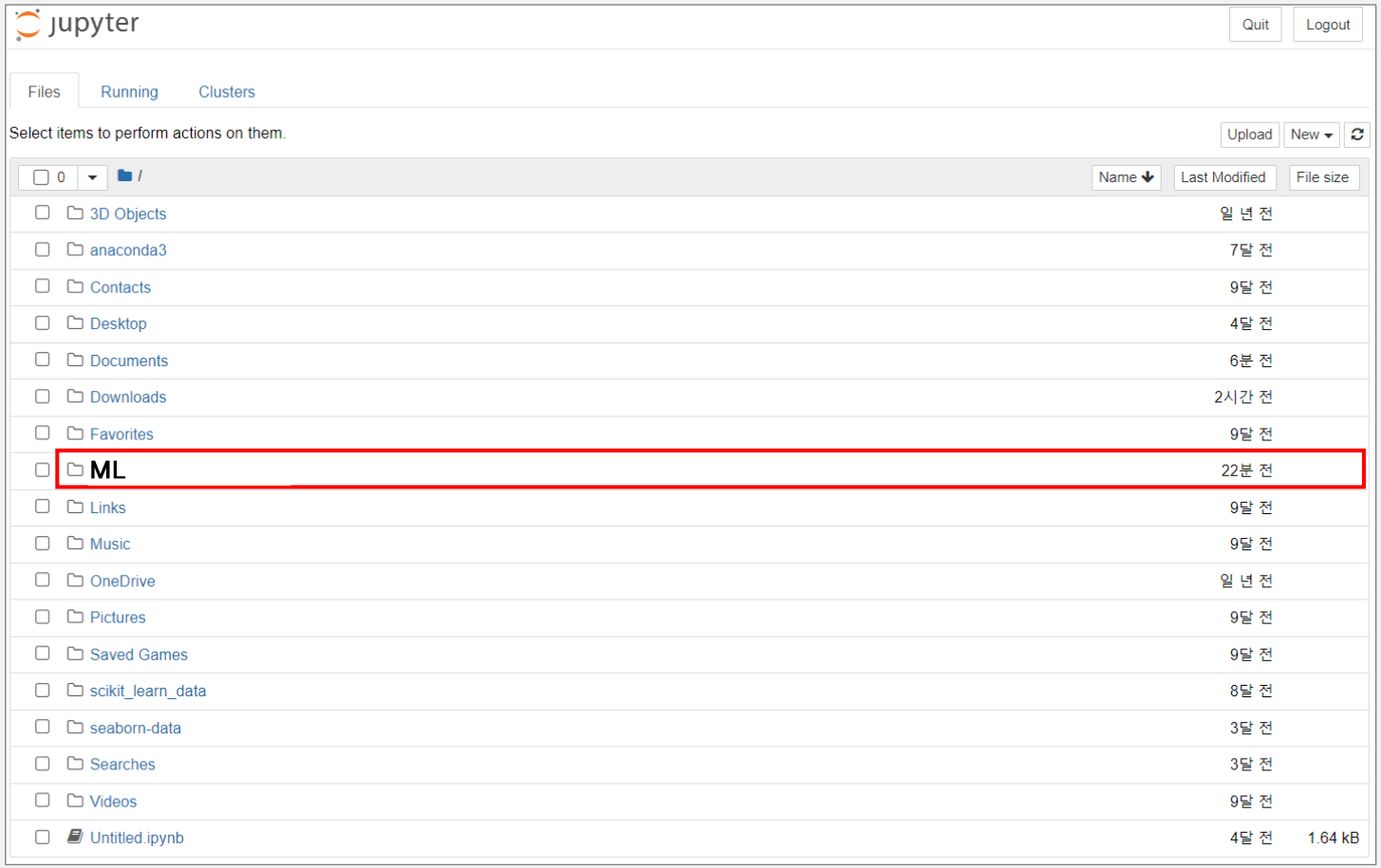

강원지역혁신플랫폼
기계학습
Machine Learning
주성분 분석 실습

학습목표

- 주성분 분석을 구현할 수 있습니다.

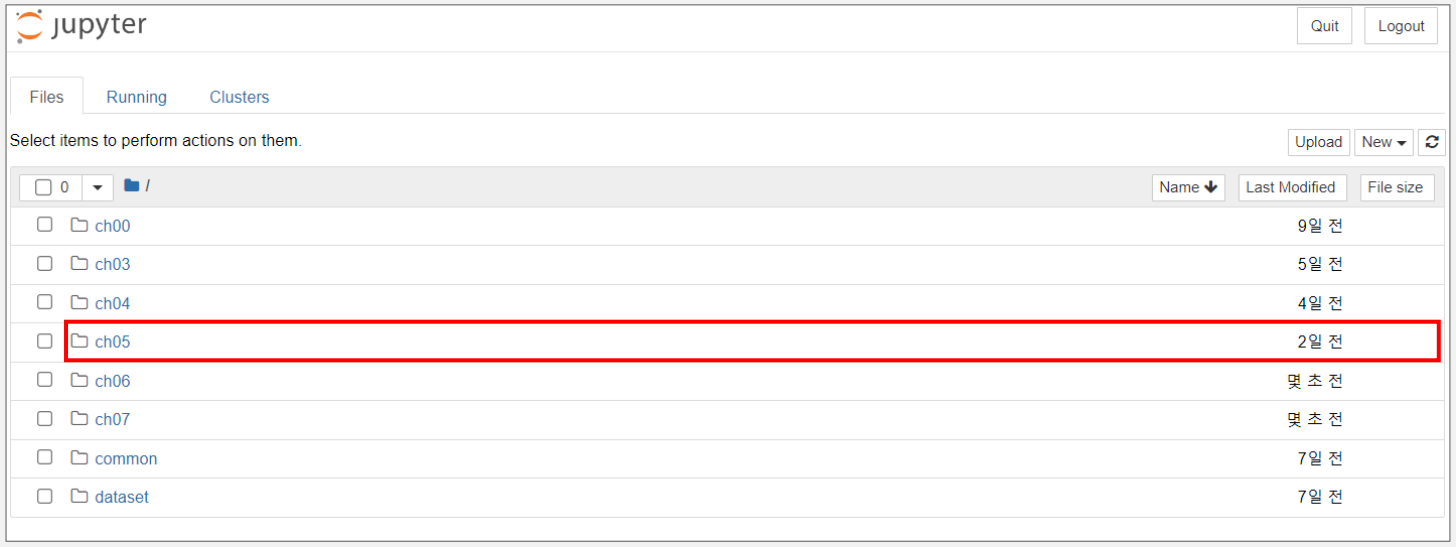
01 | ML 폴더

ML 폴더를 클릭하기



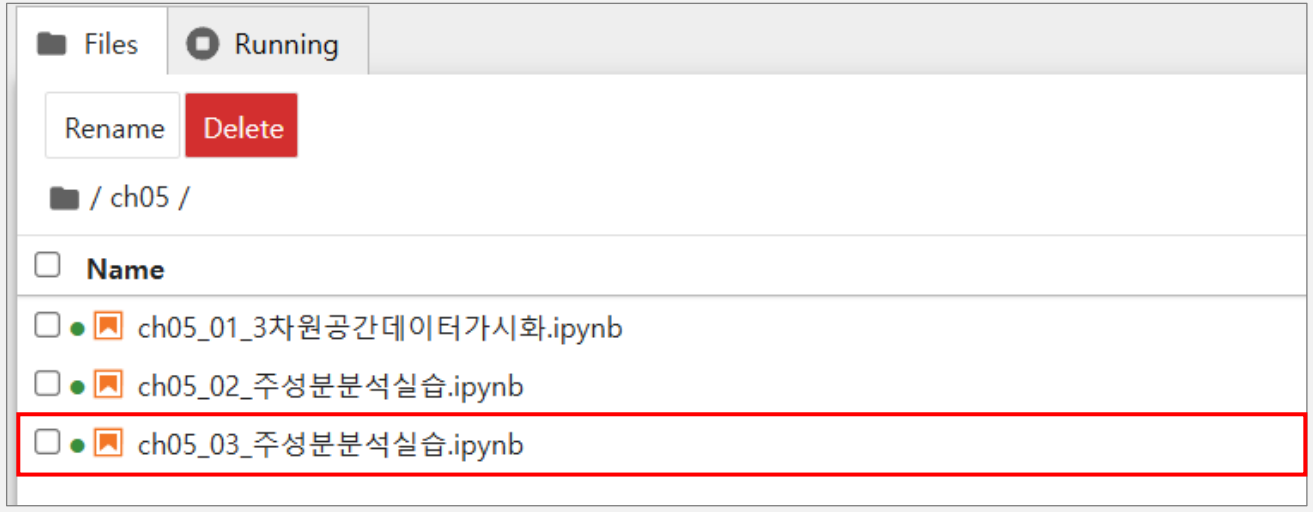
02 | ch05 폴더

ch05 폴더 클릭하기



03 | ch05_03_주성분분석실습.ipynb

✦ ch05_03_주성분분석실습.ipynb 파일 클릭하기



04 | [실습] 주성분 분석

[실습] 주성분 분석

와인(Wine) 등급 데이터셋으로 주성분 분석을 수행함

✦ 와인 등급 데이터셋은 다음과 같이 13개의 독립 변수로 구성되어 있음

변수 명		변수 명	
영문	한글	영문	한글
alcohol	알콜	flavanoids	플라보노이드 폴리페놀
malic_acid	말산	nonflavanoid_phenols	비 플라보노이드 폴리페놀
ash	회분	proanthocyanins	프로안토시아닌
alcalinity_of_ash	회분의 알칼리도	color_intensity	색상의 강도
magnesium	마그네슘	hue	색상
total_phenols	총 폴리페놀	od280/od315_of_diluted_wines	희석 와인의 OD280/OD315 비율
proline	프롤린		

04 | [실습] 주성분 분석

- 다음은 와인 데이터셋을 읽어오는 코드이다.
  - Bunch 객체는 몇 가지의 key를 제공하며 이를 통해 데이터의 정보를 쉽게 확인할 수 있음
    - Bunch 객체인 data의 keys() 함수로 주요 key를 확인할 수 있음
      - data : 모델에 제공할 학습 데이터
      - target : 정답 데이터, 라벨(label), 클래스(class) 데이터

```
# 와인 데이터 불러오기
data = datasets.load_wine()
print(type(data))
print(data.keys())
```

```
<class 'sklearn.utils._bunch.Bunch'>
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

04 | [실습] 주성분 분석

다음은 와인 데이터셋에서 독립 변수는 변수 X, 종속 변수는 변수 y에 할당하는 코드이다.

- 종속 변수의 각 레이블은 “class_0”, “class_1”, “class_2” 인 것을 알 수 있음
  - 독립 변수는 13개로 구성된 것을 알 수 있음

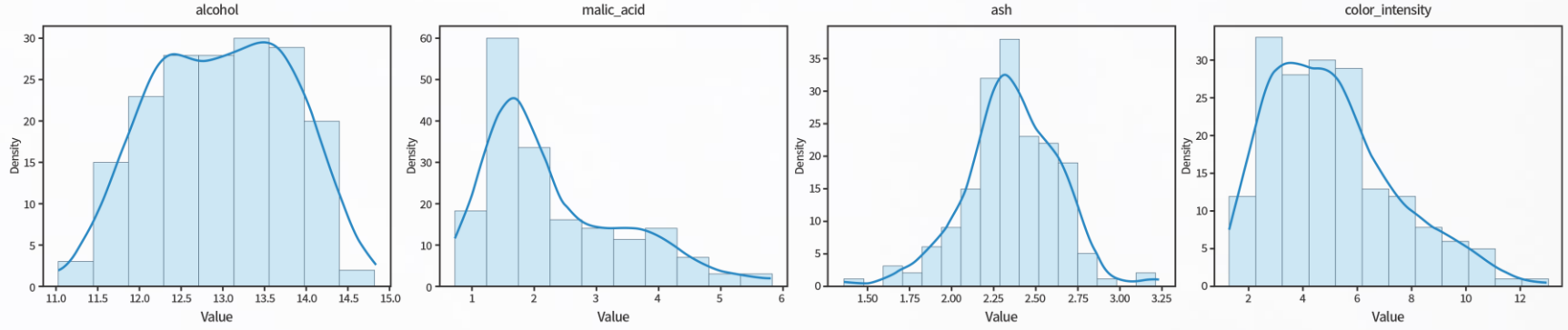
```
X = data.data
y = data.target
feature_names = data.feature_names
print(data.target_names)   # ['class_0' 'class_1' 'class_2']
print(data.feature_names)
# ['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols',
#  'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins', 'color_intensity', 'hue',
#  'od280/od315_of_diluted_wines', 'proline']
```

04 | [실습] 주성분 분석

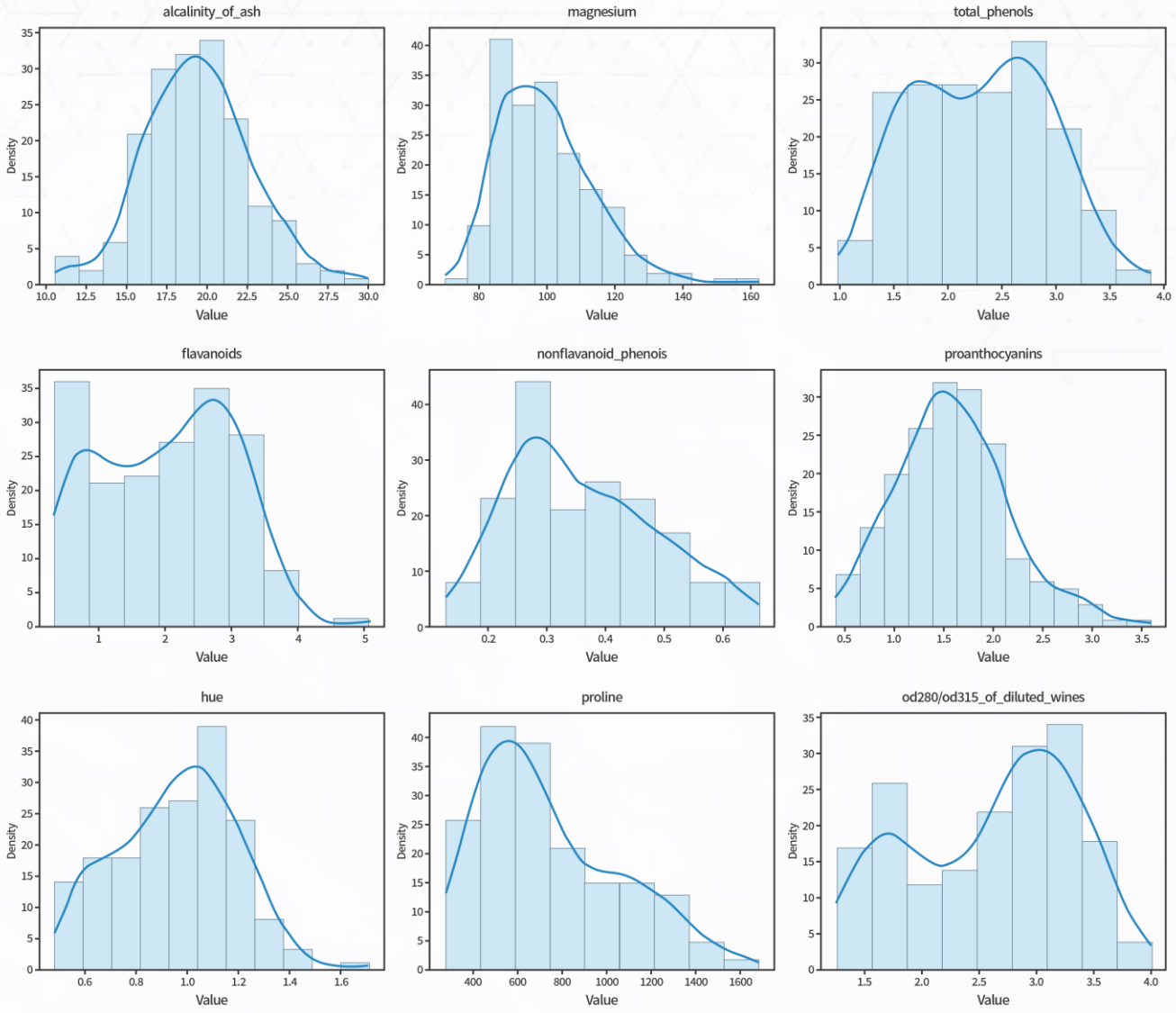
다음은 독립 변수의 히스토그램을 그리는 코드이다.

- 독립 변수들의 분포를 확인하며 정규성을 확인함
  - 변수들은 정규분포에 가까운 그래프를 그려주는 것을 확인할 수 있음

```
dfX = pd.DataFrame(X)
dfy = pd.DataFrame(y)
# DataFrame의 각 열에 대해 분포 플롯 그리기
for i in range(dfX.shape[1]):
    sns.histplot(dfX.iloc[:, i], kde=True)  # kde=True를 통해 KDE 곡선을 추가할 수 있음
    plt.title(feature_names[i])
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.show()
```



04 | [실습] 주성분 분석


alcalinity_of_ash
magnesium
total_phenols
flavanoids
nonflavanoid_phenois
proanthocyanins
hue
proline
od280/od315_of_diluted_wines
Density
Value

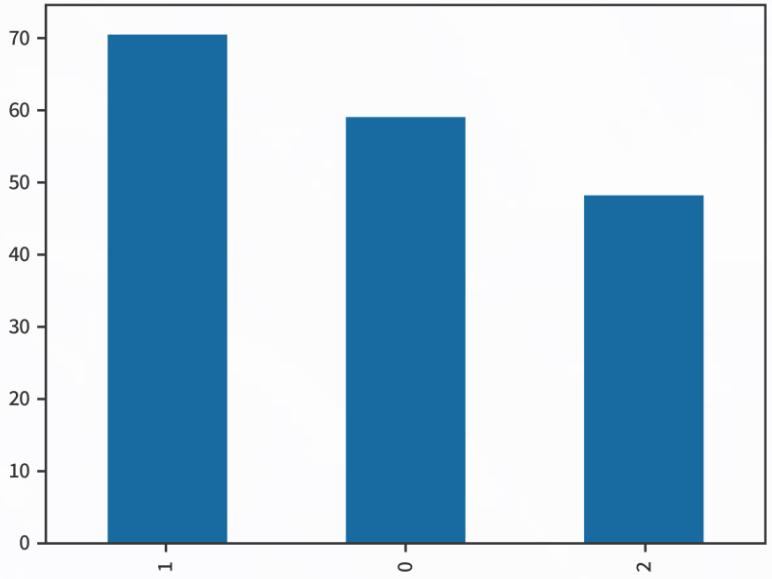
04 | [실습] 주성분 분석

다음은 종속 변수가 골고루 있는지 막대 그래프를 그리는 코드이다.

- 아래와 같이 “class_1” 이 71개, “class_0” 이 59개, “class_2”가 48개인 것을 알 수 있음

```
pd.DataFrame(y)[0].value_counts().plot(kind='bar')
pd.DataFrame(y)[0].value_counts()
```

```
1    71
0    59
2    48
Name: 0, dtype: int64
```



04 | [실습] 주성분 분석

다음은 와인 데이터셋을 훈련 데이터 80%, 시험 데이터 20%로 분리하는 코드이다.

- 아래와 같이 훈련 데이터 142개, 시험 데이터 36개로 분리된 것을 볼 수 있음

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
print(x_train.shape)  # (142, 13)
print(y_train.shape)  # (142,)
print(x_test.shape)   # (36, 13)
print(y_test.shape)   # (36,)
```

04 | [실습] 주성분 분석

다음은 훈련 데이터셋을 가공하지 않고 로지스틱 회귀 모델을 생성 및 학습하는 코드이다.

- 아래와 같이 종속 변수가 3개의 레이블이므로 다중 분류를 적용하여 모델을 생성함
  - 훈련 데이터로 모델 학습을 수행함

```
clf = LogisticRegression(max_iter=2000, random_state=0, multi_class='multinomial', solver='sag')
clf.fit(x_train, y_train)    # 학습
```

04 | [실습] 주성분 분석

다음은 데이터 가공 없이 학습된 모델에 시험 데이터로 예측 및 평가를 수행하는 코드이다.

- 모델의 정확도는 약 97%이고, 1개의 데이터가 잘못 예측된 것을 볼 수 있음

```
# 예측
y_pred = clf.predict(x_test)

# 모델 성능 평가
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.9722222222222222
Confusion Matrix:
[[13  1  0]
 [ 0 16  0]
 [ 0  0  6]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.93      0.96        14
           1       0.94      1.00      0.97        16
           2       1.00      1.00      1.00         6

    accuracy                           0.97        36
   macro avg       0.98      0.98      0.98        36
weighted avg       0.97      0.97      0.97        36
```

04 | [실습] 주성분 분석

다음은 주성분 분석 모델을 생성하고, 와인 독립 변수(X)로 학습 및 변환하는 코드이다.

- 주성분 분석 모델을 생성하여 와인 독립 변수(X)로 학습을 시킴
  - 학습된 주성분 객체에 독립 변수를 변환시켜 pcscore 변수에 할당함

```
# PCA 모델 생성하여 와인 데이터 전체의 독립 변수로 학습함
pca = PCA()
pca.fit(X)

# 주성분 분석 클래스의 객체를 이용해 독립 변수를 변환하여 PC score를 구함
pcscore = pca.transform(X)
```

04 | [실습] 주성분 분석

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**이 가장 크므로 **가장 설명력**이 **높은 축일 것**으로 생각할 수 있음

› **주성분 값**으로 **설명된 분산의 비율**을 **계산**한 결과 **PC1**이 **약 99.8%**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_))
```

```
[9.92017895e+04 1.72535266e+02 9.43811370e+00 4.99117861e+00
 1.22884523e+00 8.41063869e-01 2.78973523e-01 1.51381266e-01
 1.12096765e-01 7.17026032e-02 3.75759789e-02 2.10723661e-02
 8.20370314e-03]
[9.98091230e-01 1.73591562e-03 9.49589576e-05 5.02173562e-05
 1.23636847e-05 8.46213034e-06 2.80681456e-06 1.52308053e-06
 1.12783044e-06 7.21415811e-07 3.78060267e-07 2.12013755e-07
 8.25392788e-08]
```

04 | [실습] 주성분 분석

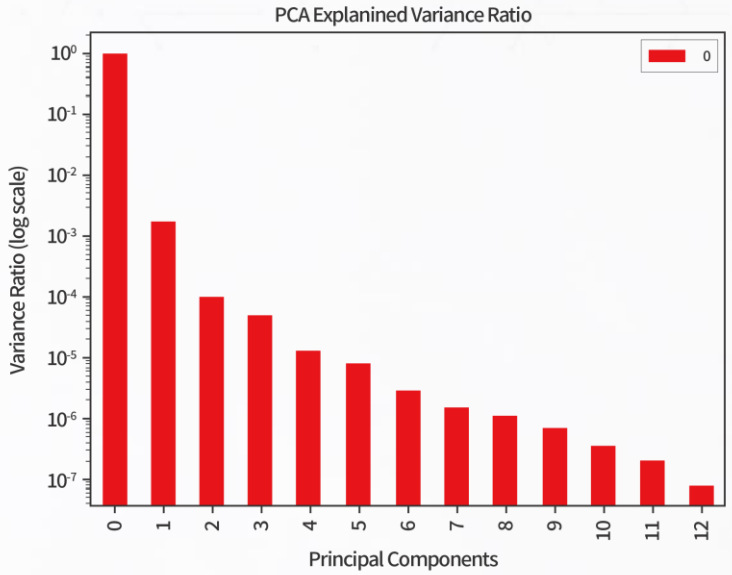
다음은 설명된 분산의 비율(explained variance ration)을 확인하고 막대 그래프로 출력하는 코드이다.

- 첫 번째 PC1의 설명력이 상당히 높은 것을 확인할 수 있음
  - 아래의 그래프와 같이 설명도 정도를 비율로 확인할 수 있음

```
PC_ratio = pca.explained_variance_ratio_
PC_ratio_df = pd.DataFrame(PC_ratio)
print(PC_ratio)

# 막대 그래프 그리기
PC_ratio_df.plot(kind='bar', logy=True, color='r')
plt.title('PCA Explained Variance Ratio')
plt.xlabel('Principal Components')
plt.ylabel('Variance Ratio (log scale)')
plt.show()
```

```
[9.98091230e-01 1.73591562e-03 9.49589576e-05 5.02173562e-05
 1.23636847e-05 8.46213034e-06 2.80681456e-06 1.52308053e-06
 1.12783044e-06 7.21415811e-07 3.78060267e-07 2.12013755e-07
 8.25392788e-08]
```



04 | [실습] 주성분 분석

다음은 주성분 분석 모델을 생성하고 훈련 데이터셋으로 학습, 학습된 모델로 훈련 데이터와 시험 데이터를 변환하는 코드이다.

- 여기서 주의할 점은 PCA 모델에 훈련 데이터로 학습시키고, 학습된 모델에 훈련 데이터와 시험 데이터를 변환한다는 것임

```
# 훈련 데이터셋으로 PCA 모델 학습
pca = PCA()pca.fit(x_train)

# 훈련 데이터셋으로 학습된 PCA 모델로 훈련 데이터와 시험 데이터 변환
# PC score 구하기 : 훈련 데이터, 시험 데이터
train_score = pca.transform(x_train)
test_score = pca.transform(x_test)
```

04 | [실습] 주성분 분석

다음은 PCA를 적용하지 않은 데이터로 로지스틱 회귀 모델을 학습, 예측, 평가하는 코드이다.

- 여기서는 훈련 데이터셋과 시험 데이터셋의 모든 독립 변수를 이용함
- 아래와 같이 모델의 정확도는 약 97.2%인 것을 알 수 있음
  - 예측 결과에서 1개의 데이터가 잘못 분류된 것을 알 수 있음

```
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
cf1 = confusion_matrix(y_test, pred)
print(cf1)
print(accuracy_score(y_test, pred))
```

```
[[13  1  0]
 [ 0 16  0]
 [ 0  0  6]]
0.9722222222222222
```

04 | [실습] 주성분 분석

다음은 PCA를 적용한 데이터로 로지스틱 회귀 모델을 학습, 예측, 평가하는 코드이다.

- 여기서는 훈련 데이터셋과 시험 데이터셋의 모든 독립 변수를 이용함
- 아래와 같이 모델의 정확도는 약 97.2%인 것을 알 수 있음
  - 예측 결과에서 1개의 데이터가 잘못 분류된 것을 알 수 있음

```
clf2.fit(train_score, y_train)
pred2 = clf2.predict(test_score)
cf2 = confusion_matrix(y_test, pred2)
print(cf2)
print(accuracy_score(y_test, pred2))
```

```
[[13  1  0]
 [ 0 16  0]
 [ 0  0  6]]
0.9722222222222222
```

04 | [실습] 주성분 분석

다음은 PCA를 적용하지 않은 데이터로 로지스틱 회귀 모델을 학습, 예측, 평가하는 코드이다.

- 여기서는 훈련 데이터셋과 시험 데이터셋의 첫 번째(alcohol)와 두 번째(malic_acid)의 독립 변수를 이용함
- 아래와 같이 모델의 정확도는 약 69.4%인 것을 알 수 있음
  - 예측 결과에서 11개의 데이터가 잘못 분류된 것을 알 수 있음

```
clf.fit(x_train[:,:2], y_train)
pred = clf.predict(x_test[:,:2])
cf1 = confusion_matrix(y_test, pred)
print(cf1)
print(accuracy_score(y_test, pred))
```

```
[[10  2  2]
 [ 1 13  2]
 [ 3  1  2]]
0.6944444444444444
```

04 | [실습] 주성분 분석

- 다음은 PCA를 적용한 데이터로 로지스틱 회귀 모델을 학습, 예측, 평가하는 코드이다.
  - 여기서는 훈련 데이터셋과 시험 데이터셋의 첫 번째와 두 번째의 독립 변수를 이용함
  - 아래와 같이 모델의 정확도는 약 83.3%인 것을 알 수 있음
    - 예측 결과에서 6개의 데이터가 잘못 분류된 것을 알 수 있음

```
clf2.fit(train_score[:,:2], y_train)
pred2 = clf2.predict(test_score[:,:2])
cf2 = confusion_matrix(y_test, pred2)
print(cf2)
print(accuracy_score(y_test, pred2))
```

```
[[13  0  1]
 [ 0 16  0]
 [ 1  4  1]]
0.8333333333333334
```

04 | [실습] 주성분 분석

다음은 훈련 데이터셋과 시험 데이터셋에서 2개의 독립 변수와 주성분 분석 클래스의 객체를 이용해 변환된 2개의 독립 변수로 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

```
[[10  2  2]
 [ 1 13  2]
 [ 3  1  2]]
0.6944444444444444
```

변수 (alcohol, malic_acid) 로 정확도와 정오분류표

```
[[13  0  1]
 [ 0 16  0]
 [ 1  4  1]]
0.8333333333333334
```

변수 (PC1, PC2) 로 정확도와 정오분류표

04 | [실습] 주성분 분석

다음은 PCA를 적용하지 않은 데이터에서 2개 독립 변수를 순서가 없이 조합하여 로지스틱 회귀 모델에 학습, 예측, 평가를 수행하고 그 결과를 저장하는 코드이다.

- 아래와 같이 모델 평가 결과를 score_board 변수에 저장함
  - 아래와 같이 0번째와 5번째 인덱스 위치에 저장된 독립 변수로 학습, 예측, 평가 결과 가장 높은 정확도(약 88.89%)가 계산됨

```
# 점수들을 모두 담기 위한 빈 리스트
score_board = []

# 순서가 없고 조합되지 않은 모든 케이스에 대해서 모델을 생성하고 결과를 확인
for comb in combinations(list(range(0,x_train.shape[1])),2):
    clf.fit(x_train[:,comb],y_train)
    pred = clf.predict(x_test[:,comb])
    score = accuracy_score(y_test, pred)
    score_board.append(score)
    print(comb)
    print(score)
```

```
(0, 5)
0.8888888888888888
```

04 | [실습] 주성분 분석

- 다음은 0번째, 5번째 위치의 독립 변수와 최대 정확도를 확인하는 코드이다.
  - 0번째의 독립 변수는 “alcohol”, 5번째의 독립 변수는 “total_phenols” 인 것을 알 수 있음
    - 최대 정확도는 약 88.89%이고, 이 값은 PCA를 사용한 것보다 더 정확도가 높은 것을 볼 수 있음

```
# 0번째, 5번째 위치의 독립 변수: alcohol total_phenols
print(feature_names[0],feature_names[5])

# 최대 점수 확인
print(max(score_board))
```

```
alcohol total_phenols
0.8888888888888888
```

04 | [실습] 주성분 분석

정리해 보면

- 데이터와 변수(feature)에 따라서 상황은 다르지만 주성분 분석(PCA)이 더 좋거나 원시 데이터(raw data)가 더 좋다고 할 수는 없음
- 현재 자신이 분석하는 데이터와 상황에 따라서 적절하게 맞는 것을 확인해서 사용하는 것이 바람직해 보임
  - 앞의 실험 결과에서 PCA가 모델의 성능을 높여주는 것은 아님
    - 하지만, 빠르게 필요 없는 변수(feature)들을 없애고 핵심적인 변수(feature)들만을 뽑아서 모델을 만드는 것에는 유용할 수 있음