모두의연구소

Python 및 전처리

Day 2

커리큘럼

- 데이터 사이언스 프로세스
- Python Library Environment
- Numpy
- Pandas
- Matplotlib / Scikit-learn
- 데이터 사이언스 실습(titanic)

모두의연구소

데이터 사이언스 프로세스

Part 1

데이터 사이언스 프로세스

데이터 사이언스 프로세스

모두의연구소

1. 문제 정의
2. 데이터 정의

3. 데이터 취득
4. 데이터 가공 (Data Preprocessing)
5. 탐색적 분석과 시각화 (Exploratory Data Analysis, Data visualization)

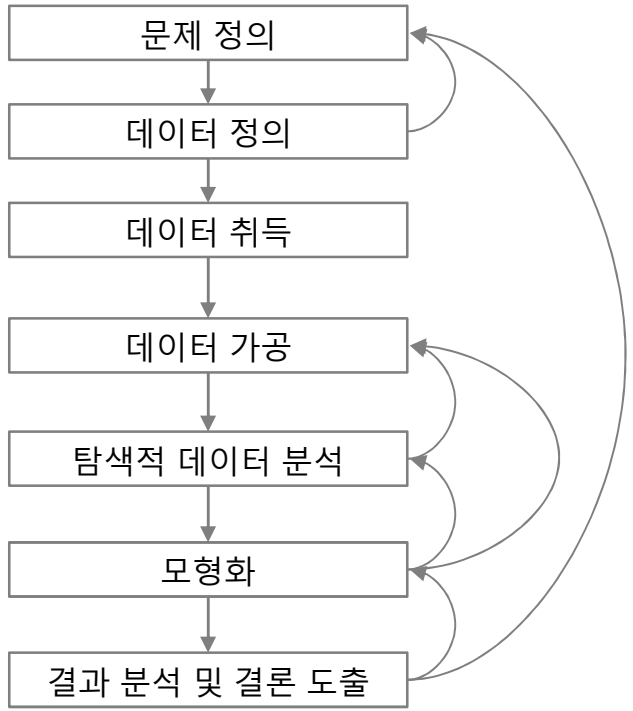
6. 모형화
7. 방법론 적용 및 결과 분석

8. 결론 도출

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



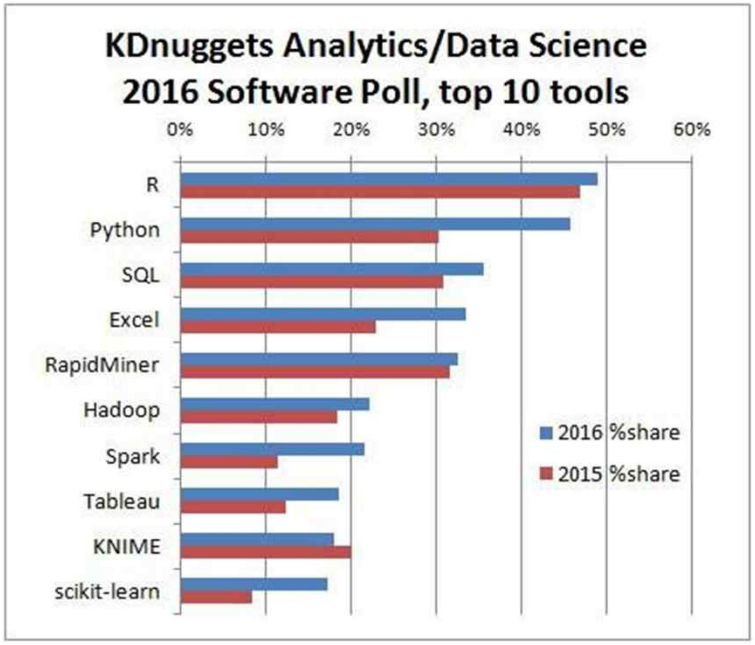
데이터 분석 과정에 대한 현실적 관점

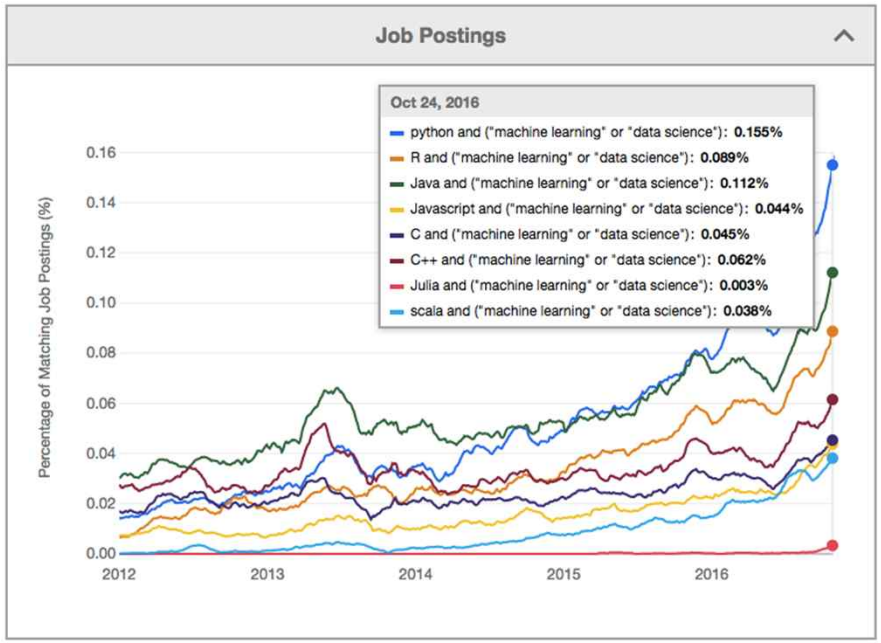
모두의연구소

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

모두의연구소

Numpy

모두의연구소

Numpy는 Numerical Python의 줄임말로 과학계산용 파운데이션 패키지.

1. Numpy

```
import numpy as np
```

```
data = np.random.randn(2,3)
data
```

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

```
data.shape
```

```
(2, 3)
```

```
data * 10
```

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

Pandas

모두의연구소

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

모두의연구소

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

2. Pandas

```
import pandas as pd
```

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

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

Matplotlib

모두의연구소

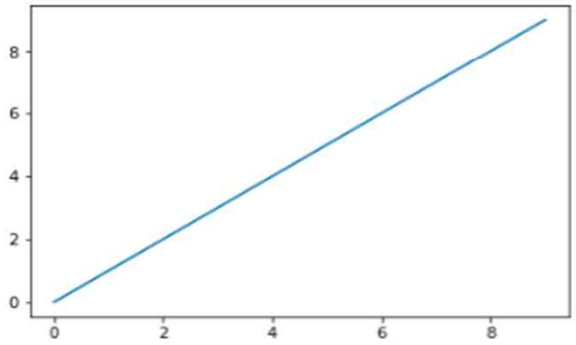
matplotlib은 그래프나 2차원 데이터 시각화를 생성하는 파이썬 라이브러리

. **3. Matplotlib**

```
from matplotlib import pyplot as plt
```

```
plt.plot(np.arange(10))
```

```
[<matplotlib.lines.Line2D at 0x117479080>]
```



Scikit-learn

모두의연구소

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 ¶

```
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

```
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

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

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

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

```
테스트 세트 정확도: 0.97
```

모두의연구소

Numpy

Part 3-1

http://www.numpy.org/

Numpy

모두의연구소

Numpy는 Numerical Python의 줄임말로 과학계산용 파운데이션 패키지.

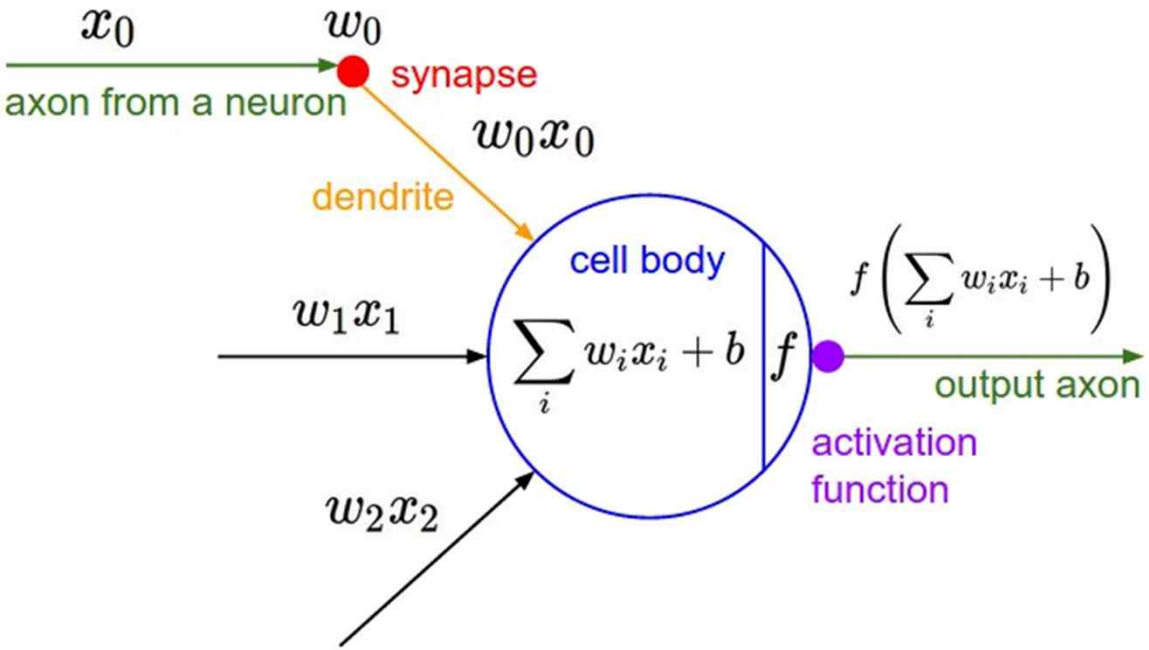
NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

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

모두의연구소

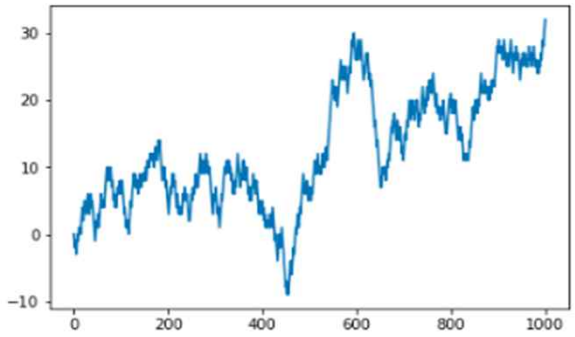
1. 계단 오르내리기 예제 - Python

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>]
```



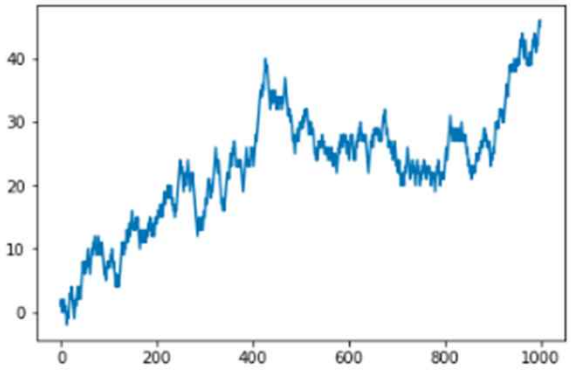
2. 계단 오르내리기 예제 - Numpy

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

모두의연구소

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

1-1. Ndarray

모두의연구소

Numpy ¶

```
import numpy as np
```

1) numpy import

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.

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

```
arr1 = np.array(data1)
```

2) Array 생성 : np.array 함수를 통해 배열을 생성 → **ndarray**

```
arr1
```

*) 하나의 데이터 type만 지원

```
array([5, 6, 7, 8])
```

```
data2 = [[1,2,3,4],[5,6,7,8]]
```

```
arr2 = np.array(data2)
```

```
arr2
```

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

- **shape** : 각 차원의 크기 (ex) arr.shape
- **dtype** : 배열에 저장된 자료형 (ex) arr.dtype

1-2. 자료형

모두의연구소

2. ndarray 자료형

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

→ single element 가 갖는 data type

dtype : 자료형 (int8, int16, int32, float16, float32, float64, bool, …)

[참고] Array Shape

1) vector

shape 0

1	2	6	3	9

ndarray shape(5,)

vector = np.array([1, 2, 6, 3, 9])

2) matrix

shape 1

shape 0

1	2	6	3	9
3	1	2	8	5

ndarray shape(2, 5)

matrix = np.array([[1, 2, 6, 3, 9], [3, 1, 2, 8,5]])

1-2. 자료형

모두의연구소

2. ndarray 자료형

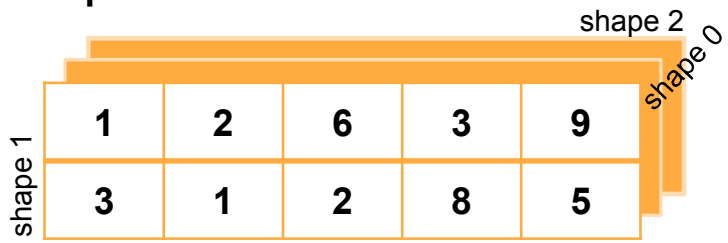
```
arr1 = np.array([1,2,3], dtype=np.float64)
arr2 = np.array([1,2,3], dtype=np.int32)
```

→ single element 가 갖는 data type

dtype : 자료형 (int8, int16, int32, float16, float32, float64, bool, …)

[참고] Array Shape

3) 3rd order tensor



ndarray shape(3, 2, 5)

```
np.array([[[1, 2, 6, 3, 9], [3, 1, 2, 8,5]],
          [[1, 2, 6, 3, 9], [3, 1, 2, 8,5]],
          [[1, 2, 6, 3, 9], [3, 1, 2, 8,5]]])
```

- ndim : number of dimension
- size : data 개수

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.

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

```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

```
arr * arr
```

```
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
```

```
arr - arr
```

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

```
1 / arr
```

```
array([[1.        , 0.5       , 0.33333333],
       [0.25      , 0.2       , 0.16666667]])
```

```
arr ** 0.5
```

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

Numpy는 array 간 기본적인 사칙 연산 지원

Matrix 내 같은 위치에 있는 element들 간 연산을 수행함

1-4. 색인과 슬라이싱

모두의연구소

4. 색인과 슬라이싱 기초

```
arr = np.arange(10)
```

```
arr
```

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

```
arr[5]
```

```
5
```

```
arr[5:8]
```

```
array([5, 6, 7])
```

```
arr[5:8] = 12
```

```
arr
```

```
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

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 벡터화된 래퍼 함수라고 생각하면 된다.

```
arr = np.arange(10)
arr
```

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

```
np.sqrt(arr)
```

[참고] np.sqrt(arr) == arr ** 0.5

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

```
np.exp(arr)
```

```
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])
```

```
x = np.random.randn(8)
y = np.random.randn(8)
```

[참고] randn(shape) : 임의의 표준정규분포 데이터 생성

```
np.maximum(x, y) # element wise operation
```

[참고] maximum(x, y) : 두 원소 중 큰값을 반환

```
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 np.where : [x **if** 조건 **else** y]

```
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])
```

```
result = np.where(cond, xarr, yarr)
```

```
result
```

```
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.

```
arr = np.random.rand(5, 4)
arr
```

```
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]])
```

```
arr.mean()
```

```
0.41445027069939266
```

```
np.mean(arr)
```

```
0.41445027069939266
```

2-2. 배열 연산(3) - 정렬

모두의연구소

3. 정렬

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

```
arr = np.random.randn(8)
arr
```

```
array([-0.72457977,  0.87587823,  1.2289314 , -0.28894393,  0.15205621,
        0.39222443, -1.86491412, -0.43901666])
```

```
arr.sort()
arr
```

```
array([-1.86491412, -0.72457977, -0.43901666, -0.28894393,  0.15205621,
        0.39222443,  0.87587823,  1.2289314 ])
```

```
arr = np.random.randn(5, 3)
arr
```

```
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]])
```

```
arr.sort(1)
arr
```

```
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.

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

```
x.dot(y)
```

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

```
from numpy.linalg import inv, qr
```

```
X = np.random.randn(5, 5)
```

```
mat = X.T.dot(X)
```

```
inv(mat)
```

```
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

3-1. 배열 재형성하기

모두의연구소

1. 배열 재형성하기 (reshape) : Array의 shape의 크기를 변경

```
arr = np.arange(8)
arr
```

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

```
arr.reshape((4,2))
```

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

```
arr.reshape((4,2)).reshape((2,4))
```

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

```
arr = np.arange(15).reshape((5, -1))
arr
```

→ size 기반으로 column의 개수를 선정

```
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.

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

```
np.concatenate([arr1, arr2], axis=0)
```

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

```
np.concatenate([arr1, arr2], axis=1)
```

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

- vstack: vertical
- hstack: horizontal

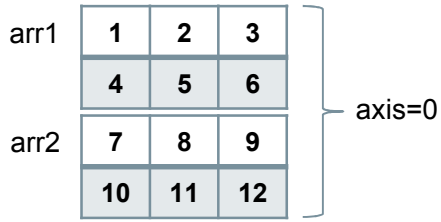
```
np.vstack((arr1, arr2))
```

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

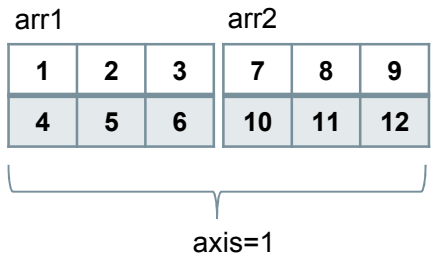
```
np.hstack((arr1, arr2))
```

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

*) **Concatenate, vstack**



*) **Concatenate, hstack**



3-3. 브로드캐스팅

모두의연구소

3. 브로드캐스팅

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

```
arr = np.arange(5)
arr
```

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

```
arr * 4
```

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

```
arr = np.random.randn(4, 3)
arr
```

```
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 ]])
```

```
arr.mean(axis=0)
```

```
array([-0.47985159,  0.27932981, -0.44077718])
```

```
demenad = arr - arr.mean(0)
demenad
```

```
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

모두의연구소

Pandas

https://pandas.pydata.org/

Pandas

Pandas는 **구조화된 데이터를 빠르고 쉬우면서도 다양한 형식으로 가공**할 수 있는 풍부한 자료 구조와 함수를 제공

home // about // get pandas // documentation // community // talks // donate

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

VERSIONS

Release
0.23.4 - August 2018
download // docs // pdf

1. Pandas 기본

1. 자료 구조 : Data Load

모두의연구소

```
1  import pandas as pd
```

pandas 라이브러리 호출

```
1  #data_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data'
2  #df = pd.read_csv(data_url, sep='\s+', header=None)
3  #df.to_csv("housing.csv", index=False)
4
5  df = pd.read_csv('housing.csv')
6  df.head()
```

CSV 파일 Read 및
data 출력

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

1. 자료 구조 : Series, DataFrame

모두의연구소

- Series와 DataFrame
  - Series : DataFrame 중 하나의 Colum에 해당하는 데이터 Object
  - DataFrame : Data 전체를 포함하는 Object

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

1. 자료 구조 : Series

- 1-1. Series 생성

```
1  pd.Series
```

```
Init signature: pd.Series(data=None, index=None, dtype=None, name=None, c
opy=False, fastpath=False)
Docstring:
One-dimensional ndarray with axis labels (including time series).
```

Series Object 생성

```
1  import pandas as pd
2
3  data = [2, 4, 6, 8, 10]
4  series = pd.Series(data)
5  series
```


index	data
0	2
1	4
2	6
3	8
4	10

Data type `dtype: int64`

1. 자료 구조 : Series

- 1-1. Series 생성 – index 및 data type 설정

```
1  data = [2, 4, 6, 8, 10]
2  idx = ['a', 'b', 'c', 'd', 'e']
3  series = pd.Series(data, dtype=np.float64, index=idx)
4  series
```

```
a     2.0
b     4.0
c     6.0
d     8.0
e    10.0
dtype: float64
```

- dtype : data type 설정
- index : index 이름 지정

1. 자료 구조 : Series

- 1-2. Series 데이터 접근

```
1 series
```

```
a     2.0
b     4.0
c     6.0
d     8.0
e    10.0
dtype: float64
```


- data index를 통해서 데이터 접근

```
1 series['c']
```

```
6.0
```

- data value 가져오기

```
1 series.values
```

```
array([ 2.,  4.,  6.,  8., 10.])
```

- data index 가져오기

```
1 series.index
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

1. 자료 구조 : Series

모두의연구소

- 1-3. Series 인덱싱/슬라이싱

```
1  data = pd.Series(['a', 'b', 'c', 'd'],
2                   index = [1, 3, 5, 7])
3  data
```

```
1    a
3    b
5    c
7    d
dtype: object
```

3-1. 인덱싱할 때 명시적인 인덱스 사용

```
1  data[3]
```

```
'b'
```

```
1  data.loc[1:3]
```

```
1    a
3    b
dtype: object
```

3-2. 슬라이싱할 때 암묵적 인덱스를 사용

```
1  data[1:3]
```

```
3    b
5    c
dtype: object
```

```
1  data.iloc[1:3]
```

```
3    b
5    c
dtype: object
```

1. 자료 구조 : Series

- 1-4. Series 데이터 수정

```
1 series
```

```
a     2.0
b     4.0
c     6.0
d     8.0
e    10.0
dtype: float64
```



```
1 series['c'] = 'c'
```

```
1 series
```

```
a     2
b     4
c     c
d     8
e    10
dtype: object
```

1. 자료 구조 : DataFrame

모두의연구소

- 2. DataFrame
  - Series를 모아서 만든 Data Table, 따라서 기본 2차원 행렬 구조

Columns

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

Index

1. 자료 구조 : DataFrame

모두의연구소

- 2-1. DataFrame 생성

```
1  pd.DataFrame
```

```
Init signature: pd.DataFrame(data=None, index=None, columns=Non
e, dtype=None, copy=False)
Docstring:
Two-dimensional size-mutable, potentially heterogeneous tabular
```

DataFrame Object 생성

```
1  raw_data = {'name' : ['Lee', 'Kim', 'Park'],
2              'age'  : [23, 62, 35],
3              'city' : ['Seoul', 'Jeju', 'Boston']}
4
5  df = pd.DataFrame(raw_data)
6  df
```

	name	age	city
0	Lee	23	Seoul
1	Kim	62	Jeju
2	Park	35	Boston

1. 자료 구조 : DataFrame

모두의연구소

- 2-1. DataFrame 생성

```
raw_data = [['Lee', 23, 'Seoul'],
            ['Kim,', 62, 'Jeju'],
            ['Park', 35, 'Boston']]
cols = ['name', 'age', 'city']
idx  = ['a', 'b', 'c']

df = pd.DataFrame(raw_data, columns=cols, index=idx)
df
```

columns, index 설정

	name	age	city
a	Lee	23	Seoul
b	Kim,	62	Jeju
c	Park	35	Boston

1. 자료 구조 : DataFrame

모두의연구소

- 2-2. DataFrame 접근

	name	age	city
0	Lee	23	Seoul
1	Kim	62	Jeju
2	Park	35	Boston

1) data column 선택

```
1 df.name
```

```
0     Lee
1     Kim
2    Park
Name: name, dtype: object
```

pandas.core.series.Series

```
1 df['name']
```

```
0     Lee
1     Kim
2    Park
Name: name, dtype: object
```

2) data Multi-column 선택

```
1 df[['name', 'city']]
```

pandas.core.frame.DataFrame

	name	city
0	Lee	Seoul
1	Kim	Jeju
2	Park	Boston

1. 자료 구조 : DataFrame

모두의연구소

- 2-3. DataFrame 신규 컬럼 추가 및 데이터 할당

	name	age	city
0	Lee	23	Seoul
1	Kim	62	Jeju
2	Park	35	Boston


```
1 df['debt'] = 0
2 df.debt = df.age > 40
3 df
```

	name	age	city	debt
0	Lee	23	Seoul	False
1	Kim,	62	Jeju	True
2	Park	35	Boston	False

1. 자료 구조 : DataFrame

모두의연구소

- 2-4. DataFrame 인덱싱/슬라이싱 - loc, iloc
  - **loc** : index location, **iloc** : index posistion

	name	age	city
1	Lee	23	Seoul
5	Kim,	62	Jeju
3	Park	35	Boston

```
1  df.loc[3]
```

index location

```
name       Park
age          35
city     Boston
Name: 3, dtype: object
```

	name	age	city
1	Lee	23	Seoul
5	Kim,	62	Jeju
3	Park	35	Boston

```
1  df.iloc[2]
```

index position

```
name       Park
age          35
city     Boston
Name: 3, dtype: object
```

1. 자료 구조 : DataFrame

모두의연구소

- 2-4. DataFrame 인덱싱/슬라이싱

```
raw_data = [['Lee', 23, 'Seoul'],
            ['Kim', 62, 'Jeju'],
            ['Park', 35, 'Boston']]
cols = ['name', 'age', 'city']
idx  = [1, 5, 3]

df = pd.DataFrame(raw_data, columns=cols, index=idx)
df
```

	name	age	city
1	Lee	23	Seoul
5	Kim	62	Jeju
3	Park	35	Boston

4-1. loc는 명시적 인덱스를 참조하는 인덱싱과 슬라이싱 가능

```
df.loc[3]
```

```
name       Park
age          35
city     Boston
Name: 3, dtype: object
```

```
df.loc[:5, ['name', 'age']]
```

	name	age
1	Lee	23
5	Kim	62

4-2. iloc는 인덱싱과 슬라이싱에서 암묵적으로 인덱스를 참조

```
df.iloc[2]
```

```
name       Park
age          35
city     Boston
Name: 3, dtype: object
```

```
df.iloc[:2, :2]
```

	name	age
1	Lee	23
5	Kim	62

1. 자료 구조 : DataFrame

모두의연구소

- 2-5. DataFrame 데이터 선택

```
raw_data = [['Lee', 23, 'Seoul'],
            ['Kim', 62, 'Jeju'],
            ['Park', 35, 'Boston']]
cols = ['name', 'age', 'city']
idx = [1, 5, 3]

df = pd.DataFrame(raw_data, columns=cols, index=idx)
```

	name	age	city
1	Lee	23	Seoul
5	Kim	62	Jeju
3	Park	35	Boston

5-1. Column 명으로 데이터 선택

```
df['name']
```

```
1     Lee
5     Kim
3    Park
Name: name, dtype: object
```

5-2. column명 없이 사용하는 index number는 row 기준

```
df[:2]
```

	name	age	city
1	Lee	23	Seoul
5	Kim	62	Jeju

1. 자료 구조 : DataFrame

모두의연구소

- 2-5. DataFrame 데이터 선택

```
raw_data = [['Lee', 23, 'Seoul'],
            ['Kim', 62, 'Jeju'],
            ['Park', 35, 'Boston']]
cols = ['name', 'age', 'city']
idx = [1, 5, 3]

df = pd.DataFrame(raw_data, columns=cols, index=idx)
```

	name	age	city
1	Lee	23	Seoul
5	Kim	62	Jeju
3	Park	35	Boston

5-3. column명과 함께 row index 사용하여 데이터 선택

```
df['name'][:2]
```

```
1    Lee
5    Kim
Name: name, dtype: object
```

```
df[:2]['name']
```

```
1    Lee
5    Kim
Name: name, dtype: object
```

```
df[['name', 'city']][:2]
```

	name	city
1	Lee	Seoul
5	Kim	Jeju

1. 자료 구조 : DataFrame

모두의연구소

- 2-6. Data Drop

```
raw_data = [['Lee', 23, 'Seoul'],
            ['Kim', 62, 'Jeju'],
            ['Park', 35, 'Boston']]
cols = ['name', 'age', 'city']
idx = ['name', 'age', 'city']

df = pd.DataFrame(raw_data, columns=cols, index=idx)
df
```

	name	age	city
name	Lee	23	Seoul
age	Kim	62	Jeju
city	Park	35	Boston

6-1. del

```
del df['name']
df
```

	age	city
name	23	Seoul
age	62	Jeju
city	35	Boston

6-2. drop (axis 축 설정)

```
df.drop('age', axis=1)
```

	name	city
name	Lee	Seoul
age	Kim	Jeju
city	Park	Boston

```
df.drop('age', axis=0)
```

	name	age	city
name	Lee	23	Seoul
city	Park	35	Boston

2. Pandas - 기본 데이터 분석

2. 기본 데이터 분석

- 1. 데이터 정렬
  - sort_index() : Row나 Column의 index를 기준으로 정렬
  - sort_values(): column 값을 기준으로 정렬

1-1. sort_index()

```
# Row index(j, i) 정렬
df.sort_index()
```

	d	a	c	b
i	4	5	6	7
j	0	1	2	3

```
# column index(d, a, c, b) 정렬
df.sort_index(axis=1)
```

	a	b	c	d
j	1	3	2	0
i	5	7	6	4

1-2. sort_values()

```
# 'a'값을 기준으로 내림차순 정렬
df.sort_values('a', ascending=False)
```

	d	a	c	b
i	4	5	6	7
j	0	1	2	3

2. 기본 데이터 분석

- 2. 데이터 살펴보기(1)

```
cols = ['a', 'b', 'c', 'd', 'e']
df = pd.DataFrame(np.random.randint(20, size=20).reshape(-1, 5),
                  columns=cols)
df
```

	a	b	c	d	e
0	11	12	3	13	11
1	1	2	10	6	15
2	4	3	19	0	10
3	7	9	9	9	0

---

1) 기초 통계량

count, mean, std, min/max등 출력

```
df.describe()
```

	a	b	c	d	e
count	4.000000	4.000000	4.000000	4.000000	4.000000
mean	5.750000	6.500000	10.250000	7.000000	9.000000
std	4.272002	4.795832	6.601767	5.477226	6.377042
min	1.000000	2.000000	3.000000	0.000000	0.000000
25%	3.250000	2.750000	7.500000	4.500000	7.500000
50%	5.500000	6.000000	9.500000	7.500000	10.500000
75%	8.000000	9.750000	12.250000	10.000000	12.000000
max	11.000000	12.000000	19.000000	13.000000	15.000000

2) DataFrame Info

자료 타입, 행/열 개수, 각 컬럼별 변수 타입, 메모리 사용량 출력

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 5 columns):
a    4 non-null int32
b    4 non-null int32
c    4 non-null int32
d    4 non-null int32
e    4 non-null int32
dtypes: int32(5)
memory usage: 160.0 bytes
```

2. 기본 데이터 분석

- 2. 데이터 살펴보기(2)

```
cols = ['a', 'b', 'c', 'd', 'e']
df = pd.DataFrame(np.random.randint(20, size=20).reshape(-1, 5),
                  columns=cols)
df
```

	a	b	c	d	e
0	11	12	3	13	11
1	1	2	10	6	15
2	4	3	19	0	10
3	7	9	9	9	0

---

3) DataFrame Size/Shape

```
df.shape
```

```
(4, 5)
```

```
len(df)
```

```
4
```

4) Unique : Series data의 유일한 값을 반환함

```
df['a'].unique()
```

```
array([11,  1,  4,  7], dtype=int64)
```

```
df['a'].nunique()
```

```
4
```

```
df['a'].value_counts()
```

```
7     1
4     1
11    1
1     1
Name: a, dtype: int64
```

2. 기본 데이터 분석

모두의연구소

- 3. 결측치 처리
  - NA 처리 Method

Method	Description
dropna	누락된 데이터가 있는 axis (0:row, 1:column) 를 제외 - how : 어느정도 누락 데이터까지 용인할 것인지는 지정할 수 있음 (ex. any, all)
fillna	누락된 데이터를 대신할 값을 채우거나 'ffill' 또는 'bfill'을 통해 값을 채움 - ffill : NaN의 값이 그 앞 데이터로 부터 채워짐 - bfill : NaN의 값이 그 뒤 데이터로 부터 채워짐
isnull	누락되거나 NA인 값을 알려주는 bool 값이 저장된 객체 반환
notnull	isnulll와 반대되는 method

2. 기본 데이터 분석

- 3. 결측치 처리
  - dropna : 누락된 데이터가 있는 axis (0:row, 1:column) 를 제외

```
data = [[1., 6.5, 3.],
        [1., np.NaN, np.NaN],
        [np.NaN, np.NaN, np.NaN],
        [np.NaN, 6.5, 3]
]
cols = ['a', 'b', 'c']

df = pd.DataFrame(data, columns=cols)
```

	a	b	c
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
df.dropna()
```

	a	b	c
0	1.0	6.5	3.0

```
df.dropna(how='all')
```

	a	b	c
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

```
df.dropna(how='all', axis=1)
```

	a	b	c
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

2. 기본 데이터 분석

- 3. 결측치 처리
  - fillna : 누락된 데이터를 대신할 값을 채우거나 ‘ffill’/‘bfill‘을 통해 값을 채움

```
data = [[1., 2.5, 7.],
        [2., np.NaN, np.NaN],
        [np.NaN, np.NaN, np.NaN],
        [np.NaN, 6.5, 3]
]
cols = ['a', 'b', 'c']

df = pd.DataFrame(data, columns=cols)
```

	a	b	c
0	1.0	2.5	7.0
1	2.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
df.fillna(0)
```

	a	b	c
0	1.0	2.5	7.0
1	2.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	6.5	3.0

```
df.fillna(df.mean())
```

	a	b	c
0	1.0	2.5	7.0
1	2.0	4.5	5.0
2	1.5	4.5	5.0
3	1.5	6.5	3.0

```
df.fillna(method='ffill')
```

	a	b	c
0	1.0	2.5	7.0
1	2.0	2.5	7.0
2	2.0	2.5	7.0
3	2.0	6.5	3.0

```
df.fillna(method='bfill')
```

	a	b	c
0	1.0	2.5	7.0
1	2.0	6.5	3.0
2	NaN	6.5	3.0
3	NaN	6.5	3.0

2. 기본 데이터 분석

- 3. 결측치 처리
  - isnull/notnull

```
data = [[1., 6.5, 3.],
        [1., np.NaN, np.NaN],
        [np.NaN, np.NaN, np.NaN],
        [np.NaN, 6.5, 3]
]
cols = ['a', 'b', 'c']

df = pd.DataFrame(data, columns=cols)
```

	a	b	c
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
df.isnull()
```

	a	b	c
0	False	False	False
1	False	True	True
2	True	True	True
3	True	False	False

```
df.notnull()
```

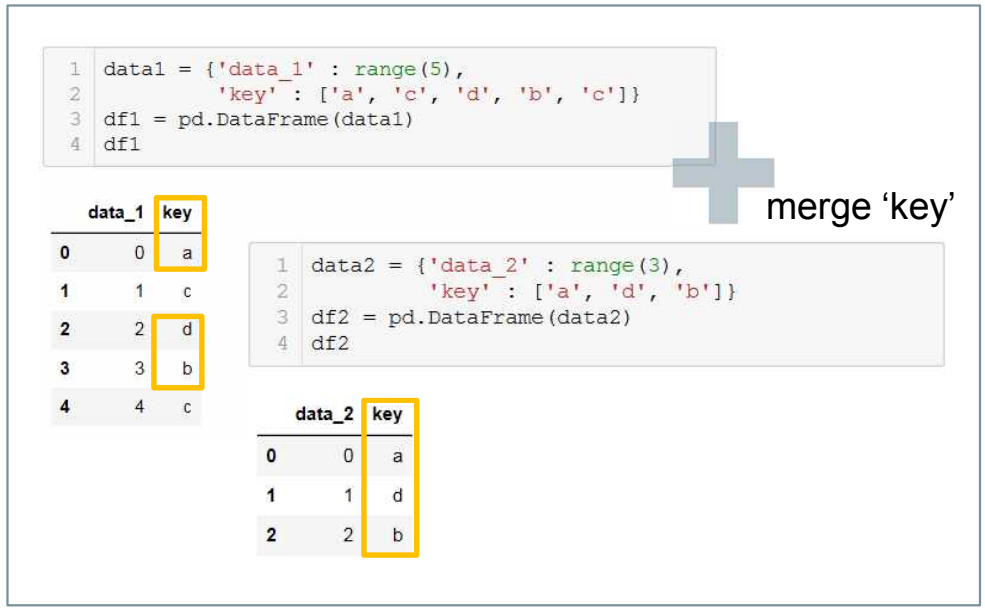
	a	b	c
0	True	True	True
1	True	False	False
2	False	False	False
3	False	True	True

3. Pandas - 그룹 함수

모두의연구소

3. 그룹 함수

- 1. Merge
  - 키를 하나 이상 사용해서 데이터 집합의 Row를 합침



	data_1	key
0	0	a
1	1	c
2	2	d
3	3	b
4	4	c

	data_2	key
0	0	a
1	1	d
2	2	b

	data_1	key	data_2
0	0	a	0
1	2	d	1
2	3	b	2

3. 그룹 함수

- 1. Merge : 공통 칼럼이 없는 경우 left_on, right_on 으로 column 지정
  - 키를 하나 이상 사용해서 데이터 집합의 Row를 합침

```
data1 = {'data_1' : range(5),
         'key_1' : ['a', 'c', 'a', 'b', 'c']}
df1 = pd.DataFrame(data1)
df1
```

	data_1	key_1
0	0	a
1	1	c
2	2	a
3	3	b
4	4	c

merge
'key_1' & 'key_2'

```
data2 = {'data_2' : range(3),
         'key_2' : ['a', 'd', 'b']}
df2 = pd.DataFrame(data2)
df2
```

	data_2	key_2
0	0	a
1	1	d
2	2	b

```
pd.merge(df1, df2, left_on='key_1', right_on='key_2')
```

	data_1	key_1	data_2	key_2
0	0	a	0	a
1	2	a	0	a
2	3	b	2	b

3. 그룹 함수

- 1. Merge : how (**left**, right, outer, inner)
  - 키를 하나 이상 사용해서 데이터 집합의 Row를 합침

```
data1 = {'data_1' : range(5),
         'key' : ['a', 'c', 'd', 'b', 'c']}
df1 = pd.DataFrame(data1)
df1
```

	data_1	key
0	0	a
1	1	c
2	2	d
3	3	b
4	4	c

merge : how ('left')

```
data2 = {'data_2' : range(3),
         'key' : ['a', 'e', 'b']}
df2 = pd.DataFrame(data2)
df2
```

	data_2	key
0	0	a
1	1	e
2	2	b

```
pd.merge(df1, df2, how='left')
```

	data_1	key	data_2
0	0	a	0.0
1	1	c	NaN
2	2	d	NaN
3	3	b	2.0
4	4	c	NaN

3. 그룹 함수

- 1. Merge : how (left, **right**, outer, inner)
  - 키를 하나 이상 사용해서 데이터 집합의 Row를 합침

```
data1 = {'data_1' : range(5),
         'key' : ['a', 'c', 'd', 'b', 'c']}
df1 = pd.DataFrame(data1)
df1
```

	data_1	key
0	0	a
1	1	c
2	2	d
3	3	b
4	4	c

merge : how ('right')

```
data2 = {'data_2' : range(3),
         'key' : ['a', 'e', 'b']}
df2 = pd.DataFrame(data2)
df2
```

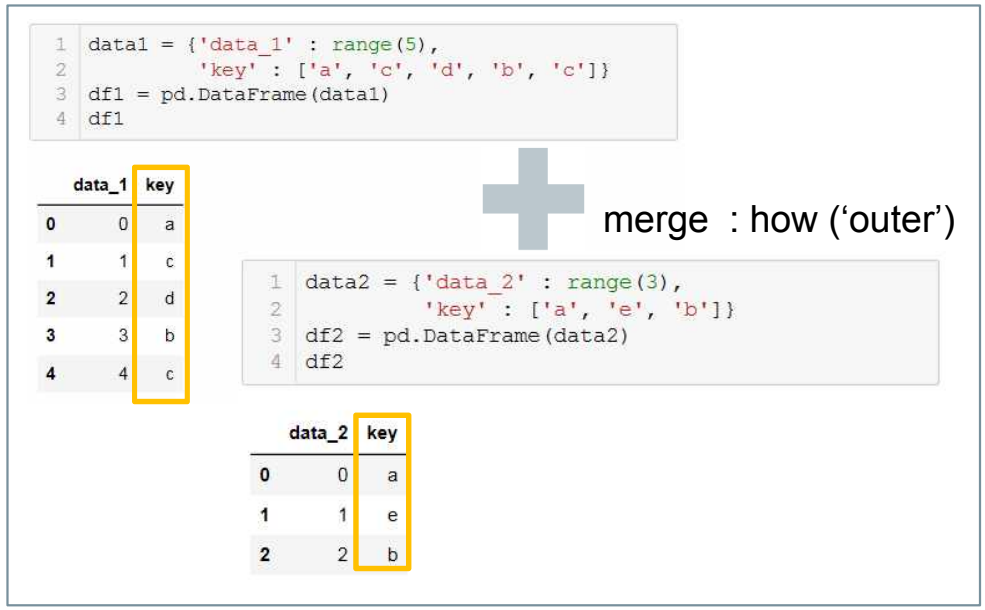
	data_2	key
0	0	a
1	1	e
2	2	b

```
pd.merge(df1, df2, how='right')
```

	data_1	key	data_2
0	0.0	a	0
1	3.0	b	2
2	NaN	e	1

3. 그룹 함수

- 1. Merge : how (left, right, **outer**, inner)
  - 키를 하나 이상 사용해서 데이터 집합의 Row를 합침



```
data1 = {'data_1' : range(5),
         'key' : ['a', 'c', 'd', 'b', 'c']}
df1 = pd.DataFrame(data1)
df1
```

	data_1	key
0	0	a
1	1	c
2	2	d
3	3	b
4	4	c

merge : how ('outer')

```
data2 = {'data_2' : range(3),
         'key' : ['a', 'e', 'b']}
df2 = pd.DataFrame(data2)
df2
```

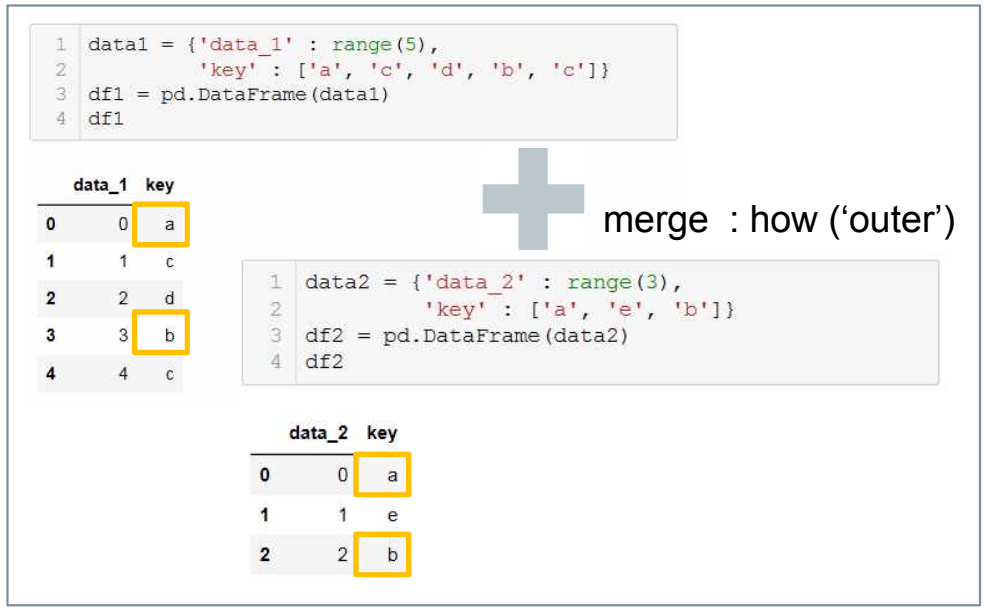
	data_2	key
0	0	a
1	1	e
2	2	b

```
pd.merge(df1, df2, how='outer')
```

	data_1	key	data_2
0	0.0	a	0.0
1	1.0	c	NaN
2	4.0	c	NaN
3	2.0	d	NaN
4	3.0	b	2.0
5	NaN	e	1.0

3. 그룹 함수

- 1. Merge : how (left, right, outer, **inner**)
  - 키를 하나 이상 사용해서 데이터 집합의 Row를 합침



```
data1 = {'data_1' : range(5),
         'key' : ['a', 'c', 'd', 'b', 'c']}
df1 = pd.DataFrame(data1)
df1
```

	data_1	key
0	0	a
1	1	c
2	2	d
3	3	b
4	4	c

merge : how ('outer')

```
data2 = {'data_2' : range(3),
         'key' : ['a', 'e', 'b']}
df2 = pd.DataFrame(data2)
df2
```

	data_2	key
0	0	a
1	1	e
2	2	b

```
pd.merge(df1, df2, how='inner')
```

	data_1	key	data_2
0	0	a	0
1	3	b	2

3. 그룹 함수

- 1. Merge
  - merge(how = _VALUE_) 사용 시, 공통 키가 없는 경우

```
1  pd.merge(df1, df2, how='right')
```

	data_1	key_1
0	0	a
1	1	c
2	2	a

+

	data_2	key_2
0	0	a
1	1	d
2	2	b

…… (중략)

```
~\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _validate_specification(self)
    951                         self.right.columns)
    952                 if len(common_cols) == 0:
--> 953                     raise MergeError('No common columns to perform merge on')
    954                 if not common_cols.is_unique:
    955                     raise MergeError("Data columns not unique: {common!r}"

MergeError: No common columns to perform merge on
```

3. 그룹 함수

- 2. concat
  - 3개의 Series 객체 연결하기

```
1  s1 = pd.Series(['a', 'b'],      index=[0, 1])
2  s2 = pd.Series(['c', 'd', 'e'], index=[2, 3, 4])
3  s3 = pd.Series(['f', 'g'],      index=[5, 6])
4
5  pd.concat([s1, s2, s3])
```

```
0    a
1    b
2    c
3    d
4    e
5    f
6    g
dtype: object
```

s1

s2

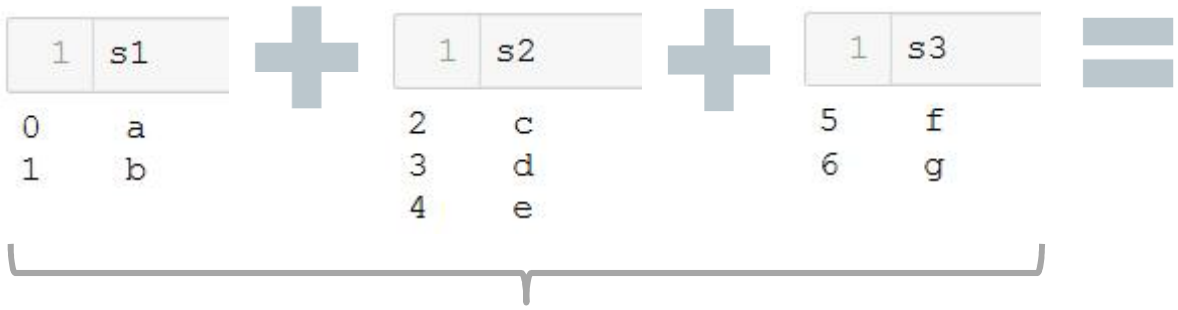
s3

s1, s2, s3 객체를 리스트로 묶어서 concat() 에 전달

3. 그룹 함수

- 2. concat
  - 3개의 Series 객체 연결하기

```
1  s1 = pd.Series(['a', 'b'],      index=[0, 1])
2  s2 = pd.Series(['c', 'd', 'e'], index=[2, 3, 4])
3  s3 = pd.Series(['f', 'g'],      index=[5, 6])
4
5  pd.concat([s1, s2, s3], axis=1)
```



Series

	0	1	2
0	a	NaN	NaN
1	b	NaN	NaN
2	NaN	c	NaN
3	NaN	d	NaN
4	NaN	e	NaN
5	NaN	NaN	f
6	NaN	NaN	g

DataFrame

3. 그룹 함수

- 2. concat
  - DataFrame 객체 연결하기

	c1	c2
a	0	1
b	2	3
c	4	5

+

	c3	c4
a	0	1
d	2	3
c	4	5
e	6	7

1. concat (axis=0)

```
1  # Index reset
2  pd.concat([df1, df2], axis=0, ignore_index=True)
```

	c1	c2	c3	c4
0	0.0	1.0	NaN	NaN
1	2.0	3.0	NaN	NaN
2	4.0	5.0	NaN	NaN
3	NaN	NaN	0.0	1.0
4	NaN	NaN	2.0	3.0
5	NaN	NaN	4.0	5.0
6	NaN	NaN	6.0	7.0

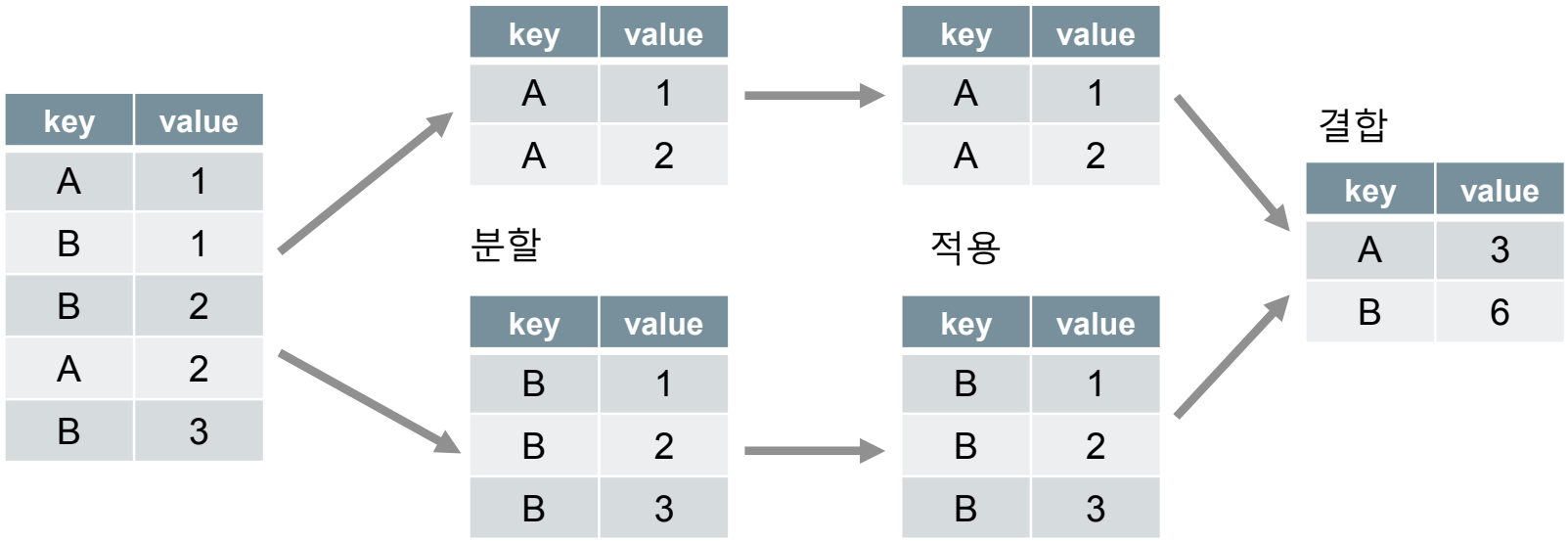
2. concat (axis=1)

```
1  pd.concat([df1, df2], axis=1)
```

	c1	c2	c3	c4
a	0.0	1.0	0.0	1.0
b	2.0	3.0	NaN	NaN
c	4.0	5.0	4.0	5.0
d	NaN	NaN	2.0	3.0
e	NaN	NaN	6.0	7.0

3. 그룹 함수

- 2. groupBy
  - 분할(Split), 적용(apply), 결합(combine)



3. 그룹 함수

- 2. groupBy
  - 분할(Split), 적용(apply), 결합(combine)

```
1 data = {'key'   : ['A', 'B', 'B', 'A', 'B'],
2         'value' : [1, 1, 2, 2, 3]}
3 df = pd.DataFrame(data)
4 df
```

	key	value
0	A	1
1	B	1
2	B	2
3	A	2
4	B	3

```
1 df.groupby('key')['value'].sum()
```

```
key
A    3
B    6
Name: value, dtype: int64
```

1) key

- 분할-적용-결합 연산을 진행, 묶음의 기준이 되는 column

2) value

- 연산의 적용을 받는 column

3) sum()

- 적용 연산

3. 그룹 함수

모두의연구소

- 2. groupBy
  - 분할(Split), 적용(apply), 결합(combine)

```
data = {'key'   : ['A', 'B', 'B', 'A', 'B'],
        'value' : [1, 1, 2, 2, 3]}
df = pd.DataFrame(data)
df
```

	key	value
0	A	1
1	B	1
2	B	2
3	A	2
4	B	3

```
# 평균
df.groupby('key')['value'].mean()
```

```
key
A    1.5
B    2.0
Name: value, dtype: float64
```

```
# Data Size (그룹의 원소 개수)
df.groupby('key')['value'].size()
```

```
key
A    2
B    3
Name: value, dtype: int64
```

3. 그룹 함수

- 2. groupBy
  - 분할(Split), 적용(apply), 결합(combine)

한개 이상의 column으로 Group 화

```
1 data = {'key1'   : ['A', 'B', 'B', 'A', 'B'],
2         'key2'   : ['C', 'C', 'C', 'D', 'D'],
3         'value'  : [1, 1, 2, 2, 3]}
4 df = pd.DataFrame(data)
5 df
```

	key1	key2	value
0	A	C	1
1	B	C	1
2	B	C	2
3	A	D	2
4	B	D	3

```
1 df.groupby(['key1', 'key2'])['value'].sum()
```

```
key1  key2
A     C       1
      D       2
B     C       3
      D       3
Name: value, dtype: int64
```

모두의연구소

Matplotlib

https://matplotlib.org/

Matplotlib

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

1. Matplotlib 시작하기

모두의연구소

matplotlib은 그래프나 2차원 데이터 시각화를 생성하는 파이썬 라이브러리이다.

- 참고 : https://matplotlib.org/tutorials/index.html

```
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

IPython 노트북에서 플로팅하기

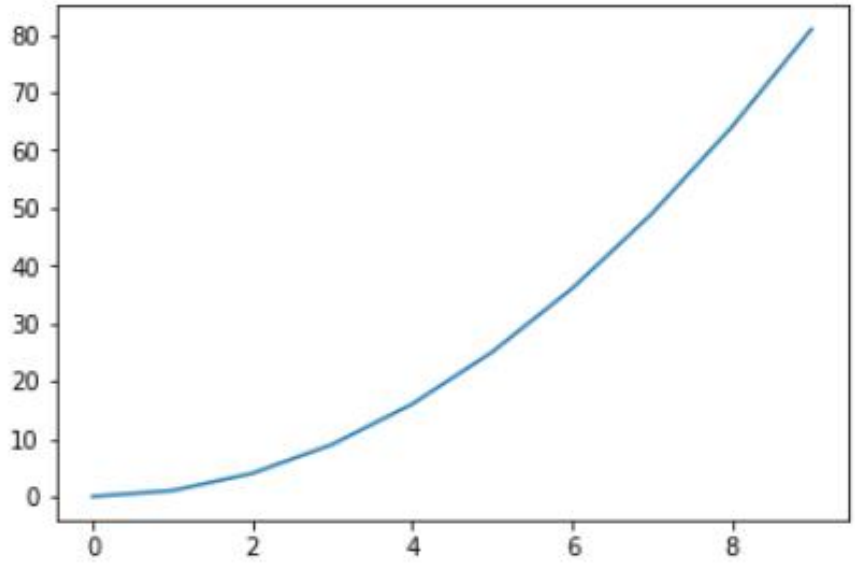
- %matplotlib inline은 노트북에 **정적** 이미지를 삽입할 수 있음

```
1 %matplotlib inline
```

1. Matplotlib 시작하기

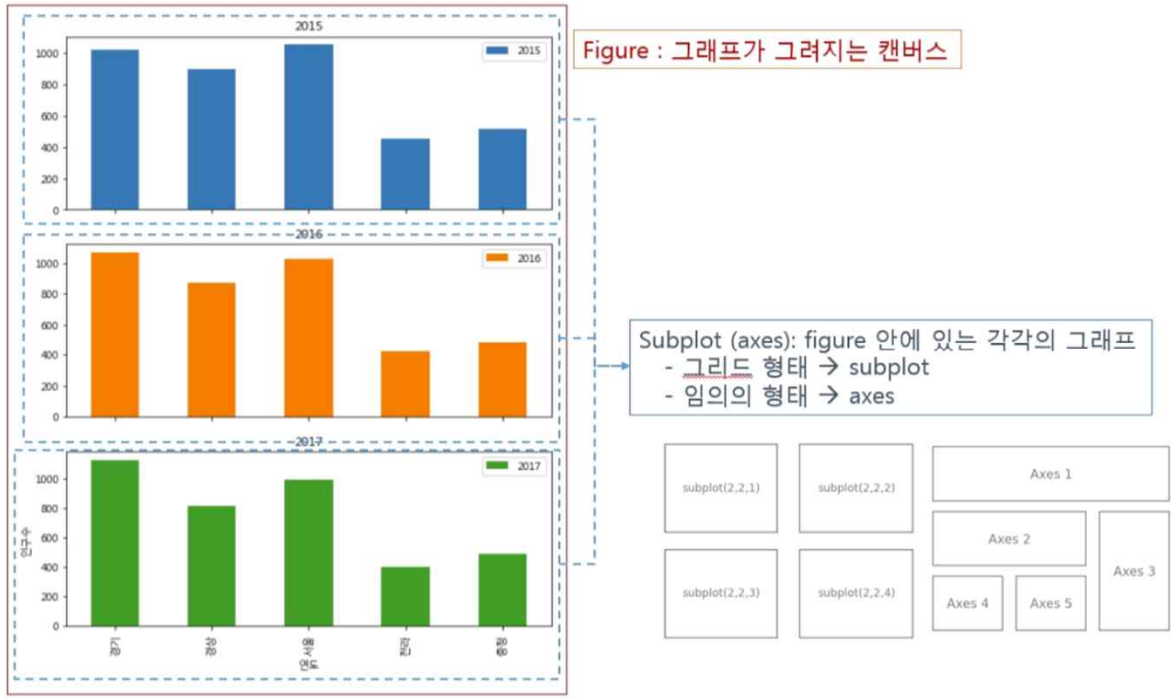
- pyplot 객체를 사용하여 데이터를 표시
- plt.show()

```
1  import matplotlib.pyplot as plt
2
3  x = range(10)
4  y = [v**2 for v in x]
5  plt.plot(x, y)
6  plt.show()
```



1. Matplotlib 시작하기

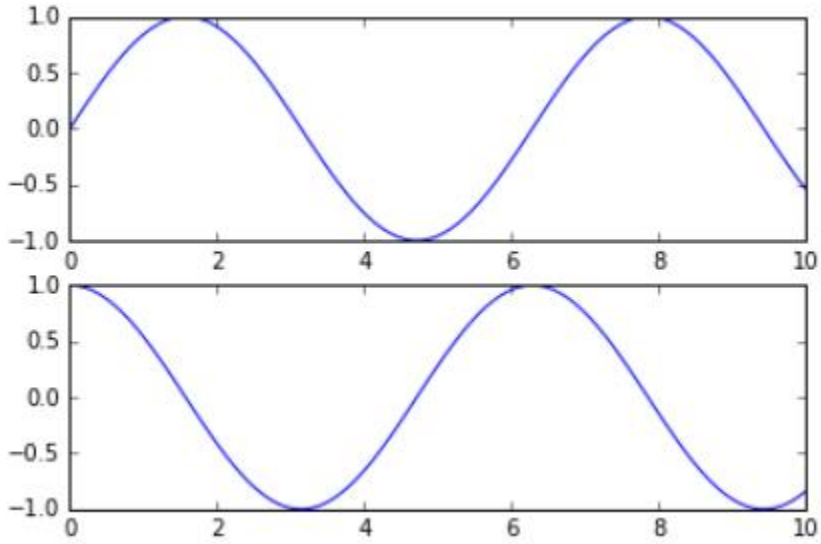
- Matplotlib 여러개의 그래프 그리기



1. Matplotlib 시작하기

- Matplotlib 여러개의 그래프 그리기 (subplot)
  - plt.subplot (row, columns, panel_number)

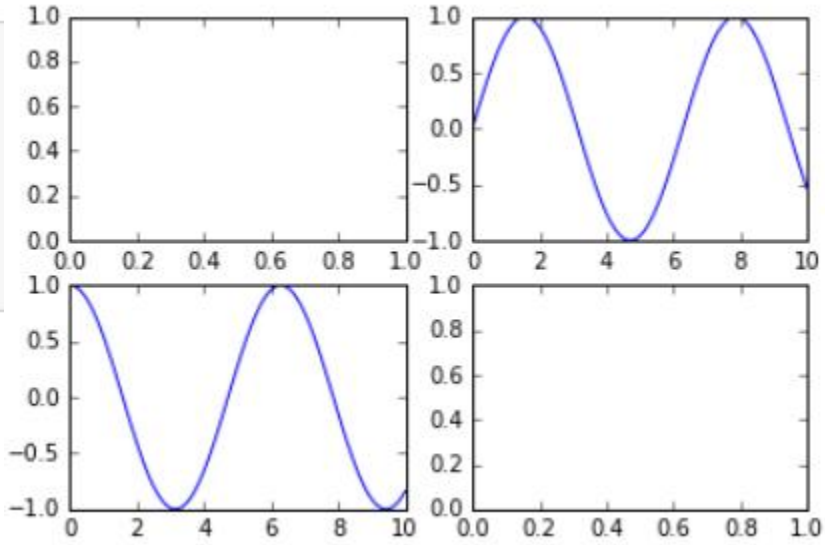
```
1  import numpy as np
2  x = np.linspace(0, 10, 100)
3
4  plt.figure() # 플롯 그림을 생
5
6  # 두 개의 패널 중 첫 번째 패널을 생성하고 현재 축(axis)을 설정
7  plt.subplot(2,1,1) # (rows, columns, panel number)
8  plt.plot(x, np.sin(x))
9
10 # 두 번째 패널을 생성하고 현재 축(axis)을 설정
11 plt.subplot(2,1,2)
12 plt.plot(x, np.cos(x))
```



1. Matplotlib 시작하기

- Matplotlib 여러개의 그래프 그리기 (figure, axes)
  - fig, ax = plt.subplot(2, 2)
  - ex) ax[0][1] 로 plot 접근

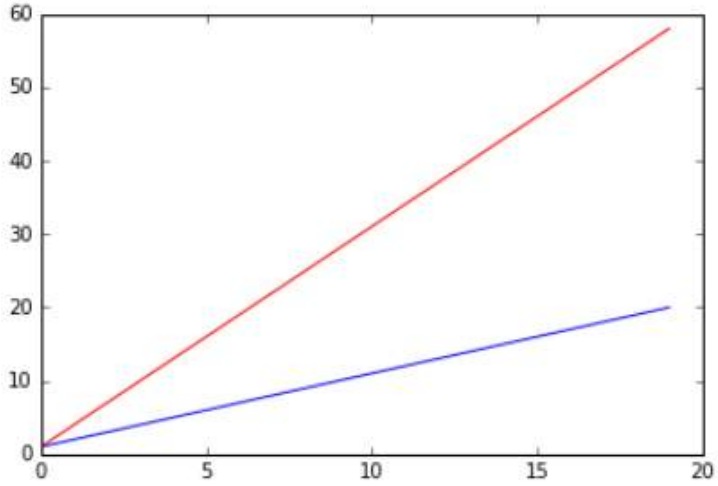
```
1  # 플롯 그리드를 생성
2  # ax는 두개의 축 객체의 배열이 됨
3  fig, ax = plt.subplots(2,2)
4
5  # 적절한 객체에서 plot() 메소드를 호출
6  ax[0][1].plot(x, np.sin(x))
7  ax[1][0].plot(x, np.cos(x))
```



1. Matplotlib 시작하기

- 스타일 변경하기 (color 지정하기)
  - 참고 : https://matplotlib.org/2.0.0/examples/color/named_colors.html

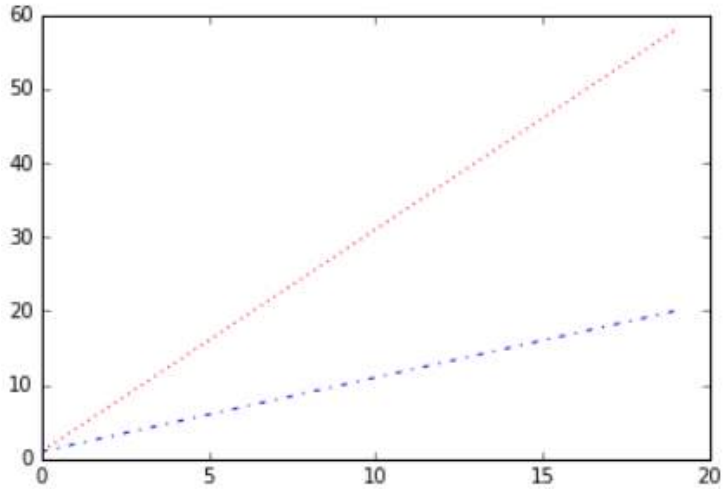
```
1  x1 = np.arange(20)
2  y1 = x1+1
3  y2 = x1*3+1
4
5  plt.plot(x1, y1, color='b')
6  plt.plot(x1, y2, c='r')
```



1. Matplotlib 시작하기

- 스타일 변경하기 (linestyle 지정하기)
  - linestyle, ls 사용

```
1  x1 = np.arange(20)
2  y1 = x1+1
3  y2 = x1*3+1
4
5  plt.plot(x1, y1, color='b', linestyle='-.')
6  plt.plot(x1, y2, c='r', ls=':')
```

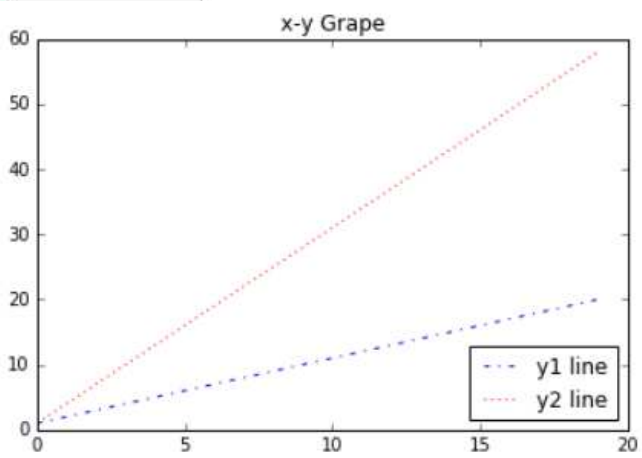


1. Matplotlib 시작하기

- 스타일 변경하기 (타이틀 및 범례 표시하기)
  - title, legend

```
x1 = np.arange(20)
y1 = x1+1
y2 = x1*3+1

plt.plot(x1, y1, color='b', linestyle='-.', label='y1 line')
plt.plot(x1, y2, c='r', ls=':', label='y2 line')
plt.title("x-y Grape")
plt.legend(loc="lower right")
```



1. Matplotlib 시작하기

모두의연구소

- Matplotlib 결과 그림파일로 저장하기

```
plt.savefig('figpath.svg')
```

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

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

```
fig.canvas.get_supported_filetypes()
```

```
{'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'}
```

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
markeredge color	mec	마커 선 색깔
markeredge width	mew	마커 선 굵기
markerfacec olor	mfc	마커 내부 색깔

2. Matplotlib의 여러가지 Plots

모두의연구소

- 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. Matplotlib의 여러가지 Plots

- 변수 개수 및 형태에 따른 그래프 종류

변수 개수	변수 형태	그래프
일변량 (변수 1개)	연속형 데이터	▪ 히스토그램 (Histogram) ▪ 커널 밀도 곡선 (Kernel Density Curve) ▪ 박스 그래프 (Box Plot) ▪ 바이올린 그래프 (Violin Plot)
	범주형 데이터 (명목형, 순서형)	▪ 막대 그림 (Bar Chart) ▪ 원 그림 (Pie Chart)
다변량 (변수 2개 이상)	**연속형 데이터**	▪ **산점도 (행렬) (Scatter Plot)** ▪ 선 그래프 (Line Plot) ▪ 시계열 그래프 (Time Series Plot)
	범주형 데이터	▪ 모자이크 그림 (Mosaic Chart)

[R 분석과 프로그래밍] http://rfriend.tistory.com

2. Matplotlib의 여러가지 Plots

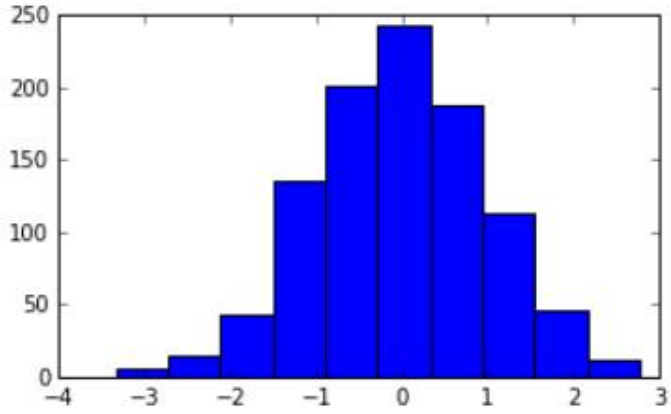
모두의연구소

- Histogram (히스토그램)
  - 값의 빈도를 분리해서 보여줌
  - 데이터 포인트는 분리되어 고른 간격의 막대로 표현되며, 데이터의 숫자가 막대 높이로 표현

데이터셋을 이해하는 가장 좋은 첫걸음은 히스토그램을 그려보는 것

> 각 변수들(데이터)의 분포를 확인

```
1  data = np.random.randn(1000)
2  plt.rcParams["figure.figsize"] = (5, 3)
3  plt.hist(data);
```



2. Matplotlib의 여러가지 Plots

모두의연구소

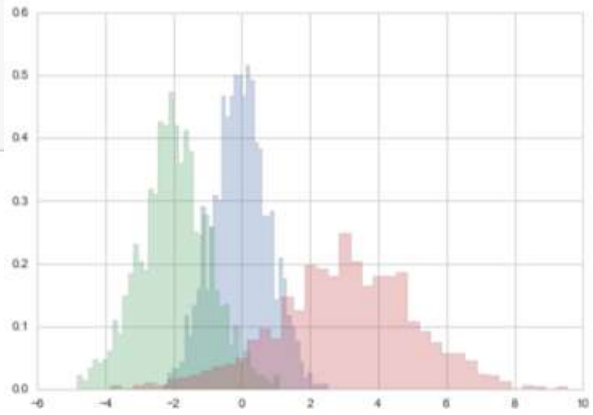
- Histogram (히스토그램)

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);
```

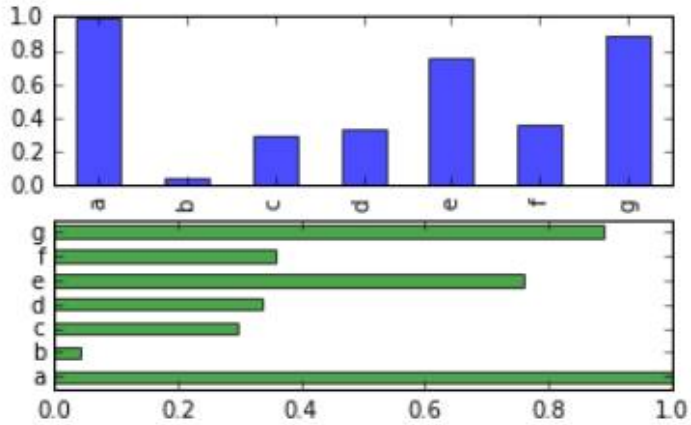


2. Matplotlib의 여러가지 Plots

- Bar Plot (바 플롯)
  - 범주형 데이터의 수치를 요약
  - 범주형 자료의 빈도수(상대도수)를 기둥의 높이로 표현하는 그래프

보고자 하는 데이터(x)가 **카테고리 값인 경우** 사용

```
1  fig, axes = plt.subplots(2, 1)
2  data = pd.Series(np.random.rand(7), index=list('abcdefg'))
3
4  data.plot.bar(ax=axes[0], color='b', alpha=0.7)
5  data.plot.barh(ax=axes[1], color='g', alpha=0.7)
```



2. Matplotlib의 여러가지 Plots

모두의연구소

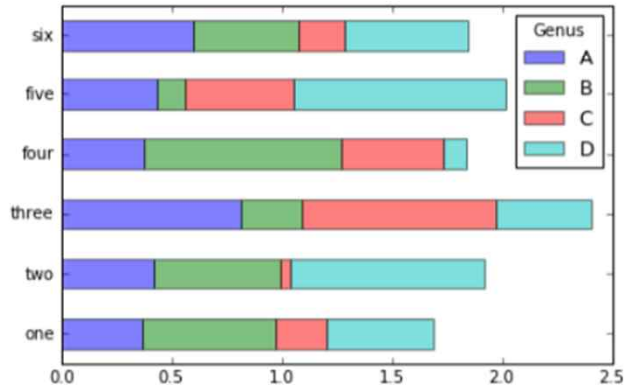
- Bar Plot (바 플롯)

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>
```



2. Matplotlib의 여러가지 Plots

모두의연구소

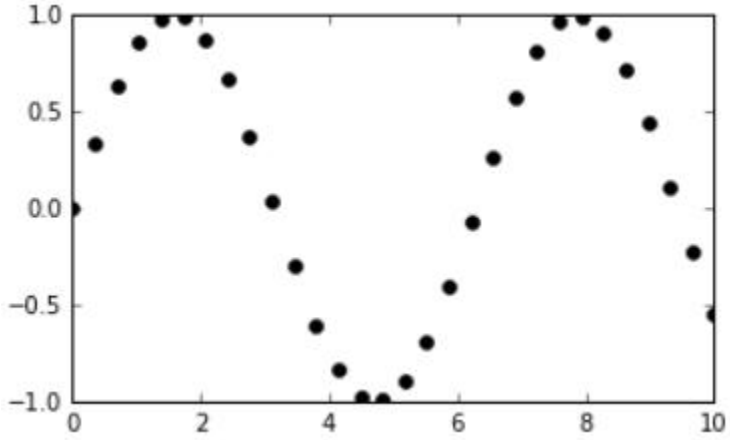
- Scatter (산점도)
  - x축과 y축에 연속형인 두 변수의 값을 점으로 뿌려준 그래프
  - 연속적인 두 변수 간의 관계를 파악하는데 유용함

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(*)

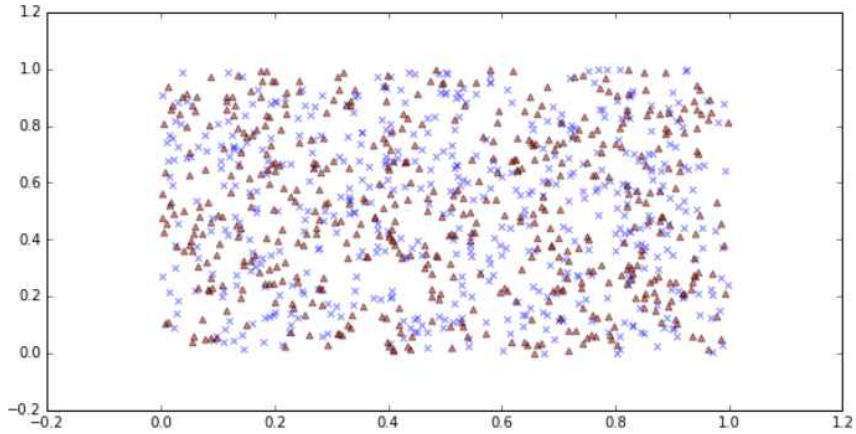
```
1  x = np.linspace(0, 10, 30)
2  y = np.sin(x)
3
4  plt.plot(x, y, 'o', color = 'black')
```



2. Matplotlib의 여러가지 Plots

- Scatter (산점도)

```
1  data_1 = np.random.rand(512, 2)
2  data_2 = np.random.rand(512, 2)
3
4  plt.figure(figsize=(10,5))
5  plt.scatter(data_1[:,0], data_1[:,1], c='b', marker='x', alpha=.5)
6  plt.scatter(data_2[:,0], data_2[:,1], c='r', marker='^', alpha=.5)
```



3. 최신 시각화 라이브러리

모두의연구소

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

모두의연구소

Scikit-learn

http://scikit-learn.org/stable/

Scikit-learn

Python의 대표적인 머신러닝 라이브러리

Scikit-learn

- 예제. 붓꽃의 품종 분류
  - 붓꽃의 품종을 알고 싶다 !
  - 어떤 품종인지 구분해 놓은 데이터를 이용하여 채집한 붓꽃의 품종을 예측
  - 지도학습
    - setosa, versicolor, virginica 종으로 분류한 측정 데이터 → Classification



1. 데이터 적재

모두의연구소

1. 데이터 적재 ¶

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

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

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

- 데이터 셋 설명

```
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.

```
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)
```

```
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,)
```

```
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,)
```

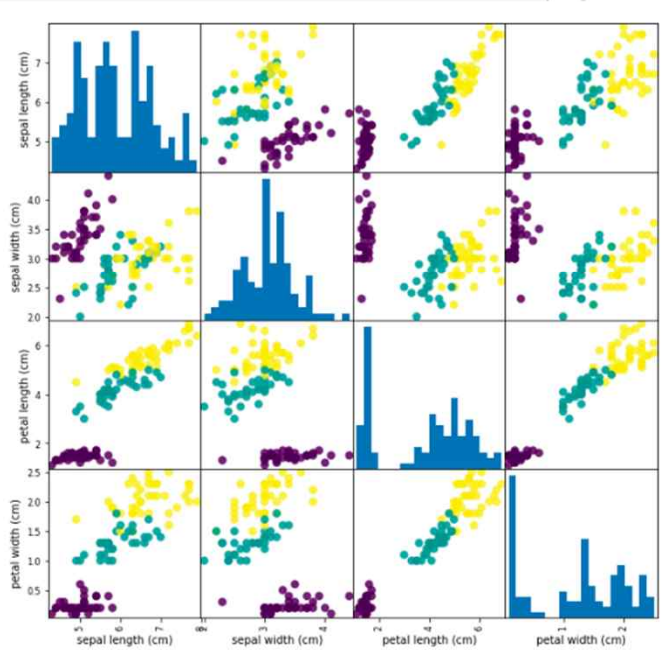
3. 데이터 탐색

모두의연구소

3. 데이터 탐색

```
import pandas as pd
%matplotlib inline
```

```
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

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

```
knn.fit(X_train, y_train)
```

```
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

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

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

```
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

```
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]
```

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

```
테스트 세트의 정확도: 0.97
```

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

```
테스트 세트의 정확도: 0.97
```

모두의연구소

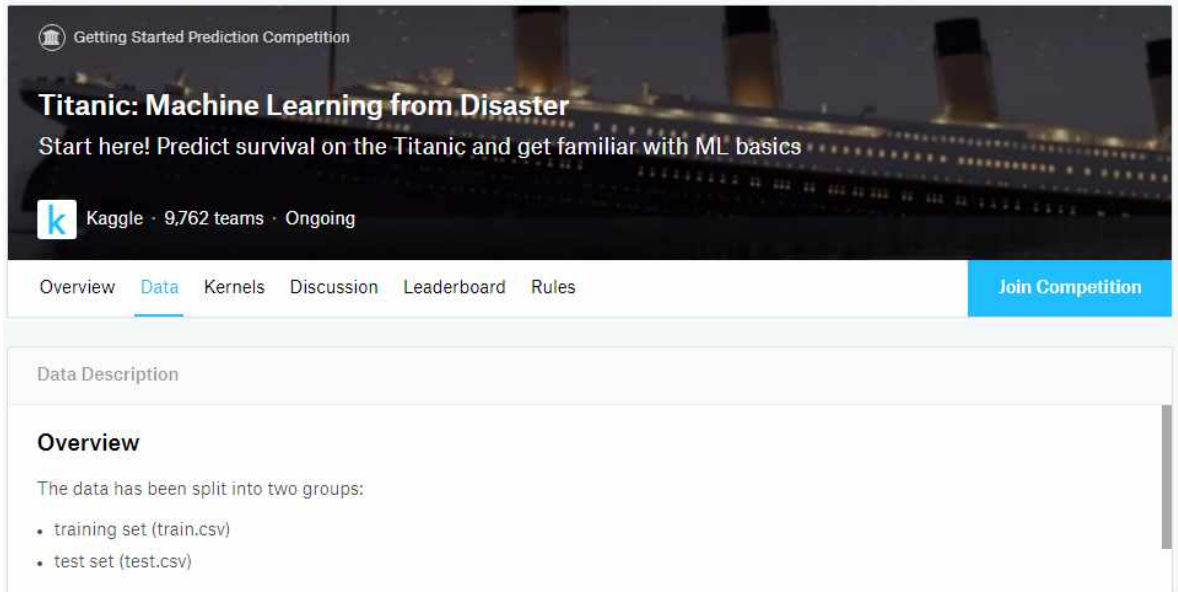
데이터 사이언스 실습(titanic)

Part 4-2

데이터 사이언스 실습(titanic)

모두의연구소

- Titanic: Machine Learning from Disaster (https://www.kaggle.com/c/titanic/data)
  - 타이타닉 호의 침몰 당시 승객 데이터를 이용하여 생존자 예측



데이터 사이언스 실습(titanic)

모두의연구소

- Titanic: Machine Learning from Disaster
  - 타이타닉 호의 침몰 당시 승객 데이터를 이용하여 생존자 예측
  - 데이터를 통해 생존자의 생사 여부와 다른 데이터들 간의 연관성을 분석하여 생존에 영향을 미치는 요소를 찾아내는 것
  - Dataset 정보
    - 생존자 이름, 성별, 나이, 티켓 요금, 생사 여부 등
  - train.csv, test.csv
    - 훈련 데이터 정보를 분석하여 model을 구성한 뒤 테스트 데이터의 생사 여부를 추측해보는 것

데이터 사이언스 실습(titanic)

모두의연구소

- Titanic: Machine Learning from Disaster
  - Dataset 정보
  - 생존자 이름, 성별, 나이, 티켓 요금, **생사 여부** 등

A	B	C	D	E	F	G	H	I	J	K	L
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20	0	0	A/5. 2151	8.05		S
14	0	3	Andersson, Mr. Anders Johan	male	39	1	5	347082	31.275		S

데이터 사이언스 실습(titanic)

모두의연구소

- Kernel (https://www.kaggle.com/ash316/eda-to-prediction-dietanic)

Contents of the Notebook:

Part1: Exploratory Data Analysis(EDA):

1)Analysis of the features.

2)Finding any relations or trends considering multiple features.

Part2: Feature Engineering and Data Cleaning:

1)Adding any few features.

2)Removing redundant features.

3)Converting features into suitable form for modeling.

Part3: Predictive Modeling

1)Running Basic Algorithms.

2)Cross Validation.

3)Ensembling.

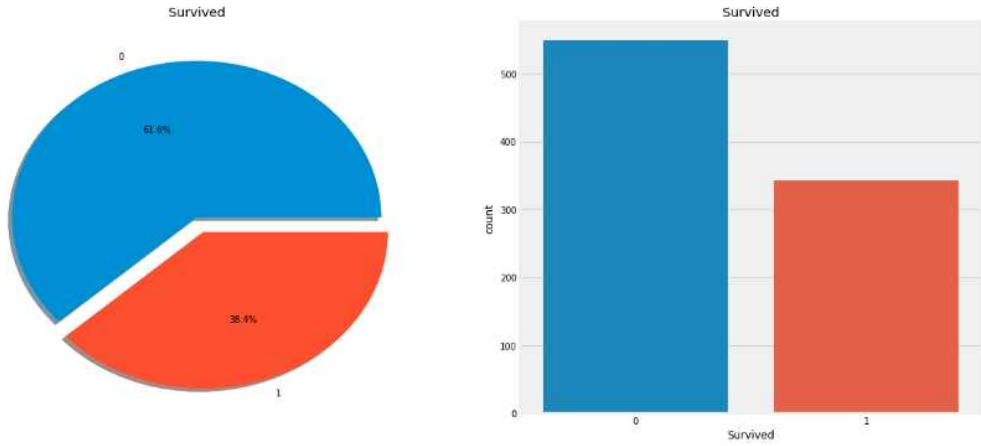
4)Important Features Extraction.

데이터 사이언스 실습(titanic)

모두의연구소

- Part 1. EDA
  - How Many Survived ?

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data['Survived'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],shadow=True
)
ax[0].set_title('Survived')
ax[0].set_ylabel('')
sns.countplot('Survived',data=data,ax=ax[1])
ax[1].set_title('Survived')
plt.show()
```



데이터 사이언스 실습(titanic)

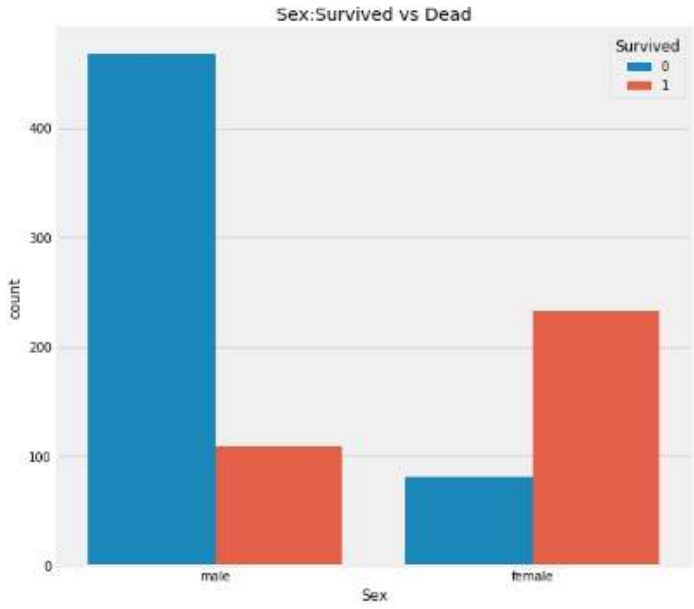
모두의연구소

- Part 1. EDA
  - Sex → Categorical Feature

```
data.groupby(['Sex','Survived'])['Survived'].count()
```

```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

```
f,ax=plt.subplots(1,2,figsize=(18,8))
data[['Sex','Survived']].groupby(['Sex']).mean().plot.bar(ax=ax[0])
ax[0].set_title('Survived vs Sex')
sns.countplot('Sex',hue='Survived',data=data,ax=ax[1])
ax[1].set_title('Sex:Survived vs Dead')
plt.show()
```



데이터 사이언스 실습(titanic)

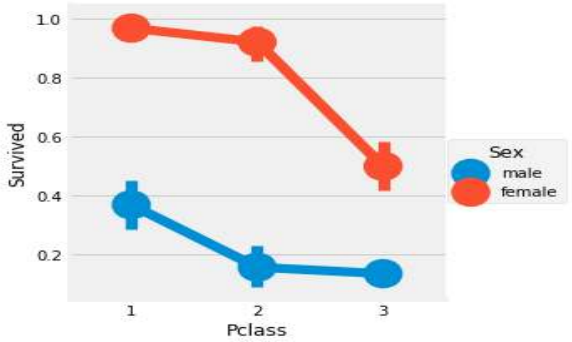
모두의연구소

- Part 1. EDA
  - Pclass → Ordinal Feature

```
pd.crosstab([data.Sex,data.Survived],data.Pclass,margins=True).style.backgr
ound_gradient(cmap='summer_r')
```

```
sns.factorplot('Pclass','Survived',hue='Sex',data=data)
plt.show()
```

	Pclass	1	2	3	All
Sex	Survived				
female	0	3	6	72	81
	1	91	70	72	233
male	0	77	91	300	468
	1	45	17	47	109
All		216	184	491	891

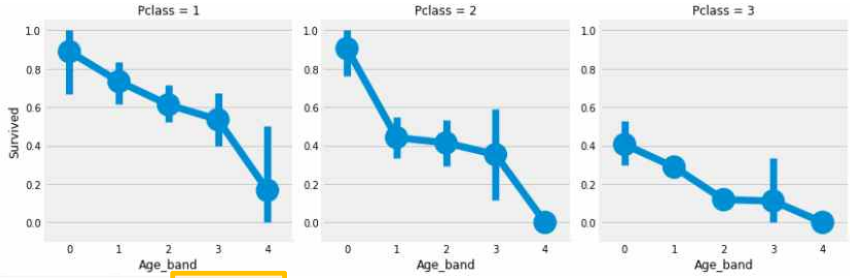


데이터 사이언스 실습(titanic)

모두의연구소

- Part 2. Feature Engineering and Data Cleansing
  - Age_band : Continuous value → Categorical value (5 Group)

```
data['Age_band']=0
data.loc[data['Age']<=16,'Age_band']=0
data.loc[(data['Age']>16)&(data['Age']<=32),'Age_band']=1
data.loc[(data['Age']>32)&(data['Age']<=48),'Age_band']=2
data.loc[(data['Age']>48)&(data['Age']<=64),'Age_band']=3
data.loc[data['Age']>64,'Age_band']=4
data.head(2)
```



Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Initial	Age_band
0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	Mr	1
1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	Mrs	2

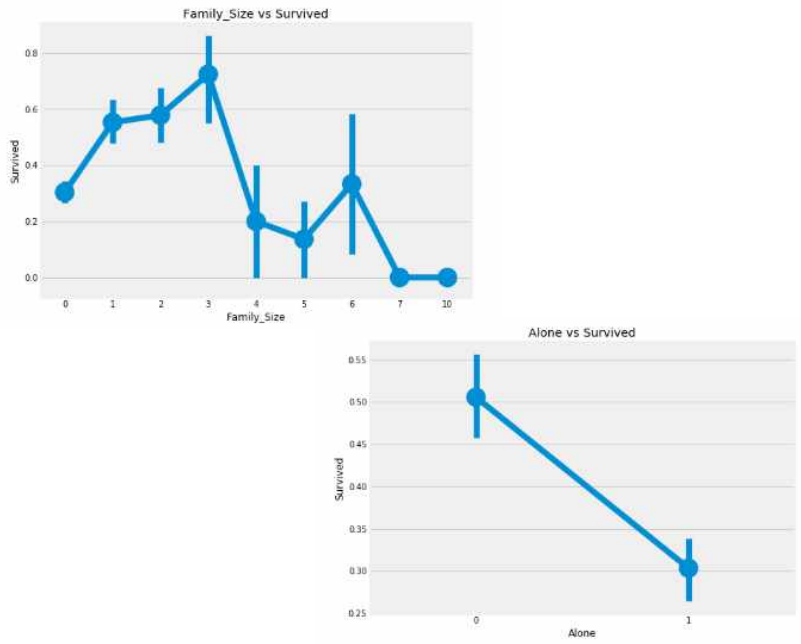
데이터 사이언스 실습(titanic)

모두의연구소

- Part 2. Feature Engineering and Data Cleansing
  - Family_Size and Alone

```
data['Family_Size']=0
data['Family_Size']=data['Parch']+data['SibSp']#family size
data['Alone']=0
data.loc[data.Family_Size==0,'Alone']=1#Alone

f,ax=plt.subplots(1,2,figsize=(18,6))
sns.factorplot('Family_Size','Survived',data=data,ax=ax[0])
ax[0].set_title('Family_Size vs Survived')
sns.factorplot('Alone','Survived',data=data,ax=ax[1])
ax[1].set_title('Alone vs Survived')
plt.close(2)
plt.close(3)
plt.show()
```



Sibsp: 함께 탑승한 형제자매, 배우자의 수
Parch: 함께 탑승한 부모, 자식의 수

데이터 사이언스 실습(titanic)

모두의연구소

- Part 2. Feature Engineering and Data Cleansing

Converting String Values into Numeric

Since we cannot pass strings to a machine learning model, we need to convert features loke Sex, Embarked, etc into numeric values.

```
data['Sex'].replace(['male','female'],[0,1],inplace=True)
data['Embarked'].replace(['S','C','Q'],[0,1,2],inplace=True)
data['Initial'].replace(['Mr','Mrs','Miss','Master','Other'],[0,1,2,3,4],inplace=True)
```

Dropping UnNeeded Features ¶

Name--> We don't need name feature as it cannot be converted into any categorical value.

Age--> We have the Age_band feature, so no need of this.

Ticket--> It is any random string that cannot be categorised.

Fare--> We have the Fare_cat feature, so unneeded

Cabin--> A lot of NaN values and also many passengers have multiple cabins. So this is a useless feature.

Fare_Range--> We have the fare_cat feature.

PassengerId--> Cannot be categorised.

데이터 사이언스 실습(titanic)

모두의연구소

- Part 3. Predictive Modeling

We have gained some insights from the EDA part. But with that, we cannot accurately predict or tell whether a passenger will survive or die. So now we will predict the whether the Passenger will survive or not using some great Classification Algorithms.Following are the algorithms I will use to make the model:

1)Logistic Regression

2)Support Vector Machines(Linear and radial)

3)Random Forest

4)K-Nearest Neighbours

5)Naive Bayes

6)Decision Tree

7)Logistic Regression

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