Customer Segmentation Analysis

This project examines sample data from Kaggle on mall customers demographics including Age, Gender, Income, and Spending Score to create a segmentation analysis applying K-means Clustering and Principal Component Analysis. Identifies and segments customers to provide insights into Age, Generations, Income, Income Brackets, Gender, and Spending Score with a focus on understanding who customers are, how spending varies across customer demographics, and how customers vary by segment.

Libraries

As a first step, import required dependencies for analysis

In [1]:

```
import numpy as np
import pandas as pd
import scipy
import scipy.stats as stats
import statistics as st
import math

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
import seaborn as sn
sn.set_theme(style="darkgrid")
sn.set()

from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

Import Data

Then load csv file into Pandas DataFrame

In [2]:

```
df_segmentation = pd.read_csv('    ')
df_segmentation
```

Out[2]:

	CustomerID	Gender	Age	Income	Spending_Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77

	CustomerID	Gender	Age	Income	Spending_Score
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

Explore Data

Identify number of imported rows as a first step in data exploration

In [3]:
```
len(df_segmentation) #200 rows imported
```

Out[3]: 200

Identify number of columns imported

In [4]:
```
len(df_segmentation.columns) #5 columns imported
```

Out[4]: 5

Number of rows and columns

In [5]:
```
df_segmentation.shape #200 rows and 5 columns imported
```

Out[5]: (200, 5)

Identify distinct columns

In [6]:
```
df_segmentation.columns
```

Out[6]:
```
Index(['CustomerID', 'Gender', 'Age', 'Income', 'Spending_Score'], dtype='object')
```

Columns Imported include:

CustomerID - customer's unique identifier

Gender - binary, female/male

Age - numberical value representative of customer's age

Income - annual income

SpendingScore - assigned score based on defined parameters like customer behavior & purchasing data

View first five rows in dataset

In [7]:
```
df_segmentation.head(5)
```

Out[7]:

	CustomerID	Gender	Age	Income	Spending_Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

View last five rows in dataset

In [8]:
```
df_segmentation.tail(5)
```

Out[8]:

	CustomerID	Gender	Age	Income	Spending_Score
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

Identify data types

In [9]:
```
df_segmentation.dtypes
```

Out[9]:
```
CustomerID         int64
Gender            object
Age                int64
Income             int64
Spending_Score     int64
dtype: object
```

Identify null values

In [10]:
```
df_segmentation.isnull().sum()
```

Out[10]:
```
CustomerID        0
Gender            0
Age               0
Income            0
Spending_Score    0
dtype: int64
```

In [11]:

```
df_segmentation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   CustomerID      200 non-null    int64 
 1   Gender          200 non-null    object
 2   Age             200 non-null    int64 
 3   Income          200 non-null    int64 
 4   Spending_Score  200 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

Descriptive Statistics

In [12]:

```
df_segmentation.describe()
```

Out[12]:

	CustomerID	Age	Income	Spending_Score
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

In [13]:

```
df_segmentation.describe().transpose()
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
CustomerID	200.0	100.50	57.879185	1.0	50.75	100.5	150.25	200.0
Age	200.0	38.85	13.969007	18.0	28.75	36.0	49.00	70.0
Income	200.0	60.56	26.264721	15.0	41.50	61.5	78.00	137.0
Spending_Score	200.0	50.20	25.823522	1.0	34.75	50.0	73.00	99.0

Identify Outliers

Age

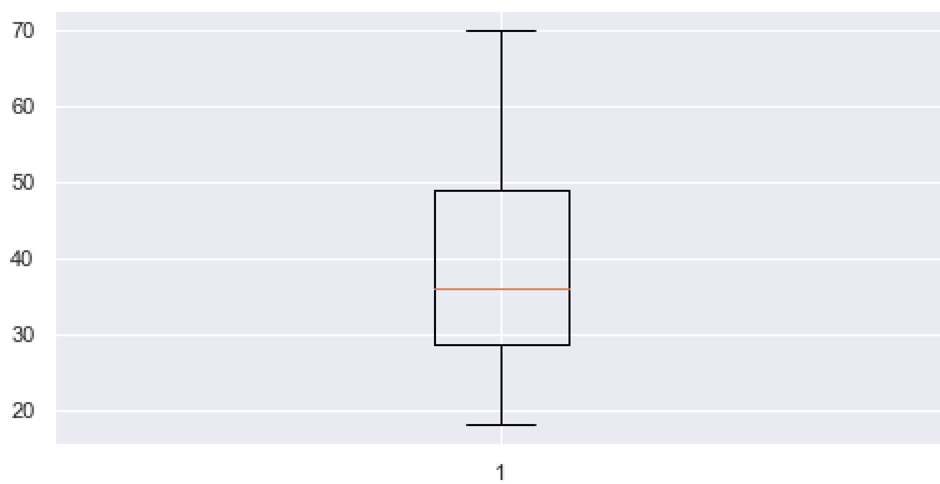
In [14]:

```
#Distribution
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,4
sn.distplot(df_segmentation["Age"], bins=10)
```

Out[14]: <AxesSubplot:xlabel='Age', ylabel='Density'>

In [15]:

```
#Boxplot
plt.boxplot(df_segmentation['Age'])
plt.show()
```



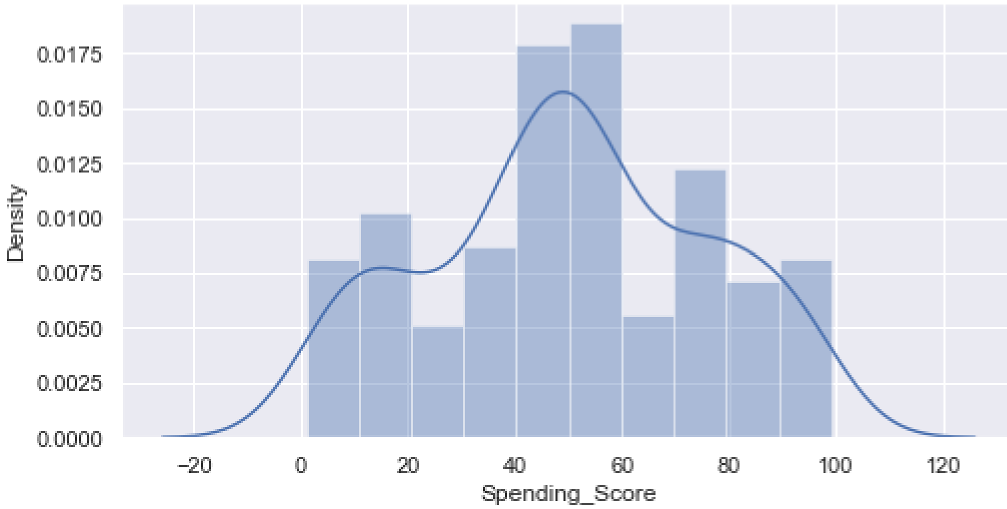
Age appears to have no outliers

Spending Score

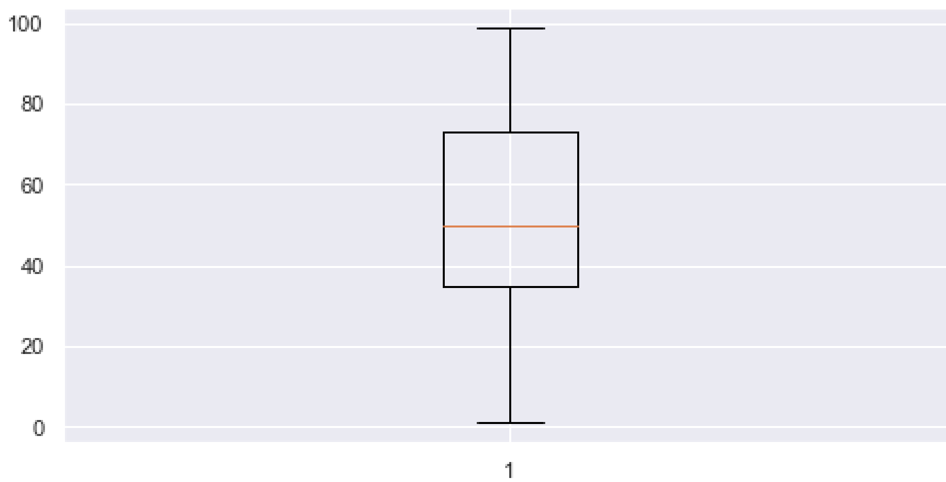
In [16]:

```
#Distribution
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,4
sn.distplot(df_segmentation["Spending_Score"], bins=10)
```

Out[16]: <AxesSubplot:xlabel='Spending_Score', ylabel='Density'>



In [17]:
```
#Spending Score
plt.boxplot(df_segmentation['Spending_Score'])
plt.show()
```

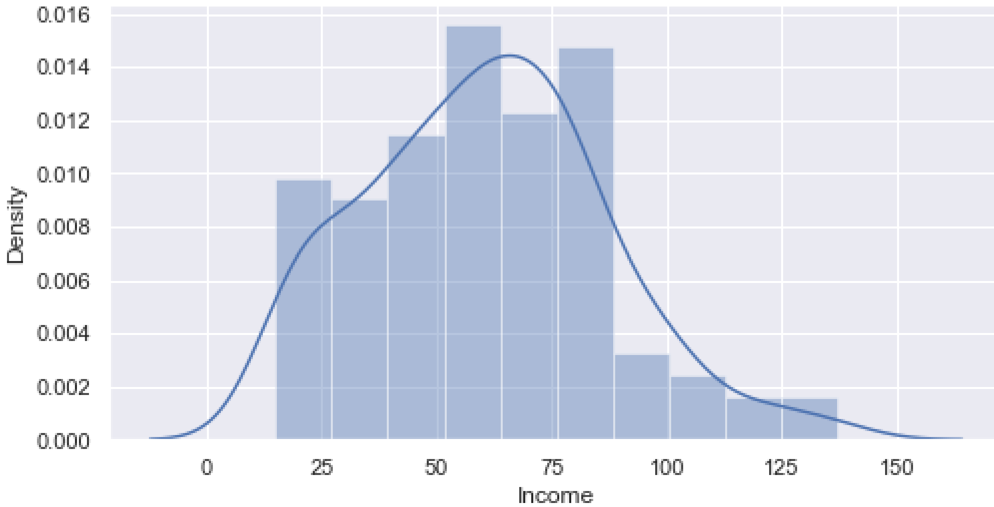


Spending Score appears to have no outliers

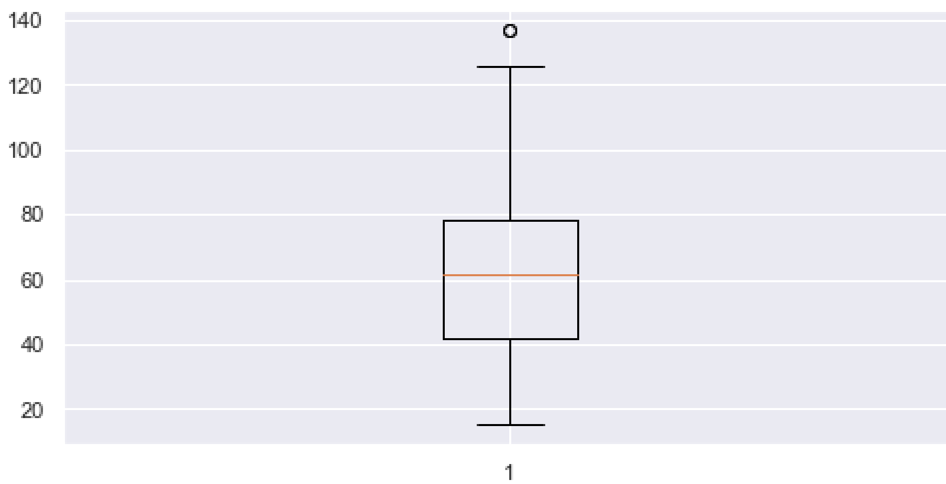
Income

In [18]:
```
#Distribution
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,4
sn.distplot(df_segmentation["Income"], bins=10)
```

Out[18]:
```
<AxesSubplot:xlabel='Income', ylabel='Density'>
```



In [19]:
```
#Boxplot
plt.boxplot(df_segmentation['Income'])
plt.show()
```



Income appears to have outliers present. As a next step, remove outliers..

Inter Quatertile Range (IQR)

In [20]:
```
Q1 = df_segmentation['Income'].quantile(0.25)
Q1
```

Out[20]: 41.5

In [21]:
```
Q3 = df_segmentation['Income'].quantile(0.75)
Q3
```

Out[21]: 78.0

In [22]:
```
IQR = Q3 - Q1
```

Identify Outliers

In [23]:

```
outliers = df_segmentation[(df_segmentation['Income'] < (Q1 - 1.5 * IQR)) | (df_segment
outliers
```

Out[23]:

	CustomerID	Gender	Age	Income	Spending_Score
198	199	Male	32	137	18
199	200	Male	30	137	83

In [24]:

```
print("Number of outliers in the Income column:", len(outliers))
```

```
Number of outliers in the Income column: 2
```

Remove Outliers

In [25]:

```
df_segmentation = df_segmentation[~((df_segmentation['Income'] < (Q1 - 1.5 * IQR)) | (d
```

In [26]:

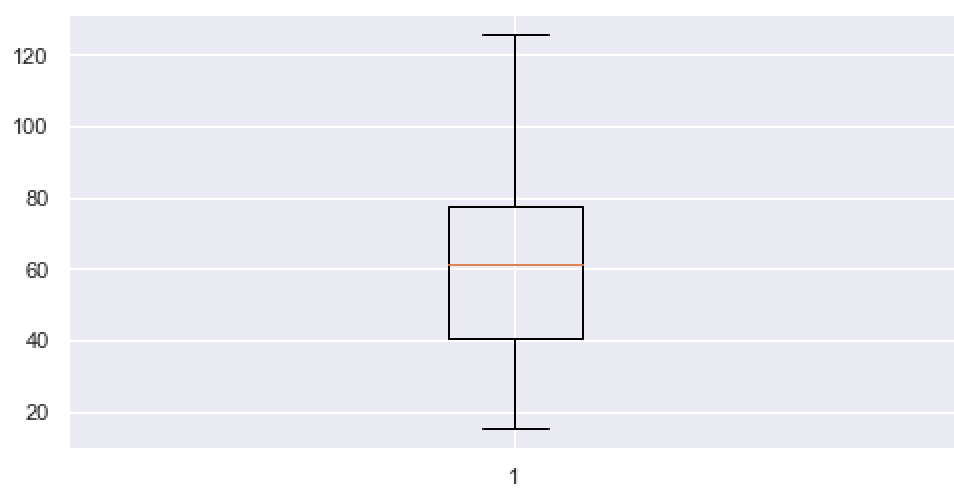
```
print("Updated shape of the dataframe:", df_segmentation.shape)
```

```
Updated shape of the dataframe: (198, 5)
```

Confirm outliers have been removed

In [27]:

```
#Boxplot
plt.boxplot(df_segmentation['Income'])
plt.show()
```



Feature Engineering

Create dummy variable for Male Gender

In [28]:

```
df_segmentation['Gender'].unique()
```

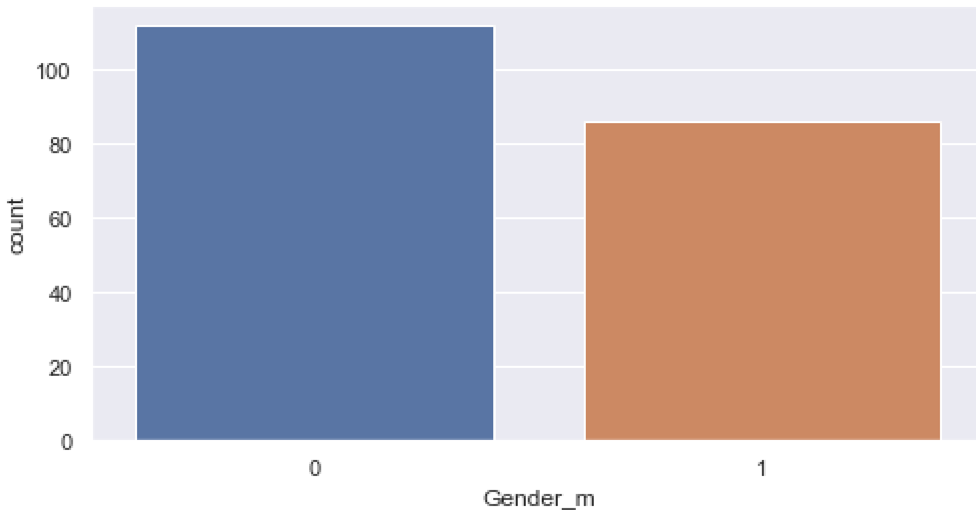
Out[28]:

```
array(['Male', 'Female'], dtype=object)
```

In [29]:
```python
def Gender_m(Gender):
    if Gender in ['Male']:
        return 1
    elif Gender in ['Female']:
        return 0
df_segmentation['Gender_m'] = df_segmentation['Gender'].apply(Gender_m)
df_segmentation['Gender_m'].unique()
```

Out[29]:
```
array([1, 0], dtype=int64)
```

In [30]:
```python
sn.countplot(x="Gender_m",data=df_segmentation);
```

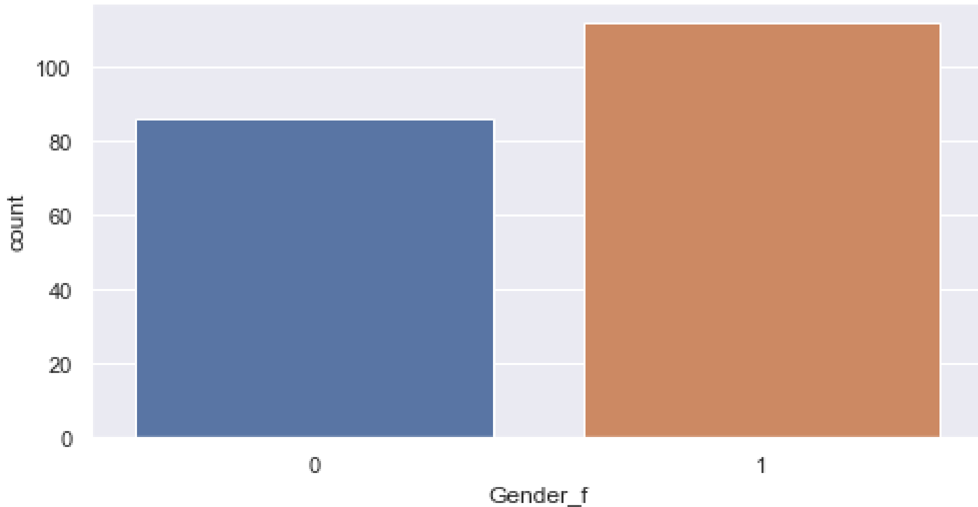


Create dummy variable for Female Gender

In [31]:
```python
def Gender_f(Gender):
    if Gender in ['Male']:
        return 0
    elif Gender in ['Female']:
        return 1
df_segmentation['Gender_f'] = df_segmentation['Gender'].apply(Gender_f)
df_segmentation['Gender_f'].unique()
```

Out[31]:
```
array([0, 1], dtype=int64)
```

In [32]:
```python
sn.countplot(x="Gender_f",data=df_segmentation);
```



Create Generations from Age

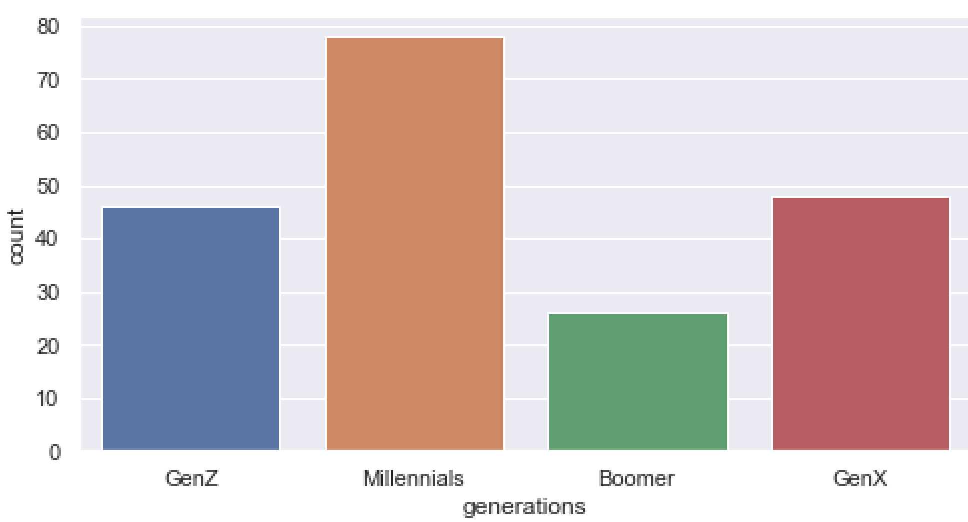
In [33]:

```
generations = []
for i in df_segmentation["Age"]:
    if i >=58:
        generations.append("Boomer")
    elif i < 58 and i >=43:
        generations.append("GenX")
    elif i < 43 and i >=28:
        generations.append("Millennials")
    elif i < 28 and i >=10:
        generations.append("GenZ")
df_segmentation["generations"] = generations
df_segmentation['generations'].unique()
```

Out[33]: `array(['GenZ', 'Millennials', 'Boomer', 'GenX'], dtype=object)`

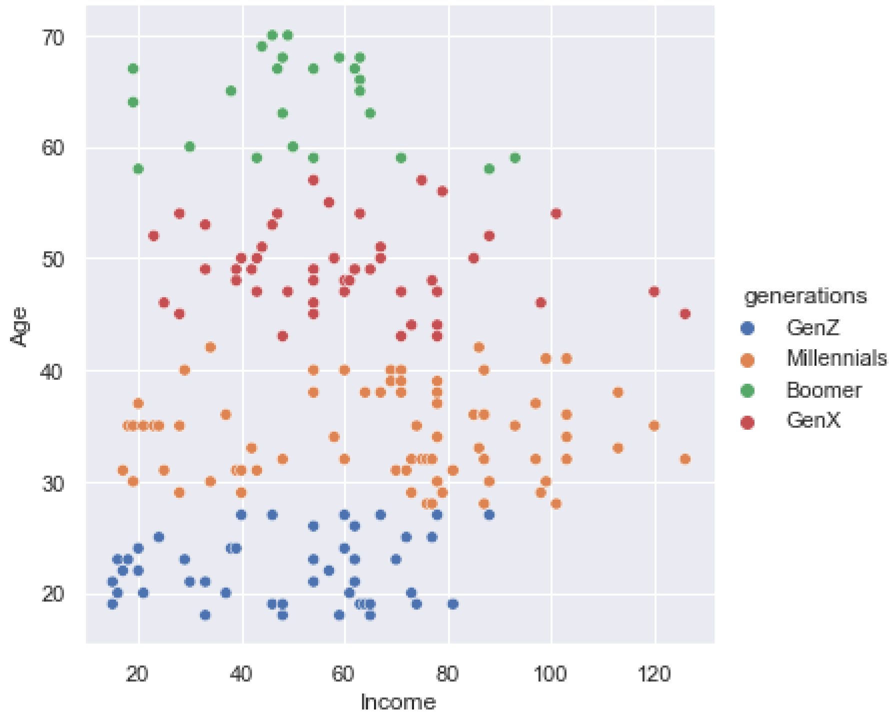
In [34]:

```
sn.countplot(x="generations",data=df_segmentation);
```



In [35]:

```
sn.relplot(data=df_segmentation, x="Income", y="Age", hue="generations",);
```



Create Income Brackets from Income

In [36]:

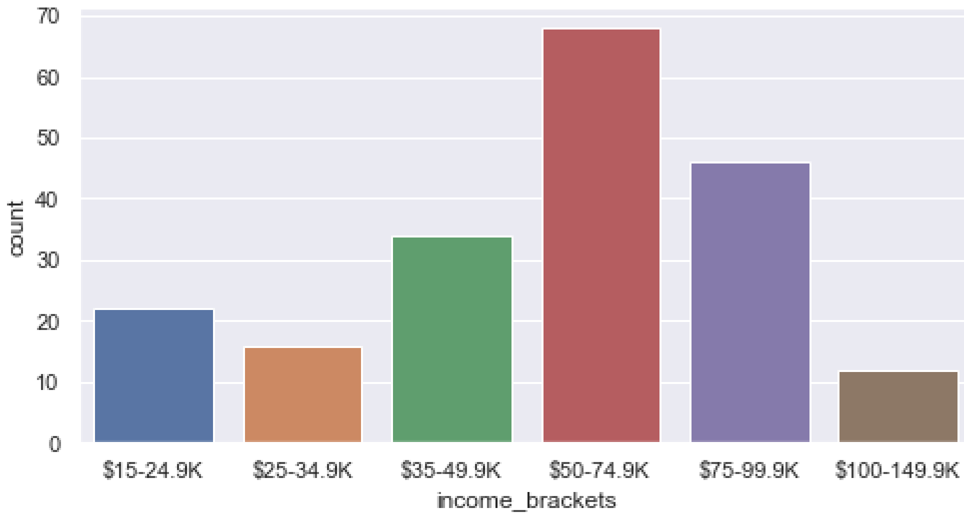
```
income_brackets = []
for i in df_segmentation["Income"]:
    if i >= 150:
        income_brackets.append("$150K+")
    elif i < 150 and i >=100:
        income_brackets.append("$100-149.9K")
    elif i < 100 and i >=75:
        income_brackets.append("$75-99.9K")
    elif i < 75 and i >= 50:
        income_brackets.append("$50-74.9K")
    elif i < 50 and i >=35:
        income_brackets.append("$35-49.9K")
    elif i < 35 and i >=25:
            income_brackets.append("$25-34.9K")
    elif i < 25 and i >=15:
        income_brackets.append("$15-24.9K")
    elif i < 15:
        income_brackets.append("<$15K")
df_segmentation["income_brackets"] = income_brackets
df_segmentation['income_brackets'].unique()
```

Out[36]:

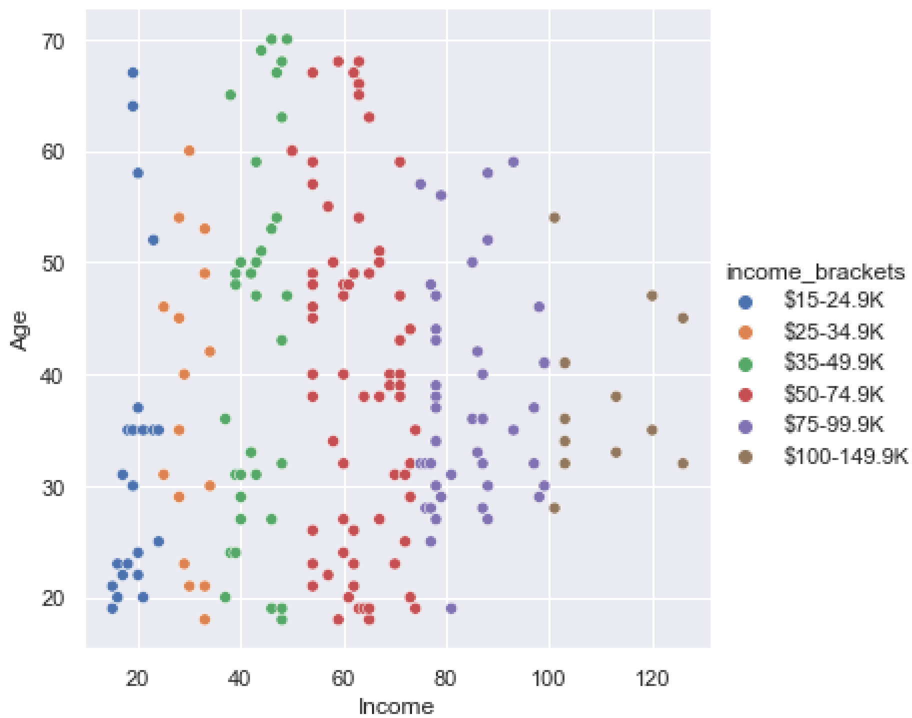
```
array(['$15-24.9K', '$25-34.9K', '$35-49.9K', '$50-74.9K', '$75-99.9K',
       '$100-149.9K'], dtype=object)
```

In [37]:

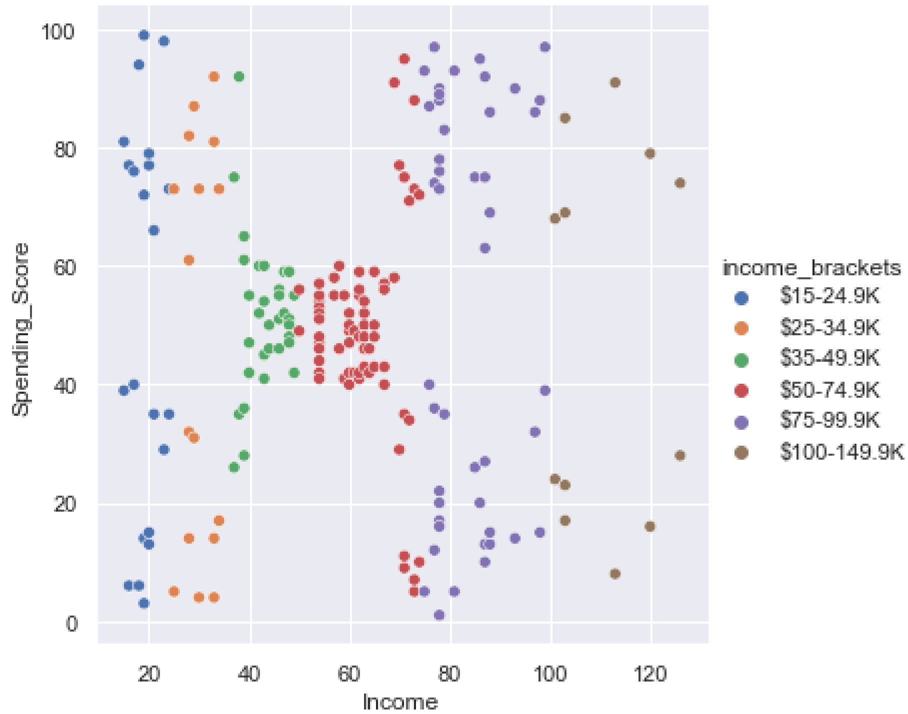
```
sn.countplot(x="income_brackets",data=df_segmentation);
```



In [38]:
```
sn.relplot(data=df_segmentation, x="Income", y="Age", hue="income_brackets",);
```



In [39]:
```
sn.relplot(data=df_segmentation, x="Income", y="Spending_Score", hue="income_brackets",
```



Explore Updated Data

Identify distinct columns

In [40]:
```
df_segmentation.columns
```

Out[40]:
```
Index(['CustomerID', 'Gender', 'Age', 'Income', 'Spending_Score', 'Gender_m',
       'Gender_f', 'generations', 'income_brackets'],
      dtype='object')
```

View first five rows

In [41]:
```
df_segmentation.head(5)
```

Out[41]:

	CustomerID	Gender	Age	Income	Spending_Score	Gender_m	Gender_f	generations	income_brack
0	1	Male	19	15	39	1	0	GenZ	$15-24.
1	2	Male	21	15	81	1	0	GenZ	$15-24.
2	3	Female	20	16	6	0	1	GenZ	$15-24.
3	4	Female	23	16	77	0	1	GenZ	$15-24.
4	5	Female	31	17	40	0	1	Millennials	$15-24.

View last five rows

In [42]:
```
df_segmentation.tail(5)
```

Out[42]:

	CustomerID	Gender	Age	Income	Spending_Score	Gender_m	Gender_f	generations	income_bra
193	194	Female	38	113	91	0	1	Millennials	$100-1
194	195	Female	47	120	16	0	1	GenX	$100-1
195	196	Female	35	120	79	0	1	Millennials	$100-1
196	197	Female	45	126	28	0	1	GenX	$100-1
197	198	Male	32	126	74	1	0	Millennials	$100-1

Identify number of columns, column labels, null values and column types

In [43]:

```
df_segmentation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 198 entries, 0 to 197
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   CustomerID       198 non-null    int64 
 1   Gender           198 non-null    object
 2   Age              198 non-null    int64 
 3   Income           198 non-null    int64 
 4   Spending_Score   198 non-null    int64 
 5   Gender_m         198 non-null    int64 
 6   Gender_f         198 non-null    int64 
 7   generations      198 non-null    object
 8   income_brackets  198 non-null    object
dtypes: int64(6), object(3)
memory usage: 23.6+ KB
```

Descriptive Statistics

In [44]:

```
df_segmentation.describe().transpose()
```

Out[44]:

	count	mean	std	min	25%	50%	75%	max
CustomerID	198.0	99.500000	57.301832	1.0	50.25	99.5	148.75	198.0
Age	198.0	38.929293	14.016852	18.0	28.25	36.0	49.00	70.0
Income	198.0	59.787879	25.237259	15.0	40.50	61.0	77.75	126.0
Spending_Score	198.0	50.196970	25.746846	1.0	35.00	50.0	72.75	99.0
Gender_m	198.0	0.434343	0.496927	0.0	0.00	0.0	1.00	1.0
Gender_f	198.0	0.565657	0.496927	0.0	0.00	1.0	1.00	1.0

View count of unique values per column for categorical variables

In [45]:

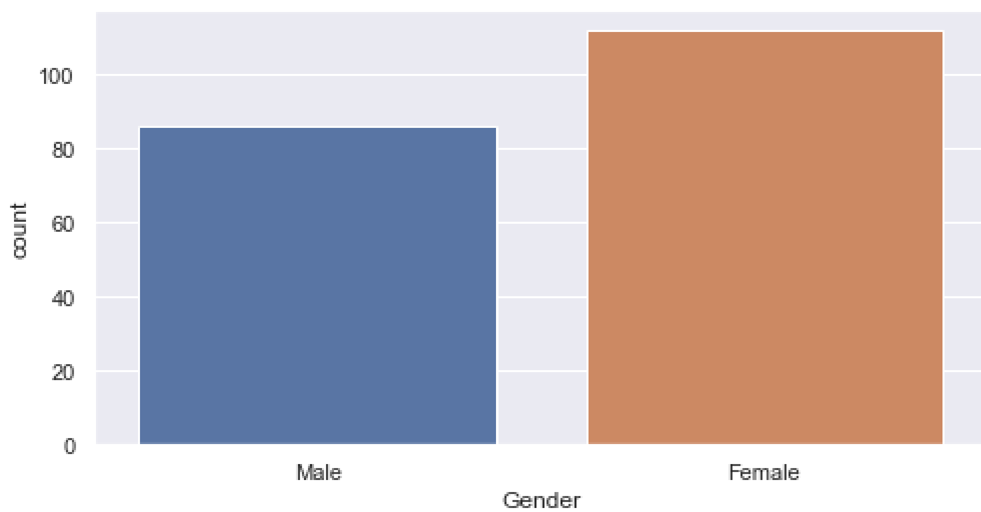
```
categorical_variables = ["Gender","generations","income_brackets"]
#Excluded columns: Gender_n, Age, Income, Gender_m, Gender_f
for column in categorical_variables:
    print(df_segmentation[column].value_counts())
    print("-" * 40)
```

```
Female    112
Male       86
Name: Gender, dtype: int64
----------------------------------------
Millennials    78
GenX           48
GenZ           46
Boomer         26
Name: generations, dtype: int64
----------------------------------------
$50-74.9K      68
$75-99.9K      46
$35-49.9K      34
$15-24.9K      22
$25-34.9K      16
$100-149.9K    12
Name: income_brackets, dtype: int64
----------------------------------------
```

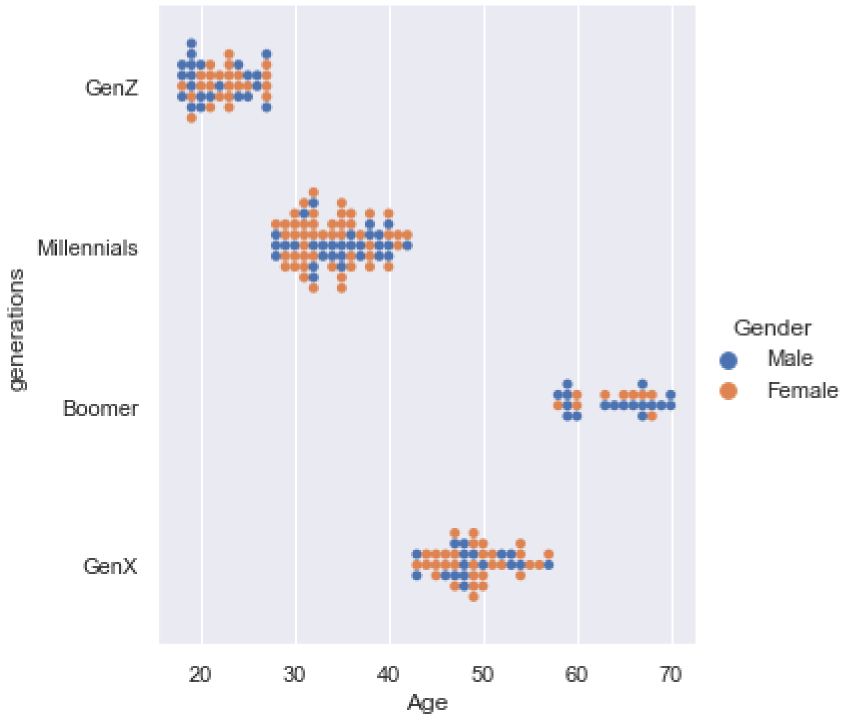
Data Visualization

Gender

In [46]:
```
sn.countplot(x="Gender",data=df_segmentation);
```

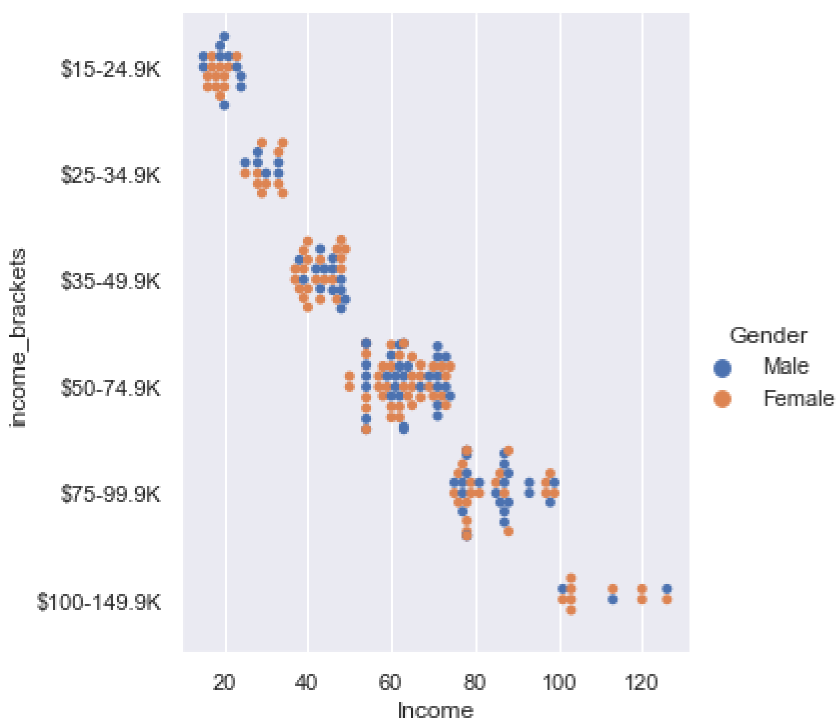


In [47]:
```
sn.catplot(data=df_segmentation, x="Age", y="generations", hue="Gender", kind="swarm");
```



In [48]:

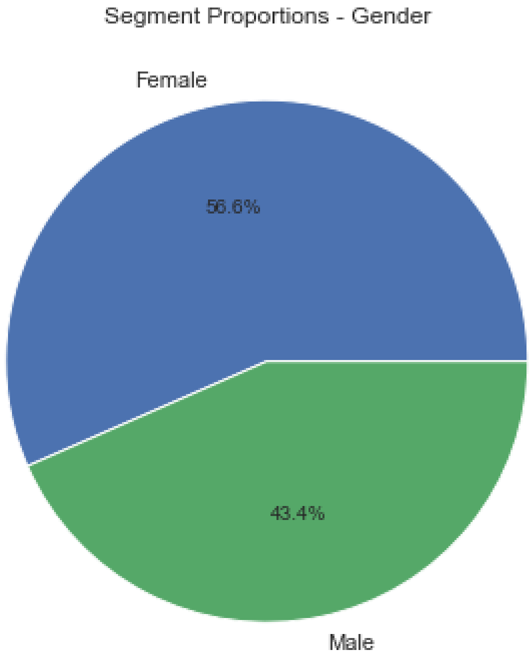
```
sn.catplot(data=df_segmentation, x="Income", y="income_brackets", hue="Gender", kind="s
```



Gender Proportion

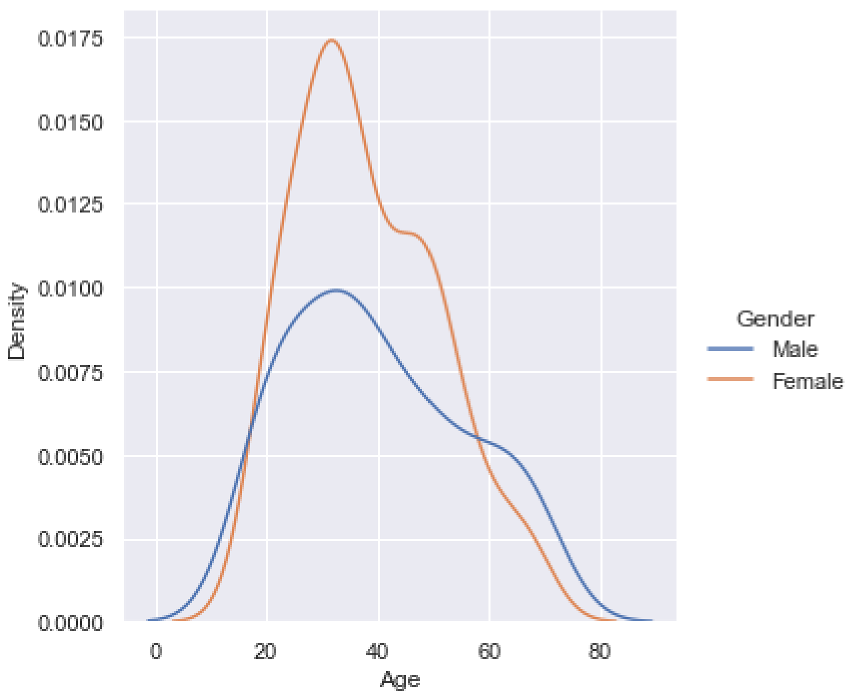
In [49]:

```
seg_prop_g = df_segmentation[['CustomerID','Gender']].groupby(['Gender']).count() / df_
plt.figure(figsize = (9, 6))
plt.pie(seg_prop_g['CustomerID'],
        labels = ['Female', 'Male'],
        autopct = '%1.1f%%',
        colors = ('b', 'g'))
plt.title('Segment Proportions - Gender');
```

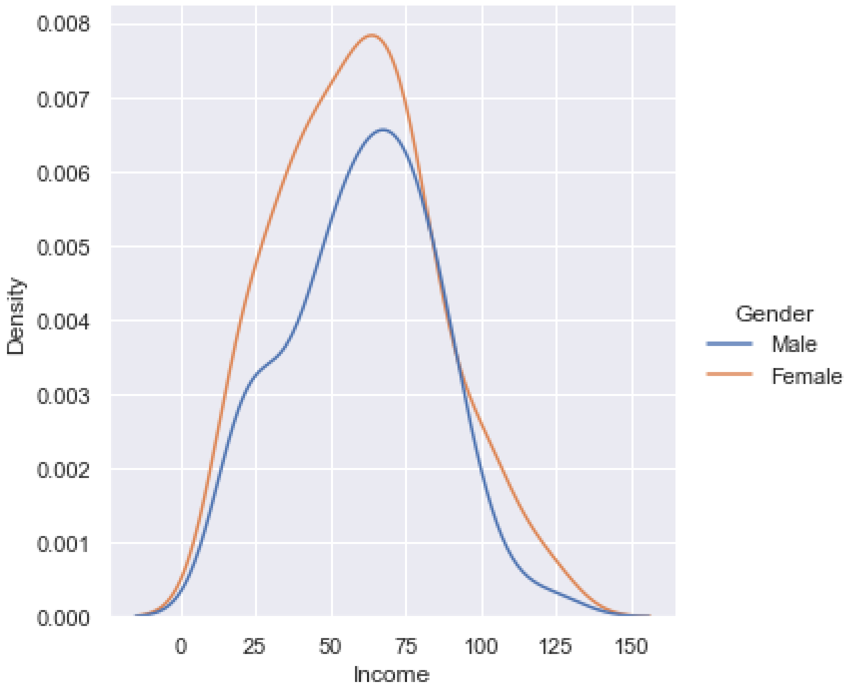


Sample skews slights Female

In [50]:
```
sn.displot(data=df_segmentation, x="Age", hue="Gender", kind="kde");
```



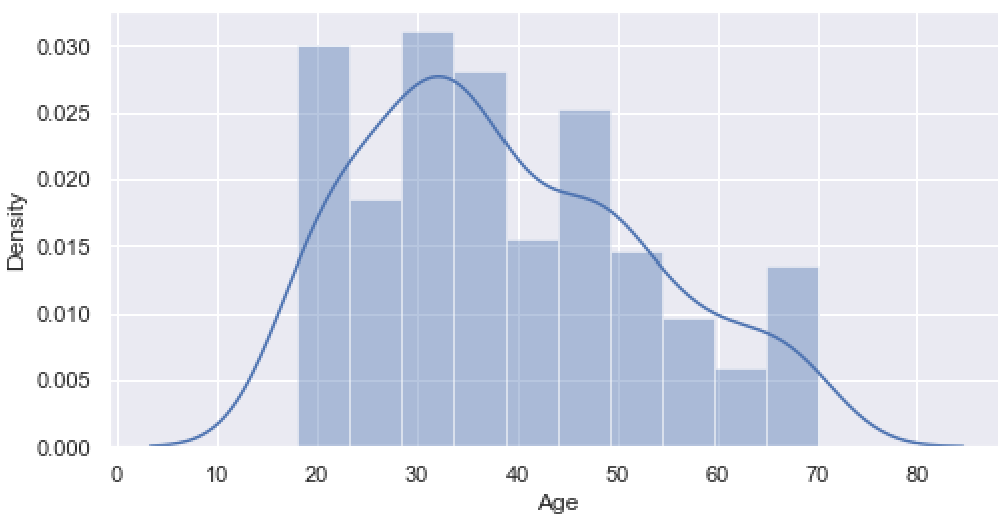
In [51]:
```
sn.displot(data=df_segmentation, x="Income", hue="Gender", kind="kde");
```



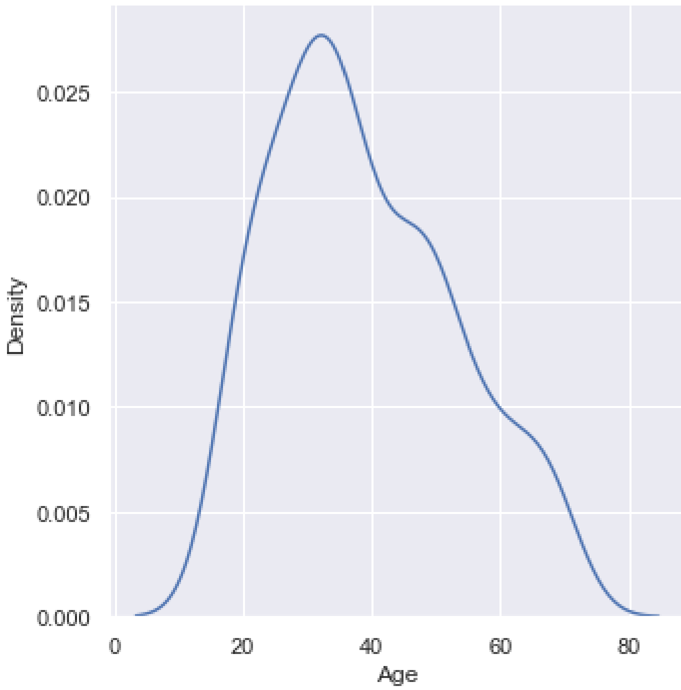
Age

In [52]:
```
#Distribution
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,4
sn.distplot(df_segmentation["Age"], bins=10)
```

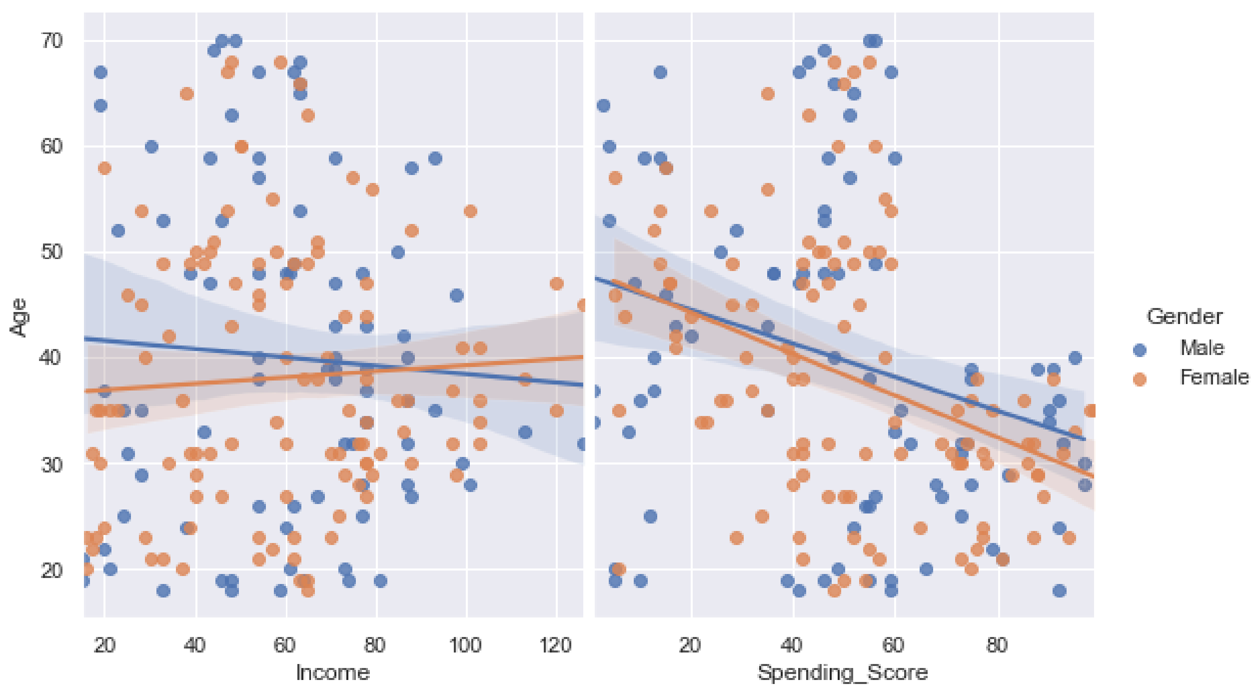
Out[52]: <AxesSubplot:xlabel='Age', ylabel='Density'>



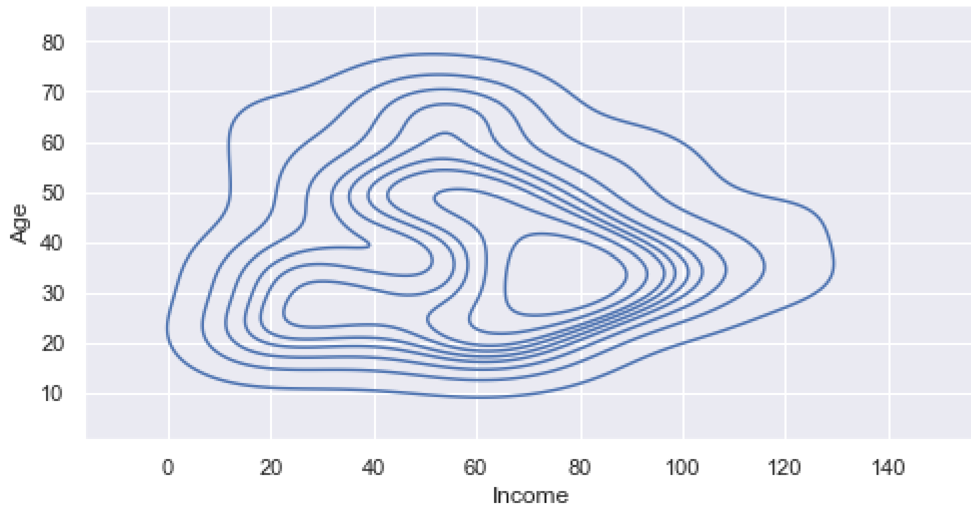
In [53]:
```
sn.displot(data=df_segmentation, x="Age", kind="kde");
```



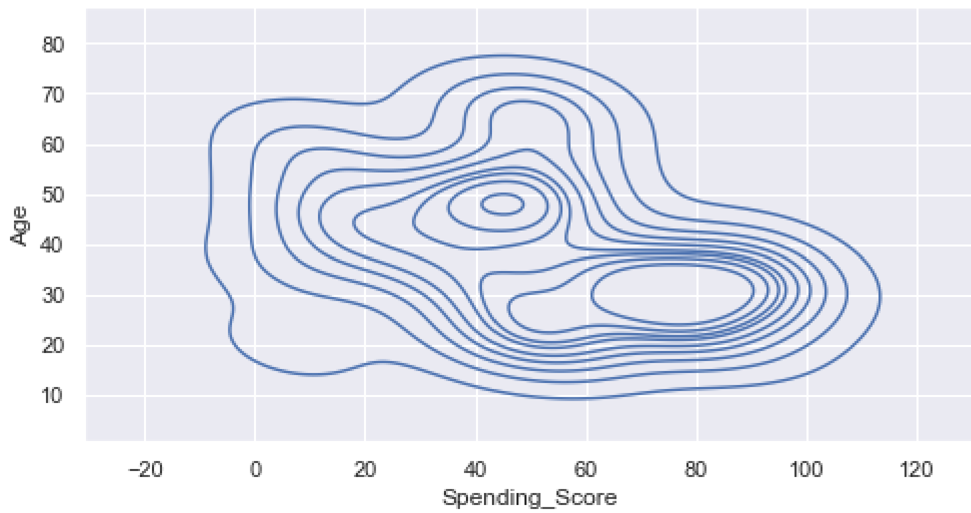
In [54]:
```
sn.pairplot(df_segmentation, x_vars=["Income", "Spending_Score"], y_vars=["Age"],
             hue="Gender", height=5, aspect=.8, kind="reg");
```



In [55]:
```
sn.kdeplot(df_segmentation.Income, df_segmentation.Age);
```

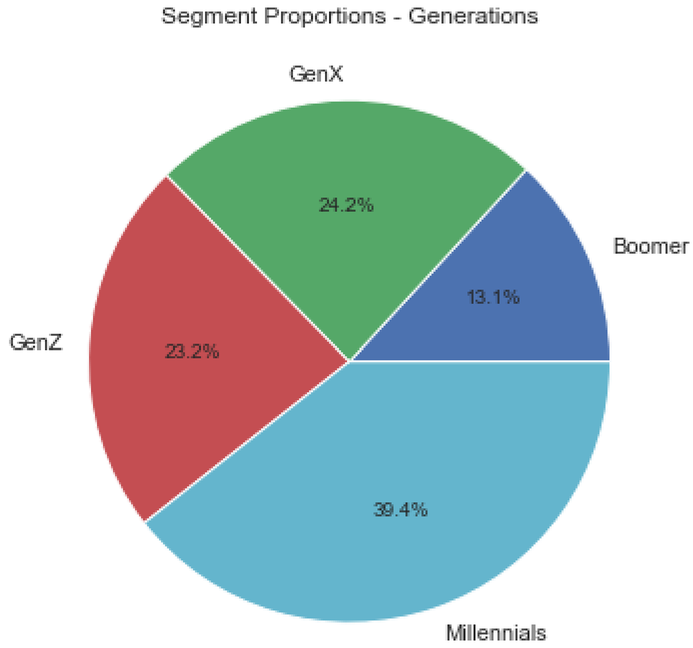


In [56]:
```
sn.kdeplot(df_segmentation.Spending_Score, df_segmentation.Age);
```

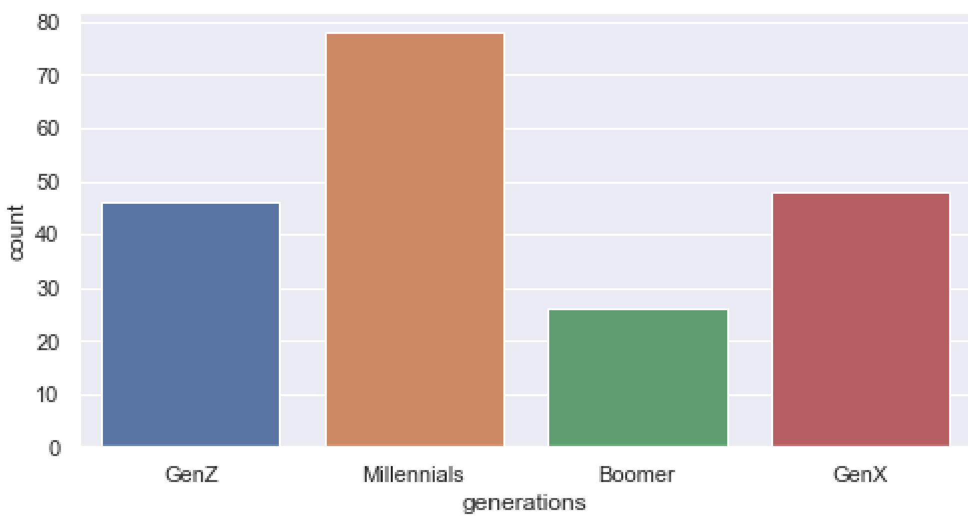


Generations

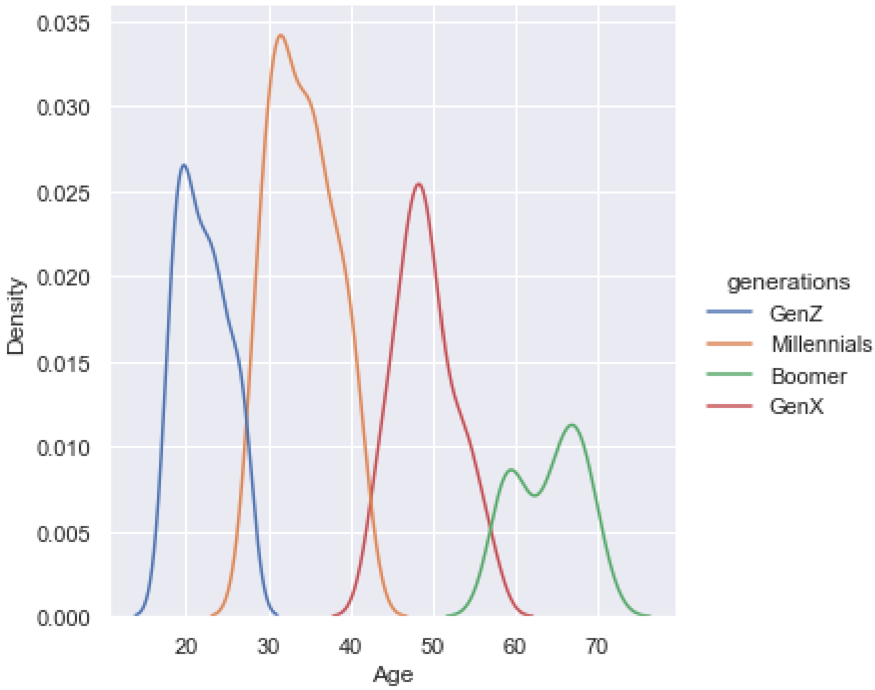
In [57]:
```
#Distribution
seg_prop_gen = df_segmentation[['CustomerID','generations']].groupby(['generations']).c
plt.figure(figsize = (9, 6))
plt.pie(seg_prop_gen['CustomerID'],
        labels = ['Boomer', 'GenX', 'GenZ','Millennials'],
        autopct = '%1.1f%%',
        colors = ('b', 'g', 'r', 'c'))
plt.title('Segment Proportions - Generations');
```



In [58]:
```
sn.countplot(x="generations",data=df_segmentation);
```

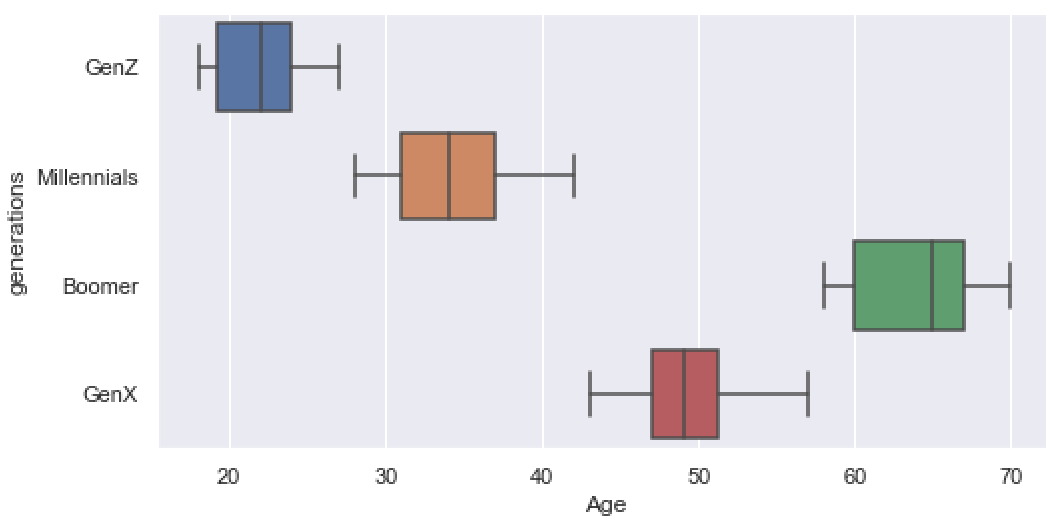


In [59]:
```
sn.displot(data=df_segmentation, x="Age", hue="generations", kind="kde");
```



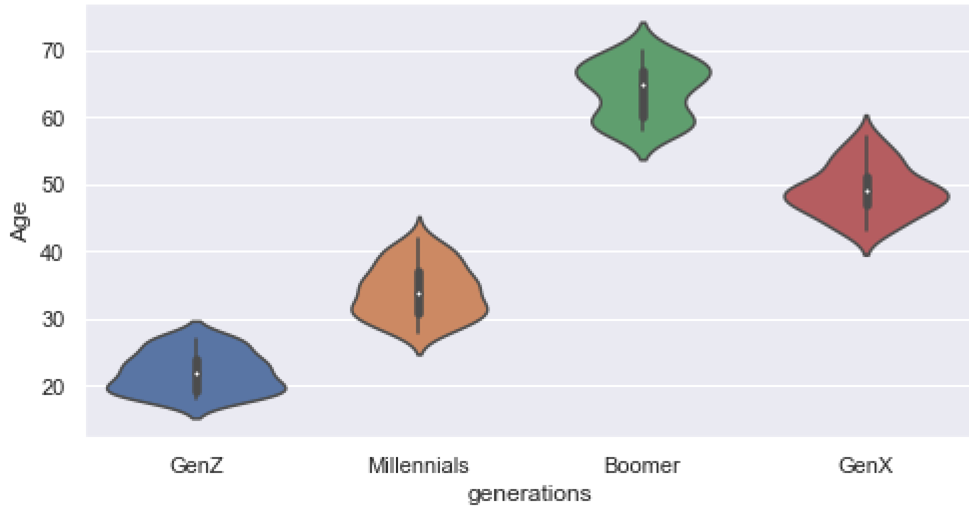
In [60]:

```
#Boxplot
sn.boxplot(x="Age", y="generations",data=df_segmentation);
```



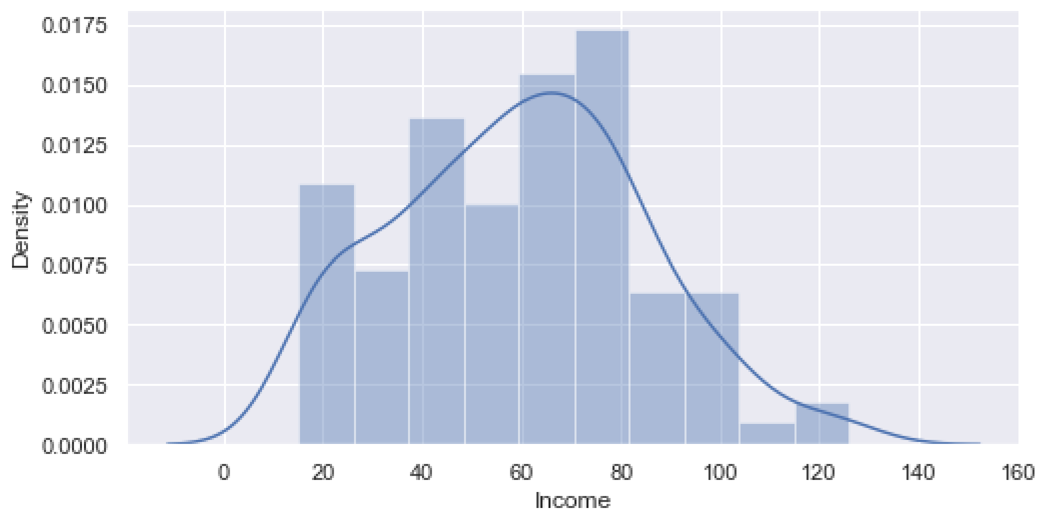
In [61]:

```
#Violinplot
sn.violinplot(data=df_segmentation, x='generations', y='Age');
```

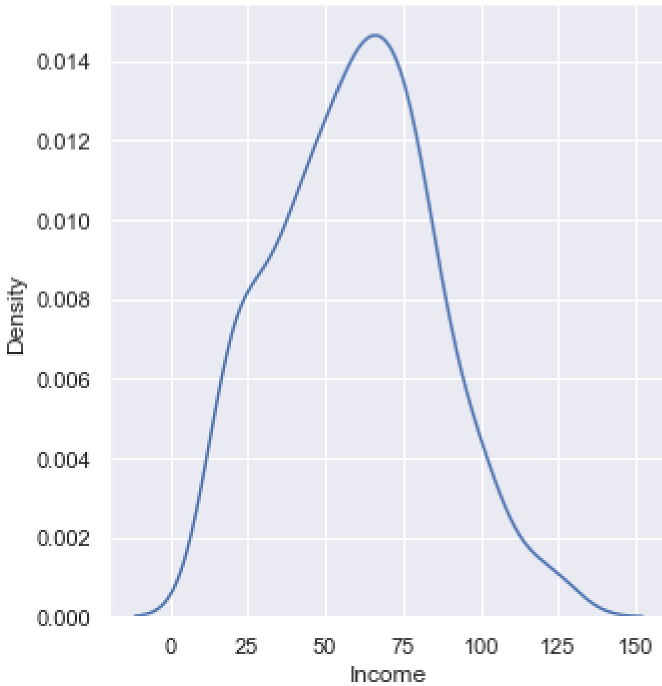


Income

In [62]:
```
#Distribution
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,4
sn.distplot(df_segmentation["Income"], bins=10);
```



In [63]:
```
sn.displot(data=df_segmentation, x="Income", kind="kde");
```

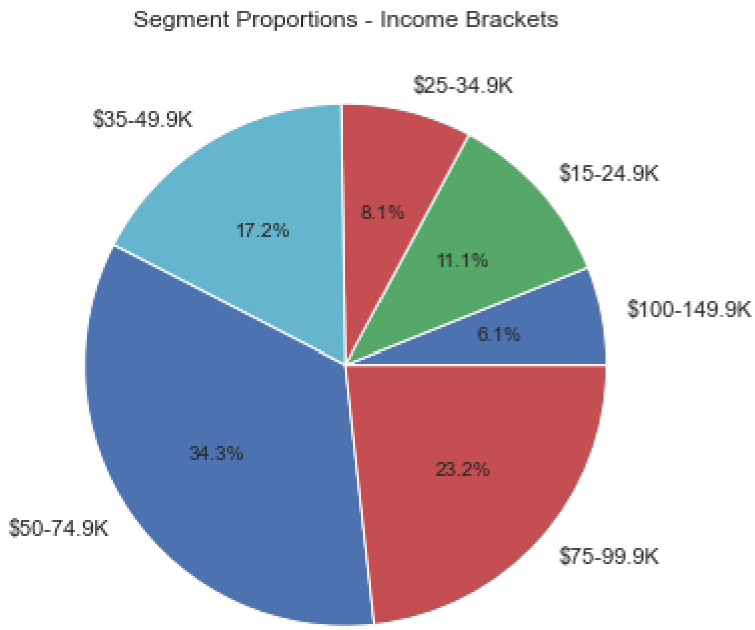


In [64]:

```
sn.lmplot(x='Age', y='Income', data=df_segmentation, fit_reg=False, hue='income_bracket
```

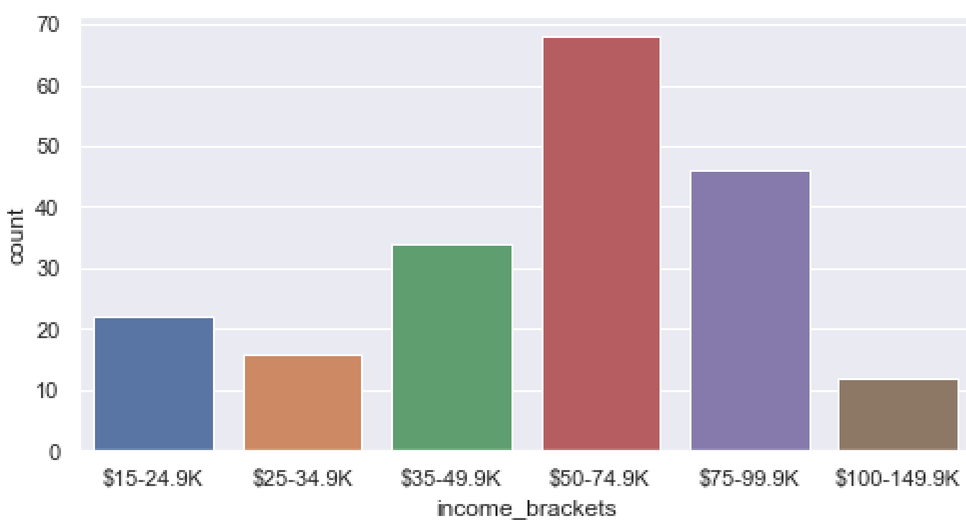
In [65]:

```
#Segment Proportions - Income Brackets
seg_prop_ii = df_segmentation[['CustomerID','income_brackets']].groupby(['income_bracke
#Piechart - Income Bracket proportion
plt.figure(figsize = (9, 6))
plt.pie(seg_prop_ii['CustomerID'],
        labels = ['$100-149.9K', '$15-24.9K', '$25-34.9K','$35-49.9K', '$50-74.9K', '$7
        autopct = '%1.1f%%',
        colors = ('b', 'g', 'r', 'c', 'b', 'r'))
plt.title('Segment Proportions - Income Brackets');
```



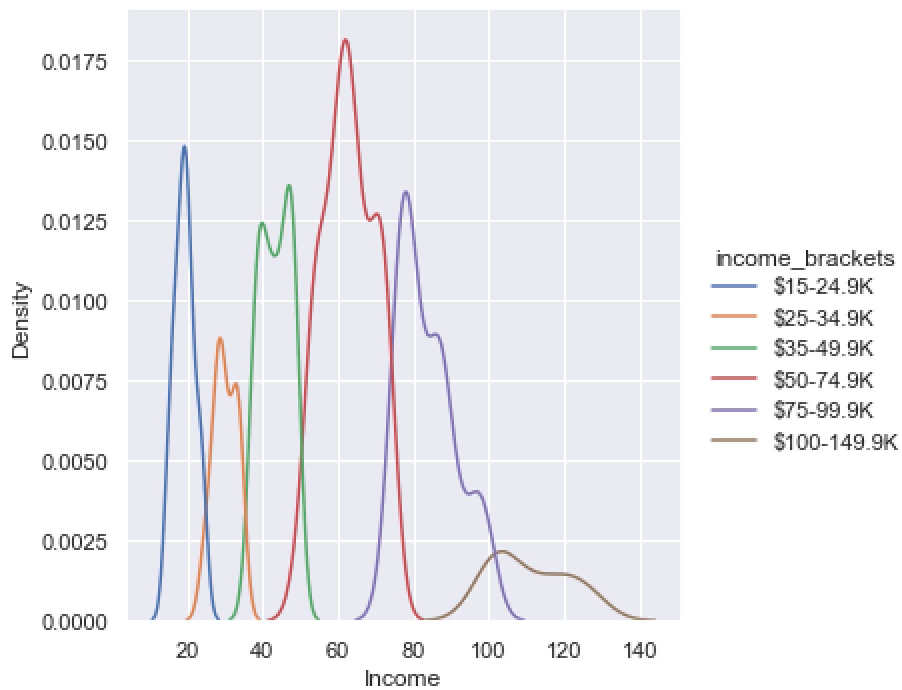
In [66]:

```
sn.countplot(x="income_brackets",data=df_segmentation);
```



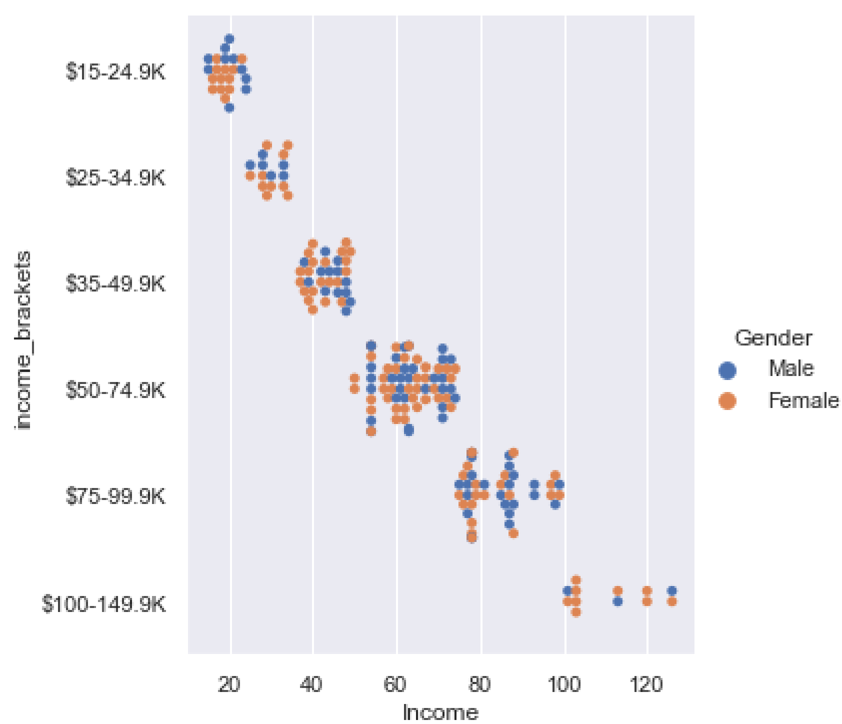
In [67]:

```
sn.displot(data=df_segmentation, x="Income", hue="income_brackets", kind="kde");
```



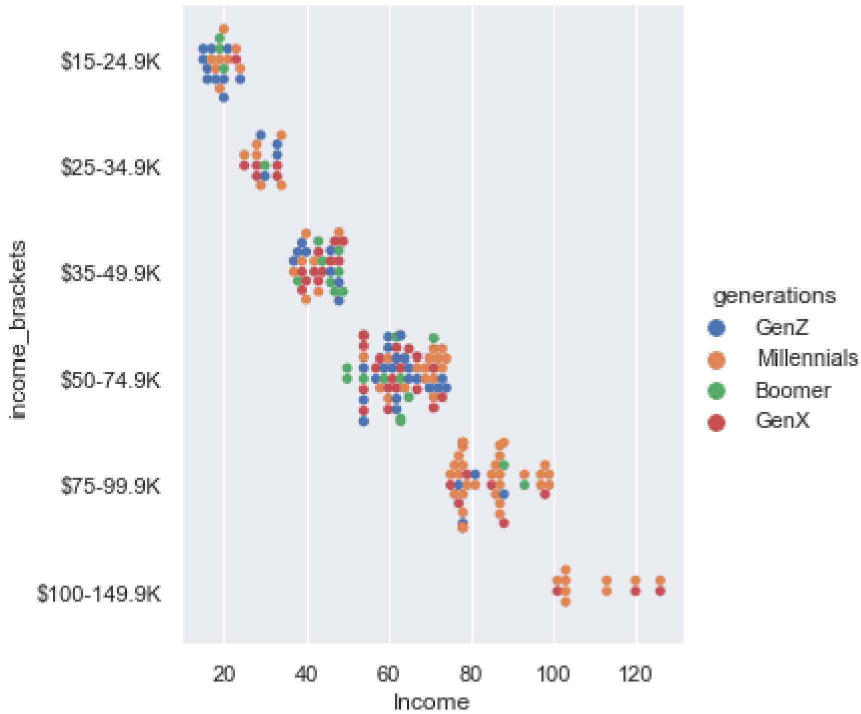
In [68]:

```
sn.catplot(data=df_segmentation, x="Income", y="income_brackets", hue="Gender", kind="s
```

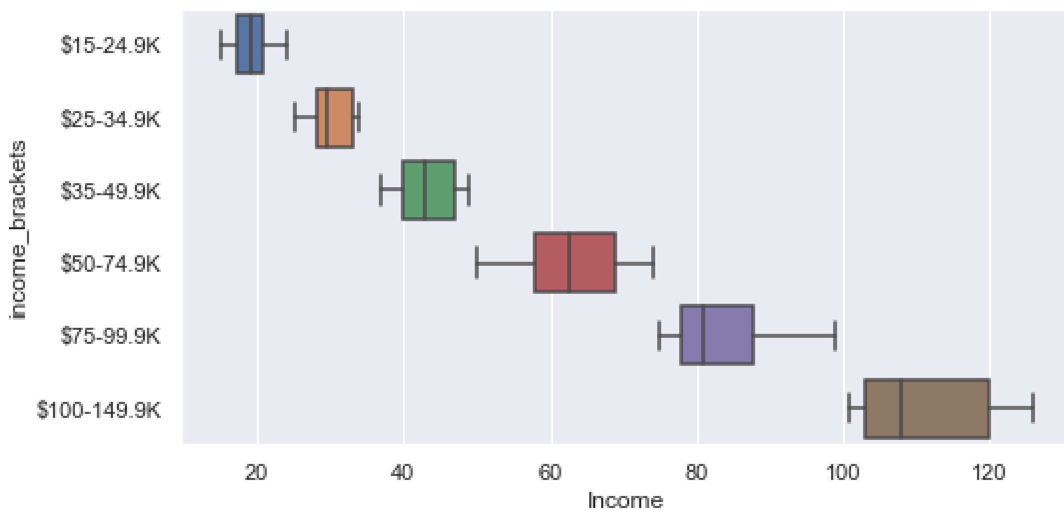


In [69]:

```
sn.catplot(data=df_segmentation, x="Income", y="income_brackets", hue="generations", ki
```

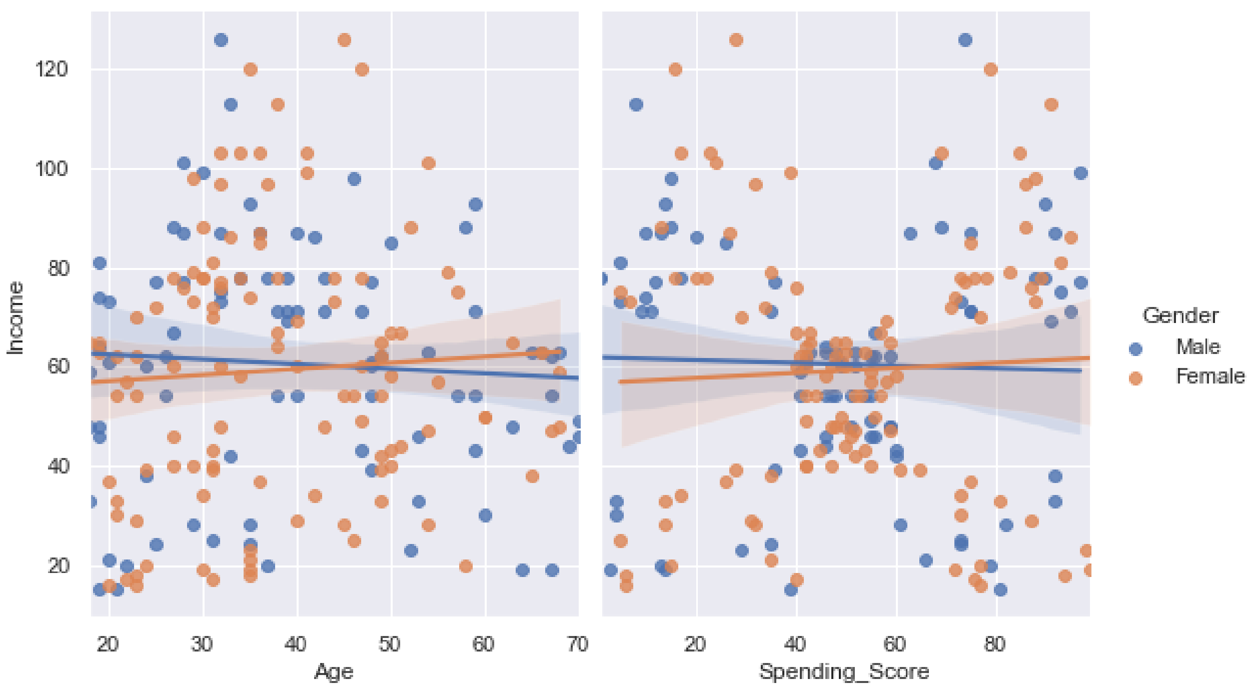


In [70]:
```
sn.boxplot(x="Income", y="income_brackets",data=df_segmentation);
```

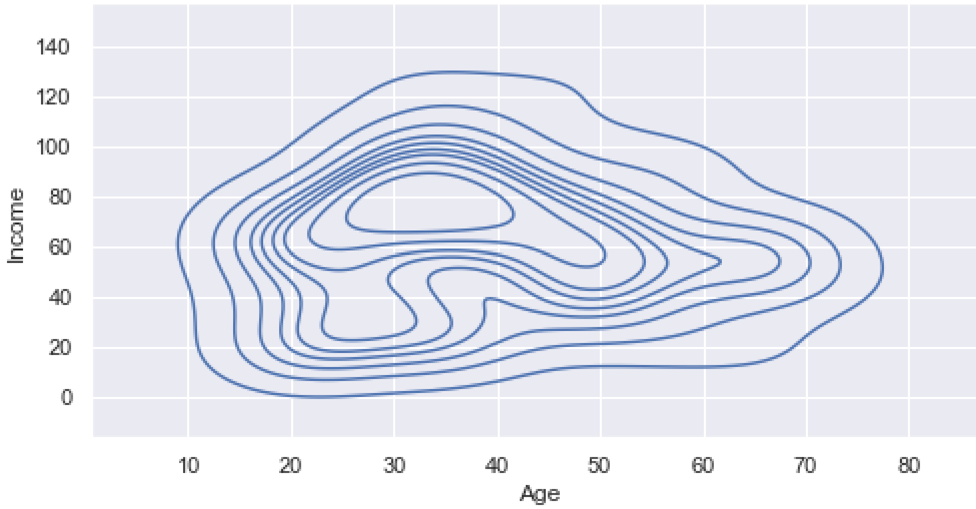


In [71]:
```
sn.violinplot(data=df_segmentation, x='income_brackets', y='Income');
```

In [72]:
```
sn.pairplot(df_segmentation, x_vars=["Age", "Spending_Score"], y_vars=["Income"],
             hue="Gender", height=5, aspect=.8, kind="reg");
```

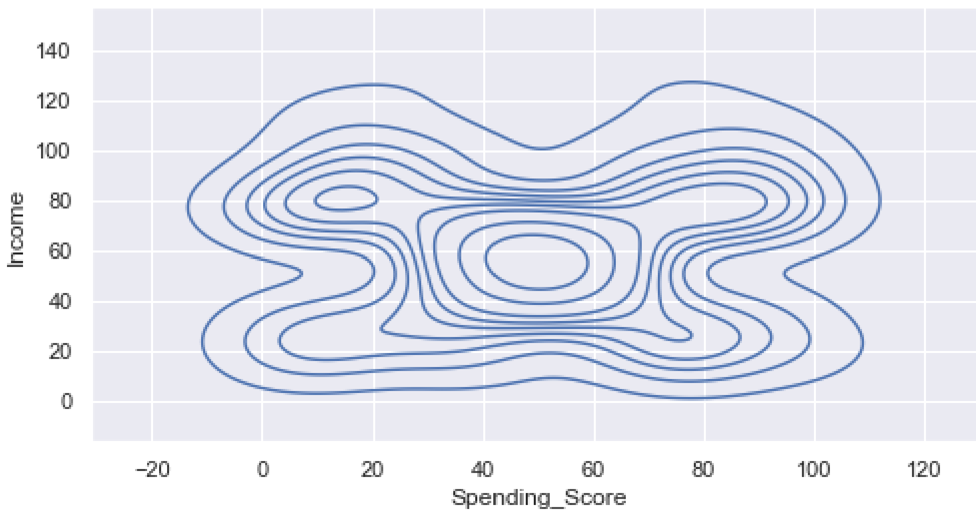


In [73]:
```
sn.kdeplot(df_segmentation.Age, df_segmentation.Income);
```



In [74]:

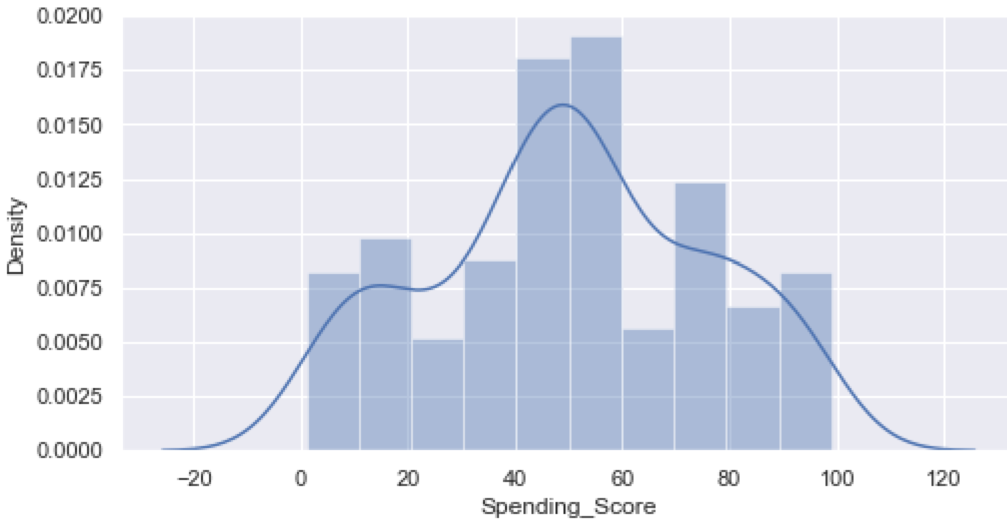
```
sn.kdeplot(df_segmentation.Spending_Score, df_segmentation.Income);
```



Spending Score

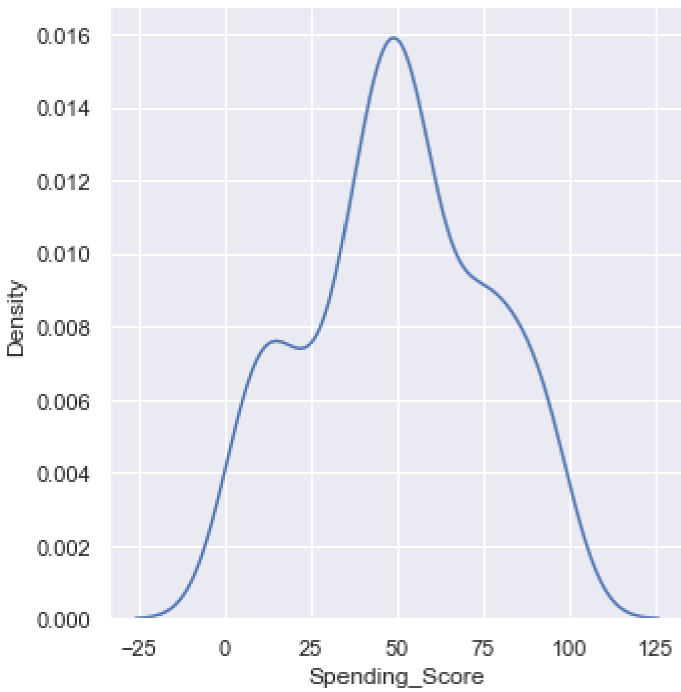
In [75]:

```
#Spending Score Distribution
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,4
sn.distplot(df_segmentation["Spending_Score"], bins=10);
```



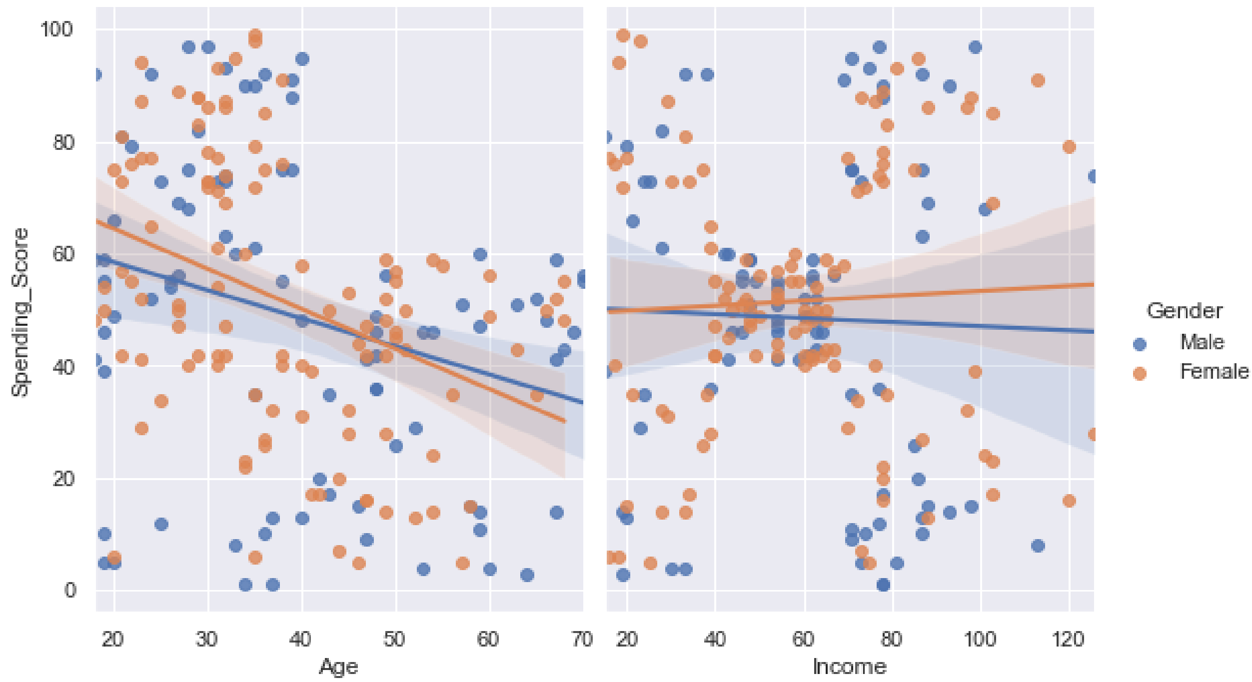
In [76]:

```
sn.displot(data=df_segmentation, x="Spending_Score", kind="kde");
```



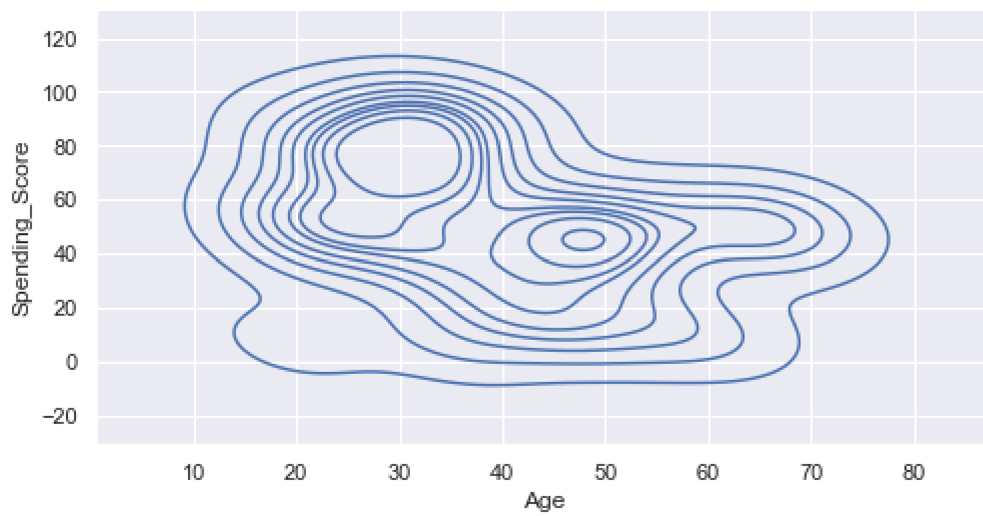
In [77]:

```
sn.pairplot(df_segmentation, x_vars=["Age", "Income"], y_vars=["Spending_Score"],
             hue="Gender", height=5, aspect=.8, kind="reg");
```



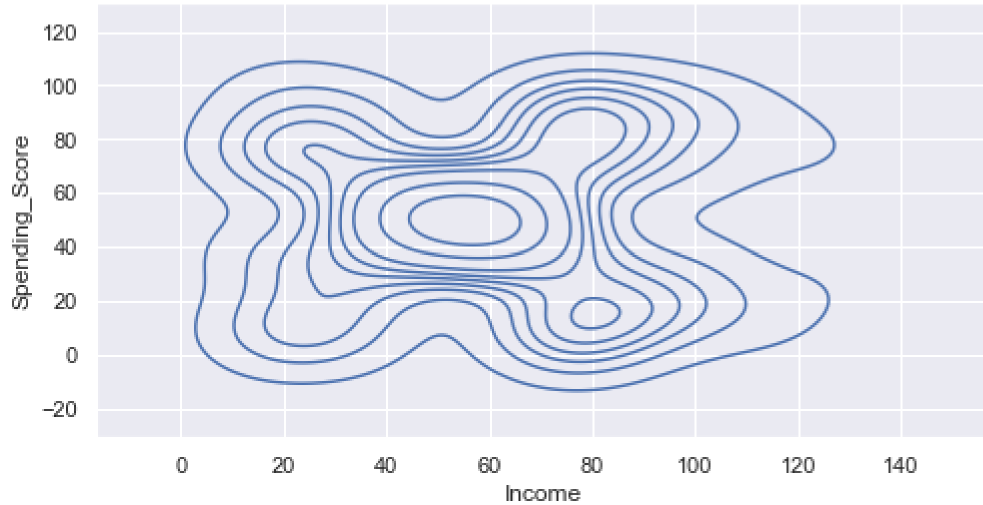
In [78]:

```
sn.kdeplot(df_segmentation.Age, df_segmentation.Spending_Score);
```



In [79]:

```
sn.kdeplot(df_segmentation.Income, df_segmentation.Spending_Score);
```



Filter Dataset

Drop columns

In [80]:
```
to_drop = ['CustomerID','generations','income_brackets','Gender','Gender_m','Spending_S
#Keeping: 'Age','Income','Gender_f',
df = df_segmentation.drop(to_drop, axis=1)
```

Column Labels

In [81]:
```
df.columns
```

Out[81]:
```
Index(['Age', 'Income', 'Gender_f'], dtype='object')
```

View first 5 rows

In [82]:
```
df.head(5)
```

Out[82]:

	Age	Income	Gender_f
0	19	15	0
1	21	15	0
2	20	16	1
3	23	16	1
4	31	17	1

View last 5 rows

In [83]:
```
df.tail(5)
```

Out[83]:

	Age	Income	Gender_f
193	38	113	1

	Age	Income	Gender_f
194	47	120	1
195	35	120	1
196	45	126	1
197	32	126	0

Identify, columns, number of rows, null values, and data types

In [84]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 198 entries, 0 to 197
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Age       198 non-null    int64
 1   Income    198 non-null    int64
 2   Gender_f  198 non-null    int64
dtypes: int64(3)
memory usage: 14.3 KB
```

Descriptive Statistics

In [85]:

```
df.describe().transpose()
```

Out[85]:

	count	mean	std	min	25%	50%	75%	max
Age	198.0	38.929293	14.016852	18.0	28.25	36.0	49.00	70.0
Income	198.0	59.787879	25.237259	15.0	40.50	61.0	77.75	126.0
Gender_f	198.0	0.565657	0.496927	0.0	0.00	1.0	1.00	1.0

Correlation Estimate

Pearsons Correlation Coefficient

In [86]:

```
df.corr()
```

Out[86]:

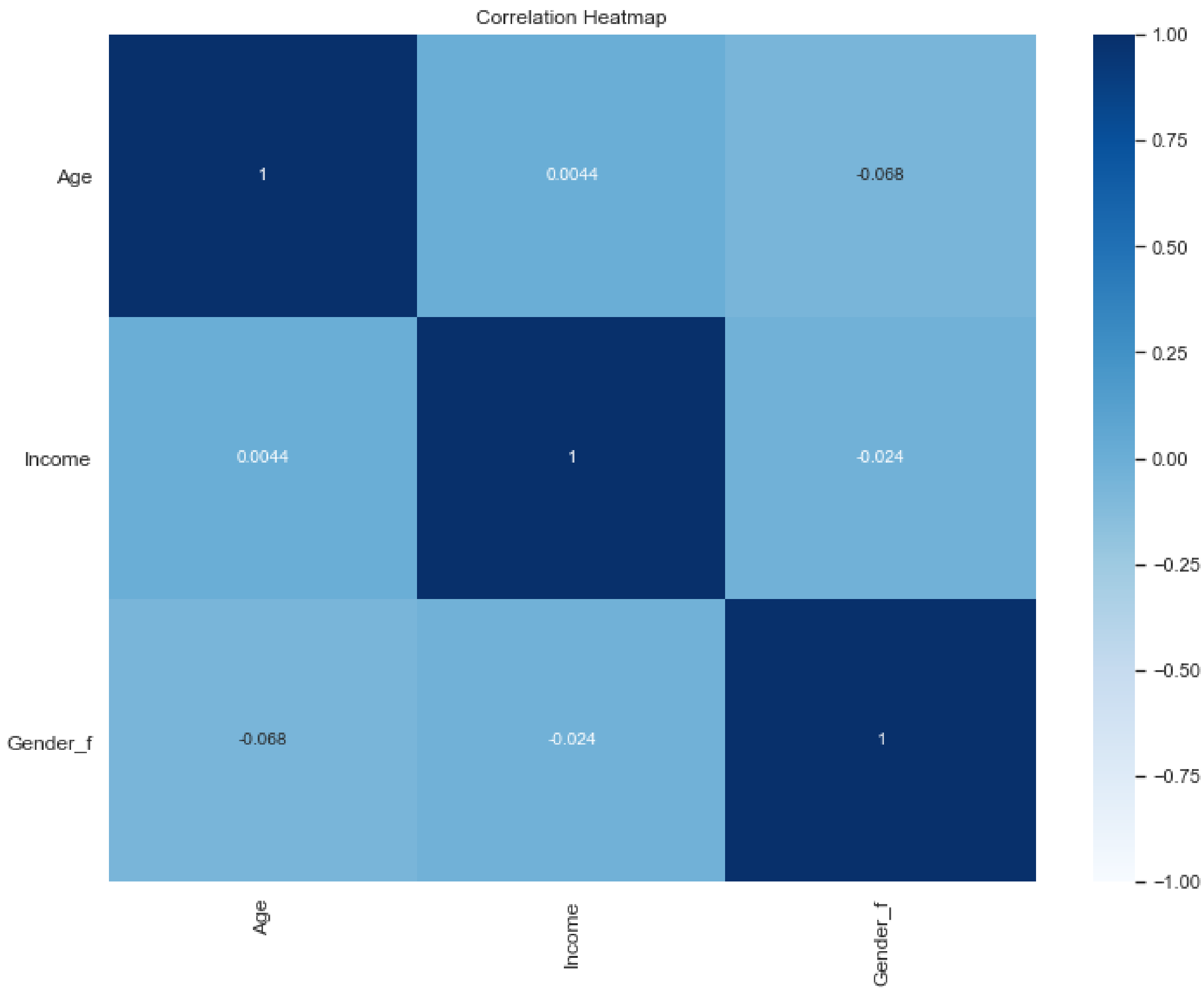
	Age	Income	Gender_f
Age	1.000000	0.004406	-0.067835
Income	0.004406	1.000000	-0.024384
Gender_f	-0.067835	-0.024384	1.000000

Correlation using Heat Map

In [87]:

```
plt.figure(figsize = (12, 9))
s = sn.heatmap(df.corr(), annot = True, cmap = 'Blues', vmin = -1, vmax = 1)
s.set_yticklabels(s.get_yticklabels(), rotation = 0, fontsize = 12)
```

```
s.set_xticklabels(s.get_xticklabels(), rotation = 90, fontsize = 12)
plt.title('Correlation Heatmap')
plt.show()
```



Standarization

Standarizing data, so that all features have equal weight

In [88]:
```
scaler = StandardScaler()
segmentation_std = scaler.fit_transform(df)
```

Hierarchical Clustering

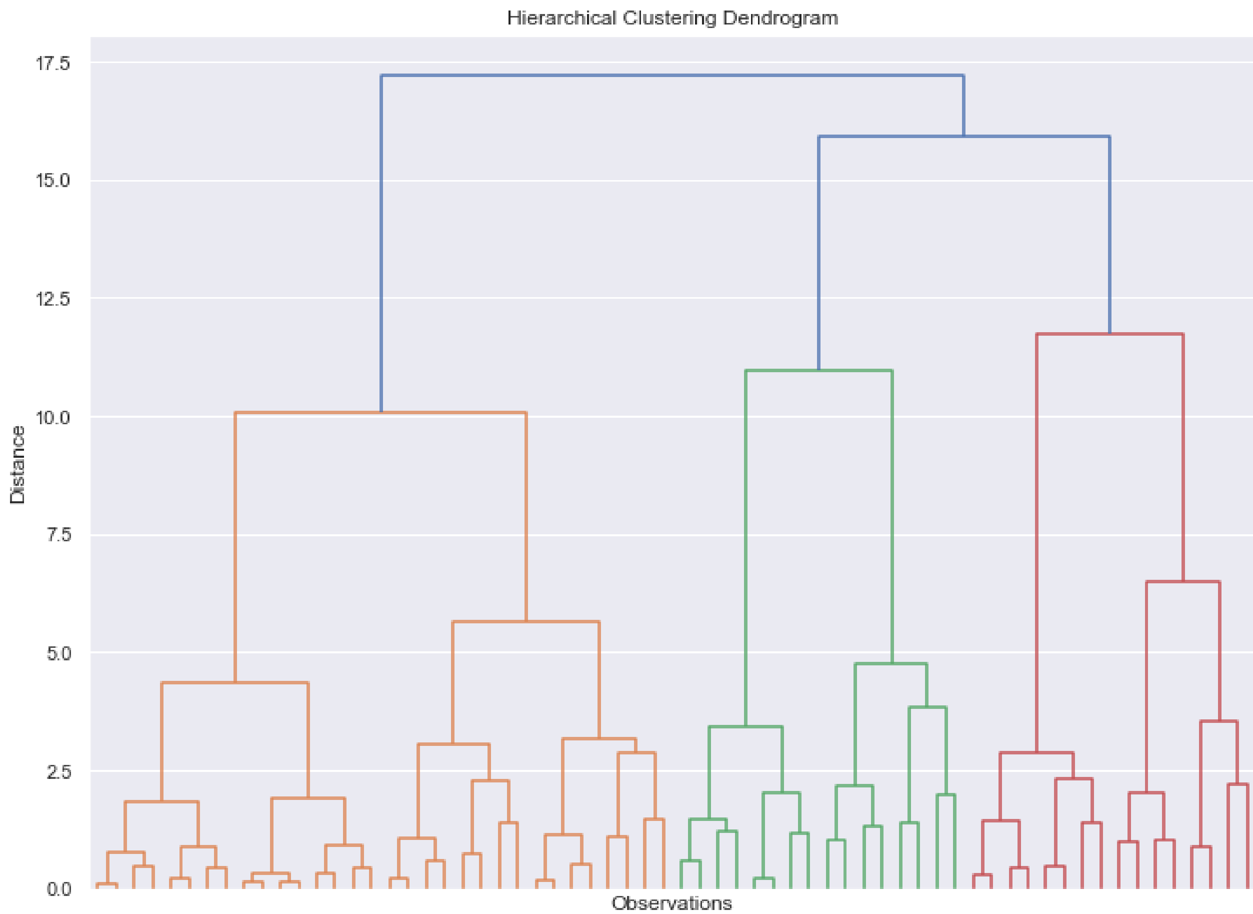
In [89]:
```
hier_clust = linkage(segmentation_std, method = 'ward')
```

Ward method calculates the average of the squares of the distances between clusters.

Plot results from Hierarchical Clustering using Dendrogram graph

In [90]:
```
plt.figure(figsize = (12,9))
plt.title('Hierarchical Clustering Dendrogram')
```

```
plt.xlabel('Observations')
plt.ylabel('Distance')
dendrogram(hier_clust, truncate_mode = 'level', p = 5, show_leaf_counts = False, no_lab
plt.show()
```



Dendrogram is a tree-like, hierarchical representation of points or observations.

The goal is to group observations together based on the distance (y-axis) between points. Less distance represents observations are higher in similarity.

To divide into subgroups we find the longest vertical line unintercepted by a horizontal line from the dendrogram. Subgroups are color coded in orange, green, and red.

K-Means

Perform K-means Clustering

K-means is a traditional clustering technique commonly used for segmentation data.

K-means Clustering steps: 1 Choose number of clusters. K in K-means represents number of clusters identified.

2 Specify cluster seed or the starting centroid. Assign each point to a cluster based on the obvervation's proximity or Euclidean squared distance from the seeds.

3 Calculate centroid or geometrical center between each cluster's observations.

Note: Outliers have been removed due to the squared Euclidean distance being very sensitive to outliers.

Within Cluster Sum of Squares (WCSS) is the sum of the variance between the observations in each cluster. Measures the distance between each observation and the centroid and calculates the square difference between the the two.

Use WCSS to determine the appropriate clustering solution.

In [91]:

```
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(segmentation_std)
    wcss.append(kmeans.inertia_)
```

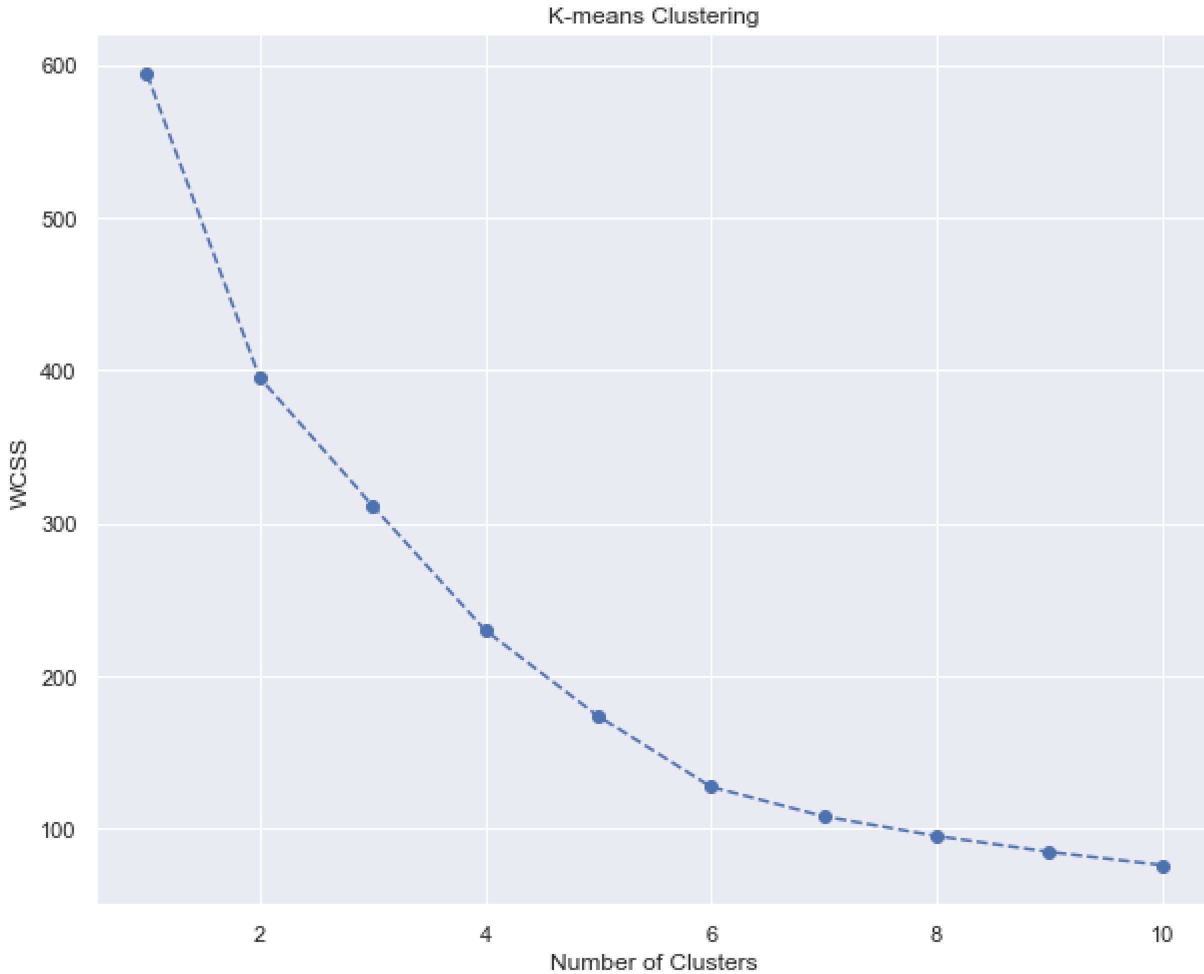
3 clusters identified based on the Dendrogram chart thus we run the code with 10 iterations using kmeans++ which is an initialization algorithm that finds the best starting points for the centroids.

Increasing iterations would not improve our results, however, for cases with higher number of clusters increasing iterations may provide more precise results.

Plot WCSS to identify number of clusters

In [92]:

```
plt.figure(figsize = (10,8))
plt.plot(range(1, 11), wcss, marker = 'o', linestyle = '--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('K-means Clustering')
plt.show()
```



The function is monotonically decreasing sometimes rapidly declining while other times more smoothly.

Depending on the shape of the graph we make a decision about the number of clusters using the Elbow Method. Usually the part before the Elbow would be steeply declining while after more smoothly.

Run K-Means with a fixed number of clusters

In [93]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
```

Specifiy number of clusters equal to 3 and divide sample into 3 subgroups.

Fit data using kmeans fit using our standarized data.

In [94]:
```
kmeans.fit(segmentation_std)
```

Out[94]:
```
KMeans(n_clusters=3, random_state=42)
```

Results

Create new dataframe with original features adding new clusters column

In [95]:
```
df_segment_kmeans = df.copy()
df_segment_kmeans['Segment K-means'] = kmeans.labels_
```

Includes predictive clusters for each observation in dataset to gain a better understanding of who our customers are.

Calculate average values per cluster

In [96]:
```
df_segment_analysis = df_segment_kmeans.groupby(['Segment K-means']).mean()
df_segment_analysis
```

Out[96]:

Segment K-means	Age	Income	Gender_f
0	32.048387	64.387097	0.000000
1	58.866667	48.933333	0.466667
2	33.758242	62.021978	1.000000

Mean values for clusters in K-means algorithm:

Segment 0 includes no Female Gender, with an average age of 32.05 years, and Income of 64.39 (highest earning subgroup).

Segment 1 is composed of Female and Male Gender almost equally, with an average age of 58.87 (oldest subgroup), and Income of 48.93.

Segment 2 includes all Female Gender, with an average age of 33.76, and Income of 62.02.

Calculate size and proportions of clusters

In [97]:
```
df_segment_analysis['N Obs'] = df_segment_kmeans[['Segment K-means','Age']].groupby(['S
df_segment_analysis['Prop Obs'] = df_segment_analysis['N Obs'] / df_segment_analysis['N
df_segment_analysis
```

Out[97]:

Segment K-means	Age	Income	Gender_f	N Obs	Prop Obs
0	32.048387	64.387097	0.000000	62	0.313131
1	58.866667	48.933333	0.466667	45	0.227273
2	33.758242	62.021978	1.000000	91	0.459596

Label Segments

In [98]:
```
df_segment_analysis.rename({0:'Segment 1',
                            1:'Segment 2',
                            2:'Segment 3'})
```

Out[98]:

	Age	Income	Gender_f	N Obs	Prop Obs
Segment K-means					
Segment 1	32.048387	64.387097	0.000000	62	0.313131
Segment 2	58.866667	48.933333	0.466667	45	0.227273
Segment 3	33.758242	62.021978	1.000000	91	0.459596

Based on Prop Obs:

Segment 1 is the second to largest subgroup with 31.31%

Segment 2 is the smallest subgroup with 22.73%

Segment 3 is the largest subgroup with 45.96%

Add segment labels

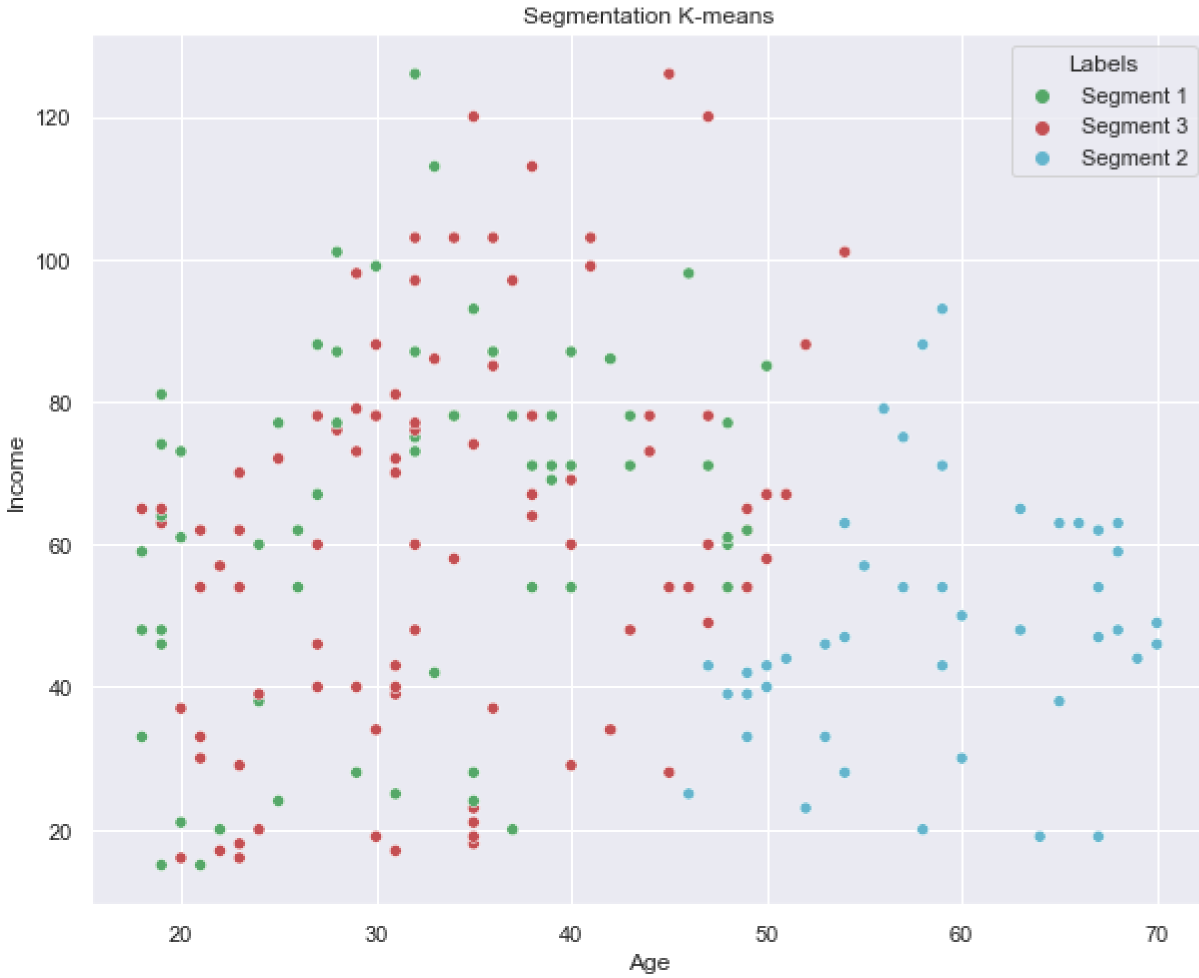
In [99]:

```
df_segment_kmeans['Labels'] = df_segment_kmeans['Segment K-means'].map({0:'Segment 1',
                                                                        1:'Segment 2',
                                                                        2:'Segment 3'})
```

Plot results from the K-means algorithm

In [100…

```
x_axis = df_segment_kmeans['Age']
y_axis = df_segment_kmeans['Income']
plt.figure(figsize = (10, 8))
sn.scatterplot(x_axis, y_axis, hue = df_segment_kmeans['Labels'], palette = ['g', 'r',
plt.title('Segmentation K-means')
plt.show()
```



Segment 2 is clearly seperated as it is highest in Age and lowest in Income. Segment 1 and 3 are group together making it difficult to gain more insight.

To gain more clarity, next we combine K-means to Principal Component Analysis.

Principal Component Analysis (PCA)

Apply PCA to find a subset of components to explain the variance. By combining K-means and PCA to obtain a better clustering solution.

In [101…

```
pca = PCA()
```

Fit PCA to standardized data

In [102…

```
pca.fit(segmentation_std)
```

Out[102…

```
PCA()
```

PCA creates as many components as there are features in our data. In our case 3 components. These components are arranged in order of importance or how much the variance in our data is explained by each component.

In [103…

```
pca.explained_variance_ratio_
```

Out[103…

```
array([0.35785905, 0.33240042, 0.30974053])
```

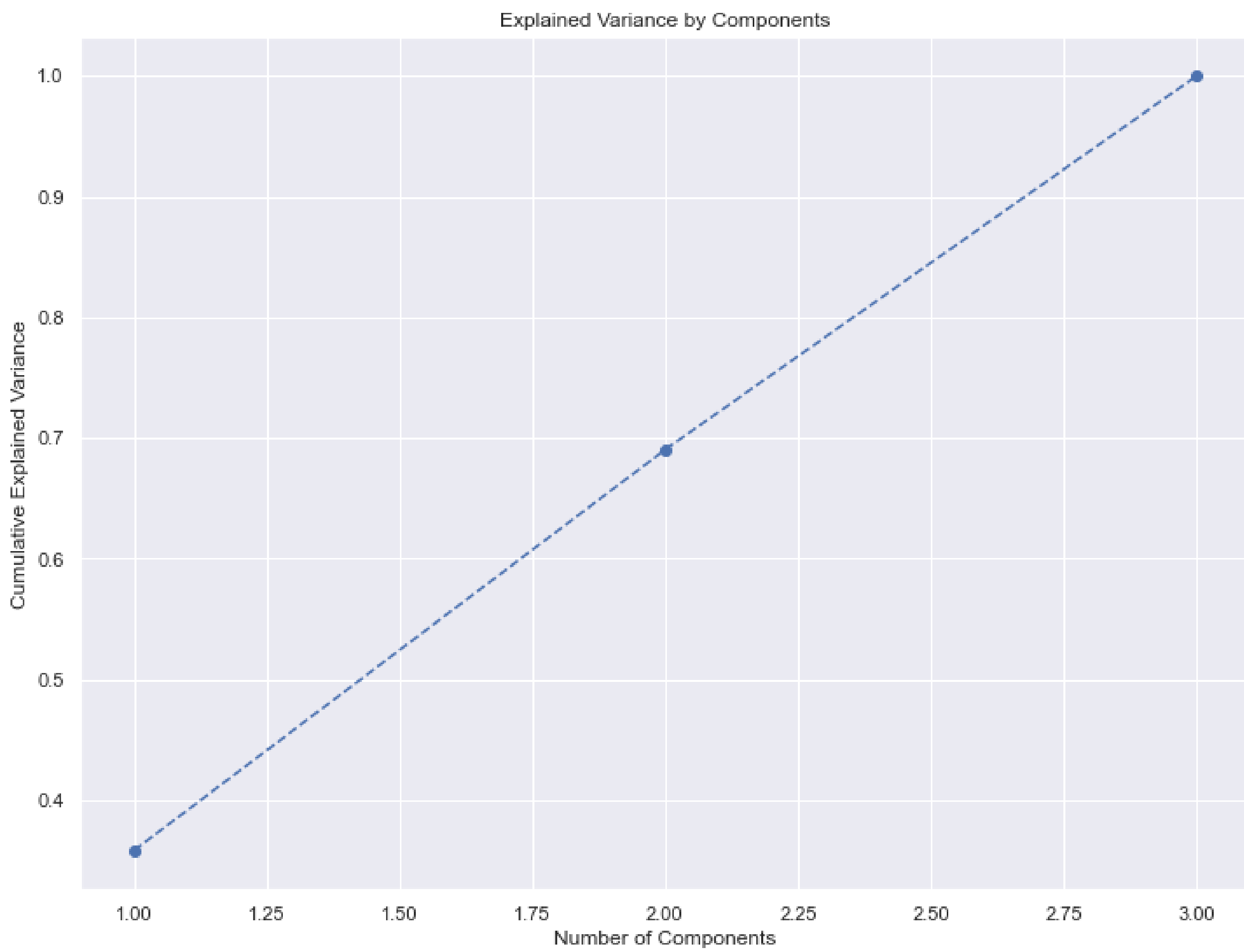
PCA components equal to 3. The array shows degree of explained variance per component. Each of our components explains for about a third and together these 3 components explain for 100% of the variability in the data.

Goal is to find a subset of components while preserving variance.

Plot Cumulative Explained Variance

In [104…

```
plt.figure(figsize = (12,9))
plt.plot(range(1,4), pca.explained_variance_ratio_.cumsum(), marker = 'o', linestyle =
plt.title('Explained Variance by Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance');
```



Rule of thumb is to keep atleast 70% and 80% of the explained variance.

In [105…

```
pca = PCA(n_components = 2)
```

We choose 2 components.

Fit model to data with selected number of components

In [106…

```
pca.fit(segmentation_std)
```

Out[106… PCA(n_components=2)

Principal Components Analysis (PCA) Results

Insights from PCA results

In [107…
```
pca.components_
```

Out[107…
```
array([[ 0.66112684,  0.27139512, -0.69946837],
       [-0.33616626,  0.94061803,  0.04722245]])
```

Array shows loadings per component on each of the original features

Loadings are the correlations between the components and original features

Creates Pandas DataFrame from array with column labels

In [108…
```
df_pca_components = pd.DataFrame(data = pca.components_, columns = df.columns.values, i
df_pca_components
```

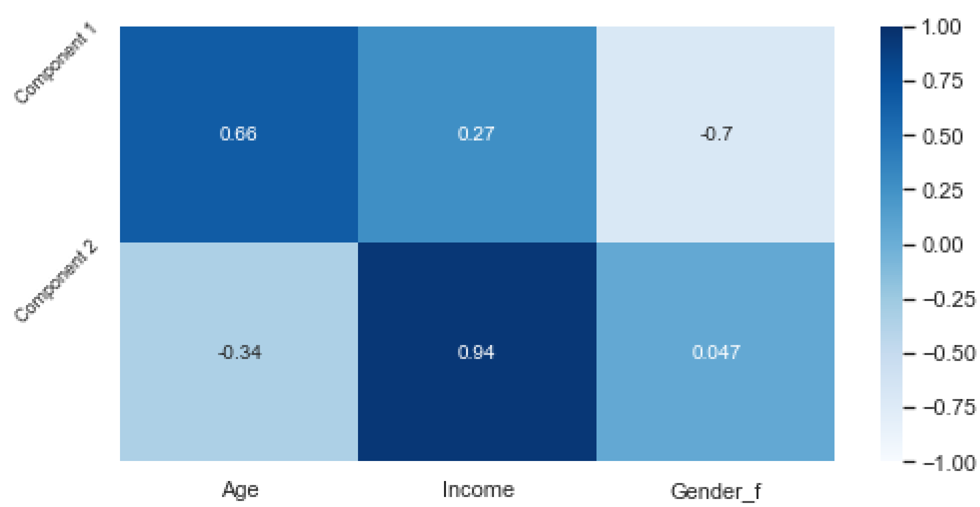
Out[108…

	Age	Income	Gender_f
Component 1	0.661127	0.271395	-0.699468
Component 2	-0.336166	0.940618	0.047222

Closer to 0 lower the loading or correlation between components and original features.

Heat Map for Principal Components against original features

In [109…
```
sn.heatmap(df_pca_components, vmin = -1, vmax = 1, cmap = 'Blues', annot = True)
plt.yticks([0, 1],
           ['Component 1', 'Component 2'],
           rotation = 45,
           fontsize = 9);
```



Component 1 positive correlation to age and income while negative to Female Gender. Most important being Female Gender and Age.

Component 2 positive correlation to Income and female gender while negative to Age. Most important Income and Female Gender.

Transform standarized data

In [110…

```
pca.transform(segmentation_std)
```

Out[110…

```
array([[-0.62700795, -1.24823477],
       [-0.53243569, -1.29632237],
       [-1.98009682, -1.13964325],
       [-1.83823843, -1.21177465],
       [-1.44916837, -1.36675958],
       [-1.87474355, -1.15036537],
       [-1.24924283, -1.4255693 ],
       [-1.81667641, -1.13704368],
       [ 1.54399202, -2.18074389],
       [-1.47489248, -1.26798481],
       [ 1.68585041, -2.25287529],
       [-1.23846182, -1.38820381],
       [-0.14009978, -1.90384576],
       [-1.74782827, -1.08635652],
       [ 0.27804746, -1.49419577],
       [-0.43124452, -1.13353876],
       [-1.21689981, -1.31347285],
       [-0.51503577, -1.04808567],
       [ 1.01968246, -1.74275634],
       [-1.19533779, -1.23874188],
       [ 0.22659923, -1.29664624],
       [-0.24626209, -1.05620823],
       [-0.65362833, -1.42849273],
       [ 0.0482357 , -1.16310555],
       [-0.24299625, -1.50874669],
       [-0.01399354, -1.0029215 ],
       [-0.66857144, -1.29235248],
       [ 0.26972325, -1.14718431],
       [-0.89422109, -1.13476799],
       [-1.69808534, -0.72602337],
       [ 1.47343857, -1.67354837],
       [-1.78187659, -0.64057029],
       [ 1.17477866, -1.39314532],
       [-0.48023595, -0.55161228],
       [-0.42552188, -1.20170027],
       [-1.74953357, -0.52847384],
       [-0.74574379, -0.99602818],
       [-1.31317738, -0.70750257],
       [-0.99711756, -0.73966893],
       [-1.75369567, -0.35496811],
       [ 0.38496127, -1.39957368],
       [-0.14261413, -0.50904767],
       [ 1.00303405, -1.04873341],
       [-1.21198621, -0.54471896],
       [-0.36083583, -0.97750737],
       [-1.54298913, -0.37641235],
       [-0.3027687 , -0.96418569],
       [-1.39034973, -0.41117827],
       [-1.29577747, -0.45926587],
       [-1.2012052 , -0.50735347],
       [-0.32849281, -0.86541093],
       [ 0.32608509, -0.57597995],
```

```
[-1.16886218, -0.39525702],
[ 1.56630553, -1.16375329],
[-0.27042567, -0.85208924],
[ 0.99887194, -0.87522768],
[-0.21235854, -0.83876756],
[ 2.04994785, -1.36682582],
[-1.32566369, -0.18698537],
[ 1.31493176, -0.90739404],
[ 2.118796  , -1.31613866],
[-0.29279673, -0.0899048 ],
[ 0.5765626 , -1.11137193],
[-0.03815712, -0.79880252],
[ 1.80935509, -1.07310109],
[-0.31852085,  0.00886996],
[-0.54752356, -0.49695522],
[ 0.63462974, -1.09805025],
[-0.27123472, -0.01517384],
[-1.06767101, -0.23247341],
[ 2.15113902, -1.20404221],
[-0.34759803, -0.55576494],
[ 0.27790269, -0.83096888],
[ 0.27790269, -0.83096888],
[ 1.6848966 , -0.75273298],
[ 0.12445425,  0.04071245],
[-0.38826526, -0.32084993],
[ 0.7864601 , -0.29590076],
[-1.42856016,  0.2081137 ],
[-0.19912073, -0.41702513],
[ 1.59032434, -0.70464538],
[ 0.69188783, -0.24781316],
[ 2.06318566, -0.94508339],
[-0.34097912, -0.34489373],
[-1.52313242,  0.2562013 ],
[ 1.16474915, -0.48825117],
[ 0.11693908, -0.44919149],
[-1.44350327,  0.34425394],
[-0.86528868,  0.09309381],
[-0.10871057, -0.291607  ],
[ 0.75322081, -0.68702994],
[-0.19992977,  0.41989027],
[ 1.2294352 , -0.26405828],
[-0.56000987,  0.02356197],
[-0.93829893,  0.21591238],
[ 0.09456803,  0.31299295],
[-0.22900695, -0.14474463],
[-1.17472959,  0.33613139],
[ 1.2402162 , -0.22669279],
[-0.08379549,  0.44653364],
[-1.3423121 ,  0.50703756],
[-0.11287267, -0.11810127],
[ 2.14943372, -0.64615953],
[ 0.21070231,  0.33963631],
[ 1.29828334, -0.21337111],
[-1.43688437,  0.55512516],
[ 0.70177258, -0.48948041],
[ 1.54549501, -0.29622463],
[ 2.20750086, -0.63283785],
[ 2.11292859, -0.58475025],
[ 2.06564246, -0.56070645],
[-1.52067562,  0.64057824],
[-0.61145811,  0.22111151],
[-0.0987386 ,  0.58267389],
[-1.54639974,  0.73935301],
[-1.49911361,  0.71530921],
[ 0.5814762 , -0.34261804],
```

```
[-0.08052965, -0.00600482],
[ 0.03560463,  0.02063854],
[-0.0116815 ,  0.04468234],
[ 0.31189347,  0.50241993],
[-0.57911509,  0.33320795],
[-0.46298081,  0.35985132],
[ 0.90088907,  0.28862528],
[-1.25606404,  0.80596142],
[-0.87777499,  0.61361101],
[ 1.11159561,  0.26718104],
[ 0.96973722,  0.33931244],
[ 1.86817372, -0.11751978],
[ 0.87516495,  0.38740005],
[ 1.30074014,  0.17100584],
[ 0.92245109,  0.36335624],
[-1.13992977,  0.83260478],
[-0.85621298,  0.68834198],
[ 0.04557659,  0.89491943],
[-0.94000423,  0.77379506],
[-0.23071225,  0.41313804],
[ 0.61301018,  0.60639382],
[ 0.00907147,  0.95632871],
[-0.64550643,  0.66689774],
[ 0.40556948,  0.1752996 ],
[ 0.63457219,  0.68112478],
[-0.95494734,  0.90993531],
[-0.76580281,  0.81376011],
[ 0.32513128,  0.92416236],
[ 0.46698968,  0.85203095],
[ 1.41271232,  0.37115493],
[-0.75502181,  0.85112559],
[-0.64966854,  0.84040347],
[ 0.76148748,  0.74513363],
[ 1.18706266,  0.52873942],
[ 0.99791814,  0.62491462],
[-0.17680722,  0.59996546],
[-0.46052401,  0.74422827],
[-0.03494882,  0.52783406],
[-0.98067146,  1.00871008],
[ 0.90334587,  0.67300223],
[-0.83881306,  0.93657867],
[ 0.76148748,  0.74513363],
[-0.83881306,  0.93657867],
[ 0.40140737,  0.34880533],
[-0.87531819,  0.99798796],
[ 0.08453852,  1.21788709],
[-0.75918391,  1.02463132],
[ 1.59353264,  0.62199119],
[-0.47962922,  1.05387425],
[ 1.22602459,  0.85170708],
[-0.61070661,  1.16337113],
[-0.45806721,  1.12860521],
[ 0.76394428,  1.12951058],
[ 1.14223333,  0.93716017],
[ 0.57479975,  1.22568578],
[ 0.9530888 ,  1.03333537],
[ 0.9530888 ,  1.03333537],
[ 0.30929191,  0.78126988],
[-0.73100299,  1.3102335 ],
[ 2.00416471,  0.54173723],
[ 0.53829462,  1.28709506],
[ 2.10535588,  0.70452084],
[ 0.97048871,  1.28157207],
[-0.30297101,  1.47821624],
[-0.53940167,  1.59843524],
```

```
       [ 1.5445412 ,  1.20391767],
       [-0.67047905,  1.70793213],
       [-0.09226446,  1.456772  ],
       [ 0.7987441 ,  1.62598397],
       [ 0.54401726,  1.21893355],
       [ 0.72573385,  1.74880254],
       [-0.04914044,  1.60623393],
       [-0.2855711 ,  1.72645294],
       [-0.38014336,  1.77454054],
       [-0.47471562,  1.82262814],
       [ 1.09153659,  2.07696933],
       [-0.08318876,  2.05202016],
       [ 0.41785348,  2.09718433],
       [-0.14958011,  2.38570994],
       [ 0.38796725,  2.36946483],
       [ 1.18440355,  2.5867644 ]])
```

In [111…

```
scores_pca = pca.transform(segmentation_std)
```

In [112…

```
scores_pca
```

Out[112…

```
array([[-0.62700795, -1.24823477],
       [-0.53243569, -1.29632237],
       [-1.98009682, -1.13964325],
       [-1.83823843, -1.21177465],
       [-1.44916837, -1.36675958],
       [-1.87474355, -1.15036537],
       [-1.24924283, -1.4255693 ],
       [-1.81667641, -1.13704368],
       [ 1.54399202, -2.18074389],
       [-1.47489248, -1.26798481],
       [ 1.68585041, -2.25287529],
       [-1.23846182, -1.38820381],
       [-0.14009978, -1.90384576],
       [-1.74782827, -1.08635652],
       [ 0.27804746, -1.49419577],
       [-0.43124452, -1.13353876],
       [-1.21689981, -1.31347285],
       [-0.51503577, -1.04808567],
       [ 1.01968246, -1.74275634],
       [-1.19533779, -1.23874188],
       [ 0.22659923, -1.29664624],
       [-0.24626209, -1.05620823],
       [-0.65362833, -1.42849273],
       [ 0.0482357 , -1.16310555],
       [-0.24299625, -1.50874669],
       [-0.01399354, -1.0029215 ],
       [-0.66857144, -1.29235248],
       [ 0.26972325, -1.14718431],
       [-0.89422109, -1.13476799],
       [-1.69808534, -0.72602337],
       [ 1.47343857, -1.67354837],
       [-1.78187659, -0.64057029],
       [ 1.17477866, -1.39314532],
       [-0.48023595, -0.55161228],
       [-0.42552188, -1.20170027],
       [-1.74953357, -0.52847384],
       [-0.74574379, -0.99602818],
       [-1.31317738, -0.70750257],
       [-0.99711756, -0.73966893],
       [-1.75369567, -0.35496811],
       [ 0.38496127, -1.39957368],
```

```
[-0.14261413, -0.50904767],
[ 1.00303405, -1.04873341],
[-1.21198621, -0.54471896],
[-0.36083583, -0.97750737],
[-1.54298913, -0.37641235],
[-0.3027687 , -0.96418569],
[-1.39034973, -0.41117827],
[-1.29577747, -0.45926587],
[-1.2012052 , -0.50735347],
[-0.32849281, -0.86541093],
[ 0.32608509, -0.57597995],
[-1.16886218, -0.39525702],
[ 1.56630553, -1.16375329],
[-0.27042567, -0.85208924],
[ 0.99887194, -0.87522768],
[-0.21235854, -0.83876756],
[ 2.04994785, -1.36682582],
[-1.32566369, -0.18698537],
[ 1.31493176, -0.90739404],
[ 2.118796  , -1.31613866],
[-0.29279673, -0.0899048 ],
[ 0.5765626 , -1.11137193],
[-0.03815712, -0.79880252],
[ 1.80935509, -1.07310109],
[-0.31852085,  0.00886996],
[-0.54752356, -0.49695522],
[ 0.63462974, -1.09805025],
[-0.27123472, -0.01517384],
[-1.06767101, -0.23247341],
[ 2.15113902, -1.20404221],
[-0.34759803, -0.55576494],
[ 0.27790269, -0.83096888],
[ 0.27790269, -0.83096888],
[ 1.6848966 , -0.75273298],
[ 0.12445425,  0.04071245],
[-0.38826526, -0.32084993],
[ 0.7864601 , -0.29590076],
[-1.42856016,  0.2081137 ],
[-0.19912073, -0.41702513],
[ 1.59032434, -0.70464538],
[ 0.69188783, -0.24781316],
[ 2.06318566, -0.94508339],
[-0.34097912, -0.34489373],
[-1.52313242,  0.2562013 ],
[ 1.16474915, -0.48825117],
[ 0.11693908, -0.44919149],
[-1.44350327,  0.34425394],
[-0.86528868,  0.09309381],
[-0.10871057, -0.291607  ],
[ 0.75322081, -0.68702994],
[-0.19992977,  0.41989027],
[ 1.2294352 , -0.26405828],
[-0.56000987,  0.02356197],
[-0.93829893,  0.21591238],
[ 0.09456803,  0.31299295],
[-0.22900695, -0.14474463],
[-1.17472959,  0.33613139],
[ 1.2402162 , -0.22669279],
[-0.08379549,  0.44653364],
[-1.3423121 ,  0.50703756],
[-0.11287267, -0.11810127],
[ 2.14943372, -0.64615953],
[ 0.21070231,  0.33963631],
[ 1.29828334, -0.21337111],
[-1.43688437,  0.55512516],
```

```
[ 0.70177258, -0.48948041],
[ 1.54549501, -0.29622463],
[ 2.20750086, -0.63283785],
[ 2.11292859, -0.58475025],
[ 2.06564246, -0.56070645],
[-1.52067562,  0.64057824],
[-0.61145811,  0.22111151],
[-0.0987386 ,  0.58267389],
[-1.54639974,  0.73935301],
[-1.49911361,  0.71530921],
[ 0.5814762 , -0.34261804],
[-0.08052965, -0.00600482],
[ 0.03560463,  0.02063854],
[-0.0116815 ,  0.04468234],
[ 0.31189347,  0.50241993],
[-0.57911509,  0.33320795],
[-0.46298081,  0.35985132],
[ 0.90088907,  0.28862528],
[-1.25606404,  0.80596142],
[-0.87777499,  0.61361101],
[ 1.11159561,  0.26718104],
[ 0.96973722,  0.33931244],
[ 1.86817372, -0.11751978],
[ 0.87516495,  0.38740005],
[ 1.30074014,  0.17100584],
[ 0.92245109,  0.36335624],
[-1.13992977,  0.83260478],
[-0.85621298,  0.68834198],
[ 0.04557659,  0.89491943],
[-0.94000423,  0.77379506],
[-0.23071225,  0.41313804],
[ 0.61301018,  0.60639382],
[ 0.00907147,  0.95632871],
[-0.64550643,  0.66689774],
[ 0.40556948,  0.1752996 ],
[ 0.63457219,  0.68112478],
[-0.95494734,  0.90993531],
[-0.76580281,  0.81376011],
[ 0.32513128,  0.92416236],
[ 0.46698968,  0.85203095],
[ 1.41271232,  0.37115493],
[-0.75502181,  0.85112559],
[-0.64966854,  0.84040347],
[ 0.76148748,  0.74513363],
[ 1.18706266,  0.52873942],
[ 0.99791814,  0.62491462],
[-0.17680722,  0.59996546],
[-0.46052401,  0.74422827],
[-0.03494882,  0.52783406],
[-0.98067146,  1.00871008],
[ 0.90334587,  0.67300223],
[-0.83881306,  0.93657867],
[ 0.76148748,  0.74513363],
[-0.83881306,  0.93657867],
[ 0.40140737,  0.34880533],
[-0.87531819,  0.99798796],
[ 0.08453852,  1.21788709],
[-0.75918391,  1.02463132],
[ 1.59353264,  0.62199119],
[-0.47962922,  1.05387425],
[ 1.22602459,  0.85170708],
[-0.61070661,  1.16337113],
[-0.45806721,  1.12860521],
[ 0.76394428,  1.12951058],
[ 1.14223333,  0.93716017],
```

```
       [ 0.57479975,  1.22568578],
       [ 0.9530888 ,  1.03333537],
       [ 0.9530888 ,  1.03333537],
       [ 0.30929191,  0.78126988],
       [-0.73100299,  1.3102335 ],
       [ 2.00416471,  0.54173723],
       [ 0.53829462,  1.28709506],
       [ 2.10535588,  0.70452084],
       [ 0.97048871,  1.28157207],
       [-0.30297101,  1.47821624],
       [-0.53940167,  1.59843524],
       [ 1.5445412 ,  1.20391767],
       [-0.67047905,  1.70793213],
       [-0.09226446,  1.456772  ],
       [ 0.7987441 ,  1.62598397],
       [ 0.54401726,  1.21893355],
       [ 0.72573385,  1.74880254],
       [-0.04914044,  1.60623393],
       [-0.2855711 ,  1.72645294],
       [-0.38014336,  1.77454054],
       [-0.47471562,  1.82262814],
       [ 1.09153659,  2.07696933],
       [-0.08318876,  2.05202016],
       [ 0.41785348,  2.09718433],
       [-0.14958011,  2.38570994],
       [ 0.38796725,  2.36946483],
       [ 1.18440355,  2.5867644 ]])
```

Each observation is explained by the 2 components. Each column represents a component.

K-Means Clustering with Principal Component Analysis (PCA)

Fit K-means using transformed data from PCA

In [113…

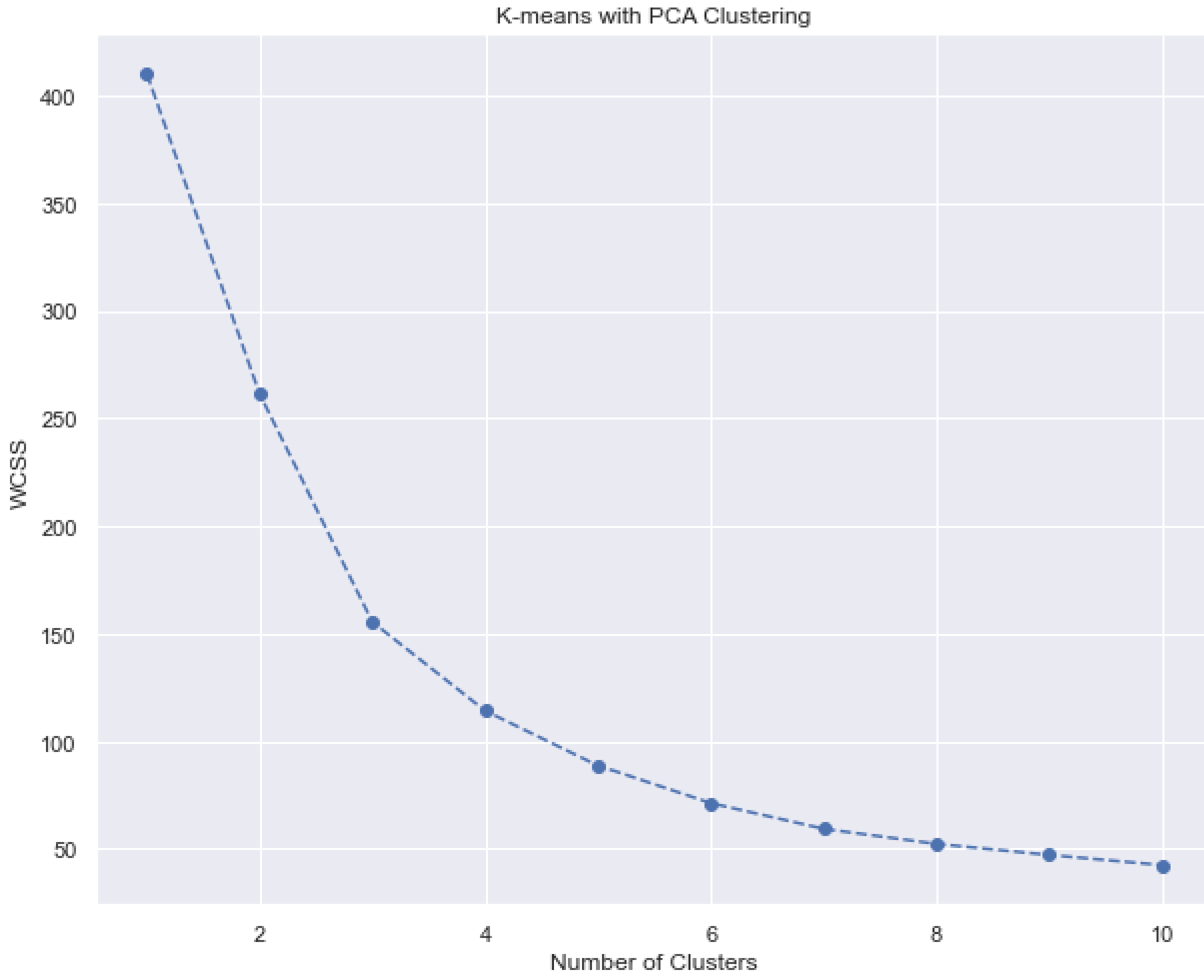
```
wcss = []
for i in range(1,11):
    kmeans_pca = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans_pca.fit(scores_pca)
    wcss.append(kmeans_pca.inertia_)
```

Component scores are standarized by definition.

Plot Within Cluster Sum of Squares for the K-means PCA model

In [114…

```
plt.figure(figsize = (10,8))
plt.plot(range(1, 11), wcss, marker = 'o', linestyle = '--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.title('K-means with PCA Clustering')
plt.show()
```



Decide on number of clusters based on K-means with PCA clusting graph. Our decision does not change, we decide on 3 clusters.

Create a PCA K-means model with 3 clusters

In [115…

```
kmeans_pca = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
```

Fit data with the K-means PCA model

In [116…

```
kmeans_pca.fit(scores_pca)
```

Out[116…

```
KMeans(n_clusters=3, random_state=42)
```

K-Means Clustering with PCA Results

Create new dataframe with original features adding PCA scores and clusters

In [117…

```
df_segment_pca_kmeans = pd.concat([df_segmentation.reset_index(drop = True), pd.DataFra
df_segment_pca_kmeans.columns.values[-2: ] = ['Component 1', 'Component 2']
df_segment_pca_kmeans['Segment K-means PCA'] = kmeans_pca.labels_
df_segment_pca_kmeans.head(5)
```

Out[117...

	CustomerID	Gender	Age	Income	Spending_Score	Gender_m	Gender_f	generations	income_brack
0	1	Male	19	15	39	1	0	GenZ	$15-24
1	2	Male	21	15	81	1	0	GenZ	$15-24
2	3	Female	20	16	6	0	1	GenZ	$15-24
3	4	Female	23	16	77	0	1	GenZ	$15-24
4	5	Female	31	17	40	0	1	Millennials	$15-24

Means by Segments

In [118...

```
df_segment_pca_kmeans_freq = df_segment_pca_kmeans.groupby(['Segment K-means PCA']).mea
df_segment_pca_kmeans_freq
```

Out[118...

Segment K-means PCA	CustomerID	Age	Income	Spending_Score	Gender_m	Gender_f	Component 1	Compor
0	51.986301	33.082192	39.684932	53.328767	0.164384	0.835616	-0.874172	-0.584
1	84.680000	54.620000	53.640000	42.180000	0.840000	0.160000	1.248117	-0.645
2	155.626667	34.160000	83.453333	52.493333	0.426667	0.573333	0.018783	0.999

Size of each cluster and proportion to data set

In [119...

```
df_segment_pca_kmeans_freq['N Obs'] = df_segment_pca_kmeans[['Segment K-means PCA','Age
df_segment_pca_kmeans_freq['Prop Obs'] = df_segment_pca_kmeans_freq['N Obs'] / df_segme
df_segment_pca_kmeans_freq = df_segment_pca_kmeans_freq.rename({0:'Segment 1',
                                                                 1:'Segment 2',
                                                                 2:'Segment 3'})

df_segment_pca_kmeans_freq
```

Out[119...

Segment K-means PCA	CustomerID	Age	Income	Spending_Score	Gender_m	Gender_f	Component 1	Compor
Segment 1	51.986301	33.082192	39.684932	53.328767	0.164384	0.835616	-0.874172	-0.584
Segment 2	84.680000	54.620000	53.640000	42.180000	0.840000	0.160000	1.248117	-0.645

	CustomerID	Age	Income	Spending_Score	Gender_m	Gender_f	Component 1	Compor
Segment K-means PCA								
Segment 3	155.626667	34.160000	83.453333	52.493333	0.426667	0.573333	0.018783	0.999

Segment 1 second to largest subgroup with 36.7%

Segment 2 smallest subgroup with 25.3%

Segment 3 largest subgroup with 37.9%

Add Labels

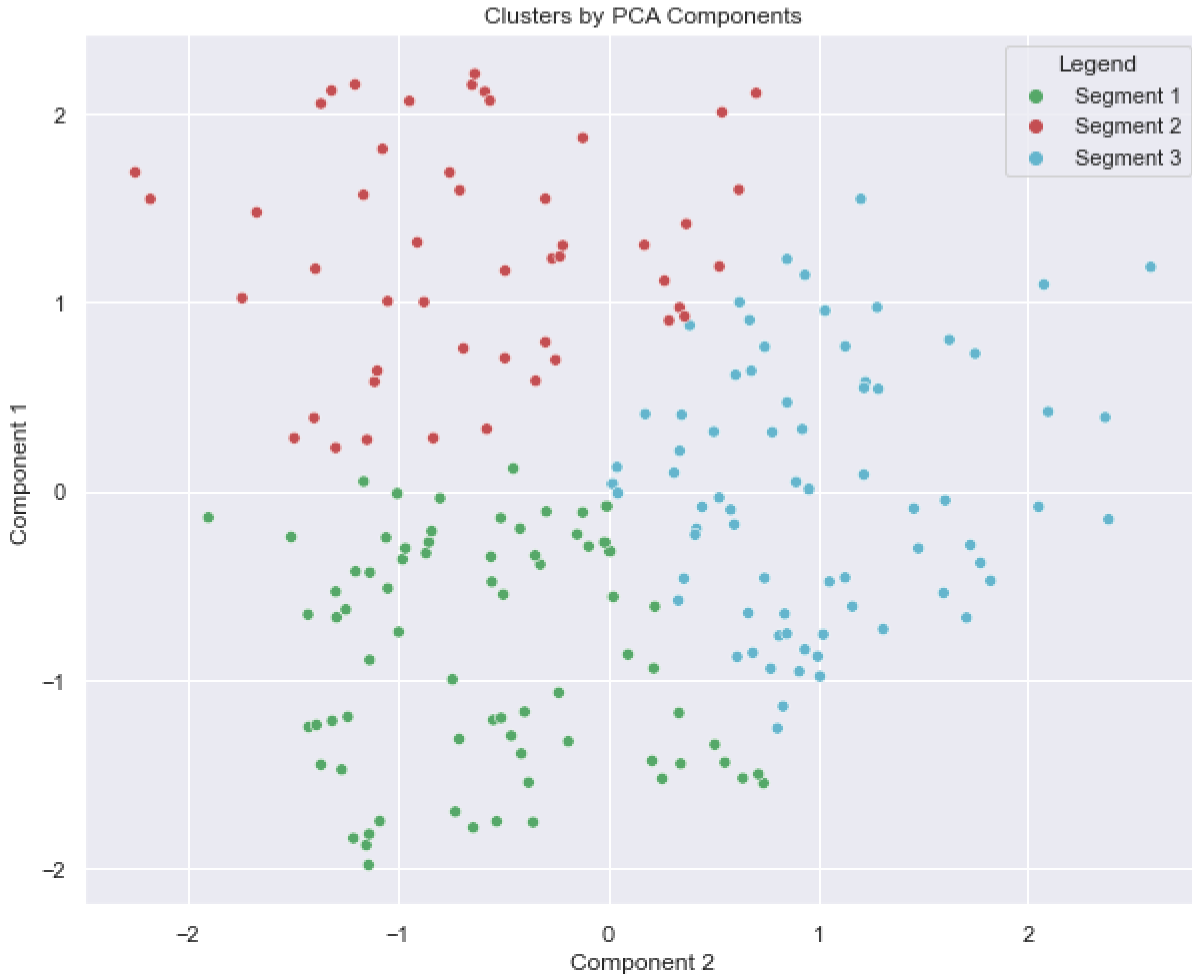
In [120…

```
df_segment_pca_kmeans['Legend'] = df_segment_pca_kmeans['Segment K-means PCA'].map({0:'
                                                                                    1:'
                                                                                    2:'
```

Plot data by PCA components

In [121…

```
x_axis = df_segment_pca_kmeans['Component 2']
y_axis = df_segment_pca_kmeans['Component 1']
plt.figure(figsize = (10, 8))
sn.scatterplot(x_axis, y_axis, hue = df_segment_pca_kmeans['Legend'], palette = ['g', '
plt.title('Clusters by PCA Components')
plt.show()
```



The Y axis is the first component and X axis is the second component

The biggest goal of PCA is for the division of subgroups to be more pronounced by reducing the number of features by combining them.

Descriptive Analysis

Create final dataset from PCA analysis

In [122…

```
df_final_data = df_segment_pca_kmeans
```

Examine column labels

In [123…

```
df_final_data.columns
```

Out[123…

```
Index(['CustomerID', 'Gender', 'Age', 'Income', 'Spending_Score', 'Gender_m',
       'Gender_f', 'generations', 'income_brackets', 'Component 1',
       'Component 2', 'Segment K-means PCA', 'Legend'],
      dtype='object')
```

Rename columns

In [124…

```
df_final_data.columns = ['CustomerID','Gender', 'Age', 'Income', 'Spending_Score', 'Gen
                         'Component_1', 'Component_2', 'Segment_K-means_PCA', 'Legend']
```

View first 5 rows

In [125…

```
df_final_data.head(5)
```

Out[125…

	CustomerID	Gender	Age	Income	Spending_Score	Gender_Male	Gender_Female	generations	incoı
0	1	Male	19	15	39	1	0	GenZ	
1	2	Male	21	15	81	1	0	GenZ	
2	3	Female	20	16	6	0	1	GenZ	
3	4	Female	23	16	77	0	1	GenZ	
4	5	Female	31	17	40	0	1	Millennials	

View last 5 rows

In [126…

```
df_final_data.tail(5)
```

Out[126…

	CustomerID	Gender	Age	Income	Spending_Score	Gender_Male	Gender_Female	generations	in
193	194	Female	38	113	91	0	1	Millennials	
194	195	Female	47	120	16	0	1	GenX	
195	196	Female	35	120	79	0	1	Millennials	
196	197	Female	45	126	28	0	1	GenX	
197	198	Male	32	126	74	1	0	Millennials	

Identify columns, null values, and data types

In [127…

```
df_final_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 198 entries, 0 to 197
Data columns (total 13 columns):
```

```
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   CustomerID          198 non-null    int64  
 1   Gender              198 non-null    object 
 2   Age                 198 non-null    int64  
 3   Income              198 non-null    int64  
 4   Spending_Score      198 non-null    int64  
 5   Gender_Male         198 non-null    int64  
 6   Gender_Female       198 non-null    int64  
 7   generations         198 non-null    object 
 8   income_brackets     198 non-null    object 
 9   Component_1         198 non-null    float64
 10  Component_2         198 non-null    float64
 11  Segment_K-means_PCA  198 non-null    int32  
 12  Legend              198 non-null    object 
dtypes: float64(2), int32(1), int64(6), object(4)
memory usage: 19.5+ KB
```

Descriptive Statistics

In [128…

```
df_final_data.describe().transpose()
```

Out[128…

	count	mean	std	min	25%	50%	75%	max
CustomerID	198.0	9.950000e+01	57.301832	1.000000	50.250000	99.500000	148.750000	198.00000
Age	198.0	3.892929e+01	14.016852	18.000000	28.250000	36.000000	49.000000	70.00000
Income	198.0	5.978788e+01	25.237259	15.000000	40.500000	61.000000	77.750000	126.00000
Spending_Score	198.0	5.019697e+01	25.746846	1.000000	35.000000	50.000000	72.750000	99.00000
Gender_Male	198.0	4.343434e-01	0.496927	0.000000	0.000000	0.000000	1.000000	1.00000
Gender_Female	198.0	5.656566e-01	0.496927	0.000000	0.000000	1.000000	1.000000	1.00000
Component_1	198.0	-7.737918e-17	1.038762	-1.980097	-0.752702	-0.110792	0.763330	2.20750
Component_2	198.0	-1.054151e-16	1.001131	-2.252875	-0.836818	0.001433	0.744907	2.58676
Segment_K-means_PCA	198.0	1.010101e+00	0.866699	0.000000	0.000000	1.000000	2.000000	2.00000

Count of unique values in each column for categorical variables

In [129…

```
categorical_variables = ["Gender", "Gender_Male", "Gender_Female", "generations", "inco
for column in categorical_variables:
    print(df_final_data[column].value_counts())
    print("-" * 40)
```

```
Female    112
Male       86
Name: Gender, dtype: int64
----------------------------------------
0    112
1     86
Name: Gender_Male, dtype: int64
----------------------------------------
1    112
```

```
0     86
Name: Gender_Female, dtype: int64
----------------------------------------
Millennials    78
GenX           48
GenZ           46
Boomer         26
Name: generations, dtype: int64
----------------------------------------
$50-74.9K      68
$75-99.9K      46
$35-49.9K      34
$15-24.9K      22
$25-34.9K      16
$100-149.9K    12
Name: income_brackets, dtype: int64
----------------------------------------
Segment 3    75
Segment 1    73
Segment 2    50
Name: Legend, dtype: int64
----------------------------------------
```

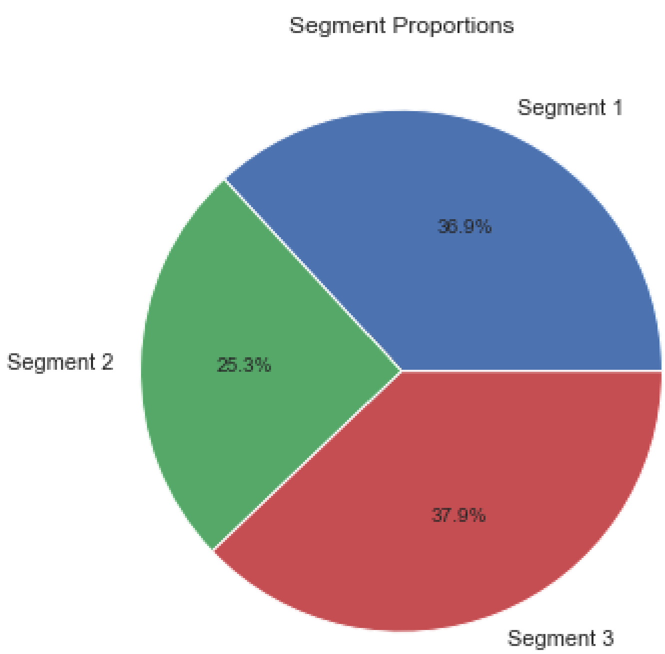
Descriptive Analysis by Segments

Segments

Segment Distribution

In [130…

```
seg_prop = df_final_data[['CustomerID', 'Legend']].groupby(['Legend']).count() / df_fin
plt.figure(figsize = (9, 6))
plt.pie(seg_prop['CustomerID'],
        labels = ['Segment 1', 'Segment 2', 'Segment 3'],
        autopct = '%1.1f%%',
        colors = ('b', 'g', 'r'))
plt.title('Segment Proportions');
```

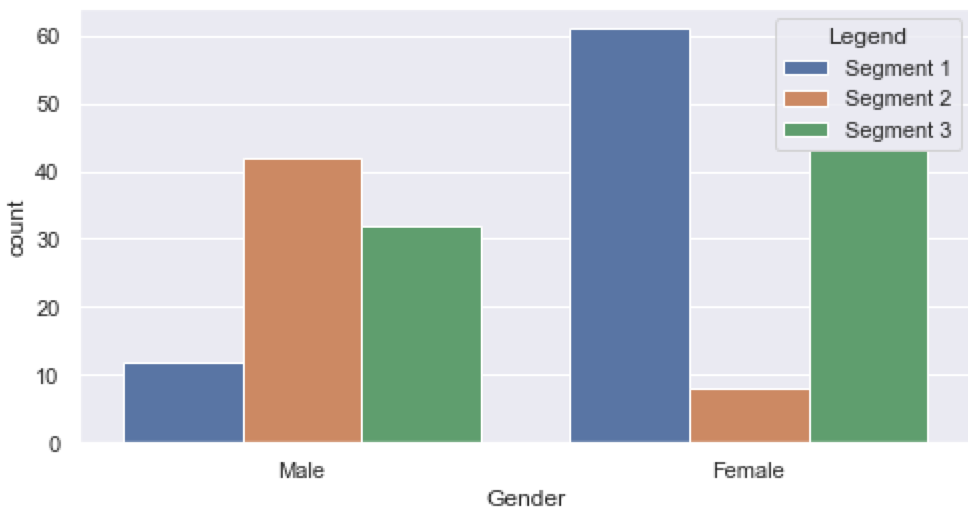


Segment 3 makes up the majority of the subgroups followed by Segment 1 then Segment 2.

Gender by Segments

In [131…

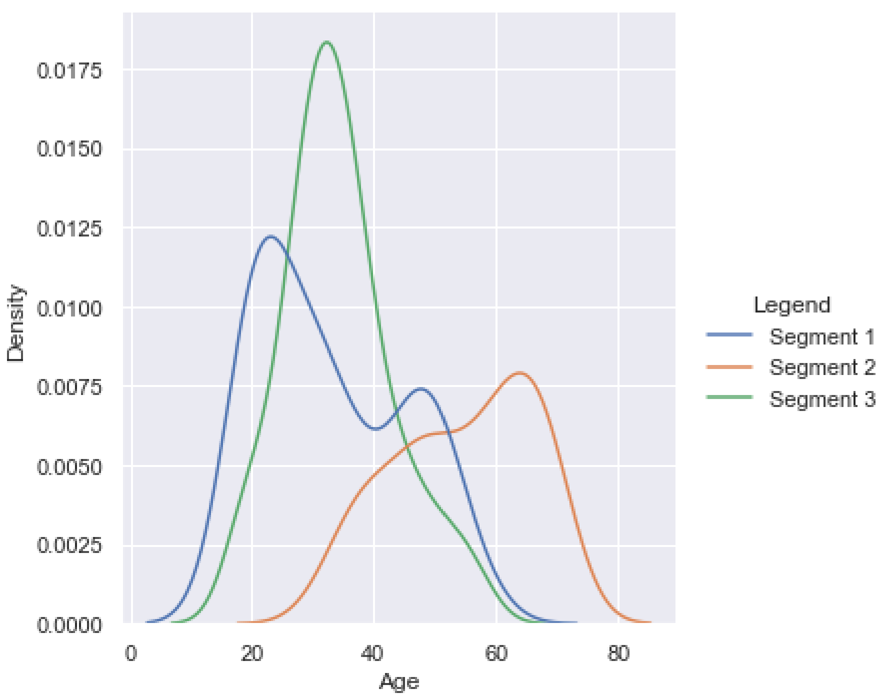
```
sn.countplot(x="Gender", hue="Legend", data=df_final_data);
```



Age by Segments

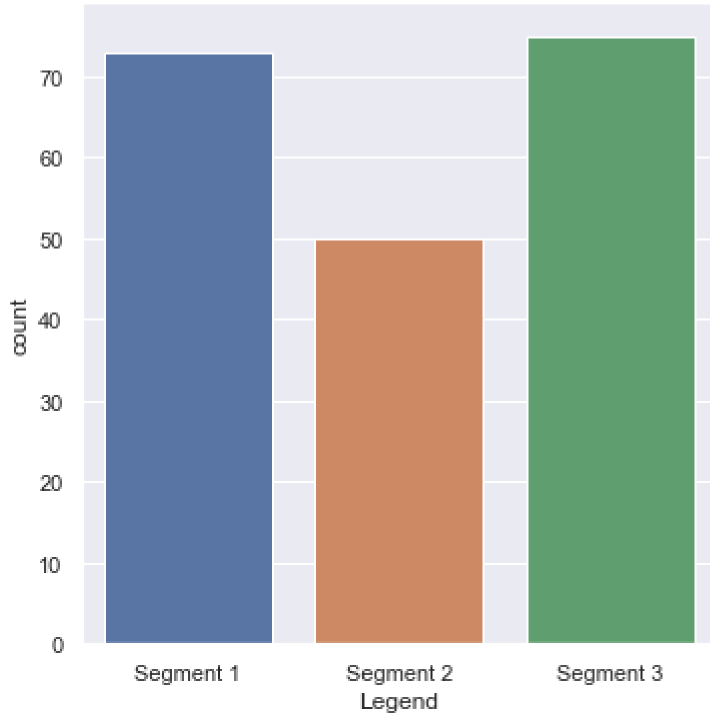
In [132…

```
sn.displot(data=df_final_data, x="Age", hue="Legend", kind="kde");
```



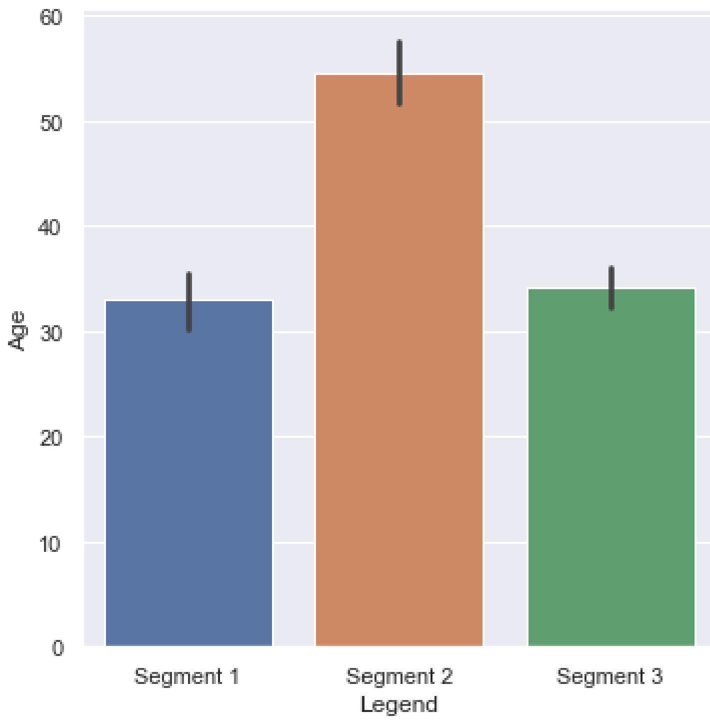
In [133…

```
sn.catplot(data=df_final_data, x="Legend", kind="count");
```



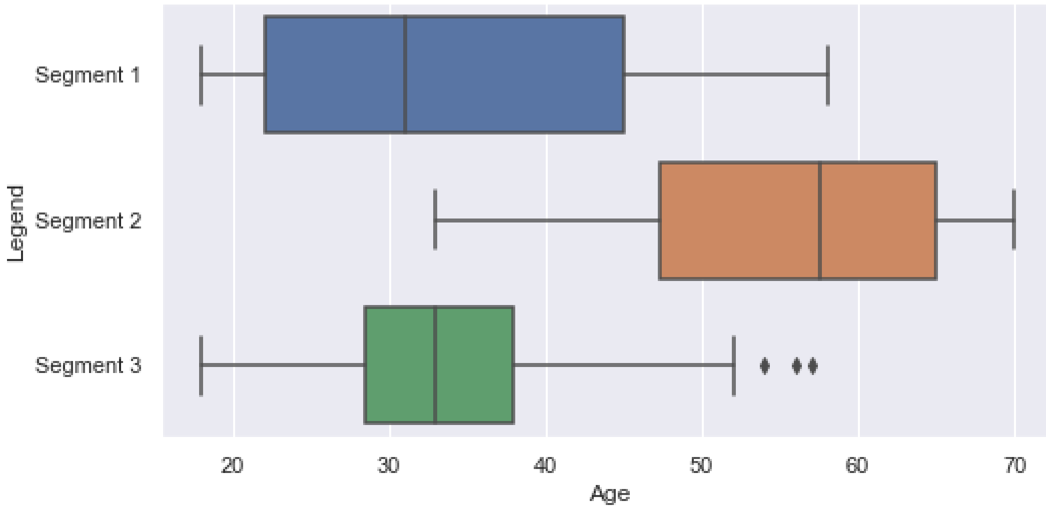
In [134…

```
sn.catplot(data=df_final_data, x="Legend", y="Age", kind="bar");
```



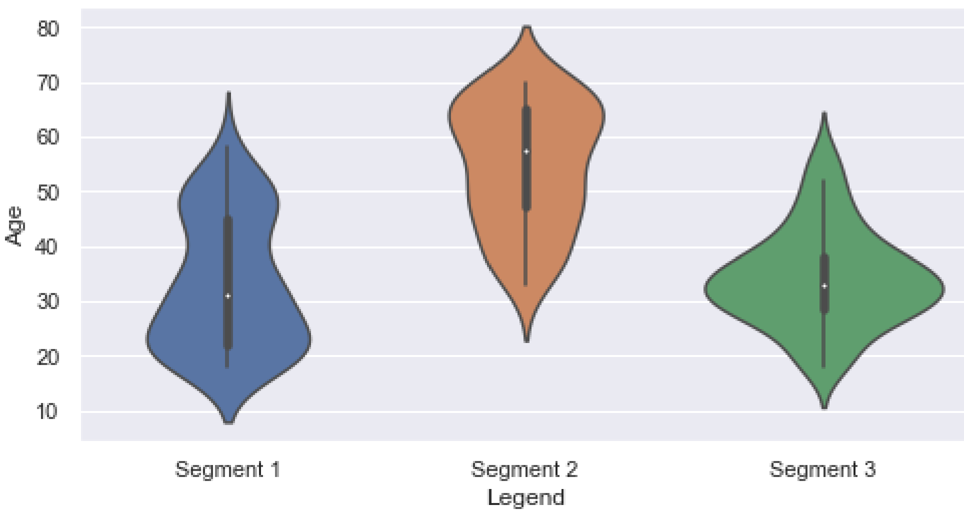
In [135…

```
sn.boxplot(x="Age", y="Legend",data=df_final_data);
```



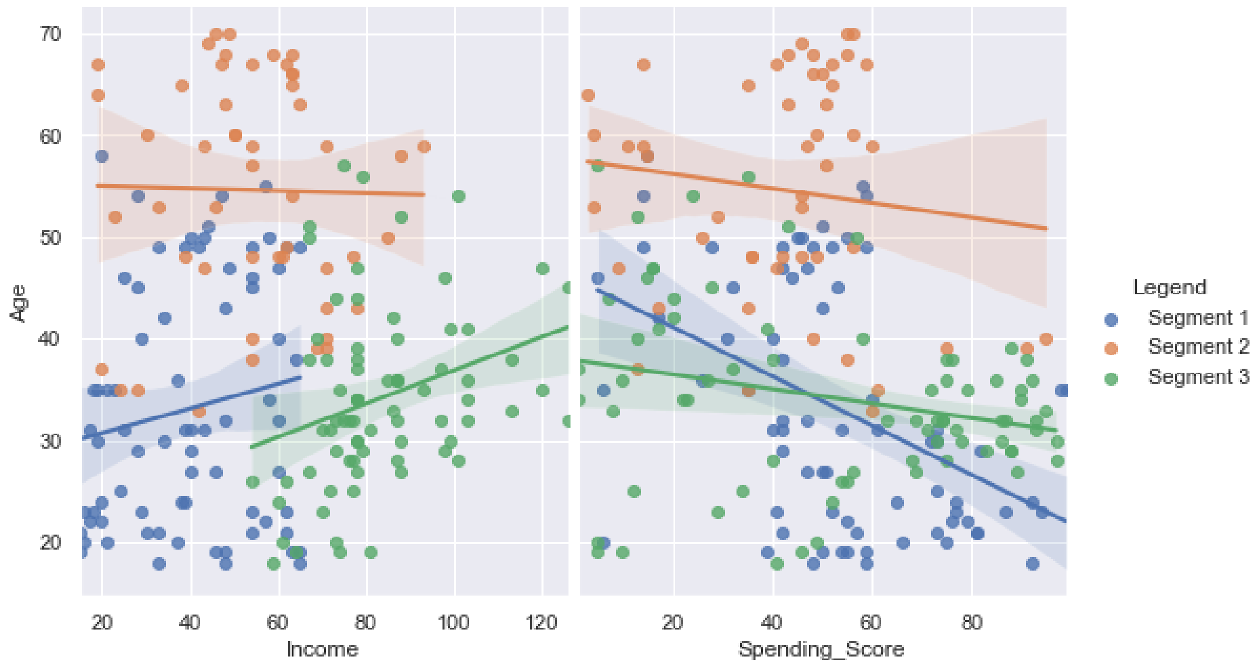
In [136…

```
sn.violinplot(data=df_final_data, x="Legend", y="Age");
```



In [137…

```
sn.pairplot(df_final_data, x_vars=["Income", "Spending_Score"], y_vars=["Age"],
             hue="Legend", height=5, aspect=.8, kind="reg");
```

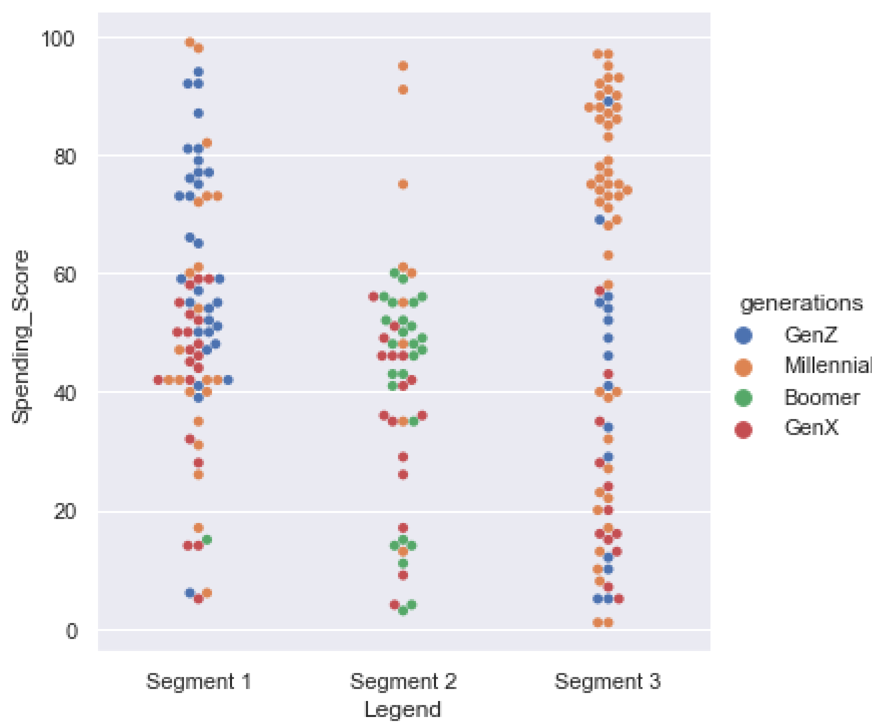


Negative relationship between Spending Scores and Age. Younger customers spending more, on average. Most significantly across Segment 1.

Positive relationship between Income and Age. Older customers having higher income levels especially for Segment 1 and 3 with Segment 1 having only slight relationship.

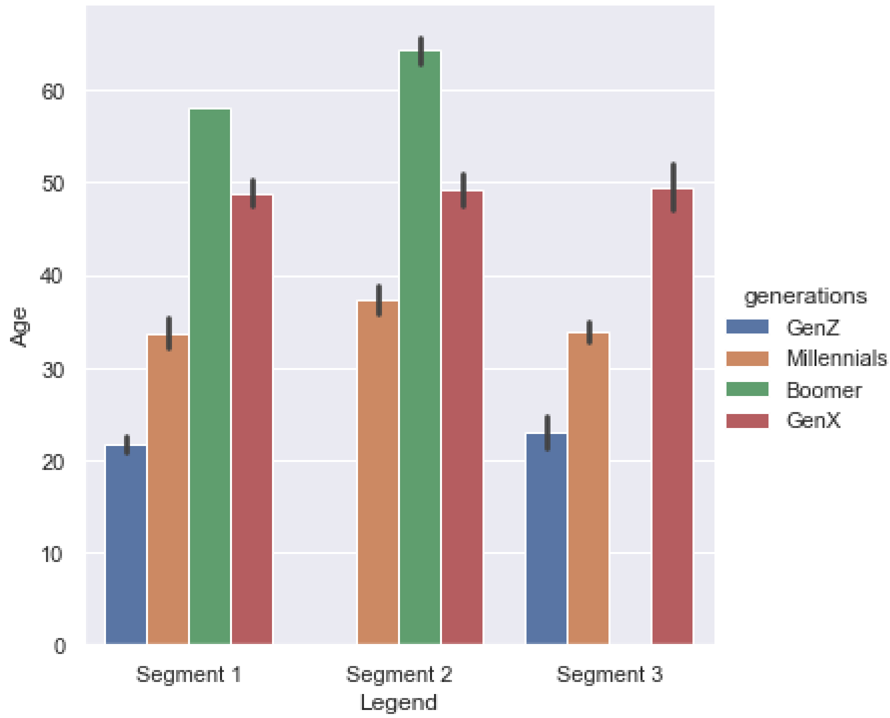
In [138...

```
sn.catplot(data=df_final_data, x="Legend", y="Spending_Score", hue="generations", kind=
```



In [139...

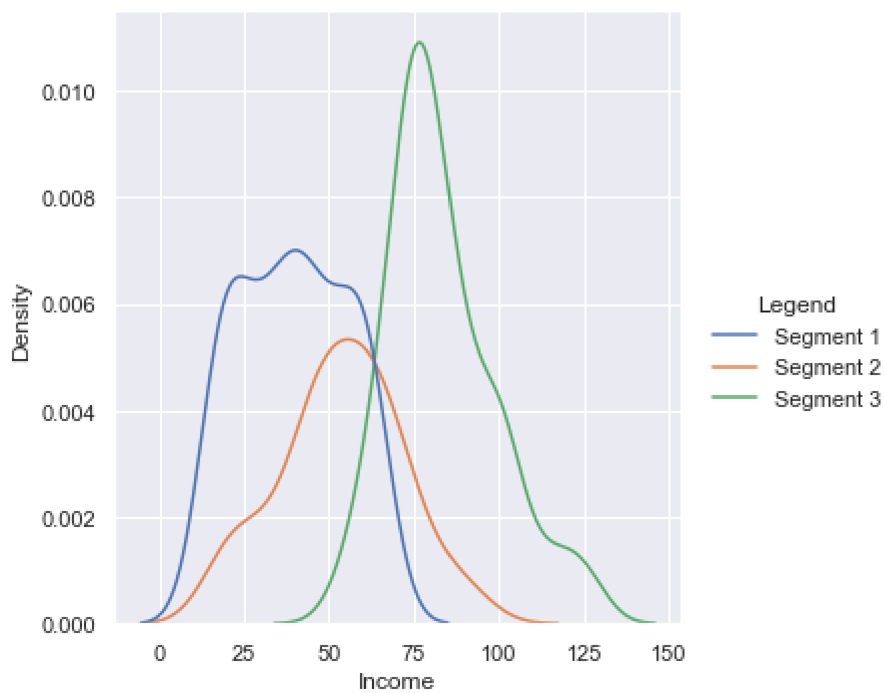
```
sn.catplot(data=df_final_data, x="Legend", y="Age", hue="generations", kind="bar");
```



Income by Segments

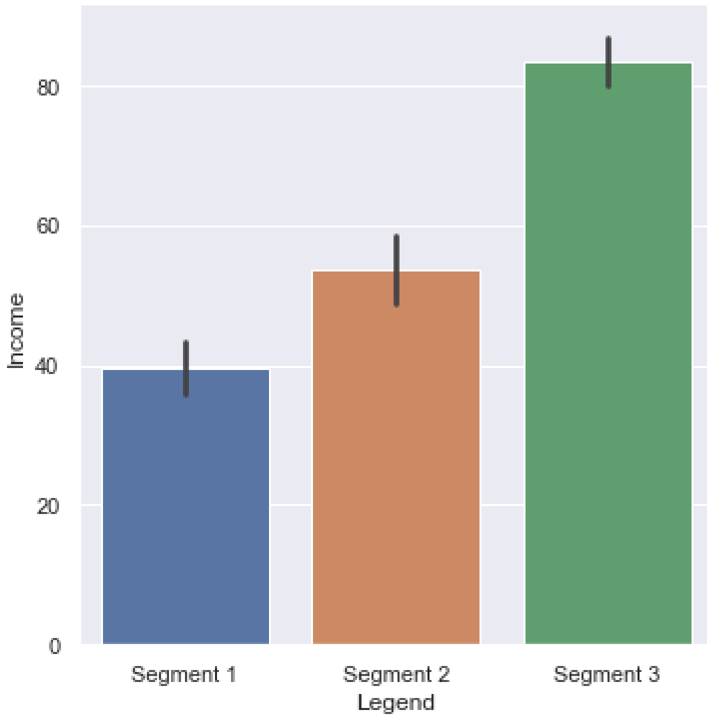
In [140…

```
sn.displot(data=df_final_data, x="Income", hue="Legend", kind="kde");
```



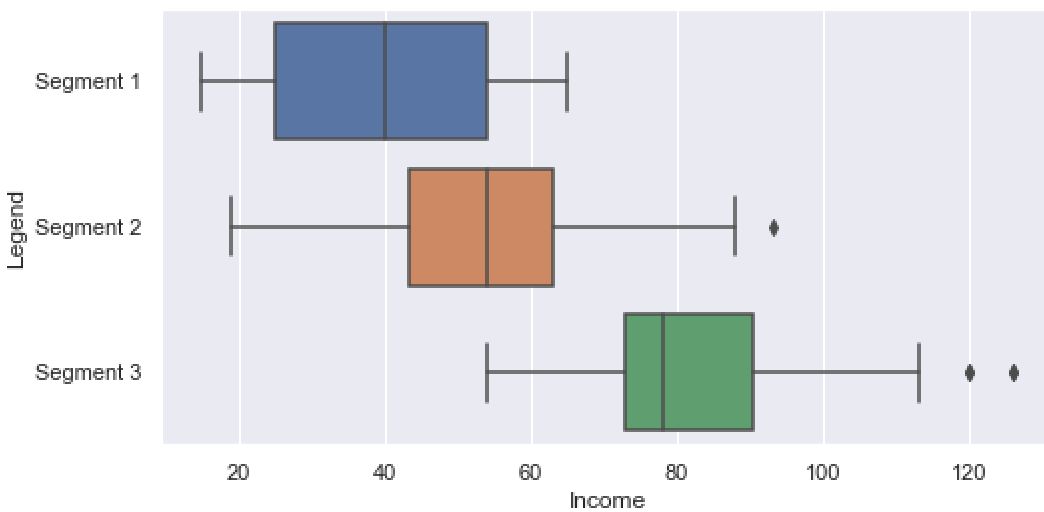
In [141…

```
sn.catplot(data=df_final_data, x="Legend", y="Income", kind="bar");
```



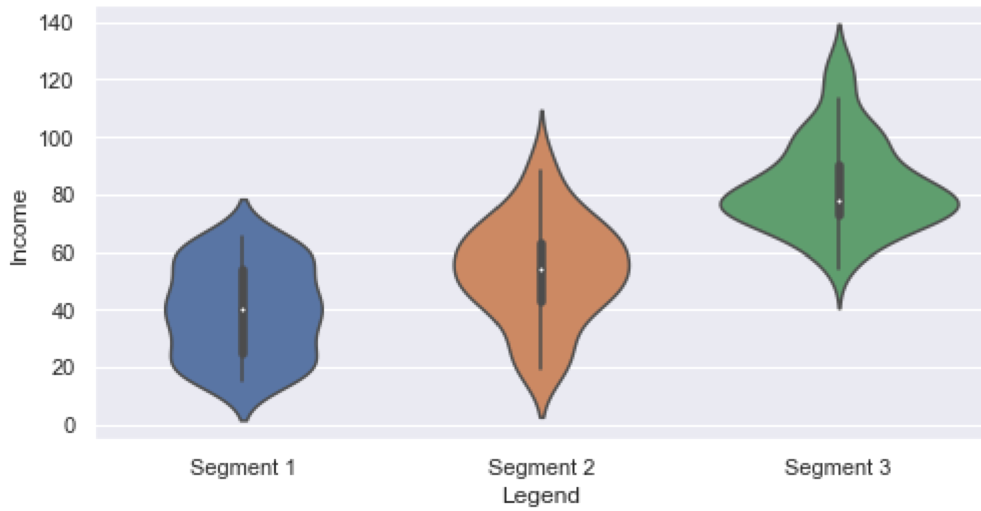
In [142…

```
sn.boxplot(x="Income", y="Legend",data=df_final_data);
```



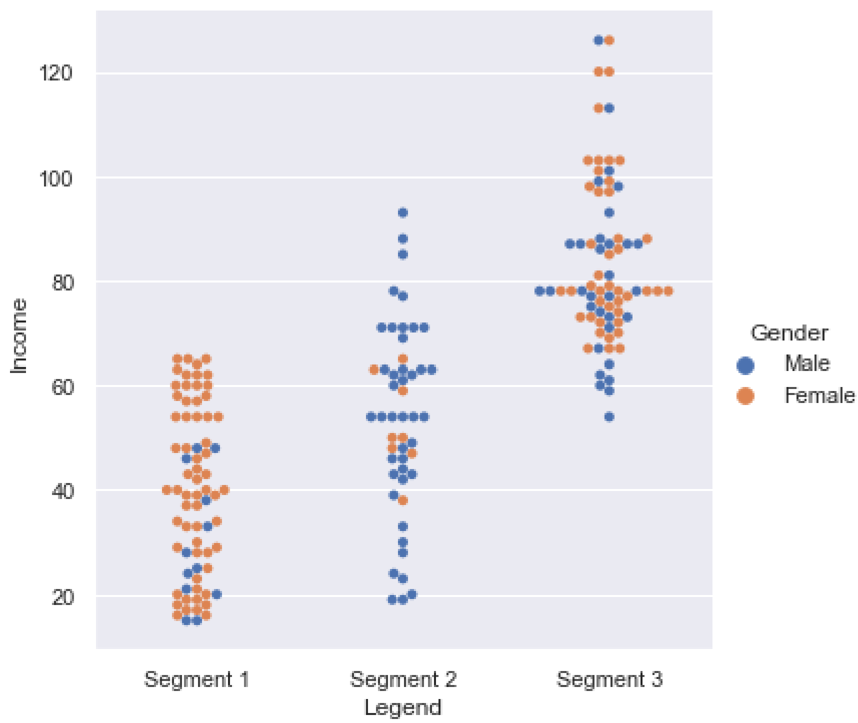
In [143…

```
sn.violinplot(data=df_final_data, x="Legend", y="Income");
```



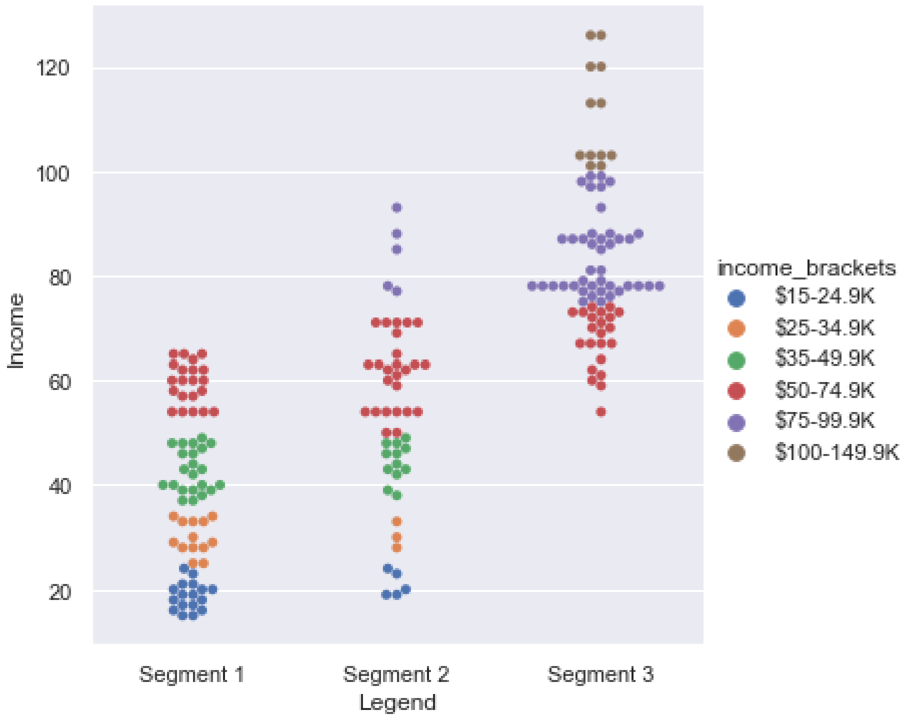
In [144…

```
sn.catplot(data=df_final_data, x="Legend", y="Income", hue="Gender", kind="swarm");
```



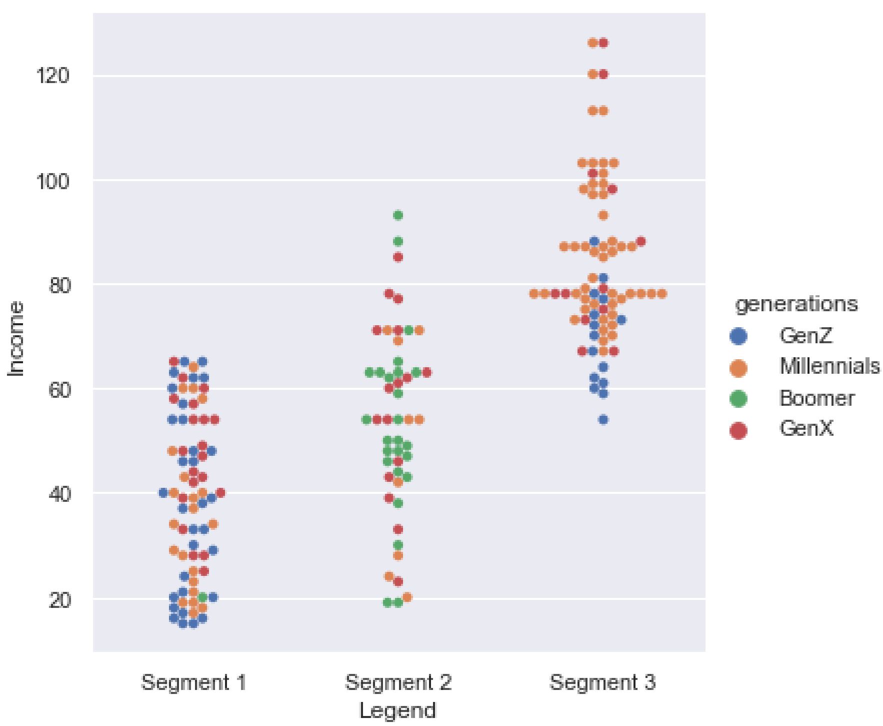
In [145…

```
sn.catplot(data=df_final_data, x="Legend", y="Income", hue="income_brackets", kind="swa
```



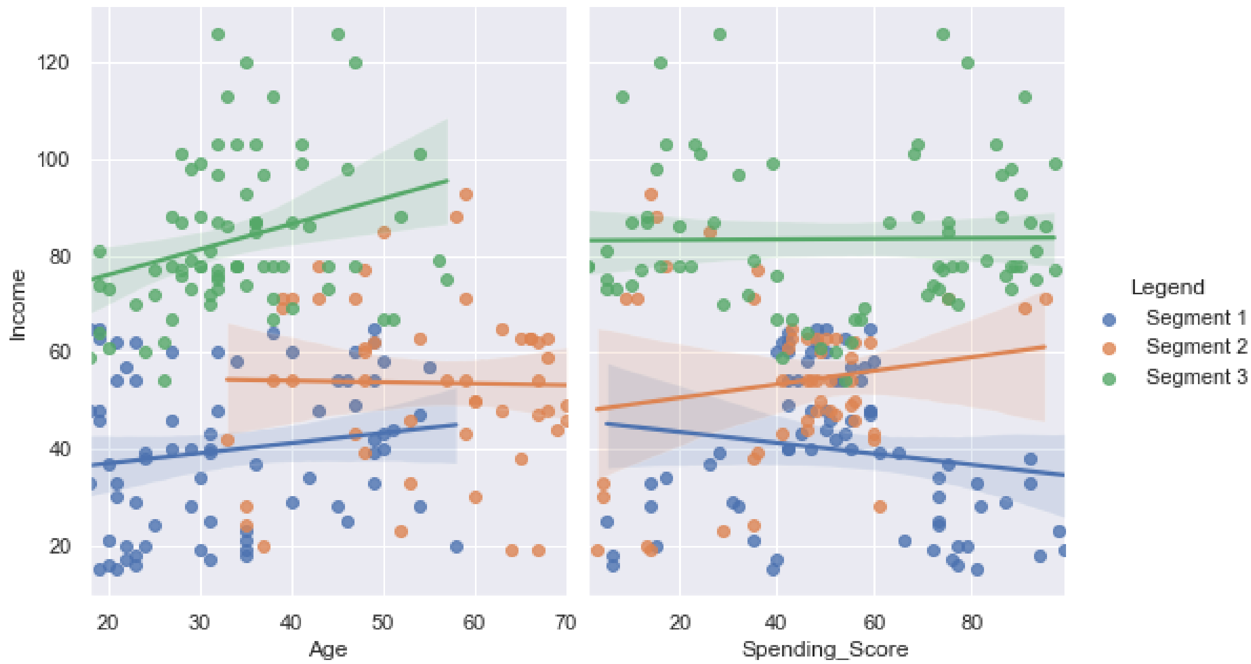
In [146…

```
sn.catplot(data=df_final_data, x="Legend", y="Income", hue="generations", kind="swarm")
```



In [147…

```
sn.pairplot(df_final_data, x_vars=["Age", "Spending_Score"], y_vars=["Income"],
            hue="Legend", height=5, aspect=.8, kind="reg");
```



Segment 1 slightly decreases in Spending Score as Income increases with greater variability at the outer ends.

Segment 2 increases in Spending Score as Income increases with a high degree of variability at the outer ends.

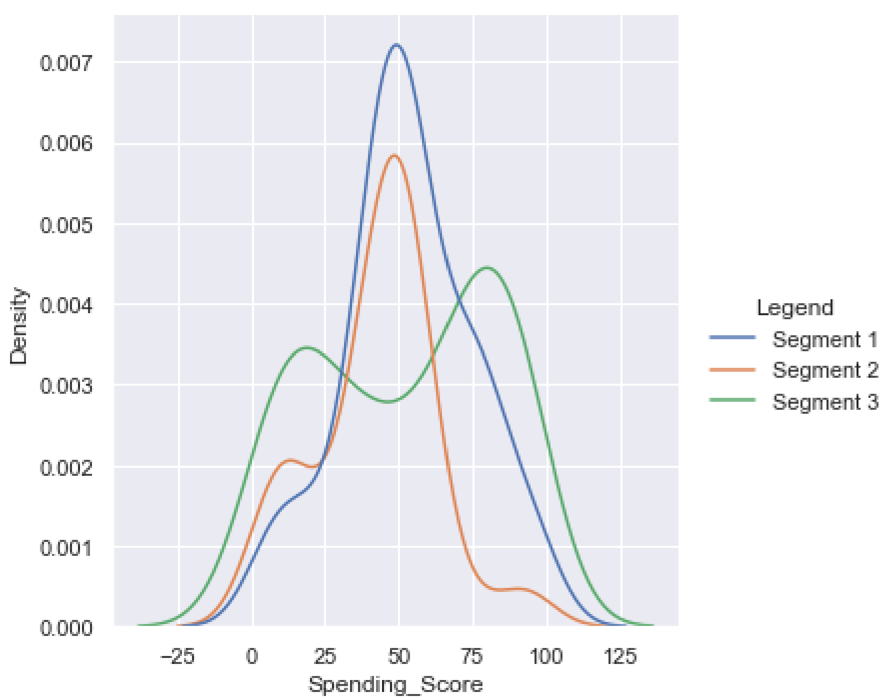
Segment 3 Spending Score remain consistent across Income with low degree of variability.

Income Increases with Age across Segment 1 and 3, however, remains constant in Segment 2.

Spending Score by Segments

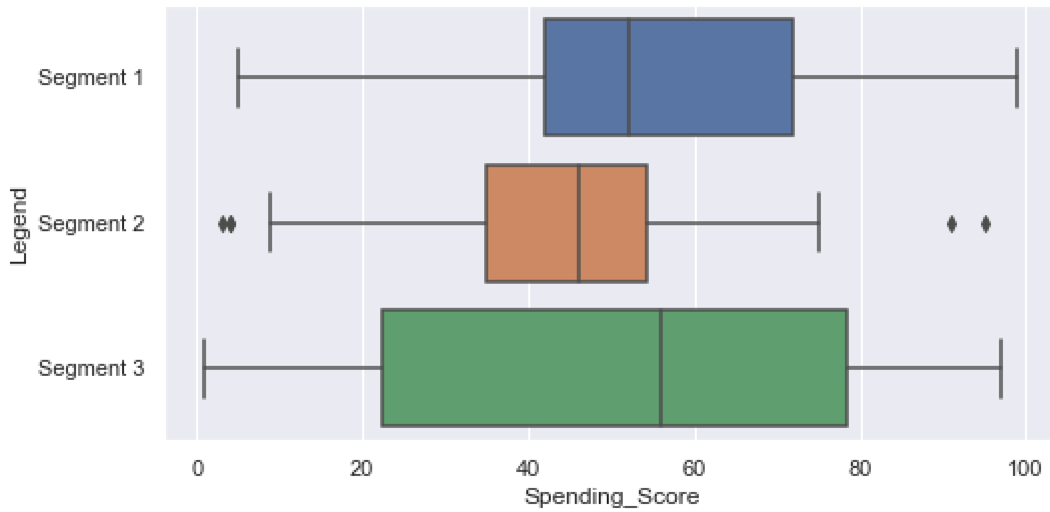
In [148…

```
sn.displot(data=df_final_data, x="Spending_Score", hue="Legend", kind="kde");
```



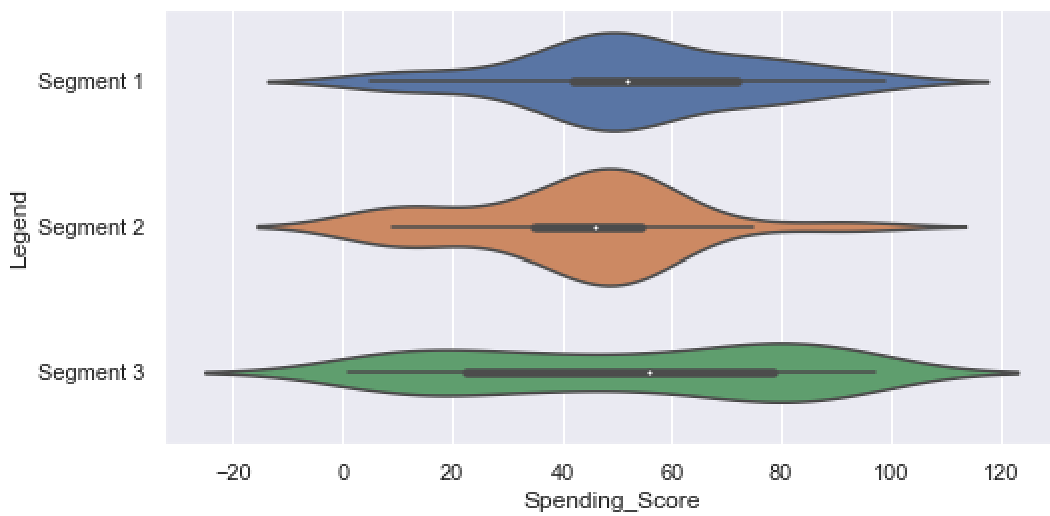
In [149…

```
sn.boxplot(x="Spending_Score", y="Legend",data=df_final_data);
```



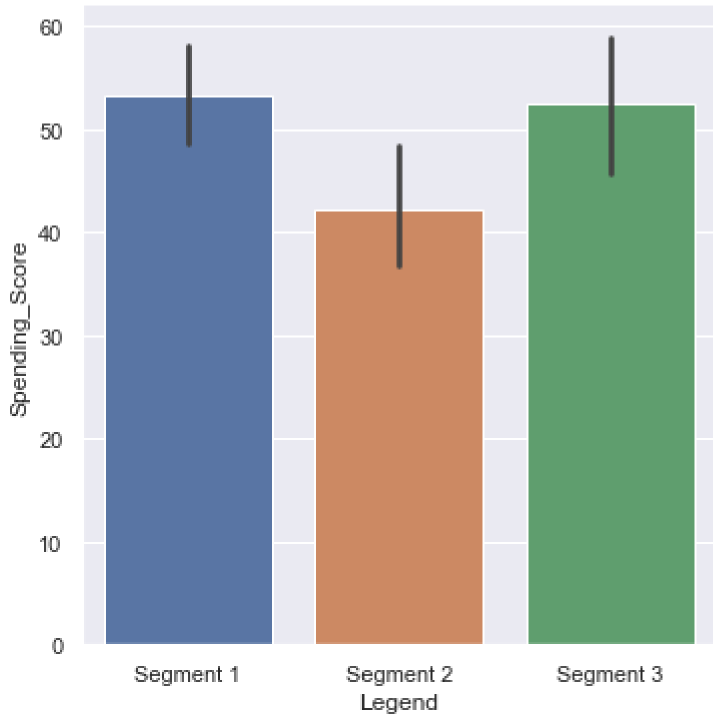
In [150…

```
sn.violinplot(x="Spending_Score", y="Legend",data=df_final_data);
```



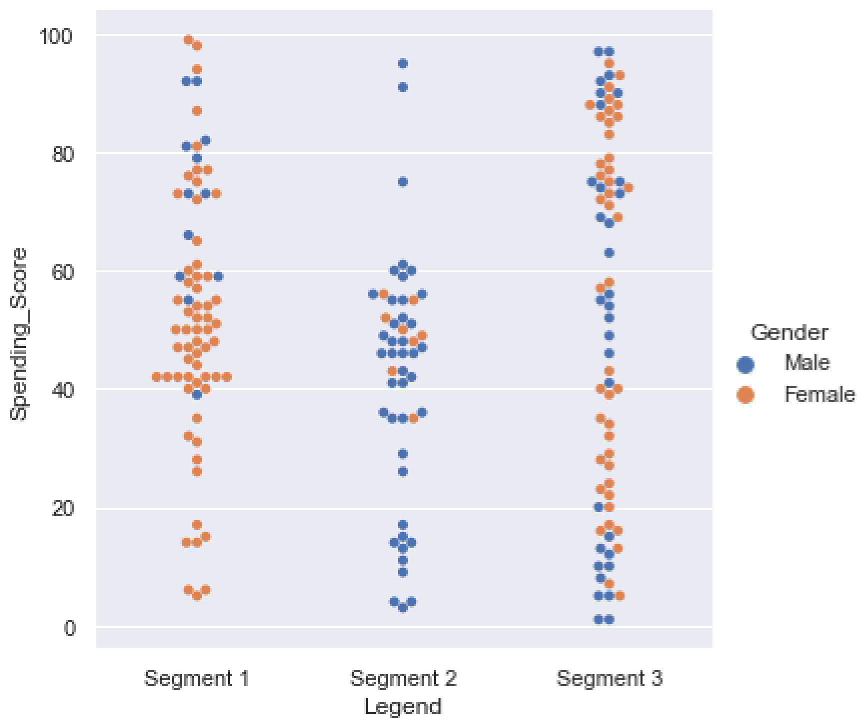
In [151…

```
sn.catplot(data=df_final_data, x="Legend", y="Spending_Score", kind="bar");
```



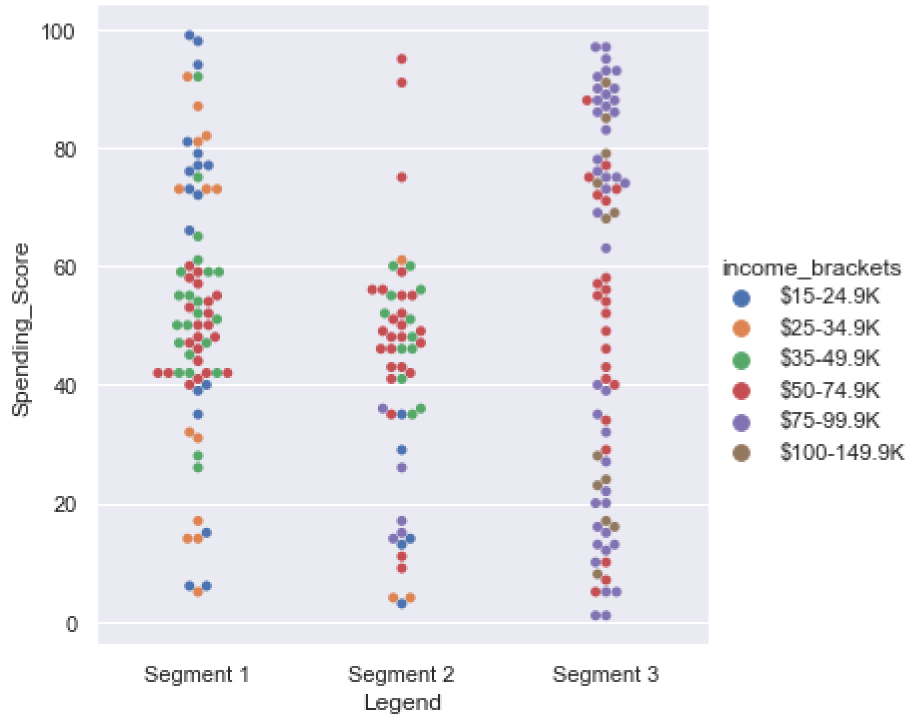
In [152…

```
sn.catplot(data=df_final_data, x="Legend", y="Spending_Score", hue="Gender", kind="swar
```



In [153…

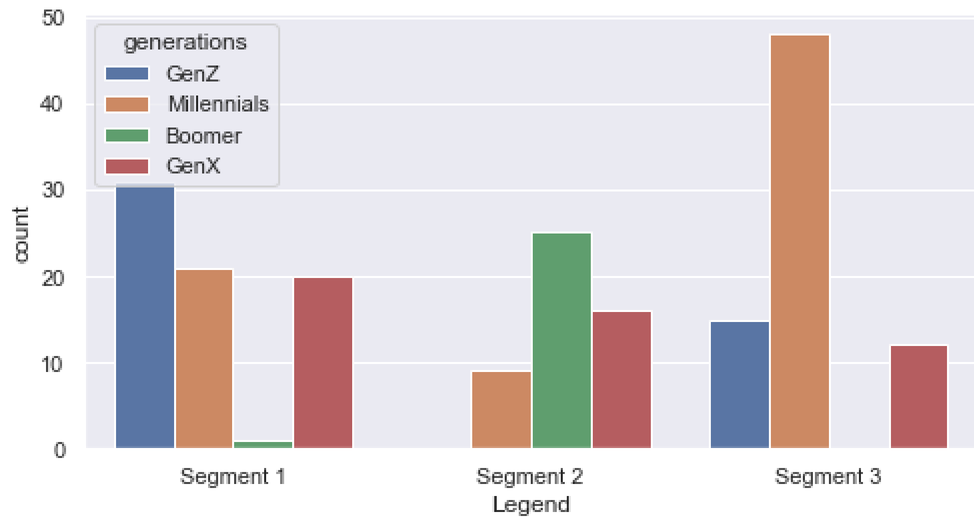
```
sn.catplot(data=df_final_data, x="Legend", y="Spending_Score", hue="income_brackets", k
```



Generation by Segments

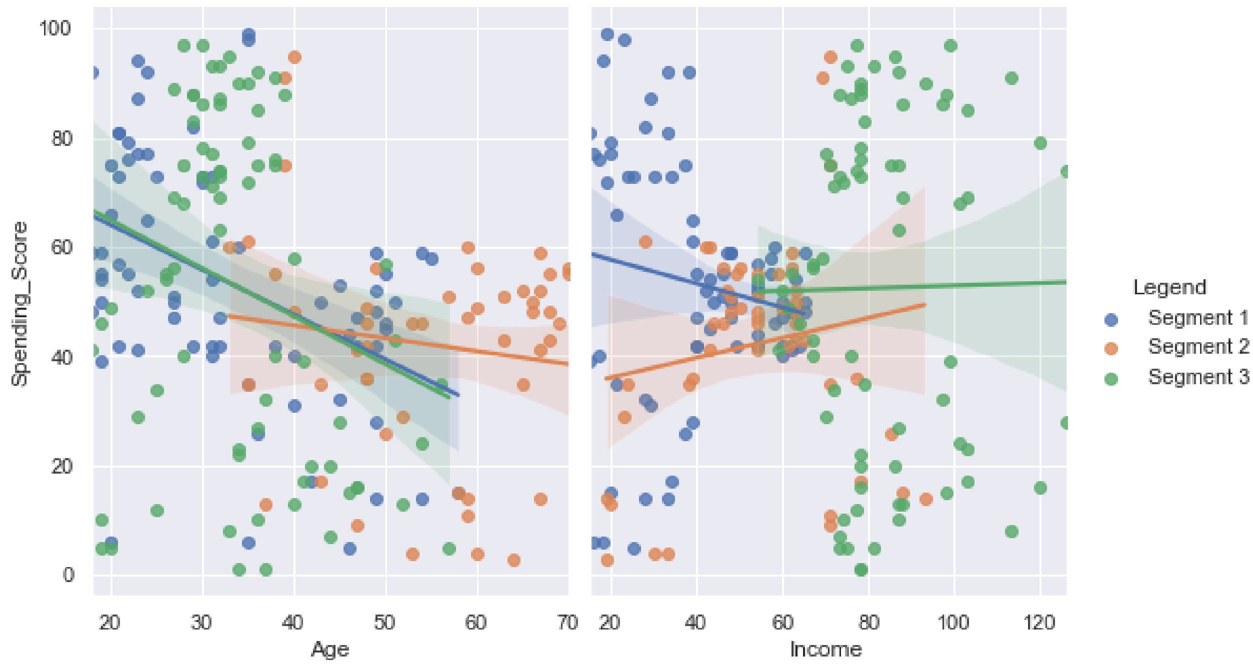
In [154…

```
sn.countplot(x="Legend", hue="generations", data=df_final_data);
```



In [155…

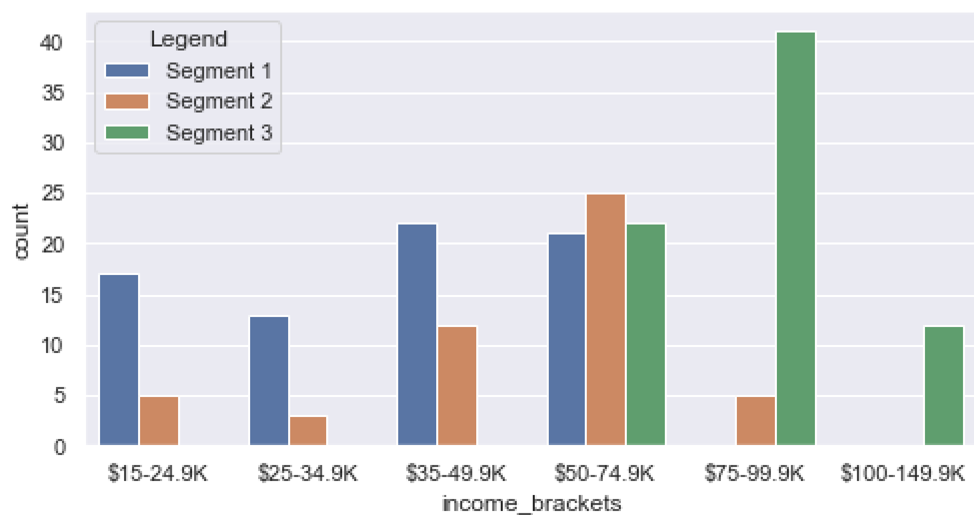
```
sn.pairplot(df_final_data, x_vars=["Age", "Income"], y_vars=["Spending_Score"],
             hue="Legend", height=5, aspect=.8, kind="reg");
```



Income Brackets

In [156…

```
sn.countplot(x="income_brackets", hue="Legend", data=df_final_data);
```



Segment 1 less affluent Income Brackets $15-74.9K

Segment 2 Income Brackets $15-99.9K skewing left

Segment 3 more affluent Income Brackets $50K+

Appendix

SQL Analysis

In [157…

```
from sqlalchemy import create_engine
import os
os.chdir('     ')
```

```
engine = create_engine('sqlite:///:memory:') #Create as table
df_final_data.to_sql('data_table',engine) #Store dataframe as table
```

In [158…

```
print(pd.read_sql_query('SELECT * FROM data_table', engine))
```

```
     index  CustomerID  Gender  Age  Income  Spending_Score  Gender_Male  \
0        0           1    Male   19      15              39            1
1        1           2    Male   21      15              81            1
2        2           3  Female   20      16               6            0
3        3           4  Female   23      16              77            0
4        4           5  Female   31      17              40            0
..     ...         ...     ...  ...     ...             ...          ...
193    193         194  Female   38     113              91            0
194    194         195  Female   47     120              16            0
195    195         196  Female   35     120              79            0
196    196         197  Female   45     126              28            0
197    197         198    Male   32     126              74            1

     Gender_Female  generations income_brackets  Component_1  Component_2  \
0                0         GenZ       $15-24.9K    -0.627008    -1.248235
1                0         GenZ       $15-24.9K    -0.532436    -1.296322
2                1         GenZ       $15-24.9K    -1.980097    -1.139643
3                1         GenZ       $15-24.9K    -1.838238    -1.211775
4                1  Millennials       $15-24.9K    -1.449168    -1.366760
..             ...          ...             ...          ...          ...
193              1  Millennials     $100-149.9K    -0.083189     2.052020
194              1         GenX     $100-149.9K     0.417853     2.097184
195              1  Millennials     $100-149.9K    -0.149580     2.385710
196              1         GenX     $100-149.9K     0.387967     2.369465
197              0  Millennials     $100-149.9K     1.184404     2.586764

     Segment_K-means_PCA     Legend
0                      0  Segment 1
1                      0  Segment 1
2                      0  Segment 1
3                      0  Segment 1
4                      0  Segment 1
..                   ...        ...
193                    2  Segment 3
194                    2  Segment 3
195                    2  Segment 3
196                    2  Segment 3
197                    2  Segment 3

[198 rows x 14 columns]
```

Total Counts

In [159…

```
#Totals
df_totals = pd.read_sql_query('SELECT (ROUND(COUNT(DISTINCT(CustomerID)),2)) AS unique_
df_totals
```

Out[159…

	unique_customers	total_age	total_income	total_spending_score
0	198.0	7708.0	11838.0	9939.0

Averages

In [160…

```
#Averages
df_avgs = pd.read_sql_query('SELECT (ROUND(COUNT(DISTINCT(CustomerID)),2)) AS unique_cu
```

```
df_avgs
```

Out[160…

	unique_customers	avg_age	avg_income	avg_spending_score
0	198.0	38.93	59.79	50.2

Generations

In [161…

```
#Generations
df_gens = pd.read_sql_query('SELECT generations, (ROUND(COUNT(DISTINCT(CustomerID)),2))
df_gens
```

Out[161…

	generations	unique_customers	avg_age	avg_income	avg_spending_score
0	Boomer	26.0	64.04	51.88	39.31
1	GenX	48.0	49.13	60.42	35.23
2	GenZ	46.0	22.13	48.50	55.63
3	Millennials	78.0	34.19	68.69	59.83

Income Brackets

In [162…

```
#Income Brackets
df_inc_bucks = pd.read_sql_query('SELECT income_brackets, (ROUND(COUNT(DISTINCT(Custome
df_inc_bucks
```

Out[162…

	income_brackets	unique_customers	avg_age	avg_income	avg_spending_score
0	$100-149.9K	12.0	37.92	111.00	48.50
1	$15-24.9K	22.0	33.32	19.27	51.23
2	$25-34.9K	16.0	37.31	30.00	46.44
3	$35-49.9K	34.0	43.32	43.35	51.24
4	$50-74.9K	68.0	40.46	62.82	49.62
5	$75-99.9K	46.0	36.93	83.83	51.54

Gender

In [163…

```
#Gender
df_gender = pd.read_sql_query('SELECT Gender, (ROUND(COUNT(DISTINCT(CustomerID)),2)) AS
df_gender
```

Out[163…

	Gender	unique_customers	avg_age	avg_income	avg_spending_score
0	Female	112.0	38.10	59.25	51.53
1	Male	86.0	40.01	60.49	48.47

Segments

In [164…

```
#Segments
df_segs = pd.read_sql_query('SELECT Legend, (ROUND(COUNT(DISTINCT(CustomerID)),2)) AS u
df_segs
```

Out[164…

	Legend	unique_customers	avg_age	avg_income	avg_spending_score
0	Segment 1	73.0	33.08	39.68	53.33
1	Segment 2	50.0	54.62	53.64	42.18
2	Segment 3	75.0	34.16	83.45	52.49

Segments by Generations

In [165…

```
#Segments by Generations
df_segs_gen = pd.read_sql_query('SELECT Legend, generations, (ROUND(COUNT(DISTINCT(Cust
df_segs_gen
```

Out[165…

	Legend	generations	unique_customers	avg_age	avg_income	avg_spending_score
0	Segment 1	Boomer	1.0	58.00	20.00	15.00
1	Segment 1	GenX	20.0	48.85	46.50	42.15
2	Segment 1	GenZ	31.0	21.71	38.42	63.00
3	Segment 1	Millennials	21.0	33.67	36.00	51.52
4	Segment 2	Boomer	25.0	64.28	53.16	40.28
5	Segment 2	GenX	16.0	49.25	57.50	35.56
6	Segment 2	Millennials	9.0	37.33	48.11	59.22
7	Segment 3	GenX	12.0	49.42	87.50	23.25
8	Segment 3	GenZ	15.0	23.00	69.33	40.40
9	Segment 3	Millennials	48.0	33.83	86.85	63.58

Segments by Income Brackets

In [166…

```
#Segments by Income Brackets
df_segs_inc_bucks = pd.read_sql_query('SELECT Legend, income_brackets, (ROUND(COUNT(DIS
df_segs_inc_bucks
```

Out[166…

	Legend	income_brackets	unique_customers	avg_age	avg_income	avg_spending_score
0	Segment 1	$15-24.9K	17.0	28.12	18.76	60.76
1	Segment 1	$25-34.9K	13.0	34.54	29.92	51.85
2	Segment 1	$35-49.9K	22.0	34.59	42.77	52.55
3	Segment 1	$50-74.9K	21.0	34.62	59.43	49.05
4	Segment 2	$15-24.9K	5.0	51.00	21.00	18.80
5	Segment 2	$25-34.9K	3.0	49.33	30.33	23.00

	Legend	income_brackets	unique_customers	avg_age	avg_income	avg_spending_score
6	Segment 2	$35-49.9K	12.0	59.33	44.42	48.83
7	Segment 2	$50-74.9K	25.0	54.32	61.28	50.08
8	Segment 2	$75-99.9K	5.0	51.60	84.20	21.60
9	Segment 3	$100-149.9K	12.0	37.92	111.00	48.50
10	Segment 3	$50-74.9K	22.0	30.27	67.82	49.64
11	Segment 3	$75-99.9K	41.0	35.15	83.78	55.20

Segments by Gender

In [167…

```
#Segment by Gender
df_segs_gender = pd.read_sql_query('SELECT Legend, Gender, (ROUND(COUNT(DISTINCT(Custom
df_segs_gender
```

Out[167…

	Legend	Gender	unique_customers	avg_age	avg_income	avg_spending_score
0	Segment 1	Female	61.0	35.25	41.57	49.89
1	Segment 1	Male	12.0	22.08	30.08	70.83
2	Segment 2	Female	8.0	64.63	52.50	48.50
3	Segment 2	Male	42.0	52.71	53.86	40.98
4	Segment 3	Female	43.0	37.21	85.58	54.42
5	Segment 3	Male	32.0	30.06	80.59	49.91

Export As CSV

In [168…

```
#Raw Data
df_final_data.to_csv('customer_segmentation_raw.csv')
```

In [169…

```
#Totals
df_totals.to_csv('customer_segmentation_totals.csv')
```

In [170…

```
#Averages
df_avgs.to_csv('customer_segmentation_avgs.csv')
```

In [171…

```
#Generations
df_gens.to_csv('customer_seg_gens.csv')
```

In [172…

```
#Income Brackets
df_inc_bucks.to_csv('customer_seg_inc_bucks.csv')
```

In [173…

```
#Gender
df_gender.to_csv('customer_seg_gender.csv')
```

In [174…

```
#Segments
df_segs.to_csv('customer_segments.csv')
```

In [175…

```
#Segments by Generations
df_segs_gen.to_csv('customer_segments_gen.csv')
```

In [176…

```
#Segments by Income Brackets
df_segs_inc_bucks.to_csv('customer_segments_income.csv')
```

In [177…

```
#Segmetns by Gender
df_segs_gender.to_csv('customers_segmentation_gender.csv')
```

In [ ]:

In [ ]: