RFM 模型注意事项

今天的分享内容比较多，一共有五个要注意的点，大家要好好看一看哦~

话不多说，我们先来看第一点。

一、其它计算阈值的方法

在课程中，我们讲到在数据量大的情况下，我们也可以通过 R、F、M 各值的原始数据，直接计算平均数或中位数的方式来获得阈值，计算起来也相对简单。

现在我们以课程中已经清洗和整理好的数据为例，直接对原始数据计算阈值。

这是我们在课程中整理好的 R、F、M 各值的原始数据：

```
# 按【用户 ID】分组后，获取【时间间隔】列的最小值、【订单号】列的数量，以及【总金额】列的总和
rfm_data = grouped_data.groupby('用户 ID', as_index=False).agg({'时间间隔': 'min', '订单号': 'count', '总金额': 'sum'})
```

```
# 修改列名
rfm_data.columns = ['用户ID', '时间间隔', '总次数', '总金额']
rfm_data
```

	用户ID	时间间隔	总次数	总金额
0	12346	347	1	77183.60
1	12347	61	7	4310.00
2	12348	97	4	1797.24
3	12349	40	1	1757.55
4	12350	332	1	334.40
...	...	...	...	...
4334	18280	181	1	180.60
4335	18281	25	1	80.82
4336	18282	237	2	178.05
4337	18283	31	16	2094.88
4338	18287	21	3	1837.28

4339 rows × 4 columns

先来看看如何通过直接计算平均值的方式获取阈值：

在上一个案例中，我们学习了agg()方法，它可以帮助我们同时获取到三列数据的均值，我们先来复习一下它的用法：

分组结果.agg(func=列名和函数名称（字符串）组成的字典)		
可使用的函数	功能	示例
min()	求最小值	data_11.groupby('名称').agg({'单价': 'min'})
max()	求最大值	data_11.groupby('名称').agg({'单价': 'max'})
sum()	求和	data_11.groupby('名称').agg({'库存': 'sum'})
mean()	求平均值	data_11.groupby('名称').agg({'单价': 'mean'})
count()	求频数	data_11.groupby('名称').agg({'地址': 'count'})
std()	求标准差	data_11.groupby('名称').agg({'单价': 'std'})

当然在这里我们不需要对数据进行分组，直接计算平均值即可：

```
rfm_data.agg({'时间间隔': 'mean', '总次数': 'mean', '总金额': 'mean'})
```

```
rfm_data.agg({'时间间隔': 'mean', '总次数': 'mean', '总金额': 'mean'})
```

```
时间间隔      126.461858
总次数          4.271952
总金额       2053.789683
dtype: float64
```

可以看到仅用了一行代码，我们已经分别得到了 R、F、M 的平均值。

然后我们再来看看如何计算这组数据的中位数：

思路非常简单，正好可以用到上一关所学的知识。我们对数据中的R、F、M值分别进行排序，然后重置排序后的行索引，然后就可以取到它们的中位数啦。

先一起来复习一下这两个方法吧：

df.sort_values(by)		
功能：对序列进行排序，默认按升序对数据进行排序		
参数	说明	示例
by	值可以是列名	data_3.sort_values(by='平均单价')

s/df.reset_index(drop)		
功能：重置行索引，并使用默认索引		
参数	说明	示例
drop	值为布尔值，默认情况下为 False。 1. 值为 False：保留原索引； 2. 值为 True：把原来的索引去掉。	data_6.reset_index(drop=True)

以时间间隔这一列为例，我们先提取【时间间隔】这一列的数据并进行排序：

```
rfm_data['时间间隔'].sort_values()
```

```
rfm_data['时间间隔'].sort_values()
```

```
4338      21
2257      21
3614      21
2177      21
3627      21
        ...
2716     718
3868     718
3104     718
355      718
1393     718
Name: 时间间隔, Length: 4339, dtype: int64
```

然后用reset_index()方法重置数据的行索引：

```
sorted_data = rfm_data['时间间隔'].sort_values()
reset_data = sorted_data.reset_index(drop=True)
reset_data
```

```
sorted_data = rfm_data['时间间隔'].sort_values()
reset_data = sorted_data.reset_index(drop=True)
reset_data
```

```
0         21
1         21
2         21
3         21
4         21
       ...
4334     718
4335     718
4336     718
4337     718
4338     718
Name: 时间间隔, Length: 4339, dtype: int64
```

可以看到一共有4339行数据，那么它的中位数自然就是第2170行的数据啦：

```
reset_data[2169]
```

```
reset_data[2169]
```

```
82
```

用同样的方法，我们分别对【总次数】和【总金额】的数据进行排序，再重置索引，就可以分别得到它们的中位数啦，进而获取到 R、F、M的阈值。

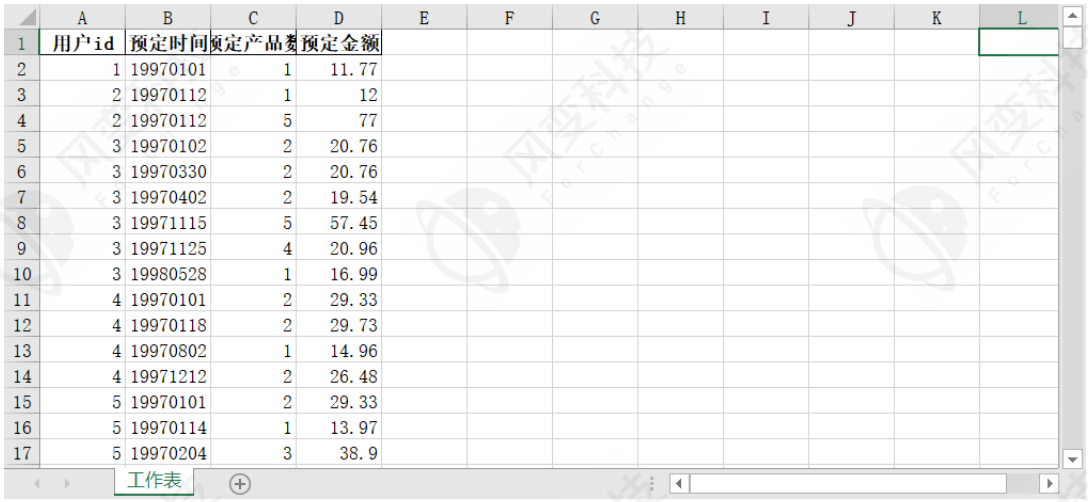
当然更直接的方法还是用前面学习的median()方法直接获取中位数啦，大家也可以自己试一试哦~

二、提取RFM值的小练习

通过学习本关的案例，我们不难发现在实际业务中使用 RFM 模型，原始数据不一定提供明确的 RFM 值，可能需要进行换算才能得到。

所以在这里，也给大家提供了一个练习，让大家可以自己尝试如何从数据中提取出 RFM 值。

数据集打开是这样的：



	A	B	C	D	E	F	G	H	I	J	K	L
1	用户id	预定时间	预定产品数	预定金额								
2	1	19970101	1	11.77								
3	2	19970112	1	12								
4	2	19970112	5	77								
5	3	19970102	2	20.76								
6	3	19970330	2	20.76								
7	3	19970402	2	19.54								
8	3	19971115	5	57.45								
9	3	19971125	4	20.96								
10	3	19980528	1	16.99								
11	4	19970101	2	29.33								
12	4	19970118	2	29.73								
13	4	19970802	1	14.96								
14	4	19971212	2	26.48								
15	5	19970101	2	29.33								
16	5	19970114	1	13.97								
17	5	19970204	3	38.9								

工作表

练习用表.xlsx
1.7MB

大家也可以自己下载这个表进行练习哦~

我们首先读取并查看这张表前10行的数据：

```
data = pd.read_excel('练习用表.xlsx')
data.head(10)
```

	用户id	预定时间	预定产品数	预定金额
0	1	19970101	1	11.77
1	2	19970112	1	12.00
2	2	19970112	5	77.00
3	3	19970102	2	20.76
4	3	19970330	2	20.76
5	3	19970402	2	19.54
6	3	19971115	5	57.45
7	3	19971125	4	20.96
8	3	19980528	1	16.99
9	4	19970101	2	29.33

查看前10行数据，每个数据一共4列，分别是用户id、预定时间、预定产品数、预定金额。

接着观察数据，看看是否能提取到 RFM 模型所需的目标数据。

如果想获取某用户的最近一次消费时间间隔（R），可以通过该用户的【用户 id】查找订单最新的一次【预定时间】，计算与当前时间的时间差。

如果想获取某用户的消费频率（F），只需要计算同一个【用户 id】出现的次数即可。

如果想获取某用户的消费金额（M），只需要将该用户的所有消费额进行加和。

我们先来查看数据的基本信息：

```
# 查看数据的基本信息总结
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69659 entries, 0 to 69658
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   用户id    69659 non-null  int64  
 1   预定时间    69659 non-null  int64  
 2   预定产品数   69659 non-null  int64  
 3   预定金额    69659 non-null  float64
dtypes: float64(1), int64(3)
memory usage: 2.1 MB
```

前3列为int类型，最后1列为float类型。

接下来我们需要对数据进行预处理，数据预处理包括空值处理、重复值处理、异常值处理，数据变换等。

首先我们查看一下数据是否有空值：

```
data.isnull().sum()
```

```
用户id      0
预定时间      0
预定产品数     0
预定金额      0
最近预定时间    0
dtype: int64
```

可以看到没有空值，数据很干净。

接着，我们来查找重复数据：

```
# 查找重复数据
data[data.duplicated()]
```

	用户id	预定时间	预定产品数	预定金额
1381	398	19980518	1	12.99
1665	499	19971001	1	11.49
1667	499	19971001	1	15.49
1669	499	19971001	1	13.99
1676	499	19971002	1	11.49
...	...	...	...	...
68228	23043	19970323	1	13.97
68282	23061	19970405	1	14.96
69151	23373	19980114	1	12.99
69178	23380	19980609	1	9.49
69218	23394	19970804	1	11.77

255 rows × 4 columns

可以看到一共有255个重复数据。接着我们可以使用 drop_duplicates() 方法删除重复数据：

```
# 删除重复值
data = data.drop_duplicates()
# 查找清洗后的数据是否存在重复数据
data[data.duplicated()]
```

	用户id	预定时间	预定产品数	预定金额

最后，使用 describe() 查看数据的描述性统计信息，检查数据极大极小值是否符合常识和业务要求：

```
data.describe()
```

	用户id	预定时间	预定产品数	预定金额
count	69404.000000	6.940400e+04	69404.000000	69404.000000
mean	11468.913766	1.997228e+07	2.414558	35.963097
std	6814.368605	3.837687e+03	2.336528	36.318489
min	1.000000	1.997010e+07	1.000000	0.000000
25%	5509.000000	1.997022e+07	1.000000	14.490000
50%	11410.000000	1.997042e+07	2.000000	25.980000
75%	17261.000000	1.997111e+07	3.000000	43.720000
max	23570.000000	1.998063e+07	99.000000	1286.010000

可以看到订单金额的最小值出现了0，可能是免费活动，参与免费获得的客户不具有明显价值，可以剔除：

```
data=data.drop(index=(data.loc[(data['预定金额']==0)].index)) #删除异常值
data.describe()
```

	用户id	预定时间	预定产品数	预定金额
count	69324.000000	6.932400e+04	69324.000000	69324.000000
mean	11470.227569	1.997228e+07	2.416191	36.004598
std	6813.909552	3.838451e+03	2.337382	36.318874
min	1.000000	1.997010e+07	1.000000	1.630000
25%	5509.750000	1.997022e+07	1.000000	14.490000
50%	11414.000000	1.997042e+07	2.000000	25.980000
75%	17262.250000	1.997111e+07	3.000000	43.730000
max	23570.000000	1.998063e+07	99.000000	1286.010000

此时观察到预定时间是int型，需要转换为时间格式：

```
data['预定时间'] = pd.to_datetime(data['预定时间'],format = '%Y%m%d')
data.head()
```

	用户id	预定时间	预定产品数	预定金额
0	1	1997-01-01	1	11.77
1	2	1997-01-12	1	12.00
2	2	1997-01-12	5	77.00
3	3	1997-01-02	2	20.76
4	3	1997-03-30	2	20.76

可以看到已成功转换为datetime类型。

```
data['预定时间'].describe()
```

```
count                   69324
unique                    546
top       1997-02-24 00:00:00
freq                      502
first     1997-01-01 00:00:00
last      1998-06-30 00:00:00
Name: 预定时间, dtype: object
```

查看预定时间的描述性统计，可以看到数据集的时间集中在1997年初到1998年中旬；
基于此，将此次观察日期定义为1998年6月30日。

RFM的定义：R为最近一次下单时间，F为购买频率，M为最近一次订单金额；
原始数据中还缺最近一次下单时间，所以需要我们计算并添加进去：

```
data['最近预定时间']= pd.to_datetime('1998-06-30')-data['预定时间']
data.head()
```

	用户id	预定时间	预定产品数	预定金额	最近预定时间
0	1	1997-01-01	1	11.77	545 days
1	2	1997-01-12	1	12.00	534 days
2	2	1997-01-12	5	77.00	534 days
3	3	1997-01-02	2	20.76	544 days
4	3	1997-03-30	2	20.76	457 days

然后我们使用agg()方法的各种函数来求出最近一次订单 R，订单频率 F 和订单总金额 M，同时我们用columns重命名列：最近预定时间 R，预定产品数 F 和预定金额 M：

```
rfm_data = data.groupby(['用户id'],as_index = False).agg({'最近预定时间':'min','预定产品数':'count','预定金额':'sum'})
rfm_data.columns = ['用户id','R','F','M']  #重命名列：最近一次订单r，订单频率f和订单总金额m
rfm_data
```

	用户id	R	F	M
0	1	545 days	1	11.77
1	2	534 days	2	89.00
2	3	33 days	6	156.46
3	4	200 days	4	100.50
4	5	178 days	11	385.61
...	...	...	...	...
23497	23566	462 days	1	36.00
23498	23567	462 days	1	20.97
23499	23568	434 days	3	121.70
23500	23569	462 days	1	25.74
23501	23570	461 days	2	94.08

23502 rows × 4 columns

由此我们就完成了对原始数据RFM的提取，然后就可以接着进行下面的操作啦~

三、如何区分 RFM 模型各值的价值大小与分数大小

在本关案例中所讲的RFM模型中，我们是按价值打分，而不是值的大小打分。

什么意思呢？也就是说，我们是通过R、F、M各值对我们实际业务的价值来打分，而不是简单的因为哪个数值大分就打的高。

比如 R 值，消费时间间隔越近，即 R 值越小，说明用户再次消费的可能性就越高，也就说明了用户的价值越高，打的分数越高。

四、RFM值的打分规则

在本关案例中，作为演示，我们假设 R、F、M 各值按价值从小到大分为 1~5 分，并制定了如下图所示的打分规则：

按价值打分	最近一消费时间间隔（R）	消费频率（F）	消费金额（M）
1	20 天以上	1 次	1000 元以内
2	10 ~ 20 天	2 次	1000 ~ 1500 元
3	5 ~ 10 天	3 次	1500 ~ 3000 元
4	3 ~ 5 天	4 次	3000 ~ 5000 元
5	3 天以内	5 次	5000 元以上

但是大家千万不要以为 RFM 的打分范围就只能是1-5分哦！

我们在实际分析的时候，要根据我们的实际业务来决定，有的可能更少，有的可能更多。

打分的目的是为了帮助我们将用户的RFM值与各值的平均值进行对比，进而判断用户属于哪种类别，所以大家在实际分析过程中一定要灵活使用，不能固化思维哦~

五、to_excel() 函数拓展

最后一点是关于我们在本案例中所学习的基础知识：to_excel()函数。它可以以 DataFrame 格式将数据写入到 Excel 文件。

它的语法是这样的：

```
df.to_excel(excel_writer, sheet_name='Sheet1', index=True)
```

功能：以 DataFrame 格式将数据写入 Excel 文件

参数	说明	示例
excel_writer	Excel 文件路径或文件对象	df.to_excel('output.xlsx')
sheet_name	Excel 文件的工作表名称，默认为 'Sheet1'	df.to_excel('output.xlsx', sheet_name='Sheet2')
index	决定是否在写入的文件里加入行索引，默认为 True	df.to_excel('intput.xlsx', sheet_name='Sheet3', index=False)

但实际上它有一个隐藏功能。

我们以课程中的例子来演示，首先通过下方代码创建一个 DataFrame 对象：

```
# 创建一个 DataFrame 对象的数据
data_2 = pd.DataFrame({'学号':[1, 2, 3], '性别':['男', '男', '女'], '年龄':['17', '17', '16'], '总分':['285', '273', '240']})
data_2
```

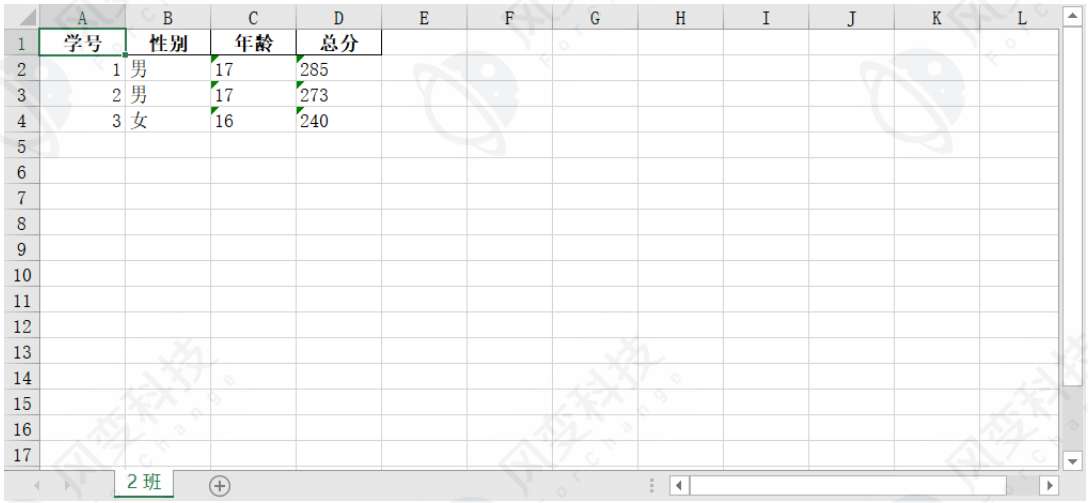
```
# 创建一个 DataFrame 对象的数据
data_2 = pd.DataFrame({'学号':[1, 2, 3], '性别':['男', '男', '女'], '年龄':['17', '17', '16'], '总分':['285', '273', '240']})
data_2
```

	学号	性别	年龄	总分
0	1	男	17	285
1	2	男	17	273
2	3	女	16	240

然后创建一个工作表名为【2 班】的工作簿【新成绩单.xlsx】，并将创建的 DataFrame 数据写入到该工作表：

```
# 将数据写入到【新成绩单.xlsx】工作簿中的【2 班】工作表
data_2.to_excel('新成绩单.xlsx', sheet_name='2 班', index=False)
```

我们打开这个【新成绩单.xlsx】的文件，可以看到是这样的：



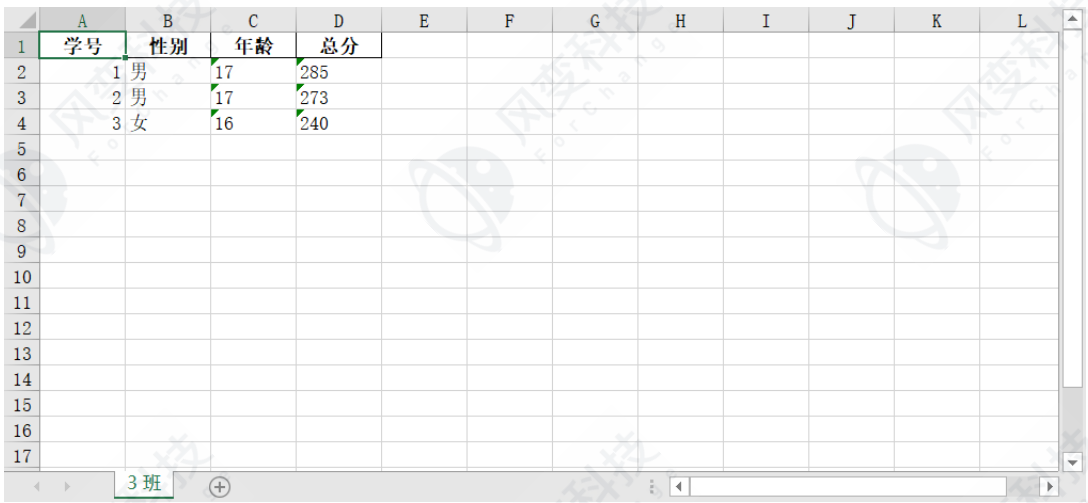
	A	B	C	D
1	学号	性别	年龄	总分
2	1	男	17	285
3	2	男	17	273
4	3	女	16	240

2 班

但是当我们第二次执行时，修改参数 sheet_name 的值为 3 班：

```
# 第二次执行下面的代码
data_2.to_excel('新成绩单.xlsx', sheet_name='3 班', index=False)
```

在本地查看生成的文件时，【新成绩单.xlsx】工作簿只会有一个工作表【3 班】：



	A	B	C	D
1	学号	性别	年龄	总分
2	1	男	17	285
3	2	男	17	273
4	3	女	16	240

3 班

原因是函数会重写整个Excel之后才会存储。也就是说它和写入模式'w'一样是具有覆盖性的，大家一定要注意哦~

【特别推荐】——风变Python学堂公众号

有Python知识干货、明星讲师直播、Python应用案例讲解等，帮大家学好Python，用好Python！
现在关注【风变Python学堂】，还可领取专属**【资料包】**，快扫下方二维码领取福利吧！