1. Project Overview

Objective: Segment retail customers into meaningful groups using transactional data, enabling targeted marketing, retention strategies, and revenue optimization.

Dataset: Online Retail Transaction Data (Invoice-level)

Approach:

- Rigorous data cleaning
- Customer-level feature engineering (RFM + behavioral)
- Proper scaling and validation
- Unsupervised learning (KMeans)
- Statistical and business interpretation*

2. Imports & Configuration

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, adjusted_rand_score

import hdbscan

RANDOM_STATE = 53
np.random.seed(RANDOM_STATE)
```

3. Data Loading

```python
data = pd.read_csv('customer_segmentation.csv', encoding='latin-1')
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
```

4. Data Cleaning

```python
# Remove cancelled invoices
data = data[~data["InvoiceNo"].astype(str).str.startswith("C")]

# Remove missing customers
data = data.dropna(subset=["CustomerID"])

# Remove invalid quantities and prices
data = data[(data["Quantity"] > 0) & (data["UnitPrice"] > 0)]

# Fix product descriptions (StockCode → mode Description)
desc_map = (
    data.dropna(subset=["Description"])
        .groupby("StockCode")["Description"]
        .agg(lambda x: x.value_counts().index[0])
)

data["Description"] = data["Description"].fillna(
    data["StockCode"].map(desc_map)
)

# Remove non-product codes
noise_codes = ["POST", "D", "M", "DOT", "BANK CHARGES"]
data = data[~data["StockCode"].isin(noise_codes)]

# Monetary value
data["TotalPrice"] = data["Quantity"] * data["UnitPrice"]
```

5. Feature Engineering

5.1 RFM Features

```python
snapshot_date = data["InvoiceDate"].max() + pd.Timedelta(days=1)

rfm = data.groupby("CustomerID").agg({
    "InvoiceDate": lambda x: (snapshot_date - x.max()).days,
    "InvoiceNo": "nunique",
    "TotalPrice": "sum"
}).reset_index()

rfm.columns = ["CustomerID", "Recency", "Frequency", "Monetary"]
```

5.2 Behavioral Features

```python
behavior = data.groupby("CustomerID").agg({
    "Quantity": "sum",
    "StockCode": "nunique"
}).reset_index()

behavior.columns = ["CustomerID", "TotalQuantity", "UniqueProducts"]

customer_df = rfm.merge(behavior, on="CustomerID")
```

6. Feature Scaling

```python
features = [
    "Recency",
    "Frequency",
    "Monetary",
    "TotalQuantity",
    "UniqueProducts"
]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(customer_df[features])
```

7. Baseline Segmentation

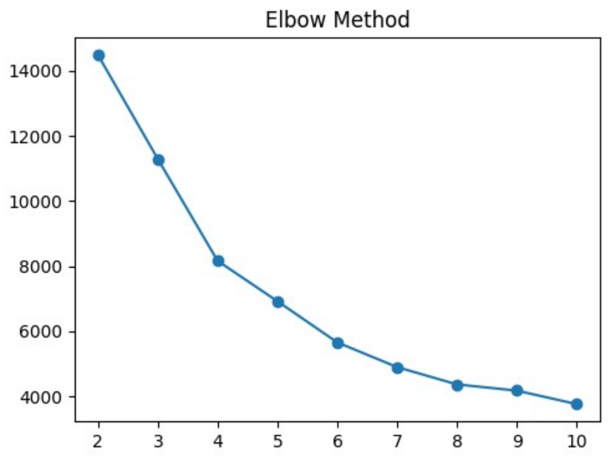
7.1 Model Selection

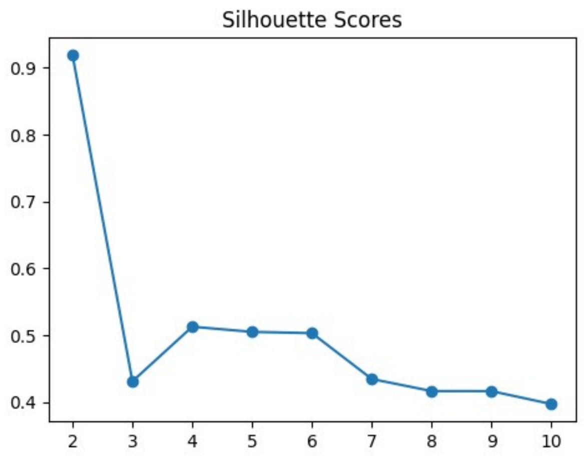
```python
inertia = []
sil_scores = []

for k in range(2, 11):
    km = KMeans(n_clusters=k, random_state=RANDOM_STATE)
    labels = km.fit_predict(X_scaled)
    inertia.append(km.inertia_)
    sil_scores.append(silhouette_score(X_scaled, labels))

plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(range(2,11), inertia, marker="o")
plt.title("Elbow Method")

plt.subplot(1,2,2)
plt.plot(range(2,11), sil_scores, marker="o")
plt.title("Silhouette Scores")
plt.show()
```





7.2 Final K-Means

```
kmeans = KMeans(n_clusters=4, random_state=RANDOM_STATE)
customer_df["KMeans_Cluster"] = kmeans.fit_predict(X_scaled)

silhouette_score(X_scaled, customer_df["KMeans_Cluster"])
```

```
0.512926083590546
```

8 . Cluster Stabillity Testing

8.1 Bootstrap Stability

```
n_bootstraps = 30
ari_scores = []

base_labels = customer_df["KMeans_Cluster"].values

for i in range(n_bootstraps):
    idx = np.random.choice(len(X_scaled), len(X_scaled), replace=True)
    X_sample = X_scaled[idx]

    km = KMeans(n_clusters=4, random_state=i)
    labels = km.fit_predict(X_sample)

    ari = adjusted_rand_score(base_labels[idx], labels)
    ari_scores.append(ari)

print(f"Mean ARI: {np.mean(ari_scores):.3f}")
print(f"Std ARI : {np.std(ari_scores):.3f}")
```

```
Mean ARI: 0.737
Std ARI : 0.305
```

Interpretation rule (state this in report):

- ARI > 0.75 → highly stable
- 0.5–0.75 → acceptable
- < 0.5 → unreliable

9 . Alternative Model I: Gaussian Mixture Model (GMM)

```
gmm = GaussianMixture(
    n_components=4,
    covariance_type="full",
    random_state=RANDOM_STATE
)

customer_df["GMM_Cluster"] = gmm.fit_predict(X_scaled)

# Soft membership confidence
gmm_probs = gmm.predict_proba(X_scaled)
customer_df["GMM_MaxProb"] = gmm_probs.max(axis=1)

silhouette_score(X_scaled, customer_df["GMM_Cluster"])
```

```
0.107471585490664
```

10. Alternative Model II: HDBSCAN (Density-Based)

```
hdb = hdbscan.HDBSCAN(
    min_cluster_size=30,
    min_samples=15
)

customer_df["HDBSCAN_Cluster"] = hdb.fit_predict(X_scaled)

customer_df["HDBSCAN_Cluster"].value_counts()
```

{"columns":
[{"name":"HDBSCAN_Cluster","rawType":"int64","type":"integer"},
{"name":"count","rawType":"int64","type":"integer"}],"ref":"dbcfabbb-b015-48f8-9df3-c84661b7f922","rows":[["-1","2818"],["4","568"],
["5","414"],["2","328"],["0","85"],["3","82"],["1","39"]],"shape":
{"columns":1,"rows":7}}

- `-1` → noise / unsegmentable customers
- Indicates niche or anomalous buyers

11 . Comparative Analysis

```
comparison = customer_df.groupby("KMeans_Cluster")[features].mean()
comparison
```

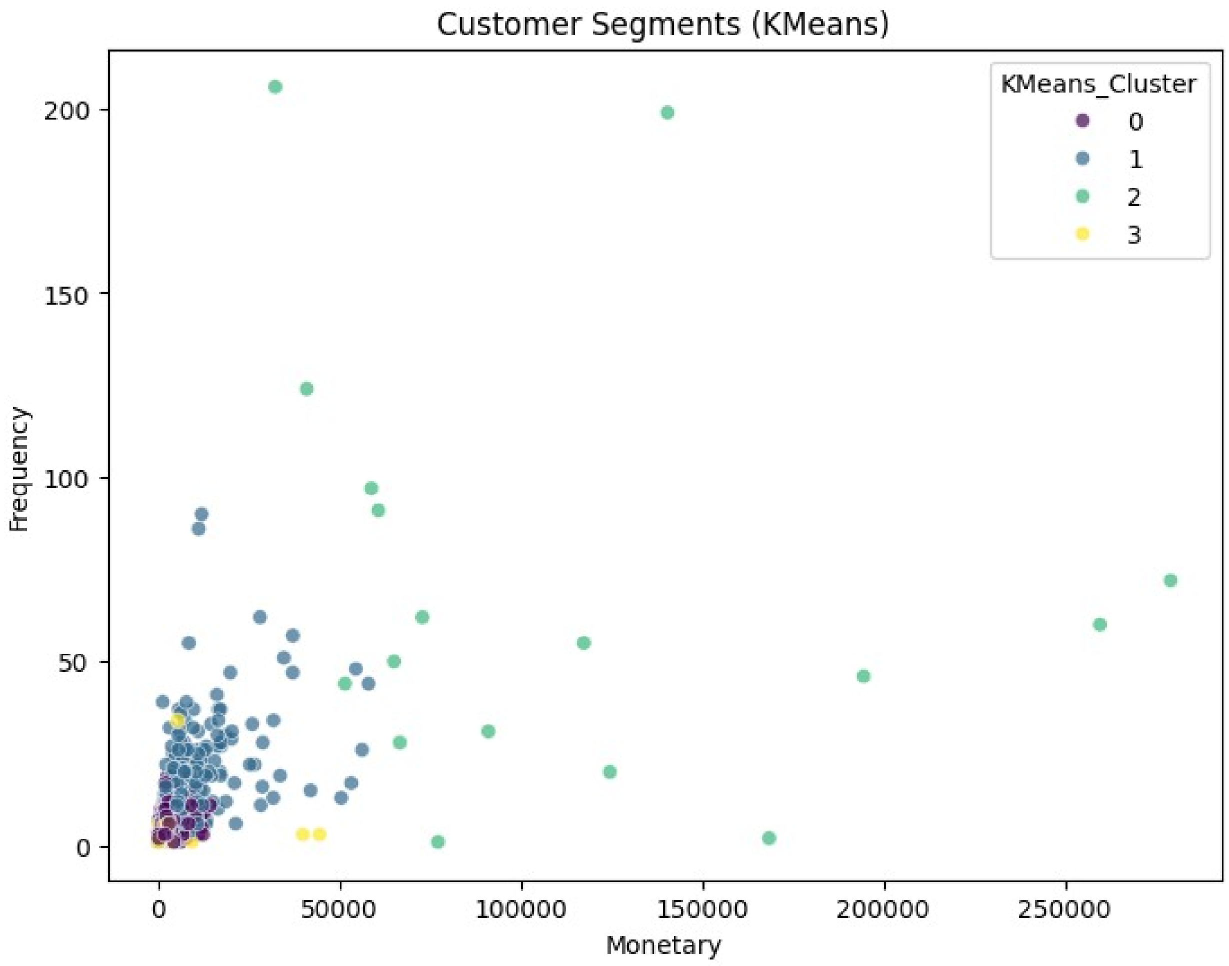
{"columns":
[{"name":"KMeans_Cluster","rawType":"int32","type":"integer"},
{"name":"Recency","rawType":"float64","type":"float"},
{"name":"Frequency","rawType":"float64","type":"float"},
{"name":"Monetary","rawType":"float64","type":"float"},
{"name":"TotalQuantity","rawType":"float64","type":"float"},
{"name":"UniqueProducts","rawType":"float64","type":"float"}],"ref":"10e7c72c-b44c-45bb-9fc0-f08621f1039a","rows":
[["0","46.32494684620836","3.2955350815024804","1138.5100715804394","701.785258681786","49.324238128986536"],
["1","16.29705215419501","14.222222222222221","6975.729750566893","4049.278911564626","207.81179138321994"],
["2","25.764705882352942","69.88235294117646","111849.4594117647","64481.64705882353","592.7058823529412"],
["3","249.92504743833018","1.560721062618596","547.5707789373814","280.0123339658444","24.040796963946867"]],"shape":{"columns":5,"rows":4}}

```
pd.crosstab(
    customer_df["KMeans_Cluster"],
    customer_df["GMM_Cluster"],
    normalize="index"
)
```

{"columns":
[{"name":"KMeans_Cluster","rawType":"int32","type":"integer"},
{"name":"0","rawType":"float64","type":"float"},
{"name":"1","rawType":"float64","type":"float"},
{"name":"2","rawType":"float64","type":"float"},
{"name":"3","rawType":"float64","type":"float"}],"ref":"98dbf94c-996d-4f18-badd-818832f80dfa","rows":
[["0","0.5262225372076541","0.1708008504606662","0.011339475549255847","0.2916371367824238"],
["1","0.0022675736961451248","0.8480725623582767","0.14965986394557823","0.0"],["2","0.0","0.0","1.0","0.0"],
["3","0.28178368121442127","0.0","0.016129032258064516","0.7020872865275142"]],"shape":{"columns":4,"rows":4}}

12. Visualization

```
plt.figure(figsize=(8,6))
sns.scatterplot(
    data=customer_df,
    x="Monetary",
    y="Frequency",
    hue="KMeans_Cluster",
    palette="viridis",
    alpha=0.7
)
plt.title("Customer Segments (KMeans)")
plt.show()
```



13. Business Interpretation

Cluster	Description
0	Bulk low-frequency buyers

Cluster	Description
1	Recent low spenders
2	High-value loyal customers
3	At-risk customers

14. Production Pipeline Design

Inference Function

```
FINAL_MODEL = kmeans   # baseline, stable, interpretable
FINAL_SCALER = scaler
FINAL_FEATURES = features

def assign_customer_segment(df):
    X = FINAL_SCALER.transform(df[FINAL_FEATURES])
    return FINAL_MODEL.predict(X)
```

Artifacts to Persist

- scaler.pkl
- kmeans_model.pkl
- Feature schema JSON

```
import joblib
import json
from pathlib import Path

ARTIFACT_DIR = Path("../artifacts")
ARTIFACT_DIR.mkdir(exist_ok=True)

joblib.dump(FINAL_SCALER, ARTIFACT_DIR / "scaler.pkl")
joblib.dump(FINAL_MODEL, ARTIFACT_DIR / "kmeans.pkl")
joblib.dump(gmm, ARTIFACT_DIR / "gmm.pkl")
```

```
['..\\artifacts\\gmm.pkl']
```

```
feature_schema = {
    "features": FINAL_FEATURES,
    "description": {
        "Recency": "Days since last purchase",
        "Frequency": "Number of invoices",
        "Monetary": "Total spend",
        "TotalQuantity": "Total items purchased",
        "UniqueProducts": "Distinct products purchased"
    }
}
```

```
with open(ARTIFACT_DIR / "feature_schema.json", "w") as f:
    json.dump(feature_schema, f, indent=4)
```