```
#############################################
#David Nichols
#hw1
#https://github.com/iss-uncc/tinyml-hw1-DavidN0809.git
#############################################



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn

from sklearn.datasets import load_iris
rng = np.random.default_rng(2022)


## Here's the information needed to do the first few tasks,
# which will give you some practice with basic Python methods

list_of_names = ['Roger', 'Mary', 'Luisa', 'Elvis']
list_of_ages  = [23, 24, 19, 86]
list_of_heights_cm = [175, 162, 178, 182]

for name in list_of_names:
  print("The name {:} is {:} letters long".format(name, len(name)))
#part b
new_list = [x for x in list_of_ages]
print(new_list)

########################################
# Here's the information for the second part, involving the linear
# classifier

# import the iris dataset as a pandas dataframe
iris_db = load_iris(as_frame=True)
x_data = iris_db['data']
y_labels = iris_db['target'] # correct numeric labels
target_names = iris_db['target_names'] # string names

# Here's a starter example of plotting the data
fig=plt.figure(figsize=(6,6), dpi= 100, facecolor='w', edgecolor='k')
l_colors = ['maroon', 'darkgreen', 'blue']
for n, species in enumerate(target_names):
  plt.scatter(x_data[y_labels==n].iloc[:,0],
              x_data[y_labels==n].iloc[:,1],
              c=l_colors[n], label=target_names[n])
plt.xlabel(iris_db['feature_names'][0])
plt.ylabel(iris_db['feature_names'][1])
plt.grid(True)
plt.legend() # uses the 'label' argument passed to scatter()
plt.tight_layout()
# uncomment this line to show the figure, or use
# interactive mode -- plt.ion() --  in iPython
# plt.show()
plt.savefig('iris_data.png')

## A trivial example classifier.  You'll copy and modify this to
# perform a linear classification function.
```

```
def classify_rand(x):
  return rng.integers(0,2, endpoint=True)

# A function to measure the accuracy of a classifier and
# create a confusion matrix.  Keras and Scikit-learn have more sophisticated
# functions that do this, but this simple version will work for
# this assignment.
def evaluate_classifier(cls_func, x_data, labels, print_confusion_matrix=True):
  n_correct = 0
  n_total = x_data.shape[0]
  cm = np.zeros((3,3))
  for i in range(n_total):
    x = x_data[i,:]
    y = cls_func(x)
    y_true = labels[i]
    cm[y_true, y] += 1
    if y == y_true:
      n_correct += 1
    acc = n_correct / n_total
  print(f"Accuracy = {n_correct} correct / {n_total} total = {100.0*acc:3.2f}%")
  if print_confusion_matrix:
    print(f"{12*' '}Estimated Labels")
    print(f"              {0:3.0f}  {1.0:3.0f}  {2.0:3.0f}")
    print(f"{12*' '} {15*'-'}")
    print(f"True    0 |   {cm[0,0]:3.0f}  {cm[0,1]:3.0f}  {cm[0,2]:3.0f} ")
    print(f"Labels: 1 |   {cm[1,0]:3.0f}  {cm[1,1]:3.0f}  {cm[1,2]:3.0f} ")
    print(f"        2 |   {cm[2,0]:3.0f}  {cm[2,1]:3.0f}  {cm[2,2]:3.0f} ")
    print(f"{40*'-'}")
  ## done printing confusion matrix

  return acc, cm

## Now evaluate the classifier we've built.  This will evaluate the
# random classifier, which should have accuracy around 33%.
acc, cm = evaluate_classifier(classify_rand, x_data.to_numpy(),
y_labels.to_numpy())

#part e
import person
# using loop to construct new Dictionary
res = dict()
for ele in list_of_names:
      res[ele] = person

#part f
list_of_ages = np.array(list_of_ages)
list_of_heights_cm = np.array(list_of_heights_cm)

#part g
average_age = np.mean(list_of_ages)

#part h
fig=plt.figure(figsize=(6,6), dpi= 100, facecolor='w', edgecolor='k')
plt.grid(True)
plt.scatter(x=list_of_ages,y=list_of_heights_cm)
plt.ylabel("height")
plt.xlabel("ages")
plt.title("Average Ages vs heights")
plt.savefig('average_age.png')
```

```
def classify_iris(x):
  #y-Wx+b
  x=np.array(x)
  w=np.random.rand(3,4)
  b=np.random.rand(3)
  y = np.matmul(w,x)
  y = np.add(y,b)
  #print("x", x.shape)
  #print("b", b.shape)
  #print("y", y.shape)
  return np.argmax(y)

print("weights are random", evaluate_classifier(classify_iris, x_data.to_numpy(),
y_labels.to_numpy()))

def classify_iris(array):
    weights = np.array([[3, 2, 1, 0],
                       [1, 2, 3, 4],
                       [2, 1, 3, 4]])
    biases = np.array([1, 2, 3])

    y = np.dot(weights, array) + biases
    output = np.argmax(y)
    return output

print("changed weights from random", evaluate_classifier(classify_iris,
x_data.to_numpy(), y_labels.to_numpy()))
```