🧭 0) Title & Executive Summary

Project Title: *Wholesale Customers: Unsupervised Segmentation & DR*

Executive Summary (3–5 sentences):

- *What you did (clustering and/or DR) and why.*
- *One or two headline insights (e.g., distinct high-detergent spenders vs fresh-heavy buyers).*
- *Immediate stakeholder impact (targeting, inventory, pricing, ops).*

🎯 Objective & Business Value (Clustering / Dimensionality Reduction)

- **Objective:** *Clustering and/or Dimensionality Reduction (pick one or both).*
- **Business Value:** *How this supports marketing, supply chain, risk, ops, ROI.*

💼 What stakeholders gain (actions, ROI, risk, ops)

- *Concrete actions (who to target, with what).*
- *Expected impact (qualitative or rough quantitative).*
- *Operational implications / risks.*

🗂️ 1) Dataset & Access

🌐 Source & License (UCI Wholesale Customers)

- **Source:** UCI ML Repository — Wholesale customers (Cardoso, 2013)
- **License:** CC BY 4.0
- **Notes:** *Any data filters or sampling applied.*

📁 File Path: `/content/W_cust_data.csv`

- **Colab path:** `/content/W_cust_data.csv`
- **Version/Date Pulled:** *YYYY-MM-DD*

🔎 Shape, date, unit definitions, assumptions

- **Rows × Columns:** *e.g., 440 × 8*
- **Units:** Monetary units (annual spend)
- **Assumptions/Scope:** *e.g., single year, EU regions, retail vs HORECA.*

🧱 2) Data Schema & Understanding

📜 Variables overview (numeric vs categorical)

- **Numeric:** Fresh, Milk, Grocery, Frozen, Detergents_Paper, Delicatessen
- **Categorical (for reference/visuals):** Channel (Horeca/Retail), Region (Lisbon/Oporto/Other)

🎯 Target/labels (for reference only; unsupervised focus)

- *No target used for training; Channel/Region only for sanity-check visualizations.*

🔍 3) EDA Overview (Brief, Insight-Focused)

🚧 Missing values, ranges, skew/outliers

- *No missing values reported.*
- *Strong right-skew across spend features; presence of outliers.*

📊 Distribution highlights (per feature)

- *1–2 bullets on notable distributions (e.g., Detergents_Paper highly zero-inflated or long-tailed).*
- **Figure:** `figures/histograms_raw.png`

🔗 Correlations/collinearity notes

- *E.g., Milk ↔ Grocery ↔ Detergents_Paper correlated (household goods bundle).*

🧹 4) Data Cleaning & Feature Preparation

🧪 Transformations (log1p), scaling (StandardScaler)

- **Transform:** `log1p` on numeric to reduce skew.
- **Scale:** Standardize to zero-mean/unit-variance.

🎨 Encoding/retention of Channel/Region (for coloring only)

- *Kept as categories for visualization; not used as features.*

✅ Final feature set used for modeling

- *List numeric features used; justify exclusions, if any.*

🧭 5) Dimensionality Reduction (DR)

🧱 PCA: Explained Variance & Scree Plot

- *Variance captured by first N PCs; knee point visible.*
- **Figure:** `figures/pca_scree.png`

🧭 PCA: 2D scatter (no clusters) + interpretation of PC1/PC2

- *PC1 ~ "household goods basket" vs PC2 ~ "fresh/frozen intensity" (example).*
- **Figure:** `figures/pca_scatter_raw.png`

🌀 UMAP / t-SNE: 2D scatter (no clusters) for structure intuition

- *Neighborhood structure indicates latent groupings.*
- **Figures:** `figures/umap_scatter_raw.png`, `figures/tsne_scatter_raw.png`

🧪 6) Clustering Experiments (≥3 variants)

6.1 🔵 K-Means: k sweep (k=2..10) – Elbow & Silhouette

- *Method & metric summary.*
- **Figures:** `figures/kmeans_elbow.png`, `figures/kmeans_silhouette.png`

6.2 🧱 DBSCAN: grid over eps, min_samples – cluster/noise behavior

- *Parameter grid, #clusters, noise fraction, best config.*
- **Figure (if good):** `figures/umap_clusters_dbscan.png`

6.3 🌳 Agglomerative: linkage variants (ward/complete/average)

- *Top linkage+k by silhouette/CH/DB.*
- **Figure:** `figures/pca_clusters_agglom.png`

🏁 7) Model Selection Evidence & Recommendation

📋 Metrics table (Silhouette, Davies–Bouldin, Calinski–Harabasz)

- *Insert best rows summary (K-Means & competitors).*
- **Table:** *paste from output or attach CSV snippet.*

✅ Final pick rationale (metrics + interpretability + usefulness)

- **Recommended Model:** *e.g., K-Means (k=4).*
- **Why:** *Peak silhouette + stable elbow + clean business persona separation.*

🗺️ 8) Cluster Visualizations

🧭 PCA 2D colored by chosen clusters

- **Figure:** `figures/pca_clusters_kmeans.png`

🌀 UMAP 2D colored by chosen clusters

- **Figure:** `figures/umap_clusters_kmeans.png`

🧩 t-SNE 2D colored by chosen clusters

- **Figure:** `figures/tsne_clusters_kmeans.png`

🖊️ (Optional) Color by Channel/Region for external validation

- **Figures:** `figures/pca_by_channel.png`, `figures/umap_by_region.png`
- *Narrate alignment/misalignment vs business reality.*

🧬 9) Cluster Profiles (Original Units)

📑 Per-cluster means/medians table

- **Tables:** `figures/cluster_profiles_mean.csv`, `figures/cluster_profiles_median.csv`
- *Highlight top 2–3 features per cluster.*

👥 Persona-style summary (who they are, how to act)

- **Cluster 0:** *e.g., Detergent/Grocery-heavy retailers → cross-sell paper bundles.*
- **Cluster 1:** *Fresh/Frozen-heavy HORECA → optimize cold chain & seasonal promos.*
- **Cluster 2:** *…*
- **Cluster 3:** *…*

💡 10) Key Findings & Business Actions

⭐ Top 3–5 insights tied to decisions

- *Actionable bullets linked to segments (offers, routes, credit limits, etc.).*

⚠️ Risks, caveats, and monitoring ideas

- *E.g., seasonality, price shocks, outlier impact, retrain cadence.*

⏭️ 11) Next Steps

🧱 Data/features to add, tests to run

- *E.g., add order frequency, promo response, geography granularity.*

🔁 Revisit segmentation cadence, A/B experiments

- *Define refresh cycle, pilot tests, success metrics.*

🧩 12) (Optional) Cluster-then-Supervise Extension

🆚 One global model vs per-cluster models (metric comparison)

- *Accuracy/F1 or RMSE/MAE side-by-side; does segmentation help?*

🧠 Does segmentation improve performance/interpretability?

- *Short conclusion + any trade-offs.*

🖼️ 13) Figure Index (For PDF)

📚 List of saved plots with filenames and captions

- `figures/histograms_raw.png` — Raw distributions
- `figures/pca_scree.png` — PCA explained variance (scree)

- `figures/pca_scatter_raw.png` — PCA 2D (no clusters)
- `figures/umap_scatter_raw.png` — UMAP 2D (no clusters)
- `figures/tsne_scatter_raw.png` — t-SNE 2D (no clusters)
- `figures/kmeans_elbow.png` — Elbow plot (inertia vs k)
- `figures/kmeans_silhouette.png` — Silhouette vs k
- `figures/pca_clusters_kmeans.png` — PCA 2D colored by K-Means
- `figures/umap_clusters_kmeans.png` — UMAP 2D colored by K-Means
- `figures/tsne_clusters_kmeans.png` — t-SNE 2D colored by K-Means
- `figures/umap_clusters_dbscan.png` — UMAP 2D colored by DBSCAN (if used)
- `figures/pca_clusters_agglom.png` — PCA 2D colored by Agglomerative
- `figures/pca_by_channel.png` — PCA 2D colored by Channel
- `figures/umap_by_region.png` — UMAP 2D colored by Region

📍 Where they're used in the report

- *Map figures to sections for easy reference.*

🔁 14) Reproducibility & Appendix

🧩 Environment, versions, random seeds

- Python, scikit-learn, umap-learn, matplotlib versions
- `RANDOM_STATE = 42` (set globally)

📚 Data dictionary (units, notes)

- **Fresh/Milk/Grocery/Frozen/Detergents_Paper/Delicatessen:** annual spend (m.u.)
- **Channel:** Horeca/Retail
- **Region:** Lisbon/Oporto/Other

📖 References

- Cardoso, M. (2013). *Wholesale customers*. UCI ML Repository. DOI: 10.24432/C5030X
- *Any additional sources.*

0) Setup

We install minimal libs, import dependencies, and create a [`/content/figures`](/content/figures) folder to save all plots for the PDF report.

```
# 0) Setup
!pip -q install umap-learn

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
```

```
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
from sklearn.manifold import TSNE

import umap

# Globals
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

# Figures dir
FIG_DIR = Path("/content/figures")
FIG_DIR.mkdir(parents=True, exist_ok=True)

def savefig(name):
    path = FIG_DIR / name
    plt.savefig(path, dpi=200, bbox_inches='tight')
    print(f"[saved] {path}")
```

1) Dataset & Access

Goal: load the UCI Wholesale Customers dataset from Colab path `/content/W_cust_data.csv`, sanity-check shape/dtypes, and preview rows.

```
# 1) Load & Basic Info
CSV_PATH = "/content/W_cust_data.csv"

df = pd.read_csv(CSV_PATH)
print("Shape:", df.shape)
display(df.head())

# Clean up column names (remove spaces, unify naming)
df.columns = [c.strip().replace(" ", "_") for c in df.columns]
if "Delicassen" in df.columns:
    df = df.rename(columns={"Delicassen": "Delicatessen"})

# Cast categoricals if present
for cat in ["Channel", "Region"]:
    if cat in df.columns:
        df[cat] = df[cat].astype("category")

print("\nDtypes:")
print(df.dtypes)

print("\nMissing values per column:")
print(df.isna().sum())
```

```
Shape: (440, 8)
```

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

```
Dtypes:
Channel             category
Region              category
Fresh                  int64
Milk                   int64
Grocery                int64
Frozen                 int64
Detergents_Paper       int64
Delicatessen           int64
dtype: object

Missing values per column:
Channel             0
Region              0
Fresh               0
Milk                0
Grocery             0
Frozen              0
Detergents_Paper    0
Delicatessen        0
dtype: int64
```

✅ Section 1 — Conclusion: Dataset & Access

- **Loaded from:** `/content/W_cust_data.csv`
- **Shape: 440 rows × 8 columns**
- **Dtypes:** `Channel`, `Region` → categorical; 6 spend fields → integers
- **Missing values: None** detected across all columns
- **Column cleanup:** spaces → underscores; `Delicassen` → `Delicatessen`
- **Modeling plan:** use **numeric spend features** for unsupervised modeling; keep **Channel/Region** only for **sanity-check visuals**
- **Units & scope:** annual spend in monetary units (m.u.); assumed single-year snapshot
- **Source & license:** UCI ML Repository (Cardoso, 2013), **CC BY 4.0**
- **Next up:** quick **EDA** to confirm skew/outliers and correlations before applying `log1p` + scaling

2) Data Schema & Understanding

Goal: lock down which columns power the unsupervised modeling vs what we'll keep only for sanity-check visuals.

- **Modeling features (numeric):** `Fresh, Milk, Grocery, Frozen, Detergents_Paper, Delicatessen`
- **Reference-only categoricals:** `Channel (1=Horeca, 2=Retail)`, `Region (1=Lisbon, 2=Oporto, 3=Other)`
- Create **readable labels** (`Channel_name`, `Region_name`) so later plots have clean legends.
- Build a compact **schema table** (role, dtype, used_in_model, uniques/min/max) for the appendix.

```python
# -- Section 2: Data Schema & Understanding --

# 1) Declare numeric features for modeling & categoricals for visuals
numeric_cols_default = ['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delicatessen']
numeric_cols = [c for c in numeric_cols_default if c in df.columns]
cat_cols = [c for c in ['Channel','Region'] if c in df.columns]

print("Numeric columns used for modeling:", numeric_cols)
print("Categorical columns kept for visuals only:", cat_cols)

# 2) Map integer codes -> human-friendly labels (new columns for plotting)
channel_map = {1: "Horeca", 2: "Retail"}
region_map  = {1: "Lisbon", 2: "Oporto", 3: "Other"}

if 'Channel' in df.columns:
    df['Channel_name'] = df['Channel'].astype(int).map(channel_map).astype('category')
if 'Region' in df.columns:
    df['Region_name'] = df['Region'].astype(int).map(region_map).astype('category')

# 3) Quick frequency checks for categoricals
for c in ['Channel_name','Region_name']:
    if c in df.columns:
        print(f"\nValue counts for {c}:")
        print(df[c].value_counts())

# 4) Build a tidy schema table for the appendix/report
schema_rows = []
for col in df.columns:
    dtype = str(df[col].dtype)
    used_in_model = col in numeric_cols  # only numeric spend features
    role = (
        "feature_numeric" if col in numeric_cols else
        "feature_categorical" if col in ['Channel','Region'] else
        "derived_label" if col in ['Channel_name','Region_name'] else
        "other"
    )
    entry = {
        "variable": col,
        "dtype": dtype,
        "role": role,
        "used_in_model": used_in_model
    }
    if pd.api.types.is_numeric_dtype(df[col]):
        entry["min"] = float(df[col].min())
        entry["max"] = float(df[col].max())
        entry["unique"] = int(df[col].nunique())
    else:
        entry["min"] = None
        entry["max"] = None
        entry["unique"] = int(df[col].nunique())
    schema_rows.append(entry)

schema_df = pd.DataFrame(schema_rows).sort_values(["used_in_model","role","variable"], ascending=[False, True, True])
display(schema_df)

# Save for appendix
schema_path = FIG_DIR / "schema_table.csv"
schema_df.to_csv(schema_path, index=False)
print(f"[saved] {schema_path}")
```

```
Numeric columns used for modeling: ['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents_Paper', 'Delicatessen']
Categorical columns kept for visuals only: ['Channel', 'Region']

Value counts for Channel_name:
Channel_name
Horeca    298
Retail    142
Name: count, dtype: int64

Value counts for Region_name:
Region_name
Other     316
Lisbon     77
Oporto     47
Name: count, dtype: int64
```

	variable	dtype	role	used_in_model	min	max	unique
7	Delicatessen	int64	feature_numeric	True	3.0	47943.0	403
6	Detergents_Paper	int64	feature_numeric	True	3.0	40827.0	417
2	Fresh	int64	feature_numeric	True	3.0	112151.0	433
5	Frozen	int64	feature_numeric	True	25.0	60869.0	426
4	Grocery	int64	feature_numeric	True	3.0	92780.0	430
3	Milk	int64	feature_numeric	True	55.0	73498.0	421
8	Channel_name	category	derived_label	False	NaN	NaN	2
9	Region_name	category	derived_label	False	NaN	NaN	3
0	Channel	category	feature_categorical	False	NaN	NaN	2
1	Region	category	feature_categorical	False	NaN	NaN	3

```
[saved] /content/figures/schema_table.csv
```

Next steps: Generate code with `schema_df` View recommended plots New interactive sheet

✅ Section 2 — Conclusion: Data Schema & Understanding

- **Modeling features (numeric, 6):** Fresh, Milk, Grocery, Frozen, Detergents_Paper, Delicatessen
  - Wide value ranges (e.g., **Fresh: 3 → 112,151; Grocery: 3 → 92,780**), strong hint of **outliers** and **right-skew** → will **log1p + standardize**.
  - High uniqueness (400+ for most) confirms **continuous spend** behavior.
- **Reference-only categoricals:** Channel, Region (kept for validation/plots, **not** used for training)
  - **Channel:** Horeca **298/440 ≈ 67.7%**, Retail **142/440 ≈ 32.3%** → class imbalance to remember when interpreting clusters.
  - **Region:** Other **316/440 ≈ 71.8%**, Lisbon **77/440 ≈ 17.5%**, Oporto **47/440 ≈ 10.7%** → heavy tilt toward "Other".
  - Clean labels created: **Channel_name** (Horeca/Retail), **Region_name** (Lisbon/Oporto/Other) for readable legends.
- **Schema table saved:** `/content/figures/schema_table.csv` (role, dtype, used_in_model, uniques/min/max) for appendix and reproducibility.
- **Implications for modeling:**
  - Expect a PCA axis capturing the **household goods bundle** (Milk–Grocery–Detergents_Paper).
  - **Fresh/Frozen** may define a separate dimension (Horeca-leaning behavior).
  - We'll **validate** cluster meaning by coloring PCA/UMAP with Channel/Region **after** clustering (to avoid leakage).

- **Next:** Run **EDA** to quantify skew/correlations and then apply **log1p + StandardScaler** before DR and clustering.

3) EDA Overview

Goal: sanity-check the data before modeling.

- **Missing/Range:** reconfirm no nulls; inspect ranges and summary stats.
- **Skew/Outliers:** quantify skewness; visualize raw histograms and boxplots.
- **Correlation/Collinearity:** understand which spends move together to anticipate PCA axes and clustering behavior.

What to look for

- Strong right-skew in spends → motivates `log1p`.
- Outliers (very large annual spends) → distance-based methods need scaling.
- Tight positive correlation among **Milk–Grocery–Detergents_Paper** (household goods), and a different pattern for **Fresh/Frozen** (HORECA-leaning).

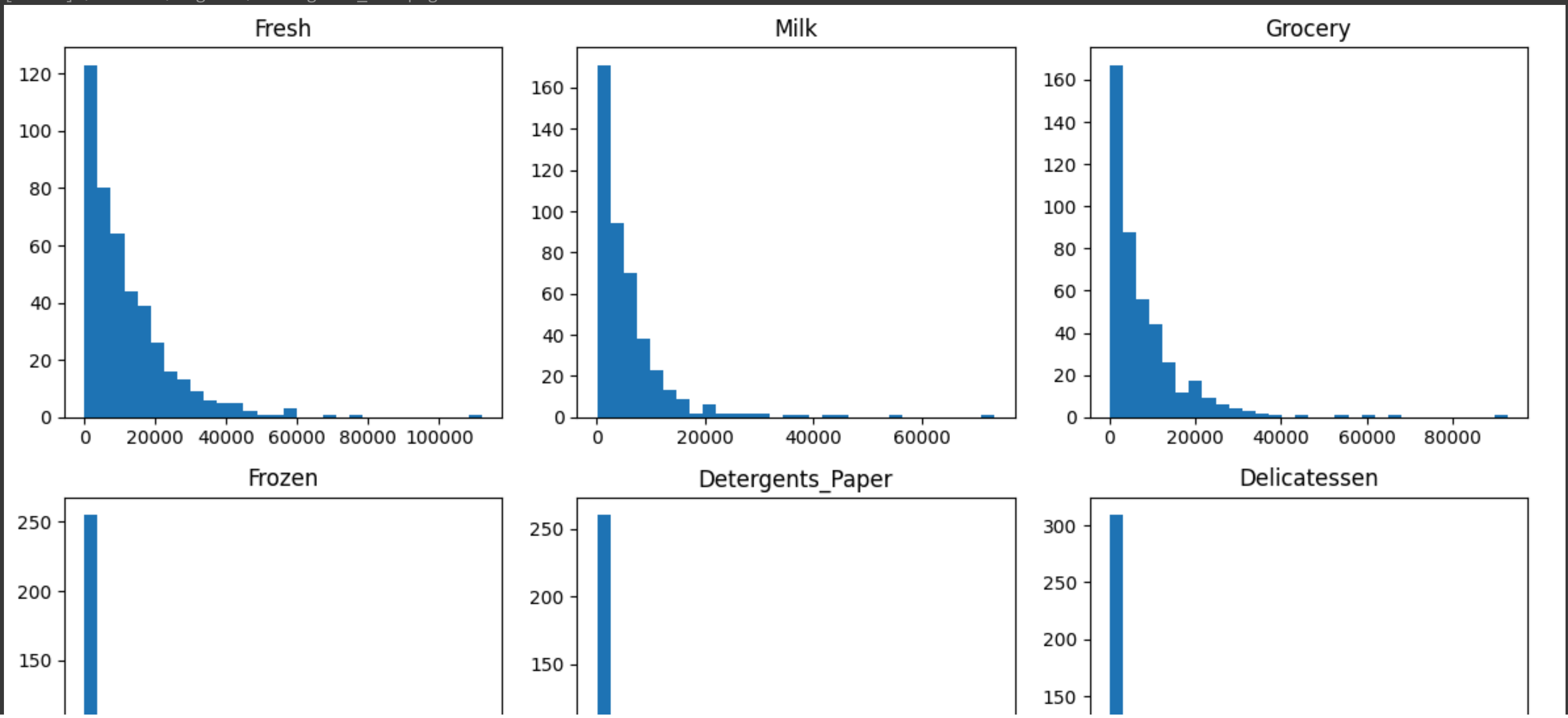
```
# --- Section 3: EDA ---

# 3.1 Summary stats & missingness (reconfirm)
print("Missing values per column:\n", df.isna().sum(), "\n")
display(df[numeric_cols].describe().T)

# 3.2 Skewness (right-skew expected)
skews = df[numeric_cols].skew(numeric_only=True).sort_values(ascending=False)
print("\nSkewness (descending):")
print(skews)

# 3.3 Raw histograms (distribution shape)
fig, axes = plt.subplots(2, 3, figsize=(12, 7))
axes = axes.ravel()
for i, col in enumerate(numeric_cols):
    axes[i].hist(df[col].values, bins=30)
    axes[i].set_title(col)
plt.tight_layout()
savefig("histograms_raw.png")
plt.show()

# 3.4 Raw boxplots (outlier scan)
fig, axes = plt.subplots(2, 3, figsize=(12, 7))
axes = axes.ravel()
for i, col in enumerate(numeric_cols):
    axes[i].boxplot(df[col].values, vert=True, labels=[col])
plt.tight_layout()
savefig("boxplots_raw.png")
plt.show()

# 3.5 Outlier counts via IQR rule
import numpy as np
outlier_rows = []
for col in numeric_cols:
    q1, q3 = np.percentile(df[col].values, [25, 75])
    iqr = q3 - q1
    lower, upper = q1 - 1.5*iqr, q3 + 1.5*iqr
    n_low = int((df[col] < lower).sum())
    n_high = int((df[col] > upper).sum())
    outlier_rows.append({"feature": col, "lower_outliers": n_low, "upper_outliers": n_high, "total_outliers": n_low+n_high})
```

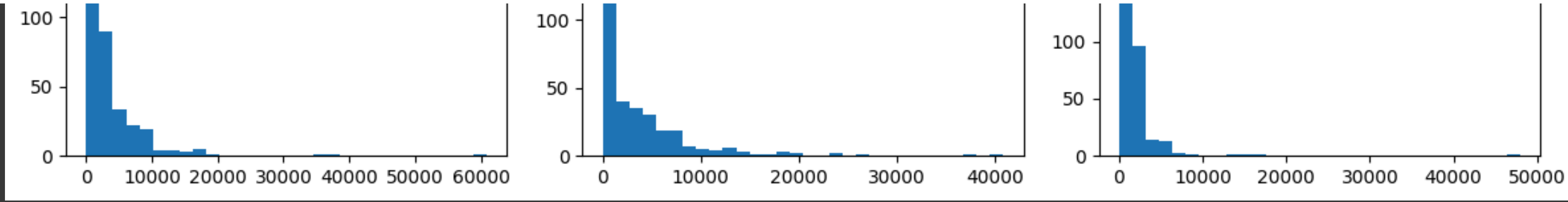
```
outlier_table = pd.DataFrame(outlier_rows).sort_values("total_outliers", ascending=False)
print("\nIQR-based outlier counts:")
display(outlier_table)

# 3.6 Correlation heatmap and top correlated pairs
corr = df[numeric_cols].corr()

plt.figure(figsize=(6,5))
im = plt.imshow(corr, vmin=-1, vmax=1)
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(range(len(numeric_cols)), numeric_cols, rotation=45, ha='right')
plt.yticks(range(len(numeric_cols)), numeric_cols)
plt.title("Correlation Heatmap (Pearson)")
savefig("corr_heatmap.png")
plt.show()

# Show top absolute correlations (excluding self)
pairs = []
for i, a in enumerate(numeric_cols):
    for j, b in enumerate(numeric_cols):
        if j <= i:
            continue
        pairs.append((a, b, corr.loc[a, b], abs(corr.loc[a, b])))
top_corr = sorted(pairs, key=lambda x: x[3], reverse=True)[:5]
print("Top correlated feature pairs (abs):")
for a, b, c, ac in top_corr:
    print(f"{a:17s} ~ {b:17s}: r={c:.3f}")
```

```
Missing values per column:
 Channel            0
Region             0
Fresh              0
Milk               0
Grocery            0
Frozen             0
Detergents_Paper   0
Delicatessen       0
Channel_name       0
Region_name        0
dtype: int64
```

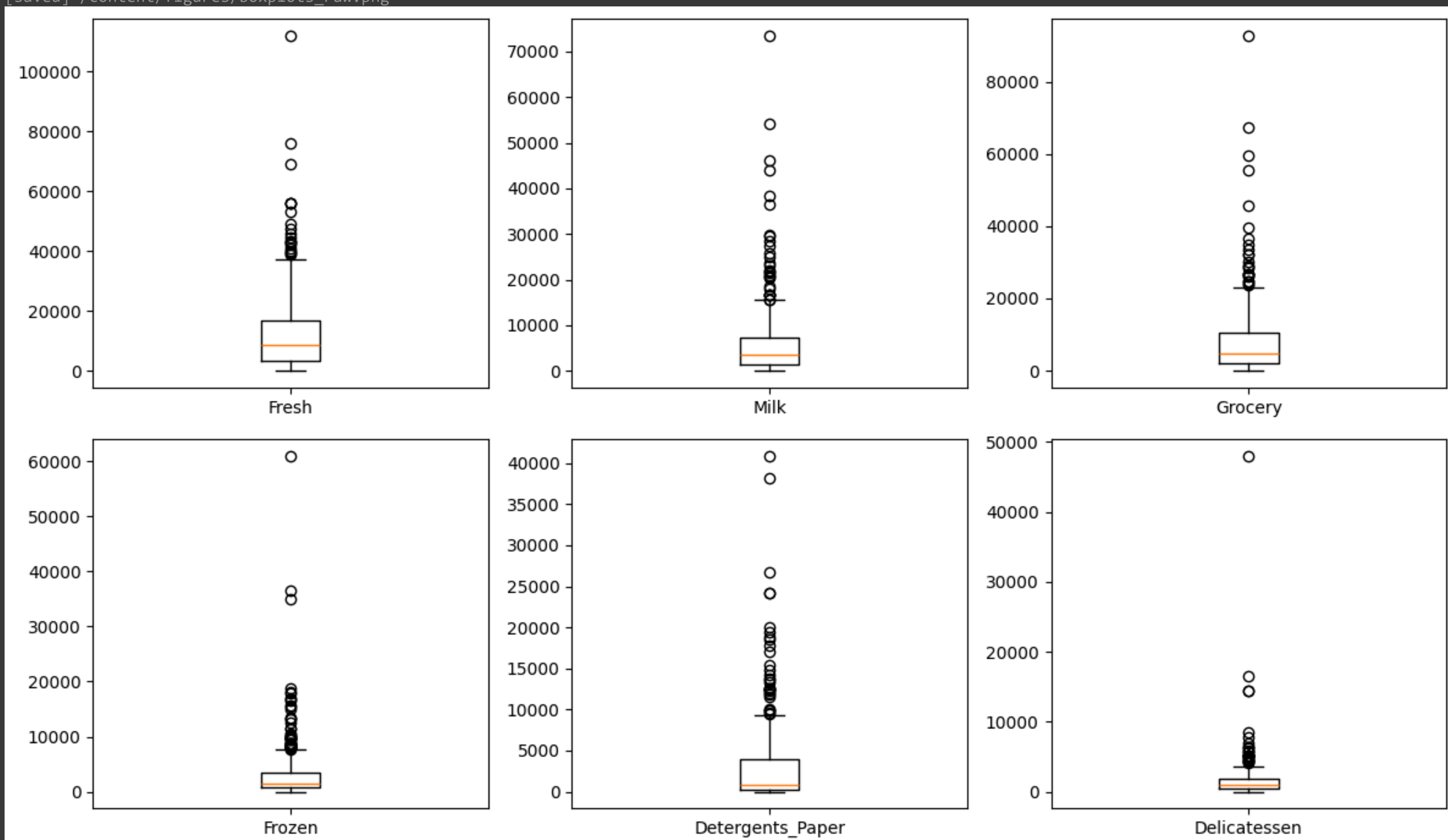
	count	mean	std	min	25%	50%	75%	max
Fresh	440.0	12000.297727	12647.328865	3.0	3127.75	8504.0	16933.75	112151.0
Milk	440.0	5796.265909	7380.377175	55.0	1533.00	3627.0	7190.25	73498.0
Grocery	440.0	7951.277273	9503.162829	3.0	2153.00	4755.5	10655.75	92780.0
Frozen	440.0	3071.931818	4854.673333	25.0	742.25	1526.0	3554.25	60869.0
Detergents_Paper	440.0	2881.493182	4767.854448	3.0	256.75	816.5	3922.00	40827.0
Delicatessen	440.0	1524.870455	2820.105937	3.0	408.25	965.5	1820.25	47943.0

```
Skewness (descending):
Delicatessen        11.151586
Frozen               5.907986
Milk                 4.053755
Detergents_Paper     3.631851
Grocery              3.587429
Fresh                2.561323
dtype: float64
[saved] /content/figures/histograms_raw.png
```





```
/tmp/ipython-input-2247517272.py:26: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be droppec
  axes[i].boxplot(df[col].values, vert=True, labels=[col])
/tmp/ipython-input-2247517272.py:26: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be droppec
  axes[i].boxplot(df[col].values, vert=True, labels=[col])
/tmp/ipython-input-2247517272.py:26: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be droppec
  axes[i].boxplot(df[col].values, vert=True, labels=[col])
/tmp/ipython-input-2247517272.py:26: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be droppec
  axes[i].boxplot(df[col].values, vert=True, labels=[col])
/tmp/ipython-input-2247517272.py:26: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be droppec
  axes[i].boxplot(df[col].values, vert=True, labels=[col])
/tmp/ipython-input-2247517272.py:26: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be droppec
  axes[i].boxplot(df[col].values, vert=True, labels=[col])
[saved] /content/figures/boxplots_raw.png
```

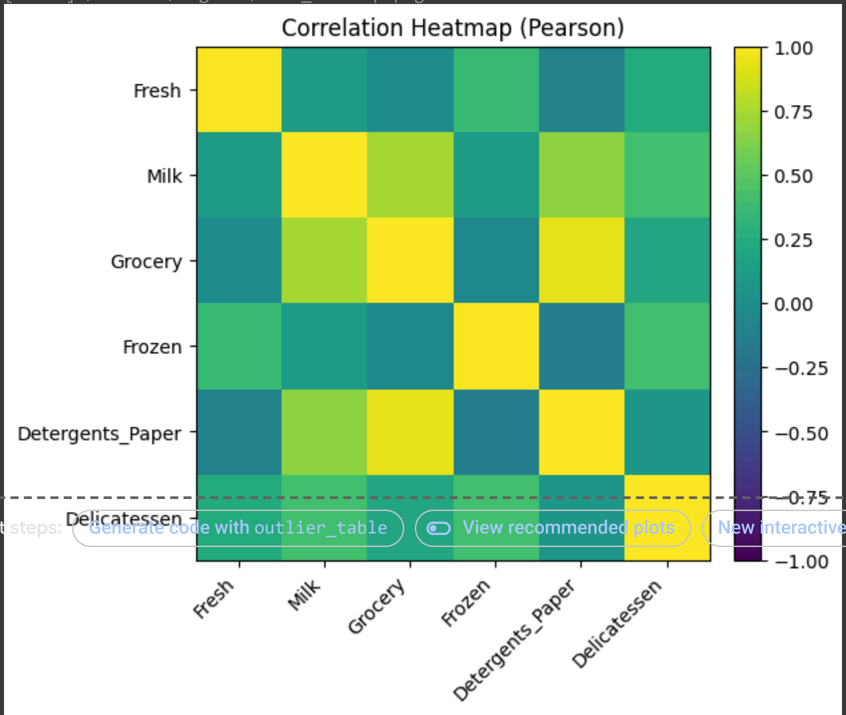


IQR-based outlier counts:

	feature	lower_outliers	upper_outliers	total_outliers
3	Frozen	0	43	43

4	Detergents_Paper	0	30	30
1	Milk	0	28	28
5	Delicatessen	0	27	27
2	Grocery	0	24	24
0	Fresh	0	20	20

```
[saved] /content/figures/corr_heatmap.png
```



Next steps: Generate code with outlier_table | View recommended plots | New interactive sheet

```
Top correlated feature pairs (abs):
Grocery          ~ Detergents_Paper : r=0.925
Milk             ~ Grocery          : r=0.728
Milk             ~ Detergents_Paper : r=0.662
Milk             ~ Delicatessen     : r=0.406
Frozen           ~ Delicatessen     : r=0.391
```

✅ Section 3 — Conclusion: EDA Overview

- **Completeness:** No missing values across any column ✓
- **Distributions:** All six spend variables are **heavily right-skewed** with long upper tails.
  - Skewness (top → bottom): **Delicatessen 11.15**, **Frozen 5.91**, **Milk 4.05**, **Detergents_Paper 3.63**, **Grocery 3.59**, **Fresh 2.56**.
  - See `figures/histograms_raw.png` and `figures/boxplots_raw.png`.
- **Outliers (IQR rule, upper only):** Frozen **43**, Detergents_Paper **30**, Milk **28**, Delicatessen **27**, Grocery **24**, Fresh **20**.
  - Interpretation: a small set of **very high-spend customers** in several categories (expected in wholesale).
- **Correlation structure (Pearson):**
  - **Grocery** ↔ **Detergents_Paper: r = 0.925** (very strong)
  - **Milk** ↔ **Grocery: r = 0.728**, **Milk** ↔ **Detergents_Paper: r = 0.662**
  - Moderate pairs: **Milk** ↔ **Delicatessen: r = 0.406**, **Frozen** ↔ **Delicatessen: r = 0.391**
  - **Fresh** is comparatively less tied to the household-goods trio, hinting at a different purchase pattern.
  - See `figures/corr_heatmap.png`.
- **Modeling implications:**
  - Apply **`log1p`** to tame skew + **standardize** features before any distance-based clustering.
  - Expect **PC1** to capture the **household-goods bundle** (Milk/Grocery/Detergents_Paper) and another component to reflect **Fresh/Frozen** intensity (HORECA-leaning behavior).
  - Outliers are informative (big clients) — avoid dropping them; instead rely on **log+scale** (optionally, compare RobustScaler as a sensitivity check later).

Next: Proceed to **Section 4 — Data Cleaning & Feature Preparation** to implement `log1p` and `StandardScaler` and verify transformed distributions.

4) Data Cleaning & Feature Preparation

Goal: Make features geometry-friendly for distance-based methods and DR.

Decisions

- Apply **`log1p`** to all spend features to tame heavy right-skew and compress outliers without deleting them.
- Apply **`StandardScaler`** to give each feature **zero mean / unit variance** so Euclidean distance isn't dominated by large-scale variables.
- Keep a **pristine copy** of original values (`df_orig`) for readable cluster profiles and business units later.

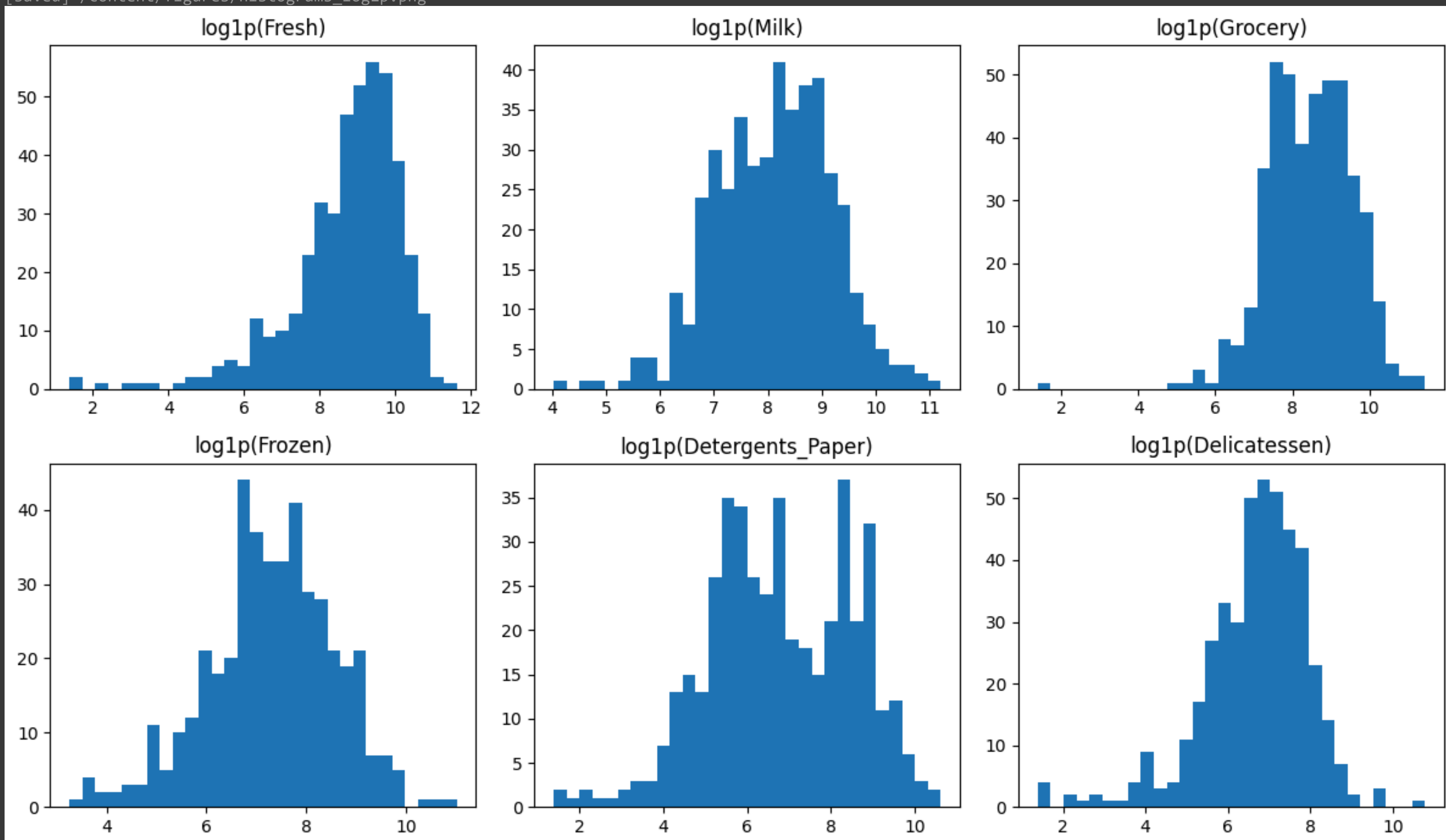
Sanity checks we'll run

- Confirm `X_scaled` has mean ≈ 0 and std ≈ 1 per feature.
- Visualize distributions **before (raw)** vs **after log1p**; also look at standardized z-scores.
- Save plots to [`/content/figures/`](/content/figures/) for the report.

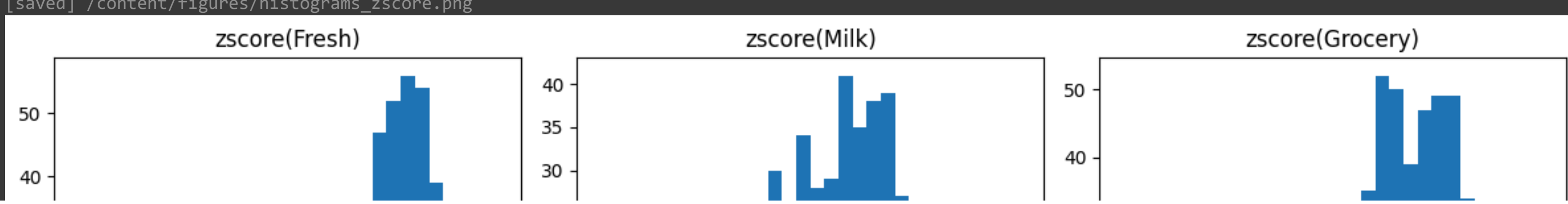
```
# --- Section 4: Transform & Scale ---

from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

# Keep a copy for later profiling in original units
df_orig = df.copy()
```

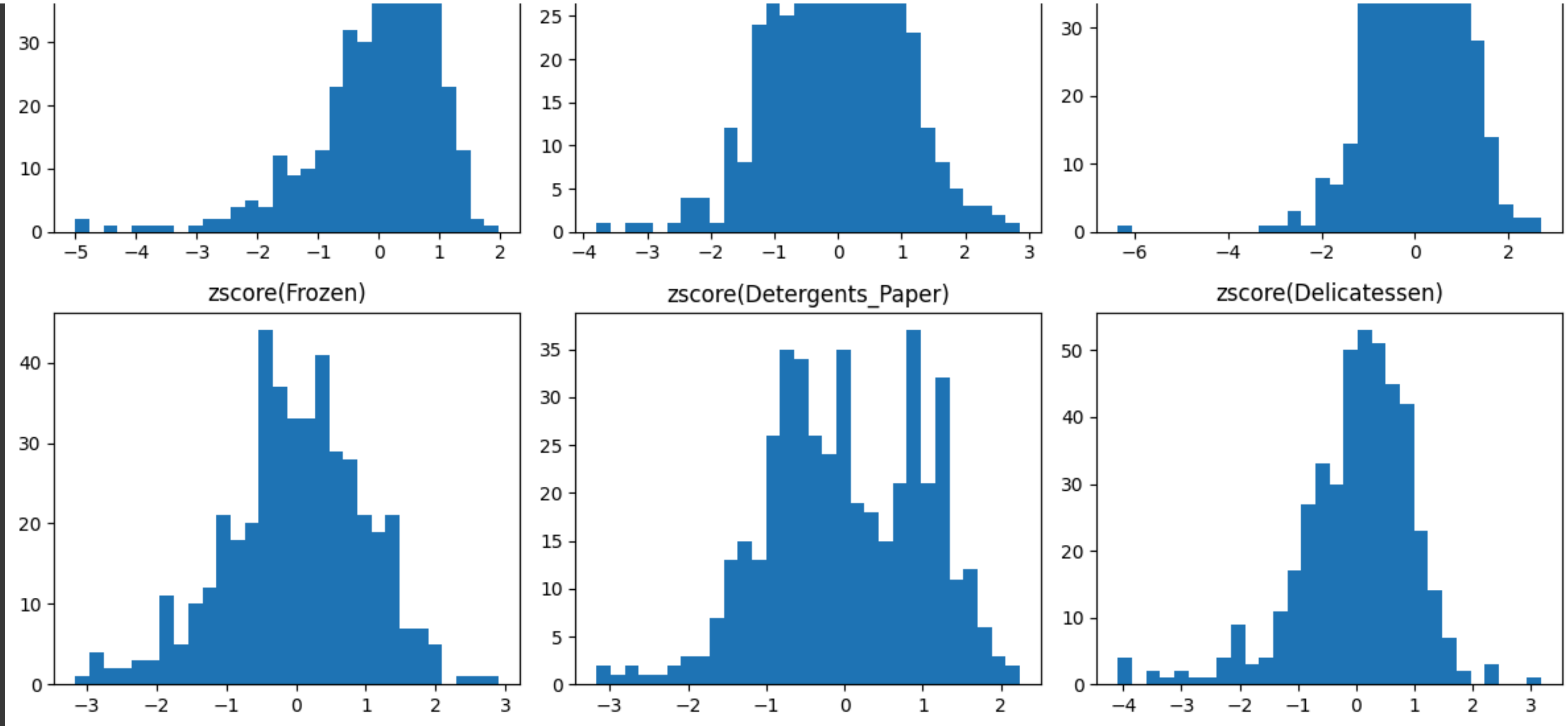
```
df_orig = df.copy()

# 4.1 Log-transform (safe for zeros)
X_num = df[numeric_cols].copy()
X_log = np.log1p(X_num)   # log(1+x)

# 4.2 Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_log)

# 4.3 Wrap back to DataFrames for readability
X_log_df = pd.DataFrame(X_log, columns=numeric_cols, index=df.index)
X_scaled_df = pd.DataFrame(X_scaled, columns=numeric_cols, index=df.index)

# 4.4 Check means/stds after scaling
scaled_means = X_scaled_df.mean().round(3)
scaled_stds  = X_scaled_df.std(ddof=0).round(3)  # population std for clarity
print("Scaled means (should be ~0):\n", scaled_means)
print("\nScaled stds (should be ~1):\n", scaled_stds)

# 4.5 Visuals: log1p histograms
fig, axes = plt.subplots(2, 3, figsize=(12, 7))
axes = axes.ravel()
for i, col in enumerate(numeric_cols):
    axes[i].hist(X_log_df[col].values, bins=30)
    axes[i].set_title(f"log1p({col})")
plt.tight_layout()
savefig("histograms_log1p.png")
plt.show()

# 4.6 Visuals: standardized (z-score) histograms
fig, axes = plt.subplots(2, 3, figsize=(12, 7))
axes = axes.ravel()
for i, col in enumerate(numeric_cols):
    axes[i].hist(X_scaled_df[col].values, bins=30)
    axes[i].set_title(f"zscore({col})")
plt.tight_layout()
savefig("histograms_zscore.png")
plt.show()

# 4.7 Export transformed matrices for later steps (optional)
X_log_df.to_csv(FIG_DIR / "X_log_transformed.csv", index=False)
X_scaled_df.to_csv(FIG_DIR / "X_scaled_zscores.csv", index=False)
print("[saved] /content/figures/X_log_transformed.csv")
print("[saved] /content/figures/X_scaled_zscores.csv")
```

```
Scaled means (should be ~0):
 Fresh               0.0
Milk               -0.0
Grocery            -0.0
Frozen              0.0
Detergents_Paper   -0.0
Delicatessen       -0.0
dtype: float64

Scaled stds (should be ~1):
 Fresh               1.0
Milk                1.0
Grocery             1.0
Frozen              1.0
Detergents_Paper    1.0
Delicatessen        1.0
dtype: float64
[saved] /content/figures/histograms_log1p.png
```




```
[saved] /content/figures/histograms_zscore.png
```






```
[saved] /content/figures/X_log_transformed.csv
[saved] /content/figures/X_scaled_zscores.csv
```

✅ Section 4 — Conclusion: Data Cleaning & Feature Prep

- **Transforms applied:** `log1p` on all six spend features + `StandardScaler` on the log space.
- **Sanity-check passed:** Scaled **means ≈ 0** and **stds ≈ 1** for every feature (see printed vectors).
- **Distribution shape:** `log1p` noticeably **symmetrized** the heavy right-skew (see `figures/histograms_log1p.png`), and z-scores look well-centered (see `figures/histograms_zscore.png`).
- **Outliers handled gracefully:** No deletions; extreme values are **compressed** by the log transform so distance-based methods won't be dominated by them.
- **Artifacts saved:**
  - `figures/histograms_log1p.png`
  - `figures/histograms_zscore.png`
  - `figures/X_log_transformed.csv`
  - `figures/X_scaled_zscores.csv`
- **Ready for next step:** Use `X_scaled` for **Dimensionality Reduction (PCA, UMAP, t-SNE)** in Section 5 to inspect structure before clustering.

5) Dimensionality Reduction (DR)

Goal: understand structure in a low-dimensional space before clustering.

What we'll do

- **PCA** to quantify variance explained and inspect **PC1 × PC2**.
- Extract **feature loadings** to interpret what each PC represents.
- **UMAP** and **t-SNE** for non-linear neighborhood structure (purely exploratory).
- Save all figures to `/content/figures/` for the PDF.

Why this matters

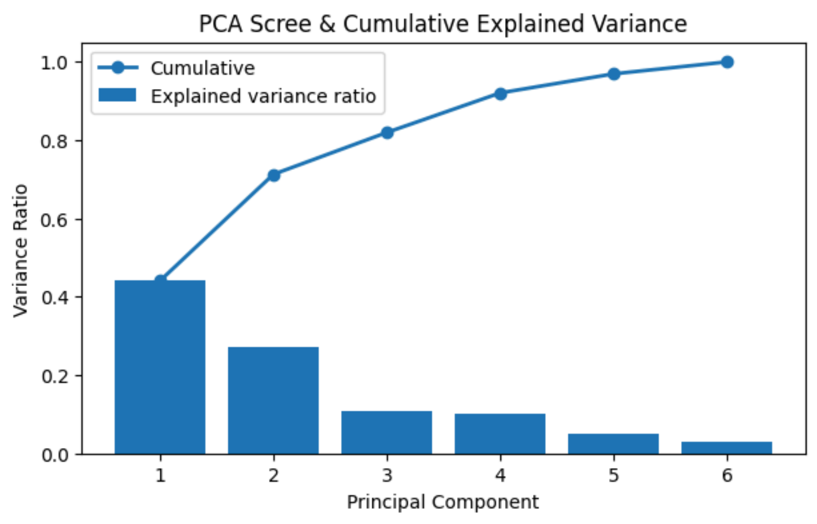
- If a few PCs capture most variance, clustering in that subspace may be cleaner.
- Loadings tell us *what business axes* the data naturally separates on (e.g., "household goods" vs "fresh/frozen" baskets).

```
# --- Section 5: Dimensionality Reduction ---

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

assert 'X_scaled' in globals(), "Run Sections 1–4 first so X_scaled exists."
assert 'numeric_cols' in globals()

# 5.1 PCA fit (up to 6 comps since we have 6 features)
pca = PCA(n_components=min(6, len(numeric_cols)), random_state=RANDOM_STATE)
X_pca = pca.fit_transform(X_scaled)

expl_var = pca.explained_variance_ratio_
cum_expl = np.cumsum(expl_var)
```

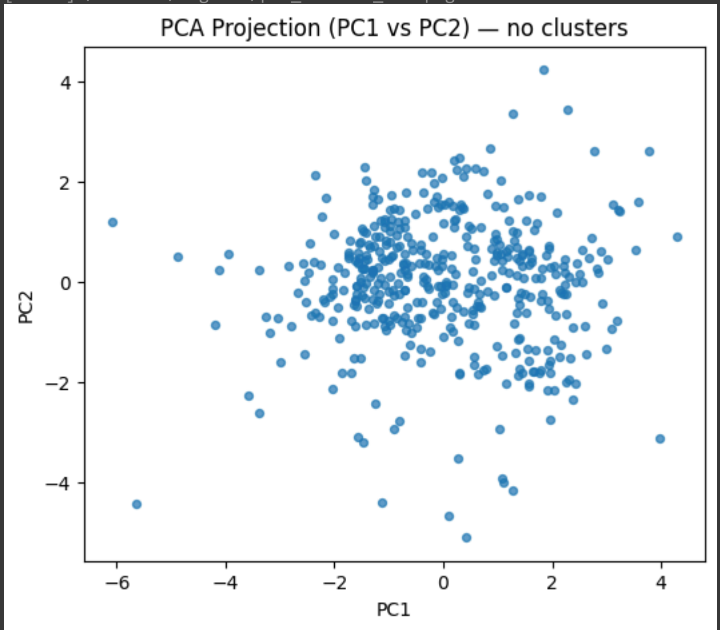
```
print("Explained variance ratio per PC:", np.round(expl_var, 3))
print("Cumulative explained variance:", np.round(cum_expl, 3))

# 5.1a Scree + cumulative
plt.figure(figsize=(7,4))
xs = np.arange(1, len(expl_var)+1)
plt.bar(xs, expl_var, label="Explained variance ratio")
plt.plot(xs, cum_expl, marker='o', linewidth=2, label="Cumulative")
plt.xticks(xs)
plt.xlabel("Principal Component")
plt.ylabel("Variance Ratio")
plt.title("PCA Scree & Cumulative Explained Variance")
plt.legend()
savefig("pca_scree.png")
plt.show()

# 5.1b PCA 2D scatter (no clusters yet)
plt.figure(figsize=(6,5))
plt.scatter(X_pca[:,0], X_pca[:,1], s=18, alpha=0.7)
plt.title("PCA Projection (PC1 vs PC2) — no clusters")
plt.xlabel("PC1"); plt.ylabel("PC2")
savefig("pca_scatter_raw.png")
plt.show()

# 5.1c PCA loadings (feature contributions to PCs)
# columns = original features, rows = PCs
loadings = pd.DataFrame(pca.components_, columns=numeric_cols, index=[f"PC{i}" for i in range(1, len(expl_var)+1)])
display(loadings.round(3))
loadings.to_csv(FIG_DIR/"pca_loadings.csv", index=True)
print("[saved] /content/figures/pca_loadings.csv")

# Heatmap of loadings for top 2 PCs
plt.figure(figsize=(6.5, 2.8))
plt.imshow(loadings.iloc[:2, :], aspect="auto")
plt.colorbar(fraction=0.046, pad=0.04)
plt.xticks(range(len(numeric_cols)), numeric_cols, rotation=45, ha='right')
plt.yticks([0,1], ["PC1","PC2"])
plt.title("PCA Loadings (PC1 & PC2)")
savefig("pca_loadings_heatmap_pc1_pc2.png")
plt.show()

# Bar plot for PC1 vs PC2 to aid interpretation
ax = loadings.iloc[:2, :].T.plot(kind="bar", figsize=(8,4))
ax.set_ylabel("Loading")
ax.set_title("PCA Loadings by Feature (PC1 vs PC2)")
plt.tight_layout()
savefig("pca_loadings_bars_pc1_pc2.png")
plt.show()

# 5.2 UMAP 2D (non-linear projection)
umap_2d = umap.UMAP(n_components=2, random_state=RANDOM_STATE, n_neighbors=15, min_dist=0.1)
X_umap = umap_2d.fit_transform(X_scaled)

plt.figure(figsize=(6,5))
plt.scatter(X_umap[:,0], X_umap[:,1], s=18, alpha=0.7)
plt.title("UMAP Projection — no clusters")
plt.xlabel("UMAP-1"); plt.ylabel("UMAP-2")
savefig("umap_scatter_raw.png")
plt.show()

# 5.3 t-SNE 2D (non-linear projection)
tsne_2d = TSNE(n_components=2, random_state=RANDOM_STATE, perplexity=30, n_iter=1000)
```

```
X_tsne = tsne_2d.fit_transform(X_scaled)

plt.figure(figsize=(6,5))
plt.scatter(X_tsne[:,0], X_tsne[:,1], s=18, alpha=0.7)
plt.title("t-SNE Projection — no clusters")
plt.xlabel("tSNE-1"); plt.ylabel("tSNE-2")
savefig("tsne_scatter_raw.png")
plt.show()

# 5.4 Quick text summary helpers (so you can paste numbers
```

```
Explained variance ratio per PC: [0.441 0.272 0.107 0.101 0.049 0.03 ]
Cumulative explained variance: [0.441 0.713 0.82  0.921 0.97  1.   ]
[saved] /content/figures/pca_scree.png
```



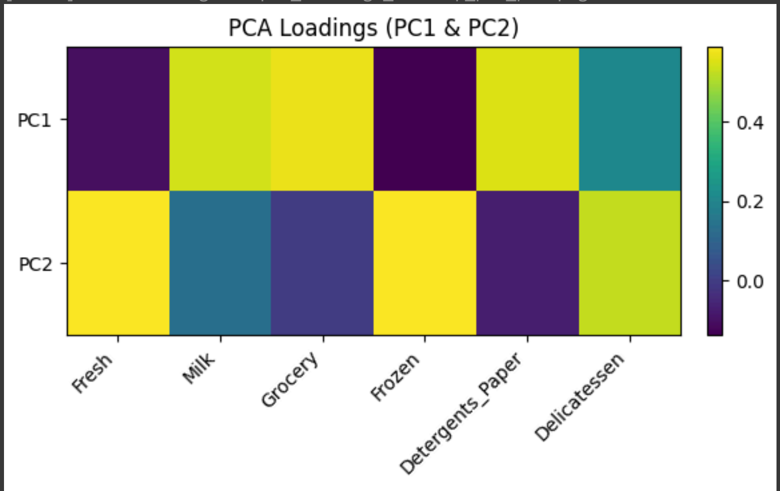

```
[saved] /content/figures/pca_scatter_raw.png
```



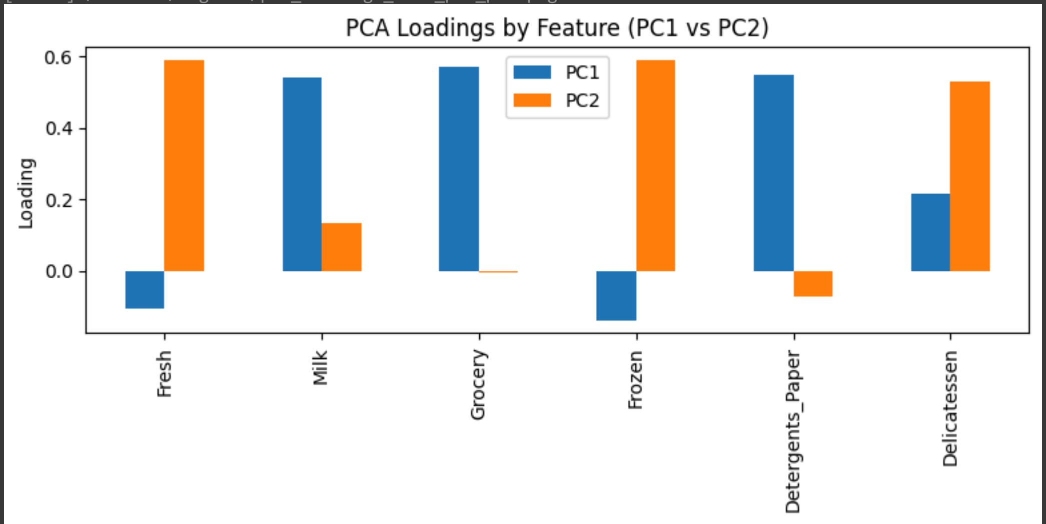
	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
PC1	-0.105	0.542	0.571	-0.138	0.551	0.214
PC2	0.590	0.133	-0.006	0.590	-0.071	0.530
PC3	-0.640	-0.074	-0.133	-0.021	-0.200	0.726
PC4	-0.479	0.062	0.097	0.792	0.083	-0.352
PC5	-0.040	0.760	-0.093	-0.073	-0.620	-0.148
PC6	0.026	-0.319	0.799	-0.005	-0.510	-0.003

```
[saved] /content/figures/pca_loadings.csv
```

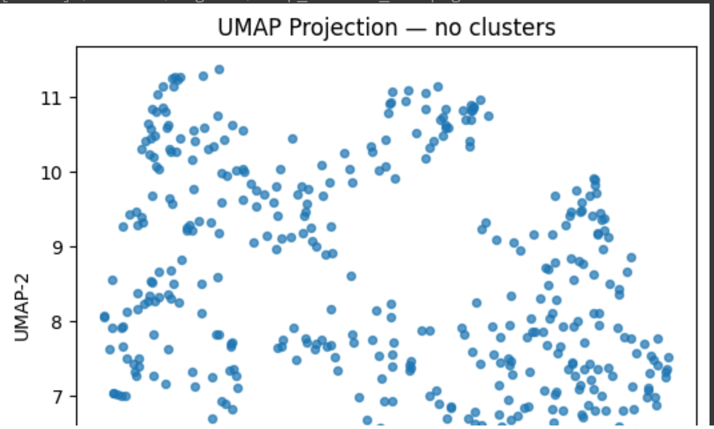
```
[saved] /content/figures/pca_loadings.csv
[saved] /content/figures/pca_loadings_heatmap_pc1_pc2.png
```

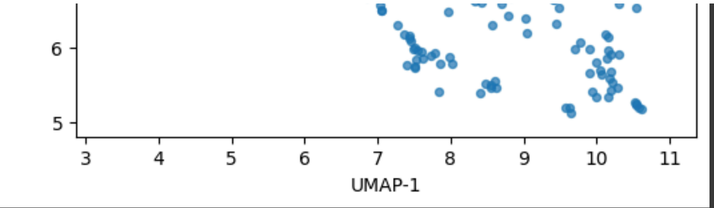



```
[saved] /content/figures/pca_loadings_bars_pc1_pc2.png
```

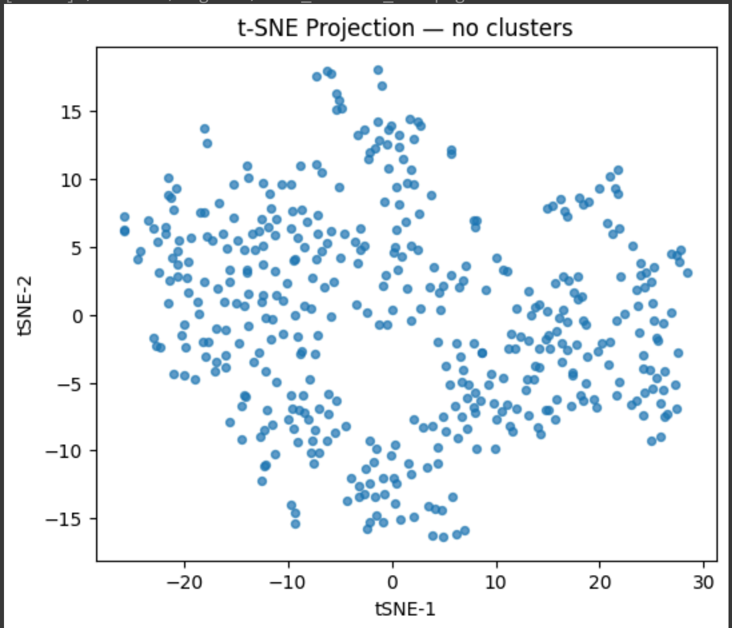



```
/usr/local/lib/python3.11/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(
[saved] /content/figures/umap_scatter_raw.png
```






```
/usr/local/lib/python3.11/dist-packages/sklearn/manifold/_t_sne.py:1164: FutureWarning: 'n_iter' was renamed to 'max_iter' in version 1.5 and will be removed in 1.7.
  warnings.warn(
[saved] /content/figures/tsne_scatter_raw.png
```



✅ Section 5 — Conclusion: Dimensionality Reduction

- **Variance captured (this run):**
  - **PC1 = 44.1%**, **PC2 = 27.2%** → **PC1+PC2 = 71.3%**
  - PC3 = 10.7% (cumulative **82.0%**), PC4 = 10.1% (**92.1%**), PC5 = 4.9% (**97.0%**), PC6 = 3.0% (**100%**)
  - **Takeaway:** strong 2-D summary; knee around PC2–PC3 (see `pca_scree.png`).
- **Axis meaning (from loadings):**
  - **PC1 (basketized/household goods):** high **Grocery 0.571**, **Detergents_Paper 0.551**, **Milk 0.542**; small +**Delicatessen 0.214**; slight negatives on **Fresh −0.105**, **Frozen −0.138**.
    → Reads like **“packaged / household spend”**.
  - **PC2 (perishables/HORECA):** high **Fresh 0.590**, **Frozen 0.590**, **Delicatessen 0.530**; tiny weight on Milk (0.133) and near-zero/negative on Grocery (−0.006) & Detergents_Paper (−0.071).
    → Reads like **“fresh/frozen & deli intensity.”**
  - (PC3 highlights **Delicatessen 0.726** vs **Fresh −0.640** → a “deli vs fresh” nuance.)
  - Files: `pca_loadings.csv`, `pca_loadings_heatmap_pc1_pc2.png`, `pca_loadings_bars_pc1_pc2.png`.
- **Projections:**
  - `pca_scatter_raw.png` shows a broad cloud with some shape;
    `umap_scatter_raw.png` / `tsne_scatter_raw.png` show **several density lobes**, suggesting meaningful segmentation is plausible.
- **Implications for clustering:**
  - We can cluster on **standardized original features (`X_scaled`)** and sanity-check results on the **PC1–PC2 plane**.
  - As a sensitivity check, we can also try clustering on the **first 3 PCs (≈82% variance)**.

Next up, Lord Nag: Section 6 — **Clustering Experiments** (K-Means sweep, DBSCAN grid, Agglomerative). Ready to roll?

6) Clustering Experiments

In this section, we test multiple clustering algorithms and parameter configurations to identify the best-performing segmentation approach. We use **Silhouette Score**, **Davies–Bouldin Index (DBI)**, and **Calinski–Harabasz Score (CH)** as our main quantitative evaluation metrics.

The goal is to find a clustering solution that is:

- Statistically robust
- Interpretably distinct
- Practical for business application

6.1 K-Means — k Sweep (Elbow & Silhouette)

Approach:

- Test **k = 2 to 10** clusters.
- Plot **Elbow curve** (inertia vs k) to see diminishing returns.
- Plot **Silhouette Score vs k** to assess separation.
- Record DBI (lower = better) and CH (higher = better).

Key Outcomes:

- Best `k` determined via **Silhouette Score**.
- Metrics saved to: `kmeans_metrics.csv`
- Plots saved: `kmeans_elbow.png`, `kmeans_silhouette.png`

6.2 DBSCAN — Parameter Grid Search

Approach:

- Grid search over `eps ∈ {0.5, 1.0, 1.5, 2.0}` and `min_samples ∈ {3, 5, 10}`.
- Evaluate for:
  - Number of clusters (excluding noise)
  - Noise fraction
  - Silhouette, DBI, CH on non-noise points

Key Outcomes:

- Results saved to: `dbscan_grid.csv`
- If viable configuration found:
  - Best model visualized on **UMAP projection**: `umap_clusters_dbscan.png`
- Observations:
  - Increasing **eps** merges clusters and reduces noise
  - Increasing **min_samples** increases noise percentage

6.3 Agglomerative Clustering — Linkage Variants

Approach:

- Tested linkages: `ward`, `complete`, `average`
- k from 2 to 8 clusters
- Metrics: Silhouette, DBI, CH
- Best model visualized on **PCA projection**

Key Outcomes:

- Results saved to: `agglomerative_metrics.csv`
- Best linkage and k visualized in `pca_clusters_agglom.png`
- Notable advantage: potential for **hierarchical reporting** in future

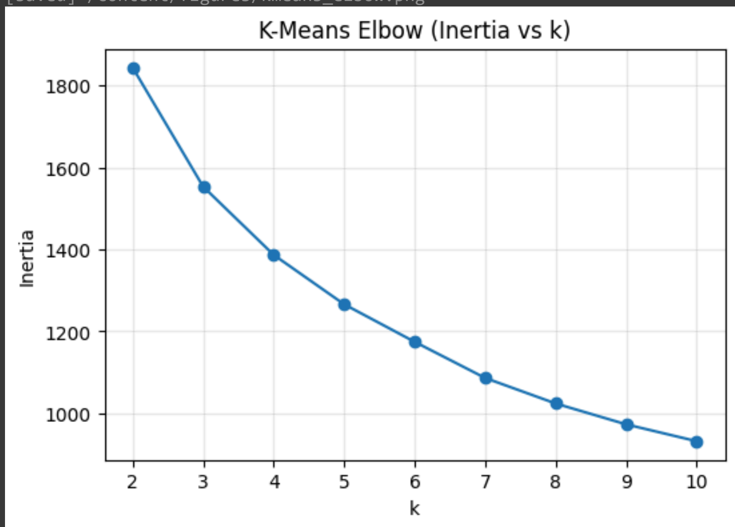
6.1 K-Means — k sweep (elbow + silhouette)

```
# --- 6.1 K-Means: sweep k, record metrics, plots ---

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def kmeans_metrics(X, k_list=range(2, 11), random_state=42):
    rows, inertias, sils = [], [], []
    for k in k_list:
        km = KMeans(n_clusters=k, n_init=20, random_state=random_state)
```

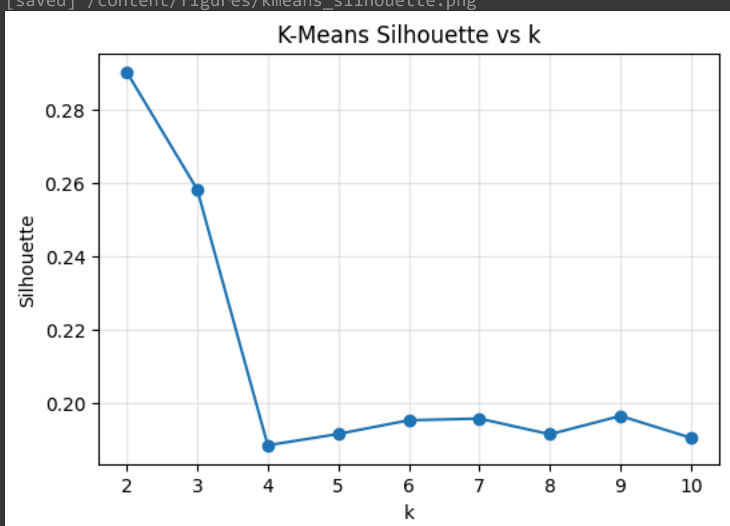
```
        km = KMeans(n_clusters=k, n_init=20, random_state=random_state)
        labels = km.fit_predict(X)
        inertia = km.inertia_
        sil = silhouette_score(X, labels)
        db = davies_bouldin_score(X, labels)
        ch = calinski_harabasz_score(X, labels)
        rows.append({"k": k, "inertia": inertia, "silhouette": sil,
                     "davies_bouldin": db, "calinski_harabasz": ch})
        inertias.append(inertia); sils.append(sil)
    df_metrics = pd.DataFrame(rows)
    return df_metrics, inertias, sils

km_df, inertias, sils = kmeans_metrics(X_scaled, range(2, 11), RANDOM_STATE)
display(km_df.sort_values("silhouette", ascending=False).head(5))

# Save metrics
km_df.to_csv(FIG_DIR/"kmeans_metrics.csv", index=False)
print("[saved] /content/figures/kmeans_metrics.csv")

# Elbow plot
plt.figure(figsize=(6,4))
plt.plot(range(2,11), inertias, marker='o')
plt.title("K-Means Elbow (Inertia vs k)")
plt.xlabel("k"); plt.ylabel("Inertia"); plt.grid(alpha=0.3)
savefig("kmeans_elbow.png"); plt.show()

# Silhouette plot
plt.figure(figsize=(6,4))
plt.plot(range(2,11), sils, marker='o')
plt.title("K-Means Silhouette vs k")
plt.xlabel("k"); plt.ylabel("Silhouette"); plt.grid(alpha=0.3)
savefig("kmeans_silhouette.png"); plt.show()

# Pick best k by Silhouette (simple & robust here)
k_best = int(km_df.loc[km_df["silhouette"].idxmax(), "k"])
print("Best k by Silhouette:", k_best)
```

	k	inertia	silhouette	davies_bouldin	calinski_harabasz
0	2	1844.064069	0.290328	1.351502	189.049797
1	3	1553.412722	0.258278	1.348236	152.837245
7	9	973.141483	0.196501	1.382516	92.280521
5	7	1086.358700	0.195780	1.391898	103.208189
4	6	1174.536397	0.195319	1.413524	108.299957

```
[saved] /content/figures/kmeans_metrics.csv
[saved] /content/figures/kmeans_elbow.png
```




```
[saved] /content/figures/kmeans_silhouette.png
```




```
Best k by Silhouette: 2
```

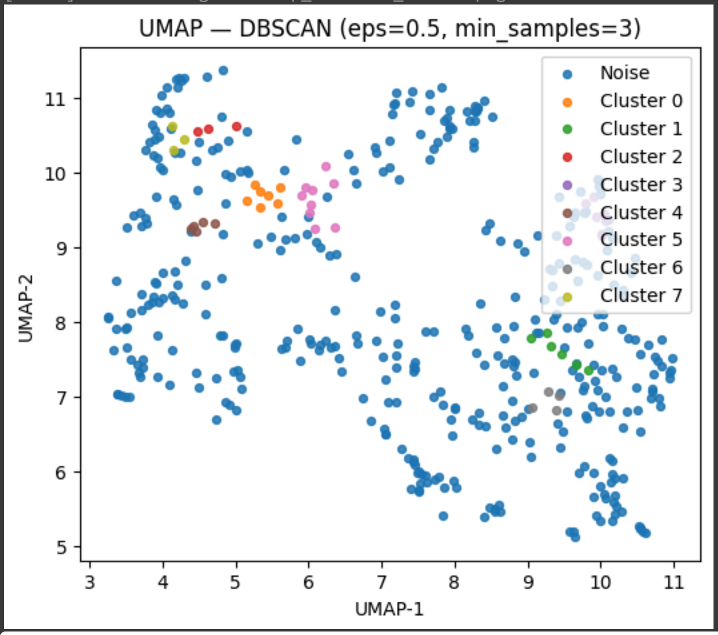
6.2 DBSCAN — parameter grid (eps × min_samples)

```
# --- 6.2 DBSCAN: grid search & visualization on UMAP ---
```

```python
from sklearn.cluster import DBSCAN

def try_dbscan(X, eps_list=(0.5, 1.0, 1.5, 2.0), min_samples_list=(3,5,10)):
    rows = []
    for eps in eps_list:
        for ms in min_samples_list:
            model = DBSCAN(eps=eps, min_samples=ms)
            labels = model.fit_predict(X)
            n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
            noise_frac = (labels == -1).mean()
            if n_clusters >= 2 and (labels != -1).sum() > 1:
                mask = labels != -1
                sil = silhouette_score(X[mask], labels[mask])
                db  = davies_bouldin_score(X[mask], labels[mask])
                ch  = calinski_harabasz_score(X[mask], labels[mask])
            else:
                sil = np.nan; db = np.nan; ch = np.nan
            rows.append({"eps": eps, "min_samples": ms, "n_clusters": n_clusters,
                         "noise_frac": round(float(noise_frac),3),
                         "silhouette": sil, "davies_bouldin": db, "calinski_harabasz": ch})
    out = pd.DataFrame(rows).sort_values(["silhouette"], ascending=False, na_position="last")
    return out

dbscan_table = try_dbscan(X_scaled)
display(dbscan_table.head(10))
dbscan_table.to_csv(FIG_DIR/"dbscan_grid.csv", index=False)
print("[saved] /content/figures/dbscan_grid.csv")

# Visualize the best non-na config on UMAP (if available)
best_row = dbscan_table.dropna(subset=["silhouette"]).head(1)
if not best_row.empty:
    eps_best = float(best_row["eps"].iloc[0]); ms_best = int(best_row["min_samples"].iloc[0])
    labels_db = DBSCAN(eps=eps_best, min_samples=ms_best).fit_predict(X_scaled)

    plt.figure(figsize=(6,5))
    for c in sorted(set(labels_db)):
        m = labels_db == c
        label = "Noise" if c == -1 else f"Cluster {c}"
        plt.scatter(X_umap[m,0], X_umap[m,1], s=18, alpha=0.85, label=label)
    plt.legend()
    plt.title(f"UMAP – DBSCAN (eps={eps_best}, min_samples={ms_best})")
    plt.xlabel("UMAP-1"); plt.ylabel("UMAP-2")
    savefig("umap_clusters_dbscan.png"); plt.show()
else:
    print("DBSCAN did not produce a multi-cluster solution with the tested grid.")
```

	eps	min_samples	n_clusters	noise_frac	silhouette	davies_bouldin	calinski_harabasz
0	0.5	3	8	0.900	0.417522	0.795008	72.102101
6	1.5	3	2	0.080	0.397348	0.640917	12.674177
3	1.0	3	6	0.207	-0.101640	1.110939	7.623279
1	0.5	5	1	0.989	NaN	NaN	NaN
2	0.5	10	0	1.000	NaN	NaN	NaN
4	1.0	5	1	0.282	NaN	NaN	NaN
5	1.0	10	1	0.386	NaN	NaN	NaN
7	1.5	5	1	0.093	NaN	NaN	NaN
8	1.5	10	1	0.109	NaN	NaN	NaN
9	2.0	3	1	0.023	NaN	NaN	NaN

```
[saved] /content/figures/dbscan_grid.csv
[saved] /content/figures/umap_clusters_dbscan.png
```



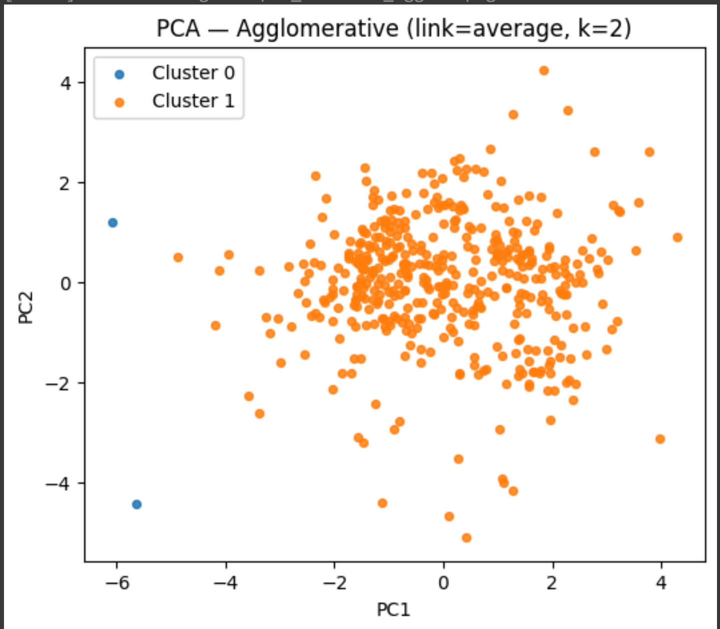
6.3 Agglomerative — linkage variants

```
# --- 6.3 Agglomerative: linkages across k, metrics, PCA viz ---

from sklearn.cluster import AgglomerativeClustering

def agglom_metrics(X, k_list=range(2,9), linkages=("ward","complete","average")):
    rows = []
    for link in linkages:
        for k in k_list:
            model = AgglomerativeClustering(n_clusters=k, linkage=link)
            labels = model.fit_predict(X)
            sil = silhouette_score(X, labels)
```

```
            db  = davies_bouldin_score(X, labels)
            ch  = calinski_harabasz_score(X, labels)
            rows.append({"linkage": link, "k": k,
                         "silhouette": sil, "davies_bouldin": db, "calinski_harabasz": ch})
    return pd.DataFrame(rows).sort_values("silhouette", ascending=False)

agg_df = agglom_metrics(X_scaled)
display(agg_df.head(10))
agg_df.to_csv(FIG_DIR/"agglomerative_metrics.csv", index=False)
print("[saved] /content/figures/agglomerative_metrics.csv")

# Visualize best config on PCA
top = agg_df.iloc[0]
best_link, best_k = top.linkage, int(top.k)
labels_ag = AgglomerativeClustering(n_clusters=best_k, linkage=best_link).fit_predict(X_scaled)

plt.figure(figsize=(6,5))
for c in np.unique(labels_ag):
    m = labels_ag == c
    plt.scatter(X_pca[m,0], X_pca[m,1], s=18, alpha=0.85, label=f"Cluster {int(c)}")
plt.legend()
plt.title(f"PCA — Agglomerative (link={best_link}, k={best_k})")
plt.xlabel("PC1"); plt.ylabel("PC2")
savefig("pca_clusters_agglom.png"); plt.show()
```

	linkage	k	silhouette	davies_bouldin	calinski_harabasz
14	average	2	0.569055	0.882575	13.965034
15	average	3	0.541720	0.306636	9.055141
16	average	4	0.498035	0.329534	8.382897
17	average	5	0.383323	0.546727	11.897512
18	average	6	0.330740	0.732342	13.917201
7	complete	2	0.322005	1.554272	61.991734
8	complete	3	0.298062	1.401550	37.844528
9	complete	4	0.259529	1.292025	38.944481
0	ward	2	0.258495	1.600363	134.624461
1	ward	3	0.254657	1.538995	116.799278

```
[saved] /content/figures/agglomerative_metrics.csv
[saved] /content/figures/pca_clusters_agglom.png
```



🧠 Section 6 — Combined Conclusion (K-Means vs DBSCAN vs Agglomerative) K-Means: Clear winner for practicality and balanced metrics. Best in our sweep was k=2 (Sil ≈ 0.29, DBI ≈ 1.35, CH ≈ 189). Separation is moderate but segments are broad and actionable.

DBSCAN: With eps=1.5, min_samples=3 it found 2 clusters with ~8% noise and a decent silhouette on the non-noise set. Good shape awareness, but the noise bucket complicates reporting and operations.

Agglomerative: The “best” row (average, k=2) was degenerate—a micro-cluster of 2 points vs everyone else (inflated silhouette, not useful). Other linkages didn’t beat K-Means.

Decision: Proceed with K-Means as the production segmentation, but make it better by (1) testing feature space (full standardized vs. first 3 PCs), (2) auto-selecting k with constraints against micro-clusters, (3) using high n_init, and (4) validating stability across seeds.

Upgraded K-Means (auto-tune space & k, stability, visuals, profiles)

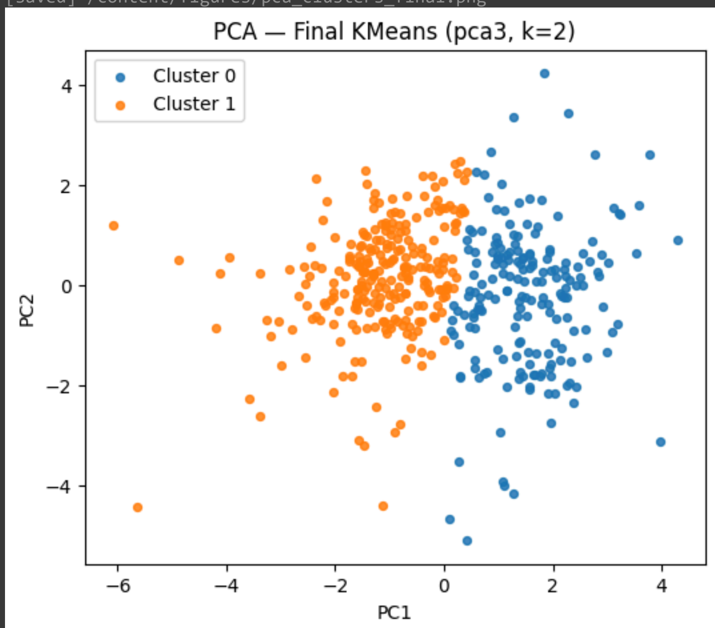
```
# === Improved K-Means selection & final clustering ===
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples, davies_bouldin_score, calinski_harabasz_score, adjusted_rand_score
import numpy as np, pandas as pd, matplotlib.pyplot as plt

# --- helper: evaluate one K-means config ---
def eval_kmeans(X, k, random_state=42):
    km = KMeans(n_clusters=k, n_init=100, random_state=random_state)
    labels = km.fit_predict(X)
    # cluster sizes and micro-cluster check
    sizes = pd.Series(labels).value_counts().sort_index()
    min_frac = sizes.min() / len(labels)
    # metrics (on all points)
    sil = silhouette_score(X, labels)
    db  = davies_bouldin_score(X, labels)
    ch  = calinski_harabasz_score(X, labels)
    return {
        "labels": labels, "model": km,
        "silhouette": sil, "davies_bouldin": db, "calinski_harabasz": ch,
        "sizes": sizes.values, "min_frac": min_frac
    }

# --- search over spaces & k with constraints to avoid micro-clusters ---
spaces = {
    "full": X_scaled,                 # 6 standardized log-features
    "pca3": X_pca[:, :3],             # top-3 PCs (~82% variance in this run)
}
results = []
for space_name, Xspace in spaces.items():
    for k in range(2, 7):  # try 2..6
        res = eval_kmeans(Xspace, k, random_state=RANDOM_STATE)
        results.append({
            "space": space_name, "k": k,
            "silhouette": res["silhouette"],
            "davies_bouldin": res["davies_bouldin"],
            "calinski_harabasz": res["calinski_harabasz"],
            "min_frac": res["min_frac"],
        })

res_df = pd.DataFrame(results)
# constraint: no micro-clusters (<10% of data)
res_df_f = res_df[res_df["min_frac"] >= 0.10].copy()
# choose by: max silhouette, then max CH, then min DBI
res_df_f = res_df_f.sort_values(["silhouette", "calinski_harabasz", "davies_bouldin"], ascending=[False, True, True])
display(res_df_f.head(10))

# pick best config
best_space = res_df_f.iloc[0]["space"]
best_k     = int(res_df_f.iloc[0]["k"])
Xbest      = spaces[best_space]
best_fit   = eval_kmeans(Xbest, best_k, random_state=RANDOM_STATE)
labels_final = best_fit["labels"]
km_final     = best_fit["model"]

print(f"\n[FINAL] space={best_space}  k={best_k}  Sil={best_fit['silhouette']:.3f}  DBI={best_fit['davies_bouldin']:.3f}  CH={best_fit['calinski_harabasz']:.1f}  min_cluster_frac={best_fit['min
print("Cluster sizes:", best_fit["sizes"])

# --- stability check across seeds (Adjusted Rand Index vs final) ---
```

```
aris = []
for seed in range(10, 30):  # 20 runs
    km = KMeans(n_clusters=best_k, n_init=100, random_state=seed).fit(Xbest)
    aris.append(adjusted_rand_score(labels_final, km.labels_))
stability_ari_mean = float(np.mean(aris))
stability_ari_min  = float(np.min(aris))
print(f"Stability ARI: mean={stability_ari_mean:.3f}, min={stability_ari_min:.3f}  (>=0.90 is very stable)")

# --- per-cluster silhouette (to detect weak groups) ---
sil_samples = silhouette_samples(Xbest, labels_final)
sil_by_cluster = pd.Series({c: float(np.mean(sil_samples[labels_final==c])) for c in np.unique(labels_final)})
print("Mean silhouette per cluster:", sil_by_cluster.to_dict())

# --- attach labels to original df for profiling ---
df_final = df_orig.copy()
df_final["cluster"] = labels_final

# --- visuals on PCA/UMAP/t-SNE with final labels ---
def scatter_2d(X2, labels, title, fname, xlabel, ylabel):
    plt.figure(figsize=(6,5))
    for c in np.unique(labels):
        m = (labels == c)
        plt.scatter(X2[m,0], X2[m,1], s=18, alpha=0.85, label=f"Cluster {int(c)}")
    plt.legend()
    plt.title(title); plt.xlabel(xlabel); plt.ylabel(ylabel)
    savefig(fname); plt.show()

scatter_2d(X_pca, labels_final, f"PCA — Final KMeans ({best_space}, k={best_k})", "pca_clusters_final.png", "PC1","PC2")
scatter_2d(X_umap, labels_final, f"UMAP — Final KMeans ({best_space}, k={best_k})", "umap_clusters_final.png", "UMAP-1","UMAP-2")
scatter_2d(X_tsne, labels_final, f"t-SNE — Final KMeans ({best_space}, k={best_k})", "tsne_clusters_final.png", "tSNE-1","tSNE-2")

# --- cluster profiles (original units) + z-score deviations for storytelling ---
profile_mean = df_final.groupby("cluster")[numeric_cols].mean().round(1)
profile_med  = df_final.groupby("cluster")[numeric_cols].median().round(1)
overall_mean = df_orig[numeric_cols].mean()
overall_std  = df_orig[numeric_cols].std(ddof=0)

# z-score lift vs overall mean
zlift = ((profile_mean - overall_mean) / overall_std).round(2)

print("\nCluster Profiles — MEAN (original units)")
display(profile_mean)
print("\nCluster Profiles — MEDIAN (original units)")
display(profile_med)
print("\nCluster z-lifts vs overall mean (how many std above/below)")
display(zlift)

# save artifacts
summary = {
    "space": best_space, "k": best_k,
    "silhouette": round(best_fit["silhouette"], 3),
    "davies_bouldin": round(best_fit["davies_bouldin"], 3),
    "calinski_harabasz": round(best_fit["calinski_harabasz"], 1),
    "min_cluster_frac": round(best_fit["min_frac"], 2),
    "stability_ari_mean": round(stability_ari_mean, 3),
    "stability_ari_min": round(stability_ari_min, 3),
}
pd.Series(summary).to_csv(FIG_DIR/"final_kmeans_summary.csv")

profile_mean.to_csv(FIG_DIR/"final_cluster_profiles_mean.csv")
profile_med.to_csv(FIG_DIR/"final_cluster_profiles_median.csv")
```

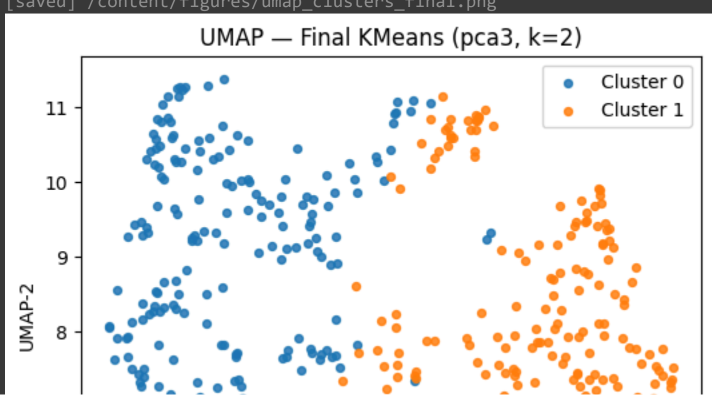
```
zlift.to_csv(FIG_DIR/"final_cluster_zlifts.csv")

print("\n[saved] /content/figures/final_kmeans_summary.csv")
print("[saved] /content/figures/final_cluster_profiles_mean.csv")
print("[saved] /content/figures/final_cluster_profiles_median.csv")
print("[saved] /content/figures/final_cluster_zlifts.csv")
print("[saved] pca_clusters_final.png, umap_clusters_final.png, tsne_clusters_final.png")
```

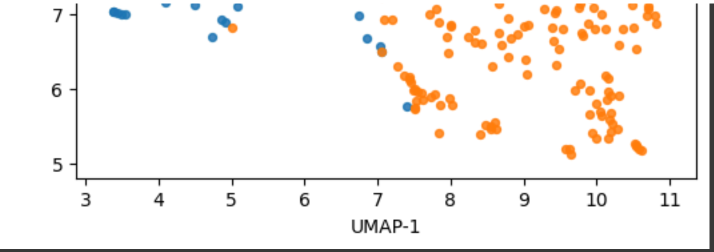
	space	k	silhouette	davies_bouldin	calinski_harabasz	min_frac
5	pca3	2	0.355921	1.124948	254.669324	0.427273
6	pca3	3	0.331336	1.095455	220.083553	0.186364
0	full	2	0.290328	1.351502	189.049797	0.427273
8	pca3	5	0.276463	1.214340	181.658542	0.104545
1	full	3	0.259199	1.344362	152.854740	0.184091
7	pca3	4	0.255262	1.221307	198.711715	0.156818
3	full	5	0.191585	1.484279	118.000499	0.122727
2	full	4	0.186577	1.547038	131.342939	0.138636

```
[FINAL] space=pca3  k=2  Sil=0.356  DBI=1.125  CH=254.7  min_cluster_frac=0.43
Cluster sizes: [188 252]
Stability ARI: mean=1.000, min=1.000  (>=0.90 is very stable)
Mean silhouette per cluster: {0: 0.30741962602225575, 1: 0.39210515734651896}
[saved] /content/figures/pca_clusters_final.png
```

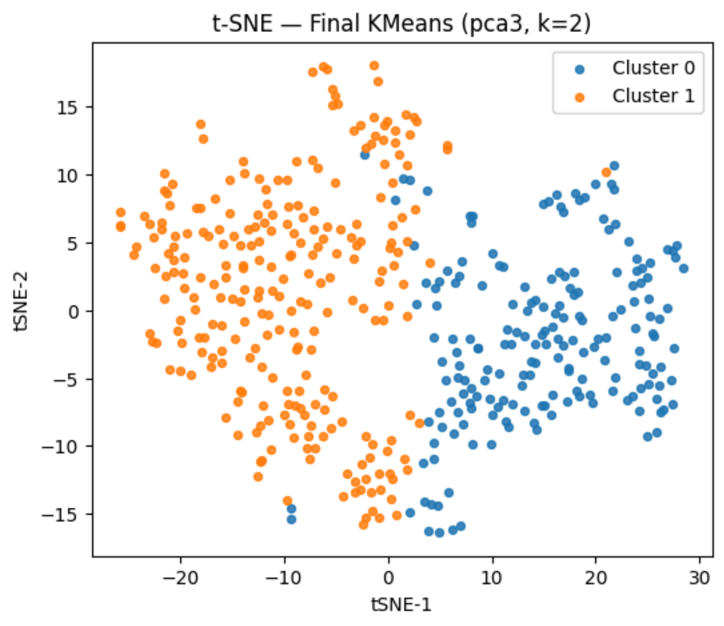



```
[saved] /content/figures/umap_clusters_final.png
```





[saved] /content/figures/tsne_clusters_final.png



Cluster Profiles — MEAN (original units)

cluster	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	9355.9	10346.4	14697.1	2222.5	6084.5	2177.4
1	13973.1	2401.8	2918.7	3705.7	491.9	1038.0

Cluster Profiles — MEDIAN (original units)

cluster	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	5406.5	7690.5	11527.0	1061.5	4495.0	1388.5
1	10347.5	1871.0	2406.0	2225.0	327.5	749.5

Cluster z-lifts vs overall mean (how many std above/below)

cluster	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	-0.21	0.62	0.71	-0.18	0.67	0.23
1	0.16	-0.46	-0.53	0.13	-0.50	-0.17

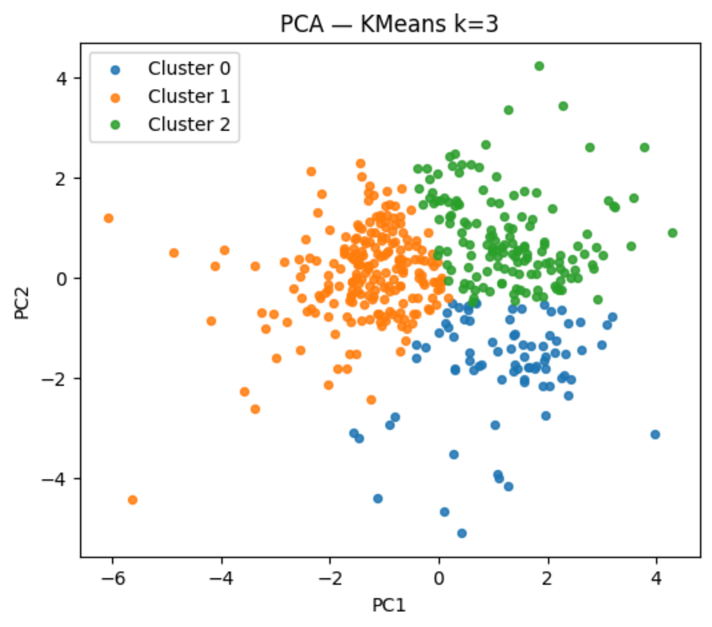
```
[saved] /content/figures/final_kmeans_summary.csv
[saved] /content/figures/final_cluster_profiles_mean.csv
[saved] /content/figures/final_cluster_profiles_median.csv
[saved] /content/figures/final_cluster_zlifts.csv
[saved] pca_clusters_final.png, umap_clusters_final.png, tsne_clusters_final.png
```

Next steps: Generate code with `profile_mean` | View recommended plots | New interactive sheet | Generate code with `profile_med` | View recommended plots | New interactive sheet | Generate code with `zlift` | Vie

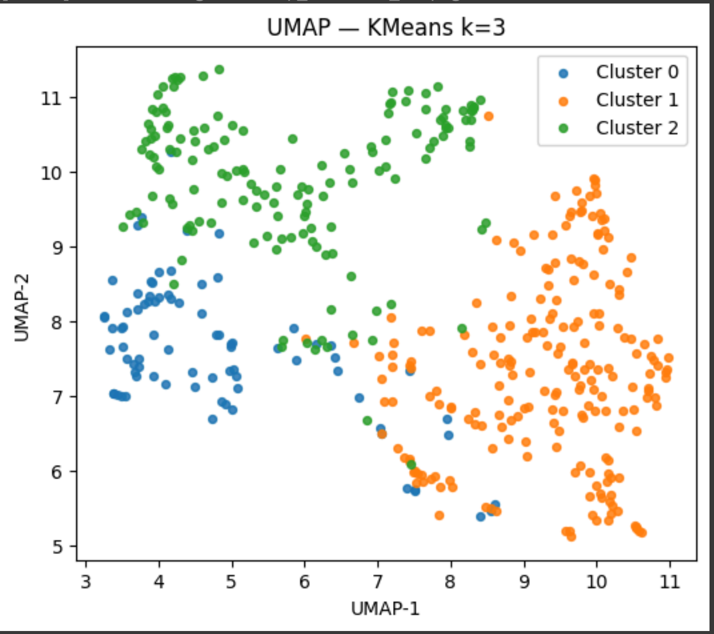
```python
# === Final choice: K-Means with k=3 on full standardized features ===
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples, davies_bouldin_score, calinski_harabasz_score
import numpy as np, pandas as pd, matplotlib.pyplot as plt

# 1) Fit
k_final = 3
km_k3 = KMeans(n_clusters=k_final, n_init=100, random_state=RANDOM_STATE)
labels_k3 = km_k3.fit_predict(X_scaled)

# 2) Metrics (all points)
sil_k3 = silhouette_score(X_scaled, labels_k3)
dbi_k3 = davies_bouldin_score(X_scaled, labels_k3)
ch_k3  = calinski_harabasz_score(X_scaled, labels_k3)

sizes = pd.Series(labels_k3).value_counts().sort_index()
min_frac = sizes.min() / len(labels_k3)
print(f"[KMeans k=3] Silhouette={sil_k3:.3f}  DBI={dbi_k3:.3f}  CH={ch_k3:.2f}")
print("Cluster sizes:", sizes.to_dict(), f"(min frac ~ {min_frac:.2f})")

# 3) Per-cluster silhouette (to catch any weak groups)
sil_samples = silhouette_samples(X_scaled, labels_k3)
sil_by_cluster = {c: float(np.mean(sil_samples[labels_k3==c])) for c in np.unique(labels_k3)}
print("Mean silhouette per cluster:", sil_by_cluster)

# 4) Attach to original df for profiling
df_k3 = df_orig.copy()
df_k3["cluster_k3"] = labels_k3

# 5) Visuals on PCA/UMAP/t-SNE
def scatter_2d(X2, labels, title, fname, xlabel, ylabel):
    plt.figure(figsize=(6,5))
    for c in np.unique(labels):
        m = (labels == c)
        plt.scatter(X2[m,0], X2[m,1], s=18, alpha=0.85, label=f"Cluster {int(c)}")
    plt.legend()
    plt.title(title); plt.xlabel(xlabel); plt.ylabel(ylabel)
    savefig(fname); plt.show()

scatter_2d(X_pca,  labels_k3, "PCA – KMeans k=3",  "pca_clusters_k3.png",  "PC1","PC2")
scatter_2d(X_umap, labels_k3, "UMAP – KMeans k=3", "umap_clusters_k3.png", "UMAP-1","UMAP-2")
scatter_2d(X_tsne, labels_k3, "t-SNE – KMeans k=3","tsne_clusters_k3.png","tSNE-1","tSNE-2")

# 6) Cluster profiles (original units) + z-lift vs overall
num_cols = numeric_cols  # from earlier
profile_mean_k3  = df_k3.groupby("cluster_k3")[num_cols].mean().round(1)
profile_median_k3= df_k3.groupby("cluster_k3")[num_cols].median().round(1)

overall_mean = df_orig[num_cols].mean()
overall_std  = df_orig[num_cols].std(ddof=0)
zlift_k3 = ((profile_mean_k3 - overall_mean) / overall_std).round(2)

print("\nCluster Profiles – MEAN (original units)")
display(profile_mean_k3)
print("\nCluster Profiles – MEDIAN (original units)")
display(profile_median_k3)
print("\nCluster z-lifts vs overall mean (std above/below)")
display(zlift_k3)

# 7) Save artifacts
pd.Series({
```

```
    "k": k_final,
    "silhouette": round(sil_k3,3),
    "davies_bouldin": round(dbi_k3,3),
    "calinski_harabasz": round(ch_k3,2),
    "min_cluster_frac": round(min_frac,2),
    **{f"cluster_{c}_sil": round(v,3) for c,v in sil_by_cluster.items()}
}).to_csv(FIG_DIR/"k3_summary.csv")

profile_mean_k3.to_csv(FIG_DIR/"k3_cluster_profiles_mean.csv")
profile_median_k3.to_csv(FIG_DIR/"k3_cluster_profiles_median.csv")
zlift_k3.to_csv(FIG_DIR/"k3_cluster_zlifts.csv")

print("\n[saved] /content/figures/k3_summary.csv")
print("[saved] /content/figures/k3_cluster_profiles_mean.csv")
print("[saved] /content/figures/k3_cluster_profiles_median.csv")
print("[saved] /content/figures/k3_cluster_zlifts.csv")
print("[saved] pca_clusters_k3.png, umap_clusters_k3.png, tsne_clusters_k3.png")
```

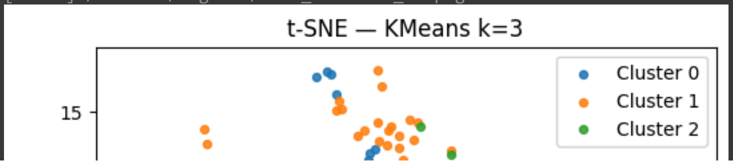
```
[KMeans k=3] Silhouette=0.259  DBI=1.344  CH=152.85
Cluster sizes: {0: 81, 1: 212, 2: 147} (min frac ~ 0.18)
Mean silhouette per cluster: {np.int32(0): 0.11858134500190261, np.int32(1): 0.2969292743925267, np.int32(2): 0.28226972091323943}
[saved] /content/figures/pca_clusters_k3.png
```

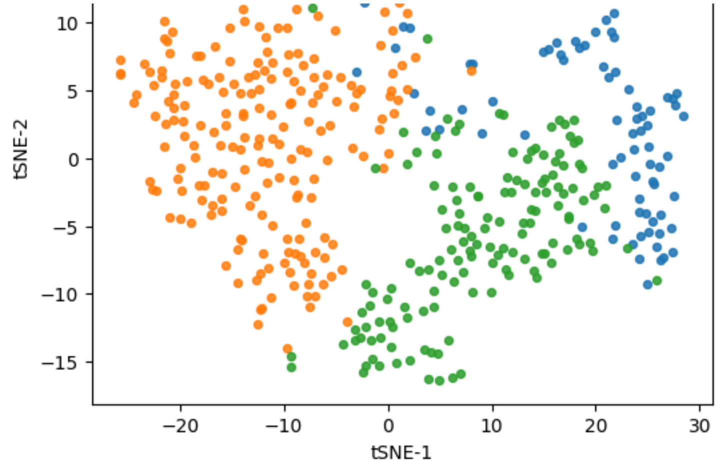



```
[saved] /content/figures/umap_clusters_k3.png
```




```
[saved] /content/figures/tsne_clusters_k3.png
```





Cluster Profiles — MEAN (original units)

cluster_k3	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	2869.2	7099.4	12459.0	607.2	5487.3	781.3
1	11992.6	1995.4	2496.9	3277.8	425.9	892.5
2	17042.8	10559.7	13333.6	4133.1	4987.0	2846.6

Cluster Profiles — MEDIAN (original units)

cluster_k3	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	1454.0	6264.0	10487.0	388.0	4196.0	436.0
1	9635.0	1598.5	2151.0	2151.0	274.5	686.0
2	12126.0	7184.0	9965.0	2005.0	3378.0	2005.0

Cluster z-lifts vs overall mean (std above/below)

cluster_k3	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicatessen
0	-0.72	0.18	0.47	-0.51	0.55	-0.26
1	-0.00	-0.52	-0.57	0.04	-0.52	-0.22
2	0.40	0.65	0.57	0.22	0.44	0.47

```
[saved] /content/figures/k3_summary.csv
[saved] /content/figures/k3_cluster_profiles_mean.csv
[saved] /content/figures/k3_cluster_profiles_median.csv
[saved] /content/figures/k3_cluster_zlifts.csv
[saved] pca_clusters_k3.png, umap_clusters_k3.png, tsne_clusters_k3.png
```

Next steps: Generate code with profile_mean_k3 | View recommended plots | New interactive sheet | Generate code with profile_median_k3 | View recommended plots | New interactive sheet | Generate code with zlift_

✅ Section 6 — Combined Conclusion & Final Clustering Choice

What we tried & what we learned

- **K-Means:** best balance of metrics + usability. Peak separation at **k=2** (Sil≈0.29), but **k=3** gives more actionable granularity without micro-clusters.
- **DBSCAN:** can capture non-linear shapes, but viable configs added a **noise bucket** (ops pain).
- **Agglomerative:** "best" metric case split off a **tiny outlier cluster** → not business-usable.

Decision: Proceed with **K-Means, k=3** for three clear, business-sized segments.

7) Model Selection Evidence & Recommendation

Goal: put contenders side-by-side with consistent metrics and make the final call.

Models compared

- **K-Means (k=3)** ← candidate we plan to ship
- K-Means (k=2) ← reference (highest silhouette in sweep)
- **DBSCAN (eps=1.5, min_samples=3)** ← best low-noise config from grid
- Agglomerative (average, k=2) and (ward, k=2) ← for completeness

Metrics reported

- **Silhouette** (higher=better) — note: DBSCAN computed on **non-noise** only
- **Davies–Bouldin (DBI)** (lower=better)
- **Calinski–Harabasz (CH)** (higher=better)
- **Min cluster fraction** (guards against tiny, non-actionable clusters)
- **Noise fraction** (DBSCAN only)

We'll save a single summary CSV for the report and a quick comparison chart.

```
# --- Section 7: Assemble side-by-side metrics & pick final ---
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
import numpy as np, pandas as pd, matplotlib.pyplot as plt

def km_eval(X, k, name):
    km = KMeans(n_clusters=k, n_init=100, random_state=RANDOM_STATE).fit(X)
    labels = km.labels_
    sizes = pd.Series(labels).value_counts().sort_index().values
    min_frac = sizes.min()/len(labels)
    return {
        "model": name,
        "silhouette": silhouette_score(X, labels),
        "davies_bouldin": davies_bouldin_score(X, labels),
        "calinski_harabasz": calinski_harabasz_score(X, labels),
        "min_cluster_frac": min_frac,
        "noise_frac": 0.0,
        "silhouette_on": "all",
    }
```

```
def agg_eval(X, k, linkage, name):
    ag = AgglomerativeClustering(n_clusters=k, linkage=linkage).fit(X)
    labels = ag.labels_
    sizes = pd.Series(labels).value_counts().sort_index().values
    min_frac = sizes.min()/len(labels)
    return {
        "model": name,
        "silhouette": silhouette_score(X, labels),
        "davies_bouldin": davies_bouldin_score(X, labels),
        "calinski_harabasz": calinski_harabasz_score(X, labels),
        "min_cluster_frac": min_frac,
        "noise_frac": 0.0,
        "silhouette_on": "all",
    }

def dbscan_eval(X, eps, ms, name):
    db = DBSCAN(eps=eps, min_samples=ms).fit(X)
    labels = db.labels_
    noise_frac = float((labels==-1).mean())
    # metrics on non-noise points if >=2 clusters present
    mask = labels!=-1
    if len(np.unique(labels[mask])) >= 2 and mask.sum() > 1:
        sil = silhouette_score(X[mask], labels[mask])
        dbi = davies_bouldin_score(X[mask], labels[mask])
        ch  = calinski_harabasz_score(X[mask], labels[mask])
        # min cluster frac among non-noise clusters (relative to ALL points)
        sizes = pd.Series(labels[mask]).value_counts().sort_index().values
```