PREDICTION OF CHRONIC KIDNEY DISEASE

In [1]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import plotly.express as px
import warnings
warnings.filterwarnings("ignore")

plt.style.use('fivethirtyeight')
%matplotlib inline
pd.set_option('display.max_columns', 26)
```

Then load data and using pandas Function Convert it into DataFrame

In [2]:

```
df=pd.read_csv('kidney_disease.csv')
df
```

Out[2]:

je	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot
.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN
.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN
.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN
.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5
.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.0	49.0	0.5	150.0	4.9
.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.0	31.0	1.2	141.0	3.5
.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.0	26.0	0.6	137.0	4.4
.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.0	50.0	1.0	135.0	4.9
.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.0	18.0	1.1	141.0	3.5

5 columns

Data Set Information:

We use the following representation to collect the dataset age - age bp - blood pressure sg - specific gravity al - albumin su - sugar rbc - red blood cells pc - pus cell pcc - pus cell clumps ba - bacteria bgr - blood glucose random bu - blood urea sc - serum creatinine sod - sodium pot - potassium hemo - hemoglobin pcv - packed cell volume wc - white blood cell count rc - red blood cell count htn - hypertension dm - diabetes mellitus cad - coronary artery disease appet - appetite pe - pedal edema ane - anemia

Attribute Information:

We use 24 + class = 25 (11 numeric ,14 nominal) 1.Age(numerical) age in years 2.Blood Pressure(numerical) bp in mm/Hg 3.Specific Gravity(nominal) sg - (1.005,1.010,1.015,1.020,1.025) 4.Albumin(nominal) al - (0,1,2,3,4,5) 5.Sugar(nominal) su - (0,1,2,3,4,5) 6.Red Blood Cells(nominal) rbc - (normal,abnormal) 7.Pus Cell (nominal) pc - (normal,abnormal) 8.Pus Cell clumps(nominal) pcc - (present,notpresent) 9.Bacteria(nominal) ba - (present,notpresent) 10.Blood Glucose Random(numerical) bgr in mgs/dl 11.Blood Urea(numerical) bu in mgs/dl 12.Serum Creatinine(numerical) sc in mgs/dl 13.Sodium(numerical) sod in mEq/L 14.Potassium(numerical) pot in mEq/L 15.Hemoglobin(numerical) hemo in gms 16.Packed Cell Volume(numerical) 17.White Blood Cell Count(numerical) wc in cells/cumm 18.Red Blood Cell Count(numerical) rc in millions/cmm 19.Hypertension(nominal) htn - (yes,no) 20.Diabetes Mellitus(nominal) dm - (yes,no) 21.Coronary Artery Disease(nominal) cad - (yes,no) 22.Appetite(nominal) appet - (good,poor) 23.Pedal Edema(nominal) pe - (yes,no) 24.Anemia(nominal) ane - (yes,no) 25.Class (nominal) class - (ckd,notckd)

DATA CLEANNING AND EDA PROCESS

Using other pandas Functions like .isnull().sum() , .value_counts() Find out messy or Null values.

In [3]:

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

Out[3]:

```
id                  0
age                 9
bp                 12
sg                 47
al                 46
su                 49
rbc               152
pc                 65
pcc                 4
ba                  4
bgr                44
bu                 19
sc                 17
sod                87
pot                88
hemo               52
pcv                70
wc                105
rc                130
htn                 2
dm                  2
cad                 2
appet               1
pe                  1
ane                 1
classification      0
dtype: int64
```

In [4]:

```
df['id'].value_counts()
```

Out[4]:

```
0      1
263    1
273    1
272    1
271    1
      ..
130    1
129    1
128    1
127    1
399    1
Name: id, Length: 400, dtype: int64
```

id column have all unique values so drop that column

In [5]:

```
del df["id"]
```

In [6]:

```
df['age'].value_counts()
```

Out[6]:

```
60.0    19
65.0    17
48.0    12
55.0    12
50.0    12
        ..
83.0     1
27.0     1
14.0     1
81.0     1
79.0     1
Name: age, Length: 76, dtype: int64
```

Age Column contains 2.25% null values so drop only null values at last after Cleanning the data

In [7]:

```
df["bp"].value_counts()
```

Out[7]:

```
80.0     116
70.0     112
60.0      71
90.0      53
100.0     25
50.0       5
110.0      3
140.0      1
180.0      1
120.0      1
Name: bp, dtype: int64
```

it contains 3% null values so replace it with mean

In [8]:

```
df['bp']=df['bp'].fillna(df['bp'].mean())
```

In [9]:

```
df['sg'].value_counts()
```

Out[9]:

```
1.020    106
1.010     84
1.025     81
1.015     75
1.005      7
Name: sg, dtype: int64
```

it contain 22 % of null values so replace it with mean

In [10]:

```
df['sg']=df['sg'].fillna(df['sg'].mean())
```

In [11]:

```
df['al'].value_counts()
```

Out[11]:

```
0.0    199
1.0     44
2.0     43
3.0     43
4.0     24
5.0      1
Name: al, dtype: int64
```

In [12]:

```
df['su'].value_counts()
```

Out[12]:

```
0.0    290
2.0     18
3.0     14
4.0     13
1.0     13
5.0      3
Name: su, dtype: int64
```

replace both the column(su , al) by mean

In [13]:

```
df['su']=df['su'].fillna(df['su'].mean())
df['al']=df['al'].fillna(df['al'].mean())
```

In [14]:

```
df['rbc'].value_counts()
```

Out[14]:

```
normal      201
abnormal     47
Name: rbc, dtype: int64
```

it contaions 38% null values and datatype is categorical so replace it with mode

In [15]:

```
df.rbc.replace(np.nan,'normal',inplace=True)
```

In [16]:

```
df['rbc'].value_counts()
```

Out[16]:

```
normal      353
abnormal     47
Name: rbc, dtype: int64
```

In [17]:

```
df['pc'].value_counts()
```

Out[17]:

```
normal      259
abnormal     76
Name: pc, dtype: int64
```

replace null values with mode

In [18]:

```
df.pc.replace(np.nan,'normal',inplace=True)
```

In [19]:

```
df['pc'].value_counts()
```

Out[19]:

```
normal      324
abnormal     76
Name: pc, dtype: int64
```

In [20]:

```
df['pcc'].value_counts()
```

Out[20]:

```
notpresent    354
present        42
Name: pcc, dtype: int64
```

In [21]:

```
df['ba'].value_counts()
```

Out[21]:

```
notpresent    374
present        22
Name: ba, dtype: int64
```

for column pcc and ba we have to drop rows after cleaning all the data

In [22]:

```
df['bgr'].value_counts()
```

Out[22]:

```
99.0     10
93.0      9
100.0     9
107.0     8
131.0     6
         ..
288.0     1
182.0     1
84.0      1
256.0     1
226.0     1
Name: bgr, Length: 146, dtype: int64
```

replace null value by mean

In [23]:

```
df['bgr']=df['bgr'].fillna(df['bgr'].mean())
```

In [24]:

```
df['bu'].value_counts()
```

Out[24]:

```
46.0     15
25.0     13
19.0     11
40.0     10
50.0      9
         ..
176.0     1
145.0     1
92.0      1
322.0     1
186.0     1
Name: bu, Length: 118, dtype: int64
```

In [25]:

```
df['sc'].value_counts()
```

Out[25]:

```
1.2     40
1.1     24
0.5     23
1.0     23
0.9     22
        ..
3.8      1
12.2     1
9.2      1
13.8     1
0.4      1
Name: sc, Length: 84, dtype: int64
```

REPLACE Both Columns('bu' , 'sc') Null values by mean

In [26]:

```
df['bu']=df['bu'].fillna(df['bu'].mean())
df['sc']=df['sc'].fillna(df['sc'].mean())
```

In [27]:

```
df['sod'].value_counts()
```

Out[27]:

```
135.0    40
140.0    25
141.0    22
139.0    21
138.0    20
142.0    20
137.0    19
150.0    17
136.0    17
147.0    13
145.0    11
132.0    10
146.0    10
131.0     9
144.0     9
133.0     8
130.0     7
134.0     6
143.0     4
124.0     3
127.0     3
122.0     2
113.0     2
120.0     2
125.0     2
128.0     2
114.0     2
126.0     1
163.0     1
115.0     1
129.0     1
4.5       1
104.0     1
111.0     1
Name: sod, dtype: int64
```

In [28]:

```
df['pot'].value_counts()
```

Out[28]:

```
3.5     30
5.0     30
4.9     27
4.7     17
4.8     16
3.9     14
3.8     14
4.1     14
4.2     14
4.0     14
4.4     14
4.5     13
4.3     12
3.7     12
3.6      8
4.6      7
3.4      5
5.2      5
5.3      4
5.7      4
3.2      3
5.5      3
6.3      3
5.4      3
2.9      3
3.3      3
5.6      2
3.0      2
6.5      2
2.5      2
5.9      2
5.8      2
7.6      1
47.0     1
6.6      1
5.1      1
6.4      1
2.8      1
2.7      1
39.0     1
Name: pot, dtype: int64
```

replace both column by mean

In [29]:

```
df['sod']=df['sod'].fillna(df['sod'].mean())
df['pot']=df['pot'].fillna(df['pot'].mean())
```

In [30]:

```
df['hemo'].value_counts()
```

Out[30]:

```
15.0    16
10.9     8
13.6     7
13.0     7
9.8      7
        ..
6.8      1
8.5      1
7.3      1
12.8     1
17.6     1
Name: hemo, Length: 115, dtype: int64
```

replace null values from 'hemo' with mean

In [31]:

```
df['hemo']=df['hemo'].fillna(df['hemo'].mean())
```

In [32]:

```
df['pcv'].value_counts()
```

Out[32]:

```
41      21
52      21
44      19
48      19
40      16
43      14
42      13
45      13
32      12
36      12
33      12
50      12
28      12
34      11
37      11
30       9
29       9
35       9
46       9
31       8
24       7
39       7
26       6
38       5
53       4
51       4
49       4
47       4
54       4
25       3
27       3
22       3
19       2
23       2
15       1
21       1
17       1
20       1
\t43     1
18       1
9        1
14       1
\t?      1
16       1
Name: pcv, dtype: int64
```

Replace messy or noisy values from 'pcv' column by np.nan

In [33]:

```
df.pcv.replace('\t43',np.nan,inplace=True)
df.pcv.replace('\t?',np.nan,inplace=True)
```

In [34]:

```
df.pcv.isnull().sum()
```

Out[34]:

```
72
```

Replace null values from 'pcv' i.e. 18% null values by mean

1st convert all string values to numeric using to_numeric() function

In [35]:

```
df['pcv']=pd.to_numeric(df['pcv'])
```

In [36]:

```
df['pcv']=df['pcv'].fillna(df['pcv'].mean())
```

In [37]:

```
df['wc'].value_counts()
```

Out[37]:

```
9800     11
6700     10
9200      9
9600      9
7200      9
         ..
19100     1
\t?       1
12300     1
14900     1
12700     1
Name: wc, Length: 92, dtype: int64
```

In [38]:

```
df['rc'].value_counts()
```

Out[38]:

```
5.2    18
4.5    16
4.9    14
4.7    11
4.8    10
3.9    10
4.6     9
3.4     9
5.9     8
5.5     8
6.1     8
5.0     8
3.7     8
5.3     7
5.8     7
5.4     7
3.8     7
5.6     6
4.3     6
4.2     6
3.2     5
4.4     5
5.7     5
6.4     5
5.1     5
6.2     5
6.5     5
4.1     5
3.6     4
6.3     4
6.0     4
4.0     3
3.3     3
4       3
3.5     3
2.9     2
3.1     2
2.6     2
2.1     2
2.5     2
2.8     2
3.0     2
2.7     2
5       2
2.3     1
\t?     1
2.4     1
3       1
8.0     1
Name: rc, dtype: int64
```

REPLACE Messy values from 'rc' and 'wc' table to np.nan

In [39]:

```
df.rc.replace('\t?',np.nan,inplace=True)
df.wc.replace('\t?',np.nan,inplace=True)
```

In [40]:

```
df['rc']=pd.to_numeric(df['rc'])
df['wc']=pd.to_numeric(df['wc'])
```

Replace null values from both the table with mean

In [41]:

```
df['rc']=df['rc'].fillna(df['rc'].mean())
df['wc']=df['wc'].fillna(df['wc'].mean())
```

In [42]:

```
df['htn'].value_counts()
```

Out[42]:

```
no     251
yes    147
Name: htn, dtype: int64
```

In [43]:

```
df['dm'].value_counts()
```

Out[43]:

```
no       258
yes      134
\tno       3
\tyes      2
 yes       1
Name: dm, dtype: int64
```

Replace the messy values from 'dm' table

In [44]:

```
df.dm.replace('\tno','no',inplace=True)
df.dm.replace('\tyes','yes',inplace=True)
df.dm.replace(' yes','yes',inplace=True)
```

In [45]:

```
df['dm'].value_counts()
```

Out[45]:

```
no     261
yes    137
Name: dm, dtype: int64
```

In [46]:

```
df['cad'].value_counts()
```

Out[46]:

```
no      362
yes      34
\tno      2
Name: cad, dtype: int64
```

Replace messy values from 'cad' column

In [47]:

```
df.cad.replace('\tno','no',inplace=True)
```

In [48]:

```
df['cad'].value_counts()
```

Out[48]:

```
no     364
yes     34
Name: cad, dtype: int64
```

In [49]:

```
df['appet'].value_counts()
```

Out[49]:

```
good    317
poor     82
Name: appet, dtype: int64
```

In [50]:

```
df['pe'].value_counts()
```

Out[50]:

```
no     323
yes     76
Name: pe, dtype: int64
```

In [51]:

```
df['ane'].value_counts()
```

Out[51]:

```
no     339
yes     60
Name: ane, dtype: int64
```

In [52]:

```
df["classification"].value_counts()
```

Out[52]:

```
ckd       248
notckd    150
ckd\t       2
Name: classification, dtype: int64
```

In this Column

ckd -Chronical KidneyDisease

notckd - Not Chronical kidney Disease

In [53]:

```
df.classification.replace('ckd\t','ckd',inplace=True)
```

In [54]:

```
df["classification"].value_counts()
```

Out[54]:

```
ckd       250
notckd    150
Name: classification, dtype: int64
```

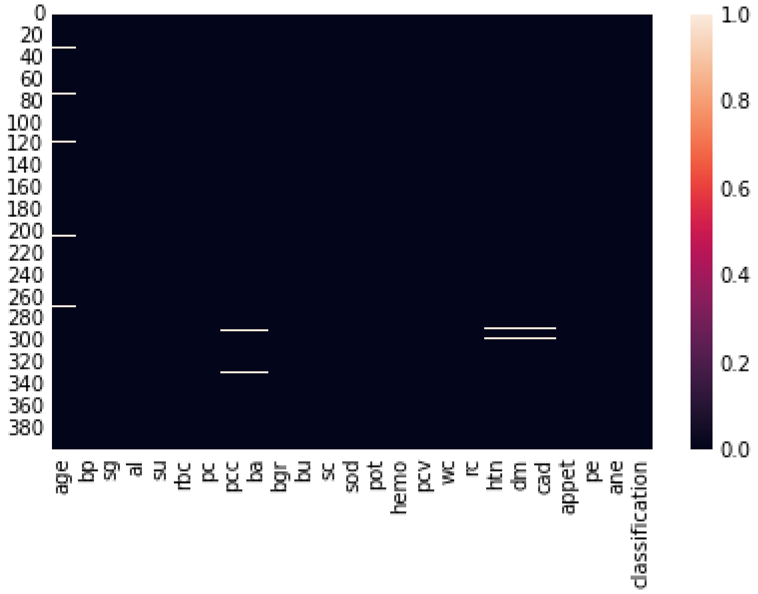
PLOT HEATMAP TO FIND OUT NULL VALUES OF DATAFRAME

In [55]:

```
sns.heatmap(df.isnull())
```

Out[55]:

```
<AxesSubplot:>
```



DROP ALL THE REMAINING NULL VALUES

In [56]:

```
df.dropna(inplace=True)
df.shape
```

Out[56]:

```
(384, 25)
```

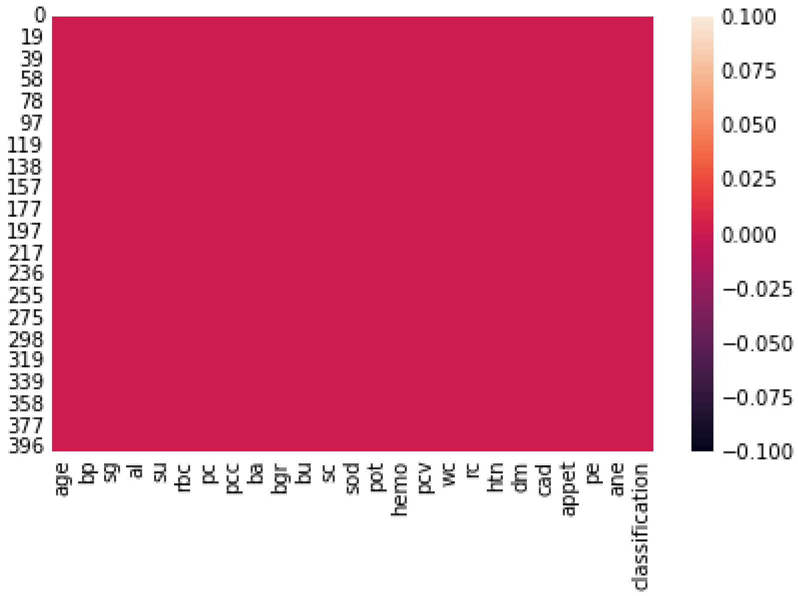
PLOT HEATMAP TO VERIFY OUR DATA IS CLEAN OR NOT

In [57]:

```
sns.heatmap(df.isnull())
```

Out[57]:

```
<AxesSubplot:>
```



In [95]:

```
newdf
```

Out[95]:

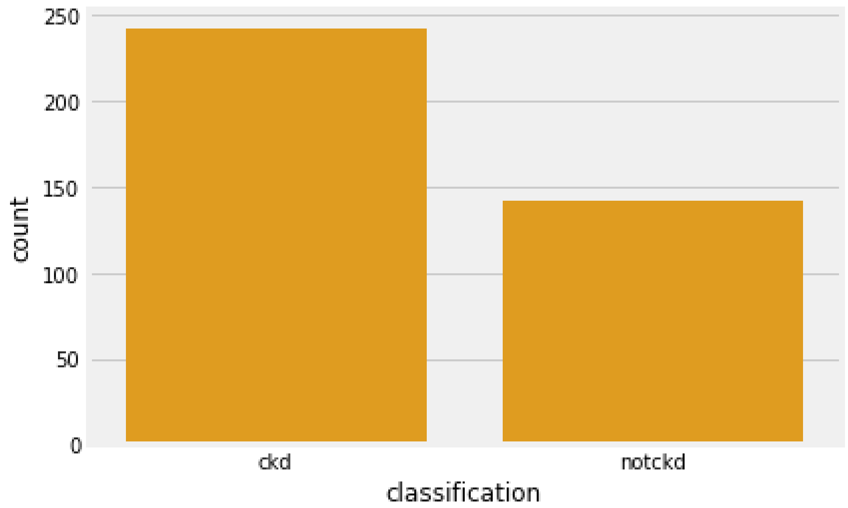
ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc
-0.237171	121.000000	-0.226698	-0.294972	-0.242488	4.627244	15.4	44.0	7800.0	5.200000
-0.237171	148.036517	-1.335735	-0.931268	-0.242488	4.627244	11.3	38.0	6000.0	4.707435
-0.237171	117.000000	0.488586	1.211516	-3.323825	2.500000	11.2	32.0	6700.0	3.900000
-0.237171	106.000000	-0.750113	-0.058653	-0.242488	4.627244	11.6	35.0	7300.0	4.600000
-0.237171	74.000000	-0.812914	-0.430601	0.588367	3.200000	12.2	39.0	7800.0	4.400000
...	...	...	...	...	...	...	...	...	...
-0.237171	140.000000	0.272018	-1.640046	2.376911	4.900000	15.7	47.0	6700.0	4.900000
-0.237171	75.000000	-0.467678	-0.294972	0.392741	3.500000	16.5	54.0	7800.0	6.200000
-0.237171	100.000000	-0.750113	-1.373266	-0.333500	4.400000	15.8	49.0	6600.0	5.400000
-0.237171	114.000000	0.304766	-0.580263	-0.664636	4.900000	14.2	51.0	7200.0	5.900000
-0.237171	131.000000	-1.335735	-0.430601	0.392741	3.500000	15.8	53.0	6800.0	6.100000

In [59]:

```
sns.countplot(x='classification',color="orange",data=df)
```

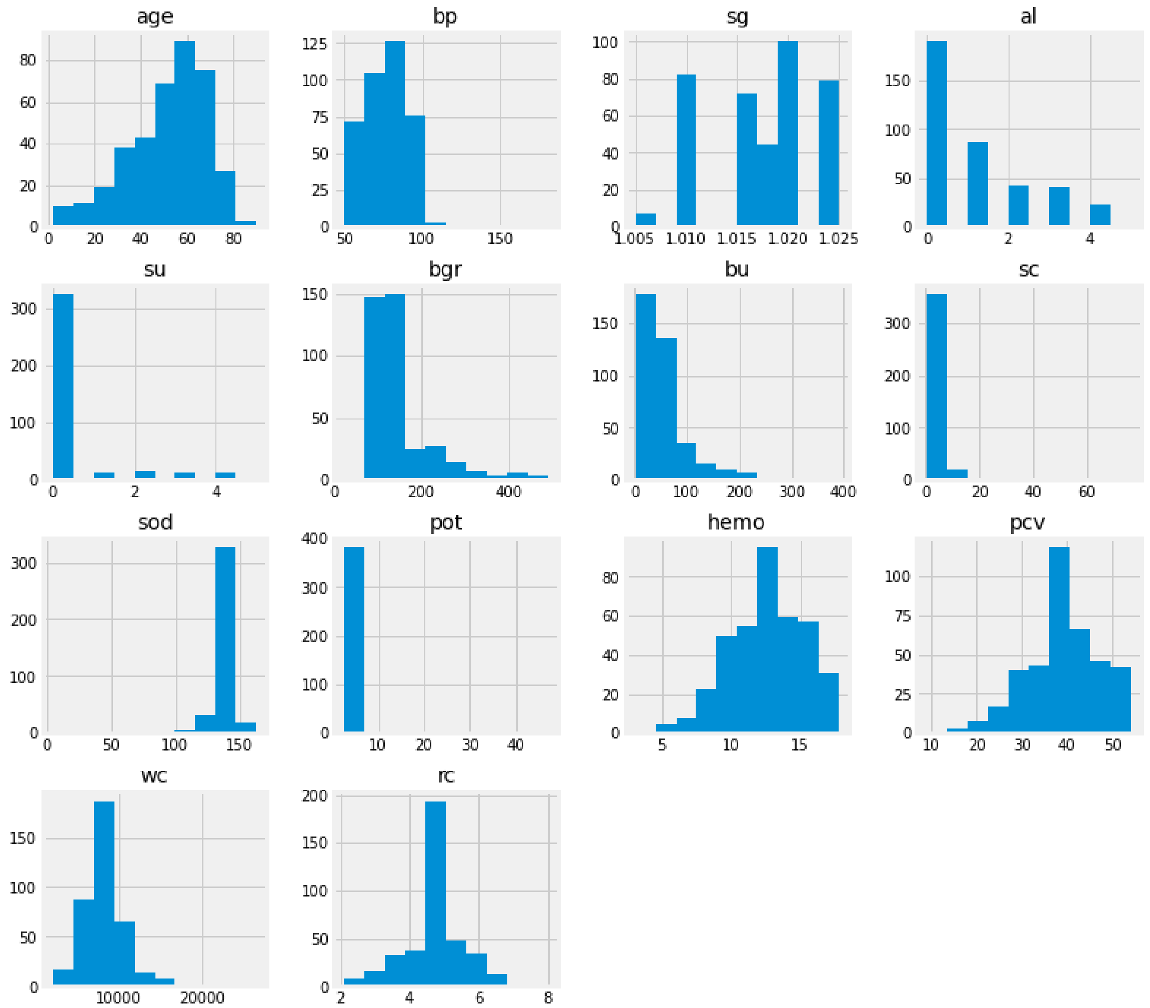
Out[59]:

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



In [60]:

```
df.hist(figsize=(12,12))
plt.show()
```

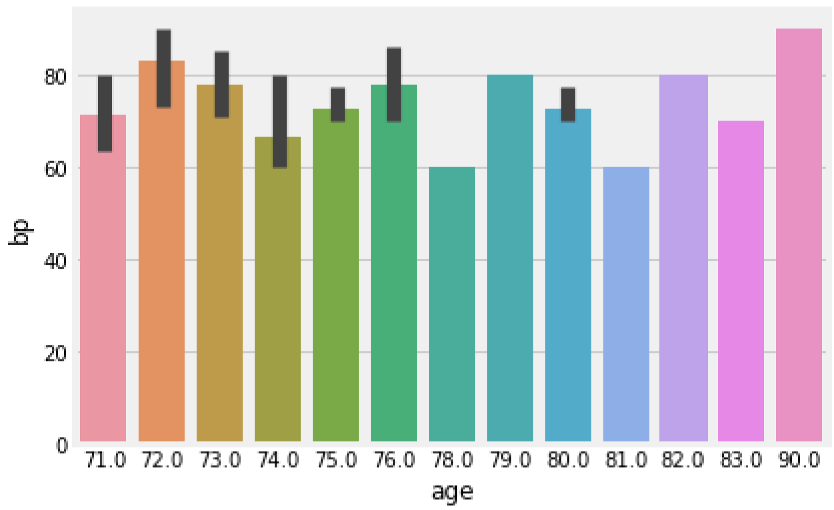


In [61]:

```
ss=df.loc[(df['age']>70)&(df['bp']<100)]
sns.barplot(x='age',y='bp',data=ss)
```

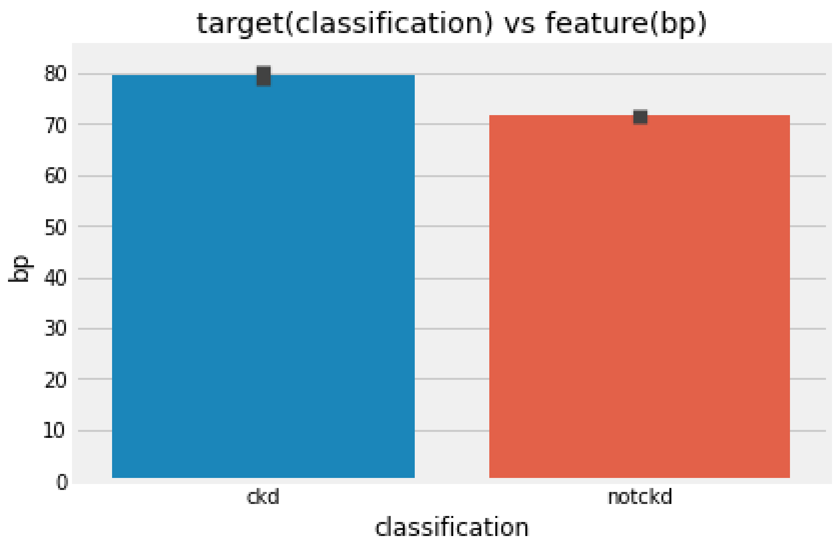
Out[61]:

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



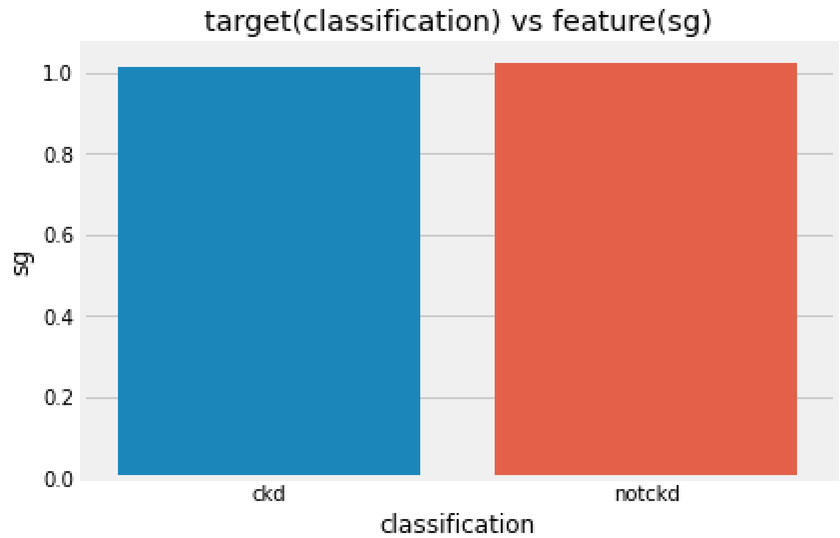
In [103]:

```
#bivariate analysis
#1 vs 1
sns.barplot(x='classification',y='bp',data=df)
plt.title('target(classification) vs feature(bp)')
plt.show()
```



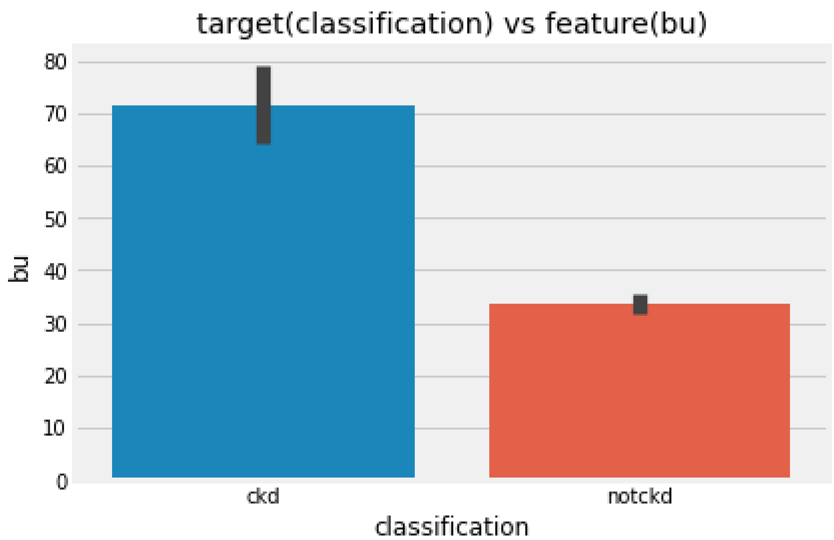
In [102]:

```
#bivariate analysis
#1 vs 1
sns.barplot(x='classification',y='sg',data=df)
plt.title('target(classification) vs feature(sg)')
plt.show()
```



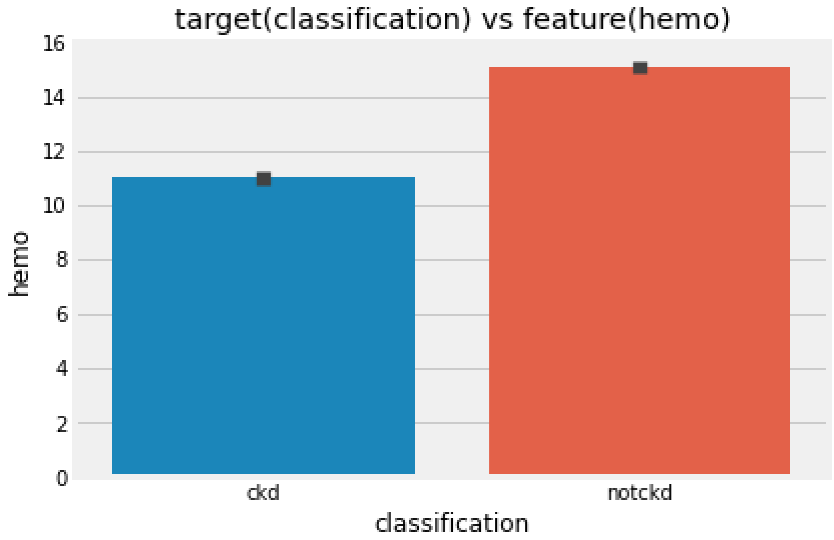
In [101]:

```
#bivariate analysis
#1 vs 1
sns.barplot(x='classification',y='bu',data=df)
plt.title('target(classification) vs feature(bu)')
plt.show()
```



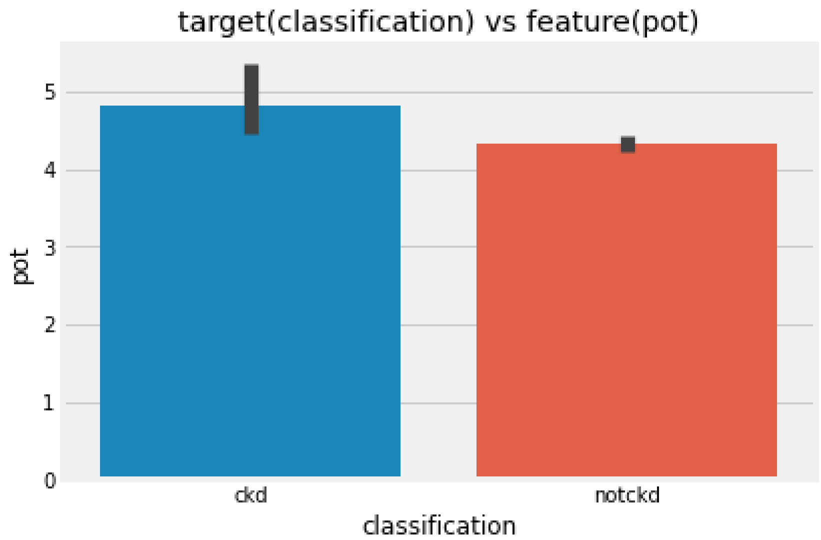
In [104]:

```
#bivariate analysis
#1 vs 1
sns.barplot(x='classification',y='hemo',data=df)
plt.title('target(classification) vs feature(hemo)')
plt.show()
```



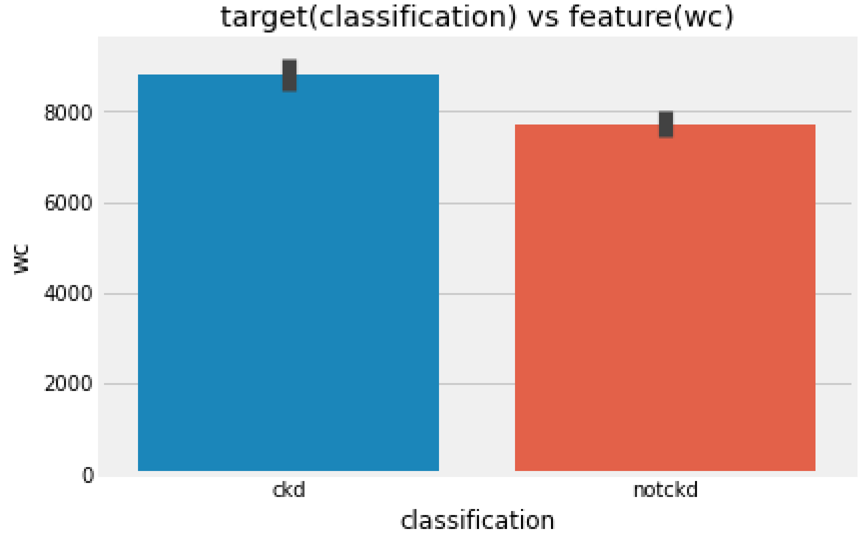
In [105]:

```
#bivariate analysis
#1 vs 1
sns.barplot(x='classification',y='pot',data=df)
plt.title('target(classification) vs feature(pot)')
plt.show()
```



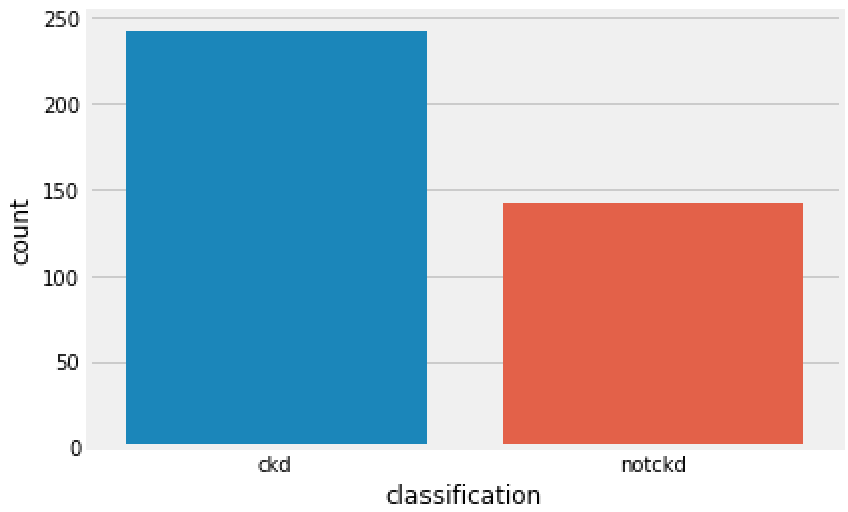
In [106]:

```
#bivariate analysis
#1 vs 1
sns.barplot(x='classification',y='wc',data=df)
plt.title('target(classification) vs feature(wc)')
plt.show()
```



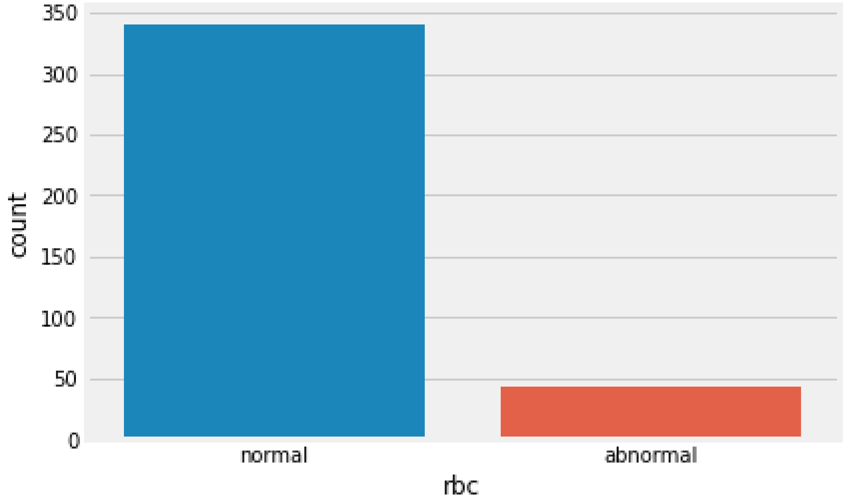
In [128]:

```
#univariate analysis
#self
sns.countplot(x='classification',data=df)
plt.show()
```



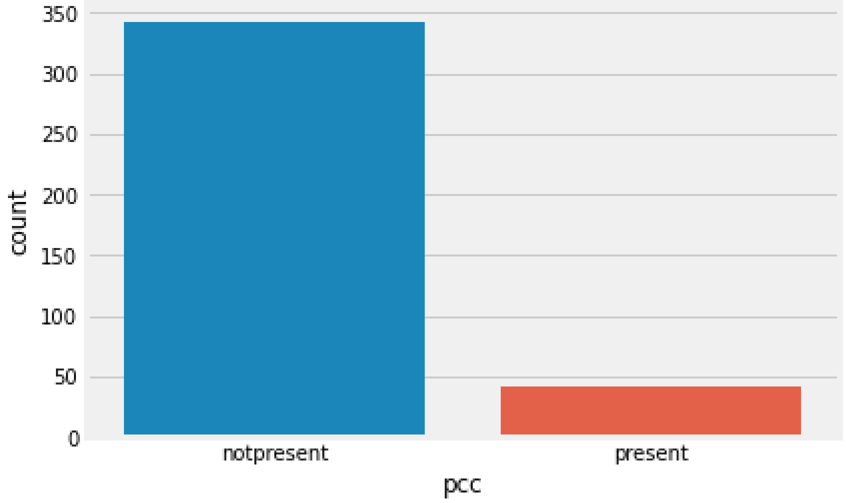
In [108]:

```
#univariate analysis
#self
sns.countplot(x='rbc',data=df)
plt.show()
```



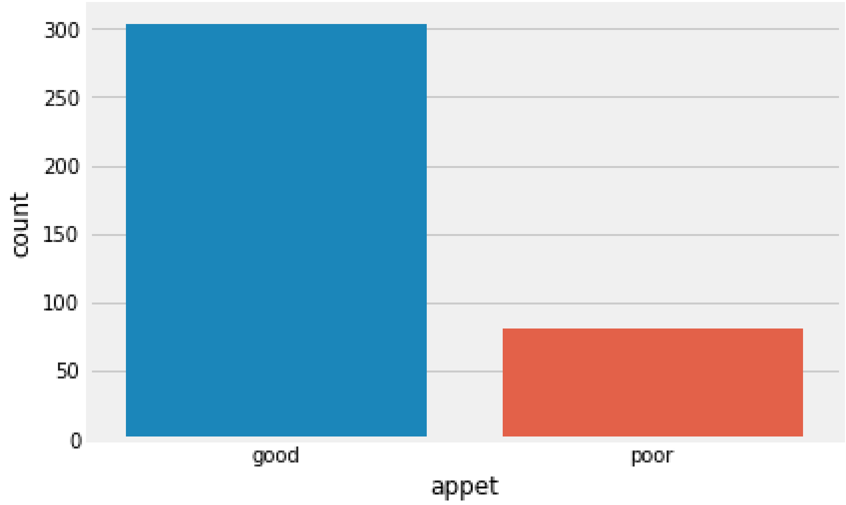
In [111]:

```
#univariate analysis
#self
sns.countplot(x='pcc',data=df)
plt.show()
```



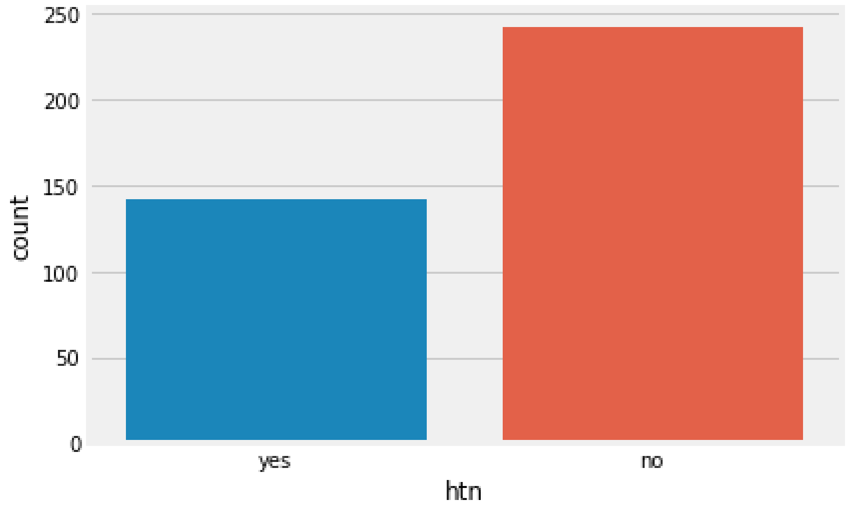
In [112]:

```
#univariate analysis
#self
sns.countplot(x='appet',data=df)
plt.show()
```



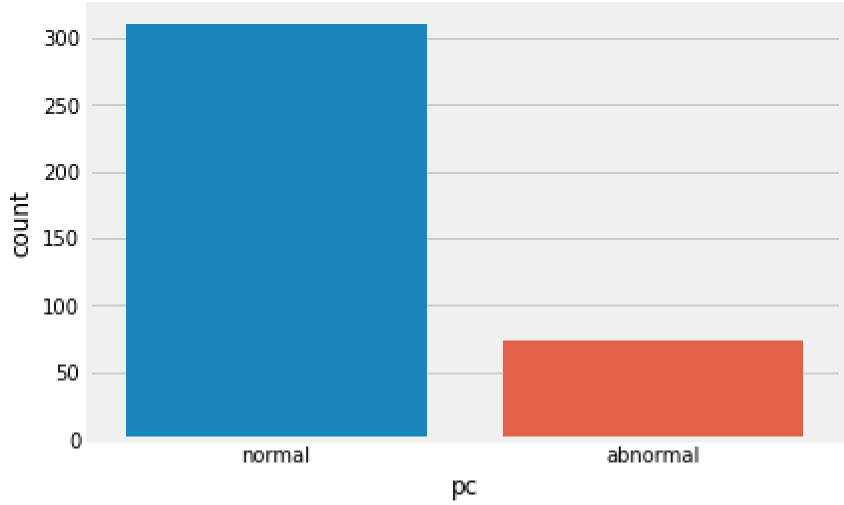
In [114]:

```
#univariate analysis
#self
sns.countplot(x='htn',data=df)
plt.show()
```



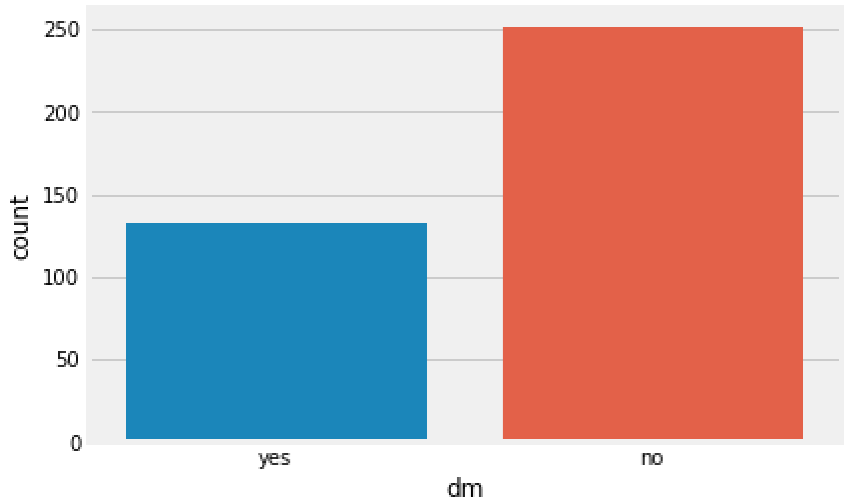
In [115]:

```
#univariate analysis
#self
sns.countplot(x='pc',data=df)
plt.show()
```



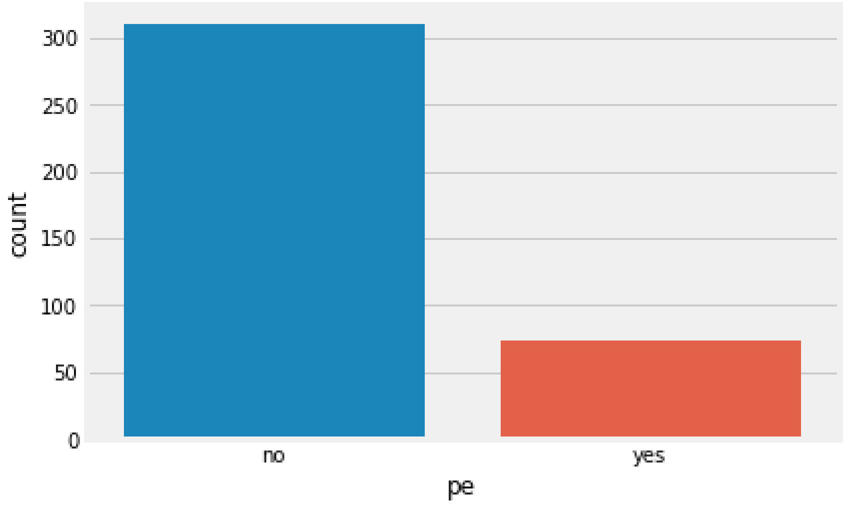
In [117]:

```
#univariate analysis
#self
sns.countplot(x='dm',data=df)
plt.show()
```



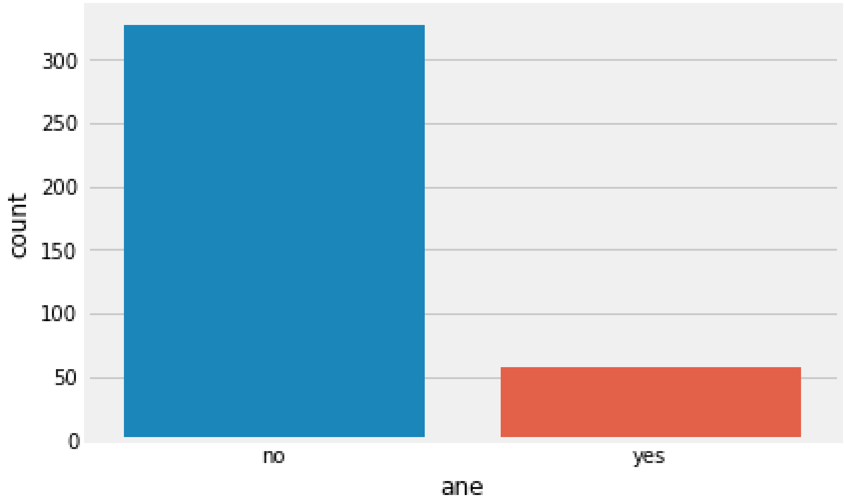
In [118]:

```
#univariate analysis
#self
sns.countplot(x='pe',data=df)
plt.show()
```



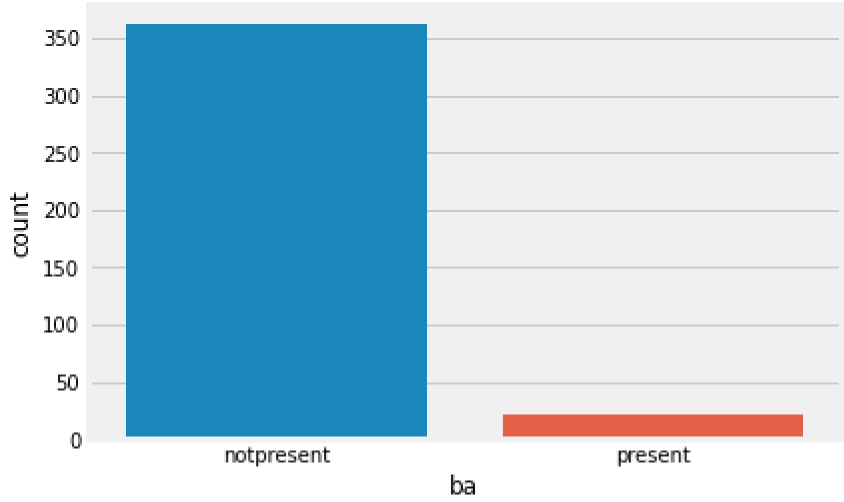
In [119]:

```
#univariate analysis
#self
sns.countplot(x='ane',data=df)
plt.show()
```



In [120]:

```
#univariate analysis
#self
sns.countplot(x='ba',data=df)
plt.show()
```

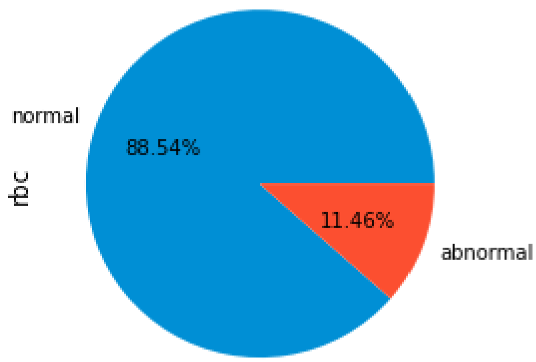


In [121]:

```
df['rbc'].value_counts().plot(kind='pie',autopct='%1.2f%%')
```

Out[121]:

```
<AxesSubplot:ylabel='rbc'>
```

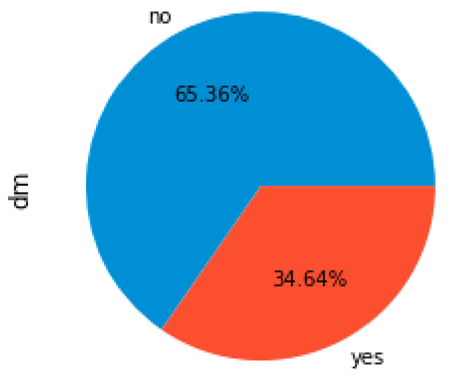


In [123]:

```
df['dm'].value_counts().plot(kind='pie',autopct='%1.2f%%')
```

Out[123]:

```
<AxesSubplot:ylabel='dm'>
```

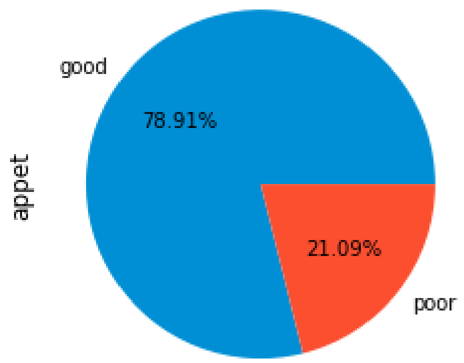


In [124]:

```
df['appet'].value_counts().plot(kind='pie',autopct='%1.2f%%')
```

Out[124]:

```
<AxesSubplot:ylabel='appet'>
```

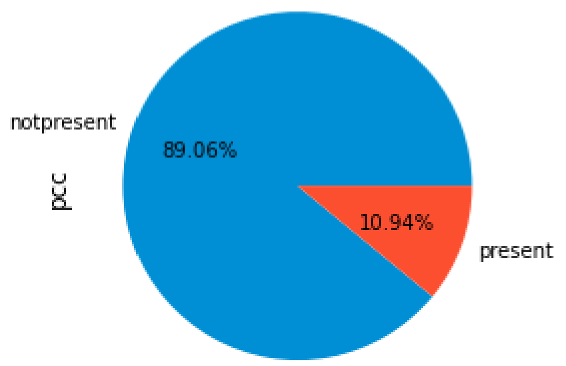


In [126]:

```
df['pcc'].value_counts().plot(kind='pie',autopct='%1.2f%%')
```

Out[126]:

```
<AxesSubplot:ylabel='pcc'>
```

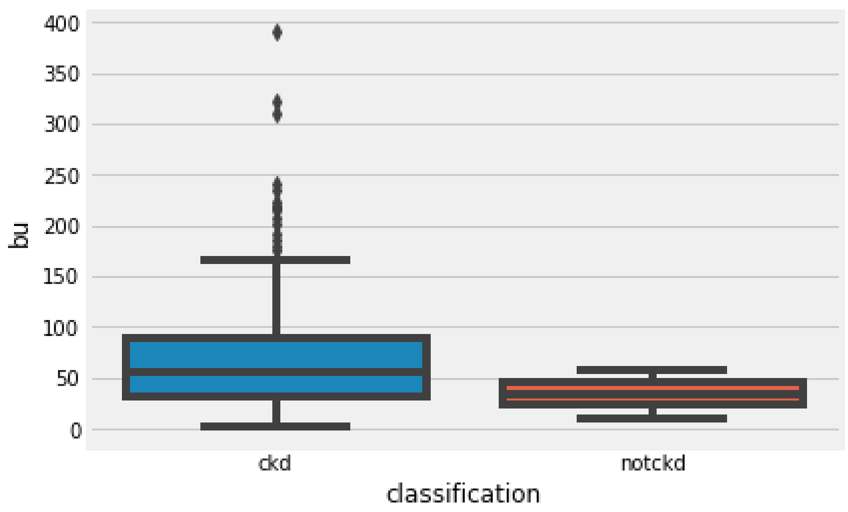


In [131]:

```
sns.boxplot(x='classification',y='bu',data=df)
```

Out[131]:

```
<AxesSubplot:xlabel='classification', ylabel='bu'>
```

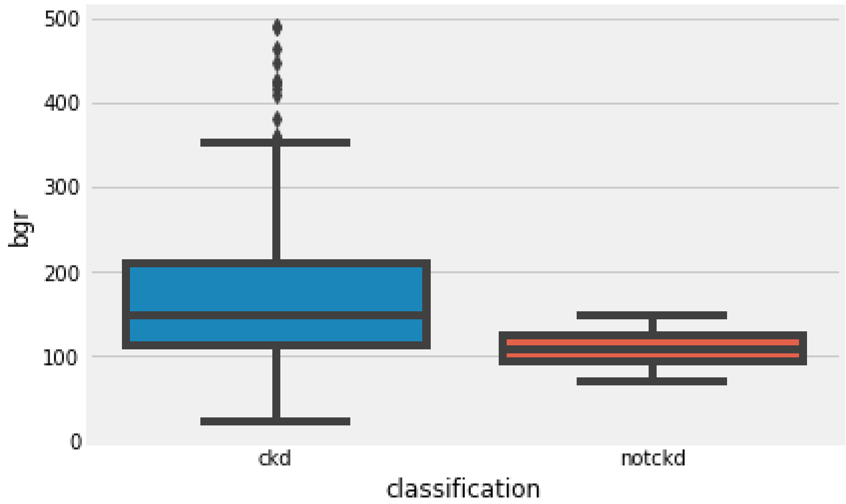


In [132]:

```
sns.boxplot(x='classification',y='bgr',data=df)
```

Out[132]:

<AxesSubplot:xlabel='classification', ylabel='bgr'>

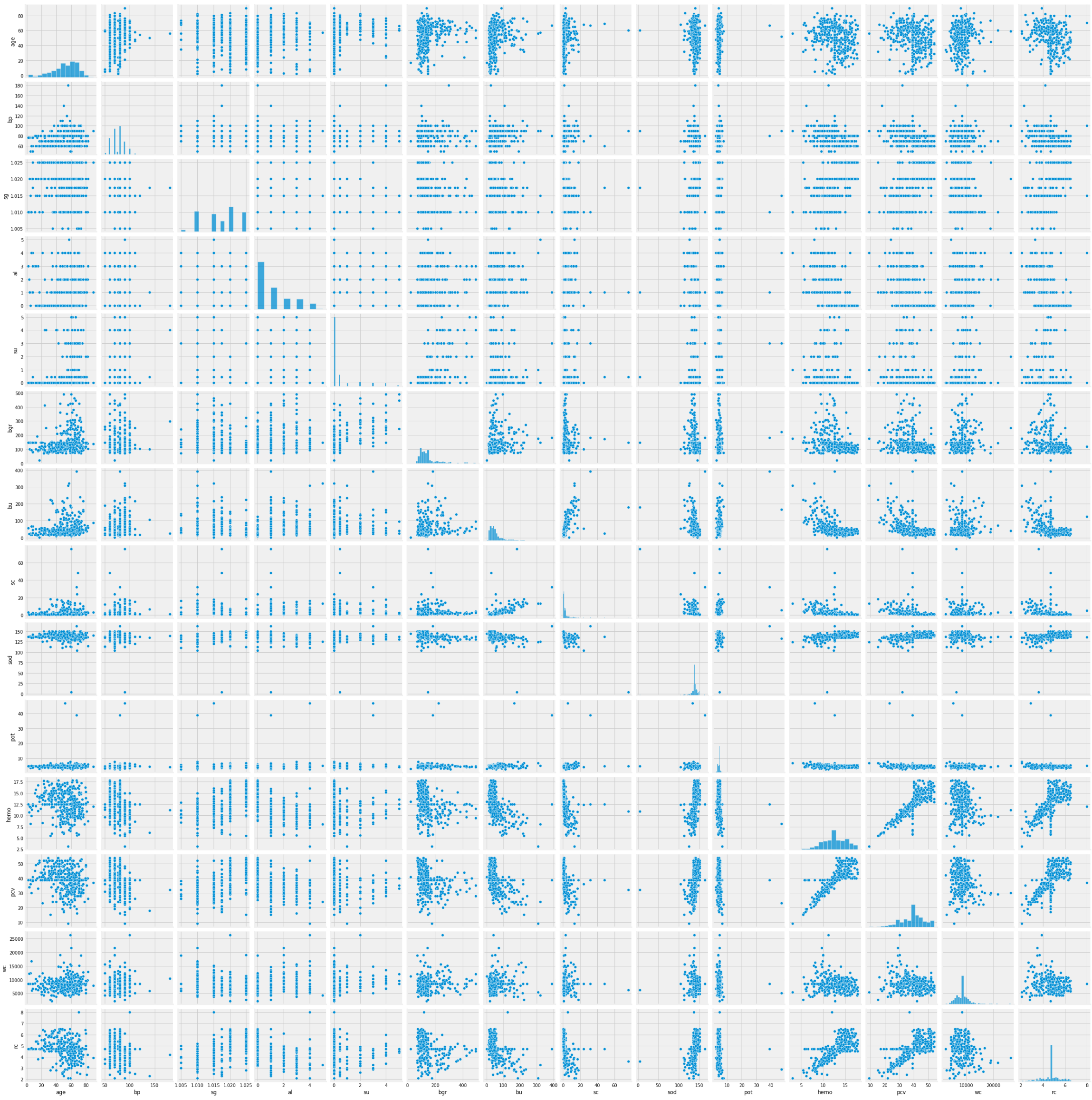


In [113]:

```
#multivariate analysis
#1 vs all
sns.pairplot(df)
```

Out[113]:

```
<seaborn.axisgrid.PairGrid at 0x196c8eba6a0>
```

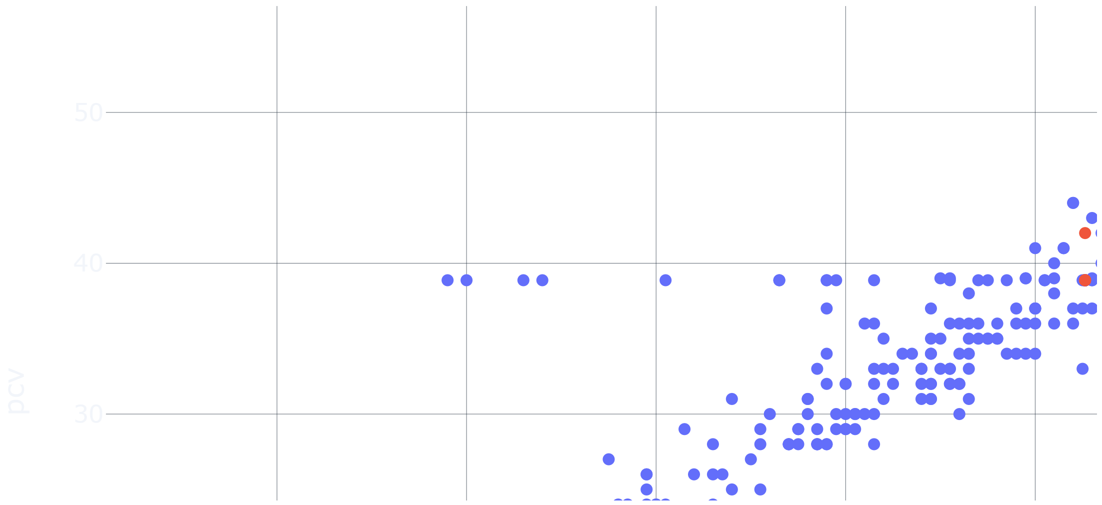


In [67]:

```
def scatter(col1, col2):
    fig = px.scatter(df, x=col1, y=col2, color="classification", template = 'plotly_dark'
    return fig.show()
```

In [68]:

```
scatter('hemo', 'pcv')
```



SPLIT DATA INTO Numerical column and Categorical Column

In [69]:

```
Numcol=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        Numcol.append(i)
Numcol
```

Out[69]:

```
['age',
 'bp',
 'sg',
 'al',
 'su',
 'bgr',
 'bu',
 'sc',
 'sod',
 'pot',
 'hemo',
 'pcv',
 'wc',
 'rc']
```

In [70]:

```
len(Numcol)
```

Out[70]:

```
14
```

In [71]:

```
catcol=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        catcol.append(i)
catcol
```

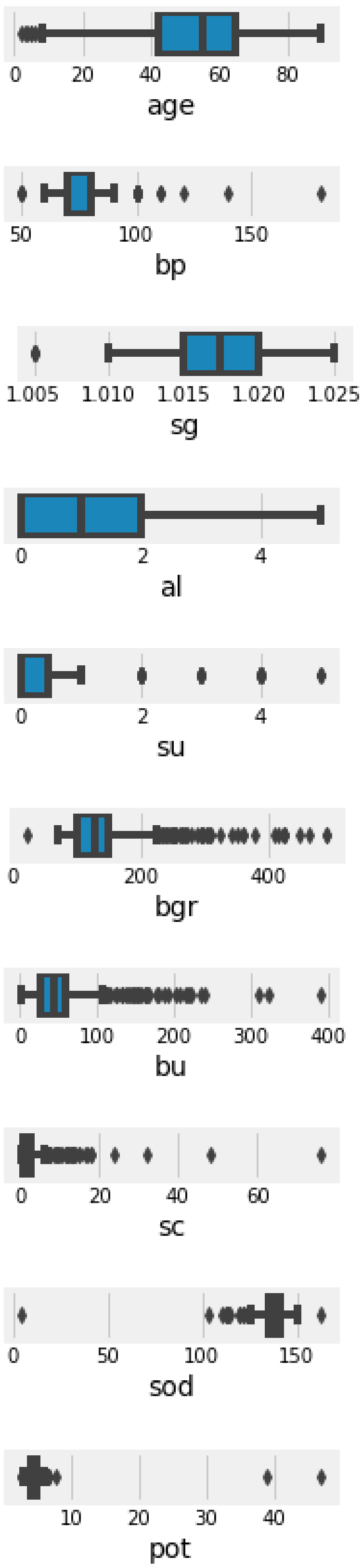
Out[71]:

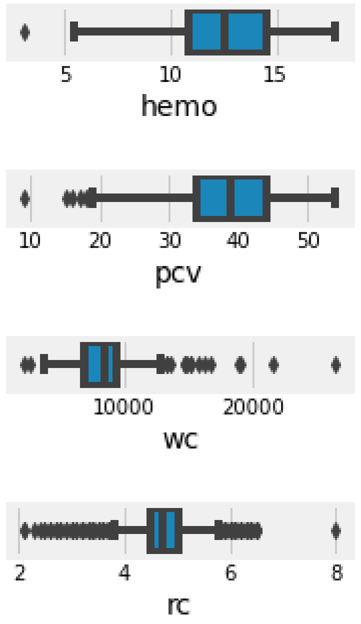
```
['rbc',
 'pc',
 'pcc',
 'ba',
 'htn',
 'dm',
 'cad',
 'appet',
 'pe',
 'ane',
 'classification']
```

PLOT BOXPLOT FOR NUMERIC VALUES TO FIND OUT OUTLIERS FROM COLUMN

In [72]:

```
plt.figure()
plotn=1
for i in Numcol:
    if plotn<=14:
        ax=plt.subplot(7,2,plotn)
        sns.boxplot(df[i])
        plt.xlabel(i,fontsize=14)
        plotn=plotn+1
        plt.show()
```





WE have outliers in every column except al REMOVE IT USING Z SCORE METHOD

In [73]:

```
f=df[['age','bp','sg','su','bgr','bu','sc','sod','pot','hemo','pcv','wc','rc']]
```

In [74]:

```
from scipy.stats import zscore
z=abs(zscore(f))
```

In [75]:

```
z
```

Out[75]:

	age	bp	sg	su	bgr	bu	sc	sod	po
0	0.204771	0.246173	0.483713	0.441058	0.369070	0.432943	0.333115	0.003940	0.00150(
1	2.589318	1.971393	0.483713	0.441058	0.011121	0.795614	0.403275	0.003940	0.00150(
2	0.609465	0.246173	1.366250	2.434326	3.629253	0.090421	0.227875	0.003940	0.00150(
3	0.204771	0.493016	2.291231	0.441058	0.422028	0.029975	0.122926	2.837614	0.74246(
4	0.030292	0.246173	1.366250	0.441058	0.567662	0.634427	0.298035	0.003940	0.00150(
...	...	...	...	...	...	...	...	...	..
395	0.202347	0.246173	0.483713	0.441058	0.117520	0.171014	0.455895	1.339763	0.09350(
396	0.553729	0.493016	1.408694	0.441058	0.978086	0.533685	0.333115	0.375753	0.39414
397	2.298519	0.246173	0.483713	0.441058	0.647099	0.634427	0.438355	0.052696	0.08065
398	2.007721	1.232204	1.408694	0.441058	0.461747	0.150866	0.368195	0.266920	0.09350(
399	0.376826	0.246173	1.408694	0.441058	0.236675	0.795614	0.350655	0.375753	0.39414

384 rows × 13 columns

In [76]:

```
newdf=df[(z<3).all(axis=1)]
newdf
```

Out[76]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc
0	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	121.000000	36.0	1.2
1	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	148.036517	18.0	0.8
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.000000	56.0	3.8
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.000000	26.0	1.4
5	60.0	90.0	1.015	3.0	0.0	normal	normal	notpresent	notpresent	74.000000	25.0	1.1
...	...	...	...	...	...	...	...	...	...	...	...	..
395	55.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	140.000000	49.0	0.5
396	42.0	70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	75.000000	31.0	1.2
397	12.0	80.0	1.020	0.0	0.0	normal	normal	notpresent	notpresent	100.000000	26.0	0.6
398	17.0	60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	114.000000	50.0	1.0
399	58.0	80.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	131.000000	18.0	1.1

338 rows × 25 columns

OUR Target is in the form of Categorical

SO 1st convert all categorical data into numeric

Because it is eassy to find out correlation Between Other Features and Target

In [77]:

```
from sklearn.preprocessing import OrdinalEncoder
oe=OrdinalEncoder()
newdf[catcol]=oe.fit_transform(newdf[catcol])
```

In [78]:

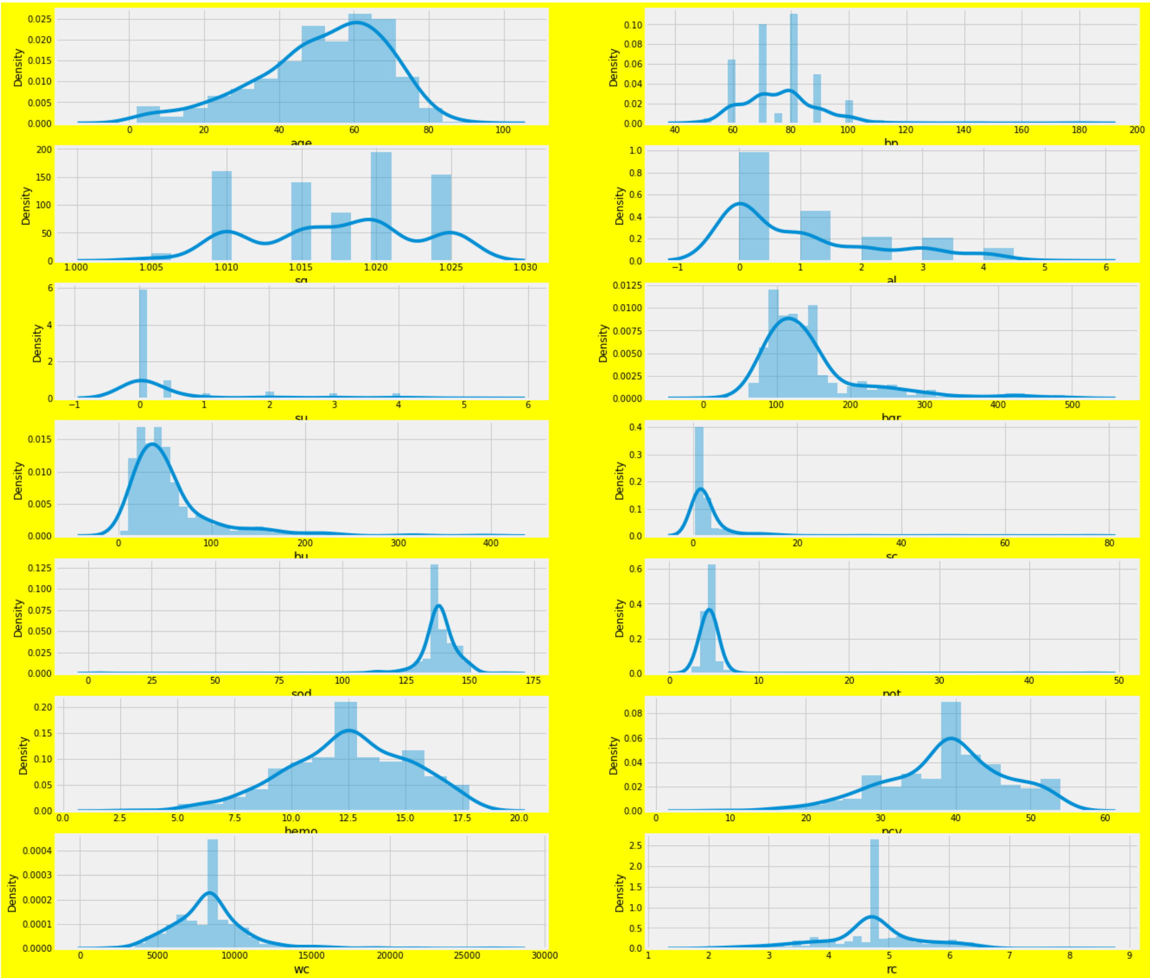
```
newdf.head()
```

Out[78]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot
0	48.0	80.0	1.020	1.0	0.0	1.0	1.0	0.0	0.0	121.000000	36.0	1.2	137.528754	4.627244
1	7.0	50.0	1.020	4.0	0.0	1.0	1.0	0.0	0.0	148.036517	18.0	0.8	137.528754	4.627244
3	48.0	70.0	1.005	4.0	0.0	1.0	0.0	1.0	0.0	117.000000	56.0	3.8	111.000000	2.500000
4	51.0	80.0	1.010	2.0	0.0	1.0	1.0	0.0	0.0	106.000000	26.0	1.4	137.528754	4.627244
5	60.0	90.0	1.015	3.0	0.0	1.0	1.0	0.0	0.0	74.000000	25.0	1.1	142.000000	3.200000

In [129]:

```
plt.figure(figsize=(20,20),facecolor='yellow')
plotn=1
for i in Numcol:
    if plotn<=14:
        ax=plt.subplot(7,2,plotn)
        sns.distplot(df[i])
        plt.xlabel(i,fontsize=14)
        plotn=plotn+1
```



Find out Skewness

In [80]:

```
newdf.skew()
```

Out[80]:

```
age              -0.610550
bp                0.278195
sg               -0.300419
al                1.217996
su                3.040360
rbc              -2.667605
pc               -1.927370
pcc               2.971076
ba                3.996959
bgr               1.748987
bu                1.772027
sc                2.980150
sod              -1.051448
pot              -0.009814
hemo             -0.306164
pcv              -0.398080
wc                0.452920
rc               -0.209005
htn               0.748432
dm                0.882954
cad               3.113080
appet             1.617831
pe                1.749822
ane               2.330257
classification    0.325129
dtype: float64
```

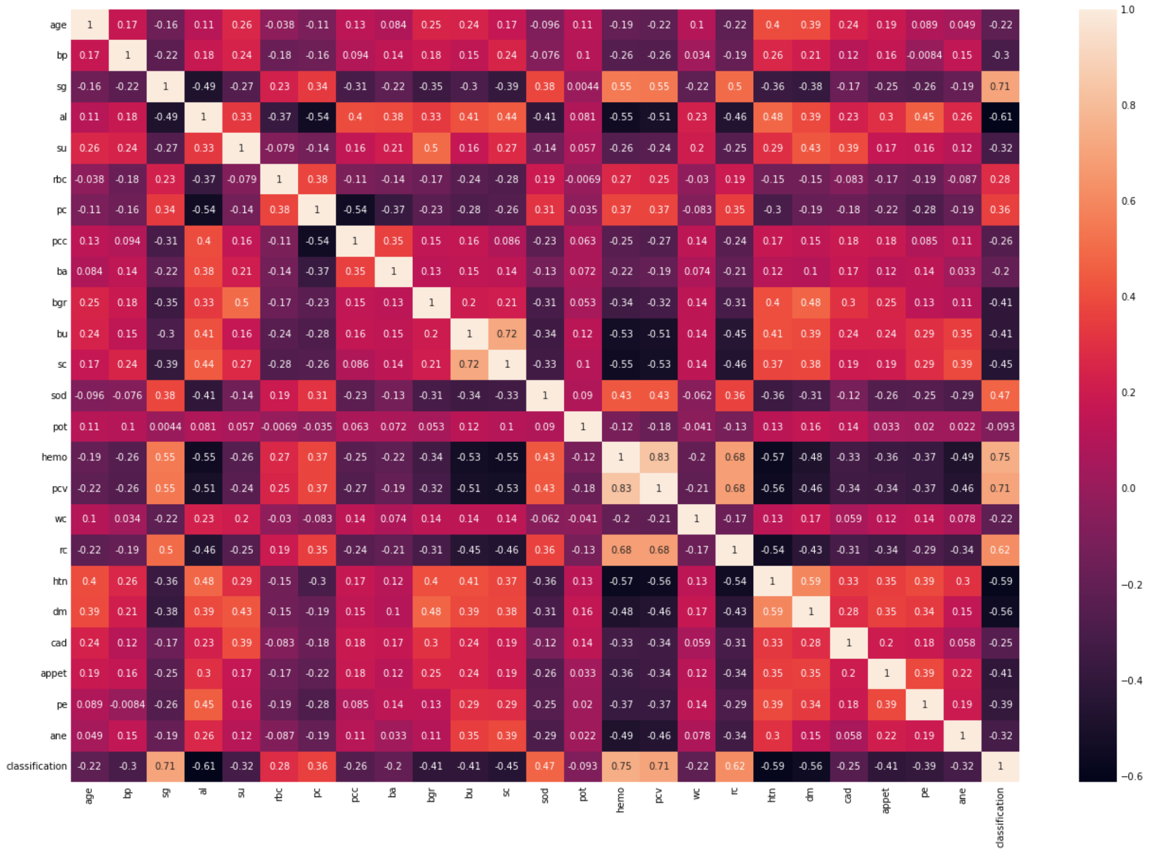
FIND OUT CORRELATION OF Features and Target using Heatmap

In [81]:

```
plt.figure(figsize=(20,15))
sns.heatmap(newdf.corr(),annot=True)
```

Out[81]:

<AxesSubplot:>



from dataset we conclude that we have skewness in Column Age, al,su,rbc,pc,pcc, ba,bu,sc,sod,htn,dm,cad,appet,pe,ane But htn,dm,al are stringly correlated with our target so dont remove skewness from them

In [82]:

```
s=['age','su','rbc','pc','pcc','ba','bu','sc','sod','cad','appet','pe','ane']
from sklearn.preprocessing import PowerTransformer
scaler=PowerTransformer(method='yeo-johnson')
newdf[s]=scaler.fit_transform(newdf[s].values)
```

In [83]:

```
newdf.skew()
```

Out[83]:

```
age              -0.222100
bp                0.278195
sg               -0.300419
al                1.217996
su                1.364572
rbc              -2.667605
pc               -1.927370
pcc               2.971076
ba                3.996959
bgr               1.748987
bu               -0.001674
sc                0.203925
sod               0.116439
pot              -0.009814
hemo             -0.306164
pcv              -0.398080
wc                0.452920
rc               -0.209005
htn               0.748432
dm                0.882954
cad               3.113080
appet             1.617831
pe                1.749822
ane               2.330257
classification    0.325129
dtype: float64
```

Split Data into Feature and Target

Store Features in- x Store Target in - y

In [84]:

```
x=newdf.drop("classification",axis=1)
x
```

Out[84]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	
0	-0.254405	80.0	1.020	1.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	121.000
1	-2.144444	50.0	1.020	4.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	148.036
3	-0.254405	70.0	1.005	4.0	-0.532545	0.334428	-2.345208	3.264226	-0.237171	117.000
4	-0.074075	80.0	1.010	2.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	106.000
5	0.493020	90.0	1.015	3.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	74.000
...	...	...	...	...	...	...	...	...	...	
395	0.173252	80.0	1.020	0.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	140.000
396	-0.601096	70.0	1.025	0.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	75.000
397	-1.989836	80.0	1.020	0.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	100.000
398	-1.807395	60.0	1.025	0.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	114.000
399	0.363732	80.0	1.025	0.0	-0.532545	0.334428	0.426401	-0.306351	-0.237171	131.000

338 rows × 24 columns

In [85]:

```
y=newdf["classification"]
y
```

Out[85]:

```
0      0.0
1      0.0
3      0.0
4      0.0
5      0.0
      ...
395    1.0
396    1.0
397    1.0
398    1.0
399    1.0
Name: classification, Length: 338, dtype: float64
```

Split Data For Training AND Testing

In [86]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=0)
```

In [87]:

```
from sklearn.metrics import classification_report, accuracy_score,confusion_matrix
```

Apply NeighborsClassifier and find out accuracy

In [88]:

```
#step 1: import the model
from sklearn.neighbors import KNeighborsClassifier
#step 2:create the object of algorithm
knn=KNeighborsClassifier(n_neighbors=5)
#step 3: train the model
knn.fit(xtrain,ytrain)
#step 4: predict
ypred=knn.predict(xtest)
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
ac=accuracy_score(ytest,ypred)
print(f"Accuracy score:{ac}\n{cm}\n{cr}")
train=knn.score(xtrain,ytrain)
test=knn.score(xtest,ytest)
print(f"Training Score:{train}\n Testing Score:{test}")
```

```
Accuracy score:0.7254901960784313
[[45 17]
 [11 29]]
              precision    recall  f1-score   support

         0.0       0.80      0.73      0.76        62
         1.0       0.63      0.72      0.67        40

    accuracy                           0.73       102
   macro avg       0.72      0.73      0.72       102
weighted avg       0.74      0.73      0.73       102

Training Score:0.8135593220338984
 Testing Score:0.7254901960784313
```

Apply DecisionTreeClassifier and find out accuracy

In [89]:

```
#step 1: import the model
from sklearn.tree import DecisionTreeClassifier
#step 2:create the object of algorithm
dtc1=DecisionTreeClassifier(max_depth=18)
#step 3: train the model
dtc1.fit(xtrain,ytrain)
#step 4: predict
ypred=dtc1.predict(xtest)
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
train=dtc1.score(xtrain,ytrain)
test=dtc1.score(xtest,ytest)
print(f"{cm}\n{cr}\nTraining Score:{train}\n Testing Score:{test}")
```

```
[[57  5]
 [ 0 40]]
              precision    recall  f1-score   support

         0.0       1.00      0.92      0.96        62
         1.0       0.89      1.00      0.94        40

    accuracy                           0.95       102
   macro avg       0.94      0.96      0.95       102
weighted avg       0.96      0.95      0.95       102

Training Score:1.0
 Testing Score:0.9509803921568627
```

Apply Naive Bayes all Algorithm to find Accuracy

Bernoulli naive Bayes

In [90]:

```
#step 1: import the model
from sklearn.naive_bayes import BernoulliNB
#step 2:create the object of algorithm
bnb=BernoulliNB()
#step 3: train the model
bnb.fit(xtrain,ytrain)
#step 4: predict
ypred=bnb.predict(xtest)
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
ac=accuracy_score(ytest,ypred)
print(f"Accuracy score:{ac}\n{cm}\n{cr}")
train=bnb.score(xtrain,ytrain)
test=bnb.score(xtest,ytest)
print(f"Training Score:{train}\n Testing Score:{test}")
```

```
Accuracy score:0.9117647058823529
[[54  8]
 [ 1 39]]
              precision    recall  f1-score   support

         0.0       0.98      0.87      0.92        62
         1.0       0.83      0.97      0.90        40

    accuracy                           0.91       102
   macro avg       0.91      0.92      0.91       102
weighted avg       0.92      0.91      0.91       102

Training Score:0.9279661016949152
 Testing Score:0.9117647058823529
```

Gaussian Naive Bayes

In [91]:

```
#step 1: import the model
from sklearn.naive_bayes import GaussianNB
#step 2:create the object of algorithm
gnb=GaussianNB()
#step 3: train the model
gnb.fit(xtrain,ytrain)
#step 4: predict
ypred=gnb.predict(xtest)
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
ac=accuracy_score(ytest,ypred)
print(f"Accuracy score:{ac}\n{cm}\n{cr}")
train=gnb.score(xtrain,ytrain)
test=gnb.score(xtest,ytest)
print(f"Training Score:{train}\n Testing Score:{test}")
```

```
Accuracy score:0.9509803921568627
[[58  4]
 [ 1 39]]
              precision    recall  f1-score   support

         0.0       0.98      0.94      0.96        62
         1.0       0.91      0.97      0.94        40

    accuracy                           0.95       102
   macro avg       0.95      0.96      0.95       102
weighted avg       0.95      0.95      0.95       102

Training Score:0.9491525423728814
 Testing Score:0.9509803921568627
```

Apply Boosting like Gradient boosting classifer , XGboosting classifier and Adaboost classifier

In [92]:

```
#step 1: import the model
from sklearn.ensemble import GradientBoostingClassifier
#step 2:create the object of algorithm
gbc=(GradientBoostingClassifier(n_estimators=2))
#step 3: train the model
gbc.fit(xtrain,ytrain)
#step 4: predict
ypred=gbc.predict(xtest)
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
ac=accuracy_score(ytest,ypred)
print(f"Accuracy score:{ac}\n{cm}\n{cr}")
train=gbc.score(xtrain,ytrain)
test=gbc.score(xtest,ytest)
print(f"Training Score:{train}\n Testing Score:{test}")
```

```
Accuracy score:0.9901960784313726
[[62  0]
 [ 1 39]]
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99        62
         1.0       1.00      0.97      0.99        40

    accuracy                           0.99       102
   macro avg       0.99      0.99      0.99       102
weighted avg       0.99      0.99      0.99       102

Training Score:0.9830508474576272
 Testing Score:0.9901960784313726
```

In [93]:

```
#step 1: import the model
from xgboost import XGBClassifier
#step 2:create the object of algorithm
xgb=(XGBClassifier(random_state=1,reg_alpha=1))
#step 3: train the model
xgb.fit(xtrain,ytrain)
#step 4: predict
ypred=xgb.predict(xtest)
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
ac=accuracy_score(ytest,ypred)
print(f"Accuracy score:{ac}\n{cm}\n{cr}")
train=xgb.score(xtrain,ytrain)
test=xgb.score(xtest,ytest)
print(f"Training Score:{train}\n Testing Score:{test}")
```

```
Accuracy score:1.0
[[62  0]
 [ 0 40]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        62
         1.0       1.00      1.00      1.00        40

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Training Score:1.0
 Testing Score:1.0
```

In [94]:

```
#step 1: import the model
from sklearn.ensemble import AdaBoostClassifier
#step 2:create the object of algorithm
adb=(AdaBoostClassifier(random_state=1))
#step 3: train the model
adb.fit(xtrain,ytrain)
#step 4: predict
ypred=adb.predict(xtest)
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)
ac=accuracy_score(ytest,ypred)
print(f"Accuracy score:{ac}\n{cm}\n{cr}")
train=adb.score(xtrain,ytrain)
test=adb.score(xtest,ytest)
print(f"Training Score:{train}\n Testing Score:{test}")
```

```
Accuracy score:1.0
[[62  0]
 [ 0 40]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        62
         1.0       1.00      1.00      1.00        40

    accuracy                           1.00       102
   macro avg       1.00      1.00      1.00       102
weighted avg       1.00      1.00      1.00       102

Training Score:1.0
 Testing Score:1.0
```