Homework 2 MLE

Bruno Morgado
RIN: 661995422

Question 1: Iris Data SVM with extra features

In [ ]:
```
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data # only selecting first three features

#Intialize matrix for new features
new_features = np.ones((np.shape(X)[0], 2))

#creating new features by multiplying two initial features
new_features[:,0] =  X[:,0]*X[:,1]
new_features[:,1] =  X[:,2]*X[:,3]

X = np.concatenate((X, new_features),axis=1)
y = iris.target
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test=train_test_split(
    X,y,
    test_size=0.40,
    train_size=0.60,
    random_state=123,
    shuffle=True,
    stratify=y)

#Importing the SVM
from sklearn import svm

#Training SVM
clf = svm.SVC()
clf.fit(X_train, y_train)

#Extracting data from prodicted
preds = clf.predict(X_test)

#Evaluate
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test,preds)
print('accuracy score :', acc)
```

```
accuracy score : 0.95
```

With the two arbitrary new features the accuracy score increased to 0.95 which is in fact higher than the 0.933 of the previous example. This suggests that there may be some non-linear features which can be used for classification.

Question 2: MNIST Digits classifier

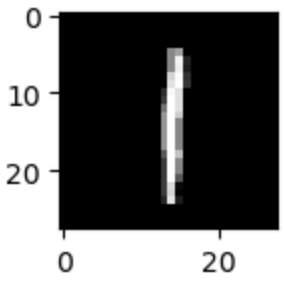
In [ ]:
```
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```
WARNING:tensorflow:From c:\Users\MorgadoBruno\AppData\Local\anaconda3\envs\ML\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

In [ ]:
```
#Loading the dataset
(train_X, train_y), (test_X, test_y) = mnist.load_data()

#printing the shapes of the vectors
print('X_train: ' + str(train_X.shape))
print('Y_train: ' + str(train_y.shape))
print('X_test:  '  + str(test_X.shape))
print('Y_test:  '  + str(test_y.shape))
```

```
X_train: (60000, 28, 28)
Y_train: (60000,)
X_test:  (10000, 28, 28)
Y_test:  (10000,)
```

In [ ]:
```
# Plot a subset of images

from matplotlib import pyplot
pyplot.subplot(330+ 1)
pyplot.imshow(train_X[454], cmap=pyplot.get_cmap('gray'))
pyplot.show()
```



In [ ]:
```
# Organize data set such that only the even numbers are present in the input data
# Exctract the indeces holding the even numbers
indeces = []
for i, a in enumerate(train_y):
    if a%2 == 0:
        indeces.append(i)

# have the data with the correct indeces
train_X_even = train_X[indeces,:,:]
train_y_even = train_y[indeces]
```

```
vec_train_X_even = np.zeros([train_y_even.size, 28*28])
for i in range(train_y_even.size):
    vec_train_X_even[i,:] = train_X_even[i].flatten()
```

In [ ]:
```
# Repeat the above steps for the test data set
indeces = []
for i, a in enumerate(test_y):
    if a%2 == 0:
        indeces.append(i)

# have the data with the correct indeces
test_X_even = test_X[indeces,:,:]
test_y_even = test_y[indeces]

vec_test_X_even = np.zeros([test_y_even.size, 28*28])
for i in range(test_y_even.size):
    vec_test_X_even[i,:] = test_X_even[i].flatten()
```

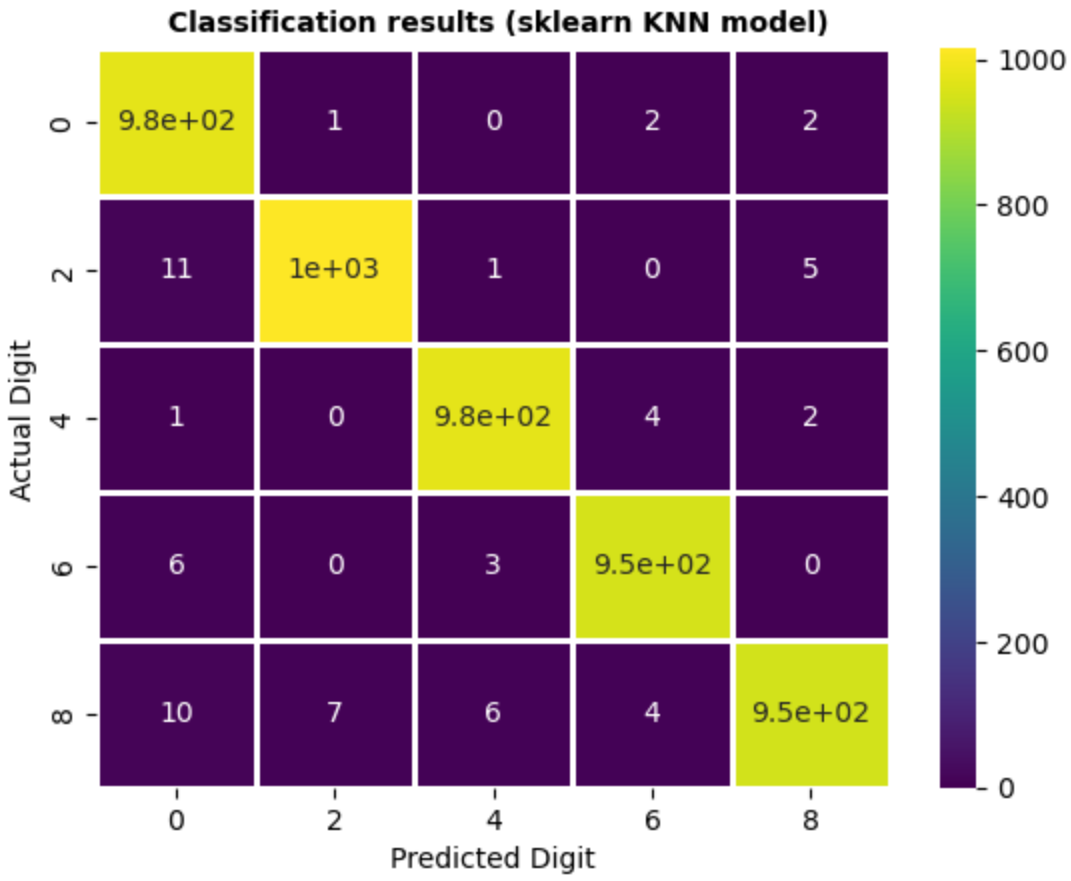
In [ ]:
```
skmodel = KNeighborsClassifier(n_neighbors=3)
skmodel.fit(vec_train_X_even,train_y_even)

print('preds:', skmodel.predict(vec_test_X_even))
print('actual:', test_y_even)
print(classification_report(test_y_even, skmodel.predict(vec_test_X_even)))
```

```
preds: [2 0 4 ... 2 4 6]
actual: [2 0 4 ... 2 4 6]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           2       0.99      0.98      0.99      1032
           4       0.99      0.99      0.99       982
           6       0.99      0.99      0.99       958
           8       0.99      0.97      0.98       974

    accuracy                           0.99      4926
   macro avg       0.99      0.99      0.99      4926
weighted avg       0.99      0.99      0.99      4926
```

This data is very high dimensional and it would be very difficult to include all possible features in a pairplot. In this sense only once some feature engineering has been done could we extract some results.

In [ ]:
```
# Plotting data in a heatmap
cm_sk = confusion_matrix(test_y_even, skmodel.predict(vec_test_X_even))
ax = sns.heatmap(cm_sk,linewidths=2, annot=True, cmap='viridis', cbar=True)

ax.set_xticklabels([0,2,4,6,8])
ax.set_yticklabels([0,2,4,6,8])
plt.xlabel('Predicted Digit')
plt.ylabel('Actual Digit')
plt.title(' Classification results (sklearn KNN model)', fontsize='medium', fontweight
```

Out[ ]:
```
Text(0.5, 1.0, ' Classification results (sklearn KNN model)')
```



In [ ]:
```
print("the number of sixes correctly classified were: " ,cm_sk[3,3])
```

```
the number of sixes correctly classified were:  949
```

Question 3: Writing a k-means algorithm

In [ ]:
```
class BM_KMeans:
    """My K-Means classifier based on Dr. Chowdury's, but with a k++ implementation.
    """
    def __init__(self, k):
        self.k = k
        self.cluster_labels = np.arange(self.k)

    # Helper Functions

    def _init_centroids(self, data):
        """This initializes the centroids using the k++ method

        Args:
            data (ndarray): input data in matrix form mxn where m is the number of ent

        Returns:
            ndarray: This is an array which holds the centroids of the data where m is
        """
        centroids = np.zeros([self.k, data.shape[1]]) # initializes the data structure
        centroids[0,:] = data[np.random.choice(data.shape[0], 1),:] # out of the data

        for i in range(1,self.k): # from 1 to k find each subsequent centroid using th
            for centroid in centroids[:i,:]: # from the first centroid only up to the
                max_dist = 0
                distances = np.array(np.linalg.norm(data-centroid, axis=1))
```

```
                max_index = np.argmax(distances)
                if distances[max_index] > max_dist:
                    max_dist = distances[max_index]
                    centroids[i,:] = data[max_index,:]

        return centroids

    # Main Functions
    def fit(self, data, max_iter = 10000):
        """_summary_

        Args:
            data (ndarray): input data in matrix form mxn where m is the number of ent
            max_iter (int, optional): the maximum number of iterations before the whil

        Returns:
            None
        """

        self.centroids = self._init_centroids(data)
        i = 1
        while i<max_iter:
            distances = np.array([np.linalg.norm(data - centroid, axis=1) for centroid
            self.clusters = np.argmin(distances, axis=0)
            new_centroids = np.array([data[self.clusters == i, :].mean(axis=0) for i i
            # check convergence
            if np.allclose(new_centroids, self.centroids, atol=1e-05):
                break
            else:
                self.centroids = new_centroids
                i += 1
        return None

    def predict(self, data):
        """ Uses the fitted centroids to classify the data

        Args:
            data (ndarray): input data in matrix form mxn where m is the number of ent

        Returns:
            ndarray: a vector which tells you which cluster each data point belongs to
        """
        distances = np.array([np.linalg.norm(data - centroid, axis=1) for centroid in
        return self.cluster_labels[np.argmin(distances, axis=0)]
```

In [ ]:

```
## Import data and the required modules
from sklearn.datasets import load_iris
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

data = load_iris()

df = pd.DataFrame()
df['sepal length'] = data['data'][:,0]
df['sepal width'] = data['data'][:,1]
df['petal width'] = data['data'][:,3]
```

```
X = df.to_numpy() # These are our features

df['target'] = data['target']

y = df['target'].to_numpy()
```

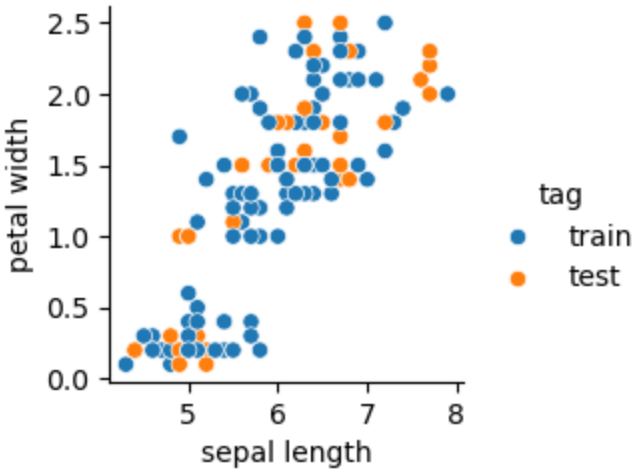
In [ ]:

```
## Create Test split and visialize it

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test, idx_train, idx_test=train_test_split(
    X,y, range(X.shape[0]),
    test_size=0.30,
    train_size=0.70,
    random_state=123,
    shuffle=True,
    stratify=y)

print('X_train shape :', X_train.shape)
print('X_test shape :', X_test.shape)
print('y_train shape :', y_train.shape)
print('y_test shape :', y_test.shape)

tag = []

for _ in range(X.shape[0]):
    if _ in idx_train:
        tag.append('train')
    else:
        tag.append('test')

df['tag'] = tag

sns.pairplot(df.drop(labels=['target'],axis=1), x_vars=['sepal length'],
    y_vars=['petal width'], hue='tag')
```

```
X_train shape : (105, 3)
X_test shape : (45, 3)
y_train shape : (105,)
y_test shape : (45,)
```

Out[ ]:

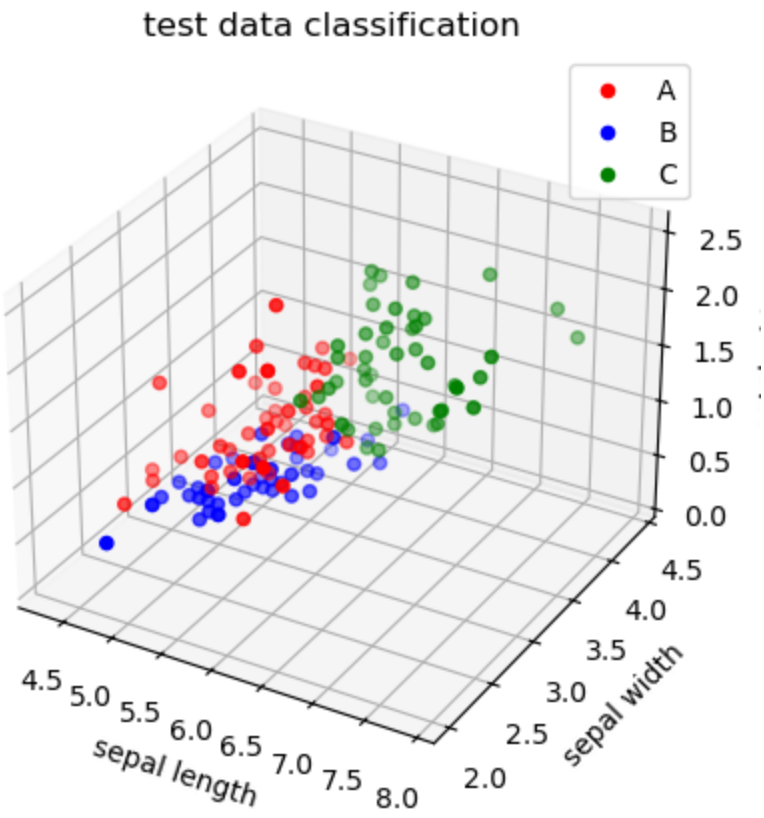
```
<seaborn.axisgrid.PairGrid at 0x190d93c94f0>
```



In [ ]:
```
# Training the centroids
model = BM_KMeans(k=3)
model.fit(X_train)
kmeans_labels = model.predict(X)
cluster_coords = model.centroids
print('predicted_labels:', kmeans_labels)
print('cluster centroids:', cluster_coords)
```

```
predicted_labels: [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 1 1 1 1 1 1 1 1 1 2 2 2 0 2 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
 0 2 2 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 2 2 2 0 0 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 2 2
 2 0]
cluster centroids: [[5.76153846 2.69487179 1.40769231]
 [5.04285714 3.43428571 0.26571429]
 [6.76774194 3.03225806 1.95806452]]
```

In [ ]:
```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import interactive
interactive(True)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

A = []
B = []
C = []

for i, label in enumerate(kmeans_labels):

    if label == 0:
        A.append(i)
    elif label == 1:
        B.append(i)
    else:
        C.append(i)

ax.scatter(X[A,0], X[A,1], X[A,2], c='r', label='A')
ax.scatter(X[B,0], X[B,1], X[B,2], c='b', label='B')
ax.scatter(X[C,0], X[C,1], X[C,2], c='g', label='C')

ax.set_xlabel('sepal length')
ax.set_ylabel('sepal width')
ax.set_zlabel('petal width')

ax.legend()

plt.title("test data classification") #title

# %matplotlib qt
# plt.show()

plt.figure().set_figwidth(8)
plt.figure().set_figheight(4)

%matplotlib inline
plt.show()
```

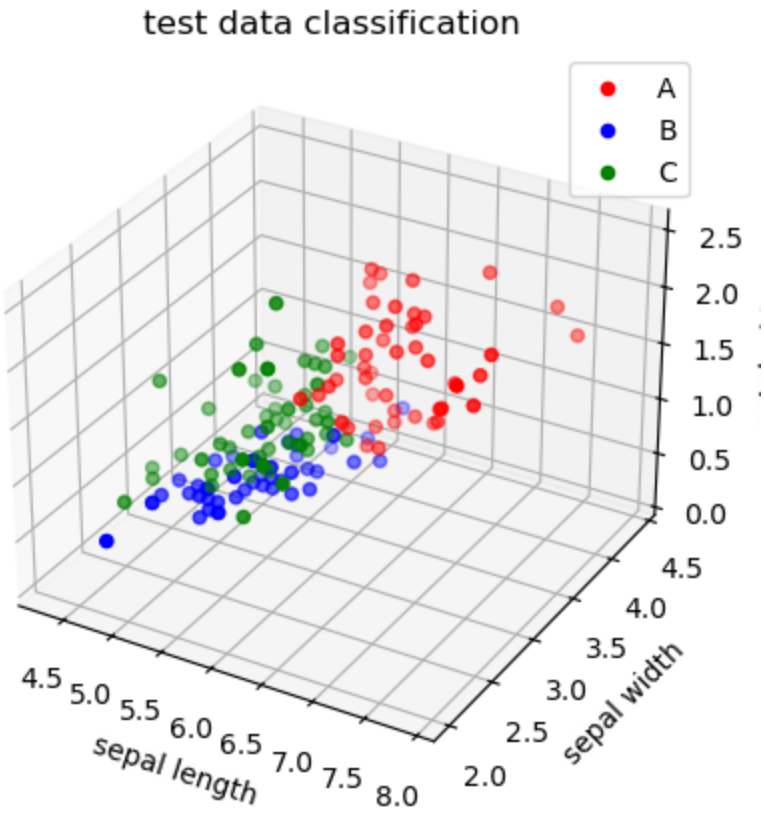



```
<Figure size 800x480 with 0 Axes>
<Figure size 640x400 with 0 Axes>
```

In [ ]:
```python
# using sklearn

from sklearn.cluster import KMeans
skmodel = KMeans(
    n_clusters=3,
    init='k-means++',
    n_init='auto',
    max_iter=300,
    tol=0.0001,
    verbose=0,
    random_state=None,
    copy_x=True,
    algorithm='lloyd',
)

skmodel.fit(X_train)
skmodel.predict(X)
skmodel_labels = skmodel.predict(X)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

A = []
B = []
C = []

for i, label in enumerate(skmodel_labels):

    if label == 0:
        A.append(i)
    elif label == 1:
```

```
        B.append(i)
    else:
        C.append(i)

ax.scatter(X[A,0], X[A,1], X[A,2], c='r', label='A')
ax.scatter(X[B,0], X[B,1], X[B,2], c='b', label='B')
ax.scatter(X[C,0], X[C,1], X[C,2], c='g', label='C')

ax.set_xlabel('sepal length')
ax.set_ylabel('sepal width')
ax.set_zlabel('petal width')

ax.legend()

plt.title("test data classification") #title

# %matplotlib qt
# plt.show()

plt.figure().set_figwidth(8)
plt.figure().set_figheight(4)

%matplotlib inline
plt.show()
```

```
c:\Users\MorgadoBruno\AppData\Local\anaconda3\envs\ML\lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



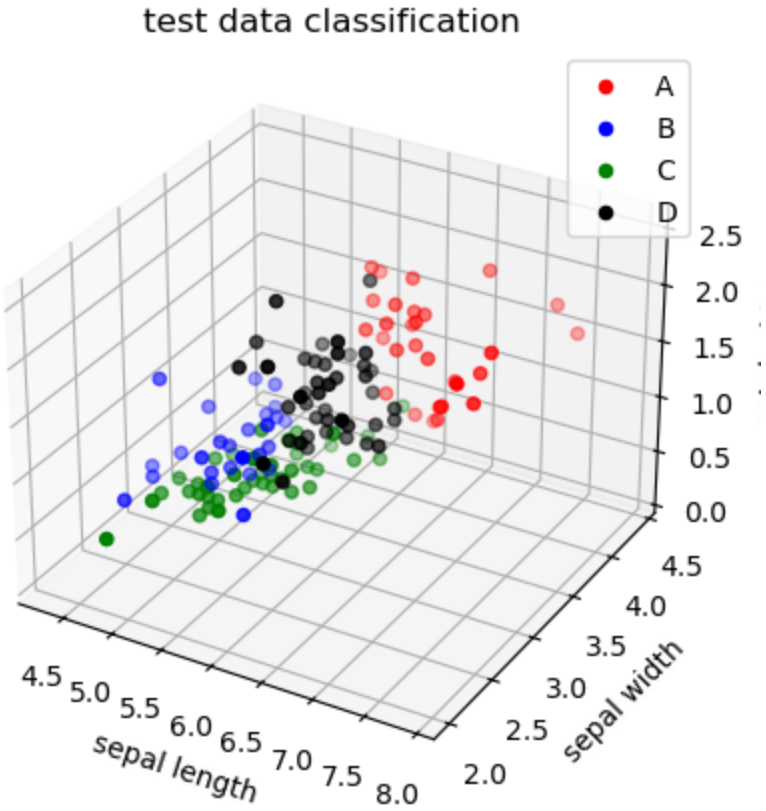

```
<Figure size 800x480 with 0 Axes>
<Figure size 640x400 with 0 Axes>
```

Overall the two models have shown good agreement between eachother especially since they both have k++ initializations.

In [ ]:
```
# using sklearn plus adding an extra category
from sklearn.cluster import KMeans
skmodel = KMeans(
    n_clusters=4,
    init='k-means++',
    n_init='auto',
    max_iter=300,
    tol=0.0001,
    verbose=0,
    random_state=None,
    copy_x=True,
    algorithm='lloyd',
)

skmodel.fit(X_train)
skmodel.predict(X)
skmodel_labels = skmodel.predict(X)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

A = []
B = []
C = []
D = []

for i, label in enumerate(skmodel_labels):

    if label == 0:
        A.append(i)
    elif label == 1:
        B.append(i)
    elif label == 2:
        C.append(i)
    else:
        D.append(i)

ax.scatter(X[A,0], X[A,1], X[A,2], c='r', label='A')
ax.scatter(X[B,0], X[B,1], X[B,2], c='b', label='B')
ax.scatter(X[C,0], X[C,1], X[C,2], c='g', label='C')
ax.scatter(X[D,0], X[D,1], X[D,2], c='k', label='D')

ax.set_xlabel('sepal length')
ax.set_ylabel('sepal width')
ax.set_zlabel('petal width')

ax.legend()

plt.title("test data classification") #title

# %matplotlib qt
# plt.show()

plt.figure().set_figwidth(8)
plt.figure().set_figheight(4)
```

```
%matplotlib inline
plt.show()
```

```
c:\Users\MorgadoBruno\AppData\Local\anaconda3\envs\ML\lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```




```
<Figure size 800x480 with 0 Axes>
<Figure size 640x400 with 0 Axes>
```

In [ ]:
```
#### Appendix: testing/debuging
data = np.array([[2,1,2],
                 [1,3,3],
                 [3,4,4],
                 [5,6,5],
                 [8,9,6]])

centroids = np.array([[1,1,1],
                      [2,2,2]])
cluster_labels = np.arange(2)
k = 2
distances = np.array([np.linalg.norm(data - centroid, axis=1) for centroid in centroid
clusters = np.argmin(distances, axis=0)
new_centroids = np.array([data[clusters == i, :].mean(axis=0) for i in range(k)])

cluster_labels[np.argmin(distances, axis=0)]
```

```
C:\Users\MorgadoBruno\AppData\Local\Temp\ipykernel_31580\4051839572.py:13: RuntimeWar
ning: Mean of empty slice.
  new_centroids = np.array([data[clusters == i, :].mean(axis=0) for i in range(k)])
c:\Users\MorgadoBruno\AppData\Local\anaconda3\envs\ML\lib\site-packages\numpy\core\_m
ethods.py:121: RuntimeWarning: invalid value encountered in divide
  ret = um.true_divide(
```

Out[ ]:
```
array([1, 1, 1, 1, 1])
```