In [1]:

```
import numpy as np,pandas as pd,matplotlib.pyplot as plt,seaborn as sns,warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv('cardio.csv',delimiter=';')
```

In [3]:

```
df.head()
```

Out[3]:

	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

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
```

In [5]:

```
df.shape
```

Out[5]:

```
(70000, 13)
```

In [6]:

```
df.isnull().any().sum()
```

Out[6]:

```
0
```

Data Preprocessing

In [7]:

```
df.drop('id',axis=1,inplace=True)
```

In [8]:

```
df['age']=round(df['age']/365)
```

In [9]:

```
df.head()
```

Out[9]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	50.0	2	168	62.0	110	80	1	1	0	0	1	0
1	55.0	1	156	85.0	140	90	3	1	0	0	1	1
2	52.0	1	165	64.0	130	70	3	1	0	0	0	1
3	48.0	2	169	82.0	150	100	1	1	0	0	1	1
4	48.0	1	156	56.0	100	60	1	1	0	0	0	0

In [10]:

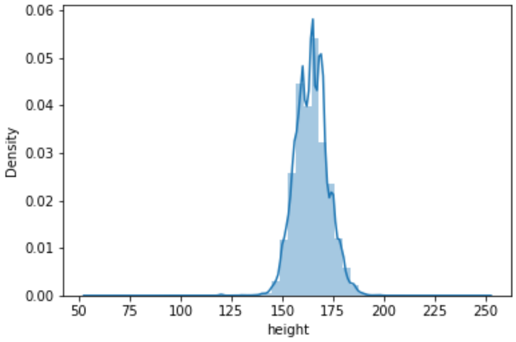
```
df['age']=df['age'].astype(int)
```

In [11]:

```
sns.distplot(df['height'])
```

Out[11]:

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

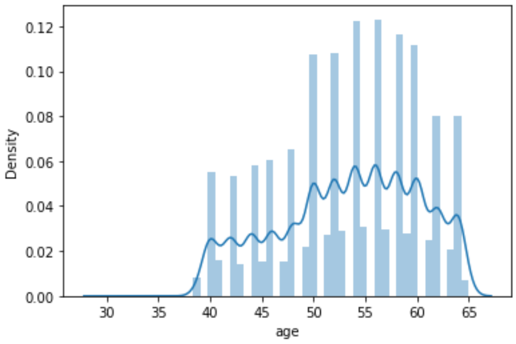


In [12]:

```
sns.distplot(df['age'])
```

Out[12]:

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



In [13]:

```
df[df['weight']<40]
```

Out[13]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
3752	42	1	120	30.00	110	70	1	1	0	0	1	0
5794	48	1	151	37.00	120	80	1	1	0	0	1	0
10447	60	1	162	38.00	100	70	1	1	0	0	1	0
10627	59	1	153	37.00	150	80	3	3	0	0	1	1
11876	48	1	157	39.00	90	70	1	1	0	0	1	0
14722	62	1	143	34.00	100	70	1	1	0	0	1	0
16322	60	1	165	35.00	100	70	1	1	0	0	1	0
16906	47	2	170	31.00	150	90	2	2	0	0	1	1
18559	50	1	160	30.00	120	80	1	1	0	0	1	1
19582	58	1	152	38.00	110	80	1	1	0	0	1	0
22016	42	1	146	32.00	100	70	1	1	0	0	0	0
25198	51	1	149	35.45	110	70	1	1	0	0	1	0
26806	64	1	157	23.00	110	80	1	1	0	0	1	0
29333	60	1	153	37.00	120	80	1	1	0	0	1	1
29488	56	2	177	22.00	120	80	1	1	1	1	1	0
31420	56	1	140	39.00	120	80	1	1	0	0	1	0
32087	43	1	143	36.00	90	60	1	1	0	0	1	0
33478	42	1	152	39.00	110	70	1	2	1	0	1	1
33511	58	1	153	34.00	110	70	3	3	0	0	1	1
33817	59	2	178	11.00	130	90	1	1	0	0	1	1
33820	62	1	145	36.00	120	80	1	1	0	0	1	0
34276	40	2	128	28.00	120	80	1	1	0	0	1	0
34282	56	1	148	36.00	140	80	1	1	0	0	1	1
34328	40	1	152	39.00	90	60	1	1	0	0	1	0
35314	54	1	146	32.00	130	80	1	2	0	0	0	0
38417	60	1	154	32.00	110	60	1	1	0	0	1	0
38743	58	1	152	38.00	150	1000	1	1	0	0	1	1
40612	64	1	154	38.00	90	70	1	2	0	0	1	0
41353	57	1	157	38.00	120	80	1	1	0	0	1	1
41905	58	1	143	30.00	103	61	2	1	0	0	1	0
43759	49	1	135	37.00	150	90	2	3	0	0	1	0
44138	60	1	151	38.00	100	70	1	1	0	0	1	1
44622	50	1	150	39.00	130	90	1	1	0	0	1	0
48080	53	1	143	33.00	100	60	1	1	0	0	1	0
48613	56	1	144	36.00	100	70	1	1	0	0	1	0
51411	58	1	155	37.00	110	70	1	1	0	0	0	1
51544	60	1	151	38.00	120	80	1	1	0	1	0	0
51837	54	2	139	34.00	120	70	1	1	0	0	1	0
53224	56	1	133	36.00	100	60	1	1	0	0	0	0
53945	62	1	156	39.00	90	60	1	1	0	0	1	0
54017	62	1	134	37.00	180	90	1	1	0	0	1	0
54682	56	1	153	39.00	110	70	1	1	0	0	1	0
55339	65	1	147	39.00	120	80	1	1	0	0	1	1
55852	64	1	152	34.00	140	90	1	1	0	0	1	1
56914	52	1	152	37.00	90	50	1	1	0	0	1	0
57858	52	2	165	10.00	180	1100	2	2	0	0	1	1
58200	62	1	169	35.00	140	90	2	1	0	0	1	1
60188	60	1	162	21.00	120	80	2	1	0	0	1	1
60699	52	1	171	29.00	110	70	2	1	0	0	1	1
63113	56	1	158	39.00	90	60	1	1	0	0	0	1
65082	62	1	145	33.00	130	1000	2	1	0	0	1	1
65650	52	1	147	38.00	100	70	3	3	0	0	1	0

In [14]:

```
df[df['weight']<40].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 52 entries, 3752 to 65650
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   age          52 non-null     int32  
 1   gender       52 non-null     int64  
 2   height       52 non-null     int64  
 3   weight       52 non-null     float64
 4   ap_hi        52 non-null     int64  
 5   ap_lo        52 non-null     int64  
 6   cholesterol  52 non-null     int64  
 7   gluc         52 non-null     int64  
 8   smoke        52 non-null     int64  
 9   alco         52 non-null     int64  
 10  active       52 non-null     int64  
 11  cardio       52 non-null     int64  
dtypes: float64(1), int32(1), int64(10)
memory usage: 5.1 KB
```

In [15]:

```
df['ap_lo'].unique()
```

Out[15]:

```
array([   80,    90,    70,   100,    60,    85,    89,   110,    65,
          63,    79,  1100,  1000,   800,   120,    50,    30,   109,
          84,  1033,   150,    91,    40,    73,    78,    75,    86,
          87,  1001,    82,    95,    69,    74,    97,    81,  1200,
          83,   119,     0,    93,   105, 10000,    99,    77,    59,
        8044,   140,    92,  1044,   108,   125,   115,    68,    61,
         106,   102,    94,    66,    52,   170,    76,   160,    62,
          96,   130,   113,    67,  9100,    10,    88,   902,     8,
         112,   104,    71,    72,  1008,    98,  2088,    20,   802,
        8000,  1022,   850,   708,    57,   101,  9011,  1011,    64,
        1007,  1177,  7100,    45,   709,  8500,    58,  1110,  8099,
        1088,   126,  1077,  1120,     7,   103,  1125,   180,   121,
        8100,   710,  5700,  8079,  1111,  1003,     6,  1900,   809,
         114,   801,  1002,    53,   111,     1,   118,    56,   182,
         810,     9,  7099, 11000,  9800,  8200,  1139,   107,   820,
          55,  1400,   190,   900,   122,  6800,   135,   700,    15,
        1101,   910,  1140,  1211,   -70,    54,  8077,   901,   880,
         870,   585,    49,   602], dtype=int64)
```

Deleting ap_hi with less than 80

In [16]:

```
df[df['ap_hi']<80]
```

Out[16]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
567	58	1	168	78.0	14	90	2	1	0	0	1	1
636	56	2	169	68.0	70	110	1	1	0	0	1	0
927	60	2	175	70.0	14	90	3	1	0	0	1	1
979	50	1	172	65.0	11	80	1	3	0	0	1	0
1600	53	1	165	66.0	12	80	1	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...
68630	58	1	160	59.0	12	80	1	1	0	0	1	0
68742	51	1	158	74.0	14	90	1	1	0	0	1	1
68998	52	1	154	77.0	14	90	1	1	0	0	1	0
69137	42	2	176	65.0	12	80	1	1	0	0	1	0
69549	58	1	155	69.0	13	90	1	3	0	0	1	1

207 rows × 12 columns

In [17]:

```
df=df[df['ap_hi']>80]
```

Deleting ap_hi with more than 250

In [18]:

```
df[df['ap_hi']>250]
```

Out[18]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
1876	41	1	160	60.0	902	60	1	1	0	0	1	0
2014	62	2	167	59.0	906	0	1	1	0	0	1	0
4817	40	1	168	63.0	909	60	2	1	0	0	1	0
7763	58	1	175	80.0	11500	90	1	1	0	0	1	1
8915	52	1	164	75.0	1420	80	2	1	0	0	1	1
9557	62	1	155	87.0	701	110	1	1	0	0	1	1
13895	44	1	168	72.0	1500	80	1	1	0	0	1	1
17713	61	2	163	50.0	907	70	3	3	0	0	1	1
23867	44	1	161	92.0	906	0	2	1	0	0	1	1
25464	43	2	169	75.0	14020	80	2	1	0	0	1	1
25519	59	1	169	71.0	14020	80	3	3	0	0	1	1
25780	50	1	157	83.0	1400	80	1	1	0	0	1	1
28147	54	2	175	87.0	1620	80	2	1	0	0	1	1
29665	48	1	154	65.0	907	70	1	1	0	0	1	0
31783	44	1	170	64.0	907	0	1	1	0	0	1	0
36894	56	2	175	78.0	1130	90	1	1	0	0	1	1
40330	56	1	162	50.0	309	0	1	1	0	0	1	0
40831	54	1	162	67.0	401	80	1	3	0	0	1	1
40852	48	1	169	70.0	16020	80	1	1	0	0	0	1
41095	58	1	160	60.0	1202	80	1	1	0	0	1	1
41505	57	1	154	41.0	806	0	1	1	0	0	1	0
42397	49	2	176	69.0	906	0	1	1	0	0	1	0
42658	56	2	182	80.0	906	60	1	1	0	0	1	1
43133	57	2	170	78.0	1400	90	2	1	0	0	1	0
43208	64	1	165	67.0	1420	80	2	1	0	0	1	1
43504	54	1	158	62.0	1300	80	3	1	0	1	1	1
46912	46	2	180	78.0	14020	90	1	1	0	0	1	1
47253	54	1	160	65.0	14020	90	1	1	0	0	1	0
48795	60	1	156	76.0	1400	90	1	1	0	0	1	1
50836	46	2	164	66.0	1409	90	1	1	0	0	1	1
51438	51	2	168	65.0	11020	80	1	1	0	0	1	1
53982	51	1	164	54.0	960	60	1	1	0	0	1	0
55459	50	1	152	76.0	13010	80	2	2	0	0	1	1
55847	64	1	161	105.0	13010	80	1	1	0	0	0	0
57291	60	2	166	73.0	1300	90	1	1	0	1	1	0
57918	64	1	153	63.0	1110	80	1	1	0	0	0	1
63996	46	1	168	69.0	1205	90	1	1	0	0	0	1
64911	46	1	157	78.0	906	60	2	1	0	0	1	0
68663	50	1	156	41.0	906	0	1	1	0	0	1	0
69370	40	1	170	74.0	2000	100	2	1	0	0	1	1

In [19]:

```
df=df[df['ap_hi']<250]
```

Deleting ap_lo with less than 60

In [20]:

```
df[df['ap_lo']<60]
```

Out[20]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
418	46	1	157	72.0	150	30	1	1	0	0	1	1
481	54	1	158	86.0	160	50	2	3	0	0	1	1
507	46	1	165	64.0	140	40	1	1	0	0	1	1
1960	48	1	155	50.0	90	50	1	1	0	0	1	0
2183	62	2	174	65.0	110	50	1	1	1	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
68135	42	1	163	63.0	162	52	1	1	0	0	1	1
68223	52	2	173	100.0	130	20	1	1	0	0	1	1
68343	60	1	151	80.0	130	50	2	2	0	0	1	0
68370	52	2	159	68.0	100	50	2	1	1	0	1	0
68568	42	1	163	71.0	110	6	1	1	0	0	1	0

147 rows × 12 columns

In [21]:

```
df=df[df['ap_lo']>60]
```

Deleting ap_lo with more than 150

In [22]:

```
df[df['ap_lo']>150]
```

Out[22]:

	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
228	48	2	183	98.0	160	1100	1	2	1	0	1	1
241	60	2	157	60.0	160	1000	2	1	0	0	0	1
260	50	1	150	83.0	140	800	1	1	0	0	1	1
329	64	1	176	63.0	160	1000	2	2	0	0	0	1
345	51	1	154	81.0	140	1000	2	1	0	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
69771	64	1	167	81.0	160	1000	1	1	0	0	1	1
69872	60	1	152	56.0	160	1000	1	1	0	0	1	1
69878	58	2	168	95.0	160	1000	1	1	0	0	1	1
69885	61	2	166	78.0	170	1000	1	1	0	0	0	0
69967	59	2	168	63.0	140	1000	1	1	0	0	1	1

968 rows × 12 columns

In [23]:

```
df=df[df['ap_lo']<150]
```

In [24]:

```
df.shape
```

Out[24]:

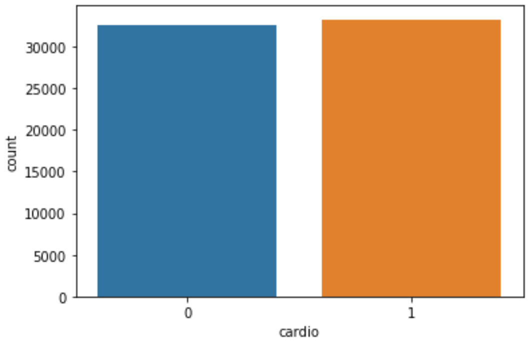
```
(65858, 12)
```

In [25]:

```
sns.countplot(df['cardio'])
```

Out[25]:

```
<AxesSubplot:xlabel='cardio', ylabel='count'>
```



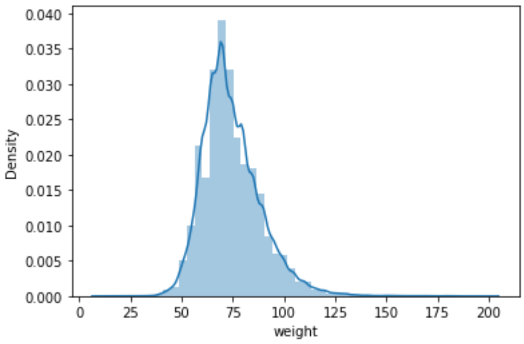
- By seeing above chart we can say that dataset is balanced
- Possibility of death chances is more than its vice versa

In [26]:

```
sns.distplot(df['weight'])
```

Out[26]:

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

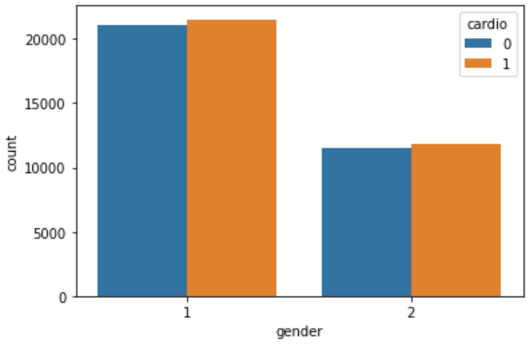


In [27]:

```
sns.countplot(df['gender'],hue=df['cardio'])
```

Out[27]:

```
<AxesSubplot:xlabel='gender', ylabel='count'>
```



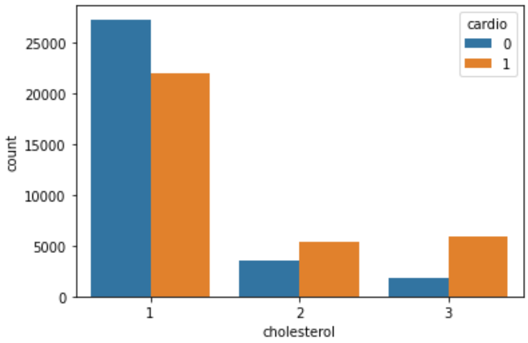
The rate of cardio is nearly equal in both genders

In [28]:

```
sns.countplot(df['cholesterol'],hue=df['cardio'])
```

Out[28]:

<AxesSubplot:xlabel='cholesterol', ylabel='count'>



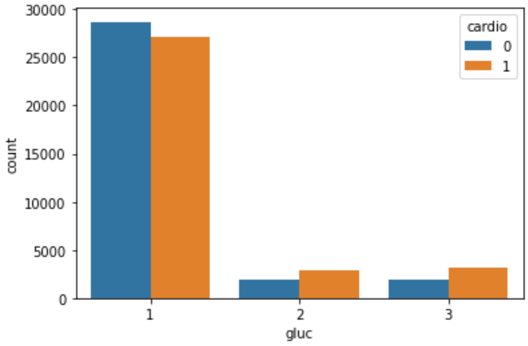
The people who are above normal have high rate of cardiac arrest as compared to others

In [29]:

```
sns.countplot(df['gluc'],hue=df['cardio'])
```

Out[29]:

<AxesSubplot:xlabel='gluc', ylabel='count'>

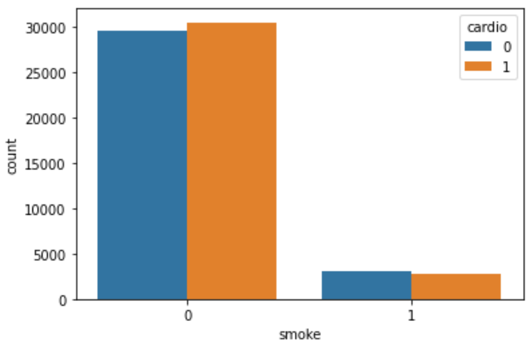


In [30]:

```
sns.countplot(df['smoke'],hue=df['cardio'])
```

Out[30]:

<AxesSubplot:xlabel='smoke', ylabel='count'>



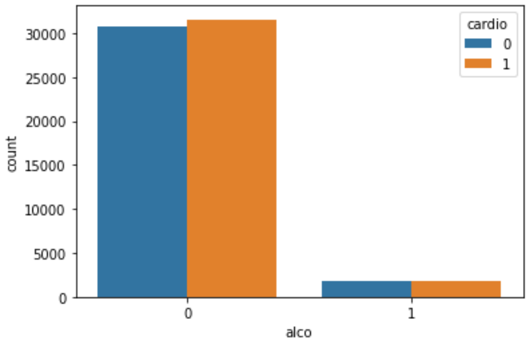
People who desn't smoke has high amount or rate of cardio

In [31]:

```
sns.countplot(df['alco'],hue=df['cardio'])
```

Out[31]:

<AxesSubplot:xlabel='alco', ylabel='count'>

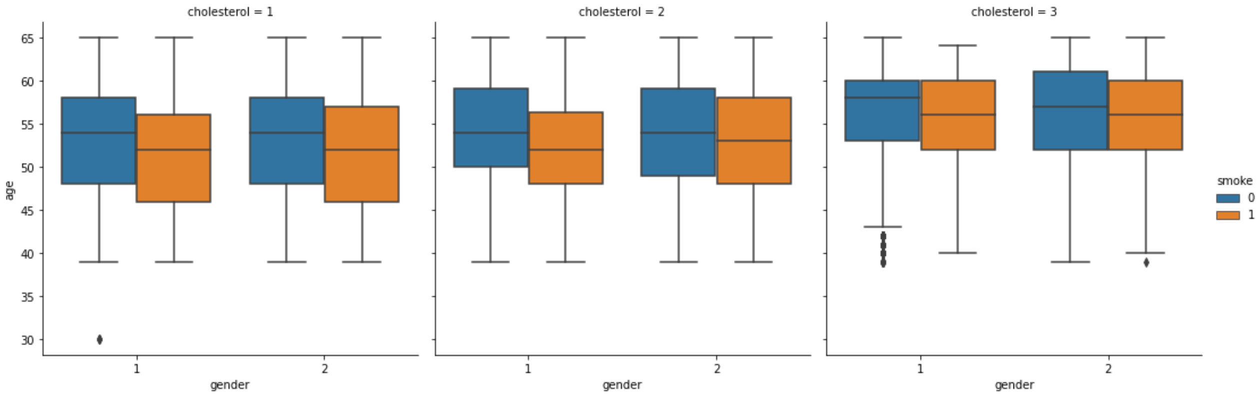


In [32]:

```
sns.catplot(x='gender',y='age',hue='smoke',col='cholesterol',kind='box',data=df)
```

Out[32]:

<seaborn.axisgrid.FacetGrid at 0x10f877b62c0>

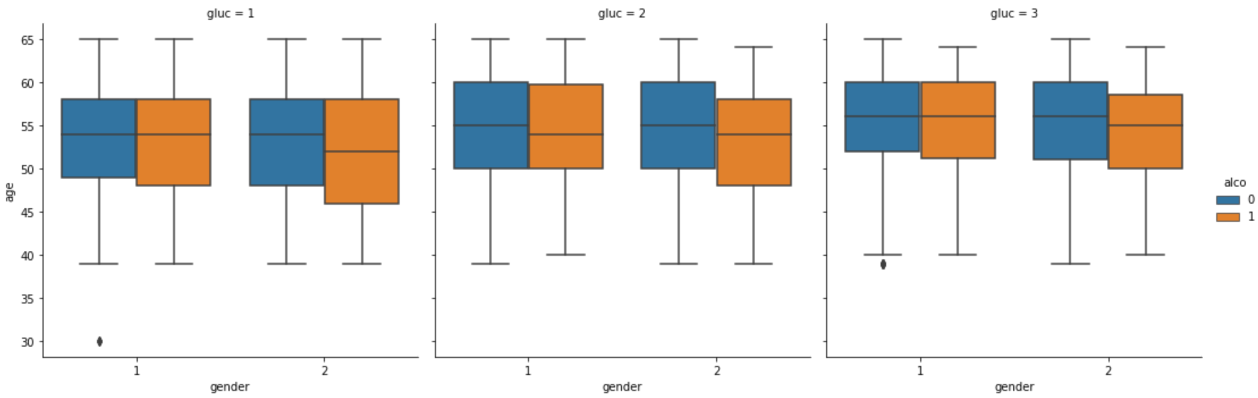


In [33]:

```
sns.catplot(x='gender',y='age',hue='alco',col='gluc',kind='box',data=df)
```

Out[33]:

<seaborn.axisgrid.FacetGrid at 0x10f87919540>

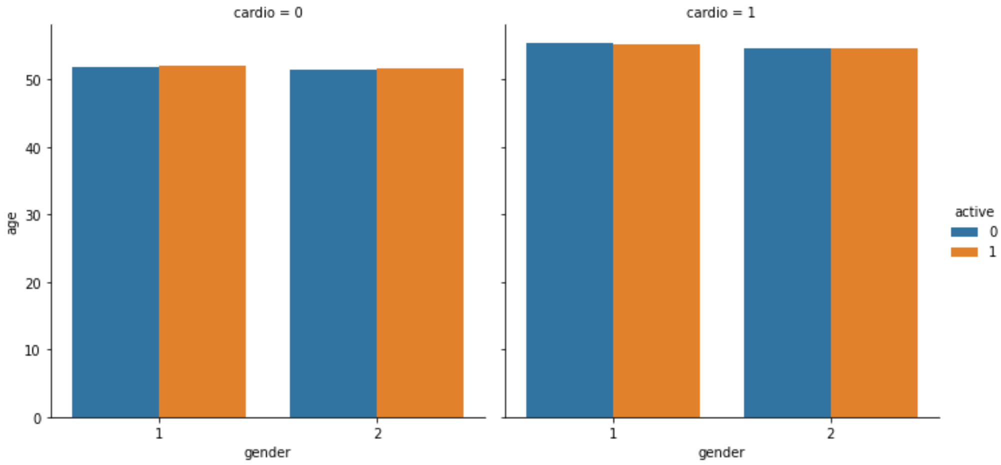


In [34]:

```
sns.catplot(x='gender',y='age',hue='active',col='cardio',kind='bar',data=df,ci=False)
```

Out[34]:

```
<seaborn.axisgrid.FacetGrid at 0x10f8e610f70>
```

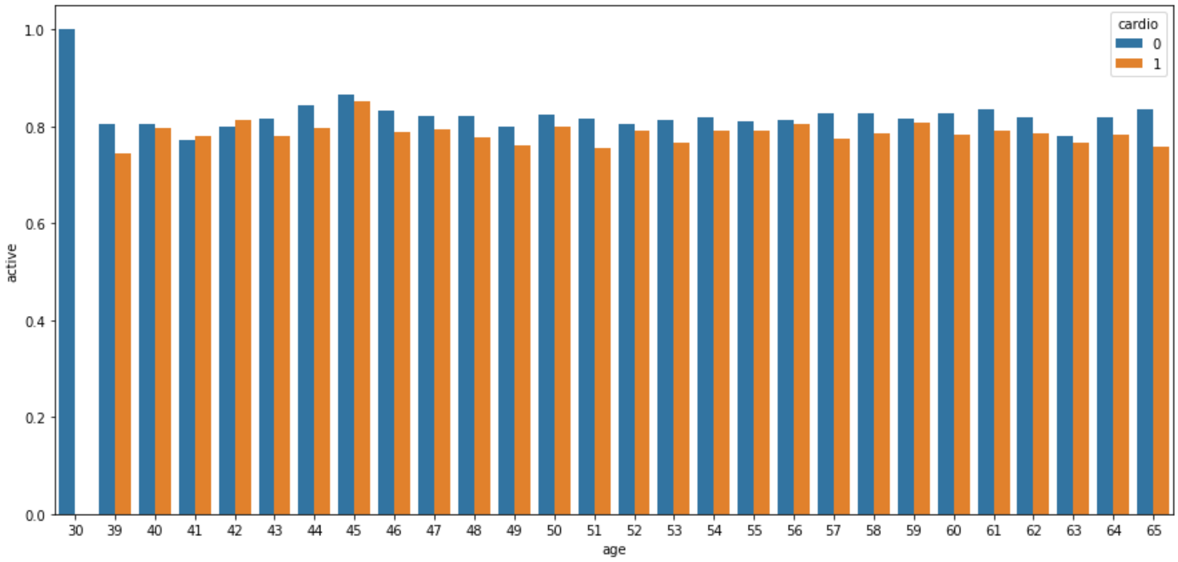


In [35]:

```
plt.figure(figsize=(15,7))
sns.barplot(df['age'],df['active'],hue=df['cardio'],ci=False)
```

Out[35]:

```
<AxesSubplot:xlabel='age', ylabel='active'>
```



In [36]:

```
col=['gluc','alco','smoke','cholesterol','active']
data=pd.melt(df,id_vars='cardio',value_vars=df[col])
data
```

Out[36]:

	cardio	variable	value
0	0	gluc	1
1	1	gluc	1
2	1	gluc	1
3	1	gluc	1
4	0	gluc	2
...	...	...	...
329285	0	active	1
329286	1	active	1
329287	1	active	0
329288	1	active	0
329289	0	active	1

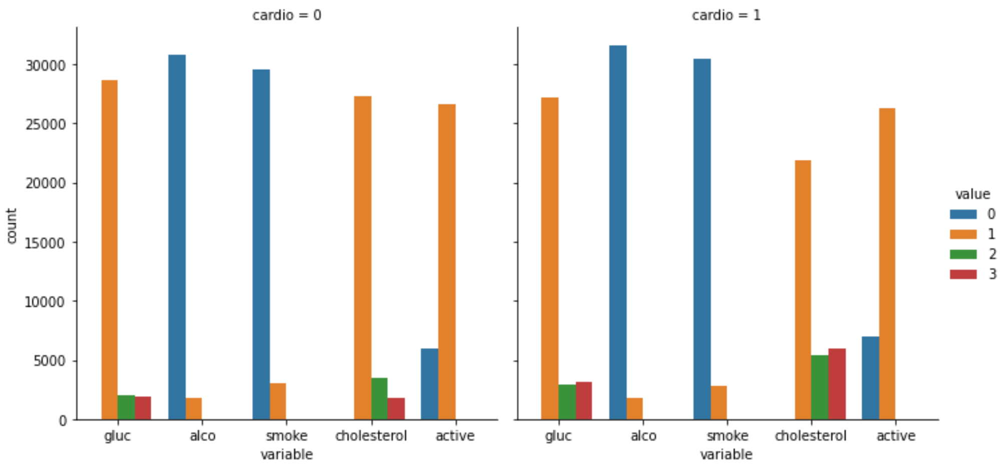
329290 rows × 3 columns

In [37]:

```
sns.catplot(x='variable',hue='value',col='cardio',kind='count',data=data)
```

Out[37]:

```
<seaborn.axisgrid.FacetGrid at 0x10f8eaa9060>
```



In [38]:

```
df.isnull().any()
```

Out[38]:

```
age            False
gender         False
height         False
weight         False
ap_hi          False
ap_lo          False
cholesterol    False
gluc           False
smoke          False
alco           False
active         False
cardio         False
dtype: bool
```

Splitting data into train and test data

In [39]:

```
x=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [40]:
```
from sklearn.model_selection import train_test_split
```

In [41]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=1)
```

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

Out[42]:
```
(52686, 11)
```

In [43]:
```
x_test.shape
```

Out[43]:
```
(13172, 11)
```

In [44]:
```
y_train.shape
```

Out[44]:
```
(52686,)
```

In [45]:
```
y_test.shape
```

Out[45]:
```
(13172,)
```

Feature Scaling

In [46]:
```
from sklearn.preprocessing import StandardScaler
```

In [47]:
```
sc=StandardScaler()
```

In [48]:
```
x_train=sc.fit_transform(x_train)
```

In [49]:
```
x_test=sc.transform(x_test)
```

Model Building

In [50]:
```
from sklearn.linear_model import LogisticRegression
```

In [51]:
```
reg=LogisticRegression()
```

In [52]:
```
reg.fit(x_train,y_train)
```

Out[52]:
```
▾ LogisticRegression
LogisticRegression()
```

In [53]:
```
y_pred_train=reg.predict(x_train)
```

In [54]:
```
y_pred_test=reg.predict(x_test)
```

Evaluation of model

In [55]:
```
from sklearn.metrics import accuracy_score
```

In [56]:
```
print('Train Data')
print(accuracy_score(y_train,y_pred_train))
```

```
Train Data
0.7256386895949588
```

In [57]:
```
print('Test Data')
print(accuracy_score(y_test,y_pred_test))
```

```
Test Data
0.7230488915882174
```

In [58]:
```
from sklearn.metrics import confusion_matrix
```

In [59]:
```
print('Train Data')
print(confusion_matrix(y_train,y_pred_train))
print('Test Data')
print(confusion_matrix(y_test,y_pred_test))
```

```
Train Data
[[20377  5682]
 [ 8773 17854]]
Test Data
[[5049 1460]
 [2188 4475]]
```

In [60]:
```
from sklearn.model_selection import cross_val_score
```

In [61]:
```
reg_score=cross_val_score(reg,x,y,scoring='accuracy',cv=5)
```

In [62]:
```
reg_score
```

Out[62]:
```
array([0.70930762, 0.71796234, 0.71219253, 0.71870017, 0.70586895])
```

In [63]:
```
mean_reg_score=np.mean(reg_score)
```

In [64]:
```
mean_reg_score
```

Out[64]:
```
0.7128063250702877
```

In [65]:
```
from sklearn.metrics import classification_report,recall_score,precision_score,f1_score,roc_auc_score
```

In [66]:

```
print('Train data')
print(classification_report(y_train,y_pred_train))
```

```
Train data
              precision    recall  f1-score   support

           0       0.70      0.78      0.74     26059
           1       0.76      0.67      0.71     26627

    accuracy                           0.73     52686
   macro avg       0.73      0.73      0.73     52686
weighted avg       0.73      0.73      0.72     52686
```

In [67]:

```
print('Test data')
print(classification_report(y_test,y_pred_test))
```

```
Test data
              precision    recall  f1-score   support

           0       0.70      0.78      0.73      6509
           1       0.75      0.67      0.71      6663

    accuracy                           0.72     13172
   macro avg       0.73      0.72      0.72     13172
weighted avg       0.73      0.72      0.72     13172
```

In [68]:

```
print('Train data')
print(recall_score(y_train,y_pred_train))
print('Test data')
print(recall_score(y_test,y_pred_test))
```

```
Train data
0.6705224020730837
Test data
0.6716193906648656
```

In [69]:

```
print('Train data')
print(precision_score(y_train,y_pred_train))
print('Test data')
print(precision_score(y_test,y_pred_test))
```

```
Train data
0.7585825968728755
Test data
0.7540016849199663
```

In [70]:

```
print('Train data')
print(f1_score(y_train,y_pred_train))
print('Test data')
print(f1_score(y_test,y_pred_test))
```

```
Train data
0.7118394035444451
Test data
0.7104302270201619
```

In [71]:

```
y_train_proba=reg.predict_proba(x_train)[:,1]
y_test_proba=reg.predict_proba(x_test)[:,1]
```

In [72]:

```
from sklearn.metrics import precision_recall_curve
```

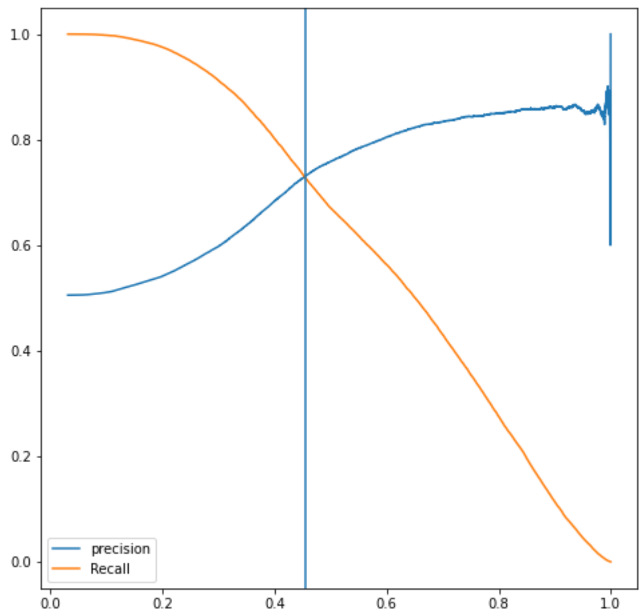
In [73]:

```
p,r,th=precision_recall_curve(y_train,y_train_proba)
```

In [74]:

```
print('                          PR CURVE')
plt.figure(figsize=(8,8))
sns.lineplot(th,p[:-1],label='precision')
sns.lineplot(th,r[:-1],label='Recall')
plt.axvline(0.455)
plt.show()
```

```
                          PR CURVE
```



In [75]:

```
def metrics(y_actual,y_proba,th):
    y_pred_temp=[1 if p>th else 0 for p in y_proba]
    accuracy=accuracy_score(y_actual,y_pred_temp)
    recall=recall_score(y_actual,y_pred_temp)
    precision=precision_score(y_actual,y_pred_temp)
    f1=f1_score(y_actual,y_pred_temp)
    roc_auc=roc_auc_score(y_actual,y_pred_temp)
    return {'Accuracy':accuracy,'Recall':recall,'Precision':precision,'F1':f1,'ROC_AUC':roc_auc}
```

In [76]:

```
metrics(y_train,y_train_proba,0.455)
```

Out[76]:

```
{'Accuracy': 0.7268913942982955,
 'Recall': 0.7275322041536786,
 'Precision': 0.7308533916849015,
 'F1': 0.72918901624226,
 'ROC_AUC': 0.7268844105307324}
```

In [77]:

```
metrics(y_test,y_test_proba,0.455)
```

Out[77]:

```
{'Accuracy': 0.7260856361979957,
 'Recall': 0.7306018310070539,
 'Precision': 0.7286334381080677,
 'F1': 0.7296163069544365,
 'ROC_AUC': 0.7260322106333473}
```

```
#ROC-AUC curve
from sklearn.metrics import roc_curve
fpr,tpr,th=roc_curve(y_train,y_train_proba)
sns.lineplot(fpr,tpr)
sns.lineplot([0.0,1.0],[0.0,1.0],color='r',linestyle='--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC-AUC Curve')
```

In [78]:

```
from sklearn.neighbors import KNeighborsClassifier
```

```
for i in range(1,16):
```

```
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,y_train)
    y_pred_train=knn.predict(x_train)
    y_pred_test=knn.predict(x_test)
    print('When neighbors is',i)
    print('Train Data')
    print(accuracy_score(y_train,y_pred_train))
    print('Test Data')
    print(accuracy_score(y_test,y_pred_test))
    print('*'*60)
```

In [ ]:

```
knn=KNeighborsClassifier(n_neighbors=13)
knn.fit(x_train,y_train)
y_pred_train=knn.predict(x_train)
y_pred_test=knn.predict(x_test)
print('Train Data')
print(accuracy_score(y_train,y_pred_train))
print('Test Data')
print(accuracy_score(y_test,y_pred_test))
```

```
p_g={'n_neighbors':np.arange(1,15),
   'metric':['minkowski','manhattan','euclidean'],
    'weights':['uniform','distance']}
```

```
from sklearn.model_selection import GridSearchCV
```

```
model=GridSearchCV(knn,p_g,cv=5,scoring='accuracy',n_jobs=-1)
```

```
model.fit(x_train,y_train)
```

```
model.best_params_
```

```
y_pred_modeltr=model.predict(x_train)
y_pred_modelte=model.predict(x_test)
```

```
print('Train Data')
print(accuracy_score(y_train,y_pred_modeltr))
print('Test Data')
print(accuracy_score(y_test,y_pred_modelte))
```

In [79]:

```
from sklearn.ensemble import AdaBoostClassifier
ad=AdaBoostClassifier()
```

In [80]:

```
ad.fit(x_train,y_train)
```

Out[80]:

```
▾ AdaBoostClassifier
AdaBoostClassifier()
```

In [81]:

```
y_pred_train_modelad=ad.predict(x_train)
y_pred_test_modelad=ad.predict(x_test)
```

In [82]:

```
print('Train Data')
print(accuracy_score(y_train,y_pred_train_modelad))
print('Test Data')
print(accuracy_score(y_test,y_pred_test_modelad))
```

```
Train Data
0.7286186083589569
Test Data
0.7234284846644398
```

In [83]:

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
```

In [84]:

```
rf.fit(x_train,y_train)
```

Out[84]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [85]:

```
y_pred_train=rf.predict(x_train)
y_pred_test=rf.predict(x_test)
```

In [86]:

```
print('Train Data')
print(accuracy_score(y_train,y_pred_train))
print('Test Data')
print(accuracy_score(y_test,y_pred_test))
```

```
Train Data
0.9747750825646282
Test Data
0.7033100516246583
```

In [87]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB,MultinomialNB
from sklearn.ensemble import AdaBoostClassifier
reg=LogisticRegression()
knn=KNeighborsClassifier()
dt=DecisionTreeClassifier()
rf=RandomForestClassifier()
b=BernoulliNB()
m=MultinomialNB()
ad=AdaBoostClassifier()
```

In [88]:

```
from sklearn.metrics import classification_report
def my_model(model):
    model.fit(x_train,y_train)
    y_pred_train=model.predict(x_train)
    y_pred_test=model.predict(x_test)
    print('Train Data')
    print(classification_report(y_train,y_pred_train))
    print('Test Data')
    print(classification_report(y_test,y_pred_test))
    return model
```

In [89]:

```
for i in [reg,knn,dt,rf,b,ad]:
    print('when my model is:',i)
    my_model(i)
    print('*'*90)
```

```
when my model is: LogisticRegression()
Train Data
              precision    recall  f1-score   support

           0       0.70      0.78      0.74     26059
           1       0.76      0.67      0.71     26627

    accuracy                           0.73     52686
   macro avg       0.73      0.73      0.73     52686
weighted avg       0.73      0.73      0.72     52686

Test Data
              precision    recall  f1-score   support

           0       0.70      0.78      0.73      6509
           1       0.75      0.67      0.71      6663

    accuracy                           0.72     13172
   macro avg       0.73      0.72      0.72     13172
weighted avg       0.73      0.72      0.72     13172

******************************************************************************************
when my model is: KNeighborsClassifier()
Train Data
              precision    recall  f1-score   support

           0       0.78      0.79      0.78     26059
           1       0.79      0.78      0.78     26627

    accuracy                           0.78     52686
   macro avg       0.78      0.78      0.78     52686
weighted avg       0.78      0.78      0.78     52686

Test Data
              precision    recall  f1-score   support

           0       0.69      0.69      0.69      6509
           1       0.70      0.70      0.70      6663

    accuracy                           0.69     13172
   macro avg       0.69      0.69      0.69     13172
weighted avg       0.69      0.69      0.69     13172

******************************************************************************************
when my model is: DecisionTreeClassifier()
Train Data
              precision    recall  f1-score   support

           0       0.96      0.99      0.98     26059
           1       0.99      0.96      0.97     26627

    accuracy                           0.97     52686
   macro avg       0.98      0.97      0.97     52686
weighted avg       0.98      0.97      0.97     52686

Test Data
              precision    recall  f1-score   support

           0       0.63      0.64      0.64      6509
           1       0.64      0.63      0.64      6663

    accuracy                           0.64     13172
   macro avg       0.64      0.64      0.64     13172
weighted avg       0.64      0.64      0.64     13172

******************************************************************************************
when my model is: RandomForestClassifier()
Train Data
              precision    recall  f1-score   support

           0       0.97      0.98      0.97     26059
           1       0.98      0.97      0.97     26627

    accuracy                           0.97     52686
   macro avg       0.97      0.97      0.97     52686
weighted avg       0.97      0.97      0.97     52686

Test Data
              precision    recall  f1-score   support

           0       0.70      0.69      0.70      6509
           1       0.70      0.71      0.71      6663

    accuracy                           0.70     13172
   macro avg       0.70      0.70      0.70     13172
weighted avg       0.70      0.70      0.70     13172

******************************************************************************************
when my model is: BernoulliNB()
Train Data
              precision    recall  f1-score   support
```

```
           0       0.69      0.76      0.72     26059
           1       0.74      0.66      0.70     26627

    accuracy                           0.71     52686
   macro avg       0.72      0.71      0.71     52686
weighted avg       0.72      0.71      0.71     52686

Test Data
              precision    recall  f1-score   support

           0       0.68      0.76      0.72      6509
           1       0.74      0.66      0.69      6663

    accuracy                           0.71     13172
   macro avg       0.71      0.71      0.71     13172
weighted avg       0.71      0.71      0.71     13172

**********************************************************************************************
when my model is: AdaBoostClassifier()
Train Data
              precision    recall  f1-score   support

           0       0.70      0.80      0.74     26059
           1       0.77      0.66      0.71     26627

    accuracy                           0.73     52686
   macro avg       0.73      0.73      0.73     52686
weighted avg       0.73      0.73      0.73     52686

Test Data
              precision    recall  f1-score   support

           0       0.69      0.79      0.74      6509
           1       0.76      0.66      0.71      6663

    accuracy                           0.72     13172
   macro avg       0.73      0.72      0.72     13172
weighted avg       0.73      0.72      0.72     13172

**********************************************************************************************
```

In [ ]:

```

```