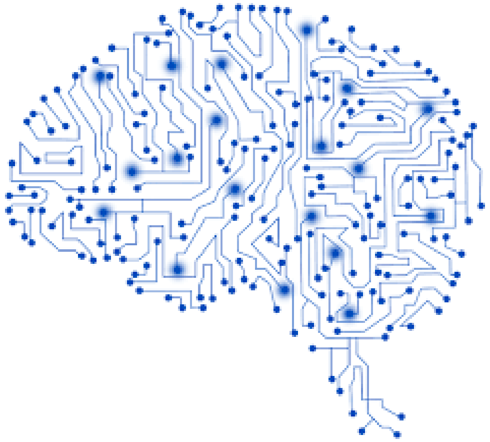
University of Minho
School of Engineering



Dados e Aprendizagem Automática

Unsupervised Learning

DAA @ MEI/1º ano – 1º Semestre

Bruno Fernandes, Filipe Gonçalves, Victor Alves, Cesar Analide

Part VI

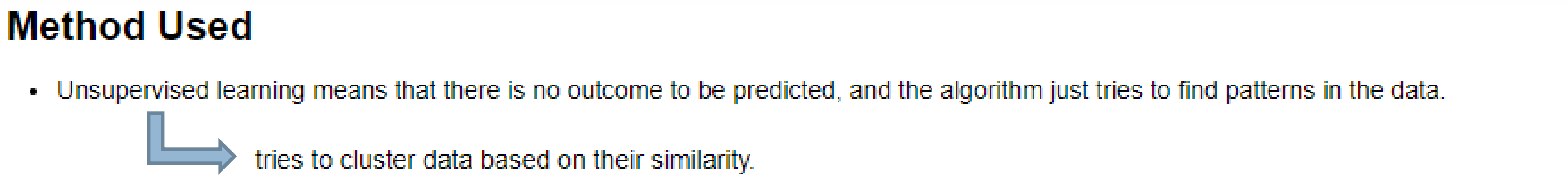
Contents

- Unsupervised Learning
    - K-Means Clustering
    - K-Medoids Clustering
- Hands On

Unsupervised Learning

Unsupervised Learning

- Exercise:
  - **Problem:** Development of a Machine Learning Model able to cluster data based on their similarity
  - **Classification Approach:** Clustering approaches to solve this problem



Method Used

- Unsupervised learning means that there is no outcome to be predicted, and the algorithm just tries to find patterns in the data.

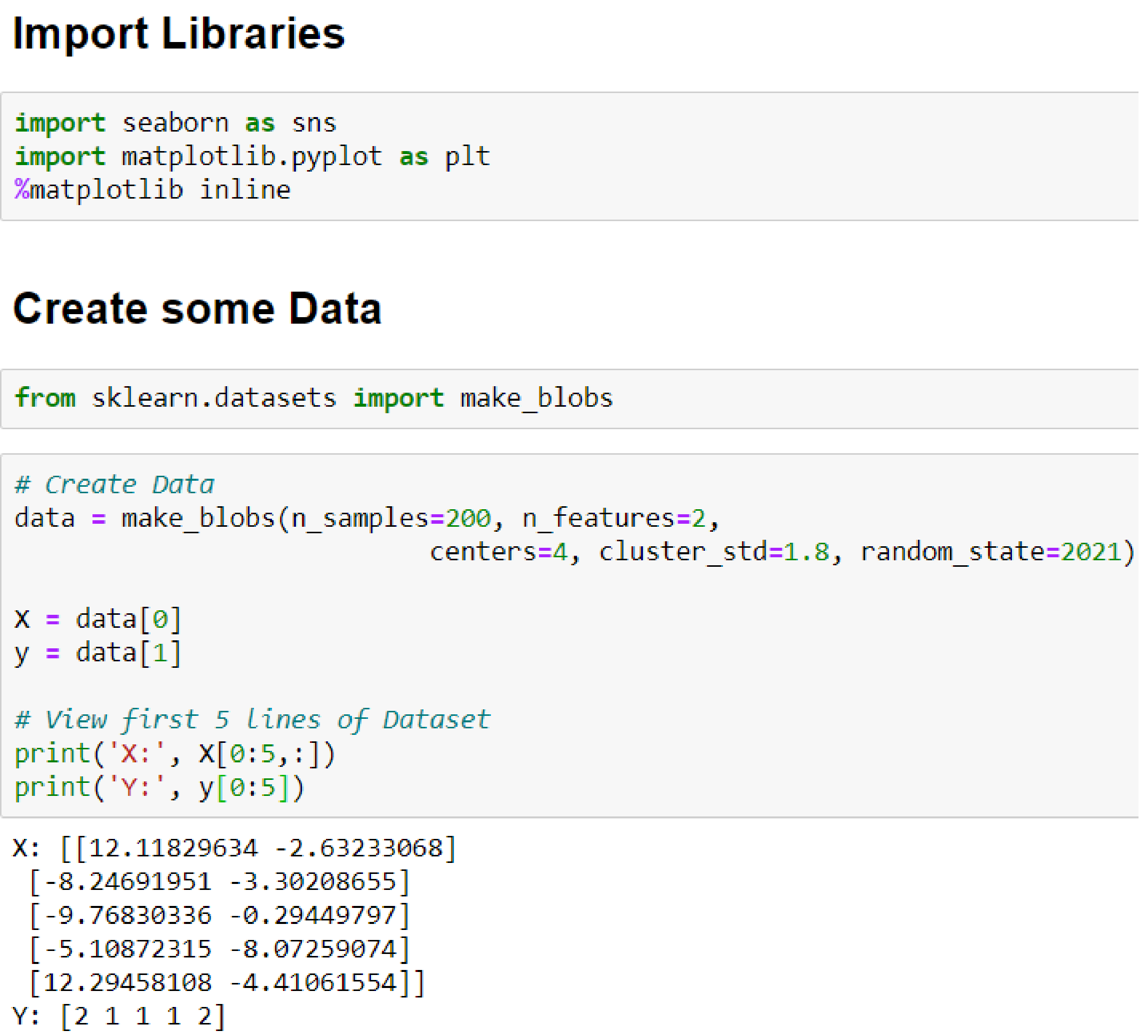
  → tries to cluster data based on their similarity.

  - **Dataset:** Generate isotropic Gaussian blobs dataset for clustering using *sklearn.datasets.make_blobs*

  - **Note** to apply K-Medoids Clustering, required to install package *scikit-learn-extra*

```
conda install -c conda-forge scikit-learn-extra
```

Unsupervised Learning



Import Libraries

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Create some Data

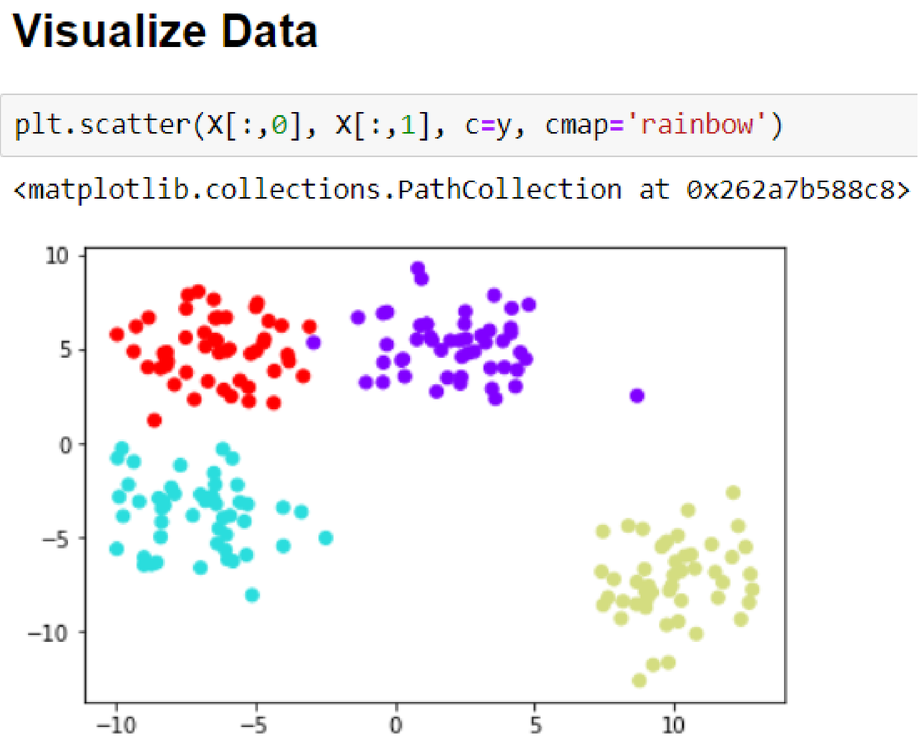
```
from sklearn.datasets import make_blobs
```

```
# Create Data
data = make_blobs(n_samples=200, n_features=2,
                           centers=4, cluster_std=1.8, random_state=2021)

X = data[0]
y = data[1]

# View first 5 lines of Dataset
print('X:', X[0:5,:])
print('Y:', y[0:5])
```

```
X: [[12.11829634 -2.63233068]
 [-8.24691951 -3.30208655]
 [-9.76830336 -0.29449797]
 [-5.10872315 -8.07259074]
 [12.29458108 -4.41061554]]
Y: [2 1 1 1 2]
```

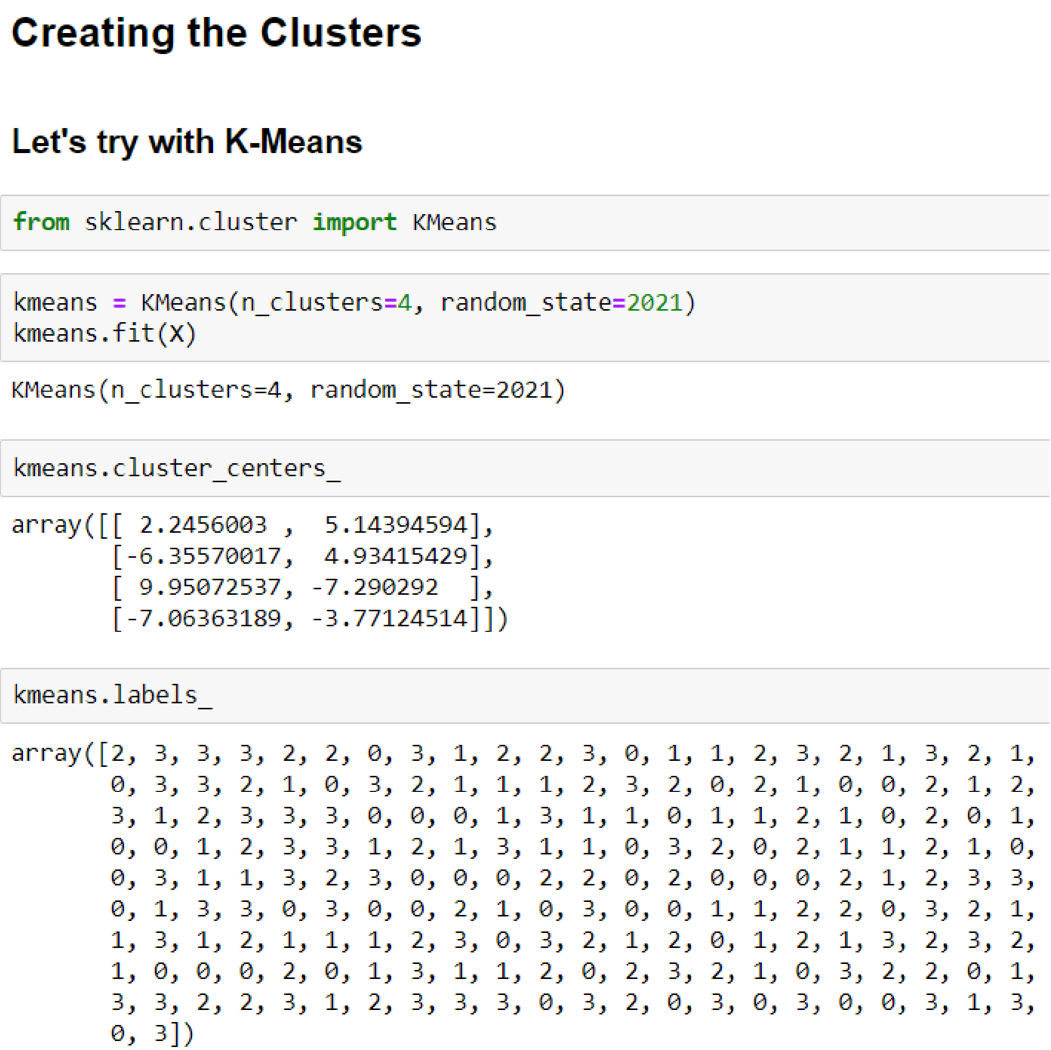


Visualize Data

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

```
<matplotlib.collections.PathCollection at 0x262a7b588c8>
```

Unsupervised Learning



Creating the Clusters

Let's try with K-Means

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=4, random_state=2021)
kmeans.fit(X)
```

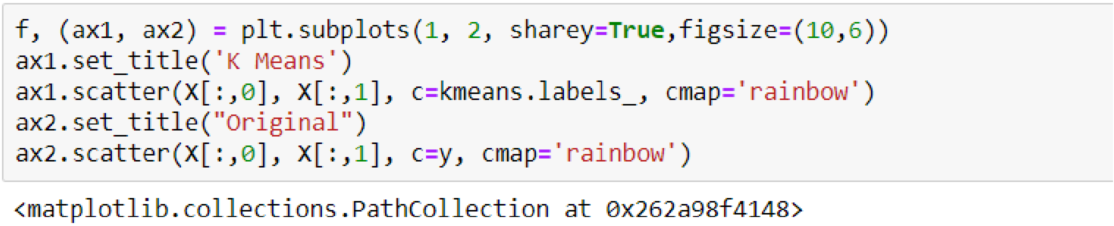
```
KMeans(n_clusters=4, random_state=2021)
```

```
kmeans.cluster_centers_
```

```
array([[ 2.2456003 ,  5.14394594],
       [-6.35570017,  4.93415429],
       [ 9.95072537, -7.290292  ],
       [-7.06363189, -3.77124514]])
```

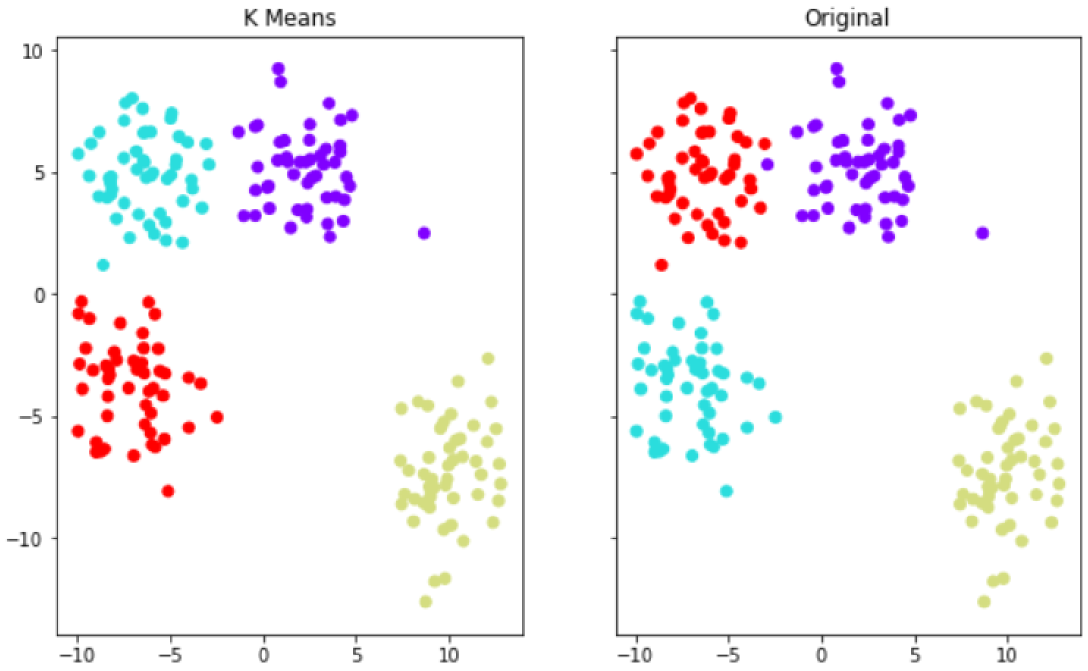
```
kmeans.labels_
```

```
array([2, 3, 3, 3, 2, 2, 0, 3, 1, 2, 2, 3, 0, 1, 1, 2, 3, 2, 1, 3, 2, 1,
       0, 3, 3, 2, 1, 0, 3, 2, 1, 1, 1, 2, 3, 2, 0, 2, 1, 0, 0, 2, 1, 2,
       3, 1, 2, 3, 3, 3, 0, 0, 0, 1, 3, 1, 1, 0, 1, 1, 2, 1, 0, 2, 0, 1,
       0, 0, 1, 2, 3, 3, 1, 2, 1, 3, 1, 1, 0, 3, 2, 0, 2, 1, 1, 2, 1, 0,
       0, 3, 1, 1, 3, 2, 3, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 1, 2, 3, 3,
       0, 1, 3, 3, 0, 3, 0, 0, 2, 1, 0, 3, 0, 0, 1, 1, 2, 2, 0, 3, 2, 1,
       1, 3, 1, 2, 1, 1, 1, 2, 3, 0, 3, 2, 1, 2, 0, 1, 2, 1, 3, 2, 3, 2,
       1, 0, 0, 0, 2, 0, 1, 3, 1, 1, 2, 0, 2, 3, 2, 1, 0, 3, 2, 2, 0, 1,
       3, 3, 2, 2, 3, 1, 2, 3, 3, 3, 0, 3, 2, 0, 3, 0, 3, 0, 0, 3, 1, 3,
       0, 3])
```



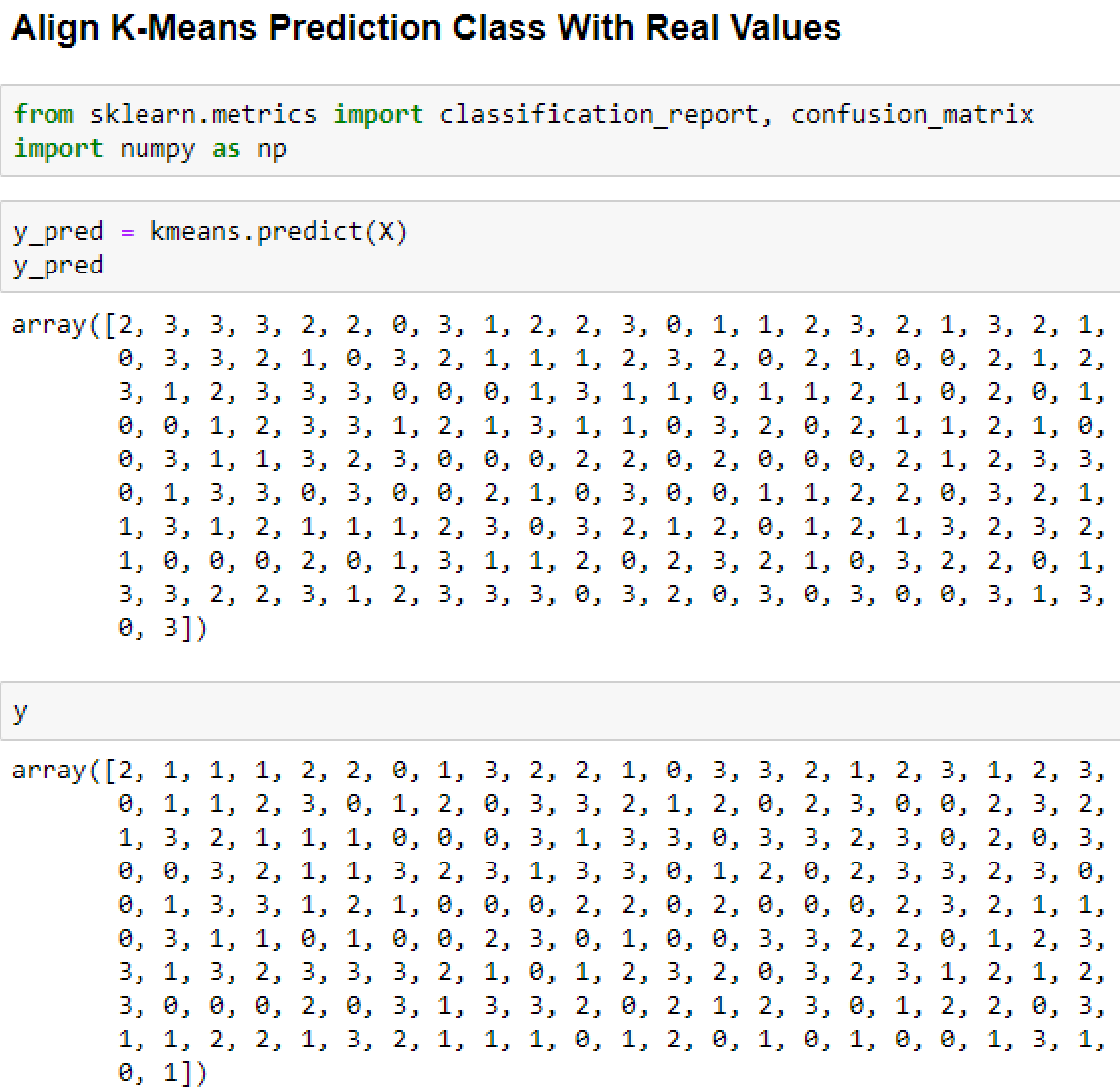
```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(10,6))
ax1.set_title('K Means')
ax1.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(X[:,0], X[:,1], c=y, cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x262a98f4148>
```



You should note, the colors are meaningless in reference between the two plots.

Unsupervised Learning



Align K-Means Prediction Class With Real Values

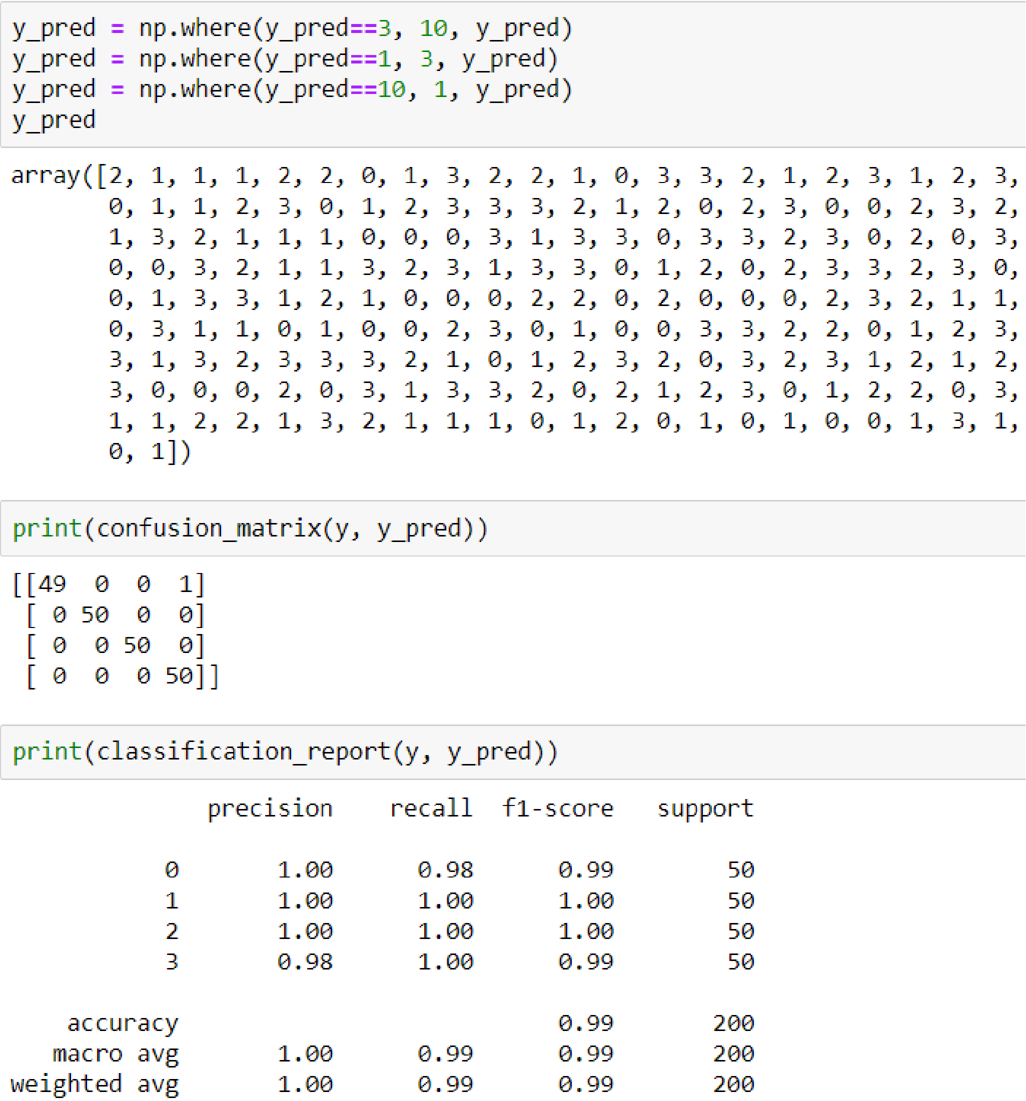
```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
```

```
y_pred = kmeans.predict(X)
y_pred
```

```
array([2, 3, 3, 3, 2, 2, 0, 3, 1, 2, 2, 3, 0, 1, 1, 2, 3, 2, 1, 3, 2, 1,
       0, 3, 3, 2, 1, 0, 3, 2, 1, 1, 1, 2, 3, 2, 0, 2, 1, 0, 0, 2, 1, 2,
       3, 1, 2, 3, 3, 3, 0, 0, 0, 1, 3, 1, 1, 0, 1, 1, 2, 1, 0, 2, 0, 1,
       0, 0, 1, 2, 3, 3, 1, 2, 1, 3, 1, 1, 0, 3, 2, 0, 2, 1, 1, 2, 1, 0,
       0, 3, 1, 1, 3, 2, 3, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 1, 2, 3, 3,
       0, 1, 3, 3, 0, 3, 0, 0, 2, 1, 0, 3, 0, 0, 1, 1, 2, 2, 0, 3, 2, 1,
       1, 3, 1, 2, 1, 1, 1, 2, 3, 0, 3, 2, 1, 2, 0, 1, 2, 1, 3, 2, 3, 2,
       1, 0, 0, 0, 2, 0, 1, 3, 1, 1, 2, 0, 2, 3, 2, 1, 0, 3, 2, 2, 0, 1,
       3, 3, 2, 2, 3, 1, 2, 3, 3, 3, 0, 3, 2, 0, 3, 0, 3, 0, 0, 3, 1, 3,
       0, 3])
```

```
y
```

```
array([2, 1, 1, 1, 2, 2, 0, 1, 3, 2, 2, 1, 0, 3, 3, 2, 1, 2, 3, 1, 2, 3,
       0, 1, 1, 2, 3, 0, 1, 2, 0, 3, 3, 2, 1, 2, 0, 2, 3, 0, 0, 2, 3, 2,
       1, 3, 2, 1, 1, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 2, 3, 0, 2, 0, 3,
       0, 0, 3, 2, 1, 1, 3, 2, 3, 1, 3, 3, 0, 1, 2, 0, 2, 3, 3, 2, 3, 0,
       0, 1, 3, 3, 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 3, 2, 1, 1,
       0, 3, 1, 1, 0, 1, 0, 0, 2, 3, 0, 1, 0, 0, 3, 3, 2, 2, 0, 1, 2, 3,
       3, 1, 3, 2, 3, 3, 3, 2, 1, 0, 1, 2, 3, 2, 0, 3, 2, 3, 1, 2, 1, 2,
       3, 0, 0, 0, 2, 0, 3, 1, 3, 3, 2, 0, 2, 1, 2, 3, 0, 1, 2, 2, 0, 3,
       1, 1, 2, 2, 1, 3, 2, 1, 1, 1, 0, 1, 2, 0, 1, 0, 1, 0, 0, 1, 3, 1,
       0, 1])
```



```
y_pred = np.where(y_pred==3, 10, y_pred)
y_pred = np.where(y_pred==1, 3, y_pred)
y_pred = np.where(y_pred==10, 1, y_pred)
y_pred
```

```
array([2, 1, 1, 1, 2, 2, 0, 1, 3, 2, 2, 1, 0, 3, 3, 2, 1, 2, 3, 1, 2, 3,
       0, 1, 1, 2, 3, 0, 1, 2, 3, 3, 3, 2, 1, 2, 0, 2, 3, 0, 0, 2, 3, 2,
       1, 3, 2, 1, 1, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 2, 3, 0, 2, 0, 3,
       0, 0, 3, 2, 1, 1, 3, 2, 3, 1, 3, 3, 0, 1, 2, 0, 2, 3, 3, 2, 3, 0,
       0, 1, 3, 3, 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 3, 2, 1, 1,
       0, 3, 1, 1, 0, 1, 0, 0, 2, 3, 0, 1, 0, 0, 3, 3, 2, 2, 0, 1, 2, 3,
       3, 1, 3, 2, 3, 3, 3, 2, 1, 0, 1, 2, 3, 2, 0, 3, 2, 3, 1, 2, 1, 2,
       3, 0, 0, 0, 2, 0, 3, 1, 3, 3, 2, 0, 2, 1, 2, 3, 0, 1, 2, 2, 0, 3,
       1, 1, 2, 2, 1, 3, 2, 1, 1, 1, 0, 1, 2, 0, 1, 0, 1, 0, 0, 1, 3, 1,
       0, 1])
```

```
print(confusion_matrix(y, y_pred))
```

```
[[49  0  0  1]
 [ 0 50  0  0]
 [ 0  0 50  0]
 [ 0  0  0 50]]
```

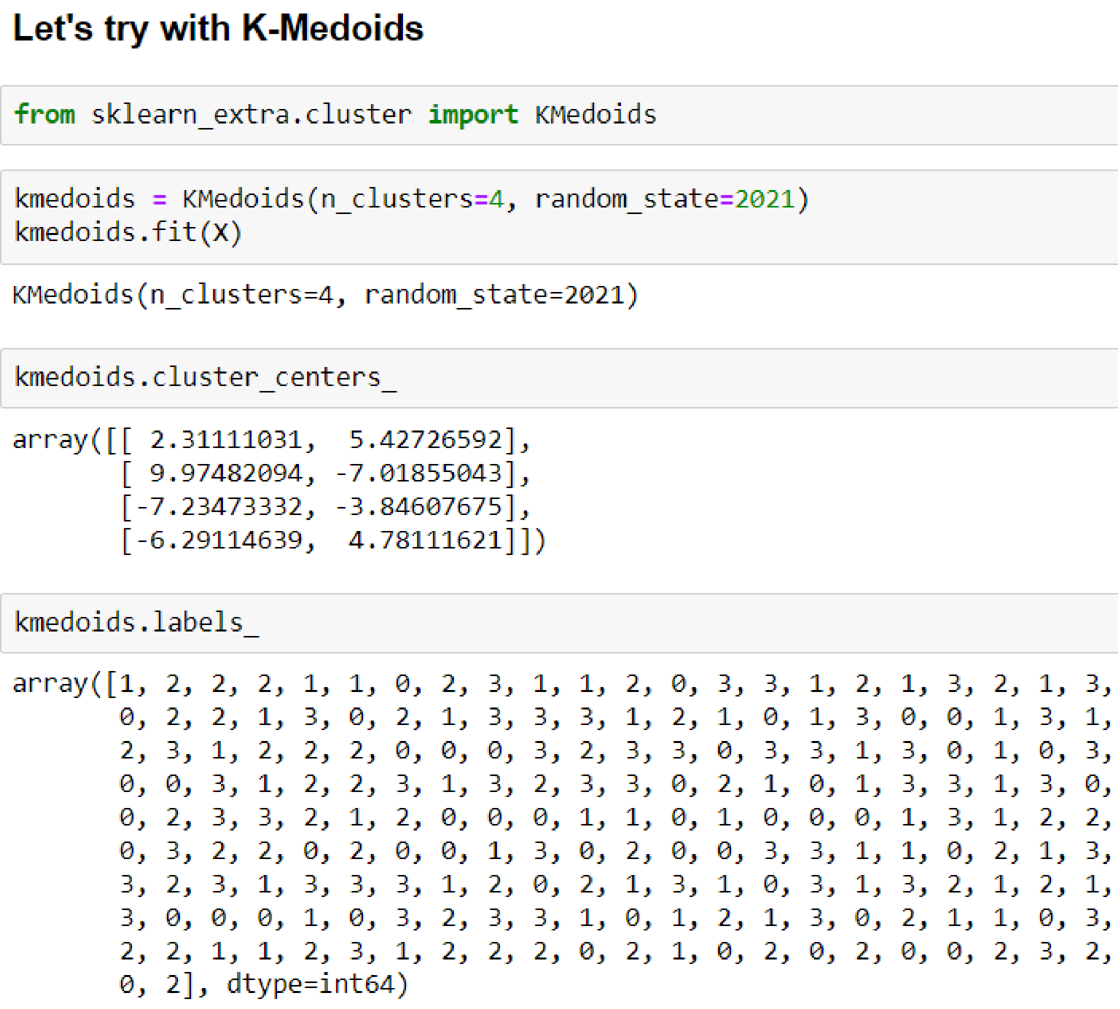
```
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50
           3       0.98      1.00      0.99        50

    accuracy                           0.99       200
   macro avg       1.00      0.99      0.99       200
weighted avg       1.00      0.99      0.99       200
```

Unsupervised Learning



Let's try with K-Medoids

```
from sklearn_extra.cluster import KMedoids
```

```
kmedoids = KMedoids(n_clusters=4, random_state=2021)
kmedoids.fit(X)
```

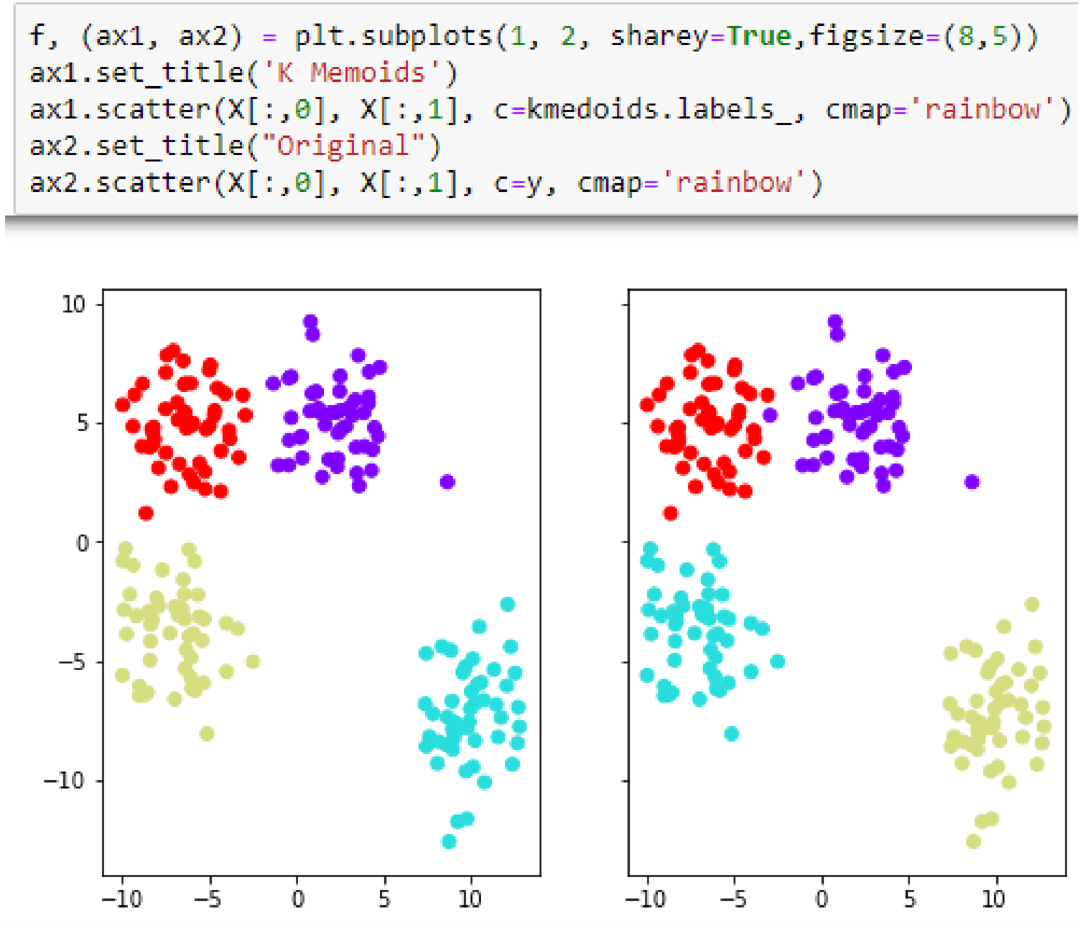
```
KMedoids(n_clusters=4, random_state=2021)
```

```
kmedoids.cluster_centers_
```

```
array([[ 2.31111031,  5.42726592],
       [ 9.97482094, -7.01855043],
       [-7.23473332, -3.84607675],
       [-6.29114639,  4.78111621]])
```

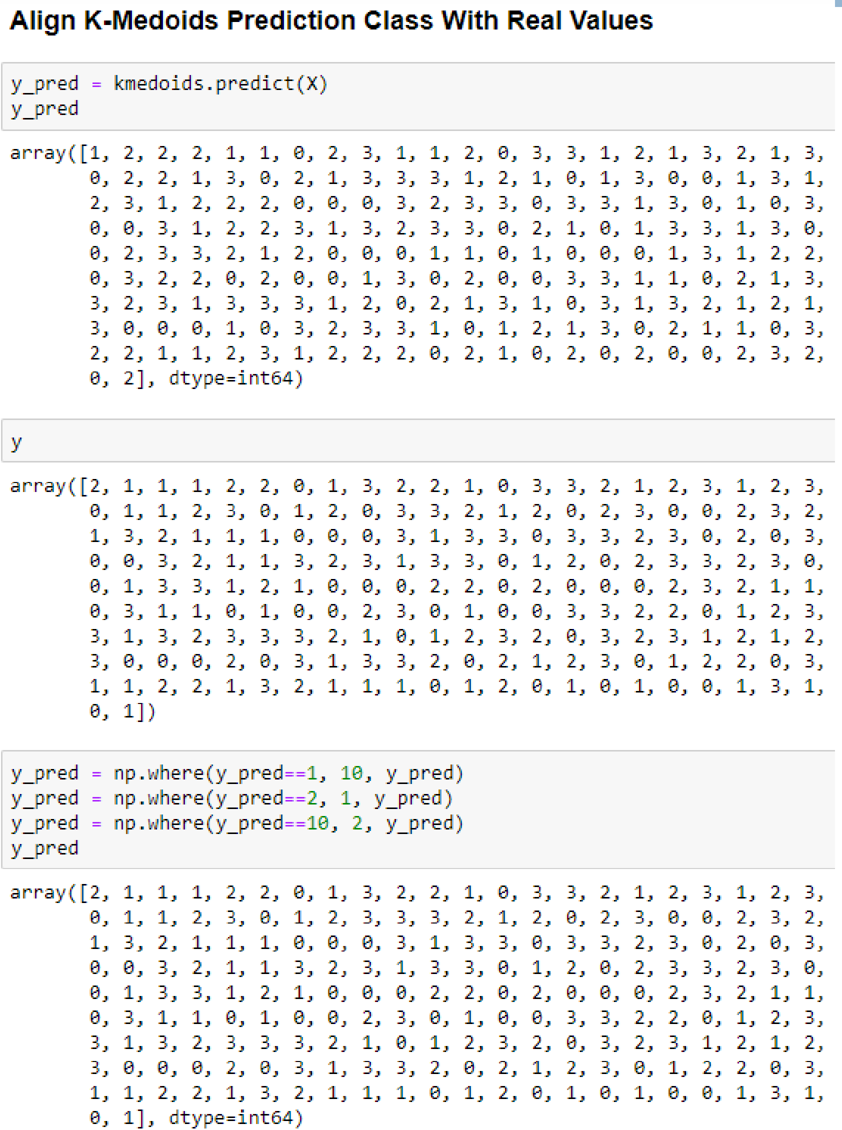
```
kmedoids.labels_
```

```
array([1, 2, 2, 2, 1, 1, 0, 2, 3, 1, 1, 2, 0, 3, 3, 1, 2, 1, 3, 2, 1, 3,
       0, 2, 2, 1, 3, 0, 2, 1, 3, 3, 3, 1, 2, 1, 0, 1, 3, 0, 0, 1, 3, 1,
       2, 3, 1, 2, 2, 2, 0, 0, 0, 3, 2, 3, 3, 0, 3, 3, 1, 3, 0, 1, 0, 3,
       0, 0, 3, 1, 2, 2, 3, 1, 3, 2, 3, 3, 0, 2, 1, 0, 1, 3, 3, 1, 3, 0,
       0, 2, 3, 3, 2, 1, 2, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 3, 1, 2, 2,
       0, 3, 2, 2, 0, 2, 0, 0, 1, 3, 0, 2, 0, 0, 3, 3, 1, 1, 0, 2, 1, 3,
       3, 2, 3, 1, 3, 3, 3, 1, 2, 0, 2, 1, 3, 1, 0, 3, 1, 3, 2, 1, 2, 1,
       3, 0, 0, 0, 1, 0, 3, 2, 3, 3, 1, 0, 1, 2, 1, 3, 0, 2, 1, 1, 0, 3,
       2, 2, 1, 1, 2, 3, 1, 2, 2, 2, 0, 2, 1, 0, 2, 0, 2, 0, 0, 2, 3, 2,
       0, 2], dtype=int64)
```



```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(8,5))
ax1.set_title('K Memoids')
ax1.scatter(X[:,0], X[:,1], c=kmedoids.labels_, cmap='rainbow')
ax2.set_title("Original")
ax2.scatter(X[:,0], X[:,1], c=y, cmap='rainbow')
```

Unsupervised Learning



Align K-Medoids Prediction Class With Real Values

```
y_pred = kmedoids.predict(X)
y_pred
```

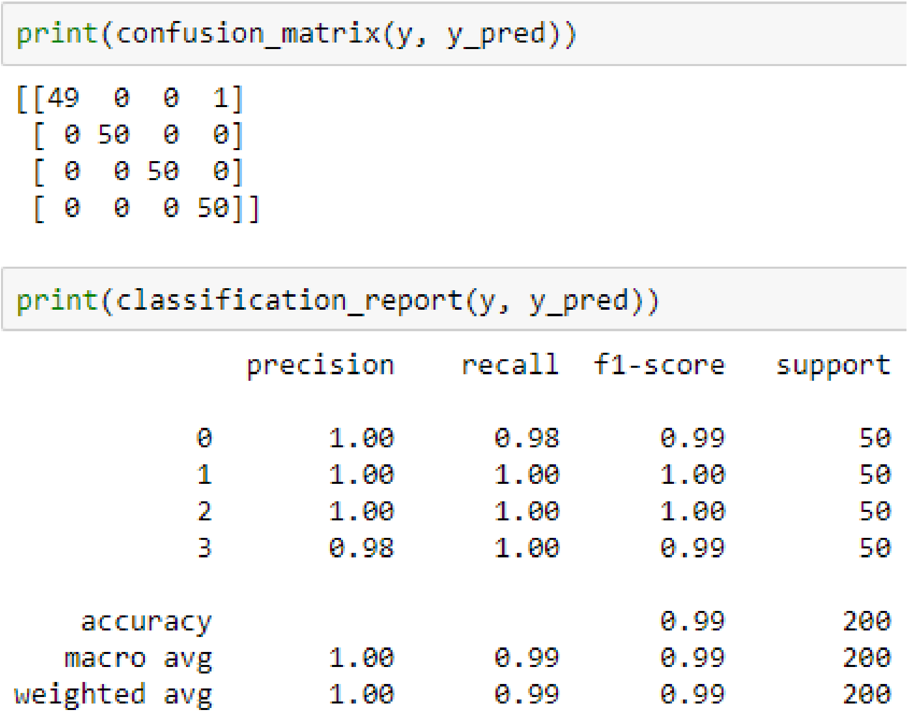
```
array([1, 2, 2, 2, 1, 1, 0, 2, 3, 1, 1, 2, 0, 3, 3, 1, 2, 1, 3, 2, 1, 3,
       0, 2, 2, 1, 3, 0, 2, 1, 3, 3, 3, 1, 2, 1, 0, 1, 3, 0, 0, 1, 3, 1,
       2, 3, 1, 2, 2, 2, 0, 0, 0, 3, 2, 3, 3, 0, 3, 3, 1, 3, 0, 1, 0, 3,
       0, 0, 3, 1, 2, 2, 3, 1, 3, 2, 3, 3, 0, 2, 1, 0, 1, 3, 3, 1, 3, 0,
       0, 2, 3, 3, 2, 1, 2, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 3, 1, 2, 2,
       0, 3, 2, 2, 0, 2, 0, 0, 1, 3, 0, 2, 0, 0, 3, 3, 1, 1, 0, 2, 1, 3,
       3, 2, 3, 1, 3, 3, 3, 1, 2, 0, 2, 1, 3, 1, 0, 3, 1, 3, 2, 1, 2, 1,
       3, 0, 0, 0, 1, 0, 3, 2, 3, 3, 1, 0, 1, 2, 1, 3, 0, 2, 1, 1, 0, 3,
       2, 2, 1, 1, 2, 3, 1, 2, 2, 2, 0, 2, 1, 0, 2, 0, 2, 0, 0, 2, 3, 2,
       0, 2], dtype=int64)
```

```
y
```

```
array([2, 1, 1, 1, 2, 2, 0, 1, 3, 2, 2, 1, 0, 3, 3, 2, 1, 2, 3, 1, 2, 3,
       0, 1, 1, 2, 3, 0, 1, 2, 0, 3, 3, 2, 1, 2, 0, 2, 3, 0, 0, 2, 3, 2,
       1, 3, 2, 1, 1, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 2, 3, 0, 2, 0, 3,
       0, 0, 3, 2, 1, 1, 3, 2, 3, 1, 3, 3, 0, 1, 2, 0, 2, 3, 3, 2, 3, 0,
       0, 1, 3, 3, 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 3, 2, 1, 1,
       0, 3, 1, 1, 0, 1, 0, 0, 2, 3, 0, 1, 0, 0, 3, 3, 2, 2, 0, 1, 2, 3,
       3, 1, 3, 2, 3, 3, 3, 2, 1, 0, 1, 2, 3, 2, 0, 3, 2, 3, 1, 2, 1, 2,
       3, 0, 0, 0, 2, 0, 3, 1, 3, 3, 2, 0, 2, 1, 2, 3, 0, 1, 2, 2, 0, 3,
       1, 1, 2, 2, 1, 3, 2, 1, 1, 1, 0, 1, 2, 0, 1, 0, 1, 0, 0, 1, 3, 1,
       0, 1])
```

```
y_pred = np.where(y_pred==1, 10, y_pred)
y_pred = np.where(y_pred==2, 1, y_pred)
y_pred = np.where(y_pred==10, 2, y_pred)
y_pred
```

```
array([2, 1, 1, 1, 2, 2, 0, 1, 3, 2, 2, 1, 0, 3, 3, 2, 1, 2, 3, 1, 2, 3,
       0, 1, 1, 2, 3, 0, 1, 2, 3, 3, 3, 2, 1, 2, 0, 2, 3, 0, 0, 2, 3, 2,
       1, 3, 2, 1, 1, 1, 0, 0, 0, 3, 1, 3, 3, 0, 3, 3, 2, 3, 0, 2, 0, 3,
       0, 0, 3, 2, 1, 1, 3, 2, 3, 1, 3, 3, 0, 1, 2, 0, 2, 3, 3, 2, 3, 0,
       0, 1, 3, 3, 1, 2, 1, 0, 0, 0, 2, 2, 0, 2, 0, 0, 0, 2, 3, 2, 1, 1,
       0, 3, 1, 1, 0, 1, 0, 0, 2, 3, 0, 1, 0, 0, 3, 3, 2, 2, 0, 1, 2, 3,
       3, 1, 3, 2, 3, 3, 3, 2, 1, 0, 1, 2, 3, 2, 0, 3, 2, 3, 1, 2, 1, 2,
       3, 0, 0, 0, 2, 0, 3, 1, 3, 3, 2, 0, 2, 1, 2, 3, 0, 1, 2, 2, 0, 3,
       1, 1, 2, 2, 1, 3, 2, 1, 1, 1, 0, 1, 2, 0, 1, 0, 1, 0, 0, 1, 3, 1,
       0, 1], dtype=int64)
```



```
print(confusion_matrix(y, y_pred))
```

```
[[49  0  0  1]
 [ 0 50  0  0]
 [ 0  0 50  0]
 [ 0  0  0 50]]
```

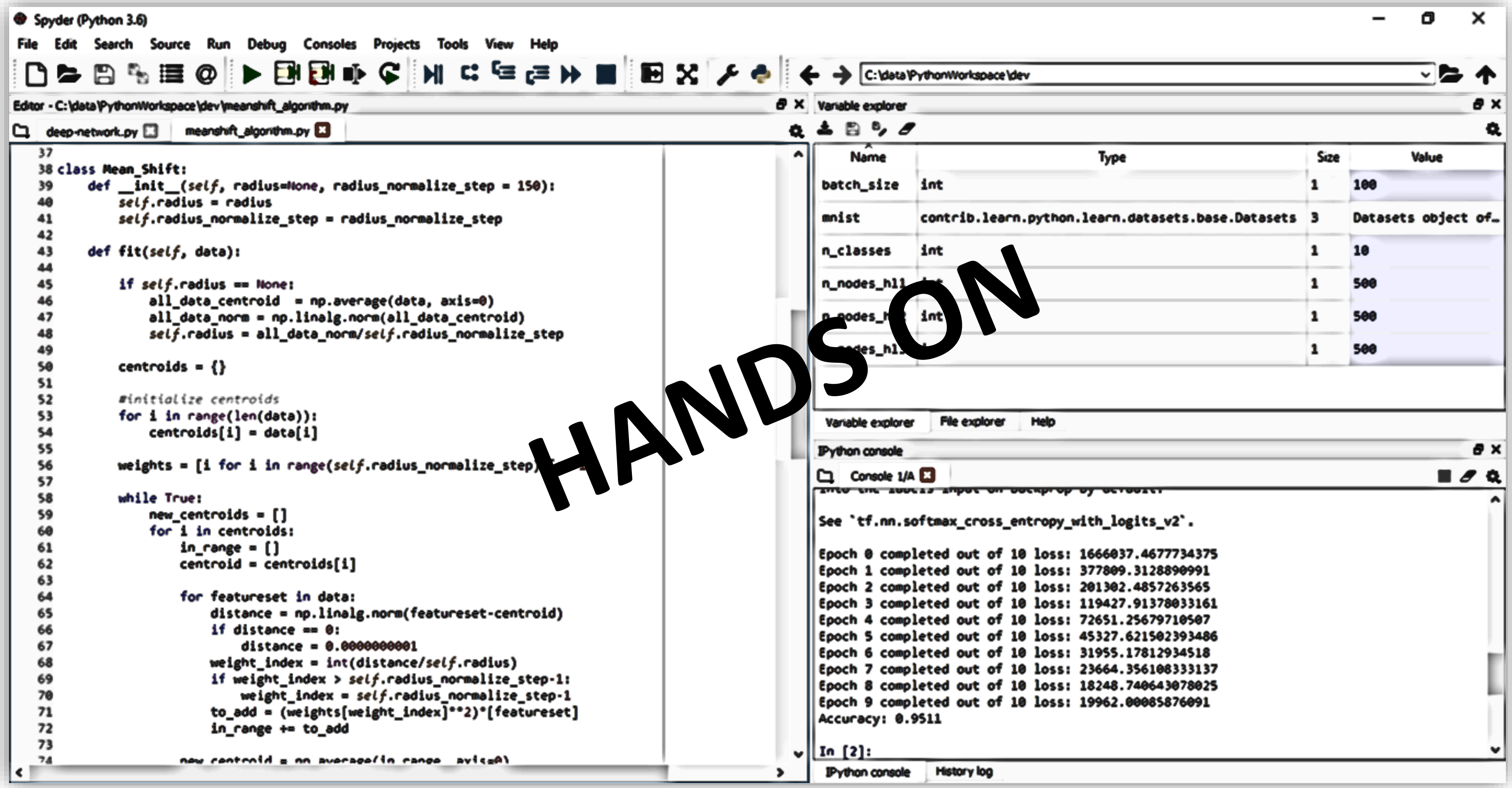
```
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50
           3       0.98      1.00      0.99        50

    accuracy                           0.99       200
   macro avg       1.00      0.99      0.99       200
weighted avg       1.00      0.99      0.99       200
```

Hands On


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\data\PythonWorkspace\dev
Editor - C:\data\PythonWorkspace\dev\meanshift_algorithm.py
deep-network.py
meanshift_algorithm.py
Variable explorer
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log

```python

class Mean_Shift:
    def __init__(self, radius=None, radius_normalize_step = 150):
        self.radius = radius
        self.radius_normalize_step = radius_normalize_step

    def fit(self, data):

        if self.radius == None:
            all_data_centroid  = np.average(data, axis=0)
            all_data_norm = np.linalg.norm(all_data_centroid)
            self.radius = all_data_norm/self.radius_normalize_step

        centroids = {}

        #initialize centroids
        for i in range(len(data)):
            centroids[i] = data[i]

        weights = [i for i in range(self.radius_normalize_step)

        while True:
            new_centroids = []
            for i in centroids:
                in_range = []
                centroid = centroids[i]

                for featureset in data:
                    distance = np.linalg.norm(featureset-centroid)
                    if distance == 0:
                        distance = 0.0000000001
                    weight_index = int(distance/self.radius)
                    if weight_index > self.radius_normalize_step-1:
                        weight_index = self.radius_normalize_step-1
                    to_add = (weights[weight_index]**2)*[featureset]
                    in_range += to_add

            new_centroid = np.average(in_range, axis=0)
```

Name	Type	Size	Value
batch_size	int	1	100
mnist	contrib.learn.python.learn.datasets.base.Datasets	3	Datasets object of…
n_classes	int	1	10
n_nodes_hl1	[illegible]	1	500
n_nodes_h[illegible]	int	1	500
[illegible]odes_hl[illegible]	[illegible]	1	500

```
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Epoch 0 completed out of 10 loss: 1666037.4677734375
Epoch 1 completed out of 10 loss: 377809.3128890991
Epoch 2 completed out of 10 loss: 201302.4857263565
Epoch 3 completed out of 10 loss: 119427.91378033161
Epoch 4 completed out of 10 loss: 72651.25679710507
Epoch 5 completed out of 10 loss: 45327.621502393486
Epoch 6 completed out of 10 loss: 31955.17812934518
Epoch 7 completed out of 10 loss: 23664.356108333137
Epoch 8 completed out of 10 loss: 18248.740643078025
Epoch 9 completed out of 10 loss: 19962.00085876091
Accuracy: 0.9511

In [2]:
```

HANDS ON