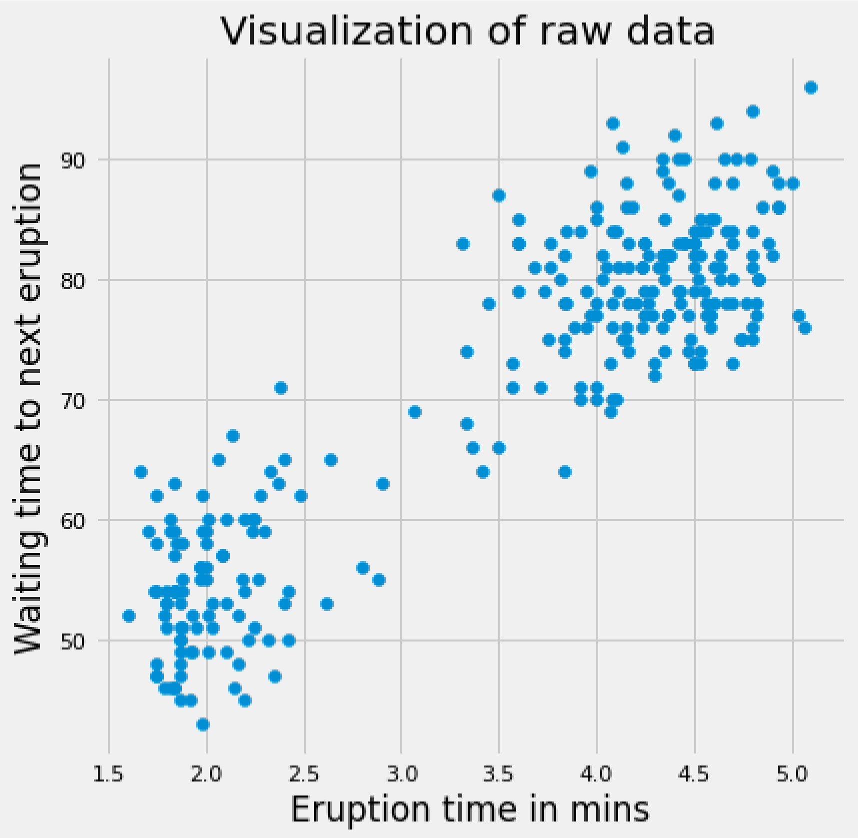
Kmeans on Geyser's Eruptions Segmentation: We'll first implement the kmeans algorithm on 2D da
272 observations and 2 features. The data covers the waiting time between eruptions and the dura
geyser in Yellowstone National Park, Wyoming, USA. We will try to find K subgroups within the data
eruptions (float): Eruption time in minutes. waiting (int): Waiting time to next eruption.

```
# Modules
import matplotlib.pyplot as plt
from matplotlib.image import imread
import pandas as pd
import seaborn as sns
from sklearn.datasets.samples_generator import (make_blobs,
                                                make_circles,
                                                make_moons)
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score

```

```
# Import the data
df = pd.read_csv('old_faithful.csv')
```

```
# Plot the data
plt.figure(figsize=(6, 6))
plt.scatter(df.iloc[:, 0], df.iloc[:, 1])
plt.xlabel('Eruption time in mins')
plt.ylabel('Waiting time to next eruption')
plt.title('Visualization of raw data');
```

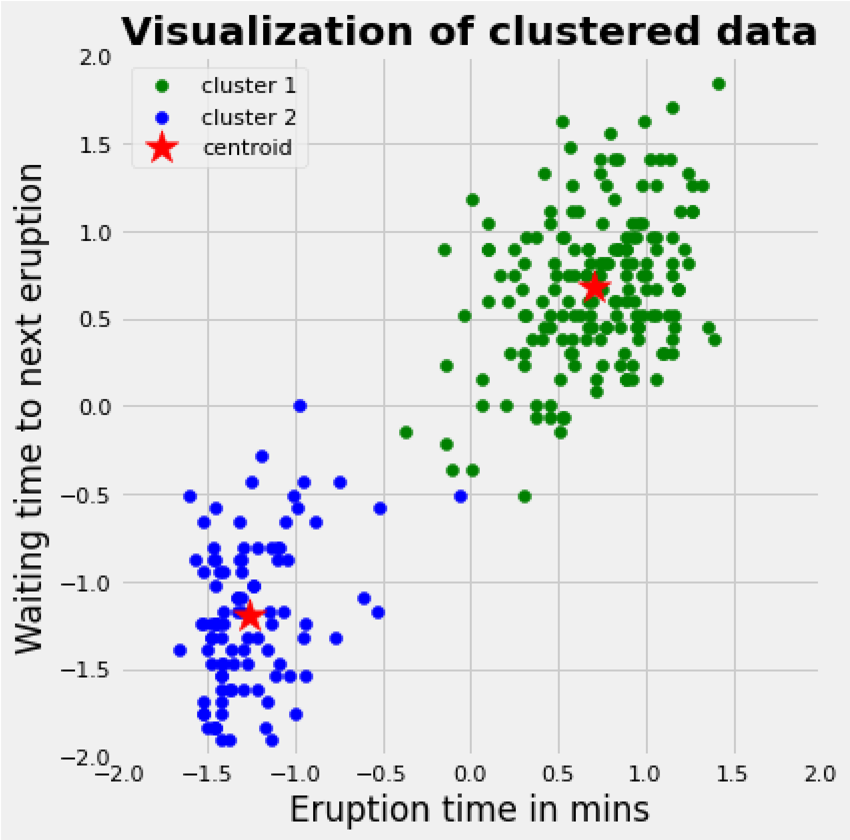


We'll use this data because it's easy to plot and visually spot the clusters since its a 2-dimension da
Let's standardize the data first and run the kmeans algorithm on the standardized data with K=2.

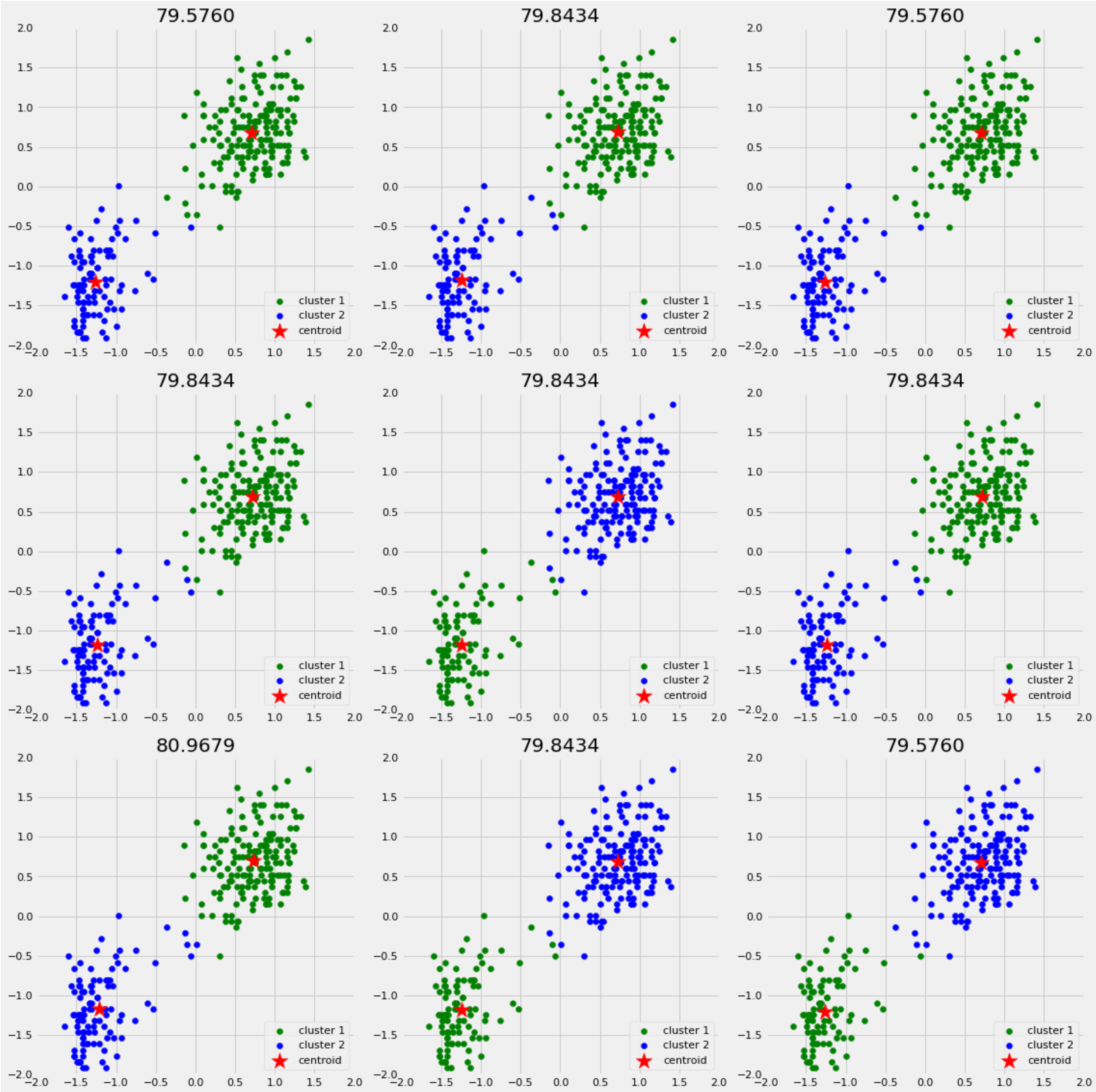
```

# Standardize the data
X_std = StandardScaler().fit_transform(df)

```

```

# Run local implementation of kmeans
km = Kmeans(n_clusters=2, max_iter=100)
km.fit(X_std)
centroids = km.centroids

```

```
# Plot the clustered data
fig, ax = plt.subplots(figsize=(6, 6))
plt.scatter(X_std[km.labels == 0, 0], X_std[km.labels == 0, 1],
            c='green', label='cluster 1')
plt.scatter(X_std[km.labels == 1, 0], X_std[km.labels == 1, 1],
            c='blue', label='cluster 2')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=300,
            c='r', label='centroid')
plt.legend()
plt.xlim([-2, 2])
plt.ylim([-2, 2])
plt.xlabel('Eruption time in mins')
plt.ylabel('Waiting time to next eruption')
plt.title('Visualization of clustered data', fontweight='bold')
ax.set_aspect('equal');
```




```

n_iter = 9
fig, ax = plt.subplots(3, 3, figsize=(16, 16))
ax = np.ravel(ax)
centers = []
for i in range(n_iter):
    # Run local implementation of kmeans
    km = Kmeans(n_clusters=2,
                max_iter=3,
                random_state=np.random.randint(0, 1000, size=1))
    km.fit(X_std)
    centroids = km.centroids
    centers.append(centroids)
    ax[i].scatter(X_std[km.labels == 0, 0], X_std[km.labels == 0, 1],
                  c='green', label='cluster 1')
    ax[i].scatter(X_std[km.labels == 1, 0], X_std[km.labels == 1, 1],
                  c='blue', label='cluster 2')
    ax[i].scatter(centroids[:, 0], centroids[:, 1],
                  c='r', marker='*', s=300, label='centroid')
    ax[i].set_xlim([-2, 2])
    ax[i].set_ylim([-2, 2])
    ax[i].legend(loc='lower right')
    ax[i].set_title(f'{km.error:.4f}')
    ax[i].set_aspect('equal')
plt.tight_layout();

```


79.5760
cluster 1
cluster 2
centroid
79.8434
cluster 1
cluster 2
centroid
79.5760
cluster 1
cluster 2
centroid
79.8434
cluster 1
cluster 2
centroid
79.8434
cluster 1
cluster 2
centroid
79.8434
cluster 1
cluster 2
centroid
80.9679
cluster 1
cluster 2
centroid
79.8434
cluster 1
cluster 2
centroid
79.5760
cluster 1
cluster 2
centroid

As the graph above shows that we only ended up with two different ways of clusterings based on c
with the lowest sum of squared distance.

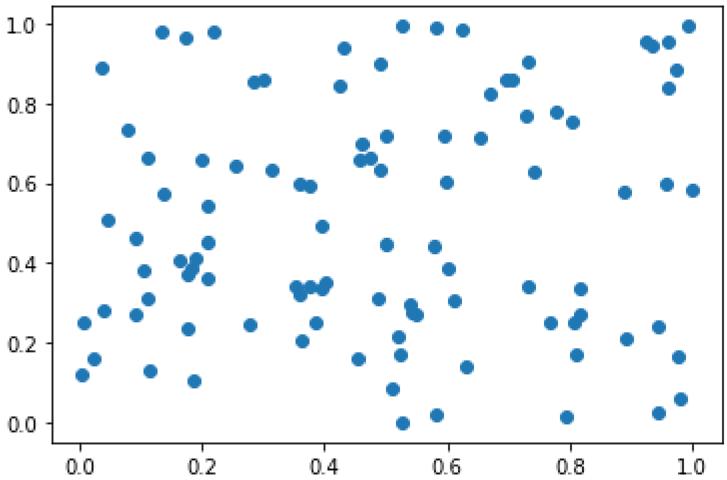
Analysis

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from sklearn.cluster import KMeans
import time
```

```
np.random.seed(100)
X = np.random.rand(100, 2)
X_test = np.random.rand(10, 2)
```

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

```
<matplotlib.collections.PathCollection at 0x7f5fc9e81898>
```



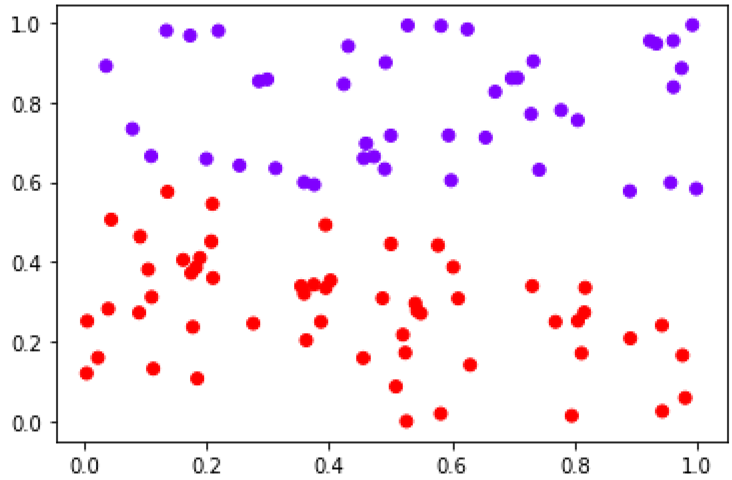
Number of Cluster = 2

```
start_time = time.time()
kmeans = KMeans(n_clusters=2, random_state=3425)
kmeans.fit(X)
end_time = time.time()
print("Time Taken : "+str(end_time-start_time))
```

```
Time Taken : 0.030069828033447266
```

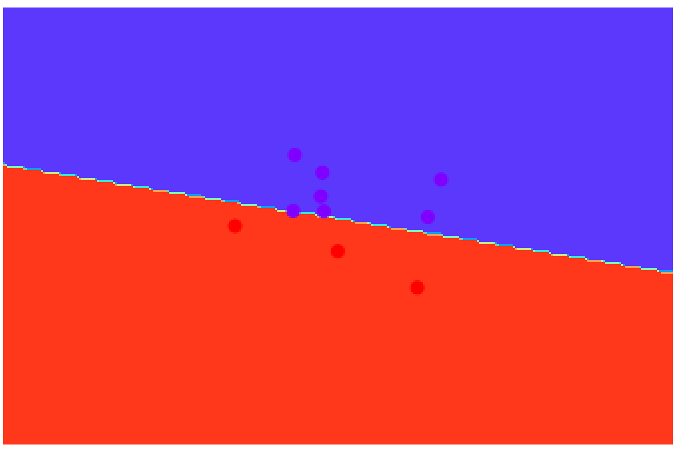
```
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

<matplotlib.collections.PathCollection at 0x7f5fc6ec0ef0>




```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# here "model" is your model's prediction (classification) function
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap='rainbow')
plt.axis('off')

# Plot also the training points
plt.scatter(X_test[:,0],X_test[:,1], c=kmeans.predict(X_test), cmap='rainbow')
```

<matplotlib.collections.PathCollection at 0x7f5fc6eacf60>



```
centroid = kmeans.predict(X_test)
```

```
sse = 0
for i in range(len(X_test)):
  temp = np.linalg.norm(X_test[i]-kmeans.cluster_centers_[centroid[i]])
  temp = temp*temp
  sse += temp
print(sse)
```

```
1.0645816624472537
```

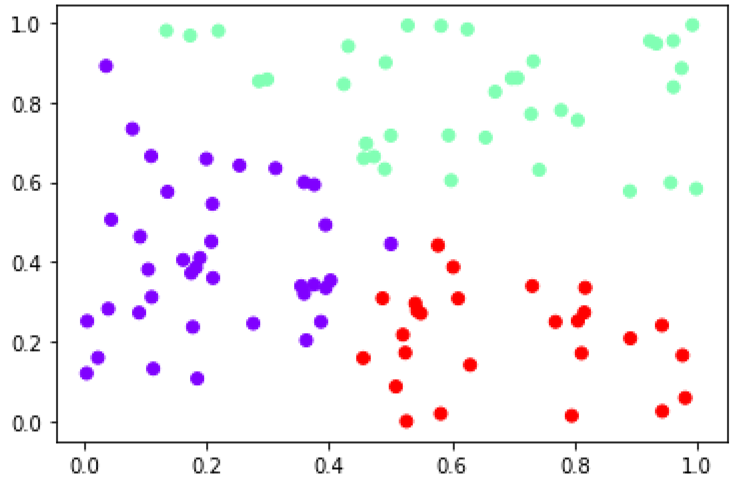
Similarly Doing for n_clusters=3

```
start_time = time.time()
kmeans = KMeans(n_clusters=3, random_state=3425)
kmeans.fit(X)
end_time = time.time()
print("Time Taken : "+str(end_time-start_time))
```

```
Time Taken : 0.033385276794433594
```

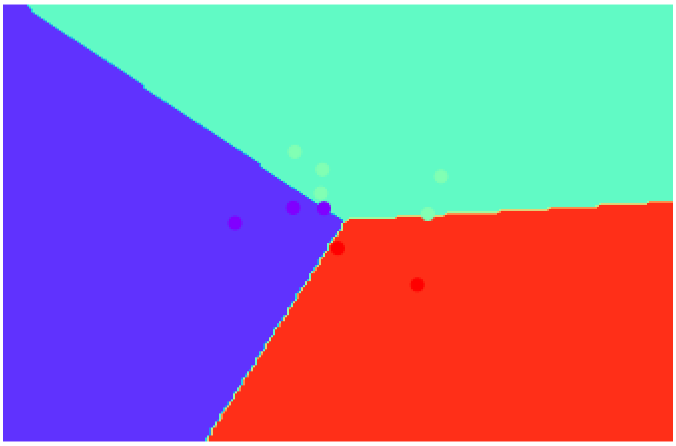
```
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f5fc6e0ef60>
```




```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# here "model" is your model's prediction (classification) function
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap='rainbow')
plt.axis('off')

# Plot also the training points
plt.scatter(X_test[:,0],X_test[:,1], c=kmeans.predict(X_test), cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f5fc6df7b38>
```



```
centroid = kmeans.predict(X_test)
sse = 0
for i in range(len(X_test)):
  temp = np.linalg.norm(X_test[i]-kmeans.cluster_centers_[centroid[i]])
  temp = temp*temp
  sse += temp
print(sse)
```

```
0.7406067549399893
```

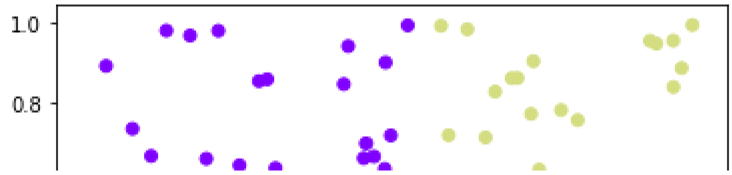
Similarly Doing for n_clusters=4

```
start_time = time.time()
kmeans = KMeans(n_clusters=4, random_state=3425)
kmeans.fit(X)
end_time = time.time()
print("Time Taken : "+str(end_time-start_time))
```

```
Time Taken : 0.03960251808166504
```

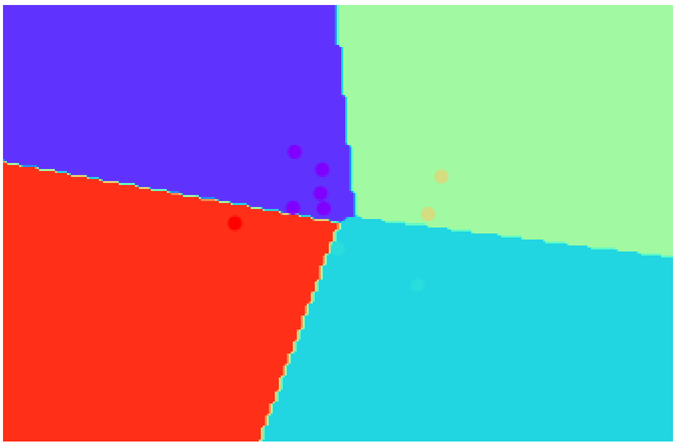
```
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f5fc6d5be48>
```




```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# here "model" is your model's prediction (classification) function
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap='rainbow')
plt.axis('off')

# Plot also the training points
plt.scatter(X_test[:,0],X_test[:,1], c=kmeans.predict(X_test), cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f5fc6cbf9e8>
```



```
centroid = kmeans.predict(X_test)
sse = 0
for i in range(len(X_test)):
  temp = np.linalg.norm(X_test[i]-kmeans.cluster_centers_[centroid[i]])
  temp = temp*temp
  sse += temp
print(sse)
```

```
0.39793603053158416
```

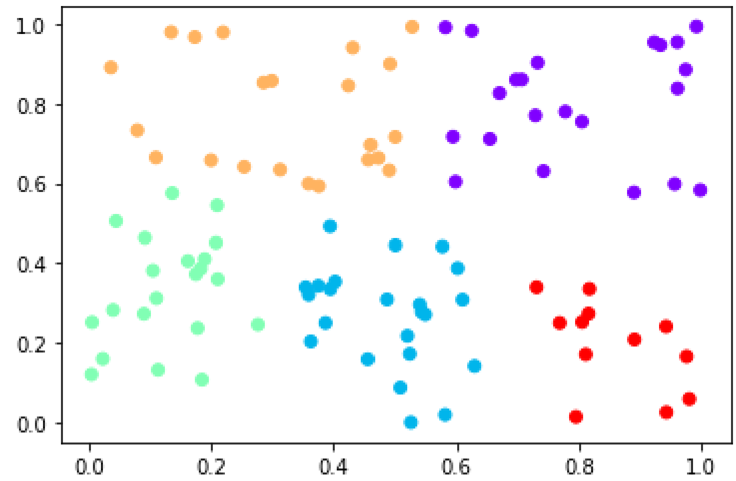
Similarly Doing for n_clusters=5

```
start_time = time.time()
kmeans = KMeans(n_clusters=5, random_state=3425)
kmeans.fit(X)
end_time = time.time()
print("Time Taken : "+str(end_time-start_time))
```

```
Time Taken : 0.04885458946228027
```

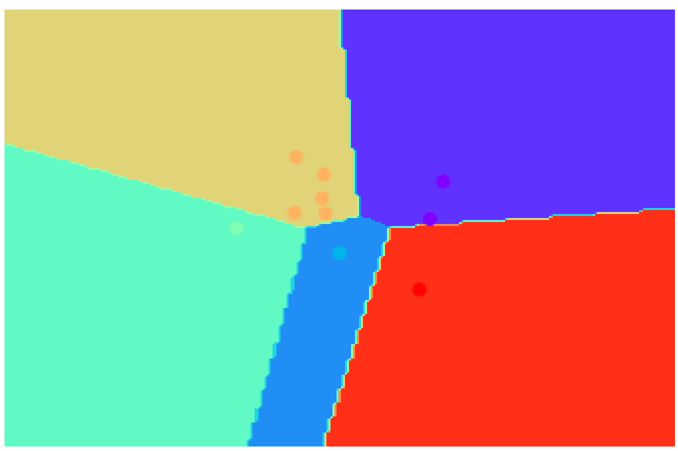
```python
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f5fc6bb7d30>
```




```python
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# here "model" is your model's prediction (classification) function
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap='rainbow')
plt.axis('off')

# Plot also the training points
plt.scatter(X_test[:,0],X_test[:,1], c=kmeans.predict(X_test), cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7f5fc6af8b00>
```



```python
centroid = kmeans.predict(X_test)
```

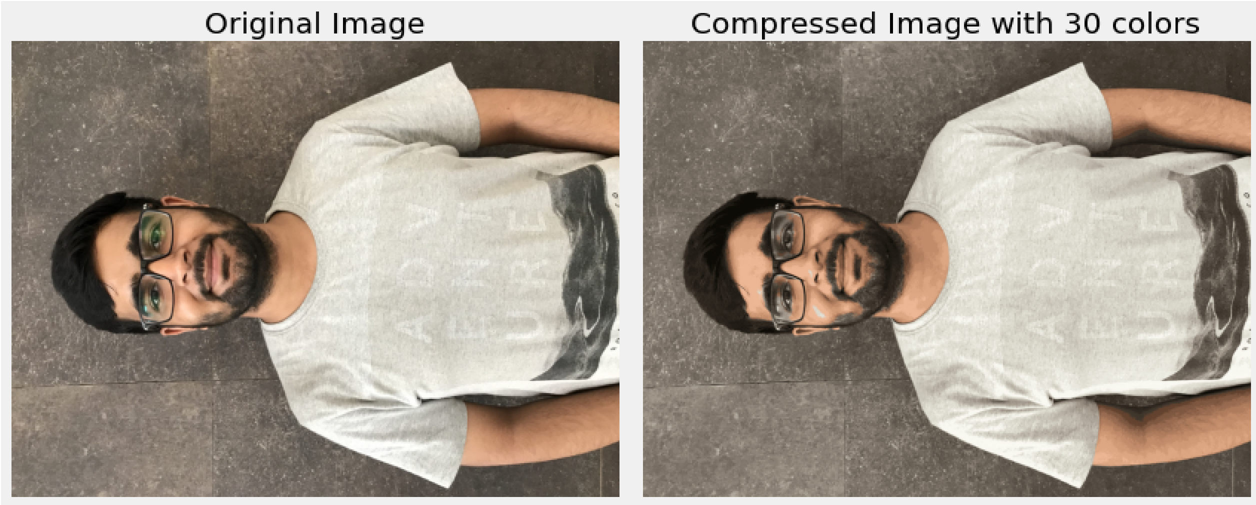
```
sse = 0
for i in range(len(X_test)):
  temp = np.linalg.norm(X_test[i]-kmeans.cluster_centers_[centroid[i]])
  temp = temp*temp
  sse += temp
print(sse)
```

```
0.30321005588436034
```

Kmeans on Image Compression:

In this part, we'll implement kmeans to compress an image. The image that we'll be working on is 3
location we would have 3 8-bit integers that specify the red, green, and blue intensity values. Our go
and represent (compress) the photo using those 30 colors only. To pick which colors to use, we'll u
every pixel as a data point. That means reshape the image from height x width x channels to (heigh
396 = 156,816 data points in 3-dimensional space which are the intensity of RGB. Doing so will allo
centroids for each pixel and would significantly reduce the size of the image by a factor of 6. The o
3,763,584 bits; however, the new compressed image would be 30 x 24 + 396 x 396 x 4 = 627,984 bi
that we'll be using centroids as a lookup for pixels' colors and that would reduce the size of each p

```
# Read the image
img = imread('IMG_6115.jpeg')
img_size = img.shape

# Reshape it to be 2-dimension
X = img.reshape(img_size[0] * img_size[1], img_size[2])

# Run the Kmeans algorithmzzz
km = KMeans(n_clusters=30)
km.fit(X)

# Use the centroids to compress the image
X_compressed = km.cluster_centers_[km.labels_]
X_compressed = np.clip(X_compressed.astype('uint8'), 0, 255)

# Reshape X_recovered to have the same dimension as the original image 128 * 128
X_compressed = X_compressed.reshape(img_size[0], img_size[1], img_size[2])

# Plot the original and the compressed image next to each other
fig, ax = plt.subplots(1, 2, figsize = (12, 8))
ax[0].imshow(img)
ax[0].set_title('Original Image',)
ax[1].imshow(X_compressed)
ax[1].set_title('Compressed Image with 30 colors')
for ax in fig.axes:
    ax.axis('off')
plt.tight_layout();
```


Original Image
Compressed Image with 30 colors