Heart Disease Prediction

In this machine learning project, I have collected the dataset from Kaggle (https://www.kaggle.com/ronitf/heart-disease-uci) and I will be using Machine Learning to predict whether any person is suffering from heart disease

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Here we will be experimenting with 3 algorithms

1. KNeighborsClassifier
2. DecisionTreeClassifier
3. RandomForestClassifier

In [2]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [3]:
```
df = pd.read_csv('dataset.csv')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

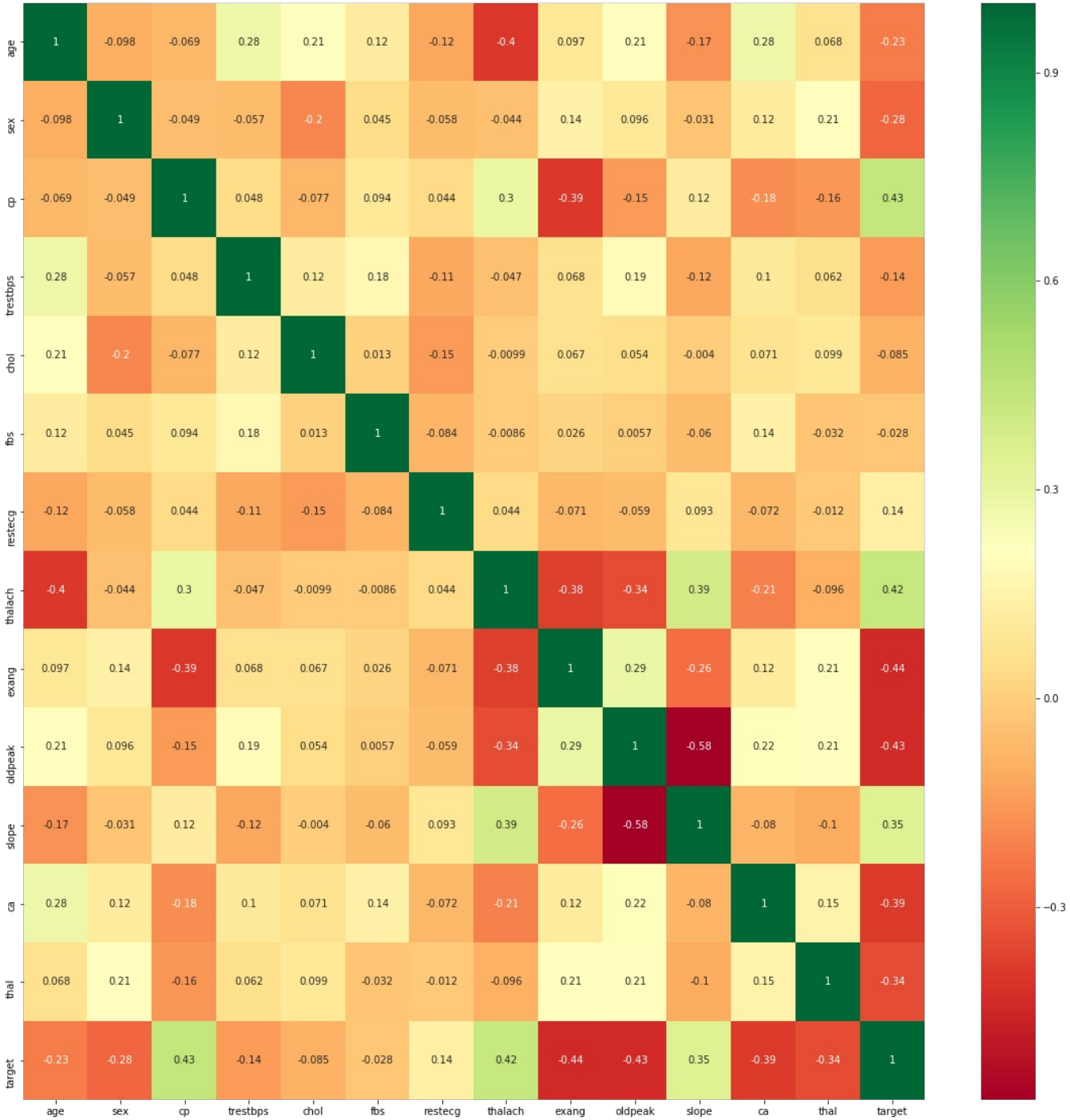
In [6]:
```
df.describe()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000

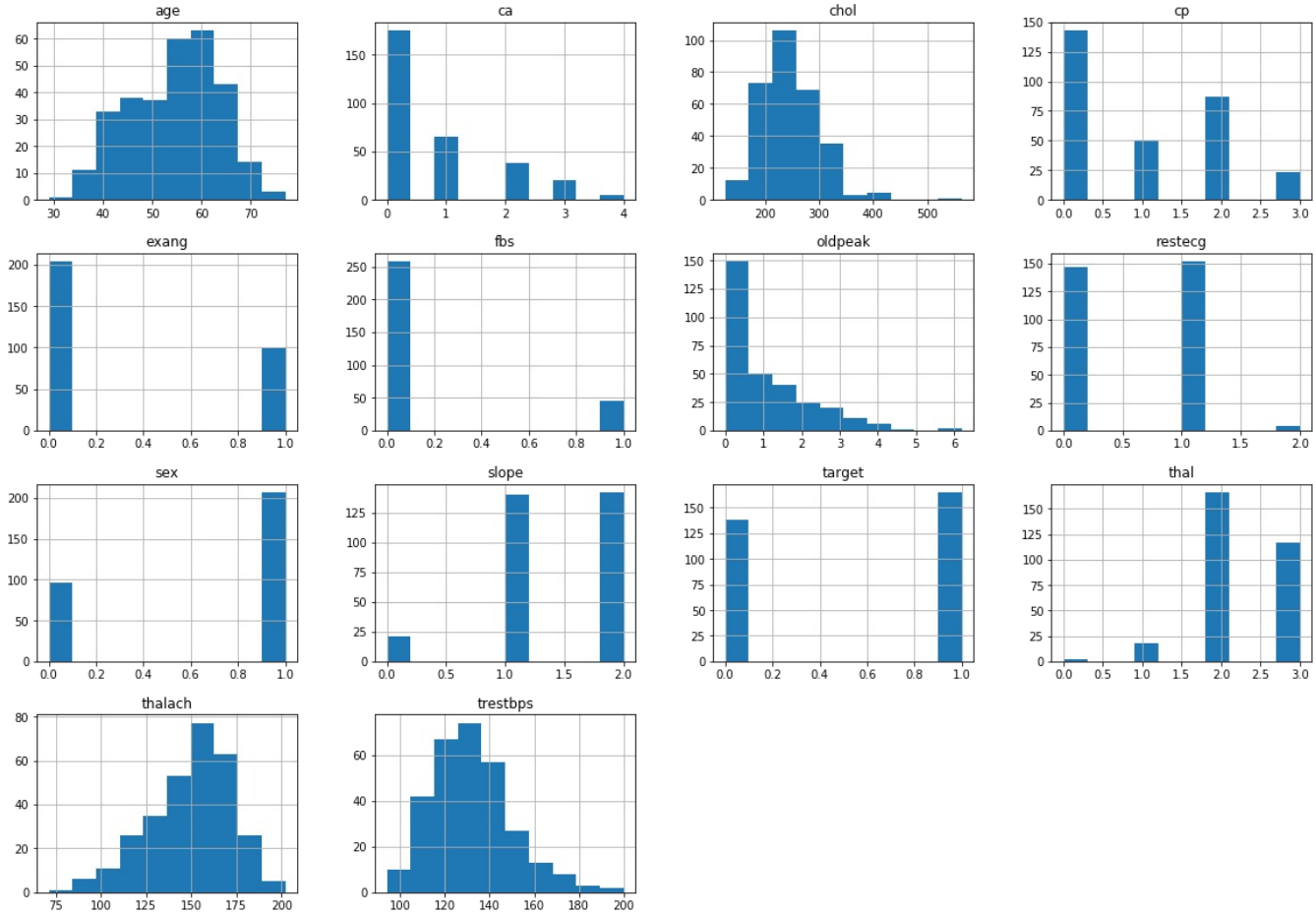
Feature Selection

In [11]:
```
import seaborn as sns
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



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

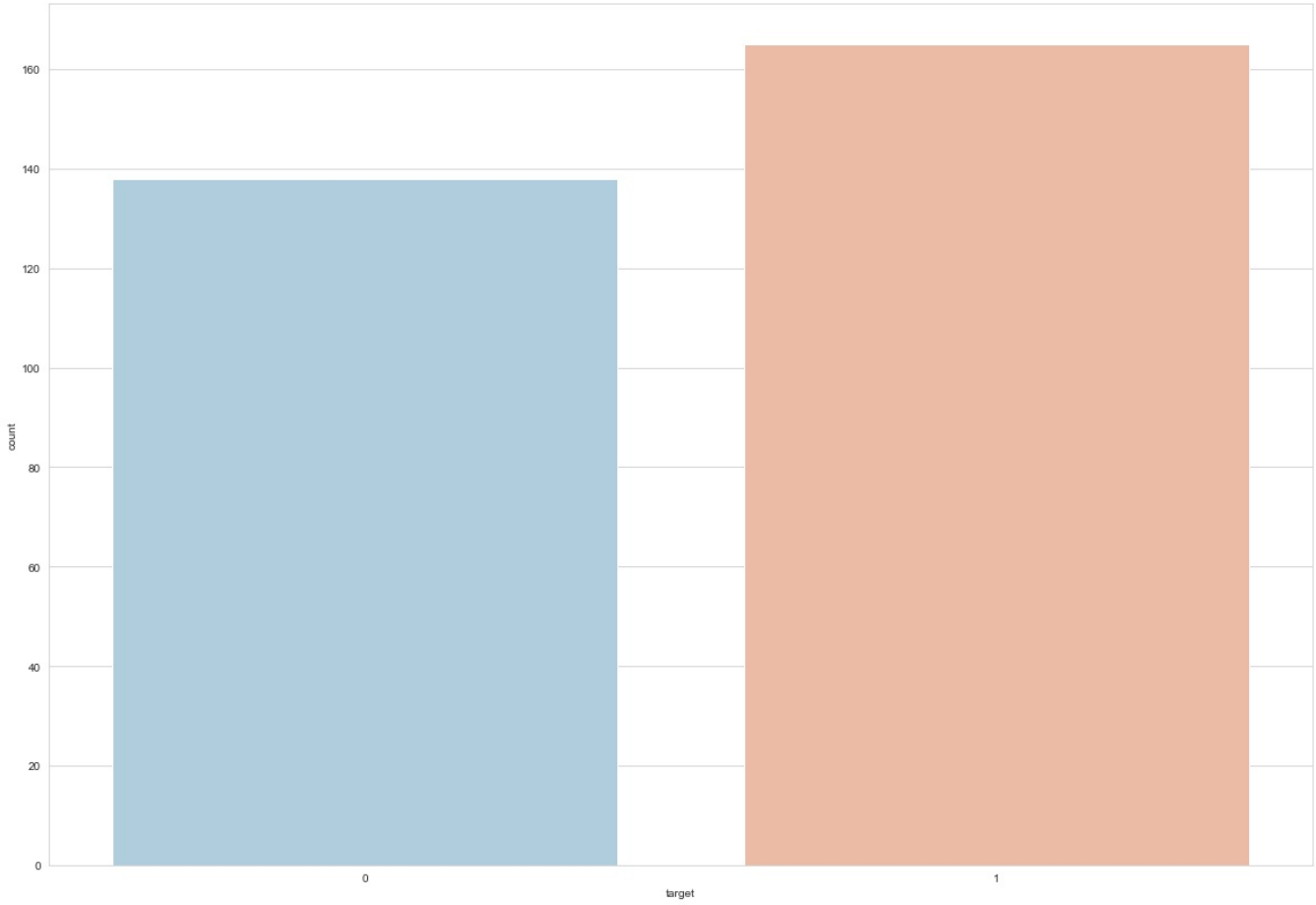
Out[14]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000017C18BE4FD0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C18C12EF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C18BDB2E8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C19D50550>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000017C1959F7B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C190A4A20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C19427C88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C1949BF28>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000017C1949BF60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C19224400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C1A03F668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C18F138D0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000017C19505B38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C19577DA0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C1A28A048>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000017C1A3B72B0>]],
      dtype=object)
```



It's always a good practice to work with a dataset where the target classes are of approximately equal size. Thus, let's check for the same.

In [16]:
```
sns.set_style('whitegrid')
sns.countplot(x='target',data=df,palette='RdBu_r')
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17c19761208>
```



Data Processing

After exploring the dataset, I observed that I need to convert some categorical variables into dummy variables and scale all the values before training the Machine Learning models. First, I'll use the `get_dummies` method to create dummy columns for categorical variables.

In [17]:
```
dataset = pd.get_dummies(df, columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'])
```

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
standardScaler = StandardScaler()
columns_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[columns_to_scale] = standardScaler.fit_transform(dataset[columns_to_scale])
```

```
C:\Users\krish.naik\AppData\Local\Continuum\anaconda3\envs\myenv\lib\site-packages\sklearn\preprocessing\data.py:625: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Users\krish.naik\AppData\Local\Continuum\anaconda3\envs\myenv\lib\site-packages\sklearn\base.py:462: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [19]:
```
dataset.head()
```

Out[19]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	0	1	0	0	...	0	1	0	0	0	0	0
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	0	1	0	0	...	0	1	0	0	0	0	0
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	1	0	0	1	...	1	1	0	0	0	0	0
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	0	1	0	1	...	1	1	0	0	0	0	0
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	1	0	1	0	...	1	1	0	0	0	0	0

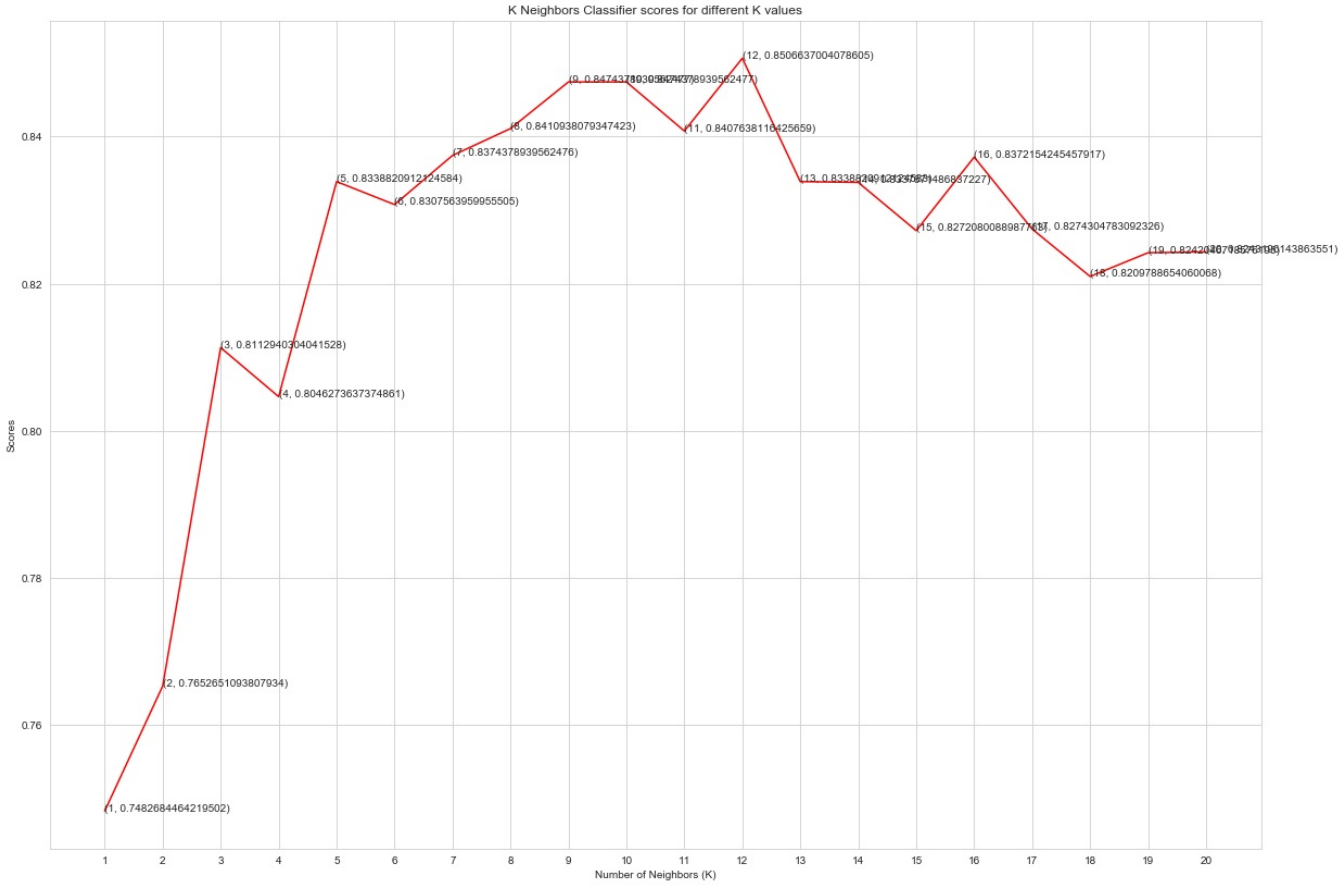
5 rows × 31 columns

In [24]:
```
y = dataset['target']
X = dataset.drop(['target'], axis = 1)
```

In [25]:
```
from sklearn.model_selection import cross_val_score
knn_scores = []
for k in range(1,21):
    knn_classifier = KNeighborsClassifier(n_neighbors = k)
    score=cross_val_score(knn_classifier,X,y,cv=10)
    knn_scores.append(score.mean())
```

In [26]:
```
plt.plot([k for k in range(1, 21)], knn_scores, color = 'red')
for i in range(1,21):
    plt.text(i, knn_scores[i-1], (i, knn_scores[i-1]))
plt.xticks([i for i in range(1, 21)])
plt.xlabel('Number of Neighbors (K)')
plt.ylabel('Scores')
plt.title('K Neighbors Classifier scores for different K values')
```

Out[26]:
```
Text(0.5, 1.0, 'K Neighbors Classifier scores for different K values')
```



In [36]:
```
knn_classifier = KNeighborsClassifier(n_neighbors = 12)
score=cross_val_score(knn_classifier,X,y,cv=10)
```

In [38]:
```
score.mean()
```

Out[38]:
```
0.8506637004078605
```

Random Forest Classifier

In [27]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [30]:
```
randomforest_classifier= RandomForestClassifier(n_estimators=10)
```

```
score=cross_val_score(randomforest_classifier,X,y,cv=10)
```

In [31]:
```
score.mean()
```

Out[31]:
```
0.8199888765294772
```

In [ ]:
```

```

In [ ]:
```

```