In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

Task 1. k-nearest neighbors algorithm (KNN)

1. Load Dataset [1 point]

- Load the csv file,'final_shuffled_breast_cancer100.csv' as `df`

In [2]:
```
df = pd.read_csv("final_shuffled_breast_cancer100.csv")
```

2. Split the independent variable set and the target variable set [1 point]

- Assign `X` to the independent variable dataset
- Assign `y` to the target variable dataset

In [3]:
```
X=df.drop('target',axis=1)
y = df['target']
```

3. Split Dataset into the train & testset [1 point]

** When you use scikit-learn method to split the train & test set :

- Set `random_state` to zero.
- the ratio of train set and test set is as follows : 80% train set / 20% test set
- Assign the variable names as follow : `X_train`, `X_test`, `y_train` , `y_test`

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=0.8, random_state=0)
```

4. Load a KNN model by scikit-learn. [1 point]

- Assign KNN model as variable name `KNN`
- Set the `n_neighbors` hyperparameter as 5

In [5]:
```
KNN = KNeighborsClassifier(n_neighbors = 5)
```

5. Predict on your test set. [1 point]

In [6]:
```
KNN.fit(X_train,y_train)
y_pred = KNN.predict(X_test)
print(y_pred)
```

```
[1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1.]
```

In [7]:
```
#check conf_matrix before making cal_confusion

from sklearn.metrics import confusion_matrix, classification_report
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
[[12  2]
 [ 0  6]]
```

In [8]:
```
len(y_pred)
```

Out[8]:
```
20
```

In [9]:
```
len(y_test)
```

Out[9]:
```
20
```

6. Evaluate the prediction result of your model.

- Calculate the confusion matrix which consists of `TP`, `FP`, `TN`, `FN` (True Positive, False Positive, True Negative, False Negative)
- Calculate `accuracy rate`, `sensitivity`, `specificity`
- Fill in the blank function in order to accomplish the aforementioned tasks. (DO NOT USE PACKAGES IN THIS TASK)

6.1 Calculate the confusion matrix which consists of `TP`, `FP`, `TN`, `FN` (True Positive, False Positive, True Negative, False Negative) [5 points]

- Fill in the blank function in order to accomplish the aforementioned tasks. (DO NOT USE PACKAGES IN THIS TASK)

In [10]:
```
def cal_confusion(y_true, y_pred):
    TP =0
    FP =0
    TN = 0
    FN = 0
    for i in range(len(y_true)):
        if y_true[i]==1 and y_pred[i]==1:
            TP = TP + 1
        if y_pred[i]==1 and y_true[i]!=y_pred[i]:
            FP = FP + 1
        if y_true[i]==0 and y_pred[i]==0:
            TN = TN + 1
        if y_pred[i]==0 and y_true[i]!=y_pred[i]:
            FN = FN + 1

    return TP, FP, TN, FN
```

In [11]:
```
y_test = y_test.values.ravel()
cal_confusion(y_test, y_pred)
```

Out[11]:
```
(6, 2, 12, 0)
```

6.2 Calculate `accuracy rate` by filling in the blank of `cal_accuracy` function. [5 points]

- Fill in the blank function in order to accomplish the aforementioned tasks. (DO NOT USE PACKAGES IN THIS TASK)

In [12]:
```
def cal_accuracy(y_true, y_pred):

    TP =0
    FP =0
    TN = 0
    FN = 0
    for i in range(len(y_true)):
        if y_true[i]==1 and y_pred[i]==1:
            TP = TP + 1
        if y_pred[i]==1 and y_true[i]!=y_pred[i]:
            FP = FP + 1
        if y_true[i]==0 and y_pred[i]==0:
            TN = TN + 1
        if y_pred[i]==0 and y_true[i]!=y_pred[i]:
            FN = FN + 1

    accuracy = 0
    accuracy = (TP+TN)/ (TP+FP+TN+FN)

    return accuracy
```

In [13]:
```
cal_accuracy(y_test, y_pred)
```

Out[13]:
```
0.9
```

6.3 Calculate `sensitivity` by filling in the blank of `cal_sensitivity` function. [5 points]

- Fill in the blank function in order to accomplish the aforementioned tasks. (DO NOT USE PACKAGES IN THIS TASK)
- When it comes to the `cal_sensitivity` function, we didn't specify the exact input variables, but just include all of `TP`, `FP`, `TN`, `FN`. You have to choose two of them and use them as the input variables of the `cal_sensitivity` function.

In [14]:
```
def cal_sensitivity(TP,FP,TN,FN):


    sensitivity = 0
    sensitivity = (TP)/ (TP+FN)

    return sensitivity
```

In [15]:
```
cal_sensitivity(6,2,12,0)
```

Out[15]:
```
1.0
```

6.4 Calculate `specificity` by filling in the blank of `cal_specificity` function. [5 points]

- Fill in the blank function in order to accomplish the aforementioned tasks. (DO NOT USE PACKAGES IN THIS TASK)
- When it comes to the `cal_specificity` function, we didn't specify the exact input variables, but just include all of `TP`, `FP`, `TN`, `FN`. You have to choose two of them and use them as the input variables of the `cal_specificity` function.

In [16]:
```
def cal_specificity(TP,FP,TN,FN):

    specificity=0
    specificity= (TN)/(TN+FP)


    return specificity
```

In [17]:
```
cal_specificity(6,2,12,0)
```

Out[17]:
```
0.8571428571428571
```

6.5 Print all of the results [1 point]

- print all of the results (confusion matrix, accuracy , sensitivity, specificity)
- fill in the below `print` function by your own results

In [18]:
```
TP = cal_confusion(y_test,y_pred)[0]
FP = cal_confusion(y_test,y_pred)[1]
TN = cal_confusion(y_test,y_pred)[2]
FN = cal_confusion(y_test,y_pred)[3]


print('confusion matrix: ', cal_confusion(y_test, y_pred)   )
print('accuracy: ',    cal_accuracy(y_test, y_pred)         )
print('sensitivity: ', cal_sensitivity(TP, FP, TN, FN)      )
print('specificity: ', cal_specificity(TP, FP, TN, FN)      )
```

```
confusion matrix:  (6, 2, 12, 0)
accuracy:  0.9
sensitivity:  1.0
specificity:  0.8571428571428571
```

6.6 Plot accuracy results as you change the K values. [3 points]

- Plot the accuracy results from changing the number of K = (1,2,3,4,5,10)

In [19]:
```
k_range = [1,2,3,4,5,10]
accuracy = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy.append(cal_accuracy(y_test, y_pred))

print(accuracy)
```

```
[0.95, 1.0, 0.9, 0.9, 0.9, 0.9]
```

In [20]:
```
plt.plot(k_range, accuracy)
plt.xlabel('Value of K for KNN')
plt.ylabel('Accuracy')
val=[1,2,3,4,5,10]
plt.xticks(val)
plt.show()

# at this case k = 2 is highest accuracy!
```