Unsupervised Learning

Agenda

- Unsupervised Learning
- Clustering
- K-Means Clustering (Step by step implementation)

Unsupervised

> *Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.*

Main Goal:

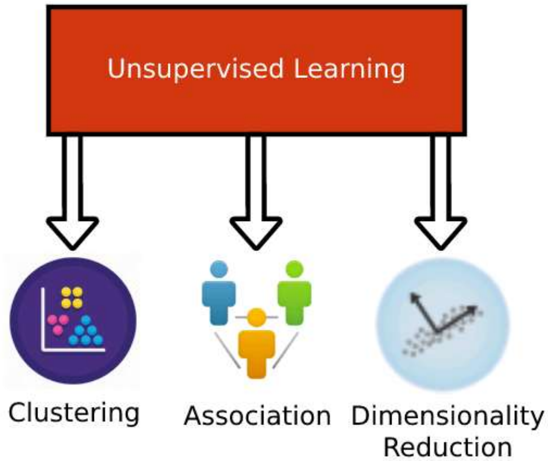
- Unsupervised learning is to ***find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.***

Unsupervised Learning Example

Data Set

Result

Types of unsupervised learning

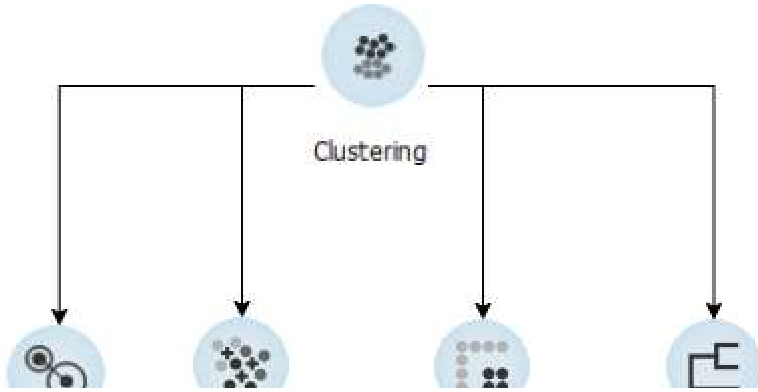


Clustering

Clustering

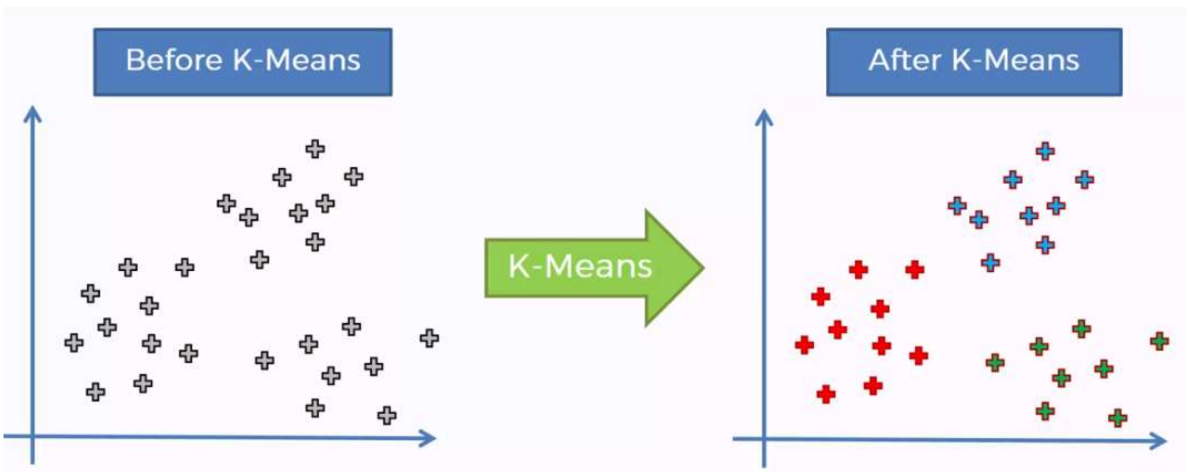
"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."

Types of Clustering Algorithms



KMeans Clustering

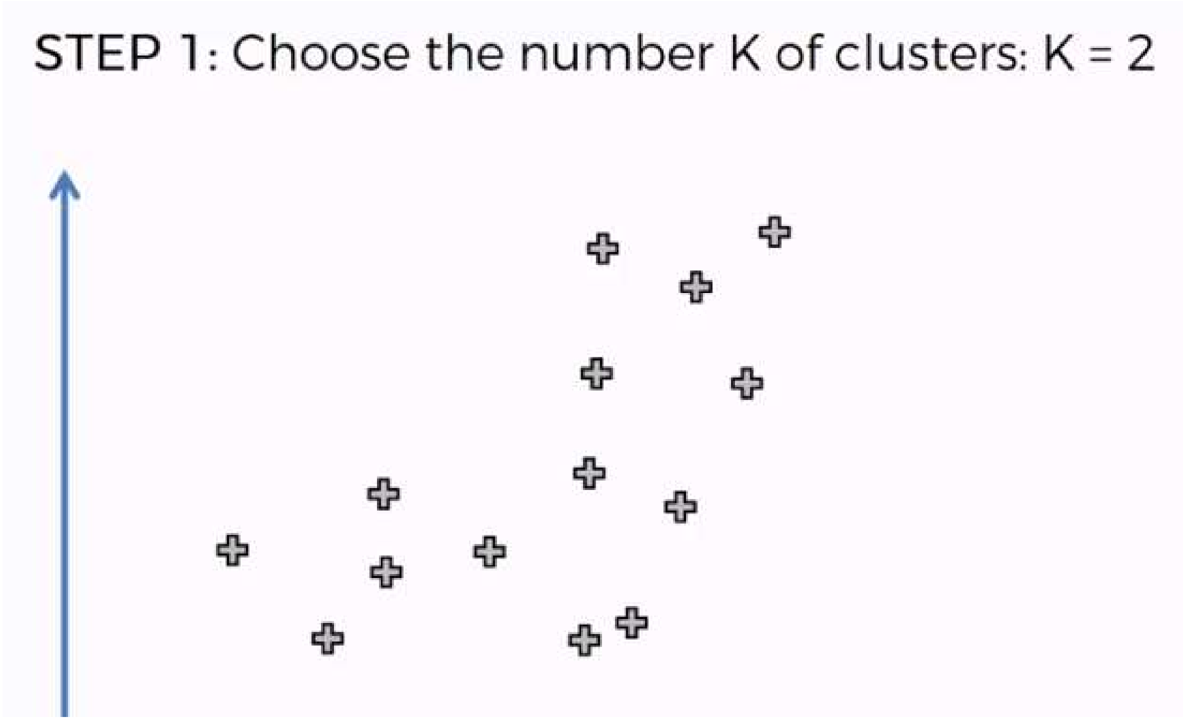
- Find groups in the data, with the number of groups represented by the variable K.
- Iteratively to assign each data point to one of K groups based on the features.



In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import random as rd
import math

import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\1528058\.conda\envs\py38\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Step by Step KMeans Implementation (Step 1)

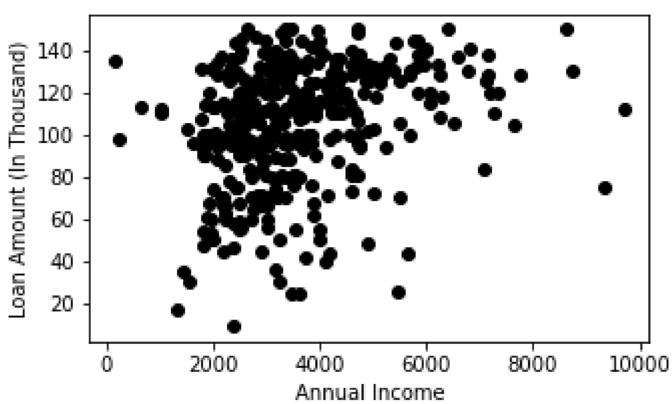


In [2]:
```
data = pd.read_csv('clustering.csv')
data.head(2)
```

Out[2]:

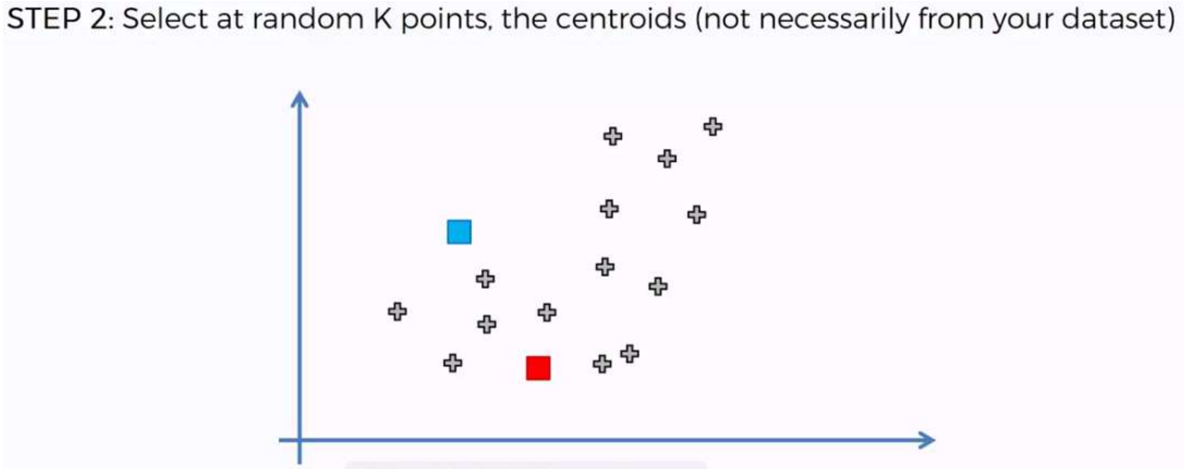
	Loan_ID	ApplicantIncome	LoanAmount
0	LP001003	4583	128.0
1	LP001005	3000	66.0

In [3]:
```
X = data[["LoanAmount", "ApplicantIncome"]]
plt.figure(figsize=(5, 3))
plt.scatter(X["ApplicantIncome"], X["LoanAmount"], c='black')
plt.xlabel('Annual Income')
plt.ylabel('Loan Amount (In Thousand)')
plt.show()
```

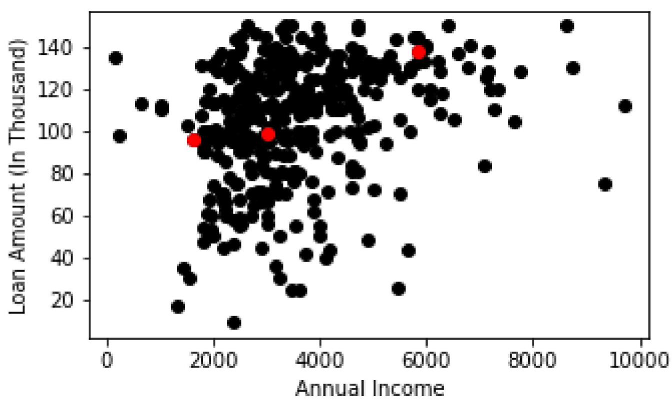


In [4]:
```
K = 3
```

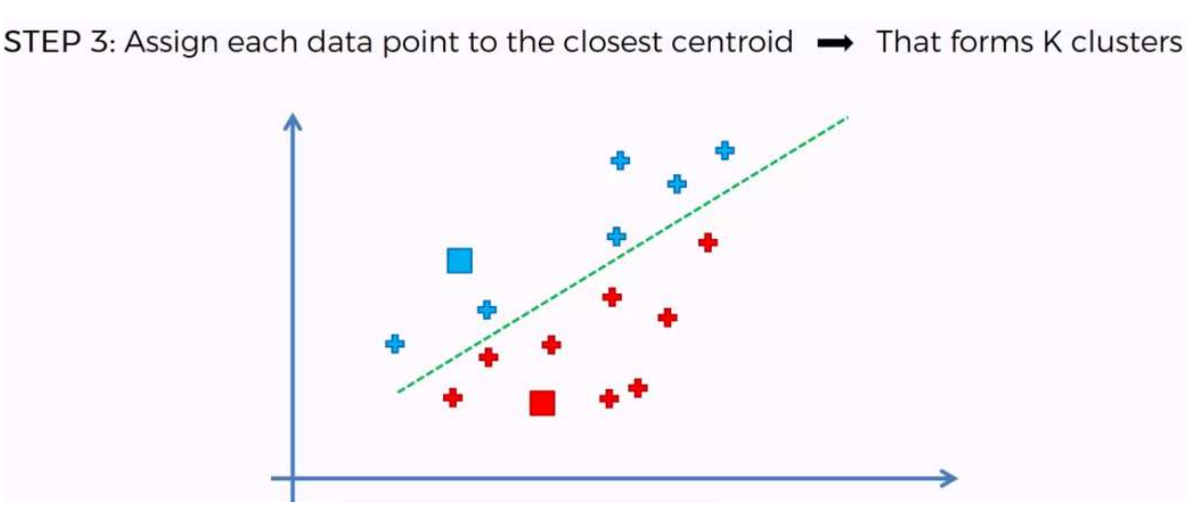
Step by Step KMeans Implementation (Step 2)



In [5]:
```
#Step 2: Select K random points as centorids

centroids = X.sample(n=K, random_state=42).reset_index(drop=True)
plt.figure(figsize=(5, 3))
plt.scatter(X["ApplicantIncome"], X["LoanAmount"], c='black')
plt.scatter(centroids["ApplicantIncome"], centroids["LoanAmount"], c='red')
plt.xlabel('Annual Income')
plt.ylabel('Loan Amount (In Thousand)')
plt.show()
```



Step by Step KMeans Implementation (Step 3)

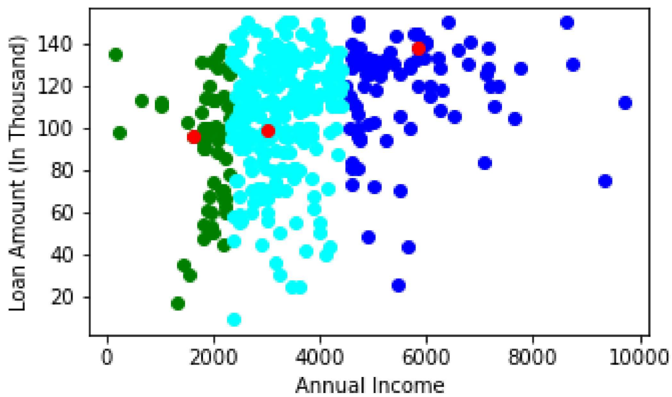


Distance Function (Euclidean Distance)

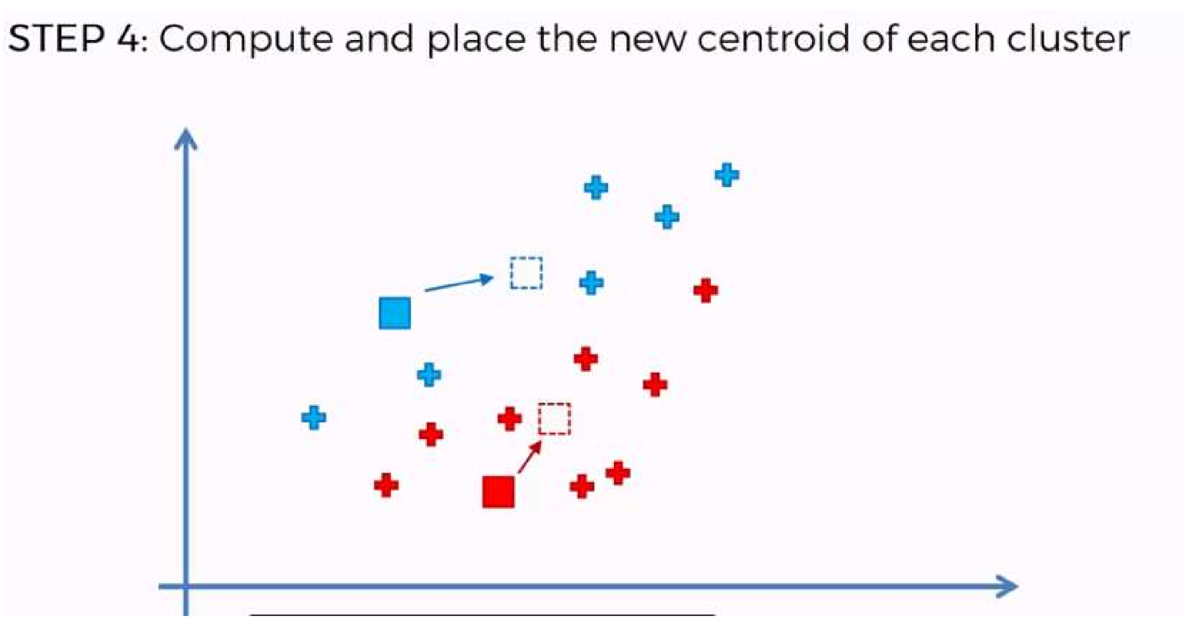
$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2}$$

In [6]:
```
def subtract(x, y):
    assert x.shape ==  y.shape # vectors must be the same length
    return x - y

def sum_of_squares(x):
    return sum(dot(x,x))

def dot(x,y):
    return x * y

def euclidean_distance(x, y):
    return math.sqrt(sum_of_squares(subtract(x,y)))

def assign_cluster_closest_centroid(data_points_df, centroids_df):
    data_points, centroids = data_points_df[["LoanAmount", "ApplicantIncome"]]
    cluster_assignments = []
    for data_point in data_points:
        min_distance = math.inf
        cluster = None
        for i in range(K):
            distance = euclidean_distance(data_point, centroids[i])
            if distance < min_distance:
                min_distance = distance
                cluster = i
        cluster_assignments.append(cluster)
    return cluster_assignments

assignments = assign_cluster_closest_centroid(X, centroids)
X["Cluster"] = assignments
```

In [7]:
```
plt.figure(figsize=(5, 3))
colors = ["blue", "green", "cyan"]
for k in range(K):
    data = X[X["Cluster"] == k]
    plt.scatter(data["ApplicantIncome"], data["LoanAmount"], c=colors[k])
plt.scatter(centroids["ApplicantIncome"], centroids["LoanAmount"], c='red')
plt.xlabel('Annual Income')
plt.ylabel('Loan Amount (In Thousand)')
plt.show()
```

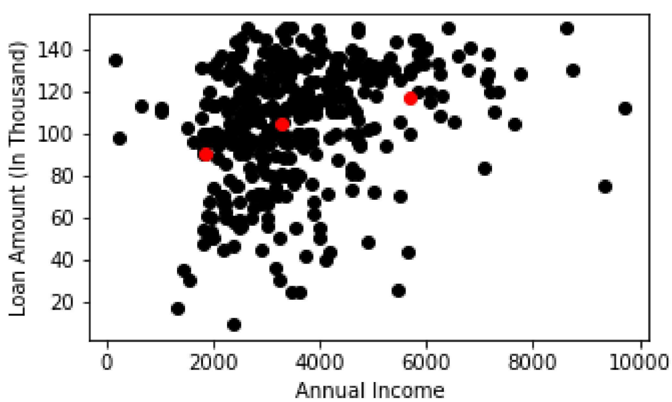


Step by Step KMeans Implementation (Step 4)

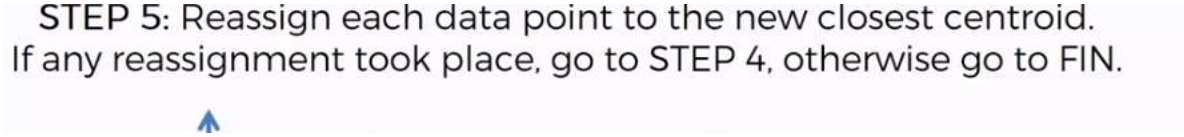


In [8]:
```
def vector_sum(x):
    return sum(x)

def vector_mean(x):
    n = x.shape[0]
    return vector_sum(x) * (1/n)

def calculate_new_centroids(data_points_df, centroids_df):
    data_points, centroids = data_points_df[["LoanAmount", "ApplicantIncome"]]
    assignments = data_points_df["Cluster"].values
    new_centroids = []
    for i in range(K):
        i_points = [p for p, a in zip(data_points, assignments) if a == i]
        if i_points:
            new_centroids.append(vector_mean(np.array(i_points)))
    return pd.DataFrame(new_centroids, columns=["LoanAmount", "ApplicantIncome

centroids = calculate_new_centroids(X, centroids)
```

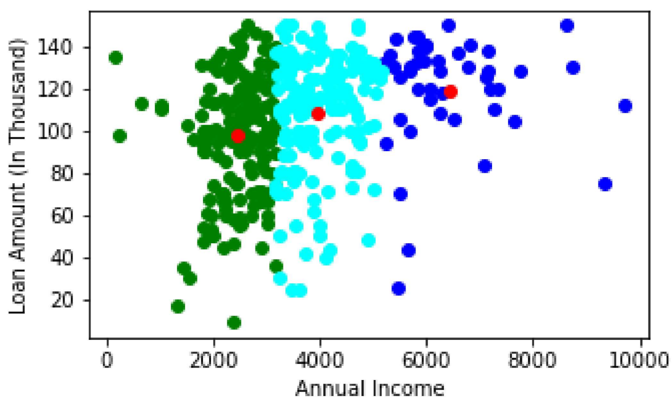
In [9]:
```
plt.figure(figsize=(5, 3))
plt.scatter(X["ApplicantIncome"], X["LoanAmount"], c='black')
plt.scatter(centroids["ApplicantIncome"], centroids["LoanAmount"], c='red')
plt.xlabel('Annual Income')
plt.ylabel('Loan Amount (In Thousand)')
plt.show()
```



Step by Step KMeans Implementation (Step 5)



In [10]:
```python
n_iter = 10
for j in range(n_iter):
    old_assignments = X["Cluster"].values
    new_assignments = assign_cluster_closest_centroid(X, centroids)
    if np.array_equal(old_assignments, new_assignments):
        break
    X["Cluster"] = new_assignments
    centroids = calculate_new_centroids(X, centroids)

plt.figure(figsize=(5, 3))
colors = ["blue", "green", "cyan"]
for k in range(K):
    data = X[X["Cluster"] == k]
    plt.scatter(data["ApplicantIncome"], data["LoanAmount"], c=colors[k])
plt.scatter(centroids["ApplicantIncome"], centroids["LoanAmount"], c='red')
plt.xlabel('Annual Income')
plt.ylabel('Loan Amount (In Thousand)')
plt.show()
```



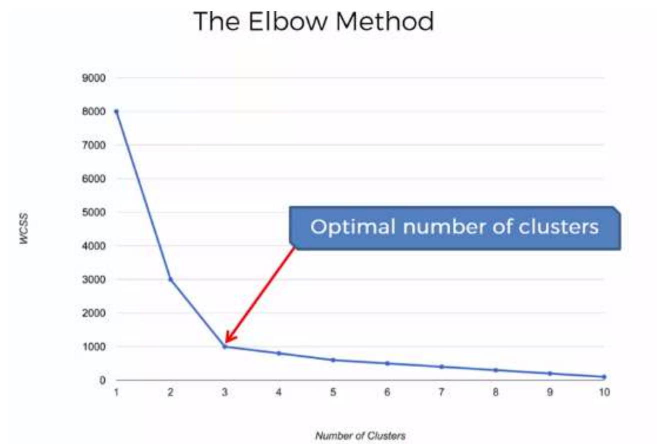
Choosing the right K

Choosing the right K

WCSS

$$\mathrm{WCSS} = \sum_{\mathrm{P_i\ in\ Cluster\ 1}} \mathrm{distance}(\mathrm{P_i}, \mathrm{C_1})^2 + \sum_{\mathrm{P_i\ in\ Cluster\ 2}} \mathrm{distance}(\mathrm{P_i}, \mathrm{C_2})^2 + \sum_{\mathrm{P_i\ in\ Cluster\ 3}} \mathrm{distance}(\mathrm{P_i}, \mathrm{C_3})^2$$

Elbow Curve



In [11]:
```
f2, ax2 = plt.subplots(1,1, figsize = (9, 6))
X, wcss = data[["LoanAmount", "ApplicantIncome"]].values, []
for i in range(1, 10):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit_predict(X)
    wcss.append(kmeans.inertia_)

ax2 = plt.plot(range(1, 10), wcss)
plt.xlabel("number of cluster (or) K")
plt.ylabel("WCSS")
plt.show()
```