Data Science 입문 with Python

2019.6 정용진

(*) Reference

- Steven S. Skiena, The Data Science Manual, Springer, 2017
- John Canny, Introduction to Data Science (lecture note), UC Berkeley, 2014
- Wes Mckinney, Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Ipython, O'Reilly, 2012
- 김화종, 데이터 사이언스 개론, 홍릉과학출판사, 2014
- 조재호, 모두의 딥 러닝, 길벗, 2017
- Many Internet sites

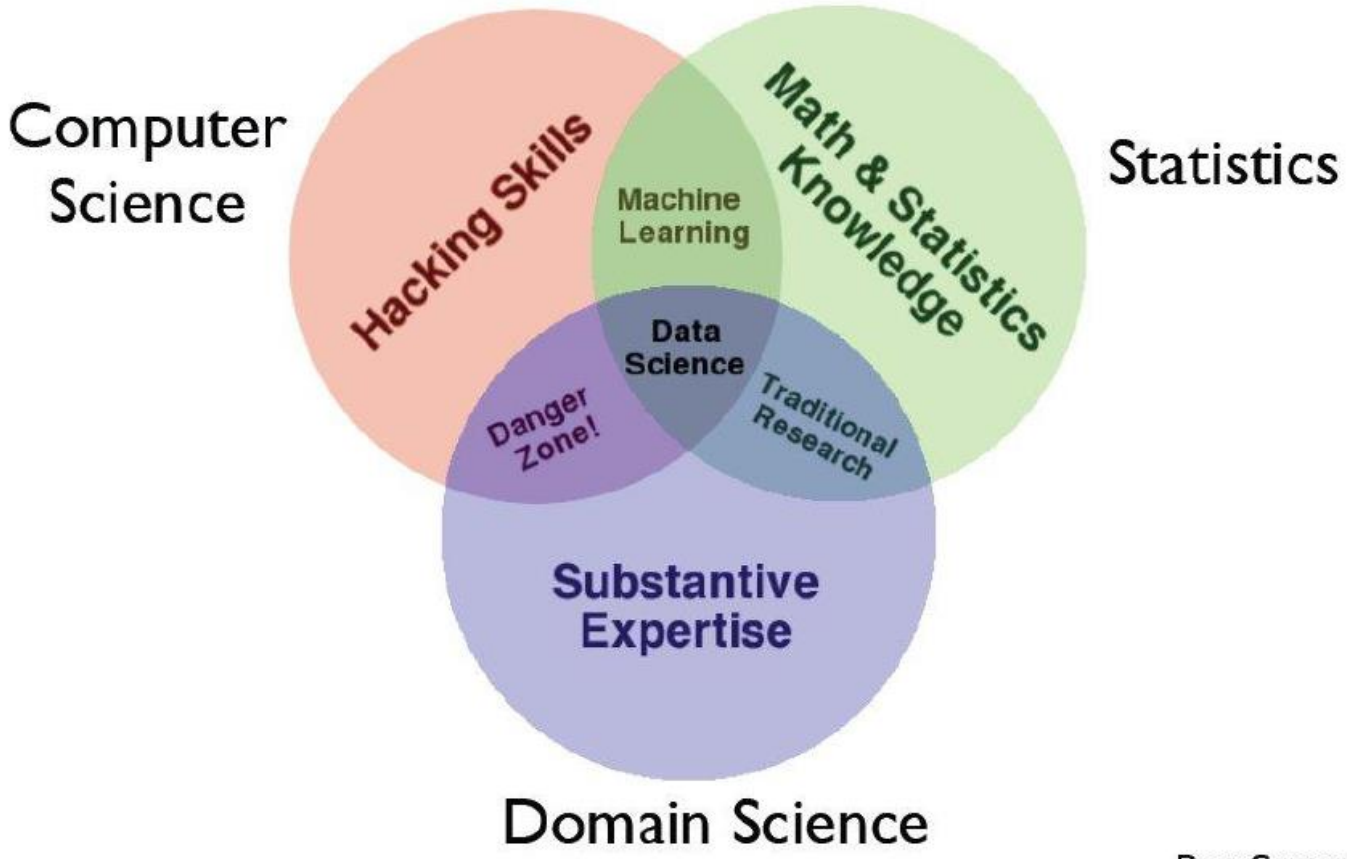
What will be the core job of the future?

- IT 전문매체 '매셔블(Mashable)' 이 소개하는 고액연봉 IT 분야 (2019.3)
  - Cyber Security Engineer
  - AI, Machine Learning Engineer
  - Full-stack Developer
  - Data Scientist
  - Python Developer
  - Java Developer
  - Cloud Engineer
  - Scrum Master
  - DevOps Engineer
  - JavaScript Developer

What is Data Science?

- Not yet well defined, but incorporates elements of
  - Exploratory Data Analysis and Visualization
  - Machine Learning and Statistics
  - High-Performance Computing Technologies for dealing with scale
- **Data Science**
  - Software Programming -> Data mining, Database
  - Statistics/mathematical modeling -> Machine Learning, Scientific Computing
  - Domain Knowledge -> Data driven business analytics
- Main applications
  - E-commerce,
  - Social media,
  - IoT,
  - Biometrics,
  - Sharing Economy, and many more

What is Data Science? – One definition



Drew Conway

What is Data Science?

❖ Contrast to Databases

	Databases	Data Science
Data Values	"Precious"	"Cheap"
Data Volume	Modest	Massive
Examples	Bank Records, Personal Records, Census, Medical Records	On line clicks, GPS logs, Tweets, Web surfing, building censor readings
Structured	Strongly (Schema)	Weak or None (Text)
Realizations	SQL	No SQL Python, R, TensorFlow, Keras
	Querying the Past	Querying the Future

What is Data Science?

❖ Good data scientist must first learn to think like a real scientist.

Computer Science	Real Science
Algorithm is the first! Data is just stuff to test algorithm performance. Mostly use random data to test algorithm performance.	Appreciate and respect Data.
Try to build their own clean and organized virtual world. -> Everything is either TRUE or FALSE.	Try to understand the complicated and messy natural world. -> Nothing is ever completely true or false.
Algorithm-Driven	Data-Driven
Try to invent rather than discover.	Try to discover things.
For the result, they care what a number is.	Care what it means.
Software Developers are hired to produce code.	Hired to produce insights.
Genius (finding the right answer)	Wisdom (avoiding the wrong answers)

What is Data Science?

❖ Contrast to Machine Learning

Machine Learning	Data Science
Develop new (individual) models	Explore many models, build and tune hybrids.
Prove mathematical properties of model.	Understand empirical properties of models.
Improve/validate on a few, relatively clean, small datasets.	Develop/use tools that can handle massive datasets
Publish a paper	Take action!

What is Anaconda?

- What is Anaconda?
  - Very popular Python development platform package for mathematics and science, and specially for data science and machine learning
  - Includes useful packages like SciPy, NumPy, Matplotlib, Pandas, etc.
- Why Anaconda?
  - > 400 packages available, 150 automatically installed
  - Free, open source
  - Support all major platforms
  - Very reliable and easy to use
  - Scale up to professional and commercial use (with fee)

Anaconda Overview

- Installation
  - Download Anaconda from https://www.anaconda.com/download/
    - Select Python 3.7 version (for Windows)
- Where to start?
  - Command line
  - Launcher: Jupyter notebook, Spyder, Ipython console
- Relevant libraries
  - Pandas (http://paandas.pydata.org)
  - Numpy (http://www.numpy.org)
  - SciPy (http://www.scipy.org)
  - Matplotlib (http://matplotlib.org)

Anaconda Packages

- Over 150 packages are **automatically installed with Anaconda**
- Over 250 additional open source packages can be individually installed from the anaconda repository at the command line, by using the "% conda install" command.
- Thousands of other packages are available from Anaconda.org site
- Others can be downloaded using "% pip install" command that is included and installed with Anaconda.
- You can also make your own custom packages using the "% conda build" command, and upload them to Anaocnda.org or other repositories.

Essential Python Modules

package	Modules with description	
NumPy		Foundational Package for scientific computing Multidimensional array objects and computational functions
pandas		Rich data structures and functions to facilitate data processing and analysis: DataFrame
SciPy		Collection of packages for performing linear algebra, statistics, optimization, and more
matplotlib	Pyplot	Data visualization
sklearn (scikit-learn)	linear_model, cluster metrics model_selection	LinearRegression, SGDVassifier, LogisticRegression Kmeans accuracy_score, classification_report, confusion_matrix roc_curve, auc train_test_split

What is Python Language?

- Completely open source, started in early 1990
- **Script language (interpreter)** , i.e. no compiler
  - Development environment = execution environment
  - Converted to (platform-independent) bytecode (and Python Virtual Machine(PVM) interprets and executes it – slow)
- Very portable, mostly runnable on all supported platforms
- Object-oriented and Functional
- Large standard libraries with huge set of external modules

Python Scripts

- Use any editor to create a Python script, say, *myscript.py*
- No compilation needed
  - Python script is **interpreted**. More precisely, it is converted to byte code (.pyc), and then executed.
  - Python is considered a mixture of complied and interpreted.
- Run script from command line
  - > python myscript.py
  - (ex) calculator, running scripts, test environment
- Run script in Notebook or IDE
  - **Jupyter** or Spyder, or other IDE
  - (ex) work processes (ideal for data processing and analysis), documentation, teaching and presentation

IPython (Interactive Python)

- Established in 2001
- Aims to extend Python's interactive capabilities beyond those shipped by default with the language
- Major features
  - Access to all session state: Ipython stores a session's inputs and outputs into a pair of numbered tables called In and Out. All outputs are also accessible as _N, where N is the number of results.
  - IPython offers a set of control commands (or magic commands)
  - Operating system access
  - Dynamic introspection and help
  - Access to program execution

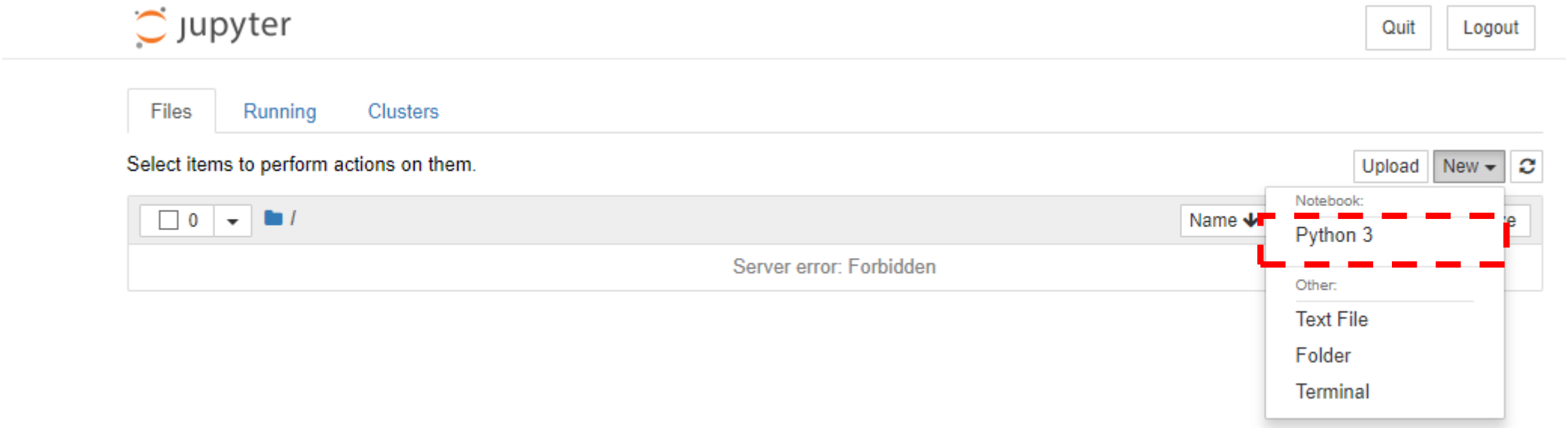
(Ipython or Jupyter) Notebook

- Convenient web-based executable script files
  - Interactive code development
  - Cell-wise execution
  - No reloading of script (.py) files necessary
  - Easy to share
  - Excellent teaching tool
- Project Jupyter was born out of the IPython Project in 2014
  - Jupyter can support (or be interfaced with) other languages (Ruby, R, Julia, etc.)
- Requires Google Chrome or Mozilla Firefox
- On-line examples
  - https://nbviewer.jupyter.org

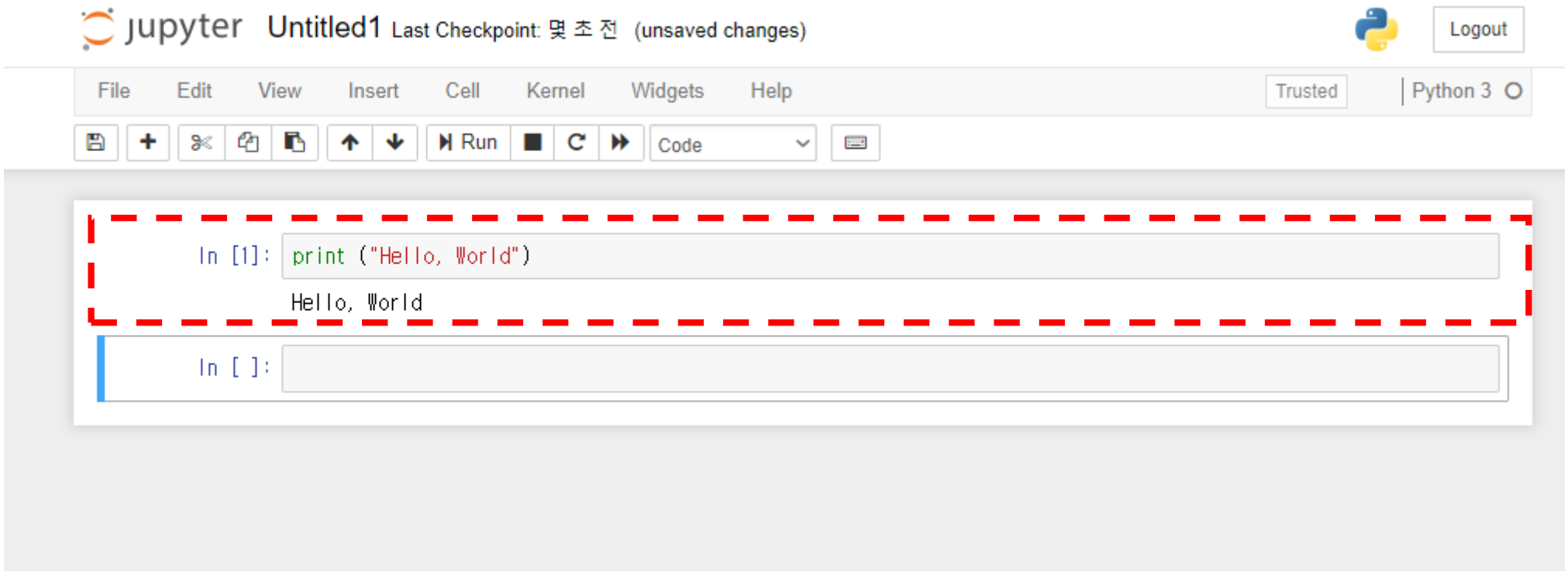
Jupyter notebook

- To start, from command line, enter “jupyter notebook” or click the icon “Jupyter Notebook” from startup menu and set the type as “Python 3”.
- Create or Open a new notebook, from the editor window
  - *File> New Notebook* or *File> Open*
- To add new contents, first select content type, then insert a cell and input the material
  - Markdown, code, heading, or Raw NBConvert
- To edit the contents, use the Edit command to cut/copy/paste
- To control code execution, use the cell commands

Jupyter notebook Python 3



Jupyter notebook Python 3



- In place: Ctrl + Enter
- To execute cell and move to next cell: Shift + Enter
  - Create new cell if necessary
- To execute and insert new cell: Alt + Enter

Convenient Features

- Syntax Highlighting
  - Automatically highlights standard functions (e.g. **for**, **range**), keywords (e.g. **in**, **and**), special characters (e.g. #)
- Auto Indent
  - Primarily driven by the colon operator (:)
  - Automatically indents blocks after if, for, while, etc.
  - Helps with debugging
- Parentheses Matching
  - Helps with debugging

Tab Completion

- Type part of the name and then press <Tab> to see options
  - The more you type, the more specific the matches are
  - Auto-completes if there's only one option
  - Use to learn about useful attributes and methods
- Us as much as possible… saves time !
  - Variable names
  - Function names, keywords, and descriptions
  - File directions and names
  - Objects, attributes, and methods
    - Need to assign to variables first in this case
- Insert **?** after a variable, function, or object to find out more information (e.g. sum?)

Keyboard Shortcuts - Jupyter

❖ Command mode (press ESC to enable)

In command mode	
Shift-Enter	run cell, select below
Ctrl-Enter	run selected cells
Alt-Enter	run cell and insert below
a/b	insert cell above/below
x/c	cut selected cells / copy selected cells
Shift-v / v	paste cells above/below
Shift-m	merge selected cells, or current cell with the cell below if only one cell is selected

In command mode	
l	toggle line numbers
o	toggle output of selected cells
h	show keyboard shortcuts
Shift-Space	scroll notebook up
Space	scroll notebook down
Window-/	toggle comment

In edit mode (press Enter)	
Ctrl-Shift-Minus	Split cell at cursor

Notebook Cell Types

- **Code cells**
  - Edit and execute cells inline, generates output as text, figures, HTML tables
  - Syntax highlighting, tab completion, introspection
  - Default for inserted cells
- **Markdown cells**
  - Rich text input, including HTML and LaTex
  - Cell replaced by text output when executed (**Documents**)
- Raw text cells
  - Executed as input (no formatting)
  - Cell remains in place
- Heading cells
  - Levels 1 through 6, similar to Microsoft Word
  - Can be used to generate Table of Contents

Python Language

- Objects, attributes, and methods
- Functions vs. object methods
- Object references
- Mutable and immutable objects

Data types

- Basic Data types
  - Int, float, boolean(bool), string(str)
  - List - mutable
  - Tuple – can read, but can not overwrite (to make computation fast), immutable
  - Dictionary – only access by keys
- (NumPy) **array** (formally called **ndarray**)
  - Although Core Python has an array data structure, but it's not efficient and useful. Instead we use "NumPy array" which is referred to by "array".
  - Similar to list, but all the elements are of the same type (int, float, Boolean, string, or other object)

Objects, attributes, and methods

- **Everything in Python is an object**.
  - Scalars, sequences, dictionaries, functions, DataFrames, modules, and more
- Each type of object has a set of
  - **Attributes**: Characteristics of the object
  - **Methods**: Functions that operate on the object (and possibly other objects)
- Atrtributes and methods are accessible by:
  - `obj.attr_name` or `getattr(obj, 'attr_name')`
  - `obj.method_name()`

Functions vs. Object Methods

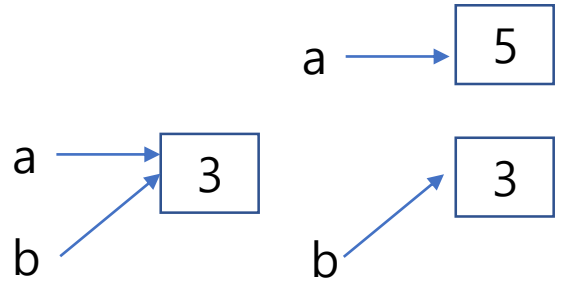
- Functions and object methods are essentially the same…
  - One or more bundled steps performed on some input
  - In some cases, there will be a function and an object method that do the same thing (e.g., sum)
- …BUT, they differ in how they are used
  - Functions are called on zero or more objects and return result(s) that can be assigned to a variable
  - Object methods are called by an object and can either update the calling object or return results

Mutable and Immutable Objects

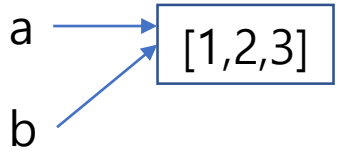
- Mutable Objects
  - Can be modified via assignment or a function/method
  - Lists, dictionaries, ndarrays, class instances
- Immutable Objects
  - Can not be modified
  - Strings, tutples, int, float, boolean

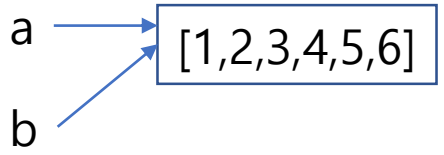
Mutable and Immutable (examples)

```
In [384]: a = 3                              # immutable variable
          b = a
          id(a), id(b)

Out[384]: (1681633568, 1681633568)

In [385]: a += 2                             # since it is immutable, a is newly created
          a,b, id(a), id(b)

Out[385]: (5, 3, 1681633632, 1681633568)
```




```
In [389]: # more examples
          a =[1,2,3]                         # when assigning a variable, you are assigning the reference.
          b = a                              # id(x) returns memory address of the object
          id(a),id(b)

Out[389]: (1559399868552, 1559399868552)

In [390]: a +=[4,5,6]                        # same id (interpreted as a.append([4,5,6]))
          a,b, id(a), id(b)                  # note that a = a + [4,5,6] will create a new object

Out[390]: ([1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6], 1559399868552, 1559399868552)
```





Object References

- Call-by-value? or Call-by-reference?

```
>>> def test(a):
        a = 2

>>> a = 1
>>> test(a); a
1
```

```
>>> def test2(a):
        a.append('world.')

>>> b = 'Hello'
>>> test2(b); b
['Hello', 'World.']
```

```
>>> a = 10
>>> b = a
>>> a += 100
>>> a, b
(110, 10)
>>> id(a), id(b)
(14073…7824, 1407…624)
```

```
>>> a = [1,2,3]
>>> b = a
>>> a += [4,5,6]
>>> a,b
([1,2,3,4,5,6],[1,2,3,4,5,6])
>>> id(a), id(b)
(225009…832, 225009…832)
```

Object References (2)

- **Call-by-Object** (or call-by-Object Reference or call-by-sharing)
  - If you pass **immutable arguments like integers, strings or tuples** to a function, the passing acts *like call-by-value*. The object reference is passed to the function parameters. They can't be changed within the function, because they can't be changed at all, i.e. they are immutable.
  - If **mutable arguments** are passed, they are also passed by object reference, but they can be changed in place in the function. If we pass a list to a function, we have to consider two cases: Elements of a list can be changed in place, i.e. the list will be changed even in the caller's scope. If a new list is assigned to the name, the old list will not be affected, i.e. the list in the caller's scope will remain untouched.

Importing Modules and Scripts

- Modules and Python scripts are loaded in the same manner. For a module or Python script P (.py):
  - (ex) **import P [as p]**
    - Loads the module or script into the workspace, with an optional shorter name
    - Can use any functionality in an OOP fashion (e.g., P.method())
  - (ex) **from python_module import ***
    - Imports all of the functionality directly into workspace
  - (ex) **from python_module import f, g, h**
    - Imports specific functions

NumPy

- Numerical Python
- Foundation for scientific computing
  - Linear algebra and random number generation
  - Integration with C/C++. Fortran for fast execution
- Provides foundation for **Pandas** (Series and DataFrame) structures
  - **ndarray :** similar to lists, but much more powerful
  - **Vectorization**: fast operations on arrays of data without the need for loops
- Primary Use:
  - Fast vectorized array operations for data munging, cleaning, filtering, transforming
  - Built-in common array algorithms
  - Efficient descriptive statistics
  - Data alignment and relational data manipulations for merging and joining multiple data sets
  - Expressing conditional logic ad array expressions instead of loops

Slicing: list and array

- 1-D array slicing (quite often used)

```
a = np.arrange(10)    # a = array([0,1,2,3,4,5,6,7,8,9])
a[start:end]          # items start through end-1
a[start:]             # items start through the rest of the array
a[:end]               # items from the beginning through end-1
a[:]                  # a copy of the whole array
a[start:end:step]     # start through not past end, by step

a[-1]                # last item in the array
a[-2:]               # last two items in the array
a[:-2]               # everything except the last two item
a[::-1]              # all items in the array, reversed
a[1::-1]             # the first two items, reversed
a[:-3:-1]            # the last two items, reversed
a[-3::-1]            # everything except the last two items, reversed
```

Slicing: list and array

- 2-D array slicing (to split loaded data into input(X) and the output(y))

```
X = [:, :-1]    # select al the rows and all columns except the last one
y = [:, -1]     # select all rows again, and index just the last column
```

Some NumPy functions

- Extensive library of mathematical functions
  - Sqrt(x), exp(x), log(x), log10(x), degree(x), radians(x), sin(x), cos(x), tan(x)
  - Arcsin(x), arccos(x), arctan(x), fabs(x), round(x), floor(x), ceil(x), sign(x)

Pandas

- Pandas
  - Provides data processing and analysis capabilities
  - Built on top of Numpy functionality
- Two data structures: **Series** and **DataFrames**
- Important statements
  - **from** pandas **import** Series, DataFrame
  - **import** pandas **as** pd
- What can be done?
  - Creating Series and DataFrame objects
  - Basic Series and DataFrame methods
  - Indexing/reindexing, slicing, and filtering
  - Mathematical operations
  - Missing data

Pandas - Series

- Similar to an ndarray…
  - Easy to perform computation
  - Indexing, slicing, filtering
- With some additional features
  - Comes with an associated array of data labels, called an **index object**
  - Access values using integer indices (like an array) or specified indices (like a dict)
  - Easy merging of data sets

Pandas - DataFrames

- 2-D tabular-like data structure
  - Similar to a dictionary of Series objects with the same indices
  - Hierarchical indexing or panel for higher dimensions
- Access rows or columns by index
- Built-in methods for data processing, computation, visualization, and aggregation
- Creating DataFrames
  - From a dictionary of equal-length sequences or Series objects
    - frame = DataFrame(D)
  - Froma 2-D ndarray or list of lists or tuples
    - frame = DataFrame(arr)

Pandas – DataFrames (example)

- From dictionary

```
In [149]: countries = ['CH','IN', 'US'] * 3
          years = [1990, 2008, 2025] * 3
          years.sort()
          pop = [1141, 849, 250, 1333, 1140, 304, 1458, 1398, 352]

In [151]: D= {'country': countries, 'year':years, 'pop':pop}; D

Out[151]: {'country': ['CH', 'IN', 'US', 'CH', 'IN', 'US', 'CH', 'IN', 'US'],
           'year': [1990, 1990, 1990, 2008, 2008, 2008, 2025, 2025, 2025],
           'pop': [1141, 849, 250, 1333, 1140, 304, 1458, 1398, 352]}

In [154]: frame = DataFrame(D, columns=['year','country','pop']); frame

Out[154]:
```

	year	country	pop
0	1990	CH	1141
1	1990	IN	849
2	1990	US	250
3	2008	CH	1333
4	2008	IN	1140
5	2008	US	304
6	2025	CH	1458
7	2025	IN	1398
8	2025	US	352

Pandas - DataFrames

- Basic DataFrame Methods
  - Indexing: columns returned as a Series
    - Frame['year'] or frame.year; frame[['year','pop']]
  - Frame.columns: returns array of column names
    - Index([year,country,pop], dtype=object)
  - .name, .index.name, .columns.name, and .values similar to Series
- Functions
  - df.sort_index(), df.sort_index(axis=1, ascending=False) // 인덱스 기준, 열 기준
  - df.sum(), df.mean()
  - df.idmax(), df.idmin() // 최대치, 최소치가 있는 위치
  - df.value_counts() // 빈도수
  - df.isin(['b','c']) //특정 항목이 들어 있는지 확인
  - df.fillna(), df.dropna() // NA 가 들어 있는 행 삭제, NA 항목에 채우기

Data Wrangling (Data Munging)

- Process of **transforming and mapping** data from one “raw” data form into another format.
- 원래의 데이터를 또다른 형태로 전환하거나 매핑하는 과정
- Data does not always come in a nice format, ready for pd.read_csv or pd.read_table
- In many cases, we will need to perform several tasks in order to get data in the exact format we want
- Typical tasks:
  - Combining and Merging Data Sets
  - Reshaping and Pivoting
  - Data Transformation
  - Removing duplicates
  - Cleaning and filtering

Feature Engineering (특성공학)

- Process of using domain knowledge of the data **to create features** that make machine learning algorithms work.
- 이미 존재하는 변수로부터 새로운 변수들을 만들어내는 과정 (ex: from Weight, Price features -> create Price_per_Weight feature)
- Process of feature engineering
  - Brainstorming or testing features
  - Deciding what features to create
  - Creating features
  - Checking how the features work with your model
  - Improving your features if needed
  - Go back to brainstorming/creating more features until the work is done

Data visualization - matplotlib

- Use:
  - %matplotlib inline magic command (once Jupyter is open)
  - import matplotlib.pyplot as plt
- Basic template
  - Create a new figure
    - fig = plt.figure()
    - fig = plt.figure(figsize = (12,8))
  - Add subplots (if necessary)
    - ax1 = fig.add_subplot(2,1,1) # 2x1 arrangement, first figure
    - ax2 = fig.add_subplot(2,1,2)
  - Create plot (plt or ax1…axN methods)
  - Label, annotate, format plot
  - Copy or save plot

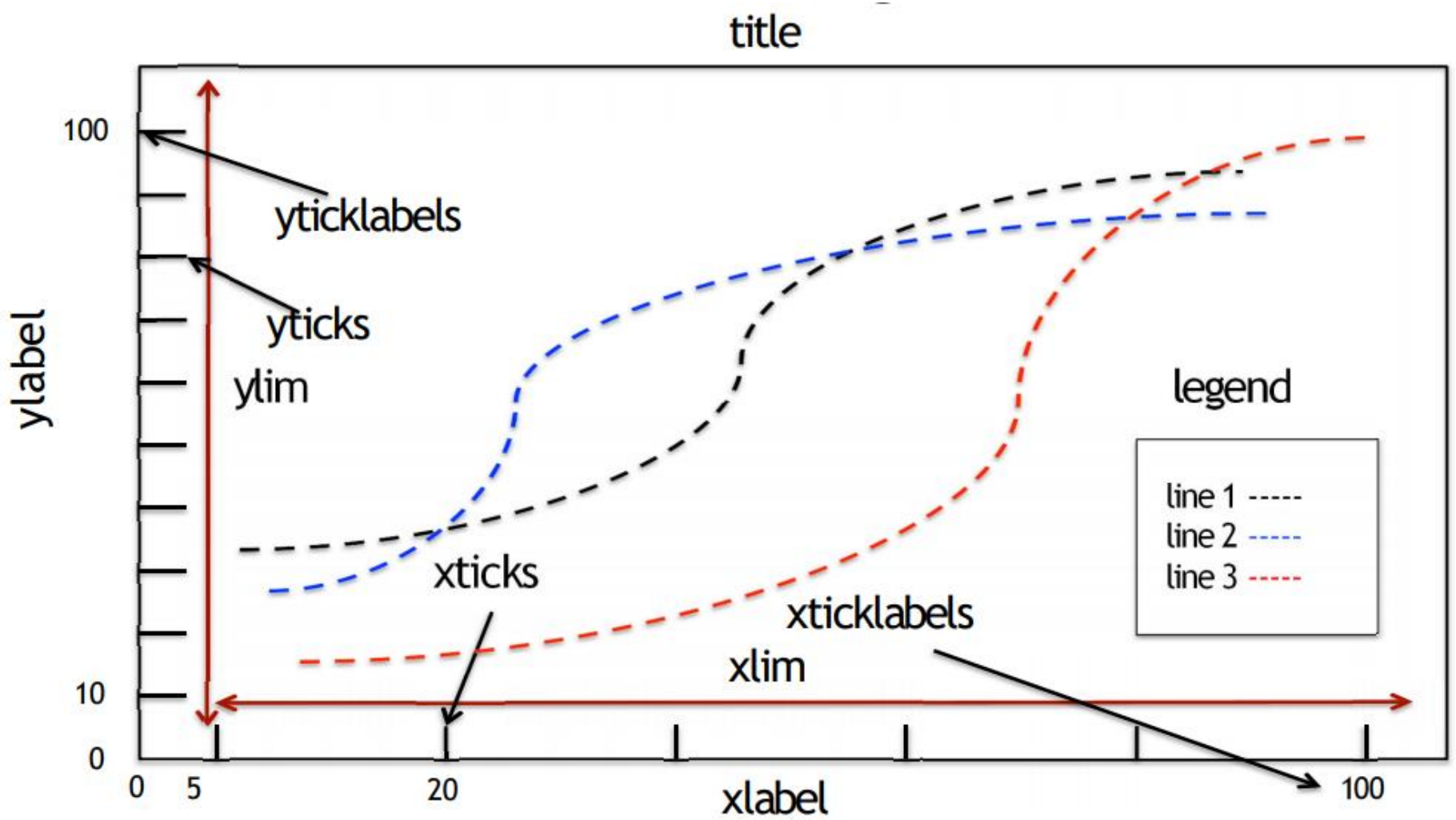
Matplotlib - Common plot types

- Line plots – trends:
  - plt.plot (x, y, '-')
- Scatter plots – comparison between lots of data
  - plt.plot (x, y, '.')
- Bar plots – comparison between few data
  - Bar (horizontal): plt.barh (x, y, width)
  - Column (vertical): plt.bar (x, y, width)
- Histogram plots – single distributions
  - plt.hist (x, bins)
- Boxplots – one or more distributions
  - plt.boxplot (x)

Matplotlib - Colors, Markers, and Line Styles

- All specified as special string characters in plot call
- Colors - Many plot types
  - Basic colors: g(reen), r(ed), b(lue), (blac)k, m(agenta), y(ellow), c(yan), w(hite)
  - For more, see http://matplotlib.org/api/colors_api.html
- Markers and Line Styles - Mostly relate to plt.plot
  - Markers: ., o, +, * (star), 1, 2, 3, 4 (triangles), s(quare), D(iamond)
  - Line styles: solid (-), dashed (--), dotted (:), dash-dot (-.)
  - linewidth keyword (float value)
- Usage
  - Style string: Combines all three (e.g., 'k.', 'g--', 'ro-')
  - Separate keyword arguments: color, linestyle, marker

Formatting plots



Formatting plots

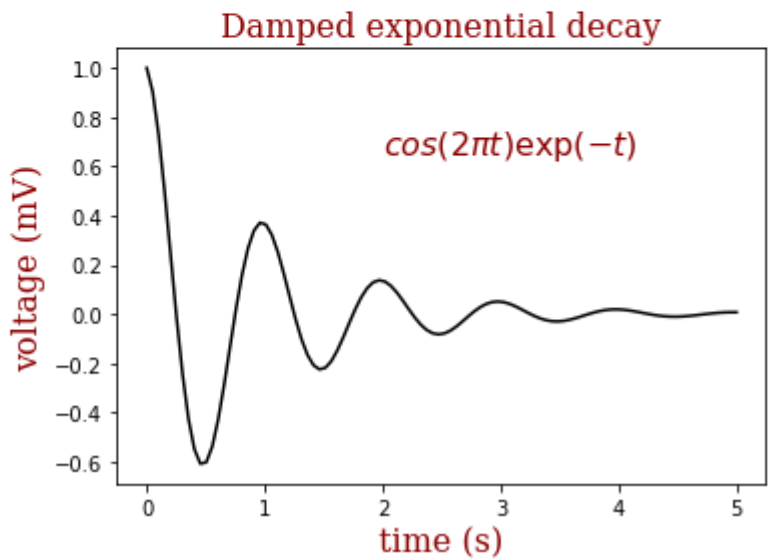
- Title
  - title('Title')
- Axis labels
  - xlabel ('Time'), ylabel ('Price)
- Axis limits
  - xlim([0,10]0, ylim
- Ticks
  - xticks([0,60,70,80,90,100]), yticks
- Tick labels – combine with ticks for text labels
  - xticklabels(['F','D','C','B','A']), yticklebals
- Legends
  - List of labels for each series: legend(('one','two','three'))
  - Use legend()
  - Location keyword: loc = 'best', 1-10 (upper right, left, center, etc.)

Annotating plots

- Text
  - text(x, y, text, fontsize)
  - arrow(x, y, dx, dy) # draws arrow from (x,y) to (x+dx, y+dy)
  - annotate (text, xy, xytext) # annotate the xy point with text positioned at xytext
- shapes
  - Rectangles, circles, polygons
  - Location, size, color, transparency (alpha)

Matplotlib - Example(1)

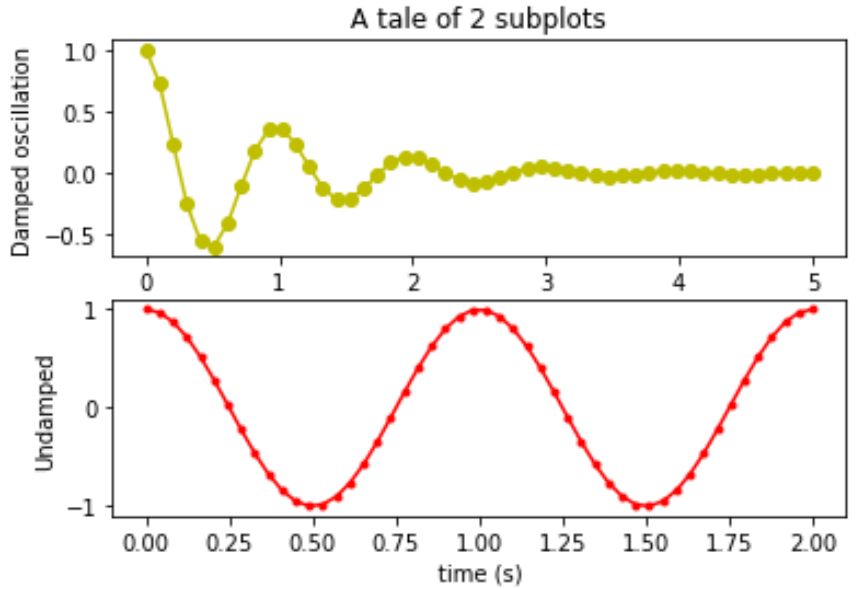
In [27]:
```
x = np.linspace(0.0,5.0,100)
y = np.cos(2*np.pi*x) * np.exp(-x)

plt.plot (x,y,'k')
plt.title('Damped exponential decay', fontdict=font)
plt.text(2, 0.65, r'$cos(2 \pi t) \exp(-t)$', fontdict=font)

plt.xlabel('time (s)', fontdict=font)
plt.ylabel('voltage (mV)', fontdict=font)

plt.subplots_adjust(left=0.15)
```



Matplotlib - Example(2)

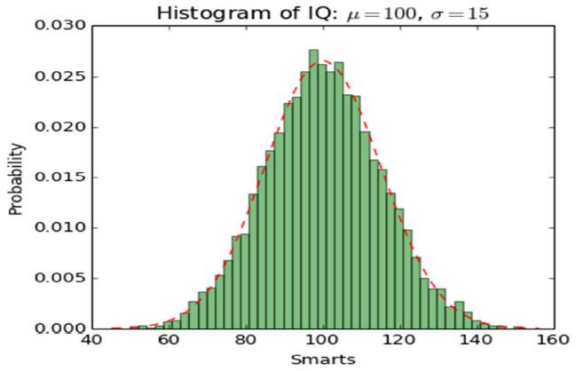
In [39]:
```
x1 = np.linspace(0.0,5.0)
x2 = np.linspace(0.0,2.0)
y1 = np.cos(2*np.pi*x1) * np.exp(-x1)
y2 = np.cos(2* np.pi* x2)

plt.subplot(2, 1, 1)
plt.plot(x1,y1,'yo-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2)
plt.plot(x2, y2,'r.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
```

Out [39]: Text(0, 0.5, 'Undamped')

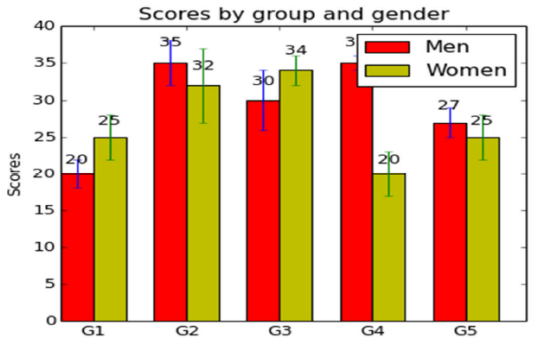


Many more examples…

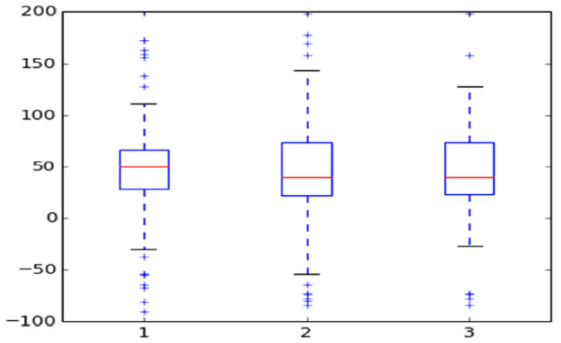
Histogram



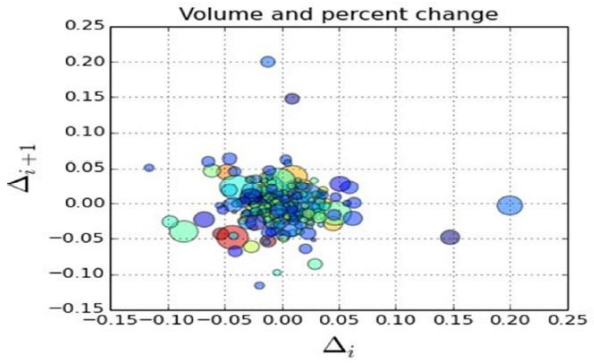
Bar Chart (with error bars and legend)

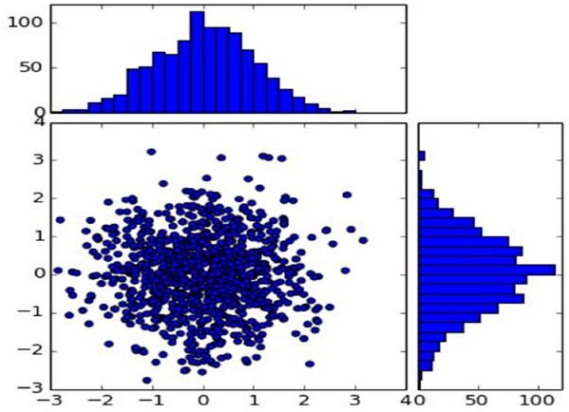


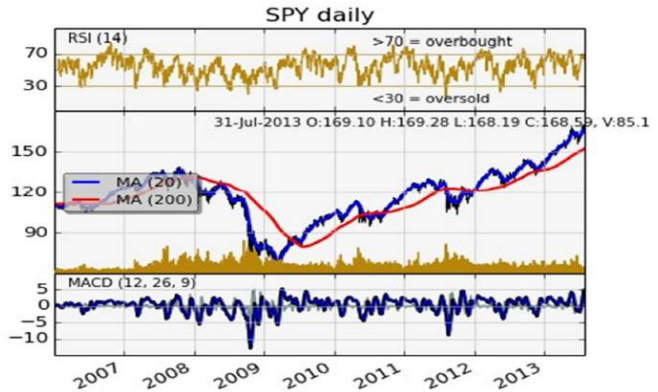
Boxplots



Scatter + Histogram







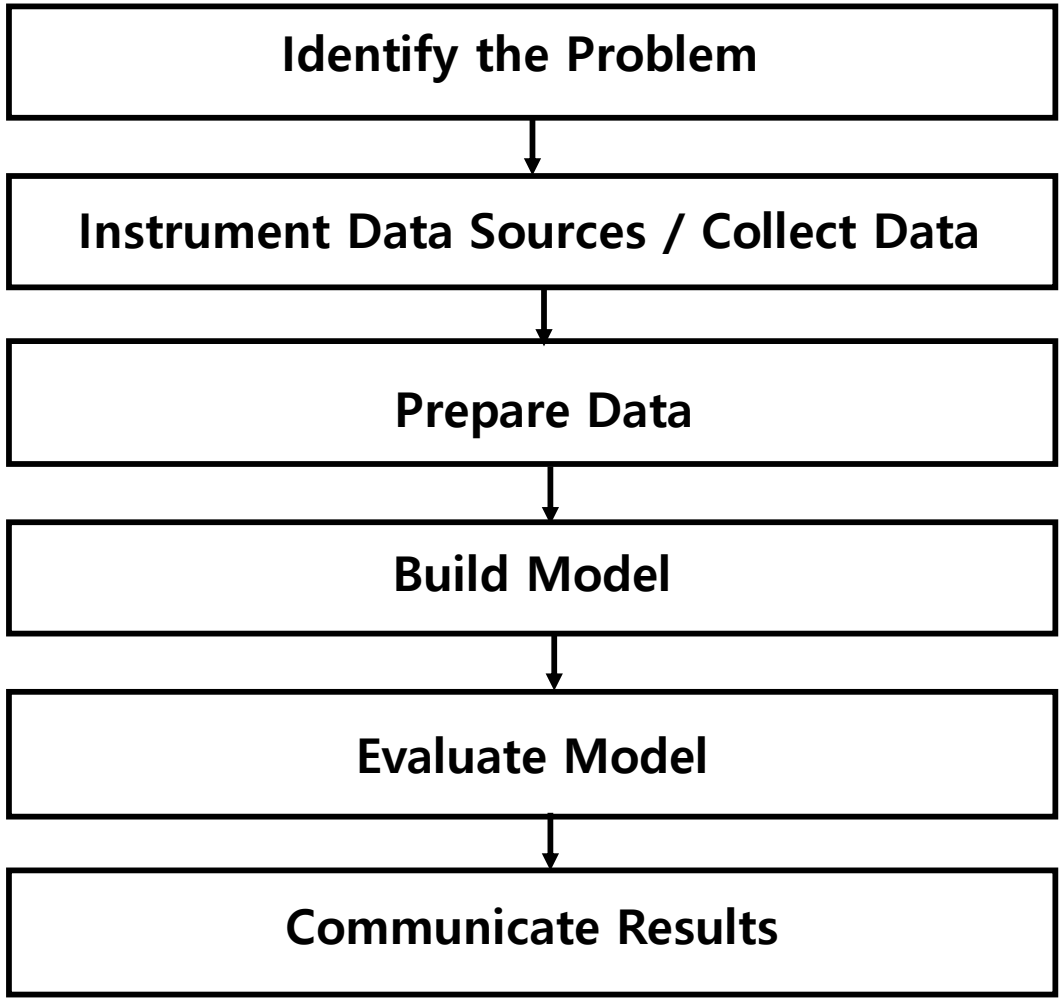
데이터 수집

- 데이터 수집 (Data Munging or Data Wrangling)
  - 데이터를 획득하고 분석에 맞게 준비하는 작업
- 데이터는 어디에?
  - 데이터 분석에 맞는 적당한 데이터를 찾는 것이 중요함
  - 찾아진 데이터를 목적에 맞게 재조정(repurposing) 할 때 창의성 필요
- 데이터 소스
  - Proprietary data source: Facebook, Google, Amazon etc.
  - Government data sets: data.gov or data.go.kr
  - Academic data sets: track down from relevant papers, and ask
  - Web Search/Scraping: fine art of stripping text/data from webpage
  - Sensor data sets: IoT do amazing things (image,video,…)
  - Crowdsourcing: Wikipedia/Freebase, IMDB
  - Sweat equity: you must work for your data instead of stealing it

데이터 수집

- 데이터 클리닝 (Data Cleaning)
  - Garbage in, garbage Out
  - 데이터 정리 과정 필요
    - Distinguishing errors from artifacts
    - Data compatibility
    - Imputation of missing values (결손값의 대체)
    - Estimating unobserved (zero) counts
    - Outlier detection
- Errors and Artifacts
  - Errors: 수집 과정에서 원천적으로 빠진 것 (복구 불가능)
  - Artifacts: 데이터를 처리하는 과정에서 발생한 문제 (복구 가능)

Data Analysis Model (Jeff Hammerbacher)



(*) integrate, transform,
clean, aggregate, alter

Machine Learning (머신러닝)

❖ What is ML?

- 데이터로부터 학습하도록 컴퓨터를 프로그래밍하는 과학 또는 예술
- 명시적인 프로그래밍 없이 컴퓨터가 학습하도록 능력을 갖추게 하는 연구 분야
- 어떤 작업 **T(task)** 에 대해 컴퓨터 성능을 **P(performance)**로 측정했을 때 경험 **E (experience)** 로 성능이 향상됐다면, 이 컴퓨터 프로그래밍은 작업 T 와 성능 측정 P 에 대해 경험 E로 학습한 것이다. (Tom Mitchell, 1997)
- (ex) T: 스팸 메일 필터, E: 일반 및 스팸 메일 샘플, P: (분류) 정확도

❖ 머신러닝 시스템의 종류

- 훈련 여부: supervised(지도학습), un-supervised(비지도), semi-supervised(준지도)
- 실시간 점진적인 학습여부: on-line(온라인학습), batch(off-line, 배치학습)
- 새로운 데이터에 대한 일반화(ex. 예측): instance-based(사례기반), model-based(모델기반)

Machine Learning

❖ Supervised Learning

- Training Data 에 **Feature**(or **attributes**) 와 **Label**(or **Target**) 포함
- 분류(**Classification**): feature 를 이용해 taget 의 class 예측 (ex: 스팸메일 분별)
- 회귀(**Regression**): feature 를 이용해 target 수치 예측 (ex: 중고차 가격 예측)
- (ex) KNN(K-Nearest Neighbor), Linear Regression, Logistic Regression, SVM(Support Vector Machine), Decision Tree, Random Forest, Neural Networks

❖ Unsupervised Learning

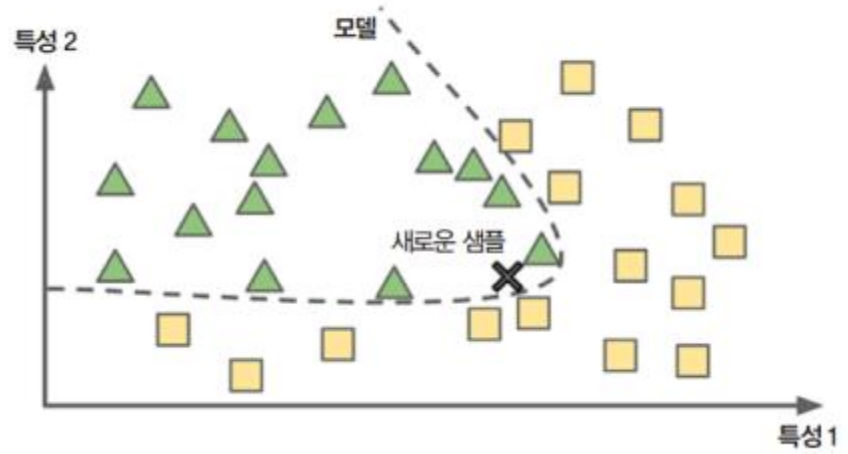
- 실시간 점진적인 학습여부: on-line(온라인학습), batch(off-line, 배치학습)
- 새로운 데이터에 대한 일반화(ex. 예측): instance-based(사례기반), model-based(모델기반)
- (ex) **Clustering**, PCA(Principal Component Analysis), Kernel-PCA

❖ Reinforcement Learning

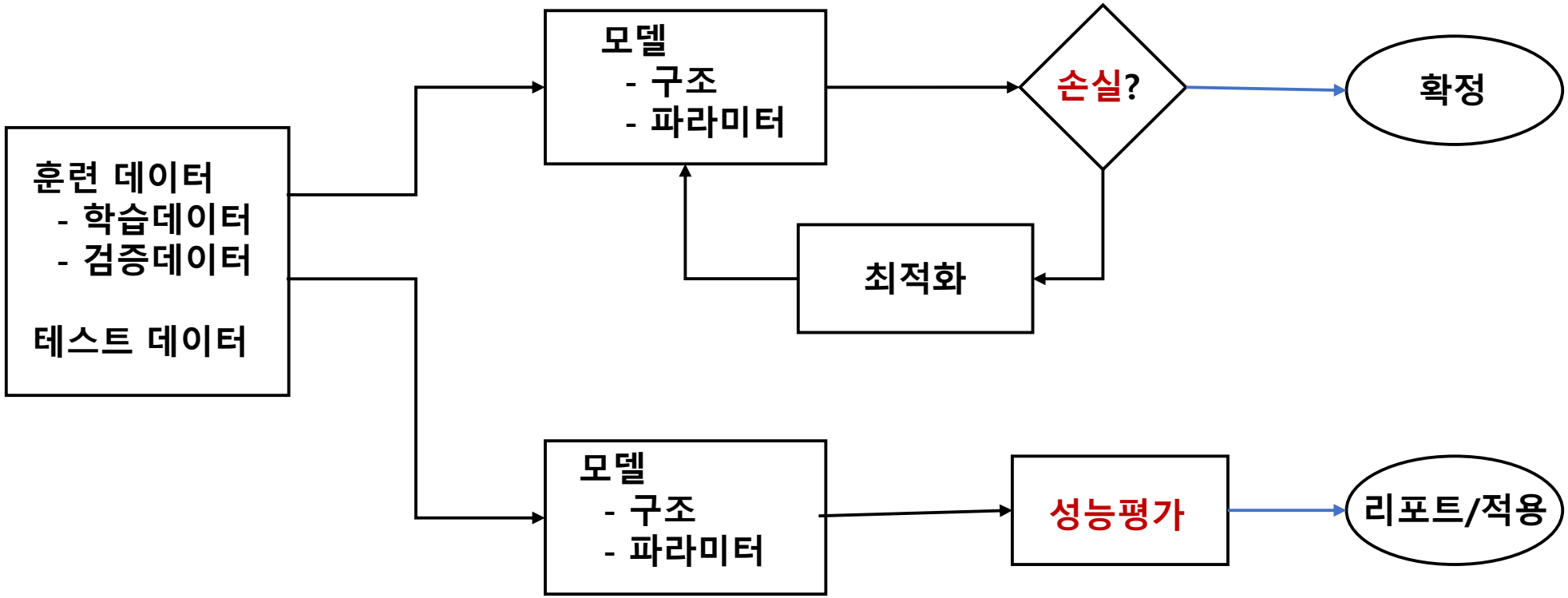
- Reward 와 penalty 를 기반으로 학습

Machine Learning

- ❖ Instance-based Learning
- ❖ Model-based Learning



Machine learning (기계학습) Model



Clustering (군집)

❖ **What is Clustering (군집)?**
- 각 객체의 유사성을 측정하여 유사성이 높은 집단으로 나눔
- 그룹에 대한 사전 정보 없음.
- 그룹의 개수나 특성에 대한 사전 정보가 주어지면 -> Classification (분류) 사용
- 군집의 개수나 구조에 대한 가정 없이 각 데이터 간의 거리를 기준으로 나눔

- **Similarity or Proximity (유사도)**
  - 항목 간의 유사한 정도를 수치로 표현
  - Euclid Distance (유클리드 거리), Manhattan Distance(맨하탄 거리), etc.
  - 범주형 – Jaccard Distance (자카드 유사도)

❖ **What is Hierarchical/Agglomerative Clustering (계층적/응집형군집)?**
- 객체간의 유사도를 계산해 가장 가까운 것들부터 차례로 군집화
- Dendrogram 을 사용해 군집 형성 과정 파악
- 방법: Single, Complete, Average, Ward(군집간 정보 손실 최소화)

Regression (회귀) – 예측, 분류

❖ **What to reduce? (Loss Function: 손실함수)**

- **MSE** (Mean Square Error)

$$MSE = \sum_{k=1}^{N} (y - \hat{y})^2$$

❖ **How Good it is? (Performance: 성능지표)**

- **R^2** (R-Squared)

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2$$

Classification (분류)

❖ What to reduce? (Loss Function: 손실함수)

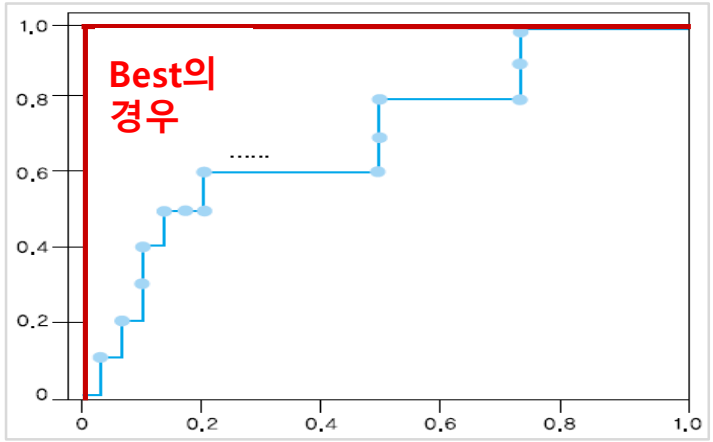
- **Cross Entropy (CE)**
- **Gini (지니계수)**

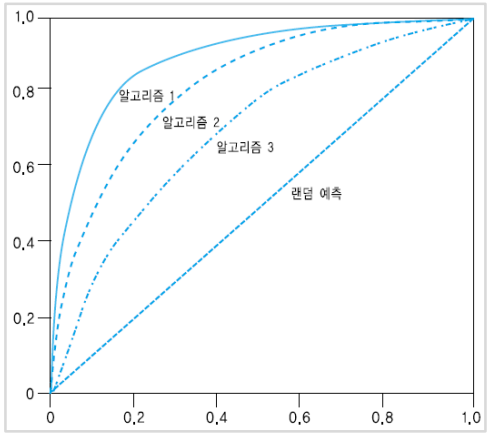
$$CE = \sum_i p_i \log\left(\frac{1}{p_i}\right)$$

$$Gini = 1 - \sum_{k=1}^{m} p_k^2$$

❖ How Good it is? (Performance: 성능지표)

- **Confusion Matrix**: Accuracy, Recall, Precision, F-1 Score
- **Ranking(순서)**: ROC (Receiver Operating Characteristic), AUC (Area Under Curve)





Schedule for Module1

	Contents	Lab	Pages
Day 1	• What is Data Science? • Probability and Statistics review • Linear Algebra • Introducing Python language	extra_lab1	
Day 2	• Numpy • Pandas • Data reading and cleaning	1,2 3,4,5 6,7,8	
Day 3	• Crawling & Scraping • Bayes algorithm	9,10,11,12,13 14,15,16	
Day 4	• Machine learning concept (Gradient Descent) • Review and Exam	extra_lab2	

Schedule for Module2

	Contents	Lab	Pages
Day 1	• What is Data Science? • Introducing Python language	extra_lab1	
Day 2	• Data Manipulation, analysis, munging, viewing • Scaling • Clustering	17,18,25 26	
Day 3	• Machine learning concept (Gradient Descent) • Linear Regression • Linear Classification	extra_lab2 29,30,31 32	
Day 4	• Logistic Regression • Review and Exam	33	