Project Overview

- You are privided a data set of patients that have or do not have a classified heard disease
- create a model that will assist the cardiologist predict is a patient will show signs of this disease based on the features of that individual
- Privide analysis and visualization on the dataset, where is the data set lacking and what can we learn for mthe data set we have
- How accurate is the model and what would you recommend to improve the perfomance of the Model(s)

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Loading Data

In [4]:
```
df = pd.read_csv("resources/heart.csv")
```

Checking for missing data

- appears we have no misisng data in the dat frame
- 303 samples with 14 features including the target
- Target being is a patient will have or not have Heart Disease

In [6]:
```
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
```

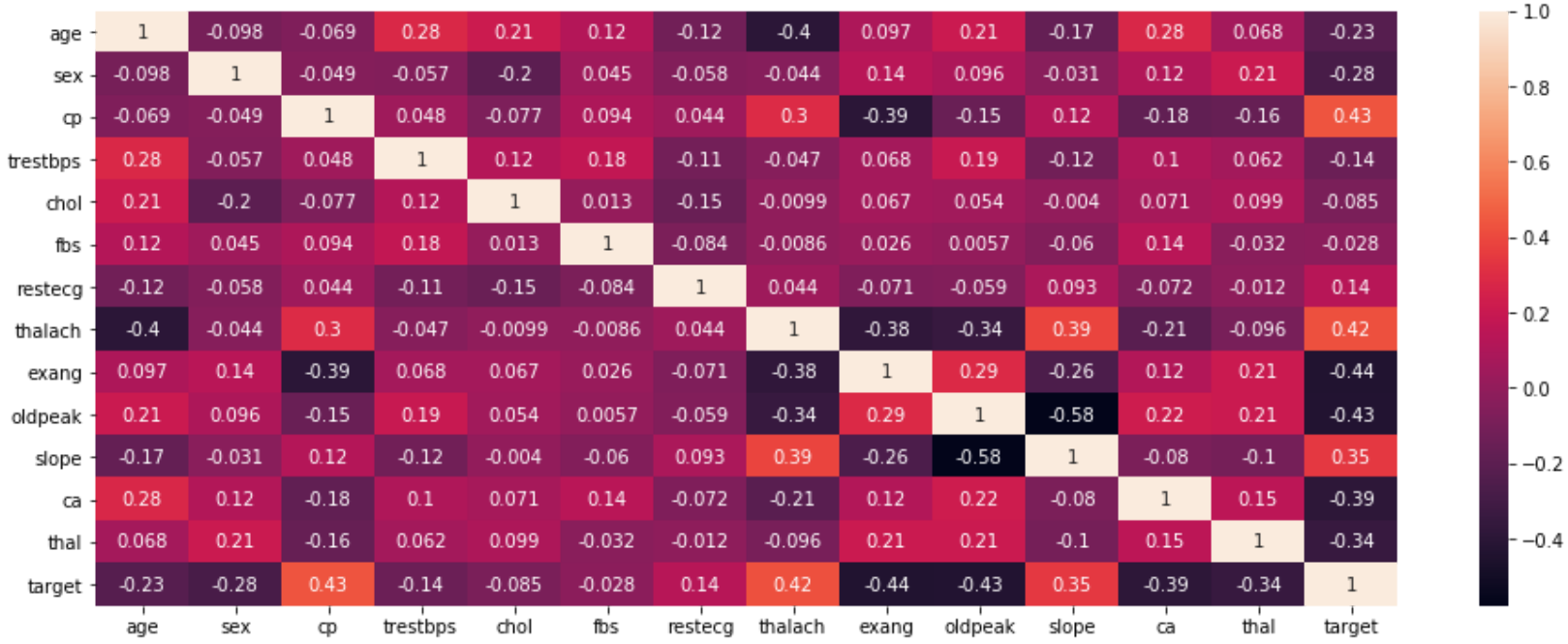
Visualizations

- Since we are not experts in this field we will look at the featues we do know and understand to make a general analysis

Coorelations analysis

In [11]:
```
plt.figure(figsize=(16,6))
sns.heatmap(df.corr(), annot=True)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e85b190>

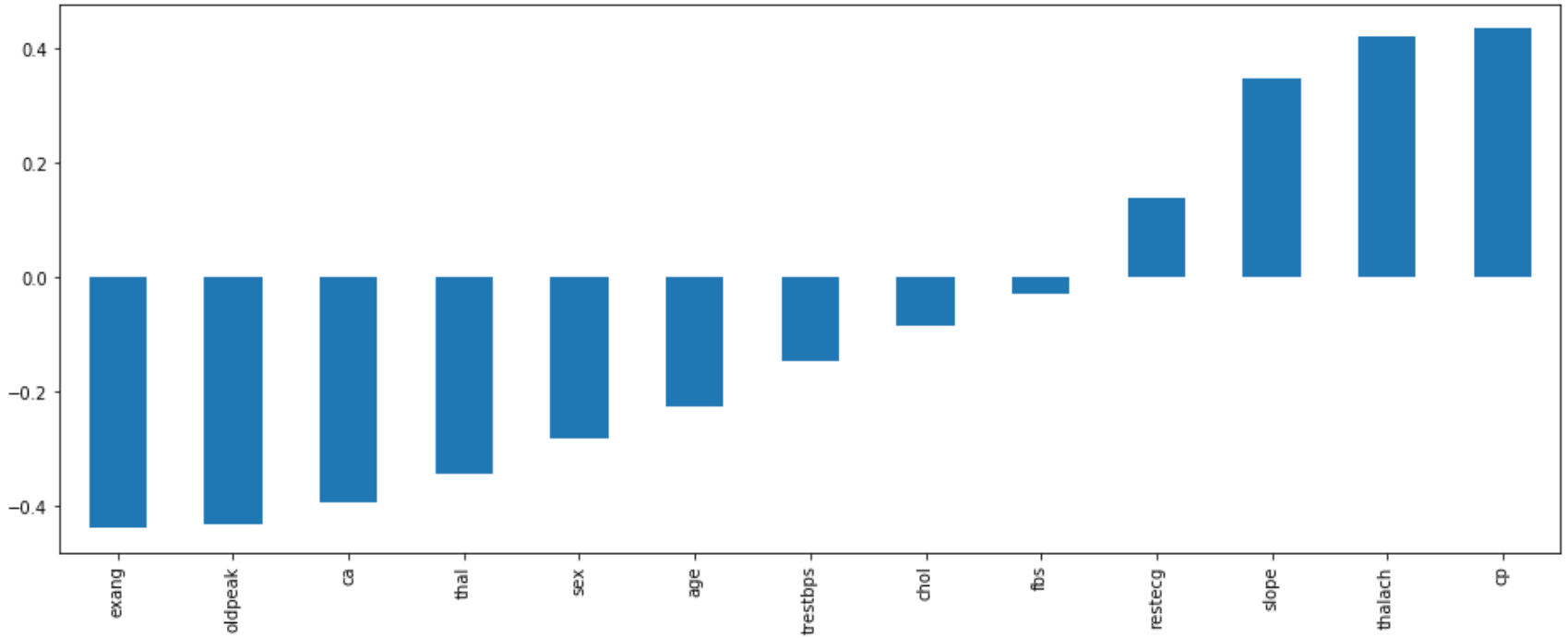


Coorelation to target

- appears the CP is the most coorelates to the target classificaiton

In [17]:
```
df.corrwith(df["target"])[:-1].sort_values().plot(kind = "bar", figsize = (16,6))
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a27ba9fd0>
```



Thalach and age

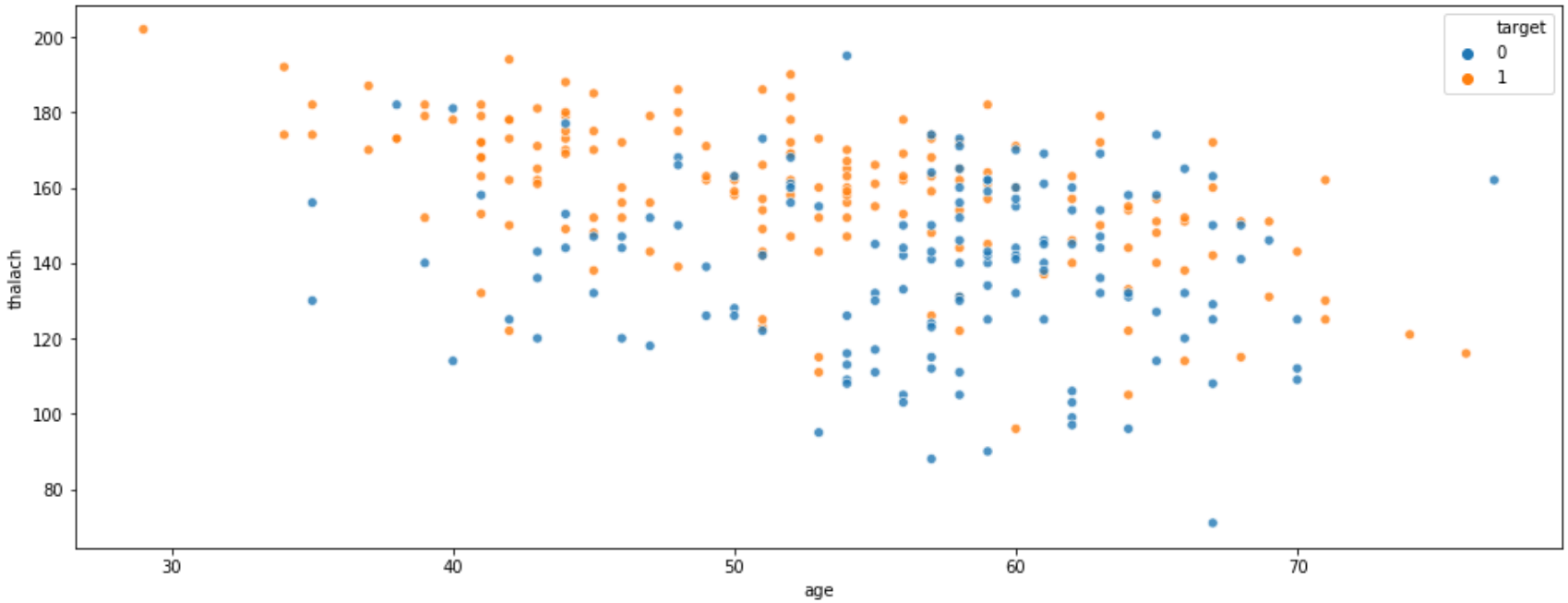
- Appars that an increase in talach value has a higher risk of hear diease
- This is inversely coorelated to the age of the age of the patient

In [ ]:
```

```

In [27]:
```
plt.figure(figsize=(16,6))
sns.scatterplot(x = df["age"], y = df["thalach"], hue=df["target"], alpha=0.8)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1a31a90150>

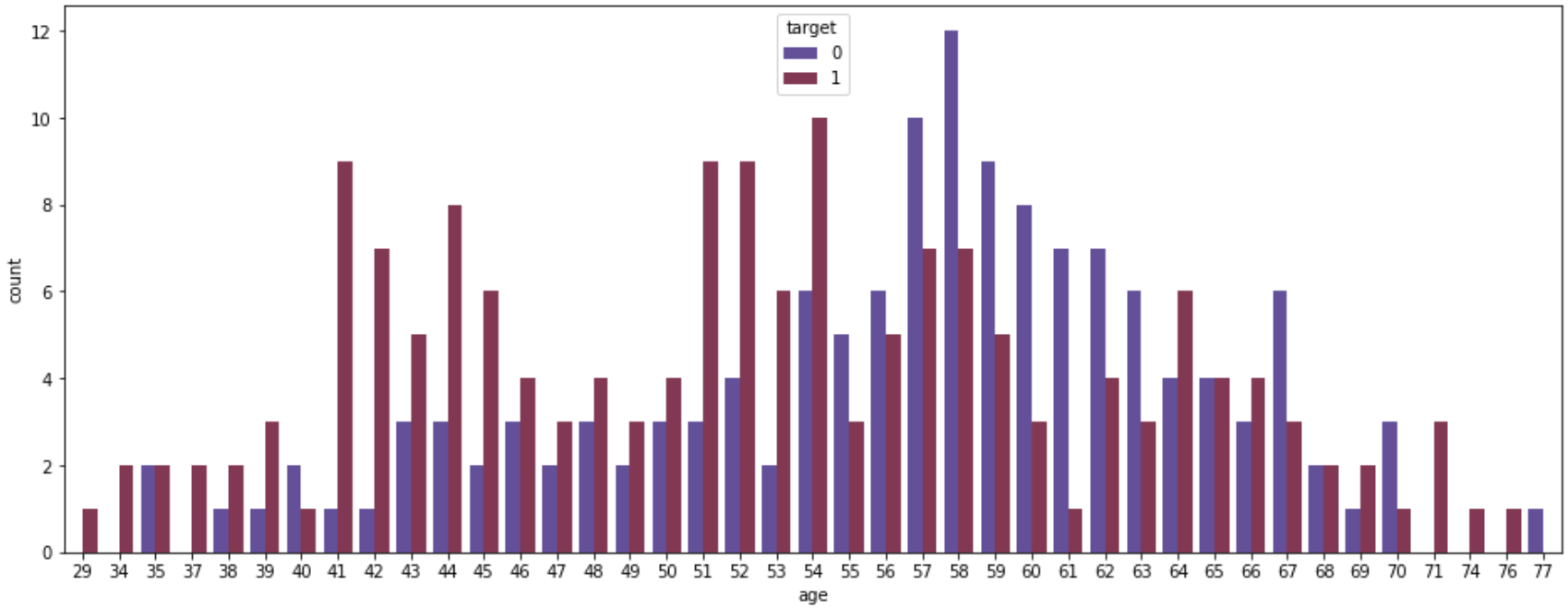


Age range of patient studied

- from the studied population studied ages around 51 to 54 tend to have more heart disease
- Majority of the patients around age 41 and 45 slo demonstrated signs
- the patients at ages 71,74,76, all showed signs of heart diesease

In [36]:
```
plt.figure(figsize=(16,6))
sns.countplot(df["age"], hue= df["target"], palette="twilight")
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x1a32702990>

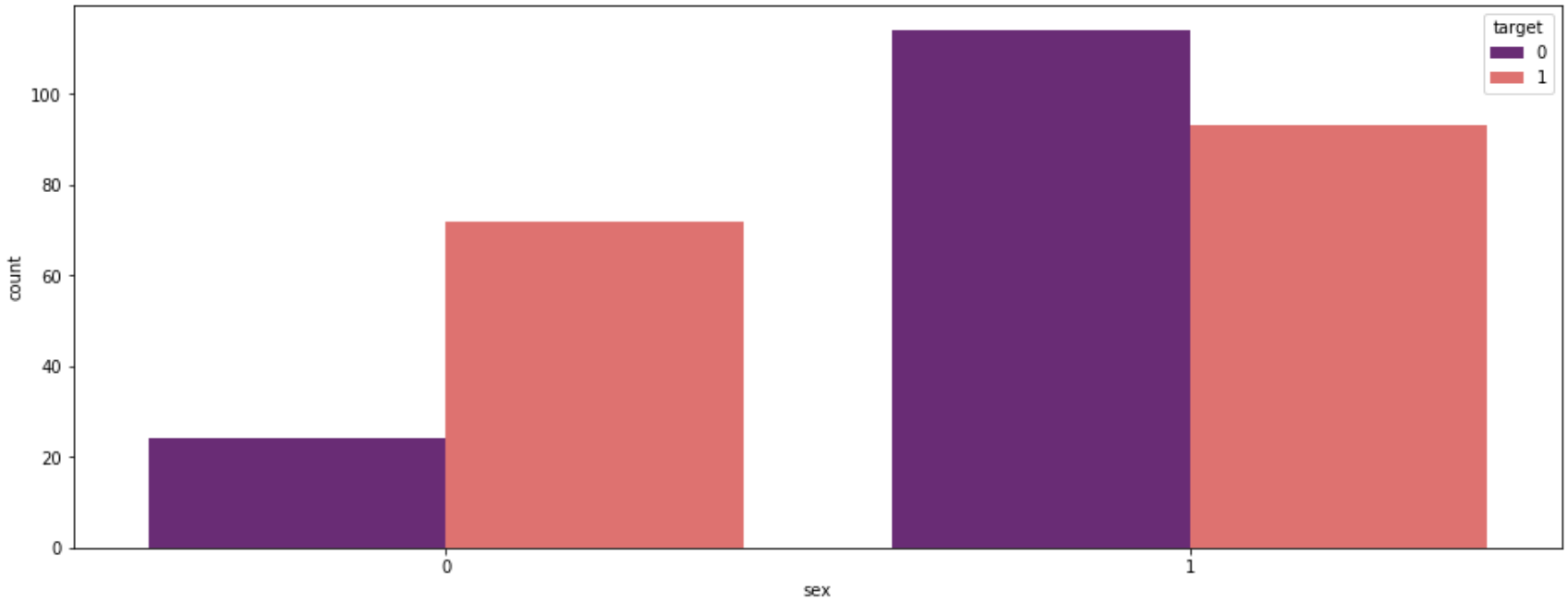


Lookinf at the sex of the patients

- since we have no information on the sex feature we will assuming 0 for female and 1 for male since men are known to suffer more from heart disease

In [41]:
```
plt.figure(figsize=(16,6))
sns.countplot(df["sex"], hue=df["target"], palette="magma")
```

Out[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a291401d0>
```

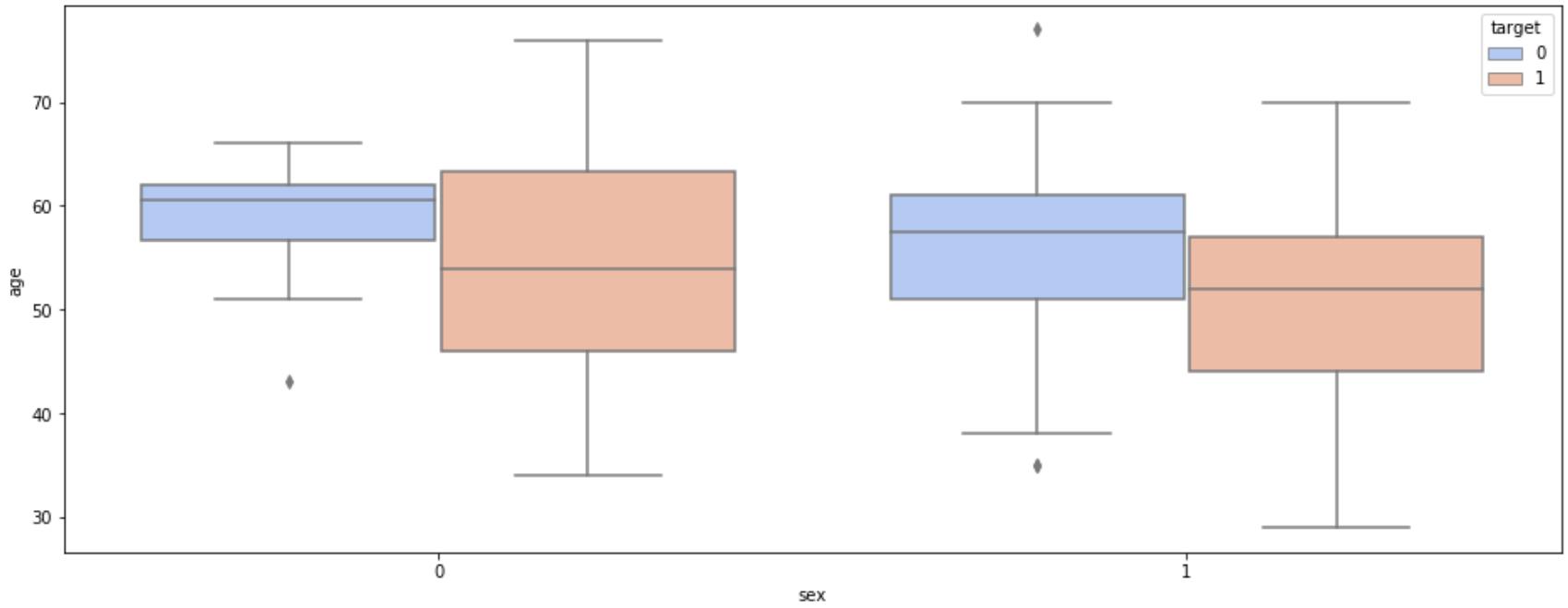


lets look at the average age of the studied group

- Appears the average age of women with heart disease is around 60, where as males seems to begin around age 55, keeping in mind there are alot more men in this group than women

In [45]:
```
plt.figure(figsize=(16,6))
sns.boxplot(x=df["sex"], y= df["age"], palette="coolwarm", hue=df["target"])
```

Out[45]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a3371e610>
```



Lets use ML to classify a patient

- This will allow us to predict if a patient has or does not have heart disease based on the trained features

In [46]:
```
X = df.drop("target",axis=1)
y = df["target"]
```

training Testing Splitting

- Setting up our data to train out model

In [47]:
```
from sklearn.model_selection import train_test_split
```

80% training and 20% testing

In [49]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

lets scale the data to take into account outliers and extreme values

In [50]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [51]:
```
scalar = MinMaxScaler()
```

In [52]:
```
X_train = scalar.fit_transform(X_train)
```

In [54]:
```
X_test = scalar.transform(X_test)
```

Import in Logistic regression model below

In [55]:
```
from sklearn.linear_model import LogisticRegression
```

In [56]:
```
model = LogisticRegression()
```

Model fitting

- Fit model to the training data

In [57]:
```
model.fit(X_train,y_train)
```

Out[57]:
```
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='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [59]:
```
model.classes_
```

Out[59]:
```
array([0, 1])
```

making predictions

- will use our model ot make prediction using data its never seen (test data)

In [60]:
```
predictions= model.predict(X_test)
```

lets compare predicitons

- Wil compare to the y test labels to see how well the model performed
- model is 85% accurate in prediciton if a patient has heart disease or not

In [61]:
```
from sklearn.metrics import confusion_matrix, classification_report, explained_variance_score
```

In [62]:
```
print(classification_report(predictions, y_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.83      0.85        30
           1       0.84      0.87      0.86        31

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```

In [63]:
```
print(confusion_matrix(predictions, y_test))
```

```
[[25  5]
 [ 4 27]]
```

In [64]:
```
print(explained_variance_score(predictions, y_test))
```

```
0.410752688172043
```

Using Deep Learning

- We will try to improve the prediction of the model using a Deep artificial Network
- Since these networks lean using back propagation this should assist in providing us a better predicitons

In [136]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Need early stopping to prevent overtraining

- will use 45 epoch without improvment to initiate the stop

In [161]:
```
stop=EarlyStopping(monitor="val_loss", mode="min", patience=30)
```

Architecting model

In [162]:
```
model = Sequential()
model.add(Dense(units = 320, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(units = 300, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(units = 280, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(units = 160, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(units = 100, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(units = 100, activation = "relu"))
model.add(Dropout(0.25))
model.add(Dense(units = 150, activation = "relu"))
model.add(Dense(units = 1, activation = "sigmoid"))
model.compile(optimizer = "adam", loss ="binary_crossentropy", metrics = ["accuracy"])
```

In [163]:
```
X = df.drop("target",axis=1).values
y = df["target"].values
```

In [164]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

scaling Data

In [165]:
```
scalar = MinMaxScaler()
```

In [166]:
```
X_train = scalar.fit_transform(X_train)
```

In [167]:
```
X_test = scalar.transform(X_test)
```

In [168]:
```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=200, callbacks= [stop])
```

```
Train on 242 samples, validate on 61 samples
Epoch 1/200
242/242 [==============================] - 3s 11ms/sample - loss: 0.6935 - accuracy: 0.4876 - val_loss: 0.68
59 - val_accuracy: 0.8033
Epoch 2/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.6845 - accuracy: 0.6116 - val_loss: 0.676
7 - val_accuracy: 0.7705
Epoch 3/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.6741 - accuracy: 0.6405 - val_loss: 0.651
9 - val_accuracy: 0.7705
Epoch 4/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.6491 - accuracy: 0.6570 - val_loss: 0.572
0 - val_accuracy: 0.8033
Epoch 5/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.5795 - accuracy: 0.7438 - val_loss: 0.471
1 - val_accuracy: 0.8197
Epoch 6/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.5323 - accuracy: 0.7727 - val_loss: 0.451
7 - val_accuracy: 0.8361
Epoch 7/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4483 - accuracy: 0.8140 - val_loss: 0.419
4 - val_accuracy: 0.8361
Epoch 8/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4836 - accuracy: 0.7645 - val_loss: 0.404
0 - val_accuracy: 0.8361
Epoch 9/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4781 - accuracy: 0.7975 - val_loss: 0.377
1 - val_accuracy: 0.8525
Epoch 10/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4510 - accuracy: 0.7769 - val_loss: 0.360
1 - val_accuracy: 0.8689
Epoch 11/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4206 - accuracy: 0.8182 - val_loss: 0.387
0 - val_accuracy: 0.8525
Epoch 12/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3856 - accuracy: 0.8595 - val_loss: 0.370
7 - val_accuracy: 0.8525
Epoch 13/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3876 - accuracy: 0.8058 - val_loss: 0.402
6 - val_accuracy: 0.8525
Epoch 14/200
```

```
242/242 [==============================] - 0s 1ms/sample - loss: 0.4034 - accuracy: 0.8388 - val_loss: 0.357
3 - val_accuracy: 0.8525
Epoch 15/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3904 - accuracy: 0.8140 - val_loss: 0.334
7 - val_accuracy: 0.9016
Epoch 16/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4066 - accuracy: 0.8306 - val_loss: 0.375
9 - val_accuracy: 0.8525
Epoch 17/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.4053 - accuracy: 0.8140 - val_loss: 0.329
6 - val_accuracy: 0.8852
Epoch 18/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3624 - accuracy: 0.8430 - val_loss: 0.410
8 - val_accuracy: 0.8525
Epoch 19/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3490 - accuracy: 0.8636 - val_loss: 0.385
8 - val_accuracy: 0.8689
Epoch 20/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3359 - accuracy: 0.8678 - val_loss: 0.376
8 - val_accuracy: 0.8689
Epoch 21/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3729 - accuracy: 0.8347 - val_loss: 0.414
3 - val_accuracy: 0.8361
Epoch 22/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3639 - accuracy: 0.8678 - val_loss: 0.338
0 - val_accuracy: 0.8852
Epoch 23/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3486 - accuracy: 0.8554 - val_loss: 0.345
1 - val_accuracy: 0.9016
Epoch 24/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3311 - accuracy: 0.8760 - val_loss: 0.405
7 - val_accuracy: 0.8689
Epoch 25/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3571 - accuracy: 0.8554 - val_loss: 0.393
6 - val_accuracy: 0.8852
Epoch 26/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3775 - accuracy: 0.8719 - val_loss: 0.379
4 - val_accuracy: 0.8689
Epoch 27/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3307 - accuracy: 0.8554 - val_loss: 0.382
6 - val_accuracy: 0.8689
Epoch 28/200
```

```
242/242 [==============================] - 0s 1ms/sample - loss: 0.3392 - accuracy: 0.8636 - val_loss: 0.330
2 - val_accuracy: 0.8852
Epoch 29/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3343 - accuracy: 0.8595 - val_loss: 0.387
3 - val_accuracy: 0.8689
Epoch 30/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3207 - accuracy: 0.8512 - val_loss: 0.424
6 - val_accuracy: 0.8197
Epoch 31/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3278 - accuracy: 0.8554 - val_loss: 0.354
2 - val_accuracy: 0.8689
Epoch 32/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3487 - accuracy: 0.8595 - val_loss: 0.356
0 - val_accuracy: 0.8852
Epoch 33/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3208 - accuracy: 0.8719 - val_loss: 0.355
8 - val_accuracy: 0.8852
Epoch 34/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2960 - accuracy: 0.8884 - val_loss: 0.365
9 - val_accuracy: 0.8689
Epoch 35/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2617 - accuracy: 0.8926 - val_loss: 0.395
2 - val_accuracy: 0.8689
Epoch 36/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3400 - accuracy: 0.8678 - val_loss: 0.388
2 - val_accuracy: 0.8689
Epoch 37/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2723 - accuracy: 0.8926 - val_loss: 0.388
7 - val_accuracy: 0.8689
Epoch 38/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2831 - accuracy: 0.9050 - val_loss: 0.377
2 - val_accuracy: 0.8852
Epoch 39/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2503 - accuracy: 0.8926 - val_loss: 0.447
8 - val_accuracy: 0.8689
Epoch 40/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3240 - accuracy: 0.8595 - val_loss: 0.392
7 - val_accuracy: 0.8689
Epoch 41/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2781 - accuracy: 0.8802 - val_loss: 0.421
3 - val_accuracy: 0.8852
Epoch 42/200
```

```
242/242 [==============================] - 0s 1ms/sample - loss: 0.2767 - accuracy: 0.8802 - val_loss: 0.395
3 - val_accuracy: 0.8852
Epoch 43/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2634 - accuracy: 0.8802 - val_loss: 0.394
4 - val_accuracy: 0.8852
Epoch 44/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.3206 - accuracy: 0.8760 - val_loss: 0.459
5 - val_accuracy: 0.8361
Epoch 45/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2502 - accuracy: 0.8843 - val_loss: 0.396
0 - val_accuracy: 0.8689
Epoch 46/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2867 - accuracy: 0.8926 - val_loss: 0.421
3 - val_accuracy: 0.8689
Epoch 47/200
242/242 [==============================] - 0s 1ms/sample - loss: 0.2535 - accuracy: 0.8926 - val_loss: 0.372
1 - val_accuracy: 0.8852
```

Out[168]: <tensorflow.python.keras.callbacks.History at 0x1a6ee1ec90>

Model Summary and predicitons

In [169]:
```
model.summary()
```

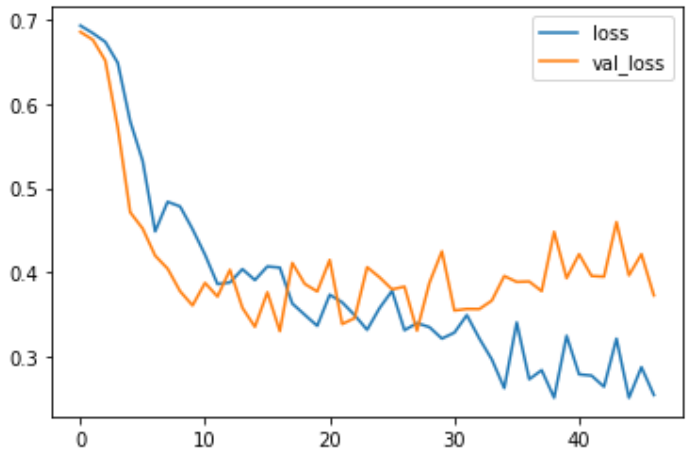
```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_56 (Dense)             multiple                  4480
_________________________________________________________________
dropout_10 (Dropout)         multiple                  0
_________________________________________________________________
dense_57 (Dense)             multiple                  96300
_________________________________________________________________
dropout_11 (Dropout)         multiple                  0
_________________________________________________________________
dense_58 (Dense)             multiple                  84280
_________________________________________________________________
dropout_12 (Dropout)         multiple                  0
_________________________________________________________________
dense_59 (Dense)             multiple                  44960
_________________________________________________________________
dropout_13 (Dropout)         multiple                  0
_________________________________________________________________
dense_60 (Dense)             multiple                  16100
_________________________________________________________________
dropout_14 (Dropout)         multiple                  0
_________________________________________________________________
dense_61 (Dense)             multiple                  10100
_________________________________________________________________
dropout_15 (Dropout)         multiple                  0
_________________________________________________________________
dense_62 (Dense)             multiple                  15150
_________________________________________________________________
dense_63 (Dense)             multiple                  151
=================================================================
Total params: 271,521
Trainable params: 271,521
Non-trainable params: 0
_________________________________________________________________
```

Model History

- Model has anot of noice
- This is because we have no feature that is strongly coorelated to the target so that model is actually performing well in this case
- The accuracy for the model is pretty good for thre data we have
- We must rememerb we only have 303 patients in this data set so maybe more data would assist in predicitons
- For now 80% is pretty good

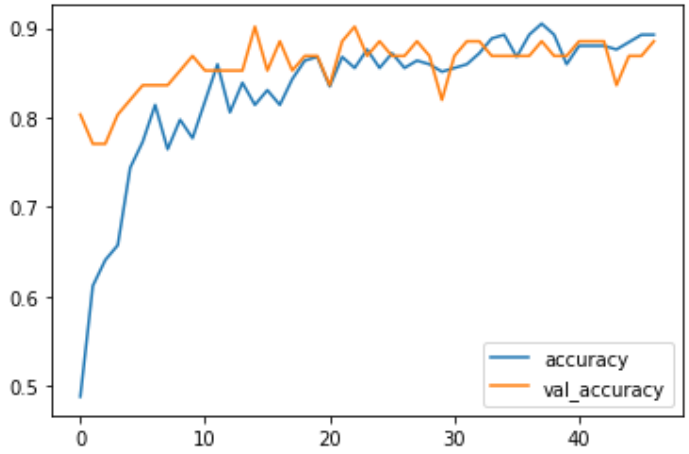
In [170]:
```
pd.DataFrame(model.history.history)[["loss", "val_loss"]].plot()
```

Out[170]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a6e8fdbd0>
```



In [171]:
```
pd.DataFrame(model.history.history)[["accuracy", "val_accuracy"]].plot()
```

Out[171]: <matplotlib.axes._subplots.AxesSubplot at 0x1a6fb1dc10>



Saving model

In [176]:
```
#pd.DataFrame(model.history.history).to_csv("model/model_history.csv")
```

In [175]:
```
#model.save("model/heart_disease_v1.h5")
```

Testing model on random data

- Since we do not have random patients we will test on data we have but not including the target,
- This will give us an idea of how the model will predict on new data that will be coming in

In [191]:
```
from random import randint
random_index = randint(0, len(df))
rand_patient = df.drop("target", axis = 1).iloc[random_index]
```

Preping random patient

- Since the model is trained on scaled data and certain dimansion we will need to convert the random patient to make a prediciton on them

In [192]:
```
X_train.shape
```

Out[192]:
```
(242, 13)
```

In [193]:
```
rand_patient = scalar.transform(rand_patient.values.reshape(1,13))
```

In [194]:
```
rand_patient.shape
```

Out[194]:
```
(1, 13)
```

Model prediciton on random patient

In [195]:
```
model.predict_classes(rand_patient)
```

Out[195]:
```
array([[0]], dtype=int32)
```

Checking True Value

- We can see that the midel is predicting pretty well
- To test simply run the random call cell down repeatly to test predicitons

In [196]:
```
df.iloc[random_index]["target"]
```

Out[196]:
```
0.0
```

In [ ]:
```

```