CUSTOMER SEGMENTATION USING DATA SCIENCE

PHASE 4 SUBMISSION DOCUMENT

PROJECT TITLE: Product Demand Prediction

Phase 4: *development part2*

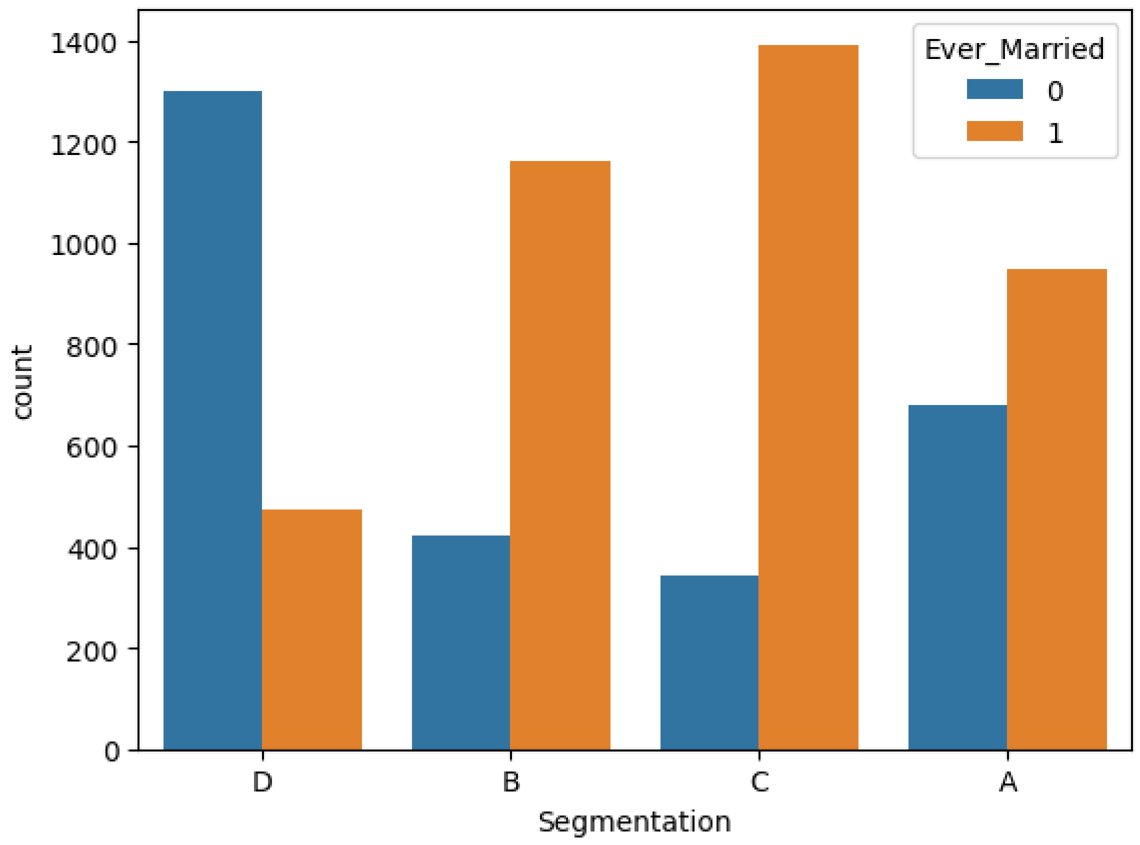
Topic: continue building the customer segmentation model by feature engineering,applying clustering algorithms,visualization,interpretation

FEATURE ENGINEERING

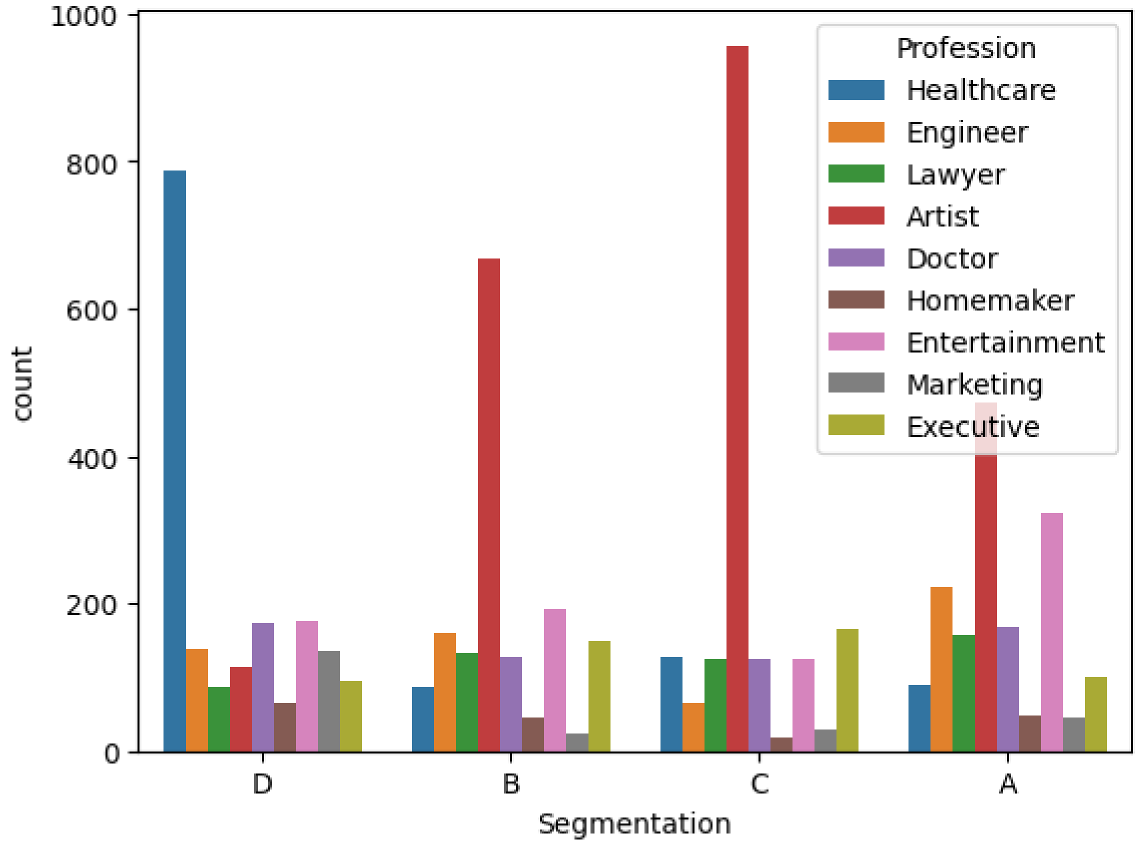
As the normal data cleaning procedure goes. Finding null values deleting them. Two columns I thought were unnecessary, The 'ID' and 'Var_1' columns therefore I dropped them.

- Made separate lists for categorical and numerical variables. Utilised them by mapping the binary cateogrical variables as 0 and 1.
- There were two columns with non-binary categorical variables. therefore created dummies off those.
- For numerical variables utilised MinMaxScaling so as to bring them under same scale.
- Mapped the target variables (A, B, C, D) as (1, 2, 3, 4) for better model fitting and performance.

Some Visualizations :

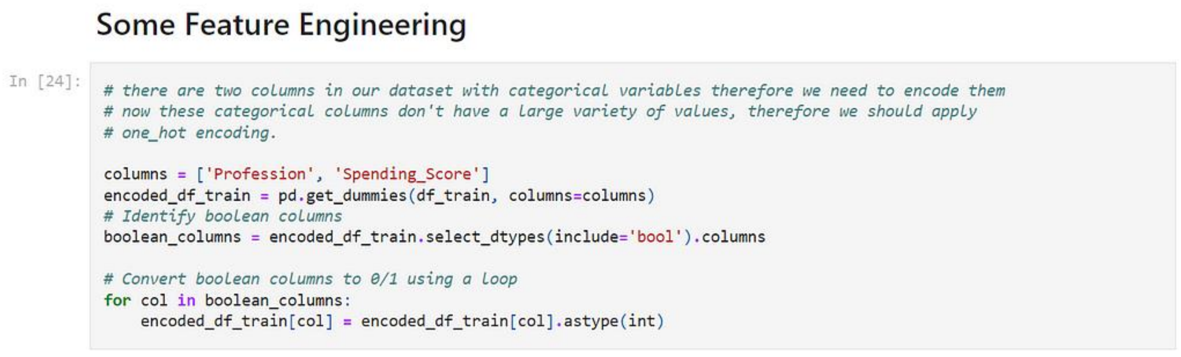


count
1400
1200
1000
800
600
400
200
0
D
B
C
A
Segmentation
Ever_Married
0
1



count
1000
800
600
400
200
0
D
B
C
A
Segmentation
Profession
Healthcare
Engineer
Lawyer
Artist
Doctor
Homemaker
Entertainment
Marketing
Executive

Quick glance at feature Engineering :

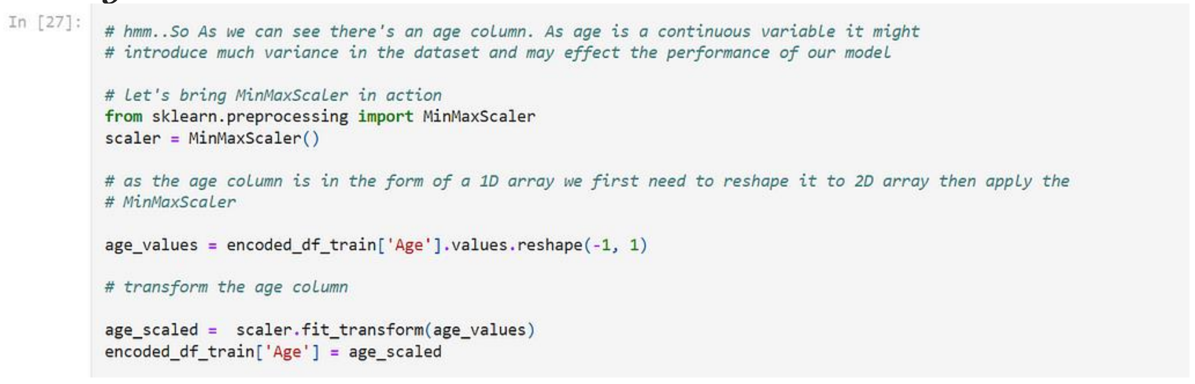


Some Feature Engineering

In [24]:

```python
# there are two columns in our dataset with categorical variables therefore we need to encode them
# now these categorical columns don't have a large variety of values, therefore we should apply
# one_hot encoding.

columns = ['Profession', 'Spending_Score']
encoded_df_train = pd.get_dummies(df_train, columns=columns)
# Identify boolean columns
boolean_columns = encoded_df_train.select_dtypes(include='bool').columns

# Convert boolean columns to 0/1 using a loop
for col in boolean_columns:
    encoded_df_train[col] = encoded_df_train[col].astype(int)
```

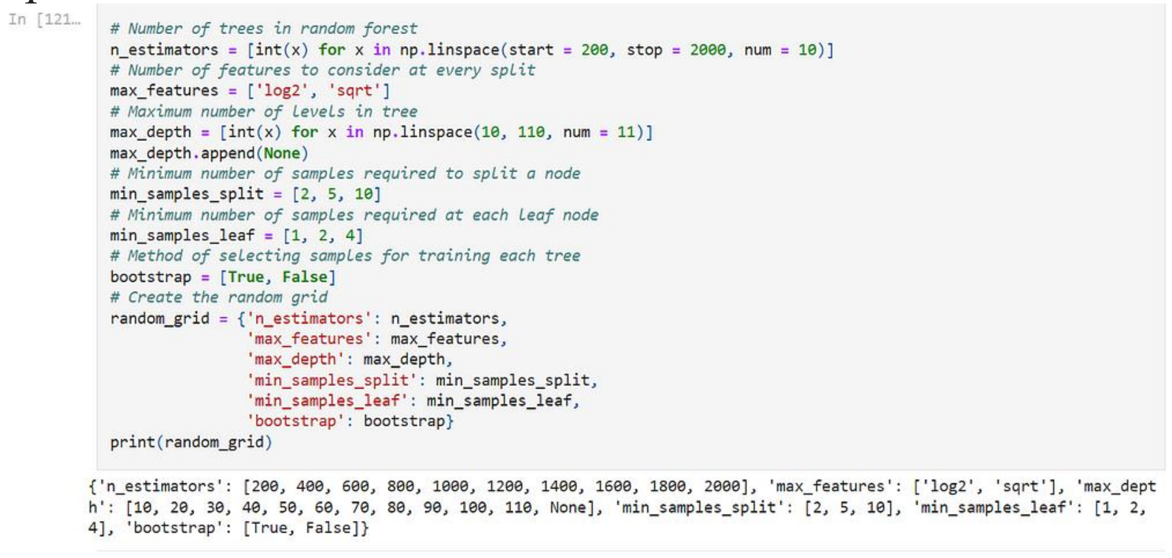
Scaling The Numerical Variables :



In [27]:

```python
# hmm..So As we can see there's an age column. As age is a continuous variable it might
# introduce much variance in the dataset and may effect the performance of our model

# let's bring MinMaxScaler in action
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

# as the age column is in the form of a 1D array we first need to reshape it to 2D array then apply the
# MinMaxScaler

age_values = encoded_df_train['Age'].values.reshape(-1, 1)

# transform the age column

age_scaled =  scaler.fit_transform(age_values)
encoded_df_train['Age'] = age_scaled
```

Preparation to utilise RandomGridSearchCV :



In [121…

```python
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['log2', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['log2', 'sqrt'], 'max_dept
h': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2,
4], 'bootstrap': [True, False]}
```

Data Exploration

Data Visualization

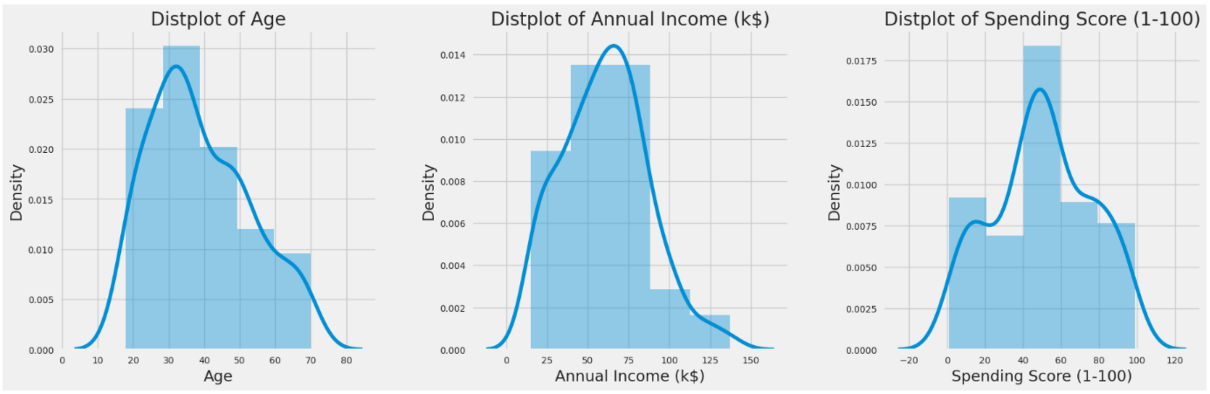
In [7]:

```
plt.style.use('fivethirtyeight')
```

Histograms

In [8]:

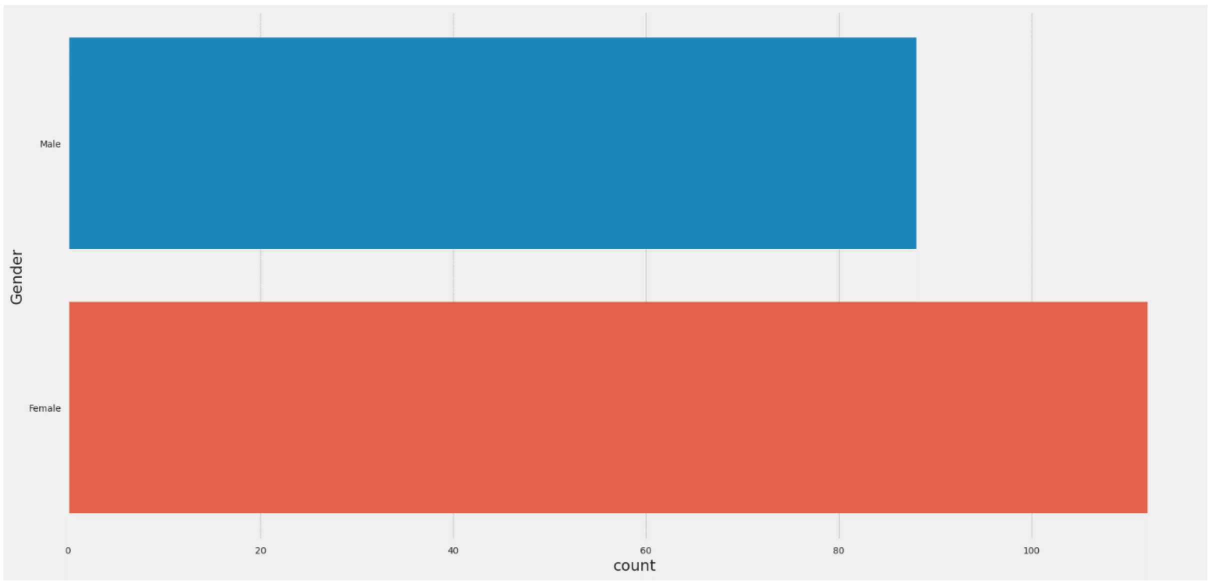
```
plt.figure(1 , figsize = (20 , 6))
n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1
    plt.subplot(1 , 3 , n)
    plt.subplots_adjust(hspace =0.3 , wspace = 0.3)
    sns.distplot(df[x] , bins = 5)
    plt.title('Distplot of {}'.format(x))
plt.show()
```



Count Plot of Gender

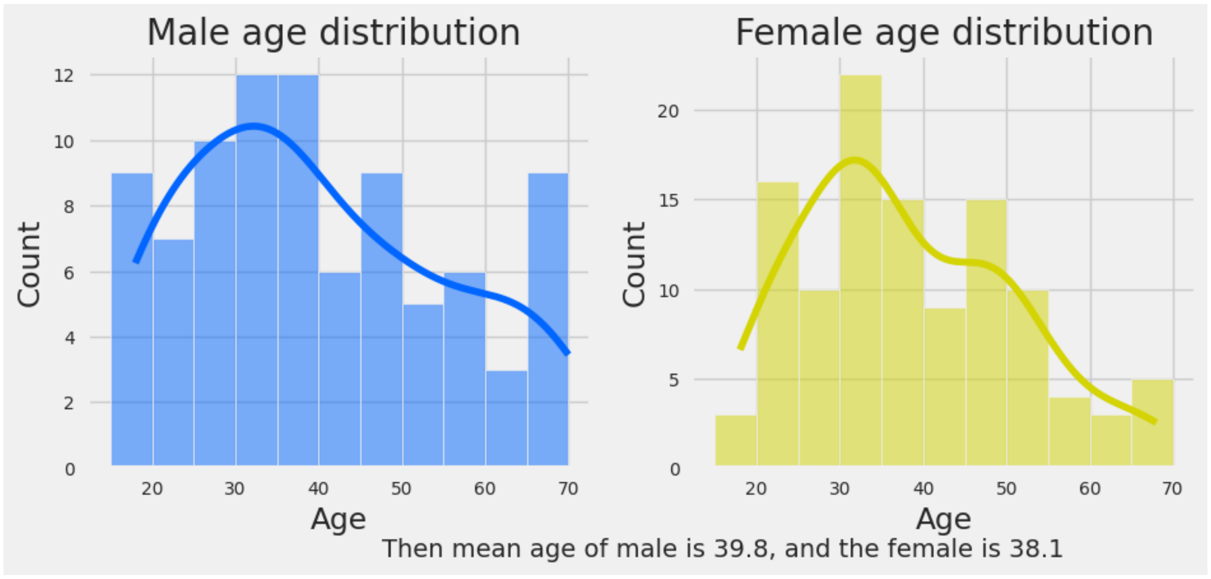
In [9]:

```
plt.figure(1 , figsize = (20 , 10))
sns.countplot(y = 'Gender' , data = df)
plt.show()
```



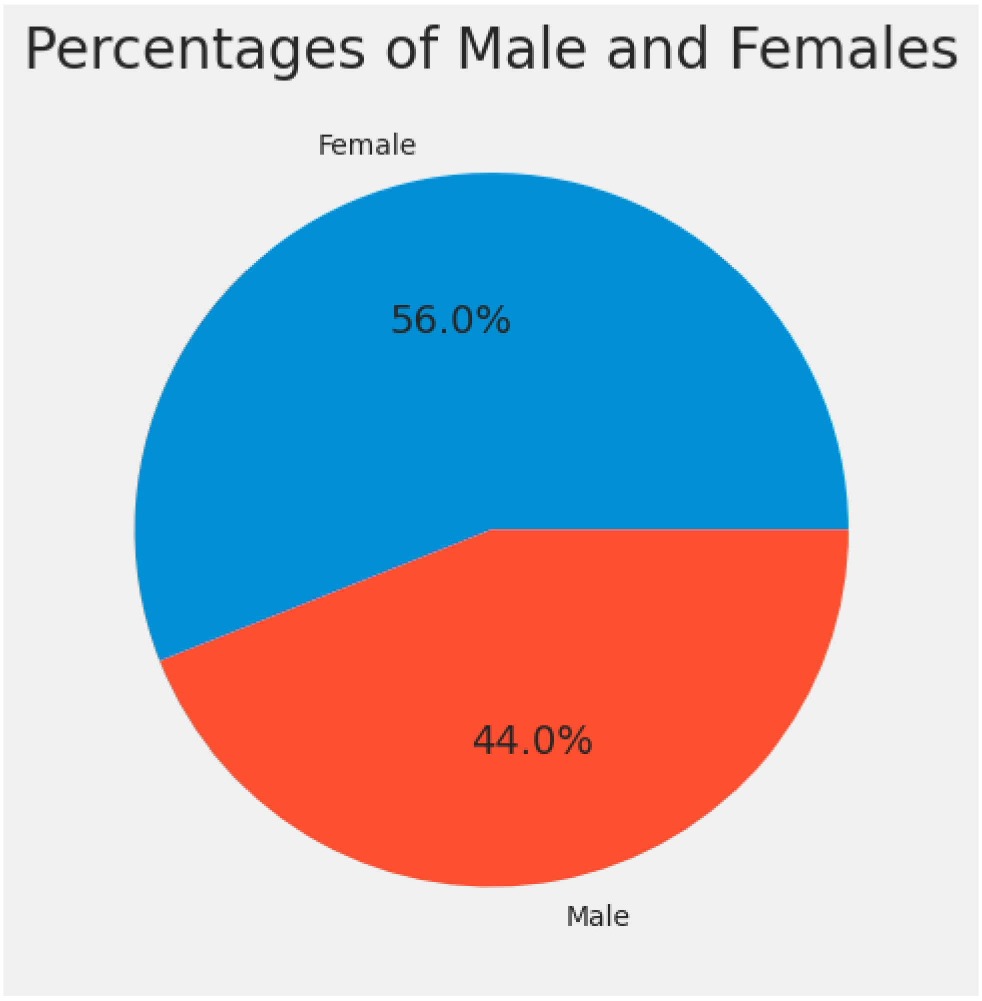
In [10]:

```
# Distribution of age with respect to gender
male = df[df.Gender == "Male"]["Age"]
female = df[df.Gender != "Male"]['Age']

plt.figure(figsize = (10,4))
plt.subplot(1,2,1)
sns.histplot(male, color='#0066ff', bins = range(15,75,5), kde = True)
plt.title("Male age distribution ")

plt.subplot(1,2,2)
sns.histplot(female, color = '#D4D404', bins = range(15,75,5), kde = True)
plt.title("Female age distribution");
plt.text(-25,-5,f"Then mean age of male is {round(male.mean(),1)}, and the
female is {round(female.mean(),1)}")

plt.show()
```



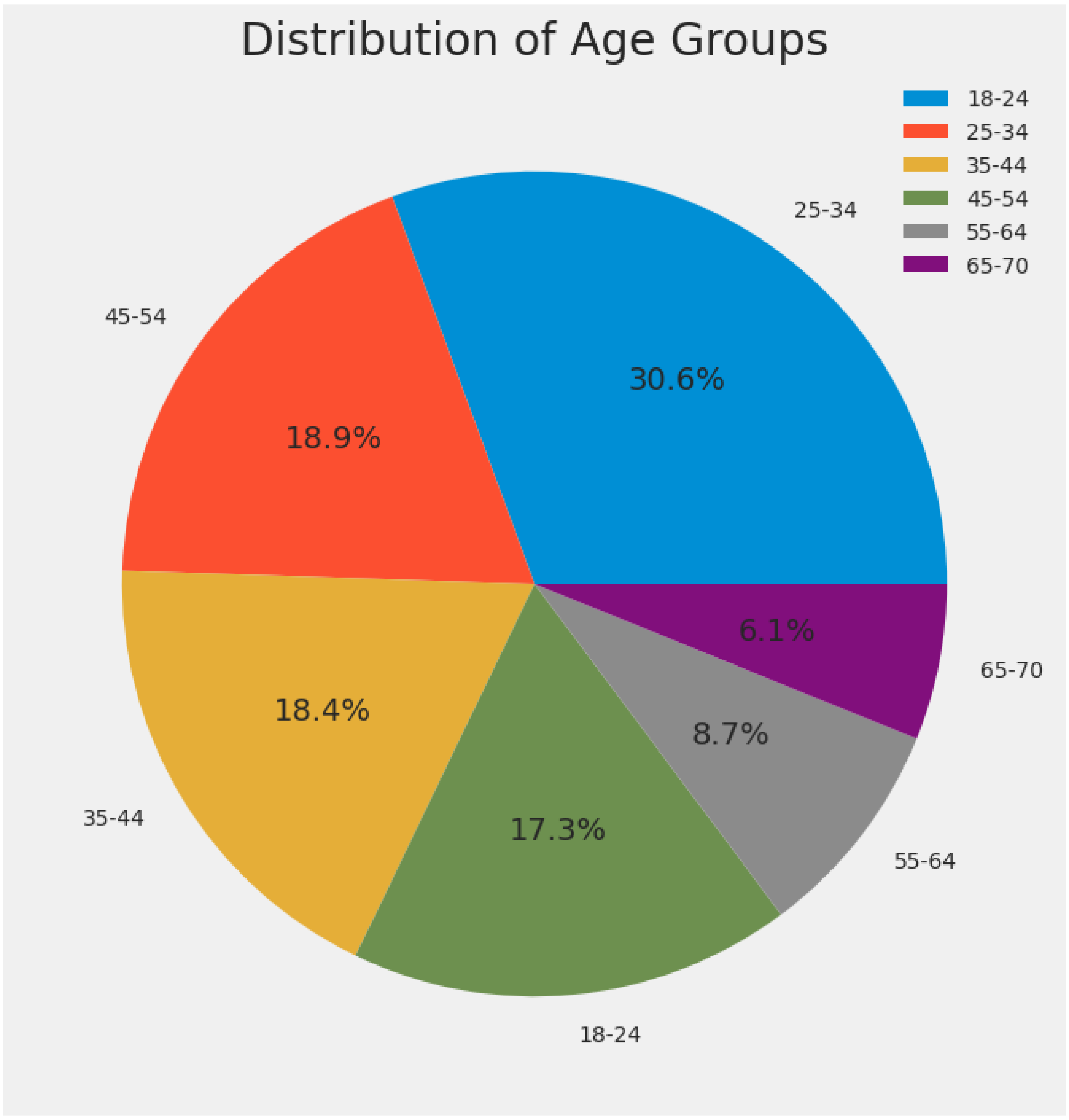
In [11]:

```
plt.pie(df.Gender.value_counts(), labels = ['Female', 'Male'], autopct ="%.
01f%%")
plt.title('Percentages of Male and Females' );
```



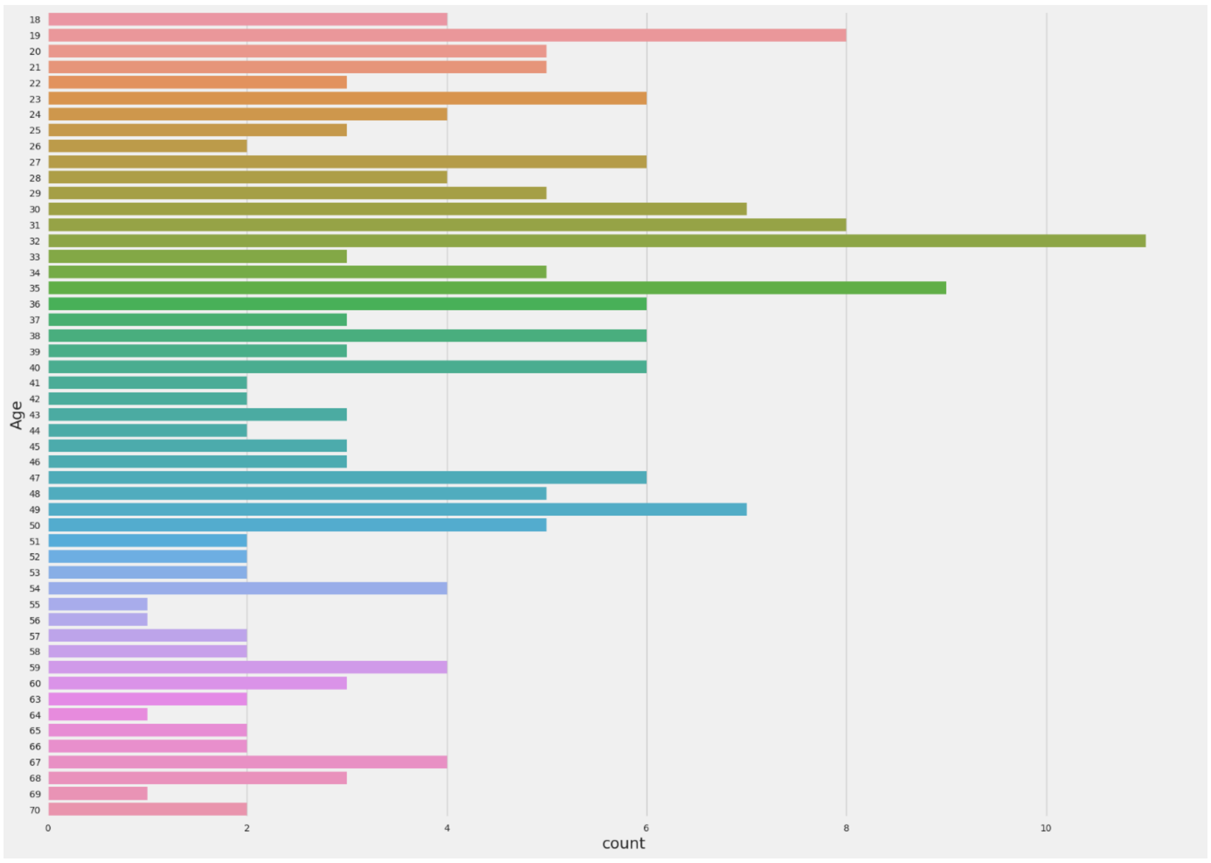
In [12]:

```
# Spending Score by age group

df['Age_Group'] = pd.cut(df.Age, bins = [18, 25, 35, 45, 55, 65, 70], label
s = ['18-24', '25-34', '35-44', '45-54', '55-64', '65-70'])

plt.figure(figsize = (8, 8))
plt.pie(df.Age_Group.value_counts(), labels = df.Age_Group.value_counts().i
ndex, autopct='%1.1f%%')
plt.title('Distribution of Age Groups')
plt.legend(['18-24', '25-34', '35-44', '45-54', '55-64', '65-70'])
plt.show()
```



In [13]:

```
plt.figure(1 , figsize = (20 , 15))
sns.countplot(y = 'Age' , data = df)
plt.show()
```

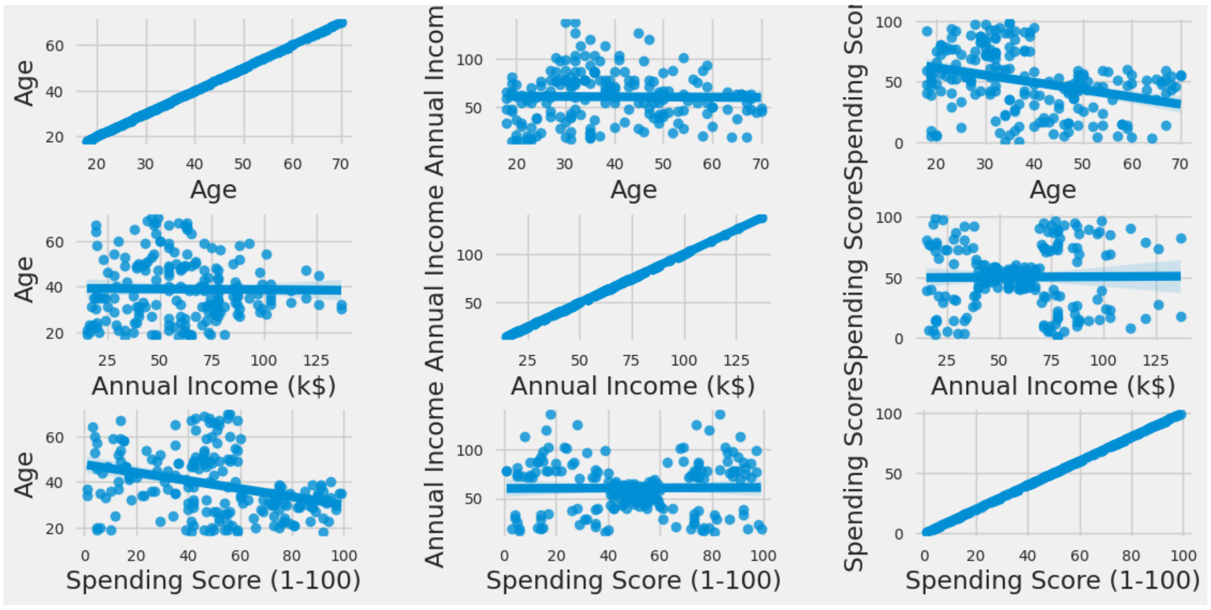


Applying clusters

Ploting the Relation between Age , Annual Income and Spending Score

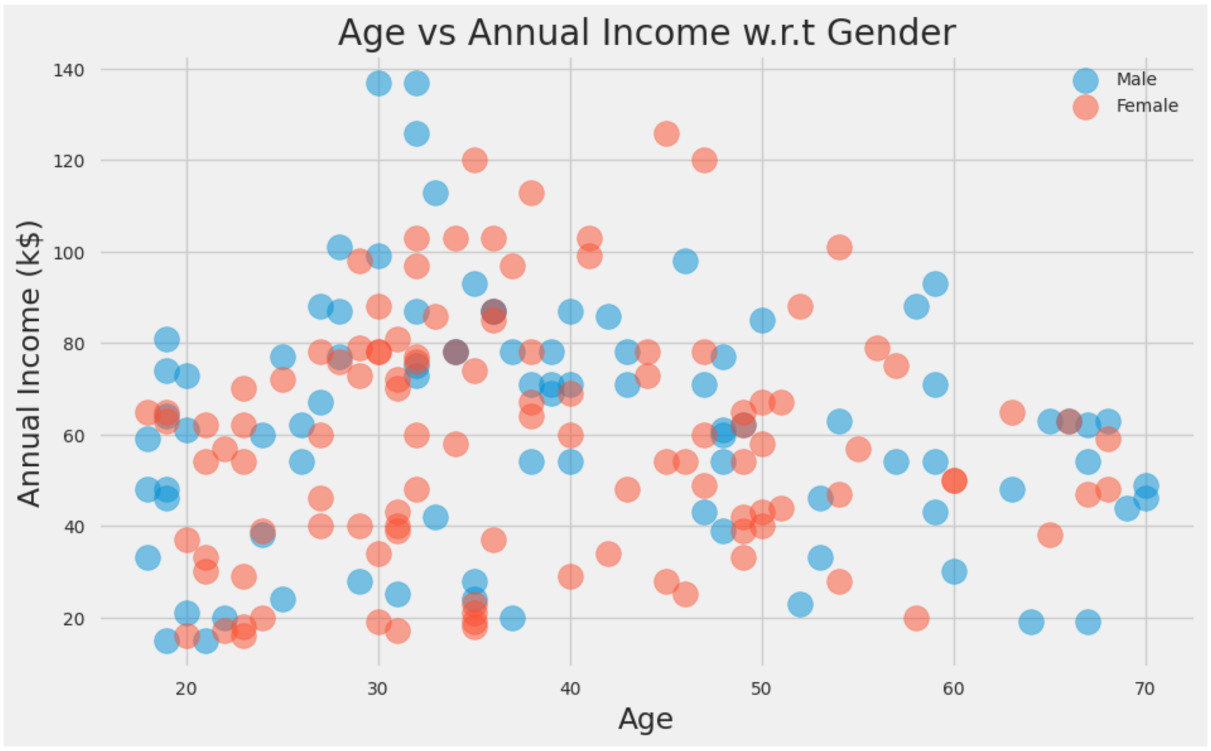
In [14]:

```
plt.figure(1 , figsize = (12 , 6))
n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    for y in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
        n += 1
        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
        sns.regplot(x = x , y = y , data = df)
        plt.ylabel(y.split()[0]+' '+y.split()[1] if len(y.split()) > 1 else
y )
plt.show()
```



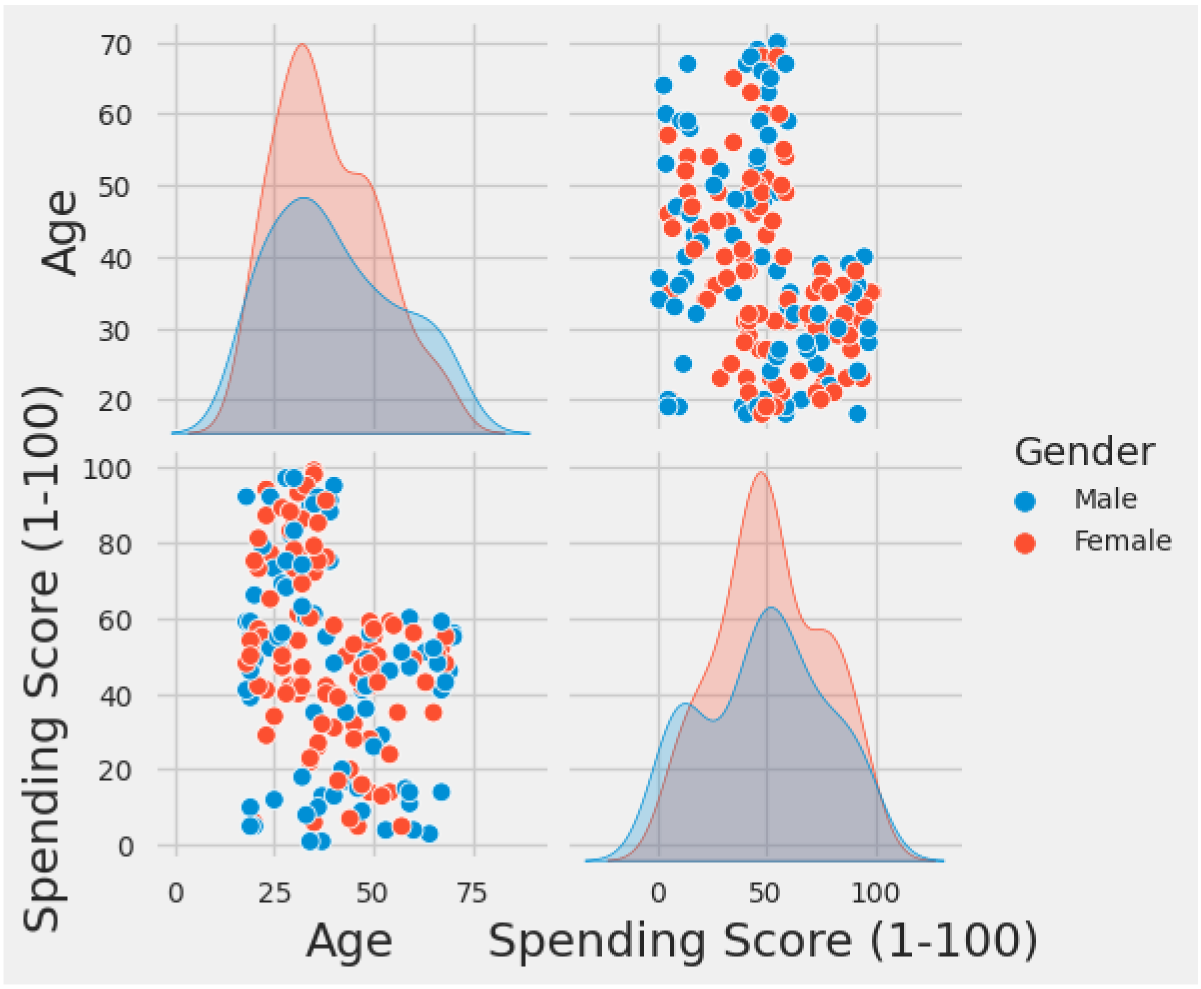
In [15]:

```
plt.figure(1 , figsize = (10 , 6))
for gender in ['Male' , 'Female']:
    plt.scatter(x = 'Age' , y = 'Annual Income (k$)' , data = df[df['Gender
'] == gender] ,
                s = 200 , alpha = 0.5 , label = gender)
plt.xlabel('Age'), plt.ylabel('Annual Income (k$)')
plt.title('Age vs Annual Income w.r.t Gender')
plt.legend()
plt.show()
```



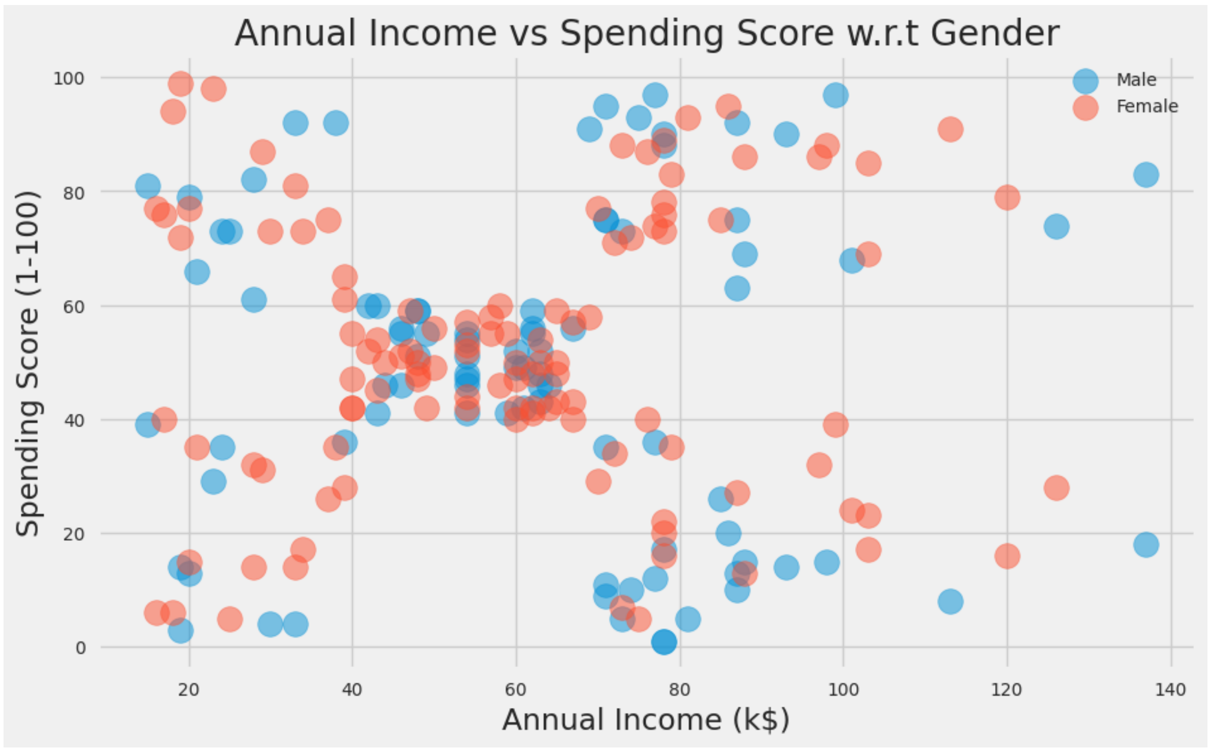
In [16]:

```
sns.pairplot(df.drop("Annual Income (k$)", axis = 1), hue = "Gender");
```



In [17]:

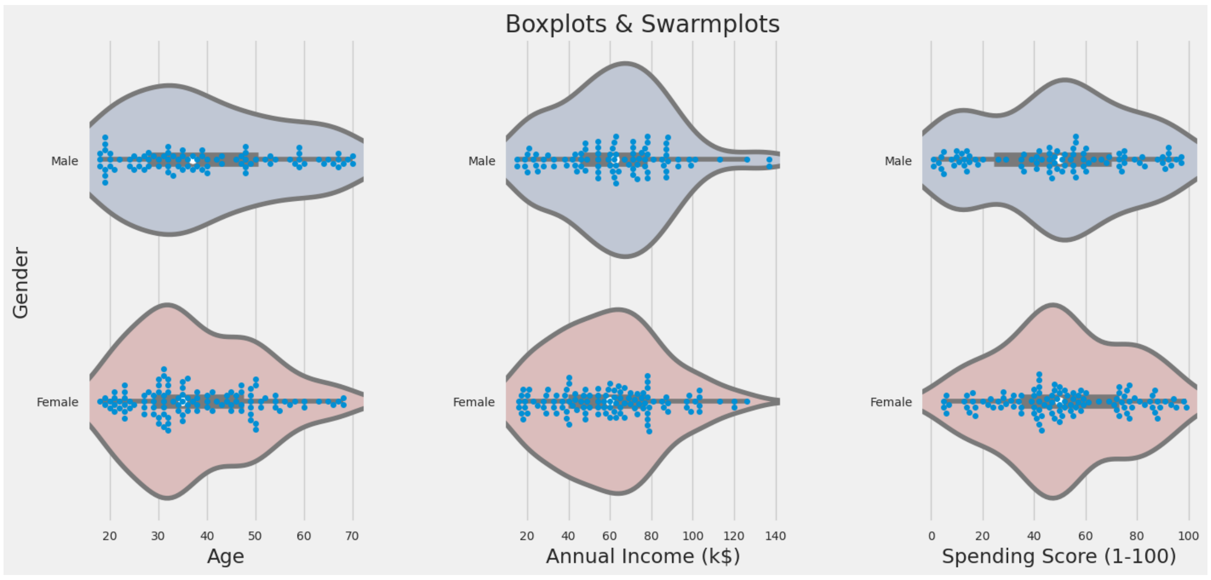
```
plt.figure(1 , figsize = (10 , 6))
for gender in ['Male' , 'Female']:
    plt.scatter(x = 'Annual Income (k$)',y = 'Spending Score (1-100)' ,
                data = df[df['Gender'] == gender] ,s = 200 , alpha = 0.5 ,
label = gender)
plt.xlabel('Annual Income (k$)'), plt.ylabel('Spending Score (1-100)')
plt.title('Annual Income vs Spending Score w.r.t Gender')
plt.legend()
plt.show()
```



Distribution of values in Age , Annual Income and Spending Score according to Gender

In [18]:

```
plt.figure(1 , figsize = (15 , 7))
n = 0
for cols in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1
    plt.subplot(1 , 3 , n)
    plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
    sns.violinplot(x = cols , y = 'Gender' , data = df , palette = 'vlag')
    sns.swarmplot(x = cols , y = 'Gender' , data = df)
    plt.ylabel('Gender' if n == 1 else '')
    plt.title('Boxplots & Swarmplots' if n == 2 else '')
plt.show()
```



Clustering using K- means

1.Segmentation using Age and Spending Score

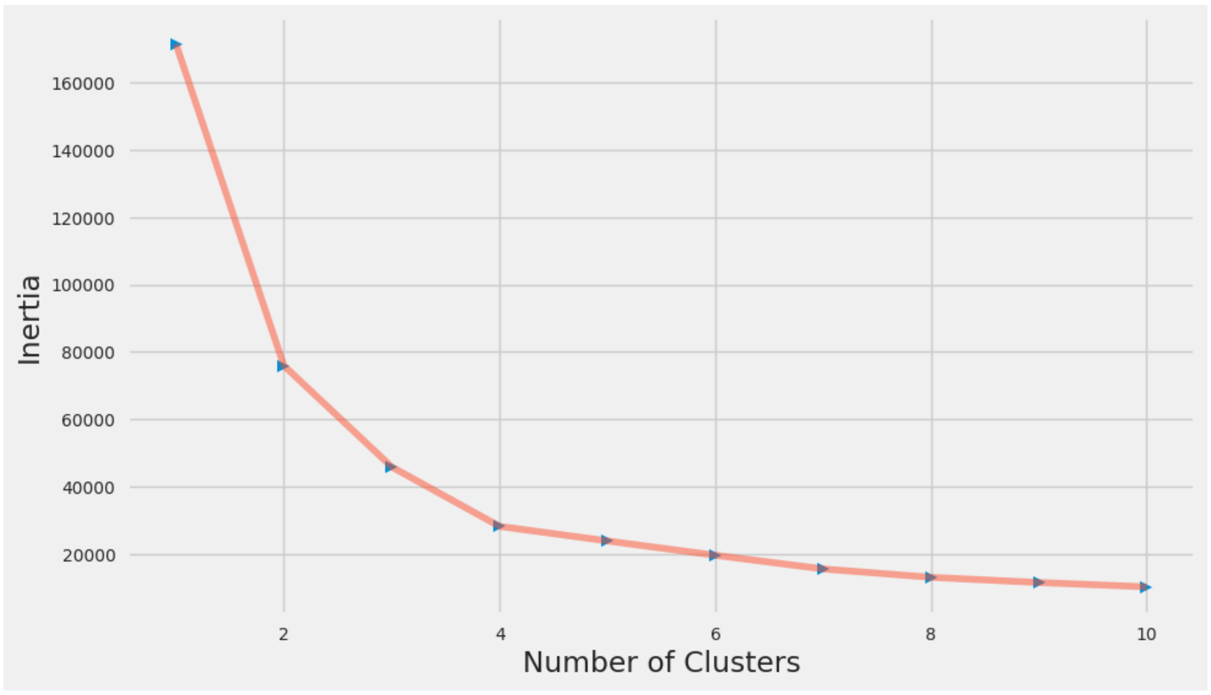
In [19]:

```
'''Age and spending Score'''
X1 = df[['Age' , 'Spending Score (1-100)']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_
iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan'
) )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

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

In [20]:

```
plt.figure(1 , figsize = (10 ,6))
plt.plot(np.arange(1 , 11) , inertia , '>')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```



In [21]:

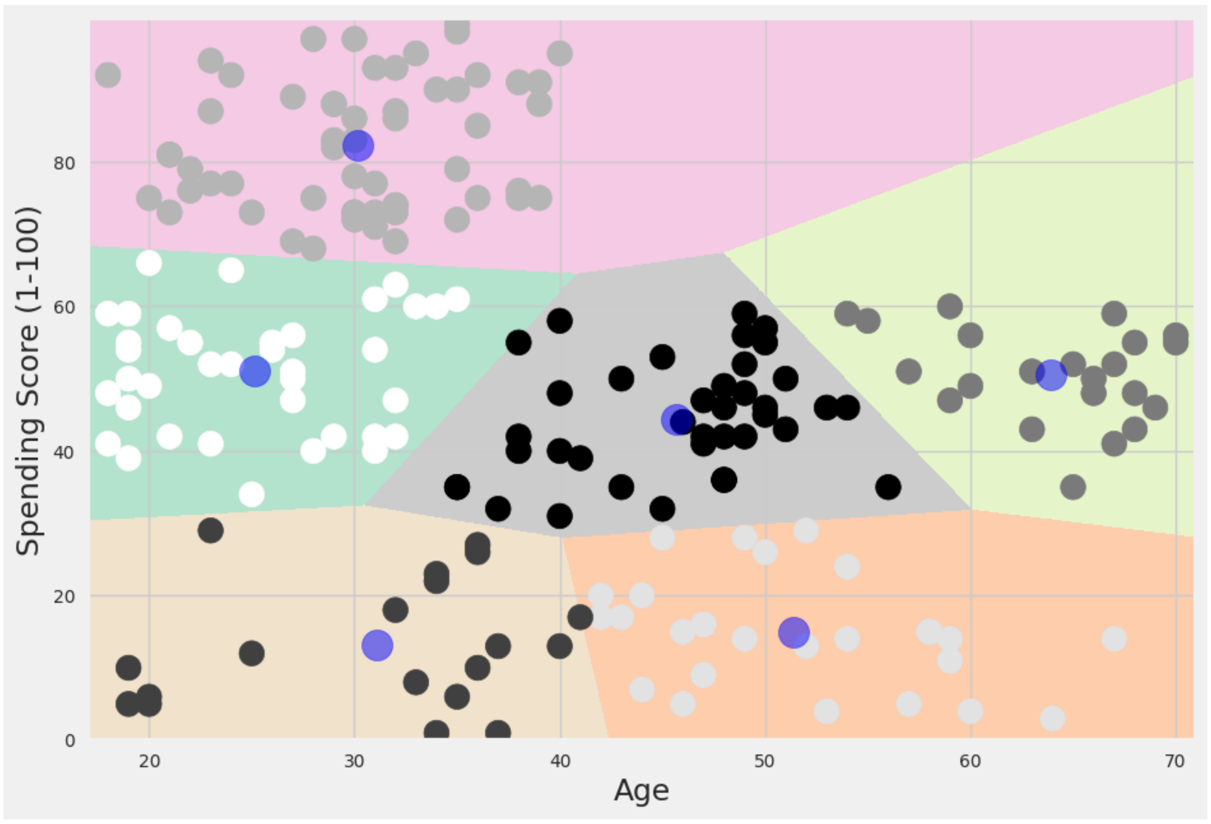
```
algorithm = (KMeans(n_clusters = 6 ,init='k-means++', n_init = 10 ,max_iter
=800,
                        tol=0.0001,  random_state= 111  , algorithm='elkan'
) )
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

In [22]:

```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h)
)
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

In [23]:

```
plt.figure(1 , figsize = (10 , 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 = df , c = labe
ls1 ,
            s = 200 )
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c =
'blue' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age')
plt.show()
```



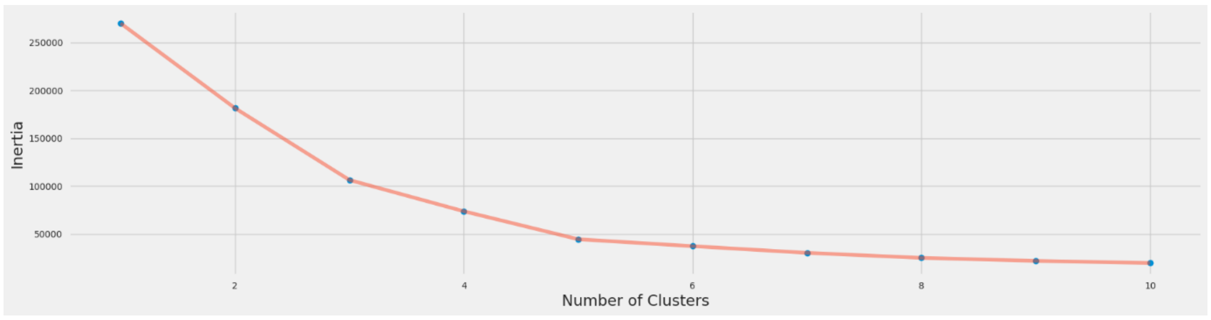
2. Segmentation using Annual Income and Spending Score

In [24]:

```
'''Annual Income and spending Score'''
X2 = df[['Annual Income (k$)' , 'Spending Score (1-100)']].iloc[: , :].valu
es
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_
iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan'
) )
    algorithm.fit(X2)
    inertia.append(algorithm.inertia_)
```

In [25]:

```
plt.figure(1 , figsize = (20 ,5))
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 [26]:

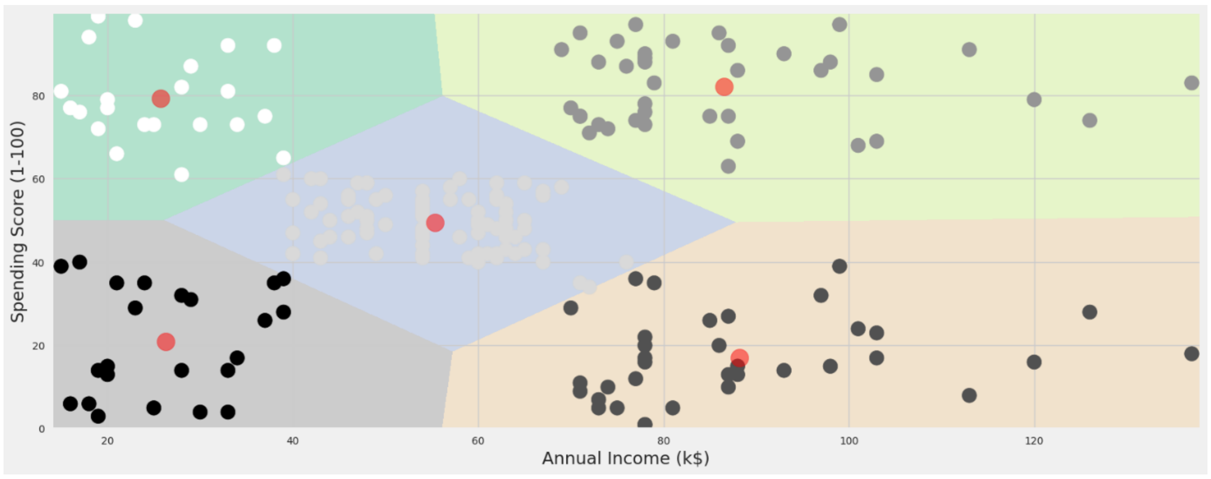
```
algorithm = (KMeans(n_clusters = 5 ,init='k-means++', n_init = 10 ,max_iter
=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan'
) )
algorithm.fit(X2)
labels2 = algorithm.labels_
centroids2 = algorithm.cluster_centers_
```

In [27]:

```
h = 0.02
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h)
)
Z2 = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
```

In [28]:

```
plt.figure(1 , figsize = (18 , 7) )
plt.clf()
Z2 = Z2.reshape(xx.shape)
plt.imshow(Z2 , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'Annual Income (k$)' ,y = 'Spending Score (1-100)' , data
= df , c = labels2 ,
            s = 200 )
plt.scatter(x = centroids2[: , 0] , y =  centroids2[: , 1] , s = 300 , c =
'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Annual Income (k$)')
plt.show()
```



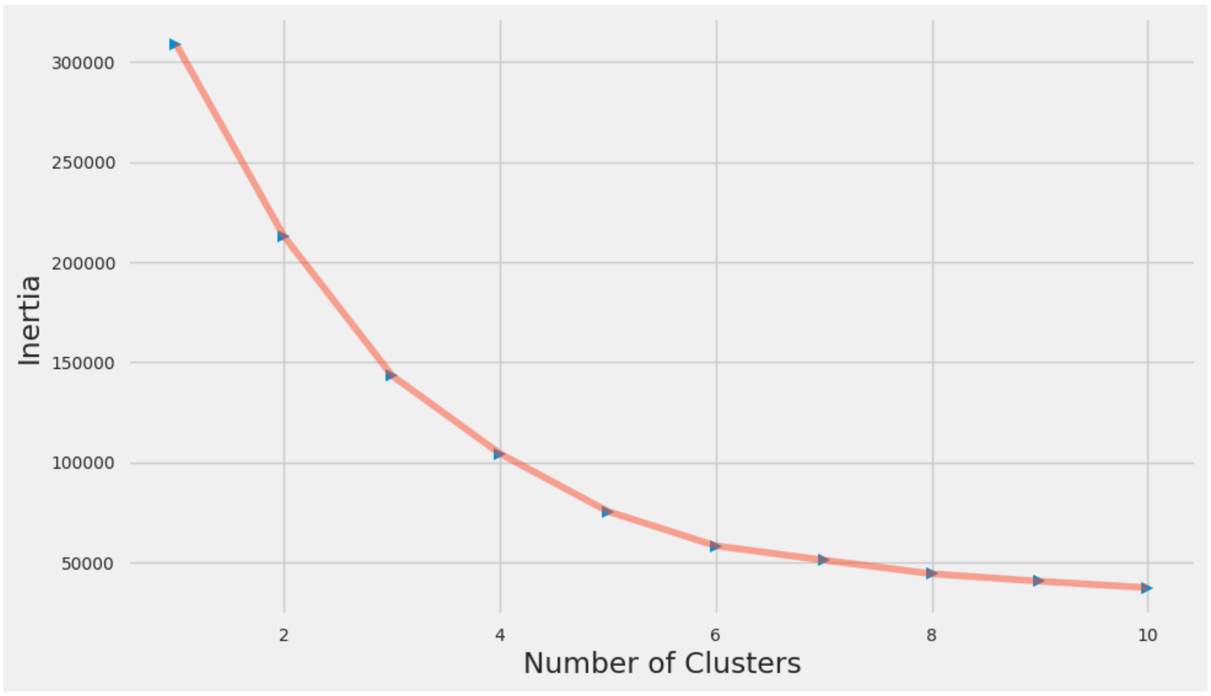
3.Segmentation using Age , Annual Income and Spending Score¶

In [29]:

```
X3 = df[['Age' , 'Annual Income (k$)' ,'Spending Score (1-100)']].iloc[: ,
:].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10 ,max_
iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan'
) )
    algorithm.fit(X3)
    inertia.append(algorithm.inertia_)
```

In [30]:

```
plt.figure(1 , figsize = (10 ,6))
plt.plot(np.arange(1 , 11) , inertia , '>')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```



In [31]:

```
algorithm = (KMeans(n_clusters = 8 ,init='k-means++', n_init = 10 ,max_iter
=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan'
) )
algorithm.fit(X3)
labels3 = algorithm.labels_
centroids3 = algorithm.cluster_centers_
```

In [32]:

```
df['label3'] =  labels3
trace1 = go.Scatter3d(
    x= df['Age'],
    y= df['Spending Score (1-100)'],
    z= df['Annual Income (k$)'],
        mode='markers',
     marker=dict(
        color = df['label3'],
        size= 30,
        line=dict(
            color= df['label3'],
            width= 12
        ),
        opacity=0.5
     )
)
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)
```

Clusters