使用第三方库

**实用的第三方软件包**

能够安装并导入第三方库很有用，但是要成为优秀的程序员，还需要知道有哪些库可以使用。大家通常通过在线推荐或同事介绍了解实用的新库。如果你是一名 Python 编程新手，可能没有很多同事，因此为了帮助你了解入门信息，下面是优达学城工程师很喜欢使用的软件包列表。（可能部分网站在国内网络中无法打开）

* [IPython](https://ipython.org/) - 更好的交互式 Python 解释器
* [requests](http://docs.python-requests.org/) - 提供易于使用的方法来发出网络请求。适用于访问网络 API。
* [Flask](http://flask.pocoo.org/) - 一个小型框架，用于构建网络应用和 API。
* [Django](https://www.djangoproject.com/) - 一个功能更丰富的网络应用构建框架。Django 尤其适合设计复杂、内容丰富的网络应用。
* [Beautiful Soup](https://www.crummy.com/software/BeautifulSoup/) - 用于解析 HTML 并从中提取信息。适合网页数据抽取。
* [pytest](http://doc.pytest.org/) - 扩展了 Python 的内置断言，并且是最具单元性的模块。
* [PyYAML](http://pyyaml.org/wiki/PyYAML) - 用于读写 [YAML](https://en.wikipedia.org/wiki/YAML) 文件。
* [NumPy](http://www.numpy.org/) - 用于使用 Python 进行科学计算的最基本软件包。它包含一个强大的 N 维数组对象和实用的线性代数功能等。
* [pandas](http://pandas.pydata.org/) - 包含高性能、数据结构和数据分析工具的库。尤其是，pandas 提供 dataframe！
* [matplotlib](http://matplotlib.org/) - 二维绘制库，会生成达到发布标准的高品质图片，并且采用各种硬拷贝格式和交互式环境。
* [ggplot](http://ggplot.yhathq.com/) - 另一种二维绘制库，基于 R's ggplot2 库。
* [Pillow](https://python-pillow.org/) - Python 图片库可以向你的 Python 解释器添加图片处理功能。
* [pyglet](http://www.pyglet.org/) - 专门面向游戏开发的跨平台应用框架。
* [Pygame](http://www.pygame.org/) - 用于编写游戏的一系列 Python 模块。
* [pytz](http://pytz.sourceforge.net/) - Python 的世界时区定义。

在线资源

# 获取所需的信息

要想成为熟练的程序员，需要掌握大量知识。需要了解库、记住语法以及其他细节。此外，让这一切更具挑战的是，技术在不断革新，因为新的技巧和工具会不断出现。

对于编程新手来说，学习所有这些细节并及时获悉新的发展动态似乎是一项不可能完成的任务。的确是这样！具有多年经验的编程专业人士实际上并不是在脑中记下百科全书一样的知识，而是掌握了快速查找信息的技巧。

# 如何搜索

下面是高效网络搜索的一些技巧：

* 在查询时，尝试使用 Python 或要使用的库的名称作为第一个字词。这样会告诉搜索引擎优先显示与你要使用的工具明确相关的结果。
* 创建良好的搜索查询需要多次尝试。如果第一次尝试时没有找到有用的结果，再试一遍。
* 尝试使用在一开始搜索时发现的网页上发现的关键字，使搜索引擎在后续搜索中转到更好的资源。
* 复制粘贴错误消息作为搜索字词。这样会出现错误解释性信息和潜在原因。错误消息可能包括你所写的特定行号引用。只在搜索中包含这些信息之前的错误消息部分。
* 如果找不到问题答案，自己提出问题！StackOverflow 等社区有一些行为规则，如果你要加入该社区，必须了解这些规则，但是别因为这些规则而不愿意使用这些资源。

### 练习题

在编程时，我遇到了下面的错误消息。

UnboundLocalError: local variable 'egg\_count' referenced before assignment

哪个搜索字词最有可能生成实用的结果？

* 

赋值前引用的 egg\_count

* 

用 Python 计数

* 

赋值前的引用

* 

Python UnboundLocalError：局部变量

提交

# 在线资源的优先级

虽然有很多关于编程的在线资源，但是并非所有资源都是同等水平的。下面的资源列表按照大致的可靠性顺序排序。

1. [**Python 教程**](https://docs.python.org/3/tutorial/) - 这部分官方文档给出了 Python 的语法和标准库。它会举例讲解，并且采用的语言比主要文档的要浅显易懂。确保阅读该文档的 Python 3 版本！
2. [**Python 语言和库参考资料**](https://docs.python.org/3/index.html) - 语言参考资料和库参考资料比教程更具技术性，但肯定是可靠的信息来源。当你越来越熟悉 Python 时，应该更频繁地使用这些资源。
3. **第三方库文档** - 第三方库会在自己的网站上发布文档，通常发布于 <https://readthedocs.org/> 。你可以根据文档质量判断第三方库的质量。如果开发者没有时间编写好的文档，很可能也没时间完善库。
4. **非常专业的网站和博客** - 前面的资源都是主要资源，他们是编写相应代码的同一作者编写的文档。主要资源是最可靠的资源。次要资源也是非常宝贵的资源。次要资源比较麻烦的是需要判断资源的可信度。[Doug Hellmann](https://doughellmann.com/blog/) 等作者和 [Eli Bendersky](http://eli.thegreenplace.net/) 等开发者的网站很棒。不出名作者的博客可能很棒，也可能很糟糕。
5. [**StackOverflow**](http://stackoverflow.com/) - 这个问答网站有很多用户访问，因此很有可能有人之前提过相关的问题，并且有人回答了！但是，答案是大家自愿提供的，质量参差不齐。在将解决方案应用到你的程序中之前，始终先理解解决方案。如果答案只有一行，没有解释，则值得怀疑。你可以在此网站上查找关于你的问题的更多信息，或发现替代性搜索字词。
6. **Bug 跟踪器** - 有时候，你可能会遇到非常罕见的问题或者非常新的问题，没有人在 StackOverflow 上提过。例如，你可能会在 GitHub 上的 bug 报告中找到关于你的错误的信息。这些 bug 报告很有用，但是你可能需要自己开展一些工程方面的研究，才能解决问题。
7. **随机网络论坛** - 有时候，搜索结果可能会生成一些自 2004 年左右就不再活跃的论坛。如果这些资源是唯一解决你的问题的资源，那么你应该重新思考下寻找解决方案的方式。

启动jupyter notebook服务器

# NumPy 简介

**NumPy** 是 Numerical Python 的简称，它是 Python 中的科学计算基本软件包。NumPy 为 Python 提供了大量数学库，使我们能够高效地进行数字计算。这些课程将简要讲解 NumPy 基本概念，并介绍一些最重要的 NumPy 功能。

在下面的课程中，你将学习：

* 如何导入 NumPy
* 如何使用各种方法创建多维 NumPy ndarray
* 如何访问和更改 ndarray 中的元素
* 如何加载和保存 ndarray
* 如何使用切片选择或更改 ndarray 的子集
* 了解 ndarray 视图和副本之间的区别
* 如何使用布尔型索引并设置操作以选择或更改 ndarray 的子集
* 如何对 ndarray 排序
* 如何对 ndarray 执行元素级操作
* 了解 NumPy 如何使用广播对不同大小的 ndarray 执行操作。

# 下载 NumPy

**Anaconda** 中包含 NumPy。如果你的计算机尚未安装 Anaconda，请参阅 Anaconda 部分，详细了解如何在 PC 或 Mac 设备上安装 Anaconda。

# NumPy 版本

和很多 Python 软件包一样，NumPy 也会不断更新。以下课程在制作时采用的是 NumPy 1.13.0 版。你可以检查你的 NumPy 版本：在 Jupyter notebook 中输入 !conda list numpy，或在 Anaconda 提示符处输入 conda list numpy。如果你的计算机安装的是另一个版本的 NumPy，你可以通过在 Anaconda 提示符处输入 conda install numpy=1.13 更新你的 NumPy 版本。随着新版 NumPy 的推出，一些功能可能会过时或被替换掉，因此确保在运行代码前，安装正确的 NumPy 版本。这样可以保证代码顺利运行。

# NumPy 文档

NumPy 是一个强大的数学库，其中包含很多函数和功能。在这些入门课程中，我们将仅介绍 NumPy 的一些基本功能。如果你想深入学习 NumPy，确保参阅 NumPy 文档：

[NumPy 手册](https://docs.scipy.org/doc/numpy-1.13.0/contents.html)  
[NumPy 用户指南](https://docs.scipy.org/doc/numpy-1.13.0/user/index.html)  
[NumPy 参考资料](https://docs.scipy.org/doc/numpy-1.13.0/reference/index.html#reference)  
[Scipy 讲座](http://www.scipy-lectures.org/intro/numpy/index.html)

# Pandas 简介

**Pandas** 是 Python 中的数据操纵和分析软件包。名称“Pandas”得名自计量经济学 Panel Data（面板数据）一词。Pandas 为 Python 带来了两个新的数据结构，即 **Pandas Series**和 **Pandas DataFrame**。借助这两个数据结构，我们能够轻松直观地处理带标签数据和关系数据。这些课程将简单介绍 Pandas，并讲解一些最重要的 Pandas 功能。

在下面的课程中，你将学习：

* 如何导入 Pandas
* 如何使用各种方法创建 Pandas Series 和 DataFrame
* 如何访问及更改 Series 和 DataFrame 中的元素
* 如何对 Series 执行算术运算
* 如何向 DataFrame 中加载数据
* 如何处理非数 (NaN) 值

学习以下课程的前提是你已经熟悉 NumPy，并且已经学习了之前的 NumPy 课程。因此，为了避免重复讲解，我们将忽略已经在 NumPy 课程中介绍过的大量细节内容。如果你尚未学习 NumPy 课程，建议先学习这些课程。

# 下载 Pandas

**Anaconda** 中包含 Pandas。如果你的计算机尚未安装 Anaconda，请参阅 Anaconda 部分，详细了解如何在 PC 或 Mac 设备上安装 Anaconda。

# Pandas 版本

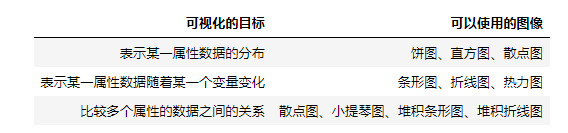
和很多 Python 软件包一样，Pandas 也会时不时地更新。以下课程在制作时采用的是 Pandas 0.22 版。你可以检查你的 Pandas 版本：在 Jupyter notebook 中输入 !conda list pandas，或在 Anaconda 提示符处输入 conda list pandas。如果你的计算机安装的是另一个版本的 Pandas，你可以通过在 Anaconda 提示符处输入 conda install pandas=0.22 更新你的 Pandas 版本。随着新版 Pandas 的推出，一些功能可能会过时或被替换掉，因此确保在运行代码前，安装正确的 Pandas 版本。这样可以保证代码顺利运行。

# Pandas 文档

Pandas 是一个强大的数据分析库，其中包含很多函数和功能。在这些入门课程中，我们将仅介绍 Pandas 的一些基本功能。如果你想深入学习 Pandas，确保参阅 Pandas 文档：

[**Pandas 文档**](https://pandas.pydata.org/pandas-docs/stable/)

# Matplotlib 和 Seaborn 可视化数据



## 条形图：

import matplotlib.pyplot as plt

import seaborn as sb

base\_color = sb.color\_palette()[0]

sb.countplot(data=pokemon, x ='generation\_id',color=base\_color)

## 饼图：

sorted\_counts = df['cat\_var'].value\_counts()

plt.pie(sorted\_counts, labels = sorted\_counts.index, startangle = 90,

counterclock = **False**);

plt.axis('square')

## 直方图：

plt.figure(figsize = [10, 5]) *# larger figure size for subplots*

*# histogram on left, example of too-large bin size*

plt.subplot(1, 2, 1) *# 1 row, 2 cols, subplot 1*

bin\_edges = np.arange(0, df['num\_var'].max()+4, 4)

plt.hist(data = df, x = 'num\_var', bins = bin\_edges)

*# histogram on right, example of too-small bin size*

plt.subplot(1, 2, 2) *# 1 row, 2 cols, subplot 2*

bin\_edges = np.arange(0, df['num\_var'].max()+1/4, 1/4)

plt.hist(data = df, x = 'num\_var', bins = bin\_edges)

### 替代方法

seaborn 函数 [**distplot**](https://seaborn.pydata.org/generated/seaborn.distplot.html) 也可以用于绘制直方图，并且与其他单变量绘图函数集成到一起。

sb.distplot(df['num\_var'])

distplot 函数具有指定直方图分箱的内置规则，默认情况下，会在数据上方绘制一个 (KDE) 核密度估计。纵轴基于 KDE，而不是直方图：长条的高度之和不一定等于 1，但是曲线下方的面积应该等于 1。如果你想详细了解 KDE，请参阅这节课末尾的补充内容。

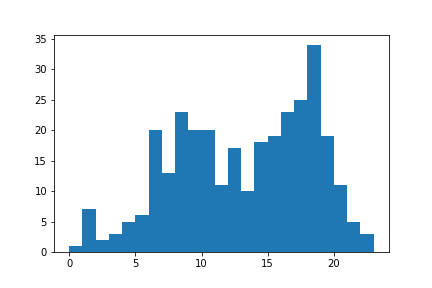
虽然默认的 distplot 分箱尺寸可能比固定的 .hist = 10 更合适，但是你依然需要进行调整，使分箱尺寸等于四舍五入的值。你可以使用其他参数设置绘制直方图并像之前一样指定分箱：

bin\_edges = np.arange(0, df['num\_var'].max()+1, 1)

sb.distplot(df['num\_var'], bins = bin\_edges, kde = **False**,

hist\_kws = {'alpha' : 1})

alpha（透明度）设置必须当做字典与 "hist\_kws" 关联，因为还有其他底层绘图函数（例如 KDE）具有自己的可选关键字参数。

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/c0f67015-ef78-4bd8-98ea-78c25b5ac5b2)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/c0f67015-ef78-4bd8-98ea-78c25b5ac5b2)**

上述代码的结果和上述分箱宽度为 1 的直方图完全一样。纵轴的单位也以计数的形式出现了。

总之，如果你只想了解数据的直方图分布情况，而不是 distplot 提供的额外信息，则为了简便，建议只使用 Matplotlib 的 hist 函数。另一方面，如果你想快速了解如何为直方图选择代表性的分箱尺寸，建议在自定义之前，先快速查看下基本的 distplot。

### 标尺和变换

某些数据分布可以进行标尺变换。最常见的示例是近似符合[**对数正态**](https://en.wikipedia.org/wiki/Log-normal_distribution)分布的数据。即采用原始单位的话，看起来非常偏态：很多数据点的值很小，有一个很长的尾部，尾部数据点的值很大。但是对这些值取对数的话，数据看起来是正态分布的。

plt.figure(figsize = [10, 5])

*# histogram on left: natural units*

plt.subplot(1, 2, 1)

bin\_edges = np.arange(0, ln\_data.max()+100, 100)

plt.hist(ln\_data, bins = bin\_edges)

*# histogram on right: directly log-transform data*

plt.subplot(1, 2, 2)

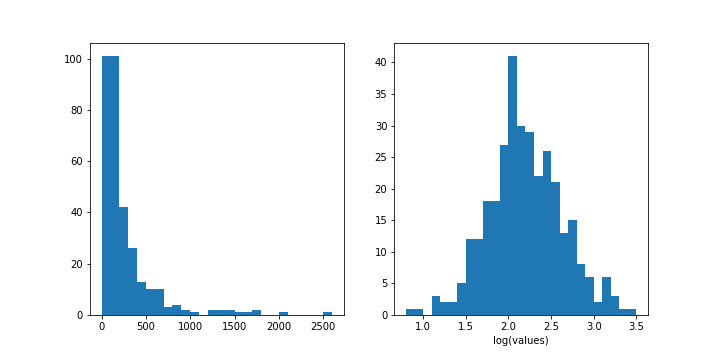
log\_ln\_data = np.log10(ln\_data)

log\_bin\_edges = np.arange(0.8, log\_ln\_data.max()+0.1, 0.1)

plt.hist(log\_ln\_data, bins = log\_bin\_edges)

plt.xlabel('log(values)') *# add axis label for clarity*

（文档[**numpy log10**](https://docs.scipy.org/doc/numpy/reference/generated/numpy.log10.html), [**matplotlib xlabel**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xlabel.html)）

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)**

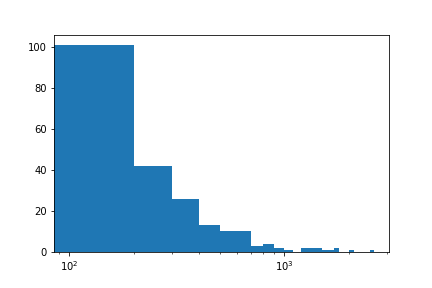
在左图中，超过 1000 的很高的值将大部分数据点推到了最左侧的分箱中。对于右图，对数变换将这些数据点看起来和剩余的数据点保持一致，整个数据看起来是单峰的。右图的最大问题是 x 轴的单位很难解释：对于很多人来说，他们只会相对娴熟地将整数的对数值转换为原始值（假设示例中的基数是 如10 等相对比较好算的值）。

这时候标尺变换就派上用场了。在标尺变换中，值之间的空隙基于变换的标尺，但是你可以用变量的原始单位解释数据。此外，你不需要设定新的特征，这很方便。Matplotlib 的 [**xscale**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xscale.html) 函数包含几个内置的变换：我们将使用 "对数" 标尺。

bin\_edges = np.arange(0, ln\_data.max()+100, 100)

plt.hist(ln\_data, bins = bin\_edges)

plt.xscale('log')

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)**

对于该图，注意两点：首先，即使数据采用的是对数标尺，分箱依然呈线性分布。意味着它们的尺寸从左到右由宽变窄，因为值会以倍数增大。其次，默认的标签设置依然很难解释，并且是稀疏的。

要处理分箱尺寸问题，我们只需将它们变成 10 的各次幂并且均匀分布。根据你所绘制的数据，2 次幂等其他次幂可能更合适。对于刻度，我们可以使用 [**xticks**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.xticks.html) 以原始单位指定位置和标签。注意：我们并没有更改数据的值，只是改变了显示方式。在 10 次幂的整数之间，我们没有表示均匀刻度的整数，但是可以很接近。对于 10 次幂对数变换，设置 1-3-10 或 1-2-5-10 这样的循环刻度很有用。

bin\_edges = 10 \*\* np.arange(0.8, np.log10(ln\_data.max())+0.1, 0.1)

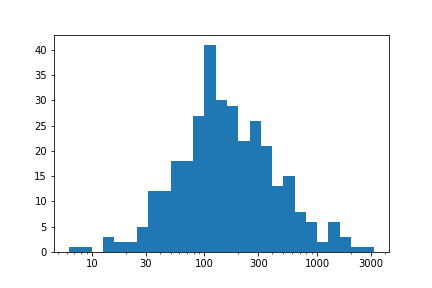
plt.hist(ln\_data, bins = bin\_edges)

plt.xscale('log')

tick\_locs = [10, 30, 100, 300, 1000, 3000]

plt.xticks(tick\_locs, tick\_locs)

请务必在 xscale 之后指定 xticks，因为该函数具有内置的刻度设置。

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)**

我们获得了和进行直接对数变换时得出的图形一样的图形，但是现在的刻度和标签看起来美观多了。

### 替代方法

注意，对数变换并不是唯一的变换方式。在进行对数变换时，数据值必须全是正数；0 或负数无法取对数。此外，对数变换表明对对数标尺进行加法将导致原始标尺出现倍数变化，这是在数据建模时需要注意的重要事项。你可以根据数据判断该选择什么类型的变换。例如，[**这篇维基百科文章的此部分**](https://en.wikipedia.org/wiki/Log-normal_distribution#Occurrence_and_applications)介绍了几个运用对数正态分布的示例场合。

如果你想使用 xscale 中未提供的其他变换，则需要进行某些特征工程。在这种情形下，我们需要写一个应用变换和还原过程的函数，以保持系统性。当我们用变换单位指定值，并且需要获得以原始单位计量的值时，还原功能就很有用。为了进行演示，假设我们想要以平方根变换的形式绘制上述数据。（或许这些数字表示面积，我们认为有必要按照半径、长度或其他一维近似值来对数据建模）。我们可以如下所示地绘制变换后的分布情况：

**def** **sqrt\_trans**(x, inverse = False):

""" transformation helper function """

**if** **not** inverse:

**return** np.sqrt(x)

**else**:

**return** x \*\* 2

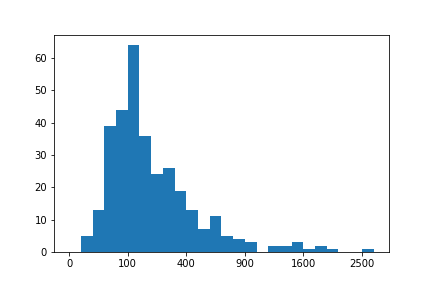
bin\_edges = np.arange(0, sqrt\_trans(ln\_data.max())+1, 1)

plt.hist(ln\_data.apply(sqrt\_trans), bins = bin\_edges)

tick\_locs = np.arange(0, sqrt\_trans(ln\_data.max())+10, 10)

plt.xticks(tick\_locs, sqrt\_trans(tick\_locs, inverse = **True**).astype(int))

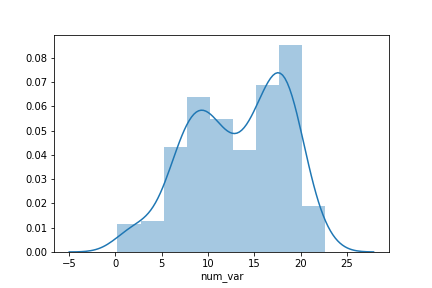
注意 ln\_data 是一个 pandas Series，因此我们可以使用该函数的 [**apply**](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.apply.html) 方法。如果是 NumPy 数组，则需要像在其他情形下一样应用该函数。刻度位置也应该用原始值指定，我们对 xticks 的第一个参数应用标准变换函数。

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/7623c273-f807-4da3-9ce4-93124bc9ce42)**

## 核密度估计

之前在这节课，你通过使用 seaborn 的[**distplot**](https://seaborn.pydata.org/generated/seaborn.distplot.html#seaborn.distplot) 函数（在直方图上方绘制 KDE）见到了[**核密度估计**](https://en.wikipedia.org/wiki/Kernel_density_estimation) (KDE) 示例。

sb.distplot(df['num\_var'])

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/cfbaee5f-1a27-4457-8a67-8421bf19cb05)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/cfbaee5f-1a27-4457-8a67-8421bf19cb05)**

核密度估计是估计变量的概率密度函数的一种方式。在 KDE 图形中，你可以将每个观测值看做一个小的块状区域。将这些块状区域都堆叠到一起可以生成最终的密度曲线。默认设置使用正态分布内核，但是能够生成 KDE 图形的大部分软件还包括其他内核函数选项。

Seaborn 的 distplot 函数会调用另一个函数 [**kdeplot**](https://seaborn.pydata.org/generated/seaborn.kdeplot.html) 来生成 KDE。以下演示代码还使用了被 distplot 调用的第三个函数 [**rugplot**](https://seaborn.pydata.org/generated/seaborn.rugplot.html)。在轴须图中，数据点被描绘成数值轴上的破折号。

data = [0.0, 3.0, 4.5, 8.0]

plt.figure(figsize = [12, 5])

*# left plot: showing kde lumps with the default settings*

plt.subplot(1, 3, 1)

sb.distplot(data, hist = **False**, rug = **True**, rug\_kws = {'color' : 'r'})

*# central plot: kde with narrow bandwidth to show individual probability lumps*

plt.subplot(1, 3, 2)

sb.distplot(data, hist = **False**, rug = **True**, rug\_kws = {'color' : 'r'},

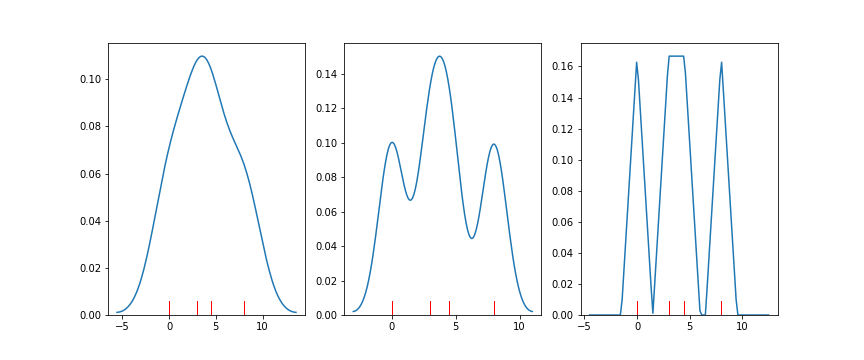
kde\_kws = {'bw' : 1})

*# right plot: choosing a different, triangular kernel function (lump shape)*

plt.subplot(1, 3, 3)

sb.distplot(data, hist = **False**, rug = **True**, rug\_kws = {'color' : 'r'},

kde\_kws = {'bw' : 1.5, 'kernel' : 'tri'})

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/cfbaee5f-1a27-4457-8a67-8421bf19cb05)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/43855abb-dae5-46c7-9167-4b001dabd41e/concepts/cfbaee5f-1a27-4457-8a67-8421bf19cb05)**

根据这种图形解释概率比标准直方图稍微复杂些。纵轴表示的是数据密度，而不是直接的概率。在 KDE 图形中，0 直线和曲线之间的总面积为 1。结果位于两个值之间的概率通过计算这两个值之间的下方面积得出。像这样不借助计算机判断面积大小很难，可能不准确。

虽然通过 KDE 做出具体的概率判断没有直方图直观，但是使用核密度估计依然存在一定的理由。如果数据点相对较少，则 KDE 可以对整体数据分布提供平滑的估计。这些信息可能无法通过直方图轻松地呈现出来，直方图中大量的不连续跳跃性可能会造成误导。

另外要注意的是，KDE 中的带宽参数会指定密度块体的宽度是多少。和直方图的分箱宽度类似，我们需要选择最能表示数据规律的带宽大小。带宽太小的话，数据看起来比实际的噪点更多，带宽太大的话，可能会消除推理数据的有用特征。这一点需要注意，以防可视化软件所选的默认带宽看起来不太合适，或者你想要进一步展开调查。

# group by

groupby就是按xx分组, 它也确实是用来实现这样功能的. 比如, 将一个数据集按A进行分组, 效果是这样

我们尝试使用groupby来尝试实现这样的功能, 不过我们不用A列, 我们将用我们数据集里面的"种族"尝试分组:

data.groupby('race')

<pandas.core.groupby.DataFrameGroupBy object at 0x104fa2208>

这里我们得到了一个叫DataFrameGroupBy的东西, 虽然 pandas 不让我们直接看它长啥样, 但是你将它想象成上面那幅分组后的图(我手绘的)是完全没有问题的.

这篇稿主要介绍如何鼓捣这个*DataFrameGroupBy*, 这个*DataFrameGroupBy*主要的功能能是允许你在不额外写循环的情况下, 快速对每一组数据进行操作

基本操作

最基本的就是组内计数, 求和, 求均值, 求方差, 求blablabla...  
比如, 要求被不同种族内被击毙人员年龄的均值:

data.groupby('race')['age'].mean()

race  
A 36.605263  
B 31.635468  
H 32.995157  
N 30.451613  
O 33.071429  
W 40.046980  
Name: age, dtype: float64

上面我们求得了各个种族中被击毙的人员的平均年龄, 得到的是一个Series, 每一行对应了每一组的mean, 除此之外你还可以换成std, median, min, max这些基本的统计数据

上面age是连续属性, 我们还可以操作离散属性, 比如对不同取值的计数: .value\_counts()  
以下尝试求不同种族内, 是否有精神异常迹象的分别有多少人

data.groupby('race')['signs\_of\_mental\_illness'].value\_counts()

race signs\_of\_mental\_illness  
A False 29  
True 10  
B False 523  
True 95  
H False 338  
True 85  
N False 23  
True 8  
O False 21  
True 7  
W False 819  
True 382  
Name: signs\_of\_mental\_illness, dtype: int64

注: 这时, 组内操作的结果不是单个值, 是一个序列, 我们可以用*.unstack()*将它展开

data.groupby('race')['signs\_of\_mental\_illness'].value\_counts().unstack()

| **signs\_of\_mental\_illness** | **False** | **True** |
| --- | --- | --- |
| **race** |  |  |
| A | 29 | 10 |
| B | 523 | 95 |
| H | 338 | 85 |
| N | 23 | 8 |
| O | 21 | 7 |
| W | 819 | 382 |

***方法总结***

* 首先通过groupby得到DataFrameGroupBy对象, 比如data.groupby('race')
* 然后选择需要研究的列, 比如['age'], 这样我们就得到了一个SeriesGroupby, 它代表每一个组都有一个Series
* 对SeriesGroupby进行操作, 比如.mean(), 相当于对每个组的Series求均值

注: 如果不选列, 那么第三步的操作会遍历所有列, pandas会对能成功操作的列进行操作, 最后返回的一个由操作成功的列组成的*DataFrame*

更多基本操作

[选择一个组](https://pandas.pydata.org/pandas-docs/stable/groupby.html#selecting-a-group)  
不细讲啦, 我自己觉得跟筛选数据差不多

## 可视化

这是我非常喜欢Groupby的一个地方, 它能够帮你很轻松地分组画图, 免去手写每个组的遍历的烦恼, 还能为你每个组分好颜色.

### 场景一: 不同种族中, 逃逸方式分别是如何分布的?

(属性A的不同分组中, 离散属性B的情况是怎么样的 )

* 一种传统做法是:
  1. 遍历每个组
  2. 然后筛选不同组的数据
  3. 逐个子集画条形图 (或者其他表示)

races = np.sort(data['race'].dropna().unique())

fig, axes = plt.subplots(1, len(races), figsize=(24, 4), sharey=True)

for ax, race in zip(axes, races):

data[data['race']==race]['flee'].value\_counts().sort\_index().plot(kind='bar', ax=ax, title=race)

还不错, 但是使用Groupby能让我们直接免去循环, 而且不需要烦人的筛选, 一行就完美搞定

data.groupby('race')['flee'].value\_counts().unstack().plot(kind='bar', figsize=(20, 4))

**方法总结**

* 首先, 得到分组操作后的结果data.groupby('race')['flee'].value\_counts()
* 这里有一个之前介绍的.unstack操作, 这会让你得到一个DateFrame, 然后调用条形图, pandas就会遍历每一个组(unstack后为每一行), 然后作各组的条形图

### 场景二: 按不同逃逸类型分组, 组内的年龄分布是如何的?

(属性A的不同分组中, 连续属性B的情况是怎么样的)

data.groupby('flee')['age'].plot(kind='kde', legend=True, figsize=(20, 5))

**方法总结**

这里data.groupby('flee')['age']是一个SeriesGroupby对象, 顾名思义, 就是每一个组都有一个Series. 因为划分了不同逃逸类型的组, 每一组包含了组内的年龄数据, 所以直接plot相当于遍历了每一个逃逸类型, 然后分别画分布图.

pandas 会为不同组的作图分配颜色, 非常方便

## 高级操作

### 场景三: 有时我们需要对组内不同列采取不同的操作

比如说, 我们按flee分组, 但是我们需要对每一组中的年龄求中位数, 对是否有精神问题求占比

这时我们可以这样做

data.groupby('race').agg({'age': np.median, 'signs\_of\_mental\_illness': np.mean})

|  | **age** | **signs\_of\_mental\_illness** |
| --- | --- | --- |
| **race** |  |  |
| A | 35.0 | 0.256410 |
| B | 30.0 | 0.153722 |
| H | 31.0 | 0.200946 |
| N | 29.0 | 0.258065 |
| O | 29.5 | 0.250000 |
| W | 38.0 | 0.318068 |

***方法总结***  
这里我们操作的data.groupby('race')是一个DataFrameGroupby, 也就是说, 每一组都有一个DataFrame

我们把对这些DataFrame的操作计划写成了了一个字典{'age': np.median, 'signs\_of\_mental\_illness': np.mean}, 然后进行agg, (aggragate, 合计)

然后我们得到了一个DataFrame, 每行对应一个组, 没列对应各组DataFrame的合计信息, 比如第二行第一列表示, 黑人被击毙者中, 年龄的中位数是30, 第二行第二列表示, 黑人被击毙者中, 有精神疾病表现的占15%

### 场景四: 我们需要同时求不同组内, 年龄的均值, 中位数, 方差

data.groupby('flee')['age'].agg([np.mean, np.median, np.std])

|  | **mean** | **median** | **std** |
| --- | --- | --- | --- |
| **flee** |  |  |  |
| Car | 33.911765 | 33.0 | 11.174234 |
| Foot | 30.972222 | 30.0 | 10.193900 |
| Not fleeing | 38.334753 | 36.0 | 13.527702 |
| Other | 33.239130 | 33.0 | 9.932043 |

***方法总结***

现在我们对一个SeriesGroupby同时进行了多种操作. 相当于同时得到了这三行的结果:

data.groupby('flee')['age'].mean()

data.groupby('flee')['age'].median()

data.groupby('flee')['age'].std()

所以这其实是[基本操作](#%E5%9F%BA%E6%9C%AC%E6%93%8D%E4%BD%9C)部分的进阶

### 场景五: 结合场景三和场景四可以吗?

答案是肯定的, 请看

data.groupby('flee').agg({'age': [np.median, np.mean], 'signs\_of\_mental\_illness': np.mean})

|  | **age** |  | **signs\_of\_mental\_illness\_mean** |
| --- | --- | --- | --- |
| **flee** | **median** | **mean** | **mean** |
| Car | 33.0 | 33.911765 | 0.114286 |
| Foot | 30.0 | 30.972222 | 0.115646 |
| Not fleeing | 36.0 | 38.334753 | 0.319174 |
| Other | 33.0 | 33.239130 | 0.072917 |

但是这里有一个问题, 这个列名分了很多层级, 我们可以进行重命名:

agg\_df = data.groupby('flee').agg({'age': [np.median, np.mean], 'signs\_of\_mental\_illness': np.mean})

agg\_df.columns = ['\_'.join(col).strip() for col in agg\_df.columns.values]

agg\_df

|  | **age\_median** | **age\_mean** | **signs\_of\_mental\_illness\_mean** |
| --- | --- | --- | --- |
| **flee** |  |  |  |
| Car | 33.0 | 33.911765 | 0.114286 |
| Foot | 30.0 | 30.972222 | 0.115646 |
| Not fleeing | 36.0 | 38.334753 | 0.319174 |
| Other | 33.0 | 33.239130 | 0.072917 |

***方法总结***   
注意这里agg接受的不一定是np.mean这些函数, 你还可以进行自定义函数哦

## 总结

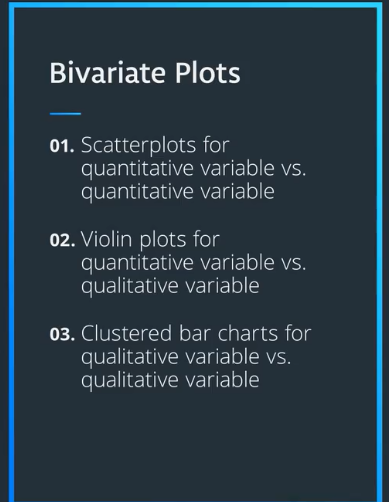
Groupby 可以简单总结为 split, apply, combine, 也就是说:

* **split** : 先将数据按一个属性分组 (得到 *DataFrameGroupby* / *SeriesGroupby* )
* **apply** : 对每一组数据进行操作 (取平均 取中值 取方差 或 自定义函数)
* **combine**: 将操作后的结果结合起来 (得到一个*DataFrame* 或 *Series* 或可视化图像)

希望看完本文你已经对groupby的使用有清晰的印象, 并充满信心, 如果你需要更细致的微操作, 多属性Groupby等, 可以进一步阅读[文档](https://pandas.pydata.org/pandas-docs/stable/groupby.html)

作者：事重  
链接：https://www.jianshu.com/p/42f1d2909bb6  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

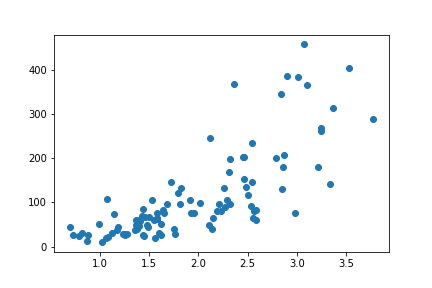
# 第二部分：



## 散点图

如果我们想研究两个数字变量之间的关系，通常会选择**散点图**。在散点图中，每个数据点都单独表示为一个点，x 轴对应一个特征值，y 轴对应另一个特征值。创建散点图的一个基本方式是利用 Matplotlib 的 [**scatter**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html) 函数：

plt.scatter(data = df, x = 'num\_var1', y = 'num\_var2')

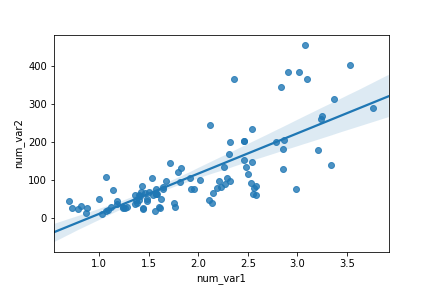
**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/9d1316b3-f339-4d52-b63f-91994aefdd40)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/9d1316b3-f339-4d52-b63f-91994aefdd40)**

可以在这两个变量之间看到一个基本的正向关系，x 轴的值越高，y 轴上的变量值也增大。

### 替代方法

Seaborn 的 [**regplot**](https://seaborn.pydata.org/generated/seaborn.regplot.html) 函数会创建散点图并进行回归函数拟合：

sb.regplot(data = df, x = 'num\_var1', y = 'num\_var2')

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/9d1316b3-f339-4d52-b63f-91994aefdd40)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/9d1316b3-f339-4d52-b63f-91994aefdd40)**

默认情况下，回归函数是线性的，并包含带阴影的回归估计置信区域。在这种情况下，因为看起来像 log(y) ~ x 关系，因此用原始单位绘制回归线不合适。如果我们不关心回归线，那么可以在 regplot 函数调用中设置 reg\_fit = False。否则，如果我们想对观察到的数据关系绘制回归线，需要变换数据，如上节课中所述。

**def** **log\_trans**(x, inverse = False):

**if** **not** inverse:

**return** np.log10(x)

**else**:

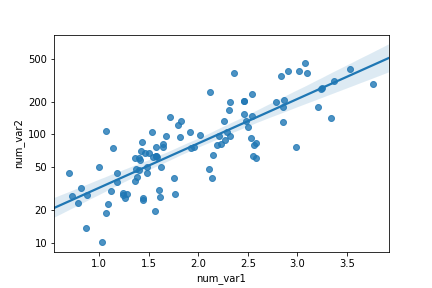
**return** np.power(10, x)

sb.regplot(df['num\_var1'], df['num\_var2'].apply(log\_trans))

tick\_locs = [10, 20, 50, 100, 200, 500]

plt.yticks(log\_trans(tick\_locs), tick\_locs)

注意，x 轴和 y 轴值可以直接设为 Series 或数组，而不是参考 dataframe 中的列。

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/9d1316b3-f339-4d52-b63f-91994aefdd40)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/9d1316b3-f339-4d52-b63f-91994aefdd40)**

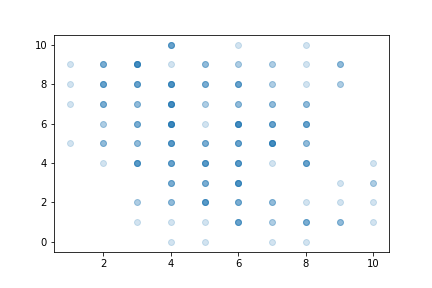
## 重叠、透明度和抖动

如果要绘制大量数据点，或者数字变量是离散值，那么直接使用散点图可能无法呈现足够的信息。图形可能会出现\_重叠\_，由于大量数据重叠到一起，导致很难看清变量之间的关系。

plt.scatter(data = df, x = 'disc\_var1', y = 'disc\_var2')

在上图中，我们可以推断出两个变量负相关，但是数据的变化程度和关系强度并不明显。在这种情形下，我们需要应用透明度和\_抖动\_，使散点图能呈现更多的信息。可以通过在 [**scatter**](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html) 函数调用中添加 "alpha" 参数来表示透明度，"alpha" 的值可以介于 0（完全不透明，不可见） 到 1（完全透明）之间。

plt.scatter(data = df, x = 'disc\_var1', y = 'disc\_var2', alpha = 1/5)

**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/bd599deb-e989-4ce2-9dc2-d62316ecc0f1)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/bd599deb-e989-4ce2-9dc2-d62316ecc0f1)**

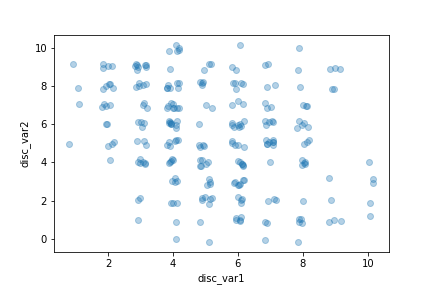
重叠的点越多，图像颜色就越深。从图中可以看出，两个数字变量之间存在一定的负相关关系。x 轴上的 0 和 10 比中间值出现的频率小多了。

除了设置透明度之外，我们还可以通过抖动使每个点稍微偏离真实值所对应的位置。这并不是 scatter 函数中的直接选项，而是 seaborn 的 [**regplot**](https://seaborn.pydata.org/generated/seaborn.regplot.html) 函数的内置选项。可以单独添加 x 轴和 y 轴抖动，不会影响到回归方程的拟合情况：

sb.regplot(data = df, x = 'disc\_var1', y = 'disc\_var2', fit\_reg = **False**,

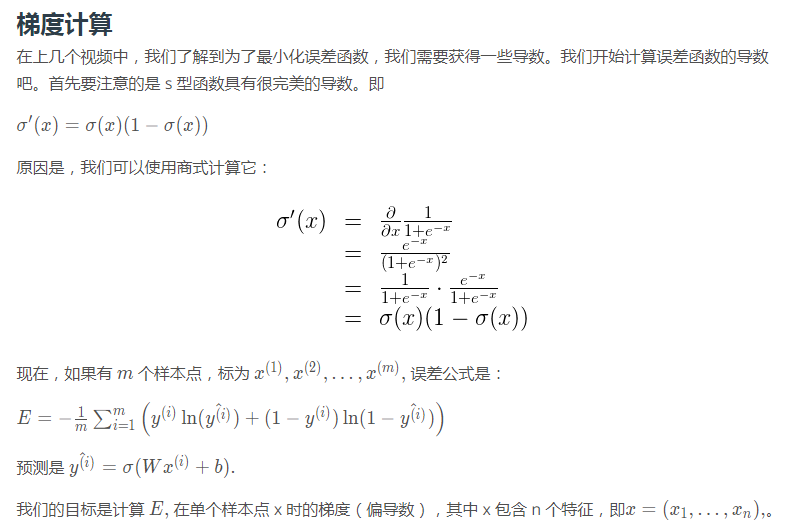
x\_jitter = 0.2, y\_jitter = 0.2, scatter\_kws = {'alpha' : 1/3})

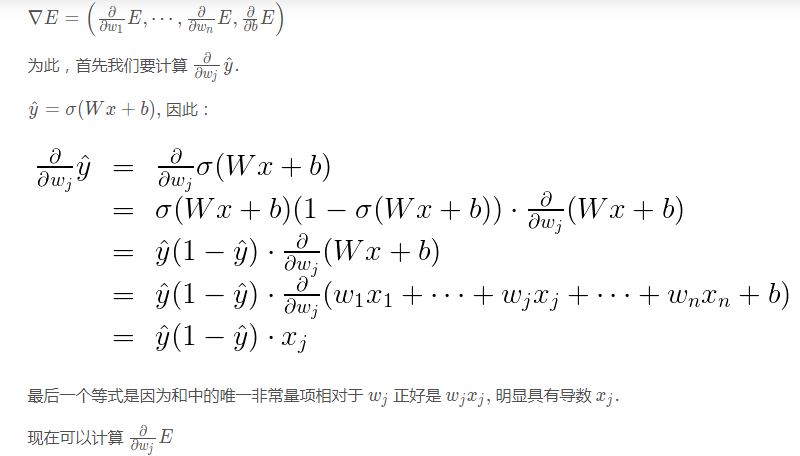
抖动设置将导致每个点统一地在真实值的 ±0.2 范围内抖动。注意透明度已经更改为分配给“scatter\_kws”参数的字典。这么设置的原因是让透明度明确地与 regplot 函数的 scatter 部分相关联。

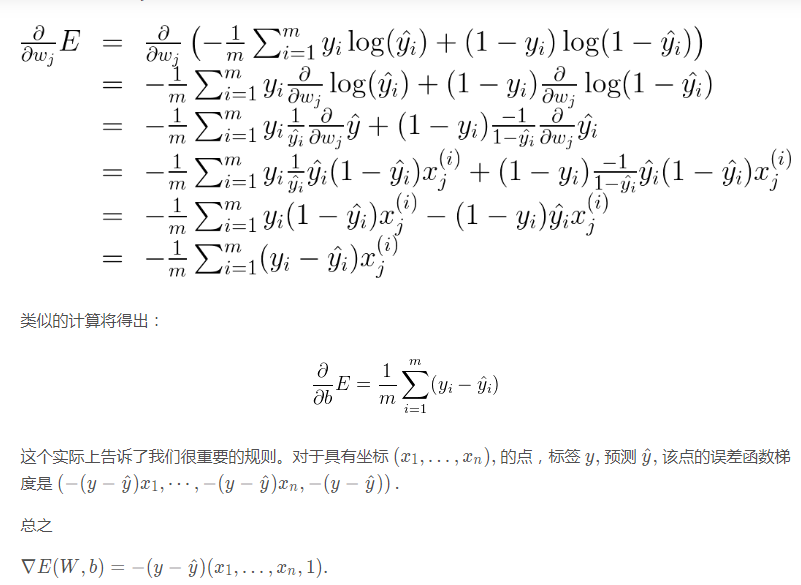
**[[](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/bd599deb-e989-4ce2-9dc2-d62316ecc0f1)](https://classroom.udacity.com/nanodegrees/nd089-cn/parts/78c9ffb7-64df-4aee-bce7-4e0c4c02fe53/modules/dd3e4af8-d576-427e-baae-925fd16ff2ff/lessons/0491d74e-dcd8-4700-a971-a7f1b0a26ddb/concepts/bd599deb-e989-4ce2-9dc2-d62316ecc0f1)**

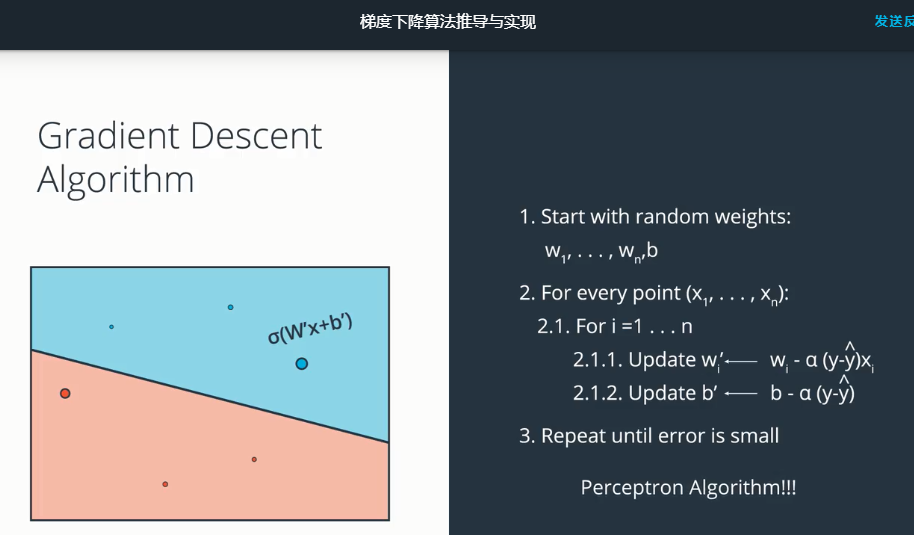
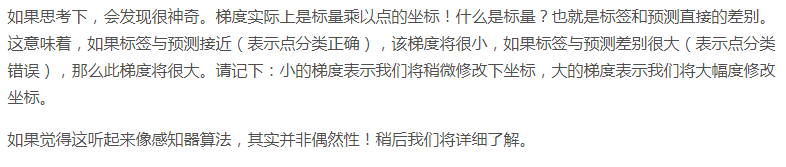
# 第三部分 神经网络

## 梯度下降实现









# 第四部分 图像分类

# 