Python Quickstart Tutorial

Injung Kim
Handong Global University

Agenda

- Introduction
- Python Core
- Scientific Packages
- Numpy
- Matplotlib

Python

- Easy to learn, rapid development.

  "Life is too short. You need python."

- Powerful
  - Rich built-in features (list, tuple, dict, etc.)
  - numpy, scipy, scikit-learn, pandas, matplotlib, xgboost, etc.

- The most popular language for deep learning.
  - PyTorch, TensorFlow

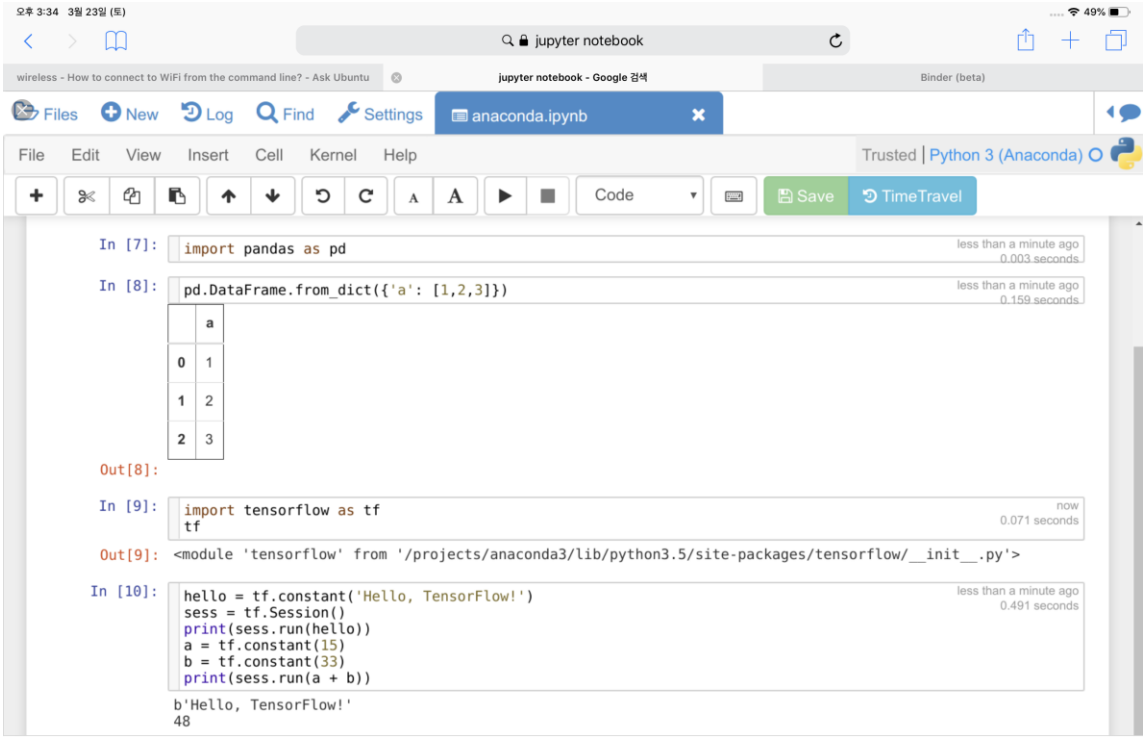
- Plenty of open source codes
  - Deep learning models and algorithms

Python vs. R

- Python
  - General purpose language
  - Easy
  - Designed for
    - Statists, big-data analyst, social scientist
  - Visualization
    - matplotlib, seaborn, plotnine, plot.ly, pyecharts, etc.
  - Deep learning frameworks
- R
  - Data analysis tool
  - Easy
  - Designed by
    - Statists, big-data analyst, social scientist
  - Visualization
    - gglot2

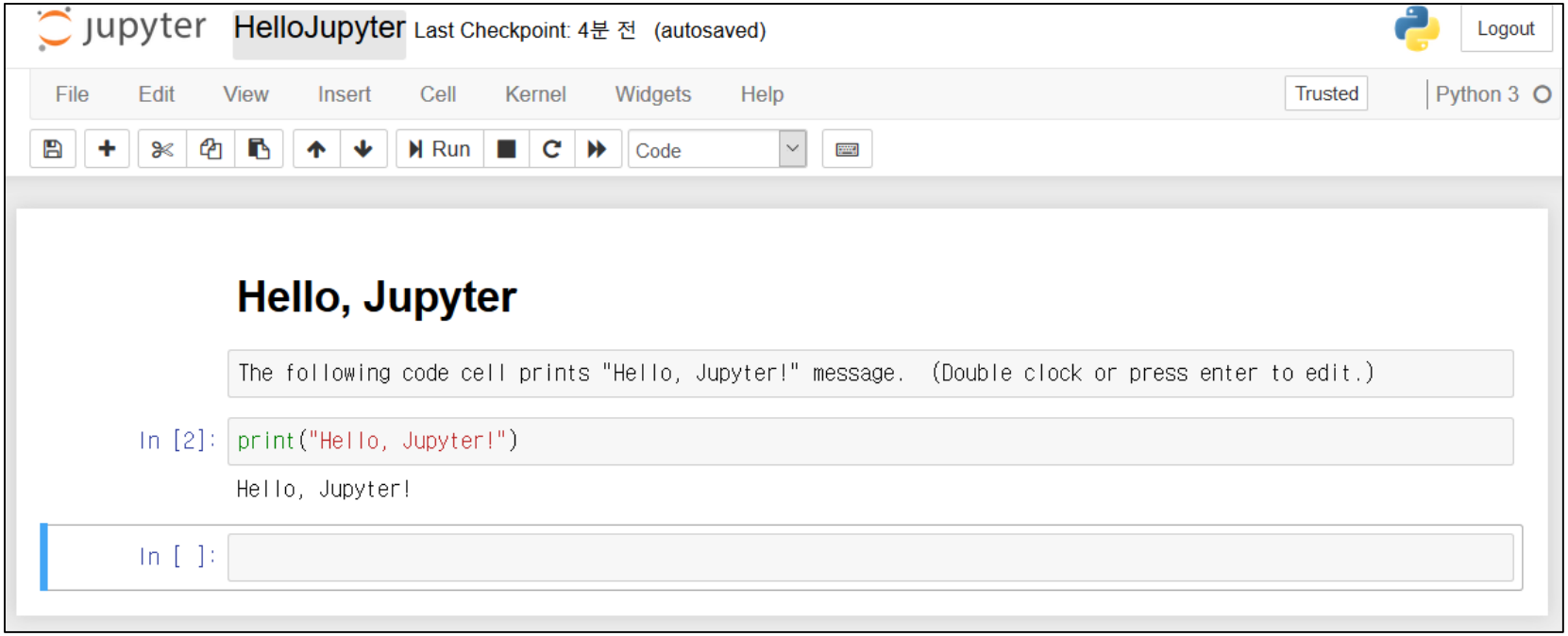
Jupyter Notebook

- Web-based interactive development environment
  - Easy to write and run code
  - Graphics support (matplotlib, etc.)
  - Popular among developers/researchers



Jupyter Notebook Cells

- Heading cells
- Code cells: write and run code
- Text cells: text contents in Markdown
  - Markdown ref.: https://heropy.blog/2017/09/30/markdown/



Environment Setting on Windows PC

- Download and install anaconda
  - https://www.anaconda.com/distribution/
- Create an environment
  - conda create –y –n <env_name> ipykernel anaconda
- Register the new environment to jupyter notebook
  - python –m ipykernel install ––user ––name <env_name> ––display-name “<display_name>”
- Disable SSL verify (optional)
  - conda config ––set ssl_verify False
- Install pytorch and torchvision
  - conda install –y –c pytorch pytorch-cpu (or pytorch)
  - conda install –y –c pytorch torchvision-cpu (or torchvision)
- Install tensorflow (optional)
  - conda install tensorflow (or tensorflow-gpu)

Writing and Running on Jupyter Notebook

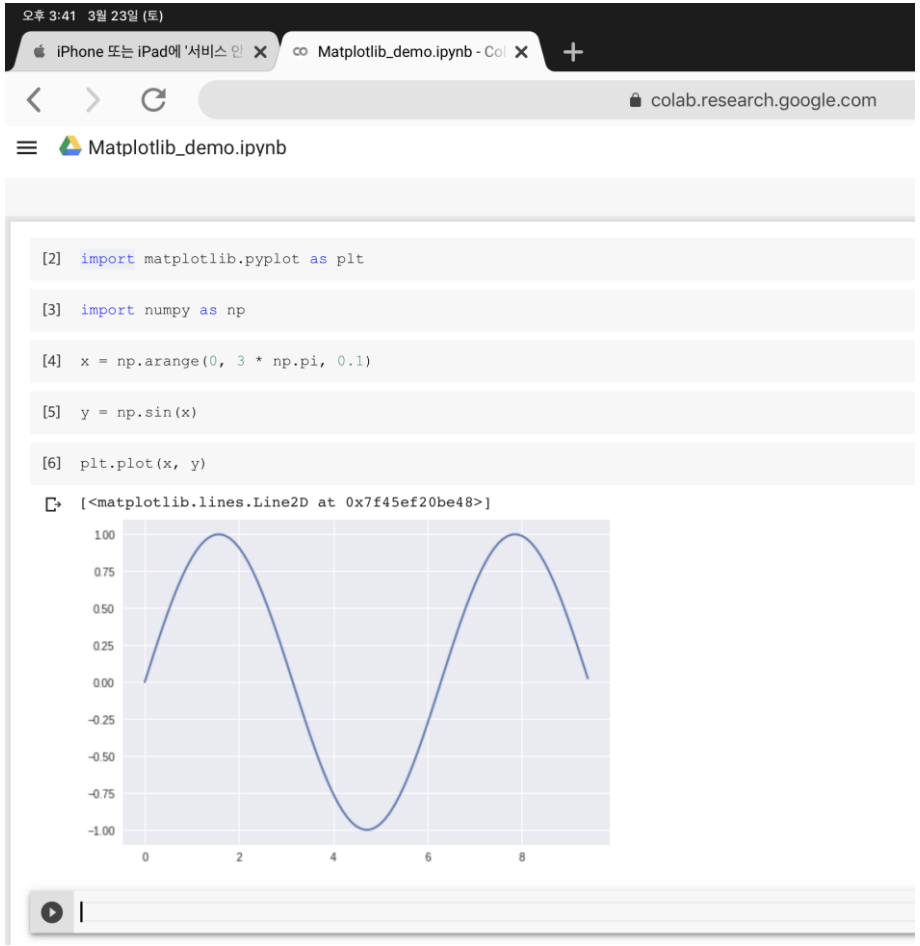
- Activate the new environment
  - conda activate <env_name>

  cf. Terminating current environment: “conda deactivate”
- Run Jupyter notebook in your working directory
  - jupyter notebook
- Create a new notebook
  - New ➔ Python 3 (or <display_name>)
- Write and run code

  Ex) print("Hello, Jupyter!")

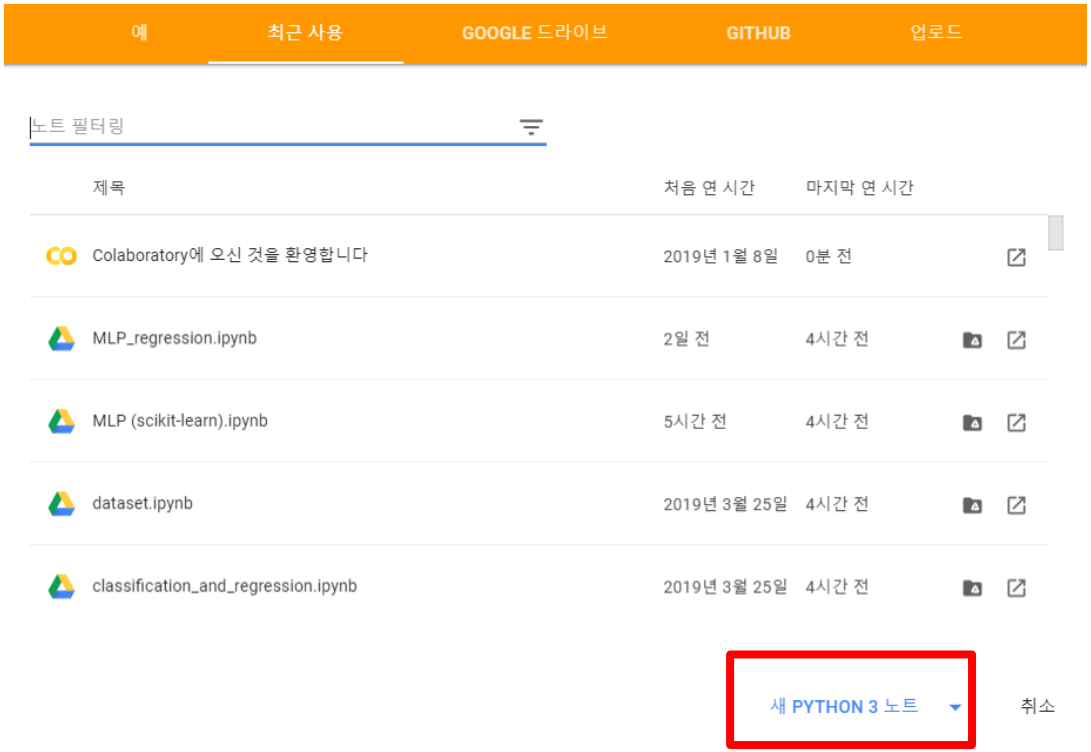
CoLab

- Jupyter-based environment for machine learning education
  - Login by Google account
  - Contains tensorflow, pytorch
- Secure virtual environment
  - Executes on virtual CPU/RAM/disk
  - GPU/TPU supports
- Session lasts only for 12 hours



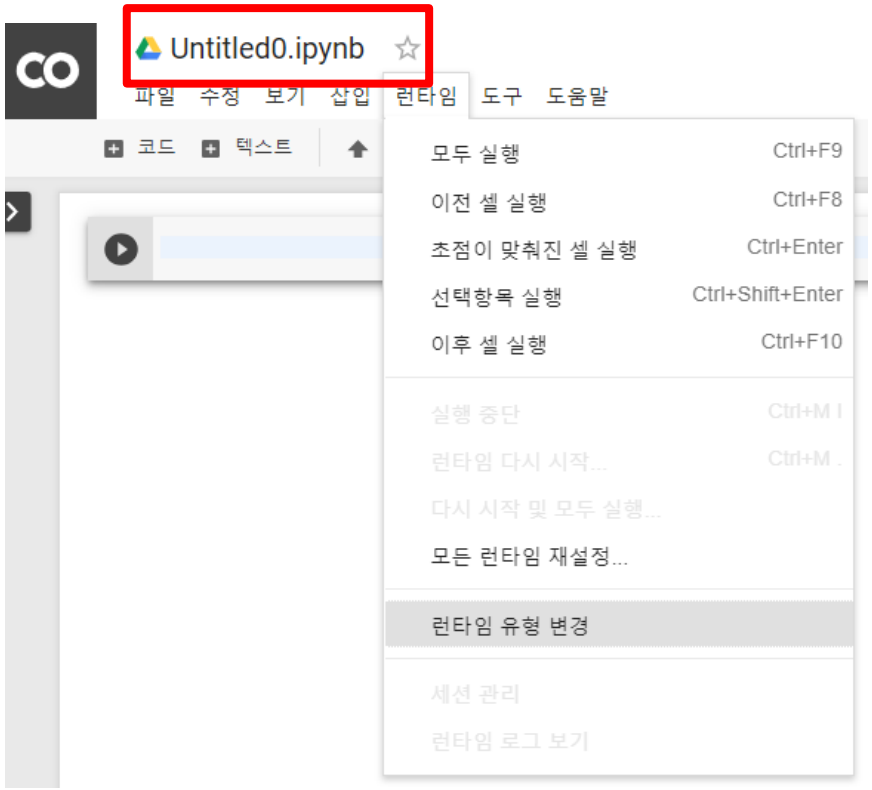
Getting Started with CoLab

- Open Chrome browser and login.
- Visit http://colab.research.google.com
- Click “새 PYTHON 3 노트”



Getting Started with CoLab

- Optionally, adjust settings
  - Rename note
  - Change runtime type to run on GPU



Getting Started with CoLab

- Install packages to use
  - # ! <shell command>

  Ex) ! pip install mglearn

  import mglearn

- Type the following code in the first cell.

  Ex) print("Hello, Jupyter!")

- Run the code (Click ">")
  - CTRL-Enter: Run current cell
  - SHIFT-Enter: Run current cell and move to the next cell
  - ALT-Enter: Run current cell and add a new cell

Loading from GitHub

- Loading codes from GitHub
  - 파일 -> 노트열기 -> GITHUB
  - Type GitHub URL
  - Select a notebook

Python Core

Basic Datatypes

- Integer
  - x = 3
  - print(type(x)) # Prints "<class 'int'>"
  - print(x) # Prints "3"
  - print(x + 1) # Addition; prints "4"
  - print(x * 2) # Multiplication; prints "6"
  - print(x ** 2) # Exponentiation; prints "9“
- Float
  - y = 2.5
  - print(type(y)) # Prints "<class 'float'>"

Basic Datatypes

- Boolean
  - t = True
  - f = False
  - print(type(t)) # Prints "<class 'bool'>"
  - print(t and f) # Logical AND; prints "False“
- String
  - hello = 'hello' # String literals can use single quotes
  - world = "world“ # or double quotes; it does not matter.
  - print(hello) # Prints "hello"
  - print(len(hello)) # String length; prints "5"

print() statement

- print()
  - print("Python is fun.")
  - a = 5
  - print("a =", a)
  - print("a =", a, end='') # does not change line

- Formatted output
  - print("{}and {}".format('spam', 'eggs')) # new style
  - print("{0}and {1} ".format('spam', 'eggs')) # new style
  - print("Text %s"%var_char) # old style

input() statement

- input()

```
print('Enter your name:')
x = input()                          # reads a string
print('Hello, ', x)

print('Enter your age:')
age = int(input())                   # reads an integer
print('Your age is ', age)
```

if-statement

- if-statement

```
if <expression>:
    <statement1>          # indented statement block
    <statement2>
...
```

Ex) Computing luggage charge

```
weight = float(input("How many pounds does your suitcase weight?"))
if weight > 50:
    print("There is a $25 charge for luggage that heavy.")
print("Thank you for your business.")
```

while-statement

- while-statement

```
while <expression>:
    <statement1>          # indented statement block
    <statement2>
    ...
```

Ex) prints "Hello Geek" three times

```
count = 0
while count < 3:
    print("Hello Geek")
    count = count+1
```

for-statement

- for-statement

  for <var> in range(<# of iterations>):
  <statement1> # indented statement block
  <statement2>
  …

  Ex) prints "Hello Geek" three times

```
for i in range(3):   # range(3) is similar to [0, 1, 2]
    print("Hello Geek")
```

  for <var> in range(<start>, <end>, <step>):
  <statement1> # indented statement block
  <statement2>
  …

  Ex) specifying start, end and step

```
for i in range(30, 100, 10):  # repeats on i for 30, 40, …, 90
    print(“i = “, i)   # prints i = 30, 40, …, 90
```

Lists

- Similar to array but resizable and can contain elements of different types
  - xs = [3, 1, 2] # Creates a list
  - print(xs, xs[2]) # Prints "[3, 1, 2] 2"
  - print(xs[-1]) # Negative indices count from the end of the list; prints "2“
  - xs[2] = 'foo' # Lists can contain elements of different types
  - print(xs) # Prints "[3, 1, 'foo']"
  - xs.append('bar') # Add a new element to the end of the list
  - print(xs) # Prints "[3, 1, 'foo', 'bar']"
  - x = xs.pop() # Remove and return the last element of the list
  - print(x, xs) # Prints "bar [3, 1, 'foo']"

List Slicing

- Slicing
  - nums = list(range(5)) # range(5) creates [0, 1, 2, 3, 4]
  - print(nums) # Prints "[0, 1, 2, 3, 4]"
  - print(nums[2:4]) # Prints "[2, 3]"
  - print(nums[2:]) # Prints "[2, 3, 4]"
  - print(nums[:2]) # Prints "[0, 1]"
  - print(nums[:]) # Prints "[0, 1, 2, 3, 4]"
  - print(nums[:-1]) # Prints "[0, 1, 2, 3]"
  - nums[2:4] = [8, 9] # Assign a new sublist to a slice
  - print(nums) # Prints "[0, 1, 8, 9, 4]"

Loops with Lists

- Loop with a list

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

- Loop with a list using enumerate()

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('%d: %s' % (idx + 1, animal))
# Prints "#1: cat", "#2: dog", "#3: monkey", each on its own
line
```

List Comprehension

- Generating a list

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x**2)        # x** 2 means x^2
print(squares)   # Prints [0, 1, 4, 9, 16]
```

- Creating a list using comprehension

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)  # Prints "[0, 4, 16]"
```

Tuples

- A tuple is an (immutable) ordered list of values

```
t = (5, 6)                # create a tuple
print(type(t))            # prints "<class 'tuple'>"

d = {(x, x + 1): x for x in range(10)}  # Create a dictionary with tuple keys
# {(0, 1): 0, (1, 2): 1, (2, 3): 2, (3, 4): 3, (4, 5): 4, (5, 6): 5, (6, 7): 6, (7, 8): 7, (8, 9): 8, (9, 10): 9}
print(d[t])               # prints "5"
print(d[(1, 2)])          # prints "1"
```

Dictionaries

- A dictionary stores (key, value) pairs

```
d = {'cat':'cute', 'dog':'furry'}   # creates a new dictionary
print(d['cat'])       # prints "cute"
print('cat' in d)     # prints "True"
d['fish'] = 'wet'     # set an entry in a dictionary
print(d['fish'])      # prints "wet“
print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
del d['fish']         # remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

Loop with Dictionary

- Loop with dictionary

```
d = {'person':2, 'cat':4, 'spider':8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs“
```

- Loop with dictionary using items()

```
d = {'person':2, 'cat':4, 'spider':8}
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))
# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

Dictionary Comprehensions

- Generating a dictionary
  - nums = [0, 1, 2, 3, 4]
  - even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
  - print(even_num_to_square) # Prints "{0: 0, 2: 4, 4: 16}"

Functions

- Defining a function

```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'
```

- Calling function

```
for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

Functions with Keyword Arguments

- Defining a function

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)
```

- Calling function

```
hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

Classes

- Defining a class

```
class Greeter(object):
    def __init__(self, name):        # Constructor
        self.name = name  # Create an instance variable

    def greet(self, loud=False):  # Instance method
        if loud:
            print('HELLO, %s!' % self.name.upper())
        else:
            print('Hello, %s' % self.name)
```

- Declaring and using class variables

```
g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

Scientific Packages

Packages for Machine Learning

- numpy (www.numpy.org)
  - Multi-dimensional array, vector/matrix operation, FFT, random number capabilities
- scikit-learn (scikit-learn.org)
  - Machine learning package in Python
  - Classic algorithms for classification, regression, clustering, etc.
- xgboost (xgboost.readthedocs.io)
  - Gradient boosting library
- pydnn (pydnn.readthedocs.io)
  - GPU neural network library for deep learning in Python
- matplotlib (matplotlib.org)
  - 2D plotting library (graph, images, etc.)
- pandas (pandas.pydata.org)
  - Python data analysis library (DataFrame, Series)
- mglearn (optional)
  - Machine learning education package (sample datasets and algorithms)

Packages for Machine Learning

- pytorch (tensorflow.org)
  - A rising deep learning framework
- torchvision
  - Computer vision package based on pytorch (datasets, models)
- tensorflow (tensorflow.org)
  - Currently, most popular deep learning framework
- keras (keras.io)
  - High-level neural networks API, written in Python
  - Runs on top of TensorFlow, CNTK, or Theano
- tensorflow-slim
  - Lightweight library for defining, training and evaluating complex models in tensorflow

Importing Packages

- Importing basic science packages
  - import numpy as np
  - import sklearn as sk
  - import xgboost as xgb
  - import matplotlib.pyplot as plt
  - import mglearn #optional
- Importing pytorch
  - import torch
  - import torchvision
  - import torch.nn as nn
- Importing tensorflow
  - import tensorflow as tf
  - from tensorflow import keras
  - import tensorflow.contrib.slim as slim

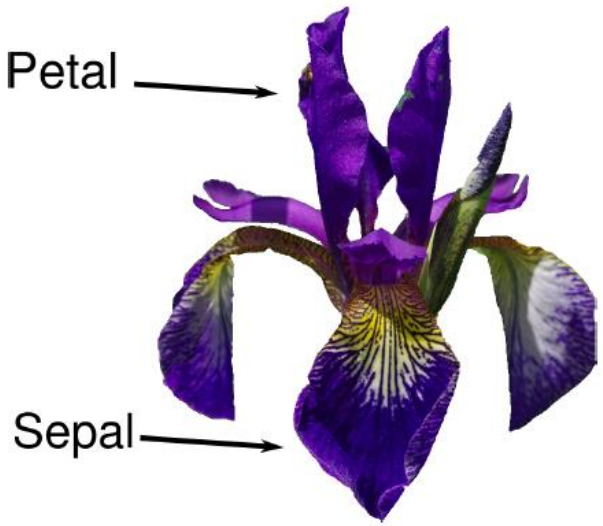
scikit-learn

- Machine learning package in Python
  - Classification
    - k-NN, decision trees, random forests, MLP, SVM, …
  - Regression
    - linear, Ridge, Lasso, ElasticNet, SVR, …
  - Clustering
    - k-Means, spectral, …
  - Dimensionality reduction
    - PCA, LDA, …
- Benchmark datasets
  - See https://scikit-learn.org/stable/datasets/index.html

Example) Iris Dataset in scikit-learn

- Finding Iris species from size of petal and sepal
  - Target classes: { 'setosa' 'versicolor' 'virginica' }
    - 0: setosa, 1: versicolor, 2: virginica
  - Input features: length/width of petal(꽃잎) and Sepal(꽃받침)
    - ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

```
[[5.9 3. 4.2 1.5]
 [5.8 2.6 4. 1.2]
 [6.8 3. 5.5 2.1]
 [4.7 3.2 1.3 0.2]
 [6.9 3.1 5.1 2.3]
 [5. 3.5 1.6 0.6]
 [5.4 3.7 1.5 0.2]
 [5. 2. 3.5 1. ]
... ]
```



Data Format for scikit-learn

- Load or create dataset

```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

Input data (Rank 2)

```
iris_dataset.data ((150, 4)) =
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

scikit-learn Model

Target data (Rank 1)

```
iris_dataset.target (150 dim) =
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

- Split training/test dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris_dataset.data,
iris_dataset.target, random_state=0)
```

pandas

- **Pandas:** a Python data analysis package
  - Provides fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive.
  - Contains DataFrame(2D), Series(1D) classes

- 10 Minutes to pandas
  - https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html

- Pytorch data loading and processing tutorial
  - https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

pandas

- DataFrame

  import numpy as np
  import pandas as pd

  Ex) Reading a csv file
  df1 = pd.read_csv(<filename>)

  Ex) Accessing a column or row
  df1['A'] # column 'A'
  df1.iloc[2] # 2nd row

  Ex) Quick statistic summary
  df1.describe()

  Ex) Converting to numpy ndarray
  array1 = df1.to_numpy()

```
In [144]: pd.read_csv('foo.csv')
Out[144]:
     Unnamed: 0          A          B          C          D
0    2000-01-01   0.266457  -0.399641  -0.219582   1.186860
1    2000-01-02  -1.170732  -0.345873   1.653061  -0.282953
2    2000-01-03  -1.734933   0.530468   2.060811  -0.515536
3    2000-01-04  -1.555121   1.452620   0.239859  -1.156896
4    2000-01-05   0.578117   0.511371   0.103552  -2.428202
5    2000-01-06   0.478344   0.449933  -0.741620  -1.962409
6    2000-01-07   1.235339  -0.091757  -1.543861  -1.084753
..          ...        ...        ...        ...        ...
993  2002-09-20 -10.628548  -9.153563  -7.883146  28.313940
994  2002-09-21 -10.390377  -8.727491  -6.399645  30.914107
995  2002-09-22  -8.985362  -8.485624  -4.669462  31.367740
996  2002-09-23  -9.558560  -8.781216  -4.499815  30.518439
997  2002-09-24  -9.902058  -9.340490  -4.386639  30.105593
998  2002-09-25 -10.216020  -9.480682  -3.933802  29.758560
999  2002-09-26 -11.856774 -10.671012  -3.216025  29.369368

[1000 rows x 5 columns]
```

```
df1.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.393515	0.038337	-0.063684	-0.334082
std	1.528779	0.919736	1.294443	0.911405
min	-2.582301	-0.899616	-1.580073	-1.557019
25%	-0.969763	-0.685161	-1.129615	-0.963755
50%	-0.577698	-0.105634	0.102901	-0.300585
75%	0.227633	0.542405	0.590073	0.470532
max	1.980984	1.460824	1.780157	0.610938

Using scikit-learn

- Import package

  Import sklearn as sk

  from sklearn.neighbors import KNeighborsClassifier

- Create classifier/regressor object

  knn = KNeighborsClassifier(n_neighbors=3)

- Train to a training dataset

  knn.fit(X_train,y_train)

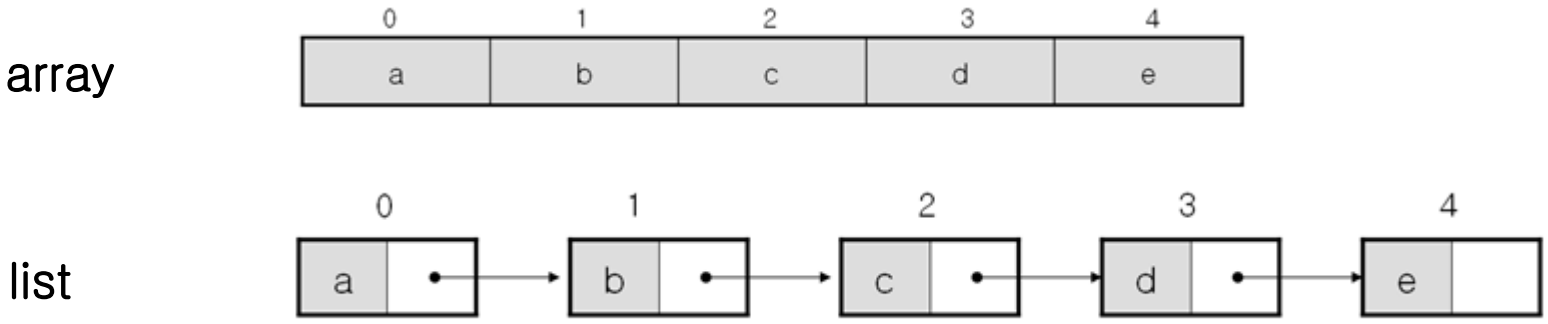
- Use it

  prediction = knn.predict(X_new)

Numpy

Numpy Arrays

- numpy.ndarray: n-dim array of the same type
  - Elements are of the same type
  - Indexed by integers or tuples
  - Efficient for random access
  - Widely used to represent vectors and matrices



Numpy Arrays

- Creating and using array

```
a = np.array([1, 2, 3])      # Create an array [1, 2, 3]
print(type(a))               # Prints "<class 'numpy.ndarray'>"
print(a.shape)               # Prints "(3,)"
print(a[0], a[1], a[2])      # Prints "1 2 3"
a[0] = 5                     # Change an element of the array
print(a)                     # Prints "[5, 2, 3]"
```

Numpy Arrays

- Creating and using 2D array

```
b = np.array([[1,2,3],[4,5,6]])    # Create a 2D array
                                   #  [[1, 2, 3],
                                   #    [4, 5, 6]]
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Creating Array

- Functions to create arrays

```
a = np.zeros((2,3))      # Create an array of all zeros
print(a)                 # Prints "[[ 0.  0.  0.]
                         #         [ 0.  0.  0.]]"

b = np.ones((1,2))       # Create an array of all ones
print(b)                 # Prints "[[ 1.  1.]]"

c = np.full((2,3), 7)    # Create a constant array
print(c)                 # Prints "[[ 7.  7.  7.]
                         #         [ 7.  7.  7.]]"

d = np.eye(2)            # Create a 2x2 identity matrix
print(d)                 # Prints "[[ 1.  0.]
                         #         [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                 # Might print "[[ 0.91940167  0.08143941]
                         #               [ 0.68744134  0.87236687]]"
```

Array Slicing

```
import numpy as np

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

b = a[:2, 1:3]
# [[2 3]
#  [6 7]]

# A slice of an array is a view into the same data,
# so modifying it will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

Array Slicing

```
import numpy as np

a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]

row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"

col_r1 = a[:, 1]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
col_r2 = a[:, 1:2]
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #         [ 6]
                             #         [10]] (3, 1)"
```

Broadcasting

- Using the smaller array multiple times to perform some operation on the larger array.

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v     # Add v to each row of x using broadcasting
print(y)      # Prints "[[ 2  2  4]
              #         [ 5  5  7]
              #         [ 8  8 10]
              #         [11 11 13]]"
```

Numpy Functions: reshape()

- Creating a 1D array
  - a = np.arange(0, 20)
  - print("a.shape = ", a.shape)

    a.shape = (20,)
  - print(a)

    [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

- reshape()
  - b = np.reshape(a, (4, 5))
  - print("b.shape = ", b.shape)

    b.shape = (4, 5)
  - print(b)

    [[0 1 2 3 4] [5 6 7 8 9] [10 11 12 13 14] [15 16 17 18 19]]

Numpy Functions: expand_dims()

- expand_dims()
  - parameter axis specifies the new axis

  - # the new dimension becomes axis 0
  - c = np.expand_dims(a, axis = 0)
  - print("c.shape = ", c.shape)
    c.shape = (1, 20)
  - print(c)
    [[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]]

  - # the new dimension becomes axis 1
  - d = np.expand_dims(a, axis = 1)
  - print("d.shape = ", d.shape)
    d.shape = (20, 1)
  - print(d)
    [[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13]
    [14] [15] [16] [17] [18] [19]]

Numpy Functions: squeeze()

- squeeze() removes single dimensional entries
  - print("c.shape = ", c.shape)
    c.shape = (1, 20)
  - f = np.squeeze(c)
  - print("f.shape = ", f.shape)
    f.shape = (20,)
  - print(f)
    [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

  - print("d.shape = ", d.shape)
    d.shape = (20, 1)
  - g = np.squeeze(d)
  - print("g.shape = ", g.shape)
    g.shape = (20,)
  - print(g)
    [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

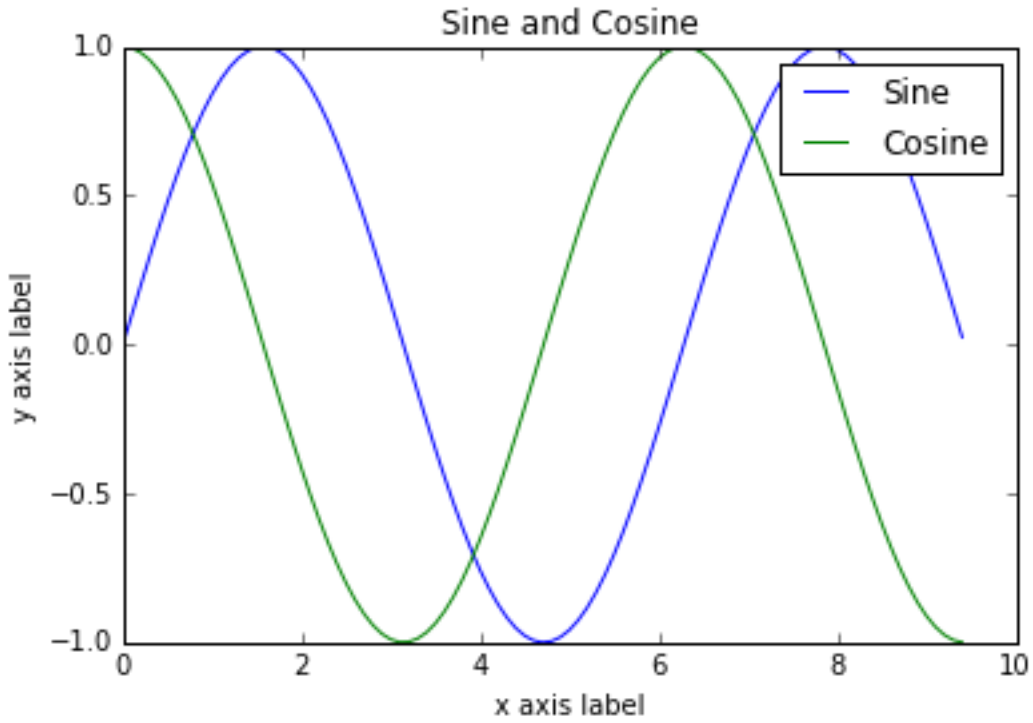
Matplotlib

Plotting

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```

- For detail, https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

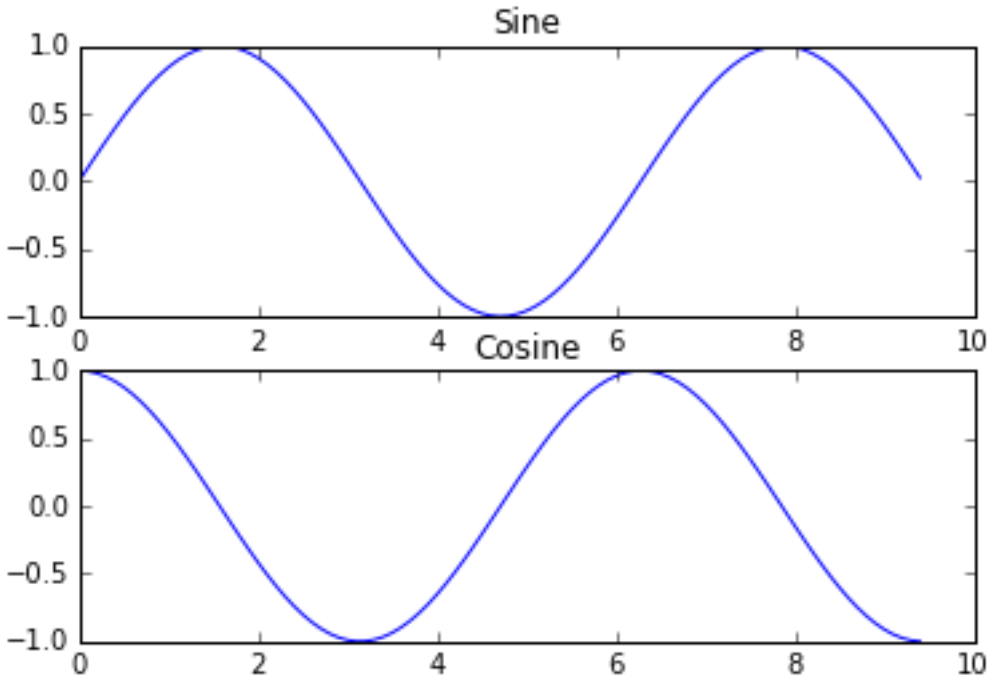
Plotting


Sine and Cosine
Sine
Cosine
1.0
0.5
0.0
−0.5
−1.0
0
2
4
6
8
10
y axis label
x axis label

Subplots

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```

Subplots



Images

```
import numpy as np
from scipy.misc import imread, imresize
import matplotlib.pyplot as plt

img = imread('assets/cat.jpg')
img_tinted = img * [1, 0.95, 0.9]

# Show the original image
plt.subplot(1, 2, 1)
plt.imshow(img)

# Show the tinted image
plt.subplot(1, 2, 2)

# A slight gotcha with imshow is that it might give strange results
# if presented with data that is not uint8. To work around this, we
# explicitly cast the image to uint8 before displaying it.
plt.imshow(np.uint8(img_tinted))
plt.show()
```

Images


0
50
100
150
200
250
300
350
400
0
50
100
150
200
0
50
100
150
200
250
300
350
400
0
50
100
150
200

Q&A

Thank you
for your attention!