第 6 章 Seaborn 绘图和可视化

6.1 Seaborn 概述

Seaborn 是一个基于 Matplotlib 的 Python 数据可视化库，它提供了一种高级绘图接口，可以绘制美观且信息丰富的各类统计图形。相对于 Matplotlib 的复杂性，Seaborn 对 Matplotlib 核心库进行了更高阶的 API 封装，因此更加简单易用。Seaborn 与 Pandas 结合的非常紧密，能够对整个数据集的数据帧或数组进行操作，通过语义映射和统计聚合构建统计图形，使用户注意力主要聚焦于“画什么”而不是“怎么画”。其主要主要优势体现在：

- 绘图接口集成度更高，使用方便
- 支持数据分布、相关性、回归、分类等多种数据分析类型
- 支持 Pandas、Numpy 等丰富的数据类型
- 绘图风格具有更高的定制性

正是由于这些特点，在进行探索性数据分析（EDA，Exploratory Data Analysis）过程中，Seaborn 往往更为高效。需要指出的是，Seaborn 与 Matplotlib 的关系是互为补充而非替代，在某些特定场景下则仍需利用 Matplotlib 的灵活性进行更为细致的个性化定制。

6.1.1 Seaborn AIP 简介

Seaborn 的绘图功能是通过一组绘图函数（API）实现的，其组织方式是分层的、扁平的和模块化的，所有功能都可以在顶层调用，各模块中的函数共享许多底层代码，因此可以通过不同方式实现相似的可视化目标。这样，探索数据集时可以很容易在不同的可视化效果之间切换，充分利用各种可视化效果的优势。

总体上，Seaborn 的绘图功能可以分为 5 个主要模块：关系图、分布图、分类图、回归图和矩阵图。此外，还有结构化网格图主要用于多个子图的集成。图形外观及美化方面 Seaborn 主要提供了主题风格和调色板 2 个模块。

Seaborn 中的 API 分为“Figure-level”（图形级）和“Axes-level”（子图级，“Axes”本义是指一个坐标系，在一个 Figure 对象中创建一个坐标系即一个绘图区，也就是子图）。图形级 API 是通过 Seaborn 对象（通常是 FacetGrid）与 Matplotlib 进行交互实现图形管理的。直观上看，对应的是图 6-1 中的红色框。每个模块都有一个图形级 API，它为其各种子图级 API 提供了统一的接口。子图级 API 是将数据绘制到单独的 matplotlib.pyplot.Axes 对象上，实现与 Matplotlib 灵活紧密的结合，对应的是图 6-1 中的蓝色框。可以把 Figure 看做“画板”而把“Axes”看做“画纸”，显然一个 Figure 对象至少包含一个 Axes 对象。

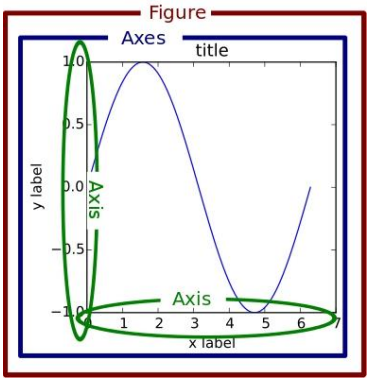


图 6-1 图形的构成

（https://matplotlib.org/1.5.1/faq/usage_faq.html#parts-of-a-figure）

如图 6-2 所示，relplot()、displot()和 catplot()分别是关系图、分布图、分类图的 Figure-level API，下面的 scatterplot()、lineplot()、histplot()、kdeplot()、ecdfplot()等分别为对应模块的 Axes-level API。

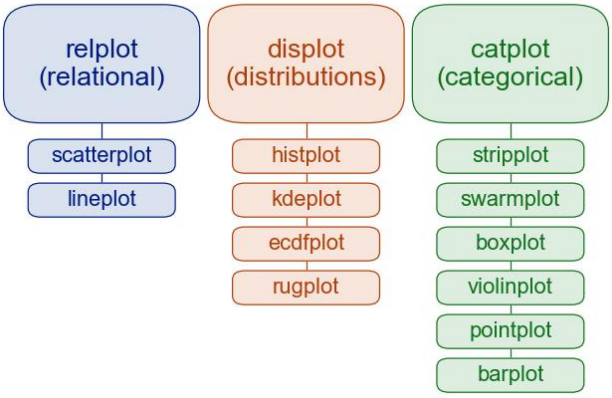
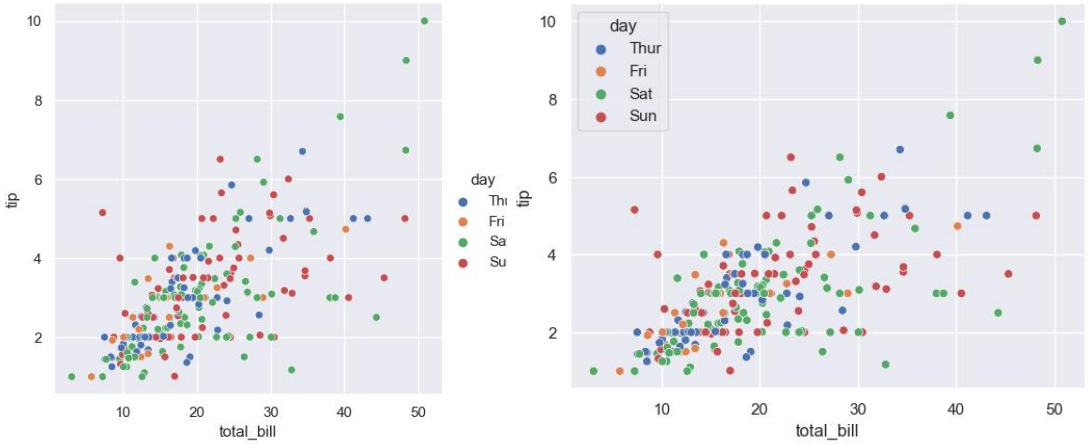


图 6-2 Figure-level API 和 Axes-levelAPI

（https://seaborn.pydata.org/tutorial/function_overview.html）

以关系图模块为例，共包含 relplot()、scatterplo()t 和 lineplot() 3 个 API。relplot()是 Figure-level 接口，scatterplot()和 lineplot()是 Axes-level 接口，可以看作是 relplot()的一种特定实现。调用 relplot()函数时，设置 kind 参数为“scatter”和“line”就分别对应 scatterplot()和 lineplot()，具体如图 6-3 所示。



a. relplot()参数 kind=“scatter”　　b. scatterplot()

图 6-3 对应的 Figure、Axes 级函数绘图对比

总的来说，Figure-level API 大多数情况下使用更加方便、绘图时的集成化程度更高，可以轻松地创建具有多个子图的复杂图形，在各种图表类型之间切换也更加容易；Axes-level API 与 Matplotlib 结合更紧密，定制更加灵活。

6.1.2 Seaborn 数据结构

Seaborn 支持多种不同数据格式，大多数函数都支持 Pandas 或 Numpy 数据对象以及 Python 的内置数据类型（如列表和字典等）。

Seaborn 中的大多数绘图函数都是面向向量的。例如在绘制变量 x 和 y 的关系图时，每个变量都被视作一个向量。Seaborn 接受具有以表格方式组织的多个向量的数据集，包括“长格式（Long-form）”和“宽格式（wide-form）”数据表，二者之间存在根本区别，Seaborn 中有不同的处理方式。

1. 长格式数据表

长格式数据表特点是每一列代表一个变量，每一行代表一个数据实例。例如，Seaborn 自带的“flights”数据集（Seaborn 自带数据集在下一节具体介绍），它记录了从 1949 年到 1960 年每个月的航空公司乘客数量。这个数据集包含三个变量（年、月和乘客数量），具体如图 6-4 所示。

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

图 6-4 “flights”数据集（长格式）

对于长格式数据，表中的列名通过显式分配给绘图函数的对应变量进行绘图。例如：

```
sns.relplot(data=flights, x="year", y="passengers",
            hue="month",kind="line")
```

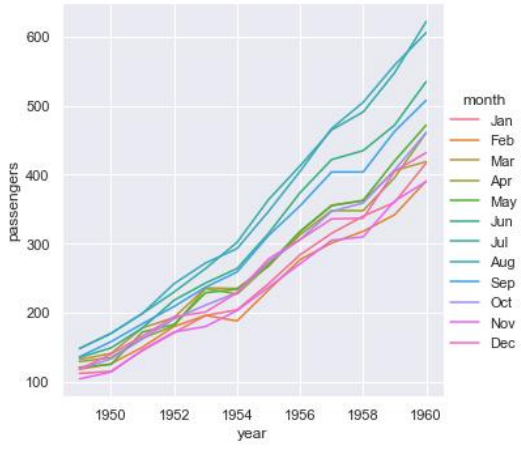


图 6-5 “flights”折线图（长格式）

2. 宽格式数据表

宽格式数据可能更加符合人的习惯，看上去更加直观，如图 6-5 所示。与上面的例子不同，年、月和乘客数量三个变量的“表格化”的组织形式不同，它们并不与“列”一一对应。

month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
year												
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166
1952	171	180	193	181	183	218	230	242	209	191	172	194
1953	196	196	236	235	229	243	264	272	237	211	180	201

图 6-6“flights”数据集（宽格式）

对于长格式数据，可以通过名称访问数据集中的变量。但在宽格式数据表中，由于维度与数据集中的变量之间存在明确的关联，每个数据在表格中的位置都是由行、列的索引决定的，绘图时 Seaborn 能够自动分配变量作为适当的图形元素。也就是说，当 x 和 y 都没有赋值时，Seaborn 将参数 data 视为宽格式，并自动分配变量完成绘图。例如：

```
sns.relplot(data=flights_wide, kind="line")
```

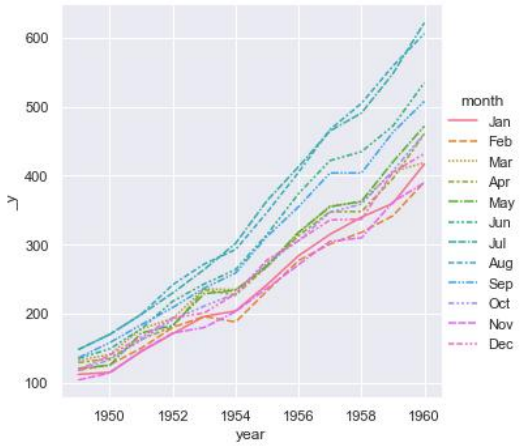


图 6-7“flights”折线图（宽格式）

3. 不规则数据表

还有一些数据集不是规范的长格式或宽格式数据表，可以称之为不规则格式。这种格式中，变量既不是由表的键唯一确定，也不是由表的列唯一确定，这经常发生在需要重复测量的数据中。在重复测量过程中，每一行对应于数据收集的一个“单元”，从而很自然地组织成一个表。例如在 seaborn 自带的“anagrams”数据集，这是一个心理学实验的简单数据集，有 20 个受试者，具体如图 6-6a 所示。绘图时，需要将其转换为长格式数据或宽格式数据。本例中，使用 pandas.DataFrame.melt()方法将其转换为标准长格式数据表，如图 6-6b 所示。

```
anagrams = sns.load_dataset("anagrams")
anagrams_long = anagrams.melt(id_vars=["subidr", "attnr"],
                              var_name="solutions",
                              value_name="score")
anagrams_long.head(20)
```

	subidr	attnr	num1	num2	num3
0	1	divided	2	4.0	7
1	2	divided	3	4.0	5
2	3	divided	3	5.0	6
3	4	divided	5	7.0	5
4	5	divided	4	5.0	8
5	6	divided	5	5.0	6
6	7	divided	5	4.5	6
7	8	divided	5	7.0	8
8	9	divided	2	3.0	7
9	10	divided	6	5.0	6
10	11	focused	6	5.0	6
11	12	focused	8	9.0	8
12	13	focused	6	5.0	9
13	14	focused	8	8.0	7
14	15	focused	8	8.0	7
15	16	focused	6	8.0	7
16	17	focused	7	7.0	6
17	18	focused	7	8.0	6
18	19	focused	5	6.0	6
19	20	focused	6	6.0	5

a. 不规则数据

	subidr	attnr	solutions	score
0	1	divided	num1	2.0
1	2	divided	num1	3.0
2	3	divided	num1	3.0
3	4	divided	num1	5.0
4	5	divided	num1	4.0
5	6	divided	num1	5.0
6	7	divided	num1	5.0
7	8	divided	num1	5.0
8	9	divided	num1	2.0
9	10	divided	num1	6.0
10	11	focused	num1	6.0
11	12	focused	num1	8.0
12	13	focused	num1	6.0
13	14	focused	num1	8.0
14	15	focused	num1	8.0
15	16	focused	num1	6.0
16	17	focused	num1	7.0
17	18	focused	num1	7.0
18	19	focused	num1	5.0
19	20	focused	num1	6.0

b. 转换后的长格式（前 20 条）

图 6-8 “anagrams”数据集

6.1.3 Seaborn 安装

Seaborn 目前最新的版本是 2021 年 8 月发布的 V0.11.2 版本，官方网址是 https://seaborn.pydata.org/。可以从 PyPi（Python Package Index ）安装 Seaborn 的官方版本：

```
pip install seaborn
```

同时，Seaborn 也包含在 Anaconda 的发行版中，可以通过 conda 命令安装：

```
conda install seaborn
```

如果需要指定安装 Seaborn 的版本，可以设置“seaborn=版本号”，例如指定安装版本为 0.11.2 的命令为：

```
conda install seaborn=0.11.2
```

Seaborn 提供了 19 个供学习的数据集，均是 csv 格式。可以通过 sns.get_dataset_names() 返回数据集名称。还可以通过 sns.load_dataset(name)下载数据集，并返回一个 pandas 的 DataFrame 对象。

本章中用的数据集包括：

（1）“flights”：航空公司从 1949 年到 1960 年每个月乘客人数数据。由年、月和乘

客数量等数据构成；

（2）“anagrams”：来自一个心理学实验的数据。二十名测试者，分为注意力集中和不集中两类，每个人玩回文字字谜游戏，每个人记录三种游戏结果得分情况。

（3）“tips”：餐厅小费数据。由星期几、就餐时间、账单总金额、小费金额、消费者性别、是否抽烟以及用餐人数等数据构成；

（4）“fmri”：功能性核磁共振成像数据。由测试者、时间点、事件、刺激类型、大脑区域和信号等数据构成；

（5）“penguins”：企鹅数据集。由种类、岛屿名称、喙长度、喙宽度、鳍足长度、体重、雌雄等数据构成；

（6）“diamonds”：钻石数据集。由价格、克拉、切割质量、台面、钻石颜色、纯度、长度、宽度、深度、总深度百分比等数据构成；

（7）“titanic”：泰坦尼克号数据集。由乘客性别、年龄、所在船仓等级及是否存活等数据构成；

（8）“anscombe”：安斯库姆四重奏数据集。包含 3 列共 4 组数据，每组 11 个点(x,y)，4 组数据均值、方差、相关系数都相同，但分布差别巨大。

（9）“iris”：鸢尾花卉数据集。由花萼长度，花萼宽度，花瓣长度，花瓣宽度及鸢尾花卉种类等数据构成。

需要注意的是，seaborn.load_dataset()下载数据集时经常因为网络连接超时而报错，可以提前下载好数据集 seaborn-data-master.zip 并解压至用户文件夹下的“seaborn-data”文件夹。数据集下载地址为 https：//github.com/mwaskom/seaborn-data。

6.1.4 第一个 Seaborn 示例

首先看一个 Seaborn 绘图的例子，运行结果如图 6-9 所示。

```
1: import seaborn as sns
2: sns.set_theme()
3: tips = sns.load_dataset("tips")
4: sns.relplot(
       data=tips,
       x="total_bill", y="tip", col="time",
       hue="smoker", style="smoker", size="size",
)
```

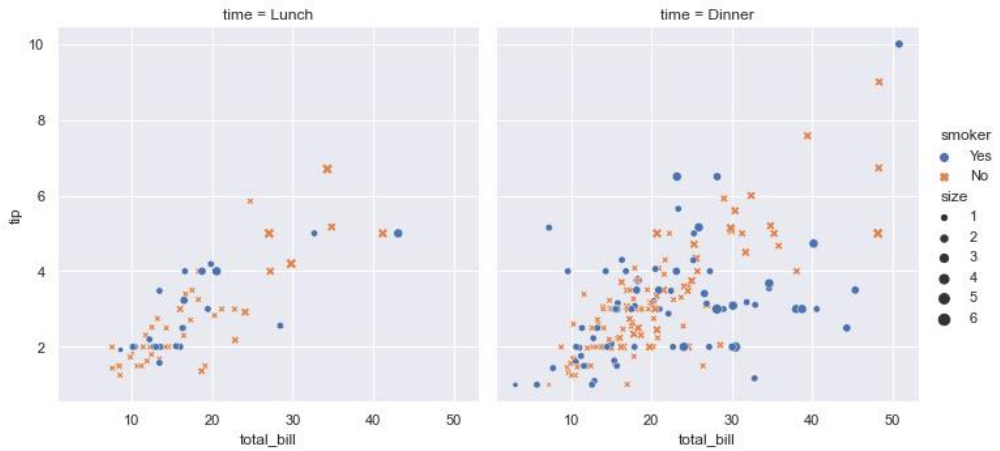


图 6-9 第一个例子的运行结果

第 1 句是导入 Seaborn 库，通常设置简称为“sns”；

第 2 句是设置 Seaborn 绘图的默认主题风格；

第 3 句是载入“tips”数据集，注意本章的示例中都将使用 load_dataset()函数来快速载入示例数据集。这些数据集就是 Pandas 数据帧，也可以通过 pandas.read_csv()加载数据集。

第 4 句是调用关系图函数 relplot()绘制图形，各参数具体含义如下：

“data”：绘制图形用到的数据集；

“x”、“y”：对应 x 轴和 y 轴的数据集向量名称；

“col”：定义了按列展开子图的变量；

“hue”：定义了产生不同颜色分组的变量；

“style”：定义了产生不同样式分组的变量；

“size”：定义了散点标记的大小。

值得注意的是，Seaborn 中各函数定义中的同名参数一般都具有相同或相似的含义和使用方法，在学习中要注意举一反三。

此外，导入相关库及装载数据集是 Seaborn 绘图的准备工作，为了节省篇幅，在此一并说明，在本章的后续例子中只列出绘图的核心代码。

导入以下 4 个库，并装载上述 9 个数据集：

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
flights = sns.load_dataset("flights")
anagrams = sns.load_dataset("anagrams")
tips = sns.load_dataset("tips")
fmri = sns.load_dataset("fmri")
penguins = sns.load_dataset("penguins")
diamonds = sns.load_dataset("diamonds")
titanic = sns.load_dataset("titanic")
anscombe = sns.load_dataset("anscombe")
iris = sns.load_dataset("iris")
```

6.2 基本绘图

6.2.1 关系图

Seaborn 提供了用于表达两个变量关系的图表，主要包括散点图和折线图两类。主要提供了 3 个接口：relplot()（关系图）、scatterplot()（散点图）和 lineplot()（折线图），其中 relplot()为 figure-level 接口，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，接口调用方式和传参模式都是一致的，其核心参数已在 6.1.4 中进行了介绍。

本节中，主要介绍 relplot()函数的用法，它是一个基于 FacetGrid 的图形级绘图接口。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。例如：

```
sns.relplot(x="total_bill", y="tip", data=tips)
```

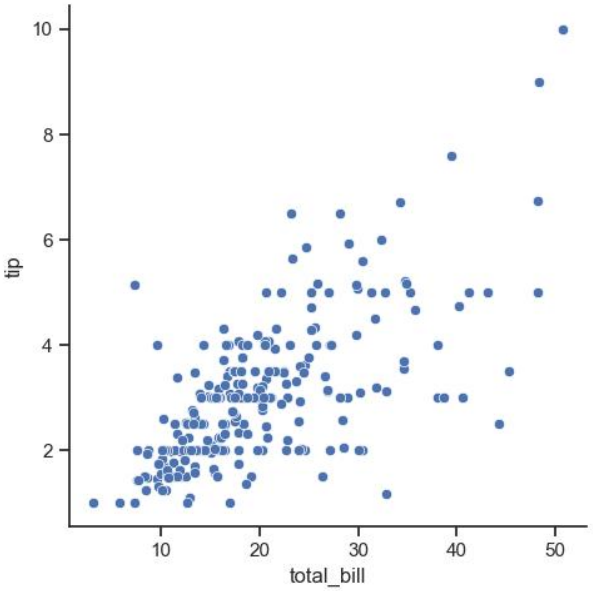
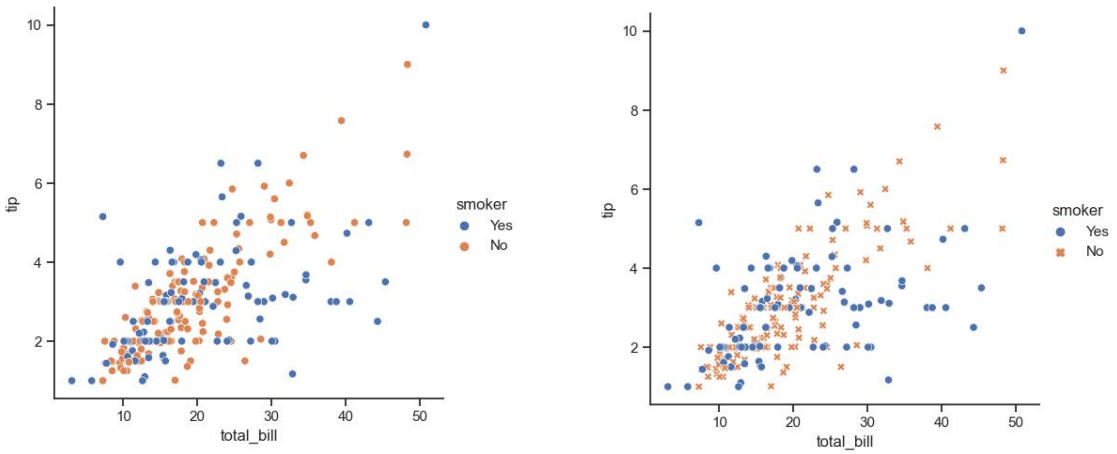


图 6-10 relplot 的基本用法

虽然散点图是二维的，但可以通过设置 hue 参数引入第三维变量，图中的点会根据第三个变量的取值自动设置不同颜色，在 Seaborn 中称之为“语义着色（hue semantic）”，如图 6-11a。为了进一步突出类别差异，可以通过设置 style 参数为每个类使用不同的标记样式，如图 6-11b。例如：

```
sns.relplot(x="total_bill", y="tip", hue="smoker",
            style="smoker", data=tips)
```



a. 仅设置 hue 参数　　b. 设置 hue 和 style 参数

图 6-11 参数 hue 和 style 的用法

可以通过独立改变每个点的色调和样式来表示四个变量，例如：

```
sns.relplot(x="total_bill", y="tip", hue="smoker",
            style="time", data=tips)
```

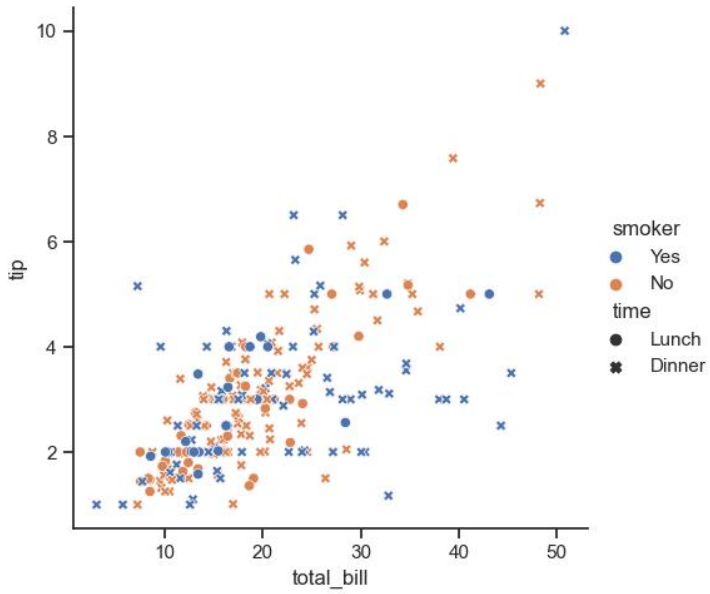


图 6-12 hue 和 style 设置不同变量

还可以通过设置 size、palette 参数改变不同类别数据点的大小及颜色。例如：

```
sns.relplot(x="total_bill", y="tip", hue="size", size="size",
            palette="ch: r=-.5,l=.75", data=tips)
```

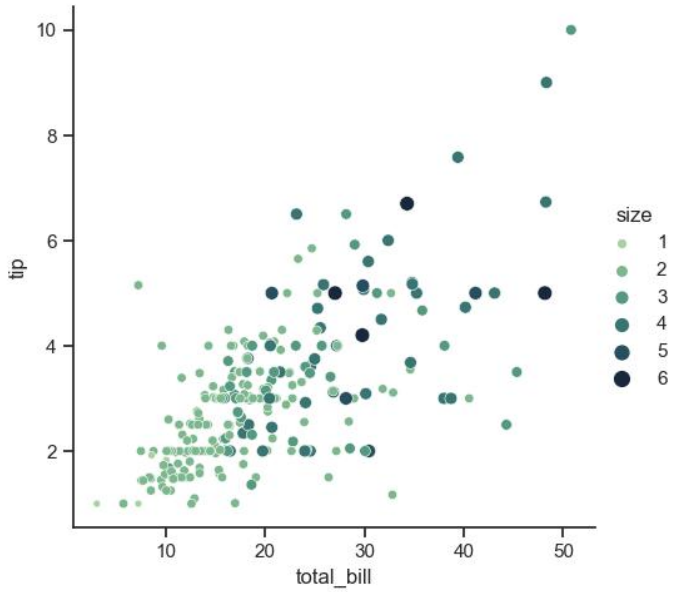


图 6-13 设置 size 和 palette 参数

在某些实际应用中，有时需要观察变量随时间变化的情况，这种情况下，折线图比散点图更为合适。可以使用 lineplot()函数或将 relplot()函数的 kind 参数设置为“line”来实现。例如：

```
df=pd.DataFrame(dict(time=np.arange(500),
                     value=np.random.randn(500).cumsum()))
g = sns.relplot(x="time", y="value", kind="line", data=df)
g.figure.autofmt_xdate() # 旋转 x 轴标签并使其右对齐
```

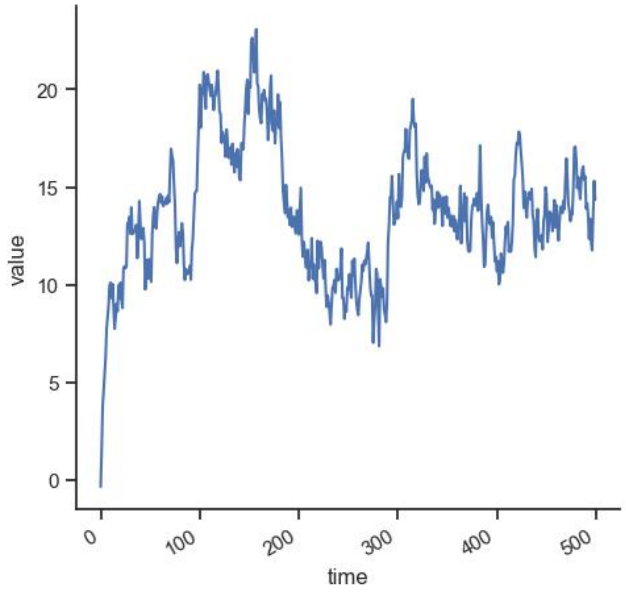
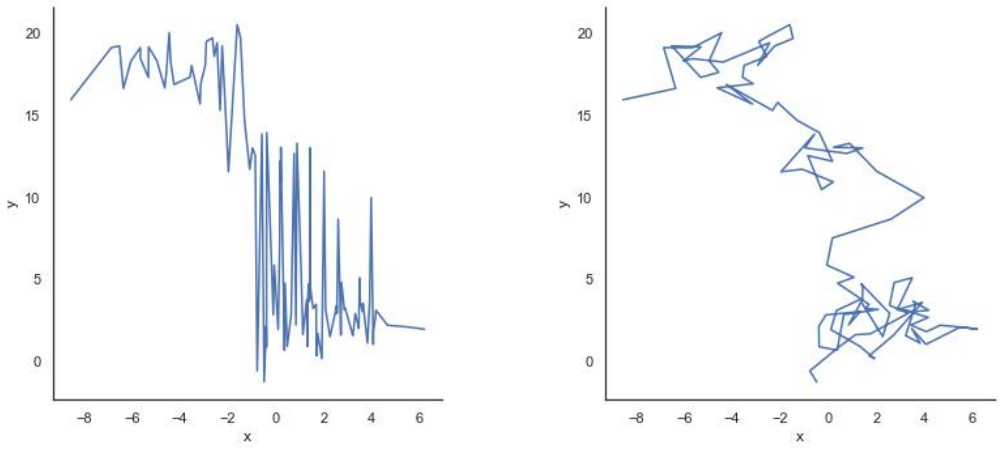


图 6-14 变量随时间变化的折线图

绘制折线图一般是将 y 视为 x 的函数，因此绘图时会自动根据 x 值的大小排序，也可以将 sort 参数设置为 False，禁止自动排序。例如：

```
df = pd.DataFrame(np.random.randn(100, 2).cumsum(axis=0),
                  columns=["x", "y"])
sns.relplot(x="x", y="y", sort=False, kind="line", data=df)
```



a. 默认 sort=True　　b. sort=False

图 6-15 折线图参数 sort 的使用

还有一种数据集是对相同的 x 取值有多个测量值 y。这种情况下 Seaborn 会对每个 x 的取值通过自动计算测量值的平均值和 95%置信区间进行绘图。例如：

```
sns.relplot(x="timepoint", y="signal", kind="line", data=fmri)
```

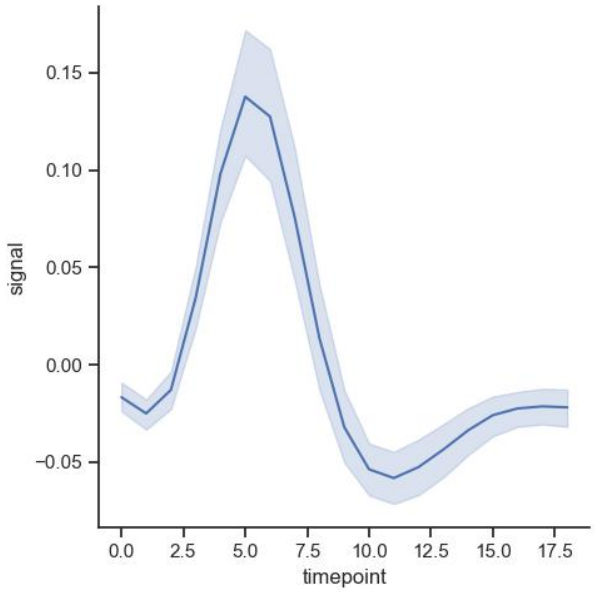
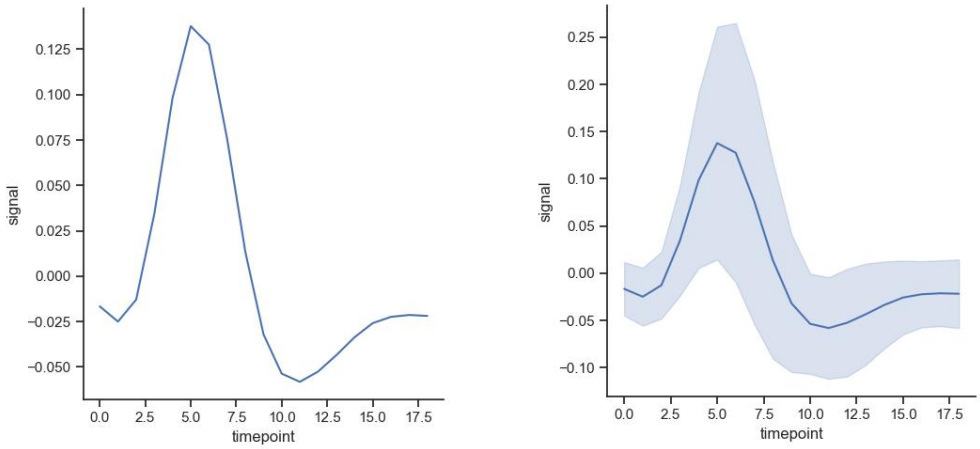


图 6-16 平均值与置信区间

可以通过设置参数“ci”关闭置信区间显示，或调整置信区间大小。ci 可以设置为整数、“sd”（绘制标准差）和“None”。例如：

```
sns.relplot(x="timepoint", y="signal", ci=None,
            kind="line", data=fmri)
sns.relplot(x="timepoint", y="signal", kind="line",
            ci="sd", data=fmri)
```



a. ci = None　　b. ci = “sd”

图 6-17 设置置信区间大小

与散点图类似，折线图也可以通过修改数据元素的颜色、大小和样式来表达三个维度的变量，其中参数“dash=False”表示禁用“破折号”型的虚线。例如：

```
sns.relplot(x="timepoint", y="signal", hue="region",
            style="event",dashes=False,markers=True,
            kind="line", data=fmri)
```

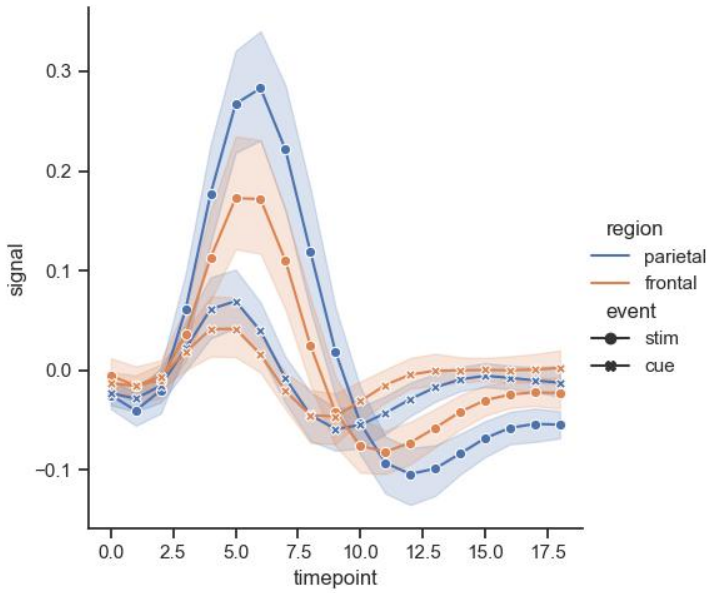


图 6-18 修改折线图的颜色和样式

在前面的例子中，可以在一个图形中同时显示多个语义变量，但这样做并不总是有效的。例如，当你想知道两个变量之间的关系是如何依赖于其他变量的，最好的方法可能是制作结构化网格图，通过多个子图的组合体现更加丰富的语义信息。前文介绍过，因为 relplot()基于 FacetGrid（将在 6.3 节中详细介绍），所以这很容易实现，可以通过设置 relplot()函数的“row”或“col”参数实现。例如：

```
sns.relplot(x="total_bill", y="tip", hue="smoker",
            col="time", data=tips)
```

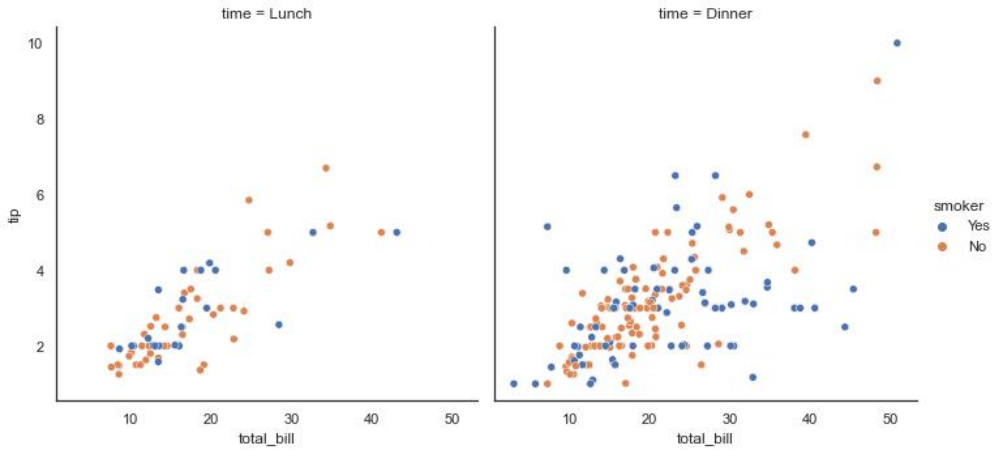


图 6-19 relplot 实现结构化网格图

6.2.2 分布图

任何对数据进行分析或建模之前的基础工作都应该是首先了解数据的分布。例如，“观测数据的覆盖范围是什么”“中心趋势是什么”“是否存在显着异常值”等等。分布图模块包含的子图级函数是 histplot()（直方图）、kdeplot()（核密度图）、ecdfplot()（经验累积分布图）和 rugplot()（轴须图）。它们被整合在图级函数 displot()、jointplot()和 pairplot()中。

直方图是最常见的可视化分布的方法，这是 displot()默认方法，对应的函数是 histplot()。直方图是一个条形图，其中代表数据变量的轴被划分为一组离散的“bin”，并且使用相应

条形的高度显示落入每个“bin”内的数据计数。例如：

```
sns.displot(penguins, x="flipper_length_mm")
```

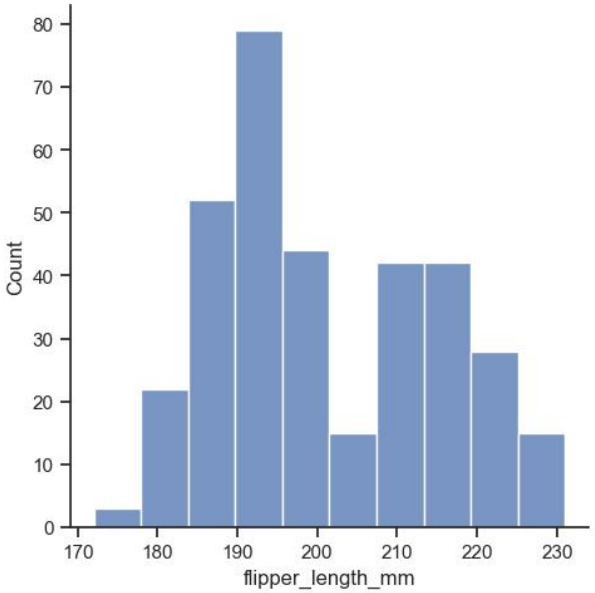
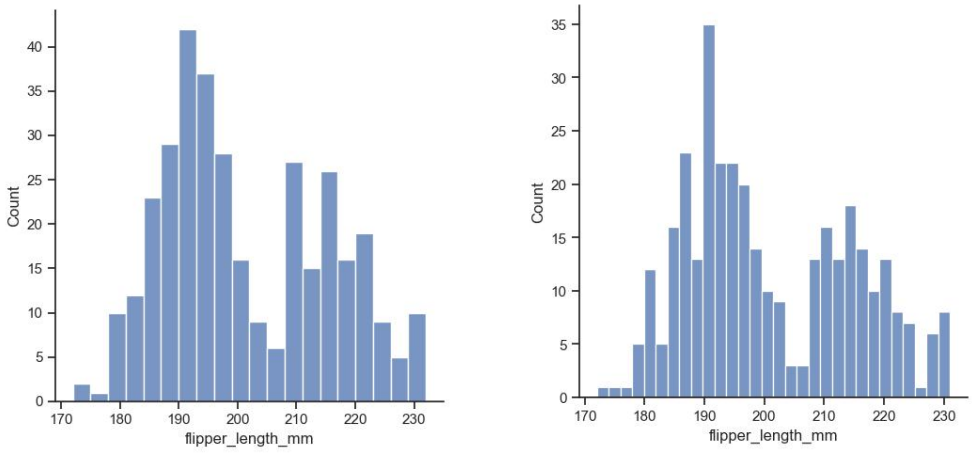


图 6-20 直方图

可以看到最常见的鳍状肢长度约为 195 毫米，但分布呈现双峰，所以这个数字不能很好地代表数据。“bin”的大小是一个重要参数，使用错误的大小可能会产生误导。默认情况下，函数会根据数据的方差和数量选择默认的“bin”大小。但是，在某些情况下直接使用默认值不一定利于观察数据分布。例如，当仅有相对少量的整型数据时，默认的“bin”会很小，图形会产生明显的间隔（“gaps”）。这就需要根据你对数据分布的直观印象动态调整“bin”的大小。可以设置参数“binwidth”选择“bin”的大小，也可以直接设置“bin”的数量。例如：

```
sns.displot(penguins, x="flipper_length_mm", binwidth=3)
sns.displot(penguins, x="flipper_length_mm", bins=30)
```



a. binwidth=3　　b. bins=30

图 6-21 设置“bin”的大小或数量

当了解了数据整体分布情况后，常常还需要进一步探究该分布是否由于数据集中的其他变量而没有充分体现其特性。例如在上例中，企鹅的鳍足长度分布为什么会呈现出双峰特性？通过设置参数“hue”可以进一步了解其成因：“hue”可以实现通过颜色区分变量的类

别。在本例中，设置 hue="species"含义是通过颜色区别企鹅的种类，分别绘制不同种类企鹅的直方图突出不同种类企鹅的鳍足长度分布特性。例如：

```
sns.displot(penguins, x="flipper_length_mm", hue="species")
```

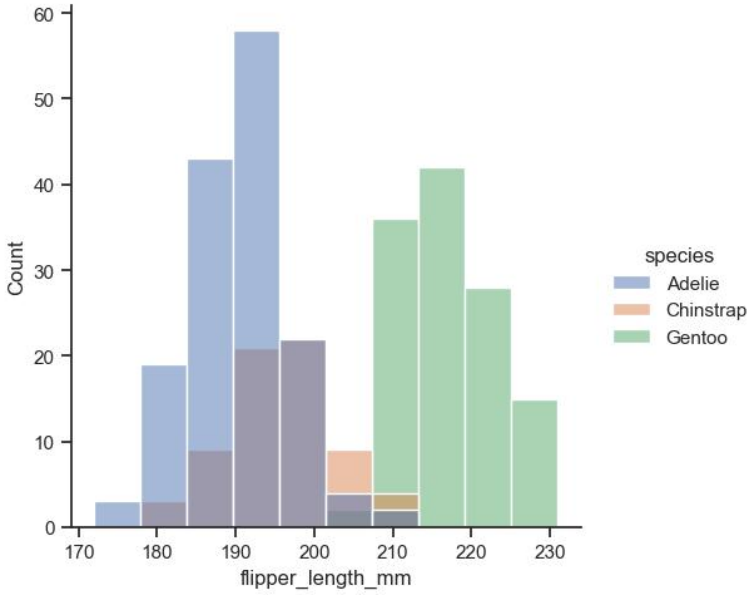
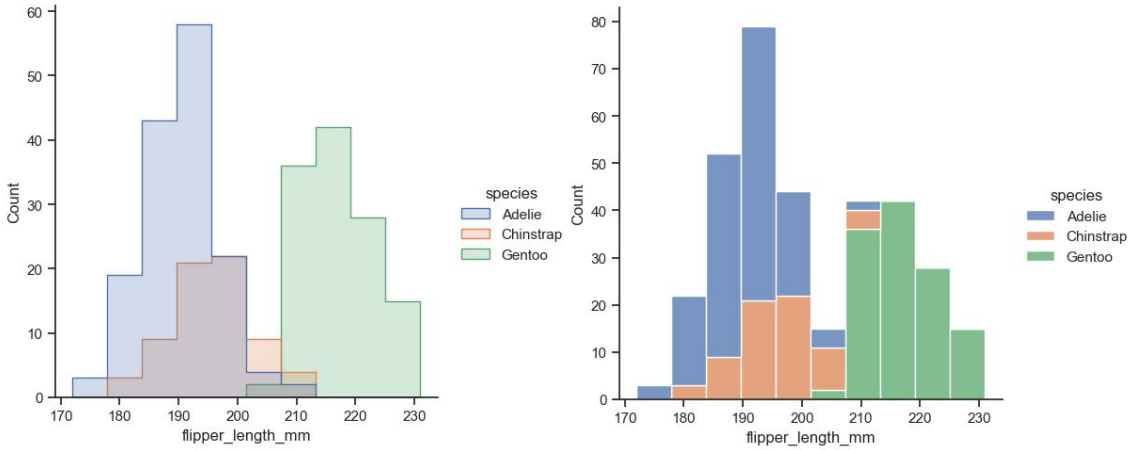


图 6-22 设置“hue”参数区别企鹅的种类

上例中，默认情况下不同的直方图是分层显示的，某些情况下不便观察，可以通过设置 element=“step”（阶梯式）或者 multiple=“stack”（堆叠式）实现不同的显示效果。

```
sns.displot(penguins, x="flipper_length_mm", hue="species",
            element="step")
sns.displot(penguins, x="flipper_length_mm", hue="species",
            multiple="stack")
```



a. 阶梯式　　b. 堆叠式

图 6-23 参数 element 和 multiple 的使用

当数据集的各类别的数据量差异较大时，根据计数比较它们的分布可能并不理想。一种常用的解决方案是通过设置参数“stat”对数据进行标准化。

```
sns.displot(penguins, x="flipper_length_mm", hue="species",
            stat="density",common_norm=False)
```

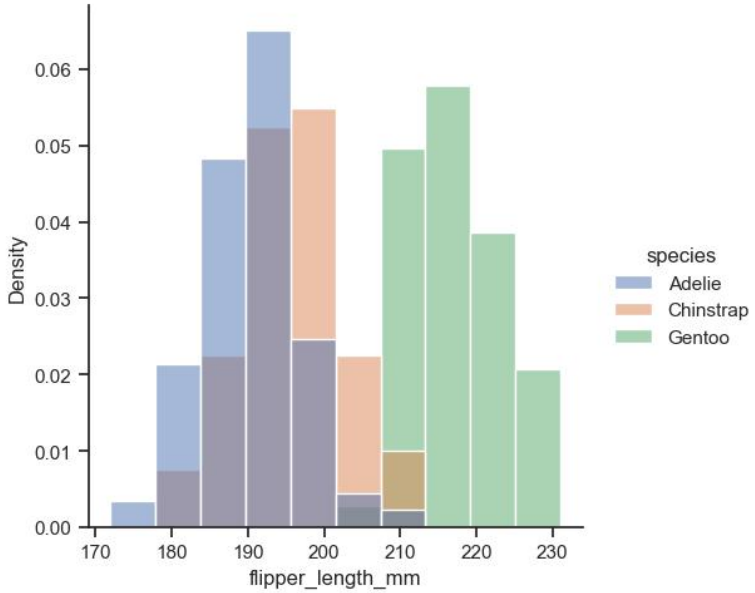


图 6-24 数据标准化 stat="density"

直方图是通过分箱和计数来近似生成数据的潜在概率密度函数。核密度估计(KDE)也为这一问题提供了不同的解决方案。KDE 图不是使用分箱，而是使用高斯核平滑产生连续的密度估计。可以使用 kdeplot()函数或将 displot()函数的 kind 参数设置为“kde”来实现。例如：

```
sns.displot(penguins, x="flipper_length_mm", kind="kde")
```

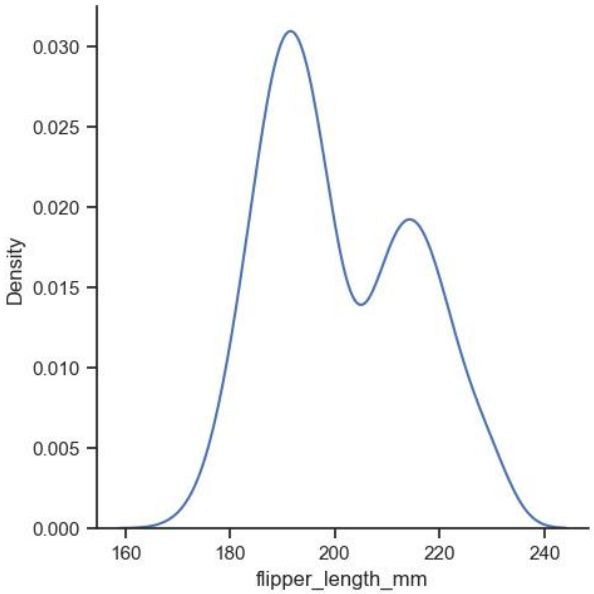
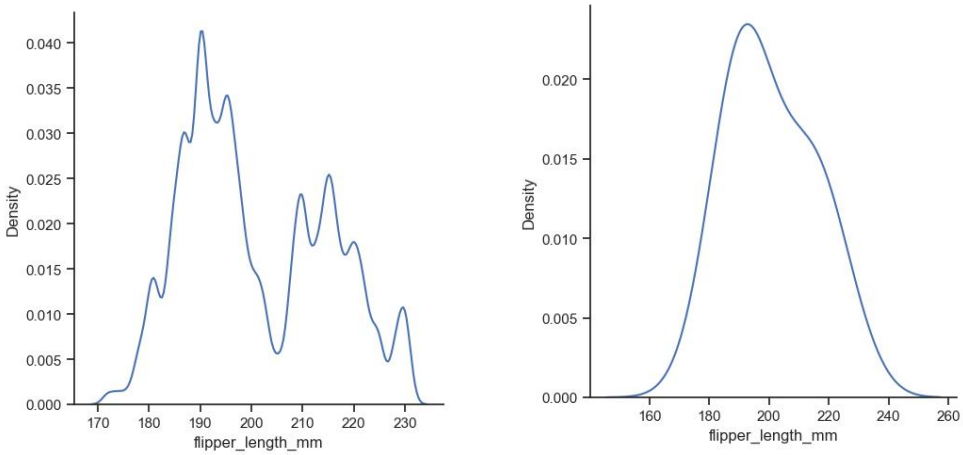


图 6-25 KDE 图

与直方图中的“bin”大小非常相似，KDE 准确表示数据的能力取决于平滑带宽的选择。过度平滑的估计可能会删除有意义的特征，但平滑不足的估计会掩盖随机噪声中的真实形状。检查模型鲁棒性最简单方法是通过设置参数“bw_adjust”调整默认带宽：窄带宽使双峰更加明显，但曲线不太平滑；相反，更大的带宽几乎完全掩盖了双峰。例如：

```
sns.displot(penguins, x="flipper_length_mm", kind="kde",
            bw_adjust=.25)
sns.displot(penguins, x="flipper_length_mm", kind="kde",
```

```
                bw_adjust=2)
```



a. bw_adjust=.25　　b. bw_adjust=2

图 6-26 调整 KDE 图的宽带

与直方图一样，可以通过设置 hue 参数，为每一个不同类别数据分别计算密度估计，并可以设置 fill 参数实现图形颜色填充。

```
sns.displot(penguins, x="flipper_length_mm", hue="species",
            kind="kde", fill=True)
```

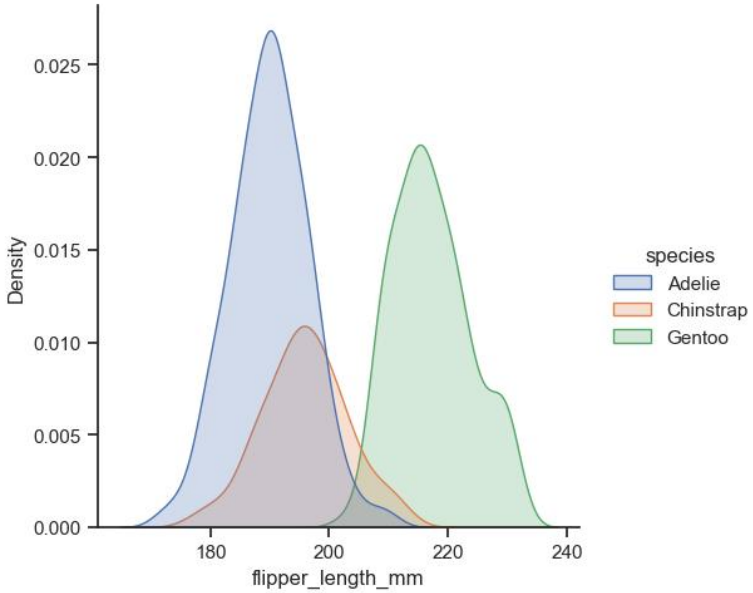


图 6-27 “hue”和“fill”参数的使用

经验累积分布图（ECDF）是通过每个数据点绘制了一条单调递增的曲线，使得曲线的高度反映了具有较小值的观测值的比例。ECDF 图有两个关键优势。与直方图或 KDE 图不同，它直接表示每个数据点。这意味着不需要考虑“bin”的大小或平滑参数。此外，由于曲线是单调递增的，因此非常适合比较多个分布。可以使用 ecdfplot()函数或将 displot()函数的 kind 参数设置为“ecdf”来实现。例如：

```
sns.displot(penguins, x="flipper_length_mm",
            hue="species", kind="ecdf")
```

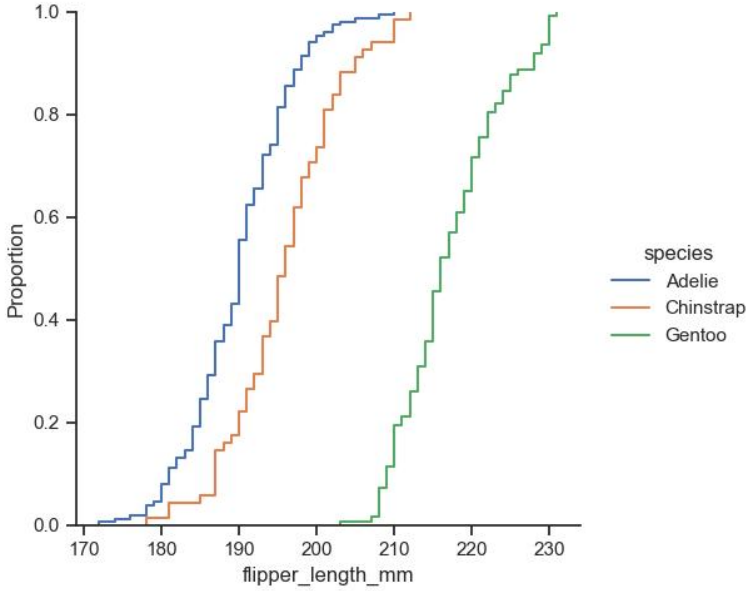


图 6-28 ECDF 图

到目前为止，所有示例都考虑了单变量分布或者仅仅把第二个变量赋给了参数 hue。但是，将第二个变量分配给 y 可以更进一步绘制二元分布（bivariate distribution）。例如。

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm")
```

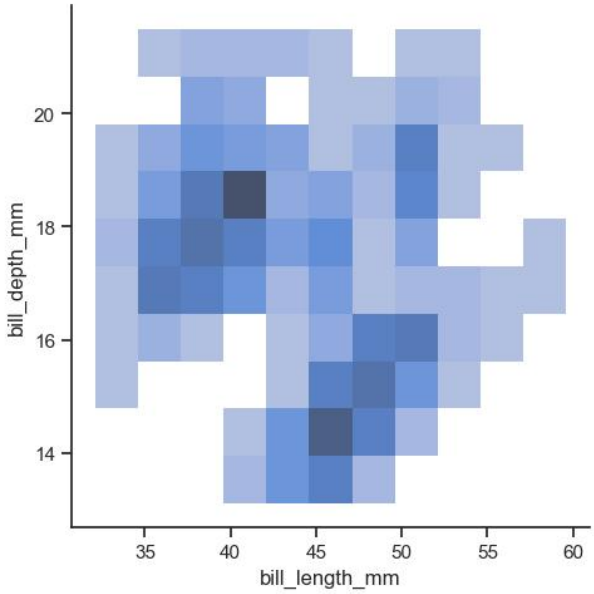
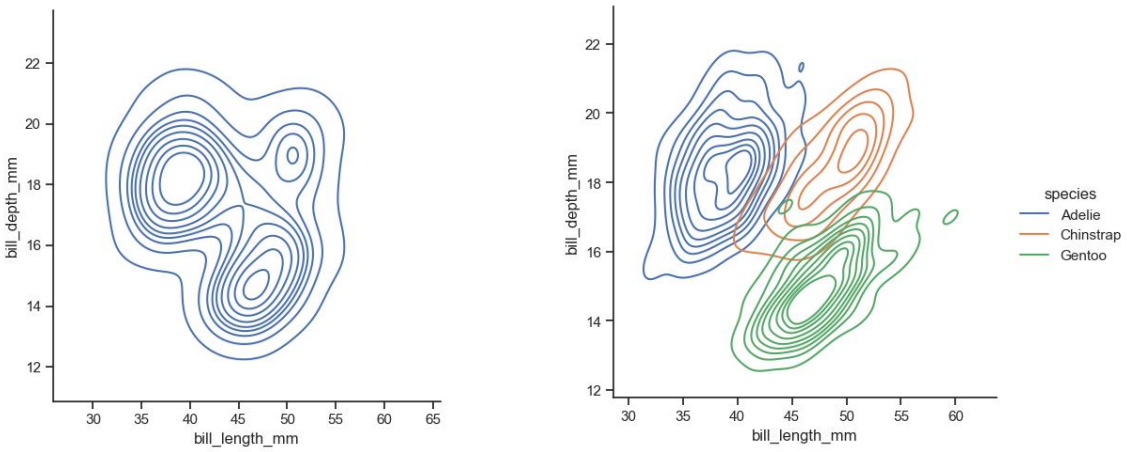


图 6-29 二元直方图

上例中的二元直方图用填充颜色的深浅表示每个矩形内数据的数量（类似于热力图 heatmap）。类似地，二元 KDE 图用 2D 高斯曲线平滑数据(x, y)。默认显示 2D 密度的轮廓。同样，也可以进一步通过参数“hue”绘制多个类别数据的分布。

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
            kind="kde")
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
            hue="species", kind="kde")
```



a. 没有设置 hue 参数　　b. hue=“species”

图 6-30 二元 KDE 图

Seaborn 还提供了另外两个图形级函数 jointplot()（联合分布图）和 pairplot()（分布组图）用直方图或核密度图绘制更加复杂的联合分布和边缘分布。 jointplot()使用两个变量的边际分布扩展二元分布图：默认情况下，使用散点图表示二元分布，使用直方图表示边际分布。例如：

```
sns.jointplot(data=penguins, x="bill_length_mm",
              y="bill_depth_mm")
```

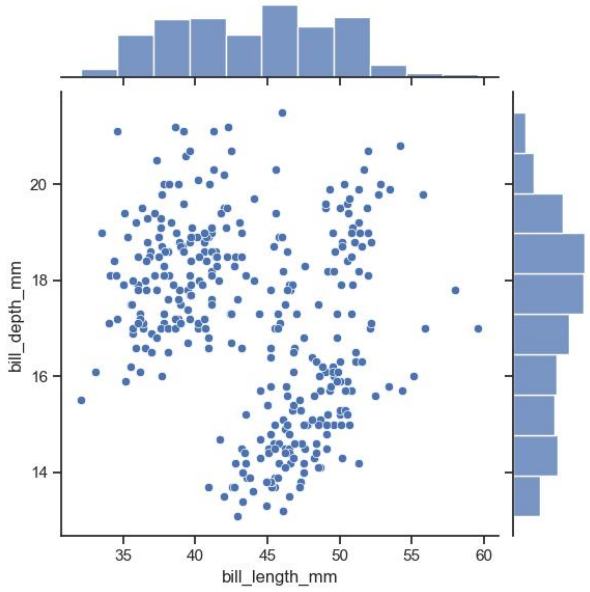


图 6-31 joinplot 基本使用

与 displot()类似，在 jointplot()中设置 kind=“kde”，将使用 KDE 图形表示联合分布和边缘分布。例如：

```
sns.jointplot(data=penguins, x="bill_length_mm",
              y="bill_depth_mm", hue="species", kind="kde")
```

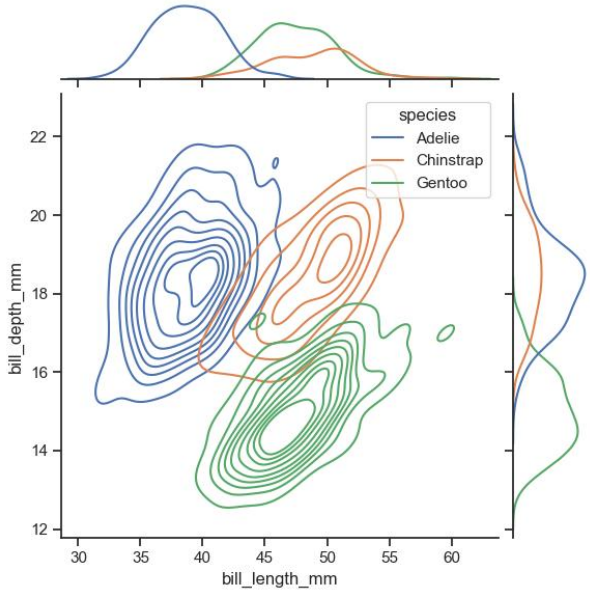


图 6-32 设置 jointplot 图形类别为“kde”

本质上，jointplot()是 JointGrid 类的一个接口，使用很方便。但有时直接使用 JointGrid 能提供了更大的灵活性。下面的例子中，联合分布采用了二元直方图，而边缘分布设置为了盒图。

```
g = sns.JointGrid(data=penguins, x="bill_length_mm",
                  y="bill_depth_mm")
g.plot_joint(sns.histplot)
g.plot_marginals(sns.boxplot)
```

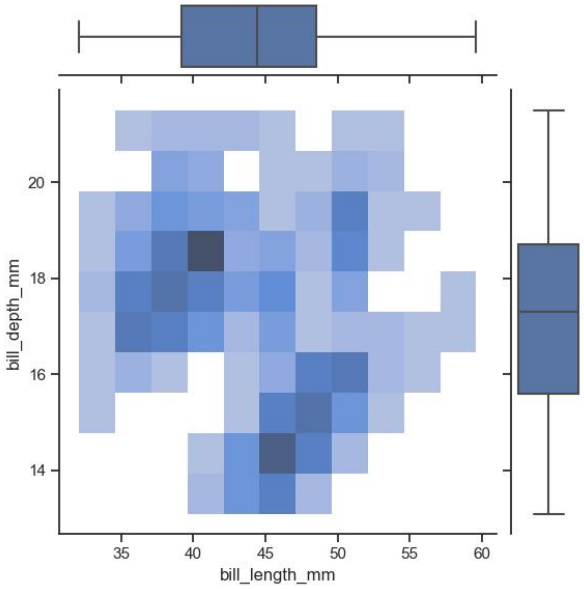


图 6-33 直接使用 JointGrid

实际上，绘制边缘函数更简单的方法是在 displot()函数中设置参数“rug=True”。例如：

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
            kind="kde", rug=True)
```

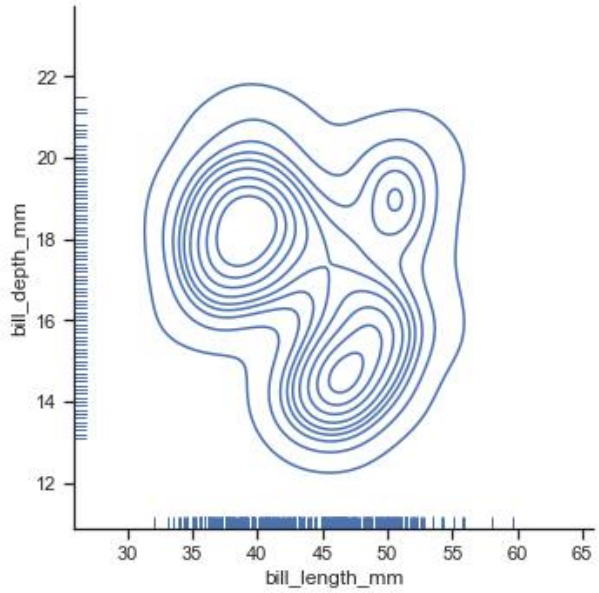
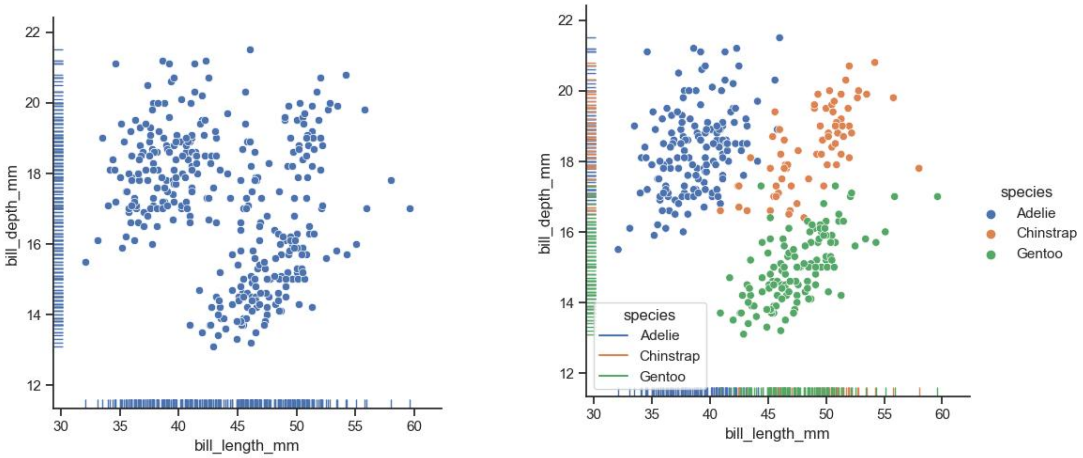


图 6-34 在 displot 函数中设置 rug=True

使用 rugplot()（轴须图）绘制边缘分布更加灵活，可以作为其他图形的补充。同样也可以通过设置参数“hue”实现用不同颜色区分不同类别的数据。例如：

```
sns.relplot(data=penguins, x="bill_length_mm",
            y="bill_depth_mm")
sns.rugplot(data=penguins, x="bill_length_mm",
            y="bill_depth_mm")
sns.relplot(data=penguins, x="bill_length_mm",
            y="bill_depth_mm", hue="species")
sns.rugplot(data=penguins, x="bill_length_mm",
            y="bill_depth_mm", hue="species")
```



a. 没有设置 hue　　b. 设置 hue=“species”

图 6-35 relplot 与 rugplot 组合

Seaborn 还提供了 pariplot()函数可以非常简单的构造一个子图矩阵来实现一次性绘制数据集中单变量分布及变量间两两组合的分布关系。如下图所示。

```
sns.pairplot(penguins)
```

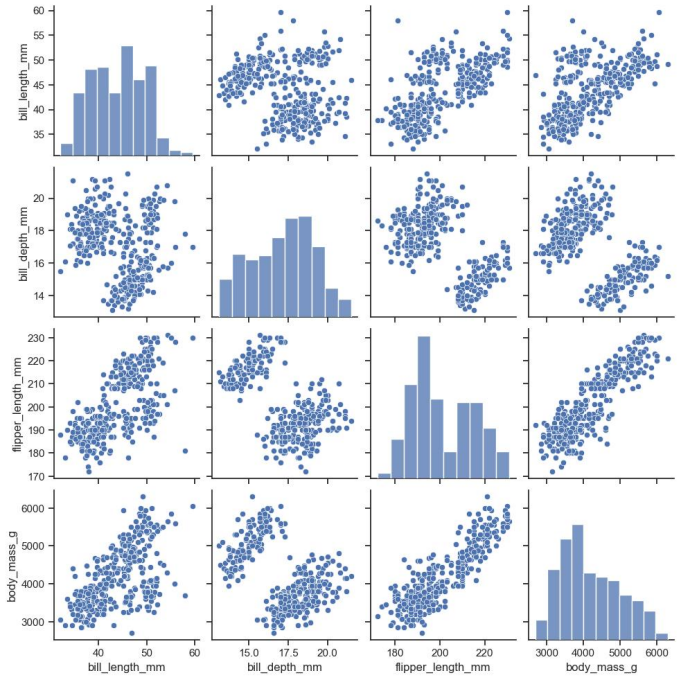


图 6-36 penguins 数据集的 pairplot 绘图

与 jointplot()函数和 JointGrid 类的关系类似，pairplot()函数是 PairGrid 类的一个接口，同样，有时直接使用 PairGrid 能提供了更大的灵活性，可以利用 map_upper()、map_lower()和 map_diag()方法分别设置子图矩阵的上、下三角及对角线上的图形。例如：

```
g = sns.PairGrid(penguins)
g.map_upper(sns.histplot)
g.map_lower(sns.kdeplot, fill=True)
g.map_diag(sns.histplot, kde=True)
```

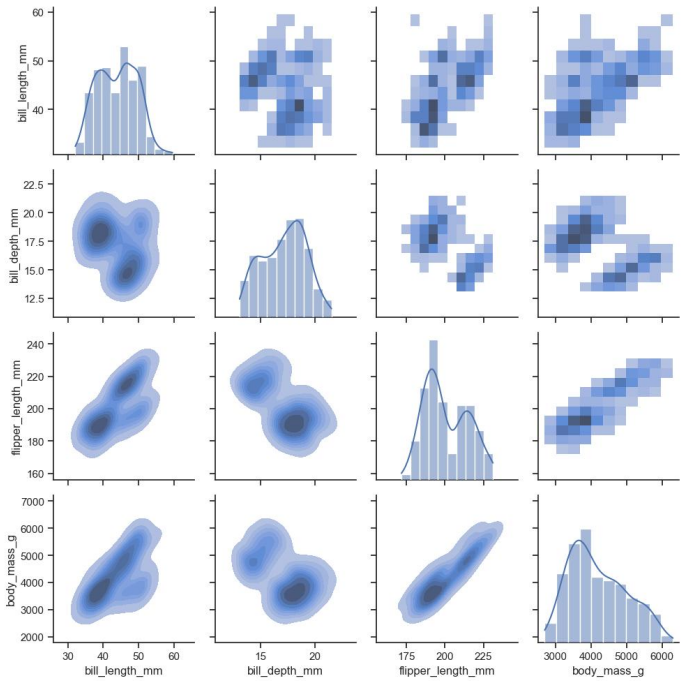


图 6-37 直接使用 PairGrid

6.2.3 分类图

在关系图中，主要学习了如何使用不同的可视化形式来表示数据集中多个变量之间的关系。在这些例子中，主要关注的是两个数值变量之间的关系。在实际应用中，如果其中一个主要变量是“分类的”（将数据分为若干个离散的组），那么使用专门的分类图可能会更加合适。Seaborn 中，catplot()函数提供了图形层次的访问接口，子图层次的接口函数可以分为分类散点图（Categorical scatterplots，主要包含 stripplot()和 swarmplot()）、分类分布图（Categorical distribution plots，主要包含 boxplot()、violinplot()和 boxenplot()）和分类估计图（Categorical estimate plots，主要包含 pointplot()、barplot()和 countplot()）三大类。catplot()是更高层次的接口，可以通过设置 kind 参数实现上述各种绘图函数效果，可选参数值有“strip”、“swarm”、“box”、“violin”、“boxen”、“point”、“bar”和“count”，默认值为“strip”。

下面主要讨论 catplot()实现不同类型绘图的方法。首先是 catplot()采用默认绘图类型的例子：

```
sns.catplot(x="day", y="total_bill", data=tips)
```

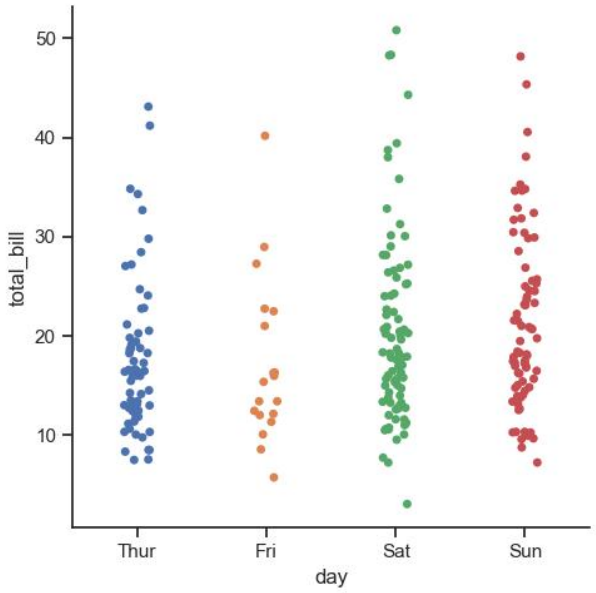


图 6-38 catplot 的基本使用

为了解决同一类别数据会落在坐标轴同一位置从而产生覆盖遮挡的问题，该函数提供了“jitter”参数对数据进行一定的“扰动”，默认值为 True，设置为 Fasle 时效果如图 6-39 所示。

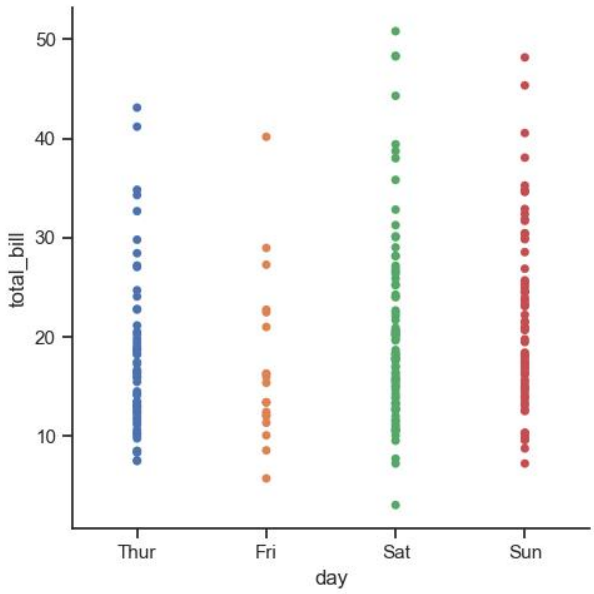


图 6-39 jitter=False 时的效果

解决遮挡问题的另一个方法是使用“swarmplot”图形，可以通过将 catplot()的 kind 参数设置为“swarm”实现。这种方法可以更好的观察数据分布，但只适用于比较小的数据集。例如：

```
sns.catplot(x="day", y="total_bill", kind="swarm", data=tips)
```

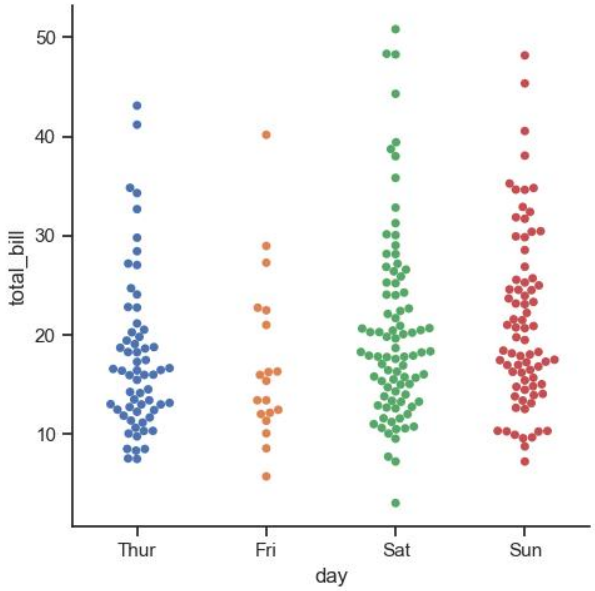


图 6-40 设置参数 kind=“swarm”

与关系图类似，可以通过 hue 参数向分类图中添加另一个维度（分类图目前不支持 size 和 style）。注意，不同的分类绘图函数处理 hue 维度的方式不同，对于散点图，是通过改变点的颜色实现的。例如：

```
sns.catplot(x="day", y="total_bill", hue="sex",
            kind="swarm", data=tips)
```

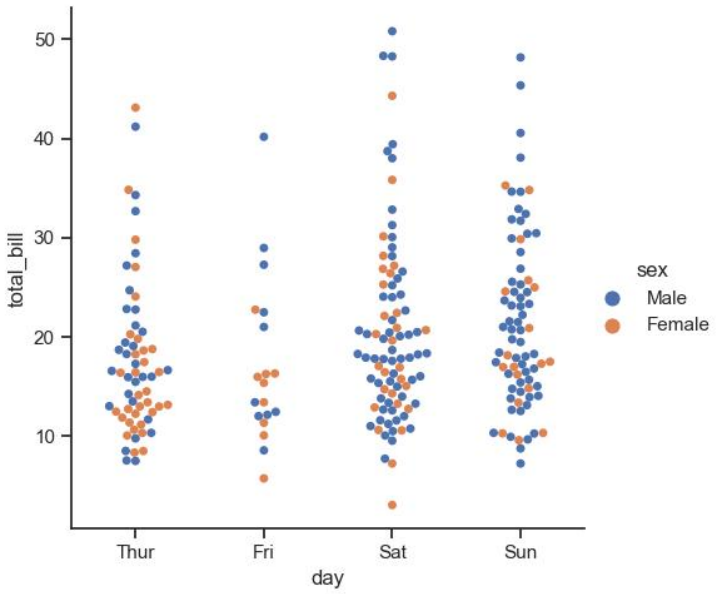
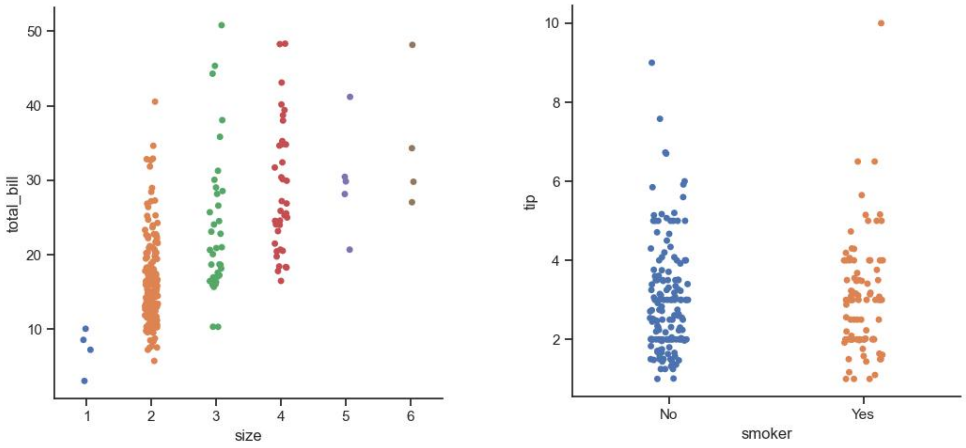


图 6-41 设置 hue="sex"

分类轴上的类别顺序可以通过参数“order”进行指定，否则将按默认排序。例如：

```
sns.catplot(x="size", y="total_bill", data=tips)
sns.catplot(x="smoker", y="tip", order=["No", "Yes"], data=tips)
```



a. 默认类别排序　　b.设置类别排序

图 6-42 分类轴上的类别排序

上述的例子中，是以横轴作为分类轴的，但当类别名称相对较长或类别很多时，以纵轴作为分类轴更加合理。例如：

```
sns.catplot(x="total_bill", y="day", hue="time",
            kind="swarm", data=tips)
```

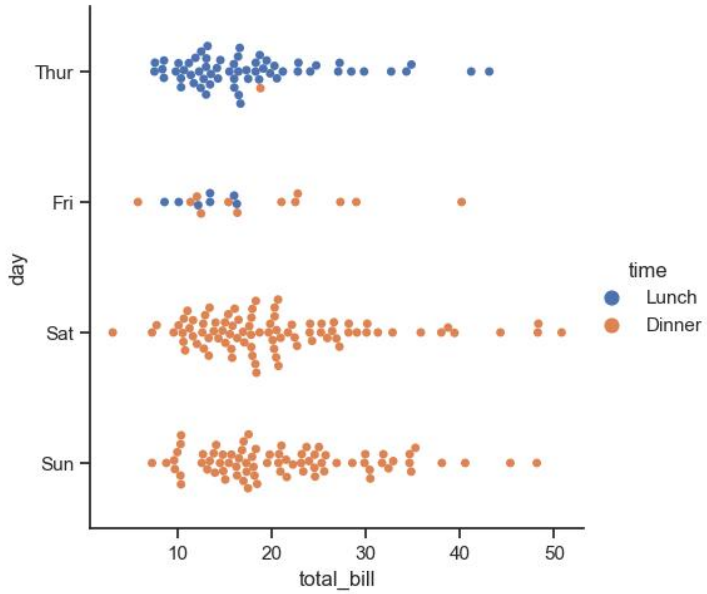


图 6-43 纵轴作为分类轴

随着数据集规模的增长，散点图无法有效的提供每个类别内部的数据分布情况。可以通过 boxplot()（盒图）和 violinplot（小提琴图）实现。

boxplot()可以显示数据分布的三个四分位数及极值。如下图所示。

```
sns.catplot(x="day", y="total_bill", kind="box", data=tips)
```

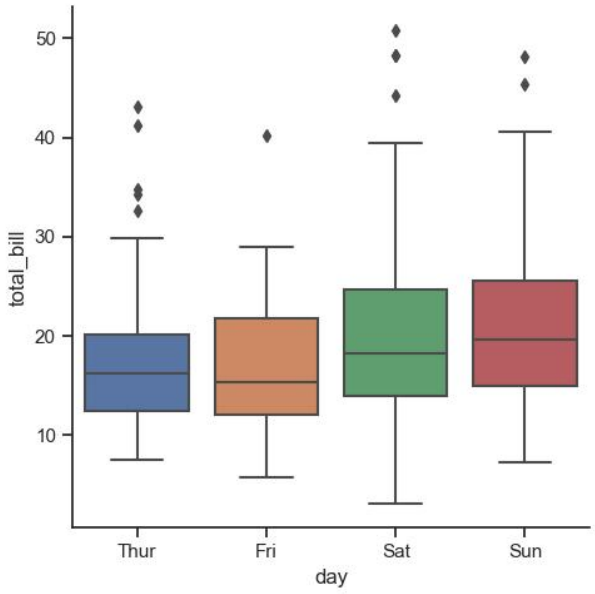


图 6-44 盒图示例

同样可以设置参数“hue”引入另一个维度，每个图框不会重叠，这称之为“避让”，默认情况下是打开的，因为一般假设“hue”引入的维度是嵌入在主分类变量中的。例如：

```
sns.catplot(x="day", y="total_bill", hue="smoker",
            kind="box", data=tips)
```

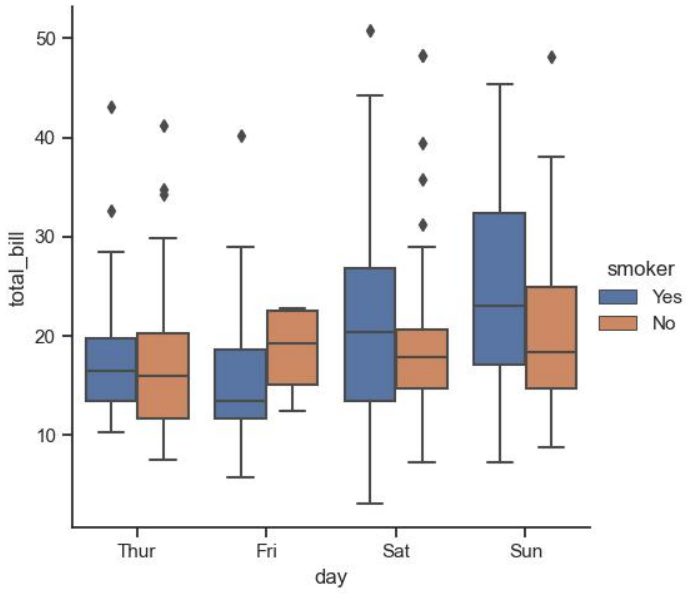


图 6-45 盒图中参数“hue”的使用

如果不是这种情况，可以通过设置参数“dodge=False”关闭自动避让，在下面的例子中，hue 变量“weekend”并没有包含在主分类变量“day”中。

```
tips["weekend"] = tips["day"].isin(["Sat", "Sun"])
sns.catplot(x="day", y="total_bill", hue="weekend",
            kind="box",dodge=False, data=tips)
```

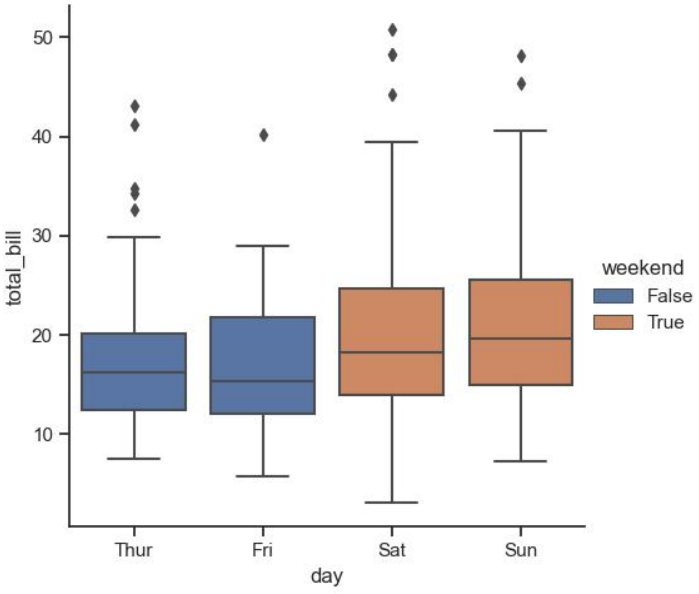


图 6-46 dodge=False 的示例

此外，Seaborn 还提供了一种适合于较大数据集的盒图函数 boxenplot()，在标准盒图的基础上增加了更多的分位数信息，可以提供更丰富的分布信息，绘图效果更为美观，因此 boxenplot()可以看做是增强版的盒图。例如：

```
sns.catplot(x="color", y="price", kind="boxen",
            data=diamonds.sort_values("color"))
```

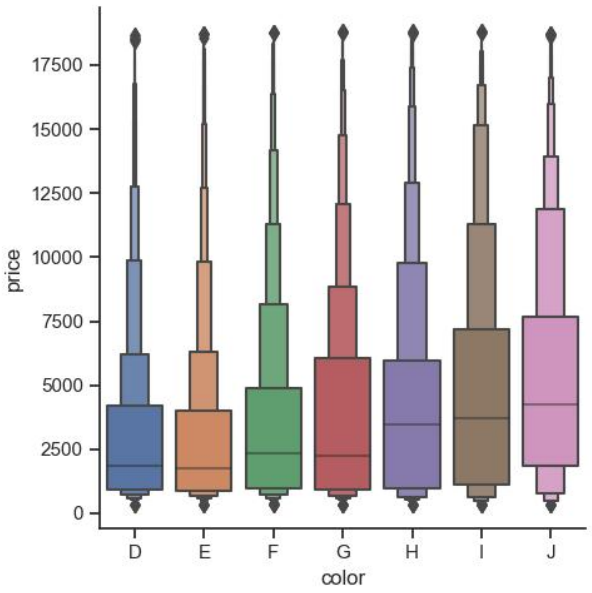


图 6-47 增强版盒图 boxenplot

violinplot()在标准盒图的基础上增加了 KDE 图的信息，可更为直观的查看数据分布情况。例如：

```
sns.violinplot(x="day",y="total_bill",hue="sex",data=tips)
```

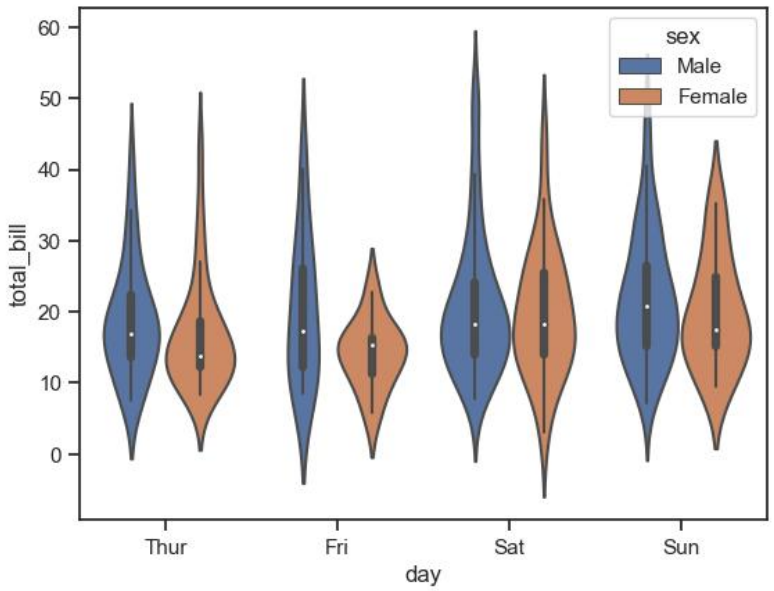


图 6-48 小提琴图（violinplot）示例

在参数“hue”仅有 2 个取值时，还可通过设置参数“split=True”实现左右数据合并显示。例如：

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

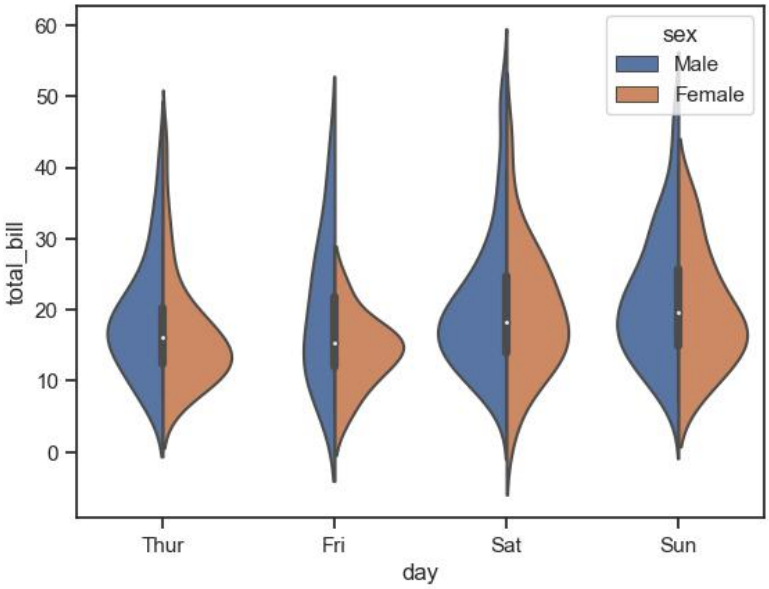


图 6-49 参数 split 的使用

此外，在实际应用中，经常将 swarmplot()或 striplot()与小提琴图结合在一起显示数据集中每个数据个体及总体分布。例如：

```
g = sns.catplot(x="day", y="total_bill", kind="violin",
                inner=None, data=tips)
sns.swarmplot(x="day", y="total_bill", color="k", size=3,
              data=tips, ax=g.ax)
```

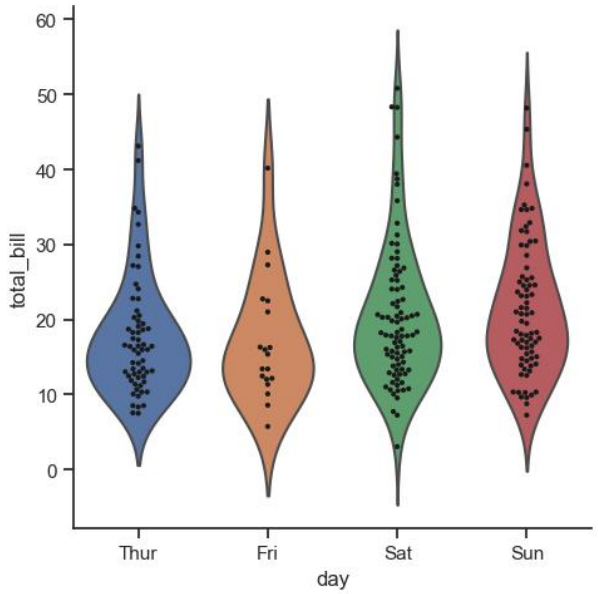


图 6-50 violinplot 与 swarmplot 结合

Seaborn 提供了 barplot()（柱状图）、countplot()（计数图）、pointplot()（点图）函数实现对数据集中趋势的估计。catplot()函数中对应 kind 参数值分别为“bar”、“count”和“point”。

柱状图（barplot）是以矩形条来表示统计量（平均值、比例、中位数等），通过误差线体现置信区间和置信区间。矩形条的长短及变化反映了数据的集中趋势。例如：

```
sns.catplot(x="sex", y="survived", hue="class",
            kind="bar", data=titanic)
```

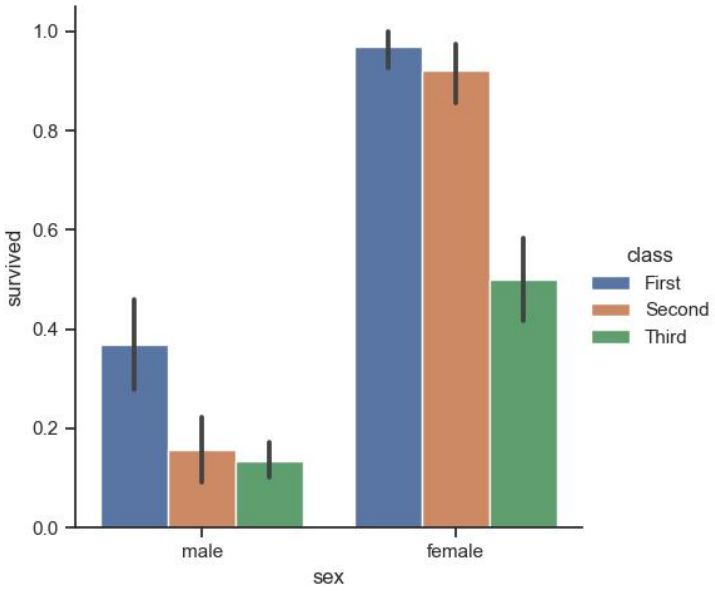


图 6-51 条形图（barplot）的基本使用

计数图（countplot）实际上是柱状图的一种特例，主要用于表示各分类的数据数量，并以柱状图的形式展现。例如：

```
sns.catplot(x="deck", kind="count",
            palette="ch:.25", data=titanic)
```

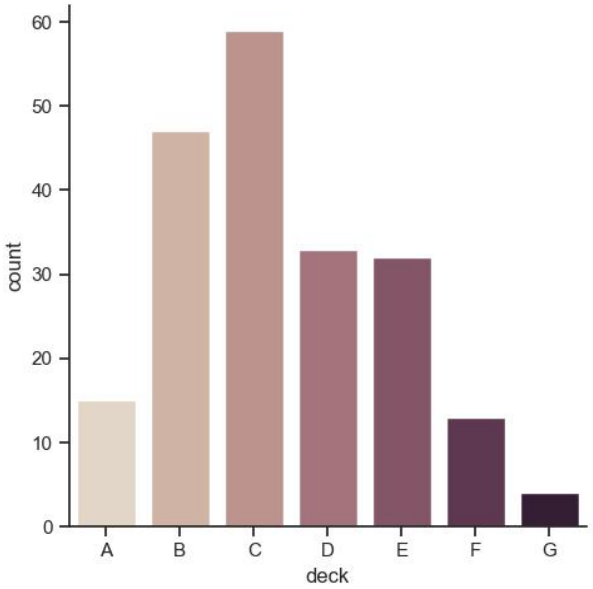
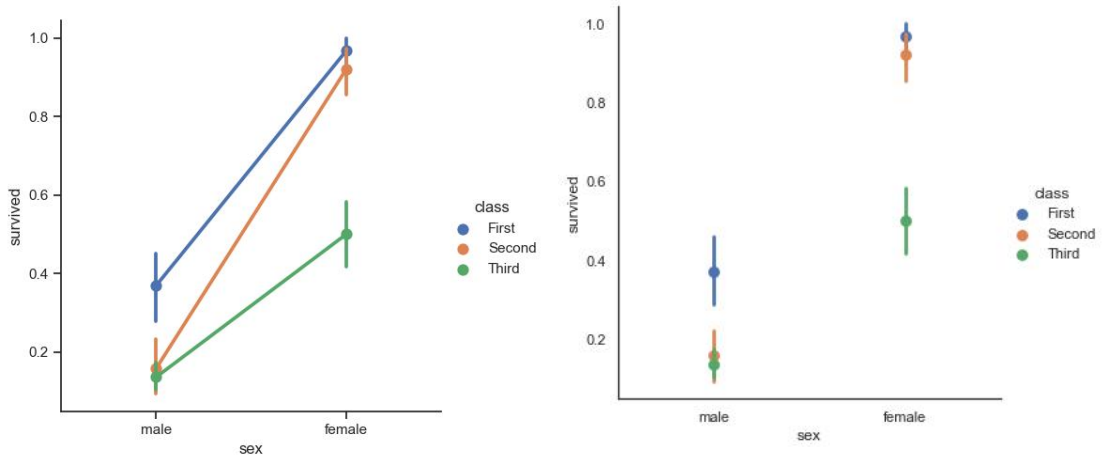


图 6-52 计数图（countplot）的基本使用

点图（pointplot）提供了另一种可视化形式，展示数据点的估计值（默认平均值）和置信区间，并连接来自同一 hue 类别的点，通过设置参数 join=False 可以取消连线。例如：

```
sns.catplot(x="sex", y="survived", hue="class",
            kind="point", data=titanic)
```



a. 默认 join=True　　b. join=False

图 6-53 点图（pointplot）的基本使用

虽然，分类图没有关系图的 style 设置，但是仍然可以通过 markers 和 linestyle 参数实现更加丰富的图形绘制。例如：

```
sns.catplot(x="class", y="survived", hue="sex",
            palette={"male": "g", "female": "m"},
            markers=["^", "o"], linestyles=["-", "--"],
            kind="point", data=titanic)
```

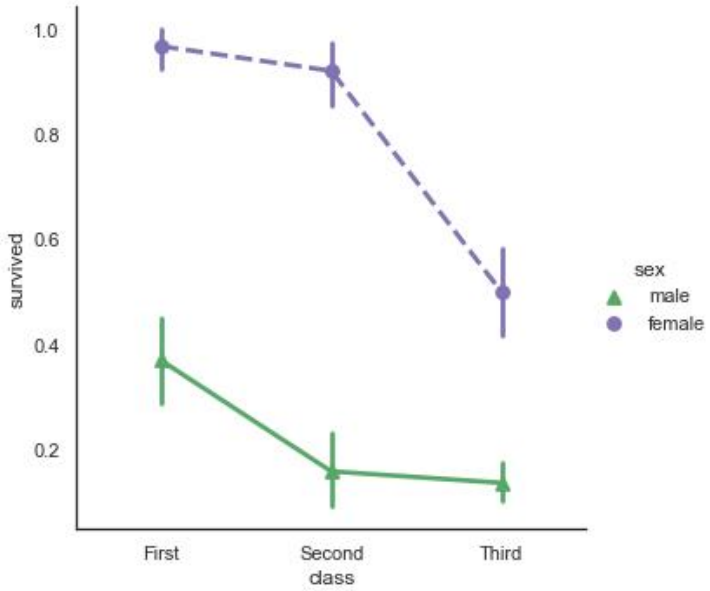


图 6-54 参数 markers 和 linestyle 的使用

与 relplot()一样，catplot()是基于 FacetGrid 的，因此很容易引入其他维度的变量来实现高维关系的可视化，通过构建多个子图的组合体来表示更加丰富的语义信息。例如：

```
sns.catplot(x="day", y="total_bill", hue="smoker",
            col="time", aspect=.7, kind="swarm", data=tips)
```

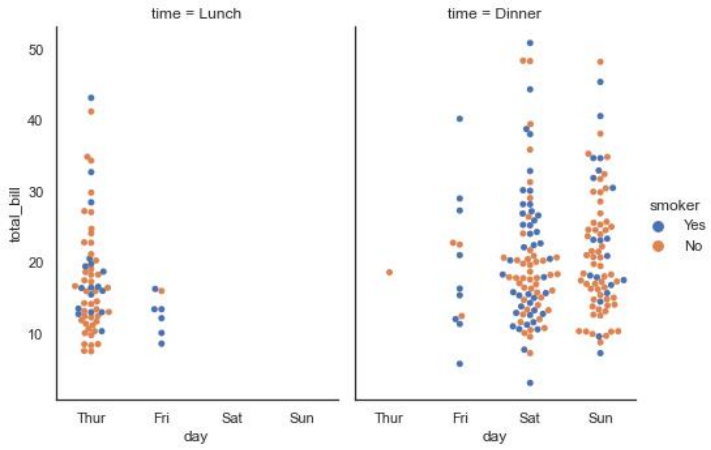


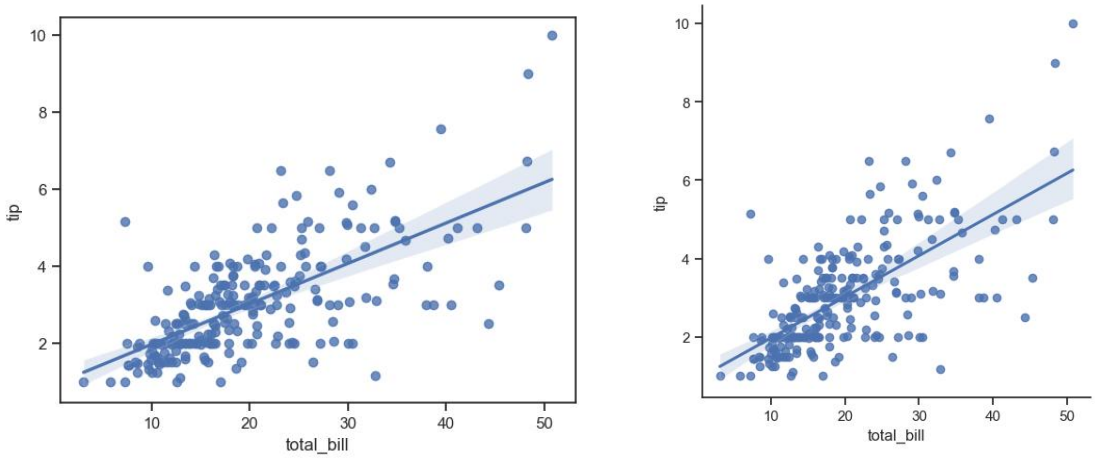
图 6-55 catplot 实现结构化网格图

6.2.4 回归图

许多数据集包含多个数值型变量，分析的目标通常是探究这些变量之间的相互关系。前文介绍了可以通过显示两个变量的联合分布来实现这一点，但是，有时使用统计模型来估计两组包含噪声的数据之间的关系更有优势，本节将讨论如何通过线性回归来实现。Seaborn 中提供了两个绘制线性回归模型的函数 regplot()和 lmplot()以及一个残差函数 residplot()。

regplot()和 lmplot()非常相似，它们共享大部分核心功能。在最简单的调用中，两个函数都绘制了两个变量 x 和 y 的散点图，然后拟合回归模型，绘制结果回归线和该回归的 95%置信区间。例如：

```
sns.regplot(x="total_bill", y="tip", data=tips)
sns.lmplot(x="total_bill", y="tip", data=tips)
```



a. regplot 绘制的图形　　b.lmplot 绘制的图形

图 6-56regplot 和 lmplot 的基本使用

可以看到，除了图形部分细节有所差异以外，绘制的图形是相同的。两者的主要区别是，regplot()接受各种格式的 x 和 y 变量，包括简单的 Numpy 数组、Pandas Series 对象，或者传递给 data 的 pandas DataFrame 对象的引用。相反，lmplot()将 data 作为必需的参数，x 和 y 变量必须指定为字符串，这种数据格式被称为“长格式”数据。

在实际应用中，有时其中一个变量取离散值（如下例中的 size 表示人数），虽然仍然可以拟合数据，但是，这种拟合结果看起来往往不是那么直观。例如：

```
sns.lmplot(x="size", y="tip", data=tips)
```

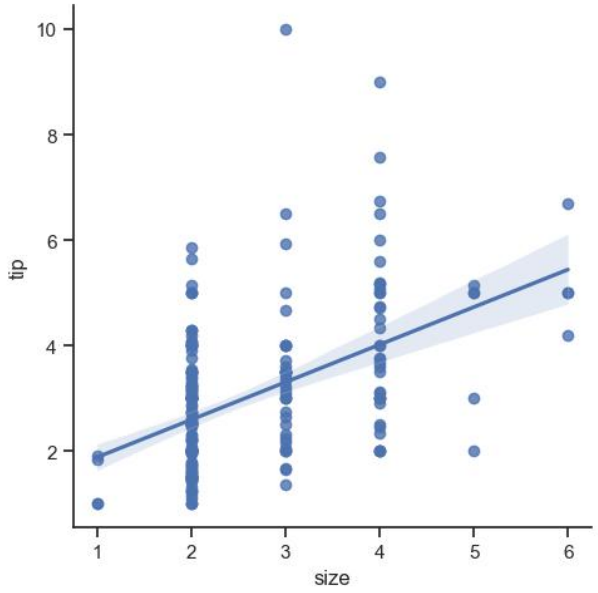


图 6-57 一个变量为离散型的情况

一个简单解决办法是对每个类别中的数据计算其估计值（通常是平均值或中位数），从而绘制出数据的集中趋势以及置信区间。例如：

```
sns.lmplot(x="size", y="tip", data=tips, x_estimator=np.mean)
```

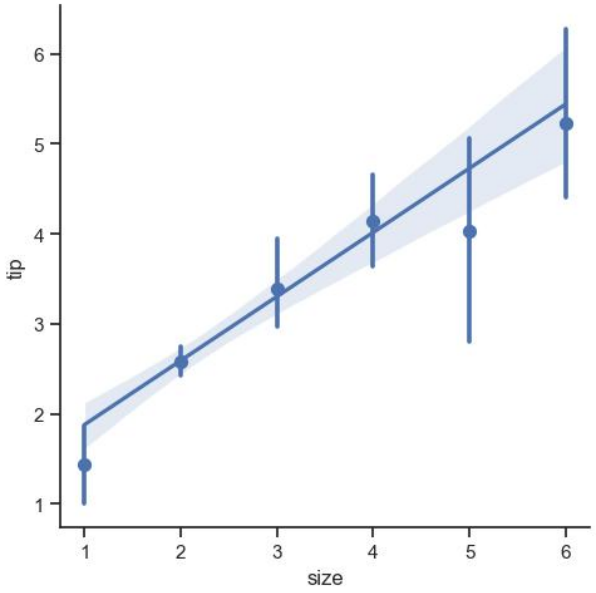
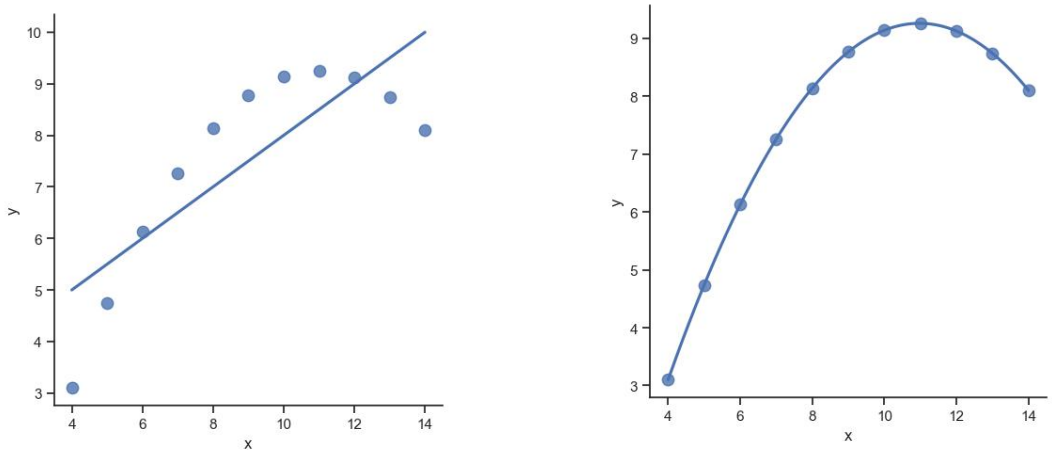


图 6-58 参数 x_estimator 的使用

此外，lmplot()和 regplot()还可以通过设置“order”参数拟合一个多项式回归模型探索数据集中存在的高阶非线性关系。例如：

```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
           ci=None, scatter_kws={"s": 80})
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
           order=2, ci=None,scatter_kws={"s": 80})
```

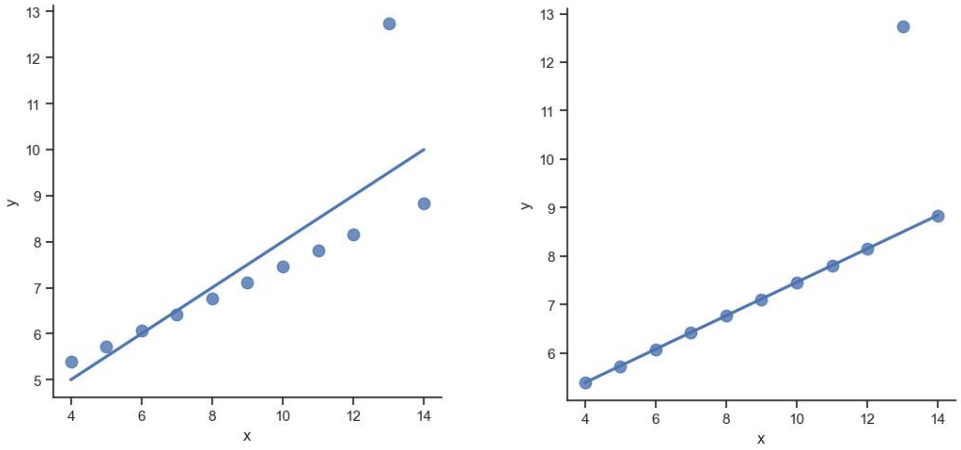


a. 线性模型　　b. 非线性模型

图 6-59 参数 order 的使用

数据集中存在离群点时会对归回模型造成较大影响，可以通过设置“robust=True”来拟合鲁棒回归模型，它使用不同的损失函数来降低相对较大的残差。例如：

```
sns.lmplot(x="x",y="y", data=anscombe.query("dataset == 'III'"),
           ci=None, scatter_kws={"s": 80})
sns.lmplot(x="x",y="y", data=anscombe.query("dataset == 'III'"),
           robust=True, ci=None, scatter_kws={"s": 80})
```

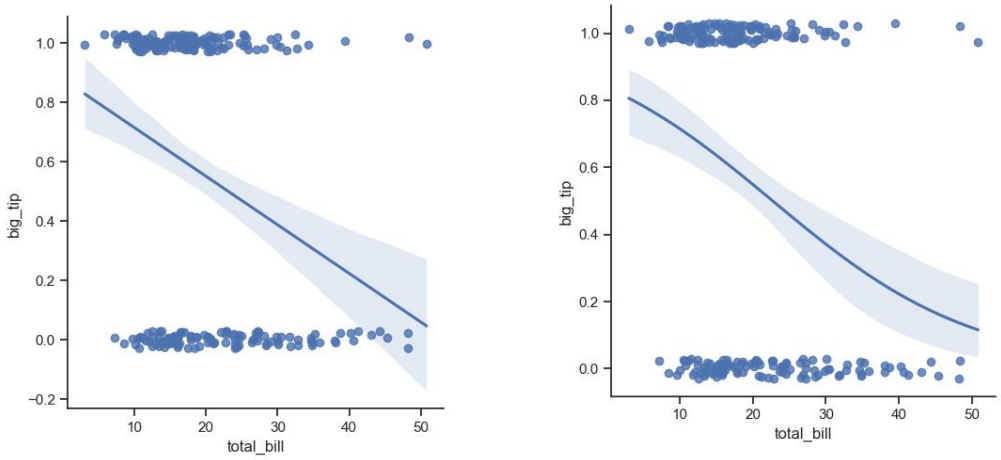


a. 非鲁棒回归　　b. 鲁棒回归

图 6-60 离群点的处理

当变量 y 是二元变量（例如只能取 0 或 1），仍然可以构建简单的回归模型，但结果没有太多的实际意义。这种情况下的解决方案是拟合逻辑回归，这样回归线显示了给定 x 值 y = 1 的估计概率。例如：

```
tips["big_tip"] = (tips.tip / tips.total_bill) > .15
sns.lmplot(x="total_bill", y="big_tip",
           data=tips, y_jitter=.03)
sns.lmplot(x="total_bill", y="big_tip", data=tips,
           logistic=True, y_jitter=.03)
```



a. 线性回归　　b. 逻辑回归

图 6-61 参数 logistic 的使用

注意，与简单回归相比，逻辑回归的计算量要大得多（鲁棒回归也是如此），可以关闭回归线周围的置信区间计算（使用 ci=None）实现更快地迭代。

残差函数 residplot()是一个检查简单回归模型是否适用于数据集的有用工具，它主要用于绘制数据残差的散点图。理想情况下，这些值应该随机分布在 y=0 的周围，并且没有明显模式。例如：

```
sns.residplot(x="x",y="y",data=anscombe.query("dataset== 'I'"),
              scatter_kws={"s": 80})
```

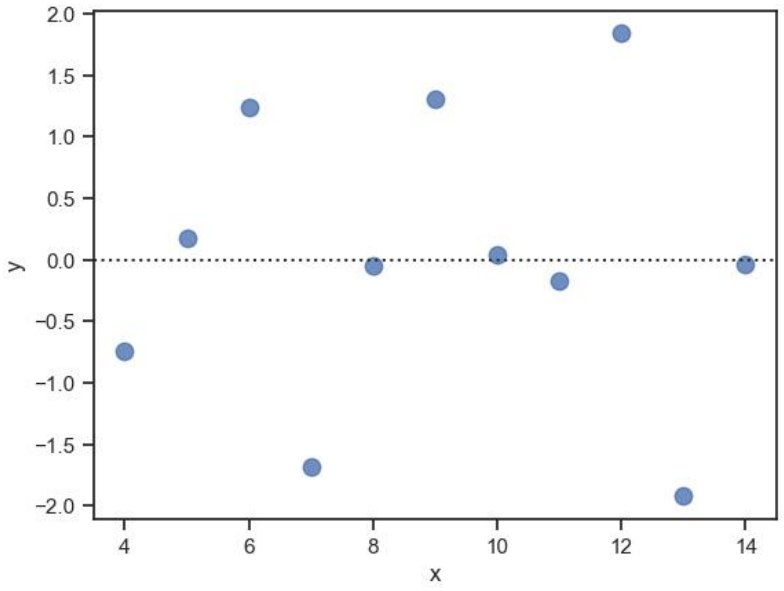


图 6-62 有效模型的残差图

如果残差有结构，说明简单线性回归是不合适的。例如：

```
sns.residplot(x="x", y="y",
              data=anscombe.query("dataset=='II'"),
              scatter_kws={"s": 80})
```

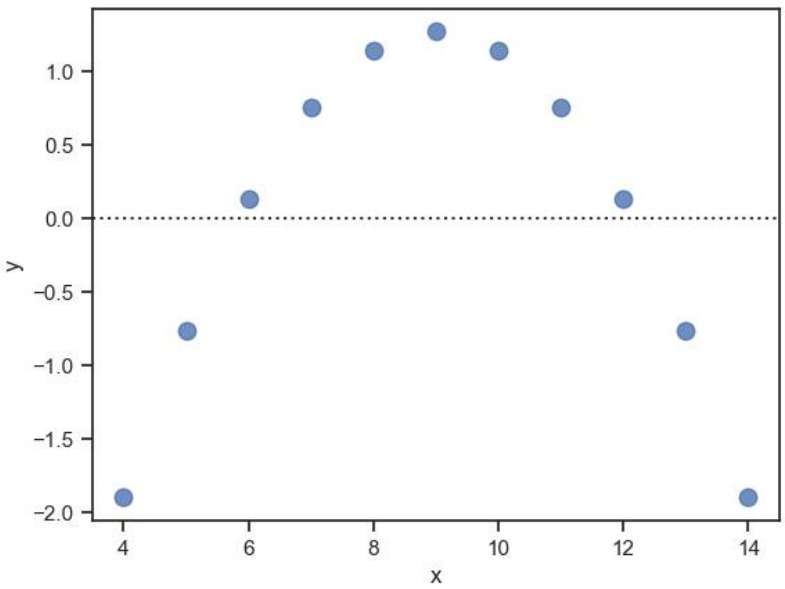


图 6-63 无效模型的残差图

上面的例子展示了表示一对变量之间关系的多种方法。然而，一个更复杂的问题是“这两个变量之间的关系如何随着第三个变量的变化而变化”，这就是 regplot()和 lmplot()的区别。regplot()总是显示单个关系，而 lmplot()是将 regplot()与 FacetGrid 结合在一起，提供了一个使用简单的接口实现在网格图上显示线性回归，最多支持三个额外类别变量的交互。

把变量关系分离出来的最好方法是在相同的坐标轴上通过颜色或标记样式区分不同的层次。例如：

```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
           markers=["o", "x"], palette="Set1")
```

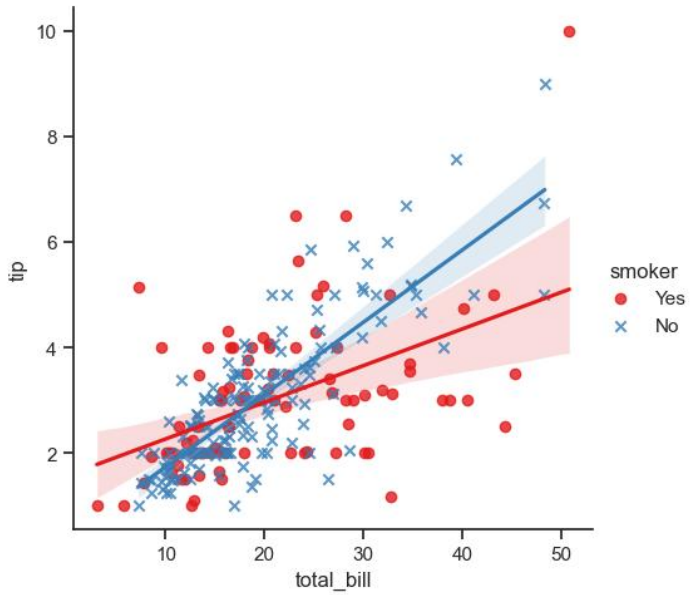


图 6-64 通过颜色及样式区分关系

如果增加更多的变量，lmplot()可以通过 FacetGrid 实现多个子图组合的效果。例如：

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

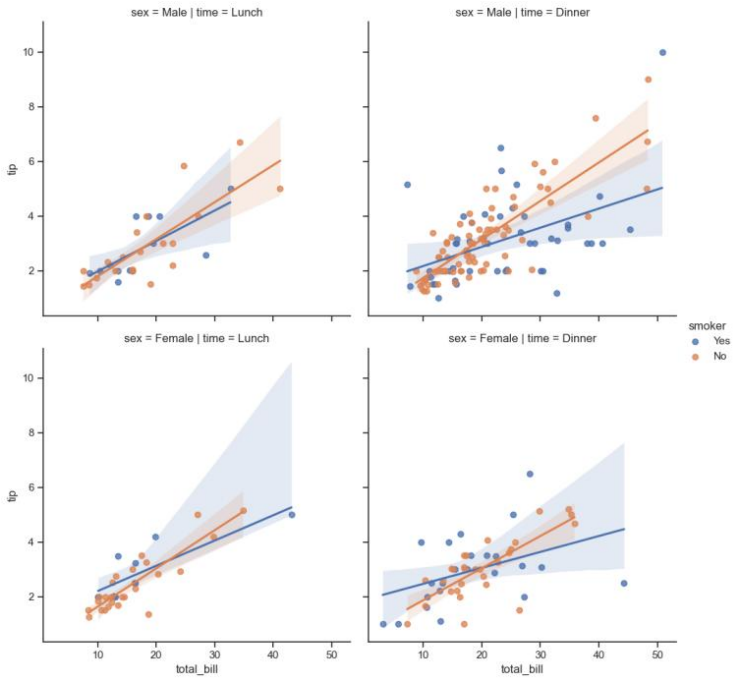


图 6-65 lmplot 实现网格图

还有一些函数可以在更复杂的组合图形中使用线性回归。例如之前介绍过的 jointplot()函数，除了前面讨论过的样式外，jointplot()可以通过设置参数 kind=“reg”来实现数据的线性回归。例如：

```
sns.jointplot(x="total_bill", y="tip",
              data=tips, kind="reg")
```

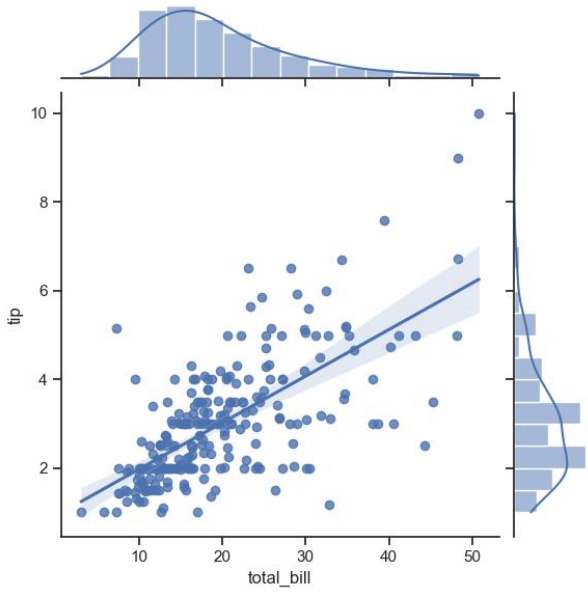


图 6-66 jointplot 实现线性回归

pairplot()函数的参数“kind”设置为“reg”本质上是 regplot()和 PairGrid 的结合，能够显示数据集中成对变量之间的线性关系。x_vars、y_vars 是两个非常重要的参数，表示分别用于设置图形行和列的数据集中的变量名。例如：

```
sns.pairplot(tips, x_vars=["total_bill", "size"],
             y_vars=["tip"],hue="smoker", height=5,
```

```
aspect=.8, kind="reg")
```

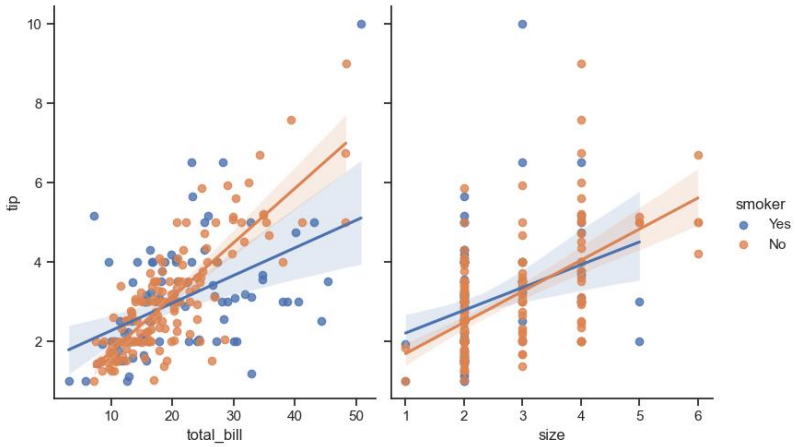


图 6-67 pairplot 实现线性回归

6.2.5 矩阵图

Seaborn 的矩阵图提供了 heatmap()（热力图）和 clustermap()（分层聚类热力图）两个绘图函数。其中热力图是将数据绘制为颜色编码矩阵；分层聚类热图用于绘制分层聚类的热力图。heatmap()基本绘制方法如下：

```
uniform_data = np.random.rand(10, 12)
ax = sns.heatmap(uniform_data)
```

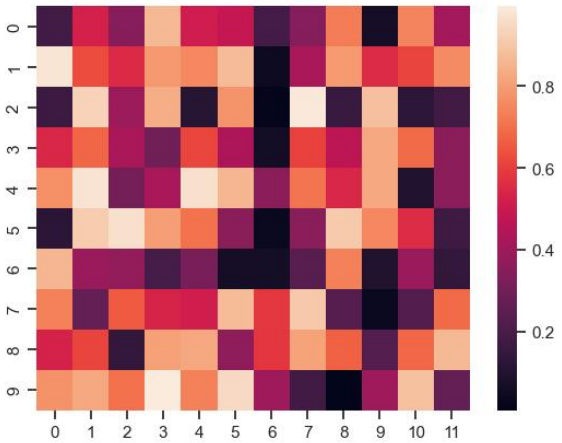


图 6-68 heatmap 的基本使用

可以设置 vmin、vmax 参数进行颜色范围的限制，含义是数据小于等于 vmin 的以 vmin 代表的颜色显示，大于等于 vmax 的则以 vmax 代表的颜色显示。例如：

```
ax = sns.heatmap(uniform_data, vmin=0, vmax=0.6)
```

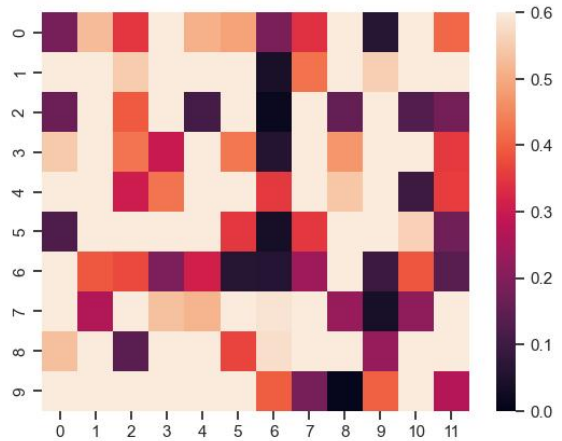


图 6-69 vmin、vmax 参数的使用

参数“center”可以指定颜色的中心值，通过该参数可以调整热力图的颜色深浅。例如绘制以 0 为中心的热力图：

```
normal_data = np.random.randn(10, 12)
ax = sns.heatmap(normal_data, center=0)
```

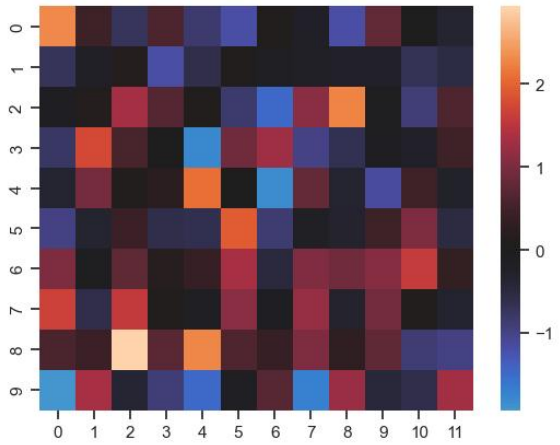


图 6-70 以 0 为中心的热力图

利用 pandas.DataFrame.pivot 方法可以按照行、列和值对 DataFrame 进行重构，就能够绘制行和列标签具有实际意义的热力图。例如：

```
flights = flights.pivot("month", "year", "passengers")
ax = sns.heatmap(flights)
```

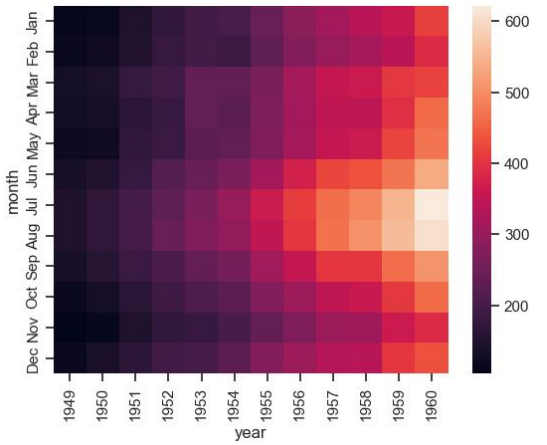


图 6-71 行、列标签具有实际意义的热力图

还可以在每个单元格中标注数据，并通过设置参数 linewidths 在每个单元格之间添加分割线，使得热力图信息更加丰富和清晰美观。例如：

```
ax = sns.heatmap(flights, annot=True, fmt="d",linewidths=.5)
```

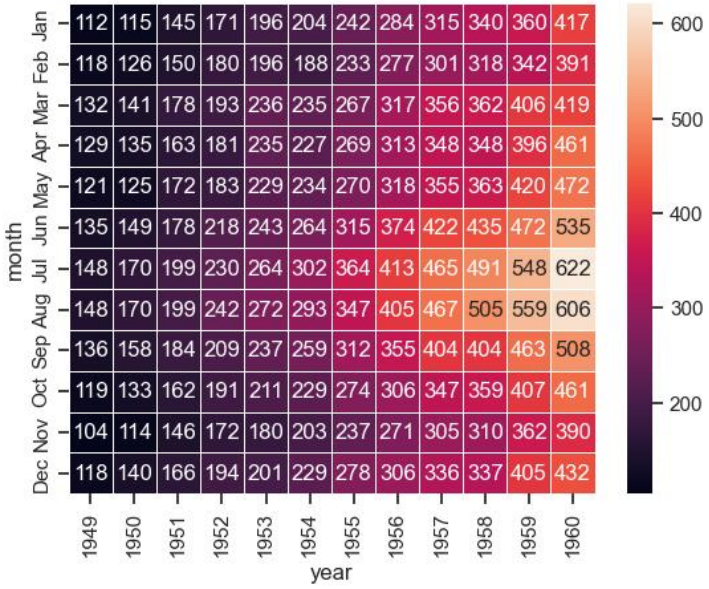


图 6-72 标注数据并增加分割线

此外，还可以对热力图的外观进行一些设置，如使用不同的颜色图，隐藏颜色条等。

```
ax = sns.heatmap(flights, cmap="YlGnBu",
                 linewidths=.5,cbar=False)
```

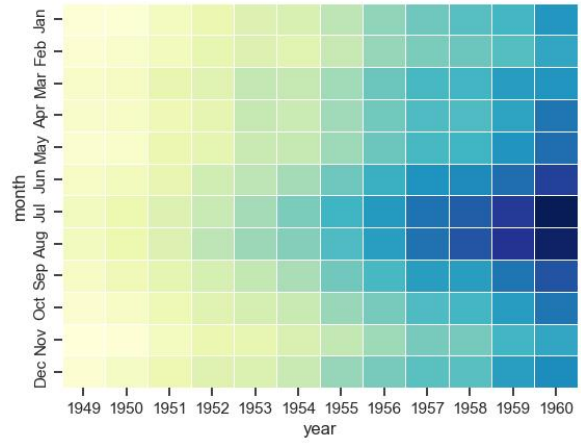


图 6-73 设置颜色图并隐藏颜色条

clustermap()函数可以对变量根据相关性进行聚类，从而发现热力图数据的潜在结构。clustermap()可以通过设置参数“metric”使用不同的相似度度量，默认为欧式距离“euclidean”，可选距离参数包括“braycurtis”、“canberra”、“chebyshev”、“cityblock”、“correlation”、“cosine”、“hamming”、“jaccard”，“mahalanobis”、“minkowski”等。还可以通过设置参数“method”使用不同的聚类方法，默认为为“average”算法，可选值包括“single”、“complete”、“weighted”、“centroid”和“median”等。另外可以设置是否对行数据或列数据进行标准化，参数“standard_scale”对应的值分别为 0 和 1。

其基本绘图方法如下：

```
g = sns.clustermap(flights, cmap="YlGnBu",standard_scale=1)
```

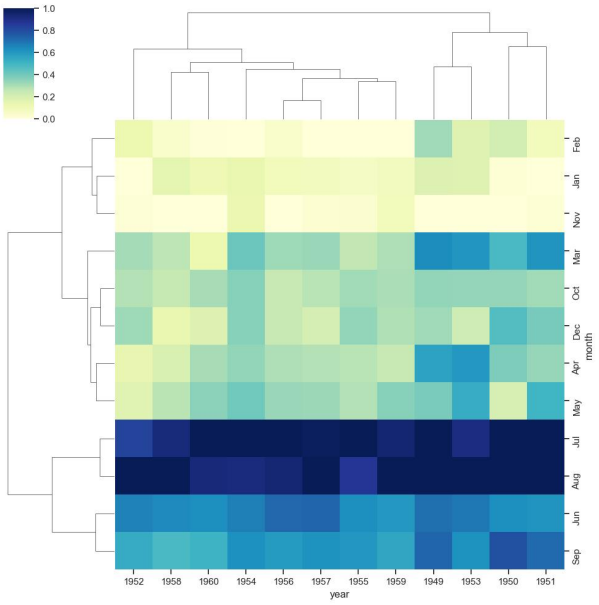


图 6-74 clustermap 的基本使用

6.3 结构化网格图

在探索多维数据时，一种高效的方法是在数据集的不同子集上绘制同一类型图形的多个实例。这种技术被称为结构化网格图，它可以快速提取关于复杂数据集的大量信息。

6.3.1 FacetGrid

当需要在数据集的子集中分别可视化变量的分布或多个变量之间的关系时，FacetGrid 非常有用。FacetGrid 最多可以绘制三个维度：row、col 和 hue。前两个表示轴变量，hue 变量可以看做第三个维度，其中不同的类别用不同的颜色绘制。前面介绍过的 relplot()、displot()、catplot()和 lmplot()都是基于 FacetGrid 的。

FacetGrid 使用时首先要通过一个 Dataframe 实例化一个对象，通过 Dataframe 的变量名设置网格的行、列和颜色维度。这些变量应该是分类的或离散的，然后变量每个类别的数据将用于网格图的一个子图。例如，假设我们想检查 tips 数据集中午餐和晚餐之间的差异，可以首先实例化一个 FacetGrid 对象，并指定 col=“time”：

```
tips = sns.load_dataset("tips")
g = sns.FacetGrid(tips, col="time")
```

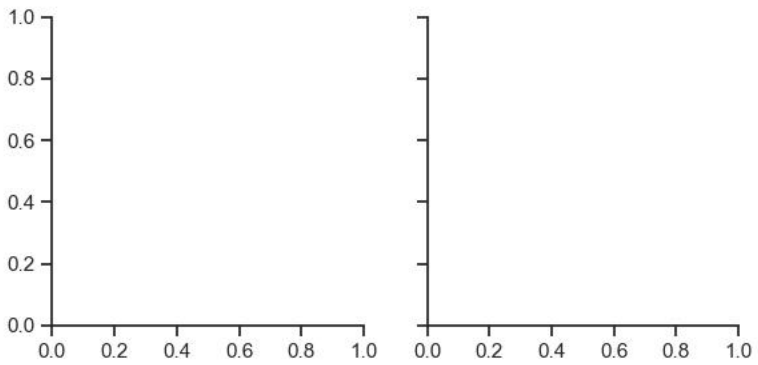


图 6-75 实例化一个 FacetGrid 对象

然后再使用 FacetGrid.map()方法在这个“空白网格”上对数据进行可视化。在下面的例子中，我们使用直方图查看每个子集（time=Lunch 或者 Dinner）中的小费金额的数据分布：

```
g.map(sns.histplot, "tip")
```

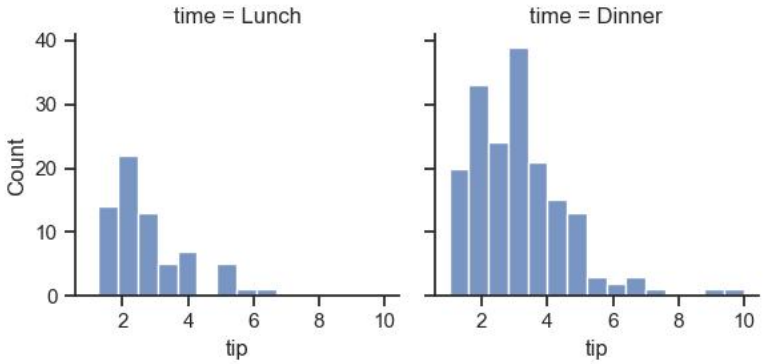


图 6-76 FacetGrid.map 方法的基本使用

在绘制关系图时，该函数可以通过设置 row、col 和 hue 的变量名称即可方便的完成图形绘制。例如：

```
g = sns.FacetGrid(tips, row="time", col="sex",
                  hue="smoker", margin_titles = True)
g.map(sns.scatterplot, "total_bill", "tip", alpha=.7)
g.add_legend()
```

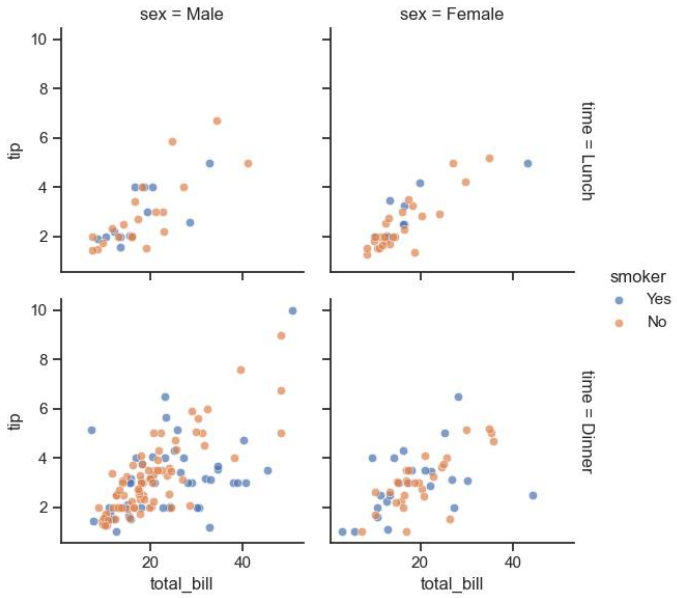


图 6-77 复杂的 FacetGrid 绘图

还可以通过设置子图高度以及横纵比来控制图形的大小。例如：

```
g = sns.FacetGrid(tips, col="day", height=4, aspect=.5)
g.map(sns.barplot, "sex", "total_bill",order=["Male", "Female"])
```

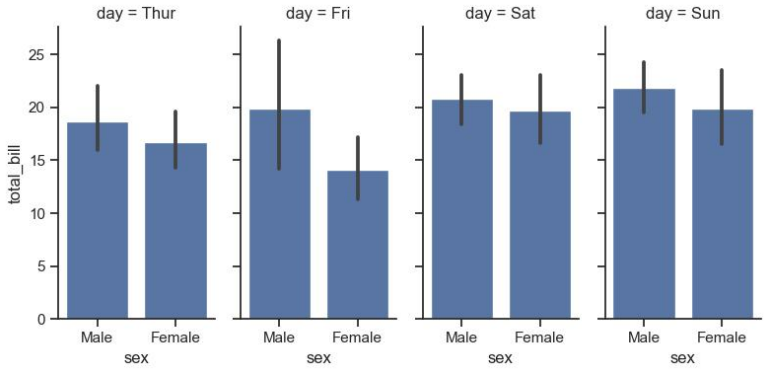


图 6-78 设置图形大小

在网格图中，各子图排列的默认顺序是根据 DataFrame 中的变量特点决定的，也可以通过设置参数“row_orde”r 或“col_order”指定子图的排列顺序。例如：

```
ordered_days = tips.day.value_counts().index
g = sns.FacetGrid(tips, row="day", row_order=ordered_days,
                  height=1.7, aspect=4,)
g.map(sns.kdeplot, "total_bill")
```

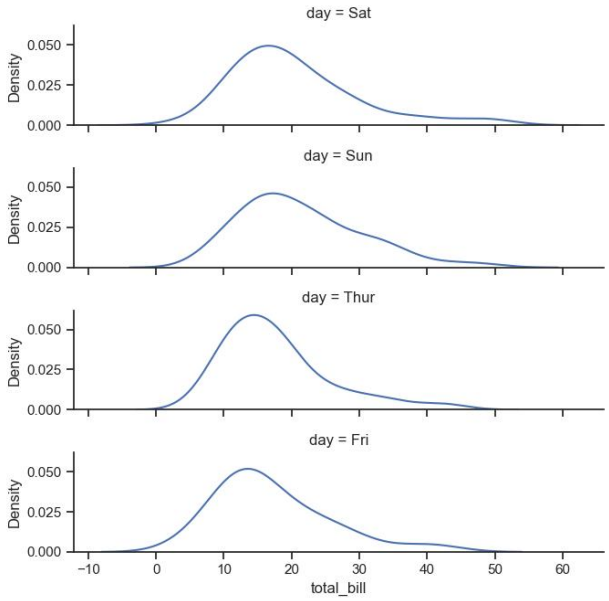


图 6-79 设置子图顺序

当一个变量有多个离散型取值时，可以沿着列依次绘制并设置“col_wrap”参数对其进行“封装”，使子图跨越多行。在执行此操作时，不能使用“row”参数。

```
attend = sns.load_dataset("attention").query("subject <= 12")
g = sns.FacetGrid(attend, col="subject", col_wrap=4,
                  height=2, ylim=(0, 10))
g.map(sns.pointplot, "solutions", "score", order=[1, 2, 3],
      color=".3", ci=None)
```

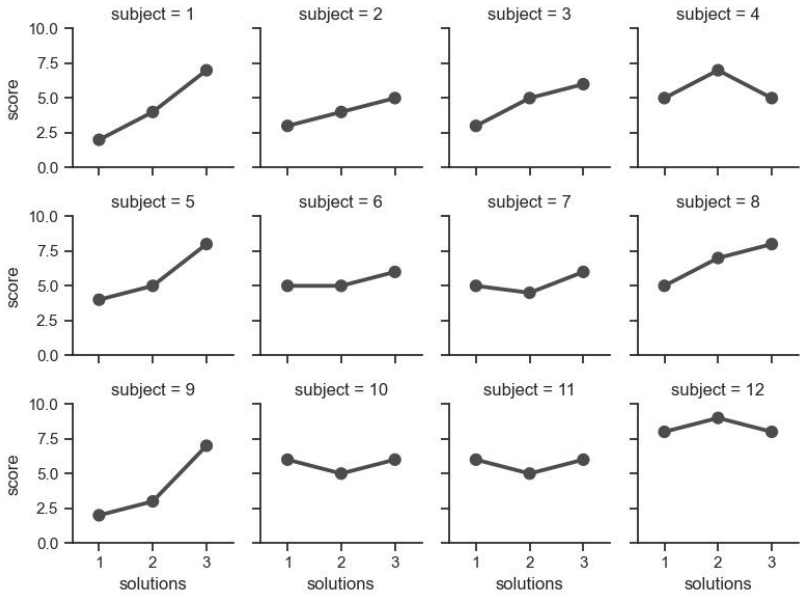


图 6-80 子图跨行显示

FacetGrid 对象上还有许多方法用于在更高的抽象级别上操作图形。最常用的方法包括 set()、set_axis_labels()等，可以方便的对坐标轴刻度、标签进行灵活的设置。例如：

```
g = sns.FacetGrid(tips, row="sex",col="smoker",
                  margin_titles=True, height=2.5)
g.map(sns.scatterplot, "total_bill", "tip", color="#334488")
g.set_axis_labels("Total bill (US Dollars)", "Tip")
g.set(xticks=[10, 30, 50], yticks=[2, 6, 10])
g.figure.subplots_adjust(wspace=.02, hspace=.02)
```

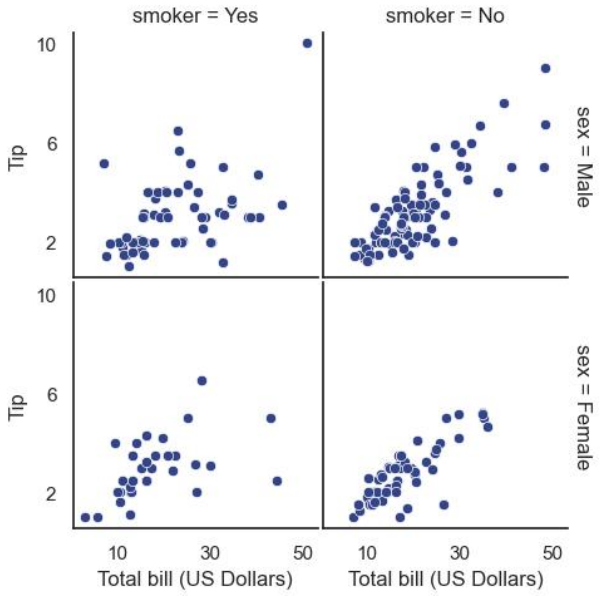


图 6-81 FacetGrid 坐标轴设置

6.3.2 PairGrid

PairGrid 支持使用相同的绘图类型快速绘制网格图。在 PairGrid 中，每一行和每一列

都被分配给不同的变量，因此可以显示数据集中的每一对变量的关系。这种类型的图有时被称为“散点图矩阵”，这是最常见的呈现方式，但 PairGrid 不限于散点图。

理解 FacetGrid 和 PairGrid 之间的区别是很重要的。在 FacetGrid 中，每个子图都以其他变量的不同类别为条件，显示相同变量对之间的关系。在 PairGrid 中，每个子图都显示不同变量对之间的关系。使用 PairGrid 可以对数据集中两两变量之间的关系进行非常快速、高级的总体呈现。PairGrid 的基本用法与 FacetGrid 非常相似：首先初始化网格，然后以参数传递的方式给 map 方法设置一个绘图函数，它将在每个子图上调用。与之相关的还有一个之前介绍过的函数 pairplot()，它以牺牲了一些灵活性为代价实现了更便捷的绘图。首先看一个简单的例子，在这个例子中把对角线上的子图设置为直方图，其他位置的子图设置为散点图，具体如下所示。

```
g = sns.PairGrid(iris)
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
```

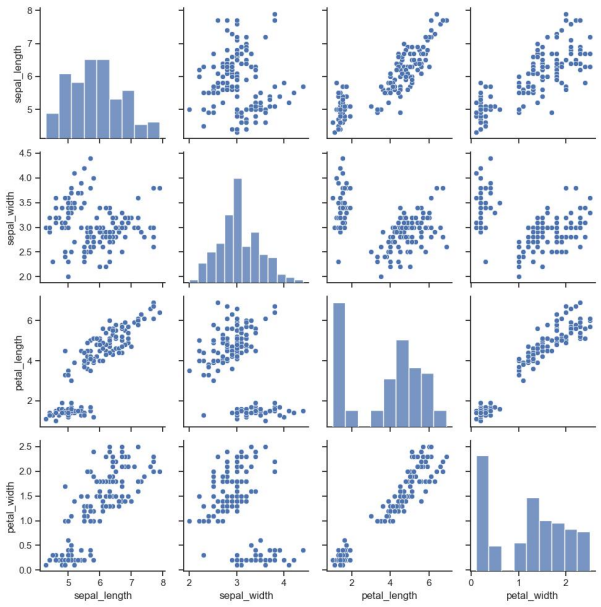


图 6-82 PairGrid 的基本使用

与前文介绍过的大多数函数一样，PairGrid 可以通过设置参数“hue”将不同类别的数据用不同颜色进行区分。例如：

```
g = sns.PairGrid(iris, hue="species")
g.map_diag(sns.histplot)
g.map_offdiag(sns.scatterplot)
g.add_legend()
```

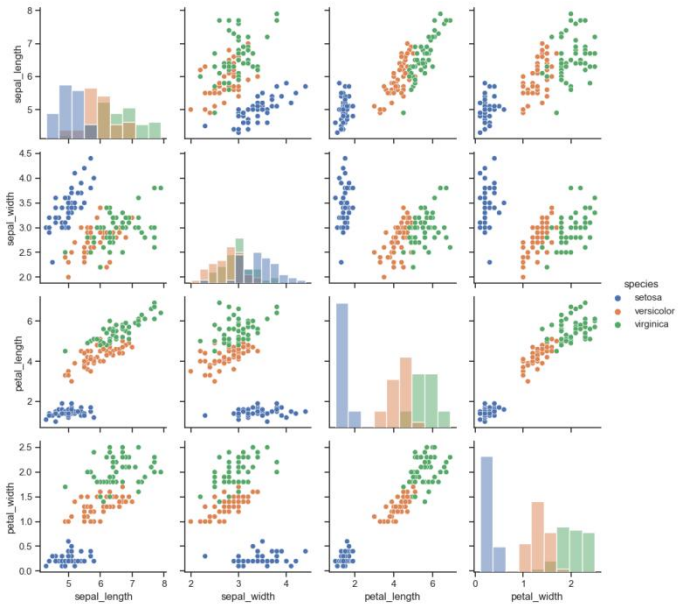


图 6-83 用不同颜色区分鸢尾花种类

默认情况下，PairGrid 使用数据集中的每个数值型的变量，也可以通过设置参数“vars”绘制特定变量之间的关系。例如：

```
g = sns.PairGrid(iris, vars=["sepal_length", "sepal_width"],
                 hue="species")
g.map(sns.scatterplot)
```

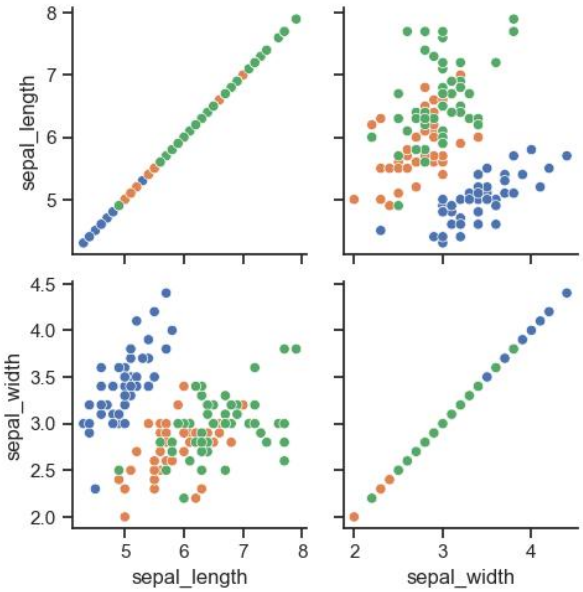


图 6-84 绘制特定变量之间的关系

在更复杂的应用中，还可以在上下三角形及对角线上使用不同的绘图函数来突出关系不同方面的特点。例如：

```
g = sns.PairGrid(iris)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot, lw=3, legend=False)
```

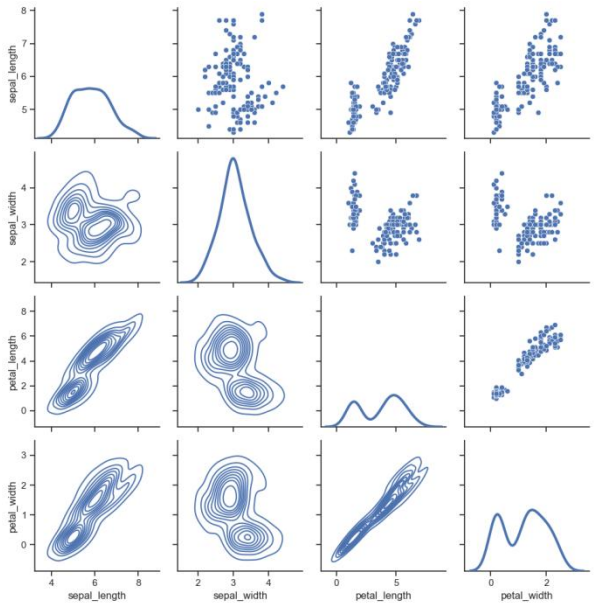


图 6-85 多种绘图类型的 PairGrid

实际上，对角线上有恒等关系的正方形网格只是一种特殊情况，还可以根据需要在行和列中绘制不同的变量。例如：

```
g = sns.PairGrid(tips, y_vars=["tip"],
                 x_vars=["total_bill", "size"], height=4)
g.map(sns.regplot, color=".3")
g.set(ylim=(-1, 11), yticks=[0, 5, 10])
```

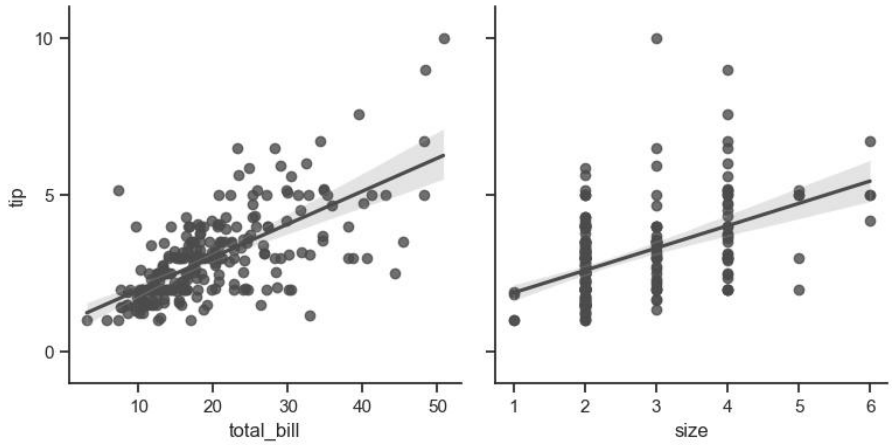


图 6-86 非正方形网格图

PairGrid 使用非常灵活，但是要快速查看数据集，使用之前介绍 pairplot()可能更加方便。该函数默认使用散点图和直方图，也可以通过参数“diag_kind”和“kind”灵活设置不同的绘图类型，在下面的例子中在对角线上绘制 kde 图，在非对角线上绘制回归图。

```
sns.pairplot(iris, hue="species", diag_kind="kde",
             kind="reg",height=2.5)
```

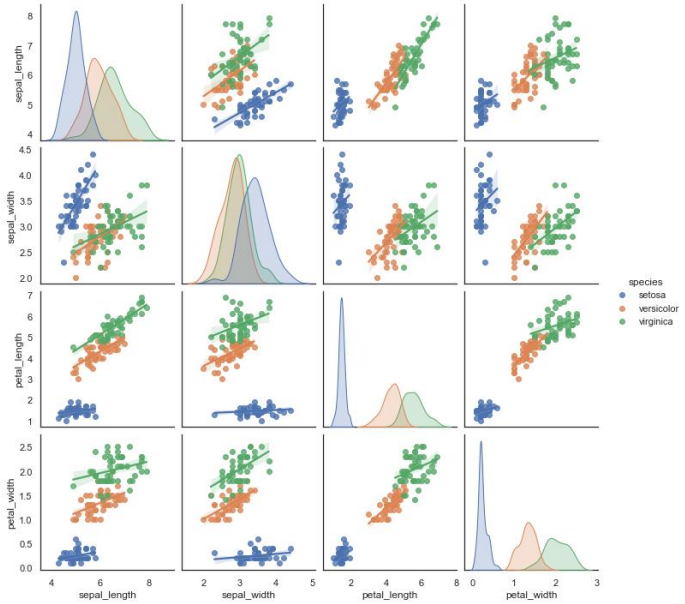


图 6-87 pairplot 设置不同绘图类型

6.4 绘图美化

6.4.1 风格设置

Seaborn 预定义了许多绘图的主题风格和一个调用接口，用于定制图形的外观。为了便于说明不同主题风格的差异，首先定义一个简单的函数 sinplot 来绘制一些偏移正弦波。

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
sinplot()
```

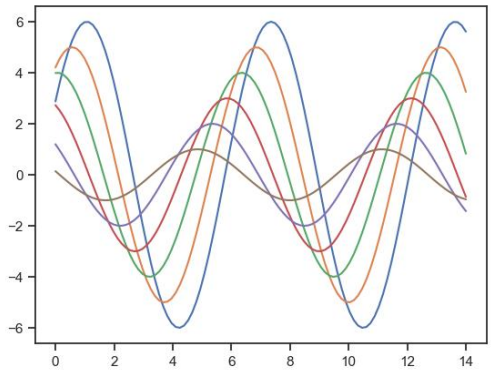


图 6-88 正弦波函数

要切换到 Seaborn 默认值，只需在绘图前调用 set_theme()函数。例如：

```
sns.set_theme()
sinplot()
```

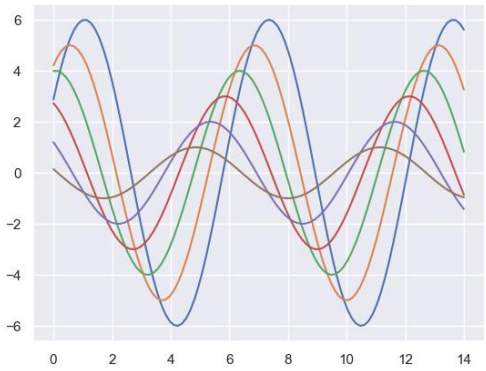


图 6-89 Seaborn 默认主题

Seaborn 的风格设置模块是把 Matplotlib 中的有关图形外观方面的参数分成了两组，一组是设置图形的主题，可以称之为“样式设置（style）”，主要包含 axes_style()和 set_style()函数；另一组是通过缩放各种图形元素使其与上下文环境更加协调，可以称之为“环境（context）设置”，可以使用 plotting_context()和 set_context()函数。

有 5 个预设的 Seaborn 主题风格：“darkgrid”、“whitegrid”、“dark”、“white”和“ticks”，可以根据应用场景与个人喜好来选择，默认主题是“darkgrid”。一般来说，网格有助于更清晰的查看定量信息，而灰色的底色能增强网格线与表示数据的图形的反差。除默认的“darkgrid”以外，其他 4 种主题风格的效果如图 6-88 所示，需要说明的是，在“with”语句中使用 axes_style()函数可以临时设置绘图参数。例如：

```
with sns.axes_style("white"):
    plt.subplot(221).set_title("style=white")
    sinplot()
with sns.axes_style("whitegrid"):
    plt.subplot(222).set_title("style=whitegrid")
    sinplot()
with sns.axes_style("dark"):
    plt.subplot(223).set_title("style=dark")
    sinplot()
with sns.axes_style("ticks"):
    plt.subplot(224).set_title("style=ticks")
    sinplot()
plt.tight_layout()
```

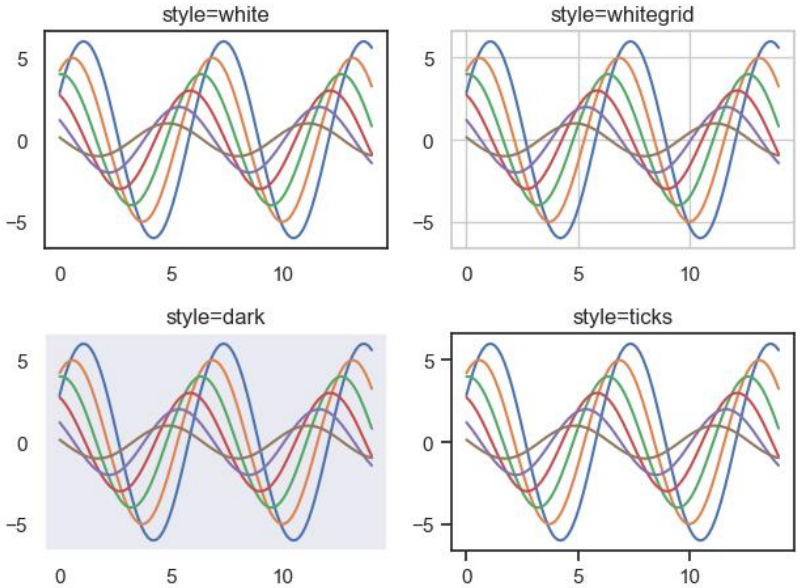
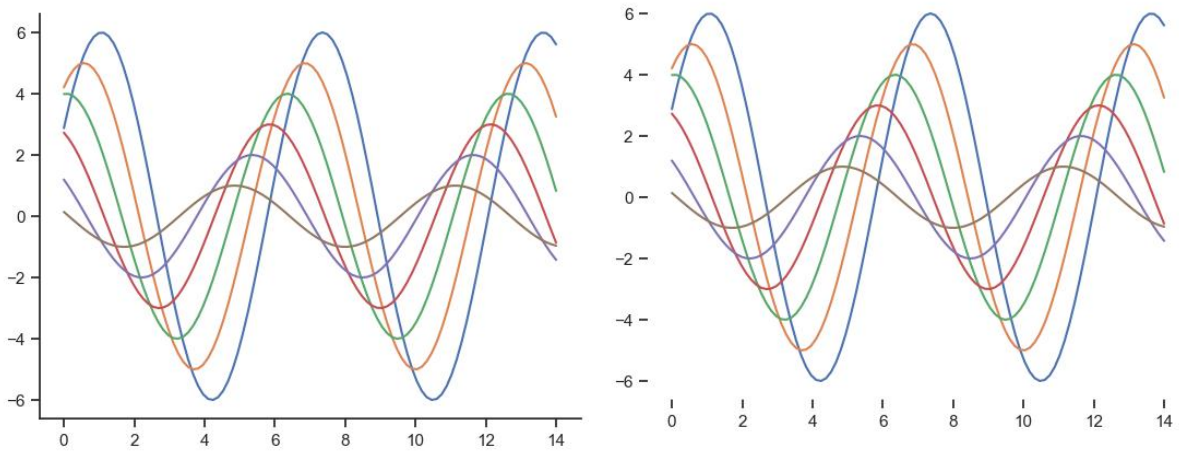


图 6-90 “white”“whitegrid”“dark”“ticks”四种主题风格

调用 despine()函数可以隐藏上边和右侧的边框，如果需要隐藏左侧或下面的边框需要设置参数 left=True 或 bottom=True。

```
sns.set_style("ticks")
sinplot()
sns.despine()
sns.despine(left=True,bottom=True)
```



a. sns.despine()　　b. sns.despine(left=True,bottom=True)

图 6-91 despine 方法的使用

支持的绘图环境主要有 4 种：“notebook”（默认环境）、“paper”、“talk”和“poster”。各种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。在设置环境之前，首先要调用 set_theme()函数设置默认参数。具体效果如图 6-90 所示。

```
sns.set_theme()
with sns.plotting_context("paper"):
    plt.subplot(221).set_title("paper")
    sinplot()
with sns.plotting_context("talk"):
    plt.subplot(222).set_title("talk")
    sinplot()
with sns.plotting_context("poster"):
    plt.subplot(223).set_title("poster")
    sinplot()
with sns.plotting_context("notebook"):
    plt.subplot(224).set_title("notebook")
    sinplot()
plt.tight_layout()
```

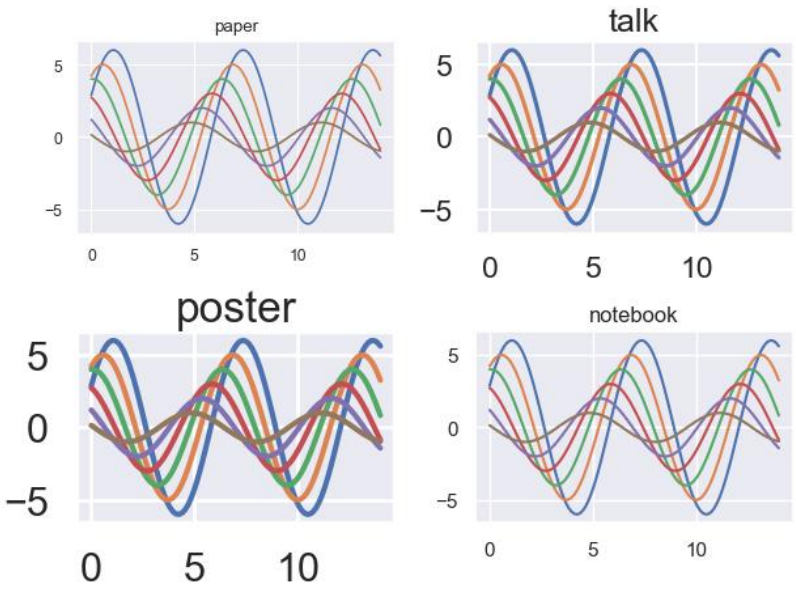


图 6-92 四种环境设置

环境配置的具体参数可以通过 sns.plotting_context()获取。例如：

```
sns.plotting_context("paper")
```

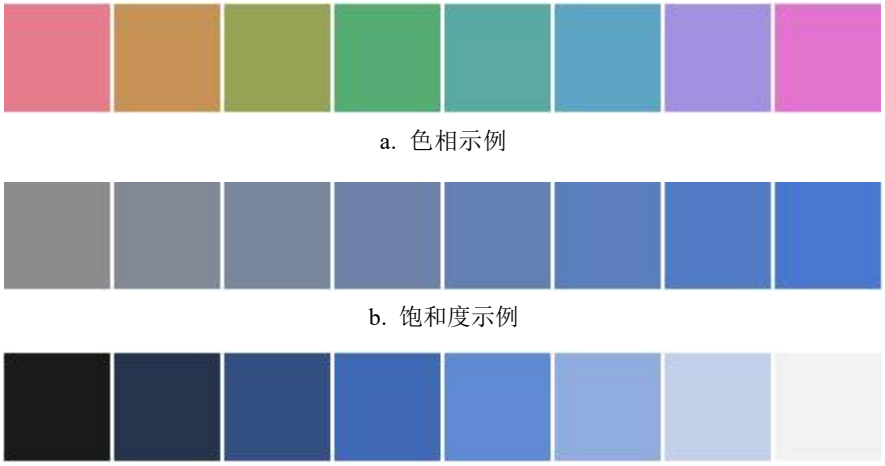
返回结果：

```
{'axes.linewidth': 1.0,
 'grid.linewidth': 0.8,
 'lines.linewidth': 1.2000000000000002,
 '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,
 'font.size': 9.600000000000001,
 'axes.labelsize': 9.600000000000001,
 'axes.titlesize': 9.600000000000001,
 'xtick.labelsize': 8.8,
 'ytick.labelsize': 8.8,
 'legend.fontsize': 8.8,
 'legend.title_fontsize': 9.600000000000001}
```

6.4.2 调色板

配色对于图形的美观和信息呈现的效果都非常重要。计算机中的颜色编码方式常见的有

RGB 和 HSL。RGB 模式中的三个分量（Red、Green、 Blue）对应的三个整数（范围在 0-255 之间）表示红、绿和蓝通道的强度，最终视觉中的颜色是由三种色彩分量混合叠加而成的。但要分析颜色的感知属性，最好从色相（Hue）、饱和度（Saturation）和亮度（Lightness）通道来考虑，即 HSL 模式。色相代表的是人眼所能感知的颜色范围，这些颜色分布在一个平面的色相环上，取值范围是 0°到 360°，每个角度可以代表一种颜色。（如图 6-93a）饱和度描述了相同色相、亮度下色彩纯度的变化，取值范围是 0%到 100%，数值越大，颜色中的灰色越少，颜色越鲜艳，呈现一种从灰度到纯色的变化。（如图 6-93b）亮度的作用是控制色彩的亮暗变化，取值范围是 0%到 100%，数值越小，色彩越暗，越接近于黑色，数值越大，色彩越亮，越接近于白色。（如图 6-93c）HSL 是工业界一种常用的色彩模式。



a. 色相示例

b. 饱和度示例

c. 亮度示例

图 6-93 HSL 色彩模式

在图中使用颜色的一般原则是以不同色相区分类别，以亮度来表示数字大小。先来看一个“色调区分类别”的例子。

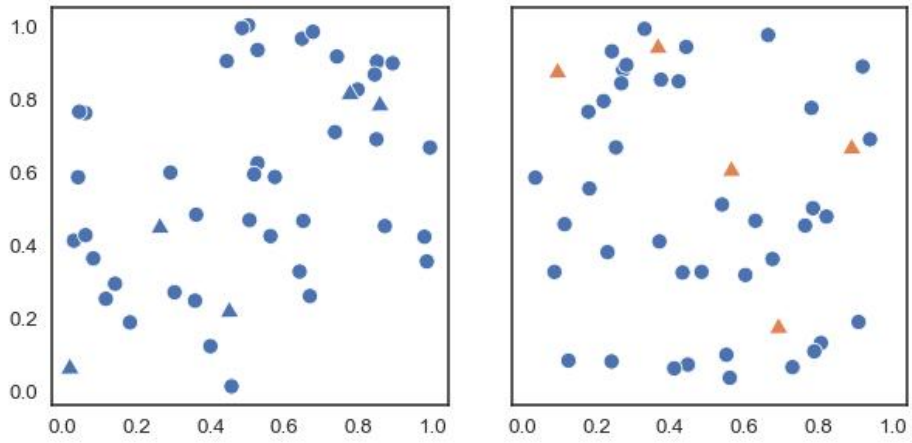


图 6-94 以色相区分类别

图 6-94 中，左侧图的两类数据元素从视觉上仅通过标记形状差异性区分并不直观，相比之下，右侧图中以色相区分更加一目了然。

下面来看一个“亮度表示数字”的例子。

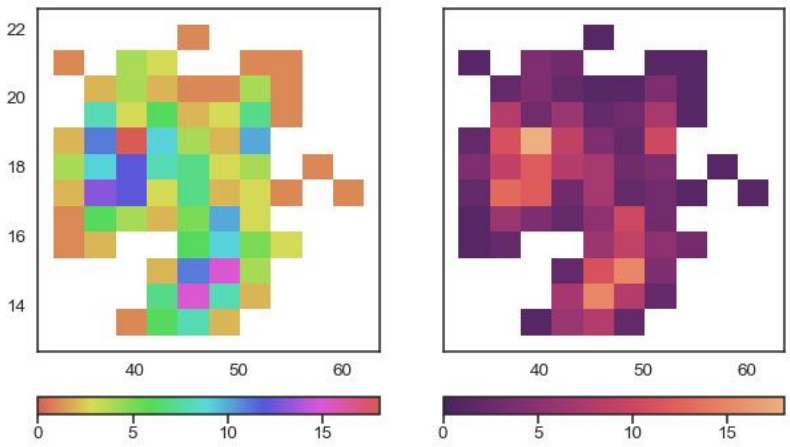


图 6-95 以亮度表示数字大小

图 6-95 中，左侧图使用基于色相的调色板，很难确定二元分布的形状。相比之下，右侧图中亮度调色板更清楚地表明有两个突出的峰值，变化的亮度有助于查看数据的潜在结构。

这些示例表明，调色板的选择不仅仅关乎美观，如果使用得当，颜色可以揭示数据中的深层次模式，因此对于特定的数据集和可视化方法，要有针对性的选择适用的调色板。

Seaborn 中，使用调色板最重要的函数是 color_palette()，它为大多数调色盘生成方式提供了统一的接口，能够被具有“palette”参数的任何函数使用。包含 2 个重要参数：

“palette”：设置调色板，可选项；可以选择 Seaborn 中预定义的调色板（“deep”、“muted”、“bright” 、“pastel”、“dark”、“colorblind”）、matplotlib 中的 colormap、“husl”和“hls”，还可以自定义调色板或赋值“None”（默认使用当前调色板）；

“n_colors”：设置调色板颜色数量，取值为整数类型，可选项。

为了适应于不同数据分析的应用场景，Seaborn 大体上把调色板分为三大类：分类调色板，适用于表示分类数据，进行定性分析；连续调色板，适用于表示数值型数据的变化趋势；发散调色板，适用于表示具有中心性和极值并向两端发散的数据。

（1）分类调色板

Seaborn 实际上有 Matplotlib 调色板的 6 种变体，分别是 deep、muted、pastel、bright、dark 和 colorblind。这些涵盖了一系列的平均亮度和饱和度值。

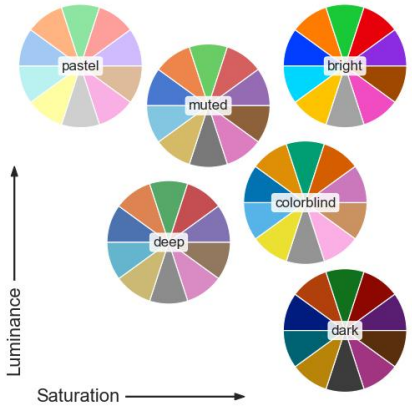


图 6-96 Seaborn 预定义的 6 种调色板

不带参数调用会返回当前默认颜色循环中的所有颜色：

```
sns.color_palette()
```

图 6-97 默认调色板

可以通过名称引用 Seaborn 的预定义调色板：

```
sns.color_palette("pastel")
```

图 6-98 "pastel"调色板

调用第三方配色工具 Color Brewer 定义的分类调色板：

```
sns.color_palette("Set2")
```

图 6-99 "Set2"调色板

如果需要灵活定义颜色数量，最常见的方法是使用 hls 色彩空间，并指定“n_colors”参数：

```
sns.color_palette("hls", 8)
```

图 6-100 "hls"调色板

在某些数据分析中，还需要生成配对的调色板：

```
sns.color_palette("Paired",10)
```

图 6-101 "Paired"调色板

（2）连续调色板

连续调色板中变化的主要维度是亮度，主要用于表示数值型数据的变化趋势，这意味着两种颜色的相对可分辨性与相应数据值之间的差异成正比。Seaborn 包括 4 种连续调色板：“rocket”、“mako”、“flare”和“crest”。前两种具有非常宽的亮度范围，非常适合应用于热力图（heatmat）。

```
sns.color_palette("rocket", as_cmap=True)
```

图 6-102 "rocket"调色板

Seaborn 也可以对连续调色板进行离散化：

```
sns.color_palette("rocket")
```

图 6-103 "rocket"调色板离散化

与 Matplotlib 中的规定一样，每个连续颜色图都有一个反转版本，其后缀为“_r”：

```
sns.color_palette("rocket_r", as_cmap=True)
```

图 6-104 "rocket"调色板反转

Matplotlib 中还内置了一种称为“立方螺旋（cubehelix）”算法生成的连续调色板：

```
sns.color_palette("cubehelix", as_cmap=True)
```

图 6-105 "cubehelix"调色板

（3）发散调色板

发散调色板类似于连续调色板，只是更加突出数据的中心点（通常是 0）和极值。发散调色板中应该有两个主要色调，分别代表数据的两个极值。Seaborn 包括 2 种发散调色板：“vlag”和“icefire”，他们都采用蓝色和红色作为两种主要色调。

```
sns.color_palette("vlag", as_cmap=True)
```

图 6-106 "vlag"调色板

Matplotlib 中还内置了 Color Brewer 定义的发散调色板。

```
sns.color_palette("Spectral", as_cmap=True)
```

图 6-107 "Spectral"调色板

Seaborn 中还提供了 diverging_palette()函数实现自定义发散调色板。

```
sns.diverging_palette(250, 30, l=65,center="dark",
                      as_cmap=True)
```

图 6-108 自定义发散调色板