연습문제2 풀이

#01. 공통 처리

1. 패키지 참조

```
import seaborn as sb
from matplotlib import pyplot as plt
from pandas import DataFrame, read_excel

# 분류 알고리즘 패키지
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# 군집 알고리즘 패키지
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

# 평가함수
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import silhouette_score
```

2. 데이터 가져오기

> https://www.kaggle.com/datasets/joshmcadams/oranges-vs-grapefruit/ 의 데이터를 내려받아 엑셀로 다시 저장함

```
origin = read_excel("https://data.hossam.kr/G02/citrus.xlsx")
print(origin.info())
origin.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   name      10000 non-null  object
 1   diameter  10000 non-null  float64
 2   weight    10000 non-null  float64
 3   red       10000 non-null  int64
 4   green     10000 non-null  int64
 5   blue      10000 non-null  int64
dtypes: float64(2), int64(3), object(1)
memory usage: 468.9+ KB
None
```

	name	diameter	weight	red	green	blue
0	orange	2.96	86.76	172	85	2
1	orange	3.91	88.05	166	78	3
2	orange	4.42	95.17	156	81	2
3	orange	4.47	95.60	163	81	4
4	orange	4.48	95.76	161	72	9

3. 데이터 전처리

라벨링을 위해서 종속변수의 종류 확인

```
labels = list(origin['name'].unique())
labels
```

```
['orange', 'grapefruit']
```

라벨링 적용

```
df = origin.copy()
df['name'] = df['name'].map({'orange':0, 'grapefruit':1})
df.head()
```

	name	diameter	weight	red	green	blue
0	0	2.96	86.76	172	85	2
1	0	3.91	88.05	166	78	3
2	0	4.42	95.17	156	81	2
3	0	4.47	95.60	163	81	4
4	0	4.48	95.76	161	72	9

독립변수, 종속변수 분리

```
x = df.drop('name', axis=1)
y = df['name']
x.shape, y.shape
```

```
((10000, 5), (10000,))
```

#02. K-NN 분류

1. 분류기 만들기

> 최적의 `k` 값을 생성하기 위해서는 k가 1부터 데이터수(여기서는 10000)건까지 반복하면서 최적의 k값을 찾아야 하지만 실행속도,시간 등을 고려하여 k의 최대값을 10으로 제한함

분류 모델 구성 (k-fold)

```
k_range = range(1, 11)
k_scores = []

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x, y, cv=10, scoring='accuracy')
    k_scores.append(scores.mean())

#k_scores
```

가장 큰 정확도와 그에 대한 k값

```
max_acc = max(k_scores)
max_index = k_scores.index(max_acc)
print("최대 정확도: {0}, 최대 정확도를 갖는 k: {1}".format(max_acc, max_index+1))
```

```
최대 정확도: 0.8426, 최대 정확도를 갖는 k: 10
```

최적의 k값을 적용한 K-NN 분류기

```
knn = KNeighborsClassifier(n_neighbors=max_index+1)
knn.fit(x, y)
y_pred = knn.predict(x)
y_pred_df = DataFrame({"y": y.values, "y_pred": y_pred})
y_pred_df
```

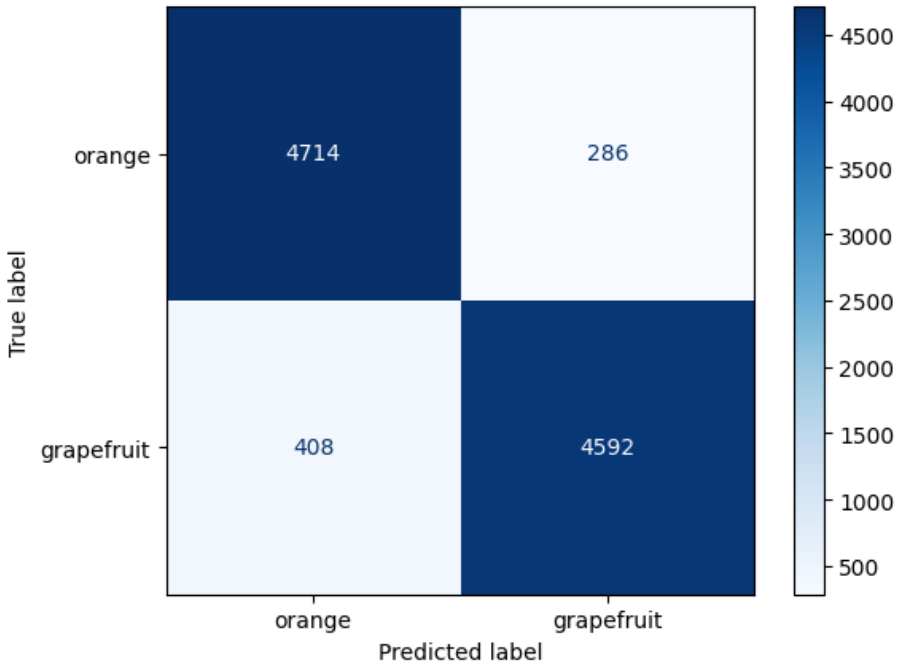
	y	y_pred
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
9995	1	1
9996	1	1
9997	1	1
9998	1	1
9999	1	1

10000 rows × 2 columns

2. 평가

혼동행렬

```
plot = ConfusionMatrixDisplay.from_predictions(
    y_pred_df['y'], y_pred_df['y_pred'],
    display_labels=labels,
    cmap=plt.cm.Blues
)

plt.show()
plt.close()
```



정확도, 정밀도, 재현율, f값

```
scores = accuracy_score(y_pred_df['y'], y_pred_df['y_pred'])
print("n_naighbors: {0}, 정확도: {1}".format(k, scores))

scores = precision_score(y_pred_df['y'], y_pred_df['y_pred'])
print("n_naighbors: {0}, 정밀도: {1}".format(k, scores))

scores = recall_score(y_pred_df['y'], y_pred_df['y_pred'])
print("n_naighbors: {0}, 재현율: {1}".format(k, scores))

scores = f1_score(y_pred_df['y'], y_pred_df['y_pred'])
print("n_naighbors: {0}, f값: {1}".format(k, scores))
```

```
n_naighbors: 10, 정확도: 0.9306
n_naighbors: 10, 정밀도: 0.941369413694137
n_naighbors: 10, 재현율: 0.9184
n_naighbors: 10, f값: 0.9297428629277181
```

#03. K-Means 군집

1. 데이터 표준화

```
scaler = MinMaxScaler()
scaler.fit(x)
n_data = scaler.transform(x)

print("각 열의 평균: ", n_data[:, 0].mean(), n_data[:, 1].mean(), n_data[:, 2].mean(),
      n_data[:, 3].mean(), n_data[:, 4].mean())
print("각 열의 최소값: ", n_data[:, 0].min(), n_data[:, 1].min(), n_data[:, 2].min(),
      n_data[:, 3].min(), n_data[:, 4].min())
print("각 열의 최대값: ", n_data[:, 0].max(), n_data[:, 1].max(), n_data[:, 2].max(),
      n_data[:, 3].max(), n_data[:, 4].max())
```

```
각 열의 평균:  0.5200656041512232 0.5052405836909871 0.5045168831168831 0.5295364705882
각 열의 최소값:  0.0 0.0 0.0 0.0 0.0
각 열의 최대값:  1.0 0.9999999999999999 0.9999999999999998 1.0 1.0
```

2. 군집 모델 만들기

inertia에 의한 최적 군집 수 찾기

```
iner = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(n_data)
    iner.append(kmeans.inertia_)

iner
```

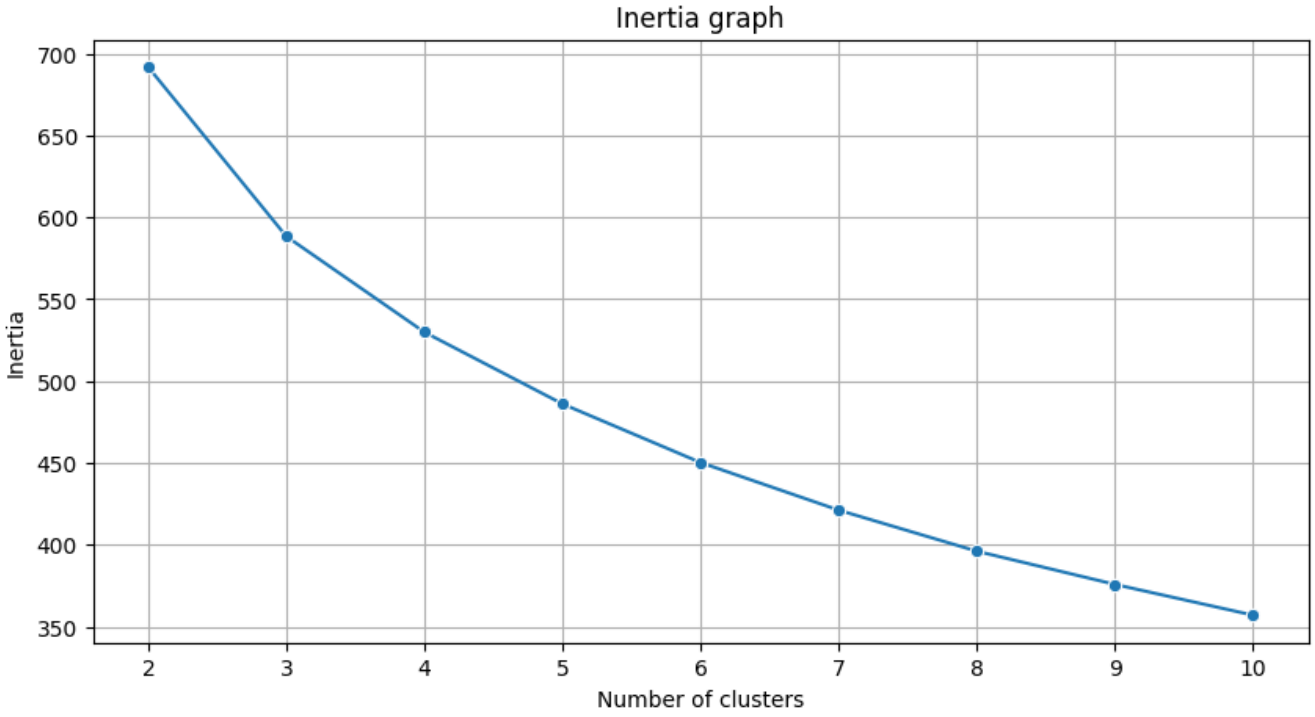
```
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
```

```
[691.7539715613655,
 588.3928583210003,
```

```
 529.8862842483146,
 486.0631965027439,
 450.2608243384459,
 421.19174298478055,
 396.1243113071512,
 375.8487932891302,
 357.07238131058807]
```

inertia 시각화

```
plt.figure(figsize=(10, 5))
sb.lineplot(x=range(2, 11), y=iner, marker='o')
plt.title("Inertia graph")
plt.xlabel("Number of clusters")
plt.ylabel("Inertia")
plt.grid()
plt.show()
plt.close()
```



실루엣 점수에 의한 판단

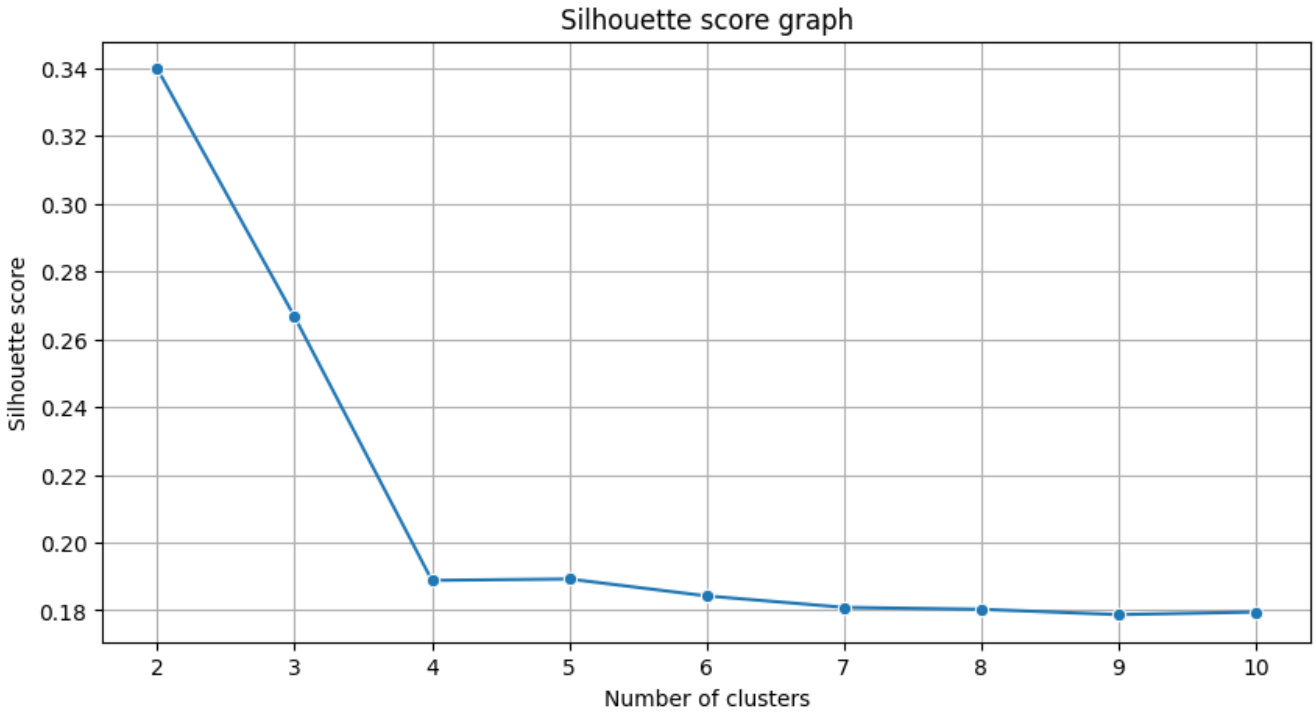
```
sil = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(n_data)
    y_pred = kmeans.predict(n_data)
    score = silhouette_score(n_data, y_pred)
    sil.append(score)

sil
```

```
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\leekh\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\cluste
  super()._check_params_vs_input(X, default_n_init=10)
```

```
[0.33987384028185325,
 0.2666267799759814,
 0.18882112674669696,
 0.18922512199942776,
 0.18422966029596058,
 0.1808585608739856,
 0.1802819264874057,
 0.17870832910080167,
 0.17946740543967518]
```

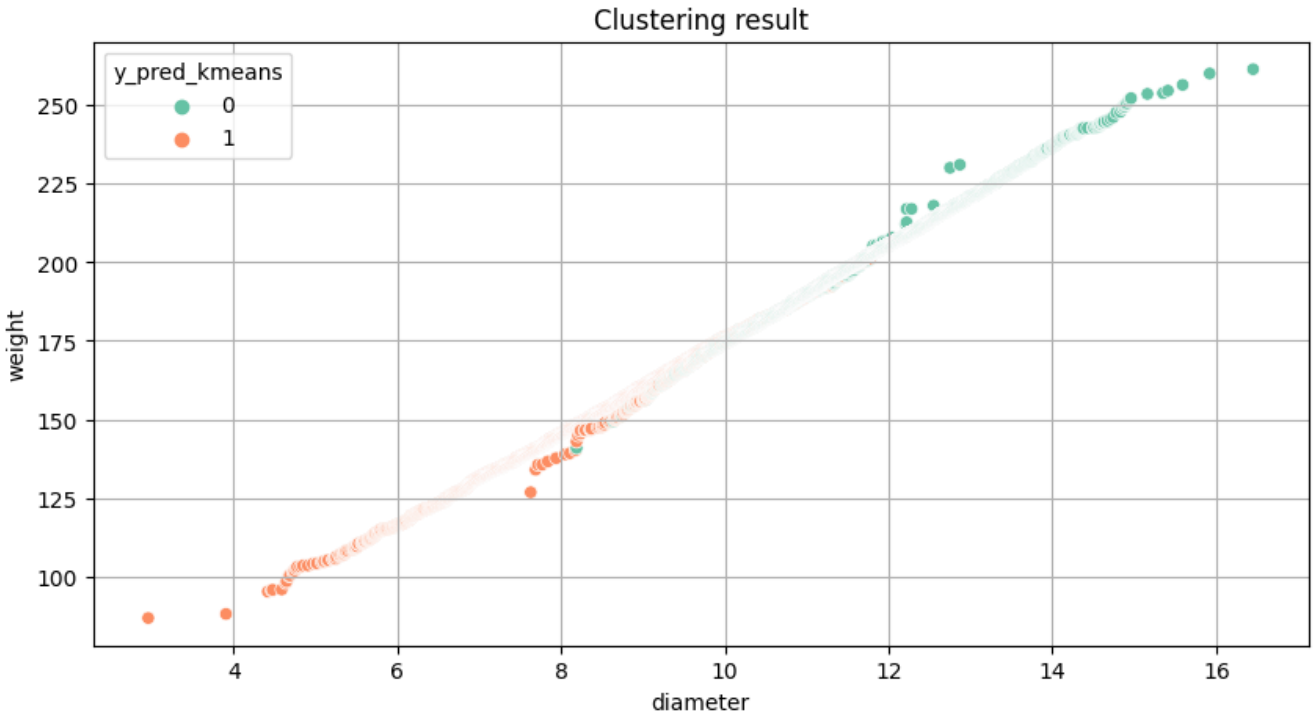
실루엣 점수 결과 시각화

```
plt.figure(figsize=(10, 5))
sb.lineplot(x=range(2, 11), y=sil, marker='o')
plt.title("Silhouette score graph")
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette score")
plt.grid()
plt.show()
plt.close()
```



군집수를 `2`로 설정하여 모델 구현 및 시각화

```
kmeans = KMeans(n_clusters=2, n_init=10, max_iter=300, random_state=777)
kmeans.fit(n_data)
y_pred = kmeans.predict(n_data)

df['y_pred_kmeans'] = y_pred

plt.figure(figsize=(10, 5))
sb.scatterplot(data=df, x='diameter', y='weight', hue='y_pred_kmeans',
               palette='Set2')
plt.title("Clustering result")
plt.xlabel("diameter")
plt.ylabel("weight")
plt.grid()
plt.show()
plt.close()
```




```
y_pred_df['y_pred_kmeans'] = y_pred
y_pred_df
```

	y	y_pred	y_pred_kmeans
0	0	0	1
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...
9995	1	1	0
9996	1	1	0
9997	1	1	0
9998	1	1	0
9999	1	1	0

10000 rows × 3 columns

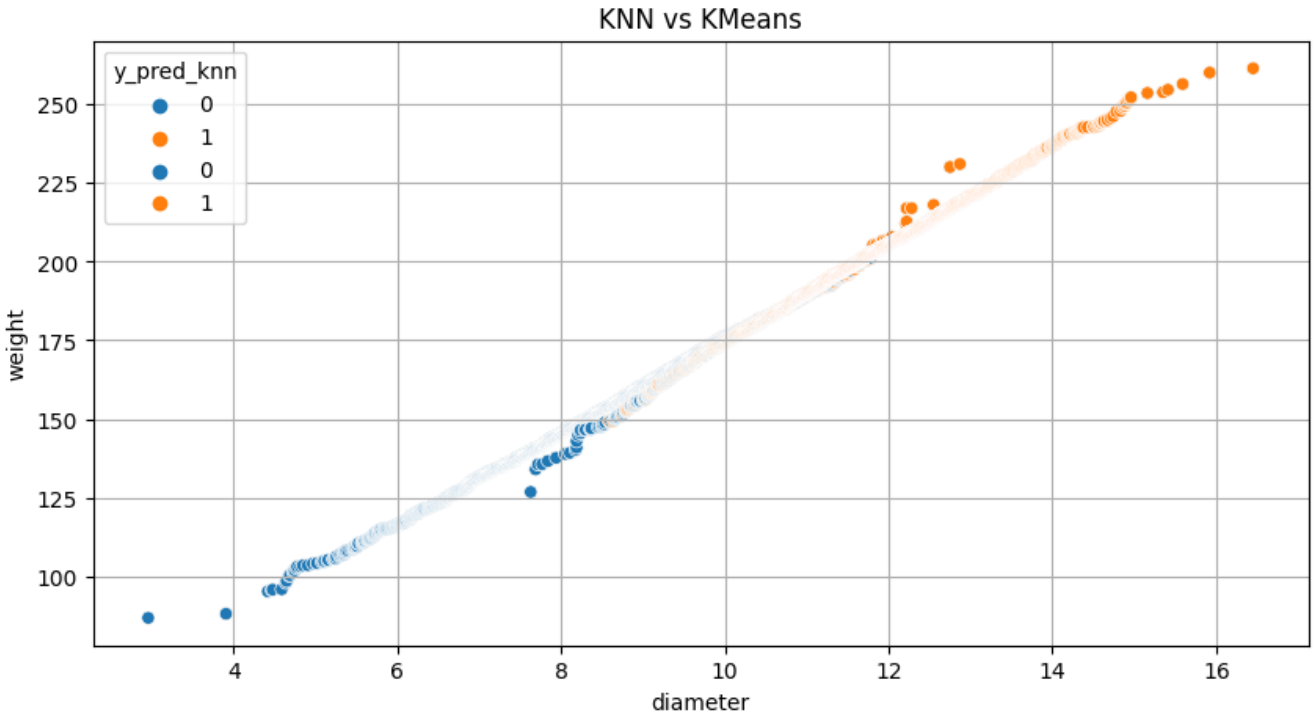
```
df['y_pred_knn'] = y_pred_df['y_pred']
df
```

	name	diameter	weight	red	green	blue	y_pred_kmeans	y_pred_knn
0	0	2.96	86.76	172	85	2	1	0
1	0	3.91	88.05	166	78	3	1	0
2	0	4.42	95.17	156	81	2	1	0

	name	diameter	weight	red	green	blue	y_pred_kmeans	y_pred_knn
3	0	4.47	95.60	163	81	4	1	0
4	0	4.48	95.76	161	72	9	1	0
...	...	...	...	...	...	...	...	...
9995	1	15.35	253.89	149	77	20	0	1
9996	1	15.41	254.67	148	68	7	0	1
9997	1	15.59	256.50	168	82	20	0	1
9998	1	15.92	260.14	142	72	11	0	1
9999	1	16.45	261.51	152	74	2	0	1

10000 rows × 8 columns

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

sb.scatterplot(data=df, x='diameter', y='weight', hue='y_pred_kmeans')
sb.scatterplot(data=df, x='diameter', y='weight', hue='y_pred_knn')

plt.title("KNN vs KMeans")
plt.xlabel("diameter")
plt.ylabel("weight")

plt.grid()
plt.show()
plt.close()
```



> KNN의 분류 결과와 KMeans의 군집 결과를 서브플롯으로 각각 시각화하여 비교해 보세요.