In [1053]:
```python
import numpy as np
import pandas as pd
# For visualizations
import matplotlib.pyplot as plt
import seaborn as sns
from mycolorpy import colorlist as mcp

# For regular expressions
import re
# For handling string
import string
# For performing mathematical operations
import math

from sklearn.metrics import silhouette_score
from scipy.stats import zscore
```

Exploratory Data Analysis

**Authored by: Sangita Baitalik

In [1054]:
```python
#An automobile company has plans to enter new markets with their existing pro
#After intensive market research, they've deduced that the behavior of new ma
#In their existing market, the sales team has classified all customers into 4
#Then, they performed segmented outreach and communication for different segm
#This strategy has work exceptionally well for them. They plan to use the sam
#You are required to help the manager to predict the right group of the new c
```

In [1055]:
```python
data=pd.read_csv("../files/customersegmentation.csv")
```

In [1056]:
```python
data.head()
```

Out[1056]:

	ID	Gender	Ever_Married	Age	Graduated	Profession	Work_Experience	Spending_Sc
0	458982	Male	Yes	61	Yes	Executive	1.0	l
1	458983	Female	Yes	63	Yes	Executive	0.0	l
2	458984	Male	Yes	39	Yes	Artist	0.0	Avei
3	458985	Male	No	23	No	Healthcare	1.0	
4	458986	Male	No	18	No	Healthcare	7.0	

In [1057]:
```python
data.shape
```

Out[1057]: (10695, 11)

In [1058]:
```
# Looking for missing values in dataset

data.isna().sum()
```

Out[1058]:
```
ID                   0
Gender               0
Ever_Married       190
Age                  0
Graduated          102
Profession         162
Work_Experience   1098
Spending_Score       0
Family_Size        448
Var_1              108
Segmentation         0
dtype: int64
```

In [1059]:
```
data = data.dropna()
data.shape
```

Out[1059]:
```
(8819, 11)
```

In [1060]:
```
data.isna().sum()
```

Out[1060]:
```
ID                 0
Gender             0
Ever_Married       0
Age                0
Graduated          0
Profession         0
Work_Experience    0
Spending_Score     0
Family_Size        0
Var_1              0
Segmentation       0
dtype: int64
```

Gender Data Visualisation

In [1061]:
```
data['Gender'].dtype
```

Out[1061]:
```
dtype('O')
```

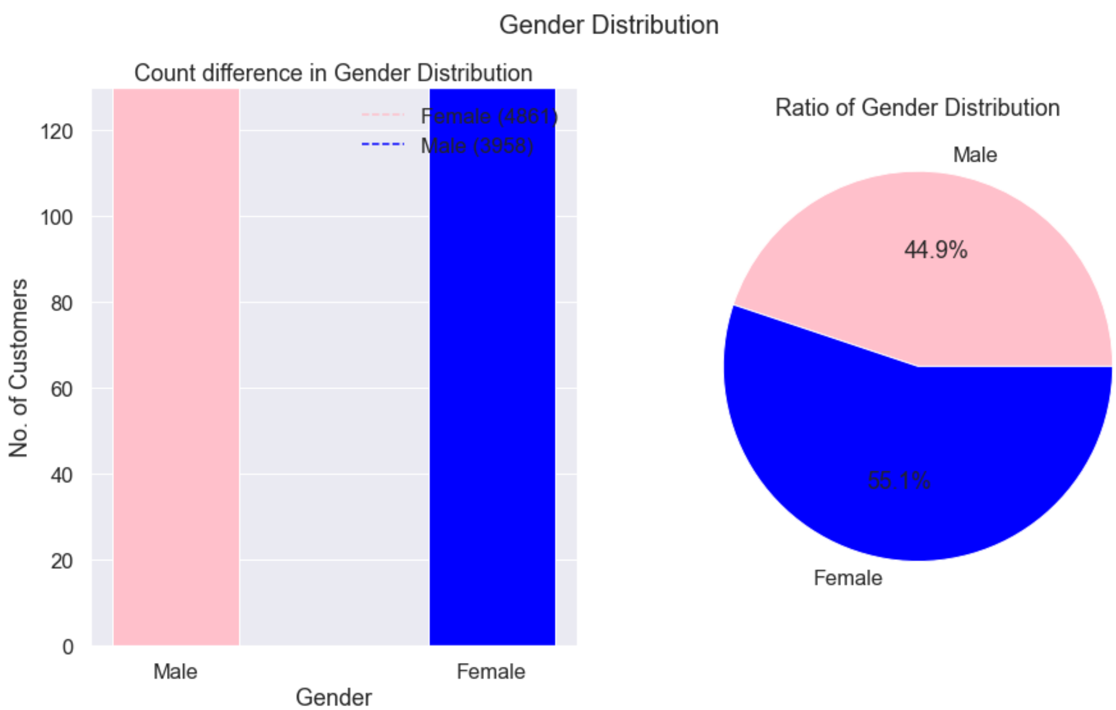
In [1062]:
```
data['Gender'].unique()
```

Out[1062]:
```
array(['Male', 'Female'], dtype=object)
```

In [1063]:
```
data['Gender'].value_counts()
```

Out[1063]:
```
Male      4861
Female    3958
Name: Gender, dtype: int64
```

In [1064]:
```
labels=data['Gender'].unique()
values=data['Gender'].value_counts(ascending=True)


fig, (ax0,ax1) = plt.subplots(ncols=2,figsize=(15,8))
bar = ax0.bar(x=labels, height=values, width=0.4, align='center', color=['pir
ax0.set(title='Count difference in Gender Distribution',xlabel='Gender', ylab
ax0.set_ylim(0,130)
ax0.axhline(y=data['Gender'].value_counts()[0], color='pink', linestyle='--',
ax0.axhline(y=data['Gender'].value_counts()[1], color='blue', linestyle='--',
ax0.legend()


ax1.pie(values,labels=labels,colors=['pink','blue'],autopct='%1.1f%%')
ax1.set(title='Ratio of Gender Distribution')
fig.suptitle('Gender Distribution', fontsize=20);
plt.show()
```



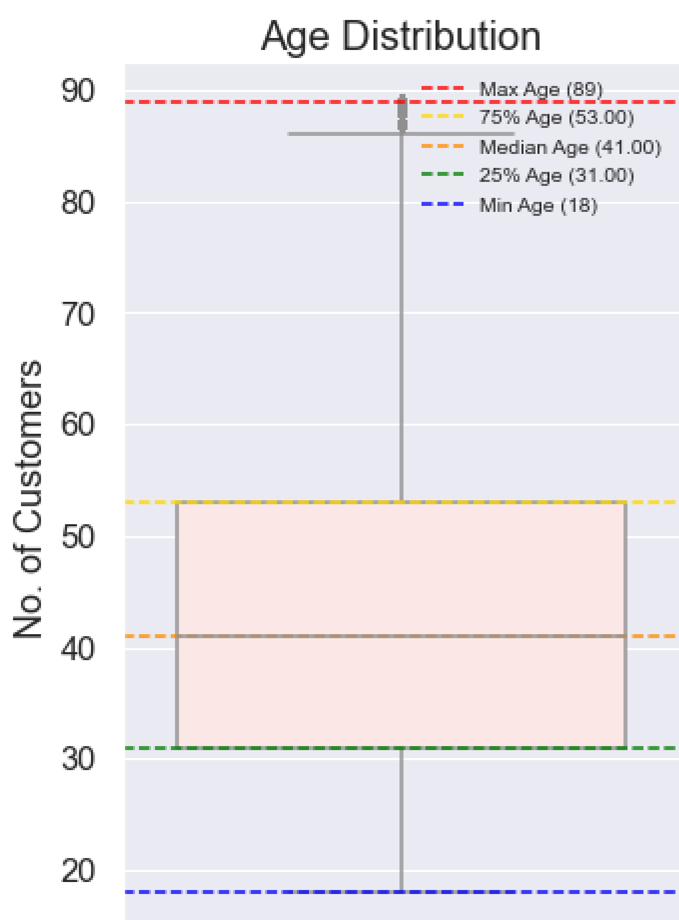
In [1065]:
```
data.Gender=pd.Categorical(data.Gender,categories=['Male','Female'],ordered=T
```

Age Data Visualisation

In [1066]:
```
data['Age'].describe()
```

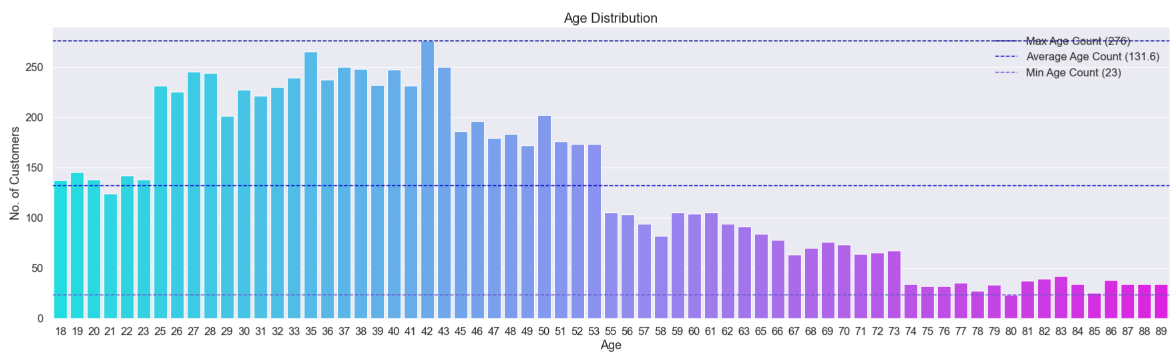
Out[1066]:
```
count    8819.000000
mean       43.517859
std        16.581537
min        18.000000
25%        31.000000
50%        41.000000
75%        53.000000
max        89.000000
Name: Age, dtype: float64
```

In [1067]:
```
fig, ax = plt.subplots(figsize=(5,8))
sns.set(font_scale=1.5)
ax = sns.boxplot(y=data["Age"], color="mistyrose")
ax.axhline(y=data['Age'].max(), linestyle='--',color='red', label=f'Max Age (
ax.axhline(y=data['Age'].describe()[6], linestyle='--',color='gold', label=f'
ax.axhline(y=data['Age'].median(), linestyle='--',color='darkorange', label=f
ax.axhline(y=data['Age'].describe()[4], linestyle='--',color='green', label=f
ax.axhline(y=data['Age'].min(), linestyle='--',color='blue', label=f'Min Age
ax.legend(fontsize='xx-small', loc='upper right')
ax.set_ylabel('No. of Customers')

plt.title('Age Distribution', fontsize = 20)
plt.show()
```



In [1068]:

```
fig, ax = plt.subplots(figsize=(30,8))
sns.set(font_scale=1.5)
ax = sns.countplot(x=data['Age'], palette='cool')
ax.axhline(y=data['Age'].value_counts().max(), linestyle='--',color='darkblue
ax.axhline(y=data['Age'].value_counts().mean(), linestyle='--',color='mediumb
ax.axhline(y=data['Age'].value_counts().min(), linestyle='--',color='slateblu
ax.legend(loc ='upper right')
ax.set_ylabel('No. of Customers')

plt.title('Age Distribution', fontsize = 20)
plt.show()
```



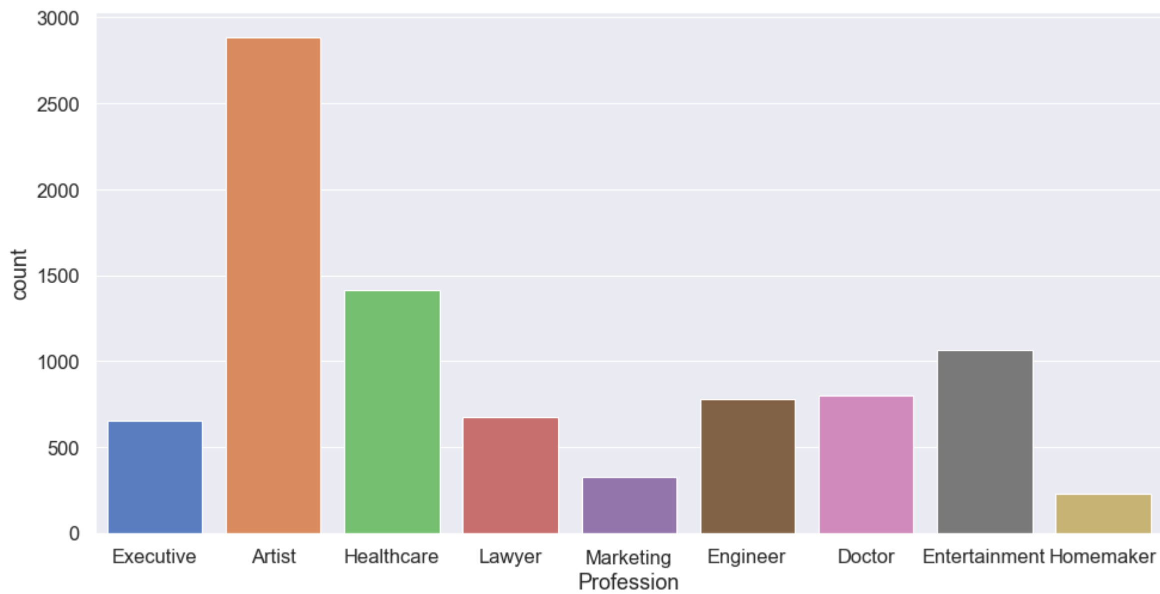
Profession Data Visualisation

In [1069]:

```
plt.figure(figsize=(16,8))
sns.countplot(data.Profession,palette='muted')
```

```
C:\Users\shrut\AppData\Local\Programs\Python\Python310\lib\site-packages\se
aborn\_decorators.py:36: FutureWarning: Pass the following variable as a ke
yword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

Out[1069]: <AxesSubplot:xlabel='Profession', ylabel='count'>



In [1070]:

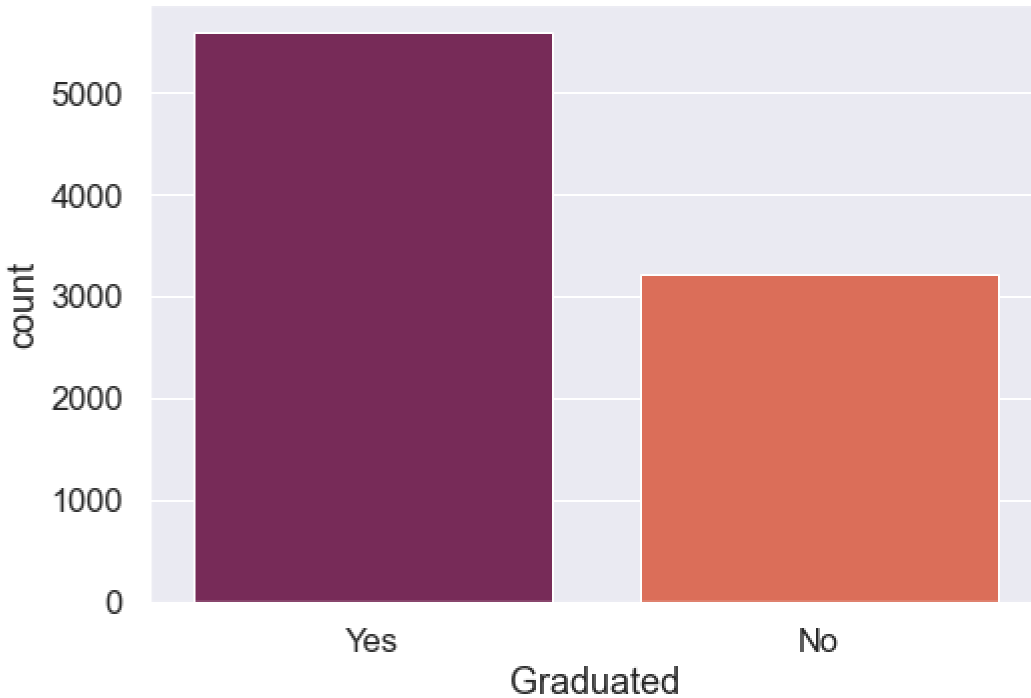
```
profession=pd.get_dummies(data.Profession)
data.drop(['Profession'],axis=1,inplace=True)
data=data.join(profession)
```

Graduated Data Visualisation

In [1071]:
```
sns.countplot(data.Graduated,palette='rocket')
```

```
C:\Users\shrut\AppData\Local\Programs\Python\Python310\lib\site-packages\se
aborn\_decorators.py:36: FutureWarning: Pass the following variable as a ke
yword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

Out[1071]: <AxesSubplot:xlabel='Graduated', ylabel='count'>



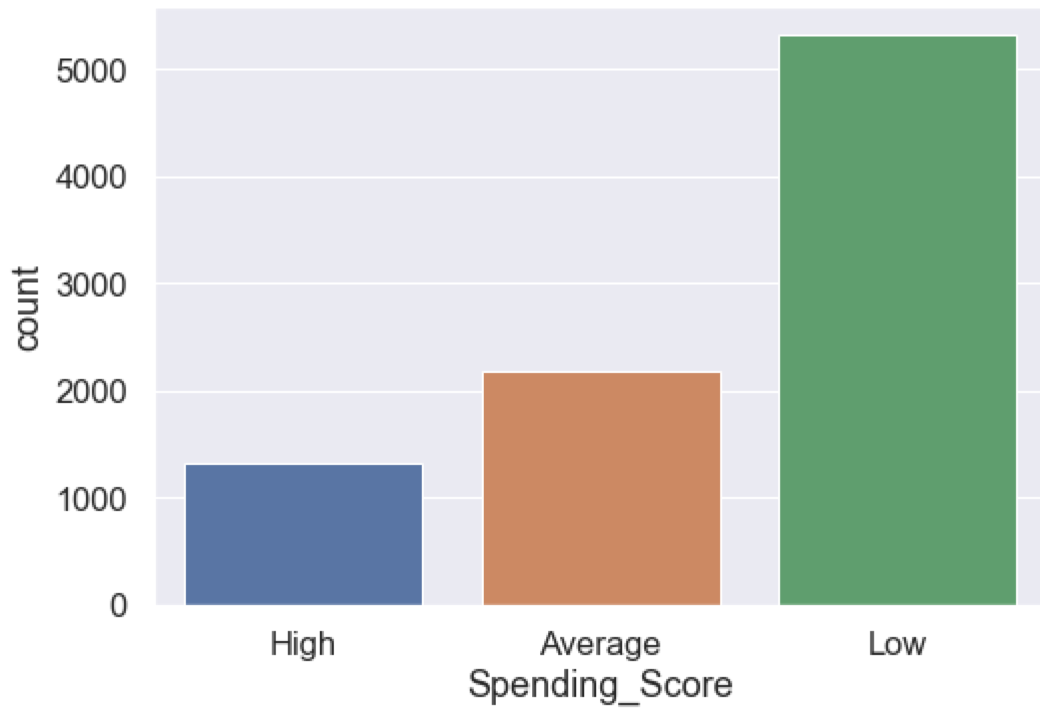
In [1072]:
```
data.Graduated=pd.Categorical(data.Graduated,categories=['No','Yes'],ordered=
```

Spending Score Visualisation

In [1073]:
```
sns.countplot(data.Spending_Score)
```

```
C:\Users\shrut\AppData\Local\Programs\Python\Python310\lib\site-packages\se
aborn\_decorators.py:36: FutureWarning: Pass the following variable as a ke
yword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

Out[1073]: `<AxesSubplot:xlabel='Spending_Score', ylabel='count'>`



In [1074]:
```
data.Spending_Score=pd.Categorical(data.Spending_Score,categories=['Low','Ave
```

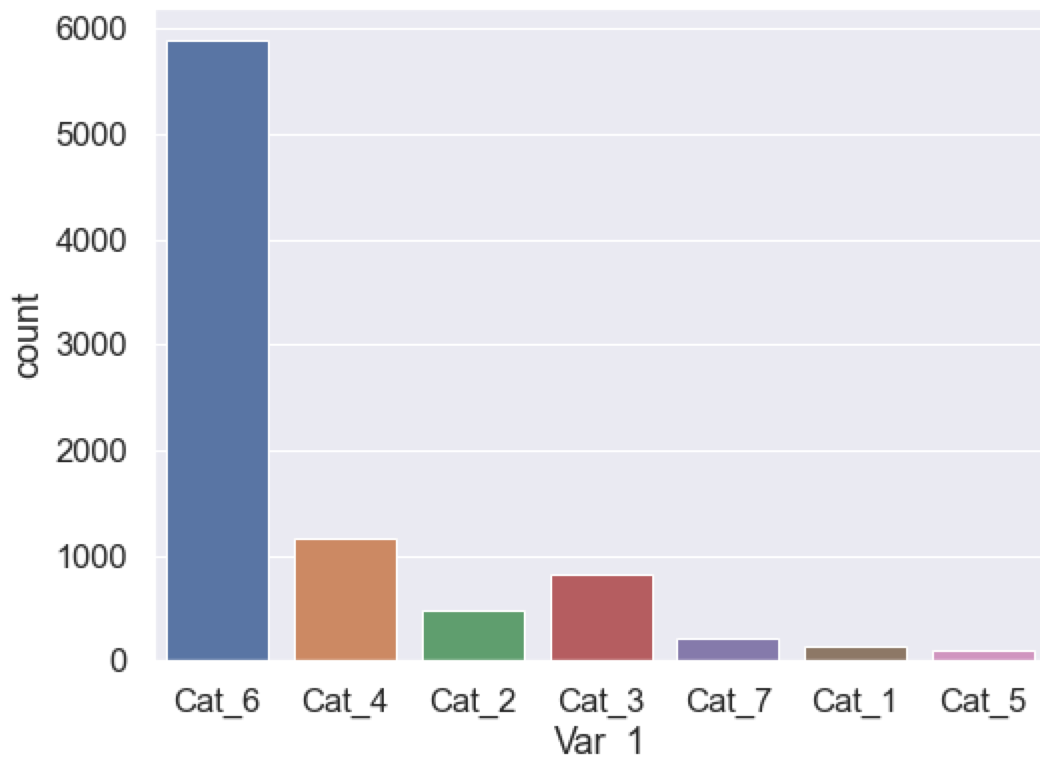
Var_1 Visualisation

In [1075]:

```
#Var_1 is income range attribute with cat_1 being the highest paid and cat_6

plt.figure(figsize=(8,6))
sns.countplot(data.Var_1)
```

```
C:\Users\shrut\AppData\Local\Programs\Python\Python310\lib\site-packages\se
aborn\_decorators.py:36: FutureWarning: Pass the following variable as a ke
yword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

Out[1075]: <AxesSubplot:xlabel='Var_1', ylabel='count'>



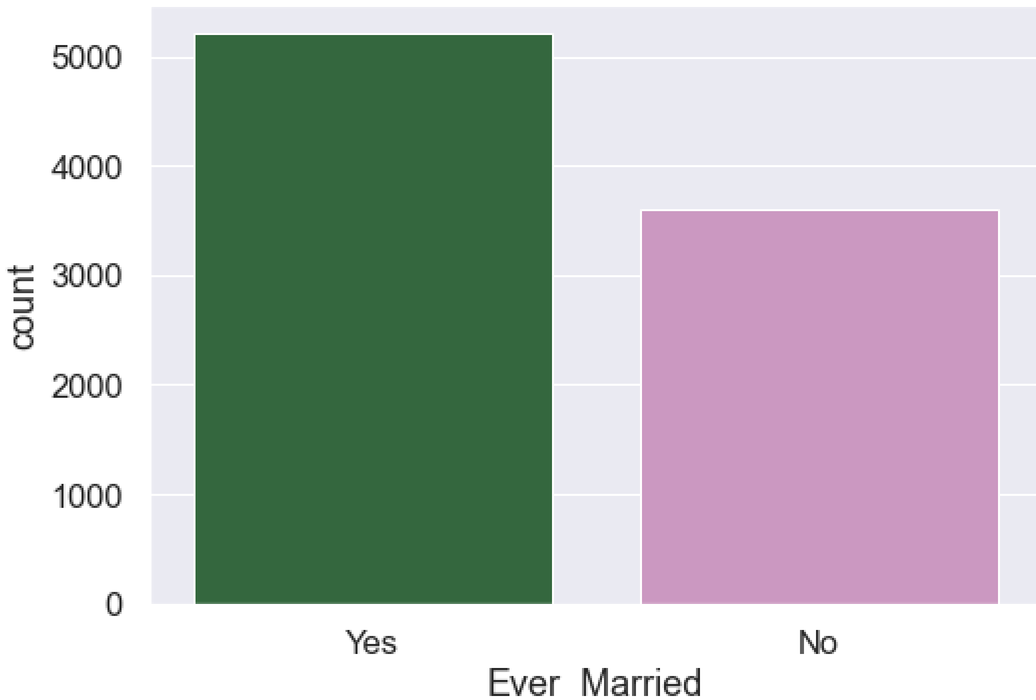
In [1076]:
```
data.Var_1=pd.Categorical(data.Var_1).codes
```

Marital Status Data Visualisation

In [1077]:
```
sns.countplot(data.Ever_Married,palette='cubehelix')
```

```
C:\Users\shrut\AppData\Local\Programs\Python\Python310\lib\site-packages\se
aborn\_decorators.py:36: FutureWarning: Pass the following variable as a ke
yword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result
in an error or misinterpretation.
  warnings.warn(
```

Out[1077]: <AxesSubplot:xlabel='Ever_Married', ylabel='count'>

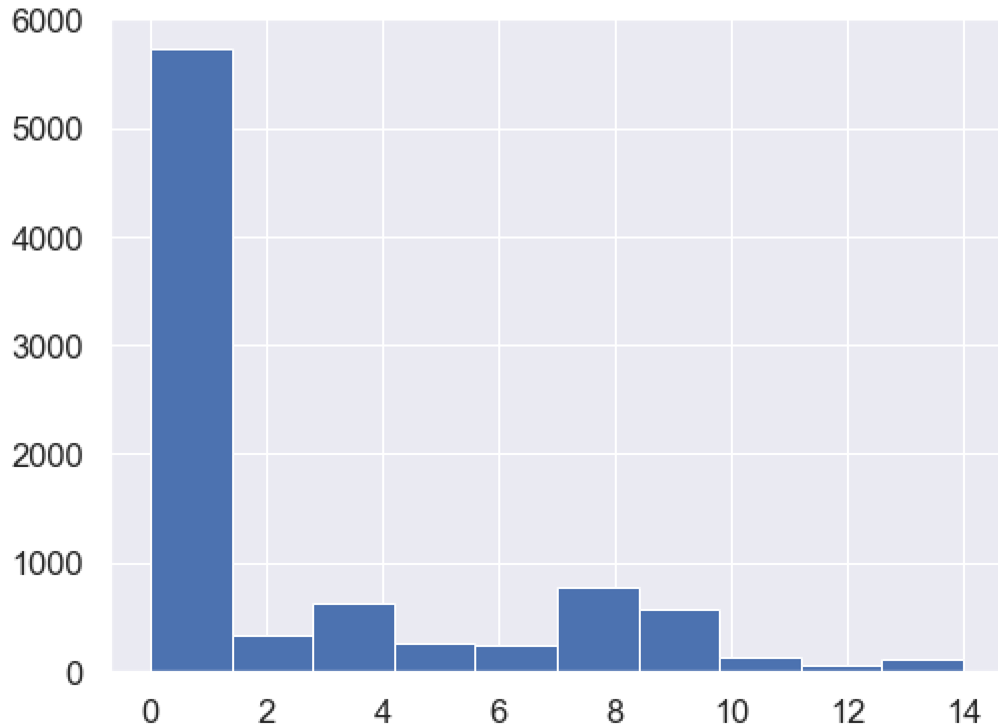


In [1078]:
```
data.Ever_Married=pd.Categorical(data.Ever_Married,categories=['No','Yes'],or
```

Work Experience Data Visualisation

In [1079]:
```
plt.figure(figsize=(8,6))
plt.hist(data.Work_Experience)
```

Out[1079]:
```
(array([5729.,  337.,  631.,  248.,  242.,  771.,  569.,  121.,   55.,
         116.]),
 array([ 0. ,  1.4,  2.8,  4.2,  5.6,  7. ,  8.4,  9.8, 11.2, 12.6, 14. ]),
 <BarContainer object of 10 artists>)
```



label=pd.Categorical(data.Segmentation,categories=['A','B','C','D']).codes
data.drop(['Segmentation'],axis=1,inplace=True) label

In [1080]:

```
correlation_data=pd.DataFrame(data)
correlation_data.drop(['ID'],axis=1,inplace=True)
correlation_data
```

Out[1080]:

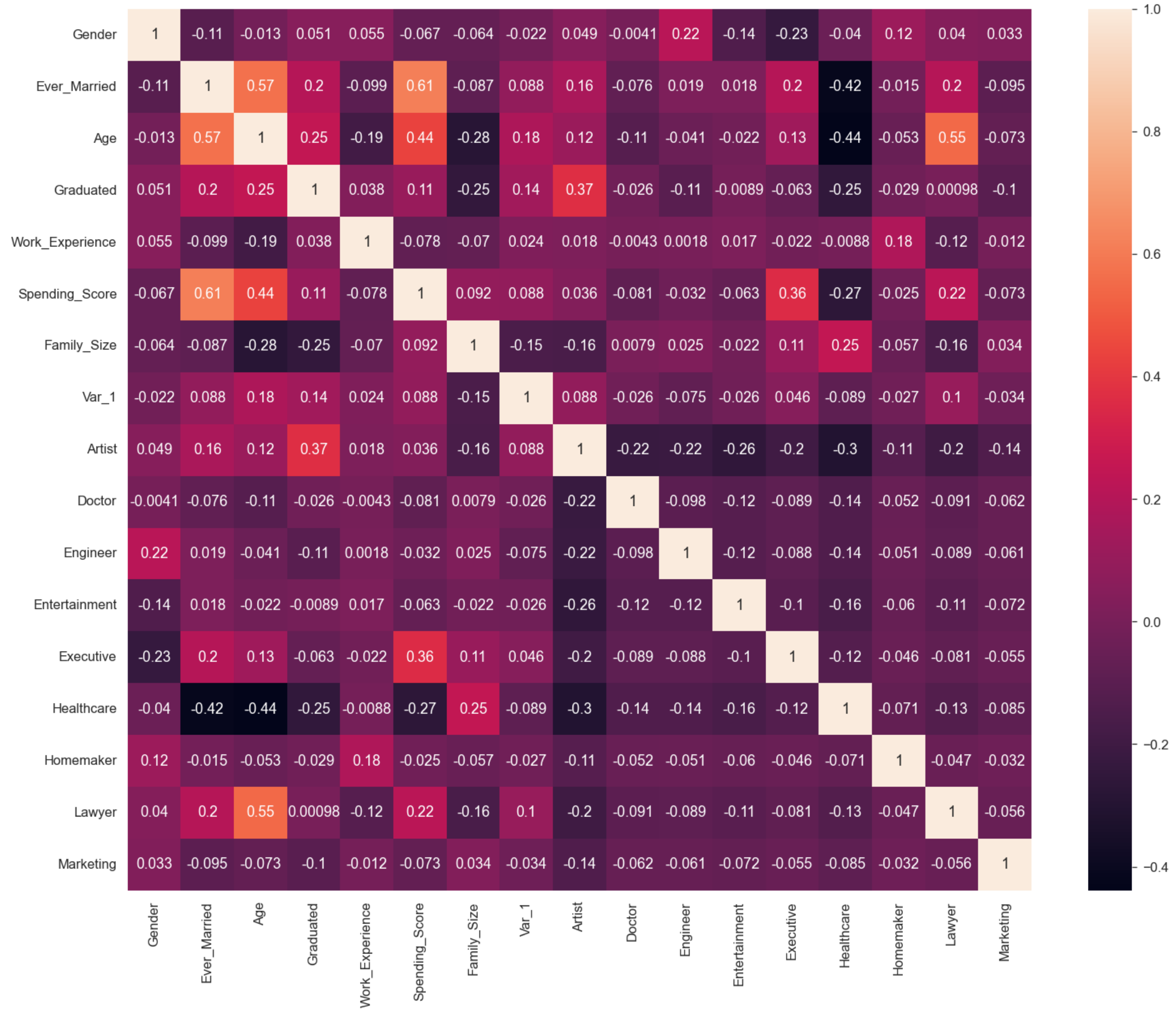
	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size
0	0	1	61	1	1.0	2	3.0
1	1	1	63	1	0.0	2	5.0
2	0	1	39	1	0.0	1	3.0
3	0	0	23	0	1.0	0	4.0
4	0	0	18	0	7.0	0	4.0
...	...	...	...	...	...	...	..
10690	1	1	43	1	0.0	1	2.0
10691	1	0	31	1	1.0	0	4.0
10692	0	0	22	0	1.0	0	3.0
10693	1	1	66	1	0.0	1	3.0
10694	1	0	43	1	1.0	0	1.0

8819 rows × 18 columns

In [1081]:

```
plt.figure(figsize=(25,20))
sns.heatmap(correlation_data.corr(),annot=True)
```

Out[1081]: <AxesSubplot:>



K-means Algorithm and Analysis

** Authored by: Shruti Chanda

Calculate the z-score and remove the outliers from the dataset.

In [1082]:
```
# Calculate z-score from the correlation except the
z_scores = zscore(correlation_data.drop('Segmentation', axis=1))

# Filtering rows with zscore less than 3
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
new_df = correlation_data[filtered_entries]
```

In [1083]:
```
new_df
```

Out[1083]:

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size
2	0	1	39	1	0.0	1	3.0
3	0	0	23	0	1.0	0	4.0
4	0	0	18	0	7.0	0	4.0
13	0	1	38	1	8.0	1	4.0
14	0	1	37	1	8.0	1	4.0
...	...	...	...	...	...	...	..
10689	1	0	43	1	9.0	0	3.0
10690	1	1	43	1	0.0	1	2.0
10691	1	0	31	1	1.0	0	4.0
10692	0	0	22	0	1.0	0	3.0
10694	1	0	43	1	1.0	0	1.0

5236 rows × 18 columns

K-means Algorithm Class

In [1084]:

```
# Generic class to fit and predict k-means clustering model
## k: the number of clusters desired
## tol: value helps identify model convergence
## max_iter: maximum number of iterations incase, model does not converges

class k_means:
    # Intitialize model parameters
    def __init__(self, k=2, tol=0.001, max_iter=300):
        self.k = k
        self.tol = tol
        self.max_iter = max_iter

    # Describes the relationship between a response variable and one or more
    def model_fit(self, data):
        # Create an empty liost for centroids
        self.centroids = {}

        # Randomly select k points to begin
        for i in range(self.k):
            self.centroids[i] = data[np.random.choice(range(len(data)), 1, re

        # Use initial centroids to label data rows to various k vales, then c
        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []

            for features in data:
                distances = [math.sqrt(np.linalg.norm(features-self.centroids
                classification = distances.index(min(distances))
                self.classifications[classification].append(features)

            prev_centroids = dict(self.centroids)

            for classification in self.classifications:
                self.centroids[classification] = np.average(self.classificati

            opt = True

            for c in self.centroids:
                orig_centroid = prev_centroids[c]
                curr_centroid = self.centroids[c]
                if np.sum((curr_centroid-orig_centroid)/orig_centroid*100.0)
                    opt = False

            if opt:
                return opt

    # Used to predict outcomes by analyzing patterns in a given set of input
    def model_predict(self, data):
        classification = []

        # Returns the classification list
        for d_row in range(len(data)):
            distances = [math.sqrt(np.linalg.norm(data[d_row]-self.centroids[
```

```
            classification.append(distances.index(min(distances)))

        return classification
```

Plotting functions

In [1085]:

```
# Graph plotting function for k-means clustering which can be extended to k=1
def plot_graph (centroids_data, classification_data, k=2):
    color_lst = mcp.gen_color(cmap="cividis", n=k)
    mark_lst = ['x', '^', '>', '<', '8', 's', 'p', 'h', 'H', 'd', 'D']

    fig = plt.figure(figsize=(20, 20))
    ax = fig.add_subplot(111, projection='3d')

    # Plot cluster centroids
    for centroid in centroids_data:
        ax.scatter(centroids_data[centroid][0], centroids_data[centroid][1],
                marker="o", color="blue", s=150, linewidths=5)

    # Plot classification data
    for classification in classification_data:
        color = color_lst[classification]
        mark =  mark_lst[classification]
        for featureset in classification_data[classification]:
                ax.scatter(featureset[0], featureset[1], featureset[2], marke

    plt.show()

# Plotting cost functions obtained to analyze optimal value of k
def plot_costs(costs, trials=2):
    x = np.arange(2,trials)
    plt.plot(x,costs)
    plt.title("Elbow curve")
    plt.xlabel("K -->")
    plt.ylabel("Dispersion")

# Plotting the silhoutte plot to analyze best value of k
def plot_silhoutte(scores, trials=2):
    x = np.arange(2,trials)
    plt.plot(x,scores)
    plt.title("Silhoutte Score for k's")
    plt.xlabel("K -->")
    plt.ylabel("Score")
```

Independent Helper Functions

In [1086]:

```
# Generic function to compoute Principal Component Analysis (PCA) for dimensi
def principalComponentAnalysis(data, n_components):

    # Mean centering the data
    X_mean = data - np.mean(data , axis = 0)

    # Calculating the covariance matrix of the mean-centered data.
    cov_mat = np.cov(X_mean , rowvar = False)

    # Calculating Eigenvalues and Eigenvectors of the covariance matrix
    eigen_vals , eigen_vecs = np.linalg.eigh(cov_mat)

    # Sort the eigenvalues in descending order
    sorted_vec = np.argsort(eigen_vals)[::-1]

    sorted_eigenvalue = eigen_vals[sorted_vec]

    # Similarly sort the eigenvectors
    sorted_eigenvectors = eigen_vecs[:,sorted_vec]

    # Select the first n eigenvectors, n is desired dimension of our final re
    eigenvector_subset = sorted_eigenvectors[:, 0:n_components]

    # Transform the data
    X_reduced = np.dot(eigenvector_subset.transpose(),X_mean.transpose()).tra

    return X_reduced

# Calculating cost function for various values of k
def cost_function(data, trials=1):
    costs = []
    scores = []
    # Run the loop for the number of trials
    for i in range(2,trials):
        # Initialize K means with different values of k
        kmeans = k_means(k=i)
        kmeans.model_fit(data)

        cluster_assignments = kmeans.centroids

        # Calculate the distance from their respective centroides for evaluat
        cost = 0
        for cluster in cluster_assignments:
            for feature in kmeans.classifications[cluster]:
                dist = np.linalg.norm(feature - cluster_assignments[cluster])
                cost += dist
        costs.append(np.array(cost))

        # Calculate the silhoutte score
        scores.append(silhouette_score(data, kmeans.model_predict(data), metr

    # Return cost and scores arrays
    return costs, scores

# Split a dataset into a train and test set
def train_test_split(df, frac=0.2):
```

```
    # get random sample
    test = df.sample(frac=frac, axis=0)

    # get everything but the test sample
    train = df.drop(index=test.index)

    return train, test
```

Data Preparation

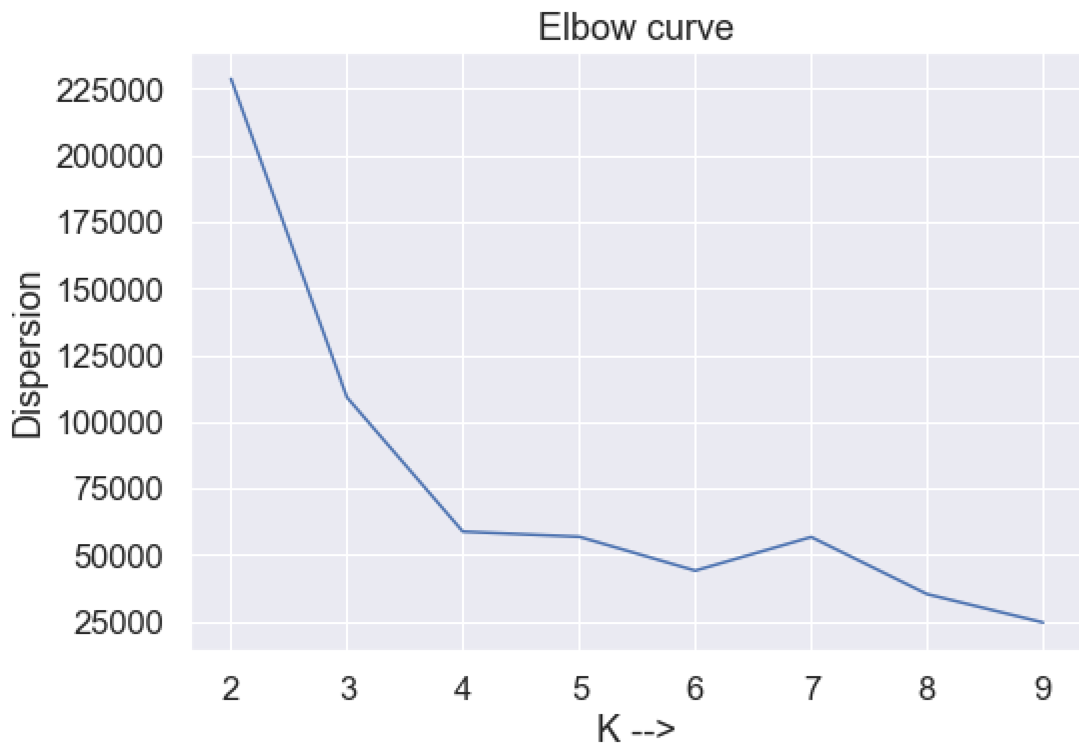
In [1087]:
```
# Split data into training and testing data
train, test = train_test_split(new_df, frac=0.6)
```

In [1088]:
```
# Apply model fit on the training data
train_x = train.drop('Segmentation', axis=1)
train_x = train[['Age', 'Graduated', 'Work_Experience', 'Spending_Score']].va
train_y = train.Segmentation
train_y = pd.Categorical(train_y,categories=['A','B','C','D'],ordered=True).c
```

Analyze for best value of k

In [1098]:

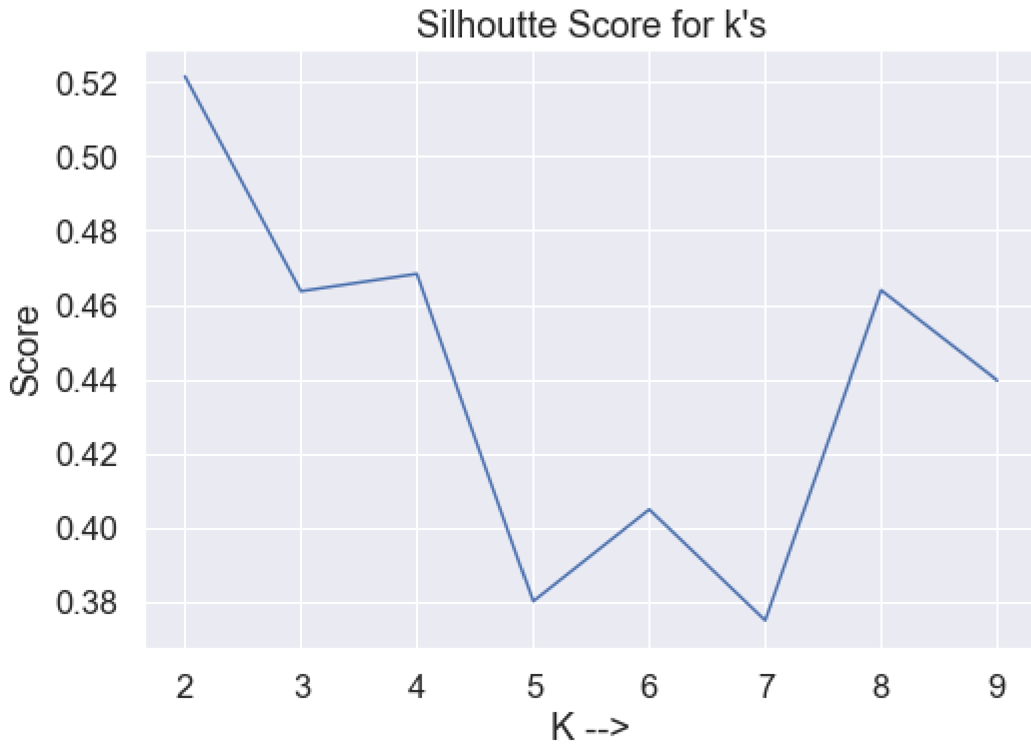
```
# Implement PCA since we have 4 features
updated_x = principalComponentAnalysis(train_x, 3)

cost_arr, scores_arr = cost_function(updated_x, trials=10)

# Generate cost plot for various values of k
plot_costs(cost_arr, trials=10)
```



Using the Elbow curve we can see that the optimal value for k is 4 and can try to vary it between 4 to 6.

In [1099]:

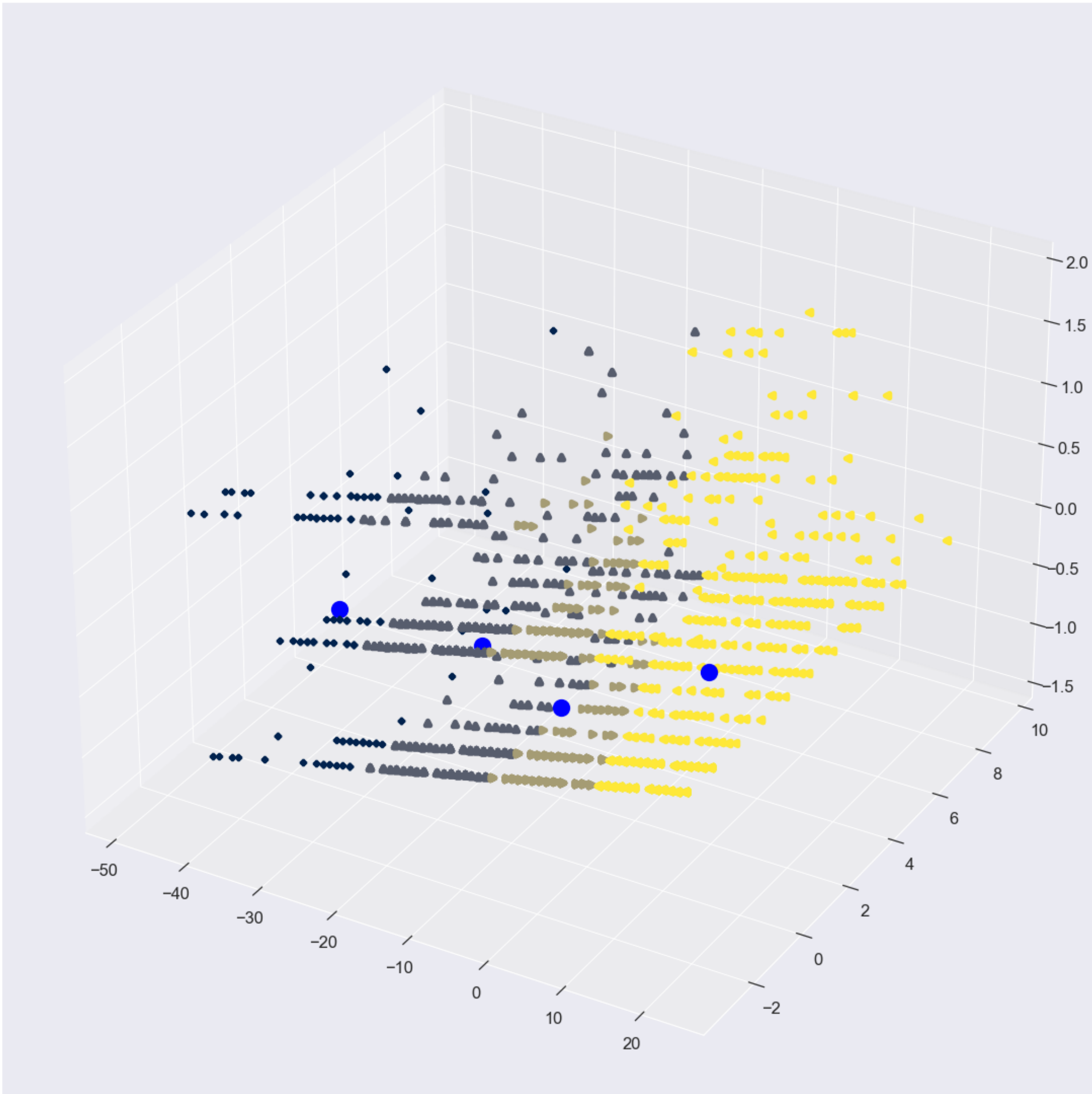
```
# Generate silhoutte plot for various values of k
plot_silhoutte(scores_arr, trials=10)
```



Using the Silhoutte score we can observe that we get the highest score with k=4.

In [1104]:

```
# Apply model fit on the training data using the optimal value of k as 4
train_kmeans = k_means(k=4)
train_kmeans.model_fit(updated_x)

# Generate k-means graph for identified clusters
plot_graph(train_kmeans.centroids, train_kmeans.classifications, k=4)
```



In [1105]:

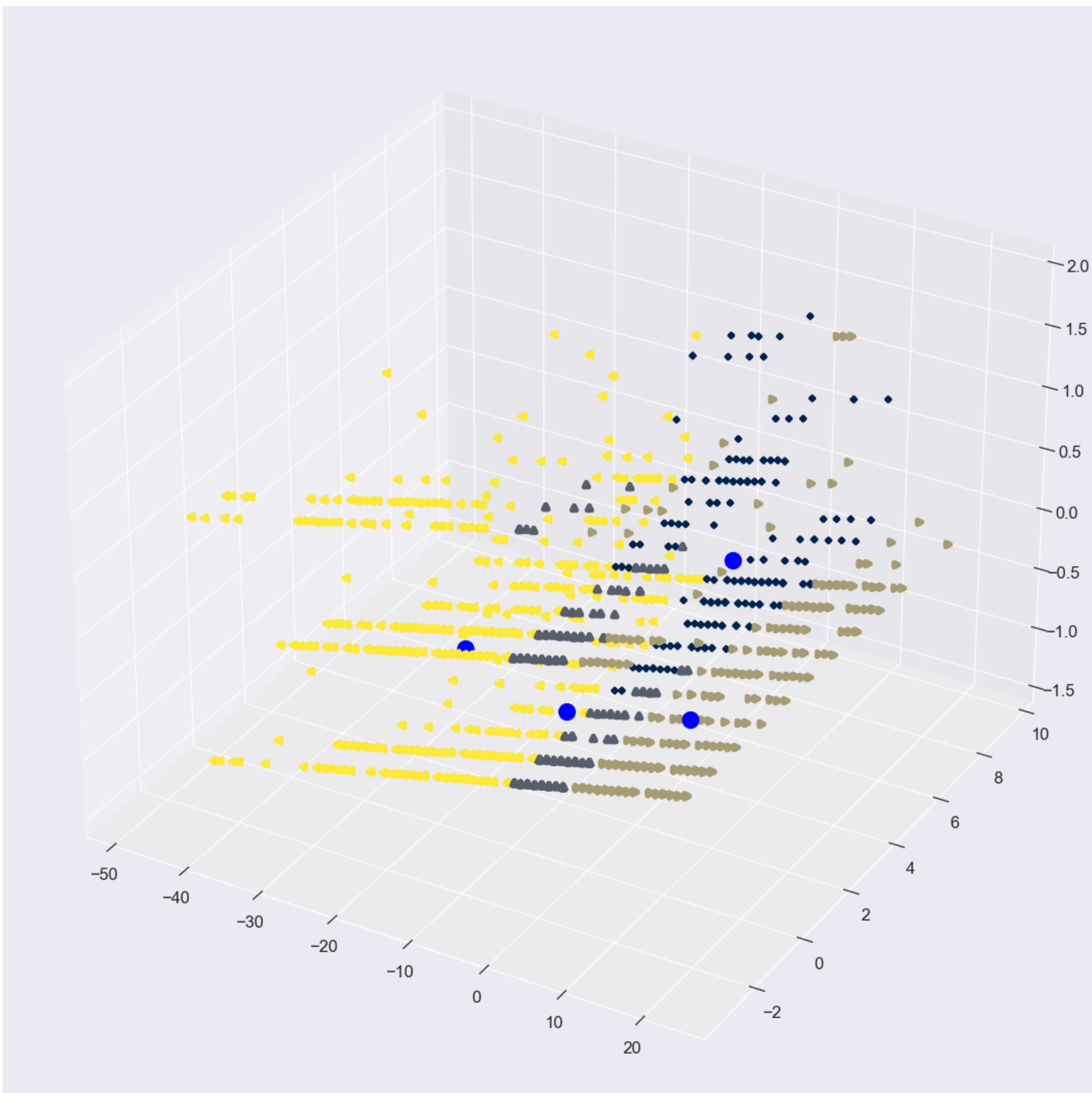
```
# Predict cluster label outcomes
out_y = train_kmeans.model_predict(updated_x)

# Analyze the model performance
score = silhouette_score(updated_x, out_y, metric='euclidean')

print(f'Silhoutte Score: {score}%')
```

```
Silhoutte Score: 0.4531733863745631%
```

Varying parameters to optimize model performance

In [1106]:

```
# Apply model fit on the training data
train_x = train.drop('Segmentation', axis=1)
# Reduce the number of deciding columns
train_x = train[['Age','Work_Experience', 'Spending_Score']].values
train_y = train.Segmentation
train_y = pd.Categorical(train_y,categories=['A','B','C','D'],ordered=True).c

# Apply PCA
updated_x = principalComponentAnalysis(train_x, 3)

# Reduce the number of tolerence
train_kmeans = k_means(k=4, tol=0.01)
train_kmeans.model_fit(updated_x)

# Visualize identified k-means clusters
plot_graph(train_kmeans.centroids, train_kmeans.classifications, k=4)
```



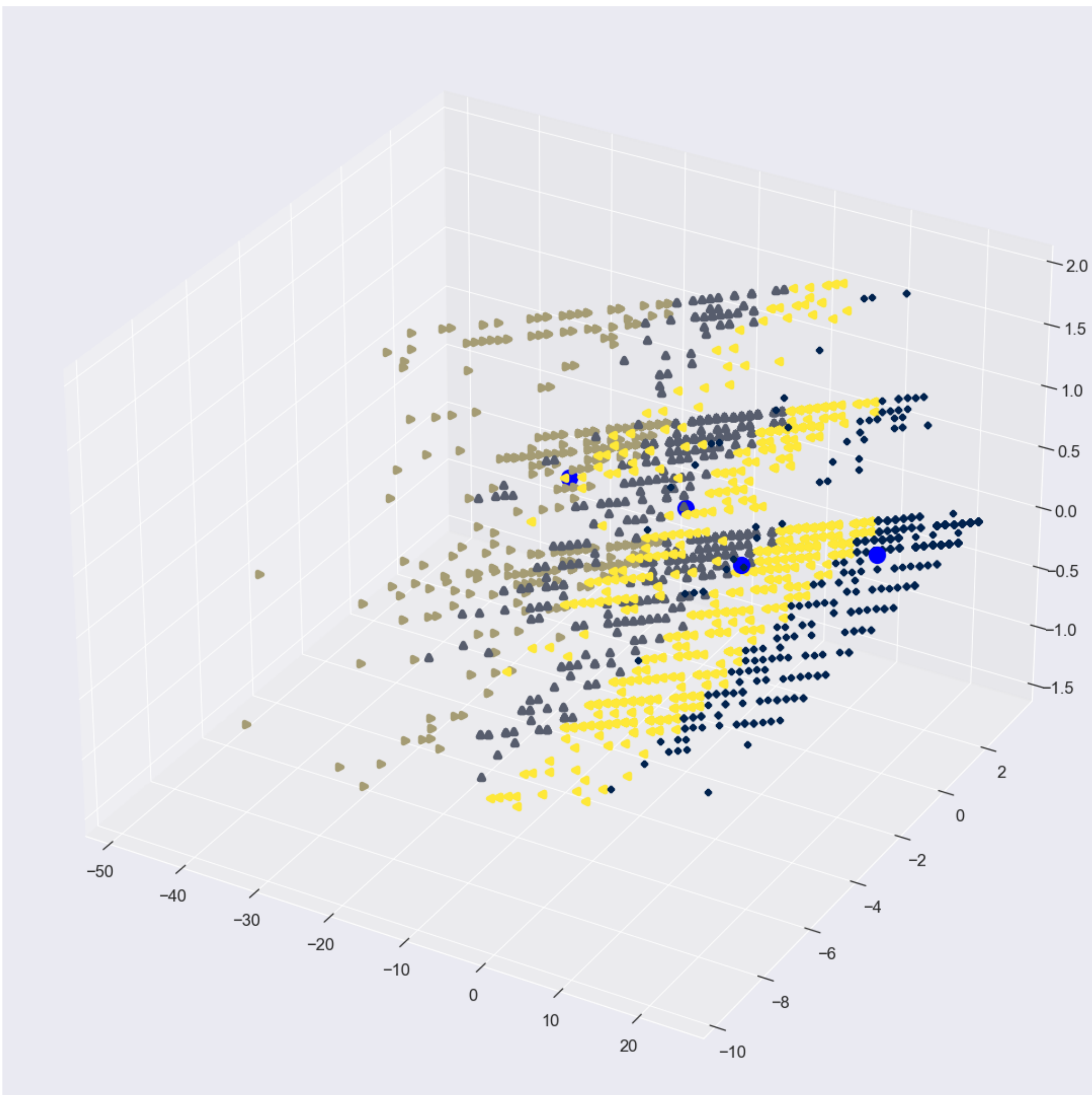
In [1107]:
```
# Predict k-means labels
out_y = train_kmeans.model_predict(updated_x)

# Calculate the silhoutte score
score = silhouette_score(updated_x, out_y, metric='euclidean')

print(f'Silhoutte Score: {score}%')
```

```
Silhoutte Score: 0.4204211821081394%
```

Apply model to test data

In [1108]:

```
# Prepare data for applying model

test_x = test.drop('Segmentation', axis=1)
test_x = test[['Age', 'Graduated', 'Work_Experience', 'Spending_Score']].valu
test_y = test.Segmentation
test_y = pd.Categorical(test_y,categories=['A','B','C','D'],ordered=True).coc

# Apply PCA to reduce dimensionality
updated_test_x = principalComponentAnalysis(test_x, 3)

# Apply model fit on the testing data
test_kmeans = k_means(k=4)
test_kmeans.model_fit(updated_test_x)

# Visulaize the identified k-means clusters
plot_graph(test_kmeans.centroids, test_kmeans.classifications, k=4)
```



In [1109]:

```
# Predict k-means labels
out_y = test_kmeans.model_predict(updated_test_x)

# Generate the silhoutte score
score = silhouette_score(updated_test_x, out_y, metric='euclidean')

print(f'Silhoutte Score: {score}%')
```

```
Silhoutte Score: 0.4486475161692587%
```

In [ ]:

DBScan

Author: Jinrong

In [352]:

```
data.head()
```

Out[352]:

er	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Var_1	Segm
0	1	61	1	1.0	2	3.0	5	
1	1	63	1	0.0	2	5.0	5	
0	1	39	1	0.0	1	3.0	5	
0	0	23	0	1.0	0	4.0	5	
0	0	18	0	7.0	0	4.0	5	

In [353]:

```
df_clean = data.drop(columns=['ID'])
df_clean.head()
```

Out[353]:

Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Var_1
0	1	61	1	1.0	2	3.0	5
1	1	63	1	0.0	2	5.0	5
0	1	39	1	0.0	1	3.0	5
0	0	23	0	1.0	0	4.0	5
0	0	18	0	7.0	0	4.0	5

In [354]:
```
df_clean.dtypes
```

Out[354]:
```
Gender                 int8
Ever_Married           int8
Age                   int64
Graduated              int8
Work_Experience     float64
Spending_Score         int8
Family_Size         float64
Var_1                  int8
Segmentation         object
Artist                uint8
Doctor                uint8
Engineer              uint8
Entertainment         uint8
Executive             uint8
Healthcare            uint8
Homemaker             uint8
Lawyer                uint8
Marketing             uint8
dtype: object
```

In [355]:
```
cat_cols = df_clean.select_dtypes('object').columns
num_cols = df_clean.select_dtypes('float64').columns
df_clean[cat_cols] = df_clean[cat_cols].apply(lambda x: x.fillna(x.value_cour
df_clean[num_cols] = df_clean[num_cols].apply(lambda x: x.fillna(x.mean()))
```

In [356]:
```
df_clean.isna().mean()
```

Out[356]:
```
Gender             0.0
Ever_Married       0.0
Age                0.0
Graduated          0.0
Work_Experience    0.0
Spending_Score     0.0
Family_Size        0.0
Var_1              0.0
Segmentation       0.0
Artist             0.0
Doctor             0.0
Engineer           0.0
Entertainment      0.0
Executive          0.0
Healthcare         0.0
Homemaker          0.0
Lawyer             0.0
Marketing          0.0
dtype: float64
```

In [357]:

```
df_dummy = pd.get_dummies(df_clean.drop(columns='Segmentation'), drop_first=T
df_dummy.head()
```

Out[357]:

	Gender	Ever_Married	Age	Graduated	Work_Experience	Spending_Score	Family_Size	Va
0	0	1	61	1	1.0	2	3.0	
1	1	1	63	1	0.0	2	5.0	
2	0	1	39	1	0.0	1	3.0	
3	0	0	23	0	1.0	0	4.0	
4	0	0	18	0	7.0	0	4.0	

In [358]:

```
class DBScan():
    def __init__(self, eps, min_samples=5):
        self.eps = eps # maximum distance between neighbor
        self.min_samples = min_samples # minimum number of neighbors

    # this function calcualte distance between all pairs
    def find_distances(self, X):
        return ((X[:,None,:] - X[None,:,:])**2).sum(axis=2)**0.5

    # return the indices of neighbors for specific observation
    def find_neightbors(self, i):
        return set(np.where(self.D[i] < self.eps)[0])

    # find clusters
    def fit(self, X):

        # calculate all pairwise distances
        self.D = self.find_distances(X)

        m = X.shape[0]
        labels_ = [-1]*m
        c = -1

        # go over each observation
        for i in range(m):

            # if it is undefined
            if labels_[i] == -1:

                # find all neighbors of the current node
                neighbors = self.find_neightbors(i)

                # if there are few than min_samples neighbors
                # label the node as noise
                if len(neighbors) < self.min_samples:
                    labels_[i] = -2
                    continue
                c += 1

                # assgin it to a new cluster
                labels_[i] = c

                # label all neighbors to the same cluster
                for x in neighbors:
                    labels_[x] = c

                # go over each neighbor
                while neighbors:
                    q = neighbors.pop()
                    labels_[q] = c

                    # find neighbor of neighbors
                    new = self.find_neightbors(q)
                    # if new neighbor are within the limit
                    if len(new) > self.min_samples:
                        # add new neighbors to the list
```

```
                        for x in new:
                            if x not in neighbors and labels_[x] in {-1,-2}:
                                neighbors.add(x)
        # store the final results
        self.labels_ = labels_
```

In [359]:
```
sc = StandardScaler()

X_scaled = df_dummy.copy()
X_scaled[:] = sc.fit_transform(df_dummy)
```

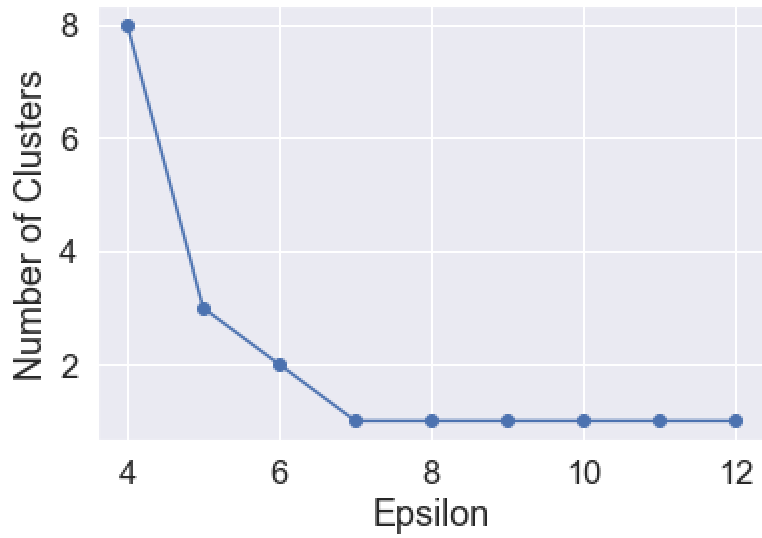
In [360]:
```
eps_val = np.arange(4, 13, 1)
n_cluster = []
for eps in eps_val:
    print(eps)
    dbscan = DBScan(eps=eps)
    dbscan.fit(X_scaled.values)
    n_cluster.append(len(set(dbscan.labels_)))
```

```
4
5
6
7
8
9
10
11
12
```

In [361]:
```
print(eps_val)
print(n_cluster)
```

```
[ 4  5  6  7  8  9 10 11 12]
[8, 3, 2, 1, 1, 1, 1, 1, 1]
```

In [362]:
```
plt.plot(eps_val, n_cluster, 'bo-')
plt.xlabel('Epsilon')
plt.ylabel('Number of Clusters');
```



In [363]:
```
dbscan = DBScan(eps=4.35)
dbscan.fit(X_scaled.values)
```

In [364]:
```
pd.Series(dbscan.labels_).value_counts()
```

Out[364]:
```
1    7613
0     652
2     325
3     229
dtype: int64
```

In [365]:
```
pd.crosstab(np.array(dbscan.labels_), data['Segmentation'])
```

Out[365]:

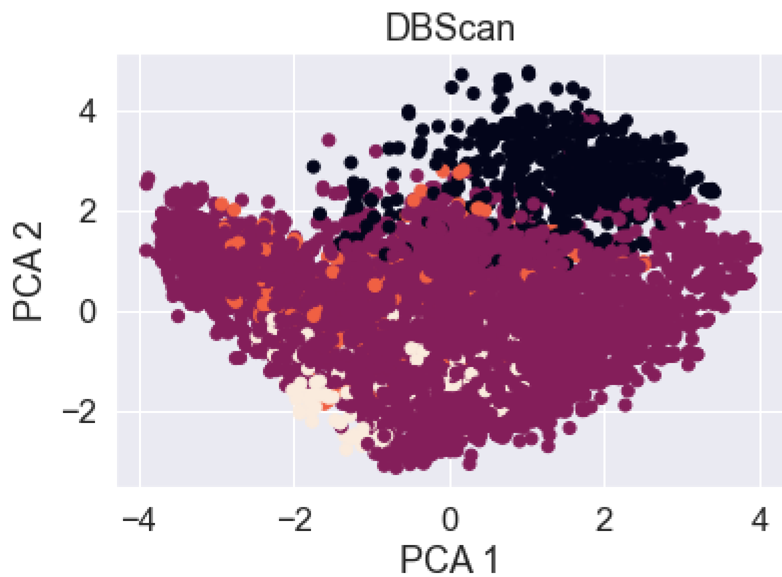
Segmentation	A	B	C	D
row_0				
0	245	149	164	94
1	3285	1355	1508	1465
2	138	24	29	134
3	102	44	19	64

In [366]:
```
adjusted_rand_score(data['Segmentation'], dbscan.labels_)
```

Out[366]:
```
0.004704905550590988
```

In [367]:
```
X_pca = pca.fit_transform(X_scaled)
```

In [368]:
```
plt.scatter(X_pca[:,0], X_pca[:,1], c=dbscan.labels_)
plt.title('DBScan')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2');
```



In [ ]:

In [ ]:

KNN Algorithm

In [118]:
```
# apply model fit on the training data
X = correlation_data.drop('Segmentation', axis=1)
X = correlation_data[['Age', 'Graduated', 'Work_Ex', 'Spending_Score']]
X
```

Out[118]:

	Age	Graduated	Work_Ex	Spending_Score
0	61	1	1.0	2
1	63	1	0.0	2
2	39	1	0.0	1
3	23	0	1.0	0
4	18	0	7.0	0
...	...	...	...	...
10690	43	1	0.0	1
10691	31	1	1.0	0
10692	22	0	1.0	0
10693	66	1	0.0	1
10694	43	1	1.0	0

8819 rows × 4 columns

In [105]:
```
# #cols = ['Segmentation', 'Artist', 'Doctor', 'Engineer', 'Entertainment', '
# X = correlation_data.drop(['Segmentation', 'Artist', 'Doctor', 'Engineer',
# X
```

In [120]:
```
y = correlation_data['Segmentation']
y
```

Out[120]:
```
0        2
1        2
2        2
3        3
4        3
        ..
10690    2
10691    3
10692    3
10693    0
10694    1
Name: Segmentation, Length: 8819, dtype: int8
```

In [107]:
```
# #plt.scatter(X[:,0], X[:,1], marker="o", c=y, s=100, cmap="plasma")
# import matplotlib.pyplot as plt
# ax.scatter(correlation_data[2], correlation_data[3], correlation_data[4], m

# plt.show()
```

In [121]:
```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ra
```

In [122]:
```
X_train, X_test, y_train, y_test
```

Out[122]:
```
(         Age  Graduated  Work_Ex  Spending_Score
 10031     50          0      1.0               1
 607       46          1      7.0               0
 7031      35          0      1.0               0
 4774      37          1      8.0               1
 10203     58          1      0.0               1
 ...      ...        ...      ...             ...
 5440      67          1      9.0               0
 9546      49          1      1.0               2
 6000      32          0      0.0               0
 4017      79          1      1.0               2
 3397      42          1      0.0               1

 [7055 rows x 4 columns],
          Age  Graduated  Work_Ex  Spending_Score
 6433      65          1      0.0               1
 455       53          1      2.0               0
 9929      73          0      0.0               2
 4105      29          0      5.0               0
 9748      43          1      1.0               0
 ...      ...        ...      ...             ...
 27        47          1      0.0               2
 8101      52          1      1.0               0
 8615      56          1      0.0               2
 681       35          1      9.0               0
 10642     42          1      1.0               1

 [1764 rows x 4 columns],
 10031     0
 607       0
 7031      0
 4774      2
 10203     2
          ..
 5440      1
 9546      2
 6000      1
 4017      1
 3397      2
 Name: Segmentation, Length: 7055, dtype: int8,
 6433      1
 455       0
 9929      2
 4105      0
 9748      0
          ..
 27        0
 8101      2
 8615      1
 681       3
 10642     2
 Name: Segmentation, Length: 1764, dtype: int8)
```

In [123]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[123]: ((7055, 4), (1764, 4))

In [124]:
```
# Changing the index of the records into range
X_train.index=range(len(X_train))
y_train.index=range(len(X_train))
X_test.index=range(len(X_test))
y_test.index=range(len(y_test))
```

K nearest neighbours by Sorting the Euclidean distance

In [125]:
```
def nearest_Neighbours(X_train,y_train,X_test,K):
    dist=[]
    for i in range(len(X_train)):
        #initialize distance = 0
        euclidean_Dist=0
        for j in range(len(X_train.columns)):
                #sum of total distance
                euclidean_Dist+=round(np.sqrt(pow((X_train.iloc[i,j]-X_test[j
        dist.append((euclidean_Dist,i,y_train.iloc[i]))
        #sort in ascending order of distance & select top k nearest distances
        dist = sorted(dist, key=lambda z: z[0])[0:K]
    return dist
```

Predicting the new data point

In [126]:
```
#Predicting the label of the new piece of data based in k-nearest neighbours

def knn_prediction(X_train,y_train,X_test,K):
    neighbours=[]
    pred_outcome=[]
    for i in range(len(X_test)):
        neighbours.append(nearest_Neighbours(X_train,y_train,X_test.iloc[i,:]
    for i in neighbours:
        top_neighbours = {}
        for j in i:
            #list of distances of top k-neighbours
            if j[-1] in top_neighbours.keys():
                top_neighbours[j[-1]]=top_neighbours[j[-1]]+1
            else:
                top_neighbours[j[-1]]=1
        pred_outcome.append(sorted(top_neighbours,key=top_neighbours.get,reve
    return pred_outcome #return the label
```

Accuracy calculation of predicted data point

In [127]:
```
# Accuracy of correctly predicted
def knn_getAccuracy(actual,predicted):
    correctly_pred=0
    for i in range(len(predicted)):
        if predicted[i]==actual[i]:
            correctly_pred+=1
    return round((correctly_pred/len(actual))*100,2)
```

Accuracy of Model

In [128]:
```
# Accuracy of predicted species
output=knn_prediction(X_train,y_train,X_test,100)

knn_getAccuracy(y_test,output)
```

Out[128]: 48.36

Checking using KNeighborsClassifier

In [129]:
```
# Packages
%matplotlib notebook
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [130]:
```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ra
```

In [131]:
```
knn = KNeighborsClassifier(n_neighbors = 5) #setting up the KNN model to use
knn.fit(X_train, y_train) #fitting the KNN
```

Out[131]:
```
KNeighborsClassifier()
```

In [150]:
```
#Checking performance on the training set
print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(
#Checking performance on the test set
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_te
```

```
Accuracy of K-NN classifier on training set: 0.53
Accuracy of K-NN classifier on test set: 0.44
```

In [182]:
```
print("Preliminary model score:")
print(knn.score(X_test,y_test))
```

```
Preliminary model score:
0.4387755102040816
```

In [183]:
```
no_neighbors = np.arange(1, 9)
train_accuracy = np.empty(len(no_neighbors))
test_accuracy = np.empty(len(no_neighbors))
```

In [184]:

```
for i, k in enumerate(no_neighbors):
    # We instantiate the classifier
    knn = KNeighborsClassifier(n_neighbors=k)
    # Fit the classifier to the training data
    knn.fit(X_train,y_train)

    # Compute accuracy on the training set
train_accuracy[i] = knn.score(X_train, y_train)

    # Compute accuracy on the testing set
test_accuracy[i] = knn.score(X_test, y_test)
# Visualization of k values vs accuracy

plt.title('k-NN: Varying Number of Neighbors')
plt.plot(no_neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot(no_neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()
```

```
<IPython.core.display.Javascript object>
```