Heart Disease - Classifications
By Kassem@elcaiseri

Heart Disease - Classifications + Visualization

---

1. Introduction

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient.

In [1]:

```python
import warnings
warnings.simplefilter("ignore")
```

2. Data Preparation

In [2]:

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list t
he files in the input directory

import os
#print(os.listdir("../input"))

# Any results you write to the current directory are saved as output.
```

In [3]:

```python
df = pd.read_csv("./heart.csv")
```

In [4]:

```
df.head()
```

Out[4]:

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

In [5]:

```
df.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 [6]:

```
# check for NAN values
df.isnull().sum()
```

Out[6]:

```
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 [7]:

```
# data size
df.shape
```

Out[7]:

```
(303, 14)
```

In [8]:

```
df.describe()
```

Out[8]:

	age	sex	cp	trestbps	chol	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000

In [9]:

```
# target distribution
df.target.value_counts()
```

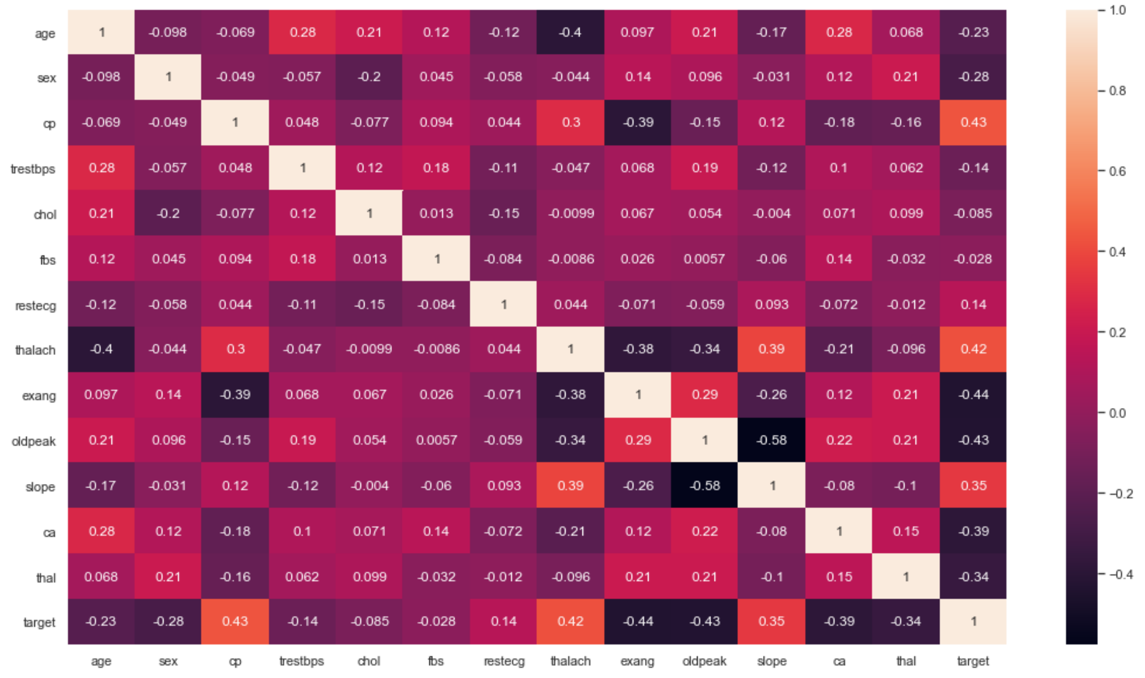
Out[9]:

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

3. Visualization

In [10]:

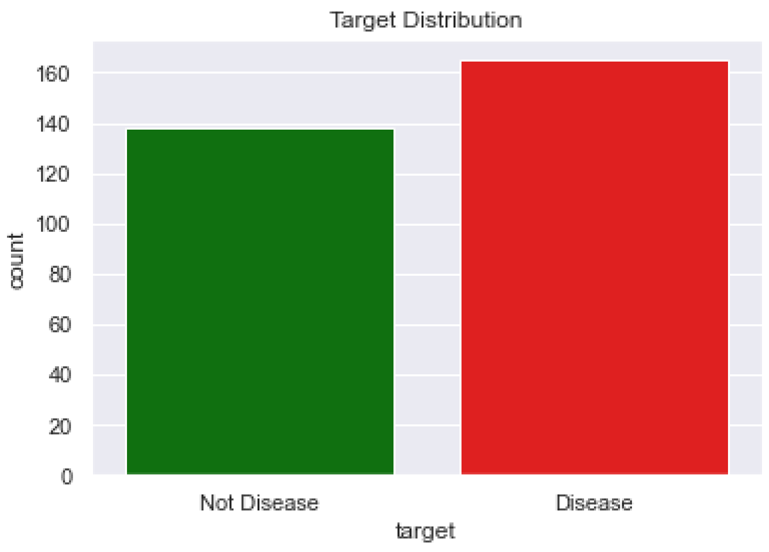
```
# show correlation matrix
corr = df.corr()
plt.figure(figsize=(18,10))
sns.heatmap(corr, annot=True, )
plt.show()
```



According to the color bar, the lighter color between features, the more correlation (linear) they had.

In [11]:

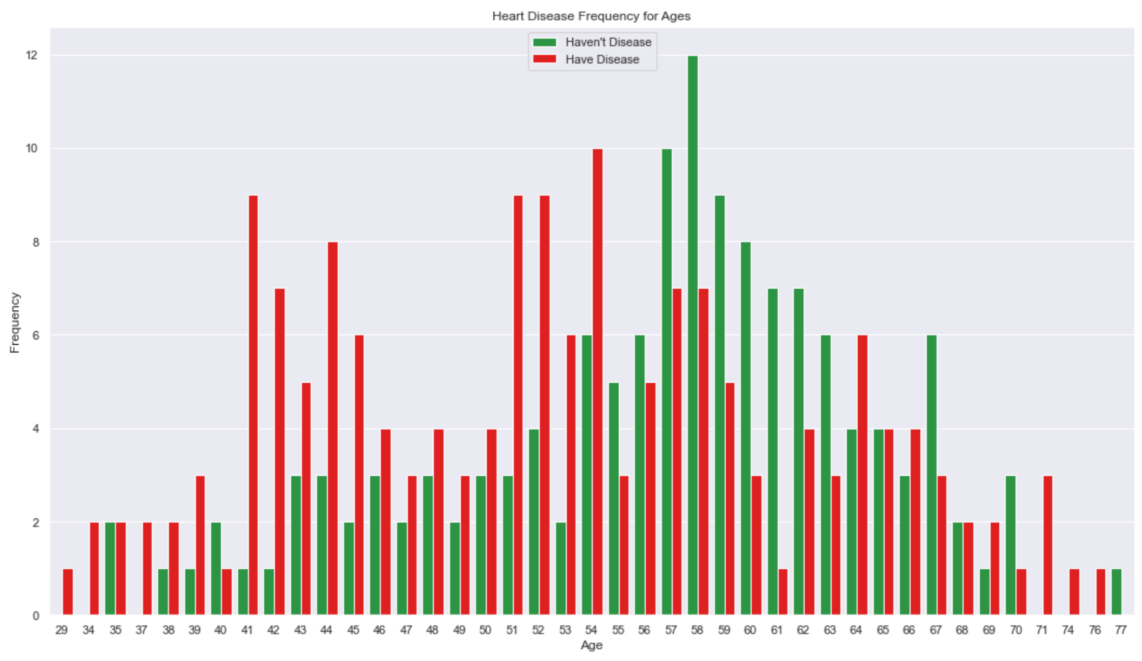
```
# plot target dist.
sns.countplot(df.target, palette=['green', 'red'])
plt.xticks([0, 1], labels=['Not Disease', 'Disease'])
plt.title("Target Distribution");
```



The dataset is quite small, but the target proportion for patients who have disease are slightly more than who have not.

In [12]:

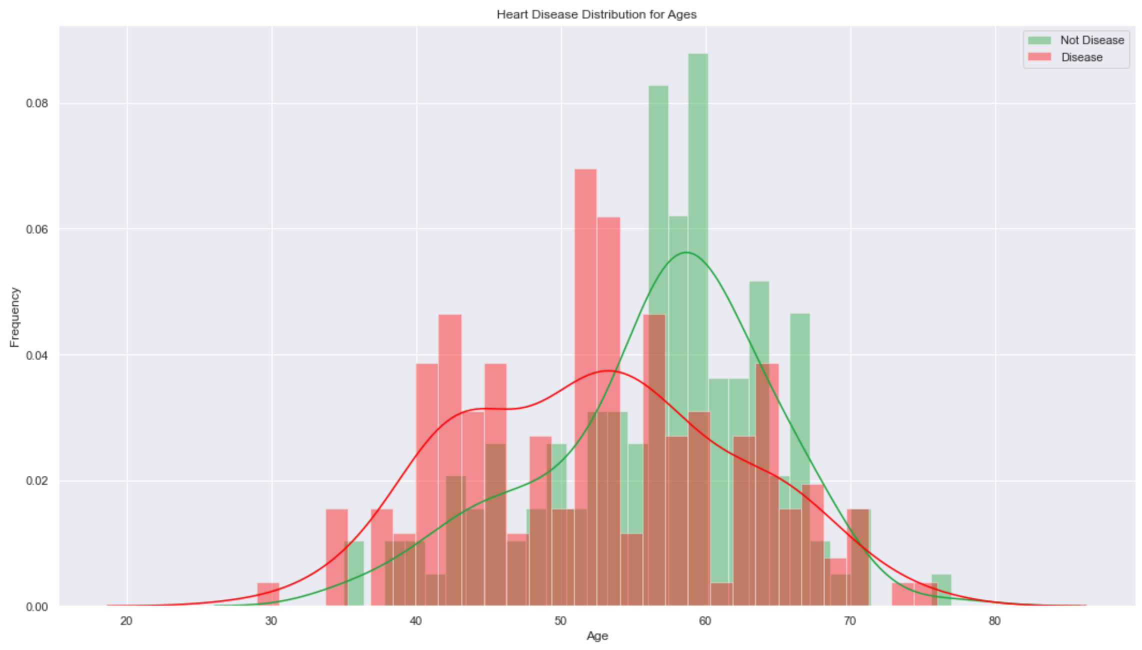
```
plt.figure(figsize=(18, 10))
sns.countplot(x='age', hue='target', data=df, palette=['#1CA53B', 'red'])
plt.legend(["Haven't Disease", "Have Disease"])
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



The disease rate is too high for age from 29~54 years old compare with whom in the same age, and this rate drop for people who older than 55 years old which quite interesting.

In [13]:

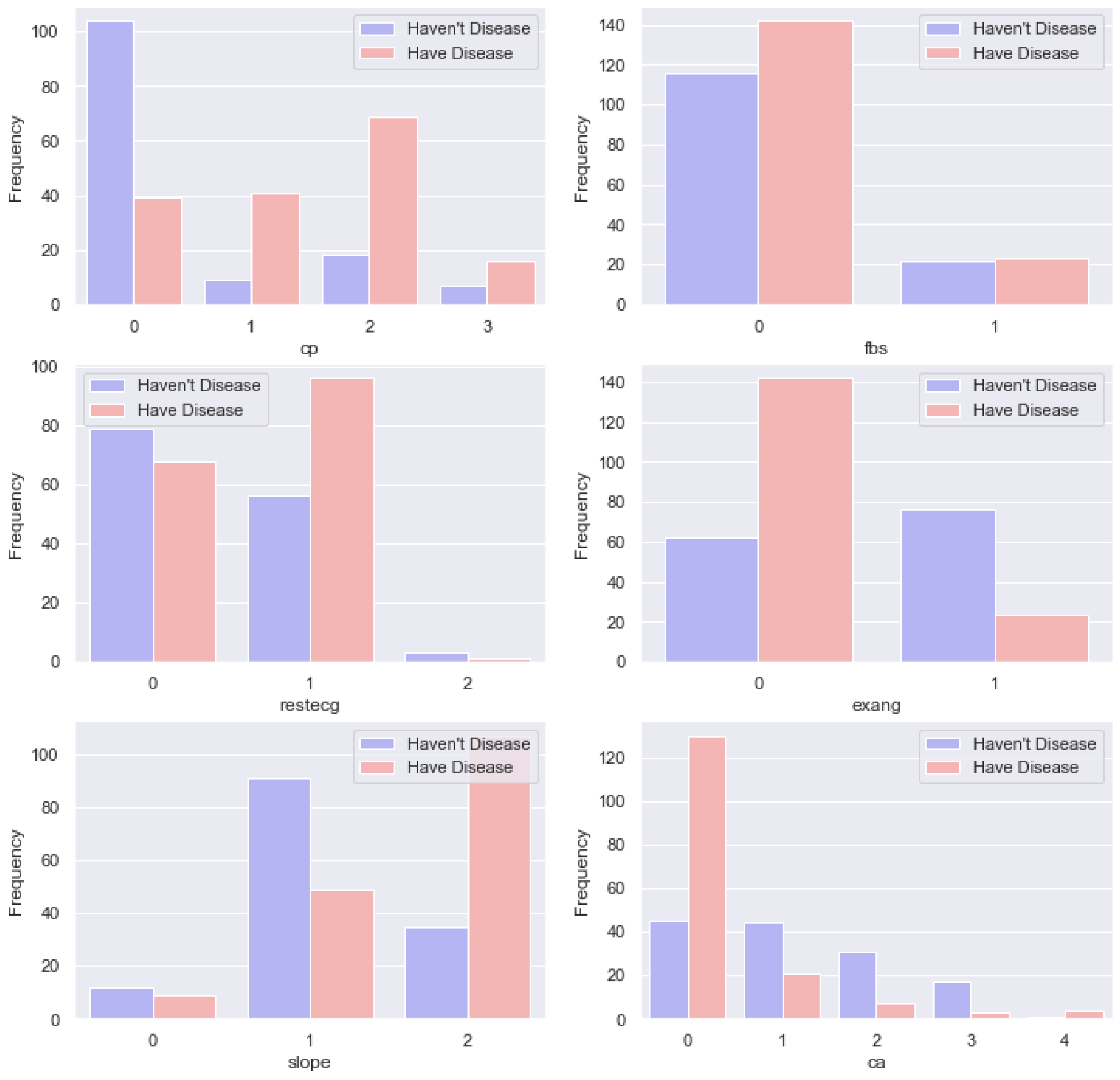
```
#sns.set_style("whitegrid")
plt.figure(figsize=(18, 10))
sns.distplot(df.age[df['target'] == 0], bins=30, color='#1CA53B', label='Not Dise
ase')
sns.distplot(df.age[df['target'] == 1], bins=30, color='red', label='Disease')
plt.legend()
plt.title('Heart Disease Distribution for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



Here is another graph which prove what we said above.

In [14]:

```
fig, axes = plt.subplots(3, 2, figsize=(12,12))
fs = ['cp', 'fbs', 'restecg','exang', 'slope', 'ca']
for i, axi in enumerate(axes.flat):
    sns.countplot(x=fs[i], hue='target', data=df, palette='bwr', ax=axi)
    axi.set(ylabel='Frequency')
    axi.legend(["Haven't Disease", "Have Disease"])
```

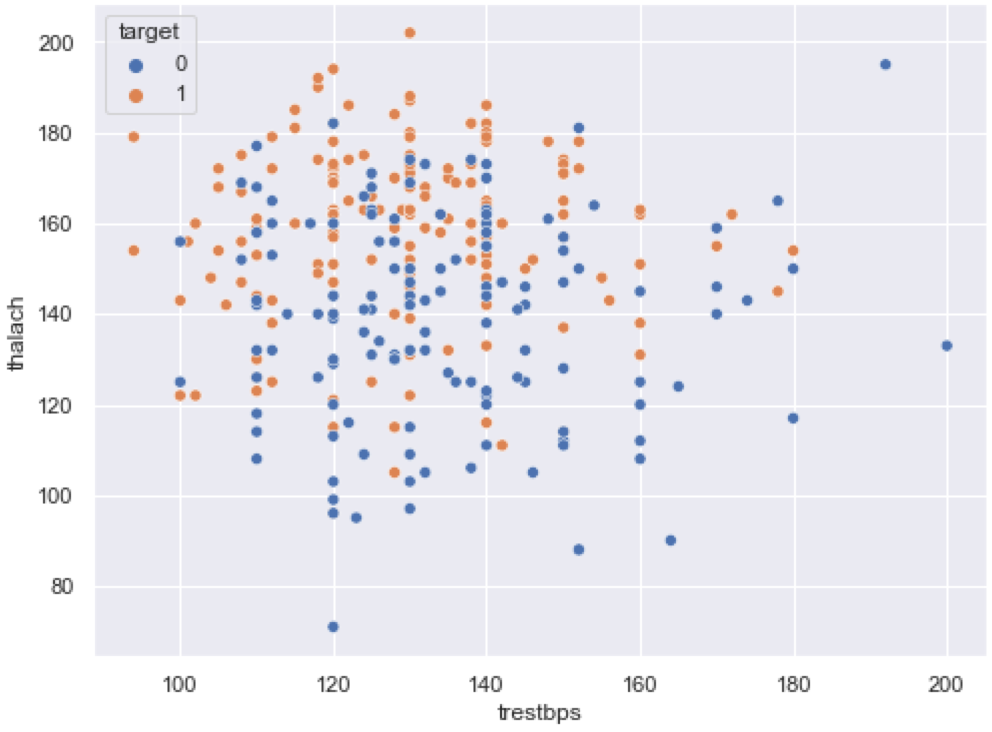


some insight from figures:

1. people with (cp==0 and slope==1) are less likely to have disease.
2. people with (ca==0, slope==2 and exang==0) are more likely to have disease.

In [15]:

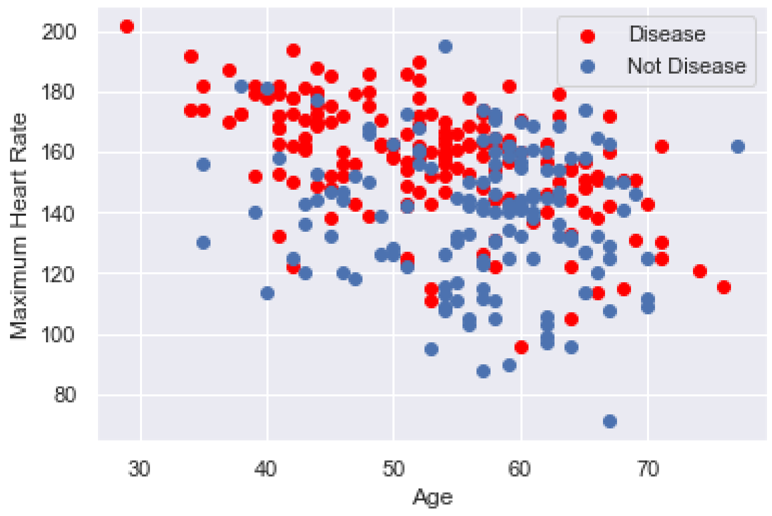
```
plt.figure(figsize=(8,6))
sns.scatterplot(x='trestbps', y='thalach', data=df, hue='target')
plt.show()
```



Most of disease case exist on the upper left of trestbps vs thalach graph.

In [16]:

```
plt.scatter(x=df.age[df.target==1], y=df.thalach[(df.target==1)], c="red")
plt.scatter(x=df.age[df.target==0], y=df.thalach[(df.target==0)])
plt.legend(["Disease", "Not Disease"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```



4. Machine Learning

In [17]:

```
from sklearn.preprocessing import StandardScaler

# Import tools
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.metrics import accuracy_score, confusion_matrix, classification_repo
rt
from sklearn.metrics import roc_curve, auc

np.random.seed(42)
```

In [18]:

```
# Define our feasures and leabels
X = df.drop(['target'], axis=1).values
y = df['target'].values
```

In [19]:

```
# normalize for training
scale = StandardScaler()
X = scale.fit_transform(X)
```

YOU HAVE TO DO THE NORMALIZEING PART AFTER SPLITTING YOUR DATA TO AVOID **DATA LEAKAGES.**

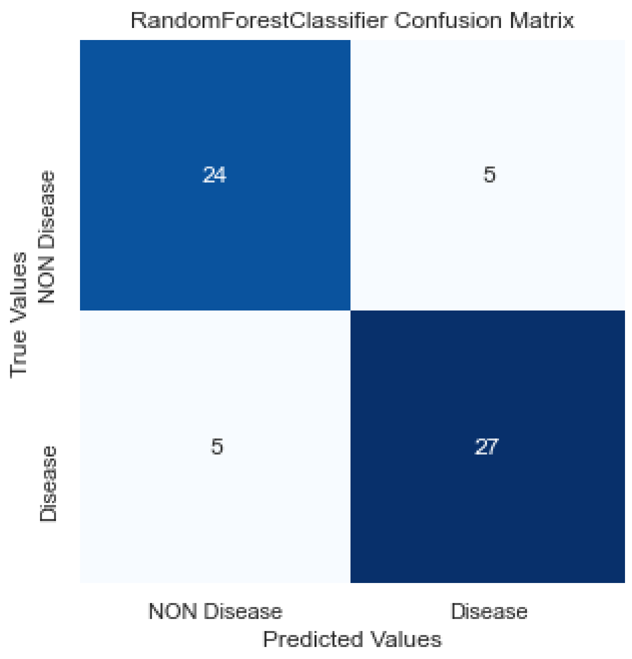
In [20]:

```
class Model:
    """
    Machine Learning model class contains all necessary features.

    Attributes:
    model: sklearn model object
    X: Features
    y: target

    Return:
    Trained model

    """
    def __init__(self, model, X, y):
        self.model = model
        self.X = X
        self.y = y
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(s
elf.X, self.y, test_size=0.2, random_state=42)

        self.model.fit(self.X_train, self.y_train)
        print(f"{self.model_str()} Model Trained..")
        self.y_pred = self.model.predict(self.X_test)

    def model_str(self):
        """Model name"""
        return str(self.model.__class__.__name__)

    def crossValScore(self, cv=5):
        """cross Val Score"""
        print(self.model_str() + "\n" + "="*60)
        scores = ["accuracy", "precision", "recall", "roc_auc"]
        for score in scores:
            cv_acc = cross_val_score(self.model,
                                     self.X_train,
                                     self.y_train,
                                     cv=cv,
                                     scoring=score).mean()

            print("Model " + score + " : " + "%.3f" % cv_acc)


    def accuracy(self):
        """accuracy"""
        accuarcy = accuracy_score(self.y_test, self.y_pred)
        #print(self.model_str() + " Model " + "Accuracy is: ")
        return accuarcy

    def confusionMatrix(self):
        """plot confusion Matrix"""
        plt.figure(figsize=(5, 5))
        mat = confusion_matrix(self.y_test, self.y_pred)
        sns.heatmap(mat, square=True,
                    annot=True,
                    cbar=False, cmap='Blues',
                    xticklabels=["NON Disease", "Disease"],
                    yticklabels=["NON Disease", "Disease"])

        plt.title(self.model_str() + " Confusion Matrix")
        plt.xlabel('Predicted Values')
        plt.ylabel('True Values');
        plt.show();

    def classificationReport(self):
        """show classification Report"""
        print(self.model_str() + " Classification Report" + "\n" + "="*60)
        print(classification_report(self.y_test,
                                    self.y_pred,
                                    target_names=['Non Disease', 'Disease']))

    def rocCurve(self):
        """show rocCurve"""
        y_prob = self.model.predict_proba(self.X_test)[:,1]
        fpr, tpr, thr = roc_curve(self.y_test, y_prob)
        lw = 2
        plt.figure(figsize=(8, 6))
        plt.plot(fpr, tpr,
                 color='darkorange',
                 lw=lw,
                 label="Curve Area = %0.3f" % auc(fpr, tpr))
        plt.plot([0, 1], [0, 1], color='green',
                 lw=lw, linestyle='--')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(self.model_str() + ' Receiver Operating Characteristic Plot')
        plt.legend(loc="lower right")
        plt.show()
```

In [21]:

```python
from sklearn.ensemble import RandomForestClassifier

# call model
clf = Model(model=RandomForestClassifier(), X=X, y=y)

clf.crossValScore(cv=5)

clf.confusionMatrix()

clf.classificationReport()
```

```
RandomForestClassifier Model Trained..
RandomForestClassifier
============================================================
Model accuracy : 0.814
Model precision : 0.828
Model recall : 0.849
Model roc_auc : 0.901
```

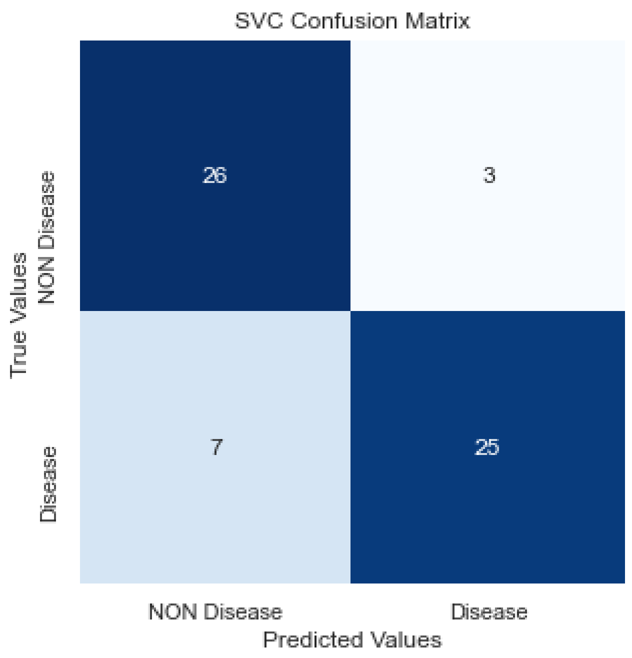


```
RandomForestClassifier Classification Report
============================================================
              precision    recall  f1-score   support

 Non Disease       0.83      0.83      0.83        29
     Disease       0.84      0.84      0.84        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61
```

In [22]:

```
from sklearn.svm import SVC

svm = Model(model=SVC(C=5, probability=True), X=X, y=y)

svm.crossValScore(cv=10)

svm.confusionMatrix()

svm.classificationReport()
```

```
SVC Model Trained..
SVC
============================================================
Model accuracy : 0.773
Model precision : 0.812
Model recall : 0.781
Model roc_auc : 0.870
```

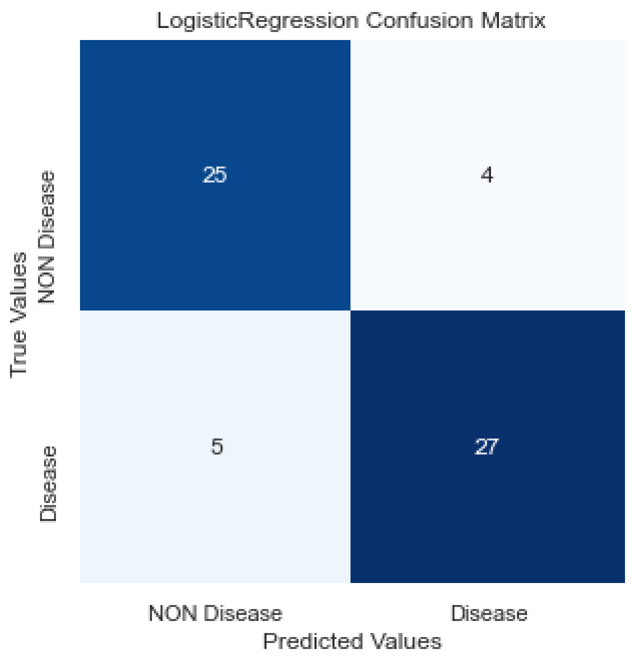


```
SVC Classification Report
============================================================
              precision    recall  f1-score   support

 Non Disease       0.79      0.90      0.84        29
     Disease       0.89      0.78      0.83        32

    accuracy                           0.84        61
   macro avg       0.84      0.84      0.84        61
weighted avg       0.84      0.84      0.84        61
```

In [23]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr = Model(model=lr, X=X, y=y)

lr.crossValScore()

lr.confusionMatrix()

lr.classificationReport()
```

```
LogisticRegression Model Trained..
LogisticRegression
============================================================
Model accuracy : 0.814
Model precision : 0.822
Model recall : 0.849
Model roc_auc : 0.886
```

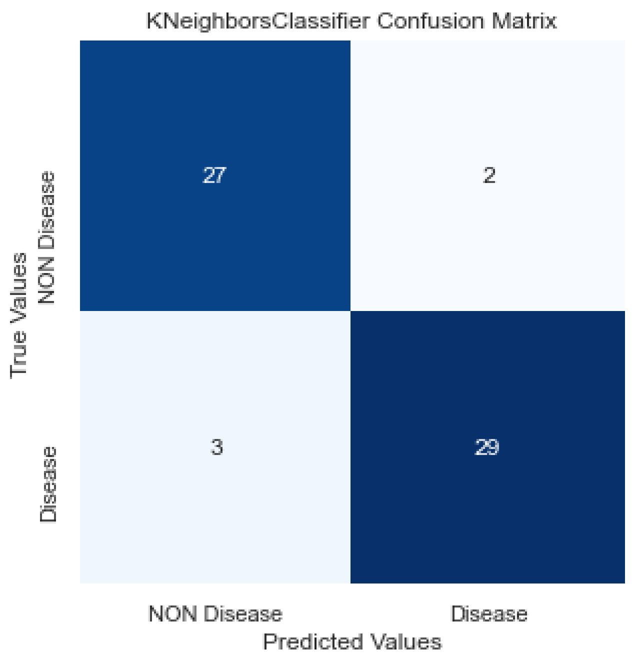


```
LogisticRegression Classification Report
============================================================
              precision    recall  f1-score   support

 Non Disease       0.83      0.86      0.85        29
     Disease       0.87      0.84      0.86        32

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier

knn = Model(model=KNeighborsClassifier(n_neighbors=5), X=X, y=y)

knn.crossValScore(cv=5)

knn.confusionMatrix()

knn.classificationReport()
```

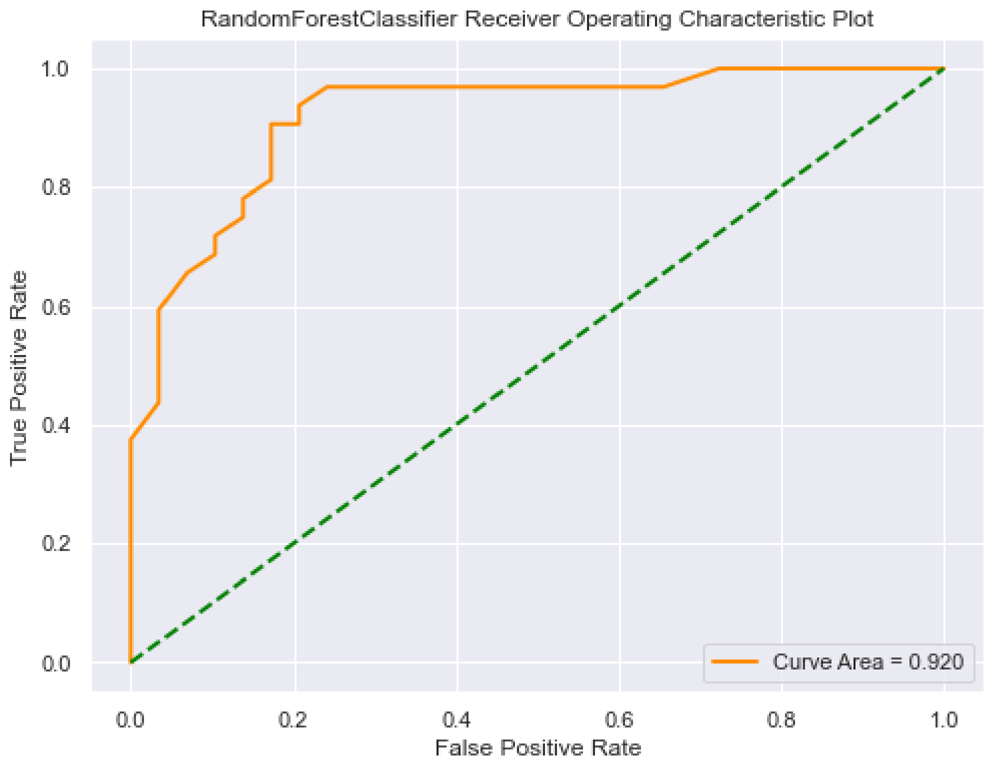
```
KNeighborsClassifier Model Trained..
KNeighborsClassifier
============================================================
Model accuracy : 0.806
Model precision : 0.813
Model recall : 0.850
Model roc_auc : 0.869
```

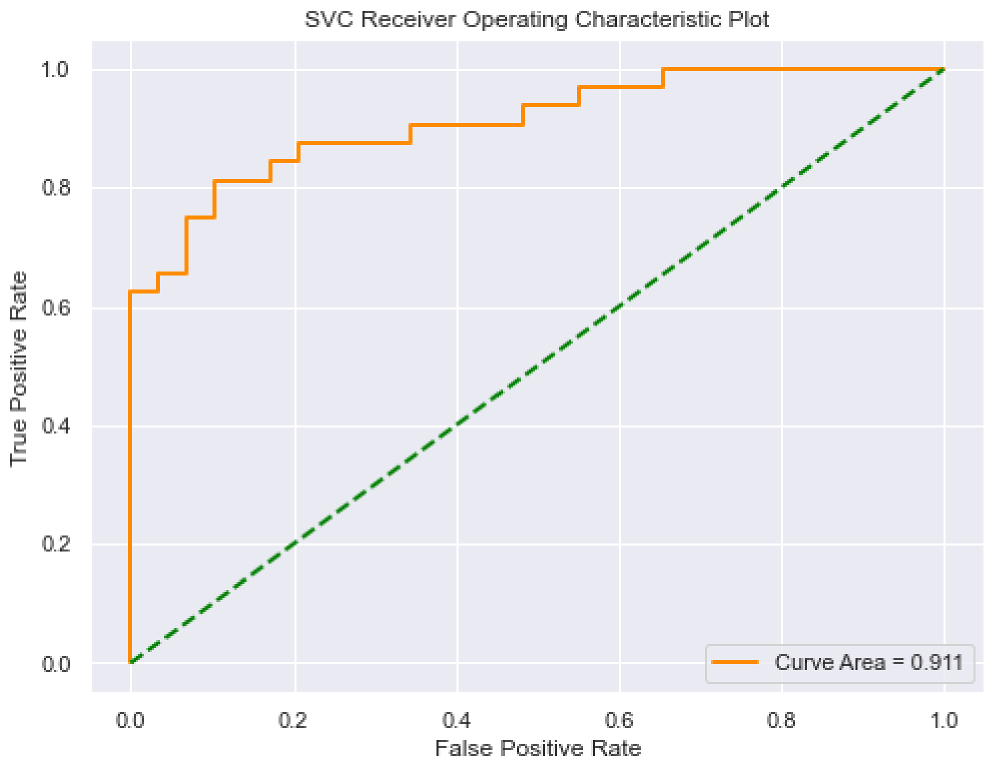


```
KNeighborsClassifier Classification Report
============================================================
              precision    recall  f1-score   support

 Non Disease       0.90      0.93      0.92        29
     Disease       0.94      0.91      0.92        32

    accuracy                           0.92        61
   macro avg       0.92      0.92      0.92        61
weighted avg       0.92      0.92      0.92        61
```

In [25]:

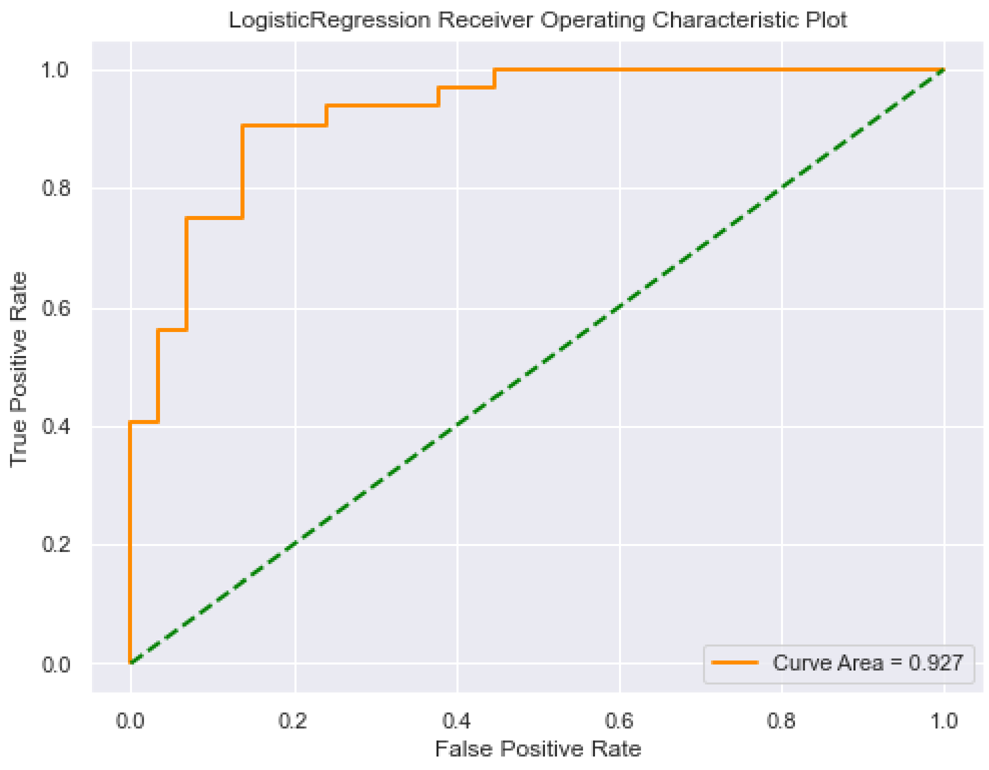
```
models = [clf, svm, lr, knn]
for model in models[:2]:
    model.rocCurve()
```

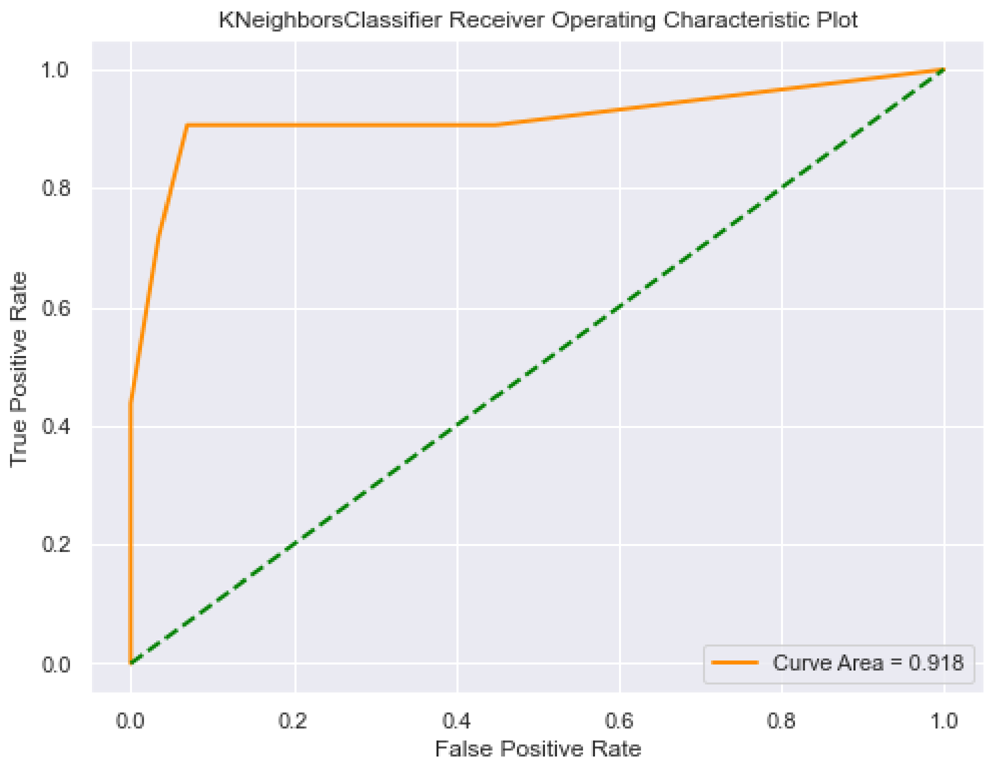




In [26]:

```
for model in models[2:]:
    model.rocCurve()
```



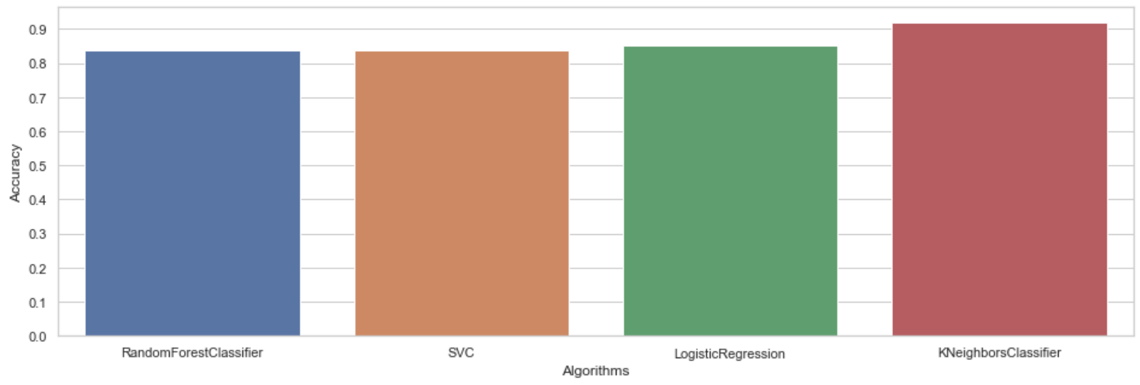


In [27]:

```
models = [clf, svm, lr, knn]
names = []
accs = []
for model in models:
    accs.append(model.accuracy());
    names.append(model.model_str());
```

In [28]:

```
sns.set_style("whitegrid")
plt.figure(figsize=(16,5))
plt.yticks(np.arange(0,1.2,0.1))
plt.ylabel("Accuracy")
plt.xlabel("Algorithms")
sns.barplot(x=names, y=accs)
plt.savefig('models_accuracy.png')
plt.show()
```



According to the accuracy metric we have a winner here.

Note that: ACCURACY metric isn't the best metric in such situation, you can test other metrics for your certain.

In [ ]:

```

```