HealthCare

By Rushikesh R. Shinde Code Cr. Simplilearn

Importing Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#import cufflinks as cf
%matplotlib inline
```

In [2]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

In [3]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import  RandomizedSearchCV, train_test_split
```

In [4]:
```
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Loading the data

In [5]:
```
data = pd.read_csv("heart.csv")
data.head(6) # Mention no of rows to be displayed from the top in the argument
```

Out[5]:

	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
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1

In [6]:
```
data.shape
#shape of the data
```

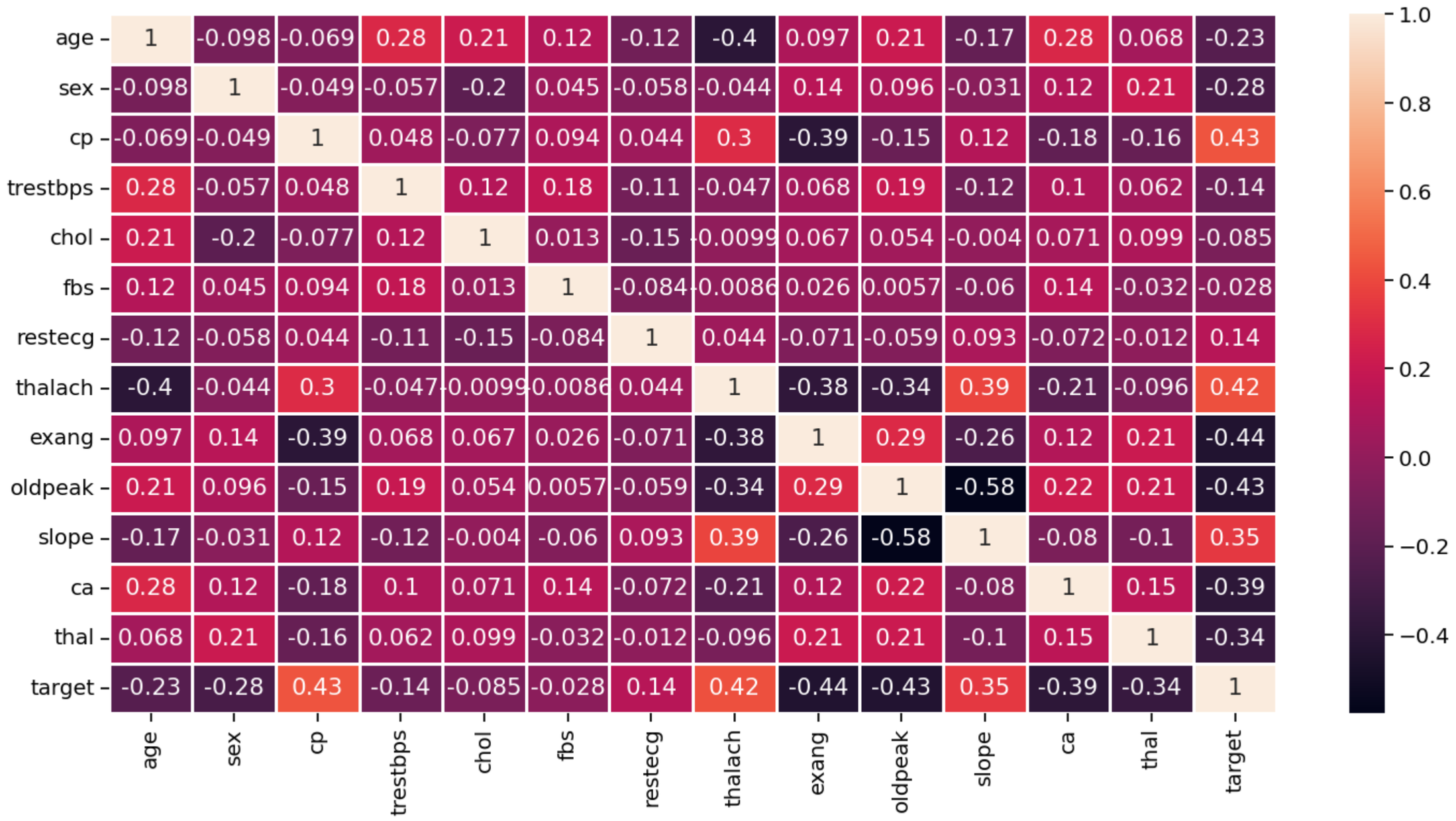
Out[6]:
```
(303, 14)
```

In [7]:
```
data.describe()
```

Out[7]:

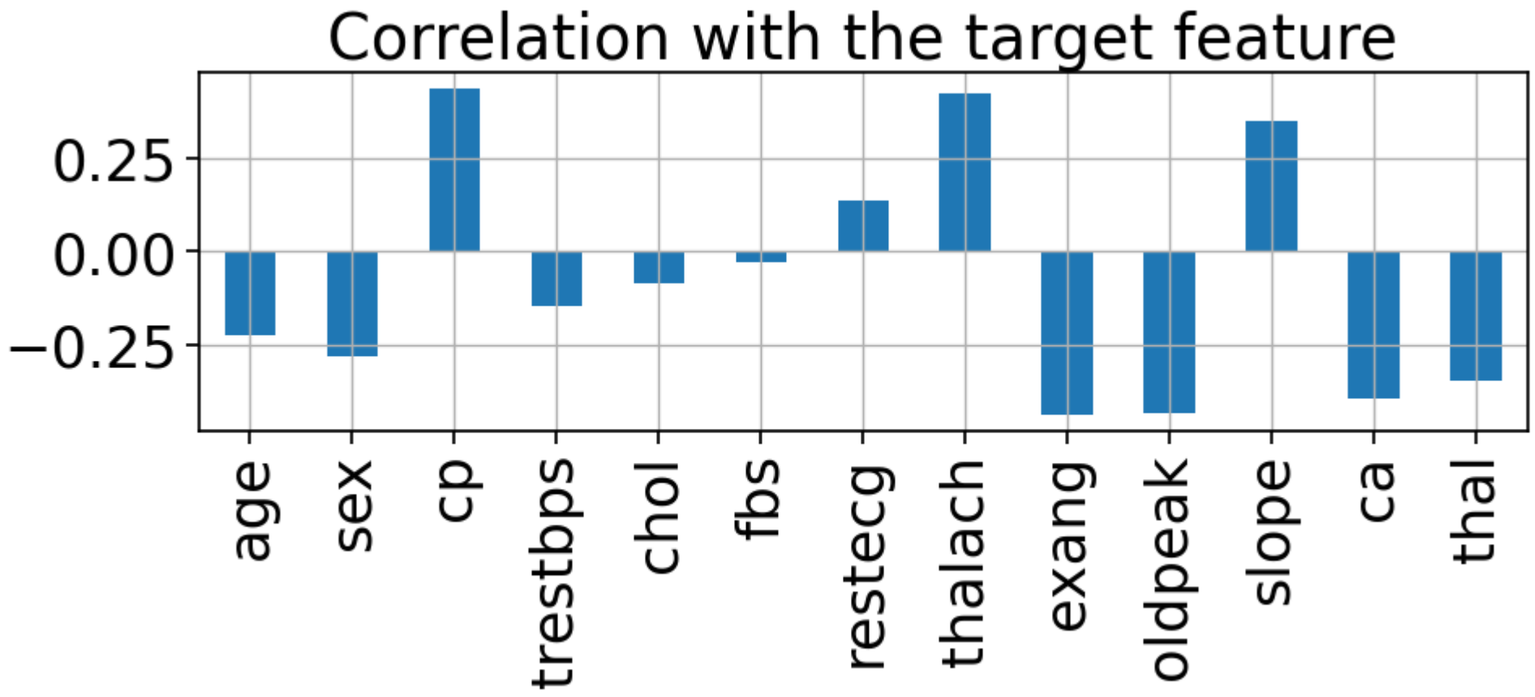
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

In [33]:
```
#creating heatmap using correlation function 'corr()'
plt.figure(figsize=(15,8))
sns.set_context('notebook',font_scale = 1.3)
sns.heatmap(data.corr(),annot=True,linewidth =2)
plt.tight_layout()
```



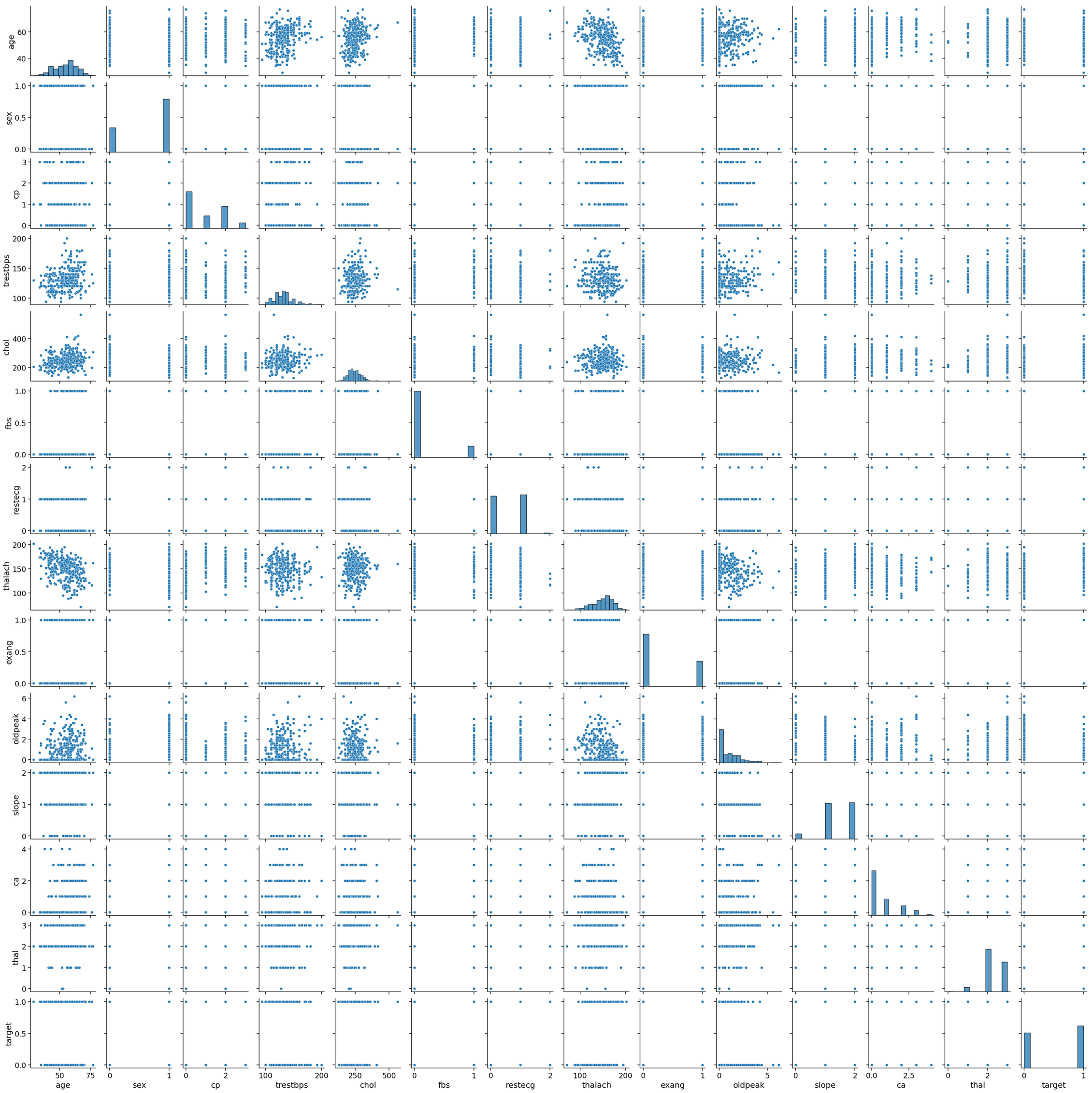
In [34]:
```
#correlation with the Target variable
sns.set_context('notebook',font_scale = 2.3)
data.drop('target', axis=1).corrwith(data.target).plot(kind='bar', grid=True, figsize=(10, 5),
```

```
                                                    title="Correlation with the target feature")
plt.tight_layout()
```

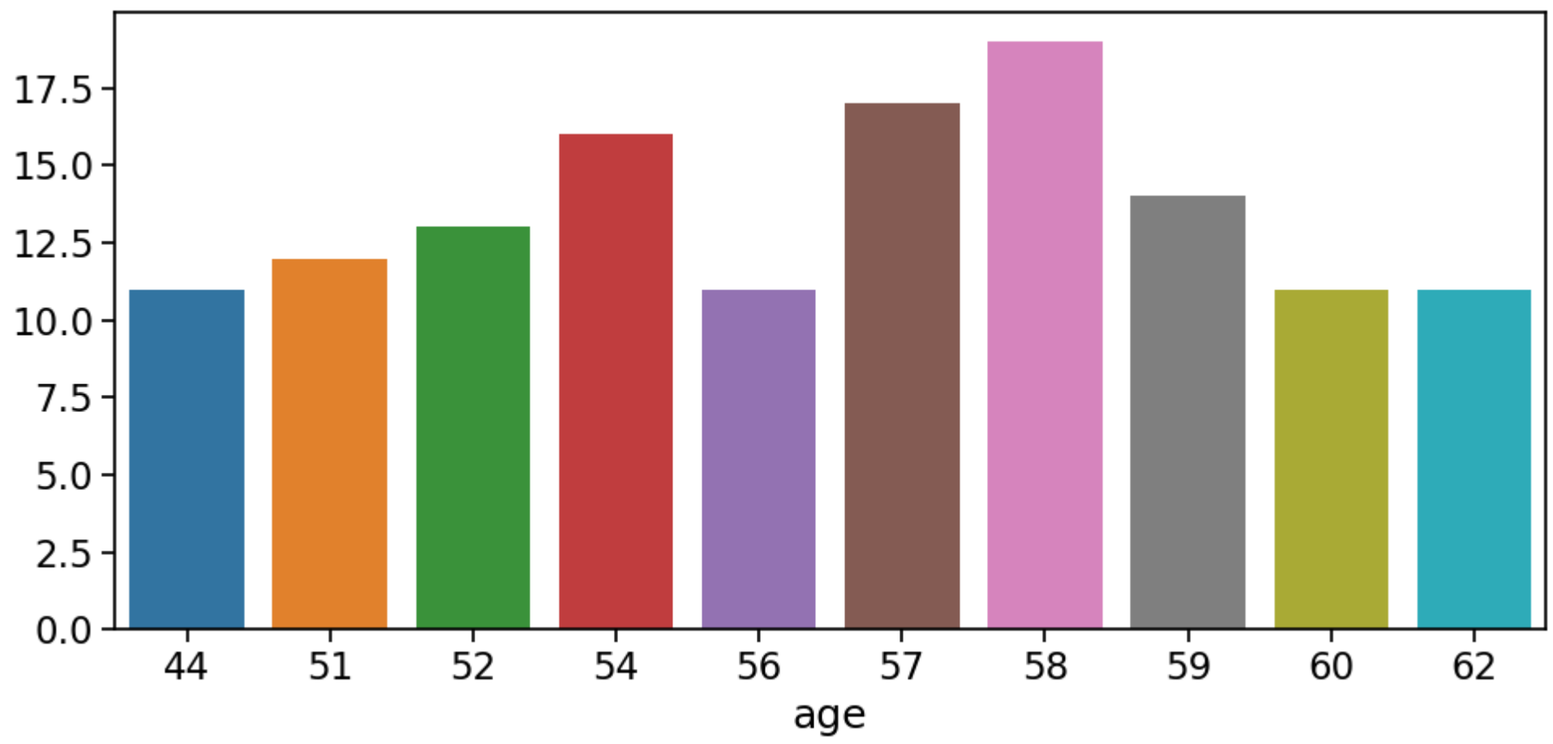


In [30]:
```
sns.pairplot(data=data,kind='scatter')
```

Out[30]:
```
<seaborn.axisgrid.PairGrid at 0x25df4d8a5b0>
```



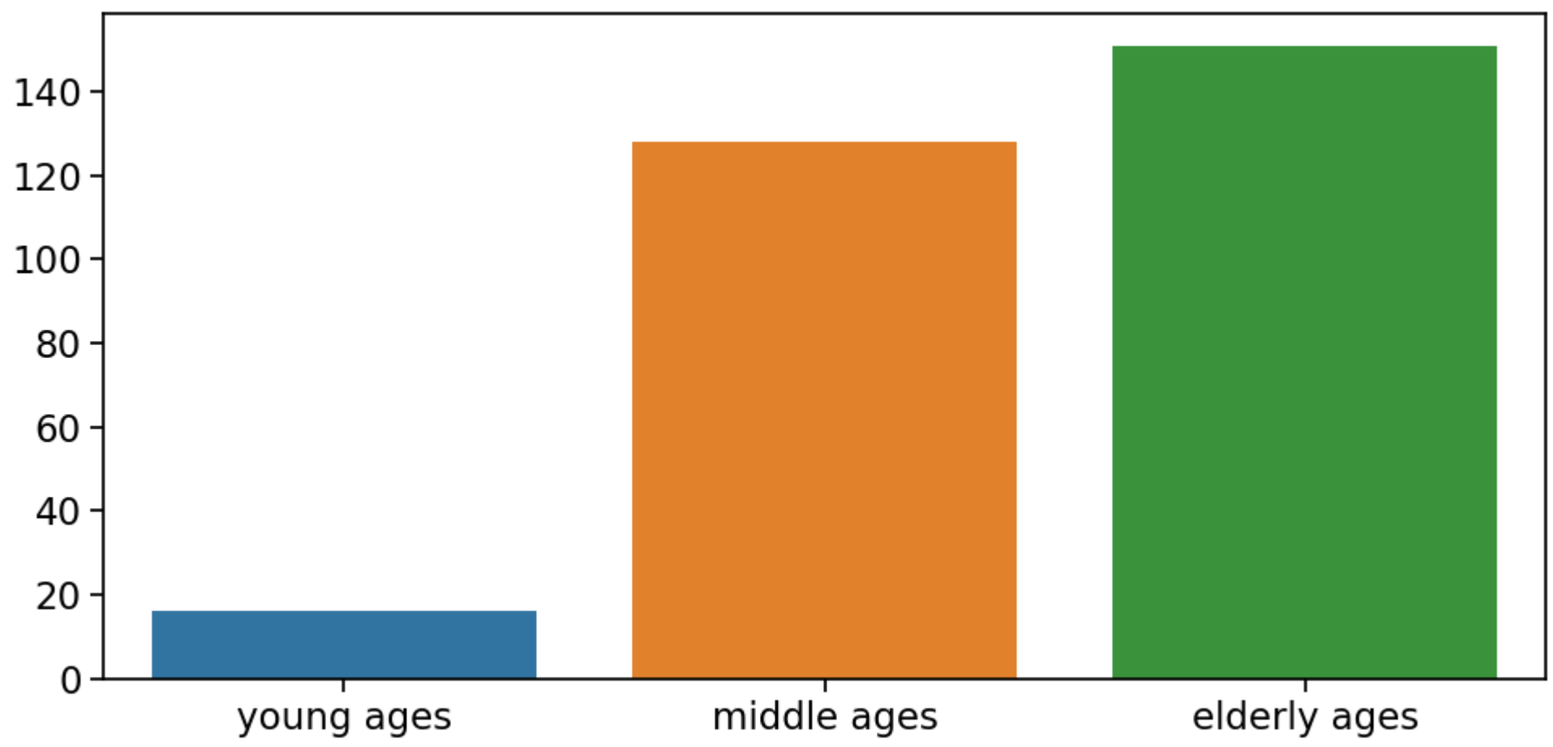
In [35]:
```
# represesnting people with their corrsponding ages
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=data.age.value_counts()[:10].index,y=data.age.value_counts()[:10].values)
plt.tight_layout()
```



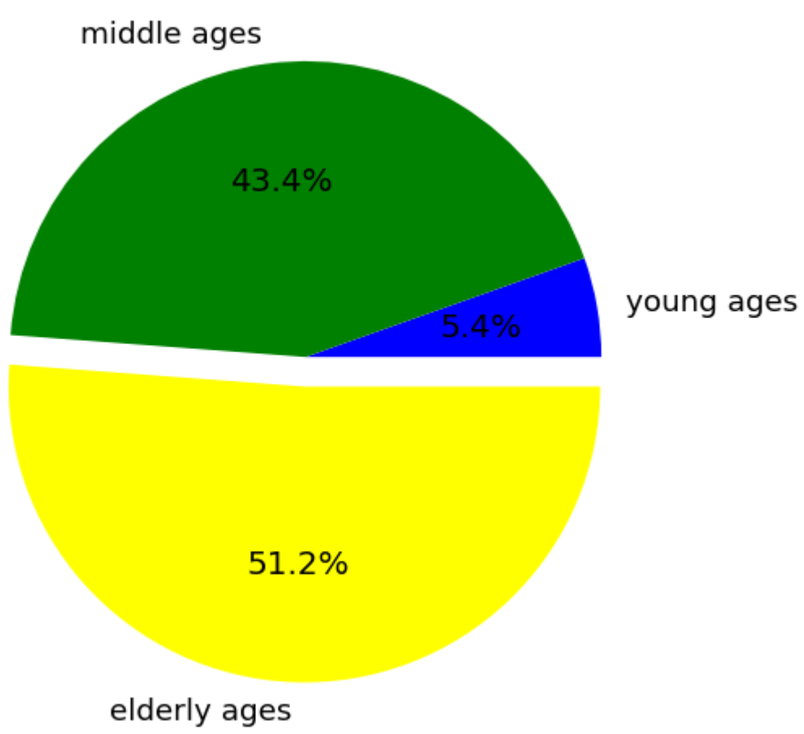
In [11]:
```
# calculating min max and mean age of the people
minAge=min(data.age)
maxAge=max(data.age)
meanAge=data.age.mean()
print('Min Age :',minAge)
print('Max Age :',maxAge)
print('Mean Age :',meanAge)
```

```
Min Age : 29
Max Age : 77
Mean Age : 54.366336633663366
```

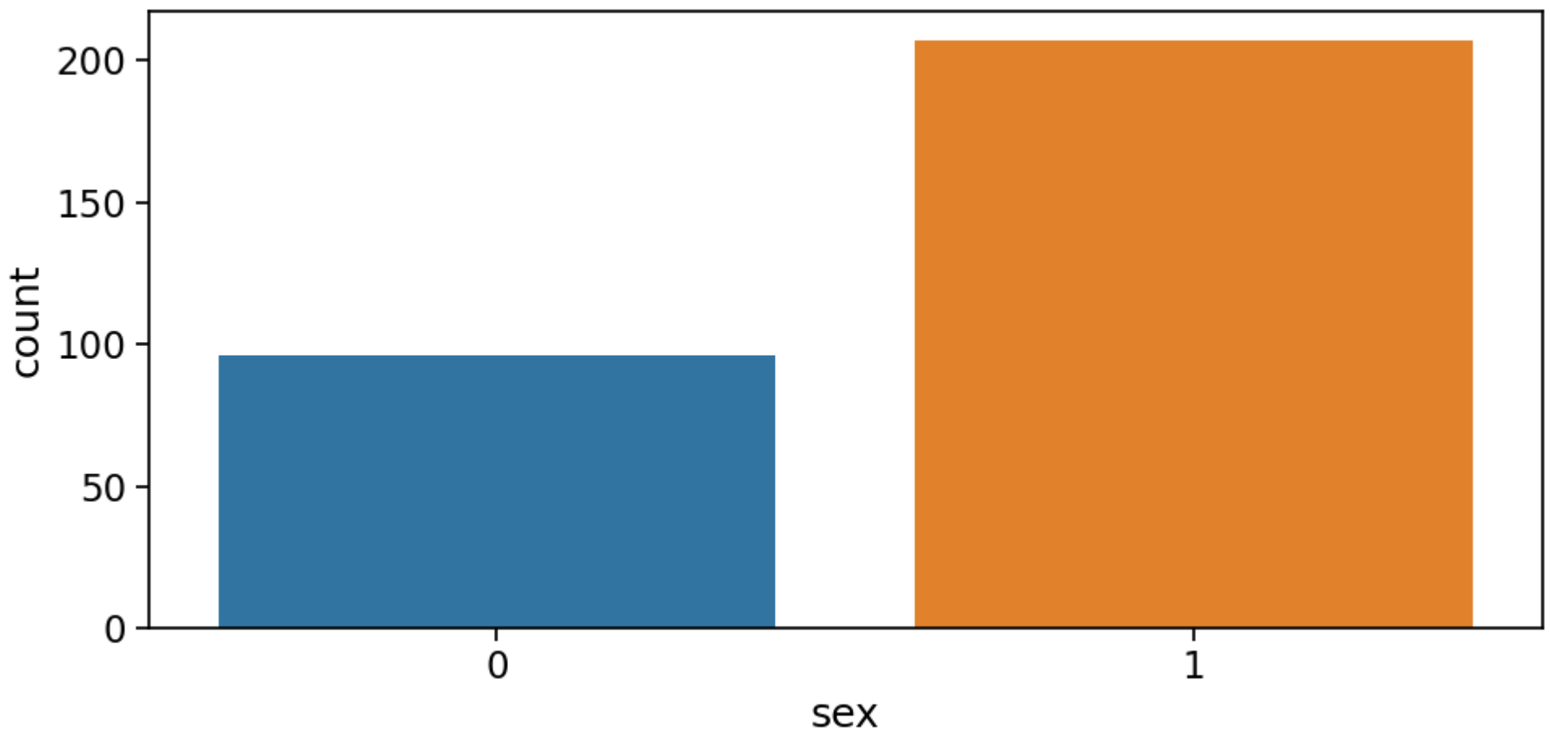
In [36]:
```
#visualisation of people with 3 categories i.e. Young Middle and Elder
Young = data[(data.age>=29)&(data.age<40)]
Middle = data[(data.age>=40)&(data.age<55)]
Elder = data[(data.age>55)]

plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=['young ages','middle ages','elderly ages'],y=[len(Young),len(Middle),len(Elder)])
plt.tight_layout()
```



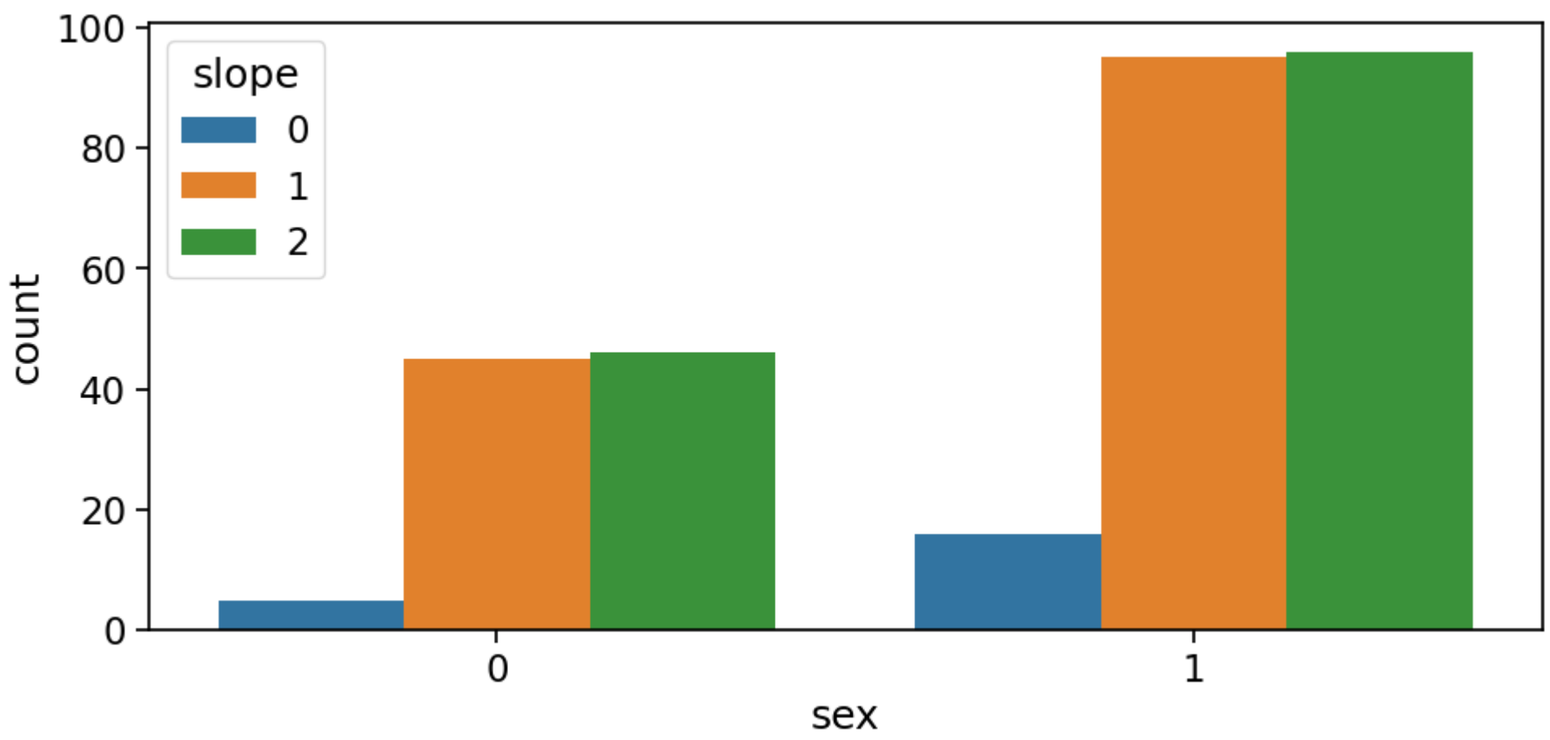
In [37]:
```
# pie chart for age distribution
colors = ['blue','green','yellow']
explode = [0,0,0.1]
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.2)
plt.pie([len(Young),len(Middle),len(Elder)],labels=['young ages','middle ages','elderly ages'],explode=explode,colors=colors, autopct='%1.1f%%')
plt.tight_layout()
```



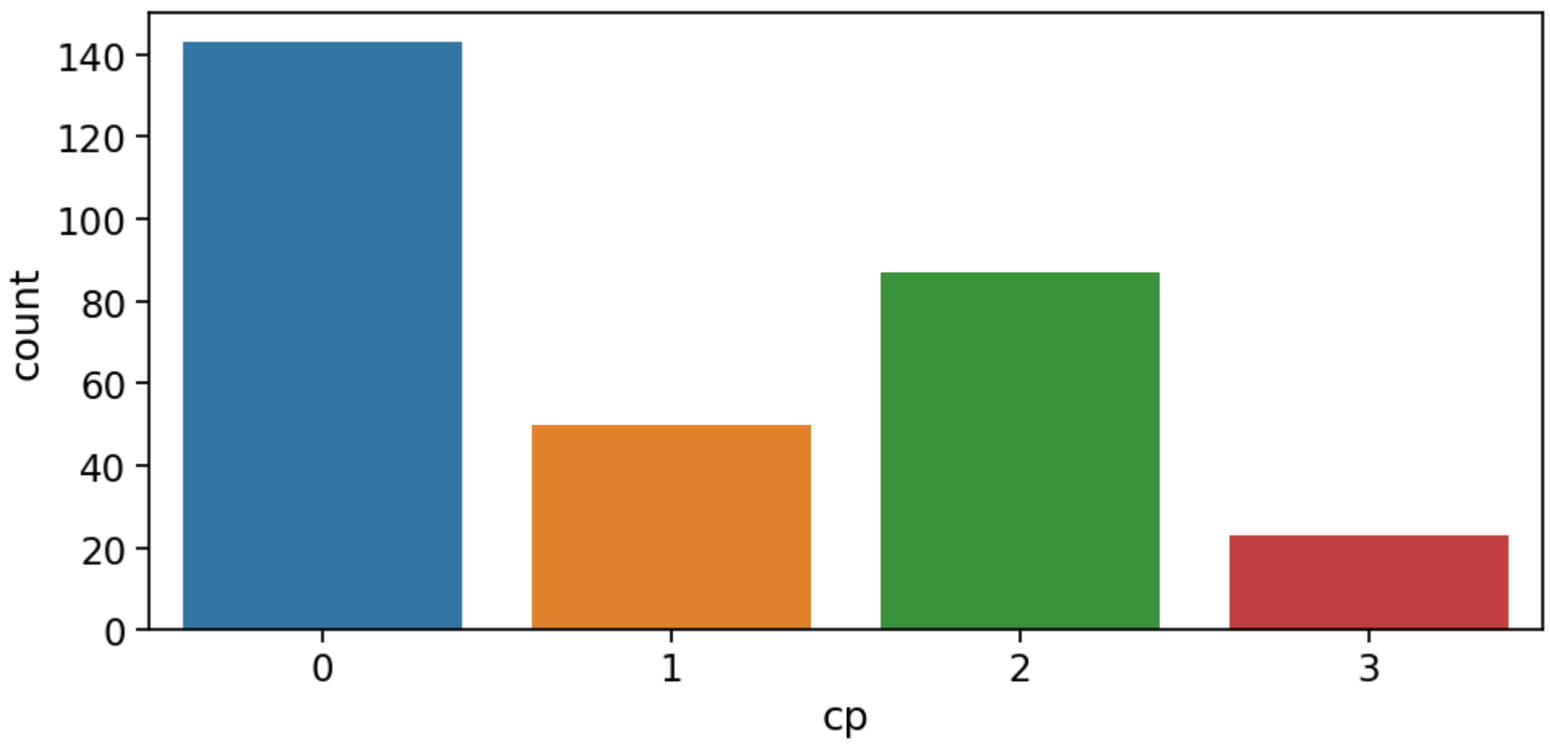
In [38]:
```
#counting the male and female pop from the data
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data=data,x='sex')
plt.tight_layout()
```



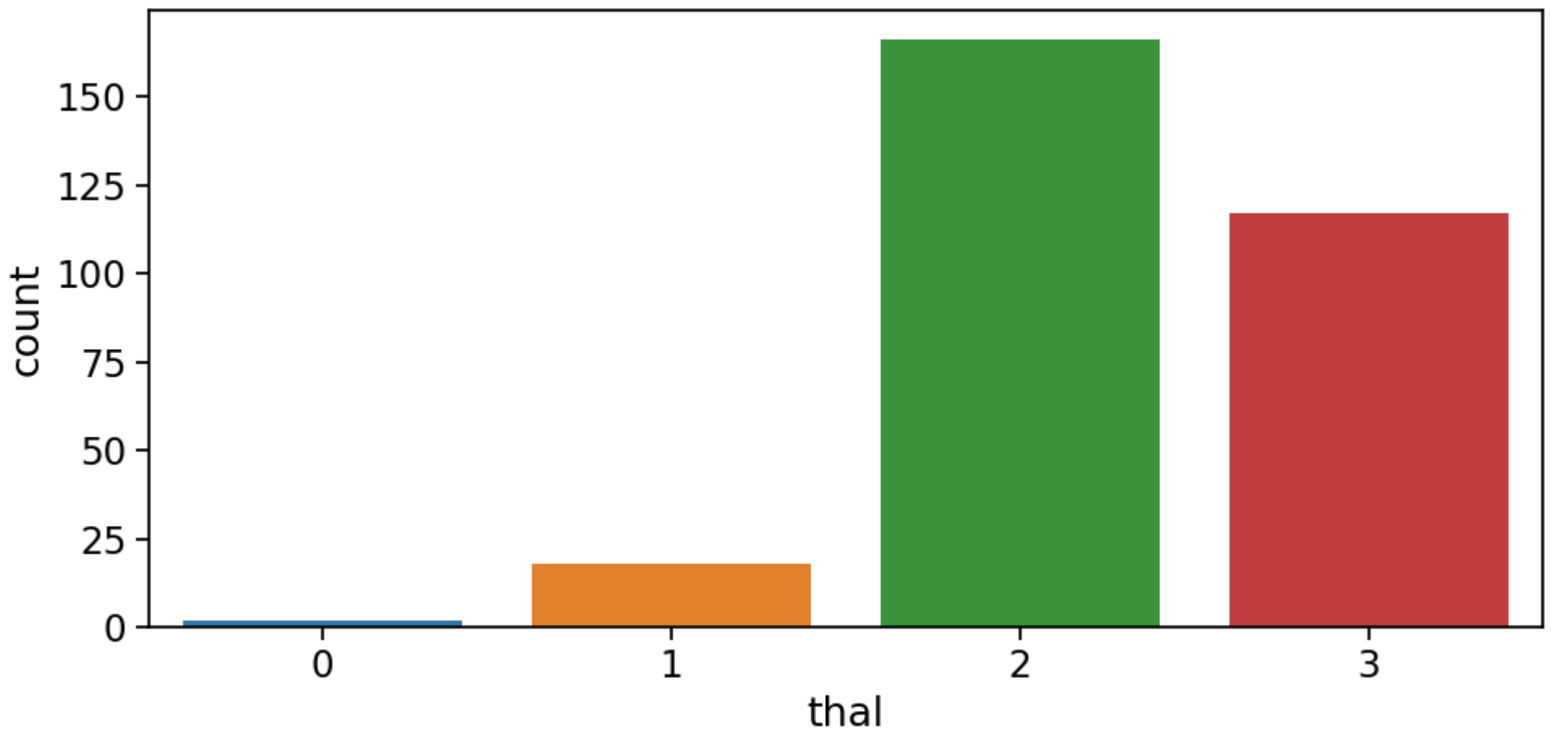
In [39]:
```
# bifurcating male and female wrt 'slope'
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data=data,x='sex',hue=data["slope"])
plt.tight_layout()
```



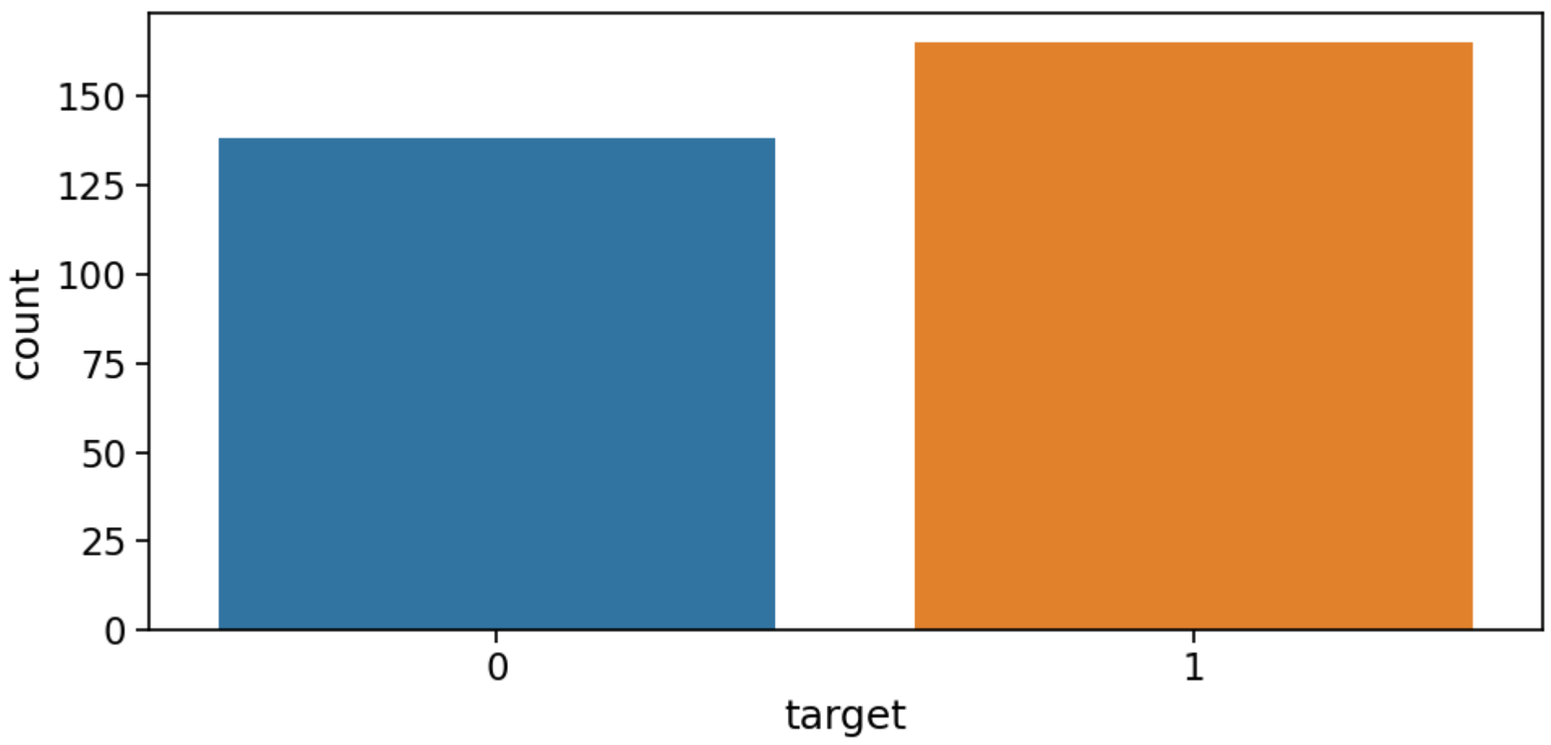
In [40]:
```
# cp count in people
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data=data,x='cp')
plt.tight_layout()
```



In [41]:
```
#thal count amongst people
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data=data,x='thal')
plt.tight_layout()
```



In [42]:
```
# representation of CVD if yes = 1 if no = 0;
plt.figure(figsize=(10,5))
sns.set_context('notebook',font_scale = 1.5)
sns.countplot(data=data,x='target')
plt.tight_layout()
```



In [19]:
```
# identifying the unique values for better understanding
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 : [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 [20]:
```
# seperating target column
categorical_val.remove('target')
```

In [21]:
```
# creating dummy variables for better readability
dfs = pd.get_dummies(data, columns = categorical_val,dtype=int)
dfs.head(6)
```

Out[21]:

	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	thal_2	thal_3
0	63	145	233	150	2.3	1	0	1	0	0	...	0	1	0	0	0	0	0	1	0	0
1	37	130	250	187	3.5	1	0	1	0	0	...	0	1	0	0	0	0	0	0	1	0
2	41	130	204	172	1.4	1	1	0	0	1	...	1	1	0	0	0	0	0	0	1	0
3	56	120	236	178	0.8	1	0	1	0	1	...	1	1	0	0	0	0	0	0	1	0
4	57	120	354	163	0.6	1	1	0	1	0	...	1	1	0	0	0	0	0	0	1	0
5	57	140	192	148	0.4	1	0	1	1	0	...	0	1	0	0	0	0	0	1	0	0

6 rows × 31 columns

In [22]:
```
# scaling the data using standard scalar
sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dfs[col_to_scale] = sc.fit_transform(dfs[col_to_scale])
dfs.head(6)
```

Out[22]:

	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	thal_2	thal_3
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	0	1	0	0	...	0	1	0	0	0	0	0	1	0	0
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	0	1	0	0	...	0	1	0	0	0	0	0	0	1	0
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	1	0	0	1	...	1	1	0	0	0	0	0	0	1	0
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	0	1	0	1	...	1	1	0	0	0	0	0	0	1	0
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	1	0	1	0	...	1	1	0	0	0	0	0	0	1	0
5	0.290464	0.478391	-1.048678	-0.072018	-0.551783	1	0	1	1	0	...	0	1	0	0	0	0	0	1	0	0

6 rows × 31 columns

Train Test Split and mdoel building

In [23]:
```
#importing train test split
X = dfs.drop('target', axis=1)
y = dfs.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [24]:
```
# Importing Logistic Regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train,y_train)*100,2)
acc_log
```

Out[24]:
```
86.79
```

In [25]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [26]:
```
# comparing KNN performance
knn = KNeighborsClassifier(n_neighbors = 10)
knn.fit(X_train,y_train)

y_pred1 = knn.predict(X_test.values)

print(accuracy_score(y_test,y_pred1)*100)
```

```
85.71428571428571
```

In [27]:
```
# comparing Random Forest Performance
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)
rf.fit(X_train,y_train)
y_pred2 = rf.predict(X_test)
```

In [28]:
```
print(accuracy_score(y_test,y_pred2)*100)
```

```
82.41758241758241
```

Logistic Regression is giving most accurate value with 88% accuracy

In [ ]:
```

```