D. Saul Jameson

https://www.linkedin.com/in/dsauljameson/

```
# Authenticate Google Cloud in Colab
from google.colab import auth
auth.authenticate_user()
```

```
# Install required libraries
!pip install --quiet google-cloud-bigquery pandas-gbq scikit-learn matplotlib seaborn
```

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from google.cloud import bigquery
from pandas_gbq import read_gbq, to_gbq
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
```

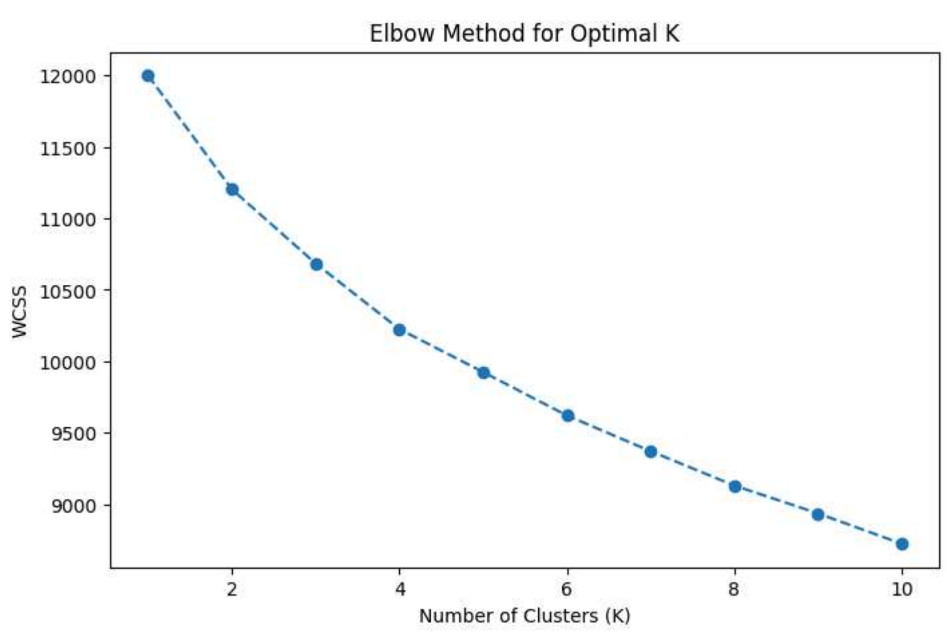
```
# Set project and table info
PROJECT_ID = "redacted"
DATASET_ID = "redacted"
TABLE_ID = "redacted"

# Load data from BigQuery
query = f"""
SELECT age, gender, customer_id, annual_income, num_transactions, avg_transaction_value,
       avg_days_between_transactions, customer_tenure, loyalty_score,
       device_preference_numeric, first_clicked_ad_numeric, customer_service_calls, return_rate
FROM `{PROJECT_ID}.{DATASET_ID}.{TABLE_ID}`
"""

df = read_gbq(query, project_id=PROJECT_ID)
```

```
Downloading: 100%|██████████|
```

Start coding or generate with AI.

```
# Encode categorical variables
label_encoders = {}
for col in ["gender"]:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Exclude customer_id from clustering but keep it in the DataFrame
features = df.drop(columns=["customer_id"])  # Only drop for clustering

# Scale numeric features (use features, not df)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
```

```
# Elbow method to choose optimal K
wcss = []
K_range = range(1, 11)


for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(scaled_features)
    wcss.append(kmeans.inertia_)


# Plot Elbow Method
plt.figure(figsize=(8,5))
plt.plot(K_range, wcss, marker="o", linestyle="--")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("WCSS")
plt.title("Elbow Method for Optimal K")
plt.show()
```



```
# Fit K-Means with optimal K
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df["kMeanClusterNum"] = kmeans.fit_predict(scaled_features)


print(df.groupby("kMeanClusterNum").mean(numeric_only=True))
```

```
                       age    gender  annual_income  num_transactions  \
kMeanClusterNum
0                53.452756  0.346457   81800.414488          37.88189
1                50.596958  1.711027   89448.818783         56.163498
2                42.289157  1.678715   79273.980562         40.638554
3                41.269231  0.525641  101733.268547         69.816239

                 avg_transaction_value  avg_days_between_transactions  \
kMeanClusterNum
0                           251.022008                      31.973228
1                           229.116768                      29.431559
2                           255.082048                      33.425703
3                           277.779530                      29.788462

                 customer_tenure  loyalty_score  device_preference_numeric  \
kMeanClusterNum
0                      11.606299      47.299567                   2.480315
1                       9.372624      47.810989                   2.505703
2                      11.614458      46.616948                   1.477912
3                      10.636752      53.149017                   1.461538

                 first_clicked_ad_numeric  customer_service_calls  return_rate
kMeanClusterNum
0                                3.102362                9.468504     0.420236
1                                3.444867                5.634981     0.648821
2                                2.534137                8.827309     0.359478
3                                3.017094                6.141026     0.625171
```

Key Takeaways

Cluster 2: High-income big spenders (100K+income,348 avg transaction).

Cluster 1: Frequent shoppers with high return rates (63% return rate).

Cluster 3: Most loyal customers (67 loyalty score) but spend the least ($165 avg transaction).

Cluster 0: Middle-of-the-road customers—moderate spending, moderate loyalty, no strong engagement.

Longer Interpretation

Cluster 0: Mid-income, moderate loyalty, infrequent shoppers These folks skew younger than the other groups—around 35 years old—and earn a solid but not top-tier income, averaging around 88K.Theiraveragetransactionisabout 230, which puts them in the middle range. They shop infrequently (about every 39 days), and their loyalty score is the lowest of all the clusters, hovering around 41.

Device preference is split between tablets and computers, and they were most likely acquired through Google ads. Customer service calls and return rates are middle-of-the-road. Overall, this is a group of moderate earners and moderate spenders who don't engage deeply with the brand.

Cluster 1: Older, frequent buyers, high returns This group is around 50 years old and earns roughly 87Kayear.Theyshopmorefrequently—every22days—andspendaround 245 per transaction. They lean toward mobile devices and were slightly more likely to have come through TikTok ads.

Their loyalty score is decent at 52.5, but what really jumps out is their return rate: it's the highest of all clusters at 63 percent. That, paired with a moderate number of customer service calls, suggests they might be impulse buyers or deal-seekers who aren't always happy with their purchases.

Cluster 2: High-income, high-spending, moderate engagement These customers earn the most—over 100K—andspendthemost,withanaveragetransactionofabout 348. They're also in the 50-year-old range and tend to shop less frequently, about every 37 days. Loyalty score sits in the middle at 46.

They're mostly mobile users, and their first clicked ad was most often on Instagram. They call customer service a moderate amount and have a fairly high return rate (around 59 percent). These look like high-value customers who spend big but don't necessarily stick around for the long haul.

Cluster 3: Budget-conscious, highly loyal, low returns This group has the lowest income (around 70K)andthelowestaveragetransactionvalue(around 165), but they make up for it with high frequency (shopping every 22 days) and a standout loyalty score of nearly 67.

They lean slightly toward desktop users and were acquired mainly through Instagram and Google. They call customer service the least and have the lowest return rate of all clusters. This is your rock-solid base—consistent, loyal shoppers who may not spend the most, but they stick around and don't cause many issues.

```
# Visualize clusters (example: income vs. transaction value)
df["kMeanClusterNum"] = df["kMeanClusterNum"].astype(str)
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x="annual_income", y="avg_transaction_value", hue="kMeanClusterNum", palette="viridis")
plt.title("K-Means Clustering: Annual Income vs. Avg Transaction Value")
plt.xlabel("Annual Income")
plt.ylabel("Average Transaction Value")
plt.legend(title="Cluster")
plt.show()
```

Show hidden output

```
TEMP_TABLE_ID = "kMeanClusterNumber"

# Select only customer_id and kMeanClusterNum for update
df_temp = df[["customer_id", "kMeanClusterNum"]]

# Convert kMeanClusterNum to integer
df_temp["kMeanClusterNum"] = df_temp["kMeanClusterNum"].astype(int)

# Upload to BigQuery
to_gbq(df_temp, f"{DATASET_ID}.{TEMP_TABLE_ID}", project_id=PROJECT_ID, if_exists="replace")

print(f"Successfully uploaded temporary table {DATASET_ID}.{TEMP_TABLE_ID}")
```

```
<ipython-input-45-467c93093a3c>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cc
  df_temp["kMeanClusterNum"] = df_temp["kMeanClusterNum"].astype(int)
100%|██████████| 1/1 [00:00<00:00, 12520.31it/s]Successfully uploaded temporary table dSaulJamesonPortfolio.kMeanClusterNumber
```