Python Simulation Setup & Linear Regression Practice (for Windows OS)

Hanwool Jeong

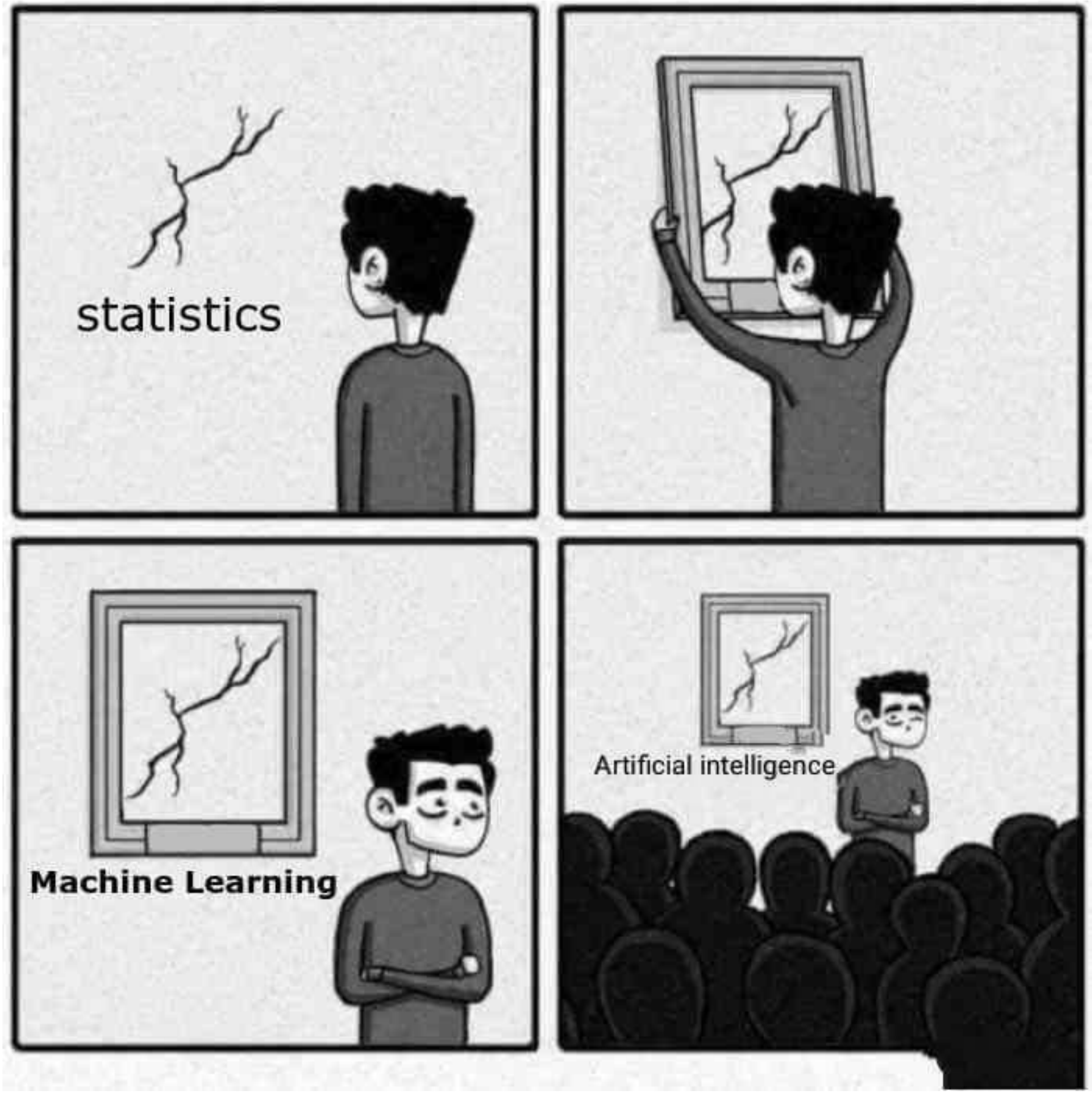
hwjeong@kw.ac.kr

Why Python?

- Easy to learn/read/**maintain**
- Abundant library/packages ➔ What is package?
- Anaconda

➔ Python distribution packaged with useful libraries for math/engineering

➔ Included libraries: Panda, **numpy**, scipy, sklearn, matplotlib

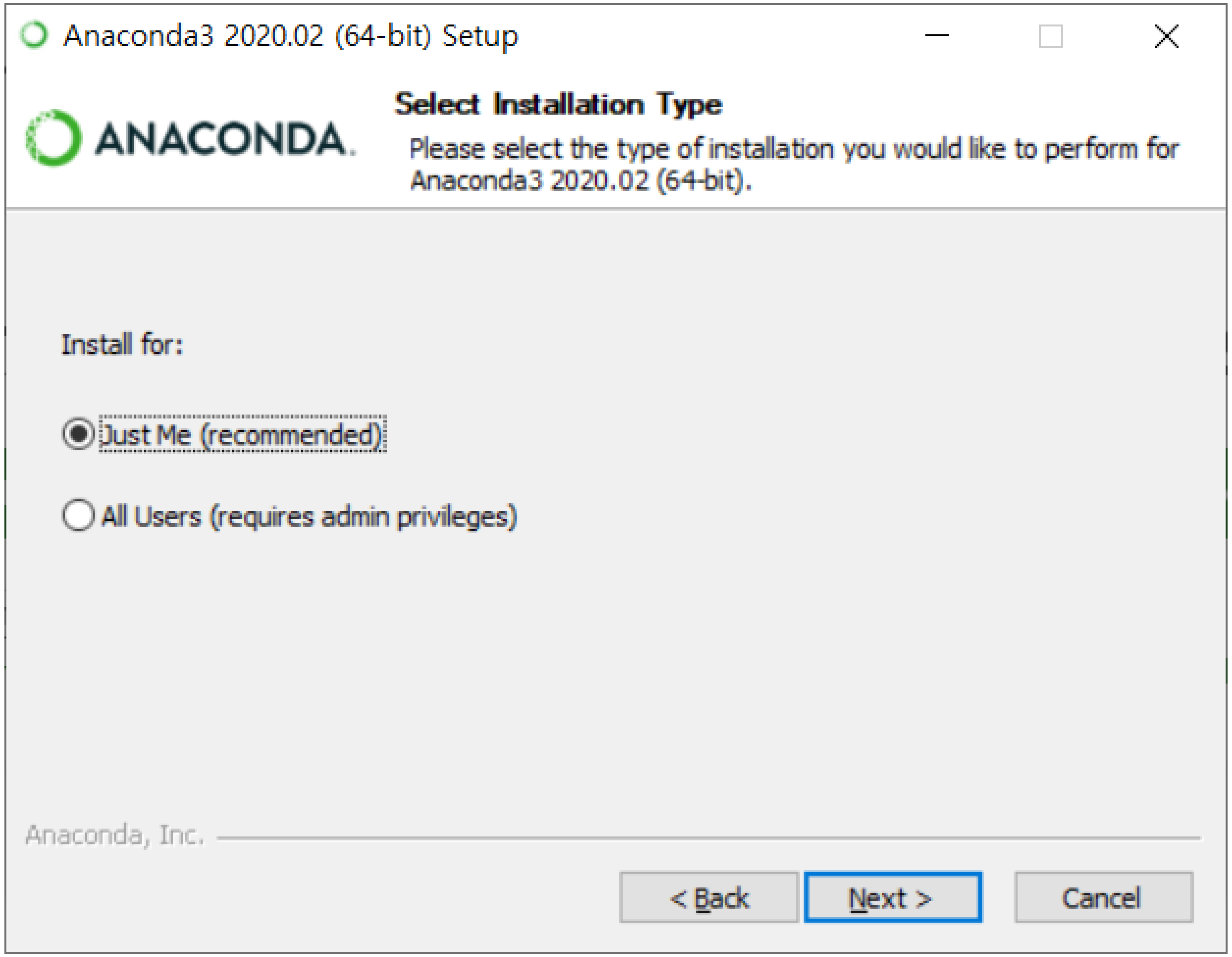


AI Expert ≠ S/W Expert

Downloading Anaconda

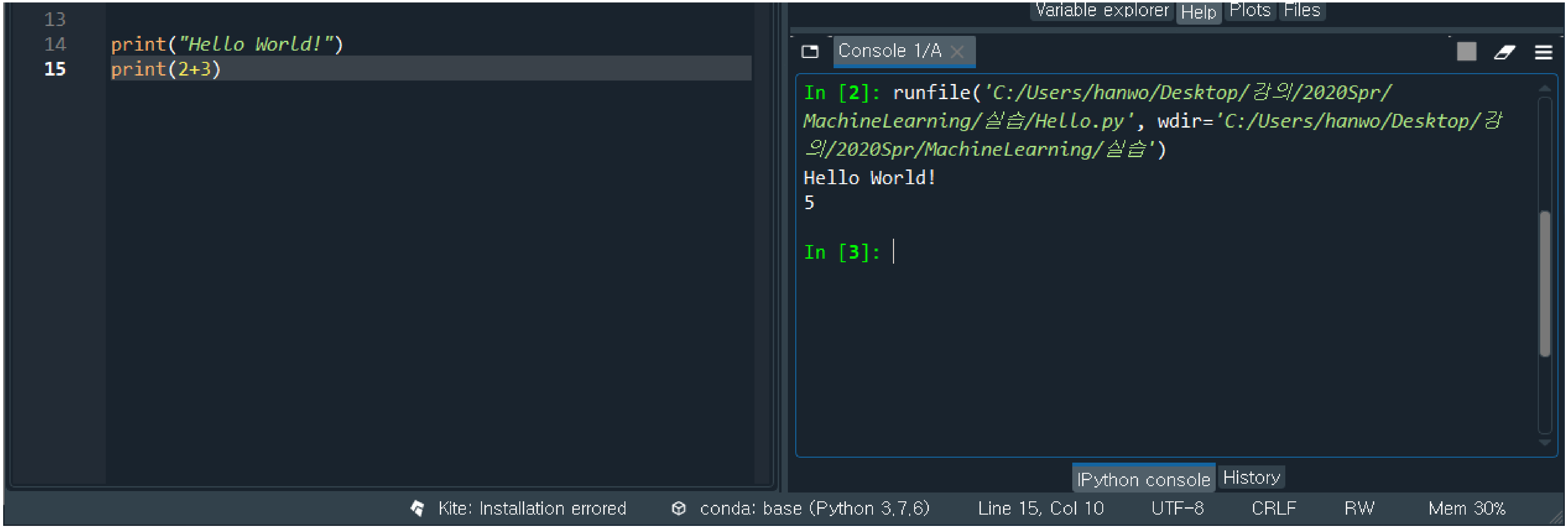
- https://www.anaconda.com/distribution/

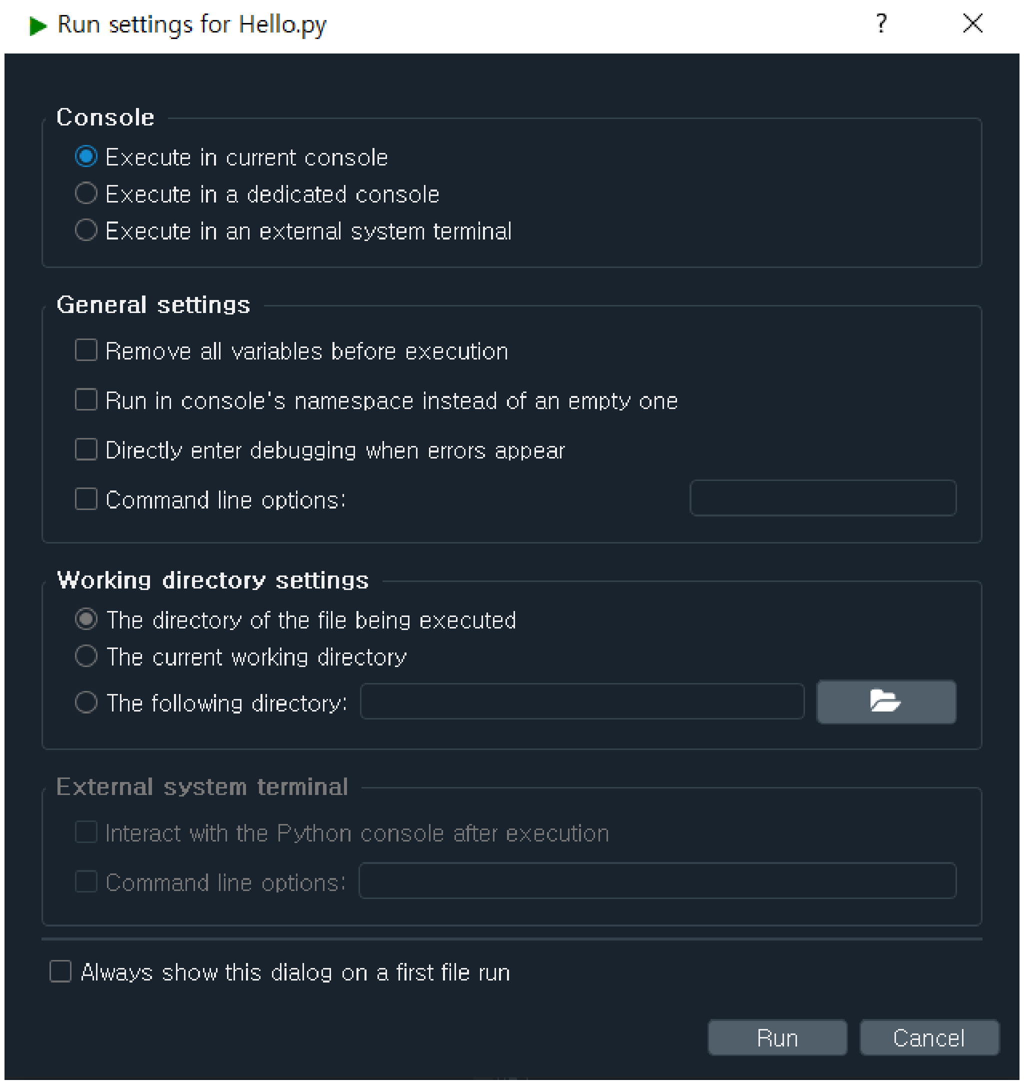
Proceeding Install Steps



Spyder

- Develop environment for anaconda: **Spyder** & **Jupyter Notebook**
- Spyder is recommended for Windows user
  - Editor
  - Variable Explorer: Examining variable dependency and revision of variables
  - Profiler: Analyzing the run time portion of each code
  - Debugger : line by line running


Variable explorer
Help
Plots
Files
Console 1/A
13
14 print("Hello World!")
15 print(2+3)
In [2]: runfile('C:/Users/hanwo/Desktop/강의/2020Spr/MachineLearning/실습/Hello.py', wdir='C:/Users/hanwo/Desktop/강의/2020Spr/MachineLearning/실습')
Hello World!
5
In [3]:
IPython console
History
Kite: Installation errored
conda: base (Python 3.7.6)
Line 15, Col 10
UTF-8
CRLF
RW
Mem 30%



Run settings for Hello.py
Console
Execute in current console
Execute in a dedicated console
Execute in an external system terminal
General settings
Remove all variables before execution
Run in console's namespace instead of an empty one
Directly enter debugging when errors appear
Command line options:
Working directory settings
The directory of the file being executed
The current working directory
The following directory:
External system terminal
Interact with the Python console after execution
Command line options:
Always show this dialog on a first file run
Run
Cancel

Tuple & List in Python

- Two representative methods for expressing array or vector
  - List declaration ➔ Used for undetermined featured data
  - Tuple is not modifiable

```python
my_list = [1,2,3,4,5]
my_tuple = (1,2,3,4,5)
```

Feature of List in Python

- + operation for list

```
midterm = [20, 40, 50]
final = [70, 80, 95]

print(midterm+final)
```

- * operation?

```
print(midterm*3)
```

Import Module in Python

- For example, can you evaluate $\sqrt{2}$ directly in python?, like

```python
print(sqrt(2))
```

- Try this. What is the role of the second line code?

```python
1 # -*- coding: utf-8 -*-
2 import math
3 print(math.sqrt(2))
```

- How about these?

```python
1 # -*- coding: utf-8 -*-
2 import math as m
3 print(m.sqrt(2))
4
```

- Is there any way we can directly use sqrt()?

Utilizing Numpy Package

- Try this and find the usefulness of numpy

```python
# -*- coding: utf-8 -*-

import numpy as np

midterm = np.array([20, 40, 50])
final = np.array([70, 80, 95])
print(midterm+final)

```

- Indexing & slicing (start or end value can be omitted)
- What else we can do with numpy?

Array Generation Functions in Numpy

- Handling 2D array

```
test2dArray = [[3,4,5],[100,200,300],[-2,-4,-5]]
test2dArrayNP = np.array(test2dArray)
```

➔ Indexing or slicing?

- **arange()** : np.arange(start, stop, step)

➔Try

```
arangeTest=np.arange(5, )
```

```
arangeTest=np.arange(3, 10)
```

```
arangeTest=np.arange(3, 10, 2)
```

- **zeros()** & **ones()**

```
zeroTest = np.zeros([2,3])
```

➔ Find about **ones_like()** or **zeros_like()**

- **linspace()** & **logspace()**

```
linspaceTest = np.linspace(0, 10, 21)
```

```
logspaceTest = np.logspace(1, 100, 4)
```

Concatenating Functions in Numpy

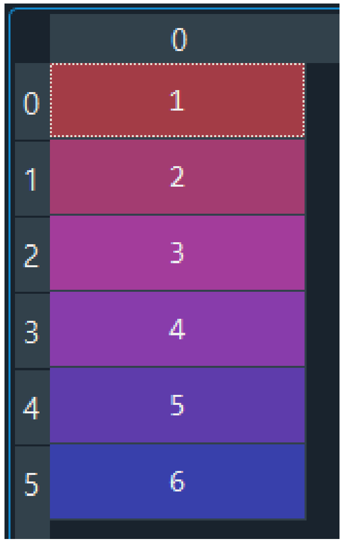
- Try **r_[va,vb]** & **c_[va,vb]** as:

  Also try xr2 = [[va], [vb]]

```
va = np.array([1, 2, 3])
vb = np.array([4, 5, 6])
xr = np.r_[va, vb]
xc = np.c_[va, vb]
```

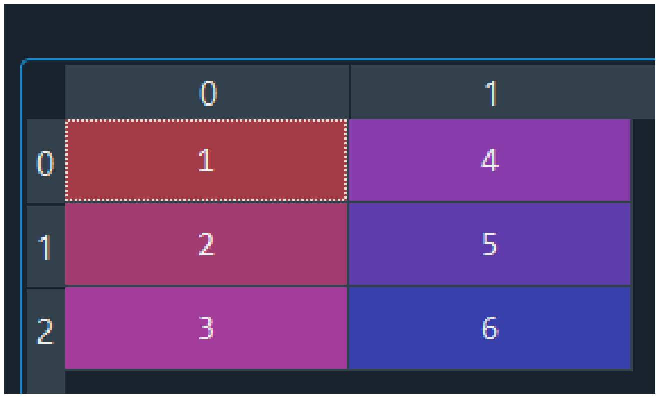
  - [numpy.r_ — NumPy v1.20 Manual]
  - [numpy.c_ — NumPy v1.20 Manual]

- Please check the results in “Variable explorer” as:



	0
0	1
1	2
2	3
3	4
4	5
5	6

xc - NumPy array



	0	1
0	1	4
1	2	5
2	3	6

Reshape() Function

- Usage of reshape():

new_array = old_array.reshape((4,3))

- Try this:

```
import numpy as np

y = np.arange(12)
print(y.reshape(3,4))
print(y.reshape(2,6))
```

Uniform Distributed
Random Number Generation

- **random.rand()** ➔ Uniform distribution between [0,1]
- You can setup a seed with **random.seed()**. Try the following two codes and run multiple times.

```
np.random.seed(10)
y = np.random.rand(5)
print(y)
```

```
y = np.random.rand(5)
print(y)
```

- **random.rand()** for generating random number [a,b]

```
y2 = (b-a)*np.random.rand(5)+a
```

- **random.randint()**

```
y3=np.random.randint(3,8, size=[4,6])
```

Gaussian (or Normal) Distributed Random Number Generation

- For standardized Normal Distribution, (μ = 0 and σ^2=1) with size of 4x7 array

```
yG = np.random.randn(4,7)
```

- You can easily generate normal random numbers following different μ and σ^2. How?

Linear Algebra

- Try this:

```
import numpy as np

A = [[2,3], [4,5]]
B = [[2,3], [4,5]]
ANP = np.array(A)
BNP = np.array(B)
print(ANP)
ANP_transpose = ANP.T
print(ANP_transpose)

C = np.dot(ANP,BNP)
print(C)
D1 = np.dot(ANP_transpose,BNP)
print(D1)
D2 = ANP.T.dot(BNP)
print(D2)
```

- Search about the following functions in the numpy and do some examples.

1) np.dot(x, y) 2) np.diag 3) np.trace 4) np.linalg.det
5) np.linalg.inv 6) np.linalg.svd 7) np.linalg.solve

Utilizing Matplotlib.pylab for Data Plot

- In the **matplotlib**, there is a subpackage called pylab. See https://matplotlib.org/api/pyplot_api.html
- We can use this subpackage to visualize data in python
- The conventional import statements:

```
import matplotlib.pylab as plt
```

Plot Data

```
import matplotlib.pylab as plt

plt.title("Test for Simple plot")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.plot([160, 165, 170, 175, 180], [60, 58, 70, 74, 72], "ro:")
plt.show()
```

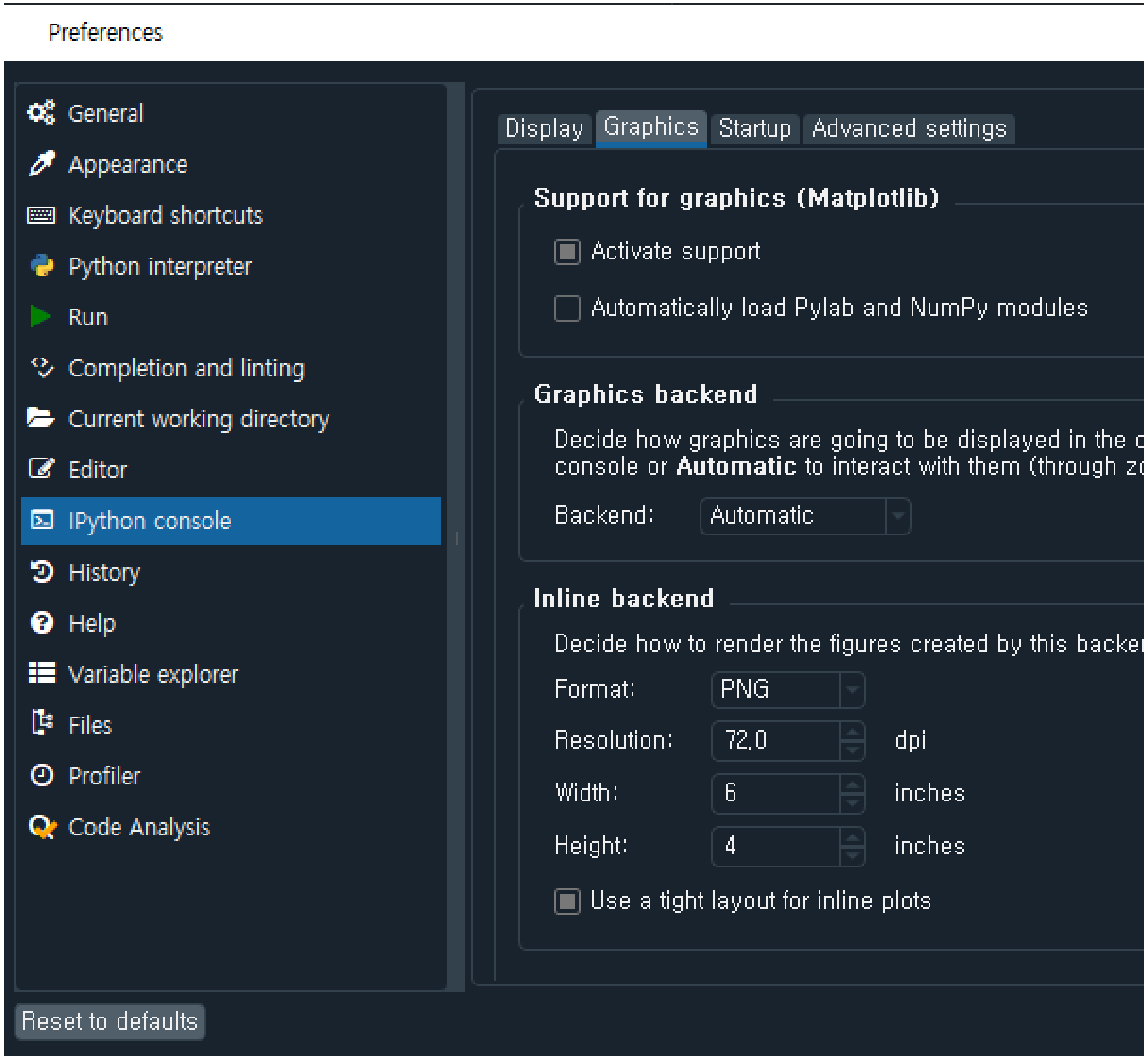
- Try this
- Please check “Plot” pane.
- About “ro:” which is the last component of plt.plot():

Color	Abbrev.
Blue	b
Green	g
Red	r
Cyan	c
Magenta	m
Yellow	y
Black	k
White	w

Marker	Meaning
.	point marker
,	pixel marker
o	circle marker
v	triangle_down marker
^	triangle_up marker
<	triangle_left marker
>	triangle_right marker
s	Square marker
p	Pentagon marker
*	Star maker

Line Style	Meaning
-	solid
--	dashed
--.	dash-dot
:	dotted

For New Window Plot,



Utilizing Pandas Loading Data

- Try this

```
import pandas as pd

dfLoad = pd.read_csv("https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/testData_H_vs_W.txt", sep="\s+")
```

- File path (Please copy it)
  https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/testData_H_vs_W.txt

- Please check whether it contains 100 height-weight combination data through "Variable Explorer."

testData_H_vs_W.txt -

파일(F) 편집(E) 서식(O)

Height	Weight
174.863	77.8929
167.431	71.69
157.067	51.2712

Index	Height	Weight
0	174.863	77.8929
1	167.431	71.69
2	157.067	51.2712
3	172.367	75.3819

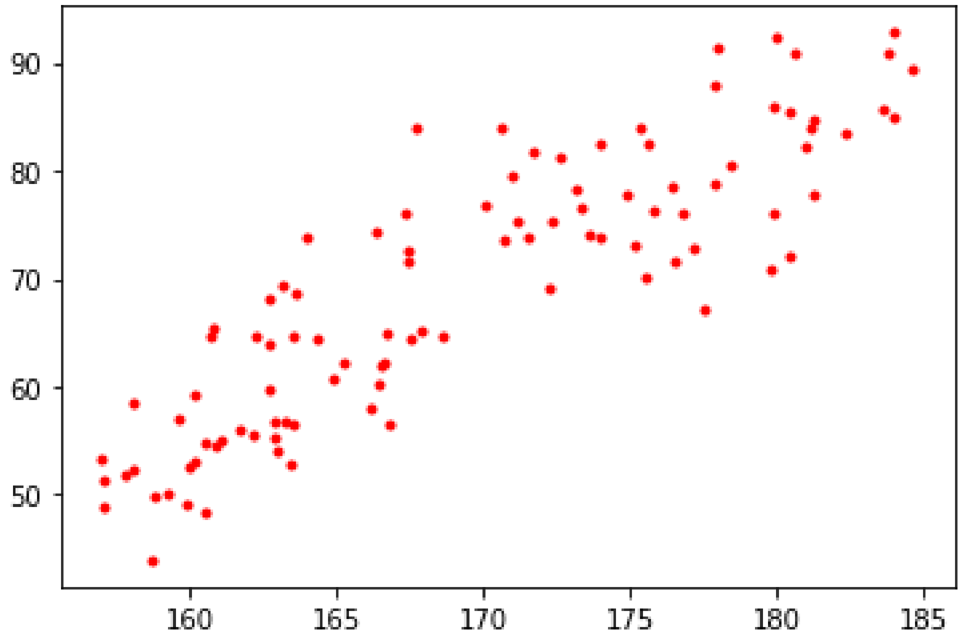
Plotting the Load Data

- Try this

```
import matplotlib.pylab as plt
import pandas as pd

dfLoad = pd.read_csv("https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/testData_H_vs_W.txt", sep="\s+")
xHeight = dfLoad["Height"]
yWeight = dfLoad["Weight"]
plt.plot(xHeight, yWeight, ".r")
```

Index	Height	Weight
0	174.863	77.8929
1	167.431	71.69
2	157.067	51.2712
3	172.367	75.3819



Revisit Normal Equation

- In linear regression, the w_{OLS} making the RSS minimized is:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{y} \Rightarrow \hat{\mathbf{w}}_{OLS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

  where ***X*** is N×D design matrix containing N feature vectors

- **Don't forget to add x_0 = 1 padding** to given data to derive w_0 for bias (or y-intercept) in linear regression model learned:

$$\hat{y} = h_w(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} = w_0 + w_1x_1 + w_2x_2 + \ldots w_Dx_D$$

Deriving wOLS in Python

- File path 2 (Please copy this): https://raw.githubusercontent.com/hanwoolJeong/lectureUniv/main/testData_LinearRegression.txt
- Try this:

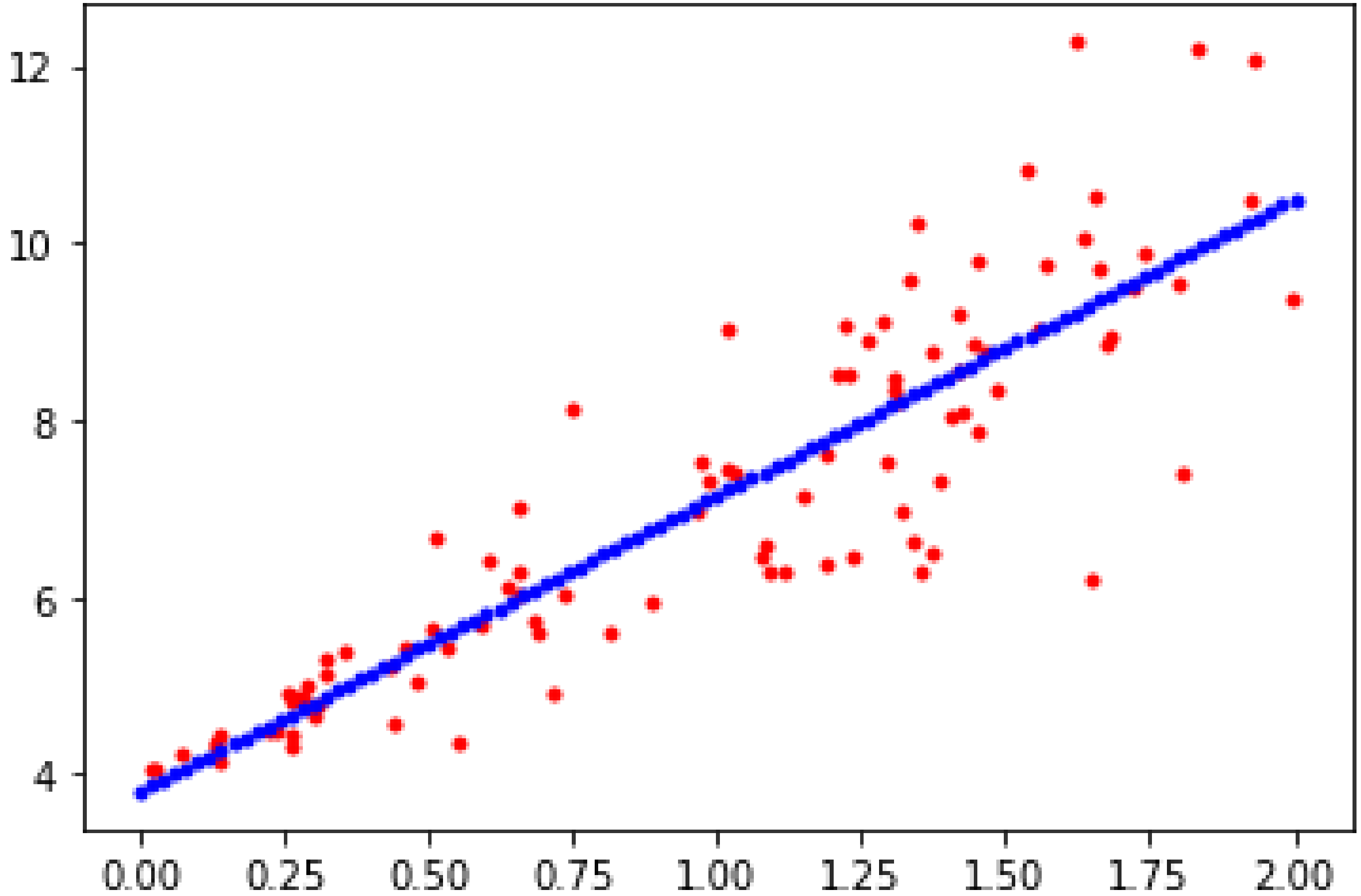
```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt

dfLoad= pd.read_csv('File path', sep="\s+")
xxRaw = np.array(dfLoad.values[:,0])
yyRaw = np.array(dfLoad.values[:,1])
plt.plot(xxRaw, yyRaw, "r.")


N = len(xxRaw)
xx_bias = np.c_[np.ones([100,1]), xxRaw] #Padding 1 as x0 to all samples
yy = yyRaw.reshape(N,1)

#Using Normal Equation:
wOLS = np.linalg.inv(xx_bias.T.dot(xx_bias)).dot(xx_bias.T).dot(yy)
x_sample = np.linspace(0, 2.0, 101)
x_sample_bias = np.c_[np.ones([101,1]), x_sample]
y_pred = wOLS.T.dot(x_sample_bias.T)
x_sample_row = x_sample.reshape(1,101)
plt.plot(x_sample_row, y_pred, "b.-")
plt.show()
```

Eye Check



For statement in Python

- **for** s is used for repeatedly performing certain jobs as:

```
for variable in list, tuple or string:
        job1
        job2
```

Colon

Indent is needed here!

- Try this:

```
scoreArray = [40, 30, 20, 50, 70]
scoreAccumulate = 0;

for score in scoreArray:
    scoreAccumulate = scoreAccumulate + score
    print("You got %d points now" %score)
    print("Your total score is %d" %scoreAccumulate)
```

- You can use the function range() with **for**:

```
for var in range(10):
    print("Your in loop number %d now" %var)
```

Revisit Batch Gradient Descent

- $w_{\text{next}} = w_{\text{present}} - \eta \nabla MSE(\boldsymbol{w})$ ← η : learning rate
  where $\nabla MSE(\boldsymbol{w}) = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)\,\boldsymbol{x}_i$
- We can express $\nabla MSE(\boldsymbol{w})$ utilizing $\boldsymbol{X}$ and represent it w/o Σ :

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

$$= -\frac{2}{N}\{(y_1 - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_1)\boldsymbol{x}_1 + (y_2 - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_2)\boldsymbol{x}_2 + \ \ldots + (y_N - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_N)\boldsymbol{x}_N\}$$

$$= -\frac{2}{N}\left\{\begin{bmatrix} y_1 - \mathbf{w}^{\mathrm{T}}\boldsymbol{x}_1 & y_2 - \mathbf{w}^{\mathrm{T}}\boldsymbol{x}_2 & \cdots & y_N - \mathbf{w}^{\mathrm{T}}\boldsymbol{x}_N \end{bmatrix}\begin{bmatrix} \boldsymbol{x}_1{}^T \\ \boldsymbol{x}_2{}^T \\ \boldsymbol{x}_3{}^T \\ \vdots \\ \boldsymbol{x}_N{}^T \end{bmatrix}\right\}^T = -\frac{2}{N}\{(\boldsymbol{y}^T - \boldsymbol{w}^T\boldsymbol{X}^T)\boldsymbol{X}\}^T$$

$$= -\frac{2}{N}\{\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{y}^T - \boldsymbol{w}^T\boldsymbol{X}^T)^T\} = -\frac{2}{N}[\boldsymbol{X}^{\mathrm{T}}\{\boldsymbol{y} - (\boldsymbol{w}^T\boldsymbol{X}^T)^T\}\ ] = -\frac{2}{N}[\boldsymbol{X}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})]$$

Batch Gradient Descent in Python

- Try this:

```
eta = 0.1   #learning rate
n_iterations = 1000
wGD = np.zeros([2,1])    #initialized to 0

for iteration in range(n_iterations):
    #gradients = - xHeight_bias.dot(wGD)
    gradients = - (2/N)*(xx_bias.T.dot(yy-xx_bias.dot(wGD)))
    #gradients = - (2/N)*(xHeight_bias.T.dot(yWeight-xHeight_bias.dot(wGD)))
    wGD = wGD - eta*gradients
```