K-means is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

Let's now take an example to understand how K-Means actually works:

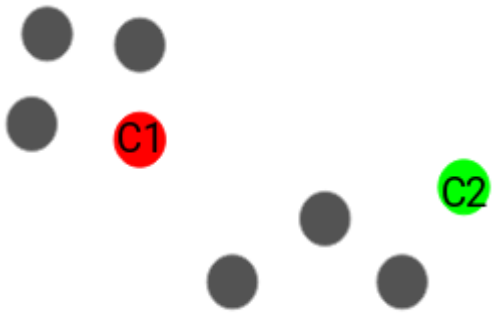
We have these 8 points and we want to apply k-means to create clusters for these points. Here's how we can do it.

Step 1: Choose the number of clusters *k*

The first step in k-means is to pick the number of clusters, k.

Step 2: Select k random points from the data as centroids

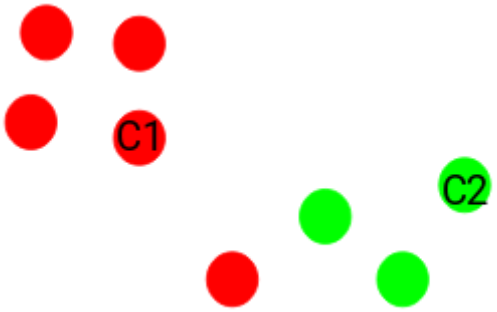
Next, we randomly select the centroid for each cluster. Let's say we want to have 2 clusters, so k is equal to 2 here. We then randomly select the centroid:



Here, the red and green circles represent the centroid for these clusters.

Step 3: Assign all the points to the closest cluster centroid

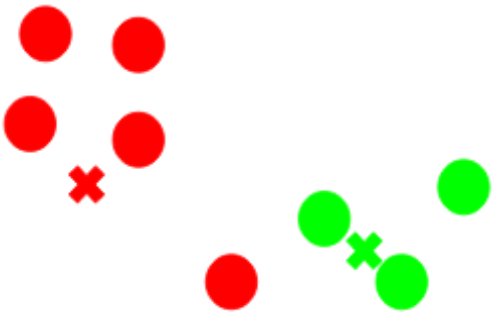
Once we have initialized the centroids, we assign each point to the closest cluster centroid:



Here you can see that the points which are closer to the red point are assigned to the red cluster whereas the points which are closer to the green point are assigned to the green cluster.

Step 4: Recompute the centroids of newly formed clusters

Now, once we have assigned all of the points to either cluster, the next step is to compute the centroids of newly formed clusters:



Here, the red and green crosses are the new centroids.

Step 5: Repeat steps 3 and 4

We then repeat steps 3 and 4:

The step of computing the centroid and assigning all the points to the cluster based on their distance from the centroid is a single iteration. But wait – when should we stop this process? It can't run till eternity, right?

Stopping Criteria for K-Means Clustering

There are essentially three stopping criteria that can be adopted to stop the K-means algorithm:

1. Centroids of newly formed clusters do not change
2. Points remain in the same cluster
3. Maximum number of iterations are reached

We can stop the algorithm if the centroids of newly formed clusters are not changing. Even after multiple iterations, if we are getting the same centroids for all the clusters, we

can say that the algorithm is not learning any new pattern and it is a sign to stop the training.

Another clear sign that we should stop the training process if the points remain in the same cluster even after training the algorithm for multiple iterations.

Finally, we can stop the training if the maximum number of iterations is reached. Suppose if we have set the number of iterations as 100. The process will repeat for 100 iterations before stopping.

Implementing K-Means Clustering in Python from Scratch

Time to fire up our Jupyter notebooks (or whichever IDE you use) and get our hands dirty in Python!

We will be working on the loan prediction dataset that you can download here. I encourage you to read more about the dataset and the problem statement here. This will help you visualize what we are working on (and why we are doing this). Two pretty important questions in any data science project.

First, import all the required libraries:

```
#import libraries

import pandas as pd

import numpy as np

import random as rd

import matplotlib.pyplot as
plt
```

view rawlibrary.py hosted with ❤ by GitHub

Now, we will read the CSV file and look at the first five rows of the data:

```
data =
pd.read_csv('clustering.csv')

data.head()
```

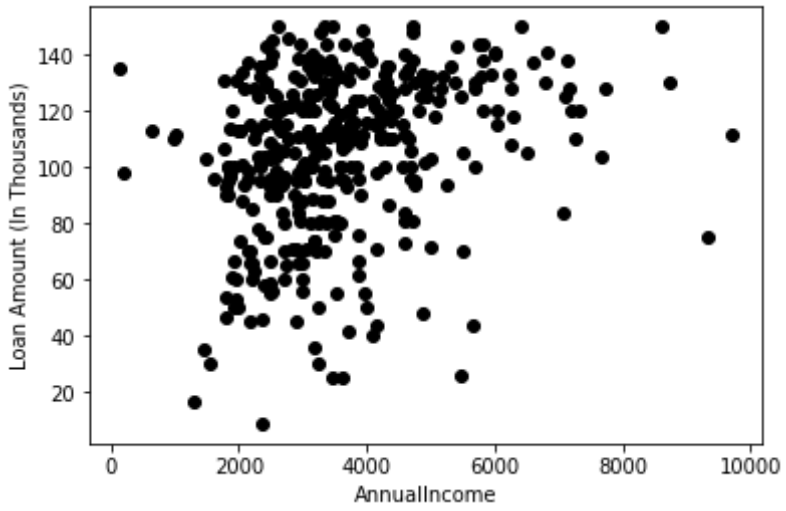
view rawdata.py hosted with ❤ by GitHub

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
1	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
2	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
3	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0
4	LP001013	Male	Yes	0	Not Graduate	No	2333	1516.0	95.0	360.0

For this article, we will be taking only two variables from the data – “LoanAmount” and “ApplicantIncome”. This will make it easy to visualize the steps as well. Let’s pick these two variables and visualize the data points:

```
X = data[["LoanAmount","ApplicantIncome"]]

#Visualise data points

plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='bla
ck')

plt.xlabel('AnnualIncome')

plt.ylabel('Loan Amount (In Thousands)')

plt.show()
```

view rawvisualization.py hosted with ❤ by GitHub

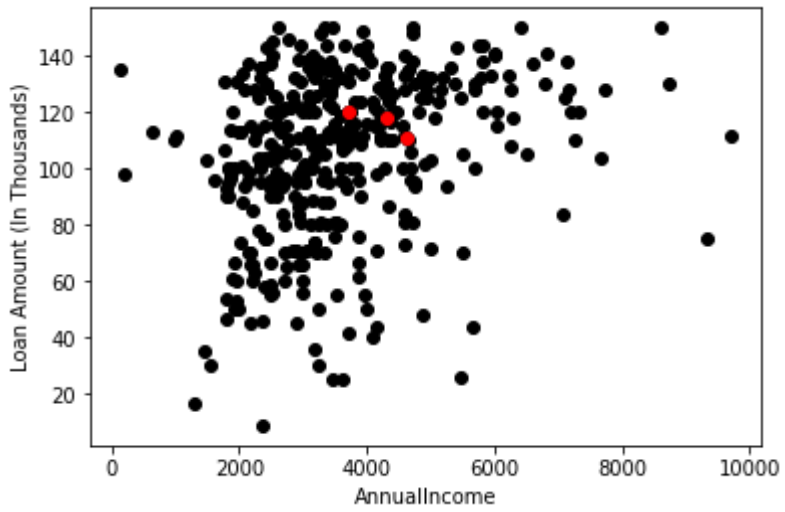


Steps 1 and 2 of K-Means were about choosing the number of clusters (k) and selecting random centroids for each cluster. We will pick 3 clusters and then select random observations from the data as the centroids:

```
# Step 1 and 2 - Choose the number of clusters (k) and select random centroid
for each cluster
```

```
#number of clusters

K=3



# Select random observation as centroids

Centroids = (X.sample(n=K))

plt.scatter(X["ApplicantIncome"],X["LoanAmount"],c='black')

plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='red')

plt.xlabel('AnnualIncome')

plt.ylabel('Loan Amount (In Thousands)')

plt.show()
```

view rawrandom_initialize.py hosted with ❤ by GitHub



Here, the red dots represent the 3 centroids for each cluster. Note that we have chosen these points randomly and hence every time you run this code, you might get different centroids.

Next, we will define some conditions to implement the K-Means Clustering algorithm. Let's first look at the code:

```
# Step 3 - Assign all the points to the closest cluster centroid

# Step 4 - Recompute centroids of newly formed clusters

# Step 5 - Repeat step 3 and 4



diff = 1

j=0



while(diff!=0):

    XD=X

    i=1

    for index1,row_c in Centroids.iterrows():

        ED=[]

        for index2,row_d in XD.iterrows():

            d1=(row_c["ApplicantIncome"]-row_d["ApplicantIncome"])**2

            d2=(row_c["LoanAmount"]-row_d["LoanAmount"])**2

            d=np.sqrt(d1+d2)

            ED.append(d)

        X[i]=ED

        i=i+1
```

```
    C=[]
    for index,row in X.iterrows():
        min_dist=row[1]
        pos=1
        for i in range(K):
            if row[i+1] < min_dist:
                min_dist = row[i+1]
                pos=i+1
        C.append(pos)
    X["Cluster"]=C
    Centroids_new =
X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
    if j == 0:
        diff=1
        j=j+1
    else:
        diff = (Centroids_new['LoanAmount'] - Centroids['LoanAmount']).sum() +
(Centroids_new['ApplicantIncome'] - Centroids['ApplicantIncome']).sum()
        print(diff.sum())
```

```
    Centroids =
X.groupby(["Cluster"]).mean()[["LoanAmount","ApplicantIncome"]]
```

view rawkmeans_scratch.py hosted with ❤ by GitHub

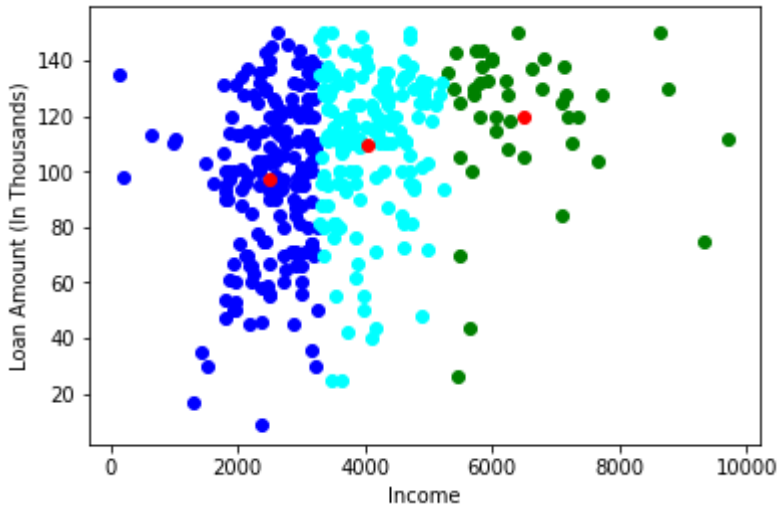
```
338.33088353093154
74.07828057315557
-47.006994092845815
-55.63613867407561
-18.485563879564225
-9.190752402517077
-9.19844100901777
-9.237706177129652
0.0
```

These values might vary every time we run this. Here, we are stopping the training when the centroids are not changing after two iterations. We have initially defined the *diff* as 1 and inside the while loop, we are calculating this *diff* as the difference between the centroids in the previous iteration and the current iteration.

When this difference is 0, we are stopping the training. Let's now visualize the clusters we have got:

```
color=['blue','green','cyan']

for k in range(K):

   data=X[X["Cluster"]==k+1]


plt.scatter(data["ApplicantIncome"],data["LoanAmount"],c=color[k])

plt.scatter(Centroids["ApplicantIncome"],Centroids["LoanAmount"],c='
red')

plt.xlabel('Income')

plt.ylabel('Loan Amount (In Thousands)')

plt.show()
```

view rawcluster_visualization.py hosted with ❤ by GitHub



Awesome! Here, we can clearly visualize three clusters. The red dots represent the centroid of each cluster. I hope you now have a clear understanding of how K-Means work.