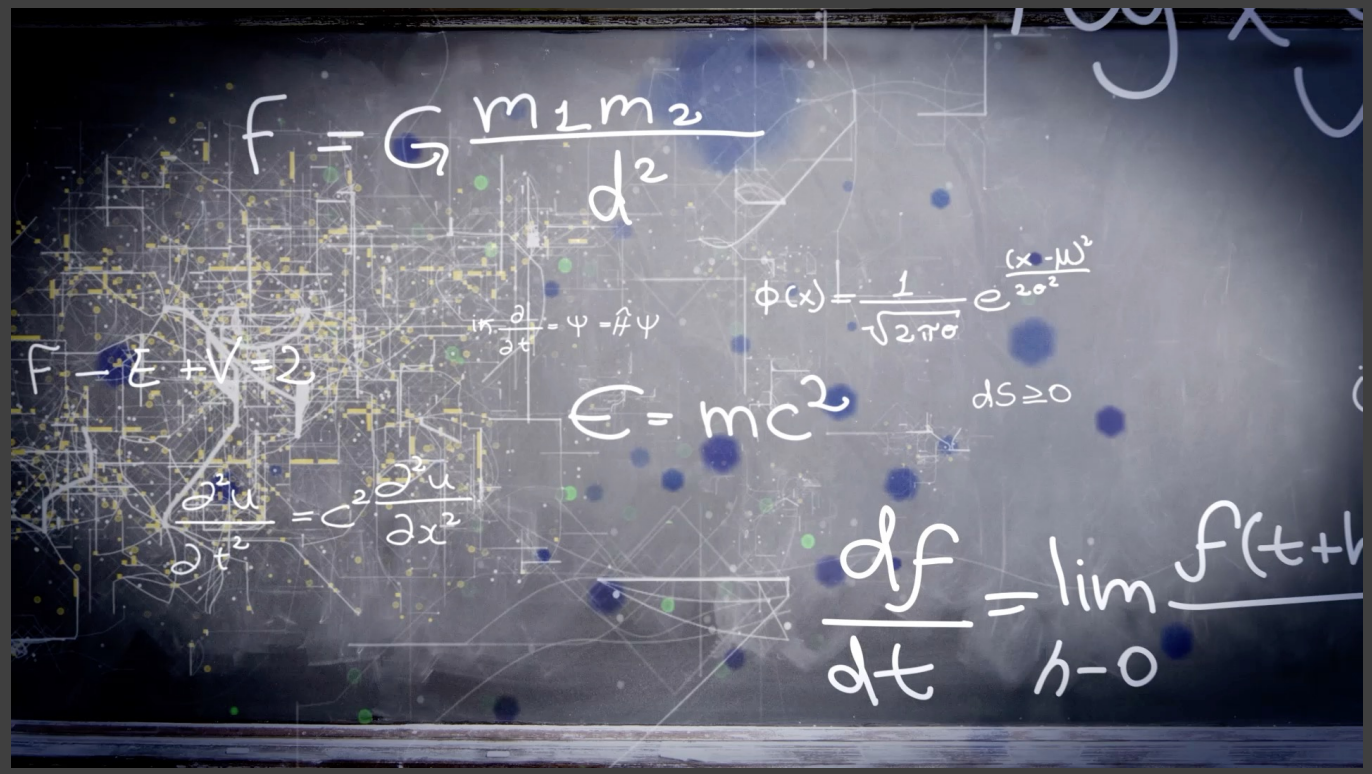
Machine Learning and Financial Applications

Lecture 1
Introduction to machine learning



Liu Peng

liupeng@smu.edu.sg

Pre-class, in-class, and after-class activities

Pre-class reading materials & recordings

Self-paced learning, covering essential contents to be covered in class

Group homework

Takeaway homework to be completed as a group and submitted before the next class

In-class quiz

Reinforce understanding via practice and discussion

Student reflections

Open forum for Q&A and discussions

TA: Brandon
brandon.lim.2019@business.smu.edu.sg
To create telegram group for ease of discussion

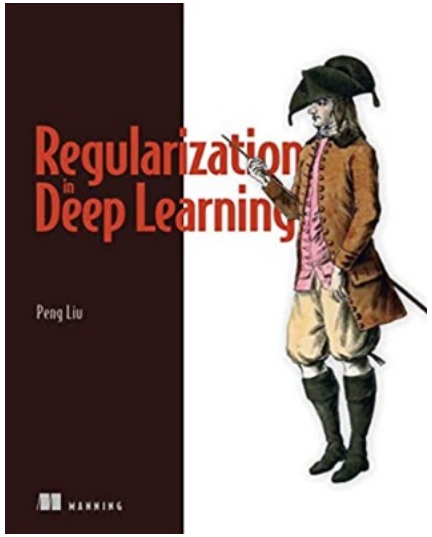
Weekly lesson plan

Session	Topics	Assignments/Activities
1	Introduction to machine learning	
2	Linear regression	Group assignment
3	Logistic regression	
4	Regularization and generalization	Group assignment
5	Decision tree	Project guideline release
6	Deep neural networks	Group assignment
7	Statistical arbitrage with machine learning	
8	Improving trading strategies with Bayesian optimization	Group assignment
9	Clustering and dimension reduction	Project presentation
10	Portfolio Optimization and Reinforcement learning	Project presentation
11	FINAL EXAM (Closed book)	

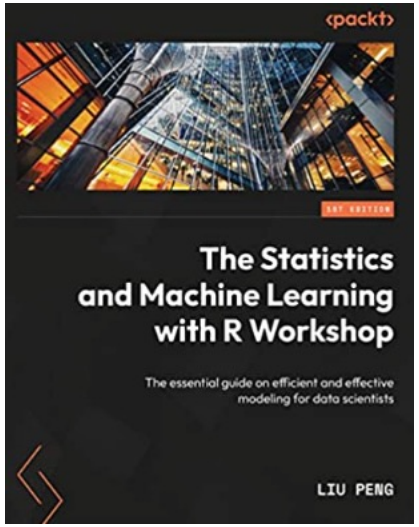
Course assessment rubics

Class participation	Group assignment	Final exam
• 20% • Engagement in discussion, critical thinking	• 40% • Team work, problem solving, oral communication	• 40% • Problem solving, decision making

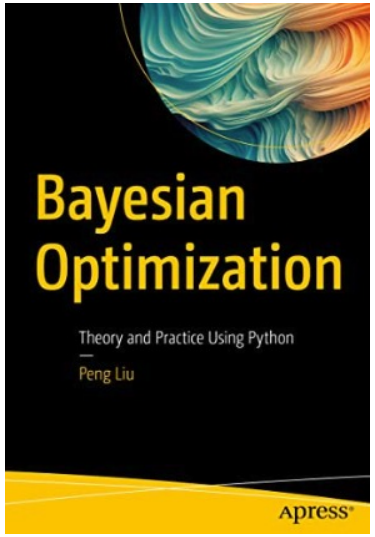
Additional resources



Regularization in Deep Learning



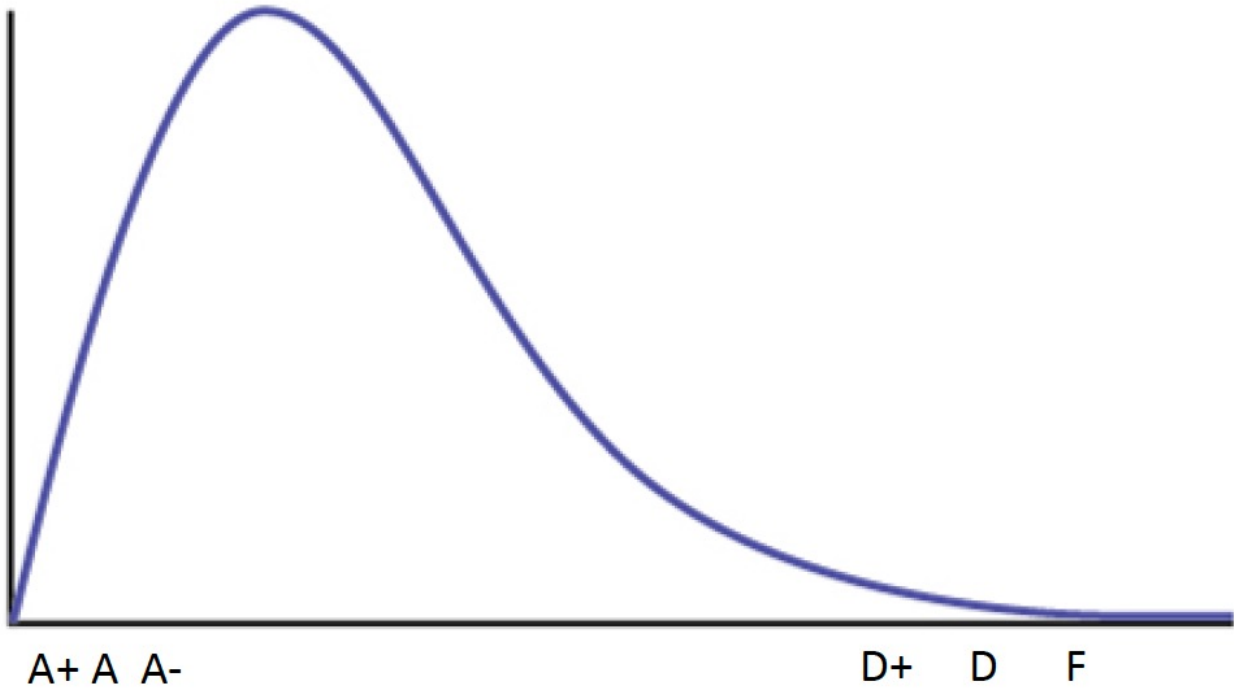
The Statistics and Machine Learning with R Workshop



Bayesian Optimization: Theory and Practice Using Python

Quantitative Trading Strategies with Python (upcoming)

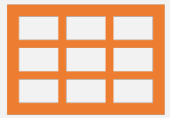
Grading curve



Not drawn to scale

Exact distribution is class-specific and confidential

Office consultation hours



10am-12pm, Tuesday

Please send me an email before you come

Room 5118, LKCSB

Alternatively, can send me an email to book other slots

Quick self-introduction (including me)

- Your name and hobby

- Form class groups

Learning outcomes

Course outlook overview

Ways to engage in learning and discussion

Getting to know machine learning

Understand major ML applications

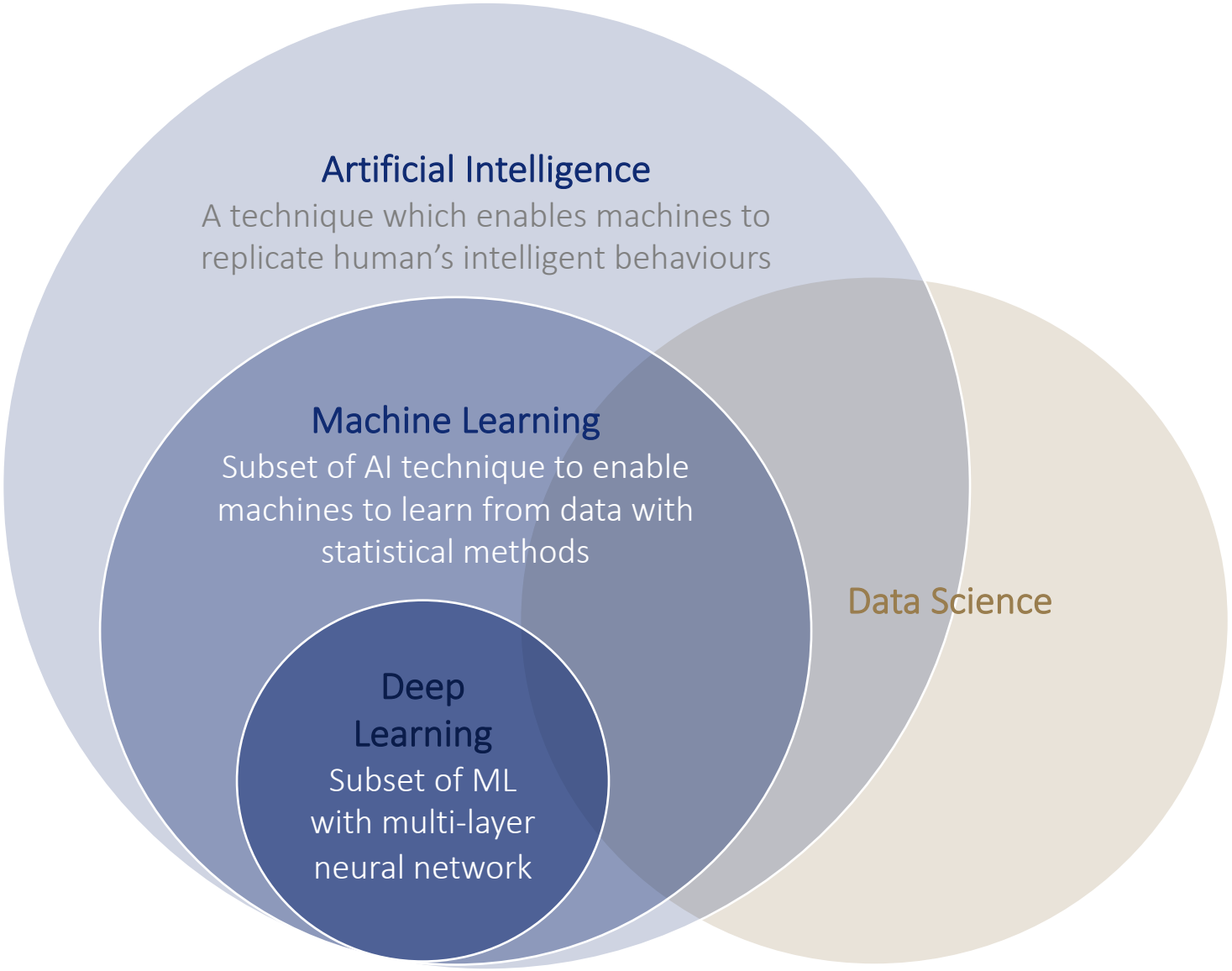
Python basics

Video tutorials

- Introduction to machine learning https://youtu.be/00t53nPpbnU
- Python programming basics https://youtu.be/u5mSDRCoaEo
- Downloading and visualizing stock prices with Python https://youtu.be/ngPjj93B5kE

What is Machine Learning?

- Data Science
- Artificial Intelligence
- Machine Learning
- Deep Learning



Different types of machine learning models

Supervised learning

Given a data set of input-output pairs, learn a function to map inputs to outputs

Classification: supervised learning task of learning a function mapping an input point to a discrete category

Regression: supervised learning task of learning a function mapping an input point to a continuous value

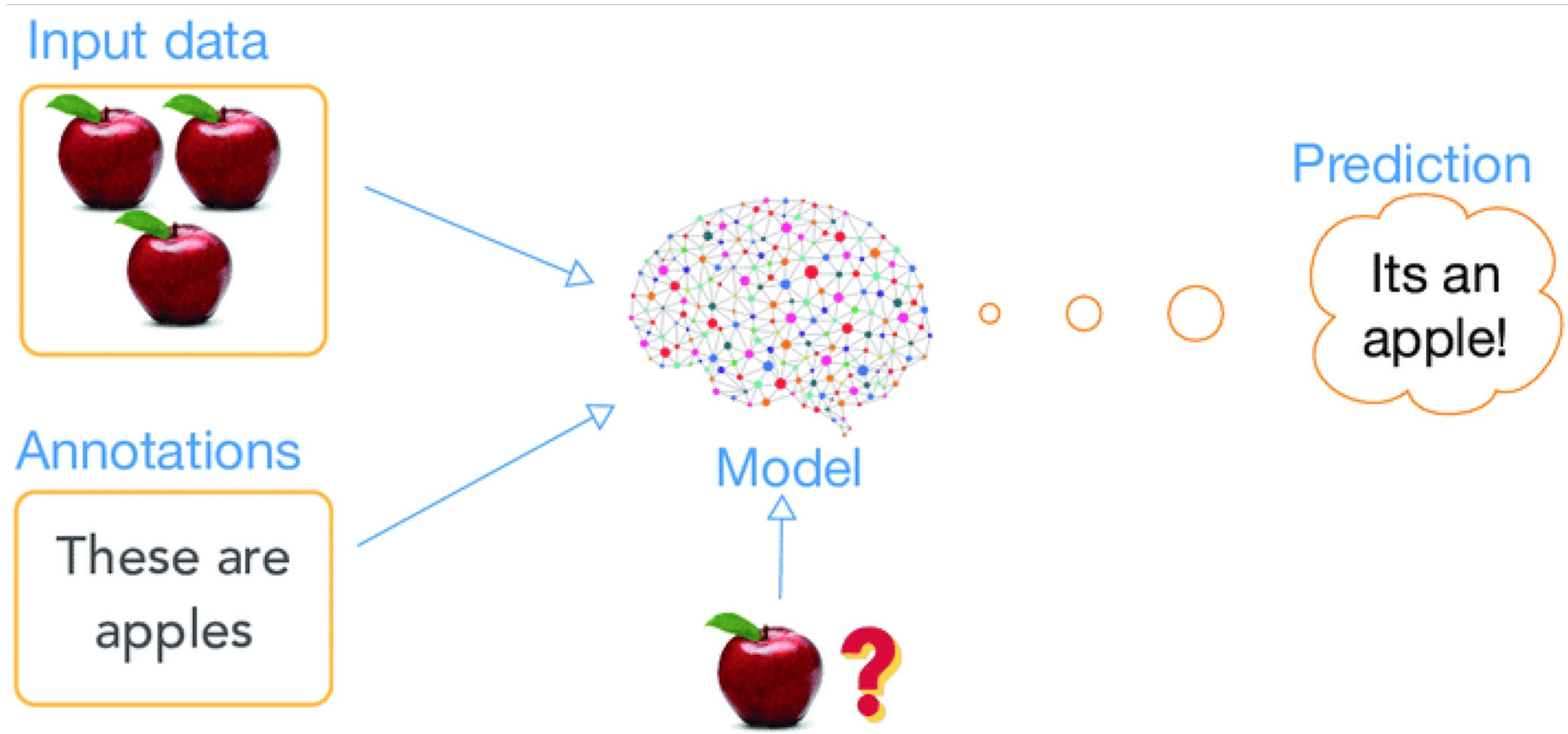
Unsupervised learning

Given input data without any additional feedback, learn patterns

Reinforcement learning

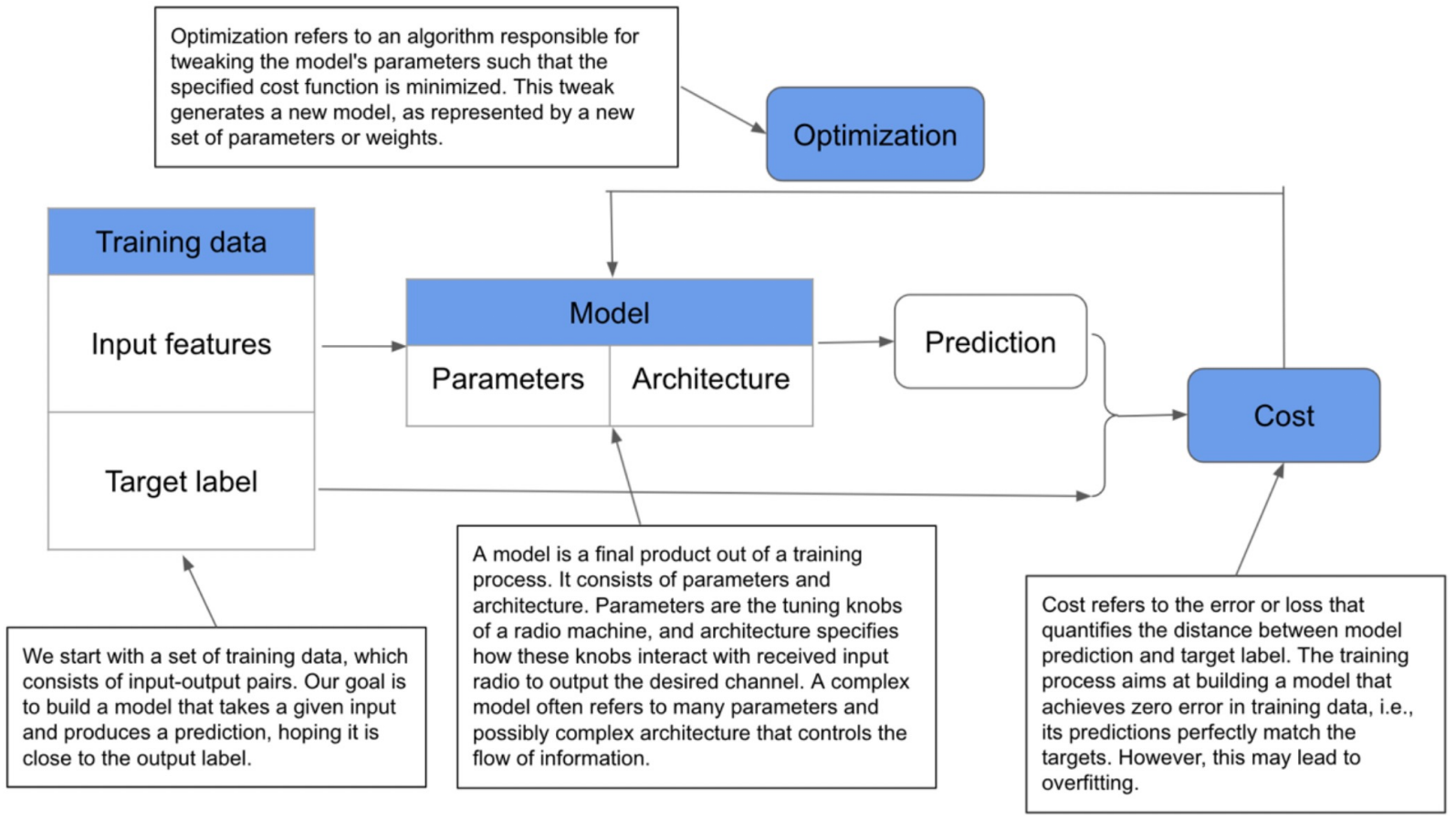
Given a set of rewards or punishments, learn what actions to take in the future

Flavors of Machine Learning: Supervised

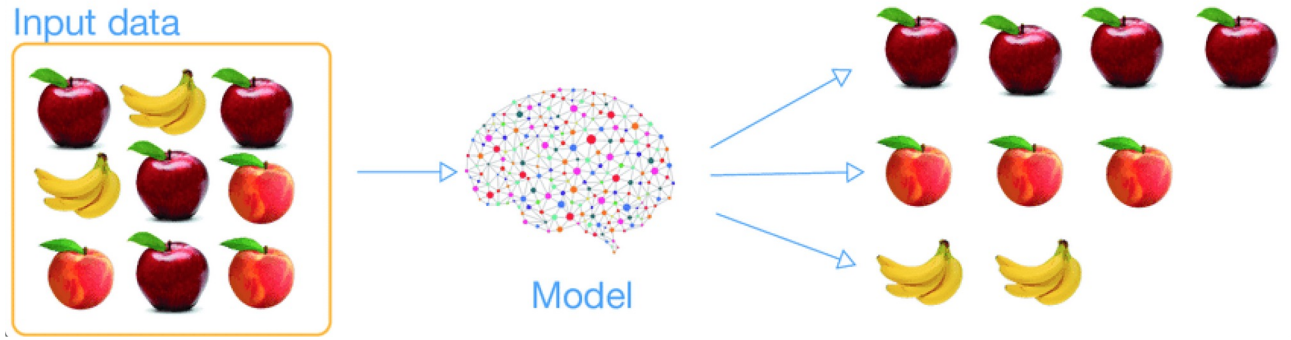


Source: Ma Yang et al.

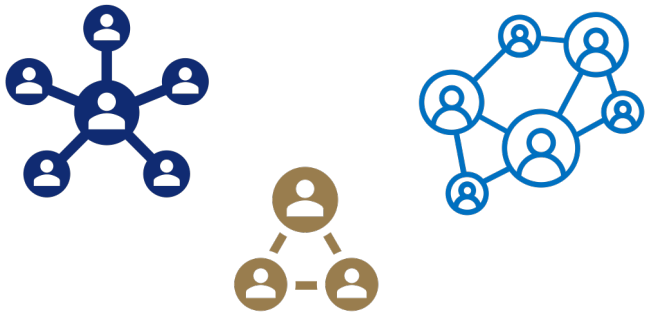
Model training workflow



Flavors of Machine Learning: Unsupervised



Social network analysis

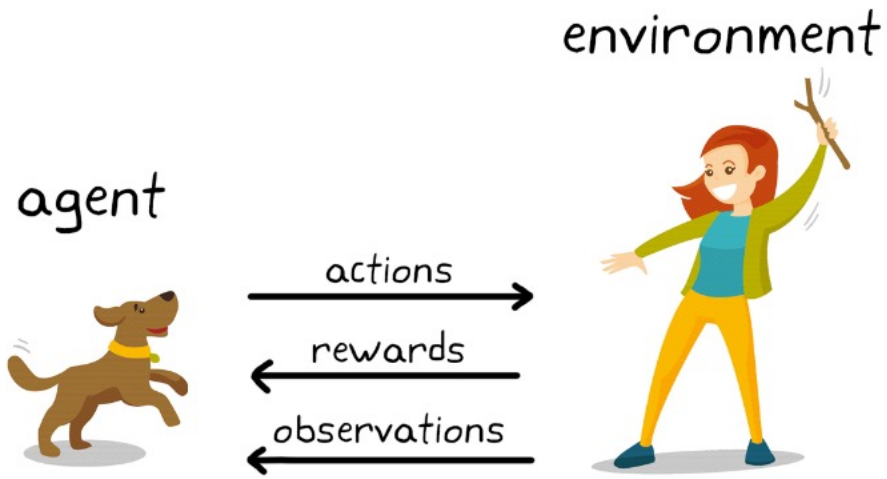


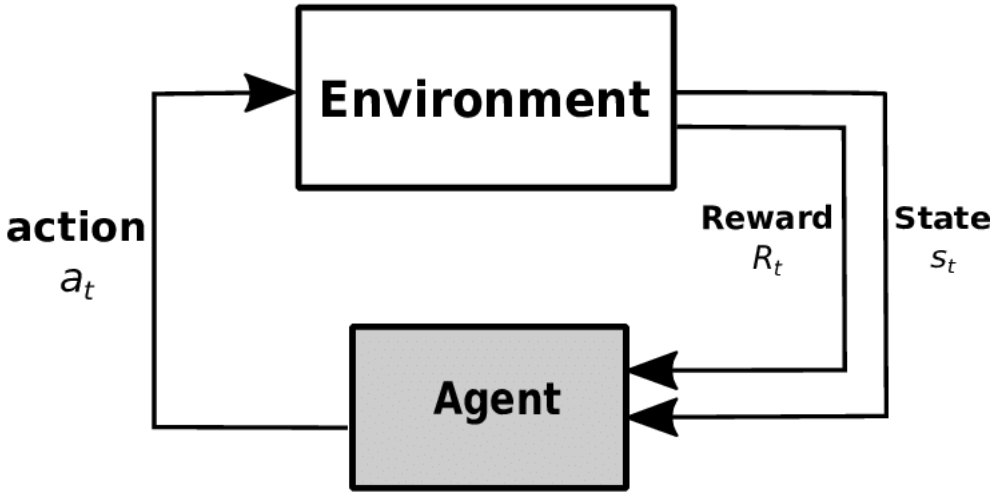
Market segmentation

Organize computing clusters

Source: Ma Yang et al.

Flavors of Machine Learning: Reinforcement



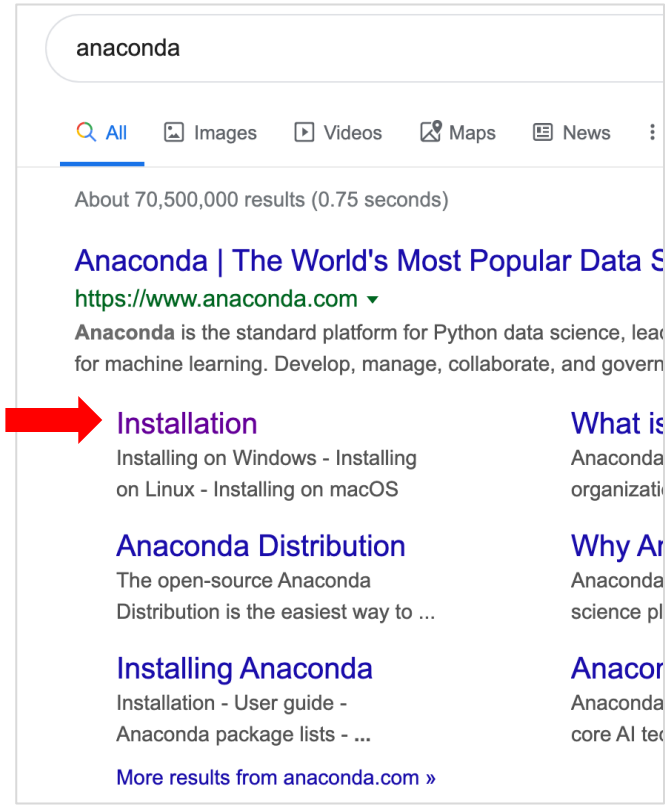


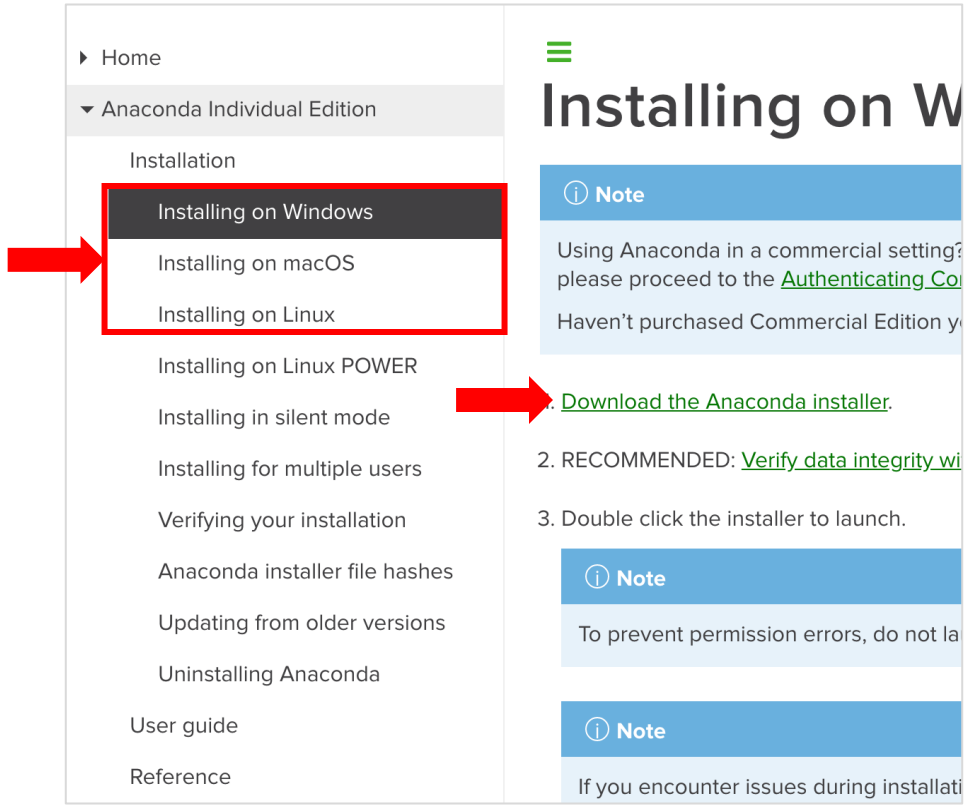
Source: Mathworks

Basics of Python Programming

- Installing Anaconda
- Why Python?
- Jupyter Notebook
- Variable
- Data type
- Control flow
- Function
- Numpy
- Pandas
- scikit-learn

Anaconda





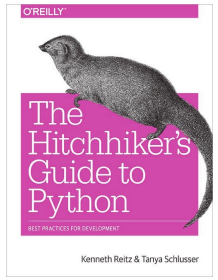
! Install for this laptop
NOT for this user only

Alternatively, use online interfaces without installation:

- https://colab.research.google.com/notebooks/intro.ipynb
- https://repl.it/languages/python3

Why Python?

- Easier to learn - high readability and simple syntax
- Extensive collection of libraries for a vast array of applications
- Supported by a huge community
- Preferred language for **Data Science** and **Machine Learning**

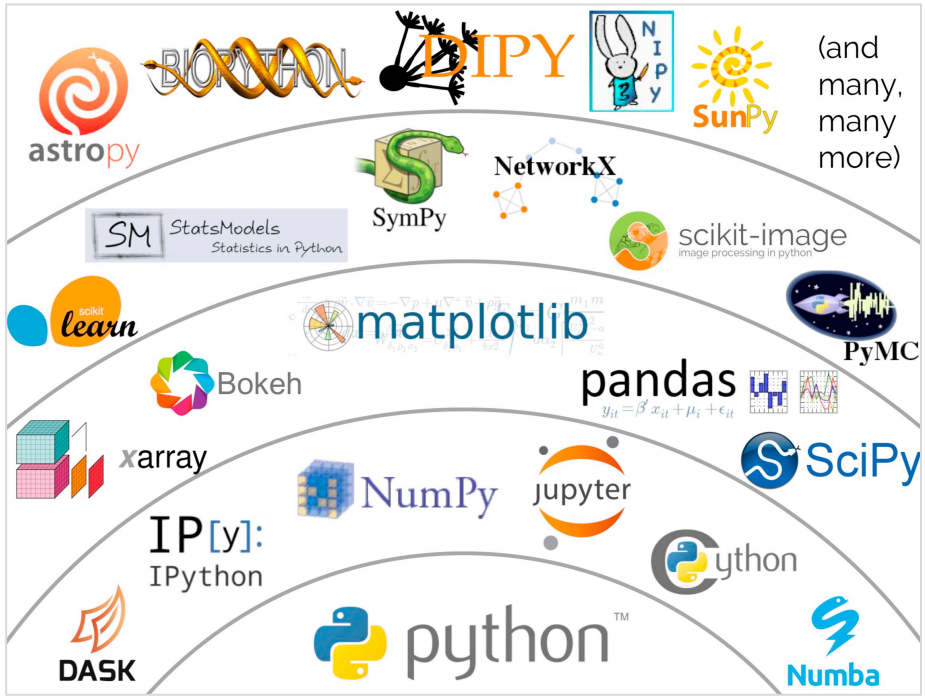


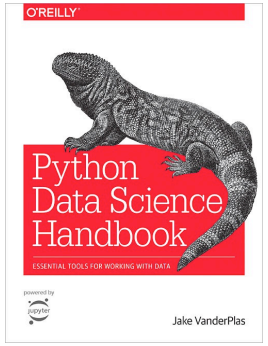
> If you ask Python programmers what they like most about Python, they will often cite its high readability. Indeed, a high level of readability is at the heart of the design of the Python language, following the recognized fact that code is read much more often than it is written.

```
In [1]:  1  print("I love python and machine learning.\n" * 5)
```

```
I love python and machine learning.
I love python and machine learning.
I love python and machine learning.
I love python and machine learning.
I love python and machine learning.
```

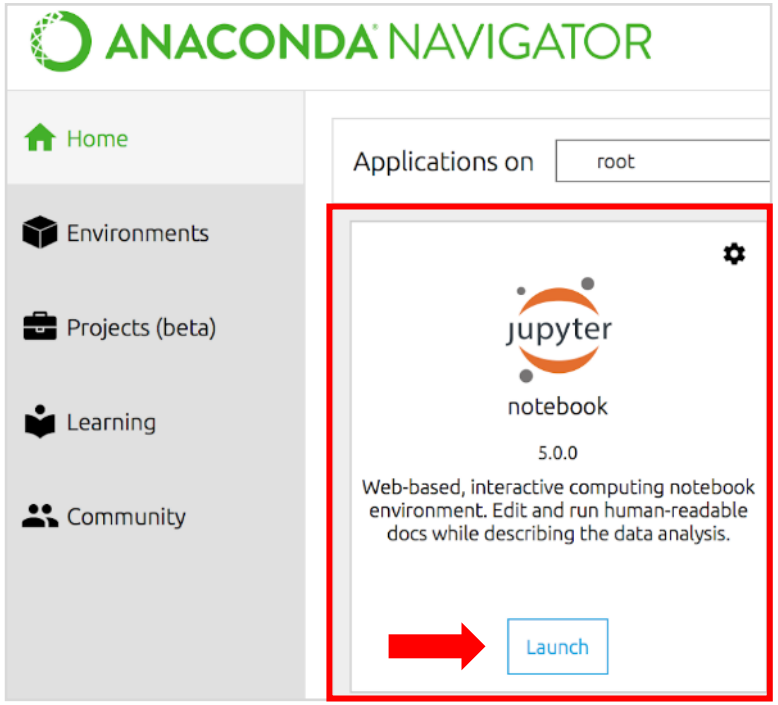
Python has a huge existing collection of libraries for various applications



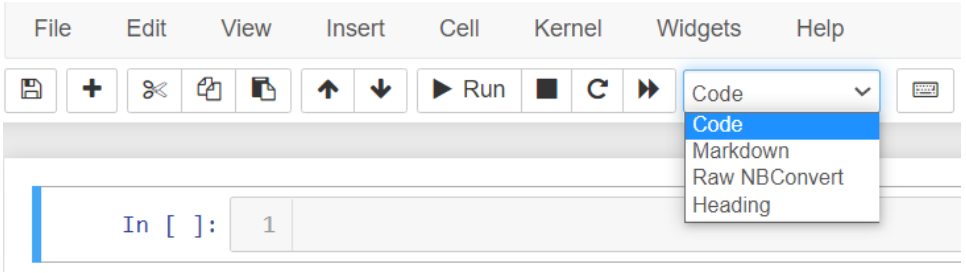


The usefulness of Python for data science stems primarily from the large and active ecosystem of third-party packages: NumPy for manipulation of homogeneous array-based data, Pandas for manipulation of heterogeneous and labeled data, SciPy for common scientific computing tasks, Matplotlib for publication-quality visualizations, IPython for interactive execution and sharing of code, Scikit-Learn for machine learning, and many more tools that will be mentioned in the following pages.

Introduction to Jupyter Notebook



- A cell can be a code cell or markdown cell



- Edit mode: Click in the cell and edit; **green** border

- To run a code cell in edit mode: Ctrl+enter (windows); control+return (Mac)
- Command mode: Click the white space in front of the cell; **blue** border

- Shortcuts in command mode:
  - A/B: insert a cell above/below
  - D,D: delete a cell
  - Z: undo delete a cell

Variable

- A variable is a placeholder in the memory
- **Variable name**: reference to the placeholder
- **Value**: data content
- **Assignment operator =**: put data in the placeholder
- Use www.pythontutor.com to trace your code!
- Restriction on variable name:
  - Only one “word”; no space in between
  - Only letters, numbers and ‘_’
  - Cannot start with a number
  - Avoid using Python keywords

```
1  a = 10
```


```
1  print = 5
2  print(10)
```

What happens to ‘print’ if it is used as a var name?

Data type

Data Type	Example
integer	`x = 1`
float	`y = 1.02`
boolean (True / False)	`likes_TV = True` `likes_sushi = False`
string	`s = 'banana'` `s = "banana"`
list	`name_list = ['John', 'Jane']`
dictionary	`contacts = {'John':66666666, 'Jane':77777777}`
ndarray (from numpy)	n-dimensional array (more later)
DataFrame (from pandas)	excel-like 2D table (more later)

Comparison operators and logical operators

Comparison operators

`>` `<` `>=` `<=` `==`
`!=`

Expressions evaluate to boolean (True/False)

```
1 print(5 >= 5)
```

```
1 print(4 != 4)
```

```
1 a = 2
2 b = 3
3 print(a > b)
```

Logical operators

`and` `or` `not`

- **`and`** returns True only if both conditions are True
- **`or`** returns True if at least one condition is True
- **`not`** returns the reverse

```
1 a = 6
2 b = 5
3 c = 0
4 print(a > b and a == c)
5 print(a > b or a == c)
6 print(not a > b)
7
```

Control flow

```
if condition1:
    body1
elif condition2:
    body2
else:
    body3
```

Inden-tation	`body1` falls under `if condition1`
	`body2` falls under `elif condition2`
	`body3` falls under `else`

3 potential outcomes	`condition1` evaluates as True => `body1` gets executed; the end `condition1` evaluates as False but `condition2` evaluates as True => `body2` gets executed; the end `condition1` and `condition2` evaluate as False => `body3` gets executed; the end

```
val1 if condition else val2
```

2 potential outcomes

- `condition` evaluates as True => `val1`
- `condition` evaluates as False => `val2`

Function

- A function performs one task
- “call” a function with round brackets to trigger it

  `func()`

- System built-in functions
  - **`print()`** displays 1 or more values
  - **`type()`** returns the data type of a value

```
1  age = 40
2  print(age)
```

```
1  age = 40
2  type(age)
```

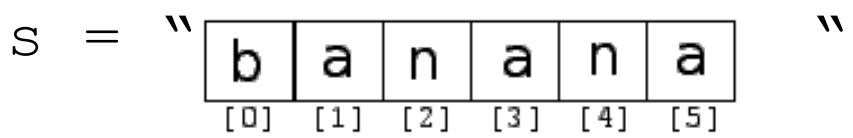
- Some functions take arguments (input values) in the brackets while some do not
  - Both **`print()`** and **`type()`** takes arguments
- Some functions return a result, which can be assigned to a variable
  - **`type()`** returns a result
  - **`print()`** does not

```
1  age = 40
2  x = print(age)
3  print(x)
```

```
1  age = 40
2  x = type(age)
3  print(x)
```

String

- A string can contains characters, symbols and numbers
- Embedded within single quote `'banana'` or double quote `"banana"`



- Index number **starts from 0**
- Use **square bracket** and index number to access a particular character

```
s[index]
```

- `len()` takes a string as argument and returns the **length**, i.e., number of characters
- string can be sliced; start index inclusive, end index **exclusive**

```
s[2:5]
```

- `in`: boolean operator

```
small_str in big_str
```

returns True if small_str is inside big_str

- +: string concatenation

```
s = str1 + str2
```

s is str1 and str2 combined to one string

List

- To store multiple values in a single variable
- Values sit in `[]`, and separated by commas
- Similar to string:
  - Index starts from 0
  - Use `[index]` to access a specific element
  - **`len`**`()` returns number of elements
  - list can be sliced; start index inclusive, end index **exclusive**
  - **`in`**: boolean operator
  - **`+`**: list concatenation

```
name_list = ['John', 'Jane']



name_list[1]
len(name_list)

name_list[0:1]

'Jay' in name_list
s = name_list + ['Jay', 'Joe']
```

Dictionary

- Collection of key-value pairs with **no order**, i.e., no index
- key can be **only string or int**; value can be any data type

If the line is too long, use '\' to break into 2 lines

```
contacts = {'John':66666666, \
'Jane':77777777}
print(contacts['John'])
contacts['John'] = 88888888
contacts['Joe'] = 99999999
```

- Pairs sit in **curly brackets** { }
- Use [ ] to work on the dictionary
  - read existing key's corresponding value
  - replace existing key's value
  - add new key-value pair

Python libraries

- Third party libraries can be installed by executing command from inside the notebook

```
!pip install <package name>
```

- **numpy**: **num**erical **py**thon; foundational library for other Data Science and Machine Learning packages
- **pandas**: mainly used for its 2D table, known as DataFrame
- **matplotlib**: data visualization and plotting
- **scikit-learn**: Machine Learning

```
!pip install numpy
!pip install pandas
!pip install matplotlib
!pip install scikit-learn
```

- Must **import** these libraries before using the data types and functions

```
import numpy as np
```

numpy ndarray

- n-dimensional array in numpy library
- `array()` creates arrays from lists
- 1D array
- 2D array has rows and columns
- 3D array is a collection of 2D array
- Attributes (not functions)
  - `ndim`: number of axes/dimensions
  - `shape`: dimensions of the array
  - `size`: total number of elements
  - `dtype`: data type of elements

```
import numpy as np
arr_1d = np.array([10, 20, 30])
arr_2d = np.array([[10,20], \
[30,40], [50,60]])
arr_3d = np.array([arr_2d, arr_2d])


print(arr_3d.ndim)
print(arr_3d.shape)
print(arr_3d.size)
print(arr_3d.dtype)
```

ndarray – data processing

- Slicing
  - Slicing 1D array is similar to slicing list
  - Slicing 2D by putting index for each dimension, separated by comma
- Statistical functions on the whole ndarray:
  - ndarray methods: `arr.mean(), .max(), .min(), .std(), .var()`
  - Numpy library functions: `np.mean(arr), np.std(arr), np.median(arr), np.percentile(arr, q)`

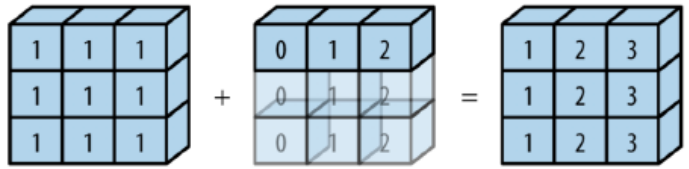
```
arr_1d[0]       arr_1d[0:3]
arr_2d[0, 0]
arr_2d[0:3, 0:2]


arr_1d.mean()
arr_2d.max()


np.mean(arr_2d)
np.std(arr_2d)
np.percentile(arr_2d, 20)
```

ndarray – element-wise operations and broadcasting

- Element-wise operations
  - A value and an array: apply the value to all elements of the array
  - Two arrays of the same shape: apply an operation to each pair of elements at the same position
- Broadcasting
  - Two arrays of different shapes
  - Must match on one dimension
  - Duplicate one array to match the other




```
x = np.array([10, 20, 30])
x2 = x + 10

y = x + x

z = x * x


arr_2d = np.array([[10,20], \
[30,40], [50,60]])
x + arr_2d  # does this work?

arr_2d = np.array([[10,20,30],\
[30,40,50]])

x + arr_2d  # does this work?
```

pandas Series

- Built in on top of numpy ndarray
- One-dimensional array of data with index
- Attributes: `.index`, `.values`

- `index` can be modified to meaningful labels
- Two ways to access an item of a Series
  - Use position-based index
  - Use label-based index
- Slice a Series with a slice of indices
  - Any differences between the two kinds of index slice?

```
import pandas as pd
age = pd.Series([2,4,1])
print(age.index)
print(age.values)
age.index = ['Milo','Coco','Lulu']
print(age)

print(age[0])
print(age['Milo’])

print(age[0:1])
print(age['Milo':'Coco'])
```

pandas DataFrame

- Built in on top of numpy ndarray
- Row: observations / records
- Column: variables / fields
- Can be created from a dictionary

Column headers

	TSLA	FB	GOOG
0	122.72	3220.08	584.76
1	123.08	3203.53	568.82
2	122.94	3186.73	593.38

Row index

- Attributes: `.columns`, `.index`, `.shape`, `.size`

```
# Prices of 3 stocks in 3 days
stock = pd.DataFrame(
{"TSLA": [122.72, 123.08, 122.94],
 "FB":[3220.08, 3203.53, 3186.73],
 "GOOG":[584.76, 568.82, 593.38]})

display(stock)

print(stock.columns)
print(stock.index)
print(stock.shape)
print(stock.size)
```

DataFrame - data processing (1/2)

- Edit the row index to be more meaningful

	TSLA	FB	GOOG
2021-01-01	122.72	3220.08	584.76
2021-01-02	123.08	3203.53	568.82
2021-01-03	122.94	3186.73	593.38

Row index

Column headers

```
stock.index = ['2021-01-01', \
'2021-01-02', '2021-01-03']
display(stock)
```

- Read column(s) with `[ ]`
  - single column name inside `[ ]`; **one column is a Series**
  - list of column names inside `[ ]` ; **multi columns is still a DataFrame**

```
tsla = stock['TSLA']
print(tsla)
subset = stock[['TSLA','GOOG']]
display(subset)
```

DataFrame - indexer loc[] and iloc[]

- **.loc[row selection, col selection]**

  1 row index 1col header

  list of row index list of column header

  slice of row index slice of column header

- All row index and col header: **label-based**, i.e., what you see when display(DataFrame)

```
stock.loc['2021-01-01', 'TSLA']
stock.loc[['2021-01-01','2021-01-03’],\
['GOOG','TSLA']]
stock.loc['2021-01-01':'2021-01-02’,\
'TSLA':'GOOG']
```

- **.iloc[row selection, col selection]**

  1 row index 1col index

  list of row index list of column index

  slice of row index slice of column index

- All row and col index: **position-based**, i.e., not visible when display(DataFrame)

```
print(stock.iloc[1, 1])
display(stock.iloc[[0,2], [0,2]])
display(stock.iloc[0:1, 0:1])
```

DataFrame – conditional ops and boolean indexing

- Conditional operations
  - Element-wise comparisons returns a **boolean DataFrame of the same shape**
  - Element-wise comparisons on a column (i.e., Series) returns a **boolean Series of the same shape**

```
display(stock > 123)


display(stock['TSLA'] > 123)
```

- Boolean indexing with `.loc[]`
  - Placing a condition in `[]` selects the rows that satisfy the condition

```
display(stock.loc[stock['TSLA'] > 123])
```

- Multiple conditions joined by: `&` (logical and), `|` (logical or), `~` (logical not)

```
display(stock.loc[(stock['TSLA'] < 123) \
& (stock['GOOG'] > 590)])
display(stock.loc[(stock['FB'] > 3200) |
(stock['FB'] < 3180)])
```

DataFrame – file read and write

- Export DataFrame to csv or excel

```
stock.to_csv('stock.csv')
```

- Import csv into a DataFrame
- Import excel into a DataFrame
- Add an argument to specify which column to use as row index

```
stock2 = pd.read_csv('stock.csv')
stock3 = pd.read_excel('stock.xlsx')
stock2 = pd.read_csv('stock.csv', \
index_col = 0)
stock3 = pd.read_excel('stock.xlsx', \
index_col = 0)
```

- Ensure the csv or excel is in the same folder as the code (.ipynb file), or the correct file path is given

Windows path

```
stock3 = pd.read_excel( \
'datasets\\stock.xlsx', index_col = 0)
```

Mac path

```
stock3 = pd.read_excel( \
'datasets/stock.xlsx', index_col = 0)
```

Lab session on Python basics

Homework

- Watch/review video tutorials and class recording for week 1 lecture (if you have not done so)
- Post learning reflections and questions if any
- Get to know your teammates and discuss ideas for final project