Problem Statement:

We have to predict if the patient has a cardiovascular desease or not

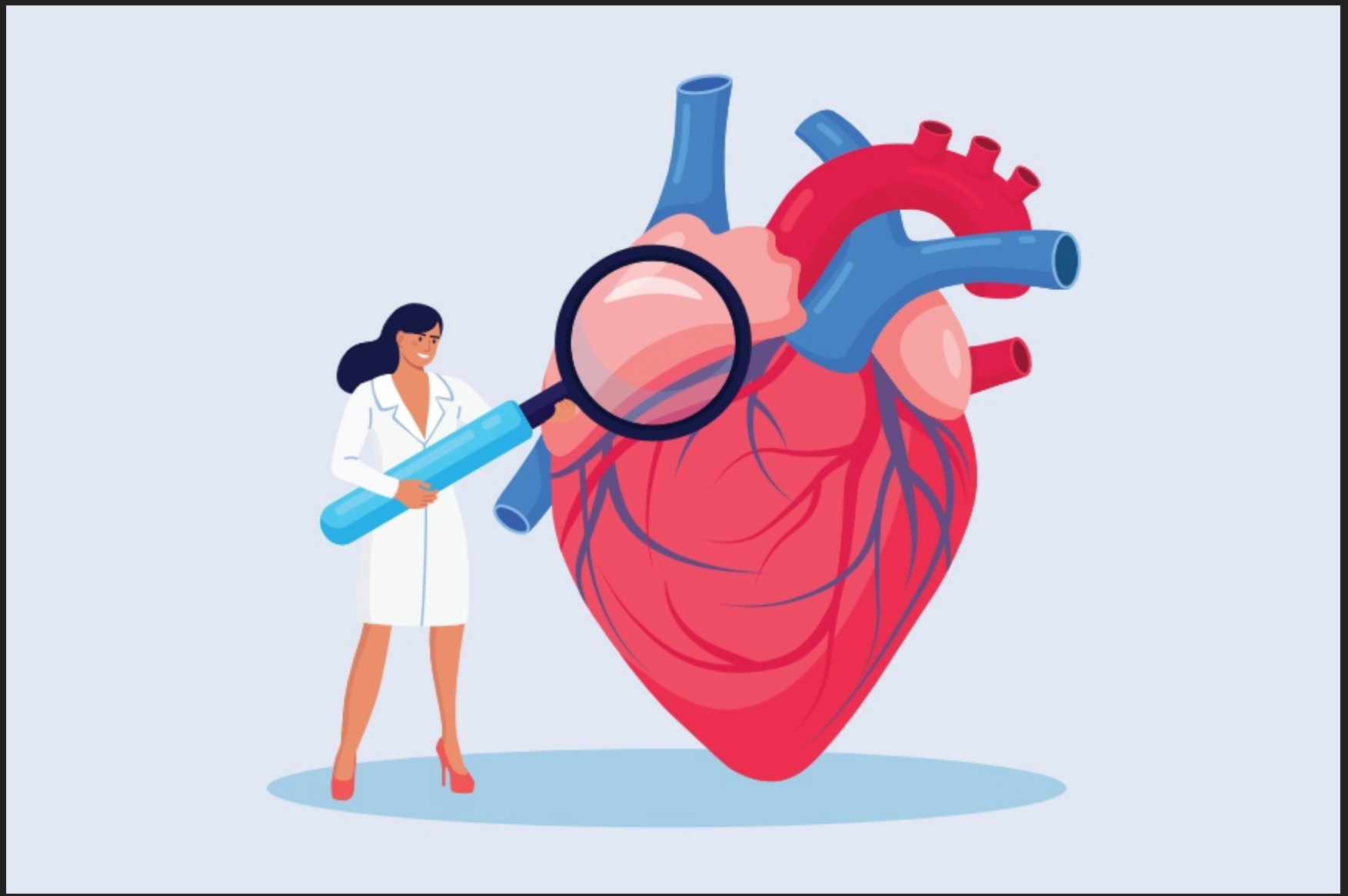


Image: Askaria School

Importing Libreries

In [1]:

```
import pandas as pd
import numpy as np
import datetime as dt
from datetime import date
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Reading Dataset

About Dataset

Data description

There are 3 types of input features:

Objective: factual information;

Examination: results of medical examination;

Subjective: information given by the patient.

Features:

- Age | Objective Feature | age | int (days)
- Height | Objective Feature | height | int (cm) |
- Weight | Objective Feature | weight | float (kg) |

- Gender | Objective Feature | gender | categorical code |
- Systolic blood pressure | Examination Feature | ap_hi | int |
- Diastolic blood pressure | Examination Feature | ap_lo | int |
- Cholesterol | Examination Feature | cholesterol | 1: normal, 2: above normal, 3: well above normal
- Glucose | Examination Feature | gluc | 1: normal, 2: above normal, 3: well above normal |
- Smoking | Subjective Feature | smoke | binary |
- Alcohol intake | Subjective Feature | alco | binary |
- Physical activity | Subjective Feature | active | binary |
- Presence or absence of cardiovascular disease | Target Variable | cardio | binary |

All of the dataset values were collected at the moment of medical examination.

In [2]:
```
df = pd.read_csv(r"C:\Users\Roshan Salunke\Downloads\Data Science Course\Projects\Cardiovascular_deseas
```

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

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Basic EDA

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   id           70000 non-null  int64  
 1   age          70000 non-null  int64  
 2   gender       70000 non-null  int64  
 3   height       70000 non-null  int64  
 4   weight       70000 non-null  float64
 5   ap_hi        70000 non-null  int64  
 6   ap_lo        70000 non-null  int64  
 7   cholesterol  70000 non-null  int64  
 8   gluc         70000 non-null  int64  
 9   smoke        70000 non-null  int64  
 10  alco         70000 non-null  int64  
 11  active       70000 non-null  int64  
 12  cardio       70000 non-null  int64  
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

- The dataset has 70k rows and 13 columns

In [5]:
```
1 df.describe()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	
count	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	7(
mean	49972.419900	19468.865814	1.349571	164.359229	74.205690	128.817286	96.630414	1.366871	1.226457	0.
std	28851.302323	2467.251667	0.476838	8.210126	14.395757	154.011419	188.472530	0.680250	0.572270	0.
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.

In [6]:
```
1 df['id'].value_counts().sum()
```

```
70000
```

- There are total unique 70k patients in the dataset
- We can drop id column as it's not important

In [7]:
```
1 df.drop('id',axis=1,inplace=True)
```

Checking NAN Values

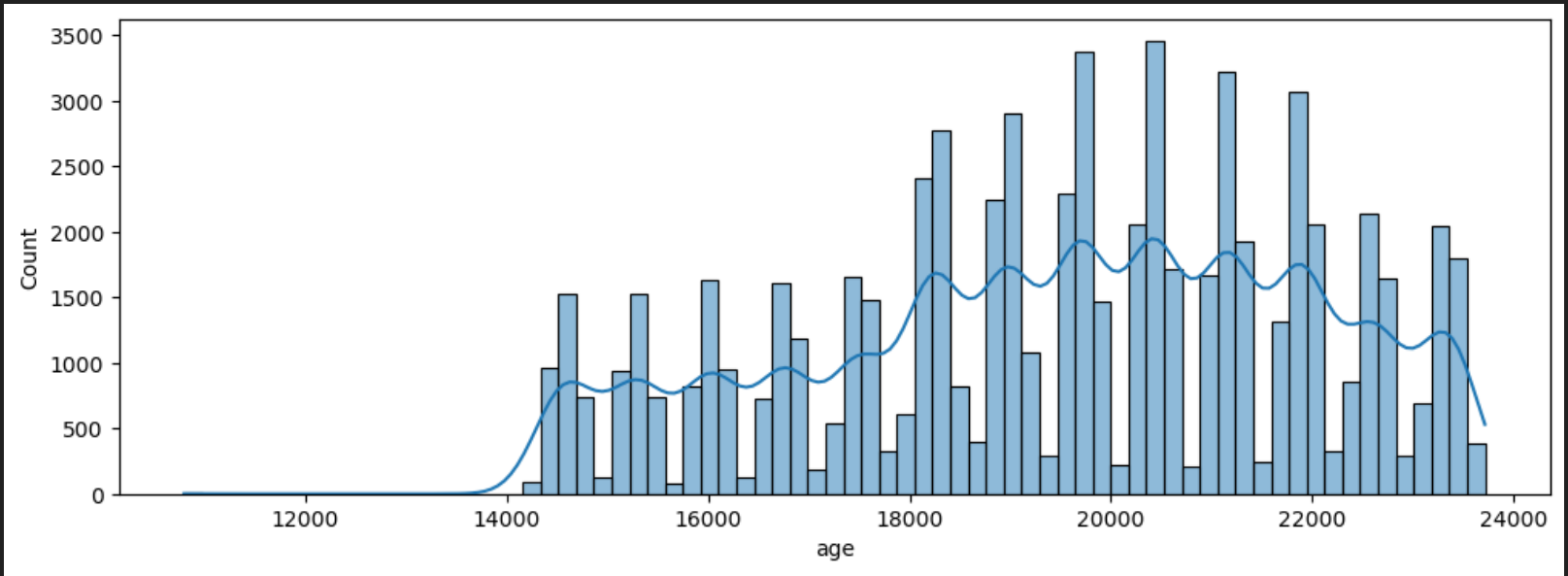
In [8]:
```
df.isnull().sum()
```

```
age            0
gender         0
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

- There are no null values in the dataset which is a good thing for us.

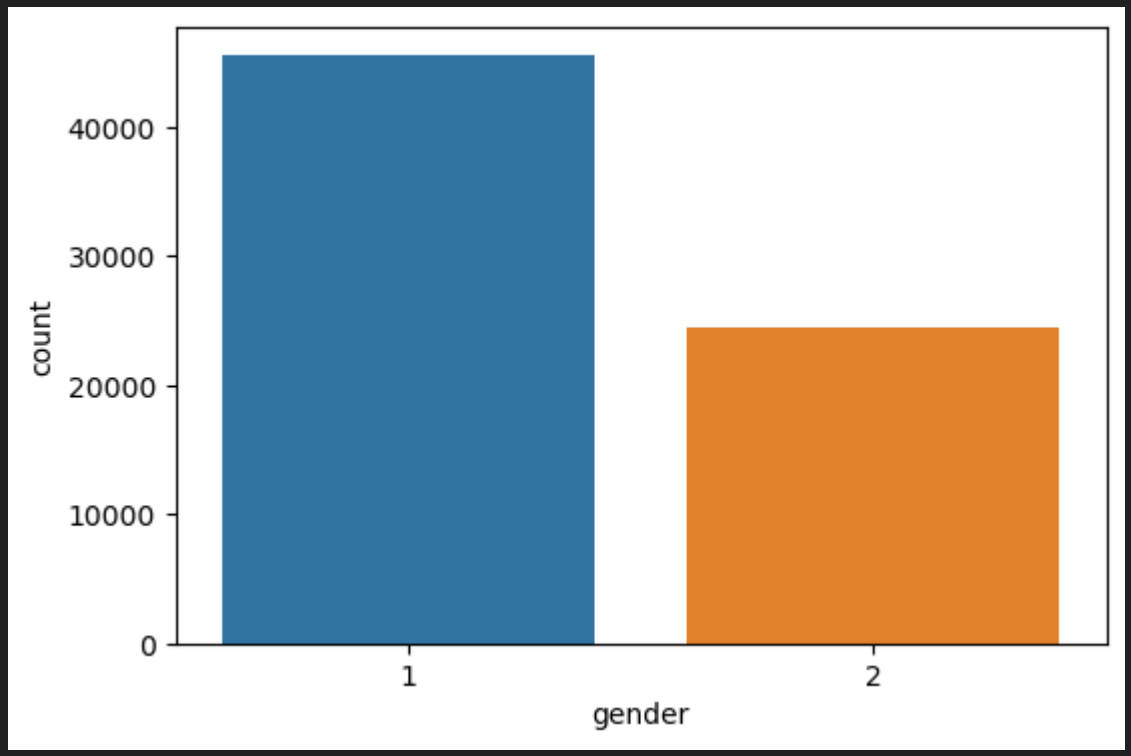
EDA

In [9]:
```
plt.figure(figsize=(12,4))
sns.histplot(df['age'],kde=True);
```



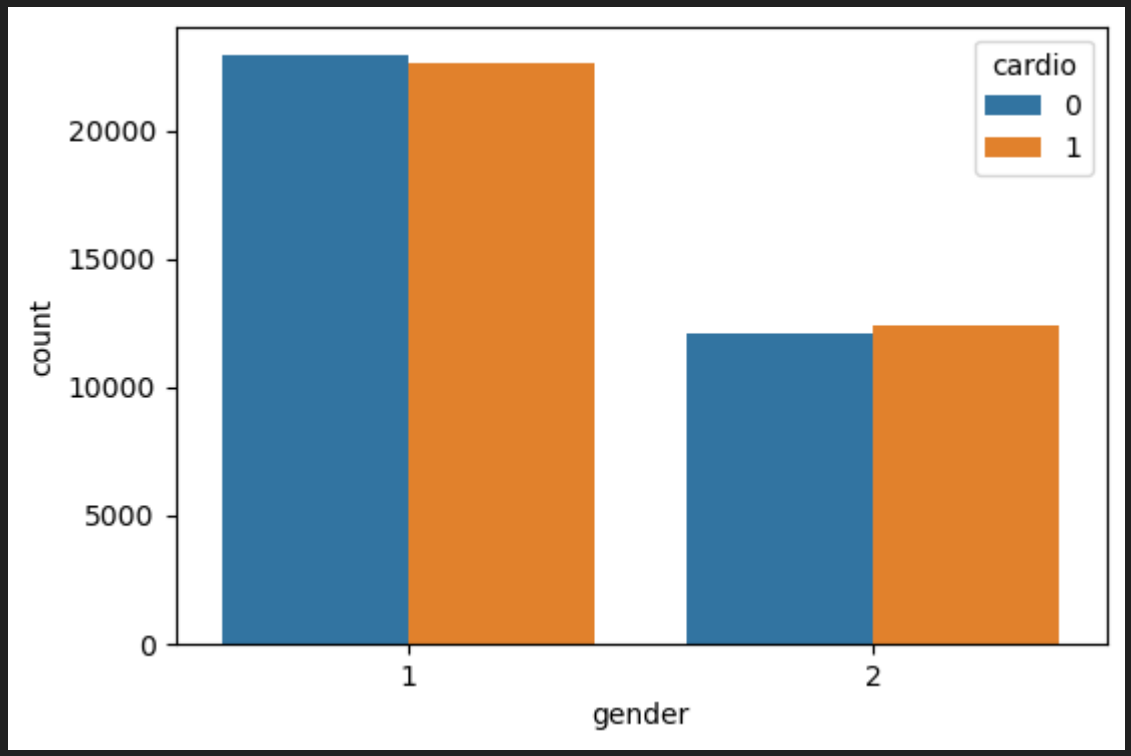
- Most people have number of days above 20k

In [10]:
```
plt.figure(figsize=(6,4))
sns.countplot(df['gender']);
```



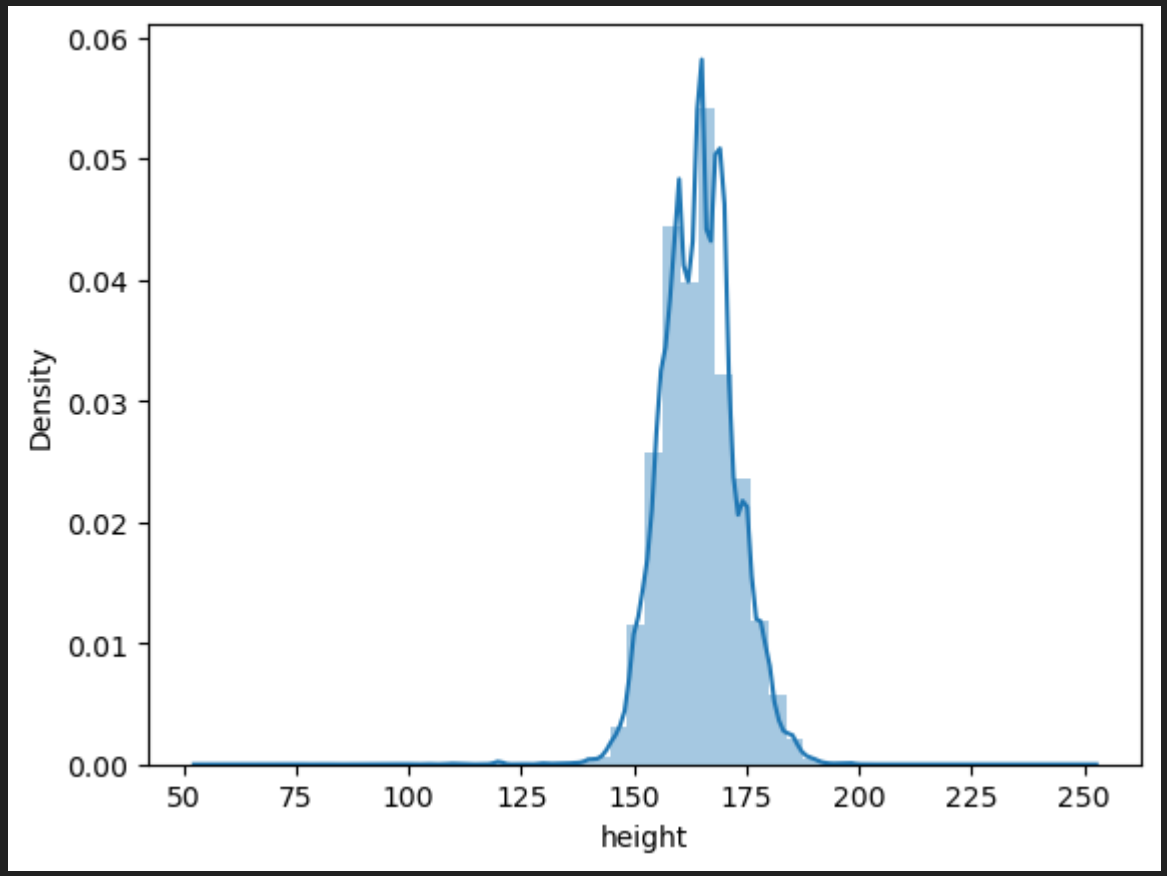
- Womens are having more cardiovascular desease.

In [11]:
```
plt.figure(figsize=(6,4))
sns.countplot(df['gender'],hue=df['cardio']);
```



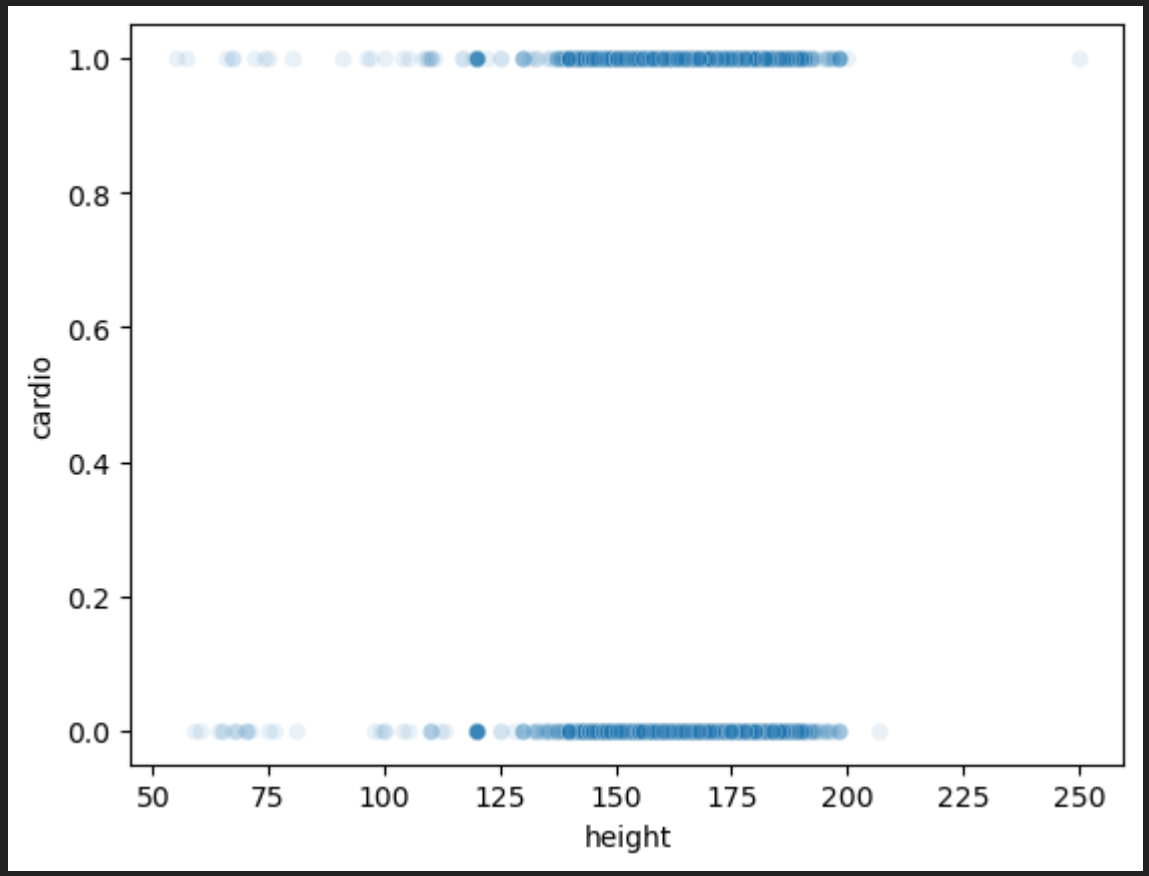
- The number is equal in women and mens of having or not having desease.

In [12]:
```
1 sns.distplot(df['height']);
```



- Most people have height around 161cm.

In [13]:
```
1 sns.scatterplot(df['height'],df['cardio'],alpha=.1);
```



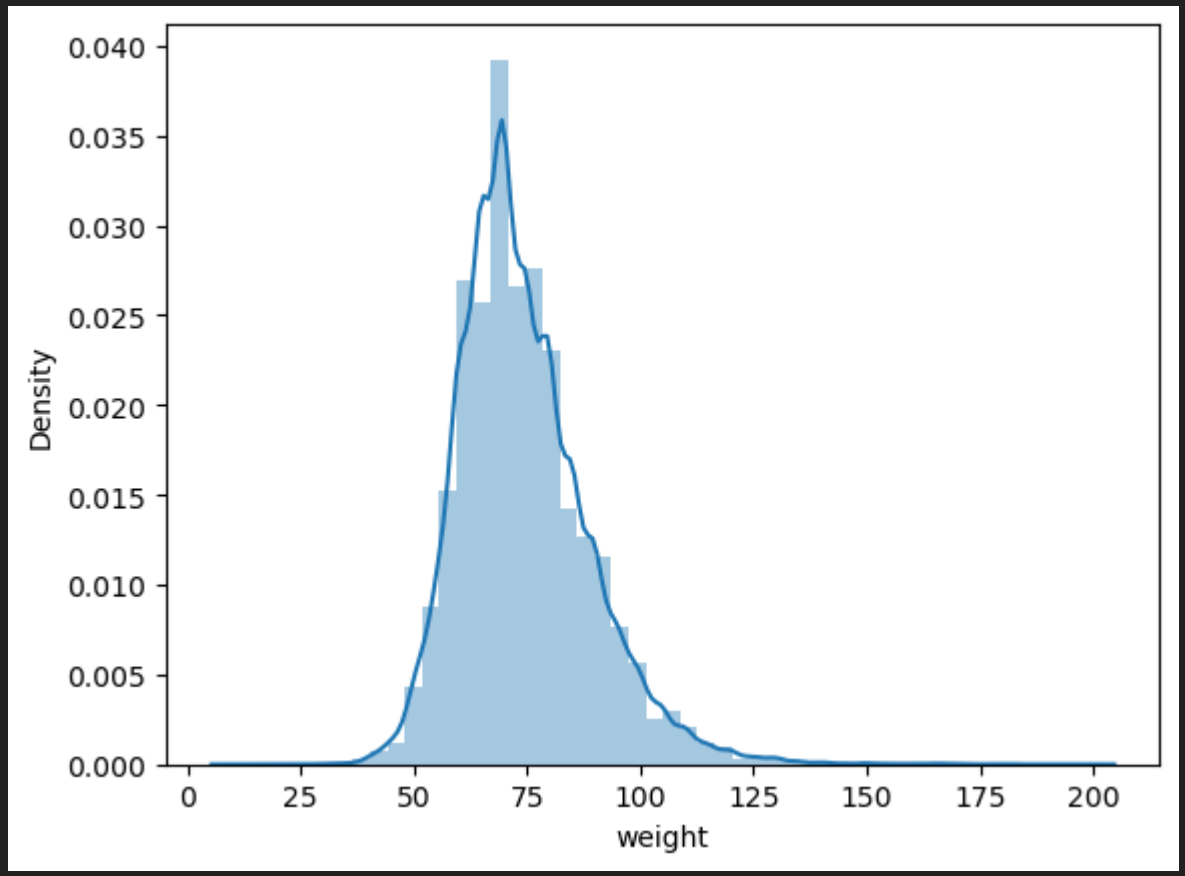
In [14]:

```
print('min heigh by cardio type',df.groupby('cardio').height.min())
print('max heigh by cardio type',df.groupby('cardio').height.max())
```

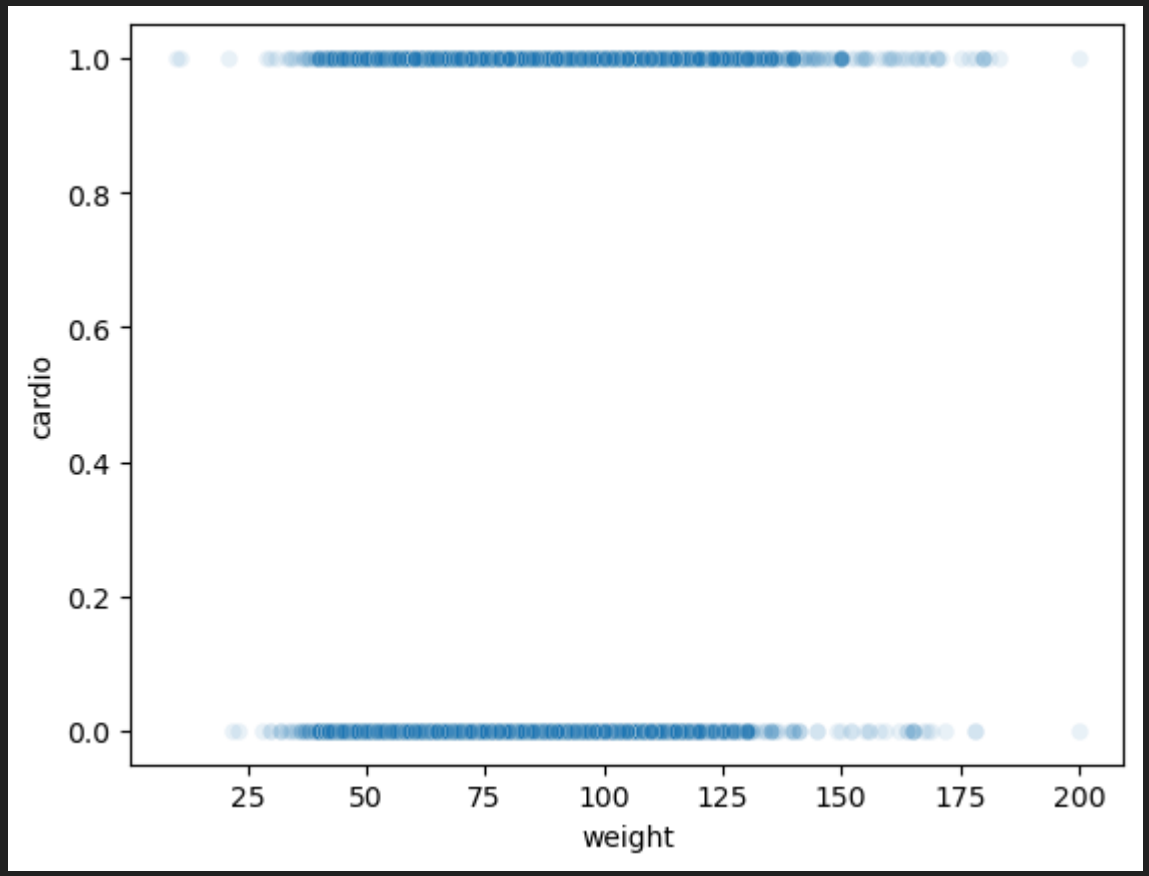
```
min heigh by cardio type cardio
0    59
1    55
Name: height, dtype: int64
max heigh by cardio type cardio
0    207
1    250
Name: height, dtype: int64
```

- The min height of people having desease is 55cm and max is 250.
- The max height of people not having desease is 59cm and max is 207.
- the distribution of height of both type of people seems to be equal.

In [15]:
```
1 sns.distplot(df['weight']);
```



In [16]:
```
1 sns.scatterplot(df['weight'],df['cardio'],alpha=.1);
```



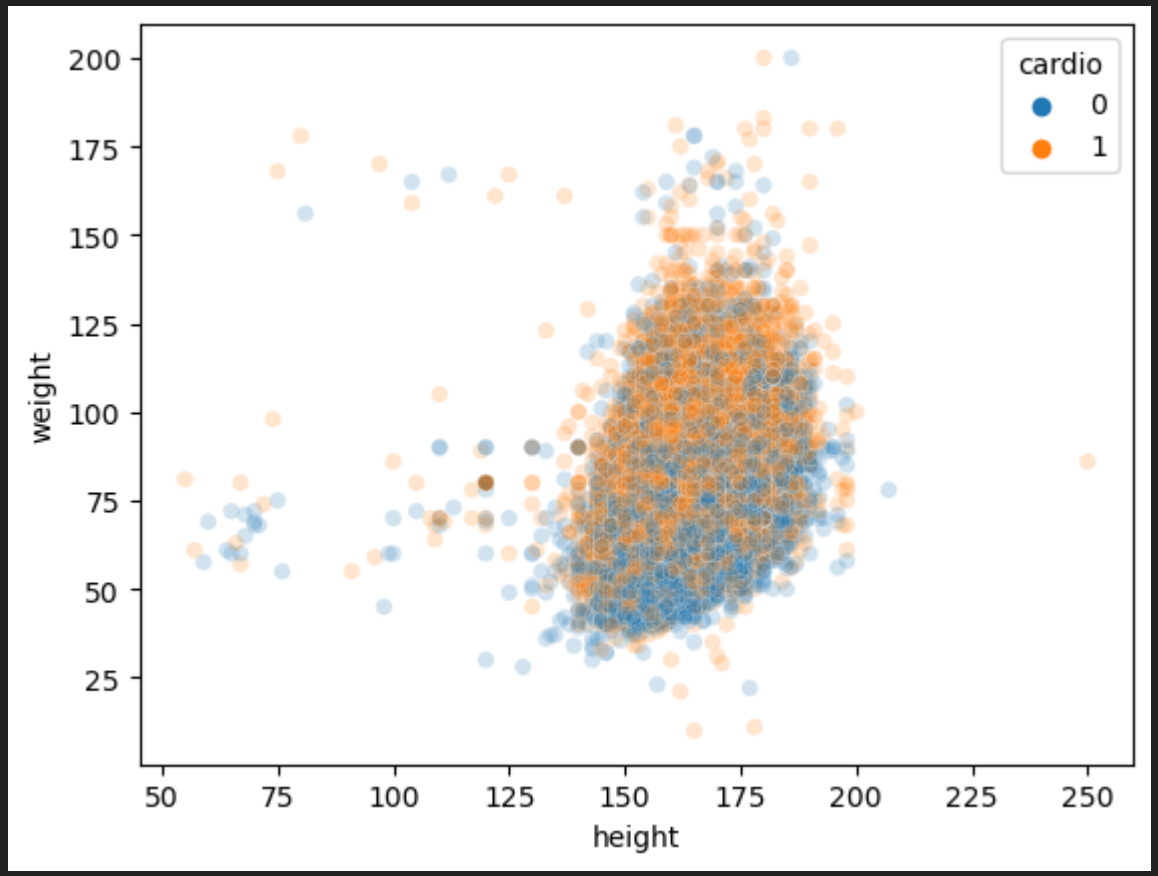
In [17]:

```
print('min weight by cardio type',df.groupby('cardio').weight.min())
print('max weight by cardio type',df.groupby('cardio').weight.max())
```

```
min weight by cardio type cardio
0    22.0
1    10.0
Name: weight, dtype: float64
max weight by cardio type cardio
0    200.0
1    200.0
Name: weight, dtype: float64
```

- The min weight of people having desease is 10kg and max is 200kg.
- The max weight of people not having desease is 20kg and max is 200kg.
- the distribution of weight of both type of people seems to be equal.

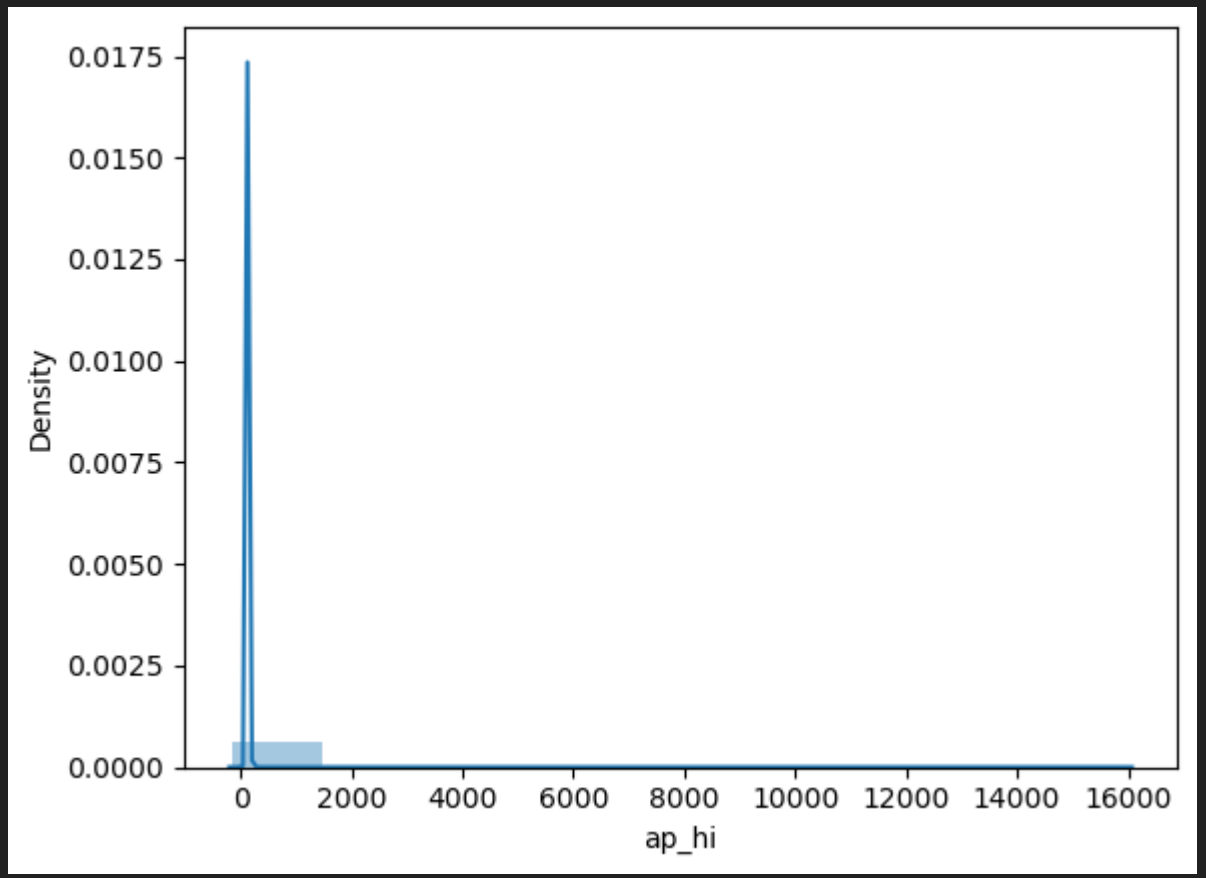
In [18]:
```
1 sns.scatterplot(df['height'],df['weight'],hue=df['cardio'],alpha=.2);
```



- There is no clear pattern in weight, height and desease of the people.

In [19]:

```
# Systolic blood pressure
sns.distplot(df['ap_hi'],bins=10);
```

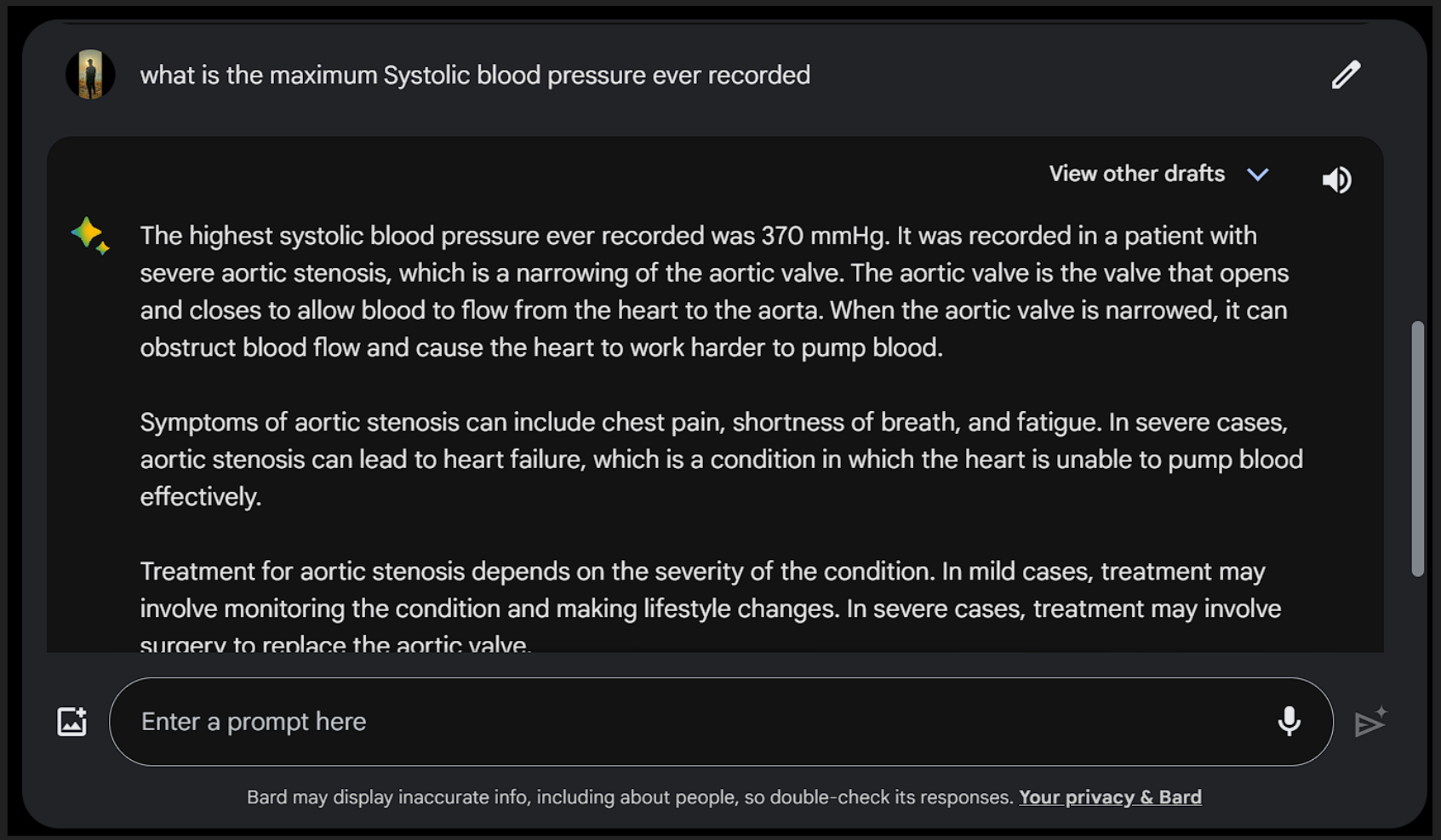


In [20]:
```
1 df['ap_hi'].describe()
```

```
count    70000.000000
mean       128.817286
std        154.011419
min       -150.000000
25%        120.000000
50%        120.000000
75%        140.000000
max      16020.000000
Name: ap_hi, dtype: float64
```

In [21]:
```
1 df['ap_hi'].value_counts().sort_values(ascending=True)
```

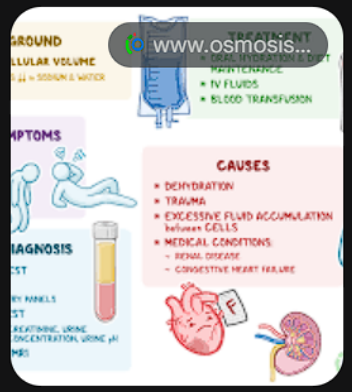
```
2000       1
187        1
1205       1
1110       1
960        1
        ...
150     4450
110     8644
130     8961
140     9506
120    27699
Name: ap_hi, Length: 153, dtype: int64
```


what is the maximum Systolic blood pressure ever recorded
View other drafts
The highest systolic blood pressure ever recorded was 370 mmHg. It was recorded in a patient with severe aortic stenosis, which is a narrowing of the aortic valve. The aortic valve is the valve that opens and closes to allow blood to flow from the heart to the aorta. When the aortic valve is narrowed, it can obstruct blood flow and cause the heart to work harder to pump blood.
Symptoms of aortic stenosis can include chest pain, shortness of breath, and fatigue. In severe cases, aortic stenosis can lead to heart failure, which is a condition in which the heart is unable to pump blood effectively.
Treatment for aortic stenosis depends on the severity of the condition. In mild cases, treatment may involve monitoring the condition and making lifestyle changes. In severe cases, treatment may involve surgery to replace the aortic valve.
Enter a prompt here
Bard may display inaccurate info, including about people, so double-check its responses. Your privacy & Bard

what is the minimum Systolic blood pressure ever recorded

View other drafts

The lowest systolic blood pressure ever recorded was 30 mmHg. This was recorded in a patient with severe hypovolemic shock.



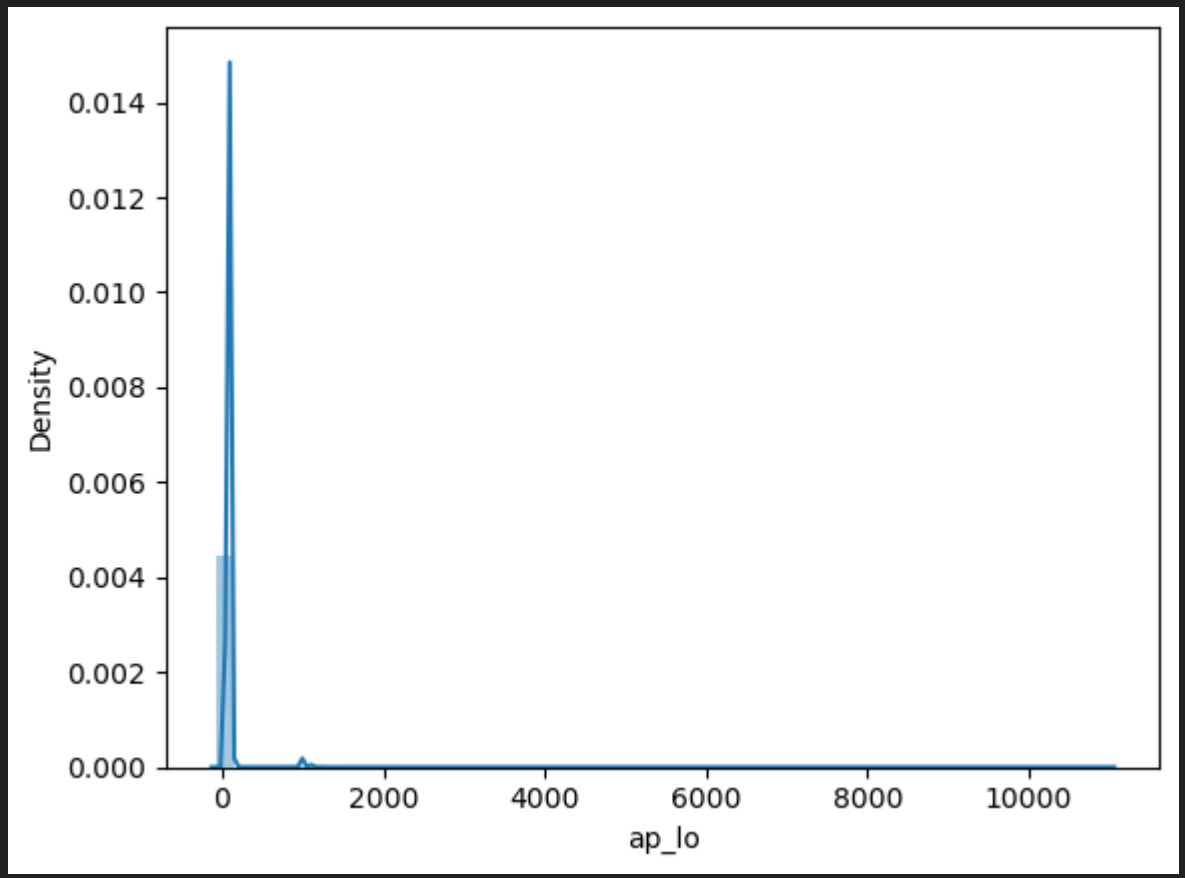
Enter a prompt here

Bard may display inaccurate info, including about people, so double-check its responses. Your privacy & Bard

The values of Systolic blood pressure are unreasonably low and high.

In [22]:

```
# Diastolic blood pressure
sns.distplot(df['ap_lo']);
```

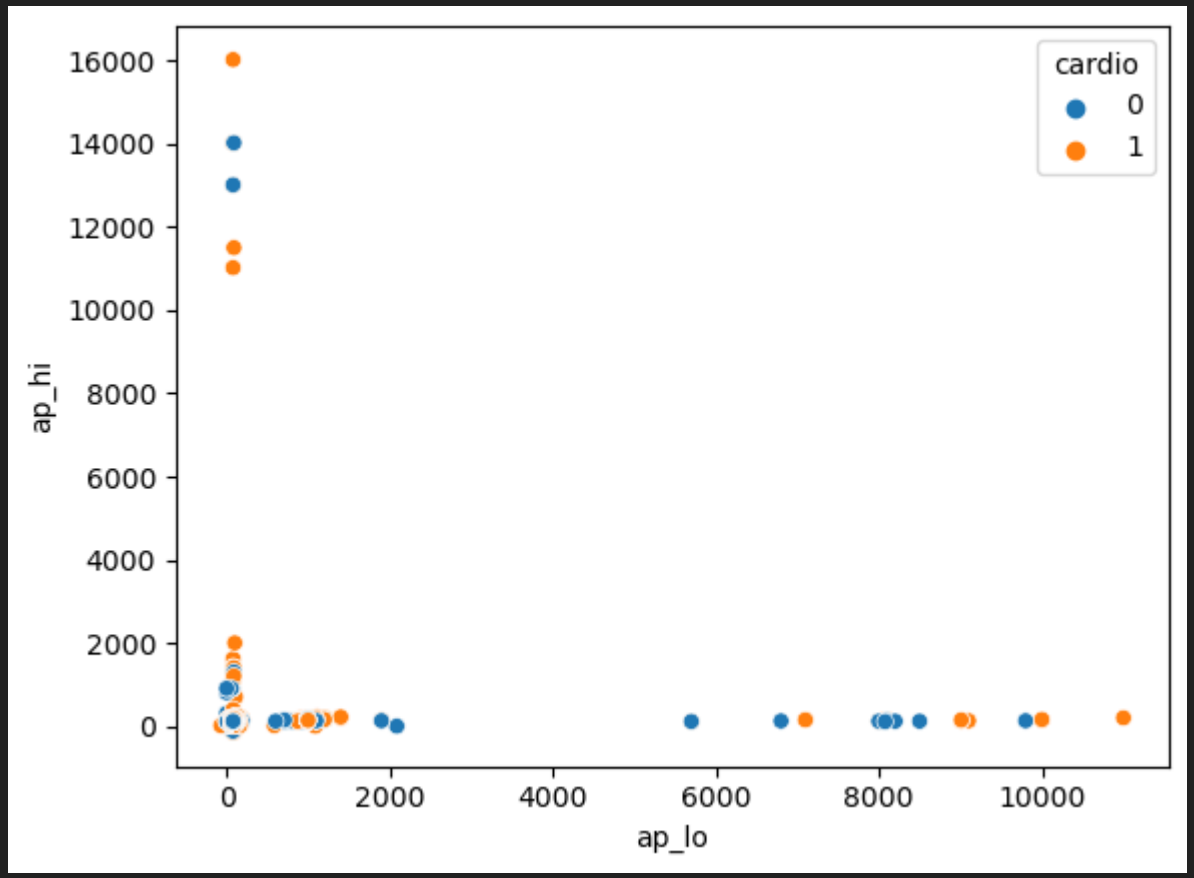


In [23]:
```
1 df['ap_lo'].describe()
```

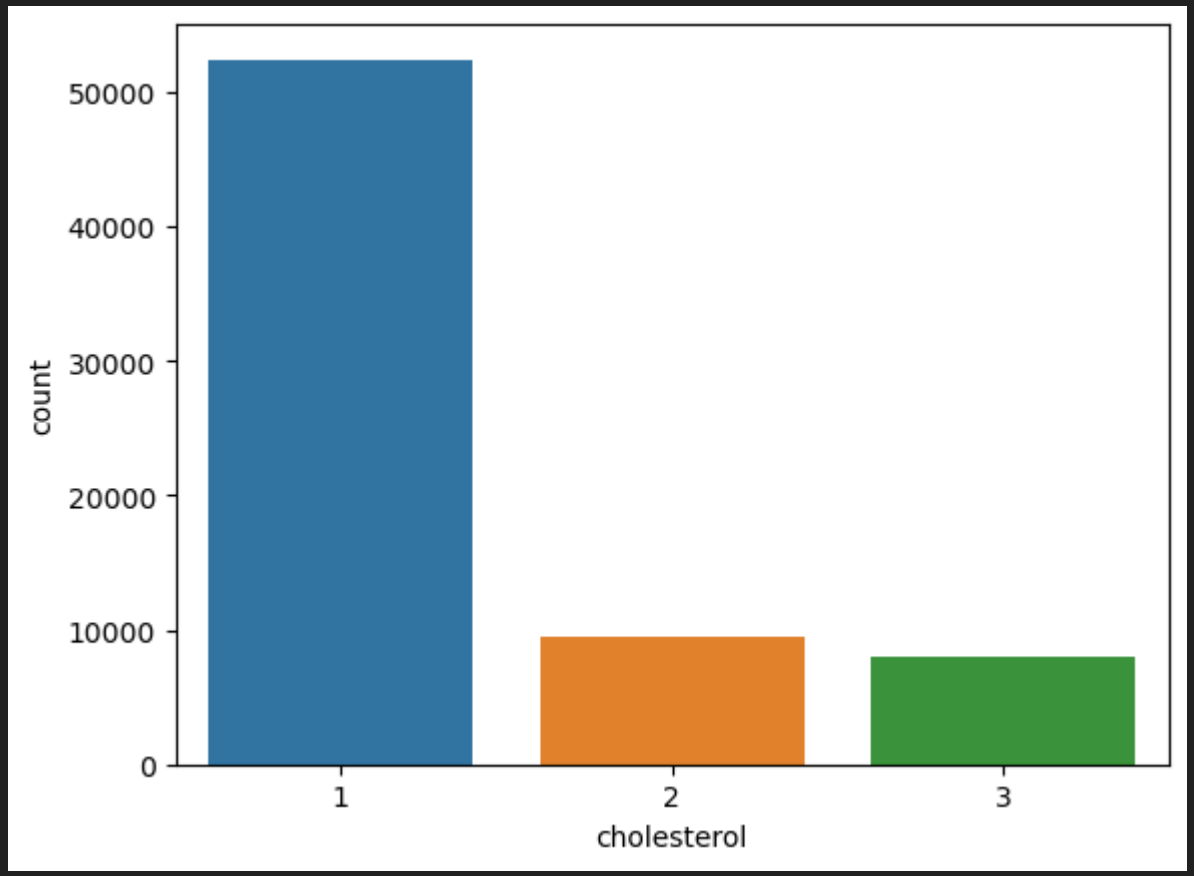
```
count    70000.000000
mean        96.630414
std        188.472530
min        -70.000000
25%         80.000000
50%         80.000000
75%         90.000000
max      11000.000000
Name: ap_lo, dtype: float64
```

> The similary thing is with Diastolic blood pressure

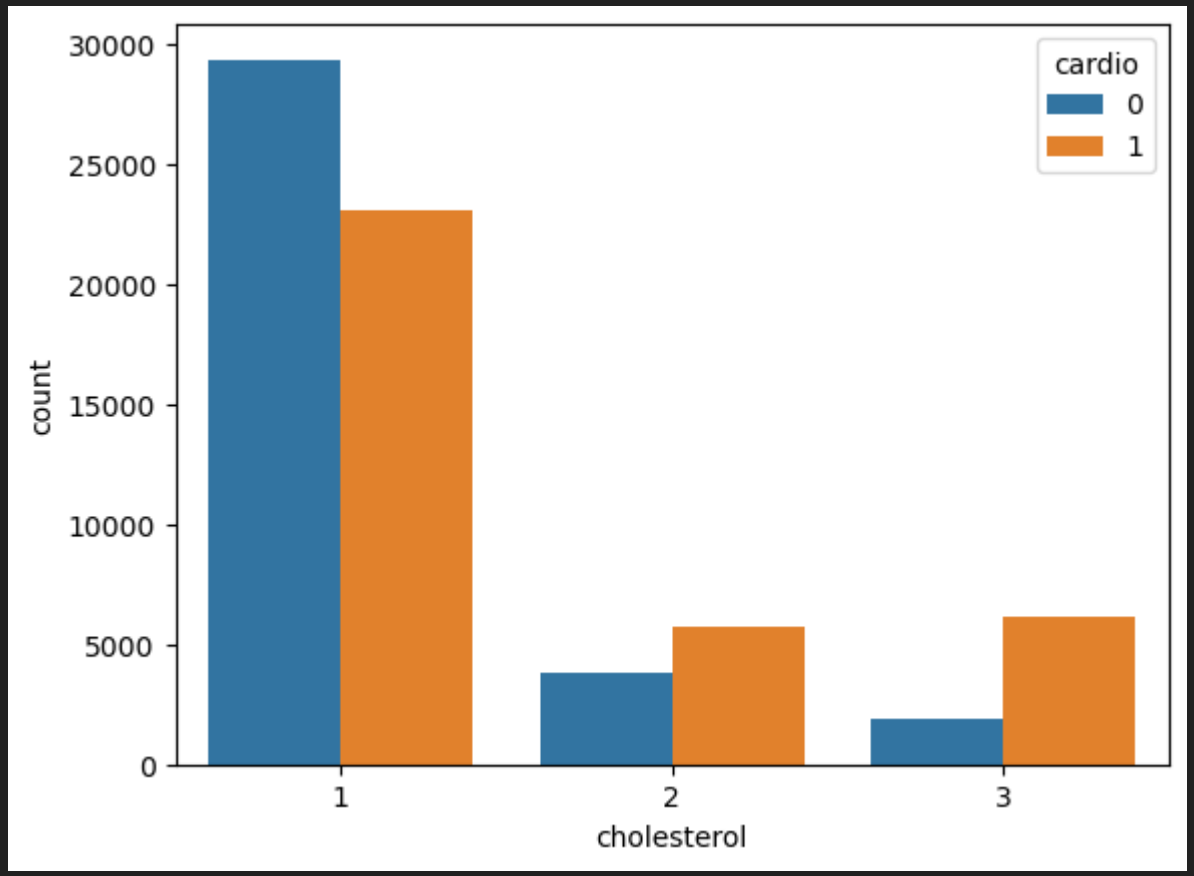
In [24]:
```
1 sns.scatterplot(df['ap_lo'],df['ap_hi'],hue=df['cardio']);
```



In [25]:
```
1 sns.countplot(df['cholesterol']);
```

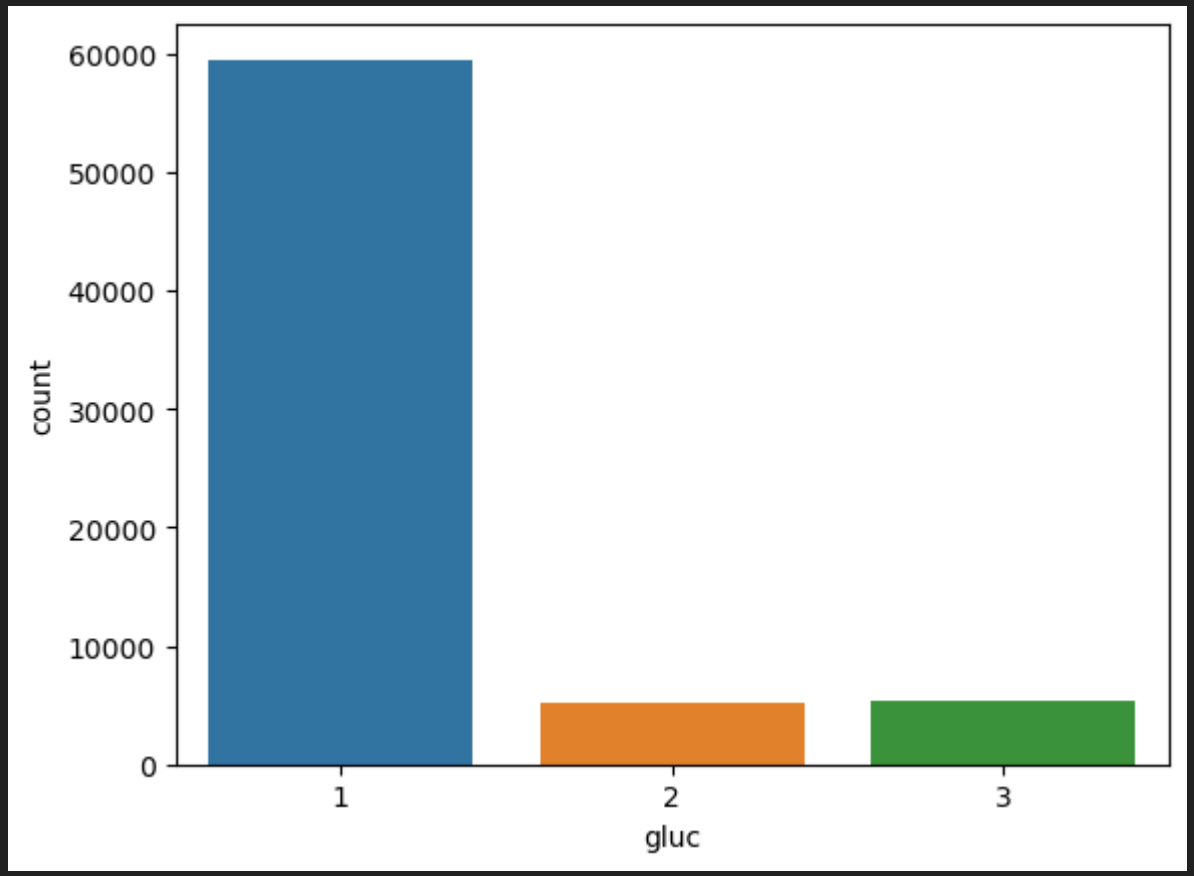


In [26]:
```
1 sns.countplot(df['cholesterol'],hue=df['cardio']);
```

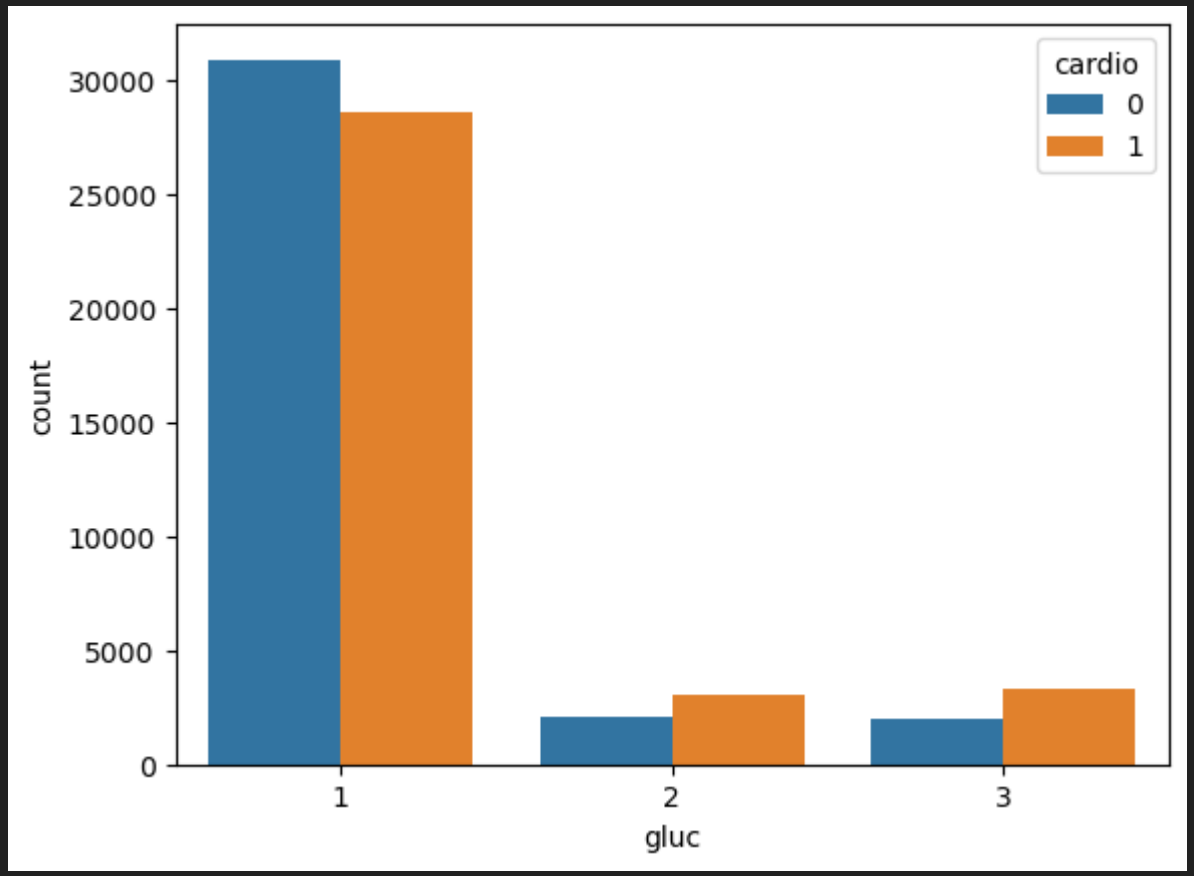


- The cholesterol of most people is normal.
- Most people are having desease where cholesterol is above normal or well above normal.

In [27]:
```
1 sns.countplot(df['gluc']);
```

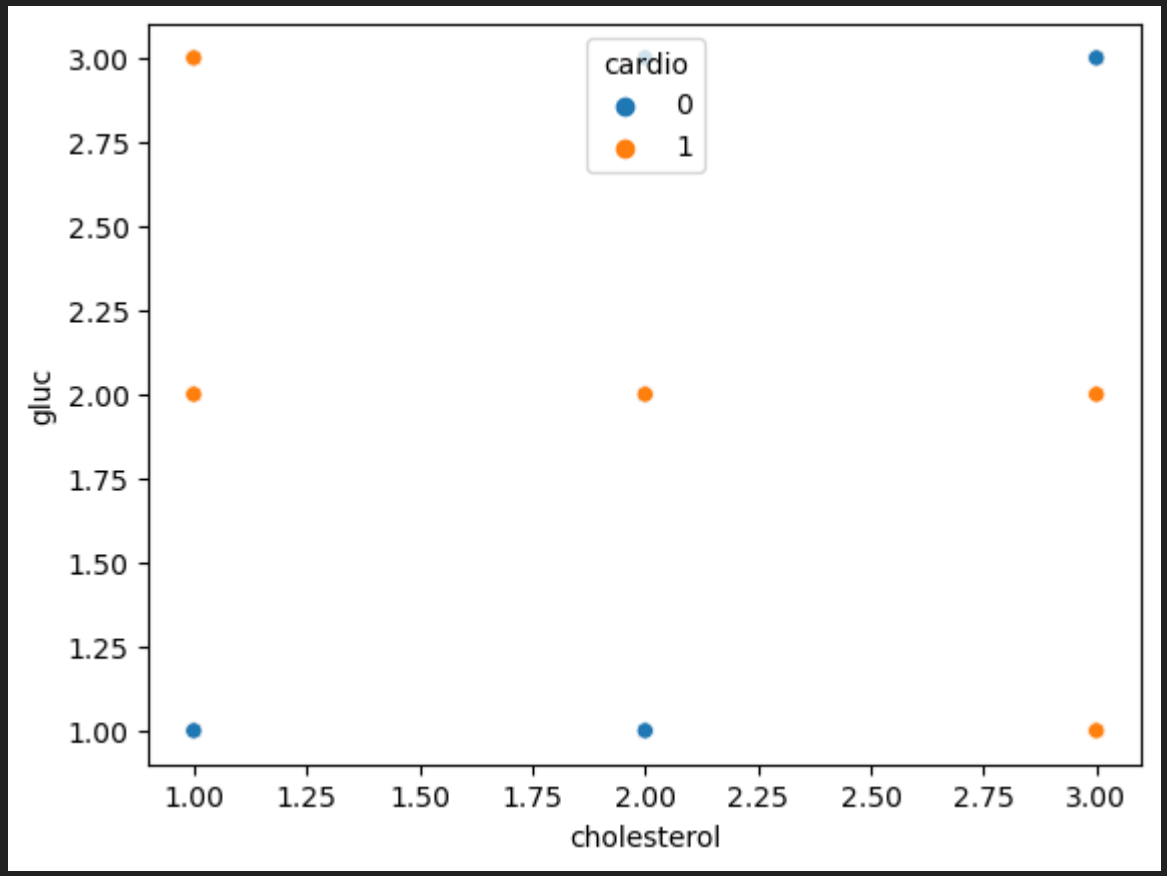


In [28]:
```
1 sns.countplot(df['gluc'],hue=df['cardio']);
```

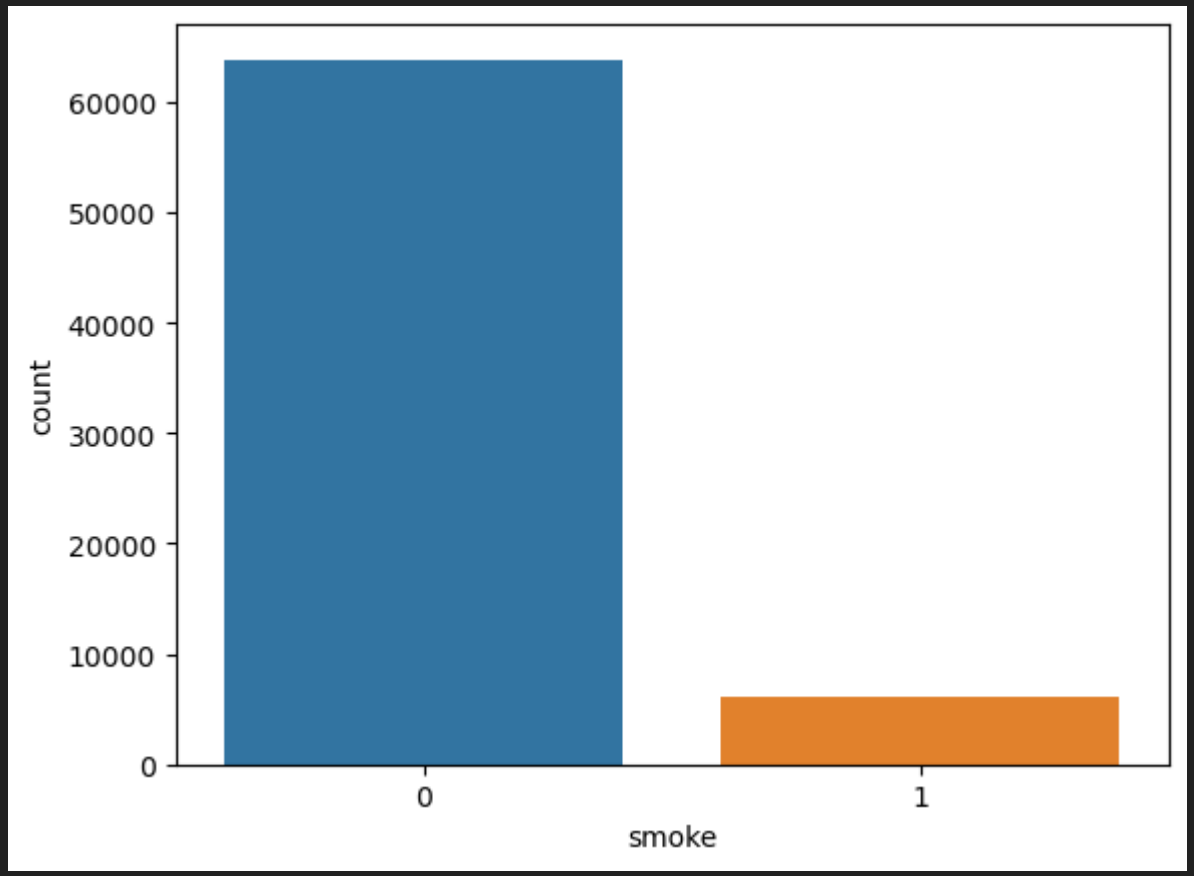


- If the glucose is above normal or well above normal then the person is more likely to have desease.

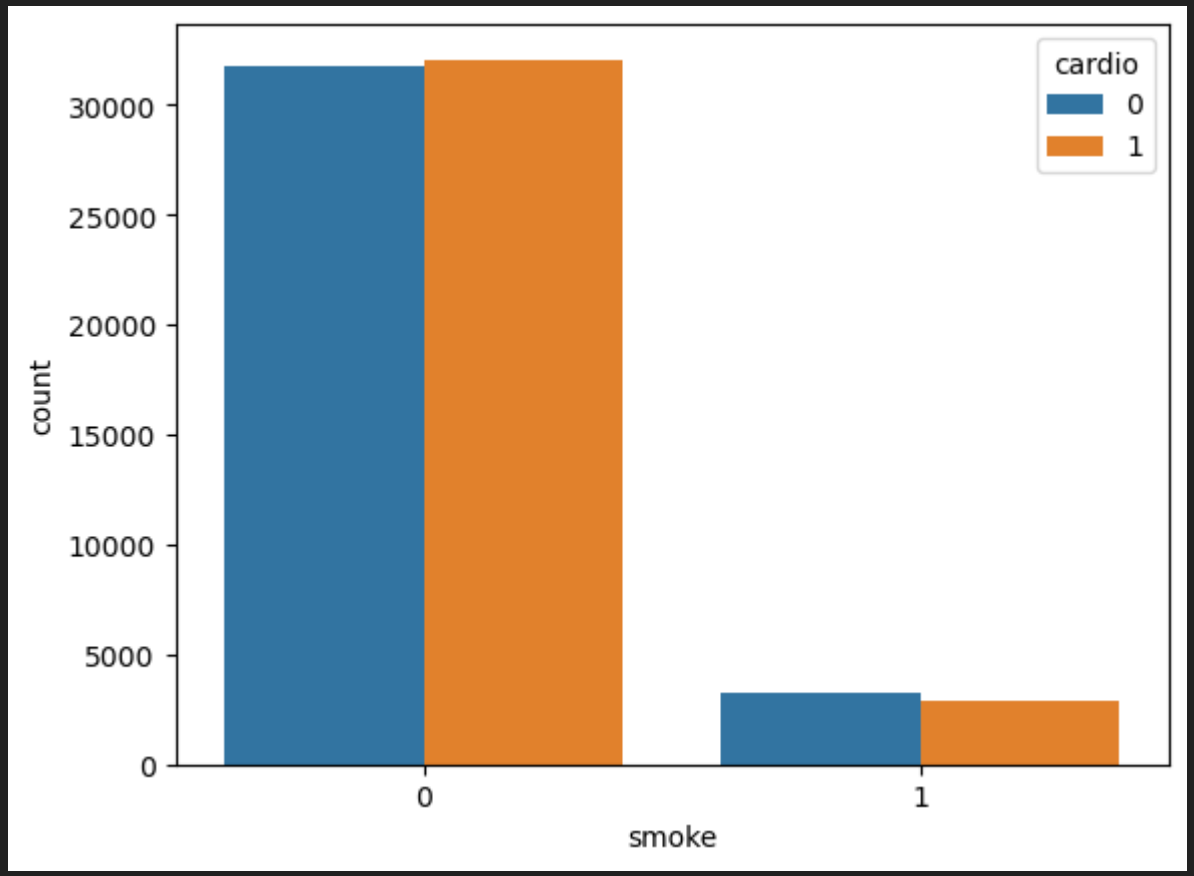
In [29]:
```
1 sns.scatterplot(df['cholesterol'],df['gluc'],df['cardio']);
```



In [30]:
```
1 sns.countplot(df['smoke']);
```

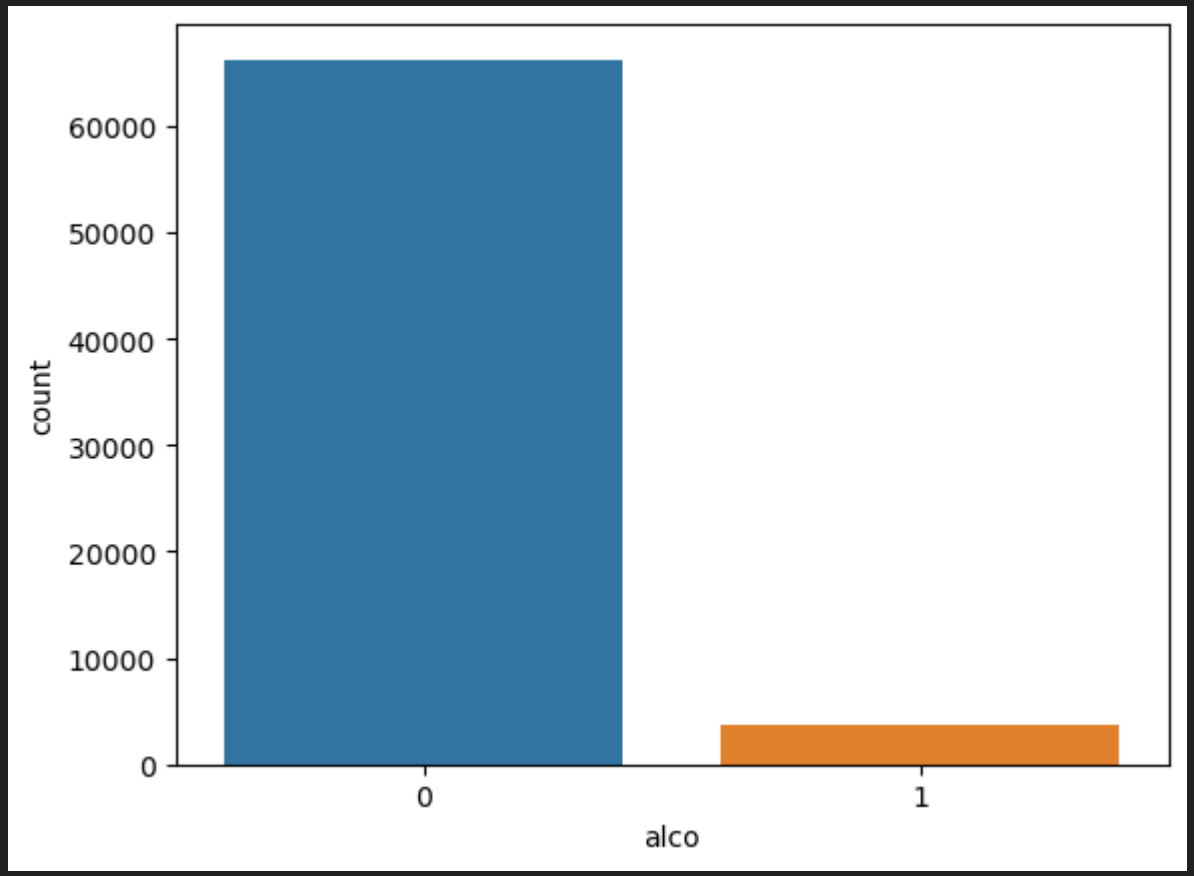


In [31]:
```
1 sns.countplot(df['smoke'],hue=df['cardio']);
```

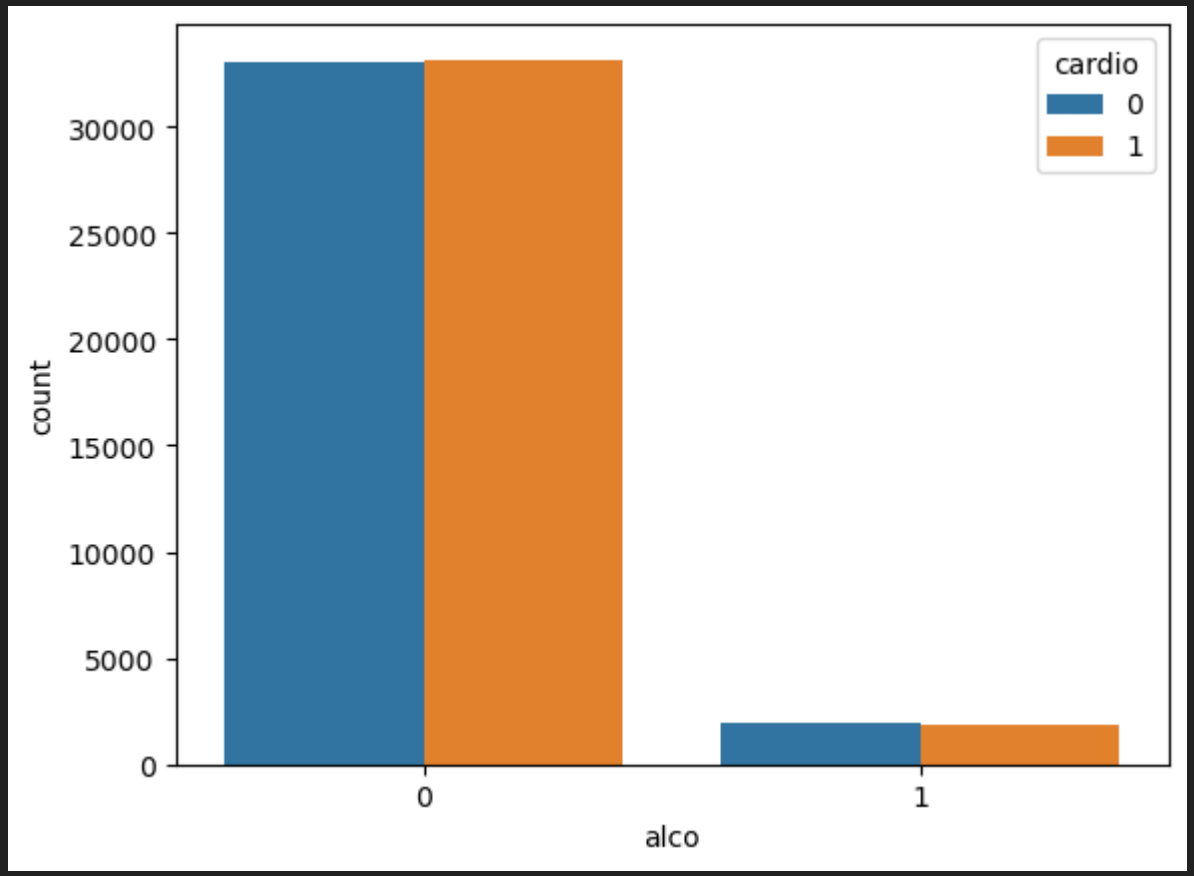


- Most people don't smoke.
- people smoking and having desease and people and not smoking having desease is somewhat equal.

In [32]:
```
1 sns.countplot(df['alco']);
```

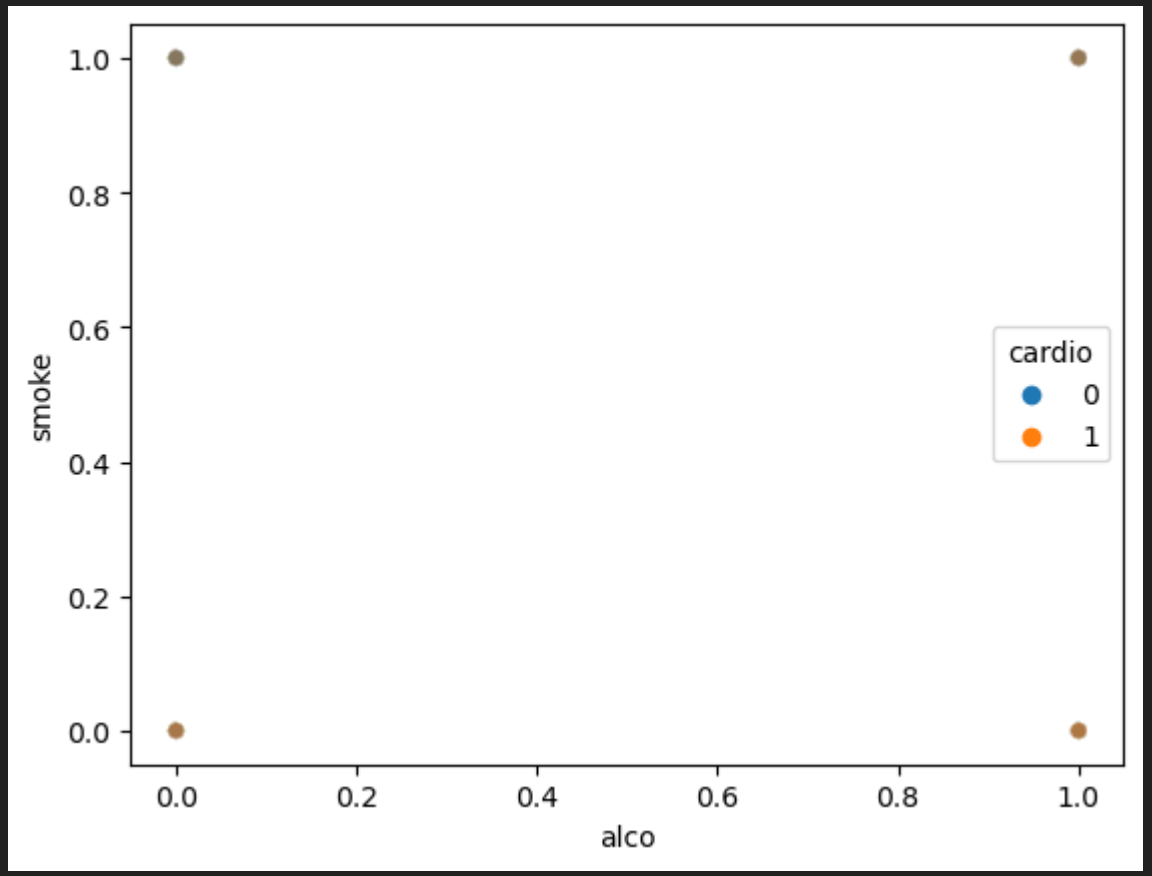


In [33]:
```
1 sns.countplot(df['alco'],hue=df['cardio']);
```

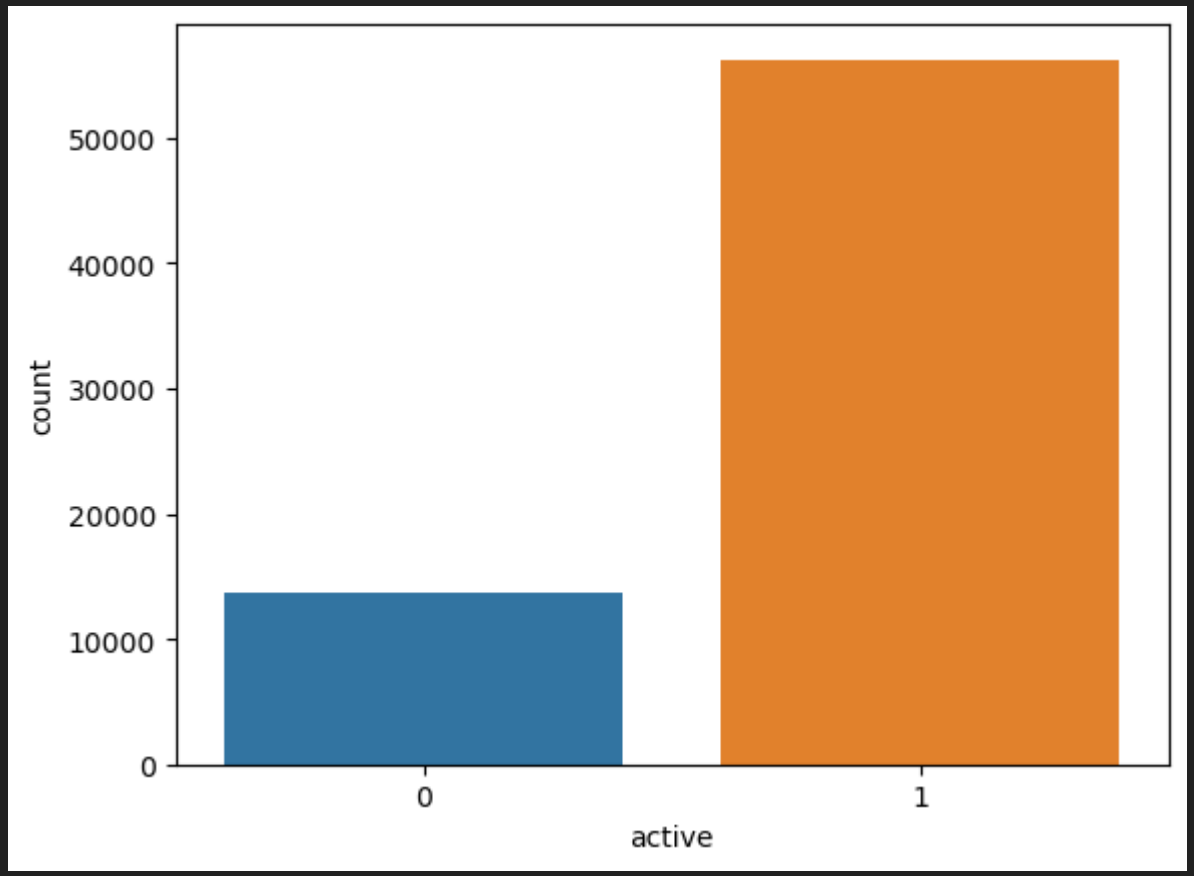


- Also alcohol intake does not show any effect.

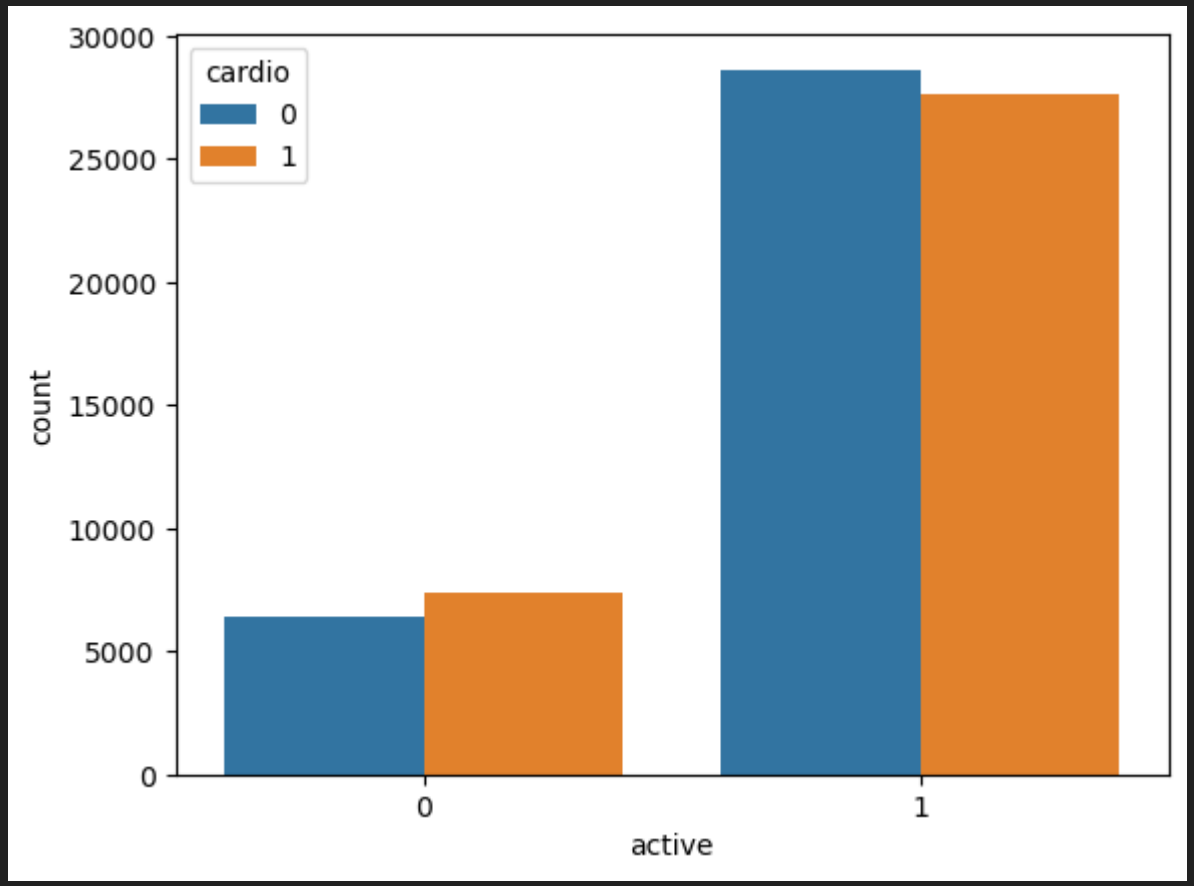
In [34]:
```
1 sns.scatterplot(df['alco'],df['smoke'],alpha=.1,hue=df['cardio']);
```



In [35]:
```
1 sns.countplot(df['active']);
```

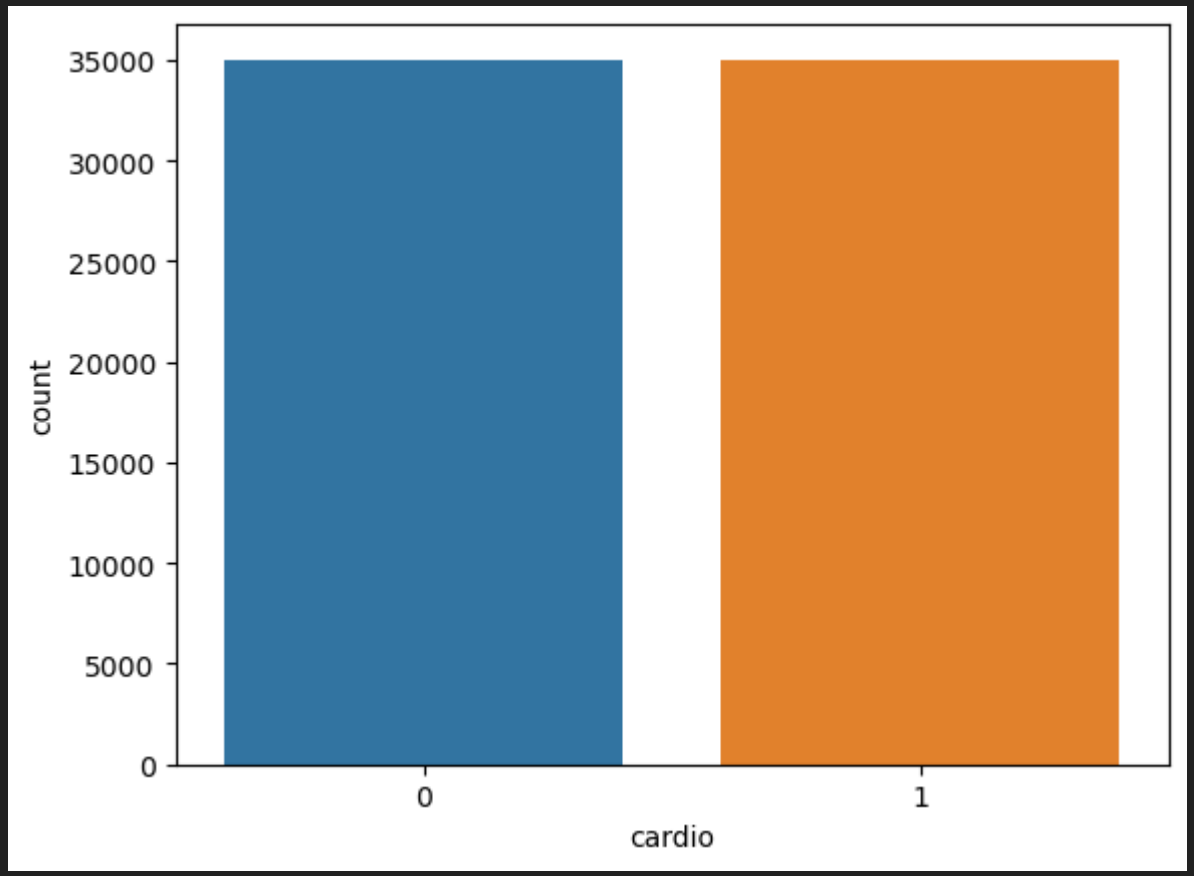


In [36]:
```
1 sns.countplot(df['active'],hue=df['cardio']);
```



- Most people have physical activity.
- The people who have physical activity have desease.

In [37]:
```
1 sns.countplot(df['cardio']);
```



- The dataset is well balanced and it's a good thing for us.

Model Building

First we will make a base model without doing anything to our features excep scalling them.

In [38]:
```
# separating target and other features
x = df.drop('cardio',axis=1)
y = df['cardio']
```

Scaling

In [39]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [40]:
```
xcols = x.columns
x = sc.fit_transform(x)
x = pd.DataFrame(x, columns=xcols)
x.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	-0.436062	1.364055	0.443452	-0.847873	-0.122182	-0.088238	-0.539322	-0.39572	-0.310879	-0.238384	0.494167
1	0.307686	-0.733108	-1.018168	0.749831	0.072610	-0.035180	2.400793	-0.39572	-0.310879	-0.238384	0.494167
2	-0.247997	-0.733108	0.078047	-0.708942	0.007679	-0.141297	2.400793	-0.39572	-0.310879	-0.238384	-2.023607
3	-0.748152	1.364055	0.565254	0.541435	0.137541	0.017879	-0.539322	-0.39572	-0.310879	-0.238384	0.494167
4	-0.808543	-0.733108	-1.018168	-1.264666	-0.187113	-0.194356	-0.539322	-0.39572	-0.310879	-0.238384	-2.023607

Splitting dataset for training and testing

In [41]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,stratify=y, random_state=42, test_size=.3)
```

In [42]:
```
x_train.shape, y_train.shape
```

```
((49000, 11), (49000,))
```

Logistics Regression

In [43]:
```
from sklearn.linear_model import LogisticRegression
```

In [44]:
```
lr = LogisticRegression()
lr.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegression()
```

In [45]:
```
pred = lr.predict(x_test)
```

In [46]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.75      0.72     10506
           1       0.73      0.67      0.70     10494

    accuracy                           0.71     21000
   macro avg       0.71      0.71      0.71     21000
weighted avg       0.71      0.71      0.71     21000
```

In [47]:

```
from sklearn.metrics import accuracy_score
print('train accuracy: ',accuracy_score(lr.predict(x_train),y_train))
print('test accuracy: ',accuracy_score(lr.predict(x_test),y_test))
```

```
train accuracy:  0.7199183673469388
test accuracy:  0.712047619047619
```

Our base model is giving an accuracy of 71% with similary precision and recall values and f1-score.

It is a good model but should be more accurate.

In [48]:
```
1 df.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	17474	1	156	56.0	100	60	1	1	0	0	0	0

Capping unreasonably high and low values.

- Here we will take normal values of ap_hi and ap_lo that are Systolic blood pressure and Diastolic blood pressure.

In [49]:

```
df = df[(df['ap_hi']>30) & (df['ap_lo']<370)]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68862 entries, 0 to 69999
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          68862 non-null  int64  
 1   gender       68862 non-null  int64  
 2   height       68862 non-null  int64  
 3   weight       68862 non-null  float64
 4   ap_hi        68862 non-null  int64  
 5   ap_lo        68862 non-null  int64  
 6   cholesterol  68862 non-null  int64  
 7   gluc         68862 non-null  int64  
 8   smoke        68862 non-null  int64  
 9   alco         68862 non-null  int64  
 10  active       68862 non-null  int64  
 11  cardio       68862 non-null  int64  
dtypes: float64(1), int64(11)
memory usage: 6.8 MB
```

In [50]:

```
df = df[(df['ap_lo']>30) & (df['ap_lo']<360)]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 68808 entries, 0 to 69999
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          68808 non-null  int64  
 1   gender       68808 non-null  int64  
 2   height       68808 non-null  int64  
 3   weight       68808 non-null  float64
 4   ap_hi        68808 non-null  int64  
 5   ap_lo        68808 non-null  int64  
 6   cholesterol  68808 non-null  int64  
 7   gluc         68808 non-null  int64  
 8   smoke        68808 non-null  int64  
 9   alco         68808 non-null  int64  
 10  active       68808 non-null  int64  
 11  cardio       68808 non-null  int64  
dtypes: float64(1), int64(11)
memory usage: 6.8 MB
```

In [51]:

```
# separating target and other features
x = df.drop(['cardio'],axis=1)
y = df['cardio']
```

In [52]:
```
xcols = x.columns
x = sc.fit_transform(x)
x = pd.DataFrame(x, columns=xcols)
x.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	-0.434070	1.366644	0.444533	-0.845879	-0.120772	-0.142767	-0.537247	-0.395119	-0.310571	-0.238081	0.494667
1	0.309396	-0.731719	-1.021605	0.759185	0.072784	0.890525	2.408324	-0.395119	-0.310571	-0.238081	0.494667
2	-0.246076	-0.731719	0.077998	-0.706308	0.008265	-1.176059	2.408324	-0.395119	-0.310571	-0.238081	-2.021563
3	-0.746042	1.366644	0.566711	0.549829	0.137303	1.923817	-0.537247	-0.395119	-0.310571	-0.238081	0.494667
4	-0.806410	-0.731719	-1.021605	-1.264591	-0.185291	-2.209351	-0.537247	-0.395119	-0.310571	-0.238081	-2.021563

In [53]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,stratify=y, random_state=42, test_size=.3)
```

In [54]:
```
lr = LogisticRegression()
lr.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegressi
on()
```

In [55]:
```
pred = lr.predict(x_test)
```

In [56]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.77      0.74     10424
           1       0.74      0.68      0.71     10219

    accuracy                           0.72     20643
   macro avg       0.72      0.72      0.72     20643
weighted avg       0.72      0.72      0.72     20643
```

In [57]:
```
print('train accuracy: ',accuracy_score(lr.predict(x_train),y_train))
print('test accuracy: ',accuracy_score(lr.predict(x_test),y_test))
```

```
train accuracy:  0.7256514066230666
test accuracy:  0.7220849682701158
```

- There is no big difference between base model and model that created after capping values.

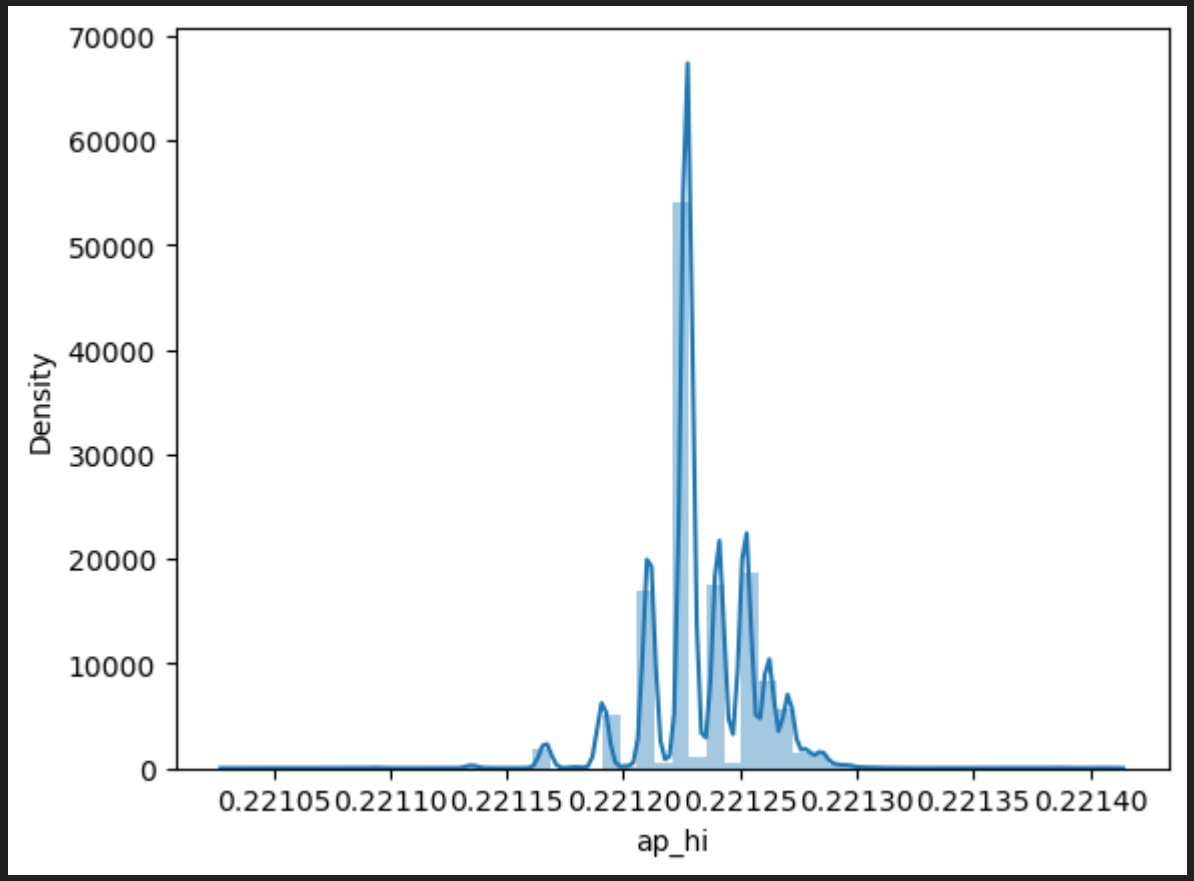
We will transform those two features and will check the performance of the model

In [58]:
```
from scipy.stats import boxcox
```

In [59]:

```
df['ap_hi'] = np.log(df['ap_hi'])
df['ap_hi'],value = boxcox(df['ap_hi'])
sns.distplot(df['ap_hi'])
```

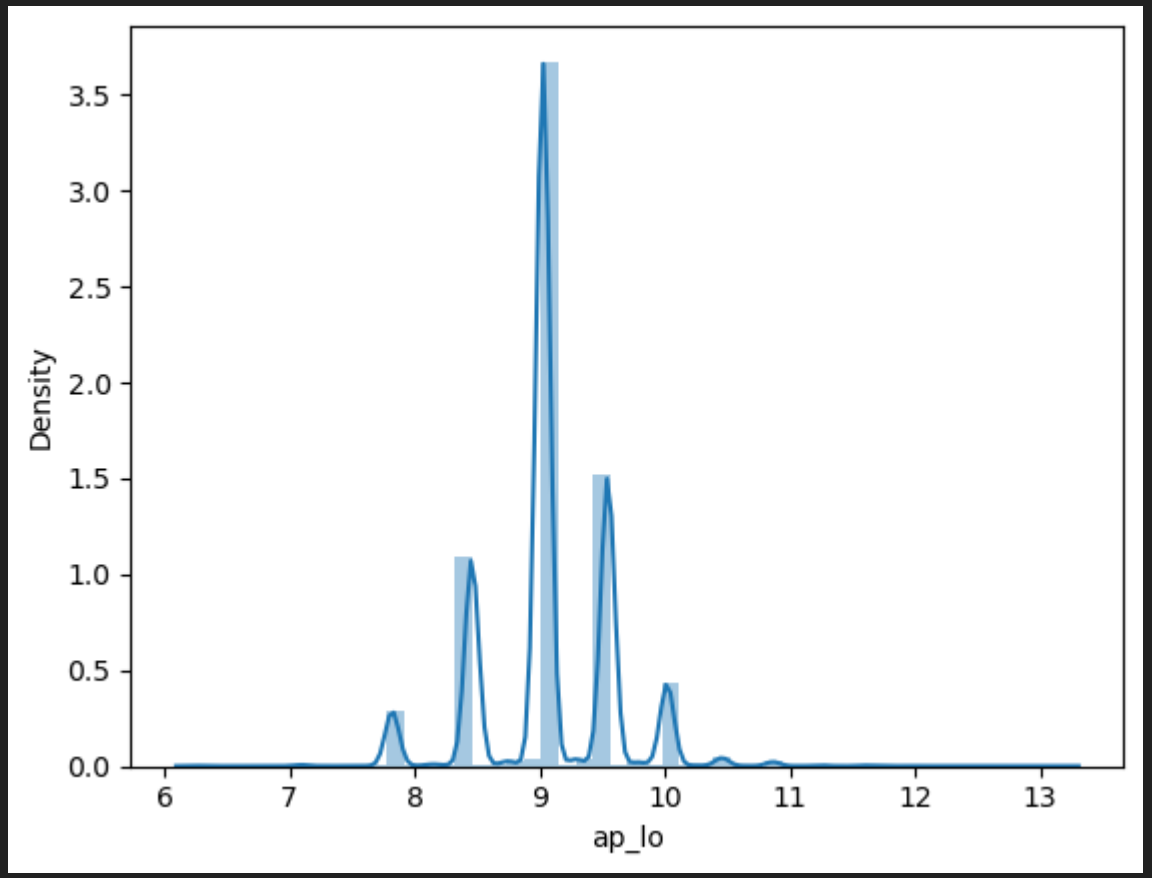
```
<AxesSubplot:xlabel='ap_hi', ylabel='Density'>
```



In [60]:

```
df['ap_lo'] = np.log(df['ap_lo'])
df['ap_lo'],value = boxcox(df['ap_lo'])
sns.distplot(df['ap_lo'])
```

```
<AxesSubplot:xlabel='ap_lo', ylabel='Density'>
```



In [61]:
```
# separating target and other features
x = df.drop(['cardio'],axis=1)
y = df['cardio']
```

In [62]:
```
x.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	18393	2	168	62.0	0.221211	9.028106	1	1	0	0	1
1	20228	1	156	85.0	0.221252	9.545271	3	1	0	0	1
2	18857	1	165	64.0	0.221241	8.458257	3	1	0	0	0
3	17623	2	169	82.0	0.221262	10.019425	1	1	0	0	1
4	17474	1	156	56.0	0.221191	7.822179	1	1	0	0	0

In [63]:
```
xcols = x.columns
x = sc.fit_transform(x)
x = pd.DataFrame(x, columns=xcols)
x.head()
```

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active
0	-0.434070	1.366644	0.444533	-0.845879	-1.033247	-0.099039	-0.537247	-0.395119	-0.310571	-0.238081	0.494667
1	0.309396	-0.731719	-1.021605	0.759185	0.857946	0.908320	2.408324	-0.395119	-0.310571	-0.238081	0.494667
2	-0.246076	-0.731719	0.077998	-0.706308	0.331128	-1.209016	2.408324	-0.395119	-0.310571	-0.238081	-2.021563
3	-0.746042	1.366644	0.566711	0.549829	1.310717	1.831899	-0.537247	-0.395119	-0.310571	-0.238081	0.494667
4	-0.806410	-0.731719	-1.021605	-1.264591	-1.939619	-2.447998	-0.537247	-0.395119	-0.310571	-0.238081	-2.021563

In [64]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y,stratify=y, random_state=42, test_size=.3)
```

In [65]:
```
lr = LogisticRegression()
lr.fit(x_train, y_train)
```

```
LogisticRegression
LogisticRegressi
on()
```

In [66]:
```
pred = lr.predict(x_test)
```

In [67]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.71      0.76      0.74     10424
           1       0.74      0.69      0.71     10219

    accuracy                           0.72     20643
   macro avg       0.73      0.72      0.72     20643
weighted avg       0.73      0.72      0.72     20643
```

In [68]:
```
print('train accuracy: ',accuracy_score(lr.predict(x_train),y_train))
print('test accuracy: ',accuracy_score(lr.predict(x_test),y_test))
```

```
train accuracy:  0.7284127478459462
test accuracy:  0.724991522550017
```

Conclusion

The models are performing same and no issue of overfitting and underfitting.

Using Logistics Regression we were able to make a good model which predict desease.

In [ ]:

```
1
```