1.판다스 사용법

In [57]:

```
import pandas as pd #데이터 분석 라이브러리 임포트
```

In [3]:

```
#pandas의 dataframe 생성
names =['bab','kate','john','merry','kay']
births=[931,421,48,578,973]
custom=[1,5,25,13,23232]

BabyDataSet = list(zip(names,births)) #zip 명령어로 데이터 형태를 리스트로 만들겠다
df=pd.DataFrame(data = BabyDataSet, columns = ['Names', 'Births'])

df.head() #데이터 프레임의 상단 부분을 출력
```

Out[3]:

	Names	Births
0	bab	931
1	kate	421
2	john	48
3	merry	578
4	kay	973

In [7]:

```
#데이터 프레임 기본 정보 출력
#데이터프레임의 열의 타입 정보 출력
print(df.dtypes)
```

```
Names    object
Births    int64
dtype: object
```

In [8]:

```
#데이터 프레임의 행의 형태 정보
print(df.index)
```

Out[8]:

```
RangeIndex(start=0, stop=5, step=1)
```

In [9]:

```
#데이터 프레임 열의 형태 정보
print(df.columns)
```

Out[9]:

```
Index(['Names', 'Births'], dtype='object')
```

In [11]:

```
#데이터 프레임의 하나의 열을 선택
print(df['Names'])
print(df['Births'])
```

```
0     bab
1    kate
2    john
3   merry
4     kay
Name: Names, dtype: object
0   931
1   421
2    48
3   578
4   973
Name: Births, dtype: int64
```

In [22]:

```
#데이터 프레임의 행 선택
df[0:3]
```

Out[22]:

	Names	Births
0	bab	931
1	kate	421
2	john	48

In [23]:

```
#1. 필터링 기능
df[df['Births']>100]
```

Out[23]:

	Names	Births
0	bab	931
1	kate	421
3	merry	578
4	kay	973

In [24]:

```
#2.평균을 계산
#mean() 함수는 각 열들의 데이터 타입을 체크한 후, 연산이 가능한 열의 평균값만을 반환
df.mean()
```

```
C:\Users\YJ\AppData\Local\Temp/ipykernel_13620/46693316.py:
2: FutureWarning: Dropping of nuisance columns in DataFrame re
ductions (with 'numeric_only=None') is deprecated; in a future ve
rsion this will raise TypeError.  Select only valid columns before ca
lling the reduction.
  df.mean()
```

Out[24]:

```
Births    590.2
dtype: float64
```

3. Numpy사용법

넘파이는 수치 계산을 위해 만들어진 파이썬 라이브러리. Numerical Python의 줄임말임 넘파이는 배열 개념으로 변수를 사용하며 벡터, 행렬 등의 연산을 쉽고 빠르게 수행

In [58]:

```
import numpy as np
```

In [32]:

```
#넘파이 배열 선언
#1차원 배열이 3개, 2차원 배열이 5개의 값을 가지는 15개 숫자를 생성하는 코드
# reshape() :차원의 배열을 생성하는 함수
arr1 = np.arange(15).reshape(3,5)
arr1
```

Out[32]:

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

In [29]:

```
#1차원 배열
arr0 = np.arange(15)
arr0
```

Out[29]:

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

In [35]:

```
# 넘파이 배열정보 확인 shpae()
arr1.shape
```

Out[35]:

```
(3, 5)
```

In [36]:

```
arr1.dtype
```

Out[36]:

```
dtype('int32')
```

In [39]:

```
#배열에 원하는 데이터값 입력  array()
arr2 = np.array([6,7,8])
arr2
```

Out[39]:

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

In [41]:

```
#zeros(): 0으로 채워진 배열을 생성
#ones() : 1으로 채워진 배열을 생성
arr3 = np.zeros((3,4))
arr3
```

Out[41]:

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

In [43]:

```
#0으로 이루어진 1차원 배열 생성
arr4 = np.zeros(15)
arr4
```

Out[43]:

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

In [49]:

```
#1으로 이루어진 다차원 배열 생성
arr5 = np.ones((2,3,4,5,6))
arr5
```

Out[49]:

```
array([[[[[1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.]],

         [[1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.]],

         [[1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.],
          [1., 1., 1., 1., 1., 1.]],

         [[1., 1., 1., 1., 1., 1.],
          [1   1   1   1   1   1 ]
```

In [50]:

```
arr6 = np.array([
   [1,2,3],
   [4,5,6]
], dtype = np.float64)

arr7 = np.array([
   [7,8,9],
   [10,11,12]
], dtype = np.float64)

#사칙연산 출력
print("arr6+ arr7=")
print(arr6+arr7,"\n")
print("arr6- arr7=")
print(arr6-arr7,"\n")
print("arr6* arr7=")
print(arr6*arr7,"\n")
print("arr6/ arr7=")
print(arr6/arr7,"\n")
```

```
arr6+ arr7=
[[ 8. 10. 12.]
 [14. 16. 18.]] 

arr6- arr7=
[[-6. -6. -6.]
 [-6. -6. -6.]] 

arr6* arr7=
[[ 7. 16. 27.]
 [40. 55. 72.]] 

arr6/ arr7=
[[0.14285714 0.25       0.33333333]
 [0.4        0.45454545 0.5       ]] 
```

4. Matplotlib

In [59]:

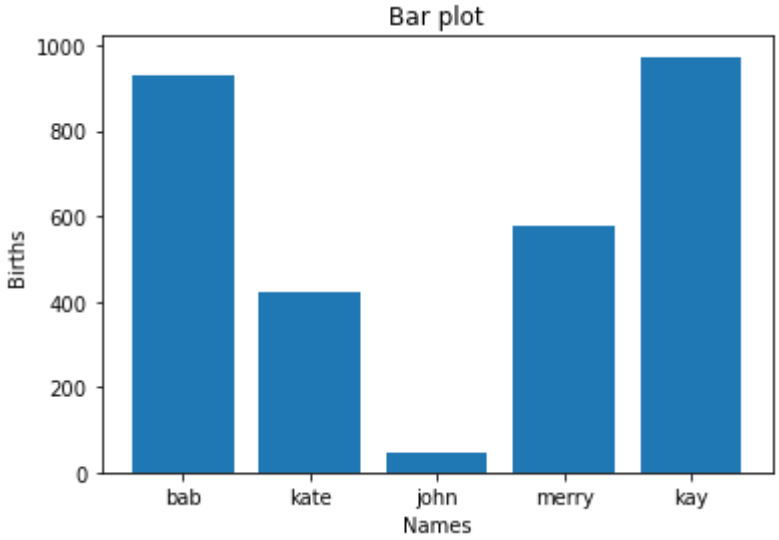
```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [53]:

```
#막대 그래프 출력하기
y = df['Births']
x = df['Names']

plt.bar(x,y) #막대 그래프 객체 생성
plt.xlabel('Names') #x축 제목
plt.ylabel('Births') #y축 제목
plt.title('Bar plot') #그래프 제목


plt.show #그래프 출력
```

Out[53]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

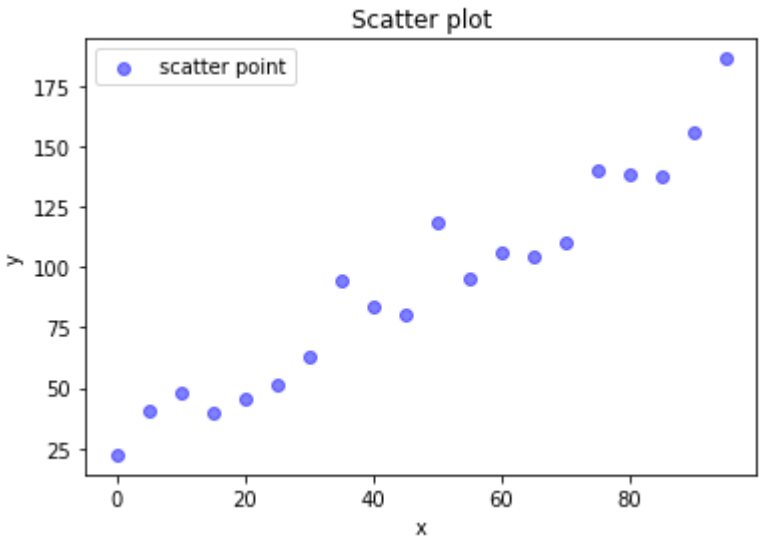


In [63]:

```
#산점도 그래프 출력
np.random.seed(19920613)

#산점도 데이터 생성
x = np.arange(0.0, 100.0, 5.0)
y = (x*1.5) + np.random.rand(20) *50

#산점도 데이터 출력
plt.scatter(x,y, c="b", alpha =0.5, label="scatter point")
plt.xlabel("x") #x 축 라벨
plt.ylabel("y") #y 출 라벨
plt.legend(loc = 'upper left') #범례위치
plt.title('Scatter plot') #제목



plt.show
```

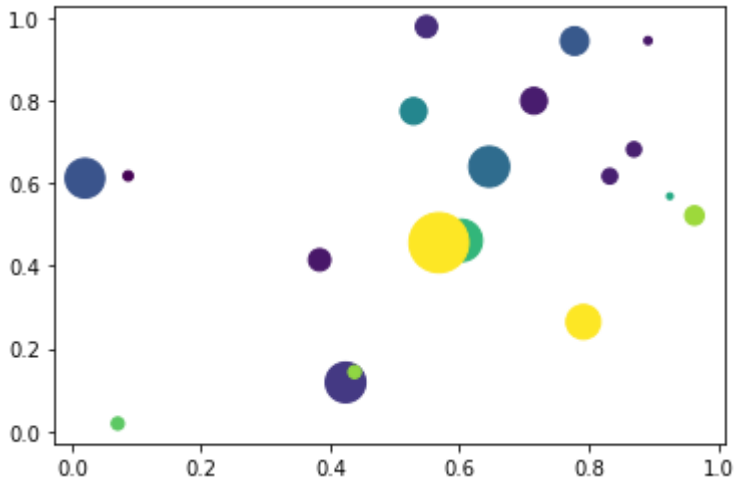
Out[63]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



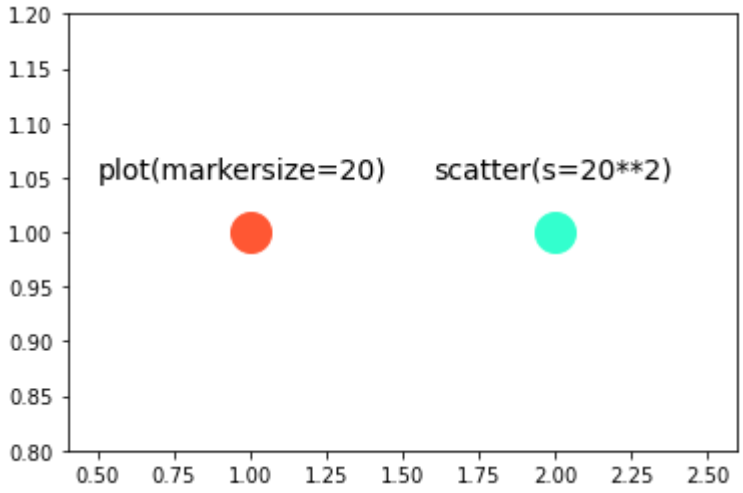
In [66]:

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)

n = 20
x = np.random.rand(n) #랜덤한 x 좌표20개
y = np.random.rand(n) #랜덤한 y 좌표20개
area = (30 * np.random.rand(n))**2 #마커의 크기
colors = np.random.rand(n) #색상

plt.scatter(x, y, s=area, c= colors) #산점도 생성
plt.show()
```



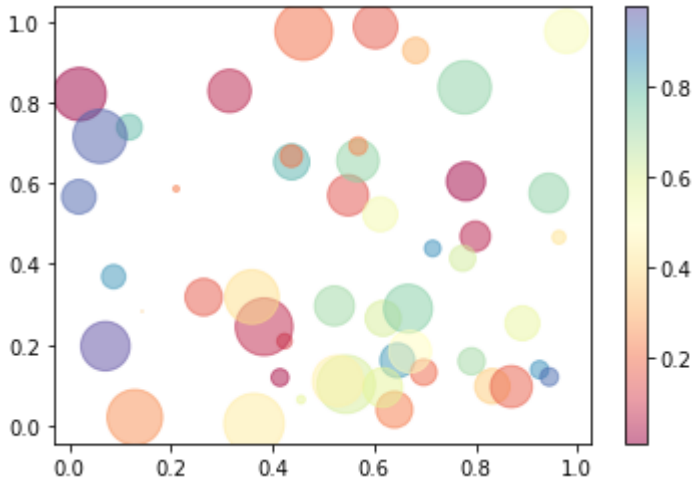
In [68]:

```
import matplotlib.pyplot as plt

plt.plot([1], [1], 'o', markersize=20, c='#FF5733')
plt.scatter([2], [1], s=20**2, c='#33FFCE')

plt.text(0.5, 1.05, 'plot(markersize=20)', fontdict={'size': 14})
plt.text(1.6, 1.05, 'scatter(s=20**2)', fontdict={'size': 14})
plt.axis([0.4, 2.6, 0.8, 1.2])
plt.show()
```



In [71]:

```
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)

n = 50
x = np.random.rand(n)
y = np.random.rand(n)
area = (30 * np.random.rand(n))**2
colors = np.random.rand(n)

plt.scatter(x, y, s=area, c=colors, alpha=0.5, cmap='Spectral') #alpha 마커의 투명도
plt.colorbar() #cmap 파라마터에 컬러맵에 해당하는 문자열 지정
plt.show()
```



In [ ]: