```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

```
from google.colab import files
uploaded=files.upload()
```

Choose files Mall_Customers.csv

- **Mall_Customers.csv**(text/csv) - 4286 bytes, last modified: 06/10/2023 - 100% done

```
df=pd.read_csv("Mall_Customers.csv")
print(df.shape)
print("The first 5 rows of the dataframe")
df.head(10)
```

```
(200, 5)
The first 5 rows of the dataframe
```

	CustomerID	Genre	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
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.columns.to_list()
```

```
['CustomerID', 'Genre', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)']
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
CustomerID	200.0	100.50	57.879185	1.0	50.75	100.5	150.25	200.0
Age	200.0	38.85	13.969007	18.0	28.75	36.0	49.00	70.0
Annual Income (k$)	200.0	60.56	26.264721	15.0	41.50	61.5	78.00	137.0
Spending Score (1-100)	200.0	50.20	25.823522	1.0	34.75	50.0	73.00	99.0

```
df.nunique()
```

	0
CustomerID	200
Genre	2
Age	51
Annual Income (k$)	64
Spending Score (1-100)	84

dtype: int64

```
df.duplicated().sum()
```

```
0
```

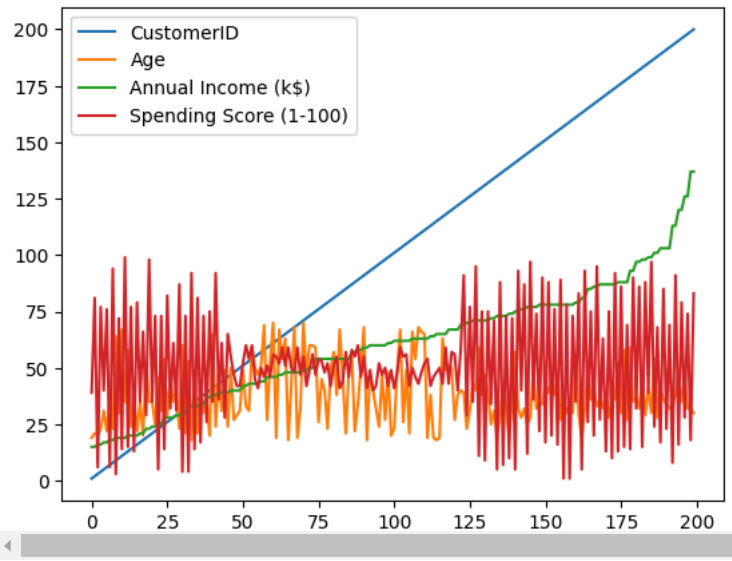
```
df.isna().sum()
```

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

dtype: int64

```
df.plot()
```

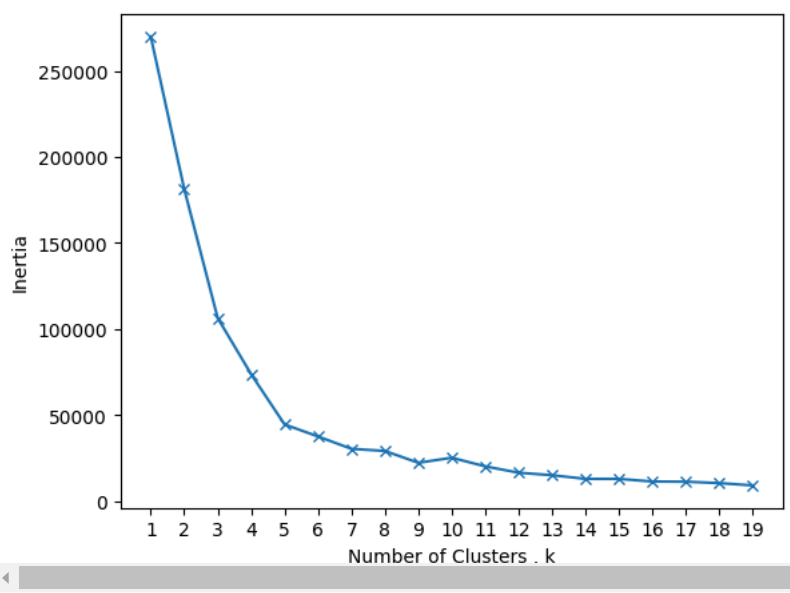
```
<Axes: >
```



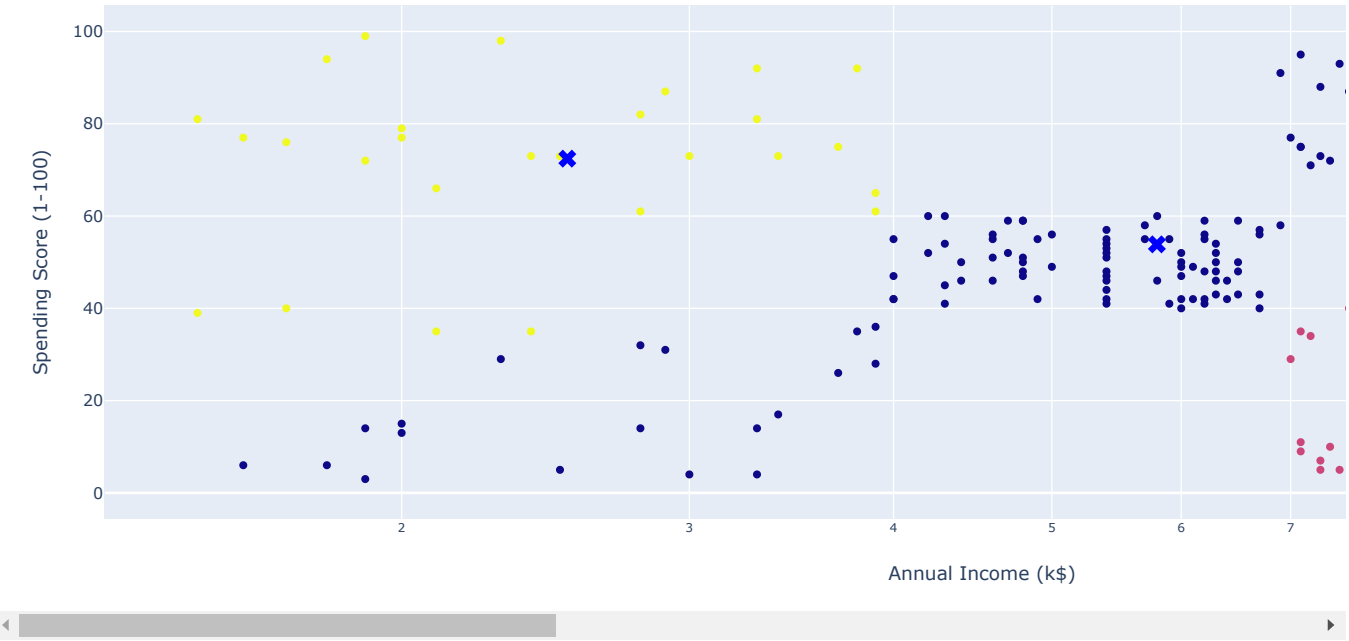

```
df = df.drop(["CustomerID","Age","Genre"],axis=1)
```

K-MEANS CLUSTERING

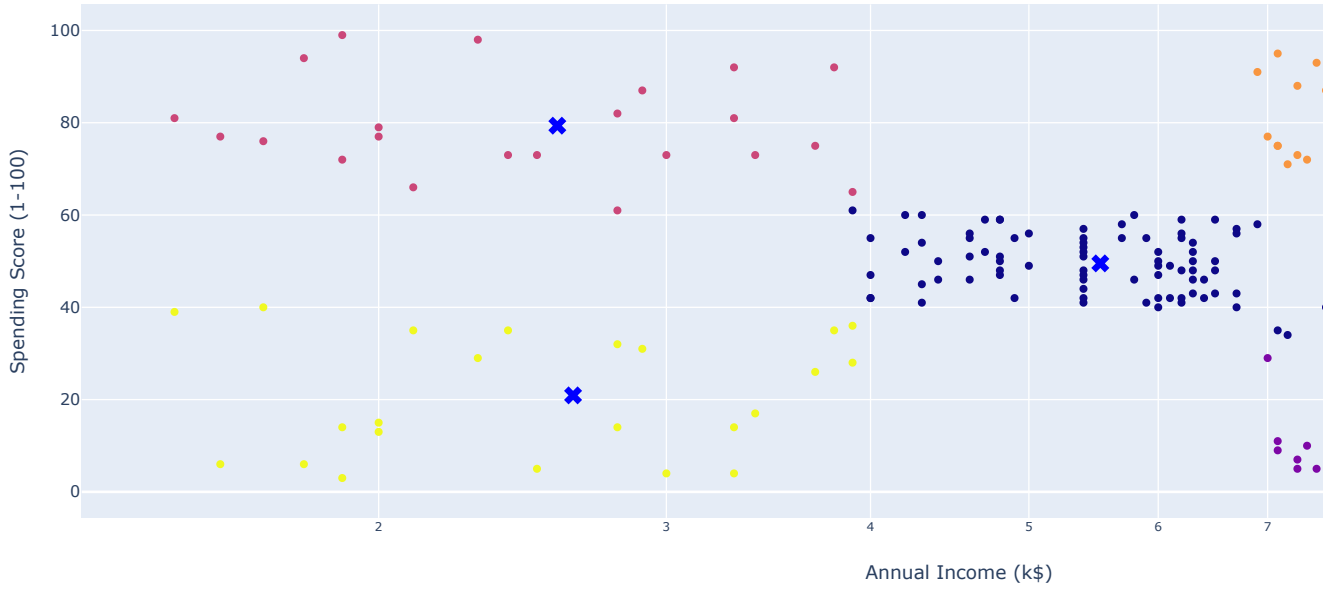
```
#elbow method
ks=range(1,20)
inertias=[]

for k in ks:
    model=KMeans(n_clusters=k)
    model.fit(df)
    inertias.append(model.inertia_)

plt.plot(ks,inertias,'-x')
plt.xlabel('Number of Clusters , k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.show()
```




```
#3-clusters method
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 1, n_init='auto')
model1 = kmeans.fit_predict(df)
centroids = kmeans.cluster_centers_
centroids_df = pd.DataFrame(centroids, columns=["Annual Income (k$)", "Spending Score (1-100)"])

fig = px.scatter(df, x="Annual Income (k$)", y="Spending Score (1-100)", log_x=True, color = model1 )
fig.add_scatter(x=centroids_df["Annual Income (k$)"], y=centroids_df["Spending Score (1-100)"],
                mode="markers", marker=dict(size=12, color="blue", symbol="x"),
                name="Cluster Centroids")

fig.show()
```




```
#5-clusters method
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 1, n_init='auto')
model1 = kmeans.fit_predict(df)
centroids = kmeans.cluster_centers_
centroids_df = pd.DataFrame(centroids, columns=["Annual Income (k$)", "Spending Score (1-100)"])

fig = px.scatter(df, x="Annual Income (k$)", y="Spending Score (1-100)", log_x=True, color = model1 )
fig.add_scatter(x=centroids_df["Annual Income (k$)"], y=centroids_df["Spending Score (1-100)"],
                mode="markers", marker=dict(size=12, color="blue", symbol="x"),
                name="Cluster Centroids")

fig.show()
```



DBSCAN

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
```

```
eps= np.arange(0.3,10.2,0.2)
min_samples = np.arange(2,11)
noise=[]
best_noise = float('inf')
best_score = float('-inf')
best_model = None
for i in eps:
    for j in min_samples:
        dbSCAN_Model = DBSCAN(eps=i, min_samples=j).fit(df)
        labels = dbSCAN_Model.labels_
        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

        if  n_clusters  < 5 or n_clusters  >10 :
            continue

        noise_percentage = (list(labels).count(-1) / len(df)) * 100
        noise.append(noise_percentage)

        score = silhouette_score(df, labels)

        if (noise_percentage < best_noise) or (noise_percentage == best_noise and score > best_score):
            best_noise = noise_percentage
            best_score = score
            best_model = dbSCAN_Model

print(f"DBSCAN Silhouette Score: {best_score}")
print(f"Noise: {best_noise}")
best_model
```

```
DBSCAN Silhouette Score: 0.448074652835006
Noise: 3.0
DBSCAN
DBSCAN(eps=9.100000000000001, min_samples=2)
```

```
eps= np.arange(0.2,10.2,0.2)
min_samples = np.arange(2,11)
noise=[]
score=[]
n_clusters_lst=[]

eps_edit=[]
min_samples_edit =[]
for i in eps:
    for j in min_samples:
```

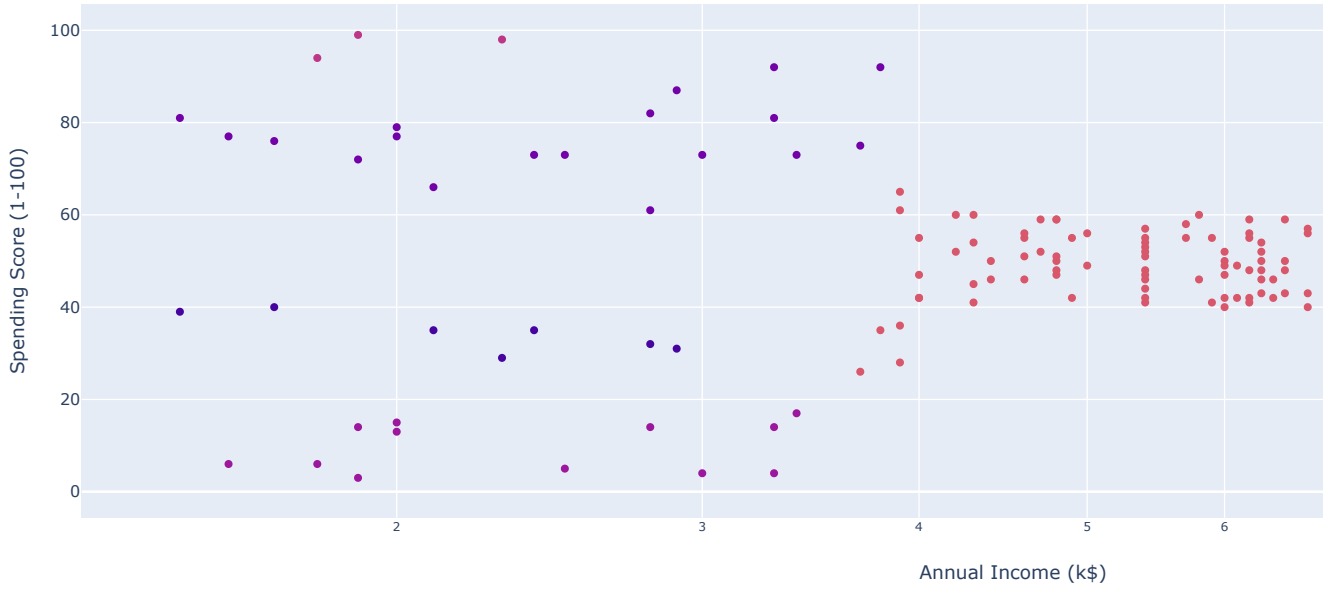
```
        dbSCAN_Model = DBSCAN(eps=i, min_samples=j).fit(df)
        labels = dbSCAN_Model.labels_
        n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
        n_clusters_lst.append(n_clusters)
        if  n_clusters  < 2 :
            continue

        noise_percentage = (list(labels).count(-1) / len(df)) * 100
        noise.append(noise_percentage)

        score.append(silhouette_score(df, labels))
        eps_edit.append(i)
        min_samples_edit.append(j)

models = pd.DataFrame({
   'epsilon' :eps_edit,
    'min samples': min_samples_edit,
    'noise':noise,
    'score': score
})
```

```
models.sort_values(by=['noise', 'score'], ascending=[True, False])
```

	epsilon	min samples	noise	score
291	9.2	2	3.0	0.448075
300	9.4	2	3.0	0.448075
309	9.6	2	3.0	0.322098
318	9.8	2	3.0	0.322098
327	10.0	2	3.0	0.322098
...	...	...	...	...
15	1.8	4	95.5	-0.348524
0	0.2	2	96.0	-0.503776
1	0.4	2	96.0	-0.503776
2	0.6	2	96.0	-0.503776
3	0.8	2	96.0	-0.503776

336 rows × 4 columns

```
dbscan =DBSCAN(eps=9.2, min_samples=2).fit_predict(df)
fig = px.scatter(df, x="Annual Income (k$)", y="Spending Score (1-100)", log_x=True, color =dbscan )
fig.show()
```



FUZZY C-MEANS

```
pip install -U scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl.metadata (2.6 kB)
Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl (920 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 920.8/920.8 kB 25.1 MB/s eta 0:00:00
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.5.0
```

```
import skfuzzy as fuzz
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
df_scaled = sc.fit_transform(df)
```

```
c = 5  # No. of clusters
m = 2  # Fuzzy coefficient
```

CONCLUSION :

In K-Means Clustering the best value for K is 5 using elbow method and it is also justified using visualization. In DBSCAN, we can observe that maximum 8 clusters are forming with Silhouette Score: 0.448074652835006 Fuzzy C-Means also we can see proper grouping of data in 5 clusters