TD2_Python

March 31, 2021

[1]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import sys
import seaborn as sns
```

[2]:
```
prostate = pd.read_table("prostate.data")
```

[3]:
```
prostate
```

[3]:
```
    Unnamed: 0    lcavol   lweight  age      lbph  svi       lcp  gleason  \
0            1 -0.579818  2.769459   50 -1.386294    0 -1.386294        6
1            2 -0.994252  3.319626   58 -1.386294    0 -1.386294        6
2            3 -0.510826  2.691243   74 -1.386294    0 -1.386294        7
3            4 -1.203973  3.282789   58 -1.386294    0 -1.386294        6
4            5  0.751416  3.432373   62 -1.386294    0 -1.386294        6
..         ...       ...       ...  ...       ...  ...       ...      ...
92          93  2.830268  3.876396   68 -1.386294    1  1.321756        7
93          94  3.821004  3.896909   44 -1.386294    1  2.169054        7
94          95  2.907447  3.396185   52 -1.386294    1  2.463853        7
95          96  2.882564  3.773910   68  1.558145    1  1.558145        7
96          97  3.471966  3.974998   68  0.438255    1  2.904165        7

    pgg45      lpsa train
0       0 -0.430783     T
1       0 -0.162519     T
2      20 -0.162519     T
3       0 -0.162519     T
4       0  0.371564     T
..    ...       ...   ...
92     60  4.385147     T
93     40  4.684443     T
94     10  5.143124     F
95     80  5.477509     T
96     20  5.582932     F
```

```
[97 rows x 11 columns]
```

[4]:
```
prostate.drop(prostate.columns[0], axis=1, inplace=True)
prostate
```

[4]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	\
0	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	0	
1	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	0	
2	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	20	
3	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	0	
4	0.751416	3.432373	62	-1.386294	0	-1.386294	6	0	
..	…	…	…	…	…	…	…	…	
92	2.830268	3.876396	68	-1.386294	1	1.321756	7	60	
93	3.821004	3.896909	44	-1.386294	1	2.169054	7	40	
94	2.907447	3.396185	52	-1.386294	1	2.463853	7	10	
95	2.882564	3.773910	68	1.558145	1	1.558145	7	80	
96	3.471966	3.974998	68	0.438255	1	2.904165	7	20	

	lpsa	train
0	-0.430783	T
1	-0.162519	T
2	-0.162519	T
3	-0.162519	T
4	0.371564	T
..	…	…
92	4.385147	T
93	4.684443	T
94	5.143124	F
95	5.477509	T
96	5.582932	F

```
[97 rows x 10 columns]
```

[5]:
```
prostate.train.replace(to_replace=['F', 'T'], value=[0, 1], inplace=True)
```

[6]:
```
prostate
```

[6]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	\
0	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	0	
1	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	0	
2	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	20	
3	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	0	
4	0.751416	3.432373	62	-1.386294	0	-1.386294	6	0	
..	…	…	…	…	…	…	…	…	
92	2.830268	3.876396	68	-1.386294	1	1.321756	7	60	
93	3.821004	3.896909	44	-1.386294	1	2.169054	7	40	
94	2.907447	3.396185	52	-1.386294	1	2.463853	7	10	

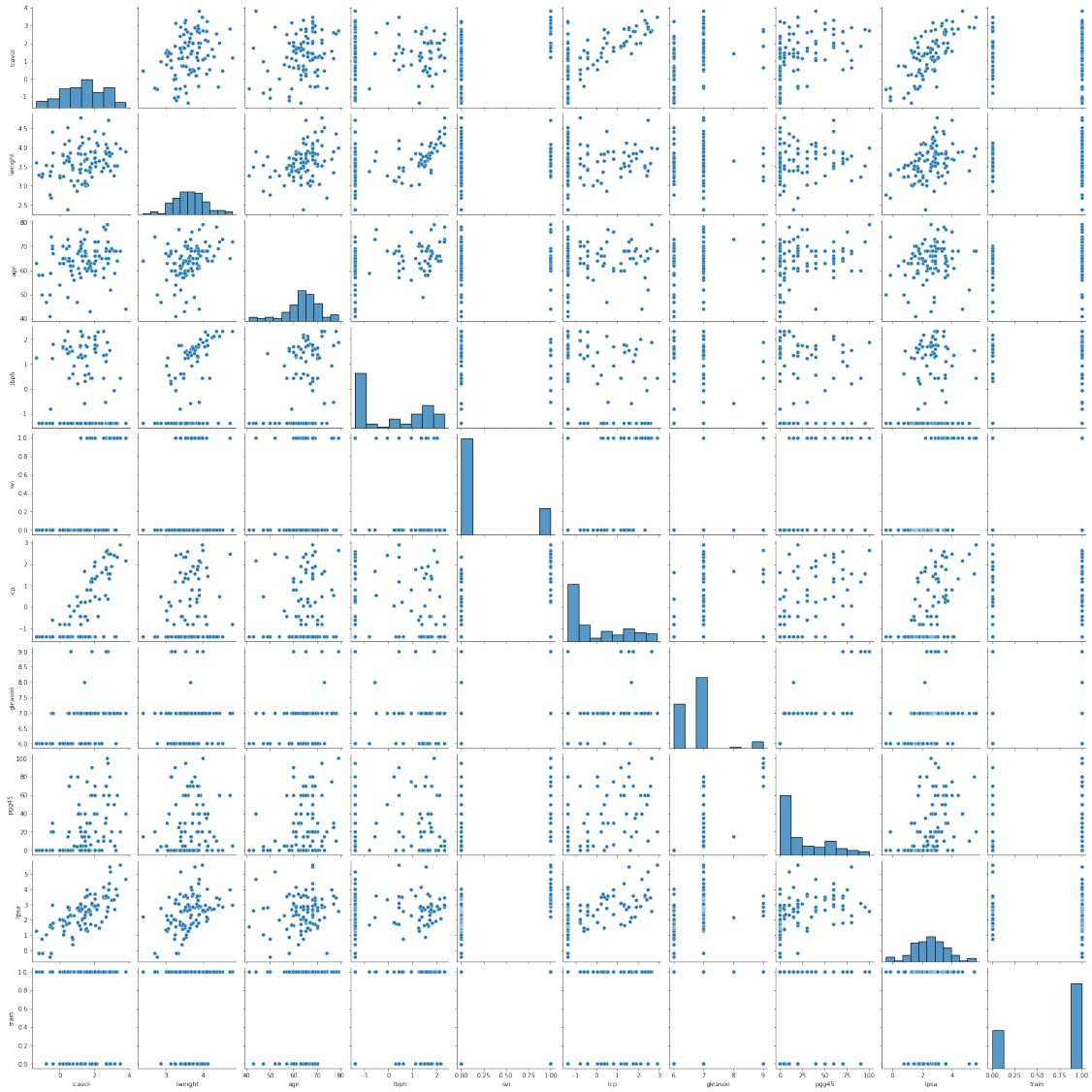
```
95  2.882564  3.773910   68  1.558145    1  1.558145        7     80
96  3.471966  3.974998   68  0.438255    1  2.904165        7     20

        lpsa  train
0  -0.430783      1
1  -0.162519      1
2  -0.162519      1
3  -0.162519      1
4   0.371564      1
..       …    …
92  4.385147      1
93  4.684443      1
94  5.143124      0
95  5.477509      1
96  5.582932      0

[97 rows x 10 columns]
```

[7]:
```
sns.pairplot(prostate)
```

[7]:
```
<seaborn.axisgrid.PairGrid at 0x24fadbf7c88>
```



[8]:
```
prostate.describe()
```

[8]:

	lcavol	lweight	age	lbph	svi	lcp	\
count	97.000000	97.000000	97.000000	97.000000	97.000000	97.000000	
mean	1.350010	3.628943	63.865979	0.100356	0.216495	-0.179366	
std	1.178625	0.428411	7.445117	1.450807	0.413995	1.398250	
min	-1.347074	2.374906	41.000000	-1.386294	0.000000	-1.386294	
25%	0.512824	3.375880	60.000000	-1.386294	0.000000	-1.386294	
50%	1.446919	3.623007	65.000000	0.300105	0.000000	-0.798508	
75%	2.127041	3.876396	68.000000	1.558145	0.000000	1.178655	
max	3.821004	4.780383	79.000000	2.326302	1.000000	2.904165	

	gleason	pgg45	lpsa	train

```
count  97.000000   97.000000  97.000000  97.000000
mean    6.752577   24.381443   2.478387   0.690722
std     0.722134   28.204035   1.154329   0.464597
min     6.000000    0.000000  -0.430783   0.000000
25%     6.000000    0.000000   1.731656   0.000000
50%     7.000000   15.000000   2.591516   1.000000
75%     7.000000   40.000000   3.056357   1.000000
max     9.000000  100.000000   5.582932   1.000000
```

[9]:
```
prostate["train"]
```

[9]:
```
0     1
1     1
2     1
3     1
4     1
     ..
92    1
93    1
94    0
95    1
96    0
Name: train, Length: 97, dtype: int64
```

[10]:
```
prostate.__dict__
```

[10]:
```
{'_is_copy': None,
 '_mgr': BlockManager
 Items: Index(['lcavol', 'lweight', 'age', 'lbph', 'svi', 'lcp', 'gleason',
'pgg45',
       'lpsa', 'train'],
      dtype='object')
 Axis 1: RangeIndex(start=0, stop=97, step=1)
 FloatBlock: [0 1 3 5 8], 5 x 97, dtype: float64
 IntBlock: [2 4 6 7 9], 5 x 97, dtype: int64,
 '_item_cache': {'train': 0     1
  1     1
  2     1
  3     1
  4     1
       ..
  92    1
  93    1
  94    0
  95    1
  96    0
  Name: train, Length: 97, dtype: int64},
```

```
 '_attrs': {}}
```

[11]:
```
prostate.__class__
```

[11]:
```
pandas.core.frame.DataFrame
```

[12]:
```
prostate.loc[:, prostate.columns != "train"]
```

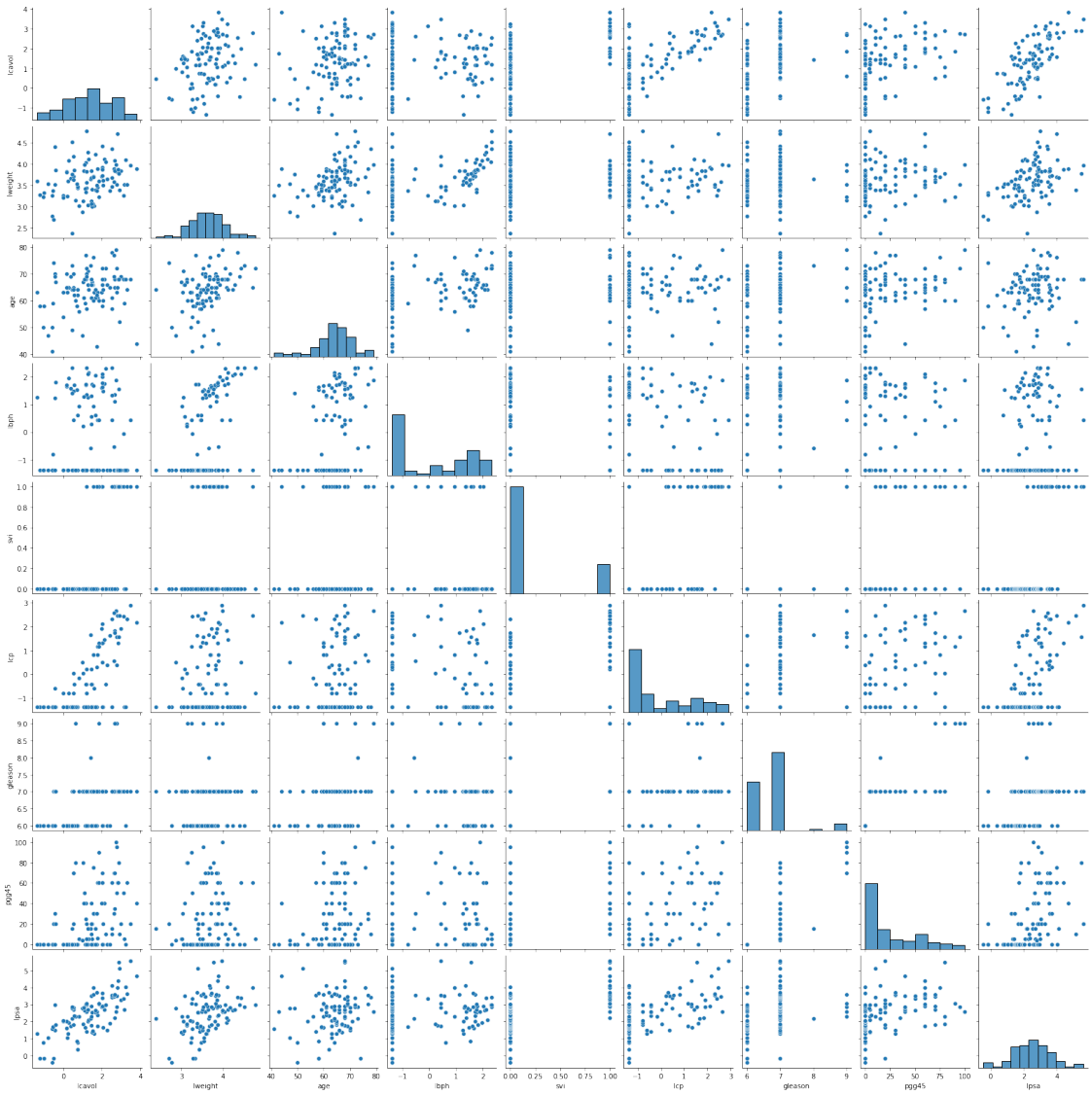
[12]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa
0	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	0	-0.430783
1	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	0	-0.162519
2	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	20	-0.162519
3	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	0	-0.162519
4	0.751416	3.432373	62	-1.386294	0	-1.386294	6	0	0.371564
..	…	…	…	…	…	…	…	…	…
92	2.830268	3.876396	68	-1.386294	1	1.321756	7	60	4.385147
93	3.821004	3.896909	44	-1.386294	1	2.169054	7	40	4.684443
94	2.907447	3.396185	52	-1.386294	1	2.463853	7	10	5.143124
95	2.882564	3.773910	68	1.558145	1	1.558145	7	80	5.477509
96	3.471966	3.974998	68	0.438255	1	2.904165	7	20	5.582932

[97 rows x 9 columns]

[13]:
```
sns.pairplot(prostate.loc[:, prostate.columns != "train"])
```

[13]:
```
<seaborn.axisgrid.PairGrid at 0x24fb3d1be88>
```



[14]:
```
prostate.loc[:, prostate.columns.isin(['lcavol','lweight','age','lbph','lpsa'])]
```

[14]:

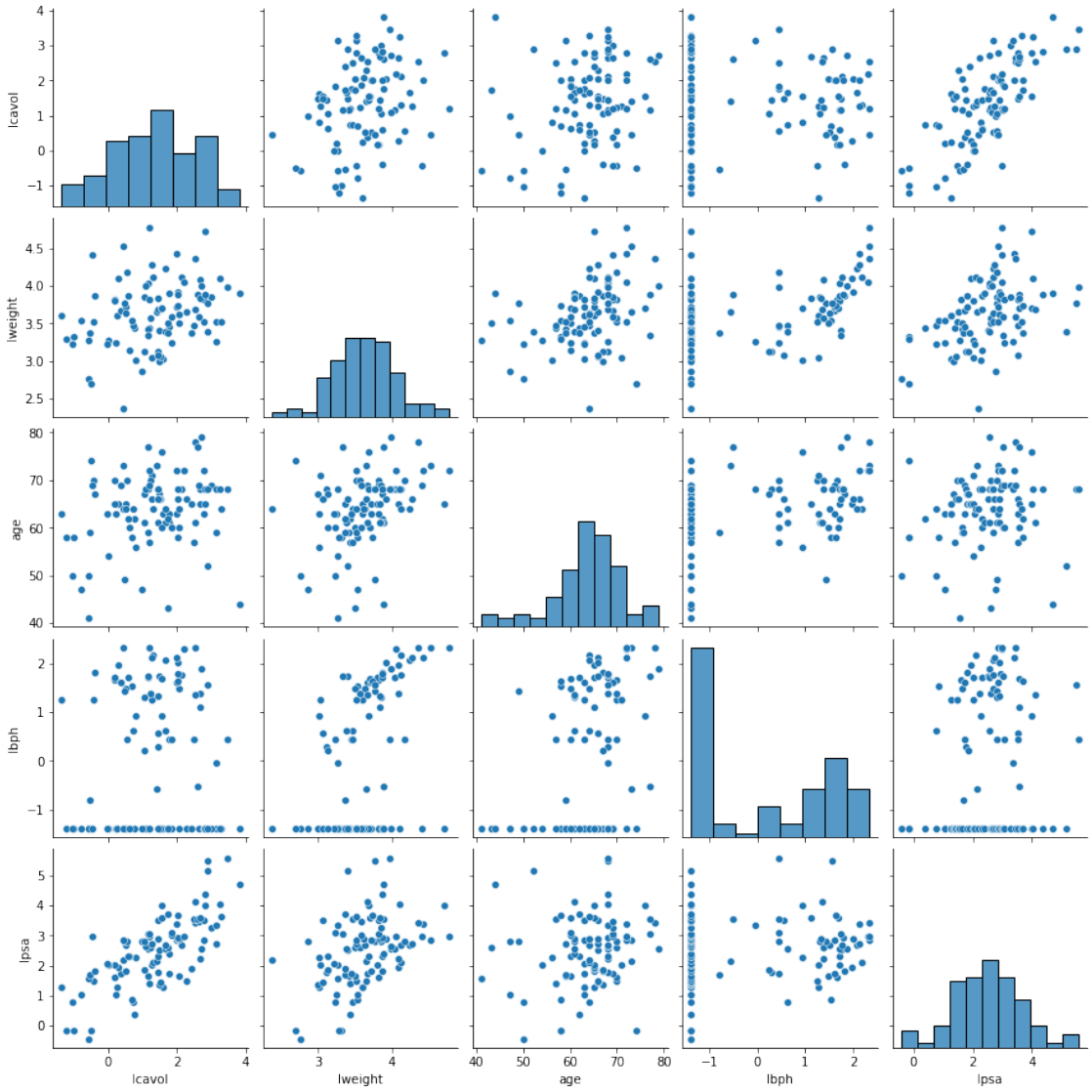
	lcavol	lweight	age	lbph	lpsa
0	-0.579818	2.769459	50	-1.386294	-0.430783
1	-0.994252	3.319626	58	-1.386294	-0.162519
2	-0.510826	2.691243	74	-1.386294	-0.162519
3	-1.203973	3.282789	58	-1.386294	-0.162519
4	0.751416	3.432373	62	-1.386294	0.371564
..	…	…	…	…	…
92	2.830268	3.876396	68	-1.386294	4.385147
93	3.821004	3.896909	44	-1.386294	4.684443
94	2.907447	3.396185	52	-1.386294	5.143124
95	2.882564	3.773910	68	1.558145	5.477509

```
96   3.471966   3.974998    68   0.438255   5.582932

[97 rows x 5 columns]
```

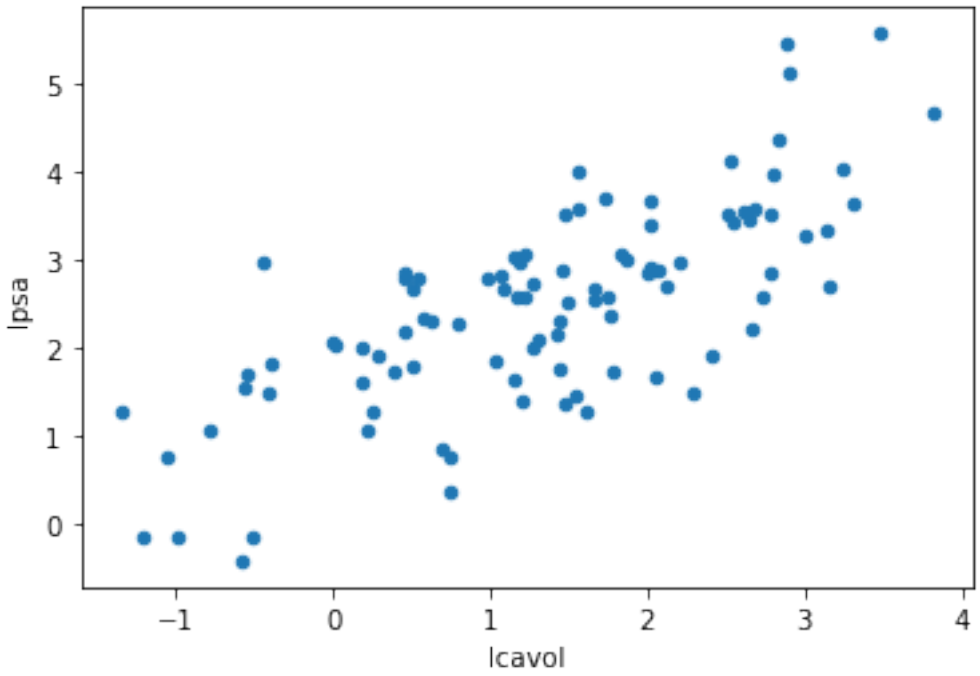
[15]:
```
sns.pairplot(prostate.loc[:, prostate.columns.
 ↪isin(['lcavol','lweight','age','lbph','lpsa'])])
```

[15]:
```
<seaborn.axisgrid.PairGrid at 0x24fb830aa48>
```



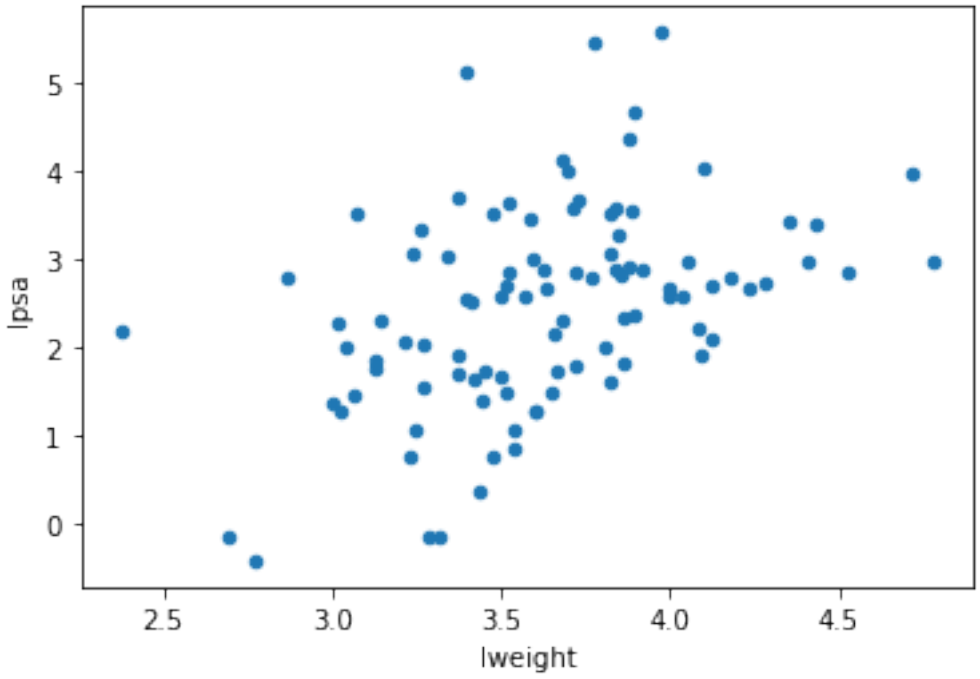
[16]:
```
fig, ax = plt.subplots() # Create the figure and axes object
# Plot the first x and y axes:
prostate.plot.scatter('lcavol', 'lpsa', ax = ax)
```

[16]: <AxesSubplot:xlabel='lcavol', ylabel='lpsa'>



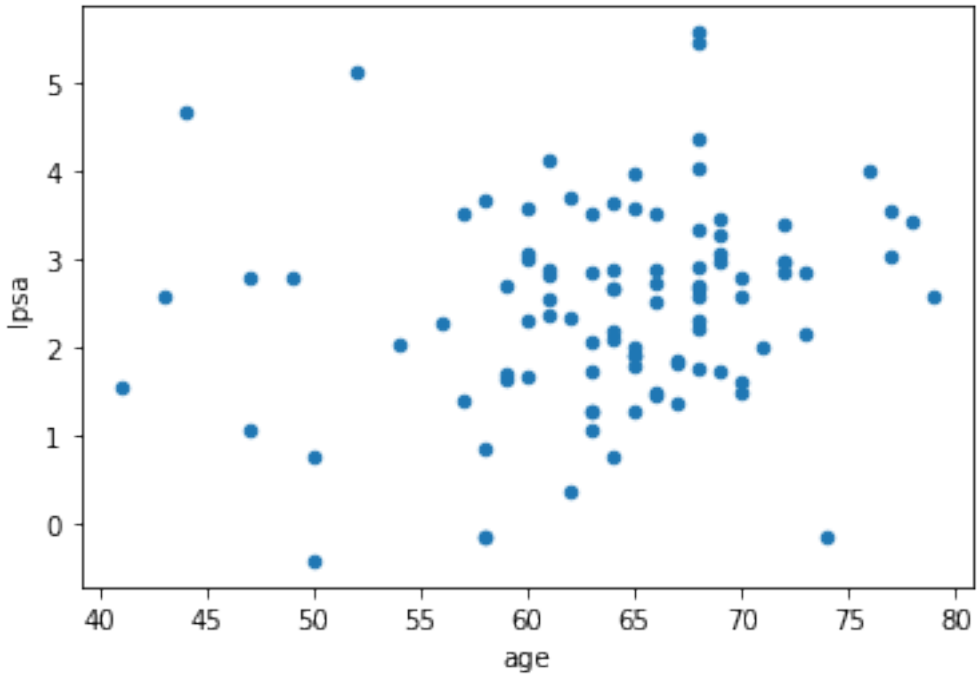
[17]:
```
fig, ax = plt.subplots()
prostate.plot.scatter('lweight', 'lpsa', ax = ax)
```

[17]: <AxesSubplot:xlabel='lweight', ylabel='lpsa'>



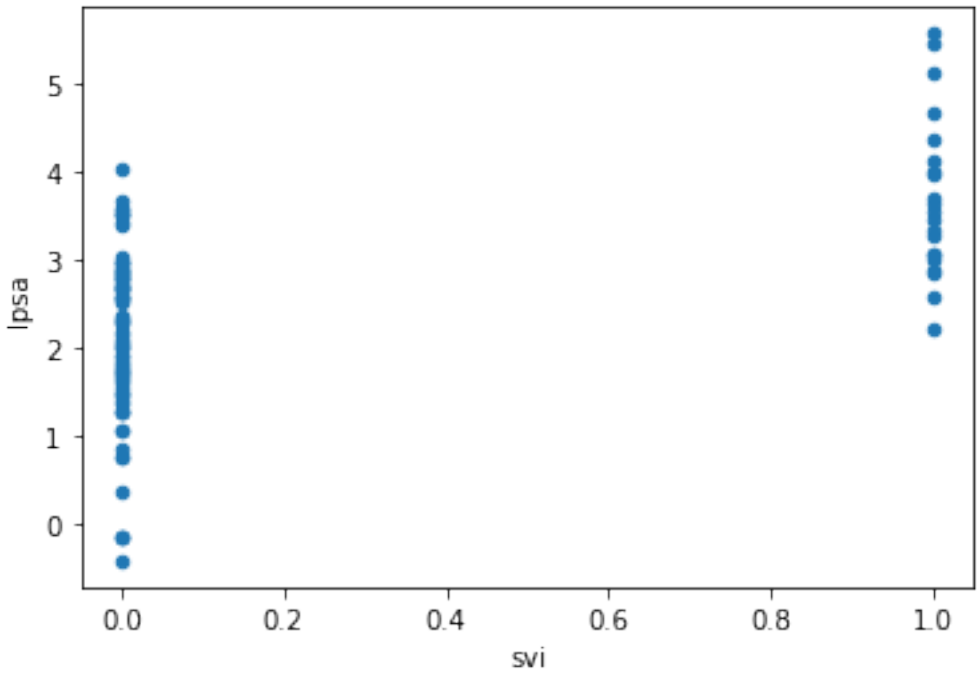
[18]:
```
fig, ax = plt.subplots()
prostate.plot.scatter('age', 'lpsa', ax = ax)
```

[18]:
```
<AxesSubplot:xlabel='age', ylabel='lpsa'>
```



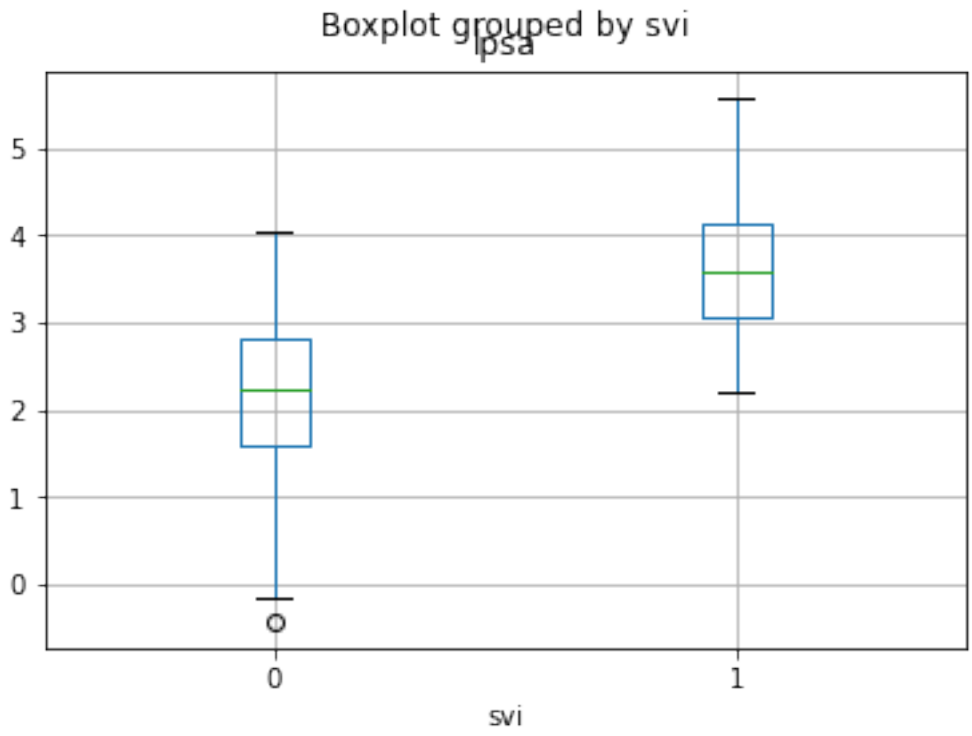
[19]:
```
fig, ax = plt.subplots()
prostate.plot.scatter('svi', 'lpsa', ax = ax)
```

[19]: <AxesSubplot:xlabel='svi', ylabel='lpsa'>



[20]:
```
fig, ax = plt.subplots()
prostate.boxplot('lpsa', 'svi', ax = ax)
```

[20]: <AxesSubplot:title={'center':'lpsa'}, xlabel='svi'>



[21]:
```
from sklearn.model_selection import train_test_split
```

[22]:
```
prostate_train = prostate.loc[prostate['train'] == 1]
prostate_train
```

[22]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa	train
0	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	0	-0.430783	1
1	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	0	-0.162519	1
2	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	20	-0.162519	1
3	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	0	-0.162519	1
4	0.751416	3.432373	62	-1.386294	0	-1.386294	6	0	0.371564	1
..	…	…	…	…	…	…	…	…	…	…
90	3.246491	4.101817	68	-1.386294	0	-1.386294	6	0	4.385147	1
91	2.532903	3.677566	61	1.348073	1	-1.386294	7	15	4.684449	1
92	2.830268	3.876396	68	-1.386294	1	1.321756	7	60	5.143227	1
93	3.821004	3.896909	44	-1.386294	1	2.169054	7	40	5.477509	1
95	2.882564	3.773910	68	1.558145	1	1.558145	7	80	5.582932	1

```
4   0.371564      1
..       …    …
90  4.029806      1
91  4.129551      1
92  4.385147      1
93  4.684443      1
95  5.477509      1

[67 rows x 10 columns]
```

[23]:
```
prostate_test = prostate.loc[prostate['train'] == 0]
prostate_test
```

[23]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa	train
6	0.737164	3.473518	64	0.615186	0	-1.386294	6	0	0.765468	0
8	-0.776529	3.539509	47	-1.386294	0	-1.386294	6	0		
9	0.223144	3.244544	63	-1.386294	0	-1.386294	6	0		
14	1.205971	3.442019	57	-1.386294	0	-0.430783	7	5		
21	2.059239	3.501043	60	1.474763	0	1.348073	7	20		
24	0.385262	3.667400	69	1.599388	0	-1.386294	6	0		
25	1.446919	3.124565	68	0.300105	0	-1.386294	6	0		
27	-0.400478	3.865979	67	1.816452	0	-1.386294	7	20		
31	0.182322	3.804438	65	1.704748	0	-1.386294	6	0		
33	0.009950	3.267666	54	-1.386294	0	-1.386294	6	0		
35	1.308333	4.119850	64	2.171337	0	-1.386294	7	5		
41	1.442202	3.682610	68	-1.386294	0	-1.386294	7	10		
43	1.771557	3.896909	61	-1.386294	0	0.810930	7	6		
47	1.163151	4.035125	68	1.713798	0	-0.430783	7	40		
48	1.745716	3.498022	43	-1.386294	0	-1.386294	6	0		
49	1.220830	3.568123	70	1.373716	0	-0.798508	6	0		
52	0.512824	3.633631	64	1.492904	0	0.048790	7	70		
53	2.127041	4.121473	68	1.766442	0	1.446919	7	40		
54	3.153590	3.516013	59	-1.386294	0	-1.386294	7	5		
56	0.974560	2.865054	47	-1.386294	0	0.500775	7	4		
61	1.997418	3.719651	63	1.619388	1	1.909542	7	40		
63	2.034706	3.917011	66	2.008214	1	2.110213	7	60		
64	2.073172	3.623007	64	-1.386294	0	-1.386294	6	0		
65	1.458615	3.836221	61	1.321756	0	-0.430783	7	20		
72	1.214913	3.825375	69	-1.386294	1	0.223144	7	20		
73	1.838961	3.236716	60	0.438255	1	1.178655	9	90		
79	2.779440	3.823192	63	-1.386294	0	0.371564	7	50		
83	2.677591	3.838376	65	1.115142	0	1.749200	9	70		
94	2.907447	3.396185	52	-1.386294	1	2.463853	7	10		
96	3.471966	3.974998	68	0.438255	1	2.904165	7	20		

```
8   1.047319      0
9   1.047319      0
14  1.398717      0
21  1.658228      0
24  1.731656      0
25  1.766442      0
27  1.816452      0
31  2.008214      0
33  2.021548      0
35  2.085672      0
41  2.307573      0
43  2.374906      0
47  2.568788      0
48  2.591516      0
49  2.591516      0
52  2.684440      0
53  2.691243      0
54  2.704711      0
56  2.788093      0
61  2.853592      0
63  2.882004      0
64  2.882004      0
65  2.887590      0
72  3.056357      0
73  3.075006      0
79  3.513037      0
83  3.570940      0
94  5.143124      0
96  5.582932      0
```

[24]:
```
X_train = prostate_train.filter(['lcavol','lweight','age','lbph'], axis=1)
X_train
```

[24]:
```
       lcavol   lweight  age      lbph
0   -0.579818  2.769459   50 -1.386294
1   -0.994252  3.319626   58 -1.386294
2   -0.510826  2.691243   74 -1.386294
3   -1.203973  3.282789   58 -1.386294
4    0.751416  3.432373   62 -1.386294
..        …       …  …       …
90   3.246491  4.101817   68 -1.386294
91   2.532903  3.677566   61  1.348073
92   2.830268  3.876396   68 -1.386294
93   3.821004  3.896909   44 -1.386294
95   2.882564  3.773910   68  1.558145

[67 rows x 4 columns]
```

[25]:
```
y_train = prostate_train.filter(['lpsa'], axis=1)
y_train
```

[25]:

	lpsa
0	-0.430783
1	-0.162519
2	-0.162519
3	-0.162519
4	0.371564
..	…
90	4.029806
91	4.129551
92	4.385147
93	4.684443
95	5.477509

[67 rows x 1 columns]

[26]:
```
X_test = prostate_test.filter(['lcavol','lweight','age','lbph'], axis=1)
X_test
```

[26]:

	lcavol	lweight	age	lbph
6	0.737164	3.473518	64	0.615186
8	-0.776529	3.539509	47	-1.386294
9	0.223144	3.244544	63	-1.386294
14	1.205971	3.442019	57	-1.386294
21	2.059239	3.501043	60	1.474763
24	0.385262	3.667400	69	1.599388
25	1.446919	3.124565	68	0.300105
27	-0.400478	3.865979	67	1.816452
31	0.182322	3.804438	65	1.704748
33	0.009950	3.267666	54	-1.386294
35	1.308333	4.119850	64	2.171337
41	1.442202	3.682610	68	-1.386294
43	1.771557	3.896909	61	-1.386294
47	1.163151	4.035125	68	1.713798
48	1.745716	3.498022	43	-1.386294
49	1.220830	3.568123	70	1.373716
52	0.512824	3.633631	64	1.492904
53	2.127041	4.121473	68	1.766442
54	3.153590	3.516013	59	-1.386294
56	0.974560	2.865054	47	-1.386294
61	1.997418	3.719651	63	1.619388
63	2.034706	3.917011	66	2.008214
64	2.073172	3.623007	64	-1.386294
65	1.458615	3.836221	61	1.321756
72	1.214913	3.825375	69	-1.386294

```
73  1.838961  3.236716   60  0.438255
79  2.779440  3.823192   63 -1.386294
83  2.677591  3.838376   65  1.115142
94  2.907447  3.396185   52 -1.386294
96  3.471966  3.974998   68  0.438255
```

[27]:
```
y_test = prostate_test.filter(['lpsa'], axis=1)
y_test
```

[27]:
```
        lpsa
6   0.765468
8   1.047319
9   1.047319
14  1.398717
21  1.658228
24  1.731656
25  1.766442
27  1.816452
31  2.008214
33  2.021548
35  2.085672
41  2.307573
43  2.374906
47  2.568788
48  2.591516
49  2.591516
52  2.684440
53  2.691243
54  2.704711
56  2.788093
61  2.853592
63  2.882004
64  2.882004
65  2.887590
72  3.056357
73  3.075006
79  3.513037
83  3.570940
94  5.143124
96  5.582932
```

[28]:
```
from sklearn import preprocessing

X_train_scaled = preprocessing.StandardScaler().fit(X_train).transform(X_train)
X_train_scaled
```

```
[28]: array([[-1.53517959, -1.81097918, -1.98042525, -1.00347165],
             [-1.87122084, -0.64791082, -0.90602451, -1.00347165],
             [-1.47923712, -1.97632998,  1.24277698, -1.00347165],
             [-2.04127153, -0.72578526, -0.90602451, -1.00347165],
             [-0.45575567, -0.40956052, -0.36882414, -1.00347165],
             [-1.91627934, -0.83986455, -1.98042525, -1.00347165],
             [-0.50300267, -0.18307203, -0.90602451,  1.00876737],
             [-0.85856191, -0.04644453,  0.03407614, -1.00347165],
             [-2.15730403, -0.05798078, -0.23452404,  0.82296074],
             [ 0.24320332, -1.27528029, -0.23452404, -1.00347165],
             [ 0.13261939, -1.32735303,  0.30267633, -1.00347165],
             [ 0.1846029 , -1.19454345,  0.16837624, -1.00347165],
             [-1.4019554 , -0.23274323,  0.70557661,  0.80727054],
             [ 0.79056879,  0.04915392,  0.16837624, -1.00347165],
             [-1.52082799, -0.75775571, -3.18912609, -1.00347165],
             [-0.9172027 ,  0.42125665,  0.70557661,  1.09230947],
             [-0.13467239, -0.4370598 , -0.77172442, -1.00347165],
             [-1.506726  , -0.5289883 , -0.77172442, -0.59885247],
             [ 0.37965154, -0.36896907, -0.23452404,  0.25250725],
             [-0.649217  ,  0.19775317,  0.03407614, -1.00347165],
             [-0.22153469, -1.05100308,  0.30267633,  0.10442939],
             [ 0.88880906, -0.5289883 ,  0.03407614, -1.00347165],
             [-0.83380138,  0.98103854,  0.03407614,  1.30204412],
             [-0.03091648, -1.24464169,  0.8398767 ,  0.82296074],
             [-1.07318635, -0.86513141, -0.23452404, -1.00347165],
             [ 0.08888214,  0.06558415,  1.10847689, -0.44831171],
             [-0.69413678, -2.6450752 , -0.10022395, -1.00347165],
             [ 1.09258592,  0.97039864,  0.43697642,  0.89645743],
             [-0.41838296, -1.29595548, -1.17462469,  0.59520809],
             [-0.56184621, -1.0234277 , -0.63742432, -1.00347165],
             [-0.59295089,  0.50709464, -0.36882414,  1.13056251],
             [ 0.13999076, -0.45792314,  0.16837624,  1.1549324 ],
             [ 0.28414784, -0.49315764, -0.50312423,  0.37430239],
             [ 1.14682673,  0.78078936,  1.91427745,  1.24460405],
             [-0.17965735,  0.77689532,  0.43697642, -1.00347165],
             [ 0.28107063,  1.28685803, -0.10022395,  1.37794753],
             [-0.03773989,  1.38262561,  0.16837624,  1.41173972],
             [-0.68902102,  0.29294996, -2.11472535,  0.93045829],
             [-0.62529655,  1.16719349,  0.70557661,  0.25250725],
             [-0.20452334,  0.47587467, -0.50312423,  0.84208358],
             [-0.69413678,  1.89923059,  1.10847689,  1.55219634],
             [ 1.18563051, -0.21397912,  0.97417679, -1.00347165],
             [ 0.57519641,  0.5334925 ,  0.43697642,  1.17846897],
             [ 0.71747029,  0.89805449,  0.97417679,  1.53930373],
             [-1.42690643,  1.65409849,  0.57127652, -1.00347165],
             [-0.09695191,  2.44017015,  0.97417679,  1.55219634],
             [ 0.44644159, -0.06958045, -0.63742432, -1.00347165],
```

```
       [-0.12444078, -0.60252898,  1.64567726,  1.1549324 ],
       [ 1.36686824,  0.47137602,  0.57127652, -1.00347165],
       [ 1.48193054, -0.76582495,  0.43697642, -0.08448679],
       [ 0.56548558,  1.70746079,  0.97417679,  1.41173972],
       [ 0.9926077 ,  1.54044202,  1.77997735,  1.55219634],
       [ 1.08232193, -0.09297217,  0.57127652, -1.00347165],
       [ 0.1251804 , -1.17483226,  0.16837624,  0.33604935],
       [ 0.97314651, -0.32257885, -1.0403246 ,  0.25250725],
       [ 1.05370436,  0.55524162,  1.64567726, -0.41238821],
       [ 0.20178245,  0.17715408, -0.63742432,  1.11804621],
       [ 1.61305928, -0.22647091, -0.10022395, -1.00347165],
       [ 0.57626825,  0.22322298, -0.90602451,  1.07907102],
       [ 0.33906587, -0.54349476, -0.36882414, -1.00347165],
       [ 1.21148426,  2.30840068,  0.03407614, -1.00347165],
       [ 0.20178245,  0.14587281,  1.51137717,  0.59520809],
       [ 1.56736151,  1.00566273,  0.43697642, -1.00347165],
       [ 0.98875265,  0.10878431, -0.50312423,  0.87880576],
       [ 1.22986935,  0.52911646,  0.43697642, -1.00347165],
       [ 2.03320176,  0.57248152, -2.78622581, -1.00347165],
       [ 1.27227305,  0.31245821,  0.43697642,  1.02341425]])
```

[29]:
```
X_test_scaled = preprocessing.StandardScaler().fit(X_test).transform(X_test)
X_test_scaled
```

[29]:
```
array([[-0.68130966, -0.54673695,  0.3032458 ,  0.31707694],
       [-2.16646222, -0.32368683, -2.15160118, -1.09240853],
       [-1.18563845, -1.32067096,  0.15884304, -1.09240853],
       [-0.22134215, -0.65320383, -0.70757354, -1.09240853],
       [ 0.61583772, -0.45370223, -0.27436525,  0.92240989],
       [-1.02657631,  0.10858581,  1.02525962,  1.01017321],
       [ 0.01506301, -1.72620097,  0.88085686,  0.09519005],
       [-1.79750135,  0.77978447,  0.7364541 ,  1.16303472],
       [-1.22569076,  0.57177538,  0.44764857,  1.08437036],
       [-1.39481197, -1.24251841, -1.14078183, -1.09240853],
       [-0.12091015,  1.63787054,  0.3032458 ,  1.41295222],
       [ 0.01043495,  0.15999573,  0.88085686, -1.09240853],
       [ 0.33357982,  0.88432812, -0.12996249, -1.09240853],
       [-0.26335479,  1.35149934,  0.88085686,  1.09074345],
       [ 0.30822582, -0.46391323, -2.72921224, -1.09240853],
       [-0.2067632 , -0.22697127,  1.16966239,  0.85125011],
       [-0.90142022, -0.00555369,  0.3032458 ,  0.93518524],
       [ 0.68236103,  1.64335629,  0.88085686,  1.1278163 ],
       [ 1.68955553, -0.4031035 , -0.41876802, -1.09240853],
       [-0.44839012, -2.60335029, -2.15160118, -1.09240853],
       [ 0.55518222,  0.28519462,  0.15884304,  1.02425811],
       [ 0.5917671 ,  0.95227305,  0.59205133,  1.29807763],
       [ 0.62950811, -0.0414629 ,  0.3032458 , -1.09240853],
```

```
       [ 0.02653852,  0.67920218, -0.12996249,  0.81465894],
       [-0.21256881,  0.64254261,  1.02525962, -1.09240853],
       [ 0.39971324, -1.34712967, -0.27436525,  0.19247851],
       [ 1.32245987,  0.63516405,  0.15884304, -1.09240853],
       [ 1.22253101,  0.6864861 ,  0.44764857,  0.66915672],
       [ 1.44805349, -0.80812313, -1.42958736, -1.09240853],
       [ 2.00192872,  1.14826959,  0.88085686,  0.19247851]])
```

[30]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[31]:
```
reg = KNeighborsRegressor(n_neighbors=5)
```

[32]:
```
reg.fit(X_train_scaled, y_train)
```

[32]:
```
KNeighborsRegressor()
```

[33]:
```
print(reg.score(X_test_scaled, y_test))
```

```
0.3548149193676786
```

[34]:
```
y_pred = reg.predict(X_test_scaled)
y_pred
```

[34]:
```
array([[2.21403244],
       [0.73883804],
       [1.16425154],
       [2.20671312],
       [2.901218  ],
       [2.4049538 ],
       [2.0335348 ],
       [1.81342916],
       [2.02549518],
       [0.6474735 ],
       [2.7109335 ],
       [2.31303426],
       [2.6695894 ],
       [2.73669252],
       [2.88508564],
       [2.63205582],
       [1.96856208],
       [2.94457494],
       [2.69789602],
       [1.42023566],
       [3.72738542],
       [2.83346724],
       [2.98858246],
       [2.86508512],
       [1.87484524],
```

```
       [2.17971144],
       [3.32820116],
       [3.54079518],
       [3.15635954],
       [3.7123247 ]])
```

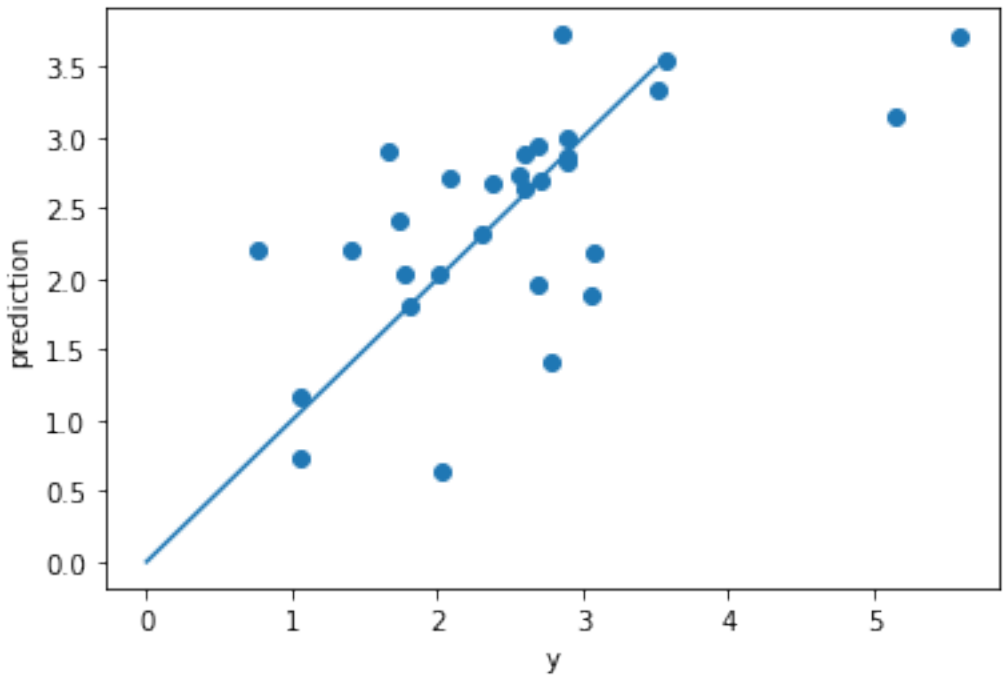
[35]:
```
from sklearn.metrics import mean_squared_error
```

[36]:
```
mean_squared_error(y_test, y_pred)
```

[36]:
```
0.6772141876165387
```

[37]:
```
fig, ax = plt.subplots()
plt.xlabel('y')
plt.ylabel('prediction')
plt.plot([0, 3.5], [0, 3.5])
plt.scatter(y_test, y_pred)
```

[37]:
```
<matplotlib.collections.PathCollection at 0x24fbba8d208>
```



[38]:
```
# Question 4
```

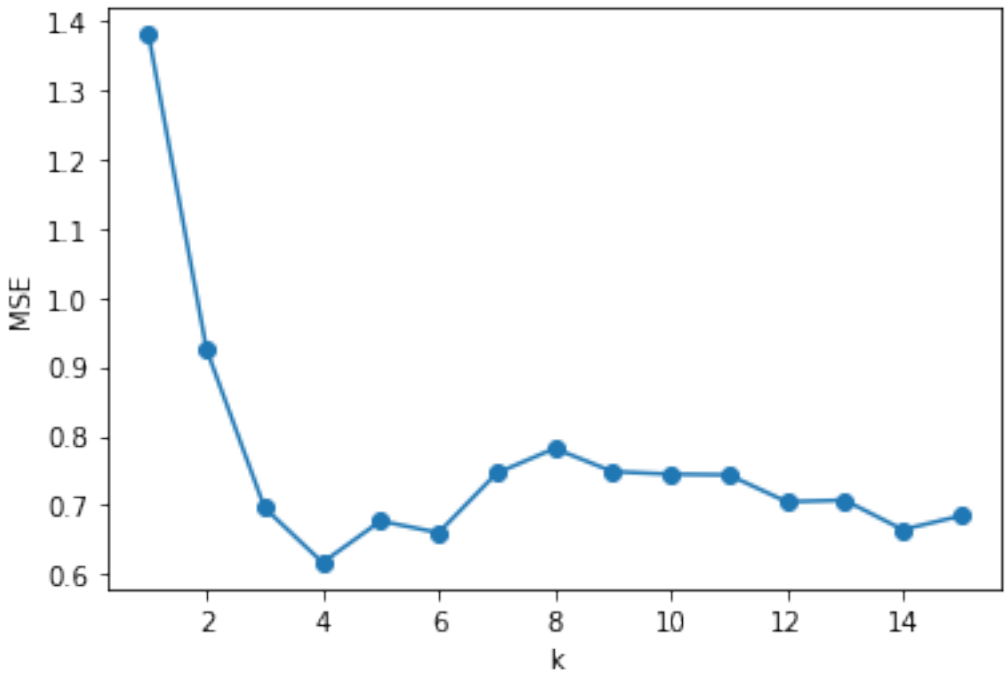
[39]:
```
MSE = [0] * 15
```

[40]:
```
for k in range(1, 16):
    reg = KNeighborsRegressor(n_neighbors=k)
    reg.fit(X_train_scaled, y_train)
    y_pred = reg.predict(X_test_scaled)
    MSE[k-1] = mean_squared_error(y_test, y_pred)
MSE
```

[40]:
```
[1.3795858731891253,
 0.9248916376349593,
 0.6962854899503435,
 0.6172386488220023,
 0.6772141876165387,
 0.6602449006455746,
 0.7461934800608266,
 0.7819454155340702,
 0.7485111941838746,
 0.7447805977158086,
 0.7438203874114898,
 0.7049888350540215,
 0.7072512454217635,
 0.6648079566911367,
 0.6845543932253605]
```

[41]:
```
fig, ax = plt.subplots()
plt.xlabel('k')
plt.ylabel('MSE')
plt.plot(range(1, 16), MSE)
plt.scatter(range(1, 16), MSE)
```

[41]:
```
<matplotlib.collections.PathCollection at 0x24fbbb09248>
```



[42]:
```
# Part II
```

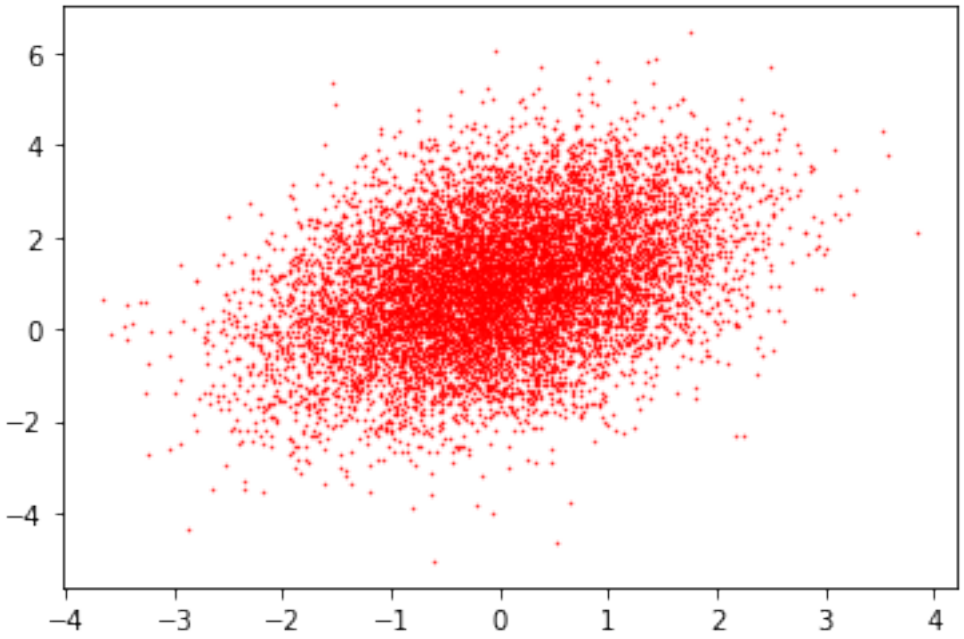
[43]:
```
# Q1
```

[ ]:
```
```

[44]:
```
mu1 = [0, 1]
mu2 = [0, 2]
mu3 = [2, 0]
Sigma1 = [[1, 0.5], [0.5, 2]]
Sigma2 = [[2, -0.5], [-0.5, 1]]
Sigma3 = [[1, 0], [0, 1]]
```

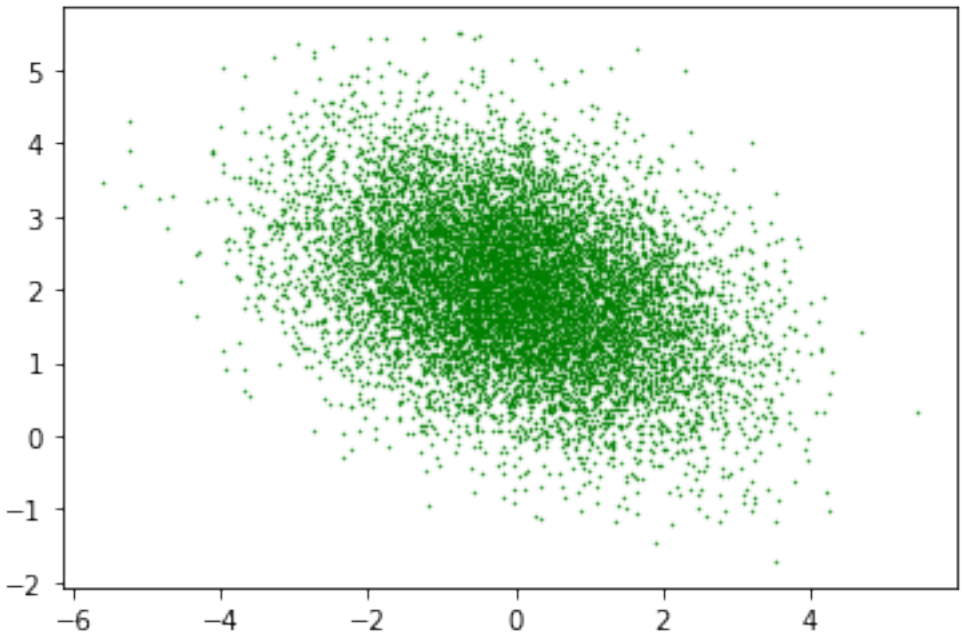
[45]:
```
xy = np.random.multivariate_normal(mu1, Sigma1, 10000)
plt.scatter(xy[:, 0], xy[:, 1], c='r', s=0.2)
```

[45]:
```
<matplotlib.collections.PathCollection at 0x24fbbb09188>
```



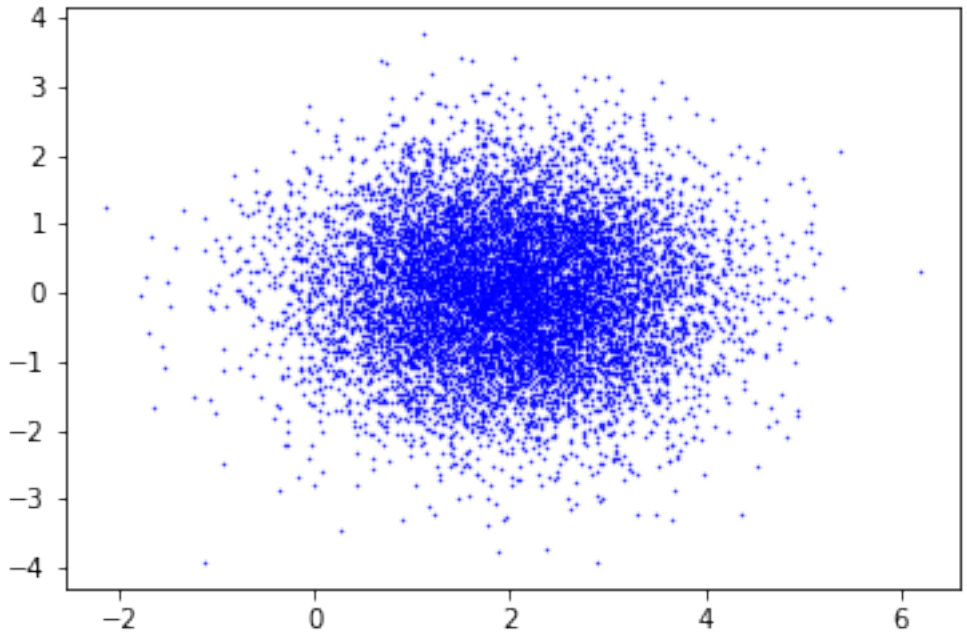
[46]:
```
xy = np.random.multivariate_normal(mu2, Sigma2, 10000)
plt.scatter(xy[:, 0], xy[:, 1], c='g', s=0.2)
```

[46]: <matplotlib.collections.PathCollection at 0x24fbbb7ea48>



[47]:
```
xy = np.random.multivariate_normal(mu3, Sigma3, 10000)
plt.scatter(xy[:, 0], xy[:, 1], c='b', s=0.2)
```

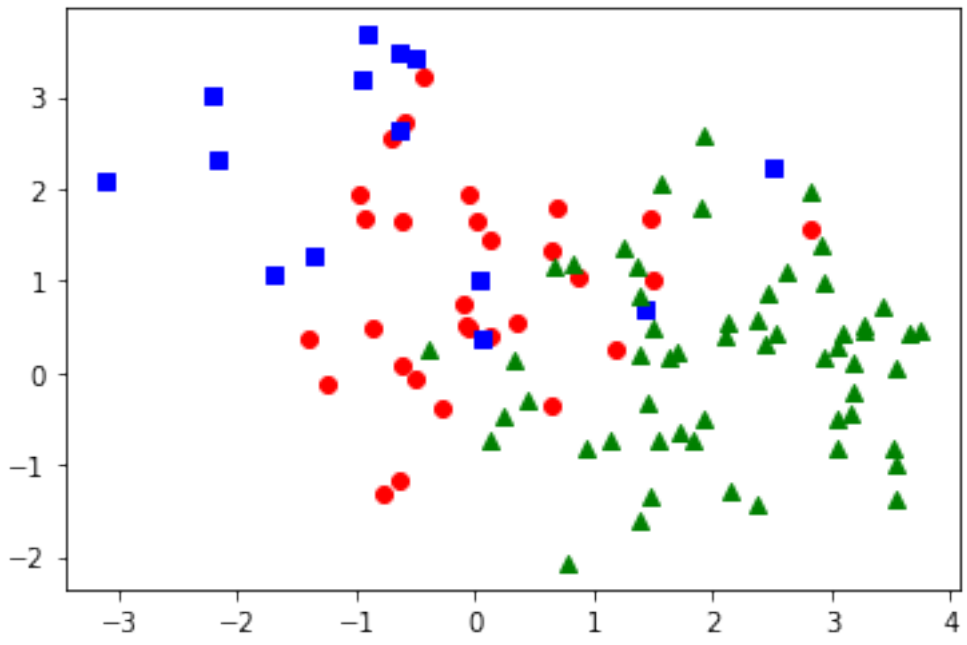
[47]: <matplotlib.collections.PathCollection at 0x24fbbbeda08>



[48]:
```
def gen_data(N, mu1, mu2, mu3, Sigma1, Sigma2, Sigma3, p1, p2):
    y = np.random.choice([1, 2, 3], N, p=[p1, p2, 1 - p1 - p2])
    X = np.zeros((N,2))
    N1 = np.count_nonzero(y == 1)
    N2 = np.count_nonzero(y == 2)
    N3 = np.count_nonzero(y == 3)
    X[y==1, ] = np.random.multivariate_normal(mu1, Sigma1, N1)
    X[y==2, ] = np.random.multivariate_normal(mu2, Sigma2, N2)
    X[y==3, ] = np.random.multivariate_normal(mu3, Sigma3, N3)
    return dict({'X':X, 'y':y})
```

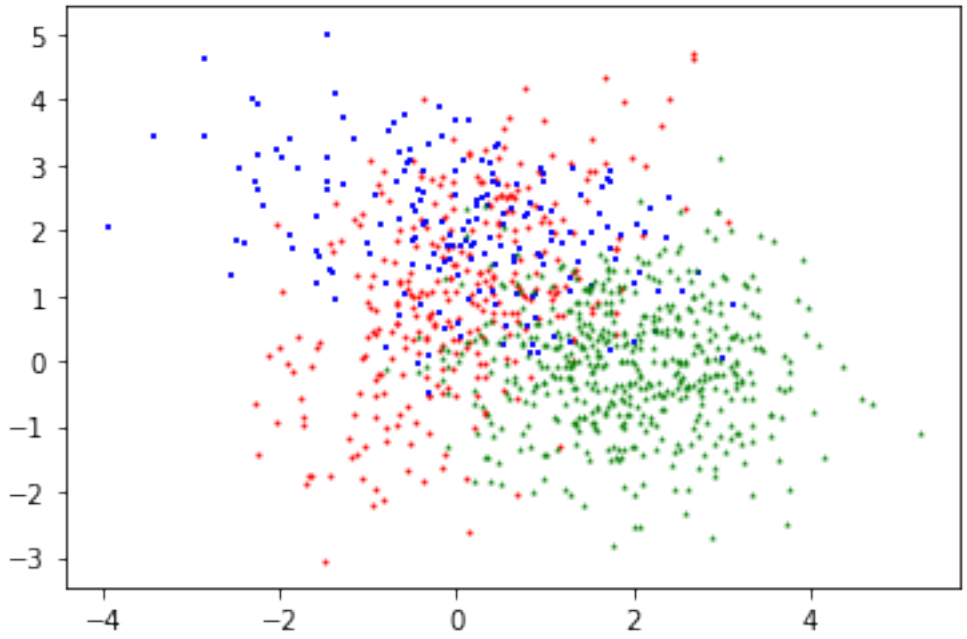
[49]:
```
train = gen_data(100,mu1,mu2,mu3,Sigma1,Sigma2,Sigma3,p1=0.3,p2=0.2)
plt.scatter(train['X'][train['y']==1, 0], train['X'][train['y']==1, 1], c='r',␣
 ↪marker='o')
plt.scatter(train['X'][train['y']==2, 0], train['X'][train['y']==2, 1], c='b',␣
 ↪marker='s')
plt.scatter(train['X'][train['y']==3, 0], train['X'][train['y']==3, 1], c='g',␣
 ↪marker='^')
```

[49]: <matplotlib.collections.PathCollection at 0x24fbbcab588>



[50]:
```
test = gen_data(1000,mu1,mu2,mu3,Sigma1,Sigma2,Sigma3,p1=0.3,p2=0.2)
plt.scatter(test['X'][test['y']==1, 0], test['X'][test['y']==1, 1], c='r',␣
 ↪marker='o', s=1)
plt.scatter(test['X'][test['y']==2, 0], test['X'][test['y']==2, 1], c='b',␣
 ↪marker='s', s=1)
plt.scatter(test['X'][test['y']==3, 0], test['X'][test['y']==3, 1], c='g',␣
 ↪marker='^', s=1)
```

[50]: <matplotlib.collections.PathCollection at 0x24fbbd57348>



[51]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[52]:
```
knn_classifier = KNeighborsClassifier(n_neighbors=5)
```

[53]:
```
knn_classifier.fit(train['X'], train['y'])
```

[53]:
```
KNeighborsClassifier()
```

[54]:
```
print(knn_classifier.score(test['X'], test['y']))
```

```
0.731
```

[55]:
```
y_pred = knn_classifier.predict(test['X'])
y_pred
```

[55]:
```
array([3, 2, 3, 1, 1, 3, 1, 1, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3,
       1, 1, 1, 3, 3, 3, 3, 3, 1, 1, 3, 2, 3, 3, 3, 3, 3, 3, 3, 3, 1,
       3, 1, 2, 3, 3, 3, 1, 3, 1, 3, 1, 3, 3, 1, 2, 1, 1, 3, 1, 1, 1, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 3, 1, 3, 3, 3, 1, 1, 3, 1, 3, 1, 1, 3,
       1, 1, 1, 3, 3, 3, 2, 3, 3, 1, 1, 1, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 1, 1, 3, 3, 1, 3, 1, 1, 3, 3, 3, 1, 3, 1, 3, 3, 3, 3, 3,
       3, 1, 1, 3, 1, 3, 2, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3,
       3, 3, 3, 3, 1, 1, 1, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 3,
       3, 2, 3, 3, 1, 2, 1, 3, 1, 3, 2, 3, 3, 1, 1, 1, 1, 1, 3, 3, 3, 3,
       1, 3, 1, 1, 3, 1, 3, 3, 3, 3, 3, 3, 1, 1, 2, 1, 3, 3, 3, 2, 3, 1,
       1, 2, 1, 3, 3, 3, 3, 3, 3, 3, 2, 1, 3, 3, 3, 3, 1, 3, 1, 1, 1, 1,
```

```
3, 3, 1, 1, 3, 3, 1, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 1, 3, 3, 1, 2,
3, 3, 3, 2, 1, 3, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 1, 3, 2, 1, 3, 1,
3, 3, 3, 1, 3, 3, 3, 3, 1, 1, 3, 1, 1, 3, 2, 3, 1, 3, 1, 3, 3, 1,
2, 1, 1, 3, 3, 2, 3, 3, 1, 1, 1, 3, 3, 3, 1, 1, 1, 3, 3, 3, 3, 3,
3, 1, 1, 3, 1, 1, 3, 1, 3, 3, 1, 3, 1, 3, 3, 1, 3, 3, 1, 1, 1, 1,
3, 3, 3, 3, 1, 3, 1, 1, 3, 3, 2, 1, 1, 3, 1, 3, 3, 1, 3, 1, 1, 3,
1, 3, 1, 3, 1, 3, 1, 3, 1, 1, 3, 3, 3, 1, 1, 3, 3, 3, 1, 1, 3, 1,
3, 3, 3, 3, 1, 3, 1, 3, 3, 1, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 1, 1,
1, 3, 3, 1, 1, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 2, 1, 3, 3, 1, 3, 2,
1, 1, 3, 1, 3, 1, 3, 3, 3, 2, 3, 2, 2, 3, 1, 1, 3, 3, 2, 1, 1, 1,
3, 3, 1, 3, 3, 1, 1, 3, 1, 3, 3, 3, 1, 3, 1, 1, 3, 3, 3, 3, 3, 1,
1, 1, 3, 3, 3, 1, 3, 3, 1, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 1, 3, 1,
1, 1, 3, 2, 3, 3, 2, 1, 1, 3, 3, 3, 1, 3, 2, 1, 2, 2, 1, 1, 3, 1,
1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 1, 1,
3, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 1, 3, 2, 3, 1, 1, 3, 3, 3, 1, 1,
2, 1, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 1, 3,
1, 1, 3, 3, 3, 1, 3, 1, 3, 3, 1, 3, 1, 3, 3, 3, 1, 1, 1, 3, 3, 3,
1, 3, 3, 3, 1, 3, 3, 3, 1, 3, 2, 1, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3,
3, 3, 3, 1, 3, 3, 3, 1, 3, 2, 1, 3, 3, 3, 3, 1, 1, 1, 3, 3, 3, 3,
1, 3, 1, 1, 2, 3, 1, 3, 3, 1, 3, 3, 3, 3, 1, 1, 3, 3, 1, 3, 1, 3,
3, 1, 3, 3, 3, 3, 1, 1, 3, 3, 3, 1, 1, 1, 3, 1, 1, 3, 2, 3, 3, 1,
3, 2, 1, 3, 3, 3, 1, 1, 3, 3, 2, 1, 3, 1, 3, 1, 3, 3, 1, 1, 3, 3,
1, 3, 3, 1, 3, 1, 1, 3, 3, 3, 1, 2, 3, 3, 3, 2, 1, 3, 3, 1, 1, 3,
3, 1, 3, 1, 3, 3, 2, 3, 3, 1, 1, 3, 1, 1, 2, 1, 3, 3, 3, 3, 3, 1,
3, 3, 3, 3, 3, 3, 3, 1, 1, 3, 3, 1, 3, 1, 1, 3, 2, 3, 1, 1, 1, 1,
2, 3, 3, 3, 2, 3, 3, 3, 2, 1, 3, 3, 1, 3, 3, 3, 3, 1, 2, 3, 3, 1,
3, 1, 1, 3, 3, 3, 1, 1, 3, 1, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 2,
1, 3, 3, 3, 1, 2, 2, 3, 2, 3, 3, 3, 3, 3, 1, 3, 3, 1, 1, 1, 2, 3,
3, 1, 3, 3, 1, 1, 1, 1, 3, 1, 3, 3, 1, 1, 3, 3, 1, 3, 1, 1, 3, 3,
1, 1, 1, 3, 3, 1, 1, 3, 3, 3, 3, 2, 3, 3, 2, 1, 1, 3, 1, 1, 1, 3,
1, 3, 3, 1, 1, 1, 3, 2, 1, 3, 1, 1, 3, 1, 3, 3, 1, 1, 2, 3, 3, 1,
3, 3, 1, 3, 1, 1, 3, 2, 3, 1, 2, 1, 1, 3, 3, 1, 1, 1, 3, 3, 1, 3,
3, 1, 3, 3, 1, 3, 1, 1, 1, 3, 3, 3, 3, 3, 1, 3, 1, 1, 1, 1, 3, 1,
1, 3, 3, 3, 3, 1, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 3, 3, 1, 1, 3, 3,
1, 1, 1, 3, 1, 3, 2, 3, 3, 3])
```

[56]:
```
mean_squared_error(test['y'], y_pred)
```

[56]:
```
0.608
```

[57]:
```
test['y'] == y_pred
```

[57]:
```
array([False, False,  True,  True, False,  True, False,  True,  True,
       False,  True,  True,  True,  True,  True,  True, False,  True,
       False,  True,  True,  True, False,  True, False, False,  True,
        True,  True,  True, False, False, False, False, False,  True,
        True, False,  True,  True, False, False,  True, False,  True,
```

```
 True,  True,  True,  True,  True,  True,  True, False,  True,
 True,  True,  True, False,  True,  True, False,  True, False,
False,  True,  True,  True,  True,  True, False,  True, False,
False,  True,  True, False, False,  True,  True,  True, False,
 True,  True,  True,  True, False,  True,  True,  True,  True,
False,  True, False, False, False, False,  True,  True,  True,
False,  True,  True,  True,  True,  True,  True,  True,  True,
 True, False, False,  True,  True,  True,  True,  True,  True,
False,  True,  True, False, False,  True, False, False,  True,
 True, False,  True, False, False,  True, False, False,  True,
 True, False, False,  True,  True,  True,  True,  True, False,
 True, False,  True, False,  True, False,  True,  True,  True,
False, False,  True, False,  True, False,  True,  True, False,
False, False, False,  True,  True,  True,  True,  True,  True,
 True, False,  True,  True,  True,  True,  True,  True, False,
False,  True, False,  True, False,  True, False,  True,  True,
False,  True, False, False,  True,  True,  True,  True,  True,
False,  True, False, False,  True,  True,  True,  True,  True,
 True,  True,  True,  True,  True, False,  True,  True, False,
 True,  True,  True,  True, False,  True,  True, False,  True,
 True,  True,  True,  True,  True,  True,  True, False,  True,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
False,  True, False,  True,  True,  True,  True,  True,  True,
False,  True,  True,  True,  True, False,  True,  True,  True,
 True, False,  True,  True,  True, False,  True,  True,  True,
False, False,  True,  True,  True,  True, False,  True,  True,
False,  True,  True, False,  True,  True,  True,  True,  True,
 True, False, False, False,  True,  True,  True,  True, False,
False,  True,  True,  True,  True, False, False, False,  True,
 True, False,  True,  True,  True,  True,  True,  True,  True,
 True,  True, False,  True,  True, False, False,  True, False,
 True,  True,  True,  True,  True,  True,  True,  True, False,
 True, False, False,  True,  True, False,  True,  True,  True,
 True, False,  True,  True,  True,  True,  True, False,  True,
 True,  True,  True,  True,  True, False,  True,  True,  True,
False,  True,  True,  True,  True,  True, False, False, False,
 True,  True, False,  True,  True, False,  True, False,  True,
False,  True,  True,  True,  True,  True,  True,  True,  True,
 True,  True,  True, False,  True,  True,  True,  True,  True,
False,  True,  True,  True, False,  True, False,  True,  True,
 True,  True, False, False, False, False,  True,  True,  True,
False,  True,  True,  True, False,  True,  True, False,  True,
 True,  True,  True,  True, False,  True,  True,  True, False,
False,  True, False,  True,  True, False,  True, False,  True,
 True,  True,  True,  True,  True, False, False, False,  True,
 True, False,  True,  True,  True,  True,  True, False,  True,
 True, False,  True,  True,  True,  True, False,  True,  True,
```

```
 True, False,  True,  True, False,  True,  True,  True,  True,
 True, False,  True,  True,  True,  True,  True,  True,  True,
False,  True,  True, False, False,  True,  True,  True,  True,
 True,  True,  True,  True,  True,  True, False, False,  True,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
 True, False, False,  True, False,  True,  True, False,  True,
 True,  True,  True,  True,  True, False,  True,  True,  True,
False,  True,  True,  True,  True,  True, False,  True,  True,
 True, False, False,  True,  True,  True,  True,  True, False,
False, False,  True,  True,  True,  True,  True,  True,  True,
False,  True,  True, False,  True,  True,  True, False,  True,
 True,  True,  True,  True, False,  True, False,  True, False,
 True,  True,  True,  True,  True,  True,  True,  True, False,
False,  True, False,  True,  True, False,  True,  True,  True,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
 True,  True,  True, False,  True,  True, False, False,  True,
 True,  True,  True, False,  True, False,  True,  True,  True,
 True, False, False,  True,  True, False,  True,  True,  True,
 True, False, False,  True,  True,  True,  True,  True,  True,
 True,  True, False,  True,  True,  True,  True,  True,  True,
 True,  True,  True,  True, False,  True,  True,  True,  True,
 True,  True,  True, False,  True,  True, False,  True,  True,
 True, False,  True, False, False, False, False,  True,  True,
False,  True,  True, False,  True, False,  True,  True,  True,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
False, False,  True,  True, False,  True,  True, False, False,
 True,  True,  True, False, False, False,  True,  True,  True,
 True,  True, False,  True, False, False, False,  True,  True,
 True,  True,  True, False,  True,  True, False,  True,  True,
 True,  True,  True, False,  True, False,  True, False,  True,
 True,  True,  True,  True,  True,  True,  True,  True, False,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
False,  True, False,  True,  True, False,  True,  True,  True,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
 True,  True,  True,  True,  True,  True,  True,  True,  True,
False,  True,  True,  True,  True,  True, False,  True,  True,
 True,  True,  True,  True,  True, False,  True,  True, False,
 True,  True,  True, False,  True,  True, False,  True,  True,
 True,  True,  True,  True,  True,  True, False,  True, False,
 True,  True,  True,  True,  True,  True, False,  True,  True,
 True,  True,  True,  True,  True,  True,  True,  True, False,
 True,  True,  True, False,  True,  True,  True,  True, False,
False,  True, False, False,  True,  True,  True,  True,  True,
False,  True,  True,  True,  True,  True, False, False, False,
 True, False,  True,  True,  True,  True,  True, False,  True,
 True,  True, False,  True, False,  True,  True,  True,  True,
 True,  True,  True,  True,  True, False,  True,  True,  True,
```

```
       True,  True,  True,  True, False,  True,  True, False, False,
       True,  True, False,  True, False,  True,  True,  True, False,
       True, False,  True,  True,  True,  True,  True,  True, False,
       True,  True,  True, False,  True,  True,  True,  True,  True,
       True,  True, False,  True, False,  True, False,  True,  True,
       True,  True,  True, False, False,  True,  True,  True, False,
       True,  True, False,  True,  True,  True,  True, False,  True,
      False,  True,  True, False,  True,  True, False,  True,  True,
       True, False,  True,  True,  True,  True,  True, False,  True,
       True,  True,  True,  True, False,  True,  True,  True,  True,
       True, False,  True,  True,  True, False, False,  True,  True,
      False,  True, False,  True,  True,  True,  True,  True, False,
       True])
```

[58]:
```
np.mean(test['y'] == y_pred)
```

[58]: 0.731

[59]:
```
err = np.mean(test['y'] != y_pred)
err
```

[59]: 0.269

[60]:
```
#Q4
```

[61]:
```
M = 10
Kmax = 20
ERR100 = np.zeros((M, Kmax))
ERR500 = np.zeros((M, Kmax))
```

[62]:
```
ERR100[9, 2]
```

[62]: 0.0

[63]:
```
for m in range(M):
    print(m+1)
    train100 = gen_data(100,mu1,mu2,mu3,Sigma1,Sigma2,Sigma3,p1=0.3,p2=0.2)
    train500 = gen_data(500,mu1,mu2,mu3,Sigma1,Sigma2,Sigma3,p1=0.3,p2=0.2)
    for k in range(Kmax):
        knn_classifier = KNeighborsClassifier(n_neighbors=k+1)
        knn_classifier.fit(train100['X'], train100['y'])
        y_pred = knn_classifier.predict(test['X'])
        ERR100[m, k] = np.mean(test['y'] != y_pred)

        knn_classifier = KNeighborsClassifier(n_neighbors=k+1)
        knn_classifier.fit(train500['X'], train500['y'])
        y_pred = knn_classifier.predict(test['X'])
        ERR500[m, k] = np.mean(test['y'] != y_pred)
```

```
1
2
3
4
5
6
7
8
9
10
```

[64]:
```
err100 = np.mean(ERR100, axis=0)
err500 = np.mean(ERR500, axis=0)
```

[65]:
```
err100
```

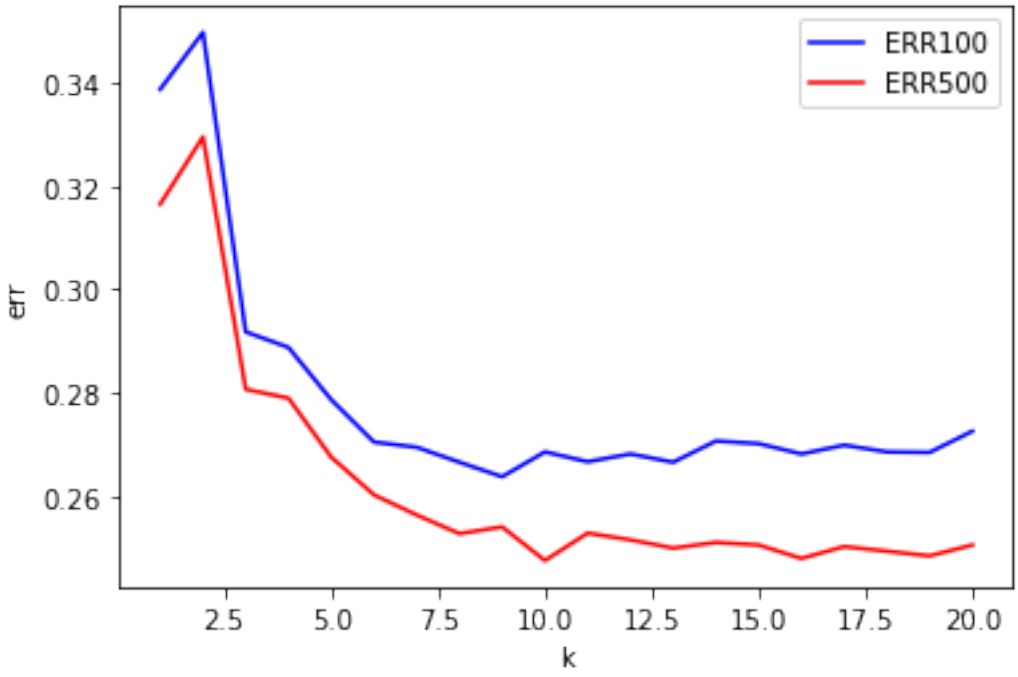
[65]:
```
array([0.3387, 0.3497, 0.2918, 0.2888, 0.2787, 0.2705, 0.2695, 0.2666,
       0.2638, 0.2686, 0.2667, 0.2682, 0.2666, 0.2707, 0.2702, 0.2682,
       0.2699, 0.2686, 0.2685, 0.2726])
```

[66]:
```
err500
```

[66]:
```
array([0.3165, 0.3295, 0.2807, 0.279 , 0.2676, 0.2603, 0.2564, 0.2528,
       0.2541, 0.2476, 0.2529, 0.2516, 0.25  , 0.2511, 0.2506, 0.248 ,
       0.2503, 0.2494, 0.2485, 0.2506])
```

[67]:
```
fig, ax = plt.subplots()
plt.xlabel('k')
plt.ylabel('err')
plt.plot(np.array(range(1, (Kmax+1))), err100, c='blue', label='ERR100')
plt.plot(np.array(range(1, (Kmax+1))), err500, c='red', label='ERR500')
plt.legend()
```

[67]:
```
<matplotlib.legend.Legend at 0x24fbbdbf8c8>
```


ERR100
ERR500
0.34
0.32
0.30
0.28
0.26
err
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
k