

[学习笔记] [机器学习] 2. Seaborn (统计图形: 单变量, 双变量, 成对变量、分类数据: 散点图, 箱型图, 提琴图, 条形图, 点图、NBA 球员数据分析案例、北京租房..)

1. 视频链接
2. 数据集下载地址: [《2. Seaborn 及练习案例》配套数据集](#)

Seaborn 是一个基于 Matplotlib 的 Python 数据可视化库。它提供了一个高级接口，用于绘制吸引人且信息丰富的统计图形。

定位:

- 知道高级绘图工具 seaborn 的使用方法
- 通过综合案例，对科学计算库中学习内容综合练习

学习目标:

- 掌握 seaborn 的使用方法
- 可以灵活使用科学计算库中各个 API
 - 会使用 seaborn 绘制单变量、双变量图形
 - 会使用 seaborn 绘制箱线图、小提琴图
 - 会使用 seaborn 绘制条形图、点图

1. 绘制统计图形

学习目标:

- 知道 seaborn 的基本使用
- 会绘制单变量分布图形
- 会绘制双变量分布图形
- 会绘制成对的双变量分布图形

Matplotlib 是一个优秀的绘图库，但是它的 API 使用起来可能会比较复杂，因为它有上千个函数和参数。Seaborn 基于 Matplotlib 核心库进行了更高级的 API 封装，可以轻松地画出更漂亮的图形。Seaborn 的漂亮主要体现在配色更加舒服，以及图形元素的样式更加细腻。在使用 Seaborn 绘制图表之前，需要安装和导入绘图的接口。

```
# 安装
pip install seaborn

# 导入
import seaborn as sns
```

Seaborn 通常被导入为 `sns`，这是一个约定俗成的简写。这是对库名称的一个模糊参考，但我们也同样可以将其视为“seaborn 名称空间”。

【拓展】Seaborn 和 Matplotlib 有什么区别？

Matplotlib 是一个用于绘制图形的 Python 库，它可以与其他库（如 Numpy 和 Pandas）一起使用。Seaborn 也是一个用于绘制图形的 Python 库，它建立在 Matplotlib 的基础上，并被认为是 Matplotlib 库的超集。Seaborn 提供了更多的功能和组织方式，可以将整个数据集视为一个单元。它使用了吸引人的主题来装饰 Matplotlib 图形，并且可以帮助可视化单变量和双变量数据。

总之，Matplotlib 更适合绘制基本图形，而 Seaborn 更适合绘制更高级的统计图形。Seaborn 提供了更多吸引人的默认颜色调色板，并且允许您可视化回归模型，而 Matplotlib 则不能。

1.1 可视化数据的分布

当处理一组数据时，通常先要做的就是了解变量是如何分布的。

- 对于单变量的数据来说采用直方图或核密度曲线是个不错的选择，
- 对于双变量来说，可采用多面板图形展现，比如散点图、二维直方图、核密度估计图形等。

核密度估计 (KDE) 是一种用于概率密度估计的核平滑方法，即一种基于核作为权重的非参数方法，用于估计随机变量的概率密度函数。它试图根据有限的数据样本推断总体的特征。

简单来说，核密度估计可以让您根据一组随机数据创建平滑曲线。它通过绘制数据并开始创建分布曲线来工作。曲线是通过在分布的每个特定位置对所有点的距离进行加权来计算的。如果有更多的点局部分组，则估计值更高，因

为在该位置看到一个点的概率增加。核函数是用于在数据集中对点进行加权的特定机制。

针对这种情况，Seaborn 库提供了对单变量和双变量分布的绘制函数，如 `displot()` 函数、`jointplot()` 函数，下面来介绍这些函数的使用。

1.2 绘制单变量分布

可以采用最简单的直方图描述单变量的分布情况。Seaborn 中提供了 `histplot()` 函数和 `displot()` 函数。

```
1. sns.histplot(data=None, x=None, y=None, hue=None, weights=None, stat="count", bins="auto", color=None, kde=False) ->
matplotlib.axes.Axes :
```

1. 作用：用于绘制直方图以显示数据集的分布。直方图是一种经典的可视化工具，它通过计算落入离散箱中的观测数量来表示一个或多个变量的分布。此函数可以将每个箱内计算的统计量标准化以估计频率、密度或概率质量，并且可以使用核密度估计添加平滑曲线，类似于 `kdeplot()`。

2. 参数：

1. `data`：输入数据结构。
1. 默认值为 `None`。
2. `x`、`y`：在 `x` 和 `y` 轴上指定位置的变量。
1. 默认值为 `None`。
3. `hue`：用于确定绘图元素颜色的语义变量。
1. 默认值为 `None`。
4. `weights`：如果提供，则按这些因子对应数据点在每个箱中的贡献进行加权。
1. 默认值为 `None`。
5. `stat`：在每个箱中计算的聚合统计量。
1. 默认值为 `'count'`。
6. `bins`：直方图的箱数或箱边界。
1. 默认值为 `'auto'`。
7. `binwidth`：每个箱的宽度，覆盖 `bins` 但可以与 `binrange` 一起使用。
1. 默认值为 `None`。
8. `color`：颜色。
1. 默认值为 `None`。
9. `kde`：用于控制是否在直方图上添加一个平滑曲线，该曲线使用核密度估计获得，类似于 `sns.kdeplot()`。
1. 默认值为 `False`。

3. 返回值：返回一个 `matplotlib.axes.Axes` 对象，该对象表示绘制数据的轴。

`sns.displot()` 是 Seaborn 库中用于绘制分布图的函数。其中，`dis` 是 `distribution` 的缩写，表示分布。该函数提供了多种方法来可视化单变量或双变量数据的分布，包括通过语义映射定义的数据子集和在多个子图上进行分面绘制。

```
2. sns.displot(data=None, x=None, y=None, hue=None, row=None, col=None, kind="hist", rug=False, color=None) -> FacetGrid :
```

1. 作用：用于绘制分布图。它提供了多种方法来可视化单变量或双变量数据的分布，包括通过语义映射和在多个子图上分面定义的数据子集。

2. 参数：

1. `data`：输入数据结构。
1. 默认值为 `None`。
2. `x`、`y`：在 `x` 和 `y` 轴上指定位置的变量。
1. 默认值为 `None`。
3. `hue`：用于确定绘图元素颜色的语义变量。
1. 默认值为 `None`。
4. `row`、`col`：定义要在不同面板上绘制的子集的变量。
1. 默认值为 `None`。
5. `kind`：用于可视化数据的方法。选择底层绘图函数并确定附加有效参数集。
1. 默认值为 `'hist'`。
6. `rug`：如果为 `True`，则使用边缘刻度显示每个观测值（如在 `rugplot()` 中）。
1. 默认值为 `False`。
7. `color`：颜色。
1. 默认值为 `None`。

3. 返回值：返回一个 `FacetGrid` 对象，该对象管理图形并提供其他方法来自定义绘图。

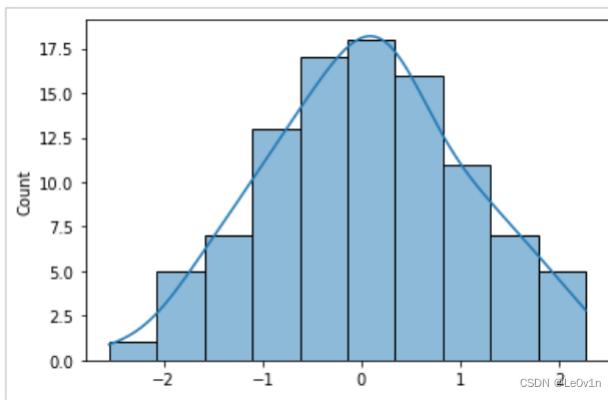
示例：绘制直方图

```
import seaborn as sns
import numpy as np
```

```
np.random.seed(0)
arr = np.random.randn(100) # 生成随机数组
```

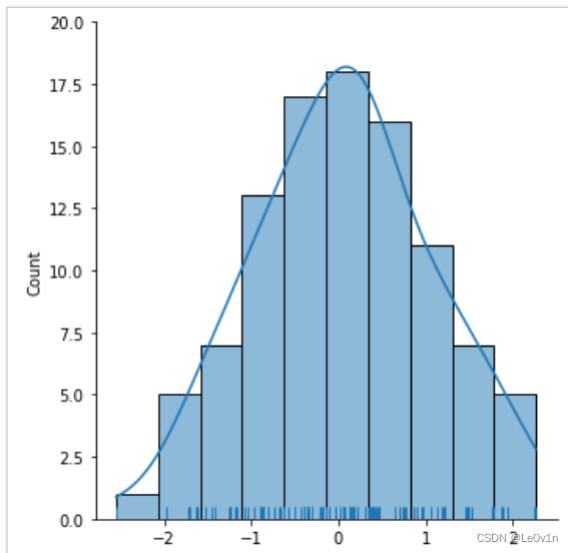
一、sns.histplot() 绘制带有 KDE 的直方图

```
# 1. sns.histplot()
ax = sns.histplot(data=arr, bins=10, kde=True)
```



二、sns.displot() 绘制带有 KDE 的直方图

```
# 2. sns.displot()
ax = sns.displot(data=arr, kind="hist", bins=10, kde=True, rug=True)
```



`rug=True` 的目的是看一下数据的集中情况。

从上图中看出：

- 直方图共有 10 个条柱，每个条柱的颜色为蓝色，并且有核密度估计曲线。
- 根据条柱的高度可知，位于 $[-1, 1]$ $[-1, 1]$ 区间的随机数值偏多，小于 -2 -2 的随机数值偏少。

通常，采用直方图可以比较直观地展现样本数据的分布情况。不过，直方图存在一些问题，它会因为条柱数量的不同导致直方图的效果有很大的差异。为了解决这个问题，可以绘制核密度估计曲线进行展现。

三、sns.displot() 绘制 KDE 核密度估计图

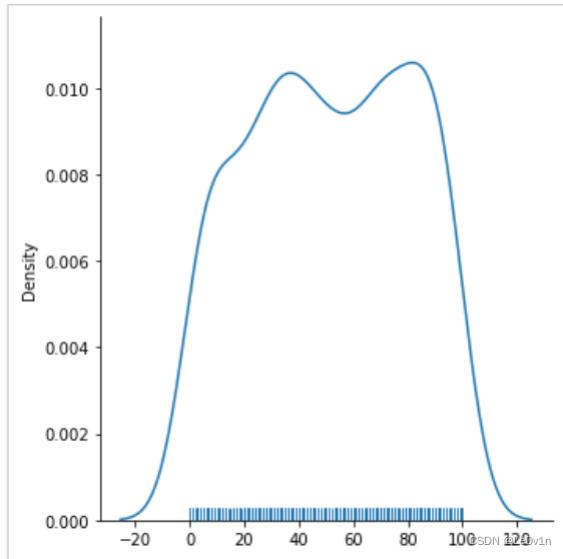
KDE 是 Kernel Density Estimation 的缩写，意为核密度估计。

- 核密度估计是在概率论中用来估计未知的密度函数，属于非参数检验方法之一，可以比较直观地看出数据样本本身的分布特征。

通过 `sns.displot()` 函数绘制核密度估计曲线的示例如下。

```
# 创建包含500个位于[0, 100]之间的随机整数数组
arr = np.random.randint(low=0, high=101, size=500)

# 绘制KDE曲线
sns.displot(data=arr, kind="kde", rug=True)
```



从上图中看出，图表中有一条核密度估计曲线，并且在 x 轴的上方生成了观测数值的小细条（`rug=True` 的作用）。

1.3 绘制双变量分布

两个变量的二元分布可视化也很有用。在 Seaborn 中最简单的方法是使用 `jointplot()` 函数，该函数可以创建一个多面板图形，比如散点图、二维直方图、核密度估计等，以显示两个变量之间的双变量关系及每个变量在单坐标轴上的单变量分布。

`jointplot()` 函数的语法如下。

- `sns.jointplot(data=None, x=None, y=None, hue=None, kind="scatter", height=6, ratio=5, space=0.2, dropna=False, xlim, ylim, color)` -> `JointGrid` :
- 作用：用于绘制两个变量的联合分布图，包括双变量图和单变量图。它提供了一个方便的接口来使用 `JointGrid` 类，并提供了几种预设的绘图类型。如果您需要更多的灵活性，应该直接使用 `JointGrid` 类。
- 参数：
 - `data` : 输入数据结构。可以是长格式的向量集合，也可以是宽格式的数据集。
 - `x` 和 `y` : 指定 x 轴和 y 轴上的变量。
 - `hue` : 指定用于确定绘图元素颜色的语义变量。
 - `kind` : 绘图类型。可选值包括：
 - 'scatter' (默认值)
 - 'kde'
 - 'hist'
 - 'hex'
 - 'reg'
 - 'resid'
 - 为 'scatter' 。
 - `height` : 图形的大小 (单位为英寸)。
 - 默认值为 6。
 - `ratio` : 联合轴高度与边缘轴高度之比。 (表示中心图与侧边图的比例。该参数的值越大，则中心图的占比会越大)
 - 默认值为 5。
 - `space` : 联合轴与边缘轴之间的空间。 (用于设置中心图与侧边图的间隔大小)
 - 默认值为 0.2。
 - `dropna` : 如果为 True，则删除 x 和 y 中缺失的观测值。
 - 默认值为 False。
 - `xlim` 和 `ylim` : 绘图前设置的轴限制。
 - `color` : 当未使用色调映射时使用的单一颜色规范。否则，绘图将尝试连接到 Matplotlib 属性循环。
- 返回值：返回一个 `JointGrid` 对象，用于管理多个子图，这些子图对应于用于绘制双变量关系或分布的联合轴和边缘轴。

下面以散点图、二维直方图、核密度估计曲线为例，使用 Seaborn 绘制这些图形。

一、绘制散点图

调用 `sns.jointplot()` 函数绘制散点图的示例如下。

```
import numpy as np
import pandas as pd
import seaborn as sns

# 固定随机种子
```

```

np.random.seed(0)

# 创建DF对象
df = pd.DataFrame({'x': np.random.randn(500),
                   'y': np.random.randn(500)})

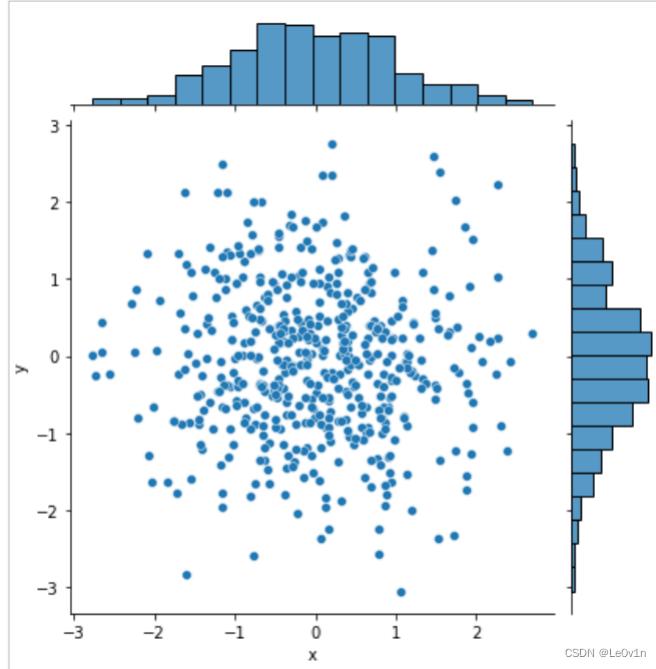
print(df.head())
"""

      x        y
0  1.764052  0.382732
1  0.400157 -0.034242
2  0.978738  1.096347
3  2.240893 -0.234216
4  1.867558 -0.347451
"""

# 绘制散点图
sns.jointplot(data=df, x='x', y='y', kind="scatter")

```

上述示例中，首先导入了 `numpy` 和 `pandas` 库，并使用 `numpy.random.seed(0)` 设置随机种子，以确保每次运行代码时生成的随机数相同。接下来，使用 `numpy.random.randn(500)` 生成两个包含 500 个随机数的一维数组，并将它们作为列添加到一个新创建的 `pandas.DataFrame` 对象中。这些随机数服从标准正态分布。最后，使用 `sns.jointplot` 函数绘制散点图。`data=df` 表示使用前面创建的 `DataFrame` 对象作为数据。`x='x'` 和 `y='y'` 表示在 x 轴和 y 轴上分别绘制数据框中名为 ‘x’ 和 ‘y’ 的列。



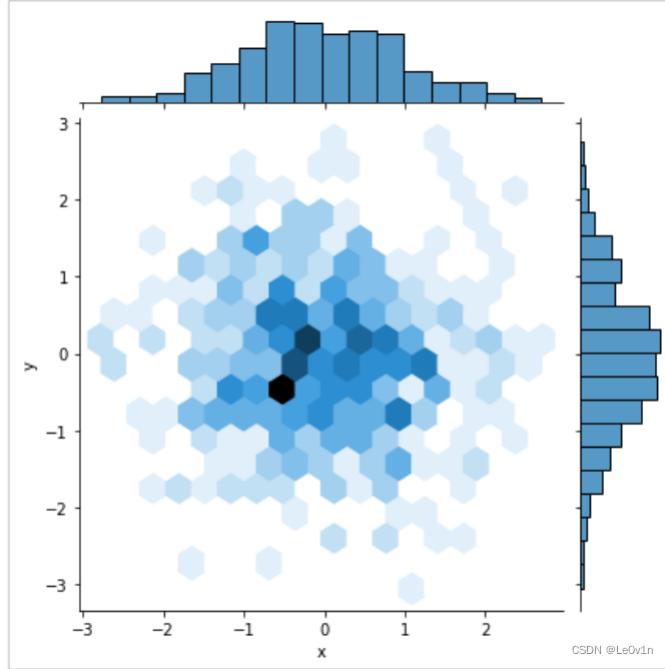
二、绘制直方图

二维直方图类似于 “六边形” 图，主要是因为它显示了落在六角形区域内的观察值的计数，适用于较大的数据集。当调用 `sns.jointplot()` 函数时，只要传入 `kind="hex"`，就可以绘制二维直方图，具体示例代码如下。

```

sns.jointplot(data=df, x='x', y='y', kind="hex")

```

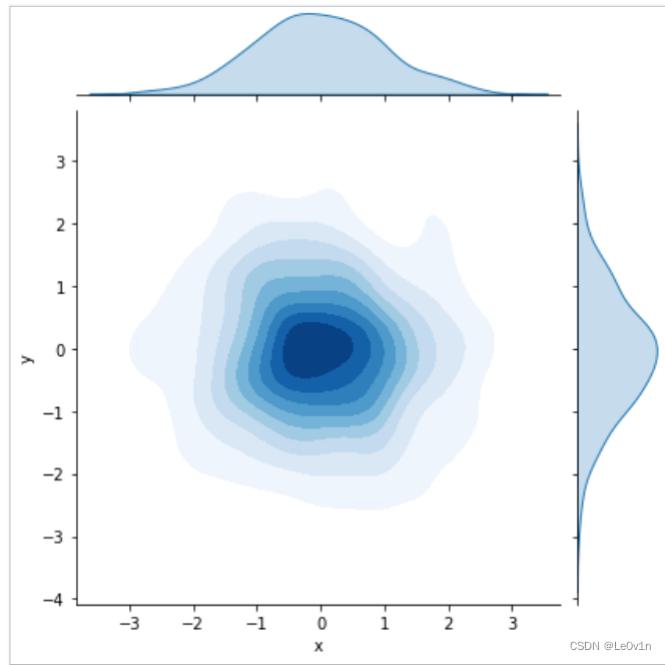


根据六边形颜色的深浅，可以观察到数据密集的程度。另外，图形的上方和右侧仍然给出了直方图。注意：在绘制二维直方图时，最好使用白色背景。

三、绘制核密度估计图形 kde

利用核密度估计同样可以查看二元分布，其用等高线图来表示。当调用 `sns.jointplot()` 函数时只要传入 `kind="kde"` 就可以绘制核密度估计图形，具体示例代码如下。

```
sns.jointplot(data=df, x='x', y='y', kind="kde", fill=True, cmap="Blues")
```



通过观等高线的颜色深浅，可以看出哪个范围的数值分布的最多，哪个范围的数值分布的最少。

1.4 绘制成对的双变量分布

要想在数据集中绘制多个成对的双变量分布，则可以使用 `sns.pairplot()` 函数实现。该函数会创建一个坐标轴矩阵，并且显示 DataFrame 对象中每对变量的关系。另外，`sns.pairplot()` 函数也可以绘制每个变量在对角轴上的单变量分布。

- `sns.pairplot(data=)` -> `PairGrid` :
 - 作用：用于绘制成对关系图。它可以绘制一个数据集中每对变量之间的散点图矩阵，对角线上则是各个变量的直方图或核密度估计图。
 - 参数：
 - `data` : DataFrame，用于绘制图形的数据。
 - `hue` : 字符串，用于指定分类变量。
 - `hue_order` : 列表，用于指定分类变量的顺序。

- `palette` : 字典或颜色序列，用于指定分类变量的颜色。
 - `vars` : 列表，用于指定要绘制的变量。
 - `x_vars` : 列表，用于指定要绘制的 x 变量。
 - `y_vars` : 列表，用于指定要绘制的 y 变量。
 - `kind` : 字符串，用于指定非对角线上图形的类型。可选值为 ‘scatter’ 和 ‘reg’ 。
 - `diag_kind` : 字符串，用于指定对角线上图形的类型。可选值为 ‘auto’ 、 ‘hist’ 和 ‘kde’ 。
 - `markers` : 字符串或字符串列表，用于指定散点图中点的标记样式。
 - `height` : 标量，用于指定每个子图的高度 (英寸) 。
 - `aspect` : 标量，用于指定每个子图的纵横比。
 - `corner` : 布尔值，如果为 True，则仅绘制下三角矩阵。
 - `dropna` : 布尔值，如果为 True，则删除包含缺失值的观测值。
 - `plot_kws` : 字典，用于传递给非对角线上图形的绘制函数。
 - `diag_kws` : 字典，用于传递给对角线上图形的绘制函数。
- 返回值: 返回一个 `PairGrid` 对象。

接下来，通过 `sns.pairplot()` 函数绘制数据集变量间关系的图形，示例代码如下：

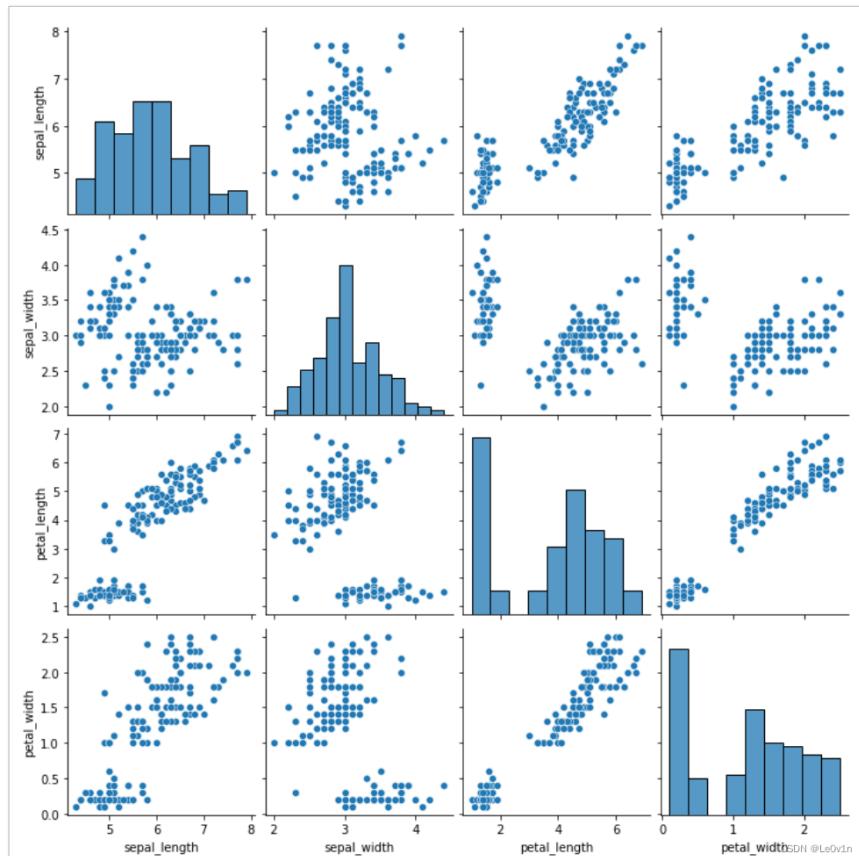
```
# 加载seaborn中的iris鸢尾花数据集
dataset = sns.load_dataset("iris")
dataset.head()
```

	<code>sepal_length</code>	<code>sepal_width</code>	<code>petal_length</code>	<code>petal_width</code>	<code>species</code>
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

CSDN @Le0vIn

上述示例中，通过 `sns.load_dataset()` 函数加载了 seaborn 中内置的数据集，根据 iris 数据集绘制多个双变量分布。

```
# 绘制多个成对的双变量分布图
sns.pairplot(data=dataset)
```



从上图可以看到，`sns.pairplot()` 函数用于绘制对关系图。它可以绘制一个数据集中每对变量之间的散点图矩阵，对角线上则是各个变量的直方图或核密度估计图。

小结：

- seaborn 的基本使用【了解】

- 绘制单变量分布图形【知道】
 - `sns.histplot()`
 - `sns.displot()`
- 绘制双变量分布图形【知道】
 - `sns.jointplot()`
- 绘制成对的双变量分布图形【知道】
 - `sns.pairplot()`

2. 用分类数据绘图

学习目标：

- 会使用 seaborn 绘制箱线图、小提琴图
- 会使用 seaborn 绘制条形图、点图

数据集中的数据类型有很多种，除了连续的特征变量之外，最常见的就是类别型的数据了，比如人的性别、学历、爱好等，这些数据类型都不能用连续的变量来表示，而是用分类的数据来表示。

Seaborn 针对分类数据提供了专门的可视化函数，这些函数大致可以分为如下三种：

1. 分类数据 散点图： `sns.swarmplot()` 与 `sns.stripplot()`
2. 分类数据的 分布图： `sns.boxplot()` 与 `sns.violinplot()`
3. 分类数据的 统计估算图： `sns.barplot()` 与 `sns.pointplot()`

下面两节将针对分类数据可绘制的图形进行简单介绍。

在画图时使用到 `tips` 数据集，该数据集的加载及详情如下。

```
tips = sns.load_dataset("tips")
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

2.1 类别散点图

1. `sns.stripplot()`
2. `sns.swarmplot()`

2.1.1 sns.stripplot()

通过 `sns.stripplot()` 函数可以画一个散点图，`sns.stripplot()` 函数的语法格式如下。

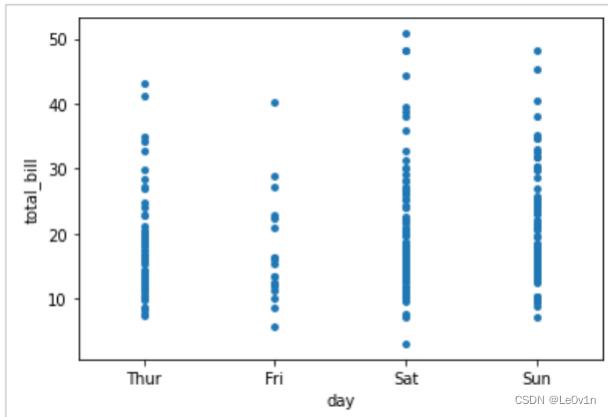
```
sns.stripplot(x=None, y=None, hue=None, data=None, order=None, hue_order=None, jitter=False)
```

`sns.stripplot()` 是 Seaborn 库中的一个函数，用于绘制分布散点图。它接受以下参数：

- `x`：指定要绘制的数据的 x 轴变量。
- `y`：指定要绘制的数据的 y 轴变量。
- `hue`：指定要使用的颜色变量。
- `data`：指定要使用的数据集。
- `order`：指定分类变量的顺序。
- `hue_order`：指定颜色变量的顺序。
- `jitter`：表示散点图的各散点在回归模型中小幅度的分布。

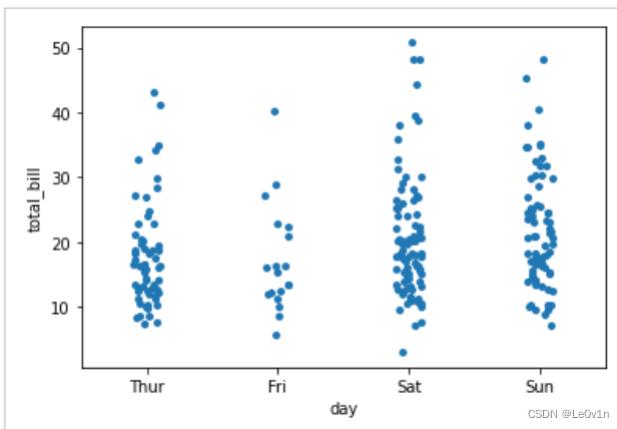
为了让大家更好地理解，接下来，通过 `sns.stripplot()` 函数绘制一个散点图，示例代码如下。

```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=False)
```



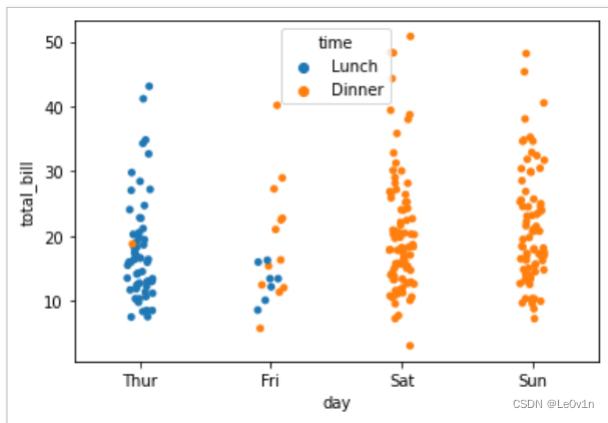
从上图中可以看出，图表中的横坐标是分类的数据，而且一些数据点会互相重叠，不易于观察。为了解决这个问题，可以在调用 `sns.stripplot()` 函数时传入 `jitter` 参数，以调整横坐标的位置，改后的示例代码如下。

```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True)
```



我们也可以加入 `time` 信息：

```
sns.stripplot(x="day", y="total_bill", data=tips, hue="time", jitter=True)
```



2.1.2 sns.swarmplot()

除此之外，还可调用 `sns.swarmplot()` 函数绘制散点图，该函数的好处是所有的数据点都不会重叠，可以很清晰地观察到数据的分布情况。`sns.swarmplot()` 函数的语法格式如下。

```
seaborn.swarmplot(data=None, x=None, y=None, hue=None, order=None, hue_order=None)
```

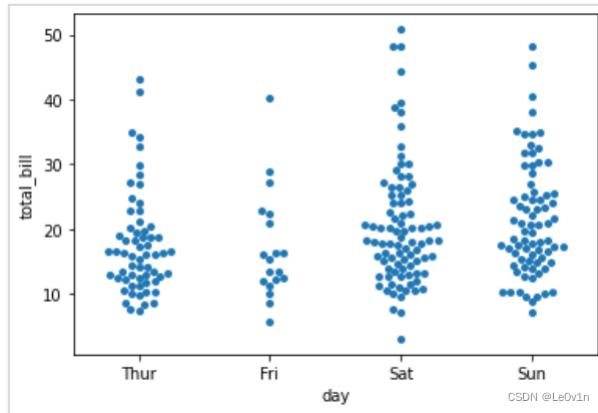
`sns.swarmplot()` 是 Seaborn 库中的一个函数，用于绘制分簇散点图。它接受以下参数：

- `x`：指定要绘制的数据的 x 轴变量。
- `y`：指定要绘制的数据的 y 轴变量。
- `hue`：指定要使用的颜色变量。
- `data`：指定要使用的数据集。
- `order`：指定分类变量的顺序。

- `hue_order` : 指定颜色变量的顺序。

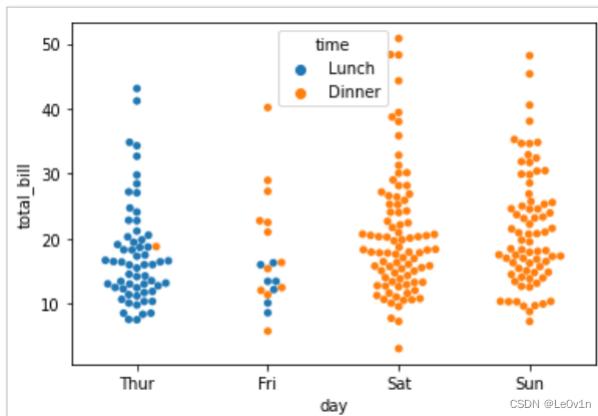
示例代码如下。

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



我们也可以加入 `time` 信息：

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



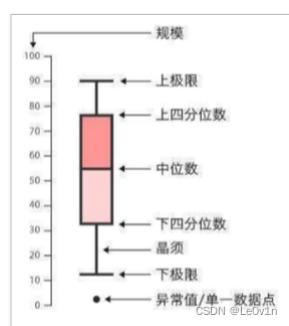
2.2 类别内的数据分布

要想查看各个分类中的数据分布，散点图是不满足需求的，原因是它不够直观。针对这种情况，我们可以绘制如下两种图形进行查看：

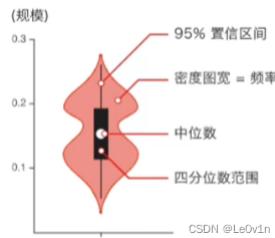
1. 箱形图

2. 小提琴图

箱形图 (Box-plot) 称为盒须图、盒式图或箱线图，是一种用作显示一组数据分散情况资料的统计图。因形状如箱子而得名。箱形图于 1977 年由美国著名统计学家约翰·图基 (John Tukey) 发明。它能显示出一组数据的最大值、最小值、中位数、及上下四分位数。



小提琴图 (Violin Plot) 用于显示数据分布及其概率密度。这种图表结合了箱形图和密度图的特征，主要用来显示数据的分布形状。



中间的黑色粗条表示四分位数范围，从其延伸的幼细黑线代表 95% 置信区间，而白点则为中位数。箱形图在数据显示方面受到限制，简单的设计往往隐藏了有关数据分布的重要细节。例如使用箱形图时，我们不能了解数据分布。虽然小提琴图可以显示更多详情，但它们也可能包含较多干扰信息。

2.2.1 绘制箱形图

seaborn 中用于绘制箱形图的函数为 `sns.boxplot()`，其语法格式如下：

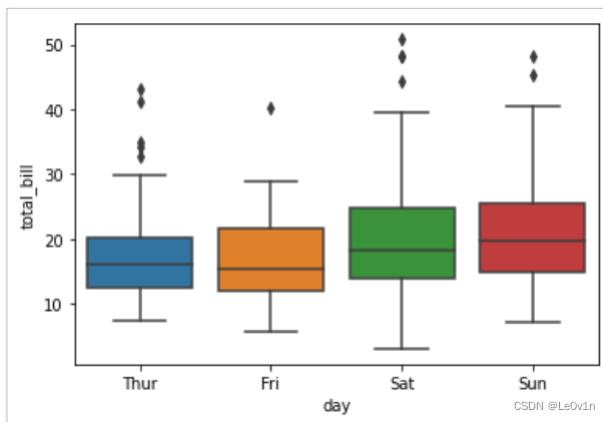
```
sns.boxplot(data=None, *, x=None, y=None, hue=None, order=None,
            hue_order=None, orient=None, color=None, palette=None,
            saturation=0.75, width=0.8, dodge=True, fliersize=5,
            linewidth=None, whis=1.5, ax=None, **kwargs)
```

参数：

- `x`：指定要绘制的数据的 x 轴变量。
- `y`：指定要绘制的数据的 y 轴变量。
- `hue`：指定要使用的颜色变量。
- `data`：指定要使用的数据集。
- `order`：指定分类变量的顺序。
- `hue_order`：指定颜色变量的顺序。
- `palette`：指定要使用的调色板。Seaborn 库提供了多种调色板，包括 `deep`、`muted`、`pastel`、`bright`、`dark` 和 `colorblind` 等，这些调色板具有不同的亮度和饱和度值。
- `saturation`：指定颜色的饱和度。饱和度越高，颜色越鲜艳；饱和度越低，颜色越暗淡。

使用 `sns.boxplot()` 函数绘制箱形图的具体示例如下。

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



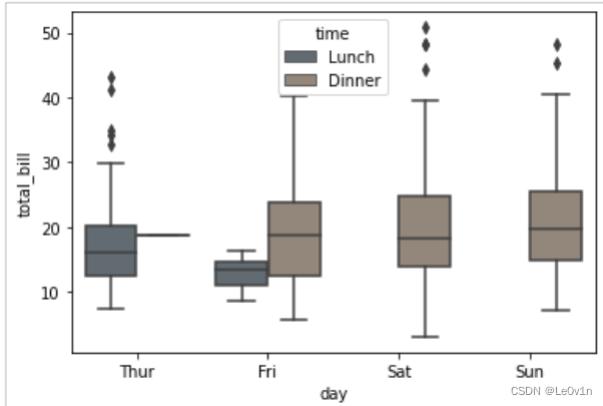
上述示例中，使用 seaborn 中内置的数据集 tips 绘制了一个箱形图，图中 x 轴的名称为 `day`，其刻度范围是 Thur ~ Sun(周四至周日)，y 轴的名称为 `total_bill`，刻度范围为 10-50 左右。

从图中可以看出：

- Thur 列大部分数据都小于 30，不过有 5 个大于 30 的异常值
- Fri 列中大部分数据都小于 30，只有 1 个异常值大于 40
- Sat 一列中有 3 个大于 40 的异常值
- Sun 一列中有 2 个大于 40 的异常值

也可以传入其他参数：

```
sns.boxplot(x="day", y="total_bill", data=tips, hue="time", saturation=0.1)
```



2.2.2 绘制提琴图

seaborn 中用于绘制提琴图的函数为 `sns.violinplot()`，其语法格式如下：

```
sns.violinplot(data=None, *, x=None, y=None, hue=None, order=None,
                hue_order=None, bw='scott', cut=2, scale='area',
                scale_hue=True, gridsize=100, width=0.8, inner='box',
                split=False, dodge=True, orient=None, linewidth=None,
                color=None, palette=None, saturation=0.75, ax=None, **kwargs)
```

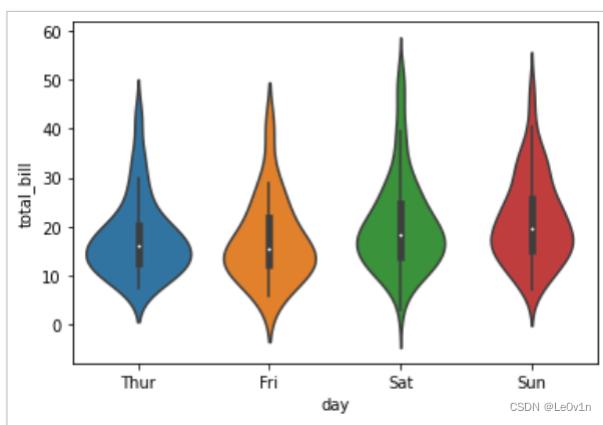
`sns.violinplot()` 是 Seaborn 库中的一个函数，用于绘制小提琴图。它接受以下参数：

- `x`：指定要绘制的数据的 x 轴变量。
- `y`：指定要绘制的数据的 y 轴变量。
- `hue`：指定要使用的颜色变量。
- `data`：指定要使用的数据集。
- `order`：指定分类变量的顺序。
- `hue_order`：指定颜色变量的顺序。
- `inner`：控制小提琴图内部数据点的表示。若为 `box`，则绘制一个微型箱型图；若为 `quartiles`，则显示四分位数线；若为 `point` 或 `stick`，则显示具体数据点或数据线；使用 `None` 则绘制不加修饰的小提琴图。

示例代码如下：

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

上述示例中，使用 seaborn 中内置的数据集绘制了一个提琴图，图中 x 轴的名称为 `day`，y 轴的名称为 `total_bill`，运行结果如图所示。



从图中可以看出，

- Thur 列中位于 5 ~ 25 之间的数值较多
- Fri 列中位于 5 ~ 30 之间的较多
- Sat 列中位于 5 ~ 35 之间的数值较多
- Sun 列中位于 5 ~ 40 之间的数值较多。

2.3 类别内的统计估计

要想查看每个分类的集中趋势，则可以使用条形图和点图进行展示。Seaborn 库中用于绘制这两种图表的具体函数如下：

1. `sns.barplot()` 函数：绘制条形图

2. `sns.pointplot()` 函数：绘制点图

这些函数的 API 与上面那些函数都是一样的，这里只讲解函数的应用，不再过多对函数的语法进行讲解了。

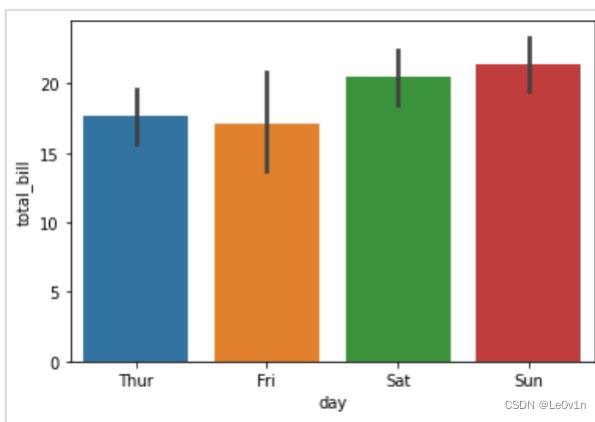
2.3.1 绘制条形图

最常用的查看集中趋势的图形就是条形图。默认情况下，`sns.barplot()` 函数会在整个数据集上使用均值进行估计。若每个类别中有多个类别时（使用了 `hue` 参数），则条形图可以使用引导来计算估计的置信区间（是指由样本统计量所构造的总体参数的估计区间），并使用误差条来表示置信区间。

使用 `sns.barplot()` 函数的示例如下：

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

运行结果如图所示。



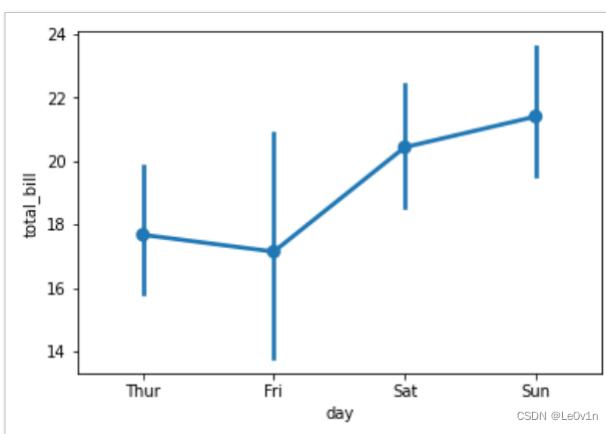
2.3.2 绘制点图

另外一种用于估计的图形是点图，可以调用 `sns.pointplot()` 函数进行绘制，该函数会用高度低计值对数据进行描述，而不是显示完整的条形，它只会绘制点估计和置信区间。

通过 `pointplot()` 函数绘制点图的示例如下：

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

运行结果如图所示。



小结：

- 类别散点图

- `sns.stripplot()`

- 类别内的数据分布

- 1. 箱线图：`sns.boxplot()`

- 2. 小提琴图：`sns.violinplot()`

- 类别内的统计估计

- 1. 条形图：`sns.barplot()`

- 2. 点图：`sns.pointplot()`

3. 综合案例一：NBA 球员数据分析

数据集下载地址 1: [Social Power NBA](#)
 数据集下载地址 2: [《2. Seaborn 及练习案例》配套数据集](#)

3.1 加载数据集

```
data = pd.read_csv("./data/nba_2017_nba_players_with_salary.csv", index_col=0)
data.head()
```

RK	PLAYER	POSITION	AGE	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	GP	MPG	ORPM	DRPM	RPM	WINS_RPM	PIE	PACE	W	SALARY_MILLIONS
0 1	Russell Westbrook	PG	28	34.6	10.2	24.0	0.425	2.5	7.2	...	81	34.6	0.74	-0.47	6.27	17.34	23.0	102.31	46		26.50		
1 2	James Harden	PG	27	36.4	8.3	18.9	0.440	3.2	9.3	...	81	36.4	0.38	-1.57	4.81	15.54	19.0	102.98	54		26.50		
2 3	Isaiah Thomas	PG	27	33.8	9.0	19.4	0.463	3.2	8.5	...	76	33.8	0.72	-3.89	1.83	8.19	16.1	99.84	51		6.59		
3 4	Anthony Davis	C	23	36.1	10.3	20.3	0.505	0.5	1.8	...	75	36.1	0.45	3.90	4.35	12.81	19.2	100.19	31		22.12		
4 6	DeMarcus Cousins	C	26	34.2	9.0	19.9	0.452	1.8	5.0	...	72	34.2	0.56	0.64	4.20	11.26	17.8	97.11	30		16.98		

5 rows × 38 columns

CSDN @LeviDin

看一下数据的统计情况:

```
print(data.shape) # (342, 38)
data.describe()
```

Rk	Age	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	GP	MPG	ORPM	DRPM	RPM	WINS_RPM	PIE	PACE	W	SALARY_MILLIONS
count	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	342.000000	
mean	217.209000	26.444444	21.5725	3.403629	7.775439	0.446006	0.857878	2.400000	0.370715	2.620175	0.581980	21.57207	-0.670202	-0.695789	-0.681915	2.861725	9.189842	98.341953	28.950200	7.294000	
std	136.403138	4.295688	8.880418	2.200672	4.846953	0.078992	0.789010	2.021718	0.134969	1.820714	2.220205	8.604121	2.063257	1.814295	2.502014	3.880014	3.585475	2.970001	14.603878	6.516126	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	-4.400000	-4.400000	-4.400000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	103.000000	22.000000	2.202500	1.800000	4.220200	0.420250	0.200000	0.800000	1.500000	0.500000	0.500000	1.500000	0.420250	0.420250	0.420250	0.200000	0.200000	0.200000	0.200000	0.200000	
50%	205.000000	28.000000	21.600000	3.000000	6.100000	0.450000	0.700000	2.200000	0.540000	2.800000	6.000000	21.600000	-0.680000	-0.100000	1.110000	1.410000	6.700000	96.250000	25.000000	2.162000	
75%	327.750000	29.000000	26.075000	4.700000	9.400000	0.491000	1.400000	3.600000	0.372500	3.700000	7.600000	29.075000	0.257500	1.687500	0.865000	4.407500	10.700000	103.000000	39.000000	11.110000	
max	482.000000	48.000000	37.800000	10.300000	24.000000	0.780000	4.100000	10.000000	1.000000	9.700000	-	82.000000	37.800000	7.270000	6.020000	8.420000	20.430000	23.000000	109.970000	66.000000	

3.2 数据分析

3.2.1 效率值相关性分析

在众多的数据中，有一项名为“RPM”（标识球员的效率值），该数据反映球员在场时对球队比赛获胜的贡献大小。最能反映球员的综合实力。

我们来看一下它与其他数据的相关性：

```
# 截取部分数据
data_corr = data.loc[:, ["RPM", "AGE", "SALARY_MILLIONS", "ORB", "DRB", "TRB", "AST", "STL", "BLK", "TOV", "PF", "POINTS", "GP", "MPG", "ORPM", "DRPM"]]
# 获取两列数据之间的相关性 (在Pandas中，`.corr()`方法用于计算DataFrame中列之间的成对相关性，排除NA / null值)
corr = data_corr.corr()

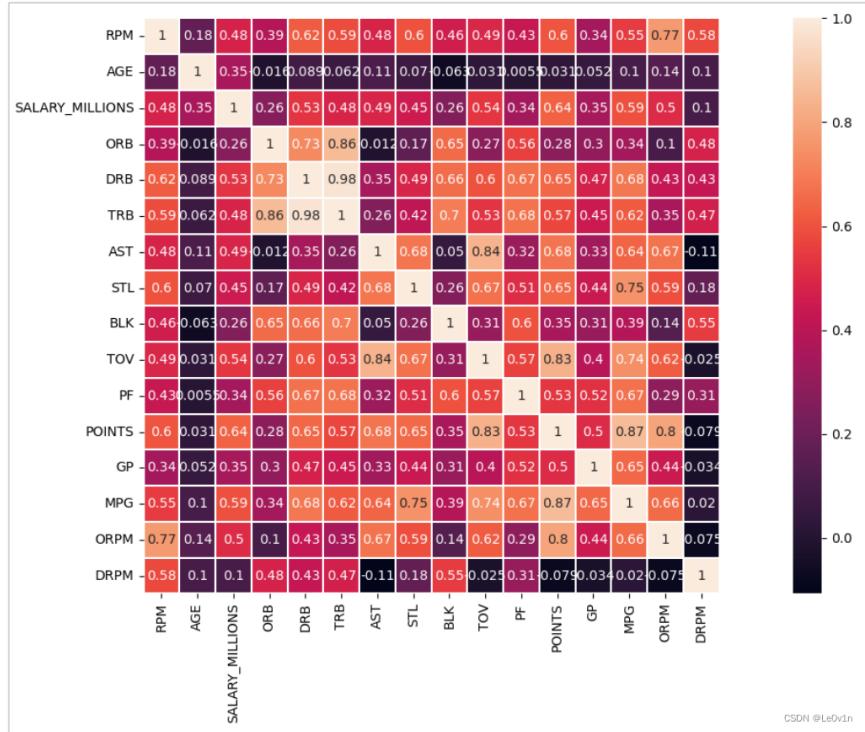
# 绘制热力图
plt.figure(figsize=(20, 8), dpi=100)
sns.heatmap(corr, square=True, linewidths=0.02, annot=True)
"""

seaborn中的heatmap函数是将多维数值变量按数值大小进行交叉热力图展示
square: 结果是否为正方形
linewidths: 用于控制热力图中单元格之间的分隔线宽度
annot: 是否在热力图中显示数据
"""

plt.show()
```

correlation : 英 [kōrə'lēfən] 美 [kō:rə'lēfən]

n. 相关性; 相关; 关联; 相互关系;



在 Pandas 中，`.corr()` 方法用于计算数据框 (DataFrame) 中列之间的成对相关性。默认情况下，它使用皮尔逊相关系数 (Pearson correlation coefficient) 来计算相关性。皮尔逊相关系数是一种度量两个变量之间线性相关程度的方法。它的取值范围为 $[-1, 1]$ ，其中：

- 1 表示完全正相关
- -1 表示完全负相关
- 0 表示无相关性

皮尔逊相关系数 (Pearson correlation coefficient) 是用来反映两个变量线性相关程度的统计量。它的计算公式为：

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

其中， n 为样本量， x_i 和 y_i 分别为两个变量的观测值， \bar{x} 和 \bar{y} 分别为两个变量的均值。

r 描述的是两个变量间线性相关强弱的程度。 r 的绝对值越大表明相关性越强。

由相关性分析的 heatmap 图可以看出，RPM 值与年龄的相关性最弱，与“进攻效率值 - ORPM”、“场均得分 - POINTS”、“场均抢断数 - STL”等比赛技术数据的相关性最强。

我在接下来的分析中将把 RPM 作为评价一个球员能力及状态的直观反应因素之一。

3.2.2 球员数据分析

3.2.2.1 基本分析

此处练习了一下 Pandas 基本的数据框相关操作，包括提取部分列、`head()` 展示、排序等，简单通过几个维度的展示，笼统地看一下 16 ~ 17 赛季那些球员冲在联盟的最前头。

```
# 薪资最高的10名球员
data.loc[:, ["PLAYER", "RPM",
           "SALARY_MILLIONS", "AGE", "MPG"]].sort_values(by="SALARY_MILLIONS", ascending=False).head(10)
```

	PLAYER	RPM	SALARY_MILLIONS	AGE	MPG
6	LeBron James	8.42	30.96	32	37.8
25	Mike Conley	4.47	26.54	29	33.2
67	Al Horford	1.82	26.54	30	32.3
0	Russell Westbrook	6.27	26.50	28	34.6
1	James Harden	4.81	26.50	27	36.4
10	Kevin Durant	5.74	26.50	28	33.4
64	Dirk Nowitzki	0.26	25.00	38	26.4
19	Carmelo Anthony	0.12	24.56	32	34.3
5	Damian Lillard	3.14	24.33	26	35.9
34	Dwyane Wade	-0.91	23.20	35	29.9

```
# 效率最高的10名球员
data.loc[:, ["PLAYER", "RPM",
           "SALARY_MILLIONS", "AGE", "MPG"]].sort_values(by="RPM", ascending=False).head(10)
```

	PLAYER	RPM	SALARY_MILLIONS	AGE	MPG
6	LeBron James	8.42	30.96	32	37.8
37	Chris Paul	7.92	22.87	31	31.5
8	Stephen Curry	7.41	12.11	28	33.4
120	Draymond Green	7.14	15.33	26	32.5
7	Kawhi Leonard	7.08	17.64	25	33.4
44	Nikola Jokic	6.73	1.36	21	27.9
12	Jimmy Butler	6.62	17.55	27	37.0
66	Rudy Gobert	6.37	2.12	24	33.9
0	Russell Westbrook	6.27	26.50	28	34.6
10	Kevin Durant	5.74	26.50	28	33.4

```
# 出场时间最高的10名球员
data.loc[:, ["PLAYER", "RPM",
           "SALARY_MILLIONS", "AGE", "MPG"]].sort_values(by="MPG", ascending=False).head(10)
```

	PLAYER	RPM	SALARY_MILLIONS	AGE	MPG
6	LeBron James	8.42	30.96	32	37.8
32	Zach LaVine	-2.97	2.24	21	37.2
14	Andrew Wiggins	-1.60	6.01	21	37.2
11	Karl-Anthony Towns	2.13	5.96	21	37.0
12	Jimmy Butler	6.62	17.55	27	37.0
17	John Wall	2.26	16.96	26	36.4
1	James Harden	4.81	26.50	27	36.4
3	Anthony Davis	4.35	22.12	23	36.1
5	Damian Lillard	3.14	24.33	26	35.9
13	Paul George	2.58	18.31	26	35.9

3.2.2.2 Seaborn 常用的三个 数据可视化 方法

一、单变量

我们先利用 seaborn 中的 `histplot()` 来分别看一下球员薪水、效率值、年龄这三个信息的分布情况。

```
# 分布及核密度展示
sns.set_style("darkgrid") # 设置seaborn的面板风格

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20, 4), dpi=100)

sns.histplot(data["SALARY_MILLIONS"], kde=True, ax=axes[0])
# axes[0].set_xticks(np.linspace(0, 40, 9)) # 把0-40之间分成9个间隔 (包含0和40)
axes[0].set_ylabel("Salary", size=10)

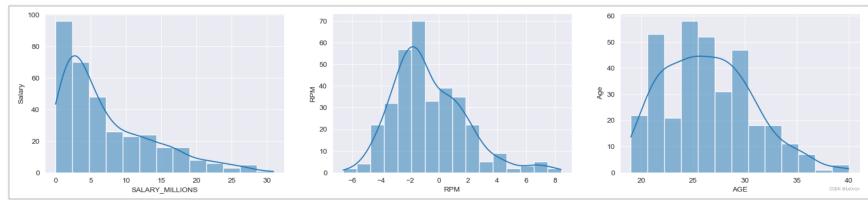
sns.histplot(data["RPM"], kde=True, ax=axes[1])
```

```

# axes[1].set_xticks(np.linspace(-10, 10, 9))
axes[1].set_ylabel("RPM", size=10)

sns.histplot(data["AGE"], kde=True, ax=axes[2])
# axes[2].set_xticks(np.linspace(20, 40, 11))
axes[2].set_ylabel("Age", size=10)
plt.show()

```



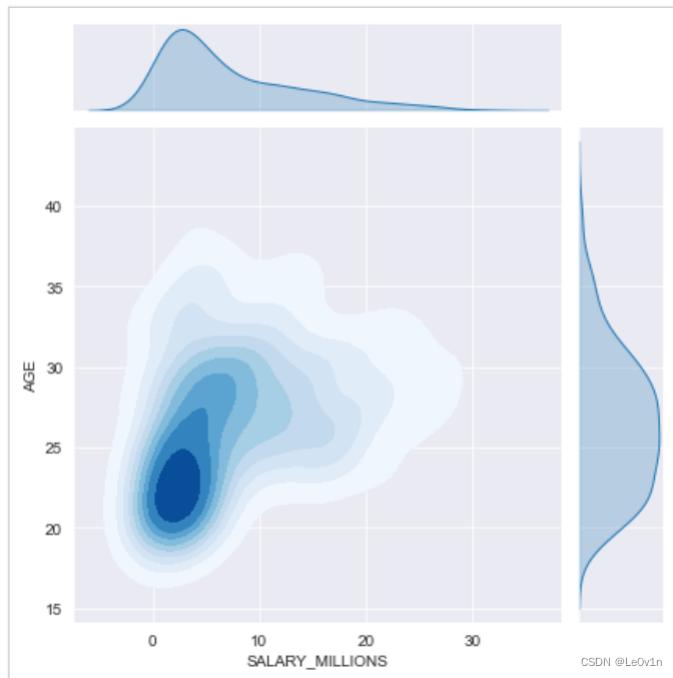
可见年龄和效率值更符合正态分布，而球员薪水更像一个偏态分布，拿高薪的球员占据较小的比例。这些与我们的主观感受基本一致，那么这些变量之间是否有什么隐藏的关系呢？这里可以用 seaborn 中的 `jointplot()` 绘制双变量之间的关系；使用 `pairplot()` 绘图展示多个变量之间的关系。

二、双变量

```

# 使用jointplot()查看年龄和薪资之间的关系
plt.figure(dpi=100)
sns.set_style("darkgrid") # 设置seaborn的面板风格
sns.jointplot(data=data, x="SALARY_MILLIONS", y="AGE", kind="kde", fill=True, cmap="Blues")
plt.show()

```



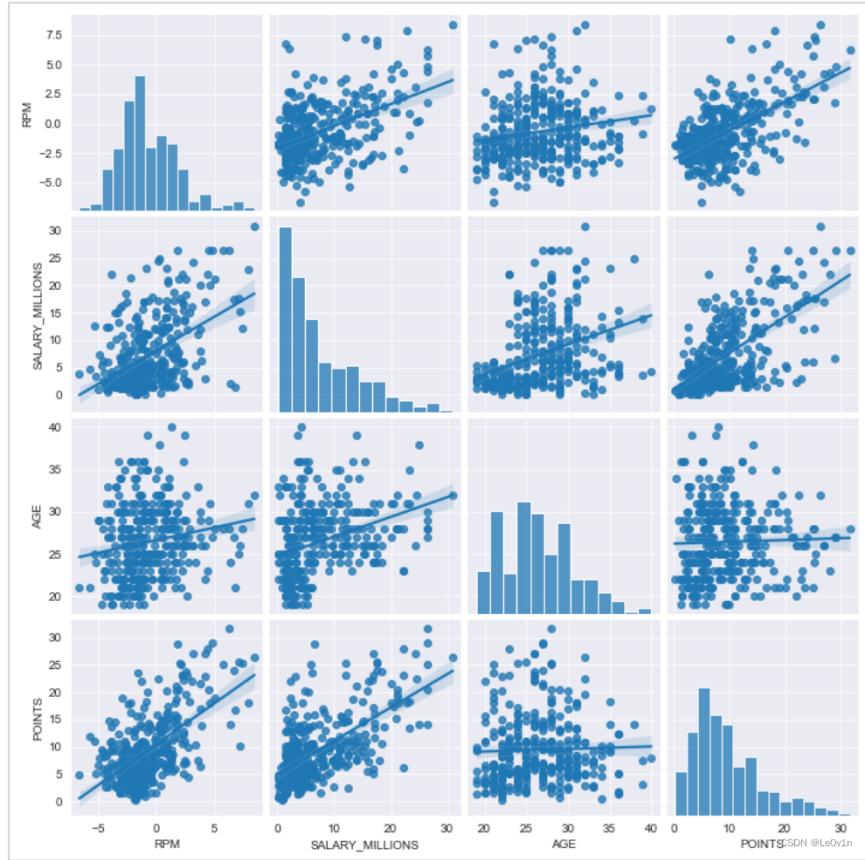
上图展示的是球员薪水与年龄的关系，采用不同的 `kind` 方式 (等高线图 / hex / 散点等)，我们可以整体感受一下年龄和薪水的集中特点，大部分球员集中在 22 ~ 25 岁拿到五百万以下的薪水，当然也有“年少成名” 和 “越老越妖”的情况。

三、多变量

```

# 相关性展示，斜对角为分布展示，可以直观地看变量是否具有线性关系
sns.pairplot(data=data.loc[:, ["RPM", "SALARY_MILLIONS", "AGE", "POINTS"]], kind="reg")
plt.show()

```



上图展示的是球员薪水、效率值、年龄及场均得分四个变量间的两两相关关系，对角线展示的是本身的分布图，由散点的趋势我们可以看出不同特征的相关程度。

整体看各维度的相关性都不是很强，正负值与薪水和场均得分呈较弱的正相关性，而年龄这一属性和其他的变量相关性较弱，究竟是家有一老如有一宝还是廉颇老矣，接下来我们从年龄维度入手进一步分析。

3.2.3 衍生变量的一些可视化实践：以年龄为例

在已有的数据集里想要生成新的变量，例如：把球员按年龄分为老中青三代，可以借助定义一个函数，再利用 `.apply()` 的方式，生成新的变量。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("./data/nba_2017_nba_players_with_salary.csv", index_col=0)

# 思路：根据已有变量生成新的变量
data["avg_point"] = data["POINTS"] / data["MP"] # 每分钟平均得分

# 分割年龄
def age_cut(df):
    if df.AGE <= 24:
        return "Young"
    elif df.AGE >= 30:
        return "Old"
    else:
        return "Gold"

data["age_cut"] = data.apply(func=lambda x: age_cut(x), axis=1) # 球员是否处于黄金年龄
data["count"] = 1 # 计数

data.head()
```

Rk	PLAYER	POSITION	AGE	MP	FG	FGA	FG%	3P	3PA	...	DRPM	RPM	WINS_RPM	PIE	PACE	W	SALARY_MILLIONS	avg_point	age_cut	count
0	1 Russell Westbrook	PG	28	34.6	10.2	24.0	0.425	2.5	7.2	...	-0.47	6.27	17.34	23.0	102.31	46	26.50	0.913295	Gold	1
1	2 James Harden	PG	27	36.4	8.3	18.9	0.440	3.2	9.3	...	-1.57	4.81	15.54	19.0	102.98	54	26.50	0.798451	Gold	1
2	3 Isaiah Thomas	PG	27	33.8	9.0	19.4	0.463	3.2	8.5	...	-3.89	1.83	8.19	16.1	99.84	51	6.59	0.855030	Gold	1
3	4 Anthony Davis	C	23	36.1	10.3	20.3	0.505	1.8	...	3.90	4.35	12.81	19.2	100.19	31	22.12	0.775623	Young	1	
4	6 DeMarcus Cousins	C	26	34.2	9.0	19.9	0.452	1.8	5.0	...	0.64	4.20	11.26	17.8	97.11	30	16.96	0.789474	Gold	1

5 rows x 41 columns

既然得到了老中青三代的标签，我们来看一下不同年龄段球员的 RPM(正负值) 与薪水之间的关系如何。

```
# 球员薪资与效率值——按年龄段来看
sns.set_style("darkgrid") # 设置seaborn的面板风格
plt.figure(figsize=(8, 8), dpi=100)
plt.title("RPM and SALARY", size=15)
```

```

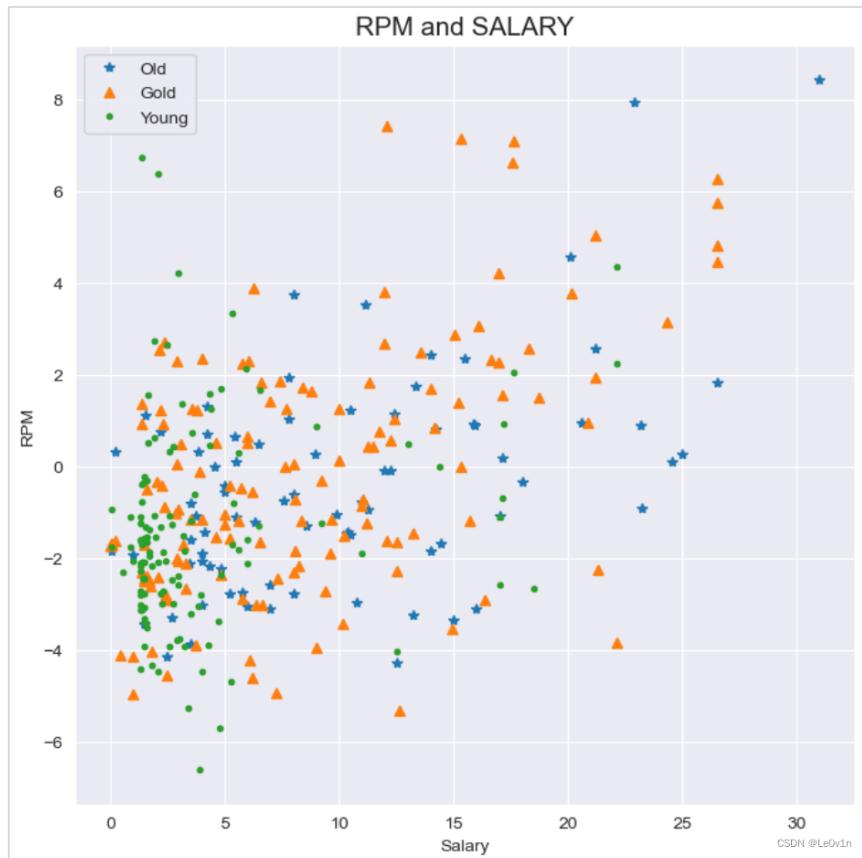
x1 = data.loc[data.age_cut == "Old"].SALARY_MILLIONS
y1 = data.loc[data.age_cut == "Old"].RPM
plt.plot(x1, y1, '+', label="Old")

x2 = data.loc[data.age_cut == "Gold"].SALARY_MILLIONS
y2 = data.loc[data.age_cut == "Gold"].RPM
plt.plot(x2, y2, '^', label="Gold")

x3 = data.loc[data.age_cut == "Young"].SALARY_MILLIONS
y3 = data.loc[data.age_cut == "Young"].RPM
plt.plot(x3, y3, '.', label="Young")

plt.xlabel("Salary")
plt.ylabel("RPM")
plt.legend()
plt.show()

```



点图横坐标为球员薪水，纵坐标为效率值。可以观测到：

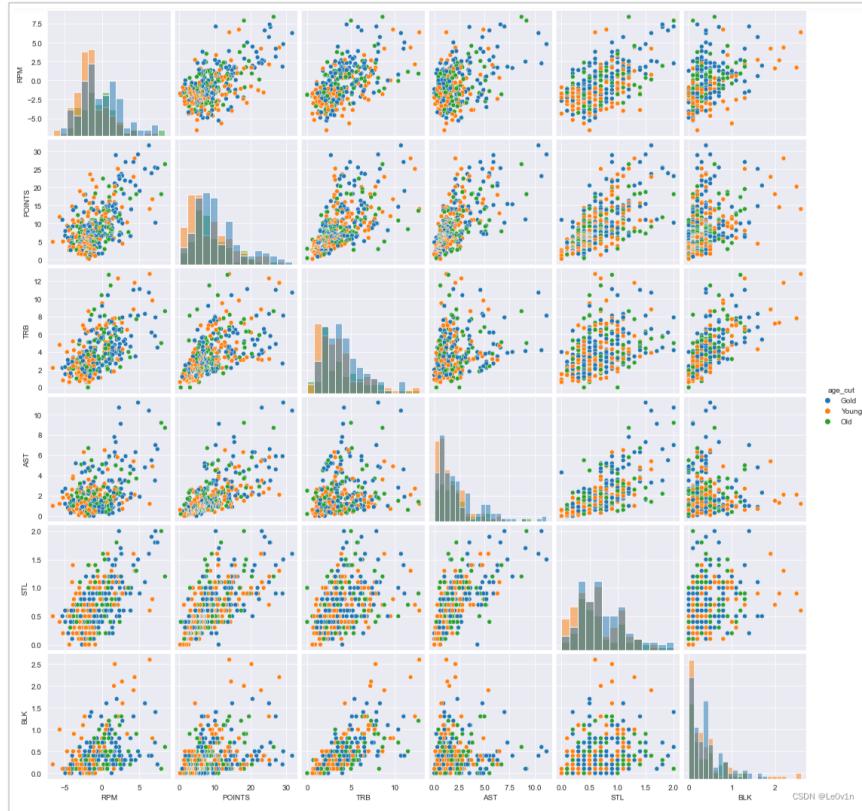
- 绝大部分的年轻球员拿着较低的薪水，数据非常集中。有两个离群点，是上文提到的戈贝尔和约基奇，两个小兄弟前途无量啊。
- 黄金年龄的球员和老球员的数据相对发散，黄金年龄球员薪水与效率值正相关性更强。第一集团有几个全明星排头兵。
- 老球员过了呼风唤雨的年纪，运动状态有所下滑，“高薪低效”的球员也稍微多一些。

用 `pairplot()` 的方法看一下老中青三代各技术统计的分布情况：

```

sns.pairplot(data.loc[:, ["RPM", "POINTS", "TRB", "AST", "STL", "BLK", "age_cut"]], hue="age_cut", diag_kind="hist")
plt.show()

```



3.3.4 球队数据分析

3.3.4.1 球队薪资排行

将数据按球队分组，平均薪水降序排列，看一下联盟十大土豪球队：

```
# 按球队分组
data_group = data.groupby(by=["TEAM"], as_index=False).agg({"SALARY_MILLIONS": np.mean,
                                                               "RPN": np.mean,
                                                               "PLAYER": np.size})
data_group = data_group.loc[data_group.PLAYER > 5] # 不考虑在赛季中转会的球员
data_group.sort_values(by="SALARY_MILLIONS", ascending=False).head(10)
```

TEAM	salary_millions	RPM	PLAYER
9 CLE	17.095000	2.566667	6
18 GS	12.701429	3.478571	7
43 POR	9.730000	-1.260000	10
48 WSH	9.628889	-0.506667	9
39 ORL	9.490000	-2.066667	9
44 SA	9.347273	0.901818	11
26 MEM	8.705000	-0.854167	12
35 NY	8.612727	-1.182727	11
11 DAL	8.480000	-1.037143	7
24 LAC	8.266000	0.319000	CSDN @Leon

骑士队和勇士队已超高的薪水排在这份榜单的前两名，群星璀璨的他们最终在季后赛中一路厮杀，双双闯入分区决赛。

排在第三的开拓者有 10 名球员上榜，可谓后补活力充沛。球队薪金结构的健康与否对球队的发展至关重要。

3.3.4.2 球队年龄结构

先胖不算胖，后胖压倒炕，优质的年轻球员储备是保持球队竞争性的关键。

我们按照分球队分年龄段，上榜球员降序排列，如上榜球员数相同，则按效率值降序排列。

```
# agg的基本使用  
data_group_2 = data.groupby(by=["age_cut"], as_index=False).agg({"SALARY_MILLIONS": np.mean,  
                                                               "RPN": np.mean,  
                                                               "PLAYER": np.size})  
  
data_group_2
```

	age_cut	salary_millions	RPM	PLAYER
0	Gold	8.636667	-0.152536	138
1	Old	9.936790	-0.460741	81
2	Young	4.047236	-1.421220	123

```
# 分组操作，按场上位置
data_group_2 = data.groupby(by=["TEAM", "age_cut"], as_index=False).agg({"SALARY_MILLIONS": np.mean,
                                                               "RPM": np.mean,
                                                               "PLAYER": np.size})
data_group_2.sort_values(by=["PLAYER", "RPM"], ascending=False).head(10)
```

	TEAM	age_cut	SALARY_MILLIONS	RPM	PLAYER
14	CHA	Young	3.835000	-0.362500	8
9	BOS	Gold	7.034286	0.647143	7
105	TOR	Young	4.158571	-0.555714	7
11	BOS	Young	2.337143	-1.821429	7
67	MIN	Gold	5.560000	0.828333	6
32	DEN	Young	2.181667	-0.206667	6
36	DET	Gold	7.638333	-0.386667	6
30	DEN	Gold	8.336667	-0.586667	6
63	MIL	Gold	9.708333	-0.625000	6
70	NO	Gold	6.720000	-0.738333	6

3.3.4.2 球队综合实力分析

最后我们来看看球队综合实力：

按照效率值降序排列前 10 名球队的相关信息如下：

```
# 数据可视化，按球队
data_group_3 = data.groupby(by=["TEAM"], as_index=False).agg({"SALARY_MILLIONS": np.mean,
                                                               "RPM": np.mean,
                                                               "PLAYER": np.size,
                                                               "POINTS": np.mean,
                                                               "eFG%": np.mean,
                                                               "MPG": np.mean,
                                                               "AGE": np.mean,
                                                               })
data_group_3 = data_group_3.loc[data_group_3.PLAYER > 5]
data_group_3.sort_values(by=["RPM"], ascending=False).head(10)
```

	TEAM	SALARY_MILLIONS	RPM	PLAYER	POINTS	eFG%	MPG	AGE
18	GS	12.701429	3.478571	7	14.528571	0.575143	26.700000	28.714286
9	CLE	17.095000	2.566667	6	15.883333	0.555833	29.766667	28.000000
44	SA	9.347273	0.901818	11	9.818182	0.524182	21.472727	29.545455
24	LAC	8.266000	0.319000	10	8.740000	0.462800	18.700000	28.600000
38	OKC	8.060000	0.255556	9	10.677778	0.518111	22.566667	25.666667
47	UTAH	6.471667	-0.107500	12	10.233333	0.515917	24.008333	26.666667
6	CHA	6.779231	-0.277692	13	8.700000	0.447923	20.884615	25.076923
46	TOR	5.668000	-0.319000	10	5.800000	0.497800	17.170000	24.200000
31	NO	8.044444	-0.327778	9	9.411111	0.523222	24.322222	27.444444
5	BOS	6.142667	-0.426667	15	8.426667	0.565067	18.886667	24.666667

勇士 (GS) 和骑士 (CLE) 占据前两名的位置，效率值反映球队实力的事实情况。老马刺 (SA) 排名第三，平均年龄达 29.5 岁排名第一，更新血液迫在眉睫。雷霆 (OKG) 由于大减少的存在能排在第 5 位，各项数据中规中矩。

利用箱线图和小提琴图看着 10 支球队的相关数据。

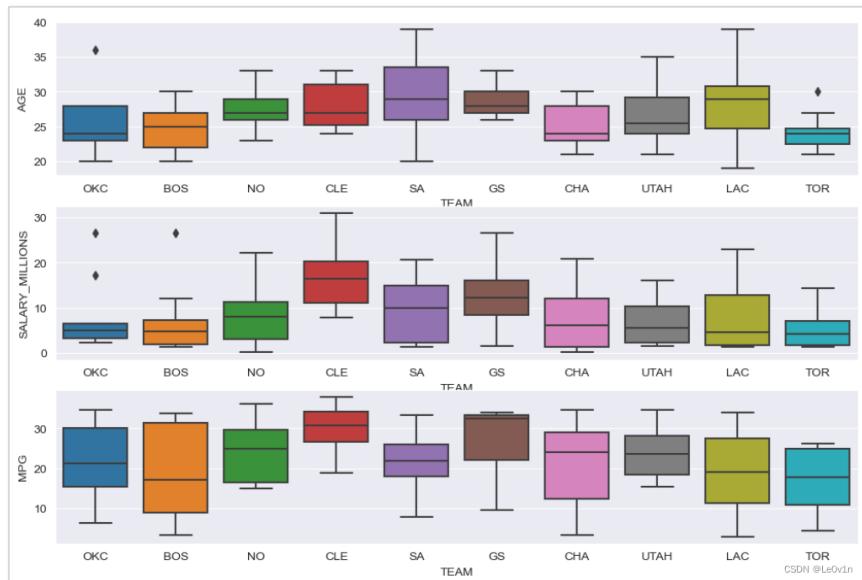
```
# 利用箱型图和小提琴图看着10支球队的相关数据
# 箱型图
sns.set_style("darkgrid") # 设置seaborn的面板风格
data_group_4 = data[data["TEAM"].isin(["GS", "CLE", "SA", "LAC", "OKC", "UTAH", "CHA",
                                       "TOR", "NO", "BOS"])]
```

```
fig, axes = plt.subplots(3, 1, figsize=(12, 8), dpi=100)
sns.boxplot(data=data_group_4, x="TEAM", y="AGE", ax=axes[0])
sns.boxplot(data=data_group_4, x="TEAM", y="SALARY_MILLIONS", ax=axes[1])
```

```

sns.boxplot(data=data_group_4, x="TEAM", y="MPG", ax=axes[2])
plt.show()

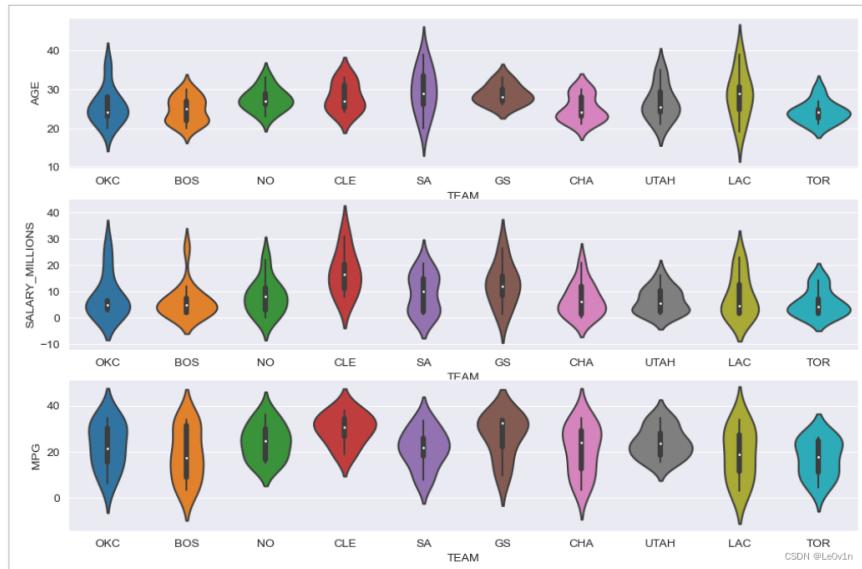
```



```

# 小提琴图
sns.set_style("darkgrid") # 设置seaborn的面板风格
data_group_4 = data[data["TEAM"].isin(["GS", "CLE", "SA", "LAC", "OKC", "UTAH", "CHA",
                                      "TOR", "NO", "BOS"])]
fig, axes = plt.subplots(3, 1, figsize=(12, 8), dpi=100)
sns.violinplot(data=data_group_4, x="TEAM", y="AGE", ax=axes[0])
sns.violinplot(data=data_group_4, x="TEAM", y="SALARY_MILLIONS", ax=axes[1])
sns.violinplot(data=data_group_4, x="TEAM", y="MPG", ax=axes[2])
plt.show()

```



从年龄结构看，老马刺年龄跨度最大，年龄中位数最高。猛龙队最年轻且年龄跨度最小，后劲十足。

从球队薪金看，勇士和骑士最高，俄村雷霆在失去杜兰特后栽了大跟头，薪金健康情况堪忧。从出场时间看。骑士队最高且跨度低。小团体战斗能力出众。

从得分来看，骑士和勇士整体出众。雷霆的威少、绿军的小托马斯、醍醐的浓眉哥以及马刺的伦纳德均是各队的离群点，双拳难敌四手。

从命中率看，命中率各队非常集中，绿凯的小托马斯作为地表最强 175 远远高于其他人。从效率值看，骑士和勇士是大赢家。

4. 综合案例二：北京租房数据统计分析

数据集下载地址：[《2. Seaborn 及练习案例》配套数据集](#)

学习目标：

- 掌握 Pandas 的读写操作
- 会使用预处理技术过滤数据

- 会使用 Matplotlib 库绘制各种图表

- 会基于数据进行独立分析

近年来随着经济的快速发展，一线城市的资源和就业机会吸引了很多外来人口，使其逐渐成为人口密集的城市之一。据统计，2017 年北京市常住外来人口已经达到了 2170.7 万人，其中绝大多数人是以租房的形式解决居住问题。

本文将租房网站上北京地区的租房数据作为参考，运用前面所学到的数据分析知识，带领大家一起来分析真实数据，并以图表的形式得到以下统计指标：

1. 统计每个区域的房源总数量，并使用热力图分析房源位置分布情况。
2. 使用条形图分析哪种户型的数量最多、更受欢迎。
3. 统计每个区域的平均租金，并结合柱状图和折线图分析各区域的房源数量和租金情况。
4. 统计面积区间的市场占有率，并使用饼图绘制各区间所占的比例。

4.1 数据集基本介绍

目前网络上有很多的租房平台，比如自如、爱屋吉屋、房天下、链家等，其中，链家是目前市场占有率最高的公司，通过链家平台可以便捷且全面地提供可靠的房源信息。

通过网络爬虫技术，爬取链家网站中列出的租房信息（爬取结束时间为 2018 年 9 月 10 日），具体包括所属区域、小区名称、房屋、价格、房屋面积、户型。需要说明的是，链家官网上并没有提供平谷、怀柔、密云、延庆等偏远地区的租房数据，所以本案例的分析不会涉及这四个地区。

将爬到的数据下载到本地，并保存在 `链家北京租房数据.csv` 文件中，打开该文件后可以看到里面有很多条（本案例爬取的数据共计 8224 条）信息，具体如下图所示。

	A	B	C	D	E
1	区域	小区名称	户型	面积(平方米)	价格(元/月)
2	东城	万国城MC	1室0厅	59.11平米	10000
3	东城	北官厅胡同	3室0厅	56.92平米	6000
4	东城	和平里三区	1室1厅	40.57平米	6900
5	东城	菊儿胡同	2室1厅	57.09平米	8000
6	东城	交道口北胡同	1室1厅	42.67平米	5500
7	东城	西营房胡同	2室1厅	54.48平米	7200
8	东城	地坛北门胡同	1室1厅	33.76平米	6000
9	东城	安外东河沿胡同	1室1厅	37.62平米	5600
10	东城	清水苑胡同	1室1厅	45.61平米	6200
11	东城	李村东里胡同	2室1厅	57.35平米	5700
12	东城	幸福北里胡同	2室1厅	51.15平米	6500
13	东城	保利蔷薇花园	2室1厅	97.11平米	10000
14	东城	东板桥西胡同	2室1厅	52.86平米	5800
15	东城	本家润园胡同	2室1厅	63.09平米	7800
16	东城	营房西街胡同	2室1厅	62.95平米	7500
17	东城	新景家园胡同	1室1厅	57.24平米	7500
18	东城	东花市北胡同	2室1厅	85.36平米	8800
19	东城	幸福家园胡同	5室2厅	226.86平米	29000
20	东城	景泰西里胡同	1室1厅	60.3平米	6200
21	东城	海景名苑胡同	1室1厅	70.86平米	12000
22	东城	和平新城胡同	2室1厅	122.76平米	14500
23	东城	太华公寓胡同	2室2厅	152.24平米	17000
24	东城	官书院胡同	2室1厅	92.01平米	16000
25	东城	幸福家园胡同	2室1厅	65.25平米	7800
26	东城	安外大街胡同	1室1厅	33.77平米	5500
27	东城	中海紫御公馆胡同	2室2厅	99.15平米	12000

4.2 数据读取

准备好数据后，我们便可以使用 Pandas 读取保存在 CSV 文件的数据，并将其转换成 DataFrame 对象展示，便于后续操作这些数据。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

file_data = pd.read_csv("./data/链家北京租房数据.csv", encoding="gbk")
file_data.head()
```

	区域	小区名称	户型	面积(㎡)	价格(元/月)
0	东城	万国城MOMA	1室0厅	59.11平米	10000
1	东城	北官厅胡同2号院	3室0厅	56.92平米	6000
2	东城	和平里三区	1室1厅	40.57平米	6900
3	东城	菊儿胡同	2室1厅	57.09平米	8000
4	东城	交道口北二条35号院	1室1厅	42.67平米	5500

CSDN @Leovin

```
file_data.shape # (8223, 5)
file_data.info()
"""
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8223 entries, 0 to 8222
Data columns (total 5 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   区域      8223 non-null   object 
 1   小区名称    8223 non-null   object 
 2   户型      8223 non-null   object 
 3   面积(㎡)    8223 non-null   object 
 4   价格(元/月) 8223 non-null   int64  
dtypes: int64(1), object(4)
memory usage: 321.3+ KB
"""

file_data.describe()
```

	价格(元/月)
count	8223.000000
mean	9512.297823
std	9186.752612
min	566.000000
25%	4800.000000
50%	6800.000000
75%	10000.000000
max	150000.000000

4.3 数据预处理

尽管从链家官网上直接爬取下来的数据大部分是比较规整的，但或多或少还是会存在一些问题，不能直接用做数据分析。为此，在使用前需要对这些数据进行一系列的检测与处理，包括处理重复值和缺失值、统一数据类型等，以保证数据具有更高的可用性。

4.3.1 重复值和空值处理

预处理的前两步就是检查缺失值和重复值。如果希望检查准备的数据中是否存在重复的数据，则可以通过 Pandas 中的 `duplicated()` 方法完成。接下来，通过 `duplicated()` 方法对北京租房数据进行检测，只要有重复的数据就会映射为 True，具体代码如下。

```
# 重复数据检测
file_data.duplicated()
```

结果如下：

```
0    False
1    False
2    False
3    False
4    False
...
8218  False
8219  False
8220  False
8221  False
8222  False
Length: 8223, dtype: bool
```

由于数据量相对较多，所以在 Jupyter Notebook 工具中有一部分数据会省略显示，但是从输出结果中仍然可以看到有多条返回结果为 True 的数据，这表明有重复的数据。这里，处理重复数据的方式是将其删除。接下来，使用 `drop_duplicates()` 方法直接删除重复的数据，具体代码如下。

```
# 删除重复数据
file_data.drop_duplicates(inplace=True)
```

```
file_data.duplicated()
```

结果：

```
0    False
1    False
2    False
3    False
4    False
...
8218  False
8219  False
8220  False
8221  False
8222  False
Length: 5773, dtype: bool
```

与上一次输出的行数相比，可以很明显地看到减少了很多条数据，只剩下了 5773 条数据。

对数据重复检测完成之后，便可以检测数据中是否存在缺失值，我们可以直接使用 `dropna()` 方法检测并删除缺失的数据，具体代码如下。

```
print(file_data.shape) # (5773, 5)

# 删除缺失数据
file_data.dropna(inplace=True)
print(file_data.shape) # (5773, 5)
```

经过缺失数据检测之后，可以发现当前数据的总行数与之前相比没有发生任何变化。因此我们断定准备好的数据中并不存在缺失的数据。

4.3.2 数据转换类型

在这套租房数据中，“面积 (m² m² m²)”一列的数据里面有中文字符，说明这一列数据都是字符串类型的。

	区域	小区名称	户型	面积(m²)	价格(元/月)
0	东城	万国城MOMA	1室0厅	59.11平米	10000
1	东城	北官厅胡同2号院	3室0厅	56.92平米	6000
2	东城	和平里三区	1室1厅	40.57平米	6900
3	东城	菊儿胡同	2室1厅	57.09平米	8000
4	东城	交道口北二条35号院	1室1厅	42.67平米	5500

为了方便后续对面积数据进行数学运算，所以需要将“面积 (m² m² m²)”一列的数据类型转换为 float 类型，具体代码如下。

```
# 创建一个空数组
data_new = np.array([])

# 取出“面积”一列数据，将每个数据末尾的中文字符去掉
data = file_data["面积(m²)"].values

print(type(data)) # <class 'numpy.ndarray'>

for i in data:
    data_new = np.append(arr=data_new, values=np.array(i[:-2])) # 不要最后两个字符

# 通过astype()方法将str类型转换为float64类型
data = data_new.astype(np.float64)

# 用新的数据替换
file_data.loc[:, "面积(m²)"] = data

file_data
```

	区域	小区名称	户型	面积(m²)	价格(元/月)
0	东城	万国城MOMA	1室0厅	59.11	10000
1	东城	北官厅胡同2号院	3室0厅	56.92	6000
2	东城	和平里三区	1室1厅	40.57	6900
3	东城	菊儿胡同	2室1厅	57.09	8000
4	东城	交道口北二条35号院	1室1厅	42.67	5500

除此之外，在“户型”一列中，大部分数据显示的是“室厅”，只有个别数据显示的是“* 房间 * 卫”（比如索引 8219 对应的一行）。为了方便后期的使用，需要将“房间”替换成“室”，以保证数据的一致性。

接下来，使用 Pandas 的 `replace()` 方法完成替换数据的操作，具体代码如下。

```
# 获取“户型”一列数据
housetype_data = file_data["户型"]
```

```

tmp_lst = []
print(type(housetype_data)) # <class 'pandas.core.series.Series'>

# 通过replace()方法进行替换
for i in housetype_data:
    new_info = i.replace("房间", "室")
    tmp_lst.append(new_info)

# 替换原有数据
file_data.loc[:, "户型"] = tmp_lst

```

通过比较处理前与处理后的数据可以发现，索引为 8219 的户型数据已经由 “4 房间 2 卫” 变成 “4 室 2 卫”，说明数据替换成功。

4.4 图表分析

数据经过预处理以后，便可以用它们来做分析了。为了能够更加直观地看到数据的变化，这里我们采用图表的方式来辅助分析。

4.4.1 房源数量、位置分布分析

如果希望统计各个区域的房源数量，以及查看这些房屋的分布情况，则需要先获取各个区的房源。为了实现这个需求，可以将整个数据按照 “区域” 列进行分组。

为了能够准确地看到各区域的房源数量，这里只需要展示 “区域” 与 “数量” 这两列的数据即可。因此，先创建一个空的 DataFrame 对象，然后再将各个区域计算的总数量作为该对象的数据进行展示，具体代码如下。

```

# 创建一个DF对象，该对象只有两列数据：区域和数量
new_df = pd.DataFrame(data={"区域": file_data["区域"].unique(), "数量": [0]*13})
new_df.head()

```

区域 数量		
0	东城	0
1	丰台	0
2	亦庄开发区	0
3	大兴	0
4	房山	0

接下来，通过 Pandas 的 groupby() 方法将 file_data 对象按照 “区域” 一列进行分组，并利用 count() 方法统计每个分组的数量，具体代码如下。

```

# 按 “区域” 列将file_data分组，并统计每个分组的数量
group_area = file_data.groupby(by="区域").count() # pandas.core.frame.DataFrame
new_df["数量"] = group_area.values
new_df

```

区域 数量		
0	东城	282
1	丰台	577
2	亦庄开发区	147
3	大兴	362
4	房山	180
5	昌平	347
6	朝阳	1597
7	海淀	605
8	石景山	175
9	西城	442
10	通州	477
11	门头沟	285
12	顺义	297

通过 sort_values() 方法对 new_df 对象排序，按照从大到小的顺序进行排列，具体代码如下。

```

# 按 “数量” 从小到大排序
new_df.sort_values(by="数量", ascending=False)

```

区域 数量		
6	朝阳	1597
7	海淀	605
1	丰台	577
10	通州	477
9	西城	442
3	大兴	362
5	昌平	347
12	顺义	297
11	门头沟	285
0	东城	282
4	房山	180
8	石景山	175
2	亦庄开发区	147

通过输出的排序结果可以看出，房源数量位于前的区域分别是朝阳区、海淀区、丰台区。

4.4.2 户型数量分析

随着人们生活水平的提高，以及各住户的生活需求，开发商设计出了各种各样的户型供人们居住。接下来，我们来分析一下户型，统计租房市场中哪种户型的房源数量偏多，并筛选出数量大于 50 的户型。

首先，我们定义一个函数来计算各种户型的数量，具体代码如下。

```
# 定义函数，用于计算各户型的数量
def house_style_total(house_array):
    style_names = np.unique(house_array)
    res = {}

    for style_name in style_names:
        mask = house_array == style_name
        tmp_arr = house_array[mask] # 选取符合条件的数据
        total_num = tmp_arr.size
        res[style_name] = total_num

    return res

# 获取户型数据
house_array = file_data["户型"] # pandas.core.series.Series
house_info = house_style_total(house_array)
print(house_info)
```

结果如下：

```
{'0室0厅': 1, '1室0卫': 10, '1室0厅': 244, '1室1卫': 126, '1室1厅': 844, '1室2厅': 13,
'2室0卫': 1, '2室0厅': 23, '2室1卫': 120, '2室1厅': 2249, '2室2卫': 22, '2室2厅': 265,
'2室3厅': 1, '3室0卫': 3, '3室0厅': 12, '3室1卫': 92, '3室1厅': 766, '3室2卫': 48,
'3室2厅': 489, '3室3卫': 1, '3室3厅': 10, '4室1卫': 15, '4室1厅': 58, '4室2卫': 24,
'4室2厅': 191, '4室3卫': 5, '4室3厅': 9, '4室5厅': 2, '5室0卫': 1, '5室0厅': 1,
'5室1卫': 3, '5室1厅': 7, '5室2卫': 49, '5室3卫': 3, '5室3厅': 24,
'5室4厅': 1, '5室5厅': 1, '6室0厅': 1, '6室1卫': 1, '6室1厅': 1, '6室2厅': 5, '6室3卫': 2,
'6室3厅': 6, '6室4卫': 2, '7室1厅': 1, '7室2厅': 2, '7室3厅': 3, '7室4厅': 1, '8室4厅': 2,
'9室1厅': 2, '9室2厅': 1, '9室5厅': 2}
```

程序输出了一个字典，其中，字典的键表示“户型”的种类，值表示该“户型”的数量。

使用字典推导式将户型数量大于 50 的元素筛选出来，并将筛选后的结果转换成 DataFrame 对象，具体代码如下。

```
# 使用字典推导式
house_type = {key: value for key, value in house_info.items() if value > 50}
print(house_type) # {'1室0厅': 244, '1室1卫': 126, '1室1厅': 844, '2室1卫': 120, '2室1厅': 2249, '2室2卫': 265, '3室1卫': 92, '3室1厅': 766,
show_houses = pd.DataFrame(data={"户型": [x for x in house_type.keys()],
                                    "数量": [x for x in house_type.values()]})
```

户型 数量		
0	1室0厅	244
1	1室1卫	126
2	1室1厅	844
3	2室1卫	120
4	2室1厅	2249
5	2室2厅	265
6	3室1卫	92
7	3室1厅	766
8	3室2厅	489
9	4室1厅	58
10	4室2厅	191

CSDN @Le0v1n

为了能够更直观地看到户型数量间的差异，我们可以使用条形图进行展示。其中，条形图纵轴代表户型种类，横坐标代表数量。代码如下：

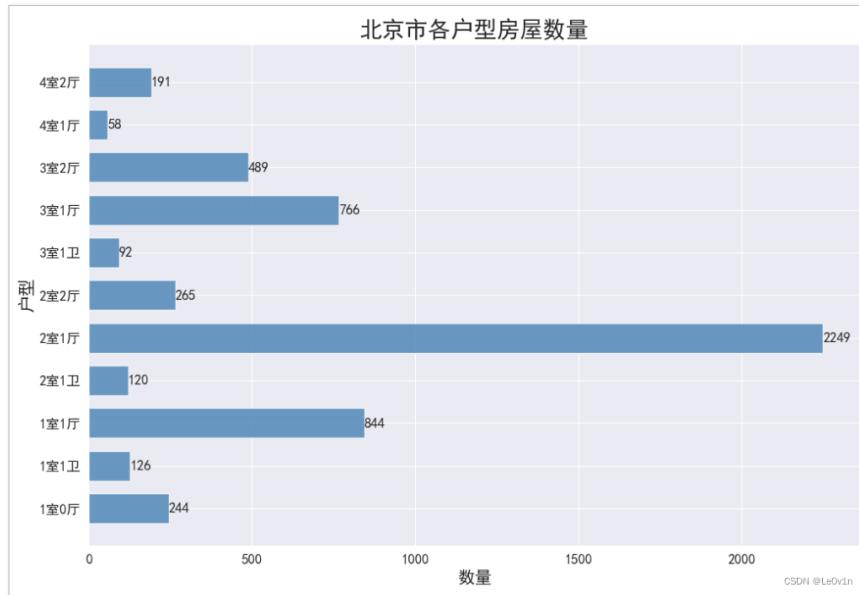
```
from pylab import plt
# 设置中文字体
mpl.rcParams["font.sans-serif"] = ["SimHei"]
# 设置正常显示符号
mpl.rcParams["axes.unicode_minus"] = False

plt.figure(figsize=(12, 8), dpi=100)
a = show_houses["户型"]
b = show_houses["数量"]

# Make a horizontal bar plot.
fig = plt.barh(y=a, width=b, height=0.7, color="steelblue", alpha=0.8)
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel("数量", size=15)
plt.ylabel("户型", size=15)
plt.title("北京市各户型房屋数量", size=20)

# 添加数值
plt.bar_label(container=fig, size=12)

plt.show()
```



通过图上可以清晰地看出，整个租房市场中户型数量较多分别为“2室1厅”、“1室1厅”、“3室1厅”的房屋，其中，“2室1厅”户型的房屋在整个租房市场中是数量最多的。

4.4.3 平均租金分析

为了进一步剖析房屋的情况，接下来，我们来分析一下各地区目前的平均租金情况。计算各区域房租的平均价格与计算各区域户型数量的方法大同小异，首先创建一个 DataFrame 对象，具体代码如下。

```
# 新建一个DataFrame对象，设置房租总额和总面积初始值为0
df_all = pd.DataFrame(data={"区域": file_data["区域"].unique(),
                            "房屋总租金": [0] * 13,
```

```
df_all
```

```
"房屋总面积": [0] * 13})
```

	区域	房屋总租金	房屋总面积
0	东城	0	0
1	丰台	0	0
2	亦庄开发区	0	0
3	大兴	0	0
4	房山	0	0
5	昌平	0	0
6	朝阳	0	0
7	海淀	0	0
8	石景山	0	0
9	西城	0	0
10	通州	0	0
11	门头沟	0	0
12	顺义	0	0

CSDN @Le0v1n

接下来，按照“区域”一列进行分组，然后调用 `sum()` 方法分别对房屋金额和房屋面积执行求和计算，具体代码如下：

```
# 计算房屋的总金额和总面积
sum_price = file_data[“价格(元/月)”].groupby(by=file_data[“区域”]).sum()
sum_area = file_data[“面积(㎡)”].groupby(by=file_data[“区域”]).sum()

df_all[“房屋总租金”] = sum_price.values
df_all[“房屋总面积”] = sum_area.values
df_all
```

	区域	房屋总租金	房屋总面积
0	东城	3945550	27353.99
1	丰台	4404893	50922.79
2	亦庄开发区	1318400	15995.53
3	大兴	2286950	35884.15
4	房山	726750	15275.41
5	昌平	2521515	35972.92
6	朝阳	20281396	166921.72
7	海淀	7279350	57210.39
8	石景山	1156500	13956.67
9	西城	5636975	37141.64
10	通州	2719600	46625.23
11	门头沟	1048300	20258.20
12	顺义	2190900	33668.97

CSDN @Le0v1n

计算出各区域房租总金额和总面积之后，便可以对每平方米的租金进行计算。在 `df_all` 对象的基础上增加一列，该列的名称为“每平方米租金(元)”，数据为求得的每平方米的平均价格，具体代码如下。

```
# 计算各区域每平米房租价格，并保留两位小数
df_all[“每平方米租金(元)”] = round(number=df_all[“房屋总租金”] / df_all[“房屋总面积”], ndigits=2)
df_all
```

	区域	房屋总租金	房屋总面积	每平方米租金(元)
0	东城	3945550	27353.99	144.24
1	丰台	4404893	50922.79	86.50
2	亦庄开发区	1318400	15995.53	82.42
3	大兴	2286950	35884.15	63.73
4	房山	726750	15275.41	47.58
5	昌平	2521515	35972.92	70.09
6	朝阳	20281396	166921.72	121.50
7	海淀	7279350	57210.39	127.24
8	石景山	1156500	13956.67	82.86
9	西城	5636975	37141.64	151.77
10	通州	2719600	46625.23	58.33
11	门头沟	1048300	20258.20	51.75
12	顺义	2190900	33668.97	65.07

CSDN @Le0y1n

为了能更加全面地了解到各个区域的租房数量与平均租金，我们可以将之前创建的 `new_df` 对象 (各区域房源数量) 与 `df_all` 对象进行合并展示。

new_df	区域	数量	区域 房屋总租金 房屋总面积 每平方米租金(元)			
			0	1	2	3
0	东城	282	0	东城	3945550	27353.99
1	丰台	577	1	丰台	4404893	50922.79
2	亦庄开发区	147	2	亦庄开发区	1318400	15995.53
3	大兴	362	3	大兴	2286950	35884.15
4	房山	180	4	房山	726750	15275.41
5	昌平	347	5	昌平	2521515	35972.92
6	朝阳	1597	6	朝阳	20281396	166921.72
7	海淀	605	7	海淀	7279350	57210.39
8	石景山	175	8	石景山	1156500	13956.67
9	西城	442	9	西城	5636975	37141.64
10	通州	477	10	通州	2719600	46625.23
11	门头沟	285	11	门头沟	1048300	20258.20
12	顺义	297	12	顺义	2190900	33668.97

CSDN @Le0y1n

df_all

由于这两个对象中都包含“区域”列，所以这里可以采用主键的方式进行合并，也就是说通过 `merge()` 函数来实现，具体代码如下。

```
# 合并new_df与df_all
df_merge = pd.merge(left=new_df, right=df_all)
df_merge
```

	区域	数量	房屋总租金	房屋总面积	每平方米租金(元)
0	东城	282	3945550	27353.99	144.24
1	丰台	577	4404893	50922.79	86.50
2	亦庄开发区	147	1318400	15995.53	82.42
3	大兴	362	2286950	35884.15	63.73
4	房山	180	726750	15275.41	47.58
5	昌平	347	2521515	35972.92	70.09
6	朝阳	1597	20281396	166921.72	121.50
7	海淀	605	7279350	57210.39	127.24
8	石景山	175	1156500	13956.67	82.86
9	西城	442	5636975	37141.64	151.77
10	通州	477	2719600	46625.23	58.33
11	门头沟	285	1048300	20258.20	51.75
12	顺义	297	2190900	33668.97	65.07

CSDN @LeOvIn

合并完数据以后，就可以借用图表来展示各地区房屋的信息，其中，房源的数量可以用柱状图中的条柱表示，每平方米租金可以用折线图中的点表示，具体代码如下。

```

import matplotlib.pyplot as plt
from pylab import mpl
# 设置中文字体
mpl.rcParams["font.sans-serif"] = ["SimHei"]
# 设置正常显示符号
mpl.rcParams["axes.unicode_minus"] = False

# 准备数据
num = df_merge["数量"]
price = df_merge["每平方米租金(元)"]
area = df_merge["区域"]

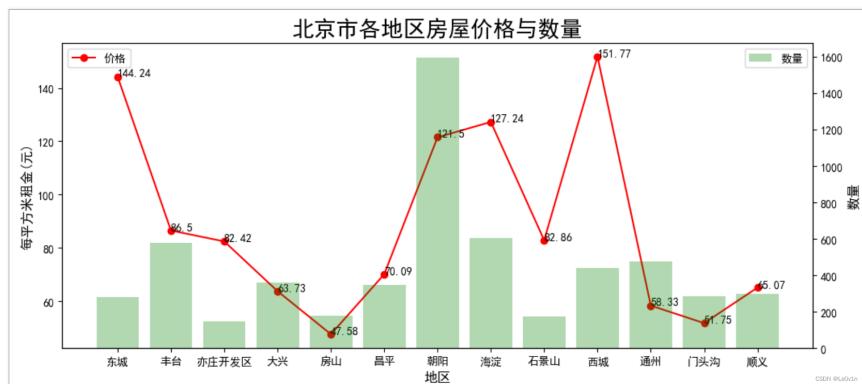
# 折线图
fig, axes = plt.subplots(1, 1, figsize=(12, 5), dpi=100)
axes.plot(area, price, "or-", label="价格")

# 在折线上显示数值
for i in range(len(area)):
    axes.annotate(str(price[i]), xy=(area[i], price[i]))
axes.set_ylabel("每平方米租金(元)", size=12)
axes.legend(loc="upper left")

# 条形图
twin_axes = axes.twinx()
"""
.twinx()方法用于创建一个新的坐标轴，它与原始坐标轴共享x轴。新的坐标轴将覆盖原始坐标轴，并且其刻度将位于右侧。
这样就可以在同一张图上绘制两个具有不同y轴刻度的图形。
"""
twin_axes.bar(area, num, alpha=0.3, color="green", label="数量")
twin_axes.set_ylabel("数量", size=12)
twin_axes.legend(loc="upper right")

plt.title("北京市各地区房屋价格与数量", size=20)
axes.set_xlabel("地区", size=12)
plt.show()

```



4.4.4 面积区间分析

下面我们将房屋的面积数据按照一定的规则划分成多个区间，看一下各面积区间的上情况，便于分析租房市场中哪种房屋类型更好出租，哪个面积区间的租户人数最多。

要想将数据划分为若干个区间，则可以使用 Pandas 中的 `cut()` 函数来实现。首先，使用 `max()` 与 `min()` 方法分别计算出房屋面积的最大值和最小值，具体代码如下。

```
# 查看房屋的最大面积和最小面积
print("房屋最大面积是{}".format(file_data['面积(㎡)'].max())) # 房屋最大面积是1133.0
print("房屋最小面积是{}".format(file_data['面积(㎡)'].min())) # 房屋最小面积是11.63

# 查看房租的最大值和最小值
print("房租最大值是{}".format(file_data['价格(元/月)'].max())) # 房租最大值是150000
print("房租最小值是{}".format(file_data['价格(元/月)'].min())) # 房租最小值是566
```

在这里，我们参照链家网站的面积区间来定义，将房屋面积划分为 8 个区间。然后使用 `describe()` 方法显示各个区间出现的次数（`counts` 表示）以及频率（`freqs` 表示）。具体代码如下。

```
# 面积划分
area_divide = [1, 30, 50, 70, 90, 120, 140, 160, 1200]
area_cut = pd.cut(x=list(file_data["面积(㎡)"]), bins=area_divide) # pandas.core.arrays.categorical.Categorical
area_cut_data = area_cut.describe()
area_cut_data
```

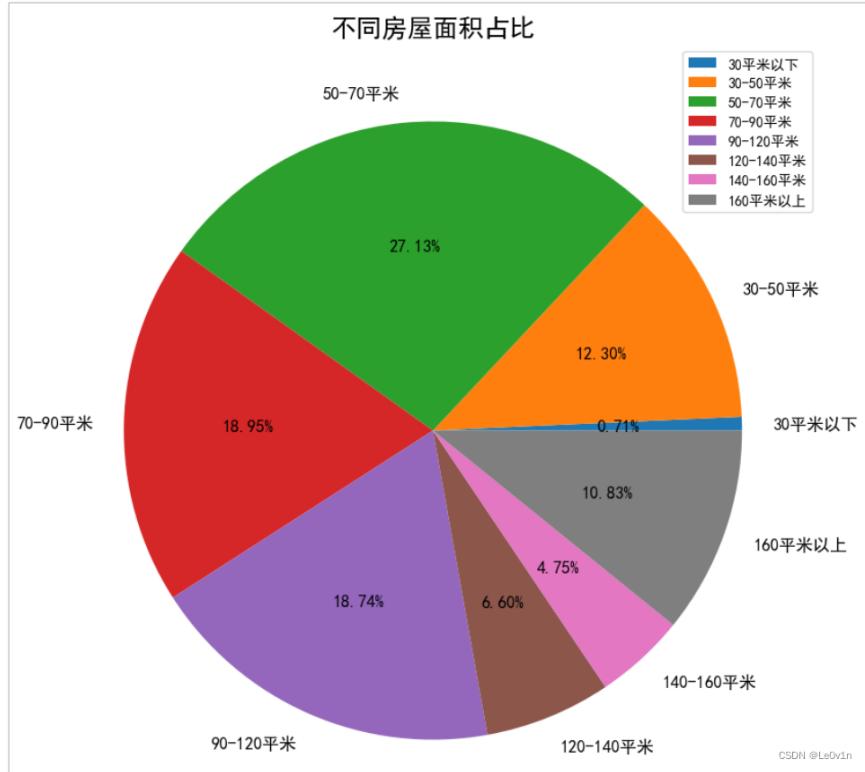
	counts	freqs
categories		
(1, 30]	41	0.007102
(30, 50]	710	0.122986
(50, 70]	1566	0.271263
(70, 90]	1094	0.189503
(90, 120]	1082	0.187424
(120, 140]	381	0.065997
(140, 160]	274	0.047462
(160, 1200]	625	0.108263

接着，使用饼图来展示各面积区间的分布情况，具体代码如下。

```
area_pct = (area_cut_data["freqs"].values) * 100 # pct=percentage/percent
labels = ["30平米以下", "30-50平米", "50-70平米", "70-90平米", "90-120平米",
          "120-140平米", "140-160平米", "160平米以上"]

# 画图
plt.figure(figsize=(20, 10), dpi=100)
"""
`plt.axes()` 函数用于在当前图形中添加一个子图。`aspect` 参数控制子图的纵横比。
当 `aspect=1` 时，子图的纵横比被设置为 1:1，这意味着子图的单位长度在 x 轴和 y 轴上相等。
"""
plt.axes(aspect=1)

autopct它用于控制 `plt.pie()` 函数中每个扇形的数值显示格式。可以使用字符串格式化语法来指定显示格式
例如 `'%1.1f%'` 表示显示一位小数的百分比。
"""
plt.pie(x=area_pct, labels=labels, autopct="%1.2f%", textprops={"fontsize": 12})
plt.title(label="不同房屋面积占比", fontsize=18)
plt.legend(loc="upper right")
plt.show()
```



通过上图可以看出，50 ~ 70 平方米的房屋在租房市场中占有率最大。总体看来，租户主要以 120 平方米以下的房屋为租住对象，其中 50 ~ 70 平方米以下的房屋为租户的首选对象。

全文完

本文由 简悦 SimpRead 优化，用以提升阅读体验

使用了 全新的简悦词法分析引擎 beta，[点击查看详细说明](#)

