Ran SONG - Data exploration and data cleaning of transaction data

January 19, 2022

0.0.1 1. Data Cleaning

[142]:
```
merchant = pd.read_csv('./eloData/merchants.csv', header=0)
```

[110]:
```
merchant.head(5)
```

[110]:
```
        merchant_id  merchant_group_id  merchant_category_id  subsector_id  \
0  M_ID_838061e48c               8353                   792             9
1  M_ID_9339d880ad               3184                   840            20
2  M_ID_e726bbae1e                447                   690             1
3  M_ID_a70e9c5f81               5026                   792             9
4  M_ID_64456c37ce               2228                   222            21

   numerical_1  numerical_2 category_1 most_recent_sales_range  \
0    -0.057471    -0.057471          N                       E
1    -0.057471    -0.057471          N                       E
2    -0.057471    -0.057471          N                       E
3    -0.057471    -0.057471          Y                       E
4    -0.057471    -0.057471          Y                       E

  most_recent_purchases_range  avg_sales_lag3  …  avg_sales_lag6  \
0                           E           -0.40  …           -2.25
1                           E           -0.72  …           -0.74
2                           E          -82.13  …          -82.13
3                           E             NaN  …             NaN
4                           E             NaN  …             NaN

   avg_purchases_lag6  active_months_lag6  avg_sales_lag12  \
0           18.666667                   6            -2.32
1            1.291667                   6            -0.57
2          260.000000                   2           -82.13
3            4.666667                   6              NaN
4            0.361111                   6              NaN

   avg_purchases_lag12  active_months_lag12  category_4  city_id state_id  \
0            13.916667                   12           N      242        9
```

```
1           1.687500                   12           N       22       16
2         260.000000                    2           N       -1        5
3           3.833333                   12           Y       -1       -1
4           0.347222                   12           Y       -1       -1

   category_2
0         1.0
1         1.0
2         5.0
3         NaN
4         NaN

[5 rows x 22 columns]
```

[111]:
```
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
```

[114]:
```
df = pd.read_excel('./eloData/Data_Dictionary.xlsx', header=2,␣
↪sheet_name='merchant')
df
```

[114]:
```
                        Columns  \
0                   merchant_id
1             merchant_group_id
2          merchant_category_id
3                  subsector_id
4                   numerical_1
5                   numerical_2
6                    category_1
7       most_recent_sales_range
8   most_recent_purchases_range
9                avg_sales_lag3
10           avg_purchases_lag3
11           active_months_lag3
12               avg_sales_lag6
13           avg_purchases_lag6
14           active_months_lag6
15              avg_sales_lag12
16          avg_purchases_lag12
17          active_months_lag12
18                   category_4
19                      city_id
20                     state_id
21                   category_2

                                          Description
0                          Unique merchant identifier
1                        Merchant group (anonymized )
2   Unique identifier for merchant category (anony…
3               Merchant category group (anonymized )
4                                 anonymized measure
5                                 anonymized measure
6                                anonymized category
7   Range of revenue (monetary units) in last acti…
8   Range of quantity of transactions in last acti…
9   Monthly average of revenue in last 3 months di…
10  Monthly average of transactions in last 3 mont…
11     Quantity of active months within last 3 months
12  Monthly average of revenue in last 6 months di…
13  Monthly average of transactions in last 6 mont…
14     Quantity of active months within last 6 months
15  Monthly average of revenue in last 12 months d…
16  Monthly average of transactions in last 12 mon…
17    Quantity of active months within last 12 months
```

```
18                                  anonymized category
19                        City identifier (anonymized )
20                       State identifier (anonymized )
21                                  anonymized category
```

It can be found that the data table provides not only the basic attribute fields of the merchant (such as category and commodity group, etc.), but also the recent transactions of the merchant. Howevr, there is still plenty of anonymity.

0.0.2 2. Data Exploration

- Correctness

[118]:
```python
print(merchant.shape, merchant['merchant_id'].nunique())
```

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

- Missing values

[120]:
```python
merchant.isnull().sum()
```

[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
```

It can be found that there are many missing values in the second anonymous categorical variable, and the number of missing values in `avg_sales_lag3/6/12` is the same, it is very likely that there are 13 merchants who have confirmed these three aspects of information at the same time.

0.0.3 3. Preprocessing

- Discrete/Continuous Field Labeling

[3]:
```
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']

assert len(category_cols) + len(numeric_cols) == merchant.shape[1]  # check for␣
 ↪completeness
```

[123]:
```
merchant[category_cols].nunique()
```

[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
```

[122]:
```
merchant[category_cols].dtypes
```

[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
```

[146]:
```
merchant[category_cols].isnull().sum()
```

[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
```

- Missing Value Labeling for Discrete Variables

Note that there are many missing values in `category_2` in discrete variables. Since the value level of this categorical variable is `1-5`, the missing value can be marked as `-1` first to facilitate subsequent data exploration:

[148]:
```
merchant['category_2'].unique()
```

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

[150]:
```
merchant['category_2'] = merchant['category_2'].fillna(-1)
```

- Discrete Variable Dictionary Encoding

Next, perform dictionary encoding on discrete variables, that is, the object object type is numerically (integer) encoded in the sort order. For example, the original category_1 value is `Y/N`. After sorting by sort, `N` is before `Y`. Therefore, the value of `N` will be recoded to `0` and the value of `Y` will be recoded to `1` during recoding. And so on.

It is worth noting that there should be three types of variable types, namely continuous variables, nominal variables and ordinal variables. Continuous variables are easy to understand. The so-called nominal variables refer to categorical variables that have no numerical significance. For example, 1 means female, `0` means male, `0` and `1` are only used as gender references, and there is no meaning of `1>0`. All ordinal variables are also discrete variables, but they have the meaning of numerical value. For example, in the most_recent_purchases_range field above, `A>B>C>D>E` in the sales level, the five value levels of the discrete variable have strict sizes Meaning, the variable is called an ordinal variable.

[4]:
```
# Dictionary encoding function
def change_object_cols(se):
    value = se.unique().tolist()
    value.sort()
    return se.map(pd.Series(range(len(value)), index=value)).values
```

[143]:
```
merchant['category_1']
```

[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
```

[144]:
```
change_object_cols(merchant['category_1'])
```

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

Transform 4 objects in `merchant`:

[145]:
```
for col in ['category_1', 'most_recent_sales_range',␣
↪'most_recent_purchases_range', 'category_4']:
    merchant[col] = change_object_cols(merchant[col])
```

- Data Exploration for Continuous Variables

[151]:
```
merchant[numeric_cols].dtypes
```

[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
```

[155]:
```
merchant[numeric_cols].isnull().sum()
```

[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
```

[156]:
```
merchant[numeric_cols].describe()
```

[156]:
```
         numerical_1    numerical_2  avg_sales_lag3  avg_purchases_lag3  \
count  334696.000000  334696.000000   334683.000000        3.346960e+05
mean        0.011476       0.008103       13.832993                 inf
std         1.098154       1.070497     2395.489999                 NaN
min        -0.057471      -0.057471      -82.130000        3.334953e-01
25%        -0.057471      -0.057471        0.880000        9.236499e-01
50%        -0.057471      -0.057471        1.000000        1.016667e+00
75%        -0.047556      -0.047556        1.160000        1.146522e+00
max       183.735111     182.079322   851844.640000                 inf

       active_months_lag3  avg_sales_lag6  avg_purchases_lag6  \
count       334696.000000    3.346830e+05        3.346960e+05
mean             2.994108    2.165079e+01                 inf
std              0.095247    3.947108e+03                 NaN
min              1.000000   -8.213000e+01        1.670447e-01
25%              3.000000    8.500000e-01        9.022475e-01
50%              3.000000    1.010000e+00        1.026961e+00
75%              3.000000    1.230000e+00        1.215575e+00
max              3.000000    1.513959e+06                 inf

       active_months_lag6  avg_sales_lag12  avg_purchases_lag12  \
count       334696.000000     3.346830e+05         3.346960e+05
mean             5.947397     2.522771e+01                  inf
std              0.394936     5.251842e+03                  NaN
min              1.000000    -8.213000e+01         9.832954e-02
25%              6.000000     8.500000e-01         8.983333e-01
50%              6.000000     1.020000e+00         1.043361e+00
75%              6.000000     1.290000e+00         1.266480e+00
max              6.000000     2.567408e+06                  inf

       active_months_lag12
count        334696.000000
mean             11.599335
std               1.520138
min               1.000000
25%              12.000000
50%              12.000000
75%              12.000000
```

```
max               12.000000
```

- `INF` values

Make `INF` the largest value.

[159]:
```python
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.inf, -99).max().max())
```

[160]:
```python
merchant[numeric_cols].describe()
```

[160]:
```
         numerical_1    numerical_2  avg_sales_lag3  avg_purchases_lag3  \
count  334696.000000  334696.000000   334683.000000       334696.000000
mean        0.011476       0.008103       13.832993            2.145143
std         1.098154       1.070497     2395.489999          213.955844
min        -0.057471      -0.057471      -82.130000            0.333495
25%        -0.057471      -0.057471        0.880000            0.923650
50%        -0.057471      -0.057471        1.000000            1.016667
75%        -0.047556      -0.047556        1.160000            1.146522
max       183.735111     182.079322   851844.640000        61851.333333

       active_months_lag3  avg_sales_lag6  avg_purchases_lag6  \
count       334696.000000    3.346830e+05       334696.000000
mean             2.994108    2.165079e+01            2.441947
std              0.095247    3.947108e+03          209.439373
min              1.000000   -8.213000e+01            0.167045
25%              3.000000    8.500000e-01            0.902247
50%              3.000000    1.010000e+00            1.026961
75%              3.000000    1.230000e+00            1.215575
max              3.000000    1.513959e+06        61851.333333

       active_months_lag6  avg_sales_lag12  avg_purchases_lag12  \
count       334696.000000     3.346830e+05        334696.000000
mean             5.947397     2.522771e+01             2.633572
std              0.394936     5.251842e+03           205.206198
min              1.000000    -8.213000e+01             0.098330
25%              6.000000     8.500000e-01             0.898333
50%              6.000000     1.020000e+00             1.043361
75%              6.000000     1.290000e+00             1.266480
max              6.000000     2.567408e+06         61851.333333

       active_months_lag12
count        334696.000000
mean             11.599335
std               1.520138
min               1.000000
25%              12.000000
```

```
50%               12.000000
75%               12.000000
max               12.000000
```

- Missing values

33 13

[161]:
```python
for col in numeric_cols:
    merchant[col] = merchant[col].fillna(merchant[col].mean())
```

[162]:
```python
merchant[numeric_cols].describe()
```

[162]:
```
         numerical_1    numerical_2  avg_sales_lag3  avg_purchases_lag3  \
count  334696.000000  334696.000000   334696.000000       334696.000000
mean        0.011476       0.008103       13.832993            2.145143
std         1.098154       1.070497     2395.443476          213.955844
min        -0.057471      -0.057471      -82.130000            0.333495
25%        -0.057471      -0.057471        0.880000            0.923650
50%        -0.057471      -0.057471        1.000000            1.016667
75%        -0.047556      -0.047556        1.160000            1.146522
max       183.735111     182.079322   851844.640000        61851.333333

       active_months_lag3  avg_sales_lag6  avg_purchases_lag6  \
count       334696.000000    3.346960e+05       334696.000000
mean             2.994108    2.165079e+01            2.441947
std              0.095247    3.947031e+03          209.439373
min              1.000000   -8.213000e+01            0.167045
25%              3.000000    8.500000e-01            0.902247
50%              3.000000    1.010000e+00            1.026961
75%              3.000000    1.230000e+00            1.215575
max              3.000000    1.513959e+06        61851.333333

       active_months_lag6  avg_sales_lag12  avg_purchases_lag12  \
count       334696.000000     3.346960e+05        334696.000000
mean             5.947397     2.522771e+01             2.633572
std              0.394936     5.251740e+03           205.206198
min              1.000000    -8.213000e+01             0.098330
25%              6.000000     8.500000e-01             0.898333
50%              6.000000     1.020000e+00             1.043361
75%              6.000000     1.290000e+00             1.266480
max              6.000000     2.567408e+06         61851.333333

       active_months_lag12
count        334696.000000
mean             11.599335
std               1.520138
min               1.000000
```

```
25%              12.000000
50%              12.000000
75%              12.000000
max              12.000000
```

Now the preprocessing of data is done.

0.1 2. Data Exploration

Next, we will interpret and explore the credit card transaction data. Transaction data is the largest and most informative dataset given in this competition, and will play a crucial role in the subsequent modeling process.

0.1.1 1. Data Preprocessing & Validation

- `historical_transactions`

This dataset records the spending records of each credit card at a specific merchant over a three-month period. The data size of this dataset is large, and the file is about `2.6G`. It is not necessary to model fields, but if effective information can be extracted from it, it can better assist modeling.

[163]:
```
history_transaction = pd.read_csv('./eloData/historical_transactions.csv',␣
↪header=0)
```

[164]:
```
history_transaction.head(5)
```

[164]:
```
  authorized_flag          card_id  city_id category_1  installments  \
0               Y  C_ID_4e6213e9bc       88          N             0
1               Y  C_ID_4e6213e9bc       88          N             0
2               Y  C_ID_4e6213e9bc       88          N             0
3               Y  C_ID_4e6213e9bc       88          N             0
4               Y  C_ID_4e6213e9bc       88          N             0

  category_3  merchant_category_id      merchant_id  month_lag  \
0          A                    80  M_ID_e020e9b302         -8
1          A                   367  M_ID_86ec983688         -7
2          A                    80  M_ID_979ed661fc         -6
3          A                   560  M_ID_e6d5ae8ea6         -5
4          A                    80  M_ID_e020e9b302        -11

   purchase_amount        purchase_date  category_2  state_id  subsector_id
0        -0.703331  2017-06-25 15:33:07         1.0        16            37
1        -0.733128  2017-07-15 12:10:45         1.0        16            16
2        -0.720386  2017-08-09 22:04:29         1.0        16            37
3        -0.735352  2017-09-02 10:06:26         1.0        16            34
4        -0.722865  2017-03-10 01:14:19         1.0        16            37
```

[165]:
```
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
```

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

[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

- new_merchant_transactions

[179]:
```python
new_transaction = pd.read_csv('./eloData/new_merchant_transactions.csv',␣
↪header=0)
```

[167]:
```python
new_transaction.head(5)
```

[167]:
```
  authorized_flag          card_id  city_id category_1  installments  \
0               Y  C_ID_415bb3a509      107          N             1
1               Y  C_ID_415bb3a509      140          N             1
2               Y  C_ID_415bb3a509      330          N             1
3               Y  C_ID_415bb3a509       -1          Y             1
4               Y  C_ID_ef55cf8d4b       -1          Y             1

  category_3  merchant_category_id      merchant_id  month_lag  \
0          B                   307  M_ID_b0c793002c          1
1          B                   307  M_ID_88920c89e8          1
2          B                   507  M_ID_ad5237ef6b          2
3          B                   661  M_ID_9e84cda3b1          1
4          B                   166  M_ID_3c86fa3831          1

   purchase_amount        purchase_date  category_2  state_id  subsector_id
0        -0.557574  2018-03-11 14:57:36         1.0         9            19
1        -0.569580  2018-03-19 18:53:37         1.0         9            19
2        -0.551037  2018-04-26 14:08:44         1.0         9            14
3        -0.671925  2018-03-07 09:43:21         NaN        -1             8
4        -0.659904  2018-03-22 21:07:53         NaN        -1            29
```

[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
```

- Compare with `merchant`

[168]:
```
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']
```

[169]:
```
new_transaction[duplicate_cols].drop_duplicates().shape
```

[169]: (291242, 7)

[170]:
```
new_transaction['merchant_id'].nunique()
```

[170]: 226129

0.1.2 2. Preprocessing

- Tag contunious/discrete vars

[6]:
```
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.shape[1]
```

[180]:
```
new_transaction[category_cols].dtypes
```

[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
```

[181]:
```
new_transaction[category_cols].isnull().sum()
```

[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
```

[176]:
```
for col in ['authorized_flag', 'category_1', 'category_3']:
    new_transaction[col] = change_object_cols(new_transaction[col].fillna(-1).
 ↪astype(str))

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

[177]:
```
new_transaction[category_cols].dtypes
```

[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
```

0.2 3. Generate Data

0.2.1 1.

0.2.2 merchants.csv

- divide continuous fields and discrete fields;
- lexicographical encoding for character discrete fields;
- For missing value processing, -1 is used to fill missing values here, which is essentially a label;
- Process the infinite value of the continuous field and replace it with the maximum value of the column;
- remove duplicate data;

0.2.3 `new_merchant_transactions.csv` & `historical_transactions.csv`

- Divide field types into discrete fields, continuous fields and time fields;
- The same as the processing method of business data, lexicographical sorting of character discrete fields, and uniform filling of missing values;
- Lexicographic sorting and encoding of the newly generated discrete fields for purchase;
- Finally, splicing multiple tables and distinguishing them by whether the `month_lag` field is greater than 0.

0.2.4 2. Cleaning dataset

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

[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')
```

[3]:
```
def change_object_cols(se):
    value = se.unique().tolist()
    value.sort()
    return se.map(pd.Series(range(len(value)), index=value)).values
```

- Preprocessing for Training/Validation

[4]:
```
se_map = change_object_cols(train['first_active_month'].
 ↪append(test['first_active_month']).astype(str))
train['first_active_month'] = se_map[:train.shape[0]]
test['first_active_month'] = se_map[train.shape[0]:]
```

- Generate & Export

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

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

[6]: 17

[9]:
```
# 1. Divide discrete field category_cols and continuous field numeric_cols␣
 ↪according to business meaning
```

```
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. lexicographical encoding for non-numeric discrete fields
for col in ['category_1', 'most_recent_sales_range',␣
 ↪'most_recent_purchases_range', 'category_4']:
    merchant[col] = change_object_cols(merchant[col])

# 3. In order to make statistics more convenient and deal with missing values,
# the discrete fields are uniformly filled with -1
merchant[category_cols] = merchant[category_cols].fillna(-1)


# 4. It is found that there are positive infinite values for discrete field␣
 ↪exploration,
# which is unacceptable for feature extraction and models,
# so infinite values need to be processed, and the maximum value is used here␣
 ↪to replace
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.inf, -99).max().max())

# 5. Fill the average
for col in numeric_cols:
    merchant[col] = merchant[col].fillna(merchant[col].mean())

# 6. Remove duplicates
duplicate_cols = ['merchant_id', 'merchant_category_id', 'subsector_id',␣
 ↪'category_1', 'city_id', 'state_id', 'category_2']
merchant = merchant.drop(duplicate_cols[1:], axis=1)
merchant = merchant.loc[merchant['merchant_id'].drop_duplicates().index.
 ↪tolist()].reset_index(drop=True)
```

- Preprocessing Transaction Data

[8]:
```
transaction = pd.concat([new_transaction, history_transaction], axis=0,␣
 ↪ignore_index=True)
del new_transaction
del history_transaction
gc.collect()
```

```
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']

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)

transaction['purchase_month'] = transaction['purchase_date'].apply(lambda x:'-'.
 ↪join(x.split(' ')[0].split('-')[:2]))
transaction['purchase_hour_section'] = transaction['purchase_date'].
 ↪apply(lambda x: x.split(' ')[1].split(':')[0]).astype(int)//6
transaction['purchase_day'] = transaction['purchase_date'].apply(lambda x:␣
 ↪datetime.strptime(x.split(" ")[0], "%Y-%m-%d").weekday())//5
del transaction['purchase_date']

transaction['purchase_month'] =␣
 ↪change_object_cols(transaction['purchase_month'].fillna(-1).astype(str))
```

- Merge tables

In the process of merging, there are two processing solutions, one is to fill the missing values with `-1`, and then all discrete fields are converted into string types (for subsequent dictionary merging), the other is to Two new columns are added, namely `purchase_day_diff` and `purchase_month_diff`. The data is the transaction data groupby with card_id, and finally the `purchase_day/month` is extracted and differentiated.

[9]:
```
cols = ['merchant_id', 'most_recent_sales_range',␣
 ↪'most_recent_purchases_range', 'category_4']
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)
```

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

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

[11]: 17

Option 2:

[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')
```

[13]:
```
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']

for col in ['category_1', 'most_recent_sales_range',␣
 ↪'most_recent_purchases_range', 'category_4']:
    merchant[col] = change_object_cols(merchant[col])

merchant[category_cols] = merchant[category_cols].fillna(-1)

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.inf, -99).max().max())

for col in numeric_cols:
    merchant[col] = merchant[col].fillna(merchant[col].mean())

duplicate_cols = ['merchant_id', 'merchant_category_id', 'subsector_id',␣
 ↪'category_1', 'city_id', 'state_id', 'category_2']
merchant = merchant.drop(duplicate_cols[1:], axis=1)
merchant = merchant.loc[merchant['merchant_id'].drop_duplicates().index.
 ↪tolist()].reset_index(drop=True)
```

[14]:
```
transaction = pd.concat([new_transaction, history_transaction], axis=0,␣
 ↪ignore_index=True)
del new_transaction
```

```
del history_transaction
gc.collect()

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']

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)

transaction['purchase_month'] = transaction['purchase_date'].apply(lambda x:'-'.
 ↪join(x.split(' ')[0].split('-')[:2]))
transaction['purchase_hour_section'] = transaction['purchase_date'].
 ↪apply(lambda x: x.split(' ')[1].split(':')[0]).astype(int)//6
transaction['purchase_day'] = transaction['purchase_date'].apply(lambda x:␣
 ↪datetime.strptime(x.split(" ")[0], "%Y-%m-%d").weekday())//5
del transaction['purchase_date']

transaction['purchase_month'] =␣
 ↪change_object_cols(transaction['purchase_month'].fillna(-1).astype(str))
```

[15]:
```
cols = ['merchant_id', 'most_recent_sales_range',␣
 ↪'most_recent_purchases_range', 'category_4']
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'].diff()
transaction['purchase_month_diff'] = transaction.
 ↪groupby("card_id")['purchase_month'].diff()
```

[16]:
```python
transaction.to_csv("preprocess/transaction_g_pre.csv", index=False)
```

[17]:
```python
del transaction
gc.collect()
```

[17]: 17