Data Science Introduction

2020.8 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
- 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
- Query "data science jobs" in Google or Daum, Naver, JobKorea

What will be the core job of the future?

- from Northeastern University
  - For four years in a row, data scientist has been named the number one job in the U.S. by Glassdoor, one of the world's largest job and recruiting sites.
  - The U.S. Bureau of Labor Statistics reports that the demand for data science skills will drive a 27.9 percent rise in employment in the field through 2026. There will be a huge demand for qualified data scientists.
- Data Scientists jobs requirements
  - Average salary (in 2020): $139,840
  - **Find, clean, and organize data** for companies.
  - **Analyze large amounts of complex raw and processed information** to find patterns that will benefit an organization and **help drive strategic business decisions.**
  - Compared to data analysts, data scientists are much more technical (past vs. future)

Data in Korea and Worldwide

- Data science Jobs in Korea
  - https://www.glassdoor.com/Job/south-korea-data-scientist-jobs-SRCH_IL.0,11_IN135_KO12,26.htm
  - https://kr.linkedin.com/jobs/data-science-jobs?position=1&pageNum=0
  - Search 'data science jobs' in Google
  - Amazon, Boston Consulting group, NVIDIA, Google, Hyperconnect, Coupang, …
- Data sites:
  - www.data.gov (home of US government's open data)
  - www.data.go.kr (Korea government's public data)
  - Now, personal information (pseudonym information) can also be used for special purposes without permission in Korea.
  - Expect many new business ideas.

Data in Korea and Worldwide

❖ 데이터 이용을 활성화하기 위한 데이터 3법 통과 (2020.1)
- 개인정보 보호법, 정보통신망법, 신용정보법
- 핵심: 가명정보를 통계작성, 연구, 공익적 기록 보존 용도로 본인 동의없이 활용 가능

- 가명정보 (예: https://brunch.co.kr/@jaeyunchoi/18)

	개념	예시	활용가능 범위
개인정보	특정 개인에 관한 정보, 개인을 알아볼 수 있게 하는 정보	강하늘, 1990년 2월 21일생, 남성, 2019년 12월 신용카드 사용금액 150만 원	사전적, 구체적 동의를 받은 범위 내에서만 활용 가능
가명정보	추가정보의 사용 없이는 특정 개인을 알아볼 수 없게 조치한 정보	강XX, 1990년생, 남성, 2019년 12월 신용카드 사용금액 150만 원	개인정보 범위에 포함되나, 다음 목적에 한하여 동의없이 활용 가능 ① 통계작성(상업적 목적 포함) ② 연구(상업적 연구 포함) ③ 공익적 기록보존 목적 등
익명정보	더 이상 개인을 알아볼 수 없게 (복원 불가능할 정도로) 조치한 정보	남성, 20대, 2019년 12월 신용카드 사용금액 100만 원 이상	개인정보가 아니므로 제한없이 자유롭게 활용 가능

What is Data Science?

❖ Definition (from Wikipedia)

- concept to unify statistics, data analysis, machine learning, domain knowledge and their related methods in order to understand and analyze actual phenomena with data
- It uses techniques and theories drawn from many fields within the context of mathematics, statistics, computer science, domain knowledge, and information science

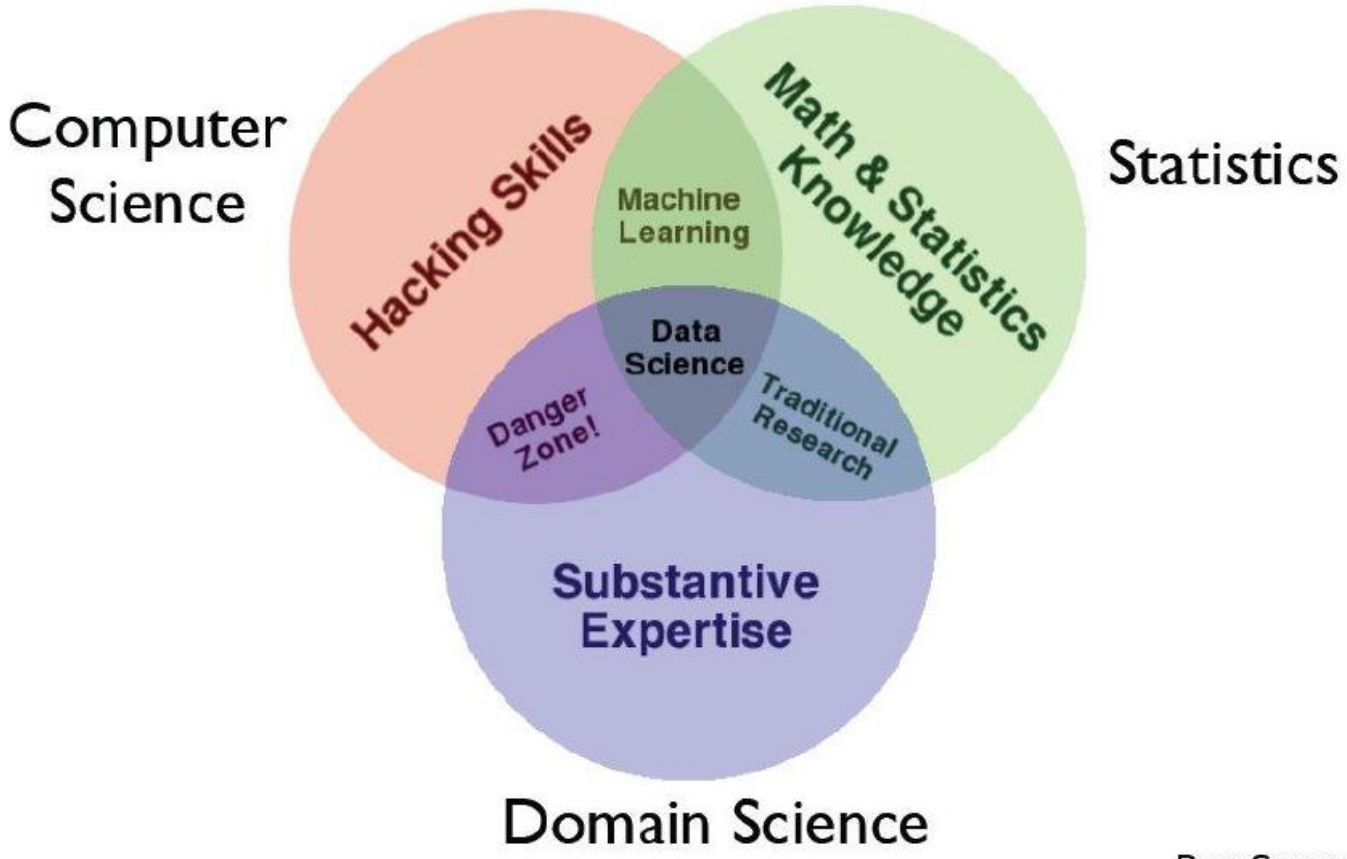
❖ Components of Data Science

- 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, etc.
- Almost all areas

What is Data Science? – One definition



Drew Conway

What is Data Science? – Simple definition

- **How is it different from traditional Statistics?**
  - **Data + Analysis and Processing**
- **Data**
  - Structured or unstructured
  - Huge amount (Big Data)
- **AI (Machine Learning)**
  - Various ML models (including Deep Learning)
  - Programming flexibility (Python)
- **Query (Past) vs. Predict (Future)**

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!

Data Science Applications and Examples

- (ref) https://builtin.com/data-science/data-science-applications-examples
- Healthcare
  - Google: machine learning for metastasis (identifying breast cancer)
  - CLUE, Germany: predict periods and forecast conditions for pregnancy
  - Oncora Medical: cancer care recommendations
- Road Travel
  - UPS: optimizing package routing (save up to $200 million)
  - Streetlight data: traffic patterns for cars, bikes, and pedestrians (use for commuter transit design)
  - Uber Eats (Uber's delivery app): optimize full delivery process
- Sports
  - Liverpool F.C.: recruited undervalued soccer players
  - RSPCT: basketball-coaching sensor (shooting analysis system)
  - British Olympic Rowing team: model athlete evolution and find a promising newbie rower

Data Science Applications and Examples

- Government
  - Equivant: data-driven crime prediction
  - ICE (Immigrations and Customs Enforcement): facial recognition in ID databases
  - IRS: tax-fraud detection
- E-commerce
  - SOVRN: automated AD placement (target campaigns to custmoers)
  - Instagram: convert users' likes and comments, their usage of other apps and their web history into predictions about the products they might buy
  - Airbnb: search that highlights areas of cool neighborhoods (high density of bookings)
- Social life
  - Tinder (most popular dating app): find a good match for singles
  - Facebook: "people you may know" sidebar (based on friend list, photos, schools, etc.)

Kaggle.com

❖ What is Kaggle?

- Owned by Google, and over 3 million data scientist registered.
- The world's largest data science and machine learning community with powerful tools and resource. (over 19,000 public datasets and 200,000 public notebooks)
- You can find and publish data sets, and all data sets are **free**.
- Can participate competitions to solve data science challenges.
- Provide self-learning courses (from Python to Deep Learning)
- Explore and run machine learning code with Kaggle Notebooks (with source codes).
- Can discuss any data science issues with experts.
- Try it at *https://www.kaggle.com*.

Data Science Work Flow

- Domain knowledge
- Business strategy

- String(structured)
- Text(unstructured)
- CSV/Excel
- JSON
- HTML/XML
- SNS

- Visualization
- Missing values
- Invalid values
- Outliers
- Categorical values
- Scaling
- Transform

- Supervised
- Unsupervised
- Error (or Loss)
- Bias and Variance
- Regularization
- CNN/RNN
- Generative model

- R-square
- Accuracy
- Precision/recall
- F-1 score
- ROC/AUC

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.

Managing conda and Anaconda

- Managing conda and anaconda
  - conda info # verify conda is installed, check version number
  - conda update conda # update the conda command
  - conda update anaconda # update anaconda meta package
- Managing packages in Python
  - conda list # view list of packages and versions
  - conda search PKG # search for a package
  - conda install PKG # install packages
  - conda update PKG
- Many more . . . (see the document)

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 and Series
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
  - Directly translate source code (do not generate compiled code)
  - 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
- **Dynamic typing** language: variable type determined at run-time (no need of variable declaration), slow but efficient memory usage

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.
- 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

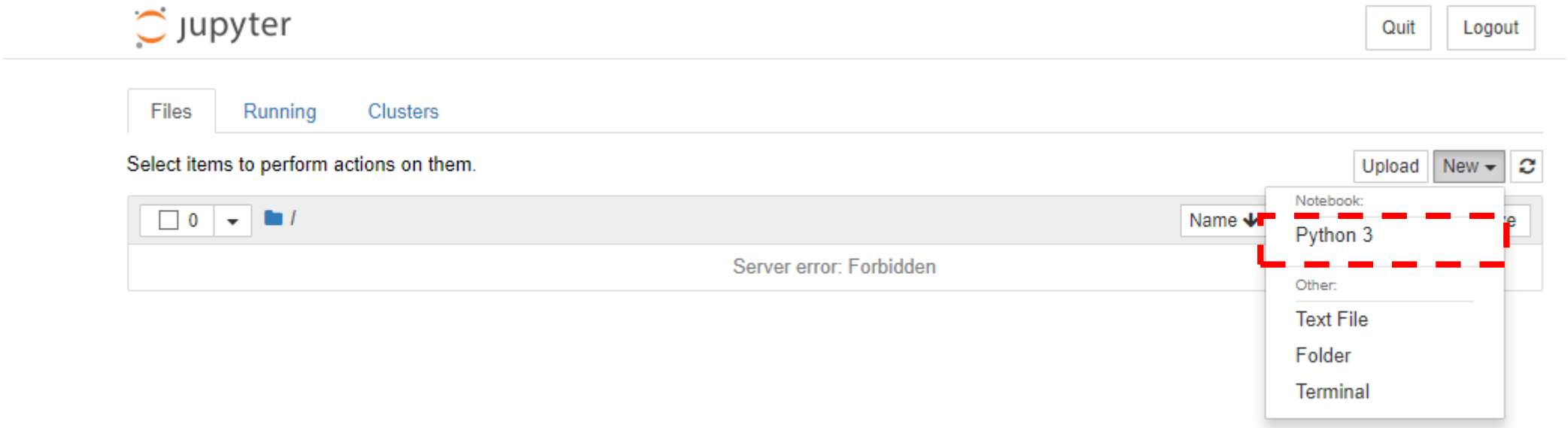
Jupyter Notebook

- 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

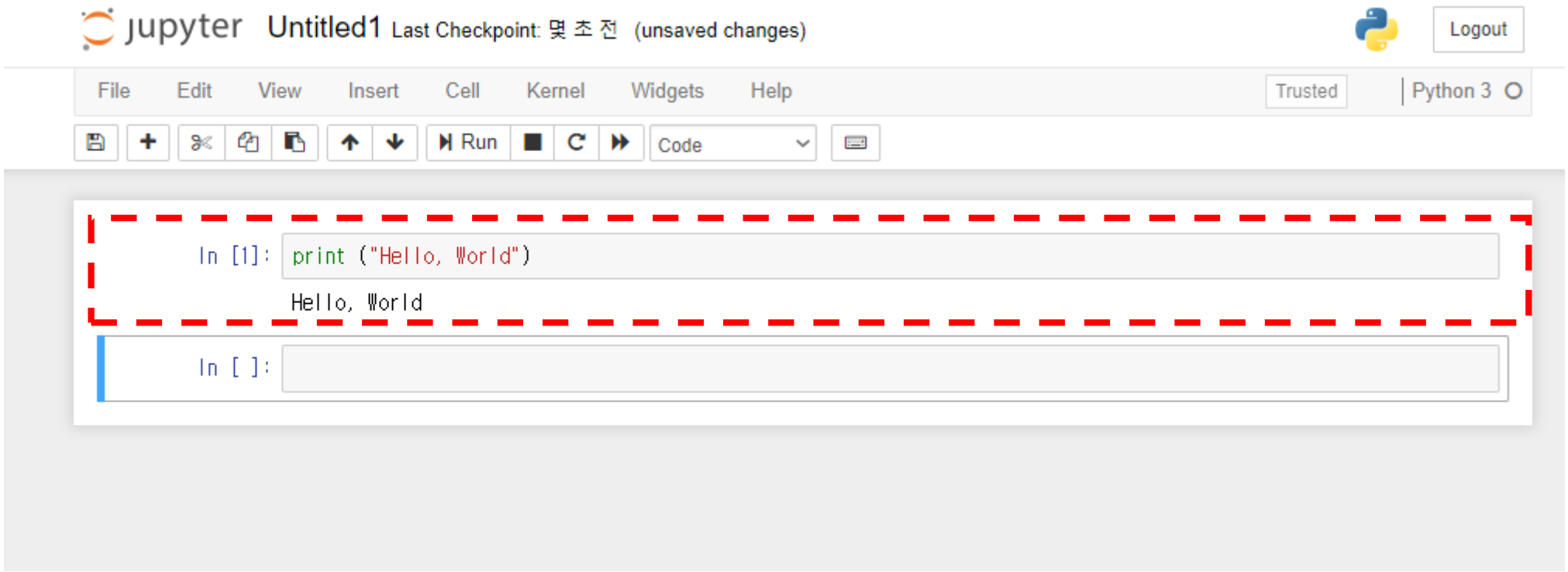
Jupyter notebook

- 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

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

Colab from Google

❖ Free cloud service from Google

- A Jupyter notebook environment that requires no setup to use
- Supports free GPU/TPU, and Runs entirely in the cloud
- provides a maximum GPU runtime of 8~**12 hours** ideally at a time

❖ Useful Shortcuts

actions	colab	jupyer
Convert to code cell	Ctl/Cmd M Y	Y
Convert to text cell	Ctl/Cmd M M	M
Split at cursor	Ctl/Cmd M -	Ctrl Shift -
Merge two cells	Ctl/Cmd M /	Shift M
Show keyboard shortcuts	Ctl/Cmd M H	H
Interrupt execution	Ctl/Cmd M I	II

Python Language

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

Data types

- **Basic types**
  - Int, float, boolean
- **Container types**
  - String: sequence of characters, “Hello”
  - List: can contain any types of variables, mutable, [1, 2.3, “Welcome”]
  - Tuple: can read, but not overwrite (to make computation fast), immutable, (1, 3, [2,3])
  - Dictionary (or dict): only access by keys, mutable, {“name”:”Kim”, “age”:25}
- **Array (or ndarray)**
  - Defined in numpy : similar to list, but much more efficient
  - all the elements are of the same type (int, float, Boolean, string, or other object)
  - Element-wise operation (vector operation)
- **DataFrame and Series**
  - Defined in pandas: provides data processing and analysis capabilities
  - Built on top of Numpy functionality
  - Table-shaped: “columns” and “index”

Objects, attributes, and methods

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

Functions vs. Object Methods

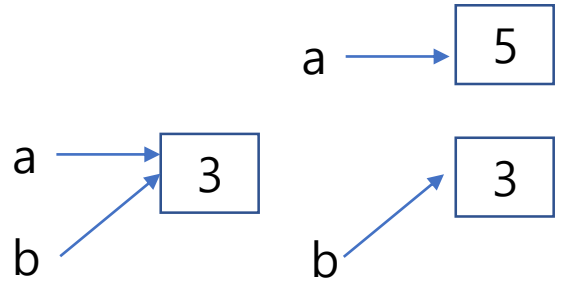
- Functions and object methods are essentially the same…
  - One or more bundled steps performed on some input
  - In some cases, there will be a function and an object method that do the same thing (e.g., sum)
- …BUT, they differ in how they are used
  - Functions are called on zero or more objects 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

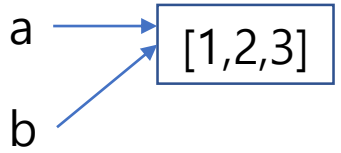
- Mutable Objects
  - Can be modified via assignment or a function/method
  - Lists, dictionaries, arrays, dataframes, class instances
- Immutable Objects
  - Can not be modified
  - int, float, Boolean, strings, tuples

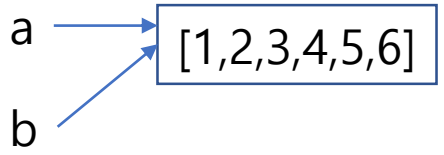
Mutable and Immutable (examples)

```
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.**
  - When you pass arguments to a function, you are passing the object reference, not a copy of the object.

Importing Modules and Scripts

- **Modules**
  - Simply a python file with a .py extension (module name is the file name)
  - Can define functions, classes and variables
- **Packages**
  - Directory which contains multiple packages or modules
  - **Must** contain a special file called __init__.py (indicates it is a Python package)
- Modules and Python scripts are loaded in the same manner. For a module or Python script P (.py):
  - import P [as p]
  - from module_name import * // import all functionality
  - from module_name import f, g, h // import specific functions
  - Import foo.bar (or from foo import bar) // import module bar from package foo
- **Built-in modules (standard library)**
  - https://docs.python.org/3/library/

Indexing and Slicing

- **for container variables: lists, arrays, tuples, and strings**
  - e.g., `A = [1,2,3]`
- **Indexing**: access item in a sequence
  - Python is zero-based: `A[0] = 1, A[1] = 2, A[2] = 3`
  - Negative indices: `A[-1] = 3, A[-2] = 2`
- **Slicing**: access subset of a sequence [*start: stop: step*]
  - `A[start=0: stop=len(A): step=1]`
  - Slicing ends before the *stop* (excluding *stop*): `A[0:1] = [1]; A[0:2] = [1,2]`
- Examples
  - `A[:] = A[::] = A = [1,2.3]`
  - `A[1:] = [2,3], A[:2] =[1,2]`
  - `A[::2] = [1,3]`
  - `A[::-1] = [3,2,1]`
  - `A[-1:0:-1] = [3,2]`
  - More…

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

NumPy - ndarray

- Numerical Python N-dimensional arrays
  - Similar to list, but much more powerful
- Fast, flexible container for data
  - Numerical, boolean, or string data
- Perform mathematical operations on entire data sets without loops (**vectorization**)
- Some common attributes (i.e., arr.*attr*)
  - ***ndim***: Number of array dimensions (e.g., 1, 2, 3)
  - ***shape***: Size of each dimension (e.g., (2, 4))
  - ***dtype***: Data type for the array (e.g., int8, float64)
- Use tab completion to explore attributes and methods

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
```

Array operations

- Between Arrays and Scalars -- Broadcasting
  - All basic operations are applied **element-wise**
  - +, -, /, *, **, %, etc.
- Universal Functions (ufunc)
  - Unary (on a single array): abs, sqrt, exp, log, ceil, floor, logical_not, and more
  - Binary (on two equal-sized arrays): +, -, /, *, **, min, max, mod, >, >=, <, <=, ==, !=, &, |, ^
- Mathematical and Statistical Functions/Methods
  - Aggregation (collection): mean(), sum(), std(), var(), min()/max(), argmin()/argmax()
  - Non-aggregation: cumsum(), cumprod()

Pandas

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

Pandas - Series

- A single column of DataFrame
- 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 by **index**, and columns by **column names**
- Built-in methods for data processing, computation, visualization, and aggregation

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(features): either by column name or attribute (ex: `df['year']` or `df.year`, `df[['year','pop']]`)
  - Indexing rows by index name or index number: `df.loc()` or `df.iloc()`
  - `df.name`, `df.index`, `df.columns`, and `df.values` (similar to Series)
- Functions
  - `df.sort_index()`, `df.sort_index(axis=1)` // sort by index or columns
  - `df.sum()`, `df.mean()`
  - `df.idmax()`, `df.idmin()` // index of max and min
  - `df.value_counts()` // counts of values
  - `df.isin(['b','c'])` // see if some elements are in df
  - `df.fillna()`, `df.dropna()` // remove or fill any columns of NaN

Data visualization - matplotlib

- Use:
  - %matplotlib inline magic command (once Jupyter is open)
  - import matplotlib.pyplot as plt
- Basic template
  - Create a new figure : (ex) 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

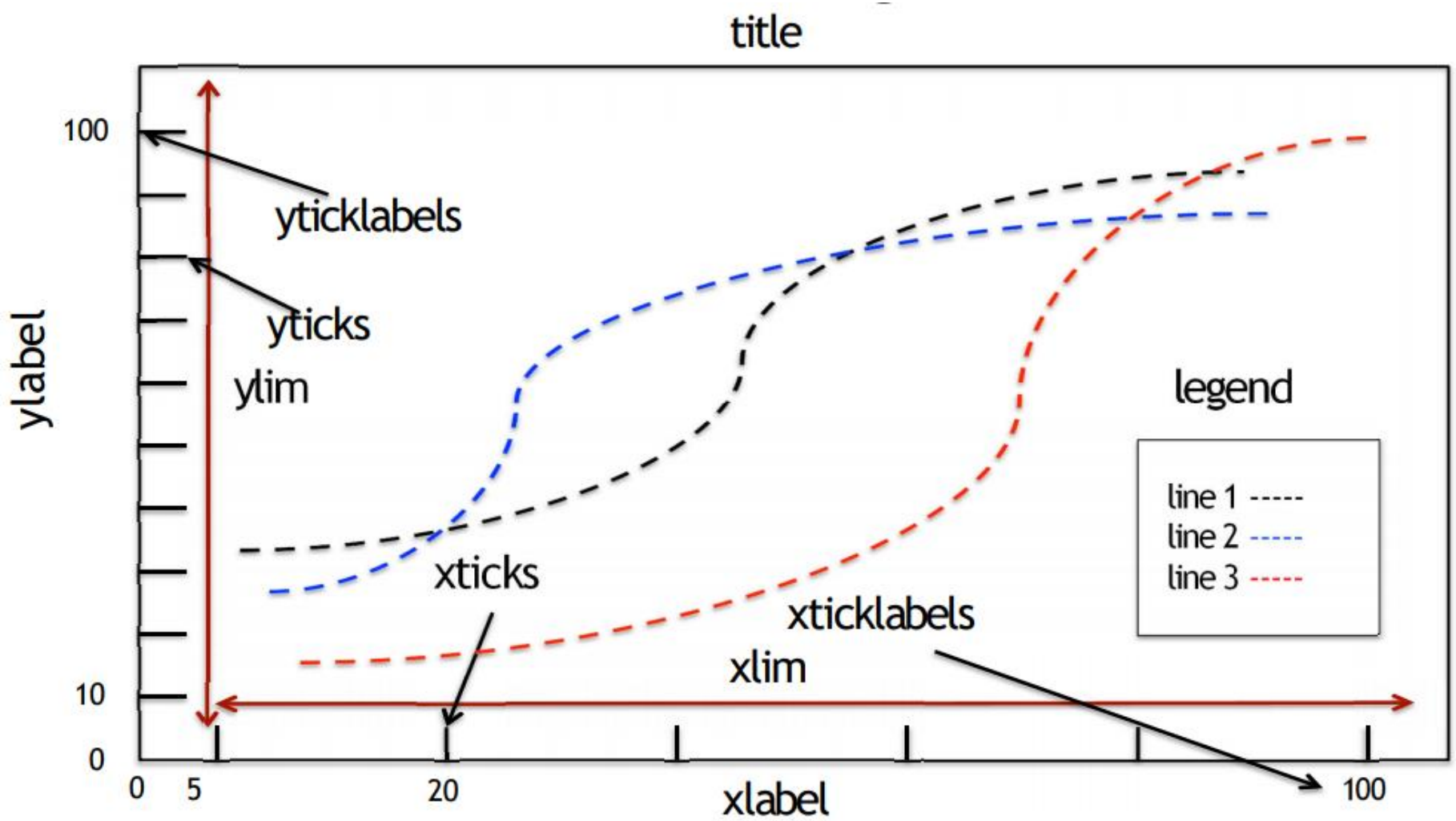
Matplotlib - Common plot types

- Line plots
  - plt.plot (x, y, '-')
- Scatter plots – comparison between lots of data
  - plt.scatter (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

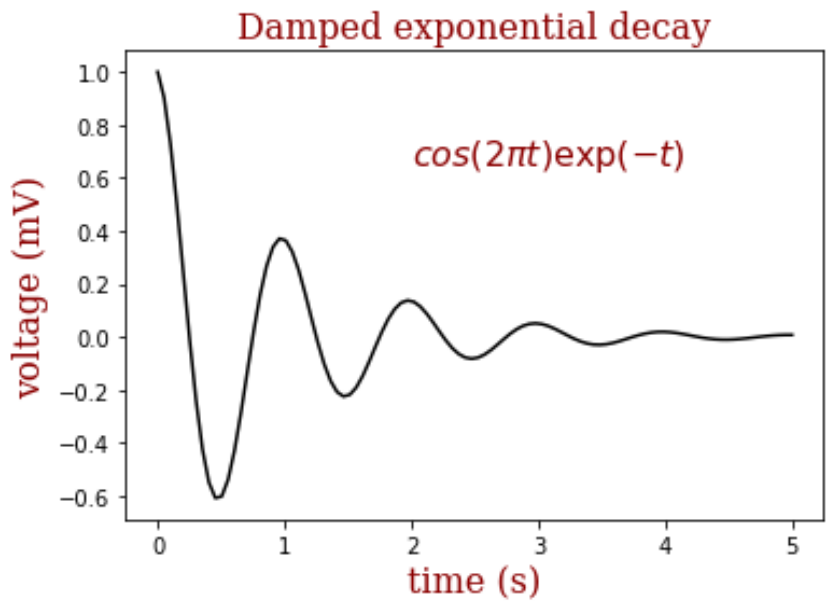
- Title
  - title('Title')
- Axis labels
  - xlabel ('Time'), ylabel ('Price)
- Axis limits
  - xlim([0,10]0, ylim
- Ticks
  - xticks([0,60,70,80,90,100]), yticks
- Tick labels – combine with ticks for text labels
  - xticklabels(['F','D','C','B','A']), yticklebals
- Legends
  - List of labels for each 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)

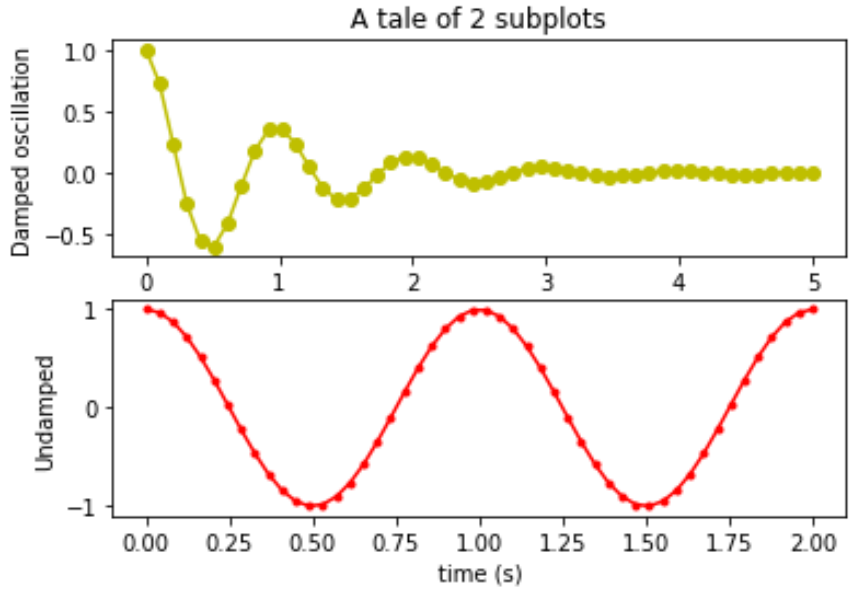
In [27]:
```
x = np.linspace(0.0,5.0,100)
y = np.cos(2*np.pi*x) * np.exp(-x)

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)

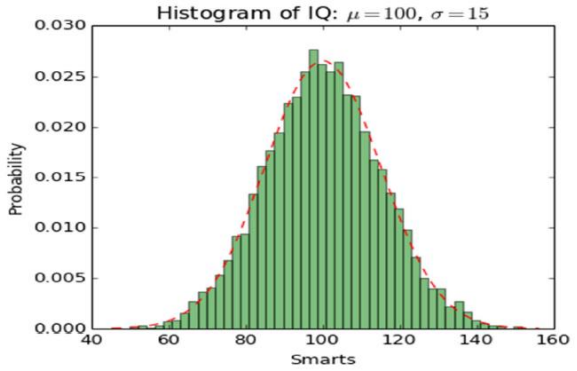
In [39]:
```
x1 = np.linspace(0.0,5.0)
x2 = np.linspace(0.0,2.0)
y1 = np.cos(2*np.pi*x1) * np.exp(-x1)
y2 = np.cos(2* np.pi* x2)

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')

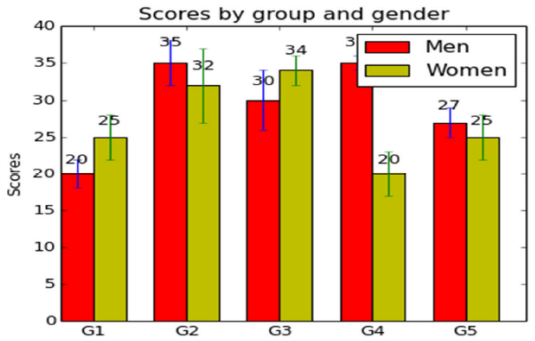


Many more examples…

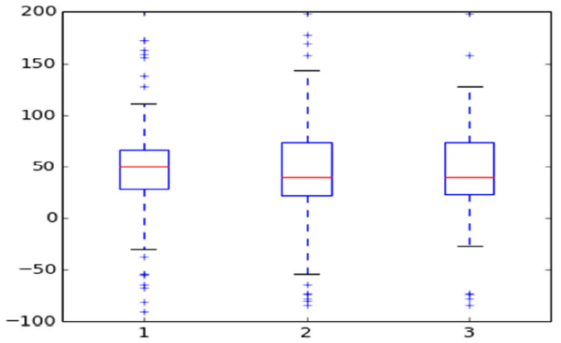
Histogram



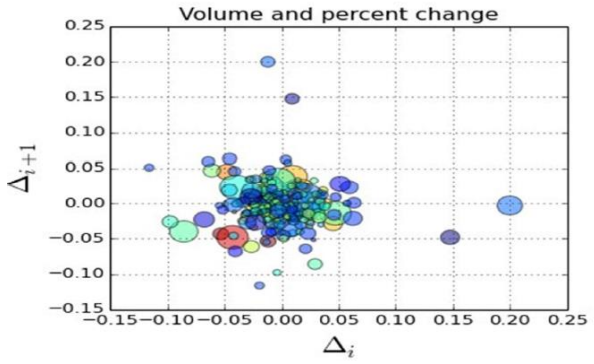
Bar Chart (with error bars and legend)

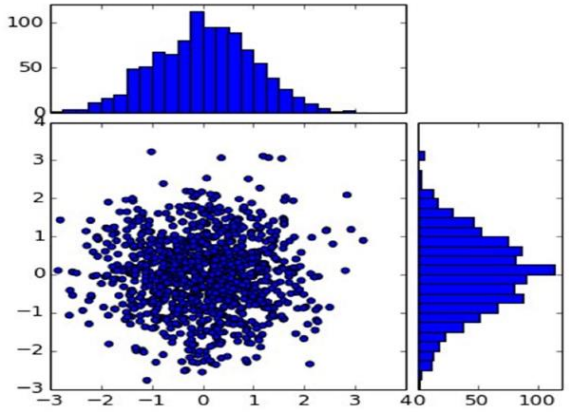


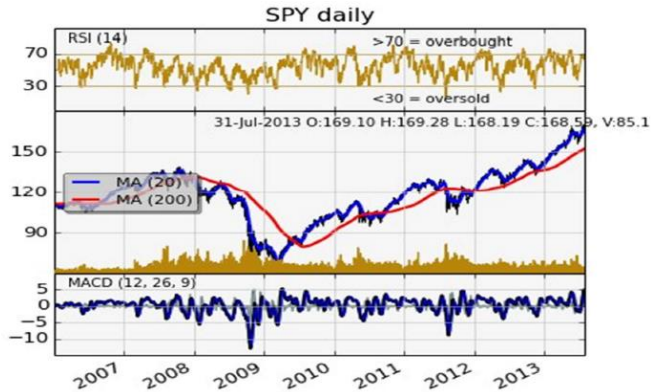
Boxplots



Scatter + Histogram







Data Collection

- **Data Collection**:
  - Very important and time-consuming process
  - Formats: String, text, CSV (comma-separated values), JSON, HTML or XML
- **Data Munging (or Wrangling)**
  - Processing data in various formats like – merging, grouping, concatenating etc. for the purpose of more appropriate and valuable for analysis
- **Data Sources at:**
  - Proprietary data source: Facebook, Google, Amazon etc.
  - Government data sets: www.data.gov or www.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 contribute time and effort for your data

Data Cleaning (-)

❖ Data Cleaning (or Preprocessing): "Garbage In, Garbage Out"

- Distinguishing errors from artifacts
- Data compatibility
- Imputation of Missing values (결손값의 대체)
- Estimating unobserved (zero) counts
- Detection and processing of Outliers and Invalid values
- Categorical Data Encoding
- Scaling or Normalization

❖ Errors and Artifacts

- Errors: 수집 과정에서 원천적으로 빠진 것 (복구 불가능)
- Artifacts: 데이터를 처리하는 과정에서 발생한 문제 (복구 가능)

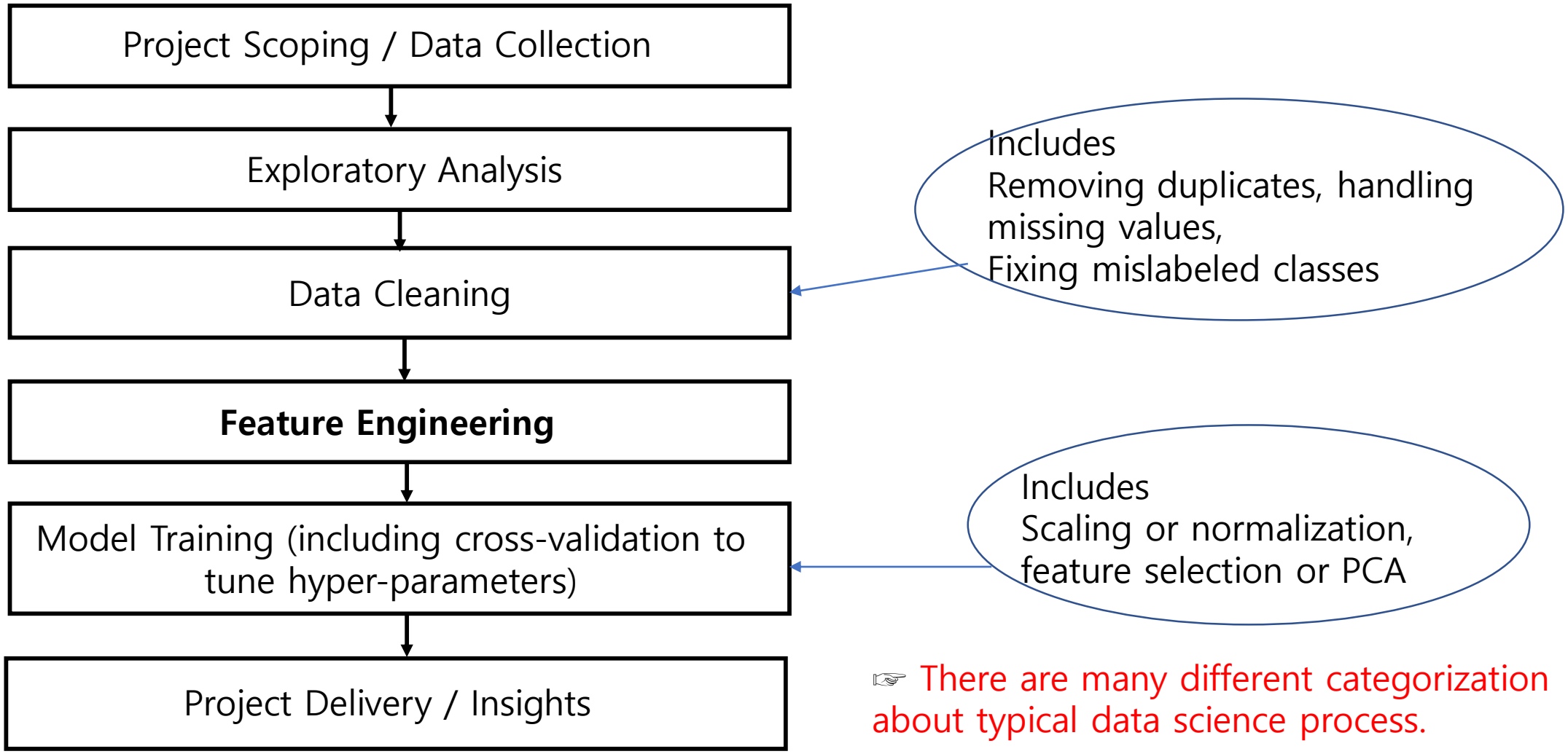
Feature Engineering(+)

- Process of using domain knowledge of the data to create features that make machine learning algorithms work (Wikipedia)
- It is the act of **extracting important features** from raw data and **transforming them** into formats suitable for machine learning.
- Process of feature engineering
  - Brainstorming or testing features
  - Deciding what features to create
  - Creating features (create interaction features, combine sparse classes, add dummy variables for categorical features, remove unused features, etc.)
  - 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 Wrangling (Data Munging)

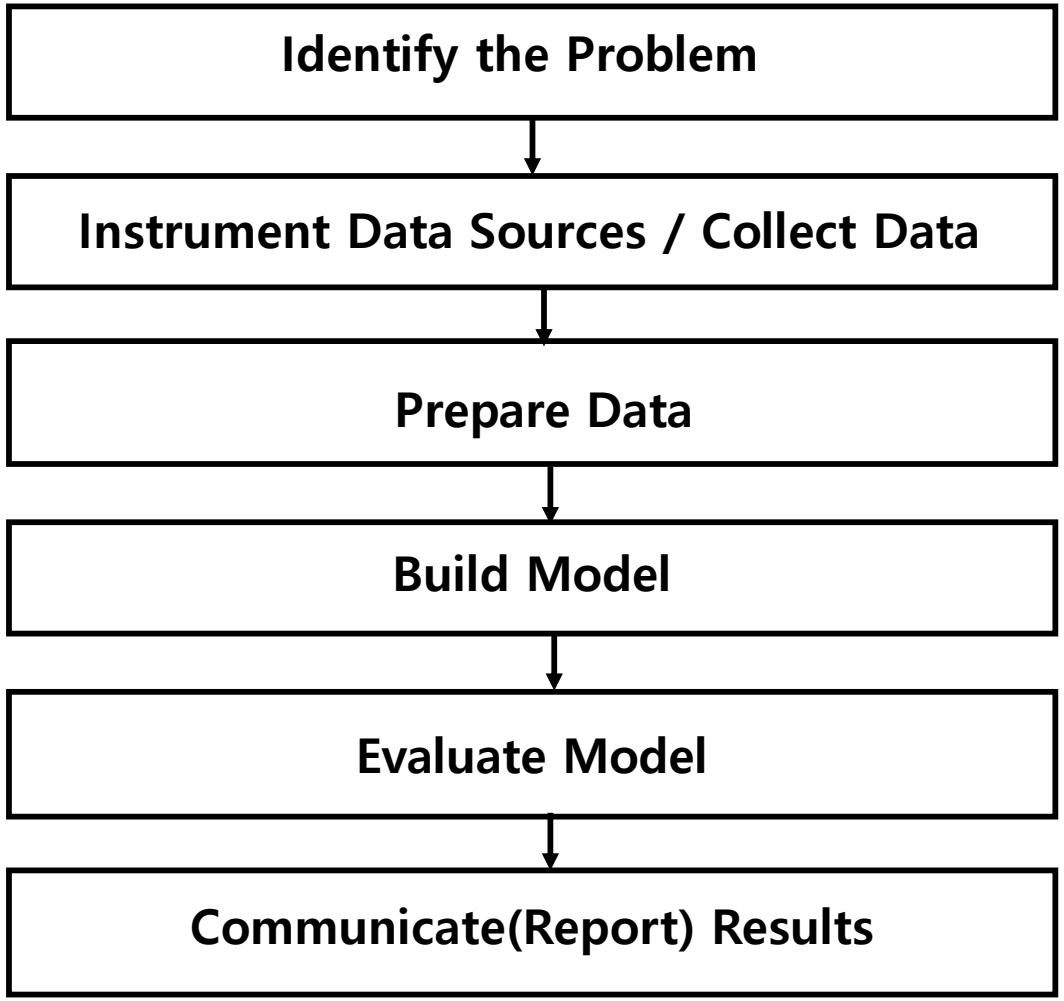
- Process of **transforming and mapping** data from one "raw" data form into another format.
- Data does not always come in a nice format, ready for pd.read_csv or pd.read_excel. So, we will need to perform several tasks in order to get data in the exact format we want.
- Typical tasks:
  - Data acquisition
  - Combining and Merging Data Sets
  - Reshaping and Pivoting
  - Data Transformation
  - Removing duplicates
  - Data Cleaning

Data Science Process (another view from ELITE Data Science)



☞ There are many different categorization about typical data science process.

Data Analysis Model (Jeff Hammerbacher)



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

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

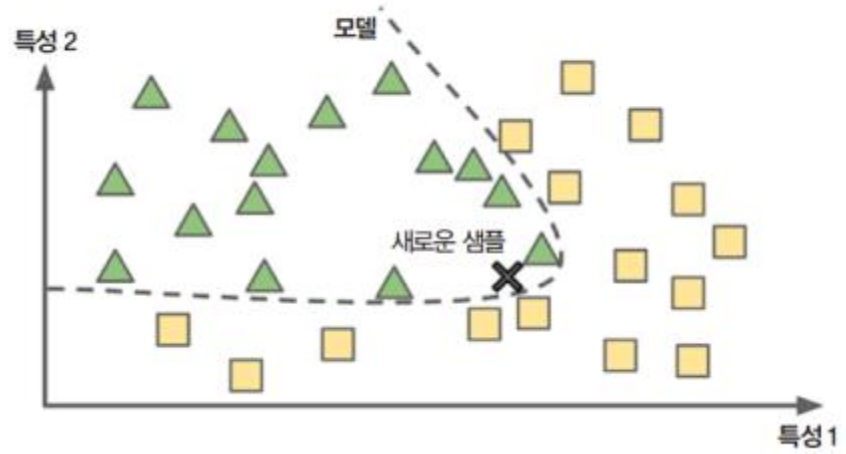
- 실시간 점진적인 학습여부: on-line(온라인학습), batch(off-line, 배치학습)
- 새로운 데이터에 대한 일반화(ex. 예측): instance-based(사례기반), model-based(모델기반)
- (ex) **Clustering**, PCA(Principal Component Analysis), Kernel-PCA

❖ Reinforcement Learning

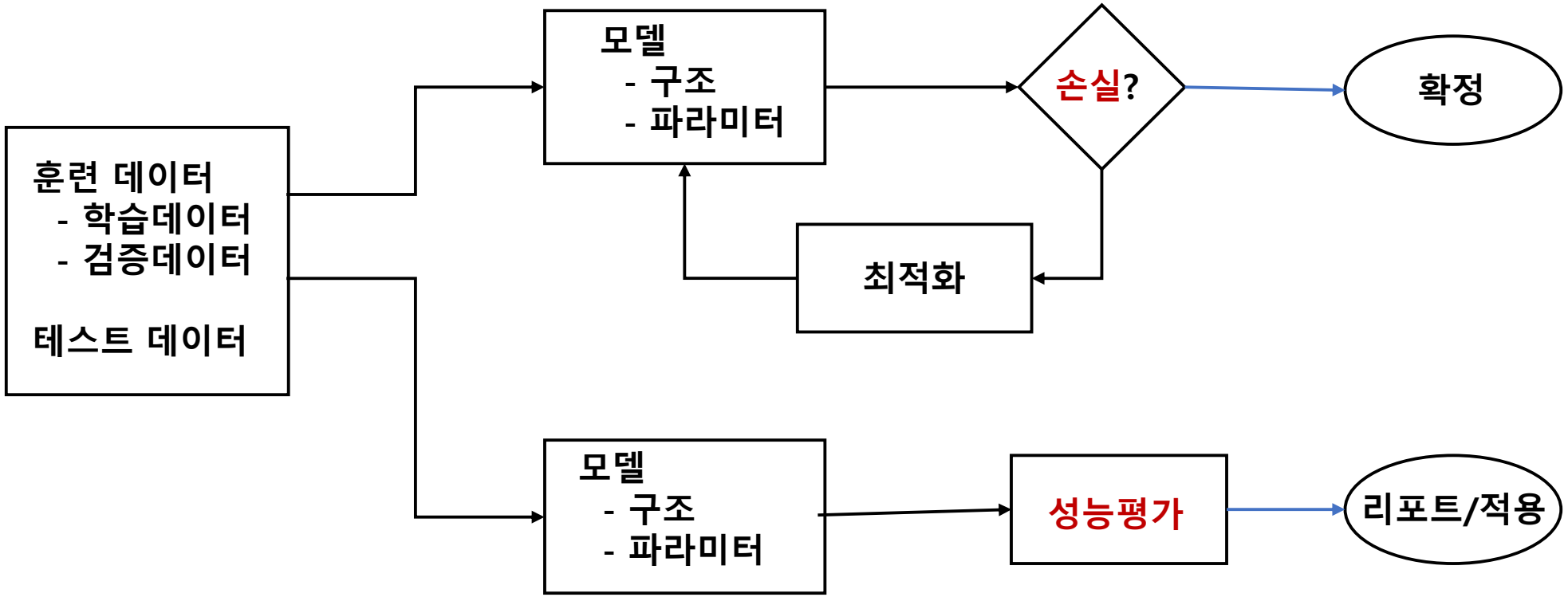
- Reward 와 penalty 를 기반으로 학습

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 is it? (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 is it? (Performance: 성능지표)

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