In [1]:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import numpy as np
from scipy.stats import iqr
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans


df = pd.read_csv("marketing_campaign.csv", sep="\t")
df.head()
```

Out[1]:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVisit
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58	635	...	
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38	11	...	
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	...	
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26	11	...	
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	...	

5 rows × 29 columns

In [2]:

```
df.dtypes
```

Out[2]:

```
ID                       int64
Year_Birth               int64
Education               object
Marital_Status          object
Income                 float64
Kidhome                  int64
Teenhome                 int64
Dt_Customer             object
Recency                  int64
MntWines                 int64
MntFruits                int64
MntMeatProducts          int64
MntFishProducts          int64
MntSweetProducts         int64
MntGoldProds             int64
NumDealsPurchases        int64
NumWebPurchases          int64
NumCatalogPurchases      int64
NumStorePurchases        int64
NumWebVisitsMonth        int64
AcceptedCmp3             int64
AcceptedCmp4             int64
AcceptedCmp5             int64
AcceptedCmp1             int64
AcceptedCmp2             int64
Complain                 int64
Z_CostContact            int64
Z_Revenue                int64
Response                 int64
dtype: object
```

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   2240 non-null   int64  
 1   Year_Birth           2240 non-null   int64  
 2   Education            2240 non-null   object 
 3   Marital_Status       2240 non-null   object 
 4   Income               2216 non-null   float64
 5   Kidhome              2240 non-null   int64  
 6   Teenhome             2240 non-null   int64  
 7   Dt_Customer          2240 non-null   object 
 8   Recency              2240 non-null   int64  
 9   MntWines             2240 non-null   int64  
 10  MntFruits            2240 non-null   int64  
 11  MntMeatProducts      2240 non-null   int64  
 12  MntFishProducts      2240 non-null   int64  
 13  MntSweetProducts     2240 non-null   int64  
 14  MntGoldProds         2240 non-null   int64  
 15  NumDealsPurchases    2240 non-null   int64  
 16  NumWebPurchases      2240 non-null   int64  
 17  NumCatalogPurchases  2240 non-null   int64  
 18  NumStorePurchases    2240 non-null   int64  
 19  NumWebVisitsMonth    2240 non-null   int64  
 20  AcceptedCmp3         2240 non-null   int64  
 21  AcceptedCmp4         2240 non-null   int64  
 22  AcceptedCmp5         2240 non-null   int64  
 23  AcceptedCmp1         2240 non-null   int64  
 24  AcceptedCmp2         2240 non-null   int64  
 25  Complain             2240 non-null   int64  
 26  Z_CostContact        2240 non-null   int64  
 27  Z_Revenue            2240 non-null   int64  
 28  Response             2240 non-null   int64  
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB
```

In [4]:
```python
df["TotalAmountSpent"] = df["MntFishProducts"] + df["MntFruits"] + df["MntGoldProds"] + df["MntSweetProducts"] +
```

Univariate analysis Univariate analysis entails evaluating a single feature in order to get insights about it. So, the initial step in performing EDA is to undertake univariate analysis, which includes evaluating descriptive or summary statistics about the feature.

For example you might check a feature distribution, proportion of a feature, and so on.

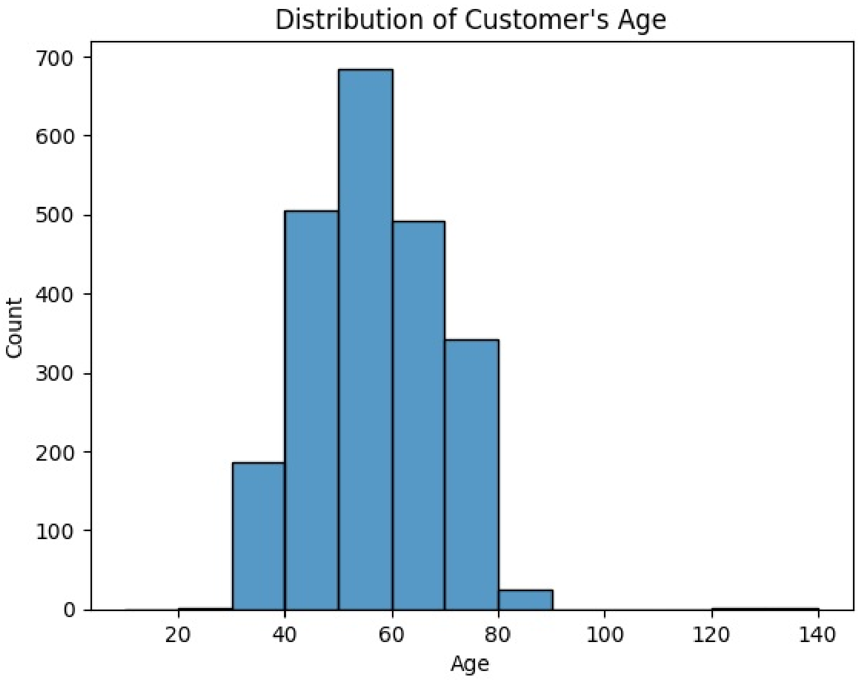
In our case, we will check the distribution of customer's ages in the dataset. We can do that by typing the following:

In [6]:
```python
print(df.columns.tolist())
```

```
['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome', 'Teenhome', 'Dt_Customer', 'Recency', '
MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts', 'MntSweetProducts', 'MntGoldProds', 'NumDealsPurch
ases', 'NumWebPurchases', 'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3', 'Acce
ptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2', 'Complain', 'Z_CostContact', 'Z_Revenue', 'Response',
'TotalAmountSpent']
```

In [11]:
```python
df.columns = df.columns.str.strip()
```

In [13]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from datetime import datetime

# Assuming 'Year_Birth' is the year of birth
current_year = datetime.now().year
df['Age'] = current_year - df['Year_Birth']

# Plotting the histogram
sns.histplot(data=df, x="Age", bins=list(range(10, 150, 10)))
plt.title("Distribution of Customer's Age")
plt.show()
```



Bivariate Analysis After you've performed univariate analysis on all your feature of interest, the next step is to perform bivariate analysis. This involves comparing two attributes at the same time.

Bivariate analysis entails determining the correlation between two features, for example.

In our case, some of the bivariate analysis we'll perform in the project include observing the average total spent across different client age groups, determining a correlation between customer income and total amount spent, and so on, as shown below.

For example, in our case we want to check the relationship between a Customer's Income and TotalAmountSpent. We can do that by typing the following:

In [15]:
```
fig = px.scatter(data_frame=df, x="Income",
                 y="TotalAmountSpent",
                 title="Relationship Between Customer's Income and Total Amount Spent",
                height=500,
                color_discrete_sequence = px.colors.qualitative.G10[1:])
fig.show()
```

We can see from the above analysis that as the Income increases so does the TotalAmountSpent. So from the analysis we can postulate that Income is one of key factor that determines how much a customer might spend.

Multivariate Analysis After you've completed univariate (analysis of single feature) and bivariate (analysis of two features) analysis, the last phase of EDA is to perform Multivariate Analysis.

Multivariate Analysis consists of understanding the relationship between two or more variables.

In our project, one of the multivariate analysis we'll do is to understand the relationship between Income, TotalAmountSpent, and Customer's Education.

In [17]:
```
fig = px.scatter(
    data_frame=df,
    x = "Income",
    y= "TotalAmountSpent",
    title = "Relationship between Income VS Total Amount Spent Based on Education",
    color = "Education",
    height=500
)
fig.show()
```

How to Build the Segmentation Model After we've finished our analysis, the next step is to create the model that will segment the customers. KMeans is the model we'll use. It is a popular segmentation model that is also quite effective.

The KMeans model is an unsupervised machine learning model that works by simply splitting N observations into K numbers of clusters. The observations are grouped into these clusters based on how close they are to the mean of that cluster, which is commonly referred to as centroids.

When you fit the features into the model and specify the number of clusters or segments you want, KMeans will output the cluster label to which each observation in the feature belongs.

Let's talk about the features you might want to fit into a KMeans model. There are no limits to the number of features you can use to build a Customer segmentation model – but in my opinion, fewer's better. This is because you will be able to grasp and interpret the outcomes of each segment more easily and clearly with fewer features.

In our scenario, we will first construct the KMeans model with two features and then build the final model with three features. But, before we get started, let's go over the KMeans assumptions, which are as follows:

The features must be numerical.

The features you're fitting into KMeans must be normally distributed. This is because KMeans (since it calculates average distance) is affected by outliers (values that deviate a lot from the others). As a result, any skewed feature must be changed in order to be normally distributed. Fortunately, we can use Numpy's logarithm transformation package np.log()

The features must also be of the same scale. For this, we'll use the Scikit-learn StandardScaler() module.

We'll design our KMeans model now that we've grasped the main concept. So, for our first model, we'll use the Income and TotalAmountSpent features.

To begin, because the Income feature has missing values, we will fill it with the median number.

In [19]:
```
df['Age'] = current_year - df['Year_Birth']
```

In [20]:
```
df = df.copy()  # Ensure you're working with a copy of the DataFrame
df['Age'] = current_year - df['Year_Birth']
```

In [23]:
```
df['Income'] = df['Income'].fillna(df['Income'].median())
```

In [25]:
```
#After that, we'll assign the features we want to work with, Income and TotalAmountSpent, to a variable called d
```

In [26]:
```
data = df[["Income", "TotalAmountSpent"]]
```

In [27]:
```
##Once that's done we will transform features and save the result into a variable called data_log.
```

In [28]:
```
df_log = np.log(data)
```

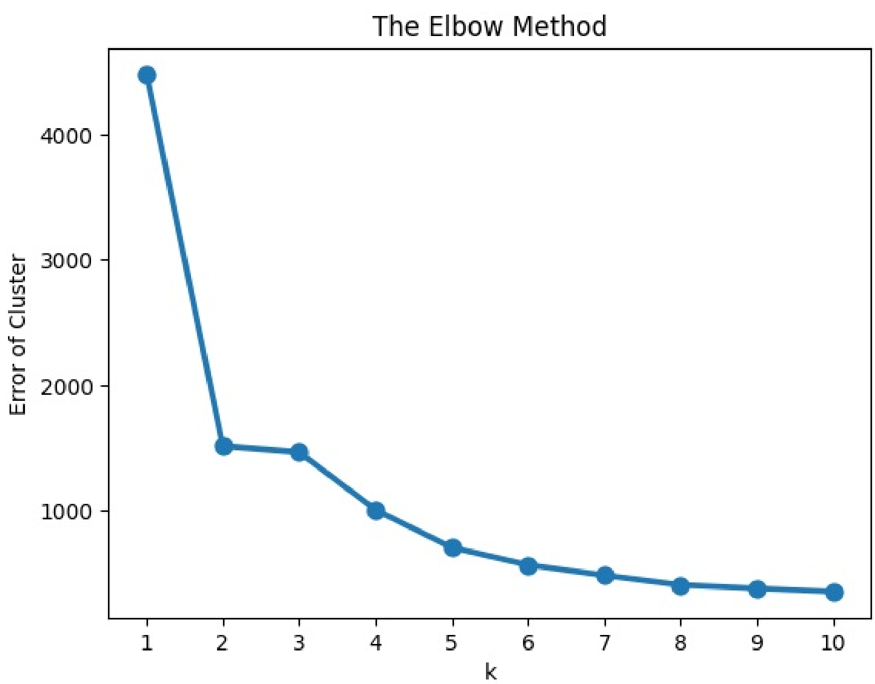
In [30]:
```
##Then we will scale the result using Scikit-learn StandardScaler():

std_scaler = StandardScaler()
df_scaled = std_scaler.fit_transform(df_log)
```

Once that's done we can then build the model. So the KMeans model requires two parameters. The first is random_state and the second one is n_clusterswhere:

n_clusters represents the number of clusters or segments to be derived from KMeans.

random_state: is required for reproducible results.

In [34]:
```
errors = []
for k in range(1, 11):
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(df_scaled)
    errors.append(model.inertia_)

plt.title('The Elbow Method')
plt.xlabel('k')
plt.ylabel('Error of Cluster')

# Corrected pointplot syntax
sns.pointplot(x=list(range(1, 11)), y=errors)

plt.show()
```



Let's summarize what the above code does. We specified the number of clusters to experiment with, which is in the range(1, 11). Then we fit the features on those clusters and added the error to the list we created before above.

Following that, we plot the error for each cluster. The diagram shows that the cluster that creates the elbow is three. So three clusters is the best value for our model. As a result, we will build the KMeans model utilizing three clusters.

In [35]:
```
model = KMeans(n_clusters = 3, random_state=42)
model.fit(df_scaled)
```

Out[35]:

KMeans

KMeans(n_clusters=3, random_state=42)

In [37]:
```
##Now we've built our model. The next thing will be to assign the cluster label for each observation. So we wil

data = data.assign(ClusterLabel = model.labels_)
```

In [38]:
```
data.groupby("ClusterLabel")[["Income", "TotalAmountSpent"]].median()
```

Out[38]:

ClusterLabel	Income	TotalAmountSpent
0	32765.0	57.0
1	157243.0	107.0
2	65203.0	934.0

We can see that there is a trend within the clusters:

Cluster 0 translates to customers who earn less and spend less.

Cluster 1 represent customers that earn more and spend more.

Cluster 2 represents customers that earn moderate and spend moderate.

In [39]:
```
fig = px.scatter(
    data_frame=data,
    x = "Income",
    y= "TotalAmountSpent",
    title = "Relationship between Income VS Total Amount Spent",
    color = "ClusterLabel",
    height=500
)
fig.show()
```

In [40]:
```
data = df[["Age", "Income", "TotalAmountSpent"]]
df_log = np.log(data)
std_scaler = StandardScaler()
df_scaled = std_scaler.fit_transform(df_log)
```

In [43]:
```
# Assuming 'data' contains the 'ClusterLabel' column
data = data.assign(ClusterLabel=model.labels_)

# Group by the 'ClusterLabel' and calculate mean and median for the specified columns
result = data.groupby("ClusterLabel").agg({"Age": "mean", "Income": "median", "TotalAmountSpent": "median"}).ro
```

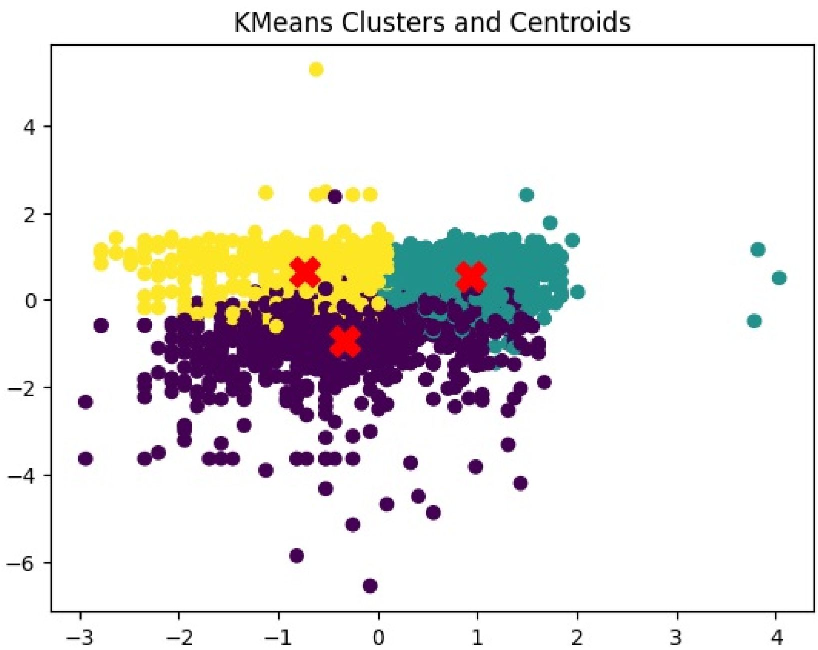
In [45]:
```
model = KMeans(n_clusters=3, random_state=42)
```

```python
model.fit(df_scaled)

data = data.assign(ClusterLabel= model.labels_)

result = data.groupby("ClusterLabel").agg({"Age":"mean", "Income":"median", "TotalAmountSpent":"median"}).round
```

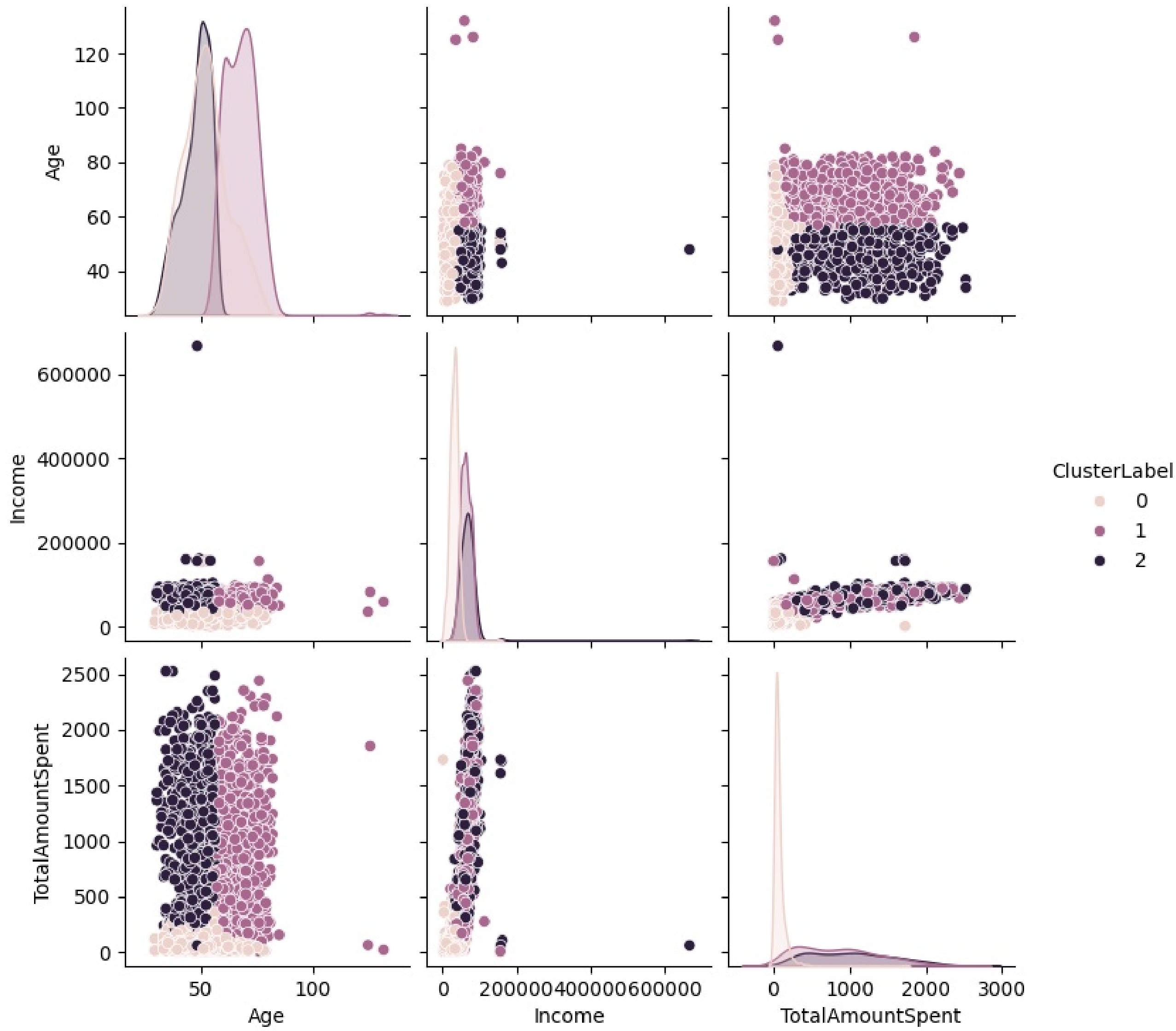
In [48]:
```python
print(result)
```

```
               Age   Income  TotalAmountSpent
ClusterLabel
0             52.0  31801.0              54.0
1             68.0  62820.0             825.0
2             47.0  67384.0            1001.0
```

In [49]:
```python
cluster_centers = model.cluster_centers_
plt.scatter(df_scaled[:, 0], df_scaled[:, 1], c=model.labels_, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], s=200, c='red', marker='X')  # Red X for centroids
plt.title('KMeans Clusters and Centroids')
plt.show()
```



In [50]:
```python
import seaborn as sns

data['ClusterLabel'] = model.labels_
sns.pairplot(data, hue='ClusterLabel', vars=['Age', 'Income', 'TotalAmountSpent'])
plt.show()
```



Cluster 0 depicts young customers that earn a lot and also spend a lot.

Cluster 1 translates to older customers that earn a lot and also spend a lot.

Cluster 2 depicts young customers that earn less and also spend less.

In [54]:
```
fig = px.scatter_3d(data_frame=data, x="Income",
                    y="TotalAmountSpent", z="Age", color="ClusterLabel", height=550,
                   title = "Visualizing Cluster Result Using 3 Features")
fig.show()
```

Conclusion In this tutorial, you learnt how to build a customer segmentation model. There are a lot of features we didn't touch on in this article. But I suggest that you experiment with it and create customer segmentation models using different features.

In [ ]:

```

```