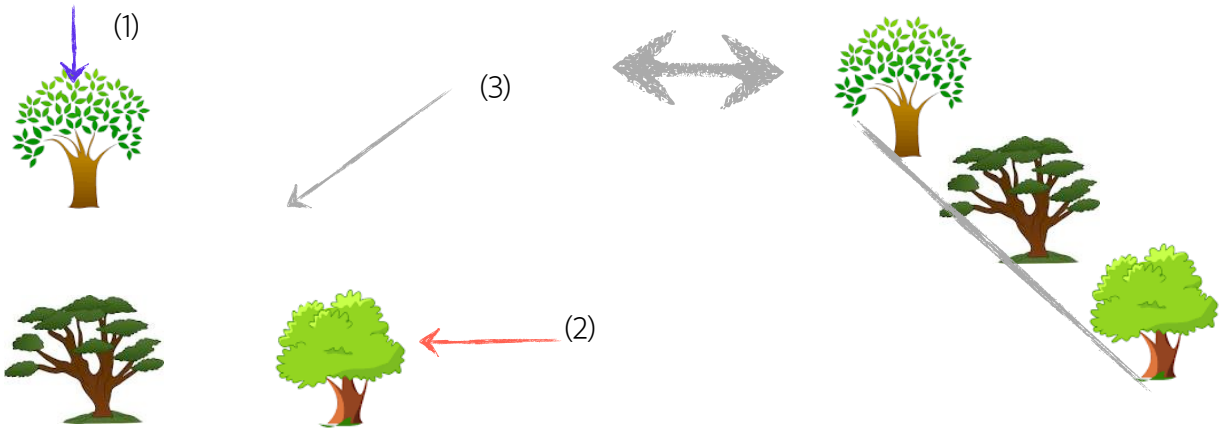
PCA 개념

공간적 개념



- 2차원 공간 정보를 1차원(직선)으로 표시한다면 어디에서 봐야 희생되는 정보(나무의 위치와 거리)가 최소일까?
- (1)에서 본다면  가 직선 상에 나타나지 않음
- (2)의 관점에서 본다면 역시  가  에 가려져 직선 상에 나타나지 않음
- 위치와 공간 정보를 최소화하는 관점은 (3)에서 보는 것이다. 그러나 이 역시 희생되는 정보가 있다.  와  의 거리는 실제 거리보다 가까워졌음
- 이는 2차원 공간 정보를 1차원 직선으로 표현하여 잃은 정보가 존재한다는 것이다
- 주성분분석은 이처럼 변수의 차원(개수)를 희생 정보를 최소화 하여 축약하는 방법이다.

변수의 개수 - 차원

- 기성복 하의 구매 - 허리둘레와 기장(우리나라는 손쉽게 줄일 수 있으므로 허리둘레만 활용)
- 예전에는 허리둘레, 허벅지 두께, 허리에서 무릎 길기, 무릎에서 발꿈치까지 길이 등 많은 측정값이 필요하였다. - 아니면 대충 허리 둘레에 맞춰 옷을 사고 줄여 입었음
- 지금은 길이에 대한 정보는 기장, 둘레에 대한 정보는 허리 둘레에 들어가 있어 이 두 값만 알면 어디서나 쉽게 바지를 구입할 수 있음
- 그럼 이 기장, 허리둘레는 이전의 길이, 허리둘레만 있나? 아니다. 예전의 관측값들의 정보를 모두 포함하고 있는 골든 지표이다.

- (기장, 허리둘레)는 예전의 모든 측정변수의 관측값으로 만들어진 주성분 변수임.
- 많은 측정값들이 2개의 주성분변수로 축약되므로 모든 사람들의 몸에 맞는 기성복은 존재하지 않는다. 기성복이 맞는 않는 사람은 big&tall 샵, 디자이너 샵에 가야 한다.

주성분분석 이론

원 변수 데이터 행렬, 변수벡터

데이터 행렬 : $X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}_{n\times p}$

변수벡터 : $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}_{p\times 1}$ (pX1) 벡터, where $\underline{x} \sim N(\underline{\mu}, \Sigma)$

$\underline{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_p \end{bmatrix}$ (mean matrix) , $\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{pmatrix}_{p\times p}$ (covariance matrix)

k번째 변수 데이터 벡터

$$\underline{x}_k = \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{kn} \end{bmatrix}_{n\times 1} \quad \text{(nX1) 벡터}$$

주성분 벡터

주성분 변수 벡터 $\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}_{p\times 1} = \begin{pmatrix} l_{11} & l_{12} & \cdots & l_{1p} \\ l_{21} & l_{22} & \cdots & l_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ l_{p1} & l_{p2} & \cdots & l_{pp} \end{pmatrix}_{p\times p} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}_{p\times 1} = L\underline{x}$

j번째 주성분변수 선형계수 벡터 : $\underline{l}_j = \begin{pmatrix} l_{j1} \\ l_{j2} \\ \vdots \\ l_{jp} \end{pmatrix}_p$: (pX1) 벡터

j번째 주성분변수 : $y_j = l_j' * \underline{x} = (l_{j1}\ l_{j2} \cdots\ l_{jp})_{1\times p} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}_{p\times 1}$: 결과 스칼라

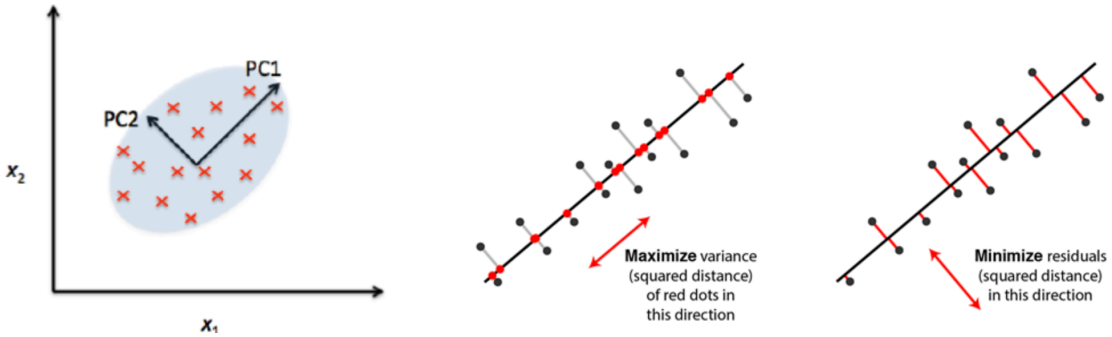
j번째 주성분 변수 데이터 : $\underline{y} = X_{n\times p} L_{p\times p}$: (nXp) 벡터

주성분 변수 성질

- 원 변수들의 선형결합이다.

$$y_j = l_j' * \underline{x} = (l_{j1}\ l_{j2} \cdots\ l_{jp})_{1\times p} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}_{p\times 1}$$

- 서로 독립이다
- $\underline{l}_i' * \underline{l}_j = 0$: 선형계수의 곱은 0이다. 즉, $E(Y_iY_j) = E(Y_i)E(Y_j)\ for\ i \neq j$
- 주성분 변수는 원 변수들의 변동을 설명하고 순서대로 설명력은 줄어든다.
- 주성분 변수 개수는 원변수의 개수만큼 존재하게 된다. 차이가 있다면 원변수는 공분산구조(상관계수가 0이 아님, 독립이 아님)를 가지고 있으나 주성분변수는 서로 독립이다.
- 주성분변수의 개수는 원변수의 개수와 동일하게 p만큼 존재한다.



주성분 구하기

개체에 대한 변수 정보는 변동(분산)에 의해 정의된다. 변수의 변동은 공분산행렬(혹은 상관행렬, 변수의 단위가 상이한 경우)에 의해 측정된다.

- 공분산행렬은 변수의 측정단위를 그대로를 반영한 것이고 상관행렬은 모든 변수의 측정단위를 표준화 한 것이다. 빅데이터 시대에는 표준화($\frac{x_i - \bar{X}_i}{s_X}$)보다는 중심화($x_i - \bar{X}_i$)를 선호한다. 이는 공분산 크기를 주성분 회전에 반영하기 위함이다.
- 그러므로 어느 행렬을 사용하는 것이 적절한가? 변수의 변동을 정확하게 반영하는 것은 공분산행렬이므로 중심화 후 공분산 행렬을 사용하는 것이 적절함

데이터 공분산(상관계수) 행렬 고유값, 고유벡터 구하기

공분산행렬 : $\Sigma = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_{pp} \end{pmatrix}_{p \times p}$

상관행렬 : $R = \begin{pmatrix} 1 & \rho_{12} & \cdots & \rho_{1p} \\ \sigma_{21} & 1 & \cdots & \rho_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ \rho_{p1} & \rho_{p2} & \cdots & 1 \end{pmatrix}_{p \times p}$

공분산행렬은 양반정치(positive definite)행렬이므로 다음을 만족하는 0보다 큰 실수이고 행렬의 차수 p개만큼 고유값(eigenvalue)이 존재한다.

(정의) 공분산행렬 Σ(상관계수행렬 R)에 대하여 $|\Sigma - \lambda I| = 0$을 만족하는 λ_i을 고유값(eigenvalues)이라 하고 고유방정식 $\Sigma \underline{x_i} = \lambda_i \underline{x_i}$ 에 λ_i를 대입하여 구한 벡터 $\underline{x_i}$를 고유벡터(eigenvectors)라 한다. 많은 고유벡터 중 $L2 - norm$이 1인 고유벡터를 부하(loading)벡터라 한다.

(고유값 구하기) $|\Sigma - \lambda I| = 0$ 만족하는 λ들을 고유값이라 한다.

주성분 고유값 및 고유벡터 성질

(공분산행렬) $\left|\Sigma - \lambda I\right| = 0$ 을 만족하는 고유값은

(1) $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_p \geq 0$ 양의 실수이고, 원변수의 개수(차수)만큼 존재한다.

(2) 고유값 λ_k에 대응하는 고유벡터 $\underline{x}_k$는 무수히 많이 존재함. - 주성분 요인 회전에 활용

(3) k-번째 주성분 변수의 분산은 λ_k이다.

(4) i-번째 주성분 변수와 k-번째 원 변수 상관계수는 서로 다른 주성분 변수는 서로 독립이다.

(5) 고유벡터 중 L2-norm=1 인 고유벡터 $\underline{e}_k$를 k-번째 주성분 변수의 선형계수로 사용한다.

(6) 원 변수의 변동(분산)합은 고유값의 합과 같다. $\sigma_{11} + \sigma_{22} + \ldots + \sigma_{pp} = \lambda_1 + \lambda_2 + \ldots + \lambda_p$

(7) k-번째 주성분 변동 기여율$= \dfrac{\lambda_k}{\lambda_1 + \lambda_2 + \ldots + \lambda_p}$

(8) 값이 제일 큰 λ_1에 의해 만들어지는 주성분변수를 제1주성분, λ_2에 의해 만들어지는 주성분변수를 제2주성분이라 함, 제3주성분, ….

(9) 그러므로 원변수가 p개이면 주성분변수도 p개만큼 구해진다.

(상관행렬) $p = \lambda_1 + \lambda_2 + \ldots + \lambda_p$

(1) 고유값 λ_k는 k-번째 주성분변수가 원변수의 변동을 설명하는 능력이다.

(2) k-번째 주성분 변동 기여율 : $\dfrac{\lambda_k}{p}$

주성분변수 구하기

|선형계수| |부하 loading|

공분산행렬 Σ 고유벡터 중 $\underline{e}'_k\underline{e}_k = 1$을 만족하는 Norm 고유벡터가 k-번째 주성분 변수의 선형계수

|주성분변수|

$\underline{y}_{k(n\times 1)} = X_{n\times p}\underline{e}_{k(p\times 1)}$: k-번째 주성분 변수

제1주성분 변수

$\underline{l}'_1\underline{l}_1 = 1$이면서 $V(\underline{l}'_1\underline{x})$을 최대화 하는 열벡터 $\underline{l}_1$를 구하고 이를 제1주성분변수 선형계수로 하여 주성분변수를 구한다. $\underline{y}_1 = \underline{l}'_1 X$

제2주성분 변수

$\underline{l}'_1\underline{l}_2 = 0,\ \underline{l}'_2\underline{l}_2 = 1$이면서 $V(\underline{l}'_2\underline{x})$을 최대화 하는 열벡터 $\underline{l}_2$를 구하고 이를 제2주성분변수 선형계수로 하여 주성분변수를 구한다. $\underline{y}_2 = \underline{l}'_2 X$

제1주성분변수와는 서로 독립이고 원변수 변동들의 설명력(원변수 변동 설명력)은 조금 낮다.

제3주성분 변수

$\underline{l}'_1\underline{l}_3 = 0,\ \underline{l}'_2\underline{l}_3 = 0,\ \underline{l}'_3\underline{l}_3 = 1$이면서 $V(\underline{l}'_3\underline{x})$을 최대화 하는 열벡터 $\underline{l}_3$를 구하고 이를 제3주성분변수 선형계수로 하여 주성분변수를 구한다. $\underline{y}_3 = \underline{l}'_3 X$

제3주성분변수는 제1, 제2 주성분변수와는 독립이고 변동 설명력은 낮은 주성분이다. 이렇게 계속 원변수의 개수만큼 주성분변수를 구한다.

통계소프트웨어는 주성분변수를 표준화(적어도 중심화)하여 사용한다.

|k-번째 주성분 변수의 변동 기여율|

$$\frac{\lambda_k}{\lambda_1+\lambda_2+\ldots+\lambda_p}(\text{under } \Sigma),\ \frac{\lambda_k}{\lambda_1+\lambda_2+\ldots+\lambda_p}(\text{under } R)$$

In Python 예제데이터

원 데이터 가정

표본크기: 변수의 개수와 확률변수(features)개수의 비는 최소 1:5를 만족해야 한다. 주성분 확률변수가 3개 적어도 15개 표본데이터가 필요하다. (Pallant, 2010)

표본데이터 확률변수는 상관관계가 높아야 한다: 상관관계가 낮은 데이터 구조의 경우 주성분방법으로 차원축소 경향성은 낮아진다.

선형성 **Linearity:** 원변수는 다변량 정규분포함수의 형태를 가지며 주성분변수는 원변수의 선형결합 향태이다.

이상치 **Outliers:** 주성분변수 결과에 불균형한(dispropotional) 영향을 미치므로 이상치가 없어여 한다.

Large variance implies more structure: 분산이 큰 축은 주성분으로 낮은 축은 노이즈(잔차)로 처리되어 차원을 축소한다.

예제 데이터 불러오기 http://203.247.53.31/Big_Data/data/KBO_bat1719.csv

```
import pandas as pd
df=pd.read_csv("/content/drive/My Drive/Data_storage/KBO/bat1719.csv")
df.info()
```

[데이터 설명] 2B 2루타 3B 3루타 HR 홈런 TB 루타 수 RBI 타점 SB 도루 성공 CS 도루 실패 BB 볼넷 HBP 사구(몸에 맞는 볼) SO 삼진아웃 GDP 병살타 SLG 장타율 OBP 출루율 E 에러 height/weight 선수의 키/몸무게 year_born 선수의 생년월일 position 선수의 수비위치 career 선수의 커리어 starting_salary 선수의 한국프로야구 입단연봉 OPS OPS(OBP+SLG)

전처리

1. 데이터 정규화

PCA는 최대 분산이 있는 성분을 식별하는 데 사용되며 성분에 대한 각 변수의 기여도는 분산 크기를 기반으로합니다. 측정 단위가 다른 비척도 데이터는 기능 간 분산의 상대적 비교를 왜곡 할 수 있으므로 PCA를 수행하기 전에 데이터를 정규화(최소 중심화)하는 것이 가장 좋습니다.

2. 고유 분해를위한 공분산 행렬 생성

모든 다른 차원 간의 가능한 모든 관계를 얻는 유용한 방법은 모두 간의 공분산을 계산하고 데이터에서 이러한 관계를 나타내는 공분산 행렬에 넣는 것입니다. 각 주성분에 의해 포착 된 누적 분산 백분율을 이해하는 것은 기능 집합을 줄이는 데 필수적인 부분입니다.

3. 최적의 주성분 개수 선택

주성분의 최적 개수는 성분 개수의 함수로 설명 된 누적 분산 비율(cumulative explained variance ratio)을 살펴봄으로써 결정됩니다.

예제 데이터 subsetting

'득점 RBI', '안타 1B single', '2타 2B double ', '3타 3B triple', '홈런 HR', '루타 Total Base', '타점 RBI run batted in', '도루 SB steal base', '도실 CB caught stealing', '볼넷 BB base on balls', '사구 HBP hit by pitch', '고4 IBB intentional base on balls', '삼진 SO strike out', '병살 GDP groud into double play', '희타 SH sacrifice hit', '희비 SF sacrifice fly', '타율 AVG', '출루(OPS)', '장타 SLG', 'OPS'=OBP(출루율)+SLG(슬러거)]

- $OBP = \frac{H + BB + HBP}{AB + BB + SF + HBP}$
- SLG=총루타(TB)
- OPS는 두 변수의 (OBP, SLG) 수학 함수이므로 제외
- 출장 100게임 이상, 2019년 선수들의 능력만 활용
- 경기 중 소속팀 옮긴 선수 제외 KN(기아->넥센), SK(SK->기아) : 최종 분석 대상 선수 수 = 87명
- 타율, 출루, 장타는 오브젝크로 읽어 숫자로 바꾸어 주었음
- 루타는 1루타+2루타+..+홈런 4루타 합이므로 제외하였음

타석에 많이 들어설수록 타점, 출루 가능성 드이 높아지므로 비율로 조사된 타율, 출루율, 장타율을 제외하고 다른 절대 척도는 타석으로 단위를 조정하였다.

```
import pandas as pd
df=pd.read_csv("/content/drive/My Drive/Data_storage/KBO/
bat1719.csv")
df0=df[(df['출장']>=100) &(df['연도']==19) &(df['팀']!='KN')
&(df['팀']!='SK')]
kbo_bat=pd.concat([df0.iloc[:,1],df0.iloc[:,10:29]], axis=1)
kbo_bat['타율'] = pd.to_numeric(kbo_bat['타율'],errors='coerce')
kbo_bat['출루'] = pd.to_numeric(kbo_bat['출루'],errors='coerce')
kbo_bat['장타'] = pd.to_numeric(kbo_bat['장타'],errors='coerce')
kbo_bat.drop(['루타'],axis=1,inplace=True)
kbo_bat.set_index('이름',inplace=True)
for k in range(0,15):
  kbo_bat.iloc[:,k]=kbo_bat.iloc[:,k]/np.array(df0['타석'])
kbo_bat.head(3)
```

이름	득점	안타	2타	3타	홈런	타점	도루	도실	볼넷	사구	고4	삼진	병살	희타	희비	타율	출루	장타
김하성	0.179200	0.265600	0.060800	0.0032	0.030400	0.166400	0.052800	0.006400	0.112000	0.011200	0.006400	0.128000	0.019200	0.0016	0.011200	0.307	0.389	0.491
양의지	0.132898	0.300654	0.056645	0.0000	0.043573	0.148148	0.008715	0.006536	0.104575	0.032680	0.008715	0.093682	0.028322	0.0000	0.013072	0.354	0.438	0.574
최정	0.141914	0.242574	0.044554	0.0000	0.047855	0.163366	0.004950	0.003300	0.113861	0.042904	0.009901	0.151815	0.006601	0.0000	0.013201	0.292	0.399	0.519

```
1 kbo_bat.shape
```

```
(87, 18)
```

=> 87명 선수, 18개 측정변수가 분석대상임

기초통계량 구하기

```
kbo_bat.describe()
```

	득점	안타	2타	3타	홈런	타점	도루	도실	볼넷
count	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000	87.000000
mean	0.123266	0.242431	0.041521	0.004464	0.018697	0.114314	0.018649	0.007828	0.086637
std	0.036660	0.034782	0.011718	0.005207	0.014523	0.035483	0.018002	0.007393	0.028175
min	0.057416	0.096774	0.000000	0.000000	0.000000	0.046875	0.000000	0.000000	0.016129
25%	0.104165	0.223930	0.035204	0.000000	0.006820	0.084957	0.004673	0.002529	0.070672
50%	0.120846	0.243564	0.042827	0.002247	0.016129	0.108696	0.013201	0.005587	0.087087
75%	0.139234	0.262041	0.049212	0.006251	0.028109	0.139699	0.031781	0.012030	0.104390
max	0.375000	0.309942	0.063622	0.019608	0.062030	0.192817	0.072089	0.046875	0.153153

상관계수 행렬

측정변수가 많은 경우 상관계수 행렬만으로 (개체) 선수들의 능력을 파악하는 것은 쉽지 않다. 그러기에 측정변수 (타자의 능력) 유사성을 고려하여 변수의 차원을 축약하여 개체를 판별하거나 분류하는 것이 적절하다.

- 상관계수의 절대값이 높은 변수들간에는 유사성이 높다. +는 양의 상관, -는 음의 상관
- 득점의 경우 많이 나가야 다음 타자의 공격으로 들어오므로 루타와 상관관계가 높다.
- 타점은 루타, 홈런과 상관관계가 높고 희생타(희생번트)와는 상관관계가 음으로 높다. 즉 희생번트가 적은 선수가 타점이 높다.

```
kbo_bat.corr(method='pearson')
```

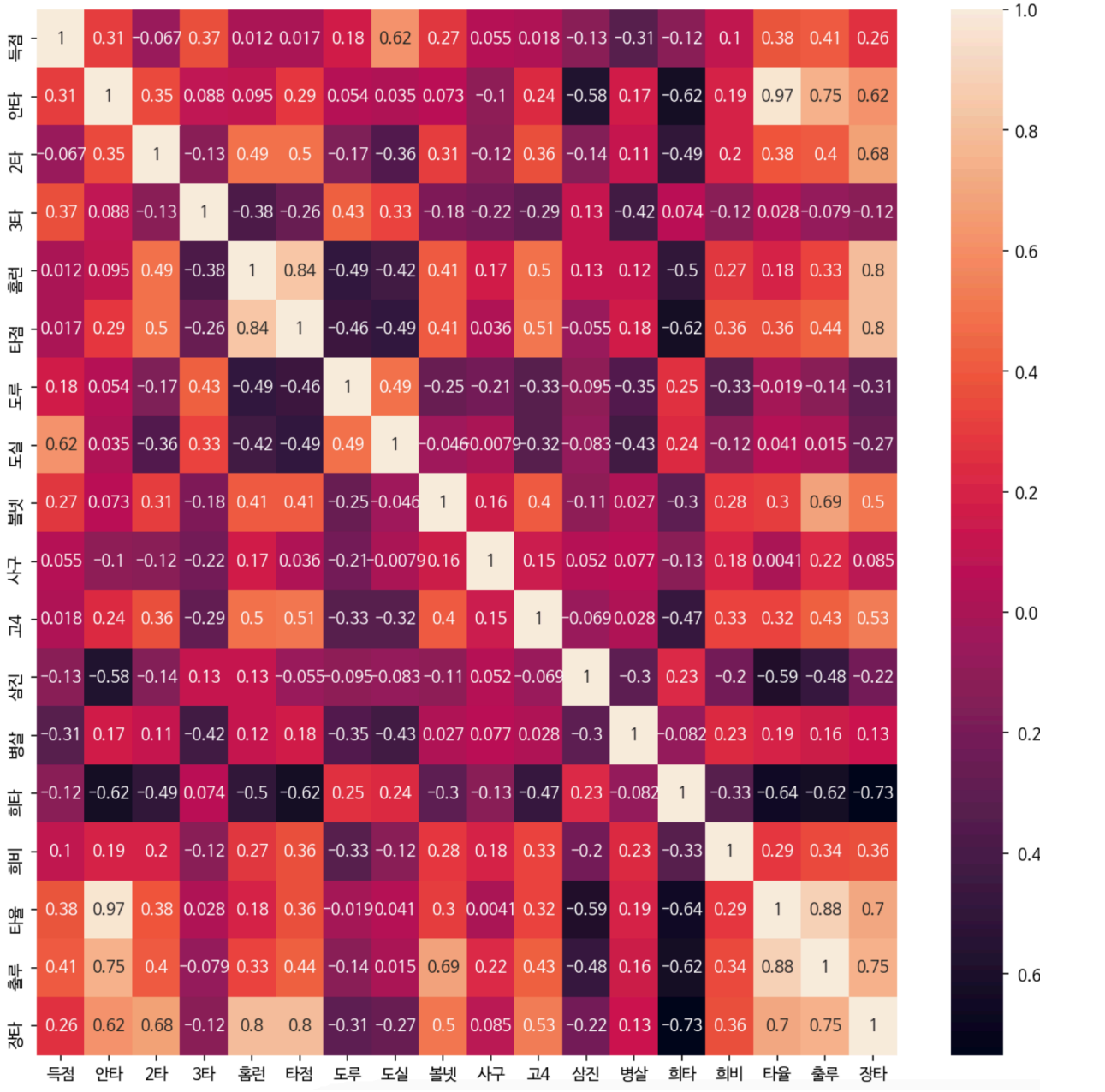
	득점	안타	2타	3타	홈런	타점	도루	도실
득점	1.000000	0.312906	-0.066731	0.366875	0.012479	0.016568	0.176983	0.620747
안타	0.312906	1.000000	0.346605	0.087687	0.095291	0.291660	0.053660	0.035024
2타	-0.066731	0.346605	1.000000	-0.132991	0.491014	0.498045	-0.169693	-0.357760
3타	0.366875	0.087687	-0.132991	1.000000	-0.381898	-0.261071	0.431648	0.333160
홈런	0.012479	0.095291	0.491014	-0.381898	1.000000	0.838829	-0.489593	-0.419439
타점	0.016568	0.291660	0.498045	-0.261071	0.838829	1.000000	-0.458363	-0.486562
도루	0.176983	0.053660	-0.169693	0.431648	-0.489593	-0.458363	1.000000	0.492612

상관계수 유의성 검정 : 귀무가설 : 득점과 희타의 상관관계는 존재하지 않는다. 서로 독립이다.

```
import scipy.stats as stats
stats.pearsonr(kbo_bat['득점'],kbo_bat['희타'])
```

득점과 희타의 상관계수는 - 음이고 유의확률이 0.26이므로 귀무가설이 채택되어 음의 상관관계는 있으나 유의하지 않다고 결론내릴 수 있다. `(-0.12186436439205267, 0.260830050711065)`

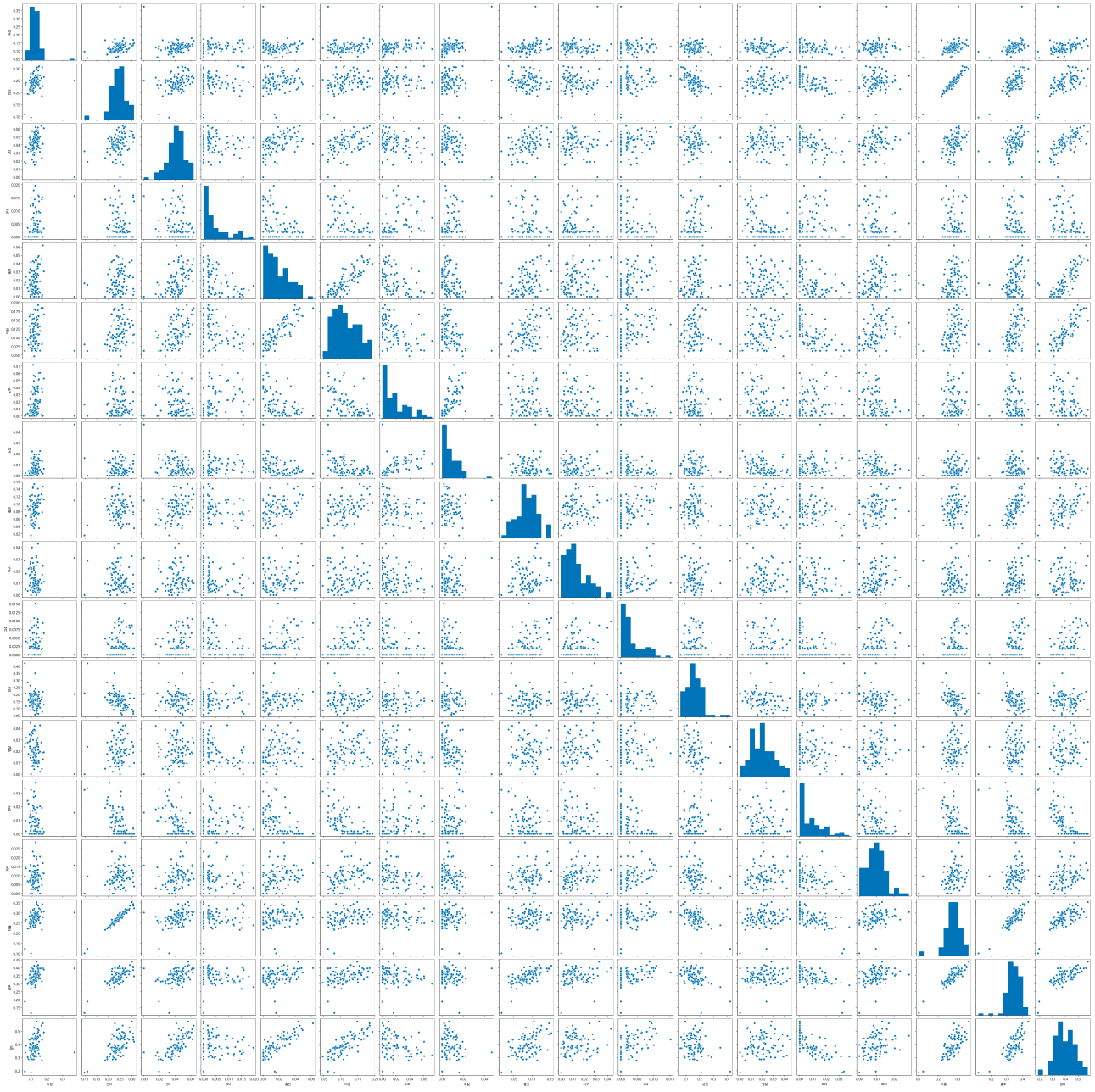
```
kbo_cor=kbo_bat.corr(method='pearson')
import seaborn as sn
plt.figure(figsize=(12, 12))
sn.heatmap(kbo_cor,annot=True)
plt.show()
```



산점도 행렬

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.pairplot(kbo_bat)
plt.show()
```

각 측정변수의 히스토그램을 보면 좌우 대칭이 아닌 분포를 보인 측정변수가 존재, 정규화 변환은 상관계수 값의 크기를 변화시키지 않으므로 **굳이 정규화 변환은 필요없다. 단 예측모형의 경우에는 필요하다.**



예제데이터 주성분 구하기

$$X_{n=87\times p=18} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \cdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

이름	득점	안타	2타	3타	홈런	타점	도루	도실	볼넷	사구	고4	삼진	병살	희타	희비	타율	출루	장타
김하성	0.179200	0.265600	0.060800	0.0032	0.030400	0.166400	0.052800	0.006400	0.112000	0.011200	0.006400	0.128000	0.019200	0.0016	0.011200	0.307	0.389	0.491
양의지	0.132898	0.300654	0.056645	0.0000	0.043573	0.148148	0.008715	0.006536	0.104575	0.032680	0.008715	0.093682	0.028322	0.0000	0.013072	0.354	0.438	0.574
최정	0.141914	0.242574	0.044554	0.0000	0.047855	0.163366	0.004950	0.003300	0.113861	0.042904	0.009901	0.151815	0.006601	0.0000	0.013201	0.292	0.399	0.519

데이터 크기의 중심화 centering = 평균 0 , 표준화 standardization 평균=0, 표준편차=1

$$X^*_{n=87\times p=18} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \cdots & & & \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{bmatrix} : c_{ij} = x_{ij} - \bar{x}_j, \bar{x}_j$$는 j번째 측정변수의 평균이다.

- with_std=False - 없으면 표준화이다. $s_{ij} = \dfrac{x_{ij} - \bar{x}_j}{s(x_j)}$

scale(with_mean=True,with_std=True) 함수에 의해 원 데이터가 표준화 되어 변환 결과가 X에 저장된다. 물론 배열 array 형태이다. X의 모든 열의 평균을 구하면 0, 표준편차를 구하면 1 이다. 분석대상 선수는 87명, 측정변수는 18개이다.

중심화가 맞나? 표준화가 맞나?
많은 논쟁이 있으나 파이썬 PCA() 모듈은 공분산으로 고유값을 구하므로 표준화 하는 것이 적절하다.
,with_mean=True,with_std=True 을 사용하지 않아도 된다.

```
from sklearn.preprocessing import scale
X=scale(kbo_bat,with_mean=True,with_std=True)
X.shape
```

```
(87, 18)
```

표준화된 데이터의 공분산 행렬을 이용하여 고유값, 놈2 1인 고유벡터를 구하여 이를 부하 Loading = 선형계수로 하여 원 변수의 선형 결합으로 주성분 변수가 구해진다. $\underline{y}$가 주성분 변수이고 L이 부하행렬이다.

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{pmatrix}_{p\times 1} = \begin{pmatrix} l_{11} & l_{12} & \cdots & l_{1p} \\ l_{21} & l_{22} & \cdots & l_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ l_{p1} & l_{p2} & \cdots & l_{pp} \end{pmatrix}_{p\times p} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}_{p\times 1} = L\underline{x}$$

80% 규칙에 의해 구하는 경우 : n_components=2을 사용하면 2개 주성분 가져온다.

```
from sklearn.decomposition import PCA
pca=PCA(0.8) #80% 만약 주성분 개수 지정 : n_components=2
kbo_pca=pca.fit_transform(X) #PC variables
kbo_pca.shape
```

```
(87, 6)
```

주성분 변수의 관측값은 kbo_pca(=pca.fit_transform(X))에 배열로 저장되고 행 차원은 원변수 차원과 동일하고 열 차원은 누적 변동 기여율이 80%를 만족하는 제 6주성분까지 저장되어 있다.

주성분 변수 누적 변동 설명 기여율 출력

- 앞에서 주성분 분석을 실시할 때 주성분 변수 개수 설정 옵션을 0.8 (80%)로 하여 누적 변동 설명 기여율이 80% 되는 주성분을 선택하였는데 주성분 6개로 80%를 넘었으므로 누적 변동 설명 기여율을 보면 80.18%로 6개로 80%를 넘는다.

```
import numpy as np
np.cumsum(pca.explained_variance_ratio_)
```

```
array([0.3516185 , 0.52731046, 0.62673944, 0.70833972, 0.7564187 ,
       0.80185609])
```

고유치

- 원변수의 개수만큼 주성분 변수가 계산되므로 고유치 개수도 원변수 개수와 같은 18이다.

```
from scipy import linalg as LA
R=np.cov(X,rowvar=False)    # calculate the covariance matrix
evals, evecs = LA.eig(R)
evals
```

```
array([6.40272759e+00+0.j, 3.19922795e+00+0.j, 1.81053237e+00+0.j,
       1.48588418e+00+0.j, 8.75484736e-01+0.j, 8.27383074e-01+0.j,
       7.07874538e-01+0.j, 6.27297934e-01+0.j, 5.67508996e-01+0.j,
```

주성분개수 구하기

앞에서 설명하였듯이 주성분변수는 원변수의 공분산행렬(혹은 상관행렬)을 이용하여 고유값을 구하고, 그에 대응하는 고유벡터를 선형계수로 하여 주성분변수를 구한다.

주성분변수는 서로 독립이며, 제1주성분이 원변수의 변동을 가장 많이 설명하고, 제2주성분, 제3주성분, .. 순이다. 그리고 주성분의 개수는 원변수의 개수만큼 존재한다.

(주성분변수의 주목적) 변수의 차원 축소, 즉 변수의 개수를 줄이는 것이 주된 목적임-어떻게 줄일까?

(Rule of Thumb) 80%

제1주성분변수의 원변수 변동을 설명하는 능력이 가장 크고, 제2주성분, 제3주성분, ... 순이다.

변동설명 기여율

[공분산행렬] k-번째 주성분변수의 변동설명 기여율(variance explianed ratio) = $\frac{\lambda_k}{\sum_i^p \lambda_i}$

[상관행렬] k-번째 주성분변수의 변동설명 기여율 = $\frac{\lambda_k}{p}$

상관행렬의 경우 대각원소가 1이므로 고유치의 합은 원변수의 개수와 동일하다.

Cochran Rule

원변수 변동설명 누적 기여율이 80%까지 주성분변수를 선택한다. 즉 20% 정보는 희생된다.

원변수의 상관관계가 높을수록 변수의 차원은 축약이 쉽게되므로 80% 규칙에 의한 주성분변수 개수는 작음(일반적으로 2~3개)

고유값 1이상

상관행렬을 이용하는 경우 원변수 변동의 합은 p이므로 평균인 1이상 고유값을 갖는 주성분변수만 선택한다. 일반적으로 고유값 1 이상이 주성분을 선택하면 누적 변동설명 기여율 80%와 일치한다.

차원 축약이 목적이므로 2개만 선택하였다.

```
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
kbo_pca=pca.fit_transform(X) #PC variables
print(kbo_pca.shape,"\n",kbo_pca[0:2,:])
```

```
(87, 2)
 [[-2.95633279  1.36481791]
 [-5.29378805  0.44833048]]
```

<= 첫번째 선수의 제 1주성분 값은 -2.956, 제2 주성분 값은 1.364, 두번째 선수의 제1 주성분 값은 -5.293, 제2 주성분 값은 0.448이다.

2개만 선택해도 각 주성분의 변동 기여율은 바뀌지 않는다. 제 1주성분 원변동 설명 기여율은 35.2%, 제2 주성분까지 누적은 52.7%로 6개 선택했을 때와 동일하다.

```
array([0.3516185 , 0.52731046])
```

주성분변수 이름부여

부하 loading

주성분변수의 선형계수행렬을 부하행렬이라 한다.

$$y_j = l_j' * \underline{x} = (l_{j1}\ l_{j2} \cdots\ l_{jp})_{1\times p} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix}_{p\times 1}$$

원변수를 표준화(단위 없음, 최소 중심화) 했으므로 선형계수(부하값) 크기에 의해 주성분 변수의 값이 결정된다. 부하 값이 큰 원변수 값이 주성분 변수 크기에 큰 영향을 미치므로 이를 이용하여 주성분 변수 이름을 부여하면 된다. 간단하게 말했지만 결코 쉬운 일이 아니다.

주성분 부하값은 pca.components_ 에 배열로 저장되어 있는데 사용하려면 전치 transpose를 해야 행의 차원이 변수의 개수와 동일해진다. 배열로 저장되어 있으므로 이를 활용하기 위하여 pd.DataFrame() 으로 데이터프레임으로 바꾸고 일단 이름을 "PC1", "PC2"로 바꾸었다.

```
kbo_cor=df.corr(method='pearson')
kbo_loading=pd.DataFrame(pca.components_.T)
kbo_loading.set_index(df_cor.index,inplace=True,,columns=['PC1','PC2'])
kbo_loading #loadings L matrix
```

주성분 변수(주성분 점수 score)의 이름을 PC1, PC2로 설정하였다.

	0	1
득점	-0.069775	0.386898
안타	-0.258766	0.326263
2타	-0.253776	-0.057208
3타	0.111769	0.314622
홈런	-0.280051	-0.248772

부하 활용 주성분 이름 부여

위와같이 주성분변수는 원변수들의 선형결합에 의해 구해지므로 적절한 이름을 붙여주어야 활용도가 높아진다. 주성분분석의 핵심적 단계이며, 분석자의 능력(다른 단계는 소프트웨어가 계산)이 필요하다.

명확하게 이해될 수 있는 이름 부여가 중요하다.

무엇을 이용하여 이름을 붙일까? 선형계수=부하 값의 크기를 이용한다.

부하 값이 크다는 것은 주성분변수가 계산될 때 그 변수의 영향이 크다는 것을 의미한다. 즉, 부하 값이 상대적으로 큰 (절대값 기준보다는) 변수들에 의해 주성분변수의 이름이 부여된다.

상관관계가 높은 변수들은 동일 주성분 내에서 부하값이 크게나타난다.

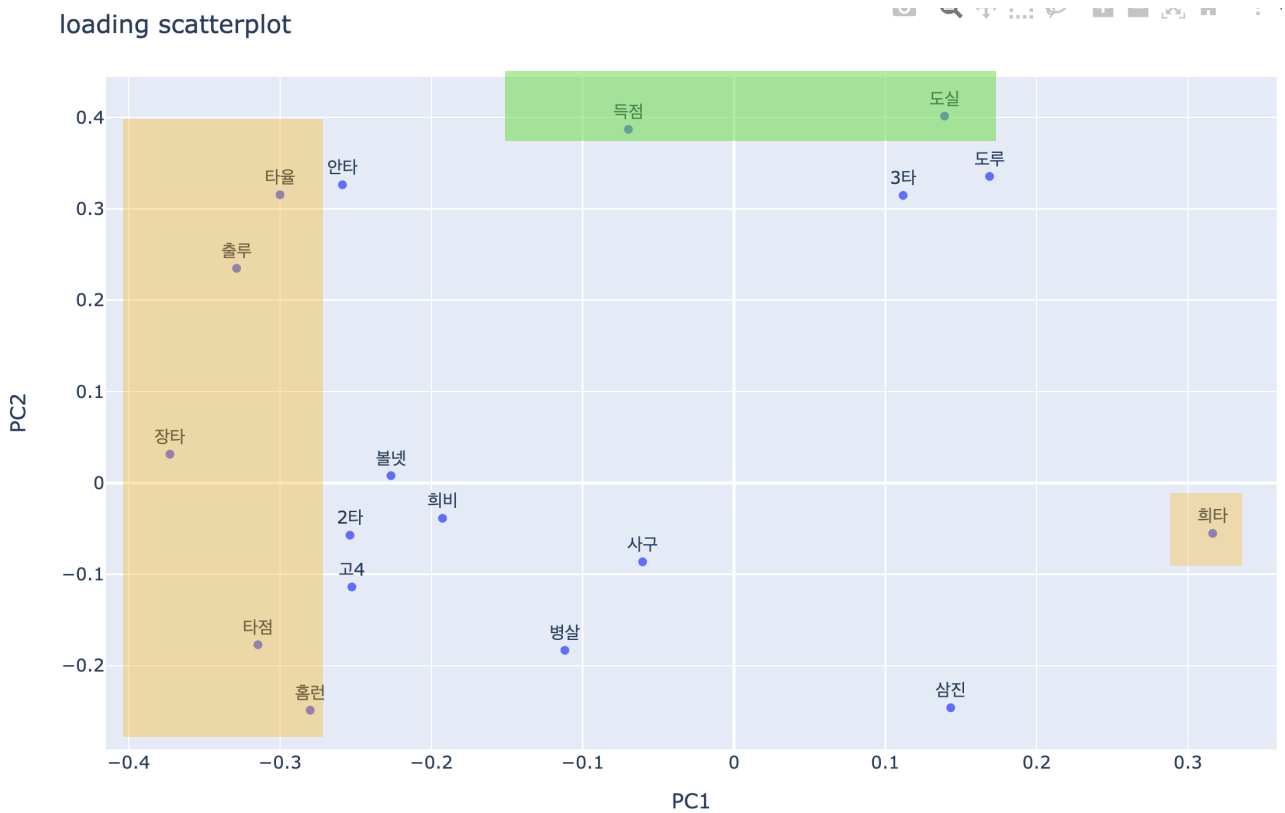
부하 값이 음인 경우는 다른 변수들과 음의 상관관계를 의미한다. 그러나 상관관계가 높으므로 변수의 유사성은 존재한다는 것을 의미한다.

부하에 대한 이름 설정의 편의성을 부하값을 산점도로 시각화 하였다.

```
import plotly.express as px
import matplotlib.pyplot as plt
fig=px.scatter(kbo_loading,x="PC1",y="PC2",text=kbo_loading.index)
fig.update_traces(textposition='top center')
fig.update_layout(
  height=600,
  title_text='loading scatterplot')
fig.show()
```

부하값의 크기 기준은 절대값이 0.5 이상(그보다 작은 경우에는 상대적으로 큰 부하값의 원변수) 원변수를 이용하여 주성분 변수이름을 부여하면 된다. 앞에서 언급하였듯이 부하값에 -는 음의 영향, 즉 주성분 변수 값이 음수가 되도록 영향을 미치는 것이므로 + 부하와는 반대로 움직인다. 부하값이 반대인 두 변수는 상관계수의 부호가 -이다.

- 제1 주성분 : 부하 + 큰 변수 = 희타, 부하 - 큰 변수=장타, 출루, 타점, 타율, 홈런 => **장타(-)**
- 제2 주성분 : 부하 + 큰 변수 = 득점, 도루실수, 3루타, 도루, 안타 타율, -큰 변수 = 홈런, 삼진 => **출루율(+)**



구글 코렙 그래프 한글 출력

코렙 첫 행에 다음 코드를 실행한다.

```
import matplotlib as mpl
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
!apt -qq -y install fonts-nanum
import matplotlib.font_manager as fm
fontpath = '/usr/share/fonts/truetype/nanum/NanumBarunGothic.ttf'
font = fm.FontProperties(fname=fontpath, size=9)
plt.rc('font', family='NanumBarunGothic')
mpl.font_manager._rebuild()
```

```
The following NEW packages will be installed:
  fonts-nanum
0 upgraded, 1 newly installed, 0 to remove and 11 not upgraded.
Need to get 9,604 kB of archives.
After this operation, 29.5 MB of additional disk space will be used.
Selecting previously unselected package fonts-nanum.
(Reading database ... 144676 files and directories currently installed.)
Preparing to unpack .../fonts-nanum_20170925-1_all.deb ...
Unpacking fonts-nanum (20170925-1) ...
Setting up fonts-nanum (20170925-1) ...
Processing triggers for fontconfig (2.12.6-0ubuntu2) ...
```

그런 후 Runtime 메뉴=> Restart runtime 선택하여 실행 한 후 다시 한 번 위의 코드를 재실행 한다.

그러면 위의 결과가 다음 결과로 바뀐다. 그러면 향후 그래프에서 한글 깨짐이 방지될 것이다. 불편한 점은 재접속 시 매번 실행해야 한다는 것이다.

```
fonts-nanum is already the newest version (20170925-1).
0 upgraded, 0 newly installed, 0 to remove and 11 not upgraded.
```

주성분분석 활용 [1]

- 변수의 차원을 축약하여 개체에 대한 정보를 얻기 쉽게 한다. 변수의 차원이 저차원(일반적으로 3개 정도)으로 축약되므로 산점도를 그려 개체의 특성을 쉽게 파악할 수 있다.
- 주성분변수가 상관계수가 0(독립)이라는 사실을 이용하여 회귀분석의 다중공선성 문제 해결을 위한 방법으로 사용한다.
- 주성분분석은 군집분석, 판별분석을 1차분석으로 활용된다. 개체의 유사성에 의해 분류된 군집의 특성을 분석하여 이름을 부여하는데 활용하거나 판별분석 결과 오분류 개체들의 특성 파악에 도움을 준다. - 데이터 변수가 다수(5개 이상)인 군집분석과 판별분석에는 항상 주성분분석이 필요하다.

제1주성분은 장타 능력이나 음의 값이 클수록 장타 능력이 큰 선수이다.

주성분 변수는 PCA() 함수에 의해 분석되고 결과는 "kbo_pca=pca.fit_transform(X) #PC variables" 문장에 의해 주성분 변수는 kbo_pca에 배열로 저장되어 있다.

제 1주성분은 음의 값이 큰 선수가 장타력이 있는 선수이므로 양의 값으로 바꾸기 위하여 작업한 것이다. 동일 이름을 사용했으므로 반드시 한 번만 실행하기 바란다. 재실행 하면 다시 음의 값을 변경된다.

```
kbo_pca[:,0]=-kbo_pca[:,0] # 재실행 하면 -가 재실행되므로 조심
```

배열인 kbo_pca을 데이터프레임으로 만들고 이름을 "장타능력", "출루능력"으로 설정하였다. 원 이름은 0, 1이다.

```
kbo_pca=pd.DataFrame(kbo_pca,columns=['장타능력','출루능력'])
kbo_pca.set_index(df.index,inplace=True)
kbo_pca.head(3)
```

이름	장타능력	출루능력
김하성	2.956333	1.364818
양의지	5.293788	0.448330
최정	3.746751	-1.059637
샌즈	3.974014	-0.503673
로하스	3.262150	-0.020180

<= 페이지 16의 kbo_pca 배열을 보면 제1 주성분 분석 값에 -가 붙어 있는 것을 알 수 있다. 첫번째 선수 김하성의 원 제1 주성분 값은 -2.956, 양의지는 -5.293 이었다.

```
import plotly.express as px
fig = px.scatter(kbo_pca,x="장타능력",y="출루능력",text=kbo_pca.index)
fig.update_traces(textposition='top center')
fig.update_layout(height=800, title_text='2019_KBO 타자')
fig.show()
```

장타능력 우수 선수 : 양의지, 박병호, 라프, 센즈, 최형우, 최정

출루능력 우수 선수 : 송민섭, 박민우, 고종욱, 이정후

주성분 능력 선수 보기

원데이터 KBO_bat와 주성분 변수(점수)를 합치고 kbo_fin에 저장하였다.

pd.nlargest() 함수에 의해 설정된 변수가 가장 큰 값을 가진 선수를 출력하였다. 향후 사용하려면 새로운 데이터프레임으로 저장하면 된다.

장타 최우수 선수는 양의지(5+) > 박병호 > 러프(4+) > 최형우 > 샌즈(3+) 순이다.

```
kbo_fin=pd.concat([kbo_bat,kbo_pca],axis=1)
kbo_fin.nlargest(5,"장타능력")
```

이름	득점	안타	2타	3타	홈런	타점	도루	도실	볼넷	사구	고4	삼진	병살	희타	희비	타율	출루	장타	장타능력	출루능력
양의지	0.132898	0.300654	0.056645	0.000000	0.043573	0.148148	0.008715	0.006536	0.104575	0.032680	0.008715	0.093682	0.028322	0.0	0.013072	0.354	0.438	0.574	5.293788	0.448330
박병호	0.172932	0.227444	0.041353	0.000000	0.062030	0.184211	0.000000	0.001880	0.146617	0.024436	0.009398	0.219925	0.013158	0.0	0.016917	0.280	0.399	0.560	4.410991	-1.696727
러프	0.140845	0.242958	0.061620	0.003521	0.038732	0.177817	0.010563	0.005282	0.140845	0.012324	0.010563	0.153169	0.031690	0.0	0.015845	0.292	0.396	0.515	4.301036	-0.866531
최형우	0.117117	0.246847	0.055856	0.001802	0.030631	0.154955	0.000000	0.001802	0.153153	0.012613	0.010811	0.138739	0.023423	0.0	0.012613	0.300	0.413	0.485	3.987318	-0.925977
샌즈	0.163132	0.261011	0.063622	0.001631	0.045677	0.184339	0.001631	0.003263	0.125612	0.009788	0.001631	0.164763	0.027732	0.0	0.008157	0.305	0.396	0.543	3.974014	-0.503673

출루율 최우수 선수는 양의지(5+) > 박병호 > 러프(4+) > 최형우 > 샌즈(3+) 순이다.

```
kbo_fin.nlargest(5,"출루능력")
```

이름	득점	안타	2타	3타	홈런	타점	도루	도실	볼넷	사구	고4	삼진	병살	희타	희비	타율	출루	장타	장타능력	출루능력
송민섭	0.375000	0.250000	0.000000	0.015625	0.000000	0.062500	0.000000	0.046875	0.109375	0.031250	0.000000	0.203125	0.000000	0.015625	0.015625	0.302	0.397	0.340	-1.970922	6.492620
박민우	0.169202	0.306084	0.043726	0.015209	0.001901	0.085551	0.034221	0.013308	0.077947	0.017110	0.001901	0.076046	0.019011	0.003802	0.011407	0.344	0.403	0.434	0.976894	4.026482
고종욱	0.148148	0.309942	0.048733	0.013645	0.005848	0.109162	0.060429	0.019493	0.035088	0.001949	0.001949	0.173489	0.009747	0.000000	0.003899	0.323	0.347	0.421	-0.516524	3.852296
이정후	0.144444	0.306349	0.049206	0.015873	0.009524	0.107937	0.020635	0.011111	0.071429	0.006349	0.000000	0.063492	0.023810	0.004762	0.006349	0.336	0.386	0.456	1.087281	3.175203
이천웅	0.143556	0.274062	0.039152	0.004894	0.003263	0.078303	0.034258	0.016313	0.092985	0.008157	0.001631	0.133768	0.021207	0.006525	0.001631	0.308	0.378	0.374	-0.553173	2.441856

주성분분석 활용 [2] anomaly

주성분 분석은 원 변수들의 선형함수를 이용하여 그 변수들의 공분산 구조를 설명하는 방법이다. 개체 수가 n, 측정변수가 p개인 데이터 행렬 로부터의 공분산 행렬을 $S_{p\times p}$라 하자. 공분산 행렬로부터 구한 고유치를 λ_i라 하고 이에 대응하는 고유벡터를 $\underline{e}_i$라 하자. 고유벡터를 선형계수(부하)로 원변수의 선형결합을 주성분 변수라 한다.

주성분 분석을 이용하여 개체의 이상 진단하는 방법은 주요 주성분들을 이용하여 원 변수들의 공통 개념 하에서의 이상 개치를 진단하거나 주요 주성분을 제외한 잔차 주성분을 이용하는 방법으로 나뉜다. 잔차 주성분 접근 방법은 데이터의 전체 변동에서 주요 공통 개념 변동을 제외한 잔차 변동에서 특이 값을 갖는 개체를 발견하게 된다 (Johnson과 Wichern, 2007).

Cochran 규칙에 의해 구한 주성분 개수가 k라 하자. 그리고 k-번째 주성분 변수를 y_k라 하자. 주성분 변수는 정규분포를 따르므로 표준정규분포의 제곱은 카이분포를 따른다. 주요 주성분에 의한 이상치 진단은 다음 통계량을 이용한다.

$$\frac{y_1^2}{\lambda_1}+\frac{y_2^2}{\lambda_2}+\ldots+\frac{y_k^2}{\lambda_k}\sim\chi^2(df=k)$$

잔차 주성분에 의한 이상치 발견 통계량을 다음과 같다.

$$\frac{y_{k+1}^2}{\lambda_{k+1}}+\frac{y_{k+2}^2}{\lambda_{k+2}}+\ldots+\frac{y_p^2}{\lambda_p}\sim\chi^2(df=p-k)$$

주성분 구하기

```
df=kbo_bat
from sklearn.preprocessing import scale
X=scale(kbo_bat,with_mean=True, with_std=True)
from sklearn.decomposition import PCA
pca=PCA(0.8) #80% 만약 주성분 개수 지정 : n_components=2
df_pca=pca.fit_transform(X) #PC variables
df_pca=pd.DataFrame(df_pca,columns=['장타능력','출루능
력','PC3','PC4','PC5','PC6'])
df_pca.set_index(df.index,inplace=True)
df_pca.head(3)
```

이름	장타능력	출루능력	PC3	PC4	PC5	PC6
김하성	-2.956333	1.364818	0.832664	-0.911198	1.000020	0.110015
양의지	-5.293788	0.448330	-0.326589	1.130861	-0.202799	1.426986
최정	-3.746751	-1.059637	2.314469	1.751678	-0.652929	1.881174

고유치 구하기

np.cov 함수에 의해 구한 고유치와 고유벡터는 배열형식, 복합소수로 출력되므로 .real은 실수 부분만 출력하라는 의미이다.

```
from scipy import linalg as LA
R=np.cov(X,rowvar=False)    # calculate the covariance matrix
evals, evecs = LA.eig(R)
evals[0].real
```

```
6.402727585879 86
```

이상치 진단 통계량 계산 및 활용

```
df_pca['anomaly_ts']=df_pca['장타능력']**2/evals[0].real+df_pca['출루능력']**2/evals[1].real
df_pca.head(3)
```

이름	장타능력	출루능력	PC3	PC4	PC5	PC6	anomaly_ts
김하성	-2.956333	1.364818	0.832664	-0.911198	1.000020	0.110015	1.947271
양의지	-5.293788	0.448330	-0.326589	1.130861	-0.202799	1.426986	4.439742
최정	-3.746751	-1.059637	2.314469	1.751678	-0.652929	1.881174	2.543495

```
import scipy.stats as st
st.chi2.ppf(0.95,2)
```

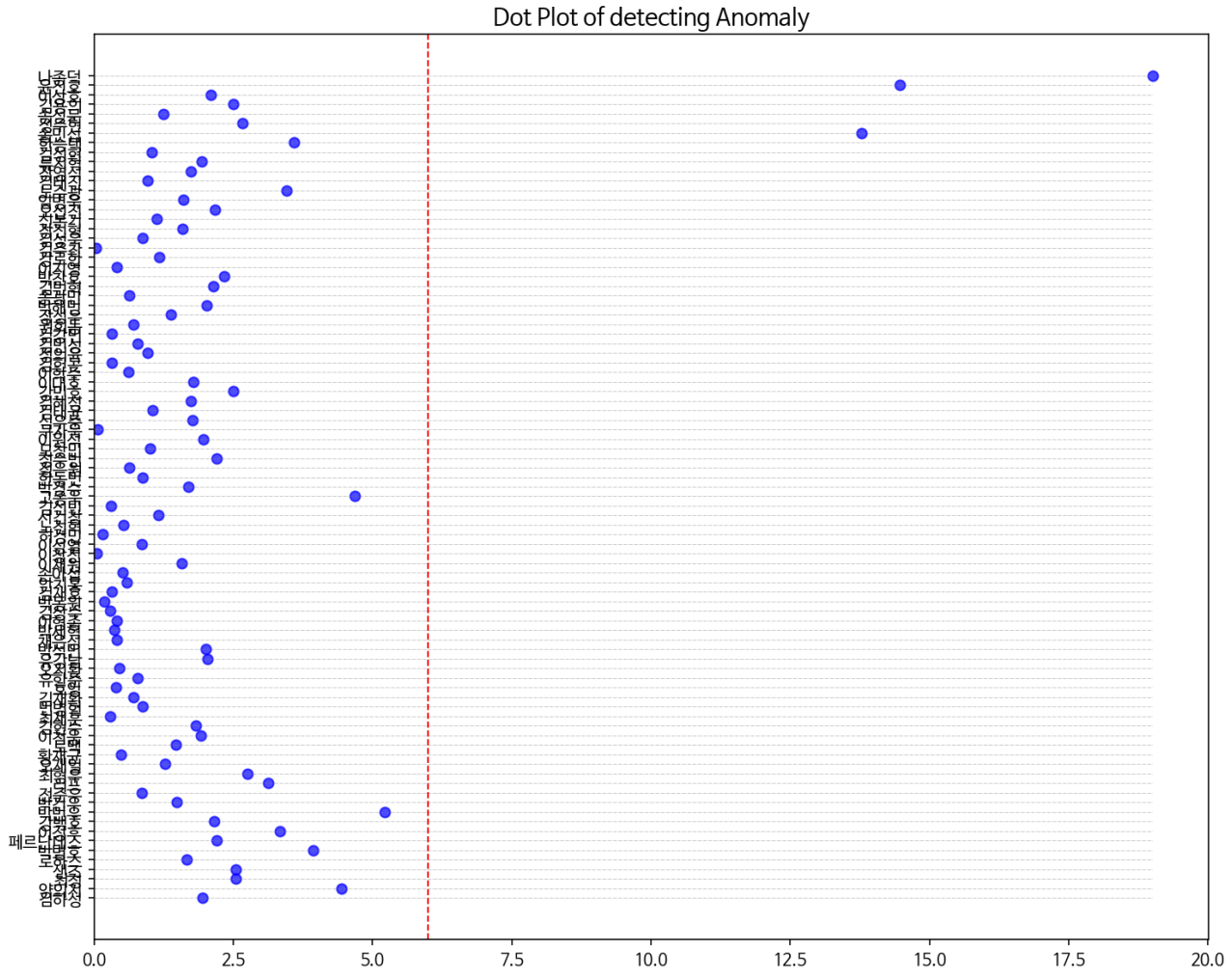
```
5.991464547107979
```

```
df_pca.loc[df_pca['anomaly_ts']>st.chi2.ppf(0.95,2),:]
```

송민섭 선수는 출루능력은 출중하여 이상치가 되었고 윤진호는 장타능력이 너무 낮고 나종덕은 장타능력, 출루능력 모두 낮아 이상 선수가 되었음.

이름	장타능력	출루능력	PC3	PC4	PC5	PC6	anomaly_ts
송민섭	-1.970922	6.492620	4.677133	5.222367	-1.804366	-1.538810	13.783038
윤진호	-8.120222	-3.650372	1.651874	-1.156510	0.852922	-0.491044	14.463560
나종덕	-6.960899	-6.049800	1.587633	1.465201	-1.395707	-0.025407	19.008017

```
import scipy.stats as st
import matplotlib.pyplot as plt
df=df_pca
fig, ax = plt.subplots(figsize=(12,10))
ax.hlines(y=df.index,xmin=0,xmax=df['anomaly_ts'].max(),
color='gray',alpha=0.7,linewidth=0.3,linestyles='dashdot')
plt.axvline(x=st.chi2.ppf(0.95,2),linewidth=1, linestyle ="--",
color ='red')
ax.scatter(y=df.index,x=df['anomaly_ts'], color='blue',alpha=0.7)
ax.set_title('Dot Plot of detecting Anomaly',
fontdict={'size':14})
ax.set_xlim(0,df['anomaly_ts'].max()+1)
plt.show()
```



SVD singular value decomposition 고유값 분해

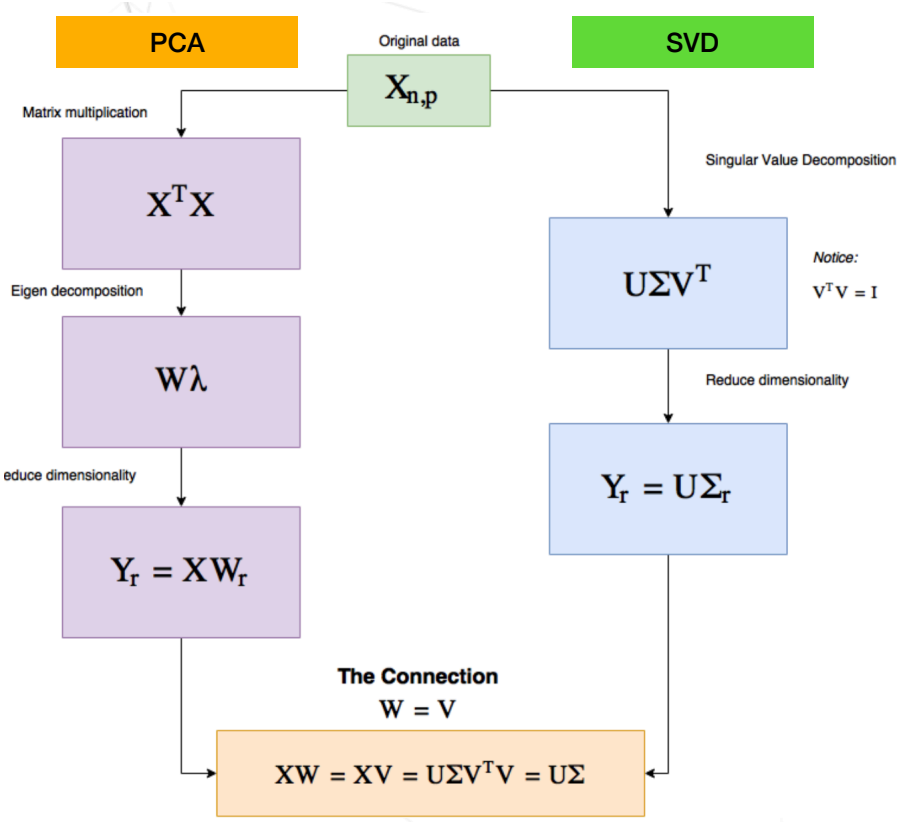
PCA와 SVD 비교

PCA는 변수의 수를 줄이지만 SVD는 변수는 물론 개체를 수까지 줄인다. SVD는 행(개체)의 수를 줄이므로 개체가 줄어들므로 ML(이미지 분류)에 주로 사용된다.

CA 차원축소는 원데이터의 공분산행렬(공분산구조)을 이용하지만 SVD 방법은 원데이터를 직교행렬, 특이값 대각원소로 분해하는 방법

축약된 주성분변수는 부하 크기에 의해 변수의 속성을 파악할 수 있으나 SVD는 가능하지 못함.

[https://tomaxent.com/2018/01/17/PCA-With-Tensorflow]



PCA

원데이터		주성분 행렬		차원축소	
$X_{n\times p}$	$=$	$Y_{n\times p} = L_{n\times p}X'$	분해	$X^*_{n\times k}$	$X^R_{n\times(p-k)}$
(단위 표준화)		$L_{n\times p}$		주요주성분 (80% 규칙)	잔차주성분 (20%)

SVD

원데이터		행렬분해		차원축소	
$X_{n\times p}$	$=$	$U_{n\times n}\Sigma_{n\times p}V_{p\times p}$	분해	$X^{*}_{m\times k}$ $X^{R2}_{(n-m)\times k}$	$X^{R}_{n\times(p-k)}$
(단위 표준화)		U: 특이값 직교행렬 Σ : 대각행렬 V : 특이값 직교행렬		행의 차수 $m < n$ 열의 차수 $k < p$	Thin SVD Compact SVD Truncate SVD

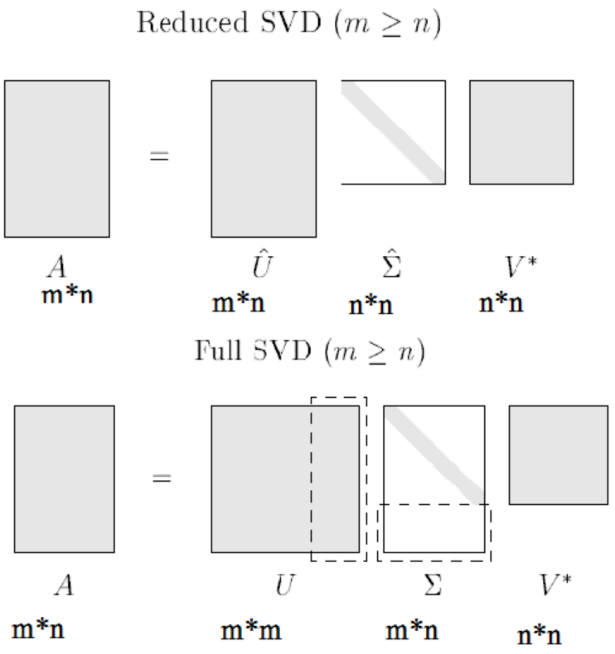
(1) $\Sigma_{n\times p}$: $X'X_{p\times p}$의 고유값 $\lambda_1, \lambda_2, \ldots, \lambda_p$ (변수 차수만큼 고유값이 존재)의 제곱근을 대각원소로 하는 비대칭 대각행렬임

(2) $U_{n\times n}$: 열벡터는 XX'의 고유벡터, $U'U = I_n$

(3) $V_{p\times p}$: 열벡터는 $X'X$의 고유벡터, $V'V = I_p$

Full, Reduced(truncate, thin) SVD

Reduced는 PCA와 동일하다.



국내 프로야구 타자 사례(계속)

Full SVD

만약 원 데이터의 단위가 유사하다면 with_mean=False, with_std=False 옵션을 사용하여 표준화 하지 않은 원 데이터를 그대로 사용하면 된다. 이미지 분류를 위하여 차원 축소가 필요하다면 표준화 필요 없다.

```
from sklearn.preprocessing import scale
X=scale(kbo_bat,with_mean=True, with_std=True)
X.shape
```

원데이터와 차수가 동일하다.

```
from numpy.linalg import svd
import numpy as np
U, s, VT = svd(X)
d=1.0/ s
D=np.zeros(X.shape)
D[:X.shape[1], :X.shape[1]] =np.diag(d)
B = VT.T.dot(D.T).dot(U.T)
B.T.shape  #축약된 데이터
```

```
(87, 18)
```

```
array([ 0.04904212,  0.93909713,  0.08296028,  0.03536963,  0.22033683,
        0.03639836,  0.05405582, -0.03428184,  0.74837278,  0.25733294,
        0.00409646,  0.00200638,  0.00748098,  0.08777844,  0.00860518,
        0.12327168, -1.16226453, -0.38238198])
```

Truncate SVD

차수를 6개로 줄이는 경우 주성분 결과와 동일하다.

```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=6)
Y=svd.fit_transform(X)
Y.shape
```

```
(87, 6)
```

```
array([-2.95633279,  1.36481791,  0.83266389, -0.91119816,  1.00002048,
        0.1100148 ])
```