电商数据分析

2020 年 6 月 11 日

电商数据分析

[108]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[109]:
```
# 加载数据，加载之前先用文本编辑器看下数据的格式，首行是什么，分隔符是什么等
df = pd.read_csv('./order_info_2016.csv', index_col='id')  #  使用文件内容的 id 列作为索引
```

[110]:
```
df.head()
```

[110]:

id	orderId	userId	productId	cityId	price	payMoney	channelId	deviceType	createTime	payTime
1	232148841	2794924	268	110001	35300	35300	9058255c90	3	2016-01-01 12:53:02	2016-01-01 12:53:24
2	222298971	1664684	801	330001	51200	49900	e0e6019897	2	2016-01-01 21:42:51	2016-01-01 21:43:30
3	211494392	2669372	934	220002	62100	62100	9058255c90	3	2016-01-01 14:10:13	2016-01-01 14:11:18
4	334575272	1924727	422	230001	50600	42000	46d5cea30d	2	2016-01-01 17:43:35	2016-01-01 17:43:53
5	144825651	4148671	473	130006	149100	142000	6ff1752b69	2	2016-01-01 18:52:04	2016-01-01 18:52:47

[111]:
```
df.info()  # 查看数据的行列数量以及数据类型
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 104557 entries, 1 to 104557
Data columns (total 10 columns):
orderId       104557 non-null int64
userId        104557 non-null int64
productId     104557 non-null int64
cityId        104557 non-null int64
price         104557 non-null int64
payMoney      104557 non-null int64
channelId     104549 non-null object
deviceType    104557 non-null int64
createTime    104557 non-null object
payTime       104557 non-null object
dtypes: int64(7), object(3)
memory usage: 8.8+ MB
```

[112]:
```
df.count()  # 查看各个列的非空数据量
```

[112]:
```
orderId       104557
userId        104557
productId     104557
cityId        104557
price         104557
payMoney      104557
channelId     104549
deviceType    104557
createTime    104557
payTime       104557
dtype: int64
```

[113]:
```
df.describe()
```

[113]:

	orderId	userId	productId	cityId	price	\
count	1.045570e+05	1.045570e+05	104557.000000	104557.000000	1.045570e+05	
mean	2.993341e+08	3.270527e+06	504.566275	154410.947225	9.167350e+04	
std	5.149818e+07	4.138208e+07	288.130647	72197.163762	9.158836e+04	
min	1.035627e+08	2.930600e+04	0.000000	30000.000000	6.000000e+02	

```
25%    2.633627e+08  2.179538e+06     254.000000  100011.000000  3.790000e+04
50%    2.989828e+08  2.705995e+06     507.000000  150001.000000  5.920000e+04
75%    3.349972e+08  3.271237e+06     758.000000  220002.000000  1.080000e+05
max    4.871430e+08  3.072939e+09    1000.000000  380001.000000  2.295600e+06
```

```
            payMoney     deviceType
count   1.045570e+05  104557.000000
mean    8.686689e+04       2.385292
std     9.072028e+04       0.648472
min    -1.000000e+03       1.000000
25%     3.360000e+04       2.000000
50%     5.500000e+04       2.000000
75%     1.040000e+05       3.000000
max     2.294200e+06       6.000000
```

[114]:
```
# 加载 device_type
device_type = pd.read_csv('./device_type.txt')
```

[115]:
```
device_type
```

[115]:
```
   id deviceType
0   1         PC
1   2    Android
2   3     iPhone
3   4        Wap
4   5      other
```

0.0.1 开始进行数据清洗

[116]:
```
# order_id
# order_id 应该是唯一的，查看是否有重复值
df.orderId.unique().size
```

[116]:
```
104530
```

0.0.2 order_id 列有重复值，处理完其它类型的问题后再删除

[117]:
```
# userId
# 对于订单数据，一个用户有可能有多个订单，重复值是合理的
df.userId.unique().size
```

[117]: 102672

[118]:
```
# productId
# productId 最小值是 0，查看值为 0 的记录数量
df.productId[(df.productId == 0)].size
```

[118]: 177

[119]:
```
# 删除异常值
df.drop(index=df[df.productId==0].index, inplace=True)
```

[120]:
```
# cityId
# cityId 值都在正常范围，不需要处理，有重复值是正常的
df.cityId.unique().size
```

[120]: 331

[121]:
```
# price
# price 没有空值，且都大于 0，单位是分，把它变成元
df.price = df.price / 100
```

[122]:
```
# payMoney
# payMoney 有负值，下单不可能是负值，所以对于负值的记录要删除掉
# 查看负值的记录
df[df.payMoney < 0]
```

[122]:

id	orderId	userId	productId	cityId	price	payMoney	channelId
25344	288096069	2145955	511	230014	111.0	-1000	df9f79c426
55044	296608442	4073997	385	120006	111.0	-1000	cbccc0808e
66897	316685479	1949907	554	60019	2084.0	-1000	9058255c90
72556	323229098	2894042	548	220008	114.0	-1000	41a4e91d29
81494	311194385	3370399	797	40001	116.0	-990	ea5648bbe2

```
87878  324068352  1873953        602  260003   292.0     -1000  9058255c90
```

```
       deviceType           createTime              payTime
id
25344           2  2016-03-11 21:32:54  2016-03-11 21:33:10
55044           2  2016-05-24 08:52:04  2016-05-24 08:52:23
66897           3  2016-03-31 11:57:46  2016-03-31 11:57:46
72556           2  2016-08-09 14:24:13  2016-08-09 14:24:26
81494           1  2016-09-06 11:20:43  2016-09-06 11:21:31
87878           3  2016-10-05 10:47:05  2016-10-05 10:47:05
```

[123]:
```
# 删除负值的记录
df.drop(index=df[df.payMoney < 0].index, inplace=True)
```

[124]:
```
# 再看下，已经没有了
df[df.payMoney < 0].index
```

[124]:
```
Int64Index([], dtype='int64', name='id')
```

[125]:
```
# 变成元
df.payMoney = df.payMoney / 100
```

[126]:
```
# channelId
# channelId 根据 info 的结果，有些 null 的数据，可能是客户端的 bug 等原因，在下单的时候没有传 channelId 字段
# 数据量大的时候，删掉少量的 null 记录不会影响统计结果
# 查看为 null 的数据
df[df.channelId.isnull()]
```

[126]:
```
          orderId   userId  productId  cityId   price  payMoney channelId  \
id
19086   284008366  3309847        698  240001  2164.0    2040.0       NaN
38175   287706890  2799815        823   70001   760.0     749.0       NaN
48073   248057459  3970570        142  130001   474.0     400.0       NaN
75949   266847859  3761925        649  120006   257.0     257.0       NaN
100952  283627429  4156620        269  280001   484.0     410.0       NaN
100953  346836140  3751526        738  100013   105.0      80.0       NaN
100954  352853915  2229389        786  240001   474.0     440.0       NaN
```

```
100955  379473081  4531810         18  180009   146.0      50.0       NaN
```

id	deviceType	createTime	payTime
19086	2	2016-03-08 22:36:12	2016-03-08 22:36:50
38175	3	2016-06-10 22:30:18	2016-06-10 22:30:47
48073	2	2016-03-30 12:59:03	2016-03-30 12:59:13
75949	2	2016-08-19 08:46:22	2016-08-19 08:46:39
100952	2	2016-12-13 13:24:37	2016-12-13 14:47:08
100953	1	2016-12-13 13:47:34	2016-12-13 13:47:44
100954	2	2016-12-13 16:54:09	2016-12-13 16:55:00
100955	3	2016-12-13 20:18:22	2016-12-13 20:18:34

[127]:
```
# 删除
df.drop(index=df[df.channelId.isnull()].index, inplace=True)
```

[128]:
```
# 再查看
df[df.channelId.isnull()]
```

[128]:
```
Empty DataFrame
Columns: [orderId, userId, productId, cityId, price, payMoney, channelId,
deviceType, createTime, payTime]
Index: []
```

[129]:
```
# deviceType 的取值与 device_type.txt 文件对比后没有问题，不需要处理
```

[130]:
```
# createTime 和 payTime 都没有 null，不过是要统计 2016 年的数据，把非 2016 年的删掉
# 先把 createTime 和 payTime 转换成 datetime 格式
df.createTime = pd.to_datetime(df.createTime)
df.payTime = pd.to_datetime(df.payTime)
df.dtypes
```

[130]:
```
orderId               int64
userId                int64
productId             int64
cityId                int64
price               float64
payMoney            float64
```

```
channelId             object
deviceType             int64
createTime    datetime64[ns]
payTime       datetime64[ns]
dtype: object
```

[131]:
```
import datetime
startTime = datetime.datetime(2016, 1, 1)
endTime = datetime.datetime(2016, 12, 31, 23, 59, 59)
```

[132]:
```
df[df.createTime < startTime]
```

[132]:

id	orderId	userId	productId	cityId	price	payMoney	channelId	deviceType	createTime	payTime
53	263312190	2497737	583	180015	336.0	336.0	9058255c90	3	2015-08-14 09:40:34	2016-01-01 14:47:14
18669	188208169	3277974	82	60021	752.0	740.0	9058255c90	3	2015-11-02 20:17:25	2016-01-19 20:06:35
36650	254118088	1861372	64	210010	249.0	249.0	9058255c90	2	2015-02-14 12:20:36	2016-02-28 13:38:41
71638	203314910	3207233	302	180011	662.0	580.0	41a4e91d29	1	2015-11-19 10:36:39	2016-08-07 12:24:35
88692	283989279	2830413	290	330005	467.0	460.0	df9f79c426	2	2015-12-26 11:19:16	2016-10-01 07:42:43

0.0.3 有 16 年之前的数据，需要删掉

[133]:
```
df.drop(index=df[df.createTime < startTime].index, inplace=True)
```

[134]:
```
df[df.createTime < startTime]  # 再查看是否有 16 年之前的数据
```

[134]:
```
Empty DataFrame
Columns: [orderId, userId, productId, cityId, price, payMoney, channelId,
deviceType, createTime, payTime]
```

```
Index: []
```

[135]:
```
df.drop(index=df[df.createTime > df.payTime].index, inplace=True)  # payTime 早于 createTime 的也需要删掉
```

[136]:
```
# 处理 16 年之后的数据
df[df.createTime > endTime]
```

[136]:
```
Empty DataFrame
Columns: [orderId, userId, productId, cityId, price, payMoney, channelId, deviceType, createTime, payTime]
Index: []
```

[137]:
```
# 看下支付时间有没有 16 年以前的
df[df.payTime < startTime]
```

[137]:
```
Empty DataFrame
Columns: [orderId, userId, productId, cityId, price, payMoney, channelId, deviceType, createTime, payTime]
Index: []
```

[138]:
```
# 把 orderId 重复的记录删掉
df.drop(index=df[df.orderId.duplicated()].index, inplace=True)
```

[139]:
```
df.describe()
```

[139]:

	orderId	userId	productId	cityId
count	1.043290e+05	1.043290e+05	104329.000000	104329.000000
mean	2.993327e+08	3.271862e+06	505.441622	154432.178100
std	5.150568e+07	4.142725e+07	287.622456	72131.599487
min	1.035627e+08	2.930600e+04	1.000000	30000.000000
25%	2.633440e+08	2.179758e+06	255.000000	100011.000000
50%	2.989888e+08	2.706195e+06	508.000000	150001.000000
75%	3.350016e+08	3.271317e+06	759.000000	220002.000000
max	4.871430e+08	3.072939e+09	1000.000000	380001.000000

	price	payMoney	deviceType
count	104329.000000	104329.000000	104329.000000

mean	917.113669	869.043120	2.385358
std	916.240224	907.576637	0.648439
min	6.000000	0.000000	1.000000
25%	379.000000	336.000000	2.000000
50%	593.000000	550.000000	2.000000
75%	1080.000000	1040.000000	3.000000
max	22956.000000	22942.000000	6.000000

0.1 数据清洗完毕，开始分析

0.1.1 先看下数据的总体情况：总订单数，总下单用户，总销售额，有流水的商品数

[140]:
```
print(df.orderId.count())
print(df.userId.unique().size)
print(df.payMoney.sum())
print(df.productId.unique().size)
```

```
104329
102447
90666399.7
1000
```

0.1.2 按照商品的 **productId,** 看下商品销量的前十和后十个

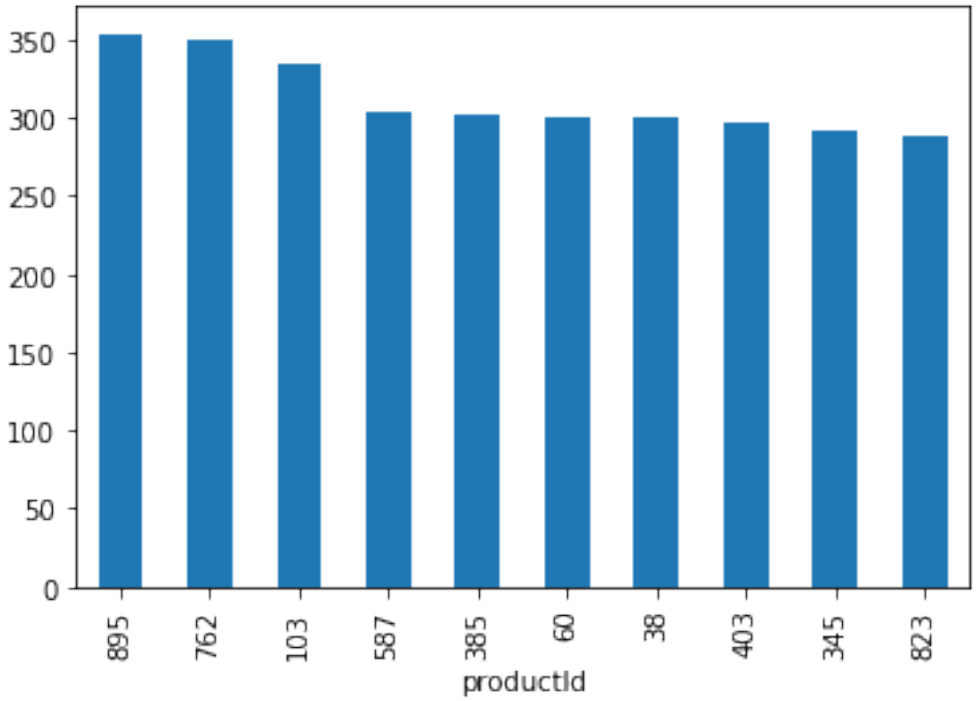
[146]:
```
productId_orderCount = df.groupby('productId').count()['orderId'].
 ↪sort_values(ascending=False)
print(productId_orderCount.head(10))  # 商品销量的前十
```

```
productId
895    354
762    350
103    334
587    303
385    302
60     301
38     301
403    297
```

```
345    292
823    288
Name: orderId, dtype: int64
```

[144]:
```
productId_orderCount.head(10).plot(kind='bar')
```

[144]: <matplotlib.axes._subplots.AxesSubplot at 0x13e04608>



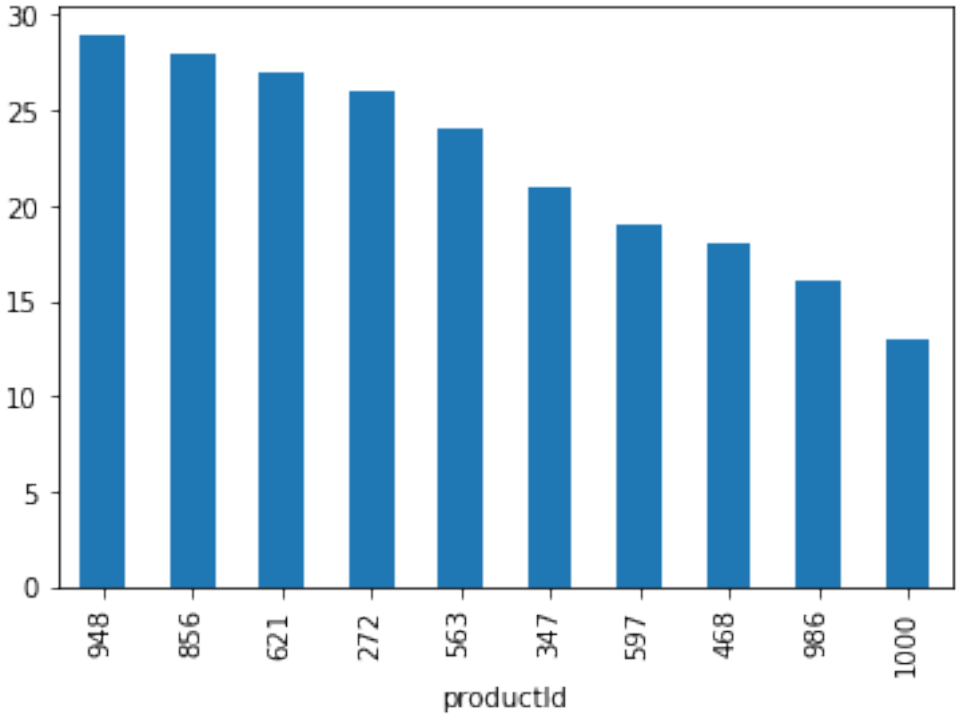
[147]:
```
print(productId_orderCount.tail(10))  # 商品销量的后十
```

```
productId
948      29
856      28
621      27
272      26
563      24
347      21
597      19
468      18
```

```
986     16
1000    13
Name: orderId, dtype: int64
```

[145]:
```
productId_orderCount.tail(10).plot(kind='bar')
```

[145]: <matplotlib.axes._subplots.AxesSubplot at 0x1242aa48>



0.1.3 按照商品的 productId, 看下商品销售额的前十和后十个

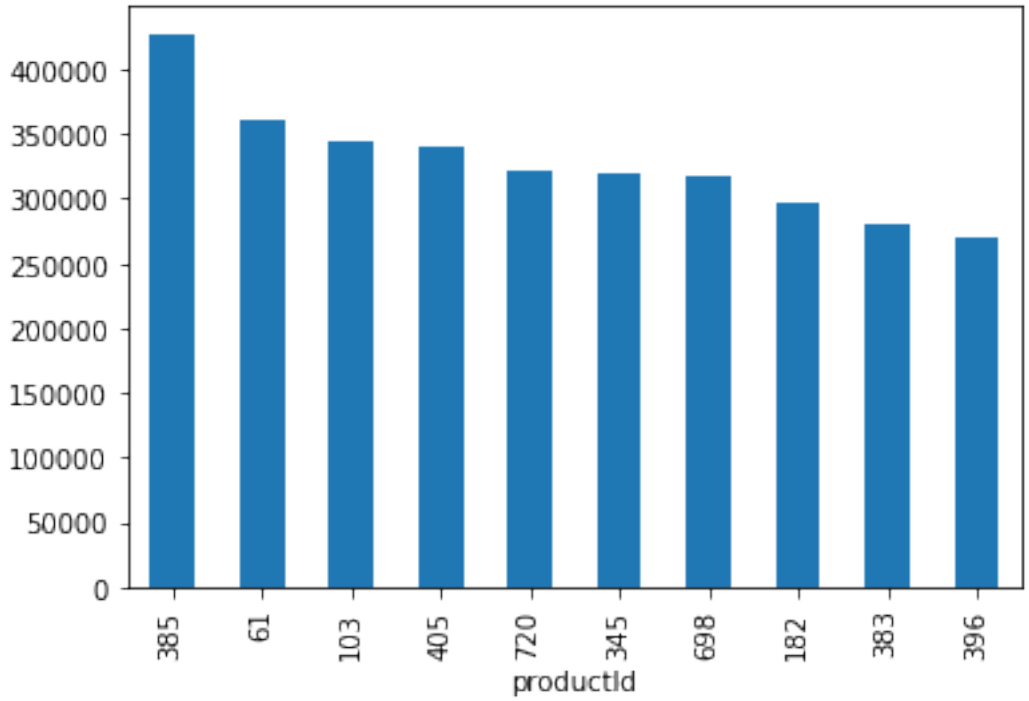
[148]:
```
productId_turnover = df.groupby('productId').sum()['payMoney'].
 ↪sort_values(ascending=False)
print(productId_turnover.head(10))  # 商品销售额的前十
```

```
productId
385     427522.1
61      361572.0
103     344641.2
405     339525.0
```

```
720    322405.1
345    320162.2
698    318458.6
182    296600.0
383    280790.0
396    269556.0
Name: payMoney, dtype: float64
```

[149]:
```
productId_turnover.head(10).plot(kind='bar')
```

[149]: <matplotlib.axes._subplots.AxesSubplot at 0x13bc5fc8>



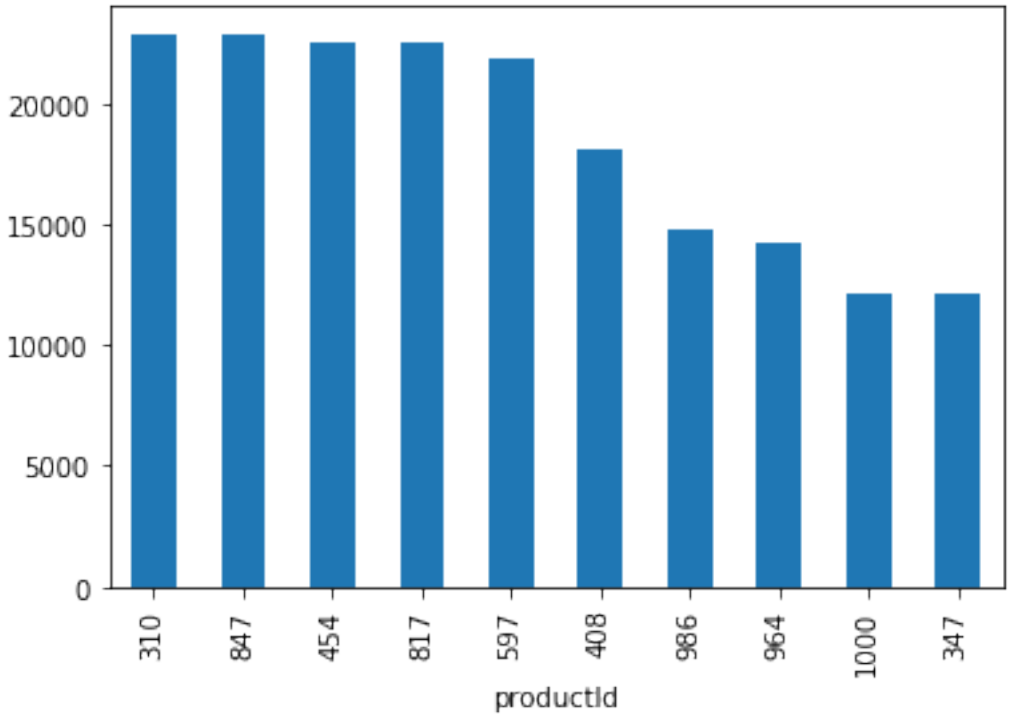
[150]:
```
print(productId_turnover.tail(10))  # 商品销售额的后十
```

```
productId
310     22879.0
847     22869.0
454     22535.0
817     22509.0
```

```
597     21847.0
408     18111.0
986     14784.0
964     14238.0
1000    12169.0
347     12070.0
Name: payMoney, dtype: float64
```

[151]:
```
productId_turnover.tail(10).plot(kind='bar')
```

[151]: <matplotlib.axes._subplots.AxesSubplot at 0x123234c8>



0.1.4 看下销量和销售额最后 100 个的交集，如果销量和销售额都不行，这些商品需要看看是不是要优化或者下架

[152]:
```
problem_productIds = productId_turnover.tail(100).index.
↪intersection(productId_orderCount.tail(100).index)
problem_productIds
```

[152]:
```
Int64Index([  14,  807,  599,  676,    7,  469,  577,  551,  318,  220,  528,
             303,  314,  359,  629,  582,  985,  218,  578,  227,  277,  145,
             855,  586,  958,   91,  856,  948,  859,  874,  806,  272,  392,
              27,  460,  436,  468,  579,  868,  137,   16,  590,  247,  569,
             242,  104,  621,  478,  310,  847,  454,  817,  597,  408,  986,
             964, 1000,  347],
            dtype='int64', name='productId')
```

0.1.5 按照城市分类, 看下商品销量的前十和后十的城市

[153]:
```
cityId_orderCount = df.groupby('cityId').count()['orderId'].
↪sort_values(ascending=False)
print(cityId_orderCount.head(10))
print(cityId_orderCount.tail(10))
```

```
cityId
110001    5490
130001    4100
60011     3639
40001     3291
220002    3051
230001    2951
240001    2753
120001    2435
220005    2168
70001     2075
Name: orderId, dtype: int64
cityId
280012    1
280010    1
280008    1
```

```
70002     1
240006    1
90024     1
180023    1
170031    1
160005    1
380001    1
Name: orderId, dtype: int64
```

0.1.6 按照城市分类, 看下商品销售额的前十和后十的城市

[154]:
```
cityId_payMoney = df.groupby('cityId').sum()['payMoney'].
 ↪sort_values(ascending=False)
print(cityId_payMoney.head(10))
print(cityId_payMoney.tail(10))
```

```
cityId
110001    6127732.8
220002    4406761.8
130001    3961264.7
220005    3374686.7
60011     2787478.1
40001     2531856.7
120001    2418538.8
70001     2192634.7
230001    2176484.0
220001    1983682.4
Name: payMoney, dtype: float64
cityId
310005    710.0
90030     700.0
280010    700.0
380001    667.0
360016    430.0
280012    420.0
90012     370.0
90029     177.0
```

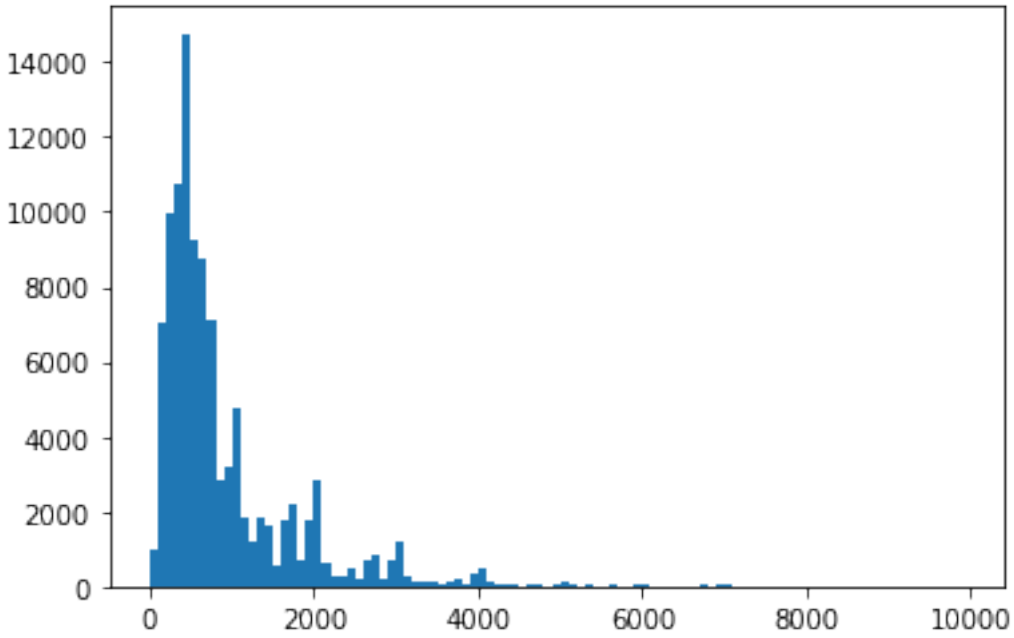
```
70002     110.0
180023     30.1
Name: payMoney, dtype: float64
```

0.1.7 对于价格，看下所有商品价格的分布，这样可以知道什么价格的商品卖的最好

[155]:
```
# price
# 先按照 100 的区间取分桶
bins = np.arange(0, 10000, 100)
pd.cut(df.price, bins).value_counts()
```

[155]:
```
(400, 500]      14791
(300, 400]      10737
(200, 300]       9966
(500, 600]       9189
(600, 700]       8777
                ...
(8200, 8300]        0
(9700, 9800]        0
(7700, 7800]        0
(7400, 7500]        0
(9800, 9900]        0
Name: price, Length: 99, dtype: int64
```

[157]:
```
# 直方图
# plt.figure(figsize=(16, 16))
plt.hist(df['price'], bins)
plt.show()
```

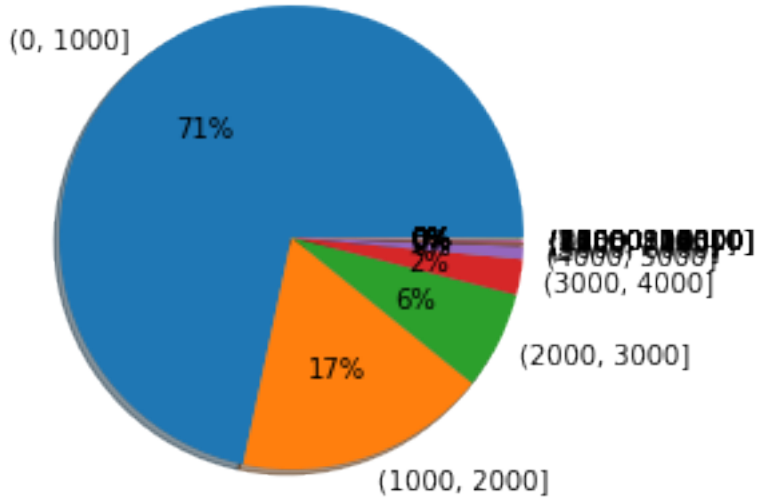


0.1.8 很多价格区间没有商品，如果有竞争对手的数据，可以看看是否需要补商品填充对应的价格区间

[158]:
```
price_cut_count = pd.cut(df.price, bins).value_counts()
zero_cut_result = (price_cut_count == 0)
zero_cut_result[zero_cut_result.values].index
```

[158]:
```
CategoricalIndex([(9600, 9700], (9500, 9600], (9400, 9500], (9300, 9400],
                  (8500, 8600], (8800, 8900], (7300, 7400], (8400, 8500],
                  (8300, 8400], (8200, 8300], (9700, 9800], (7700, 7800],
                  (7400, 7500], (9800, 9900]],
                 categories=[(0, 100], (100, 200], (200, 300], (300, 400], (400,
500], (500, 600], (600, 700], (700, 800], …], ordered=True, dtype='category')
```

[159]:
```
# 按照 1000 分桶再看下
bins = np.arange(0, 25000, 1000)
price_cut = pd.cut(df.price, bins).value_counts()
# 看看 1000 分桶的时候 5000 以下的饼图
plt.pie(x=price_cut.values, labels=price_cut.index, autopct='%d%%', shadow=True)
plt.show()
```

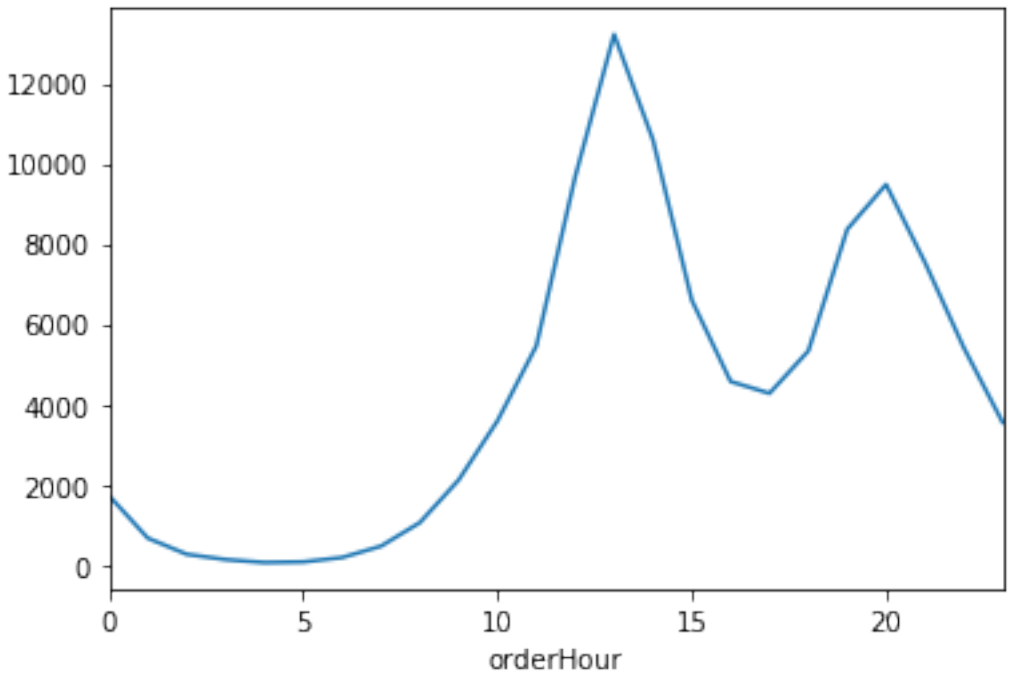


[160]:
```
# channelId
# 渠道的分析类似于 productId，可以给出成交量最多的渠道，订单数最多的渠道等，渠道很多时候是需要花钱买流量的，
# 所以还需要根据渠道的盈利情况和渠道成本进行综合比较，同时也可以渠道和商品等多个维度综合分析，看看不同的卖的最好的商品是否相同
```

0.1.9 按小时的下单量分布，可以按时间做推广，中午 12，13，14 点下单比较多，应该是午休的时候，然后是晚上 20 点左右，晚上 20 点左右几乎是所有互联网产品的一个高峰，下单高峰要注意网站的稳定性、可用性

[161]:
```
# 下单时间分析
df['orderHour'] = df.createTime.dt.hour
df.groupby('orderHour').count()['orderId'].plot()
```

[161]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15ae5c48>
```



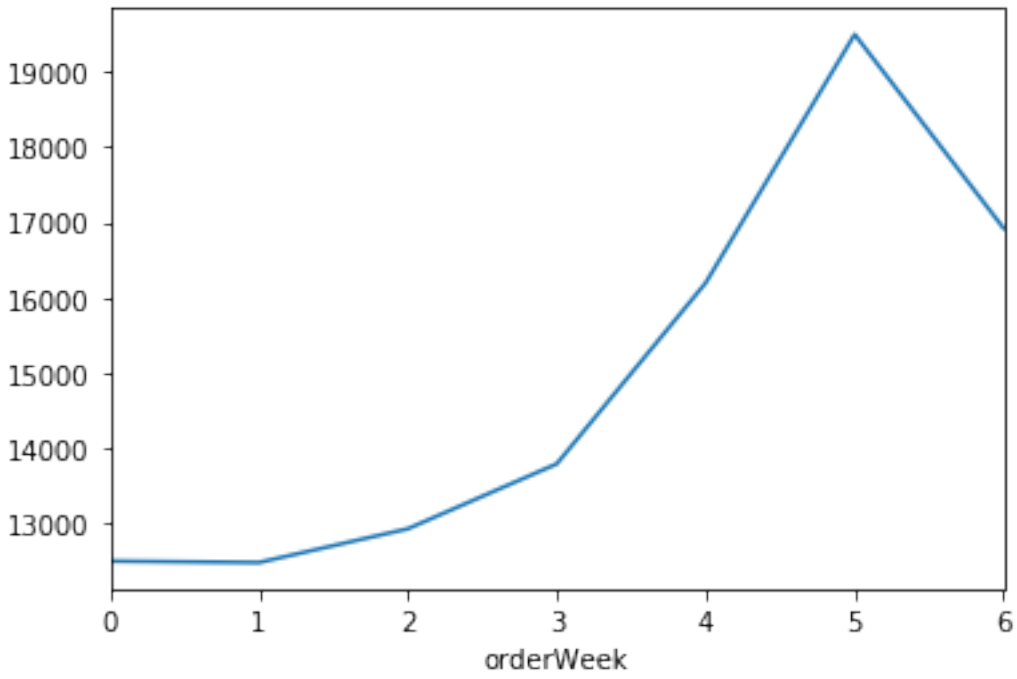
0.1.10 按照星期来看，周五下单最多，其次是周六

[162]:
```
df['orderWeek'] = df.createTime.dt.dayofweek
df.groupby('orderWeek').count()['orderId']
```

[162]:
```
orderWeek
0    12503
1    12484
2    12932
3    13794
4    16198
5    19496
6    16922
Name: orderId, dtype: int64
```

[163]:
```
df.groupby('orderWeek').count()['orderId'].plot()
```

[163]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b41608>
```



[164]:
```
# 下单后多久支付
def get_seconds(x):
    return x.total_seconds()
df['payDelta'] = (df['payTime'] - df['createTime']).apply(get_seconds)
```

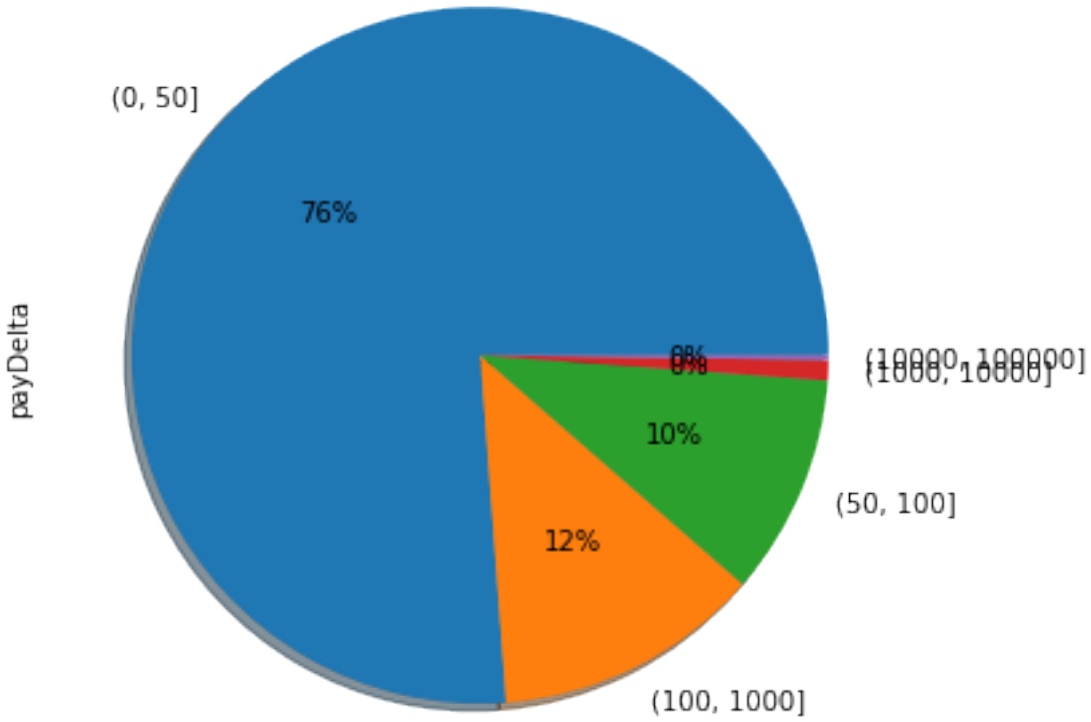
[166]:
```
bins = [0, 50, 100, 1000, 10000, 100000]
pd.cut(df.payDelta, bins).value_counts()
```

[166]:
```
(0, 50]             79229
(100, 1000]         12899
(50, 100]           10674
(1000, 10000]         968
(10000, 100000]       231
Name: payDelta, dtype: int64
```

0.1.11 绝大部分都在十几分钟之内支付完成，说明用户基本很少犹豫，购买的目的性很强

[167]:
```
# 饼图看下
pd.cut(df.payDelta, bins).value_counts().plot(kind='pie', autopct='%d%%',␣
↪shadow=True, figsize=(6, 6))
```

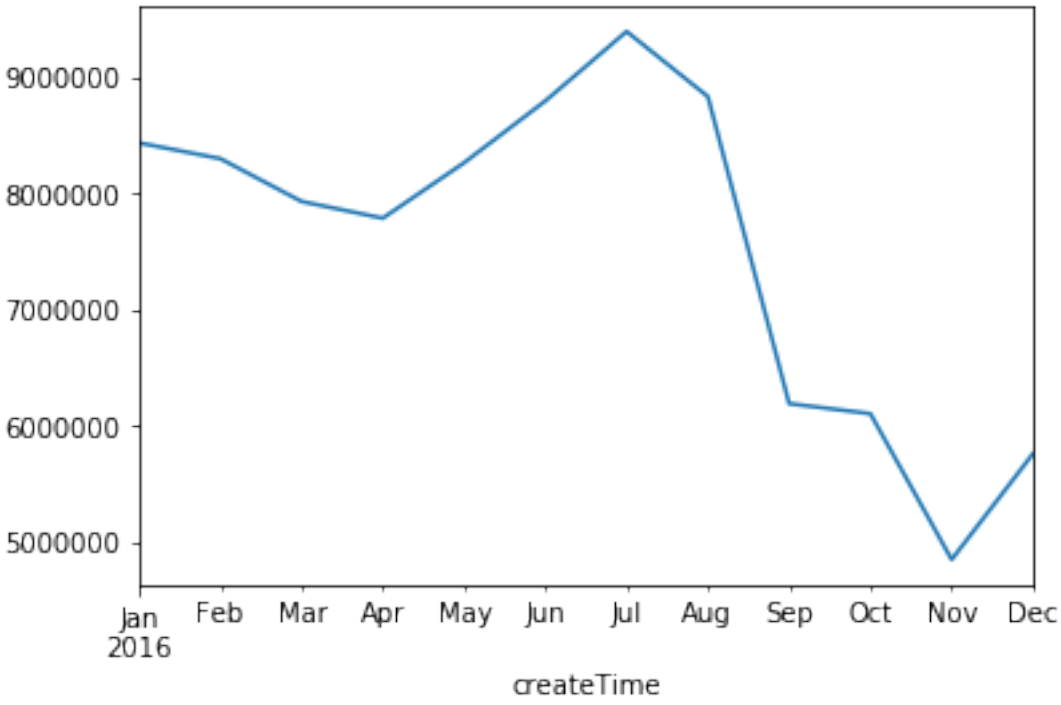
[167]: <matplotlib.axes._subplots.AxesSubplot at 0x15a053c8>



0.1.12 每月的成交额，最大值出现在七月

[168]:
```
# 月成交额
# 先把创建订单的时间设置为索引
df.set_index('createTime', inplace=True)
turnover = df.resample('M').sum()['payMoney']
turnover.plot()
```

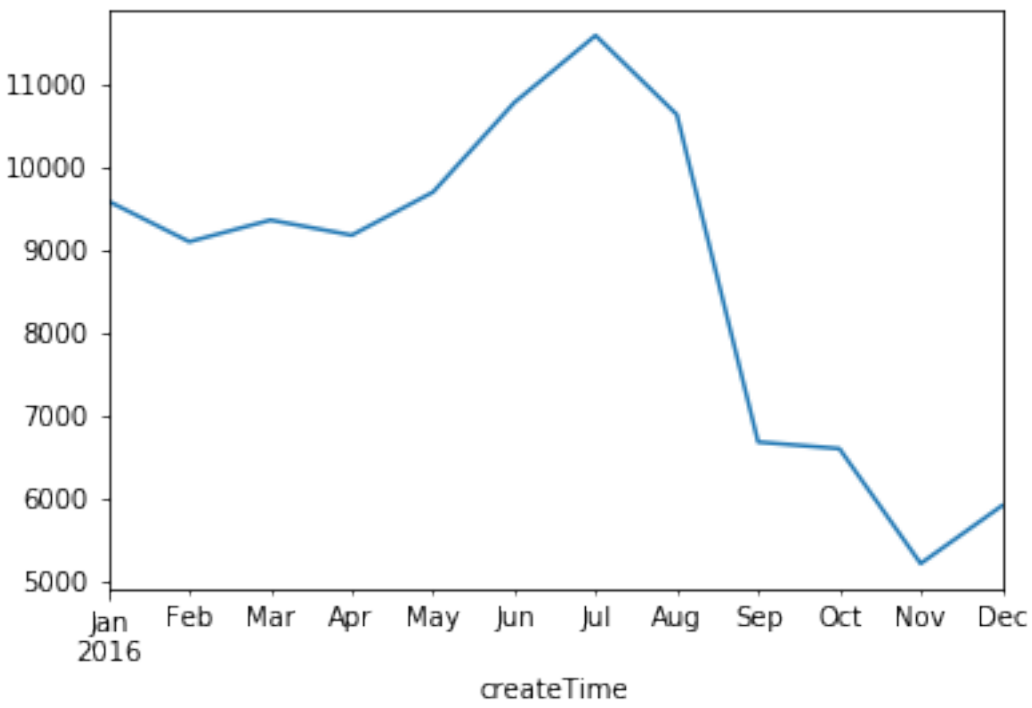
[168]: <matplotlib.axes._subplots.AxesSubplot at 0x15910e08>



0.1.13 每月的下单总数，最大值出现在七月

[169]:
```
order_count = df.resample('M').count()['orderId']
order_count.plot()
```

[169]: <matplotlib.axes._subplots.AxesSubplot at 0x159f41c8>


11000
10000
9000
8000
7000
6000
5000
Jan
2016
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
createTime