第 3 章　数据清洗与特征预处理

数据处理是建立机器学习模型的第一步，对最终结果有决定性的作用。本章重点介绍数据清洗与特征预处理。其中，数据清洗是指对缺失值、异常值和重复值等进行处理；特征预处理是指通过规范化、标准化、鲁棒化和正则化等方法将数据转化成符合算法要求的数据。最后介绍 missingno 库和词云，它们用于可视化显示数据相关信息。

3.1　数据清洗

3.1.1　数据清洗简介

在处理数据之前，需要进行数据质量分析，了解数据的功能和作用，检查原始数据中是否存在脏数据。脏数据一般是指不符合要求以及不能直接进行相应分析的数据。

脏数据往往存在如下问题：没有列头，一个列有多个参数，列数据的单位不统一，存在缺失值、空行、重复数据和非 ASCII 字符，有些列头应该是数据而不应该是列名参数，等等。可将这些问题大致归类为缺失值、异常值和重复值等噪声数据问题。而数据清洗就是发现并处理这些数据问题。

3.1.2　评价标准

对于数据的评价一般具有如下标准：

(1) 精确性。描述数据是否与其对应的客观实体的特征一致。

(2) 完整性。描述数据是否存在缺失记录或缺失字段。

(3) 一致性。描述同一实体的同一属性的值在不同系统中是否一致。

(4) 有效性。描述数据是否满足用户定义的条件或在一定的域值范围内。

(5) 唯一性。描述数据是否存在重复记录。

3.2　清洗方法

3.2.1　缺失值

缺失值通常是指记录的缺失和记录中某个字段信息的缺失，一般以空白、NaN 或其他占位符编码，采用删除法和数据填充进行处理。

- 删除法。如果某个属性的缺失值过多，可以直接删除整个属性。
- 数据填充。使用一个全局变量填充缺失值，使用属性的平均值、中间值、最大值、

最小值或更为复杂的概率统计函数值填充缺失值。

常用填充方法如表 3.1 所示。

表 3.1 常用填充方法

填充方法	方法描述
平均值/中位数	根据属性值的类型,用该属性取值的平均值/中位数填充
固定值	将缺失的属性值用一个常量替换
最近值	用最接近缺失值的属性值填补

Sklearn 中的 Imputer 类或 SimpleImputer 类用于处理缺失值。其中,Imputer 在 preprocessing 模块中,而 SimpleImputer 在 sklearn.impute 模块中。

Imputer 具体语法如下:

```
from sklearn.preprocessing import Imputer
imp=Imputer(missing_values="NaN", strategy="mean")
```

SimpleImputer 具体语法如下:

```
from sklearn.impute import SimpleImputer
imp=SimpleImputer(missing_values=np.nan, strategy="mean")
```

参数含义如下:

- missing_values=np.nan:缺失值是 NaN。
- strategy="mean":用平均值、中位数等填充缺失值。

【例 3.1】 缺失值处理示例。

```
import pandas as pd
import numpy as np
#from sklearn.preprocessing import Imputer
from sklearn.impute import SimpleImputer
df=pd.DataFrame([["XXL", 8, "black", "class 1", 22],
["L", np.nan, "gray", "class 2", 20],
["XL", 10, "blue", "class 2", 19],
["M", np.nan, "orange", "class 1", 17],
["M", 11, "green", "class 3", np.nan],
["M", 7, "red", "class 1", 22]])
df.columns=["size", "price", "color", "class", "boh"]
print(df)
#1. 创建 Imputer
#imp=Imputer(missing_values="NaN", strategy="mean" )
imp=SimpleImputer(missing_values=np.nan, strategy="mean")
#2. 使用 fit_transform 函数完成缺失值填充
df["price"]=imp.fit_transform(df[["price"]])
print(df)
```

【程序运行结果】

```
   size  price   color    class   boh
0   XXL    8.0   black  class 1  22.0
1     L    NaN    gray  class 2  20.0
2    XL   10.0    blue  class 2  19.0
3     M    NaN  orange  class 1  17.0
4     M   11.0   green  class 3   NaN
5     M    7.0     red  class 1  22.0

   size  price   color    class   boh
0   XXL    8.0   black  class 1  22.0
1     L    9.0    gray  class 2  20.0
2    XL   10.0    blue  class 2  19.0
3     M    9.0  orange  class 1  17.0
4     M   11.0   green  class 3   NaN
5     M    7.0     red  class 1  22.0
```

3.2.2 异常值

异常值指录入错误以及不合常理的数据，一般采用箱形图和标准差两种办法进行识别，分别介绍如下。

1. 采用箱形图识别异常值

在箱形图中判别异常值的方法见 2.5.5 节。异常值用中位数填充。

```
import numpy as np                                  #导入 NumPy 库
import pandas as pd                                 #导入 Pandas 库
a=data["数量"].quantile(0.75)
b=data["数量"].quantile(0.25)
c=data["数量"]
c[(c>=(a-b)*1.5+a)|(c<=b-(a-b)*1.5)]=np.nan
c.fillna(c.median(),inplace=True)
print(c.describe())
```

2. 采用标准差识别异常值

当数据服从正态分布时，异常值被定义为一组测定值中与平均值的偏差超过 3 倍标准差的值。

```
import numpy as np                                  #导入 NumPy 库
import pandas as pd                                 #导入 Pandas 库
```

```
a=data["数量"].mean()+data["数量"].std() * 3
b=data["数量"].mean()-data["数量"].std() * 3
c=data["数量"]
c[(c>=a)|(c<=b)]=np.nan
c.fillna(c.median(),inplace=True)
print(c.describe())
```

常用异常值处理方法如表 3.2 所示。

表 3.2 常用异常值处理方法

方 法	方 法 描 述
删除	直接删除含有异常值的记录
视为缺失值	利用缺失值处理方法填充
平均值修正	用前后两个观测值的平均值修正异常值

【例 3.2】 通过 z-score 法判断异常值示例。

```
import pandas as pd                          #导入 Pandas 库
import matplotlib.pyplot as plt
#构建包含异常值的矩阵
df=pd.DataFrame([[1,12],[120,17],[3,31],[5,53],[2,22],[12,32],[13,43]],
columns=['col1','col2'])
print("数据为:\n",df)                        #打印输出
#散点图
plt.scatter(df['col1'],df['col2'])
plt.show()
#通过 z-score 方法判断异常值,超过阈值为异常值
df_zscore=df.copy()                          #存储 z-score
cols=df.columns                              #获得数据框的列名
for col in cols:                             #循环读取每列
    df_col=df[col]                           #得到每列的值
    z_score=(df_col-df_col.mean())/df_col.std()     #计算每列的 z-score
    df_zscore[col]=z_score.abs() >2.2  #阈值为 2.2,大于该值为 True,否则为 False
print("异常值为:\n",df_zscore)               #打印输出
df_drop_outlier=df[df_zscore['col1']==False]     #删除异常值所在的记录行
print("处理后的数据为:\n",df_drop_outlier)
```

【程序运行结果】

```
数据为:
   col1  col2
0     1    12
1   120    17
2     3    31
3     5    53
4     2    22
5    12    32
6    13    43
```

```
异常值为：
    col1   col2
0  False  False
1   True  False
2  False  False
3  False  False
4  False  False
5  False  False
6  False  False
处理后的数据为：
   col1  col2
0     1    12
2     3    31
3     5    53
4     2    22
5    12    32
6    13    43
```

程序运行结果如图 3.1 所示。

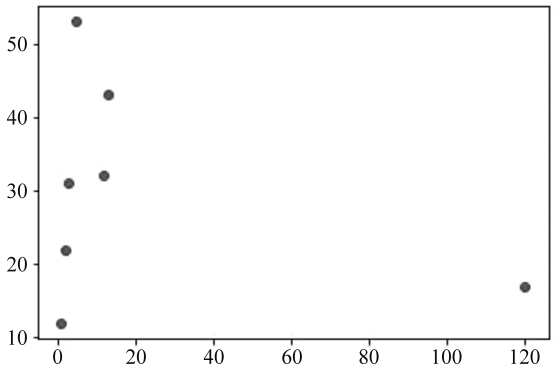

图 3.1　例 3.2 程序运行结果

3.2.3　重复值

重复值的存在会影响数据分析的准确性。目前消除重复值的基本思想是排序和合并。对数据进行排序后，比较邻近数据是否相似来检测数据是否重复。

消除重复数据的算法主要有优先队列算法、近邻排序法（sorted-neighborhood method）和多趟近邻排序法（multi-pass sorted-neighborhood method）。

3.2.4　Pandas 数据清洗函数

Pandas 提供了如下函数用于数据清洗和预处理：

- df.duplicated：判断各行是否重复，False 为非重复值。
- df.drop_duplicates：删除重复行。
- df.fillna(num)：用实数 num 填充缺失值。
- df.dropna：删除 DataFrame 数据中的缺失值，即删除 NaN 数据。其语法为

```
DataFrame.dropna(axis=0, how='any', thresh=None, subset=None, inplace=False)
```

参数说明如表 3.3 所示。

表 3.3 df.dropna 函数的参数说明

参　数	说　明
axis	0 为行，1 为列
how	any 表示删除带有 NaN 的行，all 表示删除全为 NaN 的行
thresh	取值为 int 型，保留至少指定个非 NaN 行
subset	取值为 list 型，在特定列处理缺失值
inplace	True 表示修改原文件，False 表示不修改原文件

- del df['col1']：直接删除某列。
- df.drop(['col1',…],axis=1)：删除指定列，也可以删除指定行。
- df.rename(index={'row1': 'A'},columns ={'col1': 'B'})：重命名索引名和列名。
- df.replace()：替换 DataFrame，可以用字典表示，例如{'1': 'A','2': 'B'}。
- df[].map(function)：对指定列进行函数转换。map 是 Series 中的函数。
- pd.merge(df1,df2,on='col1',how='inner',sort=True)：合并两个 DataFrame，按照共有的列作内连接(交集)，outer 为外连接(并集)，对结果排序。
- df1.combine_first(df2)：用 df2 的数据补充 df1 的缺失值。

【例 3.3】 重复值处理示例。

```
import pandas as pd                          #导入 Pandas 库
#生成异常数据
data1, data2, data3, data4=['a', 3], ['b', 2], ['a', 3], ['c', 2]
df=pd.DataFrame([data1, data2, data3, data4], columns=['col1', 'col2'])
print("数据为:\n",df)                        #打印输出

isDuplicated=df.duplicated()                 #判断重复数据记录
print("重复值为:\n",isDuplicated)            #打印输出
print("删除数据记录中所有列值相同的记录:\n",df.drop_duplicates())
print("删除数据记录中 col1 值相同的记录:\n",df.drop_duplicates(['col1']))
print("删除数据记录中 col2 值相同的记录:\n",df.drop_duplicates(['col2']))
print("删除数据记录中指定列(col1/col2)值相同的记录:\n",df.drop_duplicates
(['col1', 'col2']))
```

【程序运行结果】

```
数据为:
   col1  col2
0     a     3
1     b     2
2     a     3
3     c     2
重复值为:
0  False
1  False
2   True
3  False
dtype: bool
删除数据记录中所有列值相同的记录:
   col1  col2
0     a     3
1     b     2
3     c     2
删除数据记录中 col1 值相同的记录:
   col1  col2
0     a     3
1     b     2
3     c     2
删除数据记录中 col2 值相同的记录:
   col1  col2
0     a     3
1     b     2
删除数据记录中指定列(col1/col2)值相同的记录:
   col1  col2
0     a     3
1     b     2
3     c     2
```

【例 3.4】 df.fillna(num)示例。

```
from numpy import nan as NaN
import pandas as pd
df1=pd.DataFrame([[1,2,3],[NaN,NaN,2],[NaN,NaN,NaN],[8,8,NaN]])
print("df1:\n{}\n".format(df1))
df2=df1.fillna(100)
print("df2:\n{}\n".format(df2))
```

【程序运行结果】

```
df1:
```

```
     0    1    2
0  1.0  2.0  3.0
1  NaN  NaN  2.0
2  NaN  NaN  NaN
3  8.0  8.0  NaN

df2:
       0      1      2
0    1.0    2.0    3.0
1  100.0  100.0    2.0
2  100.0  100.0  100.0
3    8.0    8.0  100.0
```

【例 3.5】 df.dropna 示例。

```
from numpy import nan as NaN
import pandas as pd
df1=pd.DataFrame([[1,2,3],[NaN,NaN,2],[NaN,NaN,NaN],[8,8,NaN]])
print("df1:\n{}\n".format(df1))
df2=df1.dropna()
print("df2:\n{}\n".format(df2))
```

【程序运行结果】

```
df1:
     0    1    2
0  1.0  2.0  3.0
1  NaN  NaN  2.0
2  NaN  NaN  NaN
3  8.0  8.0  NaN

df2:
     0    1    2
0  1.0  2.0  3.0
```

【例 3.6】 df.replace 示例。

```
import pandas as pd
#创建数据集
df=pd.DataFrame(
        {   '名称':['产品 1','产品 2','产品 3','产品 4','产品 5','产品 6','产品 7',
            '产品 8'],
            '数量':['A','0.7','0.8','0.4','0.7','B','0.76','0.28'],
            '金额':['0','0.48','0.33','C','0.74','0','0','0.22'],
            '合计':['D','0.37','0.28','E','0.57','F','0','0.06'], }
        )
#原 DataFrame 并没有改变,改变的只是一个副本
```

```
print("df:\n{}\n".format(df))
df1=df.replace('A', 0.1)
print("df1:\n{}\n".format(df1))
#只需要替换某个数据的部分内容
df2=df['名称'].str.replace('产品', 'product')
print("df2:\n{}\n".format(df2))
#如果要改变原数据,应添加常用参数 inplace=True,用于替换部分区域
df['合计'].replace({'D':0.11111, 'F':0.22222}, inplace=True)
print("df:\n{}\n".format(df))
```

【程序运行结果】

```
df:
   合计    名称  数量  金额
0     D  产品 1     A     0
1  0.37  产品 2   0.7  0.48
2  0.28  产品 3   0.8  0.33
3     E  产品 4   0.4     C
4  0.57  产品 5   0.7  0.74
5     F  产品 6     B     0
6     0  产品 7  0.76     0
7  0.06  产品 8  0.28  0.22

df1:
   合计    名称  数量  金额
0     D  产品 1   0.1     0
1  0.37  产品 2   0.7  0.48
2  0.28  产品 3   0.8  0.33
3     E  产品 4   0.4     C
4  0.57  产品 5   0.7  0.74
5     F  产品 6     B     0
6     0  产品 7  0.76     0
7  0.06  产品 8  0.28  0.22

df2:
0  product1
1  product2
2  product3
3  product4
4  product5
5  product6
6  product7
7  product8
Name: 名称, dtype: object
```

```
df:
      合计    名称  数量  金额
0  0.11111  产品 1     A     0
1     0.37  产品 2   0.7  0.48
2     0.28  产品 3   0.8  0.33
3        E  产品 4   0.4     C
4     0.57  产品 5   0.7  0.74
5  0.22222  产品 6     B     0
6        0  产品 7  0.76     0
7     0.06  产品 8  0.28  0.22
```

【例 3.7】 df[].map 示例。

```
import pandas as pd
import numpy as np
df=pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                 'key2' : ['one', 'two', 'one', 'two', 'one'],
                 'data1' : np.arange(5),
                 'data2' : np.arange(5,10)})
print("df:\n{}\n".format(df))
df['data1']=df['data1'].map(lambda x : "%.3f"%x)
print("df:\n{}\n".format(df))
```

【程序运行结果】

```
df:
   data1  data2  key1  key2
0      0      5     a   one
1      1      6     a   two
2      2      7     b   one
3      3      8     b   two
4      4      9     a   one

df:
   data1  data2  key1  key2
0  0.000      5     a   one
1  1.000      6     a   two
2  2.000      7     b   one
3  3.000      8     b   two
4  4.000      9     a   one
```

【例 3.8】 pd.merge(df1,df2)示例。

```
import pandas as pd
left=pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                           'A': ['A0', 'A1', 'A2', 'A3'],
                           'B': ['B0', 'B1', 'B2', 'B3']})
```

```
right=pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                            'C': ['C0', 'C1', 'C2', 'C3'],
                            'D': ['D0', 'D1', 'D2', 'D3']})
result=pd.merge(left, right, on='key')
#on 参数传递的 key 作为连接键
print("left:\n{}\n".format(left))
print("right:\n{}\n".format(right))
print("merge:\n{}\n".format(result))
```

【程序运行结果】

```
left:
    A   B  key
0  A0  B0   K0
1  A1  B1   K1
2  A2  B2   K2
3  A3  B3   K3

right:
    C   D  key
0  C0  D0   K0
1  C1  D1   K1
2  C2  D2   K2
3  C3  D3   K3

merge:
    A   B  key   C   D
0  A0  B0   K0  C0  D0
1  A1  B1   K1  C1  D1
2  A2  B2   K2  C2  D2
3  A3  B3   K3  C3  D3
```

【例 3.9】 df1.combine_first(df2)示例。

```
from numpy import nan as NaN
import numpy as np
import pandas as pd
a=pd.Series([np.nan,2.5,np.nan,3.5,4.5,np.nan],index=['f','e','d','c','b',
'a'])
b=pd.Series([1,np.nan,3,4,5,np.nan],index=['f','e','d','c','b','a'])
print(a)
print(b)
c=b.combine_first(a)
print(c)
```

【程序运行结果】

```
f  NaN
e  2.5
d  NaN
c  3.5
b  4.5
a  NaN
dtype: float64
f  1.0
e  NaN
d  3.0
c  4.0
b  5.0
a  NaN
dtype: float64
f  1.0
e  2.5
d  3.0
c  4.0
b  5.0
a  NaN
dtype: float64
```

3.3 特征预处理

有一句话在业界广泛流传："数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。"这里的数据是指经过特征预处理后的数据。特征预处理就是对数据进行集成、转换、规约等一系列处理，使之适合算法模型的过程。

Sklearn 提供了 preprocessing 模块，用于进行归一化、标准化、鲁棒化、正则化等数据预处理。preprocessing 模块常用方法如表 3.4 所示。

表 3.4 preprocessing 模块常用方法

方 法 名	方法含义
preprocessing. MinMaxScaler	归一化
preprocessing. StandardScaler	标准化
preprocessing. RobustScaler	鲁棒化
preprocessing.normalize	正则化

3.3.1 归一化

归一化又称为区间缩放法，是数据处理的必要工作。由于不同评价指标往往具有不同的量纲，数值差别较大，从而影响数据分析的结果。为了让特征具有同等重要性，可以采用归一化(MinMaxScaler)将不同规格的数据转换到同一个规格。归一化利用边界值信息将特征的取值区间缩放到某个特点的范围，例如[0，1]等。

归一化计算公式如下：

$$X' = \frac{x - \min}{\max - \min}$$

$$X'' = X'(\mathrm{mx} - \mathrm{mi}) + \mathrm{mi}$$

参数解释如下：

- max：最大值。
- min：最小值。
- mx，mi：用于指定区间，默认 mx 为 1，mi 为 0。

归一化将原始数据通过线性变换缩放到[0，1]。由于异常值往往是最大值或最小值，所以归一化的鲁棒性较差。

Sklearn 提供了 MinMaxScaler 方法进行归一化，具体语法如下：

```
MinMaxScaler(feature_range=(0,1))
```

参数 feature_range=(0，1)将范围设置为 0～1。

【例 3.10】 归一化示例。

现有 3 个样本，每个样本有 4 个特征，如表 3.5 所示。

表 3.5 样本特征

特征 1	特征 2	特征 3	特征 4
90	2	10	40
60	4	15	45
75	3	13	46

```
from sklearn.preprocessing import MinMaxScaler
def Normalization():                                    #实例化一个转换器类
    Normalization=MinMaxScaler(feature_range=(0,1))     #范围设置为 0~1
    data=[[90,2,10,40],[60,4,15,45],[75,3,13,46]]
    print(data)
    #调用 fit_transform
    data_Normal=Normalization.fit_transform(data)
    print(data_Normal)
    return None
if __name__=='__main__':
```

```
    Normalization()
```

【程序运行结果】

```
[[90, 2, 10, 40], [60, 4, 15, 45], [75, 3, 13, 46]]
[[1.         0.         0.         0.         ]
 [0.         1.         1.         0.83333333 ]
 [0.5        0.5        0.6        1.         ]]
```

3.3.2 标准化

标准化(standardization)用于解决归一化容易受到样本中最大值或者最小值等异常值的影响的问题,将数据按比例缩放到特定区间。

标准差公式如下:

$$\sigma=\sqrt{\frac{1}{n}\sum_{i=1}^{N}(x_i-\mu)^2}$$

z-score 标准化转换公式如下:

$$z=\frac{x-\mu}{\sigma}$$

标准化的前提是特征值服从正态分布。进行标准化后,数据聚集在 0 附近,方差为 1,有利于模型的训练。Sklearn 提供了 StandardScaler 方法实现标准化,具体语法如下:

```
StandardScaler(copy, with_mean)
```

参数含义如下:

- copy:取值为 True 或 False。在用归一化的值替代原来的值时设置为 False。
- with_mean:取值为 True 或 False。在处理稀疏矩阵时设置为 False。

【例 3.11】 标准化示例。

```
from sklearn.preprocessing import StandardScaler
def Standardization():
    '''标准化函数'''
    std=StandardScaler()
    data=[[1.,-1.,3.],[2.,4.,2.],[4.,6.,-1.]]
    print(data)
    data_Standard=std.fit_transform(data)
    print(data_Standard)
    return None
if __name__=='__main__':
Standardization()
```

【程序运行结果】

```
[[1.0, -1.0, 3.0], [2.0, 4.0, 2.0], [4.0, 6.0, -1.0]]
```

```
[[-1.06904497  -1.35873244   0.98058068]
 [-0.26726124   0.33968311   0.39223227]
 [ 1.33630621   1.01904933  -1.37281295]]
```

3.3.3 鲁棒化

当数据包含许多异常值时，使用平均值和方差缩放均不能取得较好效果，可以使用鲁棒性缩放（RobustScaler）方法进行处理。

preprocessing 模块的 RobustScaler 方法使用中位数和四分位数进行数据转换，直接将异常值剔除，具体语法如下：

```
RobustScaler(quantile_range, with_centering, with_scaling)
```

参数含义如下：

- with_centering：布尔值，默认值为 True，表示在缩放之前将数据居中。
- with_scaling：布尔值，默认值为 True，表示将数据缩放到四分位数范围。
- quantile_range：元组，默认值为（25.0，75.0），即 IQR，表示用于计算 scale_的分位数范围。

【例 3.12】 鲁棒化示例。

```
from sklearn.preprocessing import RobustScaler
X=[[ 1., -2.,  2.],[ -2.,  1.,  3.],[ 4.,  1., -2.]]
transformer=RobustScaler().fit(X)
RobustScaler(quantile_range=(25.0,75.0),with_centering=True,with_scaling=
True)
print(transformer.transform(X))
```

【程序运行结果】

```
[[ 0.  -2.   0. ]
 [-1.   0.   0.4]
 [ 1.   0.  -1.6]]
```

3.3.4 正则化

正则化（normalization）是将每个样本缩放到单位范式，使数据分布在一个半径为 1 的圆或者球内。preprocessing 模块提供了 normalize 方法以实现正则化，具体语法如下：

```
normalize(X, norm='l2')
```

参数含义如下：

- X：样本数据。
- norm='l2'：L2 范式。

【例 3.13】 正则化示例。

```
from sklearn.preprocessing import normalize
X=[[ 1., -1.,  2.],[ 2.,  0.,  0.],[ 0.,  1., -1.]]
X_normalized=normalize(X, norm='l2')
print(X_normalized)
```

【程序运行结果】

```
[[ 0.40824829 -0.40824829   0.81649658]
 [ 1.          0.           0.        ]
 [ 0.          0.70710678  -0.70710678]]
```

3.3.5 学生数据清洗示例

【例 3.14】 学生数据清洗示例。

初始化并显示数据：

```
import pandas as pd
import numpy as np
from collections import Counter
from sklearn import preprocessing
from matplotlib import pyplot as plt
import seaborn as sns
plt.rcParams['font.sans-serif']=['SimHei'] #中文字体设置为黑体
plt.rcParams['axes.unicode_minus']=False     #解决保存图像时负号显示为方块的问题
sns.set(font='SimHei')                       #解决 Seaborn 中文显示问题
data=pd.read_excel("d:/dummy.xls")           #在 D 盘根目录下创建 dummy.xls 文件
print(data)
```

【程序运行结果】

```
   姓名  学历    成绩    能力  学校
0  小红  博士    90.0  100.0  同济
1  小黄  硕士    90.0   89.0  交大
2  小绿  本科    80.0   98.0  同济
3  小白  硕士    90.0   99.0  复旦
4  小紫  博士   100.0   78.0  同济
5  小城  本科    80.0   98.0  交大
6  校的   NaN     NaN    NaN   NaN
```

显示序列的前 n 行(默认值)：

```
print("data head:\n",data.head())
```

【程序运行结果】

```
data head:
   姓名  学历    成绩    能力  学校
0  小红  博士    90.0  100.0  同济
1  小黄  硕士    90.0   89.0  交大
2  小绿  本科    80.0   98.0  同济
3  小白  硕士    90.0   99.0  复旦
4  小紫  博士   100.0   78.0  同济
```

查看数据的行列大小：

```
print("data shape:\n",data.shape)
```

【程序运行结果】

```
data shape:
(7, 5)
```

显示指定列的数据描述属性值：

```
print("data descibe:\n",data.describe())
```

【程序运行结果】

```
data descibe:
              成绩          能力
count     6.000000     6.000000
mean     88.333333    93.666667
std       7.527727     8.640988
min      80.000000    78.000000
25%      82.500000    91.250000
50%      90.000000    98.000000
75%      90.000000    98.750000
max     100.000000   100.000000
```

进行列级别的判断，只要某一列有 NaN 或值为空，则为真：

```
data.isnull().any()
```

将列中为 NaN 或值为空的个数统计出来，并将缺失值最多的排在前面：

```
total=data.isnull().sum().sort_values(ascending=False)
print("total:\n",total)
```

【程序运行结果】

```
total:
学校    1
能力    1
成绩    1
```

```
学历    1
姓名    0
```

输出百分比：

```
percent=(data.isnull().sum()/data.isnull().count()).sort_values(ascending=
False)
missing_data=pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

导入 missingno 并删除缺失值：

```
import missingno                          #missingno 用于可视化缺失值
missingno.matrix(data)
data=data.dropna(thresh=data.shape[0] * 0.5,axis=1)
    #将至少有一半以上是非空的列筛选出来
    #如果某一行都是 NaN 才删除,默认只保留没有空值的行
data.dropna(axis=0,how='all')
print(data)
```

统计重复记录数：

```
data.duplicated().sum()
data.drop_duplicates()
data.columns                              #对数据中的连续型字段和离散型字段进行归类
id_col=['姓名']
cat_col=['学历','学校']                    #离散型无序
cont_col=['成绩','能力']                   #数值型
print (data[cat_col])                     #离散型数据部分
print (data[cont_col])                    #连续型数据部分
```

计算出现的频次：

```
for i in cat_col:
    print(pd.Series(data[i]).value_counts())
    plt.plot(data[i])
```

对于离散型数据,对其获取哑变量：

```
dummies=pd.get_dummies(data[cat_col])
print("哑变量:\n",dummies)
```

【程序运行结果】

```
哑变量:
   学历_博士  学历_本科  学历_硕士  学校_交大  学校_同济  学校_复旦
0         1         0         0         0         1         0
1         0         0         1         1         0         0
2         0         1         0         0         1         0
3         0         0         1         0         0         1
4         1         0         0         0         1         0
```

```
5          0          1          0          1          0          0
6          0          0          0          0          0          0
```

对连续型数据进行统计：

```
data[cont_col].describe()
```

对连续型数据，将偏度大于 0.75 的数值用取对数的方法进行转换，使之符合正态分布：

```
skewed_feats=data[cont_col].apply(lambda x: (x.dropna()).skew() )  #计算偏度
skewed_feats=skewed_feats[skewed_feats>0.75]
skewed_feats=skewed_feats.index
data[skewed_feats]=np.log1p(data[skewed_feats])
#print(skewed_feats)
```

对连续型数据进行标准化：

```
scaled=preprocessing.scale(data[cont_col])
scaled=pd.DataFrame(scaled,columns=cont_col)
print(scaled)
m=dummies.join(scaled)
data_cleaned=data[id_col].join(m)
print("标准化:\n",data_cleaned)
```

【程序运行结果】

```
标准化:
   姓名 学历_博士 学历_本科 学历_硕士 学校_交大 学校_同济 学校_复旦       成绩       能力
0 小红         1         0         0         0         1         0  0.242536  0.802897
1 小黄         0         0         1         1         0         0  0.242536 -0.591608
2 小绿         0         1         0         0         1         0 -1.212678  0.549350
3 小白         0         0         1         0         0         1  0.242536  0.676123
4 小紫         1         0         0         0         1         0  1.697749 -1.986112
5 小城         0         1         0         1         0         0 -1.212678  0.549350
6 校的         0         0         0         0         0         0       NaN       NaN
```

显示变量之间的相关性：

```
print("变量之间的相关性:\n",data_cleaned.corr())
```

【程序运行结果】

```
变量之间的相关性:
            学历_博士   学历_本科   学历_硕士   学校_交大   学校_同济   学校_复旦
学历_博士    1.000000 -0.400000 -0.400000 -0.400000  0.730297 -0.258199
学历_本科   -0.400000  1.000000 -0.400000  0.300000  0.091287 -0.258199
学历_硕士   -0.400000 -0.400000  1.000000  0.300000 -0.547723  0.645497
学校_交大   -0.400000  0.300000  0.300000  1.000000 -0.547723 -0.258199
学校_同济    0.730297  0.091287 -0.547723 -0.547723  1.000000 -0.353553
学校_复旦   -0.258199 -0.258199  0.645497 -0.258199 -0.353553  1.000000
成绩         0.685994 -0.857493  0.171499 -0.342997  0.242536  0.108465
能力        -0.418330  0.388449  0.029881 -0.014940 -0.211289  0.302372
```

```
      成绩        能力
 0.685994  -0.418330
-0.857493   0.388449
 0.171499   0.029881
-0.342997  -0.014940
 0.242536  -0.211289
 0.108465   0.302372
 1.000000  -0.748177
-0.748177   1.000000
```

绘制相关性的热力图：

```
def corr_heat(df):
    dfData=abs(df.corr())
    plt.subplots(figsize=(9, 9))       #设置画面大小
    sns.heatmap(dfData, annot=True, vmax=1, square=True, cmap="Blues")
    #plt.savefig('./BluesStateRelation.png')
    plt.show()
corr_heat(data_cleaned)
```

程序运行结果如图 3.2 所示。

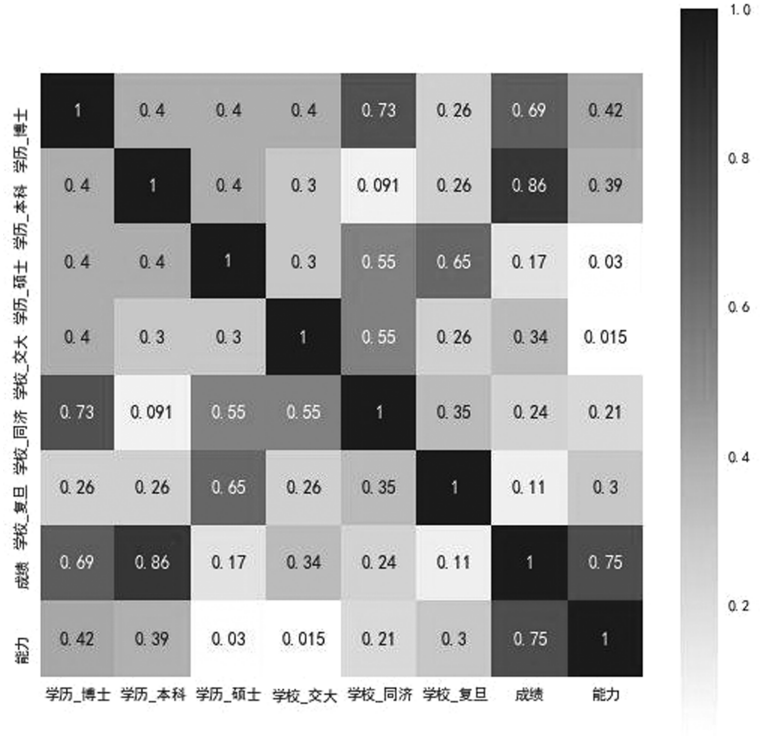

图 3.2　相关性的热力图