In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving heart.csv to heart.csv
```

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(color_codes=True)
from scipy import stats
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_pre
from sklearn import metrics
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
```

In [ ]:
```
import io
import pandas as pd


heart = pd.read_csv(io.BytesIO(uploaded['heart.csv']),sep = ',')
data1 = pd.DataFrame(heart)
```

In [ ]:
```
data1
```

Out[ ]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
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

303 rows × 14 columns

In [ ]:

```
data1.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   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [ ]:
```
data1.duplicated().sort_values()
```

Out[ ]:
```
0      False
205    False
204    False
203    False
202    False
       ...
97     False
96     False
102    False
302    False
164     True
Length: 303, dtype: bool
```

In [ ]:
```
#Check duplicate rows in data
duplicate_rows = data1[data1.duplicated()]
print("Number of duplicate rows :: ", duplicate_rows.shape)
```

```
Number of duplicate rows ::  (1, 14)
```

In [ ]:
```
#we have one duplicate row.
#Removing the duplicate row
data1 = data1.drop_duplicates()
duplicate_rows = data1[data1.duplicated()]
print("Number of duplicate rows :: ", duplicate_rows.shape)
#Number of duplicate rows after dropping one duplicate row
```

```
Number of duplicate rows ::  (0, 14)
```

In [ ]:
```
#Looking for null values
print("Null values :: ")
print(data1.isnull() .sum())
```

```
Null values ::
age         0
sex         0
```

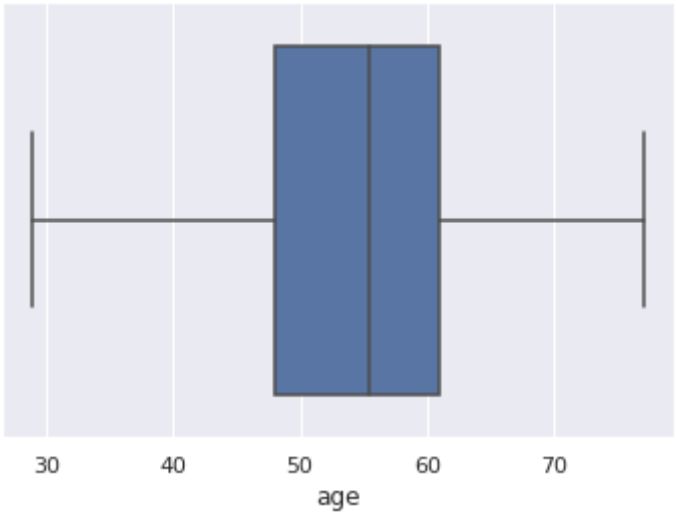
```
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

In [ ]:
```
#Check if the other data is consistent
data1.shape
```

Out[ ]:
```
(302, 14)
```

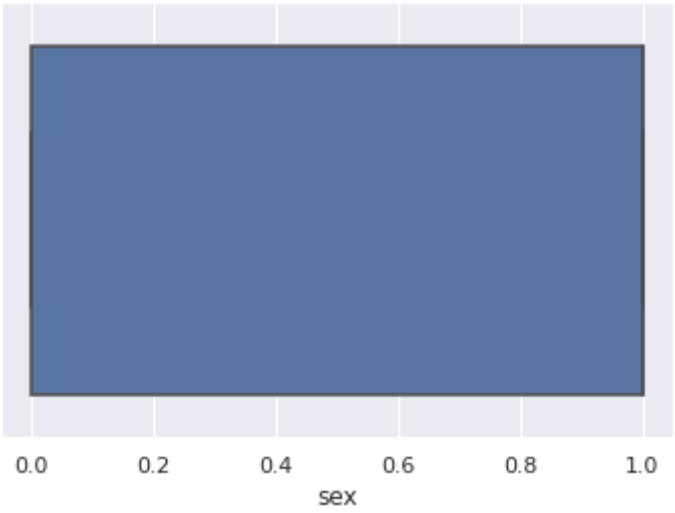
In [ ]:
```
#As there are no null values in data, we can proceed with the next steps.
#Detecting Outliers
# 1. Detecting Outliers using IQR (InterQuartile Range)
sns.boxplot(x=data1['age'])
#No Outliers observed in 'age'
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2104480c50>
```



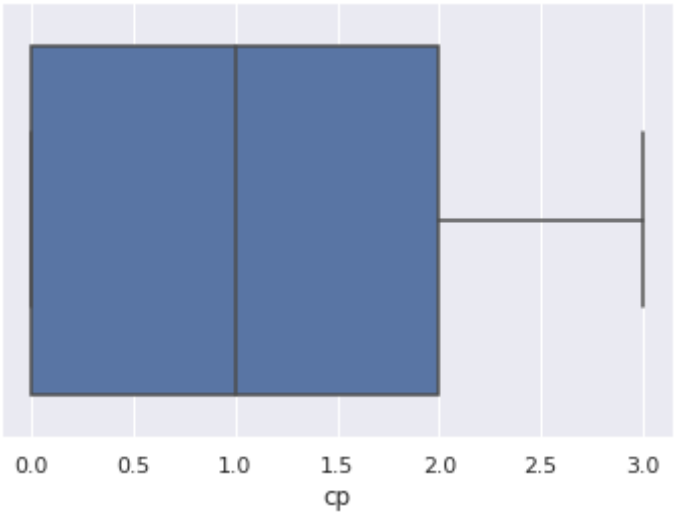
In [ ]:
```
sns.boxplot(x=data1['sex'])
#No outliers observed in sex data
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2104440d90>
```



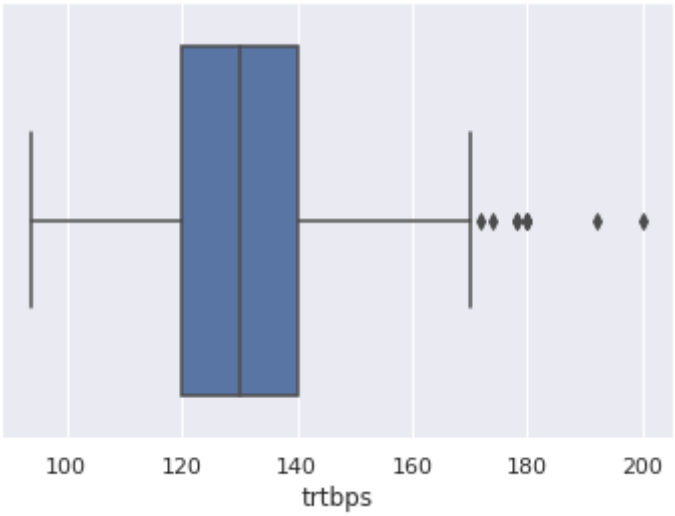
In [ ]:
```
sns.boxplot(x=data1['cp'])
#No outliers in 'cp'
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103f4af10>
```



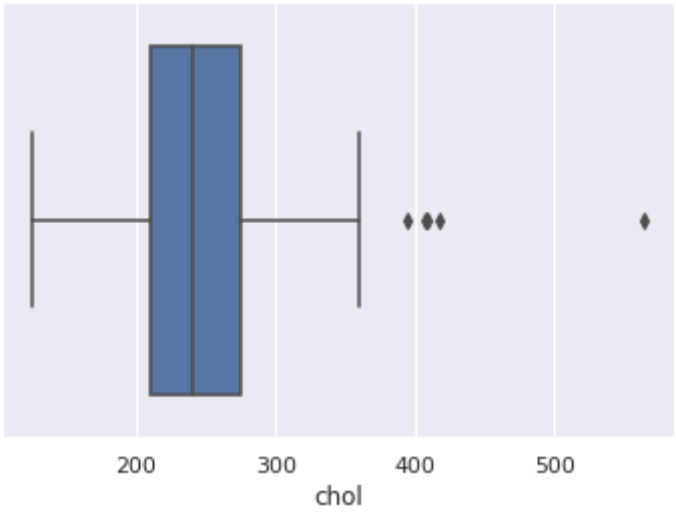
In [ ]:
```
sns.boxplot(x=data1['trtbps'])
#Some outliers are observed in 'trtbps'. They will be removed later
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103eda2d0>
```



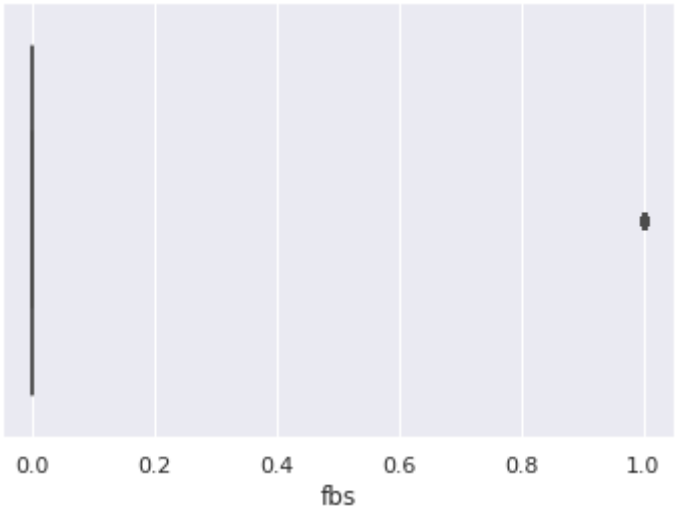
In [ ]:
```
sns.boxplot(x=data1['chol'])
#Some outliers are observed in 'chol'. They will be removed later
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103dd1390>
```



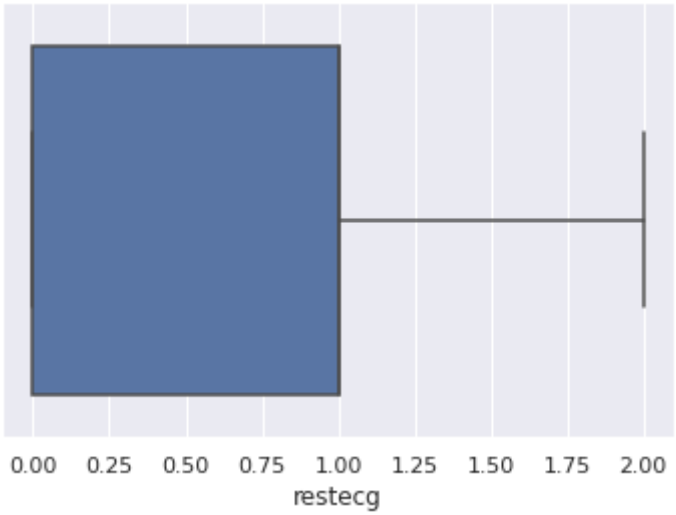
In [ ]:
```
sns.boxplot(x=data1['fbs'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103cc2350>
```



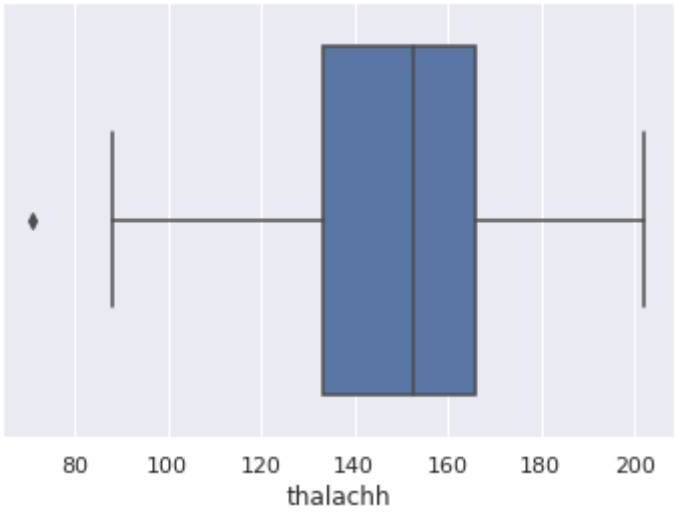
In [ ]:
```
sns.boxplot(x=data1['restecg'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103c48a90>
```



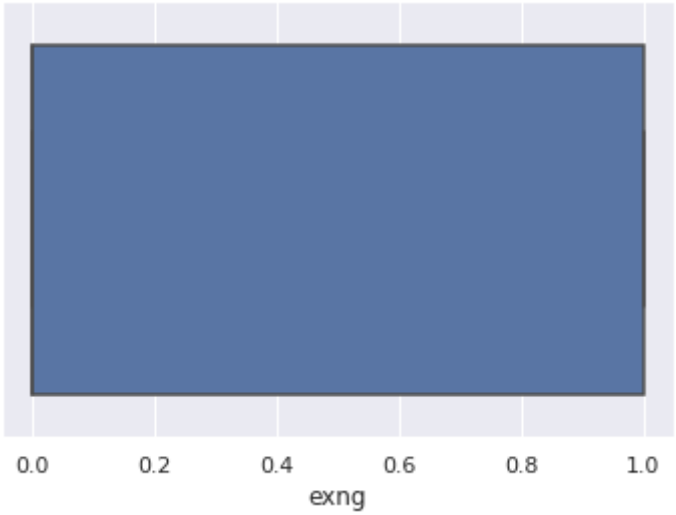
In [ ]:
```
sns.boxplot(x=data1['thalachh'])
#Outliers present in thalachh
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103cedf90>
```



In [ ]:
```
sns.boxplot(x=data1['exng'])
```

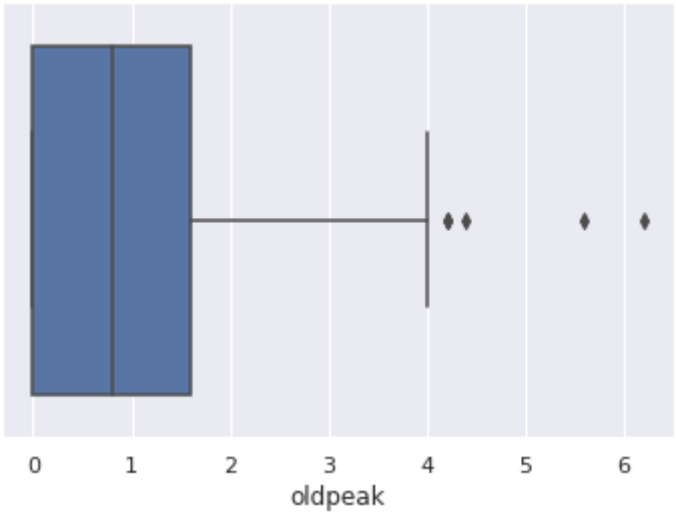
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2103b258d0>
```



In [ ]:

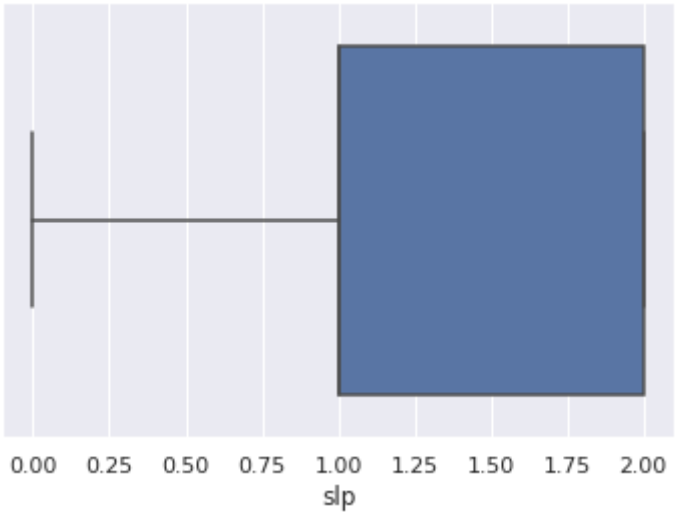
```
sns.boxplot(x=data1['oldpeak'])
#Outliers are present in 'OldPeak'
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2103aff450>



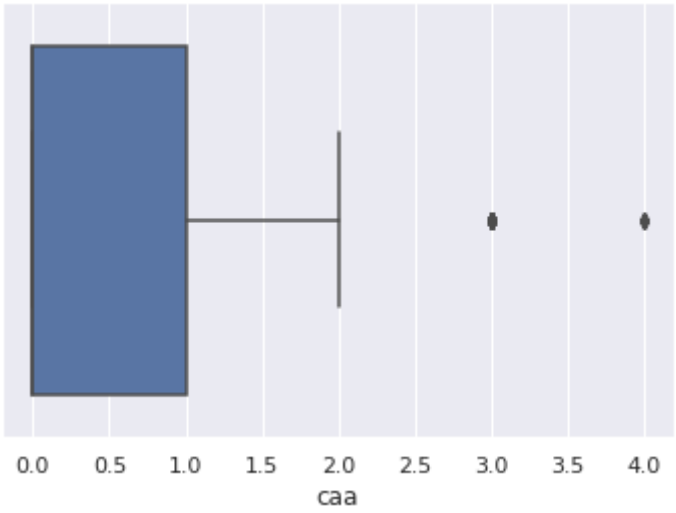
In [ ]:
```
sns.boxplot(x=data1['slp'])
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2103a74210>



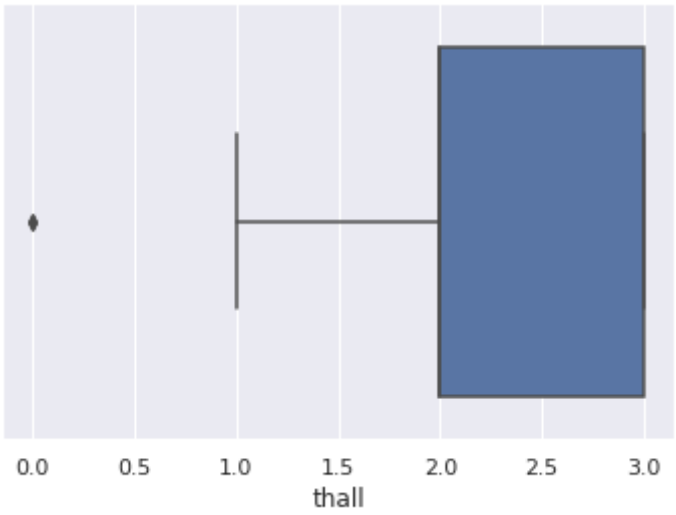
In [ ]:
```
sns.boxplot(x=data1['caa'])
#Outliers are present in 'caa'
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f21039e5e50>



In [ ]:
```
sns.boxplot(x=data1['thall'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f210395fd90>
```



In [ ]:
```
#Find the InterQuartile Range
Q1 = data1.quantile(0.25)
Q3 = data1.quantile(0.75)
IQR = Q3-Q1
print('********** InterQuartile Range **********')
print(IQR)
# Remove the outliers using IQR
data2 = data1[~((data1<(Q1-1.5*IQR))|(data1>(Q3+1.5*IQR))).any(axis=1)]
data2.shape
```

```
********** InterQuartile Range **********
age         13.5
sex          1.0
cp           2.0
trtbps      20.0
chol        63.5
fbs          0.0
restecg      1.0
thalachh    32.5
exng         1.0
oldpeak      1.6
slp          1.0
caa          1.0
```

```
thall        1.0
output       1.0
dtype: float64
(228, 14)
```

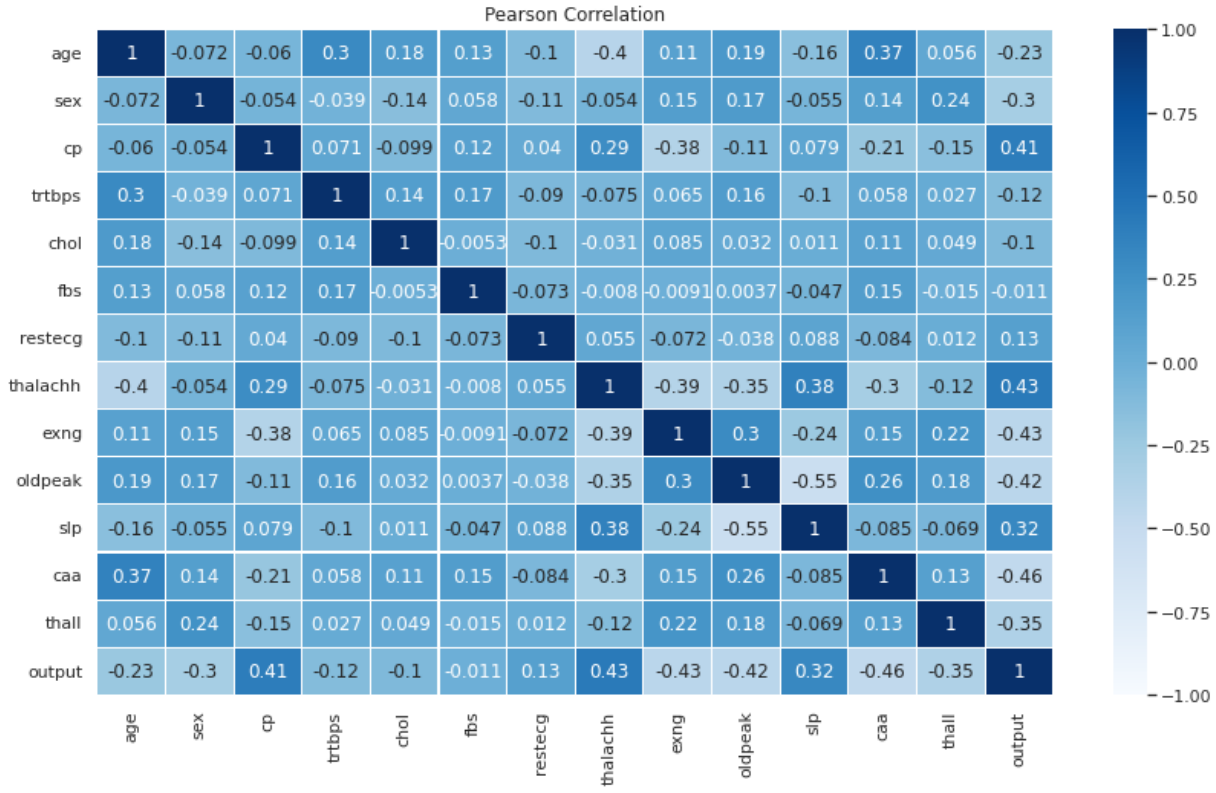
Out[ ]:

In [ ]:
```
#Removing outliers using Z-score
z = np.abs(stats.zscore(data1))
data3 = data1[(z<3).all(axis=1)]
data3.shape
```

Out[ ]:
```
(287, 14)
```

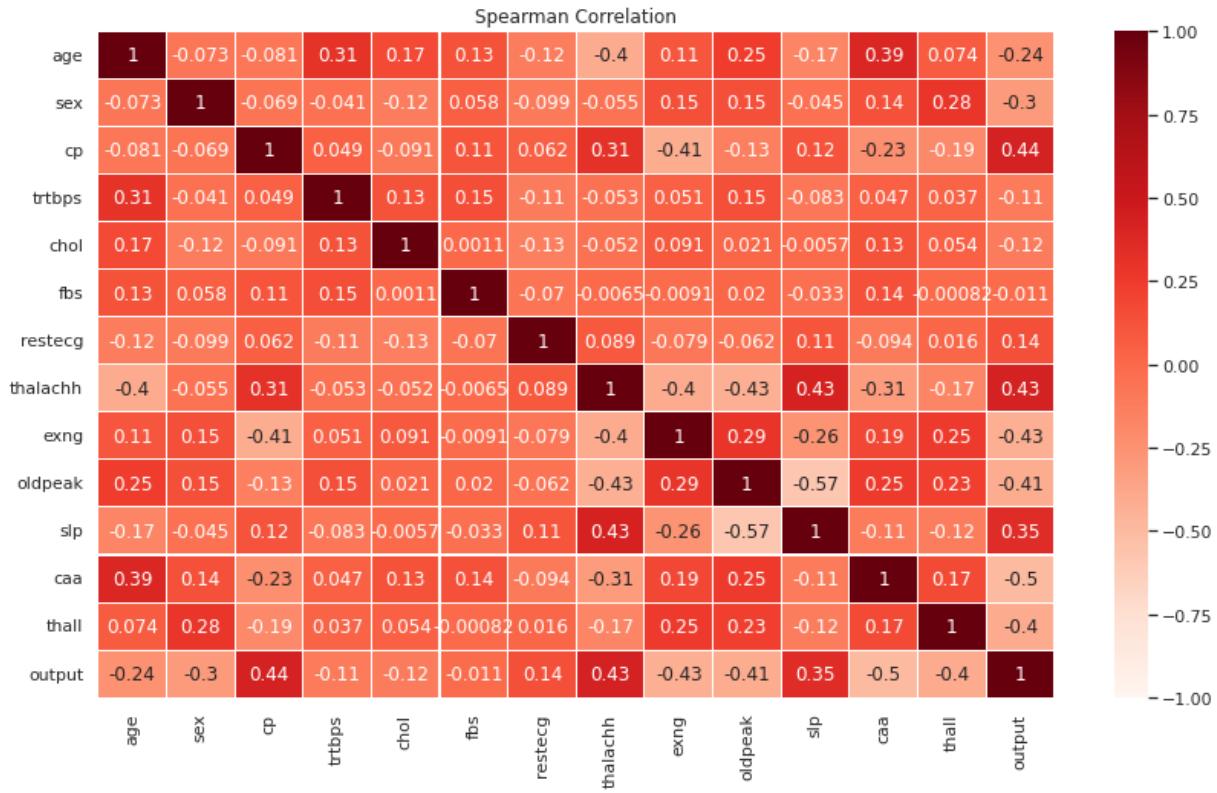
In [ ]:
```
#Finding the correlation between variables
pearsonCorr = data3.corr(method='pearson')
spearmanCorr = data3.corr(method='spearman')
fig = plt.subplots(figsize=(14,8))
sns.heatmap(pearsonCorr, vmin=-1,vmax=1, cmap = "Blues", annot=True, linewidth=0.1)
plt.title("Pearson Correlation")
```

Out[ ]:
```
Text(0.5, 1.0, 'Pearson Correlation')
```



In [ ]:
```
fig = plt.subplots(figsize=(14,8))
sns.heatmap(spearmanCorr, vmin=-1,vmax=1, cmap = "Reds", annot=True, linewidth=0.1)
plt.title("Spearman Correlation")
```

Out[ ]:
```
Text(0.5, 1.0, 'Spearman Correlation')
```



Classification

Before implementing any classification algorithm, we will divide our dataset into training data and test data. I have used 70% of the data for training and the remaining 30% will be used for testing.

In [ ]:
```
#From this we observe that the minimum correlation between output and other features
#fbs,trtbps and chol
x = data3.drop("output", axis=1)
y = data3["output"]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)
```

1. Logistic Regression Classifier

The code snippet used to build Logistic Regression Classifier is,

In [ ]:
```
#Building classification models
names = ['Age', 'Sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng',
#    ****************Logistic Regression*****************
logReg = LogisticRegression(random_state=0, solver='liblinear')
logReg.fit(x_train, y_train)
#Check accuracy of Logistic Regression
y_pred_logReg = logReg.predict(x_test)
#Model Accuracy
print("Accuracy of logistic regression classifier :: " ,metrics.accuracy_score(y_tes
#Removing the features with low correlation and checking effect on accuracy of model
x_train1 = x_train.drop("fbs",axis=1)
x_train1 = x_train1.drop("trtbps", axis=1)
x_train1 = x_train1.drop("chol", axis=1)
x_train1 = x_train1.drop("restecg", axis=1)
x_test1 = x_test.drop("fbs", axis=1)
```

```
x_test1 = x_test1.drop("trtbps", axis=1)
x_test1 = x_test1.drop("chol", axis=1)
x_test1 = x_test1.drop("restecg", axis=1)
logReg1 = LogisticRegression(random_state=0, solver='liblinear').fit(x_train1,y_trai
y_pred_logReg1 = logReg1.predict(x_test1)
print("nAccuracy of logistic regression classifier after removing features:: " ,metr
```

```
Accuracy of logistic regression classifier ::  0.8620689655172413
nAccuracy of logistic regression classifier after removing features::  0.89655172413
7931
```

K Nearest Neighbours Classifier

Implement K nearest neighbor classifier and print the accuracy of the model.

In [ ]:

```
#K Neighbours Classifier
knc =  KNeighborsClassifier()
knc.fit(x_train,y_train)
y_pred_knc = knc.predict(x_test)
print("Accuracy of K-Neighbours classifier :: ", metrics.accuracy_score(y_test,y_pre
```

```
Accuracy of K-Neighbours classifier ::  0.6666666666666666
```

Random Forest Classifier

Implement a random forest classifier using the code,

In [ ]:

```
# Using Random forest classifier
rf = RandomForestClassifier(n_estimators=500)
rf.fit(x_train,y_train)
y_pred_rf = rf.predict(x_test)
print("Accuracy of Random Forest Classifier :: ", metrics.accuracy_score(y_test, y_p
#Find the score of each feature in model and drop the features with low scores
f_imp = rf.feature_importances_
for i,v in enumerate(f_imp):
    print('Feature: %s, Score: %.5f' % (names[i],v))
```

```
Accuracy of Random Forest Classifier ::  0.8850574712643678
Feature: Age, Score: 0.11323
Feature: Sex, Score: 0.02937
Feature: cp, Score: 0.08790
Feature: trtbps, Score: 0.06807
Feature: chol, Score: 0.09871
Feature: fbs, Score: 0.01399
Feature: restecg, Score: 0.01973
Feature: thalachh, Score: 0.13220
Feature: exng, Score: 0.05298
Feature: oldpeak, Score: 0.11097
Feature: slp, Score: 0.04446
Feature: caa, Score: 0.13625
Feature: thall, Score: 0.09215
```