강원지역혁신플랫폼

기계학습

Machine Learning

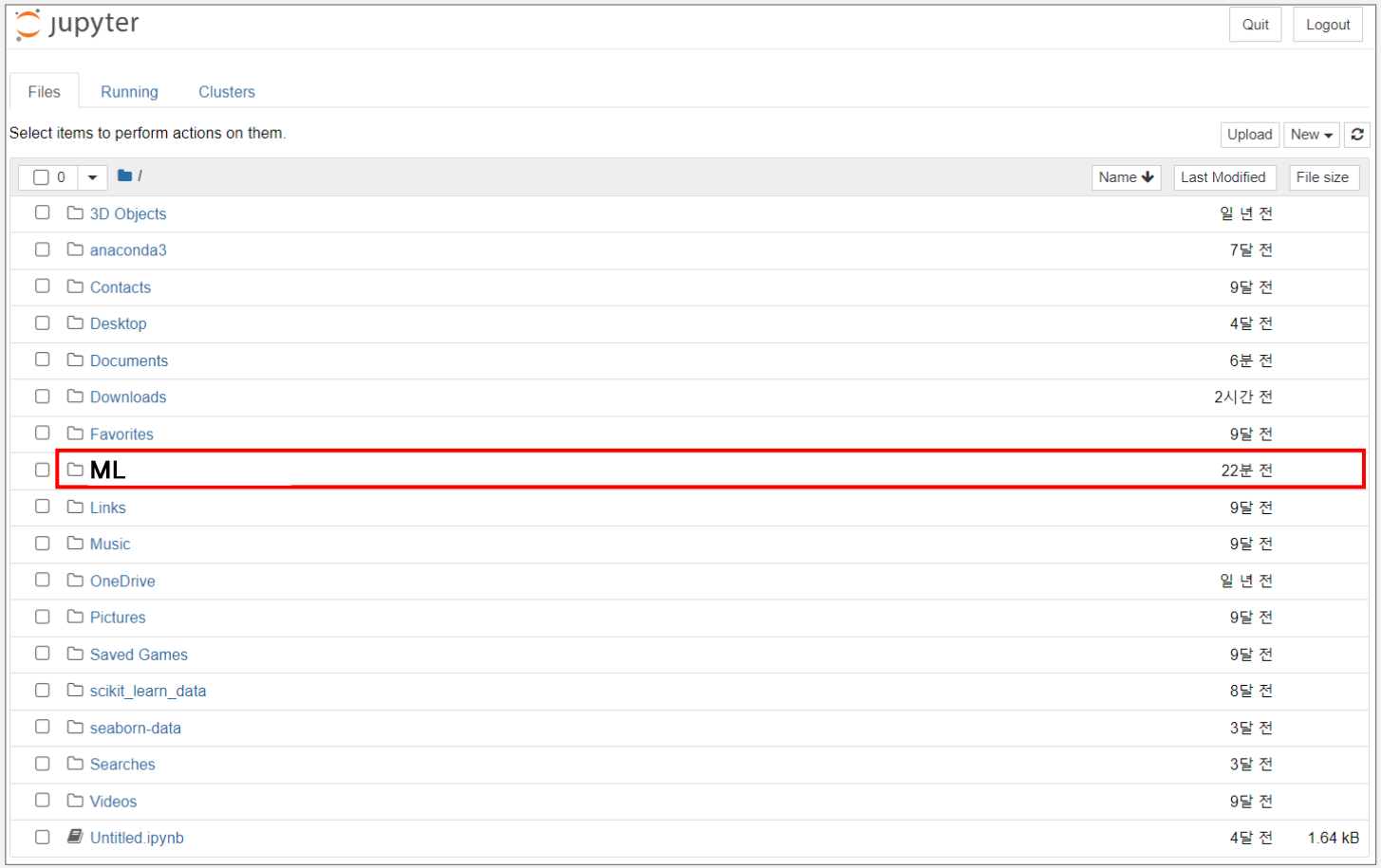
차원 축소를 위한 접근 방법

학습목표

- 차원 축소를 위한 접근 방법을 이해하고 구현할 수 있습니다.

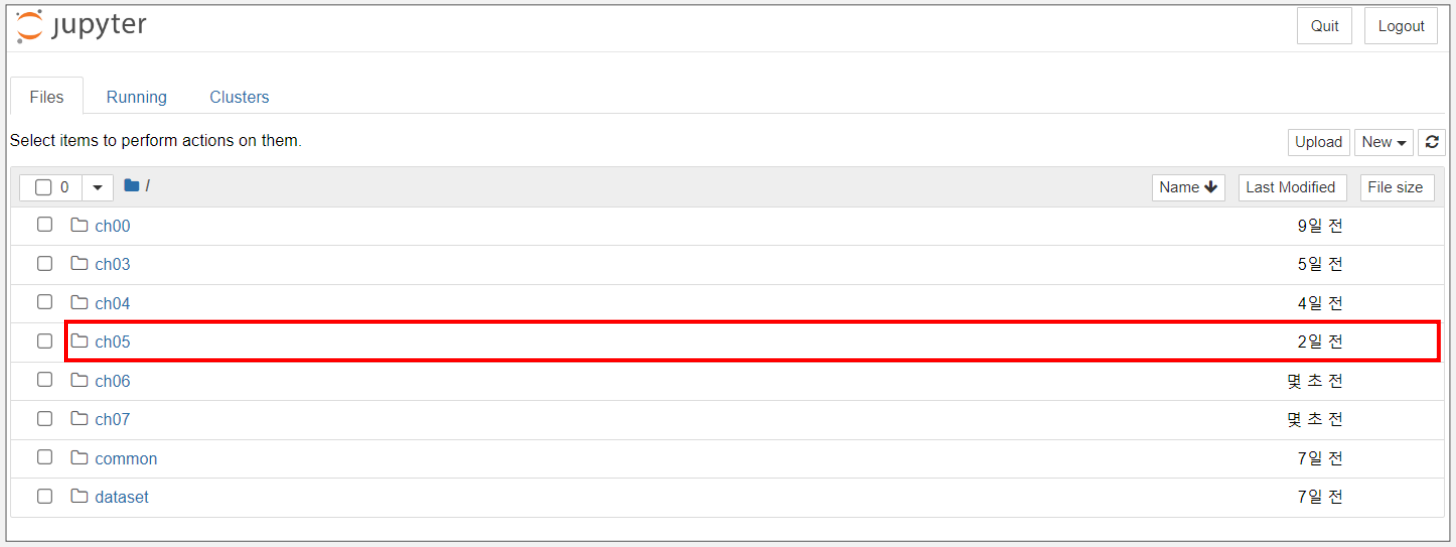
01 | ML 폴더

ML 폴더를 클릭하기



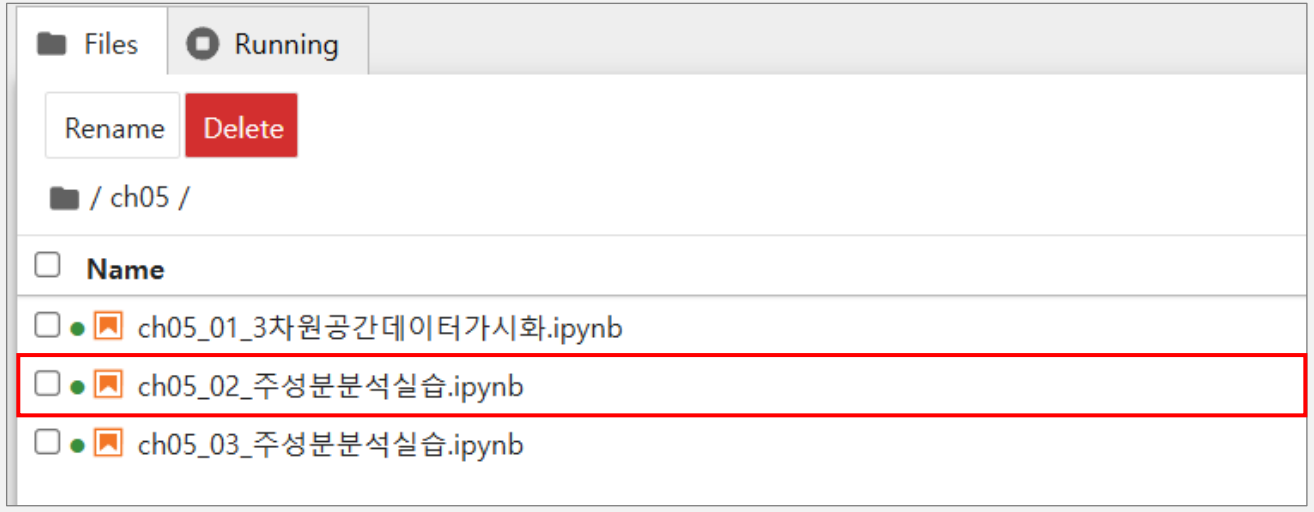
02 | ch05 폴더

ch05 폴더 클릭하기



03 | ch05_02_주성분분석실습.ipynb

✦ ch05_02_주성분분석실습.ipynb 파일 클릭하기



04 | 차원 축소를 위한 접근 방법: 특징 선택

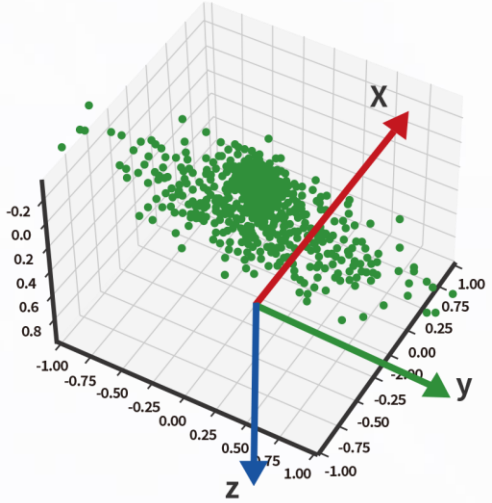
특징 선택(feature selection)

- 여러 개의 데이터를 비교할 때 일부 차원의 값은 크게 변동이 있지만, 일부 차원의 특징들은 거의 변화가 없을 수 있음
  - 이렇게 거의 변화가 없는 차원의 특징은 데이터들을 구분하는 데에 크게 중요하지 않은 특징임
    - 관련성이 낮고 중복되거나 불필요한 정보를 담은 차원을 버리고 중요한 차원만 선택하는 것이 가장 단순한 차원 축소임
      - 이런 방식을 특징 선택이라고 부름

05 | 차원 축소를 위한 접근 방법: 투영

투영(projection)

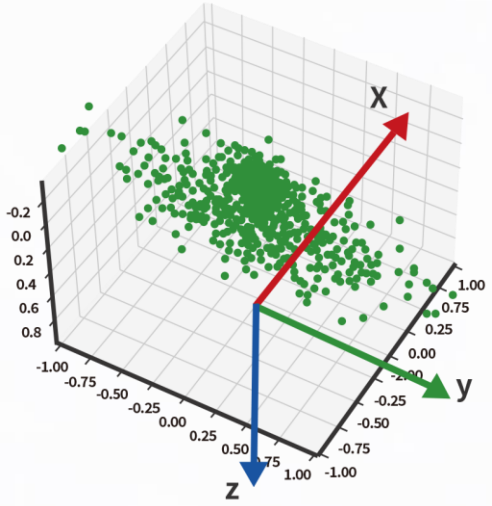
- 아래 그림과 같이 x, y, z 세 개의 축을 가진 3차원 공간에 점들로 표현된 데이터들이 존재한다고 가정해 보자.
  - 이 데이터들이 각 축에 대해서 가지는 값을 비교해 보면 아래 그림의 오른쪽과 같음
    - x축 방향으로는 데이터의 분산이 적음
    - y축 방향으로 가장 큰 분산을 보이는 것을 알 수 있음
    - z축은 y축보다는 적지만, x축 보다는 훨씬 큰 분산을 보이고 있음



출처: 강영민 외, 2021, 머신러닝

05 | 차원 축소를 위한 접근 방법: 투영

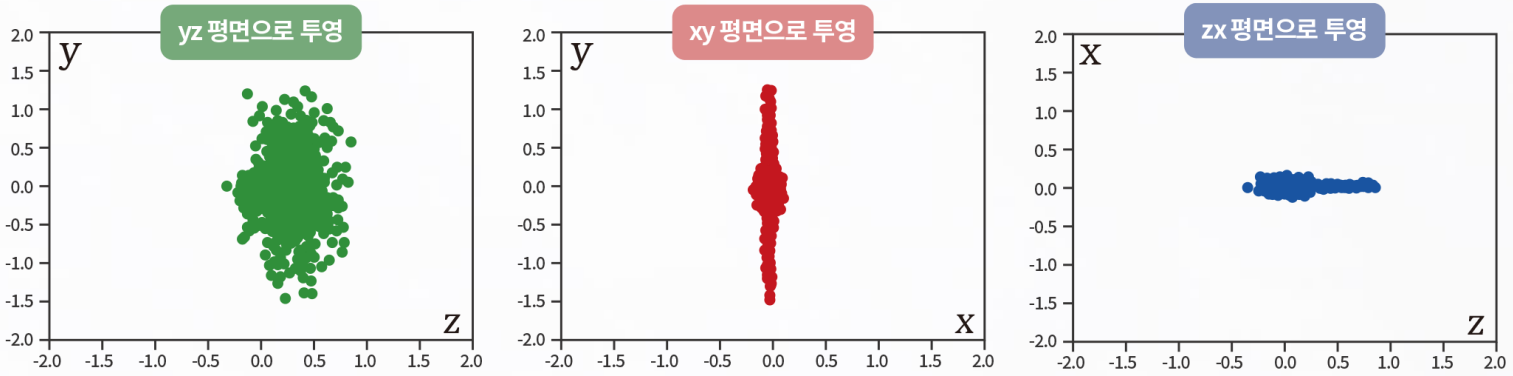
- 아래 그림에서 데이터들에 대해 중요한 축의 순서는 y, z, x의 순서라는 것을 알 수 있음
  - 데이터를 낮은 차원으로 떨어뜨리는 것을 투영(projection)이라고 부름
    - 원본 데이터의 분산을 최대로 유지하는 방향으로 투영이 일어나게 하는 것이 특징 투영(feature projection)의 목표임



출처: 강영민 외, 2021, 머신러닝

05 | 차원 축소를 위한 접근 방법: 투영

- 아래의 그림처럼 축을 하나 제외함으로써 가장 간단한 형태의 직교 투영을 할 수 있음
  - 아래 그림에서 가장 정보량이 적은 x축을 없앤 투영은 데이터의 분산을 잘 유지하고 있음
    - 두 번째로 많은 정보를 가진 z축을 없앤 가운데 그림은 데이터의 특징을 많이 잃은 것을 확인할 수 있음
    - 아래 그림의 오른쪽 끝의 그림은 가장 정보가 많은 y축을 생략해 버려 가장 많은 정보가 더 소실되었음

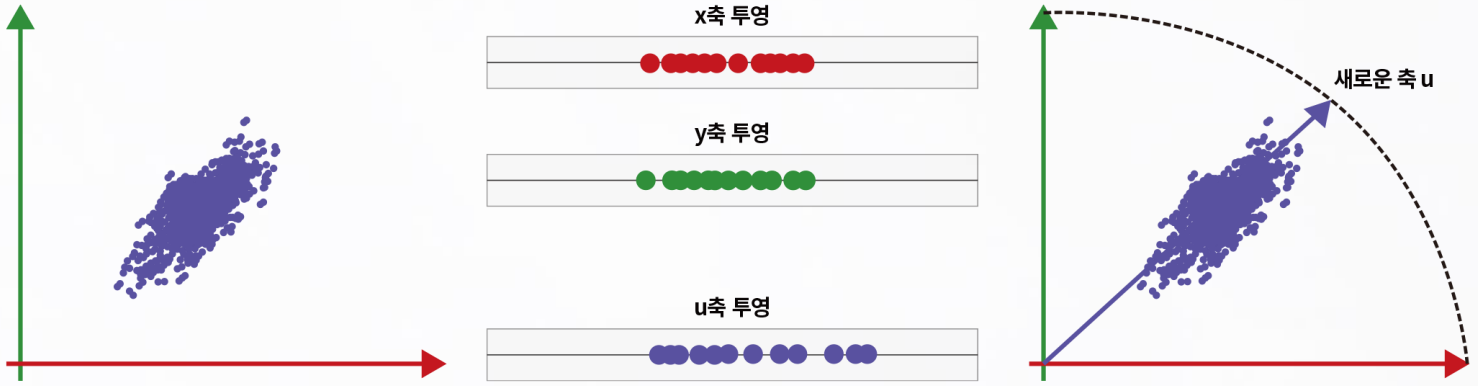


출처: 강영민 외, 2021, 머신러닝

06 | 차원 축소를 위한 접근 방법: 주성분 분석

주성분 분석(Principal Component Analysis, PCA)

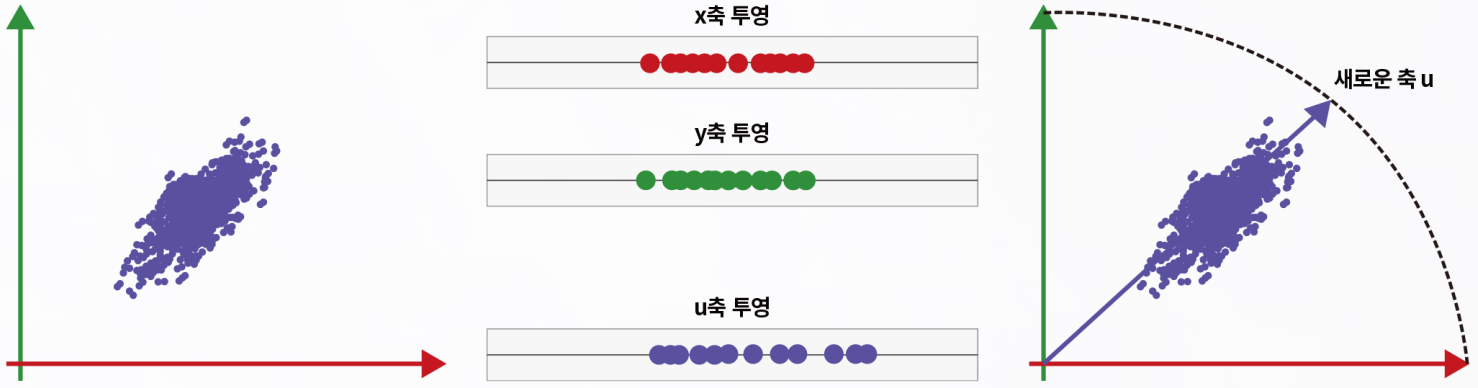
- 아래 그림과 같이 2차원 공간의 데이터가 있다고 가정하자.
  - 이 경우에는 어떤 축을 선택해야 할지 생각해 보자.
    - x축으로의 분산과 y축으로의 분산이 거의 같음



출처: 강영민 외, 2021, 머신러닝

06 | 차원 축소를 위한 접근 방법: 주성분 분석

- 아래 그림에서 x축을 선택하든지 또는 y축을 선택하든지 비슷한 수준의 정보 손실을 갖게 될 것임
  - 아래 그림의 오른쪽 끝의 그림과 같이 새로운 축 u를 도입하고, 이 축의 위로 데이터를 투영하면 어떨까?
    - 훨씬 더 분산을 잘 유지할 수 있을 것임



출처: 강영민 외, 2021, 머신러닝

06 | 차원 축소를 위한 접근 방법: 주성분 분석

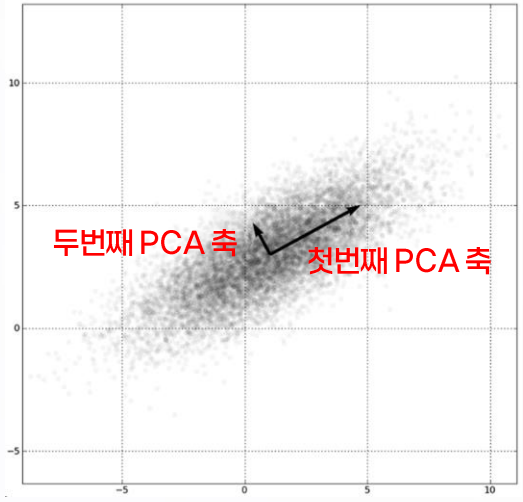
- x축을 (1, 0)으로 y축을 (0, 1)로 표현할 때
  - u축이 $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ 이라면
    - 데이터의 x축 특징과 y축 특징을 각각 $\frac{1}{\sqrt{2}}$ 만큼 고려할 때 분산이 잘 유지되는 데이터를 만들 수 있다는 것임
      - 이것이 차원 축소 문제에서 가장 일반적으로 사용되는 주성분 분석의 기본 아이디어임

u축이 $(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ 이라면

출처: 강영민 외, 2021, 머신러닝

06 | 차원 축소를 위한 접근 방법: 주성분 분석

- 데이터의 분산을 가장 잘 유지하는 축들을 주성분이라고 함
  - 주성분은 여러 개의 축으로 구성되는데 첫 번째 주성분은 데이터의 분산을 가장 잘 표현하는 방향벡터임
    - 두 번째 주성분은 첫 번째 주성분과 직교하는 축들 가운데 데이터의 분산을 가장 잘 보존하는 축(방향벡터) 임
    - k번째 주성분은 이전의 k-1개의 주성분과 모두 직교하면서 데이터의 분산을 가장 잘 보존하는 축이라고 정의할 수 있음



06 | 차원 축소를 위한 접근 방법: 주성분 분석

주성분 분석은 다음과 같은 단계로 이루어짐

1. 학습 데이터셋에서 분산이 최대인 축(axis)을 찾음
2. 이렇게 찾은 축에 직교하면서 남은 분산을 최대로 유지하는 축 찾기
3. 원하는 투영면의 차원에 도달할 때까지 2번을 반복
   (또는 데이터셋의 차원(특성 수)만큼의 축을 찾음)

✦ 찾은 순서에 따라 i 번째 발견한 축을 i 번째 주성분이라 부름

[실습] 주성분 분석이 필요한 이유

아이리스(IRIS, 붓꽃) 데이터셋으로 주성분 분석이 필요한 이유에 관해 살펴보자.

- 아이리스 데이터 셋은 꽃잎의 각 부분의 너비와 길이 등을 측정한 데이터임
  - 관측치는 150개, 속성은 6개로 구성되어 있음
  - 아이리스 꽃은 아래 그림과 같음

열이름	설명
Caseno	일련번호
Sepal Length	꽃받침의 길이 정보
Sepal Width	꽃받침의 너비 정보
Petal Length	꽃잎의 길이 정보
Petal Width	꽃잎의 너비 정보
Species	꽃의 종류 정보 (setosa, versicolor,virginica)

07 | [실습] 주성분 분석이 필요한 이유

다음은 아이리스 데이터 셋을 읽어오는 코드이다.

- 독립변수의 데이터와 속성 명으로 데이터프레임을 생성함
  - 실행 결과 아래와 같이 독립변수가 4개, 관측치가 150개인 것을 알 수 있음

```
#loading dataset
iris = datasets.load_iris()

#creating data frame for pandas
dataframe = pd.DataFrame(iris['data'], columns=iris['feature_names'])
dataframe
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

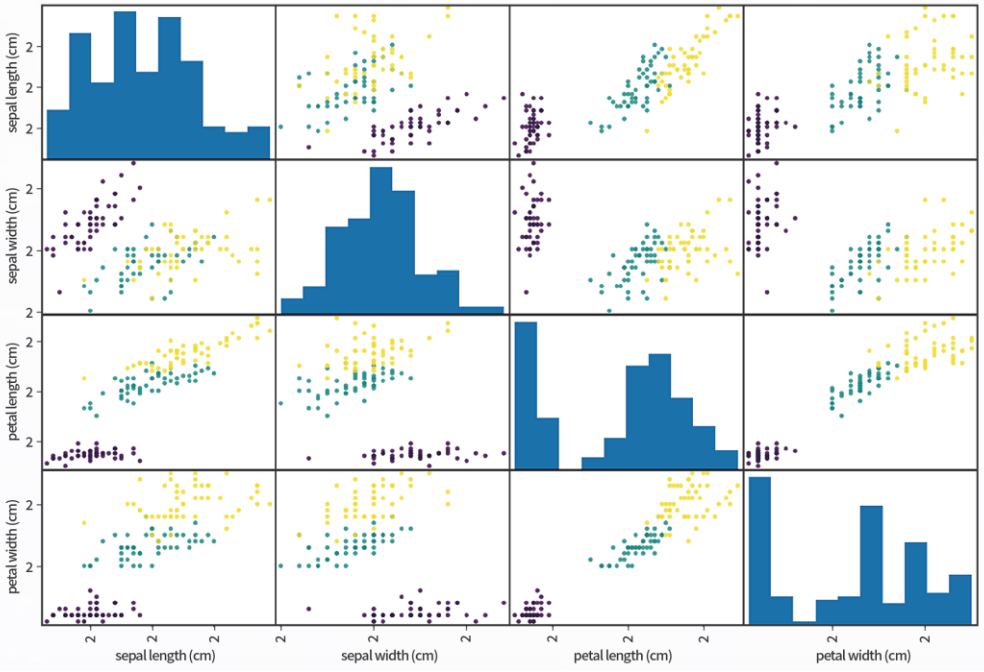
150 rows x 4 columns

07 | [실습] 주성분 분석이 필요한 이유

다음은 아이리스 데이터 셋으로 산점 행렬(scatter matrix)을 그리는 코드이다.

- 산점 행렬은 데이터프레임의 열 간의 상호 관계를 시각적으로 분석하기 위한 유용한 도구임
  - 실행 결과 데이터프레임의 열 간의 산점도를 하나의 그림으로 표시된 것을 볼 수 있음

```
scatter_matrix(dataframe, c=iris['target'], marker='o', s=10,alpha=.8, figsize=(12,8))
plt.show()
```



07 | [실습] 주성분 분석이 필요한 이유

다음은 독립변수, 종속변수, 독립변수 속성 명, 종속변수 각 레이블을 확인하는 코드이다.

- 종속변수의 각 레이블은 0=setosa, 1=versicolor, 2=virginica 인 것을 알 수 있음

```
X = iris.data                                  # 독립변수
y = iris.target                                # 종속변수
feature_names = iris.feature_names # 독립변수 속성명
target_names = iris.target_names      # 각 레이블
print(X[:5])
print(y)
print(feature_names)
print(target_names)
```

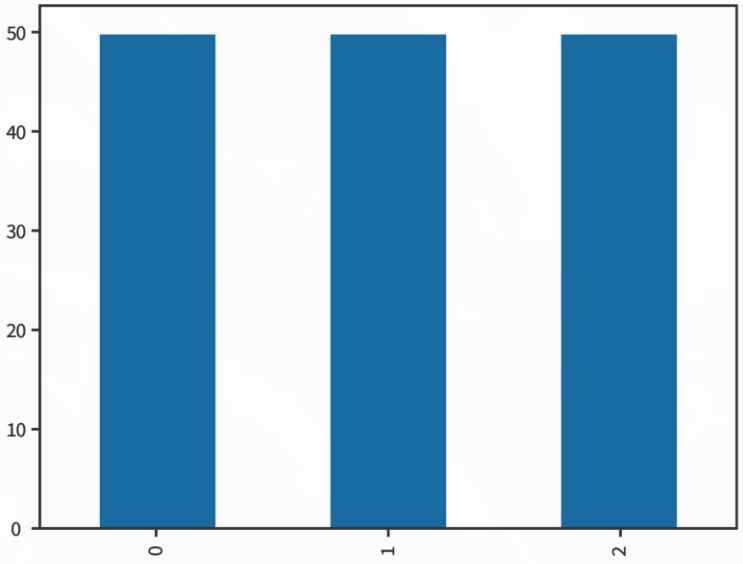
```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
```

07 | [실습] 주성분 분석이 필요한 이유

다음은 종속변수의 막대 그래프를 그리는 코드이다.

- 종속변수는 0, 1, 2로 범주화되어 있고, 각 50씩 고르게 분포하고 있는 것을 확인할 수 있음

```
# 종속변수의 빈도 막대 그래프
df_X = pd.DataFrame(X)
df_Y = pd.DataFrame(y)
df_Y[0].value_counts().plot(kind='bar')
plt.show()
```

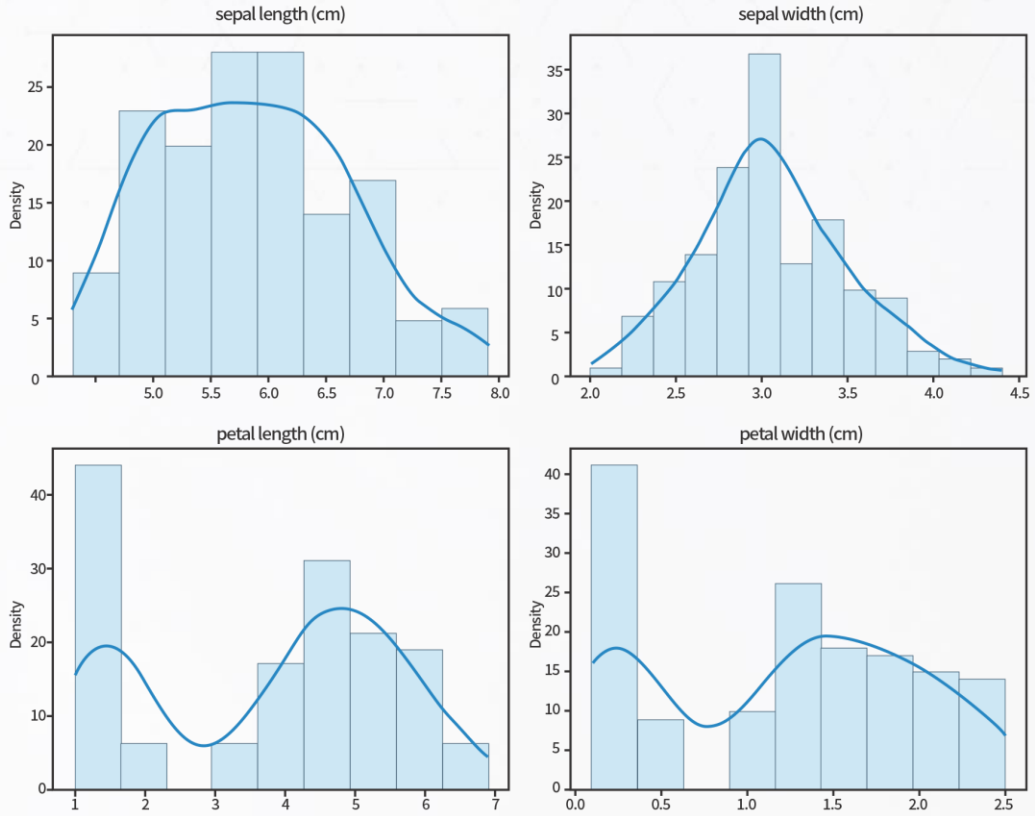


07 | [실습] 주성분 분석이 필요한 이유

다음은 독립변수 데이터프레임의 각 열에 히스토그램을 그리는 코드이다.

- 각 변수들은 정규분포에 가까운 그래프를 그려주는 것을 확인할 수 있음

```
for i in range(df_X.shape[1]):
    sns.histplot(df_X.iloc[:, i], kde=True)  # kde=True를 통해 KDE 곡선을 추가
    plt.title(feature_names[i])
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.show()
```



07 | [실습] 주성분 분석이 필요한 이유

다음은 독립변수를 주성분 분석(PCA)을 이용해 학습 및 변환하는 코드이다.

- 주성분 분석 클래스의 객체를 이용해 독립변수를 변환하는 것을 볼 수 있음

```
pca = PCA()                          # PCA 객체 생성
pca.fit(X)                           # 독립변수로 학습
PC_score = pca.transform(X)  # 독립변수(데이터)를 변환
PC_score[:5]
# array([[-2.68412563e+00,  3.19397247e-01, -2.79148276e-02, -2.26243707e-03],
#        [-2.71414169e+00, -1.77001225e-01, -2.10464272e-01, -9.90265503e-02],
#        [-2.88899057e+00, -1.44949426e-01,  1.79002563e-02, -1.99683897e-02],
#        [-2.74534286e+00, -3.18298979e-01,  3.15593736e-02,  7.55758166e-02],
#        [-2.72871654e+00,  3.26754513e-01,  9.00792406e-02,  6.12585926e-02]])
```

07 | [실습] 주성분 분석이 필요한 이유

다음은 주성분 분석을 통해 **설명된 분산**(explained variance)**값**인 **주성분 값**으로 **계산된 설명된 분산의 비율**을 **확인**하는 코드이다.

- **주성분 값**이 **클수록 설명력**이 **높음**
  - 아래의 결과에서 **첫 번째 주성분 값**(약 4.228)이 가장 크므로 **가장 설명력**이 **높은 축일 것**으로 생각할 수 있음
  - **주성분 값**으로 **설명된 분산의 비율**을 **계산**한 결과 PC1이 **약 92.46%**의 **설명력**을 **가지는 것**을 알 수 있음

```
# 설명된 분산(explained_variance)의 값
print(pca.explained_variance_)

# 고유값으로 설명된 분산의 비율(explained variance ratio)을 계산함
print(pca.explained_variance_ / np.sum(pca.explained_variance_))
```

```
[4.22824171 0.24267075 0.0782095  0.02383509]
[0.92461872 0.05306648 0.01710261 0.00521218]
```

07 | [실습] 주성분 분석이 필요한 이유

- 다음은 학습된 주성분 분석 클래스의 객체를 이용해 설명된 설명된 분산의 비율(explained variance ratio)을 확인하는 코드이다.
  - 실행 결과 PC1이 약 92.46%, PC2가 약 5.3%의 설명력을 가짐

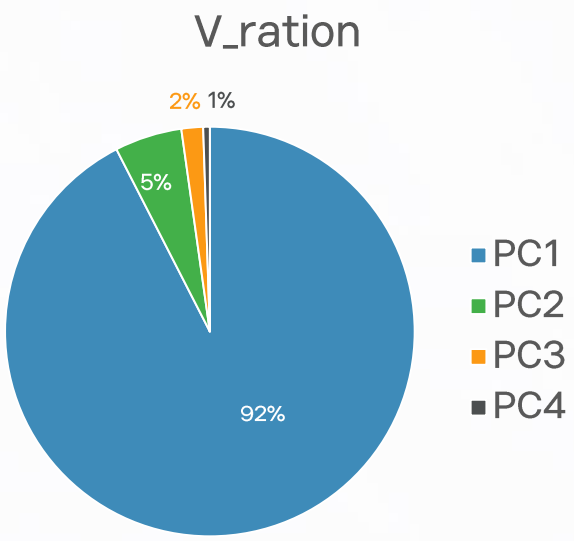
```
ratio = pca.explained_variance_ratio_ratio
# array([0.92461872, 0.05306648, 0.01710261, 0.00521218])
```

07 | [실습] 주성분 분석이 필요한 이유

다음은 학습된 주성분 분석 클래스의 객체를 이용해
설명된 분산의 비율(explained variance ratio)을 파이 그래프로 그리는 코드이다.

실행 결과 PC1이 약 92.46%, PC2가 약 5.3%의 설명력을 가짐

```
df_v = pd.DataFrame(ratio, index=['PC1','PC2','PC3','PC4'], columns=['V_ration'])
df_v.plot.pie(y='V_ration', labels=df_v.index, autopct='%1.1f%%')
df_v
```



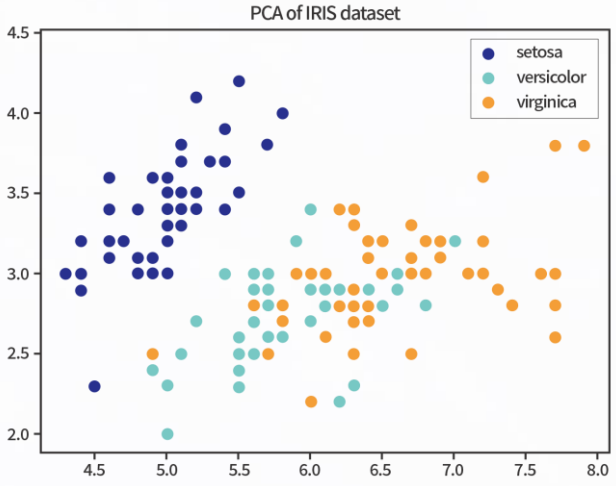
	V_ration
PC1	0.924619
PC2	0.053066
PC3	0.017103
PC4	0.005212

07 | [실습] 주성분 분석이 필요한 이유

다음은 아이리스 데이터셋의 “sepal-length”, “sepal-width” 변수로 데이터 분포를 확인하는 코드이다.

- 실행 결과 아래 그림과 같이 잘 분류가 되지 않는 것을 알 수 있음

```
plt.figure()
colors = ['navy', 'turquoise', 'darkorange']
lw = 2
for color, i, target_name in zip(colors, [0,1,2] , iris.target_names):
   plt.scatter(X[y == i, 0], X[y == i, 1], color=color, alpha=.8, lw=lw, label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of IRIS dataset')
```

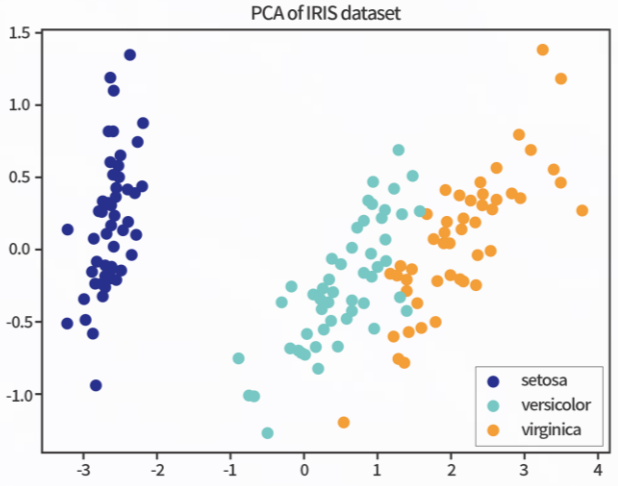


07 | [실습] 주성분 분석이 필요한 이유

다음은 주성분 분석 클래스의 객체에 의해 변환된 데이터의 PC1, PC2 값으로 데이터 분포를 확인하는 코드이다.

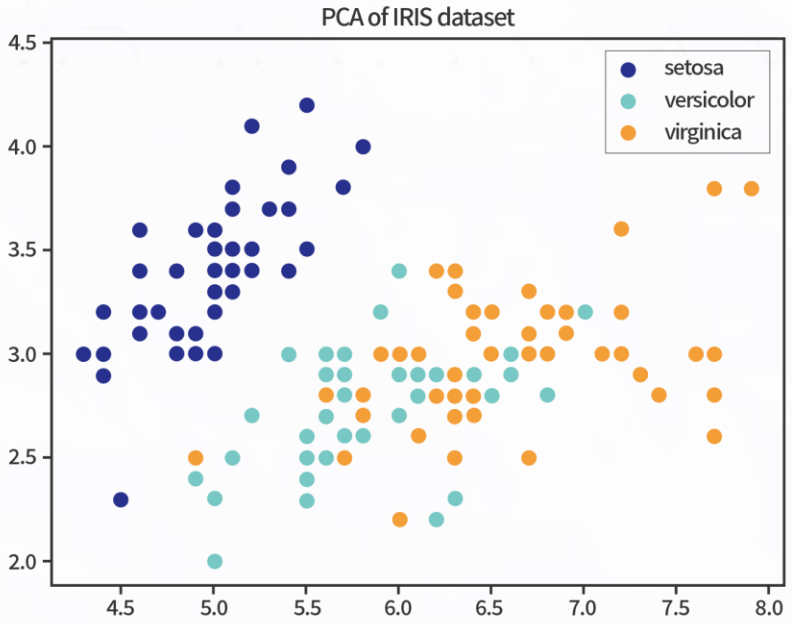
실행 결과 아래 그림과 같이 변수(sepal-length, sepal-width)의 산점도 보다 잘 분류된 것을 알 수 있음

```
plt.figure()
colors = ['navy', 'turquoise', 'darkorange']
lw = 2
for color, i, target_name in zip(colors, [0,1,2] , iris.target_names):
   plt.scatter(PC_score[y == i, 0], PC_score[y == i, 1], color=color, alpha=.8, lw=lw, label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of IRIS dataset')
```

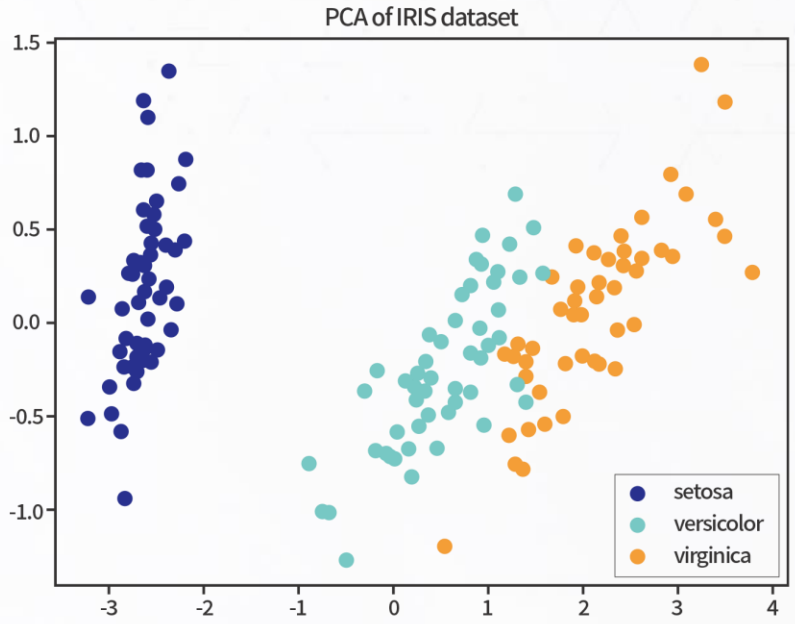


07 | [실습] 주성분 분석이 필요한 이유

- 다음은 변수(sepal-length, sepal-width)와 변수(PC1, PC2)의 산점도를 비교한 결과이다.
  - 아래 그림과 같이 PC1, PC2 산점도가 좀 더 잘 분류된 것을 볼 수 있음



변수(sepal-length, sepal-width)산점도



변수(PC1, PC2)산점도

07 | [실습] 주성분 분석이 필요한 이유

다음은 아이리스 데이터셋에서 4개의 독립변수로 로지스틱 회귀(logistic regression)모델을 만들고 학습 및 평가 결과를 확인하는 코드이다.

- 실행 결과 4개의 값이 잘못 예측된 것을 알 수 있음

```
clf = LogisticRegression(max_iter=1000, random_state=0, multi_class='multinomial')
clf.fit(X,y)
pred = clf.predict(X)
confusion_matrix(y,pred)
```

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  1, 49]], dtype=int64)
```

07 | [실습] 주성분 분석이 필요한 이유

다음은 주성분 분석 클래스의 객체를 이용해 변환된 4개의 독립변수로 로지스틱 회귀(logistic regression) 모델을 만들고 학습 및 평가 결과를 확인하는 코드이다.

- 실행 결과 4개의 값이 잘못 예측된 것을 알 수 있음

```
clf.fit(PC_score,y)                 # 모델 학습
pred = clf.predict(PC_score)  # 모델 예측
confusion_matrix(y,pred)        # 정오분류표 작성
```

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  1, 49]], dtype=int64)
```

07 | [실습] 주성분 분석이 필요한 이유

- 다음은 아이리스 데이터셋에서 4개의 독립변수와 주성분 분석 클래스의 객체를 이용해 변환된 4개의 독립변수로 로지스틱 회귀 모델을 만들고 학습 및 평가 결과를 비교한 것임
  - 아래와 같이 두 결과 모두 4개의 값이 잘못 예측된 것을 알 수 있음
    - 즉, 두 예측 결과는 동일함

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  1, 49]], dtype=int64)
```

변수 (sepal-length, sepal-width, petal-length, petal-width) 정오분류표

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  1, 49]], dtype=int64)
```

변수 (PC1, PC2, PC3, PC4) 정오분류표

07 | [실습] 주성분 분석이 필요한 이유

- 다음은 아이리스 데이터셋에서 2개의 독립변수(sepal-length, sepal-width)로 로지스틱 회귀 모델을 만들고 학습 및 평가 결과를 확인하는 코드이다.
  - 실행 결과 27개의 값이 잘못 예측된 것을 알 수 있음

```
clf = LogisticRegression(max_iter=1000, random_state=0,
multi_class='multinomial')
clf.fit(X[:,:2],y)
pred = clf.predict(X[:,:2])
confusion_matrix(y,pred)
```

```
array([[50,  0,  0],
       [ 0, 37, 13],
       [ 0, 14, 36]], dtype=int64)
```

07 | [실습] 주성분 분석이 필요한 이유

- 다음은 **주성분 분석 클래스의 객체**를 이용해 **변환된 2개**(PC1, PC2)의 **독립변수**로 **로지스틱 회귀** 모델을 만들고 **학습 및 평가 결과**를 확인하는 코드이다.
  - 실행 결과 **5개의 값**이 **잘못 예측된 것**을 알 수 있음

```
clf2 = LogisticRegression(max_iter=1000, random_state=0,
multi_class='multinomial')
clf2.fit(PC_score[:,:2],y)
pred = clf2.predict(PC_score[:,:2])
confusion_matrix(y,pred)
```

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  2, 48]], dtype=int64)
```

07 | [실습] 주성분 분석이 필요한 이유

- 다음은 아이리스 데이터셋에서 2개의 독립변수와 주성분 분석 클래스의 객체를 이용해 변환된 2개의 독립변수로 로지스틱 회귀 모델을 만들고 학습 및 평가 결과를 비교한 것임
  - 아래와 같이 두 결과가 확연하게 차이가 나는 것을 확인할 수 있음
    - 즉, PC1이 상당히 많은 설명력을 가지고 있기 때문에 PC1과 PC2 만으로도 대부분을 분류할 수 있음

```
array([[50,  0,  0],
       [ 0, 37, 13],
       [ 0, 14, 36]], dtype=int64)
```

변수 (sepal-length, sepal-width) 정오분류표

```
array([[50,  0,  0],
       [ 0, 47,  3],
       [ 0,  2, 48]], dtype=int64)
```

변수 (PC1, PC2) 정오분류표