FashionRetail_Project

Riya Marwaha

2025-09-15

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
# ======================== Retail Returns  ========================

install.packages(c("dplyr","forcats","ggplot2","caret","pROC","glmnet","randomForest","rpart","rpart.pl
```

```
## Installing packages into '/cloud/lib/x86_64-pc-linux-gnu-library/4.5'
## (as 'lib' is unspecified)
```

```
library(dplyr)
```

```
##
## 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(forcats)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(glmnet)
```

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

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

```
library(randomForest)
```

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(rpart)
library(rpart.plot)
library(tibble)
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
set.seed(123)

# ---------------------------- 1) Load & Initial Clean ----------------------------
data <- read.csv("fashiondb.csv", stringsAsFactors = FALSE)

# Drop leakage/derived/helper columns (keep only model-safe fields)
drop_cols <- c(
  "total_rows","avg_original_price","avg_current_price","kpi_avg_rating",
  "return_rate","sell_through_rate","price_mismatch_rate","price_flag",
  "expected_price","price_delta","md_fraction","current_price",
  "product_id"
)
data <- data %>% select(-any_of(drop_cols))

# Target to factor with ordered levels
data <- data %>%
  mutate(
    is_returned = case_when(
      is_returned %in% c(TRUE, 1, "TRUE", "1", "Yes") ~ "Yes",
      TRUE ~ "No"
    ),
    is_returned = factor(is_returned, levels = c("No","Yes"))
  )

# Dates
suppressWarnings({
  data$purchase_date <- as.Date(data$purchase_date, format = "%d-%m-%Y")
```

```r
})

# Categorical / numeric casts
fac_vars <- c("category","brand_clean","season","size_imputed","color")
num_vars <- c("original_price","markdown_percentage","stock_quantity","customer_rating")
data <- data %>%
  mutate(across(any_of(fac_vars), \(x) factor(x))) %>%
  mutate(across(any_of(num_vars), as.numeric))

# Handle missing values
for (nm in names(data)) {
  if (is.numeric(data[[nm]])) {
    if (anyNA(data[[nm]])) data[[nm]][is.na(data[[nm]])] <- median(data[[nm]], na.rm = TRUE)
  } else if (is.factor(data[[nm]])) {
    if (anyNA(data[[nm]])) data[[nm]] <- forcats::fct_explicit_na(data[[nm]], na_level = "Unknown")
  }
}

# Remove high-noise text column from modeling (kept in raw file if present)
if ("return_reason_clean" %in% names(data)) data$return_reason_clean <- NULL

# Drop near-zero variance predictors
nzv <- nearZeroVar(data %>% select(-is_returned), saveMetrics = TRUE)
if (any(nzv$nzv)) data <- data %>% select(-any_of(rownames(nzv)[nzv$nzv]))

# ------------------------------ 2) Train/Test Split ------------------------------
idx  <- createDataPartition(data$is_returned, p = 0.70, list = FALSE)
train <- data[ idx, , drop = FALSE]
test  <- data[-idx, , drop = FALSE]

train$is_returned <- factor(train$is_returned, levels = c("No","Yes"))
test$is_returned  <- factor(test$is_returned,  levels = c("No","Yes"))

y_train <- train$is_returned
y_test  <- test$is_returned

# Model matrices for glmnet/xgboost
x_train <- model.matrix(is_returned ~ ., data = train)[, -1, drop = FALSE]
x_test  <- model.matrix(is_returned ~ ., data = test)[,  -1, drop = FALSE]

# --------------------------------- 3) Utilities ---------------------------------
metrics_from_probs <- function(probs, truth, cutoff = 0.5, positive = "Yes") {
  pred <- factor(ifelse(probs >= cutoff, positive, setdiff(levels(truth), positive)),
                 levels = levels(truth))
  cm <- confusionMatrix(pred, truth, positive = positive)
  list(
    cm = cm,
    accuracy = as.numeric(cm$overall["Accuracy"]),
    sens = cm$byClass["Sensitivity"],
    spec = cm$byClass["Specificity"],
    bal_acc = mean(c(cm$byClass["Sensitivity"], cm$byClass["Specificity"]))
  )
}
```

```r
# ----------------------------- 4) Logistic (glmnet ridge) -------------------------
wts <- ifelse(y_train == "Yes", sum(y_train == "No")/sum(y_train == "Yes"), 1)

set.seed(123)
cv_glm <- cv.glmnet(
  x = x_train, y = y_train,
  family = "binomial", alpha = 0, type.measure = "auc",
  weights = wts, nfolds = 5, standardize = TRUE
)
best_lam <- cv_glm$lambda.min

prob_log_train <- as.numeric(predict(cv_glm, newx = x_train, s = best_lam, type = "response"))
prob_log_test  <- as.numeric(predict(cv_glm, newx = x_test,  s = best_lam, type = "response"))
stopifnot(length(prob_log_train) == nrow(train), length(prob_log_test) == nrow(test))

roc_log_train <- pROC::roc(y_train, prob_log_train, levels = c("No","Yes"), direction = "<")
log_cut <- as.numeric(pROC::coords(roc_log_train, "best", best.method = "youden",
                                   ret = "threshold", transpose = TRUE))
```

```
## Warning in coords.roc(roc_log_train, "best", best.method = "youden", ret =
## "threshold", : 'transpose=TRUE' is deprecated. Only 'transpose=FALSE' will be
## allowed in a future version.
```

```r
log_train_metrics <- metrics_from_probs(prob_log_train, y_train, cutoff = log_cut)
roc_log_test <- pROC::roc(y_test, prob_log_test, levels = c("No","Yes"), direction = "<")
log_test_metrics  <- metrics_from_probs(prob_log_test,  y_test,  cutoff = log_cut)

# ----------------------------- 5) Decision Tree ----------------------------------
set.seed(123)
tree_model <- rpart(
  is_returned ~ ., data = train, method = "class",
  control = rpart.control(cp = 0.005, minsplit = 10, maxdepth = 6)
)
tree_prob_train <- predict(tree_model, newdata = train, type = "prob")[,"Yes"]
tree_prob_test  <- predict(tree_model, newdata = test,  type = "prob")[,"Yes"]
roc_tree_train <- pROC::roc(y_train, tree_prob_train, levels = c("No","Yes"), direction = "<")
tree_cut <- as.numeric(coords(roc_tree_train, "best", best.method = "youden",
                              ret = "threshold", transpose = TRUE))
```

```
## Warning in coords.roc(roc_tree_train, "best", best.method = "youden", ret =
## "threshold", : 'transpose=TRUE' is deprecated. Only 'transpose=FALSE' will be
## allowed in a future version.
```

```r
tree_train_metrics <- metrics_from_probs(tree_prob_train, y_train, cutoff = tree_cut)
roc_tree_test <- pROC::roc(y_test, tree_prob_test, levels = c("No","Yes"), direction = "<")
tree_test_metrics  <- metrics_from_probs(tree_prob_test,  y_test,  cutoff = tree_cut)

# ----------------------------- 6) Random Forest ----------------------------------
ctrl <- trainControl(
  method = "cv", number = 3,
  classProbs = TRUE,
  summaryFunction = twoClassSummary,    # ROC/Sens/Spec
  sampling = "up",                      # balance inside CV
  savePredictions = "final"
```

```
)
set.seed(123)
rf_model <- train(
  is_returned ~ ., data = train,
  method = "rf", trControl = ctrl, metric = "ROC",
  ntree = 400, tuneLength = 5, importance = TRUE
)
rf_prob_train <- predict(rf_model, newdata = train, type = "prob")[,"Yes"]
rf_prob_test  <- predict(rf_model, newdata = test,  type = "prob")[,"Yes"]
roc_rf_train <- pROC::roc(y_train, rf_prob_train, levels = c("No","Yes"), direction = "<")
rf_cut <- as.numeric(coords(roc_rf_train, "best", best.method = "youden",
                            ret = "threshold", transpose = TRUE))
```

```
## Warning in coords.roc(roc_rf_train, "best", best.method = "youden", ret =
## "threshold", : 'transpose=TRUE' is deprecated. Only 'transpose=FALSE' will be
## allowed in a future version.
```

```
rf_train_metrics <- metrics_from_probs(rf_prob_train, y_train, cutoff = rf_cut)
roc_rf_test <- pROC::roc(y_test, rf_prob_test, levels = c("No","Yes"), direction = "<")
rf_test_metrics  <- metrics_from_probs(rf_prob_test,  y_test,  cutoff = rf_cut)

# ---------------------------- 7) XGBOOST (with imbalance) -----------------------
y_train_num <- as.integer(y_train == "Yes")
y_test_num  <- as.integer(y_test  == "Yes")

dtrain <- xgb.DMatrix(data = x_train, label = y_train_num)
dtest  <- xgb.DMatrix(data = x_test,  label = y_test_num)

pos <- sum(y_train_num == 1); neg <- sum(y_train_num == 0)
scale_pos_wt <- ifelse(pos == 0, 1, neg/pos)

params <- list(
  objective = "binary:logistic",
  eval_metric = "auc",
  eta = 0.08,
  max_depth = 4,
  min_child_weight = 5,
  subsample = 0.8,
  colsample_bytree = 0.8,
  gamma = 0,
  scale_pos_weight = scale_pos_wt,
  tree_method = "auto"
)

set.seed(123)
xgb_cv <- xgb.cv(
  params = params, data = dtrain, nrounds = 1000,
  nfold = 5, stratified = TRUE, early_stopping_rounds = 50, verbose = 0
)
best_rounds <- if (!is.null(xgb_cv$best_iteration)) xgb_cv$best_iteration else nrow(xgb_cv$evaluation_l

xgb_model <- xgb.train(params = params, data = dtrain, nrounds = best_rounds, verbose = 0)

xgb_prob_train <- predict(xgb_model, dtrain)
```

```r
xgb_prob_test  <- predict(xgb_model, dtest)

roc_xgb_train <- pROC::roc(y_train, xgb_prob_train, levels = c("No","Yes"), direction = "<")
xgb_cut <- as.numeric(coords(roc_xgb_train, "best", best.method = "youden",
                             ret = "threshold", transpose = TRUE))
```

```
## Warning in coords.roc(roc_xgb_train, "best", best.method = "youden", ret =
## "threshold", : 'transpose=TRUE' is deprecated. Only 'transpose=FALSE' will be
## allowed in a future version.
```

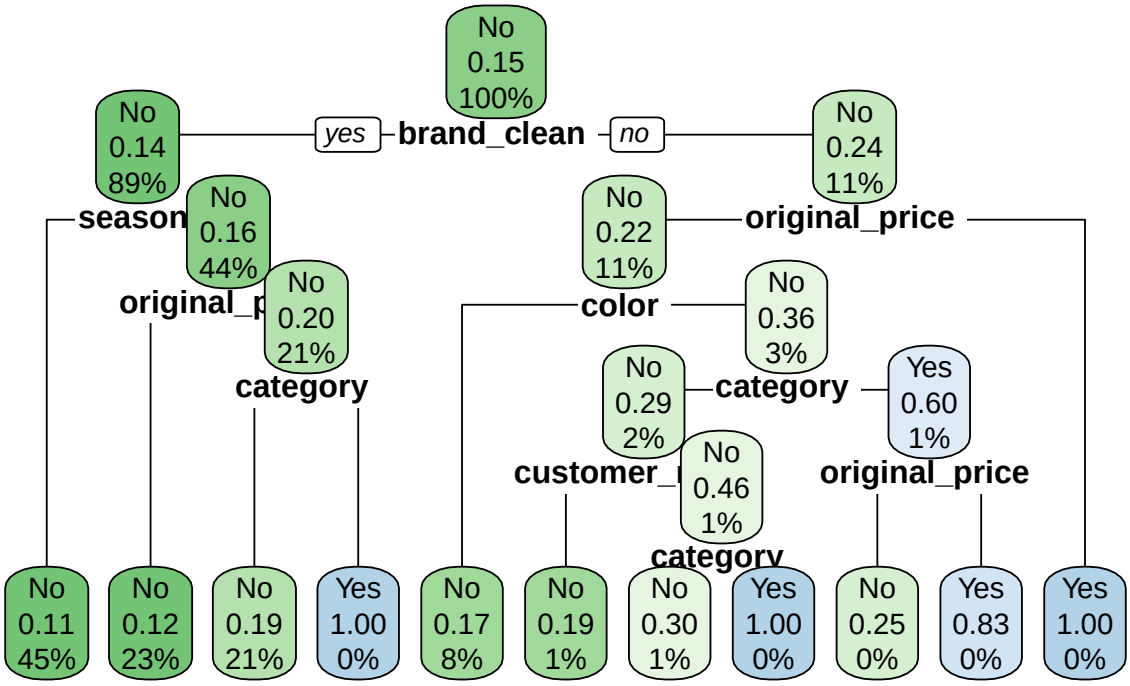
```r
xgb_train_metrics <- metrics_from_probs(xgb_prob_train, y_train, cutoff = xgb_cut)
roc_xgb_test <- pROC::roc(y_test, xgb_prob_test, levels = c("No","Yes"), direction = "<")
xgb_test_metrics  <- metrics_from_probs(xgb_prob_test,  y_test,  cutoff = xgb_cut)

cat("\n[XGBOOST]\n",
    "Best rounds:", best_rounds,
    "| Train AUC:", round(auc(roc_xgb_train),3),
    "| Test AUC:",  round(auc(roc_xgb_test),3),
    "| Test Acc:",  round(xgb_test_metrics$accuracy,3),
    "| Sens:",      round(xgb_test_metrics$sens,3),
    "| Spec:",      round(xgb_test_metrics$spec,3),
    "| Cutoff:",    round(xgb_cut,3), "\n")
```

```
##
## [XGBOOST]
##  Best rounds: 5 | Train AUC: 0.721 | Test AUC: 0.431 | Test Acc: 0.566 | Sens: 0.26 | Spec: 0.619 |
```

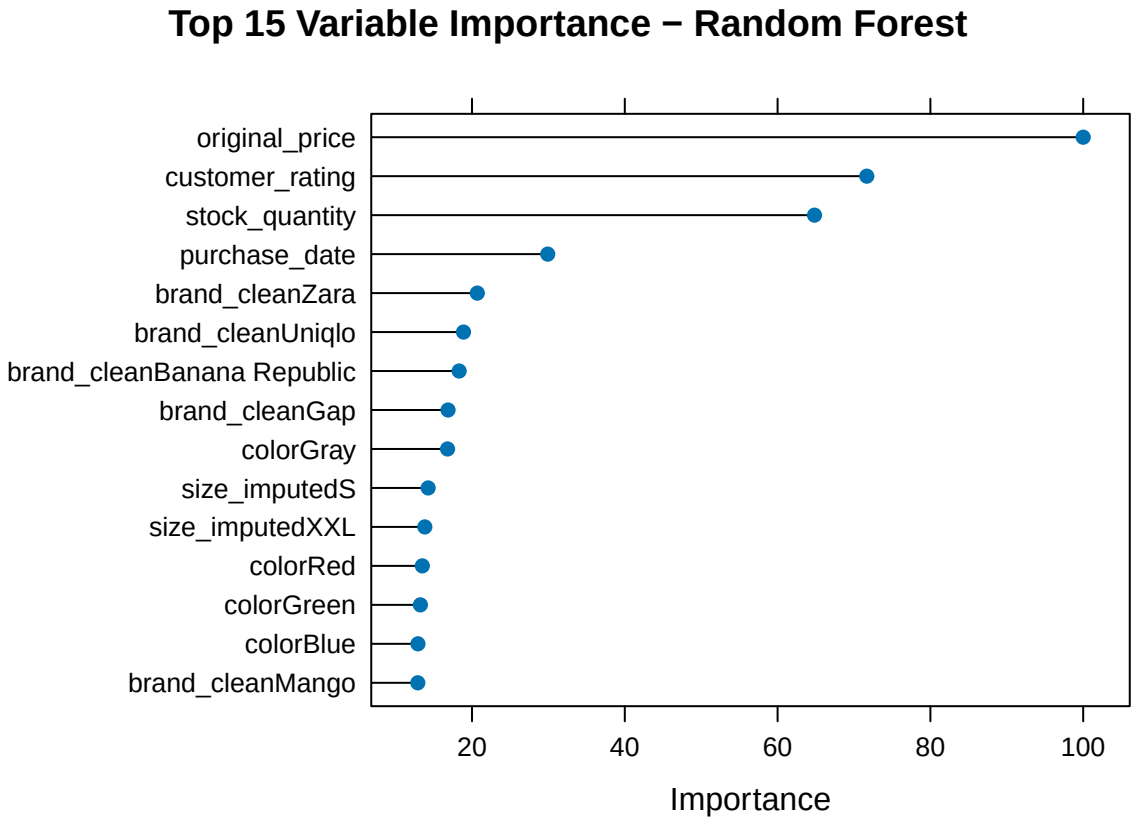
```r
# -------------------------- 8) VISUALIZATIONS --------------------------------

# Show only the variable name before any operator or list
split_var_only <- function(x, labs, digits, varlen, faclen) {
  gsub("^([^[<=>\n=,]+).*", "\\1", labs)
}

pruned_tree <- prune(tree_model, cp = 0.0059)
rpart.plot(
  pruned_tree,
  type = 2,                # split text above the node
  extra = 106,             # class prob + n (simple)
  box.palette = "GnBu",
  fallen.leaves = TRUE,
  uniform = TRUE,
  split.fun = split_var_only,   # <-- only variable names on splits
  split.cex = 1.1,         # split text size
  cex = 0.8, tweak = 1.15, # node text size / overall scale
  faclen = 0, varlen = 0,  # don't abbreviate names
  main = "Decision Tree for Product Return Prediction",
  )
```

```
## Warning: cex and tweak both specified, applying both
```

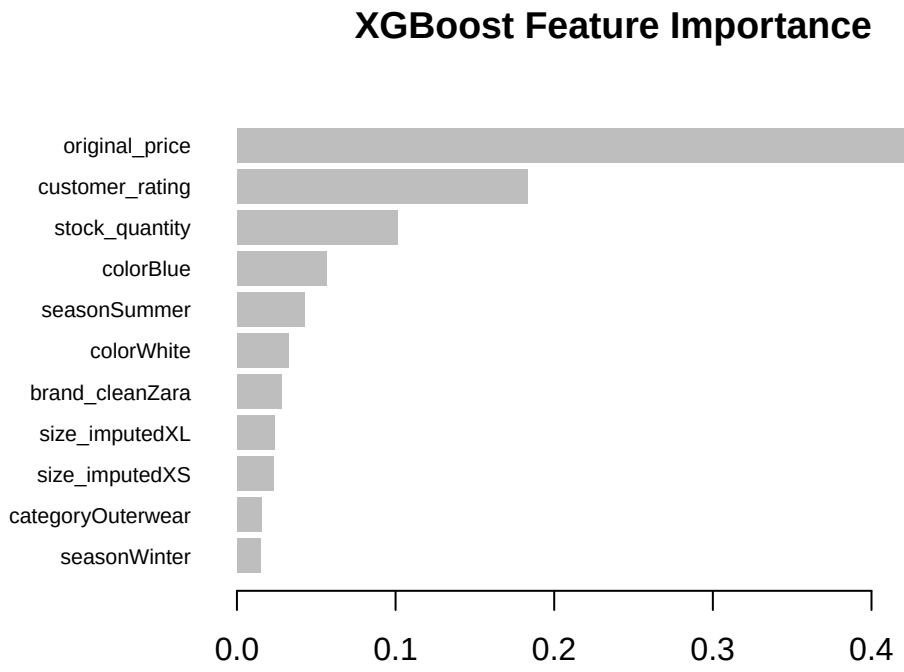
Decision Tree for Product Return Prediction



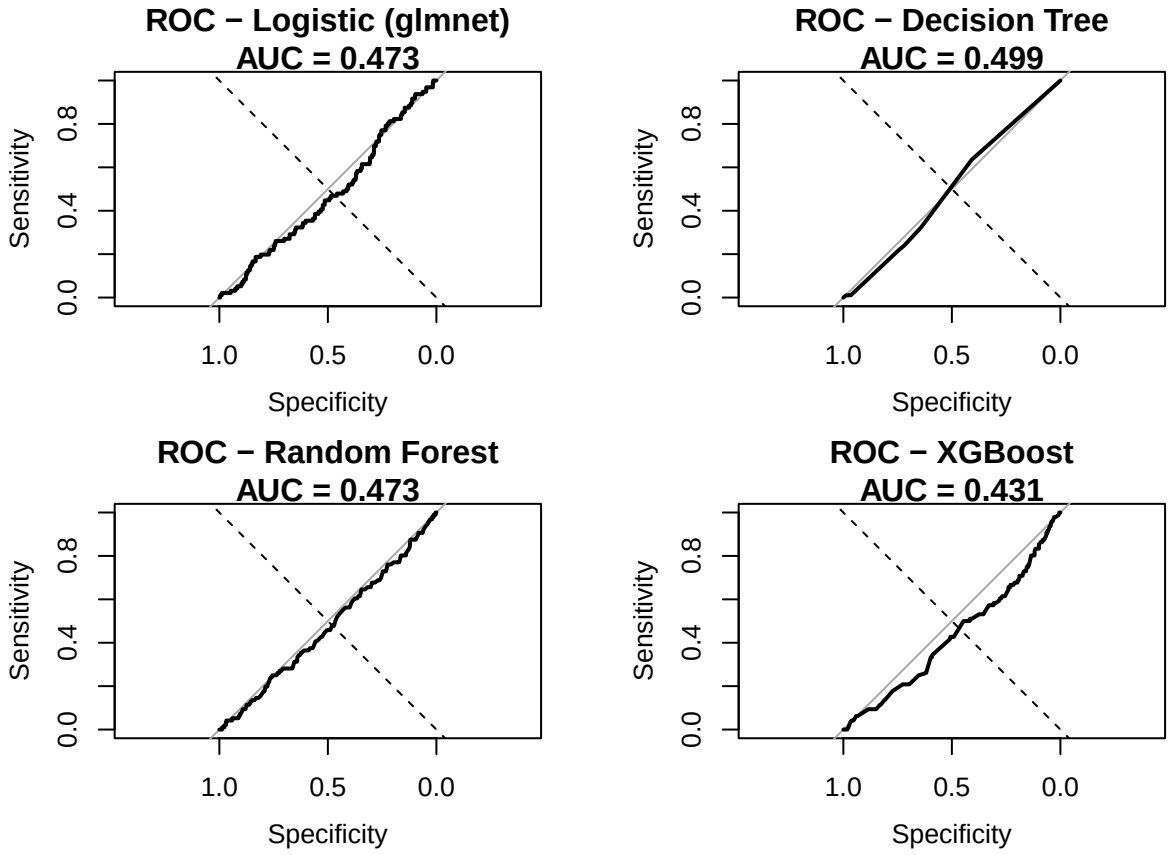

```r
# Random Forest Variable Importance
rf_vi <- varImp(rf_model)
par(mar = c(5, 12, 4, 2) + 0.5)  # more left space for labels
plot(rf_vi,
     top = 15,                          # <- change N if you want
     main = "Top 15 Variable Importance - Random Forest",
     cex = 0.95)                        # text size
```



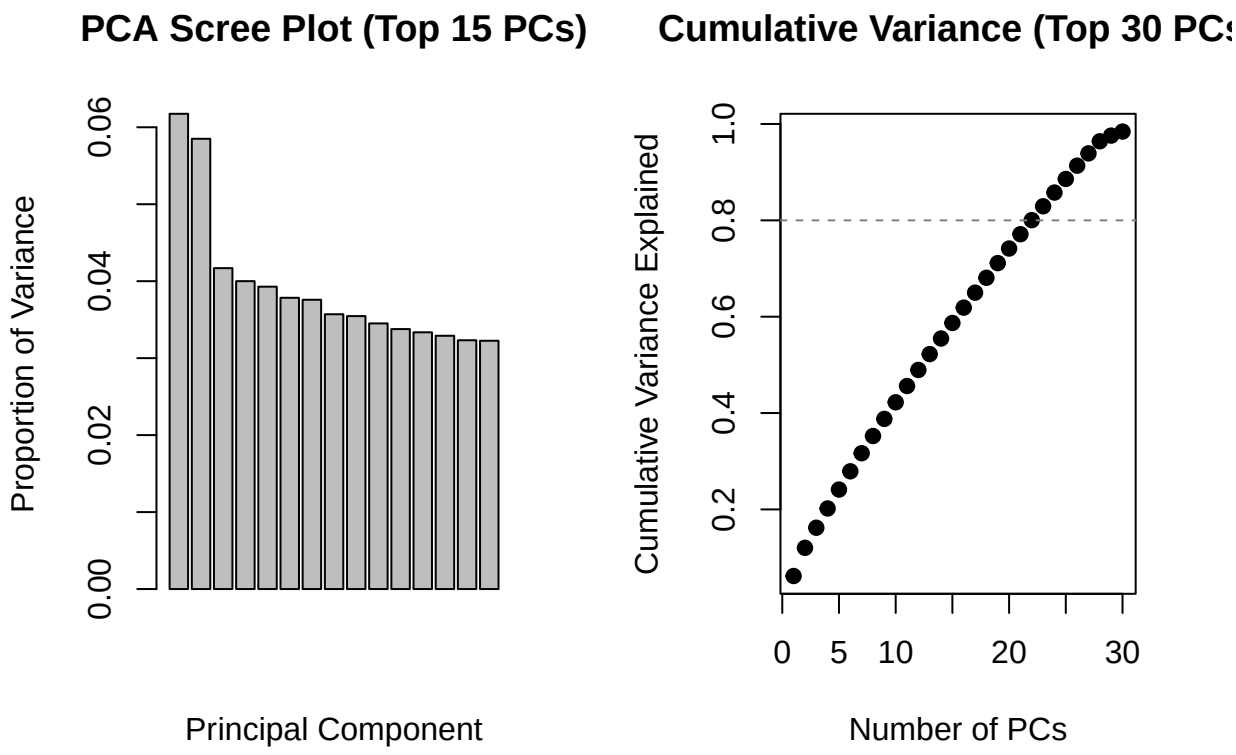

```r
par(mar = c(5, 4, 4, 2) + 0.5)

# XGBoost Feature Importance (top 15)
xgb_imp <- xgb.importance(model = xgb_model, feature_names = colnames(x_train))
xgb.plot.importance(xgb_imp[1:min(15, nrow(xgb_imp)), ], main = "XGBoost Feature Importance")
```



```r
# ROC Curves (Test) for all 4 models
par(mfrow = c(2, 2))
plot(roc_log_test,  main = paste0("ROC - Logistic (glmnet)\nAUC = ", round(auc(roc_log_test), 3))); abli
plot(roc_tree_test, main = paste0("ROC - Decision Tree\nAUC = ", round(auc(roc_tree_test), 3))); abline
plot(roc_rf_test,   main = paste0("ROC - Random Forest\nAUC = ", round(auc(roc_rf_test), 3)));   abline
plot(roc_xgb_test,  main = paste0("ROC - XGBoost\nAUC = ", round(auc(roc_xgb_test), 3)));        abline
```

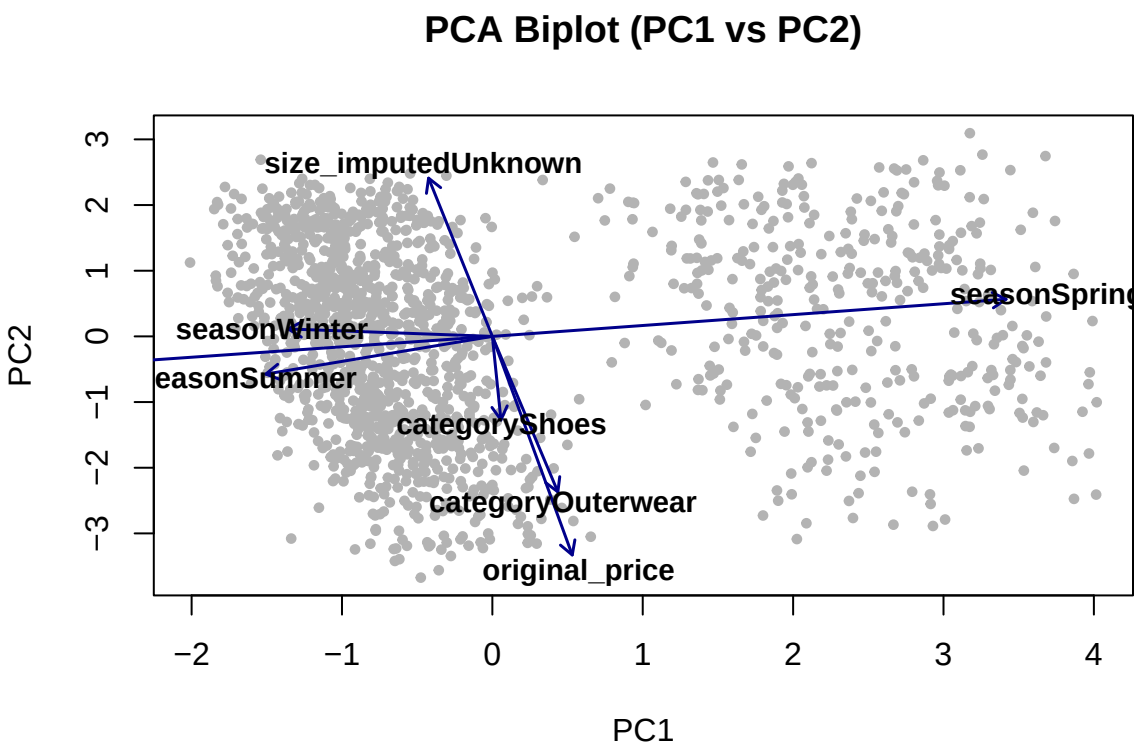



```r
par(mfrow = c(1, 1))

# ------------------------------ 9) PCA (Scree & Biplot) ------------------------
pca_fit <- prcomp(x_train, center = TRUE, scale. = TRUE)
pca_var  <- pca_fit$sdev^2
pca_prop <- pca_var / sum(pca_var)

par(mfrow = c(1, 2))
barplot(pca_prop[1:15],
        main = "PCA Scree Plot (Top 15 PCs)",
        xlab = "Principal Component", ylab = "Proportion of Variance")
plot(cumsum(pca_prop[1:30]), type = "b", pch = 19,
     xlab = "Number of PCs", ylab = "Cumulative Variance Explained",
     main = "Cumulative Variance (Top 30 PCs)")
abline(h = 0.80, lty = 2, col = "gray50")
```




```r
par(mfrow = c(1, 1))

# Simple biplot (PC1 vs PC2) with top loadings highlighted
# Plot individuals (scores)
# ---- PCA ----
pca_fit <- prcomp(x_train, center = TRUE, scale. = TRUE)

# 1. Loadings (rotation = contribution of variables to PCs)
rot <- pca_fit$rotation[, 1:2]

# 2. Pick top k variables for arrows
k <- 8   # change to however many arrows you want
ord <- order(rowSums(abs(rot)), decreasing = TRUE)[1:k]
L_top <- rownames(rot)[ord]

# 3. Scale factor so arrows fit inside point cloud
scale <- 0.85 * max(abs(pca_fit$x[,1:2])) / max(abs(rot))

# 4. Plot individuals (scores)
plot(pca_fit$x[,1], pca_fit$x[,2],
     col = "gray70", pch = 19, cex = 0.6,
     xlab = "PC1", ylab = "PC2",
     main = "PCA Biplot (PC1 vs PC2)")

# 5. Draw arrows
arrows(0, 0,
       rot[L_top,1]*scale,
       rot[L_top,2]*scale,
       col = "blue4", lwd = 1.5, length = 0.08)
```

```
# 6. Add labels
text(rot[L_top,1]*scale*1.08,
     rot[L_top,2]*scale*1.08,
     labels = L_top, col = "black", cex = 0.9, font = 2)
```

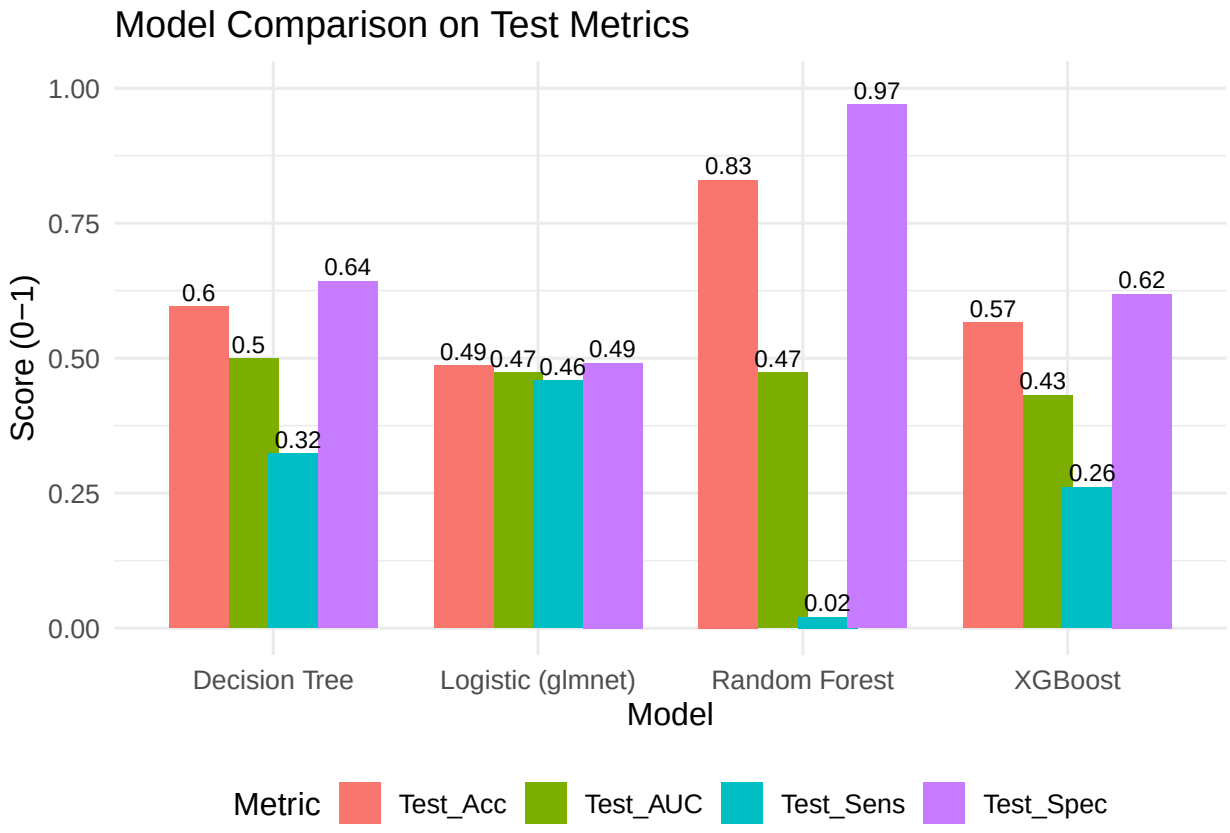



```
# ---------------------------- 10) Model Comparison (table) -----------------------
summary_df <- tibble(
  Model     = c("Logistic (glmnet)", "Decision Tree", "Random Forest", "XGBoost"),
  Test_AUC  = c(as.numeric(auc(roc_log_test)),
                as.numeric(auc(roc_tree_test)),
                as.numeric(auc(roc_rf_test)),
                as.numeric(auc(roc_xgb_test))),
  Test_Acc  = c(log_test_metrics$accuracy,
                tree_test_metrics$accuracy,
                rf_test_metrics$accuracy,
                xgb_test_metrics$accuracy),
  Test_Sens = c(log_test_metrics$sens,
                tree_test_metrics$sens,
                rf_test_metrics$sens,
                xgb_test_metrics$sens),
  Test_Spec = c(log_test_metrics$spec,
                tree_test_metrics$spec,
                rf_test_metrics$spec,
                xgb_test_metrics$spec)
)
print(summary_df)
```

```
## # A tibble: 4 x 5
##   Model             Test_AUC Test_Acc Test_Sens Test_Spec
##   <chr>                <dbl>    <dbl>     <dbl>     <dbl>
## 1 Logistic (glmnet)    0.473    0.486     0.458     0.491
```

```
## 2 Decision Tree        0.499    0.595    0.323      0.642
## 3 Random Forest        0.473    0.830    0.0208     0.969
## 4 XGBoost              0.431    0.566    0.260      0.619
```

```r
# ---------------------------- 11) Model Comparison (chart) -----------------------
# Melt-like reshape for ggplot
summary_long <- summary_df |>
  tidyr::pivot_longer(cols = -Model, names_to = "Metric", values_to = "Value")

ggplot(summary_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_col(position = position_dodge(width = 0.75)) +
  geom_text(aes(label = round(Value, 2)),
            position = position_dodge(width = 0.75), vjust = -0.35, size = 3) +
  ylim(0, 1) +
  labs(title = "Model Comparison on Test Metrics",
       x = "Model", y = "Score (0-1)") +
  theme_minimal(base_size = 12) +
  theme(legend.position = "bottom")
```




```r
# ================================================================================
```

```r
# ================== K-Means Clustering + Silhouette ========================


install.packages(c("factoextra"))
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.5'
## (as 'lib' is unspecified)
```

```r
library(dplyr)
library(cluster)
library(factoextra)
```

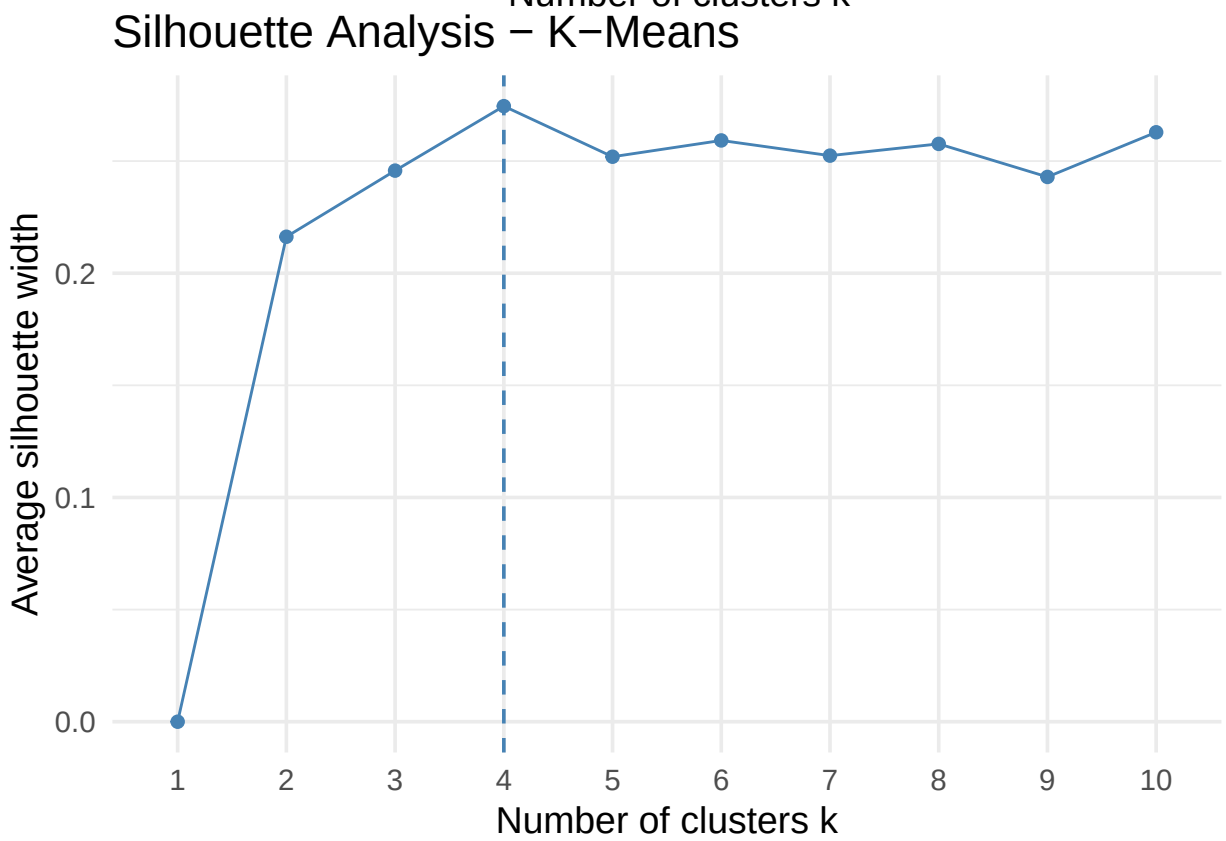
```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(ggplot2)

set.seed(123)


# --- 1) Pick numeric features robustly ---
num_data <- data[, sapply(data, is.numeric), drop = FALSE]

# Drop columns that are all NA or zero-variance
keep_ok <- sapply(num_data, function(x) !all(is.na(x)) && sd(x, na.rm = TRUE) > 0)
num_data <- num_data[, keep_ok, drop = FALSE]
stopifnot(ncol(num_data) >= 2)  # need at least 2 numeric columns

# Median-impute any remaining NAs
for (nm in names(num_data)) {
  if (anyNA(num_data[[nm]])) {
    num_data[[nm]][is.na(num_data[[nm]])] <- median(num_data[[nm]], na.rm = TRUE)
  }
}

# --- 2) Scale features ---
num_scaled <- scale(num_data)

# --- 3) Choose k by silhouette (k = 2..8) ---
k_range <- 2:8
sil_vals <- sapply(k_range, function(k) {
  km <- kmeans(num_scaled, centers = k, nstart = 25)
  ss <- silhouette(km$cluster, dist(num_scaled))
  mean(ss[, 3])  # average silhouette width
})
k_best <- k_range[which.max(sil_vals)]
cat(sprintf("Chosen k by silhouette: %d (avg silhouette = %.3f)\n", k_best, max(sil_vals)))
```

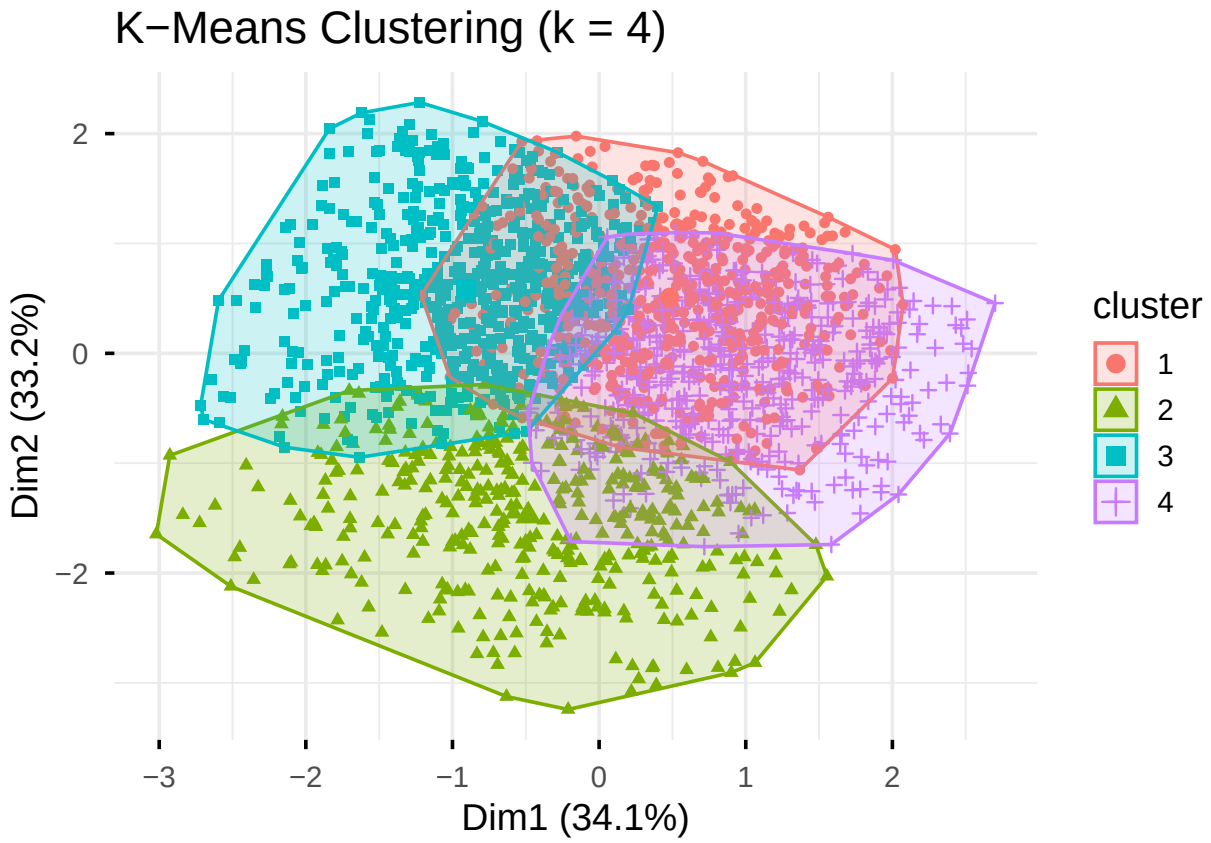
```
## Chosen k by silhouette: 4 (avg silhouette = 0.275)
```

```r
# Visuals
p_elbow <- fviz_nbclust(num_scaled, kmeans, method = "wss") +
  ggtitle("Elbow Method (WSS) - K-Means") + theme_minimal(base_size = 14)
p_sil   <- fviz_nbclust(num_scaled, kmeans, method = "silhouette") +
  ggtitle("Silhouette Analysis - K-Means") + theme_minimal(base_size = 14)
print(p_elbow); print(p_sil)
```


Elbow Method (WSS) − K−Means
Total Within Sum of Square
6000
5000
4000
3000
2000
1
2
3
4
5
6
7
8
9
10
Number of clusters k


Silhouette Analysis − K−Means
Average silhouette width
0.2
0.1
0.0
1
2
3
4
5
6
7
8
9
10
Number of clusters k

```r
# --- 4) Fit final K-Means ---
set.seed(123)
km <- kmeans(num_scaled, centers = k_best, nstart = 50)

# --- 5) Visualize clusters (PCA projection) ---
p_clusters <- fviz_cluster(
  km, data = num_scaled,
  geom = "point", ellipse.type = "convex",
  ggtheme = theme_minimal(base_size = 14)
) + ggtitle(sprintf("K-Means Clustering (k = %d)", k_best))
print(p_clusters)
```




```r
# --- 6) Attach clusters back to main data ---
data$cluster_km <- factor(km$cluster)

# --- 7) Quick cluster profile (counts, means, + return rate if available) ---
profile_numeric <- num_data %>%
  mutate(cluster_km = data$cluster_km) %>%
  group_by(cluster_km) %>%
  summarise(across(everything(), ~mean(.x, na.rm = TRUE), .names = "avg_{.col}"),
            .groups = "drop")

if ("is_returned" %in% names(data)) {
  profile_returns <- data %>%
    group_by(cluster_km) %>%
    summarise(n = n(),
              return_rate = mean(is_returned == "Yes", na.rm = TRUE),
```

```
              .groups = "drop")
  cluster_profile <- left_join(profile_returns, profile_numeric, by = "cluster_km")
} else {
  cluster_profile <- profile_numeric %>%
    mutate(n = as.integer(table(data$cluster_km))) %>%
    relocate(n, .after = cluster_km)
}

cat("\n--- Cluster Profile ---\n")
```

```
##
## --- Cluster Profile ---
```

```
print(cluster_profile, n = Inf)
```

```
## # A tibble: 4 x 6
##   cluster_km     n return_rate avg_original_price avg_stock_quantity
##   <fct>      <int>       <dbl>              <dbl>              <dbl>
## 1 1            626       0.171               74.3               38.2
## 2 2            412       0.136              179.                26.5
## 3 3            624       0.133               84.7               11.1
## 4 4            514       0.144               74.7               24.4
## # i 1 more variable: avg_customer_rating <dbl>
```

```
# centers in original scale (use medians of original numeric columns by cluster)
cluster_centers <- num_data %>%
  mutate(cluster_km = data$cluster_km) %>%
  group_by(cluster_km) %>%
  summarise(across(everything(), median, na.rm = TRUE), .groups = "drop")
```

```
## Warning: There was 1 warning in `summarise()`.
## i In argument: `across(everything(), median, na.rm = TRUE)`.
## i In group 1: `cluster_km = 1`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

```
cat("\n--- Cluster Centers (medians, original scale) ---\n")
```

```
##
## --- Cluster Centers (medians, original scale) ---
```

```
print(cluster_centers, n = Inf)
```

```
## # A tibble: 4 x 4
##   cluster_km original_price stock_quantity customer_rating
##   <fct>               <dbl>          <dbl>           <dbl>
## 1 1                    71.8             38            3.4
## 2 2                   173.              27            2.90
## 3 3                    81.2             11            3.4
## 4 4                    73.8             24            1.6
```