

Clustering

K-Means

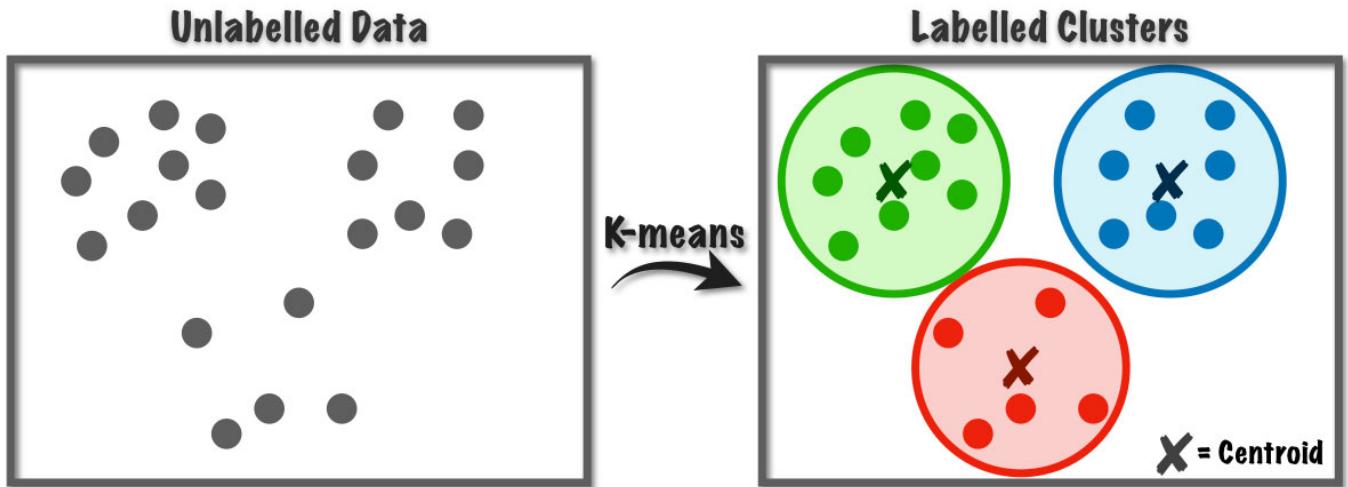


image from: [K-means: A Complete Introduction](#)

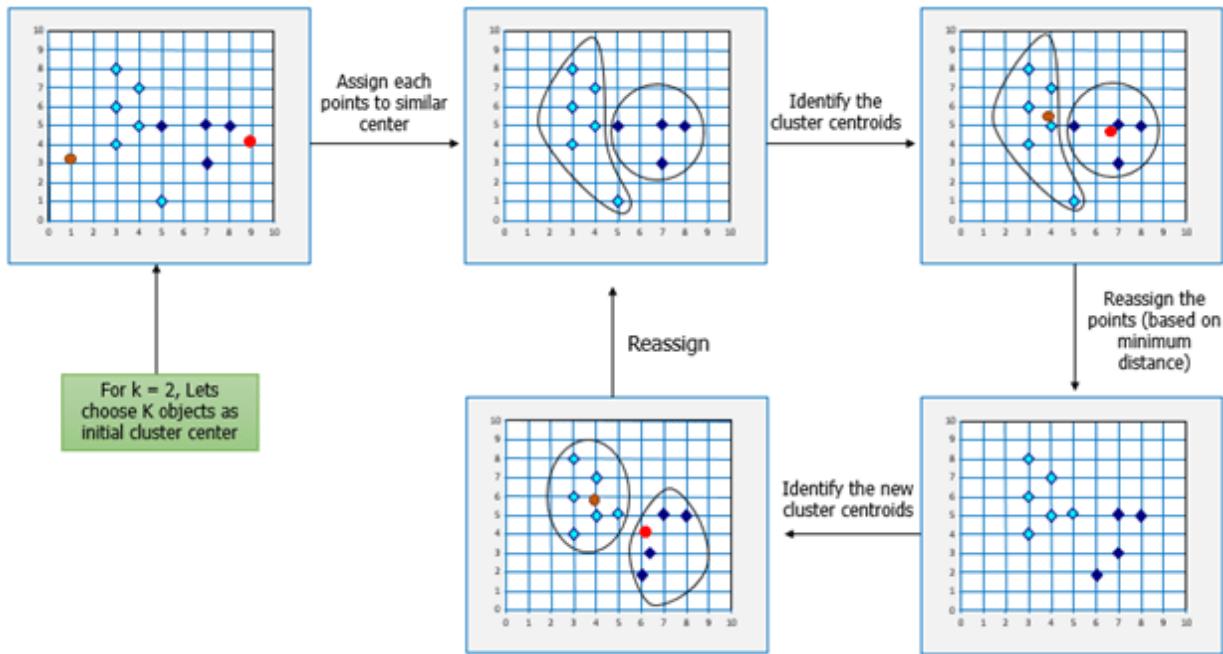
- by  [scikit-learn \(KMeans\)](#)

K-Means Clustering Algorithm

K-means เป็นอัลกอริทึมการจัดกลุ่มที่ใช้กันอย่างแพร่หลาย ชึ่งมุ่งหวังที่จะแบ่งชุดข้อมูลเป็นกลุ่ม K กลุ่ม แต่ละจุดข้อมูลจะอยู่ในกลุ่มที่มีค่าเฉลี่ย (หรือ centroid) ใกล้ที่สุด อัลกอริทึมจะทำการกำหนดกลุ่มข้อมูลและปรับปรุงค่า centroid ไปเรื่อยๆ จนกว่าจะมีค่าความแตกต่างน้อยมาก

Algorithm Steps

1. **Initialization:** เลือกจำนวนกลุ่ม K และกำหนดค่ากลางของ K โดยเลือกแบบสุ่มหรือใช้กลยุทธ์เฉพาะ
2. **Assignment Step:** สำหรับแต่ละจุดข้อมูล คำนวณระยะห่างระหว่างจุดข้อมูลและแต่ละ centroid กำหนดให้จุดข้อมูลอยู่ในกลุ่มที่ใกล้ centroid ที่สุด
3. **Update Step:** เมื่อทุกจุดข้อมูลถูกกำหนดให้อยู่ในกลุ่ม ทำการคำนวณค่า centroid ของแต่ละกลุ่มใหม่ โดยใช้ค่าเฉลี่ยของจุดข้อมูลทั้งหมดในกลุ่มนั้น
4. **Convergence Check:** ตรวจสอบว่าค่า centroid เป็นไปตามที่ต้องการ หรือไม่ หากค่า centroid เป็นไปตามที่ต้องการ ให้ดำเนินการต่อไป 3 หากไม่มีการเปลี่ยนแปลง อัลกอริทึมจะเสร็จสิ้น และกลุ่มที่ว่างเปล่าจะถูกลบออก
5. **Termination:** อัลกอริทึมจะสิ้นสุดลงเมื่อค่า centroid ไม่เปลี่ยนแปลงอย่างมีนัยสำคัญหรือเมื่อมีจำนวนการทำซ้ำที่กำหนดไว้



(image from: <https://www.edureka.co/blog/k-means-clustering/>)

Choosing the Number of Clusters (K)

การเลือกค่าที่เหมาะสมสำหรับ K มีความสำคัญ มีกลุ่มน้อยเกินไปอาจไม่สามารถจับแนวโน้มในข้อมูลได้ ในขณะที่มีกลุ่มมากเกินไปอาจทำให้มีการ overfitting วิธีการเข่นวิธี elbow method และ silhouette score สามารถช่วยในการกำหนดค่า K ที่เหมาะสม

Use Cases

การจัดกลุ่มด้วย K-means ใช้ในหลายๆ ด้าน:

- การแบ่งกลุ่มลูกค้าเพื่อการตลาดที่เป็นเป้าหมาย
- การจัดกลุ่มบทความข่าวหรือเอกสารที่คล้ายกัน
- การบีบอัดภาพเพื่อลดพื้นที่การจัดเก็บ
- การตรวจสอบข้อมูลผิดปกติเพื่อระบุรูปแบบที่ไม่เหมือนธรรมชาติ

Conclusion

K-means เป็นอัลกอริทึมที่มีประสิทธิภาพในการจัดกลุ่มจุดข้อมูลเข้าสู่กลุ่มตามความคล้ายคลึง โดยการกำหนดข้อมูลให้อยู่ในกลุ่มที่มีค่า centroid ใกล้ที่สุด ด้วยการทำซ้ำในขั้นตอนการกำหนดและการปรับปรุง อัลกอริทึม K-means จะสร้างกลุ่มที่ลดระยะห่างระหว่างจุดข้อมูลในกลุ่มเดียวกันและเพิ่มระยะห่างระหว่างกลุ่มต่างๆ เป็นเทคนิคที่ใช้ในการวิเคราะห์ข้อมูลที่หลากหลายด้วย การจัดกลุ่มข้อมูลก่อนการวิเคราะห์และการประมวลผลข้อมูล

สร้าง DataFrame

- ใช้ dataset จากไฟล์ OnlineRetail (INTERM DS).csv
- อัปโหลดขึ้น sample_data
- เรียกใช้ pandas

```
import pandas as pd
```

- อ่านไฟล์ OnlineRetail (INTERM DS).csv จาก sample_data โดยคลิกขวาที่ชื่อไฟล์ แล้วเลือก Copy path

```
df = pd.read_csv('/content/sample_data/OnlineRetail (INTERM DS).csv')
df.head()
```

	CustomerID	Amount	Frequency	Recency
0	12346	0.00	2	325
1	12347	4310.00	182	1
2	12348	1797.24	31	74
3	12349	1757.55	73	18
4	12350	334.40	17	309

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4293 entries, 0 to 4292
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   CustomerID  4293 non-null   int64  
 1   Amount       4293 non-null   float64 
 2   Frequency    4293 non-null   int64  
 3   Recency      4293 non-null   int64  
dtypes: float64(1), int64(3)
memory usage: 134.3 KB
```

```
df = pd.read_csv('/content/sample_data/OnlineRetail (INTERM DS).csv',
                 index_col='CustomerID')
df.head()
```

```
Amount  Frequency  Recency
```

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

	count	mean	std	min	25%	50%	75%	max
Amount	4293.0	1270.411464	1755.551155	-4287.63	289.36	632.97	1518.43	13677.59
Frequency	4293.0	77.483578	100.270448		1.00	17.00	40.00	97.00
Recency	4293.0	92.548567	101.006845		0.00	17.00	50.00	145.00

- ปรับสเกลฟีเจอร์ให้เท่าเทียมกัน
- เรียกใช้ StandardScaler

```
from sklearn.preprocessing import StandardScaler
```

- ปรับสเกลฟีเจอร์

```
dfs = StandardScaler().fit_transform(df)
dfs

array([[-0.72373821, -0.75288754,  2.30161144],
       [ 1.73161722,  1.04246665, -0.90646561],
       [ 0.30012791, -0.46363604, -0.18365813],
       ...,
       [-0.67769602, -0.70301659,  0.86589794],
       [-0.6231313 , -0.64317145, -0.84705678],
       [ 0.32293822, -0.07464263, -0.50050524]])
```

- สร้าง DataFrame ขึ้นมาใหม่

```
dfn = pd.DataFrame(dfs, columns=df.columns, index=df.index)
dfn
```

CustomerID	Amount	Frequency	Recency
12346	-0.723738	-0.752888	2.301611
12347	1.731617	1.042467	-0.906466
12348	0.300128	-0.463636	-0.183658

dfn.describe().T

	count	mean	std	min	25%	50%	75%
Amount	4293.0	1.986143e-17	1.000116	-3.166350	-0.558893	-0.363143	0.141293
Frequency	4293.0	-4.882602e-17	1.000116	-0.762862	-0.603275	-0.373868	0.194661
Recency	4293.0	5.379138e-17	1.000116	-0.916367	-0.748042	-0.421293	0.519346

k-mean = ស្ថិតិមាលាបងចែក

ចំណាំ 2 ចំណាំ → ចំណែនបងចែកការងារទៅ 4 ក្រុងក្រោម ក្នុងក្រោម ក្នុងក្រោម

สร้าง K-Means model

- เรียกใช้ KMeans library

from sklearn.cluster import KMeans

នៅក្នុង

..ការបង្រួចរាយកម្ម-ឱ្យលើ ដែលបាន ឱ្យពារ៉ា, ហើយ K-Means នេះ មិនមែន:

- ทดลองสร้าง model เพื่อបង្រៀន 4 ក្រុង (នៅក្នុងក្រោម)

(ក្រោម ឱ្យលើ)

kmeans = KMeans(n_clusters=4, n_init='auto', random_state=233)

នៅក្នុង = auto

(នៅក្នុង ឱ្យក្រុងដែលបានការការណ៍)

kmeans.fit(dfn) នៅក្នុងនៅក្នុង model

```
▼
          KMeans
KMeans(n_clusters=4, n_init='auto', random_state=233)
```

- Centroids - ជូនការងារក្នុង ចំណាំនៃ model KMeans នៅក្នុងនៅក្នុង x_coeff, intercept នៃ LR

kmeans.cluster_centers_

```
array([[ 3.13735223,  3.0210413 , -0.73921365],
       [-0.32333932, -0.34121055, -0.43962905],
       [-0.50370409, -0.51878687,  1.57016042],
       [ 0.84062643,  0.94783153, -0.62633315]]) .. 1
.. 2
.. 3
.. 4
```

នៅក្នុងនៅក្នុង

- ผลลัพธ์การแบ่งกลุ่ม

kmeans.labels_ ឧប្បរជាន់ក្នុងក្នុង

```
array([2, 3, 1, ..., 2, 1, 1], dtype=int32)
```

① • ค่า WCSS = ค่า $\sum_{\text{within cluster}} \sum_{\text{sum}} \text{sq.}$ $= \sum_{\text{all data}} \text{distance from centroid}^2$ กับ ค่า $\sum_{\text{all data}} \text{distance from centroid}^2$ (Centroid)
แล้วตามารวมกันทุกกลุ่ม คือ ค่า WCSS ค่าต่อไป = 9 กลุ่ม x 3 ค่า (เท่ากันกัน)

kmeans.inertia_

3361.2473491495375 \rightarrow ฟังก์ชันนี้ได้รับค่าจาก dfn. ค่าเดียวกัน

② ∵ ถ้า: ยังไงก็ตามที่ต้องการก็ต้อง.. \downarrow

หาจำนวนกลุ่ม (k) ที่เหมาะสมที่สุด

หาดูดังนี้

- Elbow point evaluation

◦ Kneed

▪ Documentation

วิธีนี้จะ, ปัจ

= ค่าที่น้อยที่สุด ก็คือ ค่าที่น้อยที่สุด คือ 8

- ทดลองแบ่งกลุ่มตั้งแต่ 1 - 10 กลุ่ม

- เก็บค่า WCSS ของแต่ละ k ไว้

```
k_values = range(1, 11)  $\downarrow$  1 - 10
scores = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init='auto', random_state=233)
    kmeans.fit(dfn)
    scores.append(kmeans.inertia_)
```

scores

[12879.000000000015,	กลุ่ม 1	- มาก
7661.7058693207455,	2	
4433.747689347625,	3	
3361.2473491495375,	4	
2802.8241284834685,	5	
2392.1847270769094,	6	
2039.3322070983968,	7	
1827.1894987451742,	8	
1659.4527155763499,	9	
1468.4115541590068]	10	

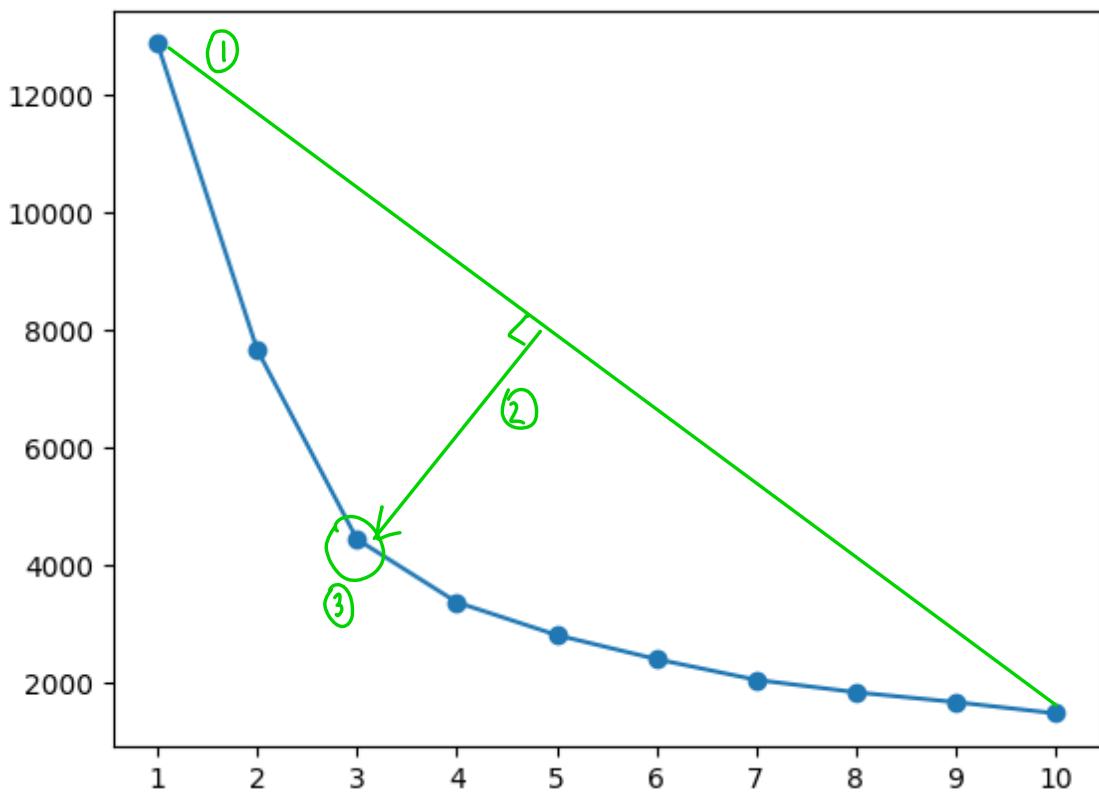
∴ ถ้าค่ามีลดลง ค่าจะต่ำ

↓
น้อย

- สร้างกราฟแสดงความสัมพันธ์ของ WCSS กับ จำนวนกลุ่ม

```
import matplotlib.pyplot as plt
```

```
plt.plot(k_values, scores, 'o-')
plt.xticks(k_values)
plt.show()
```



- ใช้ kneed library ในการหาจุดหักโค้ง
- ต้องติดตั้งเพิ่ม

```
%pip install kneed
```

แบบภาษาไทย
แบบภาษาอังกฤษ

① หาจุดหักโค้งแล้วตัดต่อเป็นช่วงๆ
② ถ้าจุดหักโค้งอยู่ในช่วงที่ 3 ให้ตัดช่วงที่ 3 ออก
③ ต่อจุดหักโค้งที่ 3 (ที่ตัดออกไปแล้ว) ให้เป็นช่วงเดียว

```
%pip install kneed
```

```
Collecting kneed
  Downloading kneed-0.8.5-py3-none-any.whl (10 kB)
Requirement already satisfied: numpy>=1.14.2 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages
Installing collected packages: kneed
Successfully installed kneed-0.8.5
```

- เรียกใช้ KneeLocator

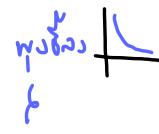
```
from kneed import KneeLocator
```

- หาจุดหักโค้ง

```
kneedle = KneeLocator(k_values, scores, curve='convex', direction='decreasing')
```

convex
concave

increasing = starting downward



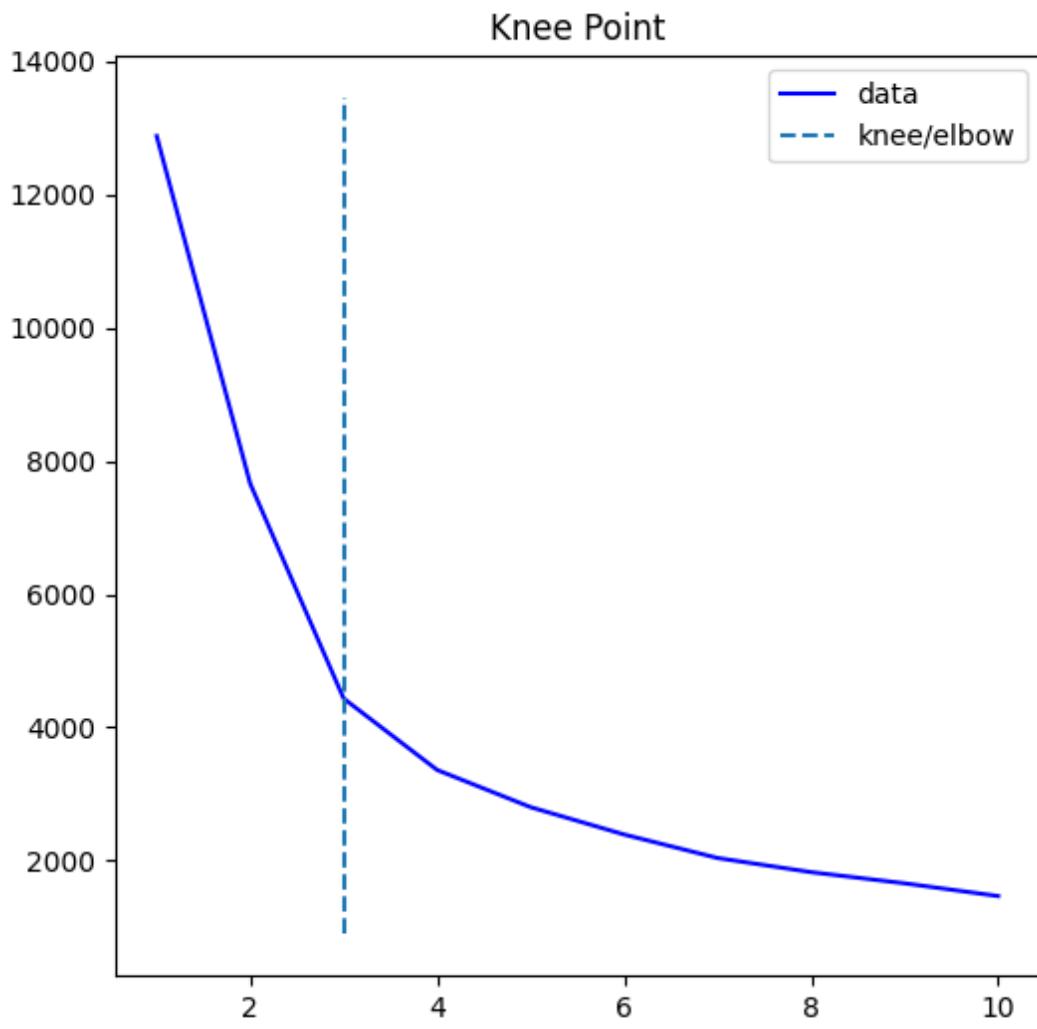
=พูดเรื่องมุมเรืองใน จุดเรือง

kneedle.elbow

3 07a 9am 3 Ամսանոն

- ទូរសព្ទរបស់ក្រសួង

```
kneedle.plot_knee()
```



- แบ่ง cluster ออกเป็น 3 กลุ่ม

```
kmeans = KMeans(n_clusters=3, n_init='auto', random_state=555, verbose=1)
kmeans.fit(dfn)
```

↓
9 ນີ້ແກຕັງຮາຍວ່າ: ເປົ້າດັບດັບມາ

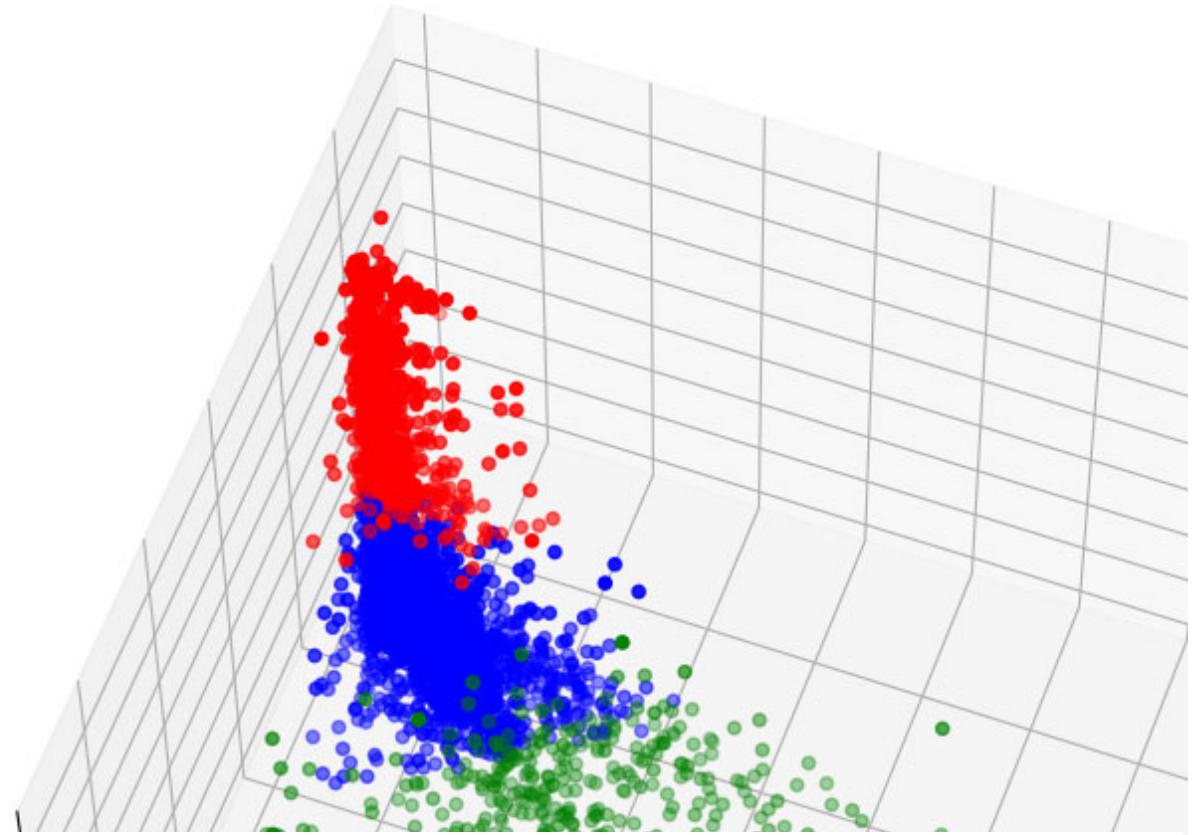
```
Initialization complete
Iteration 0, inertia 8117.517816905875.
Iteration 1, inertia 6720.330346227932.
Iteration 2, inertia 6579.922763488956.
Iteration 3, inertia 6473.332767624631.
Iteration 4, inertia 6378.715106408788.
Iteration 5, inertia 6282.052494901052.
Iteration 6, inertia 6195.628114341709.
Iteration 7, inertia 6137.526622296971.
Iteration 8, inertia 6087.618719987751.
Iteration 9, inertia 6036.940814911644.
Iteration 10, inertia 5983.321089166176.
Iteration 11, inertia 5905.746327139341.
Iteration 12, inertia 5754.337758917446.
Iteration 13, inertia 5374.846580893577.
Iteration 14, inertia 4816.080357034312.
Iteration 15, inertia 4527.477172093815.
```

```
kmeans.labels_
```

```
array([2, 0, 1, ..., 2, 1, 1], dtype=int32)
Iteration 20, inertia 4434.229849028845.
```

- แสดงกลุ่มด้วยกราฟ

```
labels = kmeans.labels_
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(dfn.Amount[labels==0], dfn.Frequency[labels==0], dfn.Recency[labels==0], c='r')
ax.scatter(dfn.Amount[labels==1], dfn.Frequency[labels==1], dfn.Recency[labels==1], c='g')
ax.scatter(dfn.Amount[labels==2], dfn.Frequency[labels==2], dfn.Recency[labels==2], c='b')
ax.set_xlabel(dfn.Amount.name)
ax.set_ylabel(dfn.Frequency.name)
ax.set_zlabel(dfn.Recency.name)
ax.view_init(60, 20)
plt.show()
```



```
kmeans.cluster_centers_
```

```
array([[-0.50237552, -0.51846606,  1.54212771],
       [ 2.08754921,  2.10491086, -0.70109576],
       [-0.1786575 , -0.17548993, -0.47650124]])
```

```
dfn.describe().T
```

	count	mean	std	min	25%	50%	75%
Amount	4293.0	1.986143e-17	1.000116	-3.166350	-0.558893	-0.363143	0.141293
Frequency	4293.0	-4.882602e-17	1.000116	-0.762862	-0.603275	-0.373868	0.194661
Recency	4293.0	5.379138e-17	1.000116	-0.916367	-0.748042	-0.421293	0.519346

Evaluating Cluster Quality

Silhouette Score

Silhouette score เป็นตัวชี้วัดที่ใช้กันอย่างแพร่หลายในการประเมินคุณภาพของกลุ่ม มันวัดความคล้ายคลึงของวัตถุต่างๆ กับกลุ่มของตัวเอง (cohesion) โดยเปรียบเทียบกับกลุ่มอื่น (separation)

Silhouette score อยู่ในช่วง -1 ถึง 1 โดยค่าสูงแสดงถึงว่าวัตถุเข้ากันได้ดีกับกลุ่มของตัวเองและเข้ากันไม่ดีกับกลุ่มใกล้เคียง

Formula: Silhouette score for a data point $i = (b(i) - a(i)) / \max(a(i), b(i))$

Where:

- a(i) คือระยะทางเฉลี่ยจาก i ไปยังจุดข้อมูลทั้งหมดในกลุ่มเดียวกัน
- b(i) คือระยะทางเฉลี่ยที่น้อยที่สุดจาก i ไปยังจุดข้อมูลทั้งหมดในกลุ่มที่แตกต่างกัน โดยพยายามที่จะเลือกกลุ่มที่มีระยะทางเฉลี่ยน้อยที่สุด
- เรียกใช้ silhouette_score library

```
from sklearn.metrics import silhouette_score

silhouette_score(dfn, kmeans.labels_)

0.5085084004374336
```

▼ Within-Cluster Sum of Squares (WCSS)

ค่าความแตกต่างภายในกลุ่ม (WCSS) วัดความเข้มงวดของกลุ่มโดยการคำนวณผลรวมของระยะทางที่ยกกำลังสองระหว่างแต่ละจุดข้อมูลและศูนย์กลางของกลุ่ม ค่า WCSS ที่ต่ำกว่าแสดงถึงกลุ่มที่เข้าใกล้กันมากขึ้น การจัดกลุ่มมีจุดมุ่งหมายในการลดค่า WCSS

Formula: WCSS สำหรับกลุ่ม k = \sum ระยะทาง(จุดข้อมูลถึงศูนย์กลาง) 2 สำหรับทุกจุดข้อมูลในกลุ่ม k

```
kmeans.inertia_
```

```
4433.747689347625
```

▼ Davies-Bouldin Index

ดัชนี Davies-Bouldin วัดความคล้ายคลึงเฉลี่ยระหว่างแต่ละกลุ่มและกลุ่มที่คล้ายกันที่สุดของกลุ่มนี้ออกจากความกระจายภายในกลุ่มและความแยกกันระหว่างกลุ่ม ค่าดัชนี Davies-Bouldin ที่ต่ำกว่าแสดงถึงการแยกกันของกลุ่มที่ดีขึ้น

Formula: Davies-Bouldin Index = $(1/n) * \sum \max((R_i + R_j) / D_{ij})$

Where:

- n คือจำนวนกลุ่ม
- R_i คือระยะทางเฉลี่ยจากทุกจุดในกลุ่ม i ไปยังศูนย์กลางของกลุ่ม i
- R_j คือระยะทางเฉลี่ยจากทุกจุดในกลุ่ม j ไปยังศูนย์กลางของกลุ่ม j
- D_{ij} คือระยะทางระหว่างศูนย์กลางของกลุ่ม i และ j
- เรียกใช้ davies_bouldin_score library

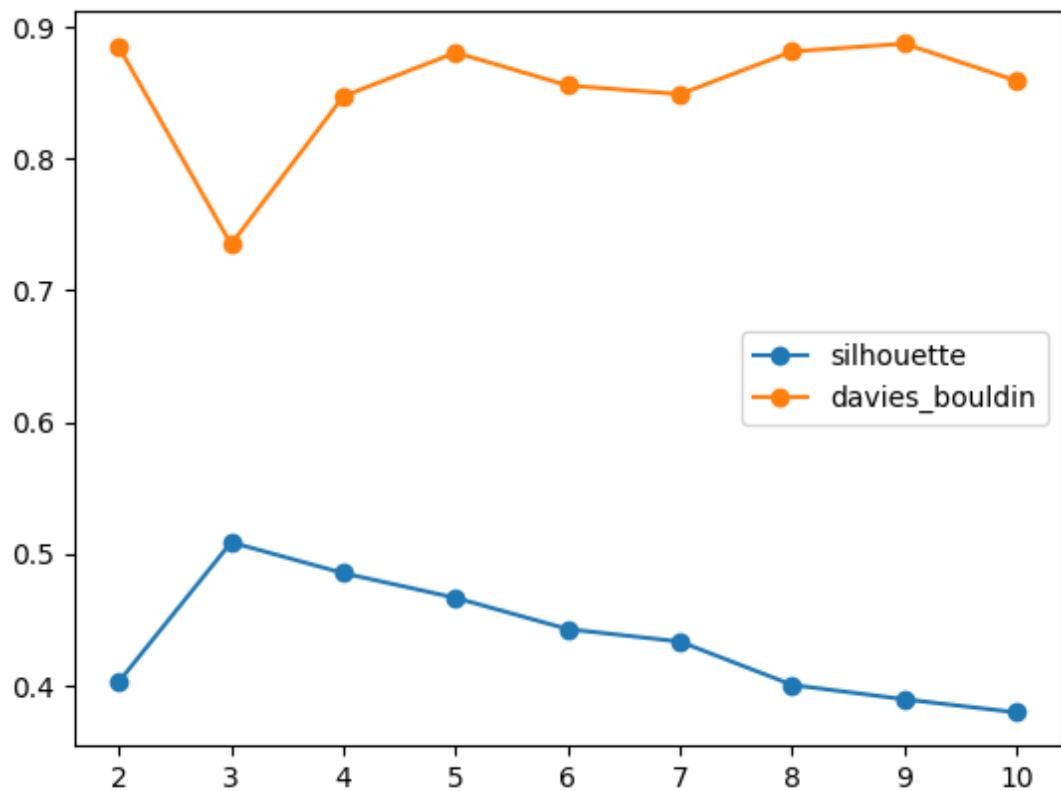
```
from sklearn.metrics import davies_bouldin_score
```

```
davies_bouldin_score(dfn, kmeans.labels_)
```

```
0.7351697985355204
```

- แสดงกราฟเปรียบเทียบ Silhouette Score และ Davies-Bouldin Index

```
k_values = range(2, 11)
sil_scores = []
dbi_scores = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, n_init='auto', random_state=555)
    kmeans.fit(dfn)
    sil_scores.append(silhouette_score(dfn, kmeans.labels_))
    dbi_scores.append(davies_bouldin_score(dfn, kmeans.labels_))
plt.plot(k_values, sil_scores, 'o-')
plt.plot(k_values, dbi_scores, 'o-')
plt.xticks(k_values)
plt.legend(['silhouette', 'davies_bouldin'])
plt.show()
```



```
kmeans = KMeans(n_clusters=3, n_init='auto', random_state=555)
kmeans.fit(dfn)
```

```
▼ KMeans
KMeans(n_clusters=3, n_init='auto', random_state=555)
```

```
kmeans.labels_
```

```
array([0, 1, 2, ..., 0, 2, 2], dtype=int32)
```

df

CustomerID	Amount	Frequency	Recency
12346	0.00	2	325
12347	4310.00	182	1
12348	1797.24	31	74
12349	1757.55	73	18
12350	334.40	17	309
...
18278	173.90	9	73
18280	180.60	10	277
18281	80.82	7	180
18282	176.60	13	7
18287	1837.28	70	42

4293 rows × 3 columns

```
df['label'] = kmeans.labels_  
df
```

new col. label

	CustomerID	Amount	Frequency	Recency	label
0	12346	0.00	2	325	0
1	12347	4310.00	182	1	1
2	12348	1797.24	31	74	2
3	12349	1757.55	73	18	2
4	12350	334.40	17	309	0
...
4288	18278	173.90	9	73	2
4289	18280	180.60	10	277	0
4290	18281	80.82	7	180	0
4291	18282	176.60	13	7	2
4292	18287	1837.28	70	42	2

4293 rows × 5 columns

```
kmeans.cluster_centers_
```

```
array([[-0.50237552, -0.51846606,  1.54212771],  
      [ 2.08754921,  2.10491086, -0.70109576],  
      [-0.1786575 , -0.17548993, -0.47650124]])
```

```
df[df.label==1] 
```

	CustomerID	Amount	Frequency	Recency	label
12347	12347	4310.00	182	1	1
12357	12357	6207.67	131	32	1
12359	12359	6245.53	254	7	1
12362	12362	5154.58	274	2	1
12370	12370	3545.69	167	50	1

18223	18223	6315.23	299	1	1
18225	18225	5361.02	286	2	1
18226	18226	5192.10	245	38	1
18229	18229	7276.90	164	11	1
18272	18272	3064.78	170	2	1

492 rows × 4 columns

```
dtt = df.copy()  
dtt['label'] = kmeans.labels_  
dtt.groupby(['label']).mean()
```

	label	Amount	Frequency	Recency
0	0	388.568259	25.502809	248.295880
1	1	4930.260325	288.313008	21.741870
2	2	956.164447	59.842664	44.432492

```
import plotly.express as px
```

```
fig = px.scatter_3d(df, x="Amount", y="Frequency", z="Recency", color='label')  
fig.update_traces(marker_size = 2)  
fig.show()
```