Heart Disease Prediction

We have a data which classified if patients have heart disease or not according to features in it. We will try to use this data to create a model which tries predict if a patient has this disease or not. We will use many (classification) algorithms.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

Read Data

In [2]:

```
# We are reading our data
df = pd.read_csv("D:/class/M.Tech 2nd/DL/lab/projects/healthcare/heart.csv")
```

In [3]:

```
# First 5 rows of our data
df.head()
```

Out[3]:

	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 contains;

- age - age in years
- sex - (1 = male; 0 = female)
- cp - chest pain type
- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
- fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg - resting electrocardiographic results
- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
- oldpeak - ST depression induced by exercise relative to rest
- slope - the slope of the peak exercise ST segment

- ca - number of major vessels (0-3) colored by flourosopy
- thal - 3 = normal; 6 = fixed defect; 7 = reversable defect
- target - have disease or not (1=yes, 0=no)

Data Exploration

In [4]:

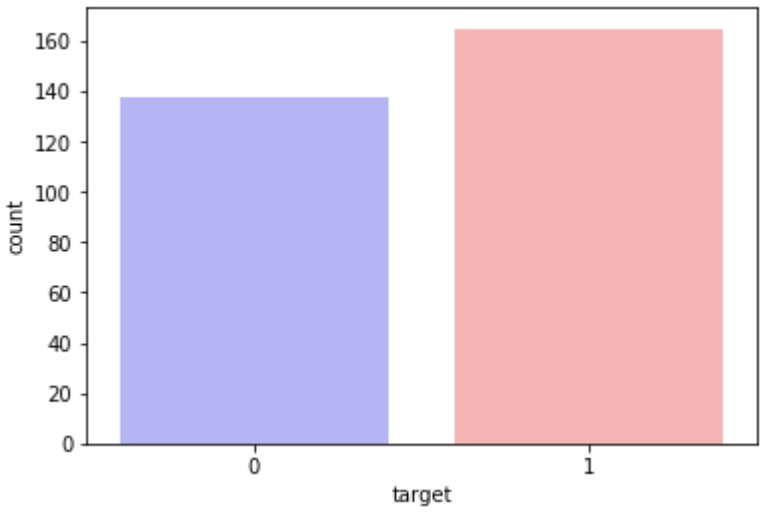
```
df.target.value_counts()
```

Out[4]:

```
1    165
0    138
Name: target, dtype: int64
```

In [5]:

```
sns.countplot(x="target", data=df, palette="bwr")
plt.show()
```



Creating Dummy Variables

Since 'cp', 'thal' and 'slope' are categorical variables we'll turn them into dummy variables.

In [6]:

```
a = pd.get_dummies(df['cp'], prefix = "cp")
b = pd.get_dummies(df['thal'], prefix = "thal")
c = pd.get_dummies(df['slope'], prefix = "slope")
```

In [7]:

```
frames = [df, a, b, c]
df = pd.concat(frames, axis = 1)
df.head()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	...	cp_1	cp_2	cp_3
0	63	1	3	145	233	1	0	150	0	2.3	...	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	...	0	1	0
2	41	0	1	130	204	0	0	172	0	1.4	...	1	0	0
3	56	1	1	120	236	0	1	178	0	0.8	...	1	0	0
4	57	0	0	120	354	0	1	163	1	0.6	...	0	0	0

5 rows × 25 columns

In [8]:

```
df = df.drop(columns = ['cp', 'thal', 'slope'])
df.head()
```

Out[8]:

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	...	cp_1	cp_2	cp_3
0	63	1	145	233	1	0	150	0	2.3	0	...	0	0	1
1	37	1	130	250	0	1	187	0	3.5	0	...	0	1	0
2	41	0	130	204	0	0	172	0	1.4	0	...	1	0	0
3	56	1	120	236	0	1	178	0	0.8	0	...	1	0	0
4	57	0	120	354	0	1	163	1	0.6	0	...	0	0	0

5 rows × 22 columns

Creating Models

We can use sklearn library or we can write functions ourselves. Let's them both. Firstly we will write our functions after that we'll use sklearn library to calculate score.

In [9]:

```
y = df.target.values
x_data = df.drop(['target'], axis = 1)
```

In [10]:

```
# Normalize
x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values
```

We will split our data. 80% of our data will be train data and 20% of it will be test data.

In [11]:

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=0)
```

Let's say weight = 0.01 and bias = 0.0

Sklearn Logistic Regression

In [32]:

```
accuracies = {}

lr = LogisticRegression()
lr.fit(x_train.T,y_train.T)
acc = lr.score(x_test.T,y_test.T)*100

accuracies['Logistic Regression'] = acc
print("Test Accuracy {:.2f}%".format(acc))
```

```
Test Accuracy 86.89%
```

1. **Our model works with 86.89% accuracy.**

Sklearn KNN

In [33]:

```
# KNN Model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 2)  # n_neighbors means k
knn.fit(x_train.T, y_train.T)
prediction = knn.predict(x_test.T)

print("{} NN Score: {:.2f}%".format(2, knn.score(x_test.T, y_test.T)*100))
```

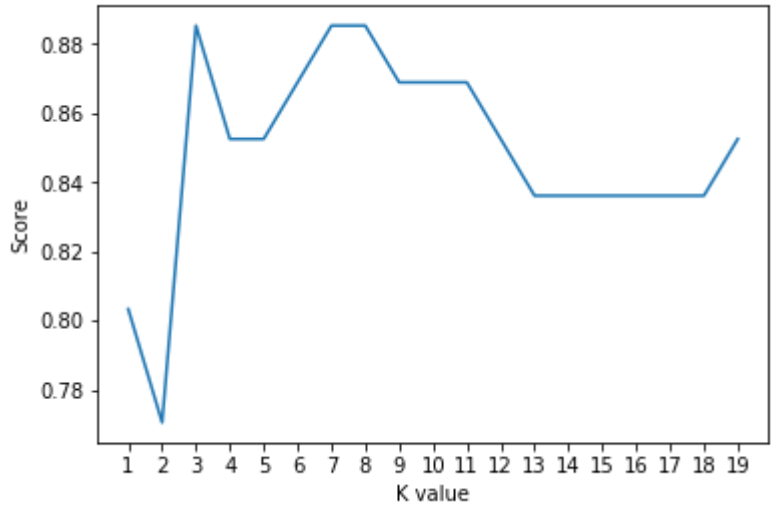
```
2 NN Score: 77.05%
```

In [34]:

```
# try ro find best k value
scoreList = []
for i in range(1,20):
    knn2 = KNeighborsClassifier(n_neighbors = i)  # n_neighbors means k
    knn2.fit(x_train.T, y_train.T)
    scoreList.append(knn2.score(x_test.T, y_test.T))

plt.plot(range(1,20), scoreList)
plt.xticks(np.arange(1,20,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()

acc = max(scoreList)*100
accuracies['KNN'] = acc
print("Maximum KNN Score is {:.2f}%".format(acc))
```




```
Maximum KNN Score is 88.52%
```

As you can see above if we define k as 3-7-8 we will reach maximum score.

KNN Model's Accuracy is 88.52%

Sklearn SVM

In [35]:

```
from sklearn.svm import SVC
```

In [36]:

```
svm = SVC(random_state = 1)
svm.fit(x_train.T, y_train.T)

acc = svm.score(x_test.T,y_test.T)*100
accuracies['SVM'] = acc
print("Test Accuracy of SVM Algorithm: {:.2f}%".format(acc))
```

```
Test Accuracy of SVM Algorithm: 86.89%
```

Test Accuracy of SVM Algorithm is 86.89%

Random Forest Classification

In [37]:

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 1)
rf.fit(x_train.T, y_train.T)

acc = rf.score(x_test.T,y_test.T)*100
accuracies['Random Forest'] = acc
print("Random Forest Algorithm Accuracy Score : {:.2f}%".format(acc))
```

```
Random Forest Algorithm Accuracy Score : 88.52%
```

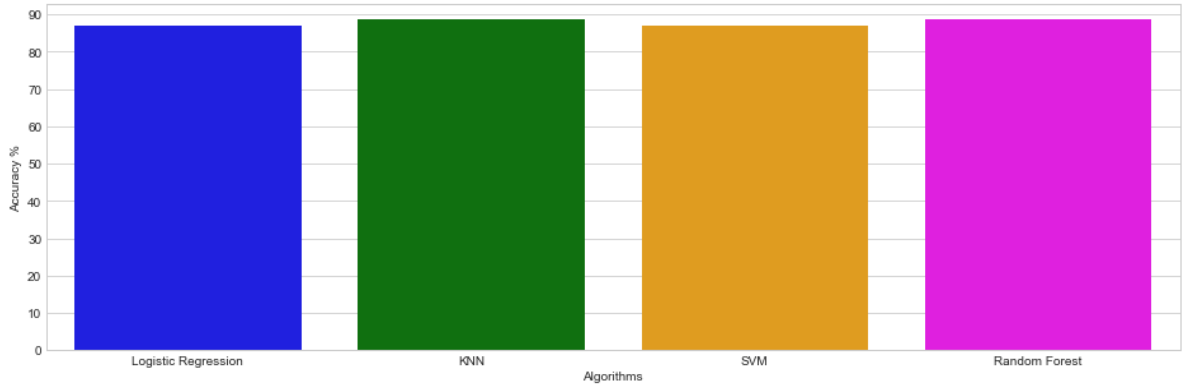
Test Accuracy of Random Forest: 88.52%

Comparing Models

In [40]:

```
colors = ["Blue", "green", "orange", "magenta"]

sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,100,10))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=list(accuracies.keys()), y=list(accuracies.values()), palette=colors)
plt.show()
```



Our models work fine but best of them are KNN and Random Forest with 88.52% of accuracy. Let's look their confusion matrixes.