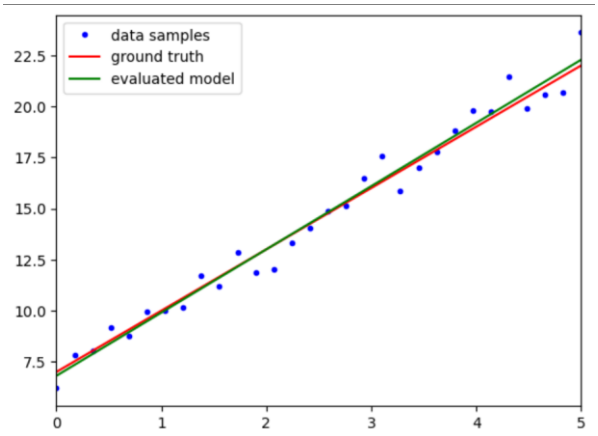
EE214 mini assignment1

20200045 김건우

Part1. Bias and Variance

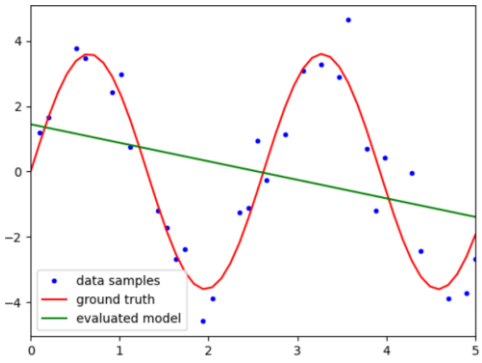
Step 1 : Linear Regression

y=3x+7+∈(noise)의 관계를 가진 데이터 샘플들을 f(x,w)=w0+w1x로 추론했다. LinearRegression 모델을 작동시켜본 결과 w0 = 3.094, w1 = 6.811이 나왔다.



Step 2-1 Polynomial Regression

Step1과는 달리 step2에서는 y=3.6sin(2.4x)+∈ 의 관계(비선형)를 가진 데이터 샘플을 이용했다.



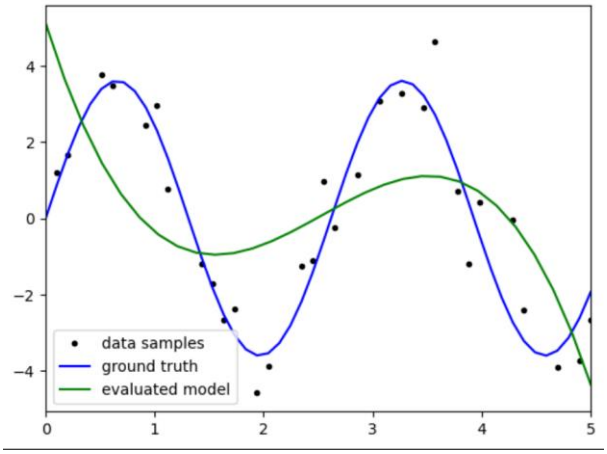
Step 1과 같은 모델로 추론한 결과의 그래프를 보면, 제대로 된 결과가 아님을 알 수 있다.

Q. Is it good choice to use linear regression or not? Write the answer and reason in your report.

선형회귀분석을 이용하는 것은 좋은 선택이 아니다. 변수 x,y가 비선형 관계를 가지고 있기 때문이다.

따라서 모델의 차수를 높여 Polynomial Regression을 시도해 보았다.

소스코드에서 정의한 fit_polynomial 함수는 x,y,degree를 input으로 받아 해당 차수(degree)에 맞게 x를 변환한다. 이후 LinearRegression과 fit 함수를 이용해 변환한 x와 y로 fitting한 모델을 반환한다. evaluate_polynomial 함수는 model, x를 input으로 받아 예상되는 y값을 반환하는 함수이다. 이 때, input의 model은 polynomial 형태여야 한다. fit_polynomial 함수와 마찬가지로 x를 degree에 맞게 변환하는 과정이 필요하며 degree는 model의 계수의 총 수-1로 구한다.

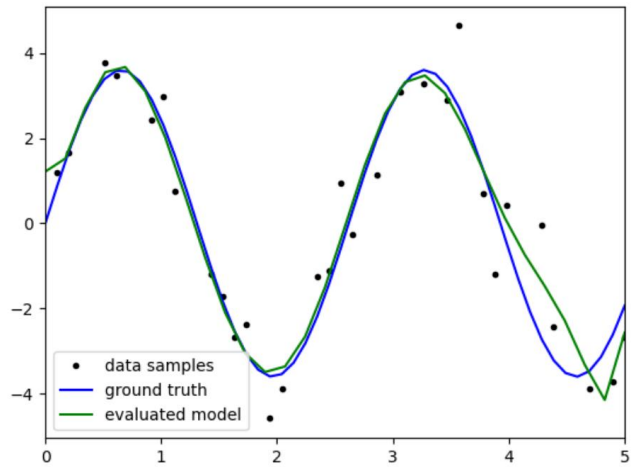


위에서 말한 두 함수 fit_polynomial과 evaluate_polynomial 함수를 이용하여 degree가 3인 evaluated model을 그래프로 나타낸 것이다.

Q. Does the above trained curve (evaluated model) seem to fit the training samples well? If not, provide an explanation as to why this curve is not that satisfactory.

위의 evaluated model은 가장 처음에 시도했던 모델보다는 향상되었지만 아직도 sample에 잘 맞지 않는 것으로 보인다. 3차로는 sin함수를 나타내기 힘들기 때문으로 추측된다. 따라서 차수를 증가시키면 더 좋은 모델을 얻을 수 있을 것으로 기대된다.

TODO : Repeat the previous process by selecting a proper value of "degree" parameter so that the obtained fitted curve is reasonably close to the ground truth, and attach the results in your report.

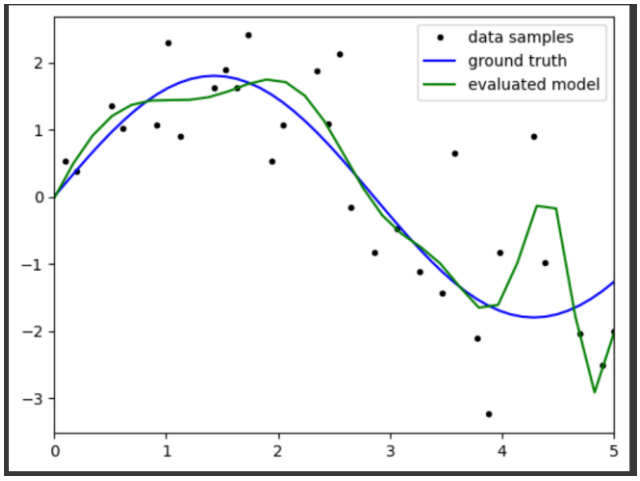


Degree를 10으로 증가시켰을 때의 그래프이다. 이전보다 훨씬 ground truth에 가까워졌음을 알 수 있다.

Step 2-2 : Regression with Gaussian basis

이번 단계에서는 다항함수가 아닌 gaussian함수를 basis로 사용해 모델을 만들 것이다. 소스코드에서 새롭게 정의한 fit_gaussian 함수의 경우 x,y,degree를 input으로 받아 해당 차수(degree)에 맞게 x를 gaussian form으로 변환한다. degree는 다항함수에서는 차수의 의미로 쓰였지만 여기서는 basis로 활용되는 gaussian 함수의 개수로 생각하면 된다. 이후 LinearRegression과 fit 함수를 이용해 변환한 x와 y로 fitting한 모델을 반환한다. evaluate_gaussian 함수는 model, x를 input으로 받아 예상되는 y값을 반환하는 함수이다. 이 때, input의 model은 gaussian 형태여야 한다. fit_polynomial 함수와 마찬가지로 x를 degree에 맞게 gaussian form으로 변환하는 과정이 필요하며 degree는 model의 계수의 총 수-1로 구한다.

TODO : Attach the following result in your report



위에서 말한 두 함수 fit_gaussian과 evaluate_gaussian 함수를 이용하여 degree가 10인 evaluated model을 그래프로 나타낸 것이다.

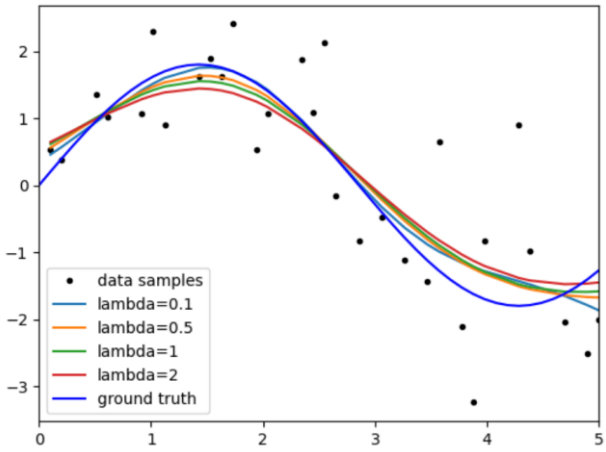
Q. Do you think this curve has a good ability to generalize on unseen test samples? If not, provide an explanation that supports your thoughts.

주어진 샘플이 어떠한 분포인지 모르는 상황에서 우리는 일반적으로 샘플이 정규분포를 따른다고 추론할 수 있다. 따라서 나는 이 곡선이 unseen test sample에 대해 좋은 능력을 가지고 있다고 생각한다.

Step 3. Linear Regression with Regularization

Fit_gaussian_regressor 함수는 fit_gaussian 함수와 비슷하지만 lambda_와 regressor를 추가적으로 input으로 받는다. 해당 파라미터는 어떤 regressor를 사용할지를 결정한다.

TODO : In the following, choose a proper value of "lambda_" parameter so that the obtained fitted curve is reasonably close to the ground truth, and attach the results in your report. (Do not modify the 'degree' parameter)

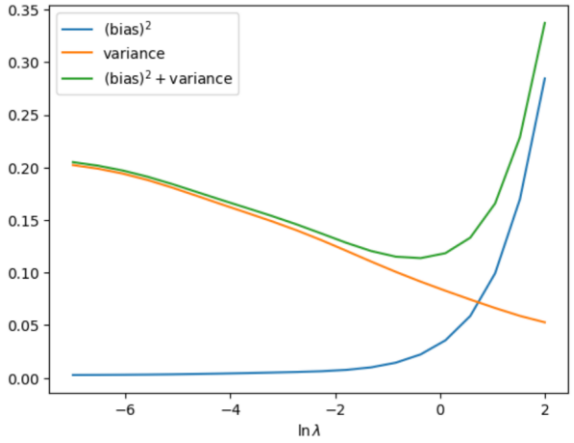


Fit_gaussian_regressor 함수와 evaluate_gaussian 함수를 이용해 degree는 10으로 고정시켜둔 채, lambda값이 0.1, 0.5, 1, 2,일 때의 그래프를 그려보았다. 그 결과 Lambda가 0.1일 때 가장 ground truth에 가까운 결과를 얻을 수 있었다.

Step 4: Compute integrated squared bias and integrated variance

문제의 조건에 맞게 30개의 데이터셋으로 구성된 모델 100개에 대해 y값을 예측하고(y_pred), 각각의 y_pred값과 ground thruth를 통해 avg_y, $bias^2$, variance를 계산해주었다. 이 과정을 lambda 값을 바꾸어주면서 반복하고 squared_bias array와 variance array에 각각 $bias^2$, variance를 추가해주었다. 최종적으로는 이를 그래프로 나타내었다.

```python
# Plot squared bias, variance and their sum according to the labmda.
loss = 1 #initialize
for lambda_ in lambdas:
    #---------------- Blank ----------------------------------------
    avg_y = np.zeros(sample_size)
    models = []
    for i in range(n_models):

        x, y, _ = nonlinear_sample(sample_size, interval, ground_truth_gauss, noise=1., seed=i)
        model = fit_gaussian_reg(x, y, degree, lambda_, Ridge)

        p_y = evaluate_gaussian(model, x_test)
        avg_y = avg_y + p_y
        models.append(p_y)

    avg_y = avg_y / n_models  #n_models =L(100) samplesize= N(30)

    ###################### compute Bias^2  ######################
    bias_val=0
    bias_val = np.sum((avg_y-ground_truth)**2)/sample_size
    squared_bias.append(bias_val)
    ##############################################################

    ###################### compute variance  ####################
    var_val = 0
    for p_y in models: #100번
        var_val += np.sum((p_y-avg_y)**2)/sample_size
    variance.append(var_val / n_models)
    ##############################################################
    #find optimal lamda value
    if loss > squared_bias[-1] + variance[-1]:
        loss = squared_bias[-1] + variance[-1]
        print('lamda val =', lambda_)
    #---------------------------------------------------------------
```



Q. Write the relationship between lambda and bias, variance.

람다가 크다는 건 페널티를 크게 하는, 즉 underfitting을 의미하고 람다가 작다는 건 overfitting을 의미한다. 즉, 람다가 커질수록 variance는 감소하고 bias값은 커지게 된다. 그래프에서도 그러한 경향성을 파악할 수 있다.

Q. Find the best lambda value through the resulting plot and explain about it.

위의 내용대로 람다가 커질수록 variance는 감소하고 bias값은 커지므로 bias²+variance가 최소가 되는 람다가 존재하게 된다. loss값이 더 작게 나오면 해당 loss값을 업데이트하고 그 때의 lambda value를 출력하는 방식으로 코드를 구성하였고, 그 결과 best lambda value는 0.691이었다.

```
lamda val = 0.0009118819655545166
lamda val = 0.0014643910485082185
lamda val = 0.0023516652636582877
lamda val = 0.0037765387311884816
lamda val = 0.006064742720220372
lamda val = 0.009739369004403387
lamda val = 0.015640450548327085
lamda val = 0.025116996105606543
lamda val = 0.040335378537834525
lamda val = 0.0647457554835706
lamda val = 0.1040214766680376
lamda val = 0.1670480665692844
lamda val = 0.26826245346996097
lamda val = 0.43080261519743523
lamda val = 0.691258252705166
```

Step 5: Repeat step 4 with own implemented Ridge regressor (My_Ridge class) using pure Numpy.

```
[24]  1 class My_Ridge:
      2     def __init__(self, alpha):
      3         self.lambda_ = alpha
      4
      5     def fit(self, X, y, lr=0.01, n_iters=1000):
      6         _, coef_size = X.shape
      7         self.coef_ = np.random.randn(coef_size)  # initialize the weights
      8         #print(self.coef_)
      9         #print(X)
     10         #print(np.sum(X,axis=0)+self.coef_)
     11         #print(y - np.dot(X,self.coef_))
     12     #------------------ Blank ----------------------------------------
     13         #X_t = X.transpose()
     14         for i in range(n_iters):
     15             diff = y - np.dot(X,self.coef_)
     16             loss = np.sum(diff**2)/(2)+ self.lambda_*np.linalg.norm(self.coef_, 2)/2
     17             grad = -diff@X + self.lambda_*self.coef_
     18
     19             self.coef_ = self.coef_ - lr * grad
     20     #-----------------------------------------------------------------
     21
     22     def predict(self, X):
     23         return np.dot(X, self.coef_) #내적
```

Step 5에서는 My_Ridge라는 class를 직접 구현해서 이를 regressor로 활용했다. My_Ridge class는 fit 함수와 predict함수를 실행할 수 있다. fit 함수는 loss함수의 gradiant를 이용해 loss가 최소가 될 수 있게 coef를 업데이트한다(총 n_iters만큼 반복). predict함수는 coef와 X를 내적한 값을 반환한다.

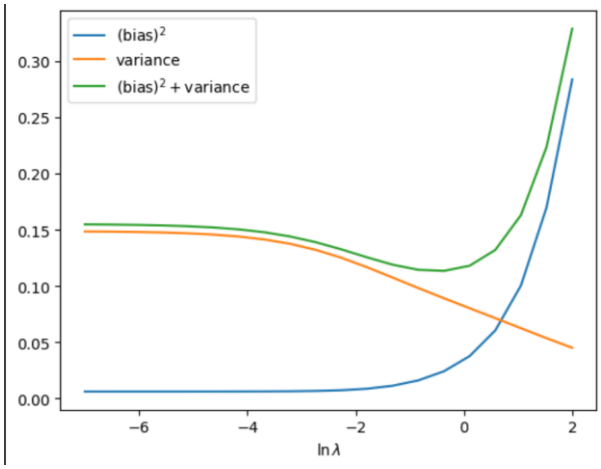
TODO : Repeat step 4 using the implemented My_Ridge and write your own description of the source code in your report.

아래 코드에서는 step4와 모두 동일하지만 model을 만들어줄 때 regressor로 My_Ridge를 써주었다.

Q. Find the best lambda value through the resulting plot and explain about it.

그래프 결과와 최적의 lambda값 모두 step4와 동일하게 나왔다.

```
15 # Plot squared bias, variance and their sum according to the lambda.
16 loss = 1 #initiallize
17 for lambda_ in lambdas:
18     #---------------- Blank ----------------------------------------
19     avg_y = np.zeros(sample_size)
20     models = []
21     for i in range(n_models):
22
23         x, y, _ = nonlinear_sample(sample_size, interval, ground_truth_gauss, noise=1., seed=i)
24         model = fit_gaussian_reg(x, y, degree, lambda_, My_Ridge)
25
26         p_y = evaluate_gaussian(model, x_test)
27         avg_y = avg_y + p_y
28         models.append(p_y)
29
30     avg_y = avg_y / n_models  #n_models =L(100) samplesize= N(30)
31
32     ###################### compute Bias^2  ######################
33     bias_val=0
34     bias_val = np.sum((avg_y-ground_truth)**2)/sample_size
35     squared_bias.append(bias_val)
36     ##############################################################
37
38     ###################### compute variance  ####################
39     var_val = 0
40     for p_y in models: #100번
41         var_val += np.sum((p_y-avg_y)**2)/sample_size
42     variance.append(var_val / n_models)
43     ##############################################################
44     #find optimal lamda value
45     if loss > squared_bias[-1] + variance[-1]:
46         loss = squared_bias[-1] + variance[-1]
47         print('lamda val =', lambda_)
48     #-----------------------------------------------------------------
49
```



```
lamda val = 0.0009118819655545166
lamda val = 0.001464391048508218 5
lamda val = 0.0023516652636582877
lamda val = 0.0037765387311884816
lamda val = 0.006064742720220372
lamda val = 0.009739369004403387
lamda val = 0.015640450548327085
lamda val = 0.025116996105606543
lamda val = 0.040335378537834525
lamda val = 0.06477457554835706
lamda val = 0.1040214766680376
lamda val = 0.1670480665692844
lamda val = 0.26826245346996097
lamda val = 0.43080261519743523
lamda val = 0.6918258252705166
```

Part2. PCA

Step 1. Load Iris dataset

Iris dataset을 불러와 dataframe을 만들어주었다.

Step 2. Using numpy to implement PCA (with Iris dataset)

(2-1)가장 먼저 데이터를 정규화해주었다. 주어진 식(Z-Score Normalization)을 이용하여 정규화하였다.

(2-2)그 다음 마찬가지로 주어진 식을 이용해 covariance martrix를 정의하고, (2-3)np.linalg.eig함수를 이용해 covariance martrix의 eigen_value와 eigen_vector를 구해주었다.

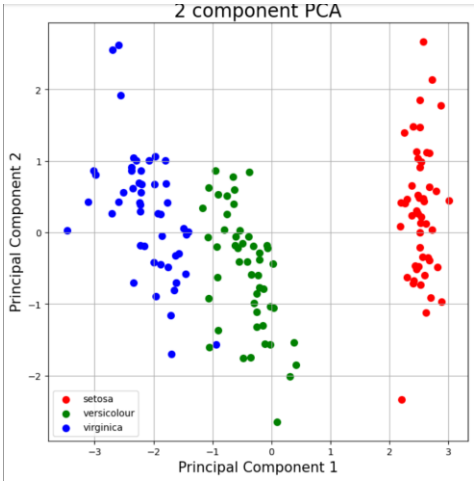
(2-4)다음으로는 PCA에 이용할 2개의 eigen_vector를 추출하였다. eigen_value가 가장 큰 두 개의 eigen_vector이다.

(2-5)각 principal component이 전체 variance의 몇 퍼센트를 나타내는지 확인한다. 첫 번째 PC는 약76%, 두 번째 PC는 약 18%이다.

PCA를 마친 데이터셋을 이용해 Iris dataset을 분류한 결과는 아래와 같다.

```
1 #Step5 : get variance ratio
2 variance_ratio = eig_vals[:2]/eig_vals.sum()
3 print('first 2 eigenvalues:', eig_vals[:2])
4 print('variance_ratio:', variance_ratio)
```

```
first 2 eigenvalues: [3.8370179  0.91413636]
variance_ratio: [0.76740358 0.18282727]
```

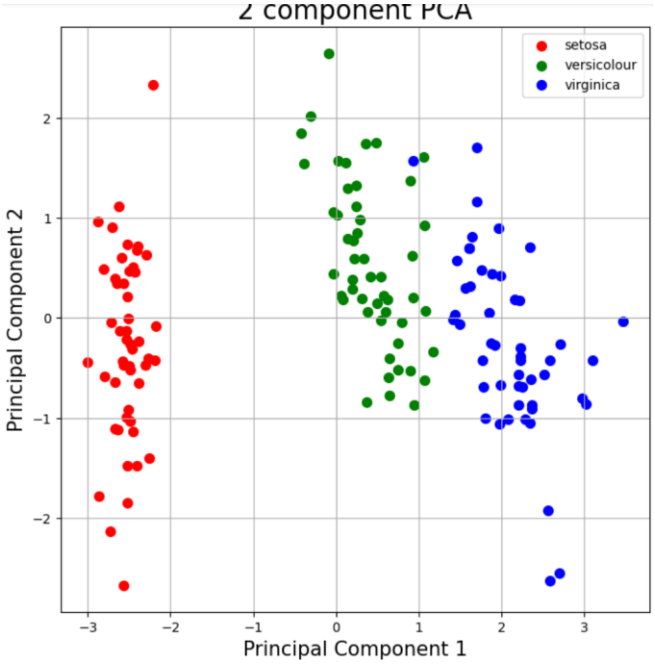


Step 2-6: Get eigenvectors and eigenvalues without using the np.linalg.eig() method.

이번엔 eigen vector를 np.linalg.eig 함수 없이 구해본다. Power method를 이용하면 eigen value가 가장 큰 eigen vector를 구할 수 있다. 해당 메서드의 핵심 작동 원리는 1과 1보다 작은 실수를 거듭제곱할 경우 1은 계속 1이지만 1보다 작은 실수는 0으로 수렴한다는 사실이다. 이를 통해 eigen value(의 절댓값)가 가장 큰 eigen vector를 구한다.

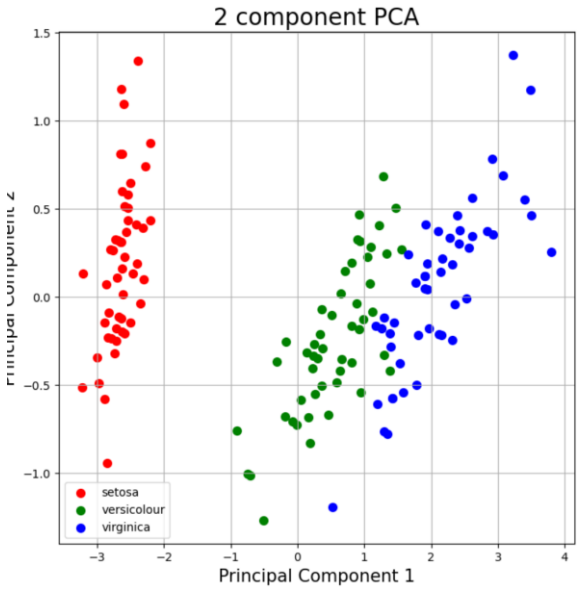
PCA를 위해서는 2개의 PC가 필요하므로 두 번째 eigen vector를 구해주어야 한다. 초기 covariance_matrix에서 첫 번째 eigen vector를 기저로 하는 성분을 빼서 두 번째로 큰 eigen value가 가장 큰 eigen value가 되는 새로운 matrix를 만들어주었다(covar_matrix_2). covar_matrix_2에 대해 다시 한 번 Power method를 적용시켜 두 번째 eigen vector를 구한다.

아래는 이렇게 구한 eigen vector를 통해 PCA를 적용시킨 후 Iris data를 분류한 결과이다.



Step 3. Using sklearn PCA package (with Iris dataset)

마지막은 sklearn 의 PCA package를 이용하는 방법이다. (3-1)StandardScaler를 이용해 정규화해준다. (3-2)다음으로 sklearn.decomposition.PCA module을 이용해 데이터를 변환시켜준다. (3-3)마지막으로 Iris data를 분류한 결과는 아래와 같다.



PCA를 이용해서 구한 eigen_value와 principal component이 전체 variance의 몇 퍼센트를 나타내는지 확인한다. 첫 번째 PC는 약92%, 두 번째 PC는 약 5%이다.

```
#Step3 : Get eigenvlaue and variance ratio of covariance matrix

print('eigen_value :', pca.explained_variance_)
print('explained variance ratio:', pca.explained_variance_ratio_)
```

```
eigen_value : [4.22824171 0.24267075]
explained variance ratio: [0.92461872 0.05306648]
```