In

[1]:

In

[2]:

**import**

pandas

**as**

pd

**import**

numpy

**as**

np

**import**

matplotlib

.

pyplot

**as**

plt

**import**

seaborn

**as**

sns

**import**

warnings

warnings

.

filterwarnings

(

'ignore'

)

df

**=**

pd

.

read\_csv

(

'Downloads/Heart\_Disease\_Prediction.csv'

)

In [3]:

df

.

head

()

Out[3]:

**Numbe**

**Chest FBS**

**EKG Max Exercise ST Slope o**

**Age Sex pain BP Cholesterol over results HR angina depression of ST vesse**

**type 120**

**flur**

1. 70 1 4 130 322 0 2 109 0 2.4 2
2. 67 0 3 115 564 0 2 160 0 1.6 2
3. 57 1 2 124 261 0 0 141 0 0.3 1
4. 64 1 4 128 263 0 0 105 1 0.2 2
5. 74 0 2 120 269 0 2 121 1 0.2 1

In [4]:

df

.

isnull

().

sum

()

Out[4]:

Age 0

Sex 0

Chest pain type 0

BP 0

Cholesterol 0

FBS over 120 0

EKG results 0

Max HR 0

Exercise angina 0

ST depression 0

Slope of ST 0

Number of vessels fluro 0

Thallium 0 Heart Disease 0 dtype: int64

[5]:

print

(

df

.

info

())

<class 'pandas.core.frame.DataFrame'> RangeIndex: 270 entries, 0 to 269

Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. Age 270 non-null int64
2. Sex 270 non-null int64
3. Chest pain type 270 non-null int64
4. BP 270 non-null int64
5. Cholesterol 270 non-null int64
6. FBS over 120 270 non-null int64
7. EKG results 270 non-null int64
8. Max HR 270 non-null int64
9. Exercise angina 270 non-null int64
10. ST depression 270 non-null float64
11. Slope of ST 270 non-null int64
12. Number of vessels fluro 270 non-null int64
13. Thallium 270 non-null int64
14. Heart Disease 270 non-null object dtypes: float64(1), int64(12), object(1) memory usage: 29.7+ KB None

In [6]:

plt

.

figure

(

figsize

**=**

(

20

,

10

))

sns

.

heatmap

(

df

.

corr

()

,

annot

**=**

**True**

,

cmap

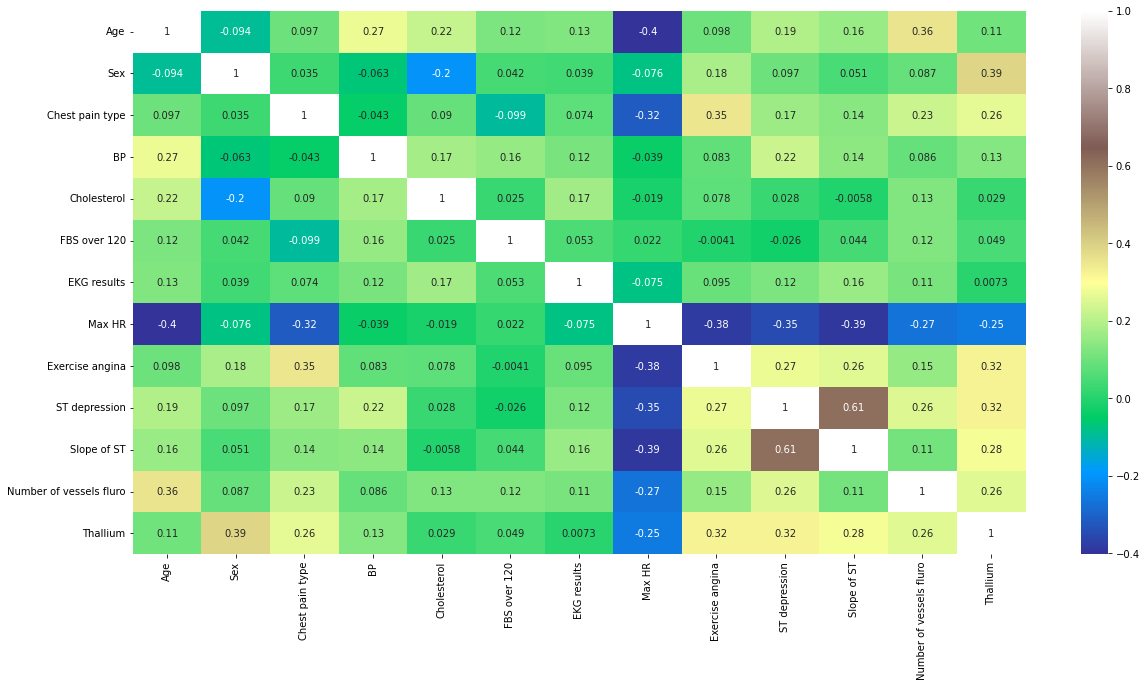
**=**

'terrain'

)

Out[6]:

<AxesSubplot:>



[7]:

sns

.

pairplot

(

data

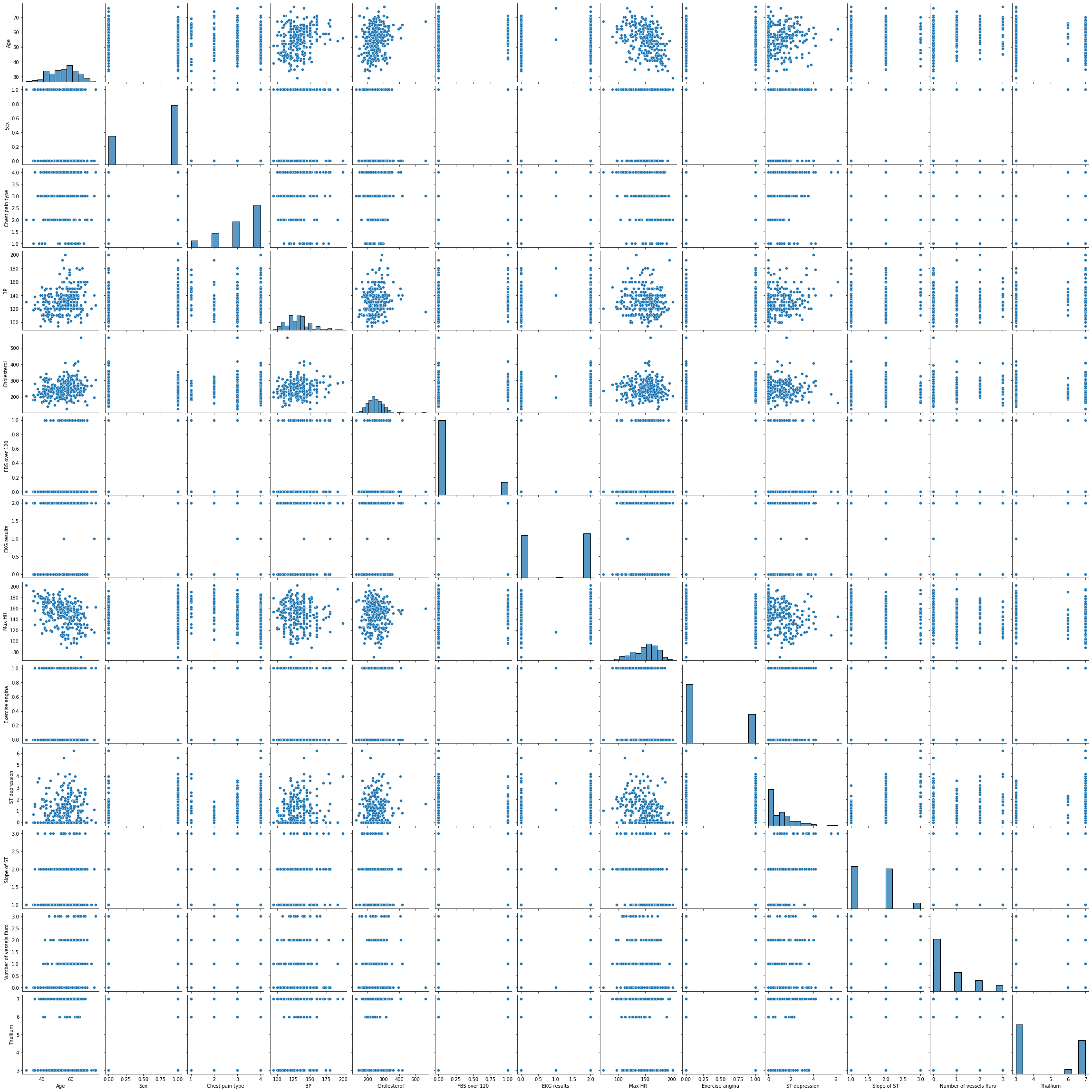
**=**

df

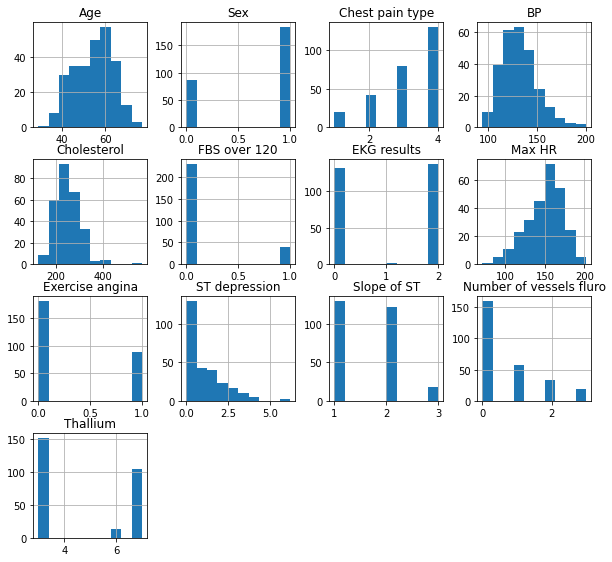
)

Out[7]:

<seaborn.axisgrid.PairGrid at 0x1bb678b6308>



[8]:



df

.

hist

(

figsize

**=**

(

10

,

12

)

,

layout

**=**

(

5

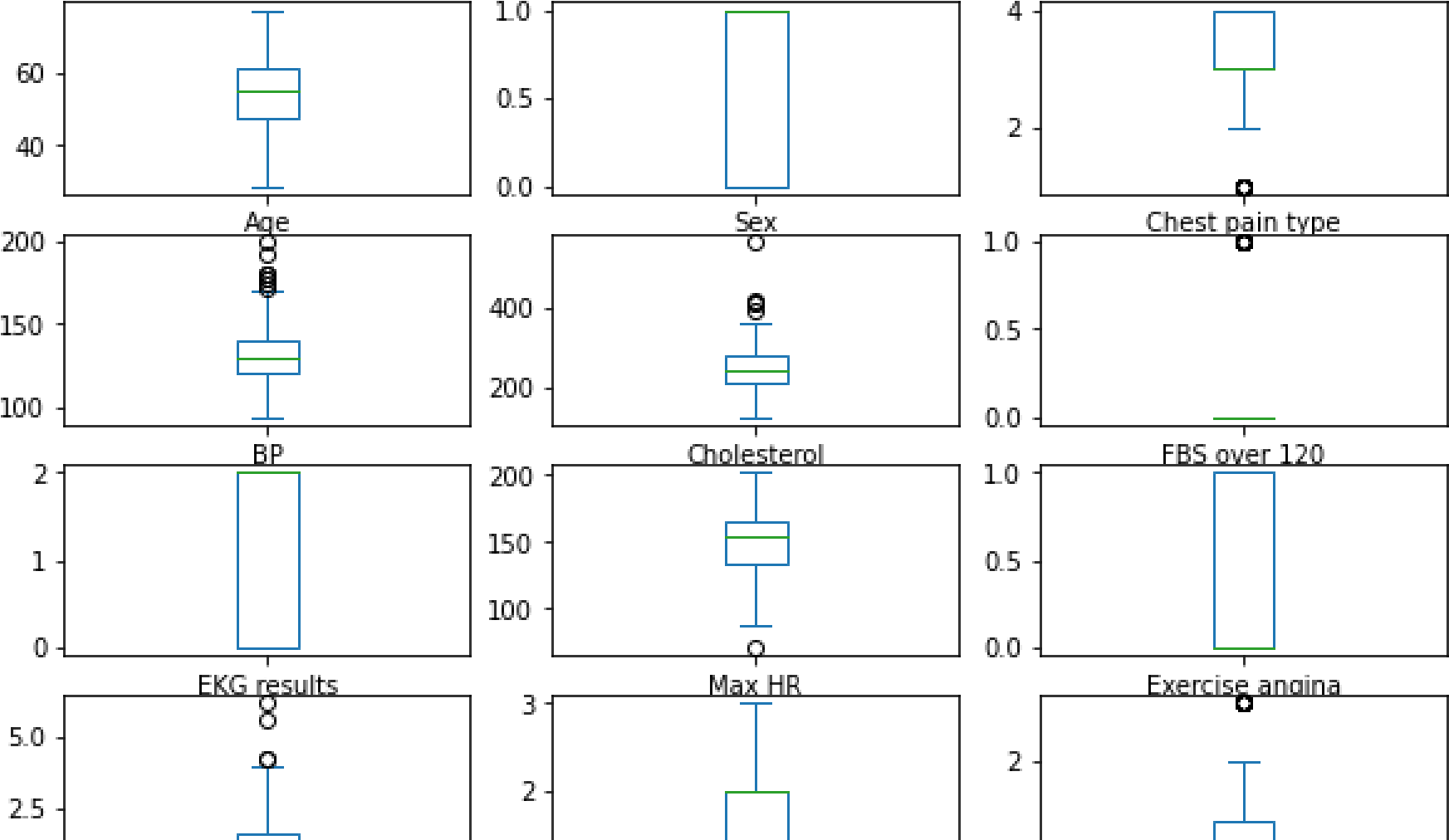
,

4

))

;

[9]:



df

.

plot

(

kind

**=**

'box'

,

subplots

**=**

**True**

,

layout

**=**

(

6

,

3

)

,

figsize

**=**

(

10

,

10

))

plt

.

show

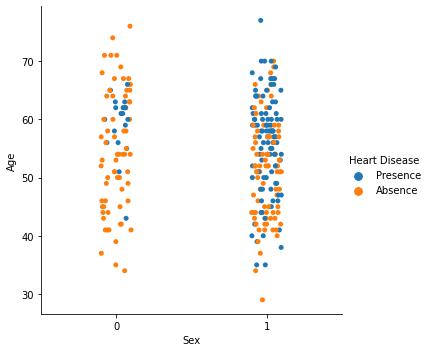
()

In [10]:

sns.catplot(data**=**df, x**=**'Sex', y**=**'Age', hue**=**'Heart Disease', palette**=**'tab10')

Out[10]:

<seaborn.axisgrid.FacetGrid at 0x1bb71a93fc8>



[11]:

sns

.

barplot

(

data

**=**

df

,

x

**=**

'Sex'

,

y

**=**

'Cholesterol'

,

hue

**=**

'Heart Disease'

,

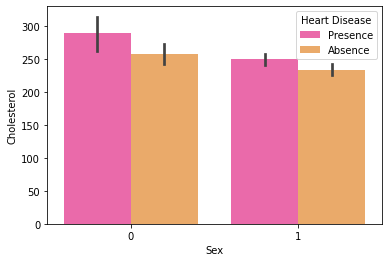
palette

**=**

'spring'

)

Out[11]: <AxesSubplot:xlabel='Sex', ylabel='Cholesterol'>



In [12]:

df

[

'Sex'

].

value\_counts

()

Out[12]:

1 183

0 87

Name: Sex, dtype: int64 In [13]:

df

[

'Chest pain type'

].

value\_counts

()

Out[13]:

4 129

3 79

2 42

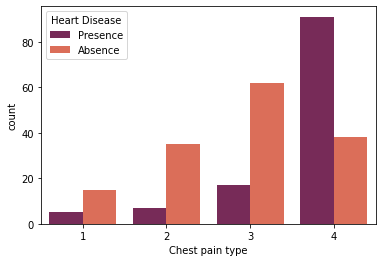
1 20

Name: Chest pain type, dtype: int64

[14]:

sns.countplot(x**=**'Chest pain type', hue**=**'Heart Disease' , data**=**df, palette**=**'rocket')

Out[14]: <AxesSubplot:xlabel='Chest pain type', ylabel='count'>



In [15]:

gen

**=**

pd

.

crosstab

(

df

[

'Sex'

]

,

df

[

'Heart Disease'

])

print

(

gen

)

Heart Disease Absence Presence

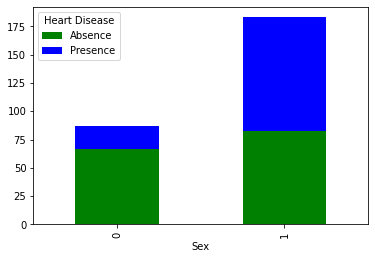
Sex

1. 67 20
2. 83 100

[16]:

gen.plot(kind**=**'bar', stacked**=**'True', color**=**['green','blue'],grid**=False**)

Out[16]: <AxesSubplot:xlabel='Sex'>



In [17]:

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.preprocessing **import** StandardScaler StandardScaler **=** StandardScaler()

columns\_to\_scale**=**['Age', 'EKG results', 'Cholesterol', 'Thallium', 'Number of vessels fluro df[columns\_to\_scale] **=** StandardScaler.fit\_transform(df[columns\_to\_scale]) In [18]:

df

.

head

()

Out[18]:

**Chest FBS**

**EKG Max Exercise ST Slope**

**Age Sex pain BP Cholesterol over results HR angina depression of ST**

**type 120**

1. 1.712094 1 4 130 1.402212 0 0.981664 109 0 2.4 2
2. 1.382140 0 3 115 6.093004 0 0.981664 160 0 1.6 2
3. 0.282294 1 2 124 0.219823 0 -1.026285 141 0 0.3 1
4. 1.052186 1 4 128 0.258589 0 -1.026285 105 1 0.2 2
5. 2.152032 0 2 120 0.374890 0 0.981664 121 1 0.2 1

[19]:

**from** sklearn.model\_selection **import** train\_test\_split **from** sklearn.preprocessing **import** StandardScaler StandardScaler **=** StandardScaler()

columns\_to\_scale**=**['Age', 'EKG results', 'Cholesterol', 'Thallium', 'Number of vessels fluro df[columns\_to\_scale] **=** StandardScaler.fit\_transform(df[columns\_to\_scale]) In [20]:

df

.

head

()

Out[20]:

**Chest FBS**

**EKG Max Exercise ST Slope**

**Age Sex pain BP Cholesterol over results HR angina depression of ST**

**type 120**

1. 1.712094 1 4 130 1.402212 0 0.981664 109 0 2.4 2
2. 1.382140 0 3 115 6.093004 0 0.981664 160 0 1.6 2
3. 0.282294 1 2 124 0.219823 0 -1.026285 141 0 0.3 1
4. 1.052186 1 4 128 0.258589 0 -1.026285 105 1 0.2 2
5. 2.152032 0 2 120 0.374890 0 0.981664 121 1 0.2 1

In

[21]:

x

**=**

df

.

drop

([

'Heart Disease'

]

,

axis

**=**

1

)

y

**=**

df

[

'Heart Disease'

]

In [22]:

x\_train, x\_test, y\_train, y\_test**=**train\_test\_split(x,y,test\_size**=**0.3, random\_state**=**40) In [23]:

x\_train- 2457

print

(

'x\_train-'

,

x\_train

.

size

)

print

(

'x\_test-'

,

x\_test

.

size

)

print

(

'y\_train-'

,

y\_train

.

size

)

print

(

'x\_test-'

,

x\_test

.

size

)

x\_test- 1053 y\_train- 189 x\_test- 1053

[24]:

In

[25]:

**from**

sklearn

.

linear\_model

**import**

LogisticRegression

lr

**=**

LogisticRegression

()

model1

**=**

lr

.

fit

(

x\_train

,

y\_train

)

prediction1

**=**

model1

.

predict

(

x\_test

)

**from**

sklearn

.

metrics

**import**

confusion\_matrix

cm

**=**

confusion\_matrix

(

y\_test

,

prediction1

)

cm

Out[25]:

array([[40, 5],

[ 9, 27]], dtype=int64) In [26]:

sns

.

heatmap

(

cm

,

annot

**=**

**True**

,

cmap

**=**

'BuPu'

)

Out[26]:

<AxesSubplot:>



In

[27]:

Testing Accuracy: 0.9382716049382716

TP

**=**

cm

[

0

][

0

]

TN

**=**

cm

[

1

][

1

]

FN

**=**

cm

[

1

][

0

]

FP

**=**

cm

[

0

][

1

]

print

(

'Testing Accuracy:'

, (

TP

**+**

TN

**+**

FN

)

**/**

(

TP

**+**

TN

**+**

FN

**+**

FP

))

In

[28]:

**from**

sklearn

.

metrics

**import**

accuracy\_score

accuracy\_score

(

y\_test

,

prediction1

)

l

**=**

accuracy\_score

(

y\_test

,

prediction1

)

[29]:

**from**

sklearn

.

metrics

**import**

classification\_report

print

(

classification\_report

(

y\_test

,

prediction1

))

precision recall f1-score support

Absence 0.82 0.89 0.85 45 Presence 0.84 0.75 0.79 36

accuracy 0.83 81 macro avg 0.83 0.82 0.82 81 weighted avg 0.83 0.83 0.83 81 In [30]:

In

[31]:

**import**

pandas

**as**

pd

**from**

sklearn

**import**

neighbors

,

metrics

**from**

sklearn

.

model\_selection

**import**

train\_test\_split

**from**

sklearn

.

neighbors

**import**

KNeighborsClassifier

**import**

numpy

**as**

np

**import**

pickle

**from**

sklearn

.

ensemble

**import**

RandomForestClassifier

**import**

pandas

**as**

pd

**from**

sklearn

.

tree

**import**

DecisionTreeClassifier

**import**

seaborn

**as**

sns

**import**

matplotlib

.

pyplot

**as**

plt

**from**

sklearn

.

metrics

**import**

accuracy\_score

In [32]:

dataset

**=**

pd

.

read\_csv

(

"Downloads/Heart\_Disease\_Prediction.csv"

)

In [33]:

KX

**=**

dataset

[[

'Age'

,

'Sex'

,

'Chest pain type'

,

'BP'

,

'Cholesterol'

,

'FBS over 120'

,

'EKG results'

In [34]:

KY

**=**

dataset

[[

'Heart Disease'

]].

values

[35]:

KX

Out[35]:

array([[70., 1., 4., ..., 2., 3., 3.],

[67., 0., 3., ..., 2., 0., 7.], [57., 1., 2., ..., 1., 0., 7.], ...,

[56., 0., 2., ..., 2., 0., 3.],

[57., 1., 4., ..., 2., 0., 6.],

[67., 1., 4., ..., 2., 3., 3.]]) In [36]:

KY

**=**

KY

.

flatten

()

print

(

KY

)

['Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'

'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'

'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'

'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'

'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'

'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'

'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'

'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence'

'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'

'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Presence'

'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence'

'Presence' 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence'

'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'

'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence'

'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence' 'Presence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'

'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'

'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence' 'Absence'

'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'

'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'

'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence'

'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'

'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'

'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'

'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Presence'

'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Presence']

[37]:

KX\_train , KX\_test , KY\_train , KY\_test **=** train\_test\_split(KX,KY,test\_size**=**0.2,random\_state In [38]:

0.6111111111111112

knn

**=**

KNeighborsClassifier

(

n\_neighbors

**=**

20

)

knn

.

fit

(

KX\_train

,

KY\_train

)

print

(

knn

.

score

(

KX\_test

,

KY\_test

))

In [39]:

pickle

.

dump

(

knn

,

open

(

'heart\_knn\_model.sav'

,

'wb'

))

In [40]:

predict\_knn

**=**

knn

.

predict

(

KX\_test

)

accuracy\_knn

**=**

metrics

.

accuracy\_score

(

KY\_test

,

predict\_knn

)

In [41]:

predict\_knn

Out[41]:

array(['Absence', 'Absence', 'Absence', 'Presence', 'Presence', 'Absence',

'Presence', 'Absence', 'Absence', 'Absence', 'Presence', 'Absence',

'Absence', 'Presence', 'Presence', 'Absence', 'Absence', 'Absence', 'Presence', 'Presence', 'Presence', 'Presence', 'Absence',

'Absence', 'Absence', 'Presence', 'Presence', 'Absence', 'Absence',

'Presence', 'Absence', 'Presence', 'Presence', 'Absence',

'Absence', 'Absence', 'Absence', 'Absence', 'Absence', 'Presence',

'Absence', 'Presence', 'Absence', 'Absence', 'Absence', 'Absence',

'Presence', 'Absence', 'Absence', 'Presence', 'Absence', 'Absence',

'Absence', 'Absence'], dtype=object) In [42]:

accuracy\_knn

Out[42]:

0.6111111111111112

In [43]:

k

**=**

accuracy\_knn

[45]:

In

[46]:

**import**

csv

**import**

pandas

**as**

pd

**import**

numpy

**as**

np

**from**

sklearn

.

naive\_bayes

**import**

GaussianNB

**from**

sklearn

.

model\_selection

**import**

train\_test\_split

**from**

sklearn

**import**

metrics

**from**

sklearn

.

metrics

**import**

confusion\_matrix

,

f1\_score

,

roc\_curve

,

auc

**import**

matplotlib

.

pyplot

**as**

plt

**from**

itertools

**import**

cycle

**from**

scipy

**import**

interp

df

**=**

pd

.

read\_csv

(

'Downloads/Heart\_Disease\_Prediction.csv'

,

header

**=**

**None**

)

In [47]:

training\_x

**=**

df

.

iloc

[

1

:

df

.

shape

[

0

]

,

0

:

13

]

In [48]:

training\_y

**=**

df

.

iloc

[

1

:

df

.

shape

[

0

]

,

13

:

14

]

In [49]:

nx

**=**

np

.

array

(

training\_x

)

ny

**=**

np

.

array

(

training\_y

)

In [52]:

**for** z **in** range(5):

print("\nTest Train Split no. ",z**+**1,"\n")

nx\_train,nx\_test,ny\_train,ny\_test **=** train\_test\_split(nx,ny,test\_size**=**0.25,random\_state**=**

*# Gaussian function of sklearn*  gnb **=** GaussianNB()

gnb.fit(nx\_train, ny\_train.ravel()) ny\_pred **=** gnb.predict(nx\_test)

Test Train Split no. 1

Test Train Split no. 2

Test Train Split no. 3

Test Train Split no. 4

Test Train Split no. 5

[61]: print("\n Naive Bayes model accuracy(in %):", metrics.accuracy\_score(ny\_test, ny\_pred))

Naive Bayes model accuracy(in %): 0.7794117647058824 In [62]:

n

**=**

metrics

.

accuracy\_score

(

ny\_test

,

ny\_pred

)

In

[64]:

In

[65]:

**import**

pandas

**as**

pd

**from**

sklearn

**import**

neighbors

,

metrics

**from**

sklearn

.

model\_selection

**import**

train\_test\_split

**from**

sklearn

.

neighbors

**import**

KNeighborsClassifier

**import**

numpy

**as**

np

**import**

pickle

**from**

sklearn

.

ensemble

**import**

RandomForestClassifier

**import**

pandas

**as**

pd

**from**

sklearn

.

tree

**import**

DecisionTreeClassifier

**import**

seaborn

**as**

sns

**import**

matplotlib

.

pyplot

**as**

plt

**from**

sklearn

.

metrics

**import**

accuracy\_score

In [67]:

dataset

**=**

pd

.

read\_csv

(

"Downloads/Heart\_Disease\_Prediction.csv"

)

In [69]:

DX

**=**

dataset

[[

'Age'

,

'Sex'

,

'Chest pain type'

,

'BP'

,

'Cholesterol'

,

'FBS over 120'

,

'EKG results'

In [70]:

dy

**=**

dataset

[[

'Heart Disease'

]].

values

In [71]:

DX

Out[71]:

array([[70., 1., 4., ..., 2., 3., 3.],

[67., 0., 3., ..., 2., 0., 7.],

[57., 1., 2., ..., 1., 0., 7.], ...,

[56., 0., 2., ..., 2., 0., 3.],

[57., 1., 4., ..., 2., 0., 6.], [67., 1., 4., ..., 2., 3., 3.]]) [72]:

dy

**=**

dy

.

flatten

()

print

(

dy

)

['Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'

'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'

'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'

'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'

'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence'

'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'

'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'

'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence'

'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Presence'

'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Presence'

'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence'

'Presence' 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence'

'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'

'Presence' 'Presence' 'Presence' 'Presence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence'

'Absence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence' 'Presence'

'Absence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence'

'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'

'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence' 'Absence'

'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence'

'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'

'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence'

'Presence' 'Absence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Absence' 'Absence' 'Absence'

'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Presence' 'Presence' 'Absence' 'Absence' 'Presence' 'Presence'

'Absence' 'Presence' 'Absence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Presence' 'Absence' 'Absence' 'Presence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Presence' 'Presence' 'Absence'

'Presence' 'Absence' 'Absence' 'Absence' 'Absence' 'Presence' 'Presence'

'Absence' 'Absence' 'Presence' 'Presence' 'Absence' 'Presence' 'Absence'

'Absence' 'Absence' 'Absence' 'Presence']

In [73]:

DX\_train , DX\_test , dy\_train , dy\_test **=** train\_test\_split(DX,dy,test\_size**=**0.2,random\_state In [74]:

**from**

sklearn

.

tree

**import**

DecisionTreeClassifier

max\_accuracy

**=**

0

[75]:

In

[85]:

**for**

x

**in**

range

(

200

):

dt

**=**

DecisionTreeClassifier

(

random\_state

**=**

x

)

dt

.

fit

(

DX\_train

,

dy\_train

)

dy\_pred\_dt

**=**

dt

.

predict

(

DX\_test

)

current\_accuracy

**=**

round

(

accuracy\_score

(

dy\_pred\_dt

,

dy\_test

)

**\***

100

,

2

)

**if**

(

current\_accuracy

**>**

max\_accuracy

):

max\_accuracy

**=**

current\_accuracy

best\_x

**=**

x

dt

**=**

DecisionTreeClassifier

(

random\_state

**=**

best\_x

)

dt

.

fit

(

DX\_train

,

dy\_train

)

dy\_pred\_dt

**=**

dt

.

predict

(

DX\_test

)

In [88]:

score\_dt

**=**

(

accuracy\_score

(

dy\_pred\_dt

,

dy\_test

))

In [89]:

print

(

"The accuracy score achieved using Decision Tree is: "

**+**

str

(

score\_dt

))

The accuracy score achieved using Decision Tree is: 0.7962962962962963 In [90]:

d

**=**

(

accuracy\_score

(

dy\_pred\_dt

,

dy\_test

))

In

[91]:

Logistic Regression : 0.8271604938271605

KNN : 0.6111111111111112

print

(

'Logistic Regression :'

,

l

)

print

(

'KNN :'

,

k

)

print

(

'Naive Bayes :'

,

n

)

print

(

'Decision Tree :'

,

d

)

Naive Bayes : 0.7794117647058824

Decision Tree : 0.7962962962962963

In

[93]:

Logistic Regression : 82.71604938271605 %

KNN : 61.111111111111114 %

print

(

'Logistic Regression :'

,

l

**\***

100

,

'%'

)

print

(

'KNN :'

,

k

**\***

100

,

'%'

)

print

(

'Naive Bayes :'

,

n

**\***

100

,

'%'

)

print

(

'Decision Tree :'

,

d

**\***

100

,

'%'

)

Naive Bayes : 77.94117647058823 %

Decision Tree : 79.62962962962963 %

[ ]: