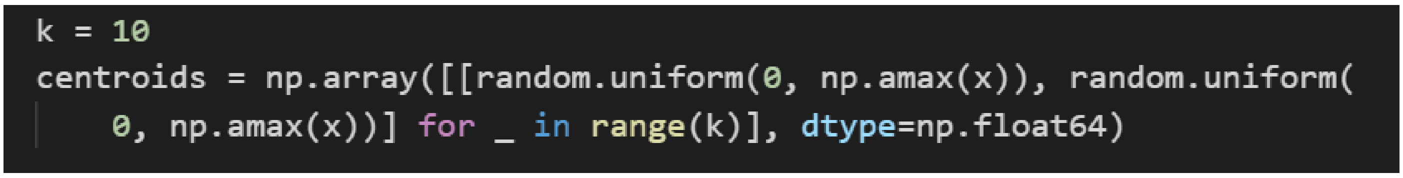
Machine Learning - Task 1 - Moshe Binieli

The method that creates random centroids is naive, we take the values from a uniform distribution in range on 0 to the maximum value of the dataset.

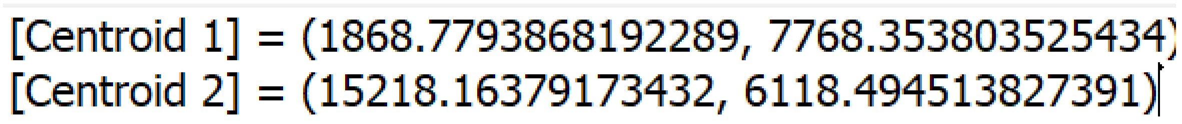


```
k = 10
centroids = np.array([[random.uniform(0, np.amax(x)), random.uniform(
    0, np.amax(x))] for _ in range(k)], dtype=np.float64)
```

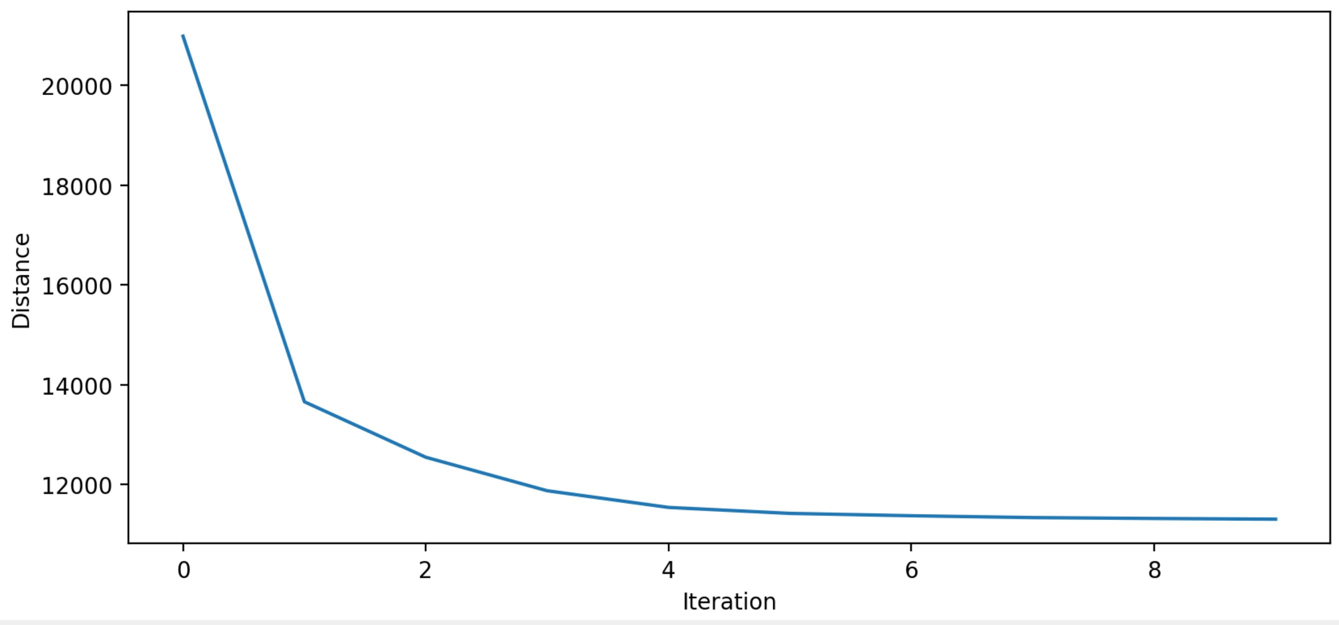
Let's run K-Means on difference K numbers for 10 iterations every time.

K = 2

After running for K=2 a couple of times we see the same behavior.
The random centroids that have been chosen are:



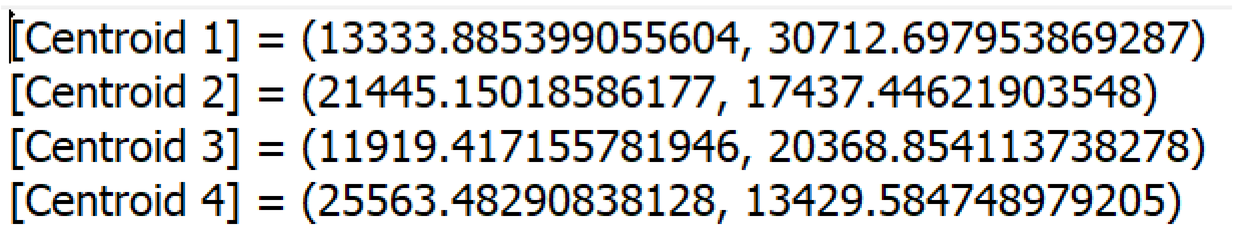
```
[Centroid 1] = (1868.7793868192289, 7768.353803525434)
[Centroid 2] = (15218.16379173432, 6118.494513827391)
```



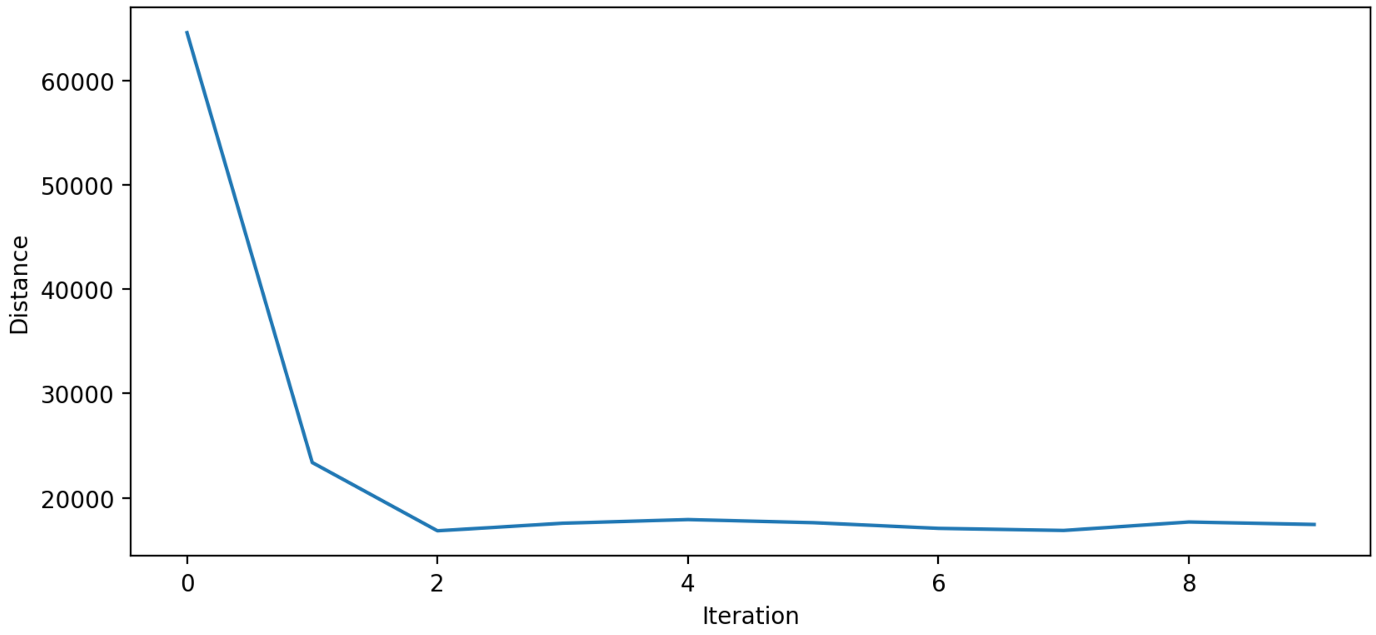
K = 4

I run K=4 couple of times, depending on centroids initialization we see different behavior, but generally it acts like the plot I show here.

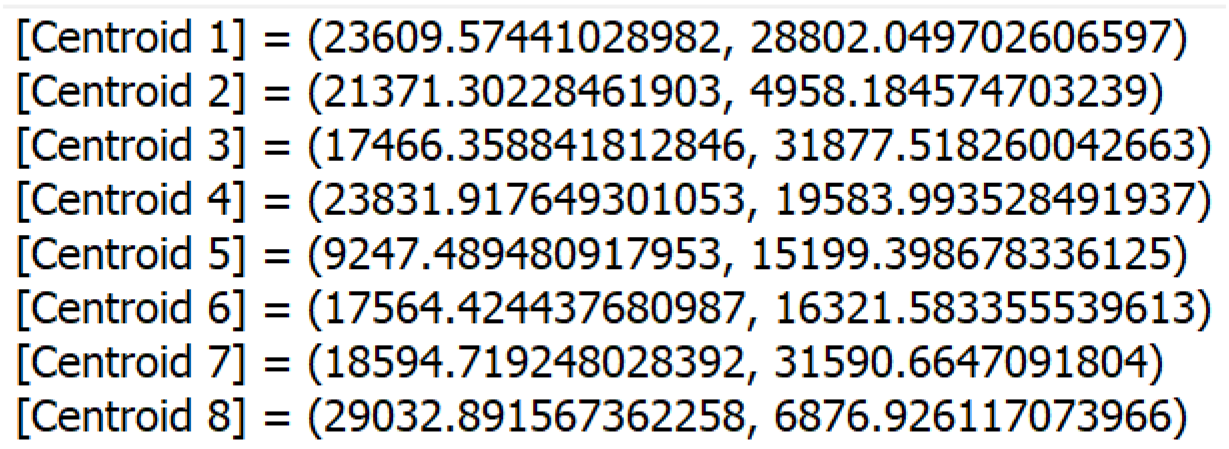
Centroids initialization:



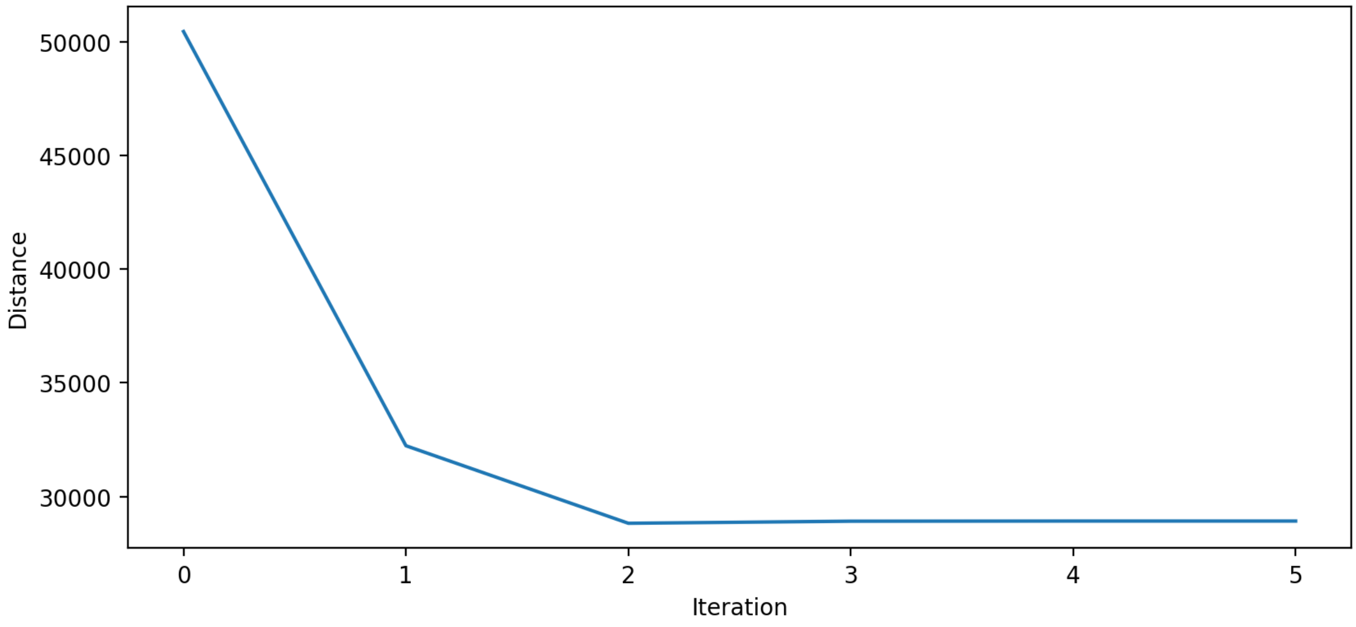
```
[Centroid 1] = (13333.885399055604, 30712.697953869287)
[Centroid 2] = (21445.15018586177, 17437.44621903548)
[Centroid 3] = (11919.417155781946, 20368.854113738278)
[Centroid 4] = (25563.48290838128, 13429.584748979205)
```



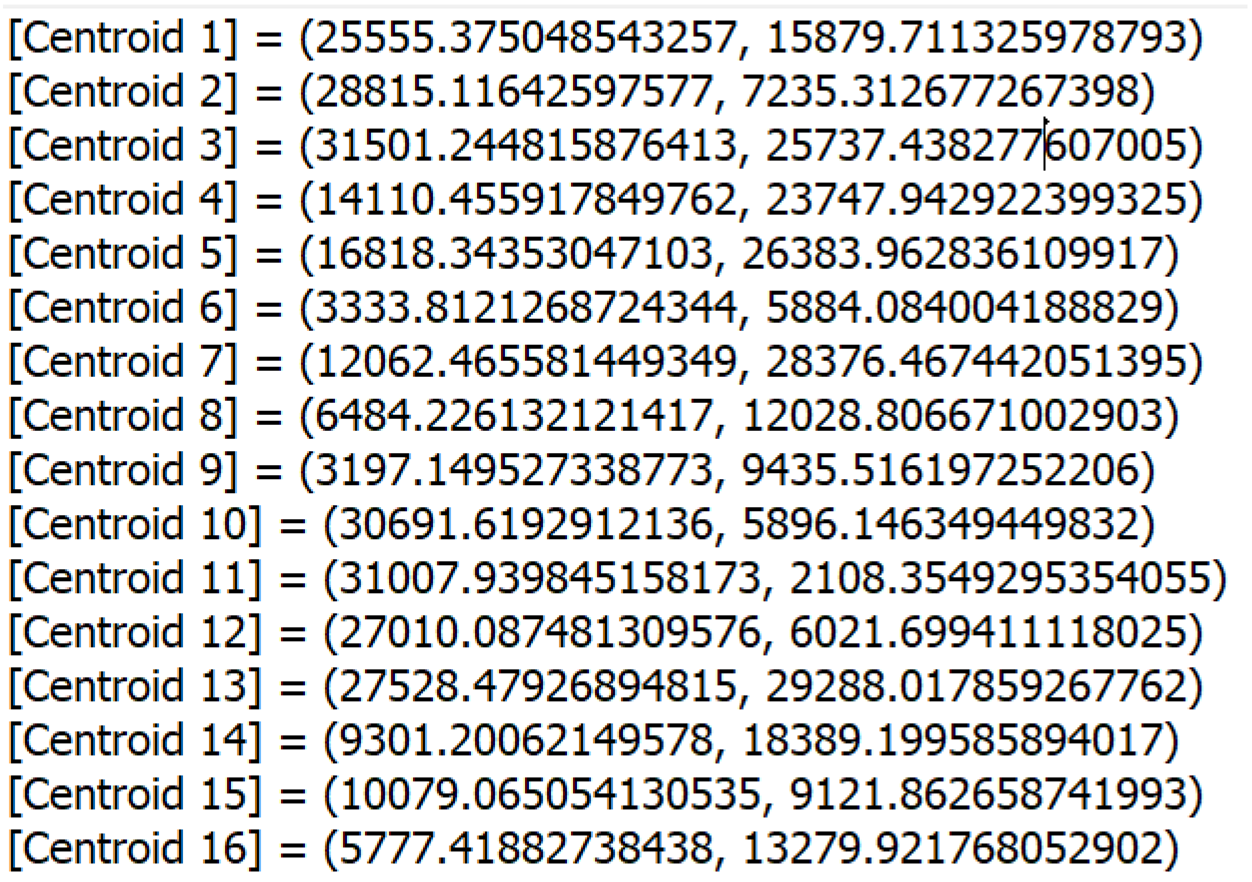
K=8



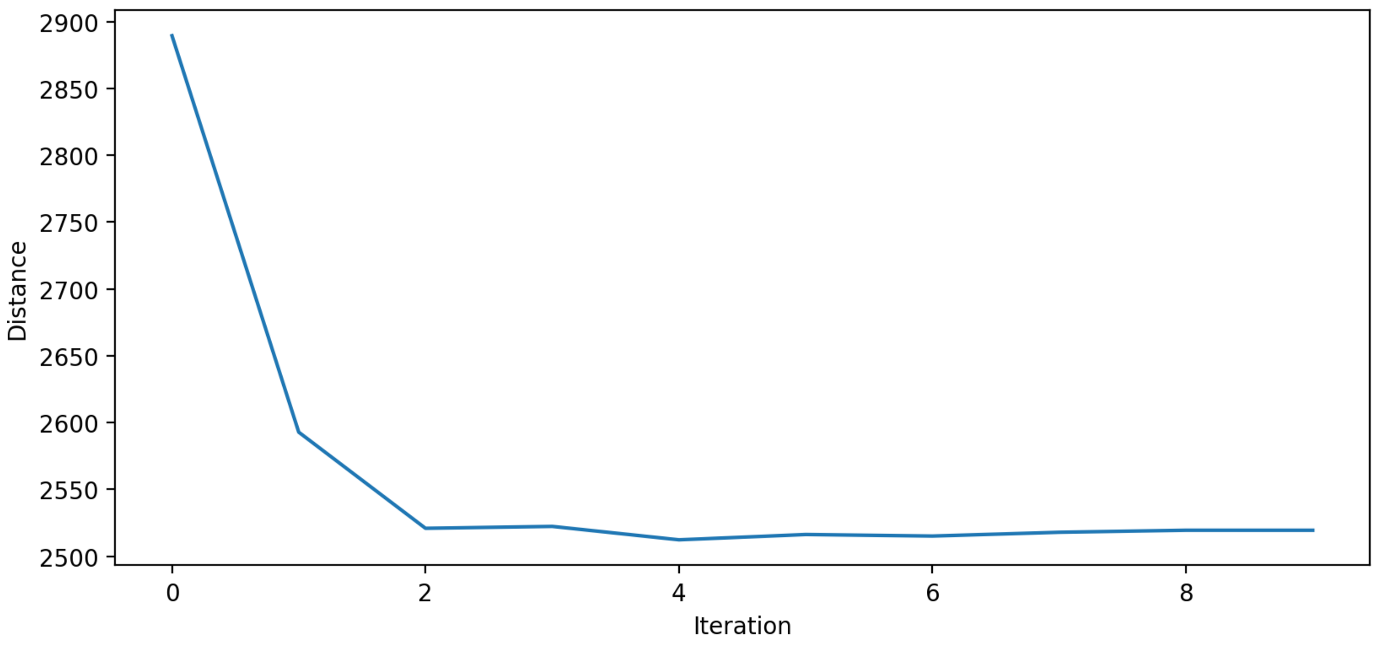
```
[Centroid 1] = (23609.57441028982, 28802.049702606597)
[Centroid 2] = (21371.30228461903, 4958.184574703239)
[Centroid 3] = (17466.358841812846, 31877.518260042663)
[Centroid 4] = (23831.917649301053, 19583.993528491937)
[Centroid 5] = (9247.489480917953, 15199.398678336125)
[Centroid 6] = (17564.424437680987, 16321.583355539613)
[Centroid 7] = (18594.719248028392, 31590.6647091804)
[Centroid 8] = (29032.891567362258, 6876.926117073966)
```



K=16



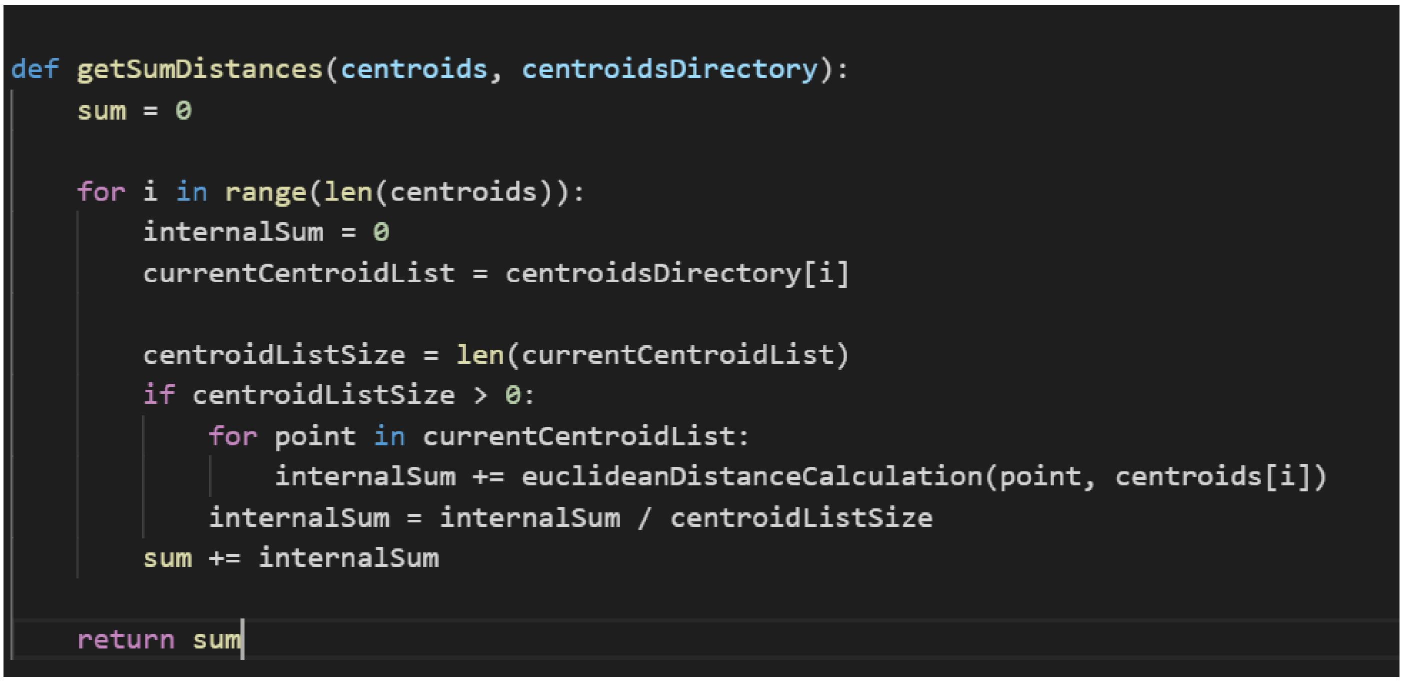
```
[Centroid 1] = (25555.375048543257, 15879.711325978793)
[Centroid 2] = (28815.11642597577, 7235.312677267398)
[Centroid 3] = (31501.244815876413, 25737.438277607005)
[Centroid 4] = (14110.455917849762, 23747.942922399325)
[Centroid 5] = (16818.34353047103, 26383.962836109917)
[Centroid 6] = (3333.8121268724344, 5884.084004188829)
[Centroid 7] = (12062.465581449349, 28376.467442051395)
[Centroid 8] = (6484.226132121417, 12028.806671002903)
[Centroid 9] = (3197.149527338773, 9435.516197252206)
[Centroid 10] = (30691.6192912136, 5896.146349449832)
[Centroid 11] = (31007.939845158173, 2108.3549295354055)
[Centroid 12] = (27010.087481309576, 6021.699411118025)
[Centroid 13] = (27528.47926894815, 29288.017859267762)
[Centroid 14] = (9301.20062149578, 18389.199585894017)
[Centroid 15] = (10079.065054130535, 9121.862658741993)
[Centroid 16] = (5777.41882738438, 13279.921768052902)
```



How do we calculate the average loss for each iteration?

Each iteration we calculate the Euclidean Distance from point to its associated centroid, then we sum up all the values we got and we divide by the size of the array.

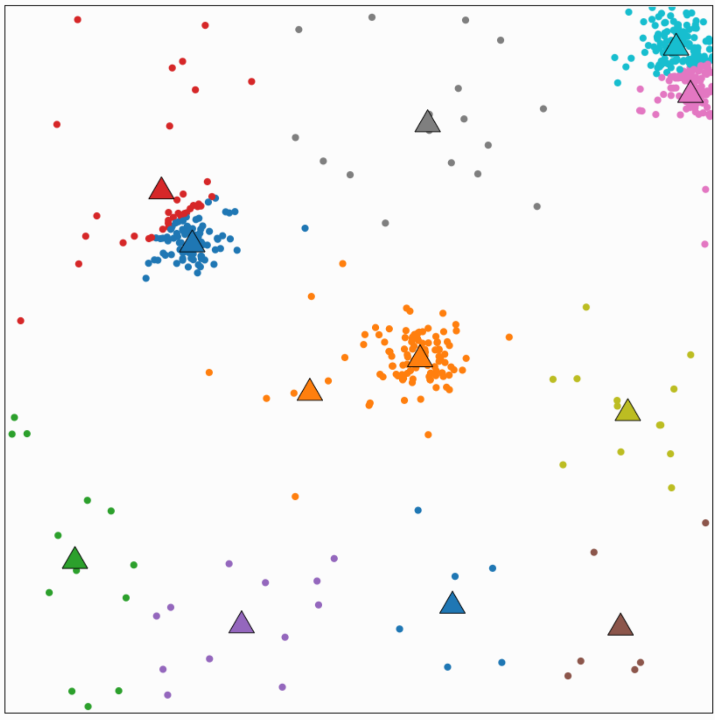
And we do it for each centroid.



```python
def getSumDistances(centroids, centroidsDirectory):
    sum = 0

    for i in range(len(centroids)):
        internalSum = 0
        currentCentroidList = centroidsDirectory[i]

        centroidListSize = len(currentCentroidList)
        if centroidListSize > 0:
            for point in currentCentroidList:
                internalSum += euclideanDistanceCalculation(point, centroids[i])
            internalSum = internalSum / centroidListSize
        sum += internalSum

    return sum
```

Conclusions

* Choosing different centroid initialization and different amount of centroids impacts on the result of the algorithm, therefore I have a lot of ways to initialize the centroids and I will get different outputs, something important to mention, NOT every centroid will get his points, because how the algorithm works, some centroids can find their selves stuck in the "space" without points close to them, for example:



Mean square point-centroid distance: 1593.82

we clearly see in the image that some centroids have low average loss value while others have high values, so there are different algorithms such as K-Means++ that solves this issue, we ain't do it here so we can get different values each time.

* According to the last section, theoretically K means should give lower average loss value each time, but as I show in the last section it will not be the case every time, so **I did run the algorithm couple of times on each K**, and the behavior was different every time, sometimes the value of K=16 was really bad compared to K=8, same thing happened with K=4 and K=2 for example.