IPA 주관 인공지능센터 기본(fundamental) 과정

- GitHub link: here
- E-Mail: windkyle7@gmail.com

What is Scikit-Learn?

- 공식 홈페이지: https://scikit-learn.org/
- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as mino
```

tidy-data: iris 불러오기

seaborn을 통해 불러올 때 다음과 같이 불러왔다.

In [2]:
```
iris_seaborn = sns.load_dataset('iris')
iris_seaborn.sample(5)
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
21	5.1	3.7	1.5	0.4	setosa
139	6.9	3.1	5.4	2.1	virginica
26	5.0	3.4	1.6	0.4	setosa
59	5.2	2.7	3.9	1.4	versicolor
140	6.7	3.1	5.6	2.4	virginica

sklearn을 통해 tidy-data 불러오기

`scikit-learn` 패키지가 없을 경우 아래의 명령어를 주피터 셀에 입력해준다.

```
dos
%pip install --upgrade sklearn
```

In [3]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

불러온 데이터셋을 `dir` 로 살펴보면 아래와 같다.

In [4]:
```
dir(iris_dataset)
```

Out[4]:
```
['DESCR', 'data', 'feature_names', 'filename', 'target', 'target_names']
```

`scikit-learn` 을 통해 불러온 데이터셋에 대한 설명을 볼 때는 `DESCR` 를 통해 볼 수 있다.

In [5]:
```
print(iris_dataset.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
```

```
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

학습에 필요한 학습 데이터셋과 정답셋을 `DataFrame` 으로 바꿔준다.

In [6]:
```
iris_data = pd.DataFrame(iris_dataset.data, columns=iris_dataset.feature_names)
iris_target = pd.DataFrame(iris_dataset.target, columns=['target'])
```

In [7]:
```
iris_data.info()
iris_data.sample(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
dtypes: float64(4)
memory usage: 4.8 KB
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
128	6.4	2.8	5.6	2.1
73	6.1	2.8	4.7	1.2
0	5.1	3.5	1.4	0.2
70	5.9	3.2	4.8	1.8
57	4.9	2.4	3.3	1.0

In [8]:
```
iris_target.info()
iris_target.sample(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 1 columns):
target    150 non-null int64
dtypes: int64(1)
memory usage: 1.2 KB
```

Out[8]:

	target
146	2
55	1
112	2
92	1
70	1

`concat` 으로 데이터들을 합쳐준다.

In [9]:
```
iris = pd.concat([iris_data, iris_target], axis=1)
```

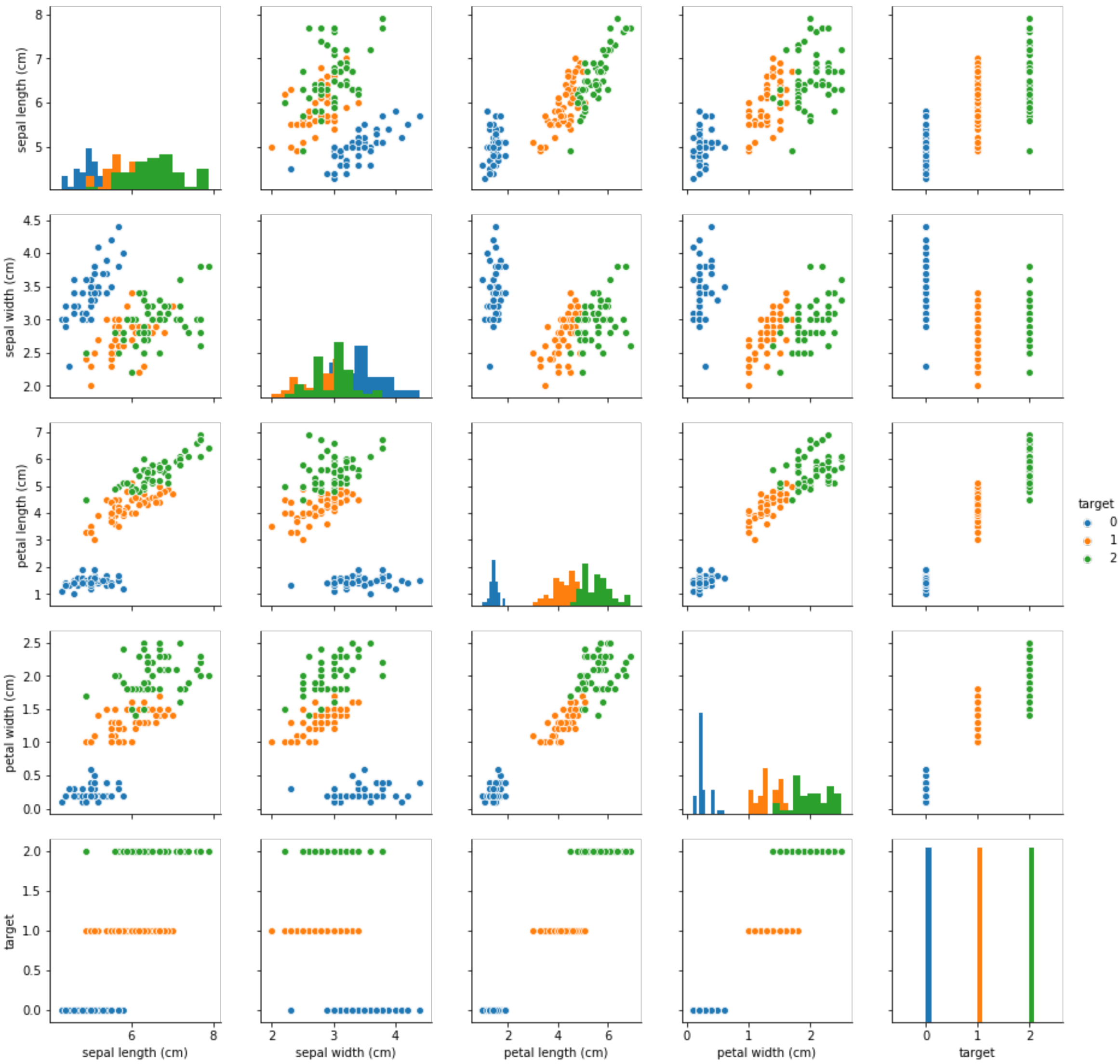
학습을 시키기 전에 `Visualization` 해본다.

- 아래의 그래프를 보면 데이터들을 classfication 할 수 있다는 것을 알 수 있다.
- classfication이 가능하다는 것은 곧 기계학습을 할 수 있다는 것을 확인할 수 있다

```
In [10]: iris['target'] = iris['target'].astype('category')
```

```
In [11]: sns.pairplot(iris, diag_kind='hist', hue='target')
```

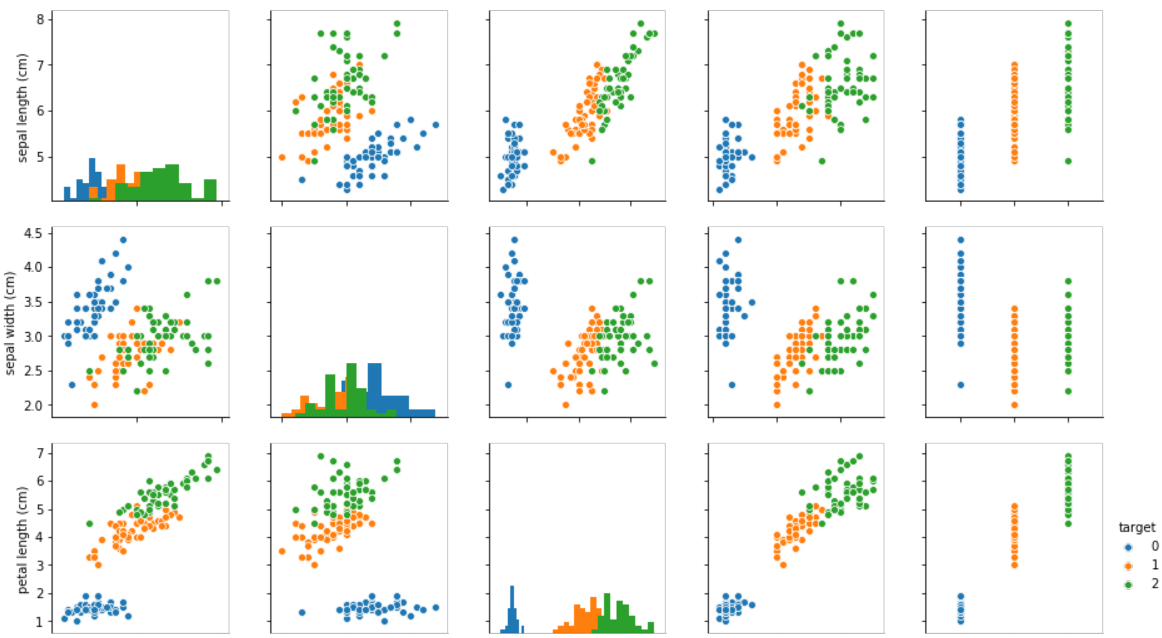
```
Out[11]: <seaborn.axisgrid.PairGrid at 0x7fc070359470>
```

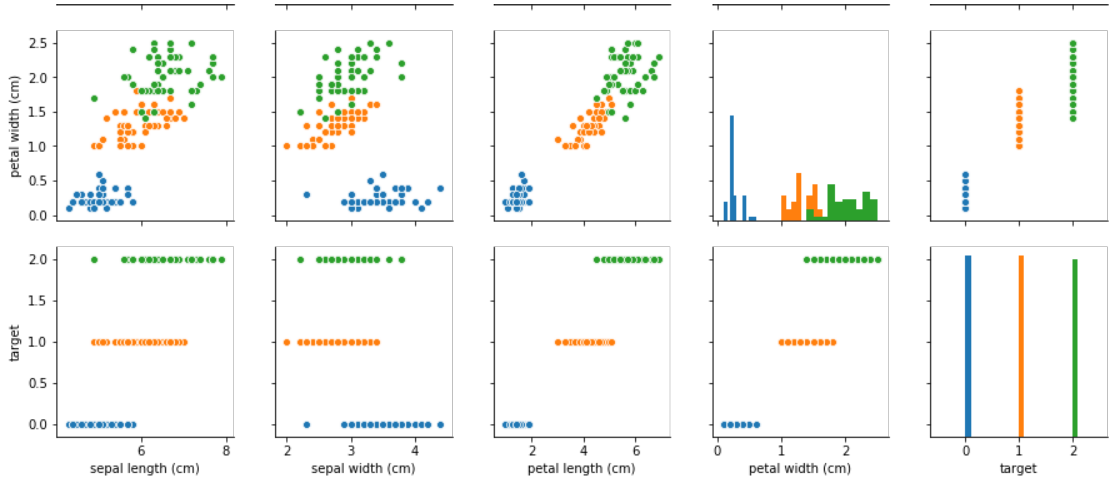


iloc로 가져오기

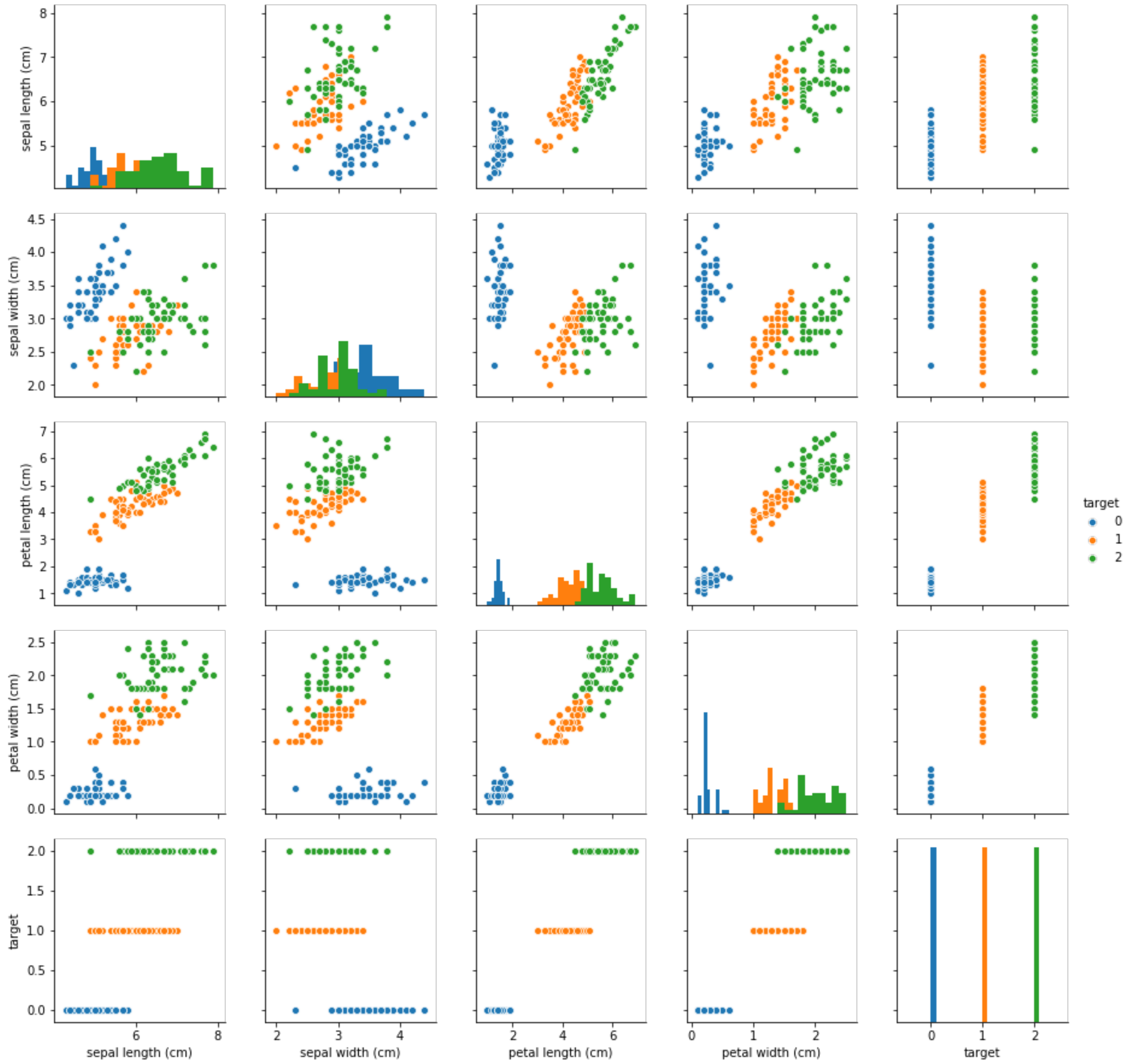
```
In [12]: sns.pairplot(iris.iloc[:-1], diag_kind='hist', hue='target')
```

```
Out[12]: <seaborn.axisgrid.PairGrid at 0x7fc038e344a8>
```






```
In [13]: sns.pairplot(iris, vars=iris.iloc[:-1], diag_kind='hist', hue='target')
Out[13]: <seaborn.axisgrid.PairGrid at 0x7fc0325ca320>
```



tidy-data: boston 가져오기

```
In [14]: from sklearn.datasets import load_boston
```

```
In [15]: boston_dataset = load_boston()
         boston_data = pd.DataFrame(boston_dataset.data,
                                    columns=boston_dataset.feature_names)
         boston_target = pd.DataFrame(boston_dataset.target, columns=['target'])
```

```
In [16]: boston_data.info()
         boston_data.sample(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
```

```
CRIM        506 non-null float64
ZN          506 non-null float64
INDUS       506 non-null float64
CHAS        506 non-null float64
NOX         506 non-null float64
RM          506 non-null float64
AGE         506 non-null float64
DIS         506 non-null float64
RAD         506 non-null float64
TAX         506 non-null float64
PTRATIO     506 non-null float64
B           506 non-null float64
LSTAT       506 non-null float64
dtypes: float64(13)
memory usage: 51.5 KB
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
171	2.31390	0.0	19.58	0.0	0.605	5.880	97.3	2.3887	5.0	403.0	14.7	348.13	12.03
322	0.35114	0.0	7.38	0.0	0.493	6.041	49.9	4.7211	5.0	287.0	19.6	396.90	7.70
379	17.86670	0.0	18.10	0.0	0.671	6.223	100.0	1.3861	24.0	666.0	20.2	393.74	21.78
483	2.81838	0.0	18.10	0.0	0.532	5.762	40.3	4.0983	24.0	666.0	20.2	392.92	10.42
333	0.05083	0.0	5.19	0.0	0.515	6.316	38.1	6.4584	5.0	224.0	20.2	389.71	5.68

In [17]:

```python
boston_target.info()
boston_target.sample(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 1 columns):
target     506 non-null float64
dtypes: float64(1)
memory usage: 4.0 KB
```

Out[17]:

	target
125	21.4
204	50.0
0	24.0
360	25.0
235	24.0

In [18]:

```python
boston_target['target'] = boston_target['target'].astype('category')
```

In [19]:

```python
boston = pd.concat([boston_data, boston_target], axis=1)
boston.info()
boston.sample(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM        506 non-null float64
ZN          506 non-null float64
INDUS       506 non-null float64
CHAS        506 non-null float64
NOX         506 non-null float64
RM          506 non-null float64
AGE         506 non-null float64
DIS         506 non-null float64
RAD         506 non-null float64
TAX         506 non-null float64
PTRATIO     506 non-null float64
B           506 non-null float64
LSTAT       506 non-null float64
target      506 non-null category
dtypes: category(1), float64(13)
memory usage: 64.2 KB
```

Out[19]:

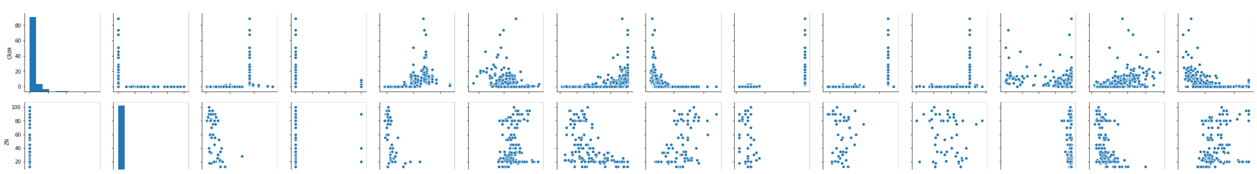
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
188	0.12579	45.0	3.44	0.0	0.437	6.556	29.1	4.5667	5.0	398.0	15.2	382.84	4.56	29.8
234	0.44791	0.0	6.20	1.0	0.507	6.726	66.5	3.6519	8.0	307.0	17.4	360.20	8.05	29.0
65	0.03584	80.0	3.37	0.0	0.398	6.290	17.8	6.6115	4.0	337.0	16.1	396.90	4.67	23.5
271	0.16211	20.0	6.96	0.0	0.464	6.240	16.3	4.4290	3.0	223.0	18.6	396.90	6.59	25.2
412	18.81100	0.0	18.10	0.0	0.597	4.628	100.0	1.5539	24.0	666.0	20.2	28.79	34.37	17.9

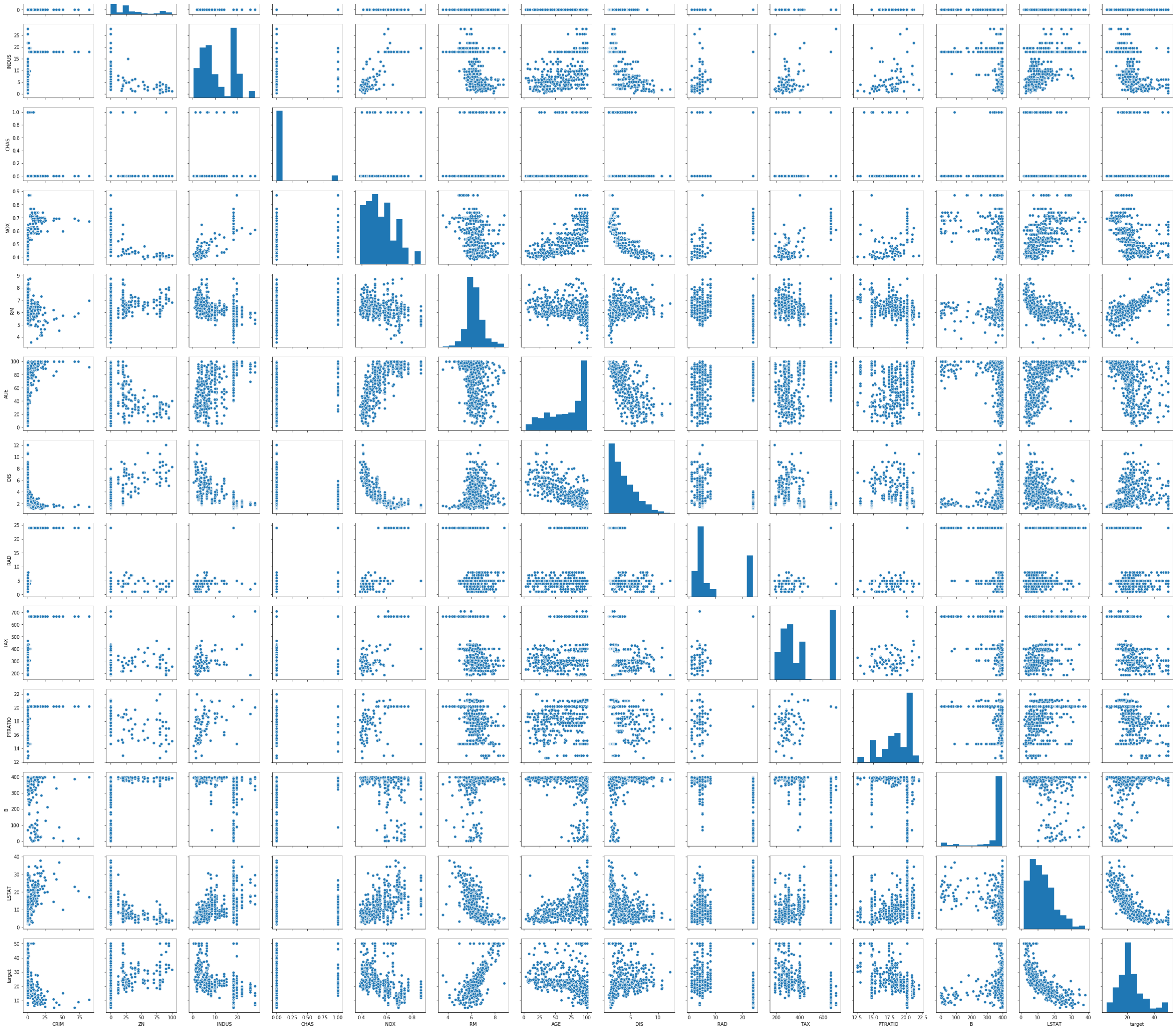
Visualization

In [20]:

```python
sns.pairplot(boston)
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x7fc03149bf60>





상관관계 확인

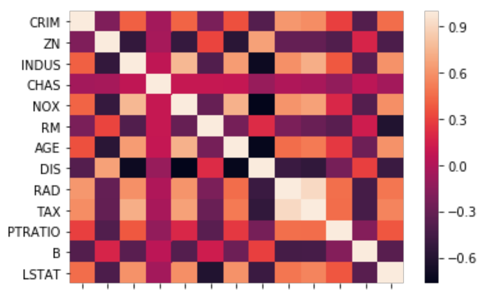
In [21]:
```
boston.corr()
```

Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366087
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.000000

In [22]:
```
sns.heatmap(boston.corr())
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc02b683cf8>



CRIM ZN INDUS CHAS NOX RM AGE DIS RAD TAX PTRATIO B LSTAT

학습 시키기

In [23]:
```python
from sklearn.neighbors import KNeighborsClassifier
```

Hyper Parameter

- 신경망 학습을 통해서 튜닝 또는 최적화 해야하는 주변수가 아니라 학습 진도율이나 일반화 변수처럼 사람들이 선험적 지식으로 설정을 하거나 외부 모델 메커니즘을 통해 자동으로 설정이 되는 변수를 의미한다.

`KNN` 모델에 적용되는 파라미터는 `Hyper Parameter` 이다.

In [24]:
```python
knn = KNeighborsClassifier()
```

`fit` 메소드를 호출할 때, `X` 에는 학습 데이터셋, `y` 에는 정답셋을 넣어준다.

In [25]:
```python
knn.fit(iris.iloc[:, :-1], iris.iloc[:, -1])
```

Out[25]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform')
```

`predict` 를 통해 예측을 시도한다.

In [26]:
```python
pred = knn.predict([[3, 3, 3, 3]])
```

예측된 결과를 확인해본다.

In [27]:
```python
iris_dataset.target_names[pred]
```

Out[27]:
```
array(['versicolor'], dtype='<U10')
```

boston 데이터로 학습

In [28]:
```python
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor()
knn.fit(boston.iloc[:, :-1], boston.iloc[:, -1])
knn.predict([boston.iloc[2, :-1]])
```

Out[28]:
```
array([25.36])
```

In [29]:
```python
boston.iloc[2, -1]
```

Out[29]:
```
34.7
```