Python warm-up

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index. The basic method to create a Series is to call:

```
pandas.Series( data, index, dtype, name, copy)
```

```
import pandas as pd

a = [1, 2, 3]

myvar = pd.Series(a)

print(myvar[1])
```

```
2
```

DataFrame is a tabular data structure, which contains a set of ordered columns, each column can be a different value type (number, string, Boolean value). DataFrame has both row index and column index. It can be regarded as a dictionary composed of Series (using one index together).

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

```
import pandas as pd

data = [['Google',10],['Runoob',12],['Wiki',13]]

df = pd.DataFrame(data,columns=['Site','Age'],dtype=float)

print(df)
```

```
     Site   Age
0  Google  10.0
1  Runoob  12.0
2    Wiki  13.0
```

Sequence is the most basic data structure in Python. Each element in the sequence is assigned a number-its position, or index, the first index is 0, the second index is 1, and so on.

Python has 6 built-in types for sequences, but the most common are lists and tuples.

Operations that can be performed on sequences include indexing, slicing, adding, multiplying, and checking members.

In addition, Python has built-in methods for determining the length of the sequence and determining the largest and smallest elements.

The **list** is the most commonly used Python data type, and it can appear as a comma-separated value in square brackets.

The data items of the list do not need to have the same type

To create a list, simply enclose the different data items separated by commas in square brackets. As follows:

```
#!/usr/bin/python

list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]

print "list1[0]: ", list1[0]
print "list2[1:5]: ", list2[1:5]
```

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

A **function** is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Python also accepts function recursion, which means a defined function can call itself.
Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(6)
```

```
Recursion Example Results
1
3
6
10
15
21
```

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

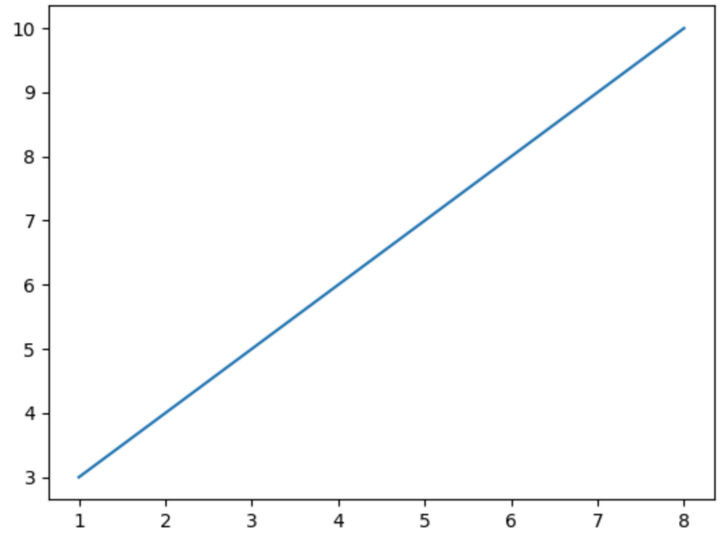
In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

Example

Draw a line in a diagram from position (1, 3) to position (8, 10):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

Result:

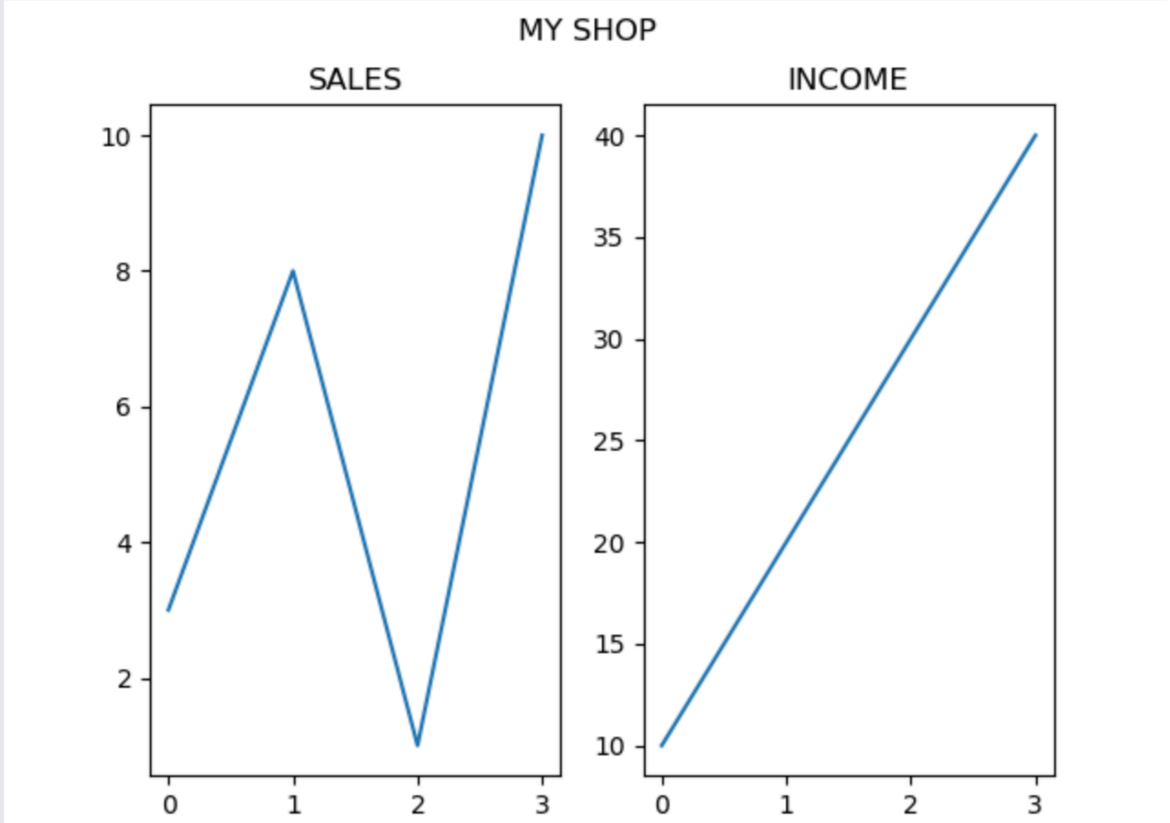


Matplotlib Subplots

With the subplots() function you can draw multiple plots in one figure:

```
import matplotlib.pyplot as plt
import numpy as np

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

Result:



pandas.DataFrame.boxplot

DataFrame.boxplot(column=None, by=None, ax=None, fontsize=None, rot=0, grid=True, figsize=None, layout=None, return_type=None, backend=None, **kwargs)

Make a box plot from DataFrame columns.

Make a box-and-whisker plot from DataFrame columns, optionally grouped by some other columns. A box plot is a method for graphically depicting groups of numerical data through their quartiles. The box extends from the Q1 to Q3 quartile values of the data, with a line at the median (Q2). The whiskers extend from the edges of box to show the range of the data. By default, they extend no more than 1.5 * IQR (IQR = Q3 - Q1) from the edges of the box, ending at the farthest data point within that interval. Outliers are plotted as separate dots.

Parameters:

column : *str or list of str, optional*

Column name or list of names, or vector. Can be any valid input to `pandas.DataFrame.groupby()`.

by : *str or array-like, optional*

Column in the DataFrame to `pandas.DataFrame.groupby()`. One box-plot will be done per value of columns in *by*.

ax : *object of class matplotlib.axes.Axes, optional*

The matplotlib axes to be used by boxplot.

fontsize : *float or str*

Tick label font size in points or as a string (e.g., *large*).

rot : *int or float, default 0*

The rotation angle of labels (in degrees) with respect to the screen coordinate system.

grid : *bool, default True*

Setting this to True will show the grid.

figsize : *A tuple (width, height) in inches*

The size of the figure to create in matplotlib.

layout : *tuple (rows, columns), optional*

For example, (3, 5) will display the subplots using 3 columns and 5 rows, starting from the top-left.

return_type : *{'axes', 'dict', 'both'} or None, default 'axes'*

The kind of object to return. The default is `axes`.

- 'axes' returns the matplotlib axes the boxplot is drawn on.
- 'dict' returns a dictionary whose values are the matplotlib Lines of the boxplot.
- 'both' returns a namedtuple with the axes and dict.
- when grouping with `by`, a Series mapping columns to `return_type` is returned.

  If `return_type` is *None*, a NumPy array of axes with the same shape as `layout` is returned.

backend : *str, default None*

Backend to use instead of the backend specified in the option `plotting.backend`. For instance, 'matplotlib'. Alternatively, to specify the `plotting.backend` for the whole session, set `pd.options.plotting.backend`.

New in version 1.0.0.

****kwargs**

All other plotting keyword arguments to be passed to `matplotlib.pyplot.boxplot()`.

Returns:

result

See Notes.

Parameters:

column : *str or list of str, optional*

Column name or list of names, or vector. Can be any valid input to `pandas.DataFrame.groupby()`.

by : *str or array-like, optional*

Column in the DataFrame to `pandas.DataFrame.groupby()`. One box-plot will be done per value of columns in *by*.

ax : *object of class matplotlib.axes.Axes, optional*

The matplotlib axes to be used by boxplot.

fontsize : *float or str*

Tick label font size in points or as a string (e.g., *large*).

rot : *int or float, default 0*

The rotation angle of labels (in degrees) with respect to the screen coordinate system.

grid : *bool, default True*

Setting this to True will show the grid.

figsize : *A tuple (width, height) in inches*

The size of the figure to create in matplotlib.

layout : *tuple (rows, columns), optional*

For example, (3, 5) will display the subplots using 3 columns and 5 rows, starting from the top-left.

return_type : *{'axes', 'dict', 'both'} or None, default 'axes'*

The kind of object to return. The default is `axes`.

- 'axes' returns the matplotlib axes the boxplot is drawn on.
- 'dict' returns a dictionary whose values are the matplotlib Lines of the boxplot.
- 'both' returns a namedtuple with the axes and dict.
- when grouping with `by`, a Series mapping columns to `return_type` is returned.

  If `return_type` is *None*, a NumPy array of axes with the same shape as `layout` is returned.

backend : *str, default None*

Backend to use instead of the backend specified in the option `plotting.backend`. For instance, 'matplotlib'. Alternatively, to specify the `plotting.backend` for the whole session, set `pd.options.plotting.backend`.

New in version 1.0.0.

****kwargs**

All other plotting keyword arguments to be passed to `matplotlib.pyplot.boxplot()`.

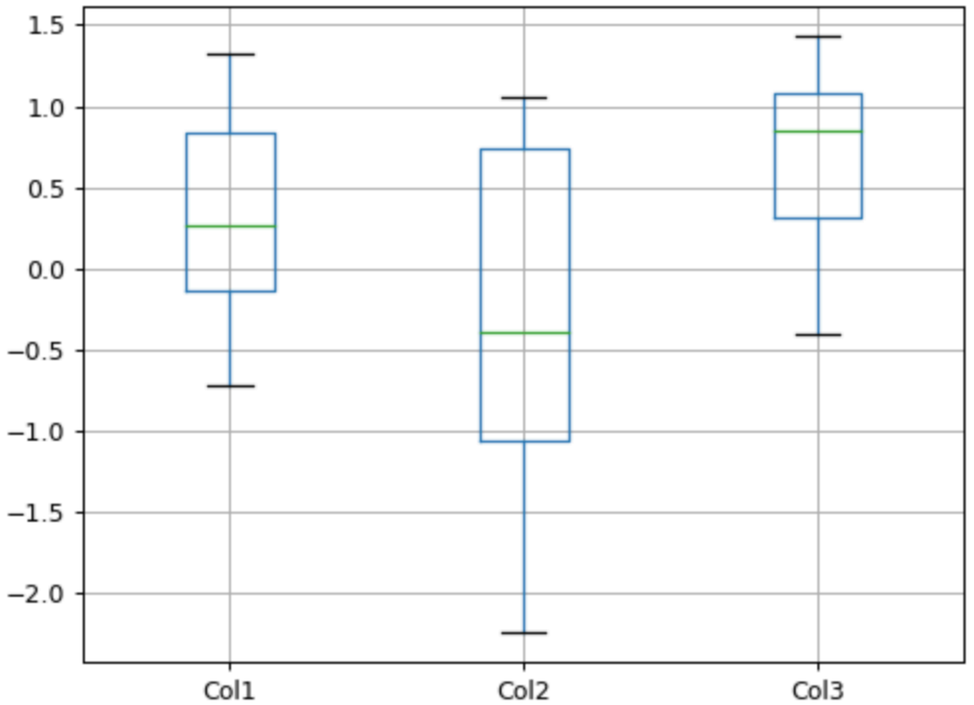
Returns:

result

See Notes.

Boxplots can be created for every column in the dataframe by `df.boxplot()` or indicating the columns to be used:

```
>>> np.random.seed(1234)
>>> df = pd.DataFrame(np.random.randn(10, 4),
...                   columns=['Col1', 'Col2', 'Col3', 'Col4'])
>>> boxplot = df.boxplot(column=['Col1', 'Col2', 'Col3'])
```



Python warm-up exercises

Rearrange the odd numbers in an array, 0 is not an odd number

Example: sort_array([5,3,2,8,1,4])==[1,3,2,8,5,4]

Python warm-up exercises

Rearrange the odd numbers in an array, 0 is not an odd number

Example: sort_array([5,3,2,8,1,4])==[1,3,2,8,5,4]

```
def sort_array(arr):
    odd_index=[ind for (ind,val) in enumerate(arr) if val%2==1]
    sorted_odd=sorted([odd for odd in arr if odd%2==1])
    j=0
    for i in odd_index:
        arr[i]=sorted_odd[j]
        j+=1

    return arr if arr!=[] else[]

sort_array(arr)
```

Answer 1

Python warm-up exercises

Rearrange the odd numbers in an array, 0 is not an odd number

Example: sort_array([5,3,2,8,1,4])==[1,3,2,8,5,4]

```
arr=[5,3,2,8,1,4]
def sort_array(arr):
    odds=[x if x%2==0 else 0 for x in arr ]#偶数位数不动，奇数为置为0
    evens=sorted([x for x in arr if x%2==1])#取出奇数为数，并排序
    even_index=0
    for index,e in enumerate(odds):
        if e==0:
            odds[index]=evens[even_index]#把奇数数从新填回列表
            even_index+=1

    return odds

sort_array(arr)
```

Answer 2

Python warm-up exercises

Rearrange the odd numbers in an array, 0 is not an odd number

Example: sort_array([5,3,2,8,1,4])==[1,3,2,8,5,4]

```
def sort_array(arr):
    odds=sorted((x for x in arr if x%2!=0),reverse=True)
    print ([x if x%2==0 else odds.pop() for x in arr])

arr=[5,3,2,8,1,4]
sort_array(arr)
```

Answer 3

Python warm-up exercises

ATM machines allow 4 or 6 digits, but these 4 or 6 digits can only be pure numbers

for example:

validate_pin('1234')==True

validate_pin('12345')==False

validate_pin('a234')==False

Python warm-up exercises

ATM machines allow 4 or 6 digits, but these 4 or 6 digits can only be pure numbers

for example:

validate_pin('1234')==True

validate_pin('12345')==False

validate_pin('a234')==False

```
def validate_pin(pin):
    try:
        if len(pin)==4 or len(pin)==6:
            for n in pin:
                if int(n)>=0 and int(n)<=9:
                    return True
        else:
            return False

    except Exception as e:
        print (e)
        return False

validate_pin('0234')
```

Answer 1

Python warm-up exercises

ATM machines allow 4 or 6 digits, but these 4 or 6 digits can only be pure numbers

for example:

validate_pin('1234')==True

validate_pin('12345')==False

validate_pin('a234')==False

```
defvalidate_pin(pin):
    returnlen(pin) in(4,6) andpin.isdigit()
    validate_pin('-1.234')
```

Answer 2

Python warm-up exercises

Give you a few strings, write a function output according to the following rules, and specify the format of the string:

accm("abcd") #A-Bb-Ccc-Dddd

accm("RqaEzty") #R-Qq-Aaa-Eeee-Zzzzz-Ttttt-Yyyyyyy

accm("cwAt") # C-Ww-Aaa-Tttt

```
def accm(chars):
    new_chars=[]
    for index,c in enumerate(chars):
        words=[]
        [words.append(c) for x in range(0,index+1)]
        new_chars.append(''.join(words).capitalize())

    return '-'.join(new_chars)

print (accm("abcd"))
print (accm("RqaEzty"))
print (accm("cwAt") )
```

Answer 1

Python warm-up exercises

Give you a few strings, write a function output according to the following rules, and specify the format of the string:

accm("abcd") #A-Bb-Ccc-Dddd

accm("RqaEzty") #R-Qq-Aaa-Eeee-Zzzzz-Ttttt-Yyyyyyy

accm("cwAt") # C-Ww-Aaa-Tttt

```
defaccm(chars):
    return'-'.join(c.upper+c.lower()*i
    for i inenumerate(chars))
```

Answer 2

Python warm-up exercises

Give you the first three numbers of the array,
the next number is the sum of the first 3 numbers,
and the first n numbers are required to be returned

```python
def tri(nums_list, n):
    a, b, c = nums_list[0], nums_list[1], nums_list[2]
    count = len(nums_list)
    while count<n:
        a, b, c = b, c, a + b + c
        count+=1
        nums_list.append(c)

    print(nums_list)

tri([0, 0, 1], 10)
```

Answer 1

Python warm-up exercises

Give you the first three numbers of the array,
the next number is the sum of the first 3 numbers,
and the first n numbers are required to be returned

```python
def tri(nums_list, n):
    def gen():
        a, b, c = nums_list[0], nums_list[1], nums_list[2]
        count = len(nums_list)
        while count<n:
            a, b, c = b, c, a + b + c
            count+=1
            yield c

    print (nums_list+list(gen()))

tri([0,0,1],10)
```

Answer 2

Python warm-up exercises

Give you the first three numbers of the array,
the next number is the sum of the first 3 numbers,
and the first n numbers are required to be returned

```
def tri2(nums_list,n):
res=nums_list[:n]
fori in range(n-3):
res.append(sum(res[-3:]))
returnres
tri2([0,0,1],10)
```

Answer 3

Python warm-up exercises

The weight of each question mark (?) is 3, and the weight of each exclamation mark (!) is 2. Put two characters on the left and two characters on the right to see if they are balanced?

Example:

balance("!!","??")=="Right"

balance("!??","?!!")=="Left"

balance("!?!!","?!?")=="Left"

balance("!!???!????","??!!?!!!!!!!")=="Balance"

```
def balance(left,right):
    weight={'?':3,'!':2}
    left_num=sum([weight.get(e) for e in left ])
    right_num=sum([weight.get(e) for e in right ])

    if left_num==right_num:
        return 'Balance'
    elif left_num>right_num:
        return 'Left'
    else:
        return 'Right'


assert balance("!!","??")=="Right"
assert balance("!??","?!!")=="Left"
assert balance("!!???!????","??!!?!!!!!!!")=="Balance"
```

Answer 1

Survival analysis

the data table consists of the following categories: Sex of the patient, individual id, ID, project-id, description, ICDOT, ICDOT-text, ICDOM, ICDOM-text, NCIt, NCIt-text, death, follow-up months, TNM, CNV coverage, CNV fraction, deletion coverage, deletion fraction, duplication coverage, duplication fraction, and age.

Progenetix Data

The content of the progenetix resource is freely accessible for research and commercial purposes, with attribution. The database and software are developed by the group of Michael Baudis at the University of Zurich and the Swiss Institute of Bioinformatics SIB. Many previous members and external collaborators have contributed to data content and resource features.

Task: Extract survival data from progenetix

Survival analysis

Copy number variations (CNVs) are structurally variant regions in which copy number differences have been observed between two or more genomes.

The term CNV coverage is used to talk about how much of the length of the genome is covered with CNVs, and CNV fraction describes what fraction of the genome is covered with CNVs.

Question1: Does the CNV coverage of any two tumour types influence the survival probability of the respective patients?

Survival analysis

The **Kaplan-Meier estimate** is one of the best options to be used to measure the fraction of subjects living for a certain amount of time after treatment or diagnosis. When talking about treatments, the effect of said intervention is assessed by measuring the number of subjects, which survived or which were saved after that intervention over a period of time. The time starting from a defined point, to the occurrence of a given event, for example death, is called survival time (in this case the age), and the analysis of group data is called survival analysis. This is why one can say it is a ‘time to event’ analysis. The Kaplan-Meier estimate is the simplest way of computing the survival over time in spite of all the difficulties associated with subjects or situations.

Exercise: Do some **Kaplan-Meier estimate** for the survival data.