The Relationship Between Cardiac Testing and Heart Disease

By Paul Venuto

Abstract

I used the Heart Disease Dataset to examine the relationship between variables such as demographics and cardiac tests and the diagnosis of heart disease, as defined by narrowing of the coronary artery. More specifically, I wanted to see if heart disease can be accurately predicted by the clinical and laboratory test data as well as demographic information. I used a combination of exploratory data analysis, data visualization, and machine learning techniques to answer the research questions. The findings indicate mild correlations between some of the variables and heart disease. Machine learning algorithms applied to the data could accurately predict heart disease in the test data from 72.1% to 83.6% of the time, which on the surface seems like a remarkable achievement given the small size of the dataset.

Motivation

In medicine, physicians face many challenges in making an accurate diagnosis. These include inconclusive and insufficient testing, limited patient access to medical care, and inconsistency with diagnostic protocols. For example, doctors may order different tests for a patient with the same symptoms, and therefore reach different diagnoses. An accurate diagnosis is critical to patient care, and may be a matter of life death.

This is personal to me as I am a physical therapist (PT), and while I don’t make diagnoses in the traditional sense, I must make an accurate assessment of my patient’s condition based on their signs and symptoms. Typically patients come to me with little diagnostic work up from their physicians, so my initial evaluation and assessment guides me in selecting optimal treatments to improve their physical condition.

My interest is in learning how data science and machine learning can be used to make more accurate medical diagnoses.

Dataset

The dataset I used for my project is the Heart Disease Dataset, which was created by Cardiologists and Researchers from Hungary, Switzerland, and the United States. Specifically, I will examine a subset of 14 of the attributes which includes demographic information as well as cardiac function test results. It includes 303 samples of test results and demographic data for cardiac patients. I found the dataset on both the UCI Machine Learning Repository and Kaggle websites.

Data Preparation and Cleaning

First, I checked for NA and null values, of which there were none, and then I checked for duplicates. There was one duplicate so it was removed.

To apply machine learning algorithms, the data was normalized and categorical variables had to be converted into indicator (dummy) variables. The resulting dataset had only 0 or 1 values for these variables, which is required for machine learning applications

Research Questions

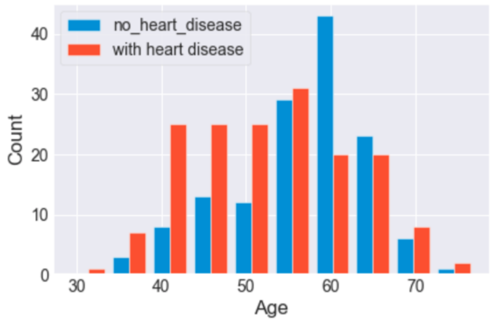
1. What is the relationship between cardiac function test results and heart disease?

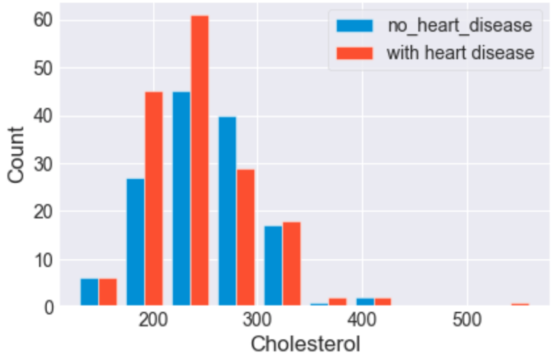
2. Can heart disease by accurately predicted by such test results?

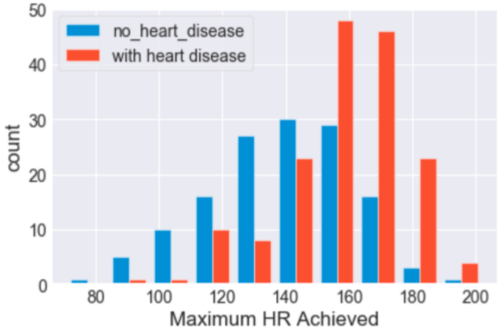
Methods

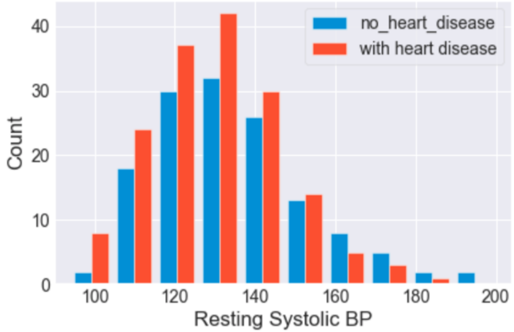
A variety of data science methods were used in the analysis of this dataset, including exploratory data analysis, data visualization, correlation, machine learning algorithms, and confusion matrixes.

Findings: which variables were associated with heart disease?









The number of samples with heart disease was similar among subjects aged 40s-60s.

For systolic blood pressure and cholesterol, the number of samples with heart disease was highest in the middle range (130s and mid 200s respectively).

For maximum HR achieved (on a stress test), the highest counts of heart disease were those in the 150s and 160s

Findings: Correlation

No strong correlations between individual variables and heart disease were found.

The strongest positive correlation was chest pain (0.43) and the strongest negative correlation was exercise-induced angina (-0.44). This is an interesting finding because intuitively one would think that exercised-induced angina would be an indicator of heart disease

target	1.000000
cp	0.432080
thalach	0.419955
slope	0.343940
restecg	0.134874
fbs	-0.026826
chol	-0.081437
trestbps	-0.146269
age	-0.221476
sex	-0.283609
thal	-0.343101
ca	-0.408992
oldpeak	-0.429146
exang	-0.435601

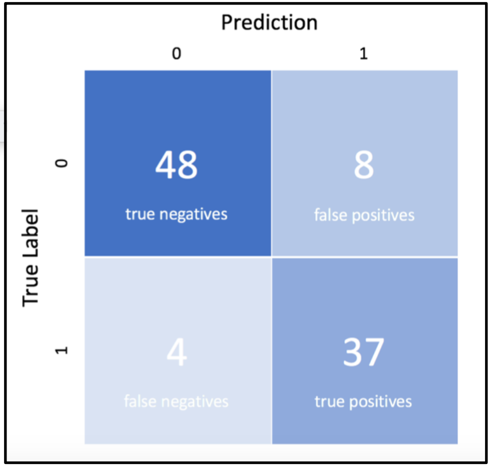
Findings: Machine Learning

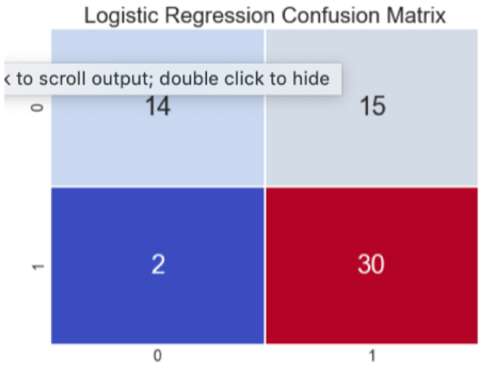
The data were split into a training and test set, and five supervised classification algorithms were applied to the test data. They were able to predict heart disease ranging from 72.1% to 83.6% of the time, depending upon the algorithm.

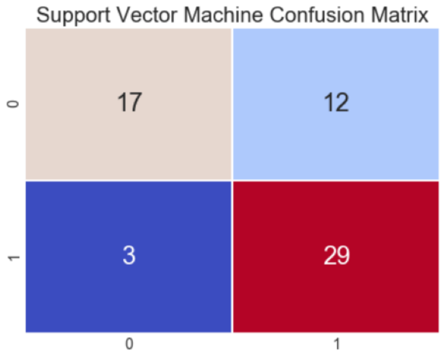
Algorithm	Accuracy Score
Logistic Regression	72.1%
Support Vector Machine	75.4%
Decision Tree	83.6%
Naïve Bayes	80.3%
Random Forest	83.6%

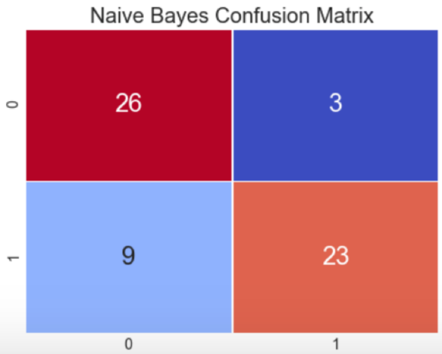
Findings: Confusion Matrixes

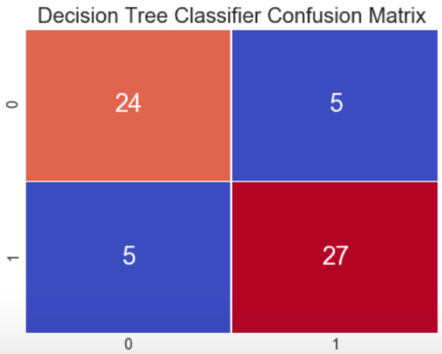
Key:

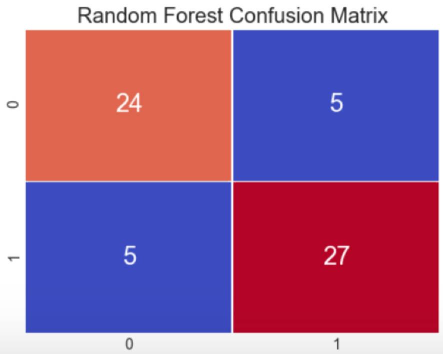












Findings: Confusion Matrixes

The confusion matrixes reveal that there are too many false positives and true negatives in the modeling for this to be considered valuable models. False negatives are particularly dangerous, as they incorrectly diagnose someone as not having heart disease, when in fact they do have heart disease.

Limitations

The dataset is small (only 303 samples), and nothing is known about the demographics of the patients except age and gender. This might explain the many false negatives and false positives in the confusion matrixes.

There are other techniques that could be applied, such as cross-validation, that may improve the machine learning modeling.

Also, the target variable was categorized based on coronary artery occlusion. But this is only one type of heart disease, and other types were not considered. It would also be interesting to know the exact amount the coronary artery was occluded. Then it could be turned into a regression problem, where the amount of heart disease is predicted.

Conclusions

The dataset has revealed some interesting associations among cardiac test results and heart disease. In particular, long held beliefs about clinical tests such as blood pressure, lab values such as cholesterol, and cardiac stress tests may need to be reconsidered.

While there are some interesting findings from the data, more data is needed to better understand the relationship between cardiac function test results and heart disease.

Based on the results of machine learning modeling, insightful inferences cannot be made from this one dataset. That would require a much richer dataset with more samples, that include people of all ethnic, racial, and geographical backgrounds. Perhaps additional cardiac function tests need to be considered as well.

Acknowledgements

The dataset is from Kaggle (https://www.kaggle.com/ronitf/heart-disease-uci) and originally published by the UC Irvine machine learning repository website (https://archive.ics.uci.edu/ml/datasets/Heart+Disease).

While no one gave me feedback on this project, I referenced several other Kaggle workbooks and websites:

https://www.kaggle.com/digvijayyadav/beginners-guide-to-simple-machine-learning

https://www.kaggle.com/purvitsharma/heart-disease-classification

https://www.jeremyjordan.me/evaluating-a-machine-learning-model/

References

VanderPlas, J. (n.d.). Python Data Science Handbook. Retrieved March 19, 2021, from https://jakevdp.github.io/PythonDataScienceHandbook/

Final Project

April 3, 2021

Ingest Data

```
In [195]: #load necessary libraries
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          %matplotlib inline
          import matplotlib.colors as colors
          import seaborn as sns
          import warnings
          warnings.filterwarnings('ignore', category=DeprecationWarning)
          plt.style.use("fivethirtyeight")
          sns.set_style("darkgrid")
          from sklearn.model_selection import train_test_split
          from sklearn.preprocessing import scale
```

```
In [150]: #read the dataset into the notebook
          df = pd.read_csv('heart.csv')
```

```
In [151]: #Dataframe size
          print("Rows:", len(df))
          print("Columns:", df.shape[1])
```

```
Rows: 303
Columns: 14
```

```
In [152]: #Dataframe type
          type(df)
```

```
Out[152]: pandas.core.frame.DataFrame
```

Exploratory Data Analysis

```
In [201]: #Look at the first 5 rows to learn more about the data
          df.head()
```

```
Out[201]:    age  sex  trestbps  chol  fbs  thalach  exang  oldpeak  ca  target  ...    \
          0   63    1       145   233    1      150      0      2.3   0       1  ...
          1   37    1       130   250    0      187      0      3.5   0       1  ...
          2   41    0       130   204    0      172      0      1.4   0       1  ...
          3   56    1       120   236    0      178      0      0.8   0       1  ...
          4   57    0       120   354    0      163      1      0.6   0       1  ...

             restecg_0  restecg_1  restecg_2  slope_0  slope_1  slope_2  thal_0  thal_1  \
          0          1          0          0        1        0        0       0       1
          1          0          1          0        1        0        0       0       0
          2          1          0          0        0        0        1       0       0
          3          0          1          0        0        0        1       0       0
          4          0          1          0        0        0        1       0       0

             thal_2  thal_3
          0       0       0
          1       1       0
          2       1       0
          3       1       0
          4       1       0

          [5 rows x 24 columns]
```

There are 13 attributes and one target variable. From the description of the dataset on Kaggle...

1. age
2. sex
3. cp: chest pain type (4 values)
4. trestbps: resting systolic blood pressure
5. chol: serum cholestoral in mg/dl
6. fbs: fasting blood sugar > 120 mg/dl
7. restecg: resting electrocardiographic results (values 0,1,2)
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina
10. oldpeak: ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
12. cs: number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. (target variable) target: 0 = no heart disease; 1 = heart disease

Additional description of each variable...

age, Float

sex - Category 0 = female 1 = male

cp, chest pain, Category 1 = typical angina, 2 = atypical angina, 3 = non-anginal pain, 4 = asymptomatic

restbp, resting blood pressure (in mm Hg), Float

chol, serum cholesterol in mg/dl, Float

fbs, fasting blood sugar, Category 0 = >=120 mg/dl 1 = <120 mg/dl

restecg, resting electrocardiographic results, Category 1 = normal 2 = having ST-T wave abnormality 3 = showing probable or definite left ventricular hypertrophy

thalach, maximum heart rate achieved, Float

exang, exercise induced angina, Category 0 = no 1 = yes

oldpeak, ST depression induced by exercise relative to rest. Float

slope, the slope of the peak exercise ST segment, Category 1 = upsloping 2 = flat 3 = downsloping

ca, The number of major blood vessels(0-3) supplying blood to heart blocked, Float

thal, thalium heart scan, Category 3 = normal (no cold spots) 6 = fixed defect (cold spots during rest and exercise) 7 = reversible defect (when cold spots only appear during exercise)

(target) (predicted attribute): diagnosis of heart disease (angiographic disease status) — Value 0: < 50% diameter narrowing — Value 1: > 50% diameter narrowing

In [154]:
```
df.dtypes
```

Out[154]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

In [155]:
```
#Let's look at some basic statistics
df.describe()
```

Out[155]:
```
              age         sex          cp    trestbps        chol         fbs  \
count  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
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000
```

```
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000
75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

Data Cleaning and Preparation

In [156]:
```python
# identify any missing values
df.isnull().sum()
```

Out[156]:
```
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 [157]:
```python
# identify any NA values
df.isna().sum()
```

Out[157]:
```
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 [158]:
```
#check for duplicates
df.duplicated().sum()
```

Out[158]: 1

In [159]:
```
#drop duplicates, then check again
df.drop_duplicates(inplace=True)
df.duplicated().sum()
```

Out[159]: 0

In [160]:
```
#how many subjects of each gender? (0=female, 1=male)
df_target = df.groupby("sex").size()
df_target
```

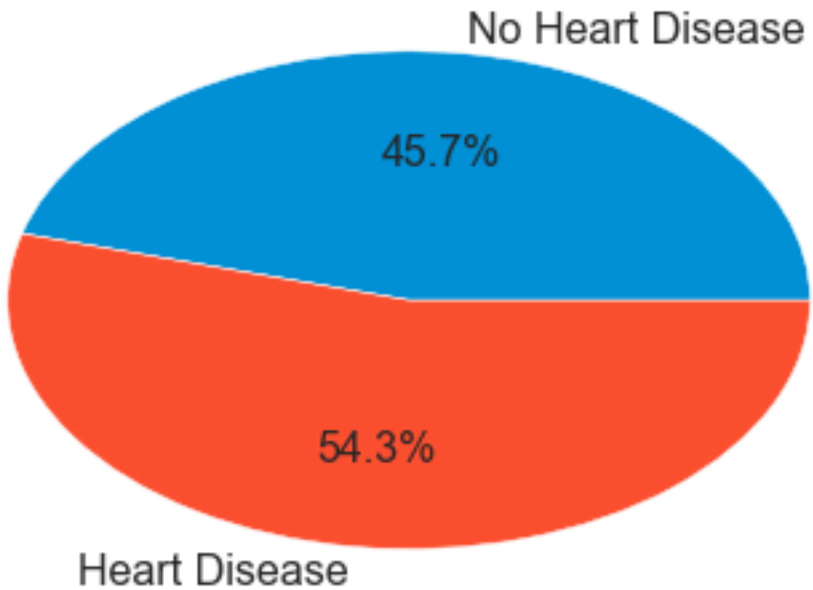
Out[160]:
```
sex
0     96
1    206
dtype: int64
```

In [161]:
```
#how many subjects have heart disease? (0=no, 1=yes)
df_target = df.groupby("target").size()
df_target
```

Out[161]:
```
target
0    138
1    164
dtype: int64
```

Data Visualization

In [162]:
```
plt.pie(df_target.values, labels = ["No Heart Disease", "Heart Disease"], autopct='%
plt.show()
```

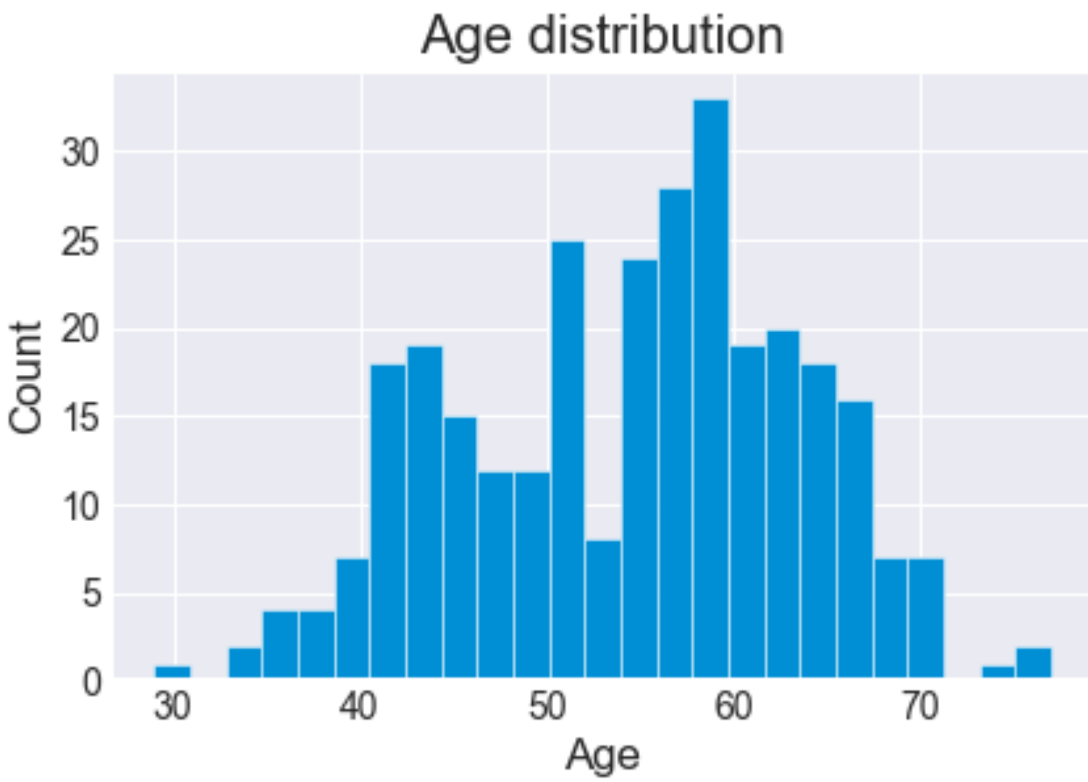



```
In [163]: #what are the age ranges, mean, and distribution?
          minAge=min(df.age)
          maxAge=max(df.age)
          meanAge=df.age.mean()
          print('Min Age :', minAge)
          print('Max Age :', maxAge)
          print("Mean Age : "+"{:.2f}".format(meanAge));
```

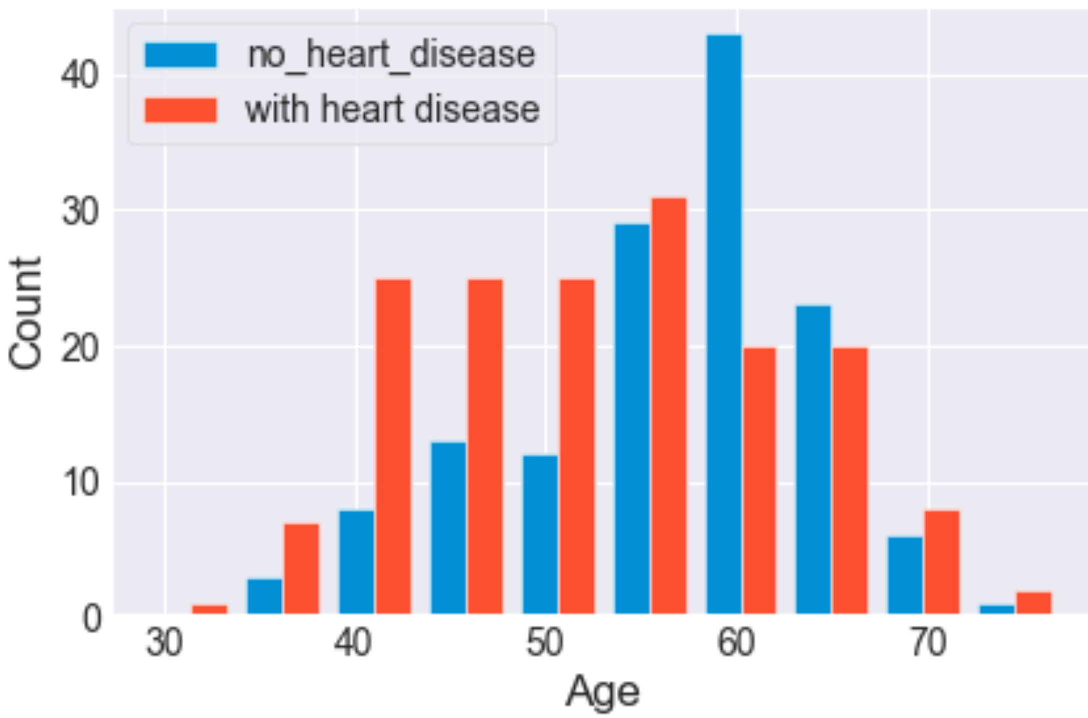
```
Min Age : 29
Max Age : 77
Mean Age : 54.42
```

```
In [164]: plt.hist(df['age'], bins = 25)
          plt.xlabel('Age')
          plt.ylabel('Count')
          plt.title('Age distribution')
```

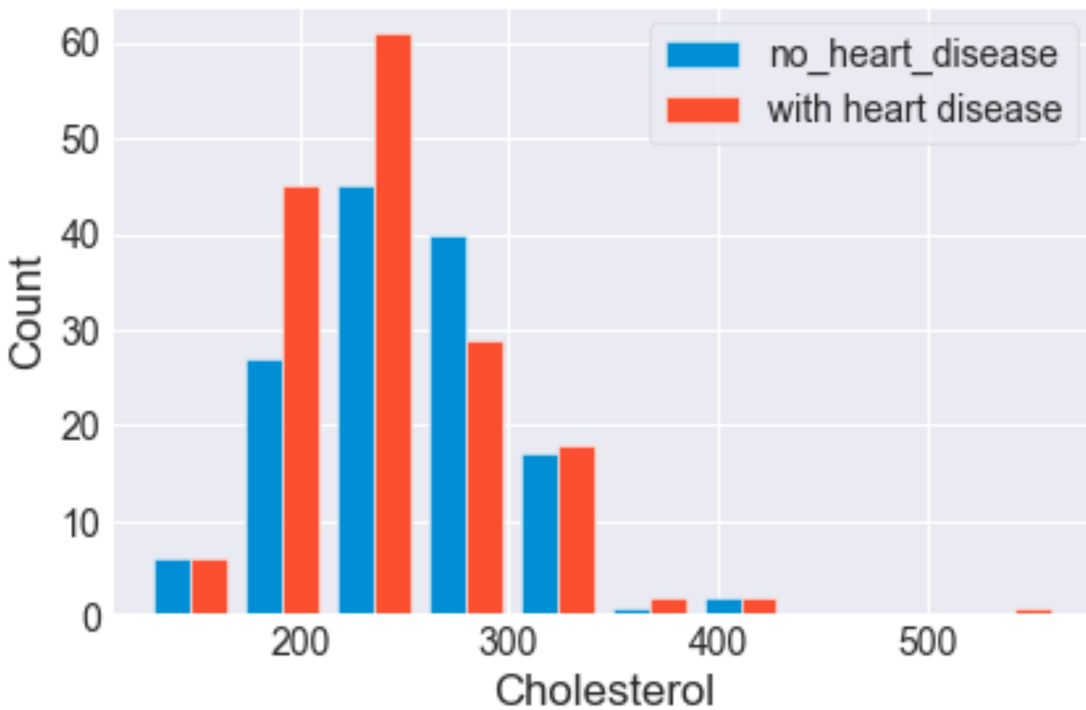
```
Out[164]: Text(0.5,1,'Age distribution')
```



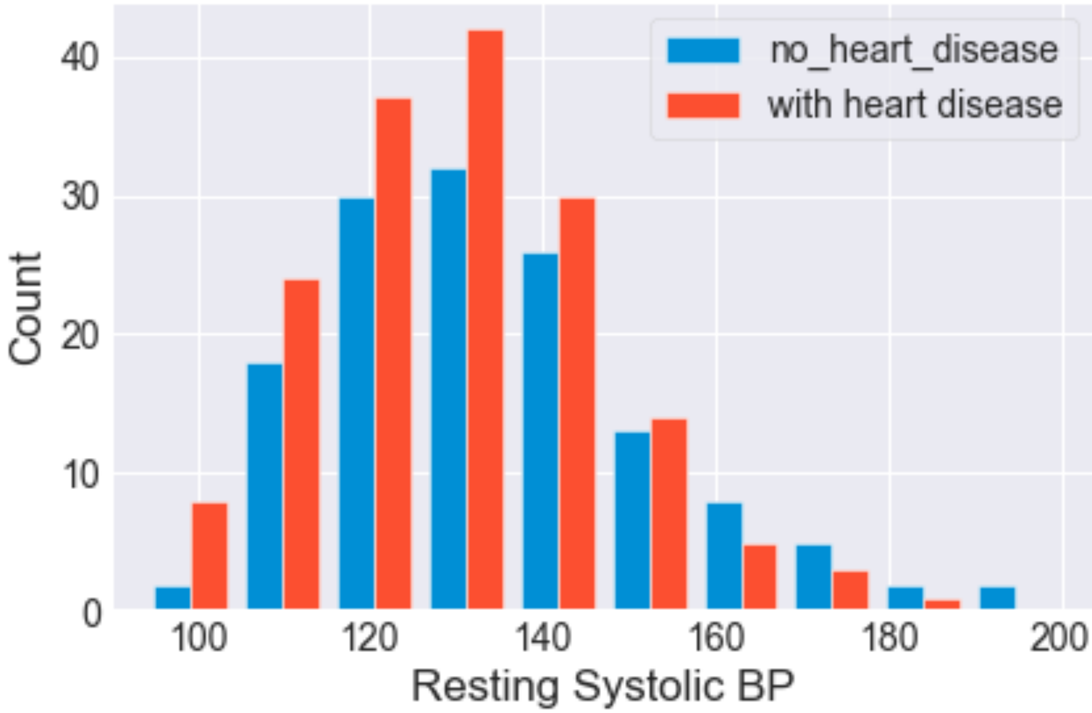
In [165]:
```
#distribution of heart disease by age
plt.hist([df[df.target==0].age, df[df.target==1].age], bins = 10, label = ["no_heart_
plt.xlabel("Age")
plt.ylabel("Count")
plt.legend()
plt.show()
```



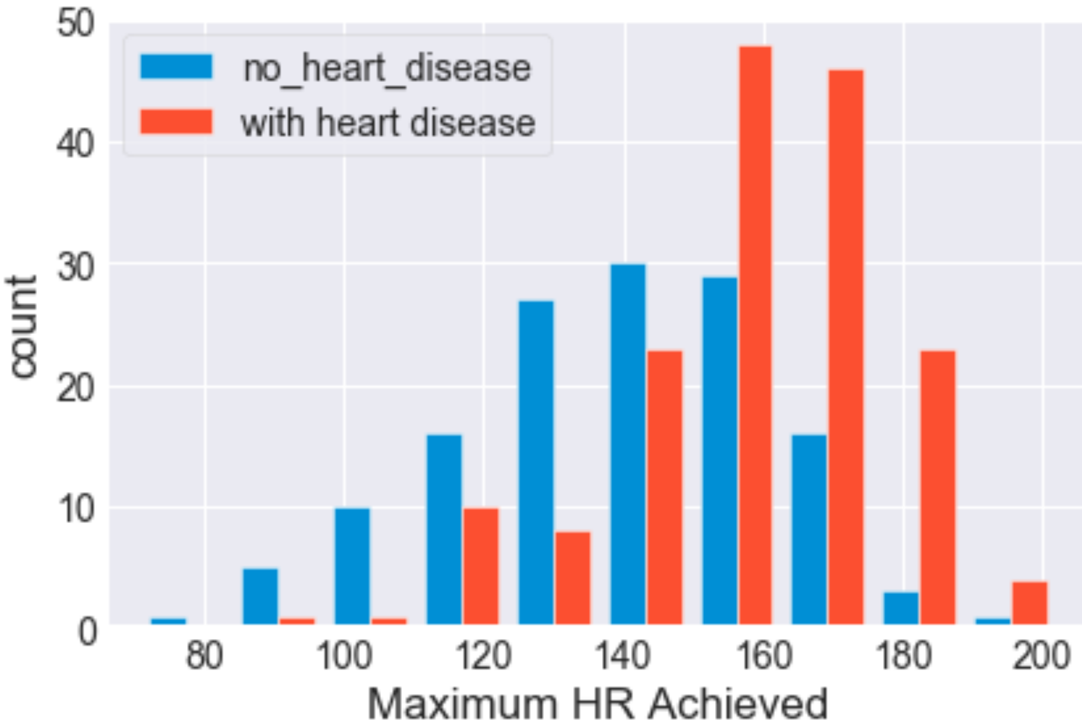

```
In [166]: #distribution of heart disease by cholesterol levels
          plt.hist([df[df.target==0].chol, df[df.target==1].chol], bins = 10, label = ["no_hea
          plt.xlabel("Cholesterol")
          plt.ylabel("Count")
          plt.legend()
          plt.show()
```



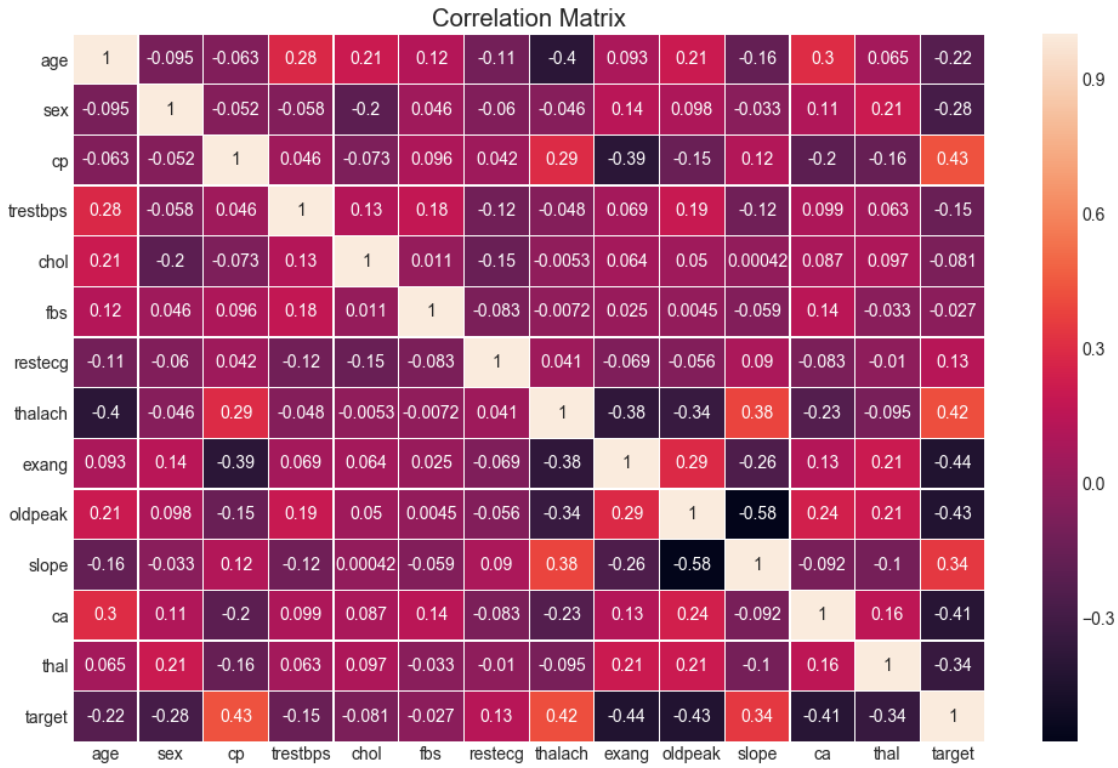
In [196]:
```
#distribution of heart disease by resting blood pressure
plt.hist([df[df.target==0].trestbps, df[df.target==1].trestbps], bins = 10, label =
plt.xlabel("Resting Systolic BP")
plt.ylabel("Count")
plt.legend()
plt.show()
```




```
In [197]: #distribution of heart disease by maximum heart rate achieved
          plt.hist([df[df.target==0].thalach, df[df.target==1].thalach], bins = 10, label = ["
          plt.xlabel("Maximum HR Achieved")
          plt.ylabel("count")
          plt.legend()
          plt.show()
```




```
In [169]: #Correlation Matrix
          plt.figure(figsize=(15,10))
          sns.heatmap(df.corr(), annot=True, linewidth=0.2)
          plt.title("Correlation Matrix")
          plt.show()
```



In [170]:
```
#Correlation with target
df.corr()['target'].sort_values(ascending=False)
```

Out[170]:
```
target      1.000000
cp          0.432080
thalach     0.419955
slope       0.343940
restecg     0.134874
fbs        -0.026826
chol       -0.081437
trestbps   -0.146269
age        -0.221476
sex        -0.283609
thal       -0.343101
ca         -0.408992
oldpeak    -0.429146
exang      -0.435601
Name: target, dtype: float64
```

Chest pain is most positively correlated with the target, and exercised induced angina is most negatively correlated with the target

Machine Learning

The next step is to implement some machine learning algorithms to see if we can predict the target variable of heart disease based on the 13 input variables. This is a binary supervised classification problem.

```
In [173]: #Convert categorical variables into indicator (dummy) variables
          cp_1 = pd.get_dummies(df['cp'], prefix = "cp")
          restecg_1 = pd.get_dummies(df['restecg'], prefix = "restecg")
          slope_1 = pd.get_dummies(df['slope'], prefix = "slope")
          thal_1 = pd.get_dummies(df['thal'], prefix = "thal")
```

```
In [174]: frames = [df, cp_1, restecg_1, slope_1, thal_1]
          df = pd.concat(frames, axis = 1)
          df.head()
```

```
Out[174]:    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
          0   63    1   3       145   233    1        0      150      0      2.3
          1   37    1   2       130   250    0        1      187      0      3.5
          2   41    0   1       130   204    0        0      172      0      1.4
          3   56    1   1       120   236    0        1      178      0      0.8
          4   57    0   0       120   354    0        1      163      1      0.6

              ...    restecg_0  restecg_1  restecg_2  slope_0  slope_1  slope_2  thal_0  \
          0   ...            1          0          0        1        0        0       0
          1   ...            0          1          0        1        0        0       0
          2   ...            1          0          0        0        0        1       0
          3   ...            0          1          0        0        0        1       0
          4   ...            0          1          0        0        0        1       0

             thal_1  thal_2  thal_3
          0       1       0       0
          1       0       1       0
          2       0       1       0
          3       0       1       0
          4       0       1       0

          [5 rows x 28 columns]
```

```
In [175]: #Then remove the original columns
          df = df.drop(columns = [ 'cp', 'restecg', 'slope', 'thal'])
          df.head()
```

```
Out[175]:    age  sex  trestbps  chol  fbs  thalach  exang  oldpeak  ca  target   ...    \
          0   63    1       145   233    1      150      0      2.3   0       1   ...
          1   37    1       130   250    0      187      0      3.5   0       1   ...
          2   41    0       130   204    0      172      0      1.4   0       1   ...
          3   56    1       120   236    0      178      0      0.8   0       1   ...
          4   57    0       120   354    0      163      1      0.6   0       1   ...

             restecg_0  restecg_1  restecg_2  slope_0  slope_1  slope_2  thal_0  thal_1  \
```

```
0          1          0          0          1          0          0          0          1
1          0          1          0          1          0          0          0          0
2          1          0          0          0          0          1          0          0
3          0          1          0          0          0          1          0          0
4          0          1          0          0          0          1          0          0

   thal_2  thal_3
0       0       0
1       1       0
2       1       0
3       1       0
4       1       0

[5 rows x 24 columns]
```

In [176]:
```python
#Separate the target variable column from the dataframe
y = df.target.values
X = df.drop(columns="target")
```

In [177]:
```python
#Split data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
```

In [178]:
```python
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(241, 23)
(241,)
(61, 23)
(61,)
```

In [179]:
```python
# Normalize the Data
from sklearn.preprocessing import Normalizer
scaler = Normalizer()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Model Selection

1. Logistic Regression
2. Support Vector Machine
3. Decision Tree
4. Naive Bayes
5. Random Forest Classifier

In [180]:
```python
#Logistic Regression
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
model.fit(X_train,y_train)
y_pred = model.predict(X_test)

from sklearn.metrics import classification_report , accuracy_score
report = classification_report(y_test,y_pred)
print(report)
print(accuracy_score(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.48      0.62        29
           1       0.67      0.94      0.78        32

    accuracy                           0.72        61
   macro avg       0.77      0.71      0.70        61
weighted avg       0.77      0.72      0.70        61

0.7213114754098361
```

In [181]:
```
#Support Vector Machine Classifier
from sklearn.svm import SVC
model2 = SVC(verbose=True)
model2.fit(X_train,y_train)
y_pred2 = model2.predict(X_test)
report_svc = classification_report(y_test,y_pred2)
print(report_svc)
print(accuracy_score(y_test,y_pred2))
```

```
[LibSVM]              precision    recall  f1-score   support

           0       0.85      0.59      0.69        29
           1       0.71      0.91      0.79        32

    accuracy                           0.75        61
   macro avg       0.78      0.75      0.74        61
weighted avg       0.78      0.75      0.75        61

0.7540983606557377
```

In [182]:
```
#Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier(criterion="entropy",max_depth=200)
model3.fit(X_train,y_train)
y_pred3 = model3.predict(X_test)
report_dtree = classification_report(y_pred3,y_test)
```

```
print(report_dtree)
print(accuracy_score(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        29
           1       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

0.8360655737704918
```

In [183]:
```
#Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
report_nb = classification_report(y_pred_nb,y_test)
print(report_nb)
acc = nb.score(X_test,y_test)*100
print(acc)
```

```
              precision    recall  f1-score   support

           0       0.90      0.74      0.81        35
           1       0.72      0.88      0.79        26

    accuracy                           0.80        61
   macro avg       0.81      0.81      0.80        61
weighted avg       0.82      0.80      0.80        61

80.32786885245902
```

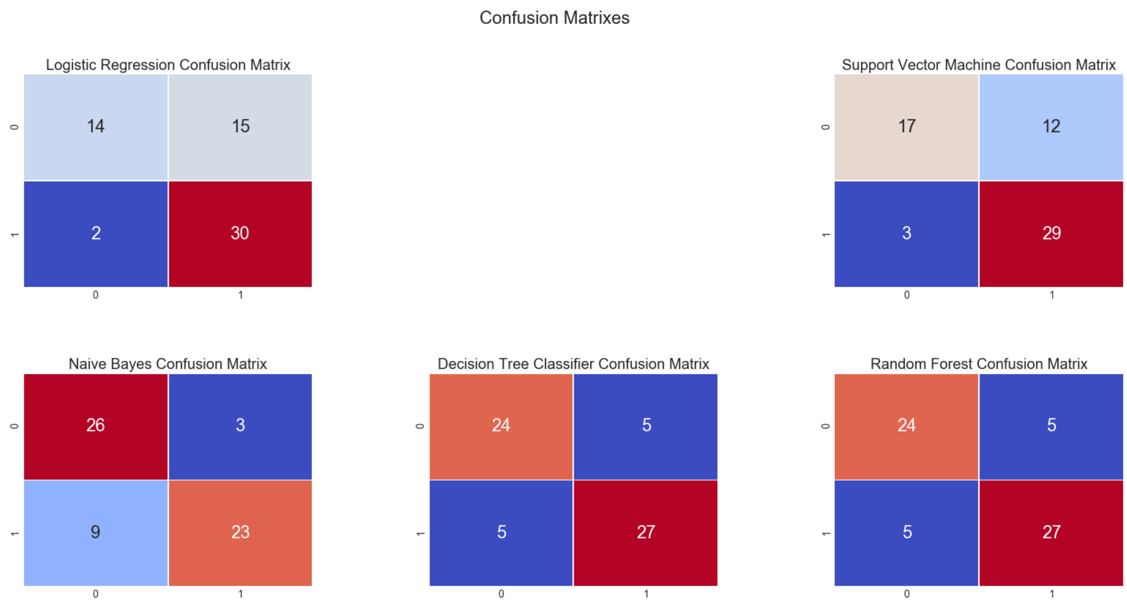
In [184]:
```
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
model_rf = RandomForestClassifier(n_estimators=1000, random_state=1)
model_rf.fit(X_train,y_train)
y_pred_rf = model_rf.predict(X_test)
report_rf2 = classification_report(y_test,y_pred_rf)
print(report_rf2)
print(accuracy_score(y_pred_rf,y_test))
```

```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        29
```

```
           1       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

0.8360655737704918
```

In [185]:
```python
#Confusion Matrixes
from sklearn.metrics import confusion_matrix
cm_lr = confusion_matrix(y_test,y_pred)
cm_svc = confusion_matrix(y_test,y_pred2)
cm_dtree = confusion_matrix(y_test,y_pred3)
cm_rf = confusion_matrix(y_test,y_pred_rf)
cm_nb = confusion_matrix(y_test,y_pred_nb)
```

In [194]:
```python
plt.figure(figsize=(24,12))

plt.suptitle("Confusion Matrixes",fontsize=24)
plt.subplots_adjust(wspace = 0.4, hspace= 0.4)

plt.subplot(2,3,1)
plt.title("Logistic Regression Confusion Matrix")
sns.heatmap(cm_lr,annot=True, cmap="coolwarm", linewidth=1, fmt="d",cbar=False, anno

plt.subplot(2,3,3)
plt.title("Support Vector Machine Confusion Matrix")
sns.heatmap(cm_svc,annot=True, cmap="coolwarm",linewidth=1, fmt="d",cbar=False, anno

plt.subplot(2,3,4)
plt.title("Naive Bayes Confusion Matrix")
sns.heatmap(cm_nb,annot=True, cmap="coolwarm",linewidth=1, fmt="d",cbar=False, annot_

plt.subplot(2,3,5)
plt.title("Decision Tree Classifier Confusion Matrix")
sns.heatmap(cm_dtree,annot=True, cmap="coolwarm",linewidth=1, fmt="d",cbar=False, ann

plt.subplot(2,3,6)
plt.title("Random Forest Confusion Matrix")
sns.heatmap(cm_rf,annot=True, cmap="coolwarm",linewidth=1, fmt="d",cbar=False, annot_

plt.show()
```


Confusion Matrixes
Logistic Regression Confusion Matrix
14
15
2
30
Support Vector Machine Confusion Matrix
17
12
3
29
Naive Bayes Confusion Matrix
26
3
9
23
Decision Tree Classifier Confusion Matrix
24
5
5
27
Random Forest Confusion Matrix
24
5
5
27
0
1

```
In [ ]:
```