2021 SMARCLE 겨울방학 인공지능 스터디

1부

11장 〈데이터 가공하기〉

SMARCLE

04 데이터 가공하기


```
print(df[['pregnant','class']].groupby(['pregnant'], as_
index=False).mean().sort_values(by='pregnant', ascending=True))
```

Groupby -> [정보]를 기준으로 하는 새 그룹을 만든다.

As_index = False -> [정보] 옆에 새로운 index를 만들어 준다.

Sort_values -> 데이터를 오름차순 정렬

SMARCLE

05

Matplotlib를 이용해 그래프로 표현하기

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Seaborn라이브러리 : 통계 그래픽 라이브러리
흔히 사용하는 다양한 시각화 패턴을 쉽게 구현할 수 있도록 도와준다

SMARCLE

05

Matplotlib를 이용해 그래프로 표현하기

```
sns.heatmap(df.corr(), linewidths=0.1, vmax=0.5, cmap=plt.cm.gist_
heat, linecolor='white', annot=True)
```

Vmax -> 색상의 밝기를 조절

Annot -> 그래프에 숫자 값을 표현할지 여부를 선택

Cmap -> 색상의 팔레트(색)를 지정

SMARCLE

05

Matplotlib를 이용해 그래프로 표현하기

```
grid = sns.FacetGrid(df, col='class')
grid.map(plt.hist, 'plasma', bins=10)
plt.show()
```

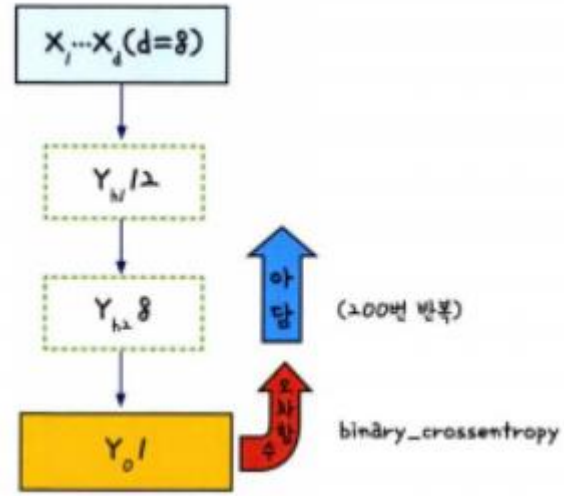
Plt.hist -> 히스토그램을 만들기

Bins -> 전체 막대의 개수

SMARCLE

06

파마 인디언의 당뇨병 예측하기




```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

은닉층 추가

```
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Loss -> MSE대신 binary_crossentropy

SMARCLE

파마 인디언의 당뇨병 예측하기

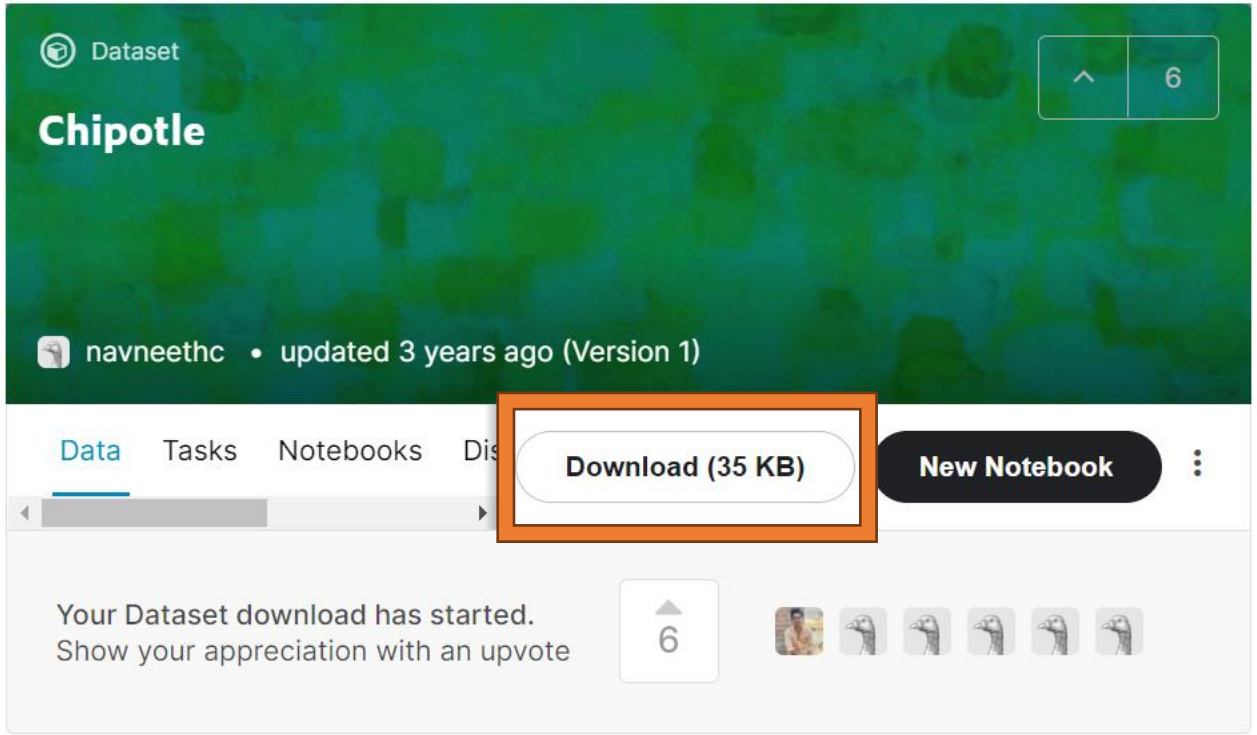
```
Epoch 178/200
768/768 [==============================] - 0s 115us/sample - loss: 0.5000 - accuracy: 0.7148
Epoch 179/200
768/768 [==============================] - 0s 191us/sample - loss: 0.4956 - accuracy: 0.7331
Epoch 180/200
768/768 [==============================] - 0s 130us/sample - loss: 0.5027 - accuracy: 0.7266
Epoch 181/200
768/768 [==============================] - 0s 107us/sample - loss: 0.4948 - accuracy: 0.7266
Epoch 182/200
768/768 [==============================] - 0s 113us/sample - loss: 0.4947 - accuracy: 0.7357
Epoch 183/200
768/768 [==============================] - 0s 120us/sample - loss: 0.5018 - accuracy: 0.7240
Epoch 184/200
768/768 [==============================] - 0s 116us/sample - loss: 0.4955 - accuracy: 0.7279
Epoch 185/200
768/768 [==============================] - 0s 118us/sample - loss: 0.4971 - accuracy: 0.7383
Epoch 186/200
768/768 [==============================] - 0s 120us/sample - loss: 0.4988 - accuracy: 0.7279
Epoch 187/200
768/768 [==============================] - 0s 116us/sample - loss: 0.4912 - accuracy: 0.7357
Epoch 188/200
768/768 [==============================] - 0s 121us/sample - loss: 0.4916 - accuracy: 0.7383
Epoch 189/200
768/768 [==============================] - 0s 118us/sample - loss: 0.4984 - accuracy: 0.7292
Epoch 190/200
768/768 [==============================] - 0s 112us/sample - loss: 0.5050 - accuracy: 0.7279
Epoch 191/200
768/768 [==============================] - 0s 113us/sample - loss: 0.4975 - accuracy: 0.7266
Epoch 192/200
768/768 [==============================] - 0s 113us/sample - loss: 0.5037 - accuracy: 0.7240
Epoch 193/200
768/768 [==============================] - 0s 126us/sample - loss: 0.5086 - accuracy: 0.7240
Epoch 194/200
768/768 [==============================] - 0s 120us/sample - loss: 0.4965 - accuracy: 0.7266
Epoch 195/200
768/768 [==============================] - 0s 116us/sample - loss: 0.5009 - accuracy: 0.7201
Epoch 196/200
768/768 [==============================] - 0s 112us/sample - loss: 0.4923 - accuracy: 0.7344
Epoch 197/200
768/768 [==============================] - 0s 117us/sample - loss: 0.4904 - accuracy: 0.7253
Epoch 198/200
768/768 [==============================] - 0s 117us/sample - loss: 0.4944 - accuracy: 0.7240
Epoch 199/200
768/768 [==============================] - 0s 118us/sample - loss: 0.4935 - accuracy: 0.7214
Epoch 200/200
768/768 [==============================] - 0s 117us/sample - loss: 0.4979 - accuracy: 0.7357
768/1 [==========================================================================================

 Accuracy: 0.7253
```

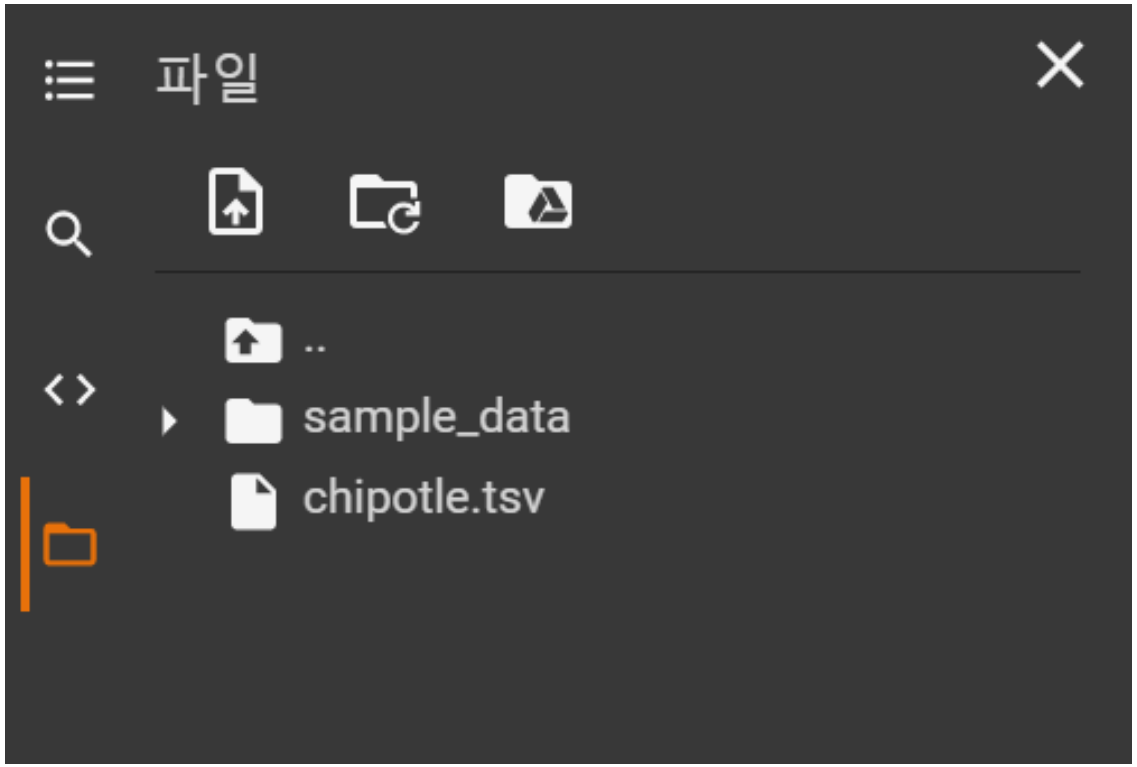
SMARCLE

04 데이터 가공하기

1. 멕시코풍 프랜차이즈 데이터 다운로드

https://www.kaggle.com/navneethc/chipotle





04

데이터 가공하기

탭문자로 열 구분

```
df = pd.read_csv("chipotle.tsv", sep="₩t")
df
```

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
...	...	...	...	...	...
4617	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Sour ...	$11.75
4618	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Sour Cream, Cheese...	$11.75
4619	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	$11.25
4620	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Lettu...	$8.75
4621	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	$8.75

4622 rows × 5 columns

데이터 가공하기

수치형 특성 확인

1. quantity
2.item_price

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   order_id            4622 non-null   int64 
 1   quantity            4622 non-null   int64 
 2   item_name           4622 non-null   object
 3   choice_description  3376 non-null   object
 4   item_price          4622 non-null   object
dtypes: int64(2), object(3)
memory usage: 180.7+ KB
```

데이터 가공하기

수치형 특성 확인

1. quantity
2.item_price

```
df["order_id"]=df["order_id"].astype(str)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   order_id            4622 non-null   object
 1   quantity            4622 non-null   int64 
 2   item_name           4622 non-null   object
 3   choice_description  3376 non-null   object
 4   item_price          4622 non-null   object
dtypes: int64(1), object(4)
memory usage: 180.7+ KB
```

데이터 가공하기

apply()를 적용하여 $를 제거

```
def to_float(v):
    return float(v[1:])
```

```
to_float("$10.5")
```

```
10.5
```

```
df["item_price"].apply(to_float)
```

```
0          2.39
1          3.39
2          3.39
3          2.39
4         16.98
          ...
4617      11.75
4618      11.75
4619      11.25
4620       8.75
4621       8.75
Name: item_price, Length: 4622, dtype: float64
```

데이터 가공하기

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4622 entries, 0 to 4621
Data columns (total 5 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   order_id            4622 non-null   int64
 1   quantity            4622 non-null   int64
 2   item_name           4622 non-null   object
 3   choice_description  3376 non-null   object
 4   item_price          4622 non-null   float64
dtypes: float64(1), int64(2), object(2)
memory usage: 180.7+ KB
```

04

데이터 가공하기

```
df["item_price"] = df["item_price"].apply(to_float)
df.head()
```

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	2.39
1	1	1	Izze	[Clementine]	3.39
2	1	1	Nantucket Nectar	[Apple]	3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	16.98

04 데이터 가공하기

.sort_values(by=<열, 열목록>, ascending=<BOOL>)

- DataFrame의 행 순서 정렬
- by: 행 순서 기준으로 사용할 열
- ascending: 오름차순 정렬(True)
  내림차순 정렬(False)

```
df.sort_values(by="item_price",ascending=False)
```

	order_id	quantity	item_name	choice_description	item_price
3598	1443	15	Chips and Fresh Tomato Salsa	NaN	44.25
3480	1398	3	Carnitas Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	35.25
1254	511	4	Chicken Burrito	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	35.00
3602	1443	4	Chicken Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	35.00
3601	1443	3	Veggie Burrito	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	33.75
...	...	...	...	...	...
3936	1578	1	Canned Soda	[Diet Dr. Pepper]	1.09
2922	1162	1	Bottled Water	NaN	1.09
1396	567	1	Canned Soda	[Coca Cola]	1.09
2562	1014	1	Canned Soda	[Coca Cola]	1.09
1457	591	1	Canned Soda	[Sprite]	1.09

4622 rows × 5 columns

데이터 가공하기

주문(order_id)기준으로 상품 가격 평균 계산

```
df.groupby("order_id")["item_price"].sum()
```

```
order_id
1        11.56
2        16.98
3        12.67
4        21.00
5        13.70
        ...
1830     23.00
1831     12.90
1832     13.20
1833     23.50
1834     28.75
Name: item_price, Length: 1834, dtype: float64
```

```
df.groupby("order_id")["item_price"].sum().mean()
```

```
18.81142857142869
```

04

데이터 가공하기

가장 비싼 주문에서 아이템이 총 몇 개 팔렸는가?

```
SUM=df.groupby("order_id").sum()
SUM
```

order_id	quantity	item_price
1	4	11.56
10	2	13.20
100	2	10.08
1000	2	20.50
1001	2	10.08
...	...	...
995	3	24.95
996	4	43.00
997	2	22.50
998	2	10.88
999	5	29.25

1834 rows × 2 columns

```
SUM.sort_values(by="item_price",ascending=False)[:5]
```

order_id	quantity	item_price
926	23	205.25
1443	35	160.74
1483	14	139.00
691	11	118.25
1786	20	114.30

04

데이터 가공하기

해당 열이름의 값들이 모두 일치하면 중복 자료로 보고 제거한다.

Chicken Bowl는 몇 번 주문했을까? (<BOOL>마스크)

```
ITEM=df[df["item_name"]=="Chicken Bowl"]
ITEM=ITEM.drop_duplicates(["item_name","order_id"])
ITEM
```

	order_id	quantity	item_name	choice_description	item_price
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	10.98
13	7	1	Chicken Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
19	10	1	Chicken Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	8.75
26	13	1	Chicken Bowl	[Roasted Chili Corn Salsa (Medium), [Pinto Bea...	8.49
...	...	...	...	...	...
4586	1824	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	11.25
4589	1825	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Sour ...	11.25
4595	1826	1	Chicken Bowl	[Tomatillo Green Chili Salsa, [Rice, Black Bea...	8.75
4599	1827	1	Chicken Bowl	[Roasted Chili Corn Salsa, [Cheese, Lettuce]]	8.75
4604	1828	1	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	8.75

615 rows × 5 columns

데이터 가공하기

Chicken Bowl를 3개 이상 주문한 횟수?

```
A=ITEM[ITEM["quantity"]>=3]
A
```

	order_id	quantity	item_name	choice_description	item_price
409	178	3	Chicken Bowl	[[Fresh Tomato Salsa (Mild), Tomatillo-Green C...	32.94
1514	616	3	Chicken Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	26.25

2021 SMARCLE 겨울방학 인공지능 스터디

2부

12장 〈다중 분류 문제〉

SMARCLE

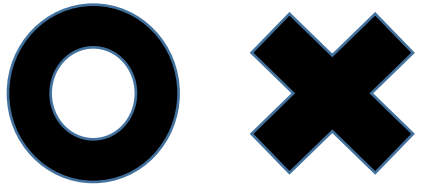
01

다중 분류 문제

이항 분류 문제

: 둘 중에 하나를 고르는 문제

	정보 1	정보 2	정보 3	…	정보 8	당뇨병 여부
1번째 인디언	6	148	72	…	50	1
2번째 인디언	1	85	66	…	31	0
3번째 인디언	8	183	64	…	32	1
…	…	…	…	…	…	…
768번째 인디언	1	93	70	…	23	0



다중 분류 문제

: 여러 답 중 하나를 고르는 문제

	정보 1	정보 2	정보 3	정보 4	품종
1번째 아이리스	5.1	3.5	4.0	0.2	Iris-setosa
2번째 아이리스	4.9	3.0	1.4	0.2	Iris-setosa
3번째 아이리스	4.7	3.2	1.3	0.3	Iris-setosa
…	…	…	…	…	…
150번째 아이리스	5.9	3.0	5.1	1.8	Iris-virginica

Iris-virginica　Iris-setosa　Iris-versicolor

SMARCLE

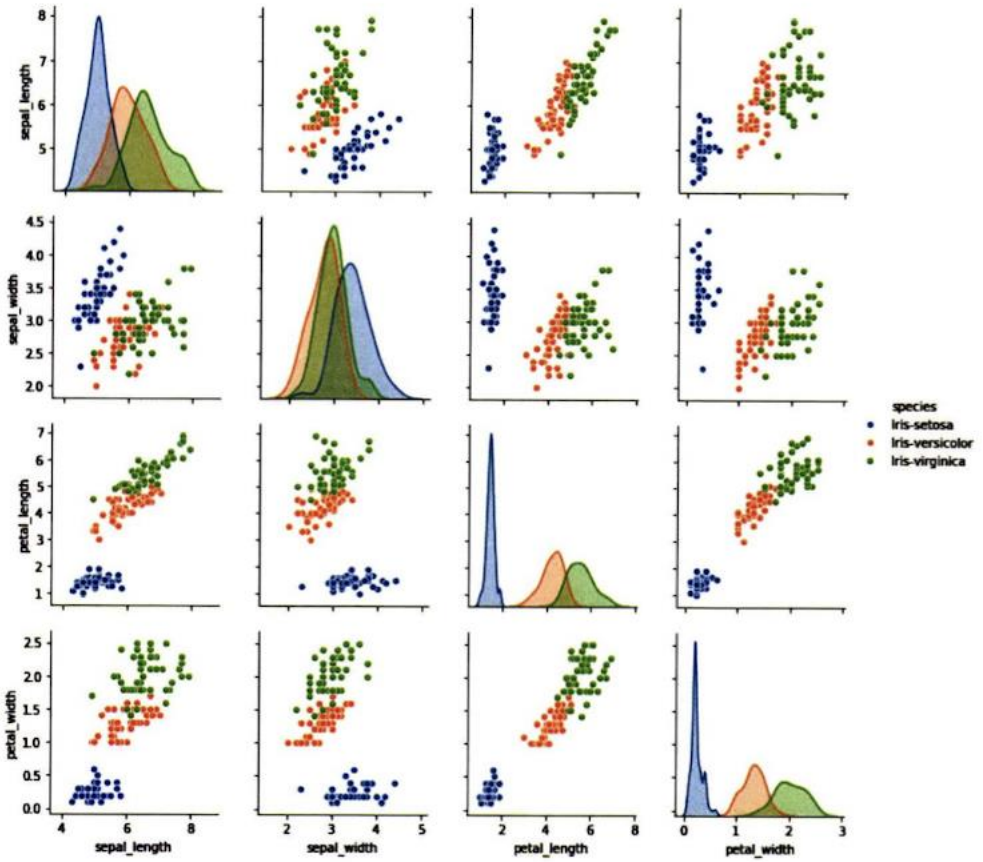
02 상관도 그래프

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5	3.6	1.4	0.2	Iris-setosa

속성별 상관관계를 알기가 어렵

품종별로 어떤 속성 차이가 있는지
한눈에 파악하기 어려움

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.pairplot(df, hue='species');
plt.show()
```



원-핫 인코딩

: 데이터를 수많은 0과 단 하나의 1의 값으로 구별하는 인코딩 방법

```
df = pd.read_csv('../dataset/iris.csv', names = ["sepal_length",
"sepal_width", "petal_length", "petal_width", "species"])

dataset = df.values
X = dataset[:,0:4].astype(float)
Y_obj = dataset[:,4]
```

1. 데이터를 X와 y로 구분

```
from sklearn.preprocessing import LabelEncoder

e = LabelEncoder()
e.fit(Y_obj)
Y = e.transform(Y_obj)
```

2. 클래스 이름을 숫자 형태로 변환

array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])

=> array([1,2,3])

```
from tensorflow.keras.utils import np_utils

Y_encoded = tf.keras.utils.to_categorical(Y)
```

3. 0과 1로 변환

array([1,2,3])

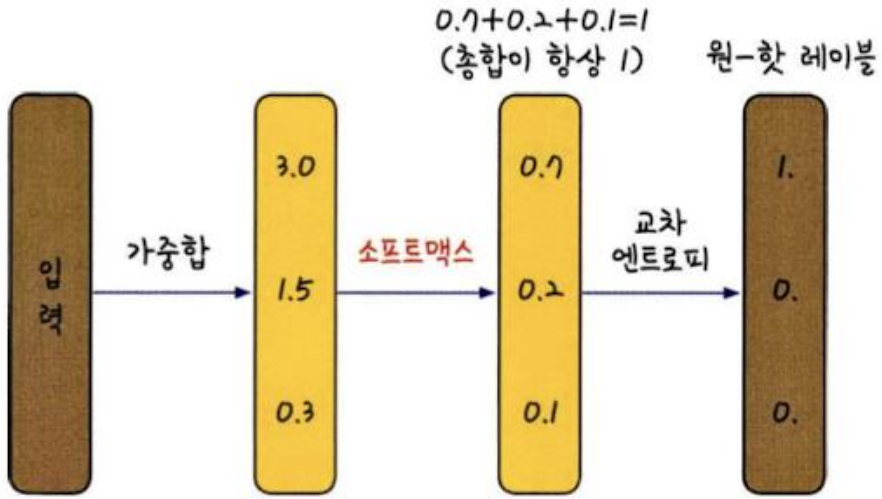
=> array([1., 0., 0.], [0., 1., 0.,], [0., 0., 1.])

SMARCLE

04 소프트맥스

```
model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Softmax : 총합이 1인 형태로 바꿔서 계산



04 코딩해보기

```
[37] data  = pd.read_csv('iris.csv', names=['sepal_length','sepal_width','petal_length','petal_width','species'])
     data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

데이터가 어떻게 이루어져 있는지 알아보자

```
[38] sns.pairplot(data,hue='species')
     plt.show()
```

각 속성에 따라 관계가 어떠한지 알 수 있다.

SMARCLE

코딩해보기

```
[39] x_data = data.iloc[:,:4]
     y_data = data.iloc[:,4]
```

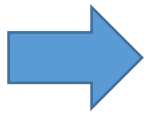
```
[40] x_data
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
y_data
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
              ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: species, Length: 150, dtype: object
```



X,Y 분류해보자

SMARCLE

04 코딩해보기

```
[42] e = LabelEncoder()
     e.fit(y_data)
     Y_fit = e.transform(y_data)
     Y_fit
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
y_encoded = to_categorical(Y_fit)
y_encoded
```

데이터 숫자화 array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'])

=> array([1,2,3])

SMARCLE

04 코딩해보기

경사 하강법의 일종

다중 분류기 때문

```
[54] model = Sequential()
     model.add(Dense(16,input_dim=4,activation='relu'))
     model.add(Dense(3,activation='softmax'))
```

```
[55] from tensorflow.keras import optimizers
     optim = optimizers.Adam(lr=0.005)
     model.compile(optimizer=optim,loss='categorical_crossentropy',metrics=['accuracy'])
```

```
[56] model.fit(x_data,y_fited,epochs=50,batch_size=1)
```

```
Epoch 43/50
150/150 [==============================] - 0s 865us/step - loss: 0.0818 - accuracy: 0.9417
Epoch 44/50
150/150 [==============================] - 0s 861us/step - loss: 0.0486 - accuracy: 0.9722
Epoch 45/50
150/150 [==============================] - 0s 829us/step - loss: 0.0641 - accuracy: 0.9735
Epoch 46/50
150/150 [==============================] - 0s 871us/step - loss: 0.1302 - accuracy: 0.9299
Epoch 47/50
150/150 [==============================] - 0s 889us/step - loss: 0.1363 - accuracy: 0.9454
Epoch 48/50
150/150 [==============================] - 0s 845us/step - loss: 0.0701 - accuracy: 0.9585
Epoch 49/50
150/150 [==============================] - 0s 877us/step - loss: 0.0617 - accuracy: 0.9821
Epoch 50/50
```

SMARCLE

코딩해보기

```
[50] Epoch 41/50
     150/150 [==============================] - 0s 914us/step - loss: 0.1069 - accuracy: 0.9600
     Epoch 42/50
     150/150 [==============================] - 0s 889us/step - loss: 0.0780 - accuracy: 0.9600
     Epoch 43/50
     150/150 [==============================] - 0s 826us/step - loss: 0.0946 - accuracy: 0.9600
     Epoch 44/50
     150/150 [==============================] - 0s 884us/step - loss: 0.0678 - accuracy: 0.9667
     Epoch 45/50
     150/150 [==============================] - 0s 931us/step - loss: 0.0702 - accuracy: 0.9733
     Epoch 46/50
     150/150 [==============================] - 0s 1ms/step - loss: 0.0735 - accuracy: 0.9600
     Epoch 47/50
     150/150 [==============================] - 0s 945us/step - loss: 0.1011 - accuracy: 0.9533
     Epoch 48/50
     150/150 [==============================] - 0s 906us/step - loss: 0.0835 - accuracy: 0.9667
     Epoch 49/50
     150/150 [==============================] - 0s 879us/step - loss: 0.0738 - accuracy: 0.9800
     Epoch 50/50
     150/150 [==============================] - 0s 936us/step - loss: 0.0660 - accuracy: 0.9667
     <tensorflow.python.keras.callbacks.History at 0x7ff00bcafcc0>
```

```
[49] print("acc : %.2f"%(model.evaluate(x_data,y_fited)[1]))

     5/5 [==============================] - 0s 2ms/step - loss: 0.0586 - accuracy: 0.9867
     acc : 0.99
```

모델 평가

SMARCLE