JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Customer Segmentation Model withApplication of Linear Algebra

Sumit Prakash Dubey, Vijay Pawar H G
Student
Reva University

Application of Linear Algebra in Machine LearningModels

Matrix Factorization:

Matrix factorization techniques such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) are used in recommendation systems, image processing,and topic modeling.

Linear Regression:

Linear regression involves solving a system of linear equations to find the coefficients of alinear relationship between variables. This is widely used in statistics, economics, and machine learning.

Optimization:

Linear algebra is fundamental in optimization problems, such as finding the minimum ormaximum of a function subject to constraints. Techniques like gradient descent and Newton's method rely heavily on linear algebra.

Markov Chains:

Markov chains, which model sequences of random events, can be represented and analyzed using matrices. Linear algebra is used to compute steady-state distributions, expected hittingtimes, and absorption probabilities.

Signal Processing:

Techniques like Fourier Transform, Discrete Cosine Transform (DCT), and Discrete WaveletTransform (DWT) rely on linear algebra for analyzing and processing signals in various applications like audio and image compression.

Graph Theory:

Graphs can be represented using adjacency matrices, and linear algebra techniques are used to analyze properties of graphs, such as connectivity, shortest paths, and network flow.

Machine Learning:

Many machine learning algorithms, including support vector machines (SVMs), neural networks, and deep learning models, rely on linear algebra operations for training andinference.

Control Theory:

Linear algebra is fundamental in control theory, where it is used to model and analyze the behavior of dynamical systems, design controllers, and stabilize systems.

Steps taken to implement model

Full EDA:

We perform exploratory data analysis (EDA) to understand the structure and distribution of the dataset, including visualizing pairwise relationships between features using statistics andLinear Algebra Techniques

Data Preprocessing:

Basic data preprocessing steps include extracting relevant features and normalizing them to ensure they have similar scales.

Principal Component Analysis (PCA):

We apply PCA for dimensionality reduction, transforming the original data into a lower- dimensional space while preserving most of its variance.

K-means Clustering on PCA-transformed Data:

K-means clustering is performed on the PCA-transformed data to segment customers based on their principal components.

Visualization:

We visualize the resulting clusters in the reduced dimensionality space to gain insights into the segmentation of customers.

Distance Calculation:

Finally, we calculate the distances between data points and cluster centroids in the reduced space, using linear algebra operations (np.linalg.norm()). This helps in understanding the distribution of data points relative to cluster centers.

Actual Model Implementation

Import All Modules

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.cluster import KMeans from sklearn.decomposition import PCA

In [1]:

```
# Importing necessary modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

Read Mall Customers Dataset

In [2]:

```
# Read the dataset
file_path = r'C:\Users\pc\Desktop\Paper-2\CustomerSegmentation_LA\CustomerSegmentat df =
pd.read_csv(file_path)
```

Exploratory Data Analysis using statistics

In [3]:

```
# Display basic information about the dataset
print(df.info())

# Display summary statistics of numerical columns
print(df.describe())

# Perform exploratory data analysis (EDA)
plt.figure(figsize=(12, 6))
sns.pairplot(df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']])plt.title('Pairplot of Features')
plt.show()
```

<class 'pandas.core.frame.DataFrame'>RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):

#	Column	Non-Null Count	Dtype
---	--------------------	--------------------	----------
0	CustomerID	200 non-null	int64
1	Genre	200 non-null	object
2	Age	200 non-null	int64
3	Annual Income (k$)	200 non-null	int64
4	Spending Score (1-100)	200 non-null	int64dtypes: int64(4), object(1)

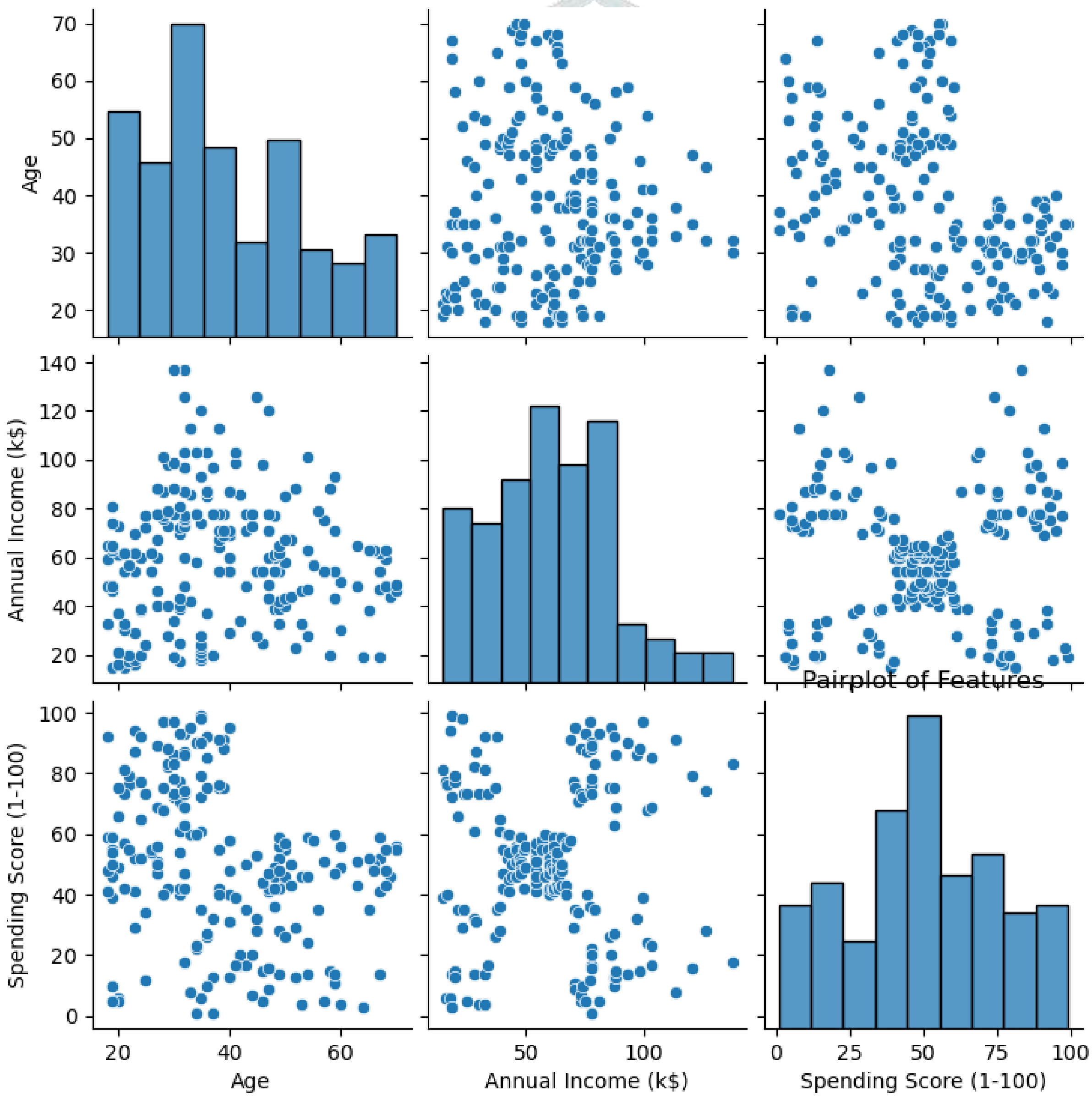
memory usage: 7.9+ KBNone

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000

std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

```
C:\Users\pc\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
self._figure.tight_layout(*args, **kwargs)
```

<Figure size 1200x600 with 0 Axes>



Feature Engineering

In [4]:

```python
# Perform data preprocessing
X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']].values
X_mean = X.mean(axis=0)
X_std = X.std(axis=0)
X_normalized = (X - X_mean) / X_std
```

Application of Linear Algebra for insights

In [5]:

```
# Covariance Matrix
cov_matrix = np.cov(X_normalized.T)

# Eigenvalues and Eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Singular Value Decomposition (SVD)
```

```
U, S, Vt = np.linalg.svd(X_normalized)

# Print insights
print("Covariance Matrix:")
print(cov_matrix)
print("\nEigenvalues:")
print(eigenvalues)
print("\nEigenvectors:")
print(eigenvectors)
print("\nSingular Values:")
print(S)
```

```
Covariance Matrix:
[[ 1.00502513   -0.01246034   -0.3288712 ]
 [-0.01246034    1.00502513    0.00995261]
 [-0.3288712     0.00995261    1.00502513]]

Eigenvalues:
[1.33465831  0.67614435  1.00427272]

Eigenvectors:
[[ 0.70638235   -0.70718844   -0.03014116]
 [-0.04802398   -0.00539792   -0.9988316 ]
 [-0.70619946   -0.70700451    0.03777499]]

Singular Values:
[16.29714709 14.1368409   11.59968646]
```

Insights From Application of Linear AlgebraTechniqques

Covariance Matrix:

Calculate the covariance matrix to understand the relationships between different features.

Eigenvalues and Eigenvectors:

Compute the eigenvalues and eigenvectors of the covariance matrix to understand theprincipal directions of variation in the data.

Singular Value Decomposition (SVD):

Perform Singular Value Decomposition to analyze the underlying structure of the data andidentify dominant patterns.

In [6]:

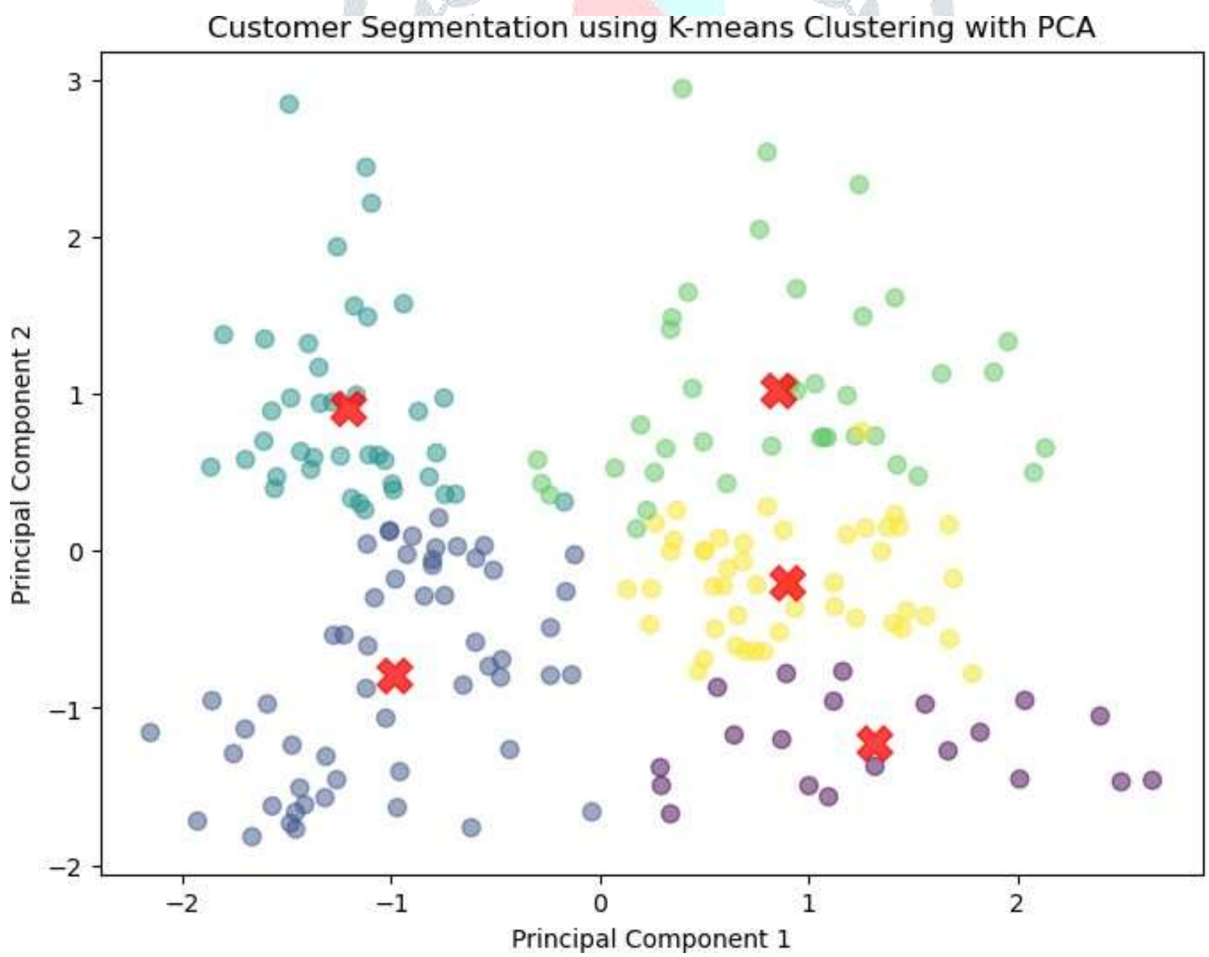
```
# Perform Principal Component Analysis (PCA) with appropriate number of components
n_components = 3  # Choose the desired dimensionality (consider explained variance
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_normalized)
```

In [7]:

```
# Suppress the warning about memory leak (optional)
import warnings
warnings.filterwarnings("ignore",  category=UserWarning)

# Perform K-means clustering on PCA-transformed data
kmeans_pca  =  KMeans(n_clusters=5,  random_state=42,  n_init=10)
kmeans_pca.fit(X_pca)
clusters_pca  =  kmeans_pca.predict(X_pca)
centroids_pca  =  kmeans_pca.cluster_centers_
```

In [8]:

```
# Visualize the clusters in the reduced dimensionality space
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,  0],  X_pca[:,  1],  c=clusters_pca,  cmap='viridis',  s=50,  alpha=0
plt.scatter(centroids_pca[:,  0],  centroids_pca[:,  1],  c='red',  s=200,  alpha=0.75,  m
plt.xlabel('Principal Component 1')
plt.ylabel('Principal  Component  2')
plt.title('Customer  Segmentation  using  K-means  Clustering  with  PCA')
plt.show()
```



In [9]:

```
# Calculate distances between data points and cluster centroids (fixed)
distances = np.zeros((len(X_normalized), len(centroids_pca)))
for i, centroid in enumerate(centroids_pca):
  centroid_reshaped = centroid.reshape(1, -1)  # Reshape to match the data points (
  distances[:, i] = np.linalg.norm(X_normalized - centroid_reshaped, axis=1)
```

```
print("Distances between data points and cluster centroids:")
print(distances)
```

Distances between data points and cluster centroids:

```
[[2.93713366  1.31758287  2.66565179  3.78067793  2.8145136 ]
 [2.73023858  1.29412187  3.18385772  3.51105833  3.34204589]
 [3.50034295  2.3028536   2.86261588  4.30589407  2.85431615]
 [2.55010088  1.14311586  3.06820513  3.38007373  3.1381104 ]
 [2.12459558  1.23465961  2.66218847  3.25690871  2.11228238]
 [2.60474936  1.10206582  3.01458554  3.38984763  3.14475178]
 [2.75879509  2.35402293  2.94096014  3.81393151  2.05749245]
 [2.75279673  1.58042398  3.40157645  3.43249828  3.53651393]
 [2.41668071  3.64257055  4.11717039  3.81888691  1.9460892 ]
 [2.00760314  0.99432218  2.92700507  3.00162946  2.65333742]
 [2.07008505  3.58061881  4.17039862  3.60338965  1.88040579]
 [2.13916382  1.86372055  3.62742331  3.04801411  3.26289277]
 [1.90064945  3.02615066  3.65855786  3.39471005  1.53498764]
 [2.45667019  1.01749077  2.94173856  3.2150915   3.02071129]
 [2.44695327  2.1415345   2.82880998  3.54362941  1.81323257]
 [2.61285172  1.09043098  2.98132016  3.3094588   3.16281113]
 [1.94892622  1.39307546  2.59800249  3.09511277  1.77641412]
 [2.68034726  0.84475001  2.69845247  3.36045572  2.97194095]
 [1.39375675  2.35788737  3.19621841  2.93853129  1.23726914]
 [2.09357624  1.77097004  3.49693562  2.9039918   3.16809657]
 [1.93557596  1.33744078  2.49146607  3.00218958  1.69354991]
 [2.34144688  0.79735074  2.73444726  3.04167656  2.81191001]
 [2.40035391  2.65995327  3.09028293  3.49911433  1.54287737]
 [1.91621689  0.87778564  2.76992348  2.76870282  2.52421904]
 [1.92396864  2.76589258  3.26153403  3.16207217  1.21595204]
 [2.14493942  1.01814142  2.8519817   2.78380508  2.80514829]
 [1.49151299  1.8436      2.72001537  2.74427978  1.14653183]
 [1.58904189  0.84474914  2.54694625  2.55588679  1.99896195]
 [1.75294095  1.59816501  2.48850792  2.81064764  1.29877723]
 [2.61615449  1.14976692  2.90843736  3.05210514  3.19250612]
 [2.30926549  3.3294937   3.64447794  3.45006169  1.51062504]
 [2.6196752   0.70493782  2.5358777   3.05592729  2.92914375]
 [2.3245115   2.95766519  3.21430614  3.31615363  1.30540696]
 [3.02583579  1.38019751  2.95352595  3.24041887  3.52229431]
 [2.00514538  2.48063436  2.88645497  3.02155605  1.05346567]
 [2.68732599  0.91758159  2.63862469  3.00111964  3.08589188]
 [2.10732777  2.0650296   2.51122863  2.97495733  1.16012718]
 [1.99348076  0.66533181  2.47829682  2.52143457  2.43058604]
 [2.11893975  1.52695709  2.10752194  2.79113556  1.31428366]
```

[2.72178594	0.70756753	2.37405752	2.92797371	2.95318213]
[1.28633393	3.02041725	3.55996389	2.54935381	1.20246944]
[2.65084553	1.26018558	2.81896607	2.81242318	3.19983618]
[1.30647483	1.88323669	2.54929133	2.28820392	0.72065738]
[1.91854335	0.42830617	2.09727945	2.35308584	2.0128411]
[1.53984418	2.1105778	2.61947873	2.47838475	0.65044039]
[2.41102365	0.22440472	2.0812396	2.6681692	2.48608828]
[0.7463705	1.79647208	2.73556473	1.90468866	1.16415416]
[2.29356064	0.50997258	1.79118849	2.64377451	1.97041419]
[2.22326918	0.74048827	1.78873128	2.62948916	1.78067778]
[2.0939087	0.8058388	1.83416415	2.54710308	1.65236322]
[0.89252454	1.74301771	2.60495363	1.87731946	1.03507218]
[1.8103136	0.57322782	2.03589031	2.18466139	1.85143321]
[1.9869686	0.49192139	1.85253573	2.29612555	1.81313741]
[0.58589034	2.43620669	3.23223596	1.83932424	1.3959934]
[1.03434864	1.87949665	2.58399349	1.96119033	0.77975125]

```
[0.89930281  1.90304182  2.65110305  1.83558917  0.916237  ]
[1.23633977  3.19907238  3.73152464  2.31630148  1.48700982]
[2.3116254   0.43148939  1.62552658  2.44031866  1.9758846 ]
[0.99177025  2.08476156  2.69590049  1.84965738  0.75196303]
[1.17656152  3.23426135  3.82667152  2.17336612  1.73008592]
[2.83391791  0.52781048  1.66404048  2.83311221  2.55528791]
[1.09366313  3.03423556  3.58826714  2.06249743  1.46278711]
[0.76085281  2.09230967  2.85282689  1.62003061  1.21360885]
[0.98071955  2.75846686  3.30891301  1.89896857  1.22005554]
[2.90894442  0.59666627  1.68610717  2.8213668   2.6746988 ]
[1.35884603  1.37396511  2.12602335  1.78685108  1.04456666]
[1.23212751  3.1276209   3.61445512  2.13506307  1.42564307]
[2.83976425  0.53672783  1.67534956  2.76195773  2.61061268]
[2.04972489  0.76356493  1.62768924  2.21085664  1.57794807]
[1.25469258  3.24647418  3.77626034  2.10256731  1.68672451]
[1.34957961  1.7499528   2.26154181  1.85375977  0.63552321]
[1.01274767  2.56816756  3.07172493  1.78125228  1.00670021]
[0.89409215  2.53860585  3.12879511  1.67103655  1.22647881]
[1.16939046  2.54020137  2.93193103  1.67524693  0.88157217]
[2.46265737  0.58562077  1.39387298  2.27410079  2.06035164]
[1.36563149  1.54853349  2.1282498   1.49925676  1.03127506]
[1.67603017  1.28042565  1.80319865  1.72397417  1.09570126]
[2.67095215  0.63360512  1.32733973  2.46716786  2.22093078]
[1.40349442  1.92445784  2.27350297  1.69085199  0.52677293]
[1.08517557  2.37618467  2.83103851  1.54804856  0.93759703]
[1.71311884  1.08671778  1.79840918  1.67676875  1.38757317]
[1.48347133  3.13996848  3.43867246  2.07360626  1.2114496 ]
[1.46451071  1.70609401  2.10355742  1.67090319  0.69311982]
[2.77953275  0.62251663  1.43001738  2.53703962  2.43281207]
[1.35155179  1.8093355   2.23510051  1.60054452  0.6965297 ]
[1.11936628  2.24440254  2.73072342  1.29262332  1.14799187]
[2.76390337  0.71350189  1.28986079  2.43829304  2.33460148]
[2.00986248  0.94183471  1.62894804  1.68993249  1.72978492]
[1.40746581  1.98848816  2.28455828  1.47027484  0.66569421]
[1.44022252  3.16962915  3.52172448  1.75301025  1.56252198]
[3.15956661  1.14736545  1.03641298  2.82671492  2.44272295]
[1.47782005  1.86094554  2.14961589  1.35035431  0.82363363]
[1.94218039  1.52194767  1.60337369  1.7555355   0.93418193]
[2.31895179  1.14245677  1.20405404  2.02862932  1.48700752]
[2.69948771  0.82726186  1.13837838  2.29711103  2.16180715]
[1.54100458  1.81730199  2.06985676  1.41147853  0.77798392]
[2.52668772  0.86547981  1.14528564  2.14841663  1.93452752]
[1.62829505  1.95495987  2.08438665  1.50781796  0.59172972]
[2.99036766  0.97711676  1.0453485   2.56170382  2.38886535]
[2.89442611  1.12251961  0.88674423  2.49006418  2.10260971]
[1.52625456  1.96492351  2.1721204   1.30913736  0.78949319]
[1.47393158  3.12377839  3.46126926  1.58190498  1.62935566]
[2.59376534  0.86593239  1.17519415  2.10709432  2.09331134]
[1.43485768  1.9173774   2.26063421  1.1255801   1.08266456]
[3.00591257  1.12653837  0.8966198   2.58924704  2.25016664]
[1.55294524  3.08824195  3.30888072  1.64387512  1.3544324 ]
[1.49589049  2.3158391   2.4707931   1.35076024  0.73689122]
[1.72287234  3.26773957  3.41295922  1.8828723   1.33867034]
[1.58031117  3.09862401  3.29676918  1.68289581  1.30971628]
[1.50050157  3.01145375  3.25465381  1.55952765  1.34916512]
```

```
[2.09818609 1.47486556 1.41067344 1.68815524 1.12946551]
[3.12564767 1.14893372 0.90532558 2.62203843 2.43558653]
[3.18977464 1.17619562 0.93025025 2.64545566 2.53069262]
[3.11307262 1.11860173 0.95352446 2.55674063 2.48692992]
[1.65630028 2.95595131 3.04774544 1.62699699 1.05596138]
[1.521053 1.96553748 2.26351976 0.96999588 1.22416659]
[1.72499158 2.22397369 2.20356307 1.3207113 0.69456734]
[1.5765948 2.06807536 2.27254346 0.93800753 1.16945355]
[2.6295471 1.05380416 1.09322128 1.95264917 2.08266083]
[2.20462683 1.58482134 1.33861509 1.68440182 1.13716108]
[1.98650636 1.54399451 1.66301774 1.14685098 1.47772882]
[2.29112927 1.92669942 2.53811322 1.35100245 2.60925317]
[3.20535093 1.66154871 0.61707406 2.64363599 2.11182842]
[2.51277301 1.39817329 1.81405496 1.58078943 2.42660707]
[2.20578774 1.99459505 1.59264192 1.61126901 0.87563703]
[2.38099774 2.10668766 2.68849885 1.37105671 2.74585005]
[2.6016612 3.30145581 2.86970838 2.46659579 1.07474993]
[2.17472001 1.61979749 1.97679568 1.11747234 2.10552549]
[2.7563156 2.78380676 2.13534247 2.48290947 1.03492466]
[2.13019454 1.66187495 2.01930142 1.0597465 2.07377453]
[3.05976343 1.57962839 0.52839231 2.3990338 2.00517563]
[2.52333003 1.36996976 1.58739132 1.53007155 2.27372947]
[3.88522681 2.49679234 1.27966526 3.40963836 2.53168172]
[2.80857964 1.69932879 2.13613705 1.7843474 2.86978837]
[2.92280636 2.75999993 1.98190578 2.57163213 1.21558758]
[2.50693515 1.45336526 1.67367068 1.45224055 2.29623008]
[3.84368154 2.36717953 1.09810264 3.31010463 2.54223513]
[2.37876671 1.55924958 1.73587462 1.23744081 2.15462943]
[2.87548944 3.39734371 2.80372974 2.62636821 1.2680941 ]
[2.78253542 1.9283332 2.36422102 1.66821769 2.93633837]
[2.91086347 1.58953484 0.56410945 2.0605402 1.89974649]
[2.71944525 1.8092709 2.14049557 1.54805063 2.75350297]
[3.56379549 2.33030592 0.98852745 2.95308862 2.1708978 ]
[3.08541433 2.03789566 2.43418835 1.96594959 3.22739324]
[2.2317538 2.35650166 1.88721376 1.41064471 0.89886441]
[2.6188652 1.60104799 1.67450678 1.40513487 2.3737324 ]
[2.98175409 2.15930986 1.0423617 2.26119536 1.54261645]
[2.72218986 1.97825387 2.28942082 1.46086912 2.80443778]
[2.79726873 2.55157962 1.69109971 2.17967869 1.1541953 ]
[2.48867901 2.07942195 2.38045813 1.13639573 2.59578688]
[2.69816977 2.51555106 1.7103757 2.05092683 1.07418266]
[2.3887307 1.83784436 1.95526523 1.00115952 2.22764225]
[2.73251852 2.73144626 1.96596802 2.16309147 1.05242604]
[3.0413341 1.85304846 2.11358016 1.87978524 3.03132953]
[3.39074256 2.83413277 1.74270939 2.89482125 1.73542546]
[2.77368818 1.65782799 1.75608658 1.55486057 2.58509661]
[3.48781856 2.77682616 1.61869488 2.97485187 1.87149677]
[2.74013178 1.58416844 1.57621172 1.5306007 2.44653088]
[2.22086062 2.83993687 2.44858691 1.4656992 0.98577431]
[2.89551365 1.7690996 1.9141262 1.66147619 2.78352733]
[4.07609048 2.67524379 1.21980391 3.42148687 2.69100089]
[2.97873611 2.07821247 2.31897861 1.67865301 3.0367933 ]
[2.64770528 2.80497372 2.04399285 1.73184321 1.14135574]
[2.67688031 1.98332772 1.82554369 1.07284584 2.37733695]
[2.96975612 2.63457717 1.56013883 2.05938052 1.39526536]
```

```
[3.05670029  2.30299776  2.43305809  1.58697167  3.10204847]
[3.04461342  2.33550448  1.06381897  1.99235414  1.63755884]
[2.87129787  1.870141    1.28208184  1.37365863  2.26011087]
[3.2103494   2.76761998  1.56661448  2.36190865  1.58747388]
[3.09358102  1.90613238  1.58681414  1.63842409  2.72531033]
[3.40020229  2.75016975  1.4193478   2.57599715  1.7977395 ]
[2.9223596   2.32899915  2.40604894  1.34453203  2.93437336]
[3.01751507  3.22220057  2.32987287  2.23563212  1.39583184]
[3.12027498  2.13159659  2.03840398  1.60260689  2.95962916]
[2.94590862  3.45769366  2.70855113  2.21877042  1.44225201]
[3.14461656  1.87724821  1.34689074  1.70218627  2.64452664]
[3.12151316  3.62951308  2.80412538  2.28441905  1.64599734]
[3.10993427  2.45038023  2.32649592  1.37400894  3.00114024]
[3.2251166   2.5761061   1.18524665  1.82791358  1.8970727 ]
[3.29823725  2.4587244   2.13292066  1.51605741  3.05165376]
[3.34531641  3.19025847  1.96877466  2.219912    1.76053626]
[3.47062128  2.49219619  2.15264608  1.74689476  3.22936607]
[3.07826708  2.65298236  1.47598135  1.47723327  1.86255369]
[3.57200344  2.70522726  2.50835333  1.86057857  3.47061396]
[3.16867532  3.41496185  2.42347567  1.89533995  1.76265756]
[3.47423713  2.36457566  1.46615676  1.71368575  2.84789569]
[3.55770732  3.13844438  1.6979481   2.27205377  2.02059403]
[3.33321105  2.71761717  2.27240733  1.33676255  3.03424666]
[3.64546665  2.87022224  1.21724001  2.28039317  2.21736329]
[3.37609355  2.48636432  1.63689111  1.47078449  2.76033322]
[4.24367612  3.48323485  1.72100669  2.91003966  2.69349732]
[3.66913841  3.18249554  2.66494246  1.55668365  3.39203953]
[4.01281741  3.83355095  2.37404952  2.47162054  2.53853751]
[3.88690373  3.2282977   2.35034055  1.7129249   3.34593335]
[4.06125346  3.78901453  2.30605763  2.2525846   2.73407782]
[4.15821883  3.37068188  2.28630176  1.99990327  3.49536094]
[4.84783746  4.07468345  2.23583942  3.09143337  3.43913477]
[4.64183049  3.83245654  2.77629526  2.45719475  4.05131836]]
```

In [10]:

```python
# Compute statistics for each cluster
for i in range(len(centroids_pca)):
    cluster_distances = distances[:, i]
    mean_distance  =  np.mean(cluster_distances)
    median_distance  =  np.median(cluster_distances)
    min_distance = np.min(cluster_distances)
    max_distance  =  np.max(cluster_distances)

    print(f"Cluster {i + 1}:")
    print(f"Mean  Distance:  {mean_distance}")
    print(f"Median  Distance:  {median_distance}")
    print(f"Minimum  Distance:  {min_distance}")
    print(f"Maximum  Distance:  {max_distance}")
    print()
```

Cluster 1:
Mean Distance: 2.4075014472813403
Median Distance: 2.49780707812159
Minimum Distance: 0.5858903437535338
Maximum Distance: 4.84783746179991

Cluster 2:
Mean Distance: 1.9938360843303542
Median Distance: 1.9275163129332378
Minimum Distance: 0.22440472070386042
Maximum Distance: 4.074683453499659

Cluster 3:
Mean Distance: 2.2431547611114797
Median Distance: 2.262530785108727
Minimum Distance: 0.528392314761445
Maximum Distance: 4.1703986182478925

Cluster 4:
Mean Distance: 2.21307746886698
Median Distance: 2.1417398478468956
Minimum Distance: 0.938007534038265
Maximum Distance: 4.305894067943124

Cluster 5:
Mean Distance: 1.9623379578935185
Median Distance: 1.9171370036374435
Minimum Distance: 0.5267729277597252
Maximum Distance: 4.051318364389533

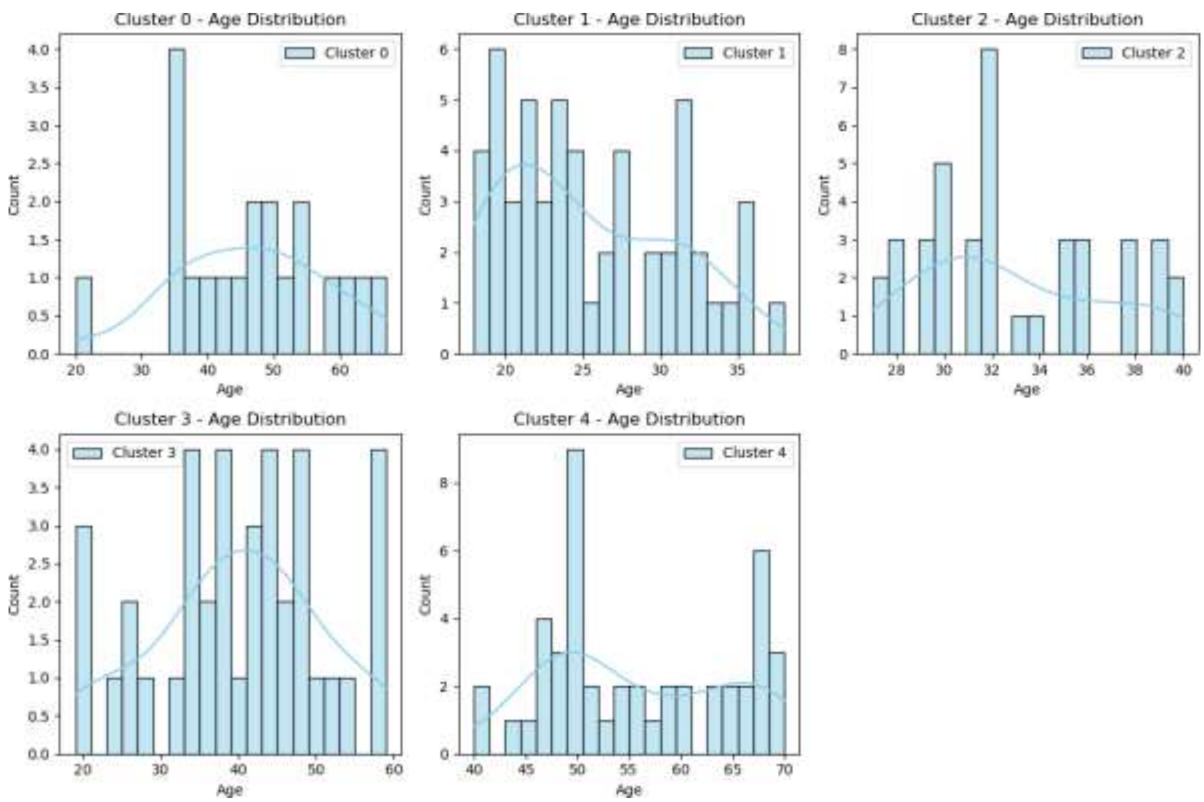
In [11]:

```
# Add cluster labels to the original DataFrame
df['Cluster'] = clusters_pca

# Analyze the characteristics of each cluster
cluster_characteristics = df.groupby('Cluster').agg({
  'Age': ['mean', 'median', 'std'],
  'Annual Income (k$)': ['mean', 'median', 'std'],
  'Spending Score (1-100)': ['mean', 'median', 'std', 'count']
}).reset_index()

# Print the characteristics of each cluster
print("Cluster  Characteristics:")
print(cluster_characteristics)

# Visualize the distribution of features within each cluster (optional)
plt.figure(figsize=(12, 8))
for i in range(len(cluster_characteristics)):
  plt.subplot(2, 3, i + 1)
  sns.histplot(data=df[df['Cluster'] == i], x='Age', kde=True, bins=20, color='skyb
  plt.title('Cluster ' + str(i) + ' - Age Distribution')
  plt.xlabel('Age')
  plt.legend()
plt.tight_layout()
plt.show()
```

Cluster Characteristics:

Cluster		Age			Annual Income (k$)			\
		mean	median	std	mean	median	std	
0	0	46.250000	47.0	11.579815	26.750000	26.5	7.311671	
1	1	25.185185	24.0	5.508395	41.092593	40.0	16.815613	
2	2	32.875000	32.0	3.857643	86.100000	78.5	16.339036	
3	3	39.871795	41.0	10.938054	86.102564	78.0	16.725013	
4	4	55.638298	54.0	8.913657	54.382979	54.0	8.818344	

Spending	Score (1-100)			
	mean	median	std	count
0	18.350000	14.5	11.935242	20
1	62.240741	58.0	16.596130	54
2	81.525000	83.0	9.999968	40
3	19.358974	17.0	11.610991	39
4	48.851064	48.0	6.303825	47



The cluster analysis reveals distinctive characteristics among different groups of customers:

Cluster 0: This group consists of individuals with a mean age of 46.25 years and relatively lower annual income, with a mean of $26,750. They exhibit a moderate spending score witha mean of 18.35, suggesting they are cautious spenders.

Cluster 1: This cluster represents younger individuals, with a mean age of 25.19 years. Theyhave a moderate annual income, averaging $41,092.59, and a relatively higher spending
score with a mean of 62.24. This group likely includes young, affluent shoppers.

Cluster 2: Individuals in this cluster are slightly older, with a mean age of 32.88 years. Theyhave a significantly higher annual income, averaging $86,100, and they also exhibit the highest spending score among the clusters, with a mean of 81.53. This suggests they are high-income, high-spending customers.

Cluster 3: Similar to Cluster 2 in terms of income, individuals in this cluster have a mean age of 39.87 years and an average annual income of $86,102. They, however, have a much lowerspending score, with a mean of 19.36. This group may be more conservative in their
spending habits despite their high income.

Cluster 4: This cluster comprises older individuals, with a mean age of 55.64 years. They have a moderate annual income, averaging $54,382.98, and a moderate spending score with a
mean of 48.85. This group likely represents middle-aged individuals with moderate spending
habits.

Visual inspection of the age distribution within each cluster confirms the differences in age demographics among the clusters. It's evident that each cluster represents a distinct
segment of customers with varying ages, income levels, and spending behaviors. This analysis can be valuable for targeted marketing strategies tailored to each customer segment.

In [12]:

```
# Further Analysis: Segmenting customers and creating targeted marketing campaigns

# Segmenting customers based on cluster labels
cluster_0 = df[df['Cluster'] == 0]
cluster_1 = df[df['Cluster'] == 1]
cluster_2 = df[df['Cluster'] == 2]
cluster_3 = df[df['Cluster'] == 3]
cluster_4 = df[df['Cluster'] == 4]

# Example targeted marketing campaign for Cluster 1 (younger, moderate income, mode
print("Targeted Marketing Campaign for Cluster 1 (Young Professionals):")
print("  - Offer discounts and promotions on products relevant to young adults.")
print("  - Advertise through social media channels popular with this age group.")
print("  - Highlight the convenience and affordability of your products.")

# You can create similar targeted
```

Targeted Marketing Campaign for Cluster 1 (Young Professionals):
- Offer discounts and promotions on products relevant to young adults.
- Advertise through social media channels popular with this age group.
- Highlight the convenience and affordability of your products.

In [ ]: