```
#####################################################
# 2019/04
# Python 3.6.3
# reference:
#####################################################

# Anaconda virtual environments
https://conda.io/docs/user-guide/tasks/manage-environments.html

# create new environment based on "base" and then install packages
# "base" contains basic & pip

>> conda info --envs    # list out all environments
>> conda create --name my_python_env_1 numpy    # Windows
>> conda create --name my_python_env_2 python=3.5 numpy
>> activate my_python_env_2
(my_python_env_2) >>
>> deactivate

>> conda env create -f my_environment.yaml -n my_python_env_3
>> source activate my_python_env_3
(my_python_env_3) >> which pip
(my_python_env_3) >> conda install mypackage
(my_python_env_3) >> conda env export > my_python_env_4.yml
(my_python_env_3) >> conda env export -n my_python_env_3 > my_python_env_5.yml
>> source deactivate
>> conda env remove -n my_python_env_3
>> conda remove -n my_python_env_3 --all    # remove virtual environment

# add my_env to Jupyter Notebook
>> source activate my_env
>> python -m ipykernel install --user --name my_env --display-name
"my_env_4_notebook"

# list all available kernels
>> jupyter kernelspec list

# remove kernel
>> jupyter kernelspec uninstall my_env

"""
# example of a YAML environment file (using latest version of packages if possible)
name: my_python_env
channels:
  - defaults
dependencies:
  - pip
  - python=3.6.7
  - pip:
    - xgboost==0.82
```

```
    - git+ssh://git@gitserver.my.com:my_port/my_folder/my_library.git
"""

>> conda list

# install package
>> conda install elasticsearch
>> conda install elasticsearch=6.3.1=py36_0
>> sudo conda install elasticsearch    # run elevated, not recommend
>> conda uninstall elasticsearch



"""
short cut for Jupyter notebook
"Shift + Enter" or "Alt + Enter"
"Shift" + Tab" for showing document of a function
"""



my_bool = True
my_string = "this is MY string variable"
my_list_nested = [1,2,3,["nested","list"],"a list"]
my_list_numeric = list(range(200,300,3))
my_list_string = ['99','8','51','42','24','23']
my_list_matrix = [[310,311,312],[313,314,315],[316,317,318],[319,320,321]]
my_tuple = (4,5,6,'7',6)
my_list_of_tuple = [(18,19),(51,52),(100,101)]
my_dict = {"hash_table_key":"hash_table_key_value","key":[8,9,10]}
my_dict_numeric = {"value_1":101,"value_2":202,"value_3":303}
my_set = {11,12,13,14,14,14,'14'}



# Python data types: int, float, string

# return remainder of division
10%3

# print formatting
print("{} at {}".format(my_string, 10))
print("{my_parm_1} at {my_parm_2}".format(my_parm_1=my_string, my_parm_2=10))

"""
methods for string
        .lower()
        .upper()
        .split()
"""
```

```
"""
methods for a list object
    .append()
    .pop()    # pop item of a list & change is permanent
    .sort()
    .index()
    # for i, v in enumerate(my_list_numeric):
    #    print(i, v)
"""

"""
dictionary has key/pair values
    .keys()
    .values()
    .items()
"""

# boolean: True / False

# tuple is immutable and can not accept item assignment
my_tuple[0]

"""
set: collection of unique elements
    .add()
"""

# operators: ==, !=, and, or, in

3 in [1,2,3]    # True
3 in my_set

# index in Python starts from 0

# list comprehension
[elem for elem in range(10)]
[(lambda x:x*2)(elem) for elem in range(10)]

# list comprehension using if to filter elements
[(lambda x:x+10)(elem) for elem in range(10) if elem%2 == 0 and elem > 5]

# list comprehension with ternary operator (if/else)
[(lambda x:x+10)(elem) if elem%2==0 else (lambda x:x-1)(elem) for elem in range(10)]

# map function, lambda expression, & filter function
list(map(int, my_list_string))
(lambda x:x+5)(8)    # call a function object returned by lambda expression
list(map(lambda x:x*2, my_list_numeric[0:8]))    # use lambda expression instead of
defining completely a full function
```

```
list(filter(lambda x:x%2 == 0, my_list_numeric))    # filter out for even numbers;
lambda x:x%2 == 0 returns True/False

# zip
my_list_tuples = list(zip(my_list_string, my_list_numeric[:6]))    # zip lists to
tuple pairs

# tuple unpacking
print("first element of each tuple")
for elem_1, elem_2 in my_list_of_tuples:
    print(elem_1)

my_index_and_value = list(enumerate(my_list_numeric))



""" ##################
if my_condition:
    my_actions
elif my_condition:
    my_actions
else:
    my_actions
################## """

""" ##################
for my_count in range(10):
    print(my_count)
################## """

""" ##################
i = 1

while i < 5:
    print("i is {}".format(i))
    i = i + 1

print(i)
################## """

""" ##################
def my_func(my_param="default value"):
    """
    this is a function
    :param my_param: an input string
    :type my_param: ``str``
    :returns: a string
    :rtype: ``str``
    """
```

```
    return my_param
################## """

""" ##################
try:
    x = int(input("Please enter a number: "))
    break
except ValueError as errors:
    print(errors)
################## """

# my_func : <function __main__.my_func>
# Ask Python what's this object? That's an object, an main function called my_func



######## NumPy ########

# NumPy array (1-D vector / 2-D matrix)
import numpy as np
my_np_array = np.array(my_list_matrix)    # create array using a Python object -
list

# multiple ways to create NumPy arrays
np.arange(25)
np.arange(0, 9, 2)
np.zeros((3, 2))    # tuple(row, column)
np.ones(10)
np.linspace(0, 10, 3)
np.eye(3)    # identity matrix

# numpy.random package
np.random.rand(4)    # from uniform distribution from 0 to 1
np.random.rand(2, 2)
np.random.randn(5)    # standard normal distrbution center at 0
np.random.randint(5, 10, 2)    # a random number from inclusive lower bound to
exclusive upper bound

# attribute & method
my_np_array.reshape(6, 2)    # re-shape method
my_np_array.dtype
my_np_array.max()    # max method
my_np_array.min()
my_np_array.sum(axis=0)    # sum of all columns
my_np_array.argmin()    # return index location of min value; .argmax()
my_np_array.reshape(2, 6).shape

from numpy.random import randint    # simplify typing
randint(2, 10)    # versus typing np.random.randint(2, 10)
```

```
"""
NumPy arrary is different from list and it can broadcast
Changes will be in original array!
An example below
"""
my_np_array_2 = np.arange(10)
print(my_np_array_2)
my_np_array_2_sliced = my_np_array_2[4:8]
my_np_array_2_sliced[:] = 99    # [:] is the key to create this
print(my_np_array_2_sliced)
print(my_np_array_2)    # value of my_array is changed too!

# to avoid the above situation, use copy method to copy an array
my_np_array_2_copied = my_np_array_2.copy()

# slicing
# my_2d_arr[i,j] is equivalent to my_2d_arr[i][j] where i is row and j is column

# conditional selection
my_np_array_2[my_np_array_2 > 5]    # inside brackets is boolean array

my_np_array_2/my_np_array_2
1/my_np_array_2
# NumPy operations: +, -, *, /
# inf (1/0) & nan (0/0)
# np.nan
# np.nan == np.nan    # False
# np.isnan(np.nan)
# pd.isnull(np.nan)
# type(np.nan)    # float

"""
np.sqrt(my_np_array_2)
    .exp()
    .max()
    .sin()    # sin(90 degrees) = np.sin(math.radians(90)) =
math.sin(math.radians(90)) = 1
    .log(np.exp(3))
    .log10()
"""

# sigmoid function 1 / (1 + e^(-x))
import matplotlib.pyplot as plt
%matplotlib inline
x = np.linspace(-100, 100)
y = 1 / (1 + np.exp(-x))
plt.plot(x, y)
```

```
######## pandas ########

# pandas; data types: Series (index by label), DataFrame
# pandas.core.series.Series, pandas.core.frame.DataFrame
# Series can hold a variety of object types (functions as well)
# NaN
# Numpy and pandas will always convert numeric to float

import pandas as pd
pd.Series(data = my_list_numeric[0:3])
my_list_label = ["Label a","Label b","Label c"]
my_pd_Series = pd.Series(np.array([1,2,3]), my_list_label)
my_pd_Series.dtype
pd.Series(my_dict_numeric)

my_pd_Series_1 = pd.Series([1,2,3,4], ["US","Germany","USSR","Japan"])
my_pd_Series_2 = pd.Series([10,20,30,40], ["US","Germany","Italy", "Japan"])
my_pd_Series_1["US"]
my_pd_Series_2[0]
my_sum = my_pd_Series_1 + my_pd_Series_2    # if index has not match, it will return
NaN
print(my_sum)
"""
pandas.core.series.Series
    .unique()
    .nunique()    # number of unique values
    .value_counts()
"""


np.random.seed(101)
df = pd.DataFrame(np.random.randn(5,4), index=["A","B","C","D","E"],
columns=["W","X","Y","Z"])
df[["X","Z"]]    # pass a list of column names
df["New"] = df["X"] + df["Y"]
df.loc["E"]    # using bracket but not parenthese
df.iloc[0]    # index position
df.loc[["B","C"],["W","Y"]]    # df.loc[["row"],["column"]]
df[df['W']>0][['Y','X']]
df[(df['W']>0) & (df['Y']>1)]    # cannot use "and"; "and" can only take single
boolean type a time but not a Series of boolean values; () is important too
df.reset_index()
df['states'] = "IN VA CA WA TX".split()
df.set_index('states', inplace=True)
df.shape    # index 0 is row, index 1 is column
df.drop("TX")
df.drop("New", axis = 1)    # inplace = True/False avoid accidentally lose data

# MultiIndex
outside = ["G1","G1","G1","G2","G2","G2"]
inside = [1,2,3,1,2,3]
```

```
hierarchy_index = list(zip(outside,inside))
hierarchy_index = pd.MultiIndex.from_tuples(hierarchy_index)
df = pd.DataFrame(np.random.randn(6,2), hierarchy_index, ["column_A","column_B"])
df.index.names=["Group","Num"]
df.loc['G1'].loc[1]
df.xs(2,level="Num")    # grab data with Num == 2; xs: cross section


# missing data
# pandas missing data: null or NaN
df =
pd.DataFrame({'column_a':[1,2,np.nan],'column_b':[5,np.nan,np.nan],'column_c':[1,2,3
]})

df.dropna()    # drop rows which have one or more missing values
df.dropna(axis=1)    # drop columns which have one or more missing values
df.dropna(thresh=2)    # drop rows which have less than two elements have values
df.fillna(value="fill values")
df["column_a"].fillna(value=df["column_a"].mean())

# groupby
my_dict_data = {'Company':['A','A','B','B','B','C','C','C','C'],
                'Name':['AA1','AA2','BB1','BB2','BB3','CC1','CC2','CC3','CC4'],
                'Number':[100,300,500,700,900,250,150,350,50]}
df = pd.DataFrame(my_dict_data)
df.groupby('Company').mean().loc['B']    # .count()
df.groupby('Company').describe().transpose()['C']    #
df.groupby("Company").describe().loc['C']

# concatenate/merge/join
my_df_1 = pd.DataFrame({"score_math":[99,77,33,66,88],
"average_math":[50,51,52,53,54], "subject_id_key":[0,1,2,3,4]}, index=[0,1,2,3,4])
my_df_2 = pd.DataFrame({"score_cs":[10,9,8,7,6], "average_cs":[8,7,6,5,4],
"subject_id_key":[4,5,6,7,0]}, index=[4,5,6,7,0])
pd.concat([my_df_1,my_df_2], axis=1)    # use "index" to concat by column
# pd.merge(left, right, how="inner", on="key")    # left, right, inner, outer
pd.merge(my_df_1, my_df_2, how="inner", on="subject_id_key")    # use "key" to merge
# difference between "merge" and "join" is that using "join" the key you wanna join
on is on "index" instead of a column ("merge")
# re-create data sets by removing subject_id_key
my_df_1 = pd.DataFrame({"score_math":[99,77,33,66,88],
"average_math":[50,51,52,53,54]}, index=[0,1,2,3,4])
my_df_2 = pd.DataFrame({"score_cs":[10,9,8,7,6], "average_cs":[8,7,6,5,4]},
index=[4,5,6,7,0])
my_df_1.join(my_df_2)
# my_df_1.join(my_df_2, lsuffix='_left')    # would work if subject_id_key isn't
removed

# operations
def my_func(x):
```

```
    return x*2
df['Number'].apply(my_func)
df['Number'].apply(lambda x: x*2)
"""
pandas.core.frame.DataFrame
    .loc[]    # df.loc[0:5,["my_column_1","my_column_2"]]
    .iloc[]    # df.iloc[0:2,0:2]
    .xs
    .index    # df.index = [10,11,12]
    .index.name
    .reset_index()
    .set_index()    # df.set_index("my_existing_column")
    .shape
    .drop('my_column', axis=1)    # drop a column
    .dropna(thresh=3)
    .fillna(value="my default value")
    .apply()
    .assign(my_new_field = df["my_str_field"].apply(len))
    .columns    # attributes of data frame itself; df.columns =
["my_column_1","my_column_2"]
    .mean()
    .describe()
    .info()
    .head()    # .head(n=5)
    .tail()
    .sort_values()    # df.sort_values("my_column")
    .isnull()
    .transpose()
    .idxmax()    # return index where max value locates; df["my_column"].idxmax();
similar to .argmax() in NumPy
"""
df.pivot_table(values="Number", index="Company")    # index is group
df.pivot_table(values="Number", index=["Company","Name"])
df.pivot_table(values="Number", index="Company", columns="Name")

# I/O: pandas.read_* & DataFrame.to_*
df = pd.read_csv("my_input_file_data.csv", keep_default_na=False,
na_values=["my_custom_missing_value"])
# By default the following values are interpreted as NaN: ‘’, ‘#N/A’, ‘#N/A N/A’, ‘
#NA’, ‘-1.#IND’, ‘-1.#QNAN’, ‘-NaN’, ‘-nan’, ‘1.#IND’, ‘1.#QNAN’, ‘N/A’, ‘NA’, ‘NULL
’, ‘NaN’, ‘n/a’, ‘nan’, ‘null’.
# na_filter
df.to_csv("my_output_file.csv", index=False)

# SQL
"""
PostgreSQL: psycopg2
MySQL: PyMySql
SQLite:
"""
```

```
from sqlalchemy import create_engine
my_engine = create_engine("sqlite:///:memory:")
df.to_sql("my_table_example", con=my_engine)
df_sql = pd.read_sql("my_table_example", con=my_engine)

# libaries for SQL/HTML: sqlalchemy, lxml, html5lib, BeautifulSoup4

pd.read_html("https://en.wikipedia.org/wiki/Federal_Reserve_Economic_Data")
pd.read_html("https://www.nasdaq.com/symbol/spy/historical")[2]



######## examples of reading/writing CSV & JSON files ########

# option 1
import csv
with open("my_input_file.csv", encoding="utf-8") as csvfile, open("my_output.csv",
"w", newline="") as f_output:
    reader = csv.DictReader(csvfile)    # csv.DictReader object
    my_header = reader.fieldnames

    writer = csv.DictWriter(f_output, fieldnames=my_header)
    writer.writeheader()
    for row in reader:
        print(row["column_1"])
        print(type(row))    # collections.OrderedDict
        writer.writerow(row)

# option 2
import csv
with open("my_file.csv", encoding="utf-8") as csvfile, open("my_output.csv", "w",
newline="") as f_output:
    reader = csv.reader(csvfile, delimiter="\t")
    header = next(reader)
    writer = csv.writer(f_output, delimiter="\t")
    writer.writerow(header)
    for row in reader:
        print(type(row))    # list
        print(row[2])
        writer.writerow(row)

# option 3
# if "\n" appears in data, for example - "column_1 column\n2 column_3", option 3
won't read data correctly
with open("my_file.csv", "r", encoding="utf-8") as f:
    my_header = next(f).rstrip().split(",")
    for line in f:
        print(line.rstrip())
        print(type(line.rstrip()))    # str
```

```
# option 4
import pandas as pd
pd.read_csv("my_file.csv", sep="\t", dtype="object", encoding="utf-8",
keep_default_na=False, na_values=["my_custom_missing_value"])

# json.dumps (option 5 - string) versus json.dump (option 6 - file)
# option 5 - json string
import json

my_list_key = ["column_1", "column_2"]
my_list_value = ["value_1", 100]
json.dumps(dict(zip(my_list_key, my_list_value)))

my_input_data = json.load(open(my_input.json))

# option 6 - jsonl file
import json

level_3 = [{"field_3": "values_3", "field_4": "values_4"}, {"field_3": "values_5",
"field_4": "values_6"}]
level_2 = {"field_1": "values_1", "field_2": level_3}
my_json_level_1 = {"key_1": [level_2]}

with open("my_jsonl_output.jsonl", "w") as f_output:
    json.dump(my_json_level_1, f_output)
    f_output.write('\n')
```