案例二：基于K-Means根据信用卡消费情况推荐银行理财产品

1. 案例目的

• 理解聚类问题的场景；

• 掌握解决聚类问题的整套流程；

• 掌握数据预处理和聚类模型的构建。

2. 案例内容

根据信用卡消费情况推荐银行理财产品案例将对8950条持卡人信用卡数据的18个信用卡相关的属性进行分析，建立聚类模型，对这8950个持卡人进行客户分组，然后把银行的理财产品推荐给优秀的分组。

3. 案例知识点

• Python语言编程；

• 数据处理；

• 聚类模型；

• 分析聚类结果。

4. 案例时长

共4学时，具体安排如下：

• 数据预处理（2学时）

• 构建聚类模型（1学时）

• 分析每一个分组（1学时）

5. 案例实验环境

**•操作系统：**

1）Linux Ubuntu 16.04 64位操作系统

**•软件环境：**

1）Python 3.7

2）Sklearn 0.23

**•开发环境与工具：**

1）Spyder 4

6. 案例分析

本案例主要分为以下3部分：

1）处理数据，为建立聚类模型做准备；

2）构建聚类模型，并使用轮廓系数的方法选择K-Means中K的个数；

3）探索K-Means得到的每一个客户分组，从每一个客户分组中得到线索，从而找到目标客户。

7. 案例实验过程

根据信用卡消费情况推荐银行理财产品，可分为以下3个步骤：

1、数据预处理；

1.1、导入数据预处理的包；

1.2、导入数据集；

1.3、了解数据集；

1.4、分析字段相关性；

1.5、处理缺失数据；

1.6、处理类别型字段；

1.7、删除无用的变量；

1.8、特征提取；

1.9、特征缩放。

2、构建聚类模型；

2.1、聚类个数判断；

2.2、构建聚类模型；

2.3、可视化聚类效果；

3、分析每一个分组；

7.1开发准备

在桌面上新建一个目录，名字叫finance。本案例中所有的数据集和代码都放在该目录下。

7.1.1 打开Spyder软件

在桌面上点击Spyder图标打开Spyder，并把工作目录切换至finance目录下。

7.1.2 获取数据

本案例需要1个数据集，是credit\_card.csv。

点击右侧菜单栏中的【资源下载】按钮，复制提供的URL前缀，如https://xxx.eec-cn.com/。

将URL前缀和数据集地址拼接到一起，如https://xxx.eec-cn.com/dataSet/systemLib/58e4e723888d4cf48b6a8e4711499821.csv下载数据集。将下载后的数据集credit\_card.csv复制到finance目录下。

7.1.3 新建代码文件

确保Spyder的工作目录为finance。使用Spyder新建一个Python文件，命名为financial\_management\_recommendation.py。

至此，整个案例的目录结构如图1所示。

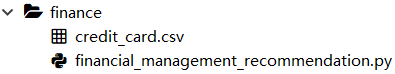


图1 案例的目录结构

7.1.4 了解数据

credit.csv存储着8950条客户信用卡数据，总共有18个字段。字段含义如下：

• CUST\_ID：信用卡持有人的身份证明

• BALANCE：信用卡可用余额

• BALANCE\_FREQUENCY：信用卡余额变动频率，得分在0到1之间（1 =频繁更新，0 =不频繁更新）

• PURCHASES：信用卡消费金额

• ONEOFF\_PURCHASES：信用卡单次最大刷卡金额

• INSTALLMENTS\_PURCHASES：信用卡分期金额

• CASH\_ADVANCE：信用卡提现金额

• PURCHASES\_FREQUENCY：信用卡刷卡频率，得分在0到1之间（1 =频繁刷卡，0 =不频繁刷卡）

• ONEOFF\_PURCHASES\_FREQUENCY：一次购买的频率（1 =频繁购买，0 =不频繁购买）

• PURCHASES\_INSTALLMENTS\_FREQUENCY：信用卡分期频率（1 =频繁分期，0 =不频繁分期）

• CASH\_ADVANCE\_FREQUENCY：信用卡提现频率

• CASH\_ADVANCE\_TRX：使用“信用卡提现”进行的交易数量

• PURCHASES\_TRX：进行的购买交易数

• CREDIT\_LIMIT：信用卡额度

• PAYMENTS：信用卡还款金额

• MINIMUM\_PAYMENTS：信用卡最低还款金额

• PRC\_FULL\_PAYMENT：信用卡全额还款次数比例

• TENURE：用户的信用卡服务的使用权

7.2数据预处理

该模块作用是加载数据源并进行必要的数据预处理。

向financial\_management\_recommendation.py中添加代码如下：

7.2.1 导入数据预处理的包

【代码7-2-1】financial\_management\_recommendation.py

#%%

# 导入包

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#%%

执行代码块。

7.2.2 导入数据集

【代码7-2-2】financial\_management\_recommendation.py

#%%

# 读取数据集

dataset = pd.read\_csv('credit\_card.csv')

#%%

执行代码块。

在Variable explorer面板中双击dataset这个变量查看它的值。dataset变量的值如图2所示。

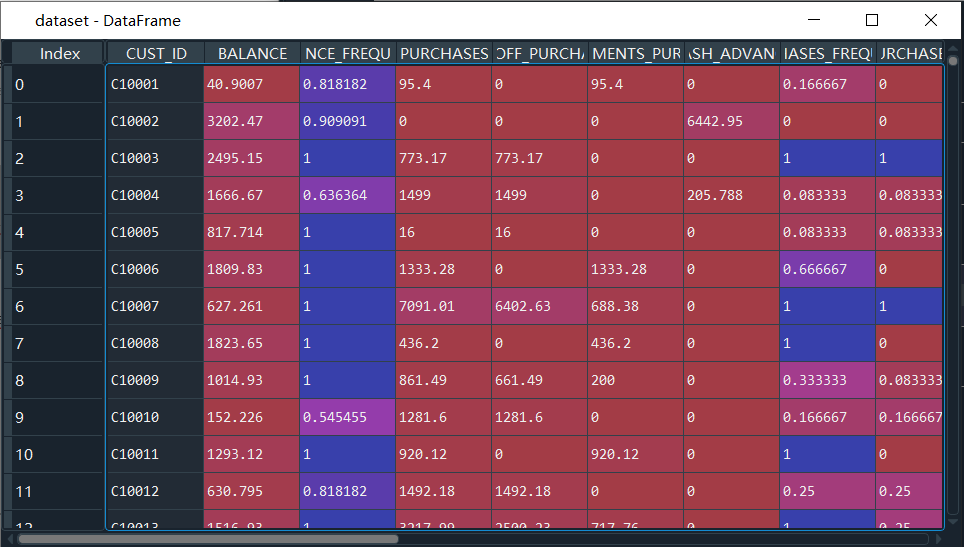


图2 dataset变量的值

18个字段中只有第一个字段CUST\_ID是字符型，其它17个都是数值型。

7.2.3 了解数据集

【代码7-2-3】financial\_management\_recommendation.py

#%%

# 查看数据集情况

dataset\_description\_df = dataset.describe()

#%%

执行代码块。

在Variable Explorer面板中双击dataset\_description\_df变量查看它的值如图3所示。

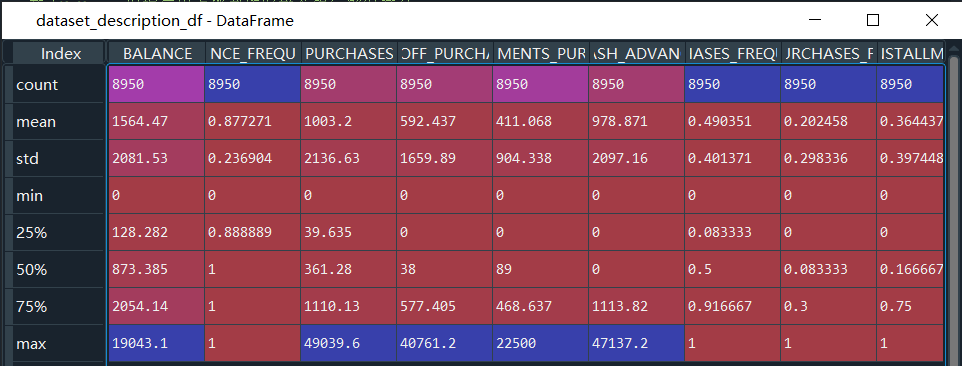


图3 dataset\_description\_df变量的值

7.2.4 分析字段相关性

通常使用热力图画出所有数值型字段的相关性。下面代码将画出17个数值型字段的热力图。

【代码7-2-4-1】financial\_management\_recommendation.py

#%%

# 查看相关性-热力图

import seaborn as sns

corrmat = dataset.corr()

plt.figure(figsize=(16,16))

plt.xticks(rotation=90)

plt.yticks(rotation=90)

sns.heatmap(corrmat, vmax=1.0, square=True)

#%%

执行代码块。

热力图如图4所示。

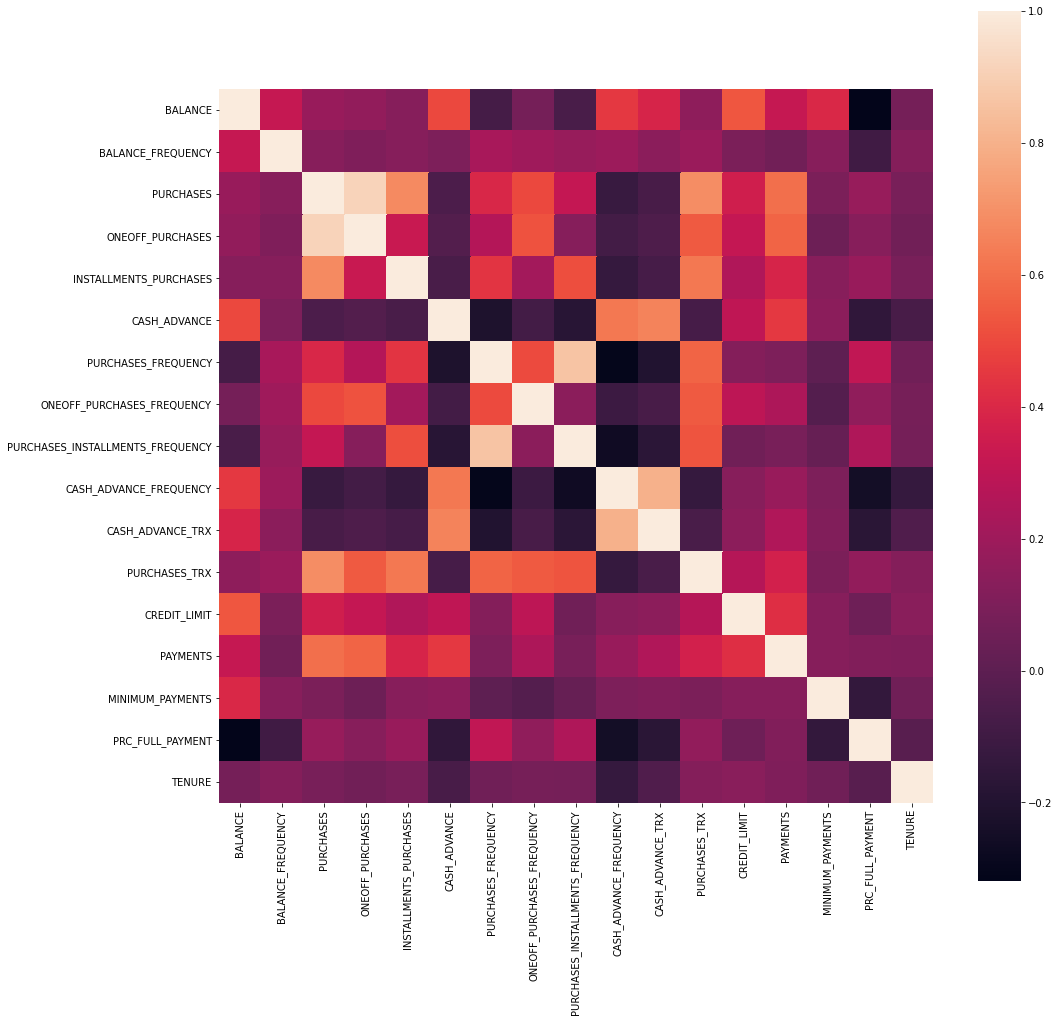


图4 各个变量的热力图

由上图可见，ONEOFF\_PURCHASES和PURCHASES字段之间有相关性，PURCHASES\_FREQUENCY和PURCHASES\_INSTALLMENTS\_FREQUENCY有相关性。根据皮尔森相关性得到具体的相关系数，和热力图相辅相成。

【代码7-2-4-2】financial\_management\_recommendation.py

#%%

# 查看相关性-皮尔森系数

corr\_data = dataset.corr(method='pearson')

#%%

执行代码块。

字段之间的皮尔森系数保存到了corr\_data变量中。在Variable explorer面板中双击corr\_data变量查看它的值，如图5所示。



图5 corr\_data变量的值

通过上图发现，PURCHASES和ONEOFF\_PURCHASES相关性为0.916845。根据皮尔森系数，数据的绝对值越接近于1，2个字段相关性越高。这意味着需要做特征选择或特征提取的工作。在接下来的步骤中将做特征提取。

7.2.5 处理缺失数据

处理缺失数据是建模之前的重要步骤。先统计一下哪些字段有缺失数据，缺失条数是多少。

【代码7-2-5-1】financial\_management\_recommendation.py

#%%

# 检查缺失数据

def get\_missing\_data\_summary():

dataset\_na = dataset.isnull().sum()

dataset\_na = dataset\_na.drop(dataset\_na[dataset\_na == 0].index).sort\_values(ascending=False)

missing\_data = pd.DataFrame({'Missing Count' :dataset\_na})

return missing\_data

missing\_data = get\_missing\_data\_summary()

#%%

执行代码块。

缺失数据的信息存储到了missing\_data变量中。双击该变量查看它的值，如图6所示。

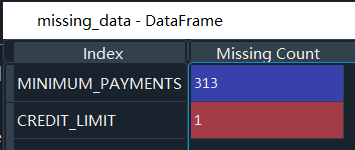


图6 missing\_data变量的值

CREDIT\_LIMIT只有1条数据，可以直接删除。

MINIMUM\_PAYMENTS意思是最低还款，有313条空数据，不能删除。根据填补空缺数据的原则，首先要依据业务来修补。在这里最低还款是基于复杂的公式计算出来的，需要咨询业务人员。在本案例中，为了方便起见，直接删除。

【代码7-2-5-2】financial\_management\_recommendation.py

#%%

# 处理缺失数据

dataset = dataset.drop(dataset[dataset['CREDIT\_LIMIT'].isnull()].index)

dataset = dataset.drop(dataset[dataset['MINIMUM\_PAYMENTS'].isnull()].index)

#%%

执行代码块。

再次检验是否有缺失数据。

【代码7-2-5-3】financial\_management\_recommendation.py

#%%

missing\_data = get\_missing\_data\_summary()

#%%

执行代码块。

缺失数据的检验结果如图7所示。

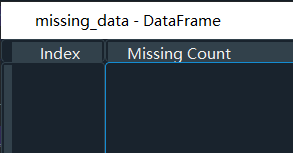


图7 missing\_data变量的值

发现已经没有缺失数据了。

7.2.6 处理类别型字段

【代码7-2-6】financial\_management\_recommendation.py

#%%

# 处理类别型字段

categorical\_features = dataset.dtypes[dataset.dtypes == 'O'].index

print('There are %d categorical features' %(len(categorical\_features)))

#%%

执行代码块。

只有1个字符型字段，这个字段是CUST\_ID（持卡人编号）。

很多时候类别型字段的类型不一定是字符串型，也有可能是数值型，比如性别字段的值是0和1。所以需要检查所有列，确保每一列都是真正的数值型。检查之后发现其余列全部都是数值型。

无类别型字段需要处理，即不需要字符编码、独热编码等操作。

7.2.7 删除无用的变量

上面刚提到的字段CUST\_ID（持卡人编号）是无用的字段，直接删除。

【代码7-2-7】financial\_management\_recommendation.py

#%%

# 移除无用的列

X = dataset.iloc[:,1:].values

#%%

执行代码块。

7.2.8 特征提取

在7.2.4分析字段相关性时，发现自变量之间具有相关性。为了使模型泛化能力更强，所以需要对数据做降维（特征选择或特征提取）。

降维的知识点超纲了，所以本部分不要求掌握。

特征提取是减少变量个数的常用方法，尤其以PCA使用居多。在聚类算法中，由于使用特征提取大大的缩减了变量的个数，使得分析起来会更加简单方便。

【代码7-2-8-1】financial\_management\_recommendation.py

#%%

# 使用PCA降维

from sklearn.decomposition import PCA

pca = PCA(n\_components = None)

X\_pca = pca.fit\_transform(X)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

#%%

执行代码块。

接下来画出number of componence vs cumulative explained variance之间的关系图。这是PCA常用的图，以此来判断前几个字段就可以解释所有字段。

【代码7-2-8-2】financial\_management\_recommendation.py

#%%

# 画出累计方差解释图，用于选择新生成的自变量的个数

plt.figure()

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_), c='orange')

plt.xlabel('number of components')

plt.ylabel('cumulative explained variance')

#%%

执行代码块。

自变量个数 vs 累计方差解释图的效果如图8所示。

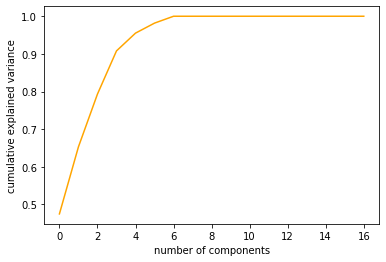


图8 自变量个数 vs 累计方差解释图的效果

如图所示，选取前2个字段可以达到80%的方差解释，基本可以解释整个数据集。这里暂时选取2个字段。同学们有时间可以尝试选取3个或4个自变量。

使用PCA转换自变量。

【代码7-2-8-3】financial\_management\_recommendation.py

#%%

# 选择新生成的自变量的个数

pca = PCA(n\_components = 2)

X\_pca = pca.fit\_transform(X)

#%%

执行代码块。

现在自变量变为X\_pca，不再是X。

PCA是线性的特征提取方法。如果同学们有时间，可以试一下非线性的特征提取方法，如Kernel PCA。

7.2.9 特征缩放

特征缩放是数据预处理的最后一步。建立模型之前建议都需要特征缩放。

【代码7-2-9】financial\_management\_recommendation.py

#%%

# 特征缩放

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_scaled = sc\_X.fit\_transform(X\_pca)

#%%

执行代码块。

数据预处理至此就结束了。下面的案例将使用这些处理好的数据建立机器学习模型，并且给不同客户推荐合适的理财产品。

7.3构建聚类模型

在聚类模型中，Kmeans使用广泛且准确性高。这里使用Kmeans来进行聚类。

7.3.1 聚类个数判断

这里使用轮廓系数来判断聚类的个数。

当K-Means中K的个数在2-20时，不断打印轮廓系数。

【代码7-3-1】financial\_management\_recommendation.py

#%%

# 训练K-Means模型

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score # 轮廓系数

# 打印轮廓系数，确定K-Means中K的个数

for i in range(2, 21):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', n\_init=10, max\_iter=300, random\_state = 0)

kmeans.fit(X\_scaled)

y\_kmeans = kmeans.predict(X\_scaled)

silhouette = silhouette\_score(X\_scaled, y\_kmeans)

print('当聚类个数是%d时，对应的轮廓系数是%.4f' %(i, silhouette))

#%%

执行代码块。

不断打印出的轮廓系数如图9所示。



图9 不断打印出的轮廓系数

通过上图发现，聚类的个数是4时，轮廓系数的值最高，达到0.5715。所以选择4个类别最合适。

7.3.2 构建聚类模型

【代码7-3-2】financial\_management\_recommendation.py

#%%

# 使用K=4建立K-Means模型

kmeans = KMeans(n\_clusters = 4, init = 'k-means++', n\_init=10, max\_iter=300, random\_state = 0)

kmeans.fit(X\_scaled)

y\_kmeans = kmeans.predict(X\_scaled)

#%%

执行代码块。

聚类模型建立完毕。

7.3.3 可视化聚类效果

可视化后，很容易看到分组是否合理。

【代码7-3-3】financial\_management\_recommendation.py

#%%

# 可视化聚类效果

plt.figure()

plt.scatter(X\_scaled[y\_kmeans == 0, 0], X\_scaled[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 0')

plt.scatter(X\_scaled[y\_kmeans == 1, 0], X\_scaled[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 1')

plt.scatter(X\_scaled[y\_kmeans == 2, 0], X\_scaled[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 2')

plt.scatter(X\_scaled[y\_kmeans == 3, 0], X\_scaled[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 3')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('pca1')

plt.ylabel('pca2')

plt.legend()

plt.show()

#%%

执行代码块。

聚类效果的可视化效果如图10所示。

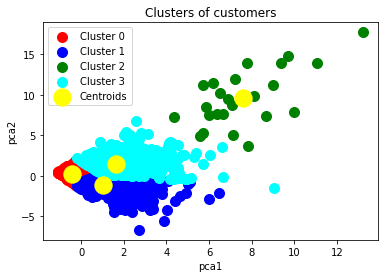


图10 聚类效果

通过上图可以发现，聚类效果看起来是合理的。是否真的合理呢，接下来需要对每一个类别进行分析。

7.4分析每一个分组

首先将聚类结果合并到原始数据集中。

【代码7-4-1】financial\_management\_recommendation.py

#%%

# 对于4个客户分组，分析每一个分组

dataset['cluster'] = y\_kmeans

#%%

执行代码块。

然后查看每个聚类的样本个数

【代码7-4-2】financial\_management\_recommendation.py

#%%

# 查看每个聚类的样本个数

print(dataset.cluster.value\_counts())

#%%

执行代码块。

每个聚类的样本个数如图11所示。

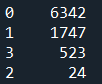


图11 每个聚类的样本个数

由上图所示，分组0客户数最多，分组2客户数最少。因为银行理财产品目标客户为高收入人群，这一群体占比少，故有理由推测，分组2、3大概率是银行的目标客户。

下面开始逐一分析各个分组数据，锁定目标客户。

首先定义4个变量，每个分组1个，把每个分组的样本存储到变量中，写代码比较方便。

【代码7-4-3】financial\_management\_recommendation.py

#%%

# 为每个分组生成单独的变量，便于代码的编写

cluster\_0 = dataset[dataset['cluster'] == 0]

cluster\_1 = dataset[dataset['cluster'] == 1]

cluster\_2 = dataset[dataset['cluster'] == 2]

cluster\_3 = dataset[dataset['cluster'] == 3]

#%%

执行代码块。

cluster\_0这个变量存储着分组0的所有样本，同理cluster\_1这个变量存储着分组1的所有样本。

首先分析信用卡额度：CREDIT\_LIMIT。

【代码7-4-4】financial\_management\_recommendation.py

#%%

# 分析 credit limit 字段

credit\_limit\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.CREDIT\_LIMIT.describe(),

'cluster\_1':cluster\_1.CREDIT\_LIMIT.describe(),

'cluster\_2':cluster\_2.CREDIT\_LIMIT.describe(),

'cluster\_3':cluster\_3.CREDIT\_LIMIT.describe()})

#%%

执行代码块。

credit\_limit\_df这个变量存储着信用卡额度的所有信息，如图12所示。

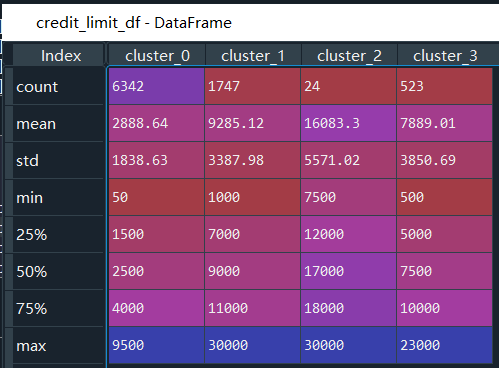


图12 credit\_limit\_df变量的值

通过平均数（mean）和中位数（50%）发现，cluster\_1和cluster\_2和cluster\_3的信用卡额度最高。cluster\_0的信用卡额度较少，且cluster\_0分组的客户数量最多，所以判断cluster\_0这个分组的人群收入较低。cluster\_1和cluster\_2和cluster\_3收入高。

再来分析买东西花了多少钱：PURCHASES。

【代码7-4-5】financial\_management\_recommendation.py

#%%

# 分析 purchases 字段

purchases\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.PURCHASES.describe(),

'cluster\_1':cluster\_1.PURCHASES.describe(),

'cluster\_2':cluster\_2.PURCHASES.describe(),

'cluster\_3':cluster\_3.PURCHASES.describe()})

#%%

执行代码块。

purchases\_df变量的值如图13所示。

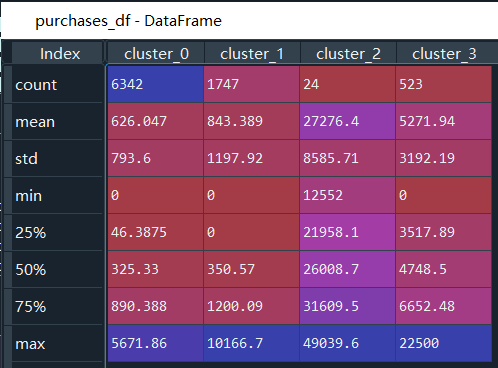


图13 purchases\_df变量的值

通过平均数（mean）和中位数（50%）发现，cluster\_2和cluster\_3买东西花的钱最多。结合上面的分析，判断分组2和分组3属于高收入，高消费群体。

再来看看信用卡取现频率：CASH\_ADVANCE\_FREQUENCY。

【代码7-4-6】financial\_management\_recommendation.py

#%%

# 分析 cash\_advance\_frequency 字段

cash\_advance\_frequency\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.CASH\_ADVANCE\_FREQUENCY.describe(),

'cluster\_1':cluster\_1.CASH\_ADVANCE\_FREQUENCY.describe(),

'cluster\_2':cluster\_2.CASH\_ADVANCE\_FREQUENCY.describe(),

'cluster\_3':cluster\_3.CASH\_ADVANCE\_FREQUENCY.describe()})

#%%

执行代码块。

cash\_advance\_frequency\_df变量的值如图14所示。

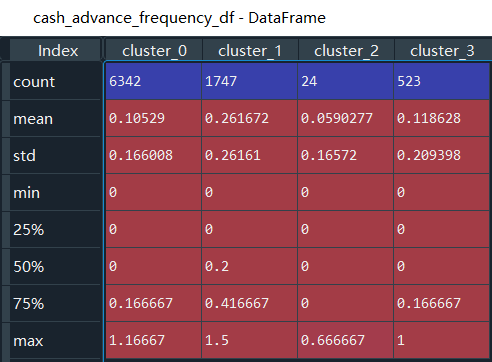


图14 cash\_advance\_frequency\_df变量的值

通过平均数（mean）和中位数（50%）发现，信用卡取现比较频繁的组是分组1，判断分组1资金可能比较紧张。这和上面的分析吻合，分组1虽然信用卡额度高，但是用信用卡消费比较低，而且还经常取现，说明分组1属于中低收入人群。

上面的分析中判断分组2、3是高收入高消费人群。结合分组2的平均曲线是0.05，是所有分组中最小的，所以判断分组2最优。这和上面的分析吻合，分组2额度高、消费高，并且不需要信用卡取现。

最后来分析全额还款情况：PRC\_FULL\_PAYMENT。

【代码7-4-7】financial\_management\_recommendation.py

#%%

# 分析 prc\_full\_payment 字段

prc\_full\_payment\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.PRC\_FULL\_PAYMENT.describe(),

'cluster\_1':cluster\_1.PRC\_FULL\_PAYMENT.describe(),

'cluster\_2':cluster\_2.PRC\_FULL\_PAYMENT.describe(),

'cluster\_3':cluster\_3.PRC\_FULL\_PAYMENT.describe()})

#%%

执行代码块。

prc\_full\_payment\_df变量的值如图15所示。

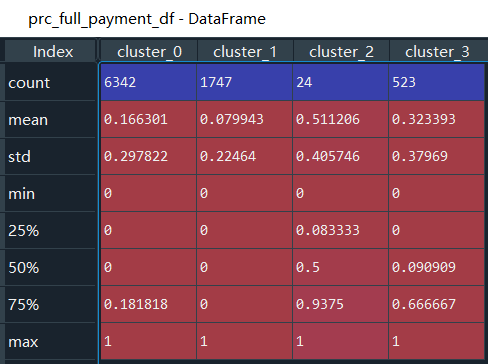


图15 prc\_full\_payment\_df变量的值

通过平均数（mean）和中位数（50%）发现，分组2每个月还款几乎都是一次性还完，再次印证了分组2有钱、不缺钱、出手扩初。

也发现cluster\_2中也有一小部分人没有把账单全部还完。把这部分人过滤出去。

【代码7-4-8】financial\_management\_recommendation.py

#%%

# 得到目标客户

target\_dataset = dataset[dataset['cluster']==2]

target\_dataset = target\_dataset.drop(target\_dataset[target\_dataset['PRC\_FULL\_PAYMENT']<0.6].index)

#%%

执行代码块。

分组2共有24个客户，过滤掉了13个客户，最终剩下11个客户。这11个客户可作为银行理财产品优质目标客户。

8. 案例代码

【案例代码】financial\_management\_recommendation.py

'''

基于K-Means根据信用卡消费情况推荐银行理财产品

'''

#%%

# 导入包

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

#%%

#%%

# 读取数据集

dataset = pd.read\_csv('credit\_card.csv')

#%%

#%%

# 查看数据集情况

dataset\_description\_df = dataset.describe()

#%%

#%%

# 查看相关性-热力图

import seaborn as sns

corrmat = dataset.corr()

plt.figure(figsize=(16,16))

plt.xticks(rotation=90)

plt.yticks(rotation=90)

sns.heatmap(corrmat, vmax=1.0, square=True)

#%%

#%%

# 查看相关性-皮尔森系数

corr\_data = dataset.corr(method='pearson')

#%%

#%%

# 检查缺失数据

def get\_missing\_data\_summary():

dataset\_na = dataset.isnull().sum()

dataset\_na = dataset\_na.drop(dataset\_na[dataset\_na == 0].index).sort\_values(ascending=False)

missing\_data = pd.DataFrame({'Missing Count' :dataset\_na})

return missing\_data

missing\_data = get\_missing\_data\_summary()

#%%

#%%

# 处理缺失数据

dataset = dataset.drop(dataset[dataset['CREDIT\_LIMIT'].isnull()].index)

dataset = dataset.drop(dataset[dataset['MINIMUM\_PAYMENTS'].isnull()].index)

#%%

#%%

# 再次检查缺失数据

missing\_data = get\_missing\_data\_summary()

#%%

#%%

# 处理类别型字段

categorical\_features = dataset.dtypes[dataset.dtypes == 'O'].index

print('There are %d categorical features' %(len(categorical\_features)))

#%%

#%%

# 移除无用的列

X = dataset.iloc[:,1:].values

#%%

#%%

# 使用PCA降维

from sklearn.decomposition import PCA

pca = PCA(n\_components = None)

X\_pca = pca.fit\_transform(X)

explained\_variance\_ratio = pca.explained\_variance\_ratio\_

#%%

#%%

# 画出累计方差解释图，用于选择新生成的自变量的个数

plt.figure()

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_), c='orange')

plt.xlabel('number of components')

plt.ylabel('cumulative explained variance')

#%%

#%%

# 选择新生成的自变量的个数

pca = PCA(n\_components = 2)

X\_pca = pca.fit\_transform(X)

#%%

#%%

# 特征缩放

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_scaled = sc\_X.fit\_transform(X\_pca)

#%%

#%%

# 训练K-Means模型

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score # 轮廓系数

# 打印轮廓系数，确定K-Means中K的个数

for i in range(2, 21):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', n\_init=10, max\_iter=300, random\_state = 0)

kmeans.fit(X\_scaled)

y\_kmeans = kmeans.predict(X\_scaled)

silhouette = silhouette\_score(X\_scaled, y\_kmeans)

print('当聚类个数是%d时，对应的轮廓系数是%.4f' %(i, silhouette))

#%%

#%%

# 使用K=4建立K-Means模型

kmeans = KMeans(n\_clusters = 4, init = 'k-means++', n\_init=10, max\_iter=300, random\_state = 0)

kmeans.fit(X\_scaled)

y\_kmeans = kmeans.predict(X\_scaled)

#%%

#%%

# 可视化聚类效果

plt.figure()

plt.scatter(X\_scaled[y\_kmeans == 0, 0], X\_scaled[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 0')

plt.scatter(X\_scaled[y\_kmeans == 1, 0], X\_scaled[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 1')

plt.scatter(X\_scaled[y\_kmeans == 2, 0], X\_scaled[y\_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 2')

plt.scatter(X\_scaled[y\_kmeans == 3, 0], X\_scaled[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cluster 3')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow', label = 'Centroids')

plt.title('Clusters of customers')

plt.xlabel('pca1')

plt.ylabel('pca2')

plt.legend()

plt.show()

#%%

#%%

# 对于4个客户分组，分析每一个分组

dataset['cluster'] = y\_kmeans

#%%

#%%

# 查看每个聚类的样本个数

print(dataset.cluster.value\_counts())

#%%

#%%

# 为每个分组生成单独的变量，便于代码的编写

cluster\_0 = dataset[dataset['cluster'] == 0]

cluster\_1 = dataset[dataset['cluster'] == 1]

cluster\_2 = dataset[dataset['cluster'] == 2]

cluster\_3 = dataset[dataset['cluster'] == 3]

#%%

#%%

# 分析 credit limit 字段

credit\_limit\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.CREDIT\_LIMIT.describe(),

'cluster\_1':cluster\_1.CREDIT\_LIMIT.describe(),

'cluster\_2':cluster\_2.CREDIT\_LIMIT.describe(),

'cluster\_3':cluster\_3.CREDIT\_LIMIT.describe()})

#%%

#%%

# 分析 purchases 字段

purchases\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.PURCHASES.describe(),

'cluster\_1':cluster\_1.PURCHASES.describe(),

'cluster\_2':cluster\_2.PURCHASES.describe(),

'cluster\_3':cluster\_3.PURCHASES.describe()})

#%%

#%%

# 分析 cash\_advance\_frequency 字段

cash\_advance\_frequency\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.CASH\_ADVANCE\_FREQUENCY.describe(),

'cluster\_1':cluster\_1.CASH\_ADVANCE\_FREQUENCY.describe(),

'cluster\_2':cluster\_2.CASH\_ADVANCE\_FREQUENCY.describe(),

'cluster\_3':cluster\_3.CASH\_ADVANCE\_FREQUENCY.describe()})

#%%

#%%

# 分析 prc\_full\_payment 字段

prc\_full\_payment\_df = pd.DataFrame(data={

'cluster\_0':cluster\_0.PRC\_FULL\_PAYMENT.describe(),

'cluster\_1':cluster\_1.PRC\_FULL\_PAYMENT.describe(),

'cluster\_2':cluster\_2.PRC\_FULL\_PAYMENT.describe(),

'cluster\_3':cluster\_3.PRC\_FULL\_PAYMENT.describe()})

#%%

#%%

# 得到目标客户

target\_dataset = dataset[dataset['cluster']==2]

target\_dataset = target\_dataset.drop(target\_dataset[target\_dataset['PRC\_FULL\_PAYMENT']<0.6].index)

#%%