```
import numpy as np
# makes printing more human-friendly
np.set_printoptions(precision=3,suppress=True)


# Load the data
from google.colab import drive
drive.mount('/content/drive')
with open('/content/drive/MyDrive/Colab Notebooks/winequality-red.csv', 'r') as f:
  data = np.genfromtxt(f,delimiter=',')

X = data[1:, :-1]
y = data[1:, -1]
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# 1a
print("Class labels: ", np.unique(y))
```

```
Class labels:  [3. 4. 5. 6. 7. 8.]
```

```
# 1b
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, stratify=y)

print("training: ", X_train.shape)
print("test: ", X_test.shape)
```

```
training:  (1279, 11)
test:  (320, 11)
```

```
# 1c
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold

kf = KFold(n_splits=3)

# Count the class distributions in each partition
for train_index, val_index in kf.split(X_train):
  models = {}
  accuracies = {}
  print("new k-fold: ");
  for i in [1,3,5,10,20,30,40,50,75,100]:
    print(f"\tTraining kNN with {i} neighbors: ")
    models[i] = KNeighborsClassifier(n_neighbors=i).fit(X_train[train_index], y_train[train_index])

    y_pred = models[i].predict(X_train[val_index])
    accuracies[i] = accuracy_score(y_train[val_index], y_pred)
    print(f"\t\taccuracy : {accuracies[i]:.3f}")
```

```
        Training kNN with 5 neighbors:
                accuracy : 0.501
        Training kNN with 10 neighbors:
                accuracy : 0.492
        Training kNN with 20 neighbors:
                accuracy : 0.485
        Training kNN with 30 neighbors:
                accuracy : 0.485
        Training kNN with 40 neighbors:
                accuracy : 0.473
        Training kNN with 50 neighbors:
                accuracy : 0.480
        Training kNN with 75 neighbors:
                accuracy : 0.480
        Training kNN with 100 neighbors:
                accuracy : 0.485
new k-fold:
        Training kNN with 1 neighbors:
```

```
                Training kNN with 10 neighbors:
                        accuracy : 0.547
                Training kNN with 20 neighbors:
                        accuracy : 0.533
                Training kNN with 30 neighbors:
                        accuracy : 0.528
                Training kNN with 40 neighbors:
                        accuracy : 0.519
                Training kNN with 50 neighbors:
                        accuracy : 0.519
                Training kNN with 75 neighbors:
                        accuracy : 0.521
                Training kNN with 100 neighbors:
                        accuracy : 0.502
        new k-fold:
                Training kNN with 1 neighbors:
                        accuracy : 0.502
                Training kNN with 3 neighbors:
                        accuracy : 0.462
                Training kNN with 5 neighbors:
                        accuracy : 0.481
                Training kNN with 10 neighbors:
                        accuracy : 0.460
                Training kNN with 20 neighbors:
                        accuracy : 0.486
                Training kNN with 30 neighbors:
                        accuracy : 0.491
                Training kNN with 40 neighbors:
                        accuracy : 0.502
                Training kNN with 50 neighbors:
                        accuracy : 0.512
                Training kNN with 75 neighbors:
                        accuracy : 0.493
                Training kNN with 100 neighbors:
                        accuracy : 0.498
```

```
# 1d
from sklearn.metrics import confusion_matrix

nneigh = 1
print("optimal k neighbors: ", nneigh)

opt_model = KNeighborsClassifier(n_neighbors=nneigh).fit(X_train, y_train)
y_pred = opt_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
print(f"accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

```
    optimal k neighbors:  1
    Confusion Matrix:
    [[ 1  0  1  0  0  0]
     [ 1  0  7  3  0  0]
     [ 2  1 94 36  2  1]
     [ 0  4 31 72 18  3]
     [ 0  1  6 10 23  0]
     [ 0  0  0  3  0  0]]
    accuracy: 0.594
```

Question 2

```
# 2
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier().fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"decision tree accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

```
    decision tree accuracy: 0.619
```

```
# 3
# Load the data
with open('/content/drive/MyDrive/Colab Notebooks/Xray.csv', 'r') as f:
  data_Xray = np.genfromtxt(f,delimiter=',')

X = data_Xray[:, :-1]
y = data_Xray[:, -1]

print(X.shape, y.shape)
```

```
(800, 6) (800,)
```

```
# Partition the data
from sklearn.model_selection import train_test_split

X_train, X_tmp, y_train, y_tmp = train_test_split(X, y, train_size=0.7, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_tmp, y_tmp, train_size = 0.5, stratify=y_tmp)

print("train: ", X_train.shape)
print("val: ", X_val.shape)
print("test: ", X_test.shape)
```

```
train:  (560, 6)
val:  (120, 6)
test:  (120, 6)
```

```
# Train the random forest classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

# Tune max_depth with validation set
model = RandomForestClassifier(n_estimators=100)

accuracies = {}

for depth in range(1,11):
  print(f"Training random forest with max_depth = {depth}")
  model.set_params(max_depth=depth)
  model.fit(X_train, y_train)
  y_pred = model.predict(X_val)

  accuracy = accuracy_score(y_val, y_pred) * 100
  accuracies[depth] = accuracy
  print(f"accuracy score: {accuracy:.1f}%")
```

```
Training random forest with max_depth = 1
accuracy score: 86.7%
Training random forest with max_depth = 2
accuracy score: 94.2%
Training random forest with max_depth = 3
accuracy score: 95.0%
Training random forest with max_depth = 4
accuracy score: 94.2%
Training random forest with max_depth = 5
accuracy score: 95.0%
Training random forest with max_depth = 6
accuracy score: 95.0%
Training random forest with max_depth = 7
accuracy score: 95.0%
Training random forest with max_depth = 8
accuracy score: 95.0%
Training random forest with max_depth = 9
accuracy score: 95.0%
Training random forest with max_depth = 10
accuracy score: 95.0%
```

```
opt_max_depth = max(accuracies, key=accuracies.get)

# Train the optimal random forest classifier
print(f"Training RandomForestClassifier with n_estimators=100, max_depth={opt_max_depth}")
opt_model = RandomForestClassifier(n_estimators=100, max_depth=opt_max_depth)
opt_model.fit(X_train, y_train)

# Test and compute metrics
opt_y_pred = opt_model.predict(X_test)

cm = confusion_matrix(y_test, opt_y_pred)
print("Confusion matrix:")
print(cm)

accuracy = accuracy_score(y_test, opt_y_pred) * 100
accuracies[depth] = accuracy
print(f"accuracy score: {accuracy:.1f}%")
```

```
Training RandomForestClassifier with n_estimators=100, max_depth=10
Confusion matrix:
[[60  0]
```

```
 [ 3 57]]
accuracy score: 97.5%
```