Notebook1 - Data Manipulation in Python

在这个Notebook中将介绍利用Python编程语言做数据处理操作的基本方法。内容包括：

1. Python Basic Data Types
2. Python Container Types
3. Numpy - Python Scientific Computing Library
4. Pandas - Python Library for Data Manipulation and Analysis
5. Load and Save Data on Google Colaboratory

参考官方文档:

- Python library文档： https://docs.python.org/3.10/library/index.html (https://docs.python.org/3.10/library/index.html)
- Python Numpy: https://numpy.org/doc/stable/reference/index.html (https://numpy.org/doc/stable/reference/index.html)
- Python Pandas: https://pandas.pydata.org/docs/reference/index.html (https://pandas.pydata.org/docs/reference/index.html)

Part1 Python Basic Data Types

- Python basic types include number (integer, float), boolean, and string

1.1 Number

In [ ]:

```
x = 1

print(type(x))

x += 1 # 注意: Python中不存在 x++ 或者 x--这种用法
print(x)

x = x ** 2
print(x)

y = 1.1
print(type(y))

y += x
print(y)
print(type(y))
```

```
<class 'int'>
2
4
<class 'float'>
5.1
<class 'float'>
```

1.2 Boolean

In [ ]:

```
a = True # 注意：这里True, False必须首字母大写，不能用true false
b = False

print(type(a))

# python逻辑运算: 或 与 有两种写法, 都可以用
print(a and b)
print(a & b)

print(a or b)
print(a | b)

print(not a)
print(a != b)
```

```
<class 'bool'>
False
False
True
True
False
True
```

1.3 String

In [ ]:

```
x1 = 'testa'
x2 = "testb" #使用单引号或者双引号都可以，但是需要配套一致
print(type(x1))

x3 = 'Try ' + x1 + ' and ' + x2
print(x3)

# Some useful string methods
print(' common use    '.strip())
print(x1.upper())
print(x1.replace('e', '(e)'))
```

```
<class 'str'>
Try testa and testb
common use
TESTA
t(e)sta
```

In [ ]:

```
# String format print
print('{} is a, and {} is b'.format(x1, x2))
print('{space1} is a, and {space2} is b'.format(space1 = x1, space2 = x2))
```

```
testa is a, and testb is b
testa is a, and testb is b
```

In [ ]:

```
# Triple quotes for multi-line strings
x4 = '''
testa

testb
'''
print(x4)
```

```

testa

testb

```

Part2 Python Container Types

- Python container types include list, tuple, dictionary, set

2.1 List

In [ ]:

```
my_list1 = [3,2,4]
print(type(my_list1))
print(my_list1)

my_list2 = [3,'2',4.1] # we can create a list with values in different data ty
print(my_list2)

my_list1[2] = True
print(my_list1)
```

```
<class 'list'>
[3, 2, 4]
[3, '2', 4.1]
[3, 2, True]
```

In [ ]:

```
my_list1.append(1)
print(my_list1)
print(my_list2.append(1)) # 思考：为什么打印出的结果是None?

my_list1.pop(1) # move the element with index 1
print(my_list1)
```

```
[3, 2, True, 1]
None
[3, True, 1]
```

In [ ]:

```
# List Index
print(my_list1[1])
print(my_list1[-1])
print(my_list1[-3])
```

```
True
1
3
```

In [ ]:

```
# List Slicing
my_list3 = list(range(1,9,2)) # 具体用法 https://docs.python.org/3.10/library/s
print(my_list3)

print(my_list3[2:])
print(my_list3[:3]) # 从首个元素开始，到index3结束，不包括 index 3
print(my_list3[:-1]) # 从首个元素开始，到最后一个元素结束，不包括最后一个元素
print(my_list3[-1:])
print(my_list3[-3:-1])
print(my_list3[::1]) # index step 1
print(my_list3[::2]) # index step 2
print(my_list3[::-1]) # 逆序
```

```
[1, 3, 5, 7]
[5, 7]
[1, 3, 5]
[1, 3, 5]
[7]
[3, 5]
[1, 3, 5, 7]
[1, 5]
[7, 5, 3, 1]
```

In [ ]:

```
# List Loop
my_list4 = list(range(1,10,2))

for ele in my_list4:
  print(ele)
```

```
1
3
5
7
9
```

In [ ]:

```
# enumerate() function: bind index with list element
for idx, ele in enumerate(my_list4):
  print(idx, ele)
```

```
0 1
1 3
2 5
3 7
4 9
```

In [ ]:

```
# list comprehension: simplify your code when creating list
print(my_list4)
my_list5 = [x ** 2 for x in my_list4]
print(my_list5)

my_list6 = [x ** 2 for x in my_list4 if x % 3 == 0]
print(my_list6)
```

```
[1, 3, 5, 7, 9]
[1, 9, 25, 49, 81]
[9, 81]
```

2.2 Tuple

In [ ]:

```
t = (1,2)
print(type(t))
print(t[0])
```

```
<class 'tuple'>
1
```

In [ ]:

```
# tuple is immutable vs. list is mutable

# t[0] = 1

l = [1,2]
l[0] = 10
print(l)
```

```
[10, 2]
```

2.3. Set

In [ ]:

```
# A set is an unordered collection of distinct elements.
my_set1 = {'a', 'b'}
print(type(my_set1))
print(my_set1)


my_set2 = {'a', 'a' ,'b'}
print(my_set2)
```

```
<class 'set'>
{'b', 'a'}
{'b', 'a'}
```

In [ ]:

```
# You can NOT access set element by index
# print(my_set2[1])
for item in my_set2:
  print(item)
```

```
b
a
```

2.4 Dictionary

In [ ]:

```
d = {'a': 1, 'b': True}
print(type(d))
print(d)

print(d['a'])
```

```
<class 'dict'>
{'a': 1, 'b': True}
1
```

In [ ]:

```
# Access key and values in dictionary
for key in d:
  print('key is %s. value is %s' % (key, d[key]))

for key, value in d.items():
  print('key is %s. value is %s' % (key, value))
```

```
key is a. value is 1
key is b. value is True
key is a. value is 1
key is b. value is True
```

Part 3 Numpy - Python Scientific Computing Library

In [ ]:

```
import numpy as np
```

3.1 Numpy ndarray

It encapsulates n-dimensional arrays of homogeneous data types

In [ ]:

```
lst1 = [1, 2, 3, 4, 5, 6]
arr1 = np.array(lst1)
print(type(arr1))
print(arr1)
print(arr1.shape)
print(arr1.size) # total number of elements

lst2 = [[1, 2, 3], [4, 5, 6]]
arr2 = np.array(lst2)
print(arr2)
print(arr2.shape)
print(arr2.size)

arr3 = np.arange(0, 10, 2)
print(arr3)
print(arr3.shape)
print(arr3.size)
```

```
<class 'numpy.ndarray'>
[1 2 3 4 5 6]
(6,)
6
[[1 2 3]
 [4 5 6]]
(2, 3)
6
[0 2 4 6 8]
(5,)
5
```

In [ ]:

```
# reshape
arr4 = arr1.reshape((2,3)) # create 2d array from 1d array with reshape
print(arr4)
print(arr4.shape)

# -1 means the value will be inferred from the length of array.
arr5 = arr1.reshape((-1,3))
print(arr5)
print(arr5.shape)
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
[[1 2 3]
 [4 5 6]]
(2, 3)
```

3.2 Array Indexing

In [ ]:

```
arr6 = np.array([[1, 2, 3], [4, 5, 6]])

print(arr6[1,2])
print(arr6[1]) # the second row
print(arr6[:,0]) # the first column
print(arr6[:,1:3])
```

```
6
[4 5 6]
[1 4]
[[2 3]
 [5 6]]
```

In [ ]:

```
# boolean array indexing
# 注意这是numpy array特有的性质，python list不具备这样的性质
arr6 = np.array([[1, 2, 3], [4, 5, 6]])
print(arr6 > 2)

# We can use boolean array indexing to construct 1d array
print(arr6[arr6 > 2])

# 在numpy中，如果要加入多个条件，必须加括号。同时，逻辑运算符必须是特殊字符 &, |, ~
# 注意区分前面1.2中，python语法中的逻辑运算符特点。
print(arr6[(arr6 % 2 == 0) & (arr6 > 5)])
print(arr6[~(arr6 % 2 == 0)])
print(arr6[(arr6 % 2 == 0) | (arr6 > 5)])
```

```
[[False False  True]
 [ True  True  True]]
[3 4 5 6]
[6]
[1 3 5]
[2 4 6]
```

3.3 Array Math

All the basic mathematical functions operate element-wise on arrays.

In [ ]:

```
a = np.array([[1,2,3],[4,5,6]]) # 一个[]对应的是一行
b = np.array([[7,8,9],[10,11,12]])

print(a.shape)
print(b.shape)
print(a)
print(b)
```

```
(2, 3)
(2, 3)
[[1 2 3]
 [4 5 6]]
[[ 7  8  9]
 [10 11 12]]
```

In [ ]:

```
# element-wise
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

```
[[ 8 10 12]
 [14 16 18]]
[[-6 -6 -6]
 [-6 -6 -6]]
[[ 7 16 27]
 [40 55 72]]
[[0.14285714 0.25       0.33333333]
 [0.4        0.45454545 0.5       ]]
```

In [ ]:

```
# matrix transpose
print(a.T)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

In [ ]:

```
# Inner product
# We use dot function to compute inner product for vectors and matrices.

print(a.T.dot(b)) # dimension 3*2, 2*3 => 3*3
print(a.dot(b.T)) # dimension 2*3, 3*2 => 2*2

# 另一种写法
print(np.dot(a.T, b))
print(np.dot(a, b.T))
```

```
[[47 52 57]
 [64 71 78]
 [81 90 99]]
[[ 50  68]
 [122 167]]
[[47 52 57]
 [64 71 78]
 [81 90 99]]
[[ 50  68]
 [122 167]]
```

3.4 Numpy中的axis

In [ ]:

```
# 对于 2-d NumPy array, axis=0对应的是列，  axis=1对应的是行
# 对于 1-d Numpy array, 只有一个axis, 此时axis=0对应的是行

a = np.array([[1,2,3],[4,5,6]])
b = np.array([8,9,10])

# np.sum
print(np.sum(a))
print(np.sum(a, axis=0))
print(np.sum(a, axis=1))

print(np.sum(b))
print(np.sum(b, axis=0))

print('----------')

# np.max
print(np.max(a))
print(np.max(a, axis=0))
print(np.max(a, axis=1))

print(np.max(b))
print(np.max(b, axis=0))
```

```
21
[5 7 9]
[ 6 15]
27
27
----------
6
[4 5 6]
[3 6]
10
10
```

3.5 Random

In [ ]:
```
# 为了保证每次产生的随机数相同，需要在调用随机数函数之前再次使用相同的seed值
# 若不需要保证随机数据的可复现性，可以不设置这个seed值。
np.random.seed(1)

### Uniform Distribution
# One random number between [0,1)
print(np.random.random())
# Random numbers between [0,1) of shape 2,3
print(np.random.random(size=[2,3]))
# Random numbers between [0,1) of shape 2,3
print(np.random.rand(2,3))
```

```
0.417022004702574
[[7.20324493e-01 1.14374817e-04 3.02332573e-01]
 [1.46755891e-01 9.23385948e-02 1.86260211e-01]]
[[0.34556073 0.39676747 0.53881673]
 [0.41919451 0.6852195  0.20445225]]
```

In [ ]:
```
### Normal Distribution
# One number from normal distribution with default params (mean=0 and variance
print(np.random.normal())
# Normal distribution with mean=0 and variance=1 of shape 2,3
print(np.random.normal(0,1,size=[2,3]))
# Normal distribution with mean=0 and variance=1 of shape 2,3
print(np.random.randn(2,3))
```

```
-0.8599066067340536
[[ 1.77260763 -1.11036305  0.18121427]
 [ 0.56434487 -0.56651023  0.7299756 ]]
[[ 0.37299379  0.53381091 -0.0919733 ]
 [ 1.91382039  0.33079713  1.14194252]]
```

Part 4 Pandas - Python Library for Data Manipulation and Analysis

Python Pandas is built on top of Numpy. We also need to import numpy when using pandas library.

In [ ]:
```
import pandas as pd
import numpy as np
```

4.1. Python Pandas Data Structures

Python Pandas includes two primary data structures: Series (1-d) and DataFrame (2-d)

- **Series**: It is generalized Numpy array. The essential difference is the presence of the index: while the Numpy Array has an implicitly defined integer index used to access the values, the Pandas Series has an explicitly defined index associated with the values.
- **DataFrame**: Most commonly used pandas object. We can think of it like a SQL table or spreadsheet.

In [ ]:

```
# Pandas Series
s1 = pd.Series([1,2,3,4])
print(type(s1))
print(s1)

s2 = pd.Series([1,2,3,4], index = ['x1', 'x2', 'x3', 'x4'])
print(type(s2))
print(s2)
```

```
<class 'pandas.core.series.Series'>
0    1
1    2
2    3
3    4
dtype: int64
<class 'pandas.core.series.Series'>
x1    1
x2    2
x3    3
x4    4
dtype: int64
```

In [ ]:

```
# Pandas DataFrame
df1 = pd.DataFrame({'year': np.arange(2000,2010), 'month': np.arange(1,11), 'r
```

In [ ]:

```
df1.head() # print the first 5 rows
```

Out[35]:

	year	month	rain
0	2000	1	1
1	2001	2	3
2	2002	3	5
3	2003	4	7
4	2004	5	9

In [ ]:

```
df1.tail() # print the last 5 rows
```

Out[36]:

	year	month	rain
5	2005	6	11
6	2006	7	13
7	2007	8	15
8	2008	9	17
9	2009	10	19

In [ ]:

```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   year    10 non-null     int64
 1   month   10 non-null     int64
 2   rain    10 non-null     int64
dtypes: int64(3)
memory usage: 368.0 bytes
```

In [ ]:

```
df1.describe()
```

Out[38]:

	year	month	rain
count	10.00000	10.00000	10.000000
mean	2004.50000	5.50000	10.000000
std	3.02765	3.02765	6.055301
min	2000.00000	1.00000	1.000000
25%	2002.25000	3.25000	5.500000
50%	2004.50000	5.50000	10.000000
75%	2006.75000	7.75000	14.500000
max	2009.00000	10.00000	19.000000

In [ ]:

```
df1.dtypes
```

Out[39]:

```
year     int64
month    int64
rain     int64
dtype: object
```

4.2 DataFrame Indexing and Slicing

In [ ]:

```
df1.head()
```

Out[40]:

	year	month	rain
0	2000	1	1
1	2001	2	3
2	2002	3	5
3	2003	4	7
4	2004	5	9

4.2.1 Use [] for object selection

In [ ]:

```
df2 = df1['year']
display(df2.head())

df3 = df1[['year', 'month']]
display(df3.head())

df4 = df1[2:5][['year', 'month']]
display(df4)
```

```
0    2000
1    2001
2    2002
3    2003
4    2004
Name: year, dtype: int64
```

	year	month
0	2000	1
1	2001	2
2	2002	3
3	2003	4
4	2004	5

	year	month
2	2002	3
3	2003	4
4	2004	5

4.2.2 Use iloc to select by position index

```
In [ ]: df5 = df1.iloc[[0,1,3],[0,2]]  # df.iloc[行信息, 列信息]
        display(df5)

        df6 = df1.iloc[:, 0:2]
        display(df6)

        df7 = df1.iloc[[3]] # df.iloc[行信息]
        display(df7)
```

	year	rain
0	2000	1
1	2001	3
3	2003	7

	year	month
0	2000	1
1	2001	2
2	2002	3
3	2003	4
4	2004	5
5	2005	6
6	2006	7
7	2007	8
8	2008	9
9	2009	10

	year	month	rain
3	2003	4	7

4.2.3 Use loc to select by row index and column label name

In [ ]:
```
df8 = df1.loc[[0,1,3],['year']]  # df.loc[行信息, 列信息]
display(df8)

df9 = df1.loc[1:3, ['year', 'rain']]
display(df9)

df10 = df1.loc[[3]] # df.loc[行信息]
display(df10)
```

	year
0	2000
1	2001
3	2003

	year	rain
1	2001	3
2	2002	5
3	2003	7

	year	month	rain
3	2003	4	7

4.2.4 Boolean indexing (与Section 3.2中 numpy array的用法类似)

In [ ]:

```
# 以上三种indexing用法，[], loc, iloc都可以使用boolean indexing.
# 但是在iloc方法中Boolean indexing的时候，需要做series to ndarray的转换，不常用。
df11 = df1[(df1['year'] <2005) & (df1['rain'] < 6)]
display(df11)

df12 = df1.loc[(df1['year'] <2005) & (df1['rain'] < 6), ['year', 'rain']]
display(df12)

df12 = df1.iloc[((df1['year'] <2005) & (df1['rain'] < 6)).values, [0, 2]]
display(df12)
```

	year	month	rain
0	2000	1	1
1	2001	2	3
2	2002	3	5

	year	rain
0	2000	1
1	2001	3
2	2002	5

	year	rain
0	2000	1
1	2001	3
2	2002	5

4.3 Apply functions to DataFrame

In [ ]:

```
df1.head()
```

Out[45]:

	year	month	rain
0	2000	1	1
1	2001	2	3
2	2002	3	5
3	2003	4	7
4	2004	5	9

In [ ]:
```
def func(input):
  return input + 1

df1['year'].apply(func)
```

Out[46]:
```
0    2001
1    2002
2    2003
3    2004
4    2005
5    2006
6    2007
7    2008
8    2009
9    2010
Name: year, dtype: int64
```

In [ ]:
```
# Use anonymous function
df1['year'].apply(lambda x: x+1)
```

Out[47]:
```
0    2001
1    2002
2    2003
3    2004
4    2005
5    2006
6    2007
7    2008
8    2009
9    2010
Name: year, dtype: int64
```

In [ ]:
```
df1.apply(lambda x: x+1)
```

Out[48]:

	year	month	rain
0	2001	2	2
1	2002	3	4
2	2003	4	6
3	2004	5	8
4	2005	6	10
5	2006	7	12
6	2007	8	14
7	2008	9	16
8	2009	10	18
9	2010	11	20

可以在apply()中使用aggregate functions (比如 count, mean) 对列或者行整体进行处理 (参考3.4 Numpy axis的介绍)

In [ ]:

```
df1.apply(lambda x: x.mean())
```

Out[49]:

```
year     2004.5
month       5.5
rain       10.0
dtype: float64
```

In [ ]:

```
df1.apply(lambda x: x.mean(), axis=0)
```

Out[50]:

```
year     2004.5
month       5.5
rain       10.0
dtype: float64
```

In [ ]:

```
df1.apply(lambda x: x.mean(), axis=1)
```

Out[51]:

```
0    667.333333
1    668.666667
2    670.000000
3    671.333333
4    672.666667
5    674.000000
6    675.333333
7    676.666667
8    678.000000
9    679.333333
dtype: float64
```

4.4 Merge Tables

In [ ]:

```
df0 = pd.DataFrame({
    'year': np.arange(2000,2010),
    'month': np.arange(1,11),
    'electricity': np.arange(100,120,2)})
```

In [ ]:

```
display(df0)
display(df1)
```

	year	month	electricity
0	2000	1	100
1	2001	2	102
2	2002	3	104
3	2003	4	106
4	2004	5	108
5	2005	6	110
6	2006	7	112
7	2007	8	114
8	2008	9	116
9	2009	10	118

	year	month	rain
0	2000	1	1
1	2001	2	3
2	2002	3	5
3	2003	4	7
4	2004	5	9
5	2005	6	11
6	2006	7	13
7	2007	8	15
8	2008	9	17
9	2009	10	19

In [ ]:

```
df_join = pd.merge(df1, df0, on=['year', 'month'])
display(df_join)
```

	year	month	rain	electricity
0	2000	1	1	100
1	2001	2	3	102
2	2002	3	5	104
3	2003	4	7	106
4	2004	5	9	108
5	2005	6	11	110
6	2006	7	13	112
7	2007	8	15	114
8	2008	9	17	116
9	2009	10	19	118

4.5 Grouping

类似于SQL中的groupby

In [ ]:

```
df_gp = pd.DataFrame({
    'year': [2000, 2000, 2000, 2001, 2001, 2002, 2002],
    'month': [1,2,3,4,3,2,5],
    'electricity': [101, 201, 302, 131, 131, 131, 123]})
```

In [ ]:

```
display(df_gp)
```

	year	month	electricity
0	2000	1	101
1	2000	2	201
2	2000	3	302
3	2001	4	131
4	2001	3	131
5	2002	2	131
6	2002	5	123

In [ ]:

```
df_gp.groupby(['year'], as_index = False).max()
```

Out[57]:

	year	month	electricity
0	2000	3	302
1	2001	4	131
2	2002	5	131

Part 5 Load and Save Data on Google Colaboratory

5.1 Load data to Google Colaboratory

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[important!!!] You need to change it to your own file path here.

In [ ]:

```
cd /content/drive/MyDrive/
```

```
/content/drive/MyDrive
```

In [ ]:

```
# Check your file list
# ls
```

In [ ]:

```
input_data = pd.read_csv('notebook1_data.csv')
```

In [ ]:

```
input_data.head()
```

5.2 Save data to Google Drive

In [ ]:

```
input_data['aisle_id'] += 100
```

In [ ]:

```
input_data.head()
```

In [ ]:

```
input_data.to_csv('notebook1_data_updated.csv', index=False)
```

In [ ]:

```
# Check your file list
# ls
```