health care

August 1, 2023

[4]:
```
import pandas as pd
import seaborn as sns
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

[5]:
```
import warnings
warnings.filterwarnings('ignore')
```

[6]:
```
data= pd.read_csv('heart.csv')
```

[7]:
```
data.head()
```

[7]:

	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

[8]:
```
data.tail()
```

[8]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak
298	57	0	0	140	241	0	1	123	1	0.2
299	45	1	3	110	264	0	1	132	0	1.2
300	68	1	0	144	193	1	1	141	0	3.4
301	57	1	0	130	131	0	1	115	1	1.2
302	57	0	1	130	236	0	0	174	0	0.0

```
     slope  ca  thal  target
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0
```

[9]:
```
data.shape
```

[9]:
```
(303, 14)
```

[10]:
```
#returns the number of unique values for each variable.
data.nunique(axis=0)
```

[10]:
```
age          41
sex           2
cp            4
trestbps     49
chol        152
fbs           2
restecg       3
thalach      91
exang         2
oldpeak      40
slope         3
ca            5
thal          4
target        2
dtype: int64
```

[11]:
```
data['target'].value_counts()
```

[11]:
```
1    165
0    138
Name: target, dtype: int64
```

1 we have a good balance between the two binary outputs

[12]:
```
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
```

[13]:
```
# Display the Missing Values
data.isnull().sum()
```

[13]:
```
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
```

[14]:
```
# decriptive statistics
data.describe()
```

[14]:

	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	

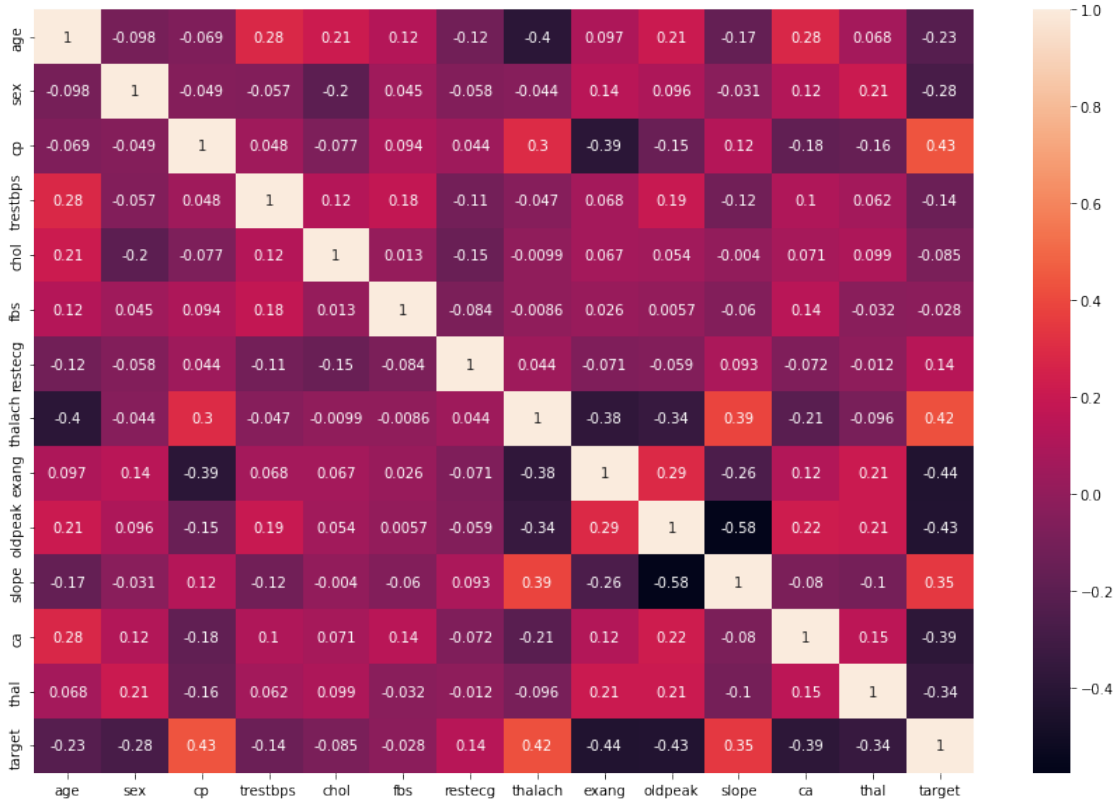
```
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

          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
```

[15]:
```
#correlations between all variables.
correlation = data.corr()
plt.subplots(figsize=(15,10))
sns.heatmap(correlation,annot = True)
```

[15]:
```
<AxesSubplot:>
```



[16]:
```
correlation['target'].abs().sort_values(ascending=False)
```

[16]:
```
target      1.000000
exang       0.436757
cp          0.433798
oldpeak     0.430696
thalach     0.421741
ca          0.391724
slope       0.345877
thal        0.344029
sex         0.280937
age         0.225439
trestbps    0.144931
restecg     0.137230
chol        0.085239
fbs         0.028046
Name: target, dtype: float64
```

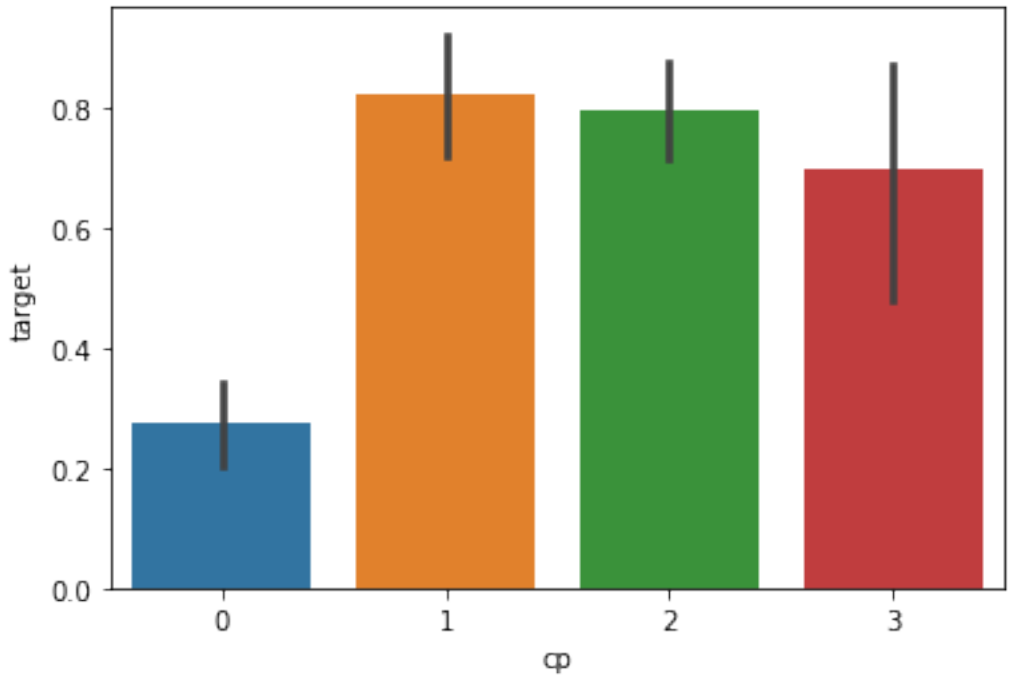
1.1 We can see there is a positive correlation between chest pain (cp) & target (DEPENDENT VARIABLE).

Cp (chest pain), is a ordinal feature with 4 values: Value 1: typical angina , Value 2: atypical angina, Value 3: non-anginal pain , Value 4: asymptomatic

[17]:
```
data['cp'].unique()
```

[17]:
```
array([3, 2, 1, 0])
```

[18]:
```
## Heart Disease Frequency vs Chest Pain
sns.barplot(data["cp"],data['target'])
plt.show()
```

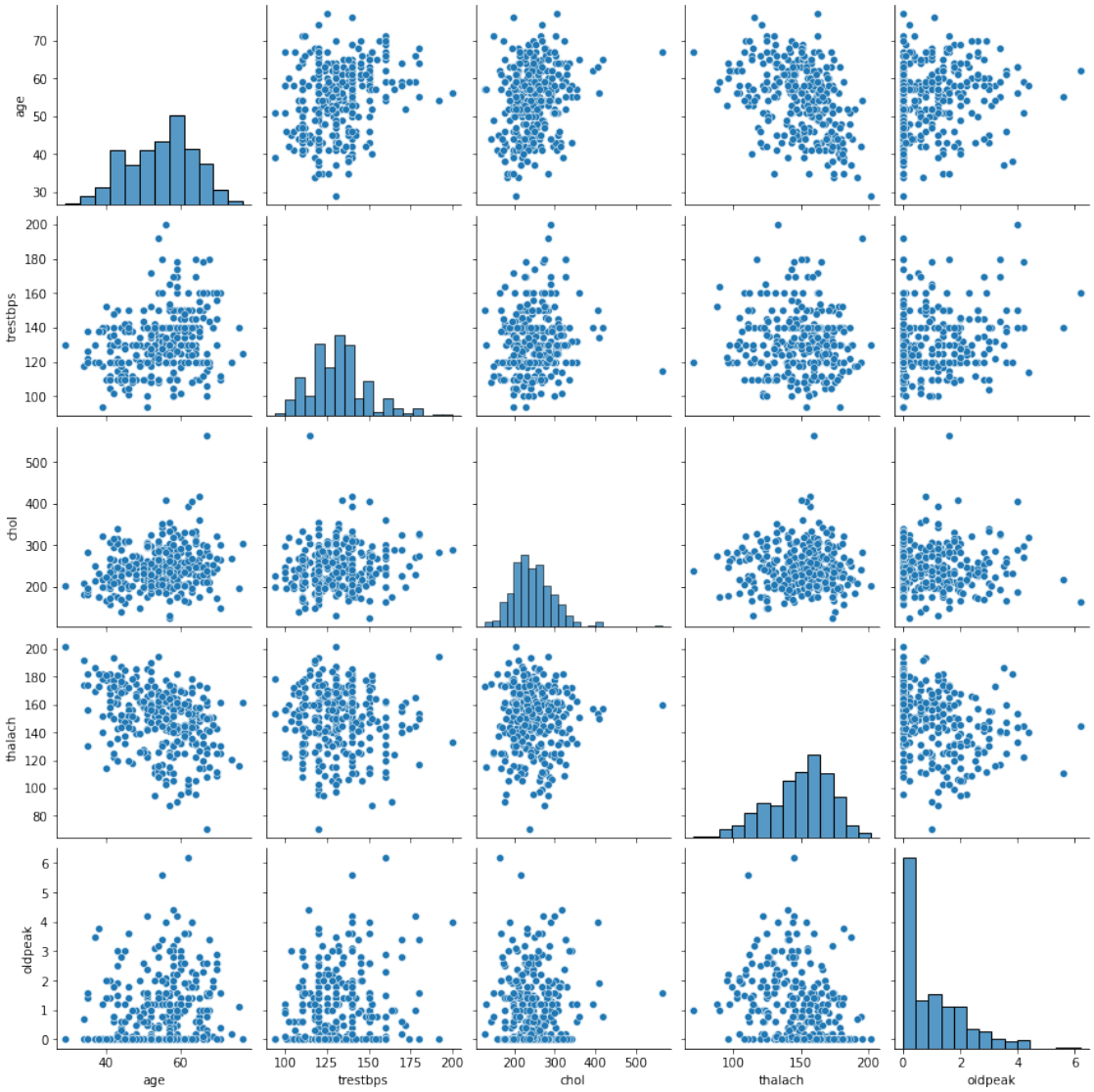


[19]:
```
#we see a negative correlation  between typical angina (exang) & our predictor.
```

[20]:
```
#the correlations between all variables(only continuous columns from data)
subData = data[['age','trestbps','chol','thalach','oldpeak']]

sns.pairplot(subData)
```

[20]:
```
<seaborn.axisgrid.PairGrid at 0x7f26455606d0>
```



[ ]:

2 Data Processing

We need to convert some categorical variables into dummy variables and scale all the values before training the Machine Learning models. We will use the get_dummies method to create dummy columns for categorical variables.

[21]:
```
data.describe()
```

[21]:
```
              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
```

[22]:
```
#segregarting the categorical variables and continuous ones
categorical = []
continous = []
for column in data.columns:
    if len(data[column].unique()) <= 10:
        categorical.append(column)
    else:
        continous.append(column)
```

[23]:
```
categorical.remove('target')
data1 = pd.get_dummies(data, columns = categorical)
```

[24]:
```
data1.head()
```

[24]:
```
   age  trestbps  chol  thalach  oldpeak  target  sex_0  sex_1  cp_0  cp_1  \
0   63       145   233      150      2.3       1      0      1     0     0
1   37       130   250      187      3.5       1      0      1     0     0
```

```
2   41       130   204      172      1.4       1      1      0     0     1
3   56       120   236      178      0.8       1      0      1     0     1
4   57       120   354      163      0.6       1      1      0     1     0

   …  slope_2  ca_0  ca_1  ca_2  ca_3  ca_4  thal_0  thal_1  thal_2  thal_3
0  …        0     1     0     0     0     0       0       1       0       0
1  …        0     1     0     0     0     0       0       0       1       0
2  …        1     1     0     0     0     0       0       0       1       0
3  …        1     1     0     0     0     0       0       0       1       0
4  …        1     1     0     0     0     0       0       0       1       0

[5 rows x 31 columns]
```

[25]:
```
#standardizing the data
from sklearn.preprocessing import StandardScaler

std = StandardScaler()
scaled_columns = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
data1[scaled_columns] = std.fit_transform(data1[scaled_columns])
```

[26]:
```
data1.head()
```

[26]:
```
        age  trestbps      chol   thalach   oldpeak  target  sex_0  sex_1  \
0  0.952197  0.763956 -0.256334  0.015443  1.087338       1      0      1
1 -1.915313 -0.092738  0.072199  1.633471  2.122573       1      0      1
2 -1.474158 -0.092738 -0.816773  0.977514  0.310912       1      1      0
3  0.180175 -0.663867 -0.198357  1.239897 -0.206705       1      0      1
4  0.290464 -0.663867  2.082050  0.583939 -0.379244       1      1      0

   cp_0  cp_1  …  slope_2  ca_0  ca_1  ca_2  ca_3  ca_4  thal_0  thal_1  \
0     0     0  …        0     1     0     0     0     0       0       1
1     0     0  …        0     1     0     0     0     0       0       0
2     0     1  …        1     1     0     0     0     0       0       0
3     0     1  …        1     1     0     0     0     0       0       0
4     1     0  …        1     1     0     0     0     0       0       0

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

[5 rows x 31 columns]
```

[27]:
```
data1.columns
```

[27]:
```
Index(['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'target', 'sex_0',
       'sex_1', 'cp_0', 'cp_1', 'cp_2', 'cp_3', 'fbs_0', 'fbs_1', 'restecg_0',
       'restecg_1', 'restecg_2', 'exang_0', 'exang_1', 'slope_0', 'slope_1',
       'slope_2', 'ca_0', 'ca_1', 'ca_2', 'ca_3', 'ca_4', 'thal_0', 'thal_1',
       'thal_2', 'thal_3'],
      dtype='object')
```

[158]:
```
#Prepare Data for Modeling
from sklearn.model_selection import train_test_split
X= data1.drop("target",axis=1)
y=data1['target']
```

[29]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪3,random_state=42)
```

[159]:
```
```

[159]:
```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

[30]:
```
#Train various Classification Models on the Training set & see which yields the␣
↪highest accuracy(supervised learning models)
#will compare the accuracy of Logistic Regression, K-NN (k-Nearest Neighbours),
#SVM (Support Vector Machine), Naives Bayes Classifier, Decision Trees, Random␣
↪Forest, and XGBoost.
```

[31]:
```
from sklearn.metrics import␣
↪confusion_matrix,classification_report,accuracy_score
```

3 Model 1 : XGBoost

[32]:
```
import xgboost as xgb
```

[33]:
```
model1 = xgb.XGBClassifier(random_state=42)
model1.fit(X_train,y_train)
```

[33]:
```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=42,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method=None, validate_parameters=False, verbosity=None)
```

[34]:
```
predict1 = model1.predict(X_test)
```

[35]:
```
print(confusion_matrix(y_test,predict1))
```

```
[[33  8]
 [ 8 42]]
```

[36]:
```
print(classification_report(y_test,predict1))
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        41
           1       0.84      0.84      0.84        50

    accuracy                           0.82        91
   macro avg       0.82      0.82      0.82        91
weighted avg       0.82      0.82      0.82        91
```

4 Model2 : Logistic Regression

[37]:
```
from sklearn.linear_model import LogisticRegression
```

[38]:
```
model2 = LogisticRegression(random_state=42)
```

[39]:
```
model2.fit(X_train,y_train)
```

[39]:
```
LogisticRegression(random_state=42)
```

[40]:
```
predict2 = model2.predict(X_test)
```

[41]:
```
print(confusion_matrix(y_test,predict2))
```

```
[[33  8]
 [ 5 45]]
```

[42]:
```
print(classification_report(y_test,predict2))
```

```
              precision    recall  f1-score   support

           0       0.87      0.80      0.84        41
           1       0.85      0.90      0.87        50

    accuracy                           0.86        91
   macro avg       0.86      0.85      0.85        91
weighted avg       0.86      0.86      0.86        91
```

5 Modle3 : K-NN (K-Nearest Neighbors)

[43]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[44]:
```
model3 = KNeighborsClassifier()
```

[45]:
```
model3.fit(X_train,y_train,)
```

[45]:
```
KNeighborsClassifier()
```

[46]:
```
predict3 = model3.predict(X_test)
```

[47]:
```
print(confusion_matrix(y_test,predict3))
```

```
[[35  6]
 [ 6 44]]
```

[48]:
```
print(classification_report(y_test,predict3))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        41
           1       0.88      0.88      0.88        50

    accuracy                           0.87        91
   macro avg       0.87      0.87      0.87        91
weighted avg       0.87      0.87      0.87        91
```

6 Model4 : Naives Bayes Classifier

[49]:
```
from sklearn.naive_bayes import GaussianNB
```

[50]:
```
modle4 = GaussianNB()
```

[51]:
```
modle4.fit(X_train,y_train)
```

[51]:
```
GaussianNB()
```

[52]:
```
predict4= modle4.predict(X_test)
```

[53]:
```
print(confusion_matrix(y_test,predict4))
```

```
[[37  4]
 [25 25]]
```

[54]:
```
print(classification_report(y_test,predict4))
```

```
              precision    recall  f1-score   support

           0       0.60      0.90      0.72        41
           1       0.86      0.50      0.63        50

    accuracy                           0.68        91
   macro avg       0.73      0.70      0.68        91
weighted avg       0.74      0.68      0.67        91
```

7 Model5 : SVM (Support Vector Machine)

[55]:
```
# importing support vector classifier
from sklearn.svm import SVC
```

[ ]:
```

```

[123]:
```
model5 = SVC(random_state=100)
```

[ ]:
```

```

[124]:
```
model5.fit(X_train,y_train)
```

[124]:
```
SVC(random_state=100)
```

[125]:
```
predict5 = model5.predict(X_test)
```

[126]:
```
print(confusion_matrix(y_test,predict5))
```

```
[[36  5]
 [ 6 44]]
```

[127]:
```
print(classification_report(y_test,predict5))
```

```
              precision    recall  f1-score   support

           0       0.86      0.88      0.87        41
           1       0.90      0.88      0.89        50

    accuracy                           0.88        91
   macro avg       0.88      0.88      0.88        91
weighted avg       0.88      0.88      0.88        91
```

8 Model6 : Decision Trees

[61]:
```
from sklearn.tree import DecisionTreeClassifier
```

[62]:
```
model6= DecisionTreeClassifier(random_state=42)
```

[63]:
```
model6.fit(X_train,y_train)
```

[63]:
```
DecisionTreeClassifier(random_state=42)
```

[64]:
```
predict6 = model6.predict(X_test)
```

[65]:
```
print(confusion_matrix(y_test,predict6))
```

```
[[34  7]
 [13 37]]
```

[66]:
```
print(classification_report(y_test,predict6))
```

```
              precision    recall  f1-score   support

           0       0.72      0.83      0.77        41
           1       0.84      0.74      0.79        50

    accuracy                           0.78        91
   macro avg       0.78      0.78      0.78        91
weighted avg       0.79      0.78      0.78        91
```

9 Model7 : Random Forest

[67]:
```
from sklearn.ensemble import RandomForestClassifier
```

[68]:
```
model7=RandomForestClassifier(random_state=42)
```

[69]:
```
model7.fit(X_train,y_train)
```

[69]:
```
RandomForestClassifier(random_state=42)
```

[70]:
```
predict7 = model7.predict(X_test)
```

[71]:
```
print(confusion_matrix(y_test,predict7))
```

```
[[33  8]
 [ 8 42]]
```

[72]:
```
print(classification_report(y_test,predict7))
```

```
              precision    recall  f1-score   support

           0       0.80      0.80      0.80        41
           1       0.84      0.84      0.84        50

    accuracy                           0.82        91
   macro avg       0.82      0.82      0.82        91
weighted avg       0.82      0.82      0.82        91
```

[73]:
```
#From comparing the 7 models,
# SVC yields the highest accuracy. With an accuracy of 88%
# im taking  With an accuracy of 80%.
```

10 Out of the 13 features we examined, the top 2significant features that helped us classify between a positive & negative

11 1- Diagnosis were chest pain type (cp)

12 2-maximum heart rate achieved (thalach)

13

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: