华东师范大学数据科学与工程学院实验报告

课程名称：数据挖掘	**年级：**2021 级	**上机实践成绩：**
指导教师：周昉	**姓名：**李睿恩	**学号：**10215501434
上机实践名称：Data Preprocessing		**上机实践日期：**2024.3.20
上机实践编号：1	**组号：**无	**上机实践时间：**22:14

一、 实验目的

- 完成数据预处理任务。

二、 实验任务

- 编写 Python 代码，实现数据预处理。
- 了解数据质量问题、数据聚合、采样、离散化与主成分分析。

三、 实验环境

本次实验基于 JetBrains 的 PyCharm 集成开发环境，使用 Python 编程语言实现。

四、 实验过程

4.1 数据质量问题

在实际应用中，我们得到的数据往往并不能直接使用。这主要是因为，我们收集的原始数据非常有可能存在数据质量问题。较差的数据质量可能会对数据挖掘产生不利影响。常见的数据质量问题包括噪声、异常值、缺失值和重复数据。在这里，我们会尝试解决一些常见的数据质量问题。

我们使用的数据是 breast-cancer-wisconsin.data，在这里我们对数据进行导入，同时导入一些之后可能会使用的模块。利用 Python，我们可以发现该数据集中存在 699 个实例，每个实例都有 11 个属性，如图 1 所示。

一种常见的数据质量问题是缺失值。在我们使用的数据中，缺失值的编码为“?”，为了更通用地处理整个数据表格，我们可以利用 data.replace('?', np.NaN)将缺失值转换为 NaN。这之后，我们还编写了一段代码以计算每列数据中缺失值的数量。代码如图 2 所示。

另外一种处理缺失值的方法是将缺失值替换为该列的中值。在这里，我们使用了 data.fillna(data.median())的代码将缺失值替换为了中值。第三种处理缺失值的方式是直接将这些缺失值的数据点删除，我们采用 data.dropna()即可实现。这部分代码如图 3 所示。

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv('breast+cancer+wisconsin+original/breast-cancer-wisconsin.data',
 header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size',
 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei',
                 'Bland Chromatin',
                'Normal Nucleoli', 'Mitoses','Class']
# 打印数据集实例个数和属性个数
print(f"实例个数: {data.shape[0]}")
print(f"属性个数: {data.shape[1]}")
```

Executed at 2024.03.19 19:37:09 in 52ms

```
实例个数: 699
属性个数: 11
```

图 1 导入模块与数据集

```python
data_nan = data.replace('?', np.NaN, inplace=False)
# print(data_nan.iloc[23])
# print(data.iloc[23])
for columnName in data_nan.columns:
    if data_nan[columnName].count() != len(data_nan):
        loc = data_nan[columnName][data_nan[columnName].isnull().values==True].index
         .tolist()
        print(f"{columnName}中有{len(loc)}个缺失值.")
    else:
        print(f"{columnName}中没有缺失值.")
```

Executed at 2024.03.19 19:37:10 in 17ms

```
Sample code中没有缺失值.
Clump Thickness中没有缺失值.
Uniformity of Cell Size中没有缺失值.
Uniformity of Cell Shape中没有缺失值.
Marginal Adhesion中没有缺失值.
Single Epithelial Cell Size中没有缺失值.
Bare Nuclei中有16个缺失值.
Bland Chromatin中没有缺失值.
Normal Nucleoli中没有缺失值.
Mitoses中没有缺失值.
Class中没有缺失值.
```

图 2 缺失值转为 NaN 并计算每列数据中缺失值的数量

```python
# 将缺失值替换为该列的中值
data_medium = data_nan.fillna(data_nan.median(), inplace=False)
# print(data_medium.iloc[23])
```

Executed at 2024.03.20 22:46:39 in 34ms

```python
# 丢弃包含缺失值的数据点，并打印丢弃后数据集的数据量
data_drop = data_nan.dropna(inplace=False)
print(data_drop.shape)
```

Executed at 2024.03.20 22:46:40 in 16ms

```
(683, 11)
```

图 3 将缺失值替换为一列的中值或丢弃包含缺失值的数据点

数据质量不佳也可能是因为存在过多异常值，而发现异常值的方法是绘制箱线图。在这里，由于第 1 个属性与第 11 个属性均是离散型变量，没有绘制箱线图的意义，因此我们只绘制第 2 个到第 9 个属性的箱线图。为了图像的绘制，我们选取了之前将缺失值换为中值的数据集用于绘制箱线图。

注意到在刚才的数据预处理中，我们发现只有 Bare Nuclei 属性中存在‘?’，因此这一列的值均是字符串对象，这是无法绘制箱线图的。因此，我们采用了 pandas.to_numeric 将这一列转为了整数。最终，我们利用 boxplot 函数绘制的箱线图如图 4 所示。

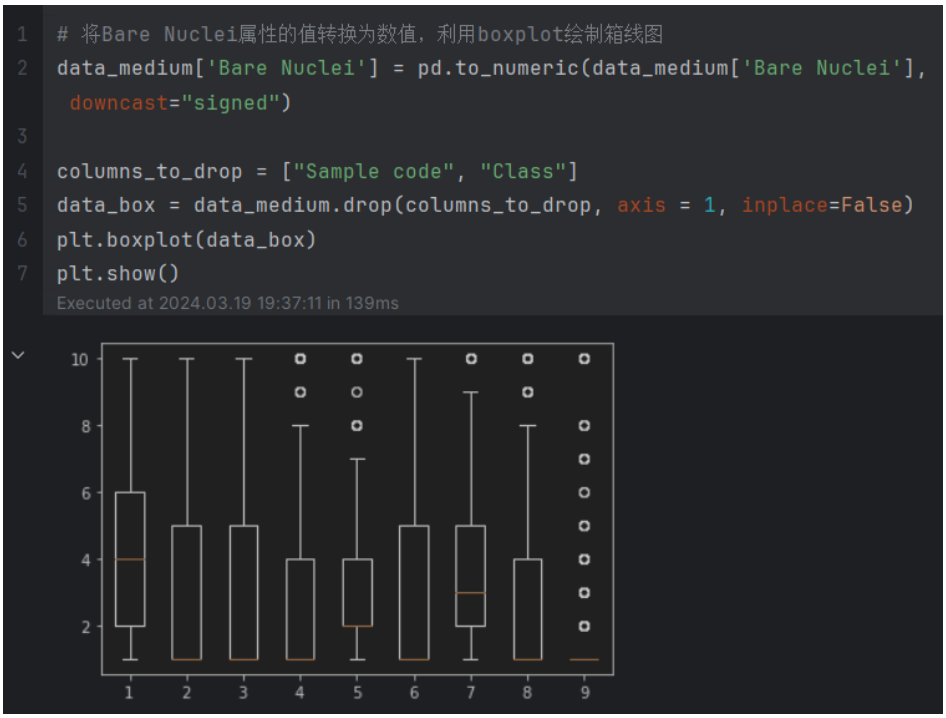

图 4 绘制箱线图

这样的箱线图充分体现出了我们的数据中存在异常值。为了丢弃异常值，我们可以计算每个属性的 Z 分数（即每个数据在该属性中标准化的结果，如图 5），并删除那些包含 Z 分数异常高或异常低的属性的实例。在这里，我们要求 9 个属性都必须小于等于 3 且大于-3，如果有一行数据中存在某个属性大于 3 或者小于等于-3，那么就需要将这行数据删除。

相比于删除异常数据，我们的方法是选择其中正常的数据。如果我们规定一个正常数据的 9 个属性都必须同时小于等于 3 并且大于-3，那么就意味着该行数据的 9 个属性中的最大值必然小于等于 3，最小值必然大于-3，而用这种方法我们可以取出符条件的数据的索引。我们可以获得 9 个属性中的最大值小于等于 3 的数据的索引，再取出 9 个属性中的最小值大于-3 的数据的索引，对两份索引进行按位与，即可得到正常数据的索引。该方法的实现如图 6.

```
# 数据标准化，计算Z分数
Z = (data_box - data_box.mean()) / data_box.std()
print(Z)
```
Executed at 2024.03.19 19:37:13 in 15ms

```
          ...          ...       ...
2   -0.179534    -0.611387 -0.343666
3   -0.179534     1.353485 -0.343666
4   -0.179534    -0.611387 -0.343666
..        ...          ...       ...
694 -0.999756    -0.611387 -0.343666
695 -0.999756    -0.611387 -0.343666
696  1.871021     2.335921  0.239398
697  2.691243     1.026006 -0.343666
698  2.691243     0.371049 -0.343666
```

图 5 计算属性的 Z 分数

```
# 按照"Z > 3 or Z <= -3"这个原则删除异常值，打印原始数据量和删除异常值后的数据量
data_medium_out = data_nan.fillna(data_nan.median(), inplace=False)
criteria = (Z.max(axis = 1) <= 3) & (Z.min(axis = 1) > -3)
data_medium_out = data_medium_out[criteria]

print(f"原始数据量: {data.shape[0]}")
print(f"删除异常后数据量: {data_medium_out.shape[0]}")
```
Executed at 2024.03.20 23:07:37 in 18ms

```
原始数据量: 699
删除异常后数据量: 648
```

图 6 删除异常值

数据质量不佳也有可能是因为其中存在重复数据。在这里，我们采用 data.duplicated()以获得重复的数据行，最后，我们利用 data.drop_duplicates()删除重复的行。该流程如图 7 所示。

```
# 检查数据中的重复样本，打印重复样本个数
duplicated_rows = data.duplicated()
print(f"重复样本个数为: {data[duplicated_rows].shape[0]}")
```
Executed at 2024.03.19 20:01:34 in 15ms

```
重复样本个数为: 8
```

```
# 删除重复行，打印删除前后数据集样本量
data_no_duplicates = data.drop_duplicates()
print(f"删除前的数据集样本量: {data.shape[0]}")
print(f"删除后的数据集样本量: {data_no_duplicates.shape[0]}")
```
Executed at 2024.03.19 20:02:17 in 31ms

图 7 删除重复数据

4.2 数据聚合

如果数据的量过大，直接地分析数据可能是一件费力且效果不佳的事。为此，我们需要进行数据聚合，其主要目的是减少要处理的数据的大小，改变分析的粒度，并提高数据的稳定性。本次实验中我们使用 DTW_prec.csv 中的降水量数据。

我们首先使用 matplotlib 模块绘制其每日时间序列的折线图，并打印数据的方差。效果如图 8.

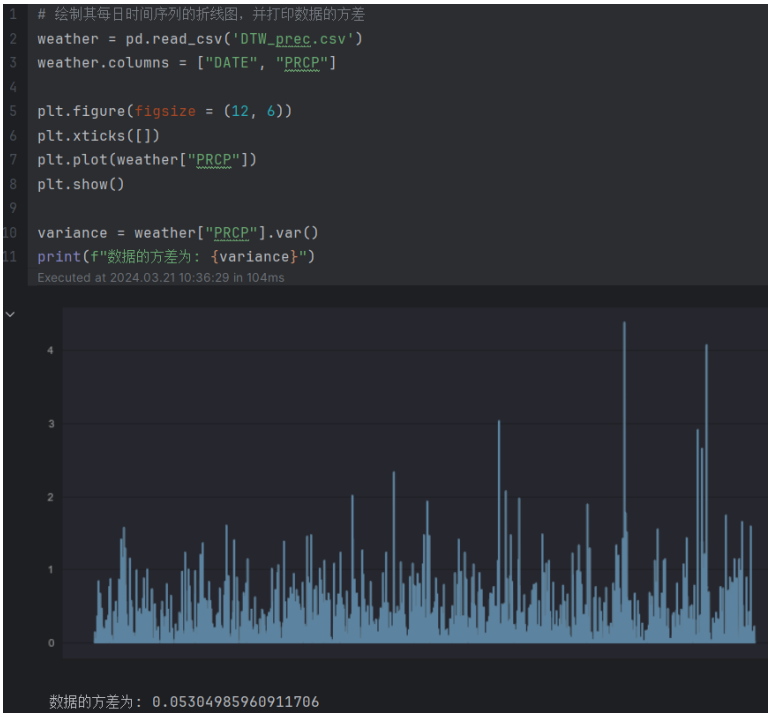

图 8 绘制原始数据图像

但这里的数据包括了每一天的数据，数据量非常大，有时候在分析气候时我们不需要细化到每一天的程度，因此我们可以进行数据聚合。我们通过将日期转换为标准格式，对日期进行切片后使用 groupby 方法，可以得到聚合后的数据。图 9 展示了每月时间序列的折线图，图 10 展示了每年时间序列的折线图。

图 9 按月份聚合的图像

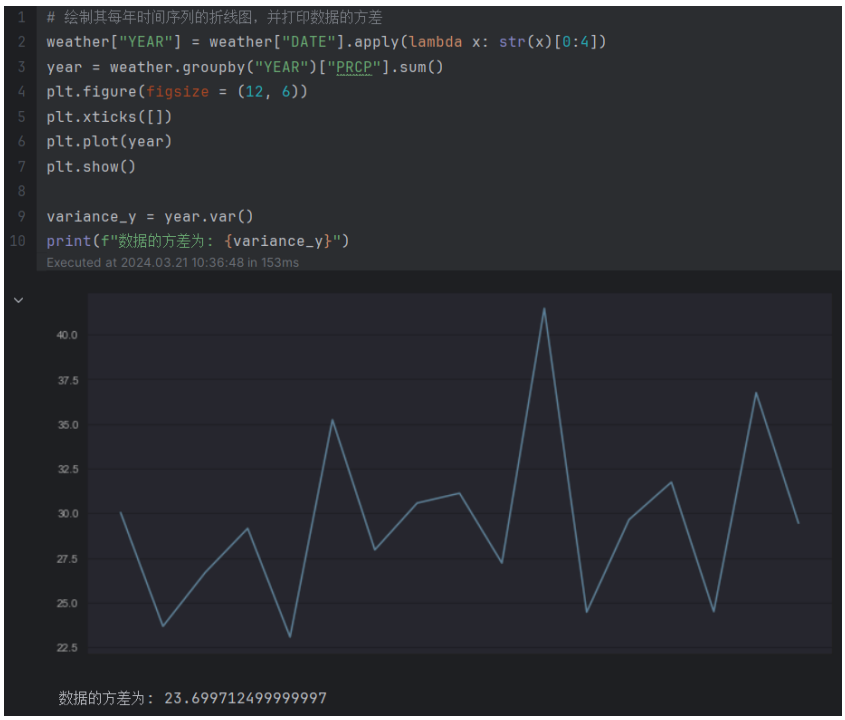

图 10 按年份聚合的图像

4.3 采样

数据过多会影响我们分析数据的效率，有时候可能只需要提取部分就足以掌握数据特征，这样的提取操作即为采样。在这里，我们将使用 breast-cancer-wisconsin.data 来实现采样。

我们使用了 data.sample(n=3)来获取大小为 3 的样本，使用 data.sample(frac = 0.01)来获取大小为整体 1%的样本。以上的采样均为不替换采样，即选定的实例会从数据集中删除，以确保每次得到的都是不一样的数据，如图 11 所示。而如果我们希望使用替换采样，即选定的实例不会从数据集中删除，我们只需要添加参数 replace=True 即可，如图 12 所示。

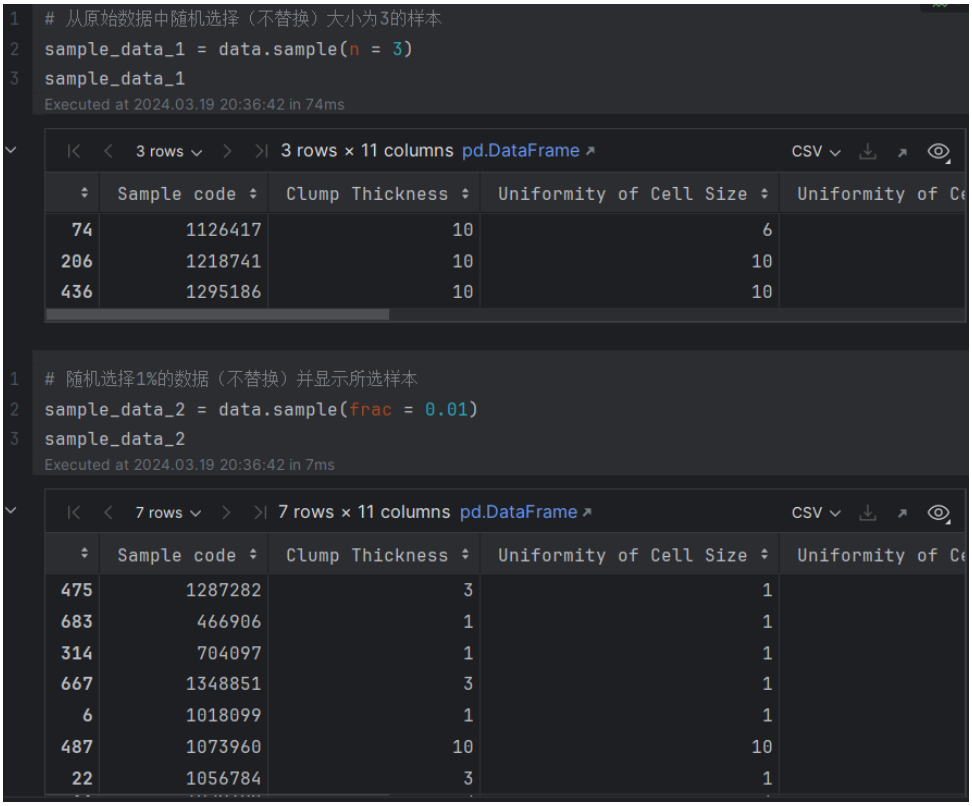

图 11 不替换采样

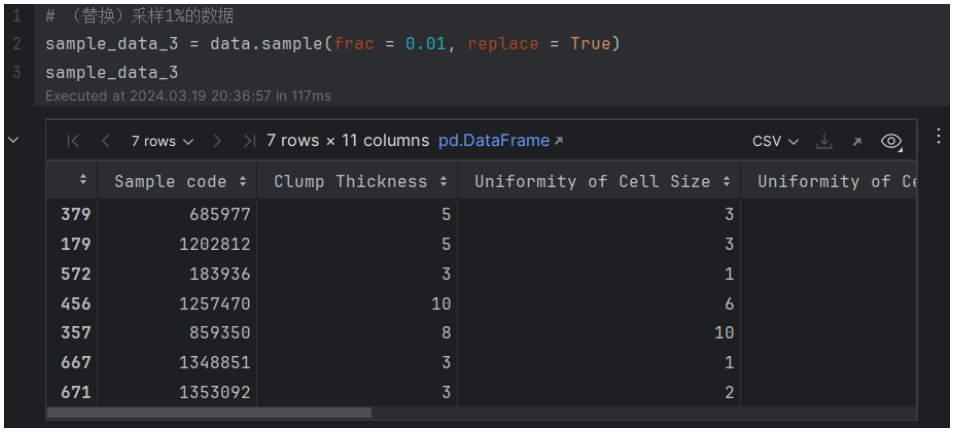

图 12 替换采样

4.4 离散化

有时候，我们希望可以将连续属性转换为分类属性。在这里，我们使用 breast-cancer-wisconsin.data 来完成离散化的实验。

我们提取数据集中的 Clump Thickness 列，并且对其中的每个值使用 value_counts()进行计数，再利用 plt.bar 来绘制直方图观看数据分布。具体操作如图 13 所示。

```
# 使用Counter对数据集Clump Thickness属性的取值进行计数
count = data["Clump Thickness"].value_counts()
count
```

```
# 绘制直方图，显示属性值的分布
plt.figure(figsize = (10, 6))
plt.bar(count.index, height = count)
plt.show()
```

Executed at 2024.03.21 10:51:38 in 191ms

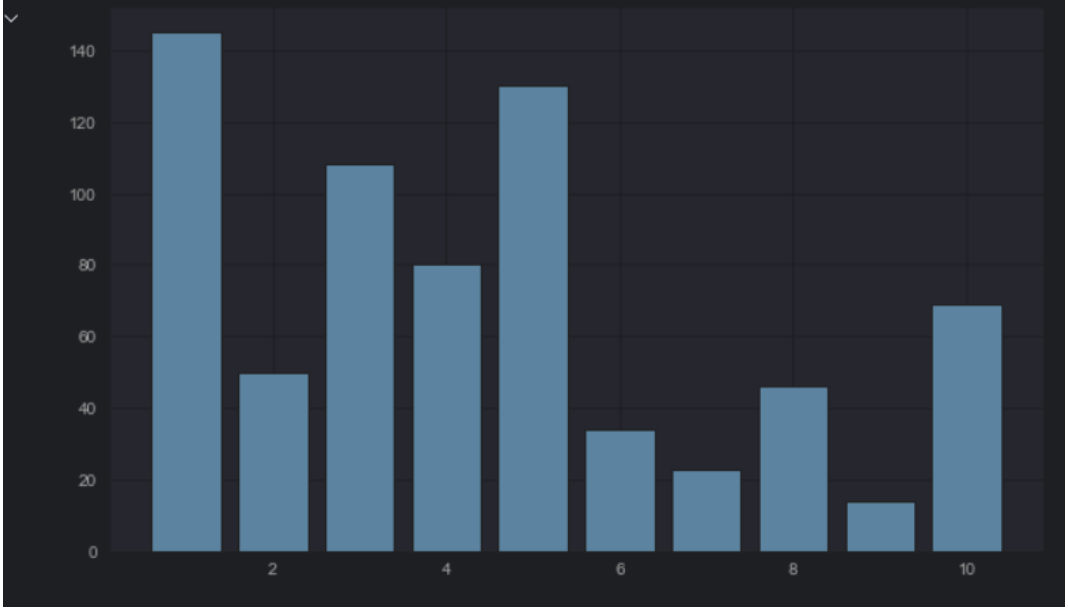

图 13 观察数据分布

接下来，我们使用两种离散化方法。一种是 pd.cut()，它可以将属性离散为多个间隔宽度相似的 bin，另一种是 pd.qcut()，可以将值划分为多个 bin，每个 bin 具有几乎相同数量的实例。效果如图 14 所示。

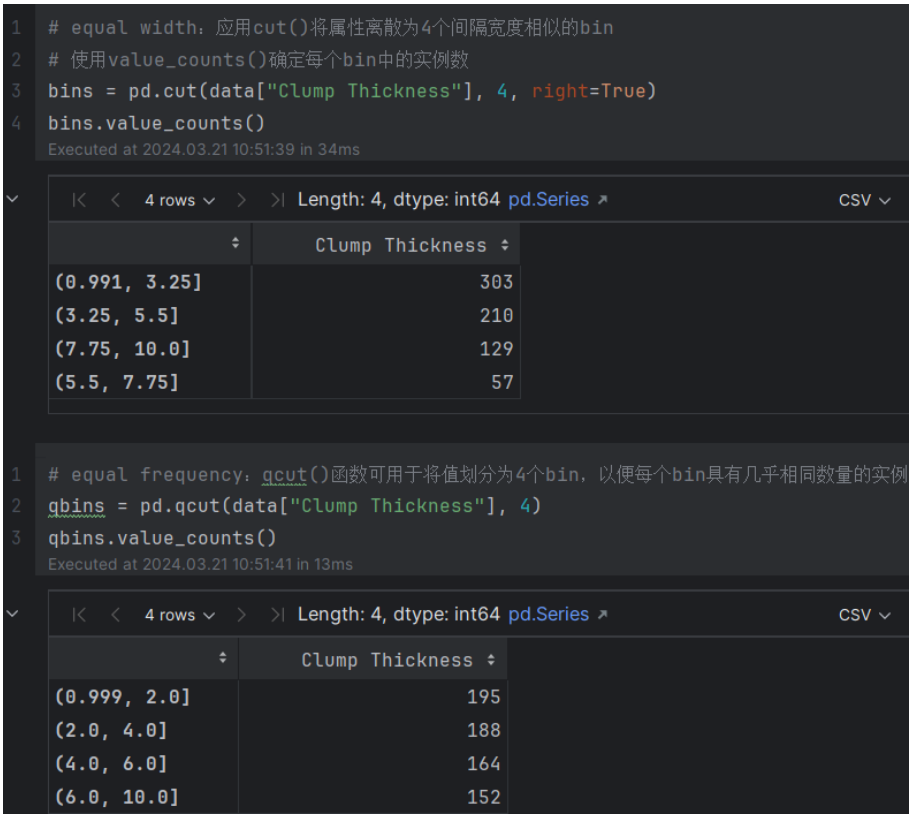

图 14 数据离散化

注意到，我们在使用 pd.cut()的时候，最小的区间左侧数值为 0.991，这是一个比较异常的现象。通过阅读源码 def cut，我们发现这主要是因为，最后显示给我们的区间必然是左开右闭或左闭右开区间，这意味着我们需要对其中的最小值或最大值的边界进行调整，从而确保所有区间可以容纳所有数据。在源码中，我们发现当列表中不存在无穷大的数值，列表的最大值与最小值不相等的时候，我们会执行正常的离散化操作，而为了可以让端点处被取到，我们会计算最大值与最小值的差的 0.001 倍作为调整量，如果我们计算的是左开右闭区间，那么我们就调整最左侧的端点，如果我们计算的是左闭右开区间，那么就调整最右侧的端点。在我们使用的数据中，最小值为 1，最大值为 10，因此调整量即为 0.009，故最左侧的端点为 1-0.009=0.991，这就解释了为什么最左侧的数值为 0.991.

如果希望最左侧的数值好看一些，比如变为 0.99，可以再添加一个参数 include_lowest=True，这可以让区间包括列表的最左边界。效果如图 15.

图 15 离散化的优化

4.5 主成分分析

主成分分析是一种通过将数据从其原始高维空间投影到低维空间来减少数据中属性数量的经典方法。在这里，我们将使用 16 张图片进行主成分分析。

我们首先将 16 张图片拼接在一起，得到一个 16*36963 规模的矩阵，代码如图 16 所示。

```
# 读取图像数据，将RGB图像转换为111x111x3=36963个特征值，最终得到一个16x36963的矩阵
from PIL import Image

matrix = np.zeros((1, 111*111*3))
for i in range(1, 17):
    img = Image.open("./pics/Picture" + str(i) + ".jpg")
    img = np.array(img).reshape(1, -1)
    matrix = np.r_[matrix, img]

matrix = matrix[1:]
```

图 16 拼接图片

接下来我们调用 sklearn.decomposition 中的 PCA 模块，对矩阵进行主成分的提取。在这里，我们只提取 2 个主成分，这让我们得到了一个 16*2 的矩阵。在得到该矩阵后，我们希望为每一个数据设置上标签，因此我们为 16 张图片分别标上了“汉堡”、“可乐”、“面条”与“鸡肉”的标签。最终得到的数据如图 17 所示。

```
# 无需编写PCA代码，直接导入sklearn.decomposition中的PCA类
from sklearn.decomposition import PCA
```
Executed at 2024.03.21 11:33:34 in 1s 357ms

```
pca = PCA(n_components = 2)
transformed_data = pca.fit_transform(matrix)
```
Executed at 2024.03.21 11:33:35 in 576ms

```
# 绘制散点图来显示投影值
df = pd.DataFrame(
    {'x1': transformed_data[:,0], 'x2': transformed_data[:,1], 'type': ['burger'] * 4 +
      ['cola'] * 4 + ['noodles'] * 4 + ['chicken'] * 4}
)
df
```
Executed at 2024.03.21 11:35:00 in 45ms

1-10 | 16 rows × 3 columns pd.DataFrame | CSV

	x1	x2	type
0	-1576.750805	6640.834024	burger
1	-493.781795	6397.442678	burger
2	990.074263	7236.018839	burger
3	2189.901032	9051.027367	burger
4	-7843.038752	-1061.248316	cola
5	-8498.425704	-5438.390444	cola
6	-11181.798256	-5320.070534	cola
7	-6851.932192	1124.734178	cola

图 17 图片拼接处理后的数据

最后，我们对得到的数据进行散点图的绘制。利用 seaborn 模块，我们可以绘制出如图 18 的散点图，可以观察到大致上 4 类图像的特征被区分开来。

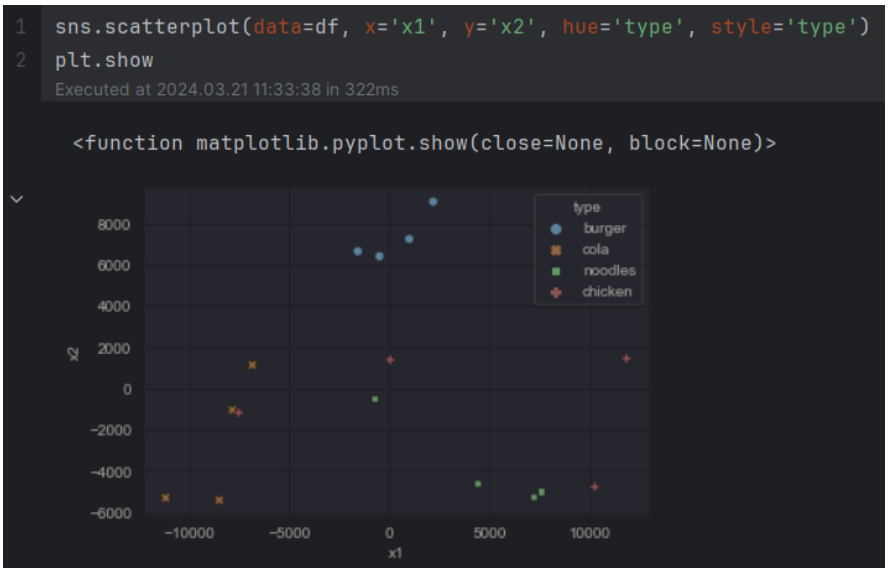

图 18 绘制主成分分析后的散点图

五、总结与分析

本次实验我们体验了部分数据预处理相关的操作。数据预处理是在进行数据分析与数据挖掘时非常重要的一步，如果数据没有进行过任何处理，最终可能导致挖掘出的特征不准确。因此，数据预处理是每次在进行数据分析与数据挖掘时最重要的前置步骤，是不可省略的。然而，Python 进行数据预处理需要借助非常多的模块的帮助，这样的操作对我来说较为陌生，我也需要再进一步地提升对 Python 数据预处理操作的熟练程度。