Heart Disease Dataset

Problem Statement

- This dataset will allow us to **predict if someone has heart disease** based on their **sex, age, blood pressure** and a variety of other metrics.

Dataset Information:

- Number of Rows : 303
- Number of Columns : 14

Feature Column Description:

- **age** : age in years
- **sex** :

  `1 = male;`

  `0 = female;`
- **cp** : chest pain type

  `1: typical angina`

  `2: atypical angina`

  `3: non-anginal pain`

  `4: asymptomatic`
- **restbp** : resting blood pressure (in mm Hg on admission to the hospital)
- **chol** : serum cholestoral in mg/dl
- **fbs** : fasting blood sugar
- **restecg** : resting electrocardiographic results

  `0: normal`

  `1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)`

  `2: showing probable or definite left ventricular hypertrophy by Estes' criteria`
- **thalach** : maximum heart rate achieved
- **exang** : exercise induced angina

1 = yes;

0 = no

- **oldpeak** : ST depression induced by exercise relative to rest
- **slope** : the slope of the peak exercise ST segment.

```
1: upsloping

2: flat

3: downsloping
```

- **ca** : number of major vessels (0-3) colored by fluoroscopy
- **thal** : this is short of thalium heart scan.

3 = normal;

6 = fixed defect;

7 = reversable defe

- **target** : diagnosis of heart disease, the predicted attribute

1: Import the modules

In [2]:

```
# Read Data
import numpy as np     # Linear Algebra (calculate the mean and standard deviation)
import pandas as pd    # manipulate data, data processing, load csv file I/O (e.g. pd.read_

# Visualization
# Seaborn
import seaborn as sns
# matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
# Plotly
import plotly
import plotly.express as px
import plotly.graph_objs as go

# style
plt.style.use("fivethirtyeight")
sns.set_style("darkgrid")

# ML model building; Pre Processing & Evaluation
from sklearn.model_selection import train_test_split                  # split  data intc
from sklearn.ensemble import RandomForestClassifier                   # this will make c
from sklearn import svm
from sklearn.svm import SVC                                          # this will make a
from sklearn.metrics import confusion_matrix, classification_report   # this creates a c
```

2: Import the data

In [3]:

```
# Import first 5 rows
df = pd.read_csv("datasets_33180_43520_heart.csv")
df.head()
```

Out[3]:

	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

In [4]:

```
# checking dimension (num of rows and columns) of dataset
df.shape
```

Out[4]:

```
(303, 14)
```

Checking for Numerical and Categorical features

In [5]:

```
# check dataframe structure like columns and its counts, datatypes & Null Values
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)
```

In [6]:

```
# check the datatypes
df.dtypes
```

Out[6]:

```
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
```

- Observed that **there is no categorical features** in this dataset. Only have **numerical features of int64 & float64**.

In [7]:

```
df.count()  # Gives number of data points in each variable
```

Out[7]:

```
age         303
sex         303
cp          303
trestbps    303
chol        303
fbs         303
restecg     303
thalach     303
exang       303
oldpeak     303
slope       303
ca          303
thal        303
target      303
dtype: int64
```

4: EDA (Exploratory Data Analysis)

- EDA is a way of **Visualizing, Summarizing and interpreting** the information that is **hidden in rows and column** format.

- Find Unwanted Columns
- Find Missing Values
- Find Features with one value

- Explore the Categorical Features
- Find Categorical Feature Distribution
- Relationship between Categorical Features and Label
- Explore the Numerical Features
- Find Discrete Numerical Features
- Relation between Discrete numerical Features and Labels
- Find Continous Numerical Features
- Distribution of Continous Numerical Features
- Relation between Continous numerical Features and Labels
- Find Outliers in numerical features
- Explore the Correlation between numerical features

1. Find Unwanted Columns

- There is no unwanted column present in given dataset to remove.

  EX: ID, S.No etc

2. Find Missing Values

In [8]:

```
df.isnull().sum()  # Listing Number of missing values by feature column wise.
```

Out[8]:

```
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 [9]:

```
# Missing value representation by Heatmap
plt.figure(figsize=(12,10))
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
plt.xticks(fontsize=14)
plt.title('Count of Missing Values by Heat Map', fontsize=20, fontweight = 'bold')
plt.show()
```

- From graph understood that there is **no missing values** in this dataset.

3. Find Features with one value

In [10]:

```
for column in df.columns:
    print(column,df[column].nunique())
```

```
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
```

- All columns have **more than 1 unique value.** No feature found with one value.
- There could be chance of only one category in a particular feature. In Categorical features, suppose gender column we have only one value ie male.Then there is no use of that feature in dataset.

To find Number of Categories in each Feature

In [11]:

```
# feature cp
df.cp.value_counts()
```

Out[11]:

```
0    143
2     87
1     50
3     23
Name: cp, dtype: int64
```

- **4 categories** in column "cp(Chest Pain)"

In [12]:

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

Out[12]:

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

- **2 categories** in column "target"

In [13]:

```
# feature sex
df.sex.value_counts() # Number of Category's in sex
```

Out[13]:

```
1    207
0     96
Name: sex, dtype: int64
```

- **2 categories** in column "sex"

In [14]:

```
# feature slope
df.slope.value_counts()
```

Out[14]:

```
2    142
1    140
0     21
Name: slope, dtype: int64
```

- **3 categories** in column "slope"

In [15]:

```
# feature restecg
df.restecg.value_counts()
```

Out[15]:

```
1    152
0    147
2      4
Name: restecg, dtype: int64
```

- **3 categories** in column "restecg"

In [16]:

```
# feature exang
df.exang.value_counts()
```

Out[16]:

```
0    204
1     99
Name: exang, dtype: int64
```

- **2 categories** in column "exang"

In [17]:

```
# feature thal
df.thal .value_counts()
```

Out[17]:

```
2    166
3    117
1     18
0      2
Name: thal, dtype: int64
```

- **4 categories** in column "thal"

4. Explore the Categorical Features

In [18]:

```
categorical_features = [feature for feature in df.columns if df[feature].dtypes=='O']
categorical_features
```

Out[18]:

```
[]
```

- There is no categorical features in this dataset.

In [19]:

```
for feature in categorical_features:
    print('The feature is {} and number of categories are {}'.format(feature,len(df[feature
```

- Steps **5 to 7 not required** since there is **no categorical features**.

8. Explore the Numerical Features

In [20]:

```
numerical_features = df.select_dtypes(exclude='object')
numerical_features
```

Out[20]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targ
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	

303 rows × 14 columns

9. Find Discrete Numerical Features

In [21]:

```
discrete_feature=[feature for feature in numerical_features if len(df[feature].unique())<25
print("Discrete Variables Count: {}".format(len(discrete_feature)))
```

```
Discrete Variables Count: 9
```

10. Find Continous Numerical Features

In [22]:

```
continuous_features=[feature for feature in numerical_features if feature not in discrete_f
print("Continuous feature Count {}".format(len(continuous_features)))
```

```
Continuous feature Count 5
```

In [23]:

```
continuous_features
```

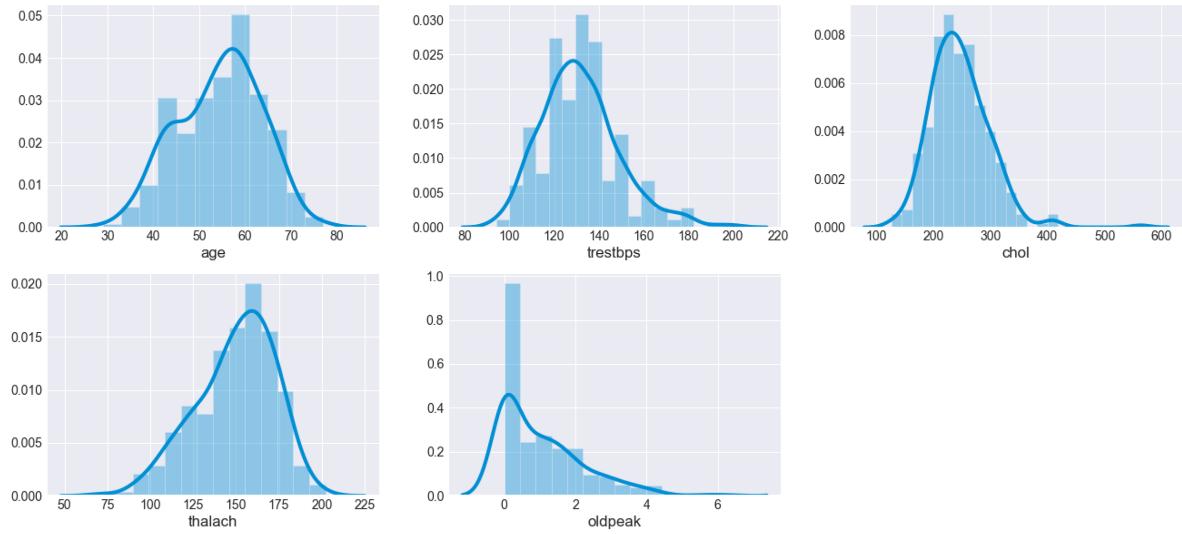
Out[23]:

```
['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
```

11. Distribution of Continous Numerical Features

In [24]:

```
# plot a univariate distribution of continues observations
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for continuous_feature in continuous_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.distplot(df[continuous_feature])
    plt.xlabel(continuous_feature)
    plotnumber+=1
plt.show()
```

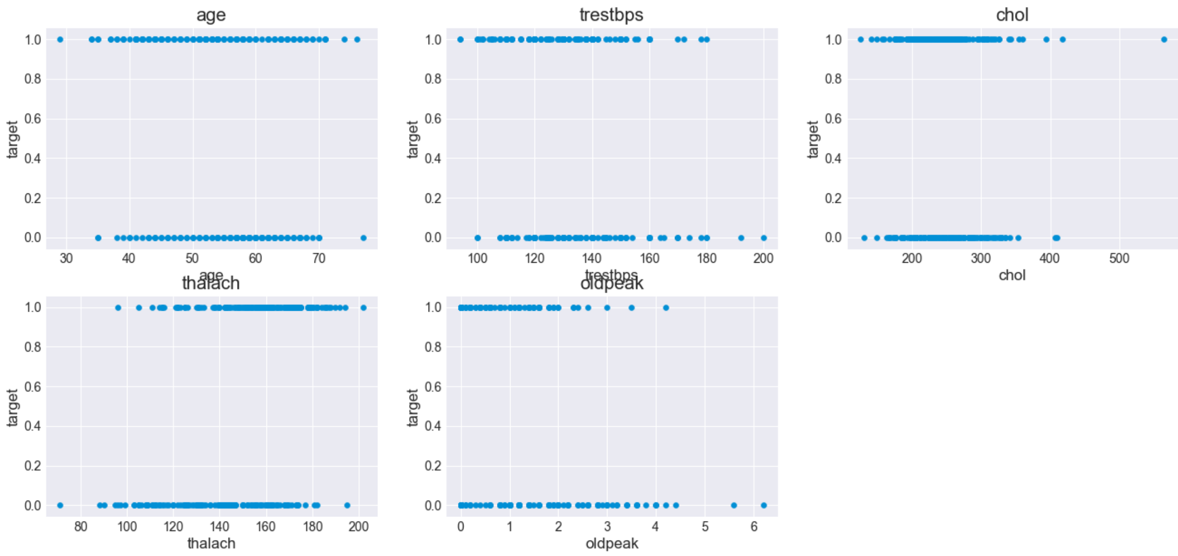


- it seems all continuous features are not normally distributed
- **Age, chol, trestbps & oldpeak** are **right skewed**
- **thalach** is **left skewed**.

12. Relation between Continous numerical Features and Labels

In [25]:

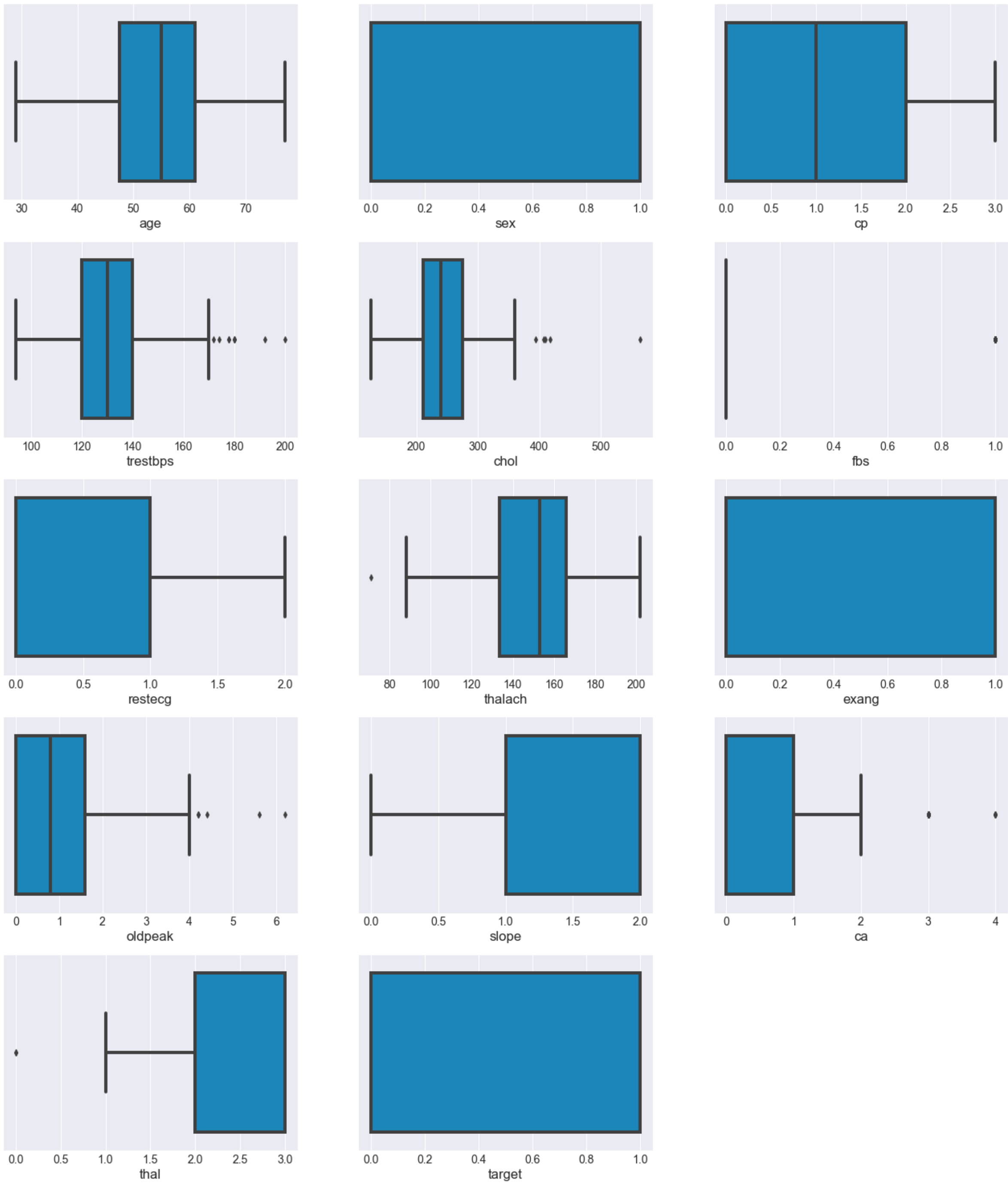
```
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for feature in continuous_features:
    data=df.copy()
    ax = plt.subplot(12,3,plotnumber)
    plt.scatter(data[feature],data['target'])
    plt.xlabel(feature)
    plt.ylabel('target')
    plt.title(feature)
    plotnumber+=1
plt.show()
```



13. Find Outliers in numerical features

In [26]:

```
# boxplot on numerical features to find outliers
plt.figure(figsize=(20,60), facecolor='white')
plotnumber =1
for numerical_feature in numerical_features:
    ax = plt.subplot(12,3,plotnumber)
    sns.boxplot(df[numerical_feature])
    plt.xlabel(numerical_feature)
    plotnumber+=1
plt.show()
```



- Found **outliers** in **"trestbps", "chol", "fbs", "thalach", "oldpeak", "ca" & "thal"**.

14. Explore the Correlation between numerical features

In [27]:

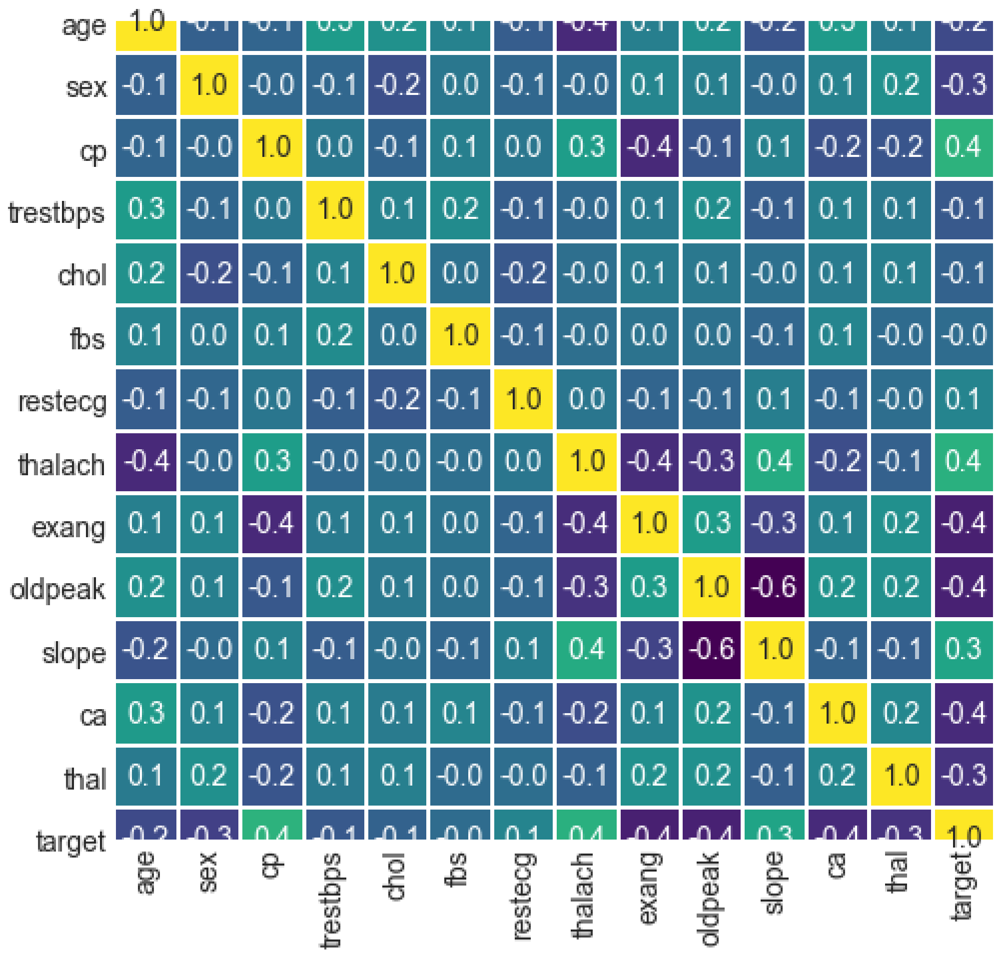
```
## Checking for correlation
df.corr()
```

Out[27]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	c
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.
trestbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.
thalach	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.
exang	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.
oldpeak	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

In [28]:

```
## Checking for correlation
cor_mat=df.corr()
fig = plt.figure(figsize=(15,7))
sns.heatmap(cor_mat, annot=True, cbar=False, cmap='viridis', fmt='.1f', linewidth=1, square
plt.show()
```

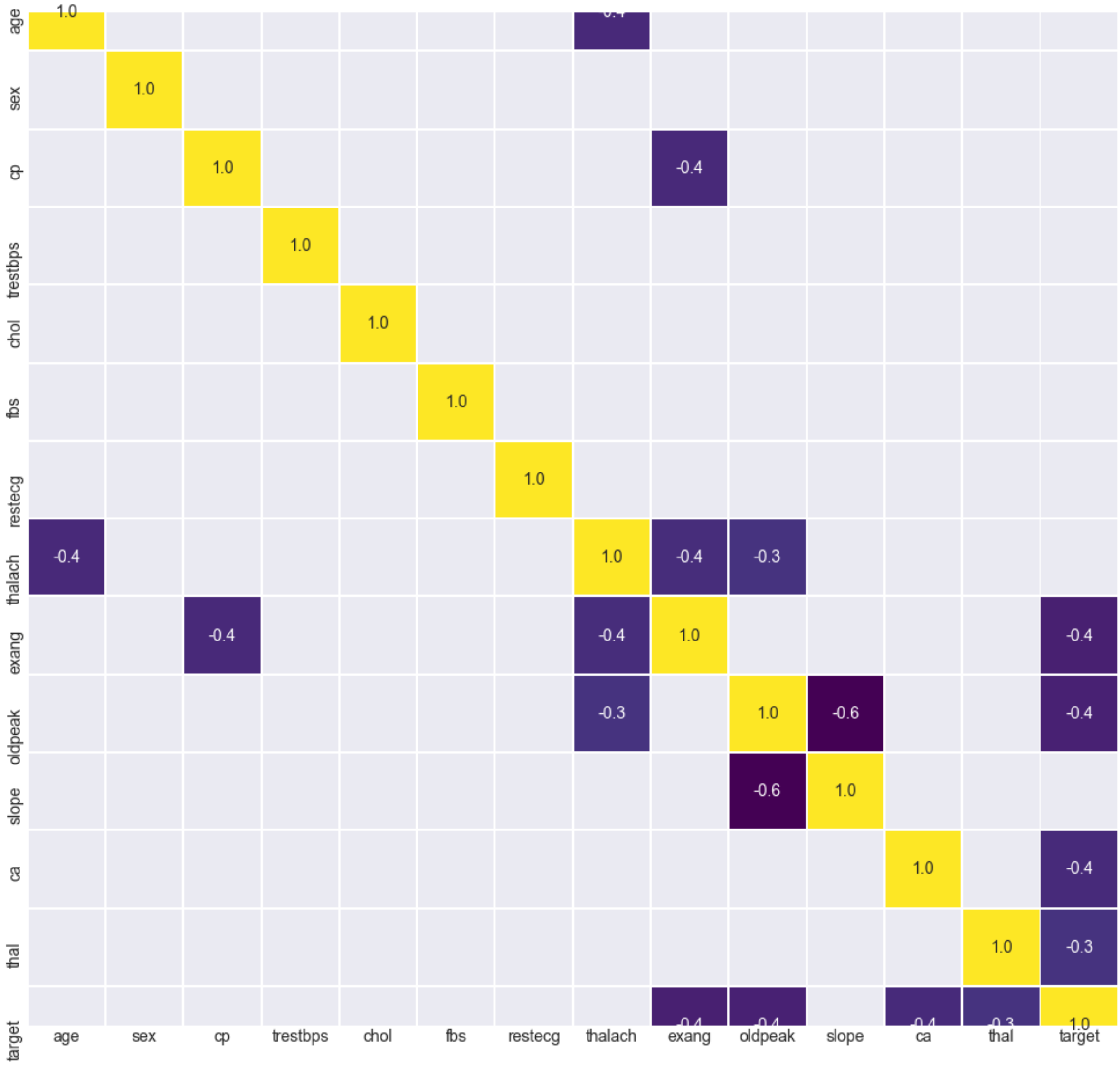


In [29]:

```
# highlight only features have correlation between 0.5 and -0.3
plt.figure(figsize=(16,15))
corr = df.corr()
sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.3)], annot=True, cbar=False,
                                    cmap='viridis', linewidth=1, fmt='.1f', square=True)
```

Out[29]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758088b9e8>
```



15. Descriptive statistics

In [30]:

```
# descriptive statistics (numerical columns)
df.describe()
```

Out[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	14
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	2
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	7
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	13
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	15
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	16
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	20

- for **each feature** it provides below:

```
1. count       --->  total no of data points

2. statistics  --->  mean, standard deviation

3. min & max   --->  values of feature

4. percentile  --->  25%, 50%, 75%
```

5. Data Visualization

- Used below **visualisation libraries**

```
1. Matplotlib

2. Seaborn (statistical data visualization)

3. Plotly
```

1. Categorical

- Categorical data :

```
1. Numerical Summaries

2. Histograms

3. Pie Charts
```

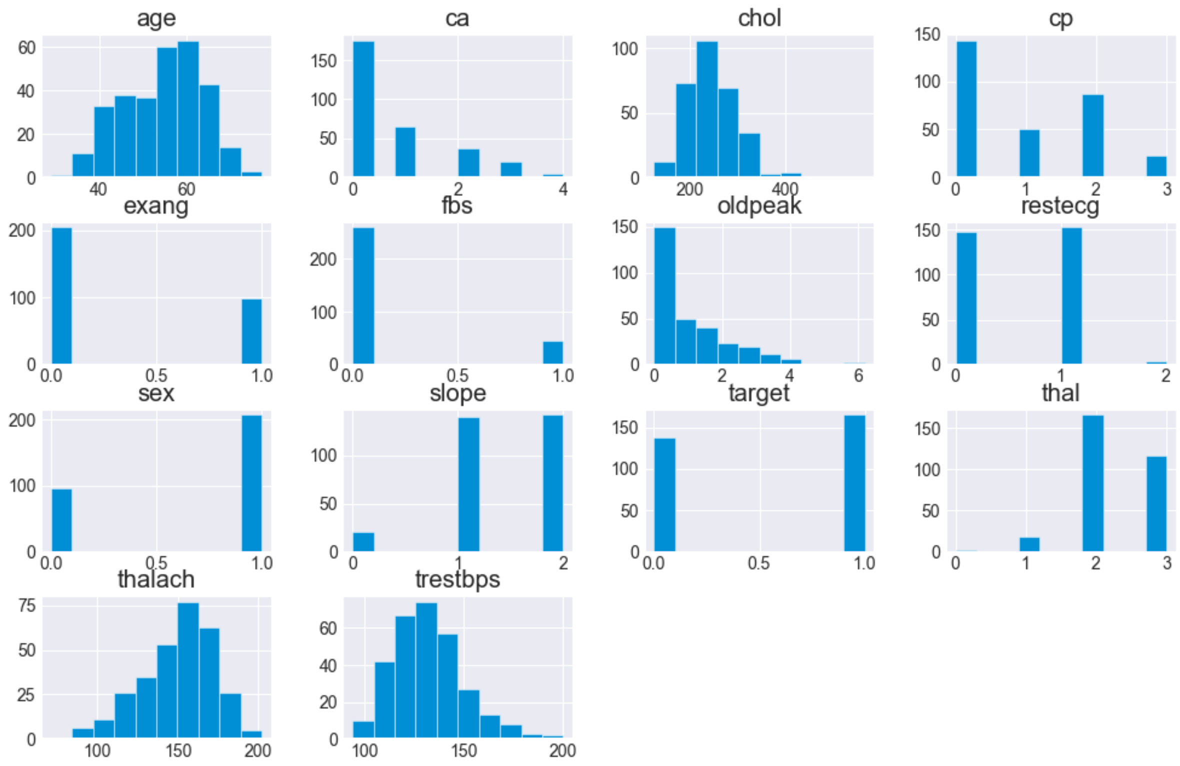
2. Univariate Analysis

- Univariate Analysis : data consists of **only one variable (only x value)**.

```
1. Line Plots / Bar Charts

2. Histograms

3. Box Plots

4. Count Plots

5. Descriptive Statistics techniques

6. Violin Plot
```

In [31]:

```
# Histogram for dataset
df.hist(figsize = (15,10))
plt.show()
```

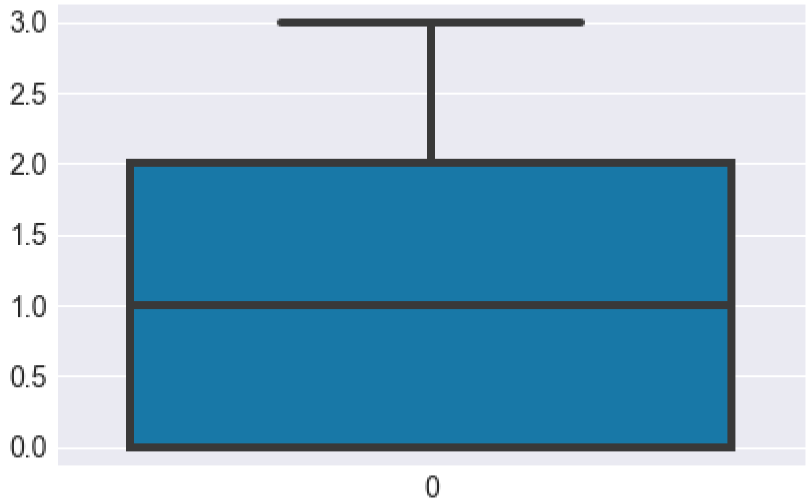


In [32]:

```
# Box Plot used to find out the outliers in feature column of "ConfirmedCases"
plt.figure(figsize=(6,4))
sns.boxplot(data=df['cp'], palette='winter')
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758044cf60>
```



Count Plot

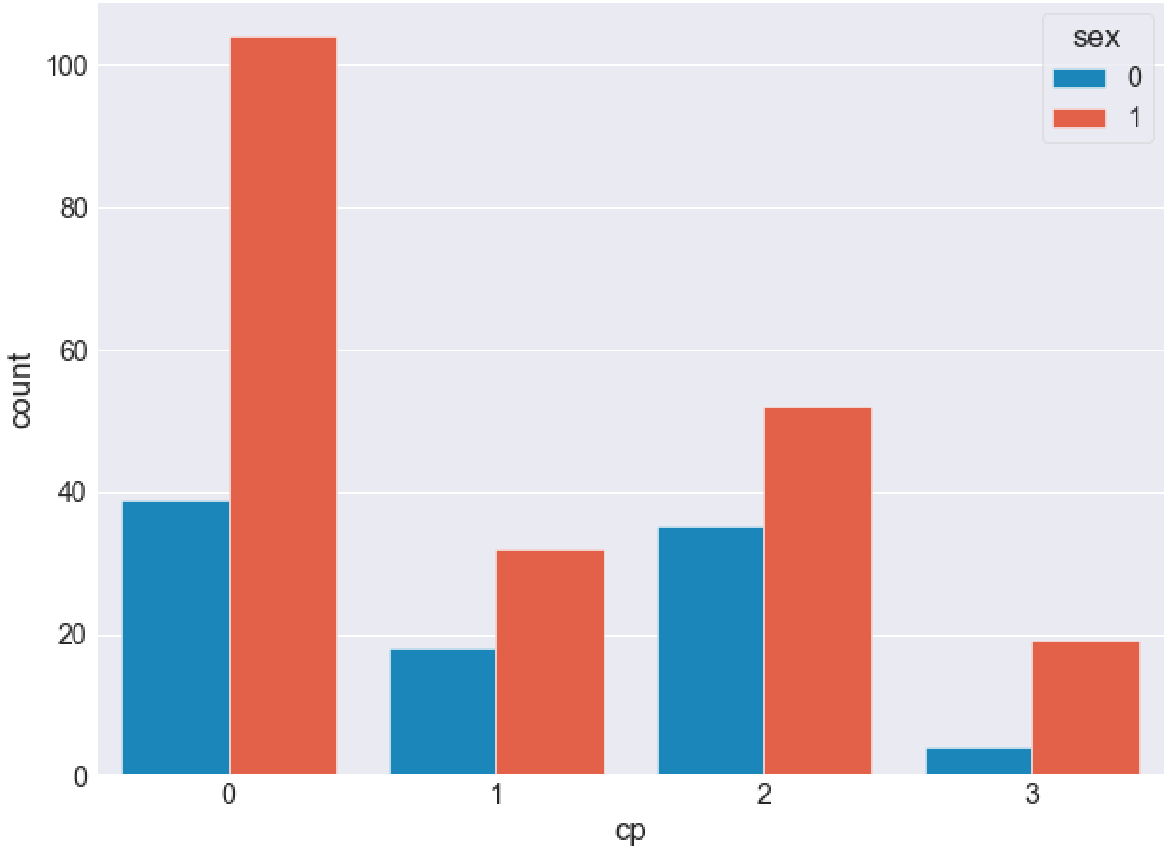
- Show the counts of observations in each categorical bin using bars

In [33]:

```
# Count Plot for "Chest Pain" & "sex"
plt.figure(figsize=(9,7))
sns.countplot(x="cp", data=df, hue="sex")
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27580941dd8>
```



- **cp : Chest pain**

  typical angina ---> Counts of Female:40; Male:100;

  atypical angina ---> Counts of Female:18; Male:35;

  non-anginal pain ---> Counts of Female:38; Male:55;

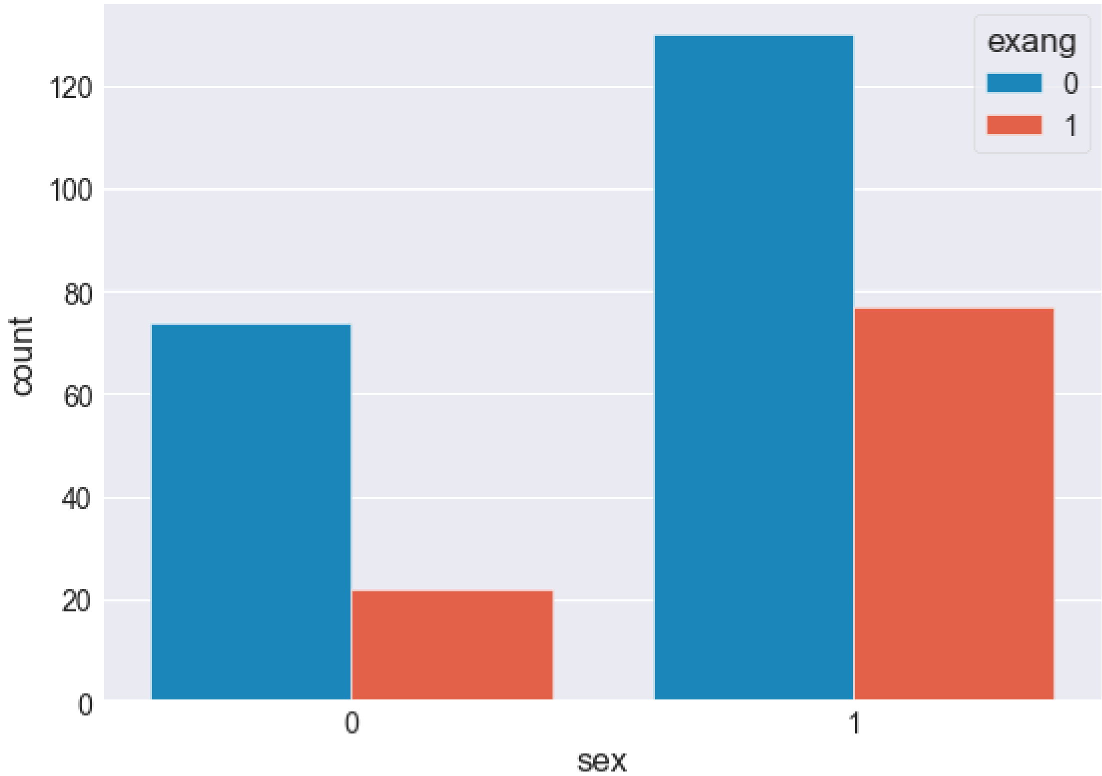
asymptomatic ---> Counts of Female:5; Male:18;

In [34]:

```
# Count Plot for "sex" & "exang"
plt.figure(figsize=(8,6))
sns.countplot(x='sex', hue='exang', data=df) # palette="Set3"
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x275ff3d87b8>
```



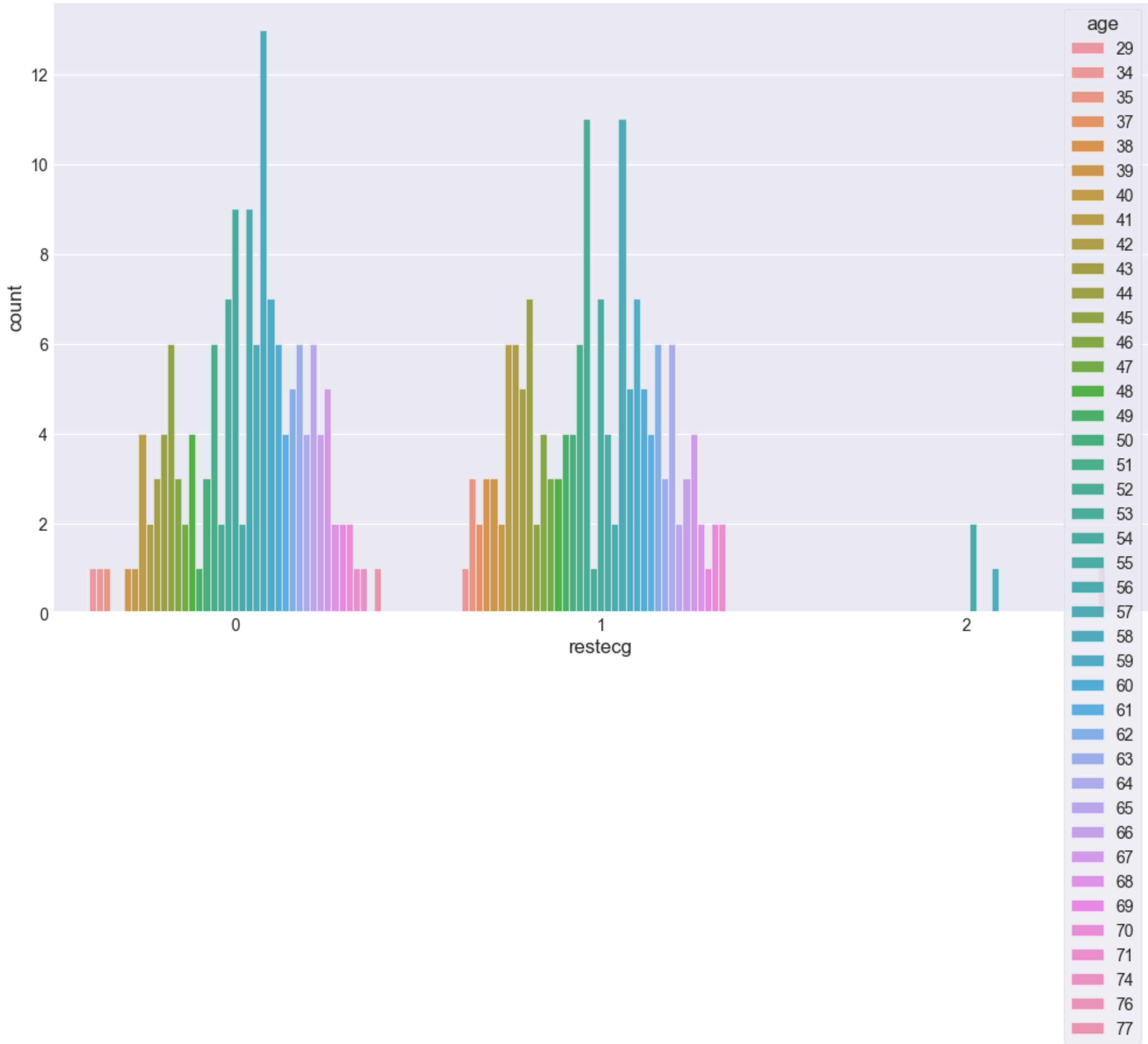
- exang ---> exercise induced angina (1 ---> Yes; 0 ---> No)
- sex ---> (1 ---> Male; 0 ---> Female)

In [35]:

```
# Count Plot for "restecg : resting electrocardiographic results" & "age"
plt.figure(figsize=(15,9))
sns.countplot(x='restecg', data=df, hue='age')
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x275811bf8d0>
```



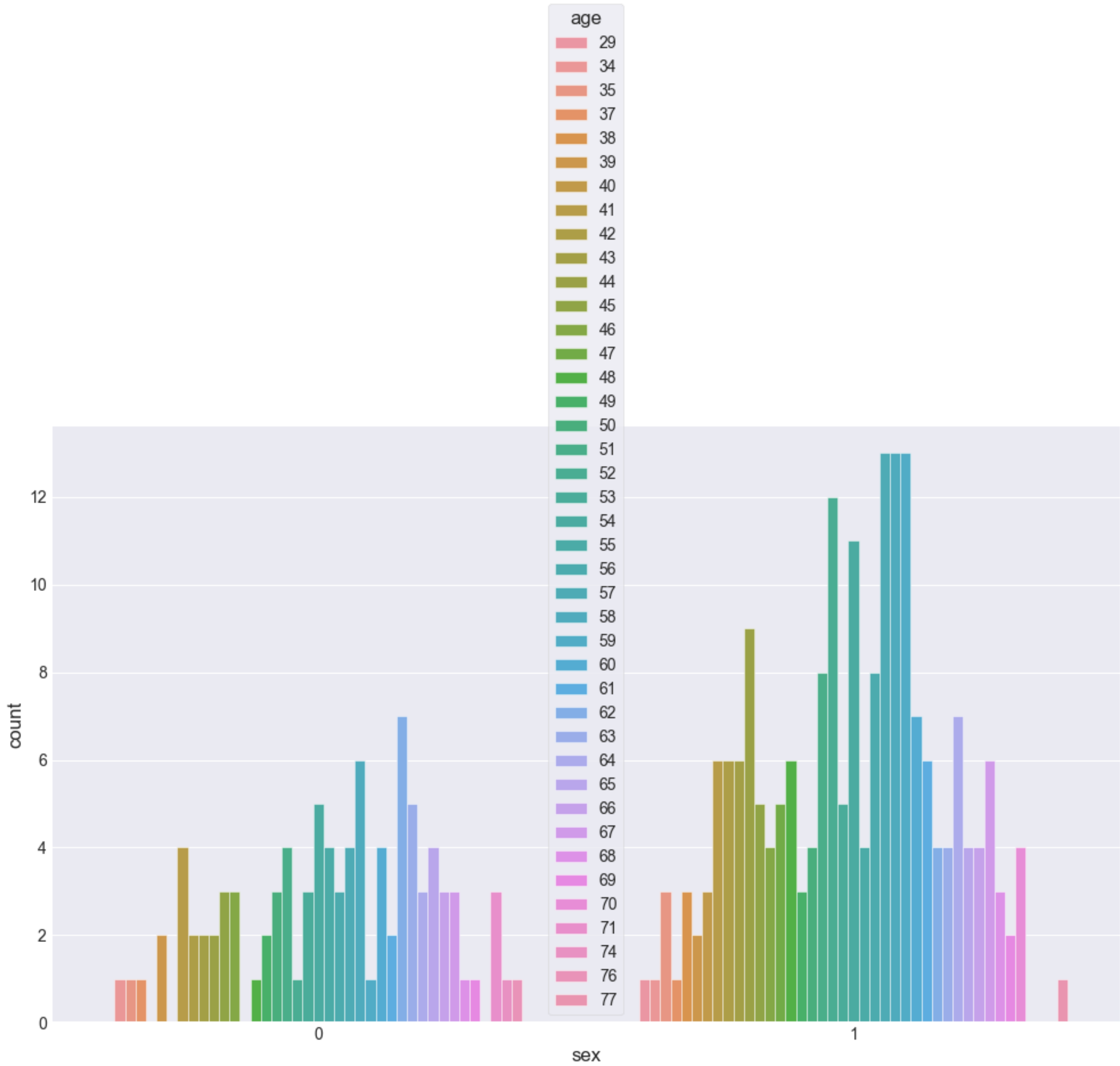
- restecg : resting electrocardiographic results shown based on age for 3 categories.

In [36]:

```
# Count Plot for "sex" & "age"
plt.figure(figsize=(15,9))
sns.countplot(x='sex', data=df, hue='age')
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x275811b3828>
```



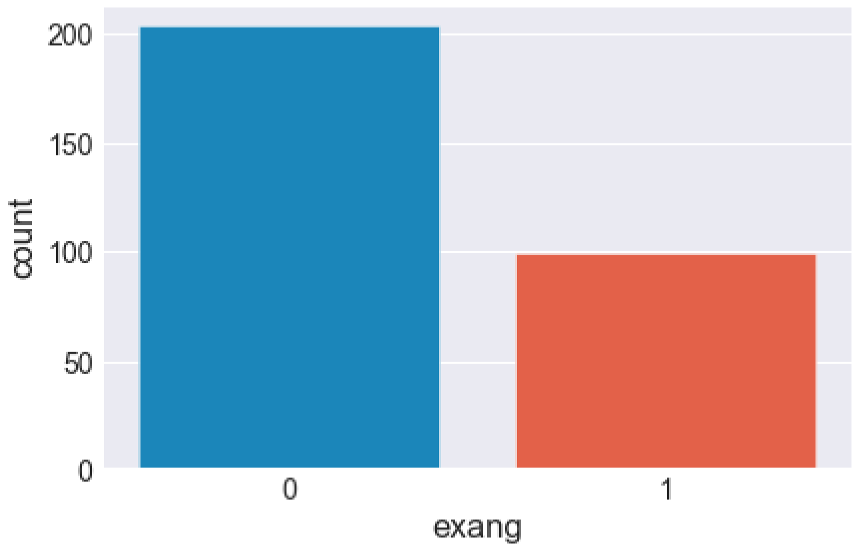
- sex (1 = male; 0 = female)
- Counts of male & female w.r.t age

In [37]:

```
sns.countplot(df.exang)
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27582db9198>
```



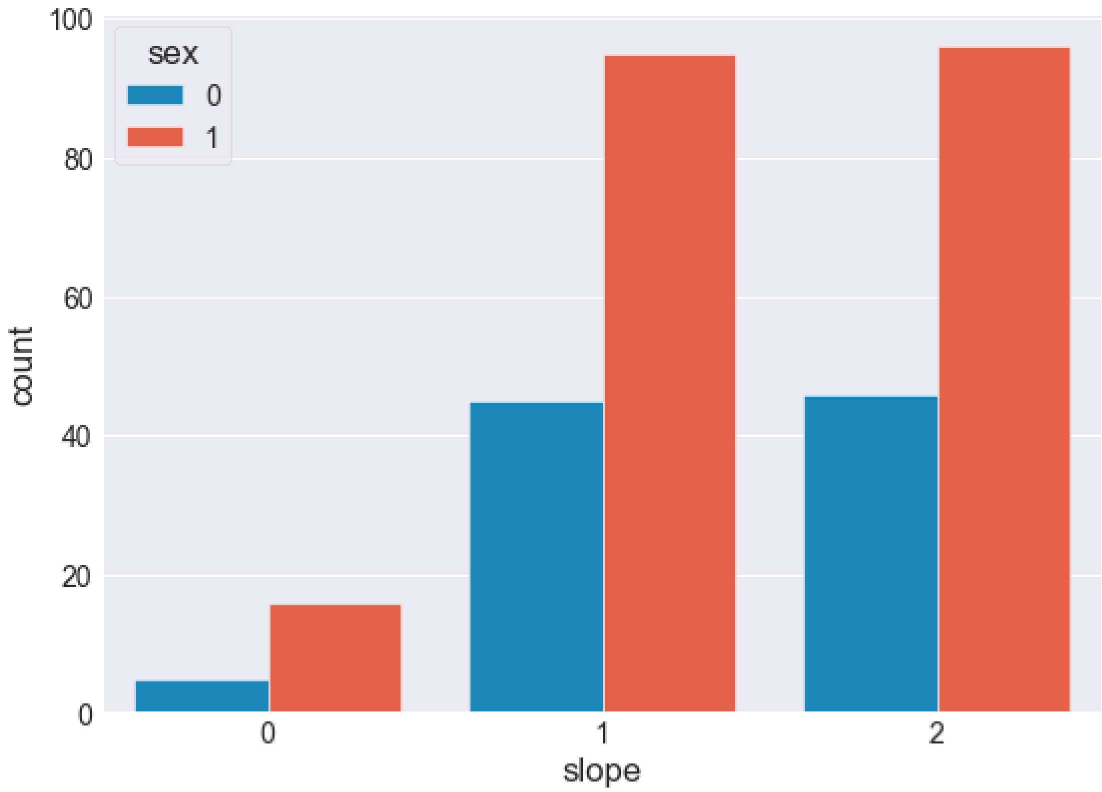
- counts of "exang : exercise induced angina" shown on category wise.

In [38]:

```
# Count plot of slope w.r.t sex
plt.figure(figsize=(8,6))
sns.countplot(x='slope', hue='sex', data=df)
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27582c8ecf8>
```

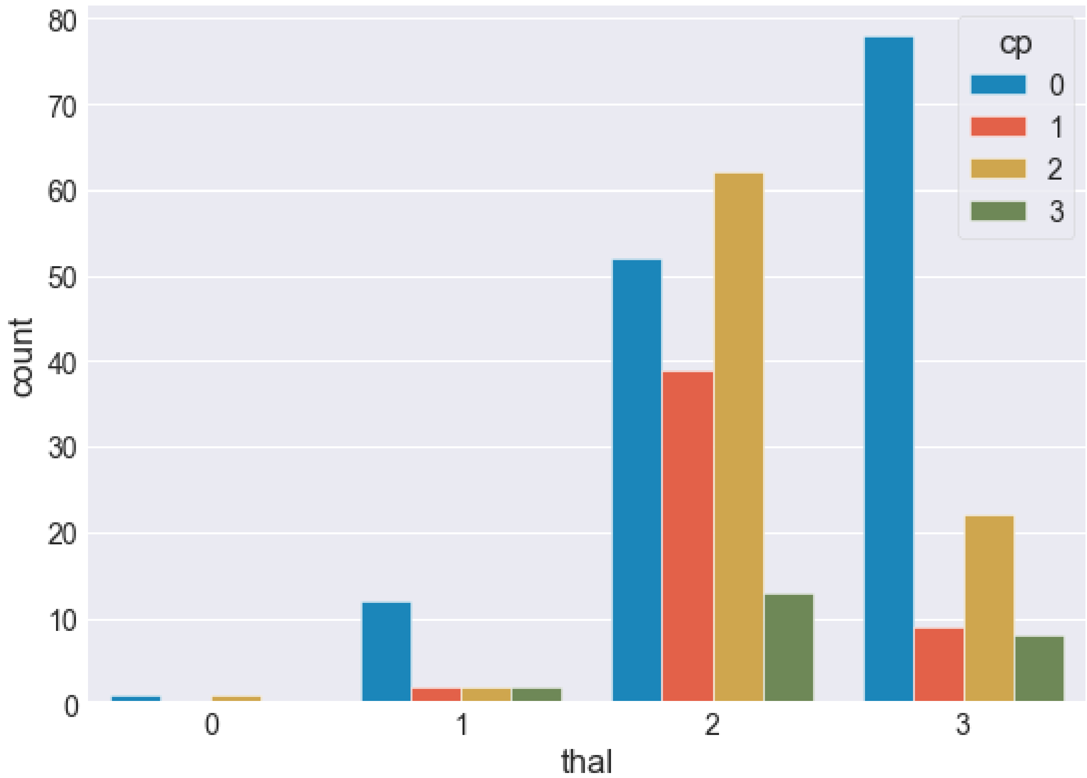


In [39]:

```
# Count plot of thal w.r.t cp
plt.figure(figsize=(8,6))
sns.countplot(x='thal', hue='cp', data=df)
```

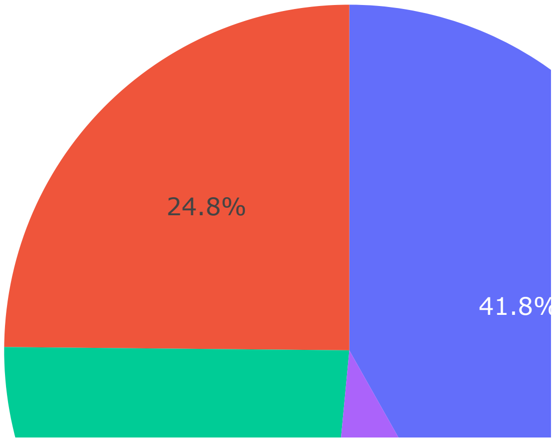
Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x275804322b0>
```



In [40]:

```
# Pie chart for Fatalities(Death)

fig = go.Figure(data=[go.Pie(labels=df['cp'],
                      values=df['target'])])

fig.show()
```



3. Bivariate Analysis

- **Bivariate Analysis** : data involves **two different variables**.

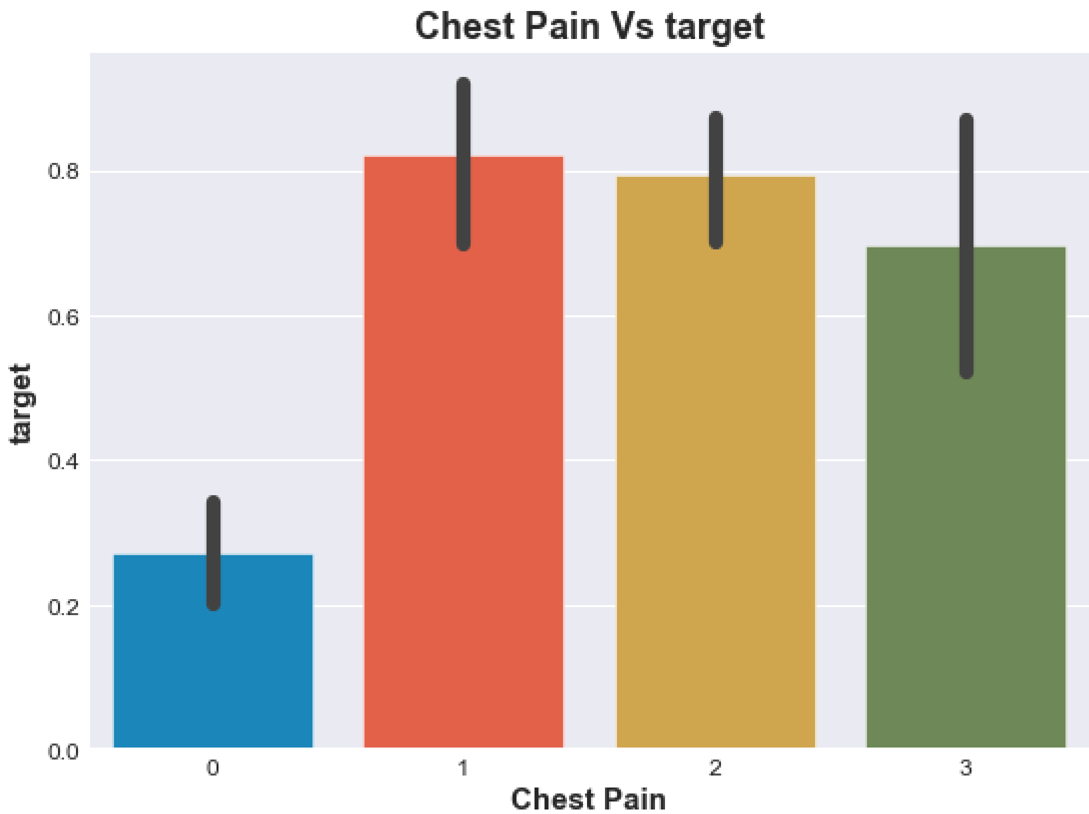
  ```
  1. Bar Charts

  2. Scatter Plots

  3. FacetGrid
  ```

- There are **three** types of bivariate analysis

  ```
  1. Numerical & Numerical

  2. Categorical & Categorical

  3. Numerical & Categorical
  ```

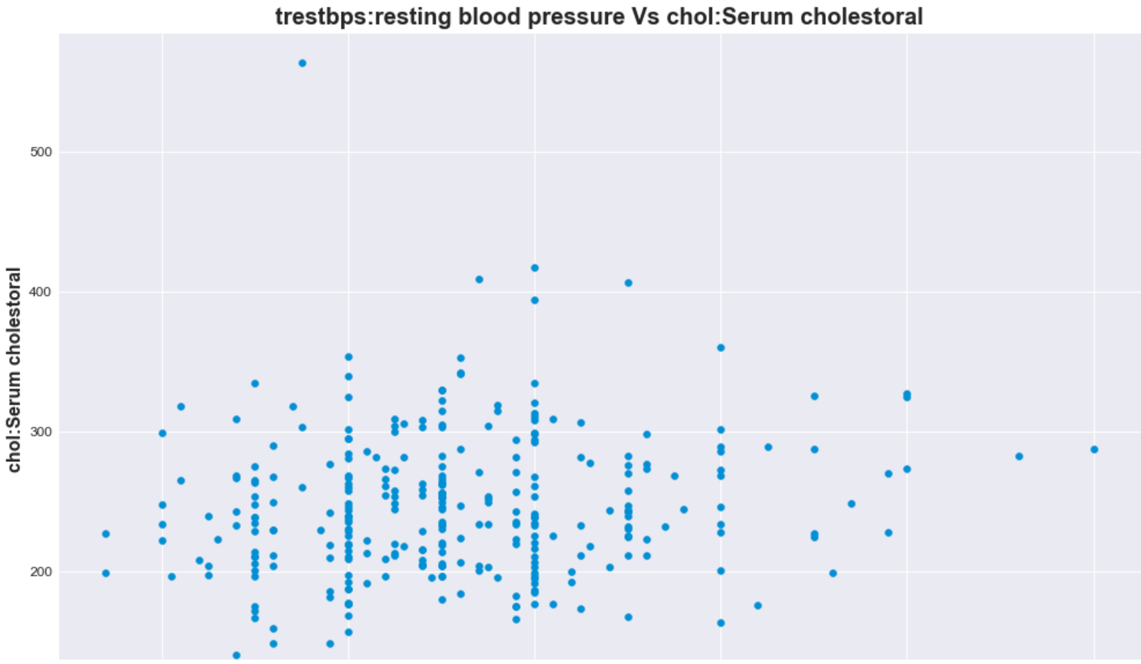
In [41]:

```
# Bar plot between "Chest Pain" & "target"
plt.figure(figsize=(8,6))
sns.barplot(x='cp', y='target', data=df)

plt.xlabel('Chest Pain', fontsize=15, fontweight='bold')
plt.ylabel('target', fontsize=15, fontweight='bold')

plt.title('Chest Pain Vs target', fontsize=18, fontweight='bold')

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```



In [42]:

```
# Scatter plot between "trestbps:resting blood pressure" & "chol:Serum cholestoral"
plt.figure(figsize=(15,10))
plt.scatter(df['trestbps'], df['chol'])

plt.xlabel('trestbps:resting blood pressure', fontsize=16, fontweight='bold')
plt.ylabel('chol:Serum cholestoral', fontsize=16, fontweight='bold')

plt.title('trestbps:resting blood pressure Vs chol:Serum cholestoral', fontsize=20, fontwei

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```

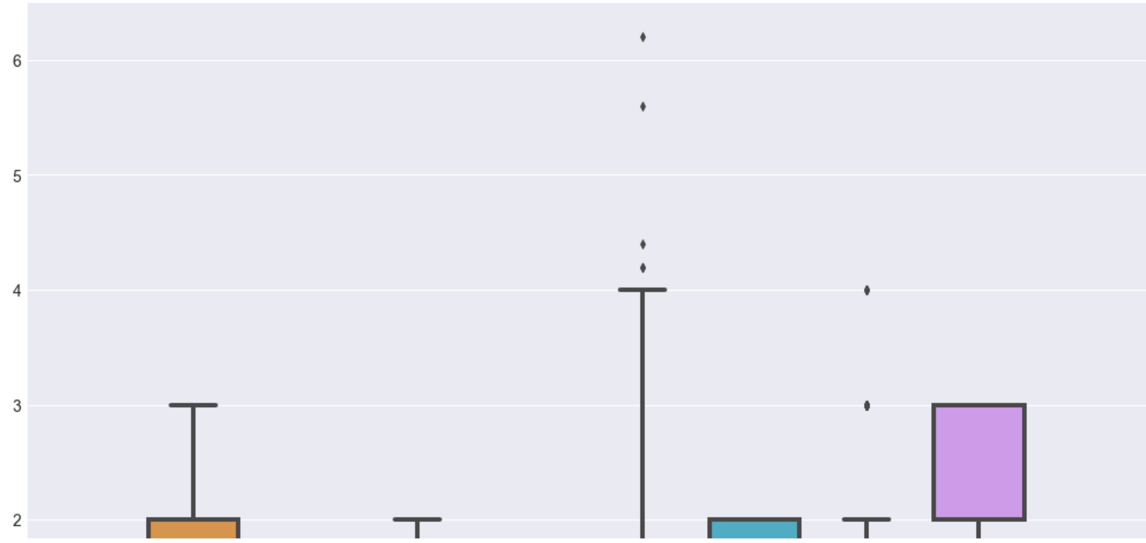


In [43]:

```
# Box plot created for features
plt.figure(figsize=(16,12))
sns.boxplot(data=df[['sex','cp','fbs','restecg','exang','oldpeak','slope','ca','thal','targ
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27582f4e550>
```



- From above graph found **outliers** in features : **"fbs", "oldpeak", "ca", "thal"**

In [44]:

```
# Box plot created for features columns
plt.figure(figsize=(10,9))
sns.boxplot(data=df[['age','trestbps','chol','thalach']])
```

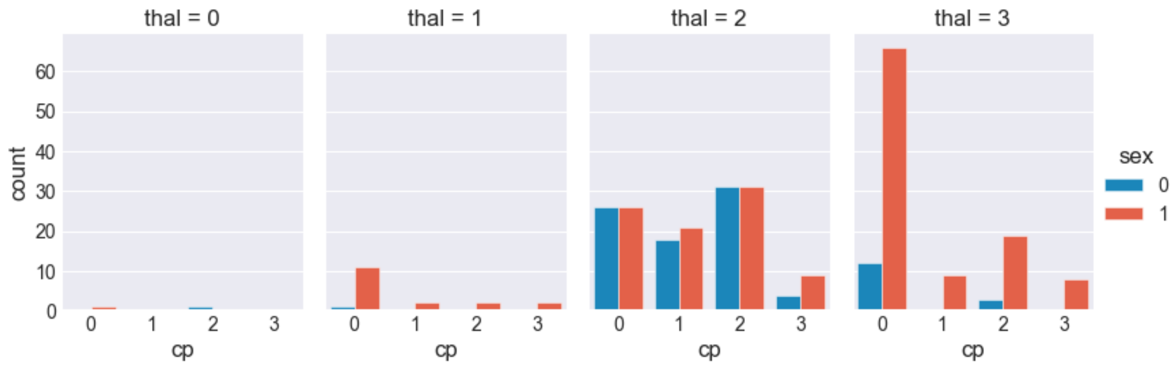
Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x275801e4048>
```

- From above graph found **outliers** in features : **"chol", "thalach"**

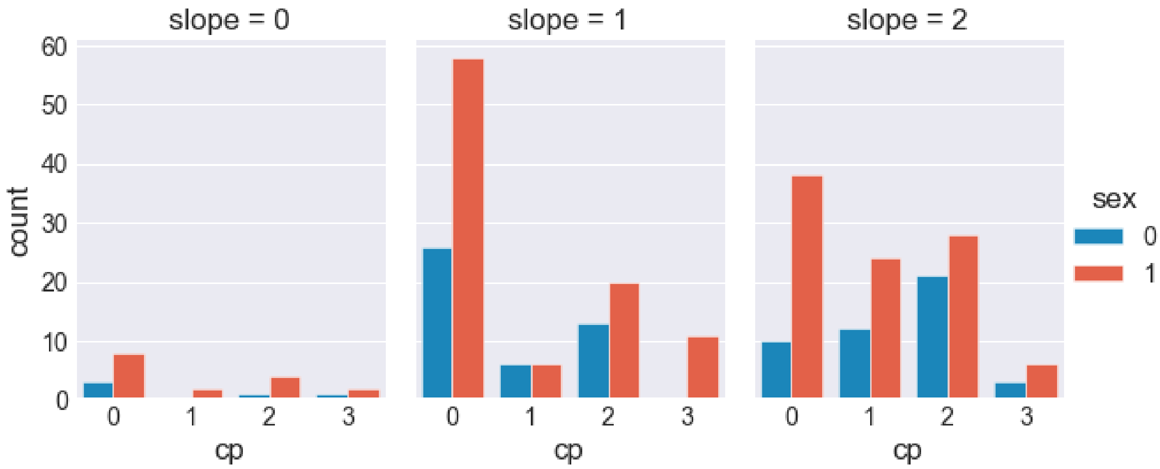
In [45]:

```
# Catplot to show Chest pain based on "thal" and "sex"
sns.catplot(x="cp", hue="sex", col="thal",
                data=df, kind="count",
                height=4, aspect=.7);
```



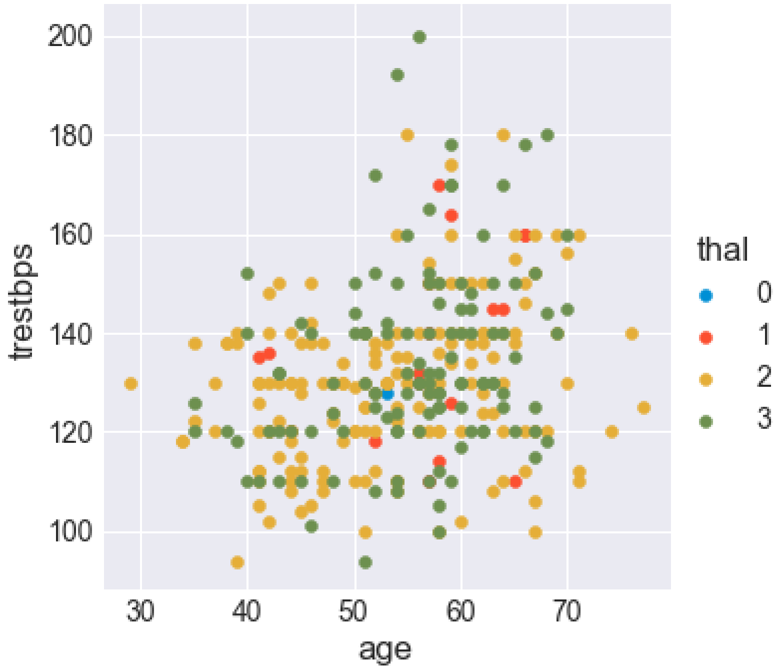
In [46]:

```
# Catplot to show Chest pain based on "thal" and "sex"
sns.catplot(x="cp", hue="sex", col="slope",
                data=df, kind="count",
                height=4, aspect=.7);
```



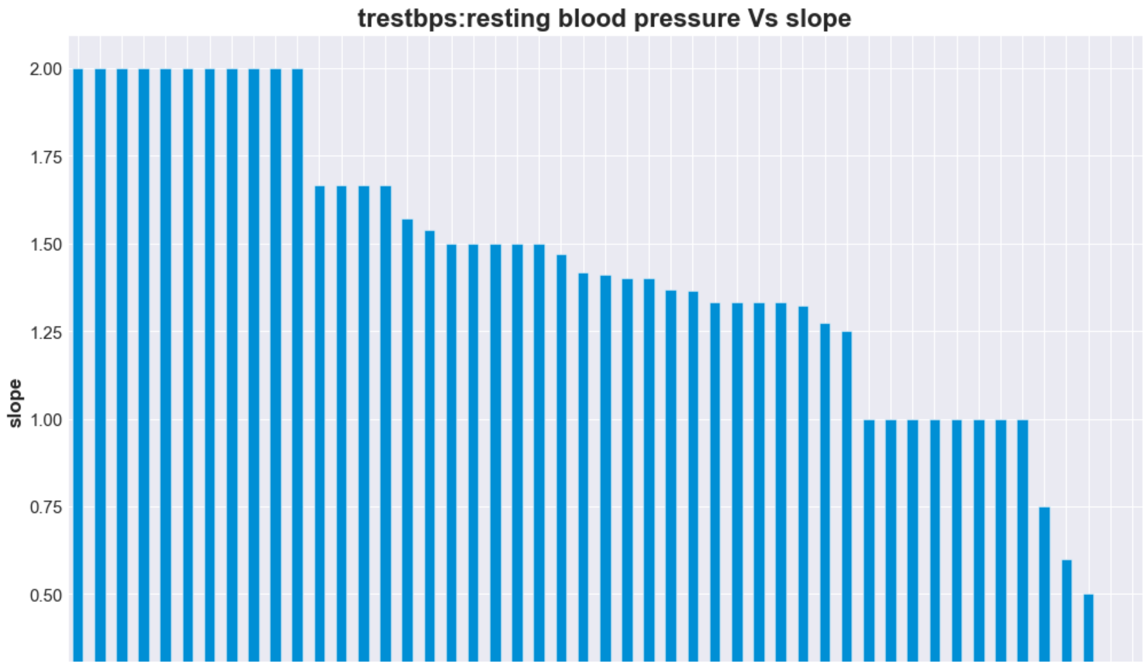
In [47]:

```
# FacetGrid
sns.FacetGrid(df, hue="thal", height=5).map(plt.scatter,"age","trestbps").add_legend();
plt.show()
```



In [48]:

```
# Bar Chart for showing count of "trestbps:resting blood pressure" and "slope"
plt.figure(figsize=(15,11))

df.groupby('trestbps').mean().sort_values(by='slope', ascending=False)['slope'].plot(kind='

plt.xlabel('trestbps:resting blood pressure', fontsize=17, fontweight = 'bold')
plt.ylabel('slope', fontsize=17, fontweight = 'bold')

plt.title('trestbps:resting blood pressure Vs slope', fontsize=22, fontweight = 'bold')

plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.show()
```



3. Multivariate Analysis

- 1. Pair Plot

In [49]:

```
## Checking for pairplot
sns.pairplot(df.drop('target', axis=1))
```

Out[49]:

```
<seaborn.axisgrid.PairGrid at 0x27582f0b748>
```

6. Check & Reduce Skewness

a. Checking Skewness for feature "age"

In [50]:

```
# Checking the skewness of "age" attributes
df['age'].skew()
```

Out[50]:

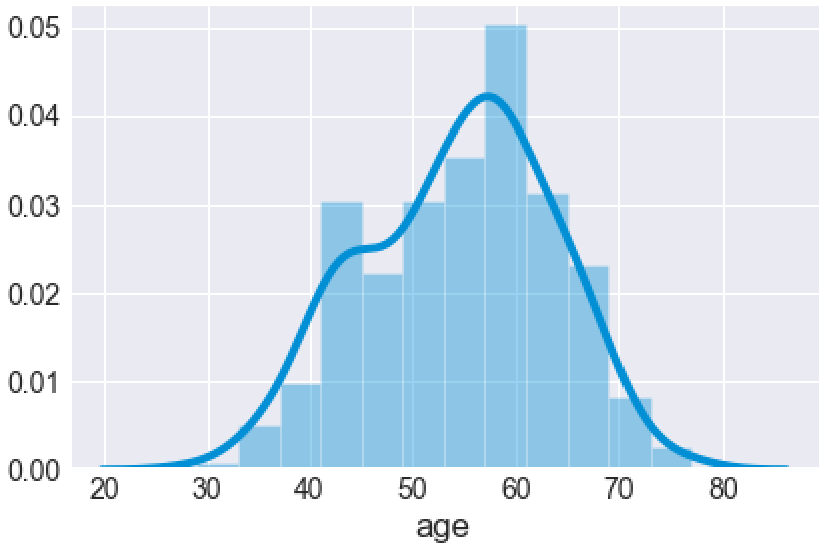
```
-0.2024633654856539
```

In [51]:

```
# plotting the histogram for  "age" attributes
sns.distplot(df['age'], hist=True)
```

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27589cc4978>
```



- feature "age" is **right skewed.**

In [52]:

```
# calculating the square for the column df['age'] column
Square_age = np.square(df['age'])
Square_age
```

Out[52]:

```
0      3969
1      1369
2      1681
3      3136
4      3249
       ...
298    3249
299    2025
300    4624
301    3249
302    3249
Name: age, Length: 303, dtype: int64
```

In [53]:

```
# checking the skewness
Square_age.skew()
```

Out[53]:

```
0.15634227492279207
```

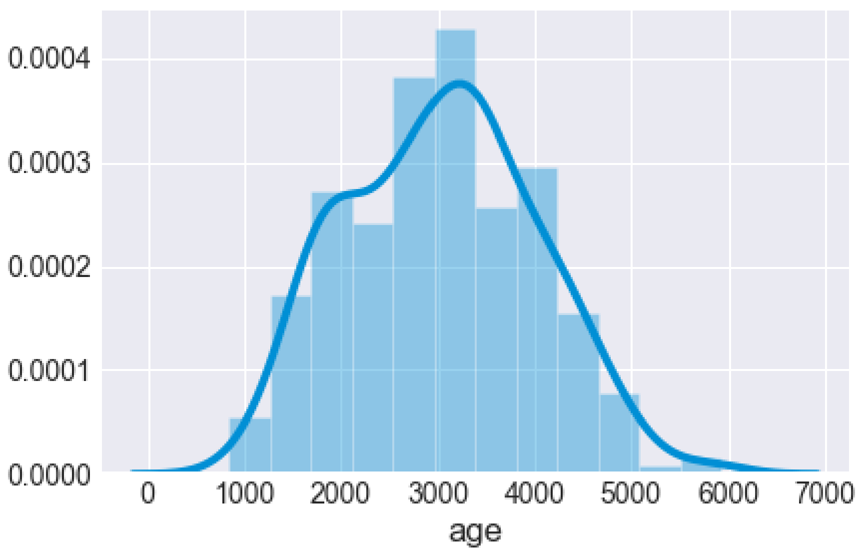
- The thumb rule is: If the skewness is between **-0.5 to +0.5** then we can say data is **fairly symmetrical.**

In [54]:

```
# plotting the density and histogram plot
sns.distplot(Square_age, hist=True)
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x27589fe9940>
```



b. Checking Skewness for feature "trestbps"

In [55]:

```
# Checking the skewness of "trestbps" attributes
df['trestbps'].skew()
```

Out[55]:

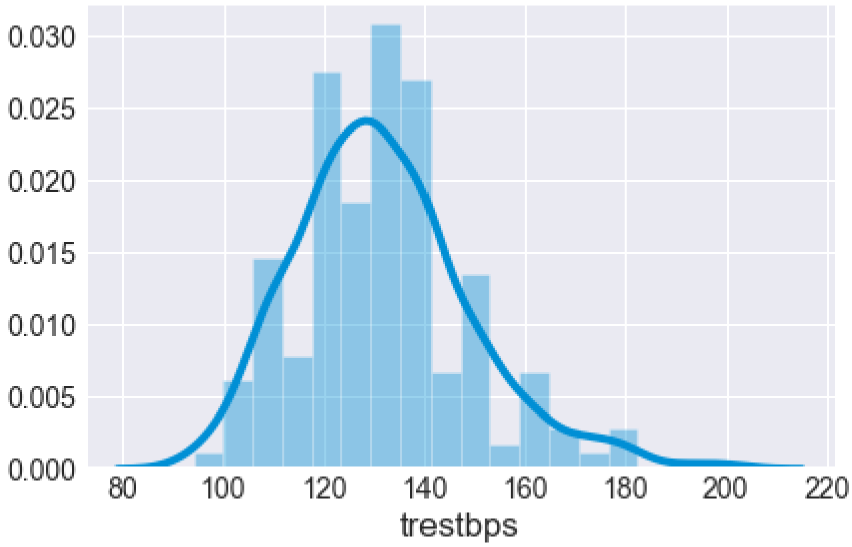
```
0.7137684379181465
```

In [56]:

```
# plotting the histogram for  "trestbps" attributes
sns.distplot(df['trestbps'], hist=True)
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758a0a41d0>
```



- feature "trestbps" is **right skewed**.

In [57]:

```
# performing the log transformation using numpy
log_trestbps = np.log(data['trestbps'])
log_trestbps
```

Out[57]:

```
0      4.976734
1      4.867534
2      4.867534
3      4.787492
4      4.787492
         ...
298    4.941642
299    4.700480
300    4.969813
301    4.867534
302    4.867534
Name: trestbps, Length: 303, dtype: float64
```

In [58]:

```
# checking the skewness after the log-transformation
log_trestbps.skew()
```

Out[58]:

```
0.2817574464672539
```

c. Checking Skewness for feature "chol"

In [59]:

```
# Checking the skewness of "chol" attributes
df['chol'].skew()
```

Out[59]:

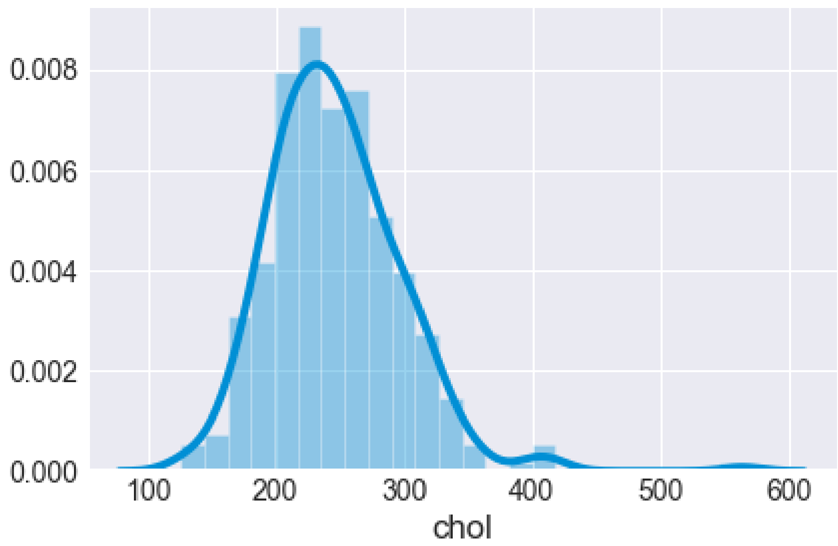
```
1.1434008206693387
```

In [60]:

```
# plotting the histogram for  "chol" attributes
sns.distplot(df['chol'], hist=True)
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758b5bf518>
```



- feature "trestbps" is **right skewed**.

In [61]:

```
# performing the log transformation using numpy
log_chol = np.log(data['chol'])
log_chol
```

Out[61]:

```
0      5.451038
1      5.521461
2      5.318120
3      5.463832
4      5.869297
         ...
298    5.484797
299    5.575949
300    5.262690
301    4.875197
302    5.463832
Name: chol, Length: 303, dtype: float64
```

In [62]:

```
# checking the skewness after the log-transformation
log_chol.skew()
```

Out[62]:

```
0.08666713455435988
```

d. Checking Skewness for feature "thalach"

In [63]:

```
# Checking the skewness of "thalach" attributes
df['thalach'].skew()
```

Out[63]:

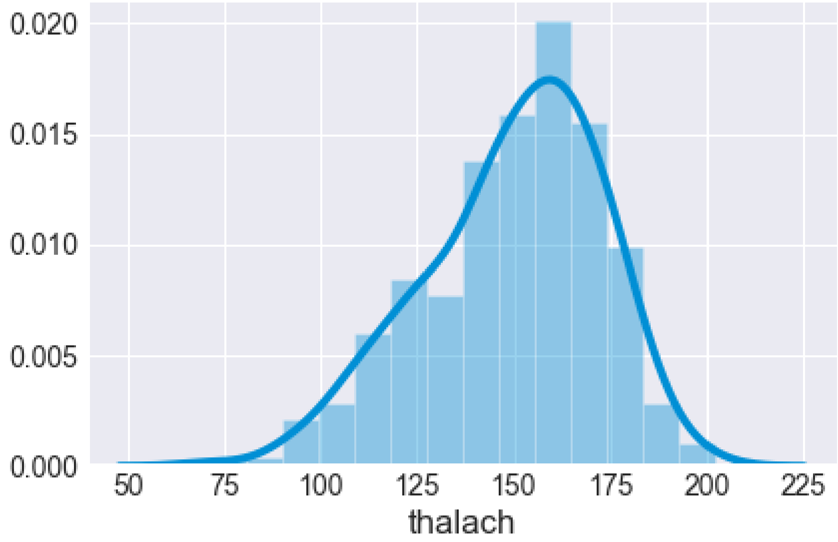
```
-0.5374096526832253
```

In [64]:

```
# plotting the histogram for  "thalach" attributes
sns.distplot(df['thalach'], hist=True)
```

Out[64]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758b65c5f8>
```



- feature "thalach" is **left skewed**.

In [65]:

```
# calculating the square for the column df['thalach'] column
Square_thalach = np.power(df['thalach'],3)
Square_thalach
```

Out[65]:

```
0      3375000
1      6539203
2      5088448
3      5639752
4      4330747
        ...
298    1860867
299    2299968
300    2803221
301    1520875
302    5268024
Name: thalach, Length: 303, dtype: int64
```

In [66]:

```
# checking the skewness
Square_thalach.skew()
```

Out[66]:

```
0.18034307083306292
```

e. Checking Skewness for feature "target"

In [67]:

```
# Checking the skewness of "target" column of dataset
df['target'].skew()
```

Out[67]:

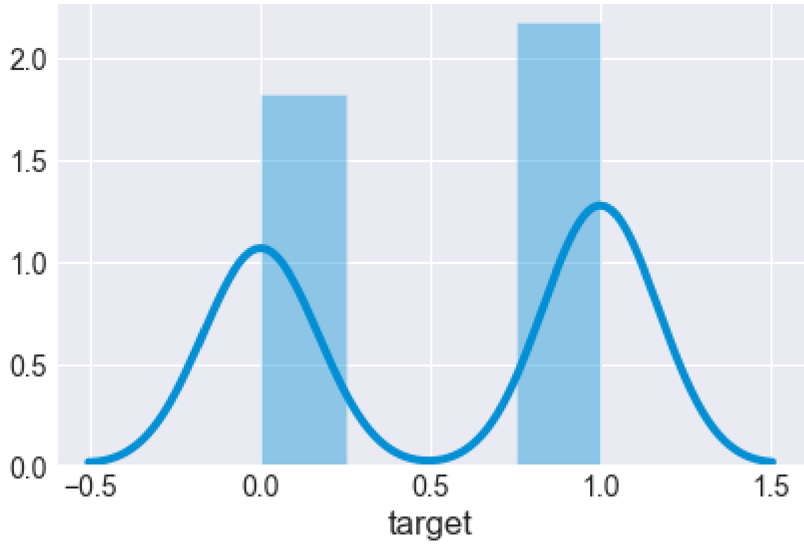
```
-0.17982105403495655
```

In [68]:

```
# density plot
sns.distplot(df['target'], hist = True)
```

Out[68]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758b681588>
```



- The thumb rule is: If the skewness is between **-0.5 to +0.5** then we can say data is **fairly symmetrical.**

7. Model building and Evaluation

In [69]:

```
X = df.drop(['target'], axis=1)     # Independant variable
y = df['target']          # Dependant variable
```

In [70]:

```
# split  data into training and testing sets of 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42)
```

In [71]:

```
# shape of X & Y test / train
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(212, 13) (91, 13) (212,) (91,)
```

a. Apply Random Forest Algorithm

In [72]:

```
model = RandomForestClassifier(n_estimators=200)
model.fit(X_train, y_train)
```

Out[72]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [73]:

```
# predicting X_test
pred_model = model.predict(X_test)
pred_model
```

Out[73]:

```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 1], dtype=int64)
```

In [74]:

```
# Check for accuracy of prediction
model.score(X_test, y_test)
```

Out[74]:

```
0.8131868131868132
```

In [75]:

```
# classification Report
print(classification_report(y_test, pred_model))
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79        41
           1       0.82      0.84      0.83        50

   micro avg       0.81      0.81      0.81        91
   macro avg       0.81      0.81      0.81        91
weighted avg       0.81      0.81      0.81        91
```

In [98]:

```
# Confusion Matrix
cf_matrix = confusion_matrix(y_test, pred_model)
cf_matrix
```

Out[98]:

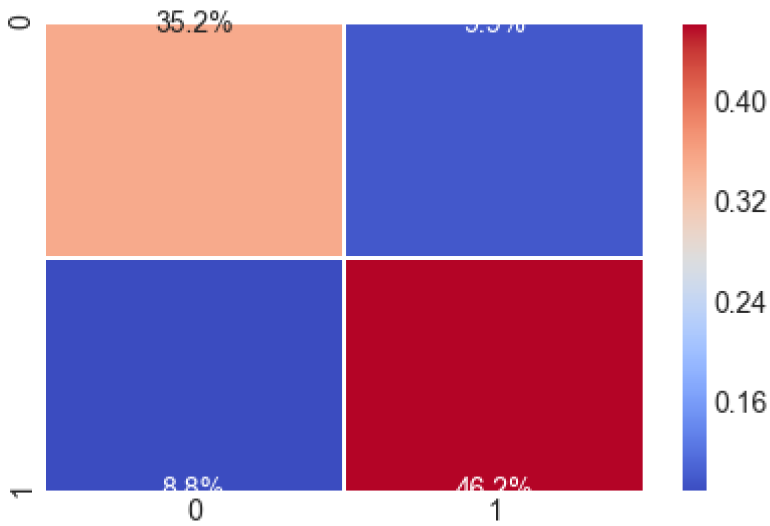
```
array([[32,  9],
       [ 8, 42]], dtype=int64)
```

In [114]:

```
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True, cmap='coolwarm', fmt='.1%', linewidth=
```

Out[114]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x275feb2bdd8>
```



b. SVM (Support Vector Machine)

In [77]:

```
svm = svm.SVC(kernel="linear")
svm.fit(X_train, y_train)
```

Out[77]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

In [78]:

```
# predicting X_test
pred_svm = svm.predict(X_test)
pred_svm
```

Out[78]:

```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 0, 1], dtype=int64)
```

In [79]:

```
# Check for accuracy of prediction
svm.score(X_test, y_test)
```

Out[79]:

```
0.8131868131868132
```

In [80]:

```
# classification Report
print(classification_report(y_test, pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79        41
           1       0.82      0.84      0.83        50

   micro avg       0.81      0.81      0.81        91
   macro avg       0.81      0.81      0.81        91
weighted avg       0.81      0.81      0.81        91
```

In [106]:

```
# Confusion Matrix
cf_matrix_svm = confusion_matrix(y_test, pred_svm)
cf_matrix_svm
```

Out[106]:

```
array([[32,  9],
       [ 8, 42]], dtype=int64)
```

In [112]:

```
plt.figure(figsize=(6,4))
sns.heatmap(cf_matrix_svm/np.sum(cf_matrix_svm), annot=True, cmap='coolwarm', fmt='.1%', li
```

Out[112]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2758cbdd160>
```