In [1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

In [2]:
```python
df = pd.read_csv("../documents/input/Heart Disease UCI.csv")
df.head()
```

Out[2]:

	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 [3]:
```python
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)
memory usage: 33.3 KB
```

In [4]:
```python
df.shape
```

Out[4]: (303, 14)

In [5]:
```python
pd.set_option("display.float", "{:.2f}".format)
df.describe()
```

Out[5]:

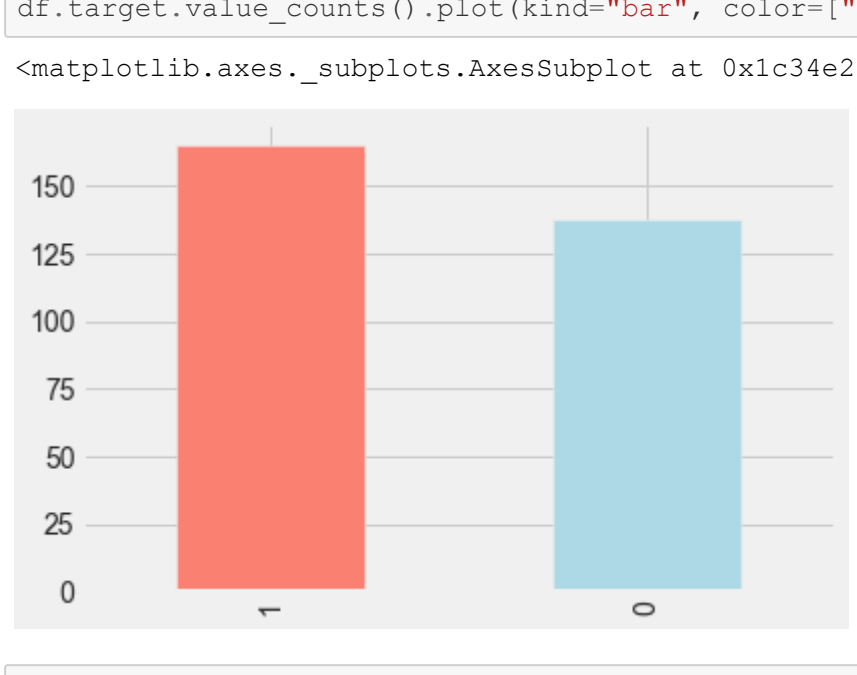
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00	303.00
mean	54.37	0.68	0.97	131.62	246.26	0.15	0.53	149.65	0.33	1.04	1.40	0.73	2.31	0.54
std	9.08	0.47	1.03	17.54	51.83	0.36	0.53	22.91	0.47	1.16	0.62	1.02	0.61	0.50
min	29.00	0.00	0.00	94.00	126.00	0.00	0.00	71.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	47.50	0.00	0.00	120.00	211.00	0.00	0.00	133.50	0.00	0.00	1.00	0.00	2.00	0.00
50%	55.00	1.00	1.00	130.00	240.00	0.00	1.00	153.00	0.00	0.80	1.00	0.00	2.00	1.00
75%	61.00	1.00	2.00	140.00	274.50	0.00	1.00	166.00	1.00	1.60	2.00	1.00	3.00	1.00
max	77.00	1.00	3.00	200.00	564.00	1.00	2.00	202.00	1.00	6.20	2.00	4.00	3.00	1.00

In [6]:
```python
df.target.value_counts()
```

Out[6]:
```
1    165
0    138
Name: target, dtype: int64
```

In [7]:
```python
df.target.value_counts().plot(kind="bar", color=["salmon", "lightblue"])
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1c34e212a48>



In [8]:
```python
# Checking for messing values
df.isna().sum()
```

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
```

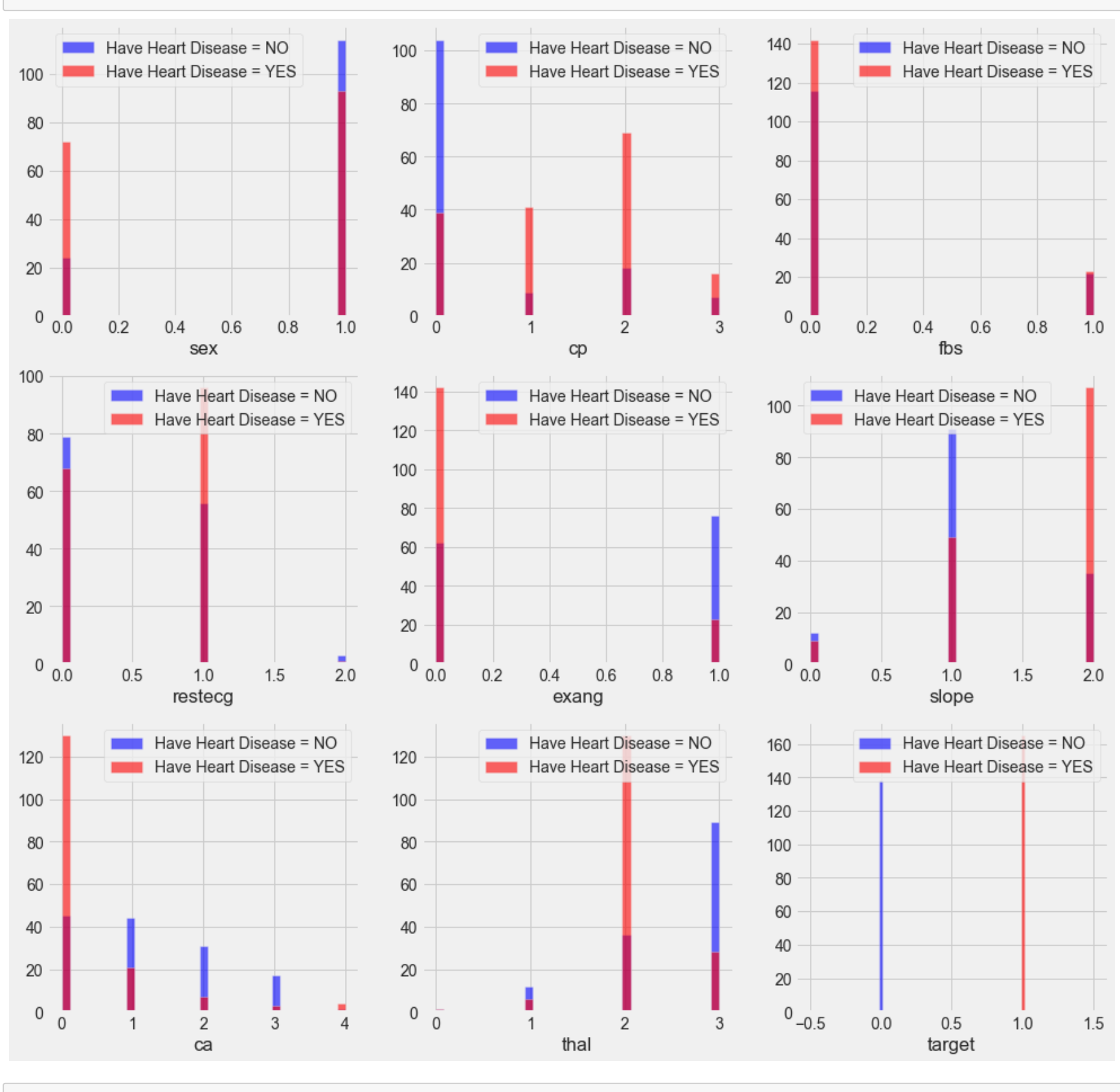
In [9]:
```python
categorical_val = []
continous_val = []
for column in df.columns:
    print('==============================')
    print(f"{column} : {df[column].unique()}")
    if len(df[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

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

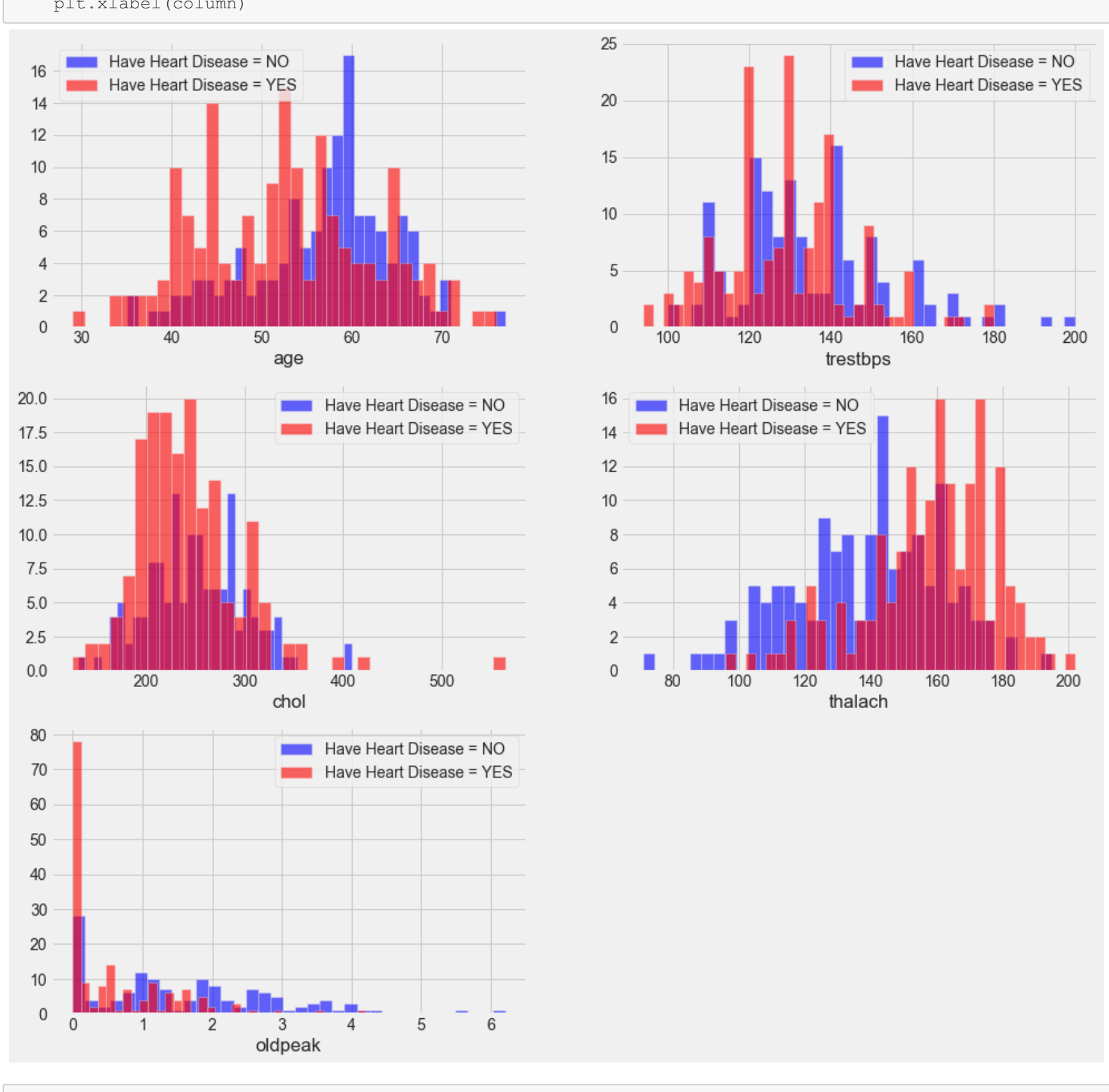
In [10]:
```python
categorical_val
```

Out[10]: ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'target']

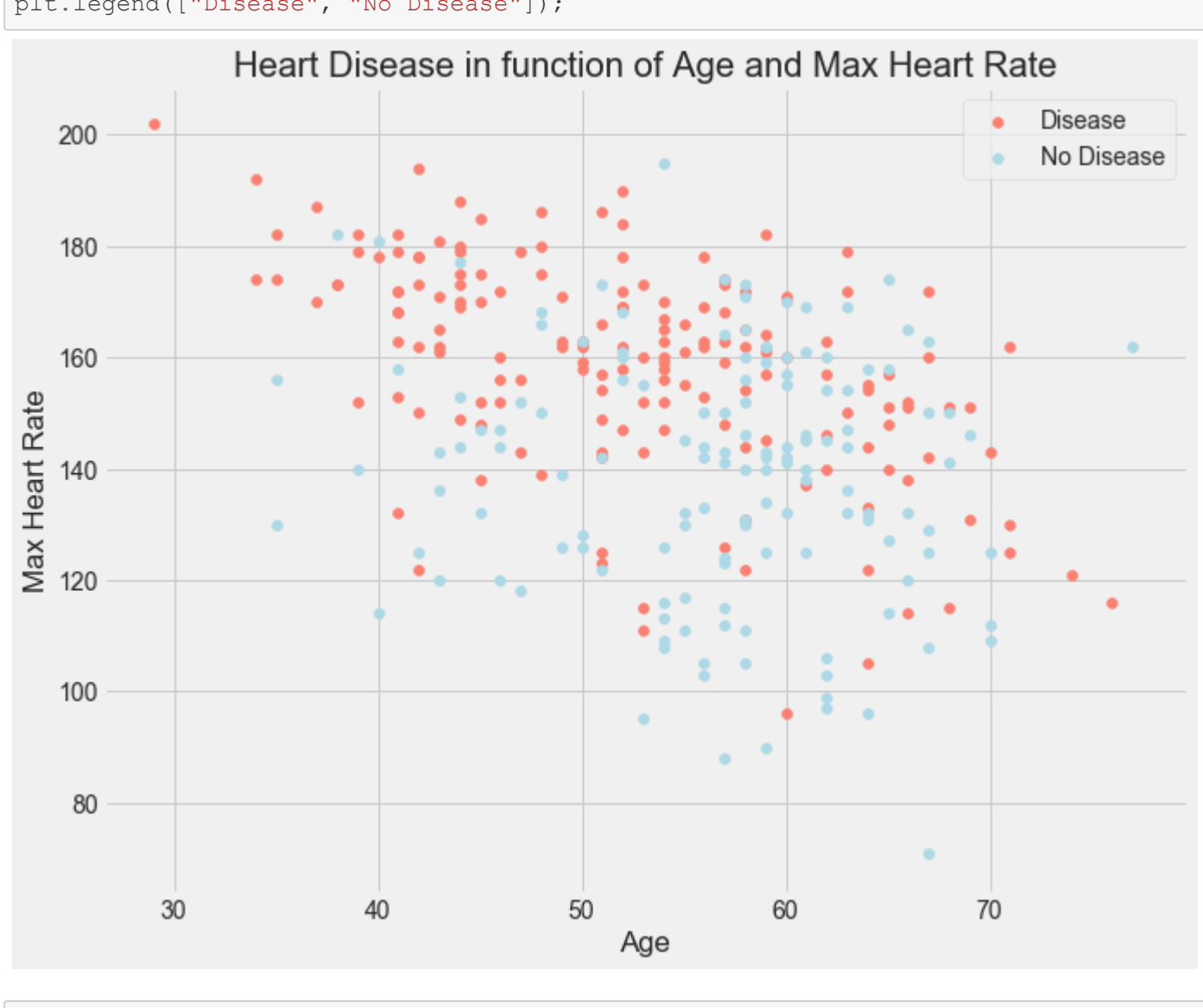
In [11]:
```python
plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1):
    plt.subplot(3, 3, i)
    df[df["target"] == 0][column].hist(bins=35, color='blue', label='Have Heart Disease = NO', alpha=0.
6)
    df[df["target"] == 1][column].hist(bins=35, color='red', label='Have Heart Disease = YES', alpha=0.
6)
    plt.legend()
    plt.xlabel(column)
```



In [12]:
```python
plt.figure(figsize=(15, 15))

for i, column in enumerate(continous_val, 1):
    plt.subplot(3, 2, i)
    df[df["target"] == 0][column].hist(bins=35, color='blue', label='Have Heart Disease = NO', alpha=0.
6)
    df[df["target"] == 1][column].hist(bins=35, color='red', label='Have Heart Disease = YES', alpha=0.
6)
    plt.legend()
    plt.xlabel(column)
```

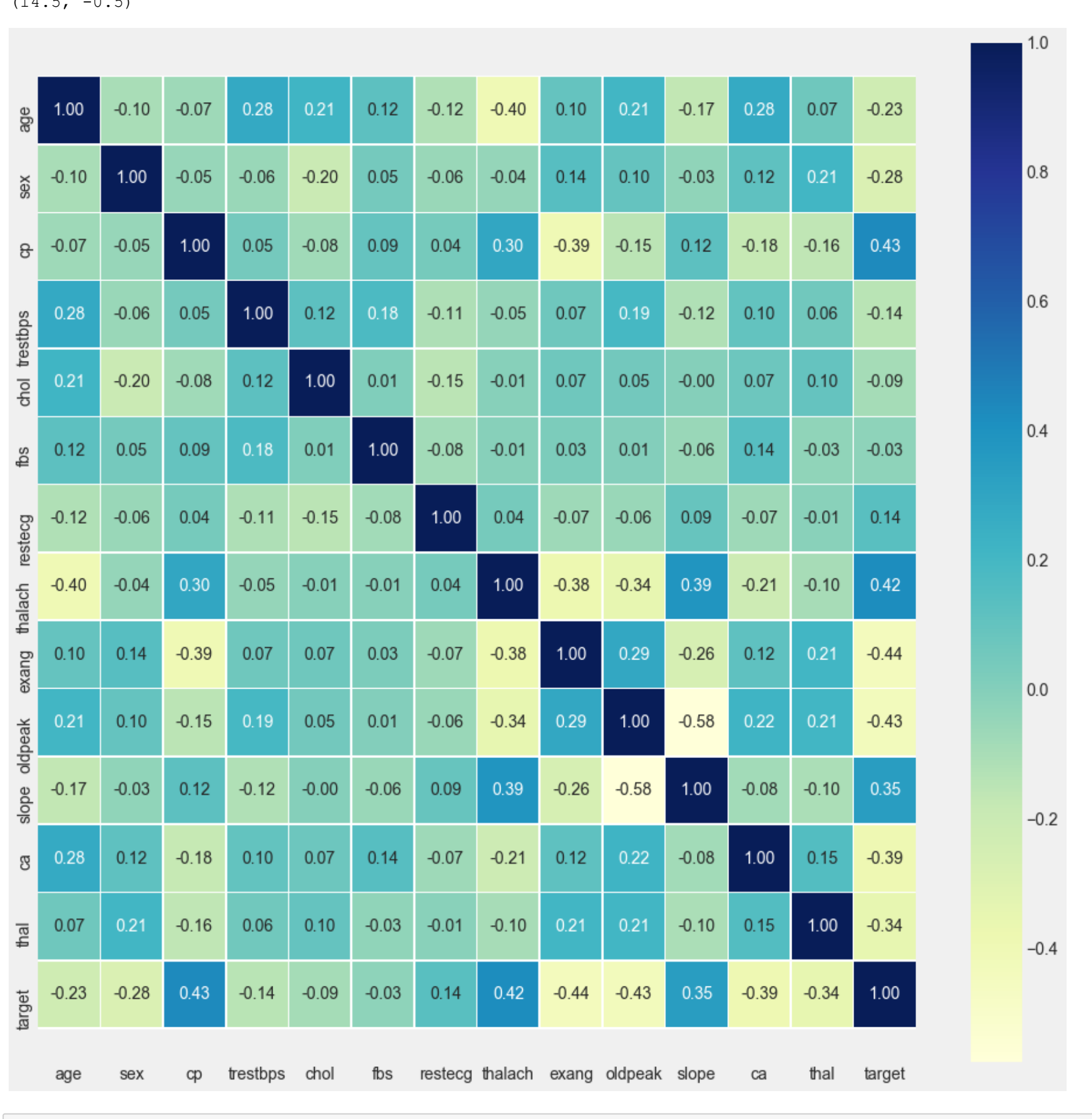


In [13]:
```python
# Create another figure
plt.figure(figsize=(10, 8))

# Scatter with postivie examples
plt.scatter(df.age[df.target==1],
            df.thalach[df.target==1],
            c="salmon")

# Scatter with negative examples
plt.scatter(df.age[df.target==0],
            df.thalach[df.target==0],
            c="lightblue")

# Add some helpful info
plt.title("Heart Disease in function of Age and Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease", "No Disease"]);
```



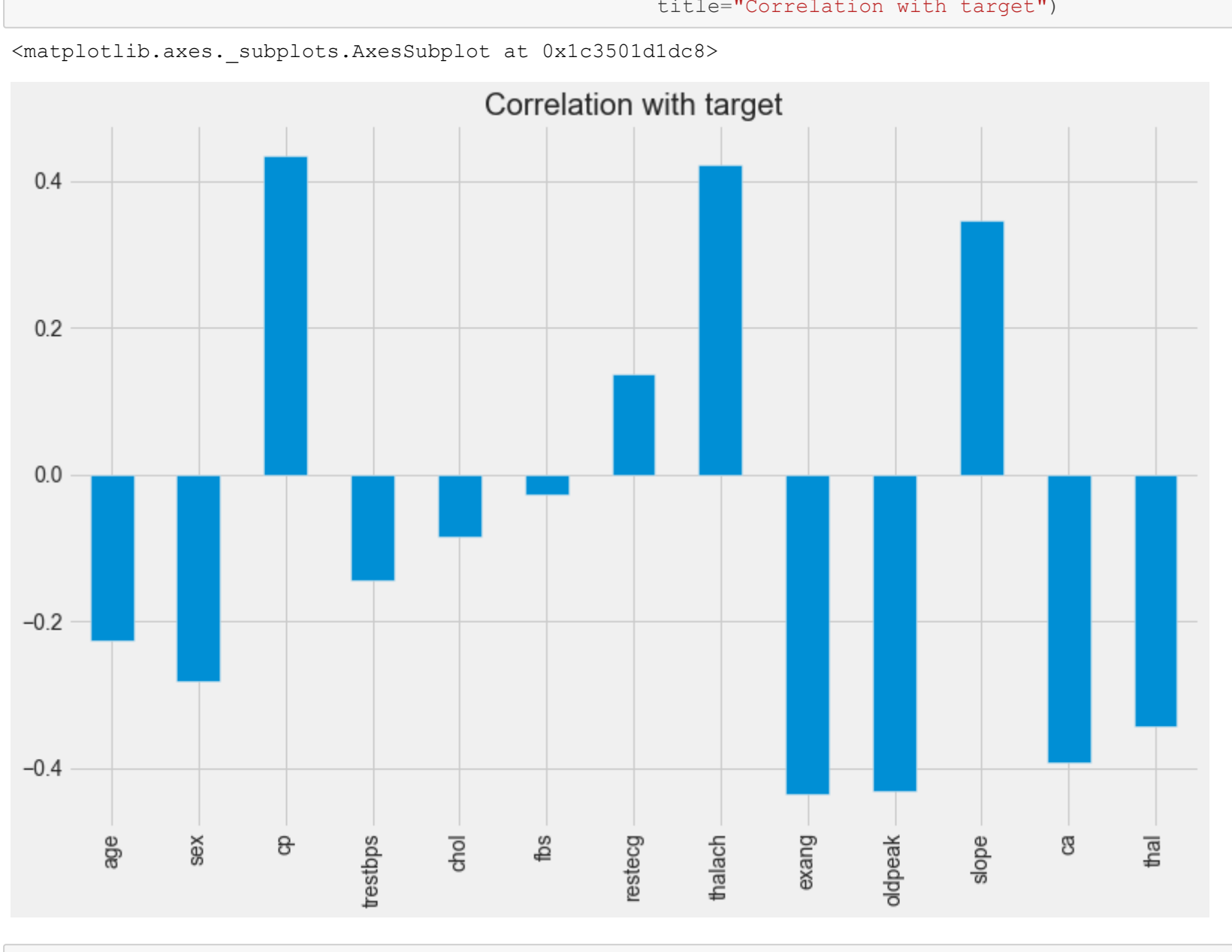
In [14]:
```python
# Let's make our correlation matrix a little prettier
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(15, 15))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[14]: (14.5, -0.5)



In [15]:
```python
df.drop('target', axis=1).corrwith(df.target).plot(kind='bar', grid=True, figsize=(12, 8),
                                                   title="Correlation with target")
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1c3501d1dc8>



In [16]:
```python
categorical_val.remove('target')
dataset = pd.get_dummies(df, columns = categorical_val)
```

In [17]:
```python
dataset.head()
```

Out[17]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1	t
0	63	145	233	150	2.30	1	0	1	0	0	...	0	1	0	0	0	0	0	1	
1	37	130	250	187	3.50	1	0	1	0	0	...	0	1	0	0	0	0	0	0	
2	41	130	204	172	1.40	1	1	0	0	1	...	1	1	0	0	0	0	0	0	
3	56	120	236	178	0.80	1	0	1	0	1	...	1	1	0	0	0	0	0	0	
4	57	120	354	163	0.60	1	1	0	1	0	...	1	1	0	0	0	0	0	0	

5 rows × 31 columns

In [18]:
```python
print(df.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 [19]:
```python
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 [20]:
```python
dataset.head()
```

Out[20]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0	ca_1	ca_2	ca_3	ca_4	thal_0	thal_1
0	0.95	0.76	-0.26	0.02	1.09	1	0	1	0	0	...	0	1	0	0	0	0	0	1
1	-1.92	-0.09	0.07	1.63	2.12	1	0	1	0	0	...	0	1	0	0	0	0	0	0
2	-1.47	-0.09	-0.82	0.98	0.31	1	1	0	0	1	...	1	1	0	0	0	0	0	0
3	0.18	-0.66	-0.20	1.24	-0.21	1	0	1	0	1	...	1	1	0	0	0	0	0	0
4	0.29	-0.66	2.08	0.58	-0.38	1	1	0	1	0	...	1	1	0	0	0	0	0	0

5 rows × 31 columns

In [21]:
```python
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_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 [22]:
```python
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, random_state=42)
```

In [23]:
```python
from sklearn.linear_model import LogisticRegression

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

Out[23]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [24]:
```python
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: 86.79%
_______________________________________________
Classification Report:  Precision Score: 85.95%
                        Recall Score: 90.43%
                        F1 score: 88.14%
_______________________________________________
Confusion Matrix:
 [[ 80  17]
 [ 11 104]]

Test Result:
================================================
Accuracy Score: 86.81%
_______________________________________________
Classification Report:  Precision Score: 86.54%
                        Recall Score: 90.00%
                        F1 score: 88.24%
_______________________________________________
Confusion Matrix:
 [[34  7]
 [ 5 45]]
```

In [25]:
```python
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_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df
```

Out[25]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81

In [26]:
```python
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: 86.79%
_______________________________________________
Classification Report:  Precision Score: 87.18%
                        Recall Score: 88.70%
                        F1 score: 87.93%
_______________________________________________
Confusion Matrix:
 [[ 82  15]
 [ 13 102]]

Test Result:
================================================
Accuracy Score: 86.81%
_______________________________________________
Classification Report:  Precision Score: 88.00%
                        Recall Score: 88.00%
                        F1 score: 88.00%
_______________________________________________
Confusion Matrix:
 [[35  6]
 [ 6 44]]
```

In [27]:
```python
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_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[27]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81
1	K-nearest neighbors	86.79	86.81

In [28]:
```python
from sklearn.svm import SVC

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

Out[28]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [29]:
```python
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: 93.40%
_______________________________________________
Classification Report:  Precision Score: 93.16%
                        Recall Score: 94.78%
                        F1 score: 93.97%
_______________________________________________
Confusion Matrix:
 [[ 89   8]
 [  6 109]]

Test Result:
================================================
Accuracy Score: 87.91%
_______________________________________________
Classification Report:  Precision Score: 89.80%
                        Recall Score: 88.00%
                        F1 score: 88.89%
_______________________________________________
Confusion Matrix:
 [[36  5]
 [ 6 44]]
```

In [30]:
```python
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, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[30]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81
1	K-nearest neighbors	86.79	86.81
2	Support Vector Machine	93.40	87.91

In [31]:
```python
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:
 [[ 97   0]
 [  0 115]]

Test Result:
================================================
Accuracy Score: 78.02%
_______________________________________________
Classification Report:  Precision Score: 84.09%
                        Recall Score: 74.00%
                        F1 score: 78.72%
_______________________________________________
Confusion Matrix:
 [[34  7]
 [13 37]]
```

In [32]:
```python
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, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[32]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81
1	K-nearest neighbors	86.79	86.81
2	Support Vector Machine	93.40	87.91
3	Decision Tree Classifier	100.00	78.02

In [33]:
```python
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:
 [[ 97   0]
 [  0 115]]

Test Result:
================================================
Accuracy Score: 82.42%
_______________________________________________
Classification Report:  Precision Score: 84.00%
                        Recall Score: 84.00%
                        F1 score: 84.00%
_______________________________________________
Confusion Matrix:
 [[33  8]
 [ 8 42]]
```

In [34]:
```python
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, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[34]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81
1	K-nearest neighbors	86.79	86.81
2	Support Vector Machine	93.40	87.91
3	Decision Tree Classifier	100.00	78.02
4	Random Forest Classifier	100.00	82.42

In [35]:
```python
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:
 [[ 97   0]
 [  0 115]]

Test Result:
================================================
Accuracy Score: 82.42%
_______________________________________________
Classification Report:  Precision Score: 84.00%
                        Recall Score: 84.00%
                        F1 score: 84.00%
_______________________________________________
Confusion Matrix:
 [[33  8]
 [ 8 42]]
```

In [36]:
```python
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_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2, ignore_index=True)
results_df
```

Out[36]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81
1	K-nearest neighbors	86.79	86.81
2	Support Vector Machine	93.40	87.91
3	Decision Tree Classifier	100.00	78.02
4	Random Forest Classifier	100.00	82.42
5	XGBoost Classifier	100.00	82.42

In [37]:
```python
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, verbose=1, cv=5, iid=True
)
# grid_search_cv.fit(X_train, y_train)
```

In [38]:
```python
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: 87.26%
_______________________________________________
Classification Report:  Precision Score: 86.07%
                        Recall Score: 91.30%
                        F1 score: 88.61%
_______________________________________________
Confusion Matrix:
 [[ 80  17]
 [ 10 105]]

Test Result:
================================================
Accuracy Score: 86.81%
_______________________________________________
Classification Report:  Precision Score: 86.54%
                        Recall Score: 90.00%
                        F1 score: 88.24%
_______________________________________________
Confusion Matrix:
 [[34  7]
 [ 5 45]]
```

In [39]:
```python
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_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df
```

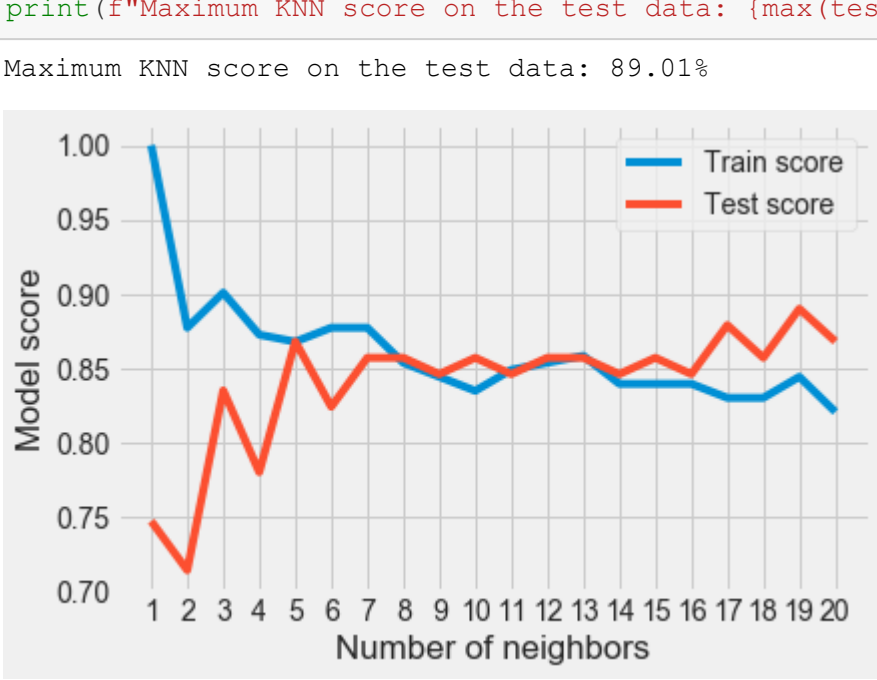
Out[39]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	87.26	86.81

In [40]:
```python
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 [41]:
```python
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: 89.01%



In [42]:
```python
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: 84.43%
_______________________________________________
Classification Report:  Precision Score: 82.54%
                        Recall Score: 90.43%
                        F1 score: 86.31%
_______________________________________________
Confusion Matrix:
 [[ 75  22]
 [ 11 104]]

Test Result:
================================================
Accuracy Score: 89.01%
_______________________________________________
Classification Report:  Precision Score: 87.04%
                        Recall Score: 94.00%
                        F1 score: 90.38%
_______________________________________________
Confusion Matrix:
 [[34  7]
 [ 3 47]]
```

In [43]:
```python
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, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[43]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	87.26	86.81
1	Tuned K-nearest neighbors	84.43	89.01

In [44]:
```python
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, scoring="accuracy")
# svm_grid.fit(X_train, y_train)
```

In [45]:
```python
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: 87.74%
_______________________________________________
Classification Report:  Precision Score: 87.39%
                        Recall Score: 90.43%
                        F1 score: 88.89%
_______________________________________________
Confusion Matrix:
 [[ 82  15]
 [ 11 104]]

Test Result:
================================================
Accuracy Score: 84.62%
_______________________________________________
Classification Report:  Precision Score: 84.62%
                        Recall Score: 88.00%
                        F1 score: 86.27%
_______________________________________________
Confusion Matrix:
 [[33  8]
 [ 6 44]]
```

In [46]:
```python
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_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[46]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	87.26	86.81
1	Tuned K-nearest neighbors	84.43	89.01
2	Tuned Support Vector Machine	87.74	84.62

In [47]:
```python
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, verbose=1, cv=3, iid=True)
# grid_search_cv.fit(X_train, y_train)
```

In [48]:
```python
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: 85.85%
_______________________________________________
Classification Report:  Precision Score: 85.12%
                        Recall Score: 89.57%
                        F1 score: 87.29%
_______________________________________________
Confusion Matrix:
 [[ 79  18]
 [ 12 103]]

Test Result:
================================================
Accuracy Score: 84.62%
_______________________________________________
Classification Report:  Precision Score: 86.00%
                        Recall Score: 86.00%
                        F1 score: 86.00%
_______________________________________________
Confusion Matrix:
 [[34  7]
 [ 7 43]]
```

In [49]:
```python
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_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[49]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	87.26	86.81
1	Tuned K-nearest neighbors	84.43	89.01
2	Tuned Support Vector Machine	87.74	84.62
3	Tuned Decision Tree Classifier	85.85	84.62

In [50]:
```python
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=random_grid, n_iter=100, cv=3
,
                               verbose=2, random_state=42, n_jobs=-1)


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

In [51]:
```python
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[51]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=70, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=4, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=400,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [52]:
```python
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: 91.51%
_______________________________________________
Classification Report:  Precision Score: 91.45%
                        Recall Score: 93.04%
                        F1 score: 92.24%
_______________________________________________
Confusion Matrix:
 [[ 87  10]
 [  8 107]]

Test Result:
================================================
Accuracy Score: 82.42%
_______________________________________________
Classification Report:  Precision Score: 84.00%
                        Recall Score: 84.00%
                        F1 score: 84.00%
_______________________________________________
Confusion Matrix:
 [[33  8]
 [ 8 42]]
```

In [53]:
```python
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_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[53]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	87.26	86.81
1	Tuned K-nearest neighbors	84.43	89.01
2	Tuned Support Vector Machine	87.74	84.62
3	Tuned Decision Tree Classifier	85.85	84.62
4	Tuned Random Forest Classifier	91.51	82.42

In [54]:
```python
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' : min_child_weight,
                       'booster' : booster, 'base_score' : base_score
                      }

xgb_model = XGBClassifier()

xgb_cv = RandomizedSearchCV(estimator=xgb_model, param_distributions=hyperparameter_grid,
                               cv=5, n_iter=650, scoring = 'accuracy',n_jobs =-1, iid=True,
                               verbose=1, return_train_score = True, random_state=42)


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

In [55]:
```python
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[55]:
```
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,
              objective='binary:logistic', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [56]:
```python
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: 96.23%
_______________________________________________
Classification Report:  Precision Score: 95.73%
                        Recall Score: 97.39%
                        F1 score: 96.55%
_______________________________________________
Confusion Matrix:
 [[ 92   5]
 [  3 112]]

Test Result:
================================================
Accuracy Score: 83.52%
_______________________________________________
Classification Report:  Precision Score: 84.31%
                        Recall Score: 86.00%
                        F1 score: 85.15%
_______________________________________________
Confusion Matrix:
 [[33  8]
 [ 7 43]]
```

In [57]:
```python
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, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

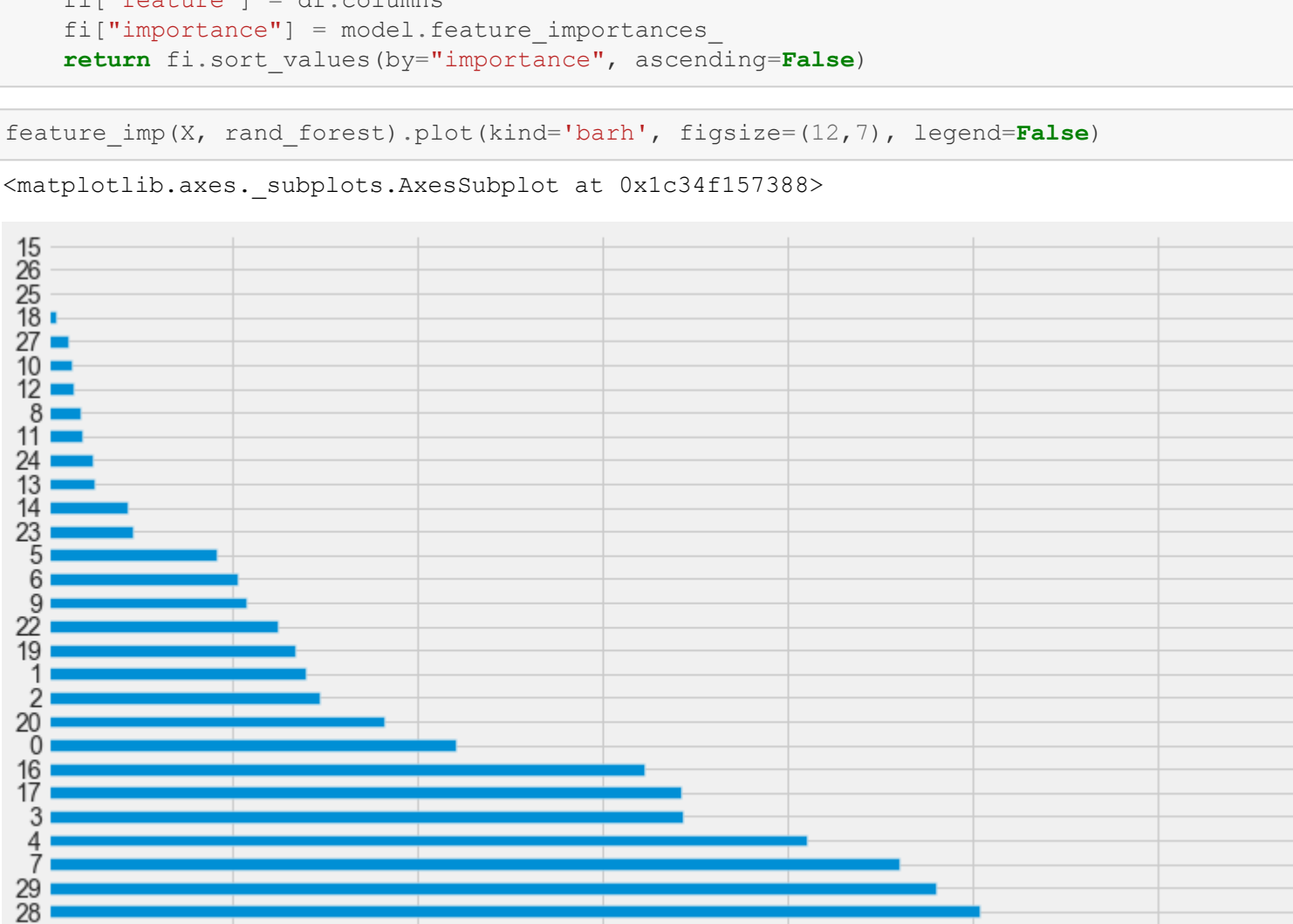
Out[57]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	87.26	86.81
1	Tuned K-nearest neighbors	84.43	89.01
2	Tuned Support Vector Machine	87.74	84.62
3	Tuned Decision Tree Classifier	85.85	84.62
4	Tuned Random Forest Classifier	91.51	82.42
5	Tuned XGBoost Classifier	96.23	83.52

In [58]:
```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)
```

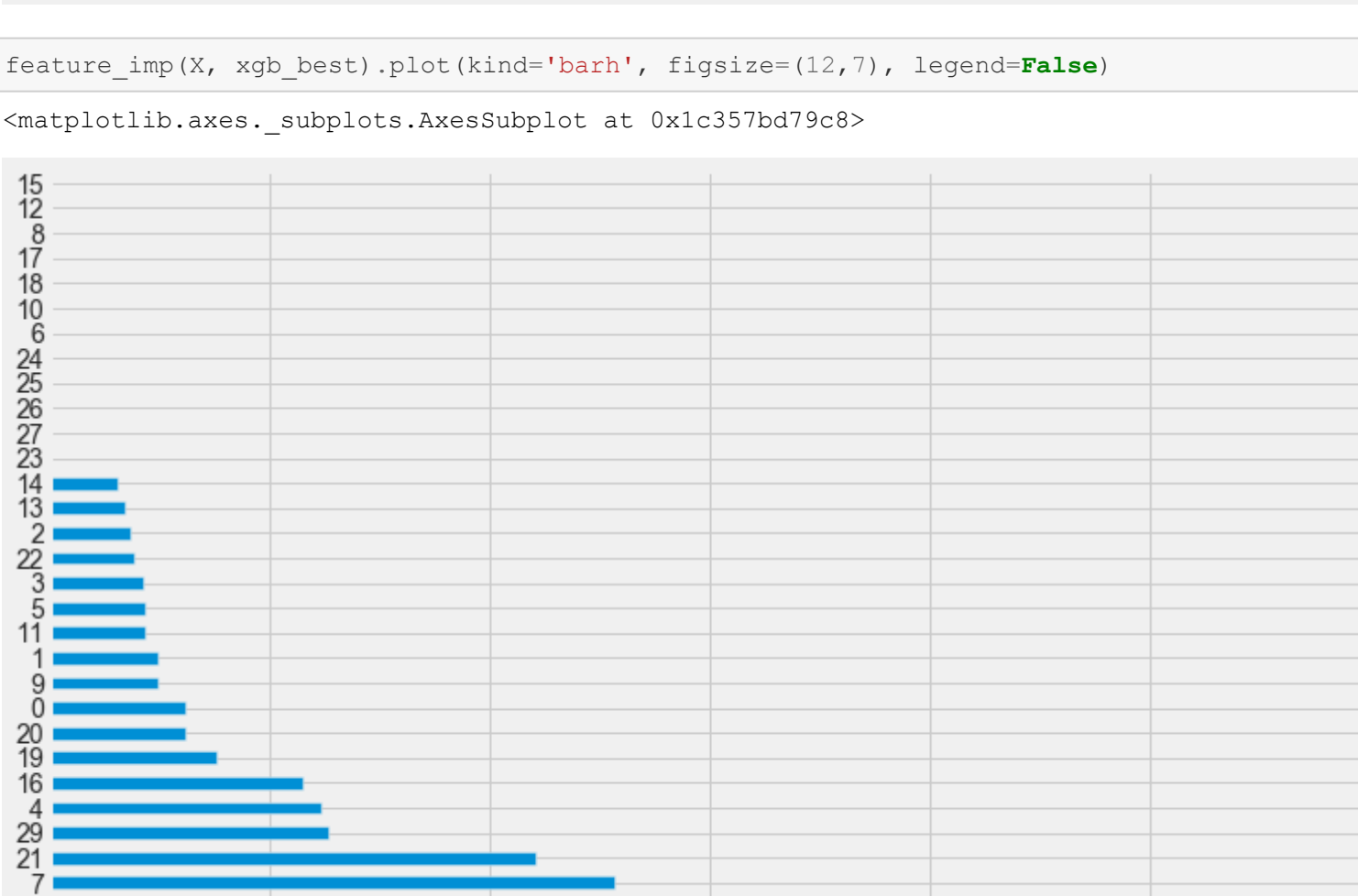
In [59]:
```python
feature_imp(X, rand_forest).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[59]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c34f157388>`



In [60]:
```python
feature_imp(X, xgb_best).plot(kind='barh', figsize=(12,7), legend=False)
```

Out[60]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c357bd79c8>`



In [ ]: