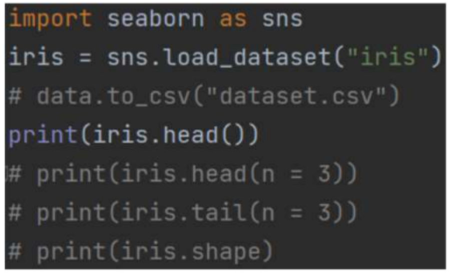
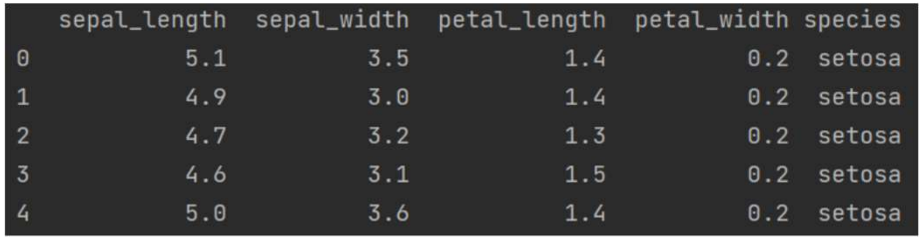
붓꽃(Iris) 품종 분류

데이터 입력



```
import seaborn as sns
iris = sns.load_dataset("iris")
# data.to_csv("dataset.csv")
print(iris.head())
# print(iris.head(n = 3))
# print(iris.tail(n = 3))
# print(iris.shape)
```



```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

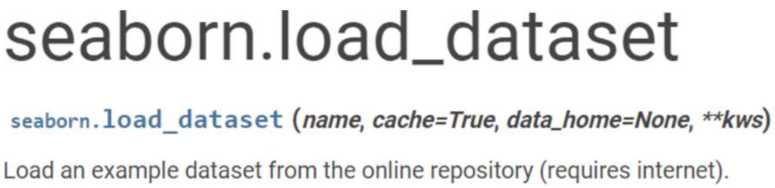
https://en.wikipedia.org/wiki/Iris_flower_data_set

Setosa

Versicolor

Verginica

붓꽃 품종



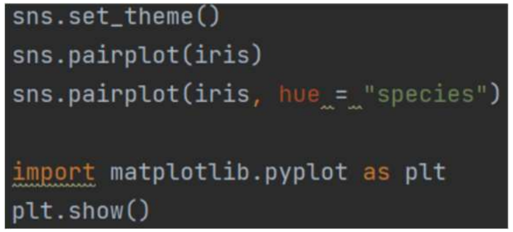
seaborn.load_dataset

`seaborn.load_dataset` (*name, cache=True, data_home=None, **kws*)

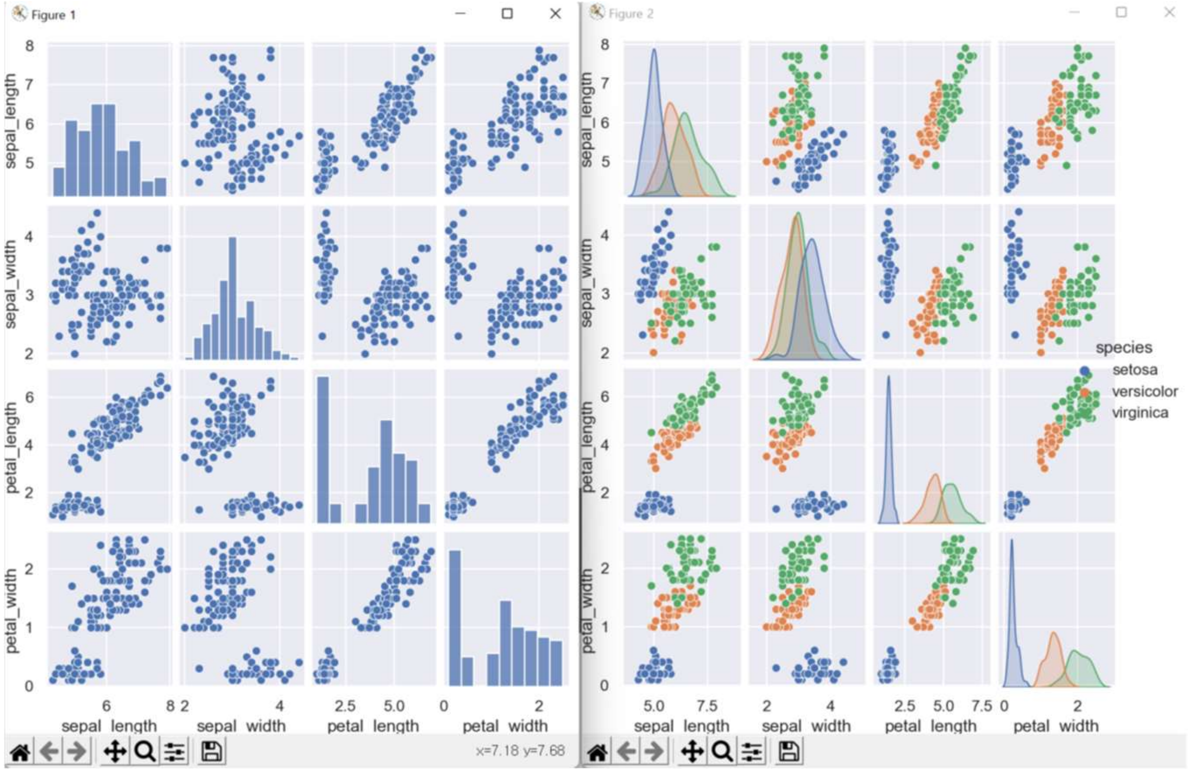
Load an example dataset from the online repository (requires internet).

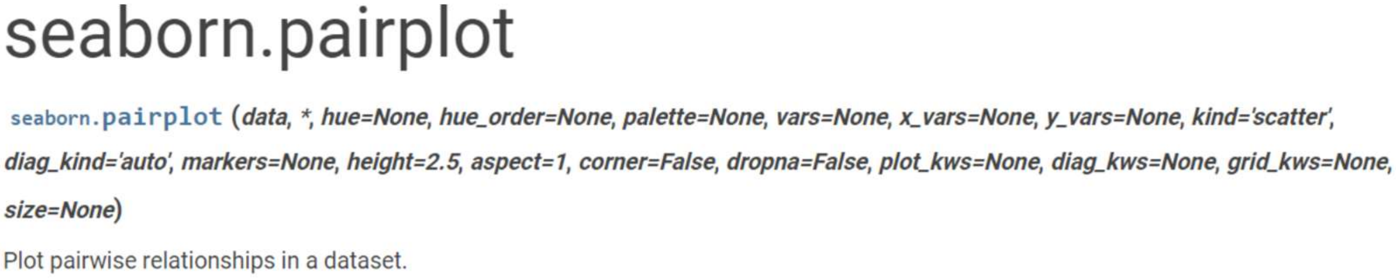
https://seaborn.pydata.org/generated/seaborn.load_dataset.html#seaborn.load_dataset

데이터 시각화



```
sns.set_theme()
sns.pairplot(iris)
sns.pairplot(iris, hue = "species")

import matplotlib.pyplot as plt
plt.show()
```





seaborn.pairplot

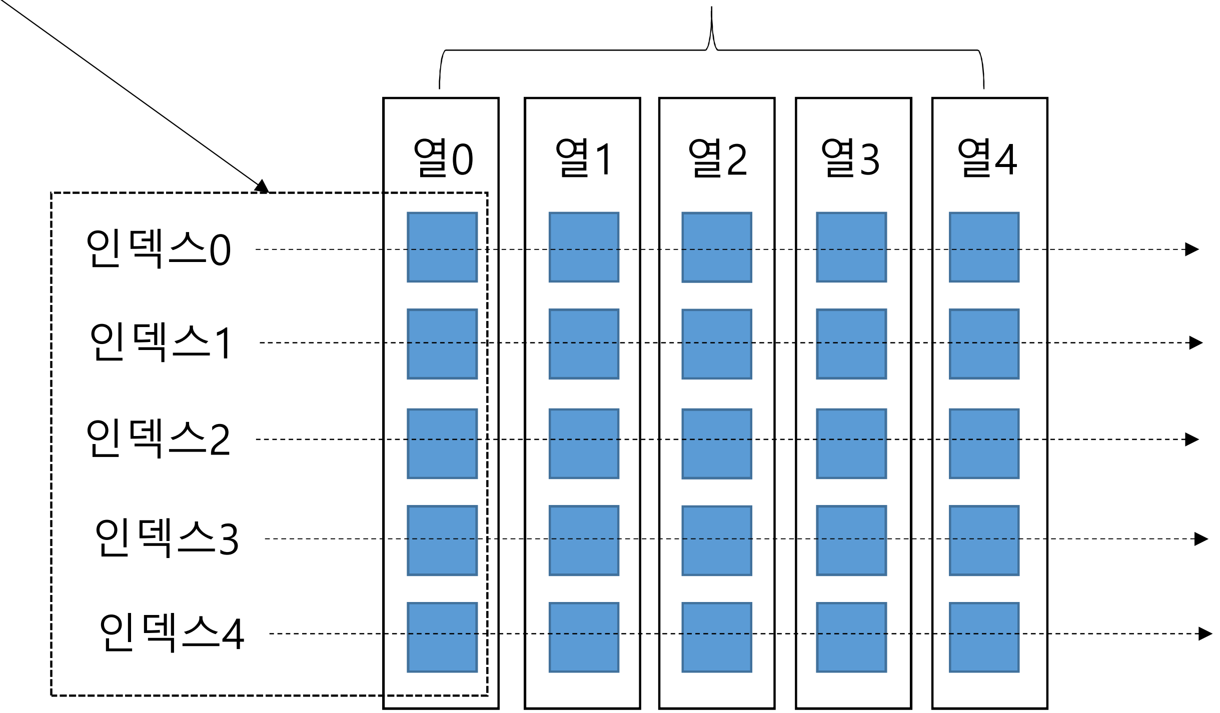
`seaborn.pairplot` (*data, *, hue=None, hue_order=None, palette=None, vars=None, x_vars=None, y_vars=None, kind='scatter', diag_kind='auto', markers=None, height=2.5, aspect=1, corner=False, dropna=False, plot_kws=None, diag_kws=None, grid_kws=None, size=None*)

Plot pairwise relationships in a dataset.

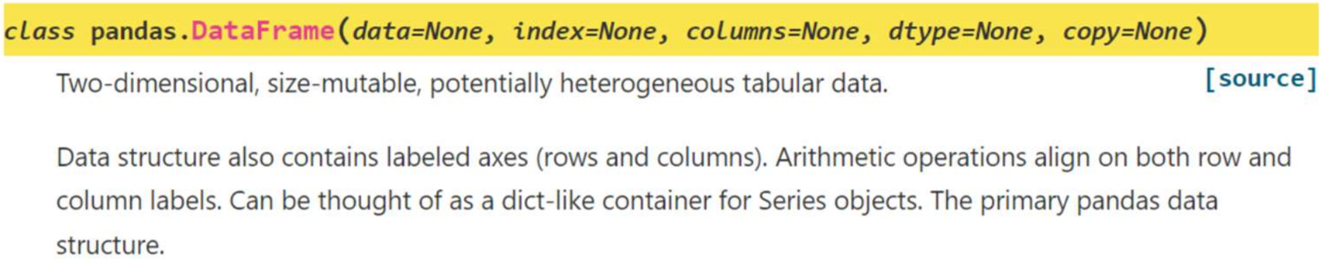
https://seaborn.pydata.org/generated/seaborn.pairplot.html#seaborn.pairplot

판다스 자료구조

- 행 방향으로는 행 인덱스, 열 방향으로는 열 이름으로 구성하는 데이터프레임(DataFrame)
- 시리즈(Series)는 데이터 값의 1차원 벡터, 데이터프레임은 2차원 구조를 의미



pandas.DataFrame



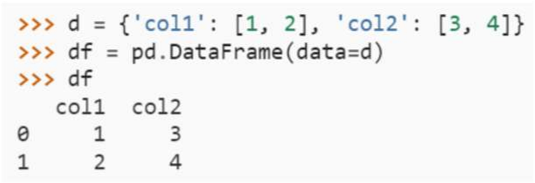
```
class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)
```

[source]

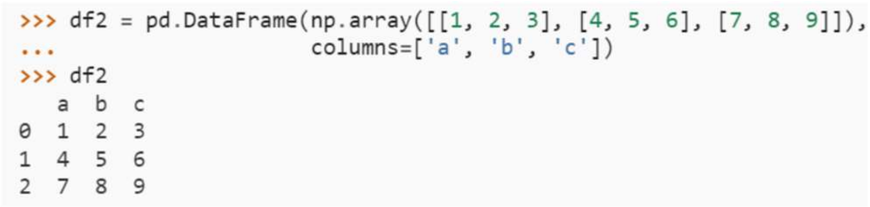
Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

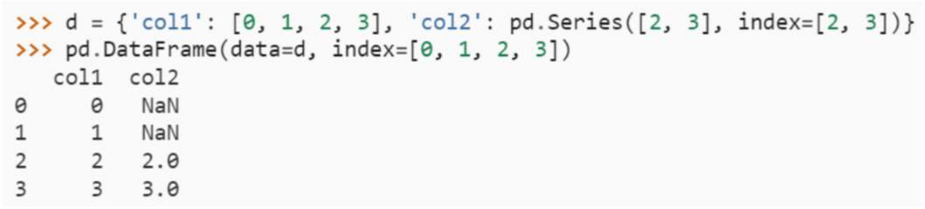
https://numpy.org/doc/stable/reference/arrays.ndarray.html



```
>>> d = {'col1': [1, 2], 'col2': [3, 4]}
>>> df = pd.DataFrame(data=d)
>>> df
   col1  col2
0     1     3
1     2     4
```



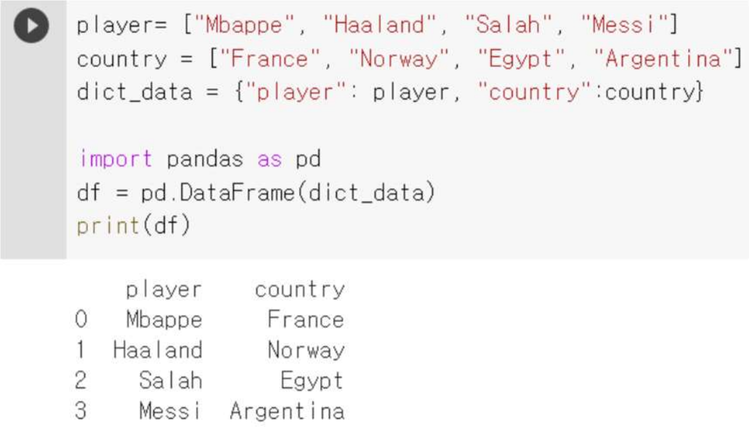
```
>>> df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
...                    columns=['a', 'b', 'c'])
>>> df2
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```



```
>>> d = {'col1': [0, 1, 2, 3], 'col2': pd.Series([2, 3], index=[2, 3])}
>>> pd.DataFrame(data=d, index=[0, 1, 2, 3])
   col1  col2
0     0   NaN
1     1   NaN
2     2   2.0
3     3   3.0
```

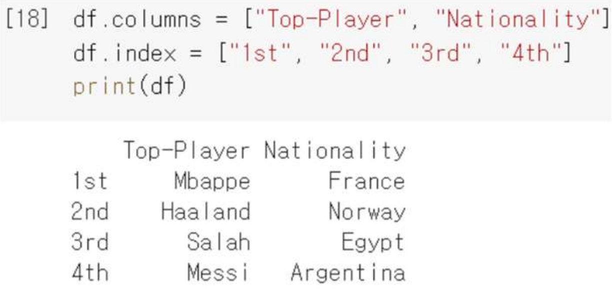
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame

데이터프레임 접근



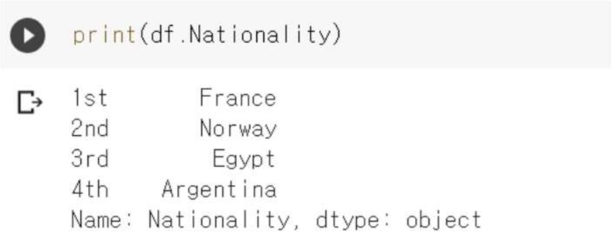
```
player= ["Mbappe", "Haaland", "Salah", "Messi"]
country = ["France", "Norway", "Egypt", "Argentina"]
dict_data = {"player": player, "country":country}

import pandas as pd
df = pd.DataFrame(dict_data)
print(df)
```

```
    player    country
0   Mbappe     France
1  Haaland     Norway
2    Salah      Egypt
3    Messi  Argentina
```



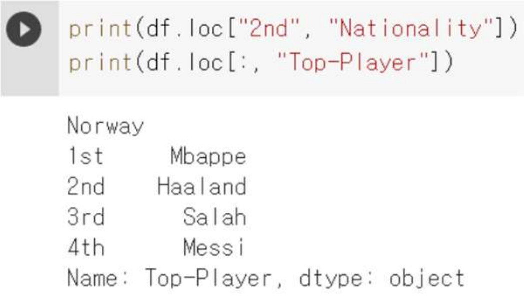
```
[18] df.columns = ["Top-Player", "Nationality"]
     df.index = ["1st", "2nd", "3rd", "4th"]
     print(df)
```

```
    Top-Player Nationality
1st     Mbappe      France
2nd    Haaland      Norway
3rd      Salah       Egypt
4th      Messi   Argentina
```



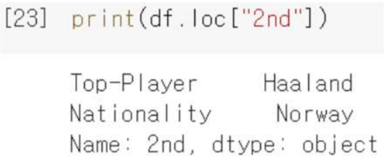
```
print(df.Nationality)
```

```
1st       France
2nd       Norway
3rd        Egypt
4th    Argentina
Name: Nationality, dtype: object
```



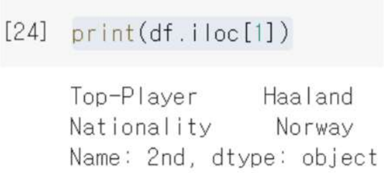
```
print(df.loc["2nd", "Nationality"])
print(df.loc[:, "Top-Player"])
```

```
Norway
1st     Mbappe
2nd    Haaland
3rd      Salah
4th      Messi
Name: Top-Player, dtype: object
```



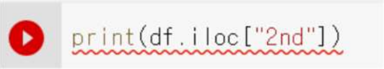
```
[23] print(df.loc["2nd"])
```

```
Top-Player     Haaland
Nationality     Norway
Name: 2nd, dtype: object
```



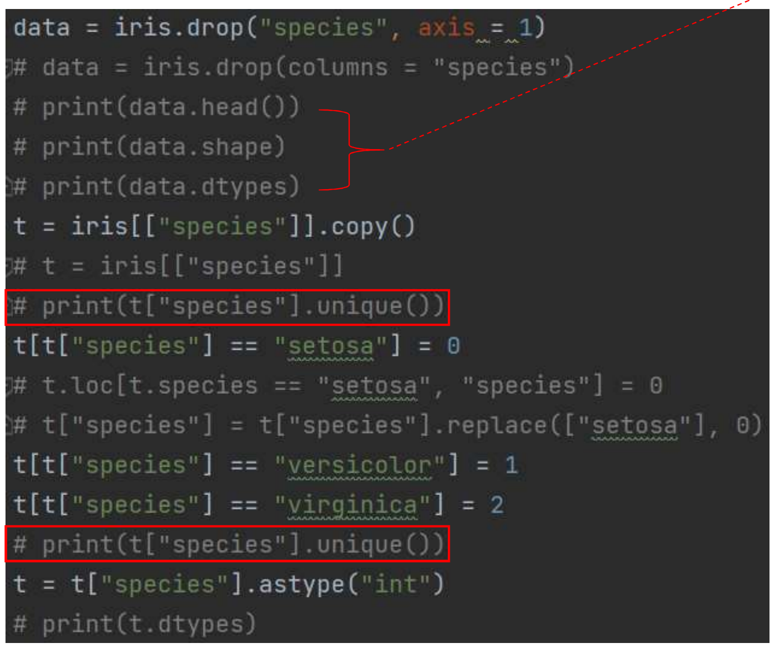
```
[24] print(df.iloc[1])
```

```
Top-Player     Haaland
Nationality     Norway
Name: 2nd, dtype: object
```

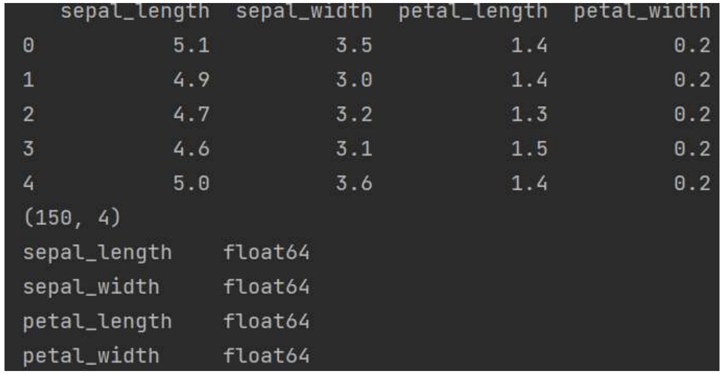


```
print(df.iloc["2nd"])
```

데이터 전처리

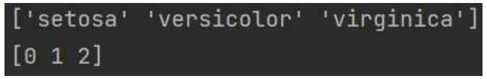


```
data = iris.drop("species", axis = 1)
# data = iris.drop(columns = "species")
# print(data.head())
# print(data.shape)
# print(data.dtypes)
t = iris[["species"]].copy()
# t = iris[["species"]]
# print(t["species"].unique())
t[t["species"] == "setosa"] = 0
# t.loc[t.species == "setosa", "species"] = 0
# t["species"] = t["species"].replace(["setosa"], 0)
t[t["species"] == "versicolor"] = 1
t[t["species"] == "virginica"] = 2
# print(t["species"].unique())
t = t["species"].astype("int")
# print(t.dtypes)
```



```
   sepal_length  sepal_width  petal_length  petal_width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          3.6           1.4          0.2
(150, 4)
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
```

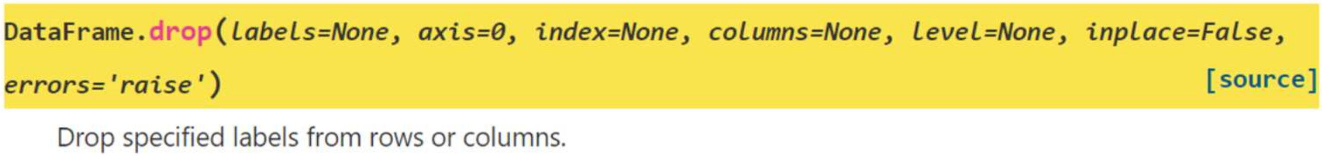
Setosa	Versicolor	Verginica
0	1	2



```
['setosa' 'versicolor' 'virginica']
[0 1 2]
```

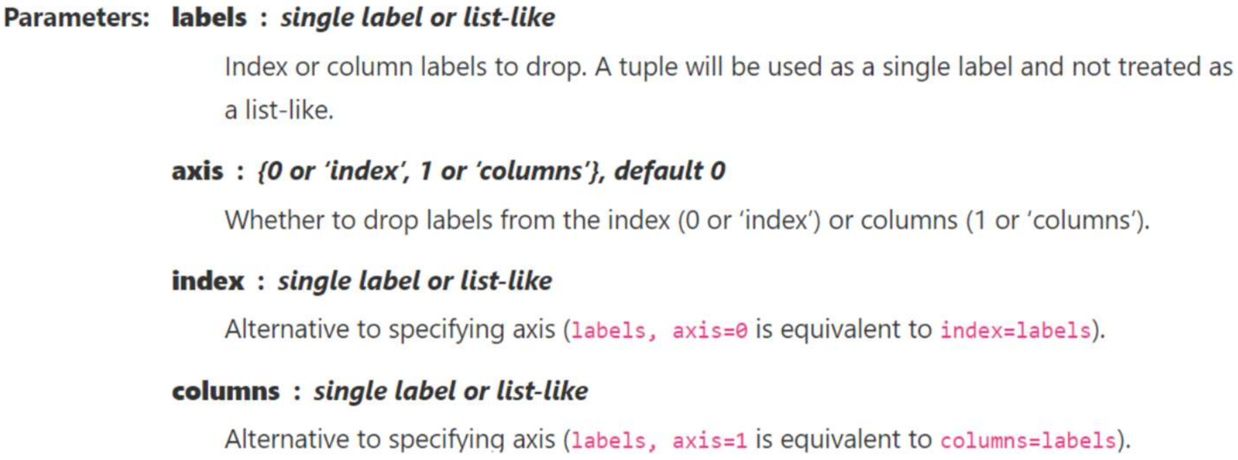
SEJONG UNIVERSITY

pandas.DataFrame.drop



```
DataFrame.drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False,
errors='raise')                                                                  [source]
```

Drop specified labels from rows or columns.



Parameters: **labels** : ***single label or list-like***

Index or column labels to drop. A tuple will be used as a single label and not treated as a list-like.

axis : ***{0 or 'index', 1 or 'columns'}, default 0***

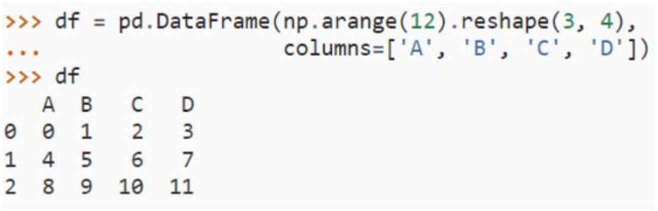
Whether to drop labels from the index (0 or 'index') or columns (1 or 'columns').

index : ***single label or list-like***

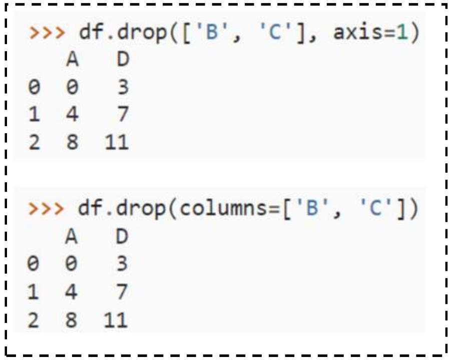
Alternative to specifying axis (`labels, axis=0` is equivalent to `index=labels`).

columns : ***single label or list-like***

Alternative to specifying axis (`labels, axis=1` is equivalent to `columns=labels`).



```
>>> df = pd.DataFrame(np.arange(12).reshape(3, 4),
...                   columns=['A', 'B', 'C', 'D'])
>>> df
   A  B   C   D
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

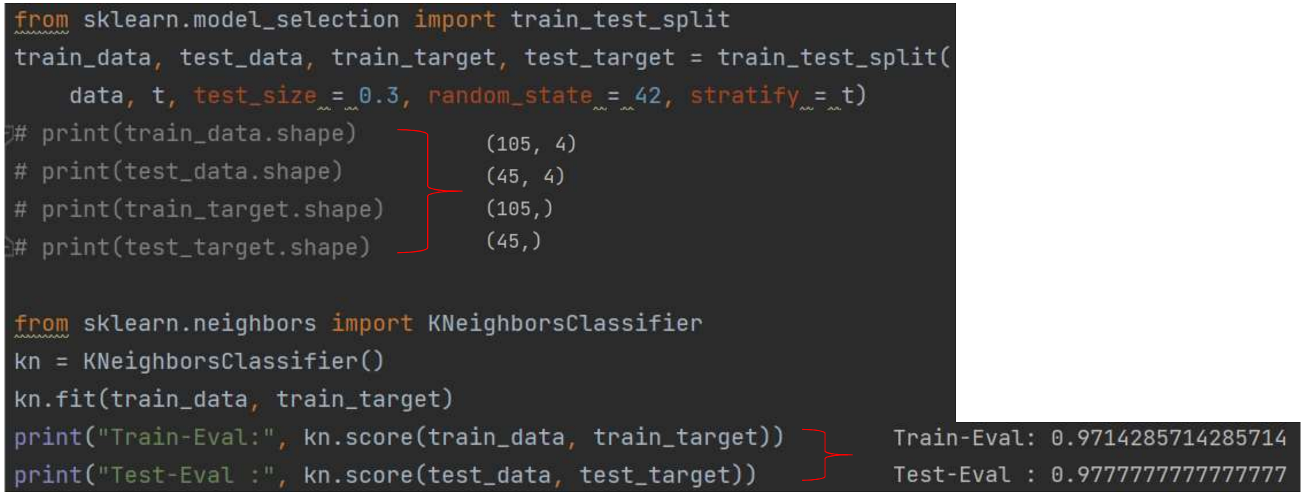


```
>>> df.drop(['B', 'C'], axis=1)
   A   D
0  0   3
1  4   7
2  8  11
```

```
>>> df.drop(columns=['B', 'C'])
   A   D
0  0   3
1  4   7
2  8  11
```

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame

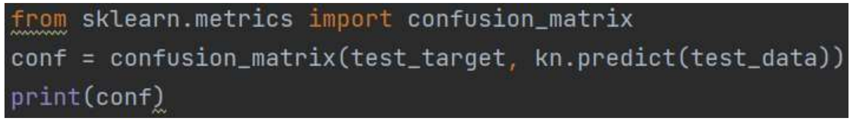
KNN



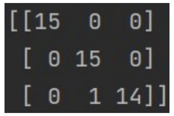
```
from sklearn.model_selection import train_test_split
train_data, test_data, train_target, test_target = train_test_split(
    data, t, test_size = 0.3, random_state = 42, stratify = t)
# print(train_data.shape)
# print(test_data.shape)
# print(train_target.shape)
# print(test_target.shape)

from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier()
kn.fit(train_data, train_target)
print("Train-Eval:", kn.score(train_data, train_target))
print("Test-Eval :", kn.score(test_data, test_target))
```

```
(105, 4)
(45, 4)
(105,)
(45,)
```

```
Train-Eval: 0.9714285714285714
Test-Eval : 0.9777777777777777
```



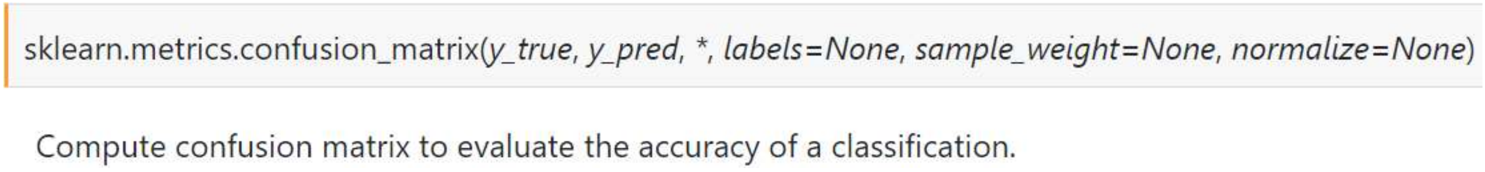
```
from sklearn.metrics import confusion_matrix
conf = confusion_matrix(test_target, kn.predict(test_data))
print(conf)
```



```
[[15  0  0]
 [ 0 15  0]
 [ 0  1 14]]
```

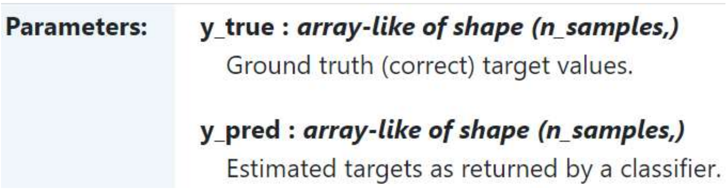
	예측0	예측1	예측2
정답0	15개	0개	0개
정답1	0개	15개	0개
정답2	0개	1개	14개

sklearn.metrics.confusion_matrix



sklearn.metrics.confusion_matrix(*y_true*, *y_pred*, ***, *labels=None*, *sample_weight=None*, *normalize=None*)

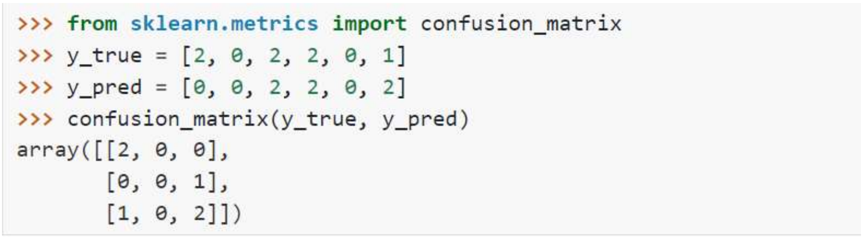
Compute confusion matrix to evaluate the accuracy of a classification.



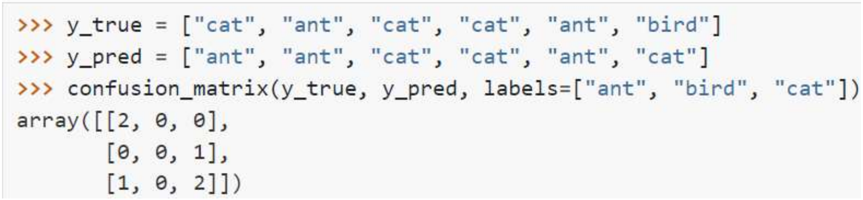
Parameters:

y_true : *array-like of shape (n_samples,)*
Ground truth (correct) target values.

y_pred : *array-like of shape (n_samples,)*
Estimated targets as returned by a classifier.



```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```



```
>>> y_true = ["cat", "ant", "cat", "cat", "ant", "bird"]
>>> y_pred = ["ant", "ant", "cat", "cat", "ant", "cat"]
>>> confusion_matrix(y_true, y_pred, labels=["ant", "bird", "cat"])
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

혼합행렬(Confusion Matrix):

원 클래스와 예측 클래스의
일치 여부를 나타내는 행렬

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_matrix

참고자료

- 지능기전공학부 최유경 교수님 자료, https://github.com/sejongresearch/2021.MachineLearning
- 코랩(Colab), https://colab.research.google.com/
- 파이썬(Python), https://www.python.org/doc/
- 사이킷런(sckit-learn), https://scikit-learn.org/stable/index.html
- 판다스(pandas), https://pandas.pydata.org/
- 맷플롯립(matplotlib), https://matplotlib.org/
- 씨본(seaborn), https://seaborn.pydata.org/
- 캐글(Kaggle), https://www.kaggle.com/
- 넘파이(numpy), https://numpy.org/doc/stable/
- 스택오퍼플러우(stackoverflow), https://stackoverflow.com/