

파이썬을 활용한 머신러닝

쉬움주의

# 파이썬으로 knn 구현하기

## ■ [쉬움주의] 유방암 데이터의 악성종양을 knn 으로 분류하기

유방암 데이터: R을 활용한 머신러닝 -에이콘 출판사

파이썬 코드 전체:

```
import pandas as pd # 데이터 전처리를 위함
import seaborn as sns # 시각화를 위함

df = pd.read_csv("d:\data\wisc_bc_data.csv")
# 설명 : R과는 다르게 stringsAsFactors=True를 지정하지 않아도 된다.
print(df)

# DataFrame 확인
print(df.shape) # ( 569, 32 ) 569행 32열이다.
print(df.info()) # R의 str 함수와 유사함. 데이터 구조 확인
print(df.describe()) #R의 summary(df)의 결과와 유사함. 요약 통계 정보

# 행을 선택하는 방법 emp[행][열] -> emp[조건][컬럼명]
print(df.iloc[0:5, ]) #df데이터 프레임에서 0~4번째 행을 가져와라. df.iloc[행번호, 열번호]
```

```
/head  
print(df.iloc[-5:, :]) #df 데이터의 끝에서 5번째 행부터 끝까지 가져와라 라는 뜻 /tail
```

```
#열을 선택하는 방법 emp[행][열] --> emp[조건][컬럼명] emp[ , c("ename", "sal")]  
print(df.iloc[ :, [0,1] ]) # 0번째 열과 1번째 열을 가져와라  
print(df.iloc[ :, : ]) #전체 열을 다 가져와라
```

```
#판다스 데이터 프레임이 어떻게 구성되었는가?  
# numpy 리스트( 일반 리스트 )로 컬럼 하나를 구성 -> 시리즈  
# numpy 리스트( 일반 리스트 )로 컬럼 여러개로 구성 -> 데이터 프레임  
# https://cafe.daum.net/oracleoracle/SqNT/19 참조  
# R에서 emp <- read.csv("emp.csv")  
# str(emp) 데이터 프레임  
# 데이터 전처리 : 정규화 ---> 훈련과 테스트로 데이터를 분리
```

```
#%%
```

```
# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트  
X = df.iloc[:, 2: ].to_numpy() #df 데이터 프레임의 2번째 열부터 끝까지를 numpy array로 변  
환해라  
#print(X)  
y = df['diagnosis'].to_numpy()  
#print(y)  
print(df.shape)
```

**#데이터 정규화를 수행한다.**

```
# 1. 스케일: 평균은 0이고 표준편차 1인 데이터로 분포 시킴  
# 2. min/max 정규화 : 0~1 사이의 숫자로 변경  
# 아래의 코드는 min/max 정규화는 아니고 scale이다.
```

```
from sklearn import preprocessing  
X=preprocessing.StandardScaler().fit(X).transform(X)  
print(X)  
# scale이 잘 되었는지 확인은 아래에 있다.  
from sklearn.model_selection import train_test_split
```

**# 훈련 데이터 70, 테스트 데이터 30으로 나눈다.**

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.3, random_state = 10)  
# 설명 : test_size=0.3으로 했기 때문에 훈련과 테스트가 7대 3비율로 나뉜다.
```

```
# random_state=10은 seed값 설정하는 부분이다.
```

```
# 어느 자리에서든 동일한 정확도를 보기 위해서이다.
```

```
print(X_train.shape) # (398,30)  
print(y_train.shape) # (398,
```

```

#%%
# 스케일링(z-score 표준화 수행 결과 확인)
for col in range(4):
    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')

for col in range(4):
    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')

# 설명 : 평균은 0에 가깝고 표준편자는 1에 가까운 결과가 출력되고 있다.

#%%
# 학습/예측(Training/Prediction)
from sklearn.neighbors import KNeighborsClassifier

# k-NN 분류기를 생성
classifier = KNeighborsClassifier(n_neighbors=5) #knn 모델 생성

# 분류기 학습
classifier.fit(X_train, y_train) # 훈련 데이터와 훈련 데이터의 라벨로 훈련을 함

# 예측
y_pred= classifier.predict(X_test) # 테스트 데이터 예측
print(y_pred)
print(len(y_pred)) #171개 예측

#%%
# 모델 평가 / 작은 이원교차표 출력
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)
# [[97  1]
# [ 5 68]]

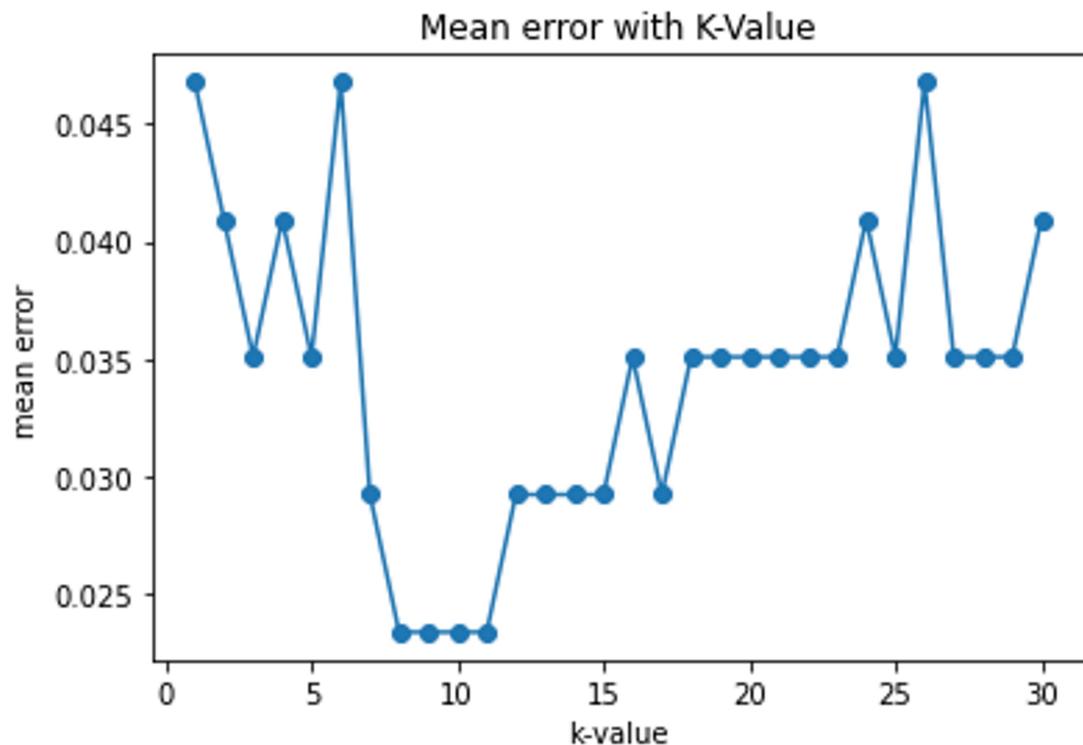
# 이원 교차표 보는 코드
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy)

# k값이 5일 때 정확도 0.96

```

# 문제1. 위의 코드에서 적절한 k 값을 알아내는 for 문을 구현하세요.



답 :

```
import numpy as np

errors = []
for i in range(1, 31):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    errors.append(np.mean(pred_i != y_test))
print(errors)
```

```
for k, i in enumerate(errors):
    print(k, '-->', i)
```

#7 ~100| error가 가장작다.

```
import matplotlib.pyplot as plt

plt.plot(range(1, 31), errors, marker='o')
plt.title('Mean error with K-Value')
plt.xlabel('k-value')
plt.ylabel('mean error')
plt.show()
```

## 문제2. 위에서 알아낸 가장 에러가 낮은 k값은 7,8,9,10이었다. k값을 7을 넣었을 때의 정확도를 보시오~

k-NN 분류기를 생성

```
classifier = KNeighborsClassifier(n_neighbors=7) #knn 모델 생성
```

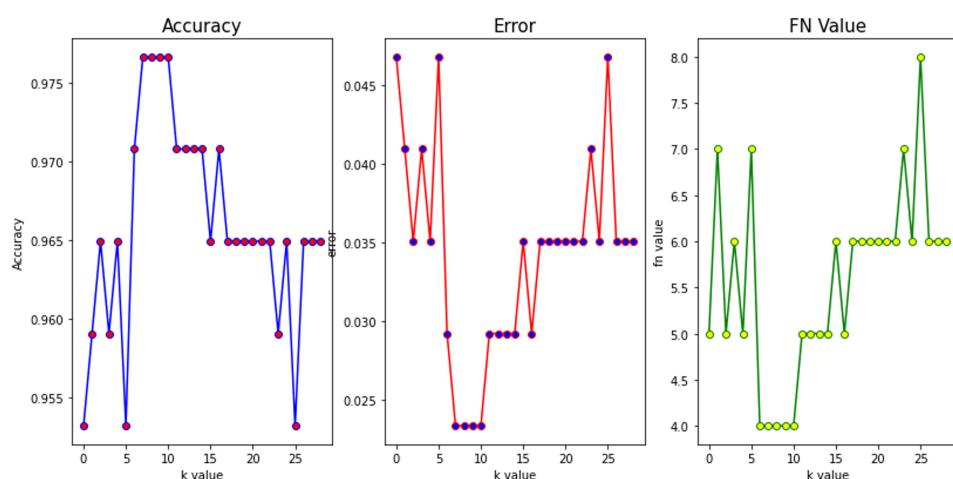
답 : 0.97의 정확도가 나온다. 의료 데이터이므로 정확도가 아주 높아야 한다. 정확도가 100%가 나오면 좋겠는데 100%가 나오기 어려우므로 FN을 0으로 만들면 정확도가 100%가 아니더라도 쓰겠다는 경우가 많다.

관심 범주는 positive가 암이므로 Negative는 정상환자이다.

False Negatvie --> 정상환자로 잘못 예측했다.

방금의 시각화는 k 값이 변경될 때마다 오류가 어떻게 되는지 2차원 그래프로 시각화 한 것이고 이번에는 k값이 변경될 때마다 FN 값과 정확도가 어떻게 되는지 확인해야 한다.

## 문제3. 위의 코드에서 적절한 k 값을 알아내는 for 문을 구현하세요.



```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np

acclist = []
err_list = []
fn_list = []
```

```
for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_pred).ravel()
    fn_list.append(fn)
    acclist.append(accuracy_score(y_test, y_pred))
    err_list.append(np.mean(y_pred != y_test))

print(f'k : {i} , acc : {accuracy_score(y_test, y_pred)} , FN : {fn}'')
```

# 설명 : .ravel() 함수를 안쓰면 작은 이원 교차표가 나오는데 .ravel()을 쓰면 이원교차표의  
#값들을 출력할 수 있다.

### #그래프 사이즈 조정 부분

```
plt.figure(figsize=(12,6))
plt.subplots_adjust(left=0.125,
                   bottom=0.1,
                   right=1,
                   top=0.9,
                   wspace=0.2,
                   hspace=0.35)
```

### # k 값이 변경될 때마다 정확도가 어떻게 되는지 시각화 하는 부분

```
plt.subplot(131)
plt.plot(acclist,color='blue', marker='o', markerfacecolor='red')
plt.title('Accuracy', size=15)
plt.xlabel("k value")
plt.ylabel('Accuracy')
```

### #k값이 변경될 때마다 에러가 어떻게 되는지 시각화 하는 부분

```
plt.subplot(132)
plt.plot(err_list, color='red', marker='o', markerfacecolor='blue')
plt.title('Error', size=15)
plt.xlabel("k value")
plt.ylabel('error')
```

### # k값이 변경될 때마다 FN이 어떻게 되는지 시각화 하는 부분

```
plt.subplot(133)
plt.plot(fn_list, color='green', marker='o', markerfacecolor='yellow')
plt.title('FN Value', size=15)
plt.xlabel("k value")
plt.ylabel('fn value')
```

```
plt.show()
```

## 문제4. iris 데이터를 knn 으로 분류하세요 !

```
import matplotlib.pyplot as plt
```

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np
import pandas as pd

# 1. 데이터 준비
col_names = ['sepal-length', 'sepal-width','petal-length', 'petal-width','Class']

# csv 파일에서 DataFrame을 생성
dataset = pd.read_csv('d:\data\iris2.csv', encoding='UTF-8', header=None,
names=col_names)
#print(dataset)

# 아래의 코드 ?

```

**문제 5. ( 점심시간 문제 )** iris 데이터에 대해서 가장 정확도가 좋은 k값을 지정해서 아이리스 데이터를 분류하는 knn 모델을 생성하는 전체코드를 올리시오!

답 :

```

acclist = []
err_list = []
f1_list = []

for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    report = classification_report(y_test, y_pred, digits=2, output_dict=True)
    f1 = report['macro avg']['f1-score']
    f1_list.append(f1)
    acclist.append(report['macro avg']['precision'])
    err_list.append(np.mean(y_pred != y_test))

print(f'k : {i} , acc : {accuracy_score(y_test, y_pred)} , F1-score : {f1}')

df_s = pd.DataFrame(data=dict(k=range(1,30),acc=acclist,err=err_list,F1_score=f1_list))

print(df_s[df_s['acc']==1])

```

**문제6. 유방암 데이터의 정확도를 더 올리기 위해서 정규화를 min max 정규화로 변경하시오!**

```
[[98  0]
 [5 68 ]]
```

```
k=12
기준정확도 : 0.9707602339181286
```

#문제 6번을 위한 준비코드 :

### # ■ [쉬움주의] 유방암 데이터의 악성종양을 knn 으로 분류하기

```
import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서
```

```
df = pd.read_csv("d:\data\wisc_bc_data.csv")
```

#### # DataFrame 확인

```
#print(df.shape)
#print(df.info())
```

```
#print(df.describe())
```

```
#print(df.iloc[0:5, ])
```

```
#print(df.iloc[-5: ,])
```

```
#print(df.iloc[ :, [0,1] ])
#print(df.iloc[ :, : ])
```

#### # X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트

```
X = df.iloc[:, 2: ].to_numpy()
```

```
y = df['diagnosis'].to_numpy()
#print(y)
```

```
#print(df.shape) # (569, 32)
```

```
#print(len(X)) # 569
```

```
#print(len(y)) # 569
```

```
from sklearn import preprocessing
```

```
#X=preprocessing.StandardScaler().fit(X).transform(X) # 정규화 하는 코드 : scale함수 적용
```

```
X=preprocessing.MinMaxScaler().fit(X).transform(X) # min/max 함수 적용
```

```
from sklearn.model_selection import train_test_split

# 훈련 데이터 70, 테스트 데이터 30으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state = 10)

print(X_train.shape)
print(y_train.shape)

# 스케일링(z-score 표준화 수행 결과 확인)
for col in range(4):
    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')

for col in range(4):
    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')

# 학습/예측(Training/Prediction)
from sklearn.neighbors import KNeighborsClassifier

# k-NN 분류기를 생성
classifier = KNeighborsClassifier(n_neighbors=12)

# 분류기 학습
classifier.fit(X_train, y_train)

# 예측
y_pred= classifier.predict(X_test)
print(y_pred)

# 작은 이원교차표 출력
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)

# 정밀도, 재현율, F1 score
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy)
```

```
#기존 정확도 : 0.9707602339181286 --> 0.9883040935672515  
# 기준보다 정확도가 더 올라갔다.
```

## ■ 2장. 나이브베이즈를 파이썬으로 구현하기

R이 좋은 함수와 패키자가 파이썬보다 더 많다. ( 역사가 더 깊다)  
파이썬으로 머신러닝을 구현하는 경우가 현업에서 더 많다.

앞에서 knn으로 머신러닝을 구현할 때 R과의 차이점?  
factor로 변환 할 필요가 없었다.

```
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split  
from sklearn import metrics  
import numpy as np  
import pandas as pd
```

### # 1. 데이터 준비

```
col_names = ['sepal-length', 'sepal-width','petal-length', 'petal-width','Class']
```

### # csv 파일에서 DataFrame을 생성

```
dataset = pd.read_csv('d:\data\iris2.csv', encoding='UTF-8', header=None,  
names=col_names)  
#print(dataset)
```

### # DataFrame 확인

```
print(dataset.shape) # (row개수, column개수)  
print(dataset.info()) # 데이터 타입, row 개수, column 개수, 컬럼 데이터 타입  
print(dataset.describe()) # 요약 통계 정보  
  
print(dataset.iloc[0:5]) # dataset.head()  
print(dataset.iloc[-5:]) # dataset.tail()
```

```
# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트  
X = dataset.iloc[:, :-1].to_numpy() # DataFrame을 np.ndarray로 변환  
#print(X)
```

```
# 전체 데이터 세트를 학습 세트(training set)와 검증 세트(test set)로 나눔  
# y = 전체 행, 마지막 열 데이터  
y = dataset.iloc[:, 4].to_numpy()  
#print(y)
```

## # 데이터 분리

```
from sklearn.model_selection import train_test_split
```

### # 전체 데이터 세트를 학습 세트(training set)와 검증 세트(test set)로 나눔

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 10)
print(len(X_train), len(X_test))
```

```
print(X_train[:3])
print(y_train[:3])
```

### # 3. 거리 계산을 위해서 각 특성들을 스케일링(표준화)

#### # Z-score 표준화: 평균을 0, 표준편차 1로 변환

```
from sklearn.preprocessing import StandardScaler
```

### # 3. 거리 계산을 위해서 각 특성들을 스케일링(표준화)

#### # Z-score 표준화: 평균을 0, 표준편차 1로 변환

```
scaler = StandardScaler() # Scaler 객체 생성
```

```
scaler.fit(X_train) # 스케일링(표준화)를 위한 평균과 표준 편차 계산
```

```
X_train = scaler.transform(X_train) # 스케일링(표준화) 수행
```

```
X_test = scaler.transform(X_test)
```

### # 스케일링(z-score 표준화 수행 결과 확인)

```
for col in range(4):
```

```
    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')
```

```
for col in range(4):
```

```
    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')
```

## # 4. 학습/예측(Training/Prediction)

```
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearn.naive_bayes import GaussianNB #Gaussian naivebayes가 훨씬 잘 분류함
```

```
#model = GaussianNB() # Gaussian Naive Bayes 모델 선택 - 연속형 자료
```

```
# model = GaussianNB(var_smoothing=1e-09) # Gaussian Naive Bayes 모델 선택 - 연속형 자료
```

```
# model = GaussianNB() 안에 값이 없어도 잘 나옴
```

```
# model = BernoulliNB(alpha=0.1)
```

```
model = BernoulliNB()
```

```
model.fit( X_train, y_train )
```

## # 예측

```
y_pred= model.predict(X_test)
```

```
print(y_pred)
```

## #5. 모델 평가

```
from sklearn.metrics import confusion_matrix  
conf_matrix= confusion_matrix(y_test, y_pred)  
print(conf_matrix)
```

# 대각선에 있는 숫자가 정답을 맞춘 것, 그 외가 틀린 것

```
from sklearn.metrics import classification_report  
report = classification_report(y_test, y_pred)  
print(report)
```

## # 이원 교차표 보는 코드

```
from sklearn import metrics  
naive_matrix = metrics.confusion_matrix(y_test,y_pred)  
print(naive_matrix)
```

## # 정확도 확인하는 코드

```
from sklearn.metrics import accuracy_score  
accuracy = accuracy_score( y_test, y_pred)  
print(accuracy) # 0.73
```

## 문제7. 위의 나이브 베이즈 모델의 성능을 더 올리시오~

기존 정확도 : 0.7333 ---> 개선 후 정확도 : 100%

↓                          ↓  
BernoulliNB              Gaussian NB

답 코드 :

```
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split  
from sklearn import metrics  
import numpy as np  
import pandas as pd
```

### # 1. 데이터 준비

```
col_names = ['sepal-length', 'sepal-width','petal-length', 'petal-width','Class']
```

### # csv 파일에서 DataFrame을 생성

```
dataset = pd.read_csv('d:\data\iris2.csv', encoding='UTF-8', header=None,  
names=col_names)  
# print(dataset)
```

### # DataFrame 확인

```
print(dataset.shape) # (row개수, column개수)
print(dataset.info()) # 데이터 타입, row 개수, column 개수, 컬럼 데이터 타입
print(dataset.describe()) # 요약 통계 정보

print(dataset.iloc[0:5]) # dataset.head()
print(dataset.iloc[-5:]) # dataset.tail()
```

```
# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = dataset.iloc[:, :-1].to_numpy() # DataFrame을 np.ndarray로 변환
#print(X)
```

```
# 전체 데이터 세트를 학습 세트(training set)와 검증 세트(test set)로 나눔
# y = 전체 행, 마지막 열 데이터
y = dataset.iloc[:, 4].to_numpy()
#print(y)
```

### # 데이터 분리

```
from sklearn.model_selection import train_test_split
```

```
# 전체 데이터 세트를 학습 세트(training set)와 검증 세트(test set)로 나눔
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 10)
print(len(X_train), len(X_test))

print(X_train[:3])
print(y_train[:3])
```

### # 3. 거리 계산을 위해서 각 특성들을 스케일링(표준화)

```
# Z-score 표준화: 평균을 0, 표준편차 1로 변환
```

```
from sklearn.preprocessing import StandardScaler
```

```
# 3. 거리 계산을 위해서 각 특성들을 스케일링(표준화)
# Z-score 표준화: 평균을 0, 표준편차 1로 변환
scaler = StandardScaler() # Scaler 객체 생성
scaler.fit(X_train) # 스케일링(표준화)를 위한 평균과 표준 편차 계산
X_train = scaler.transform(X_train) # 스케일링(표준화) 수행
X_test = scaler.transform(X_test)
```

### # 스케일링(z-score 표준화 수행 결과 확인)

```
for col in range(4):
    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')
```

```
for col in range(4):
    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')
```

#### # 4. 학습/예측(Training/Prediction)

```
from sklearn.naive_bayes import BernoulliNB  
from sklearn.naive_bayes import GaussianNB #Gaussian naivebayes가 훨씬 잘 분류함  
#model = GaussianNB() # Gaussian Naive Bayes 모델 선택 - 연속형 자료
```

```
# model = GaussianNB(var_smoothing=1e-09) # Gaussian Naive Bayes 모델 선택 - 연속형  
자료
```

```
model = GaussianNB() #안에 값이 없어도 잘 나옴  
# model = BernoulliNB(alpha=0.1)  
# model = BernoulliNB()  
model.fit( X_train, y_train )
```

#### # 예측

```
y_pred= model.predict(X_test)  
print(y_pred)
```

#### #5. 모델 평가

```
from sklearn.metrics import confusion_matrix  
conf_matrix= confusion_matrix(y_test, y_pred)  
print(conf_matrix)
```

# 대각선에 있는 숫자가 정답을 맞춘 것, 그 외가 틀린 것

```
from sklearn.metrics import classification_report  
report = classification_report(y_test, y_pred)  
print(report)
```

#### # 이원 교차표 보는 코드

```
from sklearn import metrics  
naive_matrix = metrics.confusion_matrix(y_test,y_pred)  
print(naive_matrix)
```

#### # 정확도 확인하는 코드

```
from sklearn.metrics import accuracy_score  
accuracy = accuracy_score( y_test, y_pred)  
print(accuracy) # 0.73  
#%%
```

```
import numpy as np
```

#### # 6. 모델 개선 - laplace 값을 변화시킬 때, 에러가 줄어드는지

```
errors = []  
for i in np.arange(0.0, 1.0, 0.001):  
    model = GaussianNB( var_smoothing= i)  
    model.fit(X_train, y_train)  
    pred_i = model.predict(X_test)
```

```

    errors.append(np.mean(pred_i != y_test))
print(errors)

# 여기서 에러가 가장 적은 것을 선택

import matplotlib.pyplot as plt

plt.plot( np.arange(0.0, 1.0, 0.001), errors, marker='o')
plt.title('Mean error with laplace-Value')
plt.xlabel('laplace')
plt.ylabel('mean error')
plt.show()

```

**결과 :**

```

[[10  0  0]
 [ 0 13  0]
 [ 0  0  7]]
1.0

```

## 문제8. 유방암 데이터의 나이브베이즈 모델을 파이썬으로 생성하고 정확도를 확인하시오!

```

import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np
import pandas as pd

```

### # 1. 데이터 준비

```

df = pd.read_csv("d:\data\wisc_bc_data.csv")
#print(dataset)

```

```
X = df.iloc[:, 2: ].to_numpy()
```

```
y = df['diagnosis'].to_numpy()
#print(y)
```

```
#print(df.shape) # (569, 32)
#print(len(X)) # 569
#print(len(y)) # 569
```

```
from sklearn import preprocessing
```

```
#X=preprocessing.StandardScaler().fit(X).transform(X) # 정규화 하는 코드 : scale함수 적용
```

```

X=preprocessing.MinMaxScaler().fit(X).transform(X) # min/max 함수 적용

from sklearn.model_selection import train_test_split

# 훈련 데이터 70, 테스트 데이터 30으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state = 10)

print(X_train.shape)
print(y_train.shape)

# 데이터 분리
from sklearn.model_selection import train_test_split

# 전체 데이터 세트를 학습 세트(training set)와 검증 세트(test set)로 나눔
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 10)
print(len(X_train), len(X_test))

print(X_train[:3])
print(y_train[:3])

# 3. 거리 계산을 위해서 각 특성들을 스케일링(표준화)
# Z-score 표준화: 평균을 0, 표준편차 1로 변환

from sklearn.preprocessing import StandardScaler

# 3. 거리 계산을 위해서 각 특성들을 스케일링(표준화)
# Z-score 표준화: 평균을 0, 표준편차 1로 변환
scaler = StandardScaler() # Scaler 객체 생성
scaler.fit(X_train) # 스케일링(표준화)를 위한 평균과 표준 편차 계산
X_train = scaler.transform(X_train) # 스케일링(표준화) 수행
X_test = scaler.transform(X_test)

# 스케일링(z-score 표준화 수행 결과 확인)
for col in range(4):
    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')

for col in range(4):
    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')

# 4. 학습/예측(Training/Prediction)
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB #Gaussian naivebayes가 훨씬 잘 분류함
#model = GaussianNB() # Gaussian Naive Bayes 모델 선택 - 연속형 자료

#model = GaussianNB(var_smoothing=1e-09) # Gaussian Naive Bayes모델 - 연속형 자료

```

```
model = GaussianNB() #안에 값이 없어도 잘 나옴
# model = BernoulliNB(alpha=0.1)
#model = BernoulliNB()
model.fit( X_train, y_train )
```

#### # 예측

```
y_pred= model.predict(X_test)
print(y_pred)
```

#### # 5. 모델 평가

```
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

# 대각선에 있는 숫자가 정답을 맞춘 것, 그 외가 틀린 것

```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)
```

#### # 이원 교차표 보는 코드

```
from sklearn import metrics
naive_matrix = metrics.confusion_matrix(y_test,y_pred)
print(naive_matrix)
```

#### # 정확도 확인하는 코드

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy) #0.9473684210526315
```

**문제9.** 위의 나이브 베이즈 모델을 생성할 때 위에서는 min/max 정규화를 했는데 이번에는 scale함수를 적용해서 수행하고 정확도를 확인하시오!

```
from sklearn import preprocessing
```

```
X=preprocessing.StandardScaler().fit(X).transform(X) # 정규화 하는 코드 : scale함수 적용
#X=preprocessing.MinMaxScaler().fit(X).transform(X) # min/max 함수 적용
```

#0.94로 min/max 와 차이가 없다.

## 문제10. binary.csv

데이터 선별 : 유방암 데이터, iris 데이터와 같이 종속변수가 분류이면서 수치형 데이터인 데이터로 선별을 한다.

문제 9번 코드 ++

```
# ■ [쉬움주의] 유방암 데이터의 악성종양을 knn 으로 분류하기

import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

df = pd.read_csv("d:\data\wine.csv")

# DataFrame 확인
print(df.shape) # (178, 14)
print(df.info()) #Type : 라벨컬럼
print(df.describe()) #요약정보를 나타냄
print(df)

# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = df.iloc[:, 1:14].to_numpy()
y = df['Type'].to_numpy()
#print(y)

print(df.shape) # (178, 14), length
print(len(X)) # 178
print(len(y)) # 178
#%%
# 정규화진행
from sklearn import preprocessing

X=preprocessing.StandardScaler().fit(X).transform(X)
#X=preprocessing.MinMaxScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split

# 훈련 데이터 90, 테스트 데이터 10으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1, random_state = 10)

print(X_train.shape) #(160, 13)
print(y_train.shape) # (160, )

#%%
# 스케일링(z-score 표준화 수행 결과 확인)
for col in range(4):
```

```

print('평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')

for col in range(4):
    print('평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')


# 학습/예측(Training/Prediction)
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# k-NN 분류기를 생성
#classifier = KNeighborsClassifier(n_neighbors=12)

# 나이브베이즈 분류기를 생성
classifier = GaussianNB() #


# 분류기 학습
classifier.fit(X_train, y_train)

# 예측
y_pred= classifier.predict(X_test)
print(y_pred)

# 작은 이원교차표
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)

# 정밀도 , 재현율, f1 score 확인
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy) # 0.941

```

**문제11.** 위의 와인 데이터 분류의 나이브 베이즈 모델의 정확도는 0.88이었다. 이번에는 knn으로 정확도를 확인하세요!

# ■ [쉬움주의] 유방암 데이터의 악성종양을 knn 으로 분류하기

```

import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

df = pd.read_csv("d:\data\wine.csv")

# DataFrame 확인
print(df.shape) # (178, 14)
print(df.info()) # Type : 라벨컬럼
print(df.describe()) # 요약정보를 나타냄
print(df)

# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = df.iloc[:, 1: ].to_numpy()
y = df['Type'].to_numpy()
#print(y)

print(df.shape) # (178, 14), length
print(len(X)) # 178
print(len(y)) # 178

# 정규화진행
from sklearn import preprocessing

X=preprocessing.StandardScaler().fit(X).transform(X)
#X=preprocessing.MinMaxScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split

# 훈련 데이터 90, 테스트 데이터 10으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1, random_state = 10)

print(X_train.shape) #(160, 13)
print(y_train.shape) # (160, )

# 스케일링(z-score 표준화 수행 결과 확인)
for col in range(4):
    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')

for col in range(4):
    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')

# 학습/예측(Training/Prediction)
from sklearn.neighbors import KNeighborsClassifier
#from sklearn.naive_bayes import GaussianNB

# k-NN 분류기를 생성
classifier = KNeighborsClassifier(n_neighbors=12)

```

```

# 나이브베이즈 분류기를 생성
# classifier = GaussianNB()

# 분류기 학습
classifier.fit(X_train, y_train)

# 예측
y_pred= classifier.predict(X_test)
print(y_pred)

# 작은 이원교차표
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)

# 정밀도 , 재현율, f1 score 확인
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy) # 0.9444

```

## ■ 파이썬 나이브 베이즈 사이킷런 함수 3가지

1. BernoulliNB : 이산형 데이터를 분류할 때 적합
2. GaussianNB : 연속형 데이터를 분류할 때 적합
3. MultinomialNB : ?

## 문제12. 독버섯 데이터를 나이브 베이즈 모델로 분류하기

### 1. R에서

```
mushroom <- read.csv("mushrooms.csv", header=T, stringsAsFactors=TRUE )
```

### 2. Python에서 :

```

df = pd.read_csv( 'd:\data\mushrooms.csv' )

#%%
import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

```

```

df = pd.read_csv('d:\\data\\mushrooms.csv')

df = pd.get_dummies(df)
#print(df.shape) # (8124, 23)
print(df)
print(df.shape) # (8124, 119)

# get_dummies 함수를 이용해서 값의 종류에 따라
# 전부 0 아니면 1로 변환함

# DataFrame 확인
print(df.shape) # (8124, 23)
print(df.info()) # 전부 object (문자)형으로 되어있음
print(df.describe())

# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = df.iloc[:,2: ].to_numpy()
y = df.iloc[:,1].to_numpy()
print(X)
print(y)

print(df.shape) # (8124, 119)
print(len(X)) # 8124
print(len(y)) # 8124

#from sklearn import preprocessing

#X=preprocessing.StandardScaler().fit(X).transform(X)
#X=preprocessing.MinMaxScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split

# 훈련 데이터 75, 테스트 데이터 25으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 10)

print(X_train.shape) # (6093, 22)
print(y_train.shape) # (6093,)

# 스케일링(z-score 표준화 수행 결과 확인)
#for col in range(4):
#    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')

#for col in range(4):
#    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')

# 학습/예측(Training/Pradiction)
#from sklearn.neighbors import KNeighborsClassifier
#from sklearn.naive_bayes import GaussianNB
#from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB

```

```

# k-NN 분류기를 생성
#classifier = KNeighborsClassifier(n_neighbors=12)

# 나이브베이즈 분류기를 생성
classifier = BernoulliNB() #

# 분류기 학습
classifier.fit(X_train, y_train)

# 예측
y_pred= classifier.predict(X_test)
print(y_pred)

# 작은 이원교차표
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)

# 정밀도 , 재현율, f1 score 확인
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy) # 0.9497 MultinomialNB
                # 0.9615 GaussianNB
                # 0.9350 BernoulliNB

```

**문제13. ( 오늘의 마지막 문제 2/22 )** 위에 수행했던 독버섯 분류 나이브 베이즈 모델의 정확도를 아래와 같이 0.99로 만드는 라플라스 값을 알아내시오!

힌트 :

```

# 나이브베이즈 분류기를 생성
classifier = GaussianNB(var_smoothing= ? )

```

```

errors = []
for i in np.arange(0.001, 0.01 , 0.001):
    nb = GaussianNB(var_smoothing=i)
    nb.fit(X_train, y_train)
    pred_i = nb.predict(X_test)
    errors.append(np.mean(pred_i != y_test))
print(errors)

for k, i  in zip(np.arange(0.001, 0.01 , 0.001),errors):
    print (k, '-->', i)

```



```
import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

df = pd.read_csv('d:\data\mushrooms.csv')

df = pd.get_dummies(df)
# print(df.shape) # (8124, 23)
print(df)
print(df.shape) # (8124, 119)

# get_dummies 함수를 이용해서 값의 종류에 따라
# 전부 0 아니면 1로 변환함

# DataFrame 확인
print(df.shape) # (8124, 23)
print(df.info()) # 전부 object (문자)형으로 되어있음
print(df.describe())

# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = df.iloc[:,2: ].to_numpy()
y = df.iloc[:,1].to_numpy()
print(X)
print(y)

print(df.shape) # (8124, 119)
print(len(X)) # 8124
print(len(y)) # 8124

# from sklearn import preprocessing

#X=preprocessing.StandardScaler().fit(X).transform(X)
#X=preprocessing.MinMaxScaler().fit(X).transform(X)

from sklearn.model_selection import train_test_split
```

```
# 훈련 데이터 75, 테스트 데이터 25으로 나눈다.
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 12)
```

```
print(X_train.shape) # (6093, 22)  
print(y_train.shape) # (6093,)
```

```
# 스케일링(z-score 표준화 수행 결과 확인)
```

```
#for col in range(4):  
#    print(f'평균 = {X_train[:, col].mean()}, 표준편차= {X_train[:, col].std()}')  
  
#for col in range(4):  
#    print(f'평균 = {X_test[:, col].mean()}, 표준편차= {X_test[:, col].std()}')
```

```
# 학습/예측(Training/Prediction)
```

```
#from sklearn.neighbors import KNeighborsClassifier  
from sklearn.naive_bayes import GaussianNB  
#from sklearn.naive_bayes import MultinomialNB  
#from sklearn.naive_bayes import BernoulliNB
```

```
# k-NN 분류기를 생성
```

```
classifier = KNeighborsClassifier(n_neighbors=12)
```

```
# 나이브베이즈 분류기를 생성
```

```
classifier = GaussianNB(var_smoothing=0.001) #
```

```
# 분류기 학습
```

```
classifier.fit(X_train, y_train)
```

```
# 예측
```

```
y_pred= classifier.predict(X_test)  
print(y_pred)
```

```
# 작은 이원교차표
```

```
from sklearn.metrics import confusion_matrix  
conf_matrix= confusion_matrix(y_test, y_pred)  
print(conf_matrix)
```

```
# 정밀도 , 재현율, f1 score 확인
```

```
from sklearn.metrics import classification_report  
report = classification_report(y_test, y_pred)  
print(report)
```

```
# 정확도 확인하는 코드
```

```
from sklearn.metrics import accuracy_score  
accuracy = accuracy_score( y_test, y_pred)  
print(accuracy) # 0.9497 MultinomialNB
```

```
# 0.9615 GaussianNB
# 0.9350 BernoulliNB
#%%
import numpy as np
import matplotlib.pyplot as plt

errors = []
for i in np.arange(0.001, 0.01 , 0.001):
    nb = GaussianNB(var_smoothing=i)
    nb.fit(X_train, y_train)
    pred_i = nb.predict(X_test)
    errors.append(np.mean(pred_i != y_test))
print(errors)

for k, i in zip(np.arange(0.001, 0.01 , 0.001),errors):
    print (k, '--->', i)

plt.plot( np.arange(0.001, 0.01, 0.001), errors, marker='o')
plt.title('Mean error with laplace-Value')
plt.xlabel('laplace')
plt.ylabel('mean error')
plt.show()
```

## ■ 파이썬으로 구현 방법

1. knn으로 구현하는 방법
2. 나이브 베이즈로 구현하는 방법

# 결론: 더미 변수 갯수를 항목 갯수-1개로 조정하고, MultinomialNB 를 이용한 laplace 가 아래와 같을때

# 정확도 1로 가장 신뢰있는 모델입니다.

```
# laplace      acc      err  F1_score
# 0.0001  1.000000  0.000000  1.000000
# 0.0002  0.999508  0.000492  0.999506
```

## ### 나이브 베이즈 명목형 변수 분류 ( 2/23 마지막 문제 )

```
import sys
import numpy as np
import pandas as pd
np.set_printoptions(threshold=sys.maxsize)
pd.set_option('display.max_columns', 500)
```

```
df = pd.read_csv('c:\data\mushrooms.csv')
```

# 명목형 갯수 -1 개의 갯수로 더미 변수를 만들어야 정확도가 더 올라갑니다.

```
df = pd.get_dummies(df, drop_first=True) # 더미 변수 생성
```

# 옵션 : drop\_first=True를 사용하게 되면 생성된 더미변수 중에 하나의 컬럼( 첫번째 )을 삭제 합니다.

```
X = df.iloc[:,1:].to_numpy()
y = df.iloc[:,0].to_numpy()
```

```
from sklearn.model_selection import train_test_split
```

# 훈련 데이터 75, 테스트 데이터 25으로 나눈다.

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 10)
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
```

```

from sklearn.naive_bayes import BernoulliNB
import numpy as np
from sklearn.metrics import classification_report

r= np.arange(0.0001, 0.0011 , 0.0001)
acclist = []
err_list = []
f1_list = []
for i in r:
    nb = MultinomialNB(alpha=i)
    # nb = BernoulliNB(alpha=i)
    # nb = GaussianNB(var_smoothing=i)
    nb.fit(X_train, y_train)
    y_pred = nb.predict(X_test)

    report = classification_report(y_test, y_pred, digits=2, output_dict=True)
    f1 = report['macro avg']['f1-score']
    f1_list.append(f1)
    acclist.append(report['accuracy'])
    err_list.append(np.mean(y_pred != y_test))
print(nb)
df_s = pd.DataFrame(data=dict(laplace=r, acc=acclist, err=err_list, F1_score=f1_list))

target = 0.99
df_s2 = (df_s[df_s['acc']>target])
print(df_s2)

import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.subplots_adjust(left=0.125,
                   bottom=0.1,
                   right=1,
                   top=0.9,
                   wspace=0.2,
                   hspace=0.35)

plt.subplot(131) # 추가로 그래프를 그린다.
plt.plot(r, acclist,color='blue', marker='o', markerfacecolor='red')
plt.title('Accuracy', size=15)
plt.xlabel("laplace")
plt.ylabel('Accuracy')

plt.subplot(132)
plt.plot(r, err_list, color='red', marker='o', markerfacecolor='blue')
plt.title('Error', size=15)
plt.xlabel("laplace")
plt.ylabel('error')

plt.subplot(133)
plt.plot(r, f1_list, color='green', marker='o', markerfacecolor='yellow')
plt.title('F1-Score', size=15)
plt.xlabel("laplace")

```

```
plt.ylabel('F1 Score')
```

```
plt.show()
```

## ■ 3장. 의사결정트리

의사결정트리 --> 랜덤포레스트 ( 양상블 + 의사결정트리 )

회귀분석

▣ 독버섯 데이터를 의사결정트리 알고리즘으로 분류하기 ( jupyter notebook 으로 진행 )



```
import pandas as pd # 데이터 전처리를 위해서  
import seaborn as sns # 시각화를 위해서
```

```
df = pd.read_csv('d://data//mushrooms.csv')
```

```
df = pd.get_dummies(df, drop_first = True)  
# print(df.shape) # (8124, 23)  
print(df)  
print(df.shape) # (8124, 119)
```

```
# get_dummies 함수를 이용해서 값의 종류에 따라  
# 전부 0 아니면 1로 변환함
```

```
# DataFrame 확인
print(df.shape) # (8124, 23)
print(df.info()) # 전부 object (문자)형으로 되어있음
print(df.describe())
```

### # 종속변수와 독립변수를 구성하는 작업

```
X = df.iloc[:,1:].to_numpy() #독립변수들
y = df.iloc[:,0].to_numpy() # 종속변수
print(X)
print(y)
```

```
print(df.shape) # (8124, 119)
print(len(X)) # 8124
print(len(y)) # 8124
```

```
from sklearn.model_selection import train_test_split
```

### # 훈련 데이터 75, 테스트 데이터 25으로 나눈다.

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 10)

print(X_train.shape) # (6093, 95)
print(y_train.shape) # (6093,)
```

```
# 학습/예측(Training/Pradiction)
```

```
# sklearn 라이브러리에서 Decision Tree 분류 모형 가져오기
from sklearn import tree
```

### # 의사결정트리 분류기를 생성 (criterion='entropy' 적용)

```
classifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5)
# criterion은 entropy와 gini계수가 있다.
# max_depth는 가지의 깊이를 나타냄/ 너무 깊으면 오버피팅 된다.
```

### # 분류기 학습

```
classifier.fit(X_train, y_train)
```

### # 예측

```
y_pred= classifier.predict(X_test)
print(y_pred)
```

### # 작은 이원교차표

```
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

### # 정밀도 , 재현율, f1 score 확인

```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)
```

#### # 정확도 확인하는 코드

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print(accuracy) # 1.0 Decision tree
# 0.9497 MultinomialNB
# 0.9615 GaussianNB
# 0.9350 BernoulliNB
```

## 문제. iris 데이터를 의사결정트리로 분류하시오 !

(중요한 컬럼 확인 - 정보획득량이 제일 높은것 확인)

```
import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

col_names = ['sepal-length', 'sepal-width','petal-length', 'petal-width','Class']

df = pd.read_csv('d:\data\iris2.csv', encoding='UTF-8', header=None,
names=col_names)

print(df.shape) # (149, 5)
#print(df)

# DataFrame 확인
print(df.info()) # 전부 object (문자)형으로 되어있음
print(df.describe())
print(df)

# 독립변수와 종속변수를 구성하면서 numpy array로 변환
X = df.iloc[:, :-1].to_numpy()
y = df.iloc[:, -1].to_numpy()

print(len(X)) # 149
print(len(y)) # 149
```

#### #최대 최소 값을 0~1로 해준다.

```
from sklearn import preprocessing
```

```
#X=preprocessing.StandardScaler().fit(X).transform(X)
X=preprocessing.MinMaxScaler().fit(X).transform(X)
#print(X)
```

```
from sklearn.model_selection import train_test_split
```

```
# 훈련 데이터 75, 테스트 데이터 25으로 나눈다.  
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 10)  
  
print(X_train.shape) # (111, 4)  
print(y_train.shape) # (111,)
```

```
# 학습/예측(Training/Prediction)  
# sklearn 라이브러리에서 Decision Tree 분류 모형 가져오기  
from sklearn import tree
```

```
# 의사결정트리 분류기를 생성 (criterion='entropy' 적용)  
classifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3)  
#classifier = tree.DecisionTreeClassifier(criterion='gini', max_depth=3)  
# 메뉴얼 : https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
```

## # 분류기 학습

```
classifier.fit(X_train, y_train)
```

```
# 특성 중요도 ( 정보획득량을 이용해서 종속변수에 미치는 중요도를 확인 )  
print(df.columns.values) #컬럼 출력 values를 써줘야 컬럼이 나옴  
print("특성 중요도 : \n{}".format(classifier.feature_importances_))  
# 모델명.feature_importances_ 해주면 각각의 정보 획득량이 나옴.
```

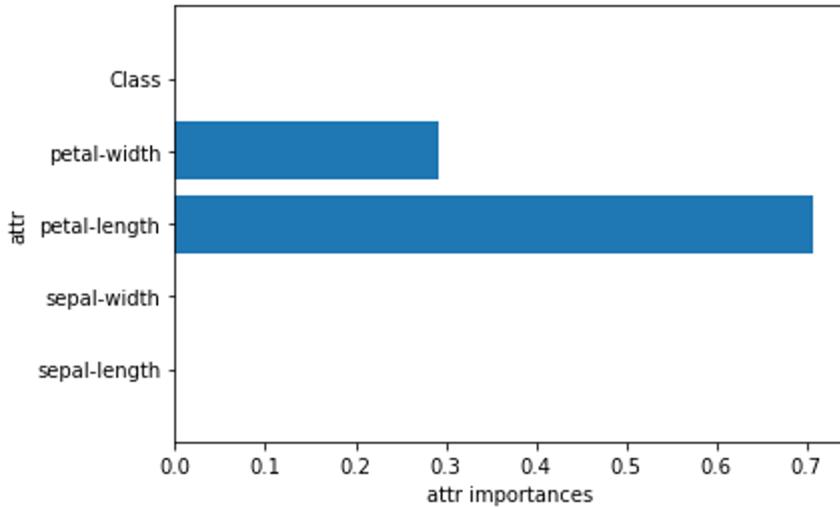
## #시각화

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

```
def plot_feature_importances_cancer(model):  
    n_features = df.shape[1]  
    plt.barh(range(n_features-1),model.feature_importances_, align='center') #막대 그래프를 가  
    로 출력  
    plt.yticks(np.arange(n_features), df.columns.values) # 막대그래프 y축의 눈금값  
    plt.xlabel("attr importances") # x축 이름  
    plt.ylabel("attr") # y축 이름  
    plt.ylim(-1,n_features)
```

# 설명 : n\_features-1은 n\_features가 iris의 컬럼의 개수가 5개인데 그 중 독립변수가 4개이므로 -1을 해 준것이다.

```
plot_feature_importances_cancer(classifier)  
plt.show()
```



```
#%%%
```

**# 테스트 데이터를 모델이 넣고 예측**

```
y_pred= classifier.predict(X_test)
```

**# 작은 이원교차표**

```
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

**# 정밀도 , 재현율, f1 score 확인**

```
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
#print(report)
```

**# 정확도 확인하는 코드**

```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print( accuracy) #0.9736842105263158
```

**# 의사결정트리를 시각화 한다.**

```
# 아나콘다 프롬프트 창 열고 conda install pydotplus 해주기
```

```
import pydotplus #의사결정트리 시각화를 위해 필요
```

```
from sklearn.tree import export_graphviz # 의사결정트리 시각화를 위해 필요
```

```
from IPython.core.display import Image #주피터노트북에서 시각화 된 것 표기
```

```
import matplotlib.pyplot as plt #스파이더에서 시각화 된것을 볼려면 필요
```

**# 그래프 설정**

```
# out_file=None : 결과를 파일로 저장하지 않겠다.
```

```
# filled=True : 상자 채우기
```

```

# rounded=True : 상자모서리 둥그렇게 만들기
# special_characters=True : 상자안에 내용 넣기

dot_data = export_graphviz(classifier, out_file=None,
                           feature_names=df.columns.values[0:4], #독립변수 4개의 컬럼명
                           class_names=classifier.classes_,
                           filled=True, rounded=True,
                           special_characters=True)

```

# 그래프 그리기

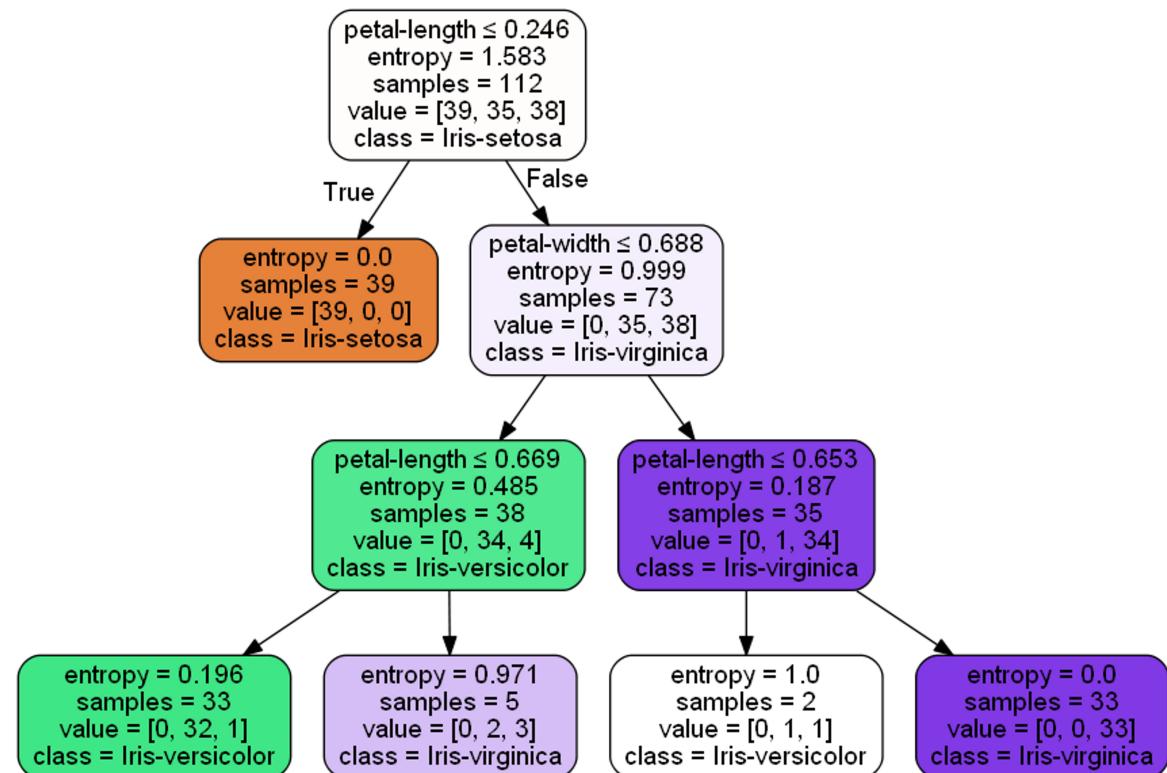
```

dot_data
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

```

# 그래프 해석

#첫번째 줄 : 분류 기준  
#entropy : 엔트로피값  
#sample : 분류한 데이터 개수  
#value : 클래스별 데이터 개수  
#class : 예측한 답



**문제15.** max\_depth를 3이 아니라 4를 주고 다시 모델을 만들고 시각

## 화 하시오!

```
# 학습/예측(Training/Prediction)
# sklearn 라이브러리에서 Decision Tree 분류 모형 가져오기
from sklearn import tree

# 의사결정트리 분류기를 생성 (criterion='entropy' 적용)
classifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4)
#classifier = tree.DecisionTreeClassifier(criterion='gini', max_depth=3)
# 메뉴얼 : https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
```

```
# 분류기 학습
classifier.fit(X_train, y_train)
```

```
# 특성 중요도
print(df.columns.values)
print("특성 중요도 : ".format(classifier.feature_importances_))
```

## 문제16. 화장품 데이터( skin.csv )를 이용해서 의사결정트리 모델을 생성하시오!

```
import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

df = pd.read_csv('d:\data\skin.csv', encoding='UTF-8')

df = pd.get_dummies(df, drop_first=True)
df

# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = df.iloc[:,1:6].to_numpy()
y = df.iloc[:,6].to_numpy()

print(X) # 30
print(y) # 30

print(len(X)) # 30
print(len(y)) # 30

from sklearn.model_selection import train_test_split

# 훈련 데이터 75, 테스트 데이터 25으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 10)

print(X_train.shape) # (22, 5)
```

```
print(y_train.shape) # (22,)

# 학습/예측(Training/Prediction)
# sklearn 라이브러리에서 Decision Tree 분류 모형 가져오기
from sklearn import tree

# 의사결정트리 분류기를 생성 (criterion='entropy' 적용)
classifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3)
#classifier = tree.DecisionTreeClassifier(criterion='gini', max_depth=3)
# 메뉴얼 : https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

# 분류기 학습
classifier.fit(X_train, y_train)

# 특성 중요도
print(df.columns.values[1:6])
print("특성 중요도 : \n".format(classifier.feature_importances_))

df.columns.values[1:6]

df.shape[1]-2

classifier.feature_importances_

import matplotlib.pyplot as plt
import numpy as np

def plot_feature_importances_cancer(model):
    n_features = df.shape[1]-2
    plt.barh(range(n_features),model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), df.columns.values[1:6])
    plt.xlabel("attr importances")
    plt.ylabel("attr")
    plt.ylim(-1,n_features)

plot_feature_importances_cancer(classifier)
plt.show()

y_pred= classifier.predict(X_test)

# 작은 이원교차표
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)

# 정밀도 , 재현율, f1 score 확인
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
```

```

print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)

classifier.classes_

import pydotplus
from sklearn.tree import export_graphviz
from IPython.core.display import Image
import matplotlib.pyplot as plt

# 그래프 설정

# out_file=None : 결과를 파일로 저장하지 않겠다.
# filled=True : 상자 채우기
# rounded=True : 상자모서리 둥그렇게 만들기
# special_characters=True : 상자안에 내용 넣기

dot_data = export_graphviz(classifier, out_file=None,
                           feature_names=df.columns.values[1:6],
                           class_names=['no','yes'],
                           filled=True, rounded=True,
                           special_characters=True)

# 그래프 그리기

dot_data
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

# 그래프 해석

#첫번째 줄 : 분류 기준
#entropy : 엔트로피값
#sample : 분류한 데이터 개수
#value : 클래스별 데이터 개수
#class : 예측한 답

```

**문제17. R을 활용하는 머신러닝에서 사용했던 독일 은행 데이터의 채무 불이행자를 예측하고 의사결정트리를 시각화 하시오!**

credit.csv 이용하기

```
import pandas as pd # 데이터 전처리를 위해서
```

```

import seaborn as sns # 시각화를 위해서

df = pd.read_csv('d:\data\credit.csv', encoding='UTF-8')

df = pd.get_dummies(df, drop_first=True)
df

X = df.iloc[:,0:35]
y = df.iloc[:, -1]
print(X)
print(y)

# X = 전체 행, 마지막 열 제외한 모든 열 데이터 -> n차원 공간의 포인트
X = df.iloc[:,0:35].to_numpy()
y = df.iloc[:, -1].to_numpy()
print(X)
print(y)

from sklearn.model_selection import train_test_split

# 훈련 데이터 75, 테스트 데이터 25으로 나눈다.
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25, random_state = 10)

print(X_train.shape) # (750, 35)
print(y_train.shape) # (750,)

from sklearn import tree

# 의사결정트리 분류기를 생성 (criterion='entropy' 적용)
classifier = tree.DecisionTreeClassifier(criterion='entropy', max_depth=5)
#classifier = tree.DecisionTreeClassifier(criterion='gini', max_depth=3)
# 메뉴얼 : https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

# 분류기 학습
classifier.fit(X_train, y_train)

# 특성 중요도
print(df.columns.values[0:35])
print("특성 중요도 : \n{}".format(classifier.feature_importances_))

import matplotlib.pyplot as plt
import numpy as np

def plot_feature_importances_cancer(model):
    n_features = df.shape[1]-1
    plt.barh(range(n_features),model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), df.columns.values[0:35])
    plt.xlabel("attr importances")
    plt.ylabel("attr")
    plt.ylim(-1,n_features)

```

```

plot_feature_importances_cancer(classifier)
plt.show()

y_pred= classifier.predict(X_test)

# 작은 이원교차표
from sklearn.metrics import confusion_matrix
conf_matrix= confusion_matrix(y_test, y_pred)
print(conf_matrix)

# 정밀도 , 재현율, f1 score 확인
from sklearn.metrics import classification_report
report = classification_report(y_test, y_pred)
print(report)

# 정확도 확인하는 코드
from sklearn.metrics import accuracy_score
accuracy = accuracy_score( y_test, y_pred)
print( accuracy)

```

**문제18.** 위의 의사결정트리의 모델을 의사결정트리 + 앙상블 기법을 적용한 랜덤포레스트를 구현하시오!

## [쉬움주의] 파이썬으로 단순회귀분석

```

# -*- coding: utf-8 -*-
### 기본 라이브러리 불러오기
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
"""

[Step 1] 데이터 준비 - read_csv() 함수로 자동차 연비 데이터셋 가져오기
"""

# CSV 파일을 데이터프레임으로 변환
df = pd.read_csv('D:\data\auto-mpg.csv', header=None)

# 열 이름 지정
df.columns = ['mpg','cylinders','displacement','horsepower','weight',
'acceleration','model year','origin','name']

# 데이터 살펴보기
print(df.head())
print('\n')

```

```

# IPython 디스플레이 설정 - 출력할 열의 개수 한도 늘리기
pd.set_option('display.max_columns', 10)
print(df.head())
print('₩n')
"""

[Step 2] 데이터 탐색
"""

# 데이터 자료형 확인
print(df.info())
print('₩n')

# 설명 : horsepower가 object여서 수치형 데이터로 변경해야 한다.
# 전처리가 필요한 문자가 포함되어 있어서 object 문자형으로 출력되고 있다.

# 데이터 통계 요약정보 확인
print(df.describe())
print('₩n')

# 설명 : horsepower가 빠졌음을 주의하기. 통계요약정보를 출력하려면 숫자형 데이터여야 함

#mpg는 mile per gallon의 약자로 영국과 미국에서는 한국과는 달리 갤런당 마일
#단위로 연비를 표시한다. 한국은 리터당 킬로미터( km/L )단위로 표시한다.
# mpg 열을 한국에서 사용하는 km/L로 변환해줘야 한다.
# 1갤런은 3.78541이다. 그리고 1마일이 1.60934km이다.
# #1mpg( mile per gallon)은 ?
# print( 1.60934/3.78541 ) #0.425 km/L

# horsepower 열의 자료형 변경 (문자열 ->숫자)
print(df['horsepower'].unique()) # horsepower 열의 고유값 확인
print('₩n')
df['horsepower'].replace('?', np.nan, inplace=True) # '?'을 np.nan( 결측치 )으로 변경
df.dropna(subset=['horsepower'], axis=0, inplace=True) # 누락데이터 행을 삭제
df['horsepower'] = df['horsepower'].astype('float') # 문자열을 실수형으로 변환
print(df.describe()) # 데이터 통계 요약정보 확인
print('₩n')
"""

[Step 3] 속성(feature 또는 variable) 선택
"""

# 분석에 활용할 열(속성)을 선택 (연비, 실린더, 출력, 중량)
ndf = df[['mpg', 'cylinders', 'horsepower', 'weight']]
print(ndf.head())
print('₩n')

### 종속 변수 Y인 "연비(mpg)"와 다른 변수 간의 선형관계를 그래프(산점도)로 확인
# Matplotlib으로 산점도 그리기

```

```
ndf.plot(kind='scatter', x='weight', y='mpg', c='coral', s=10, figsize=(10, 5))  
plt.show()  
plt.close()
```

그래프 :



#### # seaborn으로 산점도 그리기

```
fig = plt.figure(figsize=(10, 5)) # 전체 그림판 가로 10, 세로 5로 잡아주고  
ax1 = fig.add_subplot(1, 2, 1) # 첫번째 그림판 영역  
ax2 = fig.add_subplot(1, 2, 2) # 두번째 그림판 영역  
sns.regplot(x='weight', y='mpg', data=ndf, ax=ax1) # 회귀선 표시  
sns.regplot(x='weight', y='mpg', data=ndf, ax=ax2, fit_reg=False) # 회귀선 미표시  
plt.show()  
plt.close()
```

#### # seaborn 조인트 그래프 - 산점도, 히스토그램

```
sns.jointplot(x='weight', y='mpg', data=ndf) # 회귀선 없음  
sns.jointplot(x='weight', y='mpg', kind='reg', data=ndf) # 회귀선 표시  
plt.show()  
plt.close()
```

#### # seaborn pairplot으로 두 변수 간의 모든 경우의 수 그리기

```
sns.pairplot(ndf)  
plt.show()  
plt.close()  
..."
```

#### Step 4: 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

..."

#### # 속성(변수) 선택

```
X=ndf[['weight']] # 독립 변수 X  
y=ndf['mpg'] # 종속 변수 Y
```

#### # train data 와 test data로 구분(7:3 비율)

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, #독립 변수  
y, #종속 변수  
test_size=0.3, #검증 30%  
random_state=10) #랜덤 추출 값  
  
print('train data 개수: ', len(X_train))  
print('test data 개수: ', len(X_test))  
  
'''
```

### Step 5: 단순회귀분석 모형 - sklearn 사용

```
'''  
# sklearn 라이브러리에서 선형회귀분석 모듈 가져오기
```

```
from sklearn.linear_model import LinearRegression
```

```
# 단순회귀분석 모형 객체 생성
```

```
lr = LinearRegression()
```

```
# train data를 가지고 모형 학습
```

```
# R은 lm( data ~. data= data ) 이런식
```

```
lr.fit(X_train, y_train)
```

```
# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산
```

```
r_square = lr.score(X_test, y_test)
```

```
print(r_square)
```

```
print('₩n')
```

```
# 회귀식의 기울기
```

```
print('기울기 a: ', lr.coef_)
```

```
print('₩n')
```

```
# 회귀식의 y절편
```

```
print('y절편 b', lr.intercept_)
```

```
print('₩n')
```

```
# 모형에 전체 X 데이터를 입력하여 예측한 값 y_hat을 실제 값 y와 비교
```

```
y_hat = lr.predict(X)
```

```
plt.figure(figsize=(10, 5))
```

```
ax1 = sns.distplot(y, hist=False, label="y")
```

```
ax2 = sns.distplot(y_hat, hist=False, label="y_hat", ax=ax1)
```

```
plt.show()
```

```
plt.close()
```



( 오늘의 마지막 문제 2/23 ) 체중과 키 데이터를 이용해서 선형 회귀분석을 하고 seaborn 그래프로 시각화 하시오!

```
weight=[ 72, 72, 70, 43, 48, 54, 51, 52, 73, 45, 60, 62, 64, 47, 51, 74, 88, 64, 56, 56 ]  
tall = [ 176, 172, 182, 160, 163, 165, 168, 163, 182, 148, 170, 166, 172, 169, 163, 170, 182,  
174, 164, 160 ]
```

```
dict_data = { 'weight' : [ 72, 72, 70, 43, 48, 54, 51, 52, 73, 45, 60, 62, 64, 47, 51, 74, 88, 64,  
56, 56 ],  
            'tall' : [ 176, 172, 182, 160, 163, 165, 168, 163, 182, 148, 170, 166, 172, 169,  
163, 170, 182, 174, 164, 160 ] }
```

```
df = pd.DataFrame(dict_data)  
print (df)
```

답 :

# 필요한 패키지 불러오기

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns
```

# step1. 데이터 프레임으로 변환시키기

```
weight=[ 72, 72, 70, 43, 48, 54, 51, 52, 73, 45, 60, 62, 64, 47, 51, 74, 88, 64, 56, 56 ]  
tall = [ 176, 172, 182, 160, 163, 165, 168, 163, 182, 148, 170, 166, 172, 169, 163, 170, 182,  
174, 164, 160 ]
```

```
dict_data = { 'weight' : [ 72, 72, 70, 43, 48, 54, 51, 52, 73, 45, 60, 62, 64, 47, 51, 74, 88, 64,  
56, 56 ],  
            'tall' : [ 176, 172, 182, 160, 163, 165, 168, 163, 182, 148, 170, 166, 172, 169,
```

```
163, 170, 182, 174, 164, 160 ] }
```

```
df = pd.DataFrame(dict_data)
print(df)
```

### #Step 2: 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

```
# 속성(변수) 선택
X=df[['weight']] #독립 변수 X
y=df['tall'] #종속 변수 Y

# train data 와 test data로 구분(9:1 비율)
# 데이터 양이 적기 때문에 9대 1로 나눠줌
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, #독립 변수
                                                    y, #종속 변수
                                                    test_size=0.1, #검증 10%
                                                    random_state=10) #랜덤 추출 값

print('train data 개수: ', len(X_train)) #18개
print('test data 개수: ', len(X_test)) #2개
```

### #Step 3: 단순회귀분석 모형 - sklearn 사용

```
# sklearn 라이브러리에서 선형회귀분석 모듈 가져오기
from sklearn.linear_model import LinearRegression

# 단순회귀분석 모형 객체 생성
lr = LinearRegression()

# train data를 가지고 모형 학습
lr.fit(X_train, y_train)

# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산
r_square = lr.score(X_test, y_test)
print(r_square) #0.8657609246754262
```

```
# 회귀식의 기울기
print('기울기 a: ', lr.coef_) #기울기 a: [0.57510584]
```

```
# 회귀식의 y절편
print('y절편 b', lr.intercept_) #133.84081287044876
```

```
# 모형에 전체 X 데이터를 입력하여 예측한 값 y_hat을 실제 값 y와 비교  
y_hat = lr.predict(X)  
plt.figure(figsize=(10, 5))  
ax1 = sns.distplot(y, hist=False, label="y")  
ax2 = sns.distplot(y_hat, hist=False, label="y_hat", ax=ax1)  
plt.show()  
plt.close()
```



#### #Step 4: seaborn으로 시각화

```
# seaborn 조인트 그래프 - 산점도, 히스토그램  
sns.jointplot(x='weight', y='tall', kind='reg', data=df) # 회귀선 표시  
plt.show()  
plt.close()
```



✗

## < 복습 >

1. 파이썬으로 knn 구현하기
2. 파이썬으로 나이브베이즈 구현하기
3. 파이썬으로 decision tree 구현하기
4. 파이썬으로 regression 구현하기( 단순회귀분석, 다중회귀분석 )

## ■ 머신러닝 데이터 분석 5가지 단계 ( 큰 그림 생각하기 )

1. 데이터 수집과 설명 : pandas 사용
2. 데이터 탐색 및 시각화 : pandas, matplotlib, seaborn 사용
3. 머신러닝 모델 훈련 : sklearn 사용
4. 머신러닝 모델 평가 : pandas 사용
5. 머신러닝 모델 성능개선 : pandas 사용 ( 파생변수 생성 )

#성능개선 방법 : 단순 회귀를 다항회귀로 변경해서 성능을 올린다.

1. 단순회귀 : 독립변수 한개에 종속변수 한개( 선형회귀선 )
2. 다항회귀 : 독립변수 한개에 종속변수 한개( 비선형회귀선 )
3. 다중회귀 : 종속변수에 영향을 주는 독립변수가 여러개인 경우

문제. 첫번째 예제 단순회귀분석의 결과 그래프(무게와 연비간의 예측값과 실제값의 비교)를 보면 실제값은 왼쪽으로 편향되어있고 예측값을 반대로 오른쪽으로 편중되는 경향을 보인다. 따라서 독립변수(weight) 과 종속변수(mpg) 사이의 선형관계가 있지만, 모형의 오차를 더 줄일 필요가 있어 보인다. 앞에서 본 산포도를 보면 직선보다는 곡선이 더 적합해 보인다. 비선형 회귀분석을 통해 모형의 정확도를 더 높이시오.

답:

```

### 기본 라이브러리 불러오기
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

...
[Step 1 ~ 4] 데이터 준비
...
# CSV 파일을 데이터프레임으로 변환
df = pd.read_csv('d:\data\auto-mpg.csv', header=None)

# 열 이름 지정
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
              'acceleration', 'model year', 'origin', 'name']

# horsepower 열의 자료형 변경 (문자열 ->숫자)
df['horsepower'].replace('?', np.nan, inplace=True)      # '?'을 np.nan으로 변경
df.dropna(subset=['horsepower'], axis=0, inplace=True)  # 누락데이터 행을 삭제
df['horsepower'] = df['horsepower'].astype('float')     # 문자열을 실수형으로 변환

# 분석에 활용할 열(속성)을 선택 (연비, 실린더, 출력, 중량)
ndf = df[['mpg', 'cylinders', 'horsepower', 'weight']]

# ndf 데이터를 train data 와 test data로 구분(7:3 비율)
X=ndf[['weight']] #독립 변수 X
y=ndf['mpg']      #종속 변수 Y

# train data 와 test data로 구분(7:3 비율)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('훈련 데이터: ', X_train.shape)
print('검증 데이터: ', X_test.shape)
print('n')

...
Step 5: 비선형회귀분석 모형 - sklearn 사용
...

# sklearn 라이브러리에서 필요한 모듈 가져오기
from sklearn.linear_model import LinearRegression      #선형회귀분석
from sklearn.preprocessing import PolynomialFeatures   #다항식 변환

# 다항식 변환

```

```

poly = PolynomialFeatures(degree=2)      #2차항 적용/ 차 무게, 연비
X_train_poly=poly.fit_transform(X_train)  #X_train 데이터( 차 무게 값 )를 2차항으로 변형

print('원 데이터: ', X_train.shape)
print('2차항 변환 데이터: ', X_train_poly.shape)
print('Wn')

#2차항으로 변환시킨 데이터를 학습시키기 위한 회귀 모델을 생성하고 학습 시킴
pr = LinearRegression()
pr.fit(X_train_poly, y_train)

# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산
X_test_poly = poly.fit_transform(X_test)    #X_test 데이터를 2차항으로 변형
r_square = pr.score(X_test_poly,y_test)     # y_test = 라벨/ pr: 선형회귀
print(r_square)
print('Wn')

# train data의 산점도와 test data로 예측한 회귀선을 그래프로 출력
y_hat_test = pr.predict(X_test_poly)

fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(X_train, y_train, 'o', label='Train Data') # 데이터 분포
ax.plot(X_test, y_hat_test, 'r+', label='Predicted Value') # 모형이 학습한 회귀선, r+ = redplus
ax.legend(loc='best')
plt.xlabel('weight')
plt.ylabel('mpg')
plt.show()
plt.close()

# 모형에 전체 X 데이터를 입력하여 예측한 값 y_hat을 실제 값 y와 비교
X_ploy = poly.fit_transform(X)
y_hat = pr.predict(X_ploy)

plt.figure(figsize=(10, 5))
ax1 = sns.distplot(y, hist=False, label="y")
ax2 = sns.distplot(y_hat, hist=False, label="y_hat", ax=ax1)
plt.show()
plt.close()

```

**문제19.** 어제 마지막 문제로 풀었던 체중과 키와의 단순 선형 회귀분석 결과의 테스트 데이터에 대한 결정계수는 0.86이었다. 그렇다면 이번에는 다항회귀로 비선형 회귀선을 만들어 성능을 더 올리세요.

단순회귀의 결정계수 : #0.8657609246754262

다항회귀의 결정계수 : #0.9263965046109321

### # 필요한 패키지 불러오기

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns
```

### # step1. 데이터 프레임으로 변환시키기

```
weight=[ 72, 72, 70, 43, 48, 54, 51, 52, 73, 45, 60, 62, 64, 47, 51, 74, 88, 64, 56, 56 ]  
tall = [ 176, 172, 182, 160, 163, 165, 168, 163, 182, 148, 170, 166, 172, 169, 163, 170, 182, 174,  
164, 160 ]  
  
dict_data = { 'weight' : [ 72, 72, 70, 43, 48, 54, 51, 52, 73, 45, 60, 62, 64, 47, 51, 74, 88, 64, 56,  
56 ],  
             'tall' : [ 176, 172, 182, 160, 163, 165, 168, 163, 182, 148, 170, 166, 172, 169, 163, 170, 182, 174,  
164, 160 ] }  
  
df = pd.DataFrame(dict_data)  
print (df)
```

### #Step 2: 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

#### # 속성(변수) 선택

```
X=df[['weight']] #독립 변수 X
```

```
y=df['tall'] #종속 변수 Y
```

```
# train data 와 test data로 구분(9:1 비율)  
# 데이터 양이 적기 때문에 9대 1로 나눠줌  
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, #독립 변수  
y, #종속 변수  
test_size=0.1, #검증 10%  
random_state=10) #랜덤 추출 값  
  
print('train data 개수: ', len(X_train)) #18개  
print('test data 개수: ', len(X_test)) #2개
```

### #Step 3: 단순회귀분석 모형 - sklearn 사용

```
# sklearn 라이브러리에서 선형회귀분석 모듈 가져오기  
from sklearn.linear_model import LinearRegression
```

```
# 단순회귀분석 모형 객체 생성
```

```
lr = LinearRegression()
```

```
# train data를 가지고 모형 학습  
lr.fit(X_train, y_train)  
  
# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산  
r_square = lr.score(X_test, y_test)  
print(r_square) #0.8657609246754262
```

### # 회귀식의 기울기

```
print('기울기 a: ', lr.coef_) #기울기 a: [0.57510584]
```

### # 회귀식의 y절편

```
print('y절편 b', lr.intercept_) #133.84081287044876
```

```
# 모형에 전체 X 데이터를 입력하여 예측한 값 y_hat을 실제 값 y와 비교  
y_hat = lr.predict(X)  
plt.figure(figsize=(10, 5))  
ax1 = sns.distplot(y, hist=False, label="y")  
ax2 = sns.distplot(y_hat, hist=False, label="y_hat", ax=ax1)  
plt.show()  
plt.close()
```

```
# # 다항 회귀로 진행했을때
```

```
# In[2]:
```

```
...
```

### Step 4: 비선형회귀분석 모형 - sklearn 사용 # 이부분부터 추가

```
...
```

```
# sklearn 라이브러리에서 필요한 모듈 가져오기  
from sklearn.linear_model import LinearRegression      #선형회귀분석  
from sklearn.preprocessing import PolynomialFeatures   #다항식 변환  
  
# 다항식 변환  
poly = PolynomialFeatures(degree=2)                  #2차항 적용  
X_train_poly=poly.fit_transform(X_train)    #X_train 데이터를 2차항으로 변형  
  
print('원 데이터: ', X_train.shape)  
print('2차항 변환 데이터: ', X_train_poly.shape)  
print('₩n')  
  
# train data를 가지고 모형 학습  
pr = LinearRegression()  
pr.fit(X_train_poly, y_train)
```

```

# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산
X_test_poly = poly.fit_transform(X_test)      #X_test 데이터를 2차항으로 변형
r_square = pr.score(X_test_poly,y_test)
print(r_square)
print('n')

# train data의 산점도와 test data로 예측한 회귀선을 그래프로 출력
y_hat_test = pr.predict(X_test_poly)

fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(1, 1, 1)
ax.plot(X_train, y_train, 'o', label='Train Data') # 데이터 분포
ax.plot(X_test, y_hat_test, 'r+', label='Predicted Value') # 모형이 학습한 회귀선
ax.legend(loc='best')
plt.xlabel('weight')
plt.ylabel('mpg')
plt.show()
plt.close()

# 모형에 전체 X 데이터를 입력하여 예측한 값 y_hat을 실제 값 y와 비교
X_ploy = poly.fit_transform(X)
y_hat = pr.predict(X_ploy)

plt.figure(figsize=(10, 5))
ax1 = sns.distplot(y, hist=False, label="y")
ax2 = sns.distplot(y_hat, hist=False, label="y_hat", ax=ax1)
plt.show()
plt.close()

```

## ■ 6. Multi Regression ( 다중회귀 )

예제1. 미국 우주 왕복선 폭파원인

예제2. 미국 대학교 입학점수에 영향을 미치는 과목 분석

예제3. 미국 국민 의료비 지출

### ■ 예제1. 미국 우주 왕복선 폭파원인

O형링의 손상이 온도, 압력, 비행기 번호 이 3가지 중에서 어떤게 젤 영향이 큰지?

```

import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd

```

```
df = pd.read_csv( "d:\data\challenger.csv", engine='python', encoding='CP949' )
```

분석결과 설명 : o형링 파손에 영향을 주는 가장 큰 독립변수는 온도이다.

그 다음이 비행기 노후화를 나타내는 비행기 번호이다.

회귀식 :  $3.5271 - 0.0514 * x_1 + 0.0018 * x_2 + 0.0143 * x_3$

**문제20. statsmodels 패키지를 이용해서 방금 다중회귀 분석을 해보았는데 이번에는 중요한 독립변수인 temperature만 이용해서 단순회귀 분석을 진행하고 분석된 결과를 출력하세요!**

종속변수 : distress\_ct( o형링 파손 수 )

독립변수 : temperature( 온도 )

답 :

```
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
```

```
df = pd.read_csv( "d:\data\challenger.csv", engine='python', encoding='CP949' )
print(df)
```

**## 온도와 o형링 손상에 대한 단순회귀**

```
model = smf.ols( formula='distress_ct ~ temperature', data=df)
result = model.fit()
print(result.summary())
```

결과 :

OLS Regression Results

```
=====
Dep. Variable: distress_ct R-squared: 0.261
Model: OLS Adj. R-squared: 0.226
Method: Least Squares F-statistic: 7.426
Date: Wed, 24 Feb 2021 Prob (F-statistic): 0.0127
Time: 11:15:23 Log-Likelihood: -18.959
No. Observations: 23 AIC: 41.92
Df Residuals: 21 BIC: 44.19
Df Model: 1
Covariance Type: nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	3.6984	1.220	3.033	0.006	1.162	6.235

```

temperature -0.0475    0.017    -2.725    0.013    -0.084    -0.011
=====
=====

Omnibus:          18.542 Durbin-Watson:        2.116
Prob(Omnibus):    0.000 Jarque-Bera (JB):      21.747
Skew:              1.733 Prob(JB):            1.90e-05
Kurtosis:         6.267 Cond. No.           708.
=====
```

### 분석 결과:

Intercept 3.6984  
temperature -0.0475

$$y = 3.6984 - 0.0475 * x_1 \text{ (회귀식)}$$

↑                   ↑  
파손수              온도

2.21개	31도 F(화씨)
0.82개	60도 F(화씨)
0.34개	70도 F(화씨)

분석결과 : 30도에서 발사하는게 60도에서 발사하는 것 보다 3배 더 위험하고 화씨 70도에서 발사한 것 보다 7~8배 위험하다.

### ■ 예제2. 미국 대학교 입학점수에 영향을 미치는 과목 분석

데이터 : sports.csv

종속변수 : acceptance

독립변수 : academic  
sports  
music

설명 : 학과점수, 체육점수, 음악점수 중에 어떤게 더 입학하는데 더 중요한 독립변수인지?

**분석요청** : 학과점수, 체육점수, 음악점수 중에 어떤게 더 입학하는데 더 중요한 것인지 ?

3과목의 점수의 단위가 과목마다 다르다. (예: 체중과 키처럼)

그래서 이런 경우에는 표준화를 하고 회귀분석을 해야한다.

#### 1. 표준화를 안했을 때 :

## # 1. 데이터 불러오기

```
import numpy as np
import statsmodels.api as sm #회귀분석을 위해 필요
import statsmodels.formula.api as smf #회귀분석을 위해 필요
import pandas as pd
from sklearn.preprocessing import StandardScaler # 표준화를 위해 필요

df = pd.read_csv("d:\data\sports.csv", engine='python', encoding='CP949')
df.columns = ['stud_id', 'academic', 'sports', 'music', 'acceptance'] # 컬럼명을 지정
df
```

## # 2. 모델 생성하기

```
model = smf.ols( formula='acceptance ~ academic + sports + music', data=df)
result = model.fit() # 모델을 훈련시킴
print(result.summary())
```

결과 :

```
OLS Regression Results
=====
Dep. Variable: acceptance R-squared: 0.907
Model: OLS Adj. R-squared: 0.905
Method: Least Squares F-statistic: 634.8
Date: Wed, 24 Feb 2021 Prob (F-statistic): 1.21e-100
Time: 11:32:01 Log-Likelihood: -638.38
No. Observations: 200 AIC: 1285.
Df Residuals: 196 BIC: 1298.
Df Model: 3
Covariance Type: nonrobust
=====
            coef  std err      t    P>|t|    [0.025    0.975]
-----
Intercept  11.4903   1.053  10.916   0.000    9.414  13.566
academic   0.1558   0.006  26.877   0.000    0.144  0.167
sports     0.5727   0.040  14.430   0.000    0.494  0.651
music      0.1046   0.023   4.465   0.000    0.058  0.151
=====
Omnibus: 59.122 Durbin-Watson: 2.091
Prob(Omnibus): 0.000 Jarque-Bera (JB): 133.045
Skew: -1.353 Prob(JB): 1.29e-29
Kurtosis: 5.940 Cond. No. 442.
=====
```

분석결과 : 표준화를 안 했을땐느 체육점수가 학과점수보다 더 영향력이 컸다.

## 2. 표준화를 했을 때 :

## # 1. 데이터 불러오기

```
import numpy as np
import statsmodels.api as sm #회귀분석을 위해 필요
import statsmodels.formula.api as smf #회귀분석을 위해 필요
import pandas as pd
from sklearn.preprocessing import StandardScaler # 표준화를 위해 필요

df = pd.read_csv("d:\data\sports.csv", engine='python', encoding='CP949')
```

## # 2. 표준화하기

```
scaler = StandardScaler()
scaler.fit(df) # 표준화를 위해 df 데이터를 살펴본다.
df_scale = scaler.transform(df) #표준화 작업 수행하고 df를 구성
df_scale
```

## # 3. 판다스 데이터 프레임으로 구성하기

```
df_scale2 = pd.DataFrame(df_scale)
df_scale2.head
```

## # 4. 컬럼 구성하기

```
df_scale2.columns = ['stud_id', 'academic', 'sports', 'music', 'acceptance'] # 컬럼명을 지정
df_scale2.head
```

## #5. 회귀모델 생성하고 summary 결과를 본다.

```
model = smf.ols( formula='acceptance ~ academic + sports + music', data=df_scale2)
result = model.fit() # 모델을 훈련시킴
print(result.summary())
```

결과 :

### OLS Regression Results

```
=====
Dep. Variable: acceptance R-squared:          0.907
Model:           OLS   Adj. R-squared:        0.905
Method:          Least Squares F-statistic:    634.8
Date:    Wed, 24 Feb 2021 Prob (F-statistic): 1.21e-100
Time:      11:46:03 Log-Likelihood:       -46.609
No. Observations:      200   AIC:             101.2
Df Residuals:         196   BIC:            114.4
Df Model:                 3
Covariance Type:    nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	1.388e-17	0.022	6.36e-16	1.000	-0.043	0.043
academic	0.6921	0.026	26.877	0.000	0.641	0.743

```

sports      0.4400    0.030   14.430    0.000    0.380    0.500
music       0.1511    0.034   4.465     0.000    0.084    0.218
=====
Omnibus:          59.122 Durbin-Watson:        2.091
Prob(Omnibus):    0.000 Jarque-Bera (JB):    133.045
Skew:             -1.353 Prob(JB):           1.29e-29
Kurtosis:         5.940 Cond. No.            2.74
=====
```

분석결과 : 학과점수가 체육점수보다 더 높은 영향력을 가진 변수로 나타난다.

**문제21. ( 점심시간 문제 ) R을 활용한 머신러닝 수업 때 회귀분석할 때 사용했던 미국 의료비 데이터( insurance.csv )를 가지고 다중회귀분석을 하시오!**  
**회귀분석한 결과를 출력하시오!**

결정계수가 출력되는 결과 화면을 첨부해서 올리세요!

독립변수 : age, sex, bmi, children, smoker, region

종속변수 : expenses

### # 1. 데이터 불러오기

```

import numpy as np
import statsmodels.api as sm #회귀분석을 위해 필요
import statsmodels.formula.api as smf #회귀분석을 위해 필요
import pandas as pd
from sklearn.preprocessing import StandardScaler # 표준화를 위해 필요
```

```

df = pd.read_csv("d:\data\insurance.csv", engine='python', encoding='CP949')
df
```

### # 2. 모델생성

```

model = smf.ols( formula='expenses ~ age + sex + bmi + children + smoker + region',
data=df)
result = model.fit() # 모델을 훈련시킴
print(result.summary())
```

OLS Regression Results									
Dep. Variable:	expenses	R-squared:	0.751						
Model:	OLS	Adj. R-squared:	0.749						
Method:	Least Squares	F-statistic:	500.9						
Date:	Wed, 24 Feb 2021	Prob (F-statistic):	0.00						
Time:	12:06:13	Log-Likelihood:	-13548.						
No. Observations:	1338	AIC:	2.711e+04						
Df Residuals:	1329	BIC:	2.716e+04						
Df Model:	8								
Covariance Type:	nonrobust								
	coef	std err	t	P> t	[0.025	0.975]			
Intercept	-1.194e+04	987.811	-12.089	0.000	-1.39e+04	-1e+04			
sex[T.male]	-131.3520	332.935	-0.395	0.693	-784.488	521.784			
smoker[T.yes]	2.385e+04	413.139	57.723	0.000	2.3e+04	2.47e+04			
region[T.northwest]	-352.7901	476.261	-0.741	0.459	-1287.095	581.515			
region[T.southeast]	-1035.5957	478.681	-2.163	0.031	-1974.648	-96.544			
region[T.southwest]	-959.3058	477.912	-2.007	0.045	-1896.850	-21.762			
age	256.8392	11.899	21.586	0.000	233.497	280.181			
bmi	339.2899	28.598	11.864	0.000	283.187	395.393			
children	475.6889	137.800	3.452	0.001	205.360	746.017			
Omnibus:	300.499	Durbin-Watson:	2.088						
Prob(Omnibus):	0.000	Jarque-Bera (JB):	719.382						
Skew:	1.212	Prob(JB):	6.14e-157						
Kurtosis:	5.652	Cond. No.	311.						

### 분석결과 :

sex[T.male] -131.3520 : 남성은 여성에 비해 매년 의료비가 131달러 적게들거나 예상.

smoker[T.yes] 2.385e+04 : 흡연자는 비흡연자보다 매년 의료비가  $2.385 \times 10^4 = 23,850$  달러 비용이 더 든다.

age 256.8392 : 나이가 일년씩 더해질 때마다 평균적으로 의료비가 256달러 더 든다.

bmi 339.2899 : 비만지수가 증가할 때마다 339달러 더 들거나 예상.

children 475.6889 : 부양가족 한명이 더 늘어날 때마다 연간의료비가 475달러 더 든다.

지역별로는 북동지역이 북서, 남동, 남서에 비해 의료비가 더 든다.

**문제22. 비만인 사람은 의료비가 더 지출이 되는지 파생변수를 추가해서 확인하시오! bmi30이라는 파생변수를 추가하는데 bmi가 30이상이면 1, 아니면 0이라고 해서 컬럼을 하나 만드시오!**

1. R에서는 `df$bmi <- ifelse( bmi >= 30, 1, 0 )`
2. 파이썬에서는 `df[ 'bmi30' ] = df[ 'bmi' ].apply( 함수 )`

\* 파이썬 함수를 생성하는데 입력값이 30 이상이면 1이고 아니면 0을 출력하는 함수를 `func_1`이라는 이름으로 생성하시오!

```
def func_1(x):
    if x >= 30:
```

```

        return 1
    else:
        return 0

df[ 'bmi30' ] = df[ 'bmi' ].apply( func_1 )

```

### 문제23. 비만인 사람(bmi30)을 분류하는 파생변수를 추가했으면 결정계수가 올라가는지 확인하시오!

```

model = smf.ols( formula='expenses ~ age + sex + bmi + children + smoker + region +
bmi30', data=df)
result = model.fit() # 모델을 훈련시킴
print(result.summary())

```

분석결과 :

기존 0.751에서 0.756으로 올라갔다.

### 문제24. 비만이면서 흡연까지하면 의료비가 더 올라가는지 확인하시오!

```

model = smf.ols( formula='expenses ~ age + sex + bmi + children + smoker + region +
bmi30 + bmi30*smoker', data=df)
result = model.fit() # 모델을 훈련시킴
print(result.summary())

```

결과 :

OLS Regression Results

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-4740.6822	960.019	-4.938	0.000	-6624.002	-2857.362
sex[T.male]	-491.1295	246.568	-1.992	0.047	-974.834	-7.425
smoker[T.yes]	1.34e+04	443.918	30.191	0.000	1.25e+04	1.43e+04
region[T.northwest]	-266.7997	352.417	-0.757	0.449	-958.154	424.555
region[T.southeast]	-824.5730	354.812	-2.324	0.020	-1520.627	-128.519
region[T.southwest]	-1223.8697	353.685	-3.460	0.001	-1917.713	-530.026
age	263.2428	8.805	29.896	0.000	245.969	280.517

```

bmi           114.8282   34.574    3.321    0.001    47.003   182.653
children      520.4738   101.959    5.105    0.000    320.455   720.493
bmi30        -863.2546   425.900   -2.027    0.043   -1698.765  -27.744
bmi30:smoker[T.yes] 1.979e+04   610.110    32.444    0.000    1.86e+04   2.1e+04
=====
Omnibus:          871.061 Durbin-Watson:       2.060
Prob(Omnibus):    0.000 Jarque-Bera (JB):     7318.497
Skew:             3.087 Prob(JB):            0.00
Kurtosis:         12.652 Cond. No.:          420.
=====
```

설명 : 비만이면서 흡연까지 하게 되면 연간 의료비가 19,790 달러가 더 들거라 예상이 된다.

## ■ 다중공선성 확인을 파이썬으로 구현하기

회귀분석에서 사용된 모형의 일부 독립변수가 다른 독립변수와의 상관정도가 아주 높아서 회귀분석 결과에 부정적 영향을 미치는 현상을 다중공선성이라 한다.

두 독립변수들끼리 서로에게 영향을 주고 있다면 둘 중 하나의 영향력을 검증할 때 다른 하나의 영향력이 약해진다.

예 : 학업성취도, 일평균음주량, 혈중 알코올 농도  
( 종속변수 )

팽창계수가 보통은 10보다 큰 것을 골라내고 까다롭게 하려면 5보다 큰것을 골라낸다( 지운다 ).

일평균음주량, 혈중 알코올 농도 둘다 팽창계수가 높게 나온다면 둘 중에 하나를 빼고 아래와 같이

학업 성취도, 일평균 음주량 ---> 회귀분석  
학업성취도, 혈중 알코올 농도 ---> 회귀분석

예제 :

`crab.csv` : 게의 크기, 무게 등에 대한 데이터로 종속변수가 y 컬럼인데 0과 1로 분류하는 데이터이다.

### # 1. 데이터 불러오기

```

import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
```

```
df = pd.read_csv( "d:\data\crab.csv", engine='python', encoding='CP949' )
print(df)
```

## # 2. 다중회귀분석을 하고 종속변수에 영향을 주는 독립변수들이 무엇인지 확인하기

```
from statsmodels.formula.api import ols
model = ols('y ~ sat + weight + width', data =df)
result=model.fit()
print(result.summary())
```

```
OLS Regression Results
=====
Dep. Variable:          y    R-squared:     0.514
Model:                 OLS   Adj. R-squared:  0.506
Method:                Least Squares   F-statistic:   59.69
Date:      Wed, 24 Feb 2021   Prob (F-statistic): 2.30e-26
Time:      14:28:11   Log-Likelihood:   -55.831
No. Observations:      173   AIC:            119.7
Df Residuals:          169   BIC:            132.3
Df Model:               3
Covariance Type:       nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
Intercept  -0.9366    0.500   -1.872      0.063     -1.924     0.051
sat         0.0971    0.009   11.018      0.000      0.080     0.115
weight      -0.0465    0.098   -0.475      0.635     -0.240     0.147
width        0.0535    0.026    2.023      0.045      0.001     0.106
=====
Omnibus:           29.724   Durbin-Watson:   1.475
Prob(Omnibus):      0.000   Jarque-Bera (JB):  7.545
Skew:              0.086   Prob(JB):       0.0230
Kurtosis:           1.992   Cond. No.:      526.
=====
```

분석결과 : **width**는 종속변수에 영향을 주는 유의미한 독립변수로 나타나지만 **weight**는 그렇지 않게 보인다. 별로 중요한 독립변수로 보고 있지 않는다. 분석을 잘못할 수 있게 된다.

## # 3. 팽창계수를 확인한다.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
print(model.exog_names) # 모델에서 분석한 독립변수들이 출력
variance_inflation_factor( model.exog, 1) # 위의 출력된 독립변수 중에 첫번째 컬럼의
                                         # 팽창계수 확인
```

weight과 width가 높은 팽창계수를 보이고 있다.

#### #4. 위의 팽창계수가 높은 두개의 독립변수를 각각 따로따로 이용해서 모델을 생성한다.

```
model1 = ols( 'y ~ sat + width', data=df )
print( model1.fit().summary() )
```

```
OLS Regression Results
=====
Dep. Variable:          y    R-squared:     0.514
Model:                 OLS   Adj. R-squared:  0.508
Method:                Least Squares   F-statistic:   89.83
Date:      Wed, 24 Feb 2021   Prob (F-statistic):  2.39e-27
Time:      14:38:49   Log-Likelihood:   -55.947
No. Observations:      173   AIC:           117.9
Df Residuals:          170   BIC:           127.4
Df Model:                  2
Covariance Type:        nonrobust
=====
            coef    std err        t    P>|t|      [0.025      0.975]
-----
Intercept   -0.7600    0.334    -2.274    0.024    -1.420    -0.100
sat         0.0965    0.009   11.105    0.000     0.079    0.114
width       0.0426    0.013    3.285    0.001     0.017    0.068
=====
Omnibus:             30.476   Durbin-Watson:    1.463
Prob(Omnibus):        0.000   Jarque-Bera (JB):  7.700
Skew:                 0.102   Prob(JB):        0.0213
Kurtosis:              1.987   Cond. No.       347.
=====
```

```
model1 = ols( 'y ~ sat + weight', data=df )
print( model1.fit().summary() )
```

아까는 weight이 중요하지 않은 독립변수였는데 width를 빼고 분석해보니 중요한 독립변수임이 확인이 되고 있다.

#### 문제25. test\_vif1.csv를 내려받고 팽창계수를 확인하여 vif 지수가 높은 독립변수들이 무엇이 있는지 확인하시오!

데이터 설명 : 아이큐, 공부시간, 시험점수로 되어있는 데이터이다.

##### # 1. 데이터 불러오기

```
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

```

import pandas as pd

df = pd.read_csv( "d:\data\test_vif1.csv", engine='python', encoding='CP949' )
print(df)

```

## # 2. 모델생성하기

# 다중회귀분석을 하고 종속변수에 영향을 주는 독립변수들이 무엇인지 확인하기

```

from statsmodels.formula.api import ols
model = ols('시험점수 ~ 아이큐 + 공부시간', data =df)
result=model.fit()
print(result.summary())

```

```

OLS Regression Results
=====
Dep. Variable: 시험점수 R-squared: 0.905
Model: OLS Adj. R-squared: 0.878
Method: Least Squares F-statistic: 33.45
Date: Wed, 24 Feb 2021 Prob (F-statistic): 0.000262
Time: 15:02:04 Log-Likelihood: -25.952
No. Observations: 10 AIC: 57.90
Df Residuals: 7 BIC: 58.81
Df Model: 2
Covariance Type: nonrobust
=====
      coef  std err      t   P>|t|    [0.025    0.975]
-----
Intercept  23.1561  15.967    1.450    0.190   -14.600   60.913
아이큐     0.5094   0.181    2.818    0.026     0.082    0.937
공부시간   0.4671   0.172    2.717    0.030     0.061    0.874
=====
Omnibus: 0.352 Durbin-Watson: 2.255
Prob(Omnibus): 0.839 Jarque-Bera (JB): 0.090
Skew: -0.171 Prob(JB): 0.956
Kurtosis: 2.687 Cond. No. 1.35e+03
=====
```

■ 내가 추가한 파생변수가 유의미한 파생변수인지 확인을 하고 여러 독립변수들 중에 불필요한 독립변수를 제거하고 필요한 독립변수들만 골라내서 회귀분석 할 때 사용하는 `step` 함수를 파이썬으로 구현하기

## 1. R의 `step` 함수의 기능을 가지고 있는 패키지를 import 한다.

```
from sklearn.feature_selection import RFE
```

```
from sklearn.linear_model import LinearRegression  
import pandas as pd
```

## ## 2. 데이터 불러오기

```
df = pd.read_csv("d:\data\insurance.csv", engine='python', encoding='CP949')
```

## ## 3. 컬럼을 인코딩한다.

```
df = pd.get_dummies(df, drop_first=True)  
df.head() #총 9개의 컬럼이 만들어졌음
```

```
df.columns.values #컬럼명을 numpy array 형태로 출력
```

```
df.iloc[ : 1]
```

## ## 4. 독립변수와 종속변수를 numpy array 로 변환한다

```
X = df.iloc[:, [0,1,2,4,5,6,7,8]].to_numpy()  
y = df.iloc[:, 3].to_numpy()
```

## ## 5. 회귀 모델을 생성한다.

```
from sklearn.linear_model import LinearRegression  
model = LinearRegression()
```

## ## 6. step 함수를 이용해서 필요한 독립변수들을 선발

```
# selector = RFE( 회귀모델명, n_features_to_select = 선별할 독립변수의 개수, step = step 횟수 )  
selector = RFE( model, n_features_to_select = 6, step = 1)  
selector = selector.fit(X, y)  
  
print(selector.support_) #[False True True False True True True True True]  
print(selector.ranking_) #[2 1 1 3 1 1 1 1]
```

**문제26. 위의 코드를 다시 수행하는데 지금 추가한 파생변수가 필요한 파생변수인지 확인하는 작업을 수행하시오!**

```
def func_1(x):  
    if x >= 30:
```

```
        return 1
else:
    return 0
```

## 문제27. 비만이면서 흡연을 하는 사람에 대한 파생변수를 bmi30\_smoker라는 이름으로 추가하시오!

힌트 : df['bmi30\_smoker'] = ?

## 1. R 의 step 함수의 기능을 가지고 있는 패키지를 import 합니다.

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
import pandas as pd
```

## 2. 데이터 불러오기

```
df = pd.read_csv("d:\data\insurance.csv", engine='python', encoding='CP949')
print(df.head())
```

## 3. 파생변수 만들기

```
def func_1(x):
    if x >= 30:
        return 1
    else:
        return 0
```

```
df['bmi30'] = df['bmi'].apply(func_1)
df
```

```
df = pd.get_dummies(df, drop_first=True)
df.head()
```

```
df.iloc[ :, :]
```

```
df['bmi30_smoker'] = df['bmi30']*df['smoker_yes']
df
```

##4. 독립변수와 종속변수를 numpy array로 변환합니다.

```
#X = 독립변수들 ( numpy array 형태 ) --> age+sex+bmi+children+smoker+region
X = df.iloc[ :, [0, 1, 2, 4, 5, 6, 7, 8, 9,10] ].to_numpy()
# y = 종속변수 ( numpy array 형태 ) --> expenses
y = df.iloc[ :, 3 ].to_numpy()
```

## 5. 회귀 모델을 생성합니다.

```
from sklearn.linear_model import LinearRegression
```

```

model = LinearRegression()

##6. step 함수를 이용해서 필요한 독립변수들을 선별합니다.

#selector = RFE ( 회귀모델명, n_features_to_select= 선별할 독립변수의 갯수, step=스텝횟수)

selector = RFE ( model , n_features_to_select= 6 , step=1)
selector = selector.fit( X, y )

print ( selector.support_ ) # [False True True False True False True True]
                           # 8개의 독립변수들중에 선택된 5가지가 True 로 출력이 됨

print( selector.ranking_ ) # [3 1 1 4 1 2 1 1]
                           # 8개의 독립변수들의 중요도 순위가 출력이 됨

df.iloc[ :, [0, 1, 2, 4, 5, 6, 7, 8, 9,10] ]

# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산
model.fit(X, y)
r_square = model.score(X, y)

print(r_square)
print('₩n')

# 회귀식의 기울기
print('기울기 a: ', model.coef_)
print('₩n')

# 회귀식의 y절편
print('y절편 b', model.intercept_)
print('₩n')

```

**문제28. ( 오늘의 마지막 문제 2/24 )** 문제 27번에서 선별된 독립변수들만 가지고 회귀모델을 생성하고 훈련시켜 나온 결정계수가 어떻게 되는지 확인하시오!

```

# 학습을 마친 모형에 test data를 적용하여 결정계수(R-제곱) 계산
X = ?
y = ?
model.fit(X, y)
r_square = model.score(X, y)

print(r_square)
print('₩n')

df = pd.read_csv("c:\\\\data\\\\insurance.csv", engine='python', encoding='CP949')

```

```

df = pd.get_dummies(df, drop_first=True)

def func_1(x):
    if x >=30:
        return 1
    else:
        return 0

df['bmi30'] = df['bmi'].apply(func_1)
df['bmi30_smoker'] = df['bmi30']*df['smoker_yes']

df2 = df.iloc[:,[0,1,2,4,5,6,7,8,9,10]]
X = df2.to_numpy()
y = df.iloc[:, 3].to_numpy()

model = LinearRegression()
score_list=[]
x = [x for x in range(1,12)]
for i in x:
    selector = RFE(model, n_features_to_select=i, step =1 )
    selector = selector.fit(X,y)
    score_list.append(selector.score(X,y))

result = pd.DataFrame(data=dict(n_feature=x,score=score_list))
result

```

## #그래프

```

plt.figure(figsize=(12,5))
plt.subplots_adjust(left=0.125,
                   bottom=0.1,
                   right=1,
                   top=0.9,
                   wspace=0.2,
                   hspace=0.35)

plt.subplot(131) # 추가로 그래프를 그린다.
plt.plot(x,score_list,color='blue', marker='o', markerfacecolor='red')
plt.title('Accuracy', size=15)
plt.xticks([0,1,3,5,7,8,9,11])
plt.axvline(7, color='r',linestyle=':')
plt.axvline(8, color='r',linestyle='--')
plt.xlabel("n_feature")

```

```
plt.ylabel('Accuracy')
plt.show()
```

```
# 8 개 일때 중요 변수들과 다른 변수들 보기
```

```
selector = RFE(model, n_features_to_select=8, step =1 )
selector = selector.fit(X,y)
score_list.append(selector.score(X,y))
```

```
# 영향력이 큰 중요 변수들
```

```
for i,j in zip(df2.columns.values,list(selector.support_)):
    if j==True:
        print(i)
    else:
        pass
```

```
# 영향력이 작은 변수들
```

```
for i,j in zip(df2.columns.values,list(selector.support_)):
    if j==False:
        print(i)
    else:
        pass
```

## ■ R을 활용한 머신러닝부터 지금까지 복습( 큰 그림 )

머신러닝에서 뭘 발견하기 위해서 ?

정확도가 가장 높고 오차가 낮은 기계학습 모델을 생성하려고 하는게 전부가 아니다. 리눅스와 하둡 포트폴리오처럼 data에서 유용한 정보를 얻어내기 위해서 머신러닝을 활용하는 것이다.

리눅스와 하둡 : 데이터 저장공간 구성

SQL, 파이썬의 시각화 : 데이터를 읽고 정보를 추출해서 결론 도출

데이터에서 유용한 정보 발견:

1. 회귀분석 할때 : 미국 의료비 데이터

비만인 사람이 흡연까지 하면 의료비가 더 증가되는 구나 ( 결정계수가 상승 )

2. 분류모델 생성할 때 : 타이타닉 데이터

여자와 아이의 생존율이 더 높았구나 ( 정확도 상승 )

RFE 함수 : 내가 생성한 파생변수가 유의미한지 알려주는 함수

## ■ 7. 로지스틱 회귀

종속변수가 범주형인 경우에 적용되는 회귀분석 모형( 수제비: p3-8 )

( 반응변수 )

\* 회귀분석 종류

1. 단순회귀 분석 : 종속변수가 연속형 수치 데이터
2. 다항회귀 분석 : 종속변수가 연속형 수치 데이터
3. 다중회귀 분석 : 종속변수가 연속형 수치 데이터
4. 로지스틱 회귀 분석 : 종속변수가 범주형인 데이터

Kaggle 타이타닉 데이터 : seaborn 내장 데이터보다 결측치와 이상치가 더 많다.

### ( 알아낼 것 )

1. 이 데이터에 맞는 가장 좋은 머신러닝 알고리즘과 코드는 무엇인지?
2. 이 데이터에 맞는 가장 좋은 파생변수는 무엇인지?

### ( 로지스틱 회귀 간단한 방법 )

#### ■ 머신러닝 데이터 분석을 하기 위한 단계

1. 데이터 불러오기 --> 파생변수 추가
2. 데이터 탐색 및 전처리 --> 결측치 처리 - age, embark\_town, embark
3. 데이터 정규화 또는 표준화 --> 이상치에 민감하지 않도록 단위를 일정하게 조정  
--> 단위를 일정하게 조정
4. 범주형 데이터에 대한 더미변수( 인코딩 ) 생성 ( 파이썬 )
5. 훈련데이터와 테스트 데이터 분할
6. 머신러닝 모델 생성
7. 훈련데이터로 머신러닝 모델을 훈련
8. 머신러닝 모델 평가
9. 머신러닝 모델 성능 개선

#### ■ 결측치 처리

머신러닝 데이터 분석의 정확도는 분석 데이터의 품질에 의해 좌우된다. 데이터 품질을 높이기 위해서는 누락 데이터 처리, 중복 데이터 처리 등 오류를 수정하고 분석 목적에 맞게 변형하는 과정이 필요하다.

데이터 프레임에는 원소 데이터 값이 종종 누락되는 경우가 있다.

일반적으로 데이터 값이 존재하지 않는 누락 데이터를 NaN으로 표시한다.

머신러닝 분석 모형에 데이터를 입력하기 전에 반드시 누락 데이터를 제거하거나 다른 적절한 값으로 대체하는 과정이 필요하다.

누락 데이터가 많아지면 데이터의 품질이 떨어지고 머신러닝 분석 알고리즘을 왜곡하는 현상이 발생하기 때문이다. 그래서 반드시 아래의 3가지 함수를 잘 알고 있어야 한다.

##### 1. 누락 데이터를 찾는 함수

- 1.1 isnull() : 누락 데이터면 True를 반환하고 누락 데이터가 아니면 False를 반환하는 함수
- 1.2 notnull() : 누락 데이터가 아니면 True를 반환하고 누락 데이터이면 False를 반환한다.

예 : tat = sns.load\_dataset('titanic')  
tat.head().isnull()

## 문제29. emp 데이터 프레임의 누락 데이터는 몇개인가?

```
emp.isnull().sum()
```

### 2. 누락 데이터를 제거하는 함수

#### 2.1 열을 삭제한다.

```
import seaborn as sns  
tat = sns.load_dataset('titanic')  
tat.isnull().sum()  
tat.dropna( axis=1, thresh=500, inplace=True)
```

# 누락데이터 (NaN)이 500개 이상이면 열을 삭제한다.

# inplace=True를 사용하면 원본(tat)에서 삭제한다.

#아래와 같이 tat2를 만들 때 inplace=True를 안쓰고 만들면?

```
import seaborn as sns  
tat = sns.load_dataset('titanic')  
tat2 = tat.dropna( axis = 1, thresh=500 )  
tat2.columns
```

설명 : 위와 같이 tat2를 만들 때 inplace=True를 안쓰고 만들면 tat2에는 deck이 없는 상태로 만들어지고 원본인 tat에는 deck이 남아있다. 그러나 inplace=True를 쓰고 하게 되면 tat와 tat2에도 deck이 없는 상태가 된다.

axis = 1은 축( 열 )을 날리겠다는 뜻이다.

axis = 0은 행을 지우겠다는 뜻이다.

#### 2.2 행을 삭제한다.

```
import seaborn as sns  
tat = sns.load_dataset('titanic')  
tat.deck.isnull().sum()  
tat.dropna( subset=['deck'], how='any', axis = 0, inplace=True )  
tat.deck.isnull().sum()
```

## 문제30. emp 데이터 프레임에서 comm의 결측치를 확인하고 comm의 결측치 행들을 삭제하시오!

```
import pandas as pd  
emp = pd.read_csv( "d:\data\emp.csv" )  
emp.comm.isnull().sum() #10  
emp.dropna( axis = 0, inplace=True)  
emp #4개의 행만 남았다.
```

### 3. 누락 데이터를 다른 데이터로 치환하는 함수

#### 3.1 평균값으로 누락 데이터를 바꾸기

```
import seaborn as sns
```

```
tat = sns.load_dataset('titanic')
mean_age = tat['age'].mean(axis=0)
mean_age
tat['age'].fillna( mean_age, inplace=True )
```

**문제31.** 위의 예제 코드에 나이 데이터의 누락 데이터를 평균값으로 치환했을 때 와 안했을 때의 차이를 확인하시오!

```
import seaborn as sns
tat = sns.load_dataset('titanic')
mean_age = tat['age'].mean(axis=0)

print(tat['age'].isnull().sum()) #177
tat['age'].fillna( mean_age, inplace=True )
print(tat['age'].isnull().sum()) #0
```

### 3.2 최빈값으로 누락 데이터를 바꾸기

```
import seaborn as sns
tat = sns.load_dataset('titanic')
most_age = tat['age'].value_counts(dropna=True).idxmax()
print(most_age)
```

**문제32.** (점심시간 문제) 위의 코드에서 최빈값에 해당하는 나이를 출력하고 누락치를 최빈값으로 치환하시오!

```
import seaborn as sns
tat = sns.load_dataset('titanic')
most_age = tat['age'].value_counts(dropna=True).idxmax()
print(most_age) #24
tat['age'].fillna( most_age, inplace=True )
print(tat['age'].isnull().sum()) #0
```

### 3.3 이웃하고 있는 주변의 데이터로 누락 데이터를 바꾸기

문법 : df['age'].fillna( method = 'ffill', inplace=True )

```
import seaborn as sns
tat = sns.load_dataset('titanic')
tat.head()
tat['deck'].fillna(method='ffill', inplace=True ) # 결측치 아래의 데이터로 채워 넣는다.
tat['deck'].fillna(method='bfill', inplace=True ) # 결측치 위의 데이터로 채워 넣는다.
tat.head()

print(tat['deck'].isnull().sum())
```

### 3.4 누락 데이터가 아닌 데이터에 대한 회귀 예측값으로 누락 데이터를 바꾸기

결측치가 너무 많으면 컬럼을 지운다.

결측치를 다른값으로 치환하고자 한다면 평균값, 최빈값, 주변의 행들로 치환, 회귀분석의 예측값으로 치환한다.

#### 문제33. emp데이터 프레임에서 월급과 커미션 컬럼을 삭제하시오!

```
import pandas as pd  
emp = pd.read_csv("c://data//emp.csv")  
emp.drop(['sal', 'comm'], axis = 1, inplace=True)  
emp
```

#### ■ dropna의 how의 의미?

'any' : If any NA values are present, drop that row or column.

'all' : If all values are NA, drop that row or column.

예 : emp.dropna( axis=0, how='all', inplace=True )

위의 코드의 뜻은 emp 데이터 프레임에서 모든행이 NA인 행을 삭제하겠다.

그런데 emp 데이터 프레임에는 모든행이 NA인 컬럼은 없으므로 실행이 안되었다.

그래서 실행해도 14개의 행이 그대로 있는 것이다.

emp.dropna( axis=0, how='any', inplace=True)

NA가 있는 행은 무조건 지운다는 뜻이다.

#### 문제34. emp 데이터프레임에서 NA가 있는 행은 무조건 지우시오~

```
import pandas as pd  
emp = pd.read_csv("c://data//emp.csv")  
emp.dropna(axis=0, how='any', inplace=True)  
emp
```

#### ■ 타이타닉 데이터셋의 정보 중 category 데이터형은?

#	Column	Non-Null Count	Dtype
0	survived	714	non-null int64
1	pclass	714	non-null int64
2	sex	714	non-null object
3	age	714	non-null float64
4	sibsp	714	non-null int64
5	parch	714	non-null int64
6	fare	714	non-null float64
7	embarked	714	non-null object <-- 문자형

```
8 class      714 non-null  category  <-- R의 factor( 문자형 + level)와 같음  
9 who       714 non-null  object  
10 adult_male 714 non-null  bool  
11 alive     714 non-null  object  
12 alone    714 non-null  bool
```

## ■ 데이터 표준화

실무에서 접하는 데이터셋은 다양한 사람들의 손을 거쳐서 만들어진다.

여러곳에서 수집한 자료들은 대소문자 구분, 약칭활용 등 여러가지 원인에 의해서 다양한 형태로 표현된다.

잘 정리된 것으로 보이는 자료를 자세히 들여다보면 서로 다른 단위가 섞여있거나 같은 대상을 다른 형식으로 표현하는 경우가 의외로 많다.

그래서 단위환산을 통해 같은 형식으로 변환을 해 줄 필요가 있다.

예 : 모전자에서 모제품의 판매대수를 미리 예측하여 생산라인에 다음달에 몇대가 판매가 될거라는것을 예측해야하는 머신러닝 모델을 만들어야하는 경우에 전세계에서 엑셀파일을 보내오는데 나라마다 단위형식이 다르다  
(화폐단위, 연비단위 ,.....)

문제35. women\_child 파생변수를 추가하고 다시 학습 시켜서 모델의 성능을 확인하시오

## ■ 성능 개선을 위해 데이터를 시각화해서 데이터에서 스토리를 뽑아내는 방법

### \* 범주형 데이터를 처리하는 방법

#### 1. 더미변수로 처리 : 숫자 0과 1로 표현

예 : 여자 또는 아이는 1이고 아니면 0으로 표현

#### 2. 구간 분할로 처리 : 일정한 구간으로 나눔

예 : 미세먼지의 농도는 연속형 수치형 데이터여서 누군가가 나에게 오늘 미세먼지 농도가 어땠느냐 물어봤을 때 2.345 마이크로 미터\* 마이크로그램 세제곱입니다 라고 하면 전문가가 아니면 알아듣기 어렵다. 그냥 좋음, 보통, 나쁨, 매우나쁨이라고 알려주는게 훨씬 편하다.

예: 타이타닉 데이터도 low, medium-low, medium, high 구간 분할하면 기계가 훨씬 더 잘 이해해서 성능이 올라가게 된다

```
import pandas as pd
```

```

import numpy as np
# CSV 파일을 데이터프레임으로 변환
df = pd.read_csv('D:\data\auto-mpg.csv', header=None)

# 열 이름 지정
df.columns = ['mpg','cylinders','displacement','horsepower','weight',
'acceleration','model year','origin','name']
# 설명: horsepower 가 빠졌음을 확인하세요 ~ 통계요약정보를 출력하려면
# 숫자형 데이터여야 출력될 수 있습니다.

# mpg 는 mile per gallon 의 약자로 영국과 미국에서는 한국과는 달리
# 갤런당 마일 단위로 연비를 표시합니다.
# 한국은 리터당 킬로미터(km/L) 단위로 표시한다.
# mpg 열을 한국에서 사용하는 km/L 로 변환해줘야 합니다.
# 1 갤런이 3.78541 이다. 그리고 1마일이 1.60934 km 이다.
# 그렇다면 1 mpg(mile per gallon) 은 ?

# print ( 1.60934 / 3.78541 ) # 0.425 km/L

# horsepower 열의 자료형 변경 (문자열 ->숫자)
df['horsepower'].replace('?', np.nan, inplace=True) # '?'을 np.nan으로 변경

df.dropna(subset=['horsepower'], axis=0, inplace=True) # 누락데이터 행을 삭제

df['horsepower'] = df['horsepower'].astype('float') # 문자열을 실수형으로 변환
df

count,bin_dividers = np.histogram( df.horsepower, bins=3)
count # 각 구간에 속하는 값의 개수
      # array([257, 103, 32], dtype=int64)
bin_dividers #경곗값 리스트 array([ 46.        , 107.33333333, 168.66666667, 230.        ])

설명 : 107-46 = 61, 168-107 = 61, 230-168 = 61

import pandas as pd
bin_names = ['저출력', '보통출력', '고출력']
df['hp_bin']=pd.cut( x = df.horsepower, bins=bin_dividers, labels=bin_names)
df

```

설명 : 이처럼 연속형변수를 일정한 구간으로 나눠 각 구간을 범주형 이산 변수로 변환하는 과정을 구간분할( binning )이라고 한다.

판다스의 cut()함수를 이용하면 연속 데이터를 여러 구간으로 나누고 범주형 데이터로 변환할 수 있다.

**문제36. emp 데이터 프레임의 월급을 3개로 구간분할 하는데 고소득, 중간소득, 저소득으로 나눠서 출력되게 sal\_bin이라는 파생변수를 추가하시오~**

```
import pandas as pd
import numpy as np
emp = pd.read_csv("c:\data\emp3.csv")

count,bin_dividers = np.histogram(emp.sal, bins=3)

bin_names = ['저소득', '중간소득', '고소득']
emp['sal_bin']=pd.cut( x = emp.sal, bins=bin_dividers, labels=bin_names, include_lowest =
True)
emp
```

**문제37. 여자와 아이에 대한 파생변수를 추가해서 0.85까지 올렸던 코드를 로지스틱 회귀가 아니라 양상을 기법이 추가된 랜덤포레스트로 수해해서 정확도가 더 올라가는지 확인하시오!**

```
from sklearn.ensemble import RandomForestClassifier

tree_model = RandomForestClassifier( n_estimators=100,
                                      oob_score=True,
                                      random_state= 9 )

tree_model.fit( X_train, y_train )
```

추가하기

**문제38. ( 오늘의 마지막 문제 2/25 ) 로지스틱 회귀로 시본의 타이타닉 분류 모델을 만드는데 여자와 아이 파생변수말고 아래의 그래프를 보고 더 좋은 파생변수를 생각해서 학습시키고 정확도가 출력된 화면을 올리시오~**

## ■ 7장. 신경망

" 사람의 뇌를 컴퓨터를 이용한 지능처리 "

사람의 뇌 : 생물학적 신경망 내에서 반복적인 시그널이 발생할 때 그 시그널을 기억하는 일종의 학습효과가 있다.

컴퓨터 이용 : 인공신경망에서는 가중치라는 것으로 기억의 효과를 대체할 수 있음을 설명했다.

가중치 : 알아내야 할 파라미터

### ※ 파라미터와 하이퍼 파라미터의 차이 ? (수제비 3-12)

#### 1. 파라미터 ?

-> 모델 내부에서 확인이 가능한 변수로 데이터를 통해서 산출이 가능한 값, 예측을 수행할 때 모델에 의해 요구되어지는 값들, 사람에 의해 수작업으로 측정되지 않음

예 : 신경망에서의 가중치, 회귀분석에서의 결정계수, 서포트 벡터머신에서의 서포트벡터

#### 2. 파이퍼 파라미터 ?

-> 모델에서 외적인 요소로 데이터 분석을 통해 얻어지는 값이 아니라 사용자가 직접 설정해주는 값, 모델의 파라미터 값을 측정하기 위해 알고리즘 구현 과정에서 사용  
-> 하이퍼 파라미터는 주로 알고리즘 사용자에 의해 결정

예 : 신경망에서의 학습률( 러닝레이트 ), knn에서의 k값, 의사결정트리에서의 나무의 깊이, 서포트 벡터 머신에서의 C값과 gamma값

#### \* 퍼셉트론?

-> 뇌의 신경세포 하나를 컴퓨터로 흉내낸 것  
1. 뇌에 신호가 흘러 암기가 되려면 반복학습을 통해 중요도가 결정이 되어야 함.  
2. 인공신경망에서는 신호를 가중치를 활용해서 신호가 흐른다, 흐르지 않는다고 구현 함.

컴퓨터 : 정보를 메모리의 특정위치에 저장

뇌 : 정보를 저장하는 공간이 따로 없다.

신경세포( 뉴런 )의 연결관계를 변경하는 방식으로 정보 저장

R : neuralnet 함수로 구현 : 콘크리트 강도 예측

파이썬 : sklearn의 neural\_network 함수로 구현 : 콘크리트의 강도 예측

1. 예측 : 콘크리트 데이터
2. 분류 : 타이타닉 데이터

**문제39. R 을 활용한 머신러닝에서 사용한 콘크리트 데이터를 파이썬의 신경망으로 강도를 예측을 하시오 !**

답 :

#### [Step 1] 데이터 준비/ 기본 설정

```
# -*- coding: utf-8 -*-
```

#### 기본 라이브러리 불러오기

```
import pandas as pd  
'''
```

[Step 1] 데이터 준비/ 기본 설정

```
'''
```

# load\_dataset 함수를 사용하여 데이터프레임으로 변환

```
df = pd.read_csv('D:\data\concrete.csv')
```

# IPython 디스플레이 설정 - 출력할 열의 개수 한도 늘리기/ 열이 9개뿐이라 그냥 봐도됨

```
#pd.set_option('display.max_columns', 15)
```

```
#df.head()
```

#### 콘크리트 데이터 설명

1. mount of cement : 콘크리트의 총량
2. slag : 시멘트
3. ash : 분 (시멘트)
4. water : 물
5. superplasticizer : 고성능 감수재(콘크리트 강도를 높이는 첨가제)
6. coarse aggregate : 굵은 자갈
7. fine aggregate : 잔 자갈
8. aging time : 숙성시간

#### 결측치 확인

```
df.isnull().sum()
```

#### [Step 2] 데이터 탐색/ 전처리

```

df.head()
df.info()

### [Step 4] 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

"""
[Step 4] 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)
"""

# 속성(변수) 선택
X=df[['cement', 'slag','ash','water','superplastic','coarseagg','fineagg','age']] #독립 변수 X

y=df['strength'] #종속 변수 Y

X.head()

# 설명 변수 데이터를 표준화(normalization)
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
print(X)
print(type(X)) #numpy.array

# train data 와 test data로 구분(8:2 비율)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
print('₩n')

### [Step 5] 신경망 모형 - sklearn 사용

##### from sklearn.neural_network
##### 1. 분류 : MLPClassifier
##### 2. 수치예측 : MLPRegressor

from sklearn.neural_network import MLPRegressor #예측이라 해당 모듈 사용

# 3.neural network 모델 적합
clf = MLPRegressor(random_state=0, max_iter=500) #학습 500번시킴

### [Step 6] train data를 가지고 모형 학습

# train data를 가지고 모형 학습
clf.fit(X_train, y_train)

### [Step 7] test data를 가지고 y_hat을 예측

```

```

# test data를 가지고 y_hat을 예측 (분류)
y_hat = clf.predict(X_test)

print(y_hat[0:10])
print(y_test.values[0:10])
print('₩n')

### [Step 8] 모형 성능 평가 - 상관관계 계산

import scipy.stats as stats
svm_report = stats.pearsonr(y_test, y_hat) #상관계수 확인하는 코드
print(svm_report)

### 0.9 는 상관계수이고 2.385285961523278e-77가 p-value

```

**문제40.** 위의 신경망의 성능을 더 올리시오! 뉴런의 개수를 늘리거나 층을 더 늘리는 등 신경망에 연관된 하이퍼 파라미터를 알아내야 합니다.  
하이퍼 파라미터를 자동으로 알아내게하는 파이썬의 기능인 **grid search**를 이용해서 알아내시오.

힌트코드 :

```

from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV # grid search 기능을 사용할 수 있게 import

param_grid = [
    {
        'activation' : ['identity', 'logistic', 'tanh', 'relu'], #활성화 함수 종류 4가지
        'solver' : ['lbfgs', 'sgd', 'adam'], # 옵티마이저 ( 최적화 )의 종류 3가지
        'hidden_layer_sizes': [
            (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),
            (12),(13),(14),(15),(16),(17),(18),(19),(20),(21)
        ]
    }
]

# 옵티마이저? global minima를 찾아가기 위한 방법론( 수학식 )을 코드로 구현한 것
# lbfgs 옵티마이저? 가장 인기있는 옵티마이저이고 computer memory의 제한된 양만을
# 사용해서 global minima를 찾아가게끔 코드가 구현되어 있다.
# hidden_layer_sizes = ( 뉴런수, 층수 ), 위의 예제에서는 뉴런수만 정해줬고 층수는 안정했다.

# 모델을 생성한다.
clf = GridSearchCV(MLPRegressor(), param_grid, cv=3, n_jobs = -1, verbose = 2 )

```

```

#설명 : GridSearchCV( 모델명, grid 파라미터 값, k-holdout설정, n_jobs = -1, verbose = 2 )
# cv=3 은 k-holdout의 k값을 3으로 해서 훈련데이터를 3개의 파티션으로 나누고
# 2개를 훈련 데이터로 사용하고 나머지 한개를 테스트 데이터로 사용하겠다.
# n_jobs = -1 를 사용하면 지저분하게 최적의 하이퍼 파라미터를 알아내는 작업 전체를
# 보여주는게 아니라 요약해서 보여준다.
# verbose = 2는 gridSearch 하는 과정을 보여준다.
#MLPRegressor(random_state=10) 이런식으로 radom_state를 주면 할때마다 동일한 값이 나옴

```

clf.fit(X\_train, y\_train) # 위에서 만든 모델을 훈련시킨다.

```

print("Best parameters set found on development set:")
print(clf.best_params_) # 최고의 하이퍼 파라미터의 조합을 출력해준다.

```

## gridserach쓰면 캐글 순위를 올릴 수 있다!!!!!!

**문제41. 모델생성할 때 설정하는 random\_stats도 gridsearch해서 알아내게 하시오!**

```

from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import GridSearchCV # grid search 기능을 사용할 수 있게 import

param_grid = [
    {
        'activation' : ['identity', 'logistic', 'tanh', 'relu'], #활성화 함수 종류 4가지
        'solver' : ['lbfgs', 'sgd', 'adam'], # 옵티마이져 ( 최적화 )의 종류 3가지
        'hidden_layer_sizes': [
            (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),
            (12),(13),(14),(15),(16),(17),(18),(19),(20),(21)
        ],
        'random_state':[0,1,2,3,4,5,6,7,8,9,10]
    }
]

```

clf = GridSearchCV(MLPRegressor(), param\_grid, cv=3, n\_jobs = -1, verbose = 2 )

**문제42. grid search를 이용하지 말고 보스톤 하우징데이터의 수치 예측을 하는 코드를 작성하시오!**

R을 활용하는 머신러닝에서 사용했던 데이터 : boston.csv

코드이름 변경 : 신경망 코드\_concreate.ipynb ---> 신경망코드\_boston.ipynb

**문제43. ( 점심시간문제 )** 보스톤 하우징 데이터의 가격 수치예측하는 신경망 코드를 grid Search 를 이용해서 작성하고 상관계수값을 올리세요~

## ■ 8장. 서포트 벡터 머신

서포트 벡터 머신은 기계학습 분야의 하나로 패턴인식, 자료분석을 위한 지도학습 모델이며 주로 분류와 회귀분석을 위해 사용한다.

서포트 벡터 머신 알고리즘은 데이터를 분류하는 가장 큰 폭을 가진 경계를 찾는 알고리즘이다. 비선형 분류를 하기 위해서 주어진 데이터를 고차원 특징 공간으로 사상하는 작업이 필요한데, 이를 효율적으로 하기 위해 커널 트릭을 사용하기도 한다.

### 예제. 씨본의 타이타닉 데이터를 분류하는 서포트 벡터 머신 모델 코드

```
# -*- coding: utf-8 -*-

### 기본 라이브러리 불러오기
import pandas as pd
import seaborn as sns

...
[Step 1] 데이터 준비/ 기본 설정
...
# load_dataset 함수를 사용하여 데이터프레임으로 변환
df = sns.load_dataset('titanic')

# IPython 디스플레이 설정 - 출력할 열의 개수 한도 늘리기
pd.set_option('display.max_columns', 15)
df

...
[Step 2] 데이터 탐색/ 전처리
...

# NaN값이 많은 deck 열을 삭제, embarked와 내용이 겹치는 embark_town 열을 삭제
rdf = df.drop(['deck', 'embark_town'], axis=1)

### 여기서는 나이 결측치 값을 날리지만 실제 데이터에는 데이터 치환을 해줘야 한다

# age 열에 나이 데이터가 없는 모든 행을 삭제 - age 열(891개 중 177개의 NaN 값)
rdf = rdf.dropna(subset=['age'], how='any', axis=0)
```

```
# embarked 열의 NaN값을 승선도시 중에서 가장 많이 출현한 값으로 치환하기
most_freq = rdf['embarked'].value_counts(dropna=True).idxmax()
rdf['embarked'].fillna(most_freq, inplace=True)
```

rdf

...

[Step 3] 분석에 사용할 속성을 선택

...

```
# 분석에 활용할 열(속성)을 선택
```

```
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked']]
ndf
```

```
# 원핫인코딩 - 범주형 데이터를 모형이 인식할 수 있도록 숫자형으로 변환
```

```
onehot_sex = pd.get_dummies(ndf['sex'])
ndf = pd.concat([ndf, onehot_sex], axis=1)
ndf
```

```
onehot_embarked = pd.get_dummies(ndf['embarked'], prefix='town')
ndf = pd.concat([ndf, onehot_embarked], axis=1)
ndf
```

```
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
ndf
```

...

[Step 4] 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

...

```
# 속성(변수) 선택
```

```
X=ndf[['pclass', 'age', 'sibsp', 'parch', 'female', 'male',
        'town_C', 'town_Q', 'town_S']] #독립 변수 X
y=ndf['survived'] #종속 변수 Y
```

```
# 설명 변수 데이터를 표준화(normalization)
```

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)
```

```
# train data 와 test data로 구분(7:3 비율)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
```

```
print('train data 개수: ', X_train.shape)
```

```
print('test data 개수: ', X_test.shape)
```

```
print('Wn')
```

...

[Step 5] SVM 분류 모형 - sklearn 사용

```

...
# sklearn 라이브러리에서 SVM 분류 모형 가져오기
from sklearn import svm

# 모형 객체 생성 (kernel='rbf' 적용)
svm_model = svm.SVC(kernel='rbf')
svm_model

# train data를 가지고 모형 학습
svm_model.fit(X_train, y_train)

# test data를 가지고 y_hat을 예측 (분류)
y_hat = svm_model.predict(X_test)

print(y_hat[0:10])
print(y_test.values[0:10])
print('n')

# 모형 성능 평가 - Confusion Matrix 계산
from sklearn import metrics
svm_matrix = metrics.confusion_matrix(y_test, y_hat)
print(svm_matrix)
print('n')

# 모형 성능 평가 - 평가지표 계산
svm_report = metrics.classification_report(y_test, y_hat)
print(svm_report)

```

**문제44. 지금까지의 정확도는 0.81 | 데 여자와 아이먼저라는 파생변수를 추가하면 정확도가 더 올라가는지 확인하시오!**

```

# -*- coding: utf-8 -*-

### 기본 라이브러리 불러오기
import pandas as pd
import seaborn as sns

...
[Step 1] 데이터 준비/ 기본 설정
...

# load_dataset 함수를 사용하여 데이터프레임으로 변환
df = sns.load_dataset('titanic')

# IPython 디스플레이 설정 - 출력할 열의 개수 한도 늘리기

```

```
pd.set_option('display.max_columns', 15)
df
```

### 여자와 아이에 대한 파생변수 추가

```
mask = ( df.age < 10 ) | ( df.sex=='female')
mask.astype(int) # True 를 1 로 변경하고 False 를 0 으로 변경
df['women_child'] = mask.astype(int)
```

...

[Step 2] 데이터 탐색/ 전처리

...

```
# NaN값이 많은 deck 열을 삭제, embarked와 내용이 겹치는 embark_town 열을 삭제
rdf = df.drop(['deck', 'embark_town'], axis=1)
```

```
# age 열에 나이 데이터가 없는 모든 행을 삭제 - age 열(891개 중 177개의 NaN 값)
rdf = rdf.dropna(subset=['age'], how='any', axis=0)
```

```
# embarked 열의 NaN값을 승선도시 중에서 가장 많이 출현한 값으로 치환하기
most_freq = rdf['embarked'].value_counts(dropna=True).idxmax()
rdf['embarked'].fillna(most_freq, inplace=True)
rdf
```

...

[Step 3] 분석에 사용할 속성을 선택

...

```
# 분석에 활용할 열(속성)을 선택
```

```
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked','women_child']]
```

```
# 원핫인코딩 - 범주형 데이터를 모형이 인식할 수 있도록 숫자형으로 변환
```

```
onehot_sex = pd.get_dummies(ndf['sex'])
ndf = pd.concat([ndf, onehot_sex], axis=1)
```

```
onehot_embarked = pd.get_dummies(ndf['embarked'], prefix='town')
ndf = pd.concat([ndf, onehot_embarked], axis=1)
```

```
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)
ndf
```

...

[Step 4] 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

...

```
# 속성(변수) 선택
```

```
X=ndf[['pclass', 'age', 'sibsp', 'parch', 'female', 'male',
'town_C', 'town_Q', 'town_S','women_child']] #독립 변수 X
y=ndf['survived'] #종속 변수 Y
```

```
# 설명 변수 데이터를 정규화(normalization)
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

# train data 와 test data로 구분(7:3 비율)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
print('₩n')

'''
```

### [Step 5] SVM 분류 모형 - sklearn 사용

```
'''
```

```
# sklearn 라이브러리에서 SVM 분류 모형 가져오기
from sklearn import svm
```

```
# 모형 객체 생성 (kernel='rbf' 적용)
svm_model = svm.SVC(kernel='rbf')
```

```
# train data를 가지고 모형 학습
svm_model.fit(X_train, y_train)
```

```
# test data를 가지고 y_hat을 예측 (분류)
y_hat = svm_model.predict(X_test)
```

```
print(y_hat[0:10])
print(y_test.values[0:10])
print('₩n')
```

```
# 모형 성능 평가 - Confusion Matrix 계산
```

```
from sklearn import metrics
svm_matrix = metrics.confusion_matrix(y_test, y_hat)
print(svm_matrix)
print('₩n')
```

```
# 모형 성능 평가 - 평가지표 계산
```

```
svm_report = metrics.classification_report(y_test, y_hat)
print(svm_report)
```

타이타닉 svm파생변수 추가후.ipynb ---> 0.83까지 정확도 올림

## 문제45. gridsearch를 이용하여 정확도를 더 올리시오!

서포트 벡터 머신의 하이퍼 파라미터 ? C 와 gamma 와 kernel

힌트 :

...

[Step 5] SVM 분류 모형 - sklearn 사용

...

```
# sklearn 라이브러리에서 SVM 분류 모형 가져오기
```

```
from sklearn import svm
```

```
from sklearn.model_selection import GridSearchCV
```

```
# defining parameter range
```

```
param_grid = {'C': [0.1, 1, 10, 100, 1000],
```

```
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
```

```
    'kernel': ['rbf', 'poly', 'sigmoid', 'linear']}
```

```
grid = GridSearchCV(svm.SVC(), param_grid, refit = True, cv=3, n_jobs = -1, verbose = 2 )
```

```
# fitting the model for grid search
```

```
grid.fit(X_train, y_train)
```

### < 설명 >

estimator : classifier, regressor, pipeline이 사용될 수 있다.

param\_grid : 파라미터 딕셔너리. (파라미터명과 사용될 여러 파라미터 값을 지정)

scoring : 예측 성능을 측정할 평가 방법. 보통은 사이킷런에서 제공하는 문자열

(예: 'accuracy')을 넣지만 별도의 함수도 직접 지정이 가능하다

cv : 교차 검증을 위해 분할되는 폴드 수.

refit : True면 가장 최적의 하이퍼 파라미터를 찾은 뒤 입력된 estimator 객체를 해당 하이퍼 파라미터로 재학습시킨다. (default:True)

문제46. 서포트 벡터 머신으로 최고로 올릴 수 있는 타이타닉 데이터의 정확도는 0.83이었다. 오전에 배웠던 gridsearch를 이용한 신경망 코드를 사용해 수치예측이 아닌 분류로 머신러닝 함수를 변경해서 훈련시키고 정확도를 확인하시오!

```
from sklearn.neural_network import MLPRegressor --> 수치예측
from sklearn.neural_network import MLPClassifier --> 분류
```

힌트코드 :

...

[Step 5] 신경망 분류 모형 - sklearn 사용

...

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
```

```
param_grid = [
    {
        'activation' : ['identity', 'logistic', 'tanh', 'relu'],
        'solver' : ['lbfgs', 'sgd', 'adam'],
        'hidden_layer_sizes': [
            (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),
            (12),(13),(14),(15),(16),(17),(18),(19),(20),(21)
        ],
        'random_state' :[ 0,1,2,3,4,5,6,7,8,9,10]
    }
]
```

```
grid = GridSearchCV(MLPClassifier(), param_grid, cv=3, n_jobs = -1, verbose = 2 )
```

```
#####
```

```
# -*- coding: utf-8 -*-
```

```
### 기본 라이브러리 불러오기
```

```
import pandas as pd
import seaborn as sns
```

...

[Step 1] 데이터 준비/ 기본 설정

...

```
# load_dataset 함수를 사용하여 데이터프레임으로 변환
df = sns.load_dataset('titanic')
```

```
# IPython 디스플레이 설정 - 출력할 열의 개수 한도 늘리기
```

```
pd.set_option('display.max_columns', 15)
df
```

```
### 여자와 아이에 대한 파생변수 추가
```

```
mask = ( df.age < 10 ) | ( df.sex=='female')
```

```
mask.astype(int) # True 를 1 로 변경하고 False 를 0 으로 변경  
df['women_child'] = mask.astype(int)
```

...

## [Step 2] 데이터 탐색/ 전처리

...

```
# NaN값이 많은 deck 열을 삭제, embarked와 내용이 겹치는 embark_town 열을 삭제  
rdf = df.drop(['deck', 'embark_town'], axis=1)
```

```
# age 열에 나이 데이터가 없는 모든 행을 삭제 - age 열(891개 중 177개의 NaN 값)  
rdf = rdf.dropna(subset=['age'], how='any', axis=0)
```

```
# embarked 열의 NaN값을 승선도시 중에서 가장 많이 출현한 값으로 치환하기
```

```
most_freq = rdf['embarked'].value_counts(dropna=True).idxmax()  
rdf['embarked'].fillna(most_freq, inplace=True)  
rdf
```

...

## [Step 3] 분석에 사용할 속성을 선택

...

```
# 분석에 활용할 열(속성)을 선택
```

```
ndf = rdf[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'embarked', 'women_child']]
```

```
# 원핫인코딩 - 범주형 데이터를 모형이 인식할 수 있도록 숫자형으로 변환
```

```
onehot_sex = pd.get_dummies(ndf['sex'])  
ndf = pd.concat([ndf, onehot_sex], axis=1)
```

```
onehot_embarked = pd.get_dummies(ndf['embarked'], prefix='town')  
ndf = pd.concat([ndf, onehot_embarked], axis=1)
```

```
ndf.drop(['sex', 'embarked'], axis=1, inplace=True)  
ndf
```

...

## [Step 4] 데이터셋 구분 - 훈련용(train data)/ 검증용(test data)

...

```
# 속성(변수) 선택
```

```
X=ndf[['pclass', 'age', 'sibsp', 'parch', 'female', 'male',  
       'town_C', 'town_Q', 'town_S', 'women_child']] #독립 변수 X  
y=ndf['survived'] #종속 변수 Y
```

```
# 설명 변수 데이터를 정규화(normalization)
```

```
from sklearn import preprocessing  
X = preprocessing.StandardScaler().fit(X).transform(X)
```

```
# train data 와 test data로 구분(7:3 비율)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)

print('train data 개수: ', X_train.shape)
print('test data 개수: ', X_test.shape)
print('₩n')
```

""

[Step 5] 신경망 분류 모형 - sklearn 사용

""

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
```

```
param_grid = [
    {
        'activation' : ['identity', 'logistic', 'tanh', 'relu'],
        'solver' : ['lbfgs', 'sgd', 'adam'],
        'hidden_layer_sizes': [
            (1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),
            (12),(13),(14),(15),(16),(17),(18),(19),(20),(21)
        ],
        'random_state' :[ 0,1,2,3,4,5,6,7,8,9,10]
    }
]
```

```
grid = GridSearchCV(MLPClassifier(), param_grid, cv=3, n_jobs = -1, verbose = 2 )
```

```
# fitting the model for grid search
grid.fit(X_train, y_train)
```

```
# test data를 가지고 y_hat을 예측 (분류)
y_hat = grid.predict(X_test)
```

```
print(y_hat[0:10])
print(y_test.values[0:10])
print('₩n')
```

# 모형 성능 평가 - Confusion Matrix 계산

```
from sklearn import metrics
svm_matrix = metrics.confusion_matrix(y_test, y_hat)
print(svm_matrix)
print('₩n')
```

# 모형 성능 평가 - 평가지표 계산

```
svm_report = metrics.classification_report(y_test, y_hat)
print(svm_report)
```

## ■ 더 정확도를 올리기 위해서 실험해야할 내용

1. 신경망을 2층에서 3층으로 변경한다.
2. 나이의 결측치행 177개를 삭제하는게 아니라 다른 값으로 치환한다.
  - 나이의 평균값
  - 나이의 최빈값
  - 실제 캐글에서는 이름의 호칭의 평균 값 ( seaborn 타이타닉에는 이름이 없다 )
3. 나이와 운임의 이상치를 제거하고 학습 시킨다.

## ■ 9장. 연관규칙

간단한 성능 측정치( 지지도, 신뢰도, 향상도 )를 이용해서 거대한 데이터에서 데이터간의 연관성을 찾는 알고리즘

예제 :

```
import pandas as pd  
from mlxtend.preprocessing import TransactionEncoder # 연관성 분석을 위한 데이터 전처리  
from mlxtend.frequent_patterns import apriori # 연관성 분석을 위한 아프리오리 알고리즘 함수
```

```
dataset=[['사과','치즈','생수'],  
        ['생수','호두','치즈','고등어'],  
        ['수박','사과','생수'],  
        ['생수','호두','치즈','옥수수']]
```

```
te = TransactionEncoder()  
te_ary = te.fit(dataset).transform(dataset)  
print(te.columns_)  
te_ary
```

# 결과 : 산 물건은 True, 안 산 물건은 False

```
['고등어', '사과', '생수', '수박', '옥수수', '치즈', '호두']
```

```
Out[3]: array([[False,  True,  True, False, False,  True, False],  
               [ True, False,  True, False, False,  True,  True],  
               [False,  True,  True,  True, False, False, False],  
               [False, False,  True, False,  True,  True,  True]])
```

```
df = pd.DataFrame(te_ary, columns=te.columns_) #위 결과를 보기 좋게 데이터프레임으로 변경  
df
```

# 결과 :

	고등어	사과	생수	수박	옥수수	치즈	호두
0	False	True	True	False	False	True	False
1	True	False	True	False	False	True	True
2	False	True	True	True	False	False	False
3	False	False	True	False	True	True	True

```
frequent_itemsets = apriori(df, min_support=0.5, use_colnames=True)  
print(frequent_itemsets) # 사람들이 많이 산 물건들만 출력
```

	support	itemsets
0	0.50	(사과)
1	1.00	(생수)
2	0.75	(치즈)
3	0.50	(호두)
4	0.50	(사과, 생수)
5	0.75	(치즈, 생수)
6	0.50	(호두, 생수)
7	0.50	(호두, 치즈)
8	0.50	(호두, 치즈, 생수)

```
from mlxtend.frequent_patterns import association_rules # 연관성을 찾는 함수 import  
print( association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3) )  
# 신뢰도를 기준으로 물품들 간의 연관성을 출력하는데 임계치를 0.3이상인 것만 출력
```

---

	support	itemsets						
0	0.50	(사과)						
1	1.00	(생수)						
2	0.75	(치즈)						
3	0.50	(호두)						
4	0.50	(사과, 생수)						
5	0.75	(치즈, 생수)						
6	0.50	(호두, 생수)						
7	0.50	(호두, 치즈)						
8	0.50	(호두, 치즈, 생수)						
	antecedents	consequents	antecedent support	consequent support	support	support	#	
0	(사과)	(생수)	0.50	1.00	0.50	0.50		
1	(생수)	(사과)	1.00	0.50	0.50	0.50		
2	(치즈)	(생수)	0.75	1.00	0.75	0.75		
3	(생수)	(치즈)	1.00	0.75	0.75	0.75		
4	(호두)	(생수)	0.50	1.00	0.50	0.50		
5	(생수)	(호두)	1.00	0.50	0.50	0.50		
6	(호두)	(치즈)	0.50	0.75	0.50	0.50		
7	(치즈)	(호두)	0.75	0.50	0.50	0.50		
8	(호두, 치즈)	(생수)	0.50	1.00	0.50	0.50		
9	(호두, 생수)	(치즈)	0.50	0.75	0.50	0.50		
10	(치즈, 생수)	(호두)	0.75	0.50	0.50	0.50		
11	(호두)	(치즈, 생수)	0.50	0.75	0.50	0.50		
12	(치즈)	(호두, 생수)	0.75	0.50	0.50	0.50		
13	(생수)	(호두, 치즈)	1.00	0.50	0.50	0.50		
	confidence	lift	leverage	conviction				
0	1.000000	1.000000	0.000	inf				
1	0.500000	1.000000	0.000	1.0				
2	1.000000	1.000000	0.000	inf				
3	0.750000	1.000000	0.000	1.0				
4	1.000000	1.000000	0.000	inf				
5	0.500000	1.000000	0.000	1.0				
6	1.000000	1.333333	0.125	inf				
7	0.666667	1.333333	0.125	1.5				
8	1.000000	1.000000	0.000	inf				
9	1.000000	1.333333	0.125	inf				
10	0.666667	1.333333	0.125	1.5				
11	1.000000	1.333333	0.125	inf				
12	0.666667	1.333333	0.125	1.5				
13	0.500000	1.000000	0.000	1.0				

---

결론 : 생수와 사과는 같은곳에 둬야한다.

### 문제47. 아래의 데이터에서 연관성을 도출하시오!

```
dataset=[['빵','우유'],
         ['맥주','빵','기저귀','계란'],
         ['맥주','콜라','기저귀','우유'],
         ['콜라','빵','기저귀','우유']]
```

### 문제48.( 오늘의 마지막 문제 3/2 ) R을 활용한 머신러닝때 사용했던 보습학원 데이터로 연관분석을 하겠다. 보습학원이 있는 건물에는 어떤 업종이 많이 있는가?

데이터 : building3.csv

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
import numpy as np
```

### # CSV 파일을 데이터프레임으로 변환

```
#dataset = pd.read_csv('D:\data\building.csv', encoding='cp949', index_col='Unnamed: 0')
#dataset = pd.read_csv('D:\data\building2.csv', encoding='cp949',
header=None,names=['병원', '약국', '카페', '휴대폰매장', '일반음식점', '패밀리레스토랑', '당구장',
'보습학원', '슈퍼마켓',
#      '은행', '편의점', '화장품'])

dataset = pd.read_csv('D:\data\building3.csv', encoding='cp949', header=None)
dataset = dataset[ [1,2,3,4,5,6,7,8,9,10,11,12]]
```

```
dataset = dataset.fillna(0) #결측치를 0으로 바꾼다.
dataset
```

```
import numpy as np

d = np.array(dataset)
d2 = d.astype('bool') #숫자 1은 True로 변경, 숫자 0은 False로 변경
d2
df = pd.DataFrame(d2, columns=['hospital', 'pharmacy', 'cafe', 'mobile', 'restaurant',
    'familyrest', 'billiardroom', 'academy', 'supermarket', 'bank',
    'convenience', 'cosmetics']) #위에서 나온걸 보기 좋게 데이터프레임으로 변경
df
```

```
#결과 :
```



## ■ 아프리오리 알고리즘( 장바구니 분석 ) 파이썬 코드 정리

##### 1. 아프리오리 알고리즘 구현에 필요한 패키지를 임포트 합니다.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder # 연관분석을 위한 데이터 전처리
from mlxtend.frequent_patterns import apriori # 연관성 분석을 위한 아프리오리 알고리즘 함수
```

##### 2. 보습학원 데이터를 판다스 데이터 프레임으로 생성합니다.

```
dataset=pd.read_csv('d:/data/building.csv', encoding='CP949', index_col='Unnamed: 0')
dataset
```

##### 3. 결측치를 0 으로 치환합니다.

```
dataset = dataset.fillna(0)
dataset.shape # (20, 12)
dataset
```

##### 4. 1은 True 로 변경하고 0 을 false 로 변경하기 위해서 리스트에 숫자 1에 ##### 해당하는 컬럼명을 하나씩 담는다.

```
a = []
colnames = dataset.columns
for j in range(0,20): # 행이 전체가 20개여서
    b =[]
    for i,k in (enumerate(colnames)): # 모든 컬럼의 갯수
        if dataset.iloc[j,i] == True:
            b.append(k)
        else:
            pass
    a.append(b)
dataset=a
print(a)
```

##### 5. 위의 리스트의 데이터를 받아서 True, False 로 변경한다.

```
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
te_ary
```

##### 6. 위의 결과를 데이터 프레임 형태로 만듭니다.

```
df = pd.DataFrame(te_ary, columns=te.columns_) #위에서 나온걸 보기 좋게 데이터프레임으로
```

변경

df

#### 8. 지지도 0.1 이상인 아이템 목록을 출력

```
frequent_itemsets = apriori(df, min_support=0.1, use_colnames=True)
print(frequent_itemsets )
```

#### 9. 아이템간의 연관성을 출력합니다.

```
from mlxtend.frequent_patterns import association_rules
result = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3)
# print( association_rules(frequent_itemsets, metric="confidence", min_threshold=0.3) )
result # 신뢰도 기준 0.3 이상 출력해라.
# 생수와 사과는 같은 곳에 진열을 해야 한다.
```

#### 결과 해석 : 당구장과 일반음식점이 연관성이 높고 병원과 약국이 연관성이

#### 높아보여 관련성이 높다고 할 수 있습니다.

## 자료형을 frozen set에서 list로 변경합니다.

```
result["antecedents"] = result["antecedents"].apply(lambda x: ', '.join(list(x))).astype("unicode")
result["consequents"] = result["consequents"].apply(lambda x: ', '.join(list(x))).astype("unicode")
```

#### 리스트 메소드안에 str.contains 을 이용해서 보습학원만 검색합니다.

```
result[result['antecedents'].str.contains('보습학원')|result['consequents'].str.contains('보습학원')]
```

```
result[result.antecedents.apply(str).str.contains("보습학
원")|result.consequents.apply(str).str.contains("보습학원")].sort_values('lift', ascending=False)
```

## ■ 10장. kmeans 알고리즘

[ 빅데이터 기사 필기 시험 대비 ] 수제비 3-60

주어진 데이터를 k개의 군집으로 묶는 알고리즘으로 k개 만큼 군집수를 초기값으로 지정하고 각 개체를 가까운 초기값에 할당하여 군집을 형성하고 각 군집의 평균을 재 계산하여 초기값을 갱신하는 과정을 반복하여 k개의 최종군집을 형성한다.

군집 절차 순서 :

1. k개 객체 선택
2. 할당( Assignment )
3. 중심갱신

#### 4. 반복

k-means의 하이퍼 파라미터 ? k값 (사용자가 직접 정해줌)

비지도 학습 ? 비지도학습은 입력 데이터에 대한 정답인 레이블(label)이 없는 상태에서 데이터가 어떻게 구성되었는지 알아내는 기계학습 기법이다.

##### 1. k-means

2. SOM( 자기 조직화 지도 / Self Organization Map ) : 인공신경망을 활용한 비지도학습

### ■ k-means를 파이썬으로 구현하는 예제 총정리

```
from pandas import Series, DataFrame  
import numpy as np
```

```
df = DataFrame({ "x": [3,1,7,5,6,4,9,7,6,2], "y": [4,5,9,4,8,5,8,8,7,1] },  
index=['A','B','C','D','E','F','G','H','I','J'])  
df
```

# 결과 :

	x	y
A	3	4
B	1	5
C	7	9
D	5	4
E	6	8
F	4	5
G	9	8
H	7	8
I	6	7
J	2	1

### 문제49. 다음의 예제를 이용해서 k-means 분류모델을 만들고 시각화 하시오 !

예제:

```
from pandas import Series, DataFrame  
import numpy as np
```

```
data = DataFrame({ "x": [10,1,10,7,3,1,6], "y": [9,4,1,10,10,1,7] },
```

```
index=['APPLE','BACON','BANANA','CARROT','CELERY','CHEESE','TOMATO'])  
data
```

결과해석: 토마토는 사과, 당근, 샐러리와 같은 군집으로 분류되었다.

## 문제50. UCI 데이터 중 식품에 관련한 데이터를 k-means 알고리즘으로 분류하시오!

k값을 5로 주어서 크게 5개의 군집으로 식품을 분류한다.

```
# -*- coding: utf-8 -*  
### 기본 라이브러리 불러오기  
import pandas as pd  
import matplotlib.pyplot as plt
```

...

[Step 1] 데이터 준비

...

```
# Wholesale customers 데이터셋 가져오기 (출처: UCI ML Repository)  
uci_path = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20customers%20data.csv'  
df = pd.read_csv(uci_path, header=0)  
df
```

### 데이터셋 설명 : UCI 머신러닝 저장소에서 제공하는 도매업 고객 데이터셋이다. 각 고객의 연간 구매금액을 상품 카테고리별로 구분하여 정리한 데이터이다. 모두 8개의 컬럼에 440개의 행으로 되어있고 첫번째 2개의 컬럼은 상품 구매고객이 아니라 고객의 일반정보이다. channel 컬럼은 호텔/레스토랑 또는 소매점 등의 판매채널이고 region은 고객의 소재지를 나타낸다.

#### channel : 판매채널

#### region : 고객의 소재지

#### Fresh : 냉장식품

#### Milk : 유제품

#### Grocery : 식자재

#### Frpzen : 냉동식품

#### Detergents\_Paper : 세제

#### Delicassen : 햄과 같은 육가공 식품

...

[Step 2] 데이터 탐색

...

```
# 데이터 살펴보기
```

```
print(df.head())
print('₩n')

# 데이터 자료형 확인
print(df.info())
print('₩n')

# 데이터 통계 요약정보 확인
print(df.describe())
print('₩n')

...
[Step 3] 데이터 전처리
...

# 분석에 사용할 속성을 선택
X = df.iloc[:, :]
print(X[:5])
print('₩n')

# 설명 변수 데이터를 정규화
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

print(X[:5])
print('₩n')

...
[Step 4] k-means 군집 모형 - sklearn 사용
...

# sklearn 라이브러리에서 cluster 군집 모형 가져오기
from sklearn import cluster

# 모형 객체 생성
kmeans = cluster.KMeans(init='k-means++', n_clusters=5, n_init=10)

# 모형 학습
kmeans.fit(X)

# 예측 (군집)
cluster_label = kmeans.labels_
print(cluster_label)
print('₩n')

# 예측 결과를 데이터프레임에 추가
df['Cluster'] = cluster_label
print(df.head())
```

```

print('n')

# 그래프로 표현 - 시각화
df.plot(kind='scatter', x='Grocery', y='Frozen', c='Cluster', cmap='Set1',
        colorbar=False, figsize=(10, 10))
df.plot(kind='scatter', x='Milk', y='Delicassen', c='Cluster', cmap='Set1',
        colorbar=True, figsize=(10, 10))
plt.show()
plt.close()

# 큰 값으로 구성된 클러스터(0, 4)를 제외 - 값이 몰려 있는 구간을 자세하게 분석 (데이터가 많은
값들)
mask = (df['Cluster'] == 0) | (df['Cluster'] == 4)
ndf = df[~mask]

ndf.plot(kind='scatter', x='Grocery', y='Frozen', c='Cluster', cmap='Set1',
         colorbar=False, figsize=(10, 10))
ndf.plot(kind='scatter', x='Milk', y='Delicassen', c='Cluster', cmap='Set1',
         colorbar=True, figsize=(10, 10))
plt.show()
plt.close()

```

## 문제51. 유방암 데이터를 k-means로 분류하는데 파이썬을 이용해서 분류하시오!

```

# -*- coding: utf-8 -*-
### 기본 라이브러리 불러오기
import pandas as pd
import matplotlib.pyplot as plt

...
[Step 1] 데이터 준비
...

# Wholesale customers 데이터셋 가져오기 (출처: UCI ML Repository)

from pandas import Series, DataFrame
import numpy as np

df = pd.read_csv("d://data/wisc_bc_data.csv")
df.info()

...
[Step 2] 데이터 탐색
...

# 데이터 살펴보기
print(df.head())

```

```
print('₩n')

# 데이터 자료형 확인
print(df.info())
print('₩n')

# 데이터 통계 요약정보 확인
print(df.describe())
print('₩n')

...
[Step 3] 데이터 전처리
...

# 분석에 사용할 속성을 선택
X = df.iloc[:, 2:32]
print(X)
print('₩n')

# 설명 변수 데이터를 정규화
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X)

print(X[:5])
print('₩n')

...
[Step 4] k-means 군집 모형 - sklearn 사용
...

# sklearn 라이브러리에서 cluster 군집 모형 가져오기
from sklearn import cluster

# 모형 객체 생성
kmeans = cluster.KMeans(init='k-means++', n_clusters=2, n_init=10)

# 모형 학습
kmeans.fit(X)

# 예측 (군집)
cluster_label = kmeans.labels_
print(cluster_label)
print('₩n')

# 예측 결과를 데이터프레임에 추가
df['Cluster'] = cluster_label
print(df.head())
print('₩n')
```

```

# 그래프로 표현 - 시각화
df.plot(kind='scatter', x='radius_mean', y='texture_mean', c='Cluster', cmap='Set1',
        colorbar=True, figsize=(10, 10))
plt.show()
plt.close()

##### 양성 종양이면 0이고 악성 종양이면 1을 출력하는 result라는 컬럼 생성

def func(n):
    if n=='B':
        return 0
    else:
        return 1

df['result'] = df['diagnosis'].apply(func)
df

sum(df.result==df.Cluster)/len(df)

```

\* R을 활용한 머신러닝에서 수업했던 비지도학습 실습 2가지

1. k-means
2. SOM : 비지도학습 + 신경망

## 문제52. (점심시간 문제) 아래의 사이트를 참고해서 iris 데이터를 사이킷런의 SOM 패키지로 군집화하는 실습을 수행하시오!

1. 아나콘다 프롬프트 창을 열고 pip install sklearn-som 다운로드
2. 아래 사이트의 예제를 수행하고 결과를 올리시오!

<https://pypi.org/project/sklearn-som/>

```
from sklearn_som.som import SOM
```

```
from sklearn import datasets
```

```
iris = datasets.load_iris()
iris_data = iris.data[:, :2]
iris_label = iris.target
```

```
print(iris_data)
print(iris_label)
```

```
print(iris)
```

```

iris_som = SOM(m=3, n=1, dim=2) #SOM 모델생성
# 설명 : dim이 input 데이터의 shape, m이 뉴런의 개수, n이 층의 개수
iris_som.fit(iris_data)

predictions = iris_som.predict(iris_data)

predictions

result = []
for i in predictions:
    result.append(i)

print(result)

sum(iris_label == predictions) /len(iris_data)

```

## ■ 11장. 랜덤 포레스트

[ 빅데이터 기사 필기시험 ] 수제비 3-97

랜덤포레스트는 의사결정나무의 특징인 분산이 크다는 점을 고려하여 배깅과 부스팅보다 더 많은 무작위성을 주어 약한 학습기들을 생성한 후 이를 선형 결합하여 최종 학습기를 만드는 방법이다.

### 예제1. iris 데이터를 의사결정트리로 분류하는 실습 + gridsearch 기능 추가

#### 1. 필요한 패키지를 로드합니다.

```

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
import pandas as pd # 데이터 전처리를 위해서
import seaborn as sns # 시각화를 위해서

```

#### 2. iris2.csv를 불러와서 독립변수들과 종속변수를 생성합니다.

```

col_names = ['sepal-length', 'sepal-width','petal-length', 'petal-width','Class']

df = pd.read_csv('d:\data\iris2.csv', encoding='UTF-8', header=None, names=col_names)

X = df.iloc[:, :-1].to_numpy() # 독립변수와 종속변수는 다 numpy array해줘야함

```

```
y = df.iloc[:, -1].to_numpy()  
X
```

#### 3. 훈련데이터와 테스트 데이터를 8:2로 분리합니다.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,  
                                                test_size=0.2, c=121)  
print(X_train.shape)  
print(X_test.shape)
```

#### 4. 의사결정트리 모델을 생성합니다.

```
dtree = DecisionTreeClassifier()
```

#### 5. 의사결정 트리의 gridsearch 모델을 생성합니다.

```
### parameter 들을 dictionary 형태로 설정  
parameters = {'max_depth':[1,2,3,4], 'min_samples_split':[2,3,4]}
```

```
# param_grid의 하이퍼 파라미터들을 3개의 train, test set fold 로 나누어서 테스트 수행 설정.  
### refit=True 가 default 임. True이면 가장 좋은 파라미터 설정으로 재 학습 시킴.  
grid_dtree = GridSearchCV(dtree, param_grid=parameters, cv=3, refit=True)
```

```
# 봇꽃 Train 데이터로 param_grid의 하이퍼 파라미터들을 순차적으로 학습/평가 .  
grid_dtree.fit(X_train, y_train)
```

```
# GridSearchCV 결과 추출하여 DataFrame으로 변환  
scores_df = pd.DataFrame(grid_dtree.cv_results_)  
scores_df[['params', 'mean_test_score', 'rank_test_score', #  
          'split0_test_score', 'split1_test_score', 'split2_test_score']]
```

```
from sklearn.metrics import accuracy_score
```

```
print('GridSearchCV 최적 파라미터:', grid_dtree.best_params_)  
print('GridSearchCV 최고 정확도: {:.4f}'.format(grid_dtree.best_score_))
```

```
# GridSearchCV의 refit으로 이미 학습이 된 estimator 반환  
estimator = grid_dtree.best_estimator_
```

```
# GridSearchCV의 best_estimator_는 이미 최적 하이퍼 파라미터로 학습이 됨  
pred = estimator.predict(X_test)  
print('테스트 데이터 세트 정확도: {:.4f}'.format(accuracy_score(y_test, pred)))
```

예제2. iris 데이터를 랜덤포레스트로 분류하는 실습 + gridsearch 기능 추가

#### 1. 필요한 패키지를 로드합니다.

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
import pandas as pd # 데이터 전처리를 위해서
from sklearn.ensemble import RandomForestClassifier
```

#### 2. 독립변수들과 종속변수를 생성합니다.

```
col_names = ['sepal-length', 'sepal-width','petal-length', 'petal-width','Class']

df = pd.read_csv('d:\data\iris2.csv', encoding='UTF-8', header=None, names=col_names)

X = df.iloc[:, :-1].to_numpy()
y = df.iloc[:, -1].to_numpy()
print(X.shape) # (150, 4)
print(y.shape) # (150,)
```

#### 3. 훈련데이터와 테스트 데이터를 8:2로 나눕니다.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=121)

dtree = DecisionTreeClassifier()
rftree = RandomForestClassifier()
```

#### 4. 랜덤포레스트 모델을 생성합니다.

### parameter 들을 dictionary 형태로 설정

```
parameters = {
    'bootstrap': [True], # 샘플링해서 가져올지 전부를 다 가져올지 결정
    'max_depth': [80, 90, 100], #나무의 깊이
    'max_features': [2, 3], # 가장 성능이 좋은 트리의 최종 개수
    'min_samples_leaf': [3, 4, 5], # 가지의 리프의 최소 개수
    'min_samples_split': [8, 10, 12],# 가지의 split의 개수
    'n_estimators': [100, 200, 300, 1000] #나무의 개수
}
```

```
# param_grid의 하이퍼 파라미터들을 3개의 train, test set fold 로 나누어서 테스트 수행 설정.
### refit=True 가 default 임. True이면 가장 좋은 파라미터 설정으로 재 학습 시킴.
grid_dtree = GridSearchCV(rftree, param_grid=parameters, cv=3, refit=True, n_jobs = -1,
                           verbose = 2 )
```

```
# 봇꽃 Train 데이터로 param_grid의 하이퍼 파라미터들을 순차적으로 학습/평가 .
grid_dtree.fit(X_train, y_train)
```

```
# GridSearchCV 결과 추출하여 DataFrame으로 변환
scores_df = pd.DataFrame(grid_dtree.cv_results_)
```

```

scores_df[['params', 'mean_test_score', 'rank_test_score', '#  

'split0_test_score', 'split1_test_score', 'split2_test_score']]  
  

from sklearn.metrics import accuracy_score  
  

print('GridSearchCV 최적 파라미터:', grid_dtree.best_params_)  

print('GridSearchCV 최고 정확도: {:.4f}'.format(grid_dtree.best_score_))  
  

# GridSearchCV의 refit으로 이미 학습이 된 estimator 반환  

estimator = grid_dtree.best_estimator_  
  

# GridSearchCV의 best_estimator_는 이미 최적 하이퍼 파라미터로 학습이 됨  

pred = estimator.predict(X_test)  

print('테스트 데이터 세트 정확도: {:.4f}'.format(accuracy_score(y_test,pred)))

```

### #예제3. seaborn의 타이타닉 데이터로 랜덤포레스트 모델 생성하기

#[쉬움주의] 시본 타이타닉의 랜덤포레스트

```

from sklearn import metrics  

import numpy as np  
  

# 1단계 csv ---> 데이터 프레임으로 변환  
  

import pandas as pd  

import seaborn as sns  
  

df = sns.load_dataset('titanic')  
  

# 컬럼이 모두다 출력될 수 있도록 출력할 열의 개수 한도를 늘리기  

pd.set_option('display.max_columns',15)  
  

#파생변수를 생성한다.  
  

mask4 = (df.age<10) | (df.sex=='female')  

df['child_women']=mask4.astype(int)  
  

# 2.2 결측치(NaN) 을 확인한다.  

# 2.3 deck 컬럼과 embark_town 컬럼을 삭제한다.  

# 설명 : deck 결측치가 많아서 컬럼을 삭제해야함.  

#         embark 와 embark_town 이 같은 데이터여서 embark 컬럼을 삭제해야함  
  

rdf = df.drop(['deck','embark_town'], axis =1)

```

```
rdf
```

```
# 2.4 age(나이) 열에 나이가 없는 모든행을 삭제한다.
```

```
# 데이터가 한개라도 없으면 drop 해라 (how = 'any')
```

```
# 모든 데이터가 없으면 drop 해라 (how = 'all')
```

```
rdf = rdf.dropna( subset=['age'], how='any', axis=0)
```

```
len(rdf) #714
```

```
rdf
```

```
# 2.5 embark 열의 NaN 값을 승선도시중 가장 많이 출현한 값으로 치환하기
```

```
most_freq = rdf['embarked'].value_counts().idxmax()
```

```
rdf['embarked'].fillna(most_freq, inplace = True)
```

```
# 3단계 범주형 데이터를 숫자형으로 변환하기
```

```
# 3.1 feature selection (분석에 필요한 속성을 선택)
```

```
ndf = rdf[['survived','pclass','sex','age','sibsp','parch','embarked','child_women']]
```

```
ndf
```

```
# 선택된 컬럼중 2개(sex, embarked) 가 범주형이다.
```

```
#3.2 범주형 데이터를 숫자로 변환하기(원핫 인코딩)
```

```
gender = pd.get_dummies(ndf['sex'])
```

```
ndf = pd.concat([ndf,gender], axis= 1)
```

```
onehot_embarked = pd.get_dummies(ndf['embarked'])
```

```
ndf = pd.concat([ndf,onehot_embarked],axis=1)
```

```
ndf.drop(['sex','embarked'], axis=1, inplace = True)
```

```
ndf
```

```
# 4단계 정규화
```

```
# 4.1 독립변수와 종속변수(라벨) 을 지정한다.
```

```
# survived pclass age sibsp parch female male C Q S
```

```
# 라벨 데이터
```

```
# 종속변수 독립변수
```

```
x = ndf[ ['pclass', 'age' , 'sibsp', 'parch' , 'female' , 'male', 'C' , 'Q' , 'S', 'child_women'] ]
```

```
y = ndf['survived'] # 종속변수
```

```
# 4.2 독립변수들을 표준화 한다.
```

```
from sklearn import preprocessing
```

```
X = preprocessing.StandardScaler().fit(x).transform(x)
```

```
X
```

```
# 5단계 훈련 데이터를 훈련 데이터 / 테스트 데이터로 나눈다
```

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,  
                                              random_state = 33)
```

## # 6. sklearn 라이브러리에서 나이브베이즈 분류 모형 가져오기

```
from sklearn.ensemble import RandomForestClassifier
```

```
tree_model = RandomForestClassifier( n_estimators=100,  
                                    oob_score=True,  
                                    random_state= 9 )
```

```
tree_model.fit( X_train, y_train )
```

```
print ( tree_model.oob_score_) #랜덤포레스트의 평가지표
```

## # 7단계 테스트 데이터로 예측을 한다.

```
y_hat = tree_model.predict( X_test )  
y_hat
```

## # 8단계 모형의 예측능력을 평가한다.

```
from sklearn import metrics
```

```
randomforest_matrix = metrics.confusion_matrix( y_test, y_hat )
```

```
print( randomforest_matrix )
```

```
tn, fp, fn, tp = metrics.confusion_matrix( y_test, y_hat ).ravel()
```

```
f1_report = metrics.classification_report( y_test, y_hat )
```

```
print( f1_report )
```

```
#print(np.array([[tp,fp],[fn,tn]]))
```

```
from sklearn.metrics import accuracy_score  
accuracy = accuracy_score( y_test, y_hat)  
print(accuracy)
```

**문제53. 예제3 seaborn의 타이타닉 랜덤 포레스트 모델의 oob점수는 0.74였고 정확도는 0.80이었다. 예제2번에서 사용한 gridsearch 코드를 가져다가 추가하고 이 모델의 성능을 더 올리시오!**

## ■ 12장. 캐글 도전

# [쉬움주의] 현재 ongoing 중인 케글 타이타닉 로지스틱 회귀 모델 만들기

seaborn의 타이타닉 -> 신경망 : 0.86

SQL 도전: 0.78947

파이썬 첫번째 로지스틱 회귀 모델: 0.77511

파이썬 두번째 도전 : 로지스틱 회귀 모델 + gridsearch : 0.78468

파이썬 세번째 도전 : 로지스틱 회귀 모델 + gridsearch : 0.78468

1. 신경망 모델 + grid search
2. 나이의 결측치를 SQL때처럼 호칭의 평균값으로 치환
3. 랜덤포레스트모델 + grid search
4. 앙상블: xgboost모델 + grid search
5. adaboost, gradient boost + grid search

**문제54. 파이썬 두번째 도전 : 로지스틱 회귀 모델 + gridsearch를 구현해서 캐글의 점수를 확인하시오!**

# [쉬움주의] 케글 타이타닉 로지스틱 회귀

```
from sklearn import metrics  
import numpy as np
```

# 1단계 csv ---> 데이터 프레임으로 변환

```
import pandas as pd  
import seaborn as sns  
  
df = pd.read_csv("d://data//titanic//train.csv")  
df
```

```
# 컬럼이 모두다 출력될 수 있도록 출력할 열의 개수 한도를 늘리기  
pd.set_option('display.max_columns',15)
```

# 2단계 결측치 확인하고 제거하거나 치환한다.

# 2.1 타이타닉 데이터 프레임의 자료형을 확인한다.

```
mask4 = (df.Age<10) | (df.Sex=='female')  
df['child_women']=mask4.astype(int)  
df
```

```

print ( df.columns)

# embark 와 embark_town 이 같은 데이터여서 embark 컬럼을 삭제해야함

rdf = df.drop(['PassengerId','Cabin','Name','Ticket'], axis =1)
print(rdf)

#### 결측치가 있는 컬럼을 확인합니다.

df.isnull().sum()

# 2.4 age(나이) 열에 나이가 없는 모든행을 삭제한다.

# 데이터가 한개라도 없으면 drop 해라 (how = 'any')
# 모든 데이터가 없으면 drop 해라 (how = 'all')

print(rdf.shape)

#rdf = rdf.dropna( subset=['Age'], how='all', axis=0)
#rdf['Age']=rdf['Age'].fillna(method='ffill', inplace=True)

#most_freq = rdf['Age'].value_counts(dropna=True).idxmax()
#most_freq

rdf['Age'].fillna(31, inplace=True)
print(rdf.shape)

# 2.5 embark 열의 NaN 값을 승선도시중 가장 많이 출현한 값으로 치환하기
most_freq = rdf['Embarked'].value_counts().idxmax()

rdf['Embarked'].fillna(most_freq, inplace = True)

# 2.6 fare 의 이상치를 제거합니다.

local_std = rdf.Fare.std() * 5
print(local_std)

rdf = rdf[[:] [ rdf['Fare'] < local_std ]]

rdf #882건

# 3단계 범주형 데이터를 숫자형으로 변환하기

# 3.1 feature selection (분석에 필요한 속성을 선택)
#ndf = rdf[['Survived','Pclass','Sex','Age','Sibsp','Parch','Embarked','child_women']]

ndf = rdf

ndf

# 선택된 컬럼중 2개(sex, embarked) 가 범주형이다.

```

### #3.2 범주형 데이터를 숫자로 변환하기(원핫 인코딩)

```
gender = pd.get_dummies(ndf['Sex'])
ndf = pd.concat([ndf,gender], axis= 1)
onehot_embarked = pd.get_dummies(ndf['Embarked'])
ndf = pd.concat([ndf,onehot_embarked],axis=1)
ndf.drop(['Sex','Embarked'], axis=1, inplace = True)

print(ndf.columns)
```

ndf

### # 4단계 정규화

# 4.1 독립변수와 종속변수(라벨) 을 지정한다.

```
# survived pclass age sibsp parch female male C Q S
# 라벨          데이터
# 종속변수      독립변수
```

```
print(ndf.columns)
x = ndf[ ['Fare', 'Pclass', 'Age' , 'SibSp', 'Parch' , 'female' , 'male', 'C' , 'Q' , 'S','child_women'] ]
#x = ndf[ ['Fare', 'Pclass', 'Age' , 'SibSp', 'Parch' , 'female' , 'male', 'child_women', 'Parch'] ]

y = ndf['Survived'] # 종속변수
print(x)
```

### # 4.2 독립변수들을 정규화 한다.

```
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(x).transform(x)
X
```

### # 7단계 테스트 데이터로 예측을 한다.

```
x_ktest = pd.read_csv("d:\data\titanic\test.csv")
x_ktest
```

#### 테스트 데이터도 훈련 데이터와 똑같이 파생변수를 추가한다.

```
mask4 = (x_ktest.Age<10) | (x_ktest.Sex=='female')
x_ktest['child_women']=mask4.astype(int)
x_ktest
```

### # 테스트 데이터도 필요없는 컬럼을 삭제한다.

```
rdf_x_ktest = x_ktest.drop(['PassengerId','Cabin','Name','Ticket'], axis =1)
print(rdf_x_ktest)
```

### # 테스트 데이터의 나이의 결측치를 21로 채워넣는다.

```
rdf_x_ktest['Age'].fillna(21, inplace=True)
rdf_x_ktest.isnull().sum()
```

```
# 2.5 embark 열의 NaN 값을 승선도시중 가장 많이 출현한 값으로 치환하기
```

```
most_freq = rdf_x_ktest['Embarked'].value_counts().idxmax()
rdf_x_ktest['Embarked'].fillna(most_freq, inplace = True)
print(rdf_x_ktest)
```

```
# 2.6 fare 열의 NaN 값을 요금중 가장 많이 출현한 값으로 치환하기
```

```
most_freq = rdf_x_ktest['Fare'].value_counts().idxmax()
rdf_x_ktest['Fare'].fillna(most_freq, inplace = True)
print(rdf_x_ktest)
```

```
# 3단계 범주형 데이터를 숫자형으로 변환하기
```

```
# 3.1 feature selection (분석에 필요한 속성을 선택)
```

```
#ndf = rdf[['Survived','Pclass','Sex','Age','Sibsp','Parch','Embarked','child_women']]
```

```
ndf_x_ktest = rdf_x_ktest
ndf_x_ktest
```

```
# 선택된 컬럼중 2개(sex, embarked) 가 범주형이다.
```

```
#3.2 범주형 데이터를 숫자로 변환하기(원핫 인코딩)
```

```
gender = pd.get_dummies(ndf_x_ktest['Sex'])
ndf_x_ktest = pd.concat([ndf_x_ktest,gender], axis= 1)
onehot_embarked = pd.get_dummies(ndf_x_ktest['Embarked'])
ndf_x_ktest = pd.concat([ndf_x_ktest,onehot_embarked],axis=1)
ndf_x_ktest.drop(['Sex','Embarked'], axis=1, inplace = True)
```

```
ndf_x_ktest
```

```
# 4단계 정규화
```

```
# 4.1 독립변수와 종속변수(라벨) 을 지정한다.
```

```
# survived pclass age sibsp parch female male C Q S
```

```
# 라벨 데이터
```

```
# 종속변수 독립변수
```

```
print(ndf_x_ktest.columns)
```

```
x = ndf_x_ktest[ ['Fare','Pclass', 'Age' , 'SibSp', 'Parch' , 'female' , 'male', 'C' , 'Q' , 'S', 'child_women'] ]
```

```
#x = ndf_x_ktest[ ['Fare','Pclass', 'Age' , 'SibSp', 'Parch' , 'female' , 'male', 'child_women','Parch'] ]
```

```
print ( x.isnull().sum( axis=0) )
```

```
#y = ndf_x_ktest['Survived'] # 종속변수 / test 데이터는 종속변수 없음
```

```
# 4.2 독립변수들을 정규화 한다.
```

```
from sklearn import preprocessing
X_test = preprocessing.StandardScaler().fit(x).transform(x)
print(len(X))
print(X.shape)
print(X_test.shape)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=1000.0, random_state=0)
```

```
lr.fit( X, y)

from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV

param_grid = [
    {
        'activation' : ['identity', 'logistic', 'tanh', 'relu'],
        'solver' : ['lbfgs', 'sgd', 'adam'],
        'hidden_layer_sizes': [
            (1,2),(2,2),(3,2),(4,2),(5,2),(6,2),(7,2),(8,2),(9,2),(10,2),(11,2),
            (12,2),(13,2),(14,2),(15,2),(16,2),(17,2),(18,2),(19,2),(20,2),(21,2)
        ]
    }
]

grid_dtrees = GridSearchCV(MLPClassifier(), param_grid, cv=3, n_jobs = -1, verbose = 2 )

grid_dtrees.fit( X, y)
```

# 7단계 테스트 데이터로 예측을 한다.

```
y_hat = grid_dtrees.predict( X_test )
print(y_hat)

for i,a in enumerate(y_hat):
    print (str(i+892) + ',' + str(a))

test_id=x_ktest.PassengerId
pred = y_hat
test_id

submission = pd.DataFrame({'PassengerId':test_id,'Survived':pred})
submission.to_csv('d:\data\titanic\submission_result3.csv',index=False)
submission.head()
```