모두의연구소

Python 및 전처리 교육

Day 2

모두의연구소

데이터 사이언스 프로세스

Part 1

데이터 사이언스 프로세스

데이터 사이언스 프로세스

1. 문제 정의
2. 데이터 정의

3. 데이터 취득
4. 데이터 가공
5. 탐색적 분석과 시각화

6. 모형화
7. 방법론 적용 및 결과 분석

8. 결론 도출

모두의연구소

데이터 사이언스 프로세스 예시(1)

1. 문제 정의
2. 데이터 정의

3. 데이터 취득
4. 데이터 가공
5. 탐색적 분석과 시각화

6. 모형화
7. 방법론 적용 및 결과 분석

8. 결론 도출

예시: 자동 검사(Auto Defect Judge)

1. 문제 정의
   - 수동 검사 자동화

2. 데이터 정의
   - 256 * 256 pixel 이미지를 사용
   - 목표치 미검 0.3%, 과검 10% 이하

   - train data: 정상 4,000장 / 불량 1,000장
   - test data: 정상 800장 / 불량 200장

   현장과 매주 회의를 통해 데이터 노이즈 제거

데이터 사이언스 프로세스 예시(1)

1. 문제 정의
2. 데이터 정의

3. 데이터 취득
4. 데이터 가공
5. 탐색적 분석과 시각화

6. 모형화
7. 방법론 적용 및 결과 분석

8. 결론 도출

4. 데이터 취득
   - FDC 시스템

5. 데이터 가공
   - 이미지 전처리(밝기)

6. 탐색적 분석과 시각화
   - 불량 데이터 종류 확인

7. 모형화
   - 신경망 구조(Resnet 모델)

8. 방법론 적용 및 결과 분석
   - Baseline 모델 성능: 미검 3%, 과검 10%일 때 향후 방향 설정

9. 결론 도출
   - 결과 정리 및 PPT 보고

데이터 사이언스 프로세스 예시(2)

1. 문제 정의
2. 데이터 정의

3. 데이터 취득
4. 데이터 가공
5. 탐색적 분석과 시각화

6. 모형화
7. 방법론 적용 및 결과 분석

8. 결론 도출

예시: 물류 기업 물동량 예측

1. 문제 정의
   - 다음 달 발주량 일별 예측

2. 데이터 정의
   - 과거 거래별 물동량 데이터
   - Table Column: 순번, 일자, 물류 유형, 거래처 코드, 물동량

   - train: 2018년 6월까지 데이터
   - test: 2018년 7월 물동량 예측

데이터 사이언스 프로세스 예시(2)

1. 문제 정의
2. 데이터 정의

3. 데이터 취득
4. 데이터 가공
5. 탐색적 분석과 시각화

6. 모형화
7. 방법론 적용 및 결과 분석

8. 결론 도출

3. 데이터 취득
   - 발주량 데이터베이스
   - 휴일, 강수량 등 공공 데이터

4. 데이터 가공
   - 로그 변환

5. 탐색적 분석과 시각화
   - 물류 업체별 양상 파악 후 세그멘테이션
   - 요일별 세그멘테이션

6. 모형화
   - 시계열 모델(분해법, ARIM, 베이지안 방법론)을 모두 적용 후 성능 비교

7. 방법론 적용 및 결과 분석
   - 요일 별로 물동량 차이 확인
   - 베이지안 방법론이 가장 정확함

8. 결론 도출
   - 결과 정리 및 PPT 보고

데이터 사이언스 프로세스

현실적으로는 위에서 언급한 프로세스에 따라서 진행되지 않는 경우가 많음

- 데이터 수집 시 문제가 생기면 필요한 데이터나 문제 재정의가 필요

- 탐색적 데이터 분석에서 수집된 데이터의 문제가 발견
  > 신규 데이터 수집 및 가설 변경

- 모형화 작업에서 유의미한 결과 도출 실패
  > 이 경우에도 결과 공유

- 모형 결과가 유의미 하지 않은 이유를 알아내기 위해 EDA를 재수행

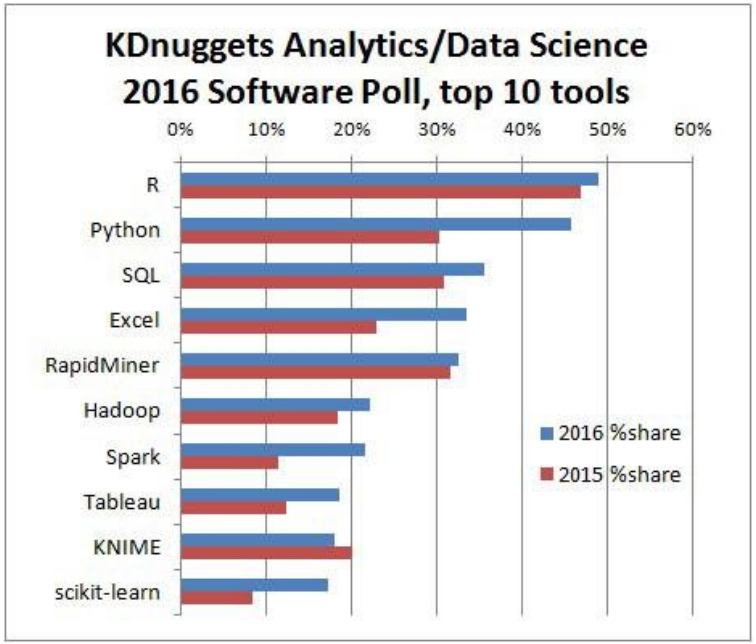
- 일반적으로 모형화 단계에서는 여러 다양한 모형을 시도 필요
  > 모형 결과 자체도 탐색적 데이터 분석 필요 경우 존재

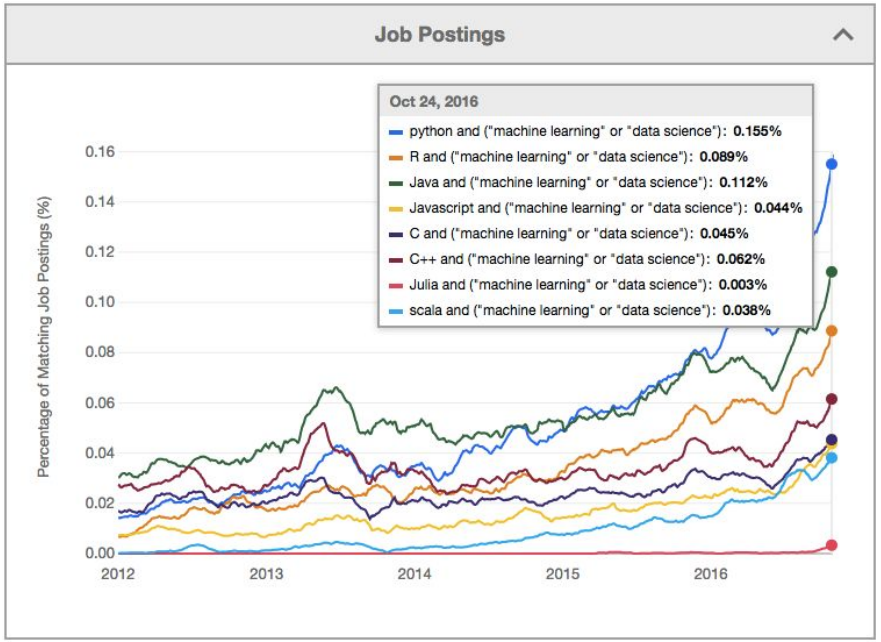
모두의연구소

Python Library Environment

Part 2

Programming language for data science





데이터 사이언스에 주로 사용하는 언어 - Python, R
특히. 딥러닝 = Python

Python 데이터 분석 라이브러리

- numpy: 과학 계산 라이브러리
- pandas: 표 형태의 데이터를 다루기 위해서 구성된 라이브러리
- matplotlib: 데이터 시각화 라이브러리
- scikit-learn: 머신러닝 라이브러리
- tensorflow, keras, pytorch: 딥러닝 라이브러리

Numpy

Numpy는 Numerical Python의 줄임말로 과학계산용 파운데이션 패키지.

- 빠르고 효율적인 다차원 배열 객체 ndarray
- 배열 원소를 다루거나 배열 간의 수학 계산을 수행하는 함수
- 디스크로부터 배열 기반의 데이터를 읽거나 쓸 수 있는 도구
- 선형대수 계산, 푸리에 변환, 난수 발생
- 파이썬과 C, C++, 포트란 코드를 통합하는 도구

Numpy

Numpy는 Numerical Python의 줄임말로 과학계산용 파운데이션 패키지.

1. Numpy

In [1]:
```
import numpy as np
```

In [11]:
```
data = np.random.randn(2,3)
data
```

Out[11]:
```
array([[-0.66697883,  0.33952722,  0.93107587],
       [ 1.13990966,  0.61061782, -0.26233902]])
```

In [12]:
```
data.shape
```

Out[12]:
```
(2, 3)
```

In [13]:
```
data * 10
```

Out[13]:
```
array([[-6.66978835,  3.39527217,  9.31075871],
       [11.39909656,  6.10617819, -2.62339025]])
```

Pandas

Pandas는 구조화된 데이터를 빠르고 쉬우면서도 다양한 형식으로 가공할 수 있는 풍부한 자료 구조와 함수를 제공

Pandas 기능

- 자동적으로 혹은 명시적으로 축의 이름에 따라 데이터를 정렬할 수 있는 자료 구조
- 통합된 시계열 기능
- 시계열 데이터와 비시계열 데이터를 함께 다룰 수 있는 통합 자료 구조
- 누락된 데이터를 유연하게 처리할 수 있는 기능
- SQL 같은 일반 데이터베이스처럼 데이터를 합치고 관계연산을 수행하는 기능

Pandas

Pandas는 구조화된 데이터를 빠르고 쉬우면서도 다양한 형식으로 가공할 수 있는 풍부한 자료 구조와 함수를 제공

2. Pandas

In [3]:
```
import pandas as pd
```

In [6]:
```
data = {'city': ['seoul', 'incheon', 'pusan'],
        'year': [2010, 2011, 2012],
        'pop': [1.5, 1.7, 3.6]}
frame = pd.DataFrame(data)
frame.head()
```

Out[6]:

	city	year	pop
0	seoul	2010	1.5
1	incheon	2011	1.7
2	pusan	2012	3.6

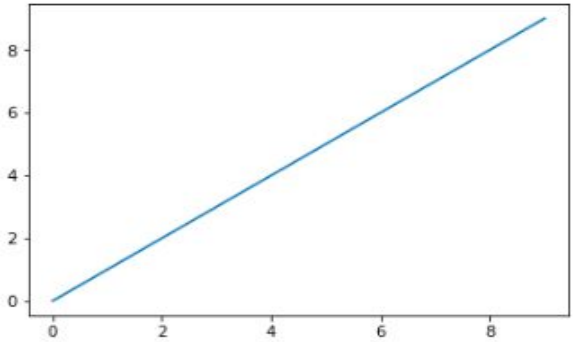
Matplotlib

matplotlib은 그래프나 2차원 데이터 시각화를 생성하는 파이썬 라이브러리

3. Matplotlib

```
In [9]: from matplotlib import pyplot as plt

In [10]: plt.plot(np.arange(10))
Out[10]: [<matplotlib.lines.Line2D at 0x117479080>]
```



Scikit-learn

Python의 대표적인 머신러닝 라이브러리

1) 데이터 전처리 및 특징 선택
2) 알고리즘: 지도 학습 및 비지도 학습
3) 모델 선택 및 평가

머신러닝 파이프라인 기능을 고루 갖추고 있음.

Scikit-learn

모두의연구소

4. Scikit laern

1. Dataset ¶

In [18]:
```
from sklearn.datasets import load_iris
iris_dateset = load_iris()
print('iris_dateset key: \n{}\n'.format(iris_dateset.keys()))
print('iris_dateset data size: \n{}'.format(iris_dateset['data'].shape))
```

```
iris_dateset key:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])

iris_dateset data size:
(150, 4)
```

2. train, test

In [19]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris_dateset['data'],
                                                    iris_dateset['target'],
                                                    random_state=0)
```

Scikit-learn

모두의연구소

3. model and evaluation

In [21]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

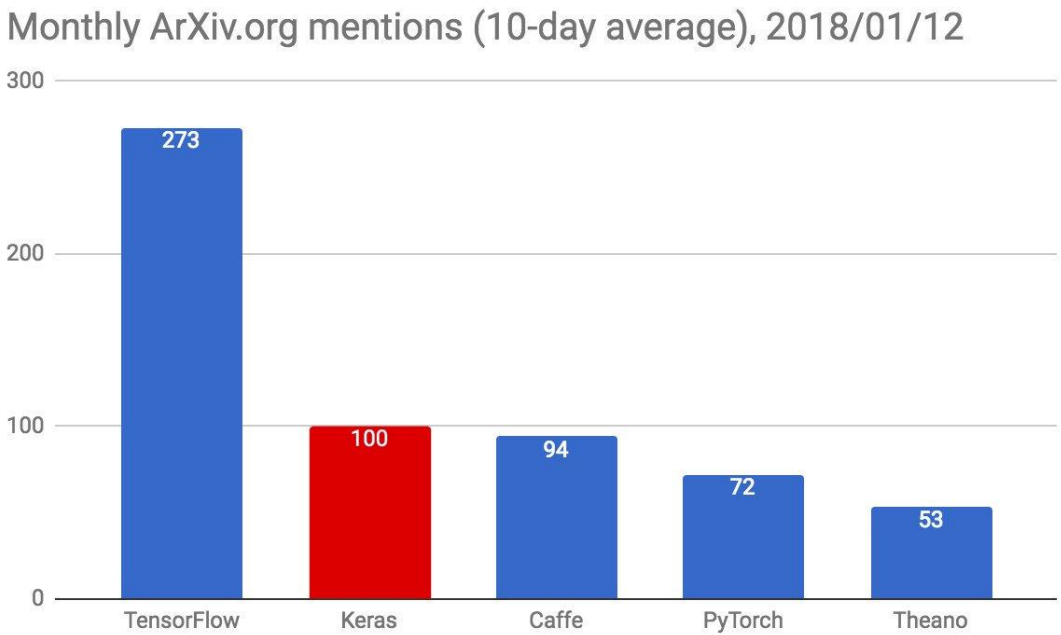
Out[21]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

In [22]:
```
y_pred = knn.predict(X_test)
print("테스트 세트 정확도: {:.2f}".format(np.mean(y_pred==y_test)))
```

```
테스트 세트 정확도: 0.97
```

Tensorflow, Keras, Pytorch

Python 딥러닝 프레임워크



Tensorflow, Keras, Pytorch

모두의연구소

Tensorflow

1) 현재 가장 많은 개발자들이 선호하는 프레임워크 중 하나로 Google에서 만든 딥러닝용 프레임워크.
2) Tensorflow는 기본적으로 C++과 Python을 지원
3) 내부적으로는 전부 C/C++ 엔진으로 되어 있어 Python으로 개발하더라도 속도 차이가 미미

Keras

1) 내부적으로 Theano와 Tensorflow를 바탕으로 하고 있으며 직관적이며 Python API 형태에 가장 가까운 인터페이스를 제공
2) Tensorflow에 통합
3) 진입 장벽이 높은 Tensorflow와는 달리 일반 Python API 형태로 제공하다보니 사용하기가 훨씬 쉬움

Pytorch

1) Torch는 Lua 언어 기반의 API를 제공하는 머신러닝 프레임워크.
2) Facebook에서 Python API 형태의 Pytorch를 오픈 소스로 공개
3) 직관적으로 사용 가능

수업 시간에 다룰 라이브러리 및 예제

- mnist - numpy, keras

https://www.kaggle.com/c/digit-recognizer

- titanic dataset - pandas, matplotlib, scikit-learn

https://www.kaggle.com/c/titanic/data

모두의연구소

모두의연구소

Numpy

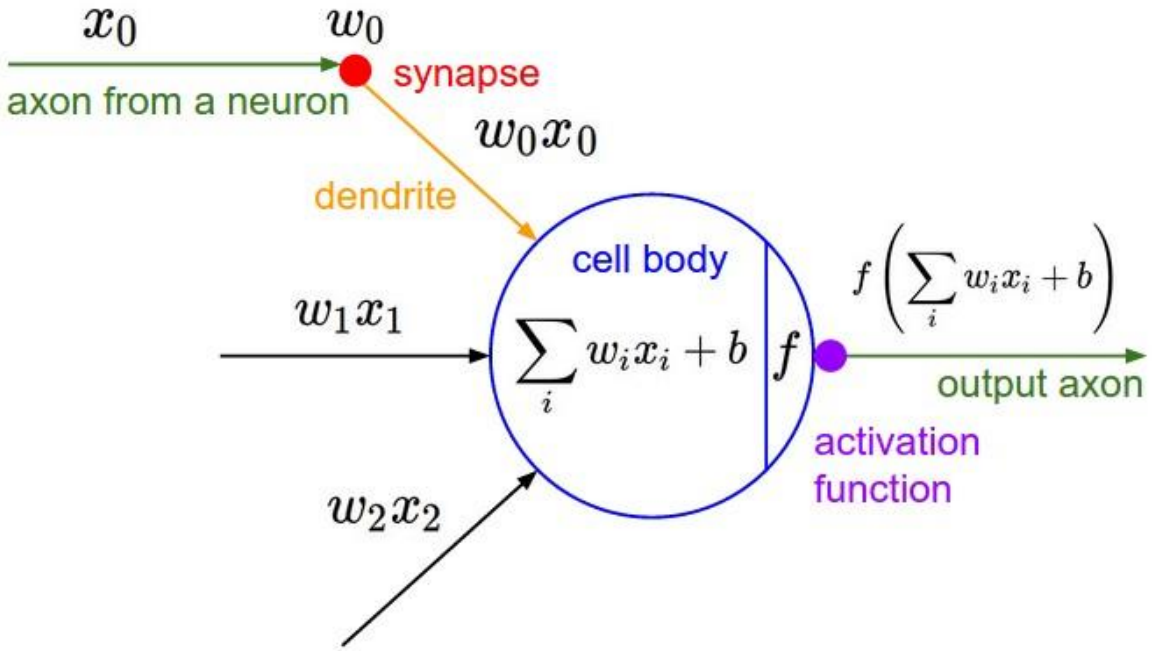
Part 3-1

Numpy

Numpy는 Numerical Python의 줄임말로 과학계산용 파운데이션 패키지.

- 빠르고 효율적인 다차원 배열 객체 ndarray
- 배열 원소를 다루거나 배열 간의 수학 계산을 수행하는 함수
- 디스크로부터 배열 기반의 데이터를 읽거나 쓸 수 있는 도구
- 선형대수 계산, 푸리에 변환, 난수 발생
- 파이썬과 C, C++, 포트란 코드를 통합하는 도구

Numpy 사용 예시



위의 연산을 for 문을 통해서 하는 것이 아니라 배열 연산을 통해서 수행

=> 배열 위주의 프로그래밍과 생각하는 방법을 능숙해지는 것이 중요

Numpy 사용 예시

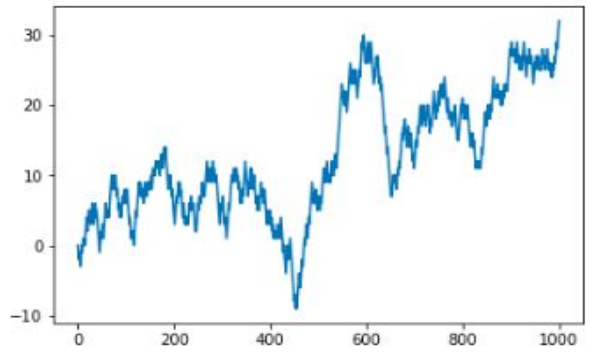
1. 계단 오르내리기 예제 - Python

In [3]:
```
position = 0
walk = [position]
steps = 1000

for i in range(steps):
    step = 1 if random.randint(0, 1) else -1
    position += step
    walk.append(position)
```

In [4]:
```
plt.plot(walk)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x11124bf28>]
```



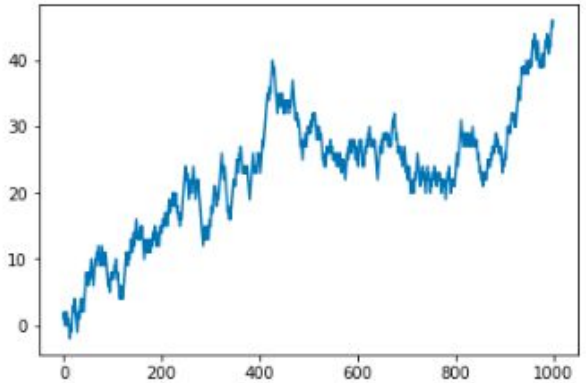
2. 계단 오르내리기 예제 - Numpy

In [6]:
```
import numpy as np

nsteps = 1000
draws = np.random.randint(0, 2, size=nsteps)
steps = np.where(draws > 0, 1, -1)
walk = steps.cumsum()
```

In [7]:
```
plt.plot(walk)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x11137b2b0>]
```



Numpy 사용 예시

3. 계단 오르내리기 예제 5000번 - Numpy

In [17]:
```
nwalks = 5000
nsteps = 1000
draws = np.random.randint(0, 2, size=(nwalks, nsteps))
steps = np.where(draws > 0, 1, -1)
walks = steps.cumsum(1)
print(walks.shape)
walks
```

```
(5000, 1000)
```

Out[17]:
```
array([[ -1,   0,  -1, ...,  -8,  -9, -10],
       [  1,   2,   1, ...,  46,  47,  46],
       [  1,   0,   1, ..., -18, -19, -20],
       ...,
       [ -1,  -2,  -1, ..., -30, -31, -30],
       [  1,   0,  -1, ..., -30, -31, -32],
       [ -1,   0,  -1, ...,  40,  39,  40]])
```

1. Numpy 기초

모두의연구소

1-1. Ndarray

Numpy ¶

```
In [7]: import numpy as np
```

1. Numpy ndarry: 다차원 배열 객체

- Numpy 핵심 기능 중 하나는 N차원 배열 객체 또는 ndarray로 파이썬에서 사용할 수 있는 대규모 데이터 집합을 담을 수 있는 빠르고 유연한 자료 구조다.

```
In [8]: data1 = [5, 6, 7, 8]

In [9]: arr1 = np.array(data1)

In [10]: arr1
Out[10]: array([5, 6, 7, 8])

In [11]: data2 = [[1,2,3,4],[5,6,7,8]]

In [12]: arr2 = np.array(data2)

In [13]: arr2
Out[13]: array([[1, 2, 3, 4],
                [5, 6, 7, 8]])
```

1-2. 자료형

모두의연구소

2. ndarray 자료형

In [65]:
```
arr1 = np.array([1,2,3], dtype=np.float64)
arr2 = np.array([1,2,3], dtype=np.int32)
```

- int8, uint8: 부호가 있는 8비트 정수형과 부호가 없는 8비트 정수형
- int16, uint16: 부호가 있는 16비트 정수형과 부호가 없는 16비트 정수형
- int32, uint32: 부호가 있는 32비트 정수형과 부호가 없는 32비트 정수형
- int64, uint64: 부호가 있는 64비트 정수형과 부호가 없는 64비트 정수형

- float16: 반 정밀도 부동소수점
- float32: 단 정밀도 부동소수점(C 언어의 float)
- float64: 배 정밀도 부동소수점(C 언의 double, 파이썬의 float)
- float128: 확장 정밀도 부동소수점

- complex64: 각각 2개의 32 비트 부동소수점 형을 가지는 복소수
- complex128: 각각 2개의 64 비트 부동소수점 형을 가지는 복소수
- compext256: 각각 2개의 128 비트 부동소수점 형을 가지는 복소수

- bool: True, False 값을 저장하는 불리언 형
- object: 파이썬 객체형
- string: 고정 길이 문자열형(각 글자는 1바이트)
- unicode_: 고정 길이 유니코드 형

1-3. 배열과 스칼라 값의 연산

모두의연구소

3. 배열과 스칼라 값의 연산

- 배열은 for 반복문을 작성하지 않고 데이터를 일괄처리할 수 있기 때문에 중요하다. 이를 벡터화라 하는데 같은 크기의 산술연산은 각 요소 단위로 적용된다.

```
In [66]: arr = np.array([[1.,2.,3.],[4.,5.,6.]])
         arr

Out[66]: array([[1., 2., 3.],
                [4., 5., 6.]])

In [67]: arr * arr

Out[67]: array([[ 1.,  4.,  9.],
                [16., 25., 36.]])

In [68]: arr - arr

Out[68]: array([[0., 0., 0.],
                [0., 0., 0.]])

In [69]: 1 / arr

Out[69]: array([[1.        , 0.5       , 0.33333333],
                [0.25      , 0.2       , 0.16666667]])

In [70]: arr ** 0.5

Out[70]: array([[1.        , 1.41421356, 1.73205081],
                [2.        , 2.23606798, 2.44948974]])
```

1-4. 색인과 슬라이싱

모두의연구소

4. 색인과 슬라이싱 기초

```
In [91]: arr = np.arange(10)

In [92]: arr
Out[92]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [93]: arr[5]
Out[93]: 5

In [94]: arr[5:8]
Out[94]: array([5, 6, 7])

In [95]: arr[5:8] = 12

In [96]: arr
Out[96]: array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

리스트와 중요한 차이점은 배열 조건은 원본 배열의 뷰이다. 즉, 데이터는 복사되지 않고 뷰에 대한 변경은 그대로 원본 배열에 반영된다.

2. Numpy 연산

모두의연구소

2-1. 유니버셜 함수 (1)

모두의연구소

1. Universal 함수

ufunc라고 불리는 유니버셜 함수는 ndarray 안에 있는 데이터 원소별로 연산을 수행하는 함수이다. 유니버셜 함수는 하나 이상의 스칼라 값을 받아서 하나 이상의 스칼라 결과 값을 반환하는 간단한 함수를 고속으로 수행할 수 있는 백터화된 래퍼 함수라고 생각하면 된다.

In [6]:
```
arr = np.arange(10)
arr
```

Out[6]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [7]:
```
np.sqrt(arr)
```

Out[7]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [8]:
```
np.exp(arr)
```

Out[8]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [9]:
```
x = np.random.randn(8)
y = np.random.randn(8)
```

In [11]:
```
np.maximum(x, y) # element wise operation
```

Out[11]:
```
array([-1.4423063 ,  0.15591954,  0.35655707,  1.9711139 ,  1.33431801,
        0.55076497,  1.65486003,  0.37015179])
```

2-2. 배열 연산(1) - 조건절

모두의연구소

1. 조건절 표현하기

In [12]:
```
xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
cond = np.array([True, False, True, True, False])
```

In [13]:
```
result = np.where(cond, xarr, yarr)
```

In [14]:
```
result
```

Out[14]:
```
array([1.1, 2.2, 1.3, 1.4, 2.5])
```

2-2. 배열 연산(2) - 통계 함수

모두의연구소

2. 수학 메서드와 통계 메서드

배열 전체 혹은 배열에서 한 축에 따르는 자료에 대한 통계를 계산하기 위한 수학 함수는 배열 메서드로 사용할 수 있다. 전체의 합이나 평균, 표준편차는 Numpy의 최상위 함수를 이용하거나, 배열의 인스턴스 메서드를 사용해서 구할 수 있다.

In [25]:
```
arr = np.random.rand(5, 4)
arr
```

Out[25]:
```
array([[0.07095817, 0.03497945, 0.25071581, 0.75634137],
       [0.72957202, 0.4753495 , 0.14654329, 0.05921105],
       [0.03028327, 0.3033298 , 0.41250839, 0.31910685],
       [0.60507106, 0.84190856, 0.86355104, 0.05448019],
       [0.99214452, 0.6498598 , 0.52063079, 0.17246049]])
```

In [26]:
```
arr.mean()
```

Out[26]:
```
0.41445027069939266
```

In [27]:
```
np.mean(arr)
```

Out[27]:
```
0.41445027069939266
```

2-2. 배열 연산(3) - 정렬

3. 정렬

- np.sort 메서드는 배열을 직접 변경하지 않고 정렬된 결과를 가지고 있는 복사본을 반환한다.

```
In [37]: arr = np.random.randn(8)
         arr
Out[37]: array([-0.72457977,  0.87587823,  1.2289314 , -0.28894393,  0.15205621,
                 0.39222443, -1.86491412, -0.43901666])

In [39]: arr.sort()
         arr
Out[39]: array([-1.86491412, -0.72457977, -0.43901666, -0.28894393,  0.15205621,
                 0.39222443,  0.87587823,  1.2289314 ])

In [40]: arr = np.random.randn(5, 3)
         arr
Out[40]: array([[ 1.6184662 , -1.08623178, -0.94366897],
                [-0.48770058,  0.19617067, -1.26855797],
                [-0.43161619, -0.80774082,  1.70783631],
                [-1.05988322, -0.69291836, -0.12831549],
                [-0.2984775 ,  0.07530742, -1.90711257]])

In [42]: arr.sort(1)
         arr
Out[42]: array([[-1.08623178, -0.94366897,  1.6184662 ],
                [-1.26855797, -0.48770058,  0.19617067],
                [-0.80774082, -0.43161619,  1.70783631],
                [-1.05988322, -0.69291836, -0.12831549],
                [-1.90711257, -0.2984775 ,  0.07530742]])
```

2-4. 선형대수

모두의연구소

3. 선형대수

내적은 중요하다.

In [43]:
```
x = np.array([[1.,2.,3.], [4.,5.,6.]])
y = np.array([[6.,23.], [-1, 7], [8, 9]])
```

In [44]:
```
x.dot(y)
```

Out[44]:
```
array([[ 28.,  64.],
       [ 67., 181.]])
```

In [45]:
```
from numpy.linalg import inv, qr
```

In [46]:
```
X = np.random.randn(5, 5)
```

In [47]:
```
mat = X.T.dot(X)
```

In [48]:
```
inv(mat)
```

Out[48]:
```
array([[ 1.10207231, -0.3894995 , -0.23120404,  1.494553  ,  0.61408901],
       [-0.3894995 ,  1.12159273,  0.11062727, -2.89157254, -0.89253246],
       [-0.23120404,  0.11062727,  0.32050522,  0.09357513,  0.24540058],
       [ 1.494553  , -2.89157254,  0.09357513, 11.2476811 ,  4.17382102],
       [ 0.61408901, -0.89253246,  0.24540058,  4.17382102,  1.95596743]])
```

3. Numpy 고급

모두의연구소

3-1. 배열 재형성하기

모두의연구소

1. 배열 재형성하기

In [2]:
```
arr = np.arange(8)
arr
```

Out[2]:
```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

In [3]:
```
arr.reshape((4,2))
```

Out[3]:
```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

In [4]:
```
arr.reshape((4,2)).reshape((2,4))
```

Out[4]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [6]:
```
arr = np.arange(15).reshape((5, -1))
arr
```

Out[6]:
```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [12, 13, 14]])
```

- reshape에 넘기는 값 중 하나는 -1이 될 수도 있는데, 이 경우에는 원본 데이터를 참조해서 적절한 값을 추론하게 됩니다.

3-2. 배열 이어나누고 붙이기

모두의연구소

2. 배열 이어붙이고 나누기

- concatenate: axis에 따라서 하나의 배열로 합친다.

```
In [11]: arr1 = np.array([[1,2,3], [4,5,6]])
         arr2 = np.array([[7,8,9], [10,11,12]])

In [12]: np.concatenate([arr1, arr2], axis=0)

Out[12]: array([[ 1,  2,  3],
                [ 4,  5,  6],
                [ 7,  8,  9],
                [10, 11, 12]])

In [13]: np.concatenate([arr1, arr2], axis=1)

Out[13]: array([[ 1,  2,  3,  7,  8,  9],
                [ 4,  5,  6, 10, 11, 12]])
```

- vstack: vertical
- hstack: horizontal

```
In [14]: np.vstack((arr1, arr2))

Out[14]: array([[ 1,  2,  3],
                [ 4,  5,  6],
                [ 7,  8,  9],
                [10, 11, 12]])

In [15]: np.hstack((arr1, arr2))

Out[15]: array([[ 1,  2,  3,  7,  8,  9],
                [ 4,  5,  6, 10, 11, 12]])
```

3-3. 브로드캐스팅

모두의연구소

3. 브로드캐스팅

- 브로드캐스팅은 다른 모양의 배열 간 산술연산을 어떻게 수행해야 하는지 설명한다. 이는 매우 강력한 기능이다.

In [18]:
```
arr = np.arange(5)
arr
```

Out[18]:
```
array([0, 1, 2, 3, 4])
```

In [19]:
```
arr * 4
```

Out[19]:
```
array([ 0,  4,  8, 12, 16])
```

In [22]:
```
arr = np.random.randn(4, 3)
arr
```

Out[22]:
```
array([[ 1.56404053,  0.09956201,  0.31560042],
       [-1.41929603,  0.98733789, -1.1162068 ],
       [-0.58501167, -0.75729012, -0.23114476],
       [-1.47913919,  0.78770946, -0.7313576 ]])
```

In [26]:
```
arr.mean(axis=0)
```

Out[26]:
```
array([-0.47985159,  0.27932981, -0.44077718])
```

In [28]:
```
demenad = arr - arr.mean(0)
demenad
```

Out[28]:
```
array([[ 2.04389212, -0.1797678 ,  0.7563776 ],
       [-0.93944444,  0.70800808, -0.67542962],
       [-0.10516008, -1.03661993,  0.20963243],
       [-0.9992876 ,  0.50837965, -0.29058041]])
```

모두의연구소

데이터 사이언스 실습(mnist)

Part 3-2

모두의연구소

Pandas / Matplotlib / Scikit-learn

Part 4-1

Pandas

Pandas는 구조화된 데이터를 빠르고 쉬우면서도 다양한 형식으로 가공할 수 있는 풍부한 자료 구조와 함수를 제공

Pandas 기능

- 자동적으로 혹은 명시적으로 축의 이름에 따라 데이터를 정렬할 수 있는 자료 구조
- 통합된 시계열 기능
- 시계열 데이터와 비시계열 데이터를 함께 다룰 수 있는 통합 자료 구조
- 누락된 데이터를 유연하게 처리할 수 있는 기능
- SQL 같은 일반 데이터베이스처럼 데이터를 합치고 관계연산을 수행하는 기능

1. Pandas 기본

모두의연구소

1. 자료 구조 : Series, DataFrame

모두의연구소

step01_Series.py

```
'''
    - Series / DataFrame 두가지가 가장 대표적인 자료구조
     - DataFrame 특정 칼럼 추출 =Series

Series 모듈 객체 특징
    리스트랑 같은 1차원 구조지만
        1. 세로로 형성
        2.index 자동생성
        3.내맘대로 컬럼명 넣어서 접근/수정 가능.
        4. numpy 와의 공통점
            -> 수학/통계 함수(메소드) 이용가능
            -> 범위 수정 가능
            -> 블럭 연산 가능
            -> index/slicing 가능
            -> DF 칼럼 구성

'''
```

step02_DataFrame.py

```
'''
DataFrame 객체 특징
    - 2차원 행렬구조
    - table 구조와 유사
    - 컬럼 단위로 다른값 타입 가능
        (  DF 칼럼 구성가능객체/타입 : numpy, Series, list, dict  )

    - 장점 : 요약통계량 함수(메소드) 사용가능
'''
```

1. 자료 구조: Series, DataFrame

```
# 1.1 specify values for each column

df = pd.DataFrame(
    {"a" : [4,5,6],
     "b" : [7,8,9],
    "c" : [10,11,12]},
    index = [1,2,3]
                        )
print (df)
```

```
   a  b   c
1  4  7  10
2  5  8  11
3  6  9  12
```

```
# 2. Specify values for each row)

df = pd.DataFrame(
    [
        [4,5,10],
        [5,5,11],
        [16,9,12]
    ],
    index = ['x','y','z'],
    columns = ['a','b','c']
) ; print(df)

```

```
    a  b   c
x   4  5  10
y   5  5  11
z  16  9  12
```

2. 데이터 접근(1)

```
1  # 특정 열 (하나) 선택하기
2  print( df11['width'] ) #or df.width
3
4  # 특정 열 (여러개) 선택하기
5  print( df11[ ['width','length','species'] ] )
```

```
n v
d 1   4
  2   5
e 2   6
Name: width, dtype: int64
    width  length  species
n v
d 1    4     7     10
  2    5     8     11
e 2    6     9     12
```

2. 데이터 접근(2)

```
# 컬럼명으로 특정범위 컬럼 나타내기

# b~c컬럼만 보여라
df.loc[:,'b':'c']
```

		b	c
n	v		
d	1	7	10
	2	8	11
e	2	9	12

```
# 1~2번째 컬럼만 보여라
# (첫 컬럼의 인덱스는 0)
df.iloc[:,[1,2]]
```

		b	c
n	v		
d	1	7	10
	2	8	11
e	2	9	12

```
# a,c컬럼에서 4 초과인 데이터만 보여라.
df.loc[ df['a'] > 4, ['a','c']]
```

		a	c
n	v		
d	2	5	11
e	2	6	12

3. 특정 데이터 변형 및 삭제

```
1  # .rename : 컬럼명변경
2  df6p = df6.rename(columns = {'foo':'poo'}) ; print(df6,'\n\n', df6p)
```

```
  foo bar  baz zoo
5  two   C   6   t
4  two   B   5   w
3  two   A   4   q
2  one   C   3   z
1  one   B   2   y
0  one   A   1   x

   poo bar  baz zoo
5  two   C   6   t
4  two   B   5   w
3  two   A   4   q
2  one   C   3   z
1  one   B   2   y
0  one   A   1   x
```

```
1  # (여러)행 삭제
2  print(df6.drop(columns=['foo','baz'])) # 열삭제
3  print(df6.drop([4,3], axis=0)) # 행삭제
```

```
  bar zoo
5   C   t
4   B   w
3   A   q
2   C   z
1   B   y
0   A   x
   foo bar  baz zoo
5  two   C   6   t
2  one   C   3   z
1  one   B   2   y
0  one   A   1   x
```

2. Pandas - 기본 데이터 분석

모두의연구소

1. 데이터 정렬

```
# .sort : 특정 컬럼의 값들을 기준으로 오름/내림차순 정렬
df5 = df_p.sort_values('zoo') ; df5 # zoo 컬럼기준 오름차순
```

	foo	bar	baz	zoo
3	two	A	4	q
5	two	C	6	t
4	two	B	5	w
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z

```
df6 = df_p.sort_values('baz', ascending=False) ; df5 #  baz 컬럼기준 내림차순
```

	foo	bar	baz	zoo
3	two	A	4	q
5	two	C	6	t
4	two	B	5	w
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z

2. 데이터 살펴보기(1)

```
1  # 기초통계량 1
2  # count, mean, std, 4분위수, 최대값,
3  # rangeIndex, 각 자료타입, 데이터타입,
4  # 메모리 사용량
5  df.describe()
```

	a	b	c
count	3.0	3.0	3.0
mean	5.0	8.0	11.0
std	1.0	1.0	1.0
min	4.0	7.0	10.0
25%	4.5	7.5	10.5
50%	5.0	8.0	11.0
75%	5.5	8.5	11.5
max	6.0	9.0	12.0

```
1  # 기초통계량 2
2  # 자료타입, 행 개수, 열 개수,
3  # 각 컬럼별 변수 타입, 메모리 사용량
4  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 3 entries, (d, 1) to (e, 2)
Data columns (total 3 columns):
a    3 non-null int64
b    3 non-null int64
c    3 non-null int64
dtypes: int64(3)
memory usage: 306.0+ bytes
```

2. 데이터 살펴보기(2)

```
1  # 이 데이터프레임의 행은 몇개인가?
2  len (df)
```

```
3
```

```
1  # a란 이름을 가진 컬럼의 유니크한 값은 총 몇개있는가?
2  df ['a'].nunique()
```

```
3
```

```
1  # a란 이름을 가진 컬럼의 유니크한 값은 무엇이며, 각각 몇개씩 있는가?
2  df ['a'].value_counts()
```

```
6   1
5   1
4   1
Name: a, dtype: int64
```

3. 결측치 처리

```
# .fillna   =>   NA 처리
# 1. (전체) 평균으로 대체
fruit_tot = fruit2 + fruit3 ; fruit_tot # 대체 전
```

```
apple     7000.0
banana       NaN
kiwi      4000.0
mango     7000.0
orange       NaN
dtype: float64
```

```
fruit_tot_NA_mean = fruit_tot.fillna(fruit_tot.mean())
fruit_tot_NA_mean  #대체 후
```

```
apple     7000.0
banana    6000.0
kiwi      4000.0
mango     7000.0
orange    6000.0
dtype: float64
```

```
# 2. 특정숫자로 대체
fruit_tot_NA_0 = fruit_tot.fillna(0) # 0으로 대체
fruit_tot_NA_0
```

```
apple     7000.0
banana       0.0
kiwi      4000.0
mango     7000.0
orange       0.0
dtype: float64
```

3. Pandas - 그룹 함수

모두의연구소

1. Merge (Join) - left/right join

모두의연구소

```
# 1. Left Join
    # "왼쪽"에 있는 DF인 adf의 기준컬럼(x1)의 값들에만 의존 ; bdf의 x1엔 있지만 adf의 x1에 없으면 누락됨
pd.merge(adf,bdf, how='left', on = 'x1')
```

	x1	x2	x3
0	A	1	T
1	B	2	F
2	C	3	NaN

```
# 2. Right Join
# "오른쪽"에 있는 df인 bdf의 기준컬럼(x1)의 값들에만 의존 = adf의 x1엔 있지만 bdf의 x1에 없으면 누락됨
pd.merge(adf,bdf, how='right', on = 'x1')
```

	x1	x2	x3
0	A	1.0	T
1	B	2.0	F
2	D	NaN	T

```
tmp = { 'x1' : ['A','B','C'], 'x2' : [1,2,3] }
adf = pd.DataFrame(tmp, columns=['x1','x2'])
```

	x1	x2
0	A	1
1	B	2
2	C	3

```
tmp2 = { 'x1' : ['A','B','D'], 'x3' : ['T','F','T'] }
bdf = pd.DataFrame(tmp2, columns=['x1','x3'])
```

	x1	x3
0	A	T
1	B	F
2	D	T

1. Merge (Join) - inner/outer join

모두의연구소

```
# 3. Inner Join
# adf, bdf 양쪽 모두의 기준컬럼에 존재해야함
pd.merge(adf,bdf, how='inner', on = 'x1')
```

	x1	x2	x3
0	A	1	T
1	B	2	F

```
# 4. Outer Join
# adf, bdf 의기준컬럼중 둘 중 하나에만 존재해도됨
pd.merge(adf,bdf, how='outer', on = 'x1')
```

	x1	x2	x3
0	A	1.0	T
1	B	2.0	F
2	C	3.0	NaN
3	D	NaN	T

```
tmp = { 'x1' : ['A','B','C'], 'x2' : [1,2,3] }
adf = pd.DataFrame(tmp, columns=['x1','x2'])
```

	x1	x2
0	A	1
1	B	2
2	C	3

```
tmp2 = { 'x1' : ['A','B','D'], 'x3' : ['T','F','T'] }
bdf = pd.DataFrame(tmp2, columns=['x1','x3'])
```

	x1	x3
0	A	T
1	B	F
2	D	T

1. Merge (Join) - concat

```
1  s1 = Series([1,3], index=['a','b'])
2  s2 = Series([5,6,7], index=['a','b','c'])
3  s3 = Series([11,13], index=['a','b'])
4  print(s1, "\n",s2, "\n",s3)
```

```
a   1
b   3
dtype: int64
 a   5
b   6
c   7
dtype: int64
 a   11
b   13
dtype: int64
```

```
1  # 5.4.1 행 단위 결합  (=다음 df가 아래에 붙음) .concat
2  pd.concat([s1, s2, s3], axis = 0)
```

```
a    1
b    3
a    5
b    6
c    7
a   11
b   13
dtype: int64
```

```
1  # 5.4.2 열 단위 결합 : DF 화
2  pd.concat([s1, s2, s3], axis = 1)
```

	0	1	2
a	1.0	5	11.0
b	3.0	6	13.0
c	NaN	7	NaN

2. Groupby

```
# 칼럼으로 그룹화 / 그룹연산
    # 그룹연산 = 같은 컬럼값끼리 다양한 연산
    # 예) key 컬럼에 a,a,b,a,b 값 존재 : -> a끼리의 합 or b끼리 평균
data_df # 그룹화 전 data_df
```

	key	col1	col2
0	a	10	100
1	a	10	100
2	b	20	200
3	a	10	100
4	b	20	200
5	b	15	140

```
group_obj = data_df.groupby('key' )  # "key"컬럼으로 그룹화한 그룹 객체 생성
group_obj # 출력해도 내용은 나오지않는다. 어떻게 활용할까?
```

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x11cffdac8>
```

```
# Groupby 활용 예
# 1. 합계
group_obj.sum
```

```
key
a  3
b  3
dtype: int64
```

```
# 2.평균
group_obj.mean()
```

key	col1	col2
a	10.000000	100.0
b	18.333333	180.0

```
# 3. 그룹사이즈 (종류별 원소갯수)
group_obj.size()
```

```
key
a  3
b  3
dtype: int64
```

Matplotlib

모두의연구소

matplotlib은 그래프나 2차원 데이터 시각화를 생성하는 파이썬 라이브러리이다. 출판물 수준의 도표를 만들 수 있도록 설계되었다. matplotlib은 3D 도식을 위한 matplot3d 및 지도 투영을 위한 basemap과 같은 다양한 확장 툴킷이 있다

- matplotlib: 데이터 시각화 라이브러리

- Plot 그리는 법
  - 1변수 범주형 - bar
  - 1변수 연속형 - histogram
  - 2변수 범주 / 연속 - boxplot
  - 2변수 연속 / 연속 - scatterplot

1. Matplotlib 시작하기

모두의연구소

파이썬의 시각화 도구인 Matplotlib은 NumPy 배열을 기반으로 만들어진 다중 플랫폼 데이터 시각화 라이브러리

In [2]:
```
import matplotlib
import matplotlib.pyplot as plt
```

- Matplotlib의 가장 중요한 특징 중 하나는 다양한 운영체제와 그래픽 백엔드에서 잘 동작한다는 점
- 아쉬운 부분은 인터페이스와 스타일이 투박하고 구식으로 느껴질 수 있음

> 최근 D3js 혹은 HTML5 캔버스 기반 웹 시각화 툴킷 등 새로운 도구들이 많음
>
> 대안으로 스타일 설정을 허용함

스타일 설정하기

In [3]:
```
plt.style.use('classic')
```

IPython 노트북에서 플로팅하기

- %matplotlib notebook 은 노트북 내에서 **대화형** 플롯을 삽입할 수 있음
- %matplotlib inline은 노트북에 **정적** 이미지를 삽입할 수 있음

In [4]:
```
%matplotlib inline
```

1-1. Matplotlib 인터페이스(매트랩 인터페이스)

Matplotlib은 매트랩 스타일의 상태 기반 인터페이스와 더 강력한 객체 지향 인터페이스라는 두 개의 인터페이스를 제공

1. 매트랩 스타일의 인터페이스

> 매트랩 스타일 도구는 파이플롯(pyplot, plt) 인터페이스에 포함돼 있음

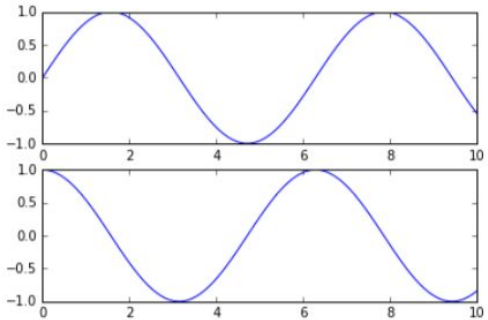
In [6]:
```
import numpy as np
x = np.linspace(0, 10, 100)

plt.figure() # 플롯 그림을 생성

# 두 개의 패널 중 첫 번째 패널을 생성하고 현재 축(axis)을 설정
plt.subplot(2,1,1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# 두 번째 패널을 생성하고 현재 축(axis)을 설정
plt.subplot(2,1,2)
plt.plot(x, np.cos(x))
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x16baf8aabe0>]
```



1-2. Matplotlib 인터페이스(객체지향 인터페이스)

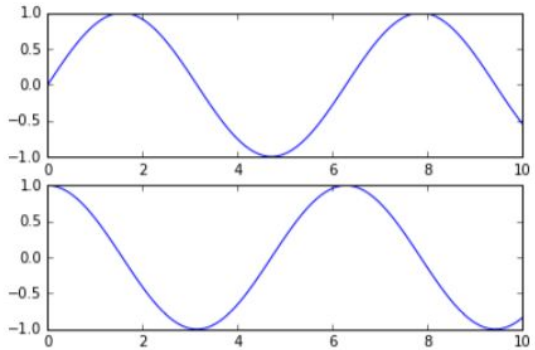
2. *객체지향 인터페이스*

> 객체지향 인터페이스에서 플로팅 함수는 '활성화된' 그림이나 축의 개념에 의존하지 않는 명시적인 Figure와 Axes 객체의 메소드(Method)

In [7]:
```
# 플롯 그리드를 생성
# ax는 두개의 축 객체의 배열이 됨
fig, ax = plt.subplots(2)

# 적절한 객체에서 plot() 메소드를 호출
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x))
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x16baf97ba58>]
```



간단한 플롯의 경우, 어떤 스타일을 사용할지 선택하는 것은 대체로 취향의 문제이지만, 플롯이 복잡해질수록 객체지향 방식이 채택될 것

1-3. Matplotlib 여러개의 그래프 그리기

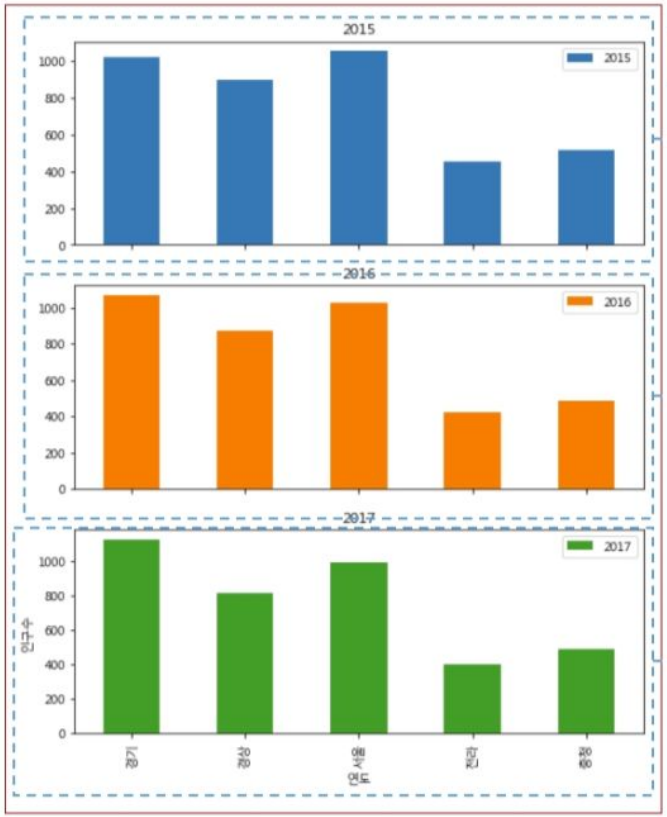
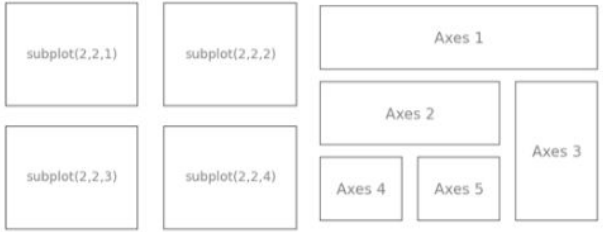


Figure : 그래프가 그려지는 캔버스

Subplot (axes): figure 안에 있는 각각의 그래프
- 그리드 형태 → subplot
- 임의의 형태 → axes

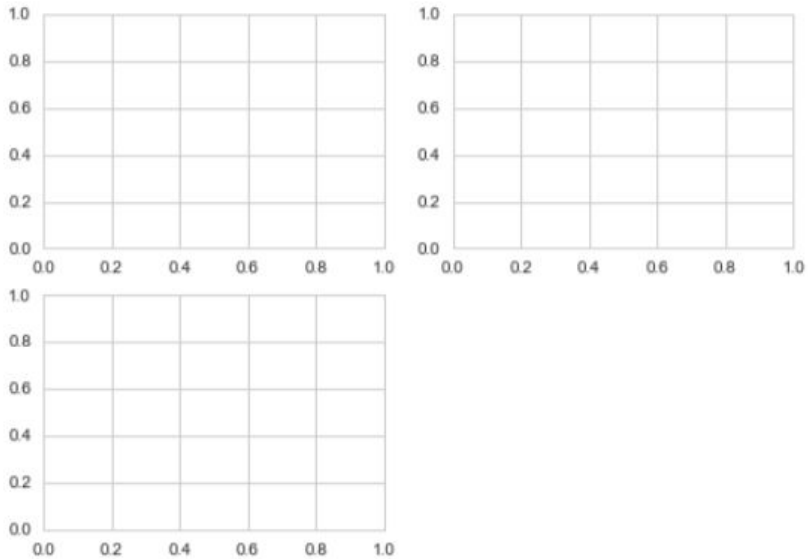


1-3. Matplotlib 여러개의 그래프 그리기

모두의연구소

In [72]:
```
fig = plt.figure()

ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```



1-4. Matplotlib 스타일 변경(Colors, Markers, and Line Styles)

모두의연구소

kind

문자열	의미
line	line plot (default)
bar	vertical bar plot
barh	horizontal bar plot
hist	histogram
box	boxplot
kde	Kernel Density Estimation plot
pie	pie plot
scatter	scatter plot

style

색깔

문자열	약자
blue	b
green	g
red	r
cyan	c
magenta	m
yellow	y
black	k
white	w

마커

마커	의미
.	point marker
,	pixel marker
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
1	tri_down marker
2	tri_up marker
3	tri_left marker
4	tri_right marker
s	square marker
p	pentagon marker
*	star marker
h	hexagon1 marker
H	hexagon2 marker
+	plus marker
x	x marker
D	diamond marker
d	thin_diamond marker

선스타일

선 스타일 문자열	의미
-	실선
--	대시선
-.	점선
:	대시-점선

기타 스타일

스타일	약자	의미
color	c	선 색깔
linewidth	lw	선 굵기
linestyle	ls	선 스타일
marker		마커 종류
markersize	ms	마커 크기
markeredgecolor	mec	마커 선 색깔
markeredgewidth	mew	마커 선 굵기
markerfacecolor	mfc	마커 내부 색깔

1-4. Matplotlib 스타일 변경(Colors, Markers, and Line Styles)

모두의연구소

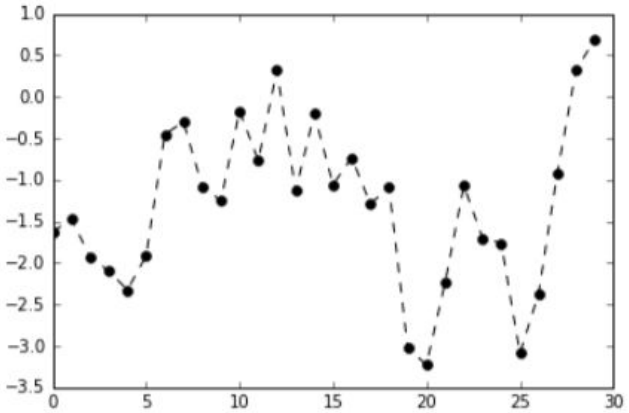
함수의 인자들을 명시적으로 적어주지 않아도 됨

ax.plot(x, y, 'g--')

> ax.plot(x, y, linestyle='--', color='g') 와 동일한 의미

In [16]:
```
from numpy.random import randn
plt.plot(randn(30).cumsum(), 'ko—')
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x16bb1025748>]
```



1-5. Matplotlib 결과 그림파일로 저장하기

모두의연구소

그림을 파일로 저장하기

Matplotlib은 다양한 형식의 그림을 저장할 수 있음

In [ ]:
```
plt.savefig('figpath.svg')
```

In [ ]:
```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

패키지를 어느 백엔드에 설치했느냐에 따라 여러 가지 파일 형식을 사용할 수 있음

In [56]:
```
fig.canvas.get_supported_filetypes()
```

Out[56]:
```
{'eps': 'Encapsulated Postscript',
 'jpeg': 'Joint Photographic Experts Group',
 'jpg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

2. Matplotlib의 여러가지 Plots

- Matplotlib는 다음과 같은 정형화된 차트나 플롯 이외에도 저수준 api를 사용한 다양한 시각화 기능 제공

▶ 라인 플롯(line plot)

▶스캐터 플롯(scatter plot)

▶컨투어 플롯(contour plot)

▶서피스 플롯(surface plot)

▶바 차트(bar chart)

▶히스토그램(histogram)

▶박스 플롯(box plot)

[Matplotlib 갤러리 웹사이트]

http://matplotlib.org/gallery.html

2-1. 산점도(Scatter Plot)

모두의연구소

Scatter Plots ¶

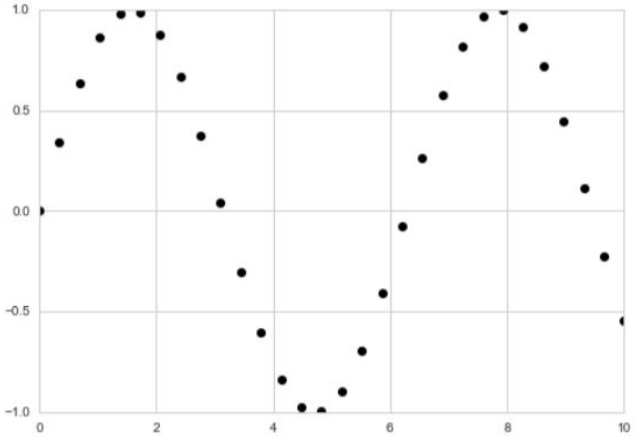
라인 플롯의 사촌 격이면서 보편적으로 사용되는 간단한 산점도

> 각 데이터가 하나의 점(dot)이나 원, 또는 다른 모양으로 표현
>
> 다양한 차원을 **동시에** 탐색 할 수 있다는 장점(*)

```
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color = 'black')
```

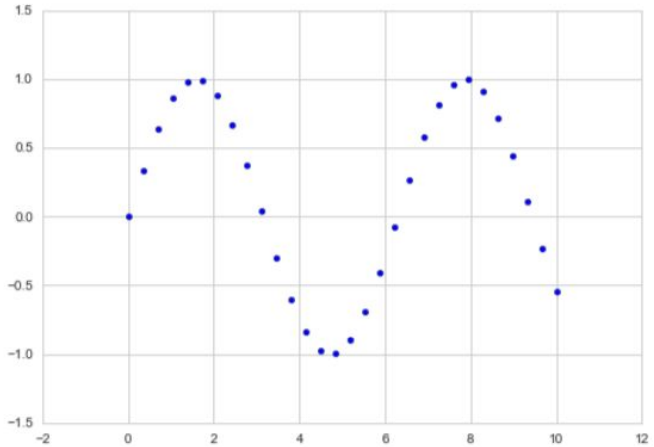
```
[<matplotlib.lines.Line2D at 0x16bb5866470>]
```



2-1. 산점도(Scatter Plot)

```
plt.scatter(x, y, marker='o')
```

```
<matplotlib.collections.PathCollection at 0x16bb584eb00>
```



Tip!

plot과 scatter의 차이: 효율성 측면에서 유의할 점

- 데이터 양이 적은 경우에는 그렇게 중요하지 않지만,
- 방대한 양의 데이터를 다루는 경우 **plt.plot**이 더 효율적임

> plt.scatter는 각 점에 대한 다양한 크기와 색상을 나타내는 능력이 있어서 렌더러가 각 점을 개별적으로 구성하는 추가 작업이 필요
>
> 반면 plt.plot에서는 점이 기본적으로 항상 서로 복제되므로 점의 모양을 결정하는 작업이 전체 데이터 집합에 대해 **한 번만** 수행

2-2. 라인 플롯(Line Plot)

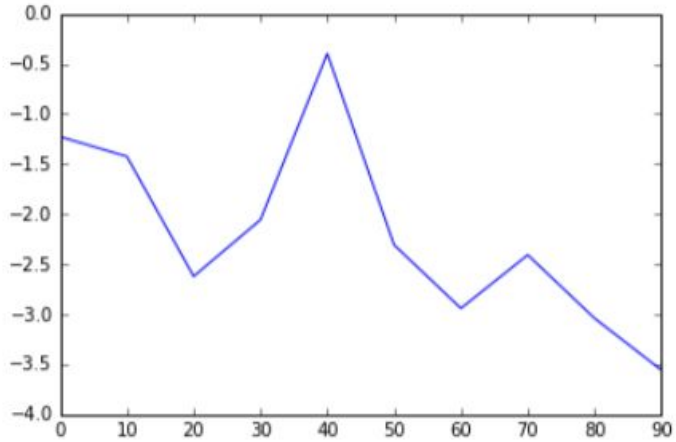
모두의연구소

Line Plots

라인 플롯은 데이터가 **시간, 순서 등에 따라 어떻게 변화하는지** 보여주기 위해 사용

```
s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
s.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bb10c69b0>
```

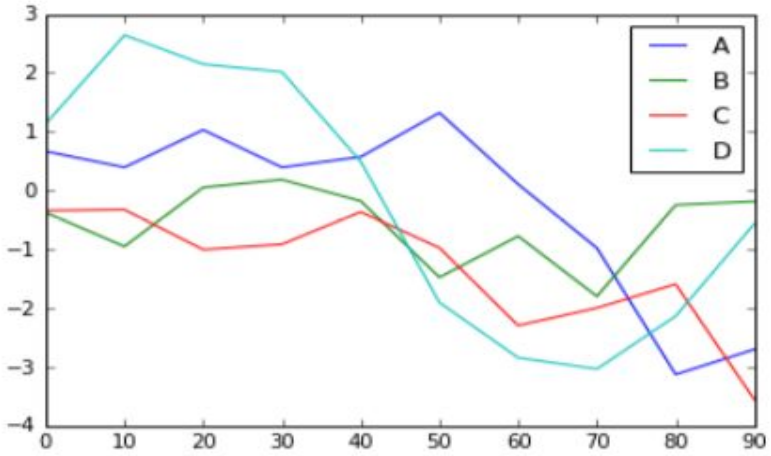


2-2. 라인 플롯(Line Plot)

Pandas DataFrame을 바로 시각화 할 수 있음

```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
                  columns=['A', 'B', 'C', 'D'],
                  index=np.arange(0, 100, 10))
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bb1323048>
```



2-3. 히스토그램(Histogram)

모두의연구소

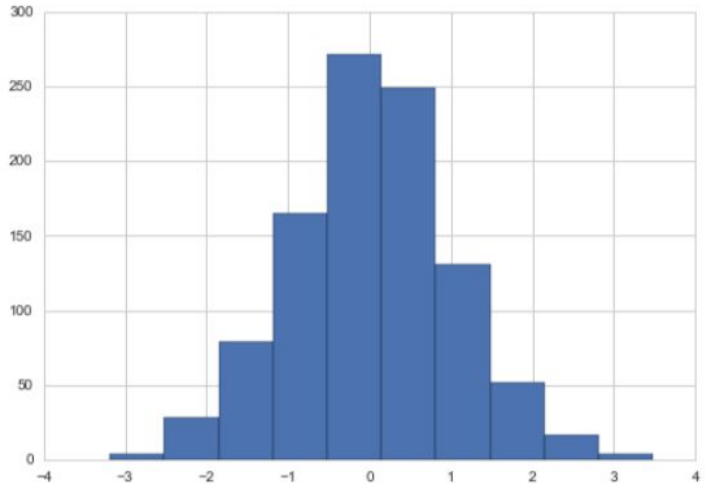
Histogram

데이터셋을 이해하는 가장 좋은 첫걸음은 히스토그램을 그려보는 것

> 각 변수들(데이터)의 분포를 확인

```
data = np.random.randn(1000)

plt.hist(data);
```



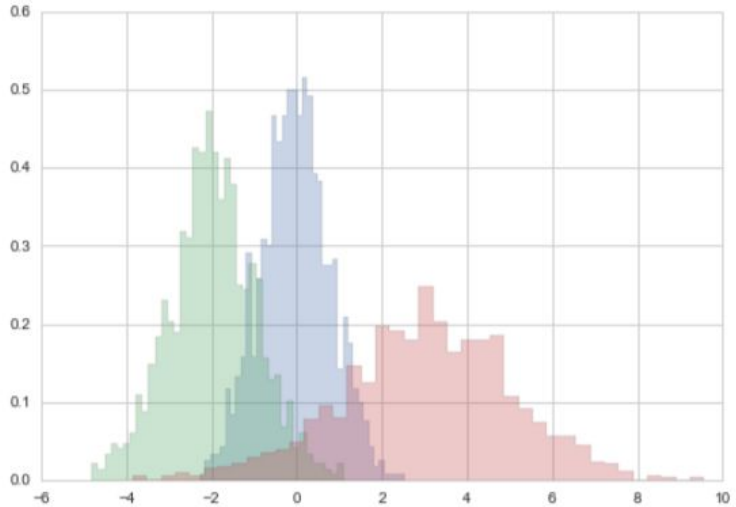
2-3. 히스토그램(Histogram)

여러가지 변수들을 한 번에 표현할 수 있음

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3, density=True, bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```



2-4. 바 플롯(Bar Plot)

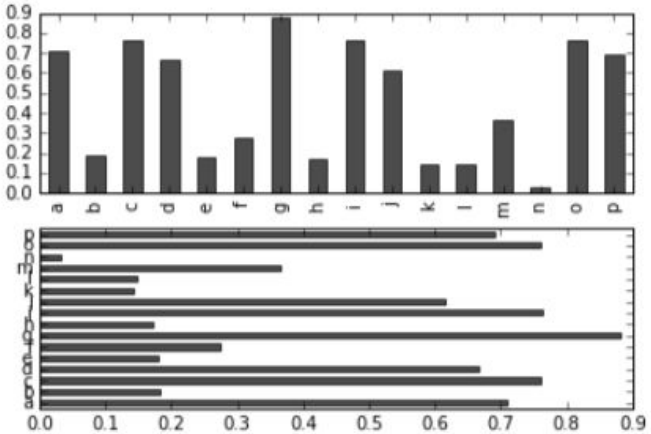
모두의연구소

Bar Plots

보고자 하는 데이터(x)가 **카테고리 값인 경우** 사용

```
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
data.plot.bar(ax=axes[0], color='k', alpha=0.7)
data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bb1465630>
```



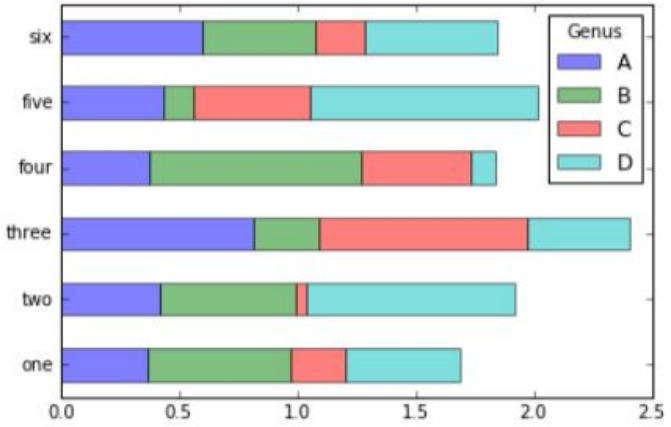
2-4. 바 플롯(Bar Plot)

누적 Bar를 그리기 위해서 stacked라는 인자에 True값을 입력(default: False)

> 데이터 시각화에 있어 각각의 카테고리 빈도의 차이가 뚜렷하지 않다면, 누적 Bar 플롯의 사용은 지양함

```
df.plot.barh(stacked=True, alpha=0.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16bb12c3208>
```



3. 최신 시각화 라이브러리

Matplotlib 기반 최신 시각화 라이브러리 소개 ¶

1. seaborn
   - matplotlib을 기반으로 만들어진 시각화 라이브러리
   - 디자인적으로 훨씬 세련됨.
   - matplotlib와 사용방식이 유사하므로 쉽고 빠르게 습득할 수 있음.
   - https://seaborn.pydata.org/

2. bokeh
   - 웹브라우저 상에서의 시각화에 효과적인 파이썬 인터랙티브 시각화 라이브러리
   - 플롯을 html 파일로 export하여 이를 웹브라우저를 통해 확인할 수 있음.
   - matplotlib와 비슷, jupyter와 호환이 잘 됨.
   - https://bokeh.pydata.org/en/latest/

3. Folium
   - 지리적 데이터 시각화에 특화된 라이브러리 (leaflet.js 기반)
   - 웹브라우저에서 확인 가능
   - 지도 데이터 사용을 위해 선행되어야 하는 작업이 원래 매우 많은데, 이러한 선행작업을 간단화함.
   - https://github.com/python-visualization/folium
   - http://python-visualization.github.io/folium/docs-v0.5.0/
   - http://pinkwink.kr/971

Scikit-learn

Python의 대표적인 머신러닝 라이브러리

1) 데이터 전처리 및 특징 선택
2) 알고리즘: 지도 학습 및 비지도 학습
3) 모델 선택 및 평가

머신러닝 파이프라인 기능을 고루 갖추고 있음.

1. 데이터 적재

모두의연구소

1. 데이터 적재 ¶

In [1]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

In [2]:
```
print("iris_dataset의 키: \n{}".format(iris_dataset.keys()))
```

```
iris_dataset의 키:
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

- 데이터 셋 설명

In [3]:
```
print(iris_dataset['DESCR'][:193] + '\n...')
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive att
...
```

2. 훈련, 평가 데이터 나누기

모두의연구소

2. 훈련 데이터와 테스트 데이터

훈련 데이터와 테스트 데이터로 나눈다. 테스트 데이터는 보지 않아야 한다.

In [13]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(iris_dataset['data'], iris_dataset['target'], random_state=0)
```

In [14]:
```
print("X_train 크기: {}".format(X_train.shape))
print("y_train 크기: {}".format(y_train.shape))
```

```
X_train 크기: (112, 4)
y_train 크기: (112,)
```

In [15]:
```
print("X_test 크기: {}".format(X_test.shape))
print("y_test 크기: {}".format(y_test.shape))
```

```
X_test 크기: (38, 4)
y_test 크기: (38,)
```

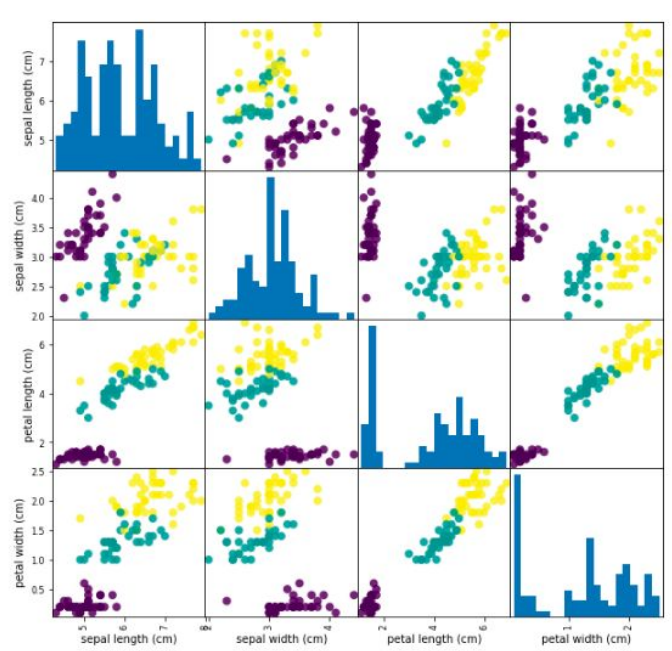
3. 데이터 탐색

모두의연구소

3. 데이터 탐색

In [22]:
```
import pandas as pd
%matplotlib inline
```

In [23]:
```
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
pd.plotting.scatter_matrix(iris_dataframe,
                           c=y_train,
                           figsize=(10, 10),
                           marker='o',
                           hist_kwds={'bins': 20},
                           s=60,
                           alpha=0.8
                          )
```



4. 머신러닝 알고리즘 적용

모두의연구소

4. 머신러닝 알고리즘 적용

In [20]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [24]:
```
knn.fit(X_train, y_train)
```

Out[24]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

5. 예측하기

모두의연구소

5. 예측하기

- 꽃받침 길이: 5cm, 폭이 2.9cm, 꽃잎 길이: 1cm, 폭이 0.2cm

In [27]:
```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X new shape: {}".format(X_new.shape))
```

```
X new shape: (1, 4)
```

In [28]:
```
prediction = knn.predict(X_new)
print("예측: {}".format(prediction))
print("예측한 타깃 이름: {}".format(iris_dataset['target_names'][prediction]))
```

```
예측: [0]
예측한 타깃 이름: ['setosa']
```

6. 모델 평가하기

모두의연구소

6. 모델 평가하기

In [29]:
```
y_pred = knn.predict(X_test)
print("테스트 세트에 대한 예측값: \n{}".format(y_pred))
```

```
테스트 세트에 대한 예측값:
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 2]
```

In [30]:
```
print("테스트 세트의 정확도: {:.2f}".format(np.mean(y_pred == y_test)))
```

```
테스트 세트의 정확도: 0.97
```

In [32]:
```
print("테스트 세트의 정확도: {:.2f}".format(knn.score(X_test, y_test)))
```

```
테스트 세트의 정확도: 0.97
```

모두의연구소

데이터 사이언스 실습(titanic)

Part 4-2

참고 자료

실리콘 밸리 과학자에게 배우는 데이터 사이언스(권재명, 제이펍)

파이썬 라이브러리를 활용한 데이터 분석(웨스 맥키니, 한빛미디어)

파이썬 라이브러리를 활용한 머신러닝(안드레아스 뮐러, 한빛미디어)

http://snowdeer.github.io/machine-learning/2018/01/04/deep-learning-comparation-among-frameworks/

https://github.com/brenden17/blog/blob/master/post/ms.scikit-learn.v.md