Customer Segmentation with **API-only** Data (No CSVs)

Source API: DummyJSON — `/users` and `/carts`

Notebook Author: Rishi Koushik Sridharan

Plan

1. Fetch data via HTTP (requests), in-memory only
2. Feature engineer customer-level metrics (frequency, spend, basket metrics, discount behavior, product diversity)
3. PCA for multivariate exploration
4. KMeans & GMM clustering
5. Hyperparameter tuning: Silhouette, Davies–Bouldin, BIC
6. Cluster profiling + optional heuristic names

0) Setup & API Fetch (in-memory)

```
import requests, math
import pandas as pd
import numpy as np

BASE = "https://dummyjson.com"

def fetch_all(endpoint, page_size=100, key=None):
    url = BASE + endpoint
    r0 = requests.get(url, params={"limit": 1, "skip": 0})
    r0.raise_for_status()
    data0 = r0.json()
    if key is None:
        key = [k for k,v in data0.items() if isinstance(v, list)][0]
    total = data0.get("total", len(data0.get(key, [])))
    pages = math.ceil(total / page_size)
    items = []
    for p in range(pages):
        r = requests.get(url, params={"limit": page_size, "skip": p*page_size})
        r.raise_for_status()
        j = r.json()
        items.extend(j.get(key, []))
    return items
```

```
users_raw = fetch_all("/users", key="users")
carts_raw = fetch_all("/carts", key="carts")

users = pd.json_normalize(users_raw)
carts = pd.json_normalize(carts_raw, sep="_")

users.shape, carts.shape
```

```
((208, 52), (50, 7))
```

1) Feature Engineering (Customer-level)

```
# Cart-level features
carts["avg_item_price"] = carts["total"] / carts["totalQuantity"]
carts["discount_rate"] = 1 - (carts["discountedTotal"] / carts["total"]).replac

def unique_product_ids(prod_list):
    try:
        return len({p.get("id") for p in prod_list})
    except Exception:
        return np.nan

carts["n_unique_products"] = carts["products"].apply(unique_product_ids)

cust = carts.groupby("userId").agg(
    n_orders=("id", "nunique"),
    total_spend=("total", "sum"),
    total_spend_discounted=("discountedTotal", "sum"),
    mean_total=("total", "mean"),
    mean_discounted_total=("discountedTotal", "mean"),
    mean_total_products=("totalProducts", "mean"),
    mean_total_qty=("totalQuantity", "mean"),
    mean_unique_products=("n_unique_products", "mean"),
    mean_avg_item_price=("avg_item_price", "mean"),
    mean_discount_rate=("discount_rate", "mean"),
).reset_index()

demo_cols = [
    "id","age","gender","email","phone",
    "address.city","address.state","address.country"
]
users_ = users[demo_cols].rename(columns={
    "id":"userId",
    "address.city":"address_city",
    "address.state":"address_state",
    "address.country":"address_country"
```

```
})
cust = cust.merge(users_, on="userId", how="left")

cust["gender_enc"] = cust["gender"].map({"male":0, "female":1}).fillna(-1)

top_countries = cust["address_country"].value_counts().head(5).index.tolist()
for c in top_countries:
    cust[f"country_{c}"] = (cust["address_country"] == c).astype(int)

cust.head()
```

	userId	n_orders	total_spend	total_spend_discounted	mean_total	mean_disco
0	6	1	1749.90	1594.33	1749.90	
1	11	1	11741.31	10940.95	11741.31	
2	15	1	3359.79	3262.64	3359.79	
3	20	1	460.87	413.86	460.87	
4	23	2	16143.72	13073.85	8071.86	

Next steps: Generate code with cust New interactive sheet

2) Preprocessing

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

num_feats = [
    "n_orders","total_spend","total_spend_discounted",
```

```
    "mean_total","mean_discounted_total",
    "mean_total_products","mean_total_qty",
    "mean_unique_products","mean_avg_item_price","mean_discount_rate",
    "age","gender_enc"
] + [c for c in cust.columns if c.startswith("country_")]

X = cust[num_feats].copy()
X = X.replace([np.inf, -np.inf], np.nan)
imp = SimpleImputer(strategy="median")
X_imp = imp.fit_transform(X)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imp)

X.shape, len(num_feats)
```

```
((45, 13), 13)
```

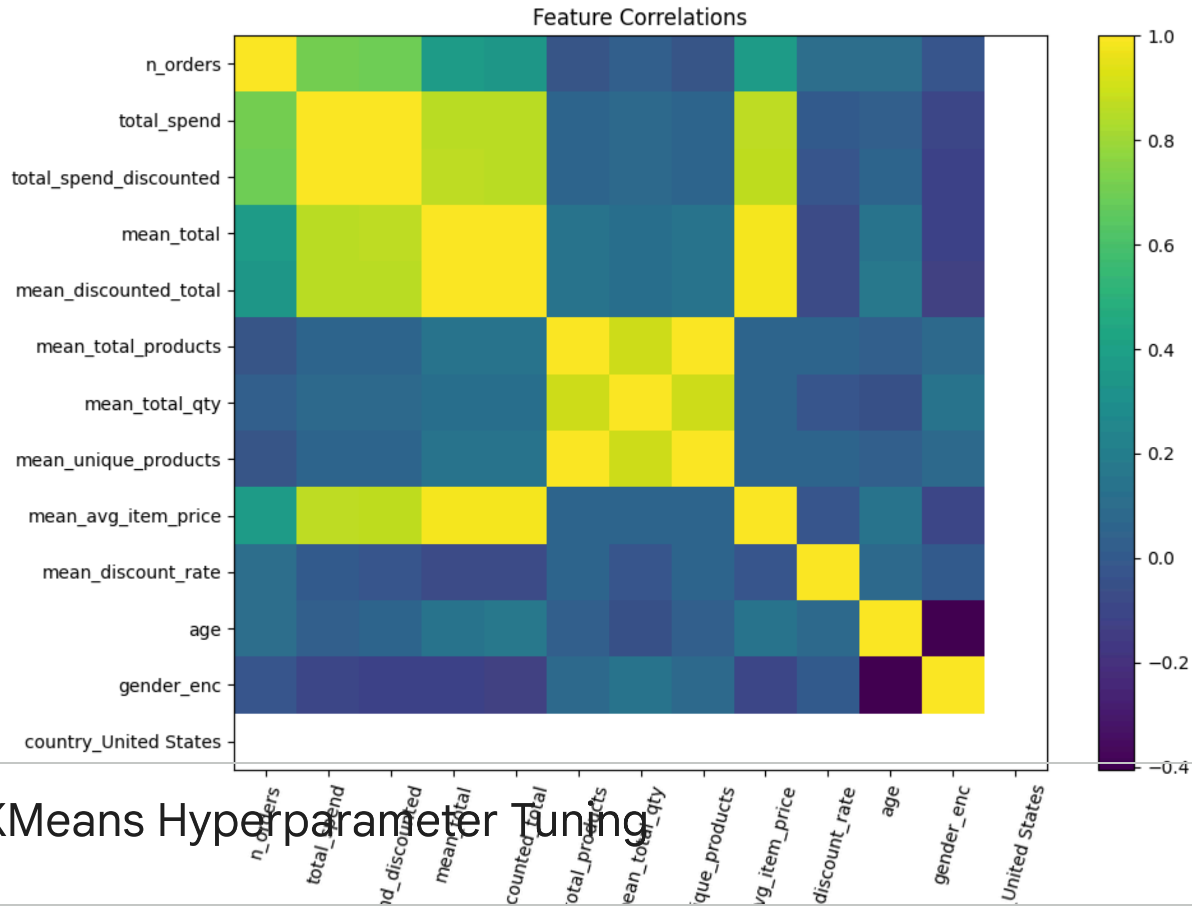
3) Multivariate Analysis (Correlation + PCA)

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

corr = pd.DataFrame(X_imp, columns=num_feats).corr()
plt.figure(figsize=(10,8))
plt.imshow(corr, aspect="auto")
plt.colorbar()
plt.xticks(range(len(num_feats)), num_feats, rotation=75)
plt.yticks(range(len(num_feats)), num_feats)
plt.title("Feature Correlations")
plt.tight_layout()
plt.show()

pca = PCA(n_components=3, random_state=42)
X_pca = pca.fit_transform(X_scaled)
print("Explained variance ratio:", pca.explained_variance_ratio_)

plt.figure(figsize=(7,6))
plt.scatter(X_pca[:,0], X_pca[:,1], s=8, alpha=0.5)
plt.xlabel("PC1"); plt.ylabel("PC2"); plt.title("PCA Scatter (PC1 vs PC2)")
plt.show()
```



4) KMeans Hyperparameter Tuning

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

k_range = range(2, 11)
km_scores = []
for k in k_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    labels = km.fit_predict(X_scaled)
    sil = silhouette_score(X_scaled, labels)
    db  = davies_bouldin_score(X_scaled, labels)
    km_scores.append({"k":k, "silhouette":sil, "davies_bouldin":db})

km_df = pd.DataFrame(km_scores)
display(km_df)

plt.figure(figsize=(7,4))
plt.plot(km_df["k"], km_df["silhouette"], marker="o")
plt.xlabel("k"); plt.ylabel("Silhouette"); plt.title("KMeans: Silhouette vs k")
plt.show()

plt.figure(figsize=(7,4))
plt.plot(km_df["k"], km_df["davies_bouldin"], marker="o")
plt.xlabel("k"); plt.ylabel("Davies-Bouldin (lower better)"); plt.title("KMeans
```

```
plt.show()
```

Next steps: Generate code with km_df New interactive sheet

	k	silhouette	davies_bouldin
0	2	0.459602	1.002061
1	3	0.275252	1.174719
2	4	0.270789	0.970392
3	5	0.234652	1.112231
4	6	0.255647	1.135959
8	10	0.254505	0.957122

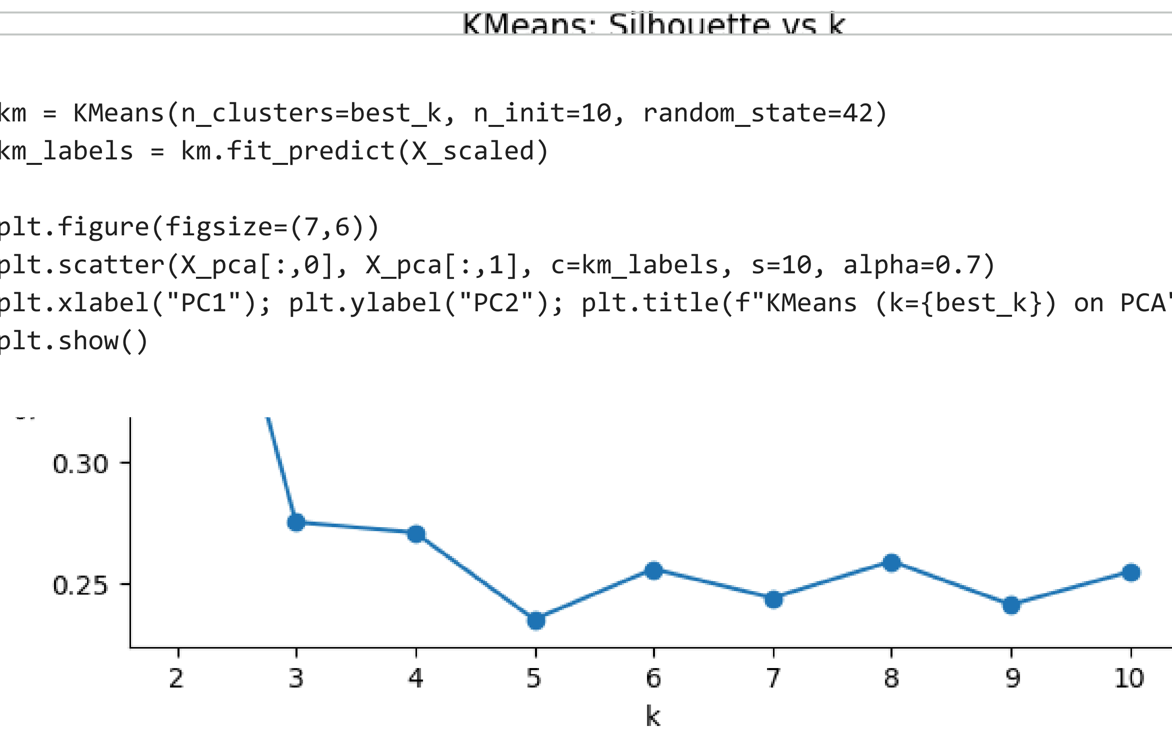
Fit Best KMeans (by Silhouette)

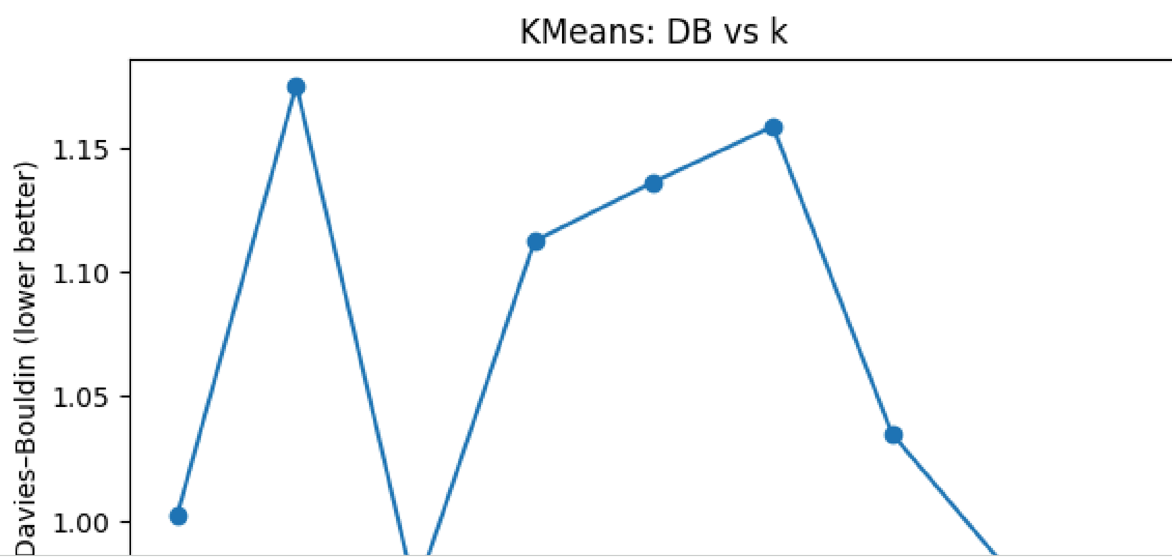
```
best_k = int(km_df.sort_values("silhouette", ascending=False).iloc[0]["k"])
best_k
```

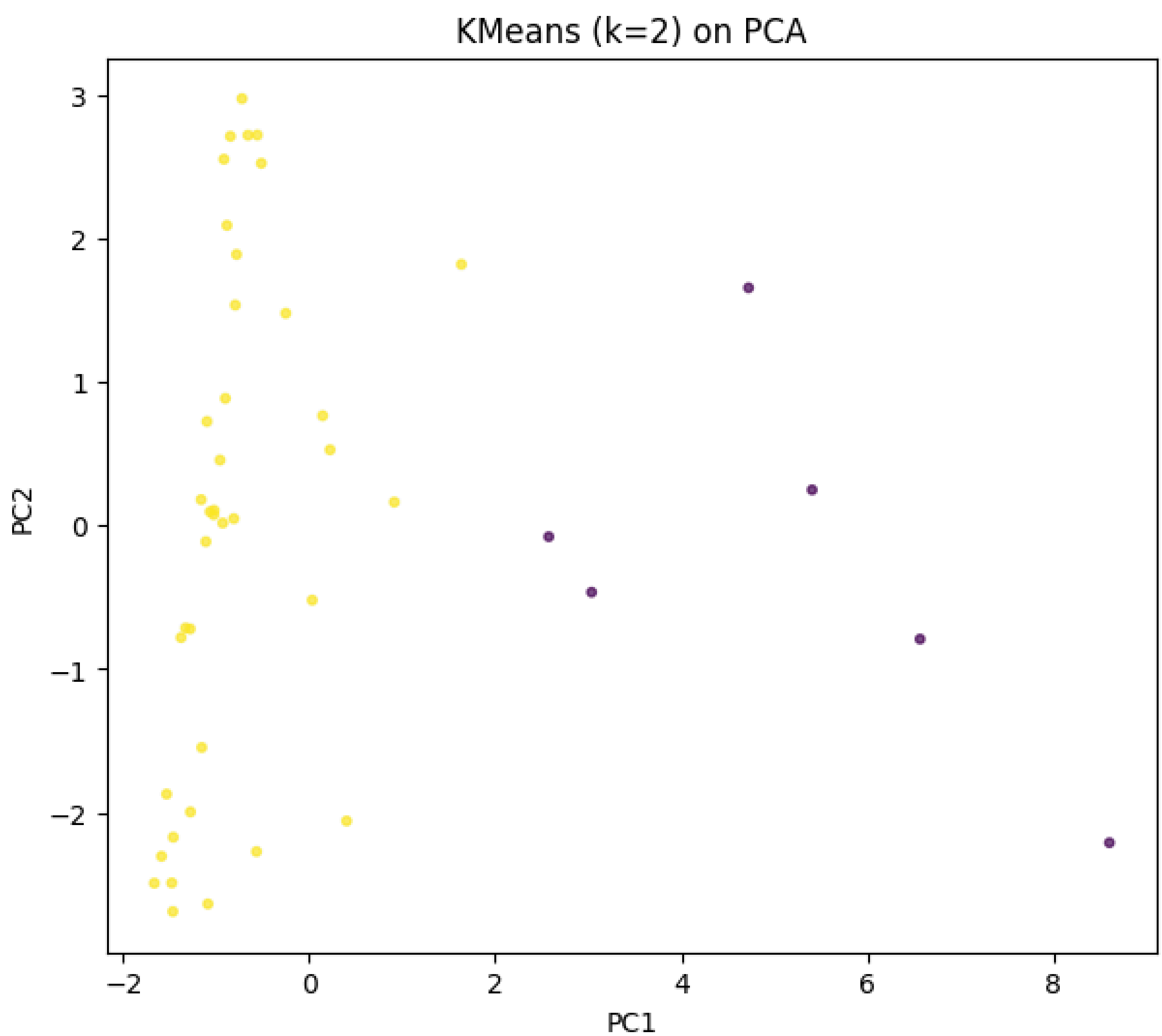
2

```
km = KMeans(n_clusters=best_k, n_init=10, random_state=42)
km_labels = km.fit_predict(X_scaled)

plt.figure(figsize=(7,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=km_labels, s=10, alpha=0.7)
plt.xlabel("PC1"); plt.ylabel("PC2"); plt.title(f"KMeans (k={best_k}) on PCA")
plt.show()
```







5) Gaussian Mixture Model (GMM) + BIC

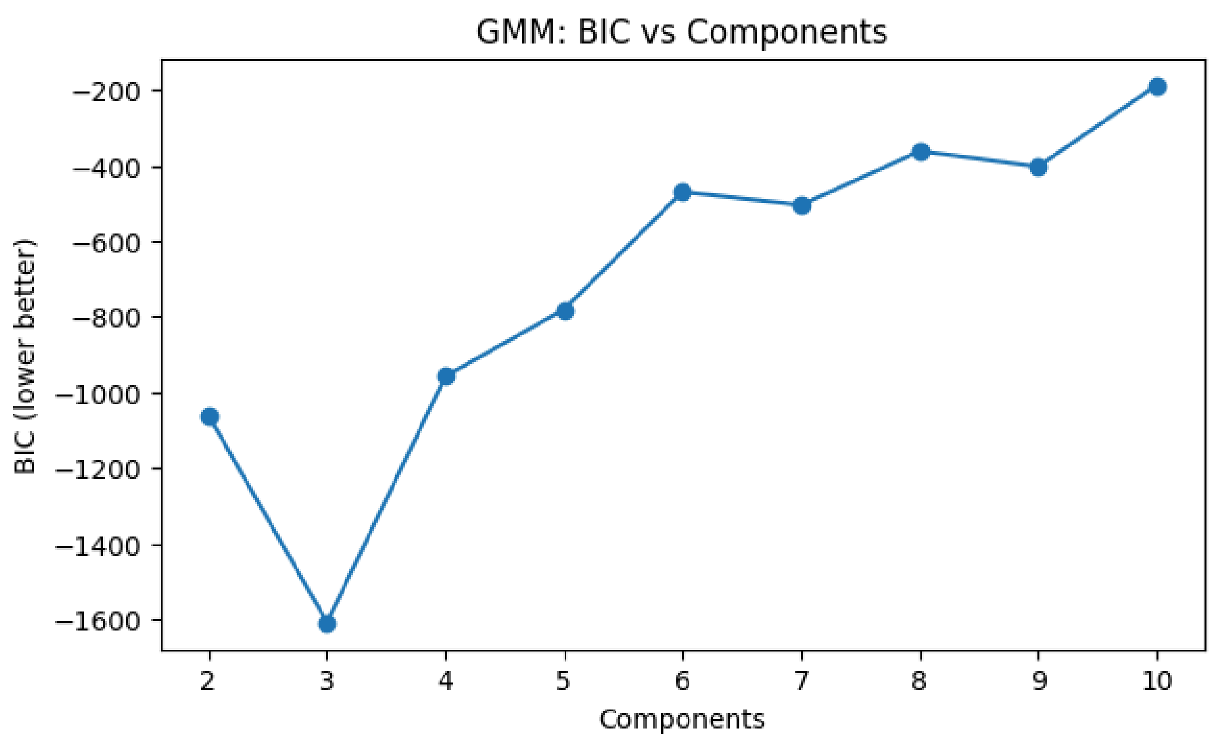
```
from sklearn.mixture import GaussianMixture

g_range = range(2, 11)
gmm_scores = []
for g in g_range:
    gmm = GaussianMixture(n_components=g, covariance_type="full", random_state=
    gmm.fit(X_scaled)
    labels = gmm.predict(X_scaled)
    bic = gmm.bic(X_scaled)
    sil = silhouette_score(X_scaled, labels)
    gmm_scores.append({"components":g, "BIC":bic, "silhouette":sil})

gmm_df = pd.DataFrame(gmm_scores)
display(gmm_df.sort_values("BIC"))
```

```
plt.figure(figsize=(7,4))
plt.plot(gmm_df["components"], gmm_df["BIC"], marker="o")
plt.xlabel("Components"); plt.ylabel("BIC (lower better)"); plt.title("GMM: BIC
plt.show()
```

	components	BIC	silhouette
1	3	-1609.212717	0.126763
0	2	-1061.456351	0.459602
2	4	-957.553734	0.234104
3	5	-780.913547	0.239528
5	7	-504.612330	0.195544
4	6	-469.443290	0.232193
7	9	-402.786204	0.232059
6	8	-362.173862	0.221488
8	10	-187.614779	0.245431

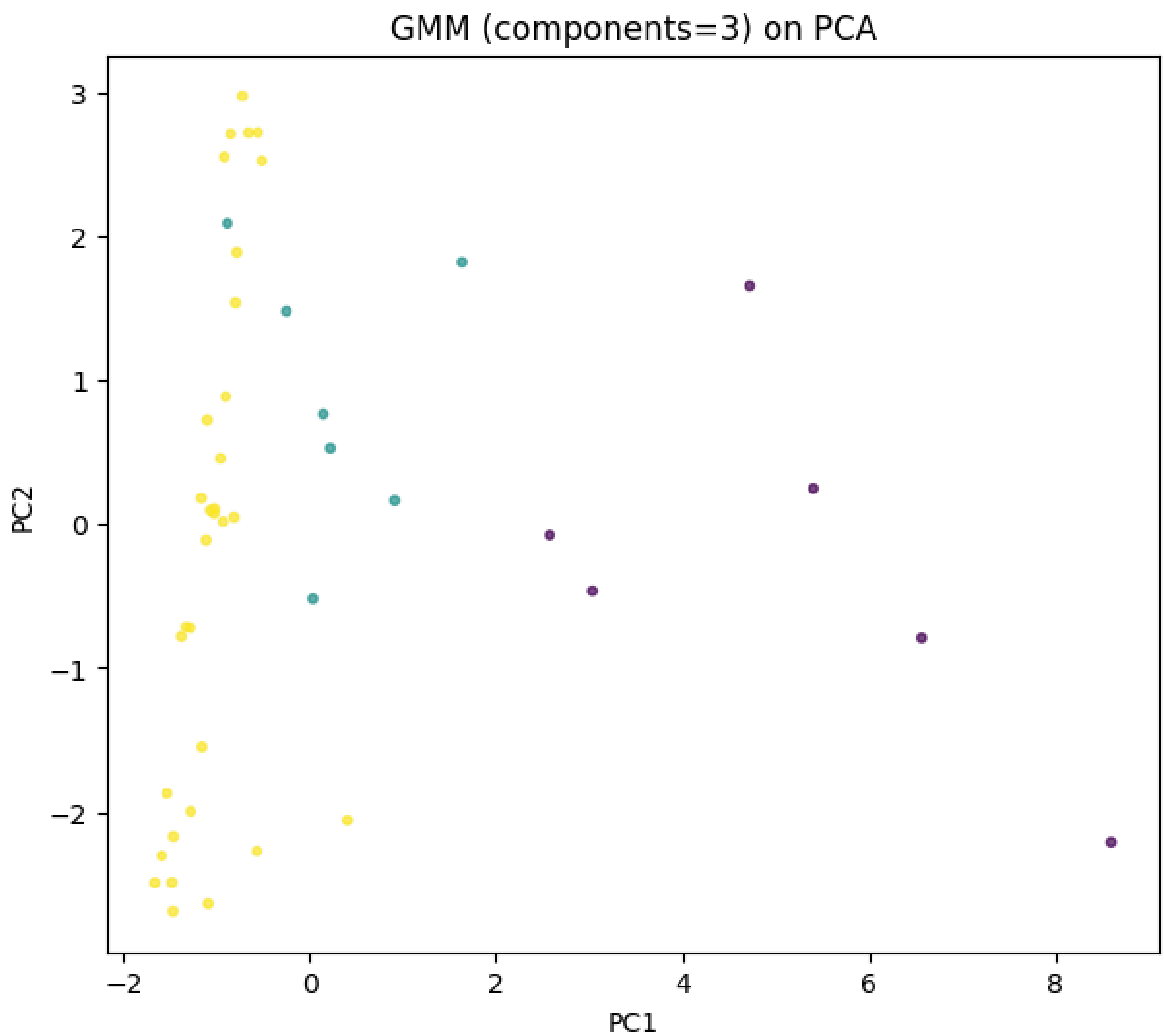



```
best_g = int(gmm_df.sort_values("BIC", ascending=True).iloc[0]["components"])
best_g
```

```
3
```

```
gmm = GaussianMixture(n_components=best_g, covariance_type="full", random_state
gmm_labels = gmm.fit_predict(X_scaled)

plt.figure(figsize=(7,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=gmm_labels, s=10, alpha=0.7)
plt.xlabel("PC1"); plt.ylabel("PC2"); plt.title(f"GMM (components={best_g}) on
plt.show()
```



6) Cluster Profiling

```
profile_cols = list(num_feats)

def profile(Ximp, labels, cols, name):
    d = pd.DataFrame(Ximp, columns=cols).copy()
    d["__label__"] = labels
```

```
    prof = d.groupby("__label__")[cols].mean().round(2)
    prof["n_customers"] = d.groupby("__label__").size()
    print(f"=== {name} profile ===")
    display(prof)
    return prof

km_profile = profile(X_imp, km_labels, profile_cols, "KMeans")
gmm_profile = profile(X_imp, gmm_labels, profile_cols, "GMM")

# Normalized line chart for KMeans
norm = (km_profile[profile_cols] - km_profile[profile_cols].min()) / (km_profil
plt.figure(figsize=(10,5))
for i, row in norm.iterrows():
    plt.plot(range(len(profile_cols)), row.values, marker="o", label=f"Cluster
plt.xticks(range(len(profile_cols)), profile_cols, rotation=75)
plt.title("KMeans Cluster Profiles (Normalized)")
plt.legend()
plt.tight_layout()
plt.show()
```