第 3 章

描述性分析

视频讲解

3.1 数据

数据(data)是统计学、数据挖掘、大数据分析和人工智能等领域的研究必备前提。根据维基百科的定义,数据是指未经过处理的原始记录。当然,实际工作中的源数据不免会有噪声数据,本书涉及的原始记录是相对的,既不含有噪声数据,也不存在异常数据值,除非有特殊说明。按照数据类型进行划分,数据包含分类数据和数值数据两种。

定义 3.1 分类数据

不可(直接)测量的数据称为分类数据。如外貌、天气、出生地、英语等级等。

分类数据可以在一定条件下作为数值数据进行处理。如某校学生 A 想调研学校的学生对 Python 语言的喜爱程度,其中一项问题设计,如表 3.1 所示。

表 3.1 某校学生对 Python 语言的调研

序号	问　　题	非常有趣	有趣	一般	无趣	非常无趣
1	你认为 Python 有趣吗					

调研完成后将问卷进行整理,剔除无效问卷,再录入数据,对选项通过下面的方式进行数值化处理,即可视为数值数据,便可进行处理和分析了。

```
score_dict = {"非常有趣": 5, "有趣": 4, "一般": 3, "无趣": 2, "非常无趣": 1}
```

待问卷数据录入后,可通过离差公式$\left(\mathrm{dev}(x_i)=50+10\times\frac{x_i-\bar{x}}{\sigma}\right)$将其转换为数值数

据。假设序号 1 问题的所有有效数据的均值为 3,标准差为 2,某学生选择了有趣,此得分为 50+10×((4−3)/2)=55.0。

定义 3.2　数值数据

可测量的数据称为数值数据。如气温、国内生产总值、身高、体重等。

数值(型)数据在日常工作处理中占主导地位,但随着自然语言(NLP)的兴起,关于文本数据的研究也日益剧增。本书主要集中探索数值(型)数据,读者若对文本数据感兴趣,可以参照一些关于自然语言处理的书籍。另外,数值数据也可以进行细分,比如可分为时间序列和非时间序列。

另外,在解决实际问题时经常存在既含有分类数据也含有数值数据的情况,因此想成为一名优秀的数据科学家,不仅需要具备相关的数学知识、编程能力,还要具备丰富的经验。从事数据科学多年的工作者,对工作流程都有清晰的流程,这里给出一个关于数据分析的大致流程图,如图 3.1 所示。

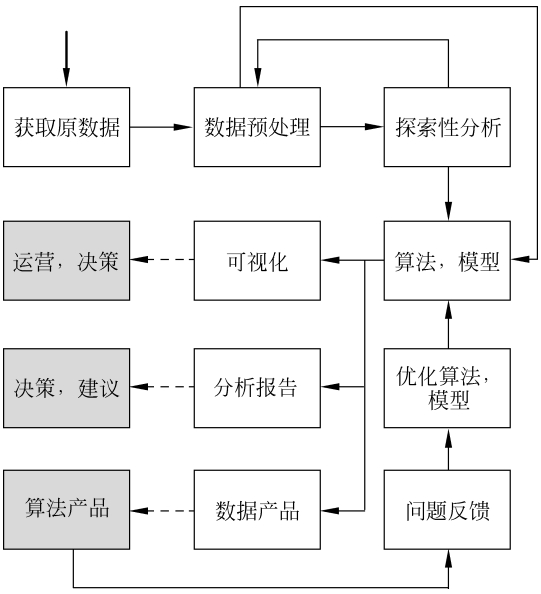

图 3.1　数据分析流程图

图 3.1 是从事数据分析或相关工作人员经常遇到的工作流程,比如数据分析师需要对公司的业务数据做数据分析,并将结论反馈给高层,以便给出数据支撑。算法产品可以作为公司的核心产品,比如头条新闻的个性化推荐算法,淘宝、天猫的商品推荐算法以及用户画像等,这些都是公司的宝贵资源。总之,数据是最客观、最真实的,若是其主观分析判断结果与数据结果相左,要么是主观判断是错的,要么就是分析数据的方法是错的。

3.2　基本统计量

基本统计量是简单的,同时也是重要的,在生活中每时每刻都在用一些基本的统计量,比如:杭州 2018 年人均可支配收入为 54 348 元(数据源于杭州市统计局);今天昼夜

温差为 8°。本节将介绍一些基本统计量，并通过 Python 语言实现其算法或定义相关的函数。

3.2.1　平均数

平均数是测量集中趋势最常见的统计量。平均数适用于数值数据，不适用于分类数据。平均数在实际工作中默认为算术平均数的情况较多，但是两者是不同的，算术平均数是平均数中的一种。常见的平均数有算术平均数(描述统计最常见)、几何平均数(金融领域较多)、调和平均数以及平方平均数。除此之外，还有一些不常见的：几何-调和平均数、算术-几何平均数和移动平均数。这里简单介绍四种常见的平均数，读者若感兴趣可检索相关的文献或资料，这里不再额外赘述。

设有两个随机变量 X 和 Y，其总体期望分别为 $E(X)=\mu_1$，$E(Y)=\mu_2$。现有对应两组样本 $x=\{x_1,x_2,\cdots,x_n\}$ 和 $y=\{y_1,y_2,\cdots,y_n\}$。

1. 算术平均数

$$\bar{x}=\frac{x_1+x_2+\cdots+x_n}{n} \tag{3.1}$$

称为算术平均数(mean，或称平均值)。对于变量 X，其期望 $E(X)=\mu_1$，对于实数 k，则有：

$$E(kX)=kE(X) \tag{3.2}$$

若两变量 X，Y 相互独立，则有 $E(XY)=E(X)E(Y)$，说明变量 X 和 Y 不相关(见相关性)。

算术平均数的代码实现，可以通过以下两种方法：

```
#样本
In [63]: sample_list = [50, 60, 75, 85, 100, 30, 48]
#样本量
In [64]: sample_num = len(sample_list)
#算术平均数, 方法 1
In [65]: mean_val = sum(sample_list) / sample_num
In [66]: mean_val
Out[66]: 64.0
#使用 NumPy 模块计算,方法 2
In [67]: import numpy as np
In [68]: np.mean(sample_list)
Out[68]: 64.0
```

2. 几何平均

几何平均数即 n 个数据相乘后开 n 次方。

$$\bar{x}=\sqrt[n]{\prod_{i=1}^{n}x_i} \tag{3.3}$$

其代码实现的方法如下：

```
#方法 1
In [69]: np.power(np.prod(sample_list), 1 / sample_num)
Out[69]: 59.86013681671229
#方法 2
In [70]: from scipy.stats import gmean                    #几何平均命令
In [71]: gmean(sample_list)
Out[71]: 59.86013681671234
#方法 3: 自定义相乘
In [74]: def prod_f(x, y):
    ...:     return x * y
In [75]: from functools import reduce
In [76]: np.power(reduce(prod_f, sample_list), 1 / sample_num)
Out[76]: 59.86013681671229
```

建议初学 Python 的读者编写代码时尽量少调用模块，通过 Python 的基本数据结构进行学习，加强练习。

3. 调和平均数

调和平均数，即 n 个数据的倒数再计算其算术平均数，再计算倒数。

$$\bar{x}=\frac{1}{\frac{\sum 1/x_i}{n}}=\frac{n}{\sum 1/x_i} \tag{3.4}$$

下面给出调和平均数的实现代码：

```
#方法 1 SciPy 模块
In [77]: from scipy.stats import hmean
In [78]: hmean(sample_list)
Out[78]: 55.585831062670295
#方法 2
In [79]: rec_sample_list = [1 / i for i in sample_list]              #元素取倒数
In [80]: rec_mean = sum(rec_sample_list) / len(rec_sample_list)      #算术平均数
In [85]: 1 / rec_mean                                                #调和平均数
Out[85]: 55.58583106267029
```

4. 平方平均数

平方平均数也就是常说的二范数，其数学表达式如下所示：

$$\bar{x}=\sqrt{\frac{x_1^2+x_2^2+\cdots+x_n^2}{n}} \tag{3.5}$$

这里通过 Python 实现该算法，并尽量不采用其他模块来实现。

```
def avg_sqr(data_list):
    '''
    计算给定数据列表的平方平均数，其数学公式如下所示：
    $\bar{x} = \sqrt{\frac{x_{1}^{2} + x_{2}^{2} + \cdots + x_{n}^{2}}{n}}$
    :param data_list: 列表，元素为数值型数据
    :return: 平方平均数
    '''
    #数据量
    data_num = len(data_list)
    #元素平方并求和,返回列表
    data_square_sum = sum([i * i for i in data_list])
    return pow((data_square_sum / data_num), 1 / 2)

if __name__ == '__main__':
    sample_list = [50, 60, 75, 85, 100, 30, 48]
    print(avg_sqr(sample_list))
```

以上代码的实现没有借助第三方模块，这利于较好地学习 Python 语言。其中 pow(x,n)相当于 x^n，但是弱于 np.power(x,n)函数，np.power(x,n)中的 x 可以是一个数值，也可以是一个列表(list)或元组(tuple)。

3.2.2　最值

最值包括极大值和极小值。关于最值的 Python 实现是非常简单的。

```
#样本
In [63]: sample_list = [50, 60, 75, 85, 100, 30, 48]
#方法 1
#最大值
In [95]: max(sample_list)
Out[95]: 100
#最小值
In [96]: min(sample_list)
Out[96]: 30
#方法 2
In [97]: min_v, *other, max_v = sorted(sample_list)
In [98]: min_v, max_v
Out[98]: (30, 100)
In [100]: other
Out[100]: [48, 50, 60, 75, 85]
```

方法 1 通过 Python 内置函数分别计算最大值和最小值；方法 2 通过命令 sorted 对列表进行升序(亦可通过参数 reverse＝True 降序)排列，再对列表进行切片，min_v、max_v 分别为升序后列表的第一个值和最后一个值，其他的值以列表的形式返回给 other，这是一个比较常用的方法。

3.2.3 中位数

中位数(median)是统计学中的专有名词,即将样本数值集合划分为数量相等或相差1的上下两部分,中位数可能是样本中的值,也可能不是。

定义 3.3 中位数

一组样本量为 n 的样本$(x_1,x_2,\cdots,x_n)$,其排序(升序、降序)后的样本$(x_1',x_2',\cdots,x_n')$。

$$M(x)=\begin{cases} x'_{\frac{n+1}{2}} & n \text{ 为奇数(odd)} \\ \dfrac{1}{2}\left(x'_{\frac{n}{2}}+x'_{\frac{n}{2}+1}\right) & n \text{ 为偶数(even)} \end{cases} \tag{3.6}$$

其代码实现如下:

```
def median_func(data_list):
    '''
    中位数:
    $M(x) = \left\{
        \begin{array}{ll}
        x^{\prime}_{\frac{n+1}{2}} &, n \mbox{为奇数} \\
        \frac{1}{2}(x^{\prime}_{\frac{n}{2}} + x^{\prime}_{\frac{n}{2} + 1}) &, n \mbox{为偶数},
        \end{array} $
    :param data_list: 列表或元组
    :return: 返回中位数
    '''
    data_sorted = sorted(data_list, reverse = False)
    data_num = len(data_sorted)
    if data_num % 2:            #奇数
        return data_sorted[(data_num + 1) / 2]
    return .5 * (data_sorted[int(data_num / 2 - 1)] + data_sorted[int(data_num / 2)])

if __name__ == '__main__':
    #NumPy模块
    import numpy as np
    data_list = [1,2,3,4]
    print(median_func(data_list))
    print(np.median(data_list))
```

自定义的中位数代码与 NumPy 模块中的命令 median 输出结果一致。需要注意的是,Python 进行切片时,第 1 个元素的 index 从 0 开始,另外可以试着验证其性能上的差异。

3.2.4 众数

众数(mode)是指一组样本中出现次数最多的值,是统计学里面比较重要的一个统计量。众数在一定程度上能反映出集中趋势。SciPy 模块也有其对应的函数命令 mode。

```
In [110]: from scipy.stats import mode
In [111]: sample_list = [50, 60, 75, 85, 100, 30, 48]
#各元素出现次数一致时，返回最少的结果
In [112]: mode(sample_list)
Out[112]: ModeResult(mode = array([30]), count = array([1]))
#各元素出现次数不一致时，返回最多的结果
In [113]: mode([1, 2, 3, 4, 4])
Out[113]: ModeResult(mode = array([4]), count = array([2]))
#若元素出现次数最多的不止一个，返回最少的结果
In [63]: mode([1, 1, 5, 5, 3, 4])
Out[63]: ModeResult(mode = array([1]), count = array([2]))
```

3.2.5　极差

极差(range，又称全距)是衡量指定变量间差异变化范围，通常极差越大，样本变化范围越大。

$$\mathrm{ptp}(x)=\max_{1\leqslant i\leqslant n}\{x_i\}-\min_{1\leqslant i\leqslant n}\{x_i\} \tag{3.7}$$

通过命令 numpy.ptp(data)可以实现，当然也可能通过 max(data) − min(data)实现。

```
In [65]: sample_list = [1, 1, 5, 5, 3, 4]
In [72]: import numpy as np
In [73]: np.ptp(sample_list)                    #极差
Out[73]: 4
In [74]: max(sample_list) - min(sample_list)    #极差
Out[74]: 4
```

3.2.6　方差

方差(variance)用于衡量样本的离散程度。方差在概率论和统计学中普通存在，通常用符号 σ^2 表示，σ 称为标准差，其数学表达式为：

$$\sigma^2=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2 \tag{3.8}$$

在统计学中，多采用以下方法计算方差(无偏性)。

$$\sigma^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2 \tag{3.9}$$

NumPy 和 SciPy 模块都有其函数命令 var，这里不通过第三方模块来实现其代码。

```
def var_func(data_list, method = None):
    '''
    方差的数学表达式:
    \sigma^{2} = \frac{1}{n}\sum_{i = 1}^{n}\left(x_{i} - \overline{x}\right)^{2}
    :param data_list: 列表或元组，元素为数值型数据
```

```
    :parm method: None, 分母:n; Not None, 分母: n - 1
    :return: 方差
    '''
    #样本量
    data_num = len(data_list)
    if method is not None:
        data_num = data_num - 1
    #算术平均数
    data_mean = sum(data_list) / data_num
    return sum([(i - data_mean) ** 2 for i in data_list]) / data_num
```

上面代码中 ** 表示幂运算(又称指数运算),x ** 2=x * x=x^2,等同于 pow(x,2)。上面代码对式(3.8)和式(3.9)进行了代码实现。

3.2.7 变异系数

变异系数(又称离散系数),即样本标准差与算术平均数的比值,可以将含有量纲的标准差进行无量纲处理。

$$\mathrm{CV}=\frac{\sigma}{\bar{x}} \tag{3.10}$$

上面给出了算术平均数和方差的公式,其代码实现也非常简单,可以通过 NumPy 模块直接求解:numpy.std(data)/numpy.mean(data)。

3.2.8 协方差

定义 3.4 协方差

假设两个变量 X,Y 的期望分别为 $E(X)=\mu_1$,$E(Y)=\mu_2$,定义 X,Y 的协方差 cov(X,Y)为:

$$\mathrm{cov}(X,Y)=E((X-\mu_1)(Y-\mu_2))=E(XY)-\mu_1\mu_2 \tag{3.11}$$

对于变量 X,Y,其协方差矩阵为:

$$\boldsymbol{\Sigma}=\begin{bmatrix}\mathrm{cov}(X,X) & \mathrm{cov}(X,Y)\\ \mathrm{cov}(Y,X) & \mathrm{cov}(Y,Y)\end{bmatrix} \tag{3.12}$$

其中 cov(X,Y)=cov(Y,X),式(3.12)中的$\boldsymbol{\Sigma}$是一个实对称矩阵。

关于协方差的计算,其实现代码如下所示。

```
def cov_func(x_arr, y_arr):
    '''
    协方差:
    $cov(X, Y) = E((X-\mu_{1})(Y-\mu_{2})) = E(XY) - \mu_{1}\mu_{2}$
    :param
        x_arr: 数据(向量)x
        y_arr: 数据(向量)y
    :return:
```

```
    '''
    x_num = len(x_arr)
    x_mean = np.mean(x_arr)
    y_mean = np.mean(y_arr)
    return np.mean((x_arr - x_mean) * (y_arr - y_mean)) * (x_num)/(x_num - 1)

if __name__ == "__main__":
    import numpy as np
    x_arr = np.arange(1, 10, .05)
    y_arr = 3 * x_arr
    X = np.stack((x_arr, y_arr), axis = 0)
    print("协方差矩阵\n", np.cov(x_arr, y_arr))
    print(cov_func(x_arr, y_arr), cov_func(x_arr, x_arr))
```

输出结果：

```
python cov.py
协方差矩阵
    [[ 6.7875 20.3625]
    [20.3625 61.0875]]
20.36250000000004 6.787500000000012
```

通过输出结果可以看出，命令 numpy.cov 输出的是协方差矩阵，显然协方差是一个实对称矩阵(半正定矩阵)。这里需要注意的是，NumPy 模块中命令 cov 在默认条件下计算协方差与定义略有差异，最后一步计算期望时分母为 $n-1$，而不是 n。

协方差用于衡量两个变量(样本)的总体误差。当两个变量相同时，协方差就是方差。协方差可以较好地反映出两个变量的变化趋势，如图 3.2 所示。

- 若 $\mathrm{cov}(X,Y)>0$，X，Y 变化趋势相同；
- 若 $\mathrm{cov}(X,Y)<0$，X，Y 变化趋势相反；
- 若 $\mathrm{cov}(X,Y)=0$，称 X，Y 不相关。

其中，ε 是扰动项。

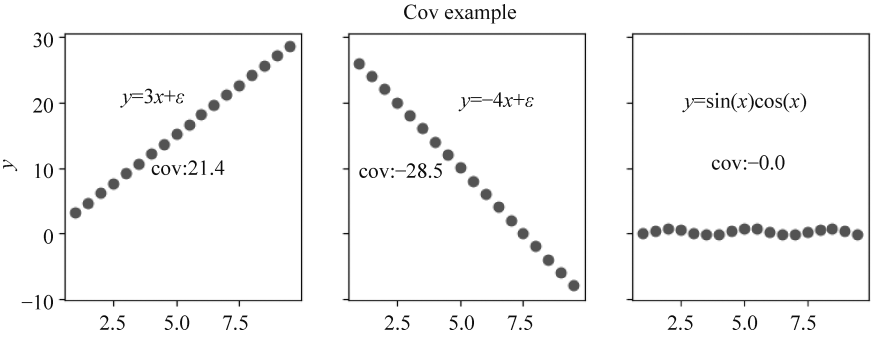

图 3.2　协方差示例图

存在 a,b,c,d 四个常数,协方差有以下较好的性质:

$$\begin{aligned}&\mathrm{cov}(X,Y)=\mathrm{cov}(Y,X)\\&\mathrm{cov}(aX+b,cY+d)=ac\,\mathrm{cov}(Y,X)\\&\mathrm{cov}(X_1+X_2,Y)=\mathrm{cov}(X_1,Y)+\mathrm{cov}(X_2,Y)\\&\mathrm{cov}(X,Y)=E(XY)-E(X)E(Y)\end{aligned}$$

3.2.9 相关系数

相关系数是衡量数值型数据之间的线性相关性,这里主要给出最为常用的皮尔逊(Pearson)相关系数,其定义如下所示。

$$\rho_{X,Y}=\frac{\mathrm{cov}(X,Y)}{\sigma_1\sigma_2}=\frac{E((X-\mu_1)(Y-\mu_2))}{\sigma_1\sigma_2} \tag{3.13}$$

其中,$\mu_1=E(X)$,$\mu_2=E(Y)$,$\sigma_1^2=E(X^2)-E^2(X)$,$\sigma_2^2=E(Y^2)-E^2(Y)$。显然,$\rho\in[-1,1]$,其 Python 代码实现较为简单,使用 NumPy 模块中的 np.corrcoef。变量 X 和 Y 之间的相关性有以下性质:

- 当 $\rho_{X,Y}>0$ 时,变量 X,Y 之间线性正相关;
- 当 $\rho_{X,Y}<0$ 时,变量 X,Y 之间线性负相关;
- 当 $\rho_{X,Y}=0$ 时,变量 X,Y 之间无线性相关;
- 当 $|\rho_{X,Y}|=1$ 时,变量 X,Y 之间完全线性相关。

通常变量 X,Y 之间的相关程度分为 3 级:

Ⅰ. $|\rho_{X,Y}|<0.4$ 为低度线性相关;

Ⅱ. $0.4\leqslant|\rho_{X,Y}|<0.7$ 为显著线性相关;

Ⅲ. $0.7\leqslant|\rho_{X,Y}|<1$ 为高度线性相关。

3.3 数据转换

在解决实际问题之前,经常需要对数据进行预处理。数据标准化作为一种简单的计算方式,可以将有量纲的数据变换成无量纲的数据。最经典的就是数据归一化处理,即将数据映射到[0,1]区间内。常用的数据预处理方法有中心化、Box-Cox 转换、min-max 标准化(离差标准化)、log 函数转换和 z-score 标准化等。在处理实际问题时需要根据具体情况分析,再确定使用哪一种标准化方法。

3.3.1 中心化

$$x'=x-\bar{x} \tag{3.14}$$

式(3.14)是一种常见的数据预处理方法。

代码实现如下所示:

```
1  #方法 1
2  In [162]: sample_list
3  Out[162]: [1, 2, 3, 1, 2, 4, 10]
```

```
In [164]: mean_val = sum(sample_list) / len(sample_list)
In [166]: [i - mean_val for i in sample_list]
Out[166]:
[-2.2857142857142856,
     -1.2857142857142856,
     -0.2857142857142856,
     -2.2857142857142856,
     -1.2857142857142856,
     0.7142857142857144,
     6.714285714285714]
#方法 2
In [167]: mean_v = np.mean(sample_list)
In [168]: np.array(sample_list) - mean_v
Out[168]:
array([-2.28571429, -1.28571429, -0.28571429, -2.28571429, -1.28571429,
         0.71428571, 6.71428571])
```

NumPy 模块中的数组支持向量化计算，其计算性能优于方法 1。

3.3.2　min-max 标准化

$$x^{*}=\frac{x-x_{\min}}{x_{\max}-x_{\min}} \tag{3.15}$$

其中，$x_{\max}$ 为数据中的最大值，$x_{\min}$ 为数据中的最小值。该方法存在一定问题，当数据更新后其最值可能发生变化，则需要更新。该方法对 x 数据没有严格的要求，使用最为广泛。其代码如下所示。

```
def min_max(data_list):
    min_v, *_, max_v = sorted(data_list)
    dff_v = max_v - min_v
    return [(value - min_v) / dff_v for value in data_list]
```

读者也可以通过相关包直接调用 sklearn.preprocessing.MinMaxScaler。

3.3.3　Box-Cox 转换

在数据预处理期间经常会遇到数据分布不满足正态分布的情况，利用 Box-Cox 转换可以有效地改善数据的正态性。Box-Cox 的转换形式如下：

$$x^{\lambda}=\begin{cases}\dfrac{x^{\lambda}-1}{\lambda}, & \lambda\neq 0\\ \ln(x), & \lambda=0\end{cases} \tag{3.16}$$

其中，x 是源数据，并且数据 $x_i>0(i=1,2,3,\cdots,n)$，x^{λ} 是处理后的正态分布数据集。关于 λ 的取值，只能通过尝试的方法来实现，$\lambda=0.5,1,1.5,2,\cdots$，一旦确定了 λ，就可以得到满足正态分布的数据。另外，若存在小于零的数据，可以对所有原始数据都加上一个常

数 a 使得所有的数据均为正值,再进行 Box-Cox 转换。

```
import numpy as np
def box_cox(data_list, args = 1):
    '''
    box-cox()方法, 用于将数据转换为正态数据
    :param data_list: 样本数据集
    :parm args: 0: log(x), 1: (x^args - 1)/args
    :return: box-cox 处理
    '''
    data_arr = np.array(data_list)
    if args == 0:
        return np.log(data_arr)
    return (np.power(data_arr, args) - 1) / args
```

3.3.4 log 函数转换

log 函数转换是针对特别的数据进行转换的一种方式,通常使用以 10 为底的 log 函数转换的方法实现归一化。

$$x^* = \frac{\log_{10}(x)}{\log_{10}(x_{\max})} \tag{3.17}$$

其中,所有的数据 x 需要满足 $x \geqslant 1$,显然 $x^* \in [0,1]$。代码如下:

```
import numpy as np
def log_norm(data_arr):
    '''
    log 函数转换\frac{\mbox{log}_{10}(x)}{\mbox{log}_{10}(x_{\max})}
    :param data_arr: array, 原数据集
    :return: array, log 函数转换后的数据
    '''
    max_v = max(data_arr)
    return np.log10(data_arr) / np.log10(max_v)
```

3.3.5 z-score 标准化

z-score 标准化又称为标准差标准化,指将数据处理成符合标准正态分布的数据。

$$x^* = \frac{x - \mu}{\sigma} \tag{3.18}$$

其中,μ 为数据集的均值(算术平均数),σ 是数据集的标准差。其代码如下:

```
import numpy as np
def z_norm(data_list):
    '''
    z-score 标准化\frac{x - \mu}{\sigma}
```

```
    :param data_list:list, 原数据集
    :return: 标准化后数据
    '''
    data_len = len(data_list)
    if data_len == 0:
        raise "数据为空"
    mean_v = np.mean(data_list)
    var_v = np.var(data_list)
    if var_v == 0:
        raise "标准差为 0"
    return (np.array(data_list) - mean_v) / var_v
```

这里通过 NumPy 模块实现，命令 numpy.array 将列表转换为数组，从而避免 for 循环，由于 NumPy 是基于 C++开发的，其执行效率很高。

除了以上几种数据转换外，还有数据平整、小数缩放、差值和比率等，读者可查阅相关资料进行了解和学习，由于实现比较简单，这里不再赘述。

3.4　常见距离

这里主要介绍几种常见的距离公式，主要有欧氏距离、曼哈顿距离、余弦值相似度等。不妨假设两个向量 $\boldsymbol{x}=(x_1,x_2,\cdots,x_m)\in\mathbb{R}^m$，$\boldsymbol{y}=(y_1,y_2,\cdots,y_m)\in\mathbb{R}^m$，下面介绍向量 $\boldsymbol{x},\boldsymbol{y}$ 距离的定义。

3.4.1　闵氏距离

$$d(\boldsymbol{x},\boldsymbol{y})=\left(\sum_{i=1}^{m}|x_i-y_i|^p\right)^{\frac{1}{p}} \tag{3.19}$$

称为 $\boldsymbol{x}-\boldsymbol{y}$ 的 p-范数。

当 p 取不同的自然数时，可以得到不同的距离公式。NumPy 模块中定义了广泛的距离公式，在 ipython 中可以通过命令 np.linalg.norm?? 查看。

- 当 $p=1$ 时，$d(\boldsymbol{x},\boldsymbol{y})=\sum_{i=1}^{m}|x_i-y_i|$，为曼哈顿距离；
- 当 $p=2$ 时，$d(\boldsymbol{x},\boldsymbol{y})=\sqrt{\sum_{i=1}^{m}|x_i-y_i|^2}$，为欧氏距离；
- 当 $p=\infty$ 时，$d(\boldsymbol{x},\boldsymbol{y})=\max_{1\leqslant i\leqslant m}|x_i-y_i|$，为切比雪夫距离。

日常工作中存在对闵氏距离进行修改的情况。

$$d(\boldsymbol{x},\boldsymbol{y})=\left(\sum_{i=1}^{m}w_i|x_i-y_i|^p\right)^{\frac{1}{p}} \tag{3.20}$$

其中 w_i 是关于 x_i 和 y_i 的加权系数或函数。以上提到的距离相应的 Python 代码如下：

```
import numpy as np
def dis_func(x_list, y_list, args = 1):
```

```
    '''
    闵氏距离, $ d(\vec{x},\vec{y}) = ({\sum_{i = 1}^{m}|x_{i} - y_{i}|^{p}})^{\frac{1}{p}} $
    args = 1, 曼哈顿距离
    args = 2, 欧氏距离
    args = p, p - 范数
    args = inf, 切比雪夫距离
    :param x_list: 数据 x
    :param y_list: 数据 y
    :return: 距离
    '''
    #list2array, 支持向量计算
    x_arr = np.array(x_list)
    y_arr = np.array(y_list)
    if x_arr.shape != y_arr.shape:
        raise "The shape of two vector(matrix) must be equal."
    if args < 1:
        raise "The value of args less than 1"
    if 1 <= args < np.inf:
        return np.power(np.sum(np.power((x_arr - y_arr), args)), 1 / args)
    return np.max(np.abs(x_arr - y_arr))
```

3.4.2 余弦值相似度

余弦相似度(cosine similarity)用向量空间中两个向量夹角的余弦值作为衡量两个个体差异的大小。余弦值越接近 1,就表明夹角接近 0,也就是说两个向量越相似,即余弦相似性。

$$\cos\theta = \frac{\sum_{i=1}^{m}(x_i y_i)}{\sqrt{\sum_{i=1}^{m} x_i^2}\sqrt{\sum_{i=1}^{m} y_i^2}} = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \ \|\boldsymbol{y}\|} \tag{3.21}$$

通常,$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的元素是分类数据数值化的数值,因此其样本矩阵是一个稀疏矩阵,余弦相似度在文本数据处理中有广泛的应用。Python 实现代码如下:

```
import numpy as np
def cos_func(x_list, y_list):
    '''
    余弦相似度函数\frac{\vec{x} \cdot \vec{y}}{||\vec{x}||||\vec{y}||}
    :param x_list: list or tuple
    :param y_list: list or tuple
    :return: x_list 与 y_list 的余弦相似度
    '''
    x_arr = np.array(x_list)
    y_arr = np.array(y_list)
    if x_arr.shape != y_arr.shape:
        raise "{} is not equal to {}".format(x_arr.shape, y_arr.shape)
    #内积
```

```
    x_y_dot = x_arr.dot(y_arr)
    x_dis = dis_func(x_arr, x_arr, args = 2)
    y_dis = dis_func(y_arr, y_arr, args = 2)
    if (x_dis == 0) or (y_dis == 0):                    #分母不为零判断
        raise "the denominator is zero"
    return x_y_dot / (x_dis * y_dis)
```

计算余弦值相似度时调用了闵式距离中的 dis_func()函数。函数的调用可以实现代码的复用及利于维护，在往后的学习中会经常遇到。另外还使用了命令 numpy.dot，它支持向量内积。

3.5　多维数据

3.5.1　矩阵

设有 n 个未知数的 m 个方程的线性代数方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases} \tag{3.22}$$

其中，a_{ij} 是第 i 个方程的第 j 个未知系数的系数，b_i 是第 i 个方程的常数项（$i=1,2,\cdots,m$；$j=1,2,\cdots,n$）。当 $b_i=0$ 全成立时，为齐次线性代数方程组，否则成为非齐次线性代数方程组。

数学上，一个 $m\times n$ 的矩阵是一个由 m 行(row)n 列(column)元素排列成的矩形阵列。其元素不局限于数值，也可以是符号或数学式。

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{3.23}$$

当 $m=n$ 时，称为方阵。式(3.22)可以写成 $\boldsymbol{Ax}=\boldsymbol{b}$ 的形式，其中 $\boldsymbol{A}\in\mathbb{R}^{m\times n}$。

在 NumPy 模块中，命令 mat(matrices)仅支持二维，命令 array 支持多维。array 包含 mat，因此后者具有前者所有的特性，建议在进行矩阵(数组)计算时，尽量不要混合使用，不然易导致错误。下面简要看下命令。

```
In [1]: from numpy import mat, array
#定义数组
In [2]: x_arr = array([[1,2,3], [2,5,7]])
In [3]: y_arr = array([[1,1,1], [2,2,2], [3,3,3]])
#数组转换为矩阵
In [4]: x_mat = mat(x_arr)
In [5]: y_mat = mat(y_arr)
#数据类型
```

```
In [7]: type(x_arr)
Out[7]: numpy.ndarray
In [8]: type(x_mat)
Out[8]: numpy.matrixlib.defmatrix.matrix
#乘法
#数组对应元素相乘
In [9]: x_arr * x_arr
Out[9]:
array([[ 1, 4, 9],
       [ 4, 25, 49]])
#矩阵乘法
In [11]: x_mat * x_mat.T
Out[11]:
matrix([[14, 33],
        [33, 78]])
```

显然,array 和 mat 之间进行转换非常简单。读者可能会提出到底用哪种类型计算好的问题,笔者认为这与解决问题有关。若问题是二维的,并且涉及矩阵的运算法则,则采用 mat 类型。在往后的学习中经常会使用 Python 的标准库(比如 NumPy 和 SciPy),原因主要有两点:

- 本书重点在于通过数据科学的算法解决实际问题;
- Python 的标准库在数值计算中,其执行效率通常高于自编的代码,毕竟很多标准库是基于 C++或 FORTRAN 开发的;
- 充分利用现有的“轮子”,也是一种明智的方法。

下面通过在 Jupyter Notebook(或 IPython3)交互环境下举例说明。首先给出求解行列式的 det 自定义函数。

```
import numpy as np
def det(data):
    '''
    求解行列式函数, 通过代数余子式的方法实现
    :param data: list or tuple
    :return: 计算行列式
    '''
    if len(data) <= 0:
        return None
    m, n = np.array(data).shape
    if m != n:
        raise "m must be equal n!"
    elif len(data) == 1:
        return data[0][0]
    else:
        result = 0
        for i in range(len(data)):
            n = [[row[a] for a in range(len(data)) if a != i] for row in data[1:]]
            result += data[0][i] * det(n) * (-1) ** (i % 2)
        return result
```

在交互环境下的实验结果如下：

```
In [43]: data_list = [[1, 2, 3],[2, 4, 8],[3, 4, 5]]
In [44]: data = np.array(data_list)
#调用 NumPy 库
In [45]: %timeit np.linalg.det(data)
10.9 μs ± 200 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
#自定义函数
In [48]: %timeit det(data)
68.7 μs ± 2.38 μs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

在求解给定行列式时，通过实验不难发现自定义函数花费的时间是标准库的 6.30 倍。关于矩阵的一系列运算的函数大多数都封存在 numpy.linalg 和 scipy.linalg 中，但是 scipy.linalg 包含 numpy.linalg 中的所有函数，另外还有一些不在 numpy.linalg 中的高级函数，读者可以通过官方文档学习。

3.5.2　特征值和特征向量

对于矩阵 $\boldsymbol{A}$，若存在标量 λ 和向量 x，使得

$$\boldsymbol{A}\boldsymbol{x}=\lambda\boldsymbol{x} \tag{3.24}$$

式中，λ 为特征值，$\boldsymbol{x}$ 为特征向量。

特征值和特征向量是非常重要的概念，在控制系统、图片压缩、因子分析、主成分分析、振动分析和应力张量等方面都有广泛的应用，后面的章节中还会多次使用。

通过 NumPy 模块求解矩阵的特征值和特征向量，其代码如下所示。

```
Out[49]: A
    matrix([[14, 33],
            [33, 78]])
In [50]: eig, eig_vec = np.linalg.eig(A)
#特征值
In [51]: eig
Out[51]: array([11.43039223, -0.31324931, -1.11714292])
#特征向量
In [52]: eig_vec
Out[52]:
matrix([[-0.31402229, -0.53730604, 0.05022683],
        [-0.73351464, 0.79114333, -0.84699731],
        [-0.60278211, -0.29222329, 0.52921908]])
```

不难发现输出的特征值数据类型是数值，特征向量是矩阵格式，并且特征值和特征向量是对应关系，比如特征值为 11.43039223，其对应的特征向量为[－0.31402229，－0.73351464，－0.60278211]。读者可以进行实践和验证。

3.5.3 多重共线性

多重共线性是指线性回归模型中的特征(自变量)之间存在某种相关或者高度相关的关系,这种现象会致使模型估计失真或难以估计准确。通常这种相关是指线性相关性,即存在某个特征可以被其他特征(可能多个)组成的线性组合来解释。统计学中,若构建多重共线性回归模型时,即将所有特征(自变量)进行考虑回归(拟合)时,将会导致方程估计的偏回归系数明显与常识不相符,最终导致模型不具有合理性和科学性。

1. 诊断方式

关于多重共线性问题,本书介绍两种常见的诊断方法。

- 计算特征(自变量)两两之间的相关系数(见式(3.13)),通常相关性系数 $\rho>0.7$ 时可考虑特征间存在多重共线性;
- 共线性诊断统计量,即容忍度(tolerance)和 VIF(方差膨胀因子)。通常 tolerance <0.2 或 VIF>5 时可考虑特征间存在多重共线性。

其中,容忍度是 1 减去指定自变量的值,为因变量,其他自变量为自变量的线性回归的决定系数 R^2 的剩余值($1-R^2$)。显然,容忍度越小,共线性越严重。方差膨胀是容忍度的倒数。

注:决定系数 R^2 用于检验回归方程对给定数据的拟合程度,其值在[0,1]区间内,R^2 越大,说明拟合效果越好。

2. 解决方法

待诊断存在多重共线后,就需要考虑如何解决多重共线性问题而构建合理的回归函数。笔者几年前接触 SPSS 时发现共提供了 5 种解决方案。由于篇幅所限,这里只扼要说明这 5 种方案。

- 进入法(enter),即人为选定特征(需要一定的经验和定性分析)强行导入模型进行拟合;
- 移除法(remove),即先通过进入法构建拟合函数,然后人为移除选定特征(自变量)再进行回归(拟合),也可与其他方法相结合使用;
- 前进法(forward section),即逐个导入特征(自变量)来构建模型的一种方式;
- 后退法(backward elimination),与前进法相反,先将全部特征值添加到模型中,再逐一剔除特征的方式构建模型;
- 逐步回归法(stepwise),建立在前进法和后退法基础上,其过程反复进行,直到没有不显著的自变量引入回归方程为止。

逐步回归法是一个经典的排除多重共线性的方法。除此之外还有差分法,其主要用于处理时间序列数据。另外还有一个非常有名的方法:岭回归法(ridge regression)。

定义 3.5 岭回归

现有一个 m 行 n 列的线性方程组 $\boldsymbol{Ax}=\boldsymbol{y}$(如式(3.22)),构建 m 个方程 $\hat{y}=\theta_0+\theta_1x_1+\theta_2x_2+\cdots+\theta_nx_n$($i=1,2,3,\cdots,m$),也就是说用 $\hat{y}$ 来近似代替 y_i,则岭回归的损失函

数为：

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i-y_i)^2+\lambda\sum_{j=1}^{n}\theta_j^2 \tag{3.25}$$

其中，λ 称为正则化参数。若 λ 选取过大，会将参数 θ 最小化导致模型欠拟合；若 λ 选取过小会致使过拟合问题。问题的难度在于 λ 的合理选取。

除此之外，还有一个非常有名的 Lasso 回归，两者的区别在于惩罚项的范数不一样，读者若感兴趣可通过文献[28]进行学习，其算法均可以通过模块 sklearn 实现，这里不再赘述。

3.6　学生基本信息实例

1. 探索性分析

描述性分析是作为数据分析工作者必须具备的技能，可将其归属为探索性分析，读者可以通过描述性分析的结果对面对的问题有一定的认识，为进一步分析起到一定的指导性作用。下面给出一组数据，含有 40 个样本，5 个维度（姓名、年龄、性别、身高和体重），如表 3.2 所示。①

表 3.2　某学校某班学生的基本信息

序号	姓　　名	年龄	性别	体重	身高	序号	姓　　名	年龄	性别	体重	身高
0	KATIE	12	F	59	95	20	FREDERICK	14	M	63	93
1	LOUISE	12	F	61	123	21	ALFRED	14	M	64	99
2	JANE	12	F	55	74	22	HENRY	14	M	65	119
3	JACLYN	12	F	66	145	23	LEWIS	14	M	64	92
4	LILLIE	12	F	52	64	24	EDWARD	14	M	68	112
5	TIM	12	M	60	84	25	CHRIS	14	M	64	99
6	JAMES	12	M	61	128	26	JEFFREY	14	M	69	113
7	ROBERT	12	M	51	79	27	MARY	15	F	62	92
8	BARBARA	13	F	60	112	28	AMY	15	F	64	112
9	ALICE	13	F	61	107	29	ROBERT	15	M	67	128
10	SUSAN	13	F	56	67	30	WILLIAM	15	M	65	111
11	JOHN	13	M	65	98	31	CLAY	15	M	66	105
12	JOE	13	M	63	105	32	MARK	15	M	62	104
13	MICHAEL	13	M	58	95	33	DANNY	15	M	66	106
14	DAVID	13	M	59	79	34	MARTHA	16	F	65	112
15	JUDY	14	F	61	81	35	MARION	16	F	60	115
16	ELIZABETH	14	F	62	91	36	PHILLIP	16	M	68	128
17	LESLIE	14	F	65	142	37	LINDA	17	F	62	116
18	CAROL	14	F	63	84	38	KIRK	17	M	68	134
19	PATTY	14	F	62	85	39	LAWRENCE	17	M	70	172

① 数据源于 JMP 软件的内置数据集（Big Class 数据）。

数据中的姓名和性别的数据类型是分类数据,其他为数值数据。不妨对年龄、身高和体重进行描述性分析。

```
import numpy as np
import pandas as pd
#数据路径
path = "./data/Big Class.xls"
#读取 xls 数据
data_df = pd.read_excel(path)
data_df[['年龄', '身高', '体重']].describe()
```

其结果如表 3.3 所示。

表 3.3 某学校学生的基本信息

计量	年　龄	身　高	体　重
count	40.000000	40.000000	40.000000
mean	13.975000	105.000000	62.550000
std	1.476092	22.201871	4.242338
min	12.000000	64.000000	51.000000
25%	13.000000	91.750000	60.750000
50%	14.000000	105.000000	63.000000
75%	15.000000	115.250000	65.000000
max	17.000000	172.000000	70.000000

对于表 3.3,通过一行命令即可得到多个基本统计量:总数(count)、均值(mean)、标准差(std)、最小值(min)、25%(四分之一分位数)、50%(二分之一分位数,即中位数)、75%(四分之三分位数)和最大值(max)。数据量(即总数)均为 40,说明不存在缺失值;年龄最小值为 12,最大为 17,极差为 5,50%为 14,与均值(13.975)差异不大。同样,还可以得到身高和体重的相关数据。

以性别为维度分组进行描述性分析,如表 3.4 所示。

表 3.4 不同性别学生的基本信息

统计量	性别	年龄	身高	体重	统计量	性别	年龄	身高	体重
count	F	18.000000	18.000000	18.000000	count	M	22.000000	22.000000	22.000000
mean	F	13.777778	100.944444	60.888889	mean	M	14.136364	108.318182	63.909091
std	F	1.555089	23.435700	3.611890	std	M	1.424127	21.099281	4.308453
min	F	12.000000	64.000000	52.000000	min	M	12.000000	79.000000	51.000000
25%	F	12.250000	84.250000	60.000000	25%	M	13.000000	95.750000	62.250000
50%	F	14.000000	101.000000	61.500000	50%	M	14.000000	105.000000	64.500000
75%	F	14.750000	114.250000	62.750000	75%	M	15.000000	117.500000	66.750000
max	F	17.000000	145.000000	66.000000	max	M	17.000000	172.000000	70.000000

下面对3个特征(维度)作出箱形图(box plot)。由于Matplotlib模块在Mac OS系统下不能正常显示中文,笔者这里提供一种简单方法。在Jupyter Notebook环境下进行作图,箱形图如图3.3所示。

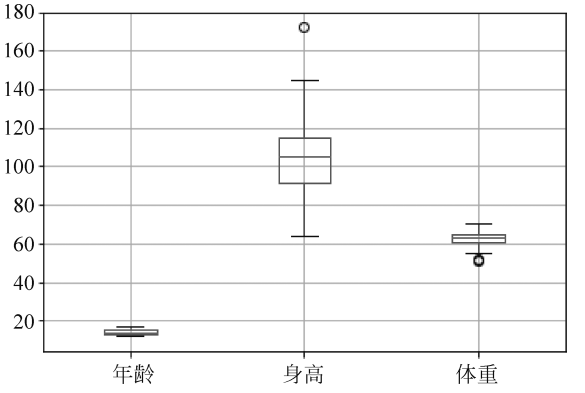

图3.3　箱形图

通过箱形图的结果,年龄集中趋势最为明显,体重集中度最差,身高次之。体重和身高存在异常值,下面只针对身高进行箱形图分析,如图3.4所示。

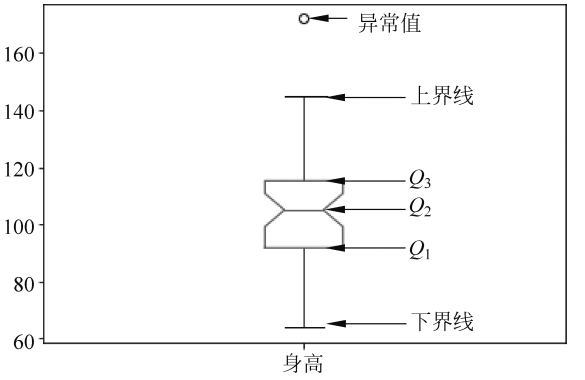

图3.4　身高箱形图

图3.3的代码如下所示。

```
import matplotlib.pyplot as plt
#内置显示图片
%matplotlib inline
#中文配置
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
#箱形图
data_df[['年龄','身高','体重']].boxplot()
#保存图片 png 在当前工作路径下
plt.savefig("boxplot.png")
```

四分位数在统计学中是一个较为重要的统计量,它可以用于识别异常值。对异常值进行检查常用 Turkey's test 方法。

定义 3.6 Turkey's test 方法

取一个常数 k(通常为 1.5 或 3),数据 x 的四分之一分位数为 Q_1,四分之三分位数为 Q_3,对于值 $x_i \in x$,有

$$x_i^{\text{异常}} = \begin{cases} x_i < Q_1 - k(Q_3 - Q_1) \\ x_i > Q_3 + k(Q_3 - Q_1) \end{cases} \tag{3.26}$$

其中,$k=1.5$ 称为中度异常值,$k=3$ 称为极度异常值。

下面是身高箱形图的作图代码,作为数据科学研究者,数据的可视化是非常重要的。

```
high_df = data_df[['身高']]
#标注内容及坐标
content_dict = {
    '异常值': [[1, 172], [1.1, 170]],
    '上界线': [[1.03, 144], [1.2, 144]],
    '下界线': [[1.03, 65], [1.2, 65]],
    '$Q_{2}$': [[1.03, 105], [1.2, 105]],
    '$Q_{1}$': [[1.03, 91.75], [1.2, 91.75]],
    '$Q_{3}$': [[1.03, 115.25], [1.2, 115.25]]}
#notch: 凹口,whis:1.5 倍四分位差
high_df.boxplot(notch = True, whis = 1.5)
plt.grid()
#添加标签内容
for key, value in content_dict.items():
    plt.annotate(r'{}'.format(key), xy = tuple(value[0]), xytext = tuple(value[1]),
                                    color = '#090909', arrowprops =
                                    dict(arrowstyle = '->',connectionstyle =
                                    'arc3, rad = 0.1', color = 'red'))
```

通过箱形图不难发现身高和体重两个指标存在异常值的情况,现在将其找出,如表 3.5 所示。

表 3.5 异常样本量

序号	姓 名	年龄	性别	体重	身高
4	LILLIE	12	F	52	64
7	ROBERT	12	M	51	79
39	LAWRENCE	17	M	70	172

现在分析异常值,通过表 3.5 的描述性分析结果不难发现,数据中年龄最小的为 12 岁,最大的为 17 岁,而 3 个异常数据中的年龄属于最小和最大情况,通过定性分析不难发现,通常身高和体重随年龄的增加而增加,待成年后趋于平稳。因此无法判断出这 3 个异常值是异常的。因此需要进一步对数据进行分析。

下面通过 pandas 和 Matplotlib 模块对特征性别、年龄、身高和体重进行条形图(分类

数据)和直方图(数值数据)绘图,首先需要对性别进行数值化处理。

```
#1: 男 2: 女
data_df['性别_数值'] = data_df['性别'].map(lambda x: 1 if x == 'M' else 2)
#直方图条形图
data_df[['性别_数值','年龄','体重','身高']].hist(figsize = (6,6))
```

其结果如图 3.5 所示。

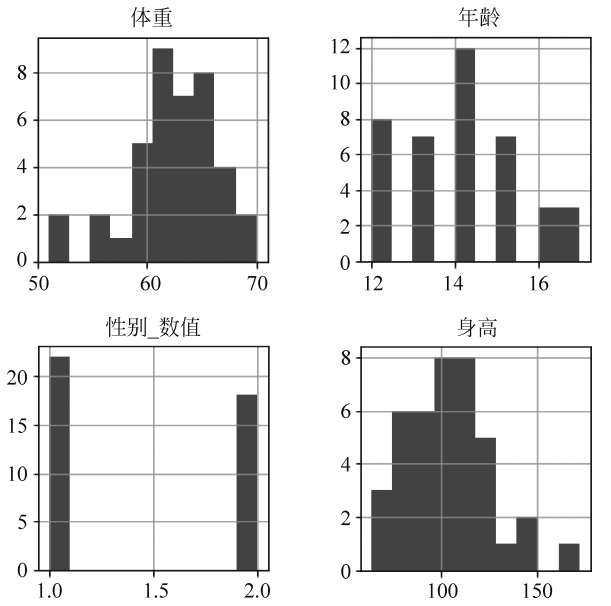

图 3.5　4 个特征的直方图(条形图)

下面计算年龄、身高和体重之间的相关性系数,结果如表 3.6 所示。

表 3.6　年龄,身高和体重之间的相关性系数

	年　龄	身　高	体　重
年龄	1.000000	0.463185	0.608260
身高	0.463185	1.000000	0.709167
体重	0.608260	0.709167	1.000000

不难看出,年龄与体重之间的相关系数为 0.608260(显著线性相关),与身高之间的相关系数为 0.463185(显著线性相关),身高与体重之间的相关系数为 0.709167(高度线性相关)。下面对年龄、身高和体重两两之间制作散点图,如图 3.6 所示。

按照性别进行分类,暂不关心哪些点是男(女)。在图 3.6 中,年龄与身高的点较分散,与体重的点也较为分散,但整体而言,身高和体重都随年龄的增长而增大,符合常识并满足正相关性的结果。体重与身高之间的相关系数为 0.709167,趋势上更为明显。其代码如下所示。

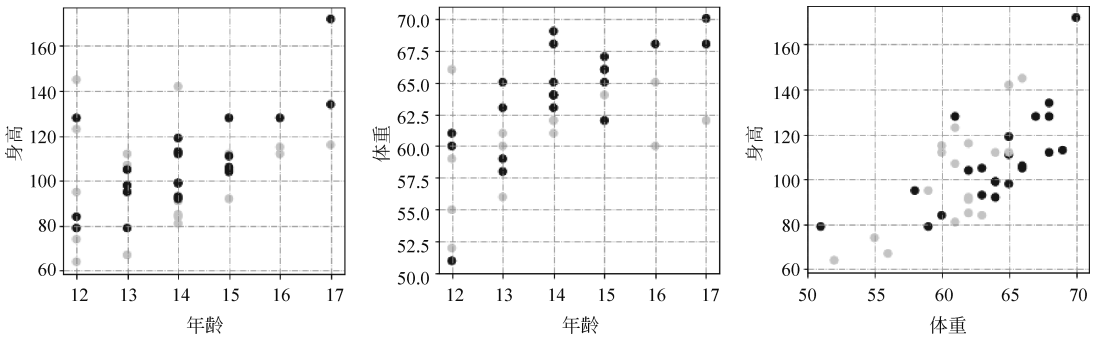

图 3.6　3 个变量之间的散点图

```
#复制
scatter_dict = {
    0:[['年龄', '身高'], [data_df['年龄'], data_df['身高']]],
    1:[['年龄', '体重'], [data_df['年龄'], data_df['体重']]],
    2:[['体重', '身高'], [ data_df['体重'], data_df['身高']]],}
#散点图
fig, axes = plt.subplots(1, 3, figsize = (14, 4))
for key, value in scatter_dict.items():
    axes[key].scatter(value[1][0], value[1][1], c = data_df['性别_数值'])
    axes[key].set_xlabel('{}'.format(value[0][0]))
    axes[key].set_ylabel('{}'.format(value[0][ - 1]))
    axes[key].grid(True, linestyle = '-.')
#plt.savefig("scatter.png")
```

2. 最小二乘法

对以上问题再深入探讨,先引入以下概念。

定义 3.7　最小二乘法

由于线性方程组(见式(3.22))可以写成 $\boldsymbol{Ax}=\boldsymbol{b}$ 的形式,现考虑最小二乘问题。

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \| \boldsymbol{Ax}-\boldsymbol{b} \|_2^2 \tag{3.27}$$

其中,$\boldsymbol{A}\in\mathbb{R}^{m\times n}$,$\boldsymbol{b}\in\mathbb{R}^m$。

- 当 $m=n$ 时,且 $\boldsymbol{A}$ 非奇异,即为一个线性方程组,解为 $\boldsymbol{x}=\boldsymbol{A}^{-1}\boldsymbol{b}$;
- 当 $m>n$ 时,约束个数大于未知量个数,此时称问题为超定的(overde-termined);
- 当 $m<n$ 时,未知量个数大于约束个数,此时称问题为欠定的(underde-termined)。

本书主要讨论超定的最小二乘问题。当 $m>n$ 时,线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的解可能不存在,这时一般考虑求最小二乘法问题。记为:

$$J(\boldsymbol{x})=\| \boldsymbol{Ax}-\boldsymbol{b} \|_2^2 \tag{3.28}$$

显然,$J(\boldsymbol{x})$是关于 $\boldsymbol{x}$ 的二次函数,而且是凸函数(Hessen 阵半正定)。$\boldsymbol{x}$ 是问题(见式(3.27))的解当且仅当 $\boldsymbol{x}$ 是 $J(\boldsymbol{x})$的稳定点(但解可能不唯一)。令其一阶导数为 0,即

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{Ax}-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}=0 \tag{3.29}$$

进而 $\boldsymbol{x}$ 的解为

$$\boldsymbol{x}=(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} \tag{3.30}$$

这里需要考虑 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ 是否可逆，通常在实际问题中的特征不存在完全正相关时，均存在可逆。当特征之间的线性相关性较强时，需要考虑多重共线性问题。

现在不妨假设身高是年龄和体重的线性组合 $y=a_1x_1+a_2x_2$（x_1 表示年龄，x_2 表示体重），通过式(3.30)求解。其代码如下所示。

```
from numpy import dot, transpose
#自变量,因变量
data_copy_df['constant'] = 1            #常数项
x_arr = data_copy_df[['年龄', '体重']].values
y_arr = data_copy_df[['身高']].values
#(A^{\top} A)^{-1}
left_value = inv(dot(transpose(x_arr), x_arr))
#A^{\top} b
right_value = dot(transpose(x_arr), y_arr)
#系数
coeff_arr = dot(left_value, right_value)
array([[0.44644474],
       [1.58801122]])
```

即 $y=0.44644474x_1+1.58801122x_2$，显然体重对身高的贡献程度最大。若考虑含有常数项，则身高是年龄和体重的线性组合，令 $y=a_1x_1+a_2x_2+\mathrm{constant}$，则计算结果为：

```
array([[ 0.75983085],
[    3.55054442],
[ -127.7051895 ]])
```

即 $y=0.75983085x_1+3.55054442x_2-127.7051895$。现在这两种结果哪个更好呢？这就需要一个评判函数，这里通过均方误差来衡量。

定义 3.8　均方误差

对于因变量 y_i，其对应的预测值为 $\hat{y}_i$，有：

$$\mathrm{RMSE}=\sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i-\hat{y}_i)^2} \tag{3.31}$$

RMSE 就是经常用到的均方误差函数，值越小说明预测值和实际值之间的差异越小。有时候也写成 $\mathrm{RMSE}=\sqrt{\frac{1}{m}\sum_{i=1}^{m}w_i(y_i-\hat{y}_i)^2}$。

通过式(3.31)对考虑含有常数项与不含常数项的线性回归计算，其误差分别为 15.431 和 17.629。似乎说明拟合函数 $y=0.75983085x_1+3.55054442x_2-127.7051895$ 更为合理。事实真是如此吗？结合生物学常识对问题进行全面剖析，这组数据是关于某校未成年学生的数据，身高除了与体重和年龄有关之外，还与性别有关。若读者感兴趣可以

进行深入探讨,这里不再深入分析这个问题。

3.7 本章小结

本章主要介绍了部分基本统计量,比如算术平均数、方差、变异系数以及相关系数等。对数据进行分析的前期,会有大量的时间用于数据预处理和数据转换,本章给出了几种常用的数据转换方法,比如中心化、min-max 标准化以及 z-score 标准化等。在多维空间中经常需要计算各种距离,本章介绍了几种常见的距离。针对这些基本概念,通过 Python 来实现其计算过程,最后给出了一个分析实例。数据科学是一门交叉学科,涉及内容非常广泛,如心理学、社会学、数学、物理学和统计学等领域,因此要求从事数据科学的人要不断学习。