In [1]:
```
from nltk.tokenize import sent_tokenize
```

In [2]:
```
file=open('nlp.txt','r')
text=file.read()
print(text)
```

```
Natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligenc
e or AI, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
NLP combines computational linguistics with statistical, machine learning, and deep learning models. Together, these technologie
s enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with
the speaker or writer's intent and sentiment.
```

In [3]:
```
s=sent_tokenize(text)
```

In [7]:
```
print("number of sentences:",len(s))
for i in range(len(s)):
    print("\nSenteces",i+1,"\n",s[i])
```

```
number of sentences: 3

Senteces 1
 Natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligen
ce or AI, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

Senteces 2
 NLP combines computational linguistics with statistical, machine learning, and deep learning models.

Senteces 3
 Together, these technologies enable computers to process human language in the form of text or voice data and to understand its
full meaning, complete with the speaker or writer's intent and sentiment.
```

In [8]:
```
from nltk.tokenize import word_tokenize
```

In [9]:
```
w=word_tokenize(text)
```

In [10]:
```
print("total no of words:",len(w))
print(w)
```

```
total no of words: 94
['Natural', 'language', 'processing', 'refers', 'to', 'the', 'branch', 'of', 'computer', 'science', 'and', 'more', 'specificall
y', ',', 'the', 'branch', 'of', 'artificial', 'intelligence', 'or', 'AI', ',', 'concerned', 'with', 'giving', 'computers', 'th
e', 'ability', 'to', 'understand', 'text', 'and', 'spoken', 'words', 'in', 'much', 'the', 'same', 'way', 'human', 'beings', 'ca
n', '.', 'NLP', 'combines', 'computational', 'linguistics', 'with', 'statistical', ',', 'machine', 'learning', ',', 'and', 'dee
p', 'learning', 'models', '.', 'Together', ',', 'these', 'technologies', 'enable', 'computers', 'to', 'process', 'human', 'langu
age', 'in', 'the', 'form', 'of', 'text', 'or', 'voice', 'data', 'and', 'to', 'understand', 'its', 'full', 'meaning', ',', 'compl
ete', 'with', 'the', 'speaker', 'or', 'writer', "'s", 'intent', 'and', 'sentiment', '.']
```

In [11]:
```
w=word_tokenize(text,preserve_line=True)
len(w)
```

Out[11]:
```
92
```

In [12]:
```
from nltk.tokenize import word_tokenize
```

In [13]:
```
file=open('opinion.txt','r')
text=file.read()
```

In [14]:
```
w=word_tokenize(text)
len(w)
```

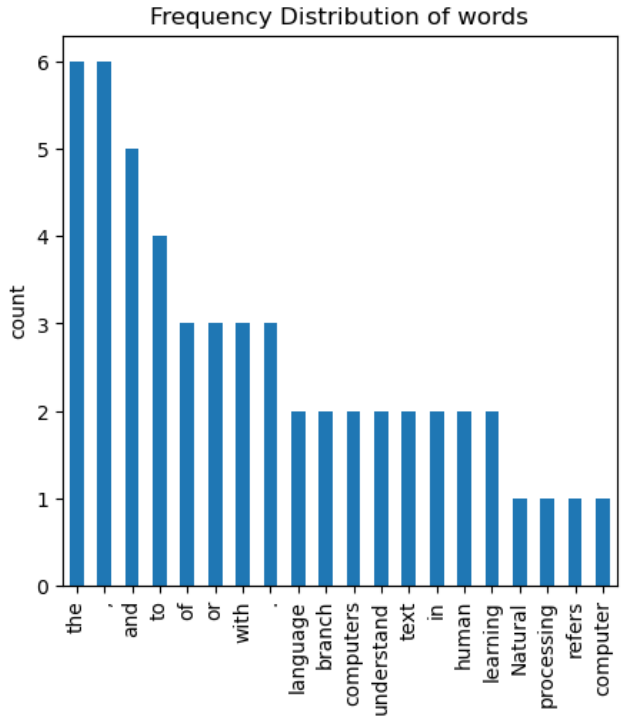
Out[14]:
```
93
```

In [16]:
```
from nltk.probability import FreqDist
afd=FreqDist(w).most_common(20)
print(afd)
```

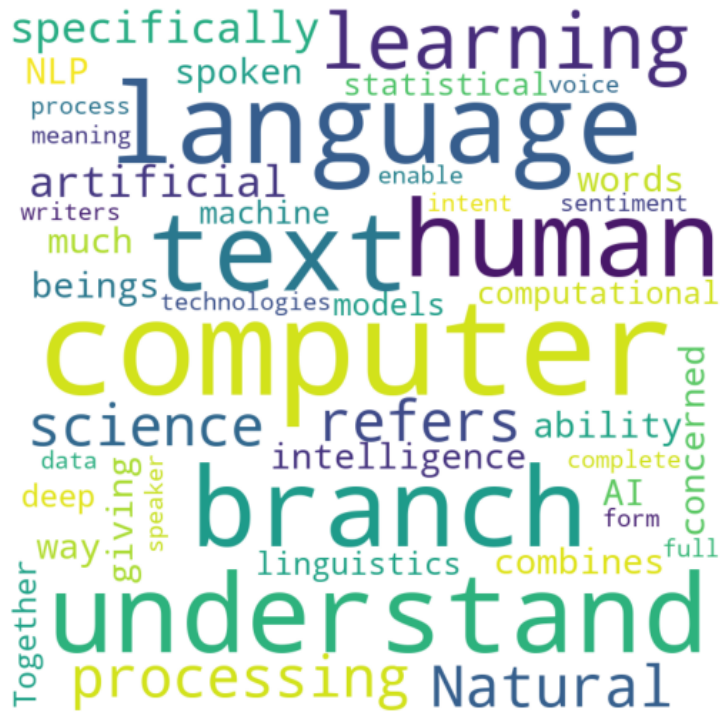
```
[('the', 6), (',', 6), ('and', 5), ('to', 4), ('of', 3), ('or', 3), ('with', 3), ('.', 3), ('language', 2), ('branch', 2), ('com
puters', 2), ('understand', 2), ('text', 2), ('in', 2), ('human', 2), ('learning', 2), ('Natural', 1), ('processing', 1), ('refe
rs', 1), ('computer', 1)]
```

In [21]:
```
import matplotlib.pyplot as plt
import pandas as pd
afd=pd.Series(dict(afd))
fig,ax=plt.subplots(figsize=(5,5))
afd.plot(kind='bar')
plt.title('Frequency Distribution of words')
plt.ylabel('count')
```

Out[21]:
```
Text(0, 0.5, 'count')
```

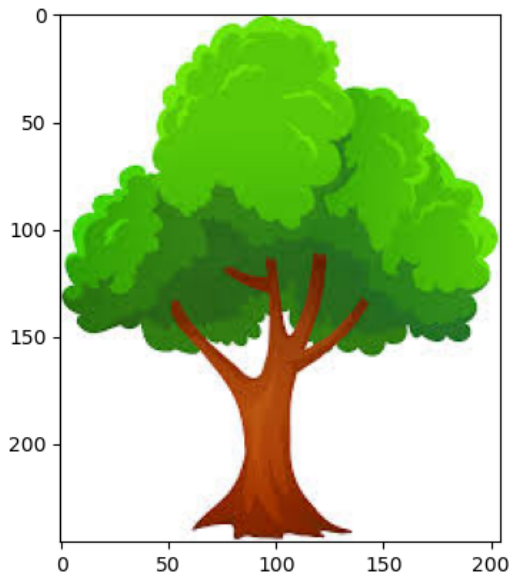


In [22]:
```
from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
s=set(STOPWORDS)
w=WordCloud(width=800,height=800,stopwords=s,background_color='white',min_font_size=10).generate(text)
plt.figure(figsize=(5,5),facecolor=None)
plt.imshow(w)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```



In [23]:
```
from skimage.io import imread
tree=imread('tree.jpeg')
plt.imshow(tree)
```

Out[23]:
```
<matplotlib.image.AxesImage at 0x1b43a75d4f0>
```



In [24]:
```
from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
s=set(STOPWORDS)
w=WordCloud(width=800,height=800,stopwords=s,background_color='white',min_font_size=10,mask=tree).generate(text)
plt.figure(figsize=(5,5),facecolor=None)
plt.imshow(w)
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
```

In [25]:
```
import nltk
from nltk.metrics.distance import edit_distance
```

In [26]:
```
nltk.download('words')
from nltk.corpus import words
cw=words.words()
```

```
[nltk_data] Error loading words: <urlopen error [Errno 11001]
[nltk_data]     getaddrinfo failed>
```

In [27]:
```
iw=['happpy','amzzzzzzzzzzzzzzing','inteliegent','cllege']
for word in iw:
    temp=[(edit_distance(word,w),w)for w in cw if w[0]==word[0]]
    print(sorted(temp,key=lambda val:val[0])[0][1])
```

```
happy
amazing
intelligent
college
```

In [28]:
```
print(text)
```

```
Natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligenc
e or AI, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
NLP combines computational linguistics with statistical, machine learning, and deep learning models. Together, these technologie
s enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with
the speaker or writers intent and sentiment.
```

In [29]:
```
text=text.lower()
print(text)
```

```
natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligenc
e or ai, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
nlp combines computational linguistics with statistical, machine learning, and deep learning models. together, these technologie
s enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with
the speaker or writers intent and sentiment.
```

In [30]:
```
text=text.upper()
print(text)
```

```
NATURAL LANGUAGE PROCESSING REFERS TO THE BRANCH OF COMPUTER SCIENCE AND MORE SPECIFICALLY, THE BRANCH OF ARTIFICIAL INTELLIGENC
E OR AI, CONCERNED WITH GIVING COMPUTERS THE ABILITY TO UNDERSTAND TEXT AND SPOKEN WORDS IN MUCH THE SAME WAY HUMAN BEINGS CAN.
NLP COMBINES COMPUTATIONAL LINGUISTICS WITH STATISTICAL, MACHINE LEARNING, AND DEEP LEARNING MODELS. TOGETHER, THESE TECHNOLOGIE
S ENABLE COMPUTERS TO PROCESS HUMAN LANGUAGE IN THE FORM OF TEXT OR VOICE DATA AND TO UNDERSTAND ITS FULL MEANING, COMPLETE WITH
THE SPEAKER OR WRITERS INTENT AND SENTIMENT.
```

In [31]:
```
text=text.lower()
print(text)
```

```
natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligenc
e or ai, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
nlp combines computational linguistics with statistical, machine learning, and deep learning models. together, these technologie
s enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with
the speaker or writers intent and sentiment.
```

In [32]:
```
import re
text=re.sub('A-Za-z0-9+',' ',text)
```

In [33]:
```
text=re.sub('\S*\d\S*','',text).strip()
```

In [34]:
```
print(text)
```

```
natural language processing refers to the branch of computer science and more specifically, the branch of artificial intelligenc
e or ai, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.
nlp combines computational linguistics with statistical, machine learning, and deep learning models. together, these technologie
s enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with
the speaker or writers intent and sentiment.
```

In [39]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
from keras.models import Sequential
from keras.layers import Dense
from sklearn import linear_model
from sklearn.metrics import mean_squared_error,mean_absolute_error
```

In [40]:
```
from tensorflow.keras.datasets import boston_housing
```

In [41]:
```
(X_train,y_train),(X_test,y_test)=boston_housing.load_data()
```

In [43]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
xts=scaler.transform(X_train)
xtss=scaler.transform(X_test)
```

In [44]:
```
model=Sequential()
model.add(Dense(128,input_dim=13,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='linear'))
```

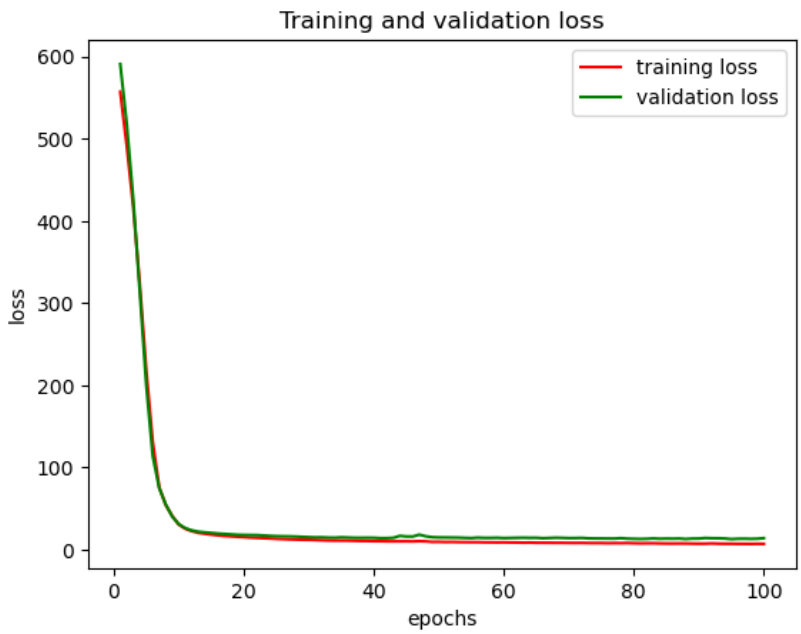
In [45]:
```
model.compile(loss="mean_squared_error",optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               1792

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 1)                 65

=================================================================
Total params: 10,113
Trainable params: 10,113
Non-trainable params: 0
_________________________________________________________________
```

In [46]:
```
history=model.fit(xts,y_train,validation_split=0.2,epochs=100)
```

```
Epoch 1/100
11/11 [==============================] - 1s 19ms/step - loss: 557.1803 - mae: 21.6952 - val_loss: 591.0316 - val_mae: 22.4798
Epoch 2/100
11/11 [==============================] - 0s 10ms/step - loss: 491.6327 - mae: 20.1423 - val_loss: 518.5300 - val_mae: 20.8277
Epoch 3/100
11/11 [==============================] - 0s 8ms/step - loss: 417.2877 - mae: 18.2381 - val_loss: 426.7161 - val_mae: 18.6190
Epoch 4/100
11/11 [==============================] - 0s 7ms/step - loss: 323.8158 - mae: 15.7198 - val_loss: 314.8686 - val_mae: 15.5700
Epoch 5/100
11/11 [==============================] - 0s 7ms/step - loss: 219.3841 - mae: 12.4983 - val_loss: 200.6183 - val_mae: 11.7559
Epoch 6/100
11/11 [==============================] - 0s 7ms/step - loss: 131.2752 - mae: 9.1717 - val_loss: 114.1032 - val_mae: 8.3631
Epoch 7/100
11/11 [==============================] - 0s 8ms/step - loss: 75.9480 - mae: 6.7917 - val_loss: 74.9093 - val_mae: 6.6123
Epoch 8/100
11/11 [==============================] - 0s 6ms/step - loss: 54.3442 - mae: 5.6346 - val_loss: 55.1284 - val_mae: 5.6099
Epoch 9/100
11/11 [==============================] - 0s 7ms/step - loss: 41.1180 - mae: 4.7316 - val_loss: 40.2237 - val_mae: 4.7762
Epoch 10/100
```

In [49]:
```
from matplotlib import pyplot as plt
loss=history.history['loss']
val_loss=history.history['val_loss']
e=range(1,len(loss)+1)
plt.plot(e,loss,'r',label='training loss')
plt.plot(e,val_loss,'g',label='validation loss')
plt.title('Training and validation loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



In [51]:
```
mse_neural,mae_neural=model.evaluate(xtss,y_test)
print("mean squared error from neural net:",mse_neural)
print("mean absolute error from neural net:",mae_neural)
```

```
4/4 [==============================] - 0s 0s/step - loss: 24.4369 - mae: 3.1795
mean squared error from neural net: 24.436948776245117
mean absolute error from neural net: 3.1794540882110596
```

In [52]:
```
p=model.predict(xtss[:5])
print("prediction values are:",p)
print("real values are:",y_test[:5])
```

```
1/1 [==============================] - 0s 92ms/step
prediction values are: [[ 8.2391  ]
 [17.724533]
 [21.247862]
 [32.558285]
 [25.145462]]
real values are: [ 7.2 18.8 19.  27.  22.2]
```

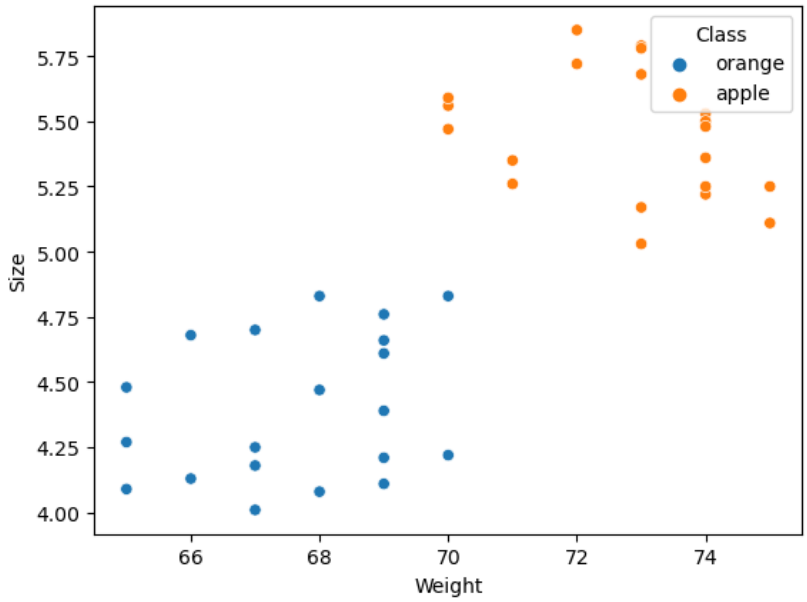
In [55]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

In [56]:
```
data=pd.read_csv('apples_and_oranges.csv')
print(data)
```

```
    Weight  Size   Class
0       69  4.39  orange
1       69  4.21  orange
2       65  4.09  orange
3       72  5.85   apple
4       67  4.70  orange
5       73  5.68   apple
6       70  5.56   apple
7       75  5.11   apple
8       74  5.36   apple
9       65  4.27  orange
10      73  5.79   apple
11      70  5.47   apple
12      74  5.53   apple
13      68  4.47  orange
14      74  5.22   apple
15      65  4.48  orange
16      69  4.66  orange
17      75  5.25   apple
18      67  4.18  orange
19      74  5.50   apple
20      66  4.13  orange
21      70  4.83  orange
22      69  4.61  orange
23      68  4.08  orange
24      67  4.25  orange
25      71  5.35   apple
26      67  4.01  orange
27      70  4.22  orange
28      74  5.25   apple
29      71  5.26   apple
30      73  5.78   apple
31      66  4.68  orange
32      72  5.72   apple
33      73  5.17   apple
34      68  4.83  orange
35      69  4.11  orange
36      69  4.76  orange
37      74  5.48   apple
38      70  5.59   apple
39      73  5.03   apple
```

In [59]:
```
import seaborn as sns
sns.scatterplot(x='Weight',y='Size',hue='Class',data=data)
```

Out[59]:
```
<AxesSubplot:xlabel='Weight', ylabel='Size'>
```



In [60]:
```
train_set,test_set=train_test_split(data,test_size=0.2,random_state=1)
print("train:",train_set)
print("test:",test_set)
```

```
train:     Weight  Size   Class
19      74  5.50   apple
26      67  4.01  orange
32      72  5.72   apple
17      75  5.25   apple
30      73  5.78   apple
36      69  4.76  orange
33      73  5.17   apple
28      74  5.25   apple
4       67  4.70  orange
14      74  5.22   apple
10      73  5.79   apple
35      69  4.11  orange
23      68  4.08  orange
24      67  4.25  orange
34      68  4.83  orange
20      66  4.13  orange
18      67  4.18  orange
25      71  5.35   apple
6       70  5.56   apple
13      68  4.47  orange
7       75  5.11   apple
38      70  5.59   apple
1       69  4.21  orange
16      69  4.66  orange
0       69  4.39  orange
15      65  4.48  orange
5       73  5.68   apple
11      70  5.47   apple
9       65  4.27  orange
8       74  5.36   apple
12      74  5.53   apple
37      74  5.48   apple
test:     Weight  Size   Class
2       65  4.09  orange
31      66  4.68  orange
3       72  5.85   apple
21      70  4.83  orange
27      70  4.22  orange
29      71  5.26   apple
22      69  4.61  orange
39      73  5.03   apple
```

In [62]:
```
x_train=train_set.iloc[:,0:2].values
y_train=train_set.iloc[:,2].values
x_test=test_set.iloc[:,0:2].values
y_test=test_set.iloc[:,2].values
print(x_train,y_train)
print(x_test,y_test)
```

```
[[74.    5.5 ]
 [67.    4.01]
 [72.    5.72]
 [75.    5.25]
 [73.    5.78]
 [69.    4.76]
 [73.    5.17]
 [74.    5.25]
 [67.    4.7 ]
 [74.    5.22]
 [73.    5.79]
 [69.    4.11]
 [68.    4.08]
 [67.    4.25]
 [68.    4.83]
 [66.    4.13]
 [67.    4.18]
 [71.    5.35]
 [70.    5.56]
 [68.    4.47]
 [75.    5.11]
 [70.    5.59]
 [69.    4.21]
 [69.    4.66]
 [69.    4.39]
 [65.    4.48]
 [73.    5.68]
 [70.    5.47]
 [65.    4.27]
 [74.    5.36]
 [74.    5.53]
 [74.    5.48]] ['apple' 'orange' 'apple' 'apple' 'apple' 'orange' 'apple' 'apple'
 'orange' 'apple' 'apple' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'apple' 'apple' 'orange' 'apple' 'apple' 'orange' 'orange'
 'orange' 'orange' 'apple' 'apple' 'orange' 'apple' 'apple' 'apple']
[[65.    4.09]
 [66.    4.68]
 [72.    5.85]
 [70.    4.83]
 [70.    4.22]
 [71.    5.26]
 [69.    4.61]
 [73.    5.03]] ['orange' 'orange' 'apple' 'orange' 'orange' 'apple' 'orange' 'apple']
```

In [63]:
```
c=SVC(kernel='rbf',random_state=1,C=1,gamma='auto')
c.fit(x_train,y_train)
```

Out[63]:
```
SVC(C=1, gamma='auto', random_state=1)
```

In [64]:
```
yp=c.predict(x_test)
print(yp)
```

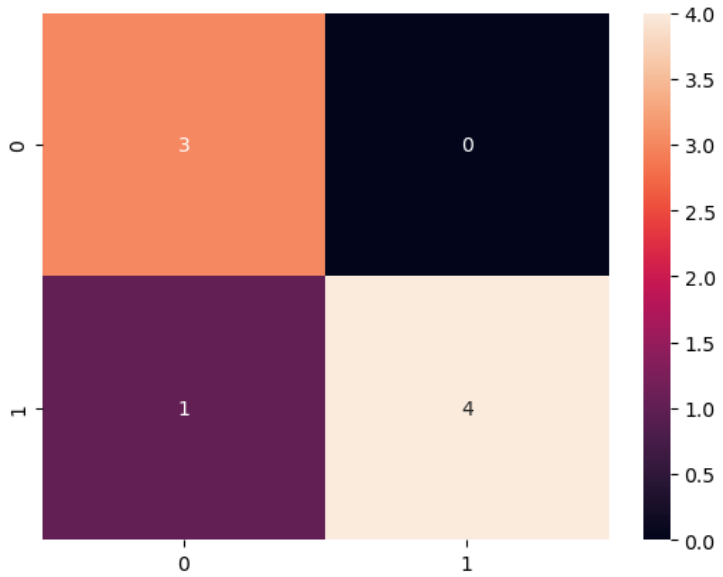
```
['orange' 'orange' 'apple' 'apple' 'orange' 'apple' 'orange' 'apple']
```

In [66]:
```
cm=confusion_matrix(y_test,yp)
print(cm)
accuracy=float(cm.diagonal().sum())/len(y_test)
print("model accuracy is:",accuracy*100,'%')
```

```
[[3 0]
 [1 4]]
model accuracy is: 87.5 %
```

In [67]:
```
sns.heatmap(cm,annot=True)
```

Out[67]: <AxesSubplot:>



In [1]:
```
import pandas as pd
f=pd.read_csv('dataset_Fish.csv')
f.head()
```

Out[1]:

	Species	Weight	Length1	Length2	Length3	Height	Width
0	Bream	242.0	23.2	25.4	30.0	11.5200	4.0200
1	Bream	290.0	24.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	23.9	26.5	31.1	12.3778	4.6961
3	Bream	363.0	26.3	29.0	33.5	12.7300	4.4555
4	Bream	430.0	26.5	29.0	34.0	12.4440	5.1340

In [2]:
```
f['Species'].unique()
```

Out[2]:
```
array(['Bream', 'Roach', 'Whitefish', 'Parkki', 'Perch', 'Pike', 'Smelt'],
      dtype=object)
```

In [3]:
```
f.isnull().sum()
```

Out[3]:
```
Species    0
Weight     0
Length1    0
Length2    0
Length3    0
Height     0
Width      0
dtype: int64
```

In [4]:
```
y=f['Species']
x=f.drop('Species',axis=1)
```

In [5]:
```
x
```

Out[5]:

	Weight	Length1	Length2	Length3	Height	Width
0	242.0	23.2	25.4	30.0	11.5200	4.0200
1	290.0	24.0	26.3	31.2	12.4800	4.3056
2	340.0	23.9	26.5	31.1	12.3778	4.6961
3	363.0	26.3	29.0	33.5	12.7300	4.4555
4	430.0	26.5	29.0	34.0	12.4440	5.1340
...	...	...	...	...	...	...
154	12.2	11.5	12.2	13.4	2.0904	1.3936
155	13.4	11.7	12.4	13.5	2.4300	1.2690
156	12.2	12.1	13.0	13.8	2.2770	1.2558
157	19.7	13.2	14.3	15.2	2.8728	2.0672
158	19.9	13.8	15.0	16.2	2.9322	1.8792

159 rows × 6 columns

In [6]:
```
y
```

Out[6]:
```
0      Bream
1      Bream
2      Bream
3      Bream
4      Bream
       ...
154    Smelt
155    Smelt
156    Smelt
157    Smelt
158    Smelt
Name: Species, Length: 159, dtype: object
```

In [14]:
```
from sklearn.preprocessing import MinMaxScaler
s=MinMaxScaler()
s.fit(x)
xs=s.transform(x)
print(xs)
```

```
[[1.46666667e-01 3.04854369e-01 3.09090909e-01 3.58108108e-01
  5.68334049e-01 4.18978349e-01]
 [1.75757576e-01 3.20388350e-01 3.25454545e-01 3.78378378e-01
  6.24055350e-01 4.59235453e-01]
 [2.06060606e-01 3.18446602e-01 3.29090909e-01 3.76689189e-01
  6.18123353e-01 5.14278868e-01]
 [2.20000000e-01 3.65048544e-01 3.74545455e-01 4.17229730e-01
  6.38566105e-01 4.80364795e-01]
 [2.60606061e-01 3.68932039e-01 3.74545455e-01 4.25675676e-01
  6.21965801e-01 5.76003608e-01]
 [2.72727273e-01 3.74757282e-01 3.87272727e-01 4.37500000e-01
  6.89202837e-01 5.46882048e-01]
 [3.03030303e-01 3.74757282e-01 3.87272727e-01 4.34121622e-01
  7.22699465e-01 5.96371786e-01]
 [2.36363636e-01 3.90291262e-01 3.92727273e-01 4.42567568e-01
  6.35083524e-01 5.13419035e-01]
 [2.72727273e-01 3.90291262e-01 3.92727273e-01 4.44256757e-01
  7.12565153e-01 5.35098106e-01]
 [3.03030303e-01 4.07766990e-01 4.05454545e-01 4.62837838e-01
```

In [8]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)
y
```

Out[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 6, 6, 6, 6, 6, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5])
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(xs,y,test_size=0.2,random_state=42)
```

In [10]:
```
print(X_train)
```

```
[[0.03121212 0.14563107 0.14181818 0.14189189 0.16623521 0.2232747 ]
 [0.34363636 0.69320388 0.68363636 0.67398649 0.35194967 0.53879116]
 [0.20606061 0.3184466  0.32909091 0.37668919 0.61812335 0.51427887]
 [0.07272727 0.24271845 0.24727273 0.24831081 0.2543213  0.33264265]
 [0.0969697  0.25242718 0.25636364 0.27871622 0.30791823 0.39082939]
 [0.07272727 0.23106796 0.22909091 0.25168919 0.2545883  0.3166864 ]
 [0.08787879 0.23883495 0.23818182 0.25844595 0.46480852 0.2973472 ]
 [0.4969697  0.56504854 0.55636364 0.54898649 0.62122865 0.88855999]
 [0.43636364 0.47572816 0.48363636 0.53716216 0.84936675 0.71075778]
 [0.3030303  0.66990291 0.66545455 0.66216216 0.30365787 0.54245602]
 [0.00454545 0.04854369 0.03818182 0.0472973  0.01413928 0.01584348]
 [0.06666667 0.24271845 0.24727273 0.24831081 0.22022103 0.41545444]
 [0.08484848 0.22330097 0.22363636 0.24324324 0.39522654 0.3167005 ]
 [0.04181818 0.17475728 0.17818182 0.19425676 0.20720778 0.25007048]
 [0.06666667 0.22330097 0.22909091 0.23141892 0.23008834 0.35343369]
 [0.37575758 0.46601942 0.47454545 0.52195946 0.80066285 0.59659732]
 [0.07878788 0.22912621 0.23454545 0.23648649 0.27022509 0.35047361]
 [0.05151515 0.20776699 0.21090909 0.20608108 0.19465308 0.24306495]
 [0.41515152 0.51456311 0.51090909 0.51013514 0.53124456 0.81985792]
```

In [11]:
```
print(X_test)
```

```
[[4.72727273e-02 1.80582524e-01 1.87272727e-01 1.79054054e-01
  2.01455719e-01 2.92596978e-01]
 [8.12121212e-03 8.15533981e-02 7.27272727e-02 7.93918919e-02
  4.07229839e-02 3.12077131e-02]
 [1.21212121e-01 4.36893204e-01 4.34545455e-01 4.39189189e-01
  2.22861985e-01 3.28146143e-01]
 [1.63636364e-01 3.12621359e-01 3.20000000e-01 3.36148649e-01
  3.86102179e-01 4.51059991e-01]
 [9.09090909e-02 2.62135922e-01 2.65454545e-01 2.65202703e-01
  2.02575949e-01 3.63441588e-01]
 [6.06060606e-01 5.04854369e-01 5.20000000e-01 5.70945946e-01
  1.00000000e+00 7.83068336e-01]
 [4.24242424e-03 5.04854369e-02 4.00000000e-02 4.72972973e-02
  0.00000000e+00 1.42083897e-02]
 [1.09090909e-01 3.12621359e-01 3.05454545e-01 3.22635135e-01
  3.11006118e-01 4.02909337e-01]
 [1.13939394e-01 2.93203883e-01 2.94545455e-01 2.93918919e-01
  2.90505323e-01 4.39529770e-01]
 [7.57575758e-01 8.64077670e-01 8.65454545e-01 8.59797297e-01
```

In [12]:
```
print(y_train)
```

```
[2 3 0 2 4 4 1 2 0 3 5 2 1 4 2 0 2 2 2 0 2 1 2 0 5 2 4 0 3 0 3 2 1 0 2 3 0
 2 1 4 2 0 4 2 0 0 2 2 4 3 2 5 3 4 2 1 2 0 2 5 2 1 5 2 2 4 0 2 2 4 4 2 2 0
 4 0 4 1 2 2 2 2 3 3 2 0 2 0 6 3 0 0 4 2 3 0 3 1 4 2 4 5 6 4 2 0 6 5 3 4 5
 0 4 3 2 2 2 2 2 2 5 0 1 2 0 2 2]
```

In [13]:
```
print(y_test)
```

```
[2 5 3 6 2 0 5 4 2 3 0 6 0 1 0 0 2 2 2 0 5 0 0 0 0 2 2 6 5 5 3 2]
```

In [15]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
lr.fit(X_train,y_train)
```

Out[15]:
```
LogisticRegression()
```

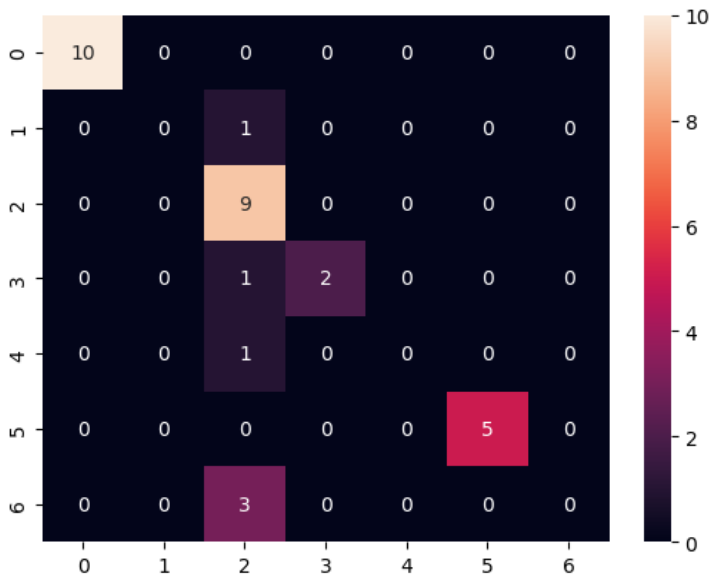
In [16]:
```
yp=lr.predict(X_test)
```

In [17]:
```
from sklearn.metrics import accuracy_score
a=accuracy_score(y_test,yp)
print("Accuracy:{:.2f}".format(a*100))
```

```
Accuracy:81.25
```

In [18]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
cf=confusion_matrix(y_test,yp)
plt.figure()
sns.heatmap(cf,annot=True)
```

Out[18]:
```
<AxesSubplot:>
```



In [40]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [41]:
```
data=pd.read_csv('Mall_Customers.csv')
print(data.shape)
```

```
(200, 5)
```

In [42]:
```
data=data.drop('CustomerID',axis=1)
data
```

Out[42]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40
...	...	...	...	...
195	Female	35	120	79
196	Female	45	126	28
197	Male	32	126	74
198	Male	32	137	18
199	Male	30	137	83

200 rows × 4 columns

In [43]:
```
data.head()
```

Out[43]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40

In [44]:
```
data.describe()
```

Out[44]:

	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000
mean	38.850000	60.560000	50.200000
std	13.969007	26.264721	25.823522
min	18.000000	15.000000	1.000000
25%	28.750000	41.500000	34.750000
50%	36.000000	61.500000	50.000000
75%	49.000000	78.000000	73.000000
max	70.000000	137.000000	99.000000

In [45]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data['Gender']=le.fit_transform(data['Gender'])
data
```

Out[45]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	19	15	39
1	1	21	15	81
2	0	20	16	6
3	0	23	16	77
4	0	31	17	40
...	...	...	...	...
195	0	35	120	79
196	0	45	126	28
197	1	32	126	74
198	1	32	137	18
199	1	30	137	83

200 rows × 4 columns

In [46]:
```
sns.pairplot(data)
```

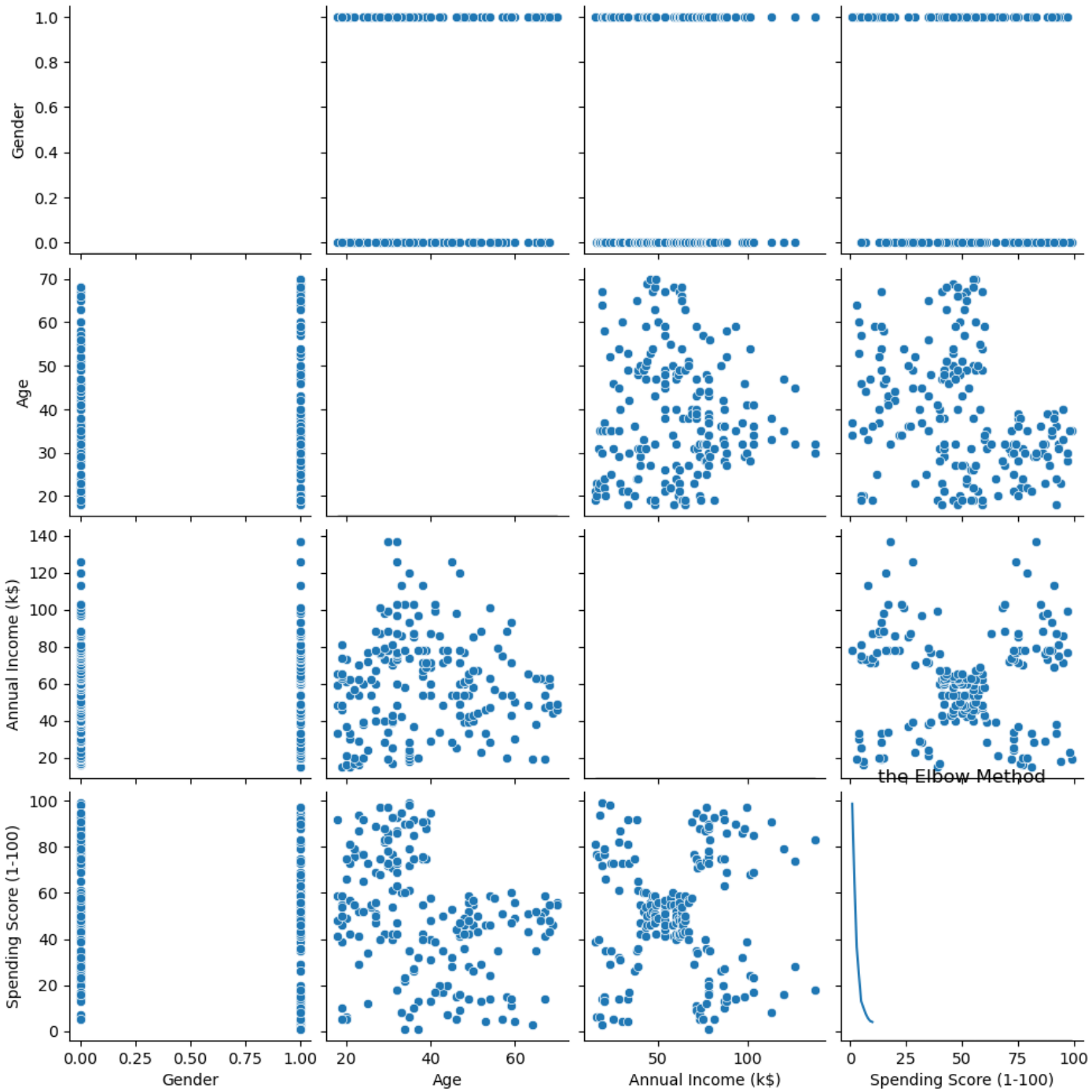
Out[46]:
```
<seaborn.axisgrid.PairGrid at 0x23acb7bce20>
```

In [47]:
```
x=data.iloc[:,2:4].values
print(x.shape)
```

```
(200, 2)
```

In [48]:
```
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('the Elbow Method')
plt.xlabel('no of clusters')
plt.ylabel('wcss')
plt.show()
```

```
C:\Users\srini\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak o
n Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_
NUM_THREADS=1.
  warnings.warn(
```



In [49]:
```
km1=KMeans(n_clusters=3,init='k-means++',max_iter=300,n_init=10,random_state=0)
y_means=km1.fit_predict(x)
```

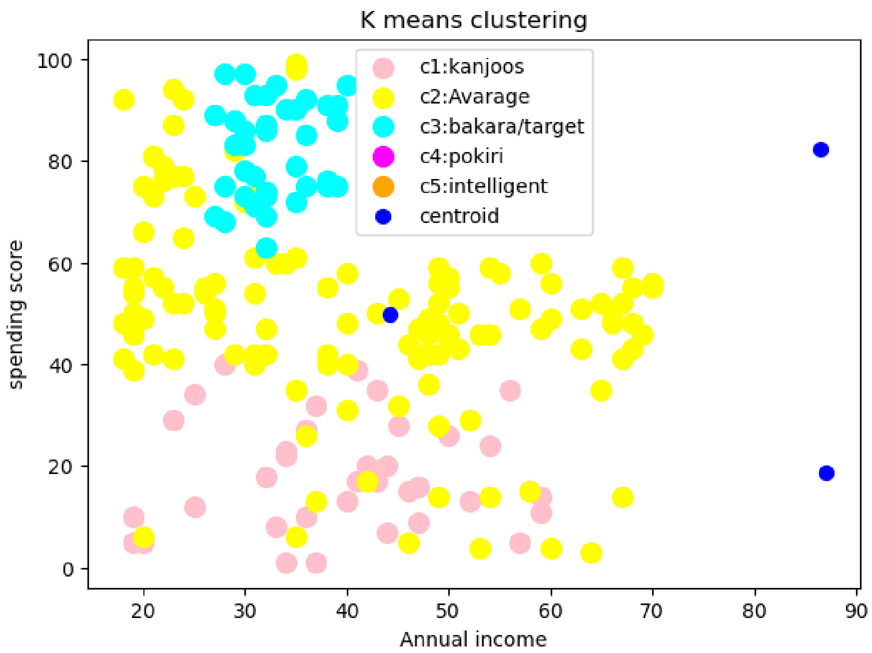
In [50]:
```
y_means
```

Out[50]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

In [60]:
```
km1.cluster_centers_
```

Out[60]:
```
array([[87.        , 18.63157895],
       [44.15447154, 49.82926829],
       [86.53846154, 82.12820513]])
```

In [61]:
```
plt.scatter(x[y_means==0,0],x[y_means==0,1],s=100,c='pink',label='c1:kanjoos')
plt.scatter(x[y_means==1,0],x[y_means==1,1],s=100,c='yellow',label='c2:Avarage')
plt.scatter(x[y_means==2,0],x[y_means==2,1],s=100,c='cyan',label='c3:bakara/target')
plt.scatter(x[y_means==3,0],x[y_means==3,1],s=100,c='magenta',label='c4:pokiri')
plt.scatter(x[y_means==4,0],x[y_means==4,1],s=100,c='orange',label='c5:intelligent')

plt.scatter(km1.cluster_centers_[:,0],km1.cluster_centers_[:,1],s=50,c='blue',label='centroid')
plt.title("K means clustering")
plt.xlabel('Annual income')
plt.ylabel('spending score')
plt.legend()
plt.show()
```



In [62]:
```
x[y_means==1,0]
```

Out[62]:
```
array([19, 21, 20, 23, 31, 22, 35, 23, 64, 30, 67, 35, 58, 24, 37, 22, 35,
       20, 52, 35, 35, 25, 46, 31, 54, 29, 45, 35, 40, 23, 60, 21, 53, 18,
       49, 21, 42, 30, 36, 20, 65, 24, 48, 31, 49, 24, 50, 27, 29, 31, 49,
       33, 31, 59, 50, 47, 51, 69, 27, 53, 70, 19, 67, 54, 63, 18, 43, 68,
       19, 32, 70, 47, 60, 60, 59, 26, 45, 40, 23, 49, 57, 38, 67, 46, 21,
       48, 55, 22, 34, 50, 68, 18, 48, 40, 32, 24, 47, 27, 48, 20, 23, 49,
       67, 26, 49, 21, 66, 54, 68, 66, 65, 19, 38, 19, 18, 19, 63, 49, 51,
       50, 27, 38, 40], dtype=int64)
```

In [63]:
```
km1.inertia_
```

Out[63]:
```
106348.37306211118
```

In [64]:
```
km1.cluster_centers_
```

Out[64]:
```
array([[87.        , 18.63157895],
       [44.15447154, 49.82926829],
       [86.53846154, 82.12820513]])
```

In [65]:
```
data.columns
```

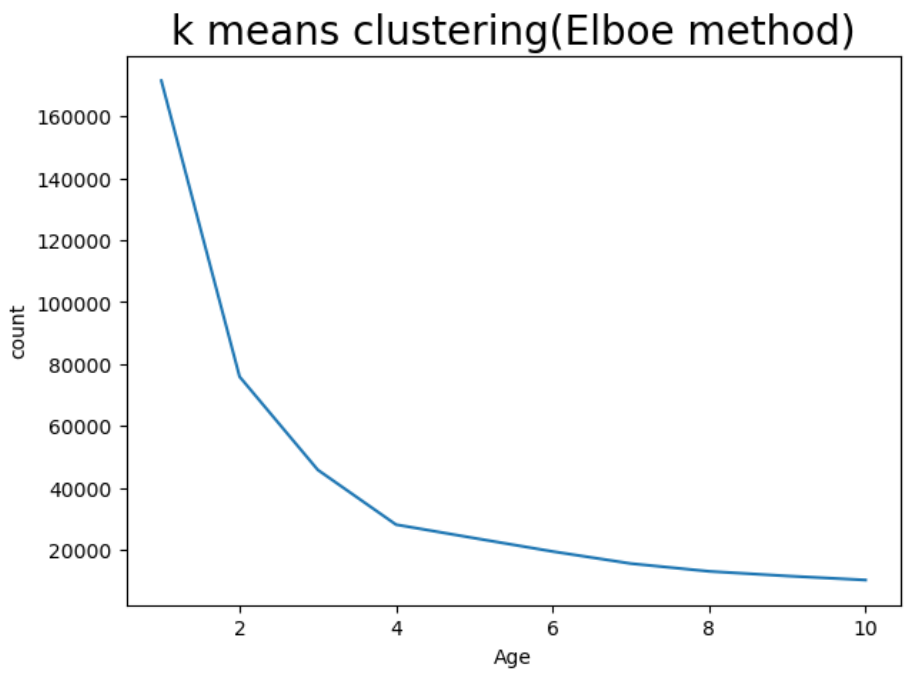
Out[65]:
```
Index(['Gender', 'Age', 'Annual Income (k$)', 'Spending Score (1-100)'], dtype='object')
```

In [66]:
```
x=data.iloc[:,[1,3]].values
x.shape
```

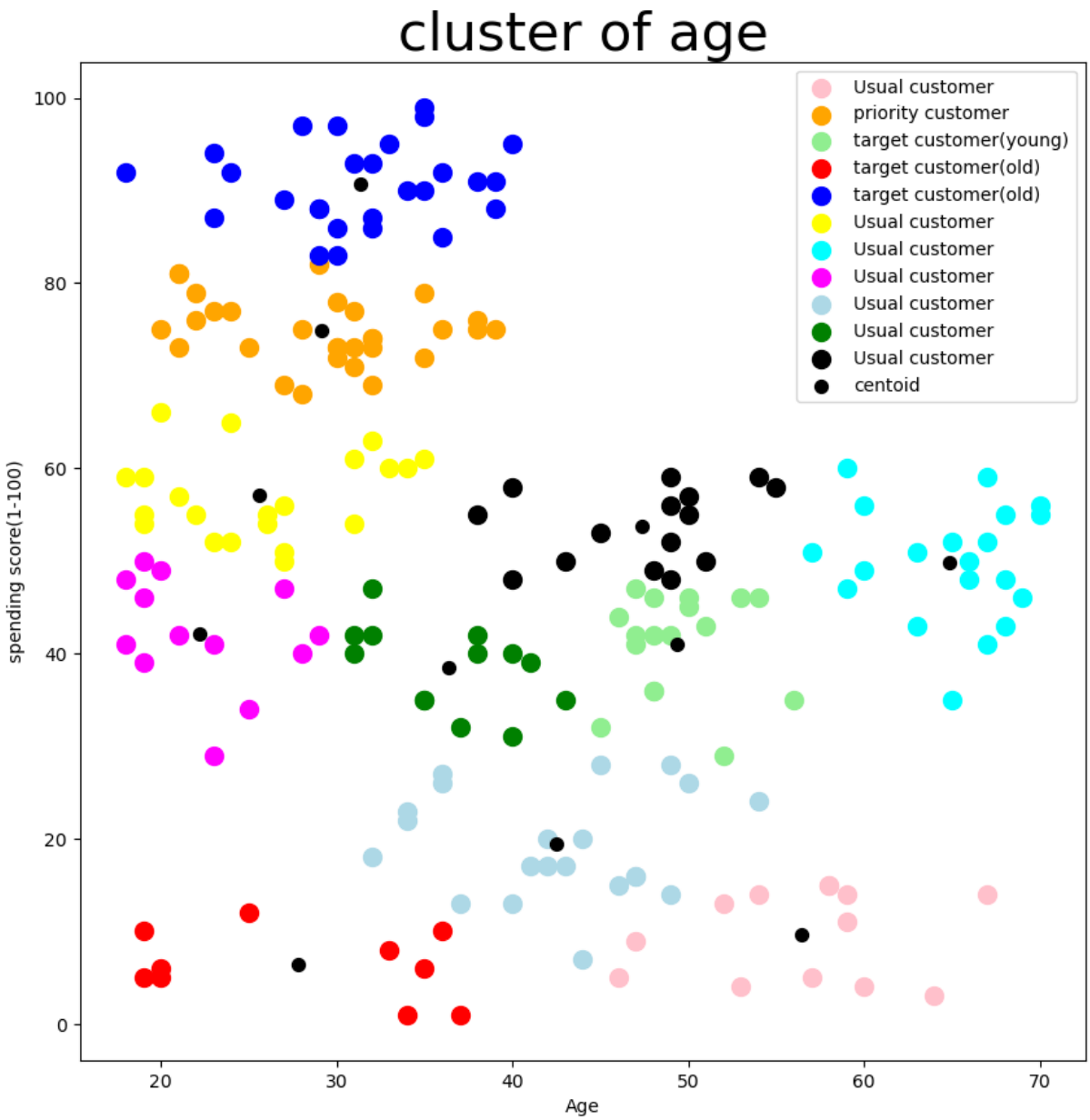
Out[66]:
```
(200, 2)
```

In [67]:
```
from sklearn.cluster import KMeans

wcss=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
plt.rcParams['figure.figsize']=(7,5)
plt.plot(range(1,11),wcss)
plt.title('k means clustering(Elboe method)',fontsize=20)
plt.xlabel('Age')
plt.ylabel('count')
plt.show()
```

```
C:\Users\srini\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak o
n Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_
NUM_THREADS=1.
  warnings.warn(
```



In [79]:
```
km2=KMeans(n_clusters=11,init='k-means++',max_iter=300,n_init=10,random_state=0)
y_means=km2.fit_predict(x)

plt.rcParams['figure.figsize']=(10,10)
plt.title('cluster of age',fontsize=30)

plt.scatter(x[y_means==0,0],x[y_means==0,1],s=100,c='pink',label='Usual customer')
plt.scatter(x[y_means==1,0],x[y_means==1,1],s=100,c='orange',label='priority customer')
plt.scatter(x[y_means==2,0],x[y_means==2,1],s=100,c='lightgreen',label='target customer(young)')
plt.scatter(x[y_means==3,0],x[y_means==3,1],s=100,c='red',label='target customer(old)')
plt.scatter(x[y_means==4,0],x[y_means==4,1],s=100,c='blue',label='target customer(old)')
plt.scatter(x[y_means==5,0],x[y_means==5,1],s=100,c='yellow',label='Usual customer')
plt.scatter(x[y_means==6,0],x[y_means==6,1],s=100,c='cyan',label='Usual customer')
plt.scatter(x[y_means==7,0],x[y_means==7,1],s=100,c='magenta',label='Usual customer')
plt.scatter(x[y_means==8,0],x[y_means==8,1],s=100,c='lightblue',label='Usual customer')
plt.scatter(x[y_means==9,0],x[y_means==9,1],s=100,c='green',label='Usual customer')
plt.scatter(x[y_means==10,0],x[y_means==10,1],s=100,c='black',label='Usual customer')
plt.scatter(km2.cluster_centers_[:,0],km2.cluster_centers_[:,1],s=50,c='black',label='centoid')

plt.xlabel('Age')
plt.ylabel('spending score(1-100)')
plt.legend()
plt.show()
```



In [86]:
```
km2.inertia_
```

Out[86]:
```
9613.709528597763
```

In [87]:
```
km2.cluster_centers_
```

Out[87]:
```
array([[56.46153846,  9.69230769],
       [29.1       , 74.83333333],
       [49.35294118, 41.05882353],
       [27.8       ,  6.4       ],
       [31.37037037, 90.7037037 ],
       [25.61904762, 57.0952381 ],
       [64.85      , 49.85      ],
       [22.23076923, 42.15384615],
       [42.47619048, 19.38095238],
       [36.38461538, 38.46153846],
       [47.33333333, 53.8       ]])
```

In [80]:
```
x=data.iloc[:,[0,3]].values
x.shape
```

Out[80]:
```
(200, 2)
```

In [83]:
```
df=pd.DataFrame(x)
df
```

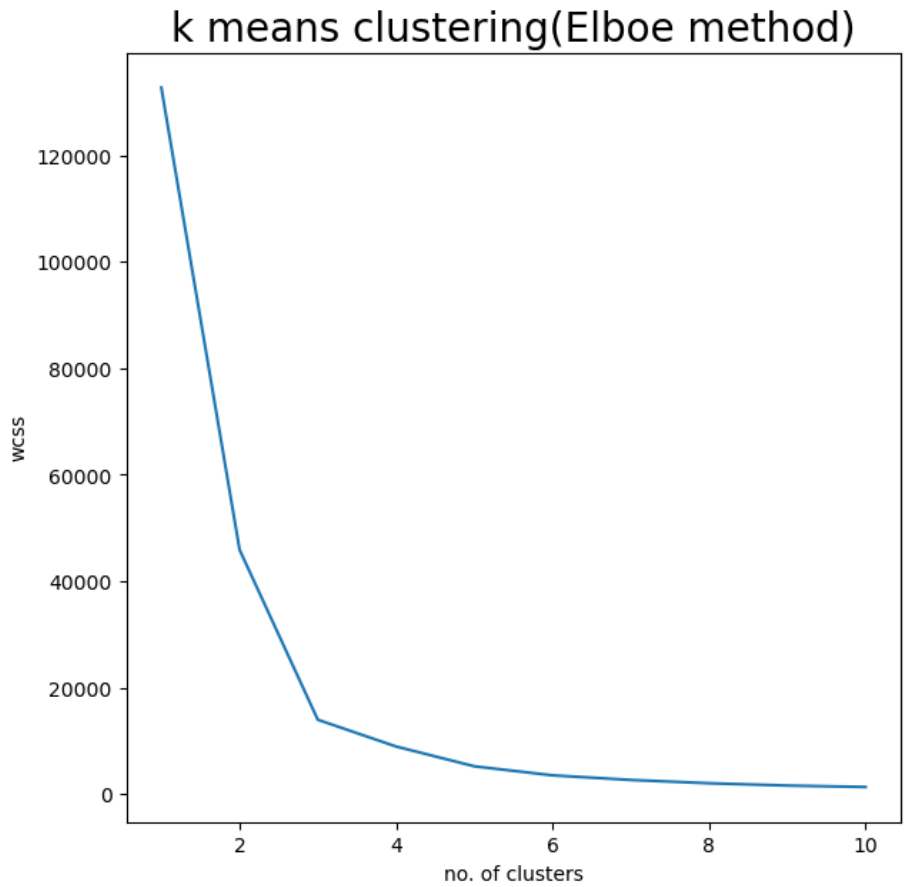
Out[83]:

	0	1
0	1	39
1	1	81
2	0	6
3	0	77
4	0	40
...	...	...
195	0	79
196	0	28
197	1	74
198	1	18
199	1	83

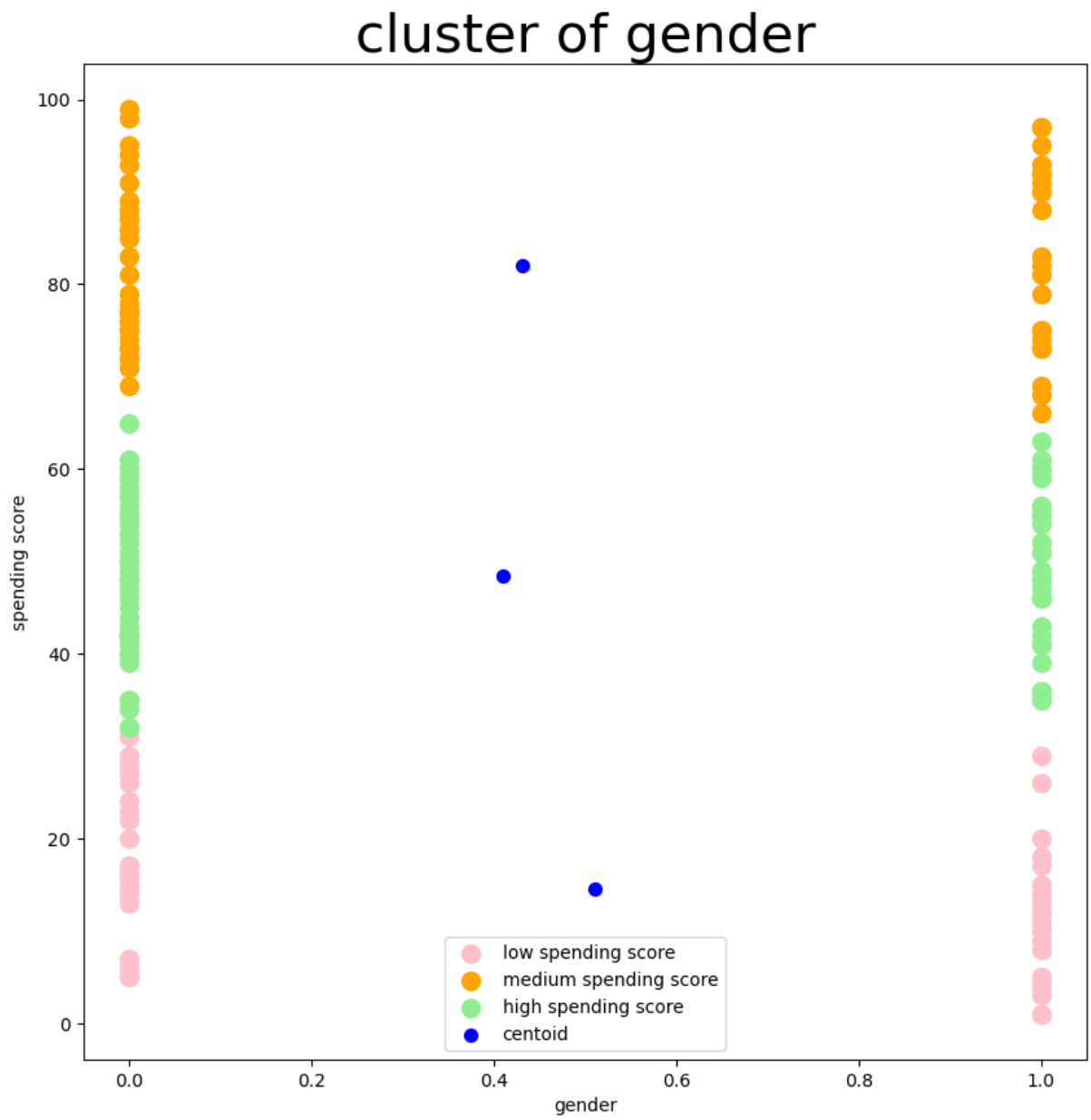
200 rows × 2 columns

In [84]:
```
from sklearn.cluster import KMeans

wcss=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i,init='k-means++',max_iter=300,n_init=10,random_state=0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
plt.rcParams['figure.figsize']=(7,7)
plt.plot(range(1,11),wcss)
plt.title('k means clustering(Elboe method)',fontsize=20)
plt.xlabel('no. of clusters')
plt.ylabel('wcss')
plt.show()
```

```
C:\Users\srini\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak o
n Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_
NUM_THREADS=1.
  warnings.warn(
```



In [90]:
```
km3=KMeans(n_clusters=3,init='k-means++',max_iter=300,n_init=10,random_state=0)
y_means=km3.fit_predict(x)

plt.rcParams['figure.figsize']=(10,10)
plt.title('cluster of gender',fontsize=30)

plt.scatter(x[y_means==0,0],x[y_means==0,1],s=100,c='pink',label='low spending score')
plt.scatter(x[y_means==1,0],x[y_means==1,1],s=100,c='orange',label='medium spending score')
plt.scatter(x[y_means==2,0],x[y_means==2,1],s=100,c='lightgreen',label='high spending score')
plt.scatter(km3.cluster_centers_[:,0],km3.cluster_centers_[:,1],s=50,c='blue',label='centoid')

plt.xlabel('gender')
plt.ylabel('spending score')
plt.legend()
plt.show()
```



In [91]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [94]:
```
from sklearn.datasets import load_boston
bdf=load_boston()
bdf.keys()
```

Out[94]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_module'])
```

In [95]:
```
b=pd.DataFrame(bdf.data,columns=bdf.feature_names)
b.head()
```

Out[95]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [96]:
```
b.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
dtypes: float64(13)
memory usage: 51.5 KB
```

In [97]:
```
b['TARGET']=bdf.target
```

In [98]:
```
b
```

Out[98]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	TARGET
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88	11.9

506 rows × 14 columns

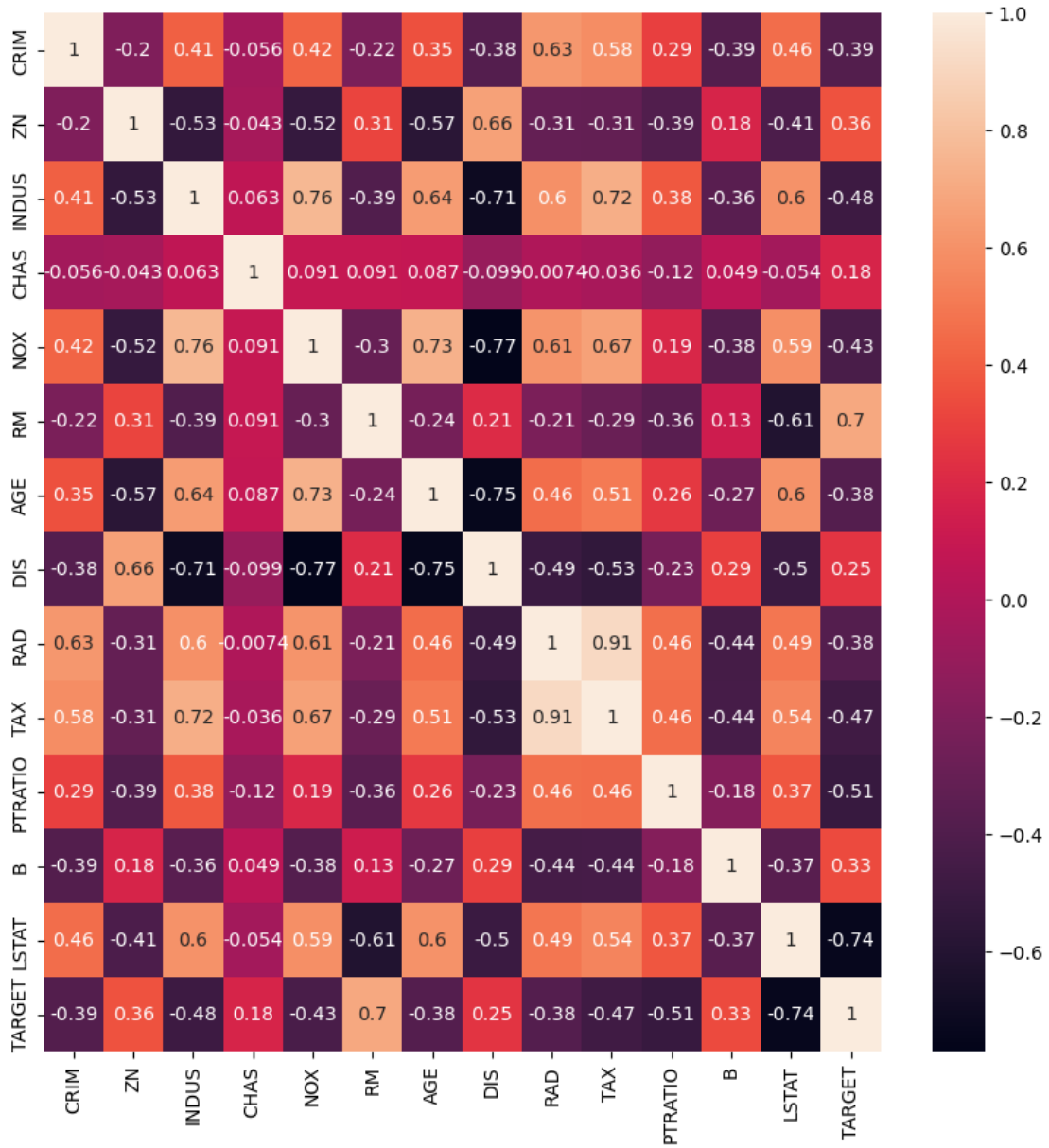
In [99]:
```
b.isnull().sum()
```

Out[99]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
TARGET     0
dtype: int64
```

In [100]:
```
cm=b.corr()
```

In [101]:
```
sns.heatmap(data=cm,annot=True)
```

Out[101]: <AxesSubplot:>



In [102]:
```
x=pd.DataFrame(np.c_[b['LSTAT'],b['RM']],columns=['LSTAT','RM'])
y=b['TARGET']
```

In [103]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=5)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

In [104]:
```
print(x_train)
```

```
     LSTAT     RM
33   18.35  5.701
283   3.16  7.923
418  20.62  5.957
502   9.08  6.120
402  20.31  6.404
..     ...    ...
486  14.98  6.114
189   5.39  7.185
495  17.60  5.670
206  10.97  6.326
355   5.57  5.936

[404 rows x 2 columns]
```

In [105]:
```
x
```

Out[105]:

	LSTAT	RM
0	4.98	6.575
1	9.14	6.421
2	4.03	7.185
3	2.94	6.998
4	5.33	7.147
...	...	...
501	9.67	6.593
502	9.08	6.120
503	5.64	6.976
504	6.48	6.794
505	7.88	6.030

506 rows × 2 columns

In [106]:
```
y
```

Out[106]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: TARGET, Length: 506, dtype: float64
```

In [107]:
```
print(y_train)
```

```
33     13.1
283    50.0
418     8.8
502    20.6
402    12.1
       ...
486    19.1
189    34.9
495    23.1
206    24.4
355    20.6
Name: TARGET, Length: 404, dtype: float64
```

In [108]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[108]:
```
LinearRegression()
```

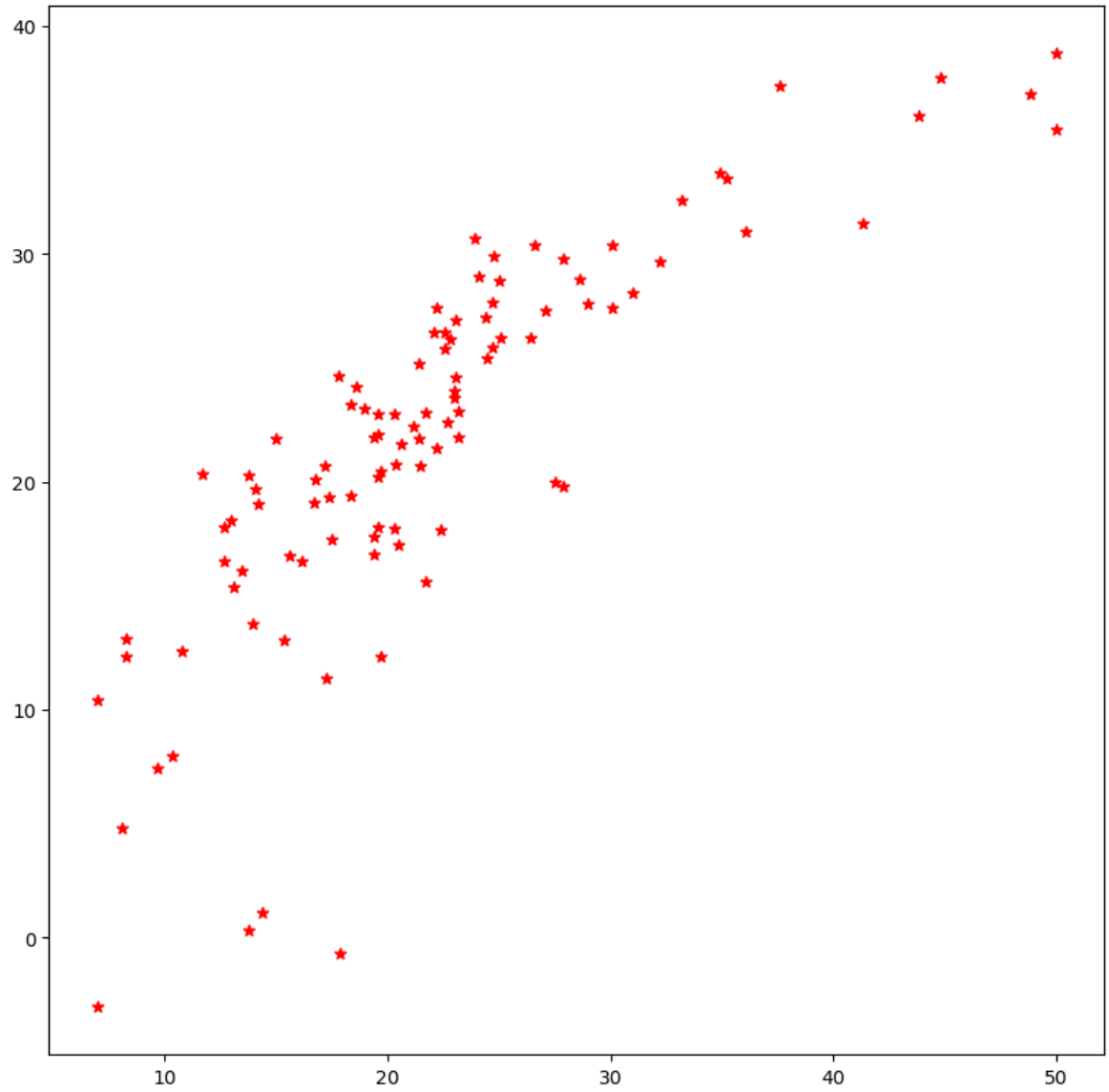
In [111]:
```
ytp=lr.predict(x_train)
rmse=(np.sqrt(mean_squared_error(y_train,ytp)))
r2=r2_score(y_train,ytp)

print("the model performance on training data")
print('---------------------------------------')
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
the model performance on training data
---------------------------------------
RMSE is 5.6371293350711955
R2 score is 0.6300745149331701
```

In [113]:
```
ytsp=lr.predict(x_test)
rmse=(np.sqrt(mean_squared_error(y_test,ytsp)))
r2=r2_score(y_test,ytsp)

print("the model performance on test data")
print('---------------------------------------')
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
the model performance on test data
---------------------------------------
RMSE is 5.137400784702911
R2 score is 0.6628996975186953
```

In [114]:
```
plt.scatter(y_test,ytsp,color='red',marker='*')
plt.show()
```



In [118]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
idf=datasets.load_iris()
x=idf.data[:,[2,3]]
y=idf.target
```

In [119]:
```
idf.keys()
```

Out[119]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [121]:
```
i=pd.DataFrame(idf.data,columns=idf.feature_names)
i.head()
```

Out[121]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

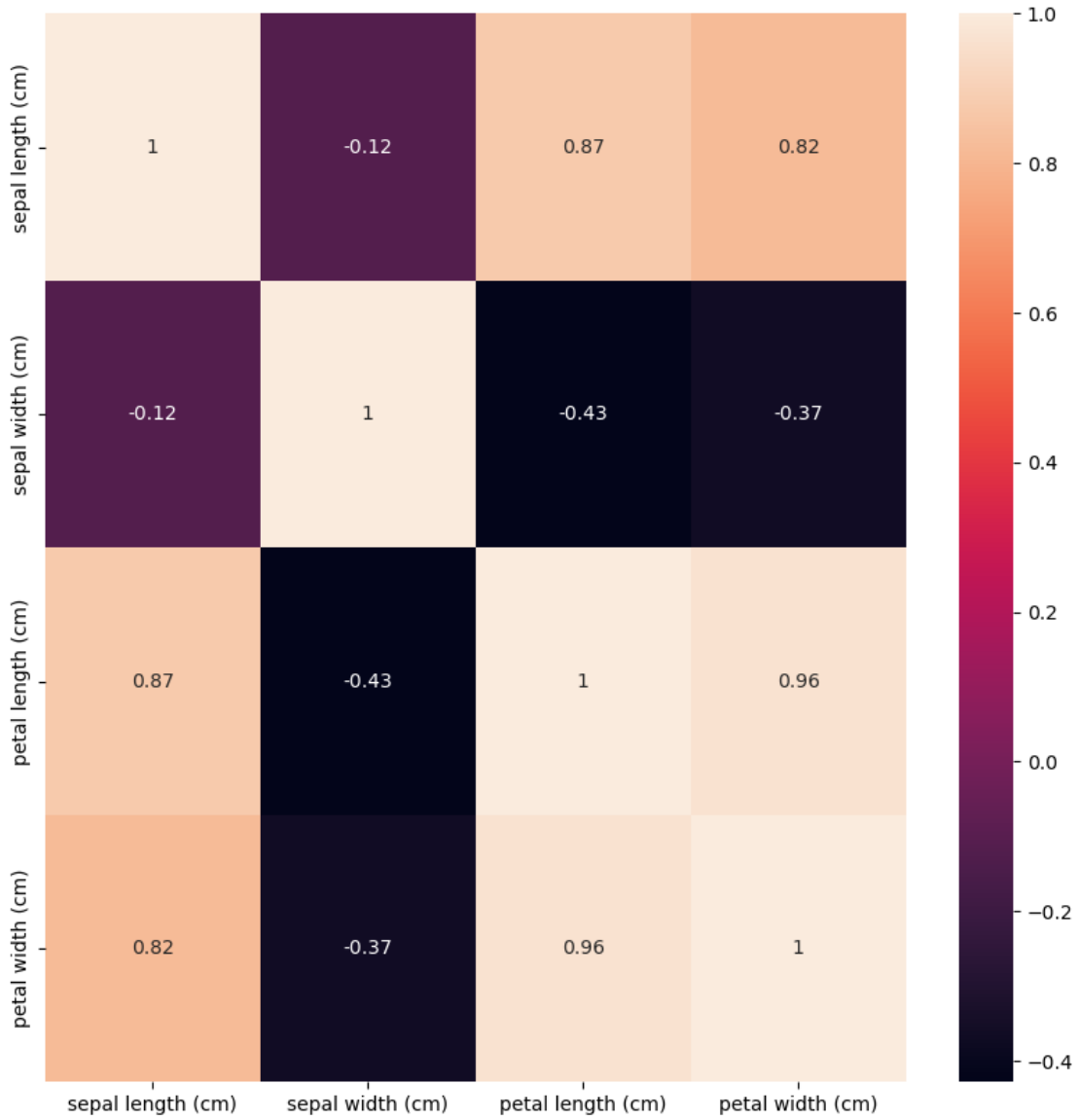
In [122]:
```
i.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

In [123]:
```
cm=i.corr()
```

In [124]:
```
sns.heatmap(data=cm,annot=True)
```

Out[124]:
```
<AxesSubplot:>
```



In [125]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=0)
print('there are {} samples in the trainig set and {} samples in test set'.format(x_train.shape[0],x_test.shape[0]))
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(x_test.shape)
```

```
there are 105 samples in the trainig set and 45 samples in test set
(105, 2)
(45, 2)
(105,)
(45, 2)
```

In [126]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

In [127]:
```
dtc=DecisionTreeClassifier()
model=dtc.fit(x_train,y_train)
ytp=model.predict(x_test)
```

In [128]:
```
pd={"criterion":['gini','entropy'],
   "max_depth":range(1,10),
   "min_samples_split":range(1,10),
   "min_samples_leaf":range(1,5)}
```

In [130]:
```
from sklearn.model_selection import GridSearchCV
g=GridSearchCV(dtc,param_grid=pd,cv=10,verbose=1,n_jobs=-1)
g.fit(x_train,y_train)
```

```
Fitting 10 folds for each of 648 candidates, totalling 6480 fits

C:\Users\srini\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:
720 fits failed out of a total of 6480.
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 error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
720 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\srini\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\srini\anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 937, in fit
    super().fit(
  File "C:\Users\srini\anaconda3\lib\site-packages\sklearn\tree\_classes.py", line 250, in fit
    raise ValueError(
ValueError: min_samples_split must be an integer greater than 1 or a float in (0.0, 1.0]; got the integer 1

  warnings.warn(some_fits_failed_message, FitFailedWarning)
```

In [132]:
```
print("accuracy:",metrics.accuracy_score(y_test,ytp))
```

```
accuracy: 0.9555555555555556
```

In [133]:
```
g.best_params_
```

Out[133]:
```
{'criterion': 'gini',
 'max_depth': 3,
 'min_samples_leaf': 1,
 'min_samples_split': 3}
```

In [134]:
```
g.best_estimator_
```

Out[134]:
```
DecisionTreeClassifier(max_depth=3, min_samples_split=3)
```

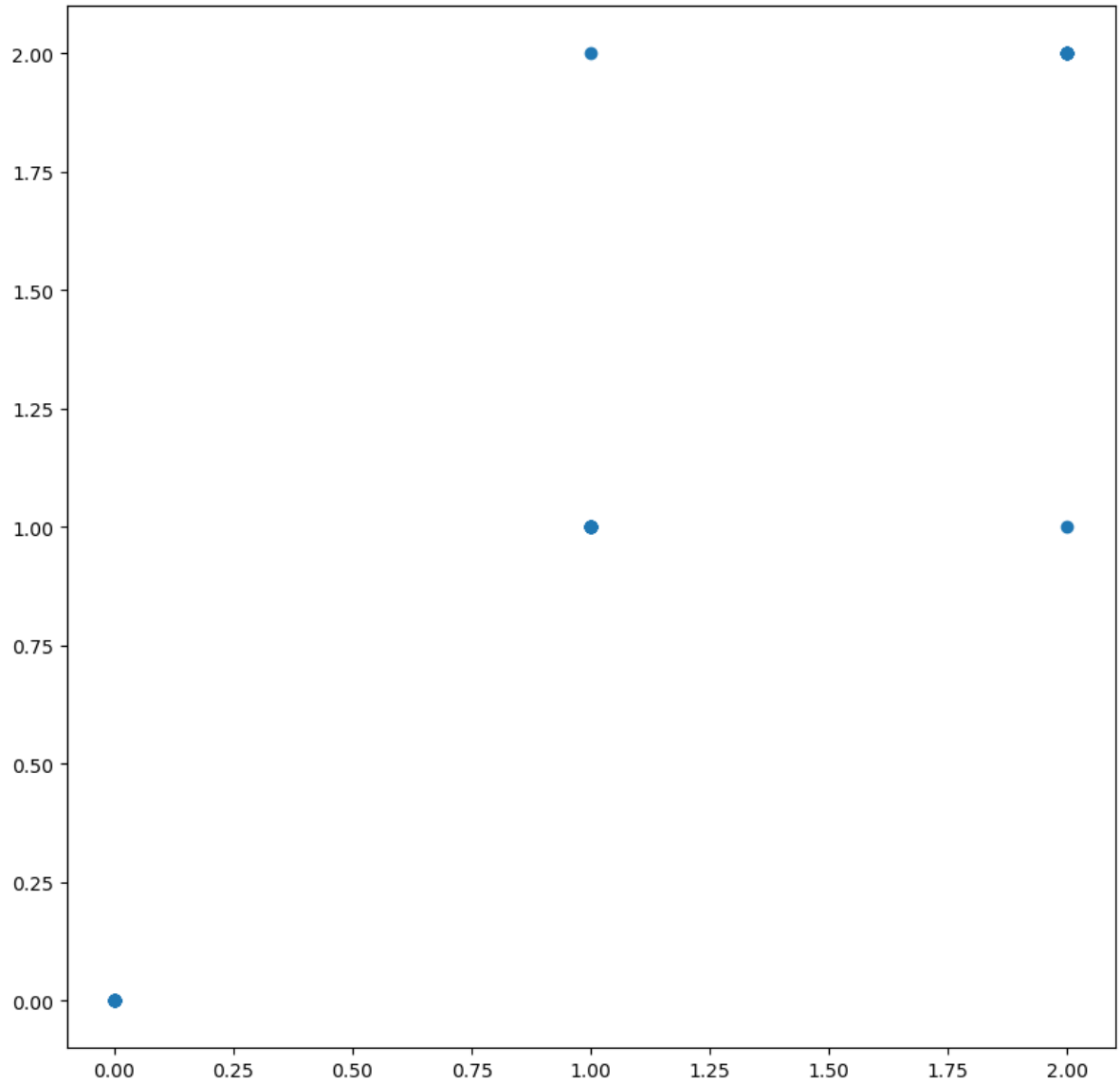
In [135]:
```
g.best_score_
```

Out[135]:
```
0.9800000000000001
```

In [137]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,ytp))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       0.94      0.94      0.94        18
           2       0.91      0.91      0.91        11

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

In [138]:
```
plt.scatter(y_test,ytp)
plt.show()
```



In [139]:
```
import pandas as pd
import missingno as msno
%matplotlib inline
```

In [140]:
```
tdf=pd.read_csv('titanic.csv')
tdf
```

Out[140]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

In [141]:
```
tdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    object
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    object
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
```

In [142]:
```
tdf.isnull()
```

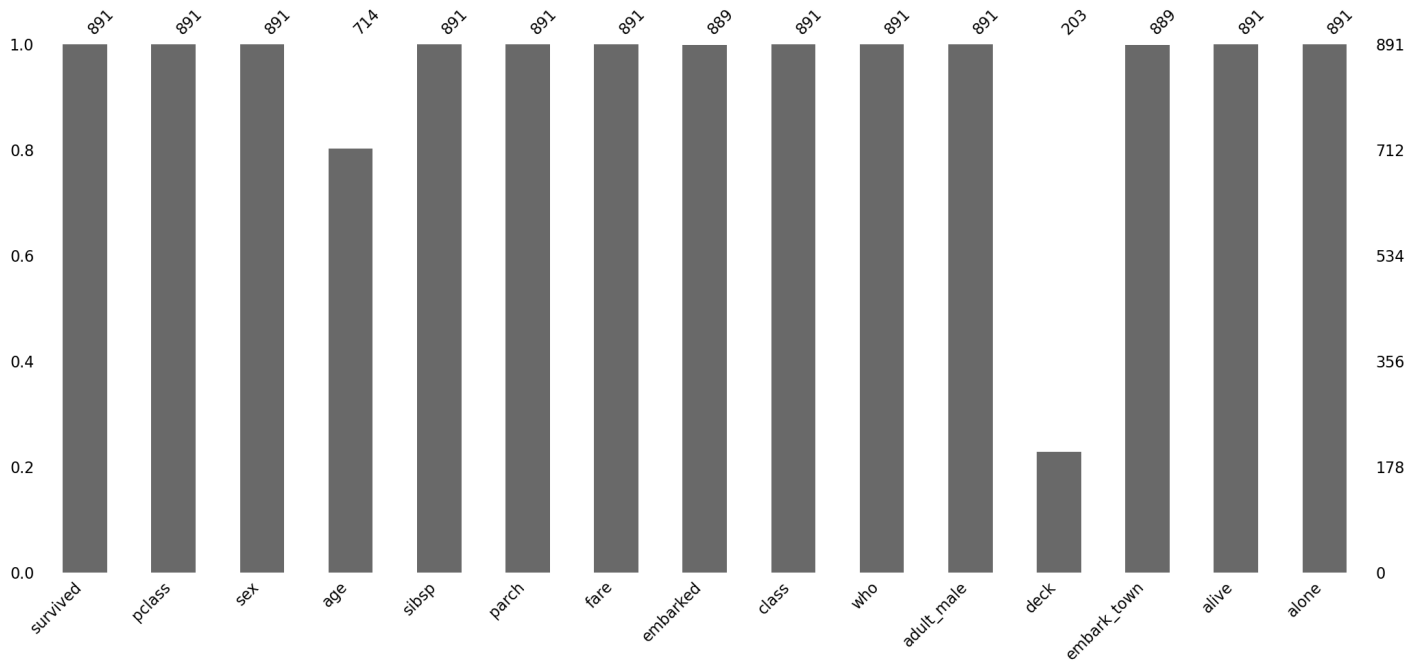
Out[142]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
887	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	True	False	False	False	False	False	False	False	True	False	False	False
889	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False

891 rows × 15 columns

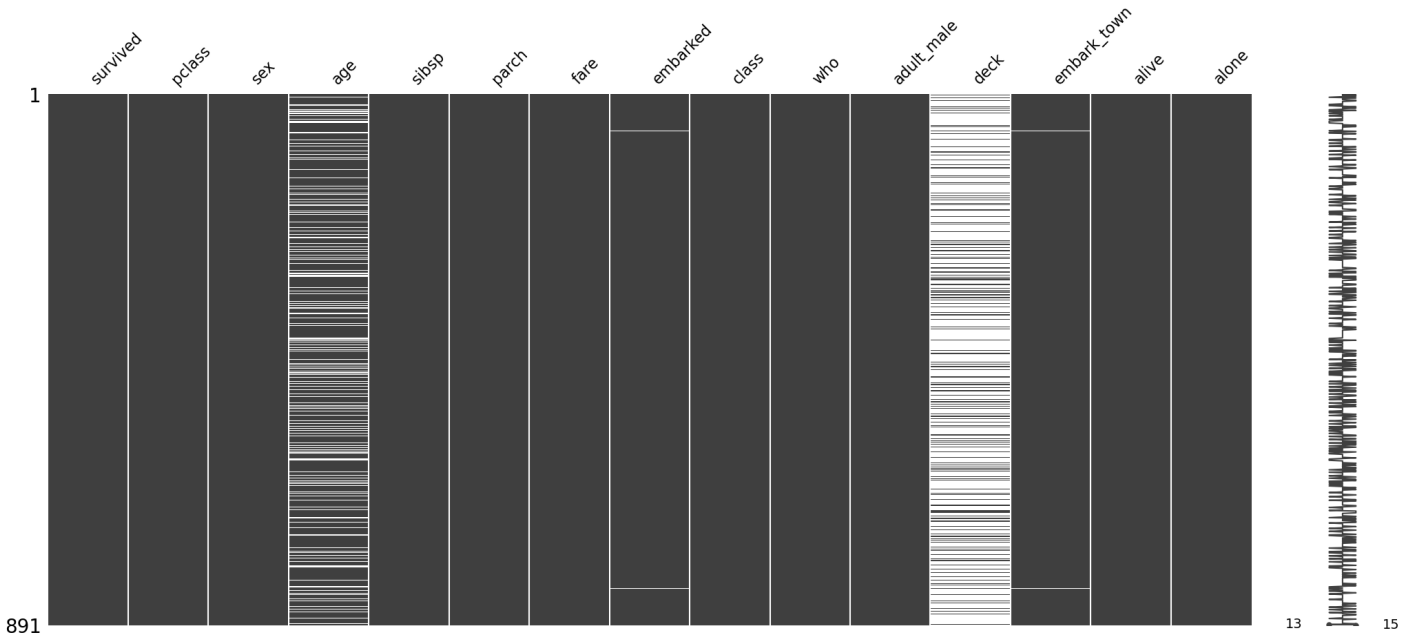
In [143]:
```
msno.bar(tdf)
```

Out[143]:
<AxesSubplot:>



In [144]:
```
msno.matrix(tdf)
```

Out[144]: <AxesSubplot:>



In [145]:
```
tdf.isnull().sum()
```

Out[145]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [147]:
```
df=tdf.dropna(axis=0)
df.isnull().sum()
```

Out[147]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [148]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182 entries, 1 to 889
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     182 non-null    int64
 1   pclass       182 non-null    int64
 2   sex          182 non-null    object
 3   age          182 non-null    float64
 4   sibsp        182 non-null    int64
 5   parch        182 non-null    int64
 6   fare         182 non-null    float64
 7   embarked     182 non-null    object
 8   class        182 non-null    object
 9   who          182 non-null    object
 10  adult_male   182 non-null    bool
 11  deck         182 non-null    object
 12  embark_town  182 non-null    object
 13  alive        182 non-null    object
 14  alone        182 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 20.3+ KB
```

In [150]:
```
tdf.columns
```

Out[150]:
```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [151]:
```
df=tdf.drop(['deck'],axis=1)
df.isnull().sum()
```

Out[151]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
embark_town      2
alive            0
alone            0
dtype: int64
```

In [152]:
```
tdf['deck'].unique()
```

Out[152]:
```
array([nan, 'C', 'E', 'G', 'D', 'A', 'B', 'F'], dtype=object)
```

In [153]:
```
tdf['deck'].isnull().sum()
```

Out[153]:
```
688
```

In [154]:
```
tdf['deck']=tdf['deck'].fillna('C')
```

In [155]:
```
tdf['deck'].isnull().sum()
```

Out[155]:
```
0
```

In [156]:
```
tdf
```

Out[156]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	C	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	C	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	C	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	C	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	C	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	C	Queenstown	no	True

891 rows × 15 columns

In [157]:
```
mean=tdf['age'].mean()
print(mean)
tdf['age']=tdf['age'].fillna(mean)
tdf['age']
```

```
29.69911764705882
```

Out[157]:
```
0      22.000000
1      38.000000
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: age, Length: 891, dtype: float64
```

In [158]:
```
mode=tdf['deck'].mode()[0]
print(mode)
tdf['deck']=tdf['deck'].fillna(mode)
```

```
C
```

In [159]:
```
tdf['deck']
```

Out[159]:
```
0      C
1      C
2      C
3      C
4      C
      ..
886    C
887    B
888    C
889    C
890    C
Name: deck, Length: 891, dtype: object
```

In [160]:
```
tdf['age']=tdf['age'].fillna(tdf['age'].median())
tdf['age']
```

Out[160]:
```
0      22.000000
1      38.000000
2      26.000000
3      35.000000
4      35.000000
         ...
886    27.000000
887    19.000000
888    29.699118
889    26.000000
890    32.000000
Name: age, Length: 891, dtype: float64
```

In [161]:
```
tdf=pd.read_csv('titanic.csv')
tdf
```

Out[161]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

In [162]:
```
ndf=tdf.fillna(method='ffill')
ndf
```

Out[162]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	C	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	C	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	C	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	19.0	1	2	23.4500	S	Third	woman	False	B	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	C	Queenstown	no	True

891 rows × 15 columns

In [169]:
```
ndf=tdf.fillna(method='bfill')
ndf
```

Out[169]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	C	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	C	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	E	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	B	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	26.0	1	2	23.4500	S	Third	woman	False	C	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

In [170]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

In [171]:
```
cdf=pd.read_csv('credit_dataset.csv')
cdf
```

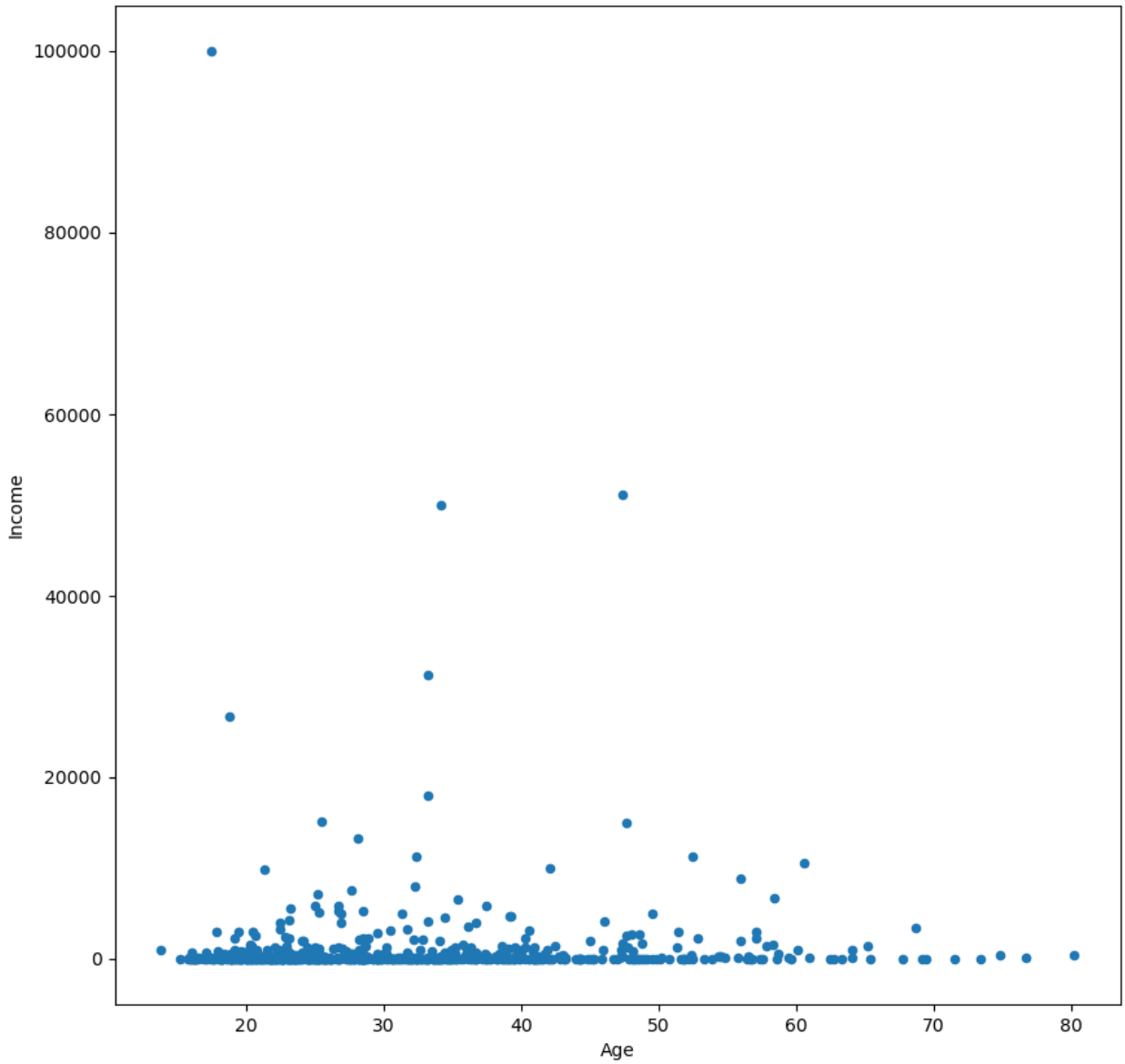
Out[171]:

	Gender	Age	Debt	Married	BankCustomer	Industry	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore	DriversLicense	Citi
0	1	30.83	0.000	1	1	Industrials	White	1.25	1	1	1	0	ByE
1	0	58.67	4.460	1	1	Materials	Black	3.04	1	1	6	0	ByE
2	0	24.50	0.500	1	1	Materials	Black	1.50	1	0	0	0	ByE
3	1	27.83	1.540	1	1	Industrials	White	3.75	1	1	5	1	ByE
4	1	20.17	5.625	1	1	Industrials	White	1.71	1	0	0	0	ByOtherMe
...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	1	21.08	10.085	0	0	Education	Black	1.25	0	0	0	0	ByE
686	0	22.67	0.750	1	1	Energy	White	2.00	0	1	2	1	ByE
687	0	25.25	13.500	0	0	Healthcare	Latino	2.00	0	1	1	1	ByE
688	1	17.92	0.205	1	1	ConsumerStaples	White	0.04	0	0	0	0	ByE
689	1	35.00	3.375	1	1	Energy	Black	8.29	0	0	0	1	ByE

690 rows × 16 columns

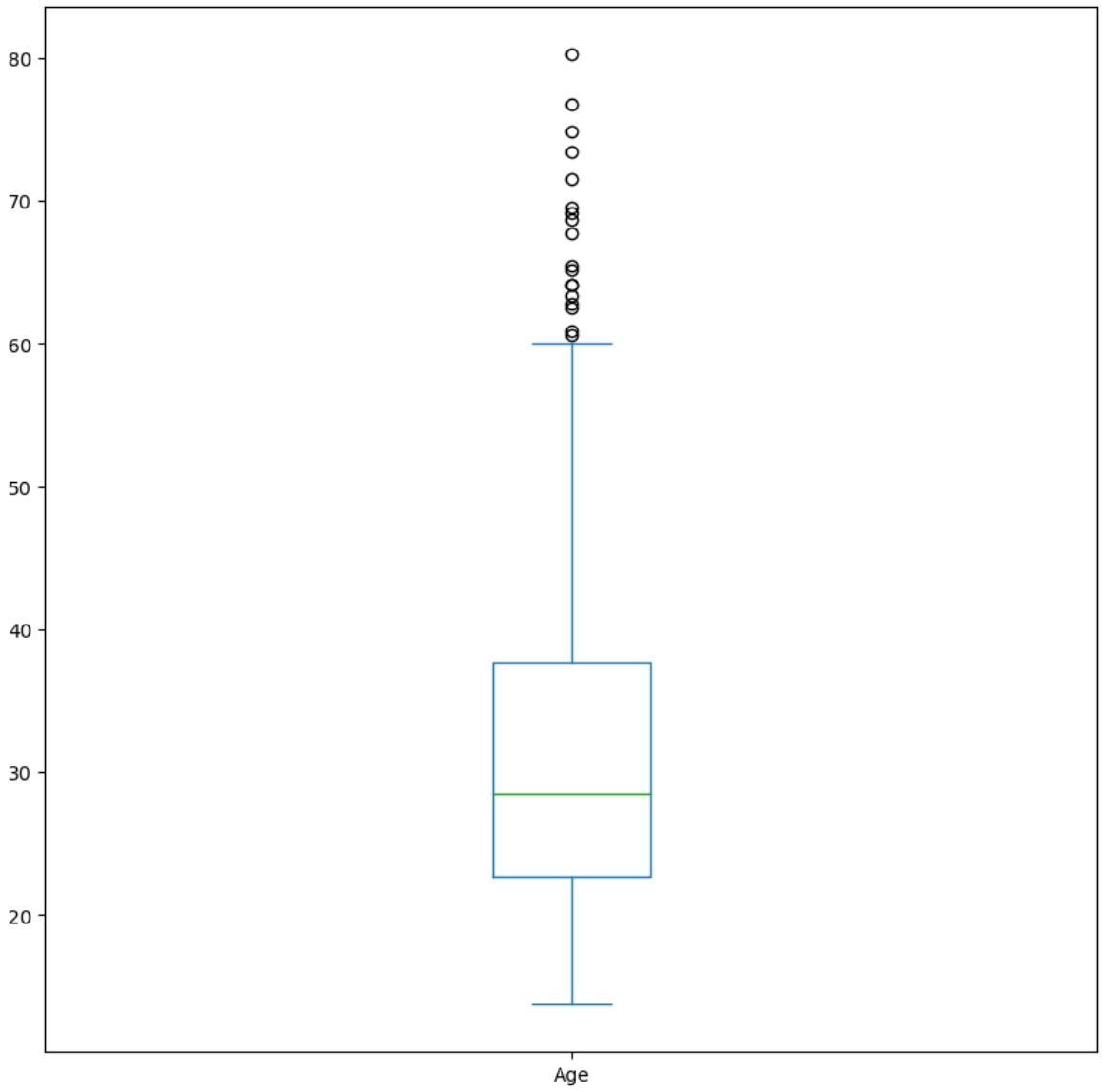
In [172]:
```
cdf.plot('Age','Income',kind='scatter',marker='o')
```

Out[172]: <AxesSubplot:xlabel='Age', ylabel='Income'>



In [186]:
```
cdf['Age'].plot(kind='box')
```

Out[186]: <AxesSubplot:>

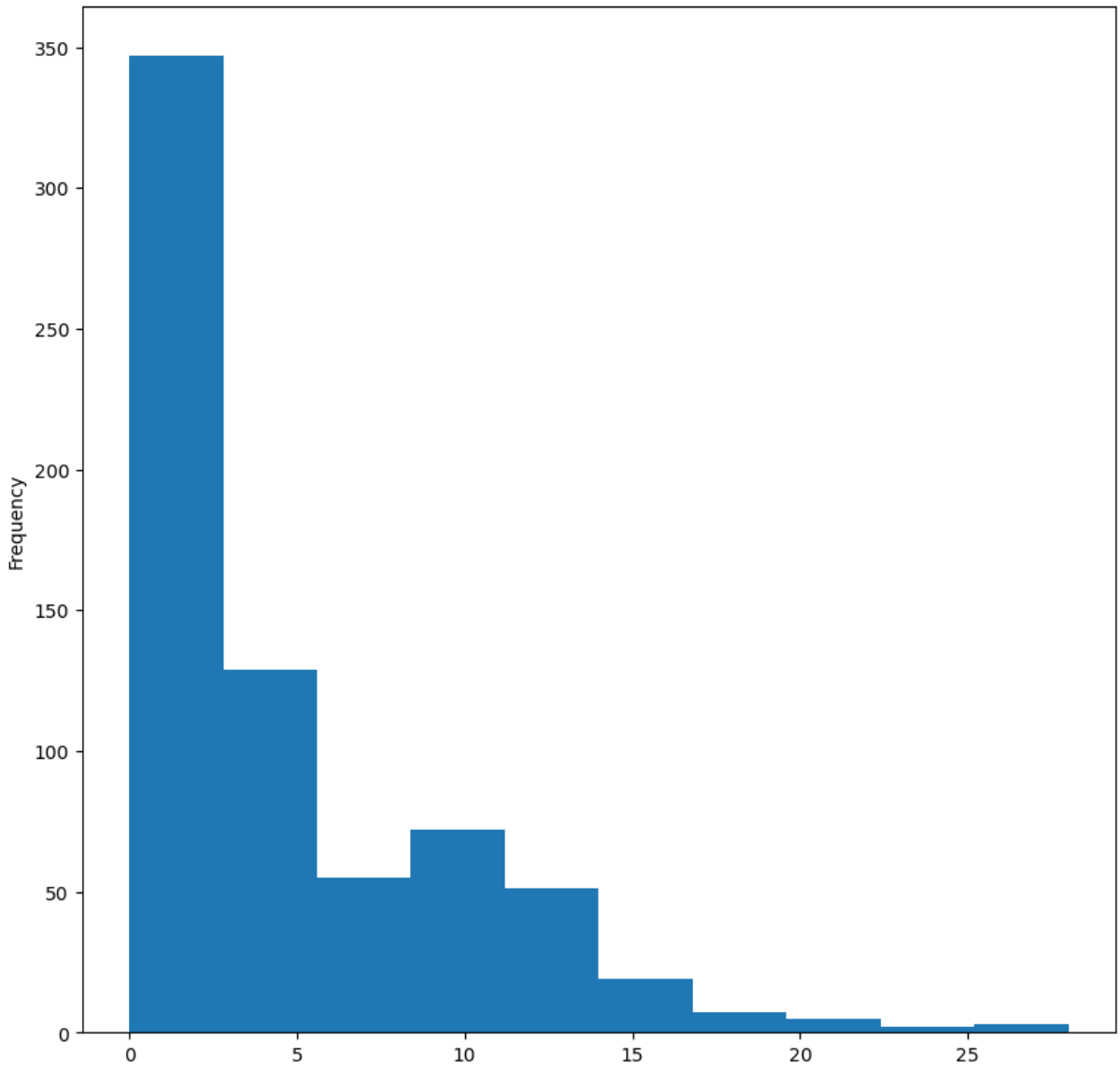


In [177]:
```
q1=np.percentile(cdf['Age'],25)
q3=np.percentile(cdf['Age'],75)
q1,q3=np.percentile(cdf['Age'],[25,75])
iqr=q3-q1
ul=q3+1.5*iqr
ll=q3-1.5*iqr
out=cdf['Age'][(cdf['Age']>ul)|(cdf['Age']<ll)]
print(out)
```

```
90     62.50
130    67.75
157    68.67
164    60.58
206    71.58
221    65.42
296    69.17
345    62.75
348    63.33
405    69.50
485    74.83
502    64.08
510    13.75
516    60.92
539    80.25
550    76.75
573    65.17
585    73.42
586    64.08
Name: Age, dtype: float64
```

In [174]:
```
cdf['Debt'].plot(kind='hist')
```

Out[174]: <AxesSubplot:ylabel='Frequency'>



In [198]:
```
data=cdf['Age']
mean=np.mean(data)
std=np.std(data)
print("mean of the dataset is:",mean)
print("std. of dataset is:",std)
threshold=3
out=[]
for i in data:
    z=(i-mean)/std
    if z>threshold:
        out.append(i)
print("outlier in dataset of z score is:",out)
```

```
mean of the dataset is: 31.514115942029015
std. of dataset is: 11.851647259734232
outlier in dataset of z score is: [67.75, 68.67, 71.58, 69.17, 69.5, 74.83, 80.25, 76.75, 73.42]
```

In [199]:
```
q1=data.quantile(0.25)
q3=data.quantile(0.75)
iqr=q3-q1
ub=q3+(1.5*iqr)
lb=q1-(1.5*iqr)
```

In [201]:
```
ui=cdf[data>ub].index
cdf.drop(ui,inplace=True)
li=cdf[data<lb].index
cdf.drop(li,inplace=True)
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 672 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          672 non-null    int64
 1   Age             672 non-null    float64
 2   Debt            672 non-null    float64
 3   Married         672 non-null    int64
 4   BankCustomer    672 non-null    int64
 5   Industry        672 non-null    object
 6   Ethnicity       672 non-null    object
 7   YearsEmployed   672 non-null    float64
 8   PriorDefault    672 non-null    int64
 9   Employed        672 non-null    int64
 10  CreditScore     672 non-null    int64
 11  DriversLicense  672 non-null    int64
 12  Citizen         672 non-null    object
 13  ZipCode         672 non-null    int64
 14  Income          672 non-null    int64
 15  Approved        672 non-null    int64
dtypes: float64(3), int64(10), object(3)
memory usage: 89.2+ KB

C:\Users\srini\AppData\Local\Temp\ipykernel_14272\292259696.py:1: UserWarning: Boolean Series key will be reindexed to match Dat
aFrame index.
  ui=cdf[data>ub].index
C:\Users\srini\AppData\Local\Temp\ipykernel_14272\292259696.py:3: UserWarning: Boolean Series key will be reindexed to match Dat
aFrame index.
  li=cdf[data<lb].index
```

In [203]:
```
m=np.mean(cdf['Age'])
print("mean:",m)
for i in cdf['Age']:
    if i <lb or i>ub:
        cdf['Age']=cdf['Age'].replace(i,m)
```

```
mean: 30.54165178571431
```

In [204]:
```
q1=np.percentile(cdf['Age'],25)
q3=np.percentile(cdf['Age'],75)
q1,q3=np.percentile(cdf['Age'],[25,75])
iqr=q3-q1
ul=q3+1.5*iqr
ll=q3-1.5*iqr
out=cdf['Age'][(cdf['Age']>ul)|(cdf['Age']<ll)]
print(out)
```

```
1      58.67
118    57.83
151    58.33
212    60.08
234    58.42
268    59.67
390    15.17
434    58.58
510    13.75
570    59.50
Name: Age, dtype: float64
```

In [206]:
```
data=cdf['Age']
mean=np.mean(data)
std=np.std(data)
print("mean of the dataset is:",mean)
print("std. of dataset is:",std)
threshold=2
out=[]
for i in data:
    z=(i-mean)/std
    if z>threshold:
        out.append(i)
print("outlier in dataset of z score is:",out)
```

```
mean of the dataset is: 30.54165178571431
std. of dataset is: 10.351584798333064
outlier in dataset of z score is: [58.67, 54.42, 56.58, 57.42, 54.58, 56.42, 54.33, 56.75, 52.33, 54.83, 52.5, 57.83, 56.5, 52.8
3, 58.33, 56.0, 57.08, 60.08, 55.92, 53.92, 58.42, 52.17, 59.67, 57.58, 56.83, 51.92, 51.83, 58.58, 53.33, 52.42, 57.08, 55.75,
51.33, 51.42, 59.5, 51.58, 52.5, 51.83]
```

In [207]:
```
m=cdf['Age'].median()
print("median:",m)
for i in cdf['Age']:
    if i<lb or i>ub:
        cdf['Age']=cdf['Age'].replace(i,m)
```

```
median: 28.25
```

In [208]:
```
for i in cdf['Age']:
    if i<lb or i>ub:
        cdf['Age']=cdf['Age'].replace(i,0)
```

In [209]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import math
```

In [211]:
```
cpdf=pd.read_csv('clean_dataset.csv')
print(cpdf.head())
```

```
   Gender    Age   Debt  Married  BankCustomer     Industry Ethnicity  \
0       1  30.83  0.000        1             1  Industrials     White
1       0  58.67  4.460        1             1    Materials     Black
2       0  24.50  0.500        1             1    Materials     Black
3       1  27.83  1.540        1             1  Industrials     White
4       1  20.17  5.625        1             1  Industrials     White

   YearsEmployed  PriorDefault  Employed  CreditScore  DriversLicense  \
0           1.25             1         1            1               0
1           3.04             1         1            6               0
2           1.50             1         0            0               0
3           3.75             1         1            5               1
4           1.71             1         0            0               0

        Citizen  ZipCode  Income  Approved
0       ByBirth      202       0         1
1       ByBirth       43     560         1
2       ByBirth      280     824         1
3       ByBirth      100       3         1
4  ByOtherMeans      120       0         1
```

In [212]:
```
cpdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Gender          690 non-null    int64
 1   Age             690 non-null    float64
 2   Debt            690 non-null    float64
 3   Married         690 non-null    int64
 4   BankCustomer    690 non-null    int64
 5   Industry        690 non-null    object
 6   Ethnicity       690 non-null    object
 7   YearsEmployed   690 non-null    float64
 8   PriorDefault    690 non-null    int64
 9   Employed        690 non-null    int64
 10  CreditScore     690 non-null    int64
 11  DriversLicense  690 non-null    int64
 12  Citizen         690 non-null    object
 13  ZipCode         690 non-null    int64
 14  Income          690 non-null    int64
 15  Approved        690 non-null    int64
dtypes: float64(3), int64(10), object(3)
memory usage: 86.4+ KB
```

In [213]:
```
cpdf.duplicated().sum()
```

Out[213]:
```
0
```

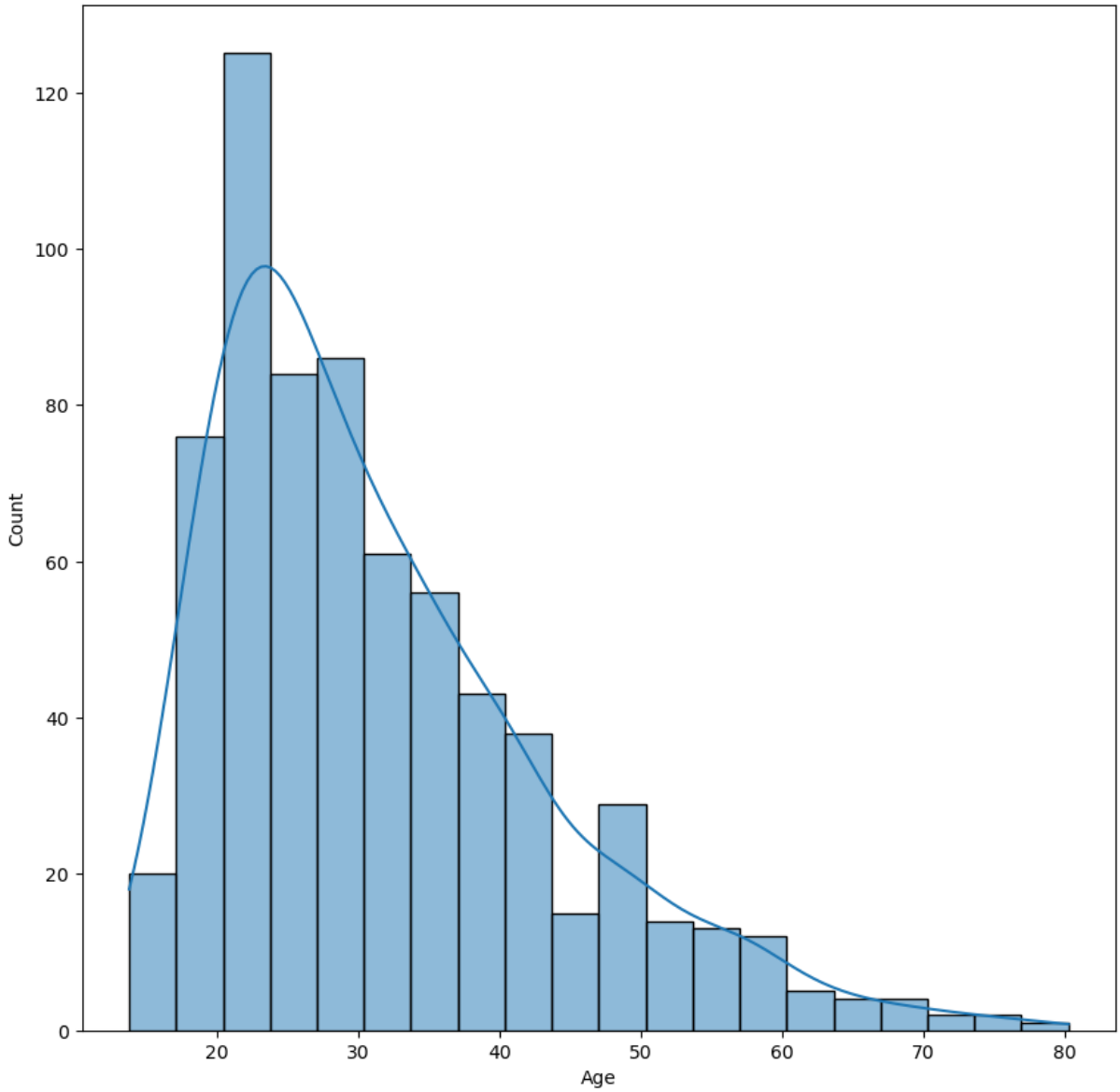
In [214]:
```
cpdf[['Age','Debt','YearsEmployed','CreditScore',"Income"]].describe()
```

Out[214]:

	Age	Debt	YearsEmployed	CreditScore	Income
count	690.000000	690.000000	690.000000	690.00000	690.000000
mean	31.514116	4.758725	2.223406	2.40000	1017.385507
std	11.860245	4.978163	3.346513	4.86294	5210.102598
min	13.750000	0.000000	0.000000	0.00000	0.000000
25%	22.670000	1.000000	0.165000	0.00000	0.000000
50%	28.460000	2.750000	1.000000	0.00000	5.000000
75%	37.707500	7.207500	2.625000	3.00000	395.500000
max	80.250000	28.000000	28.500000	67.00000	100000.000000

In [217]:
```
sns.histplot(cpdf.Age,kde=True)
```

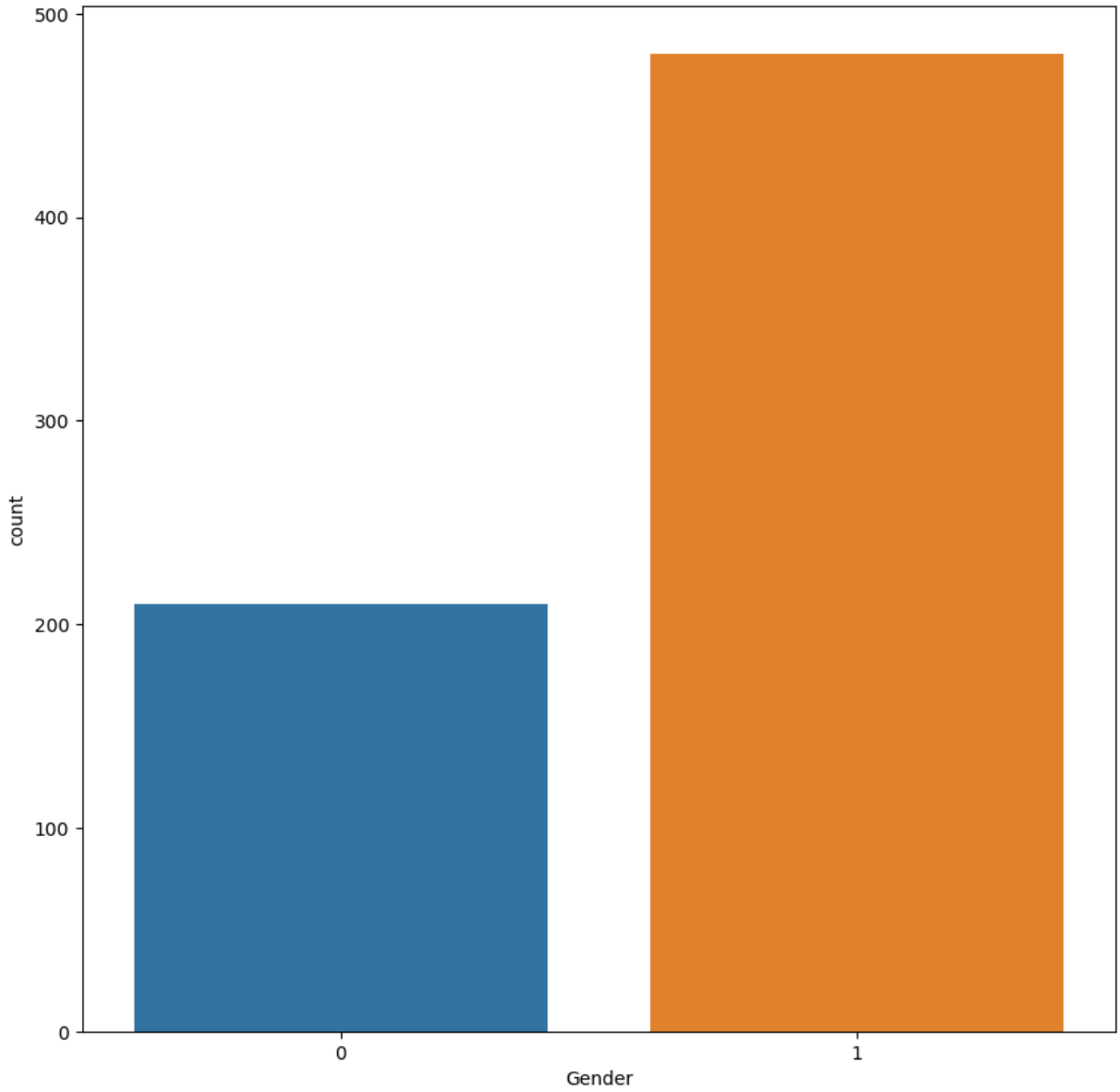
Out[217]:
```
<AxesSubplot:xlabel='Age', ylabel='Count'>
```



In [220]:
```
sns.countplot(cpdf.Gender)
```

```
C:\Users\srini\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
  warnings.warn(
```

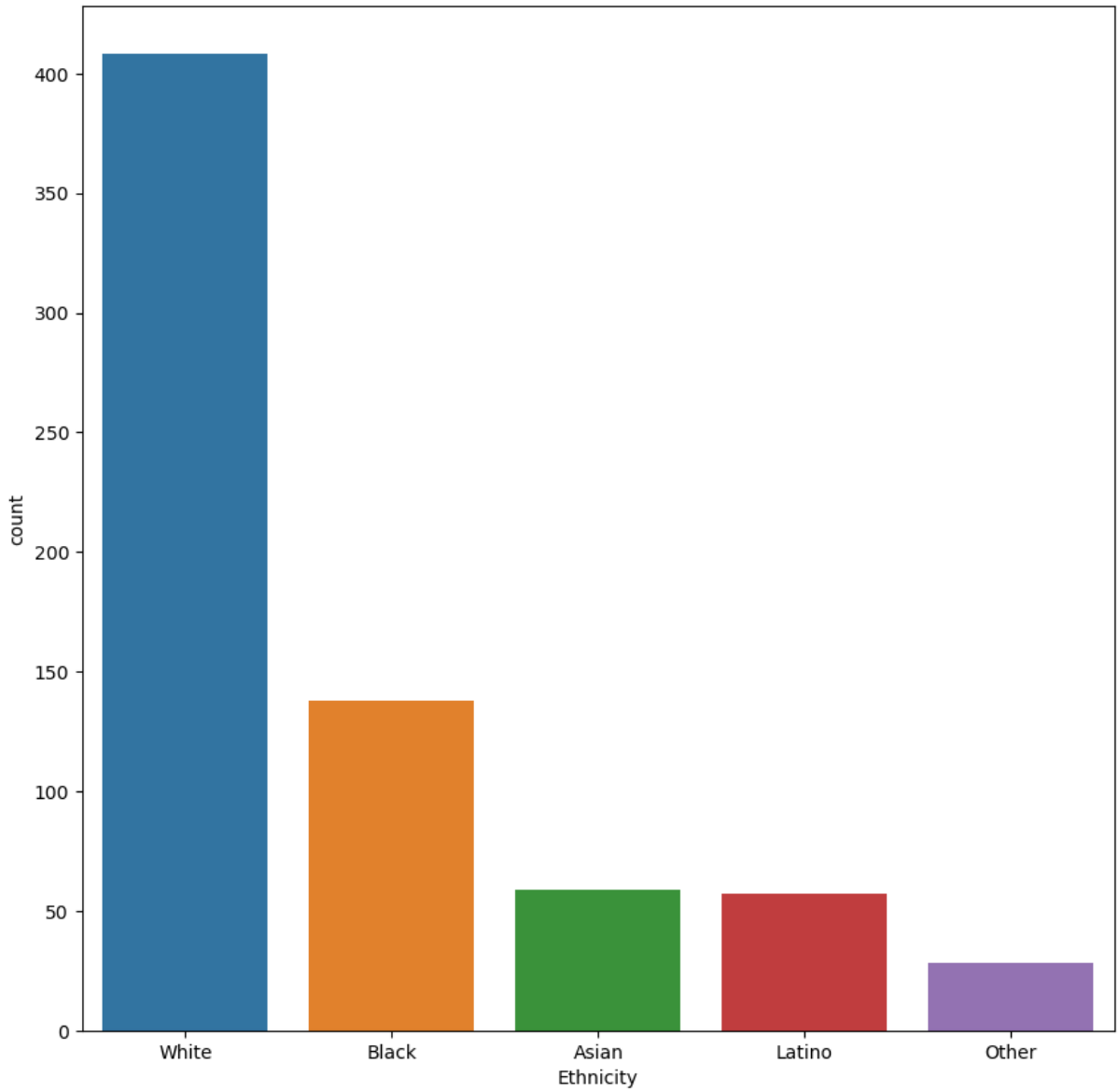
Out[220]: <AxesSubplot:xlabel='Gender', ylabel='count'>



In [221]:
```
sns.countplot(cpdf.Ethnicity)
```

```
C:\Users\srini\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
  warnings.warn(
```

Out[221]: <AxesSubplot:xlabel='Ethnicity', ylabel='count'>



In [222]:
```
cpdf[['Age','Debt','YearsEmployed','CreditScore',"Income"]].corr()
```

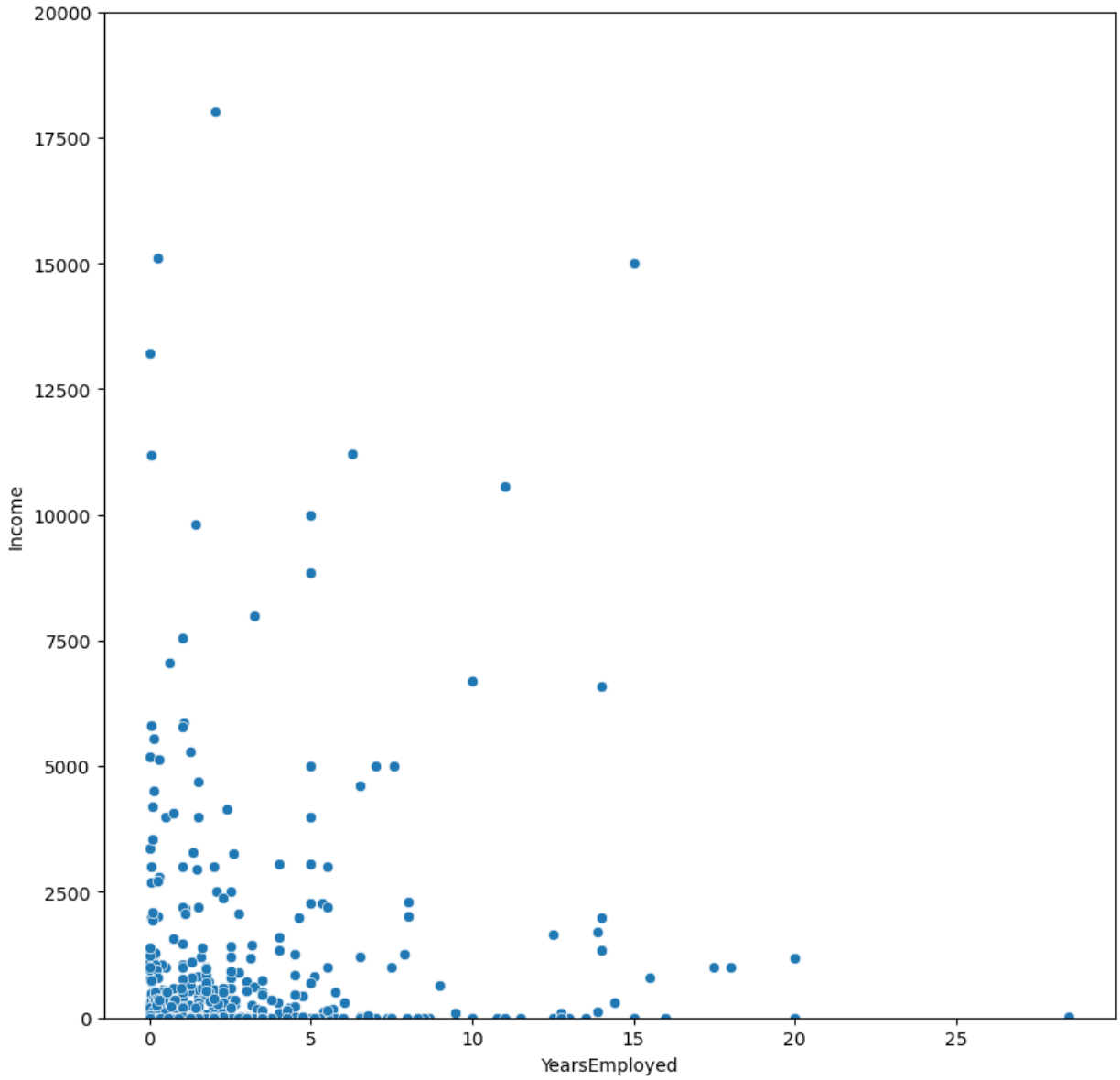
Out[222]:

	Age	Debt	YearsEmployed	CreditScore	Income
Age	1.000000	0.202177	0.391464	0.187327	0.018719
Debt	0.202177	1.000000	0.298902	0.271207	0.123121
YearsEmployed	0.391464	0.298902	1.000000	0.322330	0.051345
CreditScore	0.187327	0.271207	0.322330	1.000000	0.063692
Income	0.018719	0.123121	0.051345	0.063692	1.000000

In [224]:
```
sns.scatterplot(cpdf.YearsEmployed,cpdf.Income)
plt.ylim(0,20000)
```

```
C:\Users\srini\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword arg
s: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[224]: (0.0, 20000.0)



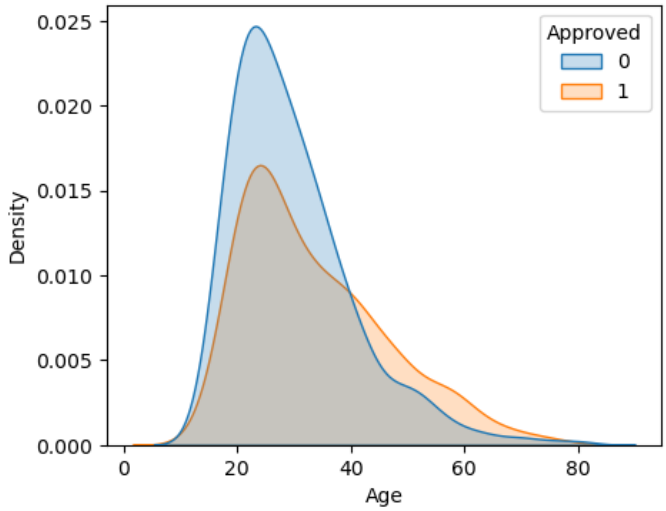
In [227]:
```
cpdf.groupby(by='Approved').agg('mean')[['Age','Debt','YearsEmployed','CreditScore','Income']]
```

Out[227]:

Approved	Age	Debt	YearsEmployed	CreditScore	Income
0	29.773029	3.839948	1.257924	0.631854	198.605744
1	33.686221	5.904951	3.427899	4.605863	2038.859935

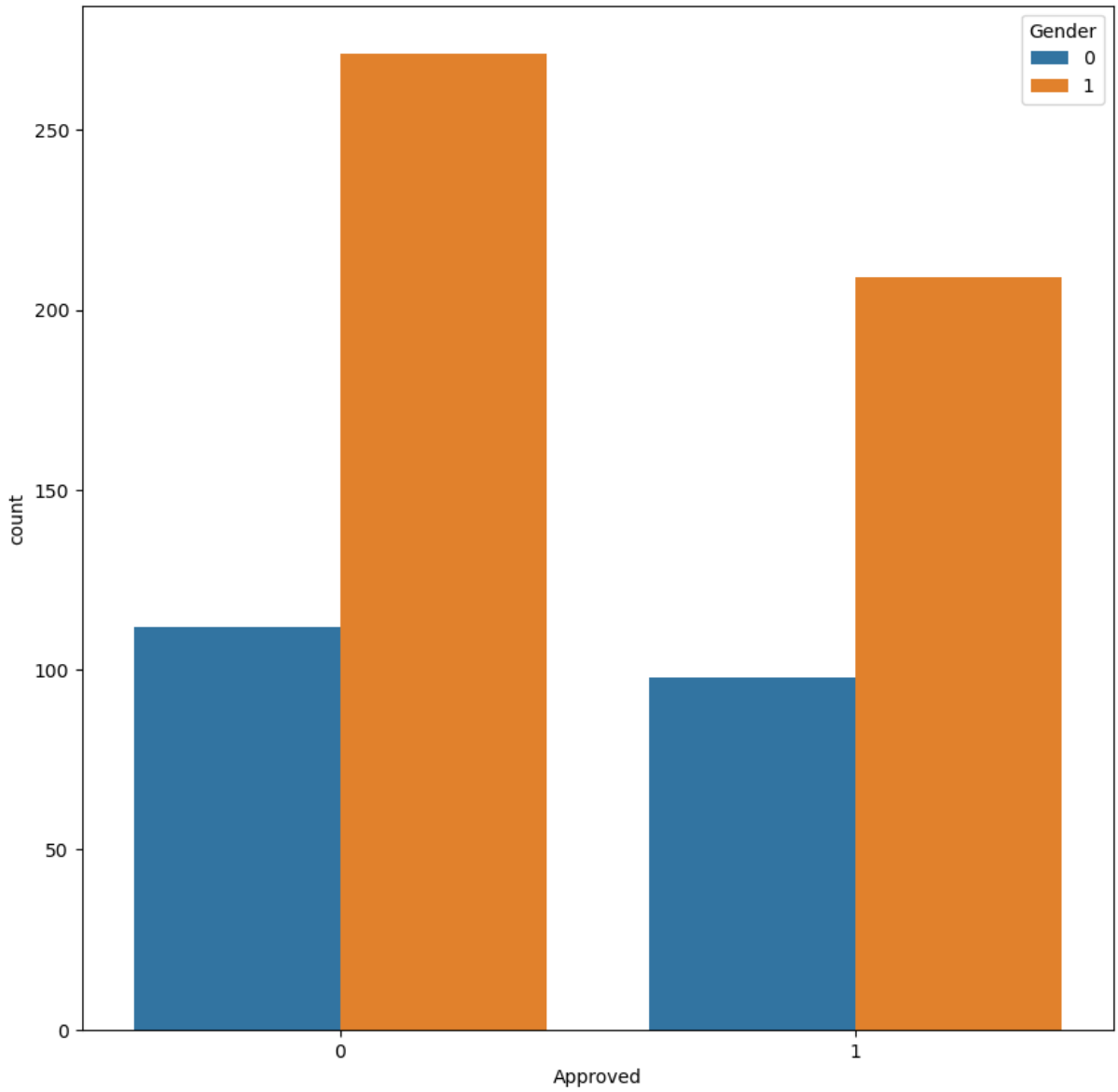
In [228]:
```
plt.figure(figsize=(5,4))
sns.kdeplot(data=cpdf,x='Age',hue='Approved',fill=True)
```

Out[228]: <AxesSubplot:xlabel='Age', ylabel='Density'>



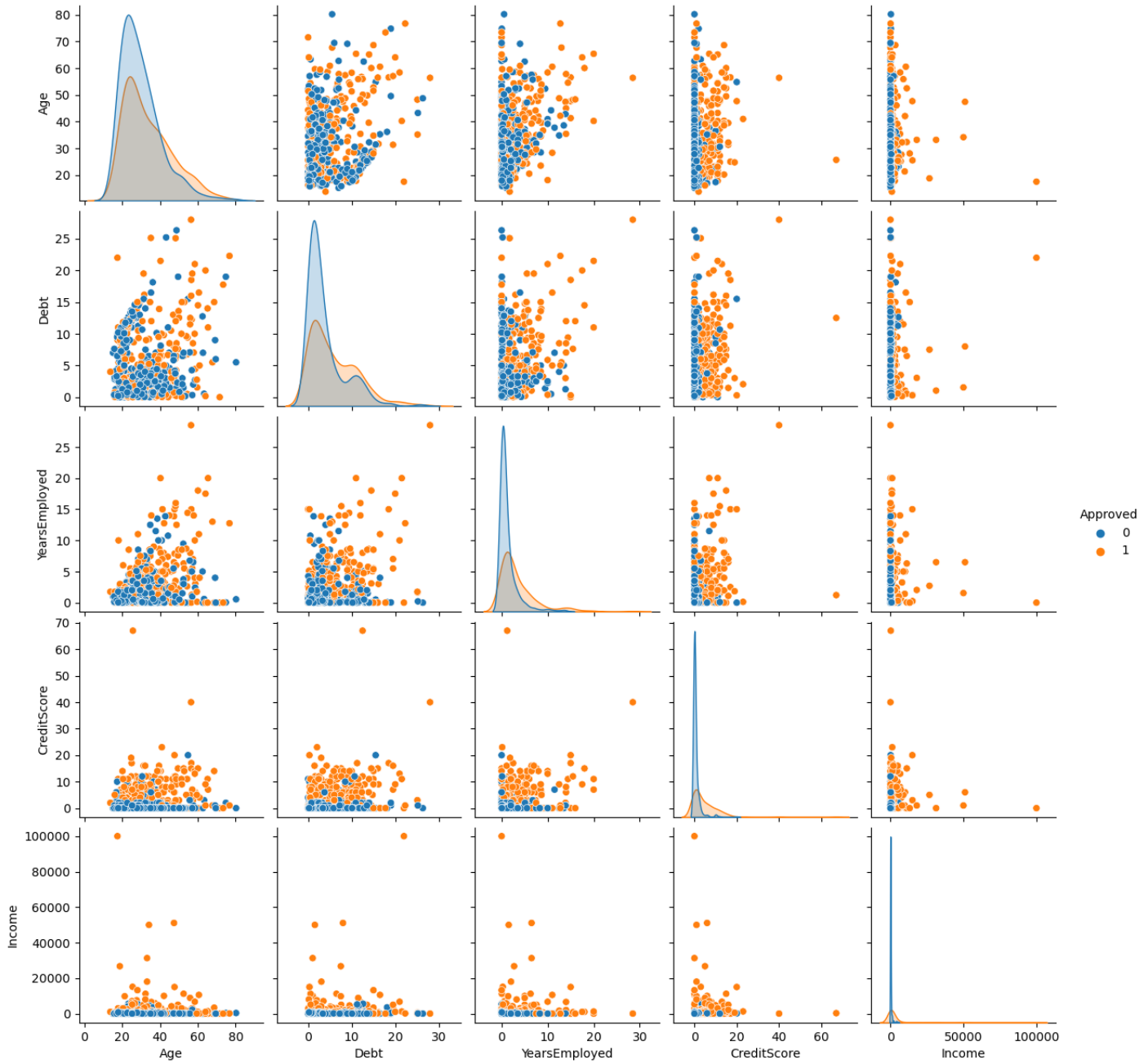
In [229]:
```
sns.countplot(data=cpdf,x='Approved',hue='Gender')
```

Out[229]: <AxesSubplot:xlabel='Approved', ylabel='count'>



In [233]:
```
sns.pairplot(data=cpdf[['Age','Debt','YearsEmployed','CreditScore','Income','Approved']],hue='Approved')
```

Out[233]:
```
<seaborn.axisgrid.PairGrid at 0x23acf1fde50>
```



In [236]:
```
import pandas as pd
import numpy as np
df=pd.read_csv('rainfall (2).csv')
```

In [237]:
```
df.head()
```

Out[237]:

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	358.5	295.5	285.0	271.9	354.8	326.0	315.2	250.9	2805.2	165.2	540.7	1207.2	892.1
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	374.4	457.2	421.3	423.1	455.6	301.2	275.8	128.3	3015.7	69.7	483.5	1757.2	705.3
2	ANDAMAN And NICOBAR ISLANDS	N & M ANDAMAN	32.7	15.9	8.6	53.4	343.6	503.3	465.4	460.9	454.8	276.1	198.6	100.0	2913.3	48.6	405.6	1884.4	574.7
3	ARUNACHAL PRADESH	LOHIT	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	3043.8	123.0	841.3	1848.5	231.0
4	ARUNACHAL PRADESH	EAST SIANG	33.3	79.5	105.9	216.5	323.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7	4034.7	112.8	645.4	3008.4	268.1

In [238]:
```
sdf=df.sort_values(by='ANNUAL',ascending=False)
h=sdf.iloc[0,1]
print("District taht gets highest annual rainfall:",h)
```

```
District taht gets highest annual rainfall: TAMENGLONG
```

In [239]:
```
ndf=df.drop(['Jan-Feb','Mar-May','Jun-Sep','Oct-Dec'],axis=1)
ndf
```

Out[239]:

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	358.5	295.5	285.0	271.9	354.8	326.0	315.2	250.9	2805.2
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	374.4	457.2	421.3	423.1	455.6	301.2	275.8	128.3	3015.7
2	ANDAMAN And NICOBAR ISLANDS	N & M ANDAMAN	32.7	15.9	8.6	53.4	343.6	503.3	465.4	460.9	454.8	276.1	198.6	100.0	2913.3
3	ARUNACHAL PRADESH	LOHIT	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	3043.8
4	ARUNACHAL PRADESH	EAST SIANG	33.3	79.5	105.9	216.5	323.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7	4034.7
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
636	KERALA	IDUKKI	13.4	22.1	43.6	150.4	232.6	651.6	788.9	527.3	308.4	343.2	172.9	48.1	3302.5
637	KERALA	KASARGOD	2.3	1.0	8.4	46.9	217.6	999.6	1108.5	636.3	263.1	234.9	84.6	18.4	3621.6
638	KERALA	PATHANAMTHITTA	19.8	45.2	73.9	184.9	294.7	556.9	539.9	352.7	266.2	359.4	213.5	51.3	2958.4
639	KERALA	WAYANAD	4.8	8.3	17.5	83.3	174.6	698.1	1110.4	592.9	230.7	213.1	93.6	25.8	3253.1
640	LAKSHADWEEP	LAKSHADWEEP	20.8	14.7	11.8	48.9	171.7	330.2	287.7	217.5	163.1	157.1	117.7	58.8	1600.0

641 rows × 15 columns

In [240]:
```
ndf=ndf.drop(['ANNUAL'],axis=1)
table=pd.pivot_table(ndf,index=['STATE_UT_NAME'])
table
```

Out[240]:

STATE_UT_NAME	APR	AUG	DEC	FEB	JAN	JUL	JUN	MAR	MAY	NOV	OCT	SE
ANDAMAN And NICOBAR ISLANDS	86.966667	385.300000	159.733333	33.266667	61.233333	390.566667	418.666667	30.800000	358.833333	263.200000	301.100000	421.7333
ANDHRA PRADESH	19.873913	179.426087	15.565217	7.352174	6.321739	185.365217	114.369565	10.095652	48.765217	58.965217	138.600000	160.3739
ARUNACHAL PRADESH	275.162500	378.600000	35.956250	93.293750	53.687500	547.581250	491.381250	165.018750	300.262500	43.187500	176.768750	366.4750
ASSAM	181.266667	377.370370	11.440741	31.714815	15.733333	494.844444	465.185185	77.762963	333.870370	24.922222	136.448148	303.8000
BIHAR	16.865789	289.481579	5.786842	9.278947	13.134211	340.836842	168.781579	9.873684	51.673684	6.715789	64.747368	223.3789
CHANDIGARH	14.800000	287.500000	23.400000	38.900000	44.300000	282.400000	120.000000	33.200000	30.100000	9.900000	31.800000	154.3000
CHATISGARH	13.116667	375.338889	5.811111	10.472222	10.377778	375.405556	180.583333	12.977778	17.483333	8.494444	61.844444	214.4444
DADAR NAGAR HAVELI	0.000000	655.900000	0.000000	0.300000	0.400000	884.500000	385.100000	0.000000	7.400000	10.500000	38.600000	391.4000
DAMAN AND DUI	0.100000	394.600000	0.450000	0.500000	0.550000	583.100000	276.500000	0.200000	4.150000	12.400000	35.550000	227.6000
DELHI	8.900000	245.500000	8.600000	16.300000	16.400000	220.700000	59.800000	15.300000	19.300000	5.600000	20.500000	110.2000
GOA	7.800000	683.800000	10.200000	0.050000	0.550000	1108.100000	908.100000	0.550000	87.750000	35.000000	155.700000	280.9000
GUJARAT	0.507692	257.630769	1.592308	0.392308	0.784615	333.838462	139.246154	1.142308	4.803846	10.826923	25.103846	148.4730
HARYANA	7.619048	190.909524	7.914286	16.457143	19.485714	180.361905	51.009524	13.738095	14.642857	5.266667	18.428571	88.7238
HIMACHAL	47.683333	322.325000	38.225000	80.450000	81.925000	343.825000	108.683333	87.633333	54.358333	16.908333	39.308333	150.2666
JAMMU AND KASHMIR	82.268182	167.918182	46.395455	91.645455	77.977273	172.090909	53.604545	119.986364	65.136364	27.159091	34.181818	78.2545
JHARKHAND	18.662500	310.316667	6.704167	16.320833	15.837500	333.854167	198.775000	16.516667	45.875000	10.212500	79.404167	250.9583
KARNATAKA	36.773333	209.256667	11.170000	2.696667	2.026667	280.700000	204.880000	7.163333	88.166667	44.350000	143.356667	164.0766
KERALA	109.021429	417.950000	38.242857	16.200000	9.542857	724.328571	658.707143	31.071429	244.728571	151.535714	290.907143	245.1571
LAKSHADWEEP	48.900000	217.500000	58.800000	14.700000	20.800000	287.700000	330.200000	11.800000	171.700000	117.700000	157.100000	163.1000
MADHYA PRADESH	3.270000	331.048000	8.790000	9.158000	12.892000	311.088000	114.686000	7.486000	7.006000	10.042000	35.270000	181.5740
MAHARASHTRA	6.974286	314.585714	7.417143	3.474286	4.791429	388.894286	240.980000	5.997143	19.925714	18.588571	75.648571	191.3114
MANIPUR	150.766667	451.800000	11.788889	55.122222	22.600000	498.055556	487.088889	82.411111	213.377778	56.000000	189.222222	278.4000
MEGHALAYA	211.228571	584.371429	11.042857	21.685714	14.900000	857.742857	757.228571	74.757143	430.042857	39.571429	225.571429	454.7000
MIZORAM	152.600000	440.588889	15.288889	29.944444	11.566667	452.311111	429.833333	96.255556	321.322222	64.633333	229.822222	372.1555
NAGALAND	134.227273	350.872727	10.354545	27.672727	18.481818	395.036364	340.318182	63.018182	213.381818	38.554545	121.154545	227.6272
ORISSA	36.653333	363.346667	5.136667	22.370000	10.810000	332.316667	212.516667	27.453333	70.723333	30.400000	116.056667	238.3666
PONDICHERRY	12.275000	116.425000	227.350000	25.425000	26.750000	78.025000	47.675000	16.725000	40.825000	395.150000	271.950000	119.9000
PUNJAB	12.160000	172.415000	13.905000	24.480000	25.965000	190.610000	46.325000	25.900000	16.165000	6.085000	21.700000	92.8350
RAJASTHAN	3.303030	194.554545	3.021212	4.721212	5.348485	195.278788	54.096970	3.815152	10.627273	6.254545	14.430303	86.1454
SIKKIM	206.900000	434.600000	20.900000	77.300000	47.550000	499.200000	483.800000	130.600000	323.550000	30.950000	209.850000	373.1500
TAMIL NADU	42.596875	91.571875	96.487500	14.021875	18.906250	72.606250	50.321875	18.068750	67.531250	184.625000	186.928125	116.3406
TRIPURA	220.750000	356.475000	11.125000	33.650000	11.225000	414.975000	465.425000	93.625000	391.575000	43.300000	176.650000	260.3500
UTTAR PRADESH	5.318310	291.232394	6.870423	13.157746	17.183099	280.067606	90.770423	10.107042	15.561972	4.576056	45.525352	175.0746
UTTARANCHAL	29.815385	426.784615	20.830769	49.592308	49.892308	432.792308	165.715385	51.669231	58.392308	9.238462	58.838462	204.4769
WEST BENGAL	56.647368	361.573684	7.363158	19.084211	15.031579	412.989474	308.531579	27.973684	139.489474	19.389474	124.373684	317.9789

In [242]:
```
ml=[1,2,3,9,4,5,6,7]
mls=list(map(lambda i:i**2,ml))
mls
```

Out[242]: [1, 4, 9, 81, 16, 25, 36, 49]

In [243]:
```
l1=[4,5,6,7,8,9,55,13]
ol=list(filter(lambda x:False if x%2==0 else True,l1))
ol
```

Out[243]: [5, 7, 9, 55, 13]

In [245]:
```
from functools import reduce
def sum(x,y):
    return x+y
l1=[1,2,3,4,5,6,7,8,9,10]
s=reduce(sum,l1)
s
```

Out[245]: 55

In [ ]: