In [ ]:

```
X = [[ 1,  2,  3, 0],
     [11, 12, 13, 1]]
```

In [11]:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=123)

X = [[ 1,  2,  3],
     [11, 12, 13],
    [23, 33, 45]]
y = [0, 1, 0]  # classes of each sample

clf.fit(X, y)

clf.predict(X)
```

Out[11]:

```
array([0, 1, 0])
```

In [14]:

```
clf.predict([234, 23, 456])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-40ed47e22509> in <module>
----> 1 clf.predict([234, 23, 456])

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict(self, X)
    627             The predicted classes.
    628         """
--> 629         proba = self.predict_proba(X)
    630 
    631         if self.n_outputs_ == 1:

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in predict_proba(self, X)
    671         check_is_fitted(self)
    672         # Check data
--> 673         X = self._validate_X_predict(X)
    674 
    675         # Assign chunk of trees to jobs

~\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py in _validate_X_predict(self, X)
    419         check_is_fitted(self)
    420 
--> 421         return self.estimators_[0]._validate_X_predict(X, check_input=True)
    422 
    423     @property

~\anaconda3\lib\site-packages\sklearn\tree\_classes.py in _validate_X_predict(self, X, check_input)
    386         """Validate X whenever one tries to predict, apply, predict_proba"""
    387         if check_input:
--> 388             X = check_array(X, dtype=DTYPE, accept_sparse="csr")
    389             if issparse(X) and (X.indices.dtype != np.intc or
    390                                 X.indptr.dtype != np.intc):

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74 

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    617             # If input is 1D raise error
    618             if array.ndim == 1:
--> 619                 raise ValueError(
    620                     "Expected 2D array, got 1D array instead:\narr
```

```
ay={}.\n"
    621                  "Reshape your data either using array.reshape
(-1, 1) if "

ValueError: Expected 2D array, got 1D array instead:
array=[234.  23. 456.].
Reshape your data either using array.reshape(-1, 1) if your data has a single fea
ture or array.reshape(1, -1) if it contains a single sample.
```

In [ ]:

```
1
```

# ( 실습 )

1. 영어, 수학, 과학 학습시간으로 합격, 불합격을 예측하시오.
    - 기존데이터
    - 영어(80), 수학(95), 과학(80) --> 합격(1)
    - 영어(67), 수학(88), 과학(75) --> 합격(1)
    - 영어(75), 수학(64), 과학(55) --> 불합격(0)
    - 영어(100), 수학(46), 과학(65) --> 불합격(0)
2. 영어(76), 수학(75), 과학(68) 시간 학습시 합격? 불합격?
3. 알고리즘은 RandomForestClassifier 를 사용.

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [7]:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=123)

X = [[ 80, 95, 80],
     [67, 88, 75],
    [75, 64, 55],
    [100, 46, 65]]
y = [1, 1, 0, 0]  # classes of each sample

clf.fit(X, y)

clf.predict([[76,75,68]])
```

Out[7]:

```
array([0])
```

In [ ]:

```

```

In [ ]:

```

```

# ( Scikit-Learn의 데이터 표현 방식)

Numpy 배열을 이용한 특징 행렬(X), 대상 벡터(y)의 생성

In [33]:

```
import numpy as np

rs = np.random.RandomState(10)

x = 10 * rs.rand(5)
y = 2 * x - 1 * rs.rand(5)
x.shape, y.shape
```

Out[33]:

```
((5,), (5,))
```

In [34]:

```
X = x.reshape(-1,1)
X.shape
```

Out[34]:

```
(5, 1)
```

In [35]:

```
X
```

Out[35]:

```
array([[7.71320643],
       [0.20751949],
       [6.33648235],
       [7.48803883],
       [4.98507012]])
```

In [18]:

```
x
```

Out[18]:

```
array([7.71320643, 0.20751949, 6.33648235, 7.48803883, 4.98507012])
```

In [38]:

```
rs = np.random.RandomState(10)

x = 10 * rs.rand(12)
x
```

Out[38]:

```
array([7.71320643, 0.20751949, 6.33648235, 7.48803883, 4.98507012,
       2.24796646, 1.98062865, 7.60530712, 1.69110837, 0.88339814,
       6.85359818, 9.53393346])
```

In [39]:

```
x.shape
```

Out[39]:

```
(12,)
```

In [45]:

```
x.reshape(4,3)
```

Out[45]:

```
array([[7.71320643, 0.20751949, 6.33648235],
       [7.48803883, 4.98507012, 2.24796646],
       [1.98062865, 7.60530712, 1.69110837],
       [0.88339814, 6.85359818, 9.53393346]])
```

In [52]:

```
X = x.reshape(-1,1)
```

In [53]:

```
X
```

Out[53]:

```
array([[7.71320643],
       [0.20751949],
       [6.33648235],
       [7.48803883],
       [4.98507012],
       [2.24796646],
       [1.98062865],
       [7.60530712],
       [1.69110837],
       [0.88339814],
       [6.85359818],
       [9.53393346]])
```

In [ ]:

```

```

In [55]:

```
x.shape
```

Out[55]:

```
(12,)
```

In [54]:

```
X.shape
```

Out[54]:

```
(12, 1)
```

In [ ]:

```

```

In [ ]:

```

```

In [15]:

```
y
```

Out[15]:

```
array([15.20161622,  0.21697612, 11.91243399, 14.80696681,  9.88180043])
```

In [ ]:

```

```

# ( Scikit-Learn의 데이터 표현 방식)

Pandas DataFrame을 이용한 특징 행렬(X), 대상벡터(y)의 생성

In [8]:

```python
import seaborn as sns
iris = sns.load_dataset("iris")
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [9]:

```python
iris.head()
```

Out[9]:

| | 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 |

In [ ]:

```

```

In [ ]:

```

```

# ( 실습.species 컬럼을 제외한 나머지 컬럼으로 X를 만들기 )

In [16]:

```
X = iris.drop("species",axis=1) ;X.head()
```

Out[16]:

| | 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 |

In [ ]:

```
X = iris.drop("species", axis=1) ;X.head()
```

In [11]:

```
X1 = iris["sepal_length"];X1
```

Out[11]:

```
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
      ...
145    6.7
146    6.3
147    6.5
148    6.2
149    5.9
Name: sepal_length, Length: 150, dtype: float64
```

In [23]:

```
X2 = iris[["sepal_length","sepal_width","petal_length","petal_width"]];X2.head()
```

Out[23]:

| | 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 |

In [28]:

```
iris.iloc[:,0:4].head(3)
```

Out[28]:

| | 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 |

In [ ]:

```

```

In [ ]:

```

```

# ( 실습.species 컬럼만 있는 y 를 만들기 )

In [26]:

```
y = iris["species"]
y.shape
```

Out[26]:

(150,)

In [17]:

```
X.shape
```

Out[17]:

(150, 4)

In [22]:

```
iris.shape
```

Out[22]:

(150, 5)

In [18]:

```
iris.head(3)
```

Out[18]:

| | 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 |

In [22]:

```
iris["species"].unique()
#iris.species
```

Out[22]:

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

In [ ]:

```

```

In [23]:

```
# 컬럼 조회
iris.columns
```

Out[23]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [24]:

```
# 결측치 확인
iris.isnull().sum()
```

Out[24]:

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [ ]:

```

```

In [ ]:

```
 
```

# ( Scikit-Learn의 데이터 표현 방식)

Bunch 객체를 이용한 특징 행렬(X), 대상벡터(y) 의 생성

In [29]:

```
from sklearn.datasets import load_iris

iris = load_iris()
type(iris)
```

Out[29]:

```
sklearn.utils.Bunch
```

In [31]:

```
iris.keys()
```

Out[31]:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'fil
ename'])
```

In [ ]:

```
 
```

In [30]:

```
iris
```

```
s\n    :Attribute Information:\n        - sepal length in cm\n        - sepal wid
th in cm\n        - petal length in cm\n        - petal width in cm\n        - cl
ass:\n                - Iris-Setosa\n                - Iris-Versicolour\n      
- Iris-Virginica\n                \n    :Summary Statistics:\n\n    =============
= ==== ==== ======= ===== ====================\n                    Min  Max   Me
an   SD   Class Correlation\n    ============== ==== ==== ======= ===== ========
============\n    sepal length:   4.3  7.9   5.84   0.83    0.7826\n    sepal wid
th:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:   1.0  6.9   3.76   1.
76    0.9490  (high!)\n    petal width:    0.1  2.5   1.20   0.76    0.9565  (hig
h!)\n    ============== ==== ==== ======= ===== ====================\n\n    :Miss
ing Attribute Values: None\n    :Class Distribution: 33.3% for each of 3 classe
s.\n    :Creator: R.A. Fisher\n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.
nasa.gov)\n    :Date: July, 1988\n\nThe famous Iris database, first used by Sir
R.A. Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s the same
as in R, but not as in the UCI\nMachine Learning Repository, which has two wrong
data points.\n\nThis is perhaps the best known database to be found in the\npatte
rn recognition literature.  Fisher\'s paper is a classic in the field and\nis ref
erenced frequently to this day.  (See Duda & Hart, for example.)  The\ndata set c
ontains 3 classes of 50 instances each, where each class refers to a\ntype of iri
```

In [ ]:

```
1
```

In [32]:

```
iris.feature_names
```

Out[32]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [33]:

```
iris.data[:5]
```

Out[33]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

In [ ]:

```
1
```

In [34]:

```
type(iris.data)
```

Out[34]:

```
numpy.ndarray
```

In [35]:

```
iris.data.shape
```

Out[35]:

```
(150, 4)
```

In [36]:

```
iris.target
```

Out[36]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [37]:

```
type(iris.target)
```

Out[37]:

```
numpy.ndarray
```

In [ ]:

```

```

In [37]:

```
iris.target.shape
```

Out[37]:

```
(150,)
```

In [38]:

```
iris.target_names
```

Out[38]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [44]:

```
X = iris.data
y = iris.target
```

In [45]:

```
X.shape
```

Out[45]:

```
(150, 4)
```

In [46]:

```
type(y)
```

Out[46]:

```
numpy.ndarray
```

In [47]:

```
y.shape
```

Out[47]:

```
(150,)
```

In [ ]:

```

```

In [ ]:

```

```

# (Scikit-Learn Estimator API 기본 활용 절차)

1. 데이터 준비

In [48]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [49]:

```
rs = np.random.RandomState(10)
x = 10 * rs.rand(100)
y = 3 * x + 2 * rs.rand(100)
```

In [54]:

```
x.shape
```

Out[54]:

```
(100,)
```

In [55]:

```
x[:5]
```

Out[55]:

```
array([7.71320643, 0.20751949, 6.33648235, 7.48803883, 4.98507012])
```

In [60]:

```
1 x.reshape(-1,2,5)[:5]
```

Out[60]:

```
array([[[7.71320643, 0.20751949, 6.33648235, 7.48803883, 4.98507012],
        [2.24796646, 1.98062865, 7.60530712, 1.69110837, 0.88339814]],

       [[6.85359818, 9.53393346, 0.03948266, 5.12192263, 8.12620962],
        [6.12526067, 7.21755317, 2.91876068, 9.17774123, 7.14575783]],

       [[5.42544368, 1.42170048, 3.7334076 , 6.74133615, 4.41833174],
        [4.34013993, 6.17766978, 5.13138243, 6.50397182, 6.01038953]],

       [[8.05223197, 5.21647152, 9.08648881, 3.19236089, 0.90459349],
        [3.00700057, 1.13984362, 8.28681326, 0.46896319, 6.26287148]],

       [[5.47586156, 8.19286996, 1.9894754 , 8.56850302, 3.51652639],
        [7.54647692, 2.95961707, 8.8393648 , 3.25511638, 1.65015898]]])
```

In [ ]:

```
1
```

In [ ]:

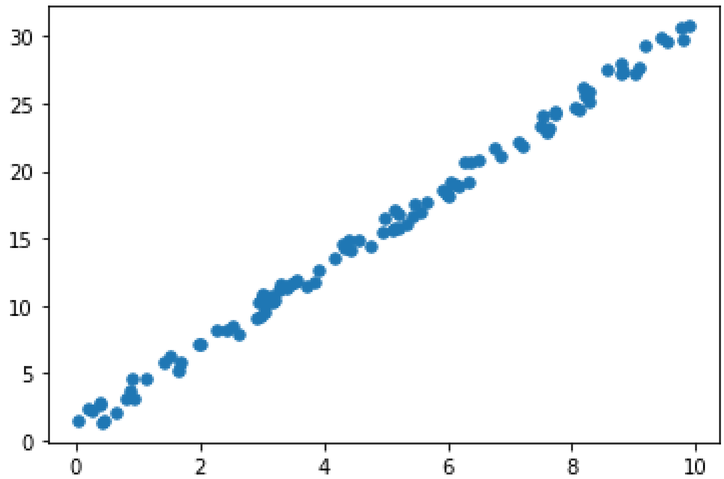
```
1
```

In [51]:

```
1 plt.scatter(x, y, s=30)
```

Out[51]:

```
<matplotlib.collections.PathCollection at 0x2398cd22f70>
```




In [ ]:

```
1
```

2. 모델 클래스 선택
3. 모델 인스턴스 생성과 하이퍼파라미터 선택

In [52]:

```
from sklearn.linear_model import LinearRegression
regr = LinearRegression()
```

In [10]:

```
from sklearn.linear_model import LinearRegression
regr = LinearRegression(fit_intercept=True)
```

In [ ]:

```

```

4. 특징 행렬과 대상벡터 준비

In [53]:

```
X = x.reshape(-1, 1)
X.shape, y.shape
```

Out[53]:

```
((100, 1), (100,))
```

In [ ]:

```

```

5. 모델을 데이터에 적합

In [64]:

```
regr.fit(X,y)
```

Out[64]:

```
LinearRegression()
```

In [63]:

```
regr.get_params()
```

Out[63]:

```
{'copy_X': True, 'fit_intercept': True, 'n_jobs': None, 'normalize': False}
```

In [ ]:

```

```

In [ ]:

```

```

In [65]:

```
regr.coef_
```

Out[65]:

```
array([2.9855087])
```

In [66]:

```
regr.intercept_
```

Out[66]:

```
0.9878534341975644
```

In [ ]:

```

```

6.새로운 데이터를 이용해 예측

In [68]:

```
x_new = np.linspace(-1, 11, num=100)
```

In [69]:

```
x_new[:5]
```

Out[69]:

```
array([-1.        , -0.87878788, -0.75757576, -0.63636364, -0.51515152])
```

In [16]:

```
x_new.shape
```

Out[16]:

```
(100,)
```

In [70]:

```
X_new = x_new.reshape(-1, 1)
X_new.shape
```

Out[70]:

```
(100, 1)
```

In [71]:

```
y_pred = regr.predict(X_new)
```

In [74]:

```
x_new[:5]
```

Out[74]:

```
array([-1.        , -0.87878788, -0.75757576, -0.63636364, -0.51515152])
```

In [72]:

```
y_pred[:5]
```

Out[72]:

```
array([-1.99765526, -1.63577542, -1.27389558, -0.91201574, -0.5501359 ])
```

In [ ]:

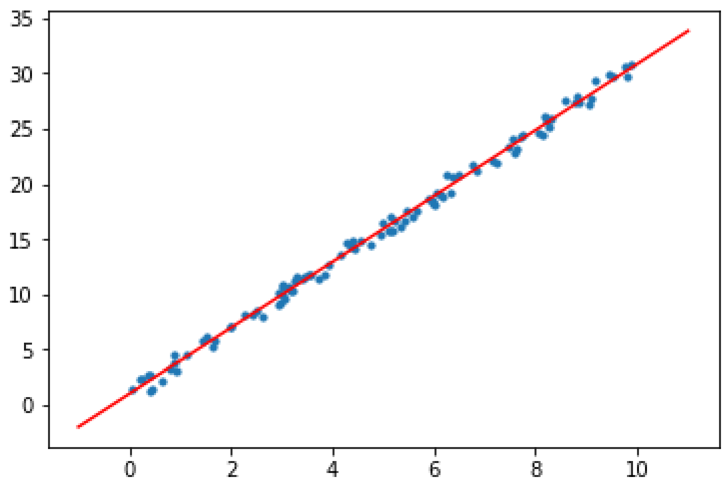
```

```

In [75]:

```
plt.plot(x_new, y_pred, c="red")
plt.scatter(x,y,s=10)
```

Out[75]:

```
<matplotlib.collections.PathCollection at 0x2398e6dab80>
```



In [ ]:

```

```

7. 모델 평가

In [72]:

```
from sklearn.metrics import mean_squared_error

rmse = np.sqrt(mean_squared_error(y, y_pred))
rmse
```

Out[72]:

13.708237122486333

In [ ]:

```

```

# ( 훈련 데이터와 테스트 데이터 )

정확도가 정말 1.0 인가?

In [87]:

```
# 데이터 읽어 오기
from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data
y = iris.target
```

In [89]:

```
X[5]
```

Out[89]:

array([5.4, 3.9, 1.7, 0.4])

In [91]:

```
y[5]
```

Out[91]:

0

In [ ]:

```

```

In [ ]:

```

```

In [81]:

```
# KNN 모델 선택
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [82]:

```
# 모델 훈련(학습)
knn.fit(X, y)
```

Out[82]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [92]:

```
y_pred = knn.predict(X)
```

In [97]:

```
np.mean(y == y_pred)
```

Out[97]:

```
1.0
```

In [94]:

```
y
```

Out[94]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [95]:

```python
y_pred
```

Out[95]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [96]:

```python
# 실제값과 예측결과값 비교
y == y_pred
```

Out[96]:

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

In [ ]:

```python

```

In [ ]:

```python

```

In [ ]:

```python

```

In [99]:

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=25)
```

In [ ]:

```python

```

In [100]:

```python
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[100]:

```
((120, 4), (120,), (30, 4), (30,))
```

In [101]:

```python
X_train[:5]
```

Out[101]:

```
array([[6.5, 2.8, 4.6, 1.5],
       [5.7, 2.5, 5. , 2. ],
       [7.7, 3. , 6.1, 2.3],
       [5. , 3.6, 1.4, 0.2],
       [6.4, 3.2, 5.3, 2.3]])
```

In [ ]:

```python

```

In [102]:

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)
```

Out[102]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [103]:

```python
y_pred = knn.predict(X_test)
y_pred
```

Out[103]:

```
array([0, 2, 2, 1, 2, 1, 2, 0, 1, 1, 0, 0, 0, 1, 0, 1, 2, 2, 1, 1, 1, 1,
       1, 0, 0, 2, 1, 2, 2, 0])
```

In [104]:

```python
np.mean(y_test == y_pred)
```

Out[104]:

```
0.9
```

In [84]:

```
knn.score(X_test, y_test)
```

Out[84]:

0.9

In [85]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[85]:

0.9

In [ ]:

```

```

In [129]:

```
from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data
y = iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=250)
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[129]:

0.9866666666666667

In [ ]:

```

```

In [137]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()

X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=250)
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

knn = KNeighborsClassifier(n_neighbors=2)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

accuracy_score(y_test, y_pred)
```

Out[137]:

0.9333333333333333

In [ ]:

```
1 .. 0.95
2 .. 0.93
```

# ( 하이퍼파라미터의 선택 )

하이퍼라라미터의 선택

In [156]:

```
train_accuracy = []
test_accuracy = []

neighbors = range(1, 30)  # range(1,20,2)

for n in neighbors:
    knn = KNeighborsClassifier(n_neighbors = n)
    knn.fit(X_train, y_train)

    print("n = ", n , " train = ", knn.score(X_train, y_train), " test = ", knn.score(X_test, y

    train_accuracy.append(knn.score(X_train, y_train))
    test_accuracy.append(knn.score(X_test, y_test))
```

```
n =  1  train =  1.0  test =  0.9555555555555556
n =  2  train =  0.9904761904761905  test =  0.9333333333333333
n =  3  train =  0.9714285714285714  test =  0.9555555555555556
n =  4  train =  0.9714285714285714  test =  0.9333333333333333
n =  5  train =  0.9809523809523809  test =  0.9777777777777777
n =  6  train =  0.9714285714285714  test =  0.9555555555555556
n =  7  train =  0.9809523809523809  test =  0.9777777777777777
n =  8  train =  0.9714285714285714  test =  0.9555555555555556
n =  9  train =  0.9714285714285714  test =  0.9555555555555556
n =  10  train =  0.9619047619047619  test =  0.9555555555555556
n =  11  train =  0.9619047619047619  test =  0.9555555555555556
n =  12  train =  0.9619047619047619  test =  0.9555555555555556
n =  13  train =  0.9809523809523809  test =  0.9777777777777777
n =  14  train =  0.9619047619047619  test =  0.9777777777777777
n =  15  train =  0.9809523809523809  test =  0.9777777777777777
n =  16  train =  0.9619047619047619  test =  0.9555555555555556
n =  17  train =  0.9714285714285714  test =  0.9777777777777777
n =  18  train =  0.9619047619047619  test =  0.9333333333333333
n =  19  train =  0.9619047619047619  test =  0.9333333333333333
n =  20  train =  0.9523809523809523  test =  0.9111111111111111
n =  21  train =  0.9523809523809523  test =  0.9111111111111111
n =  22  train =  0.9523809523809523  test =  0.9111111111111111
n =  23  train =  0.9523809523809523  test =  0.9333333333333333
n =  24  train =  0.9428571428571428  test =  0.9333333333333333
n =  25  train =  0.9523809523809523  test =  0.9333333333333333
n =  26  train =  0.9619047619047619  test =  0.9333333333333333
n =  27  train =  0.9523809523809523  test =  0.9333333333333333
n =  28  train =  0.9428571428571428  test =  0.9333333333333333
n =  29  train =  0.9428571428571428  test =  0.9333333333333333
```

In [141]:

```
# neighbors
list(range(1, 11))
```

Out[141]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [ ]:

```

```

In [ ]:

```

```

In [151]:

```
neighbors
```

Out[151]:

```
range(1, 30, 2)
```

In [ ]:

```

```

In [ ]:

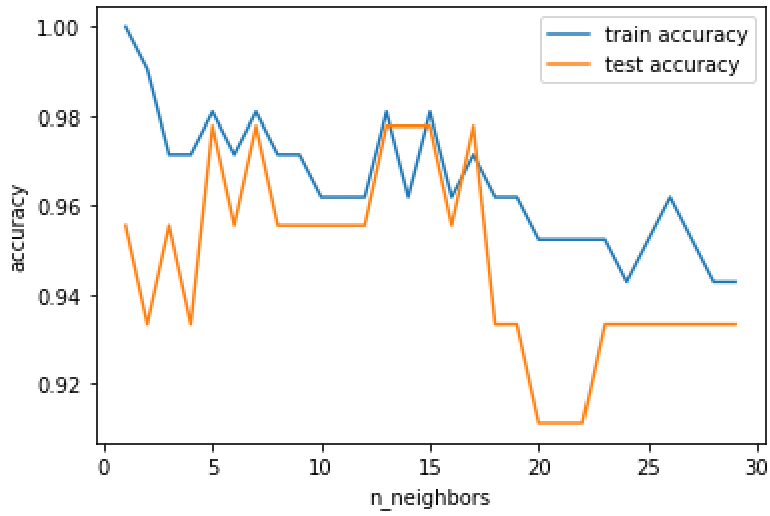
```

```

In [157]:

```
import matplotlib.pyplot as plt
#%matplotlib inline

plt.plot(neighbors, train_accuracy, label="train accuracy")
plt.plot(neighbors, test_accuracy, label="test accuracy")
plt.xlabel("n_neighbors")
plt.ylabel("accuracy")
plt.legend()
```

Out[157]:

```
<matplotlib.legend.Legend at 0x2398e4ce160>
```




In [ ]:

```

```