Heart Disease Dataset https://archive.ics.uci.edu/ml/datasets/Heart+Disease Creators of the Dataset:

1. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
2. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
3. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
4. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

Donor: David W. Aha (aha '@' ics.uci.edu) (714) 856-8779

Information on Dataset:

- 303 instances of data
- 14 columns out of 76 columns

In [1]:
```
# Reading the data
import pandas as pd
data = pd.read_csv("heart.csv")
```

## Information on Dataset

In [2]:
```
data.info
```

Out[2]:
```
<bound method DataFrame.info of      age  sex  cp  trestbps  chol  fbs  ...  exang  oldpeak  slope  ca  thal  target
0     63    1   3       145   233    1  ...      0      2.3      0   0     1       1
1     37    1   2       130   250    0  ...      0      3.5      0   0     2       1
2     41    0   1       130   204    0  ...      0      1.4      2   0     2       1
3     56    1   1       120   236    0  ...      0      0.8      2   0     2       1
4     57    0   0       120   354    0  ...      1      0.6      2   0     2       1
..   ...  ...  ..       ...   ...  ...  ...    ...      ...    ...  ..   ...     ...
298   57    0   0       140   241    0  ...      1      0.2      1   0     3       0
299   45    1   3       110   264    0  ...      0      1.2      1   0     3       0
300   68    1   0       144   193    1  ...      0      3.4      1   2     3       0
301   57    1   0       130   131    0  ...      1      1.2      1   1     3       0
302   57    0   1       130   236    0  ...      0      0.0      1   1     2       0

[303 rows x 14 columns]>
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [4]:
```
data.describe()
```

Out[4]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.000000 | 303.00 |
| **mean** | 54.366337 | 0.683168 | 0.966997 | 131.623762 | 246.264026 | 0.148515 | 0.528053 | 149.646865 | 0.326733 | 1.039604 | 1.399340 | 0.729373 | 2.31 |
| **std** | 9.082101 | 0.466011 | 1.032052 | 17.538143 | 51.830751 | 0.356198 | 0.525860 | 22.905161 | 0.469794 | 1.161075 | 0.616226 | 1.022606 | 0.61 |
| **min** | 29.000000 | 0.000000 | 0.000000 | 94.000000 | 126.000000 | 0.000000 | 0.000000 | 71.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00 |
| **25%** | 47.500000 | 0.000000 | 0.000000 | 120.000000 | 211.000000 | 0.000000 | 0.000000 | 133.500000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.00 |
| **50%** | 55.000000 | 1.000000 | 1.000000 | 130.000000 | 240.000000 | 0.000000 | 1.000000 | 153.000000 | 0.000000 | 0.800000 | 1.000000 | 0.000000 | 2.00 |
| **75%** | 61.000000 | 1.000000 | 2.000000 | 140.000000 | 274.500000 | 0.000000 | 1.000000 | 166.000000 | 1.000000 | 1.600000 | 2.000000 | 1.000000 | 3.00 |
| **max** | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 564.000000 | 1.000000 | 2.000000 | 202.000000 | 1.000000 | 6.200000 | 2.000000 | 4.000000 | 3.00 |

In [5]:
```
data.dtypes.value_counts()
```

Out[5]:
```
int64      13
float64     1
dtype: int64
```

Display a few records in dataset

In [6]:
```
data.head(10)
```

Out[6]:

| | 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 |
| **5** | 57 | 1 | 0 | 140 | 192 | 0 | 1 | 148 | 0 | 0.4 | 1 | 0 | 1 | 1 |
| **6** | 56 | 0 | 1 | 140 | 294 | 0 | 0 | 153 | 0 | 1.3 | 1 | 0 | 2 | 1 |
| **7** | 44 | 1 | 1 | 120 | 263 | 0 | 1 | 173 | 0 | 0.0 | 2 | 0 | 3 | 1 |
| **8** | 52 | 1 | 2 | 172 | 199 | 1 | 1 | 162 | 0 | 0.5 | 2 | 0 | 3 | 1 |
| **9** | 57 | 1 | 2 | 150 | 168 | 0 | 1 | 174 | 0 | 1.6 | 2 | 0 | 2 | 1 |

In [7]:
```
data.tail(10)
```

Out[7]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **293** | 67 | 1 | 2 | 152 | 212 | 0 | 0 | 150 | 0 | 0.8 | 1 | 0 | 3 | 0 |
| **294** | 44 | 1 | 0 | 120 | 169 | 0 | 1 | 144 | 1 | 2.8 | 0 | 0 | 1 | 0 |
| **295** | 63 | 1 | 0 | 140 | 187 | 0 | 0 | 144 | 1 | 4.0 | 2 | 2 | 3 | 0 |
| **296** | 63 | 0 | 0 | 124 | 197 | 0 | 1 | 136 | 1 | 0.0 | 1 | 0 | 2 | 0 |
| **297** | 59 | 1 | 0 | 164 | 176 | 1 | 0 | 90 | 0 | 1.0 | 1 | 2 | 1 | 0 |
| **298** | 57 | 0 | 0 | 140 | 241 | 0 | 1 | 123 | 1 | 0.2 | 1 | 0 | 3 | 0 |
| **299** | 45 | 1 | 3 | 110 | 264 | 0 | 1 | 132 | 0 | 1.2 | 1 | 0 | 3 | 0 |
| **300** | 68 | 1 | 0 | 144 | 193 | 1 | 1 | 141 | 0 | 3.4 | 1 | 2 | 3 | 0 |
| **301** | 57 | 1 | 0 | 130 | 131 | 0 | 1 | 115 | 1 | 1.2 | 1 | 1 | 3 | 0 |
| **302** | 57 | 0 | 1 | 130 | 236 | 0 | 0 | 174 | 0 | 0.0 | 1 | 1 | 2 | 0 |

Check if null values present in the dataset

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [9]:
```
data.isnull().any() # No null values in any columns
```

Out[9]:
```
age         False
sex         False
cp          False
trestbps    False
chol        False
fbs         False
restecg     False
thalach     False
exang       False
oldpeak     False
slope       False
ca          False
thal        False
target      False
dtype: bool
```

So, from above it is confirmed that the given dataset has no null values in it

## Differentiate between the different types of columns

Classify the categorical and numerical attributes

In [10]:
```
allColumns = data.columns.values.tolist()
numColumns = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
catColumns = [col for col in allColumns if col not in numColumns]
```

Check if duplicate values present, if present remove it

In [11]:
```
data[data.duplicated() == True]
```

Out[11]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **164** | 38 | 1 | 2 | 138 | 175 | 0 | 1 | 173 | 0 | 0.0 | 2 | 4 | 2 | 1 |

In [12]:
```
data.drop_duplicates(inplace=True)
data[data.duplicated() == True]
```

Out[12]:

| age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

So, here the duplicate values are now removed

## Exploratory Data Analysis

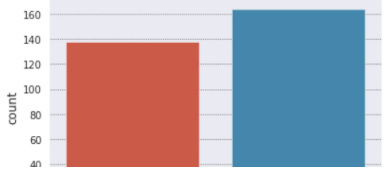
## Univariate analysis

In [26]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('ggplot')
sns.set_style('darkgrid', {"grid.color": ".3", "grid.linestyle": ":"})
```

### Graph of count vs target

In [27]:
```
sns.countplot(x=data['target'])
```

Out[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfb90c5510>
```

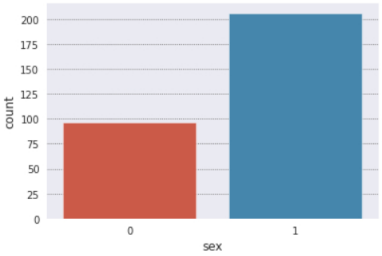



### Graph of count vs sex

In [29]:
```
sns.countplot(data['sex'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a key
word arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfb8fef950>
```



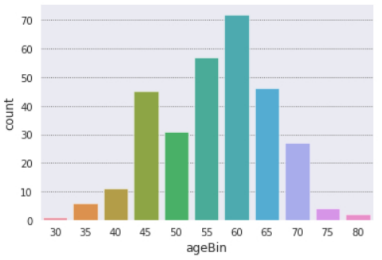

### Graph of count vs age buckets

In [30]:
```
pd.DataFrame(data['age'].describe())
data['ageBin'] = pd.cut(x=data.age, bins=[0, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80], labels=[30, 35, 40, 45, 50, 5
5, 60, 65, 70, 75, 80])
'''
(0, 30] -- 1
(30, 35] -- 6
etc
'''
print(pd.DataFrame(data['ageBin'].value_counts()))
sns.countplot(data['ageBin'])
```

```
    ageBin
60      72
55      57
65      46
45      45
50      31
70      27
40      11
35       6
75       4
80       2
30       1
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a key
word arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```

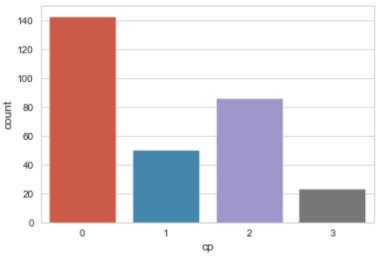
Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfb8f4ce90>
```




### Graph of count vs chest pain type

In [ ]:
```
pd.DataFrame(data['cp'].value_counts())
sns.countplot(data['cp'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f2811f28>
```



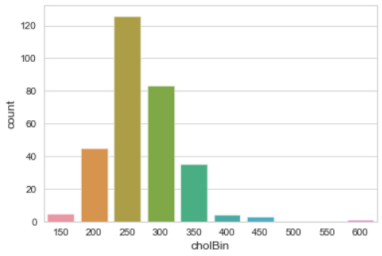

### Graph of count vs cholestrol level

In [ ]:
```
pd.DataFrame(data['chol'].describe())
'''
(125, 150],
(150, 200],
...,
```

```
(550, 600]
'''
print(range(125, 601, 50))
mylist = list(range(150, 601, 50))
mylist.append(125)
mylist.sort()
mylist
data['cholBin'] = pd.cut(data.chol, bins=mylist, labels=list(range(150, 601, 50)))
pd.DataFrame(data['cholBin'].value_counts())
sns.countplot(data['cholBin'])
```

```
range(125, 601, 50)
```

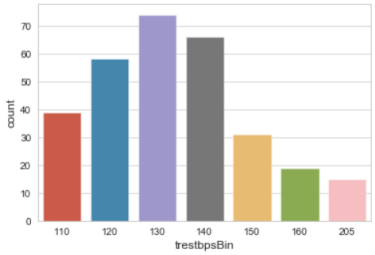
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f28c2b00>
```




### Graph of count vs trestbps

In [ ]:
```
data['trestbps'].describe()
data['trestbpsBin'] = pd.cut(data.trestbps, bins=[93, 110, 120, 130, 140, 150, 160, 205], labels=[110, 120, 130, 140, 150, 160, 205])
data['trestbpsBin'].value_counts()
sns.countplot(data.trestbpsBin)
```

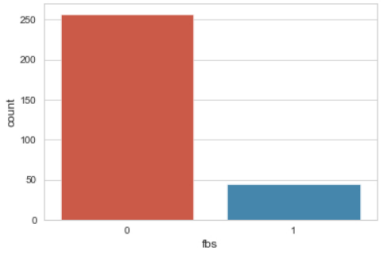
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f2771d68>
```




### Graph of count vs FBS

In [ ]:
```
# FBS
# Will be 1 if the fasting blood sugar is higher than the normal 120 mg/dl
data.fbs.unique()
# Two values = 1 and 0
sns.countplot(data.fbs)
```

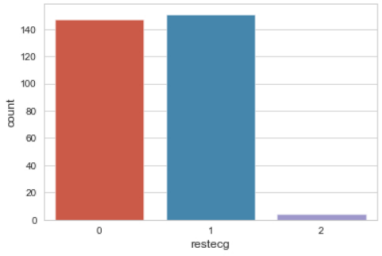
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f2723710>
```




### Graph of count vs Rest ECG

In [ ]:
```
# restecg
# Resting ECG results
data.restecg.unique()
# We get three values
# restecg: resting electrocardiographic results -- Value 0: normal -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV) -- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria 20
# What I understood is 0 is normal, 1 is pretty bad and 2 is fucked up
# Therefore ordinal categorical
sns.countplot(data.restecg)
```

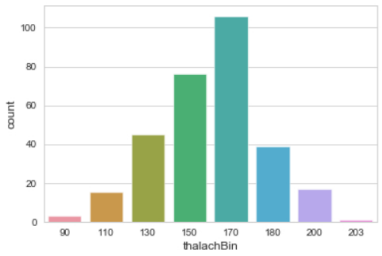
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f2755d68>
```



### Graph of count vs thalach

In [ ]:
```
# thalach
# Maximum heart rate achieved
data.thalach.unique()
# Integer numerical value
data.thalach.describe()
# min is 71
# max is 202
data['thalachBin'] = pd.cut(data.thalach, bins=[70, 90, 110, 130, 150, 170, 180, 200, 203], labels=[90, 110, 130, 150,
170, 180, 200, 203])
data.thalachBin.value_counts()
sns.countplot(data.thalachBin) # Is that a normal distributioin I see?
```

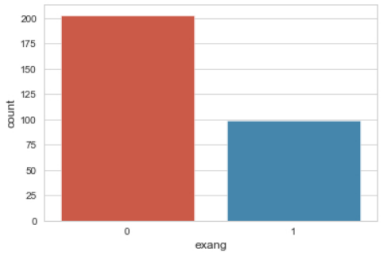
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f26511d0>
```




### Graph of count vs exang

In [ ]:
```
# exang
# Exercise included?
# 1 is Yes, 0 is No
sns.countplot(data.exang)
```

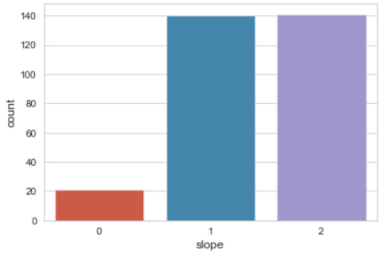
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f26372b0>
```




### Graph of count vs slope

In [ ]:
```
# Slope
# the slope of the peak exercise ST segment -- Value 1: upsloping -- Value 2: flat -- Value 3: downsloping
# I'm guessing ordinal categorical
sns.countplot(data.slope)
```

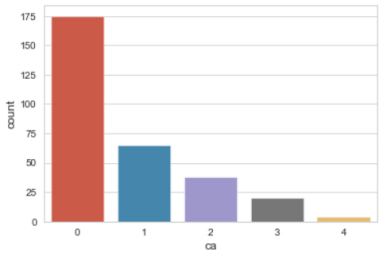
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f25708d0>
```




### Graph of count vs CA

In [ ]:
```
# ca
# number of major vessels (0-3) colored by flourosopy
data.ca.unique()
sns.countplot(data.ca)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f24d6748>
```




In [ ]:
```
# And thus the univariate analysis is complete
```

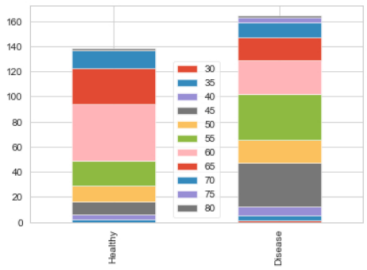
## Multivariate Analysis

## Multivariate Analysis

### Graph of Age with respect to heart disease

In [ ]:
```
target1 = data[data['target']==1]['ageBin'].value_counts()
target0 = data[data['target']==0]['ageBin'].value_counts()
temp = pd.DataFrame([target0, target1])
temp.index = ['Healthy', 'Disease']
temp.plot(kind='bar', stacked=True)
```

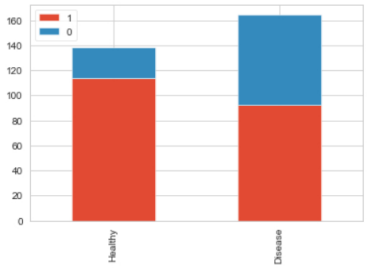
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f240a940>
```




### Graph of Sex with respect to heart disease

In [ ]:
```
target1 = data[data['target']==1]['sex'].value_counts()
target0 = data[data['target']==0]['sex'].value_counts()
tempDf = pd.DataFrame([target0, target1])
tempDf.index = ['Healthy', 'Disease']
tempDf.plot(kind='bar', stacked=True)
```

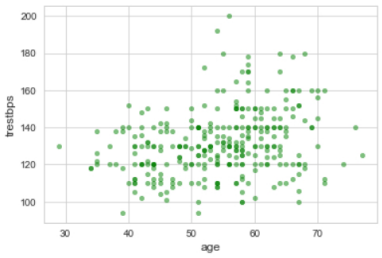
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f24aad30>
```




### Graph of Age with respect to trestbps

In [ ]:
```
# Relationship between age and trestbps
data.plot(kind='scatter', x='age', y='trestbps', color='green', alpha=0.5)
# More people will have higher blood pressure as they age
```

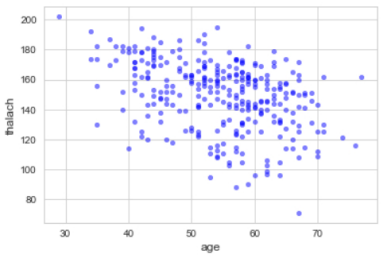
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f233aeb8>
```




### Graph of Age with respect to maximum heartrate achieved

In [ ]:
```
data.plot(kind='scatter', x='age', y='thalach', color='blue', alpha=0.5)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49f2298518>
```

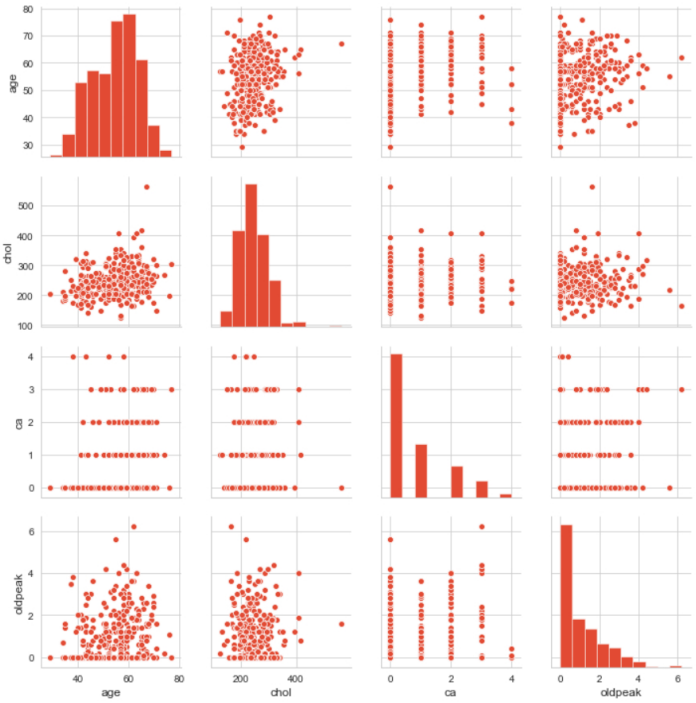



As you age, the maximum heart rate you can achieve will gradually reduce

### Graph of Age, cholestrol, ca and oldpeak

In [ ]:
```
sns.pairplot(data.loc[:, ['age', 'chol', 'ca', 'oldpeak']])
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f49f24aae80>
```




### Correlation Matrix

In [ ]:
```
dataCorr = data.corr()['target'][:-1]
goldFeaturesList = dataCorr[abs(dataCorr) > 0.1].sort_values()
goldFeaturesList
```

Out[ ]:
```
exang      -0.435601
oldpeak    -0.429146
ca         -0.408992
thal       -0.343101
sex        -0.283609
age        -0.221476
trestbps   -0.146269
restecg     0.134874
slope       0.343940
thalach     0.419955
cp          0.432080
Name: target, dtype: float64
```

## Modeling

In [ ]:
```
data = pd.read_csv('heart.csv')
y = data['target']
X = data.drop(['target'], axis=1)
from sklearn.model_selection import train_test_split
Xtrain, Xtest, yTrain, yTest = train_test_split(X, y, test_size=0.25, random_state=0)
```

## Performance metrics

In [ ]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_curve, auc

def evaluateModel(yTrue, yPredicted, modelName):
    print("Result of prediction for the model - ", modelName)
    confMatrix = confusion_matrix(yTrue, yPredicted)
    print("Confusion Matrix")
    print(confMatrix)
    precision = round(precision_score(yTrue, yPredicted), 4)
    print("Precision is ", precision)
    print("Out of all predicted as Heart Patients, {} percent actually have Heart Disease".format(precision*100))
    recall = round(recall_score(yTrue, yPredicted), 4)
    print("Recall is ", recall)
    print("Out of all actual heart patients, {0} is able to detect {1} percent of them".format(modelName, recall*100))
    print("Drawing the ROC")
    fpr, tpr, thresholds = roc_curve(yPredicted, yTrue)
    roc_auc = round(auc(fpr, tpr), 3)
    plt.figure(figsize=(10, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=1, label="{0}, area={1}".format(modelName, roc_auc))
    plt.plot([0, 1], [0, 1], color='blue', lw=1, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("Flase Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver Operating Characteristic for {}".format(modelName))
    plt.legend(loc="lower right")
    plt.show()
```

## Modeling

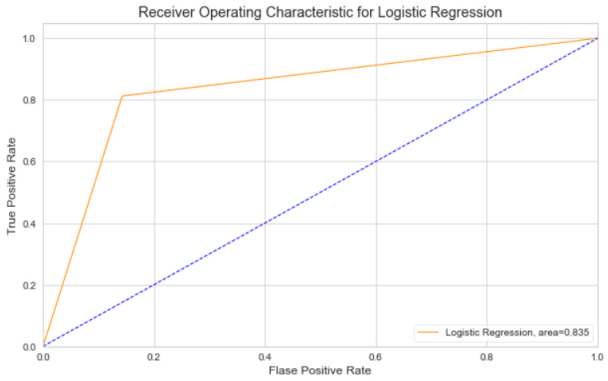
### Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
logisticRegression = LogisticRegression(random_state=0)
logisticRegression.fit(Xtrain, yTrain)
```

```
evaluateModel(yTest, yPredLogReg, "Logistic Regression")
```

```
=====================================================
Result of prediction for the model -  Logistic Regression
Confusion Matrix
[[24  9]
 [ 4 39]]
Precision is  0.8125
Out of all predicted as Heart Patients, 81.25 percent actually have Heart Disease
Recall is  0.907
Out of all actual heart patients, Logistic Regression is able to detect 90.7 percent of them
Drawing the ROC
```

```
/home/themadscientist/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Defa
ult solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

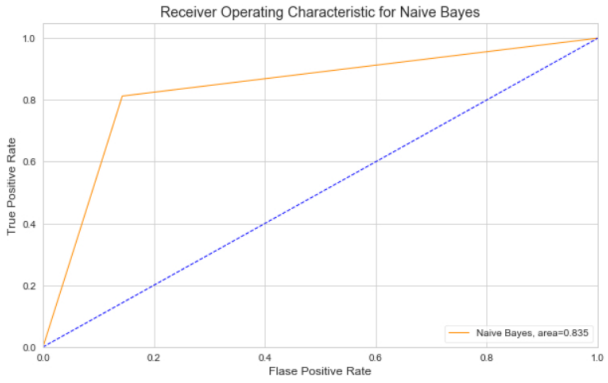



```
=====================================================
```

## Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
naiveBayes = GaussianNB(priors=None)
naiveBayes.fit(Xtrain, yTrain)
yPredNaiveBayes = naiveBayes.predict(Xtest)
evaluateModel(yTest, yPredNaiveBayes, "Naive Bayes")
```

```
=====================================================
Result of prediction for the model -  Naive Bayes
Confusion Matrix
[[24  9]
 [ 4 39]]
Precision is  0.8125
Out of all predicted as Heart Patients, 81.25 percent actually have Heart Disease
Recall is  0.907
Out of all actual heart patients, Naive Bayes is able to detect 90.7 percent of them
Drawing the ROC
```

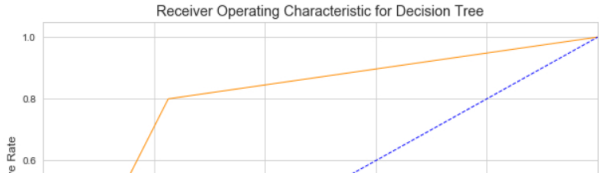


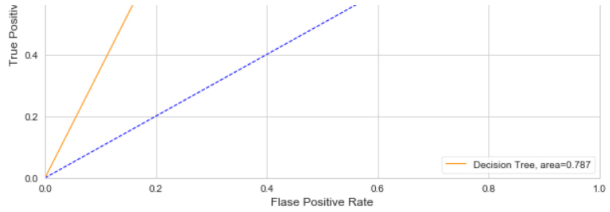

```
=====================================================
```

## Decision Tree Classifier

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
decTree = DecisionTreeClassifier(criterion='entropy', random_state=0)
decTree.fit(Xtrain, yTrain)
yPredDecTree = decTree.predict(Xtest)
evaluateModel(yTest, yPredDecTree, "Decision Tree")
```

```
=====================================================
Result of prediction for the model -  Decision Tree
Confusion Matrix
[[24  9]
 [ 7 36]]
Precision is  0.8
Out of all predicted as Heart Patients, 80.0 percent actually have Heart Disease
Recall is  0.8372
Out of all actual heart patients, Decision Tree is able to detect 83.72 percent of them
Drawing the ROC
```





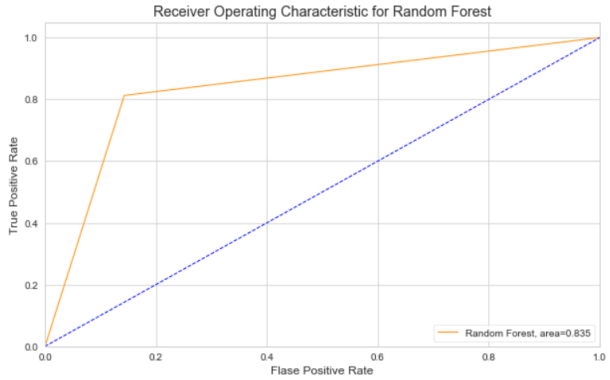

==========

## Random Forest Classifier

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
randForest = RandomForestClassifier(criterion='entropy', random_state=0)
randForest.fit(Xtrain, yTrain)
yPredRandForest = randForest.predict(Xtest)
evaluateModel(yTest, yPredRandForest, "Random Forest")
```

```
==========
Result of prediction for the model -  Random Forest
Confusion Matrix
[[24  9]
 [ 4 39]]
Precision is  0.8125
Out of all predicted as Heart Patients, 81.25 percent actually have Heart Disease
Recall is  0.907
Out of all actual heart patients, Random Forest is able to detect 90.7 percent of them
Drawing the ROC
```

```
/home/themadscientist/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The defaul
t value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

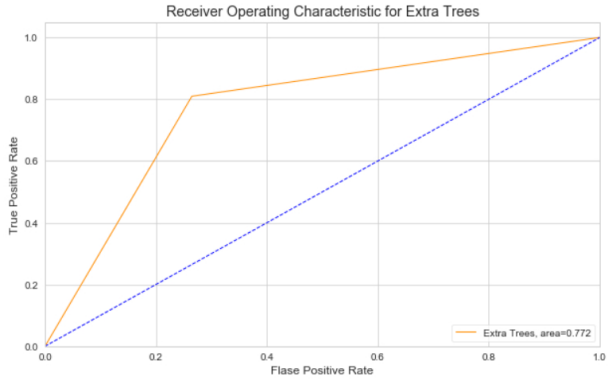



==========

## Extra Trees Classifier

In [ ]:
```
from sklearn.ensemble import ExtraTreesClassifier
extraTrees = ExtraTreesClassifier(n_estimators=10, criterion='entropy', bootstrap=False, random_state=0)
extraTrees.fit(Xtrain, yTrain)
yPredExtraTrees = extraTrees.predict(Xtest)
evaluateModel(yTest, yPredExtraTrees, "Extra Trees")
```

```
==========
Result of prediction for the model -  Extra Trees
Confusion Matrix
[[25  8]
 [ 9 34]]
Precision is  0.8095
Out of all predicted as Heart Patients, 80.95 percent actually have Heart Disease
Recall is  0.7907
Out of all actual heart patients, Extra Trees is able to detect 79.07 percent of them
Drawing the ROC
```




==========

# Results

The AUC for the different models are summarized below:

Traditional Models

1. Logistic Regression - 0.835
2. Naive Bayes -0.835
3. Decision Tree - 0.787
4. Random Forest - 0.835
5. Extra Trees - 0.772

What we can see is:

1. When a single decision tree gave an AUC of 0.787, a Random Forest trained on similar individual decision trees gave a better score of 0.835.
2. Extra Trees Classifier gave the least score