Data description

ageage in years sex(1 = male; 0 = female) cpchest pain type trestbpsresting blood pressure (in mm Hg on admission to the hospital) cholserum cholestoral in mg/dl fbs(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false) restecgresting electrocardiographic results thalachmaximum heart rate achieved exangexercise induced angina (1 = yes; 0 = no) oldpeakST depression induced by exercise relative to rest slopethe slope of the peak exercise ST segment canumber of major vessels (0-3) colored by flourosopy thal 3 = normal; 6 = fixed defect; 7 = reversable defect target1 or 0

In [1]:
```
import numpy as np
import pandas as pd
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import math
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
import seaborn as sns
```

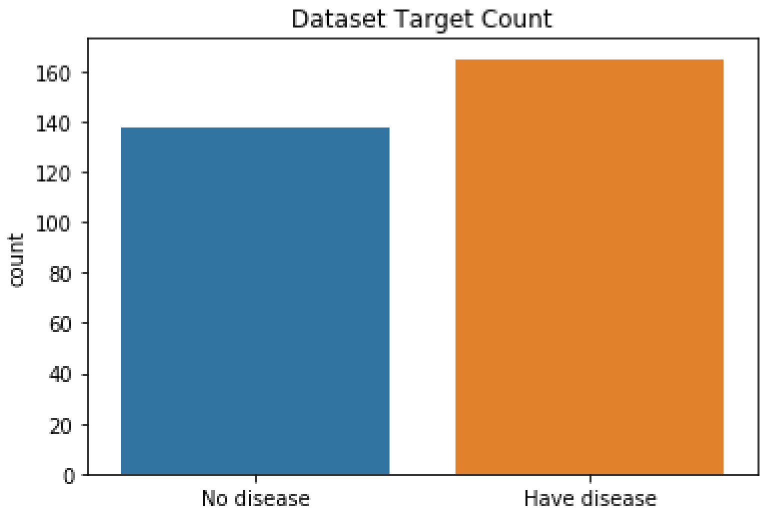
In [2]:
```
# read data from dataset
df = pd.read_csv("heart.csv")
df.head()
```

Out[2]:

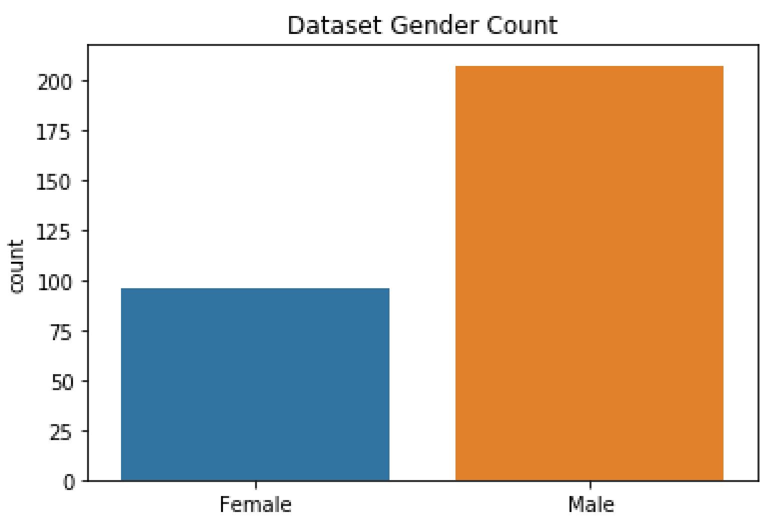
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Data Exploration

In [3]:
```
sns.countplot(x = df.target ,data = df)
plt.title('Dataset Target Count')
positions = (0, 1)
labels = ('No disease', 'Have disease')
plt.xticks(positions, labels)
plt.xlabel('')
plt.show()
```

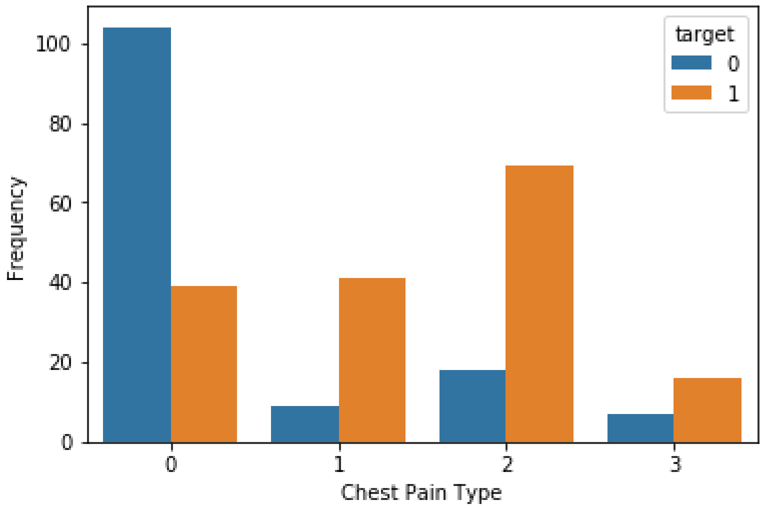


In [4]:
```
sns.countplot(x = df.sex ,data = df)
plt.title('Dataset Gender Count')
positions = (0, 1)
labels = ('Female', 'Male')
plt.xticks(positions, labels)
plt.xlabel('')
plt.show()
```



In [5]:
```
ct = pd.crosstab(df.cp, df.target)
stacked = ct.stack().reset_index().rename(columns={0:'value'})
sns.barplot(x=stacked.cp, y=stacked.value, hue=stacked.target)
plt.xlabel('Chest Pain Type')
plt.ylabel('Frequency')
```

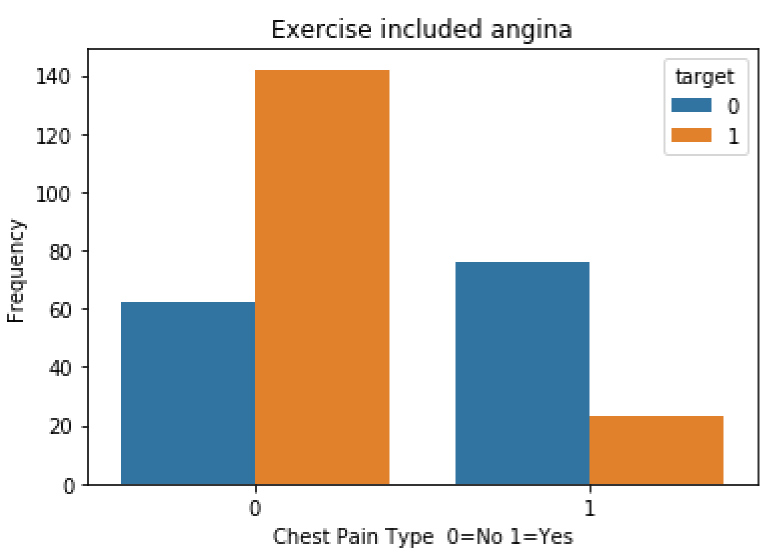
Out[5]:
```
Text(0,0.5,'Frequency')
```



In [6]:
```
df.columns
```

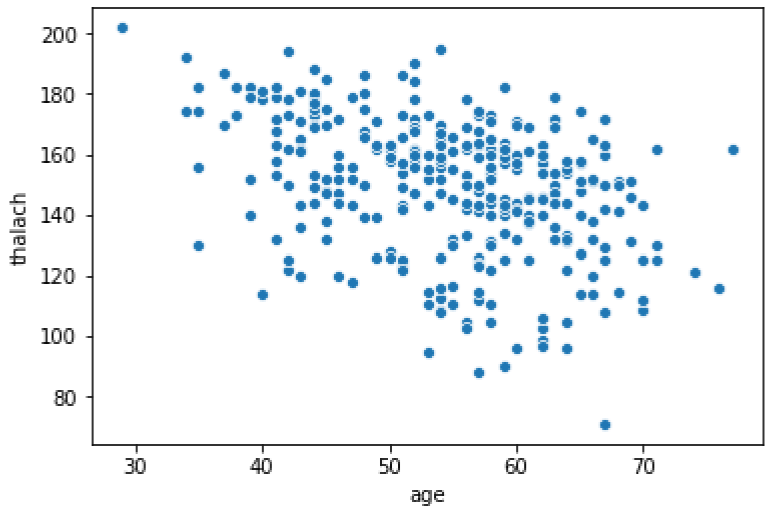
Out[6]:
```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [7]:
```
ct = pd.crosstab(df.exang, df.target)
stacked = ct.stack().reset_index().rename(columns={0:'value'})
sns.barplot(x=stacked.exang, y=stacked.value, hue=stacked.target)
plt.title('Exercise included angina')
plt.ylabel('Frequency')
plt.xlabel('Chest Pain Type  0=No 1=Yes')
plt.show()
```



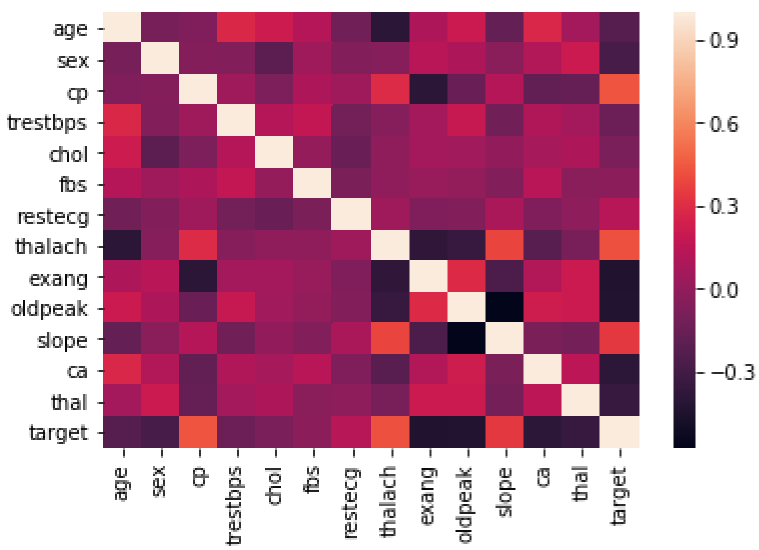
In [8]:
```
sns.scatterplot('age', 'thalach', data= df)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1e7f8c26d30>



In [9]:
```
sns.heatmap(df.corr())
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1e7f8cd1b38>



Data preparation

In [10]:
```
data = df.iloc[:,:-1]
```

In [11]:
```
y = df['target']
```

Dummy Variables using pd.get dummies

In [12]:
```
#Some column are catagorical that are not binary-> one hot
#'cp','exang','slope','ca','thal'
```

In [13]:
```
#Use dummy variable on all of these columns
#using drop first can drop the _0 column because it is a baseline column
df = pd.get_dummies(data, columns=['cp','exang','slope','ca','thal'], drop_fir
st= True)
print(df.columns)
print(df.shape)
```

```
Index(['age', 'sex', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'oldpeak', 'cp_1', 'cp_2', 'cp_3', 'exang_1', 'slope_1', 'slope_2',
       'ca_1', 'ca_2', 'ca_3', 'ca_4', 'thal_1', 'thal_2', 'thal_3'],
      dtype='object')
(303, 21)
```

In [14]:
```
df.head()
```

Out[14]:

	age	sex	trestbps	chol	fbs	restecg	thalach	oldpeak	cp_1	cp_2	...	exang_1	slope_1
0	63	1	145	233	1	0	150	2.3	0	0	...	0	0
1	37	1	130	250	0	1	187	3.5	0	1	...	0	0
2	41	0	130	204	0	0	172	1.4	1	0	...	0	0
3	56	1	120	236	0	1	178	0.8	1	0	...	0	0
4	57	0	120	354	0	1	163	0.6	0	0	...	1	0

5 rows × 21 columns

Normalization

In [15]:
```
normalized_data = (df - np.min(df)) / (np.max(df) - np.min(df))
```

Prepare data for training

In [16]:
```
train_X, test_X, train_y, test_y = train_test_split(normalized_data, y, test_s
ize = .25 , random_state=0) #split the data
```

In [17]:
```
#Convert into array from dataframe
trainX = train_X.values
testX = test_X.values
trainy = train_y.values
testy= test_y.values
print('trainX: %d, testX: %d, trainy: %d, testy: %d' %(len(trainX), len(testX
), len(trainy), len(testy)))
```

```
trainX: 227, testX: 76, trainy: 227, testy: 76
```

In [18]:
```
trainX[0]
```

Out[18]:
```
array([0.60416667, 1.        , 0.35849057, 0.22374429, 0.        ,
       0.        , 0.77862595, 0.51612903, 0.        , 1.        ,
       0.        , 0.        , 0.        , 1.        , 0.        ,
       1.        , 0.        , 0.        , 0.        , 0.        ,
       1.        ])
```

Implementing KNN Algorithm from scratch

In [19]:
```
#Set up the euclidean distance function
def eu_dis(x1, x2):
    #initial total distance is 0
    total = 0
    #loop throught the length of the list
    for i in range (len(x1)):
        #use the euclidean distance formula and return
        total += (x1[i]-x2[i])**2
    return math.sqrt(total)
```

In [20]:
```
#Set up the KNN function
def getNeighborLabel (trainX, train_y, testX, k):

    #A list to gather all the nearest neighbors for testing
    all_vote=[]
    #apply the algorithm for every single test instance (points) within the li
st
    for case in testX:

        #gather the distance for each traning point and the test set in a list
format
        distance = []
        for i in range(len(trainX)):
            dist = eu_dis(case, trainX[i])
            distance.append(dist)

        #argsort sort the distance of the list from ascending order. [:k] pick
ed out top k number of index
        #into a list called ind. Ind store the indicies of the rows of trainin
g set.
        #Convert the ind list from nparray to list for easier munipulition
        ind = np.argsort(distance)[:k]
        ind = ind.tolist()

        #Declare an empty list called toplabels to store the most common k num
ber of labels.
        #A for loop is the append the label data from the indicies stored in t
he ind list.
        topLabels =[]
        for i in ind:
            topLabels.append(train_y[i])

        #count_freq is the outcome of using Counter return of the most common
 tuples
        count_freq = Counter(topLabels).most_common()

        #The most common tuples, aka top_vote in this case, is the nearest nei
ghbor for the test instance
        top_vote = count_freq[0][0]

        #Store all the nearest neighbor in the all_vote list and return it
        all_vote.append(top_vote)

    return all_vote
```

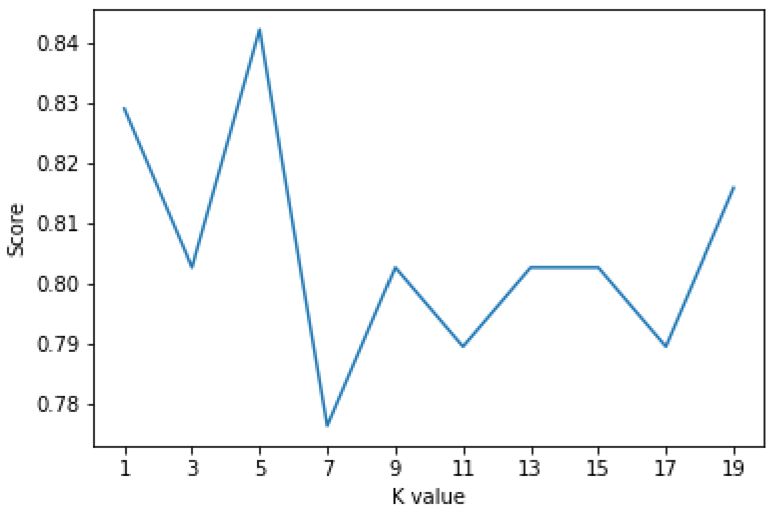
In [21]:
```
k_list = np.arange(1,21,2)
k_result = []
for k in k_list:
    k_result.append(getNeighborLabel(trainX, trainy, testX, k))

my_knn_result =[]
for result in k_result:
    my_knn_result.append(accuracy_score(result, test_y))
```

In [22]:
```
#The result of my knn model
my_knn_result
```

Out[22]:
```
[0.8289473684210527,
 0.8026315789473685,
 0.8421052631578947,
 0.7763157894736842,
 0.8026315789473685,
 0.7894736842105263,
 0.8026315789473685,
 0.8026315789473685,
 0.7894736842105263,
 0.8157894736842105]
```

In [23]:
```
plt.plot(k_list, my_knn_result)
plt.xticks(np.arange(1,21,2))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
```



Verify my KNN function with sklearn

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier
sklearn_knn_result = []
for k in k_list:
    model = KNeighborsClassifier(n_neighbors = k)
    model.fit(trainX, trainy)
    prediction = model.predict(testX)
    sklearn_knn_result.append(accuracy_score(prediction, test_y)) #The same sc
ore as my written algorithm
```

In [25]:
```
equal = max(my_knn_result) == max(sklearn_knn_result)
```

In [26]:
```
# k=3 have the highest accuracy in both knn model, the accuracy is a exact sam
e as well.
print('The k=5 has the highest accuracy of %f'% max(my_knn_result) )
print('Accuracy score of k=5 from the 2 lists are equal is', equal )
```

```
The k=5 has the highest accuracy of 0.842105
Accuracy score of k=5 from the 2 lists are equal is True
```

CV with Grid Search

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
clf =KNeighborsClassifier()
```

In [28]:
```
param_grid = dict(n_neighbors=k_list)
print(param_grid)
```

```
{'n_neighbors': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])}
```

In [29]:
```
gs_model = GridSearchCV(clf, param_grid, cv = 10)
gs_model.fit(trainX, trainy)
```

```
C:\Users\chiu\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:
841: DeprecationWarning: The default of the `iid` parameter will change from
True to False in version 0.22 and will be removed in 0.24. This will change n
umeric results when test-set sizes are unequal.
  DeprecationWarning)
```

Out[29]:
```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17,
19])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [30]:
```
gs_model.best_params_
```

Out[30]:
```
{'n_neighbors': 7}
```

In [31]:
```
#Run k=17 with my model
print('Accuracy with k=7 is', accuracy_score(getNeighborLabel(trainX, trainy,
testX, 7), testy))
```

```
Accuracy with k=7 is 0.7763157894736842
```

SVM

In [32]:
```
from sklearn.svm import SVC
```

In [33]:
```
svm = SVC(random_state = 0)
svm.fit(trainX, trainy)
score = svm.score(testX, testy)
score
```

```
C:\Users\chiu\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarn
ing: The default value of gamma will change from 'auto' to 'scale' in version
0.22 to account better for unscaled features. Set gamma explicitly to 'auto'
or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[33]: 0.8157894736842105

Comapre SVM with KNN

In [34]:
```
score
```

Out[34]: 0.8157894736842105

In [35]:
```
svm_score = []
for i in k_list:
    svm_score.append(score)
```

In [36]:
```
plt.plot(k_list, my_knn_result)
plt.plot(k_list, svm_score)
plt.xticks(np.arange(1,21,2))
plt.xlabel("K value")
plt.ylabel("Score")
plt.plot(y = score)
plt.show()
```