In [175…
```
# Import necessary libraries
#import pandas as pd
#from sklearn.feature_selection import RFE
#from sklearn.cluster import KMeans

# Load data into a Pandas DataFrame
#df = pd.read_csv('customer_data.csv')

# Select the relevant columns for segmentation
#X = df[['age', 'income', 'spending']]

# Create the RFE model and select 2 attributes
#rfe = RFE(KMeans(n_clusters=3), 2)
#rfe = rfe.fit(X, y=None)

# Print the selected features
#print(X.columns[rfe.support_])

# Use the selected features to perform customer segmentation
#X_selected = X[X.columns[rfe.support_]]
#kmeans = KMeans(n_clusters=3)
#predictions = kmeans.fit_predict(X_selected)

# Add the cluster labels as a new column in the DataFrame
#df['cluster'] = predictions

# Print the resulting DataFrame
#print(df)
```

In [176…
```
# Import necessary libraries
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.cluster import KMeans
```

In [177…
```
# Load data into a Pandas DataFrame
df = pd.read_excel(r"C:\Users\djbro\OneDrive\Desktop\Clustering\Customer Segmentation.xl
```

In [178…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
```

```
 19  NumWebVisitsMonth    2240 non-null   int64
 20  AcceptedCmp3         2240 non-null   int64
 21  AcceptedCmp4         2240 non-null   int64
 22  AcceptedCmp5         2240 non-null   int64
 23  AcceptedCmp1         2240 non-null   int64
 24  AcceptedCmp2         2240 non-null   int64
 25  Complain             2240 non-null   int64
 26  Z_CostContact        2240 non-null   int64
 27  Z_Revenue            2240 non-null   int64
 28  Response             2240 non-null   int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

In [179…
```
#Select columns we want to analyze
df = df[['Year_Birth', 'Education', 'Marital_Status','Income','Kidhome','Teenhome','MntW
```

In [180…
```
df.isnull().sum()
df.fillna(df.mean(), inplace=True)
```

```
C:\Users\djbro\AppData\Local\Temp\ipykernel_14472\1168785774.py:2: FutureWarning: Droppi
ng of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated;
in a future version this will raise TypeError.  Select only valid columns before calling
the reduction.
  df.fillna(df.mean(), inplace=True)
```

In [181…
```
df.isnull().sum()
```

Out[181]:
```
Year_Birth         0
Education          0
Marital_Status     0
Income             0
Kidhome            0
Teenhome           0
MntWines           0
dtype: int64
```

In [182…
```
df['Education'].unique()
```

Out[182]:
```
array(['Graduation', 'PhD', 'Master', 'Basic', '2n Cycle'], dtype=object)
```

In [183…
```
df['Education'] = df['Education'].map({'Basic':0,'Graduation':1,'Master':2,'2n Cycle':2,
df.head()
```

Out[183]:

	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	MntWines
0	1957	1	Single	58138.0	0	0	635
1	1954	1	Single	46344.0	1	1	11
2	1965	1	Together	71613.0	0	0	426
3	1984	1	Together	26646.0	1	0	11
4	1981	3	Married	58293.0	1	0	173

In [184…
```
df['Marital_Status'].unique()
```

Out[184]:
```
array(['Single', 'Together', 'Married', 'Divorced', 'Widow', 'Alone',
       'Absurd', 'YOLO'], dtype=object)
```

In [185…
```
df['Marital_Status'] = df['Marital_Status'].map({'Single':0,'Married':1,'Divorced':2,'Wi
df.head()
```

Out[185]:

	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	MntWines

0	1957	1	0	58138.0	0	0	635
1	1954	1	0	46344.0	1	1	11
2	1965	1	4	71613.0	0	0	426
3	1984	1	4	26646.0	1	0	11
4	1981	3	1	58293.0	1	0	173

In [186…
```
df['Marital_Status'].unique()
```

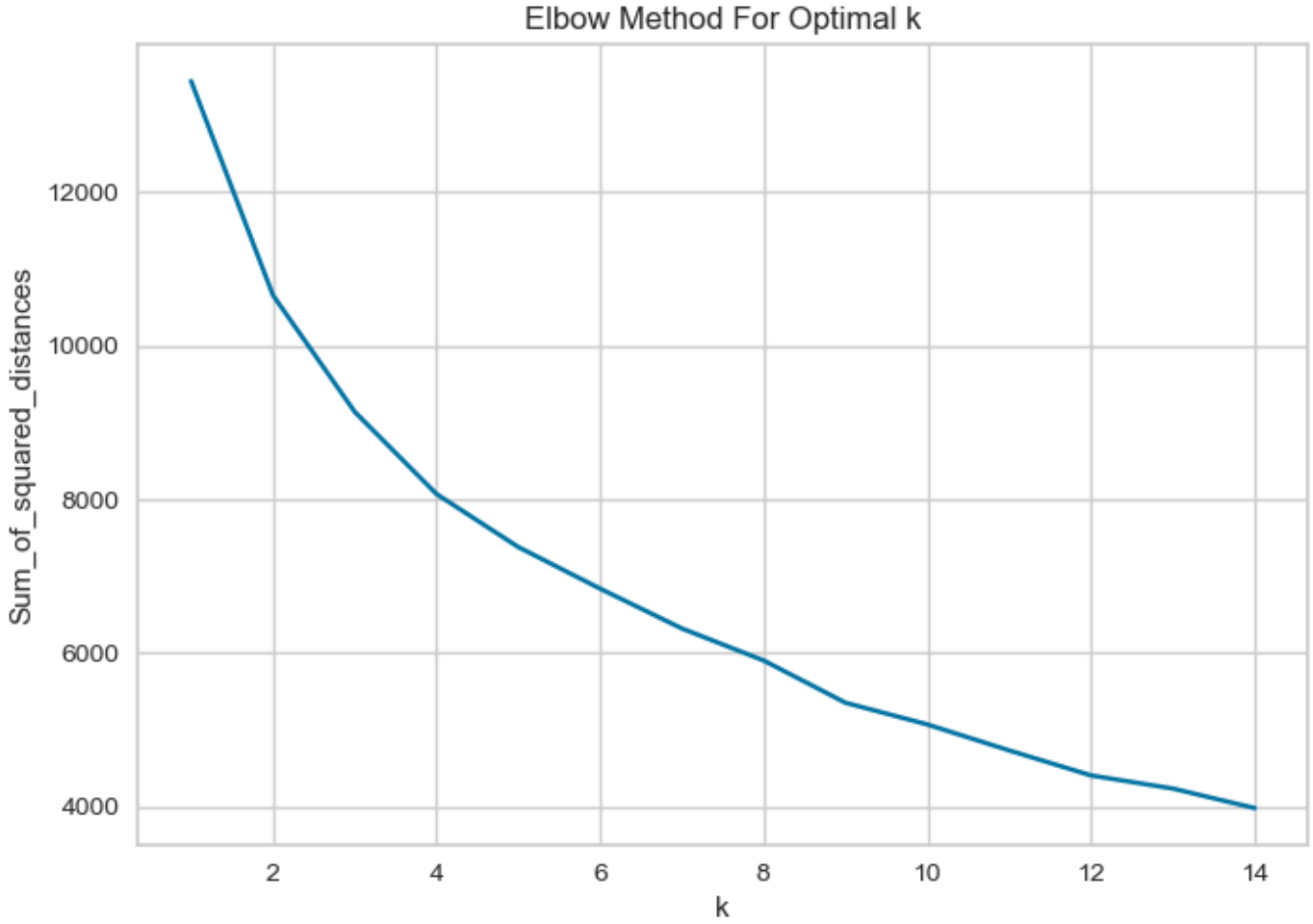
Out[186]:
```
array([0, 4, 1, 2, 3], dtype=int64)
```

In [187…
```
# Select the relevant columns for segmentation
X = df.drop('MntWines',axis=1)
y = df['MntWines']
```

In [198…
```
# Scale the data using the StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled
```

Out[198]:
```
array([[-0.98534473, -0.79291025, -1.16021939,  0.23532677, -0.82521765,
        -0.92989438],
       [-1.23573295, -0.79291025, -1.16021939, -0.23582624,  1.03255877,
         0.90693402],
       [-0.3176428 , -0.79291025,  1.50053458,  0.77363327, -0.82521765,
        -0.92989438],
       ...,
       [ 1.01776106, -0.79291025,  0.17015759,  0.18910632, -0.82521765,
        -0.92989438],
       [-1.06880747,  0.39831642,  1.50053458,  0.67903514, -0.82521765,
         0.90693402],
       [-1.23573295,  1.58954309, -0.4950309 ,  0.02483795,  1.03255877,
         0.90693402]])
```

In [189…
```
#Elbow method to minimize WSS (within-cluster Sum of Square)
import matplotlib.pyplot as plt
Sum_of_squared_distances =[]
K = range(1,15)
for k in K:
    km =KMeans(n_clusters =k)
    km =km.fit(X_scaled)
    Sum_of_squared_distances.append(km.inertia_)
###plotting Elbow
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```

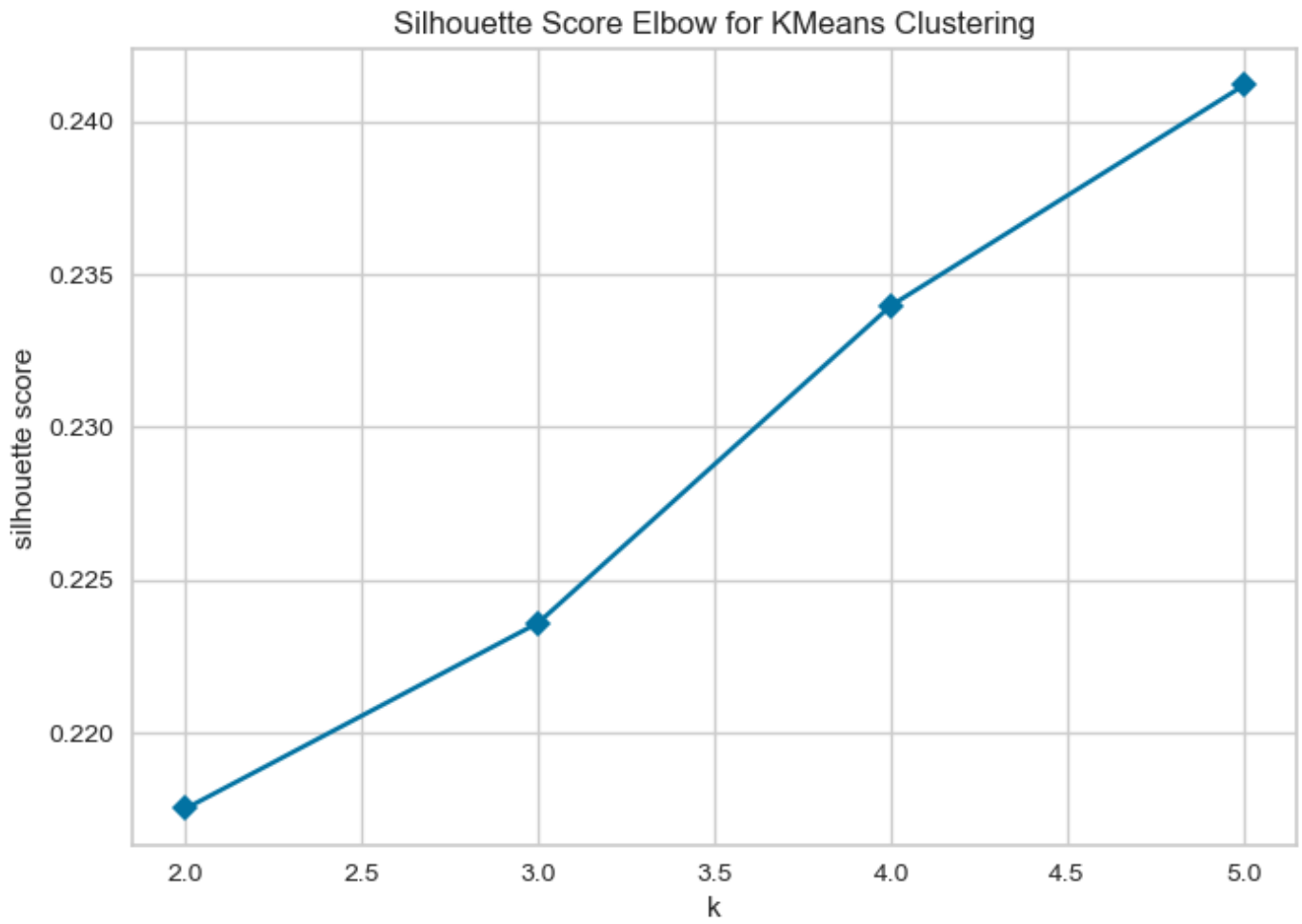


In [190…
```
#Silhouette Coefficient method, the silhouette coefficient of a data
#measures how well data are assigned to its own cluster and how far they
#are from other clusters. A silhouette close to 1 means the data points
#are in an appropriate cluster and a silhouette
#coefficient close to -1 implies out data is in the wrong cluster.
```

In [191…
```
from yellowbrick.cluster import KElbowVisualizer
from sklearn.metrics import silhouette_samples, silhouette_score

model = KMeans(random_state=123)
# Instantiate the KElbowVisualizer with the number of clusters and the metric
visualizer = KElbowVisualizer(model, k=(2,6), metric='silhouette', timings=False)
# Fit the data and visualize
visualizer.fit(X_scaled)
visualizer.poof()
```

```
C:\Users\djbro\anaconda3\lib\site-packages\yellowbrick\utils\kneed.py:156: YellowbrickWa
rning: No 'knee' or 'elbow point' detected This could be due to bad clustering, no actua
l clusters being formed etc.
  warnings.warn(warning_message, YellowbrickWarning)
C:\Users\djbro\anaconda3\lib\site-packages\yellowbrick\cluster\elbow.py:374: Yellowbrick
Warning: No 'knee' or 'elbow' point detected, pass `locate_elbow=False` to remove the wa
rning
  warnings.warn(warning_message, YellowbrickWarning)
```



Out[191]: <AxesSubplot:title={'center':'Silhouette Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='silhouette score'>

In [192…
```
# Print the selected features
print(X.columns)
```

```
Index(['Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome'],
      dtype='object')
```

In [193…
```
# Use the selected features to perform customer segmentation
X_selected = X[X.columns]
kmeans = KMeans(n_clusters=5)
predictions = kmeans.fit_predict(X_scaled)
```

In [194…
```
# Add the cluster labels as a new column in the DataFrame
df['cluster'] = predictions
```

In [195…
```
# Print the resulting DataFrame
df.head(5)
```

Out[195]:

	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	MntWines	cluster
0	1957	1	0	58138.0	0	0	635	0
1	1954	1	0	46344.0	1	1	11	3
2	1965	1	4	71613.0	0	0	426	0
3	1984	1	4	26646.0	1	0	11	1
4	1981	3	1	58293.0	1	0	173	1

In [ ]:

In [ ]: