

金融数据挖掘课程设计 作业4 聚类分析

本作业对银行零售客户做聚类分析，使用 K-means 和凝聚层次聚类两种方法，把客户按价值、风险和行为画像分群，并对结果做评价和业务解释。

数据集：financial_customer_clustering_homework.csv，共 1600 条客户记录。

3.1 数据读取与预处理

读取数据后先看样本量、字段数量和前 5 行，再查看字段类型和描述性统计，然后处理缺失值和异常值。

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import linkage, dendrogram

# 让图里能正常显示中文
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

```
In [2]: df = pd.read_csv('financial_customer_clustering_homework.csv')
print('样本量: ', df.shape[0])
print('字段数量: ', df.shape[1])
df.head()
```

样本量: 1600

字段数量: 17

```
Out[2]:
```

	customer_id	age	annual_income	total_financial_assets	loan_balance	credit_score
0	HC00001	28	100882.44	190722.03	43282.54	645.6
1	HC00002	42	256947.83	227393.60	618189.98	735.6
2	HC00003	30	67293.44	176840.74	23752.75	672.1
3	HC00004	38	165390.43	161926.11	329681.84	691.2
4	HC00005	59	278419.82	215147.04	291219.21	651.3

```
In [3]: # 字段类型
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600 entries, 0 to 1599
Data columns (total 17 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   customer_id                          1600 non-null   object
1   age                                   1600 non-null   int64
2   annual_income                        1560 non-null   float64
3   total_financial_assets               1571 non-null   float64
4   loan_balance                         1600 non-null   float64
5   credit_score                         1568 non-null   float64
6   credit_card_utilization              1600 non-null   float64
7   overdue_times_12m                   1600 non-null   int64
8   monthly_transaction_count            1600 non-null   int64
9   online_banking_ratio                 1600 non-null   float64
10  wealth_product_ratio                 1600 non-null   float64
11  debt_to_income                       1600 non-null   float64
12  investment_asset_ratio                1600 non-null   float64
13  digital_activity_score                1600 non-null   float64
14  risk_level                           1600 non-null   object
15  default_flag                         1600 non-null   int64
16  true_segment                         1600 non-null   object
dtypes: float64(10), int64(4), object(3)
memory usage: 212.6+ KB

```

```

In [4]: # 描述性统计
df.describe()

```

```

Out[4]:
```

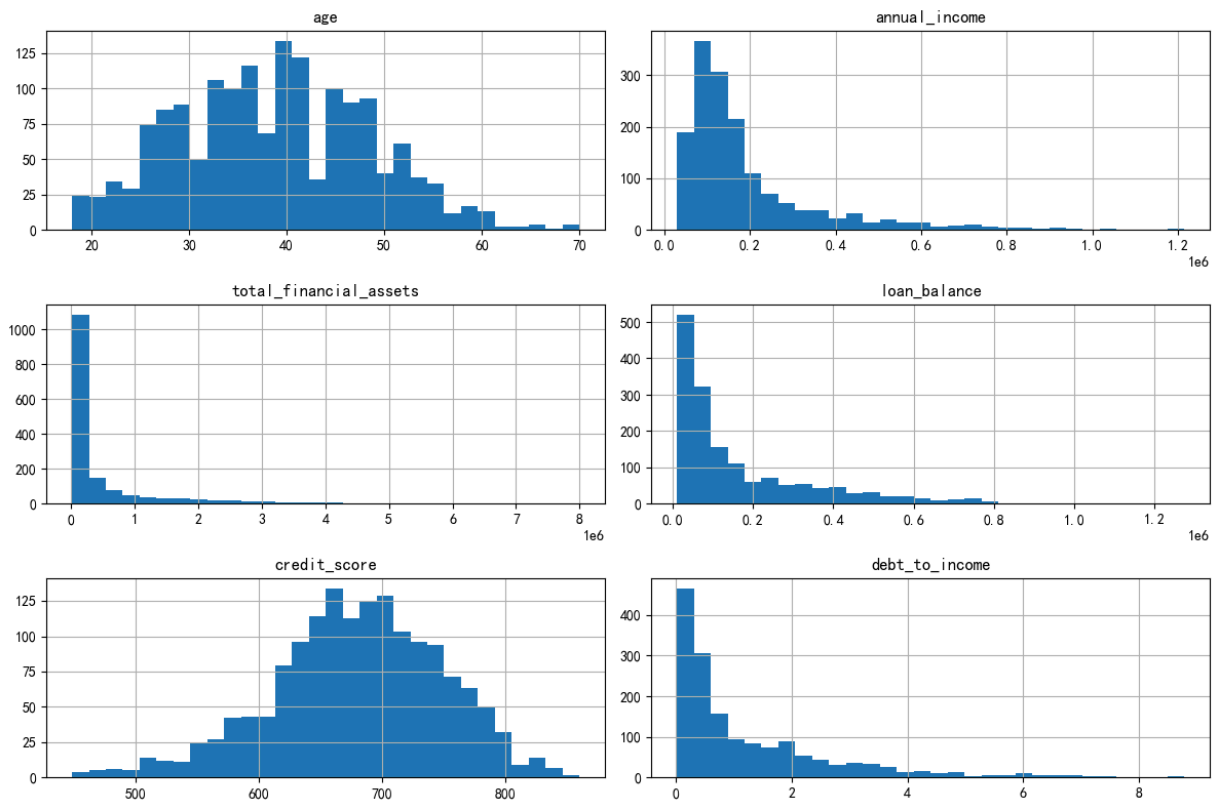
	age	annual_income	total_financial_assets	loan_balance	credit_score
count	1600.000000	1.560000e+03	1.571000e+03	1.600000e+03	1568.000000
mean	38.627500	1.918873e+05	4.822498e+05	1.696508e+05	679.875255
std	9.756895	1.650040e+05	8.942846e+05	1.864282e+05	69.963799
min	18.000000	2.926063e+04	1.017037e+04	1.041521e+04	448.200000
25%	31.000000	9.182098e+04	6.151127e+04	4.351868e+04	637.375000
50%	39.000000	1.354658e+05	1.249577e+05	8.731685e+04	682.950000
75%	46.000000	2.200696e+05	3.982867e+05	2.414378e+05	730.275000
max	70.000000	1.215178e+06	8.000000e+06	1.274051e+06	860.400000

下面画几个主要变量的分布直方图，大致了解数据的形态。

```

In [5]: cols = ['age', 'annual_income', 'total_financial_assets', 'loan_balance',
               'credit_score', 'debt_to_income']
df[cols].hist(figsize=(12, 8), bins=30)
plt.tight_layout()
plt.show()

```



缺失值处理

统计每个字段的缺失值数量，数值型变量用中位数填补。

```
In [6]: # 查看缺失值
df.isnull().sum()
```

```
Out[6]: customer_id      0
age                    0
annual_income          40
total_financial_assets 29
loan_balance           0
credit_score           32
credit_card_utilization 0
overdue_times_12m      0
monthly_transaction_count 0
online_banking_ratio    0
wealth_product_ratio     0
debt_to_income          0
investment_asset_ratio   0
digital_activity_score   0
risk_level              0
default_flag            0
true_segment            0
dtype: int64
```

```
In [7]: # 用中位数填补缺失值
for col in ['annual_income', 'total_financial_assets', 'credit_score']:
    df[col] = df[col].fillna(df[col].median())
```

```
print('填补后剩余缺失值: ', df.isnull().sum().sum())
```

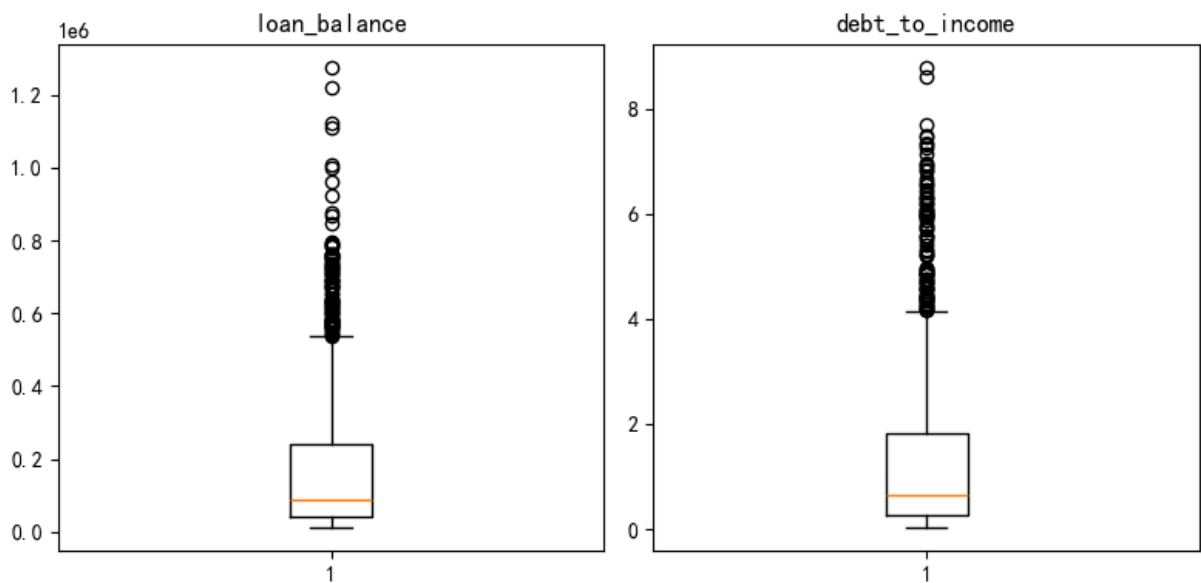
填补后剩余缺失值: 0

异常值处理

loan_balance 和 debt_to_income 这类变量可能有个别特别大的值，先用箱线图看一下，再用 1% 和 99% 分位数做截断，把超过上下界的值拉回到边界。

```
In [8]: plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.boxplot(df['loan_balance'])
plt.title('loan_balance')
plt.subplot(1, 2, 2)
plt.boxplot(df['debt_to_income'])
plt.title('debt_to_income')
plt.tight_layout()
plt.show()

# 用 1% 和 99% 分位数截断异常值
for col in ['loan_balance', 'debt_to_income']:
    low = df[col].quantile(0.01)
    high = df[col].quantile(0.99)
    df[col] = df[col].clip(low, high)
```



选择聚类变量

customer_id 是编号，不能用来聚类。risk_level、default_flag、true_segment 是结果标签，只在后面做结果解释，不放进聚类输入。其余 13 个数值变量都作为客户画像变量，满足不少于 8 个变量的要求。聚类前对这些变量做标准化。

```
In [9]: features = ['age', 'annual_income', 'total_financial_assets', 'loan_balance',
                    'credit_score', 'credit_card_utilization', 'overdue_times_12m',
                    'monthly_transaction_count', 'online_banking_ratio',
```

```

        'wealth_product_ratio', 'debt_to_income',
        'investment_asset_ratio', 'digital_activity_score']

X = df[features]

# 标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print('用于聚类的变量个数: ', len(features))

```

用于聚类的变量个数: 13

3.2 K-means 聚类

先用手肘法和轮廓系数确定合适的 K 值，再做聚类并可视化。

```

In [10]: # 计算 k=2 到 10 的 SSE 和轮廓系数
sse = []
sil = []
k_range = range(2, 11)
for k in k_range:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = km.fit_predict(X_scaled)
    sse.append(km.inertia_)
    sil.append(silhouette_score(X_scaled, labels))

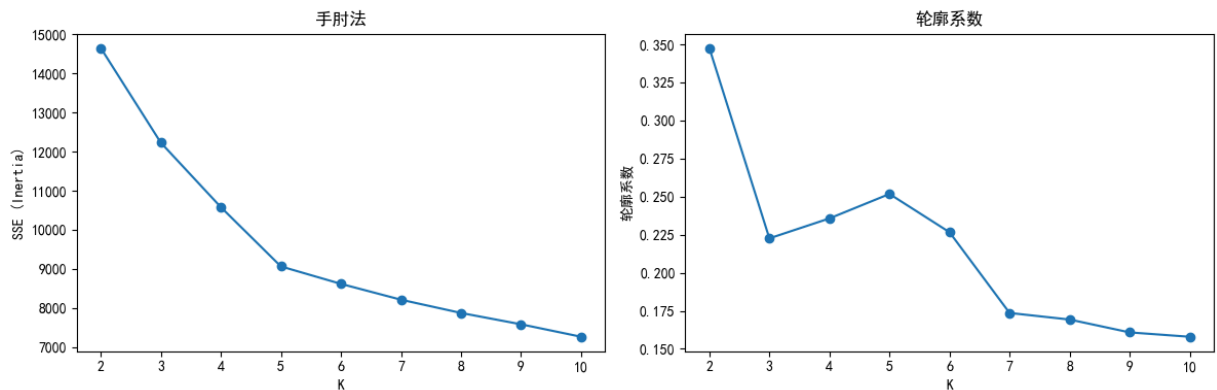
```

```

In [11]: plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(list(k_range), sse, marker='o')
plt.xlabel('K')
plt.ylabel('SSE (Inertia)')
plt.title('手肘法')

plt.subplot(1, 2, 2)
plt.plot(list(k_range), sil, marker='o')
plt.xlabel('K')
plt.ylabel('轮廓系数')
plt.title('轮廓系数')
plt.tight_layout()
plt.show()

```



从肘部折线图看，SSE 在 K=4、5 附近下降明显变缓，肘部大致在这个位置；轮廓系数在这一段也比较高。结合银行客户大致可以分成几类的业务情况，这里选 K=5。

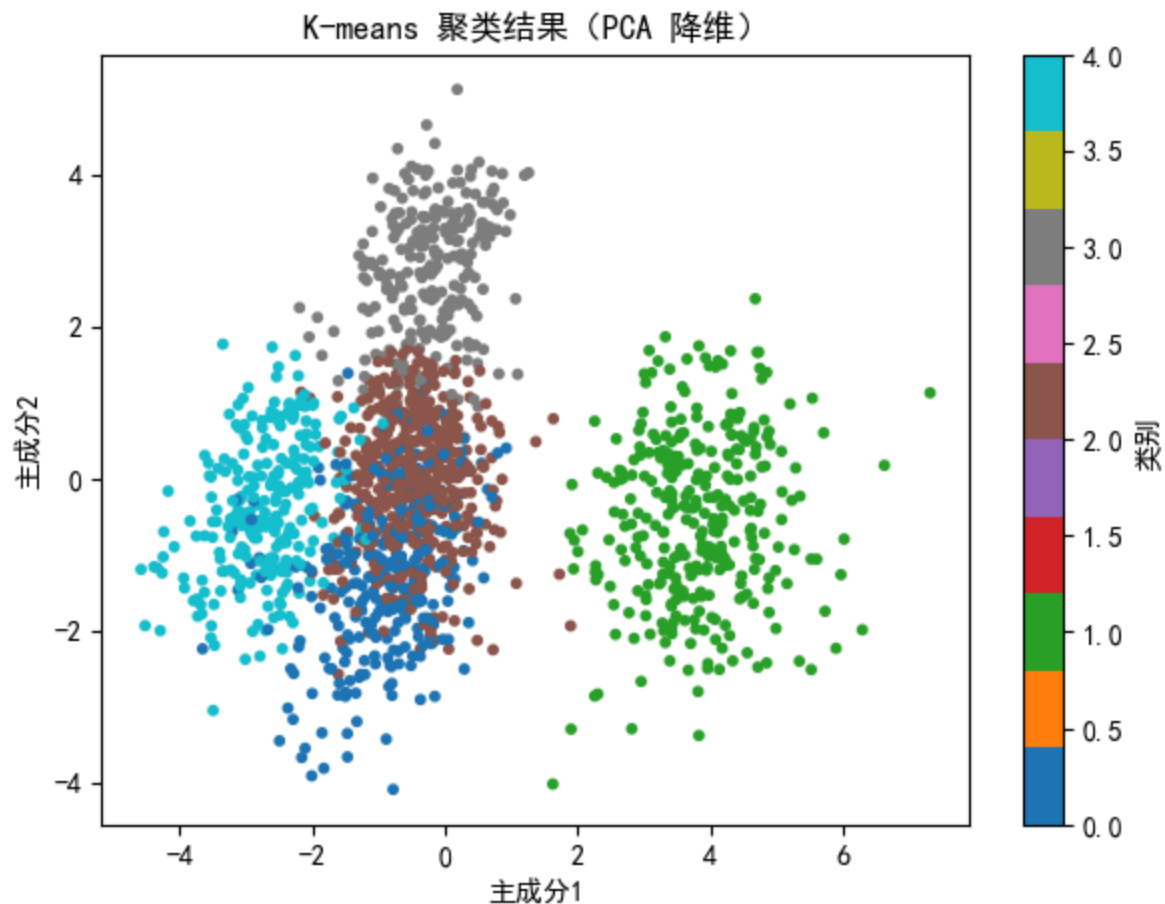
```
In [12]: # 用选定的 K 做最终聚类
k_best = 5
kmeans = KMeans(n_clusters=k_best, random_state=42, n_init=10)
df['kmeans_label'] = kmeans.fit_predict(X_scaled)
df['kmeans_label'].value_counts().sort_index()
```

```
Out[12]: kmeans_label
0      294
1      322
2      497
3      237
4      250
Name: count, dtype: int64
```

用 PCA 把 13 维变量降到 2 维，把聚类结果画成散点图。

```
In [13]: pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(7, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['kmeans_label'], cmap='tab10', s=10)
plt.xlabel('主成分1')
plt.ylabel('主成分2')
plt.title('K-means 聚类结果 (PCA 降维)')
plt.colorbar(label='类别')
plt.show()
```



输出每一类的样本数量和各变量均值，用来观察不同客户群的特征。

```
In [14]: print('每类样本数量: ')
print(df['kmeans_label'].value_counts().sort_index())
print()

# 每一类各变量的均值
df.groupby('kmeans_label')[features].mean()
```

```
每类样本数量:
kmeans_label
0      294
1      322
2      497
3      237
4      250
Name: count, dtype: int64
```

Out[14]:

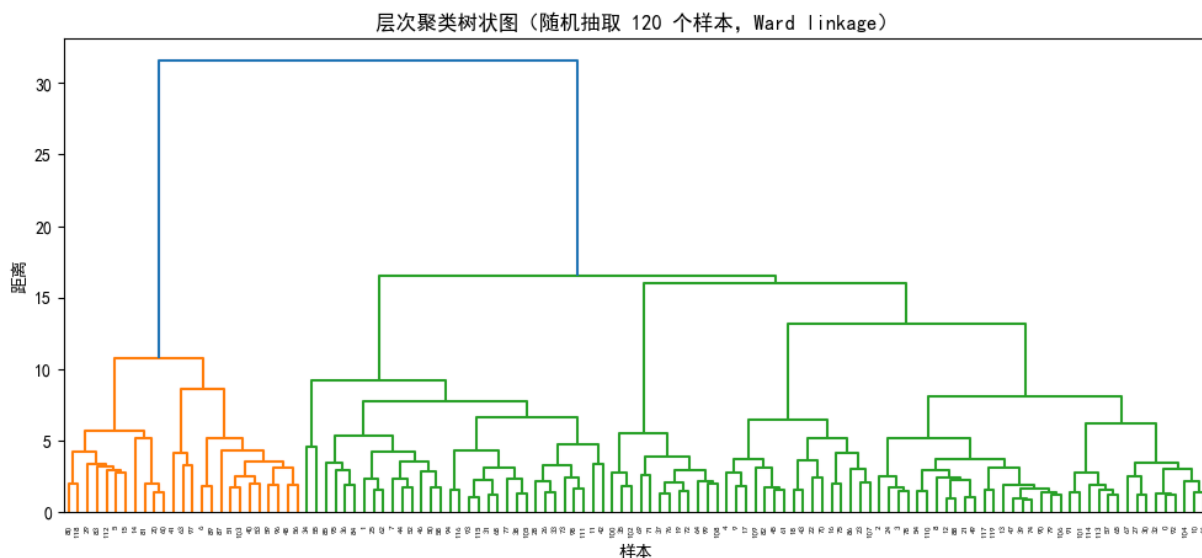
	age	annual_income	total_financial_assets	loan_balance	credit_score
kmeans_label					
0	42.523810	168094.554728	2.861379e+05	480132.417516	686.09
1	46.214286	439620.513882	1.792256e+06	86241.636331	760.88
2	38.434608	129773.599356	1.346705e+05	81646.418576	680.56
3	28.177215	100157.106878	7.140344e+04	58669.441807	659.18
4	34.564000	102202.179680	6.461408e+04	181747.910060	586.84

3.3 凝聚层次聚类

先随机抽取一部分客户画树状图（全部样本画出来太密看不清），再用 Ward linkage 对全部样本做层次聚类，聚类数和 K-means 保持一致取 5。

```
In [15]: # 随机抽取 120 个样本画树状图
np.random.seed(42)
sample_idx = np.random.choice(len(X_scaled), 120, replace=False)
sample_data = X_scaled[sample_idx]

Z = linkage(sample_data, method='ward')
plt.figure(figsize=(12, 5))
dendrogram(Z)
plt.title('层次聚类树状图（随机抽取 120 个样本，Ward linkage）')
plt.xlabel('样本')
plt.ylabel('距离')
plt.show()
```

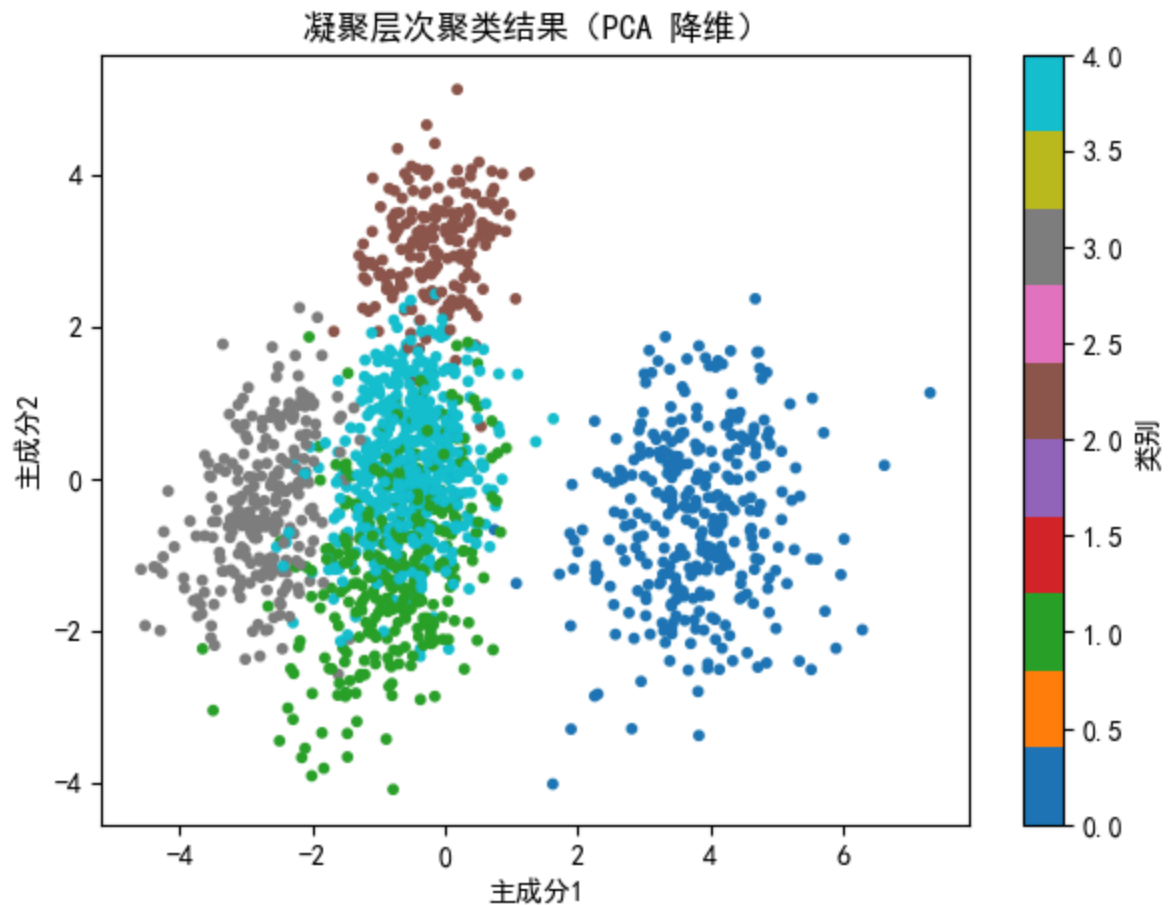


```
In [16]: # 对全部样本做凝聚层次聚类
agg = AgglomerativeClustering(n_clusters=5, linkage='ward')
df['hier_label'] = agg.fit_predict(X_scaled)
df['hier_label'].value_counts().sort_index()
```



```
Out[16]: hier_label
0      326
1      327
2      187
3      249
4      511
Name: count, dtype: int64
```

```
In [17]: plt.figure(figsize=(7, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df['hier_label'], cmap='tab10', s=10)
plt.xlabel('主成分1')
plt.ylabel('主成分2')
plt.title('凝聚层次聚类结果 (PCA 降维)')
plt.colorbar(label='类别')
plt.show()
```



比较两种聚类结果

用交叉表看 K-means 和层次聚类标签的对应关系。两种方法的类别编号不一定相同，主要看同一群客户是不是大致被分到一起。

```
In [18]: pd.crosstab(df['kmeans_label'], df['hier_label'])
```

```
Out[18]:
```

hier_label	0	1	2	3	4
kmeans_label					
0	0	268	0	11	15
1	322	0	0	0	0
2	4	43	5	6	439
3	0	11	182	3	41
4	0	5	0	229	16

3.4 无监督评价

用轮廓系数分别评价两种聚类的效果，取值范围 -1 到 1，越接近 1 说明分群越清晰。

```
In [19]: km_score = silhouette_score(X_scaled, df['kmeans_label'])
hier_score = silhouette_score(X_scaled, df['hier_label'])
print('K-means 轮廓系数:', round(km_score, 4))
print('层次聚类轮廓系数:', round(hier_score, 4))
```

K-means 轮廓系数: 0.2517
层次聚类轮廓系数: 0.2358

结果解释

数据里带有 true_segment 真实客户分群标签，这里用它和 K-means 结果做交叉表，看看聚类结果是否合理（true_segment 只用于解释，没有参与聚类）。

```
In [20]: pd.crosstab(df['kmeans_label'], df['true_segment'])
```

```
Out[20]:
```

true_segment	Affluent_Wealth_Customers	High_Risk_Credit_Customers	Mass_Salary_Customers
kmeans_label			
0	0	14	
1	322	0	
2	3	17	
3	0	6	
4	0	244	

小结

把客户的价值、风险和行为变量标准化后，用 K-means 和凝聚层次聚类都分成了 5 群。肘法和轮廓系数都支持 K=5 的选择，PCA 散点图上不同类别大致能分开。两种方法的交叉表显示大部分客户被分到对应的群里，结果比较一致。

从各类的变量均值可以看出不同客户群在收入、资产、贷款余额和数字活跃度上的差异，大致对应大众工资客户、按揭家庭客户、富裕财富客户、高风险信用客户和年轻数字客户。银行可以据此做差异化营销、财富管理推荐和信用风险识别。