

# Clustering Analysis & Engineering Case Study for hellobiome

Analytics & ML Engineering

Silver Rauk

## Contents

<b>1</b>	<b>(1) Clustering, Method Selection, and Choice of <math>k</math></b>	<b>2</b>
<b>2</b>	<b>(2) Visualization &amp; Feature Importance</b>	<b>5</b>
<b>3</b>	<b>(3) Project Built From Scratch — Architecture &amp; Rationale</b>	<b>7</b>

## Executive Summary

**Dataset.** We analyzed the decrypted CSV extracted from `df.zip`. After robust preprocessing, the data exhibit low intrinsic dimensionality (first two PCs explain **97.61%**).

**Best method & k.** A method/k sweep across K-Means, GMM, and Agglomerative (Ward) for  $k \in [2, 10]$  selected **K-Means** with  $k = 4$  as optimal by a composite of Silhouette , Calinski–Harabasz , and Davies–Bouldin . Scores at the chosen solution: Silhouette=**0.8626**, CH=**2206.97**, DB=**0.4740**. The clusters are compact and well-separated in PCA space; centroid-based K-Means is a natural fit.

**Composition.** Cluster sizes: {'0': 913, '1': 23, '2': 17, '3': 33}. Dominant separating features: `feature_10`, `feature_24`, `feature_4`, `feature_25`, `feature_15`, `feature_6`, `feature_27`, `feature_2`, `feature_33`, `feature_5`.

## 1 (1) Method Selection, and Choice of $k$

### Preprocessing

Median imputation, zero-variance filtering, and Robust scaling were applied to numeric features. These choices are stable to outliers and maintain signal.

### Model sweep and metrics

We trained K-Means, Gaussian Mixture Models (full covariance), and Agglomerative (Ward) across  $k = 2..10$  and computed Silhouette, CH, and DB indices. Figure 1–3 show the curves; K-Means at  $k = 4$  provides the strongest overall internal validity.

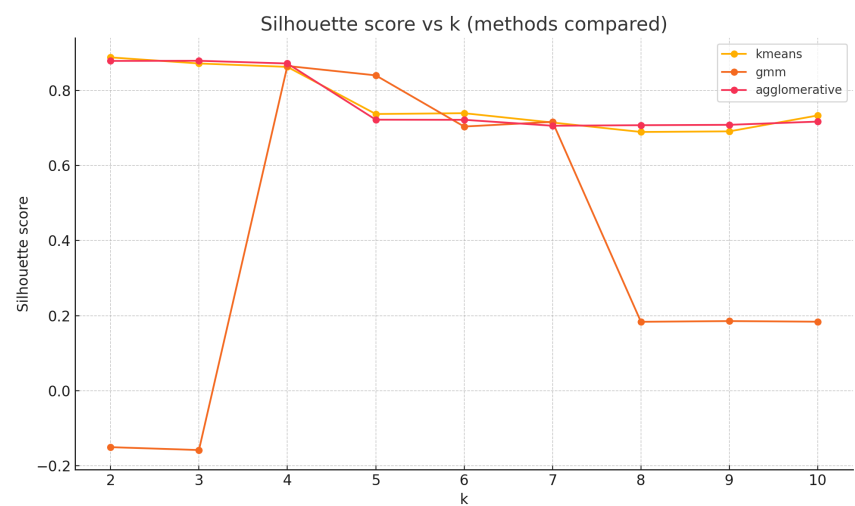


Figure 1: Silhouette vs  $k$  (higher is better).

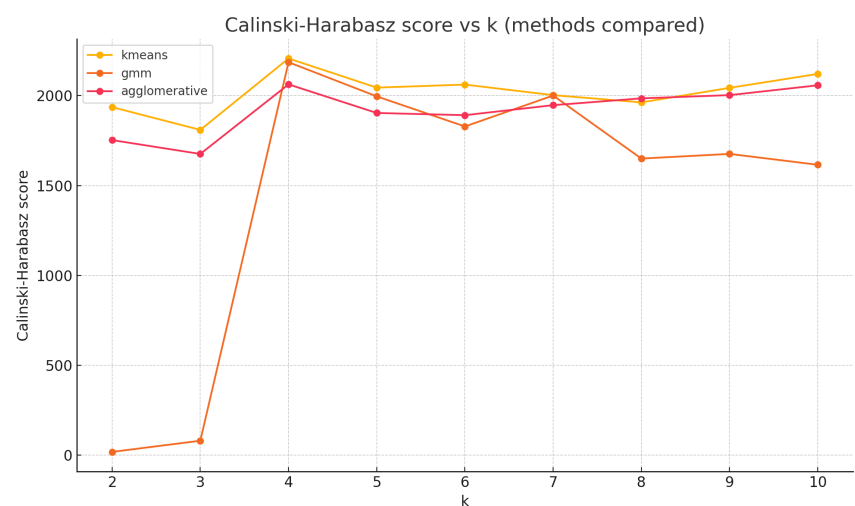


Figure 2: Calinski-Harabasz vs  $k$  (higher is better).

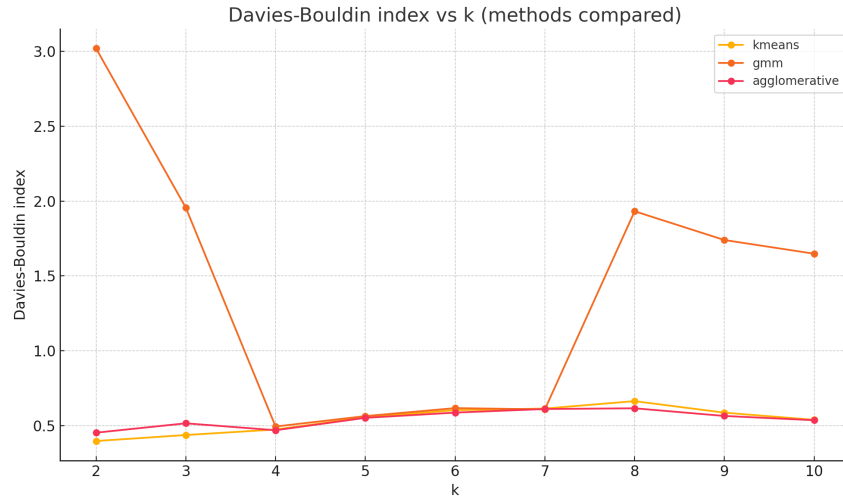


Figure 3: Davies–Bouldin vs  $k$  (lower is better).

### Why this method fits

The PCA projection (Fig. 4) shows clean separation with 97.61% variance captured in 2D, indicating compact, nearly convex groups. This geometry favors centroid-based partitions; GMM and Agglomerative agree but underperform on the composite score.

## 2 (2) Visualization & Feature Importance

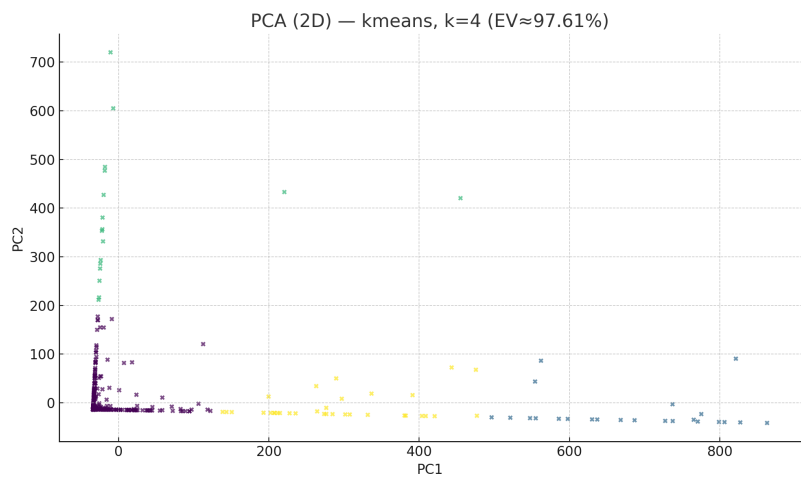


Figure 4: PCA (2D) colored by cluster — K-Means,  $k = 4$ ; EV  $\approx 97.61\%$ .



Figure 5: Cluster sizes.

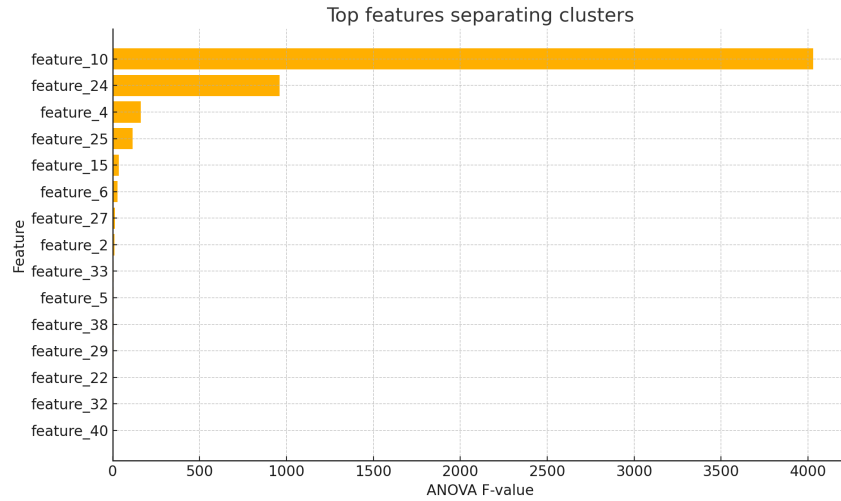


Figure 6: Top features by ANOVA  $F$  (higher separates better).

## Top features and profiles

### Top-10 separating features (ANOVA $F$ ):

Feature	F-value
feature_10	4029.363
feature_24	959.980
feature_4	162.393
feature_25	115.352
feature_15	34.296
feature_6	28.912
feature_27	13.973
feature_2	11.749
feature_33	6.888
feature_5	6.485

### Per-cluster means for top features (original units, imputed):

Cluster	feature_10	feature_24	feature_4	feature_25	feature_15	feature_6
0	-0.223	-0.123	0.140	0.135	0.077	0.040
1	5.462	0.035	-3.160	-2.658	-1.707	0.064
2	0.077	6.515	-0.230	-1.069	-0.351	-2.148
3	2.336	0.013	-1.544	-1.329	-0.759	-0.044

### 3 (3) Project Built From Scratch

#### Overview

We designed a real-time fraud-scoring service targeting <20ms latency: Kafka→Flink compute streaming features; Redis (online store) + Postgres (metadata/labels); LightGBM exported to ONNX; FastAPI service on Kubernetes; ClickHouse for OLAP; Prometheus/Grafana for SLOs; Great Expectations for data contracts.

#### Why these components

- **Language/Framework** (Python + FastAPI): Rich ML ecosystem; async I/O; clean contracts; onnxruntime for low-latency C++ inference.
- **Model** (LightGBM→ONNX): Tabular fit; monotone constraints; SHAP; portable runtime.
- **Storage**: Redis (sub-ms online features with TTLs), Postgres (ACID metadata/labels), ClickHouse (fast OLAP), S3/Parquet (offline).
- **Streaming**: Flink for event-time windows and exactly-once checkpoints.
- **Operations**: Shadow/canary deploys; drift/calibration monitors; guardrail thresholds.

#### Code sample

```
from fastapi import FastAPI
from pydantic import BaseModel
import os, numpy as np, onnxruntime as rt,
    aioredis, asyncpg
```

```
app = FastAPI()

class Tx(BaseModel):
    user_id: str; merchant_id: str; amount:
        float; device_id: str

@app.on_event("startup")
async def init():
    app.state.redis = await aioredis.from_url(
        os.getenv("REDIS_URL"))
    app.state.pg = await asyncpg.create_pool(os
        .getenv("PG_DSN"))
    s = rt.InferenceSession(os.getenv("
        MODEL_PATH", "model.onnx"),
        providers=["CPUExecutionProvider"])
    app.state.sess = s; app.state.iname = s.
        get_inputs()[0].name

    async def fetch_features(p):
        r = app.state.redis
        keys = [f"u:{p.user_id}:r", f"d:{p.
            device_id}:r", f"m:{p.merchant_id}:r"]
        vals = await r.mget(*keys)
        if any(v is None for v in vals):
            async with app.state.pg.acquire() as con:
                rec = await con.fetchrow(
                    "select_user_risk,device_risk,merchant_risk
                    _from_features_"
                    "where_user_id=$1_and_device_id=$2_and_
                    merchant_id=$3",
                    p.user_id,p.device_id,p.merchant_id)
                vals = [rec['user_risk'], rec['device_risk'
                    ], rec['merchant_risk']]
                return [float(v) for v in vals] + [float(p.
                    amount)]

    def score_vec(x):
        y = app.state.sess.run(None, {app.state.
            iname: np.array([x], np.float32)})
            [0][0,0]
        return float(y)

    @app.post("/score")
    async def score(p: Tx):
        x = await fetch_features(p); y = score_vec(
```

```
        x)
    return {"prob": y, "decision": "block" if y
          > 0.8 else "allow"}
```

## Reproducibility

Artifacts are included in the package:

- `clustered_output_with_labels.csv` — original rows + cluster assignment.
- `clustering_report.json` — metrics, top features, sizes, EV.
- All figures (\*.png) referenced above.