CUSTOMER SEGMENTATION USING PYTHON

PHASE -5 SUBMITTION

Customer segmentation is the process of dividing a customer base into distinct groups based on shared characteristics, behaviors, or demographics. This helps businesses better understand their customers and tailor their marketing and product strategies to cater to the specific needs and preferences of each segment. Python is a popular programming language for data analysis and segmentation tasks. Here's an overview of how you can perform customer segmentation using Python:

1. Data Collection:

- Start by gathering relevant customer data, which could include demographics, purchase history, website interactions, and any other relevant information.

2. Data Preprocessing:

- Clean and prepare your data by handling missing values, outliers, and data normalization.

3. Exploratory Data Analysis (EDA):

- Conduct an initial analysis of your data to gain insights and understand the distribution of your variables.
- Visualizations, such as histograms, scatter plots, and box plots, can be created using libraries like Matplotlib, Seaborn, and Pandas.

4. Feature Selection/Engineering:

- Identify the relevant features (attributes) that will be used for segmentation. You can use techniques like feature selection and dimensionality reduction (e.g., PCA) to simplify your data.

5. Choose a Segmentation Method:

- There are various techniques for customer segmentation, such as:
  - **Clustering:** Common algorithms include K-Means, Hierarchical Clustering, and DBSCAN.
  - **RFM Analysis:** Recency, Frequency, and Monetary analysis for e-commerce businesses.
  - **PCA and Dimensionality Reduction:** For reducing the number of features.
  - **Machine Learning Algorithms:** Such as decision trees, random forests, and gradient boosting.

6. Apply the Chosen Method:

- Use Python libraries like scikit-learn for clustering algorithms and other data analysis tools to perform segmentation based on the chosen method.

7. Evaluate and Interpret Segments:

- Assess the quality of your segments by metrics like Silhouette Score (for clustering) or business-specific KPIs.
- Interpret the segments to understand the characteristics and behaviors of each group.

8. Implement Targeted Marketing Strategies:

- Based on your segment insights, tailor your marketing, product recommendations, and customer service strategies to better meet the needs of each segment.

9. Monitor and Refine:

- Continuously monitor the effectiveness of your strategies and refine your segmentation as new data becomes available.

This is just a high-level overview, and customer segmentation can be a complex and iterative process. The choice of segmentation method and the

interpretation of segments should align with your business goals and the nature of your data.

INTRODUCTION OF PYTHON

Python is a versatile and popular programming language in the field of customer segmentation due to its rich ecosystem of data analysis, machine learning, and visualization libraries. It offers a wide range of tools and techniques to efficiently and effectively segment a customer base. Here's an introduction to how Python is used in customer segmentation:

1. **Data Handling and Preprocessing:** Python's data manipulation libraries, such as Pandas and NumPy, are used to handle and preprocess customer data. This includes tasks like data cleaning, feature selection, and handling missing values.
2. **Exploratory Data Analysis (EDA):** Python, along with libraries like Matplotlib and Seaborn, allows for the creation of various visualizations to explore and gain insights from the data. EDA helps in understanding the distribution of customer attributes and uncovering patterns.
3. **Feature Engineering:** Python can be used to create new features or transform existing ones to better capture customer behaviors or characteristics. For instance, you can calculate metrics like RFM (Recency, Frequency, Monetary) for e-commerce customer segmentation.
4. **Clustering Techniques:** Python provides a range of clustering algorithms through libraries like scikit-learn. K-Means, Hierarchical Clustering, and DBSCAN are commonly used for customer segmentation. These algorithms group customers with similar attributes together.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(customer_data
```

```
customer_data['Cluster'] = kmeans.labels_
```

Dimensionality Reduction: Python tools like Principal Component Analysis (PCA) are useful for reducing the dimensionality of the data while preserving important information. This can be beneficial when dealing with high-dimensional customer datasets.

python

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_data = pca.fit_transform(customer_data)
```

1. **Machine Learning Algorithms:** Besides clustering, machine learning algorithms like decision trees, random forests, and gradient boosting can be employed for segmentation. These algorithms can capture more complex relationships within the data.
2. **Model Evaluation:** Python libraries offer various metrics and tools to evaluate the quality of segments or clusters, such as the Silhouette Score for clustering models.
3. **Visualization:** Python's data visualization libraries make it easy to represent and interpret customer segments through graphs, charts, and heatmaps. This aids in conveying insights to non-technical stakeholders.
4. **Integration with Business Strategy:** Python allows businesses to implement targeted marketing strategies, product recommendations, and customer service approaches based on the identified segments.

5. **Scalability and Automation:** Python can be used to create automated scripts and workflows for periodic customer segmentation, ensuring that the process is scalable and reproducible.

In summary, Python is a powerful and flexible programming language for customer segmentation, offering a comprehensive set of tools for data manipulation, analysis, modeling, and visualization. It allows businesses to extract valuable insights from their customer data and tailor their strategies to meet the specific needs of different customer segments.

SOURCES:

https://www.kaggle.com/datasets/akram24/mall-customers

chrome,
github

Data Preparation
Data Loading
In [1]:
mall= pd.read_csv(r"/kaggle/input/customer-segmentation-tutorial-in-python/
Mall_Customers.csv")
mall.head()
Out[1]:
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 [2]:
mall.shape
Out[2]:
(200, 5)

Duplicate Check

Data Cleaning

Null Percentage: Columns

In [1]:

```
(mall.isnull().sum() * 100 / len(mall)).value_counts(ascending=False)
```

Out[1]:

```
0.0       5
dtype: int64
```

Null Count: Columns

In [2]:

```
mall.isnull().sum()
```

Out[2]:

```
CustomerID                    0
```

Gender	0
Age	0
Annual Income (k$)	0
Spending Score (1-100)	0

dtype: int64

Null Percentage: Rows

In [3]:

```
(mall.isnull().sum(axis=1) * 100 / len(mall)).value_counts(ascending=False)
```

Out[3]:

0.0 200

dtype: int64

Null Count: Rows

In [4]:

```
mall.isnull().sum(axis=1).value_counts(ascending=False)
```

Out[4]:

0 200

dtype: int64

There are no missing / Null values either in columns or rows

Exploratory Data Analytics

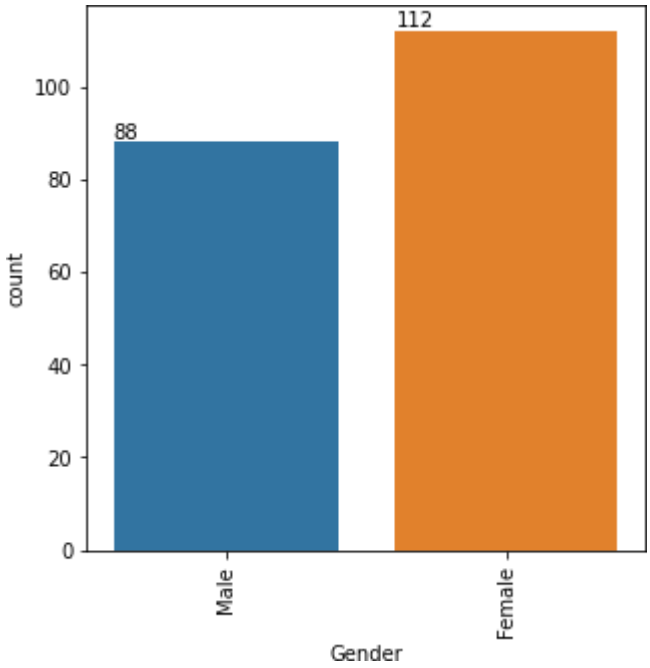
Univariate Analysis

Gender

In [5]:

```
plt.figure(figsize = (5,5))
gender = mall['Gender'].sort_values(ascending = False)
ax = sns.countplot(x='Gender', data= mall)
for p in ax.patches:
```

```
    ax.annotate(str(p.get_height()), (p.get_x() * 1.01 , p.get_height() * 1
.01))
plt.xticks(rotation=90)
plt.show()
```

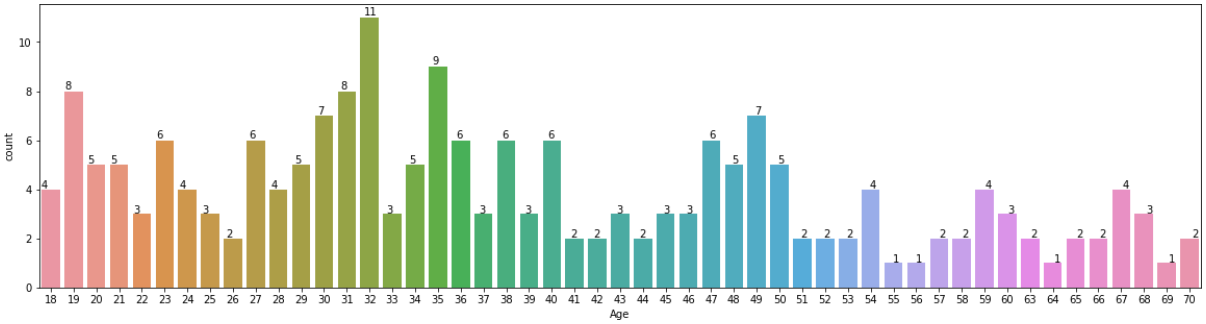


Data is not balanced, 6% more Females have participated than males

Age

In [6]:

```
plt.figure(figsize = (20,5))
gender = mall['Age'].sort_values(ascending = False)
ax = sns.countplot(x='Age', data= mall)
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.01 , p.get_height() * 1
.01))

plt.show()
```



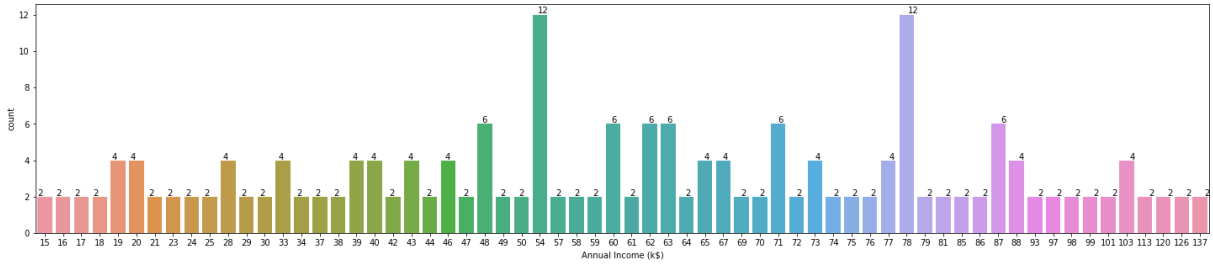
Audience are from Age 18 to 70

Annual Income (k$)

In [7]:

```
plt.figure(figsize = (25,5))
```

```
gender = mall['Annual Income (k$)'].sort_values(ascending = False)
ax = sns.countplot(x='Annual Income (k$)', data= mall)
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.01 , p.get_height() * 1
.01))

plt.show()
```

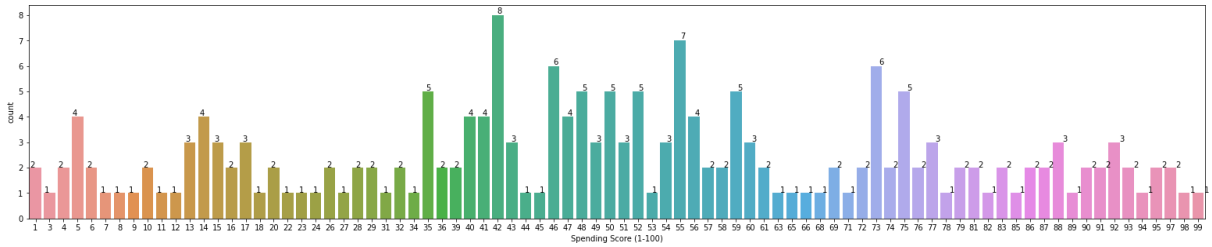


Audience are from Annual Income(k$) range between 15 to 137

Spending Score (1-100)

In [8]:

```
plt.figure(figsize = (27,5))
gender = mall['Spending Score (1-100)'].sort_values(ascending = False)
ax = sns.countplot(x='Spending Score (1-100)', data= mall)
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.01 , p.get_height() * 1
.01))

plt.show()
```



Audience are having Spending Score (1-100) between 1 to 99

In [9]:

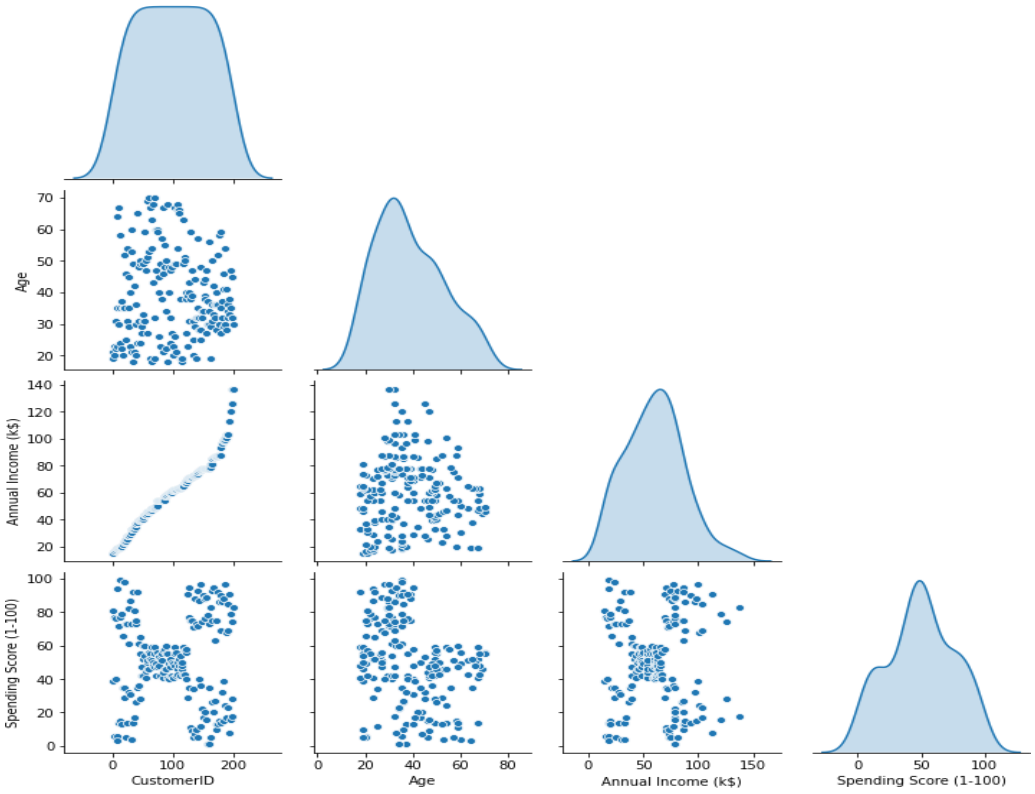
 # Let's check the correlation coefficients to see which variables are highly correlated

```
plt.figure(figsize = (5,5))
sns.heatmap(mall.corr(), annot = True, cmap="rainbow")
plt.savefig('Correlation')
plt.show()
```

- Age and Spending Score (1-100) are moderately correlated with correlation of -0.33

In [10]:

```
sns.pairplot(mall,corner=True,diag_kind="kde")
plt.show()
```

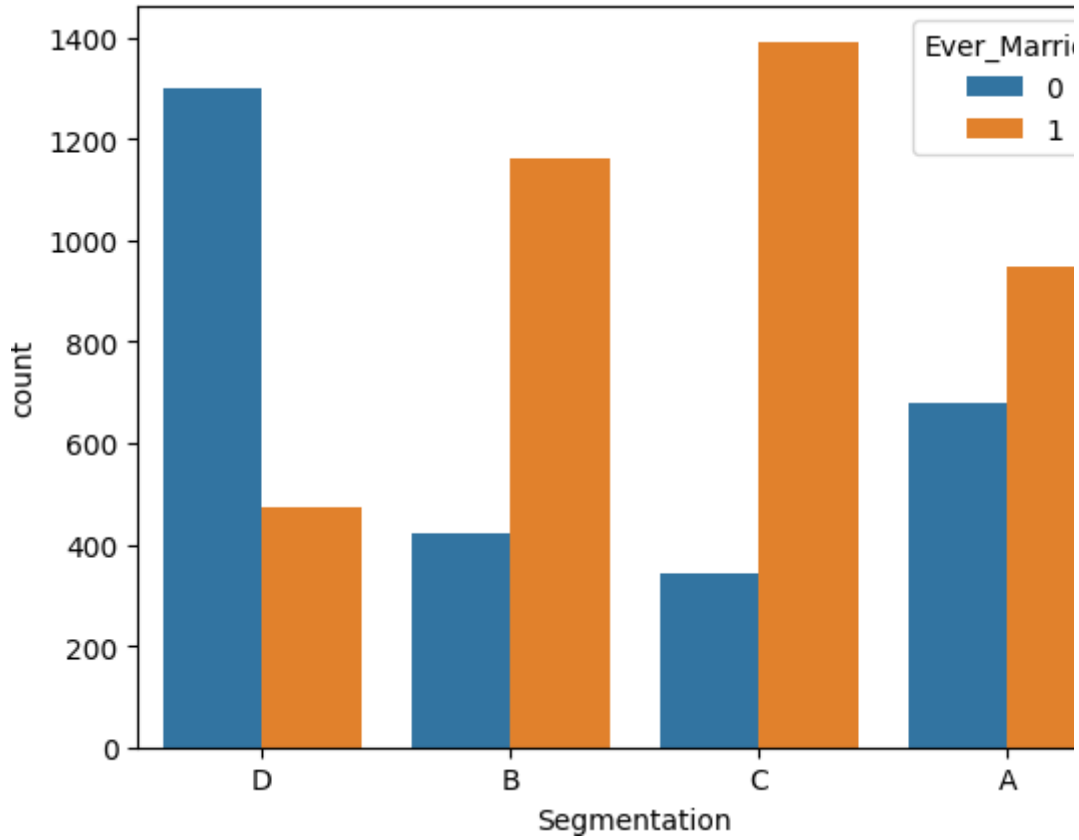


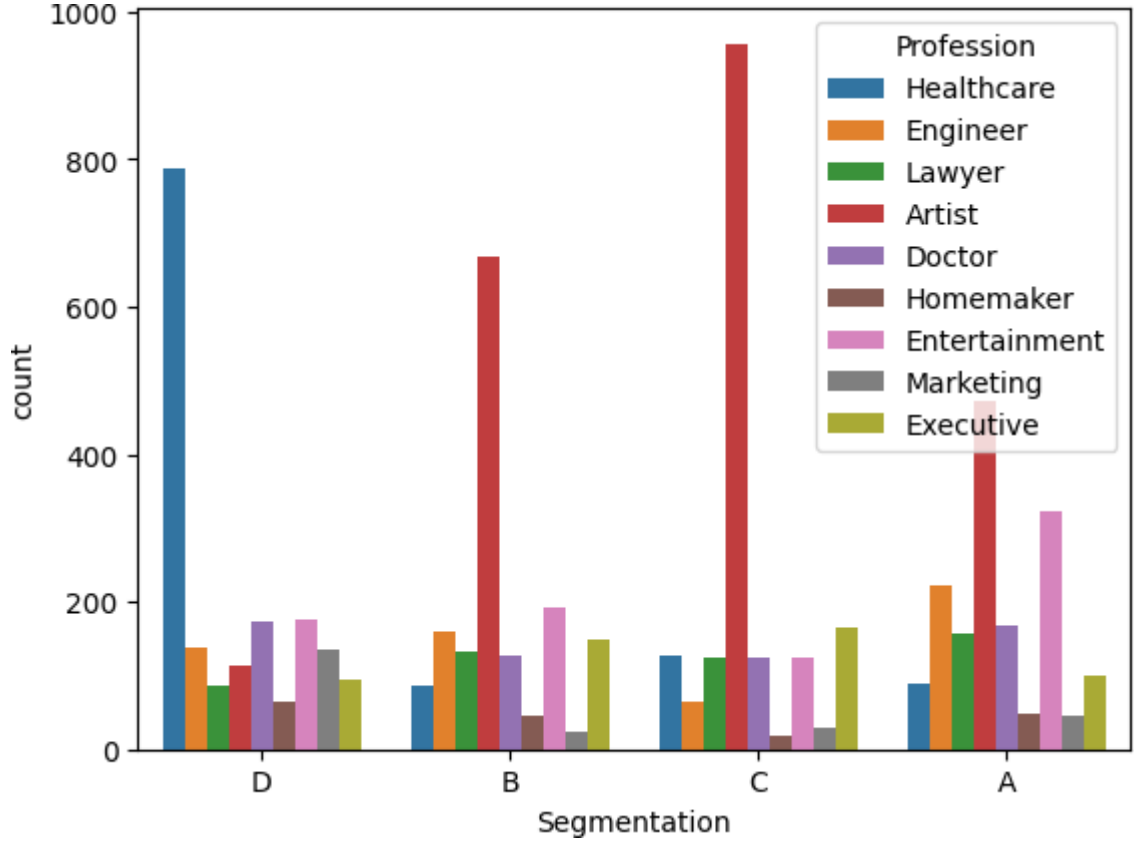
FEATURE ENGINEERING

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 categoricalvariables. 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) forbetter model fitting and performance.

Some Visualizations :





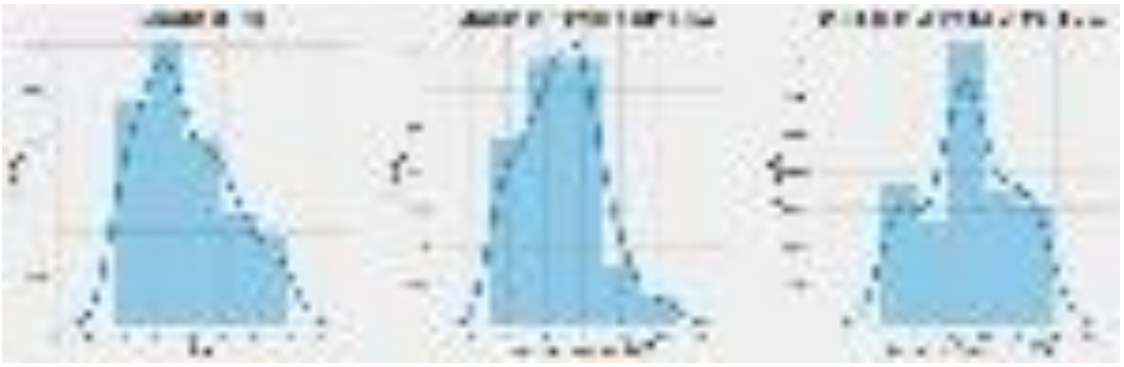
DataExploration

Data Visualization

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

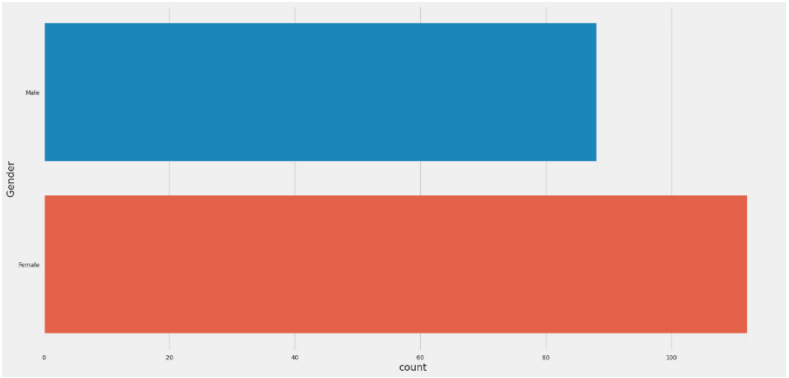
Histograms

```
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()
```



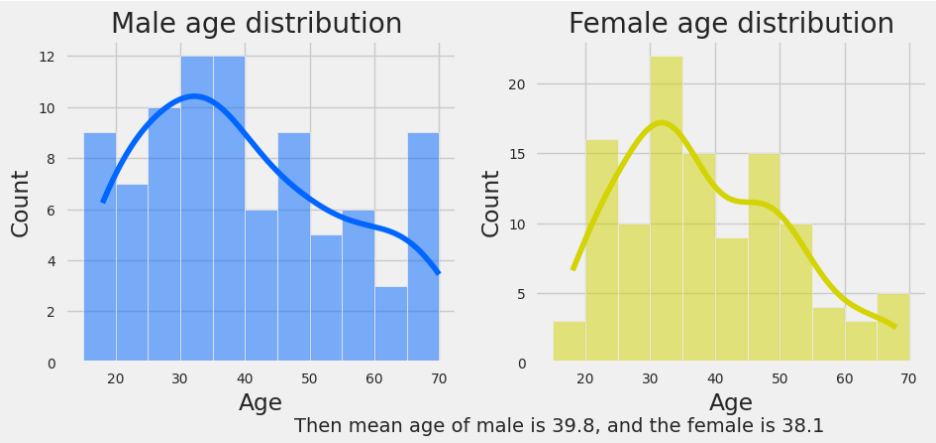
Count Plot of Gender

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

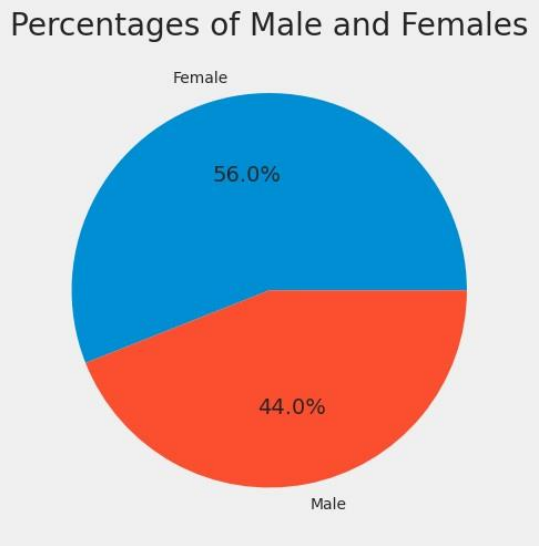


*# Distribution of age with respect to gender*male = df[df.Gender == "Male"]["Age"] female = df[df.Gender != "Male"]['Age']

```
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()
```

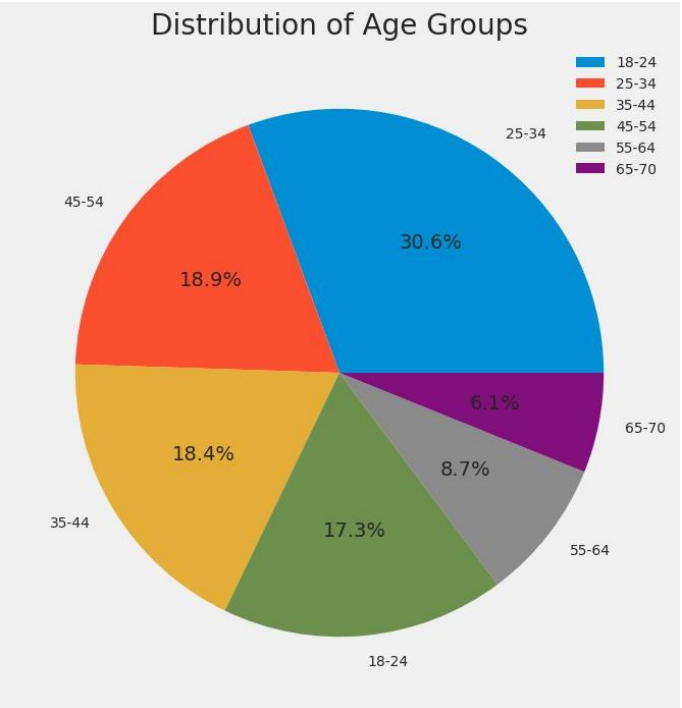


```
plt.pie(df.Gender.value_counts(), labels = ['Female', 'Male'], autopct ="%.
01f%%")
 plt.title('Percentages of Male and Females' );
```

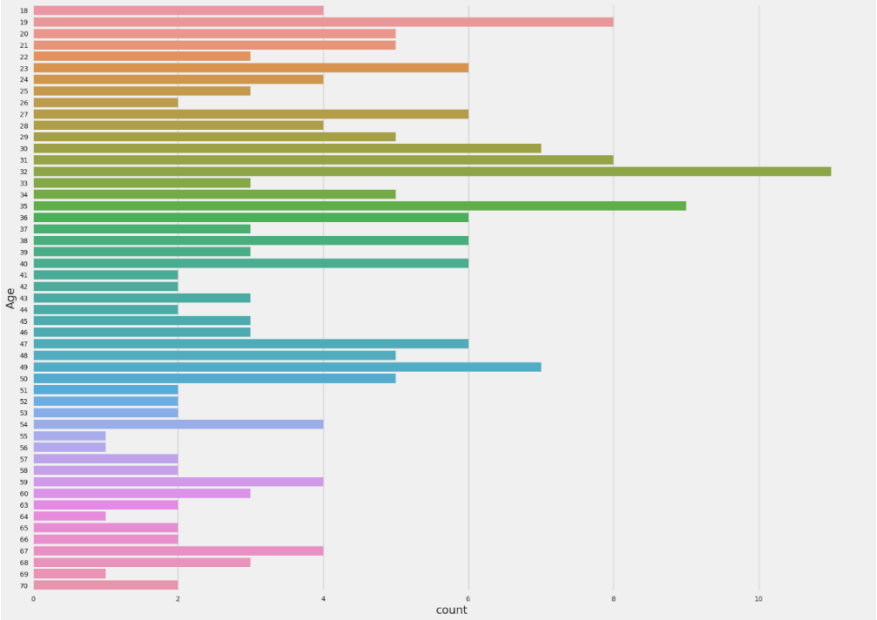


 # Spending Score by age group

```
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()
```



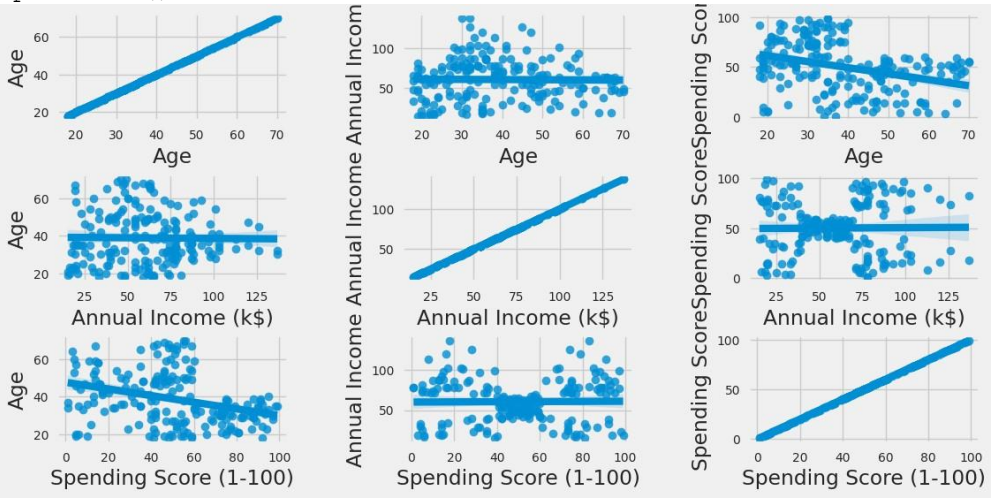
```
plt.figure(1 , figsize = (20 , 15))
sns.countplot(y = 'Age' , data = df)
plt.show()
```



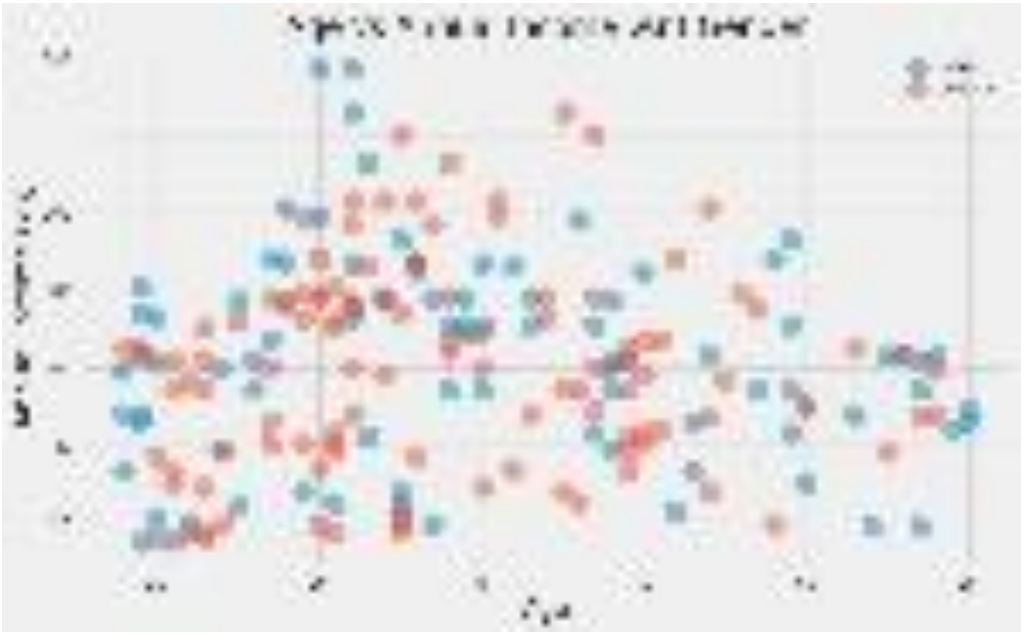
Applying clusters

Ploting the Relation between Age , Annual Income and Spending Score

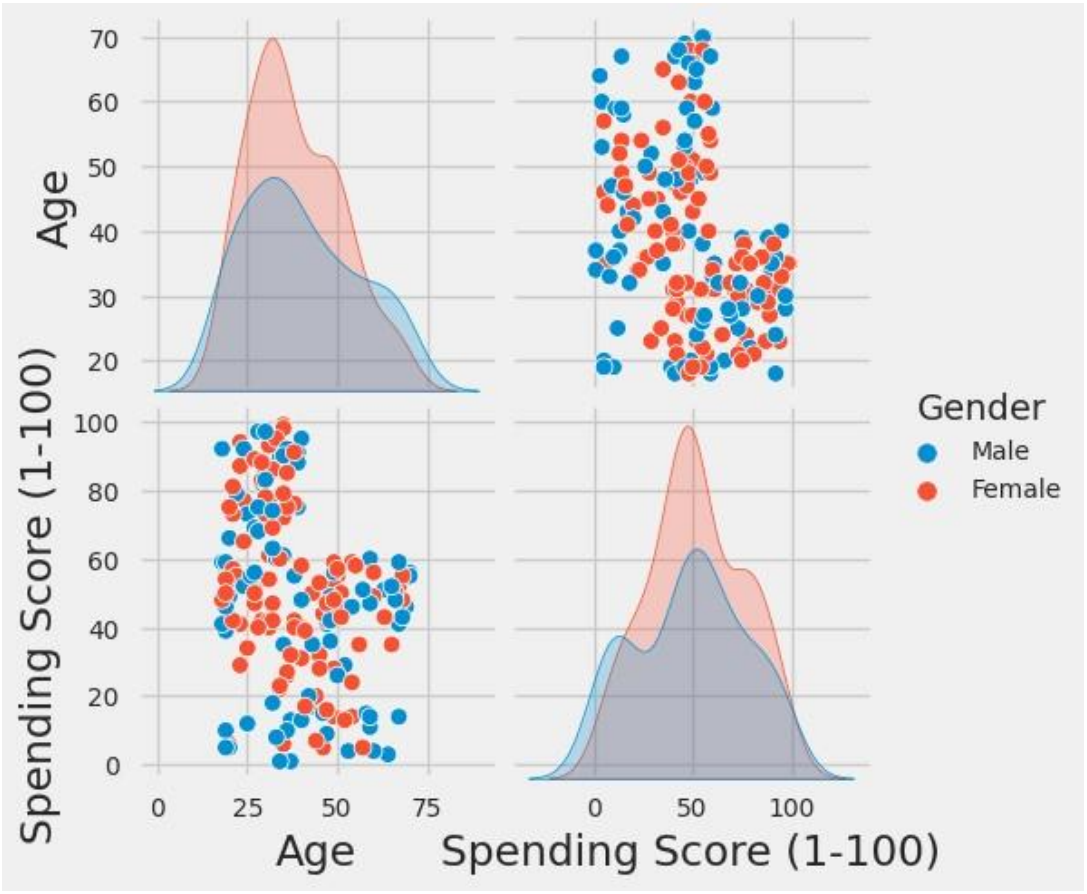
```
plt.figure(1 , figsize = (12 , 6))
n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    for y in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
        n += 1
        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
        sns.regplot(x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



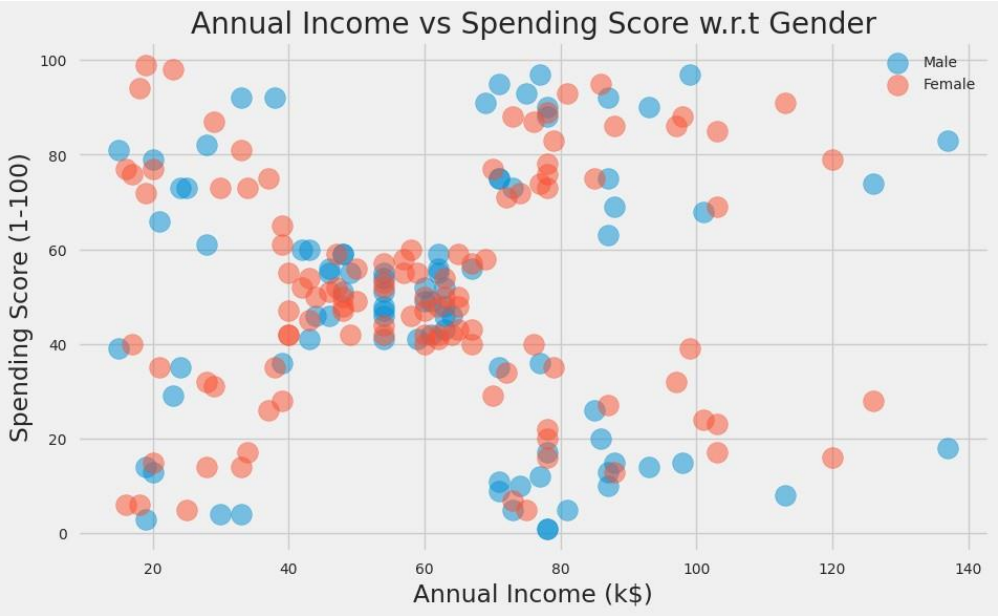
```
plt.figure(1 , figsize = (10 , 6))
for gender in ['Male' , 'Female']:
    plt.scatter(x = 'Age' , y = 'Annual Income (k$)' , data = df[df['Gender
'] == gender] ,
                s = 200 , alpha = 0.5 , label = gender)
plt.xlabel('Age'), plt.ylabel('Annual Income (k$)')
plt.title('Age vs Annual Income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



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

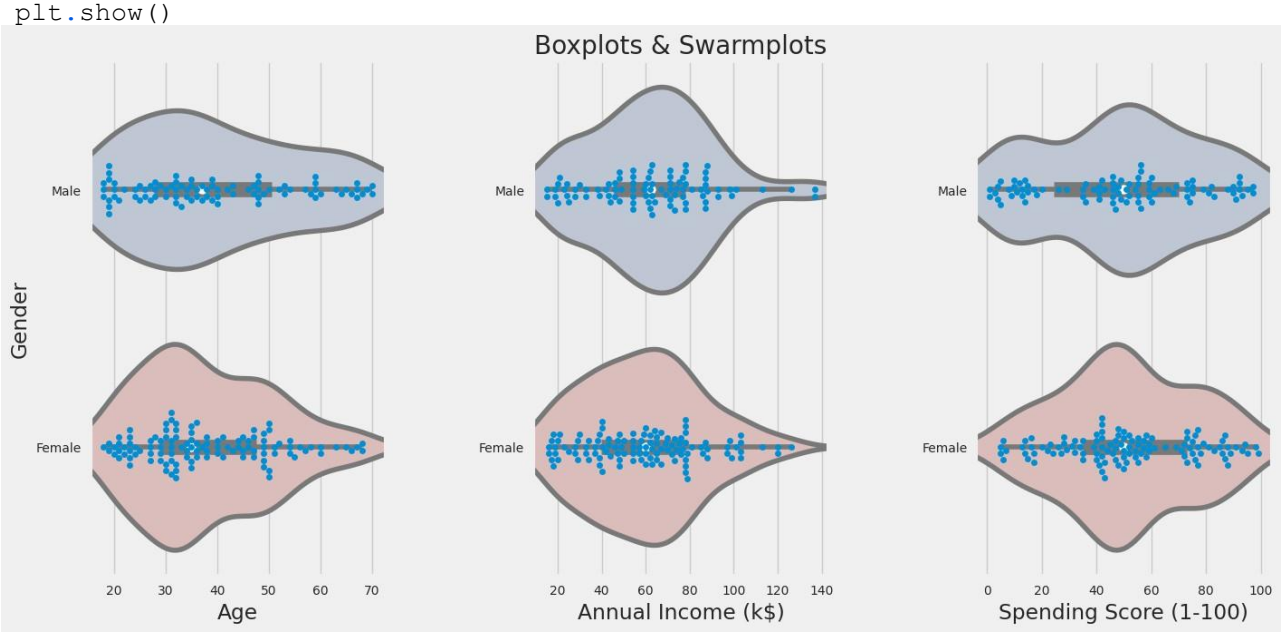

Age
Spending Score (1-100)
70
60
50
40
30
20
100
80
60
40
20
0
0
25
50
75
0
50
100
Age
Spending Score (1-100)
Gender
Male
Female

```
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

```
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

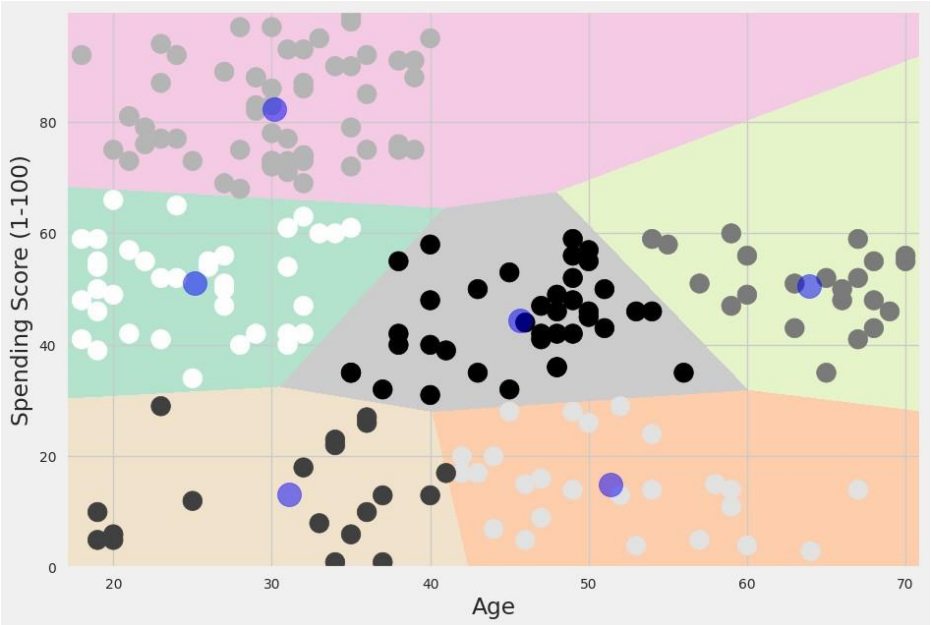
1.Segmentation using Age and Spending Score

```
'''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)

```
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()
```

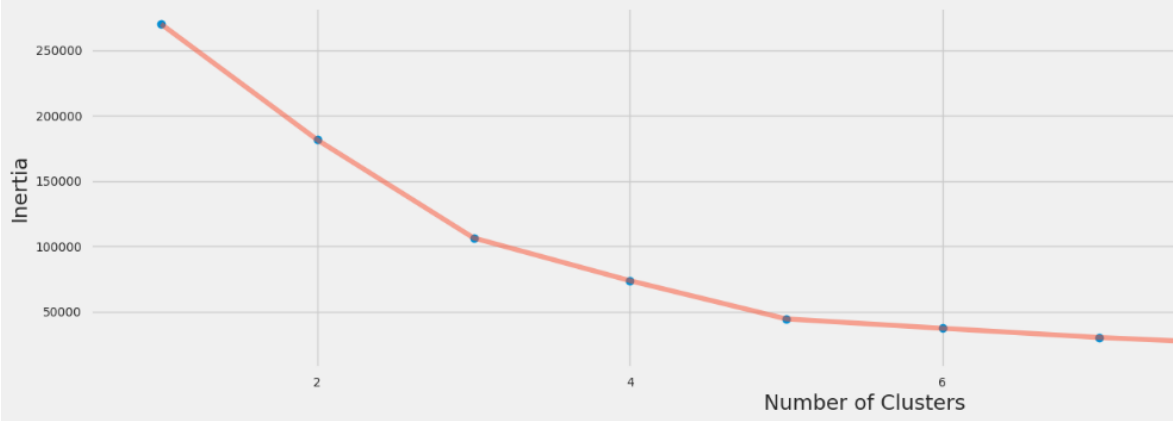
```
 algorithm = (KMeans(n_clusters = 6 ,init='k-means++', n_init = 10 ,max_iter
 =800,

 ) )
 algorithm.fit(X1)
tol=0.0001,  random_state= 111  , algorithm='elkan'
 labels1 = algorithm.labels_
 centroids1 = algorithm.cluster_centers_


 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()])



 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()
```



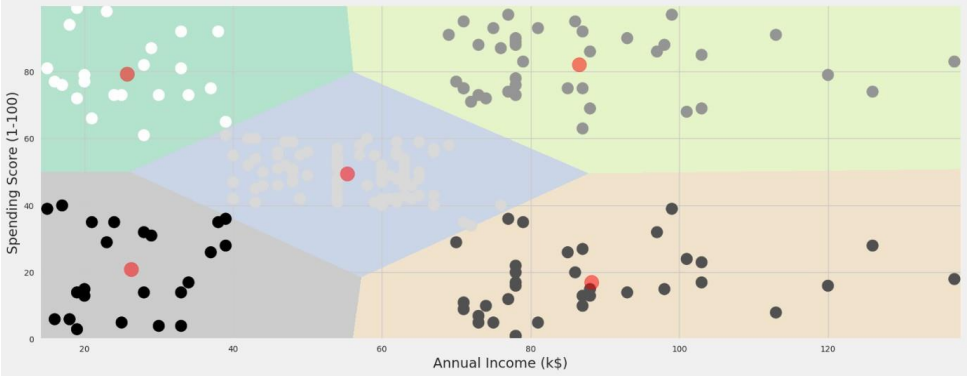
2. Segmentation using Annual Income and Spending Score

'''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=0.00
01,
) )                                                                 random_s
    algorithm.fit(X2)                                               tate=
                                                                    111  ,
                                                                    algorith
                                                                    m='elkan
                                                                    '

    inertia.append(algorithm.inertia_)


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()
```



```
algorithm = (KMeans(n_clusters = 5 ,init='k-means++', n_init = 10 ,max_iter
=300,
                                                                    tol=0.00
                                                                    01,
                                                                    random_s
                                                                    tate=
                                                                    111  ,
                                                                    algorith
                                                                    m='elkan
                                                                    '
) )
algorithm.fit(X2)

labels2 = algorithm.labels_
centroids2 = algorithm.cluster_centers_


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()])


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¶

```
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,

) )
    algorithm.fit(X3)
tol=0.0001,  random_state= 111  , algorithm='elkan'
    inertia.append(algorithm.inertia_)


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()
```

```
 algorithm = (KMeans(n_clusters = 8 ,init='k-means++', n_init = 10 ,max_iter
 =300,

 ) )
 algorithm.fit(X3)
tol=0.0001,  random_state= 111  , algorithm='elkan'
 labels3 = algorithm.labels_
 centroids3 = algorithm.cluster_centers_


 df['label3'] = labels3trace1 = 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)
```

advantages and disadvantages:

t helps marketers categorize the market and try to meet the group's demands. Market segmentation helps companies develop marketing strategies according to the customer's perspective. It helps companies not only in a signal way but in different ways. It increases the customer base, profit margins, and other benefits.

Conclusion:

In conclusion , customer segmentation through loading and

preprocessing of customer data is a critical step in understanding andeffectively targeting your customer base. By organizing and analyzingthis data, businesses can identify distinct customer groups, tailor their marketing strategies and improve overall customer experiences.This process empowers companies to make data-driven decisions andenhance customer satisfaction, ultimately