Code:

1) Importing all the necessary libraries:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import streamlit as st
import plotly as pt
from sklearn.metrics import davies_bouldin_score
from sklearn.preprocessing import LabelEncoder
```

2) Dataset Loading and preprocessing

```
df = pd.read_csv('Mall_Customers.csv')
df.isnull().sum()
df.duplicated().sum()
le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df = df.drop('CustomerID')

plt.boxplot(df['Age'], vert=True, patch_artist=True)
plt.title('Boxplot Example')
plt.xlabel('Groups')
plt.ylabel('Values')
plt.show()

plt.boxplot(df['Gender'], vert=True, patch_artist=True)
plt.title('Boxplot Example')
plt.xlabel('Groups')
plt.ylabel('Values')
plt.show()
plt.boxplot(df['Annual Income (k$)'], vert=True, patch_artist=True)
plt.title('Boxplot Example')
plt.xlabel('Groups')
plt.ylabel('Values')
plt.show()


plt.boxplot(df['Spending Score (1-100)'], vert=True, patch_artist=True)
plt.title('Boxplot Example')
plt.xlabel('Groups')
plt.ylabel('Values')
plt.show()
```

3) Data Visualisations:

3.1) Distribution of Age by Gender

```
plt.hist(df[df['Gender'] == 'Male']['Age'], alpha=0.5, label='Male',
color='blue');
plt.hist(df[df['Gender'] == 'Female']['Age'], alpha=0.5, label='Female',
color='pink');
plt.title('Distribution of Age by Gender');
plt.xlabel('Age');
plt.legend();
plt.show()
```

3.2) Gender Vs Annual Income

```
import matplotlib.pyplot as plt
df.groupby('Gender')['Annual Income (k$)'].mean().plot.bar(color=['blue',
'pink'])
plt.title("Gender Vs Annual Income")
plt.xlabel("Gender")
plt.ylabel("Annual Income (k$)")
plt.show()
```

3.3) Gender Vs Spending Score (1-100)

```
import matplotlib.pyplot as plt
df.groupby('Gender')['Spending Score (1-
100)'].mean().plot.bar(color=['royalblue', 'darkgray'])
plt.title("Gender Vs Spending Score (1-100)")
plt.xlabel("Gender")
plt.ylabel("Spending Score (1-100)")
plt.show()
```

3.4) Correlation Heatmap of Attributes

```
correlation_matrix = df.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=.5, cbar=True, vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Attributes')
plt.show()
```

4) K-means implementation

4.1) Elbow method

```
X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)','Gender']]
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

4.2) Silhouette scores

```
X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)', 'Gender']]

for k in range(2, 15):
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    labels = kmeans.fit_predict(X)
    silhouette_avg = silhouette_score(X, labels)
    print(f"For k={k}, the silhouette score is {silhouette_avg}")
```

4.3) K-means after finding k- value

```
X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)', 'Gender']]
k_optimal = 6
kmeans_optimal = KMeans(n_clusters=k_optimal, init='k-means++',
random_state=42)
labels_optimal = kmeans_optimal.fit_predict(X)
centroids_optimal = kmeans_optimal.cluster_centers_
df['Cluster_Optimal'] = labels_optimal
```

```
fig = px.scatter_3d(df, x='Age', y='Annual Income (k$)', z='Spending
Score (1-100)', color='Cluster_Optimal',
                 size_max=10, opacity=0.8, title='Optimal K-means
Clustering', color_discrete_map={i: f'cluster {i}' for i in
range(k_optimal)})
fig.update_layout(scene=dict(aspectmode="cube"))
fig.show()
```

5) Model Validation – Davies-Bouldin Index

```
db_index = davies_bouldin_score(X, labels_optimal)
print(f"Davies-Bouldin Index: {db_index}")
```

6) Model app Development – Streamlit

```
X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)', 'Gender']]
k_optimal = 6
kmeans_optimal = KMeans(n_clusters=k_optimal, init='k-means++',
random_state=42)
labels_optimal = kmeans_optimal.fit_predict(X)
df['Cluster_Optimal'] = labels_optimal
def generate_strategy(cluster):
    st.subheader(f'Marketing Strategy for Cluster {cluster}')
    if cluster == 0:
        st.subheader("these contains  Mainly older customers (average age 56)
with moderate income and spending")
        st.write("Target older customers with moderate income and spending.")
        st.write("Create special deals and promotions for loyalty.")
    elif cluster == 1:
        st.subheader("This cluster mainly contains  Younger customers (average
age 42) with higher income but lower spending")
        st.write("Attract younger customers with higher income but lower
spending.")
        st.write("Offer discounts and suggest additional products.")
    elif cluster == 2:
        st.subheader("This cluster mainly contains young customers (average age
25) with lower income but high spending")
        st.write("Focus on trendy and fashionable products.")
        st.write("Use social media for marketing to reach a younger audience.")
    elif cluster == 3:
        st.subheader("This cluster contains mainly Customers with moderate
income and spending (average age 27)")
        st.write("Targeted advertising for customers with average incomes.")
        st.write("Introduce bundles of products to promote increased spending.")
    elif cluster == 4:
        st.subheader("This Cluster mainly contains the Customers with higher
income and high spending (average age 33)")
        st.write("Emphasize premium and high-quality products")
        st.write("Host exclusive events for this cluster")
    elif cluster == 5:
        st.subheader("Older customers (average age 44) with lower income and
spending")
        st.write("Focus on affordable and value-based products.")
        st.write("Educate customers about product value.")
    else:
        st.write("Strategy not defined for this cluster.")
st.title('Cluster Analytics App')
selected_cluster = st.sidebar.selectbox('Select Cluster', ['Visualize All'] +
list(range(6)))
new_data_placeholder = st.empty()
if selected_cluster == 'Visualize All':
    fig = px.scatter_3d(df, x='Age', y='Annual Income (k$)', z='Spending Score
(1-100)', color='Cluster_Optimal',
                        size_max=10, opacity=0.8, title='Optimal K-means
Clustering',
                        color_discrete_map={i: f'cluster {i}' for i in
range(k_optimal)})
    fig.update_layout(scene=dict(aspectmode="cube"))
    st.plotly_chart(fig)
    st.sidebar.subheader('Check for New Data')
    new_data_age = st.sidebar.number_input('Age', min_value=0,
max_value=100, value=25)
    new_data_income = st.sidebar.number_input('Annual Income (k$)',
min_value=0, max_value=200, value=50)
    new_data_spending = st.sidebar.number_input('Spending Score (1-100)',
min_value=0, max_value=100, value=50)
    new_data_gender = st.sidebar.selectbox('Gender', ['Male', 'Female'])
    new_data_gender_encoded = le.transform([new_data_gender])[0]
    if st.sidebar.button('Check for Cluster'):
        new_data = np.array([[new_data_age, new_data_income,
new_data_spending, new_data_gender_encoded]])
        predicted_cluster = kmeans_optimal.predict(new_data)[0]
        st.sidebar.subheader(f'The new data belongs to Cluster
{predicted_cluster}')
        generate_strategy(predicted_cluster)
else:
    new_data_placeholder.empty()
    selected_cluster_data = df[df['Cluster_Optimal'] == selected_cluster]
    fig_selected_cluster = px.scatter_3d(selected_cluster_data, x='Age',
y='Annual Income (k$)',                                          z='Spending Score (1-100)',
title=f'Cluster {selected_cluster} Visualization')
    fig_selected_cluster.update_layout(scene=dict(aspectmode="cube"))
    st.plotly_chart(fig_selected_cluster)
    st.subheader(f'Statistics for Cluster {selected_cluster}')
    st.write(selected_cluster_data.describe())
    generate_strategy(selected_cluster)
```