NUMPY

Numpy

- **Numpy**
  - Numerical Python
  - 산술 계산을 위한 패키지
    - 다차원 배열인 ndarray는 빠른 배열 계산과 유연한 브로드캐스팅 기능을 제공한다.
    - 반복문을 작성할 필요 없이 전체 데이터 배열을 빠르게 계산할 수 있는 표준 수학 함수
    - 선형대수, 난수 생성기, 푸리에 변환 가능
  - Numpy는 일반적인 산술 데이터 처리를 위한 기본 라이브러리를 제공하기때문에 통계나 분석데이터를 처리하기 위해선 Pandas를 사용해야 한다.
  - Numpy가 파이썬 산술 계산 영역에서 중요한 위치를 차지하는 이유중 하나는 대용량 데이터 배열을 효율적으로 다룰수 있도록 설계되어 있다.

Numpy

- **Numpy 사용**
  - np 라는 이름으로 자주 사용 된다.
  - List 와 numpy 자료형은 서로 다른 객체

```
import numpy as np

my_arr = np.arange(100000)
my_list = list(range(100000))
```

Numpy

- **Numpy 사용**
  - 각각 배열과 리스트에 원소 2 를 곱해보자
  - 속도 차이 비교
  - %time 이라는 매직 함수 사용

```
%time
for _ in range(10):
    my_arr2 = my_arr*2
```

```
Wall time: 0 ns
```

```
%time for _ in range(10):my_arr2 = my_arr*2
```

```
Wall time: 2 ms
```

```
%time for _ in range(10):my_list2 = [ i*2 for i in my_list]
```

```
Wall time: 69 ms
```

Numpy

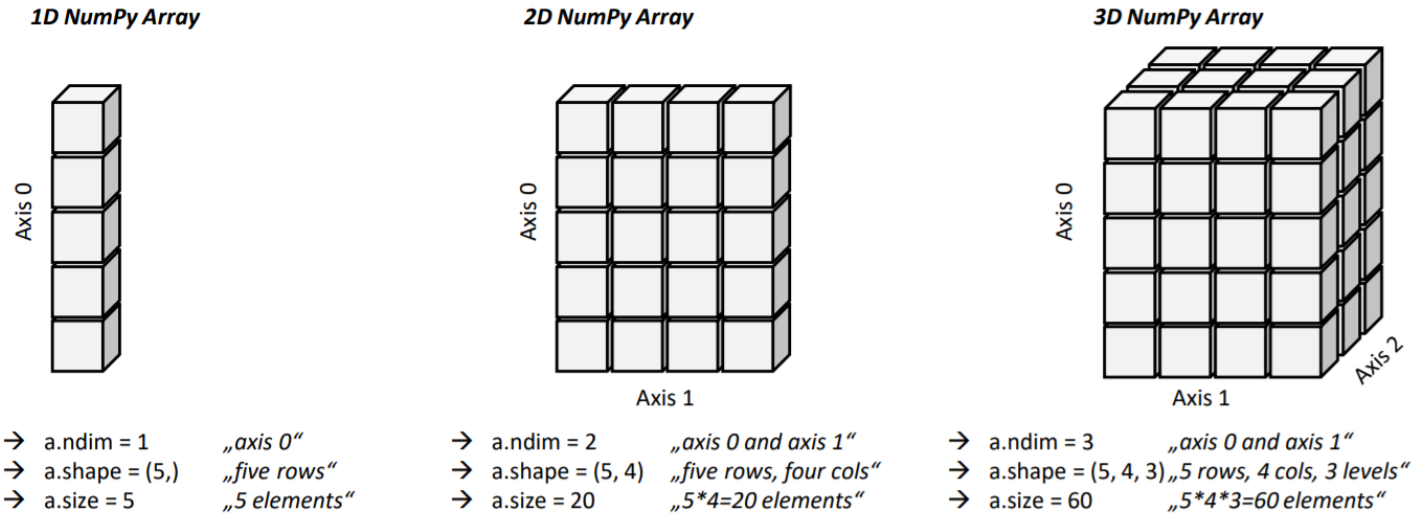
- **Numpy 사용**
  - 각각 배열과 리스트에 원소 2 를 곱해보자
  - 속도 차이 비교
  - %time 이라는 매직 함수 사용

```
import numpy as np

my_arr = np.arange(100000)
my_list = list(range(100000))
```

Numpy

- **다차원 배열 객체**
  - Numpy 는 ndarray라고 하는 N차원 배열 객체를 사용한다.
  - 배열은 스칼라 원소 간의 연산에서 사용하는 문법과 비슷한 방식을 사용해 전체 데이터 블록에 수학적인 연산을 수행할 수 있도록 해준다.
  - 사용
    - Vectors, Matrices, Tensors, Images, Etc



Numpy

- **다차원 배열 객체**
  - 2x3 배열을 랜덤한 스칼라로 만든다.

```
# 2x3 배열을 랜덤한 스칼라로 만든다.
data = np.random.randn(2,3)
```

```
data
```

```
array([[ 0.06546519,  0.58256402,  0.80577224],
       [ 1.27401936,  0.09533105, -1.99959539]])
```

Numpy

- **ndarray 생성하기**
  - array 함수를 이용해 배열을 생성할 수 있다.
  - list 생성후 array 함수 적용

```
data1 = [6,7,3,4,2]
```

```
type(data1)
```

```
list
```

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

```
arr1
```

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

Numpy

- **ndarray 생성하기2**
  - np.zeros : 메모리를 초기화 하고 0 으로 채워 넣는다.
  - np.empty : zeros와 비슷하지만, 메모리를 초기화 하지 않고 빈값을 넣기 때문에 0 이외의 값이 들어갈수 있다.
  - np.ones : 메모리를 초기화 하고 1 로 채워 넣는다.
  - np.full : 특정 값으로 채워 넣는다.
  - np. eye : NxN 단위행렬을 생성한다.
  - object.asdtype() : 함수를 통해 dtype을 변경할수 있다. (공식 문서 참고)

Numpy

- **numpy 배열의 산술 연산**
  - numpy 배열은 반복문을 작성하지 않고 데이터를 일괄 처리할 수 있다.
  - 이를 **벡터화** 라고 한다.
  - **브로드 캐스팅**과는 다른 개념

```
arr1 = np.array([[1,2,3],
                 [4,5,6]])
```

```
# 요소곱
arr1 * arr1
```

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

```
# 요소합
arr1 + arr1
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

```
# 나눗셈
1/arr1
```

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

```
arr1 >= arr1
```

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

```
# 행렬곱
np.dot(arr1,np.transpose(arr1))
```

```
array([[14, 32],
       [32, 77]])
```

Numpy

- **벡터화 연산 적용하기**
  - 배열의 여러 원소에 어떤 함수를 적용하고 싶다.
  - 함수를 벡터화 시킨후 적용한다.
  - vectorize 는 원소를 순회하는 for 루프를 구현한것으로 성능이 좋진 않다.
  - 브로드 캐스팅이 훨신 효율적이다.

```
# 행렬을 만듭니다.
mat_1 = np.array(range(1,10)).reshape(3,3)

# 100을 더하는 함수를 만듭니다.
add_100 = lambda i: i + 100

# 벡터화된 함수를 만듭니다.
vectorized_add_100 = np.vectorize(add_100)

# 행렬의 모든 원소에 함수를 적용합니다.
vectorized_add_100(mat_1)
```

```
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

```
vectorized_add_100([[[1,2]]])
```

```
array([[[101, 102]]])
```

```
# 모든 원소에 100을 더합니다.
mat_1 + 100
```

```
array([[101, 102, 103],
       [104, 105, 106],
       [107, 108, 109]])
```

Numpy

- **인덱싱과 슬라이싱**
  - Sequential 데이터 이기 때문에 인덱싱과 슬라이싱이 가능하다.

```
arr_1 = np.arange(10)
```

```
arr_1[5]
```

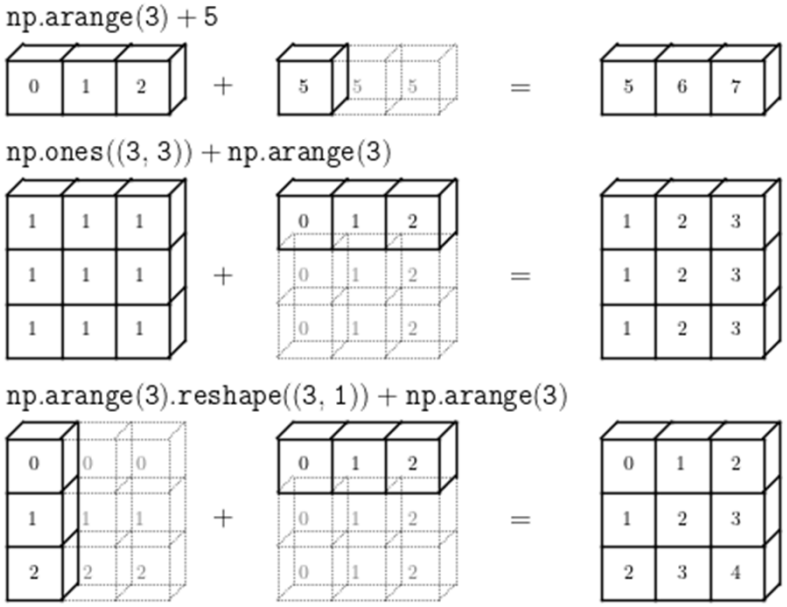
```
5
```

```
arr_1[5:8]
```

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

Numpy

- **다차원 배열 객체**
  - array에 스칼라 값을 대입하면 영역 전체로 브로드캐스팅 된다.
  - list와의 차이점은 배열 조각은 원본 배열의 View이다.즉, 데이터는 복사되지 않고 View에 대한변경은 그대로 원본 배열에 반영 된다.
  - View 대신 ndarray의 복사본을 얻고 싶다면 object.copy() 함수를 사용해 명시적으로 배열을 복사해야 한다.



Numpy

- **배열 전치와 축 바꾸기**
  - 배열 전치는 데이터를 복사하지 않고 데이터의 모양이 바뀐 뷰를 반환하는 기능
  - np.T 또는 np.transepose() 를 사용하여 변환한다.

```
arr
```

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

```
arr.T
```

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

```
arr.transpose()
```

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

Numpy

- **단항 유니버설 함수**
  - exp, sqrt
- **이항 유니버설 함수**
  - maximum
  - minimum
  - add

Numpy

- **수학 메서드와 통계 메서드**
  - 합, 평균, 표준편차 를 구할수 있다.
  - sum : 배열 전체 혹은 특정 축에 대한 모든 원소의 합을 계산한다. 크기가 0 인 배열에 대한 sum 결과는 0 이다.
  - mean : 산술 평균을 구한다. 크기가 0인 배열에 대한 mean 결과는 Nan이다.
  - std, var : 각각 표준편차(std)와 분산(var)을 구한다. 선택적으로 자유도를 줄 수 있으며 분모의 기본 값은 n 이다.
  - min, max :최솟값과 최댓값
  - argmin, argmax : 최소 원소의 색인값과 최대 원소의 색인값
  - cumsum : 각 원소의 누적합
  - cumprod : 각 원소의 누적곱

Numpy

- **집합 관련함수**
  - numpy는 1차원 ndarray를 위한 몇 가지 기본적인 집합 연산을 제공한다.
  - unique : 중복된 원소릴 제거한뒤 정렬하여 반환
  - intersec1d : 배열 x와 y에 중복
  - union1d : 두 배열의 합집합
  - setdiff1d :두 배열의 차집합

- **배열 데이터의 파일 입출력**
  - np.save
  - np.load

```
arr1 = np.arange(10)
np.save('save_array', arr1)
```

```
arr2 = np.load('save_array.npy')
```

Numpy

- **linalg 모듈**
  - 행렬의 분할과 역행렬, 행렬식과 같은 함수를 포함하고 있다.
  - diag : 정사각 행렬의 대각/비대각 원소를 1차원 배열로 반환하거나, 1차원 대각선 원소로 하고 나머지는 0으로 채운 단위행렬을 반환한다.
  - dot : 행렬 곱셈
  - trace : 행렬의 대각선 원소의 합을 계산 한다.
  - det : 행렬식을 계산한다.
  - eig : 정사각 행렬의 고윳값과 고유 벡터를 계산하다.
  - inv : 정사각 행렬의 역행렬을 계산한다.
  - qr : QR분해를 계산한다.
  - svd : 특잇값 분해(SVD)를 계산한다.
  - solve : A가 정사각 행렬일때 Ax = b를 만족하는 x를 구한다.
  - lstsq : Ax = b 를 만족하는 최소 제곱해를 구한다.

Numpy

- **난수생성**
  - np.random.radint :균일 분포의 정수 난수 1개 생성
  - np.random.rand : 0부터 1사이의 균일 분포에서 난수 생성
  - np.random.rnadn : 가우시안 표준 정규 분포에서 난수 matrix array 생성

- **유니버설 함수 :배열의 각원소를 빠르게 처리하는 함수**
  - ufunc 이라고 불리기도 하는 유니버설 함수는 ndarray 안에 있는 데이터 원소별로 연산을 수행하는 함수 이다.
  - 유니버설 함수는 하나 이상의 스칼라 값을 받아서 하나 이상의 스칼라 결과값을 반환하는 간단한 함수를 고속으로 수행

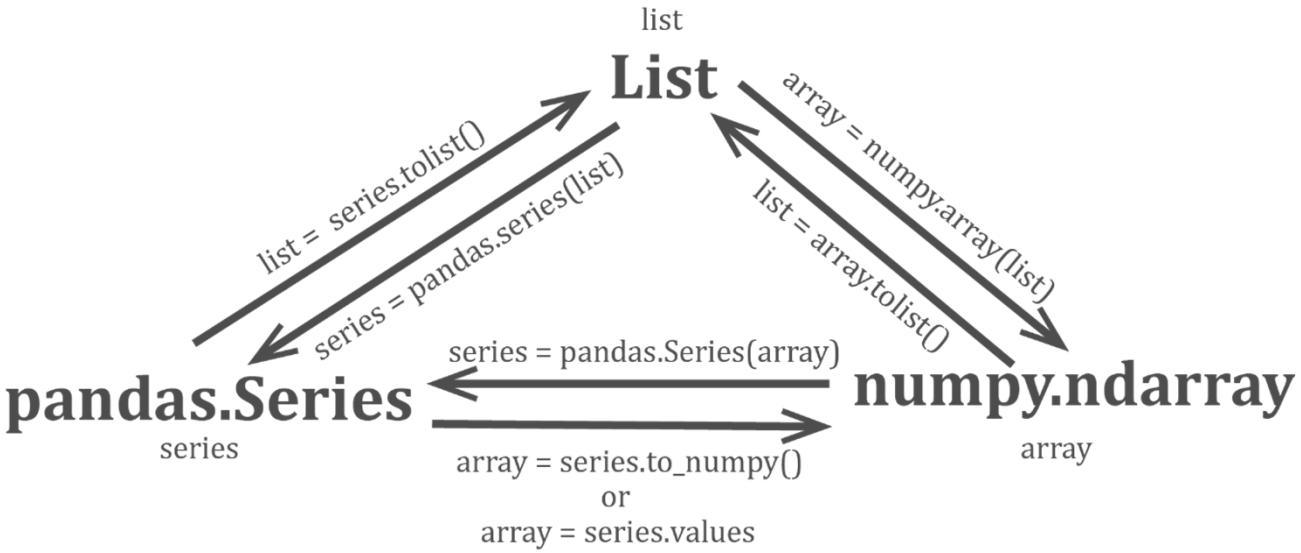
PANDAS

Pandas

- 파이썬 데이터 분석 라이브 러리
- 빠른 연산속도
- 다양한 포맷의 처리 가능
- 다양한 통계, 그래픽 함수 사용가능

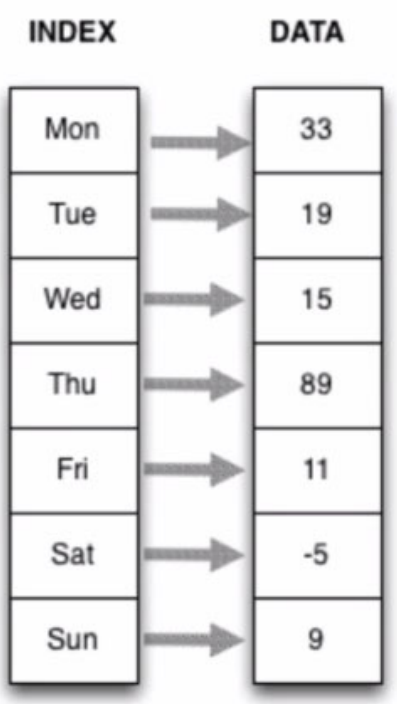
Pandas 자료구조

- **판다스 자료구조**
  - 다른 형식을 갖는 여러 종류의 데이터를 동일한 형식으로 변경
  - Series 와 DataFrame 을 통해 자료를 통일 해준다.
  - Series : 1차원 Array
  - DataFrame : 2차원 Array



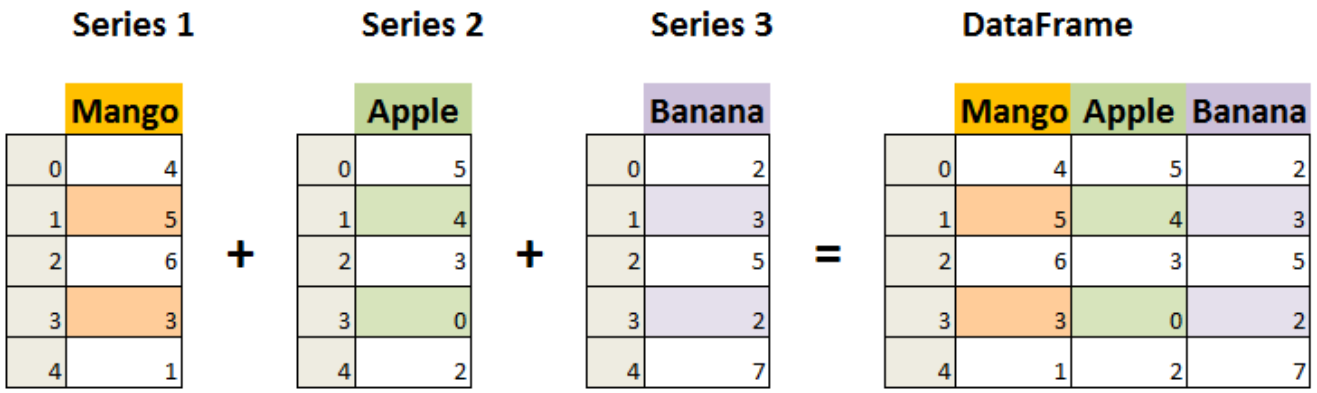
Pandas

- **Series**
  - index : value 형태로 일대일 대응
  - dictionary 와 비슷한 구조



Pandas

- **DataFrame**
  - 2차원 Array, Matrix 라고 하기도 한다.
  - row 와 column으로 구성, Series 는 row vector 라고 하기도 한다



Pandas

- **DataFrame 의 속성**

	homeTeamName	awayTeamName	startTime	duration_minutes	attendance	dayNight
0	Cubs	Reds	2016-07-05 18:20:00 UTC	188	41310	D
1	Indians	Astros	2016-09-08 16:10:00 UTC	194	15275	D
2	Padres	Giants	2016-09-25 20:40:00 UTC	185	28456	D
3	Diamondbacks	Brewers	2016-08-07 20:10:00 UTC	211	24021	D
4	Giants	Dodgers	2016-04-09 20:05:00 UTC	202	41224	D
5	Blue Jays	Indians	2016-07-02 17:07:00 UTC	199	46197	D
6	Reds	Pirates	2016-04-09 17:10:00 UTC	180	22799	D
7	Cubs	Mariners	2016-07-30 18:20:00 UTC	157	41401	D
8	Rockies	Phillies	2016-07-10 20:10:00 UTC	191	32113	D
9	Yankees	Blue Jays	2016-05-26 20:05:00 UTC	154	38391	D

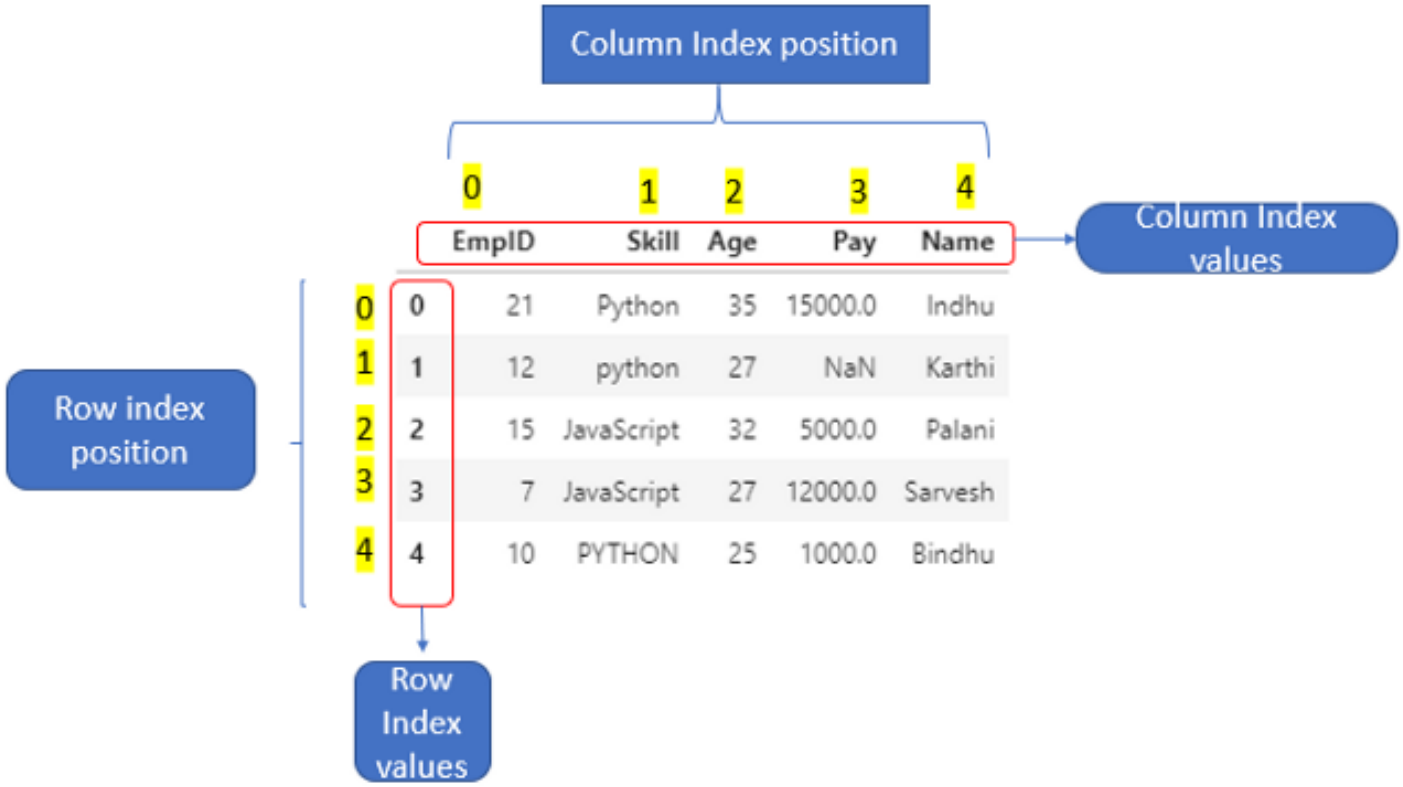
Pandas

- **DataFrame 의 인덱싱/슬라이싱**
  - 슬라이싱
    - DataFrame.iloc[start index: end index : step]
  - 인덱싱
    - DataFrame.loc[row index, col name]
    - DataFrame.iloc[row num, col num]

```
In [8]: AAPL.iloc[:5,:]
Out[8]:
              Open    High     Low   Close     Volume  Adj Close
Date
2014-09-16   99.80  101.26   98.89  100.86   66818200     100.86
2014-09-15  102.81  103.05  101.44  101.63   61216500     101.63
2014-09-12  101.21  102.19  101.08  101.66   62626100     101.66
2014-09-11  100.41  101.44   99.62  101.43   62353100     101.43
2014-09-10   98.01  101.11   97.76  101.00  100741900     101.00
```

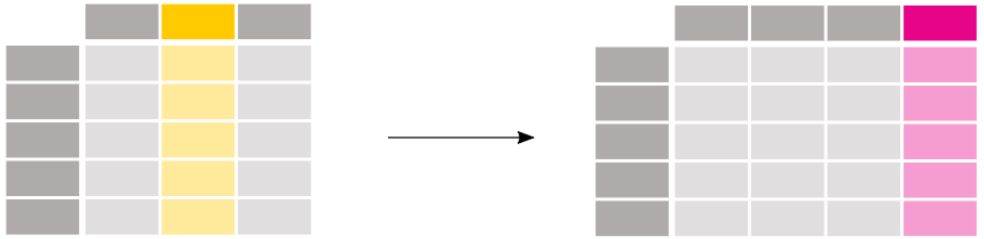
Pandas

- **DataFrame 의 인덱싱/슬라이싱**



Pandas

- **인덱스 설정**
  - DataFrame 추가/수정
    - 행추가
    - DataFrame.loc['새로운행 이름']= data(또는 배열)
    - DataFrame.loc[row index, col name] = data
    - DataFrame.iloc[row num, col num] = data
  - DataFrame 행과 열 변경
  - 행과 열 위치 변경
    - DataFrame.transpose()
    - DataFrame.T



Pandas

- **인덱스 설정**
  - DataFrame.set_index(['col name'], inplace=True)
    - inplace : 원본 데이터에 변경사항을 적용하기 위한 옵션
  - DataFrame.reindex([new_index], fill_value=<value>)
    - 인덱스를 새로운 이름으로 재지정
    - fill_value : 새로운 인덱스를 생성하고 어떤 값으로 채울지 결정
  - DataFrame.sort_index() : 행 인덱스를 기준으로 데이터를 정렬
    - ascending=False : 내림차순 정렬
    - ascending=True : 오름차순 정렬
    - DataFrame.sort_values() : 특정 열을 기준으로 정렬

Pandas

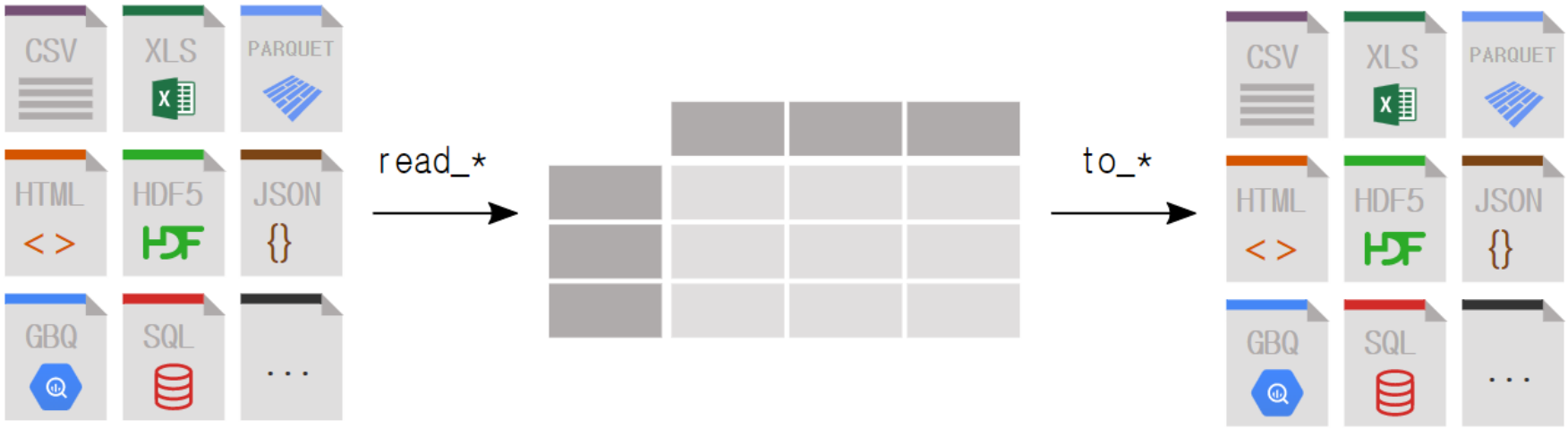
- **산술 연산**
  - 시리즈 연산
    - Series to number
      - <Series> <연산자:*+-/) <num>
    - Series to Series
      - <Series> <연산자:*+-/) <Series>
  - 참고
    - Nan <연산> <num> = Nan
    - 해결 : 연산 메소드 사용
    - Series.add(<Series2>, fill_value=0)

Pandas

- **데이터 프레임 연산**
  - DF to number
    - <DF> <연산자:*+-/) <num>

Pandas

- **외부파일 읽어오기**
  - 다양한 형태의 외부 파일을 읽어와서 데이터프레임으로 변환하는 함수를 제공



Pandas

- **인덱스 설정**
  - DF.head(n) : n번째 row 까지 미리 보기
  - DF.tail(n) : 마지막으로 부터 n 번째 row 까지 미리 보기
  - DF.shape : DF의 크기 정보 확인
  - DF.info() : DF의 기본 정보 확인
    - 클래스 유형, row index의 구성, col 이름의 종류/개수, 각 col 의 자료형과 개수
  - DF.dtypes : 각 col 의 자료형 확인
  - DF.describe() : 평균, 표준편차, 최댓값, 최솟값, 중간값 등을 요약

Pandas

- **통계 함수 적용**
  - DF.mean() : 모든 열의 평균값
  - DF["col_name"].mean() : 특정 열의 평균값
  - DF.median() : 모든 열의 중간값
  - DF["col_name"].median() : 특정 열의 중간값
  - DF.max() : 모든 열의 최댓값
  - DF["col_name"].max() : 특정 열의 최댓값
  - DF.min() : 모든 열의 최솟값
  - DF["col_name"].min() : 특정 열의 최솟값

Pandas

- **인덱스 설정**
  - DF.std() : 표준편차
  - DF["col_name"].corr() : 특정 열의 표준 편차

  - DF.std() : 모든 열의 상관계수
  - DF["col_name"].corr() : 특정 열의 상관계수

Pandas

- **인덱스 설정**
  - DF.std() : 표준편차
  - DF["col_name"].std() : 특정 열의 표준 편차

  - DF.std() : 모든 열의 상관계수
  - DF["col_name"].corr() : 특정 열의 상관계수

Pandas

- **표준 편차**
  - 자료의 관찰값이 얼마나 흩어져 있는지 나타내는 값
  - 표준 편차 0에 가까울수록 데이터가 평균에 많이 분포되어 있다.

[편차] $d = x - \bar{x}$

[분산] $s^2 = \frac{\sum(x-\bar{x})^2}{n-1}$

[표준편차] $s = \sqrt{\frac{\sum(x-\bar{x})^2}{n-1}}$

d : 편차
x : 변인
$\bar{x}$: 표본의 평균
n : 표본의 크기
s^2 : 분산
s : 표준편차

Pandas

- **상관계수**
  - cov (공분산) : 두 확률 변수의 선형관계를 나타내는 값.
    - cov > 0 : x 가 증가할때 y 도 증가한다.
    - cov < 0 : x 가 증가할때 y는 감소한다.
    - cov = 0 : x와 y는 독립이다.
  - 상관 계수 : 상관 계수는 공분산의 단위화

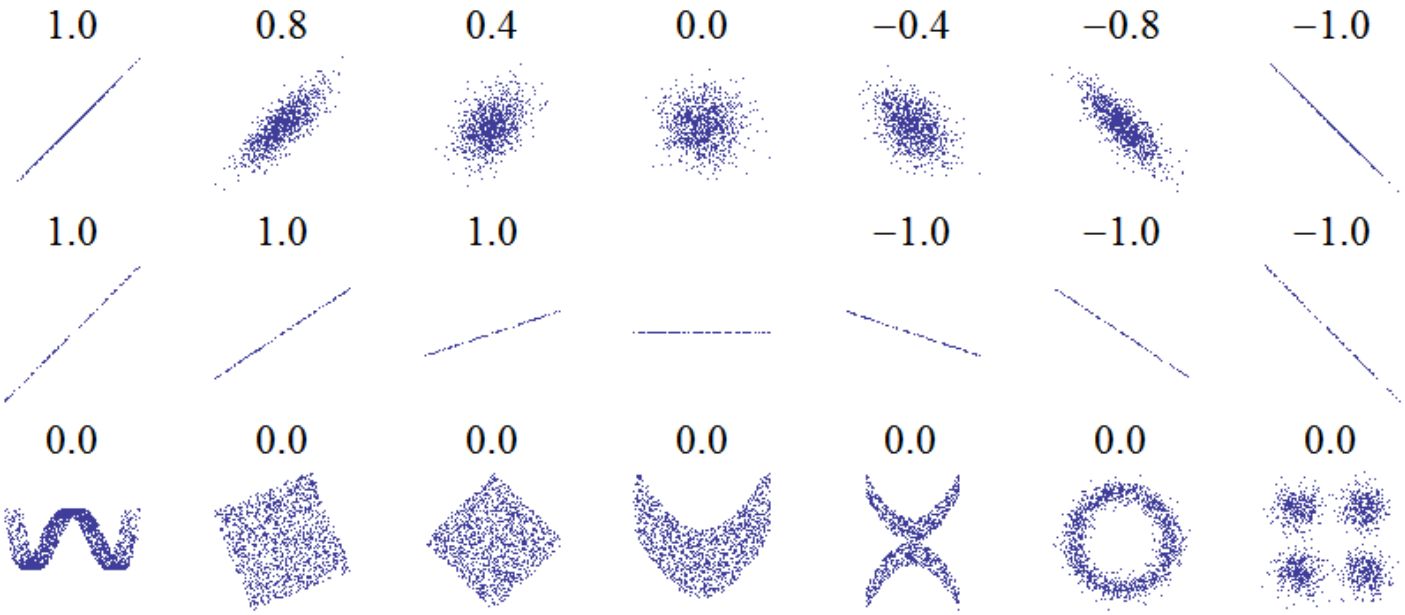
$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

$$= \frac{\mathbf{E}[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

$$= \frac{\mathbf{E}(XY) - \mathbf{E}(X)\mathbf{E}(Y)}{\sigma_X \sigma_Y}$$

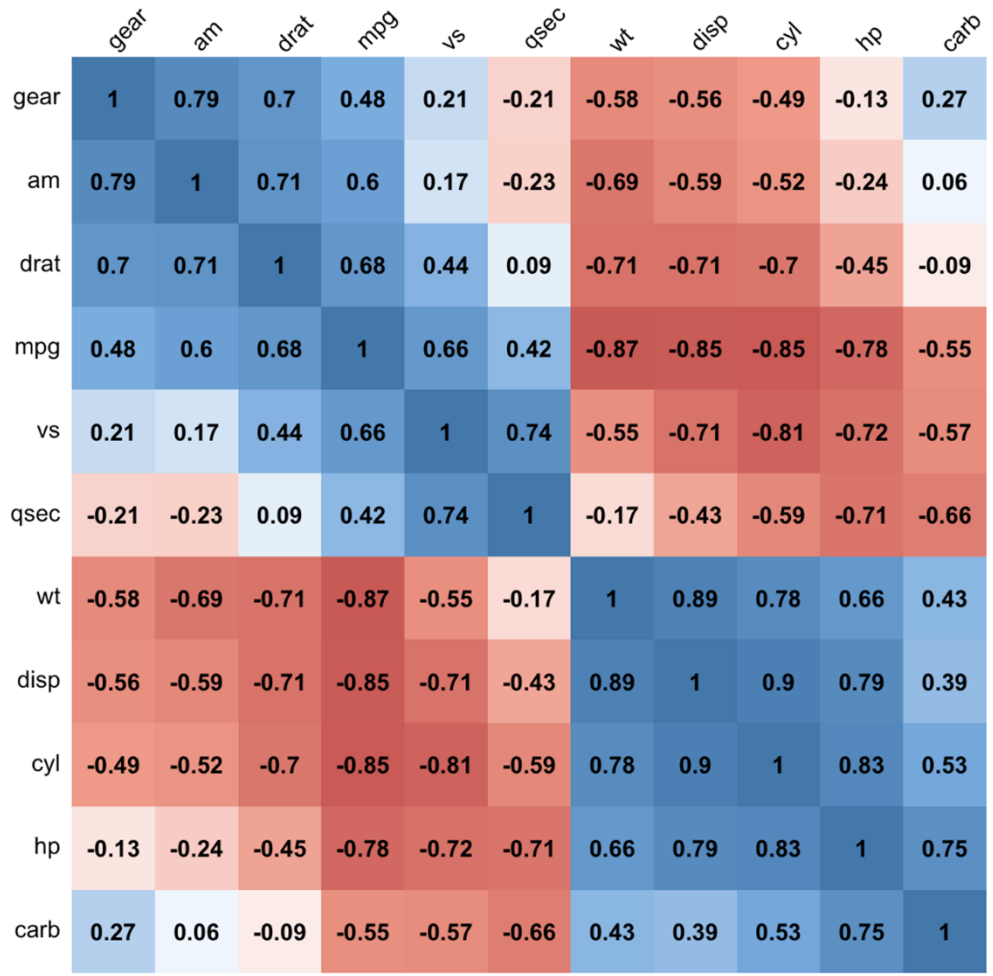
Panda

- 상관 계수별 산포도



Pandas

- **Feature 별 상관계수**



	gear	am	drat	mpg	vs	qsec	wt	disp	cyl	hp	carb
gear	1	0.79	0.7	0.48	0.21	-0.21	-0.58	-0.56	-0.49	-0.13	0.27
am	0.79	1	0.71	0.6	0.17	-0.23	-0.69	-0.59	-0.52	-0.24	0.06
drat	0.7	0.71	1	0.68	0.44	0.09	-0.71	-0.71	-0.7	-0.45	-0.09
mpg	0.48	0.6	0.68	1	0.66	0.42	-0.87	-0.85	-0.85	-0.78	-0.55
vs	0.21	0.17	0.44	0.66	1	0.74	-0.55	-0.71	-0.81	-0.72	-0.57
qsec	-0.21	-0.23	0.09	0.42	0.74	1	-0.17	-0.43	-0.59	-0.71	-0.66
wt	-0.58	-0.69	-0.71	-0.87	-0.55	-0.17	1	0.89	0.78	0.66	0.43
disp	-0.56	-0.59	-0.71	-0.85	-0.71	-0.43	0.89	1	0.9	0.79	0.39
cyl	-0.49	-0.52	-0.7	-0.85	-0.81	-0.59	0.78	0.9	1	0.83	0.53
hp	-0.13	-0.24	-0.45	-0.78	-0.72	-0.71	0.66	0.79	0.83	1	0.75
carb	0.27	0.06	-0.09	-0.55	-0.57	-0.66	0.43	0.39	0.53	0.75	1

Pandas

- **인덱스 설정**
  - Pandas 내장 그래픽 도구
  - - 선, 막대, 히스토그램, 박스플롯, 파이, 산점도, 등등 가능