Heart Disease Diagnosis

Overview

In this article we will be closely working with the heart disease diagnosis and for that we will be looking into the heart disease dataset and from that dataset we will derieve various insights that helps us know the weightage of each feature and how they are interrelated to each other but this time our soul aim is to detect the probablity of person that will be effected by a saviour heart problem or not.

Takeaways from this article

1. Data insight: As mentioned here we will be working with the heart disease detection dataset and we will be putting out interesting inference from the data to derieve some meaningful results.
2. EDA: Exploratory data analysis is the key step for the getting the meaningful results.
3. Feature engineering: After getting the insights from the data we have to alter the features so that they can move forward for the model building phase.
4. Model building: In this phase we will be building our Machine learning model for heart disease detection.

Importing Necessary Libraries

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

# Metrics for Classification technique
from sklearn.metrics import classification_report,confusion_matrix,accuracy_sc

# Scaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import  RandomizedSearchCV, train_test_split

from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

Data Loading

Our first step is to extract train and test data.

In [2]:
```
# Importing Data

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

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

Exploratory Data Analysis

In [3]:
```
#Size of the dataset
data.shape
```

Out[3]:
```
(303, 14)
```

I have a dataset with 303 rows which indicates a smaller set of data.

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

- Out of 14 features, I have 13 int type and only one with float data type.
- Woah! We have no missing values in our dataset.

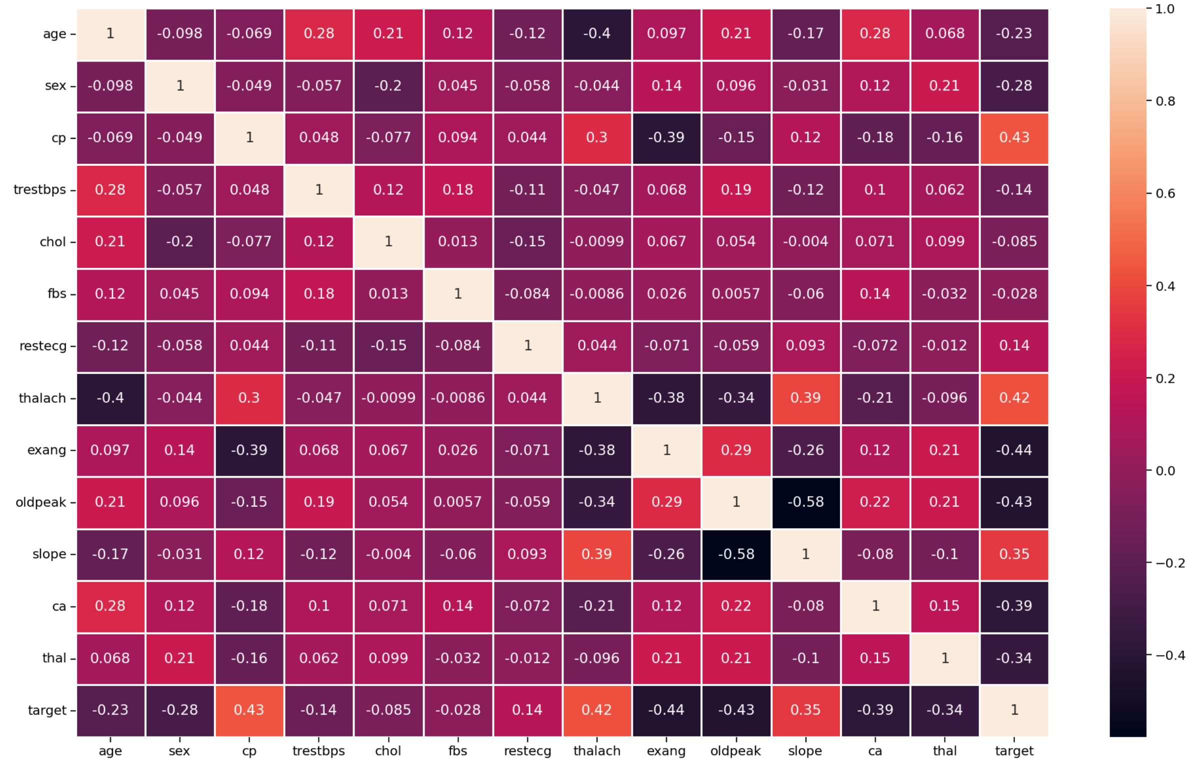
In [5]:
```
data.describe()
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202

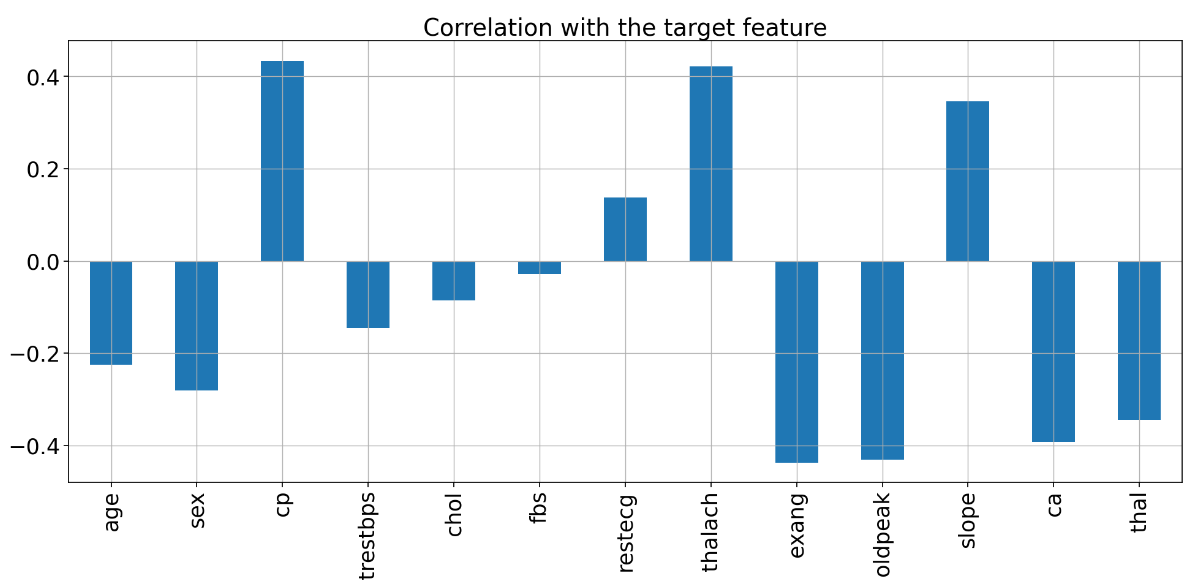
Let's check correleation between various features.

In [6]:
```
plt.figure(figsize=(20,12))
sns.set_context('notebook',font_scale = 1.3)
sns.heatmap(data.corr(),annot=True,linewidth =2)
plt.tight_layout()
```



Let's check the correlation of various features with the target feature.

In [7]:
```
# plt.figure(figsize=(20,12))
sns.set_context('notebook',font_scale = 2.3)
data.drop('target', axis=1).corrwith(data.target).plot(kind='bar', grid=True,
                                                       title="Correlation with the
plt.tight_layout()
```

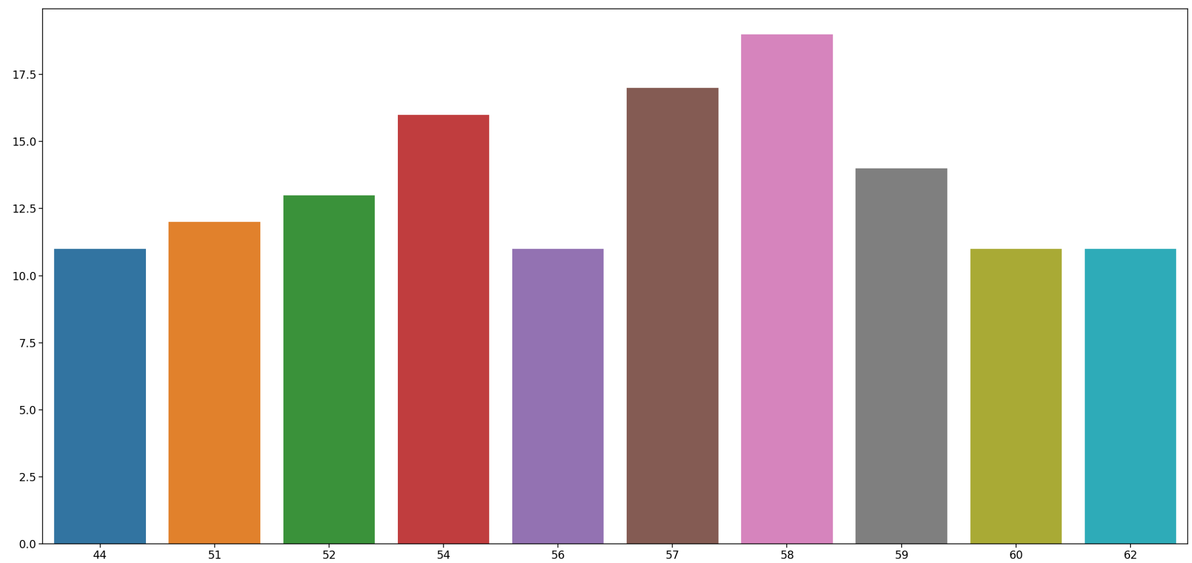


- Four feature("cp", "restecg", "thalach", "slope") are positively correlated with the target feature.
- Other features are negatively correlated with the target feature.

Individual Feature Analysis

Age("age") Analysis

In [8]:
```
# Let's check 10 ages and their count

plt.figure(figsize=(25,12))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=data.age.value_counts()[:10].index,y=data.age.value_counts()[:10
plt.tight_layout()
```



Let's check the range of age in the dataset.

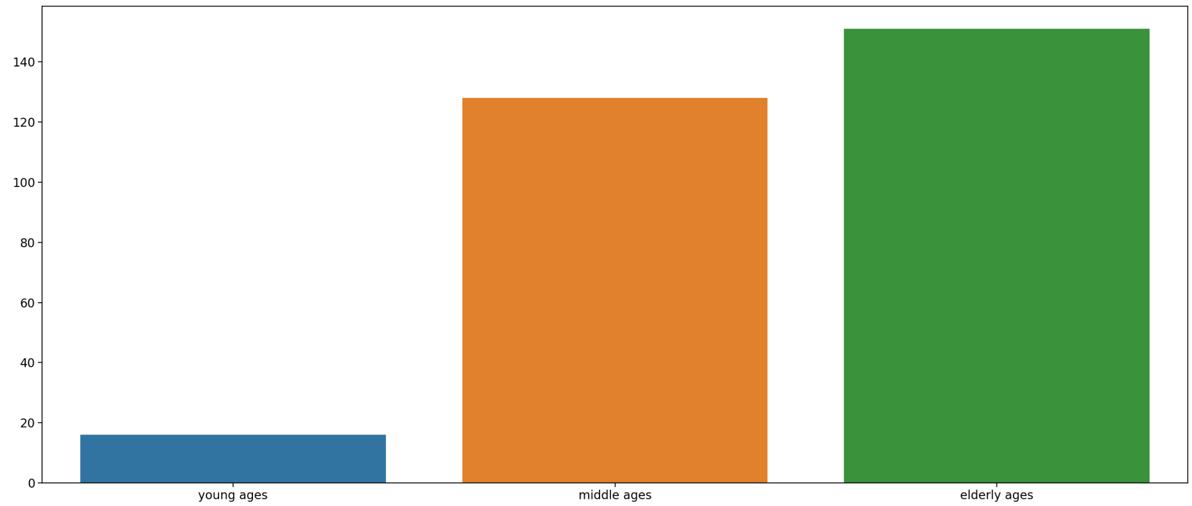
In [9]:
```
minAge=min(data.age)
maxAge=max(data.age)
meanAge=data.age.mean()
print('Min Age :',minAge)
print('Max Age :',maxAge)
print('Mean Age :',meanAge)
```

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

I should divide the Age feature into three parts - "Young", "Middle" and "Elder"

In [10]:
```
Young = data[(data.age>=29)&(data.age<40)]
Middle = data[(data.age>=40)&(data.age<55)]
Elder = data[(data.age>55)]
```

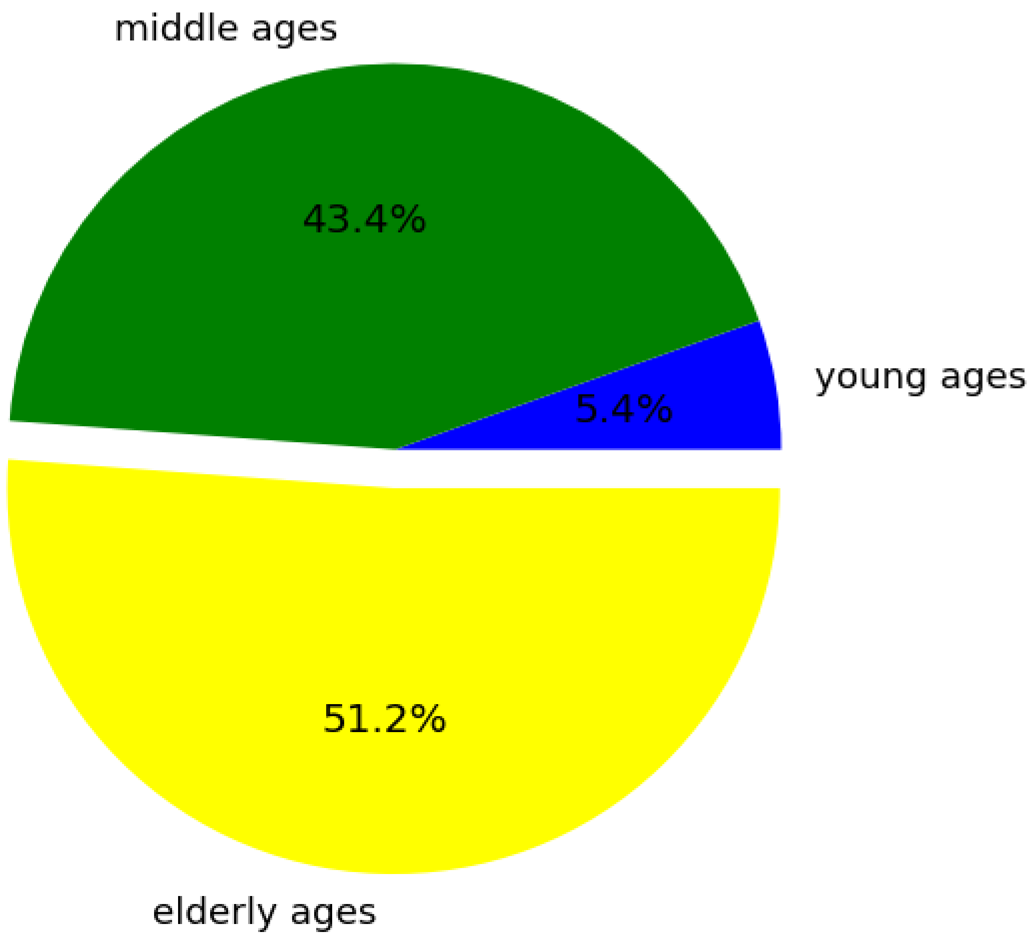
In [11]:
```
plt.figure(figsize=(23,10))
sns.set_context('notebook',font_scale = 1.5)
sns.barplot(x=['young ages','middle ages','elderly ages'],y=[len(Young),len(Mi
plt.tight_layout()
```



A large proportion of dataset contains Elder people.

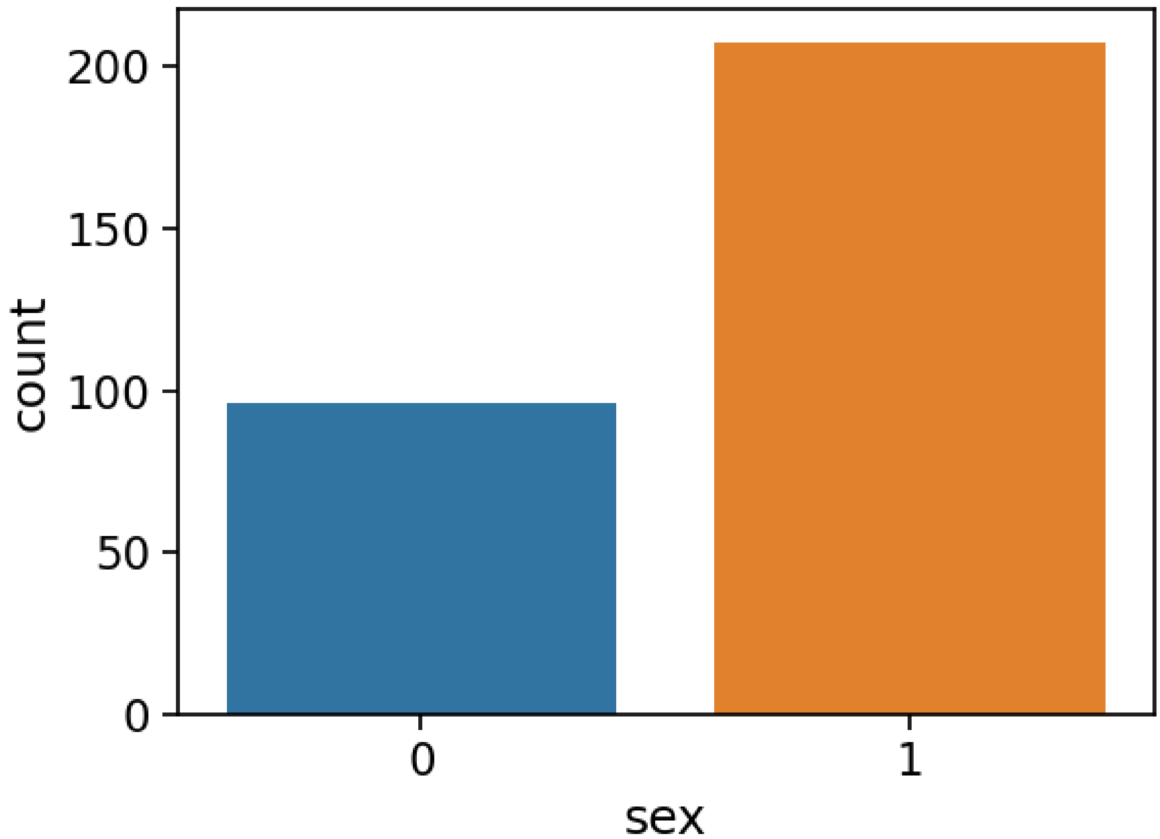
Elderly people are more likely to suffer from heart disease.

In [12]:
```
colors = ['blue','green','yellow']
explode = [0,0,0.1]
plt.figure(figsize=(6,6))
sns.set_context('notebook',font_scale = 1.2)
plt.pie([len(Young),len(Middle),len(Elder)],labels=['young ages','middle ages'
plt.tight_layout()
```



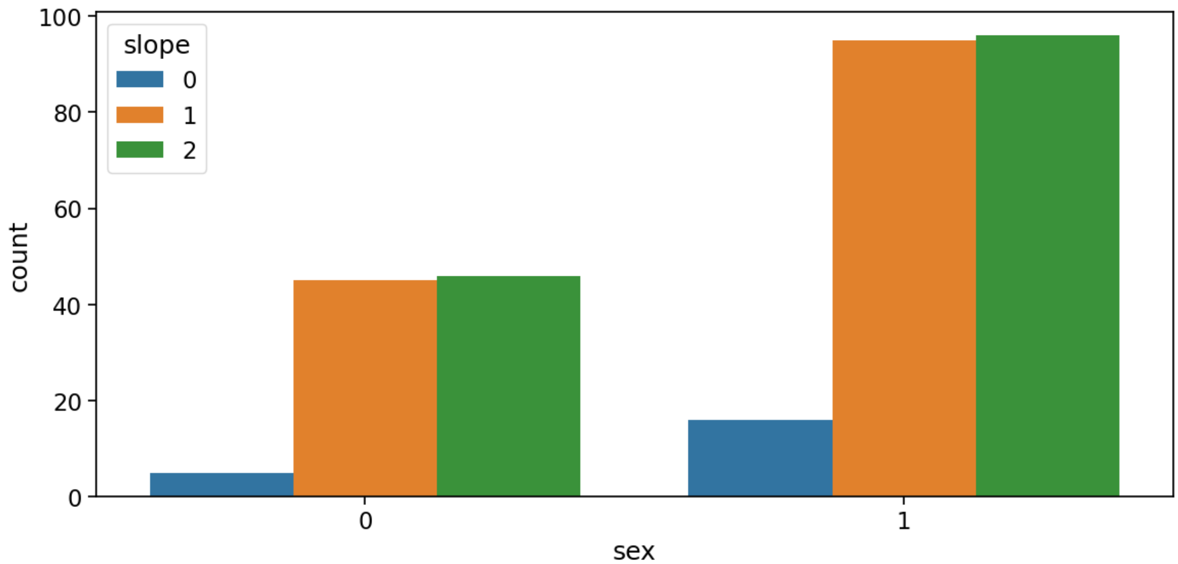
Sex("sex") Feature Analysis

In [13]:
```
sns.set_context('notebook', font_scale=1.5)
sns.countplot(x='sex', data=data)
plt.tight_layout()
plt.show()
```

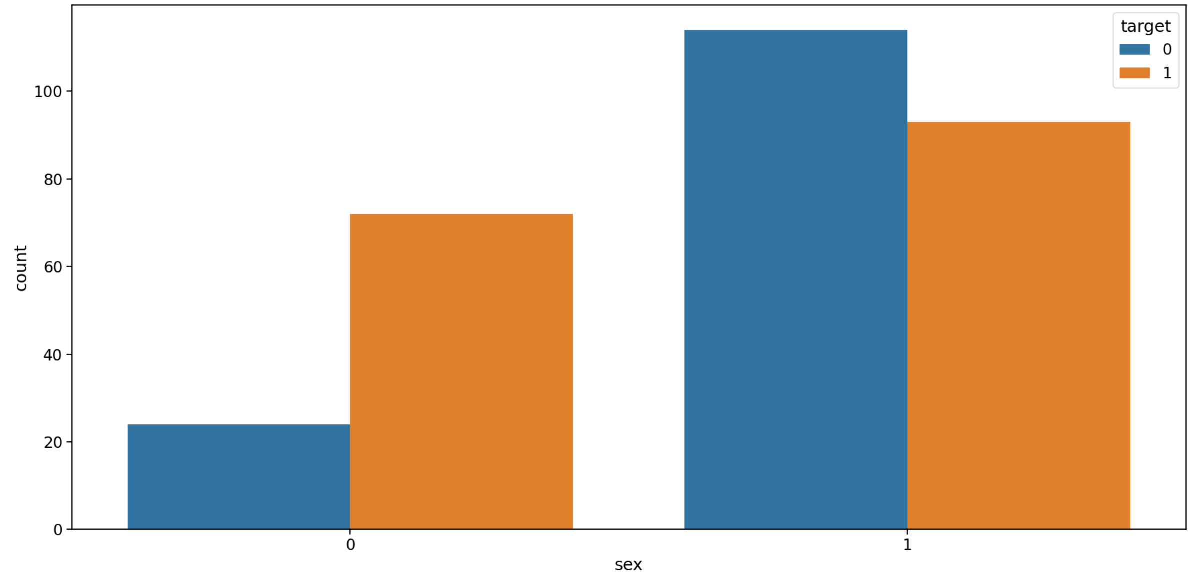


Ratio of Male to Female is approx 2:1

In [14]:
```
# Let's plot the relation between sex and slope.

plt.figure(figsize=(12, 6))
sns.set_context('notebook', font_scale=1.5)

# Assuming 'sex' is on the x-axis and 'slope' is on the y-axis
sns.countplot(x='sex', hue='slope', data=data)

plt.tight_layout()
plt.show()
```



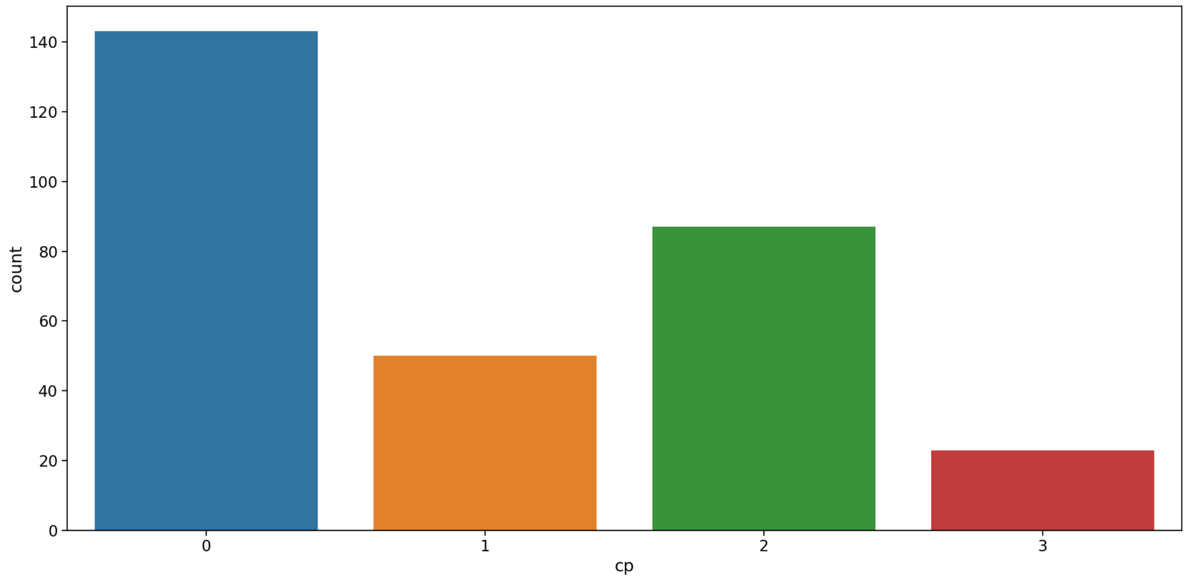
In [15]:
```
# Let's plot the relation between sex and target.

plt.figure(figsize=(18, 9))
sns.set_context('notebook', font_scale=1.5)

# Assuming 'sex' is on the x-axis
sns.countplot(x='sex', hue='target', data=data)

plt.tight_layout()
plt.show()
```



Males are more likely to have heart disease than Female.

Chest Pain Type("cp") Analysis

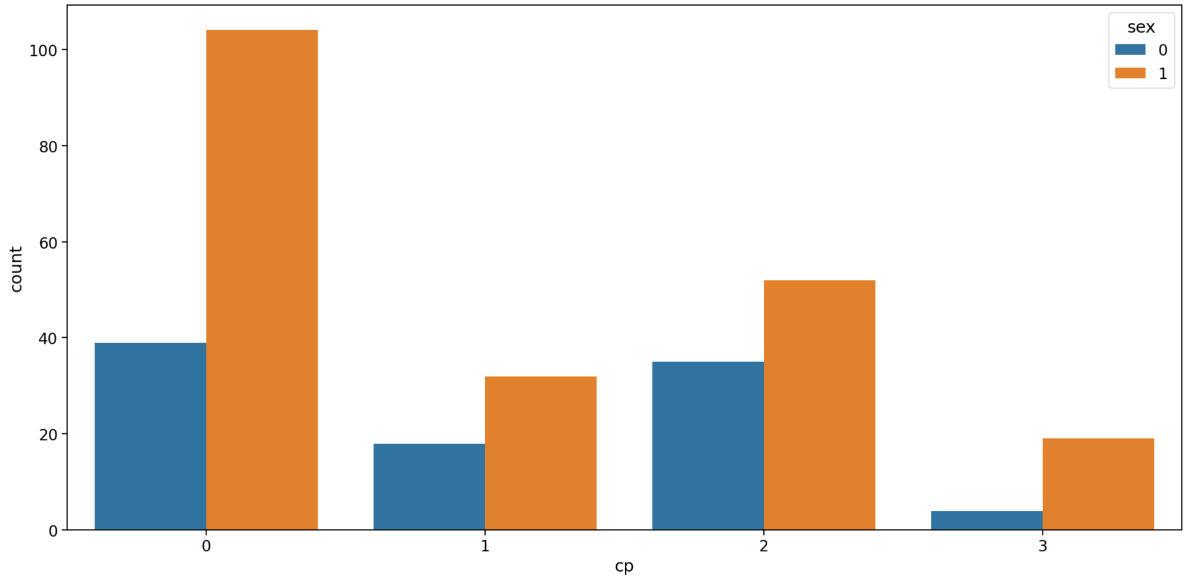
In [16]:
```
plt.figure(figsize=(18, 9))
sns.set_context('notebook', font_scale=1.5)
sns.countplot(x='cp', data=data)
plt.tight_layout()
plt.show()
```



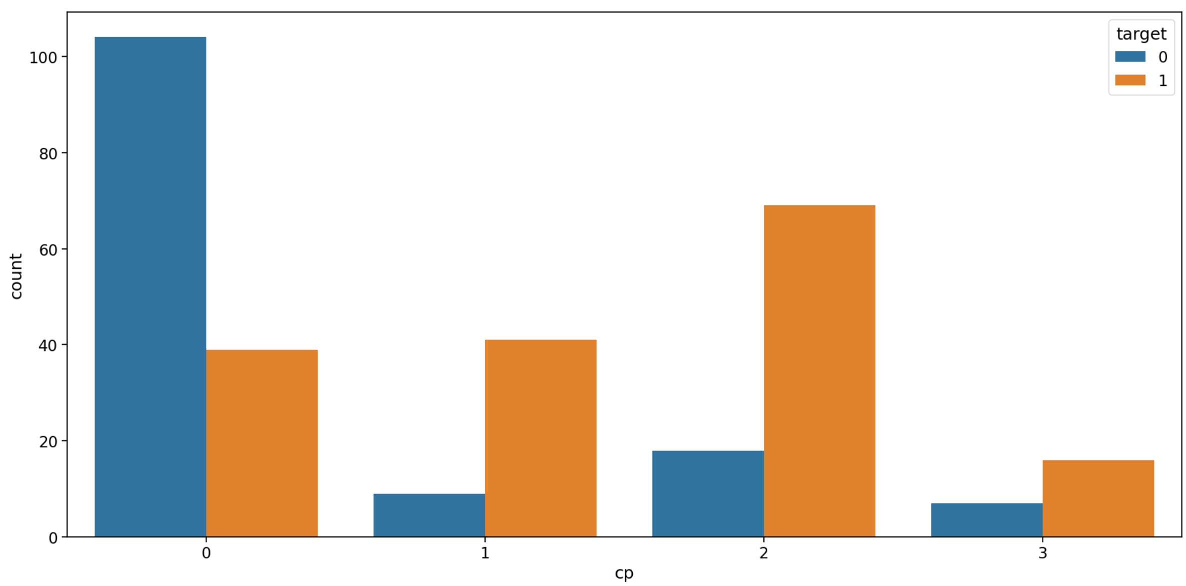
As seen, there are 4 types of chest pain

1. status at least
2. condition slightly distressed
3. condition medium problem
4. condition too bad

In [17]:
```
plt.figure(figsize=(18, 9))
sns.set_context('notebook', font_scale=1.5)

# Assuming 'cp' is on the x-axis
sns.countplot(x='cp', hue='sex', data=data)

plt.tight_layout()
plt.show()
```



In [18]:
```
plt.figure(figsize=(18,9))
sns.set_context('notebook', font_scale=1.5)
sns.countplot(x='cp', hue='target', data=data)
plt.tight_layout()
plt.show()
```

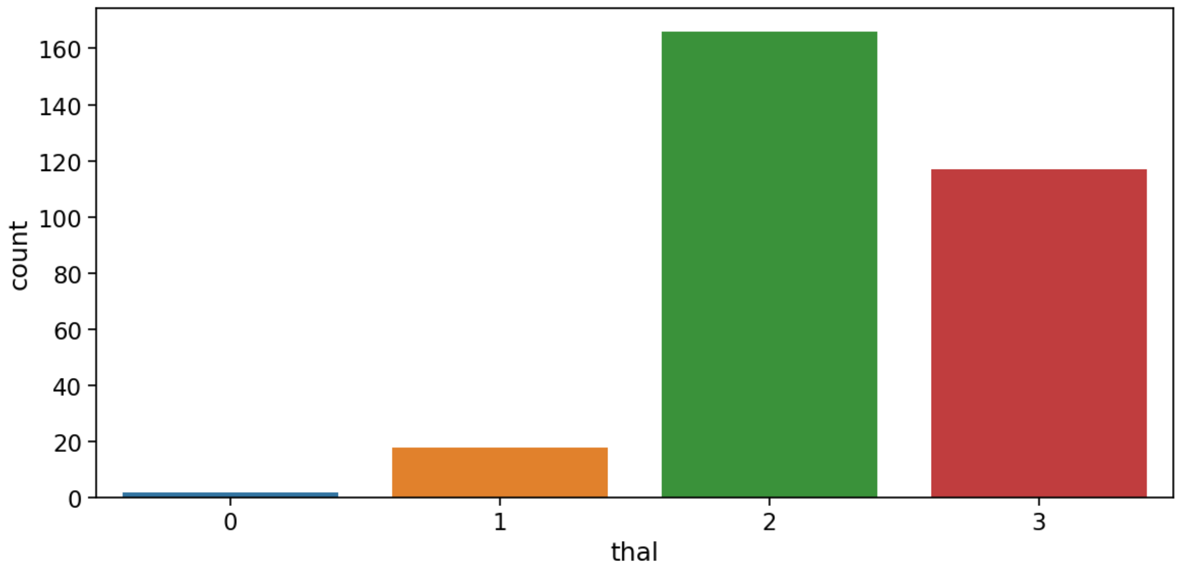


- People having least chest pain are not likely to heart disease.
- People having severe chest pain are likely to heart disease.

Elderly people are more likely to have chest pain.

Thal Analysis

In [19]:
```
plt.figure(figsize=(12, 6))
sns.set_context('notebook', font_scale=1.5)
sns.countplot(x='thal', data=data)
plt.tight_layout()
plt.show()
```



In [20]:
```
data.head()
```

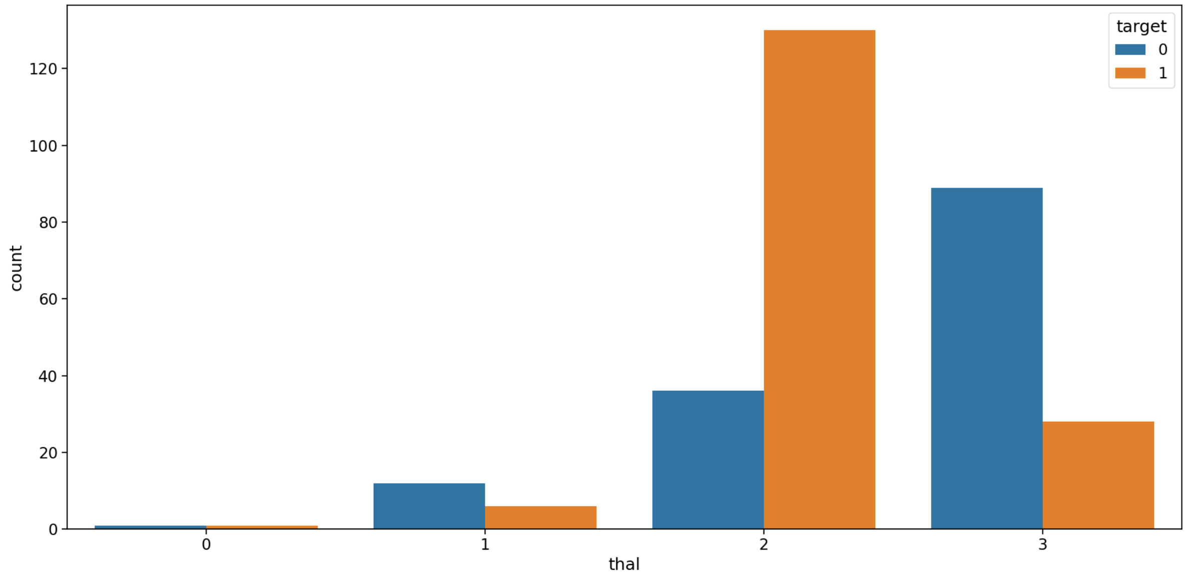
Out[20]:

	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

1. 3 = normal
2. 6 = fixed defect
3. 7 = reversable defect

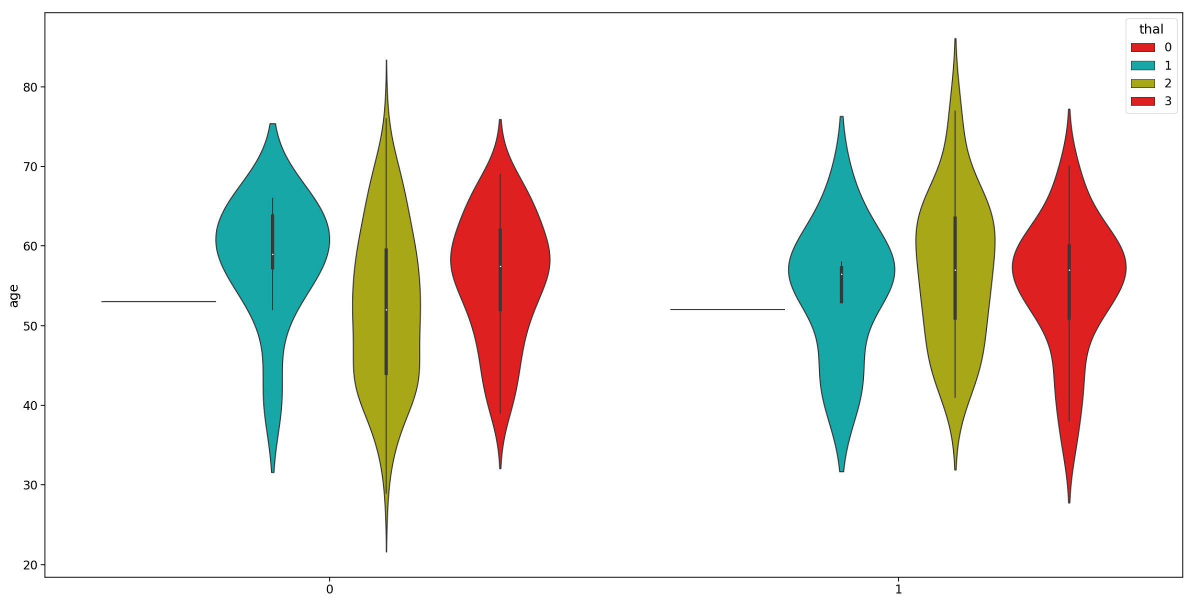
In [21]:
```
plt.figure(figsize=(18,9))
plt.figure(figsize=(18, 9))
sns.set_context('notebook', font_scale=1.5)
sns.countplot(x='thal', hue='target', data=data)
plt.tight_layout()
plt.show()
```

```
<Figure size 1800x900 with 0 Axes>
```



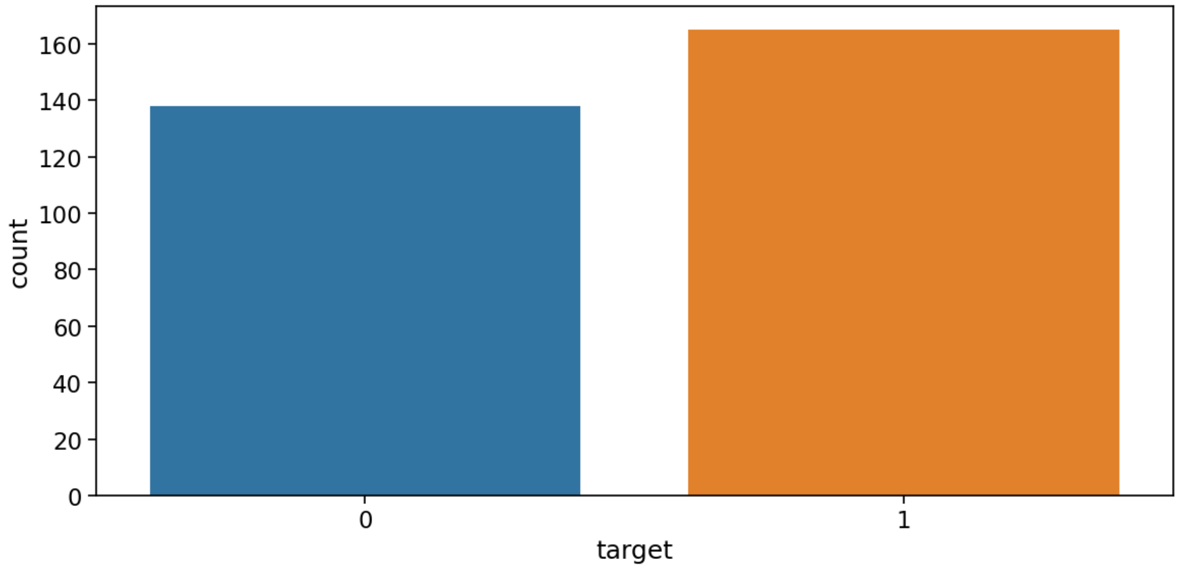
People with fixed defect are more likely to have heart disease.

In [22]:
```
plt.figure(figsize=(23,12))
sns.set_context('notebook',font_scale = 1.5)
sns.violinplot(x="exang",y="age",data=data,palette=["r", "c", "y"],hue="thal")
plt.tight_layout()
```



Target

In [23]:
```
plt.figure(figsize=(12, 6))
sns.set_context('notebook', font_scale=1.5)
sns.countplot(x='target', data=data)
plt.tight_layout()
plt.show()
```



Inference: The ratio between 1 and 0 is much less than 1.5 which indicates that target feature is not imbalanced. So for a balanced dataset, we can use accuracy_score as evaluation metrics for our model.

In [24]:
```
data.head()
```

Out[24]:

	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

Feature Enginnering

In [25]:
```
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 5
3
 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 1
08 134
 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 2
26
 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 15
1 161
 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]
```

Now here first we will be removing the target column from our set of features then we will categorised all the categorical variables using get dummies method which will create the seperate column for each category suppose X variable contains 2 types of unique values then it

In [26]:
```
categorical_val.remove('target')
dfs = pd.get_dummies(data, columns = categorical_val)
```

In [27]:
```
dfs.head(6)
```

Out[27]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...	slope_2	ca_0
0	63	145	233	150	2.3	1	0	1	0	0	...	0	1
1	37	130	250	187	3.5	1	0	1	0	0	...	0	1
2	41	130	204	172	1.4	1	1	0	0	1	...	1	1
3	56	120	236	178	0.8	1	0	1	0	1	...	1	1
4	57	120	354	163	0.6	1	1	0	1	0	...	1	1
5	57	140	192	148	0.4	1	0	1	1	0	...	0	1

6 rows × 31 columns

Now we will be using the standard scaler method to scale down the data so that it won't raise the outliers also dataset which is scaled to general units leads to have better accuracy.

In [28]:
```
sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dfs[col_to_scale] = sc.fit_transform(dfs[col_to_scale])
```

In [29]:
```
dfs.head(6)
```

Out[29]:

	age	trestbps	chol	thalach	oldpeak	target	sex_0	sex_1	cp_0	cp_1	...
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1	0	1	0	0	...
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1	0	1	0	0	...
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1	1	0	0	1	...
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1	0	1	0	1	...
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1	1	0	1	0	...
5	0.290464	0.478391	-1.048678	-0.072018	-0.551783	1	0	1	1	0	...

6 rows × 31 columns

Modelling

Splitting our dataset

In [30]:
```
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, rando
```

In [31]:
```
X_train.head()
```

Out[31]:

	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	cp_2	...	s
124	-1.694735	-2.148802	-0.913400	1.283627	-0.896862	1	0	0	0	1	...	
72	-2.797624	-0.092738	-0.816773	2.289429	-0.896862	0	1	0	1	0	...	
15	-0.481558	-0.663867	-0.526890	0.365287	0.483451	1	0	0	0	1	...	
10	-0.040403	0.478391	-0.140381	0.452748	0.138373	0	1	1	0	0	...	
163	-1.805024	0.364165	-1.377212	1.021244	-0.896862	0	1	0	0	1	...	

5 rows × 30 columns

Next I will work on following algorithms -

- KNN
- Random Forest Classifier
- XGBoost
- CatBoost

KNN

In [32]:
```
knn = KNeighborsClassifier(n_neighbors = 10)
```

In [33]:
```
knn.fit(X_train,y_train)
```

Out[33]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

In [34]:
```
print(X_test.flags)
```

```
<Flags(allows_duplicate_labels=True)>
```

In [35]:
```
X_test = np.ascontiguousarray(X_test)
```

In [36]:
```
y_pred1 = knn.predict(X_test)
```

```
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
```

In [37]:
```
print(accuracy_score(y_test,y_pred1))
```

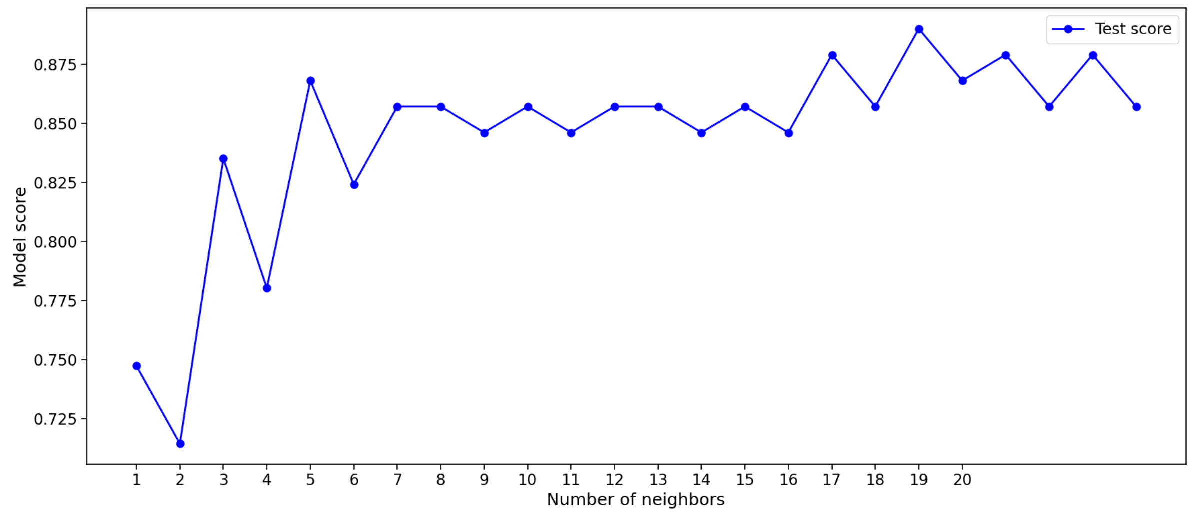
```
0.8571428571428571
```

In [38]:
```
# Hyperparameter Optimization

test_score = []
neighbors = range(1, 25)

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

```
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
```

```
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
```

In [39]:
```
plt.figure(figsize=(18, 8))
plt.plot(neighbors, test_score, label="Test score", marker='o', linestyle='-',
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()
plt.tight_layout()
plt.show()
```



At K=19, i am getting highest test accuracy.

In [40]:
```
knn = KNeighborsClassifier(n_neighbors = 19)
```

In [41]:
```
knn.fit(X_train,y_train)
```

Out[41]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=19)
```

In [42]:
```
y_pred1 = knn.predict(X_test)
```

```
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature na
mes
  warnings.warn(
```

In [43]:
```
print(accuracy_score(y_test,y_pred1))
```

```
0.8901098901098901
```

I achieved accuracy 89% with KNN Model after Hyperparameter Optimization.

Random Forest Classifier

In [44]:
```
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
y_pred2 = rfc.predict(X_test)
```

```
D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
```

In [45]:
```
print(accuracy_score(y_test,y_pred2))
```

```
0.8351648351648352
```

In [46]:
```
## Hyperparameter Optimization

max_depth = [int(x) for x in np.linspace(10, 110, num=11)]
max_depth.append(None)

params2 ={

    'n_estimators': [int(x) for x in np.linspace(start=200, stop=2000, num=10)
    'max_features': ['auto', 'sqrt'],
    'max_depth': max_depth,
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}
```

In [47]:
```
rfc = RandomForestClassifier(random_state=42)

rfcs = RandomizedSearchCV(estimator=rfc, param_distributions=params2, n_iter=1
```

In [48]:
```
rfcs.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

```
D:\User\Yamuna\Lib\site-packages\sklearn\model_selection\_validation.py:425:
FitFailedWarning:
205 fits failed out of a total of 500.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting erro
r_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
---
174 fits failed with the following error:
Traceback (most recent call last):
  File "D:\User\Yamuna\Lib\site-packages\sklearn\model_selection\_validation.
py", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "D:\User\Yamuna\Lib\site-packages\sklearn\base.py", line 1144, in wrap
per
    estimator._validate_params()
  File "D:\User\Yamuna\Lib\site-packages\sklearn\base.py", line 637, in _vali
date_params
    validate_parameter_constraints(
  File "D:\User\Yamuna\Lib\site-packages\sklearn\utils\_param_validation.py",
line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' par
ameter of RandomForestClassifier must be an int in the range [1, inf), a floa
t in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' i
nstead.

--------------------------------------------------------------------------------
---
31 fits failed with the following error:
Traceback (most recent call last):
  File "D:\User\Yamuna\Lib\site-packages\sklearn\model_selection\_validation.
py", line 732, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "D:\User\Yamuna\Lib\site-packages\sklearn\base.py", line 1144, in wrap
per
    estimator._validate_params()
  File "D:\User\Yamuna\Lib\site-packages\sklearn\base.py", line 637, in _vali
date_params
    validate_parameter_constraints(
  File "D:\User\Yamuna\Lib\site-packages\sklearn\utils\_param_validation.py",
line 95, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' par
ameter of RandomForestClassifier must be an int in the range [1, inf), a floa
t in the range (0.0, 1.0], a str among {'sqrt', 'log2'} or None. Got 'auto' i
nstead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
D:\User\Yamuna\Lib\site-packages\sklearn\model_selection\_search.py:976: User
Warning: One or more of the test scores are non-finite: [0.82513843 0.8157253
6 0.82059801        nan 0.82990033 0.81572536
 0.82059801 0.80177187        nan        nan        nan 0.80642303
 0.8158361  0.82059801 0.82535991        nan        nan 0.8158361
```

```
        nan        nan 0.8158361  0.81118494 0.82513843        nan
 0.81118494 0.81129568        nan 0.82502769        nan 0.82502769
 0.81572536 0.8158361  0.80642303 0.82990033 0.82990033 0.82535991
 0.82502769        nan        nan 0.82037652        nan 0.8345515
 0.82990033 0.8345515         nan 0.8158361         nan 0.80631229
 0.81572536 0.82048726        nan        nan        nan        nan
 0.82059801 0.81572536        nan        nan        nan 0.82978959
 0.82524917 0.82513843        nan        nan 0.81572536        nan
        nan        nan 0.81118494        nan 0.82048726 0.81572536
 0.82502769        nan        nan 0.82059801 0.81572536 0.81572536
 0.82502769        nan        nan        nan        nan        nan
 0.81572536 0.81118494 0.82513843 0.82990033 0.82990033 0.82037652
 0.82990033 0.81572536 0.82037652        nan 0.81572536        nan
        nan        nan 0.81572536        nan]
  warnings.warn(
```

Out[48]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [49]:

```
rfcs.best_estimator_
```

Out[49]:

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=60, min_samples_leaf=2, min_samples_split=1
0,
                       n_estimators=600, random_state=42)
```

In [50]:

```
y_pred2 = rfcs.predict(X_test)
print(accuracy_score(y_test,y_pred2))
```

```
0.8351648351648352

D:\User\Yamuna\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not
have valid feature names, but RandomForestClassifier was fitted with feature
names
  warnings.warn(
```

I achieved accuracy 83% approx with Random Forest Classifier Model. There is no improvement after Hyperparameter Optimization.

XGBoost

In [51]:

```
xgb = XGBClassifier(random_state = 42)
xgb.fit(X_train,y_train)
y_pred3 = xgb.predict(X_test)
```

In [52]:
```
print(accuracy_score(y_test,y_pred3))
```

```
0.8241758241758241
```

I achieved accuracy 83% approx with XGBoost Classifier Model.

CatBoost

In [53]:
```
model4 = CatBoostClassifier(random_state=42)
```

In [54]:
```
model4.fit(X_train,y_train)
y_pred4 = model4.predict(X_test)
```

```
52:	learn: 0.5164422	total: 229ms	remaining: 4.09s
53:	learn: 0.5141062	total: 231ms	remaining: 4.04s
54:	learn: 0.5117230	total: 234ms	remaining: 4.02s
55:	learn: 0.5094912	total: 236ms	remaining: 3.98s
56:	learn: 0.5079803	total: 237ms	remaining: 3.92s
57:	learn: 0.5056247	total: 239ms	remaining: 3.87s
58:	learn: 0.5034384	total: 240ms	remaining: 3.83s
59:	learn: 0.5012510	total: 242ms	remaining: 3.79s
60:	learn: 0.4984309	total: 244ms	remaining: 3.75s
61:	learn: 0.4955346	total: 245ms	remaining: 3.71s
62:	learn: 0.4930701	total: 247ms	remaining: 3.67s
63:	learn: 0.4908856	total: 249ms	remaining: 3.64s
64:	learn: 0.4887024	total: 250ms	remaining: 3.6s
65:	learn: 0.4867662	total: 252ms	remaining: 3.57s
66:	learn: 0.4846121	total: 255ms	remaining: 3.55s
67:	learn: 0.4821562	total: 257ms	remaining: 3.52s
68:	learn: 0.4800911	total: 259ms	remaining: 3.49s
69:	learn: 0.4794304	total: 259ms	remaining: 3.45s
70:	learn: 0.4772768	total: 261ms	remaining: 3.42s
```

In [55]:
```
print(accuracy_score(y_test,y_pred4))
```

```
0.8131868131868132
```

I achieved accuracy 81% approx with CatBoost Classifier Model.

Conclusion

From the above models KNN is giving us the best accuracy which is 89%.