Math 157: Intro to Mathematical Software

UC San Diego, Winter 2021

K-means Clustering (Machine Learning)

By Odemuno

Intro to Classification and Clustering in Machine Learning

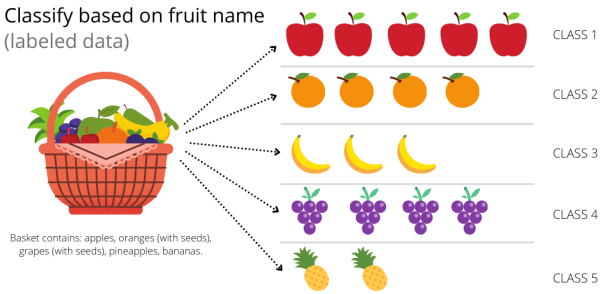
What is Classification?

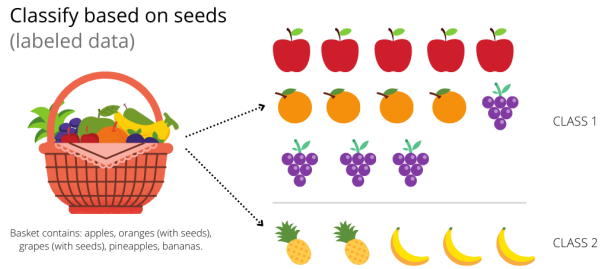
Imagine you are given a basket of 5 different types of fruits (apples, oranges (with seeds), grapes (with seeds), pineapples, bananas) and you are asked to classify them. You may be wondering how to classify them... The most obvious way is to classify based on their fruit species/ fruit name. This would give you 5 groups.

Another less obvious way to categorize them could be based on if they have seeds. This would yield you 2 groups:

- The first with apples, oranges, and grapes
- The second with the pineapples and bananas

The images below give a graphical representation of this classification.





I created the above images using Canva (graphic design software)

This is an important method for understanding datasets in machine learning. Just as a refresher, *we can define machine learning as: "...an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves."* expert.ai ML blog

What is Clustering?

In machine learning, we typically group data sets in order to understand them. The grouping of labeled datasets is called classification. For **unlabeled** data sets, this is what we call **clustering**.

> Clustering is the task of gathering samples into groups of similar samples.

These features of similarities are referred to as “clusters.” We run computer algorithms to help us decide how to classify elements in the data based on properties within the data.

Simple Steps to Clustering

Developers at Google have outlined a short, yet effective Clustering Workflow on how to cluster data generally. As with some Machine Learning problems, this workflow makes two main assumptions: the data points may be unstructured and the data points have similarities.

1. **Prepare your data**: normalize, scale and transform your feature data
2. **Create a similarity metric**: know how similar pairs of data points are
3. **Run clustering algorithm**: use the similarity metric to cluster data
4. **Interpret results and adjust your clustering**: verify the output and improve the result by iterating

Applications of Clustering

With the massive amount of unstructured data we accumulate everyday with technology, clustering is increasingly becoming a powerful computational method for many industries to learn from their data. Some of these industries include: technology, manufacturing, retail, telecommunications, healthcare, financial services, etc. These are some examples of applications in these industries:

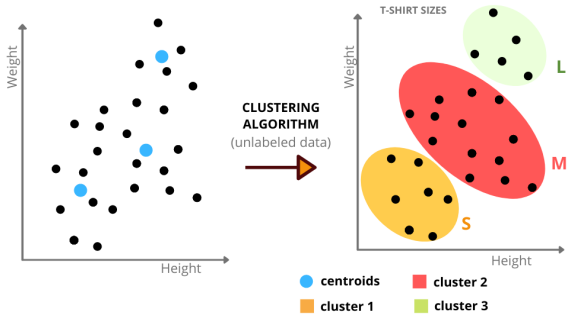
- Customer segmentation: characterizing customer groups for marketing purposes (we will explore this soon)
- Social network analysis: using text mining, network analytics, and social graphs to categorize numerous groups of platform users
- Anomaly detection: categorizing cancer tumors as benign or malignant
- Image processing: clustering for color compression in images (we will explore this soon)
- Biology: classifying different species of plants and animals
- Astronomical data analysis: categorizing stars based on their brightness
- Spam detection: using text to detect if an email is spam

Types of Clustering Algorithms

There are **4** main types of clustering algorithms:

1. **Centroid-based clustering**: This picks central clusters (known as **centroids**) to minimize the cumulative square distances from each example to its closest centroid. A classic example of this is the **K-means clustering algorithm**, where K is the number of centroids the algorithm would determine in the dataset. Clusters are represented by a central vector, which may not necessarily be a number of the data set.
    - Limitation: most K-means algorithms need the number of clusters to be specified in advance and the algorithm prefers small clusters, so you may have incorrectly cut cluster borders.

Consider this example: You work at a clothing company and a customer comes in to order a batch of approx. 30 t-shirts in Small, Medium, and Large sizes, and you are only given the weight and height distribution of the people you are producing clothes for. How do you decide how many of each size to produce? This is a perfect application of customer segmentation clustering. You can use centroid-based clustering for this to identify 3 centroids and the number of data points that fit that cluster! We will explore more cases of K-means clustering in this project!



I created the above image using Canva (graphic design software)

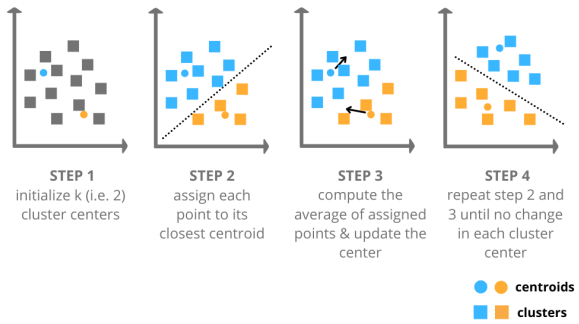
1. **Density-based clustering**: This forms arbitrary shaped-clusters for densely populated data sets. It finds high concentrations of data in a dataset. Clusters are defined as areas of higher density than the remainder of the data set.
    - Limitation: difficulty working with datasets of varying densities, and does not assign outliers to clusters
2. **Distribution-based clustering**: This consists of random distributions like the Gaussian, Normal distribution. Clusters are defined as objects belonging most likely to the same distribution.
    - Limitation: these models often suffer from overfitting. If you do not know the type of statistical distribution in your data, you should use a different algorithm
3. **Hierarchy-based clustering**: This is based on connecting objects to form clusters based on their distance. It relates objects to one another when they are close to each other, rather than far away. The two nearest clusters merge and eventually there is one cluster left. These methods produce a tree-based hierarchy of points called a dendrogram. Clusters are the maximum distance needed to connect parts of the cluster. (Note: the difference between this and K-means clustering is that K-means specifically tires to put the data into the number of clusters you specify, why hierarchical just tells you which points are most similar).

- Limitation: this cannot handle big data well as they're computationally expensive with respect to algorithm complexity

K-means Clustering Algorithm: expectation-minimization and Euclidean distance

K-means clustering is one of the simplest unsupervised machine learning algorithms that groups data based on similarities. The input of the algorithm is the features of the data set and then the model "learns" the associations between the features. This is different from other clustering algorithms because you initially specify how many clusters (K) the algorithm should identify. The "means" in K-means refers to averaging of the data to find the centroid. Here's a version of the K-means algorithm as seen in this article: K-Means Clustering in Python: A Practical Guide.

1. Specify the number K of clusters to assign and randomly initialize K centroids
2. **expectation**: Assign each point to its closest centroid
3. **maximization**: Compute the new centroid (mean) of each cluster
4. Repeat steps 2 and 3 until the centroid positions do not change



I created the above image using Canva (graphic design software)

Centroids are determined by minimizing the sum of the squares of the distance between a centroid candidate and each of its data points. These squares are what we refer to as the **Euclidean distance**.

Let n data points in a sample space be $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$. The Euclidean distance between data points x and y is

$$d_{(x,y)} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2}$$

In 2-D, we can calculate the K-means distance $dist$ between (2,2) and (5,-2) would be using the Pythagorean theorem:

$$dist = \sqrt{(2-5)^2 + (2-(-2))^2} = 5$$

In python, you can calculate the Euclidean distance between two points. Let's verify our calculations:

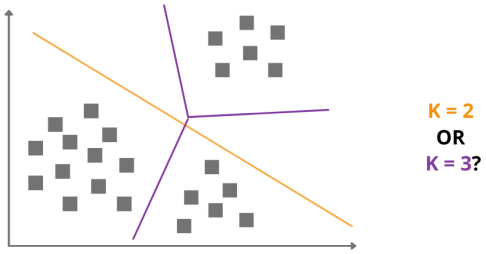
In [2]:
```
#Python code to calculate Euclidean distance using linalg.norm()
```

```
import numpy as np

# initialize data points
point1 = np.array((2,2))
point2 = np.array((5,-2))

# calculate Euclidean distance
dist = np.linalg.norm(point1 - point2)
print(dist)
```

```
5.0
```

Choosing a Value for K

Consider the image below. How would you be able to determine if 2 or 3 clusters would be best for the data points?



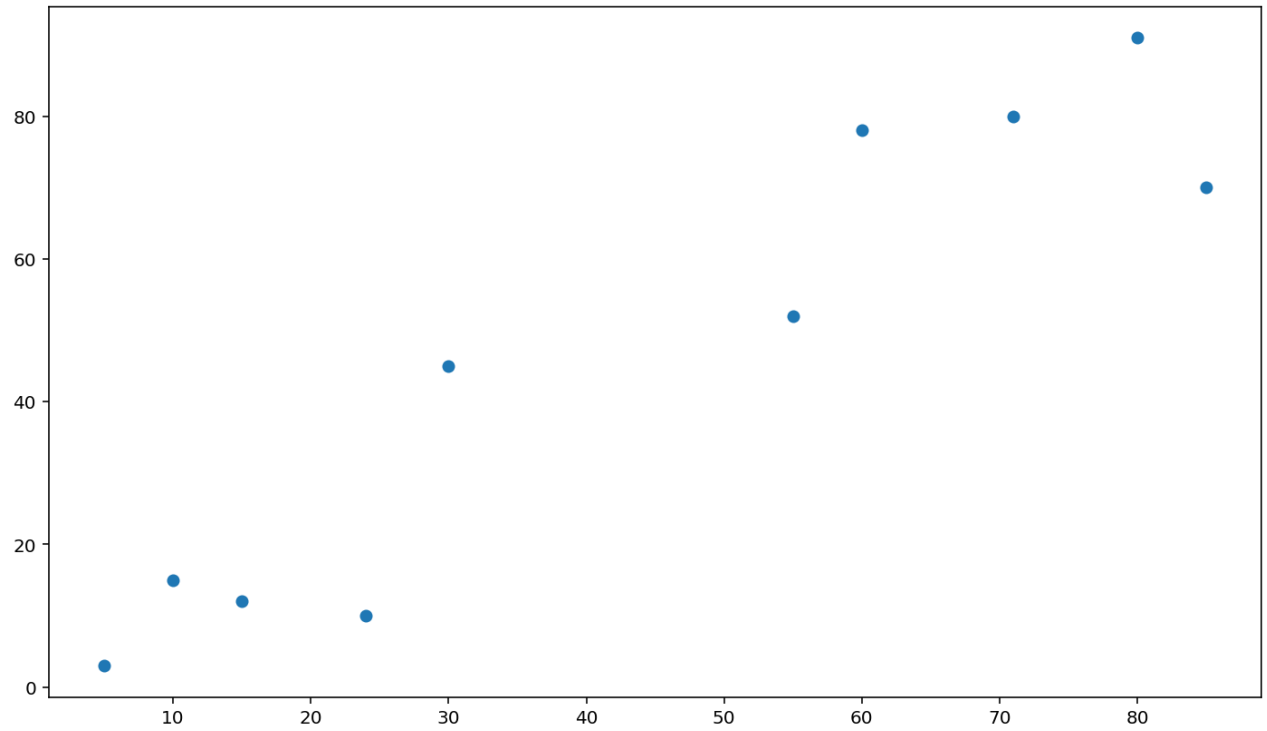
I created the above image using Canva (graphic design software)

Let's start by exploring specifying K by ourselves. As with any ML problem, it is best to start with preparing and visualizing the data.

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

# create random numpy array of 10 rows
X = np.array([[5,3],[10,15],[15,12],[24,10],[30,45],[85,70],[71,80],[60,78],[55,

# plot the data
plt.scatter(X[:,0], X[:,1], label = 'True Position')
```

Out[2]:
```
<matplotlib.collections.PathCollection at 0x7f8861d5b460>
```

Out[2]:



Now, we create clusters. If we are specifying K manually, we can choose $K = 2$ to have 2 clusters.

In [28]:
```
# compute means
kmeans = KMeans(n_clusters = 2)

# call fit to pass the data you want to cluster
kmeans.fit(X)

# show what values the final coordinates of the centroids the algorithm generate
print(kmeans.cluster_centers_)
```

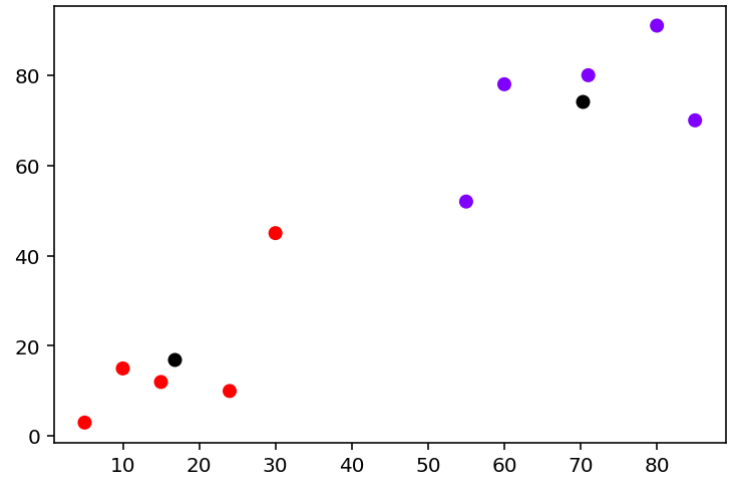
```
[[70.2 74.2]
 [16.8 17. ]]
```

Let us now visualize how the data has been clustered.

In [29]:
```
plt.scatter(X[:,0], X[:,1], c = kmeans.labels_, cmap = 'rainbow')

# include the centroids in the plot (use the color black for them)
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='b
```

Out[29]: `<matplotlib.collections.PathCollection at 0x7f44880124c0>`

Out[29]:



As we expected the left hand side forms one cluster, while the right forms another. We have now seen how to cluster random data sets into 2 clusters using Scikit-learn. How would 3 and 4 clusters look like?

Participation Check

In the two cells below, cluster the dataset X into 3 and 4 groups respectively. Then answer the following questions:

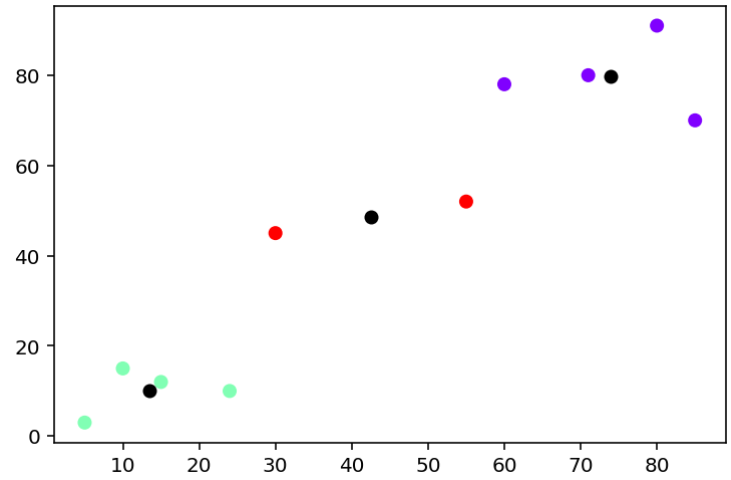
- What are the centroids for each K cluster?
- Which value of K gives the best clustering? What makes a K value "bad"?

In [17]:
```
# k = 3
kmeans3 = KMeans(n_clusters = 3)
kmeans3.fit(X)
print(kmeans3.cluster_centers_)
plt.scatter(X[:,0], X[:,1], c = kmeans3.labels_, cmap = 'rainbow')
plt.scatter(kmeans3.cluster_centers_[:,0] ,kmeans3.cluster_centers_[:,1], color=
```

```
[[74.   79.75]
 [13.5  10.  ]
 [42.5  48.5 ]]
```

Out[17]:
```
<matplotlib.collections.PathCollection at 0x7f448ecdb8b0>
```

Out[17]:

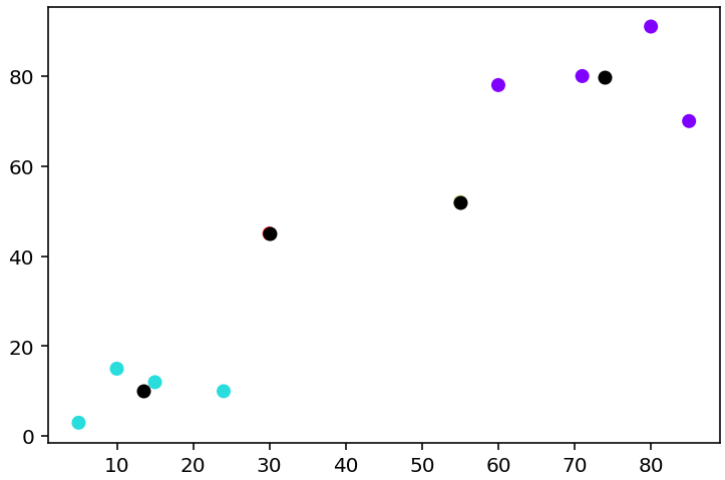


In [18]:
```
# k = 4
kmeans4 = KMeans(n_clusters = 4)
kmeans4.fit(X)
print(kmeans4.cluster_centers_)
plt.scatter(X[:,0], X[:,1], c = kmeans4.labels_, cmap = 'rainbow')
plt.scatter(kmeans4.cluster_centers_[:,0] ,kmeans4.cluster_centers_[:,1], color=
```

```
[[74.    79.75]
 [13.5   10.  ]
 [55.    52.  ]
 [30.    45.  ]]
```

Out[18]: <matplotlib.collections.PathCollection at 0x7f448ecb6970>

Out[18]:



Answers to the questions:

- the centroids are the printed arrays
- K = 2 or K = 3 gives a good clustering. I would prefer 3 because the two red dots in the middle are closest to the cluster than when K = 2. A bad cluster is when there is little association between the cluster points and the centroids (ref. Euclidean distance)

**

You probably noticed that $K = 4$ did not quite cluster the data points properly... The two middle centroids are exactly where the data points are. How can we then pick the best value for K? One of the most well-known methods for determining the optimal number of clusters is called the **Elbow Method**.

> Calculate the **Within-Cluster-Sum of Squared Errors (WSS)** for different values of K, and choose the K for which WSS becomes first starts to diminish. In the plot of **WSS-versus-K**, this is visible as an **elbow**. (source)

In simple terms, the WSS is the sum of all the square distances of the data point to its centroid. You can use the Euclidean Distance here!

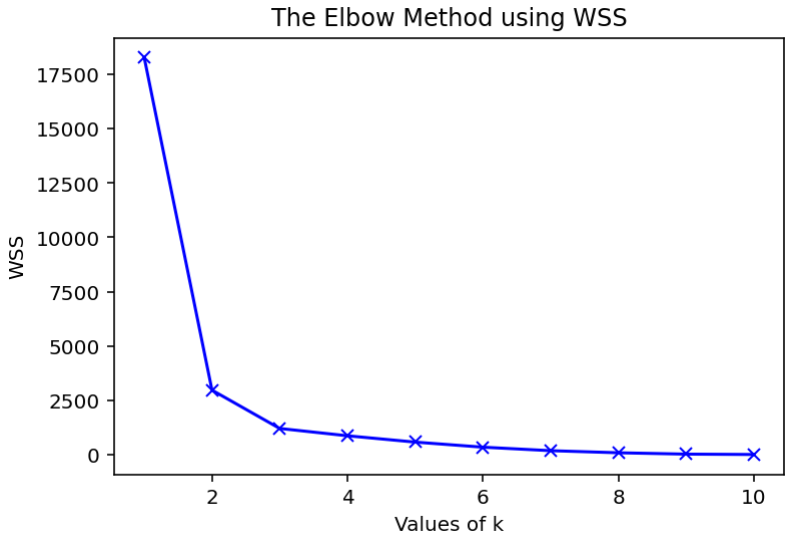
Let us implement this to find the optimal value of K for our dataset X. We can define a function WSS that returns a list of all the squared errors.

In [45]:
```
def WSS(datapoints, kmax):
    # create a list to store all the squared errors
    squaredErrors = []

    # loop through all the kvalues to calculate the WSS
    for k in range(1, kmax+1):
        kmeans = KMeans(n_clusters = k).fit(datapoints)
        centroids = kmeans.cluster_centers_
        predictClusters = kmeans.predict(datapoints)
        currentSquaredErrors = 0

        # use the Euclidean Distance to calculate the squares
        for i in range(len(datapoints)):
            currentCenter = centroids[predictClusters[i]]
            currentSquaredErrors += (datapoints[i, 0] - currentCenter[0]) ** 2 +

        squaredErrors.append(currentSquaredErrors)

    return squaredErrors
```

Using the WSS function, we can plot the Elbow Method for all values of K. For the participation check, we stopped at $K = 4$. For visualization, I will use $K = 10$.

In [52]:
```
kmax = 10
k = [i for i in range(1, kmax+1)]
WSSvalues = WSS(X, kmax)

plt.plot(k, WSSvalues, 'bx-')
plt.xlabel('Values of k')
plt.ylabel('WSS')
plt.title('The Elbow Method using WSS')
plt.show()
```

Out[52]:



The optimal value of K in this plot is typically where **the plot looks like an arm with a clear elbow**. As we can see, this is when $K = 3$ and just as expected, $K = 4$ is definitely not an ideal value because it is past this clear elbow mark.

Note that sometimes you may not have a clear elbow, so you would just have to pick K values within the range of possible options.

K-means with the famous Iris dataset

Now that we have gotten a chance to explore k-means clustering using random data points, let us use the Iris dataset on Kaggle. This is one of the best known databases for pattern recognition. The dataset has five attributes: sepal length, sepal width, petal length, petal width, and specie class (Iris Setosa, Iris Versicolour, Iris Virginica).

In [35]:
```
# import the data set and display the first 5 entries
df = pd.read_csv("Iris.csv")
df.head(5)
```

Out[35]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Select the 4 features (sepal length, sepal width, petal length, petal width) we would like to train our dataset with and implement K-means using $K = 4$ (arbitrary assignment):

In [48]:
```
# use the iloc function
x = df.iloc[:, [0,1,2,3]].values

# instantiate a KMeans class with 4 clusters
iris_kmeans = KMeans(n_clusters = 4)
y_iris_kmeans = iris_kmeans.fit_predict(x)
print(y_iris_kmeans) # the output of the clustering model
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 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 3 3 3 3 3
 3 3]
```

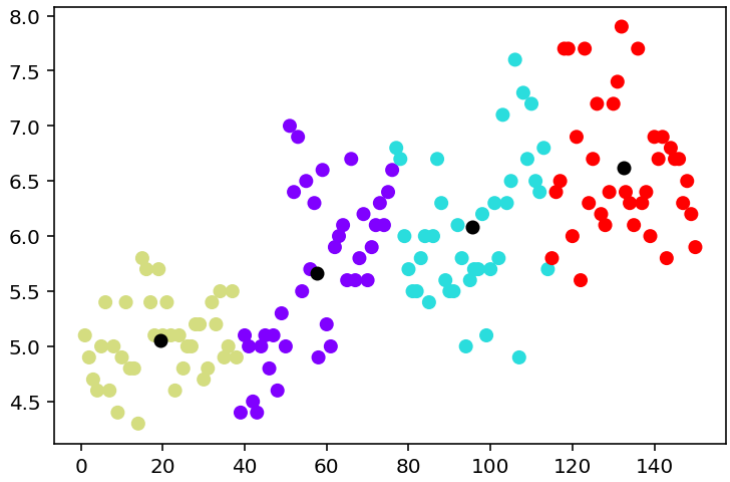
In [49]:
```
# find the 4 centroids
iris_kmeans.cluster_centers_
```

Out[49]:
```
array([[ 57.5       ,   5.66315789,   2.95      ,   3.41315789],
       [ 95.5       ,   6.07894737,   2.81315789,   4.72105263],
       [ 19.5       ,   5.05263158,   3.45263158,   1.46578947],
       [132.5       ,   6.61944444,   2.99722222,   5.52777778]])
```

In [56]:
```
# visualize these clusters and the centroids
plt.scatter(x[:,0], x[:,1], c = y_iris_kmeans, cmap = 'rainbow')
plt.scatter(iris_kmeans.cluster_centers_[:,0] ,iris_kmeans.cluster_centers_[:,1]
```

Out[56]: <matplotlib.collections.PathCollection at 0x7f44859e1820>

Out[56]:

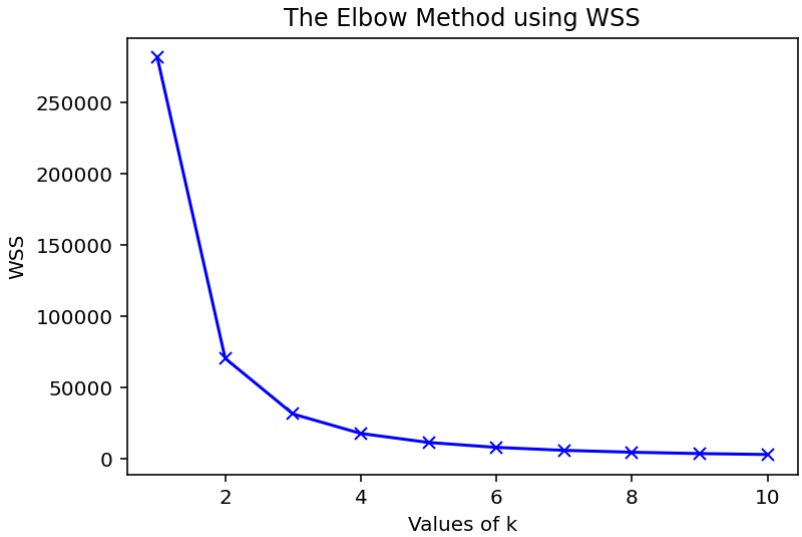


There are still some data points that are too far away from the centroids. Let us see if we can find the optimal value of K is using our WSS function defined earlier:

In [51]:

```
kmax = 10
k = [i for i in range(1,kmax+1)]
WSSvalues = WSS(x, kmax)

plt.plot(k, WSSvalues, 'bx-')
plt.xlabel('Values of k')
plt.ylabel('WSS')
plt.title('The Elbow Method using WSS')
plt.show()
```

Out[51]:



The shape of the elbow is generally at $K = 3$. This is our optimal K-value! Let's plot the clusters for these centroids.

In [53]:

```
iris_kmeans3 = KMeans(n_clusters = 3)
y_iris_kmeans3 = iris_kmeans3.fit_predict(x)
iris_kmeans3.cluster_centers_
```

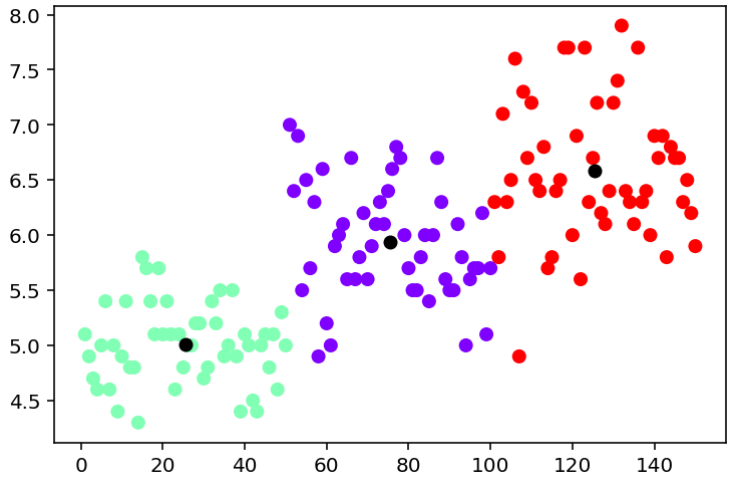
Out[53]: array([[75.5 , 5.936, 2.77 , 4.26],

```
       [ 25.5  ,   5.006,   3.418,   1.464],
       [125.5  ,   6.588,   2.974,   5.552]])
```

In [58]:
```
plt.scatter(x[:,0], x[:,1], c = y_iris_kmeans3, cmap = 'rainbow')
plt.scatter(iris_kmeans3.cluster_centers_[:,0] ,iris_kmeans3.cluster_centers_[:,
```

Out[58]:
```
<matplotlib.collections.PathCollection at 0x7f448591bf70>
```

Out[58]:



This looks so much better! $K = 3$ is indeed the optimal K value as we stated earlier on as there are 3 specie classes in the data set: Iris Setosa, Iris Versicolour, and Iris Virginica.

Conclusion

As a recap, in this presentation, we explored how we can:

- define classification and clustering
- explore the applications of clustering in machine learning
- understand the steps to clustering
- define K-means clustering
- implement the K-means algorithm to cluster data (scikit-learn)
- find the optimal K value

Here are two "mini-homework" exercises to explore K-Means further:

Exercise 1: K-means for Image Compression

Our internet is filled with millions of images and this translates into huge amounts of data being stored. One of the challenges of dealing with the large amounts of data is that it takes up a lot of space. Luckily for us, we can use image compression techniques to reduce storage space.

For example: image you have a cluster with magenta, light pink, hot pink, blush, and baby pink colors in an image. These can be grouped into one cluster called "pink." After running the image through a K-means algorithm, all the original variations of pink will be replaced with one single pink color. Thus, reducing the variants of pixels in the image.

In this section, we will be exploring how to implement K-means clustering algorithm to compress a 738x407x3 image of the UCSD Geisel Library. Our goal would be to reduce the number of colors to only 10 colors. To do this, you will first run K-Means on the colors of the pixels in the image and then you will map each pixel onto its closest centroid.

1. Load the image `geisel.png` and reshape it to be 2-Dimensional. Use `imread` from the `matplotlib.image` library.
2. Run the k-means algorithm on the data and use centroids to compress the image
3. Reshape the compressed image
4. Plot the original and compressed images
5. Answer the following questions:
    - what main differences did you notice between your original and compressed image?
    - try increasing the clusters to $K = 20$, how does that change the compressed image?

In [0]:
```
# Your code here
```

Solution

In [3]:
```
# 1. Read the image and reshape it to 2D
from matplotlib.image import imread

img = imread('geisel.jpg')
img_size = img.shape
print("Original image size:", img_size)

X = img.reshape(img_size[0] * img_size[1], img_size[2])

# 2. Run the k-means algorithm
kmeans = KMeans(n_clusters = 10)
kmeans.fit(X)

Xcomp = kmeans.cluster_centers_[kmeans.labels_]
Xcomp = np.clip(Xcomp.astype('uint8'), 0, 255) # to limit the values in an array

# 3. Reshape the compressed imag
Xcomp = Xcomp.reshape(img_size[0], img_size[1], img_size[2])
Xcomp_size = Xcomp.shape
print("Compressed image size:", Xcomp_size)

# 4. Plot the original and compressed images
fig, ax = plt.subplots(1,2, figsize = (12,8))
ax[0].imshow(img)
ax[0].set_title('Original Image')
ax[1].imshow(Xcomp)
ax[1].set_title('Compressed Image with 10 colors')
for ax in fig.axes:
    ax.axis('off')
plt.tight_layout();

# 5. Answers
# The compressed image is more pixelated. Most of the image is a variant of blue
# When I increase the cluster to 20, more colors show!
```

```
Original image size: (407, 738, 3)
Compressed image size: (407, 738, 3)
```

Out[3]:

Exercise 2: K-means for Market Customer Segmentation

Just as we had the example for finding the number of S, M and L t-shirt sizes a clothing store would need to produce for a client, we can explore other market customer segmentation problems.

In this section, we would be using a dataset pulled from the internet consisting of 30 samples and features (customer satisfaction and loyalty). The goal here is to understand all your customer groups based on their levels of satisfaction and loyalty.

1. Read the `loyalty-satisfaction.csv` file and plot the data
2. Run the K-means clustering algorithm
   - choose an arbitrary value of K
   - create a new variable `Cluster_Prediction` which would act as your "y" for your 2 features (loyalty and satisfaction)
3. Plot your results
4. Analyze your results. Is the K-value you chose optimal?
   - If it is, explain why it is.
   - If is not, find the optimal K-value using the Elbow Method.
     - Run the K-means algorithm using your new K value
     - Plot the final result and describe your findings.

In [0]:

```
# Your code here
```

Solution

In [71]:

```
# 1. Read the loyalty-satisfaction.csv file and plot the data
df = pd.read_csv('loyalty-satisfaction.csv')
df.head(5)  # view the first 5 data points
```

Out[71]:

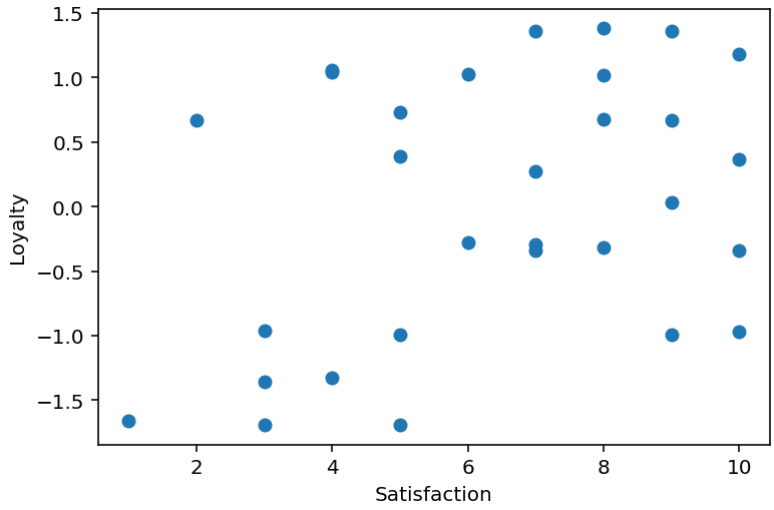
	Satisfaction	Loyalty
0	4	-1.33
1	6	-0.28
2	5	-0.99

	Satisfaction	Loyalty
3	7	-0.29
4	4	1.06

In [72]:

```
# plot data
plt.scatter(df['Satisfaction'], df['Loyalty'])
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
plt.show()
```

Out[72]:



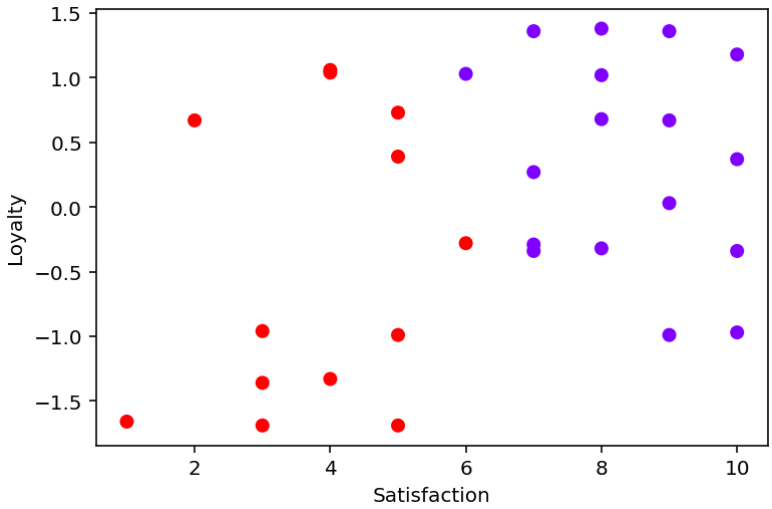
In [80]:

```
# 2. Run the k-means clustering algorithm
x = df.copy()  # copy the dataframe so we don't mess anything up
kmeans = KMeans(n_clusters = 2)  # I chose 2 arbitrarily
kmeans.fit(x)

clusters = x.copy() # copy the x dataset so that you can create a new column for
clusters['Cluster_Prediction'] = kmeans.fit_predict(x)

# 3. Plot your results
plt.scatter(clusters['Satisfaction'], clusters['Loyalty'], c = clusters['Cluster
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
```

Out[80]:

```
Text(0, 0.5, 'Loyalty')
```

Out[80]:



In [0]:
```
# 4. Analyze results
```

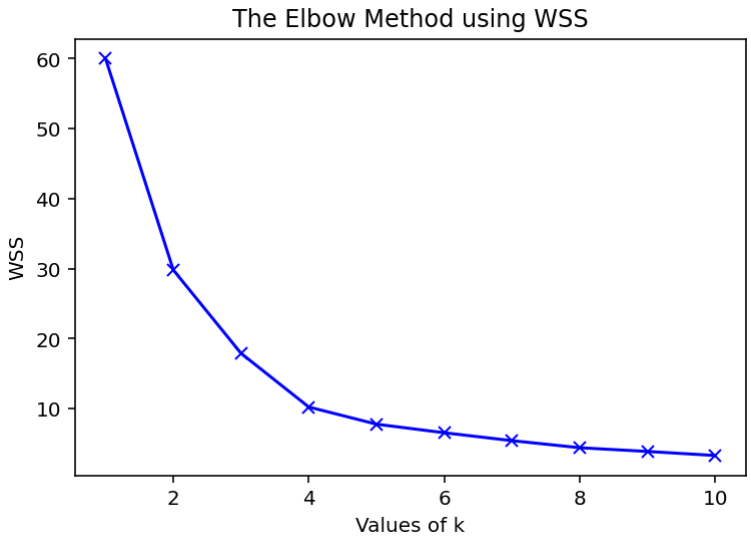
This data set favors the Satisfaction dataset. It is not categorized properly because the red cluster accounts for customers who are not really satisfied (both high and low loyalty), while the purple cluster accounts for higly satisfied customers (both high and low loyalty too). I would expect 4 main groups of customers:

- low loyalty, low satisfaction
- low loyalty, high satisfaction
- high loyalty, low satisfaction
- high loyalty, high satisfaction

Let us run the WSS function (The Elbow Method) to verify if $k = 4$ is the optimal k value in this situation.

In [83]:
```
from sklearn import preprocessing # this would help scale the data so that each

x_scaled = preprocessing.scale(x) # standardize the data

kmax = 10
k = [i for i in range(1,kmax+1)]
WSSvalues = WSS(x_scaled, kmax)

plt.plot(k, WSSvalues, 'bx-')
plt.xlabel('Values of k')
plt.ylabel('WSS')
plt.title('The Elbow Method using WSS')
plt.show()
```

Out[83]:

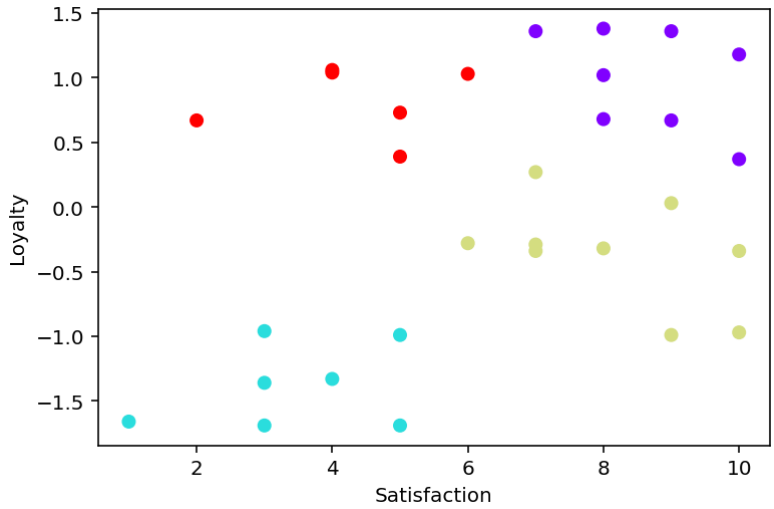


The elbow point comes at about $k = 4$ so our assumption is correct! Let us use this to find the new cluster.

In [85]:
```
kmeans = KMeans(n_clusters = 4)  # now set to 4
kmeans.fit(x)

clusters_new = x.copy() # copy the x dataset so that you can create a new column
clusters_new['Cluster_Prediction'] = kmeans.fit_predict(x_scaled)

# 3. Plot your results
plt.scatter(clusters_new['Satisfaction'], clusters_new['Loyalty'], c = clusters_
plt.xlabel('Satisfaction')
plt.ylabel('Loyalty')
```

Out[85]: `Text(0, 0.5, 'Loyalty')`

Out[85]:



- low loyalty, low satisfaction --> blue dots
- low loyalty, high satisfaction --> green dots
- high loyalty, low satisfaction --> red dots
- high loyalty, high satisfaction --> purple dots

With this, you can target the right customers for your business needs! (Credits)

Sources

- https://medium.com/code-to-express/k-means-clustering-for-beginners-using-python-from-scratch-f20e79c8ad00
- https://heartbeat.fritz.ai/k-means-clustering-using-sklearn-and-python-4a054d67b187
- https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203
- https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages
- https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1
- https://en.wikipedia.org/wiki/Cluster_analysis#Connectivity-based_clustering_(hierarchical_clustering)
- https://www.youtube.com/watch?v=4b5d3muPQmA
- https://towardsdatascience.com/k-means-clustering-with-scikit-learn-6b47a369a83c
- https://stackabuse.com/k-means-clustering-with-scikit-learn/

In [0]: