taobao_user_behavior_EDA

陈舒伦

October 27, 2020

0.1 项目简介

这个项目利用了淘宝 2017 年 11 月 25 日至 2017 年 12 月 3 日之间的用户行为数据来进行数据分析和业务优化。数据集取自：淘宝数据集
下面的内容包含了数据处理，数据分析和营收建议。数据处理利用了 MySQL 和 Pandas 两种方式来得到相同的结果，以验证分析方法和技术工具的可行性。

0.2 数据处理

[1]:
```
import pandas as pd
import numpy as np
import re
```

[2]:
```
import gc
gc.collect()
```

[2]: 20

[3]:
```
df = pd.read_csv('UserBehavior.csv', header=None,
                 names=['UserId', 'ItemId', 'CategoryId', 'BehaviorType',␣
↪'Timestamp'])
```

[179]:
```
# Try to reduce memory usage since the dataframe is large
def reduce_mem_usage(df, exclude_lst=['category']):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.select_dtypes(exclude=exclude_lst).columns:
        col_type = df[col].dtype
        if re.match("datetime.*", str(col_type)):
            continue
        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).
↪max:
```

```
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.
↪int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.
↪int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.
↪int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.
↪float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
↪float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) /␣
↪start_mem))

    return df
```

[5]:
```
df = reduce_mem_usage(df)
```

```
Memory usage of dataframe is 3820.45 MB
Memory usage after optimization is: 1623.69 MB
Decreased by 57.5%
```

[6]:
```
df.shape
```

[6]: (100150807, 5)

[7]:
```
df['UserId'].nunique()
```

[7]: 987994

[8]:
```
df.head()
```

[8]:

	UserId	ItemId	CategoryId	BehaviorType	Timestamp
0	1	2268318	2520377	pv	1511544070
1	1	2333346	2520771	pv	1511561733
2	1	2576651	149192	pv	1511572885
3	1	3830808	4181361	pv	1511593493

```
4        1  4365585     2520377            pv  1511596146
```

由于源数据较大，所以在这里仅摘取一部分数据作为示例。下面的例子里 MySQL 和 Pandas 均使用大小为 200000 的数据集为源数据集。

[9]:
```python
df.sample(n=200000, random_state=10).to_csv('user_behavior_sample.csv',␣
↪index=None)
```

[196]:
```python
df_sample = pd.read_csv('user_behavior_sample.csv')
```

[197]:
```python
df_sample.head()
```

[197]:

	UserId	ItemId	CategoryId	BehaviorType	Timestamp
0	841952	880244	4537798	pv	1511772749
1	339072	1288697	4382196	pv	1512048164
2	850248	3792765	2939262	pv	1511874286
3	941426	4367378	4956748	pv	1512315289
4	1011178	4662326	1485951	pv	1511695389

0.2.1 交替使用 MySQL 和 Pandas 来类比使用相同的方法处理数据

```sql
CREATE DATABASE test;

CREATE TABLE ub (
    UserId INT,
    ItemId INT,
    CategoryId INT,
    BehaviorType VARCHAR(10),
    Timestamp INT
);

/*
LOAD DATA LOCAL INFILE '/home/shulun/Documents/taobao_user_behavior/user_behavior_sample.csv'
INTO TABLE ub
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
*/

   Query OK, 200000 rows affected (0.98 sec)

/*
LOAD DATA LOCAL INFILE '/home/shulun/Documents/taobao_user_behavior/UserBehavior.csv'
INTO TABLE ub
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
*/
```

```
Query OK, 100150806 rows affected (7 min 15.13 sec)
```

检查重复值

```
SELECT DISTINCT UserId, ItemId, CategoryId, BehaviorType, Timestamp FROM ub;
```

[198]:
```
# Replace df with df_sample and delete df_sample
del df; gc.collect();
df = df_sample.copy()
del df_sample
```

[199]:
```
print('before:', df.shape)
df.drop_duplicates()
print('after:', df.shape)
```

```
before: (200000, 5)
after: (200000, 5)
```

Unix 时间戳转化为日期:

```
ALTER TABLE ub ADD ubtime datetime;
UPDATE ub SET ubtime = FROM_UNIXTIME(Timestamp, '%y-%m-%d %H:%i:%s');
```

长日期转化为短日期:

```
ALTER TABLE ub ADD ubdate date;
UPDATE ub SET ubdate = DATE_FORMAT(ubtime, '%Y-%m-%d');
```

提取小时数:

```
ALTER TABLE ub ADD ubhour INT;
UPDATE ub SET ubhour = HOUR(ubtime);
```

删除 Timestamp 列

```
ALTER TABLE ub DROP COLUMN Timestamp;
```

[ ]:
```
# Change timezone to China standard time, aka CST or Asia/Shanghai
df['ubtime'] = pd.to_datetime(df.Timestamp, unit='s').astype('datetime64[ns,␣
↪Asia/Shanghai]')
df.drop(['Timestamp'], axis=1, inplace=True)
```

[202]:
```
df['ubdate'] = df.ubtime.dt.date.astype('datetime64')
# df['ubdate'] = df.ubdate.dt.normalize()
```

[203]:
```
df['ubhour'] = df.ubtime.dt.hour
```

[204]:
```
print(df.dtypes); print(df.memory_usage())
```

```
UserId                                    int64
ItemId                                    int64
CategoryId                                int64
BehaviorType                             object
ubtime          datetime64[ns, Asia/Shanghai]
ubdate                           datetime64[ns]
ubhour                                    int64
dtype: object
Index               128
UserId          1600000
ItemId          1600000
CategoryId      1600000
BehaviorType    1600000
ubtime          1600000
ubdate          1600000
ubhour          1600000
dtype: int64
```

[205]:

```
df = reduce_mem_usage(df)
```

```
Memory usage of dataframe is 10.68 MB
Memory usage after optimization is: 5.72 MB
Decreased by 46.4%
```

添加主键

```
/*
ALTER TABLE ub ADD COLUMN BehaviorId INT NOT NULL AUTO_INCREMENT PRIMARY KEY FIRST;
*/
```

异常值处理

```
SELECT * FROM ub WHERE ubdate NOT BETWEEN '2017-11-25' AND '2017-12-03';
DELETE FROM ub WHERE ubdate NOT BETWEEN '2017-11-25' AND '2017-12-03';
```

[207]:

```
df = df[(df['ubdate']>='2017-11-25') & (df['ubdate']<='2017-12-03')].
 ↪reset_index(drop=True)
```

[208]:

```
print(df.dtypes); print(df.memory_usage())
```

```
UserId                                    int32
ItemId                                    int32
CategoryId                                int32
BehaviorType                           category
ubtime          datetime64[ns, Asia/Shanghai]
ubdate                           datetime64[ns]
ubhour                                     int8
dtype: object
```

```
Index                128
UserId            799544
ItemId            799544
CategoryId        799544
BehaviorType      200078
ubtime           1599088
ubdate           1599088
ubhour            199886
dtype: int64
```

[209]:
```python
'df is {:.2f} MB'.format(df.memory_usage().sum() / 1024**2)
```

[209]:
```
'df is 5.72 MB'
```

[210]:
```python
df.shape
```

[210]:
```
(199886, 7)
```

检查 NULL

```sql
SELECT
    SUM(CASE WHEN UserId IS NULL THEN 1 ELSE 0 END) AS UserId,
    SUM(CASE WHEN ItemId IS NULL THEN 1 ELSE 0 END) AS ItemId,
    SUM(CASE WHEN CategoryId IS NULL THEN 1 ELSE 0 END) AS ItemId,
    SUM(CASE WHEN BehaviorType IS NULL THEN 1 ELSE 0 END) AS BehaviorType,
    SUM(CASE WHEN ubtime IS NULL THEN 1 ELSE 0 END) AS ubtime,
    SUM(CASE WHEN ubdate IS NULL THEN 1 ELSE 0 END) AS ubdate,
    SUM(CASE WHEN ubhour IS NULL THEN 1 ELSE 0 END) AS ubhour
FROM ub;
```

去除 NULL

```sql
DELETE FROM ub WHERE ubtime IS NULL OR ubdate IS NULL OR ubhour IS NULL;
```

[213]:
```python
df.isnull().sum(axis=0)
```

[213]:
```
UserId          0
ItemId          0
CategoryId      0
BehaviorType    0
ubtime          0
ubdate          0
ubhour          0
dtype: int64
```

0.3 数据分析，交替使用 MySQL 和 Pandas 来进行相同的分析

0.3.1 总体运营指标

- **用户总体数据**

--总体浏览数量

```
SELECT COUNT(1) FROM ub WHERE BehaviorType = 'pv';
```

--总体订单数量

```
SELECT COUNT(1) FROM ub WHERE BehaviorType = 'buy';
```

- **用户人均数据**

--人均浏览数量

```
SELECT (SELECT COUNT(1) FROM ub WHERE BehaviorType = 'pv') /
(SELECT COUNT(DISTINCT UserId) FROM ub) AS 人均浏览数量;
```

--人均订单数量

```
SELECT (SELECT COUNT(1) FROM ub WHERE BehaviorType = 'buy') /
(SELECT COUNT(DISTINCT UserId) FROM ub) AS 人均订单数量;
```

[696]:

```
print('总体浏览数量: ', df[df.BehaviorType=='pv']['UserId'].count())
print('总体订单数量: ', df[df.BehaviorType=='buy']['UserId'].count())
print('人均浏览数量: ', round(df[df.BehaviorType=='pv']['UserId'].count() /␣
↪df['UserId'].nunique(), 4))
print('人均订单数量: ', round(df[df.BehaviorType=='buy']['UserId'].count() /␣
↪df['UserId'].nunique(), 4))
```

```
总体浏览数量:  178945
总体订单数量:  4001
人均浏览数量:  1.0537
人均订单数量:  0.0236
```

0.3.2 网站流量指标/销售转化指标

- **PV/UV 随时间变化趋势**

--按日统计

```
SELECT ubdate, COUNT(1) AS pv, COUNT(DISTINCT UserId) AS uv
FROM ub WHERE BehaviorType = 'pv'
GROUP BY ubdate
ORDER BY ubdate;
```

--按小时统计

```
SELECT ubhour, COUNT(1) AS pv, COUNT(DISTINCT UserId) AS uv
FROM ub WHERE BehaviorType = 'pv'
GROUP BY ubhour
ORDER BY ubhour;
```

[214]:
```
s1 = df[df['BehaviorType']=='pv'].groupby('ubdate')['UserId'].count()
s2 = df[df['BehaviorType']=='pv'].groupby('ubdate')['UserId'].nunique()
daily_pv_uv = pd.merge(s1, s2, how='inner', on=s1.index)
daily_pv_uv.columns = ['ubdate', 'pv', 'uv']
del s1, s2; gc.collect();
```

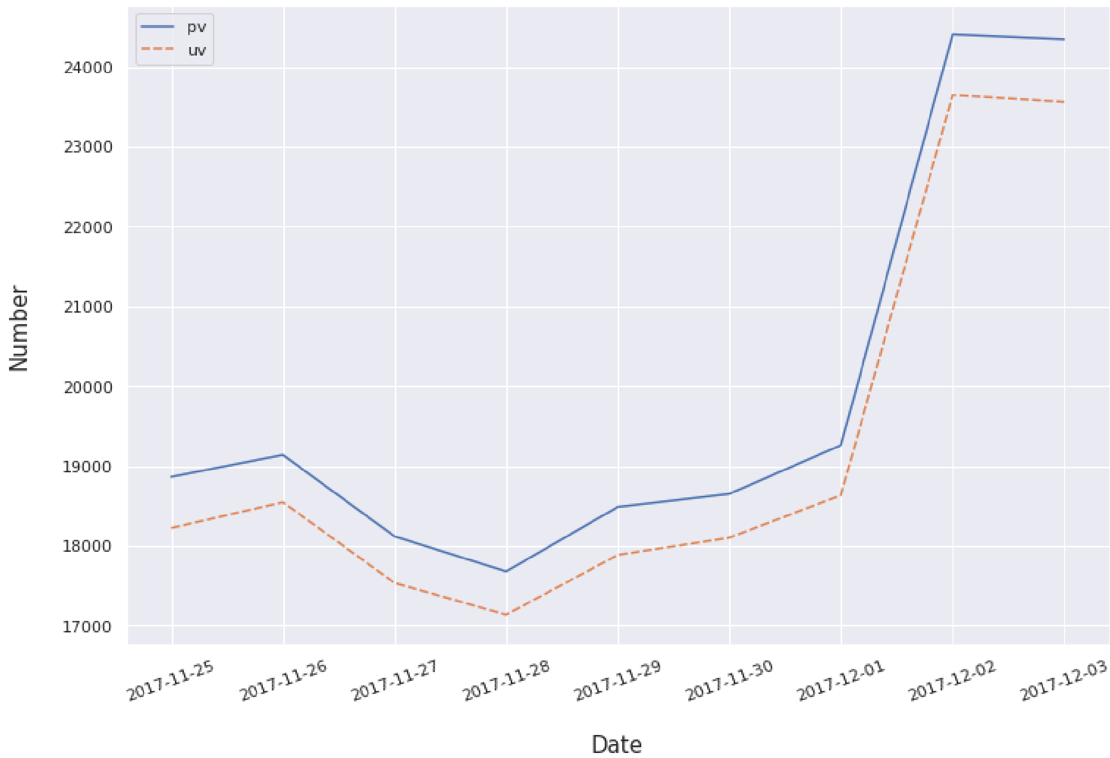
[215]:
```
daily_pv_uv
```

[215]:

	ubdate	pv	uv
0	2017-11-25	18864	18223
1	2017-11-26	19138	18545
2	2017-11-27	18118	17534
3	2017-11-28	17678	17136
4	2017-11-29	18486	17885
5	2017-11-30	18650	18101
6	2017-12-01	19260	18633
7	2017-12-02	24406	23648
8	2017-12-03	24345	23560

[674]:
```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

[679]:
```
plt.figure(figsize=(12, 8))
sns.lineplot(
    data=daily_pv_uv.set_index('ubdate')
)
plt.ylabel('Number\n', fontsize=15)
plt.xlabel('\nDate', fontsize=15)
plt.xticks(rotation=20)
plt.show()
```



[216]:
```
s1 = df[df['BehaviorType']=='pv'].groupby('ubhour')['UserId'].count()
s2 = df[df['BehaviorType']=='pv'].groupby('ubhour')['UserId'].nunique()
hourly_pv_uv = pd.merge(s1, s2, how='inner', on=s1.index)
hourly_pv_uv.columns = ['ubdate', 'pv', 'uv']
del s1, s2; gc.collect();
```

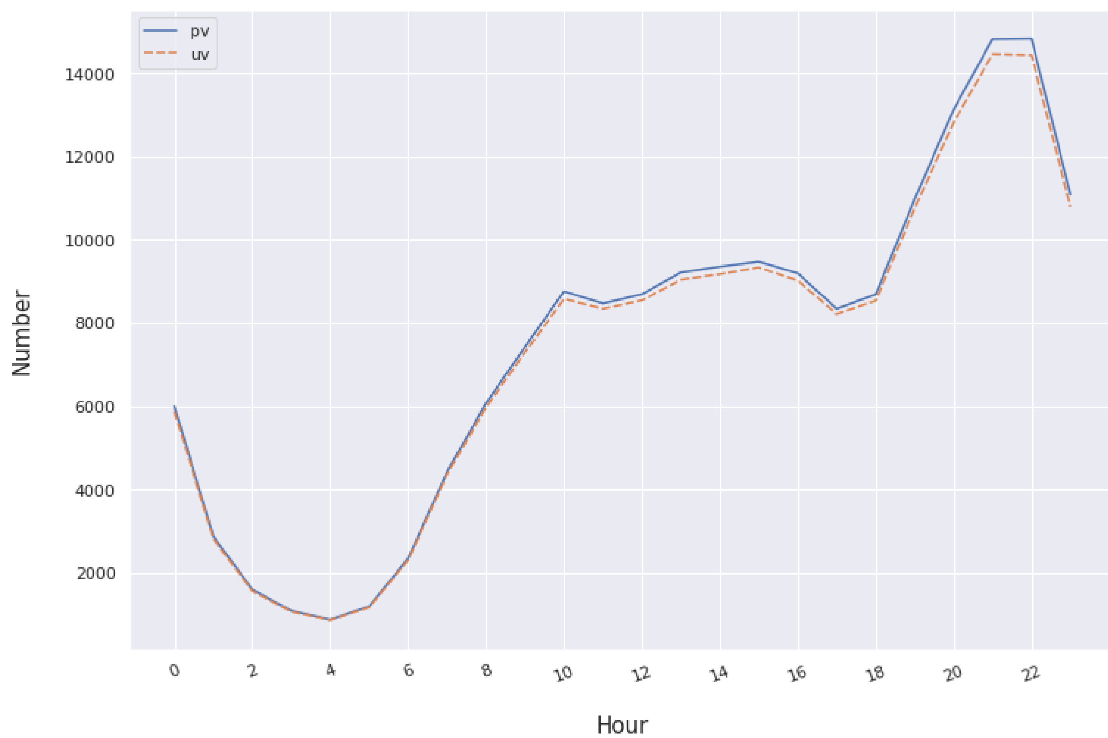
[217]:
```
hourly_pv_uv
```

[217]:

	ubdate	pv	uv
0	0	5996	5871
1	1	2868	2807
2	2	1595	1558
3	3	1083	1063
4	4	873	863
5	5	1189	1167
6	6	2344	2299
7	7	4425	4363
8	8	6075	5973
9	9	7428	7297
10	10	8761	8580
11	11	8471	8343
12	12	8693	8547
13	13	9216	9040
14	14	9356	9176

```
15        15   9480   9337
16        16   9196   9020
17        17   8343   8218
18        18   8689   8537
19        19  10976  10755
20        20  13131  12822
21        21  14826  14463
22        22  14838  14440
23        23  11093  10798
```

[683]:
```
plt.figure(figsize=(12, 8))
sns.lineplot(
    data=hourly_pv_uv.set_index('ubdate')
)
plt.ylabel('Number\n', fontsize=15)
plt.xlabel('\nHour', fontsize=15)
plt.xticks(np.arange(0, 23, 2), rotation=20)
plt.show()
```



跳失率

跳失率 = 只有点击（浏览）行为的用户/总用户数

```
SELECT
    COUNT(DISTINCT UserId) AS '只有点击行为的用户',
```

```
    CONCAT(ROUND(COUNT(DISTINCT UserId) * 100/(SELECT COUNT(DISTINCT UserId) FROM ub), 1), '%')
FROM ub
WHERE
    UserId NOT IN (SELECT DISTINCT UserId FROM ub WHERE BehaviorType='cart') AND
    UserId NOT IN (SELECT DISTINCT UserId FROM ub WHERE BehaviorType='buy') AND
    UserId NOT IN (SELECT DISTINCT UserId FROM ub WHERE BehaviorType='fav');
```

[231]:
```
cart_user_id = df[df.BehaviorType=='cart'].UserId.unique()
buy_user_id = df[df.BehaviorType=='buy'].UserId.unique()
fav_user_id = df[df.BehaviorType=='fav'].UserId.unique()
```

[298]:
```
print('跳失率： ', str(round(df[(~df.UserId.isin(cart_user_id)) &
                         (~df.UserId.isin(buy_user_id)) &
                         (~df.UserId.isin(fav_user_id))].UserId.nunique()*100 /
                          df.UserId.nunique(), 1))+'%')
```

```
跳失率：  87.9%
```

用户行为变化趋势

```
SELECT
    ubdate,
    SUM(CASE BehaviorType WHEN 'pv' THEN 1 ELSE 0 END) AS 'pv',
    SUM(CASE BehaviorType WHEN 'cart' THEN 1 ELSE 0 END) AS 'cart',
    SUM(CASE BehaviorType WHEN 'buy' THEN 1 ELSE 0 END) AS 'buy',
    SUM(CASE BehaviorType WHEN 'fav' THEN 1 ELSE 0 END) AS 'fav'
FROM ub
GROUP BY ubdate
ORDER BY ubdate;
```

[290]:
```
def reset_index(df):
    '''Returns DataFrame with index as columns'''
    index_df = df.index.to_frame(index=False)
    df = df.reset_index(drop=True)
    #  In merge is important the order in which you pass the dataframes
    #  if the index contains a Categorical.
    #  pd.merge(df, index_df, left_index=True, right_index=True) does not work
    return pd.merge(index_df, df, left_index=True, right_index=True)
```

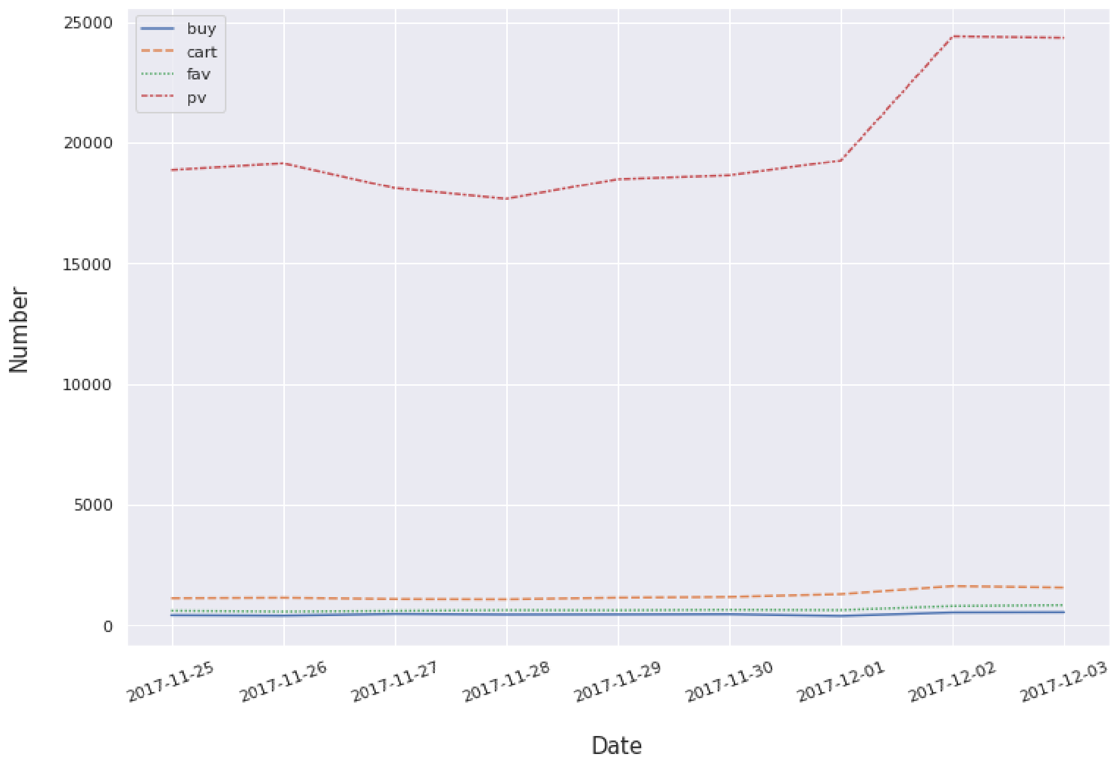
[698]:
```
daily_all = reset_index(df.groupby(['ubdate', 'BehaviorType']).size().unstack())
daily_all
# df.pivot_table(columns='BehaviorType', index='ubdate', aggfunc='size')
```

[698]:

	ubdate	buy	cart	fav	pv
0	2017-11-25	407	1116	590	18864
1	2017-11-26	390	1137	553	19138
2	2017-11-27	457	1082	585	18118
3	2017-11-28	435	1072	613	17678
4	2017-11-29	439	1139	607	18486

```
5 2017-11-30  447  1169  628  18650
6 2017-12-01  378  1282  615  19260
7 2017-12-02  519  1608  779  24406
8 2017-12-03  529  1548  817  24345
```

[699]:
```python
plt.figure(figsize=(12, 8))
sns.lineplot(
    data=daily_all.set_index('ubdate')
)
plt.ylabel('Number\n', fontsize=15)
plt.xlabel('\nDate', fontsize=15)
plt.xticks(rotation=20)
plt.show()
```



用户行为漏斗分析

- **按行为数量计算（即流量）**

```sql
SELECT
    SUM(CASE BehaviorType WHEN 'pv' THEN 1 ELSE 0 END) AS 'pv',
    SUM(CASE BehaviorType WHEN 'cart' THEN 1 ELSE 0 END) AS 'cart',
    SUM(CASE BehaviorType WHEN 'buy' THEN 1 ELSE 0 END) AS 'buy',
    SUM(CASE BehaviorType WHEN 'fav' THEN 1 ELSE 0 END) AS 'fav'
FROM ub;
```

```
SELECT
    CONCAT(ROUND(SUM(CASE BehaviorType WHEN 'buy' THEN 1 ELSE 0 END)*100/SUM(CASE BehaviorType
    CONCAT(ROUND(SUM(CASE BehaviorType WHEN 'fav' THEN 1 ELSE 0 END)*100/SUM(CASE BehaviorType
    CONCAT(ROUND(SUM(CASE BehaviorType WHEN 'cart' THEN 1 ELSE 0 END)*100/SUM(CASE BehaviorTyp
FROM ub;
```

[573]:
```
funnel_order_cols = ['pv', 'cart', 'fav', 'buy']
```

[575]:
```
bt_agg = df.groupby(['BehaviorType']).size()
bt_agg = reset_index(bt_agg.to_frame().T[funnel_order_cols]).drop([0], axis=1)
bt_agg
```

[575]:

	pv	cart	fav	buy
0	178945	11153	5787	4001

[705]:
```
funnel_pct_orders = ['购物车转化率', '收藏转化率', '购买转化率']
```

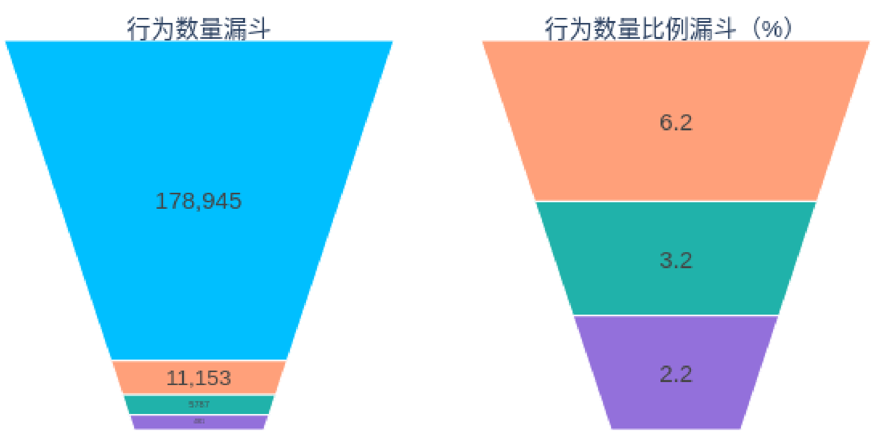
[707]:
```
bt_pct = reset_index(((df.groupby(['BehaviorType']).size()*100 /
                       df.groupby(['BehaviorType']).size().pv).round(1).
 ↪astype(str) + '%').to_frame().T)\
             .drop([0, 'pv'], axis=1).rename(columns={'buy':'购买转化率',
                                                       'cart':'购物车转化率',
                                                       'fav':'收藏转化率'})
bt_pct = bt_pct[funnel_pct_orders]
bt_pct
```

[707]:

	购物车转化率	收藏转化率	购买转化率
0	6.2%	3.2%	2.2%

[717]:
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import plotly.offline as pyo

# Set notebook mode to work in offline
pyo.init_notebook_mode()

fig = make_subplots(
    rows=1, cols=2,
    specs=[[{"type": "domain"}, {"type": "domain"}]],
)

fig.add_trace(go.Funnelarea(
    text = bt_agg.columns.values,
    values = bt_agg.iloc[0].values,
    textinfo = 'value',
    title = {"position": "top center", "text": "行为数量漏斗"},
    marker = {"colors": ["deepskyblue", "lightsalmon", "lightseagreen",␣
 ↪"mediumpurple"]}
    ), row=1, col=1)
```

```python
fig.add_trace(go.Funnelarea(
    text = bt_pct.columns.values,
    values = [float(x.strip('%')) for x in bt_pct.iloc[0].values],
    textinfo = 'value',
    title = {"position": "top center", "text": "行为数量比例漏斗 (%) "},
    marker = {"colors": ["lightsalmon", "lightseagreen", "mediumpurple"]}
    ), row=1, col=2)

fig.update_layout(height=700, showlegend=False, font=dict(size=15))
fig.show()
```



- **按 UV(独立访客) 计算**

--计算 UV

```sql
SELECT COUNT(DISTINCT UserId) FROM ub;
```

--计算四种用户行为

```
CREATE VIEW uv AS
SELECT BehaviorType, COUNT(DISTINCT UserId) AS uv
FROM ub GROUP BY BehaviorType;
```

--计算转化率

```
SELECT
    CONCAT(ROUND(SUM(CASE BehaviorType WHEN 'buy' THEN uv ELSE 0 END)*100/SUM(CASE BehaviorType
    CONCAT(ROUND(SUM(CASE BehaviorType WHEN 'fav' THEN uv ELSE 0 END)*100/SUM(CASE BehaviorType
    CONCAT(ROUND(SUM(CASE BehaviorType WHEN 'cart' THEN uv ELSE 0 END)*100/SUM(CASE BehaviorTy
FROM uv;
```

[649]:
```
print('不重复用户人数: ', df.UserId.nunique())
uv = reset_index(df.groupby('BehaviorType').UserId.nunique().to_frame().
 ↪T[funnel_order_cols]).drop([0], axis=1)
uv
```

不重复用户人数: 169825

[649]:

	pv	cart	fav	buy
0	153996	10994	5664	3983

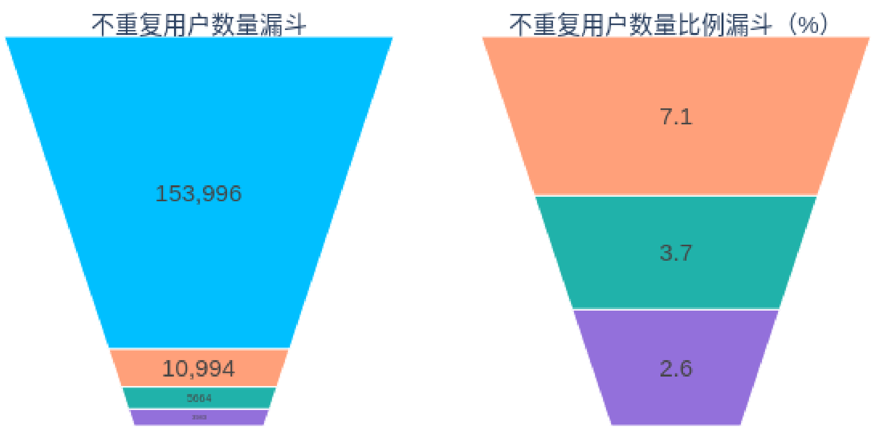
[719]:
```
uv_pct = reset_index(((df.groupby(['BehaviorType']).UserId.nunique()*100 /
             df.groupby(['BehaviorType']).UserId.nunique().pv).round(1).
 ↪astype(str) + '%').to_frame().T)\
        .drop([0, 'pv'], axis=1).rename(columns={'buy':'购买转化率',
                                                 'cart':'购物车转化率',
                                                 'fav':'收藏转化率'})
uv_pct = uv_pct[funnel_pct_orders]
uv_pct
```

[719]:

	购物车转化率	收藏转化率	购买转化率
0	7.1%	3.7%	2.6%

[720]:
```
from plotly.subplots import make_subplots

fig = make_subplots(
    rows=1, cols=2,
    specs=[[{"type": "domain"}, {"type": "domain"}]],
)

fig.add_trace(go.Funnelarea(
    text = uv.columns.values,
    values = uv.iloc[0].values,
    textinfo = 'value',
    title = {"position": "top center", "text": "不重复用户数量漏斗"},
```

```
    marker = {"colors": ["deepskyblue", "lightsalmon", "lightseagreen",␣
↪"mediumpurple"]}
    ), row=1, col=1)

fig.add_trace(go.Funnelarea(
    text = uv_pct.columns.values,
    values = [float(x.strip('%')) for x in uv_pct.iloc[0].values],
    textinfo = 'value',
    title = {"position": "top center", "text": "不重复用户数量比例漏斗 (%) "},
    marker = {"colors": ["lightsalmon", "lightseagreen", "mediumpurple"]}
    ), row=1, col=2)

fig.update_layout(height=700, showlegend=False, font=dict(size=15))
fig.show()
```



- **复购率**

复购率 = 有复购行为的用户数/有购买行为的用户数

```
CREATE OR REPLACE VIEW f AS
SELECT UserId, COUNT(1) AS '购买次数'
FROM ub
WHERE BehaviorType='buy'
GROUP BY UserId
HAVING COUNT(BehaviorType) >= 2;
```

[560]:
```
f = df[df['BehaviorType']=='buy'][['UserId', 'ItemId']].groupby('UserId').
    ↪filter(lambda x: len(x) >= 2)
f = f.groupby('UserId').count()
f.head()
```

[560]:

```
        ItemId
UserId
56954        2
60963        2
76164        2
242651       2
293601       2
```

--计算复购率

```
SELECT CONCAT(ROUND((SELECT COUNT(UserId) FROM f)*100 /
    (SELECT COUNT(DISTINCT UserId) FROM ub WHERE BehaviorType='buy'), 1), '%') AS '复购率';
```

[391]:
```
print('复购率：', str(round(len(f.index)*100/df[df.BehaviorType=='buy'].UserId.
    ↪nunique(), 1))+'%')
```

```
复购率： 0.5%
```

- **复购频率分布**

```
SELECT 购买次数, COUNT(购买次数) AS 人数
FROM (SELECT UserId, COUNT(1) AS '购买次数' FROM ub
      WHERE BehaviorType='buy' GROUP BY UserId) AS a
GROUP BY 购买次数
ORDER BY 购买次数;
```

[399]:
```
df[df.BehaviorType=='buy'][['UserId', 'ItemId']].groupby('UserId').count()\
    .reset_index().groupby('ItemId').count().reset_index().
    ↪rename(columns={'ItemId':'购买次数',
    ↪'UserId':'人数'})
```

[399]:

```
   购买次数    人数
0      1  3965
1      2    18
```

--获取每个用户的使用日期和第一次使用日期

```
CREATE VIEW retention AS
SELECT ub.UserId, ubdate, firstday
FROM ub INNER JOIN
(SELECT UserId, MIN(ubdate) AS firstday FROM ub GROUP BY UserId) AS b
ON ub.UserId = b.UserId ORDER BY ub.UserId, ubdate;
```

--计算时间间隔

```
CREATE VIEW daydiff AS
SELECT UserId, ubdate, firstday, datediff(ubdate, firstday) AS day_diff
FROM retention;
```

--计算每日的流量留存数量

```
CREATE VIEW retention2 AS
SELECT firstday,
SUM(CASE WHEN day_diff=0 THEN 1 ELSE 0 END) AS day0,
SUM(CASE WHEN day_diff=1 THEN 1 ELSE 0 END) AS day1,
SUM(CASE WHEN day_diff=2 THEN 1 ELSE 0 END) AS day2,
SUM(CASE WHEN day_diff=3 THEN 1 ELSE 0 END) AS day3,
SUM(CASE WHEN day_diff=4 THEN 1 ELSE 0 END) AS day4,
SUM(CASE WHEN day_diff=5 THEN 1 ELSE 0 END) AS day5,
SUM(CASE WHEN day_diff=6 THEN 1 ELSE 0 END) AS day6,
SUM(CASE WHEN day_diff=7 THEN 1 ELSE 0 END) AS day7,
SUM(CASE WHEN day_diff=8 THEN 1 ELSE 0 END) AS day8
FROM daydiff
GROUP BY firstday
ORDER BY firstday;
```

--计算每日的不重复用户留存数量

```
CREATE VIEW retention3 AS
SELECT
    day0.firstday,
    COALESCE(day0.cnt, 0) AS day0,
    COALESCE(day1.cnt, 0) AS day1,
    COALESCE(day2.cnt, 0) AS day2,
    COALESCE(day3.cnt, 0) AS day3,
    COALESCE(day4.cnt, 0) AS day4,
    COALESCE(day5.cnt, 0) AS day5,
    COALESCE(day6.cnt, 0) AS day6,
    COALESCE(day7.cnt, 0) AS day7,
    COALESCE(day8.cnt, 0) AS day8
FROM
    (SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=0 GROUP BY firs
    LEFT JOIN
    (SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=1 GROUP BY firs
    USING(firstday)
    LEFT JOIN
    (SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=2 GROUP BY firs
    USING(firstday)
    LEFT JOIN
    (SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=3 GROUP BY firs
    USING(firstday)
    LEFT JOIN
    (SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=4 GROUP BY firs
    USING(firstday)
```

```
LEFT JOIN
(SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=5 GROUP BY firs
USING(firstday)
LEFT JOIN
(SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=6 GROUP BY firs
USING(firstday)
LEFT JOIN
(SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=7 GROUP BY firs
USING(firstday)
LEFT JOIN
(SELECT firstday, COUNT(DISTINCT UserId) AS cnt FROM daydiff WHERE day_diff=8 GROUP BY firs
USING(firstday);
```

9 rows in set (7.44 sec)
--流量留存率计算

```
SELECT firstday,
CONCAT(ROUND(day1*100/day0, 1), '%') AS day1,
CONCAT(ROUND(day2*100/day0, 1), '%') AS day2,
CONCAT(ROUND(day3*100/day0, 1), '%') AS day3,
CONCAT(ROUND(day4*100/day0, 1), '%') AS day4,
CONCAT(ROUND(day5*100/day0, 1), '%') AS day5,
CONCAT(ROUND(day6*100/day0, 1), '%') AS day6,
CONCAT(ROUND(day7*100/day0, 1), '%') AS day7,
CONCAT(ROUND(day8*100/day0, 1), '%') AS day8
FROM retention2
ORDER BY firstday;
```

--不重复用户留存率计算

```
SELECT firstday,
CONCAT(ROUND(day1*100/day0, 1), '%') AS day1,
CONCAT(ROUND(day2*100/day0, 1), '%') AS day2,
CONCAT(ROUND(day3*100/day0, 1), '%') AS day3,
CONCAT(ROUND(day4*100/day0, 1), '%') AS day4,
CONCAT(ROUND(day5*100/day0, 1), '%') AS day5,
CONCAT(ROUND(day6*100/day0, 1), '%') AS day6,
CONCAT(ROUND(day7*100/day0, 1), '%') AS day7,
CONCAT(ROUND(day8*100/day0, 1), '%') AS day8
FROM retention3
ORDER BY firstday;
```

[435]:
```
first_days = df.groupby('UserId').ubdate.min()
retention = pd.merge(df[['UserId', 'ubdate']], first_days, how='inner',␣
 ↪left_on=df.UserId,
                     right_on=first_days.index)
retention = retention.loc[:, retention.columns!='key_0']
retention.columns = ['UserId', 'ubdate', 'firstday']
```

```python
retention.sort_values(by=['UserId', 'ubdate'], inplace=True)
del first_days; gc.collect();
retention.head()
```

[435]:

```
        UserId      ubdate   firstday
37941        6 2017-11-25 2017-11-25
41394       21 2017-11-26 2017-11-26
41395       21 2017-11-28 2017-11-26
41396       21 2017-12-03 2017-11-26
103285      32 2017-12-01 2017-12-01
```

[444]:

```python
daydiff = retention.copy()
daydiff['day_diff'] = (daydiff['ubdate'] - daydiff['firstday']).dt.days.
 ↪astype('int16')
daydiff.head()
```

[444]:

```
        UserId      ubdate   firstday  day_diff
37941        6 2017-11-25 2017-11-25         0
41394       21 2017-11-26 2017-11-26         0
41395       21 2017-11-28 2017-11-26         2
41396       21 2017-12-03 2017-11-26         7
103285      32 2017-12-01 2017-12-01         0
```

[452]:

```python
retention2 = daydiff.copy()
retention2['day0'] = retention2['day_diff'] == 0
retention2['day1'] = retention2['day_diff'] == 1
retention2['day2'] = retention2['day_diff'] == 2
retention2['day3'] = retention2['day_diff'] == 3
retention2['day4'] = retention2['day_diff'] == 4
retention2['day5'] = retention2['day_diff'] == 5
retention2['day6'] = retention2['day_diff'] == 6
retention2['day7'] = retention2['day_diff'] == 7
retention2['day8'] = retention2['day_diff'] == 8
```

[456]:

```python
# 计算每日的流量留存数量
retention2_cnt = retention2[['firstday', 'day0', 'day1', 'day2', 'day3',
                             'day4', 'day5', 'day6', 'day7', 'day8']].
 ↪groupby('firstday').sum().reset_index()
retention2_cnt
```

[456]:

```
    firstday   day0  day1  day2  day3  day4  day5  day6  day7  day8
0 2017-11-25  20977   861   681   600   607   647   659   790   779
1 2017-11-26  20357   725   633   643   636   579   784   757     0
2 2017-11-27  18836   718   670   583   564   687   664     0     0
3 2017-11-28  17847   691   601   571   611   663     0     0     0
4 2017-11-29  18060   667   595   689   638     0     0     0     0
5 2017-11-30  17760   645   667   659     0     0     0     0     0
6 2017-12-01  17922   755   646     0     0     0     0     0     0
7 2017-12-02  22329   881     0     0     0     0     0     0     0
8 2017-12-03  21552     0     0     0     0     0     0     0     0
```

[743]:
```python
# 计算每日的不重复用户留存数量
retention3 = daydiff.copy()
lst = [retention3[retention3.day_diff==i].groupby('firstday').UserId.nunique()␣
 ↪for i in range(9)]
retention3_cnt = pd.concat(lst, axis=1).reset_index().fillna(0)
del lst; gc.collect();
retention3_cnt.columns = ['firstday', 'day0', 'day1', 'day2', 'day3', 'day4',␣
 ↪'day5', 'day6', 'day7', 'day8']
retention3_cnt
```

[743]:

	firstday	day0	day1	day2	day3	day4	day5	day6	day7	day8
0	2017-11-25	20219	828.0	648.0	584.0	584.0	617.0	635.0	753.0	740.0
1	2017-11-26	19660	688.0	600.0	610.0	610.0	555.0	752.0	729.0	0.0
2	2017-11-27	18208	681.0	636.0	559.0	542.0	660.0	635.0	0.0	0.0
3	2017-11-28	17298	664.0	579.0	540.0	590.0	632.0	0.0	0.0	0.0
4	2017-11-29	17443	641.0	574.0	664.0	615.0	0.0	0.0	0.0	0.0
5	2017-11-30	17218	614.0	640.0	625.0	0.0	0.0	0.0	0.0	0.0
6	2017-12-01	17326	722.0	613.0	0.0	0.0	0.0	0.0	0.0	0.0
7	2017-12-02	21617	845.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	2017-12-03	20836	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[469]:
```python
# 流量留存率计算
retention2_pct = pd.concat([retention2_cnt.firstday,
                            (retention2_cnt.day1*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day2*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day3*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day4*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day5*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day6*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day7*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention2_cnt.day8*100 / retention2_cnt.day0).
 ↪round(1).astype(str) + '%'
                           ], axis=1)
retention2_pct.columns = ['firstday', 'day1', 'day2', 'day3', 'day4', 'day5',␣
 ↪'day6', 'day7', 'day8']
retention2_pct
```

[469]:

	firstday	day1	day2	day3	day4	day5	day6	day7	day8
0	2017-11-25	4.1%	3.2%	2.9%	2.9%	3.1%	3.1%	3.8%	3.7%
1	2017-11-26	3.6%	3.1%	3.2%	3.1%	2.8%	3.9%	3.7%	0.0%
2	2017-11-27	3.8%	3.6%	3.1%	3.0%	3.6%	3.5%	0.0%	0.0%

```
3 2017-11-28  3.9%  3.4%  3.2%  3.4%  3.7%  0.0%  0.0%  0.0%
4 2017-11-29  3.7%  3.3%  3.8%  3.5%  0.0%  0.0%  0.0%  0.0%
5 2017-11-30  3.6%  3.8%  3.7%  0.0%  0.0%  0.0%  0.0%  0.0%
6 2017-12-01  4.2%  3.6%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%
7 2017-12-02  3.9%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%
8 2017-12-03  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%
```

[744]:
```
# 不重复用户留存率计算
retention3_pct = pd.concat([retention3_cnt.firstday,
                            (retention3_cnt.day1*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day2*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day3*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day4*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day5*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day6*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day7*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%',
                            (retention3_cnt.day8*100 / retention3_cnt.day0).
 ↪round(1).astype(str) + '%'
                            ], axis=1)
retention3_pct.columns = ['firstday', 'day1', 'day2', 'day3', 'day4', 'day5',␣
 ↪'day6', 'day7', 'day8']
retention3_pct
```

[744]:
```
    firstday  day1  day2  day3  day4  day5  day6  day7  day8
0 2017-11-25  4.1%  3.2%  2.9%  2.9%  3.1%  3.1%  3.7%  3.7%
1 2017-11-26  3.5%  3.1%  3.1%  3.1%  2.8%  3.8%  3.7%  0.0%
2 2017-11-27  3.7%  3.5%  3.1%  3.0%  3.6%  3.5%  0.0%  0.0%
3 2017-11-28  3.8%  3.3%  3.1%  3.4%  3.7%  0.0%  0.0%  0.0%
4 2017-11-29  3.7%  3.3%  3.8%  3.5%  0.0%  0.0%  0.0%  0.0%
5 2017-11-30  3.6%  3.7%  3.6%  0.0%  0.0%  0.0%  0.0%  0.0%
6 2017-12-01  4.2%  3.5%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%
7 2017-12-02  3.9%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%
8 2017-12-03  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%  0.0%
```

[ ]:
```
del retention, retention2, retention2_cnt, retention2_pct;
del retenton3, retention3_cnt, retention3_pct;
gc.collect();
```

0.3.3 客户价值指标

- **RFM 用户价值模型分析**

```
/*
CREATE VIEW r AS
SELECT
    a.UserId,
    a.recent_buy_date,
    MAX(a.recent_buy_date) OVER() AS max_date
FROM
    (SELECT
        UserId,
        MAX(ubdate) AS recent_buy_date
    FROM ub
    WHERE BehaviorType = 'buy'
    GROUP BY UserId) AS a;
*/

CREATE VIEW r AS
SELECT
    UserId,
    MAX(ubdate) AS most_recent_buy_date
FROM
    ub
WHERE
    BehaviorType='buy'
GROUP BY
    UserId;
```

[478]:

```
r = df[df.BehaviorType=='buy'][['UserId', 'ubdate']].groupby('UserId').ubdate.
↪max().to_frame().reset_index()
r.columns = ['UserId', 'most_recent_buy_date']
r.head()
```

[478]:

	UserId	most_recent_buy_date
0	95	2017-11-29
1	150	2017-11-28
2	342	2017-11-27
3	410	2017-11-28
4	622	2017-11-26

建立 recency 等级划分

```
CREATE VIEW r_level AS
SELECT
    UserId,
    most_recent_buy_date,
    datediff('2017-12-03', most_recent_buy_date) AS days_till_today,
    (CASE
        WHEN datediff('2017-12-03', most_recent_buy_date) <= 4 THEN 4
        WHEN datediff('2017-12-03', most_recent_buy_date) <= 6 THEN 3
        WHEN datediff('2017-12-03', most_recent_buy_date) <= 8 THEN 2
```

```
        ELSE 1 END
    ) AS r_val
from r;
```

[487]:
```
r_level = r.copy()
r_level['days_till_today'] = (df.ubdate.max() -␣
 ↪r_level['most_recent_buy_date']).dt.days

def assign_r_val(x):
    if x <= 4:
        return 4
    elif x <= 6:
        return 3
    elif x <= 8:
        return 2
    return 1

r_level['r_val'] = r_level['days_till_today'].apply(lambda x: assign_r_val(x))
r_level.head()
```

[487]:

	UserId	most_recent_buy_date	days_till_today	r_val
0	95	2017-11-29	4	4
1	150	2017-11-28	5	3
2	342	2017-11-27	6	3
3	410	2017-11-28	5	3
4	622	2017-11-26	7	2

查询每个用户的购买次数

```
DROP VIEW IF EXISTS f;
CREATE VIEW f AS
SELECT
    UserId,
    COUNT(UserId) AS '购买次数'
FROM
    ub
WHERE
    BehaviorType = 'buy'
GROUP BY
    UserId;
```

[495]:
```
del f; gc.collect()
f = df[df.BehaviorType=='buy'][['UserId', 'ItemId']].groupby('UserId').ItemId.
 ↪count().to_frame().reset_index()
f.columns = ['UserId', '购买次数']
f.head()
```

[495]:

	UserId	购买次数
0	95	1
1	150	1

```
2      342      1
3      410      1
4      622      1
```

建立 f 等级划分

```
CREATE VIEW f_level AS
SELECT
    UserId,
    购买次数,
    (CASE
        WHEN 购买次数 <= 1 THEN 1
        WHEN 购买次数 <= 2 THEN 2
        WHEN 购买次数 <= 3 THEN 3
        WHEN 购买次数 <= 4 THEN 4
        ELSE 5 END
    ) AS f_val
FROM f;
```

[497]:
```python
f_level = f.copy()

def assign_f_val(x):
    if x <= 1:
        return 1
    elif x <= 2:
        return 2
    elif x <= 3:
        return 3
    elif x <= 4:
        return 4
    return 5

f_level['f_val'] = f_level['购买次数'].apply(lambda x: assign_f_val(x))
f_level.head()
```

[497]:

	UserId	购买次数	f_val
0	95	1	1
1	150	1	1
2	342	1	1
3	410	1	1
4	622	1	1

r_val 平均值

```
SELECT AVG(r_val) AS r_val_avg FROM r_level;
```

f_val 平均值

```
SELECT AVG(f_val) AS f_val_avg FROM f_level;
```

[500]:
```python
r_val_avg = r_level.r_val.mean()
print(round(r_val_avg, 4))
f_val_avg = f_level.f_val.mean()
print(round(f_val_avg, 4))
```

```
3.3796
1.0045
```

客户等级划分总结

```sql
SELECT @r_val_avg:=AVG(r_val) FROM r_level;
SELECT @f_val_avg:=AVG(f_val) FROM f_level;
CREATE TABLE rfm AS
SELECT
    a.*,
    b.f_val,
    b.购买次数,
    (CASE
        WHEN a.r_val > @r_val_avg AND b.f_val > @f_val_avg THEN '高价值客户'
        WHEN a.r_val < @r_val_avg AND b.f_val > @f_val_avg THEN '唤回客户'
        WHEN a.r_val > @r_val_avg AND b.f_val < @f_val_avg THEN '深耕客户'
        WHEN a.r_val < @r_val_avg AND b.f_val < @f_val_avg THEN '挽留客户'
     END) AS 客户分类
FROM
    r_level AS a,
    f_level AS b
WHERE
    a.UserId = b.UserId;
```

[519]:
```python
rfm = pd.concat([r_level, f_level[['f_val', '购买次数']]], axis=1)

def assign_category(x, y):
    if x > r_val_avg and y > f_val_avg:
        return '高价值客户'
    elif x < r_val_avg and y > f_val_avg:
        return '唤回客户'
    elif x > r_val_avg and y < f_val_avg:
        return '深耕客户'
    elif x < r_val_avg and y < f_val_avg:
        return '挽留客户'
    return ''

rfm['客户分类'] = rfm.apply(lambda x: assign_category(x.r_val, x.f_val), axis=1)
rfm.head()
```

[519]:

	UserId	most_recent_buy_date	days_till_today	r_val	f_val	购买次数	客户分类
0	95	2017-11-29	4	4	1	1	深耕客户
1	150	2017-11-28	5	3	1	1	挽留客户

```
2      342            2017-11-27                6       3       1      1  挽留客户
3      410            2017-11-28                5       3       1      1  挽留客户
4      622            2017-11-26                7       2       1      1  挽留客户
```

```
SELECT
    客户分类,
    COUNT(1) AS 数量
FROM rfm
GROUP BY 客户分类;
```

[522]:
```
rfm[['客户分类', 'UserId']].groupby('客户分类').count().reset_index().
↪rename(columns={'UserId':'数量'})
```

[522]:
```
    客户分类    数量
0   唤回客户     4
1   挽留客户  1675
2   深耕客户  2290
3  高价值客户    14
```

0.3.4 Top 10 商品/类目分析

- 按类目分析

```
CREATE VIEW cate AS
SELECT
    CategoryId,
    SUM(CASE WHEN BehaviorType='pv' THEN 1 ELSE 0 END) AS '浏览量',
    SUM(CASE WHEN BehaviorType='buy' THEN 1 ELSE 0 END) AS '购买量',
    CONCAT(ROUND(SUM(CASE WHEN BehaviorType='buy' THEN 1 ELSE 0 END)/
                 SUM(CASE WHEN BehaviorType='pv' THEN 1 ELSE 0 END)*100, 1), '%') AS '购买转化
FROM ub
GROUP BY CategoryId
HAVING SUM(CASE WHEN BehaviorType='pv' THEN 1 ELSE 0 END) > 0;
```

[554]:
```
cate = df[['CategoryId', 'BehaviorType']]
cate['浏览量'] = cate['BehaviorType']=='pv'
cate['购买量'] = cate['BehaviorType']=='buy'
cate = cate.groupby('CategoryId').sum()
cate = cate[cate.浏览量!=0]
cate['购买转化率'] = (cate.购买量 * 100/cate.浏览量).round(1).astype(str) + '%'
cate.head()
```

```
/home/shulun/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/home/shulun/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[554]:

CategoryId	浏览量	购买量	购买转化率
2171	5	0	0.0%
2410	2	0	0.0%
3579	1	1	100.0%
4907	2	0	0.0%
5064	60	0	0.0%

Top 10 购买类目

```
SELECT * FROM cate
ORDER BY 购买量 DESC LIMIT 10;
```

Top 10 浏览类目

```
SELECT * FROM cate
ORDER BY 浏览量 DESC LIMIT 10;
```

[555]:

```
cate.sort_values(by='购买量', ascending=False).reset_index().head(10)
```

[555]:

	CategoryId	浏览量	购买量	购买转化率
0	2735466	2186	84	3.8%
1	1464116	1366	77	5.6%
2	4145813	6332	72	1.1%
3	4756105	8903	63	0.7%
4	2885642	1875	55	2.9%
5	4801426	3704	49	1.3%
6	1320293	3649	38	1.0%
7	982926	5626	38	0.7%
8	4357323	1344	37	2.8%
9	4789432	651	36	5.5%

[556]:

```
cate.sort_values(by='浏览量', ascending=False).reset_index().head(10)
```

[556]:

	CategoryId	浏览量	购买量	购买转化率
0	4756105	8903	63	0.7%
1	4145813	6332	72	1.1%

```
2     2355072  6294   22  0.3%
3     3607361  5989   24  0.4%
4      982926  5626   38  0.7%
5     2520377  4086   17  0.4%
6     4801426  3704   49  1.3%
7     1320293  3649   38  1.0%
8     2465336  3061   20  0.7%
9     3002561  2843   33  1.2%
```

- 按商品分类

```
CREATE VIEW item AS
SELECT
    ItemId,
    SUM(CASE WHEN BehaviorType='pv' THEN 1 ELSE 0 END) AS '浏览量',
    SUM(CASE WHEN BehaviorType='buy' THEN 1 ELSE 0 END) AS '购买量',
    CONCAT(ROUND(SUM(CASE WHEN BehaviorType='buy' THEN 1 ELSE 0 END)/
                 SUM(CASE WHEN BehaviorType='pv' THEN 1 ELSE 0 END)*100, 1), '%') AS '购买转化
FROM ub
GROUP BY ItemId
HAVING SUM(CASE WHEN BehaviorType='pv' THEN 1 ELSE 0 END) > 0;
```

[557]:
```
item = df[['ItemId', 'BehaviorType']]
item['浏览量'] = item['BehaviorType']=='pv'
item['购买量'] = item['BehaviorType']=='buy'
item = item.groupby('ItemId').sum()
item = item[item.浏览量!=0]
item['购买转化率'] = (item.购买量 * 100/item.浏览量).round(1).astype(str) + '%'
item.head()
```

```
/home/shulun/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/home/shulun/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[557]:

```
        浏览量  购买量 购买转化率
ItemId
72        1    0   0.0%
81        1    0   0.0%
113       1    0   0.0%
116       1    0   0.0%
142       1    0   0.0%
```

Top 10 购买商品

```
SELECT * FROM item
ORDER BY 购买量 DESC LIMIT 10;
```

Top 10 浏览商品

```
SELECT * FROM item
ORDER BY 浏览量 DESC LIMIT 10;
```

[558]:

```
item.sort_values(by='购买量', ascending=False).reset_index().head(10)
```

[558]:

```
    ItemId  浏览量  购买量   购买转化率
0  3147410    4    4  100.0%
1  4443059   20    3   15.0%
2  1728241    1    3  300.0%
3  3964583   11    3   27.3%
4  4438744    2    3  150.0%
5  3202399    3    3  100.0%
6  1633923    6    2   33.3%
7  4992519    2    2  100.0%
8  1169462    6    2   33.3%
9  4499425    1    2  200.0%
```

[559]:

```
item.sort_values(by='浏览量', ascending=False).reset_index().head(10)
```

[559]:

```
    ItemId  浏览量  购买量 购买转化率
0   812879   68    0  0.0%
1  3845720   47    0  0.0%
2  1535294   39    0  0.0%
3    59883   35    0  0.0%
4  2367945   34    0  0.0%
5  2338453   33    1  3.0%
6  3520504   32    0  0.0%
7  2818406   32    0  0.0%
8  3920968   32    0  0.0%
9  2032668   31    0  0.0%
```

0.4 结论总结

1. 网站流量

- 网站整体流量略微偏低，可能由于十一月末的促销活动导致第二个周末（12-02 到 12-03）的流量大于第一个周末（11-25 到 11-26）。总体上看，人均约 1 的浏览量贡献了其 2%（0.02）的购买量。从细节上可以进行销售归因，查明销售来源从而加大投入以争取更多的销售额。
- 按日期统计，PV 和 UV 较为接近，UV 略低。2017-11-26 到 2017-12-01 为周一到周五，由于是工作日而整体流量较低，其中周二的流量最低，而从周五开始的周末流量迅速攀升达到最大值。
- 按时间统计，PV 和 UV 几乎一致，一天之内从半夜 12 点到凌晨 4 点流量下降至谷底；从凌晨 4 点到上午 10 点流量快速爬升，从 10 点到下午 6 点近似工作时间的范围内保持相对稳定在中位；然后从下午 6 点到晚上 9 点继续快速升高到高点，然后在 9 点到 10 点的黄金时间保持在高位；10 点后流量快速下降。总体上流量和工作，学习和作息的时间高度吻合，且黄金流量时段位于工作日每日的 9-10 点间，钻石流量位于周末的 9-10 点间，商家和电商可在这段时间推出新品，优惠和各种增流促销活动，以最大化流量和营收。
- 每日上午的 6-10 点和下午 6 点到晚上 10 点，流量迅速增加，一部分可归因为人员通勤带来的智能机流量，在这段时间内，商家可以加强新品上架，广告宣传和优惠促销等引流活动，尤其在移动端加大引流力度，来持续保持流量持续的快速增加。

2. 用户行为

- 网页浏览量远高于其他几项用户行为，包括放入购物车、收藏和购买等行为的数量。
- 从跳失率上看，整体的跳失率较高为 87.9%；在非跳失率的计算中，从用户行为数量比例的分析和 UV 数量比例的分析有一定差值。UV 数量比例虽然相对来讲高过用户行为数量比例 0.4-0.9 个百分点，但大体来讲这两组数据揭示了相同的流量分流的组成，均呈现购物车转化率 > 收藏转化率 > 购买转化率。
- 从行为维度来看，购物车转化率为 6.2%，收藏转化率为 3.2%，购买转化率为 2.2%，三者差值依次为 3%、1%；从不重复用户维度来看，购物车转化率为 7.1%，收藏转化率为 3.7%，购买转化率为 2.6%，三者差值依次为 3.4%、1.1%；如果把这三个行为看成时间上具有延续的事件的话，则最大的差值来源于从购物车到收藏这一步，则作为电商营运在提升加入购物车的转化率时，还应该从商品价格，页面信息等领域考虑如何让更多用户将更多商品加入收藏从而达到反复购买。
- 留存率可以从流量或 UV 的维度计算，留存率较低普遍在 3-4%，相对而言作为计算伊始的首日流量特别大所以经计算后的留存率均较低。电商营运可以尝试发起活动或周期性每天不一样的优惠促销来增强用户黏性，进而普遍提升留存率。
- 通过 RF 值的计算，为数据中的用户贴上了客户分类标签，便于之后进一步针对不同类型的用户制定相应的战略规划。

3. 商品类目

- 从这里的数据中可以看出，浏览量高的商品购买量并不高，相反购买量较高的商品并不一定都有较高的浏览量。这可能是由于这里使用的数据是时间上的切片，所以购买量高的商品应该是过去就比较受欢迎的商品。
- 对于可以连带绑定的商品或类目，可以尝试由受欢迎的商品带动新品的方式来提升新品的受欢迎程度，从而达到扩大销售类目和商品数量，进而扩大营收的目的。