课程说明：

小伙伴好呀~欢迎来到《2021机器学习实战训练营》（第二期）试学体验课！我是课程主讲老师，九天。

本次体验课为期三天（10月21-23号），期间每晚8点在我的B站直播间公开直播，直播间地址:https://live.bilibili.com/22678166

本期公开课将围绕一项kaggle竞赛案例进行深度剖析，并据此讨论算法竞赛中机器学习的一般建模流程，以及当前机器学习进行预测时最为有效的技术手段，也就是特征工程方法和集成算法的相关应用，对算法和竞赛感兴趣的小伙伴，欢迎积极参与讨论哦~

课程资料领取/数据技术交流/付费课程信息，扫码添加客服“小可爱”获取哦~

另外，《机器学习实战训练营》（第二期）本月25号即将开课，十八周80+课时体系大课限时半价，扫码咨询小可爱回复“优惠”，还可领取额外折上折优惠，课程主页：https://appze9inzwc2314.pc.xiaoe-tech.com

【Kaggle】Elo Merchant Category Recommendation

竞赛案例解析公开课

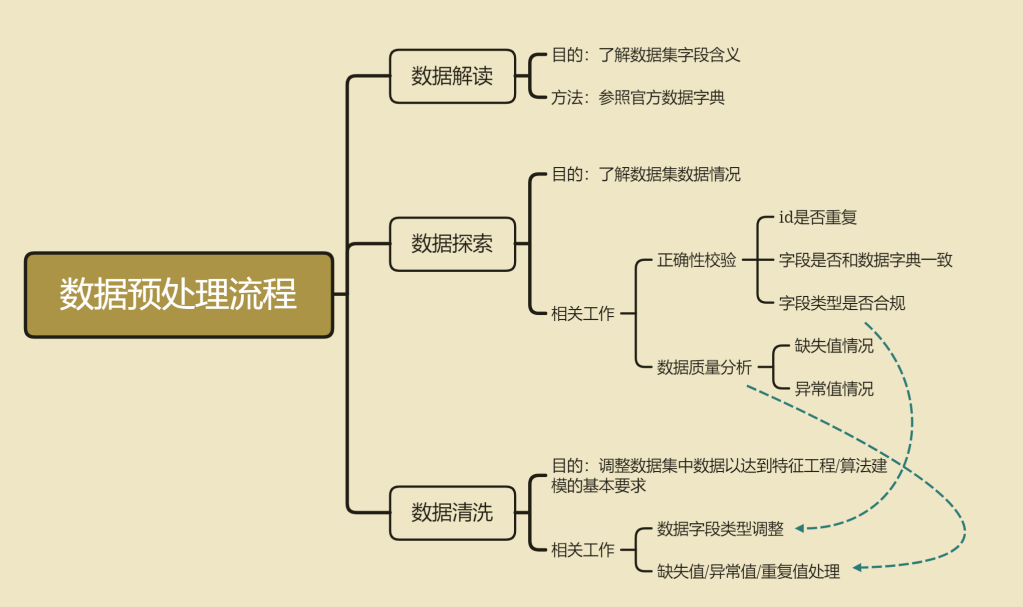
Day 2.交易数据和商户数据的数据探索与数据清洗

在对train和test数据集完成探索性分析之后，接下来我们需要进一步围绕官方给出的商户数据与信用卡交易数据进行解读和分析，并对其进行数据清洗，从而为后续的特征工程和算法建模做准备。

一般来说，在数据解读、数据探索和初步数据清洗都是同步进行的，都是前期非常重要的工作事项。其中，数据解读的目的是为了快速获取数据集的基本信息，通过比对官方给出的字段解释，快速了解数据集的字段含义，这对于许多复杂数据场景下的建模是非常有必要的。而数据探索，顾名思义，就是快速了解数据集的基本数据情况，主要工作包括数据正确性校验和数据质量

分析，核心目的是为了能够快速了解各字段的基本情况，包括默认各字段的数据类型、数据集是否存在数据不一致的情况、数据集重复值情况、缺失值情况等，当然，通过一系列的数据探索，也能够快速加深对数据集的理解。当然，数据探索结束之后，就需要进行数据清洗了，所谓数据清洗，指的是在建模/特征工程之前进行的必要的调整，以确保后续操作可执行，包括数据字段类型调整、重复值处理、缺失值处理等等，当然，有些操作可能在后续会进行些许优化，比如数据清洗阶段我们可以先尝试进行较为简单的缺失值填补，在后续的建模过程中我们还可以根据实际建模结果来调整缺失值填补策略。

我们也可将数据探索与数据清洗的过程总结如下：



接下来我们将对商户数据、交易数据的三张表进行数据探索和数据清洗。

一、商户数据解读与探索

接下来我们来看主办方给出的拓展信息。也就是信用卡交易记录和商户相关数据，这几张表中同时包含了训练集、测试集中所有信用卡的部分记录，是挖掘有效信息、提高模型建模效果的重要渠道。

1.数据解读

首先我们先来查看数据量相对较小的商户信息表，也就是merchants.csv中的相关信息。

In [14…
```
# 导入数据
merchant = pd.read_csv('./eloData/merchants.csv', header=0)
```

In [11…
```
merchant.head(5)
```

Out[110]:

	merchant_id	merchant_group_id	merchant_category_id	subsector_id	numerical_1	numerical
0	M_ID_838061e48c	8353	792	9	-0.057471	-0.0574
1	M_ID_9339d880ad	3184	840	20	-0.057471	-0.0574
2	M_ID_e726bbae1e	447	690	1	-0.057471	-0.0574
3	M_ID_a70e9c5f81	5026	792	9	-0.057471	-0.0574
4	M_ID_64456c37ce	2228	222	21	-0.057471	-0.0574

5 rows × 22 columns

In [11…]
```
merchant.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334696 entries, 0 to 334695
Data columns (total 22 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   merchant_id                 334696 non-null  object 
 1   merchant_group_id           334696 non-null  int64  
 2   merchant_category_id        334696 non-null  int64  
 3   subsector_id                334696 non-null  int64  
 4   numerical_1                 334696 non-null  float64
 5   numerical_2                 334696 non-null  float64
 6   category_1                  334696 non-null  object 
 7   most_recent_sales_range     334696 non-null  object 
 8   most_recent_purchases_range 334696 non-null  object 
 9   avg_sales_lag3              334683 non-null  float64
 10  avg_purchases_lag3          334696 non-null  float64
 11  active_months_lag3          334696 non-null  int64  
 12  avg_sales_lag6              334683 non-null  float64
 13  avg_purchases_lag6          334696 non-null  float64
 14  active_months_lag6          334696 non-null  int64  
 15  avg_sales_lag12             334683 non-null  float64
 16  avg_purchases_lag12         334696 non-null  float64
 17  active_months_lag12         334696 non-null  int64  
 18  category_4                  334696 non-null  object 
 19  city_id                     334696 non-null  int64  
 20  state_id                    334696 non-null  int64  
 21  category_2                  322809 non-null  float64
dtypes: float64(9), int64(8), object(5)
memory usage: 56.2+ MB
```

数据集基本字段解释如下：

In [11…]
```
# 在数据字典中查看各字段的解释
df = pd.read_excel('./eloData/Data_Dictionary.xlsx', header=2, sheet_name='merchant')
df
```

Out[114]:

	Columns	Description
0	merchant_id	Unique merchant identifier
1	merchant_group_id	Merchant group (anonymized)
2	merchant_category_id	Unique identifier for merchant category (anony...
3	subsector_id	Merchant category group (anonymized)
4	numerical_1	anonymized measure
5	numerical_2	anonymized measure
6	category_1	anonymized category
7	most_recent_sales_range	Range of revenue (monetary units) in last acti...
8	most_recent_purchases_range	Range of quantity of transactions in last acti...
9	avg_sales_lag3	Monthly average of revenue in last 3 months di...
10	avg_purchases_lag3	Monthly average of transactions in last 3 mont...
11	active_months_lag3	Quantity of active months within last 3 months

	Columns	Description
12	avg_sales_lag6	Monthly average of revenue in last 6 months di...
13	avg_purchases_lag6	Monthly average of transactions in last 6 mont...
14	active_months_lag6	Quantity of active months within last 6 months
15	avg_sales_lag12	Monthly average of revenue in last 12 months d...
16	avg_purchases_lag12	Monthly average of transactions in last 12 mon...
17	active_months_lag12	Quantity of active months within last 12 months
18	category_4	anonymized category
19	city_id	City identifier (anonymized)
20	state_id	State identifier (anonymized)
21	category_2	anonymized category

实际含义如下：

字段	解释
merchant_id	商户id
merchant_group_id	商户组id
merchant_category_id	商户类别id
subsector_id	商品种类群id
numerical_1	匿名数值特征1
numerical_2	匿名数值特征2
category_1	匿名离散特征1
most_recent_sales_range	上个活跃月份收入等级，有序分类变量A>B>...>E
most_recent_purchases_range	上个活跃月份交易数量等级，有序分类变量A>B>...>E
avg_sales_lag3/6/12	过去3、6、12个月的月平均收入除以上一个活跃月份的收入
avg_purchases_lag3/6/12	过去3、6、12个月的月平均交易量除以上一个活跃月份的交易量
active_months_lag3/6/12	过去3、6、12个月的活跃月份数量
category_2	匿名离散特征2

能够发现，数据表中提供不仅提供了商户的基本属性字段（如类别和商品种群等），同时也提供了商户近期的交易数据。不过和此前一样，仍然存在大量的匿名特征。

2.数据探索

在理解数据字段的基本含义后，接下来我们进一步进行数据探索：

- 正确性校验

  接下来简单对数据集的基本情况进行验证。首先是商户id出现次数的计算：

In [11…

```
print(merchant.shape, merchant['merchant_id'].nunique())
```

```
(334696, 22) 334633
```

能够看出，该表并不是一个id对应一条数据，存在一个商户有多条记录的情况。此外，由于商户

特征较多，此处我们也可以简单验证商户数据特征是否和数据字典中特征一致：

In [11…
```
print(pd.Series(merchant.columns.tolist()).sort_values().values ==  pd.Series([va[0]
```

```
[ True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True]
```

能够看出商户特征完全一致。

- 缺失值分析

  进一步，查看商户数据缺失值情况：

In [12…
```
merchant.isnull().sum()
```

Out[120]:
```
merchant_id                      0
merchant_group_id                0
merchant_category_id             0
subsector_id                     0
numerical_1                      0
numerical_2                      0
category_1                       0
most_recent_sales_range          0
most_recent_purchases_range      0
avg_sales_lag3                  13
avg_purchases_lag3               0
active_months_lag3               0
avg_sales_lag6                  13
avg_purchases_lag6               0
active_months_lag6               0
avg_sales_lag12                 13
avg_purchases_lag12              0
active_months_lag12              0
category_4                       0
city_id                          0
state_id                         0
category_2                   11887
dtype: int64
```

能够发现，第二个匿名分类变量存在较多缺失值，而avg_sales_lag3/6/12缺失值数量一致，则很有可能是存在13个商户同时确实了这三方面信息。其他数据没有缺失，数据整体来看较为完整。

3.数据预处理

接下来对商户数据进行数据预处理。由于还未进行特征工程，此处预处理只是一些不影响后续特征工程、建模或多表关联的、较为初步但又是必须要做的预处理。

- 离散/连续字段标注

由于商户数据集中特征同时存在分类变量和离散变量，因此我们首先可以根据字段的说明对不同属性特征进行统一的划分：

In [12…
```
category_cols = ['merchant_id', 'merchant_group_id', 'merchant_category_id',
       'subsector_id', 'category_1',
       'most_recent_sales_range', 'most_recent_purchases_range',
       'category_4', 'city_id', 'state_id', 'category_2']
numeric_cols = ['numerical_1', 'numerical_2',
     'avg_sales_lag3', 'avg_purchases_lag3', 'active_months_lag3',
       'avg_sales_lag6', 'avg_purchases_lag6', 'active_months_lag6',
       'avg_sales_lag12', 'avg_purchases_lag12', 'active_months_lag12']
```

```
# 检验特征是否划分完全
assert len(category_cols) + len(numeric_cols) == merchant.shape[1]
```

- 离散变量数据情况

  然后简单查看离散变量当前数据情况：

In [12…
```
# 查看分类变量的取值水平
merchant[category_cols].nunique()
```

Out[123]:
```
merchant_id                  334633
merchant_group_id            109391
merchant_category_id            324
subsector_id                     41
category_1                        2
most_recent_sales_range           5
most_recent_purchases_range       5
category_4                        2
city_id                         271
state_id                         25
category_2                        5
dtype: int64
```

In [12…
```
# 查看分类变量目前的类别
merchant[category_cols].dtypes
```

Out[122]:
```
merchant_id                  object
merchant_group_id             int64
merchant_category_id          int64
subsector_id                  int64
category_1                   object
most_recent_sales_range      object
most_recent_purchases_range  object
category_4                   object
city_id                       int64
state_id                      int64
category_2                  float64
dtype: object
```

In [14…
```
# 查看离散变量的缺失值情况
merchant[category_cols].isnull().sum()
```

Out[146]:
```
merchant_id                      0
merchant_group_id                0
merchant_category_id             0
subsector_id                     0
category_1                       0
most_recent_sales_range          0
most_recent_purchases_range      0
category_4                       0
city_id                          0
state_id                         0
category_2                   11887
dtype: int64
```

- 离散变量的缺失值标注

  注意到离散变量中的category_2存在较多缺失值，由于该分类变量取值水平为1-5，因此可以将缺失值先标注为-1，方便后续进行数据探索：

In [14…
```
merchant['category_2'].unique()
```

Out[148]:
```
array([ 1.,  5., nan,  2.,  3.,  4.])
```

In [15…
```
merchant['category_2'] = merchant['category_2'].fillna(-1)
```

- 离散变量字典编码

接下来对离散变量进行字典编码，即将object对象类型按照sort顺序进行数值化（整数）编码。例如原始category_1取值为Y/N，通过sort排序后N在Y之前，因此在重新编码时N取值会重编码为0、Y取值会重编码为1。以此类推。

需要注意的是，从严格角度来说，变量类型应该是有三类，分别是连续性变量、名义型变量以及有序变量。连续变量较好理解，所谓名义变量，指的是没有数值大小意义的分类变量，例如用1表示女、0表示男，0、1只是作为性别的指代，而没有1>0的含义。而所有有序变量，其也是离散型变量，但却有数值大小含义，如上述most_recent_purchases_range字段，销售等级中A>B>C>D>E，该离散变量的5个取值水平是有严格大小意义的，该变量就被称为有序变量。

在实际建模过程中，如果不需要提取有序变量的数值大小信息的话，可以考虑将其和名义变量一样进行独热编码。但本阶段初级预处理时暂时不考虑这些问题，先统一将object类型转化为数值型。

In [13…
```
# 字典编码函数
def change_object_cols(se):
    value = se.unique().tolist()
    value.sort()
    return se.map(pd.Series(range(len(value)), index=value)).values
```

简单测试函数效果：

In [14…
```
merchant['category_1']
```

Out[143]:
```
0         N
1         N
2         N
3         Y
4         Y
         ..
334691    N
334692    Y
334693    N
334694    Y
334695    N
Name: category_1, Length: 334696, dtype: object
```

In [14…
```
change_object_cols(merchant['category_1'])
```

Out[144]:
```
array([0, 0, 0, ..., 0, 1, 0], dtype=int64)
```

接下来，对merchant对象中的四个object类型列进行类别转化：

In [14…
```
for col in ['category_1', 'most_recent_sales_range', 'most_recent_purchases_range', '
    merchant[col] = change_object_cols(merchant[col])
```

- 连续变量的数据探索

接下来，我们继续探索连续变量

In [15…]:
```
# 查看连续变量的类别
merchant[numeric_cols].dtypes
```

Out[151]:
```
numerical_1            float64
numerical_2            float64
avg_sales_lag3         float64
avg_purchases_lag3     float64
active_months_lag3       int64
avg_sales_lag6         float64
avg_purchases_lag6     float64
active_months_lag6       int64
avg_sales_lag12        float64
avg_purchases_lag12    float64
active_months_lag12      int64
dtype: object
```

In [15…]:
```
# 连续变量的缺失值情况
merchant[numeric_cols].isnull().sum()
```

Out[155]:
```
numerical_1             0
numerical_2             0
avg_sales_lag3         13
avg_purchases_lag3      0
active_months_lag3      0
avg_sales_lag6         13
avg_purchases_lag6      0
active_months_lag6      0
avg_sales_lag12        13
avg_purchases_lag12     0
active_months_lag12     0
dtype: int64
```

In [15…]:
```
# 查看连续变量整体情况
merchant[numeric_cols].describe()
```

Out[156]:

	numerical_1	numerical_2	avg_sales_lag3	avg_purchases_lag3	active_months_lag3	avg_sa
count	334696.000000	334696.000000	334683.000000	3.346960e+05	334696.000000	3.346
mean	0.011476	0.008103	13.832993	inf	2.994108	2.165
std	1.098154	1.070497	2395.489999	NaN	0.095247	3.947
min	-0.057471	-0.057471	-82.130000	3.334953e-01	1.000000	-8.213
25%	-0.057471	-0.057471	0.880000	9.236499e-01	3.000000	8.50(
50%	-0.057471	-0.057471	1.000000	1.016667e+00	3.000000	1.010
75%	-0.047556	-0.047556	1.160000	1.146522e+00	3.000000	1.230
max	183.735111	182.079322	851844.640000	inf	3.000000	1.513

据此我们发现连续型变量中存在部分缺失值，并且部分连续变量还存在无穷值inf，需要对其进行简单处理。

- 无穷值处理

此处我们首先需要对无穷值进行处理。此处我们采用类似天花板盖帽法的方式对其进行修改，即将inf改为最大的显式数值。代码实现流程如下：

In [15…
```
inf_cols = ['avg_purchases_lag3', 'avg_purchases_lag6', 'avg_purchases_lag12']
merchant[inf_cols] = merchant[inf_cols].replace(np.inf, merchant[inf_cols].replace(np
```

In [16…
```
merchant[numeric_cols].describe()
```

Out[160]:

	numerical_1	numerical_2	avg_sales_lag3	avg_purchases_lag3	active_months_lag3	avg_sa
count	334696.000000	334696.000000	334683.000000	334696.000000	334696.000000	3.346
mean	0.011476	0.008103	13.832993	2.145143	2.994108	2.165
std	1.098154	1.070497	2395.489999	213.955844	0.095247	3.947
min	-0.057471	-0.057471	-82.130000	0.333495	1.000000	-8.213
25%	-0.057471	-0.057471	0.880000	0.923650	3.000000	8.50(
50%	-0.057471	-0.057471	1.000000	1.016667	3.000000	1.010
75%	-0.047556	-0.047556	1.160000	1.146522	3.000000	1.230
max	183.735111	182.079322	851844.640000	61851.333333	3.000000	1.513

- 缺失值处理

不同于无穷值的处理，缺失值处理方法有很多。但该数据集缺失数据较少，33万条数据中只有13条连续特征缺失值，此处我们先简单采用均值进行填补处理，后续若有需要再进行优化处理。

In [16…
```
for col in numeric_cols:
    merchant[col] = merchant[col].fillna(merchant[col].mean())
```

In [16…
```
merchant[numeric_cols].describe()
```

Out[162]:

	numerical_1	numerical_2	avg_sales_lag3	avg_purchases_lag3	active_months_lag3	avg_sa
count	334696.000000	334696.000000	334696.000000	334696.000000	334696.000000	3.346
mean	0.011476	0.008103	13.832993	2.145143	2.994108	2.165
std	1.098154	1.070497	2395.443476	213.955844	0.095247	3.947
min	-0.057471	-0.057471	-82.130000	0.333495	1.000000	-8.213
25%	-0.057471	-0.057471	0.880000	0.923650	3.000000	8.50(
50%	-0.057471	-0.057471	1.000000	1.016667	3.000000	1.010
75%	-0.047556	-0.047556	1.160000	1.146522	3.000000	1.230
max	183.735111	182.079322	851844.640000	61851.333333	3.000000	1.513

至此我们就完成了商户数据的预处理工作。

二、信用卡交易数据解读与探索

接下来对信用卡交易数据进行解读与探索。交易数据是本次竞赛中给出的规模最大、同时也

是信息量最大的数据集，在后续建模过程中将发挥至关重要的作用。

1.数据解读与验证

首先还是对数据集进行解释，以及简单验证数据集的正确性。信用卡交易记录包括了两个数据集，分别是historical_transactions和new_merchant_transactions。两个数据集字段类似，只是记录了不同时间区间的信用卡消费情况：

- historical_transactions：信用卡消费记录

该数据集记录了每张信用卡在特定商户中、三个月间的消费记录。该数据集数据规模较大，文件约有2.6G，并非必要建模字段，但若能从中提取有效信息，则能够更好的辅助建模。

In [16…
```
history_transaction = pd.read_csv('./eloData/historical_transactions.csv', header=0)
```

In [16…
```
history_transaction.head(5)
```

Out[164]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_categor
0	Y	C_ID_4e6213e9bc	88	N	0	A	
1	Y	C_ID_4e6213e9bc	88	N	0	A	
2	Y	C_ID_4e6213e9bc	88	N	0	A	
3	Y	C_ID_4e6213e9bc	88	N	0	A	
4	Y	C_ID_4e6213e9bc	88	N	0	A	

In [16…
```
history_transaction.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29112361 entries, 0 to 29112360
Data columns (total 14 columns):
 #   Column                Dtype
---  ------                -----
 0   authorized_flag       object
 1   card_id               object
 2   city_id               int64
 3   category_1            object
 4   installments          int64
 5   category_3            object
 6   merchant_category_id  int64
 7   merchant_id           object
 8   month_lag             int64
 9   purchase_amount       float64
 10  purchase_date         object
 11  category_2            float64
 12  state_id              int64
 13  subsector_id          int64
dtypes: float64(2), int64(6), object(6)
memory usage: 3.0+ GB
```

能够看到，数据集总共包括将近三千万条数据，总共有十四个字段，每个字段在数据字典中的解

释如下：

In [51]:
```
pd.read_excel('./eloData/Data Dictionary.xlsx', header=2, sheet_name='history')
```

Out[51]:

	Columns	Description
0	card_id	Card identifier
1	month_lag	month lag to reference date
2	purchase_date	Purchase date
3	authorized_flag	Y' if approved, 'N' if denied
4	category_3	anonymized category
5	installments	number of installments of purchase
6	category_1	anonymized category
7	merchant_category_id	Merchant category identifier (anonymized)
8	subsector_id	Merchant category group identifier (anonymized)
9	merchant_id	Merchant identifier (anonymized)
10	purchase_amount	Normalized purchase amount
11	city_id	City identifier (anonymized)
12	state_id	State identifier (anonymized)
13	category_2	anonymized category

实际含义如下：

字段	解释
card_id	第一无二的信用卡标志
authorized_flag	是否授权，Y/N
city_id	城市id，经过匿名处理
category_1	匿名特征，Y/N
installments	分期付款的次数
category_3	匿名类别特征，A/.../E
merchant_category_id	商户类别，匿名特征
merchant_id	商户id
month_lag	距离2018年月的2月数差
purchase_amount	标准化后的付款金额
purchase_date	付款时间
category_2	匿名类别特征2
state_id	州id，经过匿名处理
subsector_id	商户类别特征

- new_merchant_transactions：信用卡近期的交易信息

  信用卡在2018年2月之后的交易信息，和historical_transactions字段完全一致。

In [17…
```
new_transaction = pd.read_csv('./eloData/new_merchant_transactions.csv', header=0)
```

In [16…
```
new_transaction.head(5)
```

Out[167]:

	authorized_flag	card_id	city_id	category_1	installments	category_3	merchant_categor
0	Y	C_ID_415bb3a509	107	N	1	B	
1	Y	C_ID_415bb3a509	140	N	1	B	
2	Y	C_ID_415bb3a509	330	N	1	B	
3	Y	C_ID_415bb3a509	-1	Y	1	B	
4	Y	C_ID_ef55cf8d4b	-1	Y	1	B	

In [55]:
```
pd.read_csv('./eloData/new_merchant_transactions.csv', header=0).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1963031 entries, 0 to 1963030
Data columns (total 14 columns):
 #   Column                Dtype  
---  ------                -----  
 0   authorized_flag       object 
 1   card_id               object 
 2   city_id               int64  
 3   category_1            object 
 4   installments          int64  
 5   category_3            object 
 6   merchant_category_id  int64  
 7   merchant_id           object 
 8   month_lag             int64  
 9   purchase_amount       float64
 10  purchase_date         object 
 11  category_2            float64
 12  state_id              int64  
 13  subsector_id          int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 209.7+ MB
```

该数据中总共有将近200万条数据。并且我们发现，该数据集中有较多字段和商家数据merchant重复，我们可以对其进行简单检验。

- 对比merchant数据集

首先简单查看有哪些字段一致：

In [16…
```
duplicate_cols = []

for col in merchant.columns:
    if col in new_transaction.columns:
        duplicate_cols.append(col)

print(duplicate_cols)
```

```
['merchant_id', 'merchant_category_id', 'subsector_id', 'category_1', 'city_id', 'state_id', 'category_2']
```

并且我们进一步发现，交易记录中的merhcant_id信息并不唯一：

In [16…
```
# 取出和商户数据表重复字段并去重
new_transaction[duplicate_cols].drop_duplicates().shape
```

Out[169]: (291242, 7)

In [17…
```
# 商户id去重
new_transaction['merchant_id'].nunique()
```

Out[170]: 226129

造成该现象的原因可能是商铺在逐渐经营过程动态变化，而基于此，在后续的建模过程中，我们将优先使用交易记录中表中的相应记录。

2.数据预处理

接下来对交易数据进行预处理。

- 连续/离散字段标注

首先也是一样，需要对其连续/离散变量进行标注。当然该数据集中比较特殊的一点，是存在一个时间列，我们将其单独归为一类：

In [17…
```
numeric_cols = ['installments', 'month_lag', 'purchase_amount']
category_cols = ['authorized_flag', 'card_id', 'city_id', 'category_1',
       'category_3', 'merchant_category_id', 'merchant_id', 'category_2', 'state_id',
       'subsector_id']
time_cols = ['purchase_date']

assert len(numeric_cols) + len(category_cols) + len(time_cols) == new_transaction.sh
```

- 字段类型转化/缺失值填补

然后简单查看离散变量当前数据情况：

In [18…
```
# 查看分类变量的类别
new_transaction[category_cols].dtypes
```

Out[180]:
```
authorized_flag          object
card_id                  object
city_id                   int64
category_1               object
category_3               object
merchant_category_id      int64
merchant_id              object
category_2              float64
state_id                  int64
subsector_id              int64
dtype: object
```

In [18…
```
new_transaction[category_cols].isnull().sum()
```

Out[181]:
```
authorized_flag             0
card_id                     0
```

```
city_id                    0
category_1                 0
category_3             55922
merchant_category_id       0
merchant_id            26216
category_2            111745
state_id                   0
subsector_id               0
dtype: int64
```

和此前的merchant处理类似，我们对其object类型对象进行字典编码（id除外），并对利用-1对缺失值进行填补：

In [17…
```
for col in ['authorized_flag', 'category_1', 'category_3']:
    new_transaction[col] = change_object_cols(new_transaction[col].fillna(-1).astype(

new_transaction[category_cols] = new_transaction[category_cols].fillna(-1)
```

In [17…
```
new_transaction[category_cols].dtypes
```

Out[177]:
```
authorized_flag           int64
card_id                  object
city_id                   int64
category_1                int64
category_3                int64
merchant_category_id      int64
merchant_id              object
category_2              float64
state_id                  int64
subsector_id              int64
dtype: object
```

至此，我们就完成了几张表的数据预处理工作。

三、数据清洗后数据生成

1.回顾商户数据、交易数据清洗流程

当然，由于上述工作较为繁琐，我们简单总结上述针对商户数据和交易数据的完整步骤如下：

商户数据merchants.csv

- 划分连续字段和离散字段；
- 对字符型离散字段进行字典排序编码；
- 对缺失值处理，此处统一使用-1进行缺失值填充，本质上是一种标注；
- 对连续性字段的无穷值进行处理，用该列的最大值进行替换；
- 去除重复数据；

交易数据new_merchant_transactions.csv和historical_transactions.csv

- 划分字段类型，分为离散字段、连续字段和时间字段；
- 和商户数据的处理方法一样，对字符型离散字段进行字典排序，对缺失值进行统一填充；
- 对新生成的购买欲分离散字段进行字典排序编码；
- 最后对多表进行拼接，并且通过month_lag字段是否大于0来进行区分。

2.创建清洗后数据

结合训练集和测试集的清洗流程，我们可以在此统一执行所有数据的数据清洗工作，并将其最终保存为本地文件，方便后续特征工程及算法建模过程使用，其流程如下：

- 读取数据

In [1]:
```
import gc
import time
import numpy as np
import pandas as pd
from datetime import datetime
```

In [2]:
```
train = pd.read_csv('data/train.csv')
test =  pd.read_csv('data/test.csv')
merchant = pd.read_csv('data/merchants.csv')
new_transaction = pd.read_csv('data/new_merchant_transactions.csv')
history_transaction = pd.read_csv('data/historical_transactions.csv')
```

In [3]:
```
# 字典编码函数
def change_object_cols(se):
    value = se.unique().tolist()
    value.sort()
    return se.map(pd.Series(range(len(value)), index=value)).values
```

- 训练集/测试集的数据预处理

In [4]:
```
# 对首次活跃月份进行编码
se_map = change_object_cols(train['first_active_month'].append(test['first_active_mon
train['first_active_month'] = se_map[:train.shape[0]]
test['first_active_month'] = se_map[train.shape[0]:]
```

- 测试集/训练集导出与内存清理

In [5]:
```
train.to_csv("preprocess/train_pre.csv", index=False)
test.to_csv("preprocess/test_pre.csv", index=False)
```

In [6]:
```
del train
del test
gc.collect()
```

Out[6]: 17

- 商户信息预处理

In [7]:
```
# 1、根据业务含义划分离散字段category_cols与连续字段numeric_cols。
category_cols = ['merchant_id', 'merchant_group_id', 'merchant_category_id',
       'subsector_id', 'category_1',
       'most_recent_sales_range', 'most_recent_purchases_range',
       'category_4', 'city_id', 'state_id', 'category_2']
numeric_cols = ['numerical_1', 'numerical_2',
     'avg_sales_lag3', 'avg_purchases_lag3', 'active_months_lag3',
       'avg_sales_lag6', 'avg_purchases_lag6', 'active_months_lag6',
```

```
       'avg_sales_lag12', 'avg_purchases_lag12', 'active_months_lag12']

# 2、对非数值型的离散字段进行字典排序编码。
for col in ['category_1', 'most_recent_sales_range', 'most_recent_purchases_range', '
    merchant[col] = change_object_cols(merchant[col])

# 3、为了能够更方便统计，进行缺失值的处理，对离散字段统一用-1进行填充。
merchant[category_cols] = merchant[category_cols].fillna(-1)


# 4、对离散型字段探查发现有正无穷值，这是特征提取以及模型所不能接受的，因此需要对无限值
inf_cols = ['avg_purchases_lag3', 'avg_purchases_lag6', 'avg_purchases_lag12']
merchant[inf_cols] = merchant[inf_cols].replace(np.inf, merchant[inf_cols].replace(np

# 5、平均值进行填充，后续有需要再进行优化处理。
for col in numeric_cols:
    merchant[col] = merchant[col].fillna(merchant[col].mean())

# 6、去除与transaction交易记录表格重复的列，以及merchant_id的重复记录。
duplicate_cols = ['merchant_id', 'merchant_category_id', 'subsector_id', 'category_1',
merchant = merchant.drop(duplicate_cols[1:], axis=1)
merchant = merchant.loc[merchant['merchant_id'].drop_duplicates().index.tolist()].res
```

与处理完后先不着急导出或删除，后续需要和交易数据进行拼接。

- 交易数据预处理

In [8]:

```
# 1、为了统一处理，首先拼接new和history两张表格，后续可以month_lag>=0进行区分。
transaction = pd.concat([new_transaction, history_transaction], axis=0, ignore_index=
del new_transaction
del history_transaction
gc.collect()

# 2、同样划分离散字段、连续字段以及时间字段。
numeric_cols = [ 'installments', 'month_lag', 'purchase_amount']
category_cols = ['authorized_flag', 'card_id', 'city_id', 'category_1',
       'category_3', 'merchant_category_id', 'merchant_id', 'category_2', 'state_id',
       'subsector_id']
time_cols = ['purchase_date']

# 3、可仿照merchant的处理方式对字符型的离散特征进行字典序编码以及缺失值填充。
for col in ['authorized_flag', 'category_1', 'category_3']:
    transaction[col] = change_object_cols(transaction[col].fillna(-1).astype(str))
transaction[category_cols] = transaction[category_cols].fillna(-1)
transaction['category_2'] = transaction['category_2'].astype(int)

# 4、进行时间段的处理，简单起见进行月份、日期的星期数（工作日与周末）、以及
# 时间段（上午、下午、晚上、凌晨）的信息提取。
transaction['purchase_month'] = transaction['purchase_date'].apply(lambda x:'-'.join
transaction['purchase_hour_section'] = transaction['purchase_date'].apply(lambda x: x
transaction['purchase_day'] = transaction['purchase_date'].apply(lambda x: datetime.s
del transaction['purchase_date']


# 5、对新生成的购买月份离散字段进行字典序编码。
transaction['purchase_month'] = change_object_cols(transaction['purchase_month'].filln
```

完成交易数据预处理后，即可进行交易数据和商铺数据的表格合并。

- 表格合并

在合并的过程中，有两种处理方案，其一是对缺失值进行-1填补，然后将所有离散型字段化

为字符串类型（为了后续字典合并做准备），其二则是新增两列，分别是purchase_day_diff和purchase_month_diff，其数据为交易数据以card_id进行groupby并最终提取出purchase_day/month并进行差分的结果。

方案一代码如下：

In [9]:
```
# 为了方便特征的统一计算将其merge合并，重新划分相应字段种类。
cols = ['merchant_id', 'most_recent_sales_range', 'most_recent_purchases_range', 'cate
transaction = pd.merge(transaction, merchant[cols], how='left', on='merchant_id')

numeric_cols = ['purchase_amount', 'installments']

category_cols = ['authorized_flag', 'city_id', 'category_1',
       'category_3', 'merchant_category_id','month_lag','most_recent_sales_range',
                'most_recent_purchases_range', 'category_4',
                'purchase_month', 'purchase_hour_section', 'purchase_day']

id_cols = ['card_id', 'merchant_id']

transaction[cols[1:]] = transaction[cols[1:]].fillna(-1).astype(int)
transaction[category_cols] =transaction[category_cols].fillna(-1).astype(str)
```

随后将其导出为transaction_d_pre.csv

In [10]:
```
transaction.to_csv("preprocess/transaction_d_pre.csv", index=False)
```

In [11]:
```
del transaction
gc.collect()
```

Out[11]: 17

方案二代码如下：

In [12]:
```
merchant = pd.read_csv('data/merchants.csv')
new_transaction = pd.read_csv('data/new_merchant_transactions.csv')
history_transaction = pd.read_csv('data/historical_transactions.csv')
```

In [13]:
```
# 1、根据业务含义划分离散字段category_cols与连续字段numeric_cols。
category_cols = ['merchant_id', 'merchant_group_id', 'merchant_category_id',
       'subsector_id', 'category_1',
       'most_recent_sales_range', 'most_recent_purchases_range',
       'category_4', 'city_id', 'state_id', 'category_2']
numeric_cols = ['numerical_1', 'numerical_2',
     'avg_sales_lag3', 'avg_purchases_lag3', 'active_months_lag3',
       'avg_sales_lag6', 'avg_purchases_lag6', 'active_months_lag6',
       'avg_sales_lag12', 'avg_purchases_lag12', 'active_months_lag12']

# 2、对非数值型的离散字段进行字典排序编码。
for col in ['category_1', 'most_recent_sales_range', 'most_recent_purchases_range', '
    merchant[col] = change_object_cols(merchant[col])

# 3、为了能够更方便统计，进行缺失值的处理，对离散字段统一用-1进行填充。
merchant[category_cols] = merchant[category_cols].fillna(-1)


# 4、对离散型字段探查发现有正无穷值，这是特征提取以及模型所不能接受的，因此需要对无限值
inf_cols = ['avg_purchases_lag3', 'avg_purchases_lag6', 'avg_purchases_lag12']
merchant[inf_cols] = merchant[inf_cols].replace(np.inf, merchant[inf_cols].replace(np
```

```
# 5、平均值进行填充，后续有需要再进行优化处理。
for col in numeric_cols:
    merchant[col] = merchant[col].fillna(merchant[col].mean())

# 6、去除与transaction交易记录表格重复的列，以及merchant_id的重复记录。
duplicate_cols = ['merchant_id', 'merchant_category_id', 'subsector_id', 'category_1',
merchant = merchant.drop(duplicate_cols[1:], axis=1)
merchant = merchant.loc[merchant['merchant_id'].drop_duplicates().index.tolist()].res
```

In [14]:

```
# 1、为了统一处理，首先拼接new和history两张表格，后续可以month_lag>=0进行区分。
transaction = pd.concat([new_transaction, history_transaction], axis=0, ignore_index=
del new_transaction
del history_transaction
gc.collect()

# 2、同样划分离散字段、连续字段以及时间字段。
numeric_cols = [ 'installments', 'month_lag', 'purchase_amount']
category_cols = ['authorized_flag', 'card_id', 'city_id', 'category_1',
       'category_3', 'merchant_category_id', 'merchant_id', 'category_2', 'state_id',
       'subsector_id']
time_cols = ['purchase_date']

# 3、可仿照merchant的处理方式对字符型的离散特征进行字典序编码以及缺失值填充。
for col in ['authorized_flag', 'category_1', 'category_3']:
    transaction[col] = change_object_cols(transaction[col].fillna(-1).astype(str))
transaction[category_cols] = transaction[category_cols].fillna(-1)
transaction['category_2'] = transaction['category_2'].astype(int)

# 4、进行时间段的处理，简单起见进行月份、日期的星期数（工作日与周末）、以及
# 时间段（上午、下午、晚上、凌晨）的信息提取。
transaction['purchase_month'] = transaction['purchase_date'].apply(lambda x:'-'.join
transaction['purchase_hour_section'] = transaction['purchase_date'].apply(lambda x: x
transaction['purchase_day'] = transaction['purchase_date'].apply(lambda x: datetime.s
del transaction['purchase_date']


# 5、对新生成的购买月份离散字段进行字典序编码。
transaction['purchase_month'] = change_object_cols(transaction['purchase_month'].filln
```

In [15]:

```
cols = ['merchant_id', 'most_recent_sales_range', 'most_recent_purchases_range', 'cate
transaction = pd.merge(transaction, merchant[cols], how='left', on='merchant_id')

numeric_cols = ['purchase_amount', 'installments']

category_cols = ['authorized_flag', 'city_id', 'category_1',
       'category_3', 'merchant_category_id','month_lag','most_recent_sales_range',
                 'most_recent_purchases_range', 'category_4',
                 'purchase_month', 'purchase_hour_section', 'purchase_day']

id_cols = ['card_id', 'merchant_id']

transaction['purchase_day_diff'] = transaction.groupby("card_id")['purchase_day'].dif
transaction['purchase_month_diff'] = transaction.groupby("card_id")['purchase_month'].
```

随后将其导出为transaction_g_pre.csv

In [16]:

```
transaction.to_csv("preprocess/transaction_g_pre.csv", index=False)
```

In [17]:

```
del transaction
```

```
gc.collect()
```

Out[17]: 17

在导出完成这两张表之后，接下来我们将借助这些数据来进一步致性特征工程和算法建模。