20장. 군집화 (Clustering)

In [1]:
```
!pip install seaborn
!pip install plotly==4.14.3
```

```
Requirement already satisfied: seaborn in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (0.11.0)
Requirement already satisfied: scipy>=1.0 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from seaborn) (1.5.2)
Requirement already satisfied: numpy>=1.15 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from seaborn) (1.19.2)
Requirement already satisfied: pandas>=0.23 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from seaborn) (1.1.3)
Requirement already satisfied: matplotlib>=2.2 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from seaborn) (3.3.2)
Requirement already satisfied: python-dateutil>=2.7.3 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from pandas>=0.23->seaborn) (2.8.
1)
Requirement already satisfied: pytz>=2017.2 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from pandas>=0.23->seaborn) (2020.1)
Requirement already satisfied: cycler>=0.10 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from matplotlib>=2.2->seaborn) (8.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from matplotlib>
=2.2->seaborn) (2.4.7)
Requirement already satisfied: certifi>=2020.06.20 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from matplotlib>=2.2->seaborn) (2020.
6.20)
Requirement already satisfied: kiwisolver>=1.0.1 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from matplotlib>=2.2->seaborn) (1.3.0)
Requirement already satisfied: six>=1.5 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from python-dateutil>=2.7.3->pandas>=0.23->seabo
rn) (1.15.0)
Requirement already satisfied: plotly==4.14.3 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (4.14.3)
Requirement already satisfied: retrying>=1.3.3 in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from plotly==4.14.3) (1.3.3)
Requirement already satisfied: six in c:₩Users₩이준용₩.conda₩envs₩data_mining₩lib₩site-packages (from plotly==4.14.3) (1.15.0)
```

1. 데이터셋 (IRIS)

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data (https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)

In [2]:
```
import requests
import os

data = requests.get("https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data")
path = os.path.join('data', 'iris.data')
with open(path, "w") as f:
  f.write(data.text)
```

1.1 데이터셋 읽기

In [3]:
```
import pandas as pd
column_names = ['sepal length', 'sepal width', 'petal length', 'petal width', 'species']
dataset = pd.read_csv(path, names=column_names)
dataset.head()
```

Out[3]:

	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

In [4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal length  150 non-null    float64
 1   sepal width   150 non-null    float64
 2   petal length  150 non-null    float64
 3   petal width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

2. 데이터 탐색

2.1 요약 통계량

In [5]:
```
dataset.describe()
```

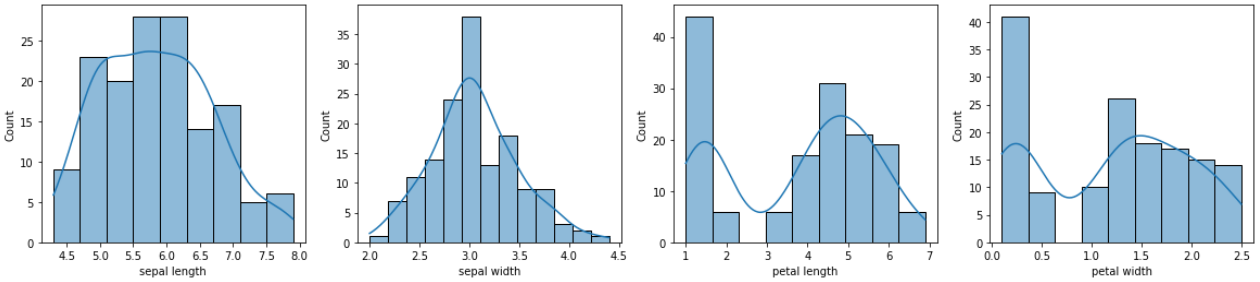
Out[5]:

	sepal length	sepal width	petal length	petal width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

2.2 단일 변수 분석

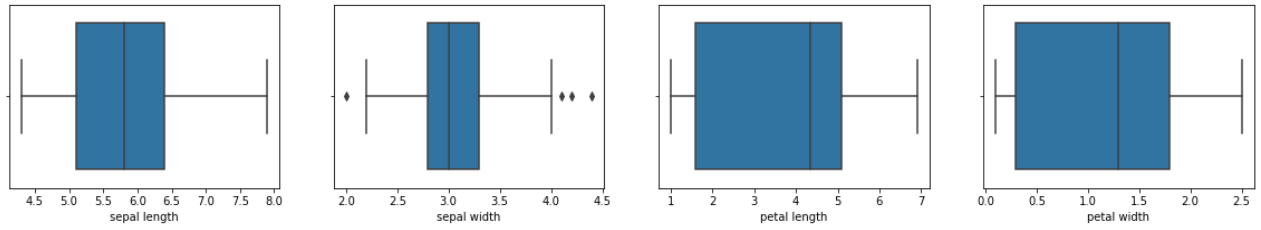
2.2.1 히스토그램

In [6]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=[20,4])
for i, column in enumerate(dataset.describe().columns):
    plt.subplot(1,4,i+1)
    sns.histplot(data=dataset, x=column, kde=True)
plt.show()
```



2.2.2 박스 플롯

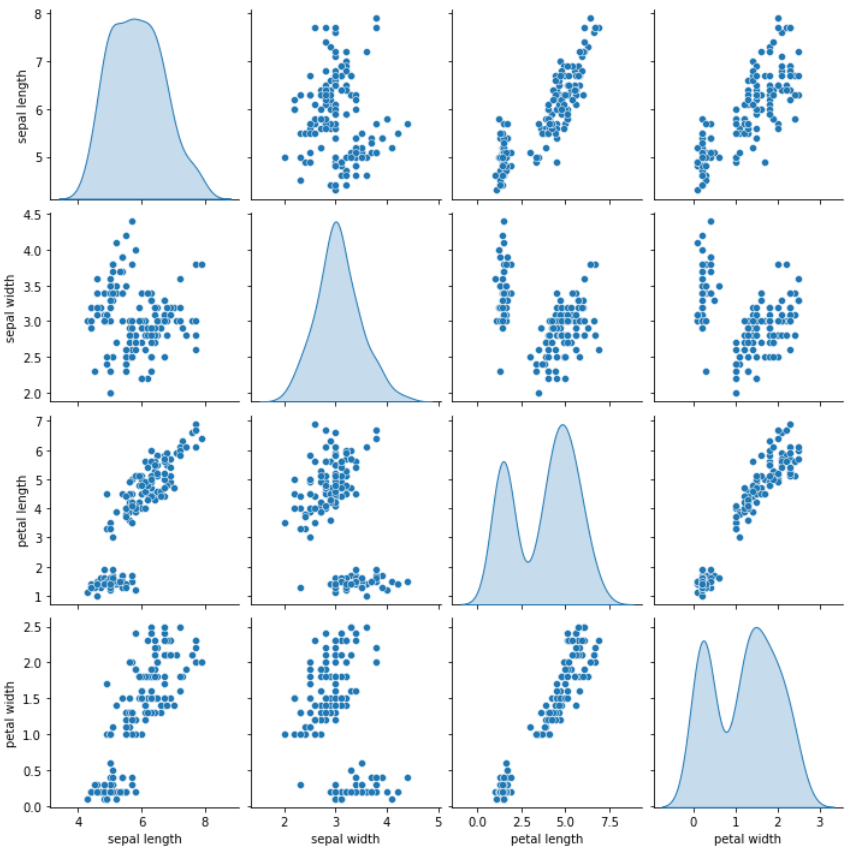
In [7]:
```
plt.figure(figsize=[20,3])
for i in enumerate(dataset.describe().columns[:4]):
    plt.subplot(1,4,i[0]+1)
    sns.boxplot(x=dataset[i[1]])

plt.show()
```



2.3 두 변수 관계 분석

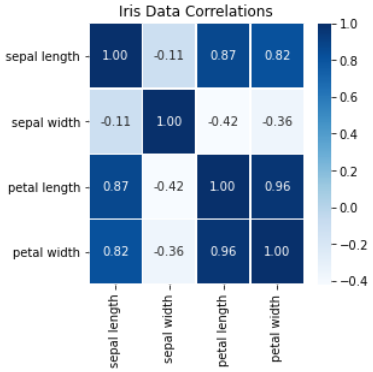
2.3.1 산포도 행렬

In [8]:
```
sns.pairplot(dataset, diag_kind="kde")
plt.show()
```



2.3.2 히트맵

In [9]:
```
fig, ax = plt.subplots(figsize=(4, 4))
sns.heatmap(dataset.corr(), linewidths=.5, annot=True, fmt=".2f", cmap='Blues')
plt.title('Iris Data Correlations')
plt.show()
```



3. 데이터 전처리

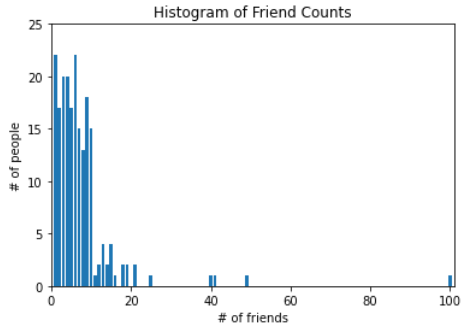
3.1 데이터 추출

In [10]:
```
clusterdata = dataset.iloc[:,:-1]
inputs = clusterdata.iloc[:,:].values.tolist()
columns = clusterdata.keys().tolist()
column2index ={ column : i for i, column in enumerate(columns)}
print('columns = ', columns)
print('column2index = ', column2index)
# print(inputs)
```

```
columns =  ['sepal length', 'sepal width', 'petal length', 'petal width']
column2index =  {'sepal length': 0, 'sepal width': 1, 'petal length': 2, 'petal width': 3}
```

3.2 데이터 표준화

In [11]:
```
from scratch.working_with_data import scale, rescale, Vector
from typing import List

inputs_normed = rescale(inputs)
# print(inputs_normed)
```



4. K-평균 군집화

Q1. 손실 곡선을 보고 K 선택하기

손실을 최소화 하는 클러스터 수 K를 찾아보시오. 단, K는 20까지 확인해 보라.

In [12]:
```
from scratch.linear_algebra import Vector
def num_differences(v1: Vector, v2: Vector) -> int:
    return len([x1 for x1, x2 in zip(v1, v2) if x1 != x2])
from typing import List
from scratch.linear_algebra import vector_mean
def cluster_means(k: int,
    inputs: List[Vector],
    assignments: List[int]) -> List[Vector]:
    # clusters[i] contains the inputs whose assignment is i
    clusters = [[] for i in range(k)]
    for input, assignment in zip(inputs, assignments):
        clusters[assignment].append(input)
    # if a cluster is empty, just use a random point
    return [vector_mean(cluster) if cluster else random.choice(inputs)
            for cluster in clusters]
import itertools
import random
import tqdm
from scratch.linear_algebra import squared_distance
class KMeans:
    def __init__(self, k: int) -> None:
        self.k = k # number of clusters
        self.means = None
    def classify(self, input: Vector) -> int:
        return min(range(self.k),
                   key=lambda i: squared_distance(input, self.means[i]))
# k=3 max(최장거리에 쓰일 함수)
    def classify_max(self, input: Vector) -> int:
        return max(range(self.k),
                   key=lambda i: squared_distance(input, self.means[i]))
    def train(self, inputs: List[Vector]) -> None:
        # Start with random assignments
        assignments = [random.randrange(self.k) for _ in inputs]
        with tqdm.tqdm(itertools.count()) as t:
            for _ in t:
                # Compute means and find new assignments
                self.means = cluster_means(self.k, inputs, assignments)
                new_assignments = [self.classify(input) for input in inputs]
                # Check how many assignments changed and if we're done
                num_changed = num_differences(assignments, new_assignments)
                if num_changed == 0:
                    return
                # Otherwise keep the new assignments, and compute new means
                assignments = new_assignments
                t.set_description(f"changed: {num_changed} / {len(inputs)}")
from matplotlib import pyplot as plt
def squared_clustering_errors(inputs: List[Vector], k: int) -> float:
    clusterer = KMeans(k)
    clusterer.train(inputs)
    means = clusterer.means
    assignments = [clusterer.classify(input) for input in inputs]
    return sum(squared_distance(input, means[cluster])
               for input, cluster in zip(inputs, assignments))
ks = range(1, 21)
errors = [squared_clustering_errors(inputs, k) for k in ks]
import random
optimal_k = errors.index(min(errors)) + 1
fig, ax = plt.subplots(figsize=(10, 5))
plt.plot(ks, errors)
plt.xticks(ks)
plt.xlabel(f"k (Optiaml k = {optimal_k})")
plt.ylabel("total squared error")
plt.title("Total Error vs. # of Clusters")
plt.show()
```

```
0it [00:00, ?it/s]
changed: 1 / 150: : 3it [00:00, 372.12it/s]
changed: 1 / 150: : 11it [00:00, 385.00it/s]
changed: 1 / 150: : 10it [00:00, 373.72it/s]
changed: 3 / 150: : 5it [00:00, 256.75it/s]
changed: 1 / 150: : 10it [00:00, 224.25it/s]
changed: 1 / 150: : 15it [00:00, 257.02it/s]
changed: 1 / 150: : 9it [00:00, 218.89it/s]
changed: 2 / 150: : 10it [00:00, 214.79it/s]
changed: 1 / 150: : 10it [00:00, 208.57it/s]
changed: 2 / 150: : 10it [00:00, 174.21it/s]
changed: 3 / 150: : 11it [00:00, 165.02it/s]
changed: 3 / 150: : 8it [00:00, 168.69it/s]
changed: 2 / 150: : 11it [00:00, 159.19it/s]
changed: 1 / 150: : 6it [00:00, 153.25it/s]
changed: 1 / 150: : 6it [00:00, 114.28it/s]
changed: 1 / 150: : 8it [00:00, 140.25it/s]
changed: 2 / 150: : 8it [00:00, 106.81it/s]
changed: 1 / 150: : 7it [00:00, 105.49it/s]
changed: 3 / 150: : 6it [00:00, 109.46it/s]
```

Q2. 군집화 및 결과 확인 (Q)

K=3으로 군집화를 해서 다음과 같이 군집화 결과를 확인해 보라.

In [13]:
```
random.seed(12) # so you get the same results as me
clusterer = KMeans(k=3)
clusterer.train(inputs)
means = sorted(clusterer.means)
print(len(means))
```

```
changed: 2 / 150: : 6it [00:00, 387.55it/s]

3
```

dataset에 k_means 컬럼 추가

In [14]:
```
assignments = [clusterer.classify(input) for input in inputs[::-1]]

# for i in reversed(assignments):
#     print(i)

dataset["k_means"] = assignments
dataset.head()
```

Out[14]:

	sepal length	sepal width	petal length	petal width	species	k_means
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1

species와 k_means 결과 비교

In [15]:
```
dataset[dataset['k_means']==0].head()
```

Out[15]:

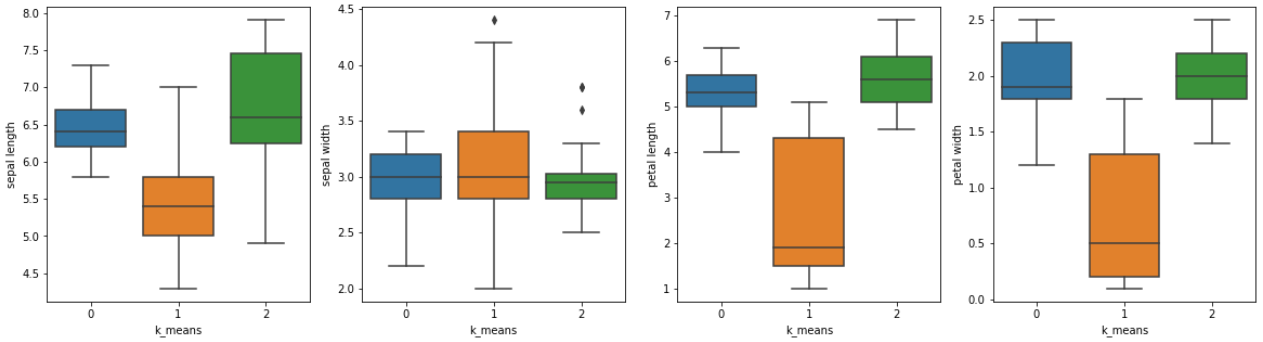
	sepal length	sepal width	petal length	petal width	species	k_means
51	6.4	3.2	4.5	1.5	Iris-versicolor	0
56	6.3	3.3	4.7	1.6	Iris-versicolor	0
92	5.8	2.6	4.0	1.2	Iris-versicolor	0
100	6.3	3.3	6.0	2.5	Iris-virginica	0
102	7.1	3.0	5.9	2.1	Iris-virginica	0

In [16]:
```
dataset.groupby(["k_means", "species"])['k_means'].count()
```

Out[16]:
```
k_means  species
0        Iris-versicolor     3
         Iris-virginica     26
1        Iris-setosa        50
         Iris-versicolor    47
2        Iris-virginica     24
Name: k_means, dtype: int64
```

군집화 결과 시각화

In [17]:
```
plt.subplots(figsize=(20, 5))
plt.subplot(1,4,1)
sns.boxplot(x='k_means', y='sepal length', data=dataset)
plt.subplot(1,4,2)
sns.boxplot(x='k_means', y='sepal width', data=dataset)
plt.subplot(1,4,3)
sns.boxplot(x='k_means', y='petal length', data=dataset)
plt.subplot(1,4,4)
sns.boxplot(x='k_means', y='petal width', data=dataset)
plt.show()
```

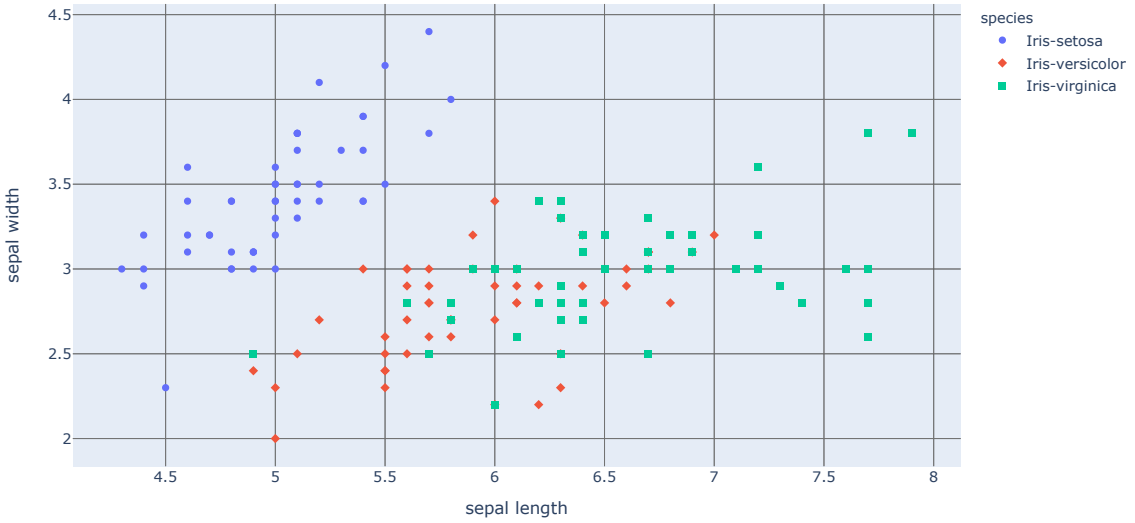


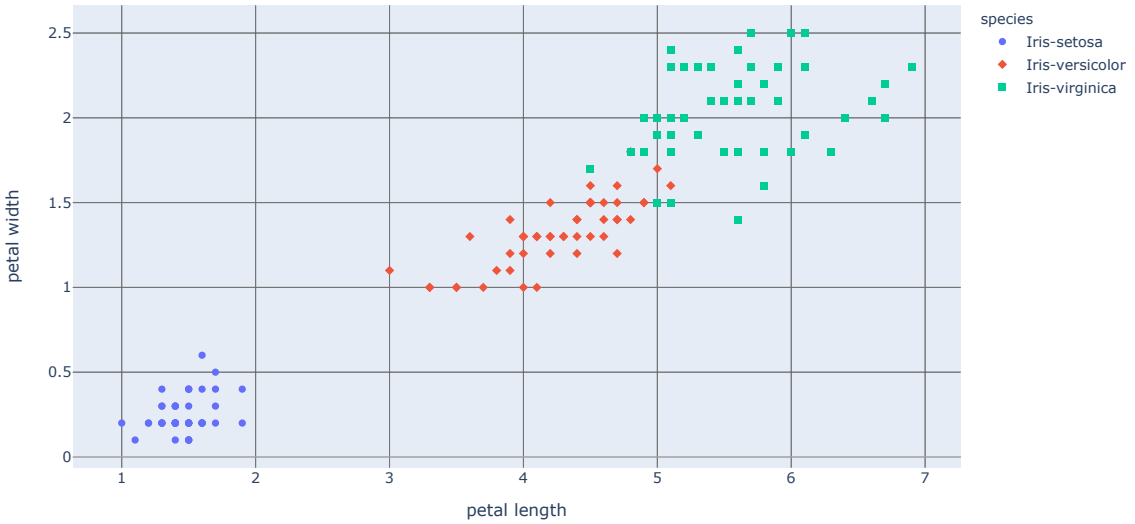
Q3. 군집화 및 결과 확인

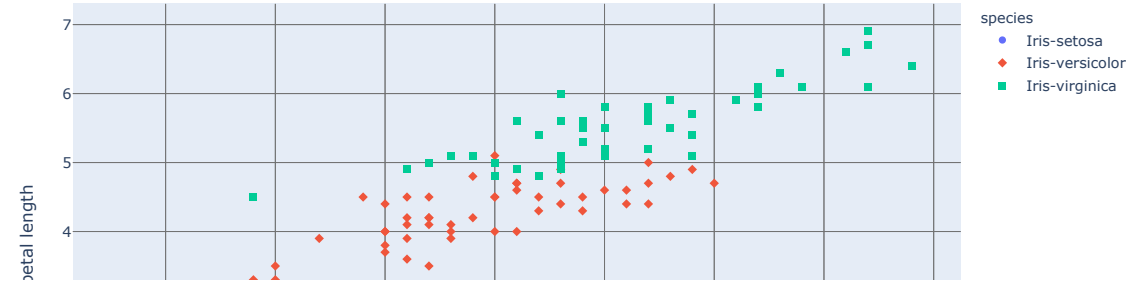
각 군집이 구분되도록 두 변수의 산포도를 그리는 함수 plot_cluster 구현하시오.

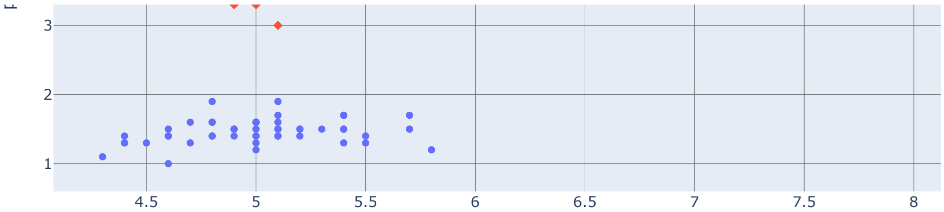
In [18]:

```
import numpy as np
import pandas as pd
import plotly.express as px
clusters = dataset
def plot_cluster(clusters, colindex1, colindex2):
    iris = clusters
    xs = colindex1
    ys = colindex2
    fig = px.scatter(iris,xs,ys, hover_data=['k_means'], color= 'species',symbol='species')
    fig.show()


plot_cluster(clusters, "sepal length","sepal width")

plot_cluster(clusters, "petal length", "petal width")

plot_cluster(clusters, "sepal length", "petal length")
```









5. 계층 군집화 (Hierarchical Clustering)

Q4. 군집화 및 결과 확인

K=3으로 최장 거리(max) 기준으로 군집화를 해서 다음과 같이 군집화 결과를 확인해 보라.

In [19]:
```
h_assignments = [clusterer.classify_max(input) for input in inputs]
# 위에 class kMeas에 def classify_max에 정의 했습니다.
print(h_assignments)
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2,
2, 2, 0, 2, 2, 0, 2, 0, 2, 0, 0, 2, 0, 2, 2, 0, 2, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 2, 0, 0, 0, 2, 2, 2, 0, 0, 2, 2, 0, 0, 0, 2, 0, 0, 0, 2, 2, 0, 2]
```

dataset에 h_clustering 컬럼 추가

In [20]:
```
dataset["h_clustering"] = h_assignments
dataset.head()
```

Out[20]:

	sepal length	sepal width	petal length	petal width	species	k_means	h_clustering
0	5.1	3.5	1.4	0.2	Iris-setosa	1	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1	1

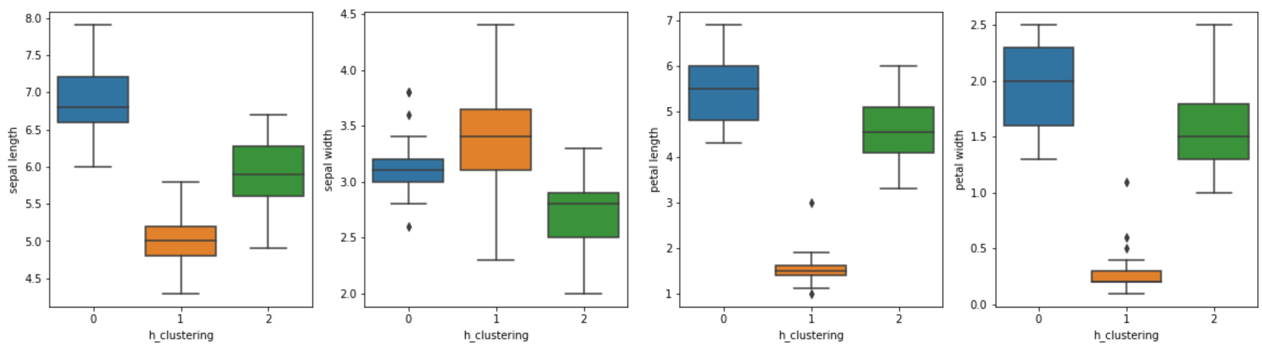
h_clustering과 species 비교

In [21]:
```
dataset.groupby(["h_clustering", "species"])['h_clustering'].count()
```

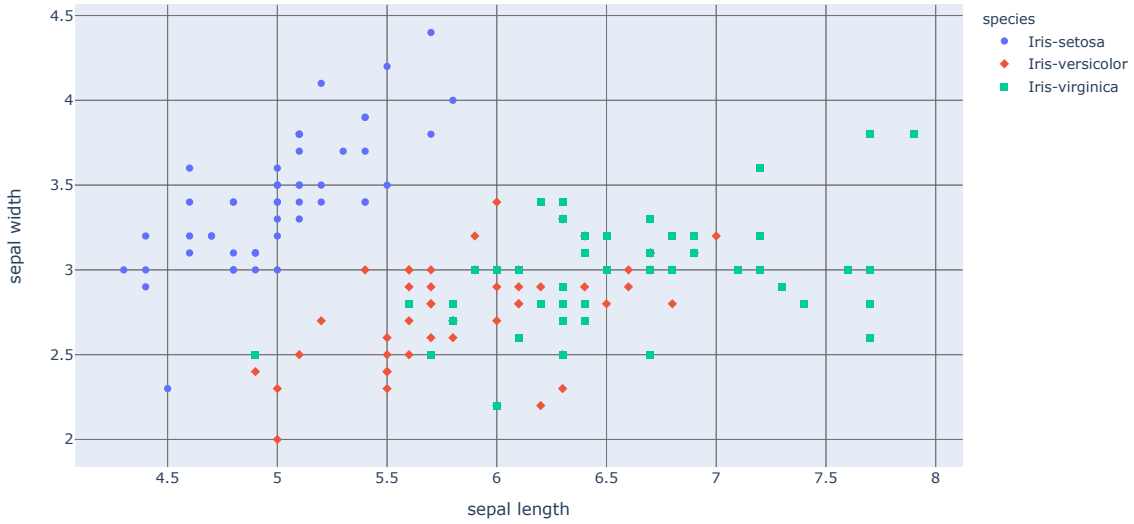
Out[21]:
```
h_clustering  species
0             Iris-versicolor    12
              Iris-virginica     25
1             Iris-setosa        50
              Iris-versicolor     1
2             Iris-versicolor    37
              Iris-virginica     25
Name: h_clustering, dtype: int64
```

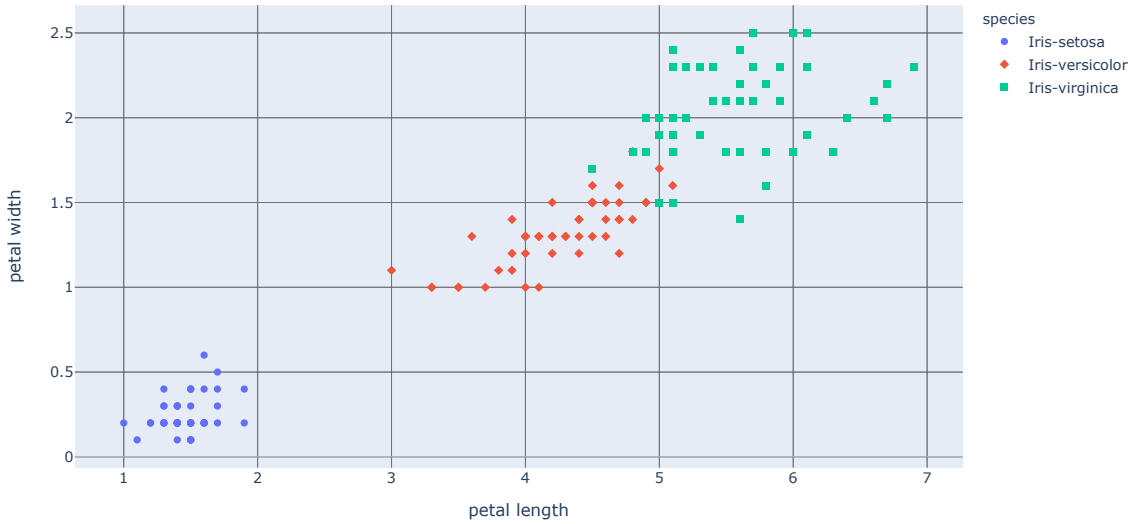
군집화 결과 시각화

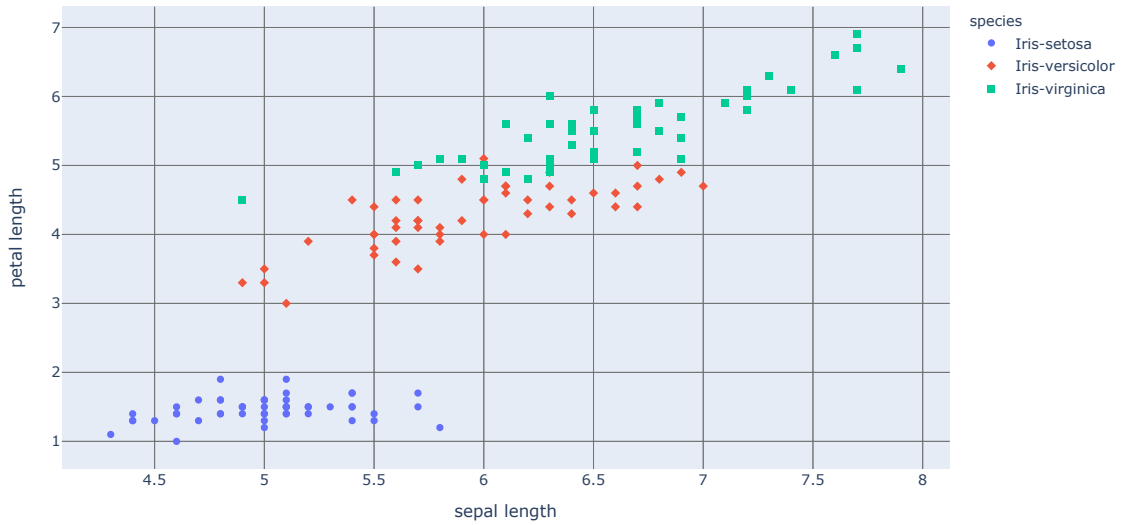
In [22]:
```
plt.subplots(figsize=(20, 5))
plt.subplot(1,4,1)
sns.boxplot(x = 'h_clustering', y = 'sepal length', data= dataset)
plt.subplot(1,4,2)
sns.boxplot(x = 'h_clustering', y = 'sepal width', data= dataset)
plt.subplot(1,4,3)
sns.boxplot(x = 'h_clustering', y = 'petal length', data= dataset)
plt.subplot(1,4,4)
sns.boxplot(x = 'h_clustering', y = 'petal width', data= dataset)
plt.show()
```



In [23]:
```
h_clusters = dataset

plot_cluster(h_clusters, "sepal length","sepal width")

plot_cluster(h_clusters, "petal length", "petal width")

plot_cluster(h_clusters, "sepal length", "petal length")
```







In [ ]: