실습을 통해 기초부터 배우는 머신러닝

18.12.01 – 18.12.29 (토 , 13:00 ~ 18:00)

Various data source

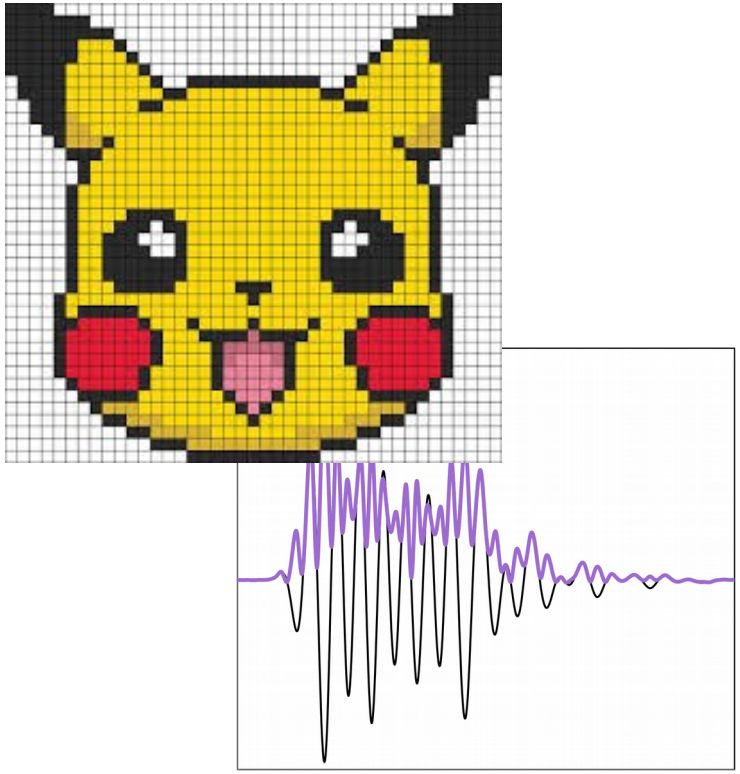
Questionnaire

Mic & Speaker

Twitter

Camera sensor

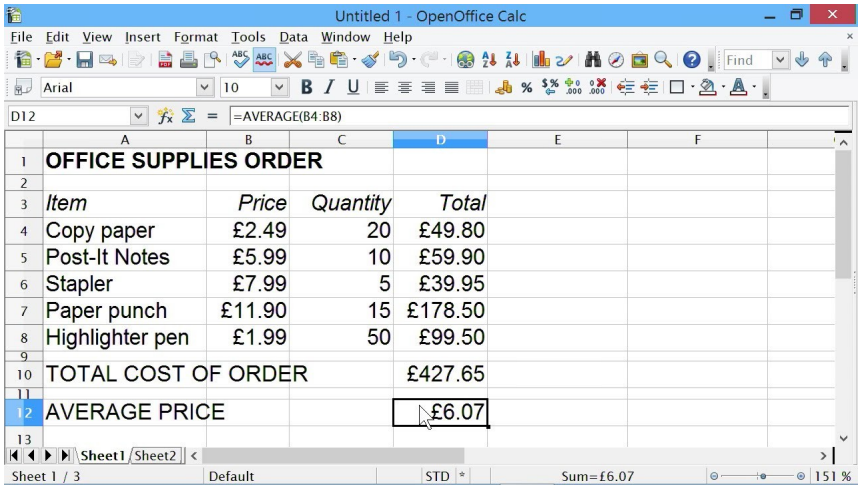
Data types



<Unstructured data >

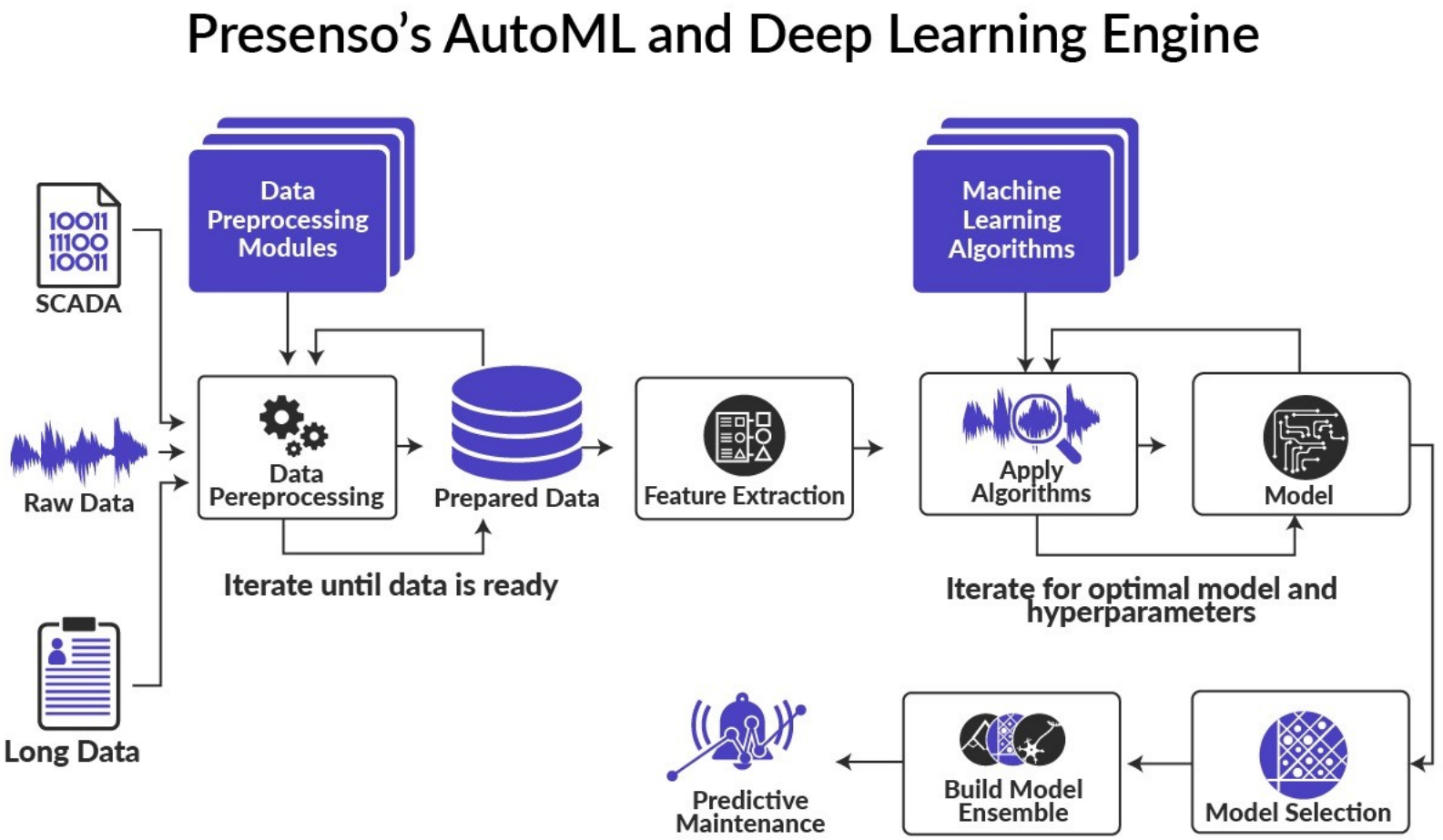
```
<h4>Indentation is useful</h4>
<table>
  <thead>
    <tr>
      <td>Name</td>
      <td>Job</td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>John</td>
```

<Semi-Structured data >



<Structured data > - 고정된 필드 / 연산 가능

Data preprocessing In ML

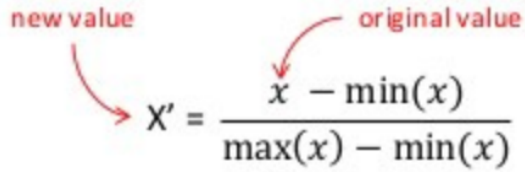


Data preprocessing examples

Nationality	Age	Salary	Gender
Spain	28	40,000	Female
Poland	38	50,000	Female
Germany		70000	Male
Poland	32	100000	Male
Spain	19	13000	Female
Germany	26	38000	Male
Germany	33	64000	Female
Spain	35		Male
Poland	24	46000	Female
Germany	20	60000	Male
Spain	31	44000	Female
Poland	27	54000	Male

Missing value

Normalization



Feature scaling

ID	Gender
1	Male
2	Female
3	Not Specified
4	Not Specified
5	Female

ID	Male	Female	Not Specified
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	0	1	0

Encoding

Week 2

Python libraries

- **numpy :** 수치형 자료 프로세싱
- **pandas :** 데이터 구조화
- **matplotlib :** 데이터 시각화 **(plotting)**

Numpy

Numpy(numerical python)

- C 언어로 구현된 파이썬 라이브러리로써 , 고성능의 수치 계산을 위해 만들어짐
- 벡터 및 행렬 연산에 있어서 매우 편리한 기능 제공
- 다차원 배열 객체 지원
- 데이터에 대한 빠른 연산 가능
- **pandas** 와 **matplotlib** 의 기반으로 사용

Numpy(numerical python)

- **ndarray : 다차원 배열 객체**

  : 모든 원소는 같은 자료형이어야 함

```
Import numpy as np
data = np.random.randn(2, 3)
print (data)
>> [[-0.0336809  -0.16725775  0.52641168]
      [-1.0133925   1.88160002 -2.59587815]]
```

Numpy(numerical python)

- **ndarray 생성**

  : 배열을 생성하는 가장 쉬운 방법은 array 함수를 이용하여 리스트를 배열로 바꾸는 것

```
list1 = [0, 1, 2, 3, 4]
arr1 = np.array(list1)
print (arr1)
print(type(arr1))
>> [1 2 3 4]
      <class 'numpy.ndarray'>
```

Numpy(numerical python)

```
list2 =[[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(list2)
print (arr2)
>> [[1 2 3 4]
       [5 6 7 8]]
```

Numpy(numerical python)

- **다양한 함수를 사용하여 초기화 된 배열도 생성 가능**

```
print (np.zeros(3))
>> [ 0.  0.  0.]
print (np.ones((2,5)))
>> [[ 1.  1.  1.  1.  1.]
      [ 1.  1.  1.  1.  1.]]
print (np.arange(10))
>> [0 1 2 3 4 5 6 7 8 9]
print (np.eye(3))
>> [[ 1.  0.  0.]
     [ 0.  1.  0.]
     [ 0.  0.  1.]]
```

Numpy(numerical python)

```
data = np.random.randn(2, 3)
print (data)
>> [[-0.0336809  -0.16725775  0.52641168]
      [-1.0133925   1.88160002 -2.59587815]]
print(data.dtype) : 데이터 타입
>> float64
print(data.shape) : 각 차원의 크기
>> (2, 3)
print (data.ndim) : 차원 수 확인 (=rank)
>> 2
```

Numpy(numerical python)

- **Data type 이나 shape 도 변경 가능**

```
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
print (arr)
>> [  3.7  -1.2  -2.6   0.5  12.9  10.1]
int_arr = arr.astype(np.int32)
print (int_arr)
>> [ 3 -1 -2  0 12 10]
new_arr = arr.reshape(2,3)
print (new_arr)
>> [[  3.7  -1.2  -2.6]
      [  0.5  12.9  10.1]]
```

Numpy(numerical python)

- **Array 연산 (덧셈)**

```
list1 = [1, 2, 3, 4]
arr = np.array(list1)
print (arr + 1)
>> [2 3 4 5]
```

** print (list1 + 1) → ?

Numpy(numerical python)

- **Array 연산 (덧셈)**

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
arr1 = np.array(list1)
arr2 = np.array(list2)
print (arr1 + arr2)
>> [5 7 9]
```

** print (list1 + list2) → ?

Numpy(numerical python)

- **Array 연산 (곱셈)**

```
list1 = [1, 2, 3, 4]
arr = np.array(list1)
print (arr * 3)
>> [3 6 9 12]
```

** print (list1 * 3) → ?

Numpy(numerical python)

- **Array 연산 (곱셈)**

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
arr1 = np.array(list1)
arr2 = np.array(list2)
print (arr1 * arr2)
>> [4 10 18]
```

** print (list1 * list2) → ?

Numpy(numerical python)

- **Array 연산 (거듭 제곱)**

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
arr1 = np.array(list1)
arr2 = np.array(list2)
print (arr1 ** arr2)
>> [1 32 729]
```

** with list → ?

Numpy(numerical python)

- **Array 연산 (뺄셈)**

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
arr1 = np.array(list1)
arr2 = np.array(list2)
print (arr1 - arr2)
>> [-3 -3 -3]
```

Numpy(numerical python)

- **Array 연산 (나눗셈)**

```
list1 = [1, 2, 3, 4]
arr = np.array(list1)
print (arr / 5)
>> [ 0.2  0.4  0.6  0.8]
```

Numpy(numerical python)

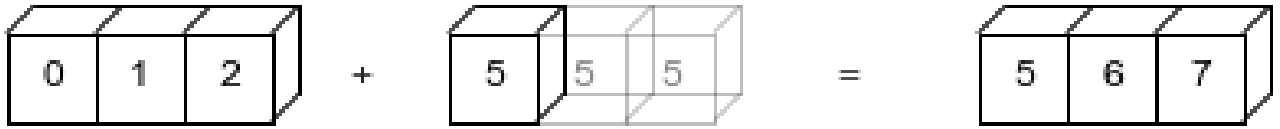
- **브로드캐스팅 (Broadcasting)**

  : 서로 다른 크기의 array 연산 가능

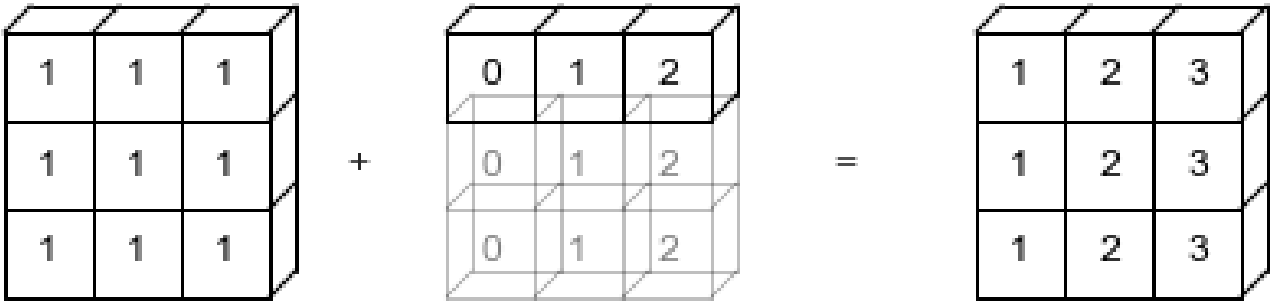
```
arr1 = np.array([[1, 2, 3],[4, 5, 6]])
arr2 = np.array([10, 11, 12])
print (arr1)
>> [[1 2 3]
      [4 5 6]]
print (arr2)
>> [10 11 12]
print (arr1 + arr2)
>> [[11 13 15]
      [14 16 18]]
```

Broadcasting

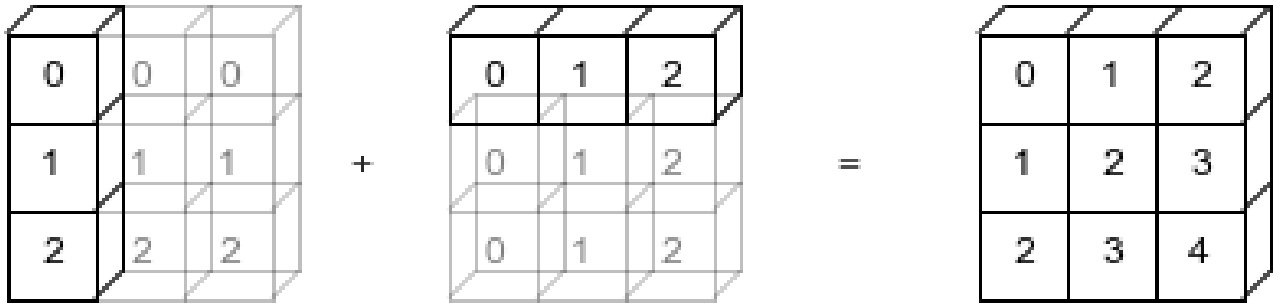
$\texttt{np.arange}(3) + 5$



$\texttt{np.ones}((3,3)) + \texttt{np.arange}(3)$



$\texttt{np.arange}(3).\texttt{reshape}((3,1)) + \texttt{np.arange}(3)$



Numpy(numerical python)

- **List VS Numpy 성능 비교**

```
a = list(range(1000)) # List
start = time.time()
[i**2 for i in a] #  원소에 하나씩 접근
print (time.time() - start)
>> 0.0011563301086425781
b = np.arange(1000) # Numpy
start = time.time()
b**2 #  벡터의 산술 연산
print (time.time() - start)
>> 0.0006370544433593375
```

Numpy(numerical python)

- **Indexing & slicing**

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print (arr)
>> [[ 1  2  3  4]
      [ 5  6  7  8]
      [ 9 10 11 12]]
print (arr[1])
>> [5 6 7 8]
print (arr[1, 1])
>> 6
print (arr[:2, 1:3]) # 2 차원  array 인덱싱 →  2 개 인자 필요
>> [[2 3]
      [6 7]]
```

Numpy(numerical python)

- **array 의 slice 는 원본 array 의 view**

```
list1 = [0,1,2,3,4,5,6,7,8,9,10]
arr1 = np.array(list1)
part_arr1 = arr1[5:8]
part_arr1[0] = 100
print ('part_arr1 :', part_arr1)
>> part_arr1 : [100   6   7]
print ('arr1 :', arr1)
>> arr1 : [  0   1   2   3   4 100   6   7   8   9  10]
```

Numpy(numerical python)

- **List 에서는 ?**

```
list1 = [0,1,2,3,4,5,6,7,8,9,10]
part_list1 = list1[5:8]
part_list1[0] = 100
print ('part_list1 :', part_list1)
>> part_list1 : [100, 6, 7]
print ('list1 :', list1)
>> list1 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Numpy(numerical python)

- **해결책은 ?**

```
original_arr = np.array(list1)
copied_arr = original_arr.copy()
part_arr2 = copied_arr[5:8]
part_arr2[0] = 100
print ('part_arr2 :', part_arr2)
>> part_arr2 : [100   6   7]
print ('copied_arr :', copied_arr)
>>copied_arr : [  0   1   2   3   4 100   6   7   8   9  10]
print ('original_arr :', original_arr)
>>original_arr : [ 0  1  2  3  4  5  6  7  8  9 10]
```

Numpy(numerical python)

- **Boolean index – 조건으로 접근**

```
names = np.array(['kim', 'lee', 'choi', 'park', 'shin'])
scores = np.array([[80, 85, 95, 100],[55, 70, 65, 90],[75, 45, 100, 50],
                   [90, 80, 70, 60],[100, 100, 100, 100]]) # 국어 / 영어 / 수학 / 과학
print (names)
>> ['kim' 'lee' 'choi' 'park' 'shin']
print (scores)
>> [[ 80  85  95 100]
    [ 55  70  65  90]
    [ 75  45 100  50]
    [ 90  80  70  60]
    [100 100 100 100]]
```

Numpy(numerical python)

```
>> [[ 80  85  95 100]
    [ 55  70  65  90]
    [ 75  45 100  50]
    [ 90  80  70  60]
    [100 100 100 100]]
print (names == ‘park’)
>> [False False False  True False]
scores[names == 'park'] #  박의 점수
>> [[90 80 70 60]]
print (scores[(names == 'park') | (names == 'lee')]) #  박 또는 이의 점수
>> [[55 70 65 90]
    [90 80 70 60]]
```

Numpy(numerical python)

```
print (names[scores[:,2] >= 90])
>> ['kim' 'choi' 'shin']
```

위의 결과가 의미하는 바는 ??

```
#  총점이  300  미만인 사람의 수학 점수는 ?
print (scores[np.sum(scores, axis=1) < 300, 2]) # column-wise
>> [ 65 100]
```

Numpy(numerical python)

- **난수 생성**

```
print(np.random.rand(2,2)) # [0,1) random number
>> [[ 0.85014778  0.97008119]
    [ 0.09285392  0.55309854]]

print(np.random.randn(2,2)) # normal distribution(mean=0, variance=1)
>> [[-0.2580551   0.75132886]
    [-0.65383838  1.31369803]]

print(np.random.randint(0, 10, size=[2,2])) # [0, 10) random integer
>> [[2 2]
    [1 4]]
```

Pandas

Pandas

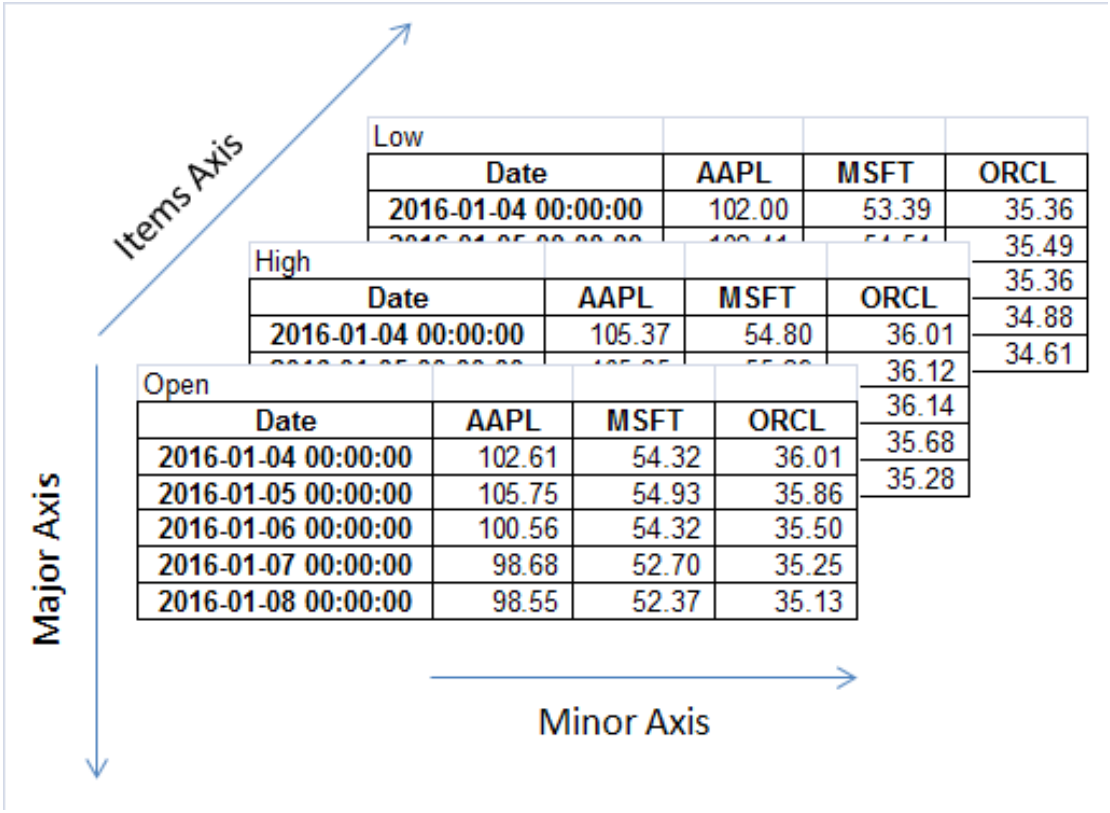
- 높은 수준의 데이터 구조와 데이터 분석 도구를 제공
- 표와 같은 스트레드시트 구조로 데이터를 다룰 수 있는 기능
- 색인을 사용하여 손쉽게 값에 접근하고 정렬하는 가능
- 누락된 데이터를 유연하게 처리할 수 있는 기능
- 데이터를 합치고 관계 연산을 수행하는 기능

Pandas – data structures

- Series : 1-D data-structure

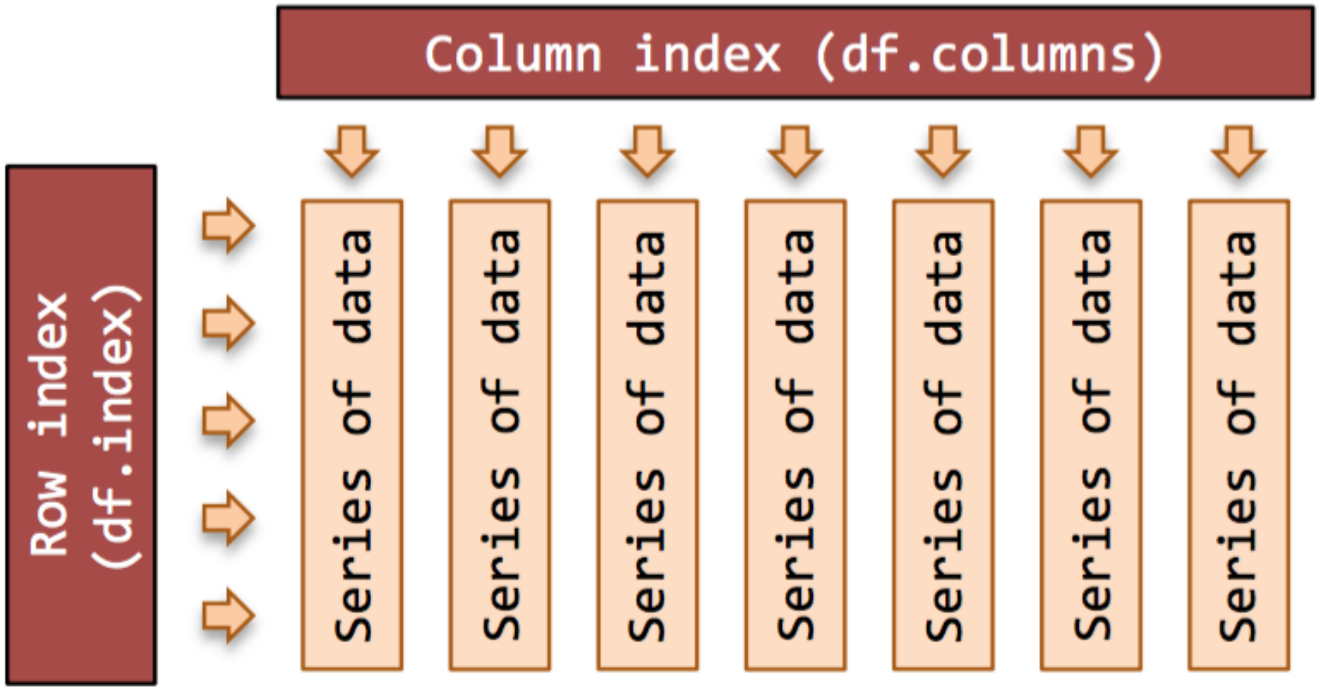
- Data Frame : 2-D data-structure

- Panel : 3-D data-structure



Pandas - Series

- 1 차원 배열과 같은 구조 : 색인 - 값 매핑



Pandas - Series

- **Series 생성**

```
Import pandas as pd
series = pd.Series([3, 5, 7, 9], index=['a', 'b', 'c', 'd'])
print (series)
>> a   3
      b   5
      c   7
      d   9
      dtype: int64
print (series.values)
>> [3 5 7 9]
print (series.index)
>> Index(['a', 'b', 'c', 'd'], dtype='object')
```

Pandas - Series

- **Series from dictionary**

```
data = {'a' : 0, 'b' : 1, 'c' : 2}
print (pd.Series(data))
>> a   0
   b   1
   c   2
   dtype: int64
s = pd.Series(data,index=['b','c','d','a'])
print (s)
>> b   1.0
   c   2.0
   d   NaN
   a   0.0
   dtype: int64
print (s[[‘a’, ‘d]])
>> a   0.0
   d   NaN
   dtype: float64
```

Pandas - Series

```
print (s)
>> a   0
   b   1
   c   2
   dtype: int64
print (s['c']) # 인덱스로 값에 접근 = print (s[2])
>> 2
print (s[s>0]) # 불리언 인덱싱
>> b   1
   c   2
   dtype: int64
print (s * 2) # 배열 연산
>> a   0
   b   2
   c   4
   dtype: int64
```

Pandas - Series

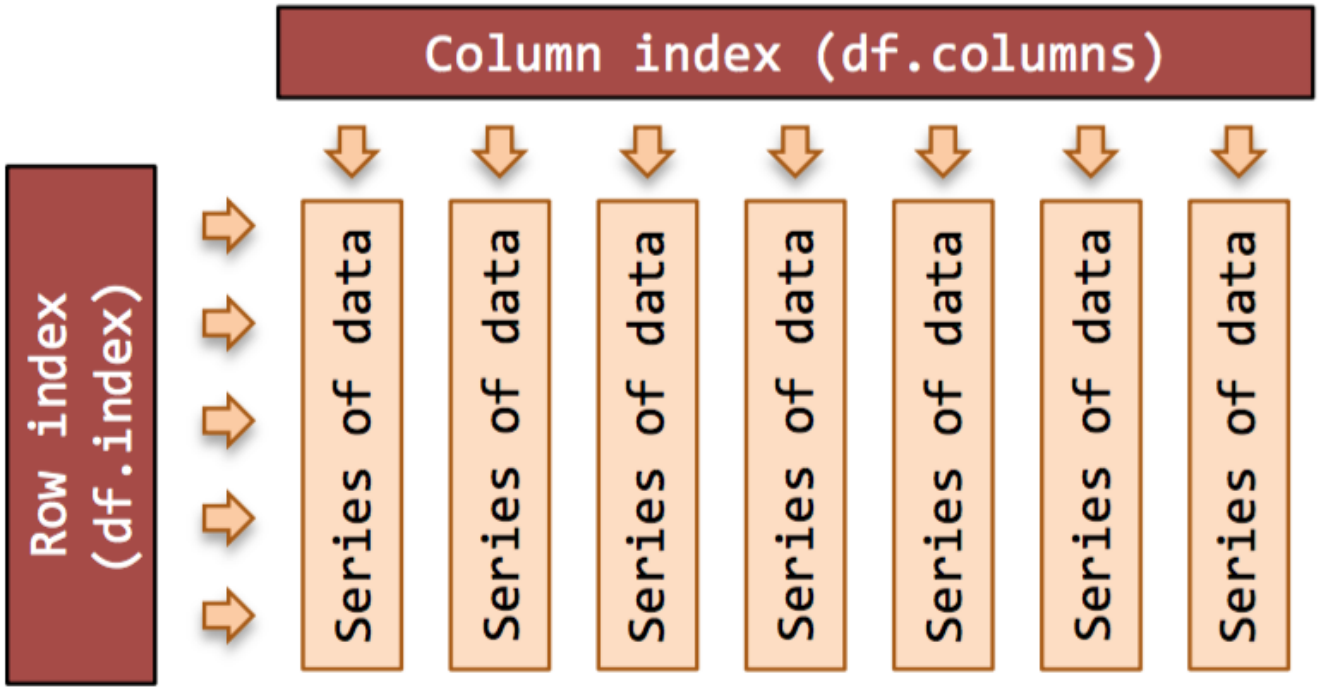
```
s1 = pd.Series([100, 200, 300], index=['Ohio', 'Texas', 'Oregon'])
s2 = pd.Series([100, 200, 400], index=['Ohio', 'Texas', 'NY'])

print (s1 + s2) #  인덱스에 고려하여 계산

>> NY        NaN
   Ohio     200.0
   Oregon     NaN
   Texas    400.0
   dtype: float64
```

Pandas – DataFrame

- **2 차원 데이터 구조 – Series 객체를 담고 있는 사전**



Pandas – DataFrame

- **DataFrame 생성**

```
data = {'state':['Ohio','Ohio','Ohio','Nevada','Nevada'],
        'year':[2000, 2001, 2002, 2001, 2002],
        'pop':[1.5, 1.7, 3.6, 2.4, 2.9]}
frame = pd.DataFrame(data=data, columns=['year','state','pop'], index=['a','b','c','d','e'])
print (frame)
>> year  state  pop
a   2000   Ohio  1.5
b   2001   Ohio  1.7
c   2002   Ohio  3.6
d   2001  Nevada  2.4
e   2002  Nevada  2.9
print (frame.index)
>> Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
print (frame.columns)
>> Index(['year', 'state', 'pop'], dtype='object')
```

Pandas – DataFrame

- **특정 column 에 접근**

```
print (frame['year']) (=frame.year)
>> a   2000
       b   2001
       c   2002
       d   2001
       e   2002
       Name: year, dtype: int64
type(frame['year'])
>> pandas.core.series.Series
```

Pandas – DataFrame

- **column 제거**

```
del frame['debt']
print (frame)
>> year  state  pop
a   2000   Ohio  1.5
b   2001   Ohio  1.7
c   2002   Ohio  3.6
d   2001 Nevada  2.4
e   2002 Nevada  2.9
```

Pandas – DataFrame

- **새로운 column 추가 – Series with index**

```
val =pd.Series([-1.2, 1.5, -1.7], index = ['b', 'c', 'e'])
frame['debt'] = val
print (frame)
>>   year  state  pop  debt
a     2000   Ohio  1.5   NaN
b     2001   Ohio  1.7  -1.2
c     2002   Ohio  3.6   1.5
d     2001 Nevada  2.4   NaN
e     2002 Nevada  2.9  -1.7
```

Pandas – DataFrame

- **column 제거**

```
del frame['debt']
print (frame)
>> year  state  pop
a   2000   Ohio  1.5
b   2001   Ohio  1.7
c   2002   Ohio  3.6
d   2001 Nevada  2.4
e   2002 Nevada  2.9
```

Pandas – DataFrame

- **column 제거**

```
print (frame.drop('debt', axis=1, inplace=True))
>> year  state  pop
a    2000   Ohio  1.5
b    2001   Ohio  1.7
c    2002   Ohio  3.6
d    2001 Nevada  2.4
e    2002 Nevada  2.9
```

Pandas – DataFrame

- **특정 row 에 접근**

```
print (frame.iloc[0]) # integer-location
>> year    2000
   state   Ohio
   pop      1.5
   Name: a, dtype: object
```

frame.iloc[0:5] # first five rows of dataframe (end : exclusive)

frame.iloc[:,0] # first column of dataframe

frame.iloc[:, 0:2] # first two columns of dataframe with all rows

Pandas – DataFrame

- **특정 row 에 접근**

```
print (frame.loc['a']) # label-based
>> year    2000
   state   Ohio
   pop      1.5
   Name: a, dtype: object
```

frame.loc['a':'c'] # rows of dataframe with index a-c (end : inclusive)

frame.loc[:,'year'] # year column of dataframe

Pandas – DataFrame

- **특정 row 에 접근**

```
print (frame.loc[frame['state'] == 'Ohio']) # boolean-indexing
>> year state  pop
a    2000  Ohio  1.5
b    2001  Ohio  1.7
c    2002  Ohio  3.6
```

Pandas – DataFrame

- **새로운 row 추가**

```
new = pd.Series([2003, 'Nevada', '2.8'], index=['year','state','pop'])
frame = frame.append(new, ignore_index=True)
print (frame)
>>   year  state  pop
0     2000   Ohio  1.5
1     2001   Ohio  1.7
2     2002   Ohio  3.6
3     2001 Nevada  2.4
4     2002 Nevada  2.9
5     2003 Nevada  2.8
```

Pandas – DataFrame

- **row 제거 by index**

```
print (frame.drop([5]))
>>   year  state  pop
0     2000   Ohio  1.5
1     2001   Ohio  1.7
2     2002   Ohio  3.6
3     2001 Nevada  2.4
4     2002 Nevada  2.9
```

Pandas – DataFrame

- **Sorting by column**

```
print (frame.sort_values(by='year'))
>> year  state  pop
0   2000   Ohio  1.5
1   2001   Ohio  1.7
3   2001 Nevada  2.4
2   2002   Ohio  3.6
4   2002 Nevada  2.9
print (frame.sort_values(by=['year','pop']))
>> year  state  pop
0   2000   Ohio  1.5
1   2001   Ohio  1.7
3   2001 Nevada  2.4
4   2002 Nevada  2.9
2   2002   Ohio  3.6
```

Pandas – DataFrame

- **Missing value 처리**

```
print (frame)
>>   year  state  pop  debt
a    2000   Ohio  1.5   NaN
b    2001   Ohio  1.7  -1.2
c    2002   Ohio  3.6   1.5
d    2001 Nevada  2.4   NaN
e    2002 Nevada  2.9  -1.7
```

Pandas – DataFrame

```
print (frame.dropna())
>>   year  state  pop  debt
b     2001   Ohio  1.7  -1.2
c     2002   Ohio  3.6   1.5
e     2002 Nevada  2.9  -1.7
print (frame.fillna(0))
>>   year  state  pop  debt
a     2000   Ohio  1.5   0.0
b     2001   Ohio  1.7  -1.2
c     2002   Ohio  3.6   1.5
d     2001 Nevada  2.4   0.0
e     2002 Nevada  2.9  -1.7
```

Pandas – DataFrame

- **Merging : 공통된 column 기준으로 병합**

```
print (left)
>> Name   id subject_id
a    Alex     1     sub1
b    Amy      2     sub2
c    Allen    3     sub4
d    Alice    4     sub6
e    Ayoung  5    sub5
print (right)
>> Name   id  subject_id
f    Billy    1     sub2
g    Brian    2     sub4
h    Bran     3     sub3
I    Bryce    4     sub6
j    Betty    5     sub5
```

Pandas – DataFrame

- **how= 'inner' : 교집합**

```
print (pd.merge(left, right, on='subject_id', how='inner'))
>>  Name_x   id_x   subject_id   Name_y   id_y
0   Amy      2      sub2         Billy    1
1   Allen    3      sub4         Brian    2
2   Alice    4      sub6         Bryce    4
3   Ayoung   5      sub5         Betty    5
```

Pandas – DataFrame

- **how= 'outer' : 합집합**

```
print (pd.merge(left, right, on='subject_id', how='outer'))
>> Name_x   id_x   subject_id   Name_y   id_y
0   Alex     1.0    sub1         NaN      NaN
1   Amy      2.0    sub2         Billy    1.0
2   Allen    3.0    sub4         Brian    2.0
3   Alice    4.0    sub6         Bryce    4.0
4   Ayoung   5.0    sub5         Betty    5.0
5   NaN      NaN    sub3         Bran     3.0
```

Pandas – DataFrame

- **how= 'left' : 왼쪽 기준**

```
print (pd.merge(left, right, on='subject_id', how='left'))
>>  Name_x   id_x   subject_id   Name_y   id_y
0   Alex     1.0    sub1         NaN      NaN
1   Amy      2.0    sub2         Billy    1.0
2   Allen    3.0    sub4         Brian    2.0
3   Alice    4.0    sub6         Bryce    4.0
4   Ayoung   5.0    sub5         Betty    5.0
```

Pandas – DataFrame

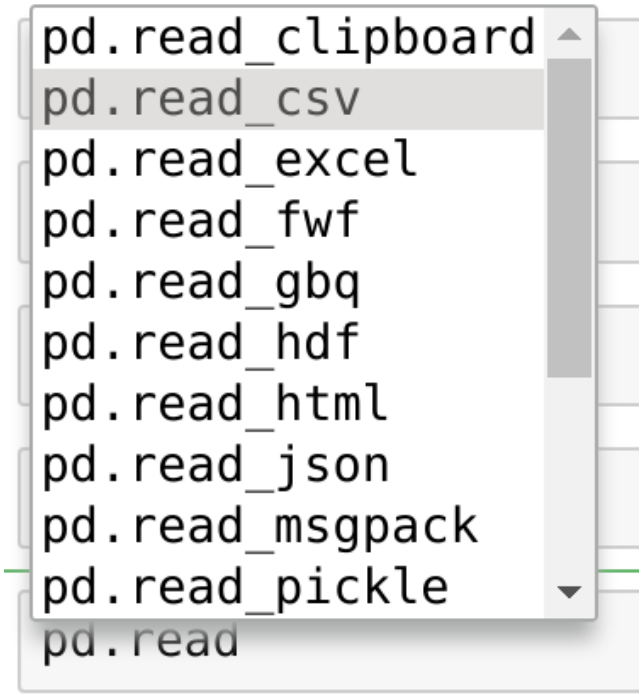
- **how= 'right' : 오른쪽 기준**

```
print (pd.merge(left, right, on='subject_id', how='right'))
>>  Name_x   id_x   subject_id   Name_y   id_y
0    Amy      2.0      sub2       Billy    1.0
1    Allen    3.0      sub4       Brian    2.0
2    Alice    4.0      sub6       Bryce    4.0
3    Ayoung   5.0      sub5       Betty    5.0
4    NaN      NaN      sub3       Bran     3.0
```

Pandas – Getting data

- 데이터 파일 읽기 - 여러가지 포맷 지원

  (CSV, Excel, HTML, JSON, SQL 등)



Pandas – Getting data

A1 법규위반

	A	B	C	D	E	F	G	H	I
1	법규위반	주야	발생건수	사망자수	부상자수	중상	경상	부상신고	
2	과속	주	159	34	334	140	178	16	
3	과속	야	218	73	348	200	139	9	
4	교차로 통행방법 위반	주	8817	82	14031	3915	9530	586	
5	교차로 통행방법 위반	야	5904	29	9728	2401	6884	443	
6	기타	주	9388	141	14070	4271	9217	582	
7	기타	야	6073	56	9218	2348	6457	413	
8	보행자 보호의무 위반	주	3772	80	3914	2063	1729	122	
9	보행자 보호의무 위반	야	3334	94	3535	1802	1626	107	
10	신호위반	주	12552	210	20396	6816	12765	815	
11	신호위반	야	12755	179	21724	7177	13749	798	
12	안전거리 미확보	주	12408	57	22392	4509	16882	1001	
13	안전거리 미확보	야	9867	40	17422	3025	13644	753	
14	안전운전 의무 불이행	주	62608	1740	91076	28081	58472	4523	
15	안전운전 의무 불이행	야	62783	2132	92942	26451	62209	4282	
16	중앙선 침범	주	6825	243	12427	4658	7339	430	
17	중앙선 침범	야	6193	202	11008	3846	6770	392	

Pandas – DataFrame

- **CSV 파일로부터 DataFrame 생성하기**

```
df = pd.read_csv('/home/dongsu/Downloads/2012 년 _1 당사자 _ 법규위반별 _ 주야별 _ 교통사고 .csv', encoding='euc-kr')
```

```
df.head()
```

	법규위반	주야	발생건수	사망자수	부상자수	중상	경상	부상신고
0	과속	주	159	34	334	140	178	16
1	과속	야	218	73	348	200	139	9
2	교차로 통행방법 위반	주	8817	82	14031	3915	9530	586
3	교차로 통행방법 위반	야	5904	29	9728	2401	6884	443
4	기타	주	9388	141	14070	4271	9217	582

```
df.shape
```

```
>>> (16, 8)
```

Pandas – DataFrame

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 8 columns):
법규위반    16 non-null object
주야       16 non-null object
발생건수    16 non-null int64
사망자수    16 non-null int64
부상자수    16 non-null int64
중상       16 non-null int64
경상       16 non-null int64
부상신고    16 non-null int64
dtypes: int64(6), object(2)
memory usage: 1.1+ KB
```

df.describe()

	발생건수	사망자수	부상자수	중상	경상	부상신고
count	16.000000	16.000000	16.000000	16.00000	16.000000	16.000000
mean	13978.500000	337.000000	21535.312500	6356.43750	14224.375000	954.500000
std	19415.823506	631.878997	28382.728283	8398.71464	18699.348084	1378.408261
min	159.000000	29.000000	334.000000	140.00000	139.000000	9.000000
25%	5371.000000	56.750000	7892.000000	2276.75000	5275.000000	324.500000
50%	7821.000000	88.000000	13229.000000	3880.50000	8278.000000	512.500000
75%	12444.000000	204.000000	20728.000000	5197.50000	13670.250000	802.250000
max	62783.000000	2132.000000	92942.000000	28081.00000	62209.000000	4523.000000

Pandas – Common Functions

- pd.reset_index() : index 를 새로운 column 으로 추가합니다 .

- pd.set_index(column) : 해당 column 을 index 로 설정합니다 .

- df.rename(columns={old_name : new_name}) : column 이름을 변경합니다 .

- df.groupby(column).agg() : 설정한 column 으로 grouping 하여 집계연산을 수행합니다 .

- pd.pivot_table(df, index, aggfunc) : 해당 column 을 기준으로 새로운 df 를 만듭니다 .

- pd.crosstab(df.index, df.columns) : 빈도를 계산하기 위한 pivot_table 의 특수한 경우

- apply(function) : function on row/column of dataframe

- applymap(function) : function on element of dataframe

- map(function) : function to series

Matplotlib

Matplotlib

- 파이썬에서 자료를 chart 나 plot 으로 시각화 (visualization) 하는 패키지
- 정형화된 차트나 플롯 이외에도 저수준 api 를 사용한 다양한 시각화 기능을 제공
- Line / scatter / contour / surface / box plot
- Histogram
- Bar chart

Matplotlib

matplotlib
Version 3.0.2

Fork me on GitHub

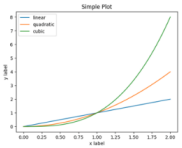
Tutorials

This page contains more in-depth guides for using Matplotlib. It is broken up into beginner, intermediate, and advanced sections, as well as sections covering specific topics.

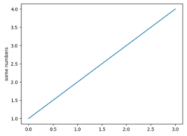
For shorter examples, see our examples page. You can also find external resources and a FAQ in our user guide.

Introductory

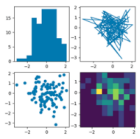
These tutorials cover the basics of creating visualizations with Matplotlib, as well as some best-practices in using the package effectively.



Usage Guide



Pyplot tutorial



Sample plots in Matplotlib

Image tutorial