상관계수 (Correlation Coefficient)

- numpy를 이용하여 데이터의 상관계수를 구합니다.
- python 코드와 numpy의 함수의 속도차이를 비교합니다.

Index

In [1]:

```
import numpy as np
```

샘플 데이터 생성

In [2]:

```
data1 = np.array([80, 85, 100, 90, 95])
data2 = np.array([70, 80, 100, 95, 95])
```

2. 분산(variance)

- 1개의 이산정도를 나타냅니다.
- 편차제곱의 평균

$$variance = \frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}, (\bar{x} : 평균)$$

In [3]:

```
# variance code
def variance(data):

    avg = np.average(data)
    var = 0
    for num in data:
        var += (num - avg) ** 2
    return var / len(data)


# test code (분산, 표준편차)
print("variance :", variance(data1), variance(data2))
print("standard deviation :", variance(data1)**0.5, variance(data2)**0.5)
```

```
variance : 50.0 126.0
standard deviation : 7.0710678118654755 11.224972160321824
```

In [4]:

```
variance(data1), variance(data2), variance(data1) ** 0.5, variance(data2) ** 0.5
```

Out[4]:

```
(50.0, 126.0, 7.0710678118654755, 11.224972160321824)
```

In [5]:

```
np.var(data1), np.var(data2), np.std(data1), np.std(data2)
```

Out[5]:

```
(50.0, 126.0, 7.0710678118654755, 11.224972160321824)
```

일반 함수와 numpy 함수의 퍼포먼스 비교

In [6]:

```
p_data1 = np.random.randint(60, 100, int(1E5))
p_data2 = np.random.randint(60, 100, int(1E5))
```

In [7]:

```
# 일반함수
```

In [8]:

```
%%time
variance(p_data1), variance(p_data2)
```

```
CPU times: user 663 ms, sys: 27.7 ms, total: 691 ms
Wall time: 719 ms
```

Out[8]:

```
(133.29922553557807, 133.87624326241075)
```

In [9]:

```
# numpy
```

In [10]:

```
%%time
np.var(p_data1), np.var(p_data2)
```

```
CPU times: user 1.58 ms, sys: 1.19 ms, total: 2.78 ms
Wall time: 2.01 ms
```

Out[10]:

```
(133.29922553559996, 133.8762432624)
```

3. 공분산(covariance)

- 2개의 확률변수의 상관정도를 나타냅니다.
- 평균 편차곱

- 방향성은 보여줄수 있으나 강도를 나타내는데 한계가 있습니다.
  - 표본데이터의 크기에 따라서 값의 차이가 큰 단점이 있습니다.

$$covariance = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{n}, (\bar{x} : x\text{의평균}, \bar{y} : y\text{의평균})$$

In [11]:

```
# covariance function
# 표본상관계수를 사용 (모상관계수를 사용하면 -1을 제거)
def covariance(data1, data2):

    x_ = np.average(data1)
    y_ = np.average(data2)

    cov = 0
    for idx in range(len(data1)):
        cov += (data1[idx] - x_) * (data2[idx] - y_)
    return cov / (len(data1) - 1)
```

In [12]:

```
# teat code 1
# data1이 커짐으로 data2도 커진다.
data1 = np.array([80, 85, 100, 90, 95])
data2 = np.array([70, 80, 100, 95, 95])
covariance(data1, data2)
```

Out[12]:

```
93.75
```

In [13]:

```
# test code 2
# data3는 커지지만 data4는 작아진다.
data3 = np.array([80, 85, 100, 90, 95])
data4 = np.array([100, 90, 70, 90, 80])
covariance(data3, data4)
```

Out[13]:

```
-87.5
```

In [14]:

```
# test code 3
# 표본데이터의 크기가 커지면 공분산 값도 커진다.
# 두 데이터의 상관정도의 강도를 나타내는데 무리가 있다.
data5 = np.array([800, 850, 1000, 900, 950])
data6 = np.array([1000, 900, 700, 900, 800])
covariance(data5, data6)
```

Out[14]:

```
-8750.0
```

4. 상관계수(correlation coefficient)

- 공분산의 한계를 극복하기 위해서 만들어집니다.

- -1 ~ 1까지의 수를 가지며 0과 가까울수록 상관도가 적음을 의미합니다.
- x의 분산과 y의 분산을 곱한 결과의 제곱근을 나눠주면 x나 y의 변화량이 클수록 0에 가까워집니다.
- https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.corrcoef.html (https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.corrcoef.html)

$$correlation - coefficient = \frac{\text{공분산}}{\sqrt{x\text{분산} \cdot y\text{분산}}}$$

- 최종 상관계수

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \cdot \sum(y-\bar{y})^2}}$$

In [15]:

```
# 상관계수 함수
def cc(data1, data2):

    x_ = np.average(data1)
    y_ = np.average(data2)

    cov, xv, yv = 0, 0, 0

    for idx in range(len(data1)):
        cov += (data1[idx] - x_) * (data2[idx] - y_)
        xv += (data1[idx] - x_) ** 2
        yv += (data2[idx] - y_) ** 2

    return cov / ((xv*yv) ** 0.5)
```

In [16]:

```
# teat code 1
data1 = np.array([80, 85, 100, 90, 95])
data2 = np.array([70, 80, 100, 95, 95])
cc(data1, data2)
```

Out[16]:

```
0.944911182523068
```

In [17]:

```
# teat code 2
data3 = np.array([80, 85, 100, 90, 95])
data4 = np.array([100, 90, 70, 90, 80])
cc(data3, data4)
```

Out[17]:

```
-0.970725343394151
```

In [18]:

```
# teat code 3
data5 = np.array([800, 850, 1000, 900, 950])
data6 = np.array([1000, 900, 700, 900, 800])
cc(data5, data6)
```

Out[18]:

```
-0.970725343394151
```

In [19]:

```
# numpy
print(np.corrcoef(data1, data2)[0][1])
print(np.corrcoef(data3, data4)[0][1])
print(np.corrcoef(data5, data6)[0][1])
```

```
0.9449111825230682
-0.970725343394151
-0.970725343394151
```

5. 결정계수(cofficient of determination: R-squared)

- x로부터 y를 예측할수 있는 정도
- 상관계수의 제곱 (상관계수를 양수화)
- 수치가 클수록 회기분석을 통해 예측할수 있는 수치의 정도가 더 정확

In [20]:

```
print(data1, data2, data4, sep="\n")
print("\ncorrcoef")
print("data1, data2 :", np.corrcoef(data1, data2)[0][1])
print("data1, data4 :", np.corrcoef(data1, data4)[0][1])
print("\nR-squred")
print("data1, data2 :", np.corrcoef(data1, data2)[0][1]**2)
print("data1, data4 :", np.corrcoef(data1, data4)[0][1]**2)
```

```
[ 80  85 100  90  95]
[ 70  80 100  95  95]
[100  90  70  90  80]

corrcoef
data1, data2 : 0.9449111825230682
data1, data4 : -0.970725343394151

R-squred
data1, data2 : 0.892857142857143
data1, data4 : 0.9423076923076923
```

6. 프리미어리그 데이터 상관계수 분석

- 2016/2017시즌의 프리미어리그 성적에서 득점과 실점 데이터중에 승점에 영향을 더 많이 준 데이터는?

In [21]:

```
import pandas as pd
```

In [22]:

```
df = pd.read_csv("datas/premierleague.csv")
df.head(2)
```

Out[22]:

	name	gf	ga	points
0	Manchester City	106	27	100
1	Manchester United	68	28	81

In [23]:

```
datas = df.values
```

- pandas dataframe의 데이터를 pickle 파일로 저장하기

In [24]:

```
import pickle
```

In [25]:

```
with open("datas/premierleague.pkl", "wb") as f:
    pickle.dump(datas, f)
```

In [26]:

```
%ls datas
```

```
advertising.plk    premierleague.csv* premierleague.pkl
```

In [27]:

```
with open("datas/premierleague.pkl", "rb") as f:
    datas = pickle.load(f)
```

In [28]:

```
# 데이터 확인
datas[:3]
```

Out[28]:

```
array([['Manchester City', 106, 27, 100],
       ['Manchester United', 68, 28, 81],
       ['Tottenham Hotspur', 74, 36, 77]], dtype=object)
```

In [29]:

```
# 득점
gf = datas[:, 1].astype(np.int)
gf
```

Out[29]:

```
array([106,  68,  74,  84,  62,  74,  36,  44,  56,  39,  45,  45,  48,
        44,  34,  28,  37,  28,  35,  31])
```

In [30]:

```
# 실점
ga = datas[:, 2].astype(np.int)
ga
```

Out[30]:

```
array([27, 28, 36, 38, 38, 51, 39, 58, 60, 47, 55, 61, 68, 64, 54, 58, 56,
       56, 68, 56])
```

In [31]:

```
# 승점
points = datas[:, -1].astype(np.int)
points
```

Out[31]:

```
array([100,  81,  77,  75,  70,  63,  54,  49,  47,  44,  44,  44,  42,
        41,  40,  37,  36,  33,  33,  31])
```

In [32]:

```
# 득점과 승점의 상관계수 출력
corr_gf = np.corrcoef(gf, points)[0, 1]
corr_gf
```

Out[32]:

```
0.9318404636463514
```

In [33]:

```
# 실점과 승점의 상관계수 출력
```

In [34]:

```
corr_ga = np.corrcoef(ga, points)[0, 1]
corr_ga
```

Out[34]:

```
-0.8705940043262674
```

In [35]:

```
# 결정계수 : coefficient of determination
```

In [36]:

```
deter = {key: np.round(value ** 2, 2) for key, value in zip(["gf", "ga"], [corr_gf, corr_ga])}
deter
```

Out[36]:

```
{'gf': 0.87, 'ga': 0.76}
```

In [37]:

```
# 득점의 결정계수가 실점의 결정계수보다 더 높음
# 승점을 예측할때 실점보다는 득점이 더 잘 예측함 (전체 데이터를 더 잘 설명함)
# 승점의 상관계수가 득점이 더 높으므로 득점이 승점에 더 많은 영향을 줌
```