

머신러닝

2차 과제

EM ALGORITHM

USING KMEANS FOR GMM

담당 교수님: 박철수

강의 시간: 월3, 수4

학 과: 컴퓨터정보공학부

학 번: 2020202055

성 명: 최소윤

1. 과제 목적

GMM(Gaussian Mixture Model)을 사용하여 EM알고리즘을 구현하는 과제이다. EM알고리즘에서는 Expectation과 Maximize 과정을 진행하는 데 주어진 수식에 알맞게 코드를 구현한다. E step에서는 posterior값인 gamma 값을 구하고 M step에서는 mean, sigma, pi 값을 각각 update한다. Matplotlib 라이브러리를 사용하여 Origin, EM, KMeans에 대한 각각의 결과값을 그래프로 나타내 비교할 수 있도록 한다.

2. 소스 코드

```
1  # -*- coding: utf-8 -*-
2
3  import copy
4  import numpy as np
5  import pandas as pd
6  import matplotlib.pyplot as plt
7  import seaborn as sns
8  from sklearn import datasets
9  from sklearn.cluster import KMeans
10 # Libraries added without justification are a minus factor.
11
12 # Seed setting
13 seed_num = 2022
14 # np.random.seed(seed_num)
15 iteration = 100 # Number of times to repeat steps E and M.
16
17
```

필요한 라이브러리들을 import하고 seed값을 고정하지 않으면 EM알고리즘 반복횟수인 iteration값을 100으로 초기화한다.

```
18  class EM:
19      """ expectation-maximization algorithm, EM algorithm
20      The EM class is a class that implements an EM algorithm using GMM and kmeans.
21
22      Within the fit function, the remaining functions should be used.
23      Other functions can be added, but all tasks must be implemented with Python built-in functions and Numpy functions.
24      You should annotate each function with a description of the function and parameters(Leave a comment).
25      """
26
27  def __init__(self, n_clusters, iteration):
28      """
29      Parameters
30      -----
31      n_clusters (int): Num of clusters (num of GMM)
32      iteration (int): Num of iteration
33          Termination conditions if the model does not converge
34      mean (ndarray): Num of clusters x Num of features
35          The mean vector that each cluster has.
36      sigma (ndarray): Num of clusters x Num of features x Num of features
37          The covariance matrix that each cluster has.
38
39      """
40      self.n_clusters = n_clusters
41      self.iteration = iteration
42      self.mean = np.zeros((n_clusters, 4))
43      self.sigma = np.zeros((n_clusters, 4, 4))
44      self.pi = np.zeros((n_clusters))
```

클래스 EM을 정의한다. __init__ 함수에는 각 변수들을 초기화하고 배열 크기 설정 작업을 진행한다.

```

46     def initialization(self, data):
47         """ 1.initialization, 10 points
48         Initial values for mean, sigma, and pi should be assigned.
49         It have a significant impact on performance.
50
51         your comment here
52
53         Initialize means, sigma, pi
54         Use np.random.choice to extract sample indexes and set a random mean value
55         Sets the initial value of the sigma as a identity matrix.
56         """
57         # your code here
58
59         # Initialize means to random data points
60         indices = np.random.choice(data.shape[0], size=self.n_clusters, replace=False)
61         self.mean = data[indices]
62
63         # Initialize covariances to identity matrices
64         self.sigma = np.stack([np.eye(data.shape[1]) for i in range(self.n_clusters)])
65
66         # Initialize mixing coefficients to be uniform
67         self.pi = np.ones(self.n_clusters) / self.n_clusters
68
69         # return nothing (return something or nothing)

```

initialization 함수로 mean, sigma, pi값을 초기화를 하는 함수이며 np.random.choice 메소드를 사용하여 데이터에서 임의의 인덱스 값을 추출하여 mean을 초기화한다. 이는 무작위로 데이터에서 골라 저장하는 것이다. Sigma는 단위 행렬로 초기화를 하고 이는 covariance값을 초기화하는 것과 같다. pi값은 1을 cluster의 개수로 나눈 값으로 초기화를 한다.

```

71     def multivariate_gaussian_distribution(self, data, mean, cov):
72         """ 2.multivariate gaussian distribution, 10 points
73         Use the linear algebraic functions of Numpy. n of this function is not self.pi
74
75         your comment here
76
77         Implement the multivariate_gaussian_distribution formula
78         Use np.linalg.det function to calculate determinants and use np.linalg.inv function to calculate inverse matrix
79
80         """
81         # your code here
82         n_features = data.shape[1] # 4
83
84         # Compute the determinant and inverse of the covariance matrix.
85         det_cov = np.linalg.det(cov)
86         inv_cov = np.linalg.inv(cov)
87
88         # Compute the probability density values using the multivariate Gaussian formula.
89         output = np.zeros(data.shape[0])
90         for i in range(data.shape[0]):
91             # Compute the difference between the data point and the mean of the cluster to which it belongs.
92             x_minus_mean = data[i] - mean
93             # Compute the exponent of the multivariate Gaussian distribution formula using the inverse of the covariance matrix.
94             exponent = -0.5 * x_minus_mean.T @ inv_cov @ x_minus_mean
95             # Compute the normalization constant of the multivariate Gaussian distribution formula.
96             norm_const = np.sqrt((2*np.pi)**n_features * det_cov)
97             # Compute the probability density value for the current data point and store it in the output array.
98             output[i] = np.exp(exponent) / norm_const
99
100        return output # something or nothing
101

```

multivariate_gaussian_distribution 함수로 다변량 정규분포를 계산하는 함수다.

$$g_{(\mu, \Sigma)}(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\Sigma)}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)}$$

위 식은 다변량 정규분포 수식이며 이를 코드화 한 것이다. `np.linalg.det`함수는 행렬식을 계산하며, `np.linalg.inv`는 역행렬을 계산한다. 전치 행렬을 나타날 때는 `x_minus_mean.T`로 표시하며 내적연산을 할 시에는 연산자 `@`를 사용하여 표현한다.

```

102     def expectation(self, data):
103         """ 3.expectation step, 20 points
104             The multivariate_gaussian_distribution(MVN) function must be used.
105
106             your comment here
107
108             gamma is posterior
109             Using multivariate_gaussian_distribution, implement the formula posterior.
110             """
111
112         # your code here
113
114         # Initialize gamma value
115         gamma = np.zeros((data.shape[0], self.n_clusters))
116
117         # Implement the formula posterior
118         for k in range(self.n_clusters):
119             gamma[:,k] = self.pi[k] * self.multivariate_gaussian_distribution(data, self.mean[k], self.sigma[k])
120
121         gamma = gamma / np.sum(gamma, axis = 1, keepdims=True)
122
123     return gamma # something or nothing

```

`expectation` 함수에서는 `gamma` 변수를 업데이트하는 부분이며 이는 `posterior`를 의미한다.

$$\gamma(z_{nk}) = \frac{\pi_k N(x_n | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j N(x_n | \mu_j, \Sigma_j)}$$

위 수식을 코드화 한 것이며 $N(x_n | \mu_k, \Sigma_k)$ 이 부분이 `multivariate_gaussian` distribution이며 함수를 호출하여 구현하였다.

```

125 def maximization(self, data, gamma):
126     """ 4 maximization step, 20 points
127     Hint: np.outer
128
129     your comment here
130
131     update pi, mean, sigma values
132     When we calculate mean, reshape(-1, 1) is intended to dimension the data multiplied by gamma[:,k].
133     """
134     # your code here
135
136     # update pi
137     self.pi = gamma.sum(axis=0) / len(data)
138
139     # Update mean
140     for k in range(self.n_clusters):
141         self.mean[k] = np.sum(gamma[:,k].reshape(-1, 1) * data, axis = 0) / gamma[:,k].sum()
142
143     n_features = data.shape[1]
144     # Update sigma
145     for k in range(self.n_clusters):
146         sigma_sum = np.zeros((n_features, n_features))
147         # iterate over each data point
148         for i in range(data.shape[0]):
149             # calculate the difference between the data point and the cluster mean
150             data_diff = data[i] - self.mean[k]
151             # calculate the outer product of the data difference with its transpose and add it to the sum
152             sigma_sum += gamma[i, k] * np.outer(data_diff.T, data_diff)
153         # divide the sum by the sum of the corresponding gamma values to get the new covariance matrix for the cluster
154         self.sigma[k] = sigma_sum / np.sum(gamma[:, k])
155
156     # return nothing (return something or nothing)

```

Maximization 함수에서는 E step에서 구한 posterior 값을 사용하여 pi, mean, sigma 값을 업데이트한다.

$$\mu_k = \frac{\sum_{n=1}^N \gamma(z_{nk}) x_n}{\sum_{n=1}^N \gamma(z_{nk})}$$

$$\Sigma_k = \frac{\sum_{n=1}^N \gamma(z_{nk})(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^N \gamma(z_{nk})}$$

$$\pi_k = \frac{1}{N} \sum_{n=1}^N \gamma(z_{nk})$$

pi, mean, sigma는 모두 posterior인 gamma 변수를 사용한다. Mean을 계산할 때 gamma값을 reshape를 해주는 이유는 1차원 배열이었던 gamma값을 2차원 배열로 변환시켜 2차원 배열인 data와 곱연산을 하기 위해서다. sigma는 data의 각 행만큼 반복문을 돌려 mean값을 뺀 행렬과 그 행렬의 전치 행렬을 외적하고 gamma값과 곱한다. 내적이 아닌 외적을 사용한 이유는 외적은 두 벡터 간의 행렬을 반환하므로 공분산 행렬을 직접 계산할 수 있기 때문이다.

```

158     def fit(self, data):
159         """ Fit clustering, 20 points
160         Functions initialization, expectation, and maximization should be used by default.
161         Termination condition. Iteration is finished or posterior is the same as before. (Beware of shallow copy)
162         Prediction for return should be formatted. Refer to iris['target'] format.
163
164         your comment here
165
166         This method fits a Gaussian Mixture Model to the input data using the Expectation-Maximization (EM) algorithm.
167         The algorithm runs for a specified number of iterations or until the posterior probabilities no longer change significantly.
168         The method returns predicted cluster labels for the input data based on the final model fit.
169         """
170
171         # your code here
172         # initialization
173         self.initialization(data)
174
175         # initialize likelihood
176         prev_likelihood = -np.inf
177
178         for i in range(self.iteration):
179             # E step
180             gamma = self.expectation(data)
181
182             # M step
183             self.maximization(data, gamma)
184
185             # compute log likelihood
186             likelihood = np.log(np.sum([self.pi[k] * self.multivariate_gaussian_distribution(data, self.mean[k], self.sigma[k]) for k in range(self.n_clusters)]))
187
188             # check convergence
189             if np.abs(likelihood - prev_likelihood) < 1e-6:
190                 break
191
192             prev_likelihood = likelihood
193
194         # return predicted labels
195         labels = np.argmax(gamma, axis=1)
196
197         return labels

```

위 함수는 fit 함수이며 Gaussian Mixture Model을 사용하여 EM 알고리즘을 통해 최적의 파라미터 값을 구하도록 모델을 fitting하는 함수이다. 처음에 initialization함수를 통해 각 파라미터 값을 초기화하고 likelihood값을 음의 무한대로 초기화한다. 설정한 iteration만큼 반복하여 EM 알고리즘을 수행한다. E step에서 expectation 함수를 사용하여 gamma 값을 구하고 M step에서 maximize 함수를 사용하여 pi, mean, sigma 값을 업데이트한다. likelihood값은 pi값에 multivariate gaussian distribution 함수의 output값과 곱한 후 log를 씌운 값으로 저장한다. Iteration 만큼 반복하다가 구한 likelihood값과 이전 likelihood값의 차이가 1e-6보다 작다면 반복문을 멈춘다. 반복문을 끝내고 labels의 값을 gamma값의 argmax함수를 통해 나온 값으로 저장한다.

```

197 ~ def plotting(data, title):
198 ~     """ 6.plotting, 20 points with report
199 ~     Default = seaborn pairplot
200 ~
201 ~     your comment here
202 ~
203 ~     Generate Graphs
204 ~     """
205 ~
206 ~     # your code here
207 ~
208 ~     # Specify a fixed color for each cluster.
209 ~     if(title=="Origin"):
210 ~         palette = {'setosa': 'blue', 'versicolor': 'orange', 'virginica': 'green'}
211 ~     else:
212 ~         palette = {0: "blue", 1: "orange", 2: "green"}
213 ~
214 ~     # Creates a figure with a 4x4 grid of subplots and assigns it to the variables fig and axes. It also sets the figure title to the title argument.
215 ~     fig, axes = plt.subplots(4, 4, figsize=(10, 10))
216 ~     fig.suptitle(title, fontsize = 16)
217 ~
218 ~     # This code loops through each pair of columns in the data argument (excluding the last column)
219 ~     # Generates either a KDE plot or a scatter plot in each corresponding subplot, depending on whether the columns are the same or different.
220 ~     for i, col1 in enumerate(data.columns[:-1]):
221 ~         for j, col2 in enumerate(data.columns[:-1]):
222 ~             ax = axes[i][j]
223 ~             if i == j:
224 ~                 sns.kdeplot(data=data, x=col2, hue="labels", fill=True, ax=ax, palette=palette)
225 ~             else:
226 ~                 sns.scatterplot(data=data, x=col1, y=col2, hue="labels", ax=ax, palette=palette)
227 ~
228 ~     plt.tight_layout()
229 ~
229 ~     # return nothing (return something or nothing)

```

plotting 함수는 데이터를 시각화 해주는 함수로 Origin, EM, KMeans에 대해 각 feature에 따라서 그레프를 출력해 준다. 이 함수는 행과 열이 같으면 kdeplot 함수를 통해 커널밀도추정 함수를 그리고, 그렇지 않다면 scatterplot 함수를 사용하여 산점도를 생성한다.

```

234 ~ if __name__ == '__main__':
235 ~     # Loading and labeling data
236 ~     iris = datasets.load_iris()
237 ~     original_data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] + ['labels'])
238 ~     original_data['labels'] = original_data['labels'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
239 ~     plotting(original_data, "origin")
240 ~
241 ~     # Only data is used w/o labels because EM and Kmeans are unsupervised learning
242 ~     data=iris['data']
243 ~
244 ~     # unsupervised learning(clustering) using EM algorithm
245 ~     EM_model = EM(n_clusters=3, iteration=iteration)
246 ~     EM_pred = EM_model.fit(data)
247 ~     EM_pd=pd.DataFrame(data=np.c_[data, EM_pred], columns=iris['feature_names'] + ['labels'])
248 ~     plotting(EM_pd, "EM")
249 ~
250 ~     # Why are these two elements almost the same? Write down the reason in your report. Additional 10 points
251 ~     print(f'pi : {EM_model.pi}')
252 ~     print(f'count / total : {np.bincount(EM_pred) / 150}')
253 ~
254 ~     # unsupervised learning(clustering) using KMeans algorithm
255 ~     KM_model = KMeans(n_clusters=3, init='random', random_state=seed_num, max_iter=iteration).fit(data)
256 ~     KM_pred = KM_model.predict(data)
257 ~     KM_pd = pd.DataFrame(data=np.c_[data, KM_pred], columns=iris['feature_names'] + ['labels'])
258 ~     plotting(KM_pd, "KMeans")
259 ~

```

```

260     # No need to explain.
261     for idx in range(2):
262         EM_point = np.argmax(np.bincount(EM_pred[idx * 50:(idx + 1) * 50]))
263         KM_point = np.argmax(np.bincount(KM_pred[idx * 50:(idx + 1) * 50]))
264         EM_pred = np.where(EM_pred == idx, 3, EM_pred)
265         EM_pred = np.where(EM_pred == EM_point, idx, EM_pred)
266         EM_pred = np.where(EM_pred == 3, EM_point, EM_pred)
267         KM_pred = np.where(KM_pred == idx, 3, KM_pred)
268         KM_pred = np.where(KM_pred == KM_point, idx, KM_pred)
269         KM_pred = np.where(KM_pred == 3, KM_point, KM_pred)
270
271     EM_hit = np.sum(iris['target'] == EM_pred)
272     KM_hit = np.sum(iris['target'] == KM_pred)
273     print(f'EM Accuracy: {round(EM_hit / 150, 2)} Hit: {EM_hit} / 150')
274     print(f'KM Accuracy: {round(KM_hit / 150, 2)} Hit: {KM_hit} / 150')
275

```

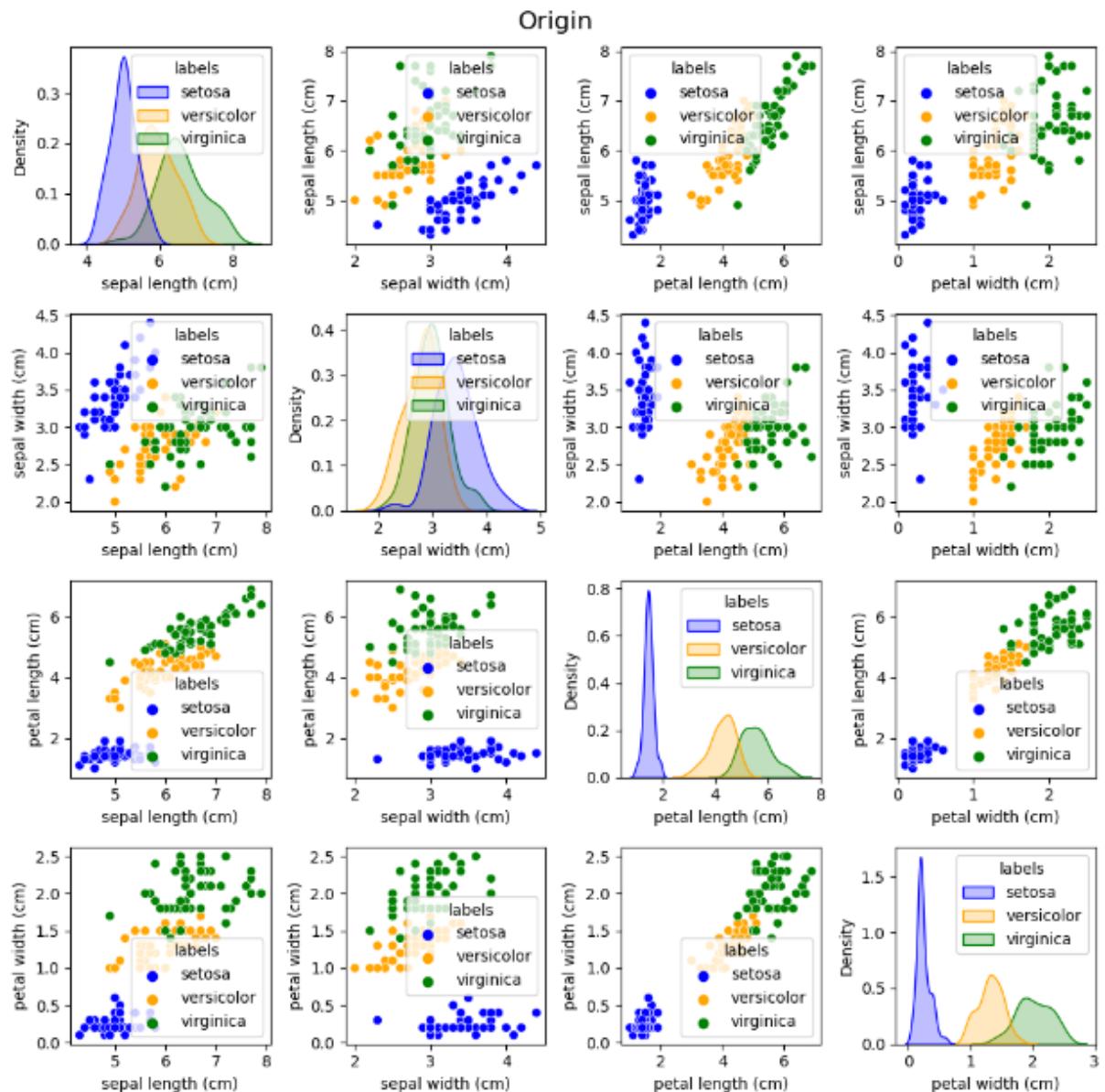
main 함수에서 iris data를 load하고 original_data에 대한 그래프를 출력한다. 그런 후에 구현한 EM 알고리즘을 통해 EM_model 구축하고 model을 fit한 후에 그래프를 출력한다.

KMeans 알고리즘을 통해 KMeans를 구축하고 모델을 설정하고 predict를 하여 그래프를 출력한다. 그 아래 코드 부분은 EM 알고리즘과 KMeans 알고리즘의 정확도를 계산하는 부분이다.

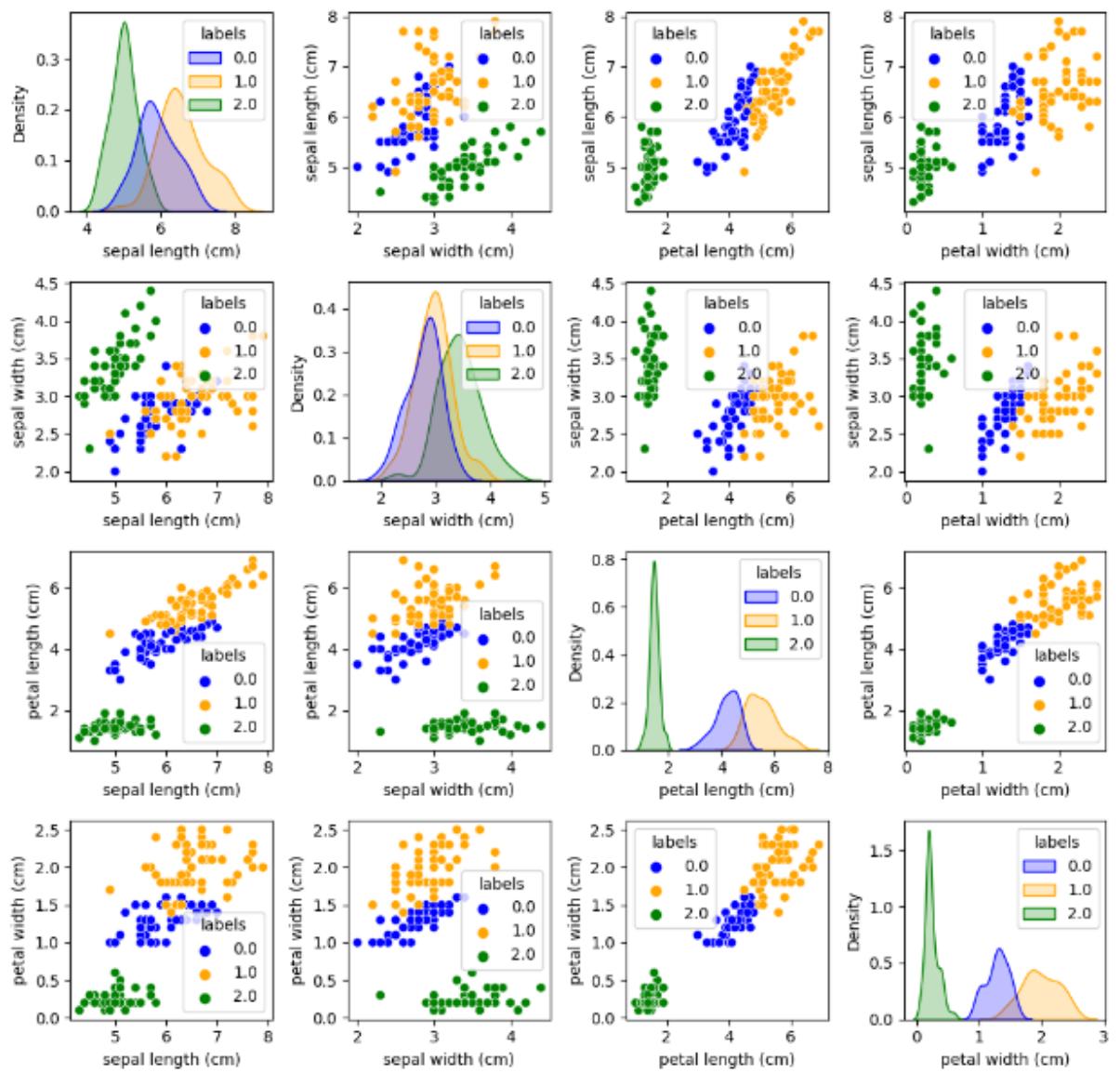
3. 실행 결과

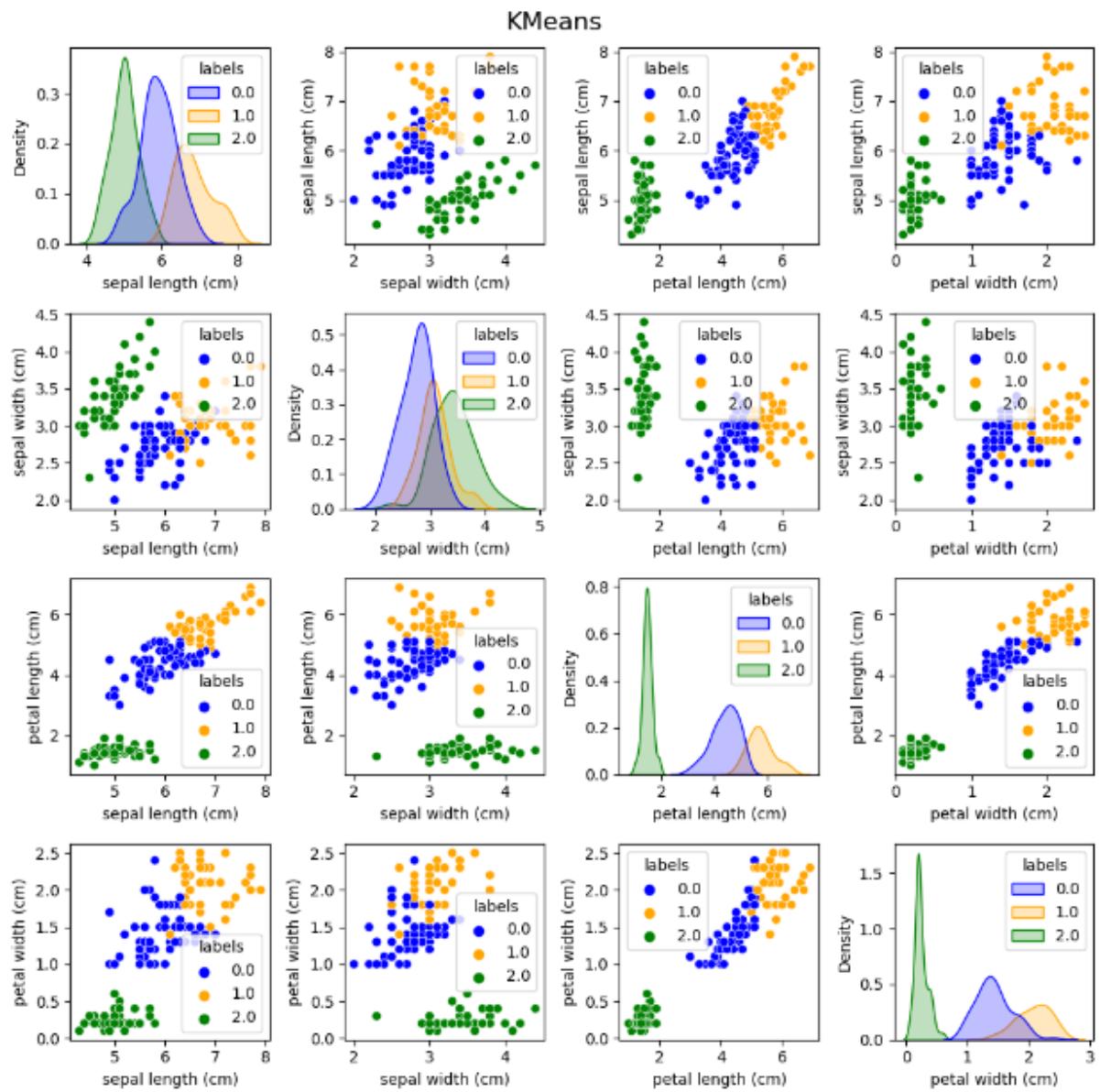
1 times

```
pi : [0.29919547 0.3674712  0.33333333]
count / total : [0.3       0.36666667 0.33333333]
/usr/local/lib/python3.10/dist-packages/sklearn/clus
      warnings.warn(
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```



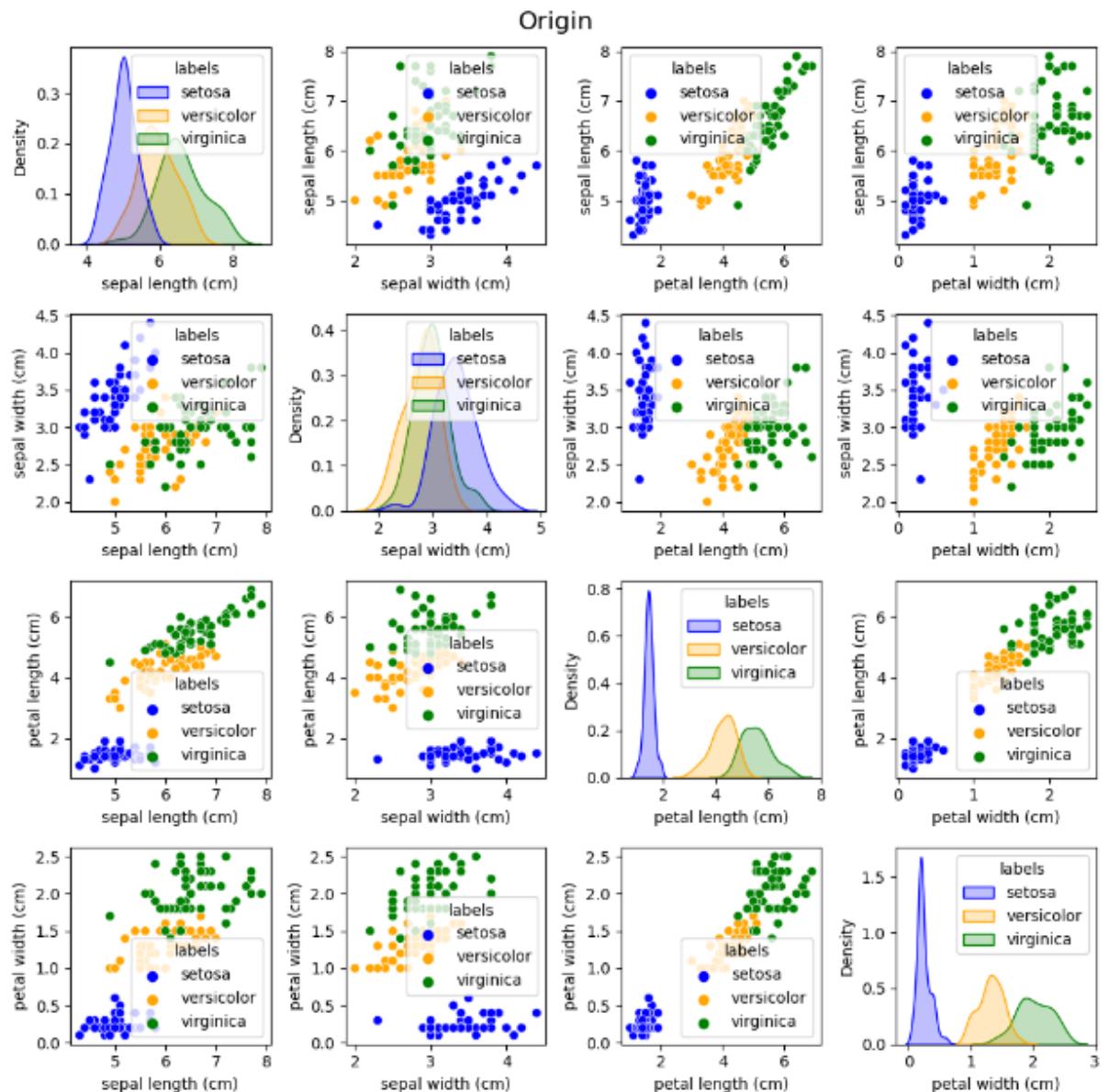
EM



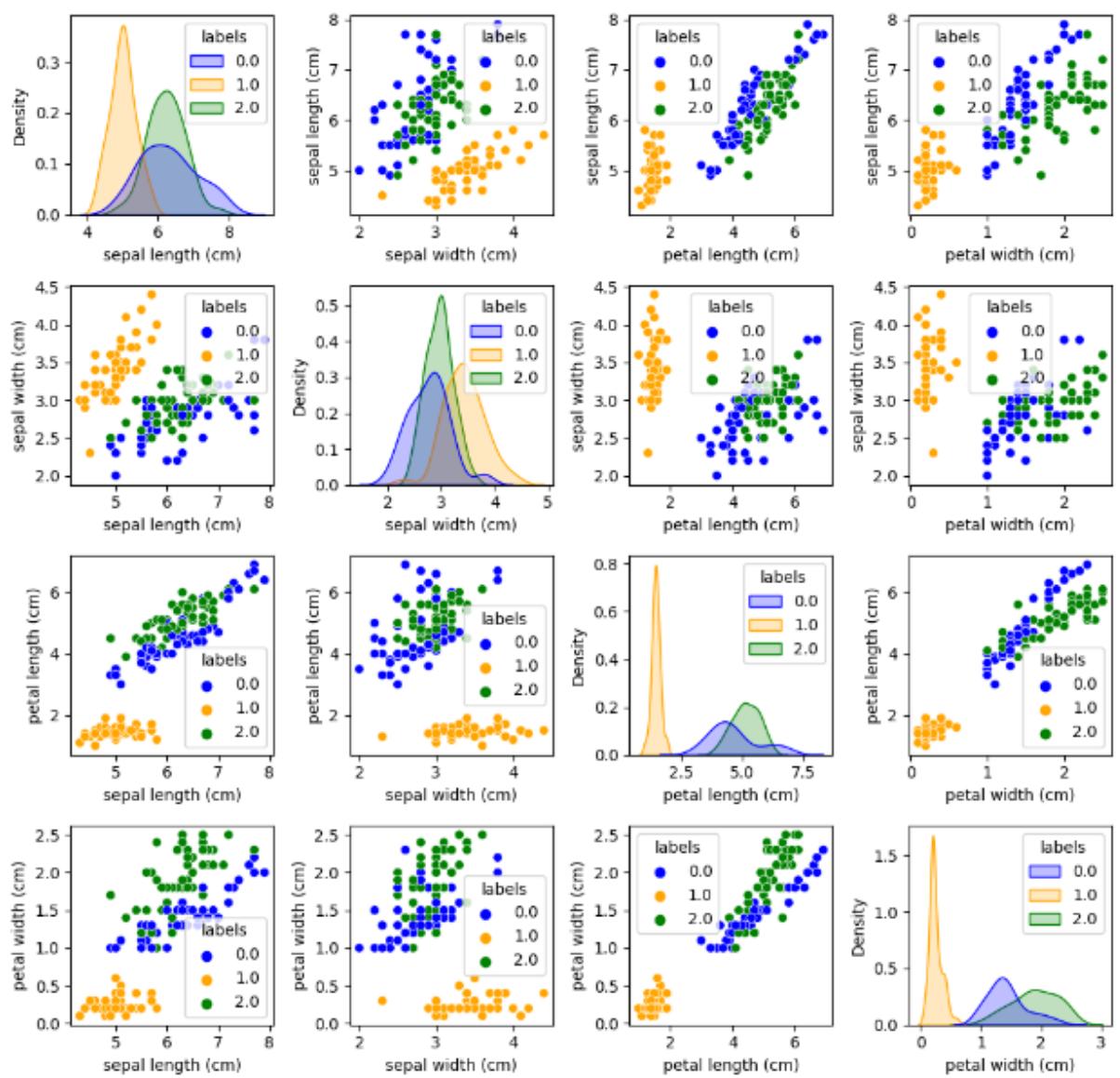


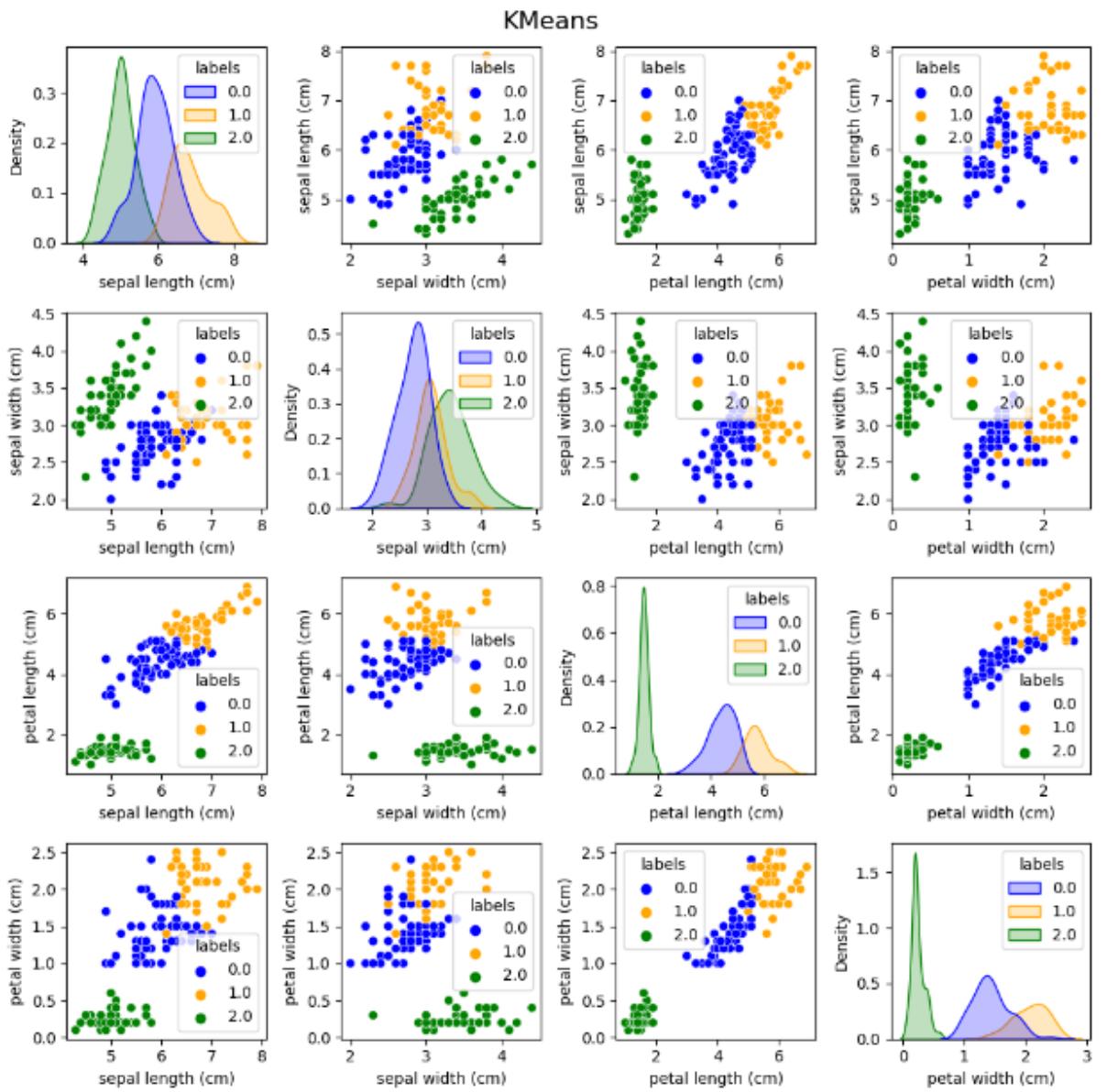
2 times

```
pi : [0.61260258 0.66619604 0.6646044 ]  
count / total : [0.61333333 0.66666666 0.66666666]  
/usr/local/lib/python3.10/dist-packages/sklearn/clu  
    warnings.warn()  
EM Accuracy: 0.82 Hit: 126 / 150  
KM Accuracy: 0.89 Hit: 134 / 150
```



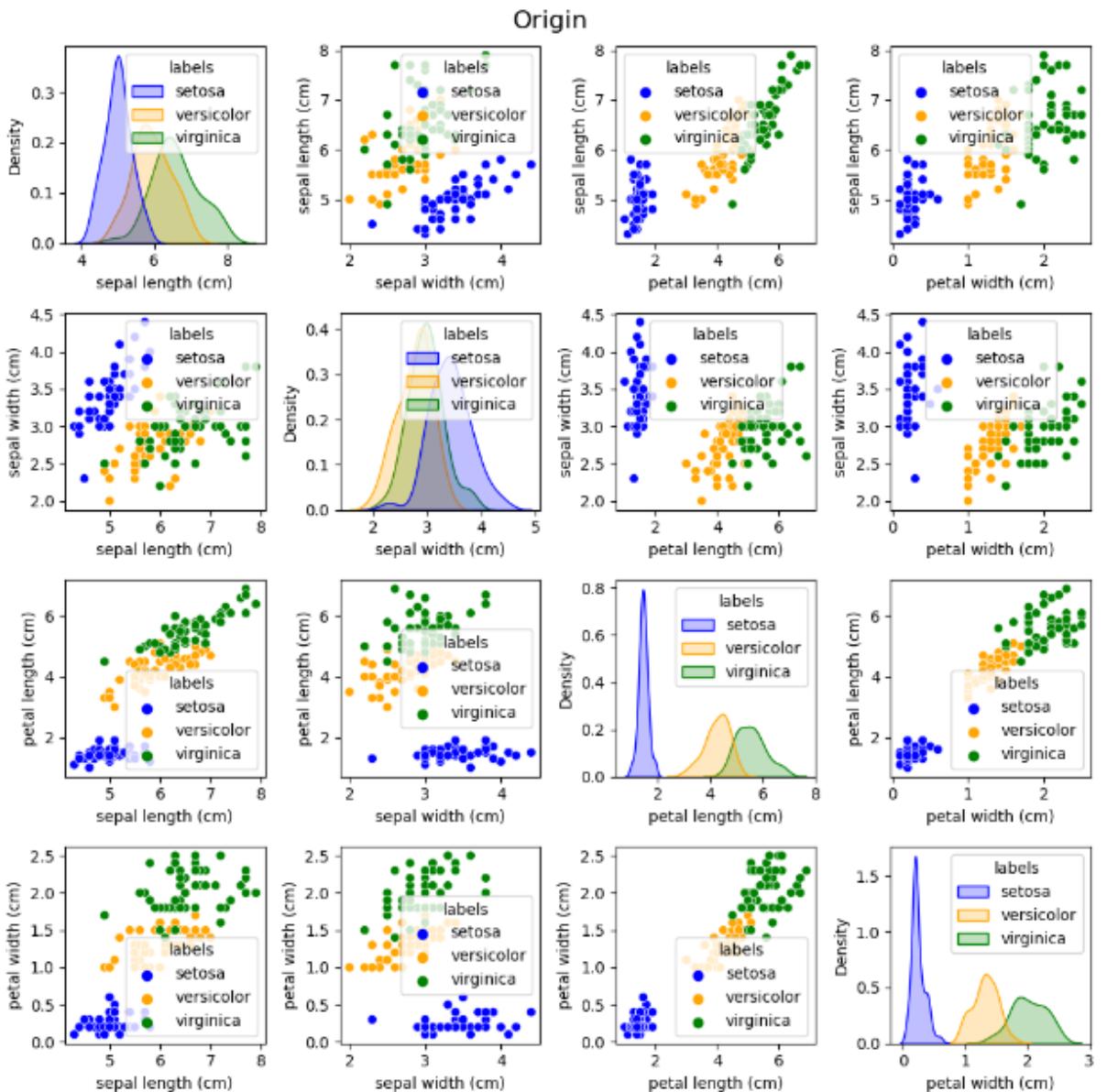
EM



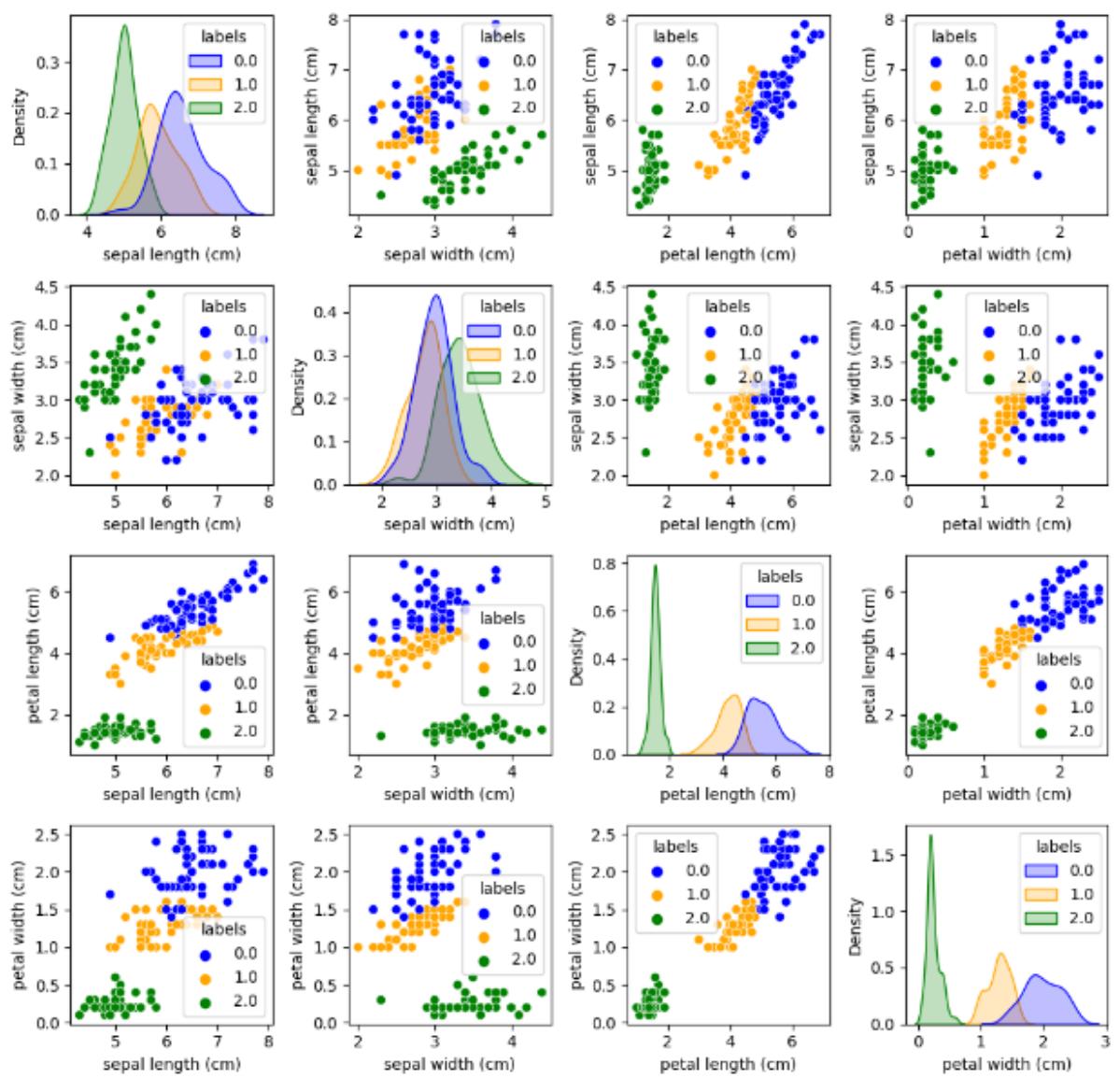


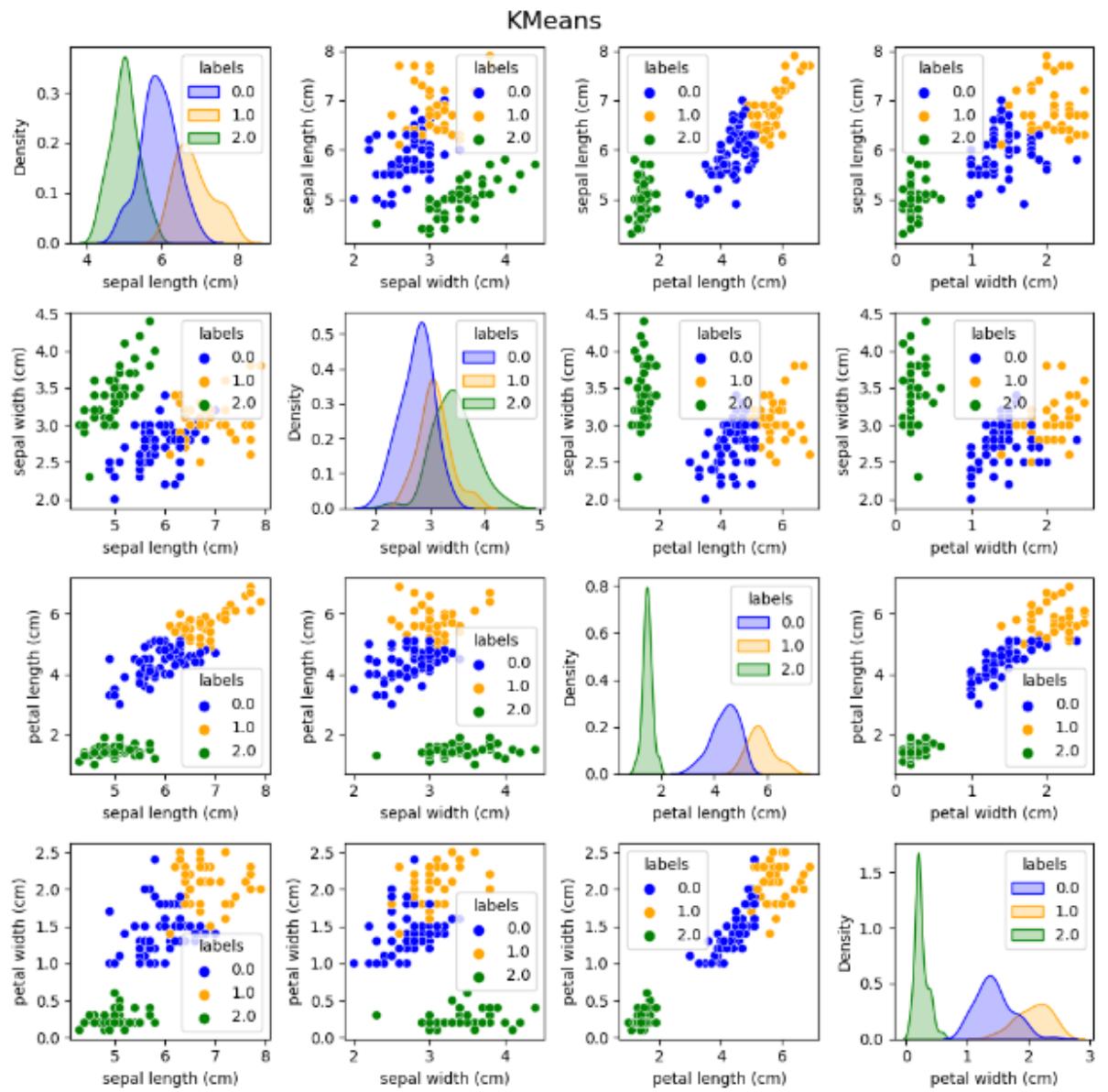
3 times

```
pi : [0.66747077 0.2991969 0.88888888]
count / total : [0.66666667 0.6 0.88888888]
/usr/local/lib/python3.10/dist-packages/sklearn/clf
warnings.warn(
    EM Accuracy: 0.87 Hit: 145 / 160
    KM Accuracy: 0.89 Hit: 184 / 160
```



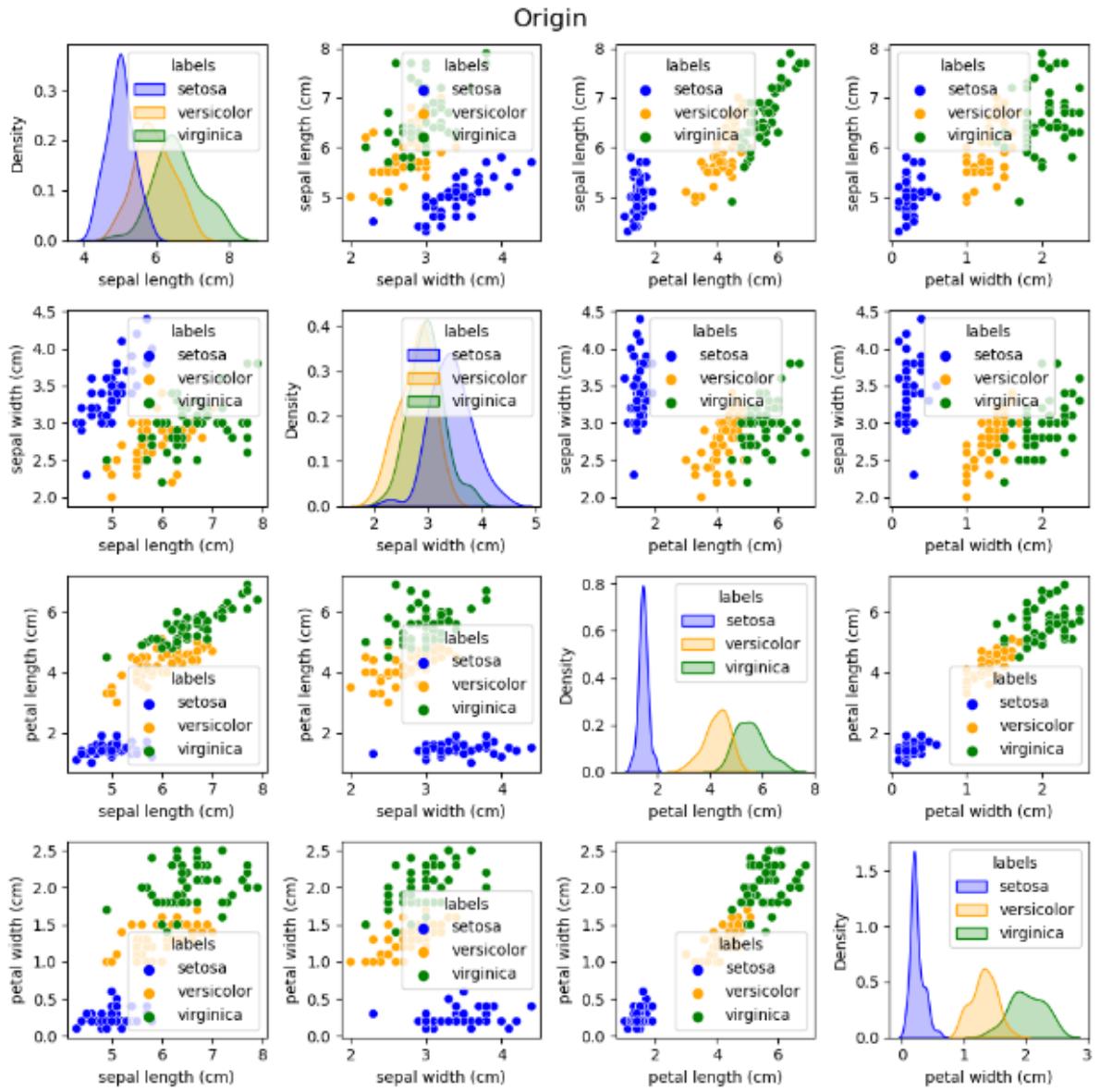
EM



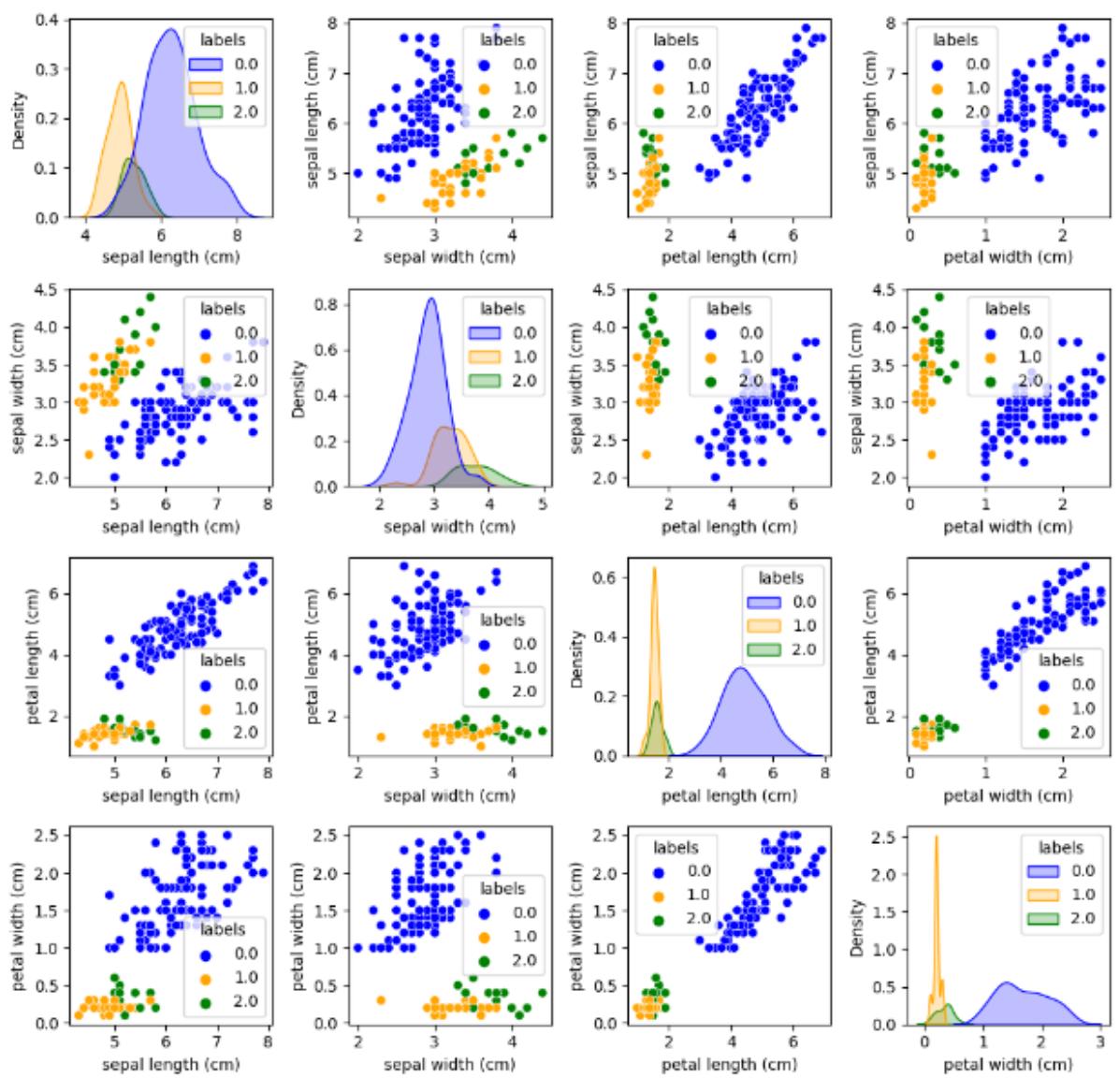


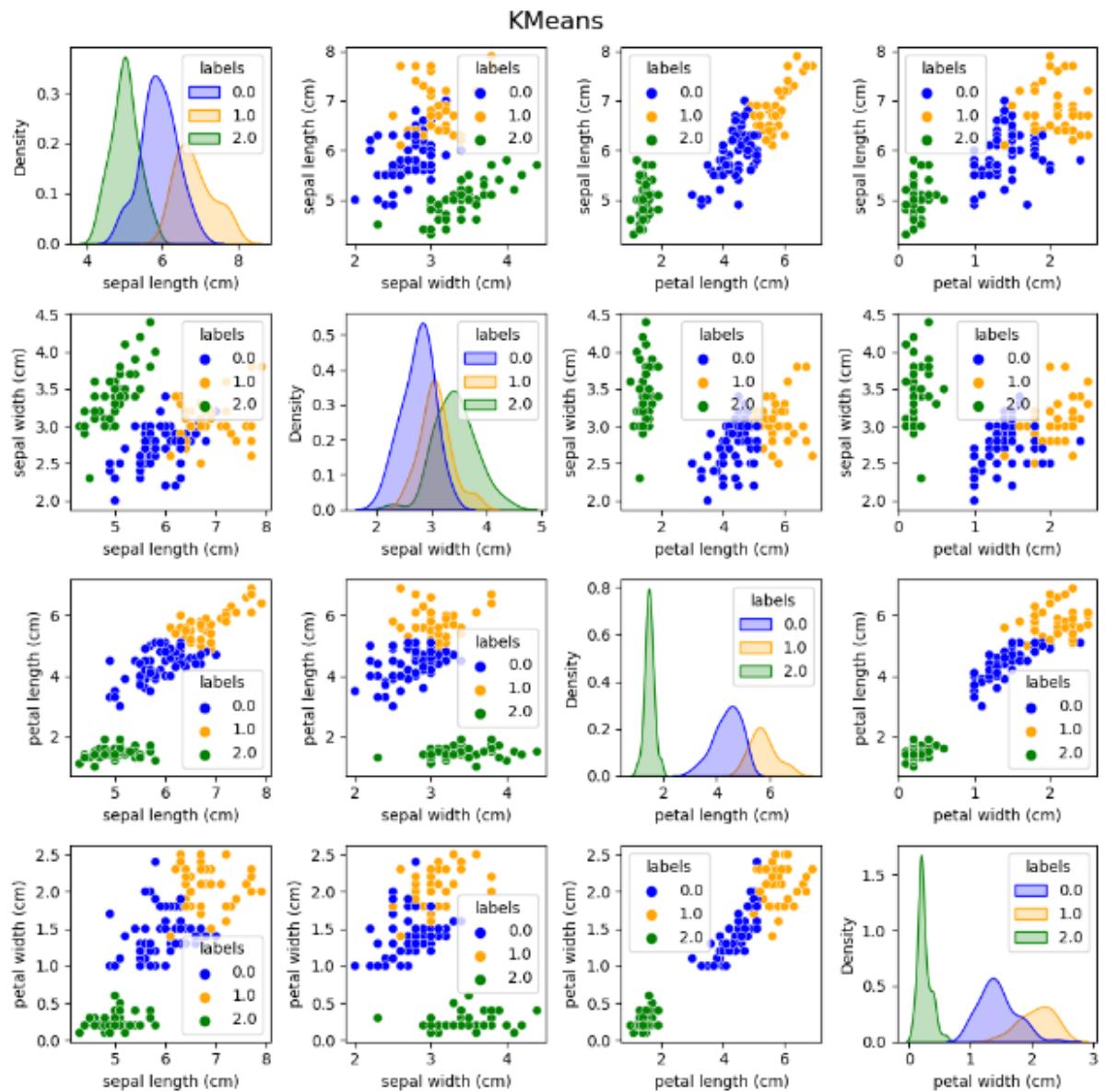
4 times

```
pi : [0.68887817 0.22700548 0.10882188]
count / total : [0.68888887 0.23883333 0.1]
/usr/local/lib/python3.10/dist-packages/sklearn/clu
warnings.warn(
EM Accuracy: 0.67    Hit: 85 / 120
KME Accuracy: 0.89    Hit: 104 / 120
```



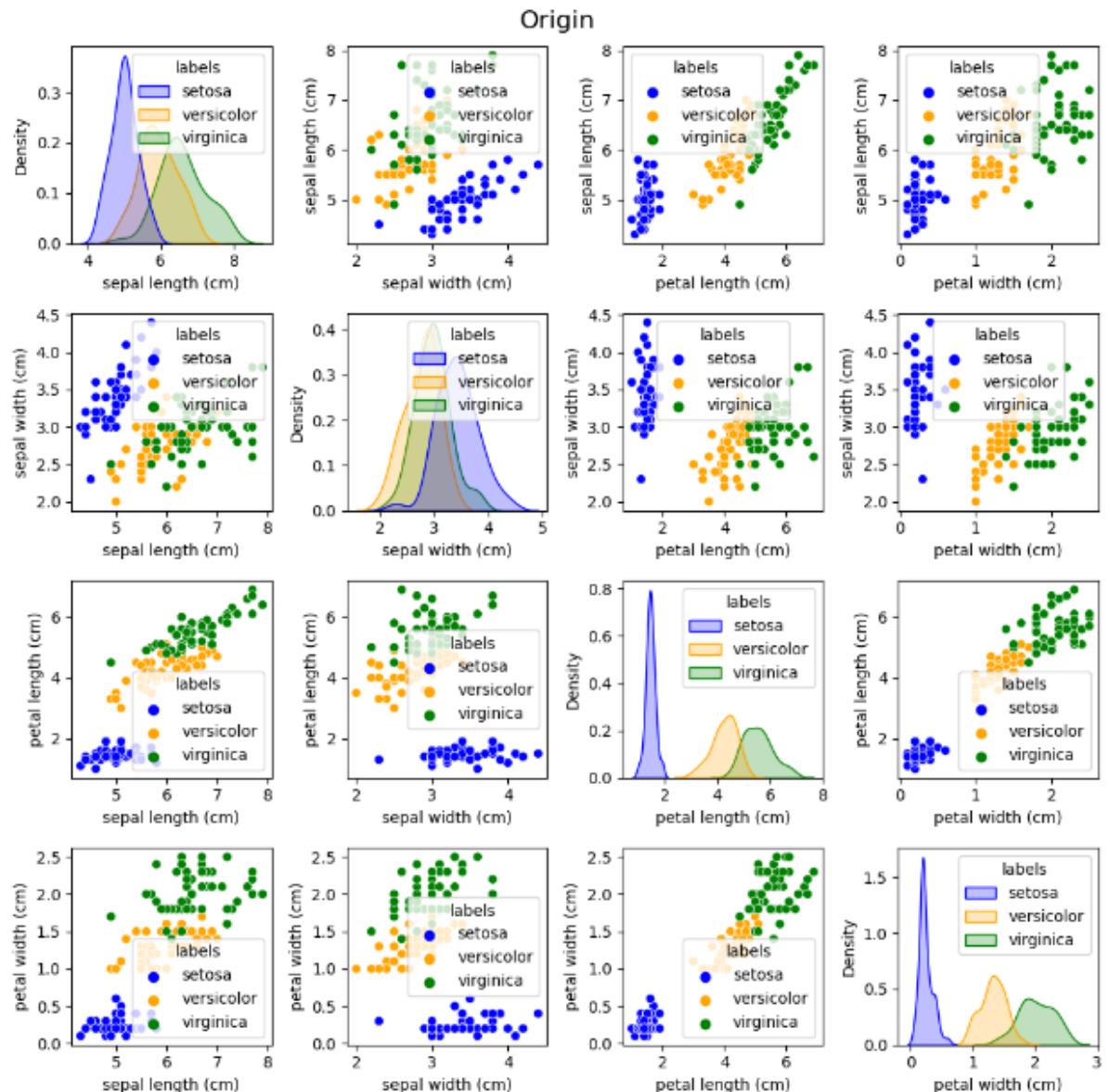
EM



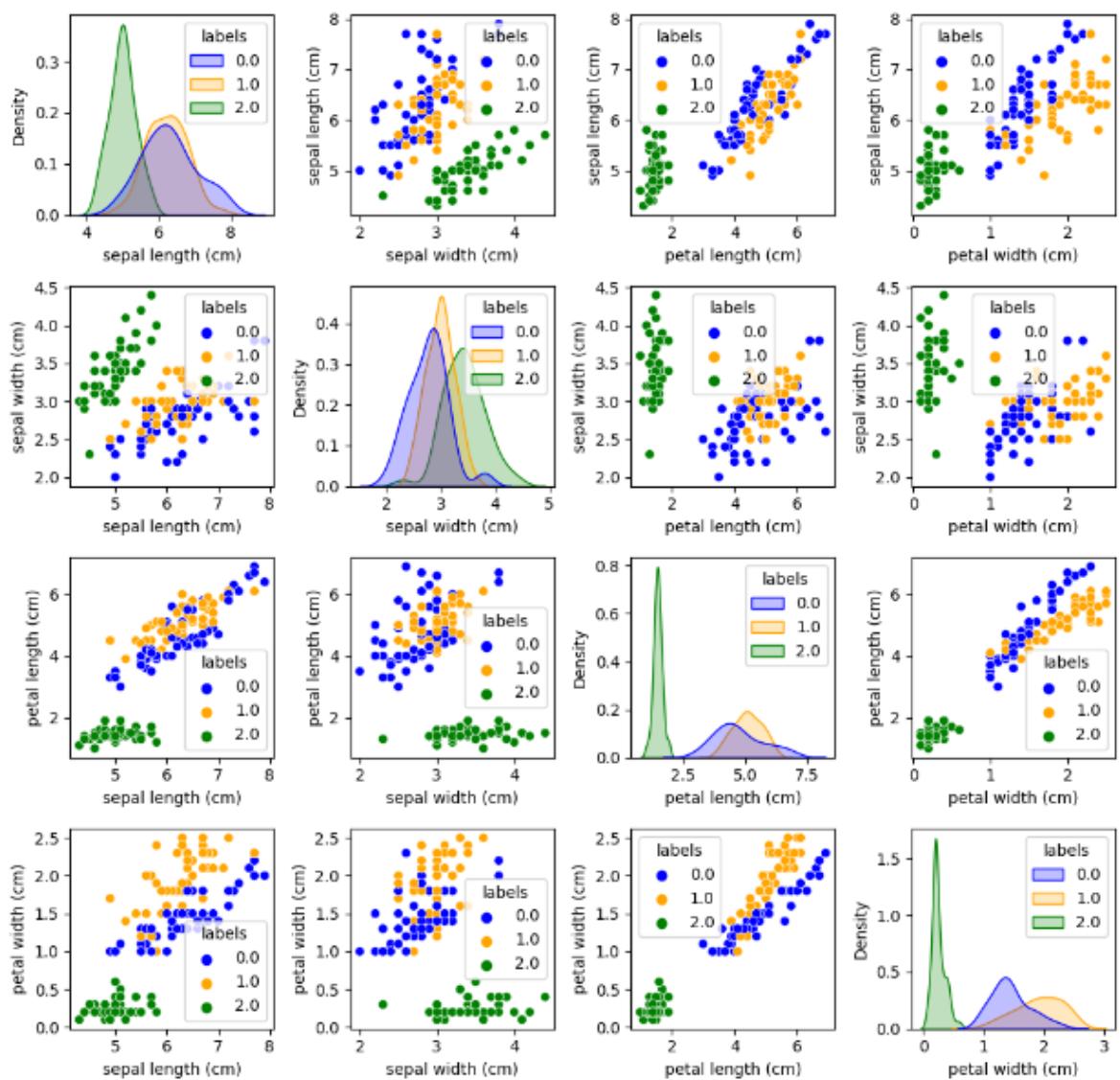


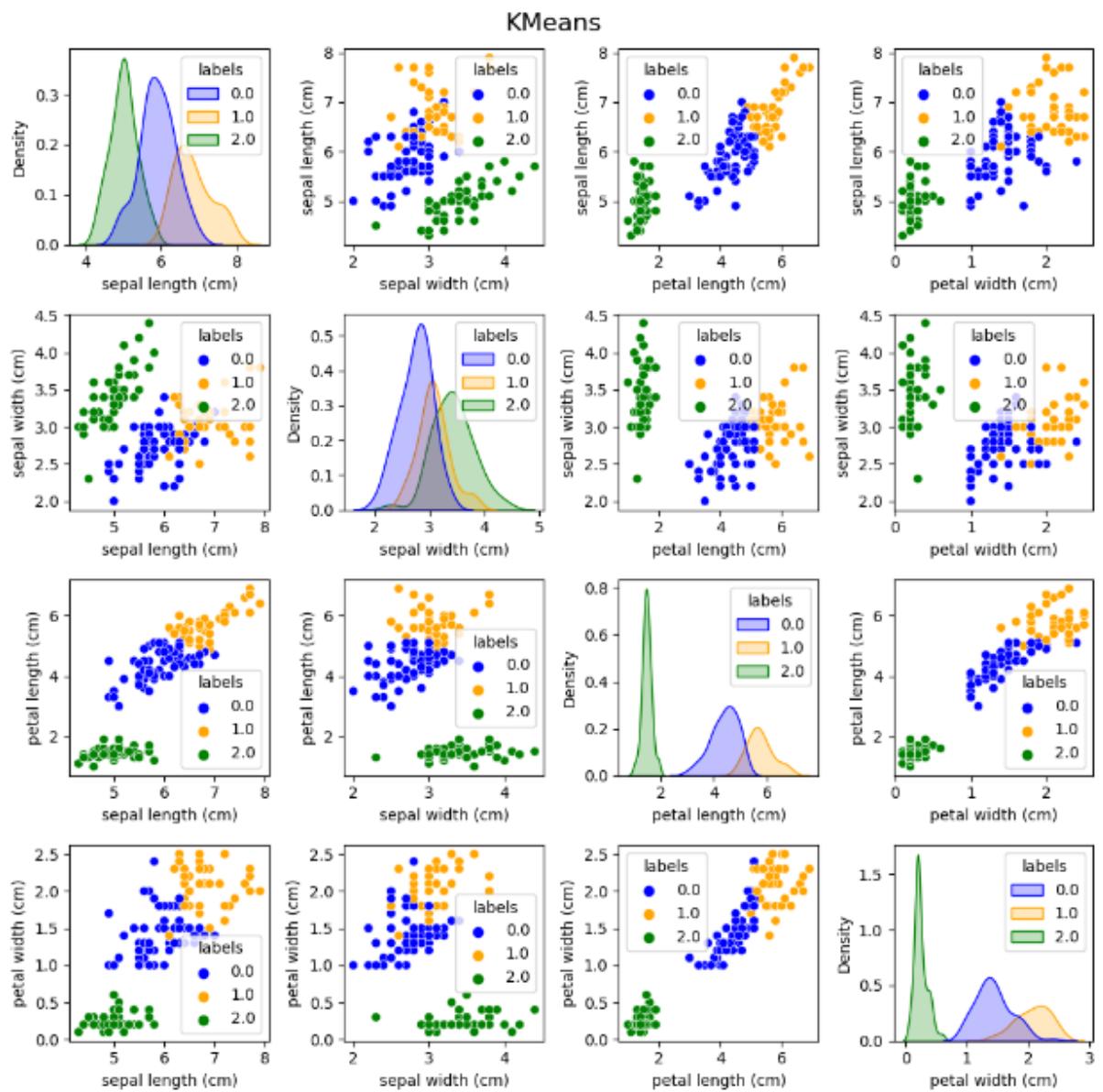
5 times

```
pi : [0.32548012 0.64164841 0.68819648]
count / total : [0.34888887 0.32 0.68888888]
/usr/local/lib/python3.10/dist-packages/sklearn/clus
    warnings.warn(
EM Accuracy: 0.8 Hit: 120 / 150
KM Accuracy: 0.89 Hit: 134 / 150
```



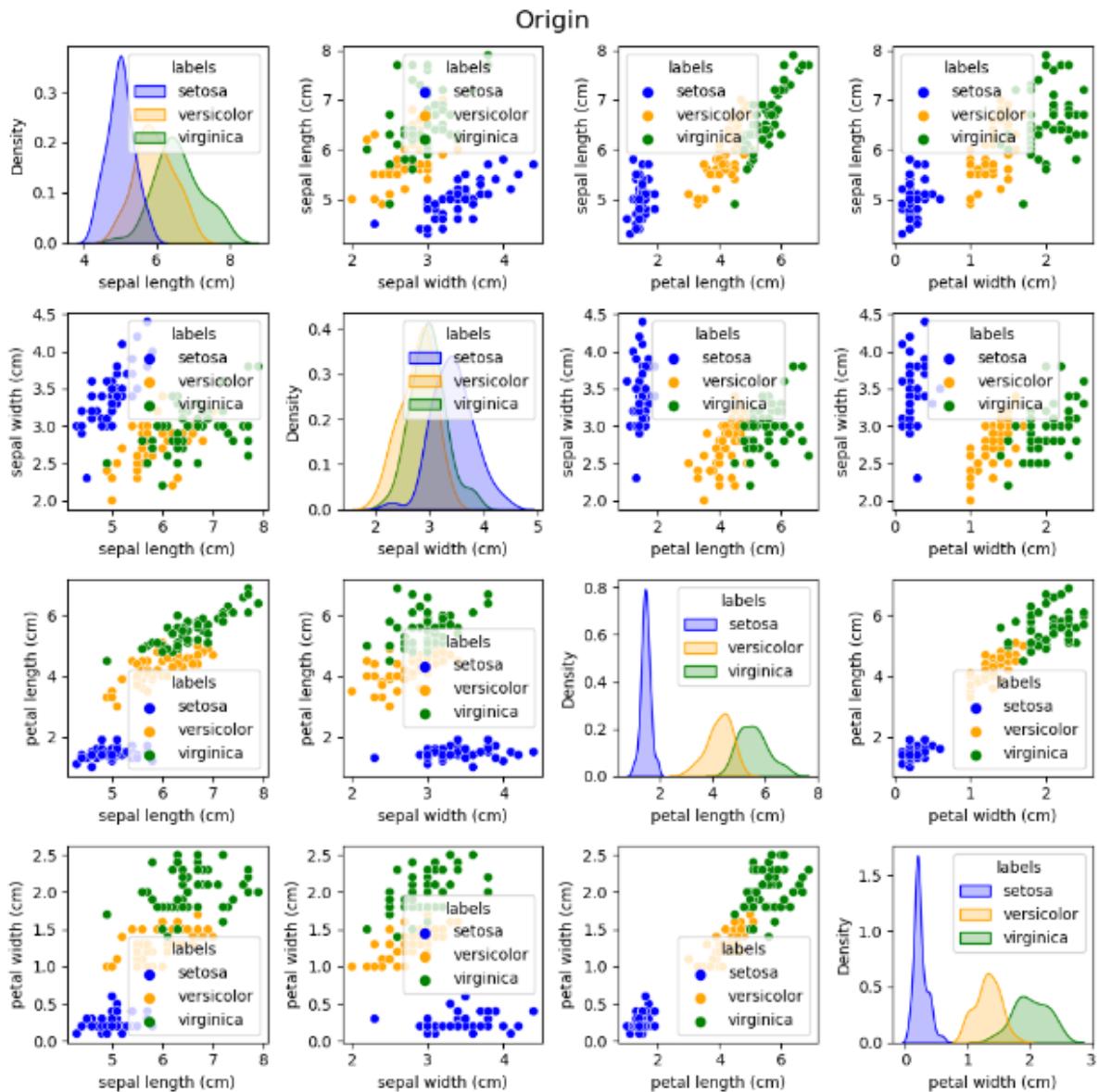
EM



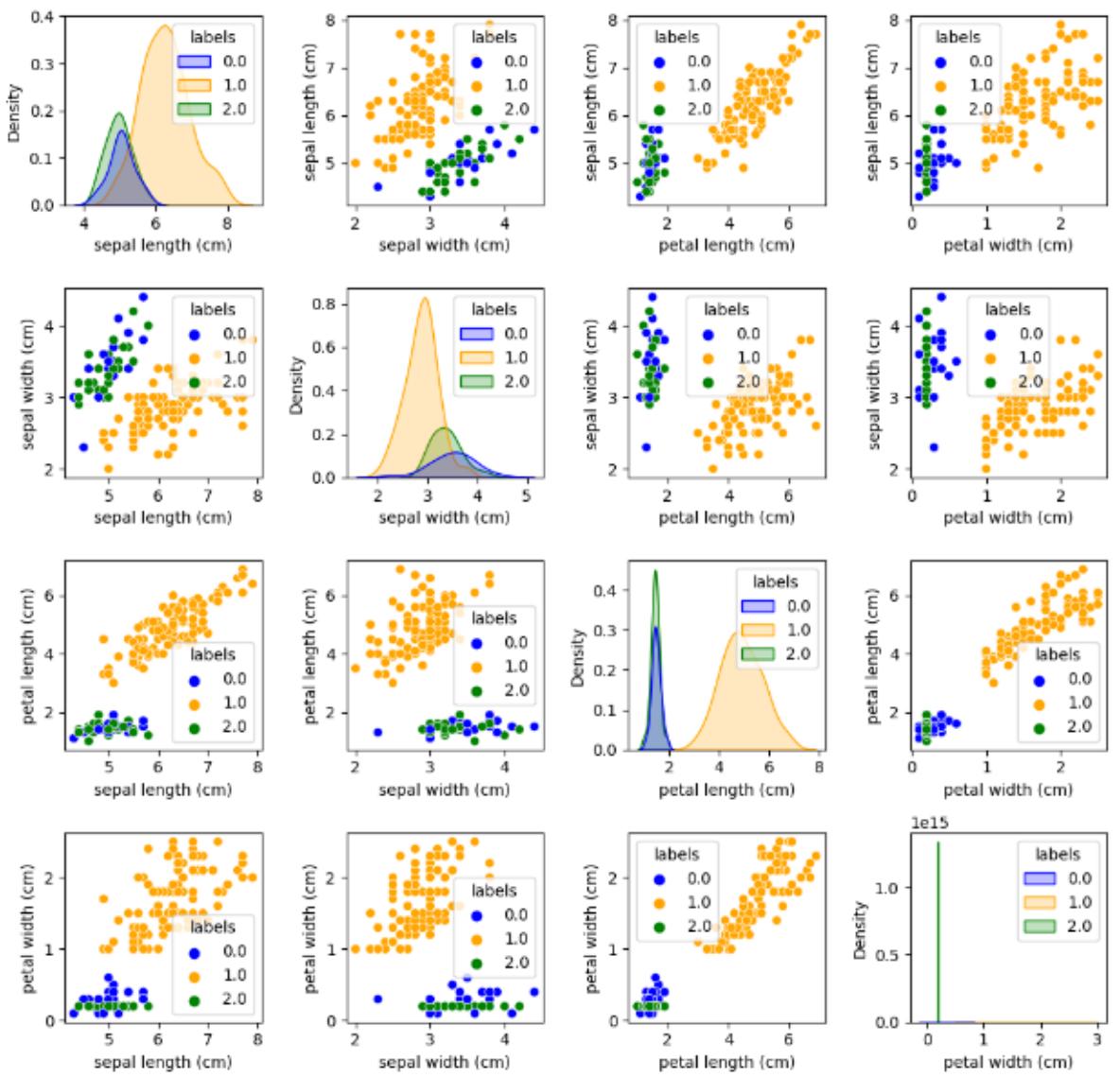


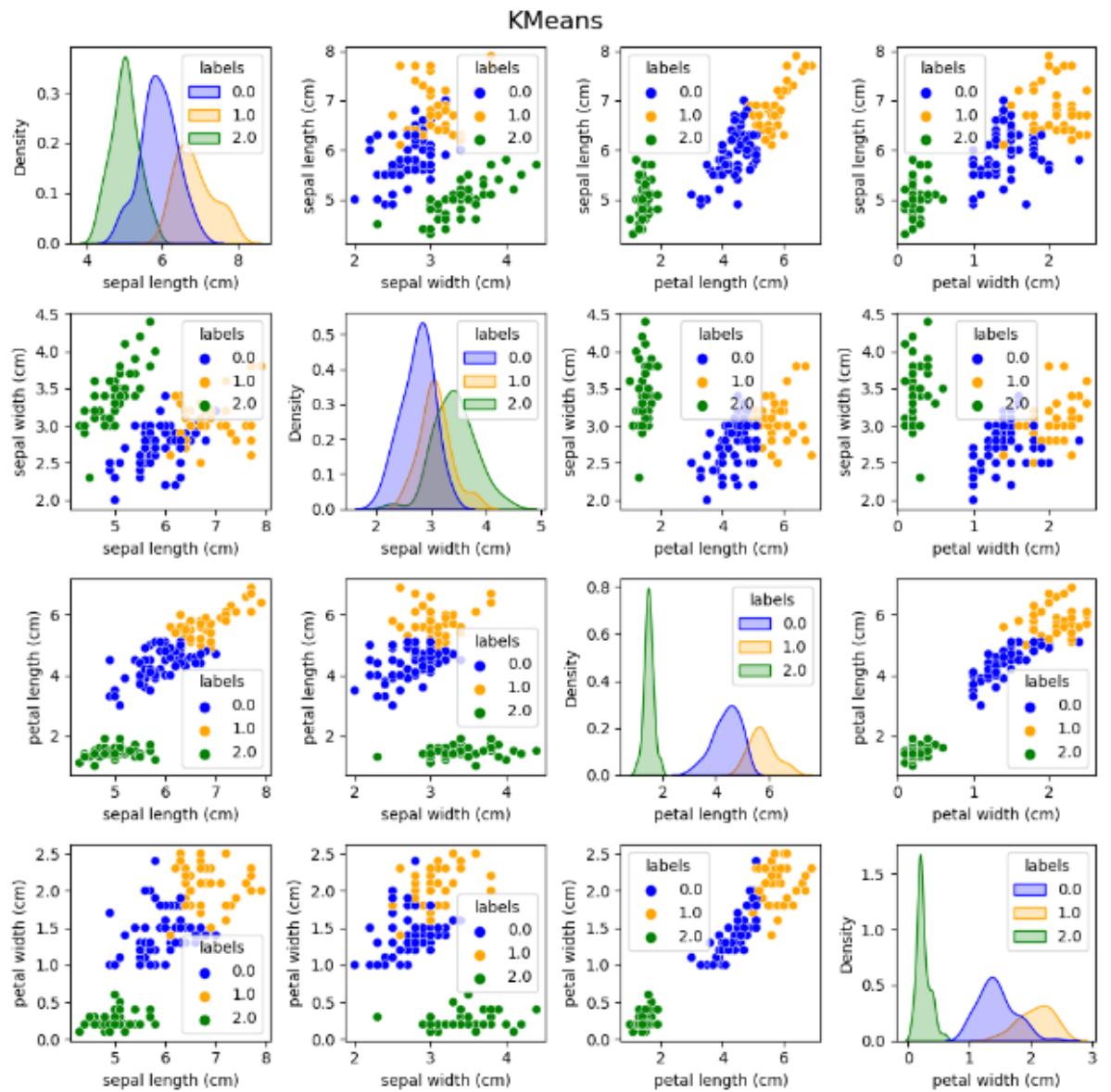
6 times

```
pi : [0.18999883 0.68888788 0.19888888]
count / total : [0.14 0.88888887 0.19888888]
/usr/local/lib/python3.10/dist-packages/sklearn/clus
    warnings.warn(
EM Accuracy: 0.68 Hit: 79 / 116
KNN Accuracy: 0.89 Hit: 104 / 116
```



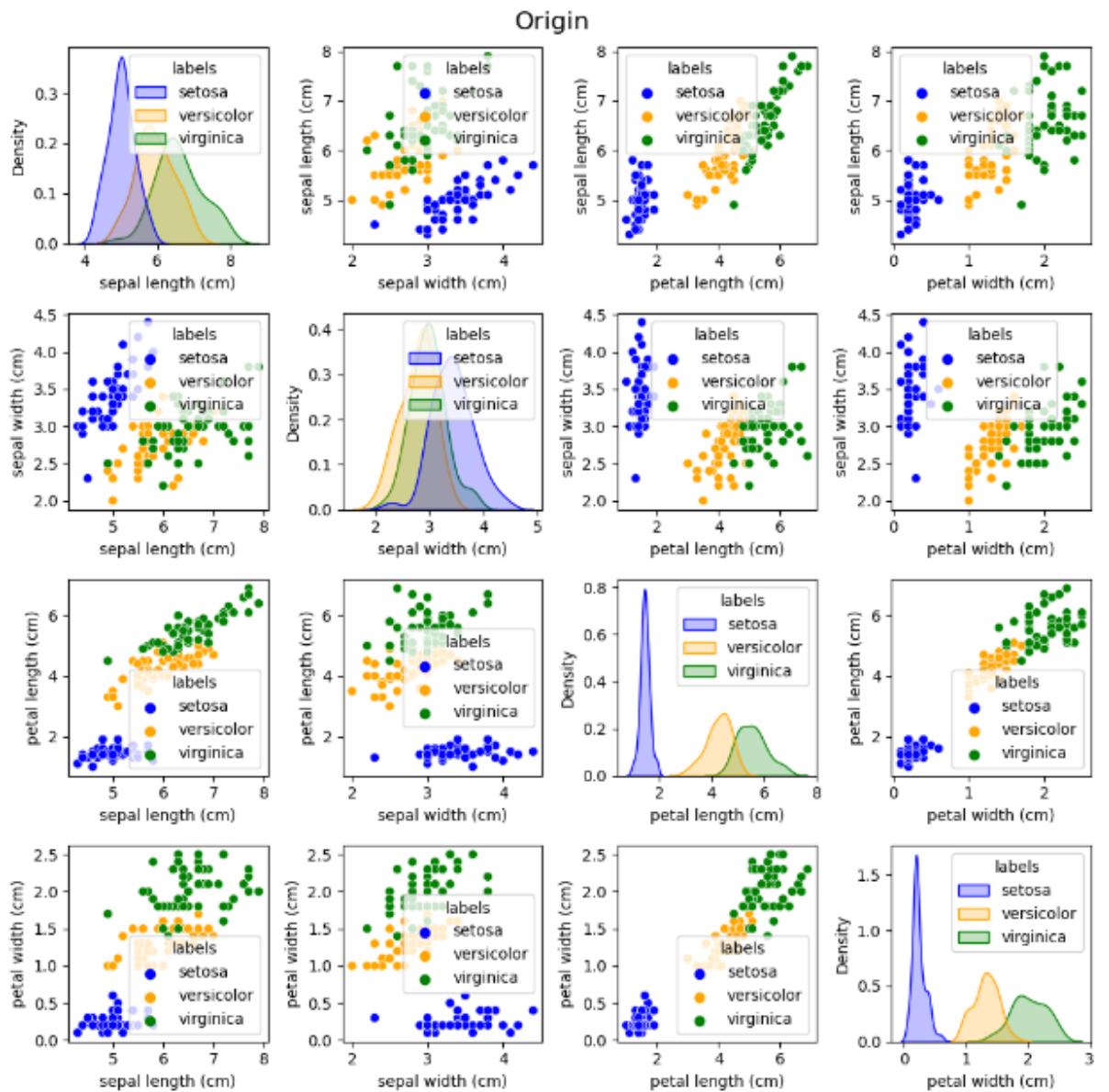
EM



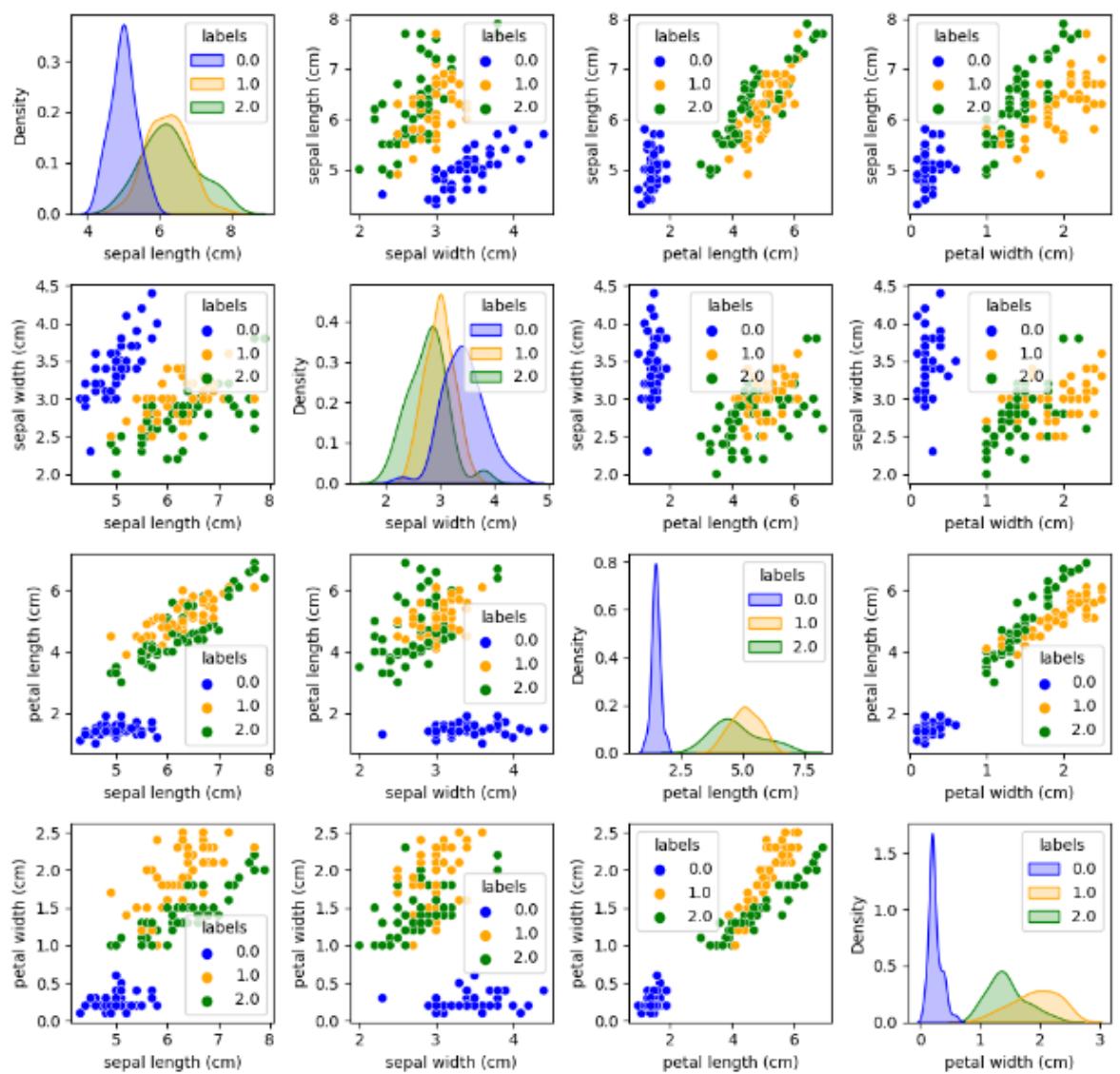


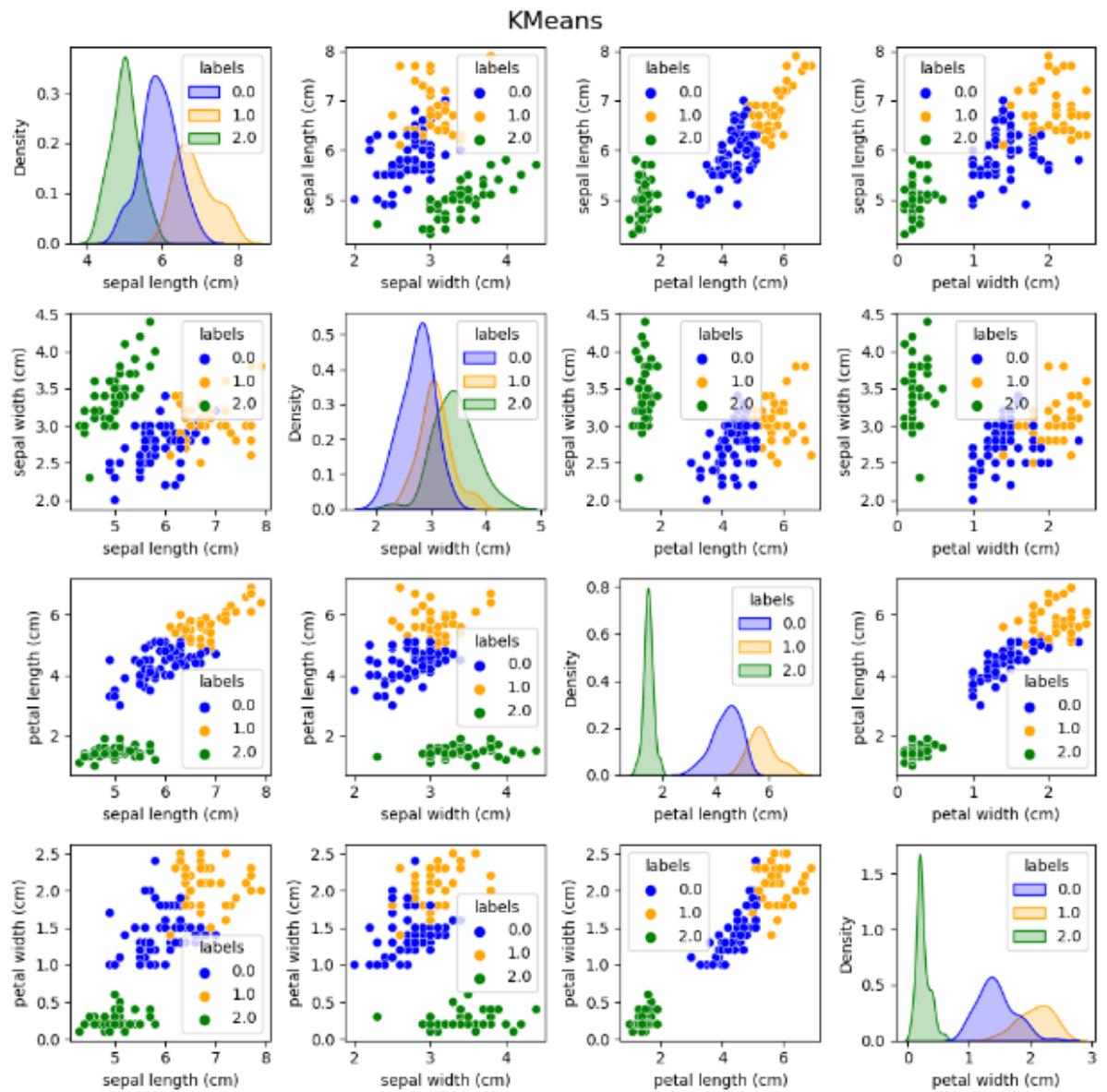
7 times

```
pl : [0.88819888 0.84060728 0.82629908]
count / total : [0.88888888 0.82 0.84888887]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster
    warnings.warn(
EM Accuracy: 0.8 Hit: 120 / 160
KM Accuracy: 0.89 Hit: 164 / 160
```



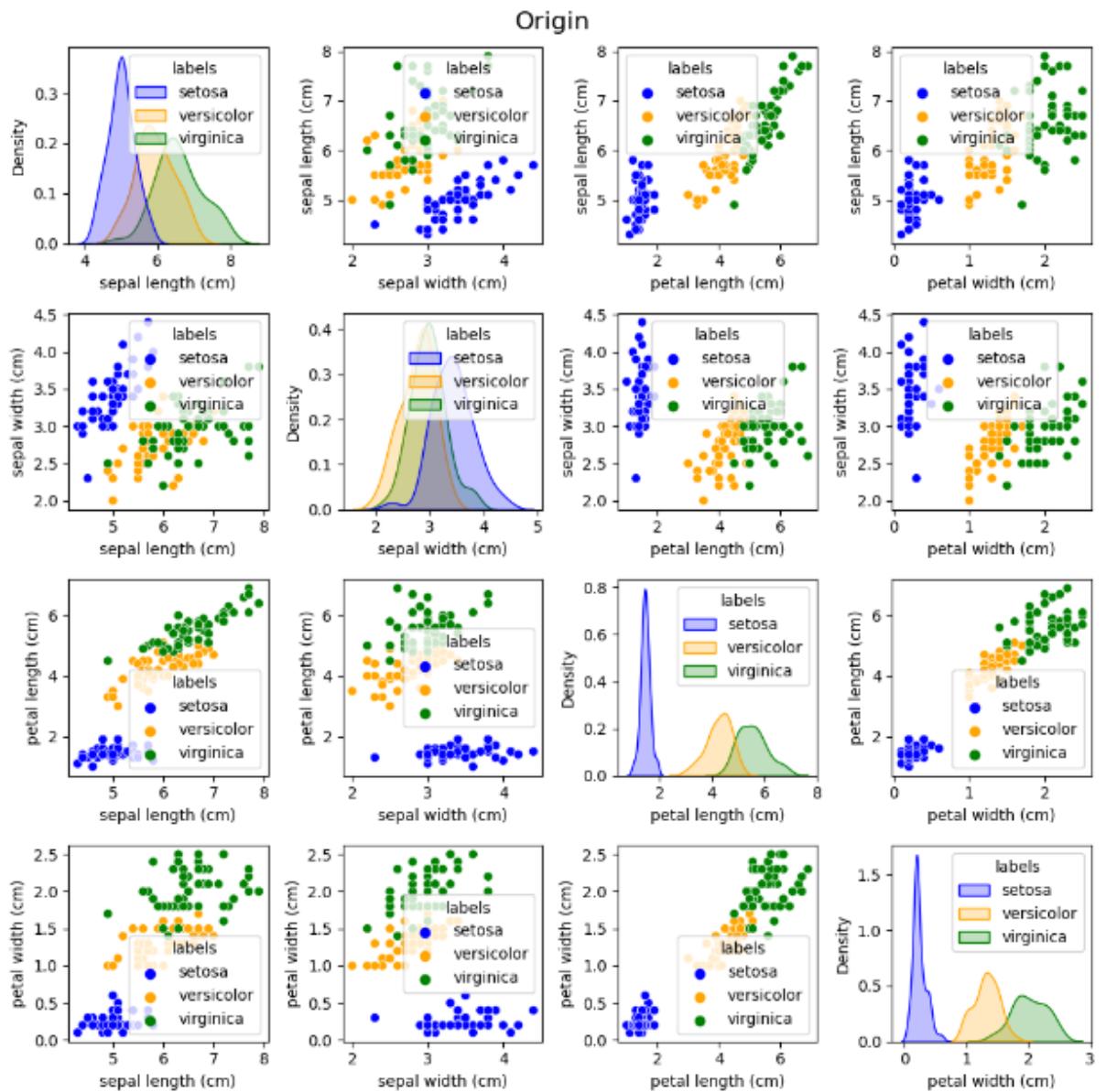
EM

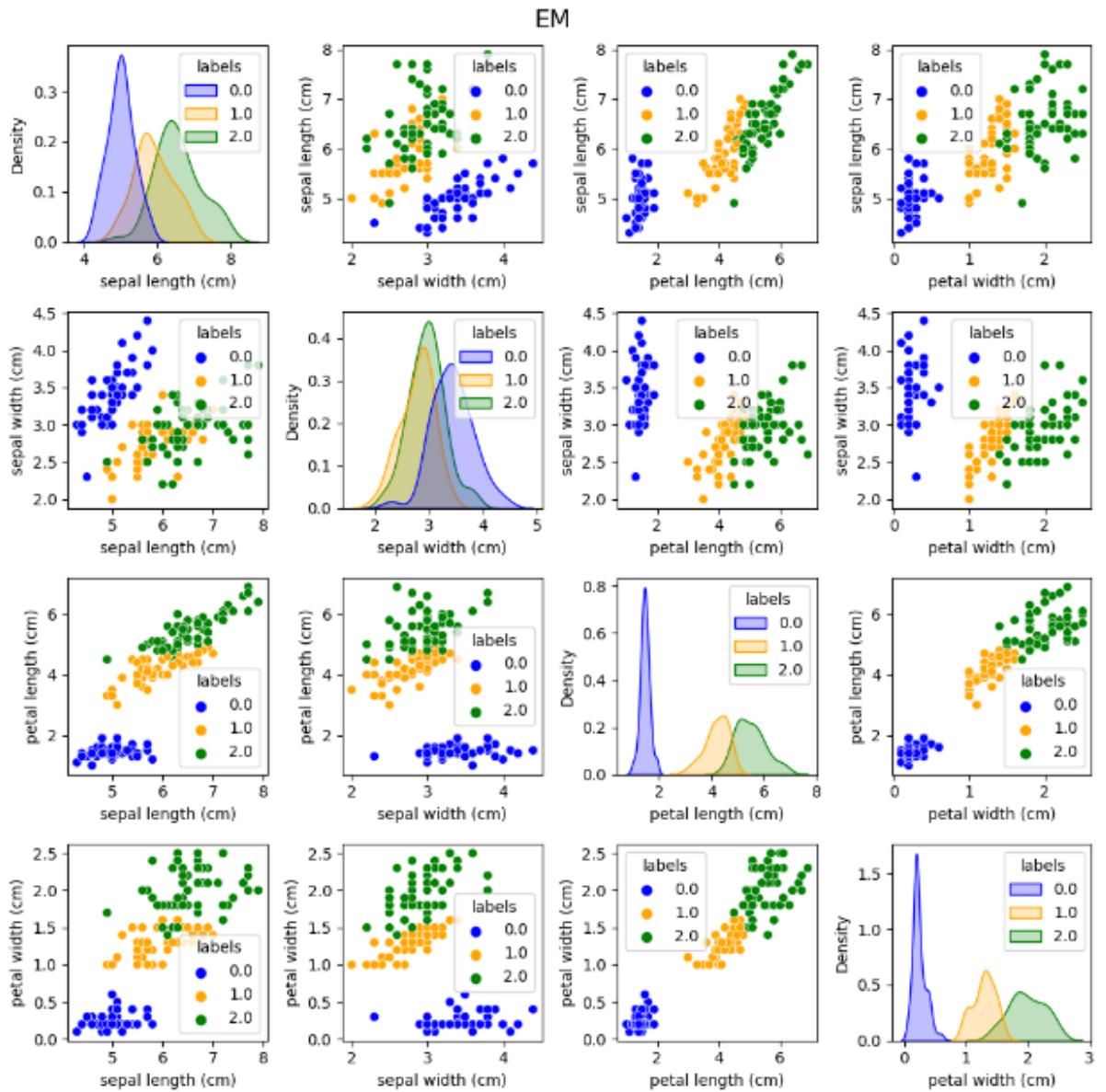




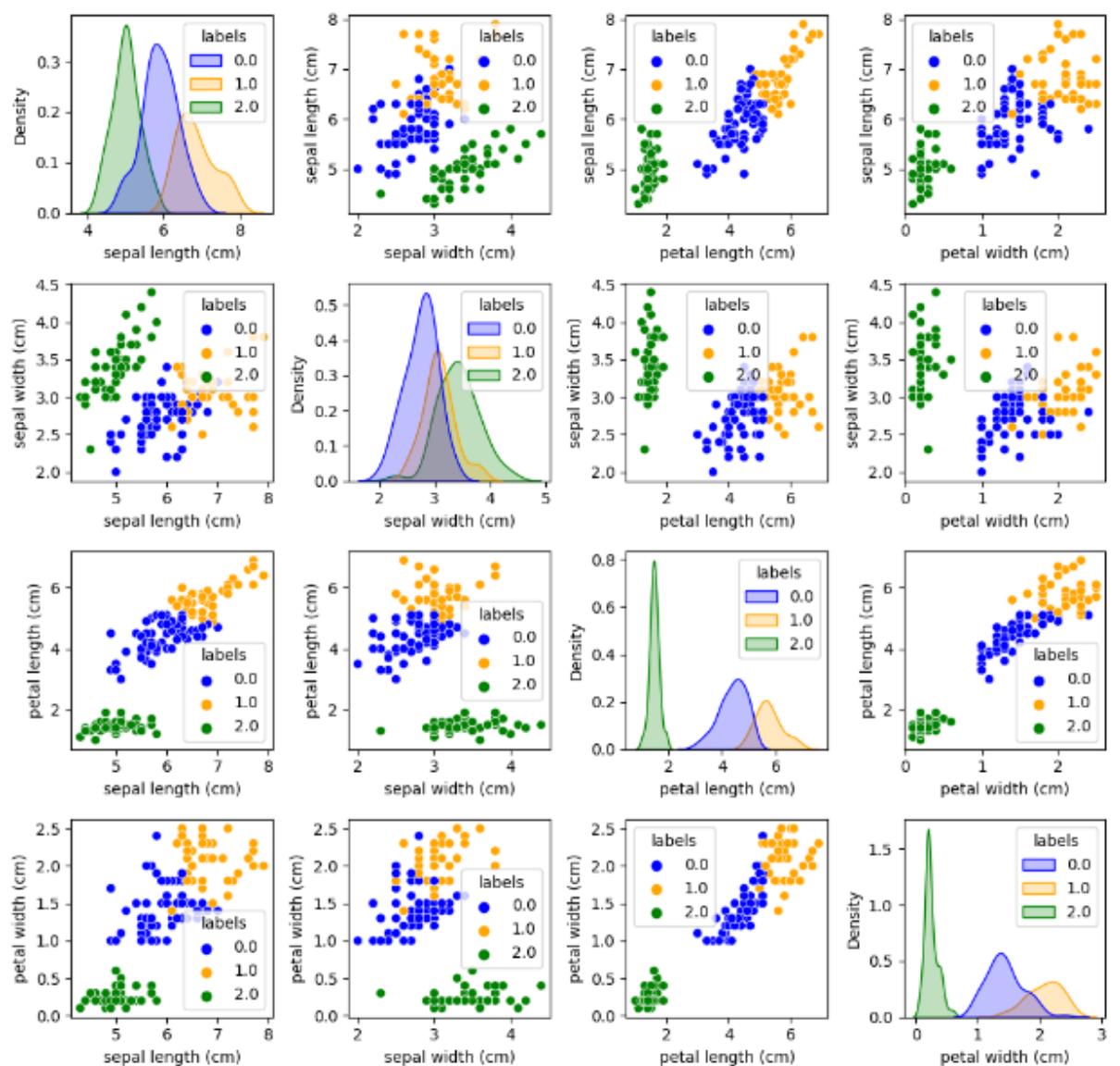
8 times

```
pi : [0.68888888 0.29919812 0.88747056]
count / total : [0.68888888 0.3 0.88888887]
/usr/local/lib/python3.10/dist-packages/sklearn/clus-
    warning.warn(
EM Accuracy: 0.97    HIT: 145 / 160
KM Accuracy: 0.89    HIT: 134 / 160
```



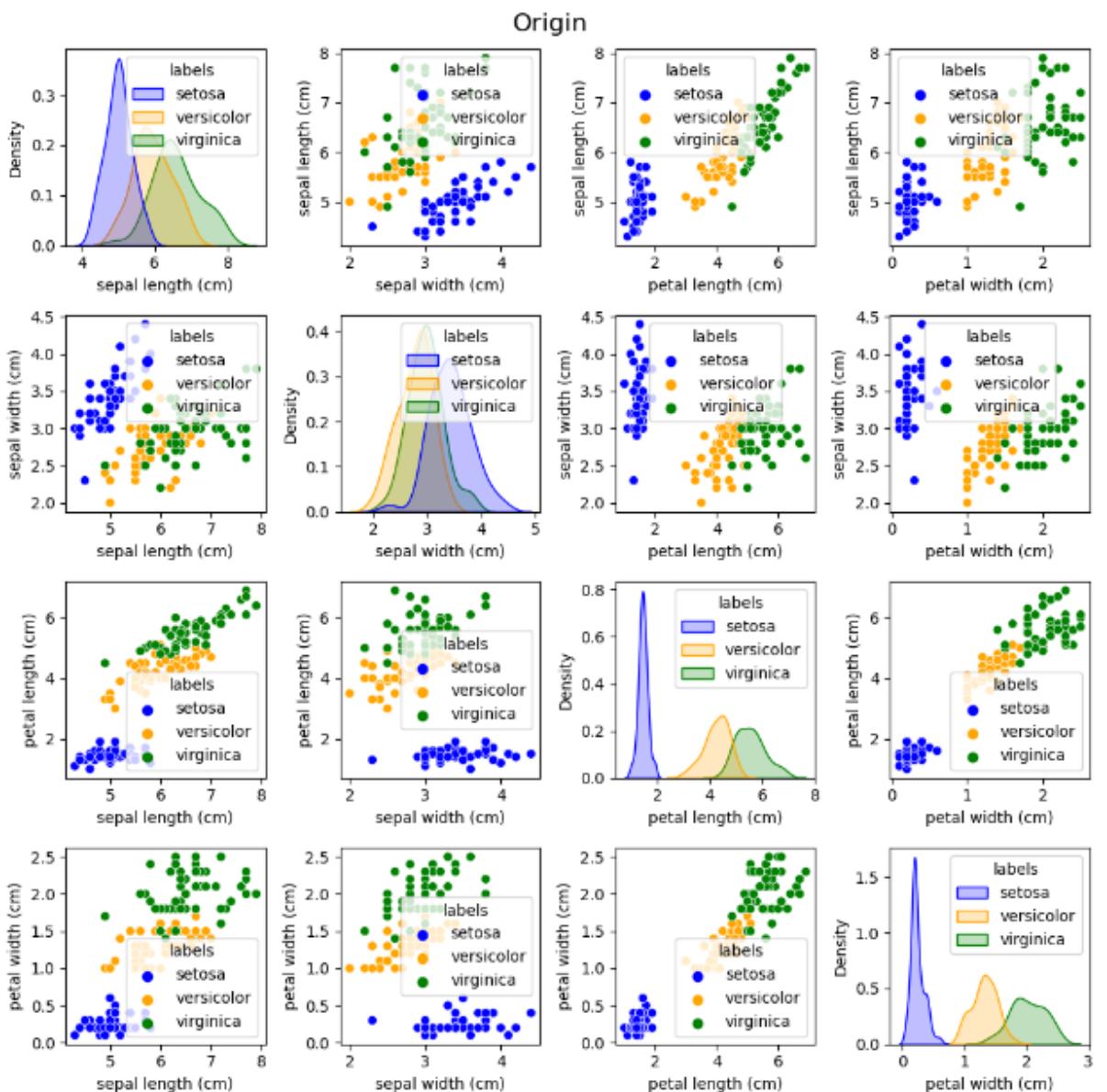


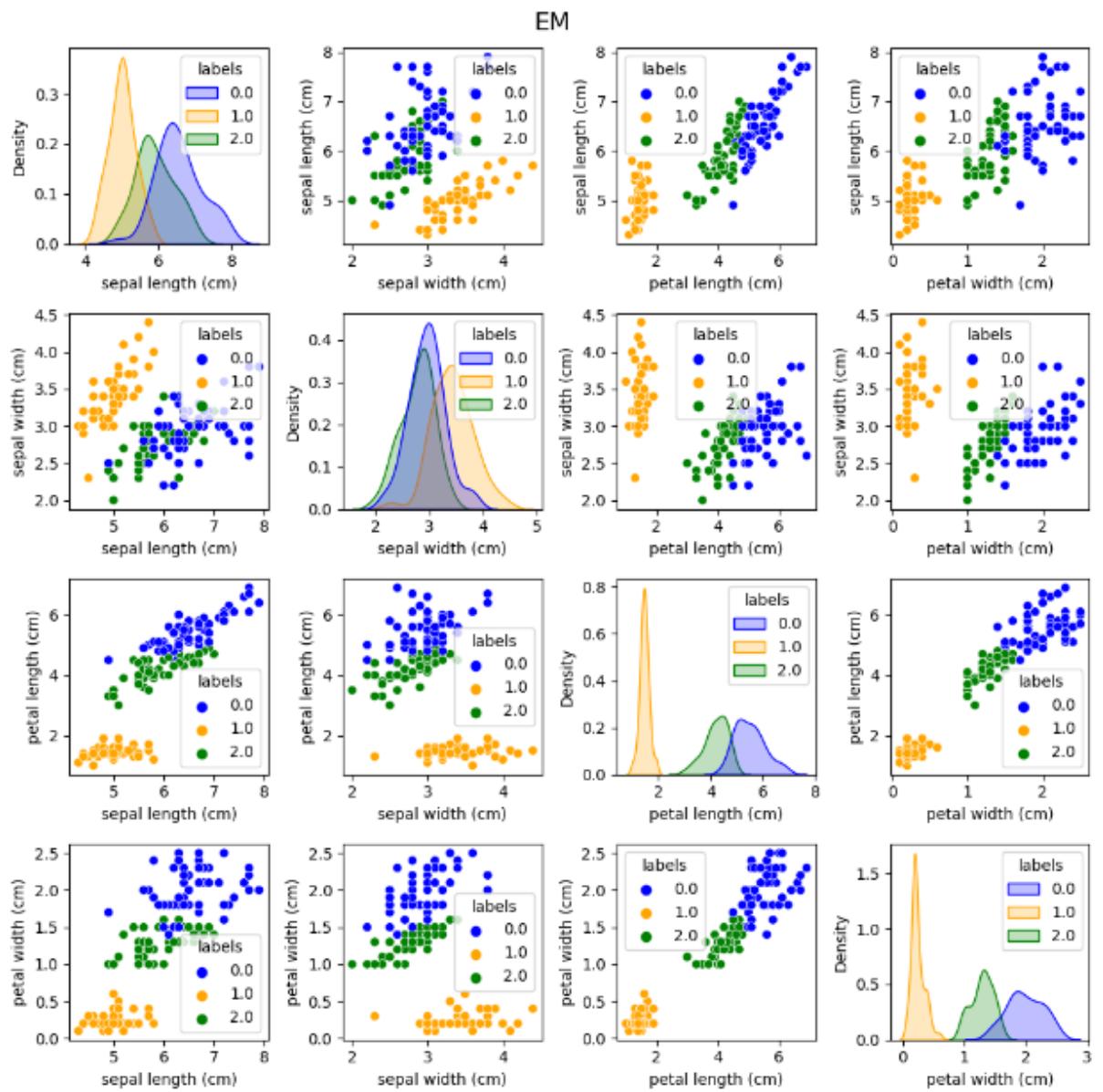
KMeans

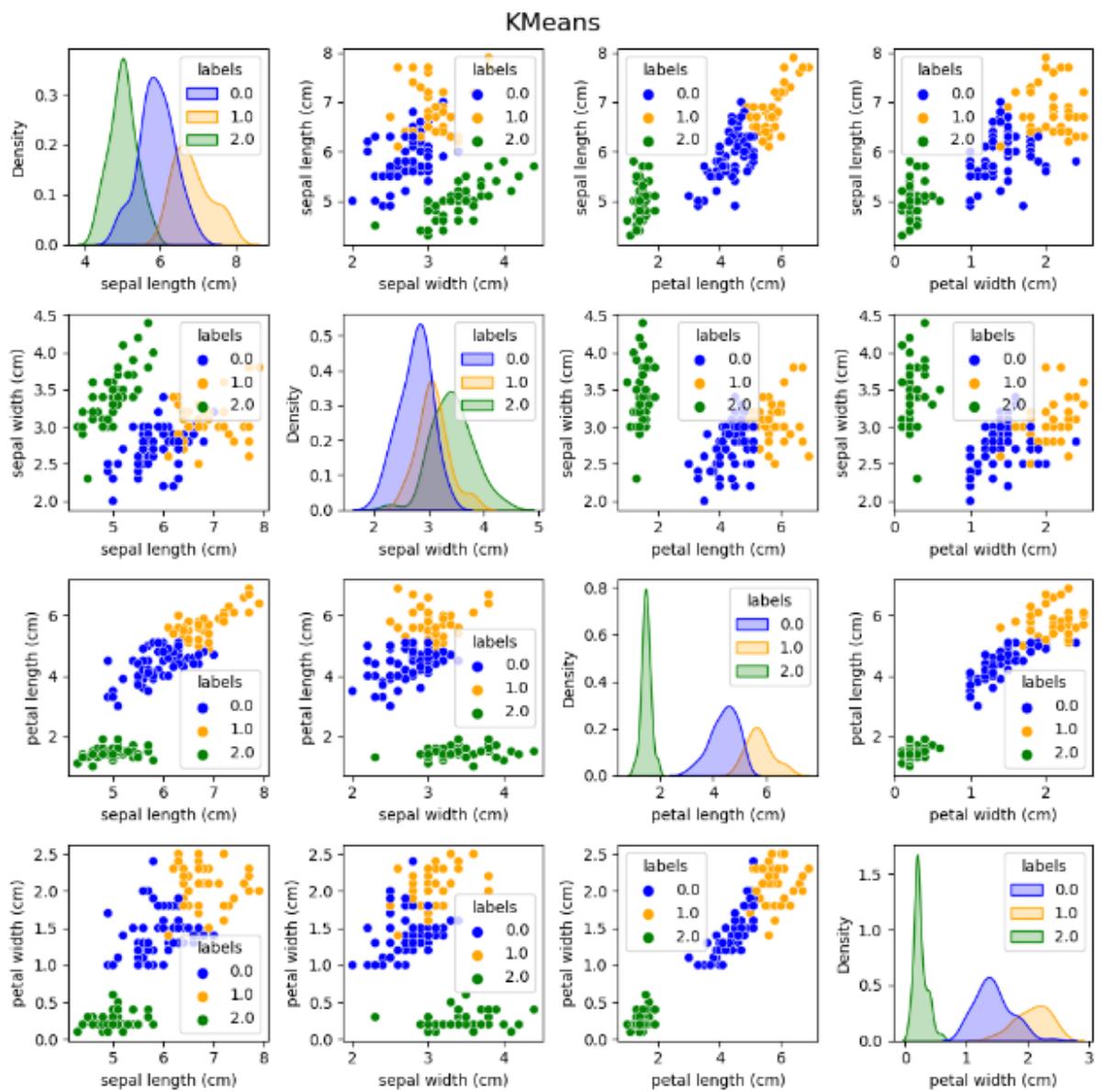


9 times

```
pI : [0.88747662 0.88888888 0.29919006]
count / total : [0.66666667 0.66666667 0.66666667]
/usr/local/lib/python3.10/dist-packages/sklearn/clu
warnings.warn(
EM Accuracy: 0.87    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

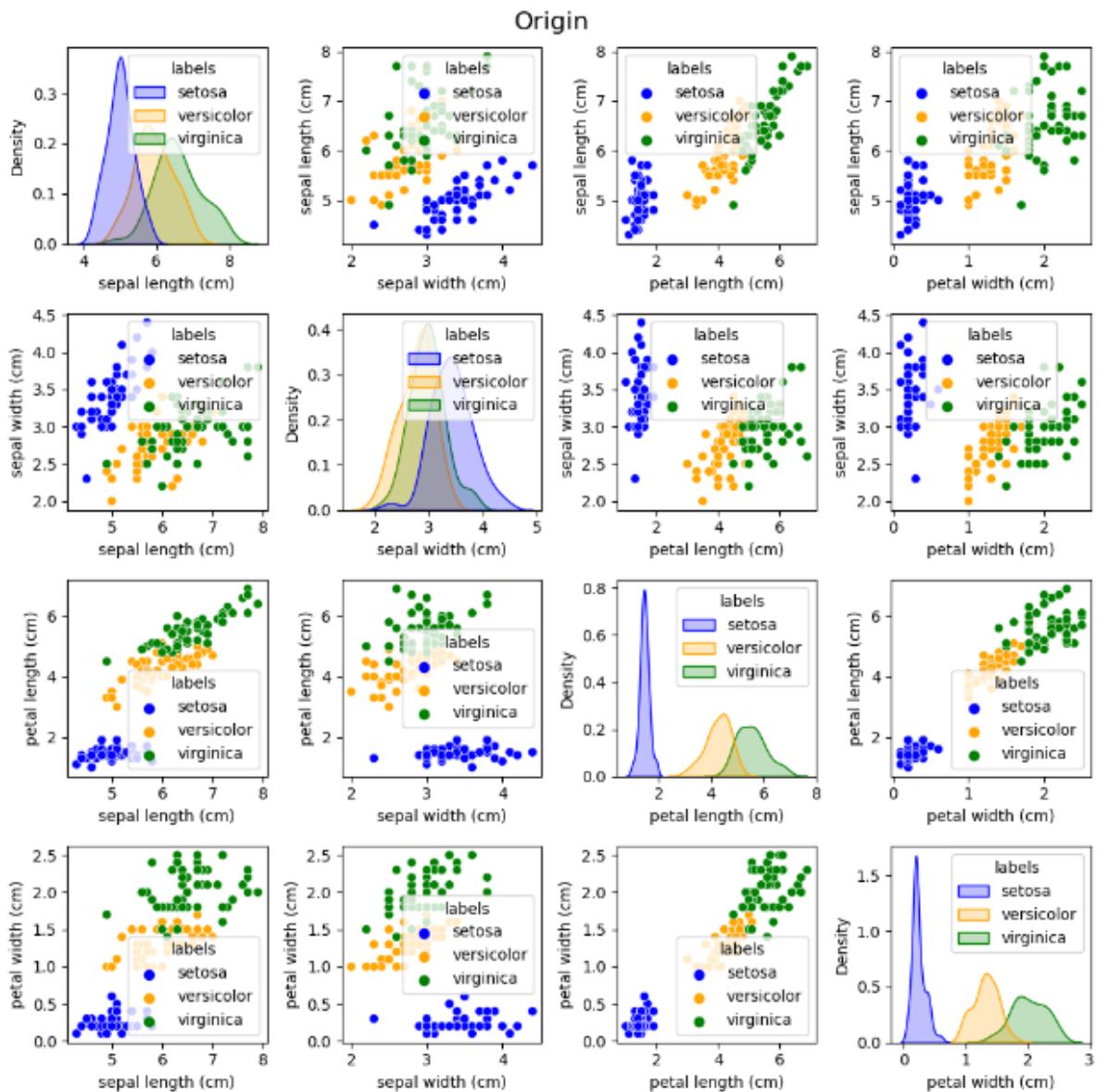




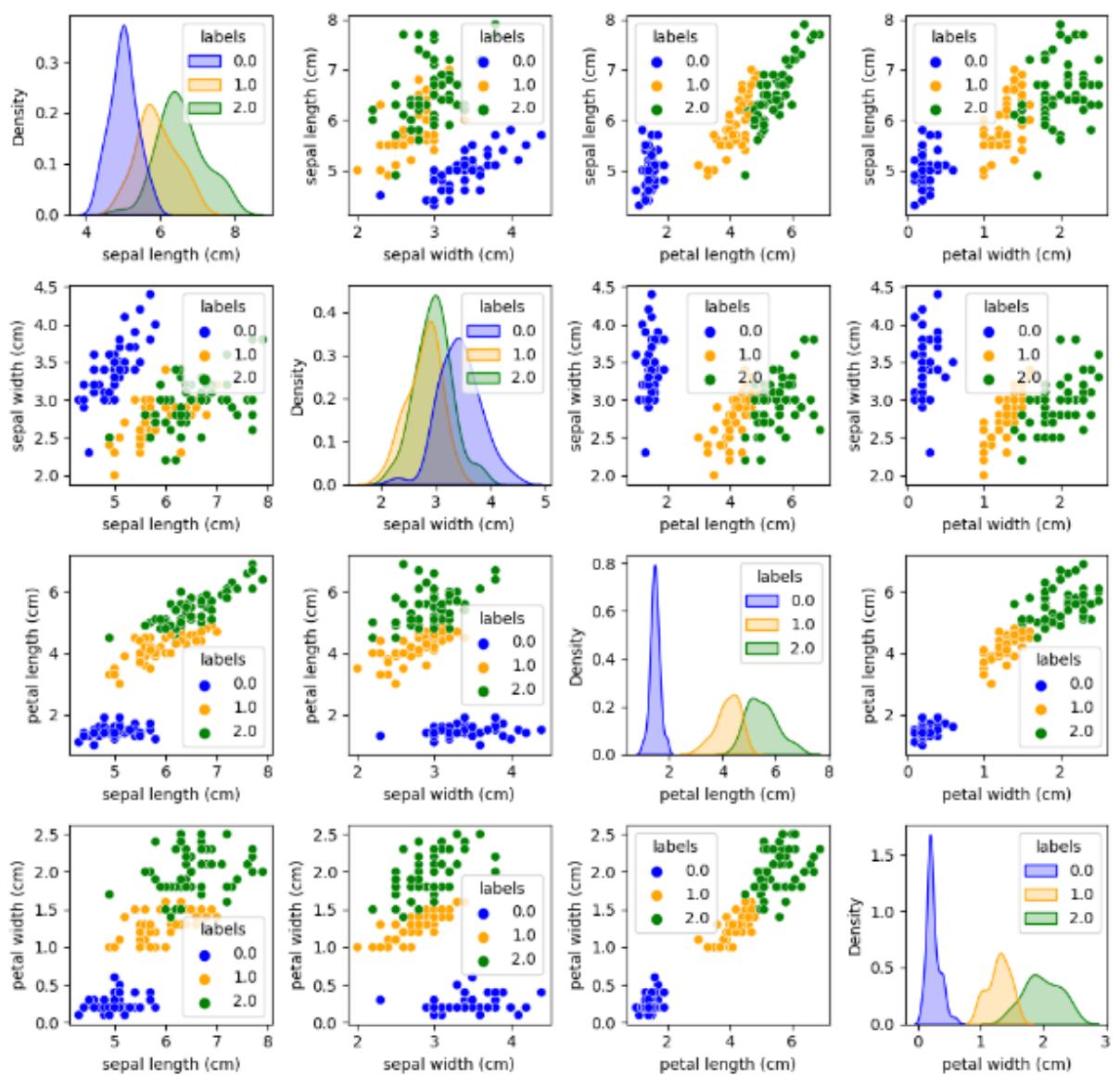


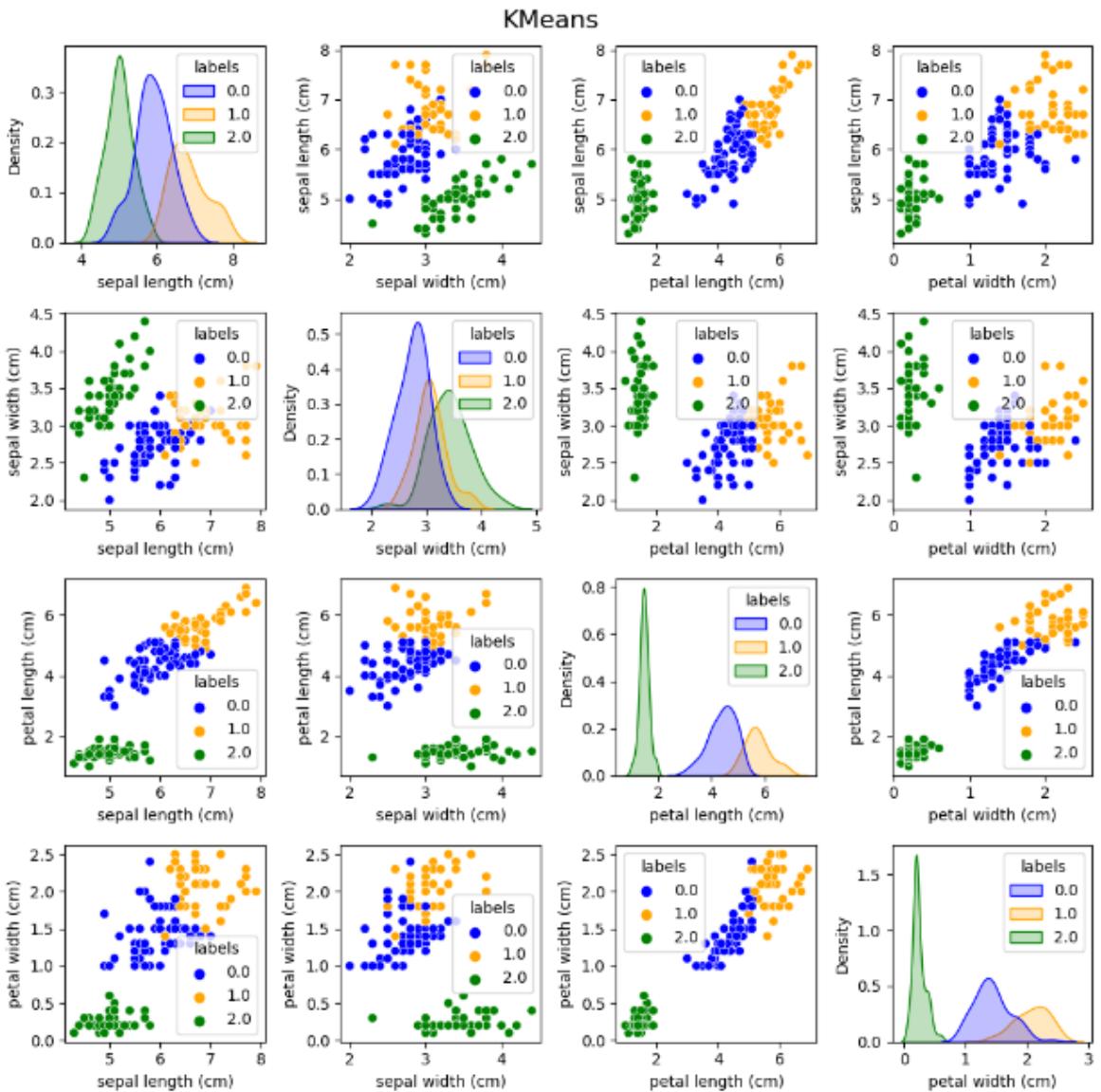
10 times

```
pi : [0.66666666 0.29919557 0.0874711 ]
count / total : [0.66666666 0.29919557 0.0874711 ]
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeanswarn()
EM Accuracy: 0.97    Hit: 145 / 160
KM Accuracy: 0.89    Hit: 134 / 160
```



EM





위 그래프는 original data, EM 알고리즘을 통해 추정된 결과, KMeans 알고리즘을 통해 추정된 결과를 그래프로 출력한 결과이다.

위 결과는 EM 알고리즘의 p_i 값과 각 클러스터 레이블을 카운트한 것이다. 결과 값에서 p_i 는 EM_model.pi를 출력한 것으로 GMM에서 추정한 각 클러스터의 사전 확률(prior)을 나타내는 값이다. 이 값은 EM 알고리즘을 통해 추정된 각 클러스터의 비율을 나타낸다. count / total에서는 EM 알고리즘을 통해 예측된 클러스터 레이블을 이용하여 각 클러스터의 비율을 출력한 것이다. np.bincount 함수를 사용하여 각 클러스터 레이블의 빈도수를 계산하고 이를 전체 데이터 수인 150으로 나누어 비율을 구한 것이다. 출력 결과는 각 클러스터의 데이터 포인트 비율이 출력된다. EM 알고리즘을 통해 최적의 파라미터에 도달할수록 EM 모델 객체의 p_i 속성과 EM 알고리즘을 통해 나온 클러스터의 비율의 값은 서로 비슷할 것이다.

EM 알고리즘의 정확도는 0.97이 제일 높게 나왔고 KMeans 알고리즘의 정확도는 0.89로 나왔다.

```
# Why are these two elements almost the same? Write down the reason in your report. Additional 10 points
print(f'pi : {EM_model.pi}')
print(f'count / total : {np.bincount(EM_pred) / 150}')
```

EM_model.pi는 GMM에서 추정한 각 클러스터의 사전 확률을 나타내는 값이다. np.bincount(EM_pred)는 EM 알고리즘으로 예측한 각 데이터 포인터의 클러스터 레이블을 카운트한 것을 반환한 것이다. 이를 전체 데이터 수인 150으로 나눠서 각 클러스터의 비율을 구한 것이다. EM_model.pi와 np.bincount(EM_pred) / 150 값이 비슷한 이유는 EM 알고리즘 성질 때문인데 EM 알고리즘은 초기에는 무작위로 클러스터를 할당하고 이후 데이터 포인트들이 새로운 클러스터로 재할당되며 이를 반복하면서 클러스터의 사전확률과 클러스터의 파라미터를 추정한다. EM 알고리즘은 모든 데이터 포인트에 대해 클러스터 재할당이 끝난 후에만 파라미터를 업데이트 하므로, 이 과정을 여러 번 반복하다 보면 클러스터의 비율과 사전 확률이 수렴하게 나온다. 그러므로 EM 알고리즘으로 추정한 각 클러스터의 비율과 사전 확률은 수렴하게 되어 거의 같은 값을 가지게 된다.

4. 고찰

EM 알고리즘은 이론적으로 익혀서 어느정도 알고 있지만 직접 코드로 구현하는 것은 막막했던 것 같다. Posterior는 multivariate_gaussian_distribution을 이용하여 구하였고 나머지 파라미터인 mean, sigma, pi는 posterior로 다 구현할 수 있다. 구현할 때 헷갈렸던 부분이 행렬 연산을 할 때 shape가 맞지 않아서 reshape와 for문을 통해 행렬 연산을 해야 한다는 점이었다. 특히 파라미터 중에 sigma를 구할 때 shape가 맞지 않아서 어떻게 연산을 해야 할 지 고민을 많이 했던 것 같다. 이번 과제를 통해 EM 알고리즘 구현에 대해 자세히 배운 것 같다.

5. 참고문헌

Seaborn을 사용한 데이터 분포 시각화 /

<https://datascienceschool.net/01%20python/05.04%20%EC%8B%9C%EB%B3%B8%EC%9D%84%20%EC%82%AC%EC%9A%A9%ED%95%9C%20%EB%8D%B0%EC%9D%B4%ED%84%BB%20%EB%B6%84%ED%8F%AC%20%EC%8B%9C%EA%B0%81%ED%99%94.html>

선형판별분석법과 이차판별분석법 /

<https://datascienceschool.net/03%20machine%20learning/11.01%20%EC%84%A0%ED%98%95%ED%8C%90%EB%B3%84%EB%B6%84%EC%84%9D%EB%B2%95%EA%B3%BC%20%EC%9D%B4%EC%B0%A8%ED%8C%90%EB%B3%84%EB%B6%84%EC%84%9D%EB%B2%95.h>

tml

메소드로 임의표본 추출하기 / <https://rfriend.tistory.com/548>

Python 행렬 생성 미 연산 / <https://generalbulldog.tistory.com/29>