

빅데이터 분석 특론

Python Scikit-learn 실습

Contents

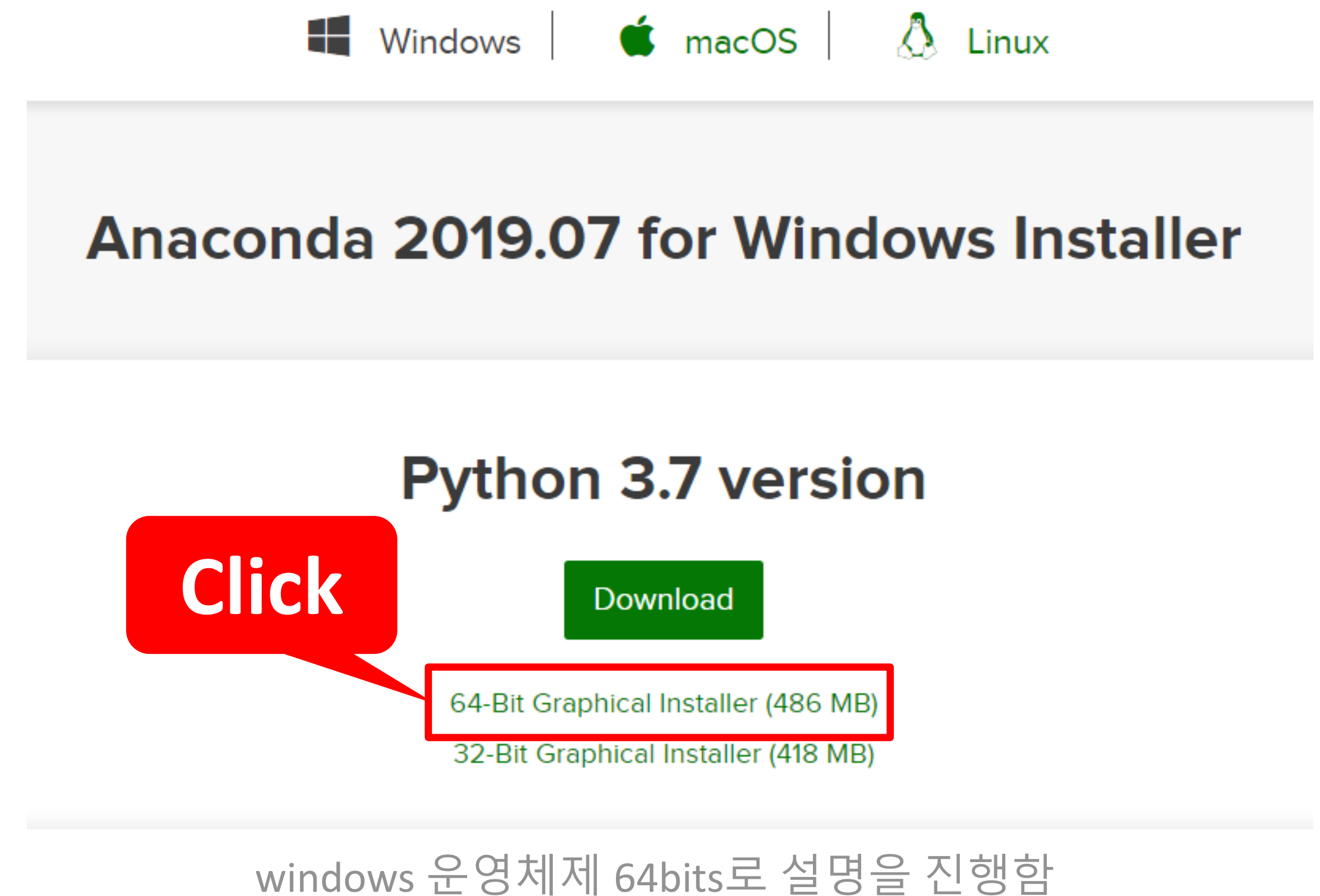
1. Anaconda 설치 및 실행
2. 패키지 설치
3. Numpy 사용법
4. Data Preprocessing
5. Regression
 - Least Squares
 - Ridge
 - Lasso

Anaconda 설치 및 실행

Anaconda 설치

➤ 설치 방법

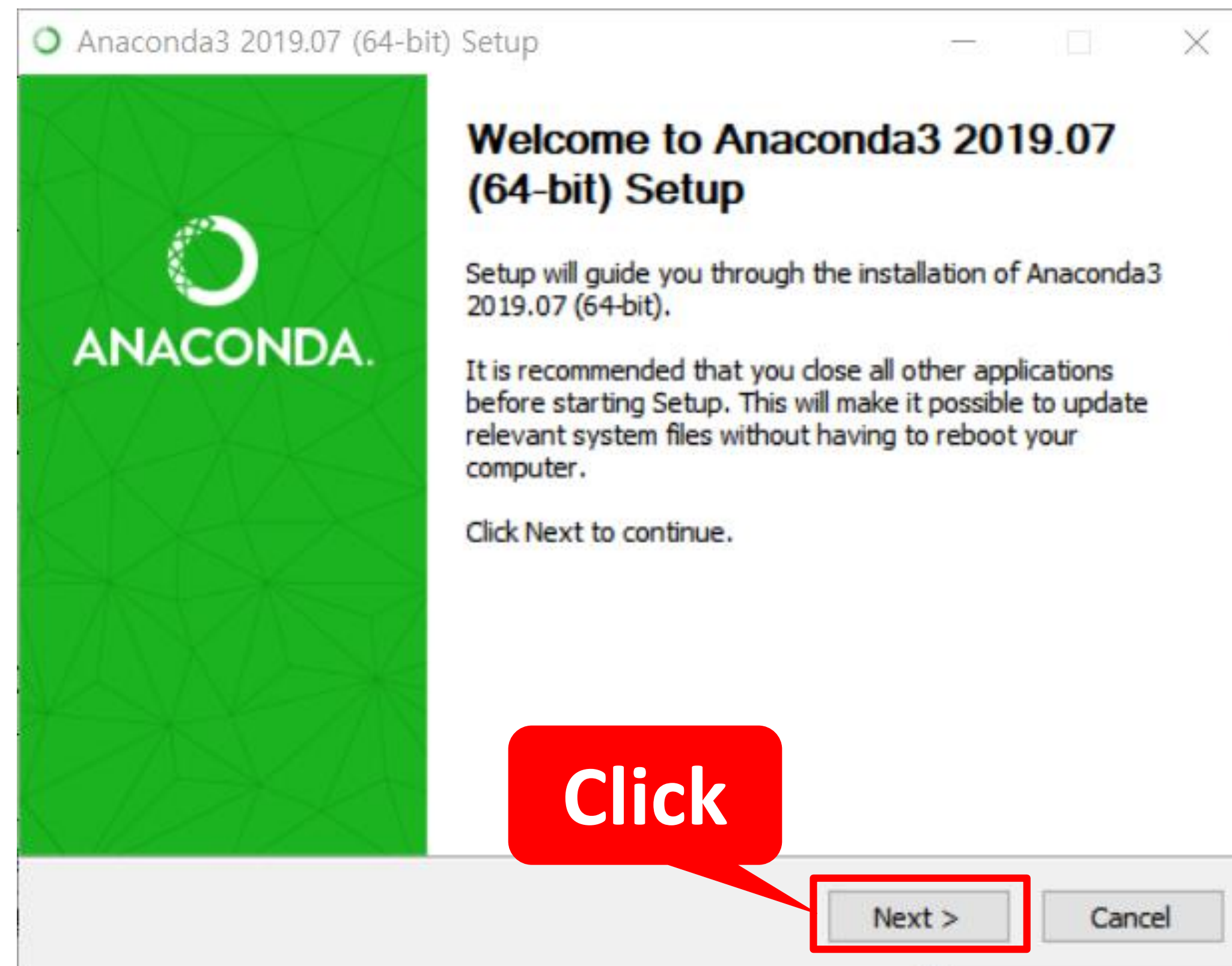
- www.anaconda.com/download/ 접속
- *Python 3.7 설치*
- *자신의 컴퓨터 사양에 맞는 것을
선택하여 설치*



Anaconda 설치

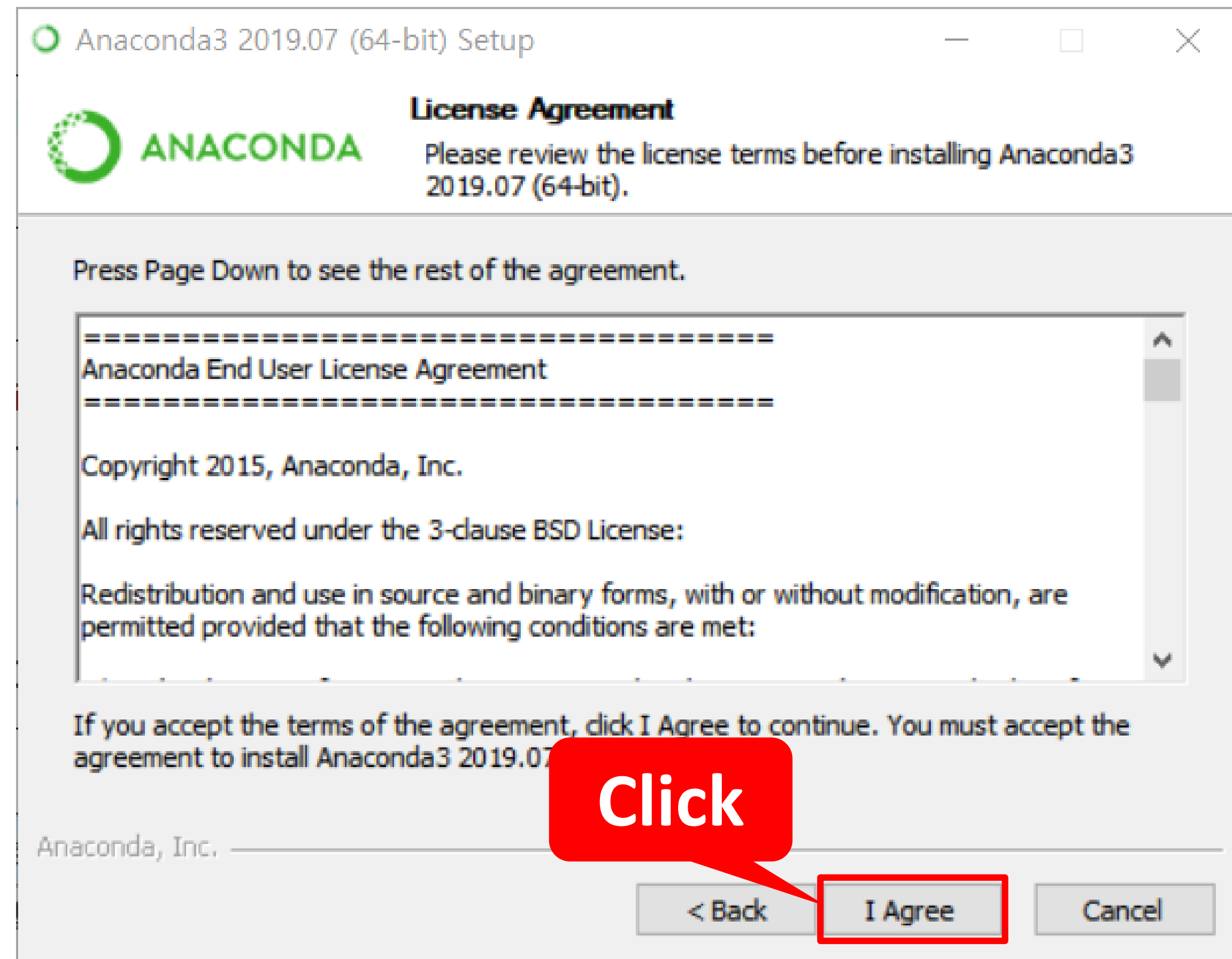
➤ 설치 방법

- 설치파일 실행시 아래와 같은 화면이 나타남



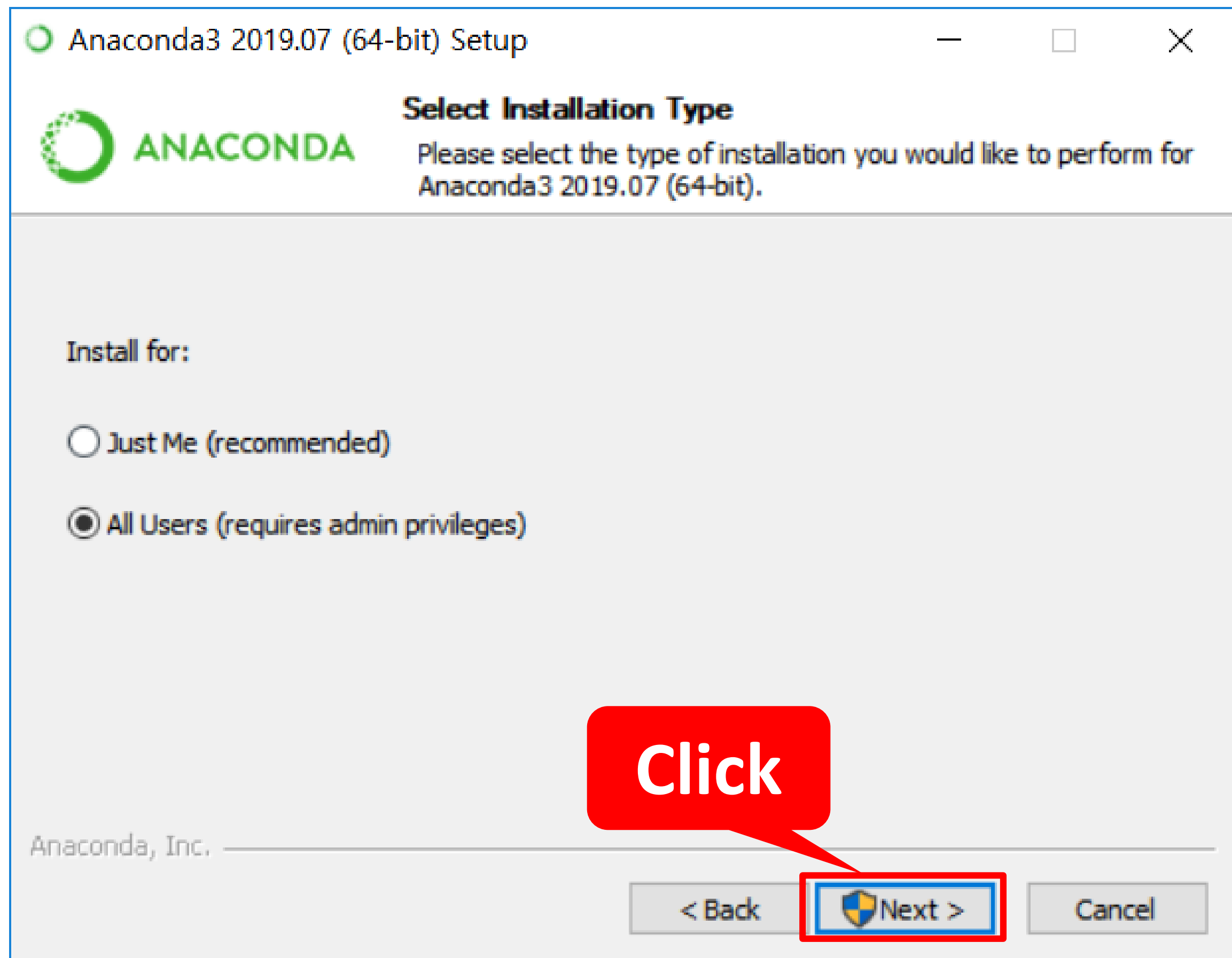
Anaconda 설치

➤ 설치 방법



Anaconda 설치

➤ 설치 방법



■ Just Me

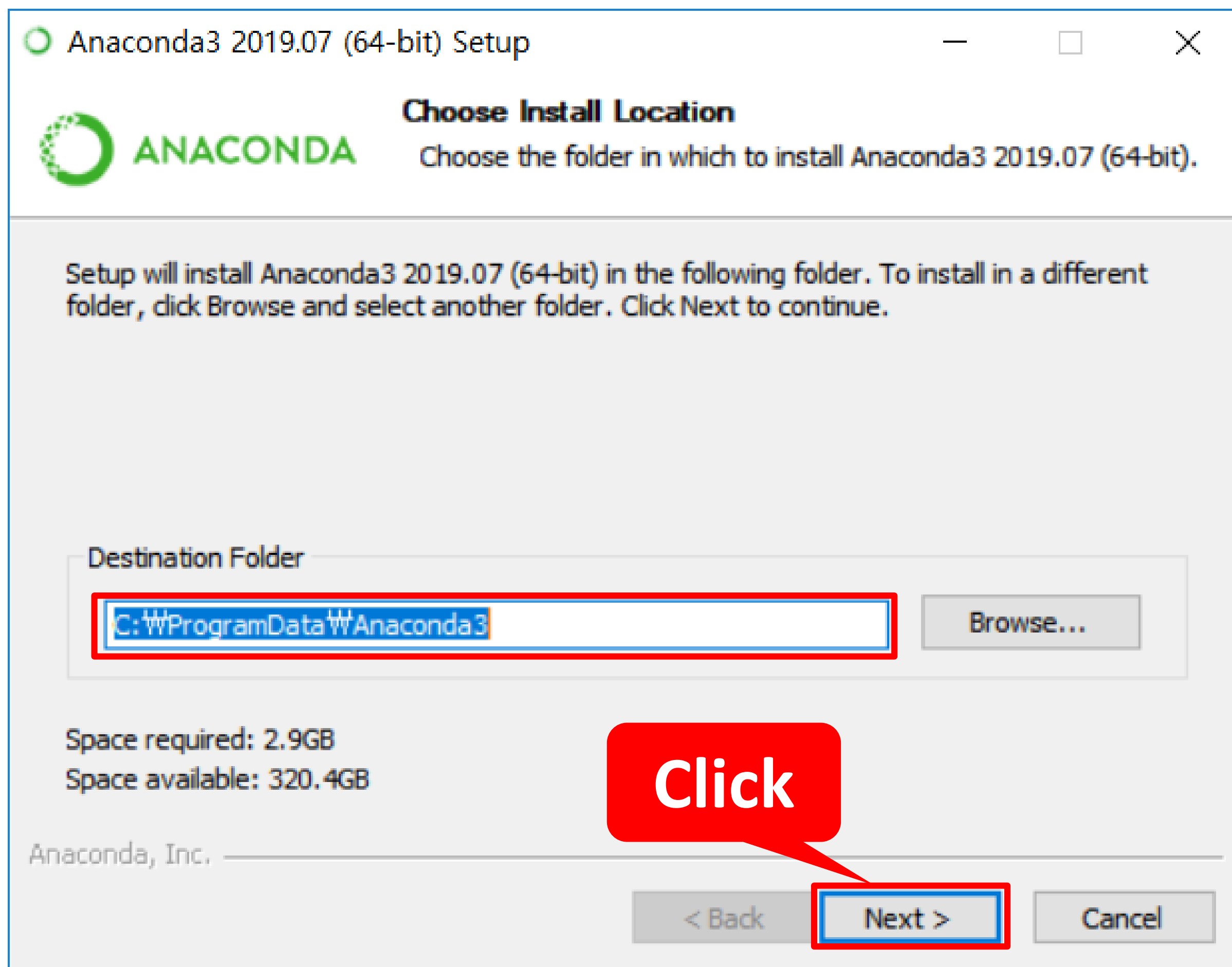
- 개인만 사용하는 컴퓨터가 아니라 여러 명이 사용하는 컴퓨터의 경우, 자신이 권한을 갖는 경로에 anaconda를 설치해야하므로 just me를 선택

■ All Users

- 모든 계정에 anaconda를 설치하기를 원하거나 개인만 사용하는 컴퓨터라면 All Users를 체크해도 무방함

Anaconda 설치

➤ 설치 방법



■ Just Me

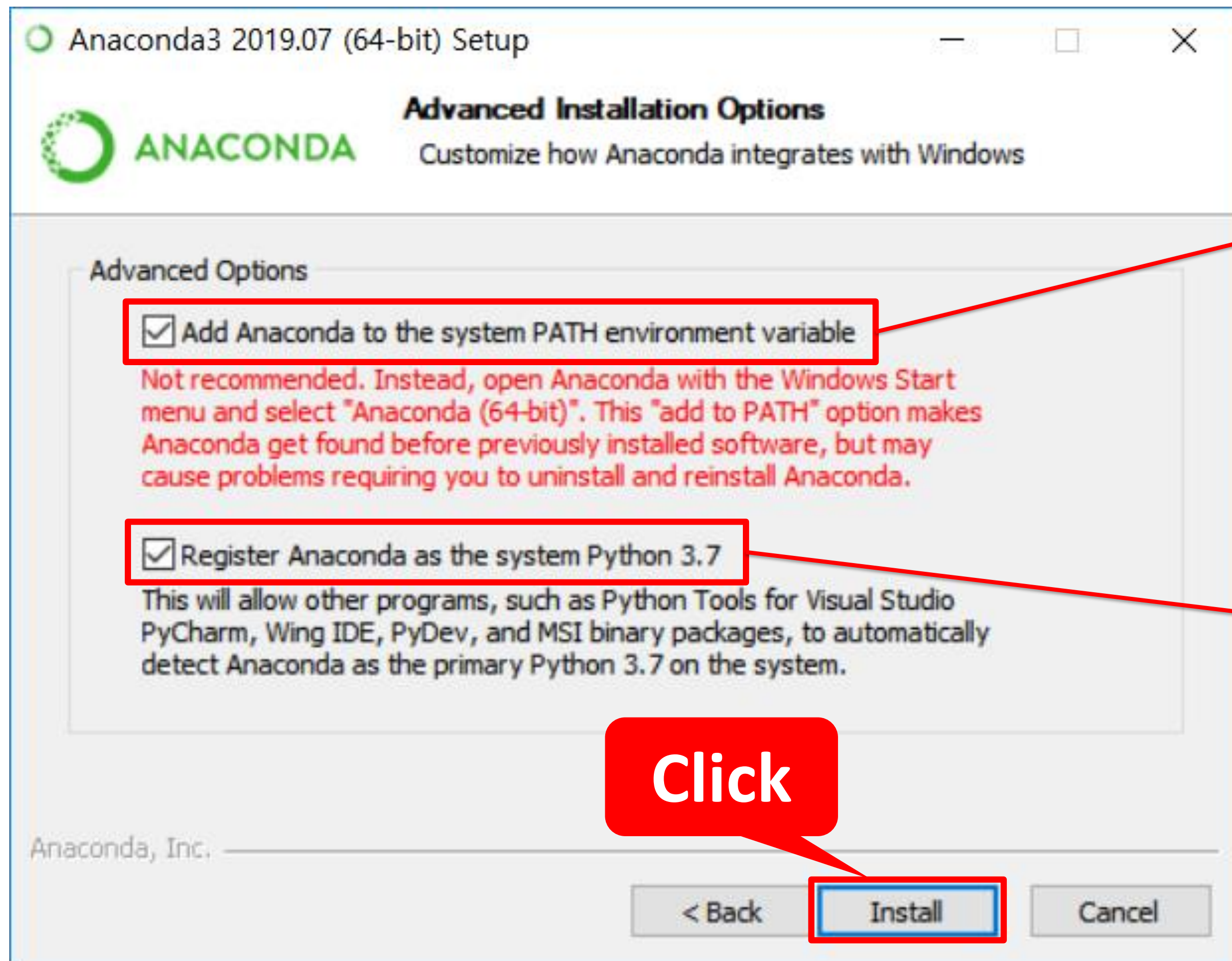
- C: \ Users \ MY_USERNAME \ AppData \ Local \ ~
- 위와 같은 경로에 설치되는 것을 확인할 수 있음

■ All Users

- C: \ ProgramData \ Anaconda3
- 위와 같은 경로에 설치되는 것을 확인할 수 있음

Anaconda 설치

➤ 설치 방법

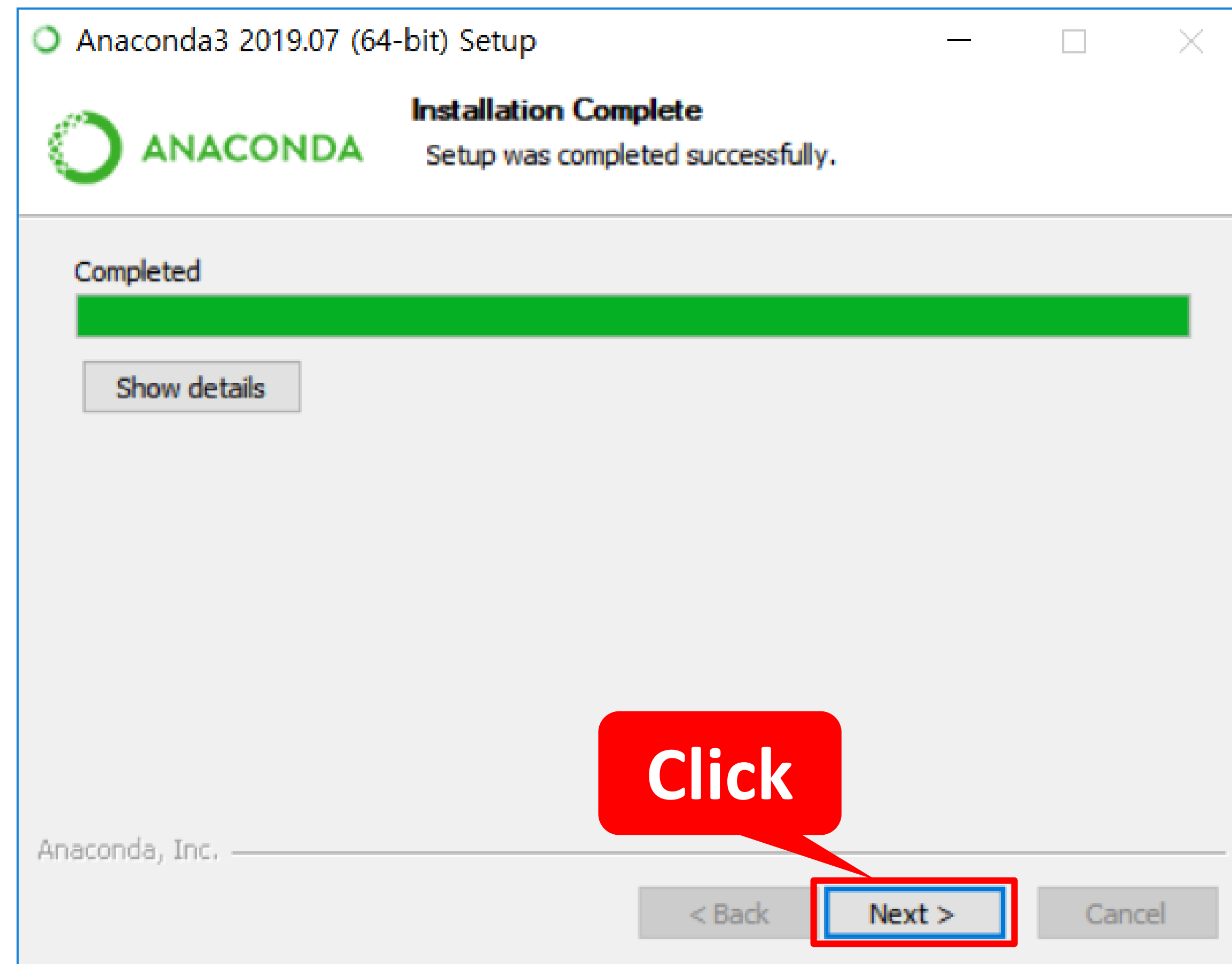


- Anaconda 이외에 다른 python 인터프리터를 환경변수에 등록해서 사용하고 있다면 해제
- 아나콘다만 사용하거나 윈도우 cmd창에서 파이썬을 실행할 경우 선택

- 아나콘다를 기본 파이썬 실행파일로 등록할지 여부
- 개발 도구나 에디터에서 아나콘다를 파이썬으로 인식

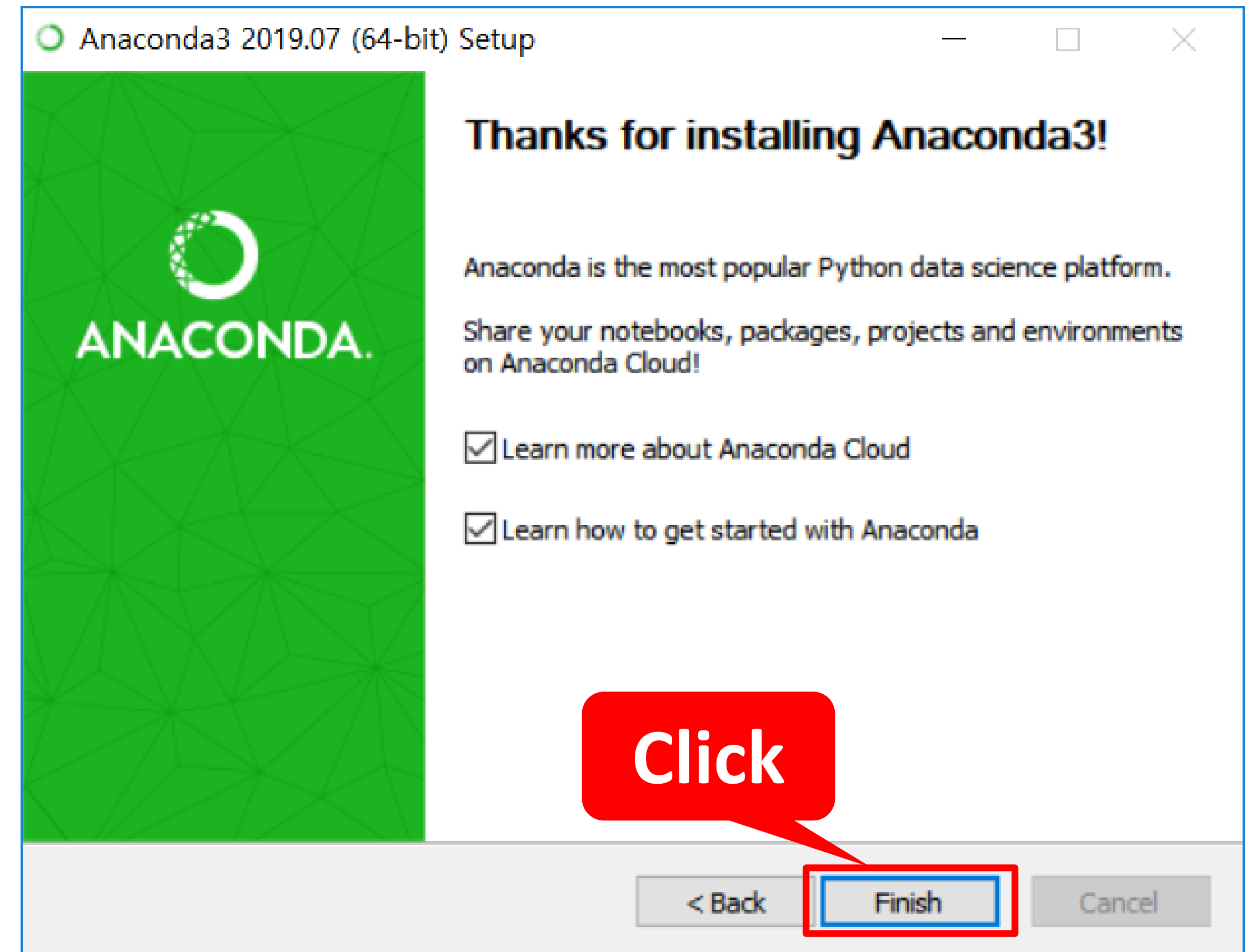
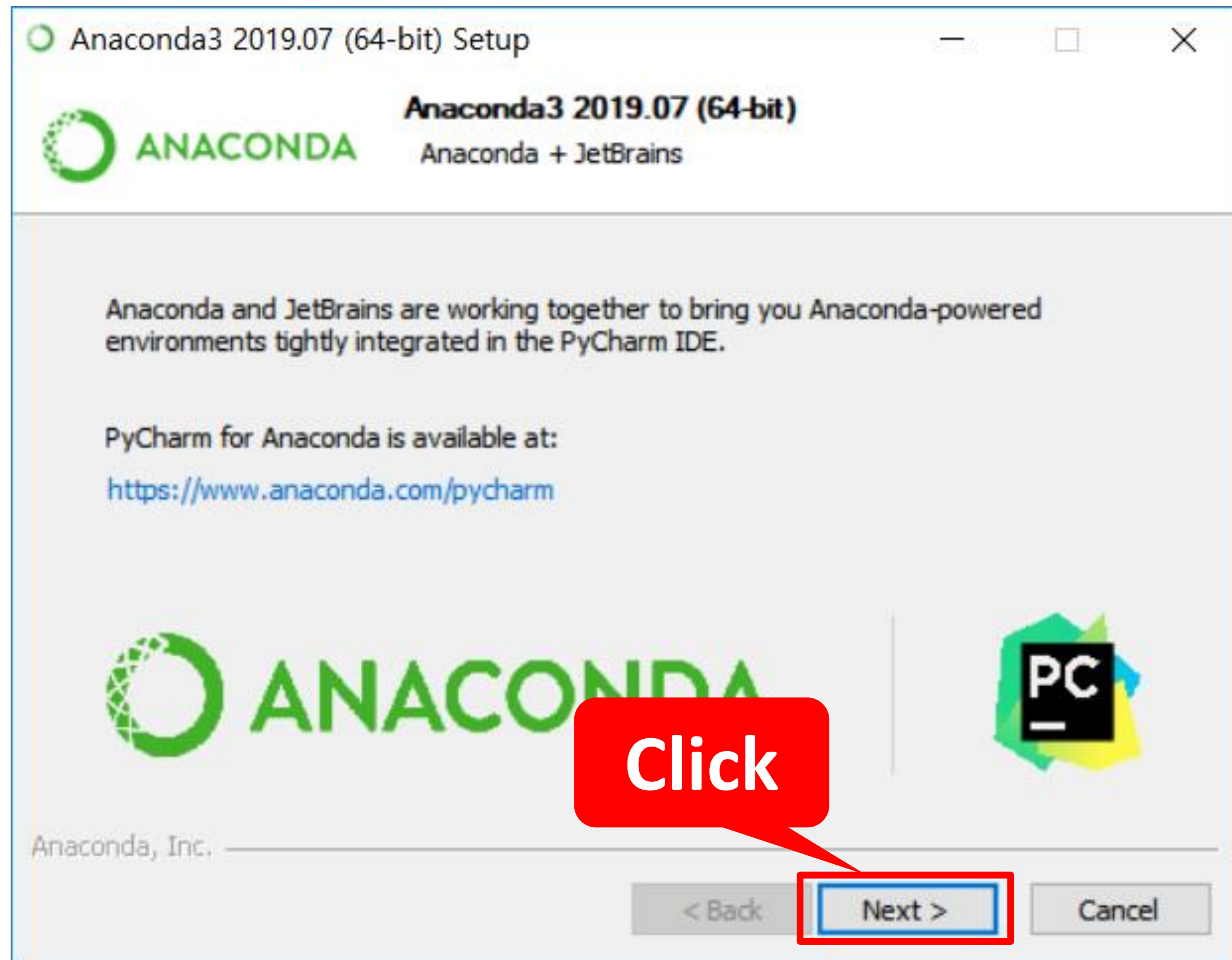
Anaconda 설치

➤ 설치 방법



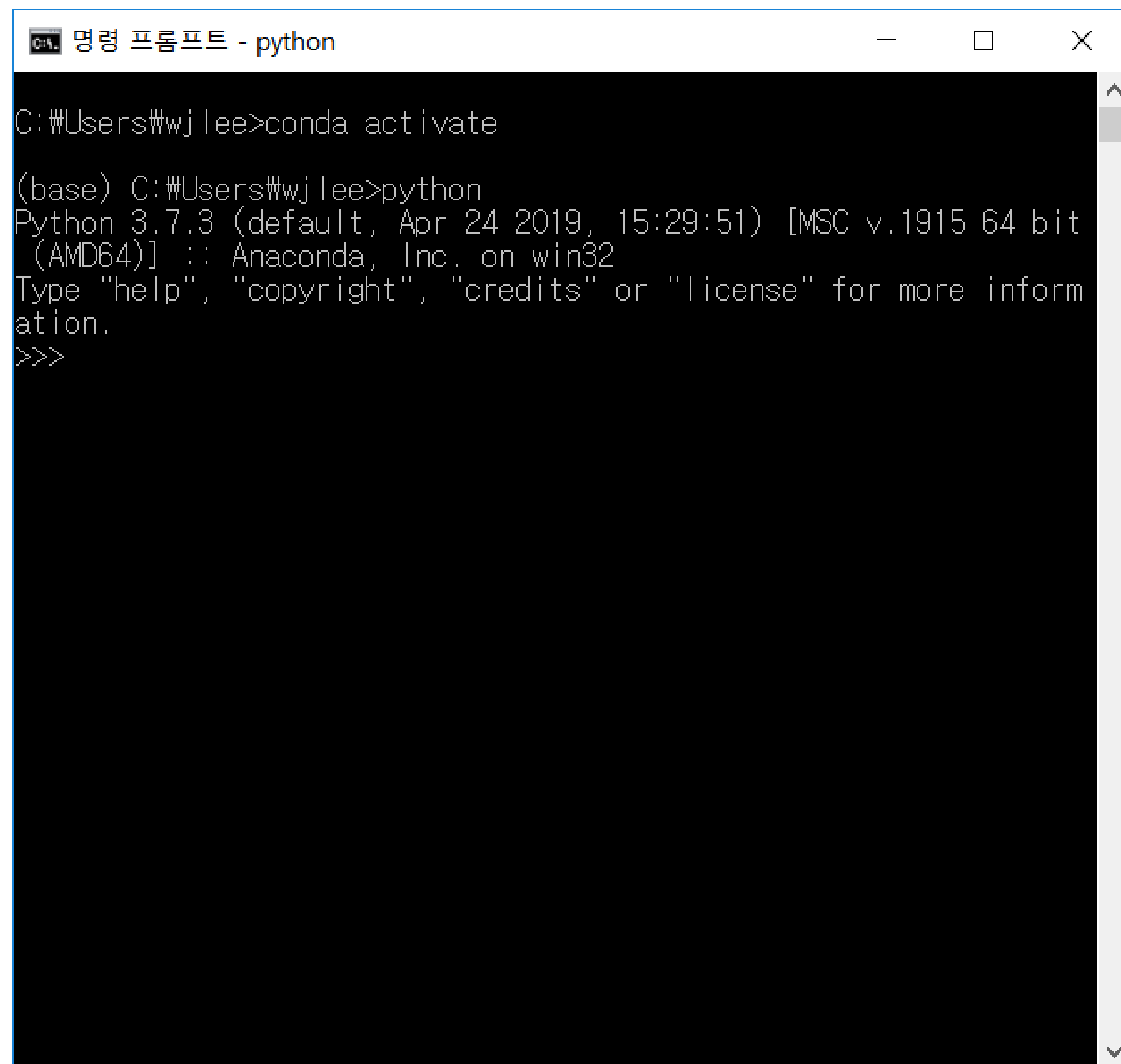
Anaconda 설치

➤ 설치 방법



Anaconda 실행

➤ Cmd창에서 Anaconda 실행

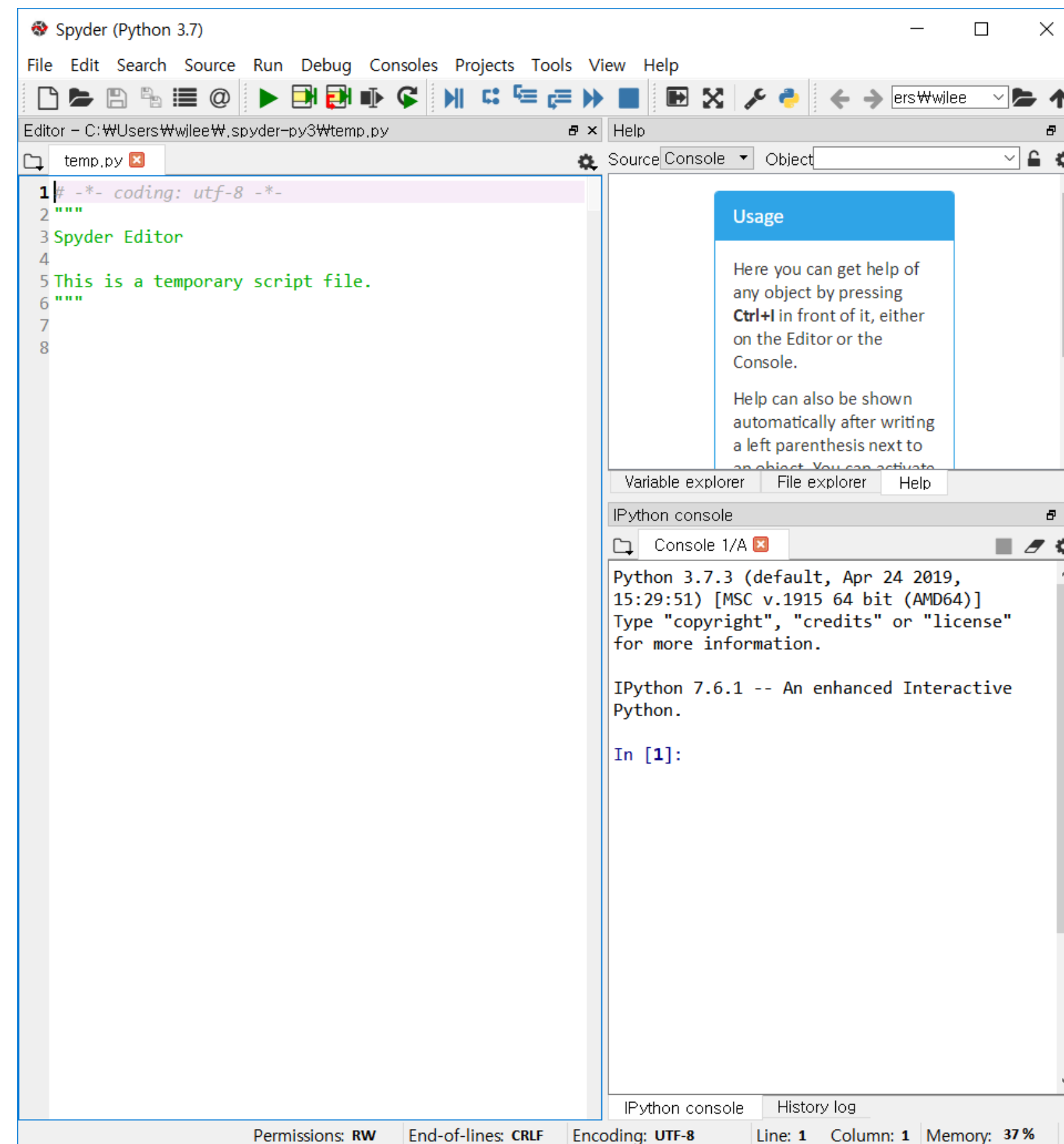
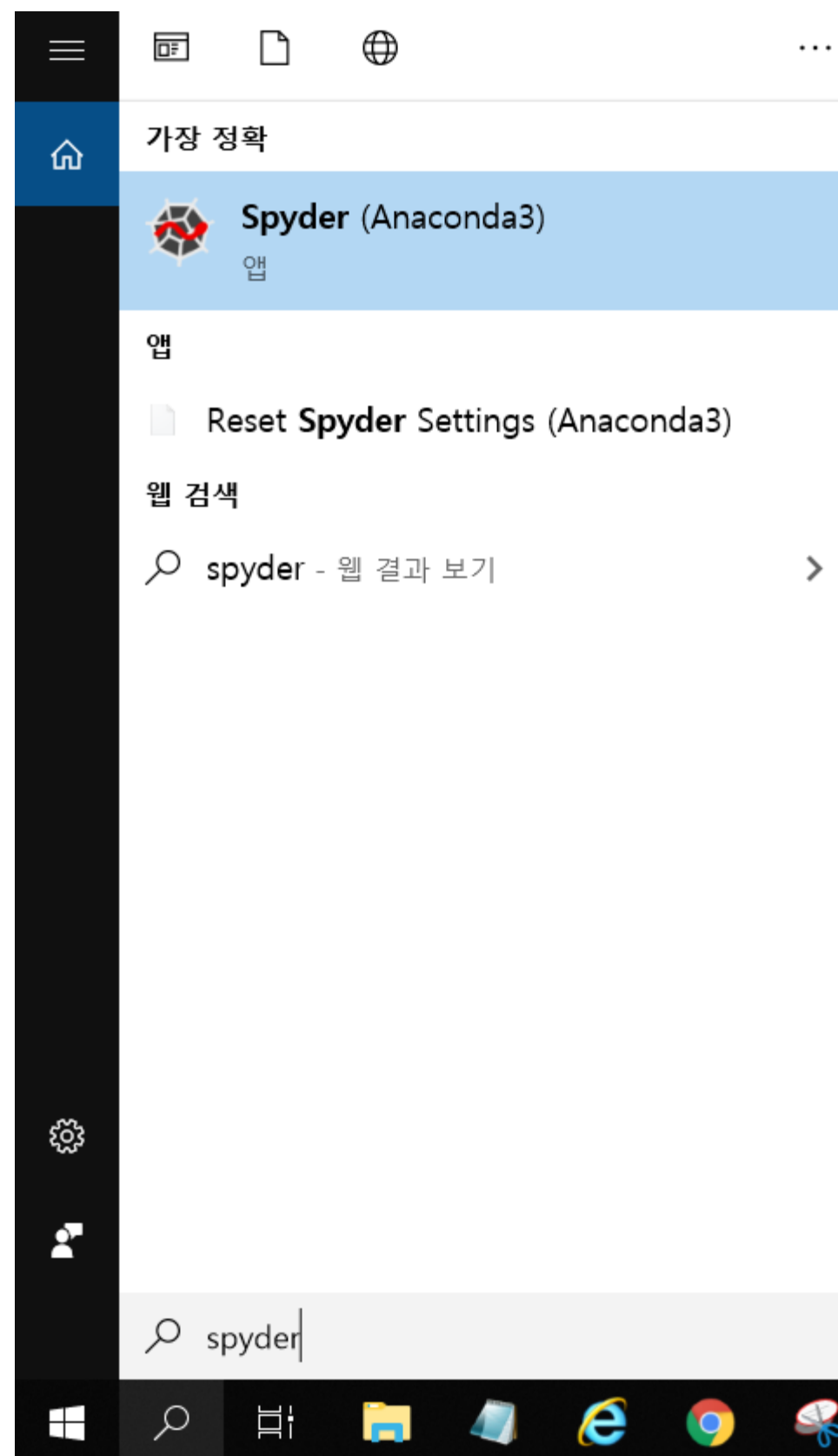


```
C:\#Users#wjlee>conda activate
(base) C:\#Users#wjlee>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit
(AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more inform
ation.
>>>
```

- Cmd창에서 anaconda를 실행가능함
 - conda activate
 - python
 - 순서대로 입력

Anaconda 실행

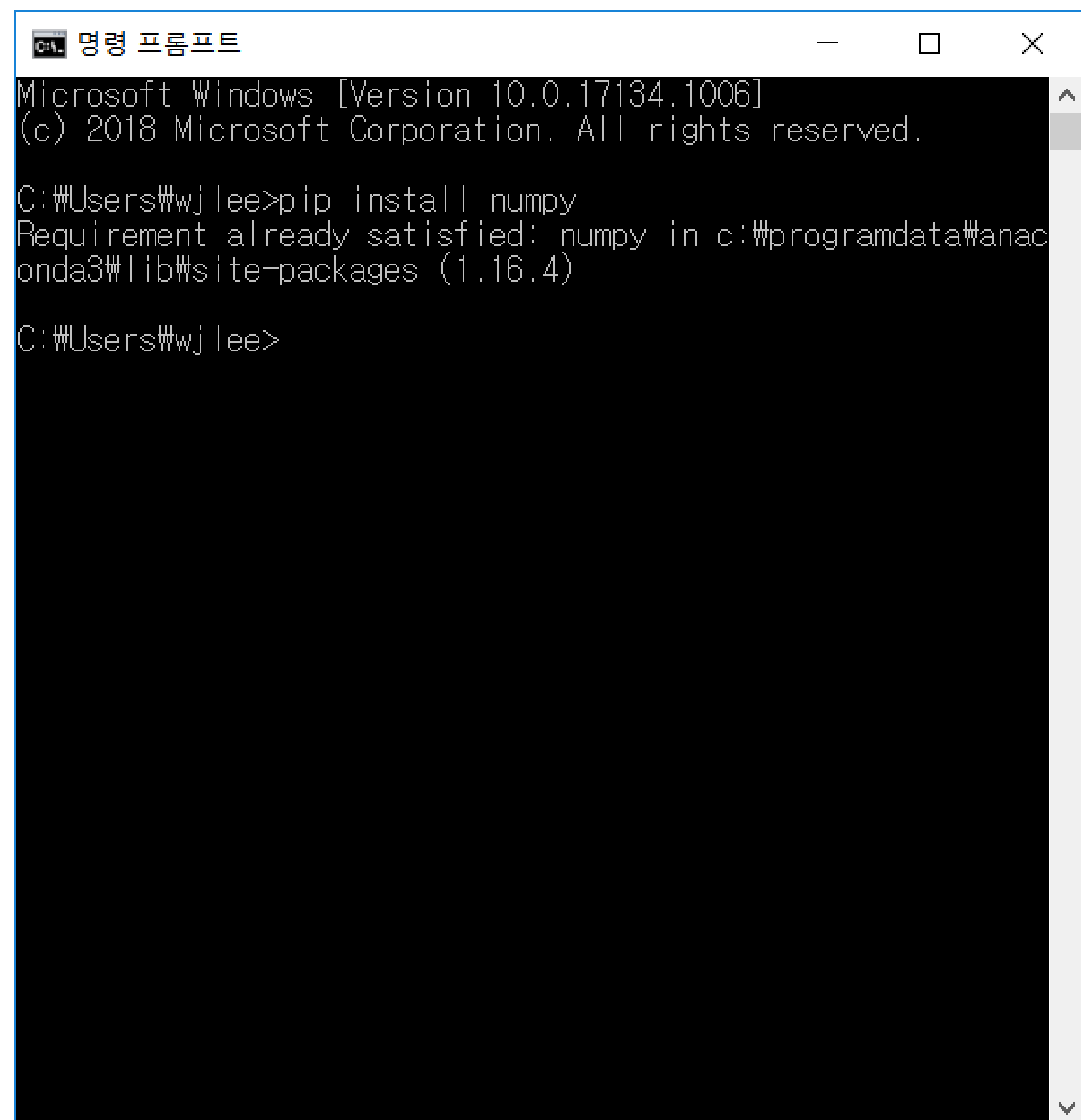
➤ Spyder editor 실행



패키지 설치

패키지 설치

➤ Cmd창을 이용하여 패키지 설치



```
Microsoft Windows [Version 10.0.17134.1006]
(c) 2018 Microsoft Corporation. All rights reserved.

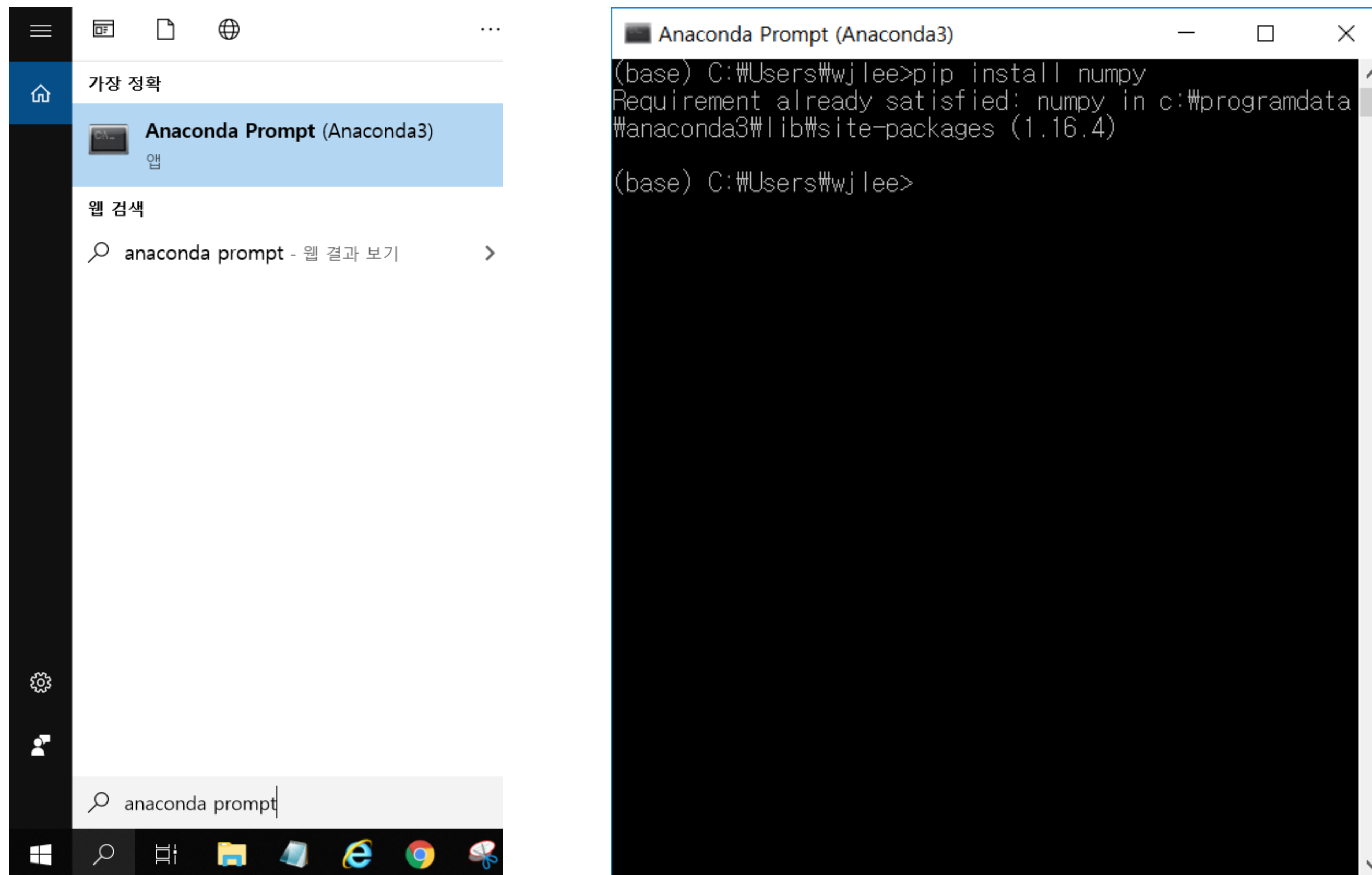
C:\Users\wjlee>pip install numpy
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.16.4)

C:\Users\wjlee>
```

- Cmd창에서 python package를 설치할 수 있음
 - pip install [Package name] 입력

패키지 설치

➤ Anaconda prompt를 이용하여 패키지 설치



■ Anaconda Prompt에서도 Package를 설치할 수 있음

- pip install [Package name]
입력

패키지 설치

➤ scikit-learn

- 파이썬 머신러닝 라이브러리
- Numpy와 Scipy를 사용
- 사용자 가이드 : http://scikit-learn.org/stable/user_guide.html
- API 문서 : <http://scikit-learn.org/stable/modules/classes.html>
- *pip install scikit-learn*

패키지 설치

➤ Numpy

- 다차원 배열을 위한 기능과 선형 대수 연산 등을 지원
- scikit-learn에서 numpy배열이 기본 데이터 구조
- *pip install numpy*
- example

```
import numpy as np  
  
x = np.array([[1,2,3], [4,5,6]])  
print("x: \n {}".format(x))
```

```
In [2]: runfile('C:/Users/shema/Downloads/prepro.py', wdir='C:/Users/shema/Downloads')  
x:  
[[1 2 3]  
 [4 5 6]]
```

패키지 설치

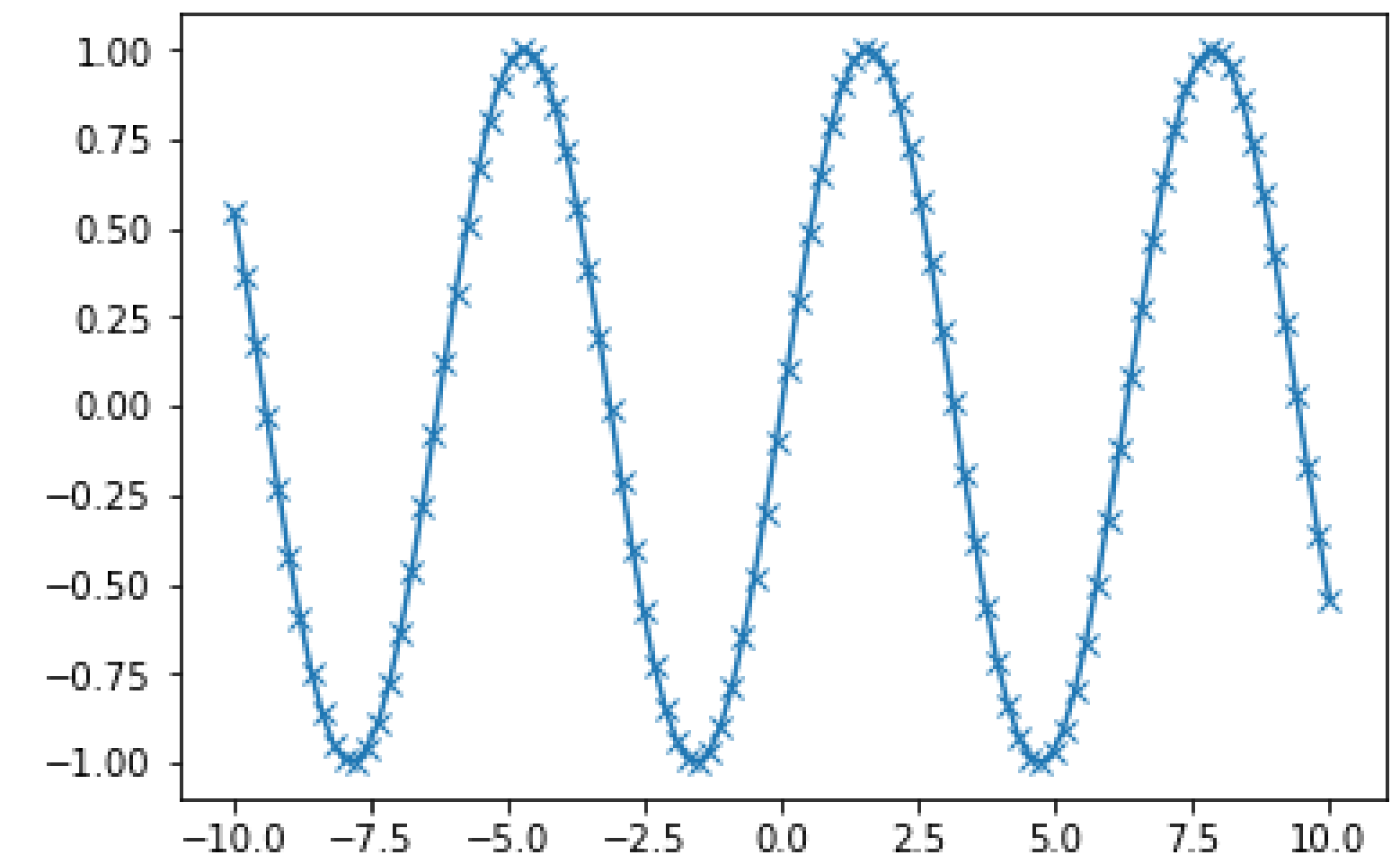
➤ matplotlib

- 파이썬 그래프 라이브러리
- User guide : <https://matplotlib.org/3.1.1/users/index.html>
- *pip install matplotlib*
- example

```
import numpy as np
import matplotlib.pyplot as plt

#-10에서 10까지 100개의 간격으로 나눈 배열들 생성
x = np.linspace(-10, 10, 100)
#사인(sin) 함수를 사용하여 y 배열들 생성
y = np.sin(x)
#플롯(plot) 함수는 한 배열의 값들 다른 배열에 대응해서 한 그래프를 그립니다.
plt.plot(x, y, marker = "x")
```

In [3]: `runfile('C:/Users/shema/Downloads/prepro.py', wdir=`



Numpy 사용법

배열 생성

- 파이썬의 리스트를 중첩해 Numpy 배열을 초기화 할 수 있고, 대괄호를 통해 각 요소에 접근할 수 있음

```
import numpy as np

a = np.array([1, 2, 3]) # rank가 1인 배열 생성
print type(a)          # 출력 "<type 'numpy.ndarray'>"
print a.shape          # 출력 "(3,)"
print a[0], a[1], a[2] # 출력 "1 2 3"
a[0] = 5               # 요소를 변경
print a                # 출력 "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]]) # rank가 2인 배열 생성
print b.shape                # 출력 "(2, 3)"
print b[0, 0], b[0, 1], b[1, 0] # 출력 "1 2 4"
```

배열 생성

➤ 리스트가 아니더라도 다양한 함수를 통해 배열 생성 가능

```
import numpy as np

a = np.zeros((2,2)) # 모든 값이 0인 배열 생성
print a           # 출력 "[[ 0.  0.]
                  #      [ 0.  0.]]"

b = np.ones((1,2)) # 모든 값이 1인 배열 생성
print b           # 출력 "[[ 1.  1.]]"

c = np.full((2,2), 7) # 모든 값이 특정 상수인 배열 생성
print c           # 출력 "[[ 7.  7.]
                  #      [ 7.  7.]]"

d = np.eye(2)      # 2x2 단위행렬 생성
print d           # 출력 "[[ 1.  0.]
                  #      [ 0.  1.]]"

e = np.random.random((2,2)) # 임의의 값으로 채워진 배열 생성
print e           # 임의의 값 출력 "[[ 0.91940167  0.08143941]
                  #      [ 0.68744134  0.87236687]]"
```


배열 인덱싱

➤ 슬라이싱 : 리스트 자료형의 방식과 동일

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# 슬라이싱을 이용하여 첫 두 행과 1열, 2열로 이루어진 부분배열을 만들어 봅시다;
# b는 shape가 (2,2)인 배열이 됩니다:
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# 슬라이싱된 배열은 원본 배열과 같은 데이터를 참조합니다,
# 즉 슬라이싱된 배열을 수정하면 원본 배열 역시 수정됩니다.
print a[0, 1]    # 출력 "2"
b[0, 0] = 77     # b[0, 0]은 a[0, 1]과 같은 데이터입니다
print a[0, 1]    # 출력 "77"
```

배열 인덱싱

➤ 정수 배열 인덱싱 : 원본과 다른 배열을 만들 수 있음

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# 정수 배열 인덱싱의 예.
# 반환되는 배열의 shape는 (3,)
print a[[0, 1, 2], [0, 1, 0]] # 출력 "[1 4 5]"

# 위에서 본 정수 배열 인덱싱 예제는 다음과 동일합니다:
print np.array([a[0, 0], a[1, 1], a[2, 0]]) # 출력 "[1 4 5]"

# 정수 배열 인덱싱을 사용할 때,
# 원본 배열의 같은 요소를 재사용할 수 있습니다:
print a[[0, 0], [1, 1]] # 출력 "[2 2]"

# 위 예제는 다음과 동일합니다
print np.array([a[0, 1], a[0, 1]]) # 출력 "[2 2]"
```

배열 인덱싱

➤ 불리언 배열 인덱싱 : 특정 조건을 만족하게 하는 요소만 선택할 때 자주 쓰임

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2) # 2보다 큰 a의 요소를 찾습니다;
                  # 이 코드는 a와 shape가 같고 불리언 자료형을
                  # 요소로 하는 numpy 배열을 반환합니다,
                  # bool_idx의 각 요소는 동일한 위치에 있는 a의
                  # 요소가 2보다 큰지를 알려줍니다.

print bool_idx      # 출력 "[[False False]
                   #      [ True  True]
                   #      [ True  True]]"

# 불리언 배열 인덱싱을 통해 bool_idx에서
# 참 값을 가지는 요소로 구성되는
# rank 1인 배열을 구성할 수 있습니다.
print a[bool_idx] # 출력 "[3 4 5 6]"

# 위에서 한 모든것을 한 문장으로 할 수 있습니다:
print a[a > 2]    # 출력 "[3 4 5 6]"
```

배열 연산

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]], dtype=np.float64)  
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
# 요소별 합; 둘 다 다음의 배열을 만듭니다  
# [[ 6.0  8.0]  
# [10.0 12.0]]  
print x + y  
print np.add(x, y)
```

```
# 요소별 차; 둘 다 다음의 배열을 만듭니다  
# [[-4.0 -4.0]  
# [-4.0 -4.0]]  
print x - y  
print np.subtract(x, y)
```

```
# 요소별 곱; 둘 다 다음의 배열을 만듭니다  
# [[ 5.0 12.0]  
# [21.0 32.0]]  
print x * y  
print np.multiply(x, y)
```

```
# 요소별 나눗셈; 둘 다 다음의 배열을 만듭니다  
# [[ 0.2          0.33333333]  
# [ 0.42857143  0.5          ]]  
print x / y  
print np.divide(x, y)
```

```
# 요소별 제곱근; 다음의 배열을 만듭니다  
# [[ 1.          1.41421356]  
# [ 1.73205081  2.          ]]  
print np.sqrt(x)
```

배열 연산

➤ 벡터의 내적, 벡터와 행렬의 곱을 위해서는 dot함수를 사용해야 함

```
import numpy as np

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# 벡터의 내적; 둘 다 결과는 219
print v.dot(w)
print np.dot(v, w)

# 행렬과 벡터의 곱; 둘 다 결과는 rank 1인 배열 [29 67]
print x.dot(v)
print np.dot(x, v)

# 행렬곱; 둘 다 결과는 rank 2인 배열
# [[19 22]
#  [43 50]]
print x.dot(y)
print np.dot(x, y)
```

Data Preprocessing

Data Preprocessing

➤ Read Data File

- np.loadtxt(“파일경로, 파일에서 사용한 구분자, 데이터 타입”)를 활용
- 또는 np.genfromtxt(“파일경로, 파일에서 사용한 구분자, 데이터타입”)을 활용
- example

```
import numpy as np
```

```
data = np.loadtxt('test.csv', delimiter = ',', dtype=float)  
print(data.shape)
```


Data Preprocessing

➤ label encoding

- Encode labels with value between 0 and n_classes-1.
- `sklearn.preprocessing.LabelEncoder()`
- example

```
In [94]: le = LabelEncoder()  
....: le.fit(["paris", "paris", "tokyo", "amsterdam"])  
....: LabelEncoder()  
....: list(le.classes_)  
....:
```

```
Out[94]: ['amsterdam', 'paris', 'tokyo']
```

```
In [95]: le.transform(["tokyo", "tokyo", "paris"])
```

```
Out[95]: array([2, 2, 1], dtype=int64)
```

Data Preprocessing

➤ One hot encoding

- Encode categorical integer features as a one-hot numeric array
- `sklearn.preprocessing.OneHotEncoder()`를 활용
- example

```
In [99]: import numpy as np
...: from sklearn.preprocessing import OneHotEncoder
...: enc = OneHotEncoder(categorical_features=[0], sparse=False)
...: X = np.array([[2, 4], [1, 1], [0, 50], [2, 25]])
...: print(X)
...:
[[ 2  4]
 [ 1  1]
 [ 0 50]
 [ 2 25]]

In [100]: X = enc.fit_transform(X)
...: print(X)
...:
[[ 0.  0.  1.  4.]
 [ 0.  1.  0.  1.]
 [ 1.  0.  0. 50.]
 [ 0.  0.  1. 25.]]
```

Data Preprocessing

➤ Shuffle

- `np.random.shuffle("데이터")`를 활용
- example

```
In [32]: import numpy as np
```

```
In [33]: M = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
In [34]: M
```

```
Out[34]:
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]])
```

```
In [35]: np.random.shuffle(M)
```

```
In [36]: M
```

```
Out[36]:
```

```
array([[7, 8, 9],  
       [1, 2, 3],  
       [4, 5, 6]])
```

Data Preprocessing

➤ Split datasets into random train and test subsets

- `sklearn.model_selection.train_test_split("데이터")`를 활용
- example

```
In [79]: import numpy as np
...: from sklearn.model_selection import train_test_split
...: X, y = np.arange(10).reshape((5, 2)), range(5)
...: print(X)
```

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

```
In [80]: X_train, X_test, y_train, y_test = train_test_split(
...:     X, y, test_size=0.33, random_state=42)
...:
```

```
In [81]: X_train
```

```
Out[81]:
```

```
array([[4, 5],
       [0, 1],
       [6, 7]])
```

Data Preprocessing

➤ Normalization

- sklearn.preprocessing안에 있는 클래스를 활용
 - ✓ StandardScaler(X) : 평균이 0, 표준편차가 1이 되도록 변환
 - ✓ MinMaxScaler(X) : 최대값, 최소값이 각각 1, 0 이 되도록 변환
 - ✓ MaxAbsScaler(X) : 0을 기준으로 절대값이 가장 큰 수가 1 또는 -1이 되도록 변환
- 사용법
 - ✓ fit() 를 실행해서 학습용 데이터로 분포 모수를 객체에 저장
 - ✓ transform() 을 실행해서 학습용 데이터를 변환
 - ✓ fit_transform() 을 실행해서 동시에 실행 가능

Data Preprocessing

➤ Normalization

- example

```
import numpy as np
from sklearn.preprocessing import StandardScaler
X = (np.arange(9, dtype=np.float) - 3).reshape(-1, 1)  # -3부터 5까지의 분포
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

```
In [38]: np.mean(X_scaled), np.std(X_scaled)
Out[38]: (0.0, 1.0)
```

Regression

Least Squares

Least squares practice

➤ Import package

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

➤ Load dataset & Preprocess

```
data = np.loadtxt("weight-height.csv", delimiter = ',', skiprows = 1, usecols = (1,2))
X = data[:int(data.shape[0]/2),0]
y = data[:int(data.shape[0]/2),1]
```

of data points = 5000
Features: height
Label: weight

Least squares practice

- Split the dataset into train & test sets

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size = 0.2, random_state = 42)
```

- Change the data shape

```
X_train = X_train.reshape(-1,1)  
X_test = X_test.reshape(-1,1)  
y_train = y_train.reshape(-1,1)  
y_test = y_test.reshape(-1,1)
```

Change the data into
one dimensional array

Least squares practice

- Train the model, and predict the label

```
# Create linear regression object  
regr = linear_model.LinearRegression()  
  
# Train the model using the training sets  
regr.fit(X_train, y_train)  
  
# Make predictions using the testing set  
prediction = regr.predict(X_test)
```

Fit : train the model
; fit(train_features, train_target)

Predict : predict the target using
trained model
; predict(test_features)

Least squares practice

➤ result

```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, prediction))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, prediction))

# Plot outputs
plt.scatter(X_test, y_test, color='black')
plt.plot(X_test, prediction, color='blue', linewidth=3)
plt.xlabel("height")
plt.ylabel("weight")

plt.xticks(())
plt.yticks(())

plt.show()
```

Mean_squared_error: find mean squared error regression loss

; mean_squared_error(test_target, prediction_target)

r2_score: ratio of variance between target value and predicted value

; r2_score(target, predicted)

- Least squares objective function

$$J_{\mathbf{w}} = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$

- Mean squared error

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

- r2_score

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Least squares practice

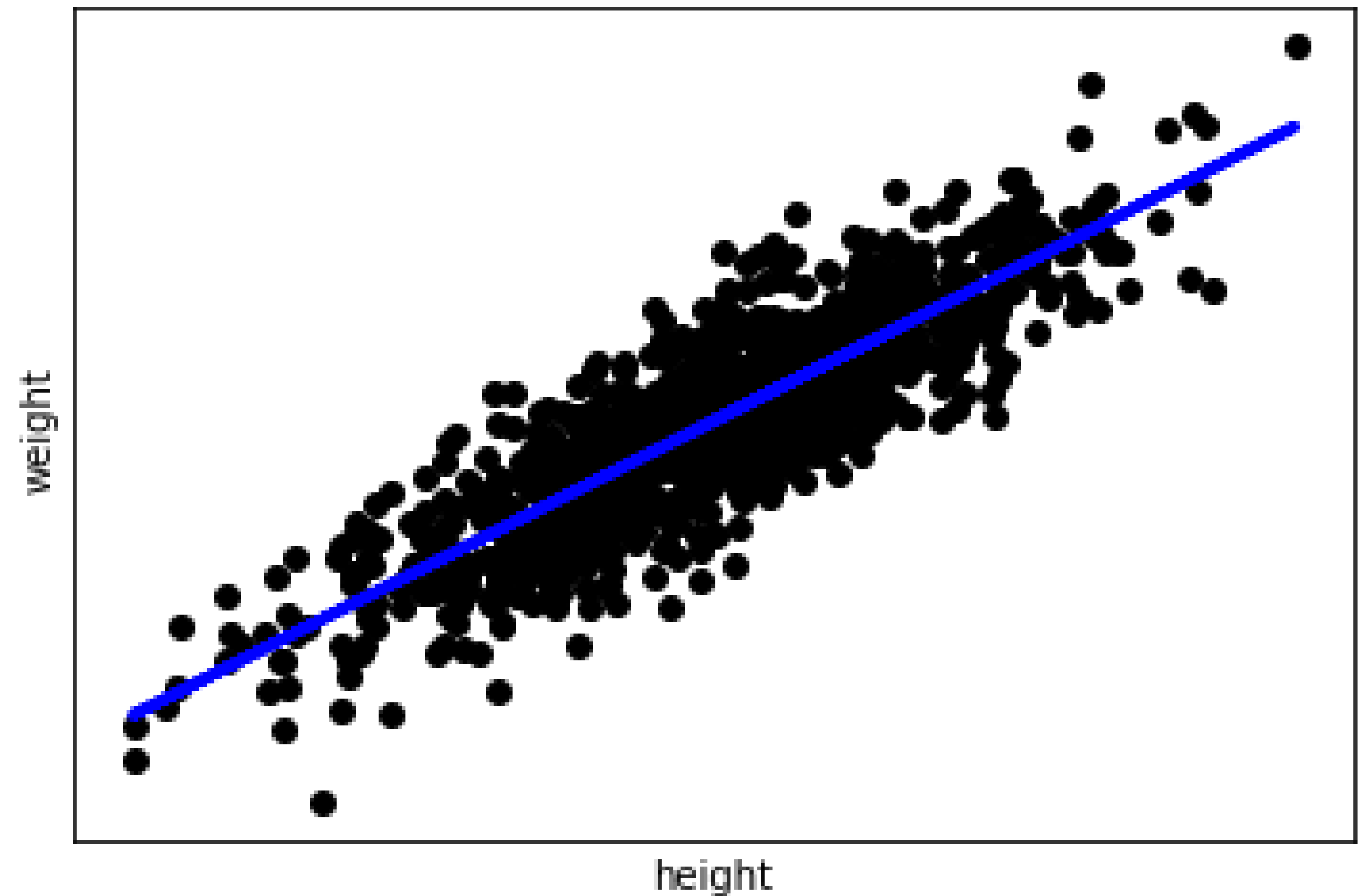
➤ result

Coefficients:

```
[[5.96322428]]
```

Mean squared error: 99.71

Variance score: 0.74



Ridge

Ridge practice

- Split the data into train, validation, test sets (3:1:1)

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size = 0.2, random_state = 42)
```

```
X_train, X_val, y_train, y_val = train_test_split(  
    X_train, y_train, test_size = 0.33, random_state = 42)
```

- Change the data shape

```
X_train = X_train.reshape(-1,1)  
X_test = X_test.reshape(-1,1)  
X_val = X_val.reshape(-1,1)  
y_train = y_train.reshape(-1,1)  
y_test = y_test.reshape(-1,1)  
y_val = y_val.reshape(-1,1)
```

Change the data into
one dimensional array

Ridge practice

➤ Find best parameter alpha using validation set

- Linear regression(Ridge) objective function

$$J_w = \underbrace{\frac{1}{2} \|X\mathbf{w} - \mathbf{y}\|_2^2}_{\text{loss}} + \underbrace{\lambda \|\mathbf{w}\|_2^2}_{L_2\text{-regularization}}$$

```
alphas = [0.0001, 0.001, 0.01, 0.1, 1]
mseList = []

for alpha in alphas:
    ridge = linear_model.Ridge(alpha = alpha)
    ridge.fit(X_train, y_train)
    prediction = ridge.predict(X_val)
    mse = mean_squared_error(y_val, prediction)
    mseList.append(mse)
```

```
idx = mseList.index(min(mseList))
```

Linear regression(Lasso) parameter:

- `alpha`: float
 - Regularization strength; reduces variance of the estimates
 - larger value specify stronger regularization
- `normalize`: Boolean
 - if True, data will be normalized before regression
 - subtracting the mean and dividing by the l2-norm
- `random_state`: int
 - random seed of shuffling data

Example:

```
Regr = linear_model.Ridge(alpha = 0.1, normalize = True,
                           random_state = 10)
```

Ridge practice

➤ Train & Test using best alpha value

```
# Create linear regression object
ridge = linear_model.Ridge(alphas[idx])

# Train the model using the training sets
ridge.fit(X_train, y_train)
```

```
# Make predictions using the testing set
prediction = ridge.predict(X_test)
```

```
# The best alpha
print("Best alpha: \n", alphas[idx])

# The coefficients
print('Coefficients: \n', ridge.coef_)

# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, prediction))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, prediction))
```

```
# Plot outputs
plt.scatter(X_test, y_test, color='black')
plt.plot(X_test, prediction, color='blue', linewidth=3)
plt.xlabel("height")
plt.ylabel("weight")

plt.xticks(())
plt.yticks(())

plt.show()
```

Ridge practice

➤ result

Best alpha:

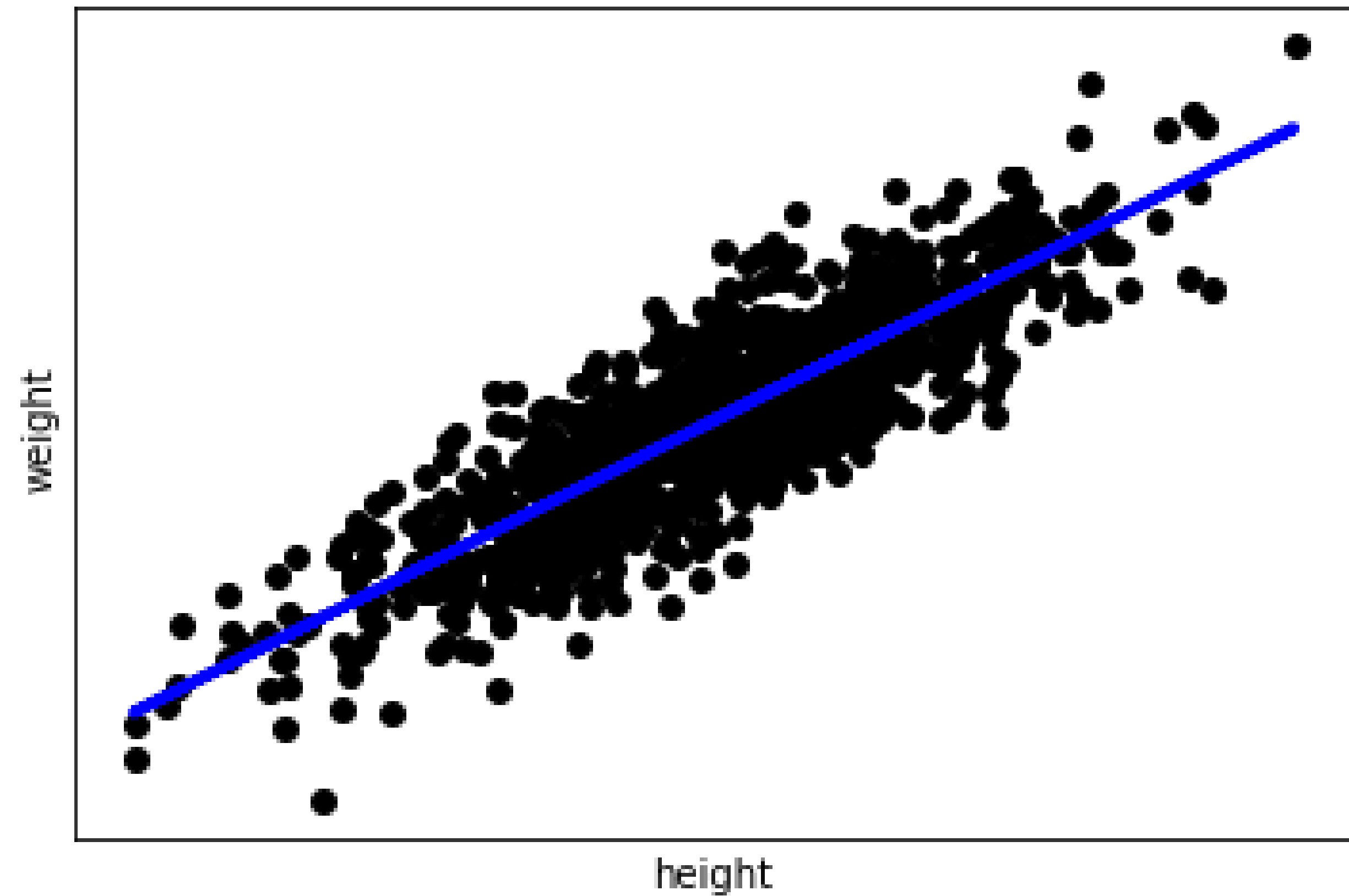
0.0001

Coefficients:

[[5.9325298]]

Mean squared error: 99.72

Variance score: 0.74



Lasso

Lasso practice

➤ Split the data into train, validation, test sets (3:1:1)

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size = 0.2, random_state = 42)
```

```
X_train, X_val, y_train, y_val = train_test_split(  
    X_train, y_train, test_size = 0.33, random_state = 42)
```

➤ Change the data shape

```
X_train = X_train.reshape(-1,1)  
X_test = X_test.reshape(-1,1)  
X_val = X_val.reshape(-1,1)  
y_train = y_train.reshape(-1,1)  
y_test = y_test.reshape(-1,1)  
y_val = y_val.reshape(-1,1)
```

Change the data into
one dimensional array

Lasso practice

➤ Find best parameter alpha using validation set

```
alphas = [0.0001, 0.001, 0.01, 0.1, 1]
mseList = []
```

```
for alpha in alphas:
    lasso = linear_model.Lasso(alpha = alpha)
    lasso.fit(X_train, y_train)
    prediction = lasso.predict(X_val)
    mse = mean_squared_error(y_val, prediction)
    mseList.append(mse)
```

```
idx = mseList.index(min(mseList))
```

- Linear regression(Lasso) objective function

$$J_w = \underbrace{\frac{1}{2} \|X\mathbf{w} - \mathbf{y}\|_2^2}_{\text{loss}} + \underbrace{\lambda \|\mathbf{w}\|_1}_{L_1\text{-regularization}}$$

Linear regression(Lasso) parameter:

- alpha**: float
 - Regularization strength; reduces variance of the estimates
 - larger value specify stronger regularization
- normalize: Boolean
 - if True, data will be normalized before regression
 - subtracting the mean and dividing by the l2-norm
- random_state: int
 - random seed of shuffling data

Example:

```
Regr = linear_model.Lasso(alpha = 0.1, normalize = True,
                           random_state = 10)
```

Lasso practice

➤ Train & Test using best alpha value

```
# Create linear regression object
lasso = linear_model.Lasso(alphas[idx])

# Train the model using the training sets
lasso.fit(X_train, y_train)
```

```
# Make predictions using the testing set
prediction = lasso.predict(X_test)
```

```
# The best alpha
print("Best alpha: \n", alphas[idx])
# The coefficients
print('Coefficients: \n', lasso.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(y_test, prediction))
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, prediction))
```

```
# Plot outputs
plt.scatter(X_test, y_test, color='black')
plt.plot(X_test, prediction, color='blue', linewidth=3)
plt.xlabel("height")
plt.ylabel("weight")

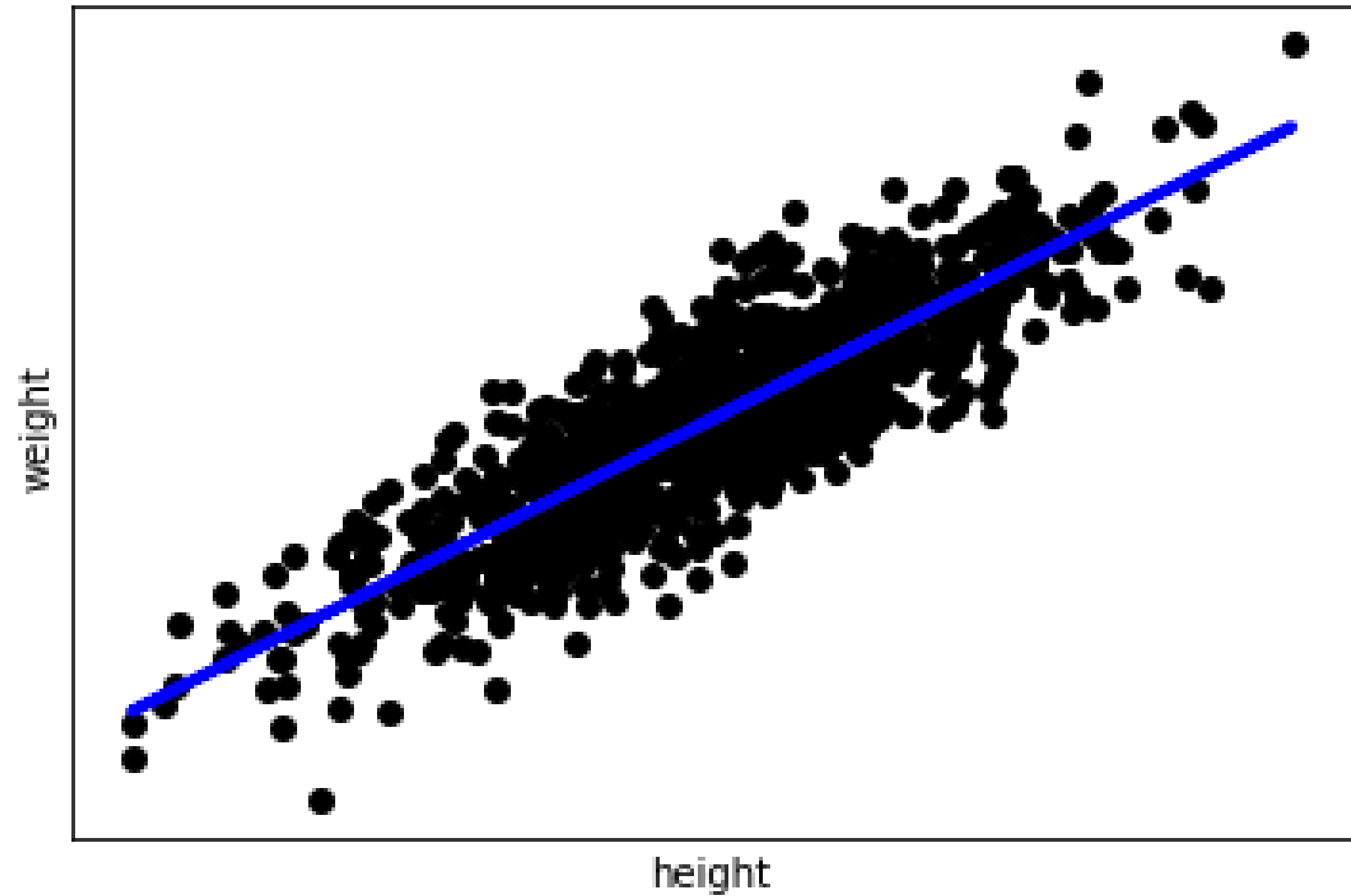
plt.xticks(())
plt.yticks(())

plt.show()
```

Lasso practice

➤ result

```
Best alpha:  
0.0001  
Coefficients:  
[5.93251779]  
Mean squared error: 99.72  
Variance score: 0.74
```



Exercise

Regression exercise

➤ Least squares, Ridge, Lasso regression 과 주어진 데이터를 활용하여 부동산 값을 예측하는 모델을 만드시오

- Dataset : Taipei, Sindian 지역의 부동산 거래 내역
 - ✓ UCI Repository : <https://archive.ics.uci.edu/ml/datasets/Real+estate+valuation+data+set>
 - ✓ 2-7 column 을 feature, 8 번째 column 을 label(가격)으로 활용해야 함
 - ✓ 데이터 별로 단위가 다르기 때문에 feature 별 normalization(StandardScaler) 이 필요함
 - ✓ Train set과 validation set, test set의 비율은 3:1:1로 할 것

Regression exercise

- Least squares, Ridge, Lasso regression 과 주어진 데이터를 활용하여 부동산 값을 예측하는 모델을 만드시오
 - Default 파라미터를 활용하여 모델을 training한 후, 모델의 coefficient 값을 분석하여 가격과 feature 사이의 관계를 파악할 것
 - 모델 별 Data normalization 전과 후의 RMSE 를 각각 report 할 것
 - Ridge, Lasso model 에서 validation set을 활용해서 alpha를 finetuning한 결과를 report할 것(alpha는 0.01, 0.1, 0.5, 1, 5, 10, 100 중에서 선택)

Regression exercise - code

➤ Import package

```
import numpy as np
from sklearn import linear_model, preprocessing
from sklearn.metrics import mean_squared_error, r2_score
```

➤ Data preprocessing

```
# data load
mat = np.genfromtxt('Real estate valuation data set.csv',
                    delimiter = ',', encoding = 'latin-1')

# Load feature
feat = np.genfromtxt('Real estate valuation data set.csv',
                    delimiter = ',', dtype = str, skip_footer= mat.shape[0] -1)
feat = list(feat[1:-1])

# head row, column 제거
mat = mat[1:,1:]
#data shuffle
np.random.seed(0)
np.random.shuffle(mat)
#feature, label로 분리
X = mat[:, :6]
Y = mat[:, 6]
```

numpy에 존재하는 함수 `genfromtxt` 를 통해 파일 읽기
IDE workspace에 파일이 있어야 함

Regression exercise - code

➤ Data preprocessing

```
# train, validation, test set으로 분리
N = mat.shape[0]
train_N = int(N * 0.6)
val_N = int(N*0.8)
#feature maxtrix 분리
train_X = X[:train_N, :]
val_X = X[train_N:val_N, :]
test_X = X[val_N:, :]
#Label maxtrix 분리
train_Y = Y[:train_N]
val_Y = Y[train_N:val_N]
test_Y = Y[val_N:]
```

mat에 저장되어 있는 데이터 개수(row 개수)를 추출함

Regression exercise - code

➤ Data preprocessing

```
#normalization
scaler = preprocessing.StandardScaler()
train_X = scaler.fit_transform(train_X)
val_X = scaler.transform(val_X)
test_X = scaler.transform(test_X)
```

Train_X에 대해서만 fit_transform을 사용하고 validation과 test데이터는 train_X의 평균과 분산으로 normalization함

➤ Train the Least squares regression

```
#Least squares
regr = linear_model.LinearRegression()
regr.fit(train_X, train_Y)
pred_Y= regr.predict(test_X)
mse = mean_squared_error(pred_Y, test_Y)
rmse = np.power(mse, 0.5)
print("Least Squares RMSE : ", rmse)
print("Least Squares variance : ", r2_score(test_Y, pred_Y))
print(feats)
print("Least Squares coef : ", regr.coef_)
print("")
```

모델 학습을 위해 fit 함수를 사용함

- Parameter로 train data와 train data의 label을 넣어 줌

Mean_squared_error함수로 RMSE를 구함

- Parameter로 data label에 대한 예측 값과, 실제 label을 넣어 줌

Regression exercise - code

➤ Train the Lasso regression

```
#lasso
alphas = [0.01, 0.1, 0.5, 1, 5, 10, 100]
mse_list = []
#finetune the alpha using the validation set
for alpha in alphas:
    lasso = linear_model.Lasso(alpha=alpha)
    lasso.fit(train_X, train_Y)
    pred_Y = lasso.predict(val_X)
    mse = mean_squared_error(pred_Y, val_Y)
    mse_list.append(mse)
idx = mse_list.index(min(mse_list))
print("Lasso best alpha : ", alphas[idx])
#evaluation
lasso = linear_model.Lasso(alpha=alphas[idx])
lasso.fit(train_X, train_Y)
pred_Y = lasso.predict(test_X)
mse = mean_squared_error(pred_Y, test_Y)
rmse = np.power(mse, 0.5)
print("Lasso RMSE : ", rmse)
print("Lasso variance : ", r2_score(test_Y, pred_Y))
print(feats)
print("Lasso coef : ", lasso.coef_)
print("")
```

Validation set으로 적합한 alpha를 찾음

제일 작은 mse의 index를 찾아서
alpha를 찾음

Regression exercise - code

➤ Train the Ridge regression

```
#Ridge
#finetune the alpha using the validation set
alphas = [0.01, 0.1, 0.5, 1, 5, 10, 100]
mse_list = []
for alpha in alphas:
    ridge= linear_model.Ridge(alpha=alpha)
    ridge.fit(train_X, train_Y)
    pred_Y = ridge.predict(val_X)
    mse = mean_squared_error(pred_Y, val_Y)
    mse_list.append(mse)
idx = mse_list.index(min(mse_list))
print("Ridge alpha : ", alphas[idx])
#evaluation
ridge = linear_model.Ridge(alpha=alphas[idx])
ridge.fit(train_X, train_Y)
y_pred = ridge.predict(test_X)
mse = mean_squared_error(y_pred, test_Y)
rmse = np.power(mse, 0.5)
print("Ridge RMSE : ", rmse)
print("Ridge variacne : ", r2_score(test_Y, y_pred))
print(feat)
print("Ridge coef : ", ridge.coef_)
```

Validation set으로 적합한 alpha를 찾음

제일 작은 mse의 index를 찾아서
alpha를 찾는다.

Regression exercise - code

➤ Result

Least Squares RMSE : 8.329080506609118

Least Squares variance : 0.5675316887713147

['X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X6 longitude']

Least Squares coef : [1.06694012 -2.73791507 -5.20141889 4.31355208 2.61784154 -0.18961479]

Lasso best alpha : 0.01

Lasso RMSE : 8.327283147025737

Lasso variance : 0.5677183161415046

['X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X6 longitude']

Lasso coef : [1.05357142 -2.7254224 -5.14083379 4.30909262 2.61887392 -0.126183]

Ridge alpha : 0.01

Ridge RMSE : 8.329064752925534

Ridge variacne : 0.5675333247172003

['X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X6 longitude']

Ridge coef : [1.06685768 -2.73779212 -5.20064144 4.31347472 2.6179507 -0.18895764]

Regression exercise

➤ Least squares, Ridge, Lasso regression 과 주어진 데이터를 활용하여 졸업
가능성을 예측하는 모델을 만드시오

■ Dataset : UCLA graduate dataset

✓ Kaggle dataset :

<https://www.kaggle.com/mohansacharya/graduate-admissions/downloads/graduate-admissions.zip/2>

✓ 2-6 column 을 feature, 7 번째 column 을 label으로 활용해야 함

✓ 데이터 별로 단위가 다르기 때문에 feature 별 normalization(MinMaxScaler) 이 필요함

✓ Train set과 validation set, test set의 비율은 6:1:1로 할 것

Regression exercise

- Least squares, Ridge, Lasso regression 과 주어진 데이터를 활용하여 졸업 가능성을 예측하는 모델을 만드시오
 - Train, Validation, Test set을 나눌 때 random state는 42로 설정할 것
 - Default 파라미터를 활용하여 모델을 train한 후, 모델의 coefficient 값을 분석하여 졸업 가능성과 feature 사이의 관계를 파악할 것
 - Ridge, Lasso model에서 validation set을 활용해서 alpha를 finetuning한 결과를 report할 것(alpha는 0.0001, 0.001, 0.01, 0.1, 1, 10 중에서 선택할 것)

Regression exercise - code

➤ Import package

```
import numpy as np
from sklearn import linear_model
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

➤ Load the data

```
# Load data
data = np.genfromtxt("Admission_Predict.csv", delimiter=',',
                    dtype = float, skip_header = 1)

feat = np.genfromtxt("Admission_Predict.csv", delimiter=',',
                    dtype = str, skip_footer = data.shape[0])
feat = list(feat[1:-1])
```

Feature의 이름을 나타내는 첫 번째 row는 불러들일 필요가 없으므로 제거

Regression exercise - code

➤ Data preprocessing

Split the data and Label

```
X = data[:,1:-1]
```

```
y = data[:, -1]
```

- 첫 번째 column은 데이터가 저장된 순서를 나타내므로 의미가 없어 제거함
- Data와 Label을 나눔

Shuffle the data and Label

```
np.random.seed(0)
```

```
np.random.shuffle(X)
```

```
np.random.shuffle(y)
```

- 데이터의 순서가 결과에 영향을 미칠 수 있으므로 데이터를 무작위로 섞음

Regression exercise - code

➤ Data preprocessing

```
# Split the data: Train, Validation, Test set
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size = 1/8, random_state = 42)
```

```
X_train, X_val, y_train, y_val = train_test_split(  
    X_train, y_train, test_size = 1/7, random_state = 42)
```

```
# Normalize the data
```

```
scaler = MinMaxScaler()
```

```
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
```

```
X_val = scaler.transform(X_val)
```

Regression exercise - code

➤ Train the Least Squares regression model

```
# Least squares
```

```
least = linear_model.LinearRegression()  
least.fit(X_train, y_train)  
prediction = least.predict(X_test)
```

- fit 함수를 통해 least squares model을 학습함

```
mse = mean_squared_error(prediction, y_test)  
rmse = np.power(mse, 0.5)
```

- Regression모델의 evaluation metric인 root mean squared error를 구함

```
print("##### Least squares results #####")  
print("RMSE: {}".format(rmse))  
print("variance: {}".format(r2_score(prediction, y_test)))  
print(feats)  
print("coefficient:", least.coef_)  
print("")
```

Regression exercise - code

➤ Train the Ridge linear regression model

```
# Ridge
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10]
mseList = []
for alpha in alphas:
    ridge = linear_model.Ridge(alpha = alpha)
    ridge.fit(X_train, y_train)
    prediction = ridge.predict(X_val)
    mse = mean_squared_error(prediction, y_val)
    mseList.append(mse)
idx = mseList.index(min(mseList))

print("##### Ridge results #####")
print("Best alpha: {}".format(alphas[idx]))
ridge = linear_model.Ridge(alphas[idx])
ridge.fit(X_train, y_train)
prediction = ridge.predict(X_test)
mse = mean_squared_error(prediction, y_test)
rmse = np.power(mse, 0.5)
print("RMSE: {}".format(rmse))
print("variance: {}".format(r2_score(prediction, y_test)))
print(feat)
print("coef : ", ridge.coef_)
print("")
```

- Ridge linear regression objective function의 lambda 후보

- Alpha값을 validation set에 다르게 적용했을 때 결과를 저장하기 위한 변수

- MSE가 가장 적은 alpha값의 위치를 찾음

- 해당 alpha값을 model에 적용함

Regression exercise - code

➤ Train the Lasso linear regression model

```
# Lasso
alphas = [0.0001, 0.001, 0.01, 0.1, 1, 10]
mseList = []
for alpha in alphas:
    lasso = linear_model.Lasso(alpha = alpha)
    lasso.fit(X_train, y_train)
    prediction = ridge.predict(X_val)
    mse = mean_squared_error(prediction, y_val)
    mseList.append(mse)
idx = mseList.index(min(mseList))

print("##### Lasso results #####")
print("Best alpha: {}".format(alphas[idx]))
lasso = linear_model.Lasso(alphas[idx])
lasso.fit(X_train, y_train)
prediction = lasso.predict(X_test)
mse = mean_squared_error(prediction, y_test)
rmse = np.power(mse, 0.5)
print("rmse: {}".format(rmse))
print("variance: {}".format(r2_score(prediction, y_test)))
print("coef : ", ridge.coef_)
```

- Ridge linear regression objective function의 lambda 후보

- Alpha값을 validation set에 다르게 적용했을 때 결과를 저장하기 위한 변수

- MSE가 가장 적은 alpha값의 위치를 찾음

- 해당 alpha값을 model에 적용함

Regression exercise - code

➤ Result

```
##### Least squares results #####
```

```
RMSE: 0.15032341322239787
```

```
variance: -73.47422970635319
```

```
['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research']
```

```
coefficient: [ 0.13808716 -0.1198897  0.01421541 -0.01201848 -0.01771465 -0.04601956  
-0.00360469]
```

```
##### Ridge results #####
```

```
Best alpha: 0.0001
```

```
RMSE: 0.15032336008317196
```

```
variance: -73.47843232523361
```

```
['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research']
```

```
coef : [ 0.13808101 -0.11988496  0.01421503 -0.01201925 -0.01771453 -0.04601725  
-0.00360437]
```

```
##### Lasso results #####
```

```
Best alpha: 0.0001
```

```
RMSE: 0.1499507857832885
```

```
variance: -87.84351498695084
```

```
['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR ', 'CGPA', 'Research']
```

```
coef : [ 0.13808101 -0.11988496  0.01421503 -0.01201925 -0.01771453 -0.04601725  
-0.00360437]
```