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

- import data set `kidney_stones`

In [2]:
```
kidney_stones = pd.read_csv('kidney-stone-dataset.csv')
kidney_stones.head()
```

Out[2]:

	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	target
0	0	1.021	4.91	725	14.0	443	2.45	0
1	1	1.017	5.74	577	20.0	296	4.49	0
2	2	1.008	7.20	321	14.9	101	2.36	0
3	3	1.011	5.51	408	12.6	224	2.15	0
4	4	1.005	6.52	187	7.5	91	1.16	0

In [3]:
```
kidney_stones.shape
```

Out[3]:
```
(90, 8)
```

In [4]:
```
kidney_stones.dtypes
```

Out[4]:
```
Unnamed: 0     int64
gravity      float64
ph           float64
osmo           int64
cond         float64
urea           int64
calc         float64
target         int64
dtype: object
```

In [5]:
```
kidney_stones.describe()
```

Out[5]:

	Unnamed: 0	gravity	ph	osmo	cond	urea	calc	target
count	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000	90.000000
mean	44.500000	1.017652	6.036811	602.333333	15.621667	258.200000	4.017789	0.500000
std	26.124701	0.006780	0.711801	238.459805	7.654448	135.381127	3.016273	0.502801
min	0.000000	1.005000	4.760000	187.000000	5.100000	10.000000	0.170000	0.000000
25%	22.250000	1.012250	5.538520	411.500000	14.150000	148.250000	1.412500	0.000000
50%	44.500000	1.018000	5.936247	572.000000	21.177172	231.500000	3.230000	0.500000
75%	66.750000	1.023000	6.490000	778.000000	26.075000	356.250000	5.965127	1.000000
max	89.000000	1.034000	7.940000	1236.000000	38.000000	620.000000	13.000000	1.000000

In [6]:
```
kidney_stones.isna().sum()
```

Out[6]:
```
Unnamed: 0    0
gravity       0
ph            0
osmo          0
cond          0
urea          0
calc          0
target        0
dtype: int64
```

In [7]:
```
kidney_stones.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90 entries, 0 to 89
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  90 non-null     int64  
 1   gravity     90 non-null     float64
 2   ph          90 non-null     float64
 3   osmo        90 non-null     int64  
 4   cond        90 non-null     float64
 5   urea        90 non-null     int64  
 6   calc        90 non-null     float64
 7   target      90 non-null     int64  
dtypes: float64(4), int64(4)
memory usage: 5.8 KB
```

In [8]:
```
kidney_stones.target.value_counts()
```

Out[8]:
```
0    45
1    45
Name: target, dtype: int64
```

drop additional index from the dataset

In [9]:
```
kidney_stones.drop('Unnamed: 0', axis = 1, inplace = True)
```

In [10]:
```
kidney_stones
```

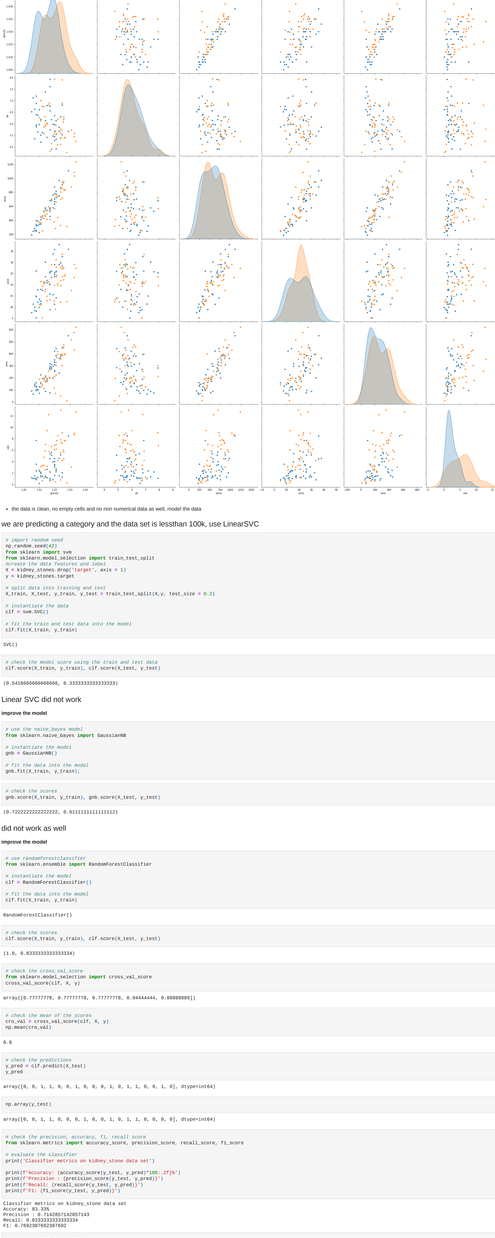
Out[10]:

	gravity	ph	osmo	cond	urea	calc	target
0	1.021000	4.910000	725	14.000000	443	2.450000	0
1	1.017000	5.740000	577	20.000000	296	4.490000	0
2	1.008000	7.200000	321	14.900000	101	2.360000	0
3	1.011000	5.510000	408	12.600000	224	2.150000	0
4	1.005000	6.520000	187	7.500000	91	1.160000	0
...	...	...	...	...	...	...	...
85	1.021452	5.556081	756	24.241481	357	7.669120	1
86	1.016501	6.900257	549	20.548790	204	5.775256	1
87	1.032754	5.443491	1085	23.188653	576	8.664169	1
88	1.025870	5.156410	325	12.134809	50	5.765820	1
89	1.013723	8.309843	472	16.907792	374	2.556405	1

90 rows × 7 columns

* visualize the data using seaborn

In [11]:
```
sns.pairplot(kidney_stones, hue = 'target', height = 5);
```



- the data is clean, no empty cells and no non numerical data as well, model the data

we are predicting a category and the data set is lessthan 100k, use LinearSVC

In [12]:
```
# import random seed
np.random.seed(42)
from sklearn import svm
from sklearn.model_selection import train_test_split
#create the data features and labels
X = kidney_stones.drop('target', axis = 1)
y = kidney_stones.target

# split data into training and test
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)

# instantiate the data
clf = svm.SVC()

# fit the train and test data into the model
clf.fit(X_train, y_train)
```

Out[12]:
```
SVC()
```

In [13]:
```
# check the model score using the train and test data
clf.score(X_train, y_train), clf.score(X_test, y_test)
```

Out[13]:
```
(0.5416666666666666, 0.3333333333333333)
```

Linear SVC did not work

improve the model

In [14]:
```
# use the naive_bayes model
from sklearn.naive_bayes import GaussianNB

# instantiate the model
gnb = GaussianNB()

# fit the data into the model
gnb.fit(X_train, y_train);
```

In [15]:
```
# check the scores
gnb.score(X_train, y_train), gnb.score(X_test, y_test)
```

Out[15]:
```
(0.7222222222222222, 0.6111111111111112)
```

did not work as well

improve the model

In [16]:
```
# use randomforestclassifier
from sklearn.ensemble import RandomForestClassifier

# instantiate the model
clf = RandomForestClassifier()

# fit the data into the model
clf.fit(X_train, y_train)
```

Out[16]:
```
RandomForestClassifier()
```

In [17]:
```
# check the scores
clf.score(X_train, y_train), clf.score(X_test, y_test)
```

Out[17]:
```
(1.0, 0.8333333333333334)
```

In [18]:
```
# check the cross_val_score
from sklearn.model_selection import cross_val_score
cross_val_score(clf, X, y)
```

Out[18]:
```
array([0.77777778, 0.77777778, 0.77777778, 0.94444444, 0.88888889])
```

In [19]:
```
# check the mean of the scores
cro_val = cross_val_score(clf, X, y)
np.mean(cro_val)
```

Out[19]:
```
0.8
```

In [20]:
```
# check the predictions
y_pred = clf.predict(X_test)
y_pred
```

Out[20]:
```
array([0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0], dtype=int64)
```

In [21]:
```
np.array(y_test)
```

Out[21]:
```
array([0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0], dtype=int64)
```

In [22]:
```
# check the precision, accuracy, f1, recall score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# evaluate the classifier
print('Classifier metrics on kidney_stone data set')

print(f'Accuracy: {accuracy_score(y_test, y_pred)*100:.2f}%')
print(f'Precision : {precision_score(y_test, y_pred)}')
print(f'Recall: {recall_score(y_test, y_pred)}')
print(f'F1: {f1_score(y_test, y_pred)}')
```

```
Classifier metrics on kidney_stone data set
Accuracy: 83.33%
Precision : 0.7142857142857143
Recall: 0.8333333333333334
F1: 0.7692307692307692
```

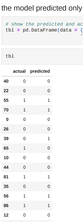
In [23]:
```
# check the confusion_matrix
from sklearn.metrics import confusion_matrix
conf_mat = confusion_matrix(y_test, y_pred)
conf_mat
```

Out[23]:
```
array([[10,  2],
       [ 1,  5]], dtype=int64)
```

In [24]:
```
# visualize confusion matrix using seaborn heatmap
def plot_mat(conf_mat):
    fig, ax = plt.subplots(figsize = (3,3))
    ax = sns.heatmap(conf_mat,
                     annot = True,
                     cbar = False)

    plt.xlabel("Actual");
    plt.ylabel("Predicted");

plot_mat(conf_mat)
```



the model predicted only 3 wrongly

In [25]:
```
# show the predicted and actual data in a table
tbl = pd.DataFrame(data = {'actual': y_test,
                           'predicted': y_pred})
```

In [26]:
```
tbl
```

Out[26]:

	actual	predicted
40	0	0
22	0	0
55	1	1
70	1	1
9	0	0
26	0	0
39	0	1
65	1	0
10	0	0
44	0	0
61	1	1
35	0	0
56	1	1
86	1	1
12	0	0
4	0	0
18	0	1
28	0	0

In [ ]:
```

```

In [ ]:
```

```