1. Affinity Propagation Clustering

Affinity Propagation is a clustering algorithm that identifies clusters by passing messages between data points. It does not require the number of clusters to be pre-specified. The algorithm works by exchanging two types of messages: *responsibility* ***(how suitable a point is as a cluster center) and*** *availability* ***(how appropriate it is to choose a point as an exemplar). The process iterates until clusters are formed around the most suitable exemplars.***

[1]:
```
#importing the Libraries#
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[2]:
```
from sklearn.cluster import AffinityPropagation
from sklearn import metrics
from sklearn.metrics import pairwise_distances
from sklearn.datasets import make_blobs
```

[3]:
```
#Reading the Dataset#
dataset=pd.read_csv('Mall_Customers.csv')
```

[4]:
```
#what saved in this file#
dataset
```

[4]:

	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
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

[5]:
```
X = dataset.iloc[:,[3,4]].values
```

```
X
```

[7]:
```
# Fit Affinity Propagation model
affinity = AffinityPropagation(random_state=5)
labels=affinity.fit_predict(X)
```

[8]:
```
labels
```

[8]:
```
array([ 2,  1,  0,  1,  2,  1,  0,  1,  0,  1,  0,  1,  0,  1,  0,  1,  2,
        1,  2,  1,  2,  1,  0,  1,  0,  1,  2,  3,  2,  1,  0,  1,  0,  1,
        0,  1,  0,  1,  2,  1,  2,  1,  2,  3,  2,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
        3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
        4,  4,  4,  4,  4,  4,  5,  4,  5,  5,  4,  4,  4,  5,  4,  5,  4,
        4,  4,  4,  5,  4,  4,  5,  4,  4,  4,  5,  5,  4,  4,  5,  4,  5,
        4,  4,  5,  4,  7,  5,  8,  5,  7,  6,  8,  6,  8,  5,  8,  6,  7,
        6,  8,  6,  8,  6,  7,  5,  7,  6,  7,  5,  8,  6,  7,  6,  7,  6,
        8,  6,  7,  6,  8,  6,  8,  5,  7,  6,  7,  6,  8,  6,  7, 10,  8,
        6,  8,  6,  7,  6,  7,  6,  8, 10,  9, 10,  9, 10,  9, 10,  9, 10,
        9, 10,  9, 10,  9, 10,  9, 10,  9, 10,  9, 10,  9], dtype=int64)
```

[9]:
```
supervised = pd.DataFrame(dataset)
```

[10]:
```
supervised['Cluster_group']= labels
```

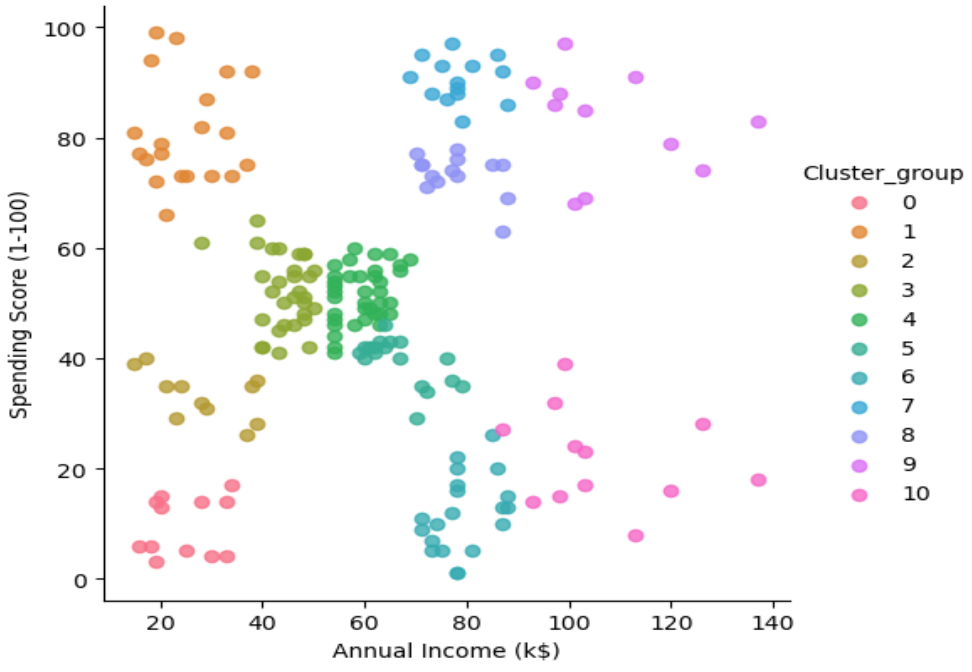
[11]:
```
supervised
```

[11]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)	Cluster_group
0	1	Male	19	15	39	2
1	2	Male	21	15	81	1
2	3	Female	20	16	6	0
3	4	Female	23	16	77	1
4	5	Female	31	17	40	2
...	...	...	...	...	...	...
195	196	Female	35	120	79	9
196	197	Female	45	126	28	10
197	198	Male	32	126	74	9
198	199	Male	32	137	18	10

[12]:
```
import seaborn as sns
facet = sns.lmplot(data=supervised, x=supervised.columns[3], y=supervised.columns[4], hue=supervised.columns[5],fit_reg=False,legend=True, legend_out=Tru
```

```
C:\Users\anandha rishi\anaconda3\Lib\site-packages\seaborn\regression.py:582: UserWarning: legend_out is deprecated from the `lmplot` function signature.
Please update your code to pass it using `facet_kws`.
  warnings.warn(msg, UserWarning)
```



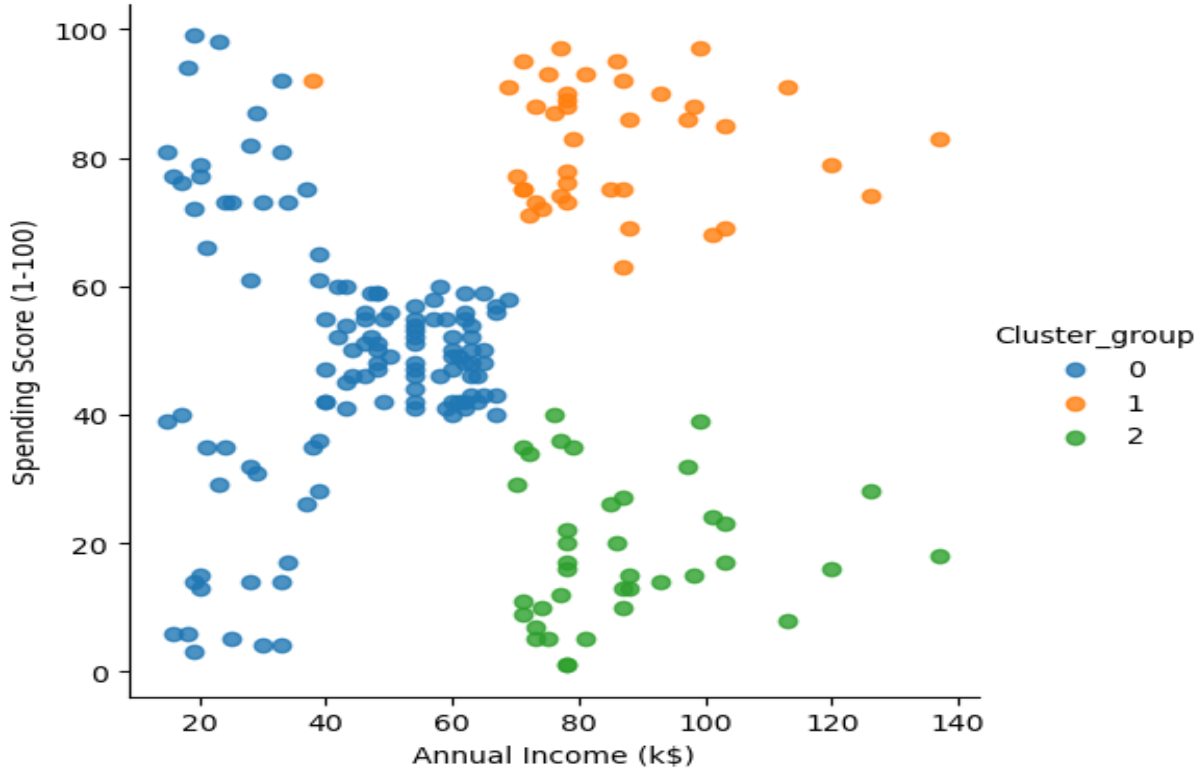
2. Mean Shift Clustering

Mean Shift is a clustering algorithm that locates the densest regions of data points to identify clusters. It is a non-parametric, centroid-based algorithm that does not require specifying the number of clusters beforehand. The algorithm works by shifting each data point towards the mean (center) of the points within a certain radius, called the bandwidth, iteratively moving points closer to areas with higher point density.

- ***Steps:***
  - ✓ ***Choose a random point and define a window around it (using bandwidth).***
  - ✓ ***Compute the mean of points inside the window.***
  - ✓ ***Shift the window to the mean.***
  - ✓ ***Repeat until convergence.***

Clusters are identified where points converge to the same mean. This algorithm is highly effective in identifying arbitrarily shaped clusters.

```
:
from sklearn.cluster import MeanShift, estimate_bandwidth
# Fit mean shift Clustering model
bandwidth = estimate_bandwidth(X, quantile=0.2)
mean_shift = MeanShift(bandwidth=bandwidth, bin_seeding=True)
mean_shift.fit(X)
```

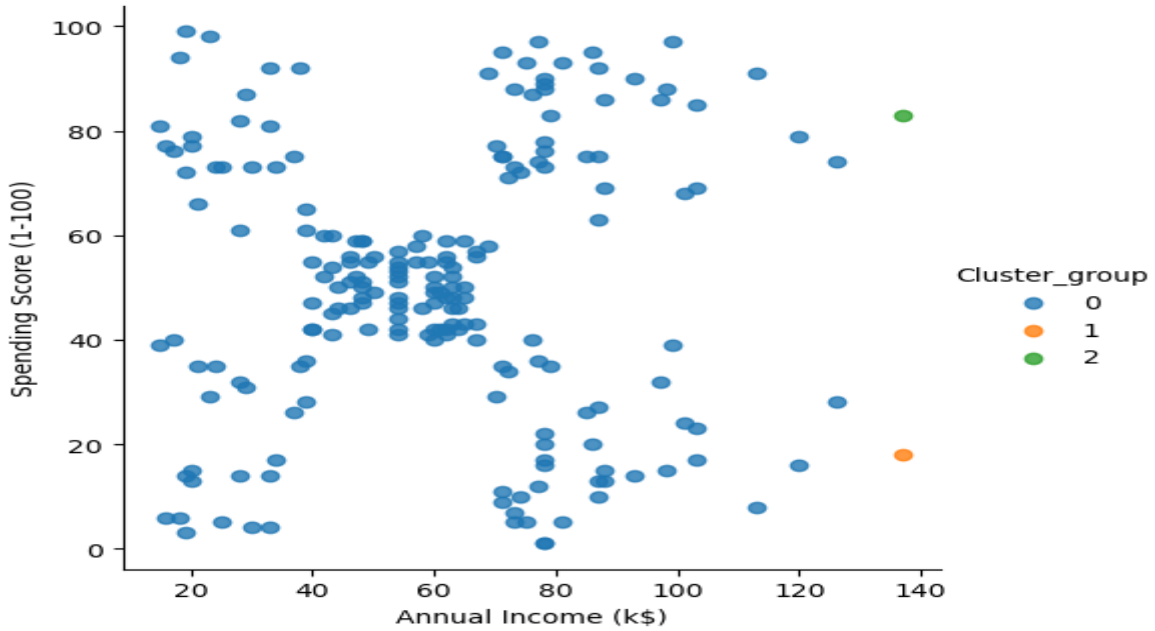


3. Spectral Clustering

Spectral Clustering is an advanced clustering technique that uses the eigenvalues of a similarity matrix to perform dimensionality reduction before clustering in fewer dimensions. This algorithm is particularly useful for identifying clusters with complex structures, which are difficult to detect with traditional methods like k-means.

Spectral Clustering is especially powerful for data that form non-convex shapes or have complex spatial distributions.

```
from sklearn.cluster import SpectralClustering
# Fit Spectral Clustering model
n_clusters = 3  # Define the number of clusters
spectral = SpectralClustering(n_clusters=n_clusters, affinity='rbf', random_state=0)
labels = spectral.fit_predict(X)
```

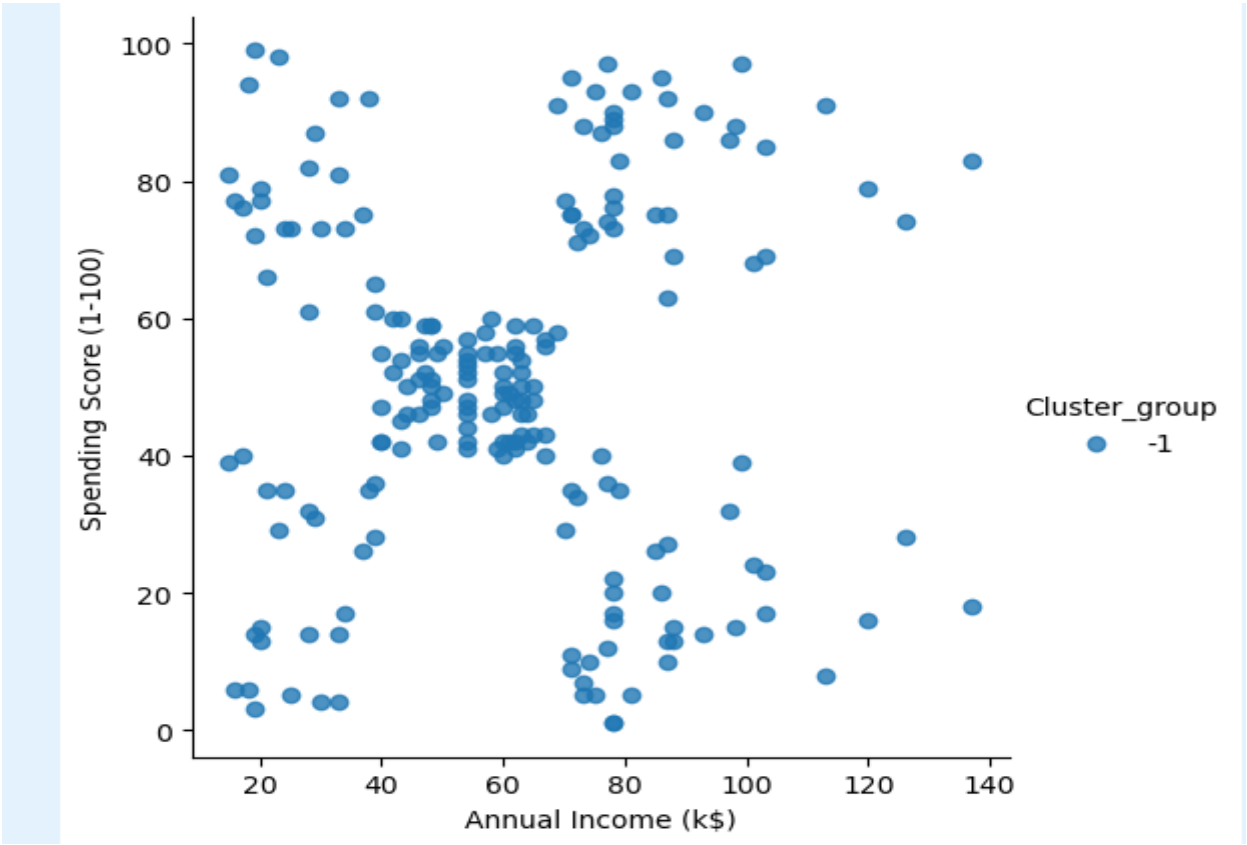


4. DBSCAN Clustering

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a powerful clustering algorithm that groups together closely packed data points while marking outliers as noise. It does not require specifying the number of clusters beforehand, making it effective for identifying clusters of arbitrary shapes and sizes.

DBSCAN is highly effective for discovering clusters of various shapes and sizes, especially when data includes noise.

```
from sklearn.cluster import DBSCAN
# Fit DBSCAN model
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)
```

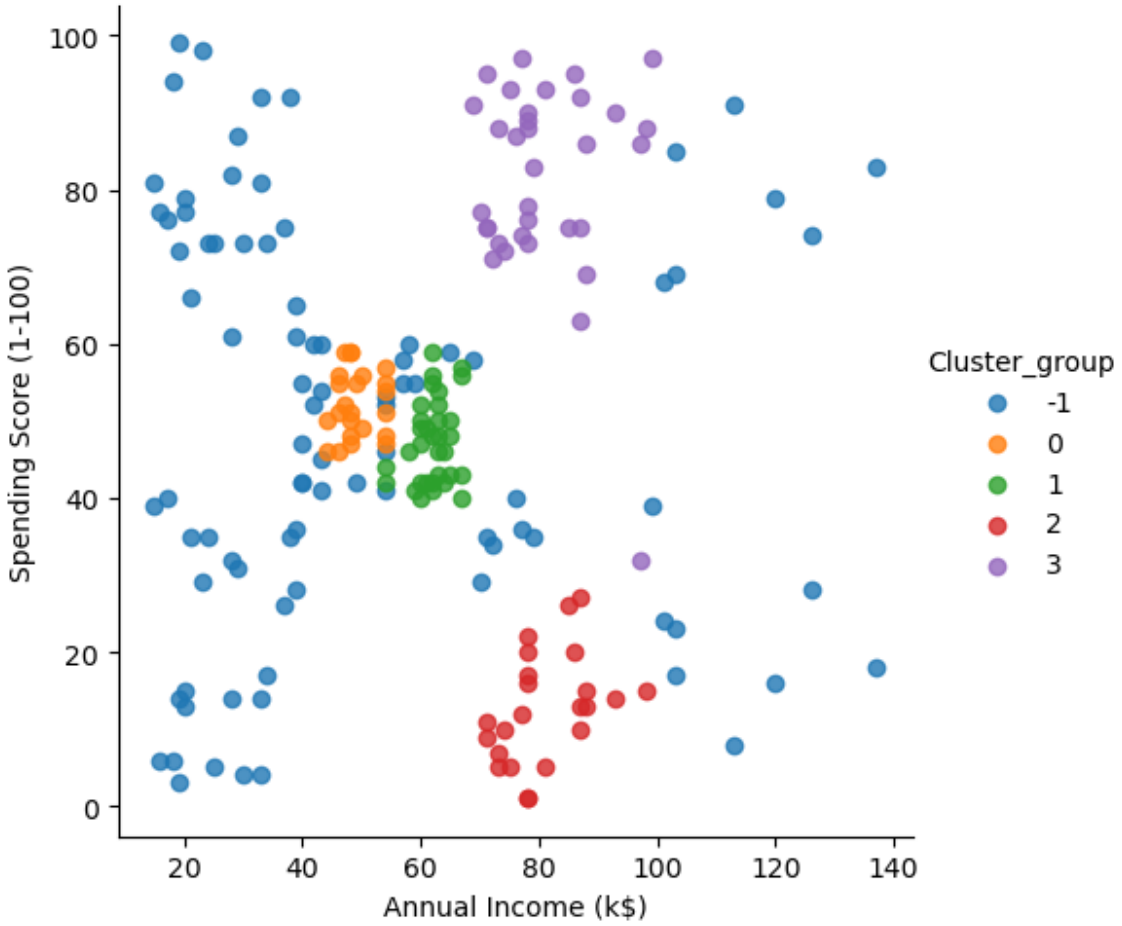


5. Optics Clustering

OPTICS (Ordering Points To Identify the Clustering Structure) is a density-based clustering algorithm that is an extension of DBSCAN. Unlike DBSCAN, which assigns points to specific clusters, OPTICS provides a clustering order of the data points, which helps identify clusters of varying densities.

OPTICS is effective in finding clusters of varying shapes and densities, making it suitable for complex datasets.

```
from sklearn.cluster import OPTICS
# Fit OPTICS model
optics_model = OPTICS(min_samples=10, xi=0.05, min_cluster_size=0.1)
labels = optics_model.fit_predict(X)
```



6. BIRCH Clustering

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a hierarchical clustering algorithm that is particularly well-suited for large datasets. It incrementally and dynamically clusters incoming data points using a tree structure called a Clustering Feature (CF) tree.

BIRCH is efficient and scales well with large datasets, making it ideal for real-time clustering.

```
from sklearn.cluster import Birch
# Fit BIRCH model
birch_model = Birch(threshold=0.3, n_clusters=3)
labels = birch_model.fit_predict(X)
```