Machine learning homework - 2

<EM Algorithm using K-means for GMM >

컴퓨터정보공학부

2020202013

고가연

I. 과제 제목 및 목적

본 과제는 GMM(Gaussian Mixture Model; 가우시안 혼합 모델)을 EM(expectation <-> maximization) 알고리즘을 사용해서 clustering(군집화)를 Python으로 구현한다. EM 알고리즘을 사용하는 GMM 모델을 이해하기 위해서 Initialization, multivariate_gaussian_distribution, expectation, maximization, fit, plotting 총 6개의 함수를 구현하는 것을 목적으로 한다. sklearn dataset의 Iris의 feature 데이터를 사용해서 3개의 class로 분류할 수 있도록 한다. 또한, EM 알고리즘을 구현한 다음 KMeans 알고리즘과의 성능을 비교 및 분석하기 위해 plot을 출력한다.

II. 소스코드에 대한 설명

GYK.py 파일에서 EM 클래스를 선언했고 그 안에 작성한 6개의 Python 함수를 아래와 같이 순서대로 설명하겠다. 해당 파일에서 sklearn dataset의 Iris feature data를 불러온 다음, 원래의 데이터 분포도와 EM 알고리즘을 적용한 후의 데이터 분포도, KMeans를 적용한 후의 데이터 분포도를 확인하고 성능을 비교한다.

또한, 파이썬에는 다른 프로그래밍 언어와 달리 main 함수가 존재하지 않기 때문에 더 효율적으로 코드를 짜기 위해서 if __name__ == '__main__': 이라는 조건문을 작성했다. 이는 직접 실행한 파일(GYK.py)이 __name__이라는 내장 변수에 __main__이라는 값이 들어가는 것이고 실제로 실행하면 이 조건문 아래는 직접 실행시켰을 때만 실행되길 원하는 코드만 넣어주는 것이라고 이해할 수 있다. 즉, __name__은 현재 모듈의 이름을 담고 있는 내장 변수이고 직접 실행된 모듈의 경우 __main__이라는 값을 가지게 되며, 직접 실행되지 않은 import된 모듈은 모듈의 이름(GYK)을 가지게 된다.

1. initialization

   -> 가우시안 혼합 모델(GMM)의 파라미터인 평균(mean), 공분산 행렬(sigma), prior probability(pi) 3가지 변수들을 초기화하는 함수이다. EM 클래스의 속성으로 선언한 mean 변수는 3x4 크기의 2차원 행렬로, 랜덤 데이터 점을 사용해서 초기화했다. 그리고 속성 변수인 sigma는 3x4x4 크기의 3차원 행렬로, 각 label에 해당하는 4x4 행렬을 np.eye() 함수를 사용해 대각 행렬로 초기화했다. 마지막 속성 변수로 pi는 각 label이 나타날 사전 확률로, 1x3 크기의 1차원 행렬이고 동일한 확률로

초기화했다.

2. multivariate_gaussian_distribution

   -> 해당 함수에서는 아래의 다변량 가우시안 분포 pdf를 계산하는 수식을 python 코드로 구현하였다.

$$\boldsymbol{N(x|\mu,\Sigma)} = \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))$$

   -> 해당 함수의 매개변수로 1x4 크기로 자른 data와, 1x4 크기로 자른 1차원의 mean과, 4x4 크기로 자른 2차원 행렬의 sigma를 받아온다. 1개의 iris는 4개의 feature를 가지기 때문에 Multivariate normal distribution을 사용해야 한다. 따라서, 위의 식을 파이썬으로 구현하기 위해서 Numpy의 선형 대수 함수들을 사용하였다. sigma의 행렬식을 구하기 위해서 np.linalg.det() 함수를 사용했고, np.linalg.solve() 함수를 사용해서 sigma의 역함수를 계산할 수 있었다. 결론적으로 계산된 pdf 값을 반환하도록 함수를 작성했다.

3. expectation

   -> 해당 함수는 EM 알고리즘에서 expectation 단계로, posterior probability를 계산하는 동작을 수행한다. 이중 for문을 사용해서 첫 번째 반복문은 data의 개수(N=150)만큼 반복하고 두 번째는 label의 개수(K=3)만큼 반복하도록 작성했다. 그리고 posterior_denom 변수를 정의해서 posterior를 계산하기 위해서 분모로 넣어야 하는 값을 저장하도록 별도의 반복문을 통해서 구했다. 아래의 수식은 posterior를 계산하는 식이다.

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

   -> multivariate_gaussian_distribution() 함수를 호출해서 각각의 pdf를 계산해서 해당 label의 pi 값을 곱해줌으로써 위의 식의 분자를 구할 수 있다. 두 번째 for문을 동작할 때마다 posterior[n][k] 값을 하나씩 계산함으로써 150x3의 matrix를 모두 다 채울 수 있다. 마지막으로 150x3

크기의 posterior를 반환한다.

4. maximization

   -> 해당 함수에서는 EM 알고리즘에서 maximization 단계로, expectation 단계에서 계산한 posterior probability를 사용해서 파라미터(mean, sigma, pi)를 추정하는 동작을 수행한다. 함수 내에 mean, sigma, pi 변수를 별도로 정의해서 계산하도록 했다. 왜냐하면 fit() 함수에서 maximization() 함수를 호출해서 3개의 파라미터가 일정한 값으로 수렴하는지를 검사하기 위해서 iteration 이전의 값들과 현재의 값들을 비교하는 조건문을 수행해야 하기 때문이다.

   -> 이중 for문을 작성함으로써 mean과 sigma를 계산하기 위해 mean_exp, sigma_exp를 새로 정의해서 각 label에 해당하는 분자 값들을 계산하였다. 먼저, mean_exp는 각 label 하나에 대해서 data의 개수만큼 반복문을 돌면서 그에 해당하는 posterior[n][k] 값과 data[n] (4x1 크기의 matrix) 값들을 곱하면서 누적하며 더한 값이다. 그리고, sigma_exp는 각 label 하나에 대해서 data[n]에서 mean[k]를 뺀 값과 그 값의 전치한 행렬을 외적 연산한 후, posterior[n][k] 값을 곱해서 누적하여 더한 값이다. 이때 외적 연산은 np.outer() 함수를 사용해 계산했다. 그리고나서, 해당 label에 대한 pi[k], mean[k], sigma[k]를 계산했다. pi[k]는 그 label에 해당하는 posterior 값 150개를 모두 더한 값을 data의 길이인 150으로 나눠서 계산했다. 그리고 mean[k]은 두 번째 for문으로 계산한 mean_exp 분자 값을 해당 label에 대한 posterior 값 150개를 모두 더한 값으로 나누어서 계산했다. 마지막으로 sigma[k]는 sigma_exp 분자 값을 해당 label에 대한 posterior 값 150개를 모두 더한 값으로 나누어서 추정하였다. 결론적으로 3개의 label(=cluster, class)에 해당하는 파라미터 값들을 모두 계산한 후, 추정한 mean, pi, sigma를 반한하였다.

5. fit

   -> 해당 함수는 앞서 구현한 initialization(), expectation(), maximization() 함수를 호출함으로써 EM 알고리즘을 적용해 가우시안 혼합 모델(GMM)을 fit하여 군집화할 수 있도록 동작한다. 반복문을 통해 iteration을 돌면서 expectation() 함수와 maximization() 함수를 번갈아 가면서 호출하

고 posterior probability와 3개의 파라미터(mean, sigma, pi)를 update하는 동작을 수행한다. 이때 반복문의 종료 조건은 모든 변수(posterior, mean, sigma, pi)가 일정한 값에 수렴하는 경우에 종료할 수 있다. 그리고 tolerance라는 수렴 허용 오차 변수를 1e-6으로 설정해 각 변수들의 이전 iteration 값과 현재 iteration 값의 차이를 계산 후, tolerance보다 차이 값이 더 작으면 수렴한 것으로 간주한다. 해당 조건문을 충족하지 못해서 반복문을 못 빠져나왔다면 다시 expectation 단계를 수행해야 하기 때문에 이전의 변수 값들을 복사하는 과정을 거치고 다음 expectation 단계에서 posterior 값을 새로 update 할 수 있도록 한다.

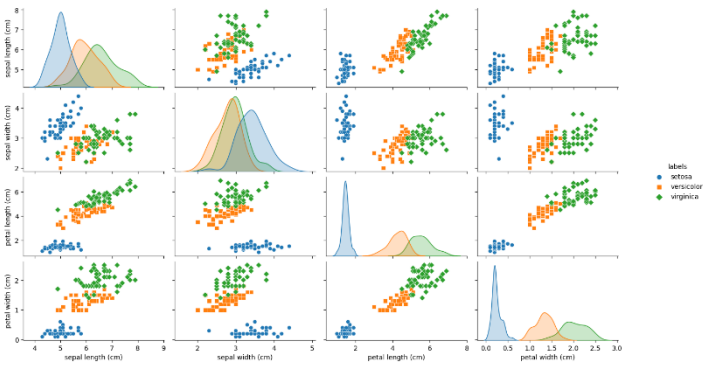
-> 수렴조건을 충족하여 반복문을 정상적으로 빠져나왔다면 최종적으로 수렴한 posterior를 가지고 MLE(Maximum Likelihood Estimation)을 사용해 prediction을 추정한다. 해당 함수의 반환 값으로 prediction을 사용함으로써 EM 알고리즘의 성능을 평가할 수 있다.

6. plotting

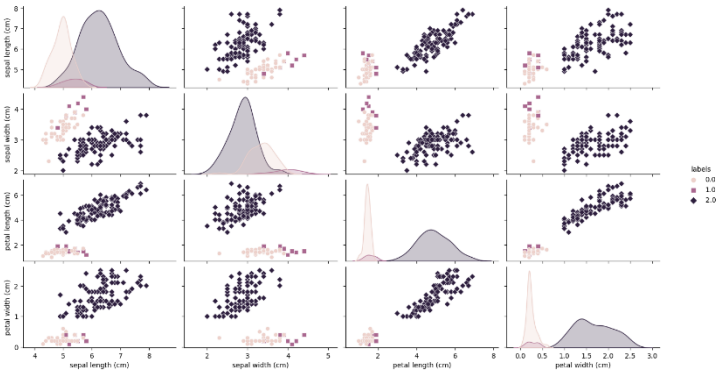
-> 마지막 함수로, 최종적으로 예측한 prediction을 사용해서 plot을 출력하는 동작을 수행한다. 이때 seaborn pairplot()를 사용했고 각 column별 데이터에 대한 상관관계나 분류적 특성을 보여주는 그래프 출력할 수 있다. 이는 clustering 알고리즘의 결과를 확인하거나 비교할 때 효과적이다.

III. 실행결과

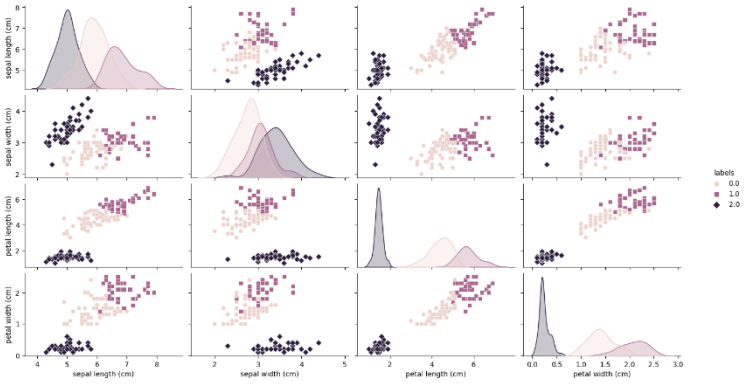
① 첫 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

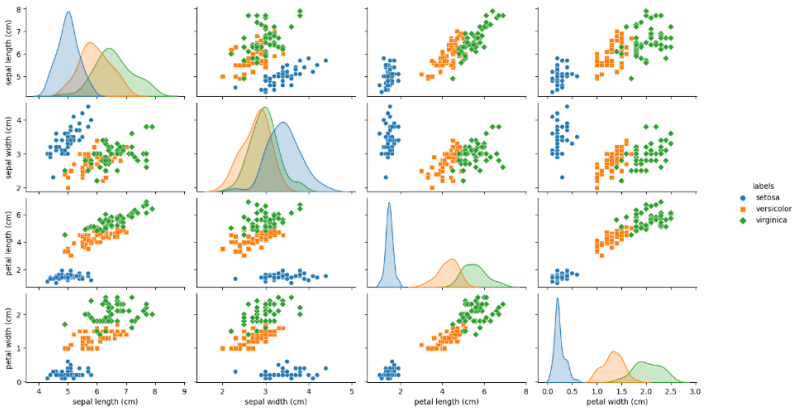


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

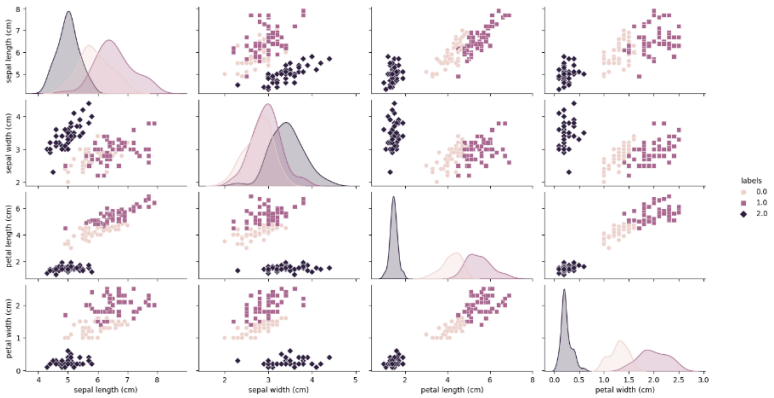
```
pi :           [0.28694449 0.04637276 0.66668275]
count / total : [0.28666667 0.04666667 0.66666667]
EM Accuracy: 0.62    Hit: 93 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

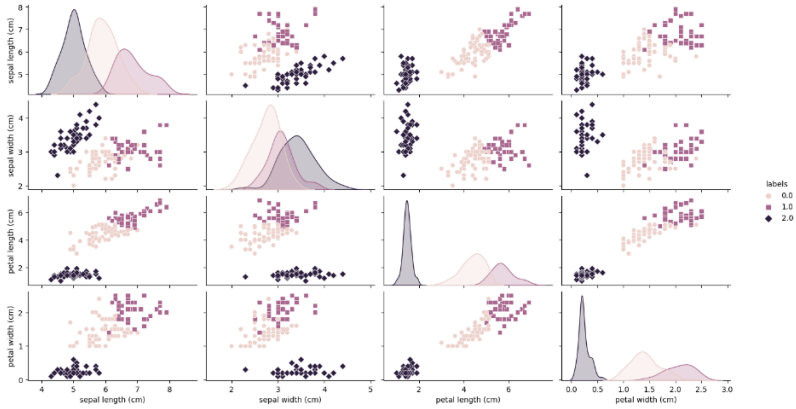
② 두 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

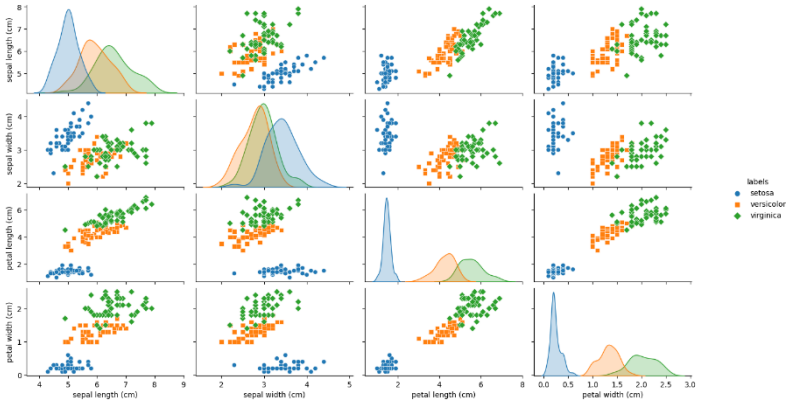


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

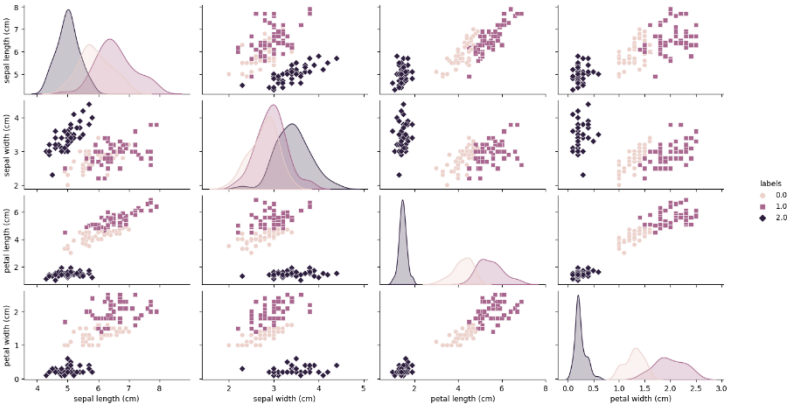
```
pi :            [0.29919315 0.36747352 0.33333333]
count / total : [0.3        0.36666667 0.33333333]
C:\Users\helle\PycharmProjects\MLproject2\venv\lib\
  warnings.warn(
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

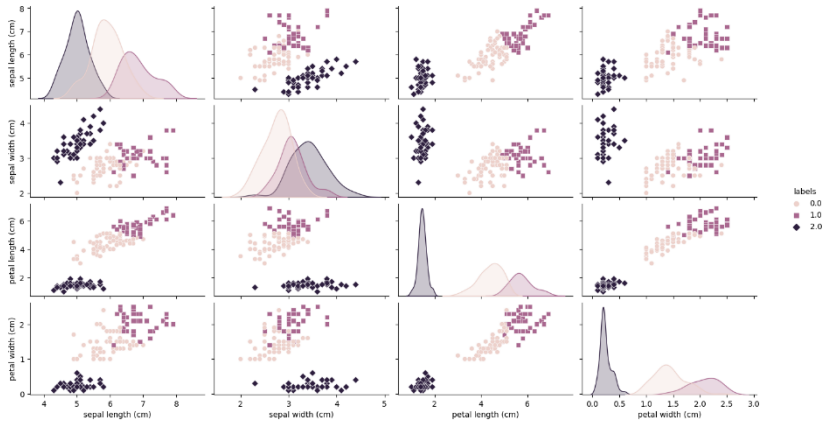
③ 세 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

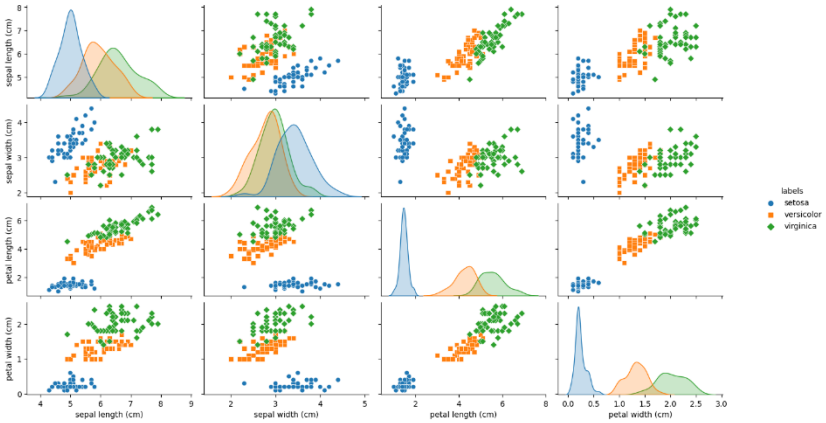


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

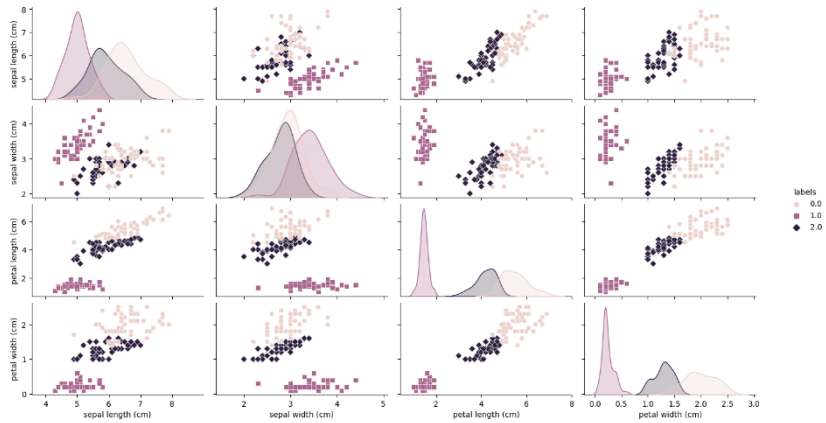
```
pi :            [0.29919325 0.36747341 0.33333333]
count / total : [0.3        0.36666667 0.33333333]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

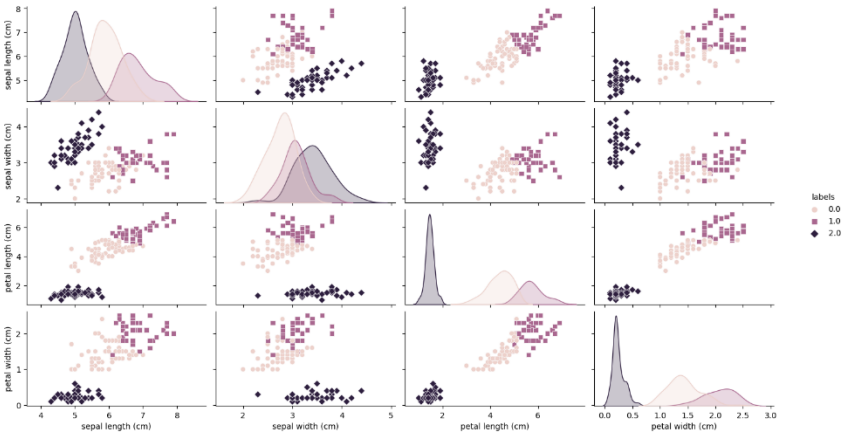
④ 네 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

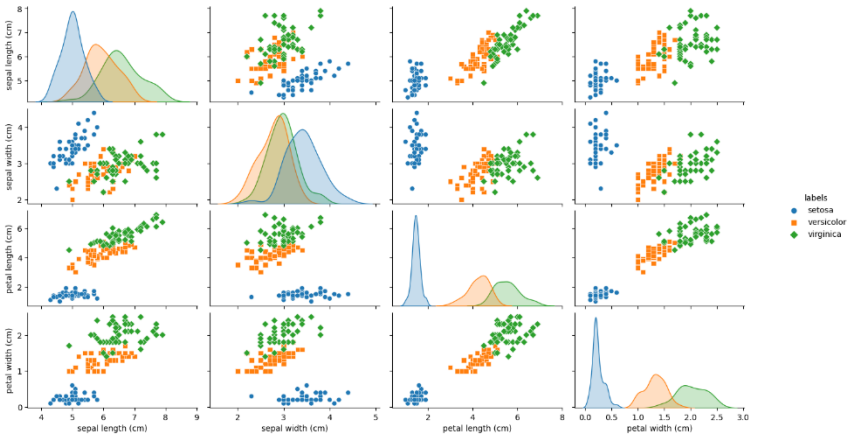


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

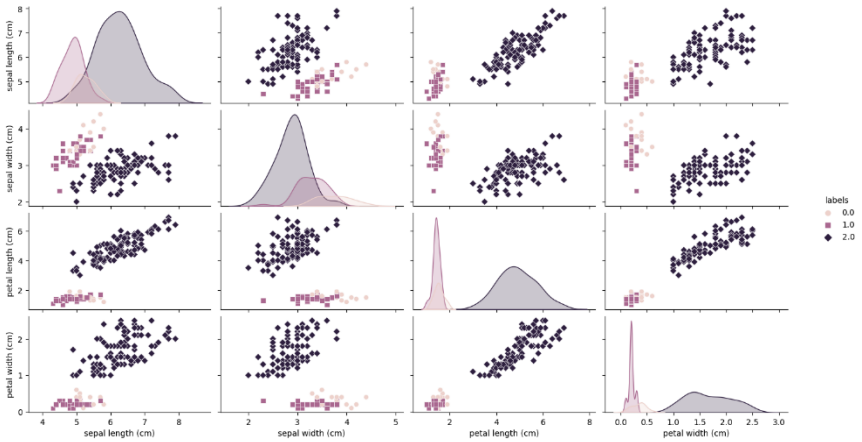
```
pi :            [0.36747353 0.33333333 0.29919314]
count / total : [0.36666667 0.33333333 0.3       ]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

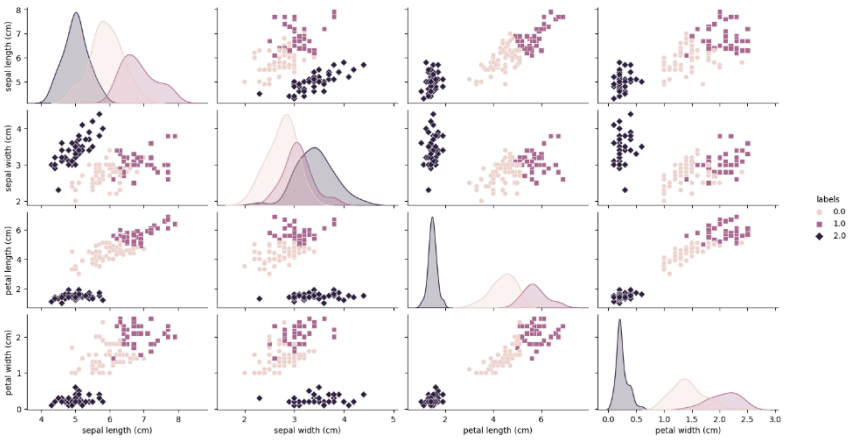
⑤ 다섯 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

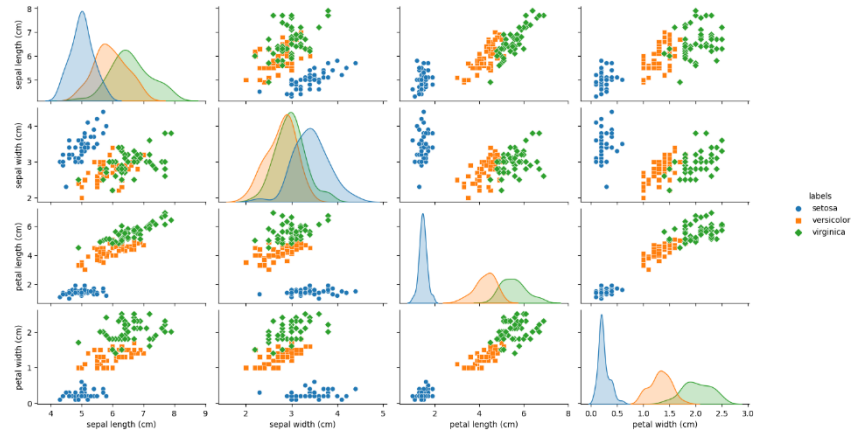


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

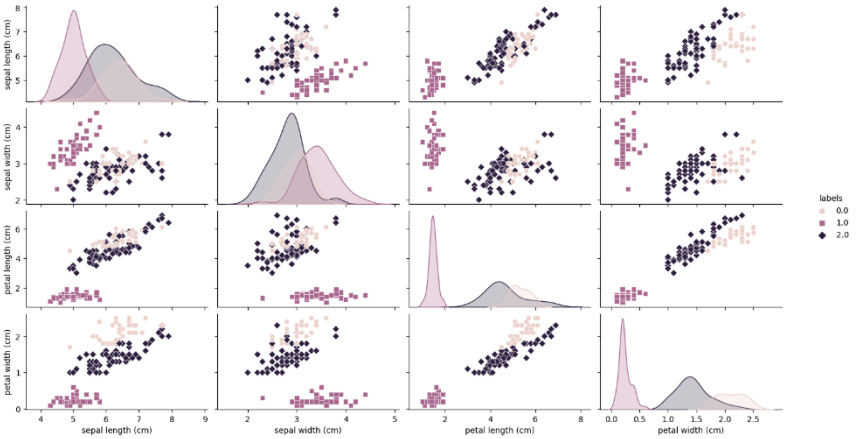
```
pi :            [0.10632189 0.22700494 0.66667317]
count / total : [0.1        0.23333333 0.66666667]
EM Accuracy: 0.57    Hit: 85 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

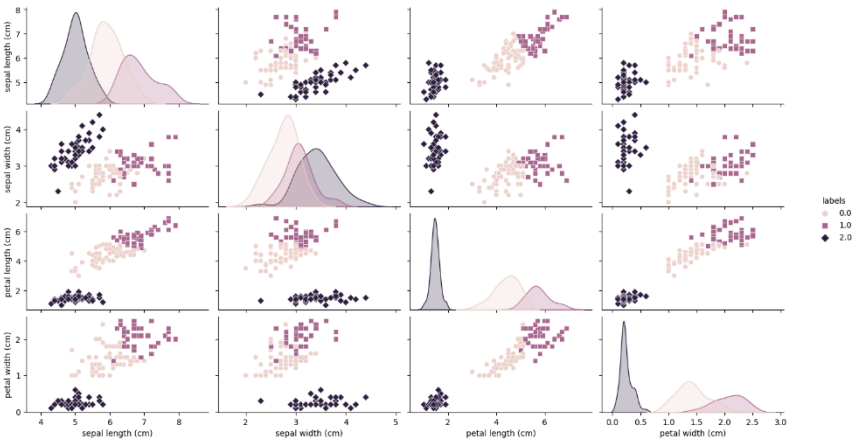
⑥ 여섯 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

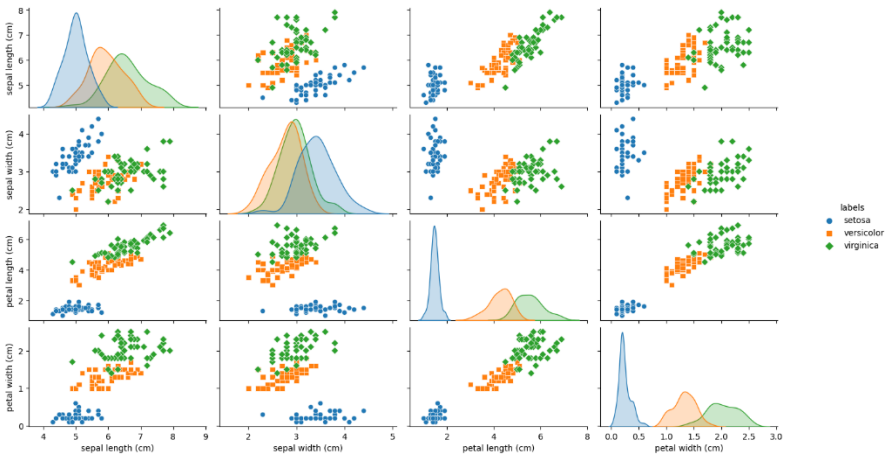


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

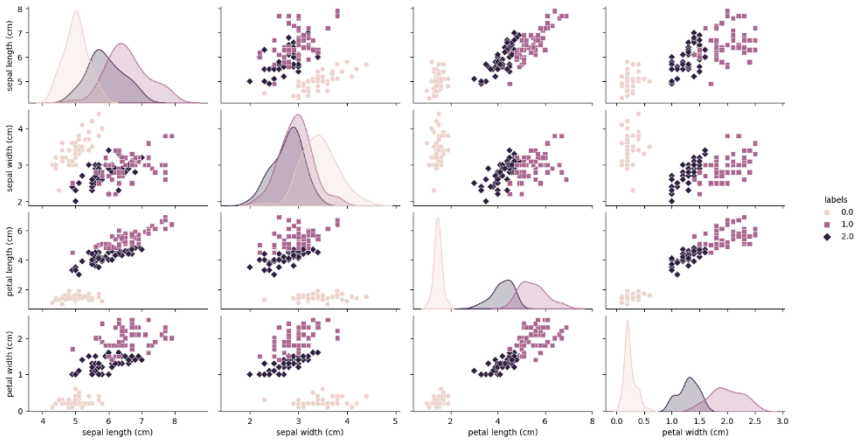
```
pi :            [0.22934268 0.33328802 0.43736929]
count / total : [0.23333333 0.33333333 0.43333333]
EM Accuracy: 0.89     Hit: 133 / 150
KM Accuracy: 0.89     Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

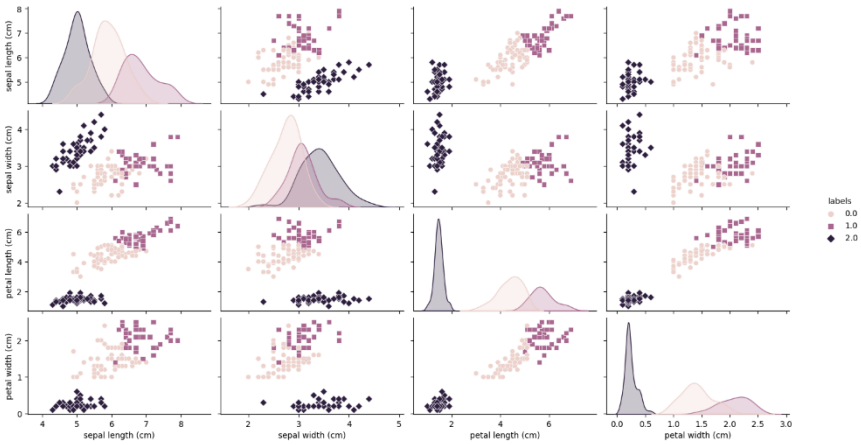
⑦ 일곱 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

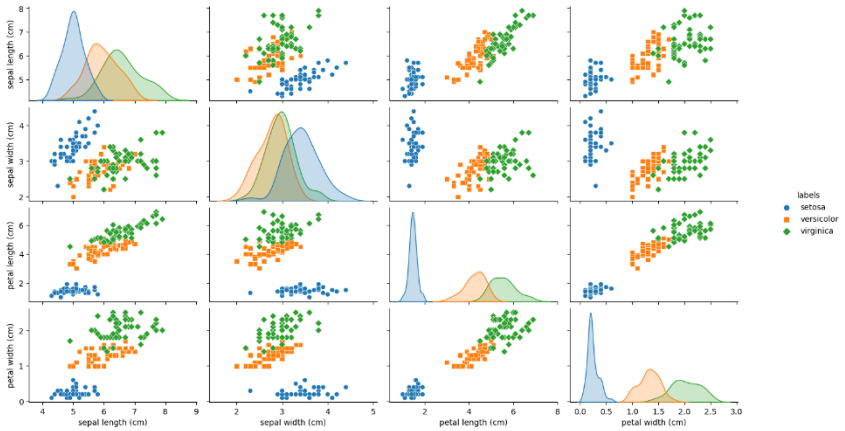


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

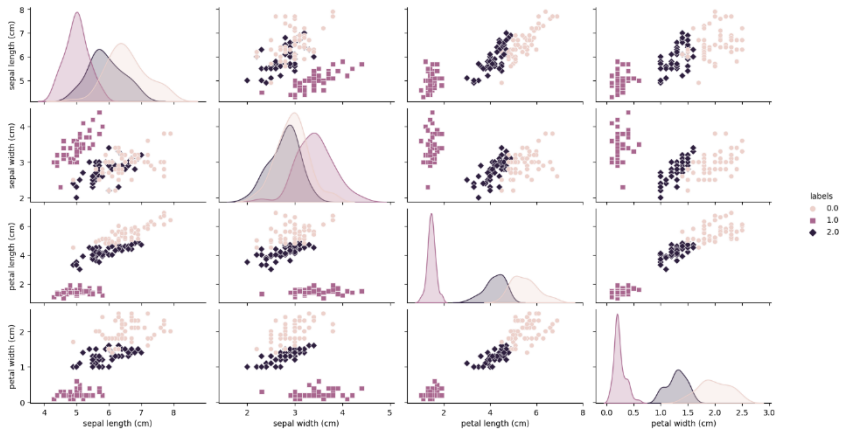
```
pi :            [0.33333333 0.36747344 0.29919323]
count / total : [0.33333333 0.36666667 0.3        ]
C:\Users\helle\PycharmProjects\MLproject2\venv\lib\
  warnings.warn(
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

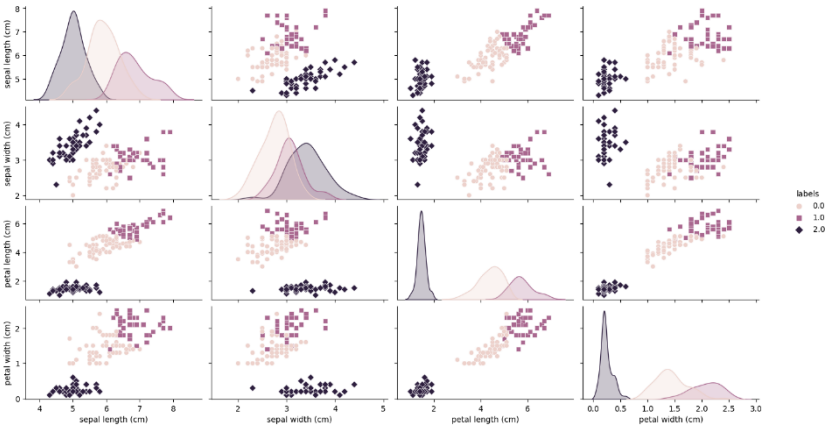
⑧ 여덟 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

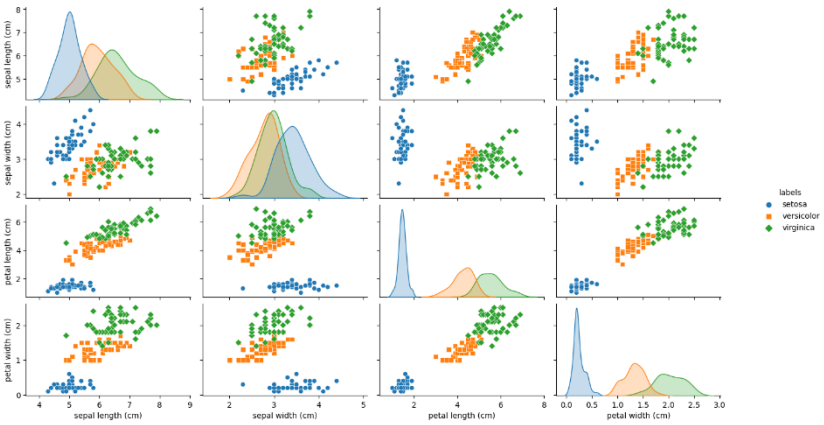


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

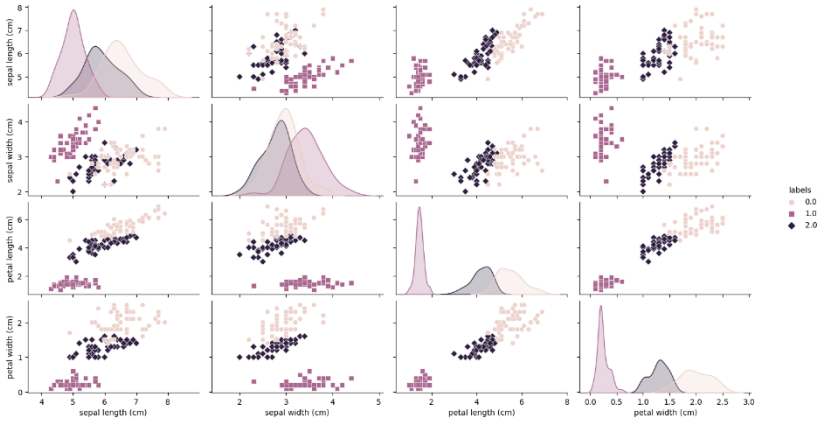
```
pi :              [0.36747342 0.33333333 0.29919325]
count / total : [0.36666667 0.33333333 0.3       ]
EM Accuracy: 0.97     Hit: 145 / 150
KM Accuracy: 0.89     Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

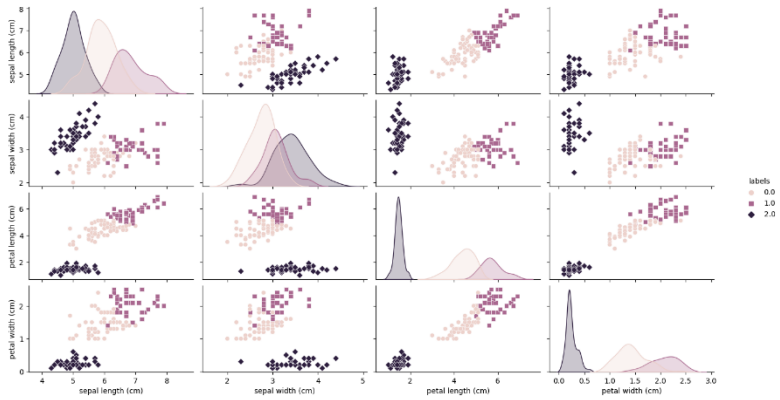
⑨ 아홉 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.

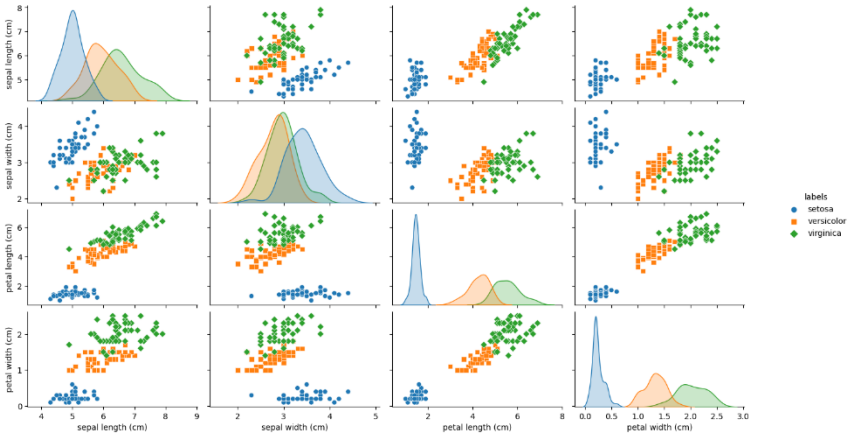


-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

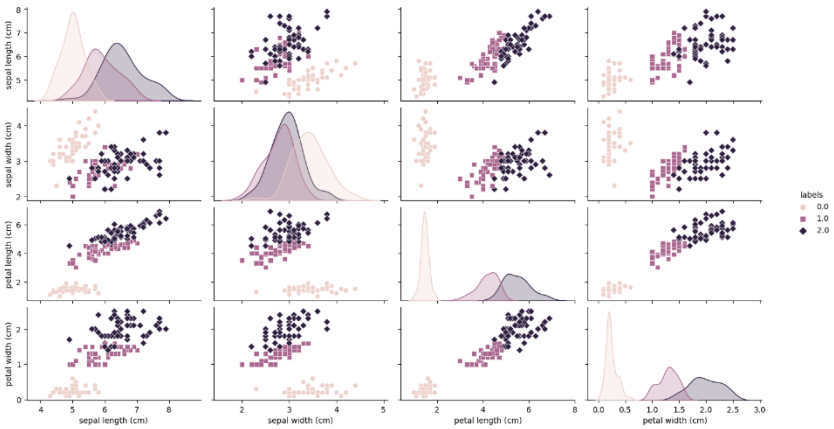
```
pi :            [0.36747344 0.33333333 0.29919323]
count / total : [0.36666667 0.33333333 0.3       ]
C:\Users\helle\PycharmProjects\MLproject2\venv\lib\
  warnings.warn(
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

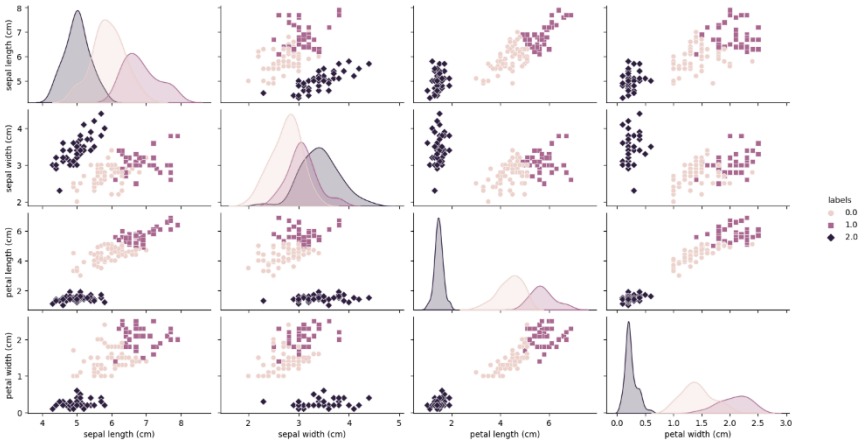
⑩ 열 번째 실행결과



-> 왼쪽의 그림은 기존의 데이터를 가지고 plot 차트를 출력한 결과이다.



-> 왼쪽의 그림은 EM 알고리즘을 사용해 군집화 한 후의 결과이다.



-> 왼쪽의 그림은 KMeans 알고리즘을 적용하여 군집화한 결과이다.

```
pi :            [0.33333333 0.29919324 0.36747342]
count / total : [0.33333333 0.3        0.36666667]
C:\Users\helle\PycharmProjects\MLproject2\venv\lib\
  warnings.warn(
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150

Process finished with exit code 0
```

-> 왼쪽의 결과화면으로, EM과 KMeans의 정확도를 비교할 수 있다.

-> 위의 첨부한 10번의 실행결과를 토대로 EM 알고리즘을 통한 군집화 결과와 KMeans를 통한 군집화 결과를 비교 및 분석하면 다음과 같다. 먼저, 3개의 클래스 분포도가 균일한 경우 EM 알고리즘의 정확도가 KMeans보다 더 높은 것을 확인했다. 반면에, 3개의 클래스 분포도의 격차가 많이 나는 경우에는 EM 알고리즘 정확도가 많이 떨어졌고 KMeans가 상대적으로 더 높은 것을 확인할 수 있었다.

-> EM 알고리즘은 군집의 형태를 더욱 정확하게 파악할 수 있으며, 군집의 크기나 밀도가 서로 다른 경우에도 유연하게 대응할 수 있다. 그러나 EM 알고리즘은 초기값에 민감하게 반응하며, 수렴이 느리고 계산 비용이 많이 드는 단점이 있다. 반면에 K-means 알고리즘은 계산 비용이 적고, 수렴이 빠르며, 초기값에 대한 민감도가 낮은 장점이 있지만, 군집의 크기나 밀도가 서로 다른 경우에는 군집화 결과가 좋지 않을 수 있다는 단점이 존재한다.

-> 아래에 출력문의 값이 거의 동일한 이유는 EM_model.pi 값은 전체에서 각 클래스가 나타날 확률(또는 빈도수)을 말하는 것이고 np.bincount() 함수를 사용해서 EM_pred 값을 계산하면 예측한 클래스가 전체에서 얼만큼 나왔는지의 빈도수를 알 수 있으므로 해당 값을 150으로 나누면 각 클래스의 확률 값을 구한

것과 동일하기 때문이다.

```
# Why are these two elements almost the same? additional 10 points
print(f'pi :            {EM_model.pi}')
print(f'count / total : {np.bincount(EM_pred) / 150}')
```

IV. 참고문헌

https://medium.com/@prateek.shubham.94/expectation-maximization-algorithm-7a4d1b65ca55

다변량 가우시안 분포(Multivariate Gaussian Distribution) (tistory.com)