Assignment 2

Group Members

Abdul Wahab Madni, Hamza Badar, Aleksandr Semenikhin

In [44]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme()
```

Ex.1: Kernel Density Estimation

In [2]:
```
tdata = pd.read_table("values.txt", header=None, dtype=float, names=["Val
tdata
```

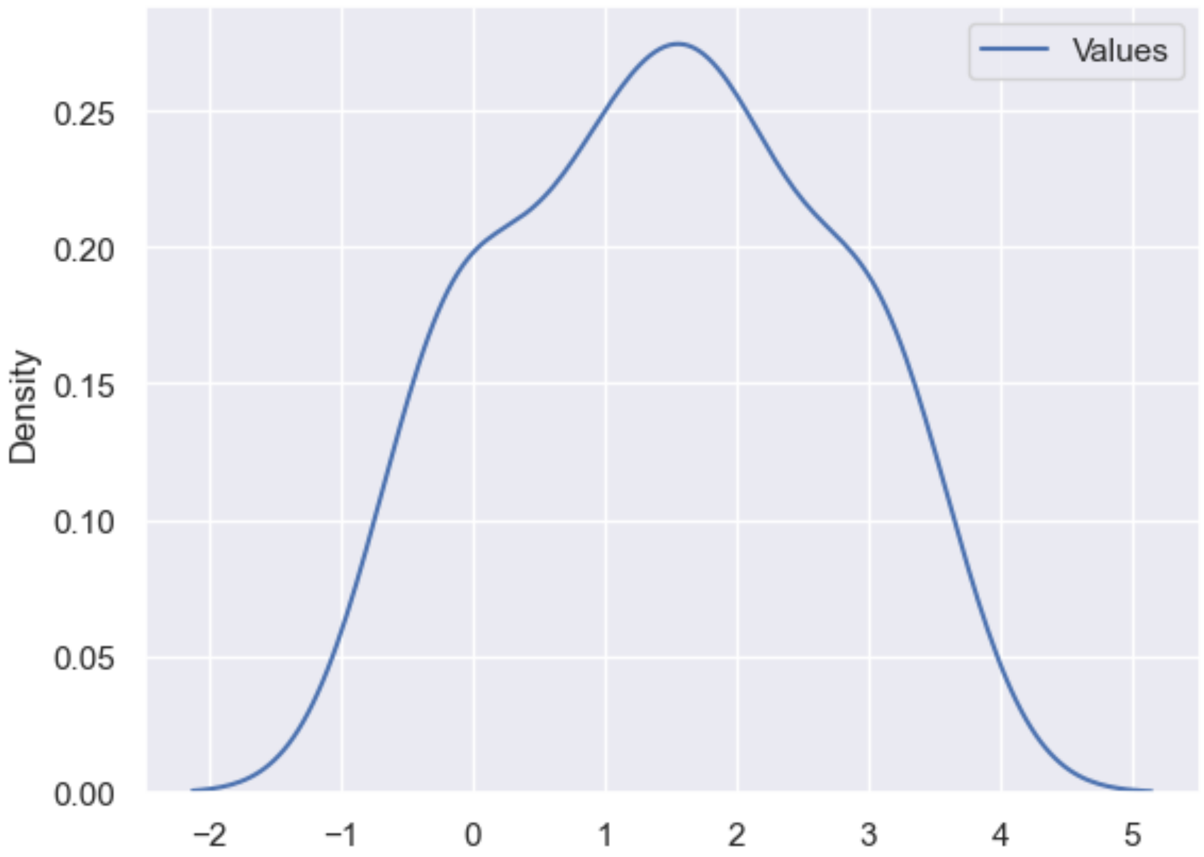
Out[2]:

	Values
0	1.824378
1	3.446273
2	-0.427239
3	1.952763
4	1.107812
5	0.880768
6	3.022545
7	0.776415
8	-0.249058
9	1.802244
10	3.100371
11	-0.083352
12	1.969301
13	0.328054
14	0.996174
15	1.715757
16	3.271927
17	1.968713
18	1.458442

19	1.762478
20	3.000002
21	-0.168349
22	2.946189
23	1.235000
24	2.831157
25	-0.358687
26	-0.011253
27	0.850426
28	1.932236
29	1.001805
30	3.108471
31	2.051314
32	0.100576
33	-0.181802
34	3.074089
35	1.110291
36	1.008308
37	-0.211590
38	1.872469
39	2.513046

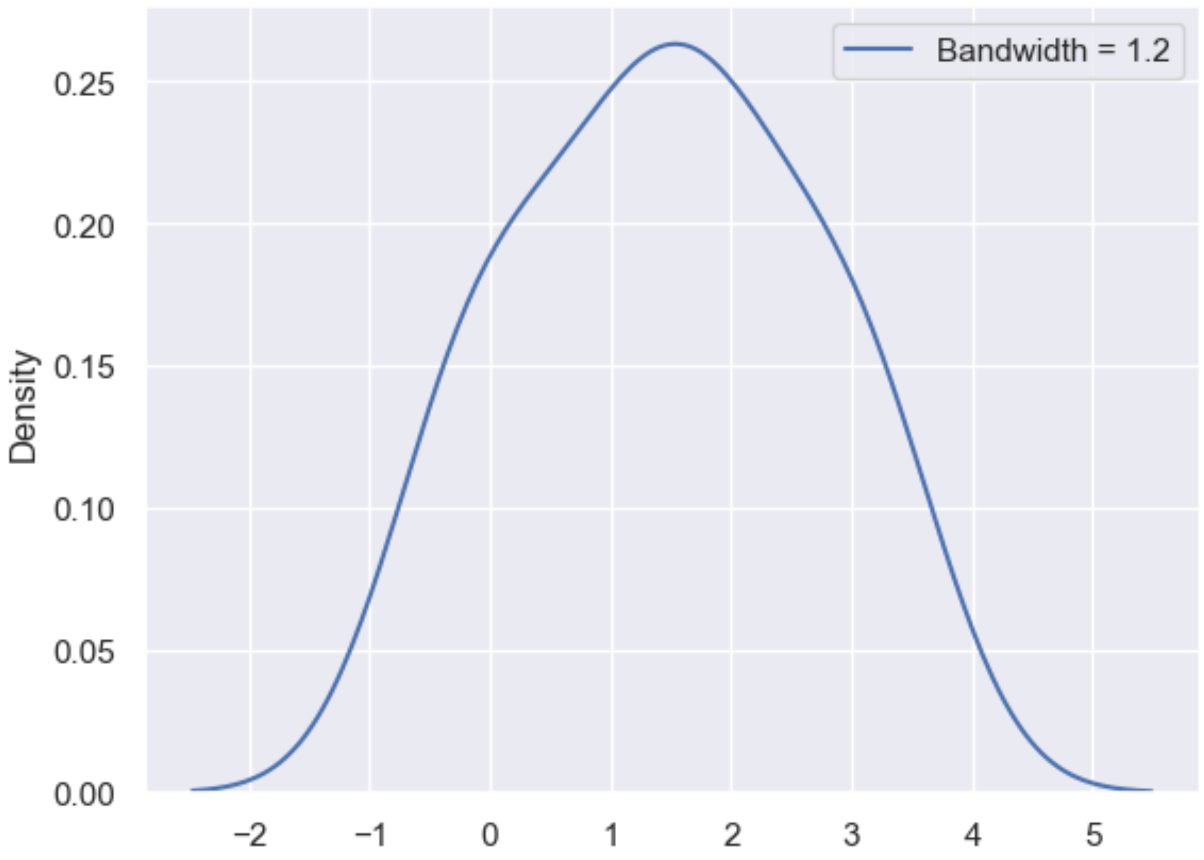
In [3]:
```
sns.kdeplot(data= tdata)
```

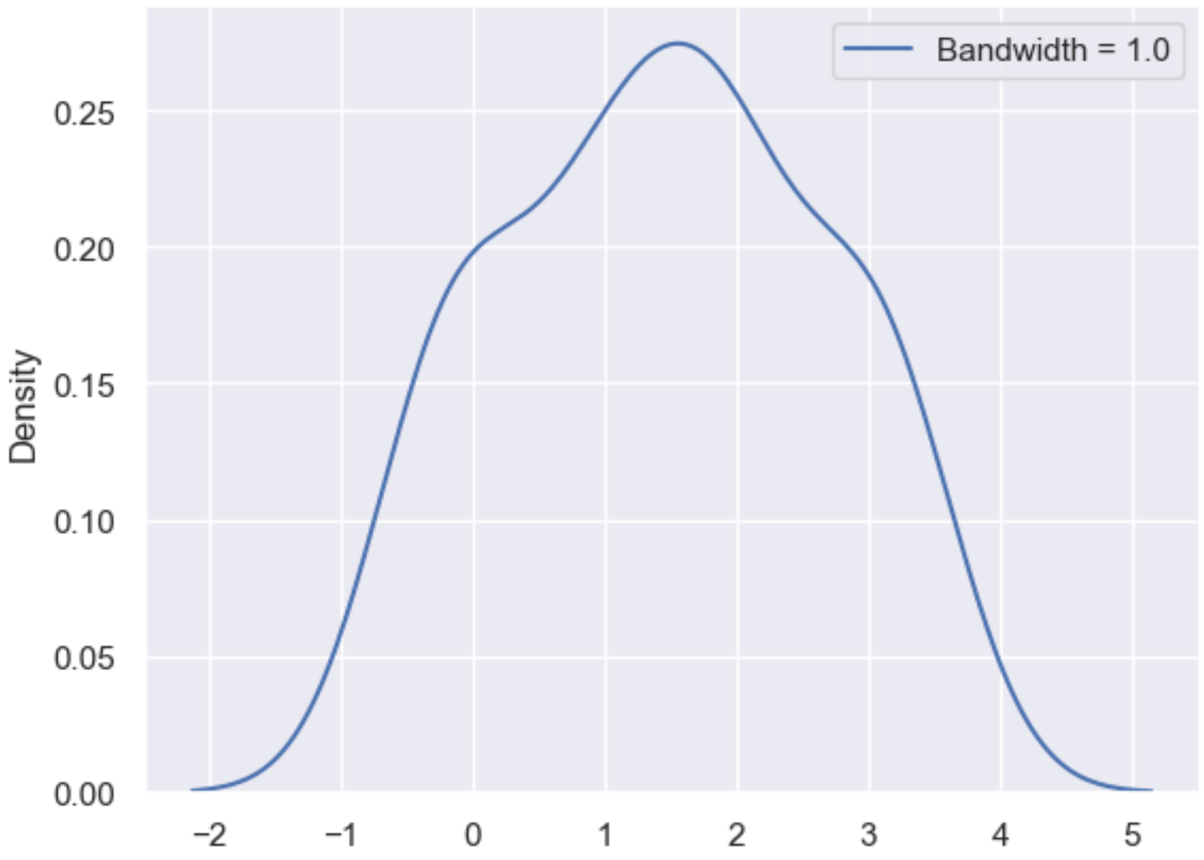
Out[3]:
```
<Axes: ylabel='Density'>
```

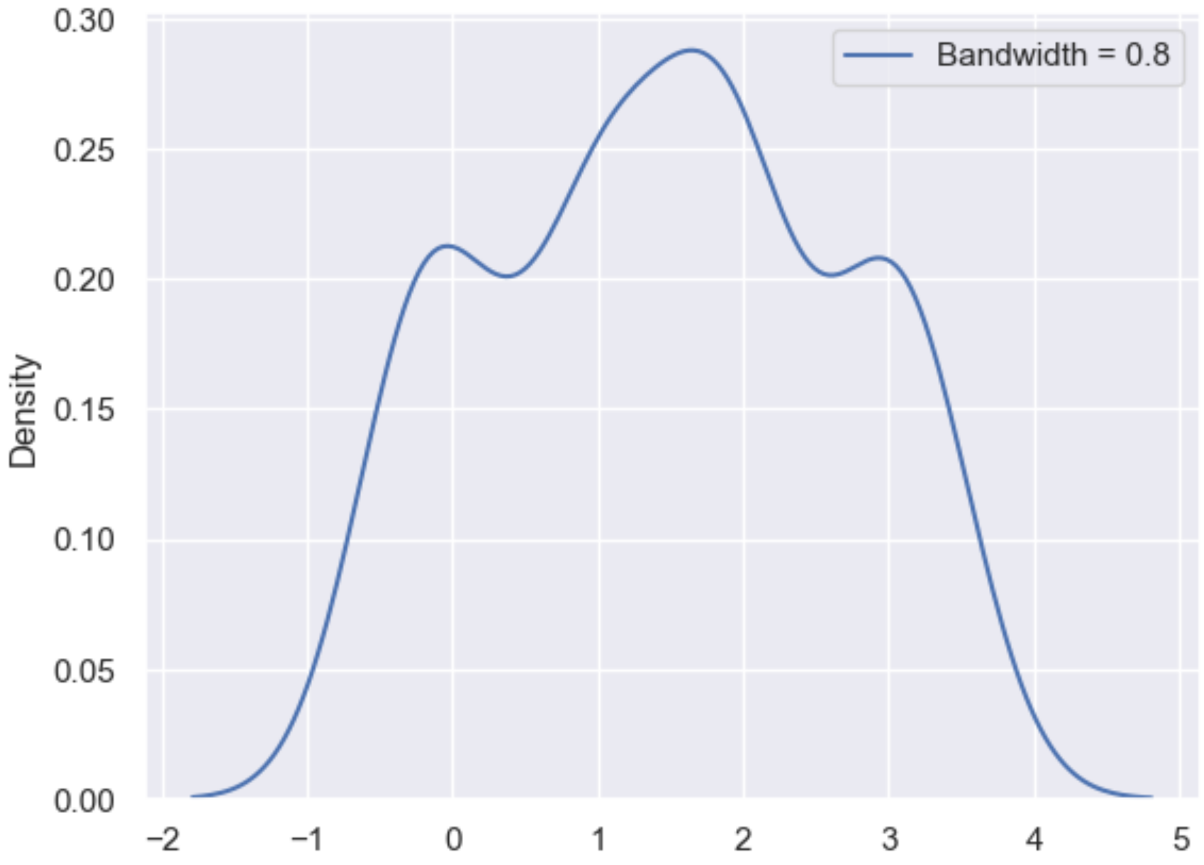


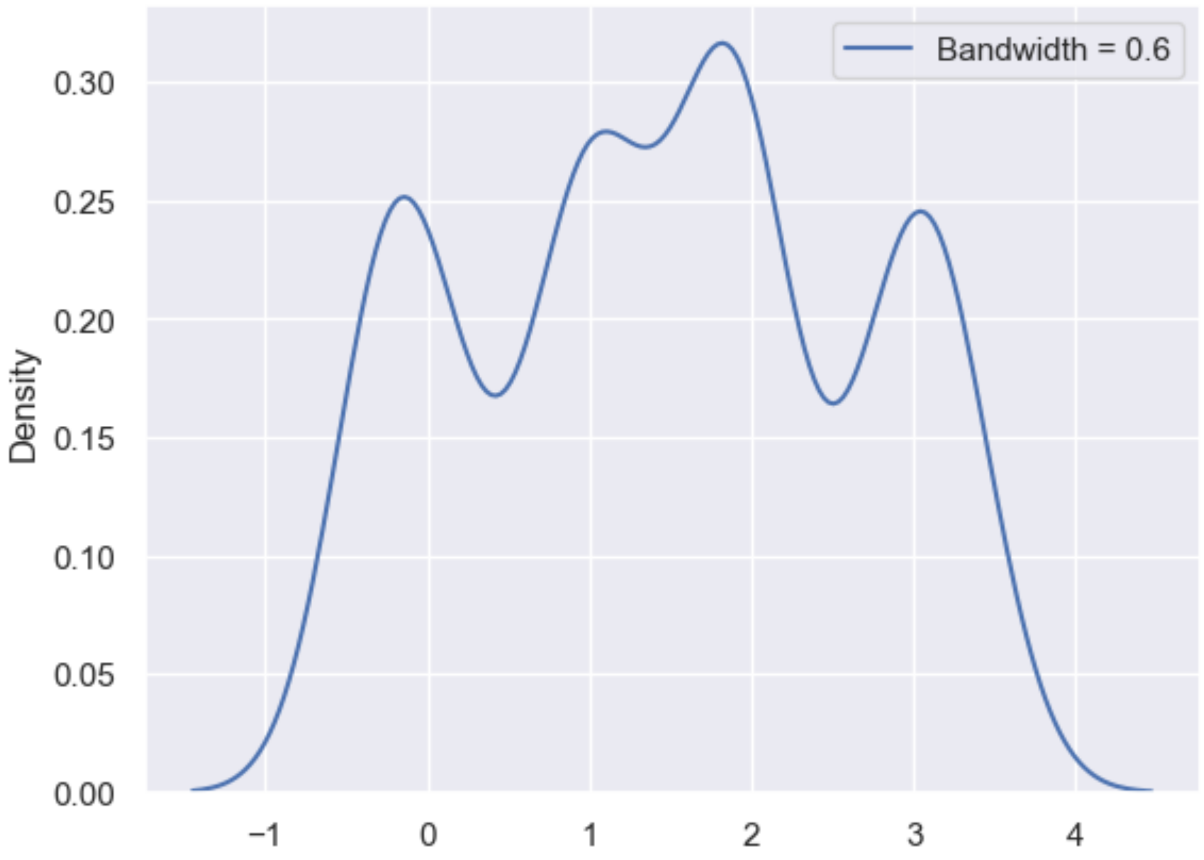
By visualization, we can see that the underlying distributiion has a mode at around 1.5.

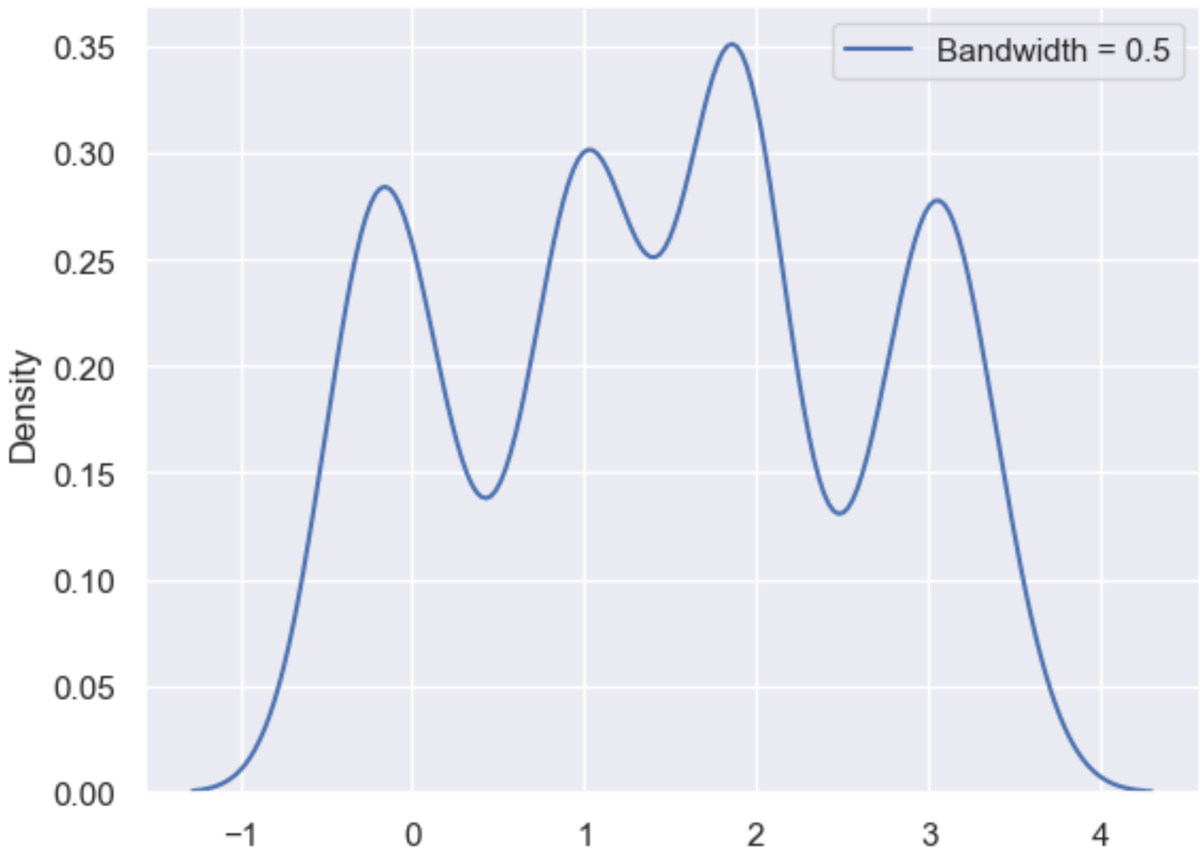
In [4]:
```
bandwidths = [1.2, 1.0, 0.8, 0.6, 0.5, 0.4, 0.2, 0.1]

for i in bandwidths:
    plt.figure()
    sns.kdeplot(tdata, bw_adjust=i, label=f'Bandwidth = {i:.1f}')
    plt.legend()
```

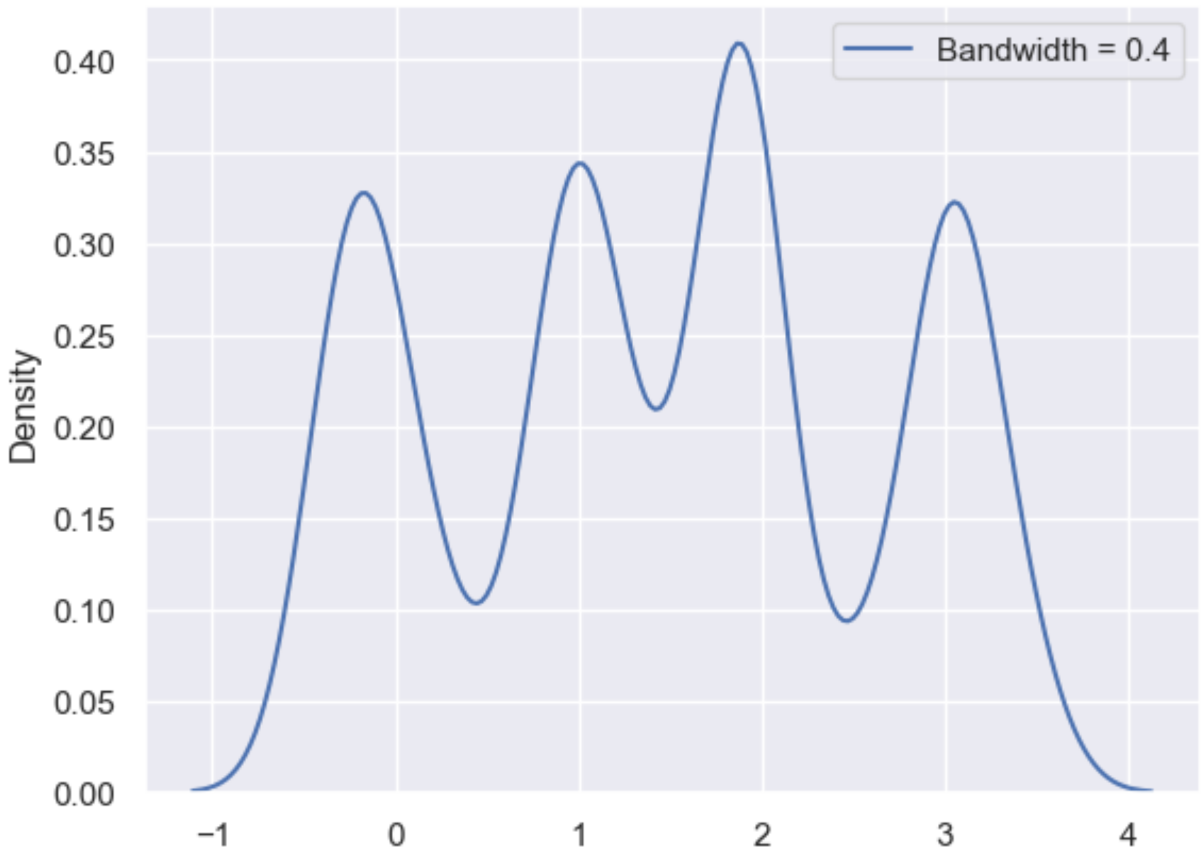

Bandwidth = 1.2
Density
0.25
0.20
0.15
0.10
0.05
0.00
−2
−1
0
1
2
3
4
5

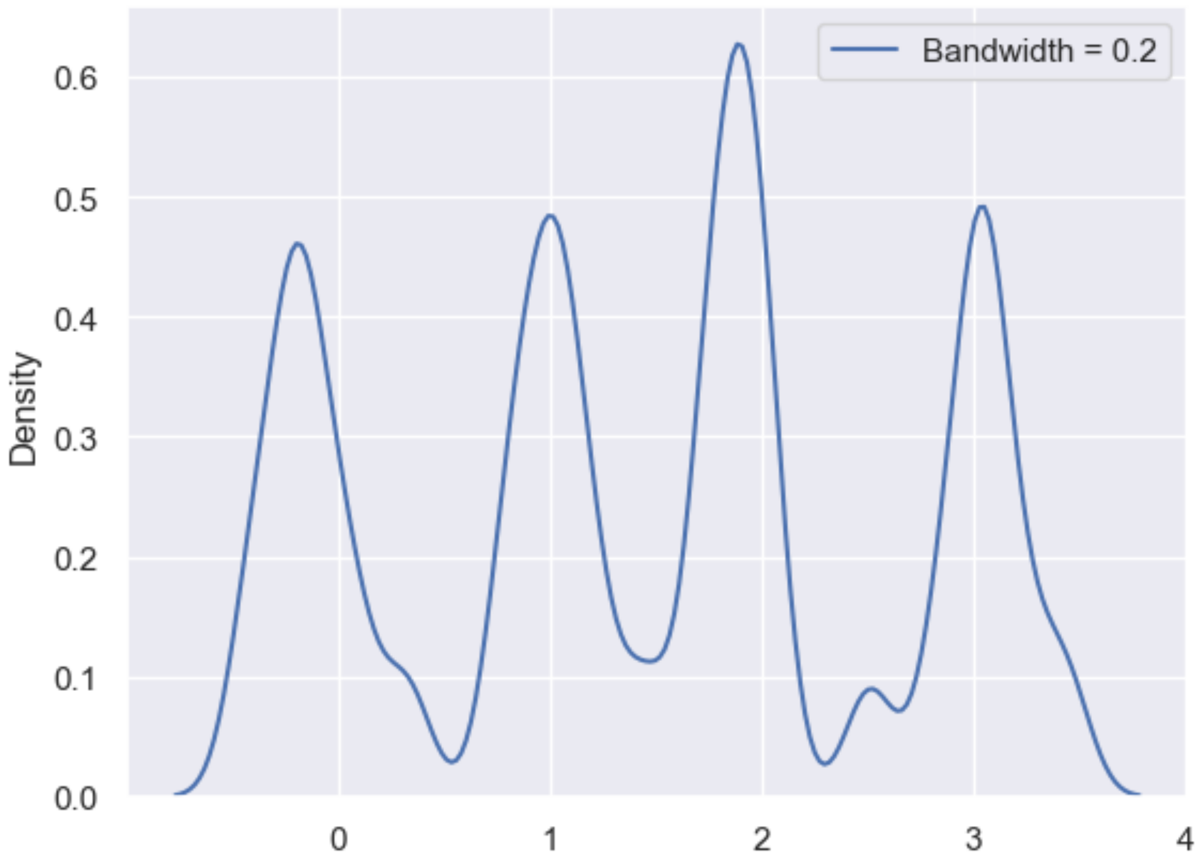

Bandwidth = 1.0
Density
0.25
0.20
0.15
0.10
0.05
0.00
−2
−1
0
1
2
3
4
5

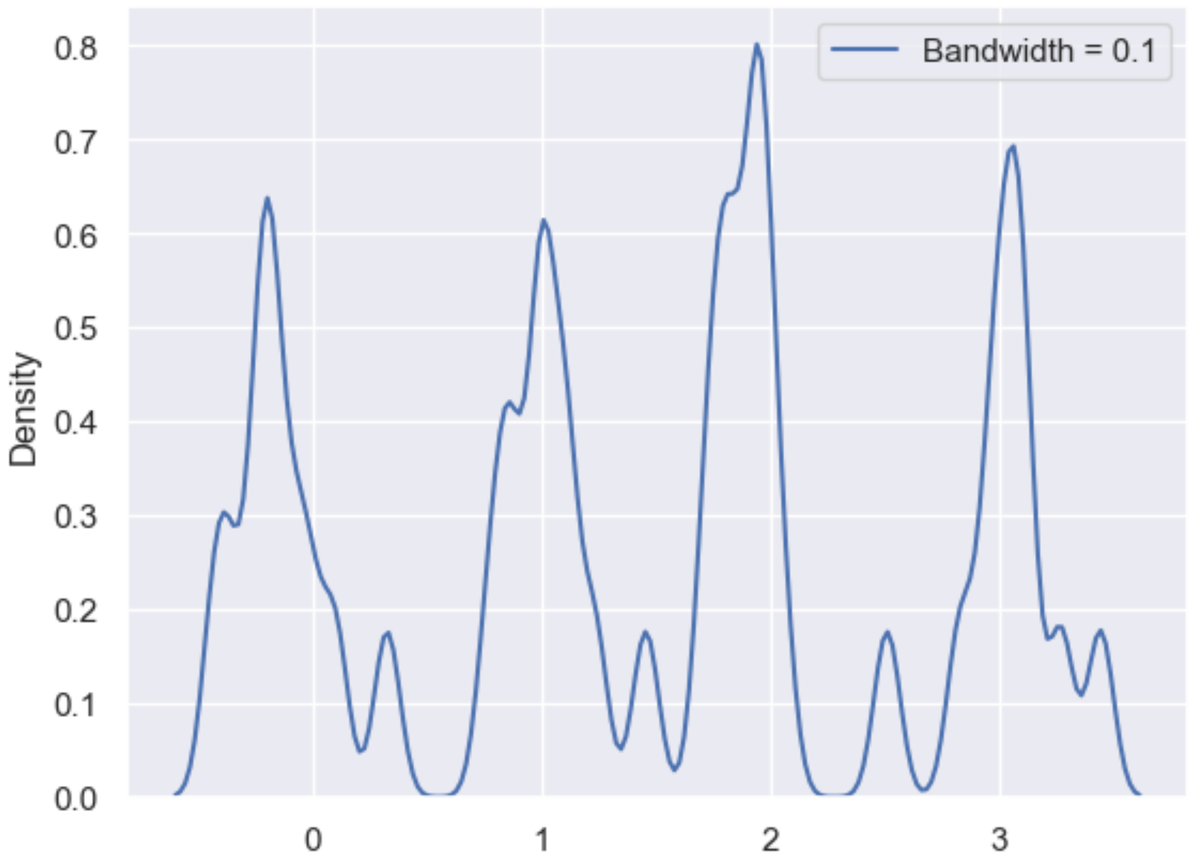

Bandwidth = 0.8
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−2
−1
0
1
2
3
4
5


Bandwidth = 0.6
Density
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4


Bandwidth = 0.5
Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4


Bandwidth = 0.4
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−1
0
1
2
3
4


Bandwidth = 0.2
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
3
4


Bandwidth = 0.1
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
1
2
3

Observations

From above KDEplots we can see that the bandwidth changes the shape of the estimated distribution greatly. As bandwidth value decreases, the distribution shape changes from the only one peak to four clear peaks (modes). The setting with 0.2 bandwidth shows the more precise distribution. From which we can observe that around -0.2, 1, 2, 3 it has the modes.

Ex. 2: Plotting Categorical Data

In [5]:
```
df2 =pd.read_excel('chronic_kidney_disease_numerical.xls')
```

In [6]:
```
df2
```

Out[6]:

	age	blood pressure	specific gravity	albumin	sugar	blood glucose random	blood urea	serum creatinine	sodium	potas
0	48.0	80.0	1.020	1.0	0.0	121.0	36.0	1.2	NaN	
1	7.0	50.0	1.020	4.0	0.0	NaN	18.0	0.8	NaN	
2	62.0	80.0	1.010	2.0	3.0	423.0	53.0	1.8	NaN	
3	48.0	70.0	1.005	4.0	0.0	117.0	56.0	3.8	111.0	
4	51.0	80.0	1.010	2.0	0.0	106.0	26.0	1.4	NaN	
...	...	...	...	...	...	...	...	...	...	
395	55.0	80.0	1.020	0.0	0.0	140.0	49.0	0.5	150.0	
396	42.0	70.0	1.025	0.0	0.0	75.0	31.0	1.2	141.0	
397	12.0	80.0	1.020	0.0	0.0	100.0	26.0	0.6	137.0	
398	17.0	60.0	1.025	0.0	0.0	114.0	50.0	1.0	135.0	
399	58.0	80.0	1.025	0.0	0.0	131.0	18.0	1.1	141.0	

400 rows × 15 columns

In [8]:
```
lm = pd.melt(df2, id_vars=['class'])
lm
```

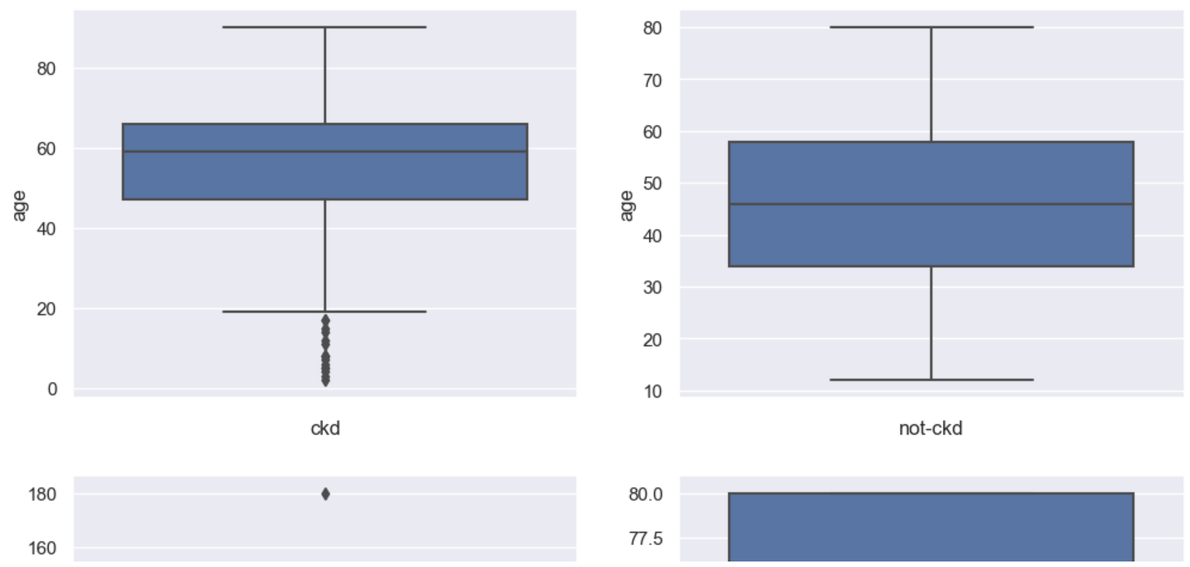
Out[8]:

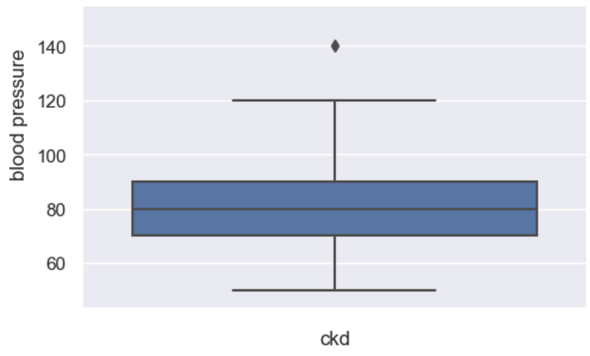
	class	variable	value
0	ckd	age	48.0
1	ckd	age	7.0
2	ckd	age	62.0
3	ckd	age	48.0
4	ckd	age	51.0
...	...	...	...
5595	notckd	red blood cell count	4.9
5596	notckd	red blood cell count	6.2
5597	notckd	red blood cell count	5.4
5598	notckd	red blood cell count	5.9
5599	notckd	red blood cell count	6.1

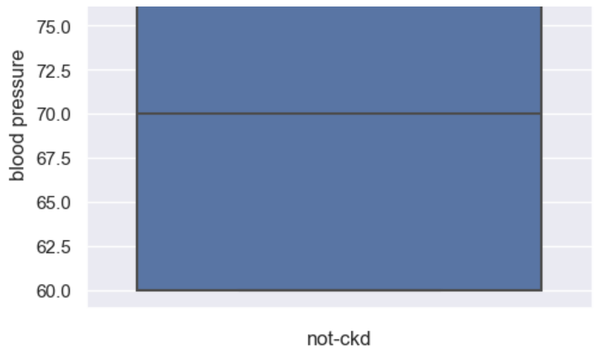
5600 rows × 3 columns

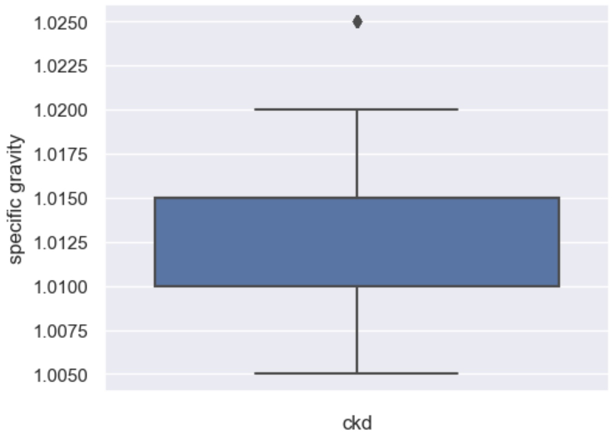
To disable sharing y axes between the two attributes, we made two groups of data on the basis of chronic kidney disease.

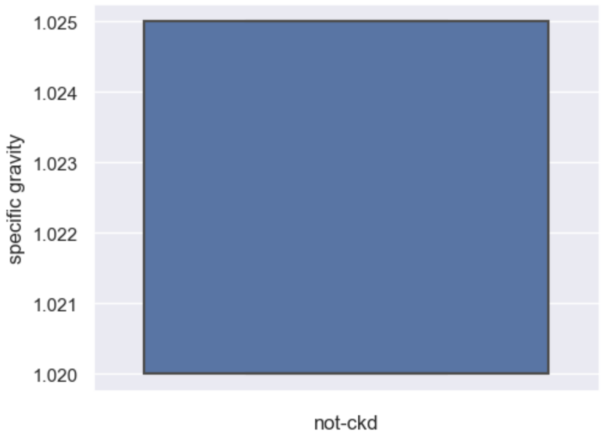
In [9]:
```
ckd_group = df2[df2['class'] == 'ckd']
non_ckd_group = df2[df2['class'] == 'notckd']
#for passing only the float and int datatype
num_attributes = [col for col in df2.columns if df2[col].dtype == 'float6

fig, axes = plt.subplots(nrows=len(num_attributes), ncols=2, figsize=(12,

for i, value in enumerate(num_attributes):
    sns.boxplot(y=value, data=ckd_group, ax=axes[i, 0])
    sns.boxplot(y=value, data=non_ckd_group, ax=axes[i, 1])
    axes[i, 0].set_xlabel('ckd')
    axes[i, 1].set_xlabel('not-ckd')
```

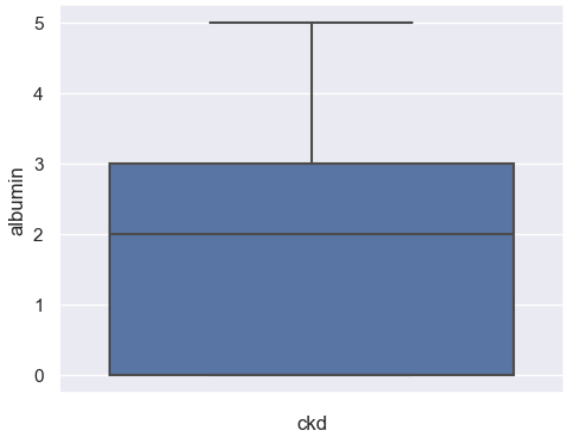


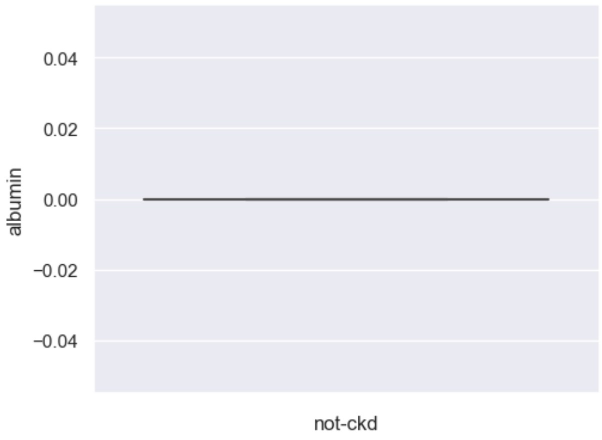

blood pressure
140
120
100
80
60
ckd

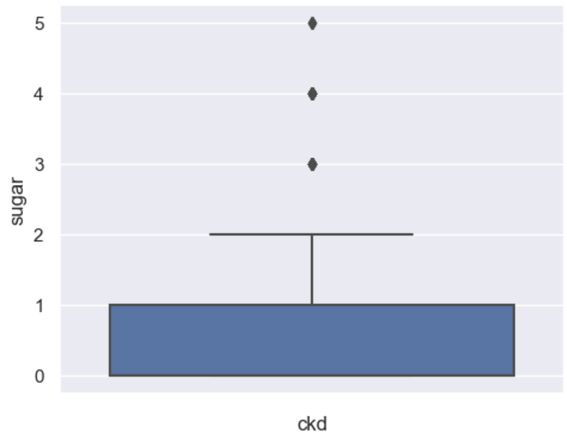

blood pressure
75.0
72.5
70.0
67.5
65.0
62.5
60.0
not-ckd

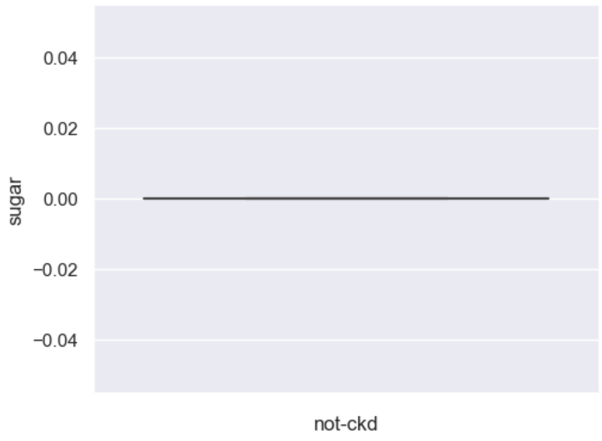

specific gravity
1.0250
1.0225
1.0200
1.0175
1.0150
1.0125
1.0100
1.0075
1.0050
ckd

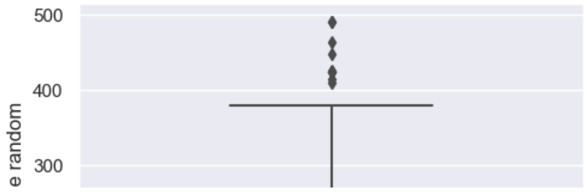

specific gravity
1.025
1.024
1.023
1.022
1.021
1.020
not-ckd

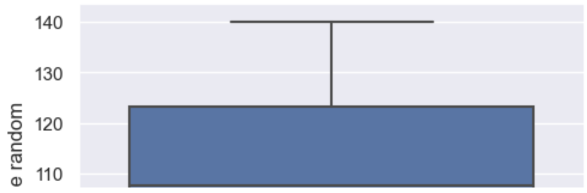

albumin
5
4
3
2
1
0
ckd

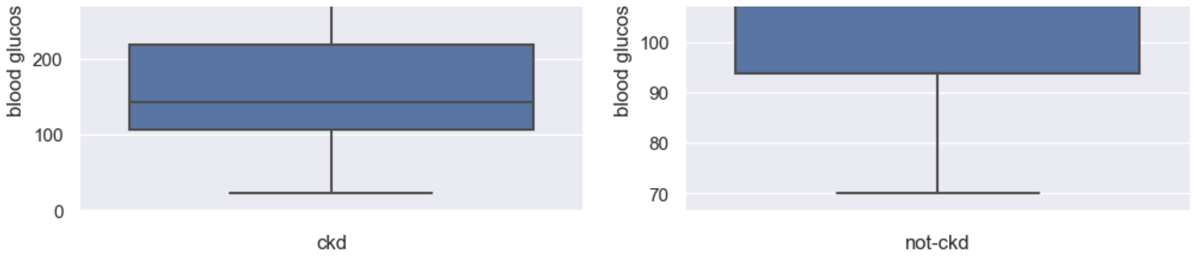

albumin
0.04
0.02
0.00
−0.02
−0.04
not-ckd

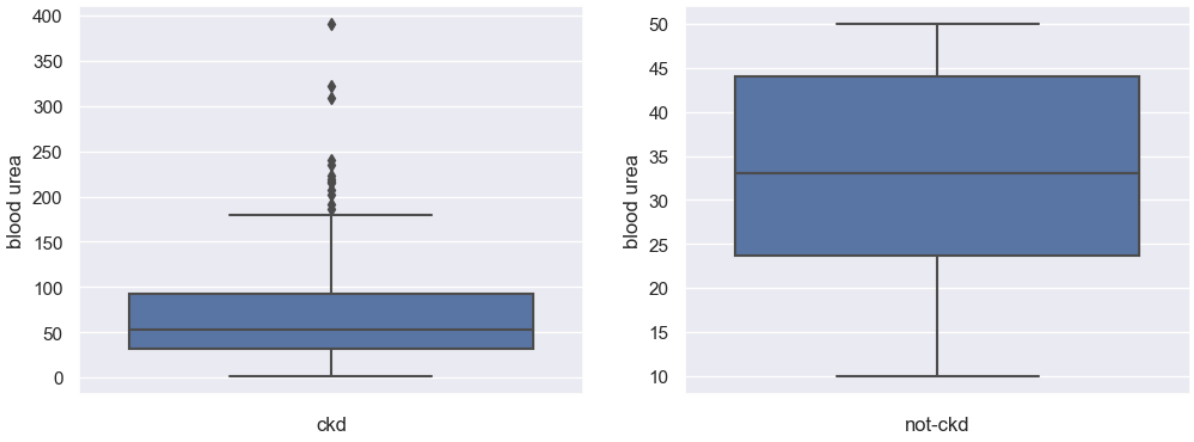

sugar
5
4
3
2
1
0
ckd

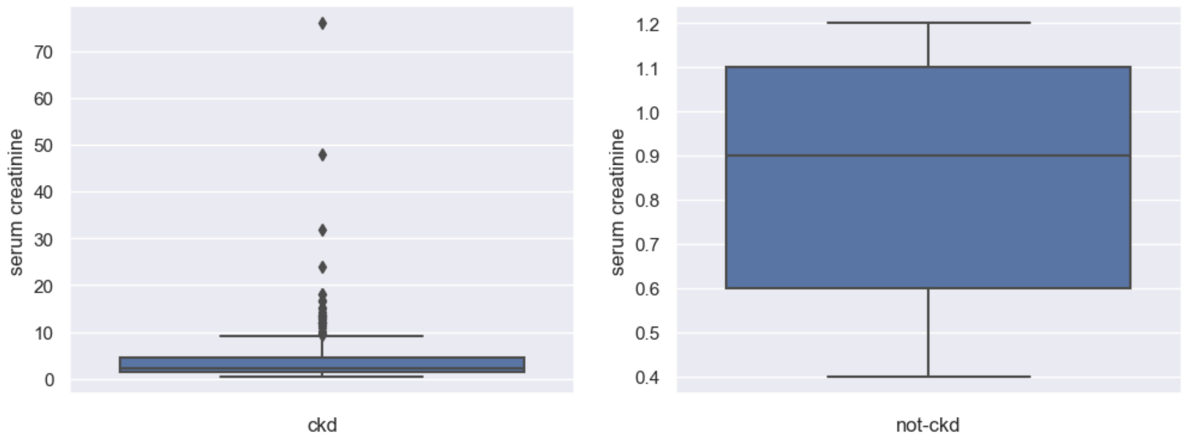

sugar
0.04
0.02
0.00
−0.02
−0.04
not-ckd

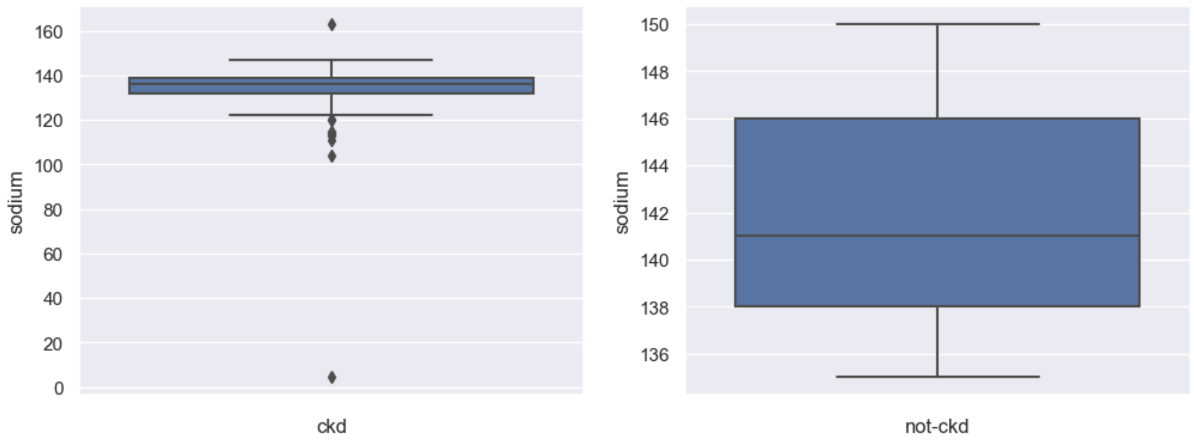

e random
500
400
300

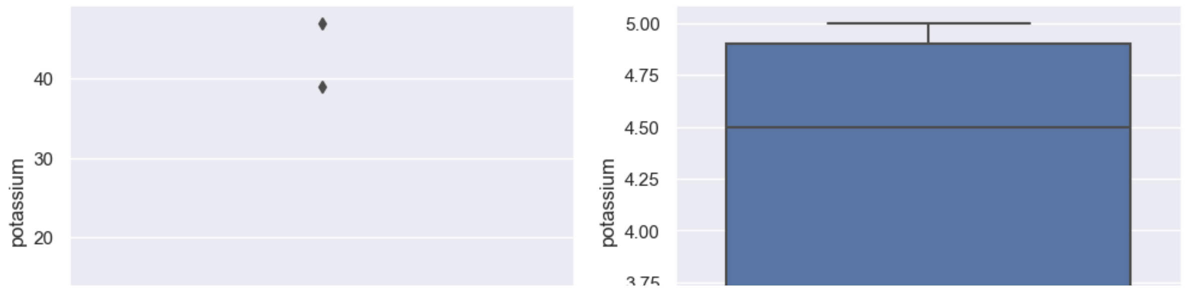

e random
140
130
120
110

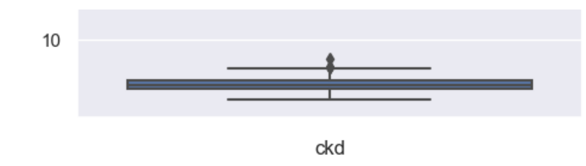

blood glucos
0
100
200
ckd
blood glucos
70
80
90
100
not-ckd

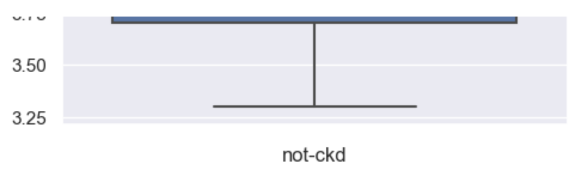

blood urea
0
50
100
150
200
250
300
350
400
ckd
blood urea
10
15
20
25
30
35
40
45
50
not-ckd

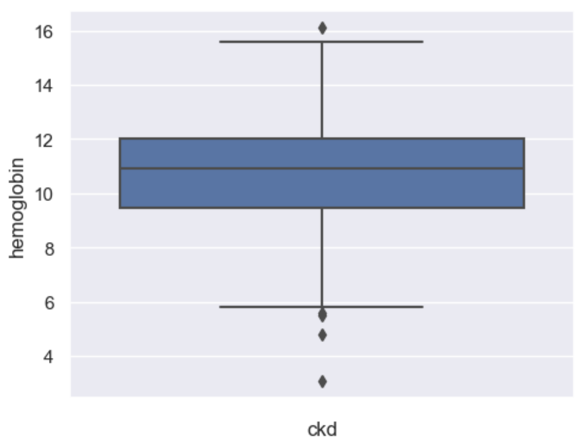

serum creatinine
0
10
20
30
40
50
60
70
ckd
serum creatinine
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
not-ckd

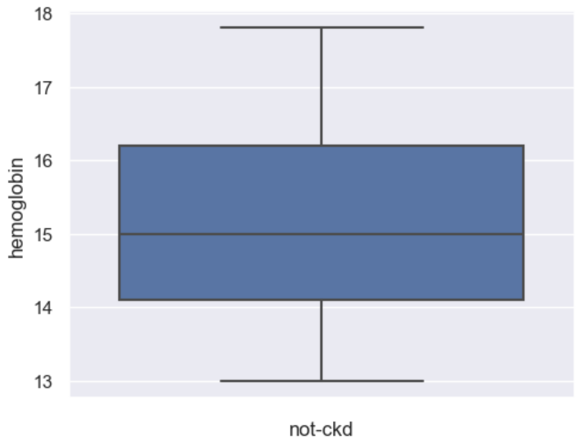

sodium
0
20
40
60
80
100
120
140
160
ckd
sodium
136
138
140
142
144
146
148
150
not-ckd

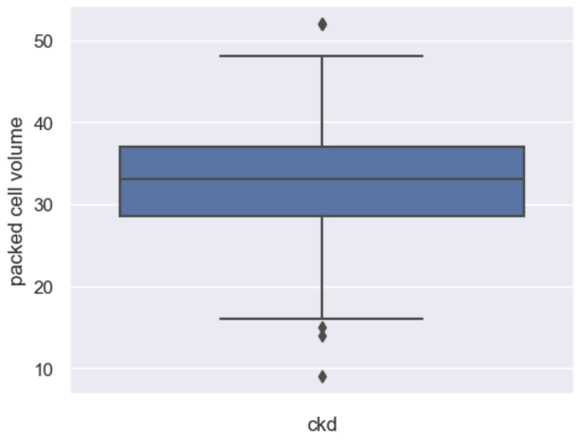

potassium
20
30
40
potassium
4.00
4.25
4.50
4.75
5.00

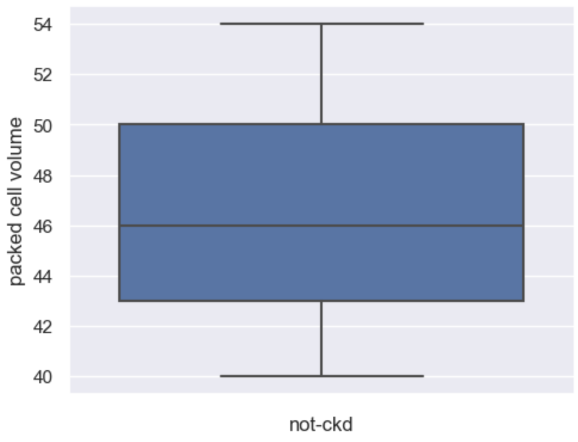

10
ckd

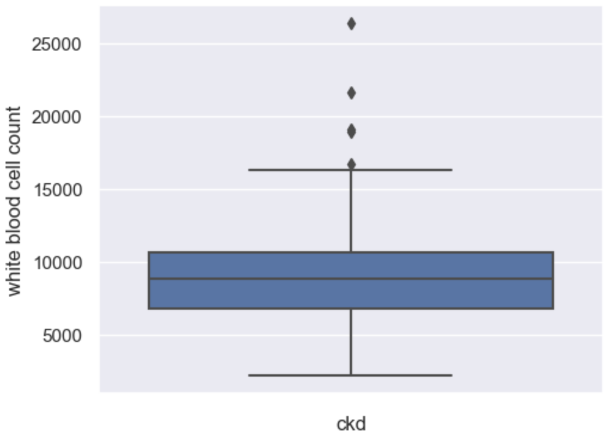

3.50
3.25
not-ckd

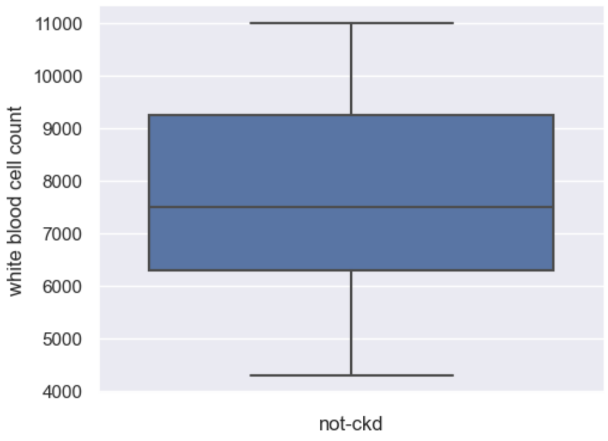

hemoglobin
16
14
12
10
8
6
4
ckd

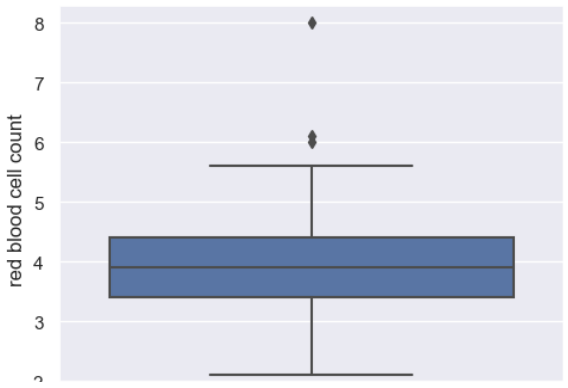

hemoglobin
18
17
16
15
14
13
not-ckd

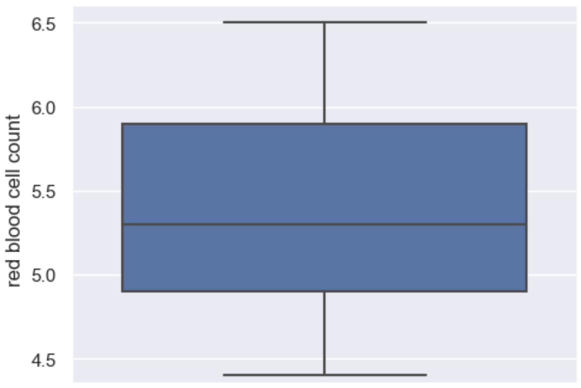

packed cell volume
50
40
30
20
10
ckd

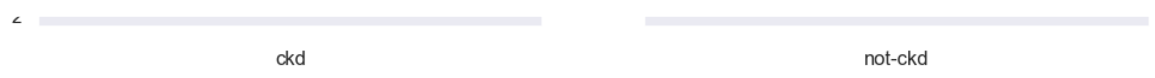

packed cell volume
54
52
50
48
46
44
42
40
not-ckd


white blood cell count
25000
20000
15000
10000
5000
ckd


white blood cell count
11000
10000
9000
8000
7000
6000
5000
4000
not-ckd


red blood cell count
8
7
6
5
4
3


red blood cell count
6.5
6.0
5.5
5.0
4.5



By visualization, we can see that the 'albumin' attribute is a high indicator of chronic kidney disease. Whereas the 'potassium' seems to be unrelated to it.

Ex.3: Multidimensional data filtering and visualization

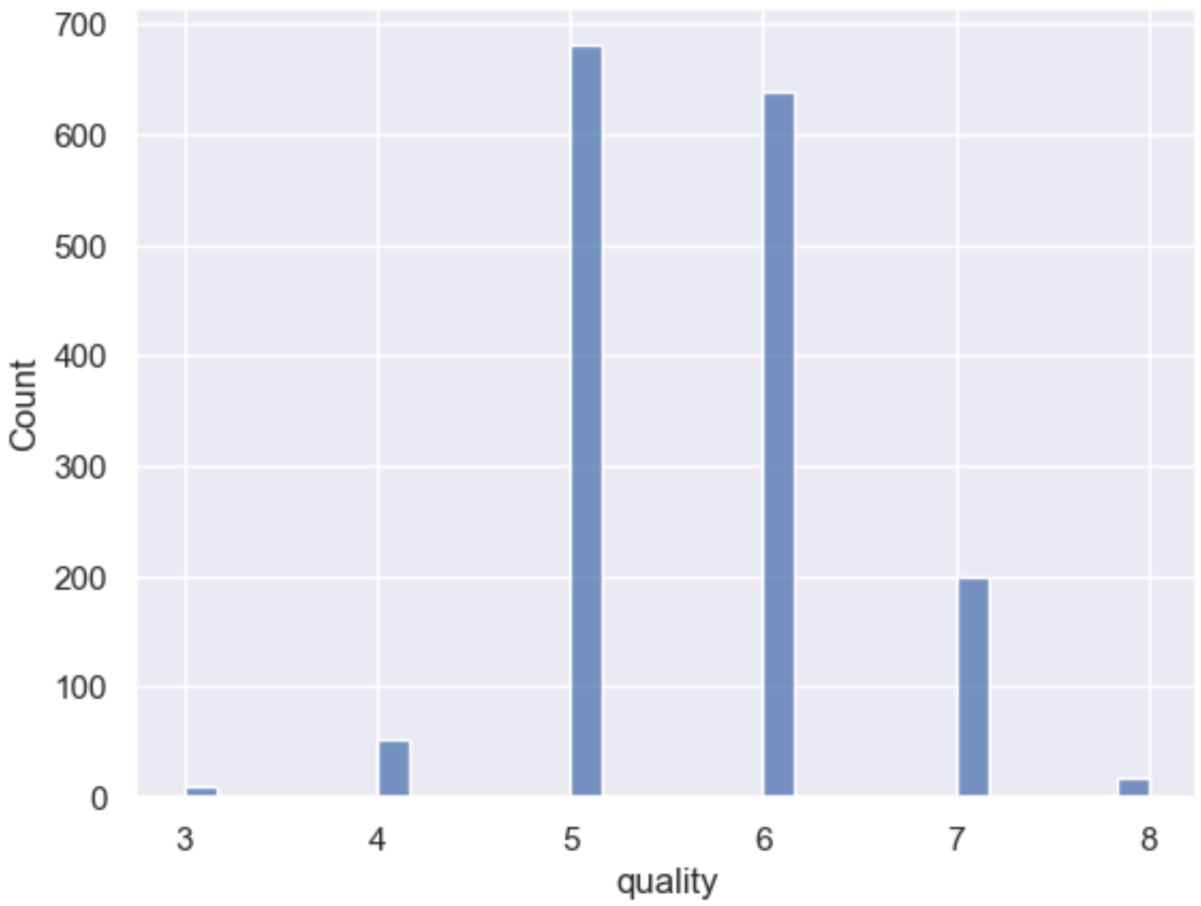
In [10]:
```
df = pd.read_csv("winequality-red.csv", sep=';')
df.head(10)
```

Out[10]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80

In [11]:
```
sns.histplot(df, x="quality")
```

Out[11]:
```
<Axes: xlabel='quality', ylabel='Count'>
```



In [12]:
```
wine_quality = {3:"low",4:"low",5:"medium",6:"medium",7:"high",8:"high"}
df_renamed = df.rename(columns={'quality':'quality bin'})
```

In [13]:
```
df_renamed['quality bin'] = df_renamed['quality bin'].map(wine_quality)
```

In [14]:
```
df_renamed["quality bin"].head(20)
```

Out[14]:
```
0     medium
1     medium
2     medium
3     medium
4     medium
5     medium
6     medium
7       high
8       high
9     medium
10    medium
11    medium
12    medium
13    medium
14    medium
15    medium
16      high
17    medium
18       low
19    medium
Name: quality bin, dtype: object
```

In [15]:
```
df_renamed.head(10)
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80

In [16]:
```
df_quality_filter = df_renamed[(df_renamed["quality bin"] == "low") | (df
```

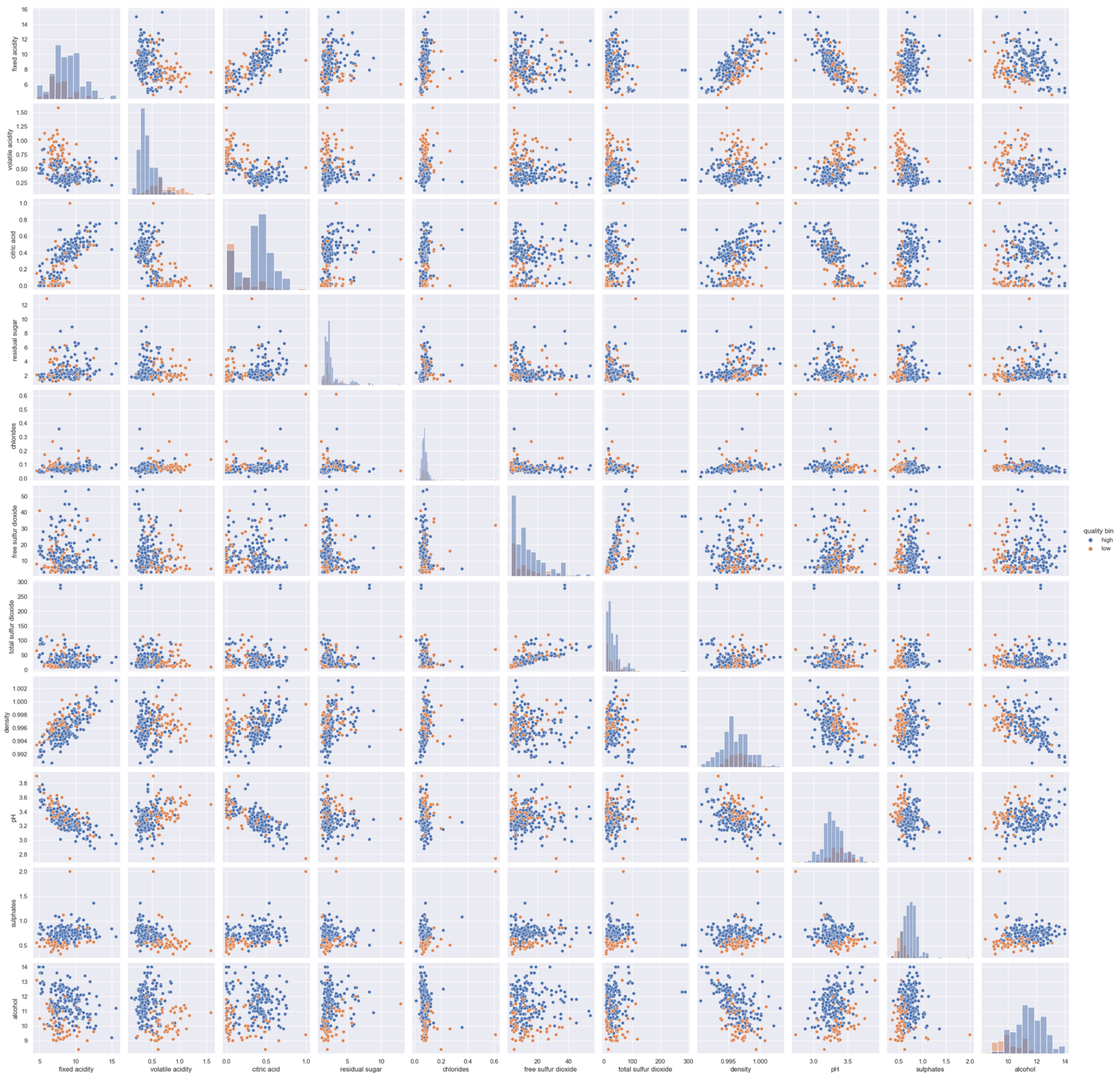
In [24]:
```
df_quality_filter.head(20)
```

Out[24]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphat
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.99460	3.39	0.
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.99680	3.36	0.
16	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.99690	3.30	0.
18	7.4	0.590	0.08	4.4	0.086	6.0	29.0	0.99740	3.38	0.
37	8.1	0.380	0.28	2.1	0.066	13.0	30.0	0.99680	3.23	0.
38	5.7	1.130	0.09	1.5	0.172	7.0	19.0	0.99400	3.50	0.
41	8.8	0.610	0.30	2.8	0.088	17.0	46.0	0.99760	3.26	0
45	4.6	0.520	0.15	2.1	0.054	8.0	65.0	0.99340	3.90	0.
62	7.5	0.520	0.16	1.9	0.085	12.0	35.0	0.99680	3.38	0.
73	8.3	0.675	0.26	2.1	0.084	11.0	43.0	0.99760	3.31	0.
79	8.3	0.625	0.20	1.5	0.080	27.0	119.0	0.99720	3.16	1
94	5.0	1.020	0.04	1.4	0.045	41.0	85.0	0.99380	3.75	0.
128	8.0	0.590	0.16	1.8	0.065	3.0	16.0	0.99620	3.42	0.
151	9.2	0.520	1.00	3.4	0.610	32.0	69.0	0.99960	2.74	2.
161	7.6	0.680	0.02	1.3	0.072	9.0	20.0	0.99650	3.17	1.
167	7.3	0.550	0.03	1.6	0.072	17.0	42.0	0.99560	3.37	0.
170	7.9	0.885	0.03	1.8	0.058	4.0	8.0	0.99720	3.36	0.
198	5.4	0.835	0.08	1.2	0.046	13.0	93.0	0.99240	3.57	0.
199	6.9	1.090	0.06	2.1	0.061	12.0	31.0	0.99480	3.51	0.
200	9.6	0.320	0.47	1.4	0.056	9.0	24.0	0.99695	3.22	0.

In [25]:
```
sns.pairplot(data=df_quality_filter, hue="quality bin",diag_kind="hist")
```

Out[25]:
```
<seaborn.axisgrid.PairGrid at 0x17f7bb8b0>
```



list of attributes that distinguish high quality out of low quality wine:

1. Volatile Acidity
2. Citric Acid
3. PH level
4. Sulphates
5. Alcohol

In [26]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [27]:
```
X = df_quality_filter.iloc[:,0:11]
y = df_quality_filter.iloc[:,11]
```

In [28]:
```
selector = SelectKBest(f_classif, k=5)
selector.fit(X,y)
cols = selector.get_support(indices=True)
df_focused = df_quality_filter.iloc[:,cols]
```

In [29]:
```
df_focused.head(10)
```

Out[29]:

	volatile acidity	citric acid	pH	sulphates	alcohol
7	0.650	0.00	3.39	0.47	10.0
8	0.580	0.02	3.36	0.57	9.5
16	0.280	0.56	3.30	0.75	10.5
18	0.590	0.08	3.38	0.50	9.0
37	0.380	0.28	3.23	0.73	9.7
38	1.130	0.09	3.50	0.48	9.8
41	0.610	0.30	3.26	0.51	9.3
45	0.520	0.15	3.90	0.56	13.1
62	0.520	0.16	3.38	0.62	9.5
73	0.675	0.26	3.31	0.53	9.2

In [41]:
```
df_focused = df_focused.copy()
df_focused["quality bin"] = df_quality_filter['quality bin']
```

In [42]:
```
df_focused
```

Out[42]:

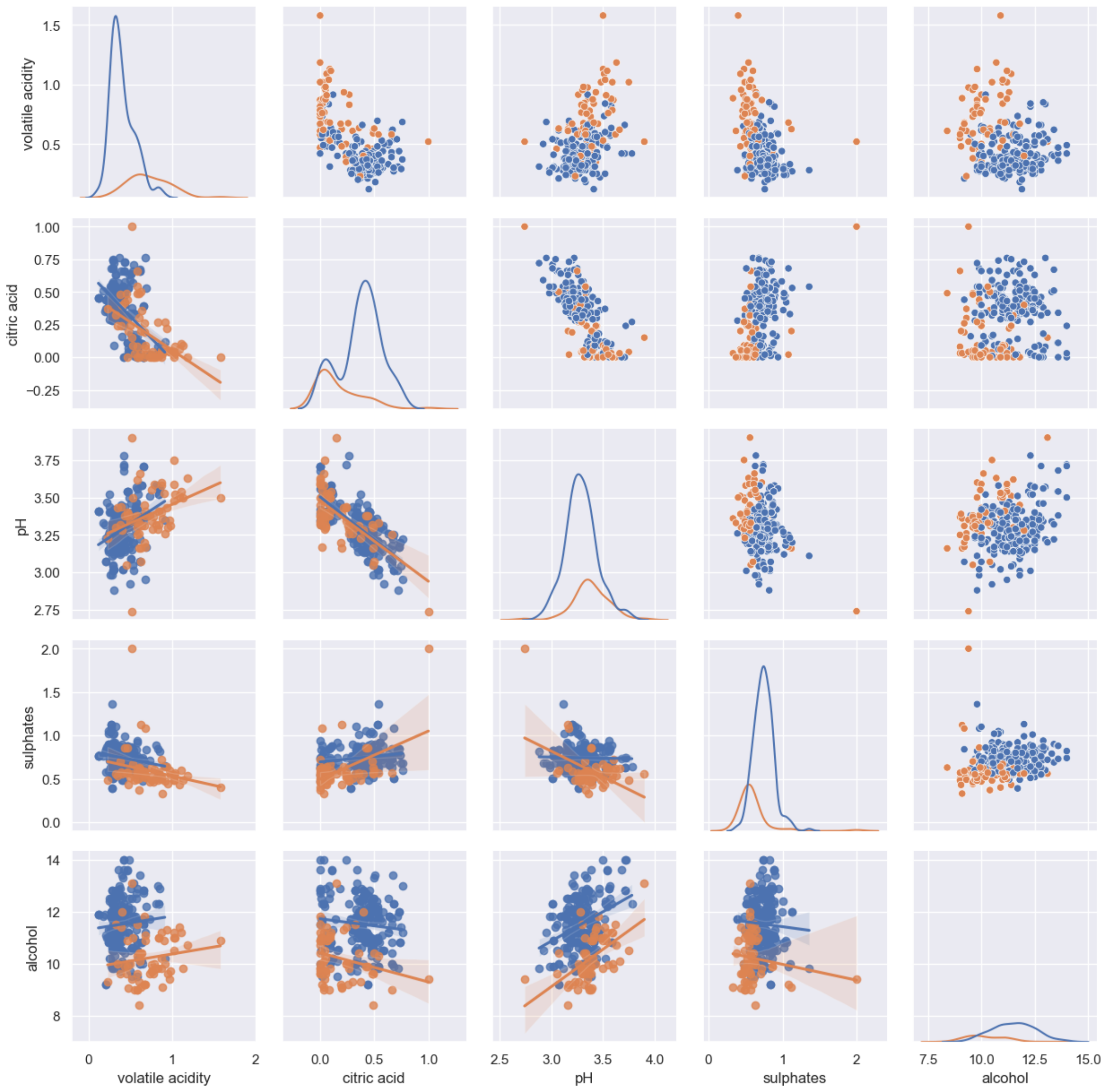
	volatile acidity	citric acid	pH	sulphates	alcohol	quality bin
7	0.65	0.00	3.39	0.47	10.00	high
8	0.58	0.02	3.36	0.57	9.50	high
16	0.28	0.56	3.30	0.75	10.50	high
18	0.59	0.08	3.38	0.50	9.00	low
37	0.38	0.28	3.23	0.73	9.70	high
...	...	...	...	...	...	...
1541	0.25	0.29	3.40	0.76	10.90	high
1544	0.37	0.43	3.17	0.81	11.20	high
1549	0.36	0.30	3.24	0.70	11.40	high
1555	0.56	0.17	3.44	0.68	10.55	high
1584	0.32	0.44	3.29	0.80	11.60	high

280 rows × 6 columns

In total, we can see, that attributes, that were chosen, are identical with those after feature selection

In [32]:
```
g = sns.PairGrid(df_focused, hue="quality bin")
g.map_diag(sns.kdeplot)
g.map_upper(sns.scatterplot)
g.map_lower(sns.regplot)
```

Out[32]:
```
<seaborn.axisgrid.PairGrid at 0x284205960>
```



Regardless of quality of wine we have Alcohol percentage, PH level and Citric Acid that have strong correlation At the same time there is Sulphates that has different correlation depending on the wine quality To my mind, citric acid definitely has multimodal distribution. Moreover, I think that Alcohol percentage can also have multimodal distribution We can easily see four outliers all in low quality wine. It's:

1. 1.580 volatile acidity
2. 1.0 citric acid
3. 2.74 pH
4. 2.0 sulphates

In [33]:
```
from numpy import sqrt

def distanceCalc(xA,xB,yA,yB):
    return sqrt(pow((xA-xB),2) + pow((yA-yB),2))
```

In [34]:
```
def DSC(highDSC,lowDSC,clustersPoints):
    dscL = dscH = 0
    for i in range(len(highDSC)):
        if (distanceCalc(clustersPoints[0][0],highDSC[i][0],clustersPoint
         dscH += 1
        # print(dscH)
  #  print("||")
    for i in range(len(lowDSC)):
        if (distanceCalc(clustersPoints[1][0],lowDSC[i][0],clustersPoints
         dscL += 1
        # print(dscL)
    return {(dscH/len(highDSC))*100,(dscL/len(lowDSC))*100}
```

In [35]:
```
clusterCenters = df_focused.groupby(["quality bin"]).mean()[:2].values
```

In [36]:
```
labels = df_focused.columns.values
lowPoints = df_focused.loc[df_focused["quality bin"] == "low", labels[:5]
highPoints = df_focused.loc[df_focused["quality bin"] == "high", labels[:
```

In [37]:
```
for i in range(1,5):
    lowPair = [[]]* len(lowPoints)
    highPair = [[]]* len(highPoints)
    for j in range(len(lowPoints)):
        lowPair[j] = [lowPoints[j][0],lowPoints[j][i]]
    for g in range(len(highPoints)):
        highPair[g] = [highPoints[g][0],highPoints[g][i]]
    firstClusters = [[clusterCenters[0][0],clusterCenters[0][i]],[cluster
    print("pair of: Volatile acidity and ",i,DSC(highPair, lowPair, first
```

```
pair of: Volatile acidity and  1 {26.984126984126984, 20.737327188940093}
pair of: Volatile acidity and  2 {16.129032258064516, 26.984126984126984}
pair of: Volatile acidity and  3 {14.285714285714285, 22.22222222222222}
pair of: Volatile acidity and  4 {25.806451612903224, 30.158730158730158}
```

In [38]:
```
for i in range(2,5):
    lowPair = [[]]* len(lowPoints)
    highPair = [[]]* len(highPoints)
    for j in range(len(lowPoints)):
        lowPair[j] = [lowPoints[j][1],lowPoints[j][i]]
    for g in range(len(highPoints)):
        highPair[g] = [highPoints[g][1],highPoints[g][i]]
    firstClusters = [[clusterCenters[0][1],clusterCenters[0][i]],[cluster
    print("pair of: Citric Acid and ",i,DSC(highPair, lowPair, firstClust
```

```
pair of: Citric Acid and  2 {28.57142857142857, 22.58064516129032}
pair of: Citric Acid and  3 {22.22222222222222, 23.04147465437788}
pair of: Citric Acid and  4 {25.806451612903224, 28.57142857142857}
```

In [110…

```
for i in range(3,5):
    lowPair = [[]]* len(lowPoints)
    highPair = [[]]* len(highPoints)
    for j in range(len(lowPoints)):
        lowPair[j] = [lowPoints[j][2],lowPoints[j][i]]
    for g in range(len(highPoints)):
        highPair[g] = [highPoints[g][2],highPoints[g][i]]
    firstClusters = [[clusterCenters[0][2],clusterCenters[0][i]],[cluster
    print("pair of: pH and ",i,DSC(highPair, lowPair, firstClusters))
```

```
pair of: pH and  3 {21.658986175115206, 15.873015873015872}
pair of: pH and  4 {33.333333333333, 26.72811059907834}
```

In [39]:

```
for i in range(4,5):
    lowPair = [[]]* len(lowPoints)
    highPair = [[]]* len(highPoints)
    for j in range(len(lowPoints)):
        lowPair[j] = [lowPoints[j][3],lowPoints[j][i]]
    for g in range(len(highPoints)):
        highPair[g] = [highPoints[g][3],highPoints[g][i]]
    firstClusters = [[clusterCenters[0][3],clusterCenters[0][i]],[cluster
    print("pair of: Sulphates and ",i,DSC(highPair, lowPair, firstCluster
```

```
pair of: Sulphates and  4 {33.333333333333, 26.72811059907834}
```

So in total we can highlight 3 main pairs of arguments:

1. Volatile acidity and Alcohol
2. pH and Alcohol
3. Sulphates and Alcohol

Ex.4: Evaluating PCP Variants

a)

in addition to standard PCP, the authors tested the following eight variations:

1. In the first variation, they added scatter plots into standard PCP. They further rotated the scatter plots by 45° on axis labels. They expected that the embedding of the scatter plots to PCP will make it more intuitive to vizualise.
2. In the second variant, they added density based coloring to the polylines of the standard PCP using lookup table(LUT). They expected it to improve visualisation because of the strong visual cue of colors.
3. They changed opacity with respect to density of the polylines to achive blending. Authors expected it to reduce the noise as noise reduction will make the pcps easy to comprehend.
4. In the this variation, authors combined previous two(color, blending) techniques together. According to them ColorBlending should have the properties of both variations hence will prove to be a more better variant.
5. This variation replaces straight lines with curves to improve the clarity and smoothness of the visualization. They expected it to remove ambiguities.
6. Random Tour variation is a rotation of PCPs by animation which shows different angles using time dependent matrix. Authors expected it to perform not as well as non-animated variations because even though it shows new patterns of data to the viewer, it is still difficult to comprehend.
7. Permutation tour is an alternative animation scheme which smoothly cycles through all the permutations, showing all possible axis in an animated fashion. For this, they had same expectations as Random Tour because both of these are animated techniques.
8. In Wobble Variation, polylines wobble around a fixed base position in a sinusoidal way and keeps clusters fairly fixed. Because of its fixed base position, authors expected it to perform better than previous animated techniques.

b)

Authors did not expected the Color, Blending and ColorBlending variations to perform that badly. To their surprise, scatter plots regardless of their simplicity outperformed every other variation. They were also wrong with placing the standard pcp at last position as it performed 2nd best. However, they guessed quite accurately about the animation related variations(random tour, permutation tour, wobble).

c)

If we had to design a visualization with 5 or less dimensions, we'll prefer the color blend variation as it makes the pcp very clear to visualize as long as you have less than 6 dimensions. otherwise, we'll prefer the scatterplot variation as it turned out to be more understandable to the viewers as the findings of Li et al. [LMvW08] and this paper clearly indicates.