In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [2]:
```
heart_data=pd.read_csv("C:/Users/hasif/Downloads/heart.csv")
print(heart_data)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
...   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
...     ...  ..   ...     ...
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

In [3]:
```
heart_data.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In [4]:
```
heart_data.tail()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0

In [5]:
```
heart_data.shape
```

Out[5]:
```
(1025, 14)
```

In [6]:
```
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

In [7]:
```
heart_data.isnull().sum()
```

Out[7]:
```
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 [8]:
```
#statistical measures
heart_data.describe()
```

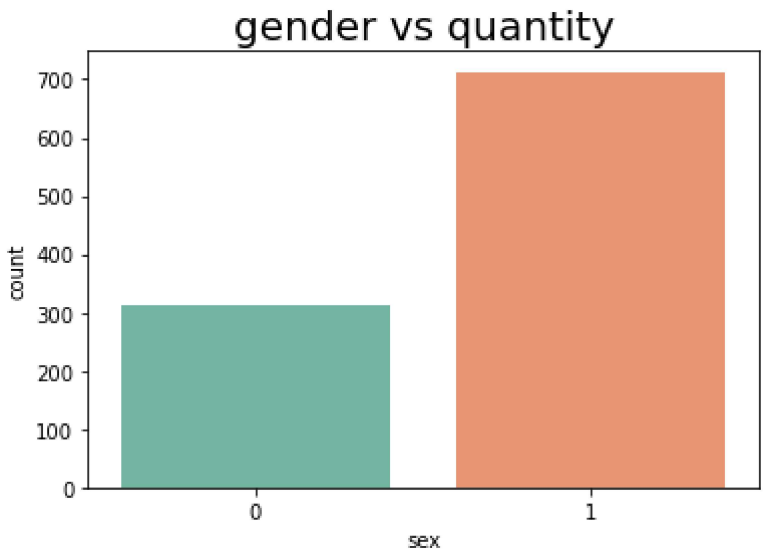
Out[8]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	
count	1025.000000	1025.000000	1025.000000	1025.000000	1025.00000	1025.000000	1025.000000	1025.000000	102
mean	54.434146	0.695610	0.942439	131.611707	246.00000	0.149268	0.529756	149.114146	
std	9.072290	0.460373	1.029641	17.516718	51.59251	0.356527	0.527878	23.005724	
min	29.000000	0.000000	0.000000	94.000000	126.00000	0.000000	0.000000	71.000000	
25%	48.000000	0.000000	0.000000	120.000000	211.00000	0.000000	0.000000	132.000000	
50%	56.000000	1.000000	1.000000	130.000000	240.00000	0.000000	1.000000	152.000000	
75%	61.000000	1.000000	2.000000	140.000000	275.00000	0.000000	1.000000	166.000000	
max	77.000000	1.000000	3.000000	200.000000	564.00000	1.000000	2.000000	202.000000	

In [9]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

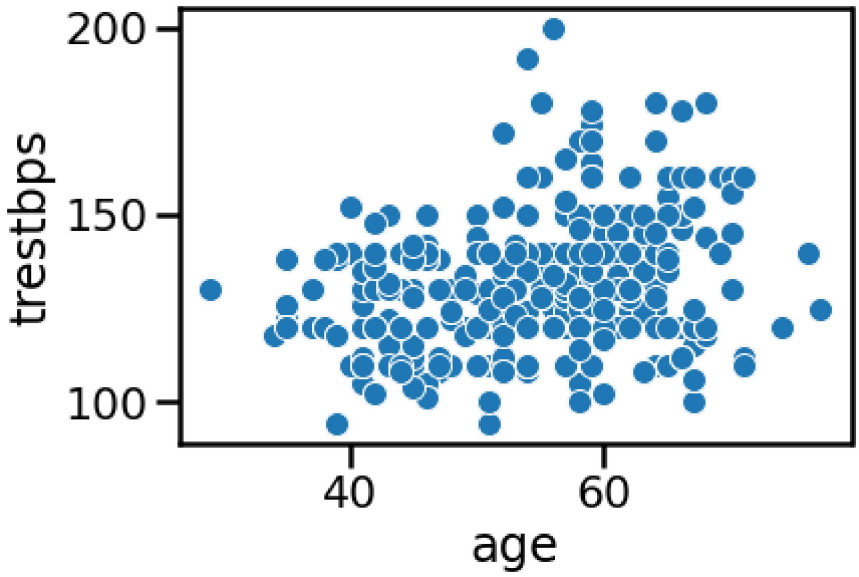
In [10]:
```
sns.countplot(heart_data['sex'],palette='Set2')
plt.title('gender vs quantity',fontsize=20)
plt.show()
```

```
E:\anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



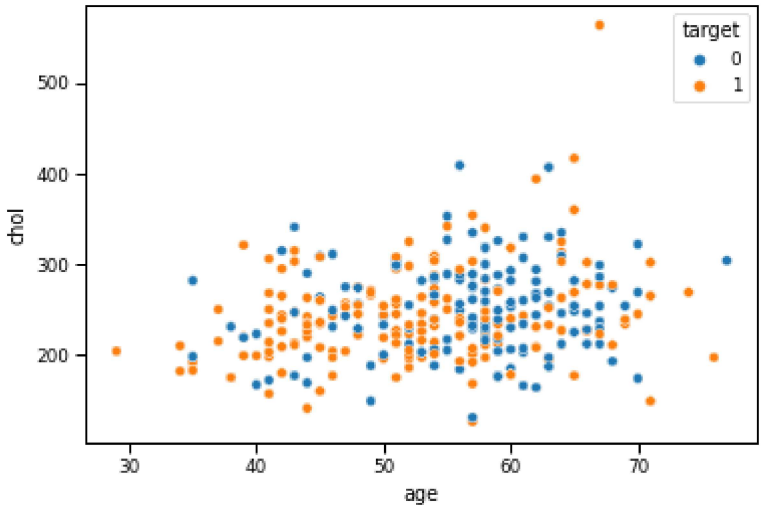
In [11]:
```
sns.set_context("poster")
sns.scatterplot(x='age',y='trestbps',data=heart_data)
```

Out[11]:
```
<AxesSubplot:xlabel='age', ylabel='trestbps'>
```



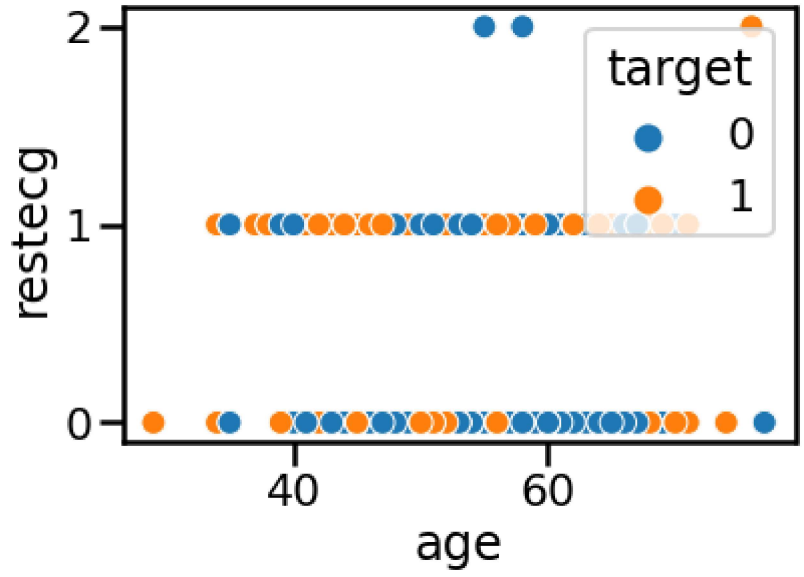
In [12]:
```
sns.set_context("paper")
sns.scatterplot(x='age',y='chol',data=heart_data,hue='target')
```

Out[12]:
```
<AxesSubplot:xlabel='age', ylabel='chol'>
```



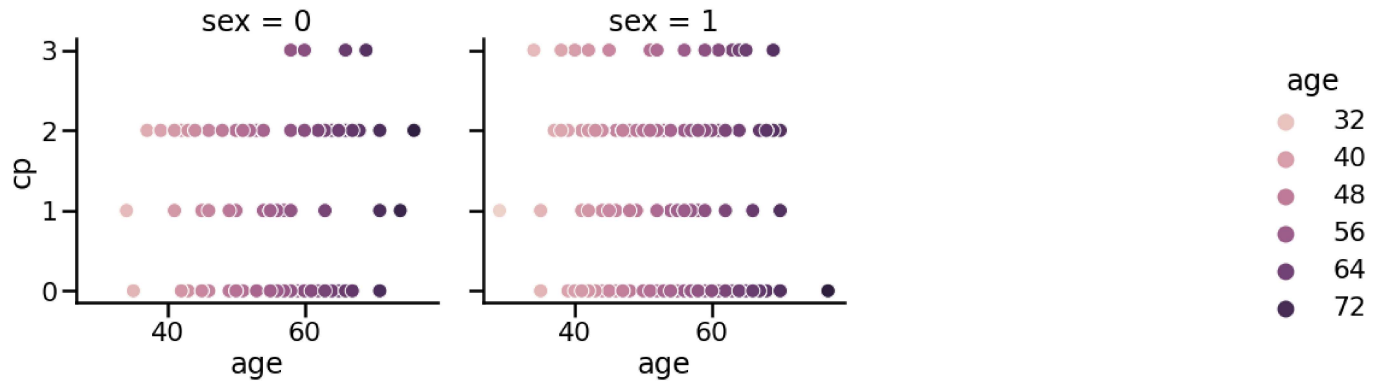
In [13]:
```
sns.set_context("poster")
sns.scatterplot(x='age',y='restecg',data=heart_data,hue='target')
```

Out[13]:
```
<AxesSubplot:xlabel='age', ylabel='restecg'>
```

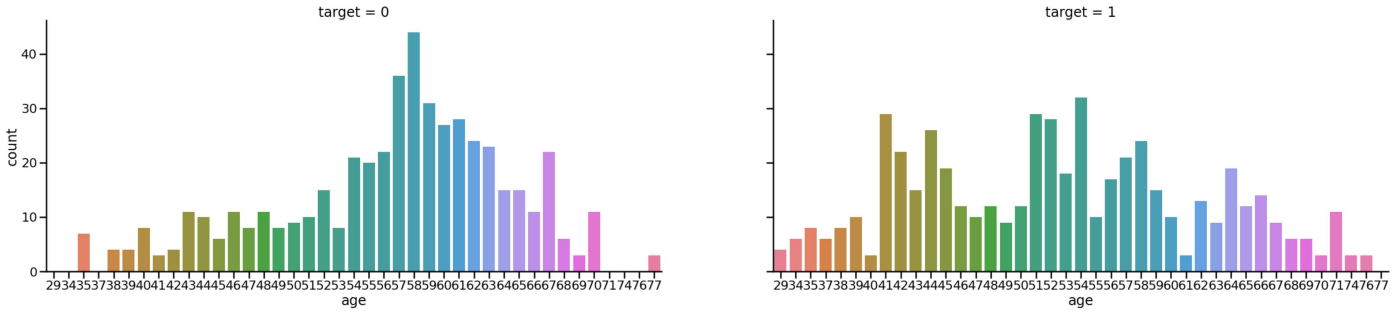


In [14]:
```
sns.set_context('poster')
sns.relplot(data=heart_data,x='age',y='cp',hue='age',col='sex',col_wrap=3)
```

Out[14]:
```
<seaborn.axisgrid.FacetGrid at 0x25014f81f40>
```



In [15]:
```
sns.set_context('poster')
sns.catplot(x='age',col='target',data=heart_data,kind='count',palette='husl')
plt.gcf().set_size_inches(40,10)
plt.show()
```



In [16]:
```
#checking distribution of target
heart_data['target'].value_counts()
#1-->stroke
#0-->healty
```

Out[16]:
```
1    526
0    499
Name: target, dtype: int64
```

In [17]:
```
X=heart_data.drop(columns='target',axis=1)
Y=heart_data['target']
```

In [18]:
```
print(X)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
...   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal
0         2   2     3
1         0   0     3
2         0   0     3
3         2   1     3
4         1   3     2
...     ...  ..   ...
1020      2   0     2
1021      1   1     3
1022      1   1     2
1023      2   0     2
1024      1   1     3

[1025 rows x 13 columns]
```

In [19]:
```
print(Y)
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1020    1
1021    0
1022    0
1023    1
1024    0
Name: target, Length: 1025, dtype: int64
```

In [20]:
```
#splitting the data into training data and test data
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=2)
```

In [21]:
```
X_train.shape
```

Out[21]:
```
(820, 13)
```

In [22]:
```
print(X_test.shape)
```

```
(205, 13)
```

In [23]:
```
X.shape
```

Out[23]:
```
(1025, 13)
```

In [24]:
```
#Model training
#logistic training
model=LogisticRegression()
model.fit(X_train,Y_train)
```

```
E:\anaconda\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs f
ailed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[24]:
```
LogisticRegression()
```

In [25]:
```
#model evaluation
#accuracy score
#on trained data
X_train_prediction=model.predict(X_train)
training_data_accuracy=accuracy_score(X_train_prediction,Y_train)
```

In [26]:
```
print('Accuracy on training data:',training_data_accuracy)
```

```
Accuracy on training data: 0.8524390243902439
```

In [27]:
```
X_test_prediction=model.predict(X_test)
test_data_accuracy=accuracy_score(X_test_prediction,Y_test)
```

In [28]:
```
print('accuracy on test data:',test_data_accuracy)
```

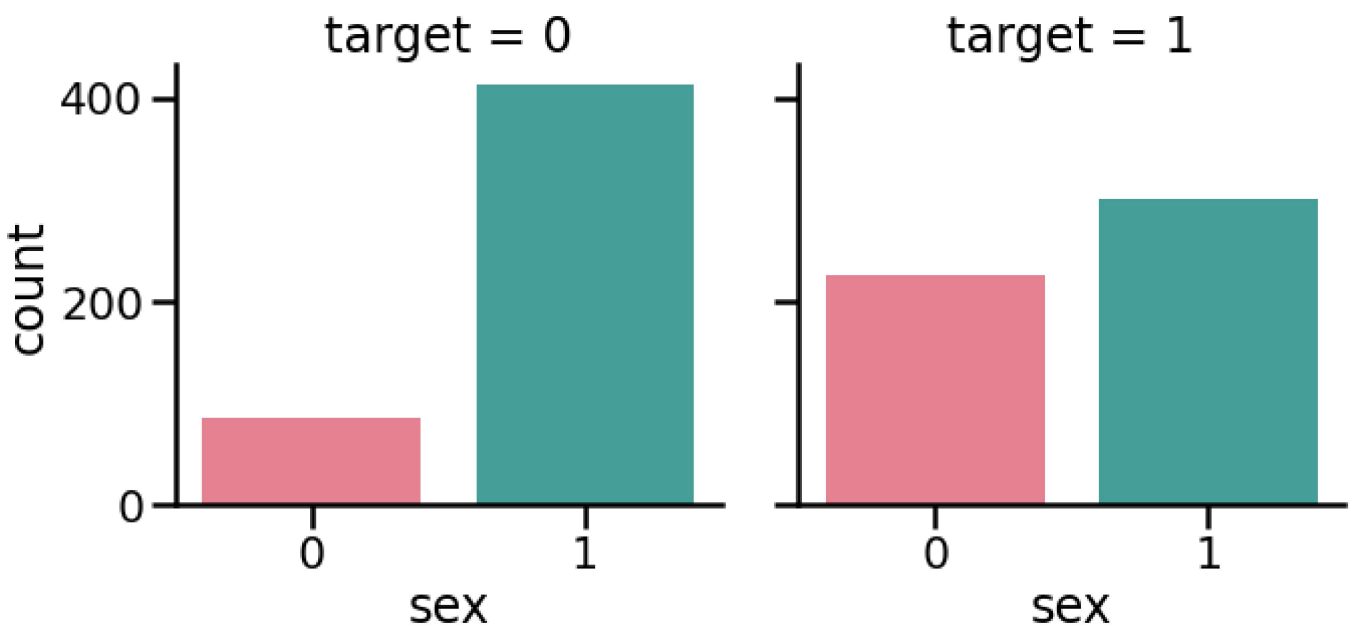
```
accuracy on test data: 0.8048780487804879
```

In [29]:
```
#BUILDING A PREDICTIVE MODEL
input_data=(58,1,0,114,318,0,2,140,0,4.4,0,3,1)
#change input data into numpy array
input_data_as_numpy_array=np.asarray(input_data)

#reshape the numpy array as we are predicting
input_data_reshaped=input_data_as_numpy_array.reshape(1,-1)
prediction=model.predict(input_data_reshaped)
#print(prediction)
if (prediction[0]==0):
    print('the person doesnot have a heart disease')
else:
    print('the person has a heart disease')
```

```
the person doesnot have a heart disease
```

In [30]:
```
sns.catplot(x='sex',col='target',data=heart_data,kind='count',palette='husl')
plt.show()
```



In [ ]: