```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

```
file_path = '/content/CC GENERAL.csv'
credit_card_data = pd.read_csv(file_path)

credit_card_data.head()
```

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333

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

Data Cleaning

```
# Drop irrelevant columns (e.g., 'CUST_ID') if present
if 'CUST_ID' in credit_card_data.columns:
    credit_card_data = credit_card_data.drop(columns=['CUST_ID'])

# Handle missing values by replacing with the column mean
credit_card_data.fillna(credit_card_data.mean(), inplace=True)

# Check the cleaned data
credit_card_data.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PU
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

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

```
# Scale the data using StandardScaler
scaler = StandardScaler()
credit_card_data_scaled = scaler.fit_transform(credit_card_data)

# Convert back to DataFrame for readability (optional)
scaled_df = pd.DataFrame(credit_card_data_scaled, columns=credit_card_data.columns)

# Display the first few rows of scaled data
scaled_df.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCH
0	-0.731989	-0.249434	-0.424900	-0.356934	-0.349079	-0.466786	-0.806490	
1	0.786961	0.134325	-0.469552	-0.356934	-0.454576	2.605605	-1.221758	
2	0.447135	0.518084	-0.107668	0.108889	-0.454576	-0.466786	1.269843	
3	0.049099	-1.016953	0.232058	0.546189	-0.454576	-0.368653	-1.014125	
4	-0.358775	0.518084	-0.462063	-0.347294	-0.454576	-0.466786	-1.014125	

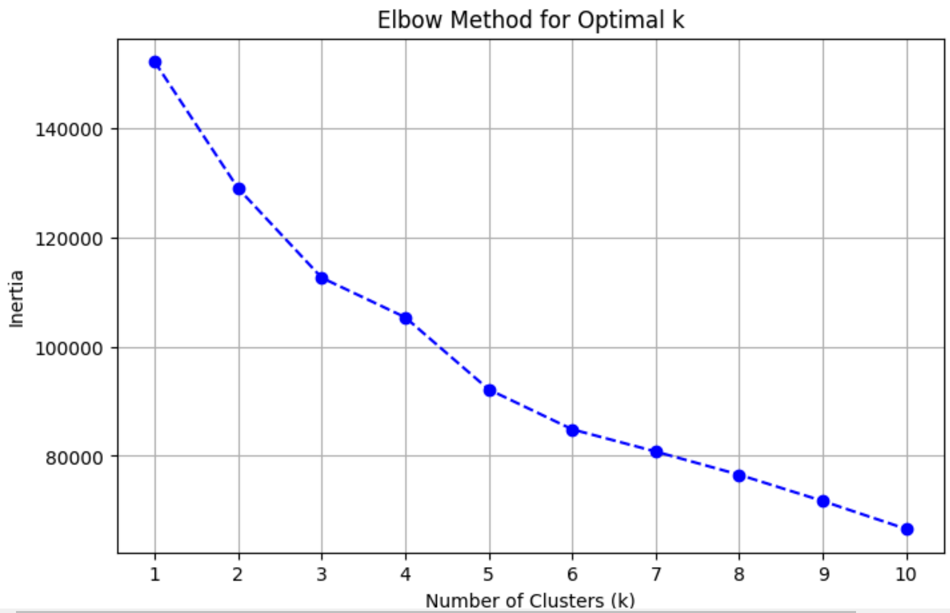
Next steps: Generate code with scaled_df View recommended plots New interactive sheet

Determine Optimal k Using the Elbow Method

```
# Determine the optimal number of clusters using the Elbow Method
inertia = []
k_range = range(1, 11)  # Test for k from 1 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(credit_card_data_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o', linestyle='--', color='b')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```



Perform K-Means Clustering

```
# Perform k-means clustering
optimal_k = 4  # Replace with your chosen value from the elbow plot
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(credit_card_data_scaled)

# Add the cluster labels to the original dataset
credit_card_data['Cluster'] = clusters

# Display the dataset with cluster labels
credit_card_data.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PU
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

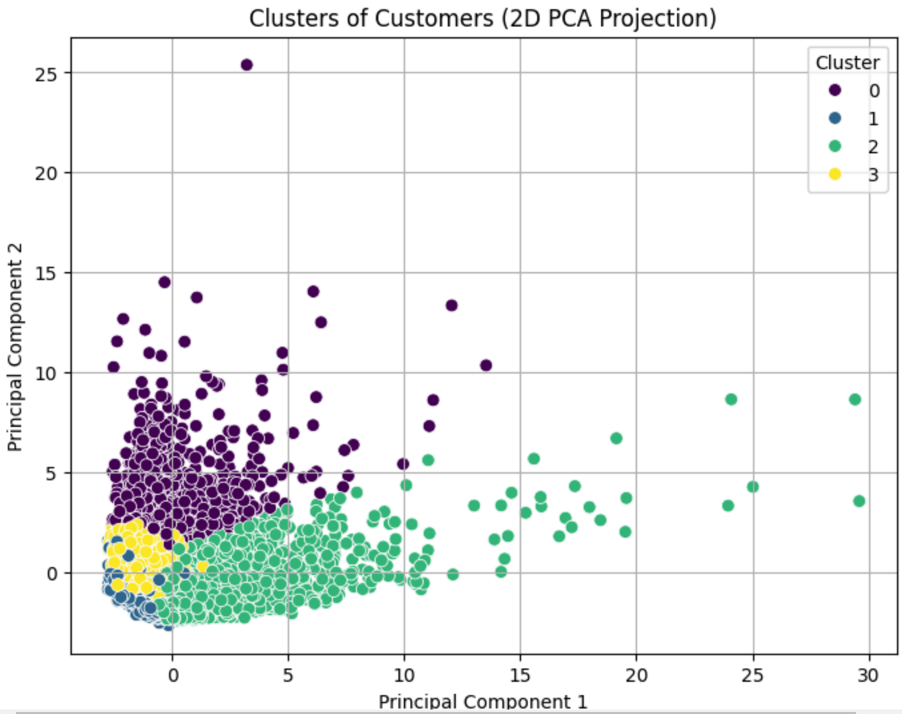
Next steps: Generate code with credit_card_data | View recommended plots | New interactive sheet

Visualize Clusters (2D PCA Projection)

```
from sklearn.decomposition import PCA

# Reduce the data to 2D using PCA for visualization
pca = PCA(n_components=2)
pca_components = pca.fit_transform(credit_card_data_scaled)

# Plot the clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_components[:, 0], y=pca_components[:, 1], hue=clusters, palette='viridis', s=50)
plt.title('Clusters of Customers (2D PCA Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



Install Required Library for K-Medoids

```
# Install scikit-learn-extra for K-Medoids
!pip install scikit-learn-extra
```

```
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.6 kB)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.26.4)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.13.1)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.5.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-ext
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-le
Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 22.1 MB/s eta 0:00:00
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.3.0
```

Perform K-Medoids Clustering

```
from sklearn_extra.cluster import KMedoids

# Perform k-medoids clustering
kmedoids = KMedoids(n_clusters=optimal_k, random_state=42)
medoid_clusters = kmedoids.fit_predict(credit_card_data_scaled)

# Add medoid cluster labels to the original dataset
credit_card_data['Medoid_Cluster'] = medoid_clusters

# Display the dataset with medoid cluster labels
```

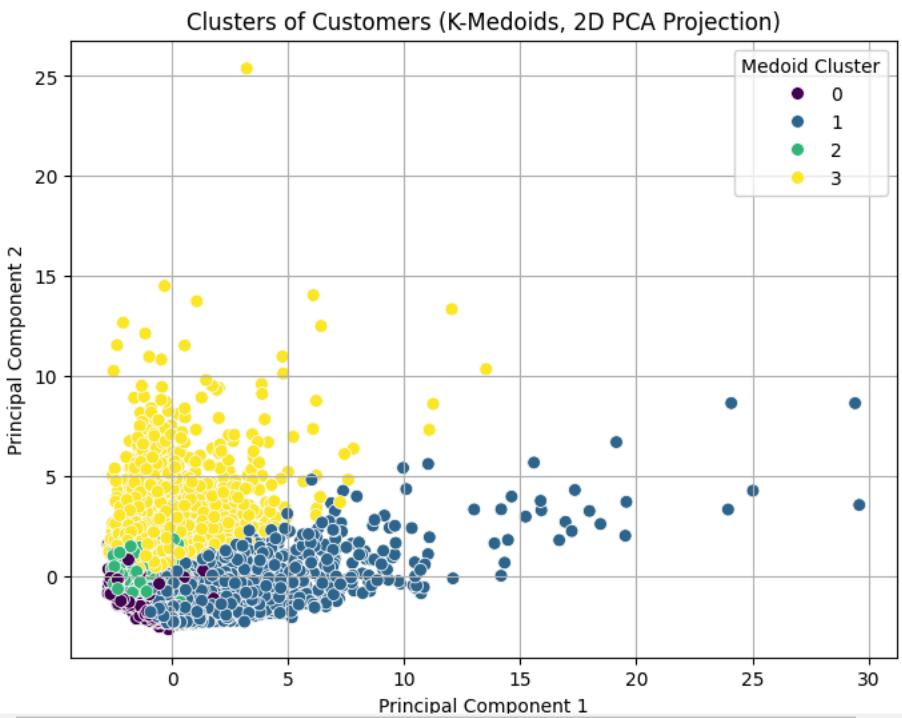
```
credit_card_data.head()
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PU
0	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	
1	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	
2	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	
3	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	
4	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	

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

Visualize K-Medoids Clusters (2D PCA Projection)

```
# Use the same PCA components for visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x=pca_components[:, 0], y=pca_components[:, 1], hue=medoid_clusters, palette='viridis', s=50)
plt.title('Clusters of Customers (K-Medoids, 2D PCA Projection)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Medoid Cluster')
plt.grid(True)
plt.show()
```



Compare Cluster Assignments

```
# Compare cluster assignments
comparison = credit_card_data[['Cluster', 'Medoid_Cluster']]
print("Comparison of Clusters:")
print(comparison.head())

# Optional: Calculate Adjusted Rand Index (ARI) to assess similarity
from sklearn.metrics import adjusted_rand_score
ari_score = adjusted_rand_score(credit_card_data['Cluster'], credit_card_data['Medoid_Cluster'])
print(f"Adjusted Rand Index (ARI) between K-Means and K-Medoids: {ari_score}")
```

```
Comparison of Clusters:
   Cluster  Medoid_Cluster
0        3               2
1        0               3
2        2               1
3        3               0
4        3               2
```

```
Adjusted Rand Index (ARI) between K-Means and K-Medoids: 0.7208125862486644
```