/* elice */

양재 AI School 인공지능 캠프

머신러닝 (Machine Learning)을 위한 라이브러리

박상수 선생님

커리큘럼

1 **머신러닝 라이브러리 개요**

머신러닝 라이브러리에 대한 개요 및 활용에 대해 배워보고
활용하는 방법에 대해 알아봅니다.

2 **NumPy, Pandas 활용 및 데이터 시각화**

머신러닝의 구현에 사용되는 라이브러리인 NumPy, Pandnas 에 대해서
학습하고, 시각화 라이브러리인 Matplotlib, Seabron 라이브러리를 활용하여
시각화 합니다.

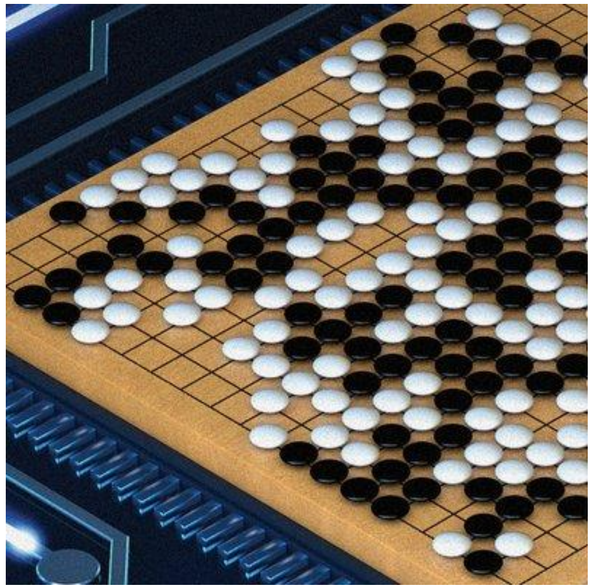
목차

1. Machine Learning 개요
2. Machine Learning과 수학
3. 기본기: NumPy, Matplotlib
4. 고급기: Pandas, Seaborn

Machine Learning 이란 ?

러닝머신 (Running Machine)이 아니라 머신러닝

Machine Learning이 적용된 분야

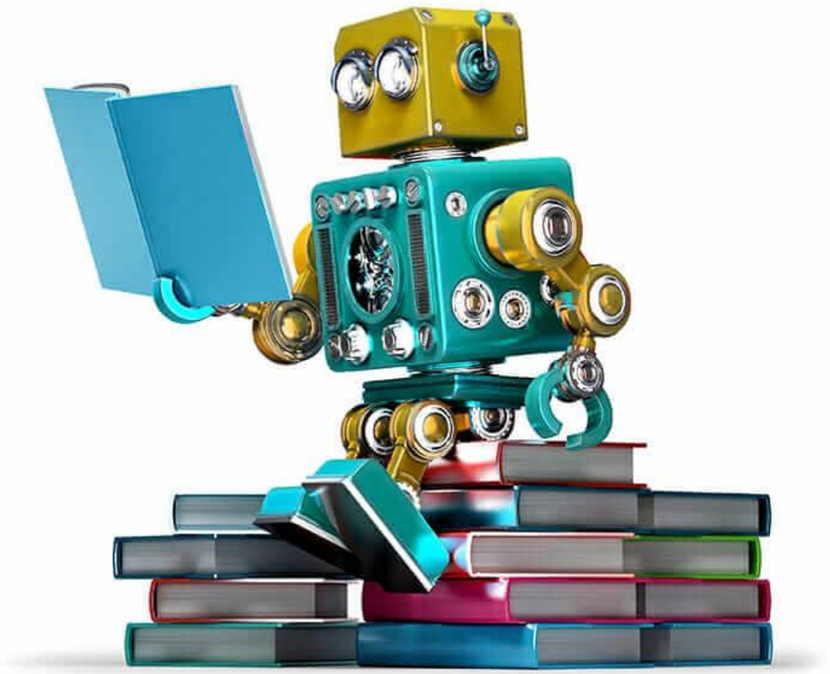


알파고

자율주행차

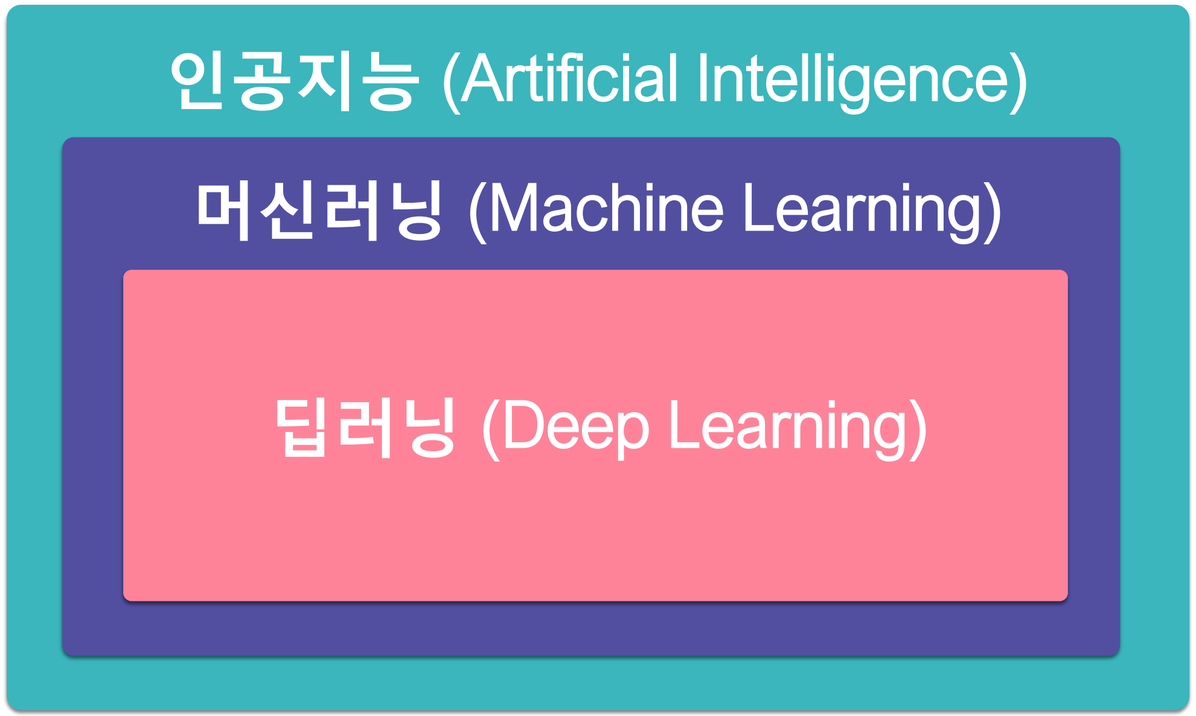
인공지능 스피커

Machine Learning이란 ?



머신러닝은 프로그램의 한 종류로 데이터를
학습할 수 있는 알고리즘을 주어서 컴퓨터가 어떠한 경우에
대해서 **예측**할 수 있도록 하는 프로그래밍 방법

Machine Learning이란 ?

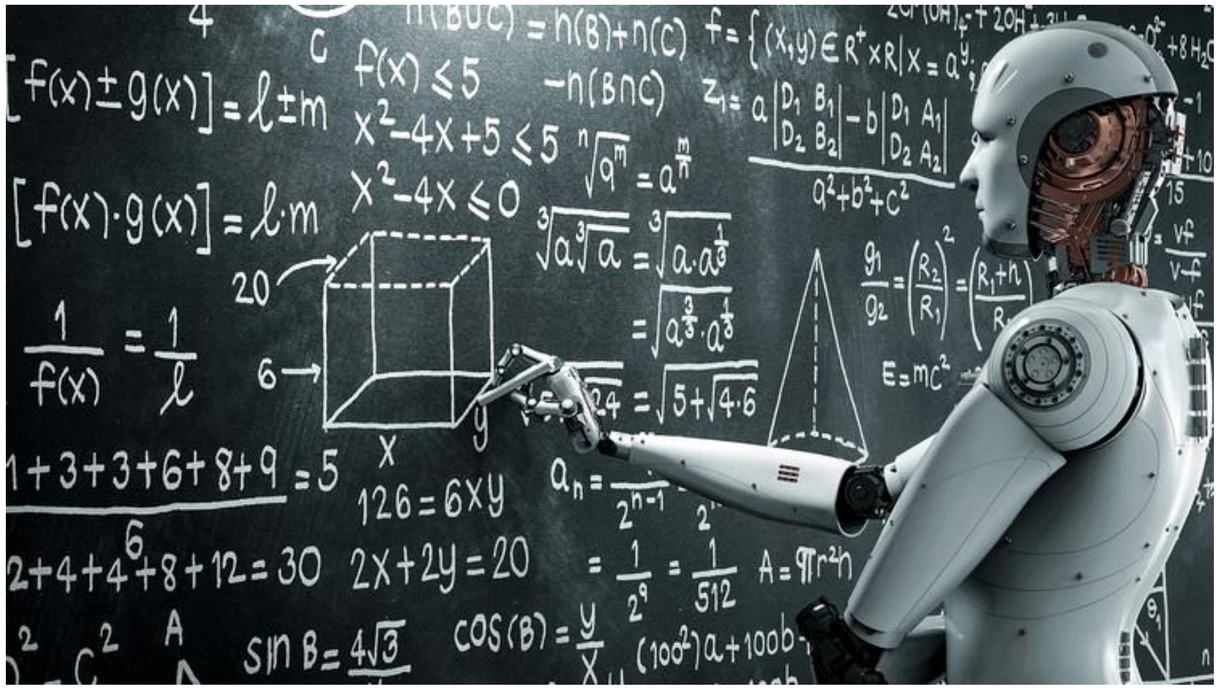


인공지능: 기계 혹은 시스템에 의해 만들어진 지능
머신러닝: 기계가 직접 데이터를 학습함으로써, 패턴을 찾는 방법
딥러닝: 데이터 특징을 사람이 추출하지 않는 방법

Machine Learning과 수학

왜 수학과 관련된 라이브러리를 배워야 하는가

Machine Learning에서의 수학



확률과 통계, 선형대수, 다변수 미적분학 등등 다양한 수학이 사용됨

프로그래밍의 난이도: C/C++ (↑), Python (↓)

Python을 사용하여 머신러닝을 쉽게 코딩

C/C++ vs Python

```
// Python “hello world”
print “Hello, World!”;
```

```
// C-Language “hello world”
#include <stdio.h>
int main() {
  printf (“Hello, World!\n”);
  return 0;
}
```

간결하고 사용하기 쉬운 Python
Python을 사용하여 행렬 및 벡터 연산을 쉽게 가능

C/C++ vs Python

```
// C-Language  “matrix multiplication”
for (i=0; i<2; i++) {
    for (j=0; j<3; j++) {
        sum=0;
        for (k=0; k<4; k++) {
……
```

```
// Python “matrix multiplication”
// Using numpy
>>> a*b
```

C언어에서 행렬 곱셈을 구현하는 것은 복잡함
하지만 Python에서는 한 줄로 구현 가능
미분, 적분과 같은 복잡한 기능도 쉽게 사용 가능

기본기: NumPy

Python이 느려서 답답하다고 ? C언어는 어렵다고 ?

그러지 마. NumPy가 있으니까!

Python 배열 #1

```
>>> a = [1, 3, 5, 3, 15]  // 1차원 벡터
>>> 2*a
 [1, 3, 5, 3, 15, 1, 3, 5, 3, 15]
>>> a+a
 [1, 3, 5, 3, 15, 1, 3, 5, 3, 15]
>>>
```

리스트를 이용한 연산은 **리스트 덧셈 ,리스트 숫자의 곱셈**이 정의됨
연산 결과는 공학 분야에서 사용되는 **벡터의 연산**과는 다른 결과

Python 배열 #2

```
>>> import  array as ar
>>> a = ar.array('i', [1, 2, 3])  // 'i'는 정수형 데이터를 의미
>>> b = ar.array('i', [4, 5, 6])
>>> print(a+b)
>>> array('i', [1, 2, 3, 4, 5, 6])
 >>>
```

Python의 리스트는 숫자 뿐만 아니라 **다양한 데이터 형**의 사용 가능
Array객체에 **사칙 연산**을 적용, **리스트 형과 같이 처리**되기 때문에
벡터와 **행렬**을 처리하기에 적합하지 않음

NumPy 배열

```
>>> import  numpy // NumPy의 변수나 함수를 사용하기 위한 정의
>>> import numpy as newname  // NumPy를 불러 newname으로 사용
// NumPy에 있는 모든 객체를 모두 불러, 모듈 이름없이 함수와 변수 사용
>>> from  numpy import *
>>>
```

NumPy는 과학 연산에 사용되는 **다차원 배열 처리**를 위한 패키지
숫자 연산을 할 때, 리스트나 Array를 사용하는 것보다
NumPy를 사용하면 더 **효율적**이고 편리하게 연산 가능

NumPy 배열 예제 #1

```
>>> import  numpy  as np
>>> a = np.array([0, 1, 2, 3]) // NumPy 배열 초기화
>>> a
array([0, ,1, 2, 3])
>>> b = np.arange(10).reshape(2, 5) // 범위(0~9), 배열 모양 (2,5)
```

array()는 리스트를 **NumPy 배열**로 변환하는 함수
arrange()는 주어진 구간에서 **균일한 간격의 숫자**를 만드는 함수
NumPy의 주요 객체는 **동일한 데이터 형**을 갖는 원소로만 구성됨

NumPy 배열 예제 #1

```
>>> b
array([[0, ,1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> b.shape // b 배열의 모양
(2,5) // rank 2
>>> b.size // b 배열의 크기
10
```

각 원소의 인덱스는 **정수의 튜플**로 표시됨
NumPy에서 **차원은 축 (axis)**라 부르고, **축의 개수**를 **rank**라고 함

NumPy 배열 예제 #2

다음과 같은 **2x3 행렬 A, B**를 계산하는 경우는 ?

$$\boldsymbol{A} = \begin{vmatrix} 1 & 2 & 3 \\ 3 & 2 & 5 \end{vmatrix} \quad \boldsymbol{B} = \begin{vmatrix} -1 & 3 & 5 \\ 1 & 4 & 2 \end{vmatrix}$$

$$\boldsymbol{A} + \boldsymbol{B} = \begin{vmatrix} 1-1 & 2+3 & 3+5 \\ 3+1 & 2+4 & 5+2 \end{vmatrix} = \begin{vmatrix} 0 & 5 & 8 \\ 4 & 6 & 7 \end{vmatrix}$$

$$\boldsymbol{A} - \boldsymbol{B} = \begin{vmatrix} 1-(-1) & 2-3 & 3-5 \\ 3-1 & 2+4 & 5-2 \end{vmatrix} = \begin{vmatrix} 2 & -1 & -2 \\ 2 & -2 & 3 \end{vmatrix}$$

NumPy 배열 예제 #2

```
>>> a = np.array([1, 2, 3, 3, 2, 5]).reshape(2, 3)
>>> b = np.array([-1, 3, 5], [1, 4, 2])
>>> a+b
array([[0, ,5,  8],
          [4, 6, 7]])
>>> a-b
array([[2, -1,  -2],
          [2, -2, 3]])
```

Python에서는 **크기가 같은 배열**에 대해서 **+,-,*,/ 등의 산술연산** 가능

NumPy 배열 예제 #2

```
>>> a = np.array([1, 2, 3, 3, 2, 5]).reshape(2, 3)
>>> b = np.array([-1, 3, 5], [1, 4, 2])
>>> a*b
array([[-1, 6,  15],
       [3, 8, 10]])
>>> a/b
array([[-1, 0,  0],
       [3, 0, 2]])
```

각 원소의 형태가 정수형 이므로, **나눗셈 연산**에서 **자리올림** 발생

NumPy 배열 예제 #3

전치행렬을 계산하는 경우는 ?

$$A^T B = \begin{vmatrix} 1 & 3 \\ 2 & 2 \\ 3 & 5 \end{vmatrix} * \begin{vmatrix} -1 & 3 & 5 \\ 1 & 4 & 2 \end{vmatrix}$$

$$= \begin{vmatrix} 1*(-1)+2*3+3*5 & 1*1+2*4+3*2 \\ 3*(-1)+2*3+3*5 & 3*1+2*4+5*2 \end{vmatrix}$$

$$= \begin{vmatrix} 20 & 15 \\ 28 & 21 \end{vmatrix}$$

NumPy 배열 예제 #3

```
>>> np.dot(a, b.transpose())
>>> array([[20, 15],
           [28, 21]])
>>> np.dot(a, b.T)
>>> array([[20, 15],
           [28, 21]])
```

transpose() 함수는 행렬의 **전치 행렬**을 만드는 함수
dot() 함수는 **행렬의 내적**을 계산하는 함수
b.transpose()대신에 **b.T**라고 사용 가능

기본기: Matplotlib

데이터는 시각화 시켜야 제맛!

데이터의 시각화

데이터: 현실세계의 일들을 관찰, 측정해서 얻은 값
정보: 데이터를 처리해서 얻는 **의미 있는 값**
데이터의 시각화: 정보를 찾기 위한 **데이터의 분석 방법**

Matplotlib: 그래프 그리기

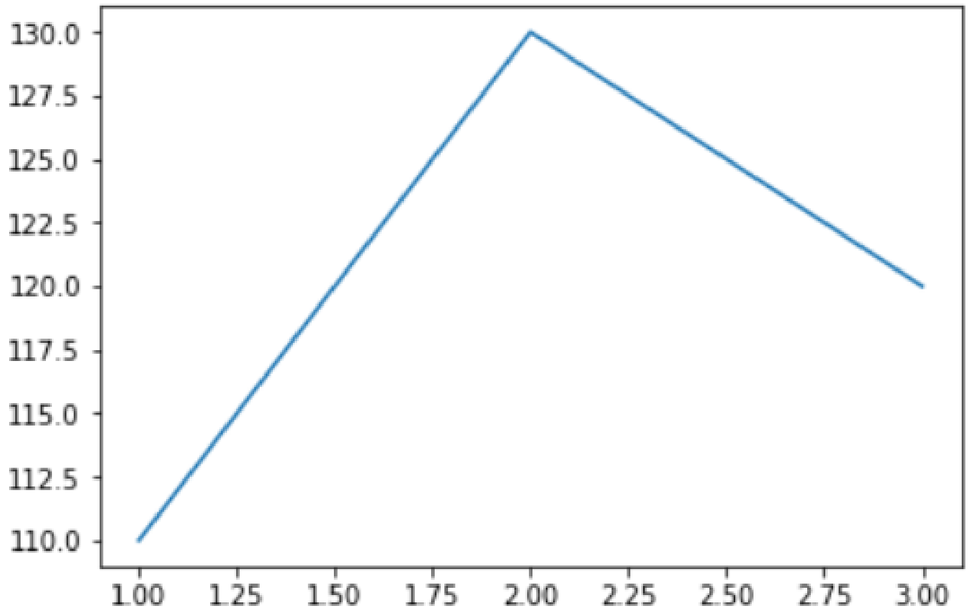
```
>>> from matplotlib import pyplot as plt
// x값(1, 2, 3)과 y축 값 (110, 130, 120)을 사용하는 그래프
>>> plt.plot([1, 2, 3], [110, 130, 120])
>>> plt.show() // 그래프를 그리는 함수
```

Matplotlib는 데이터를 **시각화** 할 때 필요한 패키지
plot() 함수를 사용하여 데이터를 시각화

Matplotlib: 그래프 그리기

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

plt.plot([1,2,3], [110,130,120])
plt.show()
```



Matplotlib: 제목과 축 레이블

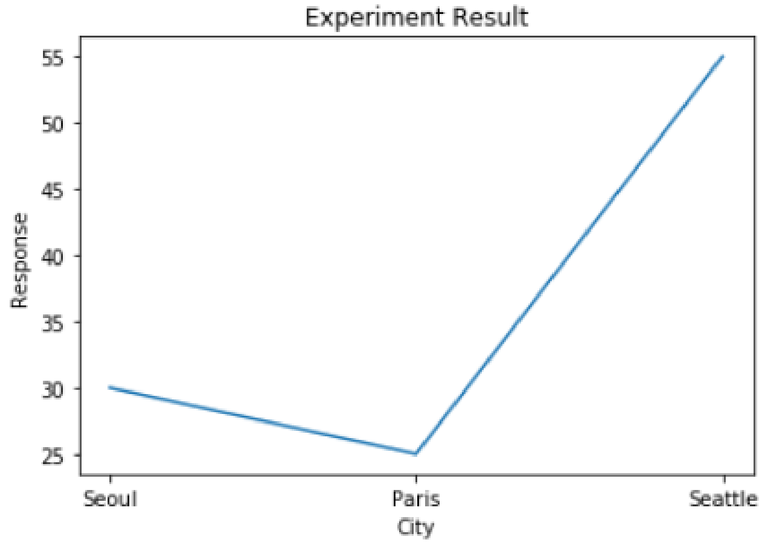
```
>>> from matplotlib import pyplot as plt
>>> plt.plot([“Seoul”, “Paris”, “Seattle”], [30, 25, 55])
>>> plt.xlabel(‘City’)
>>> plt.ylabel(‘Response’)
>>> plt.title(‘Experiment Result’)
>>> plt.show()
```

plot에 **X, Y축 레이블이나 제목**을 붙이기 위해서는
plot.xlabel(축이름), **Plot.xlabel(축이름)**, **plot.title(제목)** 함수 사용

Matplotlib: 제목과 축 레이블

In [3]:
```
from matplotlib import pyplot as plt

plt.plot(["Seoul","Paris","Seattle"], [30,25,55])
plt.xlabel('City')
plt.ylabel('Response')
plt.title('Experiment Result')
plt.show()
```



Matplotlib: 범례 추가

```
>>> from matplotlib import pyplot as plt
>>> plt.plot([1, 2, 3], [1, 4, 9])
>>> plt.plot([2, 3, 4], [5, 6, 7])
>>> plt.xlabel('Sequence')
>>> plt.ylabel('Time(secs)')
>>> plt.title('Experiment Result')
>>> plt.legend(['Mouse', 'Cat'])
>>> plt.show()
```

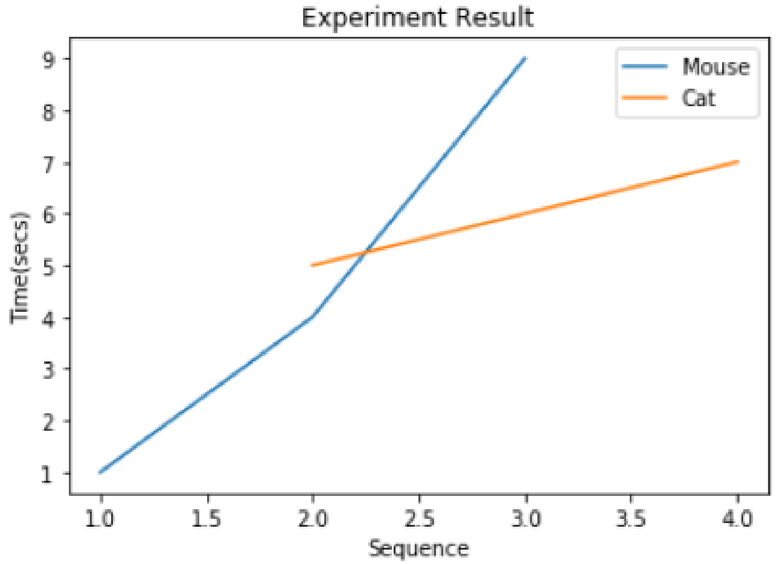
여러 개의 라인들을 추가하기 위해서는, **plt.plot()**을 여러 번 사용
각 라인에 대한 **범례 추가**는 **plt.legend([라인1, 라인2])** 함수 사용

Matplotlib: 범례 추가

In [6]:

```
from matplotlib import pyplot as plt

plt.plot([1,2,3], [1,4,9])
plt.plot([2,3,4],[5,6,7])
plt.xlabel('Sequence')
plt.ylabel('Time(secs)')
plt.title('Experiment Result')
plt.legend(['Mouse', 'Cat'])
plt.show()
```



고급기: Pandas

Data Analysis를 쉽고 강력하게 하는 방법!

Pandas란 ?

	A	B	C	D	E	F
1		작업 목록				
2		내 작업	시작 날짜	기한	완료율	완료 메모
3		[작업]	[날짜]	[날짜]	0%	
4		[작업]	[날짜]	[날짜]	50%	
5		[작업]	[날짜]	[날짜]	100%	
6						
7						

Pandas는 **행과 열로 이뤄진 데이터 객체**를 생성/수정을 위한 패키지

Pandas 자료구조: Series

```
>>> import numpy as np // numpy도 함께 import
>>> import pandas as pd
// Series 정의하기
>>> obj = pd.Series([4, 7, -5 ])
>>> obj
    0    4
    1    7
    2   -5
    dtype: int64
```

Pandas는 두 종류의 자료구조 (Series, DataFrame)의 사용 가능

Series는 **1차원 배열**과 같은 자료구조

1차원 배열의 값 뿐만 아니라 **각 값에 연결된 인덱스 값**도 저장

Pandas 자료구조: Series

```
>>> import numpy as np // numpy도 함께 import
>>> import pandas as pd
// Series 정의하기
>>> obj = pd.Series([4, 7, -5 ], index = [‘2016-01-01’, ‘2016-01-02’, ‘2016-01-03’])
>>> obj
    2016-01-01      4
    2016-01-02      7
    2016-01-03     -5
    dtype: int64
```

파이썬 리스트와는 달리 **인덱싱 값을 사용자가 설정 가능**
Index에 **인덱싱할 값**을 넘겨주면 해당 값으로 인덱싱

Pandas 자료구조: Series 예제

```
>>> import numpy as np // numpy도 함께 import
>>> import pandas as pd
>>> stock0 = pd.Seris([10, 20, 30], index=['naver', 'skt', 'LG'])
>>> stock1 = pd.Seris([30, 20, 10], index=['LG', 'naver', 'skt'])
>>> merge = stock0 + stock1
    LG    40
   naver 40
   skt 40
   dtype: int64
```

Pandas의 Series는 **인덱싱이 서로 다른 경우**에도
인덱싱이 같은 값까지 덧셈을 수행

Pandas 자료구조: DataFrame

일자별 주가 | 일봉차트 | 자세히

일자	시가	고가	저가	종가	전일비	등락률	거래량
16.02.29	11,650	12,100	11,600	11,900	▲300	+2.59%	225,844
16.02.26	11,100	11,800	11,050	11,600	▲600	+5.45%	385,241
16.02.25	11,200	11,200	10,900	11,000	▼100	-0.90%	161,214
16.02.24	11,100	11,100	10,950	11,100	▲50	+0.45%	77,201
16.02.23	11,000	11,150	10,900	11,050	▲100	+0.91%	113,131
16.02.22	10,950	11,050	10,850	10,950	▼100	-0.90%	138,387
16.02.19	10,950	11,100	10,800	11,050	-0	0.00%	76,105
16.02.18	11,050	11,200	10,950	11,050	▲250	+2.31%	83,611
16.02.17	11,150	11,300	10,800	10,800	▼350	-3.14%	189,480
16.02.16	10,950	11,200	10,850	11,150	▲300	+2.76%	133,359

DataFrame은 **2차원 배열**과 같은 자료구조
여러 개의 칼럼 (Column)으로 구성된 **2차원 형태의 자료구조**
예) **주가 데이터**: 로우 (일자), 칼럼 (시가, 고가, 저가 등등)

Pandas 자료구조: DataFrame

```
>>> from pandas import Series, DataFrame
>>> raw_data = {‘col0’: [1, 2, 3, 4],
                ‘col1’: [10, 20, 30, 40],
                ‘col2’: [100, 200, 300, 400]}
>>> data = DataFrame(raw_data)
```

DataFrame 객체를 생성하기 위해서는 **딕셔너리**를 사용
각 칼럼에 대한 저장된 데이터를 사용하여 DataFrame 객체로 생성

Pandas 자료구조: DataFrame

```
>>> data
   col0  col1  col2
0     1    10   100
1     2    20   200
2     3    30   300
3     4    40   400
```

‘col0’, ‘col1’, ‘col2’은 DataFrame의 **각 컬럼을 인덱싱** 하는데 사용
로우 방향으로는 Series와 유사하게 정수 값으로 자동으로 인덱싱 됨

Pandas 자료구조: DataFrame 예제

```
>>> from pandas import Series, DataFrame
>>> daeshin = {'open':  [11650, 11100, 11200, 11100, 11000],
               'high':  [12100, 11800, 11200, 11100, 11150],
               'low' :  [11600, 11050, 10900, 10950, 10900],
               'close': [11900, 11600, 11000, 11100, 11050]}
>>> daeshin_day = DataFrame(daeshin)
>>> daeshin_day
    close    high    low    open
0  11900   12100 11600 11650
……
```

출력 값이 시가, 고가, 저가, 종가 순으로 저장되어 있음
각 칼럼의 값이 정수 값으로 되어 있음

Pandas 자료구조: DataFrame 예제

```
>>> date = ['16.02.29', '16.02.26', '16.02.25', '16.02.24', '16.02.23']
>>> daeshin_day = DataFrame(daeshin, columns=['open', 'high', 'low', 'close'],
                                                    index=date)
>>> daeshin_day
              open   high    low    close
16.02.29  11650  12100  11600  11900
16.02.26  11100  11800  11050  11600
16.02.25  11200  11200  10900  11000
……
```

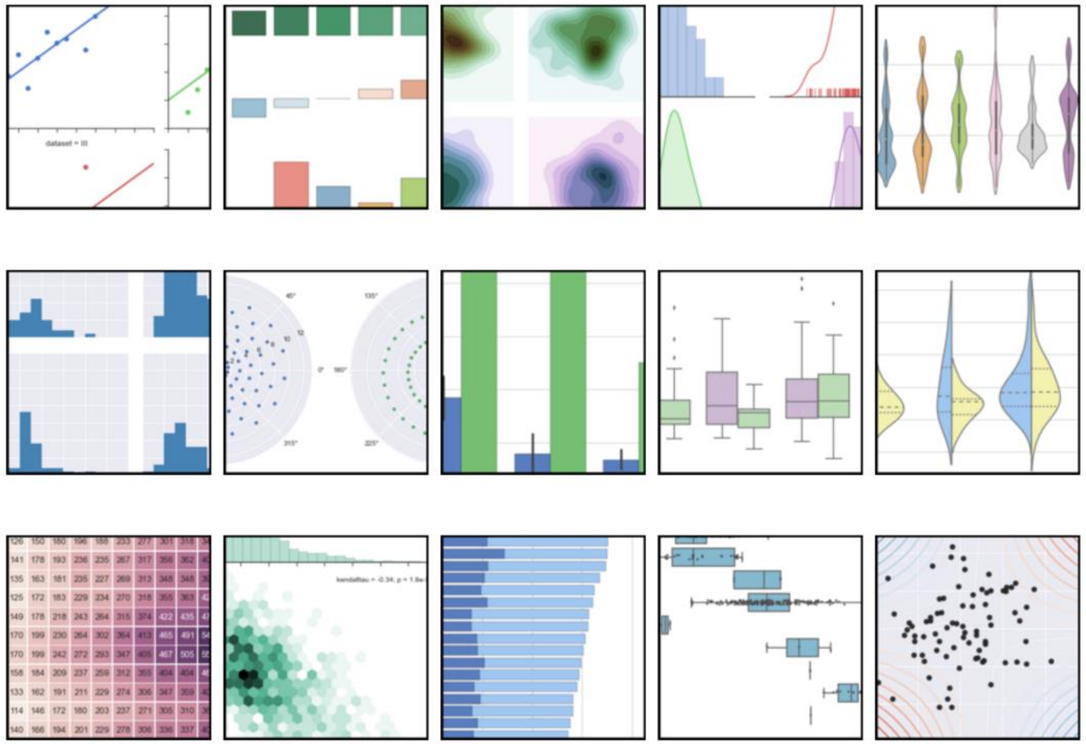
DataFrame 객체를 생성하는 시점에 **인덱스 지정** 가능

인덱싱에 사용할 값을 만든 후, DataFrame 객체 생성 시점에 지정

고급기: Seaborn

화려한 그래프를 그리고 싶다면 ?

Seaborn이란 ?



Matplotlib를 기반으로 다양한 색상 테마, 통계용 차트 기능이 추가된 패키지

Seaborn: 데이터 불러오기

```
>>> import seaborn as sns
>>> from matplotlib import pyplot as plt
>>> df=sns.load_dataset('iris') // iris (붓꽃) 데이터 로드
// df = pandas.read_csv('iris.csv')
>>> df.head() // 전체 데이터에서 처음 5개의 row 데이터 표시 (내용 확인)
```

그래프를 그리기 위해서는 **데이터의 로드 과정**이 필요
Seaborn의 **load_dataset()** 또는 **Pandas**의 **read_csv()** 함수 사용

Seaborn: 데이터 불러오기

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

iris 데이터 셋은 150개의 송이 붓꽃 데이터
sepal (꽃잎), petal (꽃받침), species (품종)

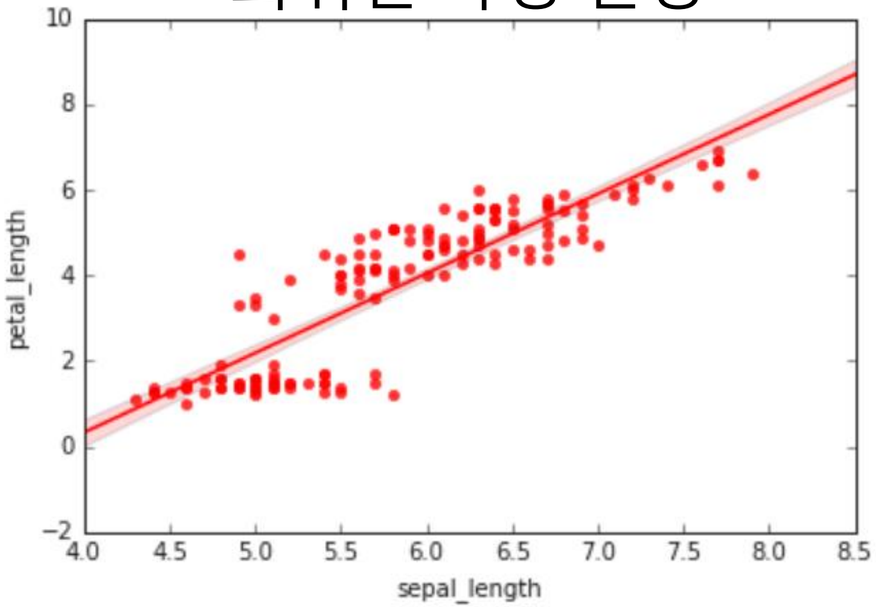
Seaborn: 산점도 그리기

```
>>> sns.regplot(x=df["sepal_length"], y=df["petal_length"]) // 검은색
>>> sns.regplot(x=df["sepal_length"], y=df["petal_length"], color='red’) // 색상
// 선의 속성만 변경
>>> sns.regplot(x=df["sepal_length"], y=df["petal_length"], line_kws={'color': 'red’})
// 점의 속성만 변경
>>> sns.regplot(x=df["sepal_length"], y=df["petal_length"], scatter_kws={'color': 'red'})
```

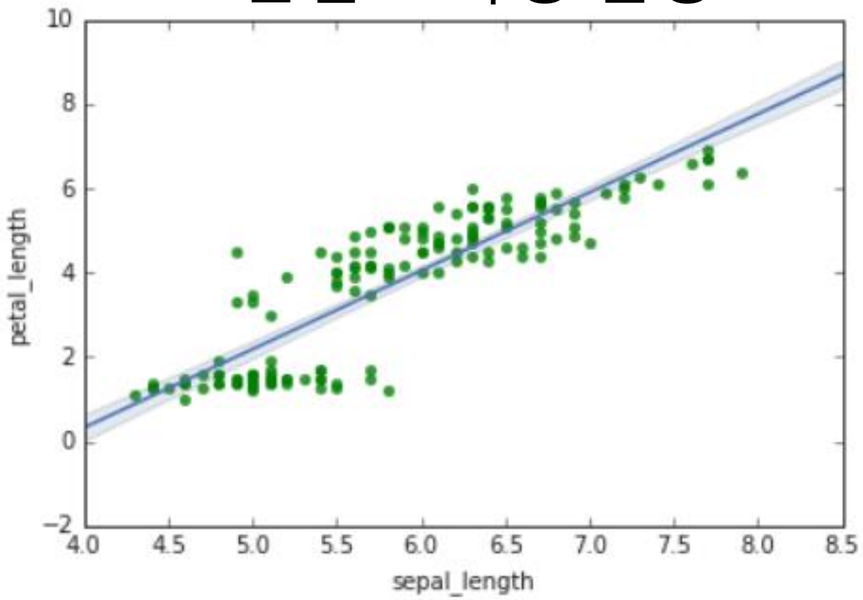
regplot: **산점도와 회귀선**을 같이 그리는 그래프
df[‘로우 이름’]: 특정 로우 이름에 해당되는 데이터 사용
link_kws/scaffer_kws: **회귀선** 또는 **산점도**의 속성을 변경 (딕셔너리 형식)

Seaborn: 산점도 그리기

회귀선 속성 변경



산점도 속성 변경



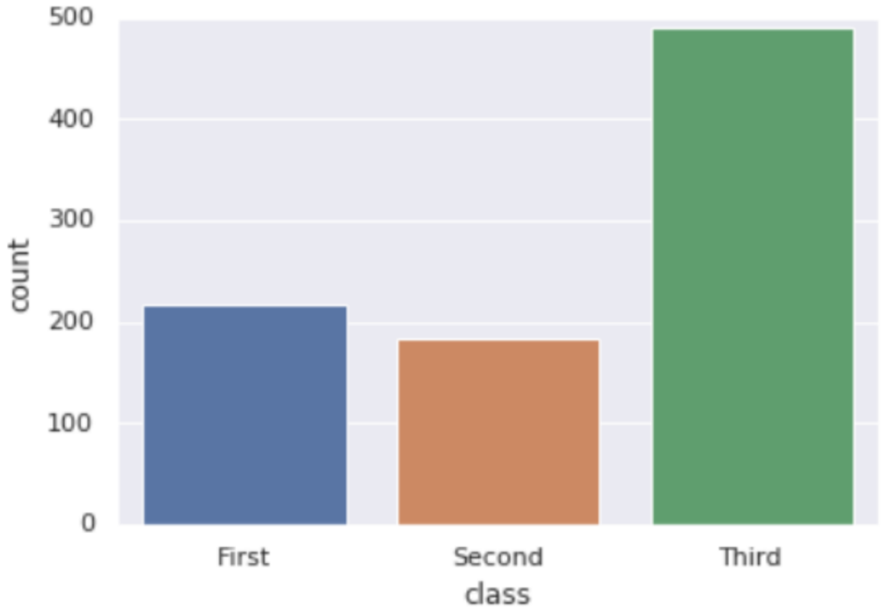
Seaborn: 카운트 플롯

```
>>> import seaborn as sns
>>> sns.set(style="darkgrid") // seaborn 스타일 설정 (배경)
>>> titanic = sns.load_dataset("titanic") // 데이터 로드
>>> ax = sns.countplot(x="class", data=titanic)
>>> ax = sns.countplot(y="class", data=titanic)
```

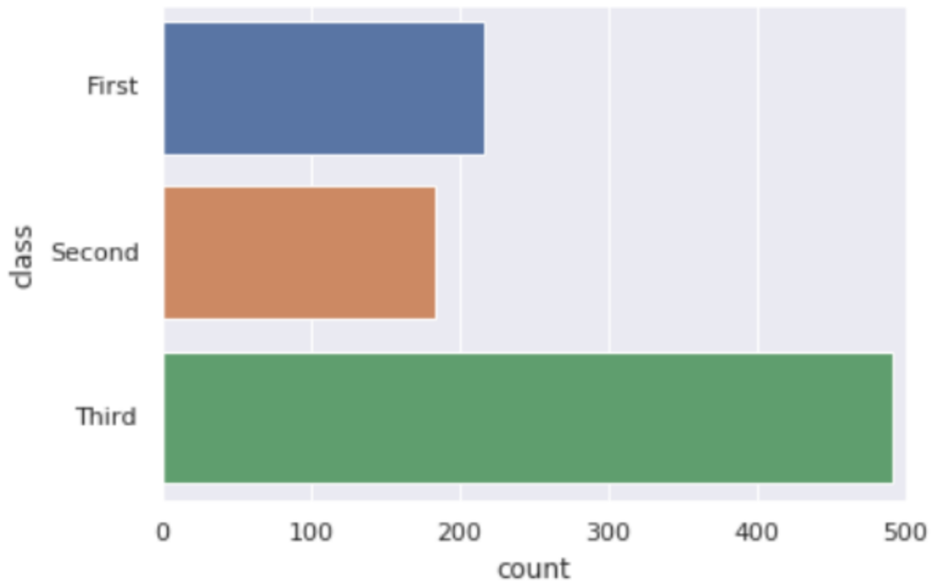
카운트플롯: 각 카테고리 값 별로 얼마나 있는지 표시하는 그래프
x인수 (열 이름 문자열), **data 인수** (해당 데이터 프레임)

Seaborn: 카운트 플롯

x-axis



y-axis



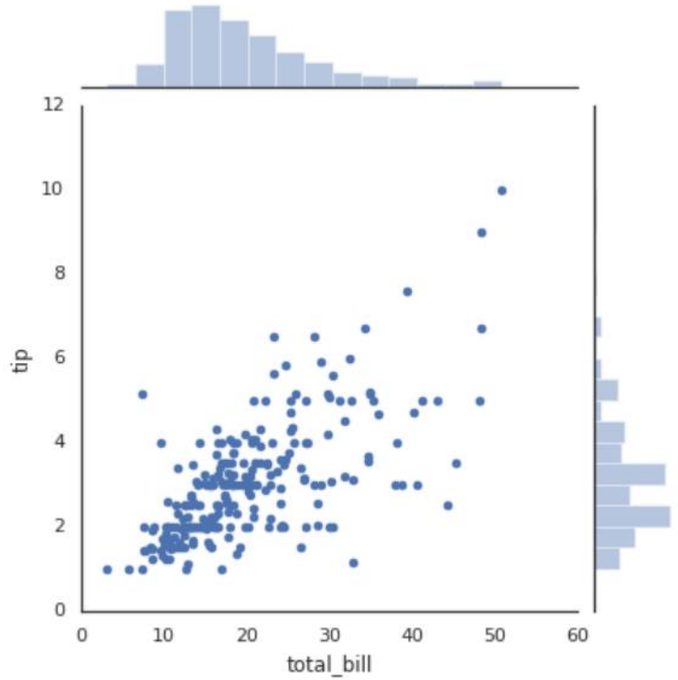
Seaborn: 2차원 실수 데이터

```
>>> import numpy as np, pandas as pd;
>>> import seaborn as sns; sns.set(style="white") // 배경 설정
>>> tips = sns.load_dataset("tips")
>>> g = sns.jointplot(x="total_bill", y="tip", data=tips)
>>> g = sns.jointplot("total_bill", "tip", data=tips, kind="reg") // reg (regression + scatter)
```

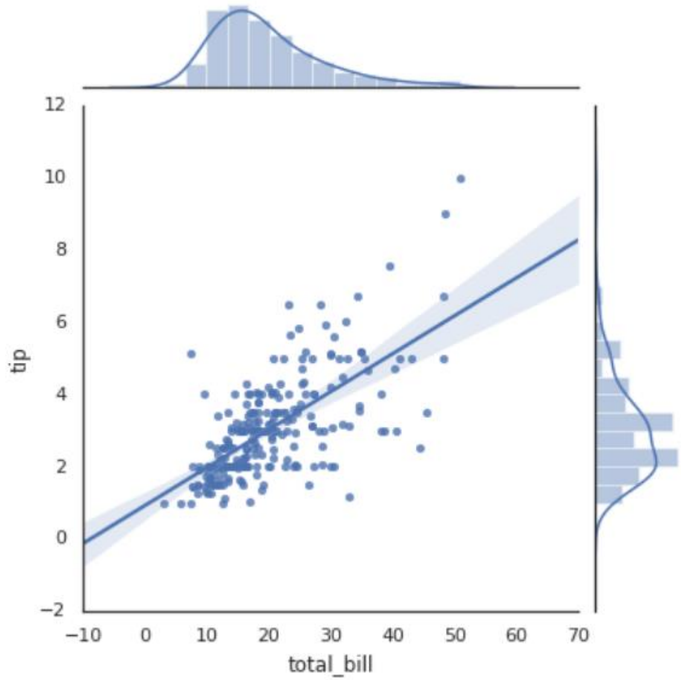
데이터가 **2차원**이고, **연속적인 실수 값**이라면 **스캐터 (scatter) 플롯**
jointplot 명령은 **스캐터 플롯** 뿐만 아니라, **히스토그램**도 가능
x/y인수 (열 또는 행 이름 문자열), **data 인수** (해당 데이터 프레임),
kind (차트 종류, 기본은 scatter)

Seaborn: 2차원 실수 데이터

기본 설정



regression + scatter



/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice