day-type (Holiday / Weekend / Weekday) mix by store
store_daytype_mix <- transactionsFS %>%
  group_by(STORENAME, DayType) %>%
  summarise(DaySales = sum(NETAMOUNT, na.rm = TRUE), .groups = "drop") %>%
  group_by(STORENAME) %>%
  mutate(DayShare = DaySales / sum(DaySales, na.rm = TRUE)) %>%
  ungroup() %>%
  select(STORENAME, DayType, DayShare) %>%
  pivot_wider(
    names_from  = DayType,
    values_from = DayShare,
    values_fill = 0,
    names_prefix = "Share_"
  )

store_daytype_mix %>% count(STORENAME)
```

```
## # A tibble: 27 × 2
##    STORENAME              n
##    <chr>              <int>
##  1 Aspen Hill             1
##  2 Burtonsville           1
##  3 Cabin John             1
##  4 Clarksburg Village     1
##  5 Cloverly               1
##  6 Darnestown             1
##  7 Downtown Rockville     1
##  8 Fallsgrove             1
##  9 Flower                 1
## 10 Gaithersburg           1
## # ℹ 17 more rows
```

```
store_daytype_mix %>%
  mutate(sum_shares = Share_Holiday + Share_Weekday + Share_Weekend) %>%
  select(STORENAME, sum_shares)
```

```
## # A tibble: 27 × 2
##    STORENAME          sum_shares
##    <chr>                   <dbl>
##  1 Aspen Hill                  1
##  2 Burtonsville                1
##  3 Cabin John                  1
##  4 Clarksburg Village          1
##  5 Cloverly                    1
##  6 Darnestown                  1
##  7 Downtown Rockville          1
##  8 Fallsgrove                  1
##  9 Flower                      1
## 10 Gaithersburg                1
## # ℹ 17 more rows
```

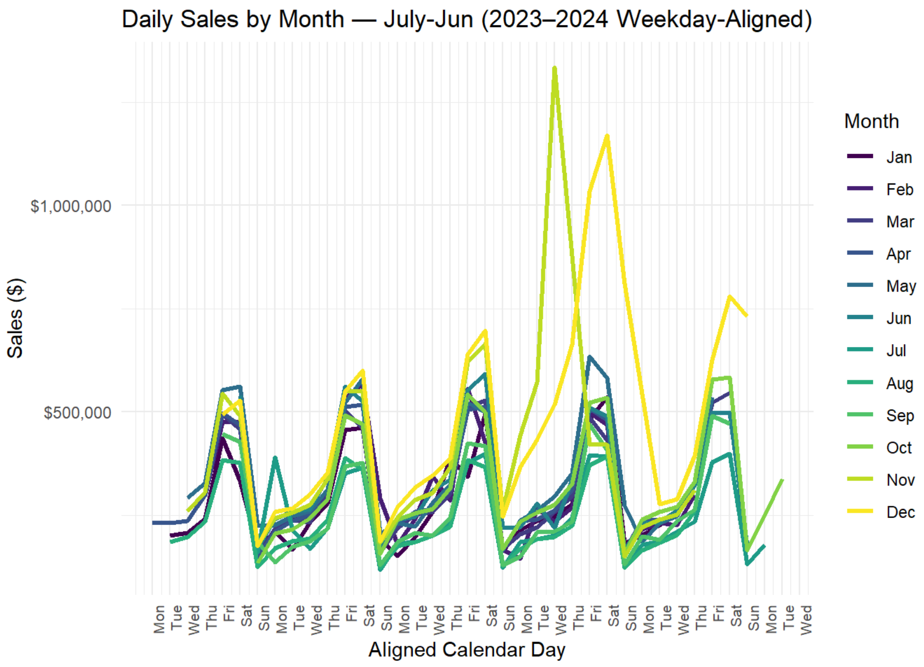
Interpretation: Store-Level Category and Day-Type Mix

Stores exhibit substantial heterogeneity in both category composition and temporal demand structure. Because department and day-type shares sum to one within each store, differences reflect true trade-offs rather than independent effects. These compositional differences imply that stores operate under distinct demand profiles, justifying their explicit inclusion as predictors in the regression models rather than treating them as background context.

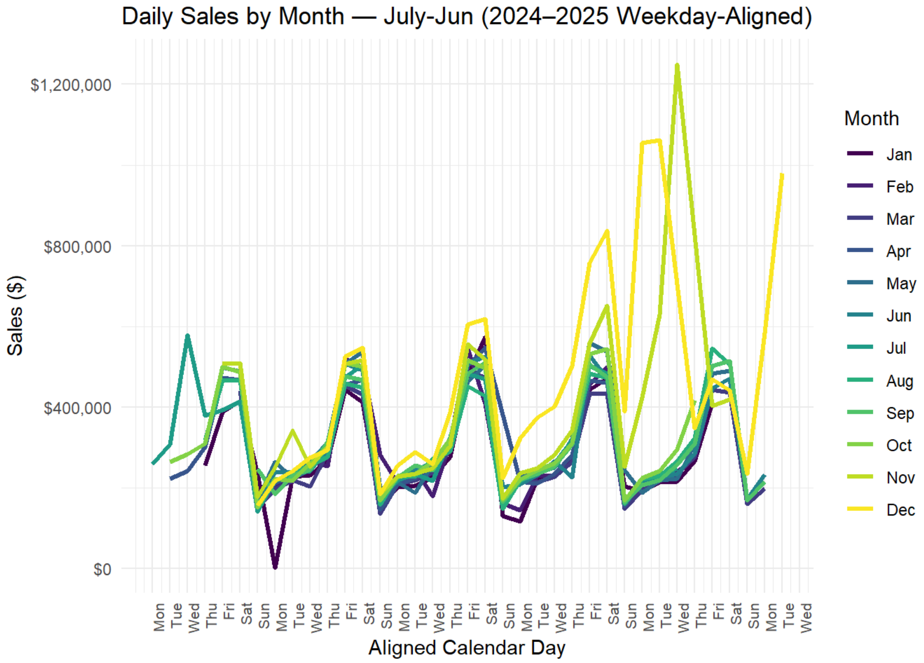
12. Daily Sales & FY-Comparison – Calendar Alignment

This section examines daily sales patterns across fiscal years while accounting for calendar effects. Daily total sales are aggregated across all stores and then aligned by weekday within each month to control for differences in calendar structure between years. This weekday-aligned approach allows for more meaningful comparison of intra-month sales patterns by reducing distortions caused by shifting weekends and holidays. Separate visualizations are produced for FY23–24 and FY24–25 to facilitate direct comparison of seasonal and weekday-driven demand patterns.

```
# Aggregate daily sales across all stores
daily_sales_ts <- transactionsFS %>%
  group_by(TRANSDATE) %>%
  summarise(TotalSales = sum(NETAMOUNT, na.rm = TRUE), .groups = "drop") %>%
  mutate(
    Day   = day(TRANSDATE),
    Month = month(TRANSDATE, label = TRUE, abbr = TRUE),
    Year  = year(TRANSDATE)
  )

# Align Y23–24 by weekday within month
y23_24_aligned <- daily_sales_ts %>%
  filter(
    TRANSDATE >= as.Date("2023-07-01"),
    TRANSDATE <= as.Date("2024-06-30")
  ) %>%
  mutate(
    FirstOfMonth  = floor_date(TRANSDATE, "month"),
    WeekdayOffset = wday(FirstOfMonth, week_start = 1) - 1,
    DayAligned    = day(TRANSDATE) + WeekdayOffset
  )

# Align Y24–25 by weekday within month
y24_25_aligned <- daily_sales_ts %>%
  filter(
    TRANSDATE >= as.Date("2024-07-01"),
    TRANSDATE <= as.Date("2025-06-30")
  ) %>%
  mutate(
    FirstOfMonth  = floor_date(TRANSDATE, "month"),
    WeekdayOffset = wday(FirstOfMonth, week_start = 1) - 1,
    DayAligned    = day(TRANSDATE) + WeekdayOffset
  )

# Label for aligned weekdays
weekday_labels <- rep(c("Mon","Tue","Wed","Thu","Fri","Sat","Sun"), length.out = 40)

# Weekday-aligned daily sales – Y23–24
ggplot(y23_24_aligned, aes(x = DayAligned, y = TotalSales, color = Month, group = Month)) +
  geom_line(linewidth = 1.1) +
  labs(
    title = "Daily Sales by Month — July-Jun (2023–2024 Weekday-Aligned)",
    x     = "Aligned Calendar Day",
    y     = "Sales ($)"
  ) +
  scale_x_continuous(
    breaks = seq(1, 40, by = 1),
    labels = weekday_labels
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 8))
```



```
# Weekday-aligned daily sales – Y24-25
ggplot(y24_25_aligned, aes(x = DayAligned, y = TotalSales, color = Month, group = Month)) +
  geom_line(linewidth = 1.1) +
  labs(
    title = "Daily Sales by Month – July-Jun (2024-2025 Weekday-Aligned)",
    x     = "Aligned Calendar Day",
    y     = "Sales ($)"
  ) +
  scale_x_continuous(
    breaks = seq(1, 40, by = 1),
    labels = weekday_labels
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 8))
```



# Interpretation: Weekday-Aligned

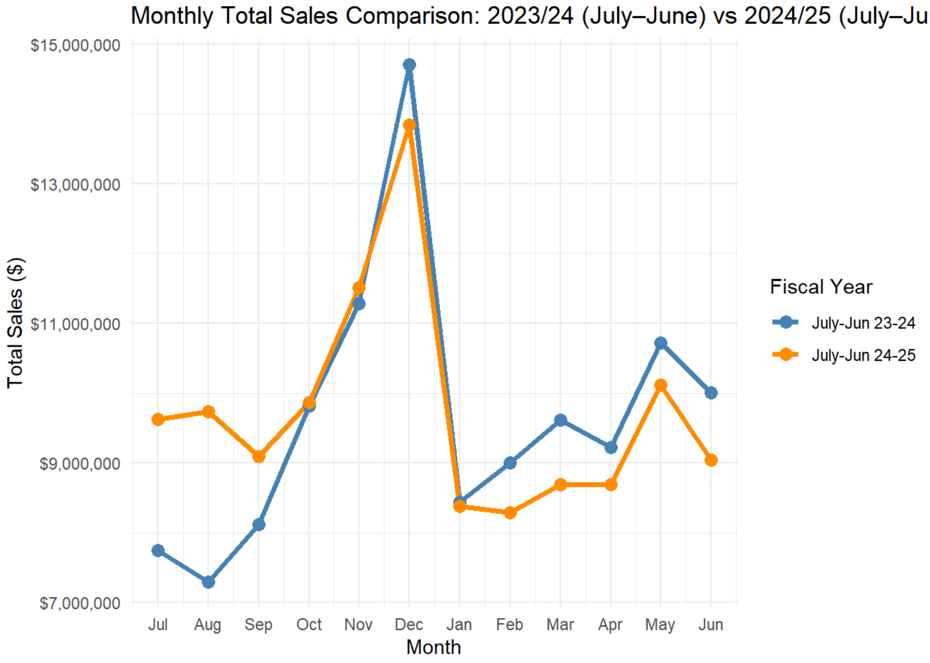
Daily Sales Patterns

Weekday alignment reveals recurring intra-month sales patterns that are consistent across years, with visible peaks and troughs tied to the weekly sales cycle. While overall sales levels differ between FY23–24 and FY24–25, the similarity in shape across months suggests that short-term demand is strongly influenced by calendar structure rather than random variation. This supports the use of calendar-adjusted comparisons in subsequent performance analysis.

13. Monthly Sales Comparison: July-Jun (23–24 vs 24–25)

To evaluate broader year-over-year trends, monthly total sales are aggregated for each year using a consistent July–June reporting window. Monthly sales trajectories for FY23–24 and FY24–25 are plotted together to assess differences in overall growth, seasonality, and relative performance across corresponding months. This comparison provides a high-level view of year performance that complements the more granular daily analysis.

```
# Year-month string for monthly aggregation
transactionsFS <- transactionsFS %>%
  mutate(YearMonth = format(TRANSDATE, "%Y-%m"))

# Monthly total sales - July-Jun 23-24
y23_24_monthlyFS <- transactionsFS %>%
  filter(TRANSDATE >= "2023-07-01" & TRANSDATE <= "2024-06-30") %>%
  group_by(YearMonth) %>%
  summarise(TotalSales = sum(NETAMOUNT), .groups = "drop") %>%
  arrange(YearMonth) %>%
  mutate(FYear = "July-Jun 23-24", MonthIndex = row_number())

# Monthly total sales - July-Jun 24-25
y24_25_monthlyFS <- transactionsFS %>%
  filter(TRANSDATE >= "2024-07-01" & TRANSDATE <= "2025-06-30") %>%
  group_by(YearMonth) %>%
  summarise(TotalSales = sum(NETAMOUNT), .groups = "drop") %>%
  arrange(YearMonth) %>%
  mutate(FYear = "July-Jun 24-25", MonthIndex = row_number())

# Combine fiscal years for comparison
combined_monthlyFS <- rbind(y23_24_monthlyFS, y24_25_monthlyFS)

ggplot(combined_monthlyFS, aes(x = MonthIndex, y = TotalSales, color = FYear, group = FYear)) +
  geom_line(linewidth = 1.3) +
  geom_point(size = 3) +
  scale_x_continuous(
    breaks = 1:12,
    labels = c("Jul","Aug","Sep","Oct","Nov","Dec","Jan","Feb","Mar","Apr","May","Jun")
  ) +
  scale_color_manual(values = c("July-Jun 23-24"="steelblue", "July-Jun 24-25"="darkorange")) +
  labs(
    title = "Monthly Total Sales Comparison: 2023/24 (July–June) vs 2024/25 (July–June)",
    x     = "Month",
    y     = "Total Sales ($)",
    color = "Fiscal Year"
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```



#Interpretation: Fiscal-Year Monthly Sales Comparison

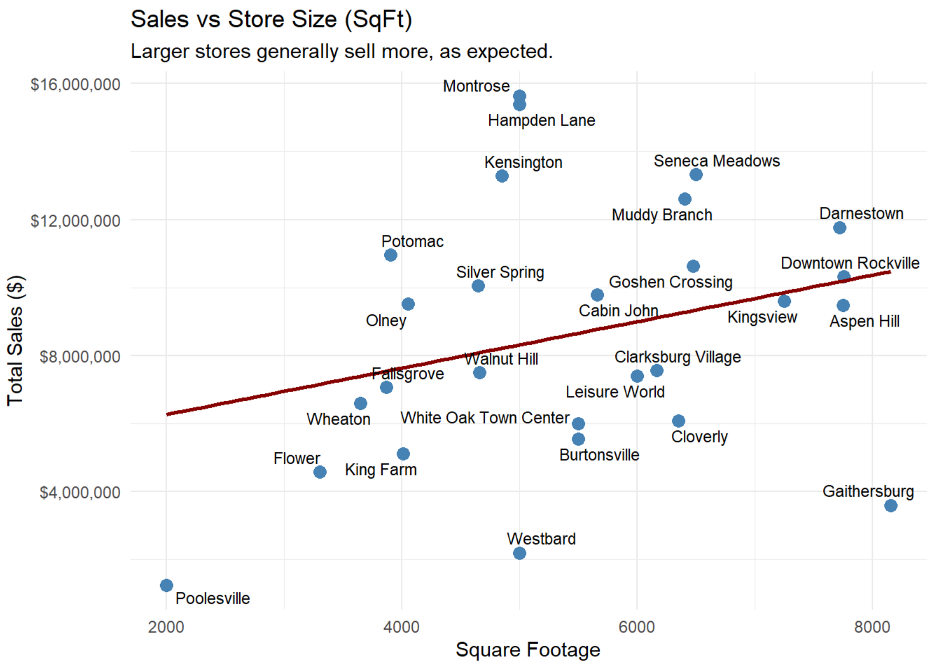
Monthly sales exhibit clear seasonality across both years, with comparable peaks and declines occurring in similar months. Differences in the relative height of the lines indicate changes in overall sales levels between years, while the shared seasonal pattern suggests stable demand timing. This comparison provides context for evaluating whether store-level performance changes reflect broader system-wide trends or localized effects.

14. Demographics + Sales Relationships

This section explores the relationship between store-level sales performance and selected demographic and structural characteristics of each store’s trade area. Store sales are examined in relation to physical store size, total trade-area population, population aged 25 and older, and households below the poverty level. Scatter plots with fitted linear trends and correlation estimates are used to assess the strength and direction of these relationships. The analysis is exploratory in nature and is intended to identify whether demographic context meaningfully explains variation in store-level sales.

```
# Select and standardize the demographic fields
demographics_selected <- demographics %>%
  mutate(STORENAME = trimws(STORENAME)) %>%
  select(
    STORENAME,
    TotalPopulation,
    Older25Years,
    PovertyLevel
    )

# Merge store KPIs with demographics
store_demo <- store_summary %>%
  left_join(demographics_selected, by = "STORENAME") %>%
  filter(!is.na(TotalPopulation))

# Sales vs store size

ggplot(store_demo,
       aes(x = SquareFootage, y = TotalSales)) +
  geom_point(size = 3, color = "steelblue") +
  geom_smooth(method = "lm", se = FALSE, color = "darkred") +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
  labs(
    title = "Sales vs Store Size (SqFt)",
    x     = "Square Footage",
    y     = "Total Sales ($)",
    subtitle = "Larger stores generally sell more, as expected."
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

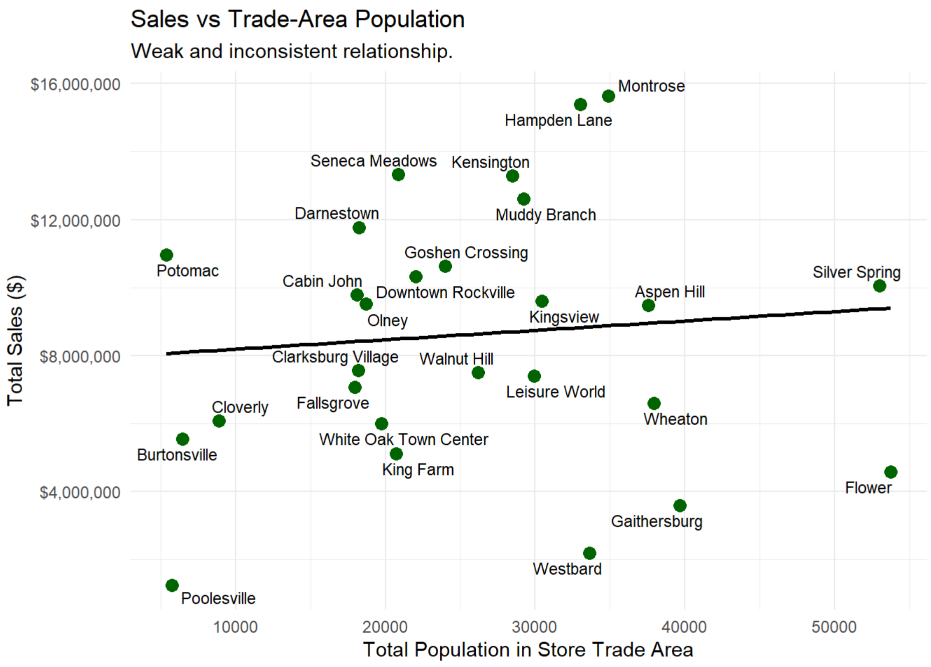


```
cor(store_demo$TotalSales, store_demo$SquareFootage, use = "complete.obs")
```

```
## [1] 0.2813958
```

```
# Sales vs total population in trade area

ggplot(store_demo,
       aes(x = TotalPopulation, y = TotalSales)) +
  geom_point(size = 3, color = "darkgreen") +
  geom_smooth(method = "lm", se = FALSE, color = "black") +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
  labs(
    title = "Sales vs Trade-Area Population",
    x     = "Total Population in Store Trade Area",
    y     = "Total Sales ($)",
    subtitle = "Weak and inconsistent relationship."
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

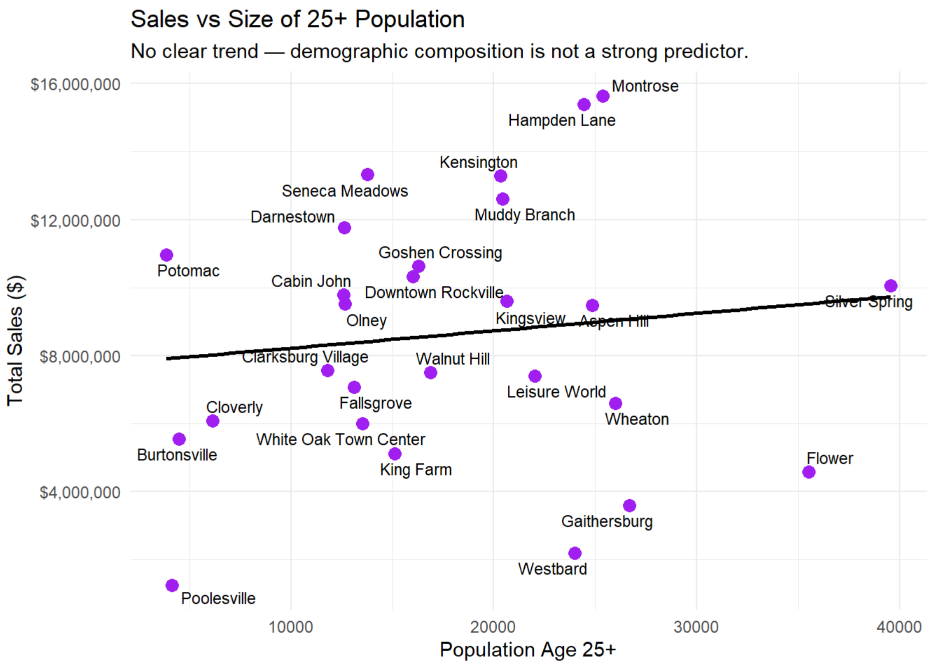


```
cor(store_demo$TotalSales, store_demo$TotalPopulation, use = "complete.obs")
```

```
## [1] 0.0928748
```

```
# Sales vs 25+ population

ggplot(store_demo,
       aes(x = Older25Years, y = TotalSales)) +
  geom_point(size = 3, color = "purple") +
  geom_smooth(method = "lm", se = FALSE, color = "black") +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
labs(
    title = "Sales vs Size of 25+ Population",
    x     = "Population Age 25+",
    y     = "Total Sales ($)",
    subtitle = "No clear trend — demographic composition is not a strong predictor."
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

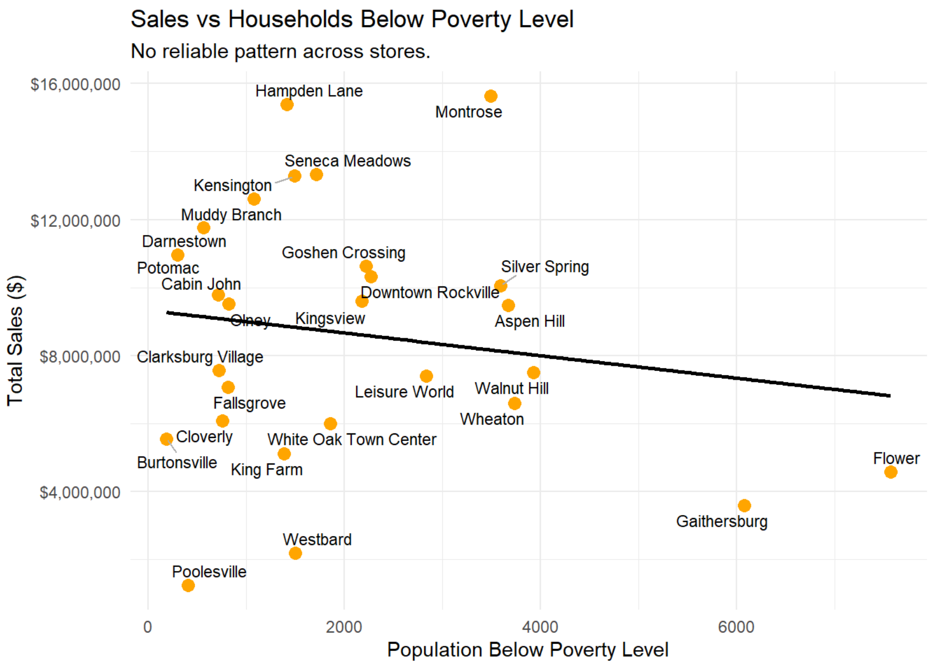


```
cor(store_demo$TotalSales, store_demo$Older25Years, use = "complete.obs")
```

```
## [1] 0.1209889
```

```
ggplot(store_demo,
       aes(x = PovertyLevel,
           y = TotalSales)) +
  geom_point(size = 3, color = "orange") +
  geom_smooth(method = "lm", se = FALSE, color = "black") +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
  labs(
    title = "Sales vs Households Below Poverty Level",
    x     = "Population Below Poverty Level",
    y     = "Total Sales ($)",
    subtitle = "No reliable pattern across stores."
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```


Sales vs Households Below Poverty Level

No reliable pattern across stores.

```
cor(store_demo$TotalSales, store_demo$PovertyLevel, use = "complete.obs")
```

```
## [1] -0.1579369
```

15. Quadrant Analyses (Sales vs Traffic vs Baskets per SqFt)

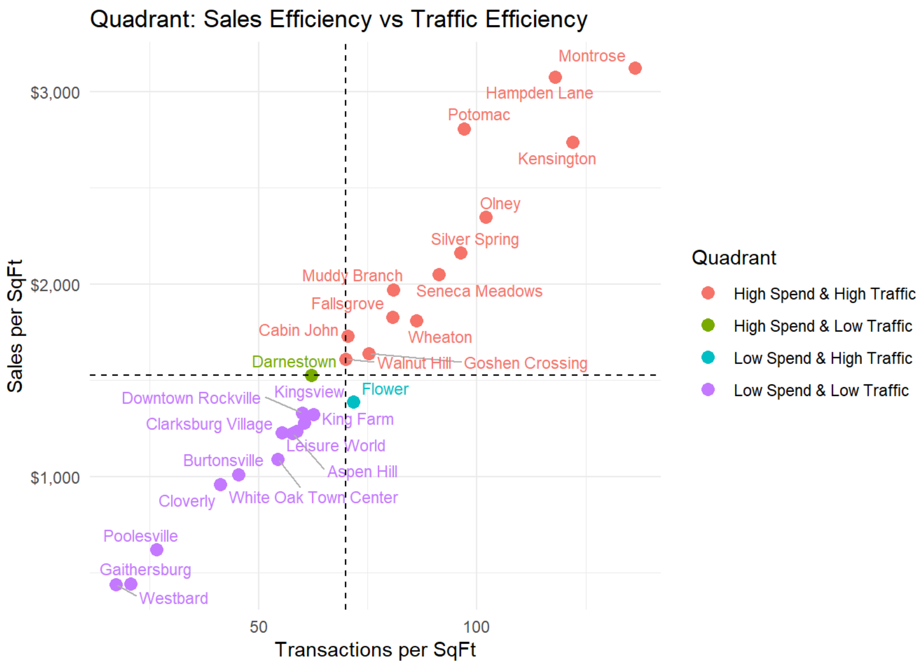
To diagnose relative store performance across multiple dimensions simultaneously, quadrant analyses are conducted using normalized store-level key performance indicators. Sales efficiency, traffic efficiency, and basket density are measured on a per–square-foot basis to ensure comparability across stores of different sizes. Median values are used as cutoffs to classify stores into high and low performance groups across paired dimensions. These quadrant visualizations provide an intuitive framework for identifying over- and under-performing stores and for distinguishing whether performance differences are driven primarily by traffic volume, basket behavior, or sales efficiency.

```
# Build final store-level dataset with KPIs, demographics, and mix
store_final <- store_summary %>%
  left_join(demographics_selected, by = "STORENAME") %>%
  left_join(store_category_mix, by = "STORENAME") %>%
  left_join(store_daytype_mix,   by = "STORENAME") %>%
  mutate(
    TotalSalesPerSqFt        = TotalSales / SquareFootage,
    TotalTransactionsPerSqFt = TotalTransactions / SquareFootage,
    TotalBasketsPerSqFt      = TotalBaskets / SquareFootage,
    Avg_BasketValue          = TotalSales / TotalBaskets
  ) %>%
  distinct(STORENAME, .keep_all = TRUE) %>%
  relocate(SquareFootage, TotalPopulation, .after = STORENAME)

# Check for duplicate store rows (should be none)
store_final %>%
  count(STORENAME) %>%
  filter(n > 1)
```

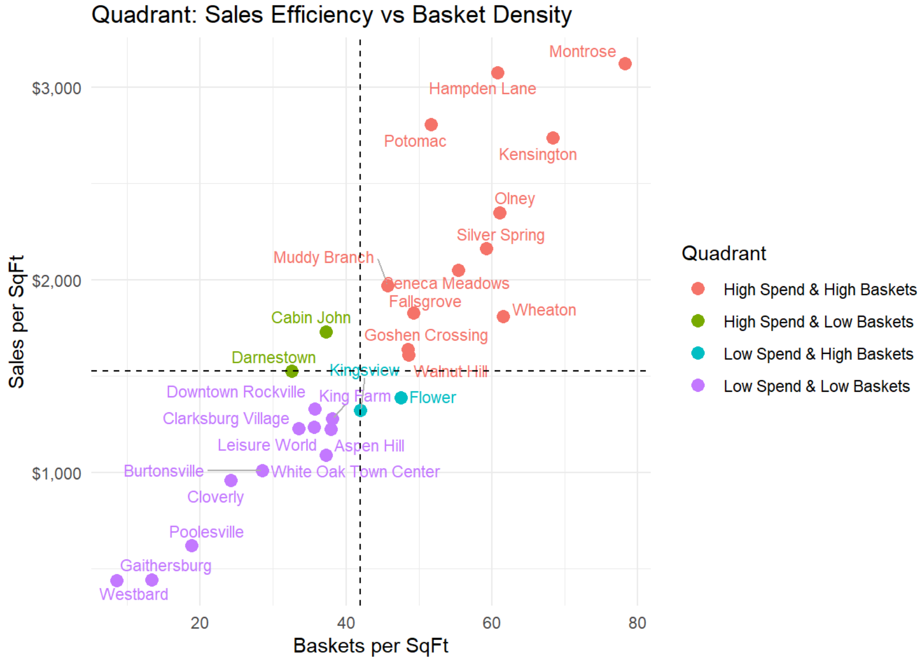
```
## # A tibble: 0 × 2
## # ℹ 2 variables: STORENAME <chr>, n <int>
```

```
# Median cutoffs for quadrants
med_sales_per_sqft <- median(store_final$TotalSalesPerSqFt,        na.rm = TRUE)
med_trans_per_sqft <- median(store_final$TotalTransactionsPerSqFt, na.rm = TRUE)
med_bask_per_sqft  <- median(store_final$TotalBasketsPerSqFt,      na.rm = TRUE)

# Assign quadrant labels for different KPI pairings
store_quadrant <- store_final %>%
  mutate(
    Quad_SalesTraffic = case_when(
      TotalSalesPerSqFt        >= med_sales_per_sqft &
        TotalTransactionsPerSqFt >= med_trans_per_sqft ~ "High Spend & High Traffic",
      TotalSalesPerSqFt        >= med_sales_per_sqft &
        TotalTransactionsPerSqFt <  med_trans_per_sqft ~ "High Spend & Low Traffic",
      TotalSalesPerSqFt        <  med_sales_per_sqft &
        TotalTransactionsPerSqFt >= med_trans_per_sqft ~ "Low Spend & High Traffic",
      TRUE ~ "Low Spend & Low Traffic"
    ),
    Quad_SalesBaskets = case_when(
      TotalSalesPerSqFt     >= med_sales_per_sqft &
        TotalBasketsPerSqFt >= med_bask_per_sqft ~ "High Spend & High Baskets",
      TotalSalesPerSqFt     >= med_sales_per_sqft &
        TotalBasketsPerSqFt <  med_bask_per_sqft ~ "High Spend & Low Baskets",
      TotalSalesPerSqFt     <  med_sales_per_sqft &
        TotalBasketsPerSqFt >= med_bask_per_sqft ~ "Low Spend & High Baskets",
      TRUE ~ "Low Spend & Low Baskets"
    ),
    Quad_TrafficBaskets = case_when(
      TotalTransactionsPerSqFt >= med_trans_per_sqft &
        TotalBasketsPerSqFt    >= med_bask_per_sqft ~ "High Traffic & High Baskets",
      TotalTransactionsPerSqFt >= med_trans_per_sqft &
        TotalBasketsPerSqFt    <  med_bask_per_sqft ~ "High Traffic & Low Baskets",
      TotalTransactionsPerSqFt <  med_trans_per_sqft &
        TotalBasketsPerSqFt    >= med_bask_per_sqft ~ "Low Traffic & High Baskets",
      TRUE ~ "Low Traffic & Low Baskets"
    )
  )
```

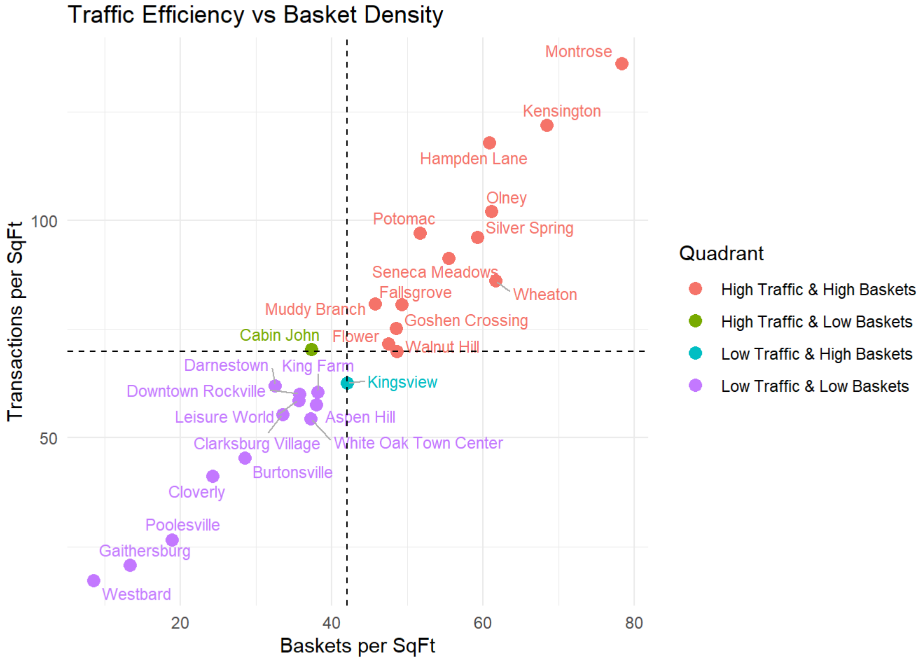
```
# Quadrant plot: Sales per SqFt vs Transactions per SqFt
ggplot(store_quadrant,
       aes(x = TotalTransactionsPerSqFt,
           y = TotalSalesPerSqFt,
           color = Quad_SalesTraffic)) +
  geom_point(size = 3) +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
  geom_vline(xintercept = med_trans_per_sqft, linetype = "dashed") +
  geom_hline(yintercept = med_sales_per_sqft,  linetype = "dashed") +
  labs(
    title = "Quadrant: Sales Efficiency vs Traffic Efficiency",
    x     = "Transactions per SqFt",
    y     = "Sales per SqFt",
    color = "Quadrant"
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```



```r
# Quadrant plot: Sales per SqFt vs Baskets per SqFt
ggplot(store_quadrant,
       aes(x = TotalBasketsPerSqFt,
           y = TotalSalesPerSqFt,
           color = Quad_SalesBaskets)) +
  geom_point(size = 3) +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
  geom_vline(xintercept = med_bask_per_sqft, linetype = "dashed") +
  geom_hline(yintercept = med_sales_per_sqft, linetype = "dashed") +
  labs(
    title = "Quadrant: Sales Efficiency vs Basket Density",
    x     = "Baskets per SqFt",
    y     = "Sales per SqFt",
    color = "Quadrant"
  ) +
  scale_y_continuous(labels = dollar) +
  theme_minimal()
```



```r
# Quadrant plot: Transactions per SqFt vs Baskets per SqFt
ggplot(store_quadrant,
       aes(x = TotalBasketsPerSqFt,
           y = TotalTransactionsPerSqFt,
           color = Quad_TrafficBaskets)) +
  geom_point(size = 3) +
  ggrepel::geom_text_repel(
    aes(label = STORENAME),
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70",
    show.legend   = FALSE
  ) +
  geom_vline(xintercept = med_bask_per_sqft,  linetype = "dashed") +
  geom_hline(yintercept = med_trans_per_sqft, linetype = "dashed") +
  labs(
    title = "Traffic Efficiency vs Basket Density",
    x     = "Baskets per SqFt",
    y     = "Transactions per SqFt",
    color = "Quadrant"
  ) +
  theme_minimal()
```



# Interpretation: Store Performance

Quadrants

The quadrant analyses reveal meaningful heterogeneity in store performance across sales efficiency, traffic intensity, and basket density. Some stores achieve high sales primarily through strong traffic, while others rely on higher basket value or transaction density. The dispersion across quadrants indicates that under- and over-performance can arise from different underlying mechanisms, reinforcing the need to evaluate stores along multiple dimensions rather than relying on a single performance metric.

16. Store-Level Performance Drivers (Model Results)

This section investigates the drivers of store-level sales performance using multivariate regression models. To enable fair comparison across stores of different sizes, sales performance is measured as sales per square foot, and a logarithmic transformation is applied to stabilize variance and reduce the influence of extreme values. The modeling dataset is constructed at the store level, incorporating operational metrics, basket characteristics, category mix, temporal mix, and selected demographic variables.

Two complementary modeling approaches are used. First, an ordinary least squares (OLS) regression is estimated to provide interpretable coefficient estimates and statistical significance measures. Second, an Elastic Net regression is applied to address potential multicollinearity among predictors and to assess the robustness of results under regularization. Both models use the same predictor set to ensure comparability.

Model diagnostics are then examined using predicted versus actual values, residual distributions, and residual plots. These diagnostics assess overall fit, identify systematic patterns, and highlight stores that perform substantially above or below model expectations. Residual-based rankings are used to distinguish over- and under-performing stores after controlling for observable characteristics, providing a structured framework for identifying performance differences not explained by size, traffic, or market context.

```
# Build modeling dataset (one row per store)
model_data <- store_final %>%
  filter(
    !is.na(SquareFootage), SquareFootage > 0,
    !is.na(TotalSalesPerSqFt), TotalSalesPerSqFt > 0
  ) %>%
  mutate(
    SalesPerSqFt     = TotalSalesPerSqFt,
    log_SalesPerSqFt = log(SalesPerSqFt)
  ) %>%
  select(
    STORENAME,
    SalesPerSqFt,
    log_SalesPerSqFt,
    SquareFootage,
    TotalPopulation,
    Older25Years,
    PovertyLevel,
    Avg_BasketValue,
    AvgItemsPerBasket,
    TotalTransactionsPerSqFt,
    Share_BEER,
    Share_LIQ,
    Share_WINE,
    Share_Weekday,
    Share_Weekend
  ) %>%
  tidyr::drop_na()

# Basic sanity checks
nrow(model_data)
```

```
## [1] 27
```

```
str(model_data)
```

```
## tibble [27 × 15] (S3: tbl_df/tbl/data.frame)
##  $ STORENAME               : chr [1:27] "Aspen Hill" "Burtonsville" "Cabin John" "Clarksburg Village" ...
##  $ SalesPerSqFt            : num [1:27] 1223 1011 1730 1228 958 ...
##  $ log_SalesPerSqFt        : num [1:27] 7.11 6.92 7.46 7.11 6.86 ...
##  $ SquareFootage           : num [1:27] 7748 5500 5660 6163 6350 ...
##  $ TotalPopulation         : num [1:27] 37572 6478 18098 18185 8890 ...
##  $ Older25Years            : num [1:27] 24844 4479 12593 11796 6162 ...
##  $ PovertyLevel            : num [1:27] 3670 185 715 721 752 ...
##  $ Avg_BasketValue         : num [1:27] 32.2 35.5 46.4 36.6 39.6 ...
##  $ AvgItemsPerBasket       : num [1:27] 2.09 2.21 2.56 2.19 2.36 ...
##  $ TotalTransactionsPerSqFt: num [1:27] 57.7 45.5 70.4 55.4 41.2 ...
##  $ Share_BEER              : num [1:27] 0.0919 0.0725 0.0651 0.0773 0.057 ...
##  $ Share_LIQ               : num [1:27] 0.698 0.632 0.554 0.618 0.67 ...
##  $ Share_WINE              : num [1:27] 0.203 0.288 0.37 0.297 0.266 ...
##  $ Share_Weekday           : num [1:27] 0.633 0.691 0.686 0.674 0.686 ...
##  $ Share_Weekend           : num [1:27] 0.361 0.302 0.308 0.32 0.308 ...
```

```
summary(model_data$SalesPerSqFt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   439.9  1225.5  1524.8  1629.1  2010.0  3125.4
```

Interpretation: Modeling Dataset Construction

The final modeling dataset consists of 27 stores with complete and internally consistent measurements across operational, compositional, and demographic dimensions. The log transformation of sales per square foot reduces skewness and stabilizes variance, improving the suitability of linear modeling. Given the ratio of predictors to observations, the dataset supports explanatory analysis and relative ranking but requires cautious interpretation of coefficient magnitude and significance.

```r
# Predictor set for both OLS and Elastic Net
predictors <- c(
  "SquareFootage",
  "TotalPopulation",
  "Older25Years",
  "PovertyLevel",
  "Avg_BasketValue",
  "AvgItemsPerBasket",
  "TotalTransactionsPerSqFt",
  "Share_BEER",
  "Share_LIQ",
  "Share_WINE",
  "Share_Weekday",
  "Share_Weekend"
)

# Formula on log scale
model_formula <- as.formula(
  paste("log_SalesPerSqFt ~", paste(predictors, collapse = " + "))
)

model_formula
```

```
## log_SalesPerSqFt ~ SquareFootage + TotalPopulation + Older25Years + 
##     PovertyLevel + Avg_BasketValue + AvgItemsPerBasket + TotalTransactionsPerSqFt + 
##     Share_BEER + Share_LIQ + Share_WINE + Share_Weekday + Share_Weekend
```

```r
# Fit OLS model
lm_model <- lm(model_formula, data = model_data)
summary(lm_model)
```

```
## 
## Call:
## lm(formula = model_formula, data = model_data)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -0.15501 -0.05167  0.01520  0.05135  0.18978 
## 
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)    
## (Intercept)               2.796e+01  4.198e+01   0.666  0.51616    
## SquareFootage             1.632e-05  1.986e-05   0.822  0.42497    
## TotalPopulation           2.411e-05  3.004e-05   0.802  0.43577    
## Older25Years             -3.104e-05  3.924e-05  -0.791  0.44211    
## PovertyLevel             -1.621e-05  3.163e-05  -0.513  0.61626    
## Avg_BasketValue           2.041e-02  1.012e-02   2.016  0.06342 .  
## AvgItemsPerBasket        -2.195e-01  2.440e-01  -0.900  0.38351    
## TotalTransactionsPerSqFt  1.607e-02  9.151e-04  17.565  6.2e-11 ***
## Share_BEER                5.682e+01  1.914e+01   2.968  0.01017 *  
## Share_LIQ                 5.569e+01  1.845e+01   3.019  0.00920 ** 
## Share_WINE                5.566e+01  1.808e+01   3.079  0.00817 ** 
## Share_Weekday            -7.674e+01  3.565e+01  -2.153  0.04928 *  
## Share_Weekend            -8.051e+01  3.600e+01  -2.236  0.04212 *  
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.1092 on 14 degrees of freedom
## Multiple R-squared:  0.9756, Adjusted R-squared:  0.9546 
## F-statistic: 46.57 on 12 and 14 DF,  p-value: 3.703e-09
```

Drivers of Sales per Square Foot

The store-level OLS model explains variation in log-transformed Sales per Square Foot as a function of store size, local demographics, basket characteristics, traffic intensity, temporal mix, and category mix. With an adjusted R² of approximately 0.96, the model captures the majority of cross-store differences in sales productivity across the 27-store system.

The dominant driver of sales productivity is traffic intensity. TotalTransactionsPerSqFt is highly statistically significant ($p < 1e-10$) and exhibits the largest practical effect size, indicating that stores processing more transactions per unit of space consistently generate higher sales per square foot. This result is robust across model specifications and persists under penalized regression.

Category mix also contributes meaningfully to performance. The revenue shares of Beer, Liquor, and Wine are all positively associated with Sales per SqFt relative to the omitted Miscellaneous category. Because category shares sum to one, these coefficients should be interpreted as contrasts rather than independent levers. Stores whose revenue is concentrated in core alcohol categories tend to use their selling space more productively than stores with heavier miscellaneous sales.

Basket value plays a secondary role. Average basket value has a positive but borderline-significant association (p ≈ 0.06), suggesting that increasing spend per visit can improve productivity, but its effect is weaker than that of traffic or category mix.

Demographics and store size function primarily as controls rather than drivers. Square footage, total population, age composition, and poverty level do not exhibit strong independent effects once traffic and mix are accounted for. Within this relatively homogeneous county system, operational factors dominate structural ones.

Given the small sample size relative to the number of predictors, this model should be interpreted as descriptive and diagnostic rather than predictive. Its primary value lies in ranking drivers, benchmarking stores against modeled expectations, and identifying relative over- and under-performance.

```
# Rank OLS coefficients by magnitude
lm_tidy <- broom::tidy(lm_model) %>%
  dplyr::filter(term != "(Intercept)") %>%
  dplyr::arrange(desc(abs(estimate)))
lm_tidy
```

```
## # A tibble: 12 × 5
##    term                        estimate  std.error statistic  p.value
##    <chr>                          <dbl>      <dbl>     <dbl>    <dbl>
##  1 Share_Weekend            -80.5       36.0          -2.24  4.21e- 2
##  2 Share_Weekday            -76.7       35.7          -2.15  4.93e- 2
##  3 Share_BEER                56.8       19.1           2.97  1.02e- 2
##  4 Share_LIQ                 55.7       18.4           3.02  9.20e- 3
##  5 Share_WINE                55.7       18.1           3.08  8.17e- 3
##  6 AvgItemsPerBasket         -0.220      0.244        -0.900 3.84e- 1
##  7 Avg_BasketValue            0.0204     0.0101        2.02  6.34e- 2
##  8 TotalTransactionsPerSqFt   0.0161     0.000915     17.6   6.20e-11
##  9 Older25Years              -0.0000310  0.0000392    -0.791 4.42e- 1
## 10 TotalPopulation            0.0000241  0.0000300     0.802 4.36e- 1
## 11 SquareFootage              0.0000163  0.0000199     0.822 4.25e- 1
## 12 PovertyLevel              -0.0000162  0.0000316    -0.513 6.16e- 1
```

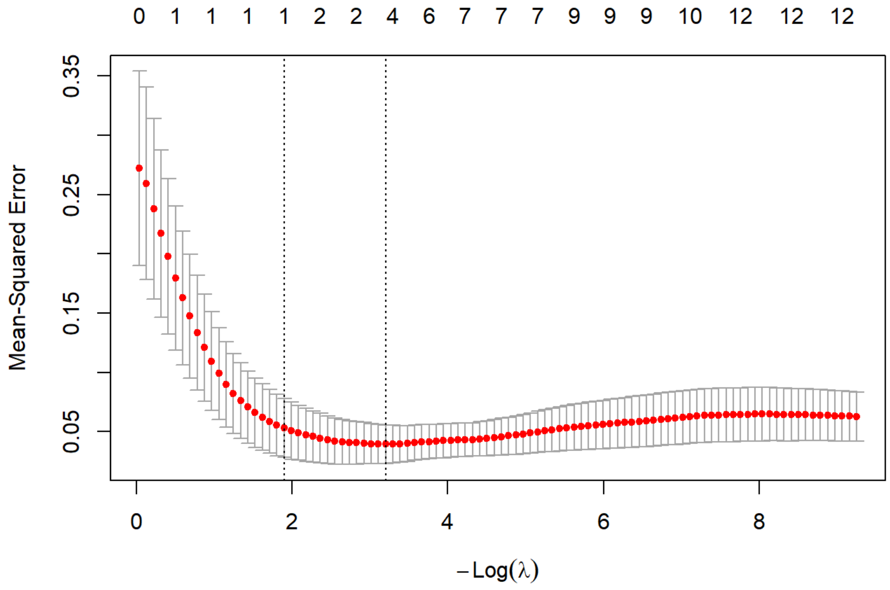
```
# Matrix X and response y for glmnet
x <- model_data %>%
  dplyr::select(all_of(predictors)) %>%
  as.matrix()

y <- model_data$log_SalesPerSqFt

set.seed(123)

# LOOCV Elastic Net to select lambda
cv_fit <- cv.glmnet(
  x, y,
  alpha       = 0.5,
  nfolds      = nrow(model_data),
  standardize = TRUE
)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
best_lambda <- cv_fit$lambda.1se
best_lambda
```

```
## [1] 0.149597
```

```
plot(cv_fit)
```



```
# Fit Elastic Net at selected lambda
enet_model <- glmnet(
  x, y,
  alpha       = 0.5,
  lambda      = best_lambda,
  standardize = TRUE
)

coef(enet_model)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                s0
## (Intercept)             6.40934930
## SquareFootage           .
## TotalPopulation         .
## Older25Years            .
## PovertyLevel            .
## Avg_BasketValue         .
## AvgItemsPerBasket       .
## TotalTransactionsPerSqFt 0.01229276
## Share_BEER              .
## Share_LIQ               .
## Share_WINE              .
## Share_Weekday           .
## Share_Weekend           .
```

Interpretation: OLS Coefficient Magnitudes

Transaction intensity (TotalTransactionsPerSqFt) emerges as the dominant predictor of sales productivity, with a highly statistically significant coefficient and a magnitude that exceeds all other predictors. Category mix variables (BEER, LIQ, WINE shares relative to MISC) are also consistently positive and significant, indicating that stores oriented toward core alcohol categories are more productive per square foot. In contrast, demographic variables and store size exhibit weak and statistically insignificant associations, suggesting they function primarily as controls rather than primary drivers.

```
# Add OLS predictions and residuals
model_data <- model_data %>%
  mutate(
    Pred_log_OLS    = predict(lm_model, newdata = .),
    Pred_Sales_OLS  = exp(Pred_log_OLS),
    Residual_OLS    = SalesPerSqFt - Pred_Sales_OLS,
    ResidualPct_OLS = Residual_OLS / Pred_Sales_OLS
  )

# In-sample fit metrics
lm_rmse <- sqrt(mean(model_data$Residual_OLS^2, na.rm = TRUE))
lm_mae  <- mean(abs(model_data$Residual_OLS),   na.rm = TRUE)

lm_rmse
```

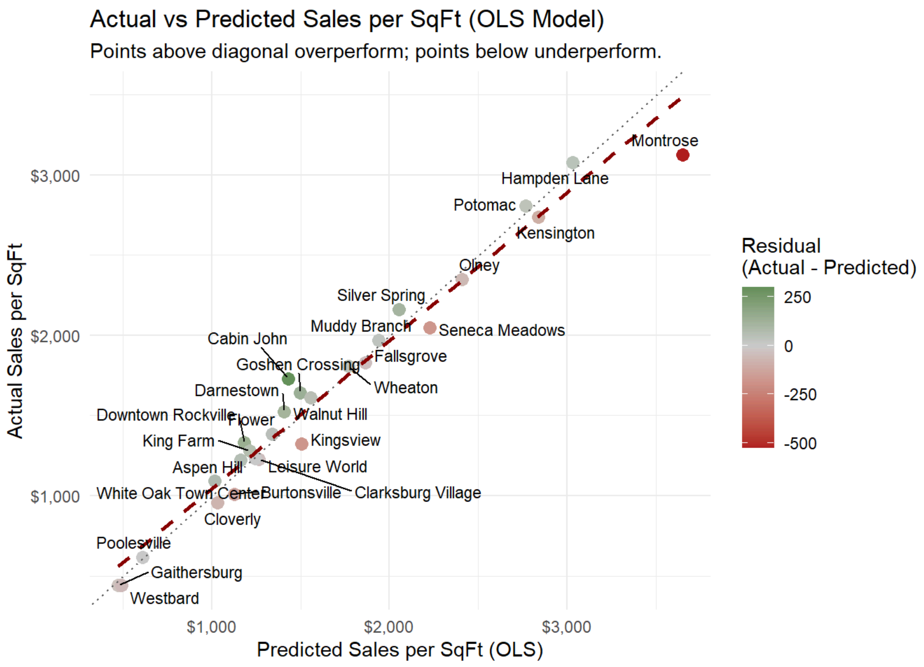
```
## [1] 144.1134
```

```
lm_mae
```

```
## [1] 99.46201
```

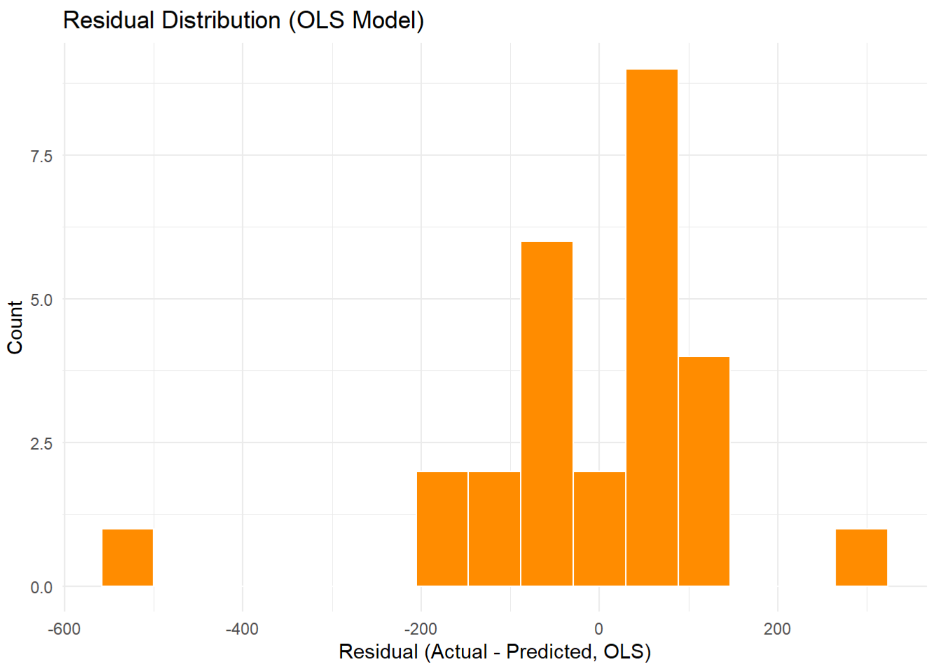
```
# Actual vs predicted sales per SqFt
ggplot(model_data,
       aes(x = Pred_Sales_OLS,
           y = SalesPerSqFt)) +
  geom_point(aes(color = Residual_OLS), size = 3) +
  geom_abline(slope = 1, intercept = 0,
              linetype = "dotted", color = "gray40") +
  geom_smooth(method = "lm", se = FALSE,
              linetype = "dashed", color = "darkred") +
  ggrepel::geom_text_repel(aes(label = STORENAME),
                           size = 3, max.overlaps = 100) +
  scale_x_continuous(labels = scales::dollar) +
  scale_y_continuous(labels = scales::dollar) +
  scale_color_gradient2(
    low      = "firebrick",
    mid      = "gray80",
    high     = "darkgreen",
    midpoint = 0
  ) +
  labs(
    title    = "Actual vs Predicted Sales per SqFt (OLS Model)",
    subtitle = "Points above diagonal overperform; points below underperform.",
    x        = "Predicted Sales per SqFt (OLS)",
    y        = "Actual Sales per SqFt",
    color    = "Residual\n(Actual - Predicted)"
  ) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

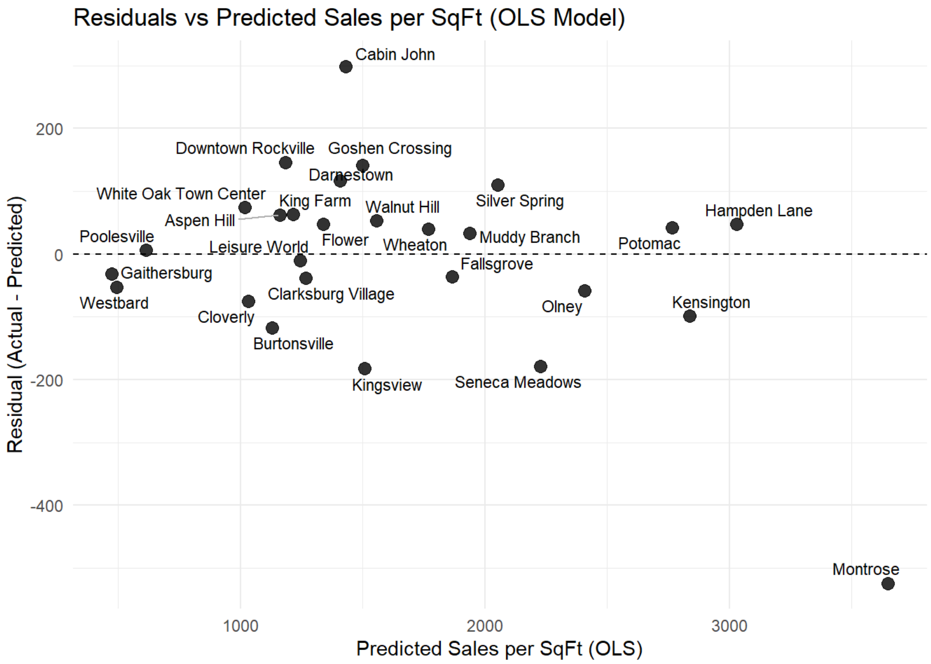

Actual vs Predicted Sales per SqFt (OLS Model)

Points above diagonal overperform; points below underperform.

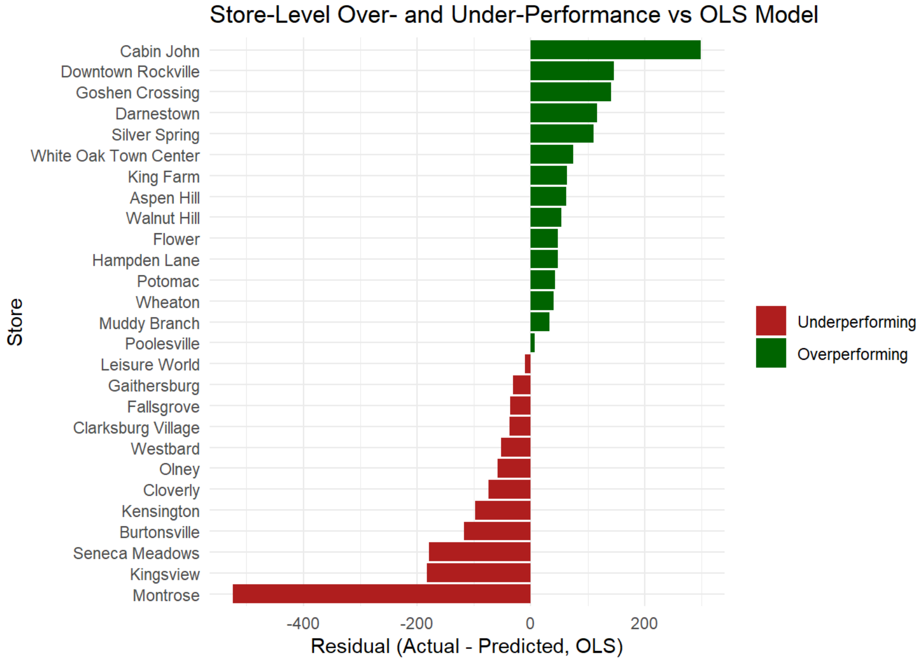
```
# Distribution of OLS residuals
ggplot(model_data, aes(x = Residual_OLS)) +
  geom_histogram(bins = 15, fill = "darkorange", color = "white") +
  labs(
    title = "Residual Distribution (OLS Model)",
    x     = "Residual (Actual - Predicted, OLS)",
    y     = "Count"
  ) +
  theme_minimal()
```


Residual Distribution (OLS Model)

```
# Residuals vs predicted values
ggplot(model_data,
       aes(x = Pred_Sales_OLS,
           y = Residual_OLS,
           label = STORENAME)) +
  geom_point(size = 3, alpha = 0.8) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  ggrepel::geom_text_repel(
    size          = 3,
    max.overlaps  = 100,
    segment.color = "grey70"
  ) +
  labs(
    title = "Residuals vs Predicted Sales per SqFt (OLS Model)",
    x     = "Predicted Sales per SqFt (OLS)",
    y     = "Residual (Actual - Predicted)"
  ) +
  theme_minimal()
```



```
# Store ranking by residual (over/under-performance vs model)
ggplot(model_data,
       aes(x = reorder(STORENAME, Residual_OLS),
           y = Residual_OLS,
           fill = Residual_OLS > 0)) +
  geom_col() +
  coord_flip() +
  scale_fill_manual(
    values = c("TRUE" = "darkgreen",
               "FALSE" = "firebrick"),
    labels = c("FALSE" = "Underperforming",
               "TRUE"  = "Overperforming"),
    name   = ""
  ) +
  labs(
    title = "Store-Level Over- and Under-Performance vs OLS Model",
    x     = "Store",
    y     = "Residual (Actual - Predicted, OLS)"
  ) +
  theme_minimal()
```



#Interpretation: Over- and Under-

Performance Relative to the Model

The residual-based ranking isolates systematic deviations between observed store performance and model-predicted performance after controlling for store size, traffic intensity, basket characteristics, category mix, and local demographics. Positive residuals identify stores that generate higher sales per square foot than expected given their structural characteristics, while negative residuals flag stores whose realized performance falls short of modeled expectations.

These residuals should not be interpreted as noise. Instead, they represent operational inefficiencies or advantages that are not fully captured by observable inputs. As such, they provide a defensible basis for prioritizing managerial attention: over-performing stores may reflect best practices worth replicating, while under-performing stores represent candidates for targeted intervention rather than blanket system-wide changes.

17. Ridge and LASSO on Store Data (glmnet-style)

This section extends the store-level modeling framework by applying regularized regression methods to assess model stability and predictor relevance in the presence of multicollinearity. Using the same predictor set defined in the previous section, Ridge and LASSO regression models are estimated on a training subset of stores and evaluated on a held-out test set. Both models operate on the logarithmic scale of sales per square foot and are subsequently back-transformed to the original scale for interpretability.

Ridge regression is used to shrink coefficient magnitudes while retaining all predictors, providing insight into how strongly variables contribute when multicollinearity is present. LASSO regression, in contrast, performs variable selection by shrinking some coefficients exactly to zero, thereby identifying a reduced subset of predictors that retain explanatory power. Model performance is evaluated using root mean squared error (RMSE) and mean absolute error (MAE) on the test set to assess out-of-sample predictive behavior.

```
set.seed(123)

# Train/test split
n <- nrow(model_data)
train_idx <- sample(seq_len(n), size = round(0.8 * n))

store_train <- model_data[train_idx, ]
store_test  <- model_data[-train_idx, ]


# x/y matrices for glmnet (train only)
x_train <- store_train %>%
  dplyr::select(all_of(predictors)) %>%
  as.matrix()

y_train <- store_train$log_SalesPerSqFt

# Test matrices
x_test  <- store_test %>%
  dplyr::select(all_of(predictors)) %>%
  as.matrix()

y_test  <- store_test$log_SalesPerSqFt

# Ridge: alpha = 0
set.seed(123)
cv_ridge <- cv.glmnet(x_train, y_train, alpha = 0)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
lambda_ridge <- cv_ridge$lambda.min
lambda_ridge
```

```
## [1] 0.04717665
```

```
ridge_model <- glmnet(
  x_train, y_train,
  alpha  = 0,
  lambda = lambda_ridge
)

coef(ridge_model)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                        s0
## (Intercept)               6.651349e+00
## SquareFootage             3.284430e-05
## TotalPopulation           8.241549e-07
## Older25Years             -2.807930e-07
## PovertyLevel             -4.145401e-05
## Avg_BasketValue          -4.012647e-03
## AvgItemsPerBasket        -3.176657e-01
## TotalTransactionsPerSqFt  1.518497e-02
## Share_BEER                1.532015e+00
## Share_LIQ                -1.514047e-01
## Share_WINE                2.627269e-01
## Share_Weekday             1.057589e+00
## Share_Weekend            -1.513256e+00
```

```
# Predict on test stores (on log scale, then back-transform)
pred_log_ridge <- predict(ridge_model, newx = x_test)
pred_Sales_ridge <- exp(pred_log_ridge)

rmse_ridge <- sqrt(mean((store_test$SalesPerSqFt - pred_Sales_ridge)^2))
mae_ridge  <- mean(abs(store_test$SalesPerSqFt - pred_Sales_ridge))

c(RMSE = rmse_ridge, MAE = mae_ridge)
```

```
##     RMSE      MAE 
## 483.3029 388.3795 
```

```
# LASSO: alpha = 1
set.seed(123)
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```
lambda_lasso <- cv_lasso$lambda.min
lambda_lasso
```

```
## [1] 0.06687169
```

```
lasso_model <- glmnet(
  x_train, y_train,
  alpha  = 1,
  lambda = lambda_lasso
)

# Coefficients (which predictors survive)
coef(lasso_model)
```

```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                                  s0
## (Intercept)              6.25328765
## SquareFootage            .         
## TotalPopulation          .         
## Older25Years             .         
## PovertyLevel             .         
## Avg_BasketValue          .         
## AvgItemsPerBasket        .         
## TotalTransactionsPerSqFt 0.01421421
## Share_BEER               .         
## Share_LIQ                .         
## Share_WINE               .         
## Share_Weekday            .         
## Share_Weekend            .         
```

```
# Predict on test stores (again transform back from log)
pred_log_lasso  <- predict(lasso_model, newx = x_test)
pred_Sales_lasso <- exp(pred_log_lasso)

rmse_lasso <- sqrt(mean((store_test$SalesPerSqFt - pred_Sales_lasso)^2))
mae_lasso  <- mean(abs(store_test$SalesPerSqFt - pred_Sales_lasso))

c(RMSE = rmse_lasso, MAE = mae_lasso)
```

```
##     RMSE      MAE 
## 363.4062 248.7671 
```

18. Elastic Net – Predictions, Metrics, and Drivers

Elastic Net regression is used to balance the properties of Ridge and LASSO by simultaneously performing coefficient shrinkage and variable selection. Elastic Net predictions are generated for all stores, and residuals are computed to enable direct comparison with the OLS model. In-sample performance metrics for OLS and Elastic Net are compared to assess relative fit, while coefficient estimates are examined to identify robust performance drivers that persist under regularization.

To evaluate model generalizability, a simple train–test validation is conducted at the store level. OLS models are refit on a reduced training sample and evaluated on held-out stores to assess changes in predictive accuracy. Given the limited number of stores, these validation results are interpreted as diagnostic rather than predictive, emphasizing the descriptive purpose of the modeling exercise. Finally, performance metrics across OLS, Elastic Net, Ridge, and LASSO models are summarized and visualized to highlight trade-offs between interpretability, stability, and predictive error.

```
# Elastic Net predictions and residuals
pred_log_enet <- as.numeric(predict(enet_model, newx = x))

model_data <- model_data %>%
  mutate(
    Pred_log_ENET   = pred_log_enet,
    Pred_Sales_ENET = exp(Pred_log_ENET),
    Residual_ENET   = SalesPerSqFt - Pred_Sales_ENET
  )

# Compare in-sample metrics: OLS vs ENET
rmse_ols  <- sqrt(mean(model_data$Residual_OLS^2,  na.rm = TRUE))
mae_ols   <- mean(abs(model_data$Residual_OLS),    na.rm = TRUE)

rmse_enet <- sqrt(mean(model_data$Residual_ENET^2, na.rm = TRUE))
mae_enet  <- mean(abs(model_data$Residual_ENET),   na.rm = TRUE)

fit_compare <- tibble(
  Model = c("OLS", "Elastic Net"),
  RMSE  = c(rmse_ols, rmse_enet),
  MAE   = c(mae_ols,  mae_enet)
)

fit_compare
```

```
## # A tibble: 2 × 3
##   Model        RMSE   MAE
##   <chr>       <dbl> <dbl>
## 1 OLS          144.  99.5
## 2 Elastic Net  245. 173.
```

```
# Extract and tidy Elastic Net coefficients
enet_mat <- as.matrix(coef(enet_model))

enet_tidy <- tibble(
  term     = rownames(enet_mat),
  estimate = as.numeric(enet_mat[, 1])
) %>%
  dplyr::filter(term != "(Intercept)", estimate != 0) %>%
  dplyr::mutate(
    effect_index = (exp(estimate) - 1) * 100
  ) %>%
  dplyr::arrange(desc(abs(effect_index)))

enet_tidy
```

```
## # A tibble: 1 × 3
##   term                      estimate effect_index
##   <chr>                        <dbl>        <dbl>
## 1 TotalTransactionsPerSqFt   0.0123         1.24
```

```
# Compare which variables OLS and ENET use
ols_terms  <- lm_tidy %>% dplyr::pull(term)
enet_terms <- enet_tidy %>% dplyr::pull(term)

drivers_compare <- tibble(
  term = sort(unique(c(ols_terms, enet_terms)))
) %>%
  mutate(
    In_OLS  = term %in% ols_terms,
    In_ENET = term %in% enet_terms
  )

drivers_compare
```

```
## # A tibble: 12 × 3
##    term                     In_OLS In_ENET
##    <chr>                    <lgl>  <lgl>  
##  1 Avg_BasketValue          TRUE   FALSE  
##  2 AvgItemsPerBasket        TRUE   FALSE  
##  3 Older25Years             TRUE   FALSE  
##  4 PovertyLevel             TRUE   FALSE  
##  5 Share_BEER               TRUE   FALSE  
##  6 Share_LIQ                TRUE   FALSE  
##  7 Share_Weekday            TRUE   FALSE  
##  8 Share_Weekend            TRUE   FALSE  
##  9 Share_WINE               TRUE   FALSE  
## 10 SquareFootage            TRUE   FALSE  
## 11 TotalPopulation          TRUE   FALSE  
## 12 TotalTransactionsPerSqFt TRUE   TRUE
```

```
# 18.1 Simple Out-of-Sample Check (Train/Test)

set.seed(123)  # fix split for reproducibility

# 18 training / 9 test
n_stores   <- nrow(model_data)
train_idx  <- sample(seq_len(n_stores), size = 18)
test_idx   <- setdiff(seq_len(n_stores), train_idx)

train_data <- model_data[train_idx, ]
test_data  <- model_data[test_idx, ]

# Refit OLS on training only
lm_train <- lm(model_formula, data = train_data)

# Predict on held-out test stores
test_data <- test_data %>%
  mutate(
    Pred_log_OLS_test   = predict(lm_train, newdata = test_data),
    Pred_Sales_OLS_test = exp(Pred_log_OLS_test),
    Residual_OLS_test   = SalesPerSqFt - Pred_Sales_OLS_test
  )

# Out-of-sample metrics
rmse_test <- sqrt(mean(test_data$Residual_OLS_test^2, na.rm = TRUE))
mae_test  <- mean(abs(test_data$Residual_OLS_test), na.rm = TRUE)

# Compare in-sample vs test
ols_fit_summary <- tibble(
  Dataset = c("In-sample (all 27 stores)", "Out-of-sample (9 test stores)"),
  RMSE    = c(lm_rmse, rmse_test),
  MAE     = c(lm_mae,  mae_test)
)

ols_fit_summary
```

```
## # A tibble: 2 × 3
##   Dataset                        RMSE   MAE
##   <chr>                         <dbl> <dbl>
## 1 In-sample (all 27 stores)      144.  99.5
## 2 Out-of-sample (9 test stores)  153. 128.
```

```
cat("\nIn-sample RMSE (OLS):", round(lm_rmse, 1),
    "\nTest RMSE (OLS):",      round(rmse_test, 1), "\n")
```

```
## 
## In-sample RMSE (OLS): 144.1 
## Test RMSE (OLS): 152.5
```

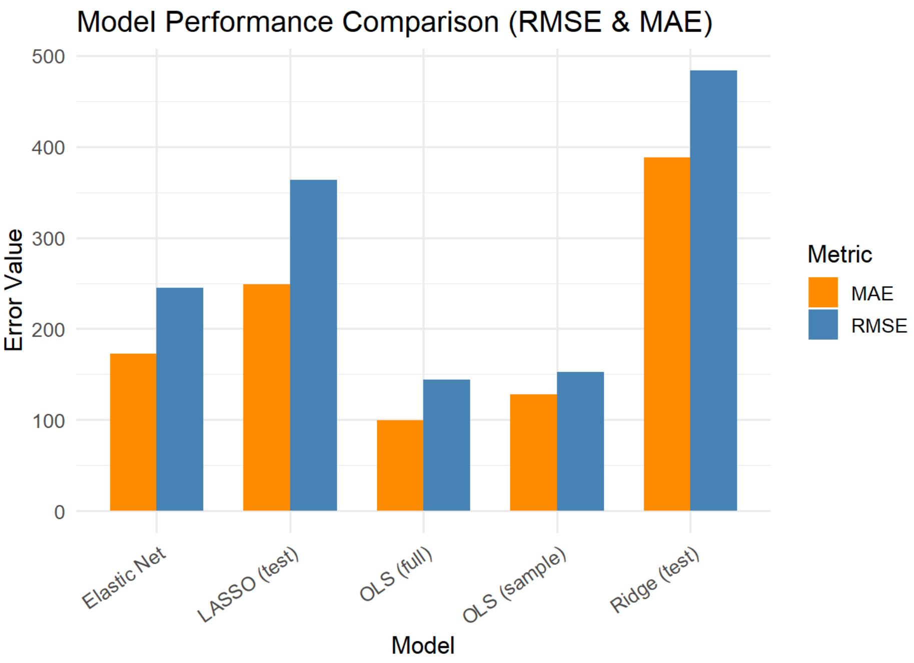
```
cat("In-sample MAE (OLS):",   round(lm_mae, 1),
    "\nTest MAE (OLS):",      round(mae_test, 1),  "\n\n")
```

```
## In-sample MAE (OLS): 99.5 
## Test MAE (OLS): 127.7
```

```
if (rmse_test > 1.5 * lm_rmse) {
  message("With only 27 stores, this model is descriptive, not a production forecasting model.")
}

model_compare <- tibble(
  Model = c("OLS (full)", "Elastic Net", "Ridge (test)", "LASSO (test)"),
  RMSE  = c(lm_rmse, rmse_enet, rmse_ridge, rmse_lasso),
  MAE   = c(lm_mae,  mae_enet,  mae_ridge,  mae_lasso)
)
model_compare
```

```
## # A tibble: 4 × 3
##   Model         RMSE   MAE
##   <chr>        <dbl> <dbl>
## 1 OLS (full)    144.  99.5
## 2 Elastic Net   245. 173.
## 3 Ridge (test)  483. 388.
## 4 LASSO (test)  363. 249.
```

```
model_df <- tibble(
  Model = c("OLS (full)", "OLS (sample)", "Elastic Net", "Ridge (test)", "LASSO (test)"),
  RMSE  = c(144.11, 152.52, 245.04, 483.30, 363.41),
  MAE   = c(99.46, 127.68, 172.54, 388.38, 248.77)
)

# Convert to long format for plotting
model_long <- model_df %>%
  pivot_longer(cols = c(RMSE, MAE), names_to = "Metric", values_to = "Value")

# Side-by-side bar chart
ggplot(model_long,
       aes(x = Model, y = Value, fill = Metric)) +
  geom_col(position = "dodge", width = 0.7) +
  labs(
    title = "Model Performance Comparison (RMSE & MAE)",
    x = "Model",
    y = "Error Value",
    fill = "Metric"
  ) +
  scale_fill_manual(values = c("RMSE" = "steelblue", "MAE" = "darkorange")) +
  theme_minimal(base_size = 13) +
  theme(
    axis.text.x = element_text(angle = 35, hjust = 1)
  )
```



Penalized Models (Ridge, LASSO, Elastic Net) – Narrative

To test whether simpler, more robust models outperform OLS in a small-sample setting, Ridge, LASSO, and Elastic Net regressions are estimated using the same predictor set and tuned via cross-validation.

The Elastic Net model (α = 0.5) collapses to an extremely sparse solution, retaining only TotalTransactionsPerSqFt as a non-zero coefficient. This confirms traffic intensity as the single most robust predictor of sales productivity. However, this simplicity comes at a cost: Elastic Net exhibits substantially worse in-sample fit than OLS, indicating that basket metrics and mix variables, while less robust individually, jointly explain meaningful variation.

Train–test splits further show that neither Ridge nor LASSO improves predictive performance relative to OLS. With only 27 stores, the variance reduction gained through regularization does not offset the information loss from coefficient shrinkage.

The correct conclusion is not that regularization “fails,” but that data depth limits its usefulness in this setting. Penalized models confirm the primacy of traffic, but OLS remains the most informative descriptive model given the available data.

19. Store-Level Over / Under-Performance vs OLS Model

This section summarizes store-level performance relative to model expectations using residuals from the OLS regression. For each store, actual sales per square foot are compared to model-predicted values, and the resulting residuals indicate the degree of over- or under-performance after controlling for observable characteristics such as store size, traffic, basket behavior, product mix, and demographics.

Positive residuals identify stores that outperform the model’s expectations, while negative residuals indicate under-performance. Ranking stores by residual size provides a concise diagnostic view of which locations deviate most strongly from modeled behavior and highlights candidates for deeper qualitative or operational investigation.

```r
# Table of store-level actual vs predicted vs residual
store_perf_ols <- model_data %>%
  select(
    STORENAME,
    SalesPerSqFt,
    Pred_Sales_OLS,
    Residual_OLS
  ) %>%
  arrange(desc(Residual_OLS))

# Top 5 over-performers
head(store_perf_ols, 5)
```

```
## # A tibble: 5 × 4
##   STORENAME          SalesPerSqFt Pred_Sales_OLS Residual_OLS
##   <chr>                     <dbl>          <dbl>        <dbl>
## 1 Cabin John                1730.          1431.         299.
## 2 Downtown Rockville        1331.          1184.         146.
## 3 Goshen Crossing           1641.          1499.         141.
## 4 Darnestown                1525.          1408.         117.
## 5 Silver Spring             2162.          2051.         111.
```

```r
# Bottom 5 under-performers
tail(store_perf_ols, 5)
```

```
## # A tibble: 5 × 4
##   STORENAME      SalesPerSqFt Pred_Sales_OLS Residual_OLS
##   <chr>                 <dbl>          <dbl>        <dbl>
## 1 Kensington            2738.          2836.        -98.4
## 2 Burtonsville          1011.          1128.       -117. 
## 3 Seneca Meadows        2049.          2227.       -179. 
## 4 Kingsview             1324.          1506.       -182. 
## 5 Montrose              3125.          3649.       -524. 
```

Interpretation: Store-Level Over- and Under-Performance

Store-level residuals are computed as the difference between actual Sales per Square Foot and OLS-predicted Sales per Square Foot. These residuals measure performance relative to modeled expectations, conditional on store size, traffic, basket characteristics, mix, and demographics.

Positive residuals indicate stores that generate more sales per square foot than the model would predict, while negative residuals identify stores that underperform relative to their structural and operational fundamentals.

The residual ranking reveals that several stores substantially outperform expectations, converting their space and traffic into revenue particularly efficiently. Conversely, several stores underperform relative to their modeled potential. Importantly, this is not a ranking of “good” versus “bad” stores. Some underperforming stores still exhibit high absolute Sales per SqFt but appear to leave additional value unrealized given their fundamentals.

Residual analysis therefore serves as a potential lens, not a performance verdict. It highlights where managerial attention may yield the greatest marginal gains.

20. Market Basket Analysis (Y24–25)

This section applies Market Basket Analysis to examine item co-purchase patterns during FY24–25 transactions. The analysis focuses on non-refund transactions and constructs one basket per transaction, where each basket represents the set of distinct items purchased together in a single transaction.

To reduce noise and computational complexity, the analysis is restricted to the 200 most frequently purchased items across all transactions. This filtering step ensures that association rules are driven by commonly purchased products rather than rare or idiosyncratic items. Association rules are generated using the Apriori algorithm with conservative support and confidence thresholds, and rule length is limited to two- and three-item combinations to maintain interpretability.

The resulting rules are filtered by lift to identify item combinations that co-occur more frequently than would be expected by chance. These associations are intended to provide descriptive insight into common purchasing patterns and potential cross-selling relationships rather than to serve as prescriptive or causal recommendations.

```
# Restrict to non-refund Y24-25 transactions
transactions_fs <- transactions %>%
  filter(!IsRefund)

# Top 200 items by frequency across all baskets
item_freq <- transactions_fs %>%
  count(DESCRIPTION, sort = TRUE)

top_n_items <- item_freq %>%
  slice_head(n = 200) %>%
  pull(DESCRIPTION)

# Build one basket per TRANSACTIONID using top-N items only
baskets_mba <- transactions_fs %>%
  filter(DESCRIPTION %in% top_n_items) %>%
  group_by(TRANSACTIONID) %>%
  summarise(
    Items = list(unique(DESCRIPTION)),
    .groups = "drop"
  )

basket_list <- baskets_mba$Items
transactions_mba <- as(basket_list, "transactions")

summary(transactions_mba)
```

```
## transactions as itemMatrix in sparse format with
##  3238237 rows (elements/itemsets/transactions) and
##  200 columns (items) and a density of 0.006168515 
## 
## most frequent items:
##       TITOS HM VODKA 1 75L FIREBALL CINN WSKY 50ML 10 
##                     173297                     134240 
## TITO S HANDMADE VODK 750ML           SMIRNOFF 80 50ML 
##                      89444                      70644 
##         BOWMAN VODKA 375ML                    (Other) 
##                      58983                    3468415 
## 
## element (itemset/transaction) length distribution:
## sizes
##       1       2       3       4       5       6       7       8       9      10 
## 2686573  437669   74543   20467    7948    4002    2256    1603    1041     674 
##      11      12      13      14      15      16      17      18      19      20 
##     447     322     234     148     118      72      38      27      13      11 
##      21      22      23      24      25      26      27      30      31 
##      11       8       3       1       1       3       1       1       2 
## 
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   1.000   1.000   1.000   1.234   1.000  31.000 
## 
## includes extended item information - examples:
##                      labels
## 1 1800 SILVER TEQUILA 1 75L
## 2 1800 TEQUILA SILVER 750ML
## 3      99 SCHNPS APPLE 50ML
```

```
# Apriori rules on reduced item set
rules <- apriori(
  transactions_mba,
  parameter = list(
    supp   = 0.0002,
    conf   = 0.20,
    minlen = 2,
    maxlen = 3,
    target = "rules"
  )
)
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.2    0.1    1 none FALSE            TRUE       5   2e-04      2
##  maxlen target  ext
##       3  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 647
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[200 item(s), 3238237 transaction(s)] done [0.81s].
## sorting and recoding items ... [200 item(s)] done [0.06s].
## creating transaction tree ... done [5.65s].
## checking subsets of size 1 2 3
```

```
## Warning in apriori(transactions_mba, parameter = list(supp = 2e-04, conf = 0.2,
## : Mining stopped (maxlen reached). Only patterns up to a length of 3 returned!
```

```
##  done [0.03s].
## writing ... [14 rule(s)] done [0.00s].
## creating S4 object  ... done [0.71s].
```

```
# Filter rules with modest lift and sort by lift
rules_filtered <- subset(rules, lift > 1.05)
rules_top <- sort(rules_filtered, by = "lift", decreasing = TRUE)

# Top 20 strongest rules
inspect(head(rules_top, 20))
```

```
##      lhs                                 rhs                                  support confidence     coverage     lift co
unt
## [1]  {NEW AMSTERDAM PASSION F 50ML,
##       NEW AMSTERDAM PEACH 50ML}       => {NEW AMSTERDAM APPLE 50ML}      0.0004425247  0.4490755 0.0009854127 89.22647  1
433
## [2]  {NEW AMSTERDAM GRAPEFRUIT 50ML,
##       NEW AMSTERDAM PEACH 50ML}       => {NEW AMSTERDAM APPLE 50ML}      0.0002952224  0.4310189 0.0006849406 85.63882
956
## [3]  {NEW AMSTERDAM APPLE 50ML,
##       NEW AMSTERDAM GRAPEFRUIT 50ML}  => {NEW AMSTERDAM PASSION F 50ML}  0.0002661942  0.4434156 0.0006003267 81.53804
862
## [4]  {NEW AMSTERDAM GRAPEFRUIT 50ML,
##       NEW AMSTERDAM PEACH 50ML}       => {NEW AMSTERDAM PASSION F 50ML}  0.0002890462  0.4220018 0.0006849406 77.60033
936
## [5]  {NEW AMSTERDAM GRAPEFRUIT 50ML,
##       NEW AMSTERDAM PASSION F 50ML}   => {NEW AMSTERDAM APPLE 50ML}      0.0002661942  0.3895165 0.0006833966 77.39273
862
## [6]  {NEW AMSTERDAM APPLE 50ML,
##       NEW AMSTERDAM GRAPEFRUIT 50ML}  => {NEW AMSTERDAM PEACH 50ML}      0.0002952224  0.4917695 0.0006003267 74.15098
956
## [7]  {NEW AMSTERDAM APPLE 50ML,
##       NEW AMSTERDAM PASSION F 50ML}   => {NEW AMSTERDAM PEACH 50ML}      0.0004425247  0.4737190 0.0009341503 71.42924  1
433
## [8]  {NEW AMSTERDAM APPLE 50ML,
##       NEW AMSTERDAM PEACH 50ML}       => {NEW AMSTERDAM PASSION F 50ML}  0.0004425247  0.3759182 0.0011771838 69.12618  1
433
## [9]  {NEW AMSTERDAM GRAPEFRUIT 50ML,
##       NEW AMSTERDAM PASSION F 50ML}   => {NEW AMSTERDAM PEACH 50ML}      0.0002890462  0.4229553 0.0006833966 63.77488
936
## [10] {NEW AMSTERDAM PASSION F 50ML,
##       NEW AMSTERDAM PEACH 50ML}       => {NEW AMSTERDAM GRAPEFRUIT 50ML} 0.0002890462  0.2933250 0.0009854127 56.52221
936
## [11] {NEW AMSTERDAM APPLE 50ML,
##       NEW AMSTERDAM PASSION F 50ML}   => {NEW AMSTERDAM GRAPEFRUIT 50ML} 0.0002661942  0.2849587 0.0009341503 54.91007
862
## [12] {NEW AMSTERDAM APPLE 50ML,
##       NEW AMSTERDAM PEACH 50ML}       => {NEW AMSTERDAM GRAPEFRUIT 50ML} 0.0002952224  0.2507870 0.0011771838 48.32536
956
## [13] {NEW AMSTERDAM APPLE 50ML}       => {NEW AMSTERDAM PEACH 50ML}      0.0011771838  0.2338937 0.0050329855 35.26743  3
812
## [14] {L MARCA PRO ROSE 750ML}         => {L MARCA PROSECCO 750ML}        0.0008535509  0.2408085 0.0035445213 14.48868  2
764
```

Interpretation: Market Basket Associations

Market basket analysis is conducted using non-refund transactions, restricting the item universe to the top 200 SKUs by frequency to ensure computational feasibility. Each transaction is treated as a basket of unique item descriptions, and association rules are estimated using the Apriori algorithm with moderate support and confidence thresholds.

The resulting rule set is highly concentrated. The strongest associations occur within narrow product clusters, particularly among New Amsterdam 50ml flavored SKUs, where customers frequently purchase multiple flavors together. A smaller but clear association also exists between closely related prosecco and prosecco-rosé products.

These results indicate that high-lift associations do exist, but they are primarily micro-patterns within specific brands and size formats rather than broad cross-category relationships. At current thresholds, the analysis does not yet yield system-wide merchandising guidance.

To become operationally useful, future market basket analyses would need to be segmented by category, store type, or margin class, and focused on strategic rather than purely high-volume items. As implemented here, the MBA provides insight into localized bundling behavior rather than actionable planogram design.

21. Bootstrapping

This section applies bootstrap resampling methods to quantify uncertainty in key performance metrics without relying on strong parametric assumptions. Bootstrapping is particularly appropriate in this context because several metrics of interest, such as basket values and store-level sales efficiency, exhibit skewed distributions and are based on relatively small sample sizes at the store level.

First, bootstrap resampling is used to estimate a confidence interval for the median basket value at the transaction level. The median is selected as a robust measure of central tendency that is less sensitive to extreme values than the mean. To manage computational constraints, a random subsample of baskets is used when the total number of transactions exceeds a predefined threshold, while preserving the underlying distribution.

Second, bootstrapping is applied at the store level to compare mean sales per square foot between contrasting performance quadrants. Specifically, bootstrap distributions are generated for stores classified as “High Spend & High Traffic” and “Low Spend & Low Traffic.” Percentile-based confidence intervals are used to assess whether differences in average performance between these groups are substantively meaningful.

These results are intended to support descriptive comparison and uncertainty assessment rather than formal hypothesis testing.

```
# 21.1 Per-transaction basket metrics

# Build per-transaction basket metrics
basket_metrics <- transactions %>%
  filter(!IsRefund) %>%
  group_by(TRANSACTIONID) %>%
  summarise(
    BasketTotal = sum(NETAMOUNT, na.rm = TRUE),
    BasketItems = n_distinct(ITEMID),
    .groups     = "drop"
  )

basket_values <- basket_metrics %>%
  filter(!is.na(BasketTotal)) %>%
  pull(BasketTotal)

length(basket_values)
```

```
## [1] 6061745
```

```
# Subsample baskets for memory control
set.seed(123)

max_n <- 50000L
if (length(basket_values) > max_n) {
  basket_sample <- sample(basket_values, size = max_n, replace = FALSE)
} else {
  basket_sample <- basket_values
}

length(basket_sample)
```

```
## [1] 50000
```

```
# Bootstrap statistic: median basket total
boot_median_fn <- function(data, indices) {
  median(data[indices], na.rm = TRUE)
}

set.seed(123)

boot_median_res <- boot(
  data      = basket_sample,
  statistic = boot_median_fn,
  R         = 1000
)

# Percentile confidence interval for median basket value
ci_lower <- quantile(boot_median_res$t, 0.025, na.rm = TRUE)
ci_upper <- quantile(boot_median_res$t, 0.975, na.rm = TRUE)

ci_lower
```

```
##  2.5% 
## 23.99
```

```
ci_upper
```

```
## 97.5% 
## 24.97
```

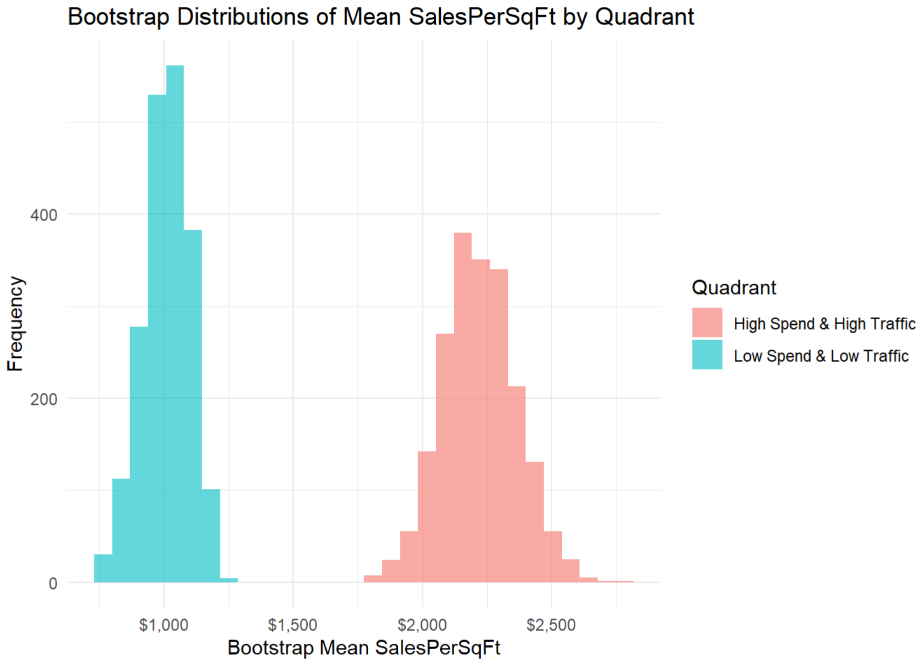
```r
# 21.2 Bootstrap by quadrant: High vs Low Sales/Traffic -------

# Prepare data for bootstrapping by quadrant
quad_data <- store_quadrant %>%
  select(STORENAME, TotalSalesPerSqFt, Quad_SalesTraffic) %>%
  filter(!is.na(TotalSalesPerSqFt), !is.na(Quad_SalesTraffic))

# Bootstrap function for mean SalesPerSqFt
boot_mean_quadrant <- function(data, indices) {
  d <- data[indices, , drop = FALSE]
  mean(d$TotalSalesPerSqFt, na.rm = TRUE)
}

# Compare High Spend & High Traffic vs Low Spend & Low Traffic
quad_high <- quad_data %>%
  filter(Quad_SalesTraffic == "High Spend & High Traffic")

quad_low <- quad_data %>%
  filter(Quad_SalesTraffic == "Low Spend & Low Traffic")

set.seed(123)

boot_high <- boot(
  data      = quad_high,
  statistic = boot_mean_quadrant,
  R         = 2000
)

boot_low <- boot(
  data      = quad_low,
  statistic = boot_mean_quadrant,
  R         = 2000
)

# Percentile CIs for each quadrant mean
boot_high_ci <- boot.ci(boot_high, type = "perc")
boot_low_ci  <- boot.ci(boot_low,  type = "perc")

boot_high_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
## 
## CALL : 
## boot.ci(boot.out = boot_high, type = "perc")
## 
## Intervals : 
## Level     Percentile     
## 95%   (1947, 2512 )  
## Calculations and Intervals on Original Scale
```

```r
boot_low_ci
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
## 
## CALL : 
## boot.ci(boot.out = boot_low, type = "perc")
## 
## Intervals : 
## Level     Percentile     
## 95%   ( 819, 1177 )  
## Calculations and Intervals on Original Scale
```

```
# Combine bootstrap draws for visualization
boot_quad_df <- tibble(
  mean_SalesPerSqFt = c(boot_high$t[, 1], boot_low$t[, 1]),
  Quadrant          = rep(
    c("High Spend & High Traffic",
      "Low Spend & Low Traffic"),
    each = nrow(boot_high$t)
  )
)

# Summary table for export
boot_summary <- tibble(
  Group    = c("High Spend & High Traffic",
               "Low Spend & Low Traffic"),
  Mean     = c(boot_high$t0, boot_low$t0),
  CI_Lower = c(boot_high_ci$percent[4], boot_low_ci$percent[4]),
  CI_Upper = c(boot_high_ci$percent[5], boot_low_ci$percent[5])
)

# Plot bootstrap distributions
ggplot(boot_quad_df,
       aes(x = mean_SalesPerSqFt, fill = Quadrant)) +
  geom_histogram(position = "identity",
                 alpha    = 0.6,
                 bins     = 30) +
  labs(
    title = "Bootstrap Distributions of Mean SalesPerSqFt by Quadrant",
    x     = "Bootstrap Mean SalesPerSqFt",
    y     = "Frequency",
    fill  = "Quadrant"
  ) +
  scale_x_continuous(labels = scales::dollar) +
  theme_minimal()
```



# Interpretation: Bootstrap

Confidence Interval for Basket Value

Bootstrapping is used to quantify uncertainty around key performance metrics without relying on parametric assumptions.

First, a non-parametric bootstrap of transaction-level basket totals estimates the median basket value. Using 1,000 resamples of a large random subsample, the 95% confidence interval for the median basket value is approximately $24–$25. This confirms that the typical customer makes a relatively modest purchase, and that higher average basket values observed at the store level are driven by a smaller number of large transactions.

Second, mean Sales per Square Foot is bootstrapped for two structurally distinct store groups: High Spend & High Traffic and Low Spend & Low Traffic. The resulting confidence intervals are well separated, with high-high stores exhibiting roughly double the productivity of low-low stores. This difference is far larger than can be explained by sampling variability alone.

These results confirm that quadrant position reflects economically meaningful differences, not noise. Moving a store across quadrants represents a substantial productivity shift rather than a marginal improvement.

Interpretation: Performance Differences Across Store Quadrants

Bootstrap distributions for mean sales per square foot show clear separation between high spend/high traffic stores and low spend/low traffic stores. The limited overlap between confidence intervals suggests that these performance differences are systematic rather than driven by random variation. This reinforces the validity of the quadrant framework as a meaningful classification of store performance regimes.

22. Store Typology & Action Flags

This section synthesizes results from the quadrant analyses and regression modeling into a consolidated store-level typology. Store performance is characterized using normalized efficiency metrics (sales, transactions, and baskets per square foot), quadrant classifications, and residuals from the OLS model. Residual-based performance flags are used to indicate whether a store performs above or below model expectations after controlling for observable characteristics.

Based on these combined indicators, descriptive opportunity flags are assigned to highlight potential performance levers associated with traffic intensity and basket behavior. These flags are not intended as prescriptive recommendations but rather as a structured diagnostic framework to support comparative evaluation across stores. The resulting typology provides a concise summary of each store’s relative position within the system and identifies dimensions along which performance differences are most pronounced.

```
# 21.1 Combine quadrants and model residuals
store_typology <- store_quadrant %>%
  select(
    STORENAME,
    TotalSalesPerSqFt,
    TotalTransactionsPerSqFt,
    TotalBasketsPerSqFt,
    Quad_SalesTraffic,
    Quad_SalesBaskets
  ) %>%
  left_join(
    model_data %>% select(STORENAME, Residual_OLS),
    by = "STORENAME"
  ) %>%
  mutate(
    PerfFlag = case_when(
      Residual_OLS >=  75 ~ "Strong over-performer",
      Residual_OLS >=  25 ~ "Moderate over-performer",
      Residual_OLS <= -75 ~ "Strong under-performer",
      Residual_OLS <= -25 ~ "Moderate under-performer",
      TRUE                ~ "Near model expectation"
    )
  )
```

```
# 21.2 Simple opportunity flags (what lever to pull)
store_typology <- store_typology %>%
  mutate(
    Traffic_Opportunity = case_when(
      Quad_SalesTraffic == "High Spend & Low Traffic" ~ "Increase traffic (awareness/location/selection)",
      Quad_SalesTraffic == "Low Spend & Low Traffic"  ~ "Fundamental issue: traffic + value",
      Quad_SalesTraffic == "Low Spend & High Traffic" ~ "Improve conversion / basket value",
      TRUE                                            ~ "Traffic OK"
    ),
    Basket_Opportunity = case_when(
      Quad_SalesBaskets == "High Spend & Low Baskets" ~ "Increase visit frequency / basket count",
      Quad_SalesBaskets == "Low Spend & High Baskets" ~ "Increase basket value (mix, pricing, upsell)",
      Quad_SalesBaskets == "Low Spend & Low Baskets"  ~ "Structural weakness (frequency + value)",
      TRUE                                            ~ "Basket density OK"
    )
  )

# 21.3 Final typology table
store_typology %>%
  arrange(desc(TotalSalesPerSqFt)) %>%
  select(
    STORENAME,
    TotalSalesPerSqFt,
    TotalTransactionsPerSqFt,
    TotalBasketsPerSqFt,
    Quad_SalesTraffic,
    Quad_SalesBaskets,
    PerfFlag,
    Traffic_Opportunity,
    Basket_Opportunity
  )
```

```
## # A tibble: 27 × 9
##    STORENAME      TotalSalesPerSqFt TotalTransactionsPerSqFt TotalBasketsPerSqFt
##    <chr>                      <dbl>                    <dbl>               <dbl>
##  1 Montrose                   3125.                    136.                 78.3
##  2 Hampden Lane               3077.                    118.                 60.8
##  3 Potomac                    2808.                     97.1                51.6
##  4 Kensington                 2738.                    122.                 68.4
##  5 Olney                      2349.                    102.                 61.1
##  6 Silver Spring              2162.                     96.2                59.2
##  7 Seneca Meadows             2049.                     91.3                55.5
##  8 Muddy Branch               1971.                     80.9                45.7
##  9 Fallsgrove                 1829.                     80.7                49.3
## 10 Wheaton                    1809.                     86.1                61.6
## # ℹ 17 more rows
## # ℹ 5 more variables: Quad_SalesTraffic <chr>, Quad_SalesBaskets <chr>,
## #   PerfFlag <chr>, Traffic_Opportunity <chr>, Basket_Opportunity <chr>
```

Interpretation: Store Typology and Operational Implications

The final store typology integrates three analytical elements: quadrant position based on traffic and basket density, residual-based performance relative to the OLS model, and qualitative opportunity flags derived from these measures.

Quadrants describe a store’s structural position, indicating whether it is fundamentally strong or weak in traffic and sales efficiency. Residuals capture performance relative to potential, indicating whether a store is outperforming or underperforming what its fundamentals would suggest.

These dimensions are combined into performance flags and translated into clear operational guidance. Stores classified as high-priority are those that underperform relative to expectations, regardless of their absolute revenue levels. Several high-priority stores are among the system’s strongest in raw sales terms, but the analysis suggests they could deliver materially higher productivity if specific constraints were addressed.

This typology serves as the bridge from analytics to action. It identifies which stores to focus on first and which lever—traffic, basket value, or both—is most likely to yield gains, providing a structured framework for targeted intervention rather than one-size-fits-all recommendations.

23. Priority Store List

This section derives a prioritized list of stores for further review based on residual performance relative to the OLS model. Stores are ranked by residual values, with lower residuals indicating greater under-performance after controlling for observable factors such as store size, traffic intensity, basket behavior, category mix, and demographic context. Performance categories are translated into broad priority levels to distinguish stores that warrant closer analytical attention from those performing near or above expectations.

The resulting priority list is intended as a screening tool rather than a definitive action plan. It highlights locations where performance deviates most strongly from modeled expectations and where additional qualitative investigation may be most informative. This approach supports resource allocation for further analysis while avoiding prescriptive operational conclusions.

```
priority_stores <- store_typology %>%
  arrange(Residual_OLS) %>%   # worst first
  mutate(
    ActionPriority = case_when(
      PerfFlag %in% c("Strong under-performer", "Moderate under-performer") ~ "High",
      PerfFlag == "Near model expectation"                                   ~ "Medium",
      TRUE                                                                  ~ "Low"
    )
  )

# Top 7 under-performer
priority_stores %>%
  filter(ActionPriority == "High") %>%
  head(7)
```

```
## # A tibble: 7 × 11
##   STORENAME      TotalSalesPerSqFt TotalTransactionsPerSqFt TotalBasketsPerSqFt
##   <chr>                      <dbl>                    <dbl>               <dbl>
## 1 Montrose                   3125.                    136.                 78.3
## 2 Kingsview                  1324.                     62.6                42.0
## 3 Seneca Meadows             2049.                     91.3                55.5
## 4 Burtonsville               1011.                     45.5                28.5
## 5 Kensington                 2738.                    122.                 68.4
## 6 Cloverly                    958.                     41.2                24.2
## 7 Olney                      2349.                    102.                 61.1
## # ℹ 7 more variables: Quad_SalesTraffic <chr>, Quad_SalesBaskets <chr>,
## #   Residual_OLS <dbl>, PerfFlag <chr>, Traffic_Opportunity <chr>,
## #   Basket_Opportunity <chr>, ActionPriority <chr>
```

Interpretation: Priority Stores for Intervention

The priority list highlights stores with the largest negative residuals relative to model expectations, indicating underperformance beyond what can be explained by observable characteristics. These locations represent the highest potential return for targeted operational interventions and form a natural starting point for deeper qualitative review.

Limitations and Next Steps

This analysis delivers a coherent view of store-level performance, but it is important to be clear about its limits.

Key limitations

- **Omitted variables.** With only 27 stores and 12 predictors, even a well-specified OLS is at risk of over-interpreting noise. The high R² is partly a function of model complexity relative to N.
- **Omitted variables.** Critical drivers such as rent, competition density, store hours, staffing quality, local traffic patterns, and detailed assortment depth are not in the data. Their omission limits the model’s ability to fully explain why some stores over- or under-perform.
- **Collinearity in mix variables.** BEER, LIQ, WINE, and MISC shares must sum to 1. Individual coefficients on these shares should therefore be read as a contrast versus MISC, not as fully independent effects. The main message is: being a “core alcohol” store rather than MISC-heavy is favorable for productivity.
- **MBA is shallow at current thresholds.** With the chosen support and confidence levels, market basket analysis mostly surfaces tight flavor-family patterns (New Amsterdam minis, Prosecco variants), not a broad set of actionable cross-category rules.

Concrete next steps:

1. Refine the model with more years of data and richer store-level features (competition, hours, staffing) to separate persistent signals from one-year noise.
2. Decompose the quadrants by category (e.g., liquor-only `SalesPerSqFt`) to see whether stores are strong or weak in specific departments.
3. Expand MBA by:
   - Running separate models within large categories.
   - Lowering support thresholds within-store or within-category.
   - Focusing on rules that involve high-margin items.
4. Turn the typology into an action plan, assigning owners and timelines for each high-priority store, with before/after tracking of `SalesPerSqFt` , traffic, and basket metrics.

```r
# EXPORT KEY ANALYTIC TABLES

write_csv(store_final, "ABS_Store_Final_KPIs.csv")
write_csv(model_data, "ABS_Model_Data_For_Regression.csv")
write_csv(store_perf_ols, "ABS_Store_Over_Under_Performance_OLS.csv")
write_csv(store_typology, "ABS_Store_Typology_And_Action_Flags.csv")
write_csv(priority_stores, "ABS_Priority_Store_List.csv")

rules_df <- as.data.frame(as(rules_top, "data.frame"))
write_csv(rules_df, "ABS_Market_Basket_Association_Rules.csv")

write_csv(boot_summary, "ABS_Bootstrap_Quadrant_Comparisons.csv")

# END EXPORT BLOCK
```