11. python 데이터 분석 라이브러리

1. 데이터분석 라이브러리개요
2. numpy
3. pandas
4. matplatib

1. 데이터분석 라이브러리의 개요

- **python의 주요 데이터분석 라이브러리**
  - 넘파이(NumPy)
    - Python 데이터분석의 기본적인 기능들을 제공
    - 벡터 및 행렬 연산과 관련된 편리한 기능들 제공
  - 판다스(Pandas)
    - Series, DataFrame 등의 자료 구조를 활용하여 데이터 분석에 우수한 성능 발휘
    - 대량의 데이터를 빠른 속도로 처리 가능
  - 맷플롯립(Matplotlib)
    - 데이터 분석 결과에 대한 시각화를 빠르고 직관적으로 수행

1. 데이터분석 라이브러리의 개요

실습예제

```
#%매직명령어
#그래프 출력 과정을 불 수 있도록 설정
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib
```

라이브러리 설치 : console에서 다음과 같이 입력

```
pip install numpy
pip install pandas
pip install matplot
```

```
#50개의 난수 생성
data=np.random.rand(50)
print(type(data))
print(data)
```

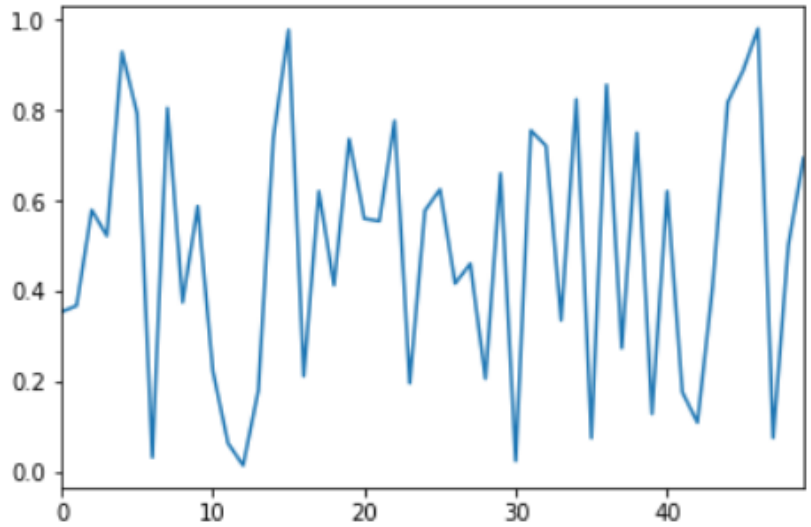
```
# 넘파이 배열을 판다스의 시리즈 자료형으로 변환(인덱스,데이터의 조합)
seri=pd.Series(data)
print(type(seri))
print(seri)
```

1. 데이터분석 라이브러리의 개요

실습예제

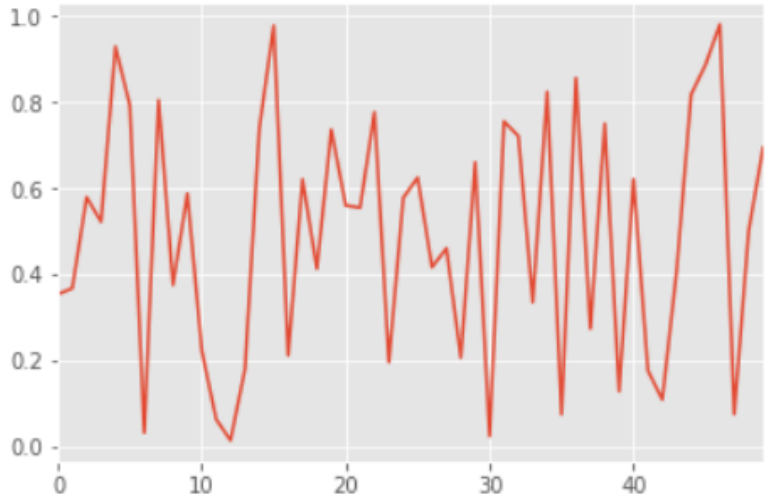
```
1  # x축 인덱스 값, y축 랜덤 값
2  seri.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10d242390>
```




```
1  #그래프 스타일 변경
2  matplotlib.style.use("ggplot")
3
4  # x축 인덱스 값, y축 랜덤 값
5  seri.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10cec6160>
```



1. 데이터분석 라이브러리의 개요

실습 예제

```
1 #rand(rows, cols)
2 #50개의 난수 생성
3 data_set=np.random.rand(10,3)
4 print(data_set)
5 print(type(data_set)) # 변수의 자료형 확인
6 print(data_set.shape) #행렬의 차원 확인
```

```
[[0.34485559 0.28497075 0.78006357]
 [0.50854493 0.67141959 0.30071159]
 [0.41970225 0.4573258  0.96723893]
 [0.1959138  0.42263276 0.49936199]
 [0.71791036 0.59629664 0.03647939]
 [0.38704617 0.08416631 0.48284201]
 [0.66055622 0.01640587 0.37022955]
 [0.73999638 0.16915373 0.41333187]
 [0.41217824 0.54741988 0.75350144]
 [0.25845176 0.9562979  0.08382933]]
<class 'numpy.ndarray'>
(10, 3)
```

1. 데이터분석 라이브러리의 개요

□ 실습 예제

```
1  #넘파이 행렬을 판다스의 데이터 프레임으로 변환
2  #데이터프레임: 행과 열로 데이터를 조회할 수 있음
3  df=pd.DataFrame(data_set, columns=["A","B","C"])
4  print(df)
5  print(type(df))
```

```
          A         B         C
0  0.344856  0.284971  0.780064
1  0.508545  0.671420  0.300712
2  0.419702  0.457326  0.967239
3  0.195914  0.422633  0.499362
4  0.717910  0.596297  0.036479
5  0.387046  0.084166  0.482842
6  0.660556  0.016406  0.370230
7  0.739996  0.169154  0.413332
8  0.412178  0.547420  0.753501
9  0.258452  0.956298  0.083829
<class 'pandas.core.frame.DataFrame'>
```

```
1  #넘파이 행렬을 판다스의 데이터 프레임으로 변환
2  #데이터프레임: 행과 열로 데이터를 조회할 수 있음
3  df=pd.DataFrame(data_set, columns=["A","B","C"])
4  df
5  #print(df)
6  #print(type(df))
```

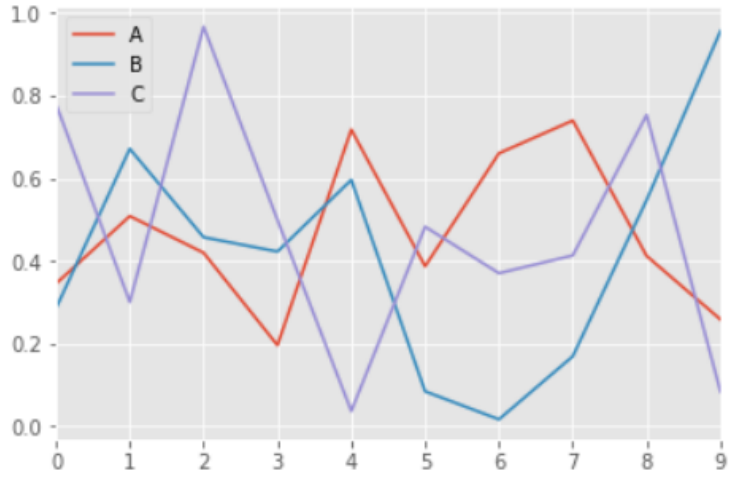
	A	B	C
0	0.344856	0.284971	0.780064
1	0.508545	0.671420	0.300712
2	0.419702	0.457326	0.967239
3	0.195914	0.422633	0.499362
4	0.717910	0.596297	0.036479
5	0.387046	0.084166	0.482842
6	0.660556	0.016406	0.370230
7	0.739996	0.169154	0.413332
8	0.412178	0.547420	0.753501
9	0.258452	0.956298	0.083829

1. 데이터분석 라이브러리의 개요

실습 예제

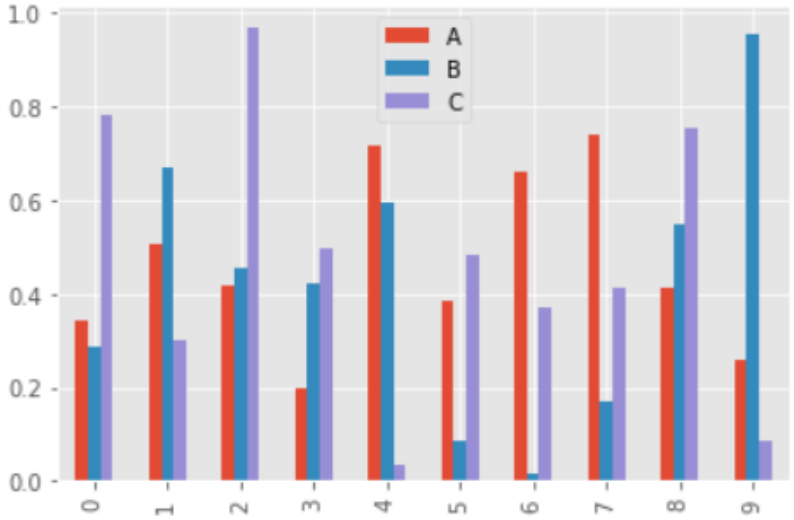
```
1  #데이터 프레임 자료로 그래프로 출력
2  df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10d92bcf8>
```




```
1  df.plot(kind="bar")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10d9d6710>
```

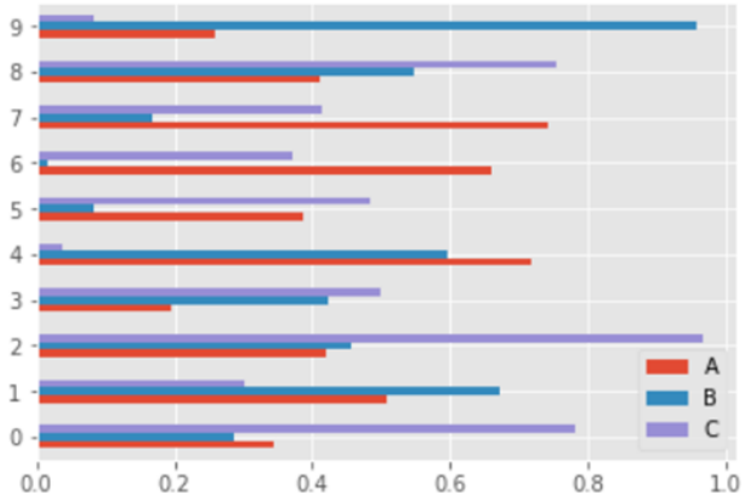


1. 데이터분석 라이브러리의 개요

□ 실습예제

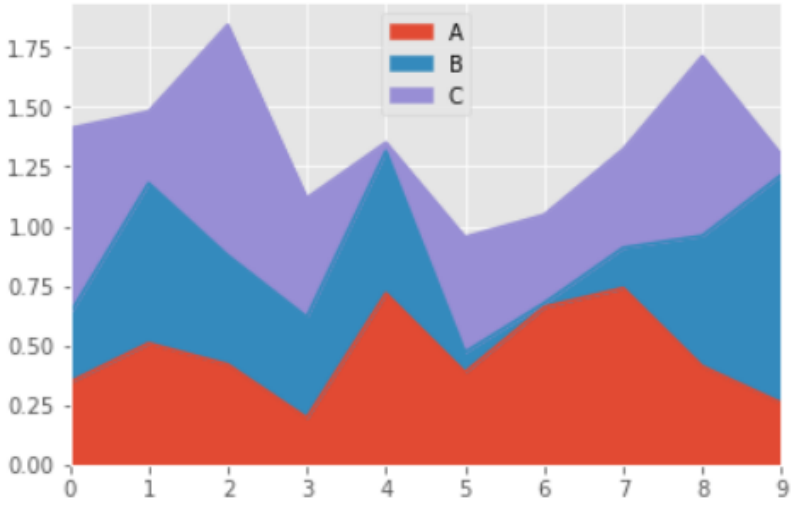
```
1 df.plot(kind="barh")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10da6a198>
```




```
1 df.plot(kind="area")
```

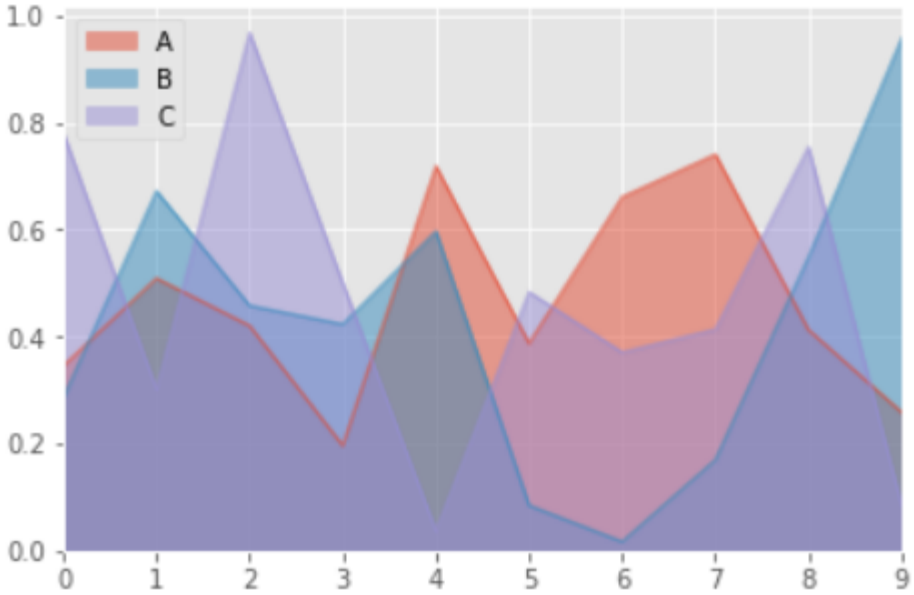
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10db1a908>
```



1. 데이터분석 라이브러리의 개요

```
1  df.plot(kind="area",stacked=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10dbc92b0>
```

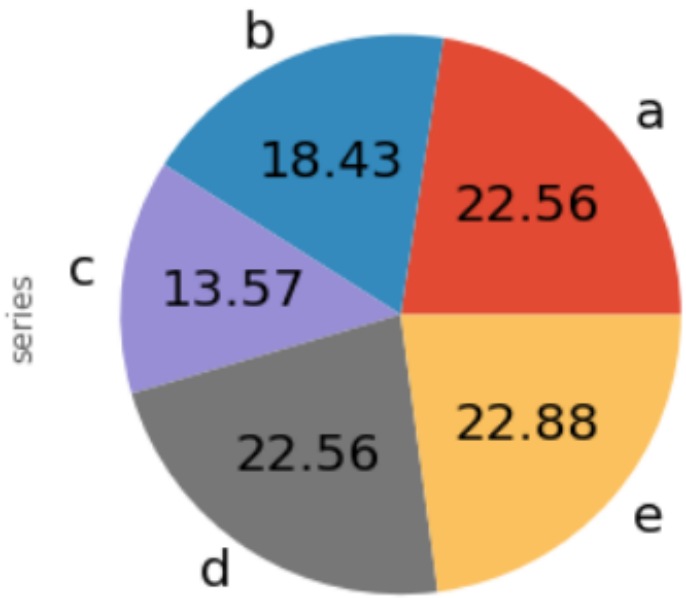


1. 데이터분석 라이브러리의 개요

실습

```
1  #넘파이의 런덤값 5개->판다스의 시리즈 자료형 변환
2  seri2=pd.Series(np.random.rand(5),index=["a","b","c","d","e"],name="series")
3  seri2.plot(kind="pie",autopct="%.2f",fontsize=20,figsize=(8,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a10ecf2b00>
```



2. NumPy

numpy

- 벡터 및 행렬 연산에 특화된 라이브러리
- array단위 데이터를 관리함, 행렬(matrix)와 비슷함
- 동적으로 생성되는 리스트와 달리 numpy 배열은 정적 메모리 할당
- 딥런닝을 위한 텐서플로우에서 효과적 활용
- pandas와 함께 데이터 분석에 많이 사용됨

순서	**①** **②** **③**	**numpy 모듈 가져오기** **파이썬 데이터형(예: 리스를) numpy형식으로 변환** **numpy에서 제공하는 기능 수행(예 평균)**
명령문	① ② ③	import numpy as np x= np.array([1, 3, 5]) print(x.mean(x))

2. NumPy

Numpy 배열

```
1  #넘파이 모듈 import
2  import numpy as np
```

```
1  #리스트
2  data1=[1,2,3,4,5]
3  #리스트를 넘파일 배열로 변환
4  arr1=np.array(data1)
5  print(arr1)
6  print(type(arr1))
7  print(arr1.shape) #배열의 차원
```

```
[1 2 3 4 5]
<class 'numpy.ndarray'>
(5,)
```

1행 5열

x=np.array([1,2,3,4])(O)
x=np.array(1,2,3,4) (X)

###
data1.mean() # 에러발생
arr1.mean() #가능

```
1  data2=[[1,2,3],[4,5,6]]
2  arr2=np.array(data2)
3  print(arr2)
4  print(arr2.shape) # (2,3) 2god 3duf 2ckdnjs godduf
```

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

동일
data2=[1,2,3,4,5,6]
arr2=np.array(data2).reshape(2,3)

2. NumPy

Numpy 배열

1(one)값으로 채우기

```
y1=np.ones(10) #1차원 배열에 1채우기
y2=np.ones([3,4]) #2차원 배열에 1채우기
y3=np.ones(12).reshape(3,4) # y2와 동일
```

제로(zero) 값으로 채우기

```
y4=np.zeros(10) #1차원 배열에 1채우기
y5=np.zeros((3,4)) #2차원 배열에 1채우기
y6=np.zeros(12).reshape(3,4) # y5와 동일
```

2. Numpy

Numpy 배열

```
1  #배열의 인덱싱
2  #스트링 리스트를 넘파이 배열로 변환
3  name=np.array(['김철수','이승호','박철로','김철수','홍성민','마이클','존'])
4  #7god 4dufdml skstnqkftod ,rand 정규분포 난수 발생
5  data=np.random.randn(7,4)
```

```
1  name
```

```
array(['김철수', '이승호', '박철로', '김철수', '홍성민', '마이클', '존'], dtype='<U3')
```

```
1  data
```

```
array([[ 0.029571  , -1.02470138,  0.19541975, -1.15945331],
       [-0.24573456,  0.35462478, -0.18879155, -1.066812  ],
       [ 0.21052688,  0.4470919 ,  0.38105085, -1.65561695],
       [ 1.13060954,  1.16859601, -1.01551847,  0.52469282],
       [ 0.48930345,  0.10967681,  2.66805596, -0.69250182],
       [-1.42472847,  0.5924564 ,  2.04153651,  1.36501049],
       [ 1.21215466, -0.74601457,  1.02244184, -0.31328491]])
```

2. Numpy

Numpy 배열 슬라이싱

list 슬라이싱

```
list1=[[1,11],[2,12],[3,13]]
print(list1[1][1])
print(list1[1,1])
print(list[:,1])
```

numpy 배열 슬라이싱

```
import numpy as np
list1=[[1,11],[2,12],[3,13]]
np_arr=np.array(list1)
print(np_arr[1][1])
```

```
import numpy as np
list1=[[1,11],[2,12],[3,13]]
np_arr=np.array(list1)
print(np_arr[1,1])
```

```
import numpy as np
list1=[[1,11],[2,12],[3,13]]
np_arr=np.array(list1)
print(np_arr[:,1])
```

```
import numpy as np
list1=[[1,11],[2,12],[3,13]]
np_arr=np.array(list1)
print(np_arr[:2,1])
```

numpy 배열 연산

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

```
np.minimum(x,y)
np.maximum(x,y)
```

```
print(x+y), x+10
print(x-y)
print(x*y)
print(x/y)
print(x//y)
print(x%y)
```

```
a=np.arange(1,13).reshape(3,4)
np.sqrt(a)
np.mean(a) #a.mean()
np.sum(a), # a.sum()
np.sum(a, axis=0) # 열합계
np.sum(a, axis=1) # 행합계
np.sqrt(a)
```

```
b=[[5,3,4],[2, 5,1],[7,6,9]]
arr_b=np.array(b)
np.sort(b)
np.sort(b, axis=1) #행기준 정렬
np.sort(b, axis=0) #열기준 정렬
np.sort(b, axis=0)[::-1] #내림차순 열기준 정렬
##내림차순 정렬은 열기준만 가능
arr_b.transpose() #전치행렬
```

실습

numpy 배열 연산

```
1  names=np.array(['김철수','이승호','박철로','김철수','홍성민','마이클','존'])
2  ages=np.array([20,25,20,23,28,29,30,31,30,40,45])
3  print(np.unique(names))
4  print(len(np.unique(names)))
5  print(np.unique(ages))
```

```
['김철수' '마이클' '박철로' '이승호' '존' '홍성민']
6
[20 23 25 28 29 30 31 40 45]
```

numpy 배열 연산

불리언(Boolean)

```
b=[[5,3,4],[2, 5,1],[7,6,9]]
arr_b=np.array(b)

b>5        #리스트일 때 결과
arr_b>b  #numpy배열일 때 결과
np.all(arr_b > 5) #arr_b의 모든 요소가 5보다 클때 True
np.any(arr_b>5) #arr_b의 어떤 요소가 5보다 클때 True
```

등간격 나누기

```
np.linspace(0,15,4) # 시작값, 종료값, 생성갯수
np.linspace(0,15,4,retstep=True)
np.linspace(0,15,4,endpoint=False)
np.logspace(1,2,4) # log10 10
```

데이터분석 실습-

영화평점 데이터 실습

- https://grouplens.org/datasets/movielens
- older datasets /MovieLens 1M Dataset / ml-1.zip 다운로드
- http:files.grouplens.org/datasets/movielens/ml-lm.zip
- 다운로드 후 d:/data/movielens 디렉토리에 압축해제
- 영화에 대한 평점 데이터 불러오기

```
data=np.loadtxt("D:/data/movielens/ratings.dat",
                delimiter="::",dtype=np.int64)
```

- 처음 5행의 데이터만 확인

```
data[:5,:]
```

- 데이터의 형태 확인(행,열)

```
data.shape
```

```
1  data.shape
(1000209, 4)
```

영화평점 데이터 실습

- 전체 평균 평점 계산

```
mean_rating_total=data[:,2].mean() #data[:,2] 전체 행, 2번째 열
mean_rating_total
```

- 사용자 아이디 수집

```
user_ids=np.unique(data[:,0])
print(user_ids)
print(len(user_ids))
```

영화평점 데이터 실습

- 사용자별 평점평균

```
mean_values=[]
for user_id in user_ids:
    data_for_user=data[data[:,0]==user_id,:]
    value=data_for_user[:,2].mean()
    mean_values.append([user_id,value])

mean_values[:5] :# 전체에서 0~6번행까지만 출력
```

- 리스트를 넘파이 배열로 변환

```
mean_array=np.array(mean_values,dtype=np.float32)
print(mean_array[:5])
print(mean_array.shape)
```

- 평점결과 csv 파일로 저장

```
np.savetxt("d:/data/movielens/resutl.csv",mean_array,
           fmt="%.1f",delimiter=",")
```

3.Pandas

판다스(Pandas)

- Numpy와 함께 데이터분석 라이브러리
- 행과 열로 구성된 데이터 객체를 만들어 다룸
- 안정적으로 대용량의 데이터들을 처리 도구
- Pandas의 기본적으로 정의되는 자료구조: Series, DataFrame
- 빅 데이터 분석에서 높은 수준의 성능, 대량의 데이터를 빠른 속도로 처리 가능
- Series – 1차원 배열, 복수의 행(row)과 하나의 열로 구성, Index로 데이터에 접근
- DataFrame-2차원 배열(다수의 행과 열고 구성), 표 형태

```
import numpy as np
import pandas as pd
```

3. Pandas

Series

```
# pandas 사용하기
import numpy as np # numpy 도 함께
import import pandas as pd
```

```
# Series 정의하기
x = pd.Series([4, 3, 5, 8])
print(x)
print(type(x))

# Series의 값만 확인하기
print(x.values)

# Series의 인덱스만 확인하기
print(x.index)

# Series의 자료형 확인하기
print(x.dtype)
```

Series

```
# 인덱스 설정.
x2 = pd.Series([4, 7, 5, 3], index=['d', 'b', 'a', 'c'])
print(x2)

#dictionary 자료형으로 Series data로 만들기
- dictionary의 key가 Series의 index가 된다
sdata = {'Kim': 35000, 'Beomwoo': 67000, 'Joan': 12000, 'Choi': 4000}
obj3 = pd.Series(sdata)
print(obj3)

obj3.name = 'Salary'
obj3.index.name = "Names"
print(obj3)

# index 변경
obj3.index = ['A', 'B', 'C', 'D']
print(obj3)
```

Series

```
# index 사용
x=pd.Series([7,3,5,8], index=['서울','대구','부산','광주'])
print(x)
print(x['서울'])
print(x[['서울','부산']])

#Series 연산
x=pd.Series([7,3,5,8], index=['서울','대구','부산','광주'])
y=pd.Series([2,4,5,1], index=['대구','부산','서울','대전'])
print(x+y)

model=[1,3,2,4,2,3]
x=pd.Series(model)
print(pd.unique(x))
```

```
# Series 슬라이스(주의할 점: 인덱스가 재설정되지 않는다.
a=x[3:6]
print(a.index)  -> 결과?
a[2] -> 결과는?
a.index=[0,1,2] # 인덱스 변경
```

Dataframe(2차원 배열)

```
# Data Frame 정의하기
# 이전에 DataFrame에 들어갈 데이터를 정의해주어야 하는데,
# 이는 python의 dictionary 또는 numpy의 array로 정의할 수 있다.
data = {'name': ['민준', '현우', '서연', '동현', '지현'],
        'year': [2013, 2014, 2015, 2016, 2015],
        'points': [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
df

## 행과 열의 구조를 가진 데이터가 생긴다.
```

	name	year	points
0	민준	2013	1.5
1	현우	2014	1.7
2	서연	2015	3.6
3	동현	2016	2.4
4	지현	2015	2.9

DataFrame

```
data = [['Beomwoo', 'Beomwoo', 'Beomwoo', 'Kim', 'Park'],
        'year': [2013, 2014, 2015, 2016, 2015],
        'points': [1.5, 1.7, 3.6, 2.4, 2.9]]
df = pd.DataFrame(data,columns=['name',year,')
df
```

	names	years	points
0	민준	2013	1.5
1	현우	2014	1.5
2	서연	2015	1.7
3	동현	2015	2.4
4	지현	2016	2.9

```
# 행 방향의 index
print(df.index)
# 열 방향의
print(df.columns)

# 각 인덱스에 대한 이름 설정하기
df.index.name = 'Num'
df.columns.name = 'Info'
Print(df)
```

DataFrame

```
data = {'name': ['Beomwoo', 'Beomwoo', 'Beomwoo',  ' Kim', 'Park'],
        'year': [2013, 2014, 2015, 2016, 2015],
        'points': [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data)
df
```

```
# 각 인덱스에 대한 이름 설정하기
df.index.name = 'Num'
df.columns.name = 'Info'
df
```

Info	name	year	points
Num			
0	Beomwoo	2013	1.5
1	Beomwoo	2014	1.7
2	Beomwoo	2015	3.6
3	Kim	2016	2.4
4	Park	2015	2.9

DataFrame

```
# DataFrame을 만들면서 columns와 index를 설정할 수 있다.
df2 = pd.DataFrame(data, columns=['year', 'name', 'points', 'penalty'],
                   index=['one', 'two', 'three', 'four', 'five'])
df2
```

	year	name	points	penalty
one	2013	Beomwoo	1.5	NaN
two	2014	Beomwoo	1.7	NaN
three	2015	Beomwoo	3.6	NaN
four	2016	Kim	2.4	NaN
five	2015	Park	2.9	NaN

DataFrame을 정의하면서, data로 들어가는 python dictionary와 columns의 순서가 달라도 알아서 맞춰서 정의된다.
하지만 data에 포함되어 있지 않은 값은 NaN(Not a Number)으로 나타나게 되는데, 이는 null과 같은 개념이다.
NaN값은 추후에 어떠한 방법으로도 처리가 되지 않는 데이터이다.
따라서 올바른 데이터 처리를 위해 추가적으로 값을 넣어줘야 한다.

DataFrame

# describe() 함수: DataFrame의 계산 가능한 값들에 대한 다양한 계산 값을 보줌
df2.describe()

	year	points
count	5.000000	5.000000
mean	2014.600000	2.420000
std	1.140175	0.864292
min	2013.000000	1.500000
25%	2014.000000	1.700000
50%	2015.000000	2.400000
75%	2015.000000	2.900000
max	2016.000000	3.600000

DataFrame Indexing

```
data = {"names": ["Kilho", "Kilho", "Kilho", "Charles", "Charles"],
        "year": [2014, 2015, 2016, 2015, 2016],
        "points": [1.5, 1.7, 3.6, 2.4, 2.9]}
df = pd.DataFrame(data, columns=["year", "names", "points", "penalty"],
                                    index=["one", "two", "three", "four", "five"])
df
```

	year	names	points	penalty
one	2014	Kilho	1.5	NaN
two	2015	Kilho	1.7	NaN
three	2016	Kilho	3.6	NaN
four	2015	Charles	2.4	NaN
five	2016	Charles	2.9	NaN

DataFrame Indexing

DataFrame에서 열을 선택하고 조작하기

```
print(df['year'])
print(df.year)
print(df[['year','points']])
```

- 특정 열에 값을 대입

```
df['penalty'] = 0.5
df
# 또는
df['penalty'] = [0.1, 0.2, 0.3, 0.4, 0.5] # python의 List나 numpy의 array
df
```

DataFrame Indexing

- *새로운 열을 추가하기*

```
df['zeros'] = np.arange(5) #df.zeros는 불가,생성되지 않은 필드
df
```

- *Series로 열 추가*

```
val = pd.Series([-1.2, -1.5, -1.7], index=['two','four','five'])
df['debt'] = val
df
```

Series로 열 추가할 때는 val와 같이 넣으려는 data의 index에 맞춰서 데이터가 들어간다. python list나 numpy array로 데이터를 넣을때와 가장 큰 차이점이다.

```
df['net_points'] = df['points'] - df['penalty']
df['high_points'] = df['net_points'] > 2.0
df
```

DataFrame Indexing

- 열 삭제

```
del df['high_points']
del df['net_points']
del df['zeros']
print(df)

print(df.columns)
```

```
df.index.name = 'Order'
df.columns.name = 'Info'
df
```

DataFrame Indexing

- DataFrame에서 행 선택 및 조작

```
# 0번째 부터 2(3-1) 번째까지 가져온다. # 뒤에 써준 숫자번째의 행은 뺀다.
df[0:3]
```

```
# tow라는 행부터 four라는 행까지 가져온다.
# 뒤에 써준 이름의 행을 빼지 않는다.
df['two':'four'] # 하지만 비추천!]
```

```
# 아래 방법을 권장한다.
# .loc 또는 .iloc 함수를 사용하는 방법.
print(df.loc['two'])  # 반환 형태는 Series
print(df.loc['two':'four'])
print(df.loc['two':'four', 'points'])
print(df.loc[:,'year'] # == df['year'])
print(df.loc[:,['year','names']])
print(df.loc['three':'five','year':'penalty'])
```

DataFrame Indexing

행삽입

```
df.loc['six',:] = [2013,'Jun',4.0,0.1,2.1]
print(df)
```

```
# .iloc 사용:: index 번호를 사용한다.
print(df.iloc[3]) # 3번째 행을 가져온다.

print(df.iloc[3:5, 0:2])

print(df.iloc[[0,1,3], [1,2])

print(df.iloc[:,1:4])

print(df.iloc[1,1])
```

DataFrame에서의 boolean Indexing

```
print(df['year'] > 2014)
print(df.loc[df['year']>2014,:])
print(df.loc[df['names'] == 'Kilho',['names','points']])
# numpy에서와 같이 논리연산을 응용할 수 있다.
df.loc[(df['points']>2)&(df['points']<3),:]
# 새로운 값을 대입할 수도 있다.
df.loc[df['points'] > 3, 'penalty'] = 0
print(df)
```

5. Data

```
# DataFrame을 만들때 index, column을 설정하지 않으면 기본값으로 0부터 시작하는 정수형 숫자로 입력된다.
df = pd.DataFrame(np.random.randn(6,4)) df
print(df)
```

```
df.columns = ['A', 'B', 'C', 'D'] df.index = pd.date_range('20160701', periods=6)
#pandas에서 제공하는 date range함수는 datetime 자료형으로 구성된, 날짜 시각등을 알 수 있는 자료형을 만드는 함수
print(df.index)
print(df)
```

Data

```
# np.nan은 NaN값을 의미한다.
df['F'] = [1.0, np.nan, 3.5, 6.1, np.nan, 7.0]
df
```

NaN 없애기

```
# 행의 값중 하나라도 nan인 경우 그 행을 없앤다.
df.dropna(how='any')
df
```

```
# 행의 값의 모든 값이 nan인 경우 그 행으 없앤다.
df.dropna(how='all')
```

주의 drop함수는 특정 행 또는 열을 drop하고난 DataFrame을 반환한다.
즉, 반환을 받지 않으면 기존의 DataFrame은 그대로이다.
아니면, inplace=True라는 인자를 추가하여, 반환을 받지 않고서도
기존의 DataFrame이 변경되도록 한다

Data

```
#NaN값에 값 넣기
df.fillna(value=0.5)

# nan값인지 확인하기
df.isnull()

# F열에서 nan값을 포함하는 행만 추출하기
df.loc[df.isnull()['F'],:]

pd.to_datetime('20160701')
```

Data(삭제)

```
# 특정 행 drop하기
df.drop(pd.to_datetime('20160701'))

# 2개 이상도 가능
 df.drop([pd.to_datetime('20160702'),pd.to_datetime('20160704')])

# 특정 열 삭제하기
df.drop('F', axis = 1)

# 2개 이상의 열도 가능
df.drop(['B','D'], axis = 1)
```

분석용 함수들

```
data = [[1.4, np.nan], [7.1, -4.5], [np.nan, np.nan], [0.75, -1.3]]
df = pd.DataFrame(data, columns=["one", "two"], index=["a", "b", "c", "d"])
df
```

```
# 행방향으로의 합(즉, 각 열의 합)
df.sum(axis=0)

# 열방향으로의 합(즉, 각 행의 합)
df.sum(axis=1)

# NaN값은 배제하고 계산
#NaN 값을 배제하지 않고 계산하려면 아래와 같이 skipna에 대해 false를 지정.
df.sum(axis=1, skipna=False)

# 특정 행 또는 특정 열에서만 계산하기
df['one'].sum()

df.loc['b'].sum()
```

pandas에서 DataFrame에 적용되는 함수들

sum() 함수 이외에도 pandas에서 DataFrame에 적용되는 함수는 다음의 것들이 있다.
count 전체 성분의 (NaN이 아닌) 값의 갯수를 계산
min, max 전체 성분의 최솟, 최댓값을 계산
argmin, argmax 전체 성분의 최솟값, 최댓값이 위치한 (정수)인덱스를 반환
idxmin, idxmax 전체 인덱스 중 최솟값, 최댓값을 반환
quantile 전체 성분의 특정 사분위수에 해당하는 값을 반환 (0~1 사이)
sum 전체 성분의 합을 계산
mean 전체 성분의 평균을 계산
median 전체 성분의 중간값을 반환
mad 전체 성분의 평균값으로부터의 절대 편차(absolute deviation)의 평균을 계산
std, var 전체 성분의 표준편차, 분산을 계산
cumsum 맨 첫 번째 성분부터 각 성분까지의 누적합을 계산 (0에서부터 계속 더해짐)
cumprod 맨 첫번째 성분부터 각 성분까지의 누적곱을 계산 (1에서부터 계속 곱해짐)

함수 사용

```
df2 = pd.DataFrame(np.random.randn(6, 4),
                   columns=["A", "B", "C", "D"],
                   index=pd.date_range("20160701", periods=6))
df2
```

```
# A열과 B열의 상관계수 구하기
df2['A'].corr(df2['B'])

# B열과 C열의 공분산 구하기
df2['B'].cov(df2['C'])
```

- **groupby()함수**
  - df.groupby(df[‘year’).mean()
  - df.groupby(‘year’).mean()

정렬함수 및 기타함수

```
dates = df2.index random_dates = np.random.permutation(dates)
df2 = df2.reindex(index=random_dates, columns=["D", "B", "C", "A"])
df2
```

```
# index와 column의 순서가 섞여있다.
# 이때 index가 오름차순이 되도록 정렬해보자
df2.sort_index(axis=0)

# column을 기준으로?
df2.sort_index(axis=1)

# 내림차순으로는?
df2.sort_index(axis=1, ascending=False)

# 값 기준 정렬하기
# D열의 값이 오름차순이 되도록 정렬하기
df2.sort_values(by='D')
```

정렬함수 및 기타함수

```
# B열의 값이 내림차순이 되도록 정렬하기
df2.sort_values(by='B', ascending=False)

df2["E"] = np.random.randint(0, 6, size=6)
df2["F"] = ["alpha", "beta", "gamma", "gamma", "alpha", "gamma"]
df2

# E열과 F열을 동시에 고려하여, 오름차순으로 하려면?
df2.sort_values(by=['E','F'])

# 지정한 행 또는 열에서 중복값을 제외한 유니크한 값만 얻기
df2['F'].unique()

# 지정한 행 또는 열에서 값에 따른 개수 얻기
df2['F'].value_counts()
```

정렬함수 및 기타함수

```
# 지정한 행 또는 열에서 값에 따른 개수 얻기
df2['F'].value_counts()

# 지정한 행 또는 열에서 입력한 값이 있는지 확인하기
df2['F'].isin(['alpha','beta'])
# 아래와 같이 응용할 수 있다.

# F열의 값이 alpha나 beta인 모든 행 구하기
df2.loc[df2['F'].isin(['alpha','beta']),:]

# F열의 값이 alpha나 beta인 모든 행 구하기
df2.loc[df2['F'].isin(['alpha','beta']),:]
```

사용자 정의 함수 적용

```
df3 = pd.DataFrame(np.random.randn(4, 3),
            columns=["b", "d", "e"],
            index=["Seoul", "Incheon", "Busan", "Daegu"])
df3

func = lambda x: x.max() - x.min()

def func(x):
            return x.max()-x.min()

df3.apply(func, axis=0) #열방향

df3.apply(func.axis=1) #행방향
```

2. pandas 파일 처리

□ 파이썬 문법 사용 파일 처리(csv모듈사용(x))

```
input_file="d:/data/input.csv"
output_file="d:/data/output1.csv"
#nuwline='' 개행문자를 무시하고 읽음, 이옵션이 없으면 빈라인이 추가됨
with open(input_file,'r', newline='')as reader:
    with open(output_file,'w',newline='')as writer:
        header=reader.readline()  #첫줄을 읽음(헤더)
        header=header.strip()      #strip() :문자열 좌우 공백,탭,개행문자 제거
        header_list=header.split(',')  #","를 기준으로 문자열을 나눠서 리스트르 리턴
        print(header_list)
        # header_list의 각문자 사이에 쉼표를 사입하고 리스트를 문자열로 변환
       # map(str,list) : list 요소를 String 으로 설정
       #''.join(list) : list의 요소를 ","를 삽입하여 연결
        writer.write(','.join(map(str,header_list))+'₩n') #파일에 기록
        for row in reader:
            row=row.strip()
            row_list=row.split(',')
            print(row_list)
            writer.write(','.join(map(str,row_list))+'₩n')
```

Pandas 사용 파일처리

```
import sys
input_file="d:/data/input.csv"
output_file="d:/data/output2.csv"

#csv 파일을 읽어서 데이터프레임 양식으로 저장
data_frame=pd.read_csv(input_file)
print(data_frame)

# 데이터프레임 데이터를 csv로 저장
data_frame.to_csv(output_file,index=False)
```

파이썬에 내장된 csv 모듈 사용 파일 처리

```
import csv
import sys
input_file="d:/data/input.csv"
output_file="d:/data/output3.csv"
with open(input_file,'r', newline='')as csv_in_file:
    with open(output_file,'w',newline='')as csv_out_file:

        #필드 구분자를 쉼표(,)로 설정
        filereader=csv.reader(csv_in_file,delimiter=',')
        filewriter=csv.writer(csv_out_file,delimiter=',')

        for row_list in filereader:
            print(row_list)
            filewriter.writerow(row_list)
```

파이썬 CSV 모듈 사용(원하는 행만 선택)

```
import csv
import sys
input_file="d:/data/input.csv"
output_file="d:/data/output4.csv"
with open(input_file,'r', newline='')as csv_in_file:
    with open(output_file,'w',newline='')as csv_out_file:
        filereader=csv.reader(csv_in_file)
        filewriter=csv.writer(csv_out_file)
        header=next(filereader)   #한 라인을 읽음(첫 라인)
        filewriter.writerow(header)
        for row in filereader:
            supplier=str(row[0]).strip()    #행의 첫번째 필드의 좌우 공백 제거
            cost=str(row[3]).strip('$').replace(',','')   # '$ 문자를 제거하고 (,)를 제거
            # supplier가 A이고 cost가 600000인 데이터 선택
            if supplier=='A' and float(cost) >= 600000.0:
                filewriter.writerow(row)
                print(row)
```

pandas 사용 필터

```
import pandas as pd
import sys
input_file="d:/data/input.csv"
output_file="d:/data/output5.csv"
#csv 파일을 읽어서 데이터프레임으로 저장
df=pd.read_csv(input_file)
#데이터프레임에서 Cost 필드(실수값)를 스트링으로 바꾼 후
#$문자을 제거하고 (,)를 공백으로 바꿈
df['Cost']=df['Cost'].str.strip('$').str.replace(",","").astype(float)
#Supplier Name 필드에 A가 포함되거나 Cost가 600000보다
#큰 데이터만 선택하여 새로운 데이터프레임에 저장
result=df.loc[(df['Supplier Name'].str.contains('A')) & (df['Cost']>600000.0),:]
result.to_csv(output_file,index=False)
```

pandas-데이터베이스 처리

```
import pandas as pd
import numpy as no
from sqlalchemy import create_engine

# mysql에 접속
# mysql+mysqldb://아이디:비번@호스트/데이터베이스
engine=create_engine("mysql+mysqldb://pgm:1234@localhost/pyboard",
            encoding='utf-8')
conn=engine.connect()
df=pd.read_csv('d:/data/input.csv',engine='python',encoding='utf-8')
df.head()

df.columns=['Supplier Name','Invoice Number','Part Number','Cost','Purchase Date']
# replace : 테이블 삭제 후 저장
# applend : 레코드 추가
# fail : 실패, 기본동작
df.to_sql(name='input',con=engine,if_exists='replace',index=False)
```

실습-항공운항데이터 분석

- **url : http://stat-computing.org/dataexpo/2009/**

```
df1=pd.read_csv("data/2006.csv",sep=",")
df2=pd.read_csv("data/2007.csv",sep=",")
df3=pd.read_csv("data/2008.csv",sep=",")
df1.append(df2)
df1.append(df3)
df1.shape
```

```
df1.columns  # 필드 확인
```

```
# 전체 데이터에서 년,월, 도착지연시간,출발 지연시간 추출
df2=df1[['Year','Month','ArrDelay','DepDelay']]
```

```
df2.head() # 레코드 0번에서 5개보기
df2.tail() # 끝에서 끝에서 5개 보기
df2.shape
df2=df2.dropna(how="any") #NaN 값이 있는 레코드 제거
#년, 월 필드로 group by 하여 도착지연시간과 출발 지연시간 합 구하기
result=df2.groupby(['Year','Month'], as_index=False).sum()
result
result.to_csv("d:/data/airline/result.csv",sep=",") #결과를 csv파일로 저장
```

3. Matplotlib

```
# 매직 명령어
# 그래프를 조작할 수 있음
# 그래프 제목 오른쪽의 stop interation 버튼을 누르기 전까지 그래프 수정가능
# %matplotlib nbagg
# 생성되 닝후에는 조작할 수 없는 옵션

%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
# 난수 발생, 정규분포 난수 n개 생성
# 시리즈 : 1차원 배열,데이터 프레임 : 2차원 배열
# cumsum() :누적합
s=pd.Series(np.random.randn(10).cumsum(), index=np.arange(0,100,10))
s
```

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

```
df.plot()
```

```
%matplotlib nbagg #그래프를 조작
#%matplotlib inline

df["B"].plot()
df["C"].plot()
df["D"].plot()
```

```
s2=pd.Series(np.random.rand(16),index=list('abcdefghijklmnop'))
s2
```

```
s2.plot()
s2.plot(kind='bar')
```

```
s2.plot(kind='barh')
```

```
from matplotlib import rc, font_manager
# 한글 처리를 위해 폰트 설정
font_name=font_manager
            .FontProperties(fname="c:/Windows/Fonts/malgun.ttf").get_name()
rc('font',family=font_name)
df2=pd.DataFrame(np.random.rand(6,4),
                 index=['one','two','three','four','five','six'],
                 columns=pd.Index(['A','B','C','D'],name="종류"))
df2
```

```
df2.plot(kind='barh',stacked=True)
```

```
# 히스토그램 :x변수가 가질 수 있는 값의 구간, 갯수를 막대 형태로 출력
# 히스토그램은 값만 필요하고 인텍스는 필요하지 않음
# normal() 정규분포
s3=pd.Series(np.random.normal(0,100,size=200))
s3.head()
```

```
# 음수 부호가 깨지지 않도록 설정
plt.rcParams['axes.unicode_minus']=False
# 축 구간은 기본값으로 10개로 설정 됨
s3.hist()
# 축 구간은 기본값으로 10개로 설정 됨
s3.hist(bins=50)
# normed 정규분호
# 정규분포 : 종 모양을 가진다.
s3.hist(bins=100, normed=True)
```

```
# 산점도 : 서로 다른 독립변수 x1,x2의 관계를 파악할 때 사용
# normal(mu,sigma,size=(rows,cols))
#mu :중앙값(median),sigma : 표준편차
x1=np.random.normal(1,1,size=(100,1))
x2=np.random.normal(-2,4,size=(100,1))
# concatenate : 열방향 연결, axis=1 가로 방향 ,axis=0 세로방향
x=np.concatenate((x1,x2),axis=1)
# 첫 5행 출력
print(x[:5])
```

```
df3=pd.DataFrame(x,columns=["x1","x2"])
df3.head()
```

```
# x1과 x2의 상관관계
# x1(x축), x2(y축)
# 양의 상관관계 : 산점도가 좌특하단에서 우측 상단으로
# 음의 상관관계 : 산점도가 좌측상단에서 우측하단으로
# 이 그래프에서 상관관계를 찾을 수 없음
plt.scatter(df3["x1"],df3["x2"])
```

```
import csv
from datetime import datetime
from matplotlib import pyplot as plt
filename='d:/data/temp/temperature_2014.csv'
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
    dates, highs,lows=[],[],[]
    for row in reader:
        try:
            # strftime : 날짜를 문자열로
            # strptime : 문자열을 날짜로
            current_date=datetime.strptime(row[0],"%Y-%m-%d")
            high=int(row[1]) # 최고 온도
            low=int(row[3]) # 최저 온도
        except ValueError:
            print(current_date,'missing date')
        else: # 예외가 발생하지 않으면 리스트 추가
            dates.append(current_date)
            highs.append(high)
            lows.append(low)
```

```
# 그래프 생성
fig=plt.figure(dpi=128,figsize=(10,6))
#  x축 dates, y축 highs, 색상 red
# 투명도 0.5, 범위 :0.0~1.0
plt.plot(dates,highs, c='red',alpha=0.5)
plt.plot(dates,lows, c='blue',alpha=0.5)
# 두개의 선 사이(highs~lows)를 색칠함, alpha=0.1 투명도 0.1
plt.fill_between(dates,highs,lows,facecolor='blue',alpha=0.1)
# 그래프 스타일
plt.title('온도 통계',fontsize=20)
plt.xlabel("기간",fontsize=16)
plt.ylabel("온도",fontsize=16)
plt.tick_params(axis='both',labelsize=16) # 눈금 표시
plt.show()
```

```
# 막대 그래프- # R용 시각화 패키지 ggplot의 스타일 적용
plt.style.use('ggplot')
customers=['A','B','C','D','E']
customers_index=range(len(customers))
sale_amounts=[127,90,201,111,232]
#figure(그림) 객체를 만든다.
fig=plt.figure()
# figure 객체 하위에 그래프를 추가, # 1행 1열 1개의 하위 그래프
ax1=fig.add_subplot(1,1,1)
# 막대 그래프 bar(x축, y축, 가운데 정렬, 막대색상)
ax1.bar(customers_index, sale_amounts,align='center',color='darkblue')
# x축 눈금 위치
ax1.xaxis.set_ticks_position('bottom')
# y축 눈금 위치
ax1.yaxis.set_ticks_position('left')
# xticks(수치데이터, 대체 텍스트)
# 0,1,2,3,4 대신 A,B,C,D,E로 출력됨
plt.xticks(customers_index,customers,rotation=0,fontsize='small')
plt.xlabel('고객명')
plt.ylabel('판매금액')
plt.title('판매통계')
# 그래프를 이미지로 저장,dpi 해성도, tight 그림을 둘러산 여백 제거
plt.savefig('d:/data/images/bar_plot.png',dpi=400,bbox_inches='tight')
plt.show()
```

```
import matplotlib.pyplot as plt
# x축 데이터
input_values=[1,2,3,4,5]
# y축 데이터
squares=[1,4,9,16,25]
# 그래프 출력-선그래프 plot(x,y)
plt.plot(input_values,squares, linewidth=5)
#plt.scatter(input_values,squares,c='blue') # 산점도 그래프
# 그래프 제목
plt.title("Square Numbers", fontsize=24)
# x축 레이블
plt.xlabel("Value", fontsize=14)
# y축 레이블
plt.ylabel("Square of Value", fontsize=14)
# 눈금 사이즈
plt.tick_params(axis='both', labelsize=14)
plt.show()
```

```
import matplotlib.pyplot as plt
# 1~1000 리스트
x_values=list(range(1,100))
# 제곱한 값
y_values=[x**2 for x in x_values]
# scatter(x좌표, x좌표...) 산점도(데이터를 점으로 표현됨)
# c="blue" 색상 설정
# edgecolor='none' 외곽선 제거
# s 점의 개수
plt.scatter(x_values, y_values,c='blue',edgecolor='none',s=5)
plt.title("Square number",fontsize=14)
plt.xlabel("Value",fontsize=14)
plt.ylabel("Squar of value",fontsize=14)
plt.tick_params(axis="both",labelsize=14)
# 축의 번위 지정(x최소 값, x 최대값, y최소값, y최대값)
plt.axis([0,100,0,10000])
plt.show()
```