Healthcare

September 13, 2024

[38]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

[39]:
```
#Load the file into dataframe
data =pd.read_excel('data.xlsx')
data.head()
```

[39]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
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	

	ca	thal	target
0	0	1	1
1	0	2	1
2	0	2	1
3	0	2	1
4	0	2	1

[40]:
```
# Information about dataset
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
```

[41]:
```
# To check for null values
data.isnull().sum()
```

[41]:
```
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
```

[42]:
```
# No null values
```

[43]:
```
data.duplicated().sum()
```

[43]:
```
1
```

[44]:
```
data.drop_duplicates(inplace=True)
```

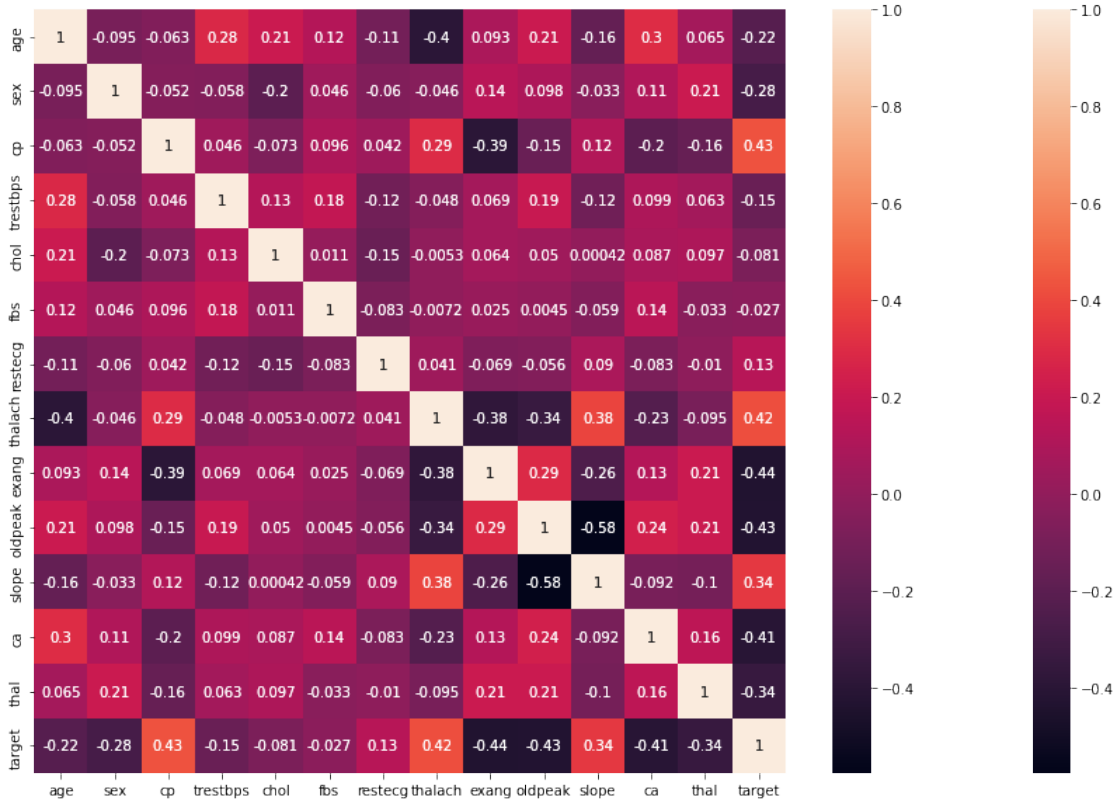
[45]:
```
# To analyze the Statistical Summary
data.describe()
```

[45]:

	age	sex	cp	trestbps	chol	fbs	\
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000	
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686	
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	

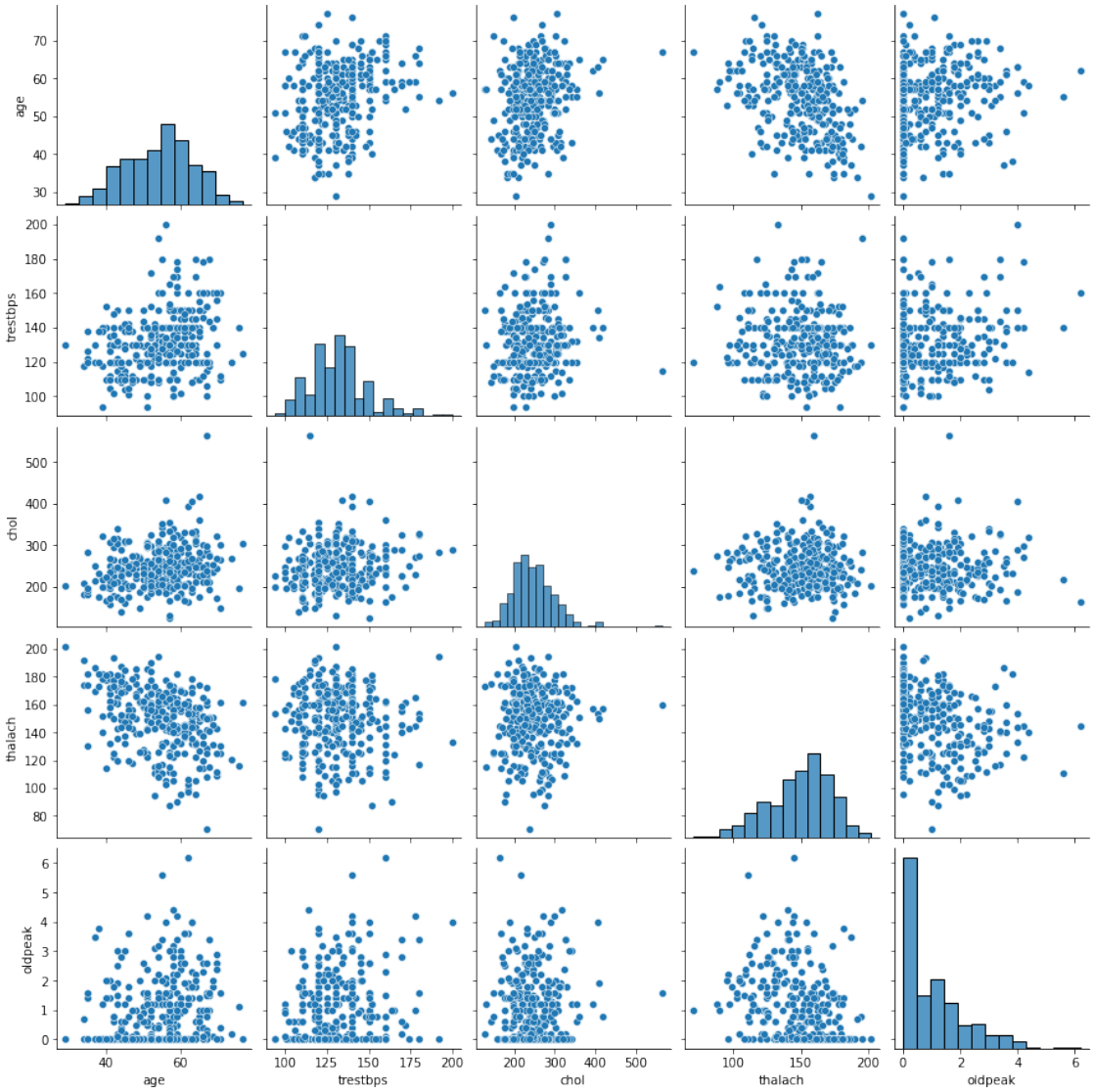
```
75%     61.00000    1.000000    2.000000  140.000000  274.750000    0.000000
max     77.00000    1.000000    3.000000  200.000000  564.000000    1.000000

          restecg     thalach       exang     oldpeak       slope          ca  \
count  302.000000  302.000000  302.000000  302.000000  302.000000  302.000000
mean     0.526490  149.569536    0.327815    1.043046    1.397351    0.718543
std      0.526027   22.903527    0.470196    1.161452    0.616274    1.006748
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000  133.250000    0.000000    0.000000    1.000000    0.000000
50%      1.000000  152.500000    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  302.000000  302.000000
mean     2.314570    0.543046
std      0.613026    0.498970
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
```

[46]:
```
# To check the correlation between variables and identify them calculate␣
 ↪correlation matrix

corr = data.corr()
plt.subplots(figsize=(15,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,␣
 ↪annot=True)
sns.heatmap(corr, xticklabels=corr.columns,yticklabels=corr.columns, annot=True)
plt.show()
```



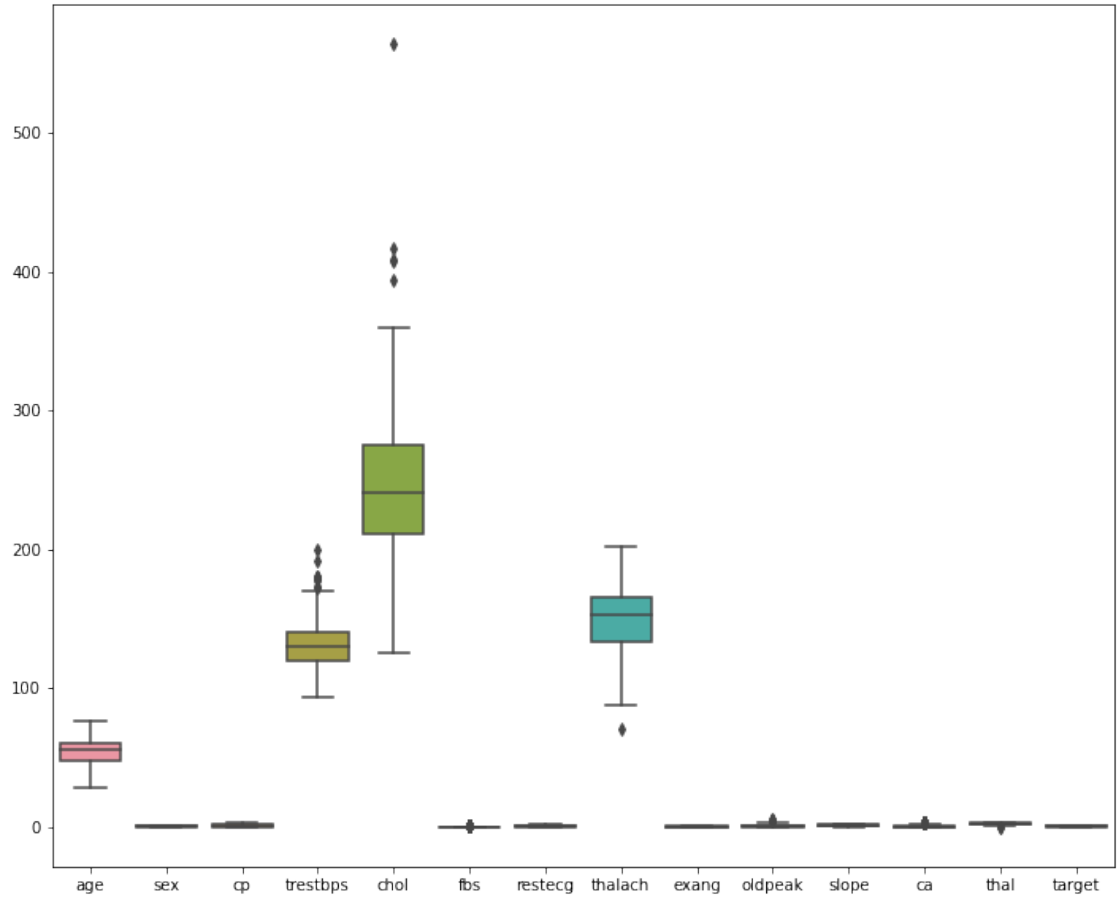
[47]:
```
# High correlation between the chest pain variable and target variable  and a
↪huge negative correlation between exang i.e exercise induced angina which is
↪justified scientifically
```

[48]:
```
# Using pairplots to see the continuous columns variable correlation

data1 = data[['age','trestbps','chol','thalach','oldpeak']]
sns.pairplot(data1)
plt.show()
```



[49]:
```
# Outlier detection
# Since the dataset is not large, we cannot discard the outliers. We will treat␣
↪the outliers as potential observations.
```

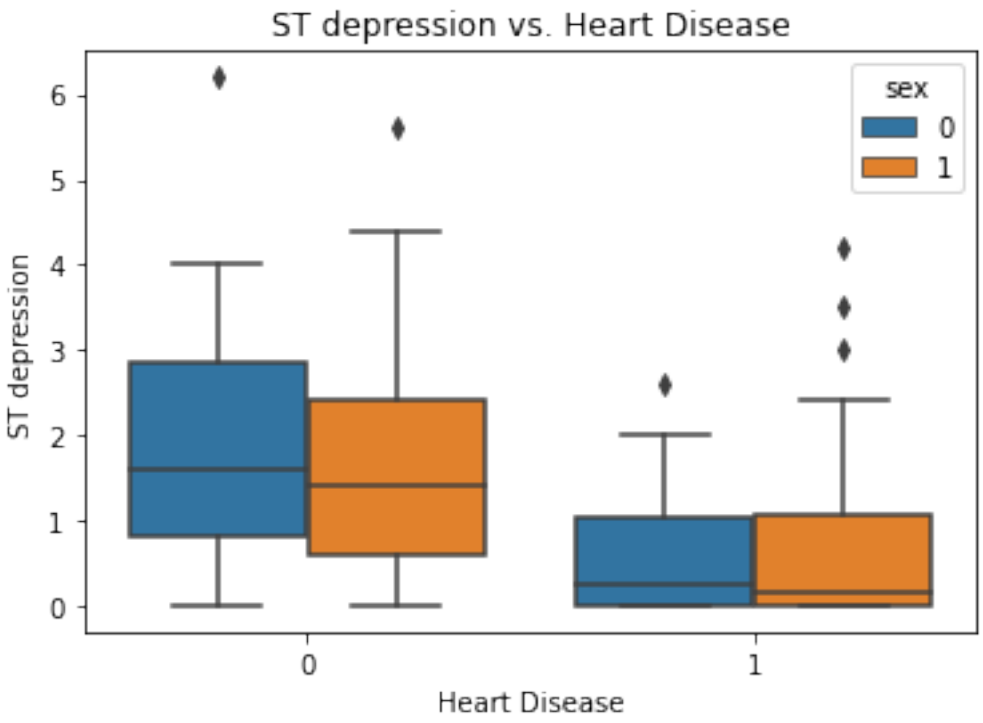
[50]:
```
fig, ax = plt.subplots(figsize=(12,10))
sns.boxplot(data=data, ax=ax)
plt.show()
```



[51]:
```
# Considering our dataset, the response variable target has two outcomes␣
↪"Patients with Heart Disease" and "Patients without Heart Disease". Let us␣
↪now observe their distribution in the dataset.
data['target'].value_counts()
```

[51]:
```
1    164
0    138
Name: target, dtype: int64
```

[52]:
```
sns.boxplot(x="target", y="oldpeak", hue="sex", data=data);

plt.title('ST depression vs. Heart Disease')
plt.xlabel('Heart Disease')
plt.ylabel('ST depression')
plt.show()
```



[53]:
```
# Heart disease Positive patients exhibit a lowered median for ST depression␣
 ↪level, while negative patients have higher levels.
# No much differences between male & female target outcomes, expect for the␣
 ↪fact that males have slightly larger ranges of ST Depression.
```

[54]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

[55]:
```
# Defining X and Y for modeling
X = data.drop("target",axis=1)
y = data["target"]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
 ↪20,stratify=y,random_state=7)
```

[56]:
```
#Normalize: Standardizing the data will transform the data so that its␣
 ↪distribution will have a
#mean of 0 and a standard deviation of 1

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[57]:
```
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

[57]:
```
LogisticRegression()
```

[58]:
```
pred = lr.predict(X_test)
lr.score(X_test,y_test)
lr.score(X_train,y_train)
```

[58]:
```
0.8589211618257261
```

[59]:
```
from sklearn.metrics import accuracy_score
```

[60]:
```
# Accuracy on Test data
accuracy_score(y_test, pred)
```

[60]:
```
0.8032786885245902
```

[61]:
```
# Accuracy on Train data
accuracy_score(y_train, lr.predict(X_train))
```

[61]:
```
0.8589211618257261
```

[62]:
```
# Good Accuracy Achieved
```