# Heart Disease Prediction

## Importing libraries ¶

In [1]:
```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor


%matplotlib inline
```

# Reading Exploring Dataset

In [2]:
```
data = pd.read_csv('heart.csv')
```

In [3]:
```
# shape
print(data.shape)
```

```
(1025, 14)
```

In [4]:
```
data.dtypes
```

Out[4]:
```
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 [5]:
```python
# descriptions
print(data.describe())
```

```
               age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
25%       0.000000     0.000000   132.000000     0.000000     0.000000
50%       0.000000     1.000000   152.000000     0.000000     0.800000
75%       0.000000     1.000000   166.000000     1.000000     1.800000
max       1.000000     2.000000   202.000000     1.000000     6.200000

             slope           ca         thal       target
count  1025.000000  1025.000000  1025.000000  1025.000000
mean      1.385366     0.754146     2.323902     0.513171
std       0.617755     1.030798     0.620660     0.500070
min       0.000000     0.000000     0.000000     0.000000
25%       1.000000     0.000000     2.000000     0.000000
50%       1.000000     0.000000     2.000000     1.000000
75%       2.000000     1.000000     3.000000     1.000000
max       2.000000     4.000000     3.000000     1.000000
```

In [6]:
```python
# Checking for missing values
data.isna().sum()
```

Out[6]:
```
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 [7]:
```
# Checking for missing values another method
pd.isnull(data).any()
```

Out[7]:
```
age         False
sex         False
cp          False
trestbps    False
chol        False
fbs         False
restecg     False
thalach     False
exang       False
oldpeak     False
slope       False
ca          False
thal        False
target      False
dtype: bool
```

In [8]:
```
#counting sex , here male=1 and female = 0
data['sex'].value_counts()
```

Out[8]:
```
1    713
0    312
Name: sex, dtype: int64
```

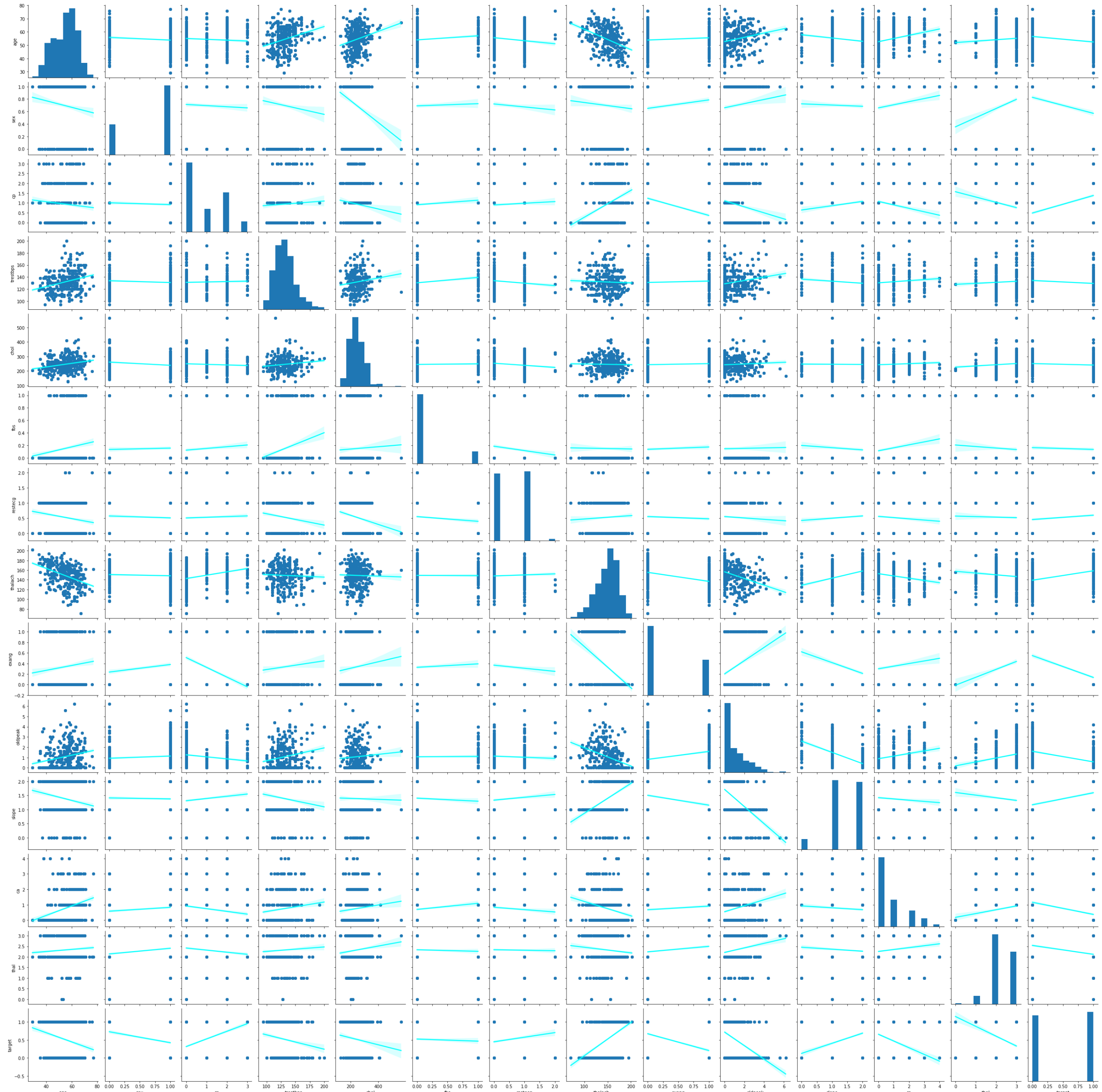
In [56]:
```
# count by presence of disease , here 1 for yes and 0 for no
data['target'].value_counts()

#print(data.groupby('target').size())   #this is another method
```

Out[56]:
```
1    526
0    499
Name: target, dtype: int64
```

## Initial visualization

In [10]:

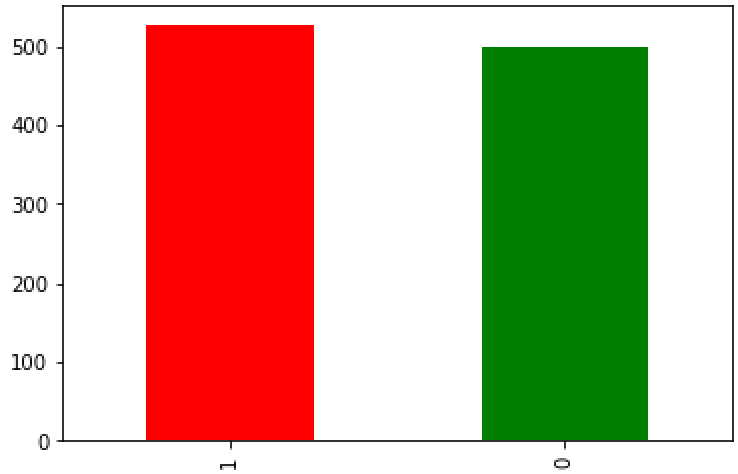
```
%%time
sns.pairplot(data, kind='reg', plot_kws={'line_kws':{'color': 'cyan'}})
plt.show()
```



```
Wall time: 1min 25s
```

In [12]:
```
data.target.value_counts().plot(kind="bar", color=["red", "green"])
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x20ed87c85c8>



In [13]:
```
categorical_val = []
continous_val = []
for column in data.columns:
    print('==============================')
    print(f"{column} : {data[column].unique()}")
    if len(data[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

```
==============================
age : [52 53 70 61 62 58 55 46 54 71 43 34 51 50 60 67 45 63 42 44 56 57 59 64
 65 41 66 38 49 48 29 37 47 68 76 40 39 77 69 35 74]
==============================
sex : [1 0]
==============================
cp : [0 1 2 3]
==============================
trestbps : [125 140 145 148 138 100 114 160 120 122 112 132 118 128 124 106 104 135
 130 136 180 129 150 178 146 117 152 154 170 134 174 144 108 123 110 142
 126 192 115  94 200 165 102 105 155 172 164 156 101]
==============================
chol : [212 203 174 294 248 318 289 249 286 149 341 210 298 204 308 266 244 211
 185 223 208 252 209 307 233 319 256 327 169 131 269 196 231 213 271 263
 229 360 258 330 342 226 228 278 230 283 241 175 188 217 193 245 232 299
 288 197 315 215 164 326 207 177 257 255 187 201 220 268 267 236 303 282
 126 309 186 275 281 206 335 218 254 295 417 260 240 302 192 225 325 235
 274 234 182 167 172 321 300 199 564 157 304 222 184 354 160 247 239 246
 409 293 180 250 221 200 227 243 311 261 242 205 306 219 353 198 394 183
 237 224 265 313 340 259 270 216 264 276 322 214 273 253 176 284 305 168
 407 290 277 262 195 166 178 141]
==============================
fbs : [0 1]
==============================
restecg : [1 0 2]
==============================
thalach : [168 155 125 161 106 122 140 145 144 116 136 192 156 142 109 162 165 148
 172 173 146 179 152 117 115 112 163 147 182 105 150 151 169 166 178 132
 160 123 139 111 180 164 202 157 159 170 138 175 158 126 143 141 167  95
 190 118 103 181 108 177 134 120 171 149 154 153  88 174 114 195 133  96
 124 131 185 194 128 127 186 184 188 130  71 137  99 121 187  97  90 129
 113]
==============================
exang : [0 1]
==============================
oldpeak : [1.  3.1 2.6 0.  1.9 4.4 0.8 3.2 1.6 3.  0.7 4.2 1.5 2.2 1.1 0.3 0.4 0.6
 3.4 2.8 1.2 2.9 3.6 1.4 0.2 2.  5.6 0.9 1.8 6.2 4.  2.5 0.5 0.1 2.1 2.4
 3.8 2.3 1.3 3.5]
==============================
slope : [2 0 1]
==============================
```

```
ca : [2 0 1 3 4]
==============================
thal : [3 2 1 0]
==============================
target : [0 1]
```

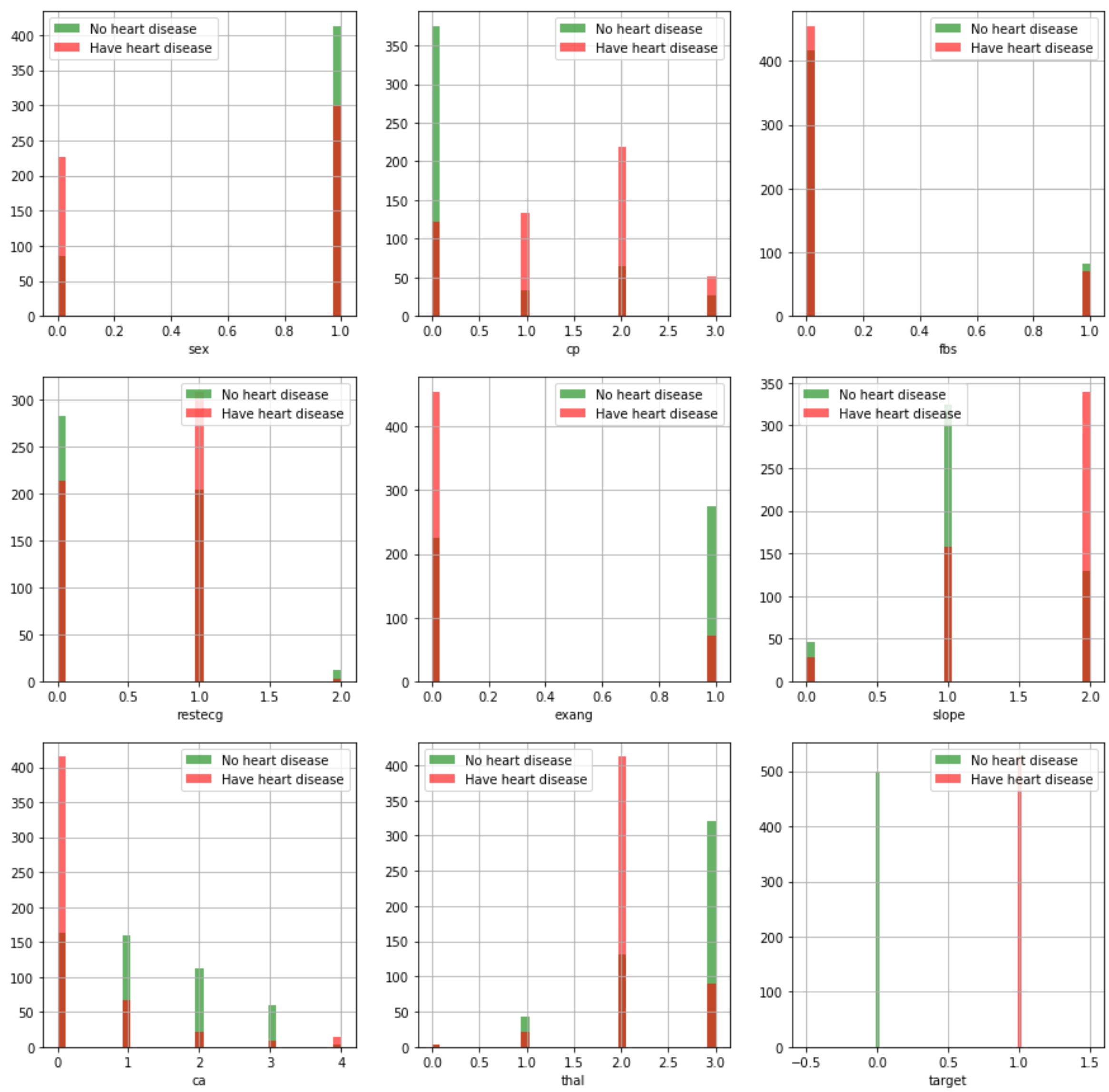
In [14]:

```
categorical_val
```

Out[14]: ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']

In [15]:

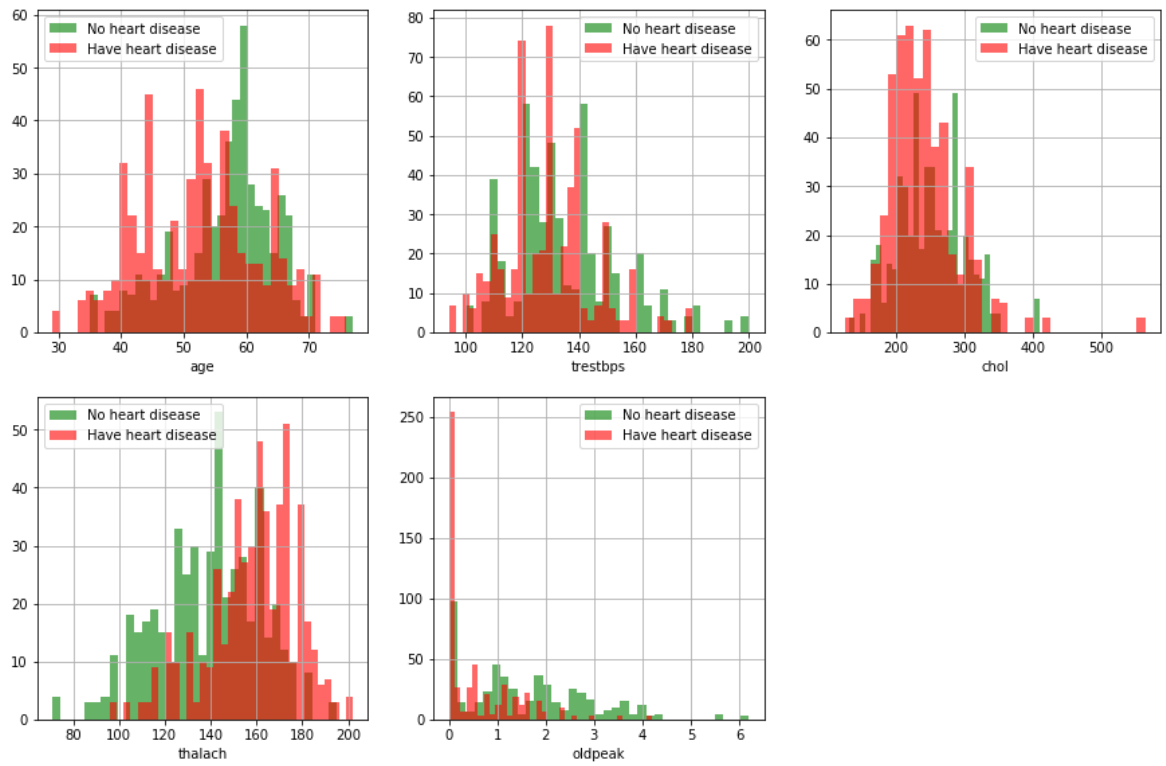
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1):
    plt.subplot(3, 3, i)
    data[data["target"] == 0][column].hist(bins=35, color='green', label='No
    data[data["target"] == 1][column].hist(bins=35, color='red', label='Have
    plt.legend()
    plt.xlabel(column)
```



In [16]:
```
continous_val
```

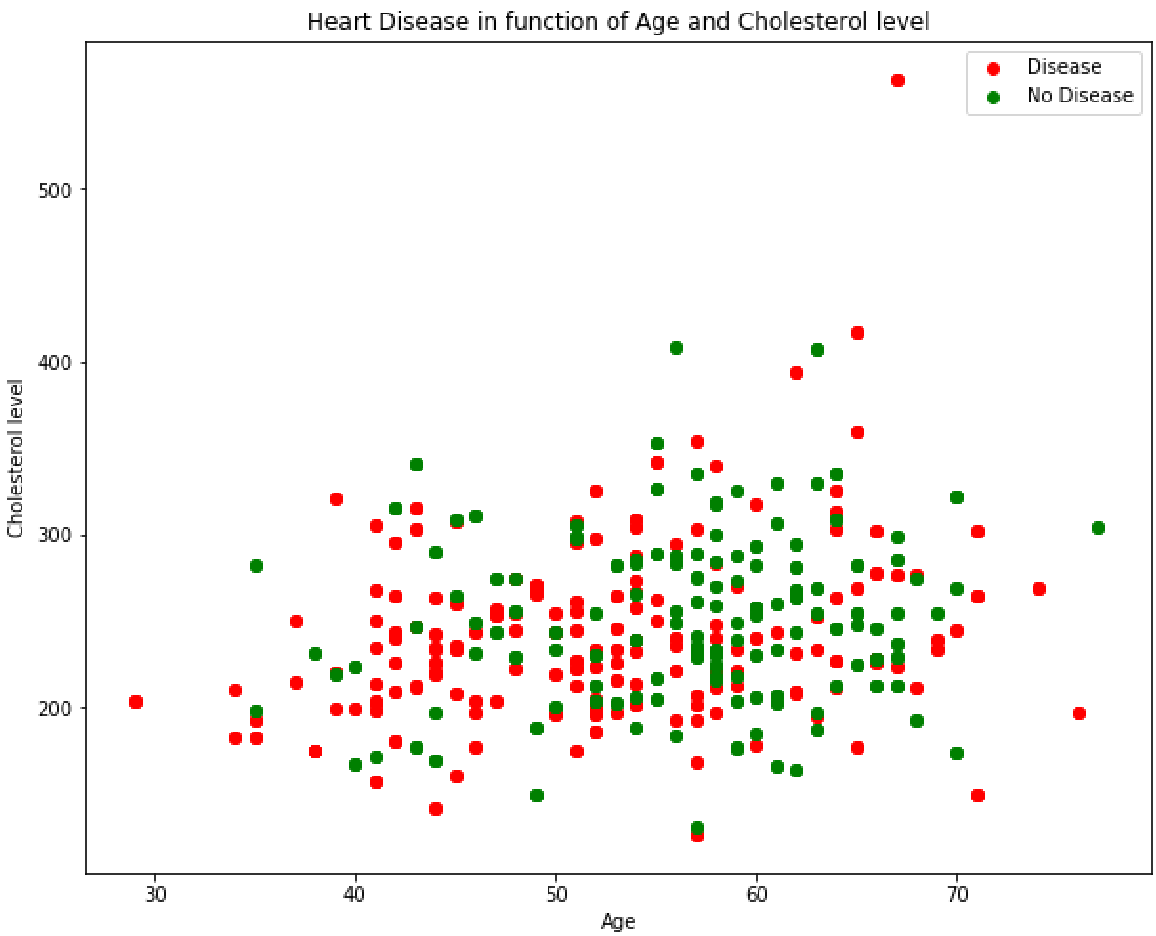
Out[16]: ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']

In [17]:
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(continous_val, 1):
    plt.subplot(3, 3, i)
    data[data["target"] == 0][column].hist(bins=35, color='green', label='No
    data[data["target"] == 1][column].hist(bins=35, color='red', label='Have
    plt.legend()
    plt.xlabel(column)
```



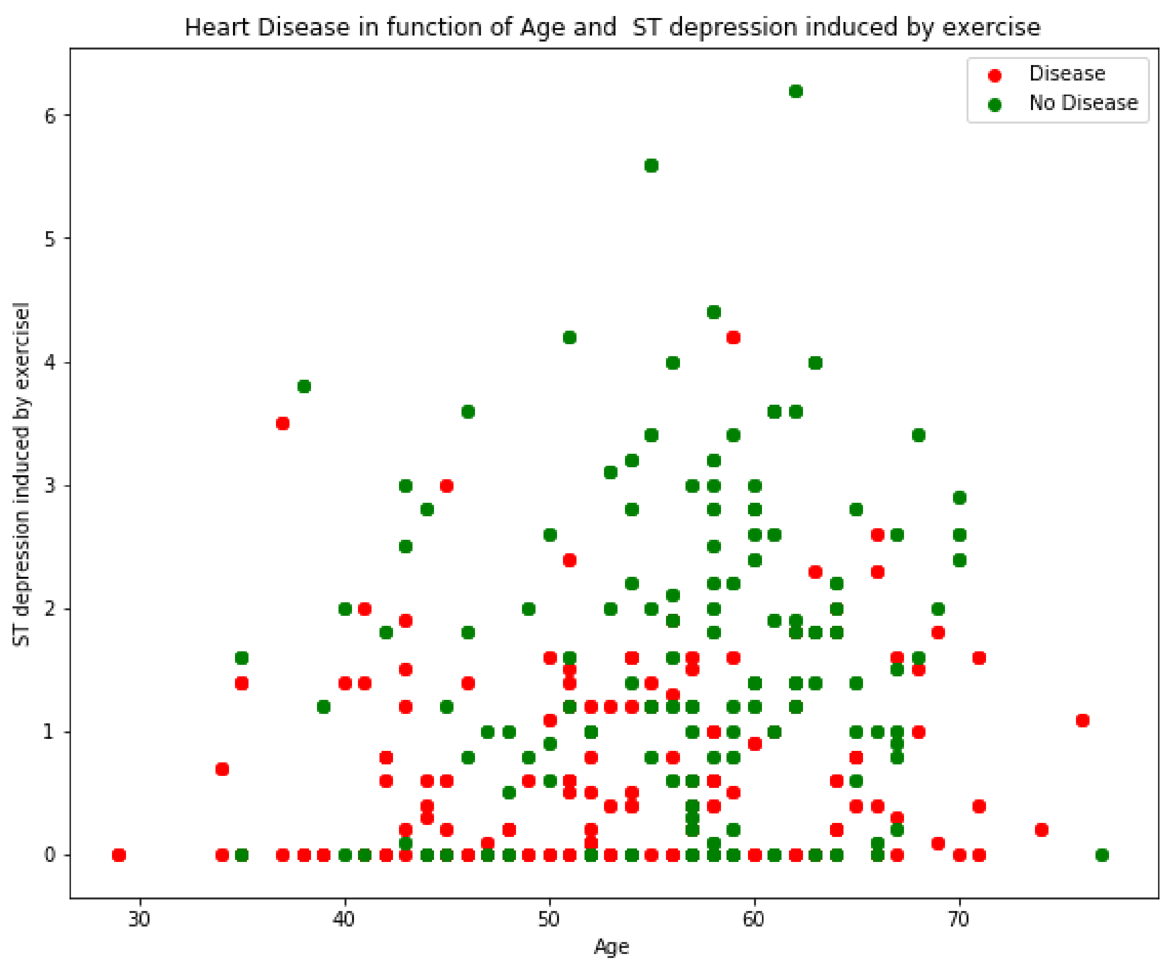
In [18]:

```
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with postivie examples
plt.scatter(data.age[data.target==1],
            data.chol[data.target==1],
            c="red")

# Scatter with negative examples
plt.scatter(data.age[data.target==0],
            data.chol[data.target==0],
            c="green")
# Add some helpful info
plt.title("Heart Disease in function of Age and Cholesterol level")
plt.xlabel("Age")
plt.ylabel("Cholesterol level")
plt.legend(["Disease", "No Disease"]);
```



In [19]:

```
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with postivie examples
plt.scatter(data.age[data.target==1],
            data.oldpeak[data.target==1],
            c="red")

# Scatter with negative examples
plt.scatter(data.age[data.target==0],
            data.oldpeak[data.target==0],
            c="green")
# Add some helpful info
plt.title("Heart Disease in function of Age and  ST depression induced by exe
plt.xlabel("Age")
plt.ylabel("ST depression induced by exercisel")
plt.legend(["Disease", "No Disease"]);
```



In [20]:

```
data.corr() # Pearson Correlation Coefficients
```

Out[20]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach |
|---|---|---|---|---|---|---|---|---|
| **age** | 1.000000 | -0.103240 | -0.071966 | 0.271121 | 0.219823 | 0.121243 | -0.132696 | -0.390227 |
| **sex** | -0.103240 | 1.000000 | -0.041119 | -0.078974 | -0.198258 | 0.027200 | -0.055117 | -0.049365 |
| **cp** | -0.071966 | -0.041119 | 1.000000 | 0.038177 | -0.081641 | 0.079294 | 0.043581 | 0.306839 |
| **trestbps** | 0.271121 | -0.078974 | 0.038177 | 1.000000 | 0.127977 | 0.181767 | -0.123794 | -0.039264 |
| **chol** | 0.219823 | -0.198258 | -0.081641 | 0.127977 | 1.000000 | 0.026917 | -0.147410 | -0.021772 |
| **fbs** | 0.121243 | 0.027200 | 0.079294 | 0.181767 | 0.026917 | 1.000000 | -0.104051 | -0.008866 |
| **restecg** | -0.132696 | -0.055117 | 0.043581 | -0.123794 | -0.147410 | -0.104051 | 1.000000 | 0.048411 |
| **thalach** | -0.390227 | -0.049365 | 0.306839 | -0.039264 | -0.021772 | -0.008866 | 0.048411 | 1.000000 |
| **exang** | 0.088163 | 0.139157 | -0.401513 | 0.061197 | 0.067382 | 0.049261 | -0.065606 | -0.380281 |
| **oldpeak** | 0.208137 | 0.084687 | -0.174733 | 0.187434 | 0.064880 | 0.010859 | -0.050114 | -0.349796 |
| **slope** | -0.169105 | -0.026666 | 0.131633 | -0.120445 | -0.014248 | -0.061902 | 0.086086 | 0.395308 |
| **ca** | 0.271551 | 0.111729 | -0.176206 | 0.104554 | 0.074259 | 0.137156 | -0.078072 | -0.207888 |
| **thal** | 0.072297 | 0.198424 | -0.163341 | 0.059276 | 0.100244 | -0.042177 | -0.020504 | -0.098068 |
| **target** | -0.229324 | -0.279501 | 0.434854 | -0.138772 | -0.099966 | -0.041164 | 0.134468 | 0.422895 |

In [21]:

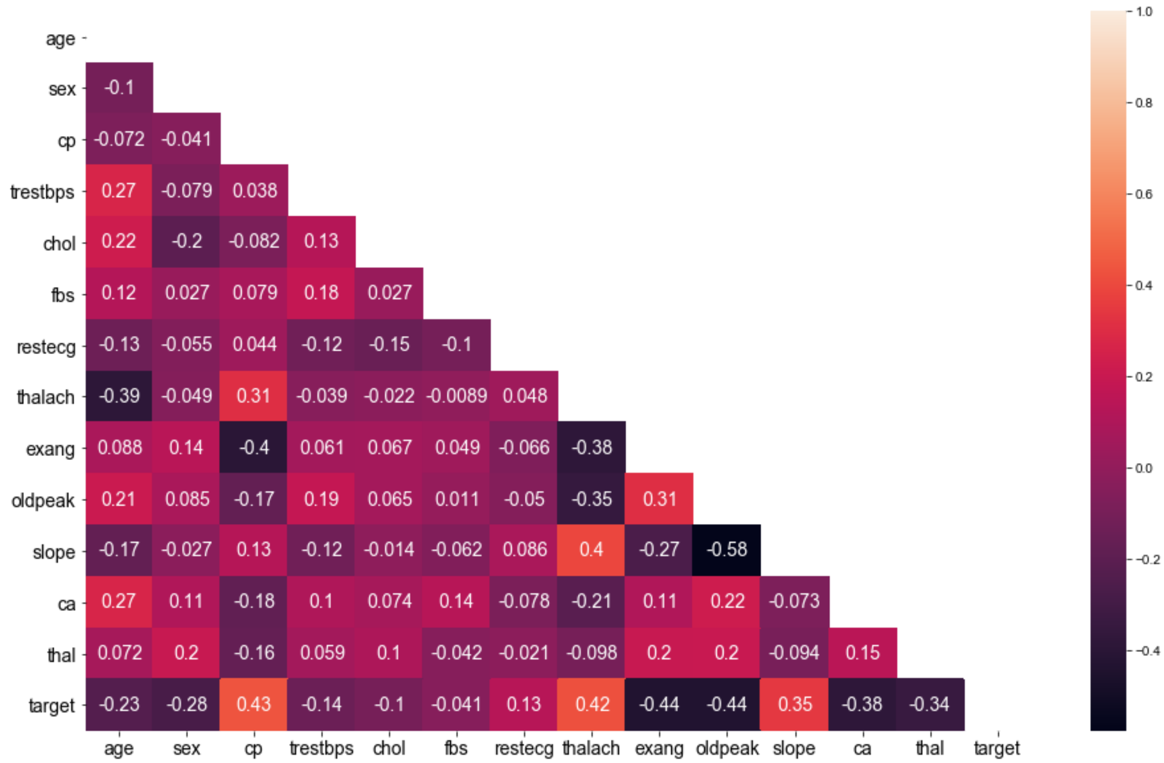
```
mask = np.zeros_like(data.corr())
triangle_indices = np.triu_indices_from(mask)
mask[triangle_indices] = True
mask
```

Out[21]:

```
array([[1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

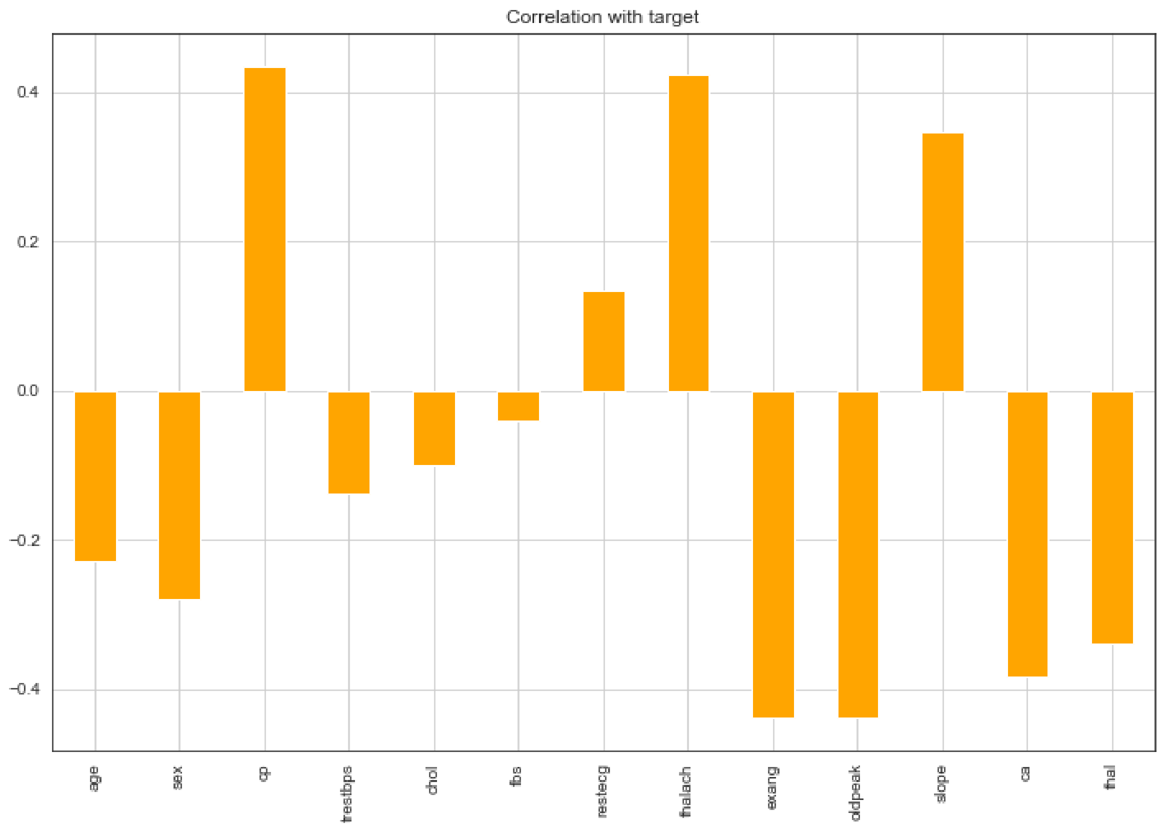
In [22]:

```
plt.figure(figsize=(16,10))
sns.heatmap(data.corr(), mask=mask, annot=True, annot_kws={"size": 14})
sns.set_style('white')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



In [23]:
```
data.drop('target', axis=1).corrwith(data.target).plot(kind='bar', grid=True,
                                                        title="Correlation with ta
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x20ed5cc1748>




In [24]:
```
categorical_val.remove('target')
dataset = pd.get_dummies(data, columns = categorical_val)
```

In [25]:
```
dataset.head()
```

Out[25]:

| | age | trestbps | chol | thalach | oldpeak | target | sex_0 | sex_1 | cp_0 | cp_1 | ... | slope_2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 52 | 125 | 212 | 168 | 1.0 | 0 | 0 | 1 | 1 | 0 | ... | 1 |
| **1** | 53 | 140 | 203 | 155 | 3.1 | 0 | 0 | 1 | 1 | 0 | ... | 0 |
| **2** | 70 | 145 | 174 | 125 | 2.6 | 0 | 0 | 1 | 1 | 0 | ... | 0 |
| **3** | 61 | 148 | 203 | 161 | 0.0 | 0 | 0 | 1 | 1 | 0 | ... | 1 |
| **4** | 62 | 138 | 294 | 106 | 1.9 | 0 | 1 | 0 | 1 | 0 | ... | 0 |

5 rows × 31 columns

In [26]:
```
print(data.columns)
print(dataset.columns)
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
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')
```

In [27]:
```
dataset.describe()
```

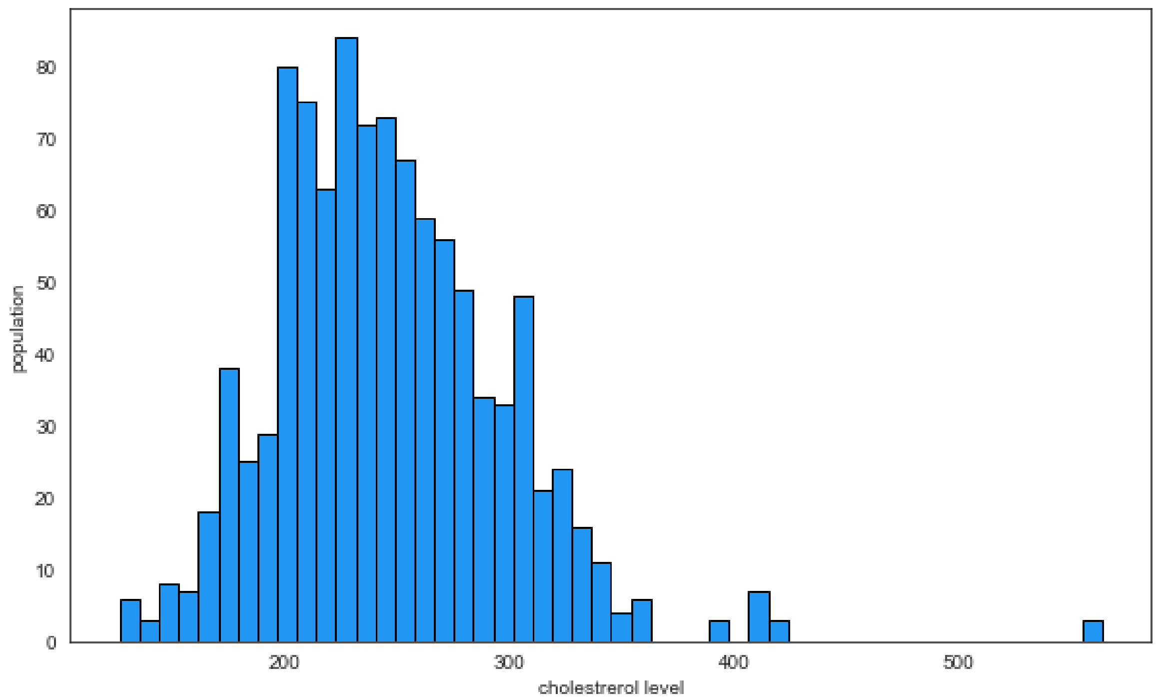
Out[27]:

| | age | trestbps | chol | thalach | oldpeak | target | se |
|---|---|---|---|---|---|---|---|
| **count** | 1025.000000 | 1025.000000 | 1025.00000 | 1025.000000 | 1025.000000 | 1025.000000 | 1025.000 |
| **mean** | 54.434146 | 131.611707 | 246.00000 | 149.114146 | 1.071512 | 0.513171 | 0.304 |
| **std** | 9.072290 | 17.516718 | 51.59251 | 23.005724 | 1.175053 | 0.500070 | 0.460 |
| **min** | 29.000000 | 94.000000 | 126.00000 | 71.000000 | 0.000000 | 0.000000 | 0.000 |
| **25%** | 48.000000 | 120.000000 | 211.00000 | 132.000000 | 0.000000 | 0.000000 | 0.000 |
| **50%** | 56.000000 | 130.000000 | 240.00000 | 152.000000 | 0.800000 | 1.000000 | 0.000 |
| **75%** | 61.000000 | 140.000000 | 275.00000 | 166.000000 | 1.800000 | 1.000000 | 1.000 |
| **max** | 77.000000 | 200.000000 | 564.00000 | 202.000000 | 6.200000 | 1.000000 | 1.000 |

8 rows × 31 columns

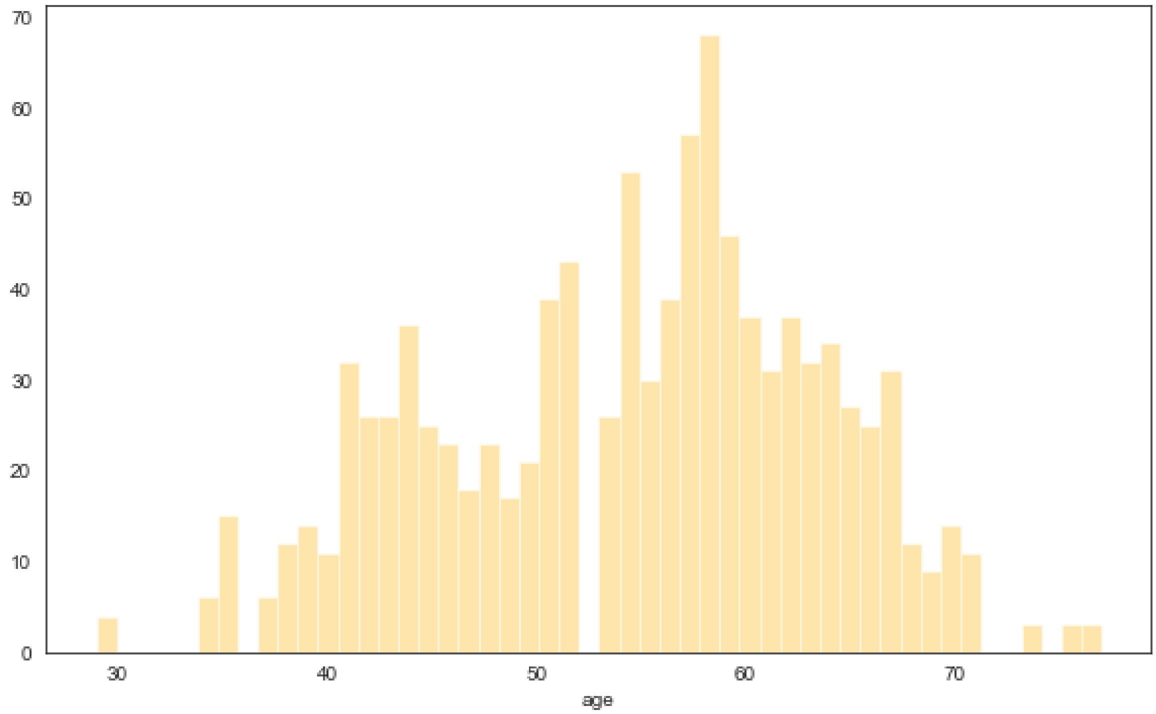
# Visualising Data - Histograms, Distributions and Bar Charts

In [28]:
```
plt.figure(figsize=(10, 6))
plt.hist(dataset['chol'], bins=50, ec='black', color='#2196f3')
plt.xlabel('cholestrerol level')
plt.ylabel('population')
plt.show()
```



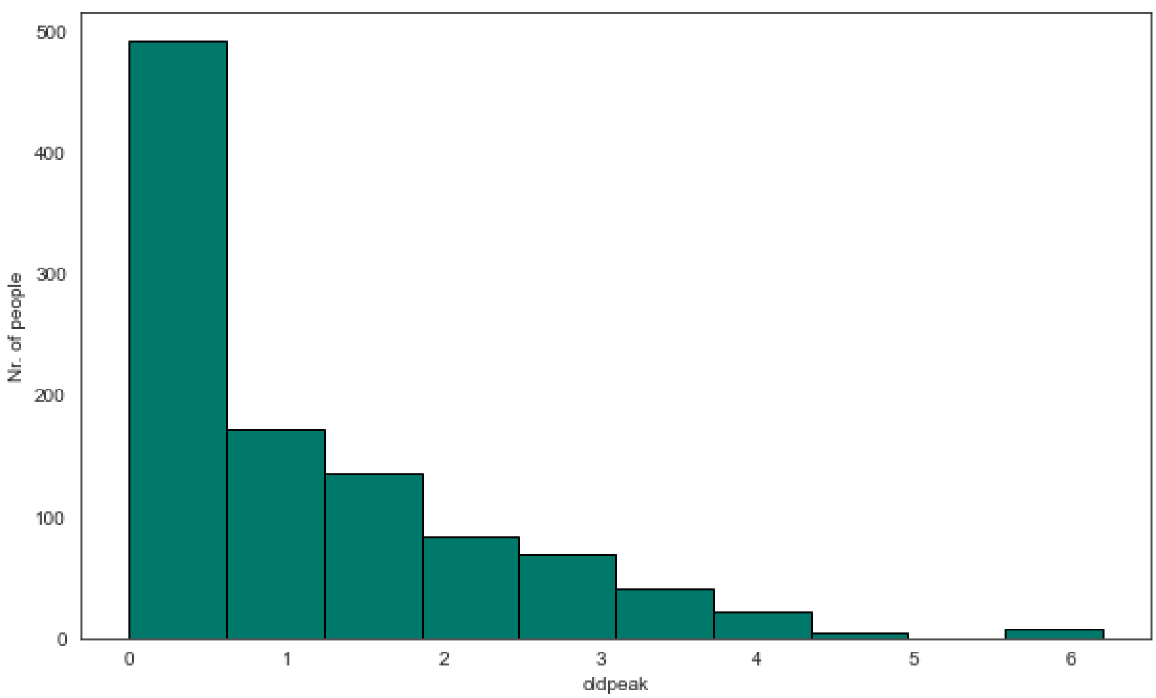
In [29]:

```
plt.figure(figsize=(10, 6))
sns.distplot(dataset['age'], bins=50, hist=True, kde=False, color='#fbc02d')
plt.show()
```




In [30]:

```
plt.figure(figsize=(10, 6))
plt.hist(dataset['oldpeak'], ec='black', color='#00796b')
plt.xlabel('oldpeak')
plt.ylabel('Nr. of people')
plt.show()
```



In [31]:

```
from sklearn.preprocessing import StandardScaler

s_sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[col_to_scale] = s_sc.fit_transform(dataset[col_to_scale])
```

In [32]:

```
dataset.head()
```

Out[32]:

| | age | trestbps | chol | thalach | oldpeak | target | sex_0 | sex_1 | cp_0 | cp_1 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.268437 | -0.377636 | -0.659332 | 0.821321 | -0.060888 | 0 | 0 | 1 | 1 | 0 | ... |
| **1** | -0.158157 | 0.479107 | -0.833861 | 0.255968 | 1.727137 | 0 | 0 | 1 | 1 | 0 | ... |
| **2** | 1.716595 | 0.764688 | -1.396233 | -1.048692 | 1.301417 | 0 | 0 | 1 | 1 | 0 | ... |
| **3** | 0.724079 | 0.936037 | -0.833861 | 0.516900 | -0.912329 | 0 | 0 | 1 | 1 | 0 | ... |
| **4** | 0.834359 | 0.364875 | 0.930822 | -1.874977 | 0.705408 | 0 | 1 | 0 | 1 | 0 | ... |

5 rows × 31 columns

In [33]:

```
dataset.describe()
```

Out[33]:

| | age | trestbps | chol | thalach | oldpeak | target |
|---|---|---|---|---|---|---|
| **count** | 1.025000e+03 | 1.025000e+03 | 1.025000e+03 | 1.025000e+03 | 1.025000e+03 | 1025.000000 |
| **mean** | -3.323629e-16 | -6.590934e-16 | 6.282238e-18 | -3.812668e-16 | -2.341217e-16 | 0.513171 |
| **std** | 1.000488e+00 | 1.000488e+00 | 1.000488e+00 | 1.000488e+00 | 1.000488e+00 | 0.500070 |
| **min** | -2.804866e+00 | -2.148237e+00 | -2.327054e+00 | -3.397080e+00 | -9.123291e-01 | 0.000000 |
| **25%** | -7.095548e-01 | -6.632165e-01 | -6.787242e-01 | -7.442713e-01 | -9.123291e-01 | 0.000000 |
| **50%** | 1.726817e-01 | -9.205458e-02 | -1.163527e-01 | 1.255019e-01 | -2.311765e-01 | 1.000000 |
| **75%** | 7.240794e-01 | 4.791073e-01 | 5.623715e-01 | 7.343432e-01 | 6.202642e-01 | 1.000000 |
| **max** | 2.488552e+00 | 3.906079e+00 | 6.166694e+00 | 2.299935e+00 | 4.366603e+00 | 1.000000 |

8 rows × 31 columns

In [90]:

```
dataset.corr() # Pearson Correlation Coefficients
```

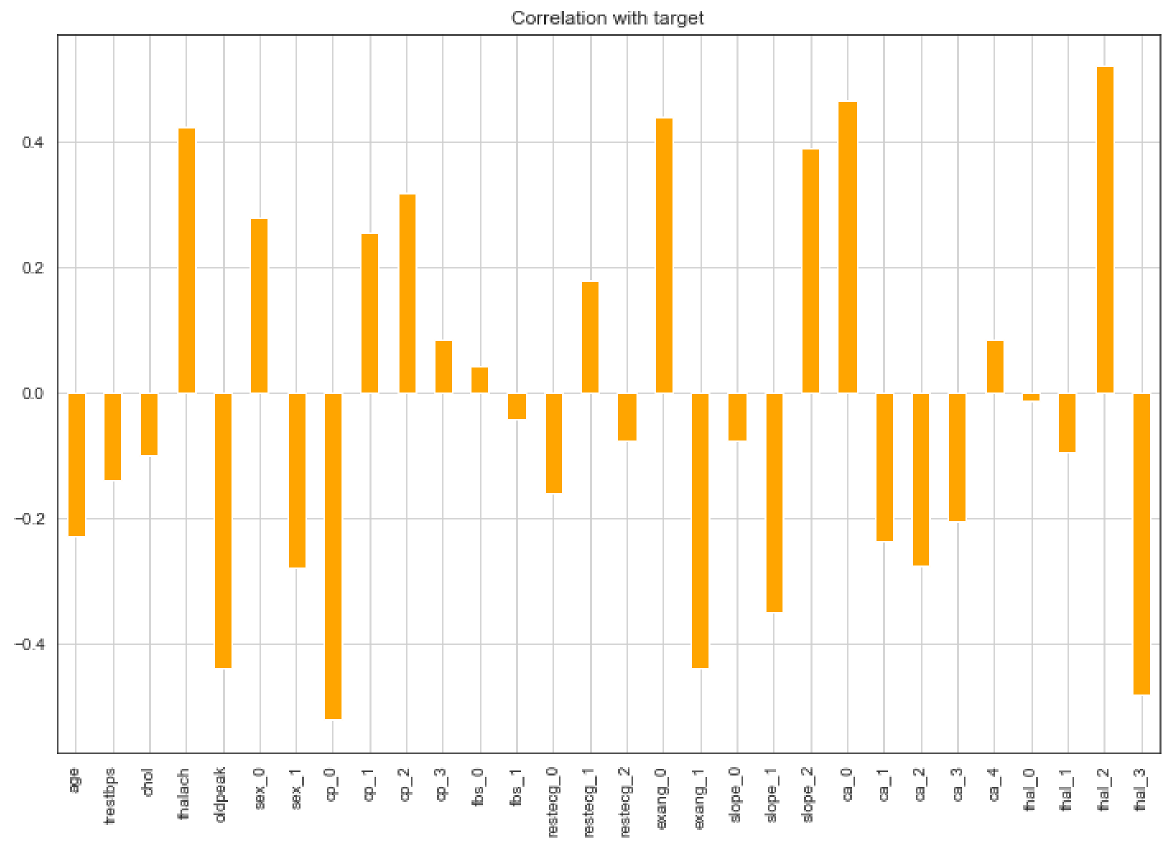
Out[90]:

| | age | trestbps | chol | thalach | oldpeak | target | sex_0 | sex_ |
|---|---|---|---|---|---|---|---|---|
| **age** | 1.000000 | 0.271121 | 0.219823 | -0.390227 | 0.208137 | -0.229324 | 0.103240 | -0.10324 |
| **trestbps** | 0.271121 | 1.000000 | 0.127977 | -0.039264 | 0.187434 | -0.138772 | 0.078974 | -0.07897 |
| **chol** | 0.219823 | 0.127977 | 1.000000 | -0.021772 | 0.064880 | -0.099966 | 0.198258 | -0.19825 |
| **thalach** | -0.390227 | -0.039264 | -0.021772 | 1.000000 | -0.349796 | 0.422895 | 0.049365 | -0.04936 |
| **oldpeak** | 0.208137 | 0.187434 | 0.064880 | -0.349796 | 1.000000 | -0.438441 | -0.084687 | 0.08468 |
| **target** | -0.229324 | -0.138772 | -0.099966 | 0.422895 | -0.438441 | 1.000000 | 0.279501 | -0.27950 |
| **sex_0** | 0.103240 | 0.078974 | 0.198258 | 0.049365 | -0.084687 | 0.279501 | 1.000000 | -1.00000 |
| **sex_1** | -0.103240 | -0.078974 | -0.198258 | -0.049365 | 0.084687 | -0.279501 | -1.000000 | 1.00000 |
| **cp_0** | 0.144501 | 0.033336 | 0.075143 | -0.382343 | 0.303432 | -0.519621 | -0.077558 | 0.07755 |
| **cp_1** | -0.155137 | -0.087992 | -0.011117 | 0.250678 | -0.280812 | 0.255288 | 0.035405 | -0.03540 |
| **cp_2** | -0.062574 | -0.054250 | -0.045654 | 0.161594 | -0.151284 | 0.319504 | 0.106842 | -0.10684 |
| **cp_3** | 0.049622 | 0.152188 | -0.049381 | 0.099348 | 0.074983 | 0.085054 | -0.083960 | 0.08396 |
| **fbs_0** | -0.121243 | -0.181767 | -0.026917 | 0.008866 | -0.010859 | 0.041164 | 0.027200 | -0.02720 |
| **fbs_1** | 0.121243 | 0.181767 | 0.026917 | -0.008866 | 0.010859 | -0.041164 | -0.027200 | 0.02720 |
| **restecg_0** | 0.158063 | 0.145613 | 0.167019 | -0.080033 | 0.096832 | -0.160308 | -0.030893 | 0.03089 |
| **restecg_1** | -0.175956 | -0.160461 | -0.178332 | 0.108908 | -0.140693 | 0.178573 | 0.003595 | -0.00359 |
| **restecg_2** | 0.074802 | 0.062100 | 0.047423 | -0.120381 | 0.182811 | -0.076357 | 0.113602 | -0.11360 |
| **exang_0** | -0.088163 | -0.061197 | -0.067382 | 0.380281 | -0.310844 | 0.438029 | 0.139157 | -0.13915 |
| **exang_1** | 0.088163 | 0.061197 | 0.067382 | -0.380281 | 0.310844 | -0.438029 | -0.139157 | 0.13915 |
| **slope_0** | 0.034451 | 0.113408 | -0.043568 | -0.065811 | 0.393521 | -0.075227 | -0.045260 | 0.04526 |
| **slope_1** | 0.173471 | 0.031390 | 0.062809 | -0.420784 | 0.303453 | -0.349417 | 0.013950 | -0.01395 |
| **slope_2** | -0.191688 | -0.090362 | -0.040292 | 0.455748 | -0.508445 | 0.389140 | 0.009537 | -0.00953 |
| **ca_0** | -0.354714 | -0.053888 | -0.085077 | 0.273016 | -0.206092 | 0.465981 | 0.119980 | -0.11998 |
| **ca_1** | 0.189677 | -0.069120 | 0.018573 | -0.202614 | -0.013949 | -0.235299 | -0.101220 | 0.10122 |
| **ca_2** | 0.218247 | 0.095713 | 0.061398 | -0.044465 | 0.223046 | -0.276566 | 0.026489 | -0.02648 |
| **ca_3** | 0.162754 | 0.082236 | 0.110765 | -0.181157 | 0.188517 | -0.205720 | -0.059255 | 0.05925 |
| **ca_4** | -0.130087 | 0.019086 | -0.106299 | 0.068462 | -0.109960 | 0.085639 | -0.088441 | 0.08844 |
| **thal_0** | -0.018340 | -0.017106 | -0.059268 | -0.038534 | -0.035308 | -0.014035 | 0.022379 | -0.02237 |
| **thal_1** | 0.048565 | 0.076197 | -0.085388 | -0.148055 | 0.106853 | -0.095541 | -0.135659 | 0.13565 |
| **thal_2** | -0.127881 | -0.139099 | -0.012472 | 0.284543 | -0.338063 | 0.519543 | 0.367115 | -0.36711 |
| **thal_3** | 0.109369 | 0.106942 | 0.064841 | -0.210261 | 0.297545 | -0.479709 | -0.310740 | 0.31074 |

31 rows × 31 columns

In [93]:
```
dataset.drop('target', axis=1).corrwith(dataset.target).plot(kind='bar', grid
                                                   title="Correlation with ta
```

Out[93]: <matplotlib.axes._subplots.AxesSubplot at 0x20ed46ace48>




In [34]:
```
dataset['target'].corr(dataset['age'])
```

Out[34]: -0.22932355126761098

In [35]:
```
dataset['target'].corr(dataset['chol'])
```

Out[35]: -0.0999655942325411

In [36]:
```
dataset['target'].corr(dataset['trestbps'])
```

Out[36]: -0.13877173373730095

In [37]:
```
dataset['target'].corr(dataset['thalach'])
```

Out[37]: 0.4228954964828711

## machine learning algorithms

In [38]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        print("Train Result:\n================================================
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_train, pred) * 100:.2f}
        print(f"\t\t\tRecall Score: {recall_score(y_train, pred) * 100:.2f}%"
        print(f"\t\t\tF1 score: {f1_score(y_train, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        print("Test Result:\n=================================================
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print("Classification Report:", end='')
        print(f"\tPrecision Score: {precision_score(y_test, pred) * 100:.2f}%
        print(f"\t\t\tRecall Score: {recall_score(y_test, pred) * 100:.2f}%")
        print(f"\t\t\tF1 score: {f1_score(y_test, pred) * 100:.2f}%")
        print("_______________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

In [39]:
```
from sklearn.model_selection import train_test_split

X = dataset.drop('target', axis=1)
y = dataset.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ranc
```

In [40]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(X_train, y_train)
```

Out[40]: LogisticRegression(solver='liblinear')

In [41]:
```
print_score(log_reg, X_train, y_train, X_test, y_test, train=True)
print_score(log_reg, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 89.54%
_______________________________________________
Classification Report:  Precision Score: 88.52%
                        Recall Score: 92.04%
                        F1 score: 90.25%
_______________________________________________
Confusion Matrix:
 [[295  45]
 [ 30 347]]

Test Result:
================================================
Accuracy Score: 81.82%
_______________________________________________
Classification Report:  Precision Score: 78.88%
                        Recall Score: 85.23%
                        F1 score: 81.94%
_______________________________________________
Confusion Matrix:
 [[125  34]
 [ 22 127]]
```

In [42]:
```
test_score = accuracy_score(y_test, log_reg.predict(X_test)) * 100
train_score = accuracy_score(y_train, log_reg.predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Regression", train_score, test_scc
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df
```

Out[42]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |

## K-nearest neighbors

In [43]:

```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)

print_score(knn_classifier, X_train, y_train, X_test, y_test, train=True)
print_score(knn_classifier, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 91.77%
_______________________________________________
Classification Report:  Precision Score: 92.06%
                        Recall Score: 92.31%
                        F1 score: 92.19%
_______________________________________________
Confusion Matrix:
 [[310  30]
 [ 29 348]]

Test Result:
================================================
Accuracy Score: 81.82%
_______________________________________________
Classification Report:  Precision Score: 78.18%
                        Recall Score: 86.58%
                        F1 score: 82.17%
_______________________________________________
Confusion Matrix:
 [[123  36]
 [ 20 129]]
```

In [44]:

```
test_score = accuracy_score(y_test, knn_classifier.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_classifier.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["K-nearest neighbors", train_score, test_s
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[44]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |
| **1** | K-nearest neighbors | 91.771269 | 81.818182 |

## Support Vector machine

In [45]:
```
from sklearn.svm import SVC


svm_model = SVC(kernel='rbf', gamma=0.1, C=1.0)
svm_model.fit(X_train, y_train)
```

Out[45]:
```
SVC(gamma=0.1)
```

In [46]:
```
print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 95.40%
_______________________________________________
Classification Report:	Precision Score: 94.10%
			Recall Score: 97.35%
			F1 score: 95.70%
_______________________________________________
Confusion Matrix:
 [[317  23]
 [ 10 367]]

Test Result:
================================================
Accuracy Score: 90.26%
_______________________________________________
Classification Report:	Precision Score: 86.50%
			Recall Score: 94.63%
			F1 score: 90.38%
_______________________________________________
Confusion Matrix:
 [[137  22]
 [  8 141]]
```

In [47]:
```
test_score = accuracy_score(y_test, svm_model.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_model.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Support Vector Machine", train_score, tes
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[47]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |
| **1** | K-nearest neighbors | 91.771269 | 81.818182 |
| **2** | Support Vector Machine | 95.397490 | 90.259740 |

# Decision Tree Classifier

In [48]:

```
from sklearn.tree import DecisionTreeClassifier


tree = DecisionTreeClassifier(random_state=42)
tree.fit(X_train, y_train)

print_score(tree, X_train, y_train, X_test, y_test, train=True)
print_score(tree, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[340   0]
 [  0 377]]

Test Result:
================================================
Accuracy Score: 97.08%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 93.96%
                        F1 score: 96.89%
_______________________________________________
Confusion Matrix:
 [[159   0]
 [  9 140]]
```

In [49]:

```
test_score = accuracy_score(y_test, tree.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Decision Tree Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[49]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |
| **1** | K-nearest neighbors | 91.771269 | 81.818182 |
| **2** | Support Vector Machine | 95.397490 | 90.259740 |
| **3** | Decision Tree Classifier | 100.000000 | 97.077922 |

## Random Forest

In [50]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

rand_forest = RandomForestClassifier(n_estimators=1000, random_state=42)
rand_forest.fit(X_train, y_train)

print_score(rand_forest, X_train, y_train, X_test, y_test, train=True)
print_score(rand_forest, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[340   0]
 [  0 377]]

Test Result:
================================================
Accuracy Score: 98.05%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 95.97%
                        F1 score: 97.95%
_______________________________________________
Confusion Matrix:
 [[159   0]
 [  6 143]]
```

In [51]:
```
test_score = accuracy_score(y_test, rand_forest.predict(X_test)) * 100
train_score = accuracy_score(y_train, rand_forest.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Forest Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[51]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |
| **1** | K-nearest neighbors | 91.771269 | 81.818182 |
| **2** | Support Vector Machine | 95.397490 | 90.259740 |
| **3** | Decision Tree Classifier | 100.000000 | 97.077922 |
| **4** | Random Forest Classifier | 100.000000 | 98.051948 |

# XGBoost Classifer

In [52]:
```
#pip install xgboost     installing xgboost
```

In [53]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier()
xgb.fit(X_train, y_train)

print_score(xgb, X_train, y_train, X_test, y_test, train=True)
print_score(xgb, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 100.00%
                        F1 score: 100.00%
_______________________________________________
Confusion Matrix:
 [[340   0]
 [  0 377]]

Test Result:
================================================
Accuracy Score: 98.05%
_______________________________________________
Classification Report:  Precision Score: 100.00%
                        Recall Score: 95.97%
                        F1 score: 97.95%
_______________________________________________
Confusion Matrix:
 [[159   0]
 [  6 143]]
```

In [57]:
```
test_score = accuracy_score(y_test, xgb.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["XGBoost Classifier", train_score, test_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[57]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |
| **1** | K-nearest neighbors | 91.771269 | 81.818182 |
| **2** | Support Vector Machine | 95.397490 | 90.259740 |
| **3** | Decision Tree Classifier | 100.000000 | 97.077922 |
| **4** | Random Forest Classifier | 100.000000 | 98.051948 |
| **5** | XGBoost Classifier | 100.000000 | 98.051948 |

# Using Hyperparameter Tuning

## Logistic Regression Hyperparameter Tuning

In [58]:
```
from sklearn.model_selection import GridSearchCV

params = {"C": np.logspace(-4, 4, 20),
          "solver": ["liblinear"]}

log_reg = LogisticRegression()

grid_search_cv = GridSearchCV(log_reg, params, scoring="accuracy", n_jobs=-1,
# grid_search_cv.fit(X_train, y_train)
```

In [59]:
```
# grid_search_cv.best_estimator_
```

In [60]:

```
log_reg = LogisticRegression(C=0.615848211066026,
                             solver='liblinear')

log_reg.fit(X_train, y_train)

print_score(log_reg, X_train, y_train, X_test, y_test, train=True)
print_score(log_reg, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 89.12%
_______________________________________________
Classification Report:  Precision Score: 88.43%
                        Recall Score: 91.25%
                        F1 score: 89.82%
_______________________________________________
Confusion Matrix:
 [[295  45]
 [ 33 344]]

Test Result:
================================================
Accuracy Score: 81.49%
_______________________________________________
Classification Report:  Precision Score: 78.75%
                        Recall Score: 84.56%
                        F1 score: 81.55%
_______________________________________________
Confusion Matrix:
 [[125  34]
 [ 23 126]]
```

In [61]:

```
test_score = accuracy_score(y_test, log_reg.predict(X_test)) * 100
train_score = accuracy_score(y_train, log_reg.predict(X_train)) * 100

tuning_results_df = pd.DataFrame(data=[["Tuned Logistic Regression", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df
```

Out[61]:

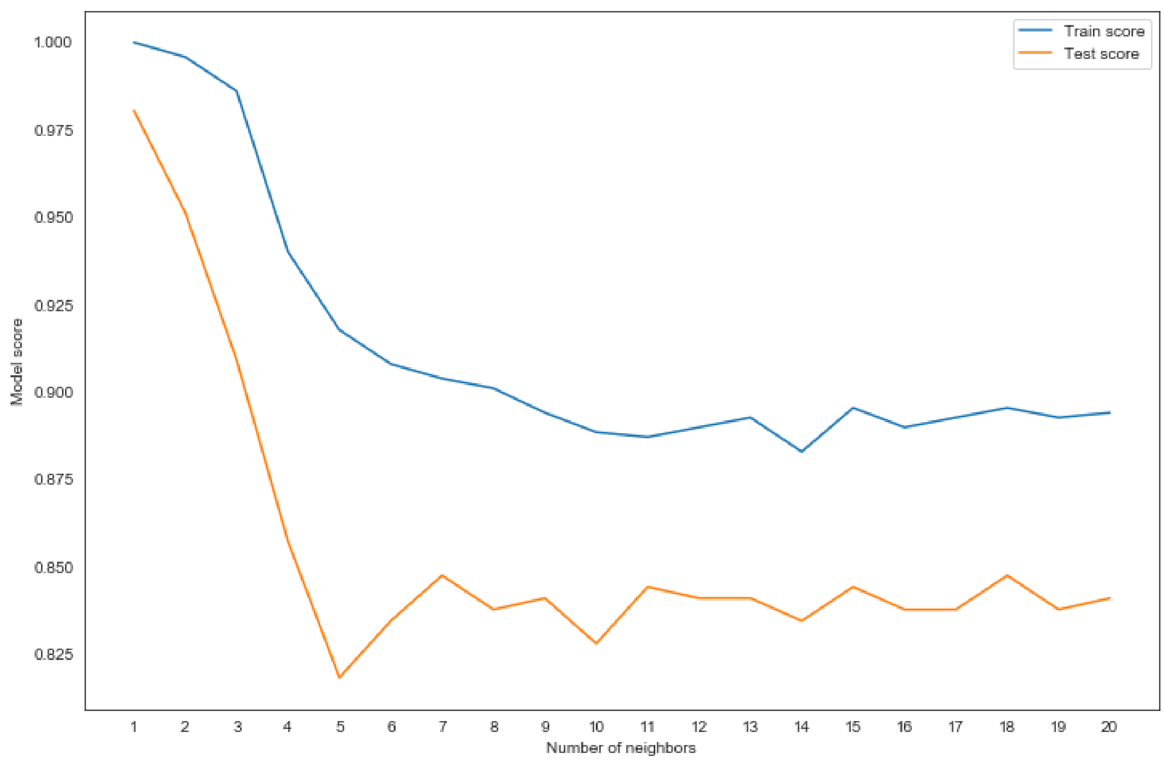
| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Tuned Logistic Regression | 89.121339 | 81.493506 |

## K-nearest neighbors Hyperparameter Tuning

In [62]:
```
train_score = []
test_score = []
neighbors = range(1, 21)

for k in neighbors:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    train_score.append(accuracy_score(y_train, model.predict(X_train)))
    test_score.append(accuracy_score(y_test, model.predict(X_test)))
```

In [63]:
```
plt.figure(figsize=(12, 8))

plt.plot(neighbors, train_score, label="Train score")
plt.plot(neighbors, test_score, label="Test score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data: {max(test_score)*100:.2f}%")
```

```
Maximum KNN score on the test data: 98.05%
```



In [64]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=19)
knn_classifier.fit(X_train, y_train)

print_score(knn_classifier, X_train, y_train, X_test, y_test, train=True)
print_score(knn_classifier, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 89.26%
_______________________________________________
Classification Report:  Precision Score: 88.46%
                        Recall Score: 91.51%
                        F1 score: 89.96%
_______________________________________________
Confusion Matrix:
 [[295  45]
 [ 32 345]]

Test Result:
================================================
Accuracy Score: 83.77%
_______________________________________________
Classification Report:  Precision Score: 79.29%
                        Recall Score: 89.93%
                        F1 score: 84.28%
_______________________________________________
Confusion Matrix:
 [[124  35]
 [ 15 134]]
```

In [65]:
```
test_score = accuracy_score(y_test, knn_classifier.predict(X_test)) * 100
train_score = accuracy_score(y_train, knn_classifier.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned K-nearest neighbors", train_score,
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[65]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Tuned Logistic Regression | 89.121339 | 81.493506 |
| **1** | Tuned K-nearest neighbors | 89.260809 | 83.766234 |

In [66]:
```
### Support Vector Machine Hyperparameter Tuning
```

In [67]:
```
svm_model = SVC(kernel='rbf', gamma=0.1, C=1.0)

params = {"C":(0.1, 0.5, 1, 2, 5, 10, 20),
          "gamma":(0.001, 0.01, 0.1, 0.25, 0.5, 0.75, 1),
          "kernel":('linear', 'poly', 'rbf')}

svm_grid = GridSearchCV(svm_model, params, n_jobs=-1, cv=5, verbose=1, scorir
# svm_grid.fit(X_train, y_train)
```

In [68]:
```
# svm_grid.best_estimator_
```

In [69]:
```
svm_model = SVC(C=5, gamma=0.01, kernel='rbf')
svm_model.fit(X_train, y_train)

print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 90.66%
_______________________________________________
Classification Report:  Precision Score: 90.36%
                        Recall Score: 92.04%
                        F1 score: 91.20%
_______________________________________________
Confusion Matrix:
 [[303  37]
 [ 30 347]]

Test Result:
================================================
Accuracy Score: 82.47%
_______________________________________________
Classification Report:  Precision Score: 79.87%
                        Recall Score: 85.23%
                        F1 score: 82.47%
_______________________________________________
Confusion Matrix:
 [[127  32]
 [ 22 127]]
```

In [70]:

```
test_score = accuracy_score(y_test, svm_model.predict(X_test)) * 100
train_score = accuracy_score(y_train, svm_model.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Support Vector Machine", train_scor
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[70]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Tuned Logistic Regression | 89.121339 | 81.493506 |
| **1** | Tuned K-nearest neighbors | 89.260809 | 83.766234 |
| **2** | Tuned Support Vector Machine | 90.655509 | 82.467532 |

## Decision Tree Classifier Hyperparameter Tuning

In [71]:

```
params = {"criterion":("gini", "entropy"),
          "splitter":("best", "random"),
          "max_depth":(list(range(1, 20))),
          "min_samples_split":[2, 3, 4],
          "min_samples_leaf":list(range(1, 20))
          }

tree = DecisionTreeClassifier(random_state=42)
grid_search_cv = GridSearchCV(tree, params, scoring="accuracy", n_jobs=-1, ve
# grid_search_cv.fit(X_train, y_train)
```

In [72]:

```
# grid_search_cv.best_estimator_
```

In [73]:
```
tree = DecisionTreeClassifier(criterion='gini',
                              max_depth=3,
                              min_samples_leaf=2,
                              min_samples_split=2,
                              splitter='random')
tree.fit(X_train, y_train)

print_score(tree, X_train, y_train, X_test, y_test, train=True)
print_score(tree, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 86.05%
_______________________________________________
Classification Report:  Precision Score: 87.95%
                        Recall Score: 85.15%
                        F1 score: 86.52%
_______________________________________________
Confusion Matrix:
 [[296  44]
 [ 56 321]]

Test Result:
================================================
Accuracy Score: 82.14%
_______________________________________________
Classification Report:  Precision Score: 80.52%
                        Recall Score: 83.22%
                        F1 score: 81.85%
_______________________________________________
Confusion Matrix:
 [[129  30]
 [ 25 124]]
```

In [74]:
```
test_score = accuracy_score(y_test, tree.predict(X_test)) * 100
train_score = accuracy_score(y_train, tree.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Decision Tree Classifier", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[74]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| 0 | Tuned Logistic Regression | 89.121339 | 81.493506 |
| 1 | Tuned K-nearest neighbors | 89.260809 | 83.766234 |
| 2 | Tuned Support Vector Machine | 90.655509 | 82.467532 |
| 3 | Tuned Decision Tree Classifier | 86.052999 | 82.142857 |

## Random Forest Classifier Hyperparameter Tuning

In [76]:
```
from sklearn.model_selection import RandomizedSearchCV

n_estimators = [int(x) for x in np.linspace(start=200, stop=2000, num=10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators, 'max_features': max_features,
               'max_depth': max_depth, 'min_samples_split': min_samples_split
               'min_samples_leaf': min_samples_leaf, 'bootstrap': bootstrap}


rand_forest = RandomForestClassifier(random_state=42)

rf_random = RandomizedSearchCV(estimator=rand_forest, param_distributions=rar
                               verbose=2, random_state=42, n_jobs=-1)
# rf_random.fit(X_train, y_train)
```

In [77]:
```
# rf_random.best_estimator_
```

In [78]:
```
rand_forest = RandomForestClassifier(bootstrap=True,
                                     max_depth=70,
                                     max_features='auto',
                                     min_samples_leaf=4,
                                     min_samples_split=10,
                                     n_estimators=400)
rand_forest.fit(X_train, y_train)
```

Out[78]:
```
RandomForestClassifier(max_depth=70, min_samples_leaf=4, min_samples_split=
10,
                       n_estimators=400)
```

In [79]:
```
print_score(rand_forest, X_train, y_train, X_test, y_test, train=True)
print_score(rand_forest, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 96.51%
_______________________________________________
Classification Report:  Precision Score: 95.60%
                        Recall Score: 97.88%
                        F1 score: 96.72%
_______________________________________________
Confusion Matrix:
 [[323  17]
 [  8 369]]

Test Result:
================================================
Accuracy Score: 91.23%
_______________________________________________
Classification Report:  Precision Score: 89.10%
                        Recall Score: 93.29%
                        F1 score: 91.15%
_______________________________________________
Confusion Matrix:
 [[142  17]
 [ 10 139]]
```

In [80]:
```
test_score = accuracy_score(y_test, rand_forest.predict(X_test)) * 100
train_score = accuracy_score(y_train, rand_forest.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Random Forest Classifier", train_sc
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[80]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Tuned Logistic Regression | 89.121339 | 81.493506 |
| **1** | Tuned K-nearest neighbors | 89.260809 | 83.766234 |
| **2** | Tuned Support Vector Machine | 90.655509 | 82.467532 |
| **3** | Tuned Decision Tree Classifier | 86.052999 | 82.142857 |
| **4** | Tuned Random Forest Classifier | 96.513250 | 91.233766 |

## XGBoost Classifier Hyperparameter Tuning

In [81]:
```
n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
booster = ['gbtree', 'gblinear']
base_score = [0.25, 0.5, 0.75, 0.99]
learning_rate = [0.05, 0.1, 0.15, 0.20]
min_child_weight = [1, 2, 3, 4]

hyperparameter_grid = {'n_estimators': n_estimators, 'max_depth': max_depth,
                       'learning_rate' : learning_rate, 'min_child_weight' :
                       'booster' : booster, 'base_score' : base_score
                        }

xgb_model = XGBClassifier()

xgb_cv = RandomizedSearchCV(estimator=xgb_model, param_distributions=hyperpar
                               cv=5, n_iter=650, scoring = 'accuracy',n_jobs
                               verbose=1, return_train_score = True, random_s


# xgb_cv.fit(X_train, y_train)
```

In [82]:
```
# xgb_cv.best_estimator_
```

In [83]:
```
xgb_best = XGBClassifier(base_score=0.25,
                         booster='gbtree',
                         learning_rate=0.05,
                         max_depth=5,
                         min_child_weight=2,
                         n_estimators=100)
xgb_best.fit(X_train, y_train)
```

Out[83]:
```
XGBClassifier(base_score=0.25, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.05, max_delta_step=0, max_depth=5,
              min_child_weight=2, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state
=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [84]:
```
print_score(xgb_best, X_train, y_train, X_test, y_test, train=True)
print_score(xgb_best, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 99.58%
_______________________________________________
Classification Report:  Precision Score: 99.21%
                        Recall Score: 100.00%
                        F1 score: 99.60%
_______________________________________________
Confusion Matrix:
 [[337   3]
 [  0 377]]

Test Result:
================================================
Accuracy Score: 97.73%
_______________________________________________
Classification Report:  Precision Score: 97.33%
                        Recall Score: 97.99%
                        F1 score: 97.66%
_______________________________________________
Confusion Matrix:
 [[155   4]
 [  3 146]]
```

In [85]:
```
test_score = accuracy_score(y_test, xgb_best.predict(X_test)) * 100
train_score = accuracy_score(y_train, xgb_best.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned XGBoost Classifier", train_score, t
                          columns=['Model', 'Training Accuracy %', 'Testing A
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[85]:

| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Tuned Logistic Regression | 89.121339 | 81.493506 |
| **1** | Tuned K-nearest neighbors | 89.260809 | 83.766234 |
| **2** | Tuned Support Vector Machine | 90.655509 | 82.467532 |
| **3** | Tuned Decision Tree Classifier | 86.052999 | 82.142857 |
| **4** | Tuned Random Forest Classifier | 96.513250 | 91.233766 |
| **5** | Tuned XGBoost Classifier | 99.581590 | 97.727273 |

In [86]:
```
results_df
```

Out[86]:

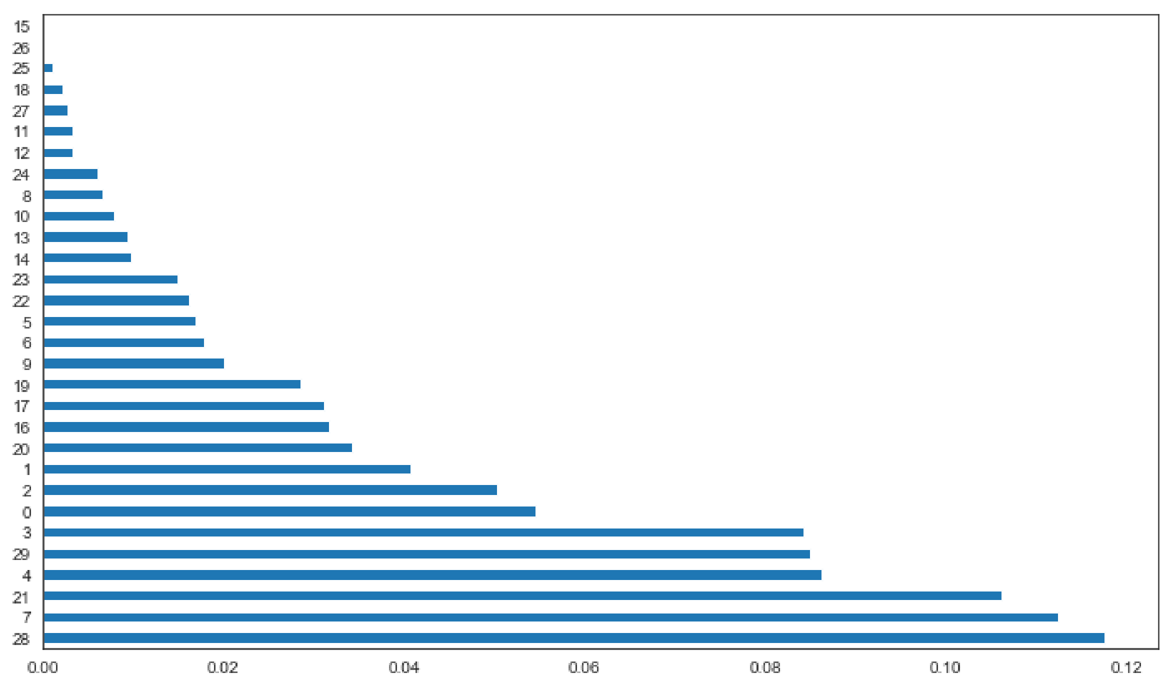
| | Model | Training Accuracy % | Testing Accuracy % |
|---|---|---|---|
| **0** | Logistic Regression | 89.539749 | 81.818182 |
| **1** | K-nearest neighbors | 91.771269 | 81.818182 |
| **2** | Support Vector Machine | 95.397490 | 90.259740 |
| **3** | Decision Tree Classifier | 100.000000 | 97.077922 |
| **4** | Random Forest Classifier | 100.000000 | 98.051948 |
| **5** | XGBoost Classifier | 100.000000 | 98.051948 |

# Features Importance According to Random Forest and XGBoost

In [87]:
```python
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)
```

In [88]:
```python
feature_imp(X, rand_forest).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x20ed8560648>



In [89]:
```
feature_imp(X, xgb_best).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[89]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20ed62c9608>
```