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

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE

import warnings
warnings.filterwarnings('ignore')
```

```
#loading the dataset
df = pd.read_csv('/content/dataset.csv')

#structure
#print(df.head(), '\n')
print(f"Dataset Shape: {df.shape}\n")
#print(df.info(), '\n')
#print(df.describe().T, '\n')
```

```
Dataset Shape: (2240, 29)
```

```
#missing values
missing_info = df.isnull().sum()
missing_cols = missing_info[missing_info > 0]
if not missing_cols.empty:
    print("Columns with missing values:\n", missing_cols)

    #droping missing values
    df.dropna(inplace=True)
    print(f"Remaining rows after removing missing values: {len(df)}\n")
print(f"New Dataset Shape: {df.shape}\n")
```

```
Columns with missing values:
 Income    24
dtype: int64
Remaining rows after removing missing values: 2216

New Dataset Shape: (2216, 29)
```

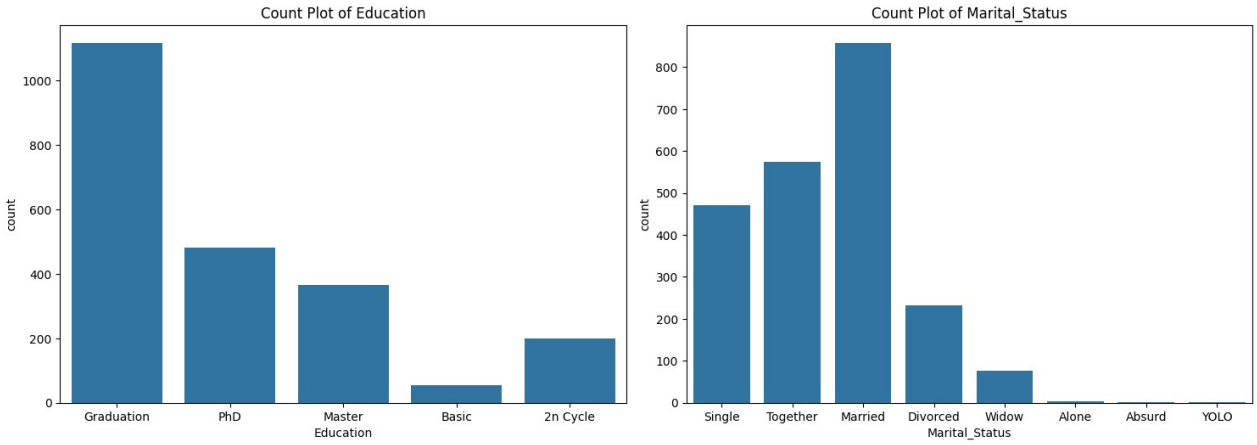
```
#extracting date components
parts = df["Dt_Customer"].str.split("-", expand=True)
df["day"] = parts[0].astype(int)
df["month"] = parts[1].astype(int)
df["year"] = parts[2].astype(int)

#droping unnecessary columns
```

```
#droping unnecessary columns
df.drop(['Z_CostContact', 'Z_Revenue', 'Dt_Customer'], axis=1, inplace=True)
```

```
#separate numeric and categorical columns
objects = df.select_dtypes(include='object').columns.tolist()
floats = df.select_dtypes(include='float').columns.tolist()

print("Categorical columns:", objects)
print("Numeric columns:", floats)
```

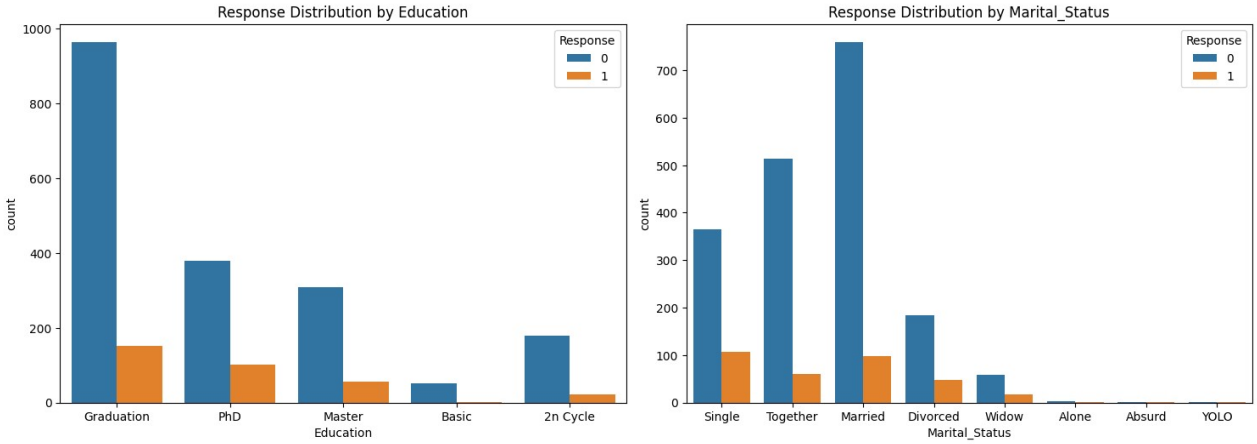
```
Categorical columns: ['Education', 'Marital_Status']
Numeric columns: ['Income']
```

```
plt.figure(figsize=(15, 10))
for i, col in enumerate(objects):
    plt.subplot(2, 2, i + 1)
    sb.countplot(x=col, data=df)
    plt.title(f"Count Plot of {col}")
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(15, 10))
for i, col in enumerate(objects):
```

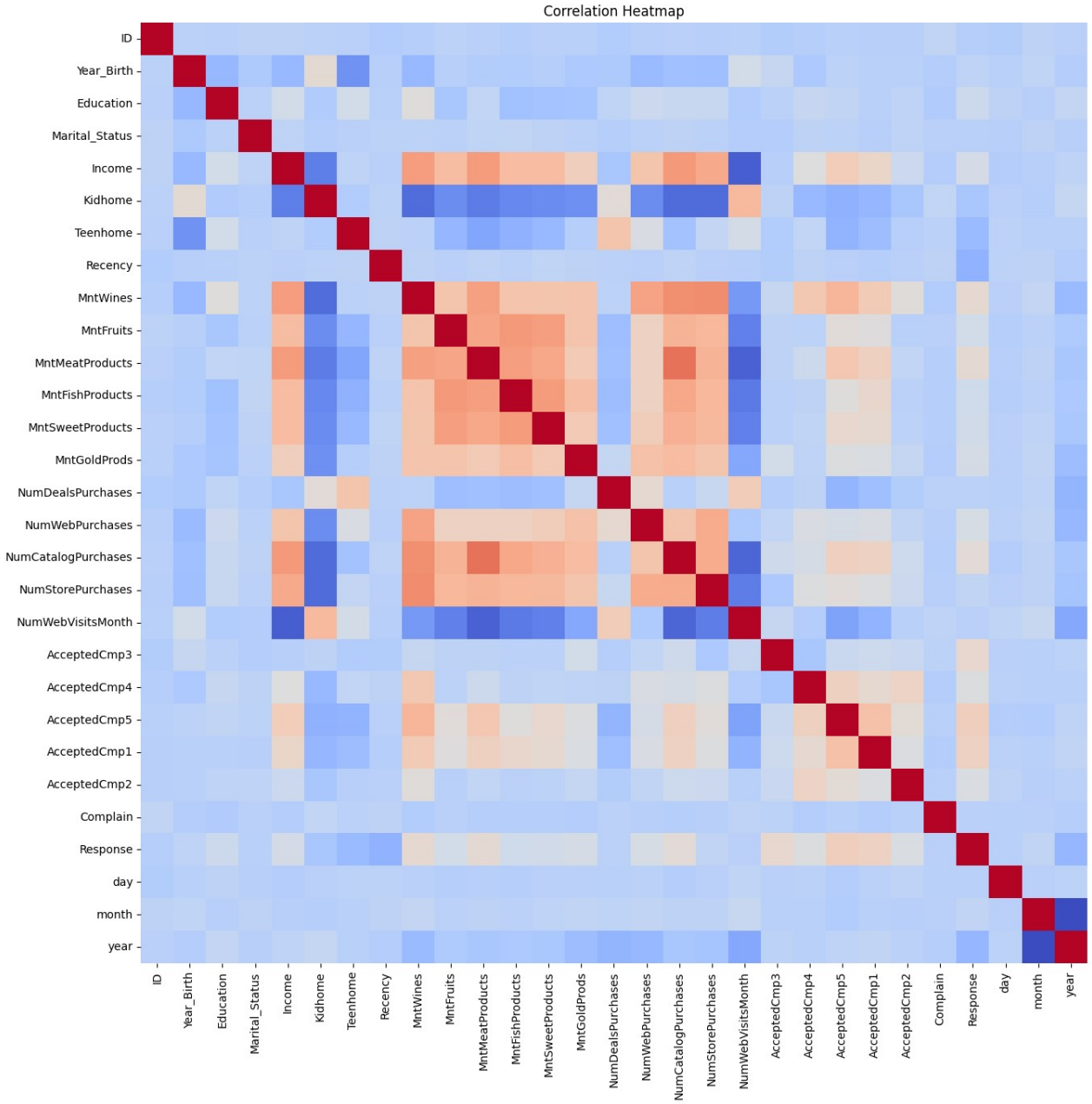
```
    plt.subplot(2, 2, i + 1)
    sb.countplot(x=col, hue='Response', data=df)
    plt.title(f"Response Distribution by {col}")
plt.tight_layout()
plt.show()
```




```
# Loop over each categorical (object) column
for col in objects:
      # Initialize LabelEncoder for each column
      le = LabelEncoder()

      # Apply label encoding
      df[col] = le.fit_transform(df[col])

      # Print the mapping for the current column
      print(f"Label encoding for {col}:")
      label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
      print(label_mapping)
      print("-" * 50)
```

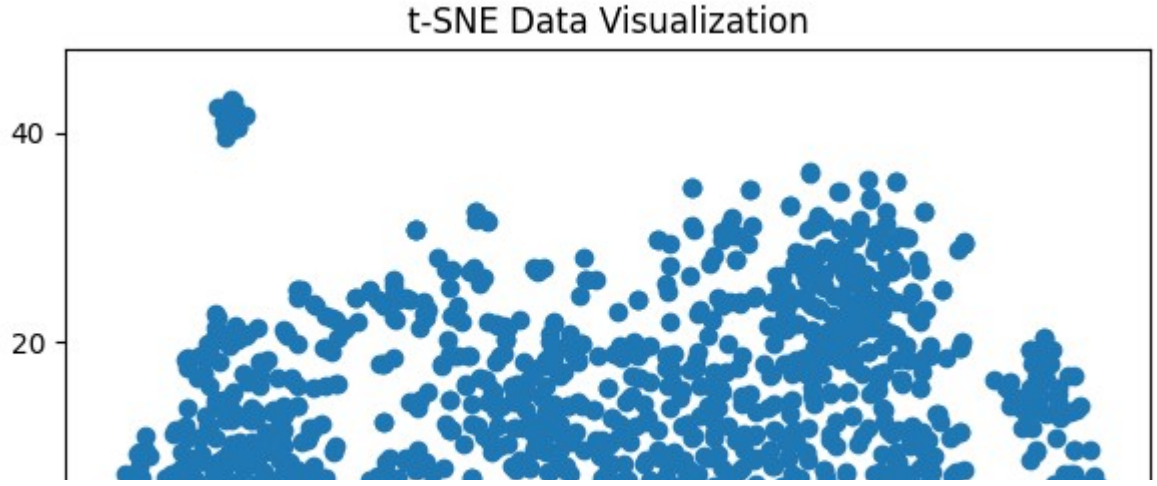
```
Label encoding for Education:
{'2n Cycle': 0, 'Basic': 1, 'Graduation': 2, 'Master': 3, 'PhD': 4}
--------------------------------------------------
Label encoding for Marital_Status:
{'Absurd': 0, 'Alone': 1, 'Divorced': 2, 'Married': 3, 'Single': 4, 'Together': 5,
--------------------------------------------------
```

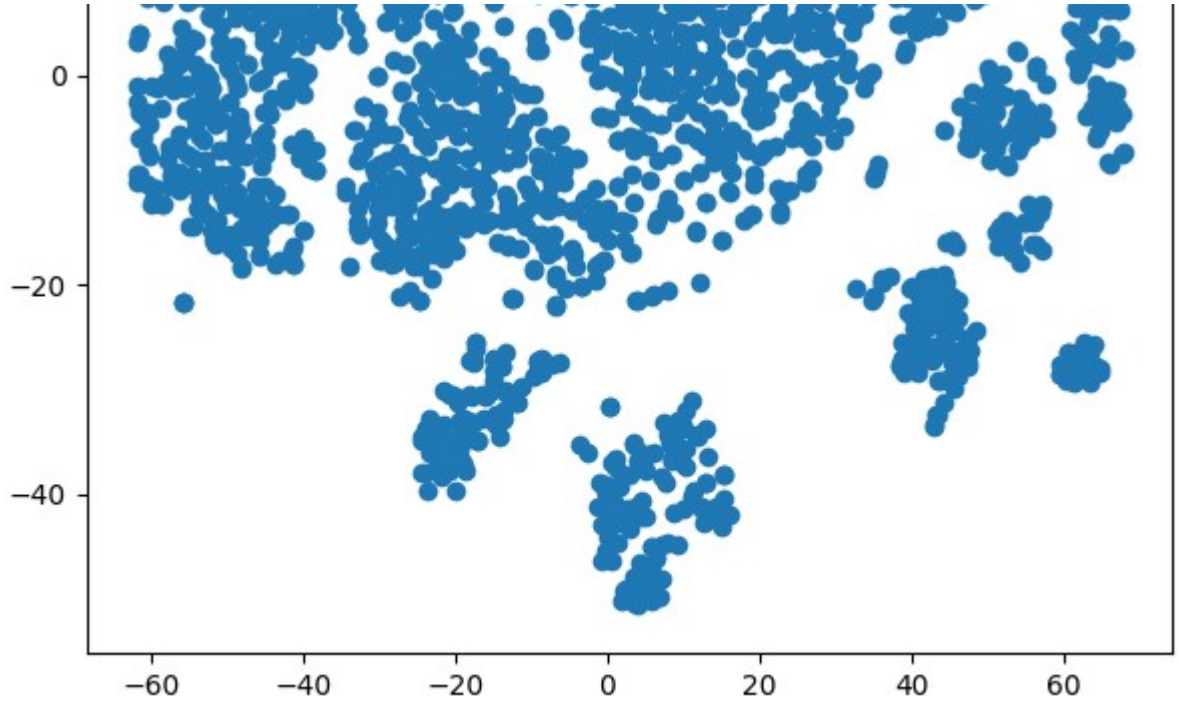
```
plt.figure(figsize=(15, 15))
sb.heatmap(df.corr(), annot=False, cmap='coolwarm', cbar=False)
plt.title("Correlation Heatmap")
plt.show()
```



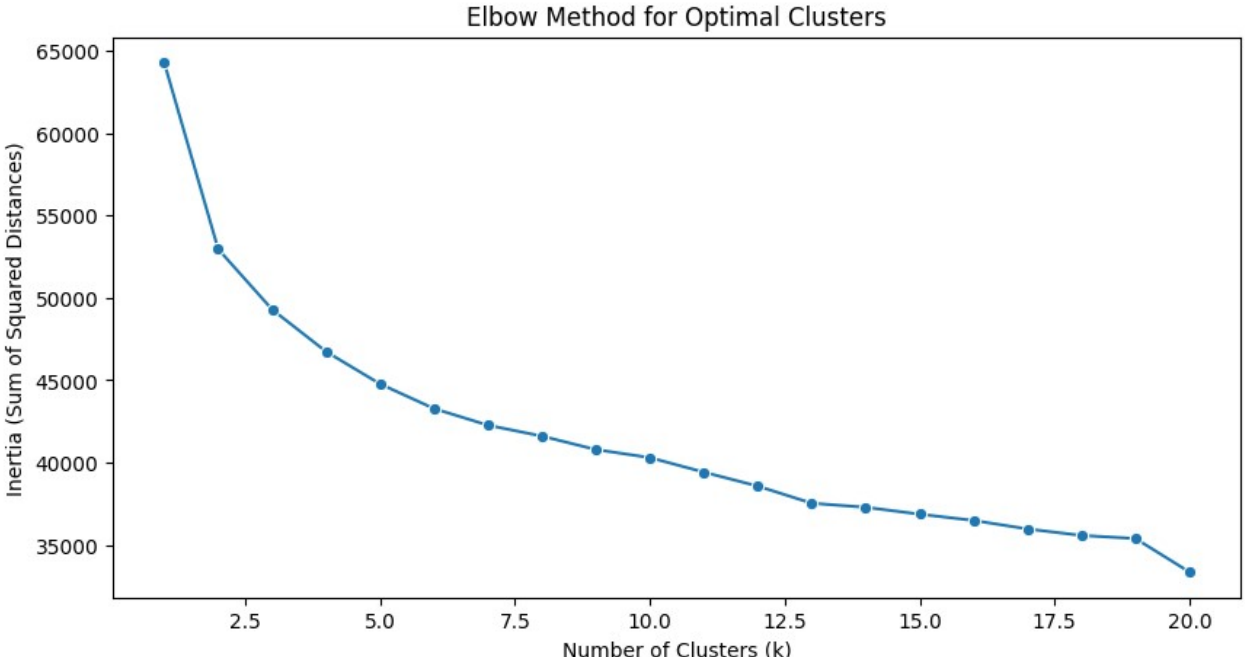
```
#standardize the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)
```

```
# Apply t-SNE
tsne = TSNE(n_components=2, random_state=0)
tsne_data = tsne.fit_transform(scaled_data)

# Scatter plot of t-SNE data
plt.figure(figsize=(7, 7))
plt.scatter(tsne_data[:, 0], tsne_data[:, 1])
plt.title("t-SNE Data Visualization")
plt.show()
```

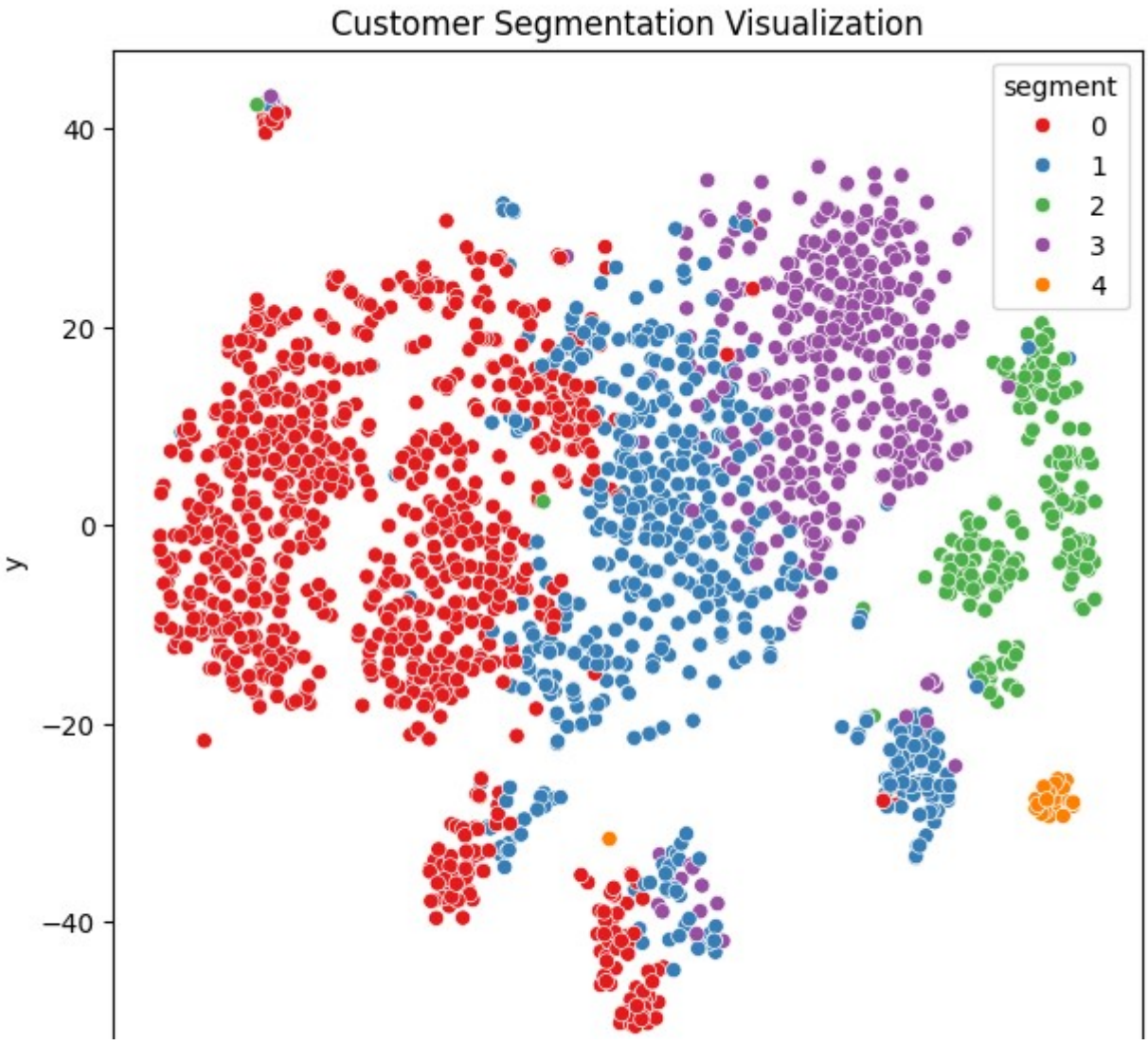


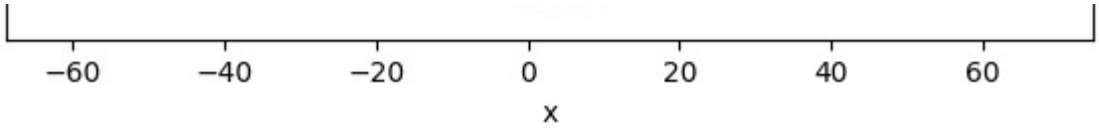



```
#finding the optimal number of clusters
errors = []
for n_clusters in range(1, 21):
      model = KMeans(init='k-means++', n_clusters=n_clusters, max_iter=500, random_sta
      model.fit(scaled_data)
      errors.append(model.inertia_)

#plotting elbow curve
plt.figure(figsize=(10, 5))
sb.lineplot(x=range(1, 21), y=errors, marker='o')
plt.title("Elbow Method for Optimal Clusters")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia (Sum of Squared Distances)")
plt.show()
```



```
# Train KMeans with optimal clusters
optimal_k = 5
model = KMeans(init='k-means++', n_clusters=optimal_k, max_iter=500, random_state=22)
segments = model.fit_predict(scaled_data)

# Add segments to DataFrame
df_tsne = pd.DataFrame({'x': tsne_data[:, 0], 'y': tsne_data[:, 1], 'segment': segment

# Scatter plot with segmentation
plt.figure(figsize=(7, 7))
sb.scatterplot(x='x', y='y', hue='segment', palette='Set1', data=df_tsne)
plt.title("Customer Segmentation Visualization")
plt.show()
```





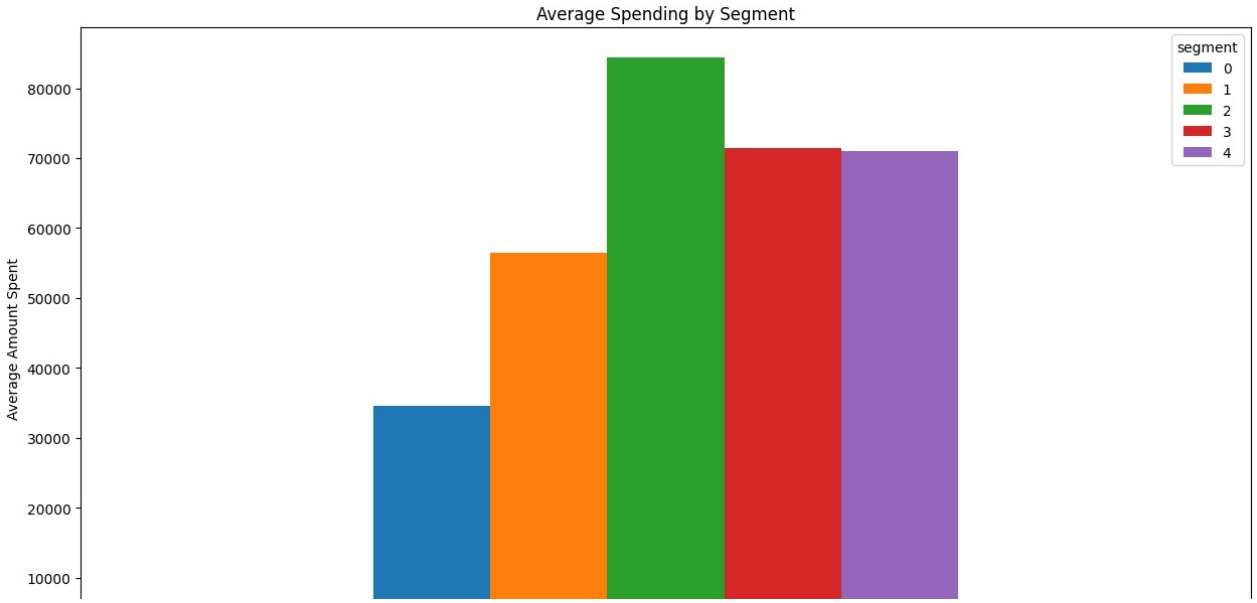

```
# Add segments to the original DataFrame
df['segment'] = segments
# Group data by segments for descriptive analysis
segment_summary = df.groupby('segment').mean()
segment_summary
```

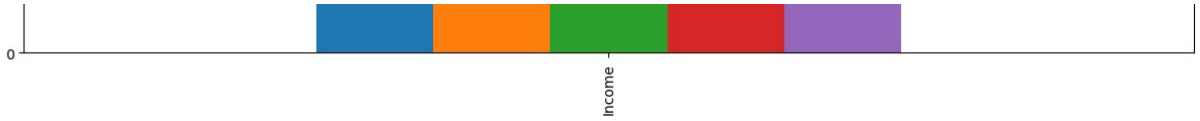
segment	ID	Year_Birth	Education	Marital_Status	Income	Kidhome
0	5619.848849	1971.942943	2.254254	3.709710	34589.510511	0.802803
1	5431.747731	1963.929220	2.666062	3.716878	56529.713249	0.255898
2	5545.189744	1968.435897	2.369231	3.743590	84429.558974	0.051282
3	5760.045351	1968.074830	2.367347	3.755102	71387.090703	0.054422
4	5172.566667	1968.133333	2.600000	3.900000	71054.833333	0.066667

5 rows × 29 columns

```
# Visualize average spending by segment
plt.figure(figsize=(15, 8))
segment_summary[floats].T.plot(kind='bar', figsize=(15, 8))
plt.title("Average Spending by Segment")
plt.ylabel("Average Amount Spent")
plt.show()
```

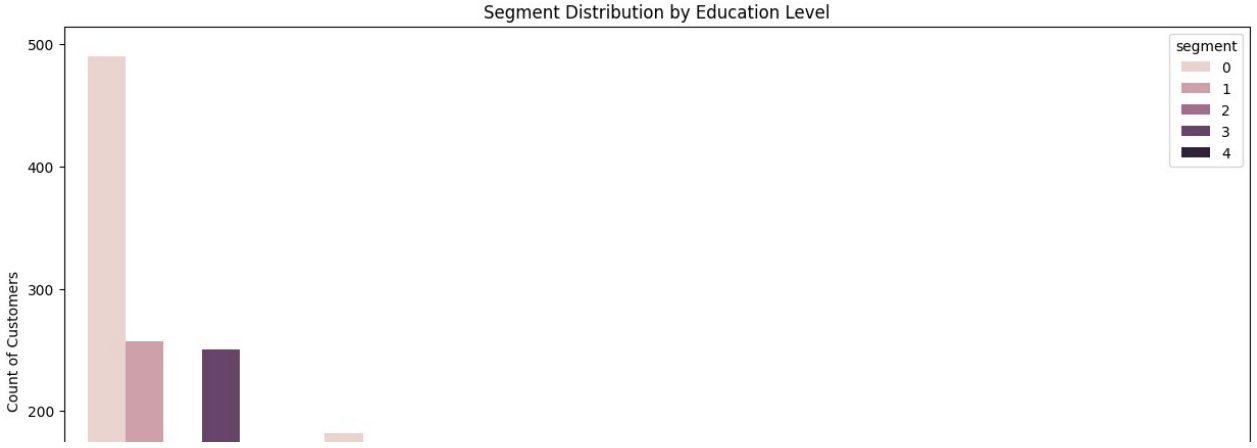
```
<Figure size 1500x800 with 0 Axes>
```

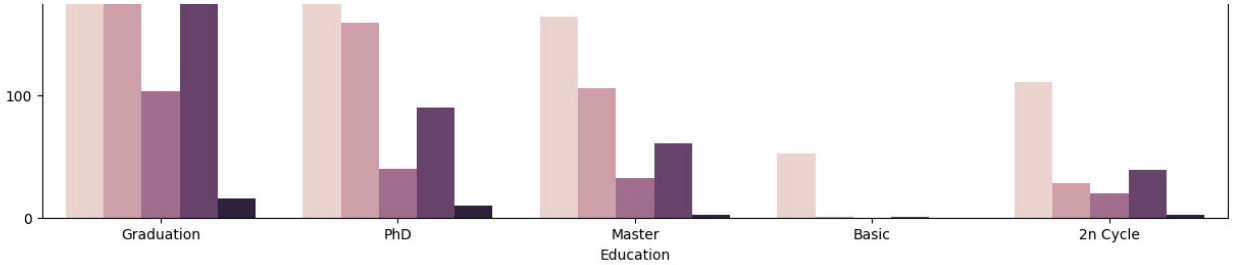




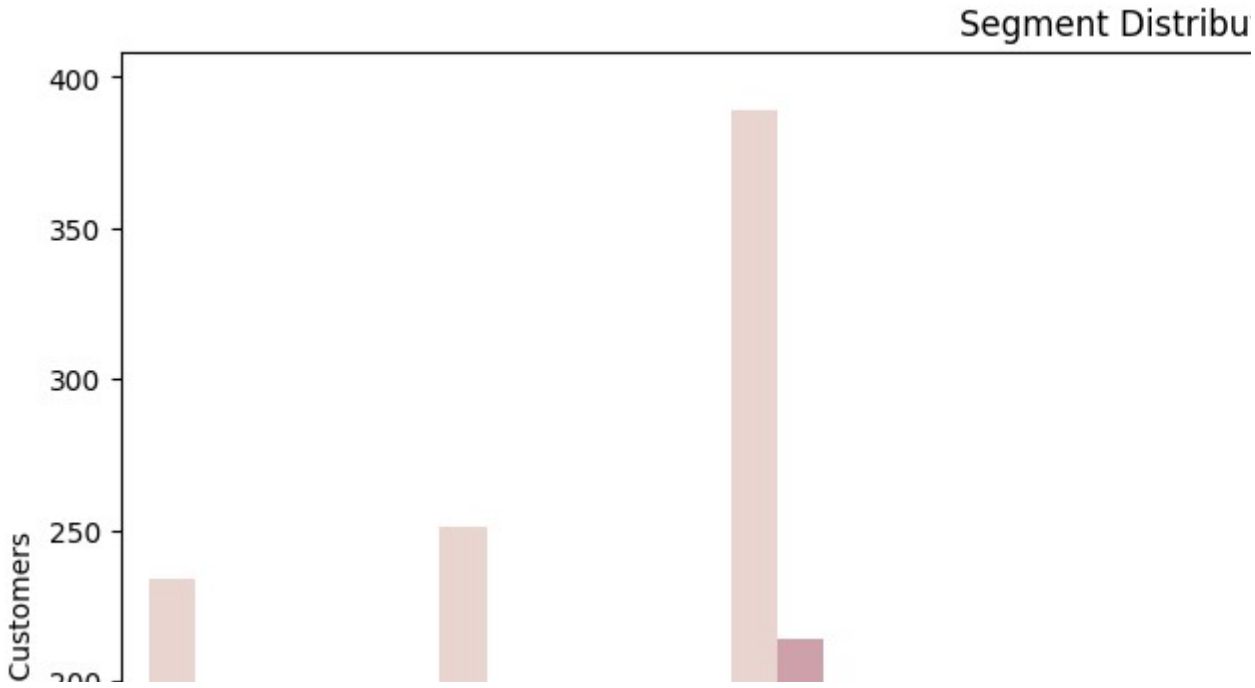

```
# Mapping for Education levels
education_mapping = {0: '2n Cycle', 1: 'Basic', 2: 'Graduation', 3: 'Master', 4: 'PhD'
df['Education'] = df['Education'].map(education_mapping)
# Mapping for Marital Status levels
marital_mapping = {0: 'Absurd', 1: 'Alone', 2: 'Divorced', 3: 'Married', 4: 'Single',5
df['Marital_Status'] = df['Marital_Status'].map(marital_mapping)
```

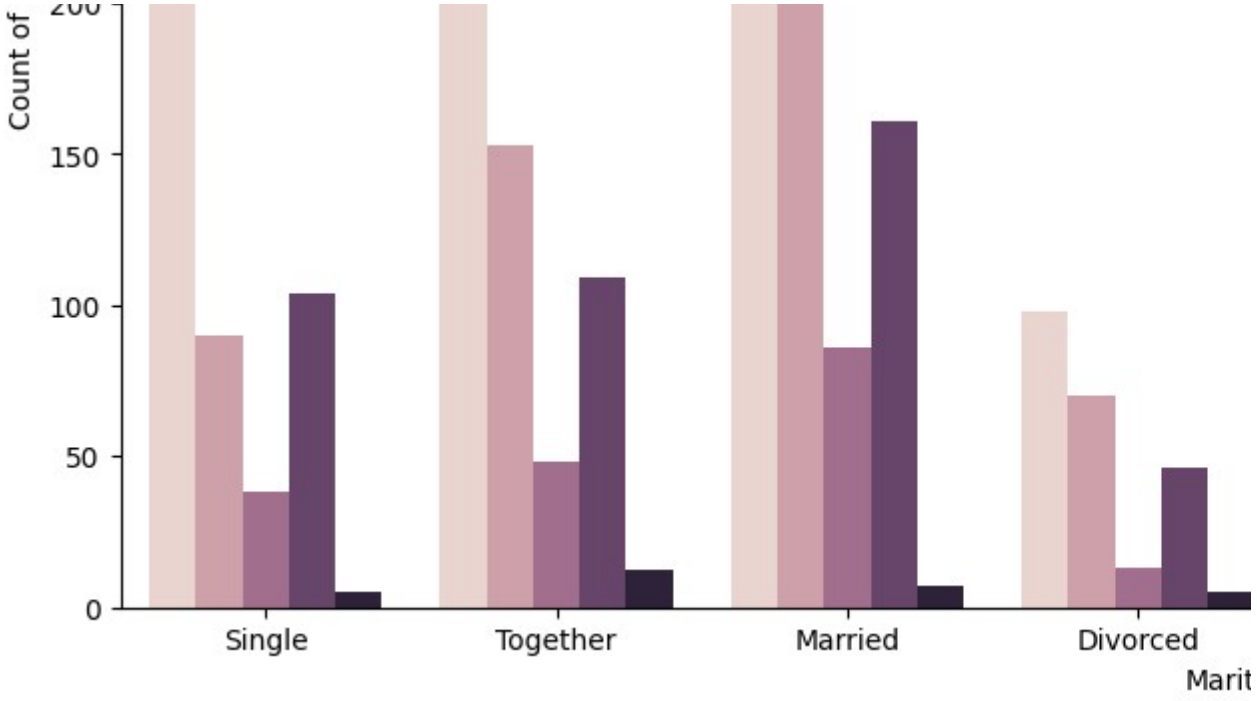
```
# Visualize segment distribution by Education
plt.figure(figsize=(15, 8))
sb.countplot(x='Education', hue='segment', data=df)
plt.title("Segment Distribution by Education Level")
plt.ylabel("Count of Customers")
plt.show()
```






```
# Visualize segment distribution by Marital Status
plt.figure(figsize=(15, 8))
sb.countplot(x='Marital_Status', hue='segment', data=df)
plt.title("Segment Distribution by Marital Status")
plt.ylabel("Count of Customers")
plt.show()
```




Count of
200
150
100
50
0
Single
Together
Married
Divorced
Marit