Python 数据科学入门手册

目 录

一、Numpy 入门详细教程

序言：python 数据科学基础库主要是三剑客：numpy，pandas 以及 matplotlib，每个库都集成了大量的方法接口，配合使用功能强大。平时虽然一直在用，也看过很多教程，但纸上得来终觉浅，还是需要自己系统梳理总结才能印象深刻。本篇先从 numpy 开始，对 numpy 常用的方法进行思维导图式梳理，多数方法仅拉单列表，部分接口辅以解释说明及代码案例。最后分享了个人关于 axis 和广播机制的理解。

01 基本介绍

numpy：numerical python 缩写，提供了底层基于 C 语言实现的数值计算库，与 python 内置的 list 和 array 数据结构相比，其支持更加规范的数据类型和极其丰富的操作接口，速度也更快。

numpy 的两个重要对象是 ndarray 和 ufunc，其中前者是数据结构的基础，后者是接口方法的基础。

ufunc，通函数，其意义是可以像执行标量运算一样执行数组运算，本质即是通过隐式的循环对各个位置依次进行标量运算。只不过这里的隐式循环交由底层 C 语言实现，因此相比直接用 python 循环实现，ufunc 语法更为简洁、效率更为高效。

索引、迭代和切片操作方式与普通列表比较类似，但是支持更为强大的 bool 索引。

这部分内容比较基础，仅补充一个个人认为比较有用的 ufunc 加聚合的例子。ufunc 本身属于方法（方法即是类内的函数接口），ufunc 之上还支持 4 个方法：

- reduce，聚合方法
- accumulate，累计聚合
- reduceat，按指定轴向、指定切片聚合
- outer：外积

当然，后两个用处较少也不易理解，前两个在有些场景下则比较有用：

```
a = np.arange(12).reshape(-1, 3)
print('original a: \n', a)
print('add.accumulate: \n', np.add.accumulate(a, axis=1))
print('multiply.reduce: \n', np.multiply.reduce(a, axis=0))
```

```
original a:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
add.accumulate:
 [[ 0  1  3]
 [ 3  7 12]
 [ 6 13 21]
 [ 9 19 30]]
multiply.reduce:
 [  0 280 880]
```

02 数组创建

numpy 中支持 5 类创建数组的方式：

- 从普通数据结构创建，如列表、元组等
- 从特定的 array 结构创建，支持大量方法，例如 ones、zeros、empty 等等
  - empty 接收指定大小创建空数组，这里空数组的意义在于未进行数值初始赋值，随机产生，因而速度要更快一些

▪ linspace 和 arange 功能类似，前者创建指定个数的数值，后者按固定步长创建，其中 linspace 默认包含终点值（可以通过 endpoint 参数设置为 false），而 arange 则不含终点

- 从磁盘读取特定的文件格式
- 从缓存或字符读入数组
- 从特定的库函数创建，例如 random 随机数包

以上方法中，最为常用的是方法 1、2、5。

03 数组增删

numpy 提供了与列表类似的增删操作，其中

- append 是在指定维度后面拼接数据，要求相应维度大小匹配
- insert 可以在指定维度任意位置插入数据，要求维度大小匹配
- delete 删除指定维度下的特定索引对应数据

```
a = np.arange(12).reshape(-1, 3)
b = np.array([11, 12, 13]).reshape(-1, 3)
np.append(a, b, axis=0)
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11],
       [11, 12, 13]])
```

```
a = np.arange(12).reshape(-1, 3)
b = np.array([11, 12, 13, 14])
np.insert(a, 0, b, axis=1)
```

```
array([[11,  0,  1,  2],
       [12,  3,  4,  5],
       [13,  6,  7,  8],
       [14,  9, 10, 11]])
```

```
a = np.arange(12).reshape(-1, 3)
np.delete(a, 0)
```

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

三种方法需要接收一个 axis 参数，如果未指定，则均会先对目标数组展平至一维数组后再执行相应操作。

04 数组变形

数组变形是指对给定数组重新整合各维度大小的过程，numpy 封装了 4 类基本的变形操作：转置、展平、尺寸重整和复制。主要方法接口如下：

- reshape 常用于对给定数组指定维度大小，原数组不变，返回一个具有新形状的新数组；如果想对原数组执行 inplace 变形操作，则可以直接指定其形状为合适维度

```
a = np.arange(12)
a.shape = (3, 4)
a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
a = np.arange(12).reshape(4, 3)
a.reshape(3, 4)
a
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

resize 与 reshape 功能类似，主要有 3 点区别：

- resize 面向对象操作时，执行 inplace 操作，调用 np.resize 类方法时则不改变原数组形状；而 reshape 无论如何都不改变原数组形状
- resize 变形后的数组大小可以不和原数组一致，会自动根据新尺寸情况进行截断或拼接
- 正因为 resize 可以执行截断，所以要求接收确切的尺寸参数，不允许出现-1 这样的"非法"数值；而 reshape 中常用-1 的技巧实现某一维度的自动计算

另外，当 resize 新尺寸参数与原数组大小不一致时，要求操作对象具有原数组的，而不能是 view 或简单赋值。（具体参考 08 视图与拷贝一节）

ravel 和 flat 功能类似，均返回对数组执行展平后的结果，且不改变原数组形状，区别在于：

- 前者是方法接口，而后者是属性接口，
- 前者返回对象类型仍然是数组，而后者返回对象类型是专用的 flatten 类型，一般用作迭代器对象
- transpose 与 T 均执行转置操作，前者是方法，后者是属性

tile 和 repeat 方法类似，均为对给定数组执行复制操作，区别在于：

- tile 面向整个数组复制，而 repeat 面向数组元素复制

▪ tile 不接收维度参数，而 repeat 需指定维度参数，否则会对数组先展平再复制

```
a = np.array([[1, 2], [3, 4]])
np.tile(a, reps=(2, 2))
```

```
array([[1, 2, 1, 2],
       [3, 4, 3, 4],
       [1, 2, 1, 2],
       [3, 4, 3, 4]])
```

```
a = np.array([[1, 2], [3, 4]])
np.repeat(a, repeats=2, axis=0)
```

```
array([[1, 2],
       [1, 2],
       [3, 4],
       [3, 4]])
```

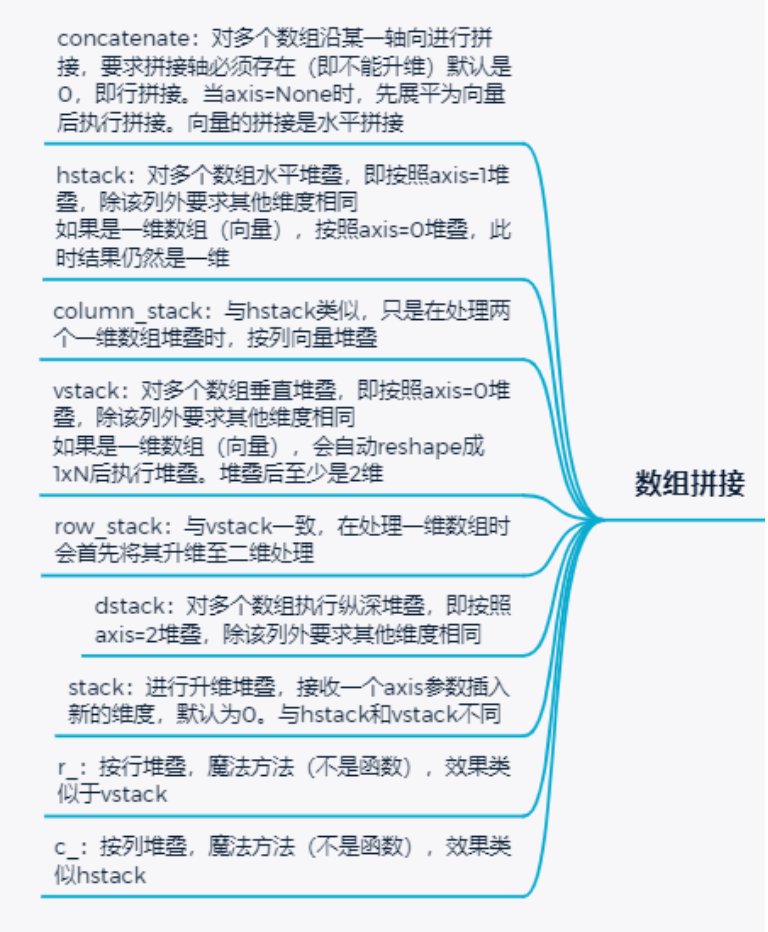
```
a = np.array([[1, 2], [3, 4]])
np.repeat(a, repeats=2, axis=1)
```

```
array([[1, 1, 2, 2],
       [3, 3, 4, 4]])
```

```
a = np.array([[1, 2], [3, 4]])
np.repeat(a, repeats=2)
```

```
array([1, 1, 2, 2, 3, 3, 4, 4])
```

05 数组拼接



数组拼接也是常用操作之一，主要有 3 类接口：

- concatenate，对给定的多个数组按某一轴进行拼接，要求所有数组具有相同的维度（ndim 相等）、且在非拼接轴大小一致
- stack 系列，共 6 个方法：
- hstack，column_stack：功能基本一致，均为水平堆叠（axis=1），或者说按列堆叠。唯一的区别在于在处理一维数组时：hstack 按 axis=0 堆叠，且不要

求两个一维数组长度一致，堆叠后仍然是一个一维数组；而 column_stack 则会自动将两个一维数组变形为 Nx1 的二维数组，并仍然按 axis=1 堆叠，自然也就要求二者长度一致，堆叠后是一个 Nx2 的二维数组

- vstack，row_stack，功能一致，均为垂直堆叠，或者说按行堆叠，axis=0
- dstack，主要面向三维数组，执行 axis=2 方向堆叠，输入数组不足 3 维时会首先转换为 3 维，主要适用于图像处理等领域
- stack，进行升维堆叠，执行效果与前几种堆叠方式基本不同，要求所有数组必须具有相同尺寸。堆叠后，一维变二维、二维变三维……
- 魔法方法：r_[]，c_[]，效果分别与 row_stack 和 column_stack 类似，但具体语法要求略有不同。另外，虽然不是函数，但第一个参数可以是一个字符串实现特定功能设置。

06 数组切分

数组切分可以看做是数组拼接的逆操作，分别对应：

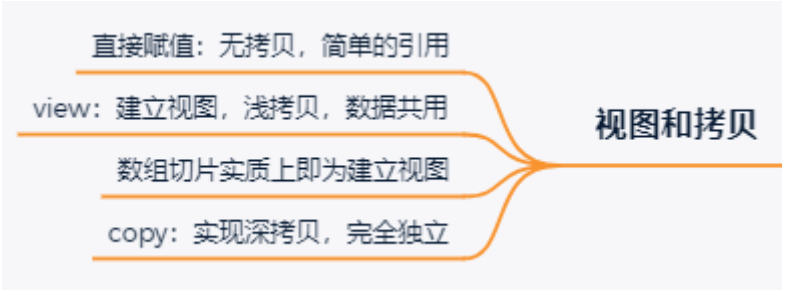
- hsplit：水平切分，要求切分后大小相等，维数不变，可以切分一维数组
- vsplit：垂直切分，要求切分后大小相等，维数不变，要求至少二维以上
- dsplit：纵深切分，要求切分后大小相等，维数不变，至少三维数组
- split：通过接收一个 axis 参数实现任意切分，默认 axis=0，若设置 axis=1 或 2 则可分别实现 vstack 和 dstack
- array_split：前面 4 个方法均要求实现相同大小的子数组切分，当切分份数无法实现整除时会报错。array_split 则可以适用于近似相等条件下的切分，也接受一个 axis 参数实现指定轴向

07 基本统计量

numpy 可以很方便的实现基本统计量，而且每种方法均包括对象方法和类方法：

- max,argmax 分别返回最大值和最大值对应索引,可接收一个 axis 参数，指定轴线的聚合统计。对于二维及以上数组，若不指定 axis，即 axis=None，此时对数组所有数值求聚合统计
- min，argmin，与最大一致
- mean、std，分别求均值和标准差，也可接收一个缺省参数 axis 实现特定轴向聚合统计或全局聚合
- var、cov，分别求方差和协方差，与均值标准差类似
- sort、argsort，分别返回排序后的数组和相应索引，接收一个 axis 参数，默认为 axis=-1，按最后一个轴向，若 axis=None 表示先展平成一维数组后再排序；另外可设置排序算法，如快排、堆排或归并等

08 视图与拷贝



与列表的操作类似，numpy 的数组类型也存在深浅拷贝之分：

- 直接赋值：无拷贝，相当于是引用
- view()：建立视图，浅拷贝，形状可以不一致，但数据相同
- copy()：深拷贝，完全独立的对象

```
a = np.arange(3)
b = a
id(b) == id(a)
```

```
True
```

```
a = np.arange(3)
b = a.view()
b.shape = (1, 3)#浅拷贝可以更改形状
print(a, b)
```

```
[0 1 2] [[0 1 2]]
```

```
a = np.arange(3)
b = a.view()
print(id(a) == id(b))
b[1] = 111 #浅拷贝数据共享
print(a, b)
```

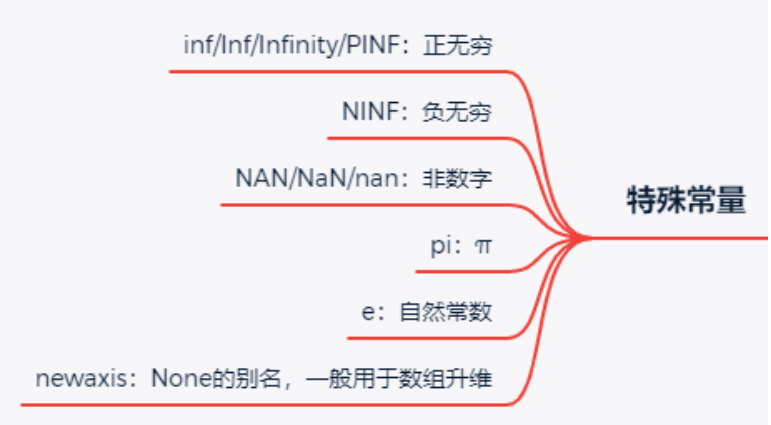
```
False
[  0 111   2] [  0 111   2]
```

```
a = np.arange(3)
b = a.copy()
b[1] = 111 #深拷贝完全独立
print(a, b)
```

```
[0 1 2] [  0 111   2]
```

注：正因为赋值和 view 操作后两个数组的数据共享，所以在前面 resize 试图更改数组形状时可以执行、但更改元素个数时会报错。

09 特殊常量



numpy 提供了一些特殊的常量，值得注意的是 np.newaxis 可以用作是对数组执行升维操作，效果与设置为 None 一致。

10 随机数包

Random 是 numpy 下的一个子包，内置了大量的随机数方法接口，包括绝大部分概率分布接口，常用的主要还是均匀分布和正态分布：

- 均匀分布：random、rand、uniform，三者功能具有相似性，其中前两者均产生指定个数的 0-1 之间均匀分布，而 uniform 可通过设置参数实现任意区间的均匀分布；当需要产生整数均匀分布时，可用 randint

```
np.random.uniform(low=1, high=2, size=(2, 3)) #接受起始值的均匀分布
```

```
array([[1.37799404, 1.09221701, 1.6534109 ],
       [1.55784076, 1.36156476, 1.2250545 ]])
```

```
np.random.random(size=(2, 3)) #接受一个size元组参数作为维度
```

```
array([[0.78031476, 0.30636353, 0.22195788],
       [0.38797126, 0.93638365, 0.97599542]])
```

```
np.random.rand(2, 3) #接受多个整数作为维度，不是元组
```

```
array([[0.20724288, 0.0514672 , 0.44080984],
       [0.02987621, 0.45683322, 0.64914405]])
```

- 正态分布：randn，normal，前者生成标准正态分布（均值为 0，方差为 1），后者产生任意正态分布，接收一个 loc 参数作为均值，scale 参数作为标准差
- permutation、shuffle，对给定序列实现随机排列，前者返回一个新数组，后者是 inplace 操作
- seed，因为计算机中的随机数严格讲都是伪随机，需要依赖一个随机数种子来不断生成新的随机数，seed 可以用于固定这个随机种子。当指定随机数种子后，后续的随机将得到固化

11 线性代数包

除了随机数包，numpy 下的另一个常用包是线性代数包，常见的矩阵操作均位于此包下。由于点积 dot()和向量点积 vdot()操作使用较为频繁，所以全局可用。

12 关于 axis 的理解

由于 numpy 的基本数据结构是多维数组，很多接口方法均存在维度的问题，按照不同维度执行操作结果往往不同，例如拼接、拆分、聚合统计等，此时一般需要设置一个维度参数，即 axis。由于很多教程因为翻译或语言习惯不同，存在众说纷纭、口径不一的问题，有的说 axis=0 是横轴，有的说是纵向，所以如何理解 axis 的含义可能是很多 numpy 初学者的常见困扰之一，笔者也是如此。

这一问题困扰了好久，直至一次无意间看到了相关源码中的注释：

```
np.sort?
Signature: np.sort(a, axis=-1, kind=None, order=None)
Docstring:
Return a sorted copy of an array.

Parameters
----------
a : array_like
    Array to be sorted.
axis : int or None, optional
    Axis along which to sort. If None, the array is flattened before
    sorting. The default is -1, which sorts along the last axis.
```

例如，在 sort 方法中，axis 参数的解释为"Axis along which to sort"，翻译过来就是沿着某一轴执行排序。这里的**沿着一词**用得恰到好处，形象的描述了参数 axis 的作用，即相关操作是如何与轴向建立联系的，在具体解释之前，先介绍下 axis 从小到大的顺序问题。axis 从小到大对应轴的出场顺序先后，或者说变化快慢：axis=0 对应主轴，沿着行变化的方向，可以理解为在多重 for 循环中最外面的一层，对应行坐标，数值变化最慢；而 axis=1 对应次轴，沿着列变化的方向，在多重 for 循环中变化要快于 axis=0 的轴向。类似地，如果有更高维度则依次递增。

至此，再来理解这里 axis 沿着的意义。举个例子，axis=0 代表沿着行变化的方向，那么自然地，切分方法 split(axis=0)接口对应 vsplit，因为是对行切分，即垂直切分；而 split(axis=1)接口则对应 hsplit，因为是对列切分，即水平切分；split(axis=2)则对应 dsplit。类似的，np.sort(axis=0)必然是沿着行方向排序，也就是分别对每一列执

行排序。想必这样理解，应该不会存在混淆了。

13 关于广播机制

可能困扰 numpy 初学者的另一个用法是 numpy 的一大利器：广播机制。广播机制是指执行 ufunc 方法（即对应位置元素 1 对 1 执行标量运算）时，可以确保在数组间形状不完全相同时也可以自动的通过广播机制扩散到相同形状，进而执行相应的 ufunc 方法。当然，这里的广播机制是有条件的：

```
General Broadcasting Rules
==========================
When operating on two arrays, NumPy compares their shapes element-wise.
It starts with the trailing dimensions and works its way forward.  Two
dimensions are compatible when

1) they are equal, or
2) one of them is 1
```

条件很简单，即从两个数组的最后维度开始比较，如果该维度满足维度相等或者其中一个大小为 1，则可以实现广播。当然，维度相等时相当于未广播，所以严格的说广播仅适用于某一维度从 1 广播到 N；如果当前维度满足广播要求，则同时前移一个维度继续比较。为了直观理解这个广播条件，举个例子，下面的情况均满足广播条件：

```
  A      (4d array):  8 x 1 x 6 x 1
  B      (3d array):      7 x 1 x 5
  Result (4d array):  8 x 7 x 6 x 5

Here are some more examples::

  A      (2d array):  5 x 4
  B      (1d array):      1
  Result (2d array):  5 x 4

  A      (2d array):  5 x 4
  B      (1d array):      4
  Result (2d array):  5 x 4

  A      (3d array):  15 x 3 x 5
  B      (3d array):  15 x 1 x 5
  Result (3d array):  15 x 3 x 5

  A      (3d array):  15 x 3 x 5
  B      (2d array):      3 x 5
  Result (3d array):  15 x 3 x 5

  A      (3d array):  15 x 3 x 5
  B      (2d array):      3 x 1
  Result (3d array):  15 x 3 x 5
```

而如下例子则无法完成广播：

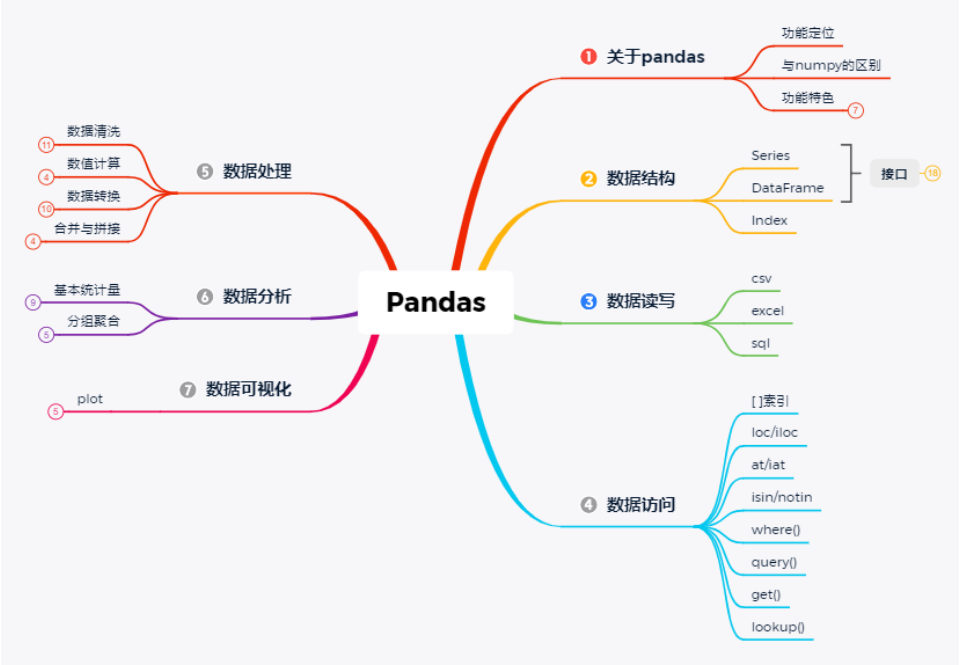
```
Here are examples of shapes that do not broadcast::

  A      (1d array):  3
  B      (1d array):  4 # trailing dimensions do not match

  A      (2d array):      2 x 1
  B      (3d array):  8 x 4 x 3 # second from last dimensions mismatched
```

好吧，以上例子其实都源自 numpy 官方文档。具体可参考../numpy/doc/Broadcasting.py 文件。另外，doc 包下还包括很多说明文档，对于深刻理解 numpy 运行机制大有裨益。

再补充一句：这里或许有人好奇，为什么必须要 1 对 N 才能广播，N 的任意因数（比如 N/2、N/3 等）不是都可以"合理"广播到 N 吗？对此，个人也曾有此困惑，我的理解是这里的合理只是数学意义下的合理，但数组表征值意义下往往不合理，因为缺乏解释性！比如 2 可以广播到 12，但此时该怎样理解这其中的广播意义呢？奇偶不同？那 3 广播到 12 呢？4 广播到 12 呢？还是欠缺解释性。所以 numpy 限制必须是 1 广播到 N 或者二者相等，才可以广播。

二、Pandas 入门详细教程

序言：前面系统性介绍了 Numpy 的入门基本知识，接下来则对 pandas 进行入门详细介绍，通过本篇你将系统性了解 pandas 为何会有数据分析界"瑞士军刀"的盛誉。



01 关于 pandas

pandas，python+data+analysis 的组合缩写，是 python 中基于 numpy 和 matplotlib 的第三方数据分析库，与后两者共同构成了 python 数据分析的基础工具包，享有数分三剑客之名。

正因为 pandas 是在 numpy 基础上实现，其核心数据结构与 numpy 的 ndarray 十分相似，但 pandas 与 numpy 的关系不是替代，而是互为补充。二者之间主要区别是：

从数据结构上看：

- numpy 的核心数据结构是 ndarray，支持任意维数的数组，但要求单个数组内所有数据是同质的，即类型必须相同；而 pandas 的核心数据结构是 series 和 dataframe，仅支持一维和二维数据，但数据内部可以是异构数据，仅要求同列数据类型一致即可
- numpy 的数据结构仅支持数字索引，而 pandas 数据结构则同时支持数字索引和标签索引

从功能定位上看：

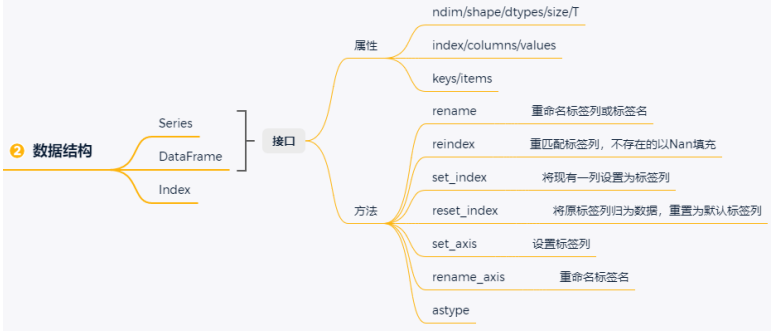
- numpy 虽然也支持字符串等其他数据类型，但仍然主要是用于数值计算，尤其是内部集成了大量矩阵计算模块，例如基本的矩阵运算、线性代数、fft、生成随机数等，支持灵活的广播机制
- pandas 主要用于数据处理与分析，支持包括数据读写、数值计算、数据处理、数据分析和数据可视化全套流程操作

pandas 主要面向数据处理与分析，主要具有以下功能特色：

- 按索引匹配的广播机制，这里的广播机制与 numpy 广播机制还有很大不同
- 便捷的数据读写操作，相比于 numpy 仅支持数字索引，pandas 的两种数据结构均支持标签索引，包括 bool 索引也是支持的
- 类比 SQL 的 join 和 groupby 功能，pandas 可以很容易实现 SQL 这两个核心功能，实际上，SQL 的绝大部分 DQL 和 DML 操作在 pandas 中都可以实现
- 类比 Excel 的数据透视表功能，Excel 中最为强大的数据分析工具之一是数据透视表，这在 pandas 中也可轻松实现
- 自带正则表达式的字符串向量化操作，对 pandas 中的一列字符串进行通函数操作，而且自带正则表达式的大部分接口
- 丰富的时间序列向量化处理接口
- 常用的数据分析与统计功能，包括基本统计量、分组统计分析等
- 集成 matplotlib 的常用可视化接口，无论是 series 还是 dataframe，均支持面向对象的绘图接口

正是由于具有这些强大的数据分析与处理能力，pandas 还有数据处理中"瑞士军刀"的美名。

02 数据结构



pandas 核心数据结构有两种，即一维的 series 和二维的 dataframe，二者可以分别看做是在 numpy 一维数组和二维数组的基础上增加了相应的标签信息。正因如此，可以从两个角度理解 series 和 dataframe：

- series 和 dataframe 分别是一维和二维数组，因为是数组，所以 numpy 中关于数组的用法基本可以直接应用到这两个数据结构，包括数据创建、切片访问、通函数、广播机制等
- series 是带标签的一维数组，所以还可以看做是**类字典**结构：标签是 key，取值是 value；而 dataframe 则可以看做是嵌套字典结构，其中列名是 key，每一列的 series 是 value。所以从这个角度讲，pandas 数据创建的一种灵活方式就是通过字典或者嵌套字典，同时也自然衍生出了适用于 series 和 dataframe 的类似字典访问的接口，即通过 loc 索引访问。

注意，这里强调 series 和 dataframe 是一个类字典结构而非真正意义上的字典，原因在于 series 中允许标签名重复、dataframe 中则允许列名和标签名均有重复，而这是一个真正字典所不允许的。

考虑 series 和 dataframe 兼具 numpy 数组和字典的特性，那么就不难理解二者的以下属性：

- ndim/shape/dtypes/size/T，分别表示了数据的维数、形状、数据类型和元素个数以及转置结果。其中，由于 pandas 允许数据类型是异构的，各列之间可能含有多种不同的数据类型，所以 dtype 取其复数形式 dtypes。与此同时，series 因为只有一列，所以数据类型自然也就只有一种，pandas 为了兼容二者，series 的数据类型属性既可以用 dtype 也可以用 dtypes 获取；而 dataframe 则只能用 dtypes。
- index/columns/values，分别对应了行标签、列标签和数据，其中数据就是一个格式向上兼容所有列数据类型的 array。为了沿袭字典中的访问习惯，还可以用 keys()访问标签信息，在 series 返回 index 标签，在 dataframe 中则返回 columns 列名；可以用 items()访问键值对，但一般用处不大。

这里提到了 index 和 columns 分别代表行标签和列标签，就不得不提到 pandas 中的另一个数据结构：Index，例如 series 中标签列、dataframe 中行标签和列标签均属于这种数据结构。既然是数据结构，就必然有数据类型 dtype 属性，例如数值型、字符串型或时间类型等，其类型绝大多数场合并不是我们关注的主体，但有些时候值得注意，如后文中提到的通过[]执行标签切片访问行的过程。此外，index 数据结构还有名字属性 name（默认为 None）、形状属性 shape 等。

```
df = pd.DataFrame([[1, 2], [2, 4]])
print('index: ', df.index)
print('index name: ', df.index.name)
print('columns dtype: ', df.columns.dtype)
```

```
index:  RangeIndex(start=0, stop=2, step=1)
index name:  None
columns dtype:  int64
```

关于 series 和 dataframe 数据结构本身，有大量的方法可用于重构结构信息：

- rename，可以对标签名重命名，也可以重置 index 和 columns 的部分标签列信息，接收标量（用于对标签名重命名）或字典（用于重命名行标签和列标签）
- reindex，接收一个新的序列与已有标签列匹配，当原标签列中不存在相应信息时，填充 NAN 或者可选的填充值
- set_index/reset_index，互为逆操作，前者是将已有的一列信息设置为标签列，而后者是将原标签列归为数据，并重置为默认数字标签
- set_axis，设置标签列，一次只能设置一列信息，与 rename 功能相近，但接收参数为一个序列更改全部标签列信息（rename 中是接收字典，允许只更改部分信息）
- rename_axis，重命名标签名，rename 中也可实现相同功能

在 pandas 早些版本中，除一维数据结构 series 和二维数据结构 dataframe 外，还支持三维数据结构 panel。这三者是构成递进包容关系，panel 即是 dataframe 的容器，用于存储多个 dataframe。2019 年 7 月，随着 pandas 0.25 版本的推出，pandas 团队宣布正式弃用 panel 数据结构，而相应功能建议由多层索引实现。

也正因为 pandas 这 3 种独特的数据结构，个人一度认为 pandas 包名解释为：pandas = panel + dataframe + series，根据维数取相应的首字母个数，从而构成 pandas，这是个人非常喜欢的一种关于 pandas 缩写的解释。

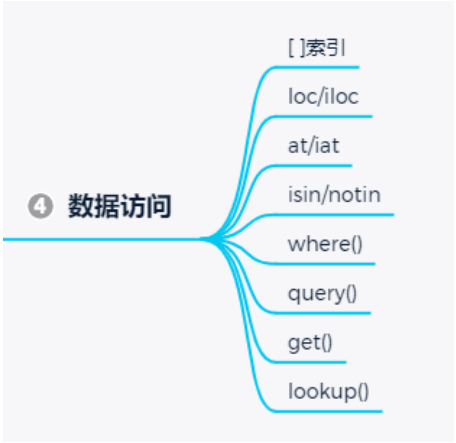
03 数据读写

pandas 支持大部分的主流文件格式进行数据读写，常用格式及接口为：

- 文本文件，主要包括 csv 和 txt 两种等，相应接口为 read_csv()和 to_csv()，分别用于读写数据
- Excel 文件，包括 xls 和 xlsx 两种格式均得到支持，底层是调用了 xlwt 和 xlrd 进行 excel 文件操作，相应接口为 read_excel()和 to_excel()
- SQL 文件，支持大部分主流关系型数据库，例如 MySQL，需要相应的数据库模块支持，相应接口为 read_sql()和 to_sql()

此外，pandas 还支持 html、json 等文件格式的读写操作。

04 数据访问



series 和 dataframe 兼具 numpy 数组和字典的结构特性，所以数据访问都是从这两方面入手。同时，也支持 bool 索引进行数据访问和筛选。

[]，这是一个非常便捷的访问方式，不过需区分 series 和 dataframe 两种数据结构理解：

- series：既可以用标签也可以用数字索引访问单个元素，还可以用相应的切片访问多个值，因为只有一维信息，自然毫无悬念
- dataframe：无法访问单个元素，只能返回一列、多列或多行：单值或多值（多个列名组成的列表）访问时按列进行查询，单值访问不存在列名歧义时还可直接用属性符号" ."访问。切片形式访问时按行进行查询，又区分数字切片和标签切片两种情况：当输入数字索引切片时，类似于普通列表切片；当输入标签切片时，执行**范围查询**（即无需切片首末值存在于标签列中），包含两端标签结果，无匹配行时返回为空，但要求标签切片类型与索引类型一致。例如，当标签列类型（可通过 df.index.dtype 查看）为时间类型时，若使用无法隐式转换为时间的字符串作为索引切片，则引发报错

```
df
```

	aa	ab	ac
2020-05-05	0	1	2
2020-05-06	3	4	5
2020-05-07	6	7	8
2020-05-08	9	10	11

```
df.index
```

```
DatetimeIndex(['2020-05-05', '2020-05-06', '2020-05-07', '2020-05-08'],
```

```
df['2020-05':'2020-05'] #全部匹配，返回全部结果
```

	aa	ab	ac
2020-05-05	0	1	2
2020-05-06	3	4	5
2020-05-07	6	7	8
2020-05-08	9	10	11

```
df['A':'B'] #切片为字符串类型，且无法隐式转换为日期切片类型，引发报错
```

```
ValueError                                Traceback (most recent call l
```

切片类型与索引列类型不一致时，引发报错

- loc/iloc，最为常用的两种数据访问方法，其中 loc 按标签值访问、iloc 按数字索引访问，均支持单值访问或切片查询。与[]访问类似，loc 按标签访问时也是执行范围查询，包含两端结果

• at/iat，loc 和 iloc 的特殊形式，不支持切片访问，仅可以用单个标签值或单个索引值进行访问，一般返回标量结果，除非标签值存在重复

• isin/notin，条件范围查询，即根据特定列值是否存在于指定列表返回相应的结果

• where，仍然是执行条件查询，但会返回全部结果，只是将不满足匹配条件的结果赋值为 NaN 或其他指定值，可用于筛选或屏蔽值

• query，按列对 dataframe 执行条件查询，一般可用常规的条件查询替代

• get，由于 series 和 dataframe 均可以看做是类字典结构，所以也可使用字典中的 get()方法，主要适用于不确定数据结构中是否包含该标签时，与字典的 get 方法完全一致

• lookup，loc 的一种特殊形式，分别传入一组行标签和列标签，lookup 解析成一组行列坐标，返回相应结果：

pandas 中支持大量的数据访问接口，但万变不离其宗：只要联想两种数据结构兼具 numpy 数组和字典的双重特性，就不难理解这些数据访问的逻辑原理。当然，重点还是掌握[]、loc 和 iloc 三种方法。

loc 和 iloc 应该理解为是 series 和 dataframe 的属性而非函数，应用 loc 和 iloc 进行数据访问就是根据属性值访问的过程

另外，在 pandas 早些版本中，还存在 loc 和 iloc 的兼容结构，即 ix，可混合使用标签和数字索引，但往往容易混乱，所以现已弃用

05 数据处理

pandas 最为强大的功能当然是数据处理和分析，可独立完成数据分析前的绝大部分数据预处理需求。简单归纳来看，主要可分为以下几个方面：

1 数据清洗

数据处理中的清洗工作主要包括对空值、重复值和异常值的处理：

①空值

▪ 判断空值，isna 或 isnull，二者等价，用于判断一个 series 或 dataframe 各元素值是否为空的 bool 结果。需注意对空值的界定：即 None 或 numpy.nan 才算空值，而空字符串、空列表等则不属于空值；类似地，notna 和 notnull 则用于判断是否非空

▪ 填充空值，fillna，按一定策略对空值进行填充，如常数填充、向前/向后填充等，也可通过 inplace 参数确定是否本地更改

▪ 删除空值，dropna，删除存在空值的整行或整列，可通过 axis 设置，也包括 inplace 参数

②重复值

▪ 检测重复值，duplicated，检测各行是否重复，返回一个行索引的 bool 结果，可通过 keep 参数设置保留第一行/最后一行/无保留，例如 keep=first 意味着在存在重复的多行时，首行被认为是合法的而可以保留

▪ 删除重复值，drop_duplicates，按行检测并删除重复的记录，也可通过 keep 参数设置保留项。由于该方法默认是按行进行检测，如果存在某个需要需要按列删除，则可以先转置再执行该方法

③异常值，判断异常值的标准依赖具体分析数据，所以这里仅给出两种处理异常值的可选方法

▪ 删除，drop，接受参数在特定轴线执行删除一条或多条记录，可通过 axis 参数设置是按行删除还是按列删除

▪ 替换，replace，非常强大的功能，对 series 或 dataframe 中每个元素执行按条件替换操作，还可开启正则表达式功能

2 数值计算

由于 pandas 是在 numpy 的基础上实现的，所以 numpy 的常用数值计算操作在 pandas 中也适用：

• 通函数 ufunc，即可以像操作标量一样对 series 或 dataframe 中的所有元素执行同一操作，这与 numpy 的特性是一致的，例如前文提到的 replace 函数，本质上可算作是通函数。如下实现对数据表中逐元素求平方

• 广播机制，即当维度或形状不匹配时，会按一定条件广播后计算。由于 pandas 是带标签的数组，所以在广播过程中会自动按标签匹配进行广播，而非类似 numpy 那种纯粹按顺序进行广播。例如，如下示例中执行一个 dataframe 和 series 相乘，虽然二者维度不等、大小不等、标签顺序也不一致，但仍能按标签匹配得到预期结果

```python
df = pd.DataFrame(data=[[100, 90, 80], [80, 78, 87]],
                  index=['一班', '二班'],
                  columns=['数学平均分', '语文平均分', '英语平均分'])
df
```

	数学平均分	语文平均分	英语平均分
一班	100	90	80
二班	80	78	87

```python
sr = pd.Series(data=[20, 10, 15], index=['一班', '三班', '二班'])
sr
```

```
一班    20
三班    10
二班    15
dtype: int64
```

```python
df.mul(sr, axis=0)  # 根据各班各课程平均分和班级人数求总分
```

	数学平均分	语文平均分	英语平均分
一班	2000.0	1800.0	1600.0
三班	NaN	NaN	NaN
二班	1200.0	1170.0	1305.0

- 字符串向量化，即对于数据类型为字符串格式的一列执行向量化的字符串操作，本质上是调用 series.str 属性的系列接口，完成相应的字符串操作。尤为强大的是，除了常用的字符串操作方法，str 属性接口中还集成了正则表达式的大部分功能，这使得 pandas 在处理字符串列时，兼具高效和强力。例如如下代码可用于统计每个句子中单词的个数

需注意的是，这里的字符串接口与 python 中普通字符串的接口形式上很是相近，但二者是不一样的。

- 时间类型向量化操作，如字符串一样，在 pandas 中另一个得到"优待"的数据类型是时间类型，正如字符串列可用 str 属性调用字符串接口一样，时间类型列可用 dt 属性调用相应接口，这在处理时间类型时会十分有效。

3 数据转换

前文提到，在处理特定值时可用 replace 对每个元素执行相同的操作，然而 replace 一般仅能用于简单的替换操作，所以 pandas 还提供了更为强大的数据转换方法

- map，**适用于 series 对象**，功能与 python 中的普通 map 函数类似，即对给定序列中的每个值执行相同的映射操作，不同的是 series 中的 map 接口的映射方式既可以是一个函数，也可以是一个字典

```
df  #原数据
```

	aa	ab	ac
AA	0	1	2
AB	3	4	5
BA	6	7	8
BB	9	10	11

```
df['aa'].map({0:'zero', 6:'six'}) #字典转换
```

```
AA    zero
AB     NaN
BA     six
BB     NaN
Name: aa, dtype: object
```

```
df['aa'].map(lambda x:x*2) #用函数实现aa列乘方
```

```
AA     0
AB     6
BA    12
BB    18
Name: aa, dtype: int64
```

- apply，**既适用于 series 对象也适用于 dataframe 对象**，但对二者处理的粒度是不一样的：apply 应用于 series 时是逐元素执行函数操作；apply 应用于 dataframe 时是逐行或者逐列执行函数操作（通过 axis 参数设置对行还是对列，默认是行），仅接收函数作为参数

```
df  #原数据
```

	aa	ab	ac
AA	0	1	2
AB	3	4	5
BA	6	7	8
BB	9	10	11

```
df.apply(lambda x:x/np.max(x)) #各行归一化
```

	aa	ab	ac
AA	0.000000	0.1	0.181818
AB	0.333333	0.4	0.454545
BA	0.666667	0.7	0.727273
BB	1.000000	1.0	1.000000

```
df['aa'].apply(lambda x:x*2) #aa列乘方
```

```
AA     0
AB     6
BA    12
BB    18
Name: aa, dtype: int64
```

- applymap，**仅适用于 dataframe 对象**，且是对 dataframe 中的每个元素执行函数操作，从这个角度讲，与 replace 类似，applymap 可看作是 dataframe 对象的通函数。

4 合并与拼接

pandas 中又一个重量级数据处理功能是对多个 dataframe 进行合并与拼接，对应 SQL 中两个非常重要的操作：union 和 join。pandas 完成这两个功能主要依赖以下函数：

- concat，与 numpy 中的 concatenate 类似，但功能更为强大，可通过一个 axis 参数设置是横向或者拼接，**要求非拼接轴向标签唯一**（例如沿着行进行拼接时，要求每个 df 内部列名是唯一的，但两个 df 间可以重复，毕竟有相同列才有拼接的实际意义）
- merge，完全类似于 SQL 中的 join 语法，仅支持横向拼接，通过设置连接字段，实现对同一记录的不同列信息连接，支持 inner、left、right 和 outer4 种连接方式，但只能实现 SQL 中的等值连接
- join，语法和功能与 merge 一致，不同的是 merge 既可以用 pandas 接口调用，也可以用 dataframe 对象接口调用，而 join 则只适用于 dataframe 对象接口
- append，concat 执行 axis=0 时的一个简化接口，类似列表的 append 函数一样

实际上，concat 通过设置 axis=1 也可实现与 merge 类似的效果，二者的区别在于：merge 允许连接字段重复，类似一对多或者多对一连接，此时将产生笛卡尔积结果；而 concat 则不允许重复，仅能一对一拼接。

```
pd.concat([df1, df2], join='outer', axis=1, sort=True)
```

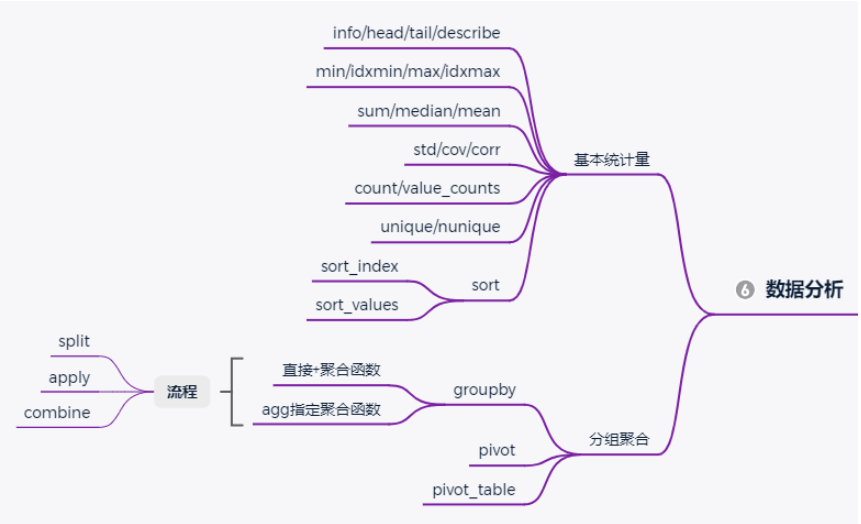
	C	B	A	E	D	C
1	1.0	1.0	1.0	2.0	2.0	2.0
2	1.0	1.0	1.0	NaN	NaN	NaN
3	1.0	1.0	1.0	NaN	NaN	NaN
4	NaN	NaN	NaN	2.0	2.0	2.0
5	NaN	NaN	NaN	2.0	2.0	2.0

```
pd.merge(df1, df2, how='outer', left_index=True, right_index=True, sort=True)
```

	C_x	B	A	E	D	C_y
1	1.0	1.0	1.0	2.0	2.0	2.0
2	1.0	1.0	1.0	NaN	NaN	NaN
3	1.0	1.0	1.0	NaN	NaN	NaN
4	NaN	NaN	NaN	2.0	2.0	2.0
5	NaN	NaN	NaN	2.0	2.0	2.0

通过设置参数，concat 和 merge 实现相同效果

06 数据分析



pandas 中的另一大类功能是数据分析，通过丰富的接口，可实现大量的统计需求，包括 Excel 和 SQL 中的大部分分析过程，在 pandas 中均可以实现。

1 基本统计量

pandas 内置了丰富的统计接口，这是与 numpy 是一致的，同时又包括一些常用统计信息的集成接口。

- info，展示行标签、列标签、以及各列基本信息，包括元素个数和非空个数及数据类型等
- head/tail，从头/尾抽样指定条数记录
- describe，展示数据的基本统计指标，包括计数、均值、方差、4 分位数等，还可接收一个百分位参数列表展示更多信息

count、value_counts，前者既适用于 series 也适用于 dataframe，用于按列统计个数，实现忽略空值后的计数；而 value_counts 则仅适用于 series，执行分组统计，并默认按频数高低执行降序排列，在统计分析中很有用

unique、nunique，也是仅适用于 series 对象，统计唯一值信息，前者返回唯一值结果列表，后者返回唯一值个数(number of unique)

```
sr = pd.Series(['A', 'A', 'B'])
sr.unique()
```

```
array(['A', 'B'], dtype=object)
```

```
sr.nunique()
```

```
2
```

- sort_index、sort_values，既适用于 series 也适用于 dataframe，sort_index 是对标签列执行排序，如果是 dataframe 可通过 axis 参数设置是对行标签还是列标签执行排序；sort_values 是按值排序，如果是 dataframe 对象，也可通过 axis 参数设置排序方向是行还是列，同时根据 by 参数传入指定的行或者列，可传入多行或多列并分别设置升序降序参数，非常灵活。另外，在标签列已经命名的情况下，sort_values 可通过 by 标签名实现与 sort_index 相同的效果。

```
df
```

	aa	ab	ac
AA	0	1	2
AB	3	4	5
BA	6	7	8
BB	9	10	11

原数据表

```
df.index.name = 'idx'
df
```

	aa	ab	ac
idx			
AA	0	1	2
AB	3	4	5
BA	6	7	8
BB	9	10	11

索引列命名

```
df.sort_index(ascending=False)
```

	aa	ab	ac
idx			
BB	9	10	11
BA	6	7	8
AB	3	4	5
AA	0	1	2

按索引排序

```
df.sort_values(by='idx', ascending=False)
```

	aa	ab	ac
idx			
BB	9	10	11
BA	6	7	8
AB	3	4	5
AA	0	1	2

按值排序
值传入索引名

2 分组聚合

pandas 的另一个强大的数据分析功能是分组聚合以及数据透视表，前者堪比 SQL 中的 groupby，后者媲美 Excel 中的数据透视表。

- groupby，类比 SQL 中的 group by 功能，即按某一列或多列执行分组。一般而言，分组的目的是为了后续的聚合统计，所有 groupby 函数一般不单独使用，而需要级联其他聚合函数共同完成特定需求，例如分组求和、分组求均值等。级联其他聚合函数的方式一般有两种：单一的聚合需求用 groupby+聚合函数即可，复杂的大量聚合则可借用 agg 函数，agg 函数接受多种参数形式作为聚合函数，功能更为强大。
- pivot，pivot 英文有"支点"或者"旋转"的意思，排序算法中经典的快速排序就是不断根据 pivot 不断将数据二分，从而加速排序过程。用在这里，实际上就是执行行列重整。例如，以某列取值为重整后行标签，以另一列取值作为重整后的列标签，以其他列取值作为填充 value，即实现了数据表的行列重整。以 SQL 中经典的学生成绩表为例，给定原始学生—课程—成绩表，需重整为学生 vs 课程的成绩表，则可应用 pivot 实现：

```
df = pd.DataFrame({"姓名": ['小王', '小王', '小李', '小李'],
                   "课程": ['C', 'Java', 'Python', 'C'],
                   "成绩": [80, 90, 78, 90]})
df
```

	姓名	课程	成绩
0	小王	C	80
1	小王	Java	90
2	小李	Python	78
3	小李	C	90

```
df.pivot(index='姓名', columns='课程', values='成绩')
```

课程	C	Java	Python
姓名			
小李	90.0	NaN	78.0
小王	80.0	90.0	NaN

另外，还有一对函数也常用于数据重整，即 stack 和 unstack，其中 unstack 执行效果与 pivot 非常类似，而 stack 则是 unstack 的逆过程。

- pivot_table，有了 pivot 就不难理解 pivot_table，实际上它是在前者的基础上增加了聚合的过程，类似于 Excel 中的数据透视表功能。仍然考虑前述学生成绩表的例子，但是再增加一列班级信息，需求是统计各班级每门课程的平均分。由于此时各班的每门课成绩信息不唯一，所以直接用 pivot 进行重整会报错，此时即需要对各班各门课程成绩进行聚合后重整，比如取平均分。

07 数据可视化

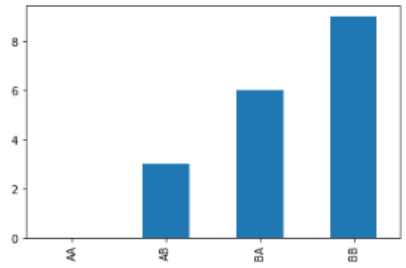
pandas 集成了 matplotlib 中的常用可视化图形接口，可通过 series 和 dataframe 两种数据结构面向对象的接口方式简单调用。关于面向对象接口和 plt 接口绘图方式的区别，可参考前文。

两种数据结构作图，区别仅在于 series 是绘制单个图形，而 dataframe 则是绘制一组图形，且在 dataframe 绘图结果中以列名为标签自动添加 legend。另外，均支持两种形式的绘图接口：

- plot 属性+相应绘图接口，如 plot.bar()用于绘制条形图
- plot()方法并通过传入 kind 参数选择相应绘图类型，如 plot(kind='bar')

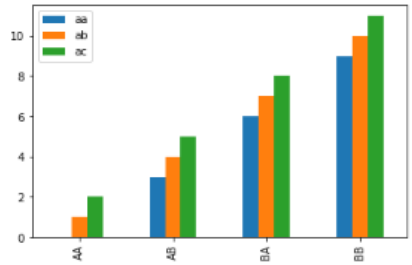
```
df['aa'].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dbda281240>
```




```
df.plot.bar()  # dataframe绘图，自动添加Legend标签
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dbd8fc4fd0>
```

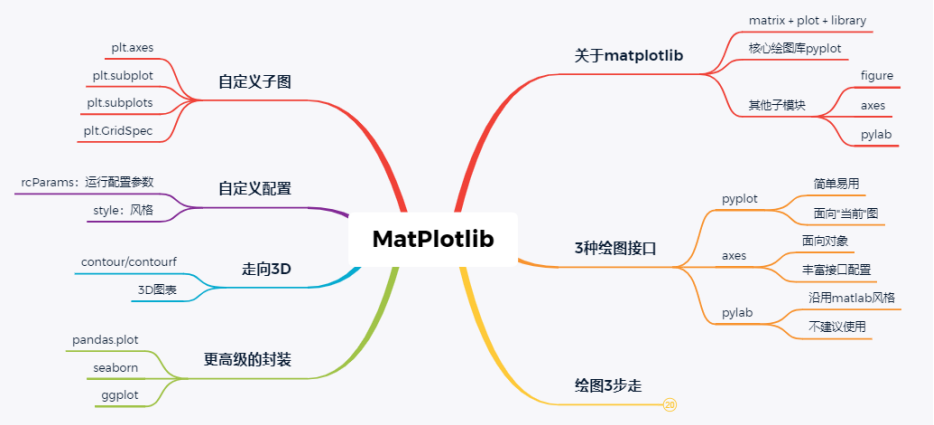


不过，pandas 绘图中仅集成了常用的图表接口，更多复杂的绘图需求往往还需依赖 matplotlib 或者其他可视化库。

三、Matplotlib 入门详细教程

序言：前文分别介绍了 numpy 和 pandas 的常用接口及使用，并对部分接口方法进行了详细对比。与之齐名，matplotlib 作为数据科学的的另一必备库，算得上是 python 可视化领域的元老，更是很多高级可视化库的底层基础，其重要性不言而喻。

本篇对 matplotlib 进行系统性介绍，不会面面俱到，但求体系完备、详略得当。



01 关于 matplotlib

matplotlib 是 python 的一个绘图库，与 numpy、pandas 共享数据科学三剑客的美誉，也是很多高级可视化库的基础。matplotlib 不是 python 内置库，调用前需手动安装，且需依赖 numpy 库。截至当前，matplotlib 发行版本号为 3.2.1，适配 python3.6 及以上版本。

matplotlib，是 matrix + plot + library 的缩写，虽然命名很是直观，但个人接触之初却是常常不禁嗤之以鼻：

- 类比 numpy、pandas、sklearn 这些简洁易写的库名，matplotlib 由三个单词缩略而成，冗余得多；尤其是后面加了个 lib，好像不加 lib 就不是库？
- matplotlib 自身名字长也就罢了，但调用它的时候居然还不能简单的直接调用，而是要用它的子模块 pyplot。那既然 pyplot 是核心绘图模块，为什么不把其接口引入到顶层呢？那样直接 import matplotlib 就行，而无需每次都 import matplotlib.pyplot as plt 了
- 虽然 pyplot 是 matplotlib 下的子模块，但二者的调用关系却不是 matplotlib 调用 pyplot，而是在 pyplot 中调用 matplotlib，略显本末倒置？

当然，我等作为使用者、调包侠，自然是无法领会开发者的独特考虑，也绝无资格对其评三道四，仅做吐槽一二。

```
from cycler import cycler
import matplotlib
import matplotlib.colorbar
import matplotlib.image
from matplotlib import rcsetup, style
from matplotlib import _pylab_helpers, interactive
from matplotlib import cbook
from matplotlib.cbook import dedent, deprecated, silent_list, warn_deprecated
from matplotlib import docstring
from matplotlib.backend_bases import FigureCanvasBase
from matplotlib.figure import Figure, figaspect
from matplotlib.gridspec import GridSpec
from matplotlib import rcParams, rcParamsDefault, get_backend, rcParamsOrig
from matplotlib import rc_context
from matplotlib.rcsetup import interactive_bk as _interactive_bk
from matplotlib.artist import getp, get, Artist
from matplotlib.artist import setp as _setp
from matplotlib.axes import Axes, Subplot
from matplotlib.projections import PolarAxes
from matplotlib import mlab  # for _csv2rec, detrend_none, window_hanning
from matplotlib.scale import get_scale_docs, get_scale_names
```

pyplot 部分调用模块

前面说到，调用 matplotlib 库绘图一般是用 pyplot 子模块，其集成了绝大部分常用方法接口，查看 pyplot 源码文件可以发现，它内部调用了 matplotlib 路径下的大部分子模块（不是全部），共同完成各种丰富的绘图功能。其中有两个需要重点指出：figure 和 axes，其中前者为所有绘图操作定义了顶层类对象 Figure，相当于是提供了画板；而后者则定义了画板中的每一个绘图对象 Axes，相当于画板内的各个子图。换句话说，figure 是 axes 的父容器，而 axes 是 figure 的内部元素，而我们常用的各种图表、图例、坐标轴等则又是 axes 的内部元素。

当然，之所以不能称 pyplot 为一级命名空间的原因，不仅仅在于它在形式上隶属于 matplotlib，最主要的在于它还不算是 matplotlib 的“独裁者”，因为 matplotlib 的另一个重要模块——pylab——或许称得上是真正意义上的集大成者：pylab 被定位是 python 中对 MATLAB 的替代产品，也就是说凡是 MATLAB 可以实现的功能，pylab 通通都要有，例如矩阵运算（包括常规矩阵运算、线性代数、随机数、FFT 等）、绘图功能等等。

```
## We are still importing too many things from mlab; more cleanup is needed.

from matplotlib.mlab import (
    demean, detrend, detrend_linear, detrend_mean, detrend_none,
    window_hanning, window_none)

from matplotlib import cbook, mlab, pyplot as plt
from matplotlib.pyplot import *

from numpy import *
from numpy.fft import *
from numpy.random import *
from numpy.linalg import *

import numpy as np
import numpy.ma as ma
```

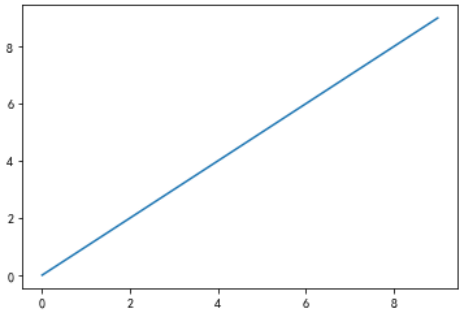
pylab 导入的那些重量级模块

至此，关于 matplotlib 的 pyplot 和 pylab 两个子模块，我们可以得出 2 点结论：

- pyplot 的功能定位决定其不能成为一级命名空间：即便是寻找 matplotlib 的替代包名，那么也该是 pylab 而不是 pyplot
- 简单地讲，以后也不用 import numpy 和 import matplotlib.pyplot 了，直接 import matplotlib.pylab 就够了，毕竟它集成了二者的全部功能

```
import matplotlib.pylab as plb
print('创建矩阵', plb.array([1, 2], dtype=int))  # pylab调用了numpy全部接口
print('生成随机数', plb.random())  # pylab调用了numpy.random子模块全部接口
plb.plot(range(10))  # pylab调用了pyplot全部接口
```

```
创建矩阵 [1 2]
生成随机数 0.7998278327099363
[<matplotlib.lines.Line2D at 0x23d1feee6a0>]
```



pylab 集成了 numpy 和 pyplot 全部功能

当了解 pylab 模块功能之后，才真正理解开发者的深谋远虑：原以为 matplotlib 的意思是"面向矩阵的绘图库"，哪知其真正意义是"矩阵+绘图库"，绘图只是它的一半。不过，也正因为 pylab 模块集成了过多的功能，直接调用并不是一个明智的选择，官方已不建议用其绘图。

注：按照惯例，本文后续多以 plt 作为 matplotlib.pyplot 别名使用。

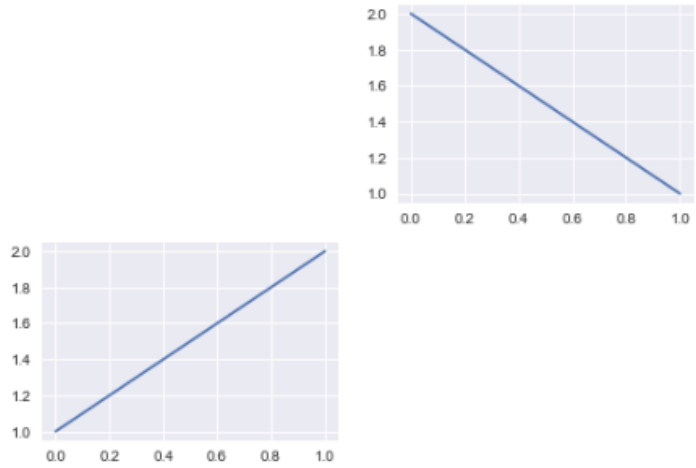
02 三种绘图接口

用 matplotlib 绘制可视化图表，主要有 3 种接口形式：

- plt 接口，例如常用的 plt.plot()，用官方文档的原话，它是 matplotlib 的一个 state-based 交互接口，相关操作不面向特定的实例对象，而是面向"当前"图
- 面向对象接口，这里的面向对象主要是指 Figure 和 Axes 两类对象。前文提到，Figure 提供了容纳多个 Axes 的画板，而 Axes 则是所有图标数据、图例配置等绘图形元素的容器。面向对象的绘图，就是通过调用 Figure 或 Axes 两类实例的方法完成绘图的过程（当然，Figure 和 Axes 发挥的作用是不同的）。通俗的说，就是将 plt 中的图形赋值给一个 Figure 或 Axes 实例，方便后续调用操作
- pylab 接口，如前所述，其引入了 numpy 和 pyplot 的所有接口，自然也可用于绘制图表，仍然可看做是 pyplot 接口形式。因其过于庞大官方不建议使用

```
ax = plt.subplot(222)  # 创建一个子图，并赋值给ax实例
plt.subplot(223)
plt.plot([1, 2])  # plt接口绘图，"当前"图是最近创建的subplot(223)
ax.plot([2, 1])  # 面向对象绘图，绘图对象是ax，即subplot(222)
```

```
[<matplotlib.lines.Line2D at 0x153a68c5eb8>]
```



plt 接口和面向对象接口混合绘图

鉴于 pylab 的特殊性，matplotlib 绘图主要采用前 2 种方式。而在二者之间：

- 如果是简单的单图表绘制，或者是交互实验环境，则 plt 接口足以满足需要，且操作简单易用
- 如果是多图表绘制，需要相对复杂的图例配置和其他自定义设置，那么毫无疑问面向对象接口绘图是当之无愧的不二选择

需要指出，Axes 从形式上是坐标轴 axis 一词的复数形式，但意义上却远非 2 个或多个坐标轴那么简单：如果将 Figure 比作是画板的话，那么 Axes 就是画板中的各个子图，这个子图提供了真正用于绘图的空间，除了包含纯粹的两个坐标轴(axes)外，自然还包括图形、图例等。所以准确的讲，如果说 Axes 和坐标轴有何关联的话，那么 Axes 应该算是广义的坐标轴，或简单称之为子图即可。

03 绘图 3 步走

如同把大象装进冰箱需要 3 步一样，用 matplotlib 绘图一般也可以分 3 步。下面以 plt 接口绘图为例，面向对象接口绘图流程完全一致，仅仅是个别接口方法名略有改动：

①创建画板，包括创建 figure 和 axes 对象，常用有 3 种方法

• plt.figure，主要接收一个元组作为 figsize 参数设置图形大小，返回一个 figure 对象用于提供画板

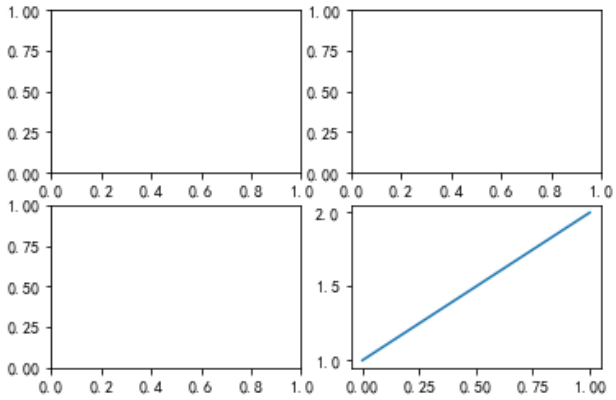
• plt.axes，接收一个 figure 或在当前画板上添加一个子图，返回该 axes 对象，并将其设置为"当前"图，缺省时会在绘图前自动添加

• plt.subplot，主要接收 3 个数字或 1 个 3 位数（自动解析成 3 个数字，要求解析后数值合理）作为子图的行数、列数和当前子图索引，索引从 1 开始（与 MATLAB 保存一致），返回一个 axes 对象用于绘图操作。这里，可以理解成是先隐式执行了 plt.figure，然后在创建的 figure 对象上添加子图，并返回当前子图实例

• plt.subplots，主要接收一个行数 nrows 和列数 ncols 作为参数（不含第三个数字），创建一个 figure 对象和相应数量的 axes 对象，同时返回该 figure 对象和 axes 对象嵌套列表，并默认选择最后一个子图作为"当前"图

```
fig, axes = plt.subplots(2, 2)
print('fig: ', fig)
print('axes: ', axes)
plt.plot([1, 2])
```

```
fig:  Figure(432x288)
axes:  [[<matplotlib.axes._subplots.AxesSubplot object at 0x0000023D2B1B4390>
  <matplotlib.axes._subplots.AxesSubplot object at 0x0000023D2B1F0320>]
 [<matplotlib.axes._subplots.AxesSubplot object at 0x0000023D2B21E860>
  <matplotlib.axes._subplots.AxesSubplot object at 0x0000023D2B24BDD8>]]
[<matplotlib.lines.Line2D at 0x23d2b2863c8>]
```



plt.subplots 同时返回 figure 和 axes 实例
默认将最后一个 axes 子图作为"当前"图

②绘制图表，常用图表形式包括：

▪ plot，折线图或点图，实际是调用了 line 模块下的 Line2D 图表接口

▪ scatter，散点图，常用于表述两组数据间的分布关系，也可由特殊形式下的 plot 实现

▪ bar/barh，条形图或柱状图，常用于表达一组离散数据的大小关系，比如一年内每个月的销售额数据；默认竖直条形图，可选 barh 绘制水平条形图

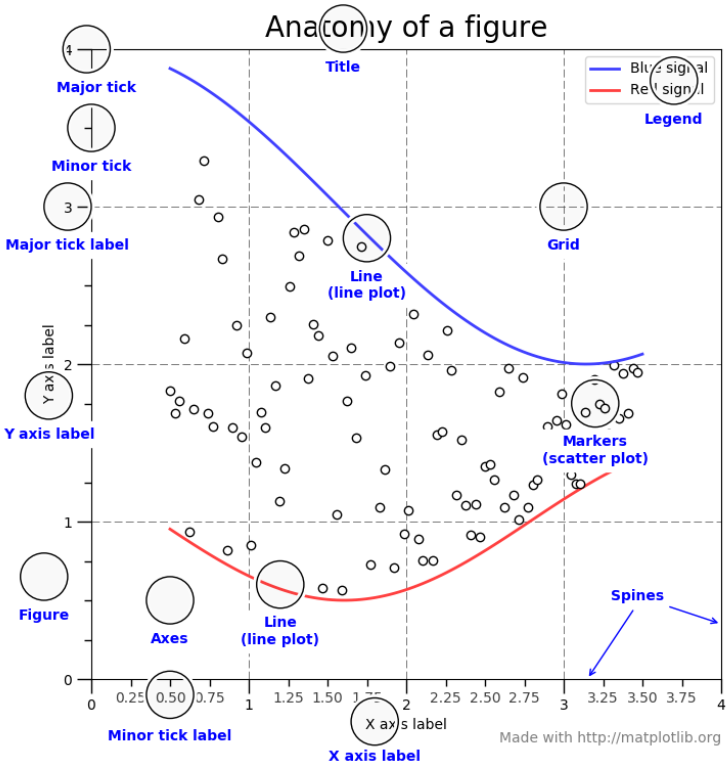
▪ hist，直方图，形式上与条形图很像，但表达意义却完全不同：直方图用于统计一组连续数据的分区间分布情况，比如有 1000 个正态分布的随机抽样，

那么其直方图应该是大致满足钟型分布；条形图主要是适用于一组离散标签下的数量对比

- pie，饼图，主要用于表达构成或比例关系，一般适用于少量对比
- imshow，显示图像，根据像素点数据完成绘图并显示

③配置图例：对所绘图形进一步添加图例元素，例如设置标题、坐标轴、文字说明等，常用接口如下：

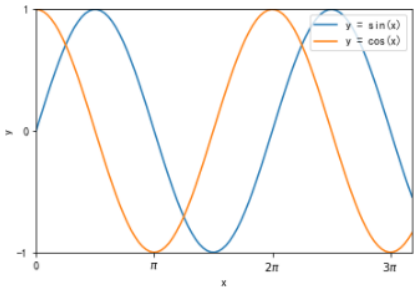
- title，设置图表标题
- axis/xlim/ylim，设置相应坐标轴范围，其中 axis 是对后 xlim 和 ylim 的集成，接受 4 个参数分别作为 x 和 y 轴的范围参数
- grid，添加图表网格线
- legend，在图表中添加 label 图例参数后，通过 legend 进行显示
- xlabel/ylabel，分别用于设置 x、y 轴标题
- xticks/yticks，分别用于自定义坐标轴刻度显示
- text/arrow/annotation，分别在图例指定位置添加文字、箭头和标记，一般很少用



关于图例配置的官方解释

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, label='y = sin(x)')
plt.plot(x, y2, label='y = cos(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.axis([0, 10, -1, 1])
plt.legend(loc=1)
plt.xticks([0, 3.1416, 6.2832, 9.4248], ['0', '$\pi$', '$2\pi$', '$3\pi$'])
plt.yticks([-1, 0, 1])
```

```
([<matplotlib.axis.YTick at 0x23d2cca94e0>,
  <matplotlib.axis.YTick at 0x23d2ccc5400>,
  <matplotlib.axis.YTick at 0x23d2cc942e8>],
 <a list of 3 Text yticklabel objects>)
```



plt 接口绘图中配置常用图例

前面提到，绘图接口有 2 种形式，分别是面向"当前"图的 plt 接口和面向对象接口，在这 2 种方式的相应接口中，多数接口名是一致的，例如：plt.plot()和 axes.plot()、plt.legend()和 axes.legend()，但也有一些不一致的接口：

- plt.axes()——fig.add_axes()
- plt.subplot()——fig.add_subplot()
- plt.GridSped()——fig.add_gridspec()
- plt.xlabel()——axes.set_xlabel()
- plt.ylabel()——axes.set_ylabel()
- plt.xlim()——axes.set_xlim()
- plt.ylim()——axes.set_ylim()
- plt.title()——axes.set_title()

对此，一方面两类接口虽然略有区别，但也还算有规律；另一方面，在面向对象绘图配置图例时，有更为便捷的设置图例接口 axes.set()，其可以接收多种参数一次性完成所有配置，这也正是面向对象绘图的强大之处。

04 自定义子图

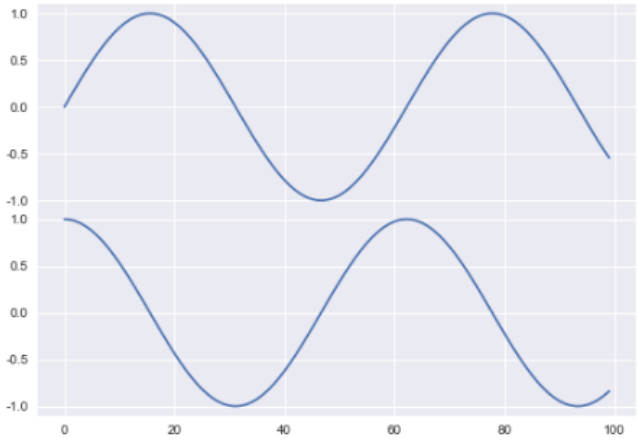
前面提到，figure 为绘图创建了画板，而 axes 基于当前画板创建了 1 个或多个子图对象。为了创建各种形式的子图，matplotlib 主要支持 4 种添加子图的方式。

常用的添加子图的方法莫过于 subplot 和 subplots 两个接口，其中前者用于一次添加一个子图，而后者则是创建一组子图。

除此之外，plt.axes 也可通过接收尺寸参数实现多子图绘制：在添加子图时传入一个含有 4 个数值的元组，分别表示子图的底坐标和左坐标（设置子图原点位置）、宽度和高度（设置子图大小），从而间接实现子图仅占据画板的一块子区域。相应的方法接口在面向对象接口中是 fig.add_axes()，仅仅是接口名字不同，但参数和原理是一致的。例如：

```
plt.figure()
x = np.linspace(0, 10, 100)
plt.axes((0.1, 0.5, 0.8, 0.4))#设置上子图位置
plt.plot(np.sin(x))
plt.axes((0.1, 0.1, 0.8, 0.4))#设置下子图位置
plt.plot(np.cos(x))
```

```
[<matplotlib.lines.Line2D at 0x153a5f4c5c0>]
```



应用 plt.axes 绘制多子图

通过 axes 绘制多子图，应对简单需求尚可，但面对复杂图表绘制时难免过于繁琐：需要手工计算各子图的原点位置和大小，意味着可能需要多次尝试。此时，可选的另一种绘制多子图的接口是 plt.GridSpec。实际上，GridSpec 只是对 subplot 接口的一个变形，本质上仍然是执行类似 subplot 多子图流程：通过切片将多子图合并，实现不规则多子图的绘制。与 subplot、axes 在面向对象和 plt 两类绘图接口间的区别类似，GridSpec 在面向对象时的接口为 add_gridspec()，具体可参考官网案例。

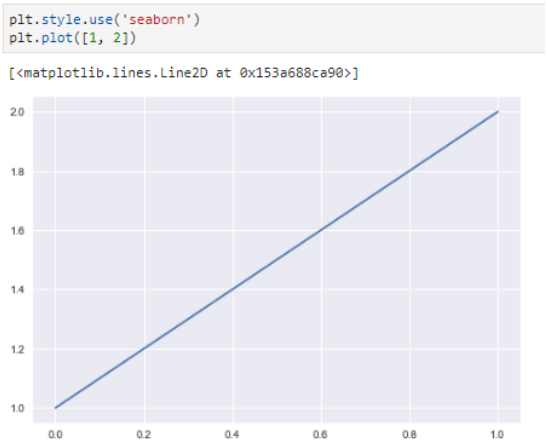
05 自定义配置

实际上，前述在配置图例过程中，每次绘制都需要进行大量自定义代码设置（这也是 matplotlib 的一个短板），在少量绘图工作时尚可接受，但在大量相似绘图存在重复操作时，仍然采取这一方法不会是一个明智的选择（虽然也可以简单的封装成一个函数）。

为此，matplotlib 提供了自定义参数实现批量配置——rcParams，全称 runtime configuration Parameters，即运行时配置参数。顾名思义，就是在 python 程序运行时临时执行的配置参数。rcParams 是一个字典格式，当前共有 299 个

键值对，分别对应一组参数配置选项。其中用得最多的可能是通过设置字体和减号编码来解决乱码的问题，但实际上它的功能强大之处可远非如此。

另一个简单易用的自定义配置选项是style，即设置绘图风格，最早在matplotlib1.4版本中引入，当前共支持 26 种绘图风格，这里的绘图风格类似于很多 IDE 支持不同主题。可以通过 plt.style.available 命令查看，返回一个可选风格的列表。例如，以下命令设置绘图为 senborn 风格



设置 seaborn 绘图风格

06 走向 3D

在可视化愈发重要的当下，matplotlib 当然不仅支持简单的 2D 图表绘制，其也提供了对 3D 绘图的丰富接口。

- contour，实际上是一个伪 3D 图形，仍然是在 2 维空间绘图，但可以表达 3 维信息。例如在机器学习中，contour 常用于绘制分类算法的超平面

如果需要绘制真 3D 图形，则需要额外导入 matplotlib 专用 3D 绘图库：mpl_toolkits，包括 3D 版的 Axes 对象和常用图表的 3D 版：

- plot3D，3D 版 plot，可用于绘制 3 维空间的折线图或点图
- scatter3D，3 维散点图
- bar3D，3 维条形图
- contour3D，3 维等高线

07 更高级的封装

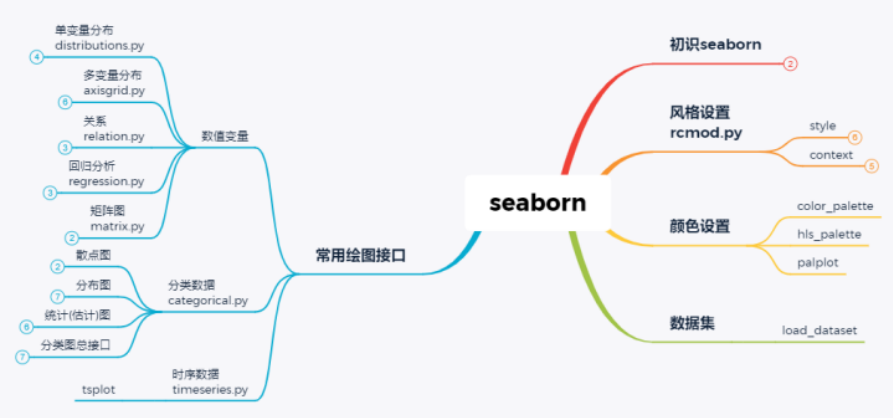
matplotlib 提供了大量丰富的可视化绘图接口，但仍然存在短板：例如绘图操作略显繁琐、图表不够美观。为此，在 matplotlib 基础上产生了一些封装更为便捷的可视化库，实现更为简单易用的接口和美观的图表形式，包括：

- pandas.plot，一个最直接的对 matplotlib 绘图的封装，接口方法非常接近
- seaborn，是对 matplotlib 的高级封装，具有更为美观的图形样式和颜色配置，并提供了常用的统计图形接口，如 pairplot()适用于表达多组数据间的关系

- ggplot，也是对 matplotlib 进行二次封装的可视化库，主要适用于 pandas 的 DataFrame 数据结构

四、Seaborn 入门详细教程

导读：前文分别对 python 数据分析三剑客（numpy、pandas、matplotlib）进行了逐一详细入门介绍，接下来推出系列第 4 篇教程：seaborn。这是一个基于 matplotlib 进行高级封装的可视化库，相比之下，绘制图表更为集成化、绘图风格具有更高的定制性。



01 初始 seaborn

seaborn 是 python 中的一个可视化库，是对 matplotlib 进行二次封装而成，既然是基于 matplotlib，所以 seaborn 的很多图表接口和参数设置与其很是接近。相比 matplotlib 而言，个人认为 seaborn 的几个鲜明特点如下：

- 绘图接口更为集成，可通过少量参数设置实现大量封装绘图
- 多数图表具有统计学含义，例如分布、关系、统计、回归等
- 对 Pandas 和 Numpy 数据类型支持非常友好
- 风格设置更为多样，例如风格、绘图环境和颜色配置等

正是由于 seaborn 的这些特点，在进行 EDA（Exploratory Data Analysis，探索性数据分析）过程中，seaborn 往往更为高效。然而也需指出，seaborn 与 matplotlib 的关系是互为补充而非替代：多数场合中 seaborn 是绘图首选，而在某些特定场景下则仍需用 matplotlib 进行更为细致的个性化定制。

按照惯例，后文将 seaborn 简写为 sns。至于 seaborn 简写为 sns 而非 sbn 的原因，感兴趣者可自行查阅(关键词：why import seaborn as sns?)。

02 风格设置

seaborn 的风格设置主要分为两类，其一是风格(style)设置，其二是环境(context)设置。

1. 风格设置

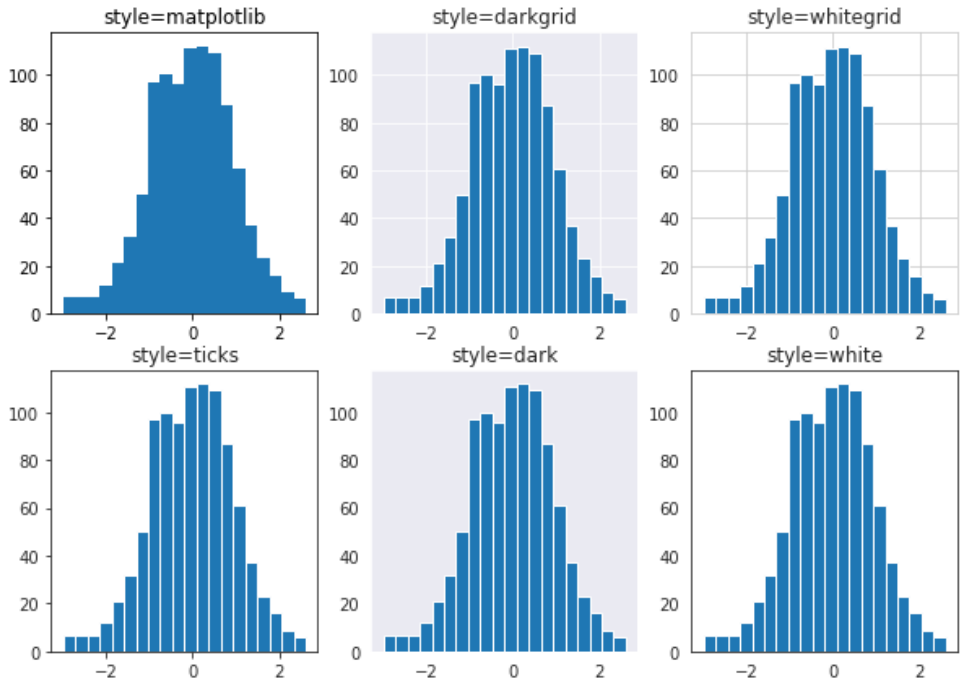
seaborn 设置风格的方法主要有三种：

- set，通用设置接口

- set_style，风格专用设置接口，设置后全局风格随之改变
- axes_style，设置当前图（axes 级）的风格，同时返回设置后的风格系列参数，支持 with 关键字用法

当前支持的风格主要有 5 种：

- darkgrid，默认风格
- whitegrid
- dark
- white
- ticks



seaborn 5 种内置风格与 matplotlib 绘图风格对比

相比 matplotlib 绘图风格，seaborn 绘制的直方图会自动增加空白间隔，图像更为清爽。而不同 seaborn 风格间，则主要是绘图背景色的差异。

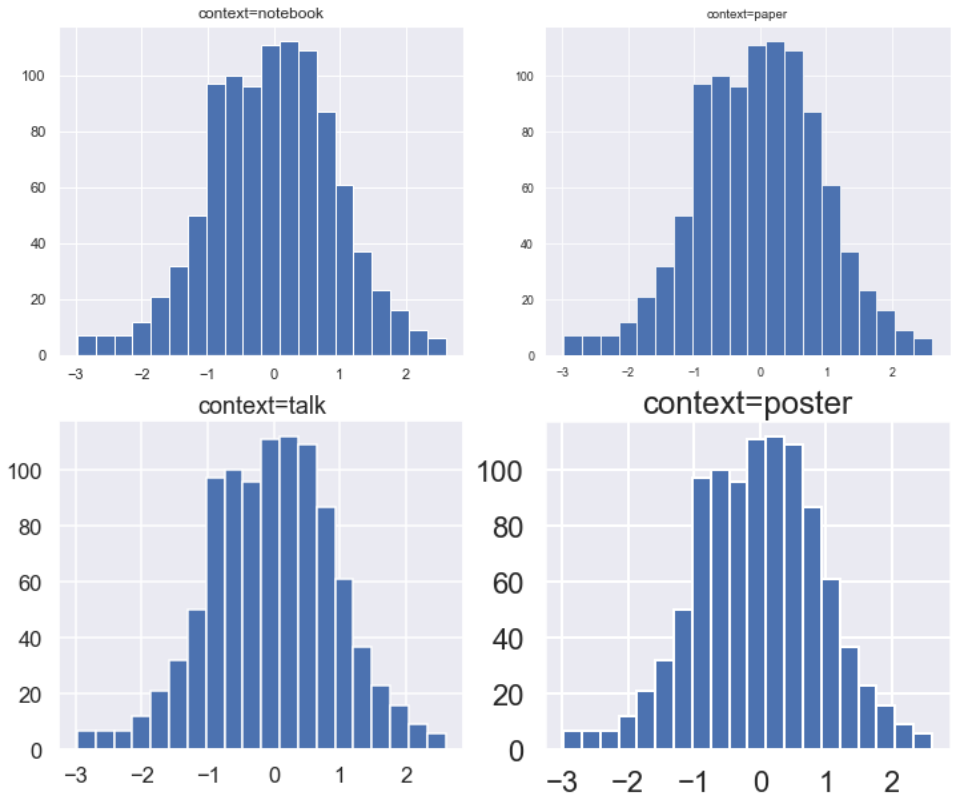
2. 环境设置

设置环境的方法也有 3 种：

- set，通用设置接口
- set_context，环境设置专用接口，设置后全局绘图环境随之改变
- plotting_context，设置当前图（axes 级）的绘图环境，同时返回设置后的环境系列参数，支持 with 关键字用法

当前支持的绘图环境主要有 4 种：

- notebook，默认环境
- paper
- talk
- poster



seaborn 4 种绘图环境对比

可以看出，4 种默认绘图环境最直观的区别在于字体大小的不同，而其他方面也均略有差异。详细对比下 4 种绘图环境下的系列参数设置：

```
sns.plotting_context('notebook')
```

```
{'font.size': 12,
 'axes.labelsize': 12,
 'axes.titlesize': 12,
 'xtick.labelsize': 11,
 'ytick.labelsize': 11,
 'legend.fontsize': 11,
 'axes.linewidth': 1.25,
 'grid.linewidth': 1,
 'lines.linewidth': 1.5,
 'lines.markersize': 6,
 'patch.linewidth': 1,
 'xtick.major.width': 1.25,
 'ytick.major.width': 1.25,
 'xtick.minor.width': 1,
 'ytick.minor.width': 1,
 'xtick.major.size': 6,
 'ytick.major.size': 6,
 'xtick.minor.size': 4,
 'ytick.minor.size': 4}
```

```
sns.plotting_context('paper')
```

```
{'font.size': 9.600000000000001,
 'axes.labelsize': 9.600000000000001,
 'axes.titlesize': 9.600000000000001,
 'xtick.labelsize': 8.8,
 'ytick.labelsize': 8.8,
 'legend.fontsize': 8.8,
 'axes.linewidth': 1.0,
 'grid.linewidth': 0.8,
 'lines.linewidth': 1.200000000000002,
 'lines.markersize': 4.800000000000001,
 'patch.linewidth': 0.8,
 'xtick.major.width': 1.0,
 'ytick.major.width': 1.0,
 'xtick.minor.width': 0.8,
 'ytick.minor.width': 0.8,
 'xtick.major.size': 4.800000000000001,
 'ytick.major.size': 4.800000000000001,
 'xtick.minor.size': 3.2,
 'ytick.minor.size': 3.2}
```

```
sns.plotting_context('talk')
```

```
{'font.size': 18.0,
 'axes.labelsize': 18.0,
 'axes.titlesize': 18.0,
 'xtick.labelsize': 16.5,
 'ytick.labelsize': 16.5,
 'legend.fontsize': 16.5,
 'axes.linewidth': 1.875,
 'grid.linewidth': 1.5,
 'lines.linewidth': 2.25,
 'lines.markersize': 9.0,
 'patch.linewidth': 1.5,
 'xtick.major.width': 1.875,
 'ytick.major.width': 1.875,
 'xtick.minor.width': 1.5,
 'ytick.minor.width': 1.5,
 'xtick.major.size': 9.0,
 'ytick.major.size': 9.0,
 'xtick.minor.size': 6.0,
 'ytick.minor.size': 6.0}
```

```
sns.plotting_context('poster')
```

```
{'font.size': 24,
 'axes.labelsize': 24,
 'axes.titlesize': 24,
 'xtick.labelsize': 22,
 'ytick.labelsize': 22,
 'legend.fontsize': 22,
 'axes.linewidth': 2.5,
 'grid.linewidth': 2,
 'lines.linewidth': 3.0,
 'lines.markersize': 12,
 'patch.linewidth': 2,
 'xtick.major.width': 2.5,
 'ytick.major.width': 2.5,
 'xtick.minor.width': 2,
 'ytick.minor.width': 2,
 'xtick.major.size': 12,
 'ytick.major.size': 12,
 'xtick.minor.size': 8,
 'ytick.minor.size': 8}
```

03 颜色设置

seaborn 风格多变的另一大特色就是支持个性化的颜色配置。颜色配置的方法有多种，常用方法包括以下两个：

- color_palette，基于 RGB 原理设置颜色的接口，可接收一个调色板对象作为参数，同时可以设置颜色数量
- hls_palette，基于 Hue(色相)、Luminance(亮度)、Saturation(饱和度)原理设置颜色的接口，除了颜色数量参数外，另外 3 个重要参数即是 hls

同时，为了便于查看调色板样式，seaborn 还提供了一个专门绘制颜色结果的方

法 palplot。

```
sns.palplot(sns.hls_palette(n_colors=8))
```

```
sns.palplot(sns.color_palette(n_colors=8))
```

hls_palette 提供了均匀过渡的 8 种颜色样例，而 color_palette 则只是提供了 8 种不同颜色

04 数据集

seaborn 自带了一些经典的数据集，用于基本的绘制图表示例数据。在联网状态下，可通过 load_dataset()接口进行获取，首次下载后后续即可通过缓存加载。返回数据集格式为 Pandas.DataFrame 对象。

当前内置了 10 几个数据集，常用的经典数据集如下：

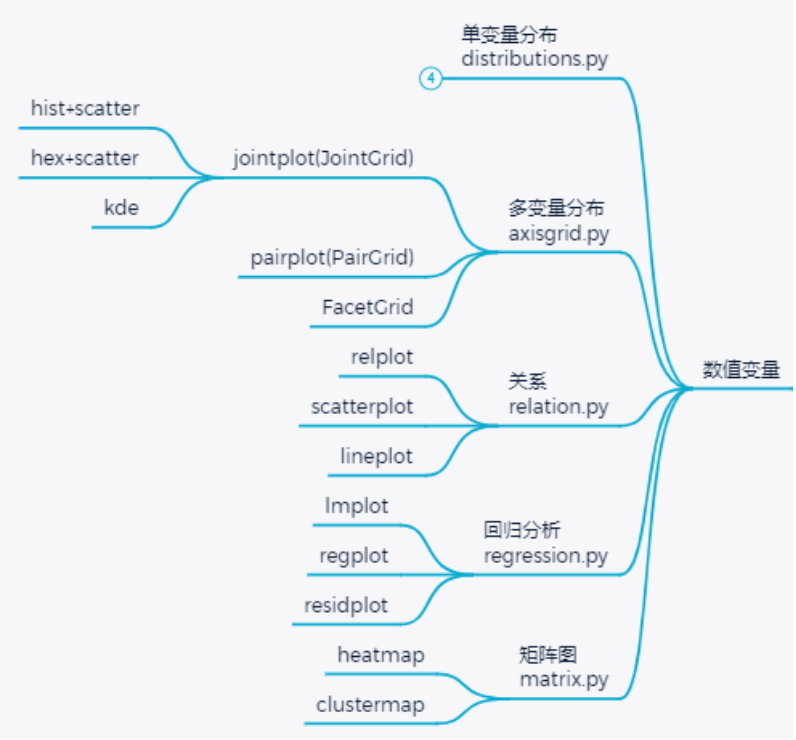
- iris：鸢尾花，与 sklearn 中数据集一致，仅有数值型数据
- tips：小费，主要是中餐和晚餐的数据集，既有分类数据也有数值数据，可用于探索数据间的回归关系
- titanic：泰坦尼克，经典数据集

本文后续所有绘图主要基于前 2 个数据集完成。

05 常用绘制图表

seaborn 内置了大量集成绘图接口，往往仅需一行代码即可实现美观的图表结果。按照数据类型，大体可分为连续性（数值变量）和离散型（分类数据）两类接口。

1. 数值变量



1.1 单变量分布

变量分布可用于表达一组数值的分布趋势，包括集中程度、离散程度等。seaborn 中提供了 3 种表达单变量分布的绘图接口

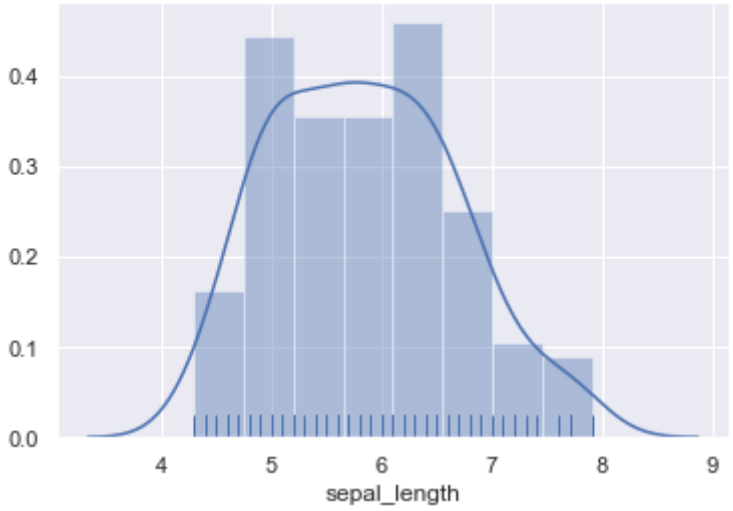
- distplot

distribution+plot，接口内置了直方图（histogram）、核密度估计图（kde，kernel density estimation）以及 rug 图（直译为地毯，绘图方式就是将数值出现的位置原原本本的以小柱状的方式添加在图表底部），3 种图表均可通过相应参数设置开关状态，默认情况下是绘制 hist+kde。

distplot 支持 3 种格式数据：pandas.series、numpy 中的 1darray 以及普通的 list 类型。以鸢尾花数据为例，并添加 rug 图可得如下图表：

```
sns.distplot(iris['sepal_length'], rug=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e2e097080>
```



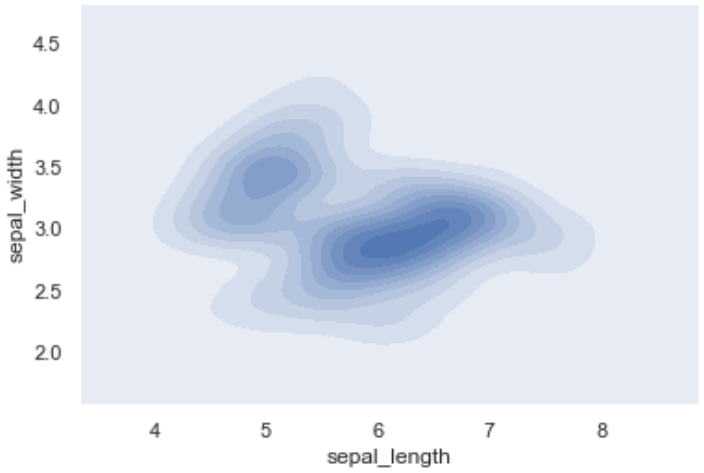
- kdeplot

kdeplot 是一个专门绘制核密度估计图的接口，虽然 distplot 中内置了 kdeplot 图表，并且可通过仅开启 kde 开关实现 kdeplot 的功能，但 kdeplot 实际上支持更为丰富的功能，比如当传入 2 个变量时绘制的即为热力图效果。

仍以鸢尾花为例，绘制双变量核密度估计图，并添加阴影得到如下图表：

```
sns.kdeplot(iris['sepal_length'], iris['sepal_width'], shade=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e2e0cde10>
```

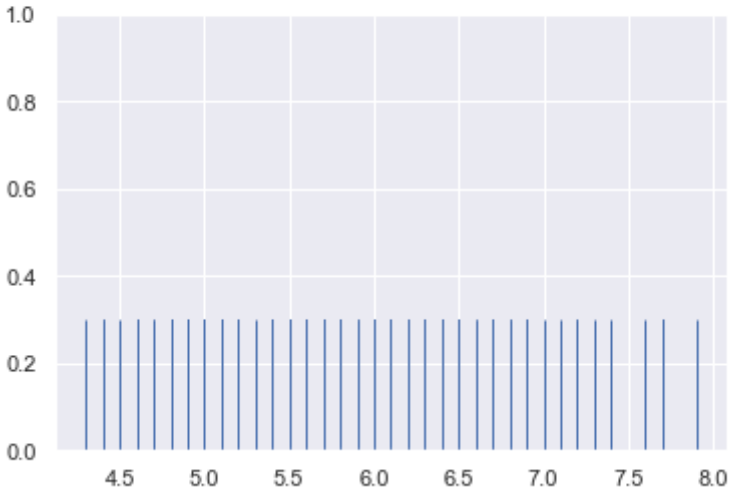


- rugplot

这是一个不太常用的图表类型，其绘图方式比较朴素：即原原本本的将变量出现的位置绘制在相应坐标轴上，同时忽略出现次数的影响。

```
sns.rugplot(iris['sepal_length'], height=0.3)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e2da918d0>
```



1.2 多变量分布

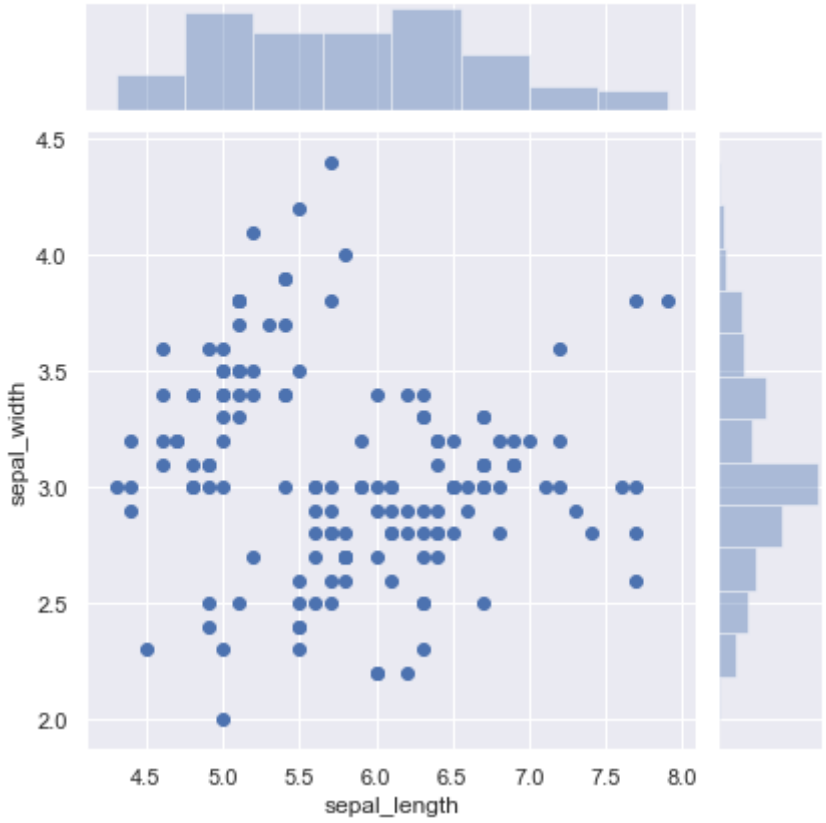
单变量分布仅可用于观察单一维度的变化关系，为了探究多变量间分布关系时，如下绘图接口更为有效：

- jointplot

joint 意为联合，顾名思义 jointplot 是一个双变量分布图表接口。绘图结果主要有三部分：绘图主体用于表达两个变量对应的散点图分布，在其上侧和右侧分别体现 2 个变量的直方图分布：

```
sns.jointplot(x='sepal_length', y='sepal_width', data=iris)
```

```
<seaborn.axisgrid.JointGrid at 0x29e2e0e4208>
```

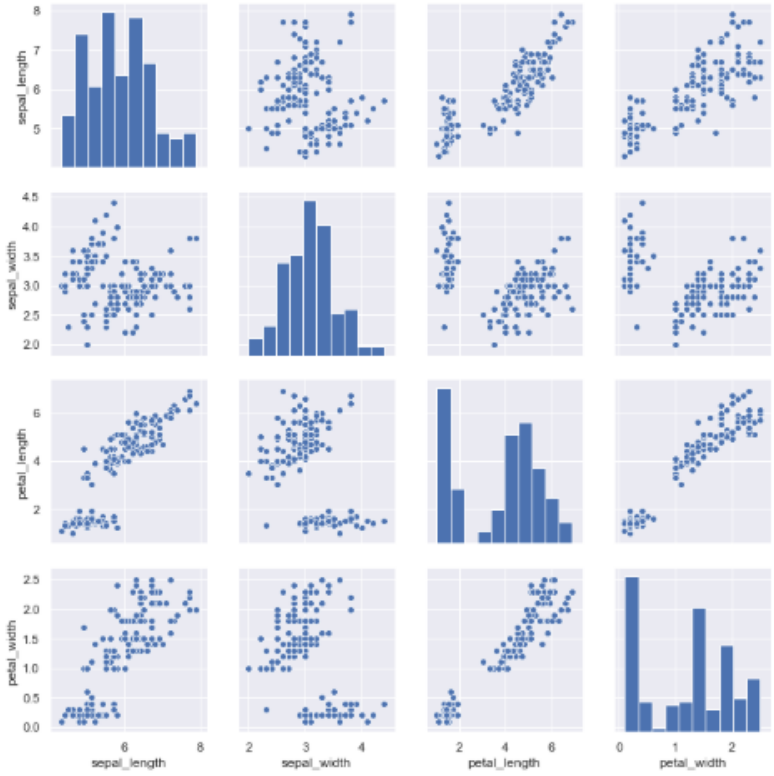


- pairplot

当变量数不止 2 个时，pairplot 是查看各变量间分布关系的首选。它将变量的任意两两组合分布绘制成一个子图，对角线用直方图、而其余子图用相应变量分别作为 x、y 轴绘制散点图。显然，绘制结果中的上三角和下三角部分的子图是镜像的。

```
sns.pairplot(data=iris)
```

```
<seaborn.axisgrid.PairGrid at 0x29e2d940f60>
```

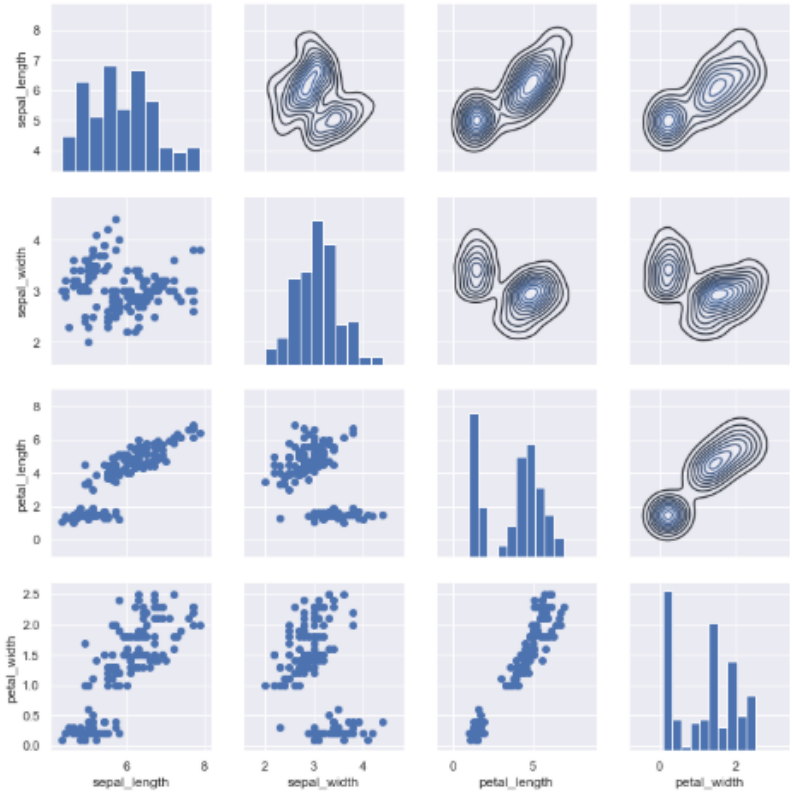


实际上，查看 seaborn 源码可以发现，其绘图接口大多依赖于一个类实现。例如：jointplot 在 seaborn 中实际上先实现了一个名为 JointGrid 的类，然后在调用 jointplot 时即是调用该类实现。相比之下，JointGrid 可以实现更为丰富的可定制绘图接口，而 jointplot 则是其一个简单的样例实现。类似地，pairplot 则是依赖于 PairGrid 类实现。

例如，如下案例调用了 PairGrid 类实现，与标准 pairplot 不同的是上三角子图选用了 kde 图表，效果更为丰富。

```
pg = sns.PairGrid(iris)
pg.map_diag(plt.hist)
pg.map_lower(plt.scatter)
pg.map_upper(sns.kdeplot)
```

```
<seaborn.axisgrid.PairGrid at 0x29e2f9fab70>
```



与此同时，seaborn 中的绘图接口虽然大多依赖于相应的类实现，但却并未开放所有的类接口。实际上，**可供用户调用的类只有 3 个**，除了前面提到的 **JointGrid** 和 **PairGrid** 外，还有一个是 **FacetGrid**，它是一个 seaborn 中很多其他绘图接口的基类。

1.3 关系型图表

seaborn 还提供了几个用于**表达双变量关系**的图表，主要包括点图和线图两类。主要提供了 3 个接口，relplot(relation+plot)、scatterplot 和 lineplot，其中 relplot 为 figure-level（可简单理解为操作对象是 matplotlib 中 figure），而后两者是 axes-level（对应操作对象是 matplotlib 中的 axes），但实际上接口调用方式和传参模式都是一致的，其核心参数主要包括以下 4 个：

- data，pandas.dataframe 对象，后面的 x、y 和 hue 均为源于 data 中的某一列值
- x，绘图的 x 轴变量
- y，绘图的 y 轴变量
- hue，区分维度，一般为分类型变量

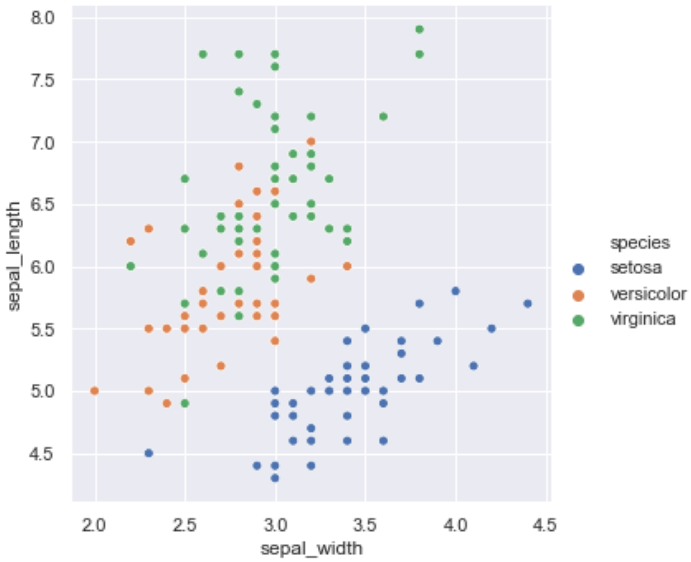
同时，relplot 可通过 kind 参数选择绘制图表是 scatter 还是 line 类型。默认为 scatter 类型。

- relplot

仍以鸢尾花数据集为例，绘制不同种类花的两变量散点图如下：

```
sns.relplot(x='sepal_width', y='sepal_length', hue='species', data=iris)
```

```
<seaborn.axisgrid.FacetGrid at 0x29e30f1e198>
```

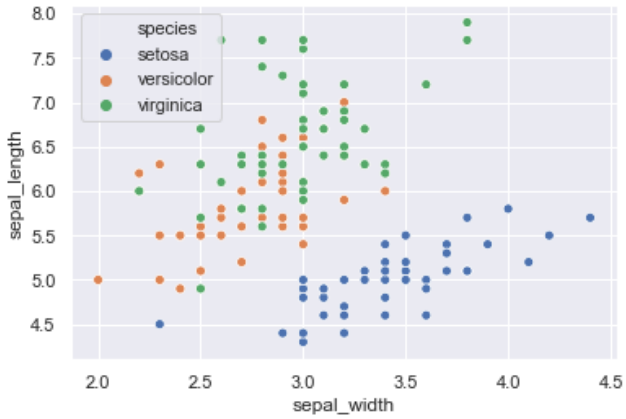


- scatterplot

也可实现同样的散点图效果:

```
sns.scatterplot(x='sepal_width', y='sepal_length', hue='species', data=iris)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e311b1080>
```

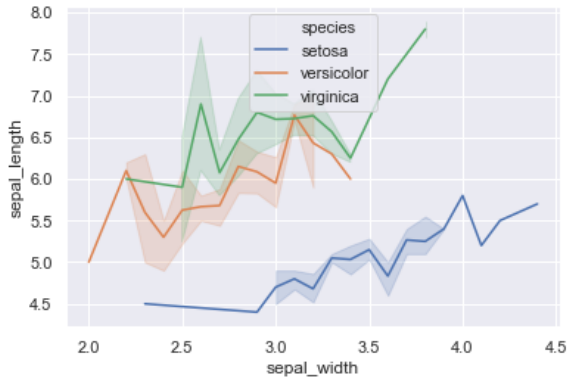


- lineplot

lineplot 不同于 matplotlib 中的折线图，会将同一 x 轴下的多个 y 轴的统计量（默认为均值）作为折线图中的点的位置，并辅以阴影表达其置信区间。可用于快速观察点的分布趋势。

```
sns.lineplot(x='sepal_width', y='sepal_length', data=iris, hue='species')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e31210dd8>
```



1.4 回归分析

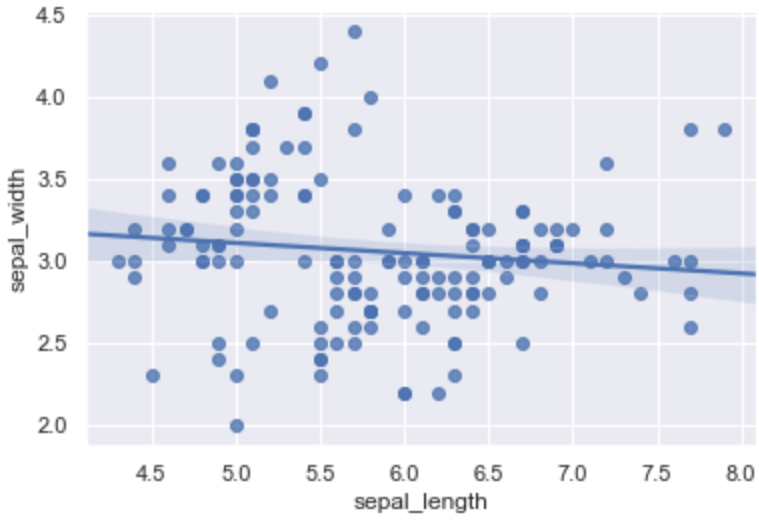
在查看双变量分布关系的基础上，seaborn 还提供了简单的回归接口。另外，还可设置回归模型的阶数，例如设置 order=2 时可以拟合出抛物线型回归线。

- regplot

基础回归模型接口，即 regression+plot。绘图结果为散点图+回归直线即置信区间。另外，还可通过 logistic 参数设置是否启用逻辑回归。

```
sns.regplot(x='sepal_length', y='sepal_width', data=iris)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e312c5cf8>
```

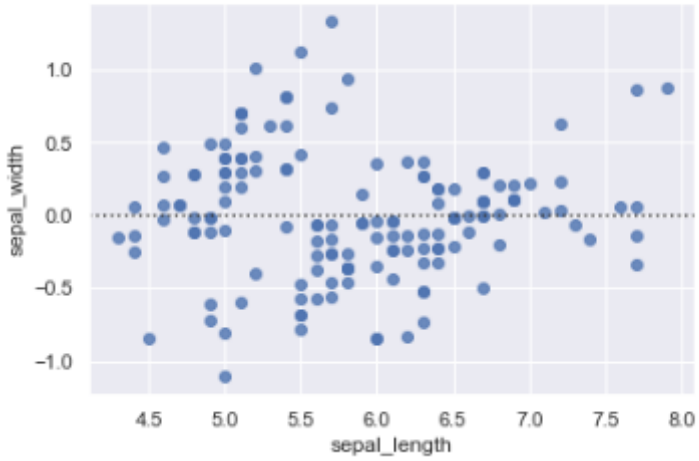


- residplot

residplot 提供了拟合后的残差分布图，相当于先执行 lmplot 中的回归拟合，而后将回归值与真实值相减结果作为绘图数据。直观来看，当残差结果随机分布于 y=0 上下较小的区间时，说明具有较好的回归效果。

```
sns.residplot(x='sepal_length', y='sepal_width', data=iris)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e386e2898>
```

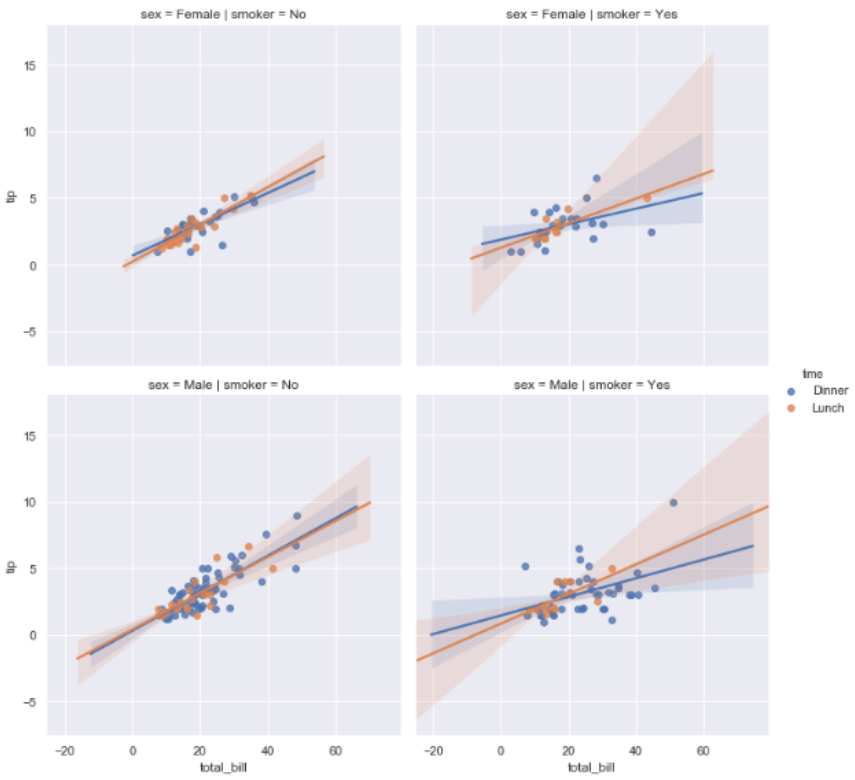


- lmplot

lmplot=regplot+FacetGrid，也是用于绘制回归图表，但功能相比更为强大，除了增加 hue 参数支持分类回归外，还可添加 row 和 col 参数（二者均为 FacetGrid 中的常规参数，用于添加多子图的行和列）实现更多的分类回归关系。这里以 seaborn 中的小费数据集进行绘制，得到如下回归图表：

```
sns.lmplot(x='total_bill', y='tip', row='sex', col='smoker', hue='time', data=tips)
```

```
<seaborn.axisgrid.FacetGrid at 0x29e328973c8>
```



1.5 矩阵图

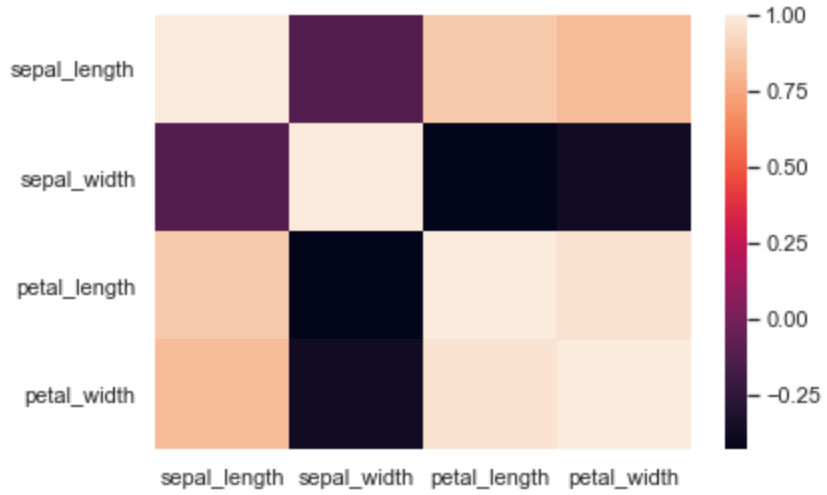
矩阵图主要用于表达一组数值型数据的大小关系，在探索数据相关性时也较为实用。

- heatmap

原原本本的将一组数据以热力图矩阵的形式展现出来，同时可通过设置数值上下限和颜色板实现更为美观的效果。如下图表展示了鸢尾花数据集中各变量间的相关系数，从中可以很容易看出 sepal_length、petal_length、petal_width 三者之间彼此呈现较强的相关性，而 sepal_width 则与它们相关性不大。

```
sns.heatmap(iris.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e32c76048>
```

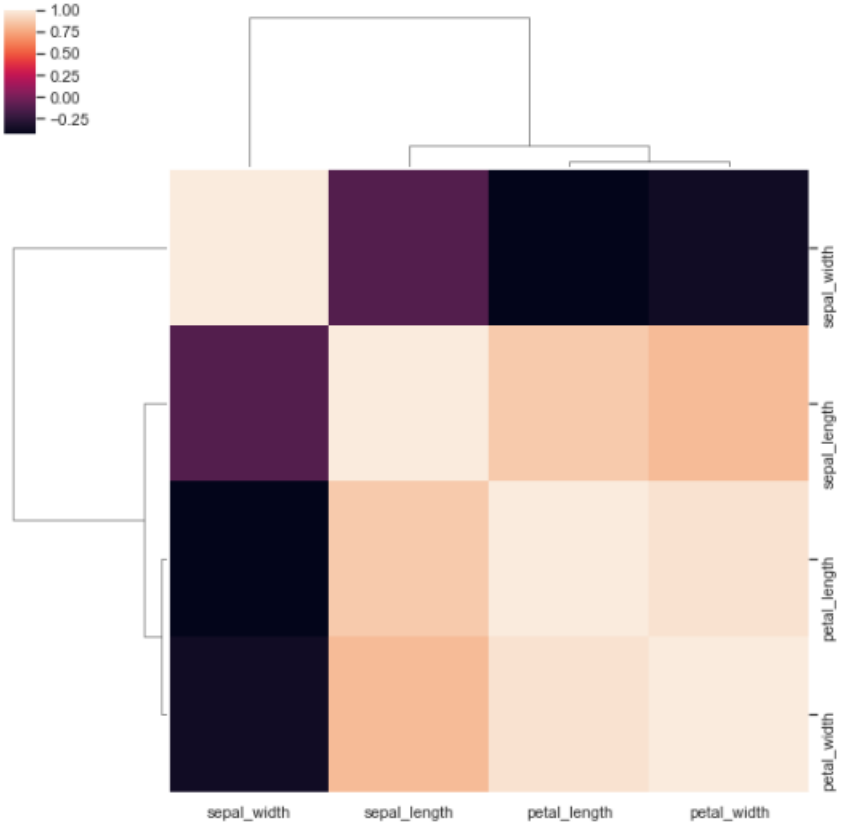


- clustermap

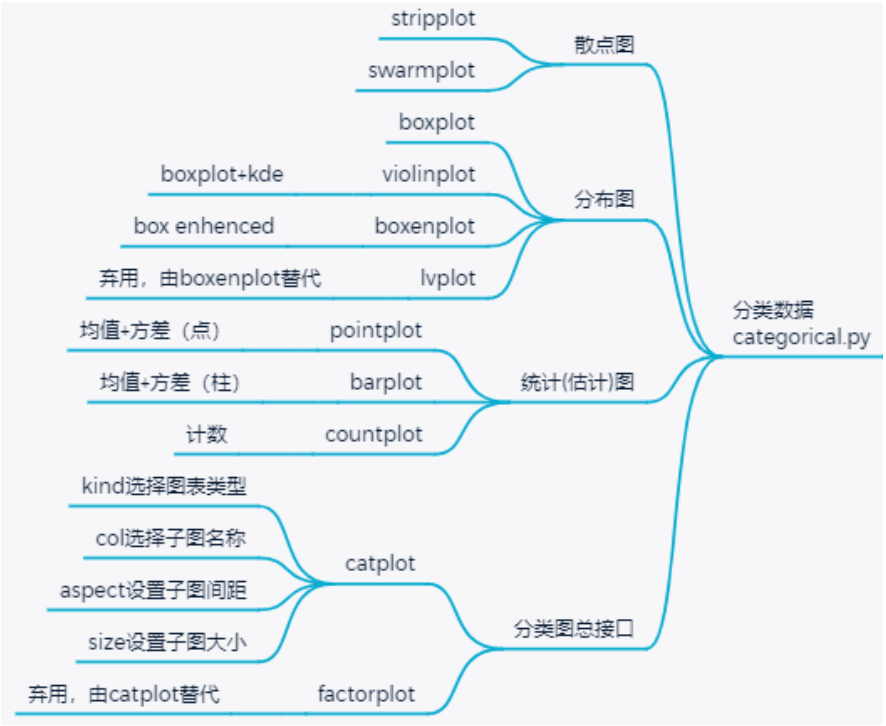
在 heatmap 的基础上，clustermap 进一步挖掘各行数据间的相关性，并逐一按最小合并的原则进行聚类，给出了聚类后的热力图：

```
sns.clustermap(iris.corr())
```

```
<seaborn.matrix.ClusterGrid at 0x29e32ca1f98>
```



2.分类数据



2.1 散点图

分类数据散点图接口主要用于当一列数据是分类变量时。相比于两列数据均为

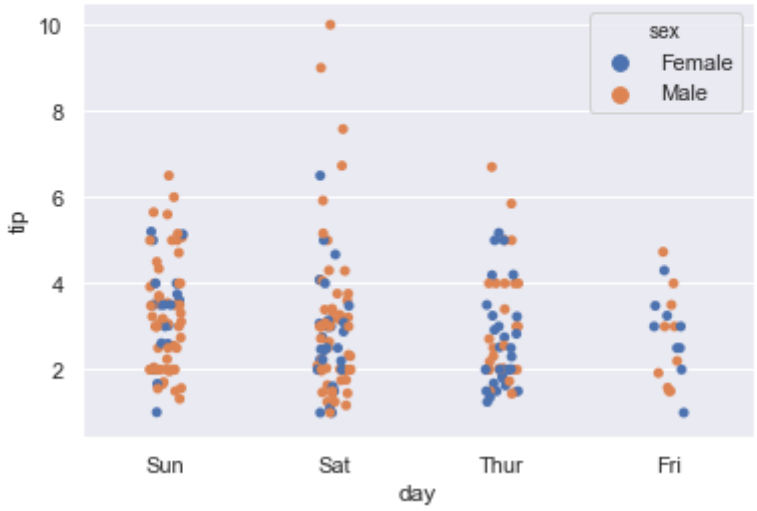
数值型数据，可以想象分类数据的散点图将会是多条竖直的散点线。绘图接口有 stripplot 和 swarmplot 两种，常用参数是一致的，主要包括：

- x，散点图的 x 轴数据，一般为分类型数据
- y，散点图的 y 轴数据，一般为数值型数据
- hue，区分维度，相当于增加了第三个参数
- data，pandas.dataframe 对象，以上几个参数一般为 data 中的某一列
- stripplot

常规的散点图接口，可通过 jitter 参数开启散点左右"抖动"效果（实际即为在水平方向上加了一个随机数控制 x 坐标，默认 jitter=True；当设置 jitter 为 False 时，散点图均严格位于一条直线上）

```
sns.stripplot(x='day', y='tip', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e32de60f0>
```

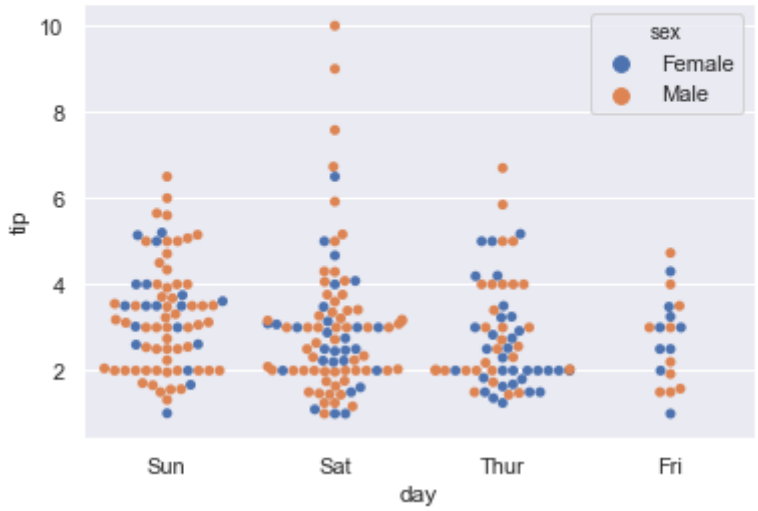


- swarmplot

在 stripplot 的基础上，不仅将散点图通过抖动来实现相对分离，而且会严格讲各散点一字排开，从而便于直观观察散点的分布聚集情况：

```
sns.swarmplot(x='day', y='tip', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e334ea748>
```



2.2 分布图

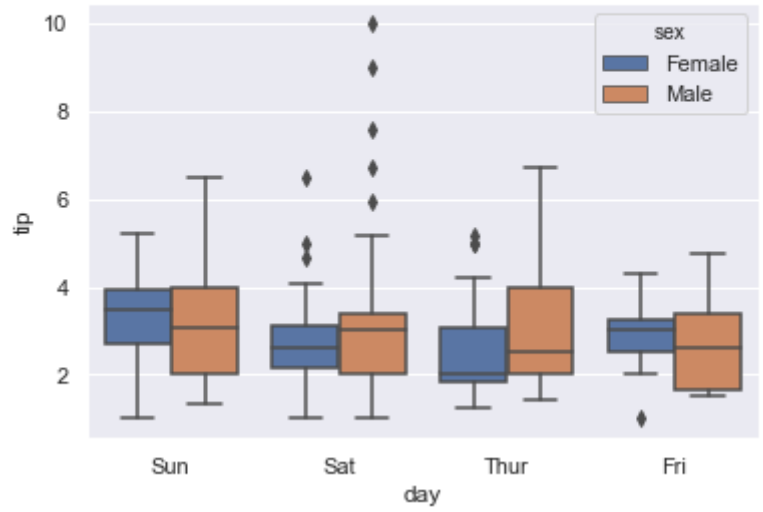
与数值型变量分布类似，seaborn 也提供了几个分类型数据常用的分布绘图接口。且主要参数与前述的散点图接口参数是十分相近的。

- boxplot

箱线图，也叫盒须图，表达了各分类下数据 4 分位数和离群点信息，常用于查看数据异常值等。

```
sns.boxplot(x='day', y='tip', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e335520f0>
```



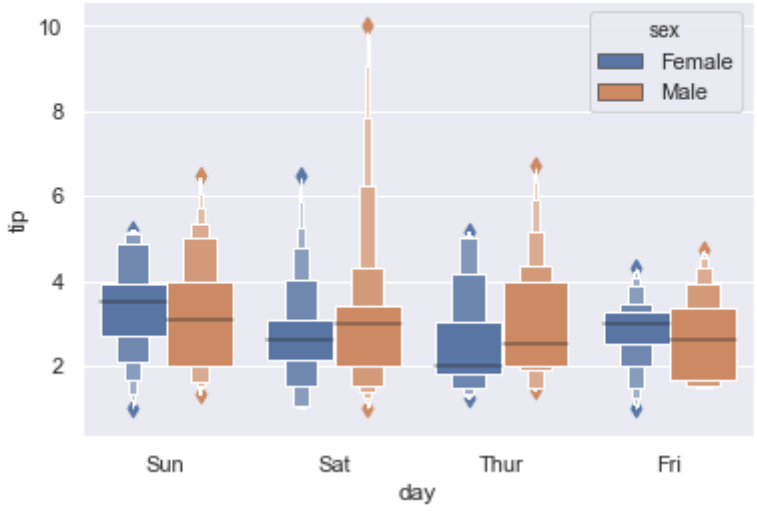
从各日期的小费箱线图中可以看出，周六这一天小费数值更为离散，且男性的小费数值随机性更强；而其他三天的小费数据相对更为稳定。

- boxenplot

是一个增强版的箱线图，即 box+enhenced+plot，在标准箱线图的基础上增加了更多的分位数信息，绘图效果更为美观，信息量更大。

```
sns.boxenplot(x='day', y='tip', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e336494e0>
```



- lvplot

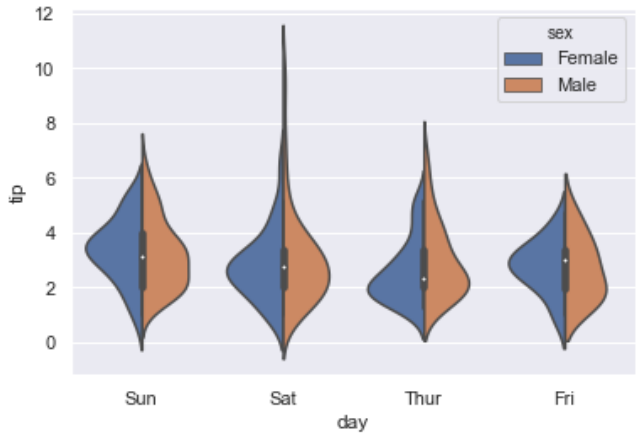
lvplot=letter value+plot，是 boxenplot 的前身，绘图效果与后者一致。现已被 boxenplot 所替代，不再提倡使用。

- violinplot

小提琴图，相当于 boxplot+kdeplot，即在标准箱线图的基础上增加了 kde 图的信息，从而可更为直观的查看数据分布情况。因其绘图结果常常酷似小提琴形状，因而得名 violinplot。在 hue 分类仅有 2 个取值时，还可通过设置 split 参数实现左右数据合并显示。

```
sns.violinplot(x='day', y='tip', hue='sex', data=tips, split=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e33895550>
```



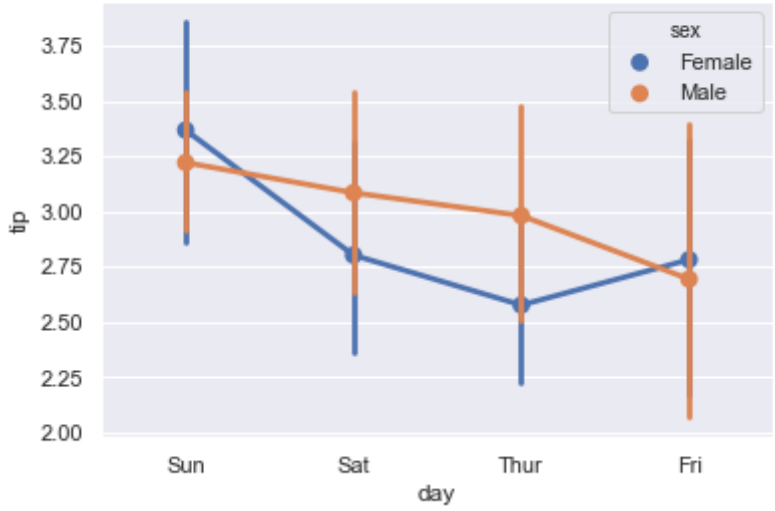
2.3 统计(估计)图

- pointplot

pointplot 给出了数据的统计量（默认统计量为均值）和相应置信区间（confidence intervals，默认值为 95%，即参数 ci=95），并以相应的点和线进行绘图显示：

```
sns.pointplot(x='day', y='tip', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e324ff978>
```

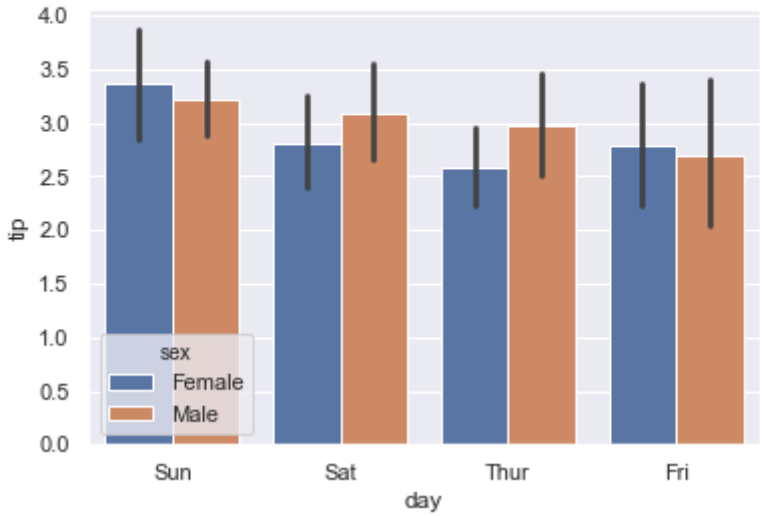


- barplot

与 pointplot 用折线表达统计量变化不同，barplot 以柱状图表达统计量，而置信区间则与前者一致，仅仅是适用场景不同而已。

```
sns.barplot(x='day', y='tip', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e3396cda0>
```



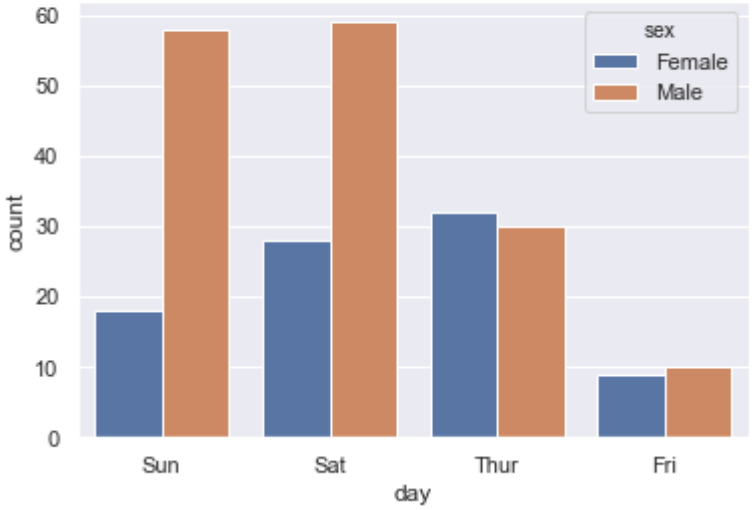
注：当 x 轴分类变量为连续日期数据时，选用 pointplot 得到的绘图意义更为明确；而对于其他分类型变量，则选用 barplot 更为合适。

- countplot

这是一个功能比较简单的统计图表，仅用于表达各分类值计数，并以柱状图的形式展现：

```
sns.countplot(x='day', hue='sex', data=tips)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x29e33a01208>
```



2.4 figure-level 分类绘图总接口

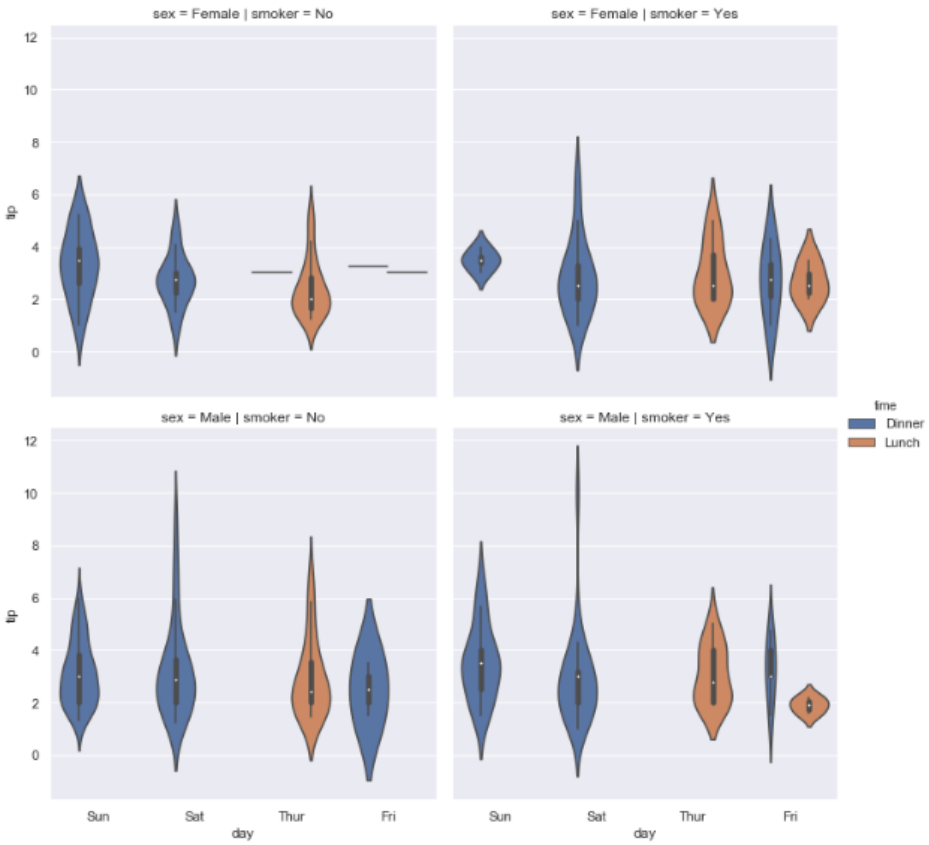
最后，seaborn 还提供了一个用于分类数据绘图的 figure-level 接口 catplot，catplot 与其他分类数据绘图接口的关系相当于 lmplot 与 regplot 的关系；同时 catplot 中还可通过 kind 参数实现前面除 countplot 外的所有绘图接口，功能更为强大。kind 默认为 strip，此时等效于 stripplot 接口。

- catplot

catplot=category+plot，用其实现分类条件下小提琴图。

```
sns.catplot(x='day', y='tip',row='sex', col='smoker', hue='time', data=tips, kind='violin')
```

```
<seaborn.axisgrid.FacetGrid at 0x29e356507b8>
```



- factorplot

factorplot 是 catplot 的前身，二者实现功能完全一致，现已被后者更名替代，官方不再推荐使用。

另外，seaborn 中还提供了一个时序数据绘图接口 tsplot，个人用的较少。

06 小结

最后简要总结 seaborn 制作可视化图表的几个要点：

- 绝大多数绘图接口名字均为 XXXXplot 形式
- 绘图数据对象主要区分连续型的数值变量和离散型的分类数据
- 绘图接口中的传参类型以 pandas.dataframe 为主，当提供了 dataframe 对象作为 data 参数后，x、y 以及 hue 即可用相应的列名作为参数，但也支持 numpy 的数组类型和 list 类型
- 绘图接口底层大多依赖一个相应的类来实现，但对外开放的只有 3 个类：PairGrid、JointGrid 和 FacetGrid
- 接口包括了常用的分布、关系、统计、回归类图表
- 可灵活设置绘图风格、环境和颜色

五、SKLearn 库主要模块功能简介

导读：曾经，当我初次接触数据分析三剑客（numpy、pandas、matplotlib）时，感觉每个库的功能都很多很杂，所以在差不多理清了各模块功能后便相继推出了各自教程（文末附链接）；后来，当接触了机器学习库 sklearn 之后，才发现三剑客也不外如是，相比 sklearn 简直是小巫见大巫；再后来，又开始了 pyspark 的学习之旅，发现无论是模块体积还是功能细分，pyspark 又都完爆 sklearn；最近，逐渐入坑深度学习 PyTorch 框架，终于意识到 python 数据科学库没有最大，只有更大……

鉴于机器学习本身理论性很强，加之 sklearn 库功能强大 API 众多，自然不是总结一份教程所能涵盖的。所以这一次，仅对其中的各子模块进行梳理和介绍，以期通过本文能对 sklearn 迅速建立宏观框架。

01 sklearn 简介

sklearn，全称 scikit-learn，是 python 中的机器学习库，建立在 numpy、scipy、matplotlib 等数据科学包的基础之上，涵盖了机器学习中的样例数据、数据预处理、模型验证、特征选择、分类、回归、聚类、降维等几乎所有环节，功能十分强大，目前 sklearn 版本是 0.23。与深度学习库存在 pytorch、TensorFlow 等多种框架可选不同，sklearn 是 python 中传统机器学习的首选库，不存在其他竞争者。

本文将分别围绕下图中各大子模块进行分别介绍，不会面面俱到、但求提纲挈领。

02 样例数据集

sklearn 为初学者提供了一些经典数据集，通过这些数据集可快速搭建机器学习任务、对比模型性能。数据集主要围绕分类和回归两类经典任务，对于不同需求，常用数据集简介如下：

- load_breast_cancer：乳腺癌数据集，特征为连续数值变量，标签为 0 或 1 的二分类任务
- load_iris：经典鸢尾花数据集，特征为连续数值变量，标签为 0/1/2 的三分类任务，且各类样本数量均衡，均为 50 个
- load_wine：红酒数据集，与鸢尾花数据集特点类似，也是用于连续特征的 3 分类任务，不同之处在于各类样本数量轻微不均衡

- load_digits：小型手写数字数据集（之所以称为小型，是因为还有大型的手写数字数据集 mnist），包含 0-9 共 10 种标签，各类样本均衡，与前面 3 个数据集最大不同在于特征也是离散数值 0—16 之间，例如在进行多项式朴素贝叶斯模型、ID3 树模型时，可用该数据集
- load_boston：波士顿房价数据集，连续特征拟合房价，适用于回归任务

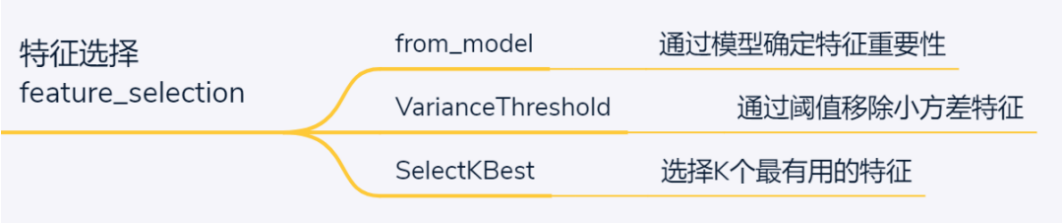
值得指出，sklearn 除了 load 系列经典数据集外，还支持自定义数据集 make 系列和下载数据集 fetch 系列（load 系列为安装 sklearn 库时自带，而 fetch 则需额外下载），这为更多的学习任务场景提供了便利。

03 数据预处理

sklearn 中的各模型均有规范的数据输入输出格式，一般以 np.array 和 pd.dataframe 为标准格式，所以一些字符串的离散标签是不能直接用于模型训练的；同时为了加快模型训练速度和保证训练精度，往往还需对数据进行预处理，例如在以距离作为度量进行训练时则必须考虑去量纲化的问题。为此，sklearn 提供了一些常用的数据预处理功能，常用的包括：

- MinMaxScaler：归一化去量纲处理，适用于数据有明显的上下限，不会存在严重的异常值，例如考试得分 0-100 之间的数据可首选归一化处理
- StandardScaler：标准化去量纲处理，适用于可能存在极大或极小的异常值，此时用 MinMaxScaler 时，可能因单个异常点而将其他数值变换的过于集中，而用标准正态分布去量纲则可有效避免这一问题
- Binarizer：二值化处理，适用于将连续变量离散化
- OneHotEncoder：独热编码，一种经典的编码方式，适用于离散标签间不存在明确的大小相对关系时。例如对于民族特征进行编码时，若将其编码为 0-55 的数值，则对于以距离作为度量的模型则意味着民族之间存在"大小"和"远近"关系，而用独热编码则将每个民族转换为一个由 1 个"1"和 55 个"0"组成的向量。弊端就是当分类标签过多时，容易带来维度灾难，而特征又过于稀疏
- Ordinary：数值编码，适用于某些标签编码为数值后不影响模型理解和训练时。例如，当民族为待分类标签时，则可将其简单编码为 0-55 之间的数字

04 特征选择



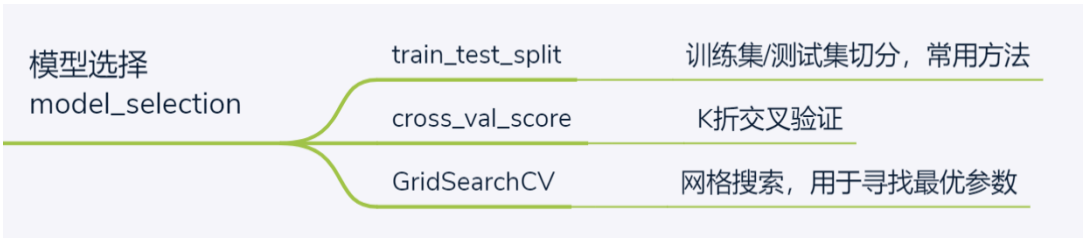
机器学习中有句经典的台词是：数据和特征决定学习上限，模型和算法只是逼近这个上限，可见特征工程在机器学习中的重要性。一般而言，传统机器学习中的特征工程主要包括两方面需求：

- 特征维度过多时，为加快模型训练速度，进行特征选择即过滤掉不重要的特征；
- 特征较少或模型训练性能不好时，可通过对问题的理解尝试构建特征提升维度。

这里简单介绍几种特征选择的方式：

- from_model：顾名思义，从模型选择特征，这是因为很多模型在训练后都提供了特征的重要性结果 feature_importance，据此可作为特征选择的依据
- VarianceThreshold：根据方差阈值做特征选择，实际上当某一特征的方差越大时意味着该特征越能带来更好的分类区分度，否则由于特征取值比较集中，很难对应不同的分类效果
- SelectKBest：指定 K 个特征选择结果，具体也需依赖选择的标准

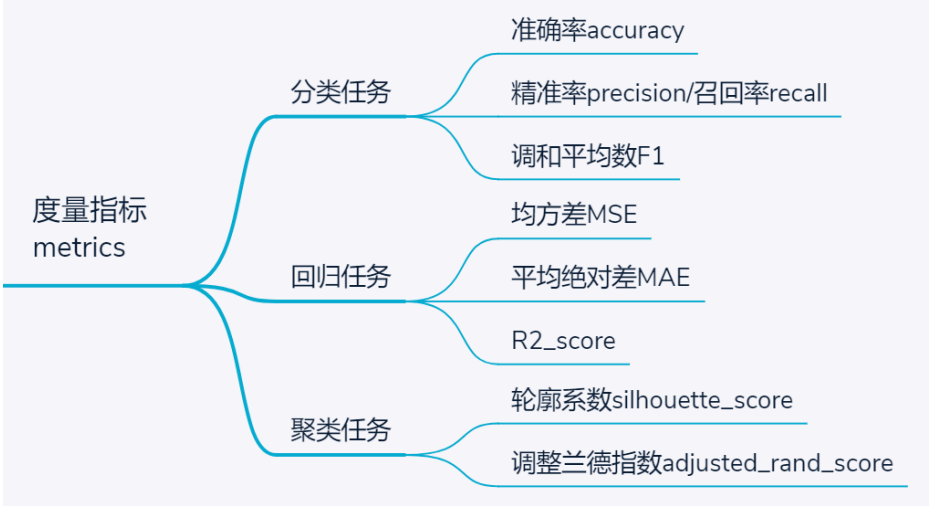
05 模型选择



模型选择是机器学习中的重要环节，涉及到的操作包括数据集切分、参数调整和验证等。对应常用函数包括：

- train_test_split：常用操作之一，切分数据集和测试集，可设置切分比例
- cross_val_score：交叉验证，默认 K=5 折，相当于把数据集平均切分为 5 份，并逐一选择其中一份作为测试集、其余作为训练集进行训练及评分，最后返回 K 个评分
- GridSearchCV：调参常用方法，通过字典类型设置一组候选参数，并制定度量标准，最后返回评分最高的参数

06 度量指标



不同的度量指标可以学到不同的最优模型。对于不同类型任务，sklearn 提供了多种度量指标，包括：

- 分类任务：准确率，所有样本中分类正确样本所占比例；精准率和召回率，一对相互矛盾的指标，适用于分类样本数量不均衡时，此时为了保证既定目标，可只选其中一个指标；调和平均数 F1，相当于兼顾了精准率和召回率两项指标
- 回归任务：常用的包括 MSE、MAE，但 R2_score 实质上是更为公允和直观的评价指标，其意义是 R2_Score = MSE / VAR，即预测分类和实际分类的均方差与实际分类方差的比值
- 聚类任务：聚类任务属于无监督学习，所以根据是否有先验标签信息，聚类结果的度量指标包括轮廓系数（无需先验标签，用组内距离与组外最近距离的比值度量）、调整兰德指数（基于真实分簇标签和聚类标签计算）

07 降维

降维也属于无监督学习的一种，当特征维度过多时可通过矩阵的 QR 分解实现在尽可能保留原有信息的情况下降低维度，一般用于图像数据预处理，且降维后的特征与原特征没有直接联系，使得模型训练不再具有可解释性。

08 聚类

聚类是一种典型的无监督学习任务，但也是实际应用中较为常见的需求。在不提供样本真实标签的情况下，基于某些特征对样本进行物以类聚。根据聚类的原理，主要包括三种：

- 基于距离聚类，典型的就是 K 均值聚类，通过不断迭代和重新寻找最小距离，对所有样本划分为 K 个簇，有一款小游戏《拥挤城市》应该就是基于 K 均值聚类实现

• 基于密度聚类，与距离聚类不同，基于密度聚类的思想是源于通过距离判断样本是否连通（需指定连通距离的阈值），从而完成样本划分。由于划分结果仅取决于连通距离的阈值，所以不可指定聚类的簇数。典型算法模型是 DBSCAN

• 基于层次聚类，具体又可细分为自顶向下和自底向上，以自底向上层次聚类为例：首先将所有样本划分为一类，此时聚类簇数 K=样本个数 N，遍历寻找 K 个簇间最相近的两个簇并完成合并，此时还有 K-1 个簇，如此循环直至划分为指定的聚类簇数。当然，这里评价最相近的两个簇的标准又可细分为最小距离、最大距离和平均距离。

09 基本学习模型

分类和回归任务是机器学习中的经典场景，同属于有监督学习。经典的学习算法主要包括 5 种：

- 线性模型，回归任务中对应线性回归，分类任务则对应即逻辑回归，或者叫对数几率回归，实质是通过线性回归拟合对数几率的方式来实现二分类
- K 近邻，最简单易懂的机器学习模型，无需训练（惰性模型），仅仅是通过判断自己所处位置周边的样本判断类比或者拟合结果
- 支持向量机，一个经典的机器学习模型，最初也是源于线性分类，通过最大化间隔实现最可靠的分类边界。业界相传：支持向量机有三宝、间隔对偶核函数。其中"间隔"由硬间隔升级为软间隔解决了带异常值的线性不可分场景，"对偶"是在优化过程中求解拉格朗日问题的一个小技巧，而核函数才是支持向量机的核心，通过核实的核函数可以实现由线性可分向线性不可分的升级、同时避免了维度灾难
- 朴素贝叶斯，源于概率论中贝叶斯全概率公式，模型训练的过程就是拟合各特征分布概率的过程，而预测的过程则是标出具有最大概率的类比，是一个纯粹的依据概率完成分类任务的模型。而像逻辑回归、K 近邻、支持向量机以及决策树，虽然也都可以预测出各类别概率，但并不是纯粹意义上的概率
- 决策树，这是一个直观而又强大的机器学习模型，训练过程主要包括特征选择-切分-剪枝，典型的 3 个决策树是 ID3、C4.5 和 CART，其中 CART 树既可用于分类也可用于回归。更重要的是，决策树不仅模型自身颇具研究价值，还是众多集成学习模型的基学习器。

在以上 5 个经典的基本学习模型中，除了朴素贝叶斯仅用于分类任务外，其他 4 个模型都是既可分类也可回归的模型。

10 集成学习模型

当基本学习模型性能难以满足需求时，集成学习便应运而生。集成学习，顾名思义，就是将多个基学习器的结果集成起来汇聚出最终结果。而根据汇聚的过程，集成学习主要包括 3 种流派：

- bagging，即 bootstrap aggregating，通过自助取样（有放回取样）实现并行训练多个差异化的基学习器，虽然每个学习器效果可能并不突出，但通过最后投票得到的最终结果性能却会稳步提升。当基学习器采取决策树时，bagging 思想的集成学习

模型就是随机森林。另外，与 bagging 对应的另一种方式是无放回取样，相应的方法叫 pasting，不过应用较少

- boosting，即提升法。与 bagging 模型并行独立训练多个基学习器不同，boosting 的思想是基于前面训练结果逐渐训练更好的模型，属于串行的模式。根据实现细节不同，又具体分为两种 boosting 模型，分别是 Adaboost 和 GBDT，二者的核心思想差异在于前者的提升聚焦于之前分错的样本、而后者的提升聚焦于之前漏学的残差。另外一个大热的 XGBoost 是对 GBDT 的一个改进，实质思想是一致的。
- stacking，即堆栈法，基本流程与 bagging 类似而又不同：stacking 也是并行独立训练多个基学习器，而后又将这些训练的结果作为特征进行再次学习。有些类似于深度学习中的多层神经网络。

11 小节

以上，对 sklearn 中的常用子模块进行了粗略简介，基本涵盖了常用的模型和辅助函数，对于 sklearn 入门来说是足够的。当然，本文仅旨在建立对 sklearn 库的宏观框架，更为深入的学习当然还是要查阅专项教程。