Course No. : DSECL ZC556

Course Title : Stream Processing and Analytics

Assignment : 2

Group No : 231

Group Member Names:

1.AMRITESH KUMAR DAS (2021sc04432)

2.ASHIQUE ZZAMAN (2021sc04612)

3.BISHNU CHARAN SINHA (2021sc04431)

4.RAKSHANDA KAUL (2021sc04406)

In [28]:
```
#Import libraaries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly as py
import plotly.graph_objs as go
from sklearn.cluster import KMeans
import warnings
import os
warnings.filterwarnings("ignore")
```

In [29]:
```
# Read data from the CSV file 'pizza_customers.csv' into a DataFrame 'pizza_df'
pizza_df = pd.read_csv('pizza_customers.csv')
```

In [30]:
```
# Display the first few rows of the DataFrame 'pizza_df' to inspect the data
pizza_df.head()
```

Out[30]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
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

In [31]:
```
# Get the shape (number of rows and columns) of the DataFrame 'pizza_df'
pizza_df.shape
```

Out[31]:
```
(200, 5)
```

In [32]:
```
# Generate descriptive statistics for the DataFrame 'pizza_df'
pizza_df.describe()
```

Out[32]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
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

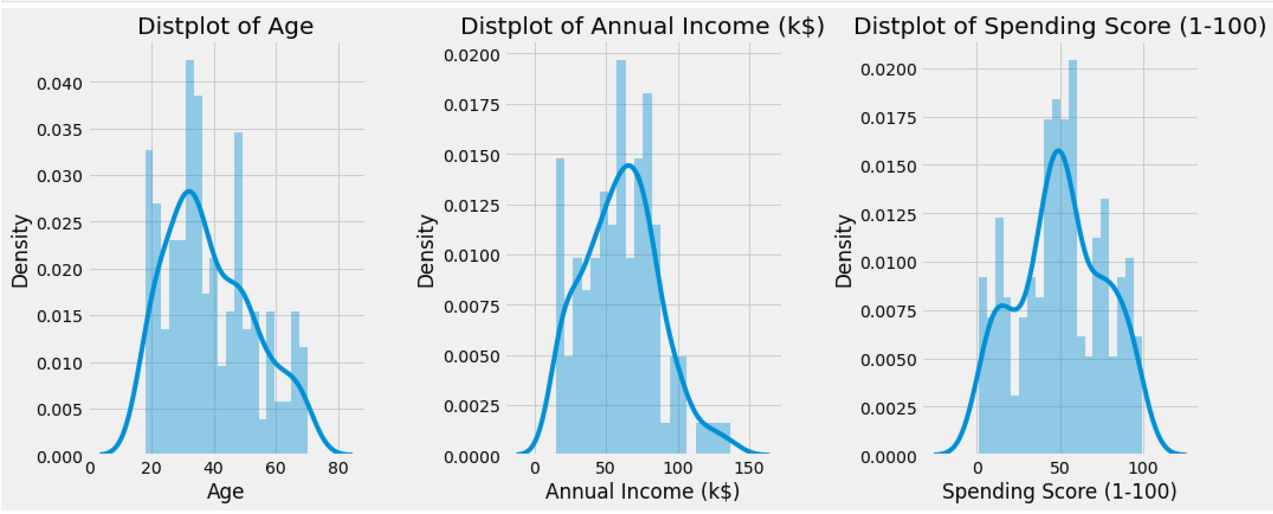
In [33]:
```
# Get the data types of columns in the DataFrame 'pizza_df'
pizza_df.dtypes
```

Out[33]:
```
CustomerID                 int64
Gender                    object
Age                        int64
Annual Income (k$)         int64
Spending Score (1-100)     int64
dtype: object
```

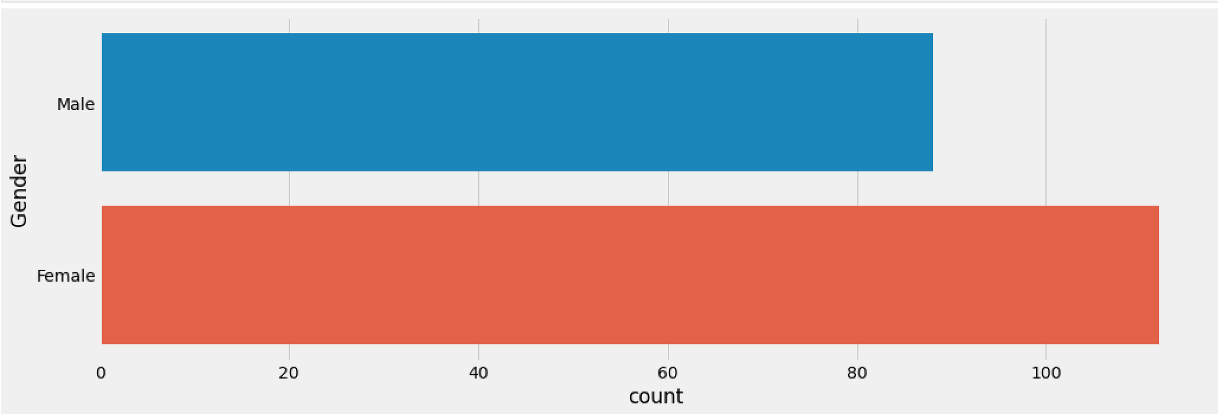
In [34]:
```
# Count the number of missing (null) values in each column of the DataFrame 'pizza_df'
pizza_df.isnull().sum()
```

Out[34]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

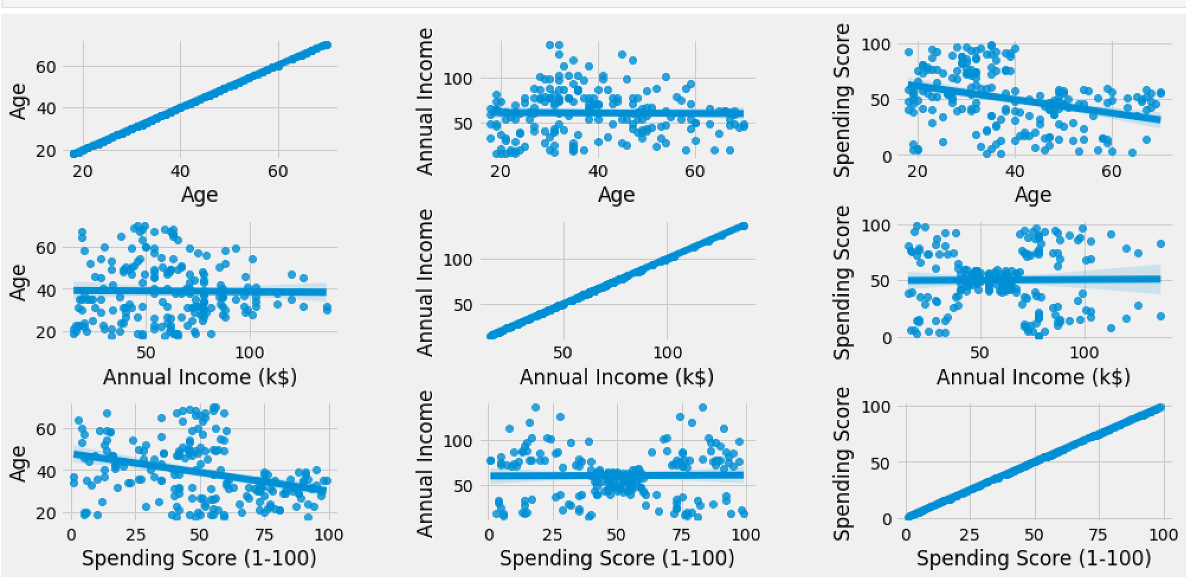
In [35]:
```
# Set the plotting style to 'fivethirtyeight' for a specific visual style
plt.style.use('fivethirtyeight')
# Create a figure with one row and three columns, specifying the figure size
plt.figure(1 , figsize = (15 , 6))
# Initialize a variable 'n' to keep track of the subplot number
n = 0
# Iterate through the list of column names: 'Age', 'Annual Income (k$)', and 'Spending Score (1-100)'
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1 # Increment the subplot number
    plt.subplot(1 , 3 , n) # Create a subplot within the figure
    plt.subplots_adjust(hspace =0.5 , wspace = 0.5) # Adjust spacing between subplots
    sns.distplot(pizza_df[x] , bins = 20) # Create a distribution plot for the current column
    plt.title('Distplot of {}'.format(x)) # Set the title of the subplot
# Display the figure with the subplots
plt.show()
```



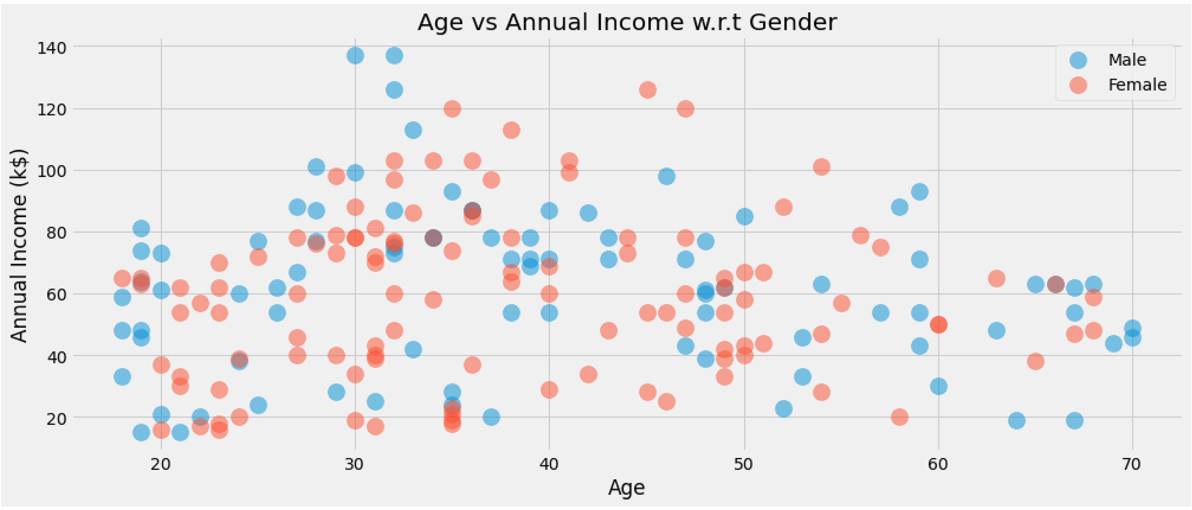
In [36]:
```
# Create a figure with one row and a specified figure size
plt.figure(1 , figsize = (15 , 5))
# Create a countplot using Seaborn to visualize the distribution of 'Gender' from the DataFrame 'pizza_df'
sns.countplot(y = 'Gender' , data = pizza_df)
# Display the plot
plt.show()
```



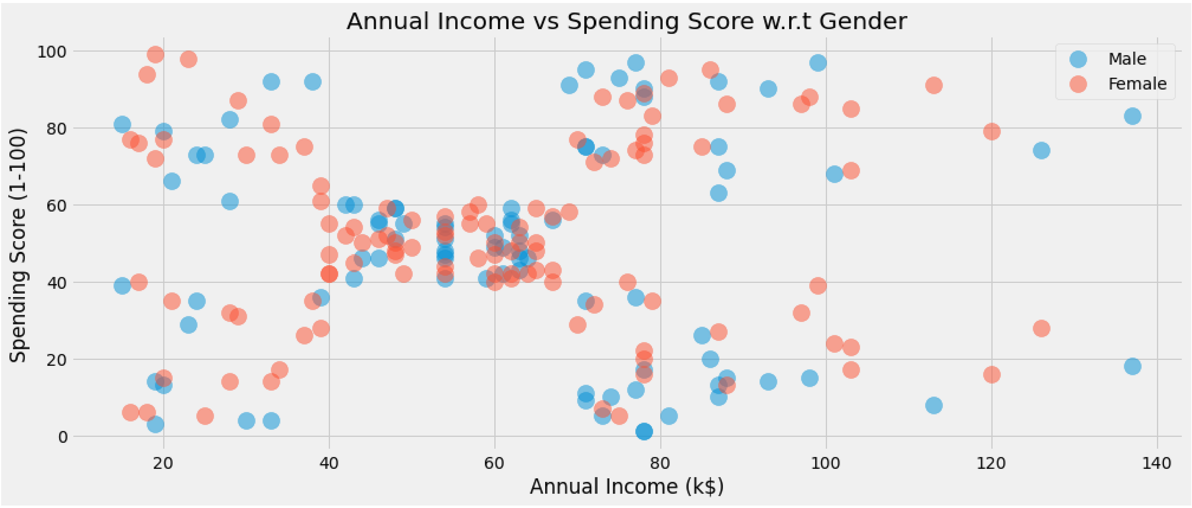
In [37]:
```
# Create a figure with one row and a specified figure size
plt.figure(1, figsize=(15, 7))

# Initialize a variable 'n' to keep track of the subplot number
n = 0

# Iterate through the list of column names for both x and y axes
for x in ['Age', 'Annual Income (k$)', 'Spending Score (1-100)']:
    for y in ['Age', 'Annual Income (k$)', 'Spending Score (1-100)']:
        n += 1  # Increment the subplot number
        plt.subplot(3, 3, n)  # Create a subplot within the figure
        plt.subplots_adjust(hspace=0.5, wspace=0.5)  # Adjust spacing between subplots

        # Create a regression plot (scatter plot with regression line) for the current x and y columns
        sns.regplot(x=x, y=y, data=pizza_df)

        # Set the y-axis label based on the column name, splitting if necessary
        plt.ylabel(y.split()[0] + ' ' + y.split()[1] if len(y.split()) > 1 else y)

# Display the figure with the subplots
plt.show()
```



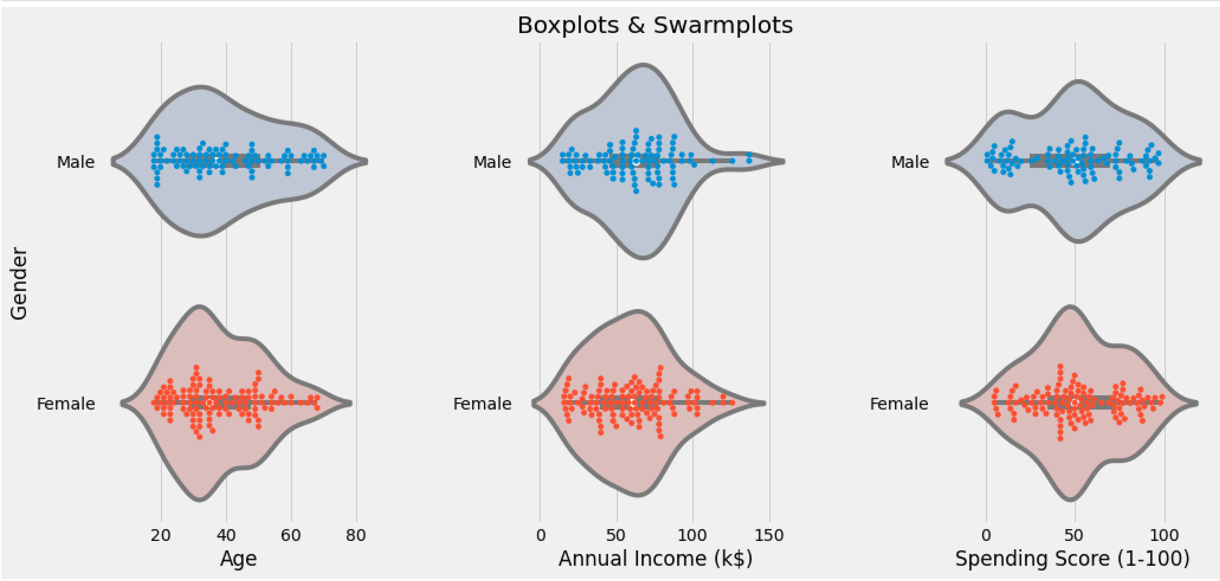
In [38]:
```
# Create a figure with a specified figure size
plt.figure(1, figsize=(15, 6))

# Iterate through the two gender categories: 'Male' and 'Female'
for gender in ['Male', 'Female']:
    # Create a scatter plot for 'Age' vs. 'Annual Income (k$)' based on the gender category
    plt.scatter(x='Age', y='Annual Income (k$)', data=pizza_df[pizza_df['Gender'] == gender],
                s=200, alpha=0.5, label=gender)

# Set the x and y-axis labels and title for the plot
plt.xlabel('Age')
plt.ylabel('Annual Income (k$)')
plt.title('Age vs Annual Income w.r.t Gender')

# Display a legend to distinguish between 'Male' and 'Female' data points
plt.legend()

# Show the plot
plt.show()
```



In [39]:
```python
# Create a figure with a specified figure size
plt.figure(1, figsize=(15, 6))

# Iterate through the two gender categories: 'Male' and 'Female'
for gender in ['Male', 'Female']:
    # Create a scatter plot for 'Annual Income (k$)' vs. 'Spending Score (1-100)' based on the gender category
    plt.scatter(x='Annual Income (k$)', y='Spending Score (1-100)',
                data=pizza_df[pizza_df['Gender'] == gender], s=200, alpha=0.5, label=gender)

# Set the x and y-axis labels and title for the plot
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Annual Income vs Spending Score w.r.t Gender')

# Display a legend to distinguish between 'Male' and 'Female' data points
plt.legend()

# Show the plot
plt.show()
```



In [40]:
```python
# Create a figure with one row and a specified figure size
plt.figure(1, figsize=(15, 7))

# Initialize a variable 'n' to keep track of the subplot number
n = 0

# Iterate through the columns 'Age', 'Annual Income (k$)', and 'Spending Score (1-100)'
for cols in ['Age', 'Annual Income (k$)', 'Spending Score (1-100)']:
    n += 1  # Increment the subplot number
    plt.subplot(1, 3, n)  # Create a subplot within the figure
    plt.subplots_adjust(hspace=0.5, wspace=0.5)  # Adjust spacing between subplots

    # Create a violin plot for the current column 'cols' with 'Gender' on the y-axis
    sns.violinplot(x=cols, y='Gender', data=pizza_df, palette='vlag')

    # Create a swarm plot to show individual data points for the current column 'cols' and 'Gender'
    sns.swarmplot(x=cols, y='Gender', data=pizza_df)

    # Set the y-axis label for the first subplot
    plt.ylabel('Gender' if n == 1 else '')

    # Set the title for the second subplot
    plt.title('Boxplots & Swarmplots' if n == 2 else '')

# Show the plot
plt.show()
```



Clustering using K- means

1.Segmentation using Age and Spending Score

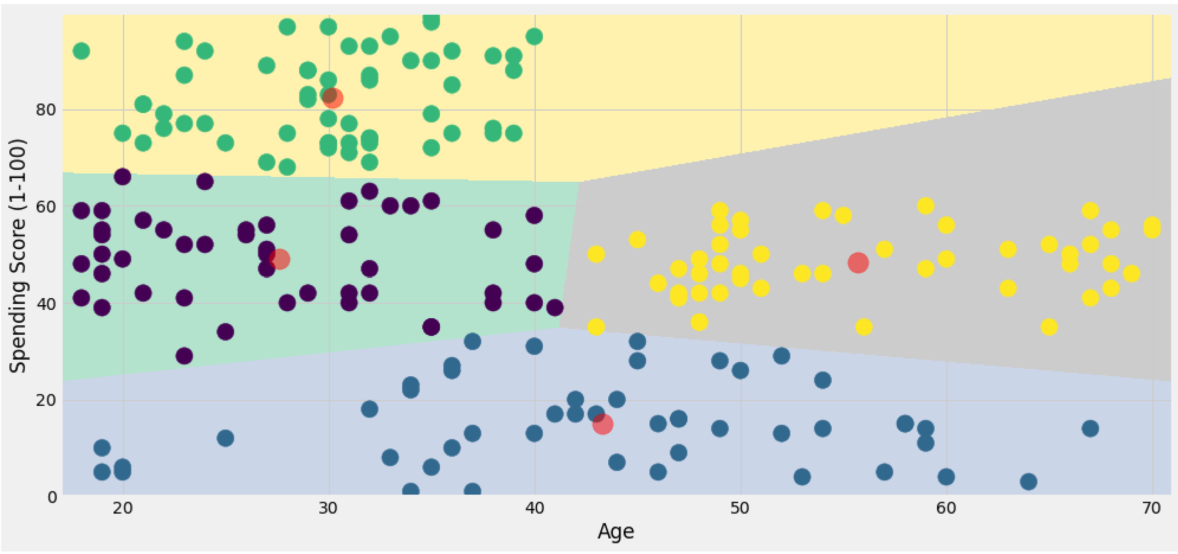
In [41]:
```python
'''Age and spending Score'''
# Create a data matrix 'X1' containing 'Age' and 'Spending Score (1-100)' columns from the DataFrame
X1 = pizza_df[['Age' , 'Spending Score (1-100)']].iloc[: , :].values
# Initialize an empty list 'inertia' to store inertia values for different values of k
inertia = []
# Iterate through values of k from 1 to 10
for n in range(1 , 11):
    # Instantiate a k-means clustering algorithm with specific parameters
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    # Fit the algorithm to the data matrix 'X1'
    algorithm.fit(X1)
    # Calculate and append the inertia (within-cluster sum of squares) for the current k
    inertia.append(algorithm.inertia_)
```

Selecting N Clusters based in Inertia (Squared Distance between Centroids and data points, should be less)

In [42]:
```python
# Create a figure with a specified figure size
plt.figure(1, figsize=(15, 6))

# Create a scatter plot of inertia values against the number of clusters (k)
plt.plot(np.arange(1, 11), inertia, 'o', label='Inertia values')
plt.plot(np.arange(1, 11), inertia, '-', alpha=0.5)

# Set the x-axis label as 'Number of Clusters' and the y-axis label as 'Inertia'
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')

# Show the plot
plt.show()
```

In [43]:
```python
# Initialize a K-means clustering algorithm with specific parameters:
# - n_clusters is set to 4 for the desired number of clusters.
# - init is set to 'k-means++' for smart initialization.
# - n_init is set to 10 for the number of times the algorithm is reinitialized.
# - max_iter specifies the maximum number of iterations for each initialization.
# - tol sets the tolerance for convergence.
# - random_state is set to 111 for reproducibility.
# - algorithm is set to 'elkan', an optimized variant of K-means.
algorithm = KMeans(n_clusters=4, init='k-means++', n_init=10, max_iter=300,
                   tol=0.0001, random_state=111, algorithm='elkan')

# Fit the K-means algorithm to the data matrix X1, performing clustering.
algorithm.fit(X1)

# Extract cluster labels (labels1) for each data point in X1.
cluster1 = algorithm.labels_

# Determine cluster centroids (centroids1) for the clusters found by the algorithm.
centroids1 = algorithm.cluster_centers_

# Define a step size (h) for creating a mesh grid.
h = 0.02

# Define the minimum and maximum values for x and y dimensions.
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1

# Create a mesh grid (xx and yy) for contour plotting.
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict cluster assignments (Z) for each point in the grid using the trained K-means model.
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

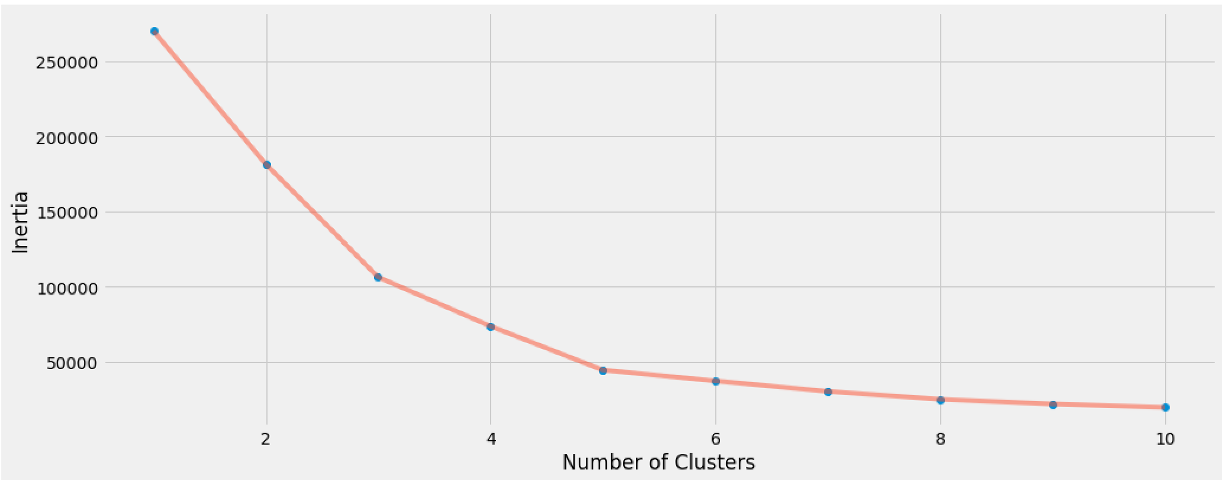
In [44]:
```python
plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'Age' ,y = 'Spending Score (1-100)' , data = pizza_df , c = cluster1 ,
            s = 200 )
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c = 'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age')
plt.show()
```



2. Segmentation using Annual Income and Spending Score

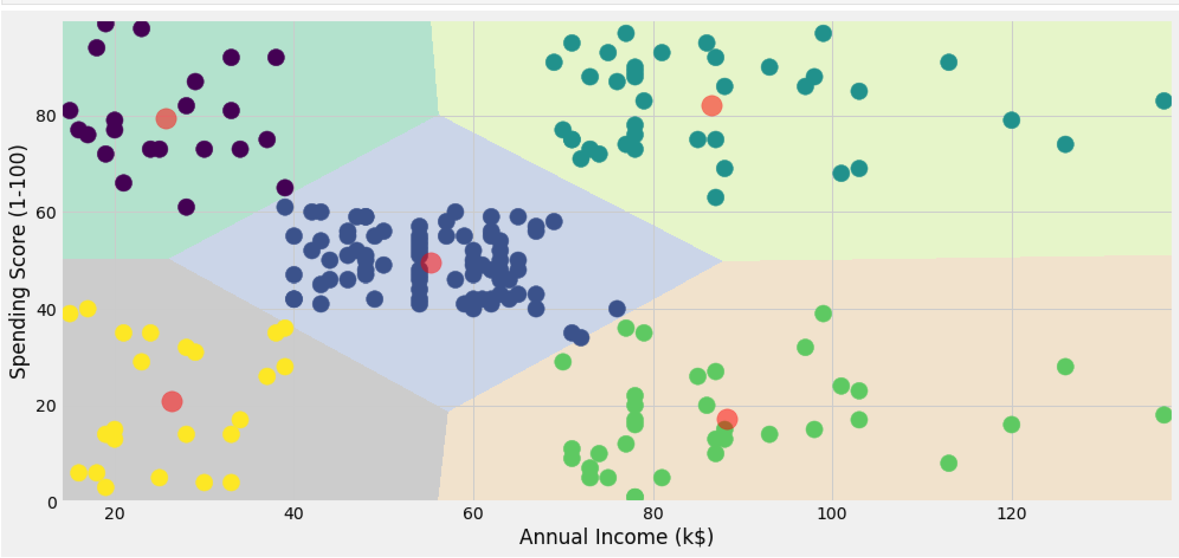
In [45]:
```
'''Annual Income and spending Score'''
# Create a data matrix 'X2' containing 'Annual Income (k$)' and 'Spending Score (1-100)' columns from the DataFrame
X2 = pizza_df[['Annual Income (k$)', 'Spending Score (1-100)']].iloc[:, :].values

# Initialize an empty list 'inertia' to store inertia values for different values of k
inertia = []

# Iterate through values of k from 1 to 10
for n in range(1, 11):
    # Initialize a K-means clustering algorithm with specific parameters
    algorithm = KMeans(n_clusters=n, init='k-means++', n_init=10, max_iter=300,
                       tol=0.0001, random_state=111, algorithm='elkan')

    # Fit the K-means algorithm to the data matrix 'X2'
    algorithm.fit(X2)

    # Calculate and append the inertia (within-cluster sum of squares) for the current k
    inertia.append(algorithm.inertia_)
```

In [46]:
```
# Create a figure with a specified figure size
plt.figure(1, figsize=(15, 6))

# Create a line plot of inertia values against the number of clusters (k)
plt.plot(np.arange(1, 11), inertia, 'o', label='Inertia values')
plt.plot(np.arange(1, 11), inertia, '-', alpha=0.5)

# Set the x-axis label as 'Number of Clusters' and the y-axis label as 'Inertia'
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')

# Show the plot
plt.show()
```



In [47]:
```
# Initialize a K-means clustering algorithm with specific parameters:
# - n_clusters is set to 5 for the desired number of clusters.
# - init is set to 'k-means++' for smart initialization.
# - n_init is set to 10 for the number of times the algorithm is reinitialized.
# - max_iter specifies the maximum number of iterations for each initialization.
# - tol sets the tolerance for convergence.
# - random_state is set to 111 for reproducibility.
# - algorithm is set to 'elkan', an optimized variant of K-means.
algorithm = KMeans(n_clusters=5, init='k-means++', n_init=10, max_iter=300,
                   tol=0.0001, random_state=111, algorithm='elkan')

# Fit the K-means algorithm to the data matrix 'X2', performing clustering.
algorithm.fit(X2)

# Extract cluster labels (cluster2) for each data point in X2.
cluster2 = algorithm.labels_

# Determine cluster centroids (centroids2) for the clusters found by the algorithm.
centroids2 = algorithm.cluster_centers_

# Define a step size (h) for creating a mesh grid.
h = 0.02

# Define the minimum and maximum values for x and y dimensions.
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1

# Create a mesh grid (xx and yy) for contour plotting.
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Predict cluster assignments (Z2) for each point in the grid using the trained K-means model.
Z2 = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

In [48]:
```
# Create a new figure with a specified size
plt.figure(1, figsize=(15, 7))

# Clear the current figure
plt.clf()

# Reshape the Z2 values to match the shape of the mesh grid (xx)
Z2 = Z2.reshape(xx.shape)

# Display the clustering results using an image plot
plt.imshow(Z2, interpolation='nearest', extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap=plt.cm.Pastel2, aspect='auto', origin='lower')

# Scatter plot the data points based on 'Annual Income (k$)' and 'Spending Score (1-100)'
plt.scatter(x='Annual Income (k$)', y='Spending Score (1-100)', data=pizza_df, c=cluster2, s=200)

# Scatter plot the cluster centroids in red
plt.scatter(x=centroids2[:, 0], y=centroids2[:, 1], s=300, c='red', alpha=0.5)

# Set the y-axis label as 'Spending Score (1-100)' and the x-axis label as 'Annual Income (k$)'
plt.ylabel('Spending Score (1-100)')
plt.xlabel('Annual Income (k$)')

# Show the plot
plt.show()
```

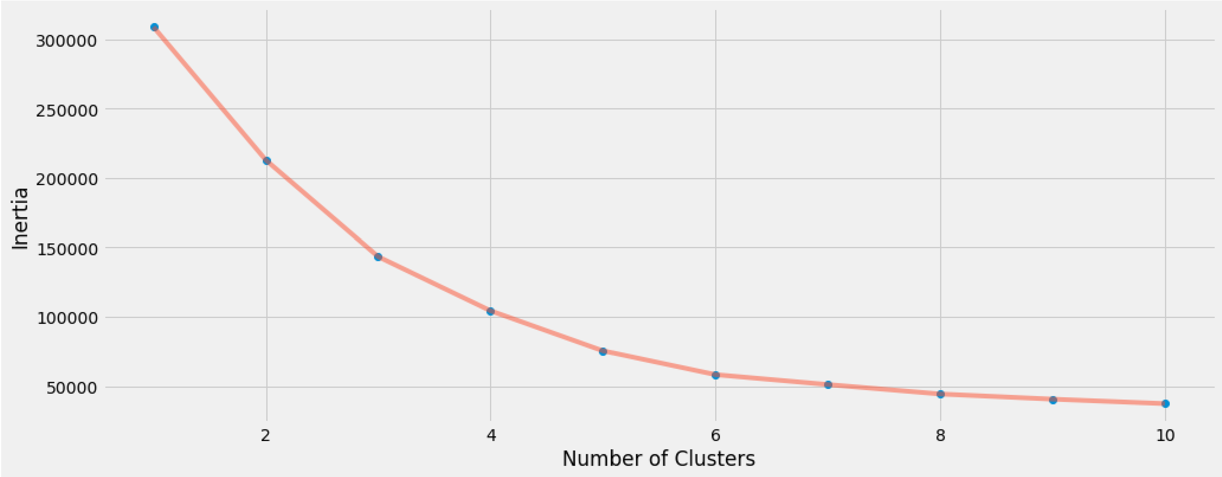


3.Segmentation using Age , Annual Income and Spending Score

In [49]:
```
# Create a data matrix 'X3' containing 'Age,' 'Annual Income (k$),' and 'Spending Score (1-100)' columns from the DataFrame
X3 = pizza_df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].iloc[:, :].values

# Initialize an empty list 'inertia' to store inertia values for different values of k
inertia = []

# Iterate through values of k from 1 to 10
for n in range(1, 11):
    # Initialize a K-means clustering algorithm with specific parameters
    algorithm = KMeans(n_clusters=n, init='k-means++', n_init=10, max_iter=300,
                       tol=0.0001, random_state=111, algorithm='elkan')

    # Fit the K-means algorithm to the data matrix 'X3'
    algorithm.fit(X3)

    # Calculate and append the inertia (within-cluster sum of squares) for the current k
    inertia.append(algorithm.inertia_)
```

In [50]:
```
plt.figure(1 , figsize = (15 ,6))
plt.plot(np.arange(1 , 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```

In [51]:
```
# Create a figure for the plot with a specified size (15 units wide and 6 units tall)
plt.figure(1, figsize=(15, 6))

# Create a line plot of inertia values against the number of clusters (k)
plt.plot(np.arange(1, 11), inertia, 'o', label='Inertia values')  # 'o' markers for data points
plt.plot(np.arange(1, 11), inertia, '-', alpha=0.5)  # Solid line with reduced opacity

# Set the x-axis label as 'Number of Clusters' and the y-axis label as 'Inertia'
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')

# Show the plot
plt.show()
```



In [52]:

```
# Initialize a K-means clustering algorithm with specific parameters:
# - n_clusters is set to 6 for the desired number of clusters.
# - init is set to 'k-means++' for smart initialization.
# - n_init is set to 10 for the number of times the algorithm is reinitialized.
# - max_iter specifies the maximum number of iterations for each initialization.
# - tol sets the tolerance for convergence.
# - random_state is set to 111 for reproducibility.
# - algorithm is set to 'elkan', an optimized variant of K-means.
algorithm = KMeans(n_clusters=6, init='k-means++', n_init=10, max_iter=300,
                   tol=0.0001, random_state=111, algorithm='elkan')

# Fit the K-means algorithm to the data matrix 'X3'
algorithm.fit(X3)

# Extract cluster labels (labels3) for each data point in X3.
cluster3 = algorithm.labels_

# Determine cluster centroids (centroids3) for the clusters found by the algorithm.
centroids3 = algorithm.cluster_centers_
```

In [53]:

```
pizza_df['cluster3'] =  cluster3
trace1 = go.Scatter3d(
    x= pizza_df['Age'],
    y= pizza_df['Spending Score (1-100)'],
    z= pizza_df['Annual Income (k$)'],
    mode='markers',
     marker=dict(
        color = pizza_df['cluster3'],
        size= 20,
        line=dict(
            color= pizza_df['cluster3'],
            width= 12
        ),
        opacity=0.8
     )
)
data = [trace1]
layout = go.Layout(
#     margin=dict(
#         l=0,
#         r=0,
#         b=0,
#         t=0
#     )
    title= 'Clusters',
    scene = dict(
            xaxis = dict(title  = 'Age'),
            yaxis = dict(title  = 'Spending Score'),
            zaxis = dict(title  = 'Annual Income')
        )
)
fig = go.Figure(data=data, layout=layout)
py.offline.iplot(fig)
```

In [57]:

```
# Save the DataFrame 'pizza_df' to a CSV file
pizza_df.to_csv('pizzacustomer_segmented.csv', sep=',', encoding='utf-8', header=True, index=False)
```

In [65]: