```
import pandas as pd

# Load the data
df_segmentation = pd.read_csv('/content/client_churn_synthetic.csv')

# Display the first few rows and information about the data
display(df_segmentation.head())
df_segmentation.info()
```

	customer_id	gender	senior_citizen	partner	dependents	tenure	contract	payment_methoc
0	CUST100000	Male	0	No	No	22	Month-to-month	Credit carc (automatic
1	CUST100001	Female	0	Yes	No	25	Month-to-month	Mailed checl
2	CUST100002	Female	0	Yes	No	16	Month-to-month	Electronic checl
3	CUST100003	Female	0	No	Yes	14	Month-to-month	Bank transfe (automatic
4	CUST100004	Male	0	No	No	13	Month-to-month	Mailed checl

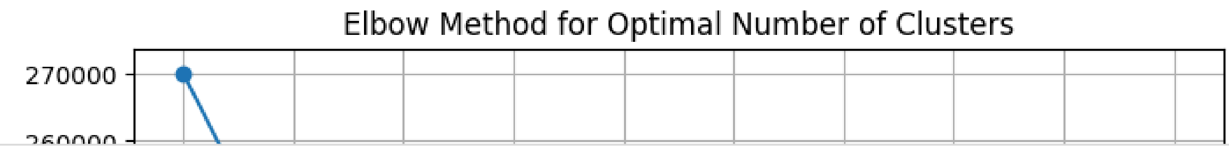
Loading...

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   customer_id        10000 non-null  object 
 1   gender             10000 non-null  object 
 2   senior_citizen     10000 non-null  int64  
 3   partner            10000 non-null  object 
 4   dependents         10000 non-null  object 
 5   tenure             10000 non-null  int64  
 6   contract           10000 non-null  object 
 7   payment_method     10000 non-null  object 
 8   internet_service   10000 non-null  object 
 9   monthly_charges    10000 non-null  float64
 10  total_charges      10000 non-null  float64
 11  multiple_lines     10000 non-null  object 
 12  online_security    10000 non-null  object 
 13  online_backup      10000 non-null  object 
 14  device_protection  10000 non-null  object 
 15  tech_support       10000 non-null  object 
 16  streaming_tv       10000 non-null  object 
 17  streaming_movies   10000 non-null  object 
 18  churn              10000 non-null  int64  
dtypes: float64(2), int64(3), object(14)
memory usage: 1.4+ MB
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

```python
# Exclude the 'churn' column from features for segmentation if it was included
# Ensure all columns are numeric before scaling and PCA
X_segmentation = df_segmentation_processed.select_dtypes(include=['int64', 'float64', 'bool


# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_segmentation)

# Apply PCA for dimensionality reduction
# Let's start by reducing to 2 components for visualization purposes
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_scaled)

# Determine the optimal number of clusters using the Elbow Method
inertia = []
# Trying a range of cluster numbers, for example from 1 to 10
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10) # Explicitly set n_init
    kmeans.fit(X_scaled) # Use scaled data for elbow method
    inertia.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 4))
plt.plot(range(1, 11), inertia, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()

# Based on the elbow method, choose an appropriate number of clusters
# Let's assume from the plot that 3 or 4 might be a good number (this is subjective and dep
# For demonstration, let's choose 3 clusters
n_clusters = 3

# Apply K-Means clustering with the chosen number of clusters
kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10) # Explicitly set n_init
df_segmentation_processed['segment'] = kmeans.fit_predict(X_scaled) # Use scaled data for c

# Display the first few rows with the assigned segment
display(df_segmentation_processed.head())

# Visualize the clusters in the PCA-reduced space
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_segmentation_processed['segment'], cma
plt.title('Customer Segments (PCA Reduced)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(scatter, label='Segment')
plt.grid(True)
plt.show()
```

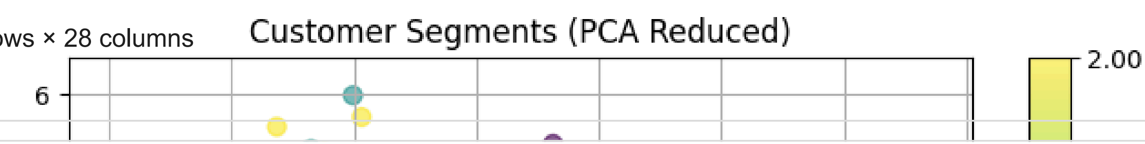
Loading...




```
# Feature Engineering for Segmentation
# Assuming 'tenure', 'monthly_charges', and 'total_charges' are key for behavior

# Create a new feature for the average monthly charge over tenure
# Avoid division by zero if tenure is 0
df_segmentation_processed['average_monthly_charge'] = df_segmentation_processed.apply(
    lambda row: row['total_charges'] / row['tenure'] if row['tenure'] != 0 else 0, axis=1
)

# Create a new feature for the ratio of total charges to monthly charges (can indicate cons
# Avoid division by zero if monthly_charges is 0
df_segmentation_processed['total_to_monthly_charges_ratio'] = df_segmentation_processed.app
    lambda row: row['total_charges'] / row['monthly_charges'] if row['monthly_charges'] !=
)

display(df_segmentation_processed.head())
```

	senior_citizen	tenure	monthly_charges	total_charges	churn	contract_One year	contract_Two year
0	0	22	50.44	1127.91	1	False	False
1	0	25	37.63	924.65	1	False	False
2	0	16	52.87	841.74	0	False	False
3	0	14	23.34	322.18	1	False	False
4	0	13	22.57	282.98	0	False	False

5 rows × 28 columns




```
# Check for missing values
print("Missing values before preprocessing:")
print(df_segmentation.isnull().sum())

# Handle missing values (example: fill with median for numerical, mode for categorica
for col in df_segmentation.columns:
    if df_segmentation[col].dtype == 'object':
        # Use .loc to avoid the SettingWithCopyWarning and FutureWarning
        df_segmentation.loc[:, col] = df_segmentation[col].fillna(df_segmentation[col
    else:
        # Use .loc to avoid the SettingWithCopyWarning and FutureWarning
        df_segmentation.loc[:, col] = df_segmentation[col].fillna(df_segmentation[col

print("\nMissing values after preprocessing:")
print(df_segmentation.isnull().sum())

# Convert 'total_charges' to numeric, coercing errors
df_segmentation['total_charges'] = pd.to_numeric(df_segmentation['total_charges'], er
```

```
# Fill any new NaNs created by coercion in 'total_charges'
# Address FutureWarning by not using inplace=True
df_segmentation['total_charges'] = df_segmentation['total_charges'].fillna(df_segment


# For segmentation, we might not need all columns, especially customer_id and churn
# We will focus on columns related to purchasing behavior or service usage
# Let's start by selecting potentially relevant columns.
# We'll need to decide which columns represent "purchasing behavior".
# Assuming 'tenure', 'monthly_charges', and 'total_charges' are relevant,
# and possibly some service-related columns that imply usage/behavior.
# For this example, let's select numerical features and some relevant categorical one

# Identify numerical features
numerical_features_segmentation = df_segmentation.select_dtypes(include=['int64', 'fl

# Identify some relevant categorical features that might reflect behavior or service
# Excluding customer_id and churn for segmentation
categorical_features_segmentation = ['contract', 'payment_method', 'internet_service'
                                     'multiple_lines', 'online_security', 'online_bac
                                     'device_protection', 'tech_support', 'streaming_
                                     'streaming_movies']

# Combine the relevant features
features_for_segmentation = numerical_features_segmentation + categorical_features_se

# Create a new dataframe with selected features
df_segmentation_selected = df_segmentation[features_for_segmentation].copy()

# One-Hot Encode categorical features for segmentation
df_segmentation_processed = pd.get_dummies(df_segmentation_selected, columns=categori


# Display the first few rows of the preprocessed data for segmentation
display(df_segmentation_processed.head())
df_segmentation_processed.info()
```

Loading...