Predict Future Sales Solution

@lazyerIJ

Lists

1. File Description (1주차)
2. Evaluation Function (1주차)
3. EDA - basic, kernel (1주차, 2주차)
4. Pre processing(3주차)
5. Model
6. Tuning
7. Result

1. File Description

File descriptions

- sales_train.csv - the training set. Daily historical data from January 2013 to October 2015.
- test.csv - the test set. You need to forecast the sales for these shops and products for November 2015.
- sample_submission.csv - a sample submission file in the correct format.
- items.csv - supplemental information about the items/products.
- item_categories.csv - supplemental information about the items categories.
- shops.csv- supplemental information about the shops.

Data fields

- ID - an Id that represents a (Shop, Item) tuple within the test set
- shop_id - unique identifier of a shop
- item_id - unique identifier of a product
- item_category_id - unique identifier of item category
- item_cnt_day - number of products sold. You are predicting a monthly amount of this measure
- item_price - current price of an item
- date - date in format dd/mm/yyyy
- date_block_num - a consecutive month number, used for convenience. January 2013 is 0, February 2013 is 1,..., October 2015 is 33
- item_name - name of item
- shop_name - name of shop
- item_category_name - name of item category

Files : 6 files (1 for submission, 1 for test, 4 for train)
Format : csv

2. Evaluation Function

[Evaluation Function]

RMSE (Root Mean Square Error)

[Target]

각 item_id에 대하여 다음달에 각각의 shop_id에서 판매된 item_cnt_day 의 총 판매 수

(ex. **item_id 1** = [**shop_id==1 12월 매출**] + [**shop_id==2 12월 매출**] + … + [**shop_id==1234 12월 매출**])

[Point]

Time series data

```
ID,item_cnt_month
0,0.5
1,0.5
2,0.5
3,0.5
etc.
```

sample_submission.csv

3. EDA - basic

3-1) test.csv

[test.head]

```
test = pd.read_csv('../input/test.csv')
test.head(3)
```

	ID	shop_id	item_id
0	0	5	5037
1	1	5	5320
2	2	5	5233

```
print('[*]size of test : {}'.format(test.shape))
print('[*]nunique of shop_id : {}'.format(test.shop_id.nunique()))
print('[*]nunique of shop_id count : {}'\
      .format(test.groupby('shop_id').size().unique()))
```

```
[*]size of test : (214200, 3)
[*]nunique of shop_id : 42
[*]nunique of shop_id count : [5100]
```

[test.csv]

- Columns : ID, shop_id, item_id
- Data size : 214200
- 1 shop_id : 5100 rows
- Predict : **(shop_id, item_id) -> item_cnt_month**

3-2) item_categories.csv

[item_cats.head]

	item_category_name	item_category_id
0	PC - Гарнитуры/Наушники	0
1	Аксессуары - PS2	1
2	Аксессуары - PS3	2
3	Аксессуары - PS4	3
4	Аксессуары - PSP	4
5	Аксессуары - PSVita	5
6	Аксессуары - XBOX 360	6
7	Аксессуары - XBOX ONE	7
8	Билеты (Цифра)	8
9	Доставка товара	9
10	Игровые консоли - PS2	10
11	Игровые консоли - PS3	11
12	Игровые консоли - PS4	12
13	Игровые консоли - PSP	13
14	Игровые консоли - PSVita	14
15	Игровые консоли - XBOX 360	15
16	Игровые консоли - XBOX ONE	16
17	Игровые консоли - Прочие	17
18	Игры - PS2	18
19	Игры - PS3	19

type

subtype

[nunique]

	nunique
item_category_name	84
item_category_id	84

[item_category_name]

- ' - ' 으로 어떠한 구분이 이루어 진다.
- type은 같으나 subtype이 다른 경우가 존재한다.

3-2) item_categories.csv

[type & subtype 생성]

```
item_cats['split'] = item_cats['item_category_name'].str.split('-')
item_cats['type'] = item_cats['split']\
                    .map(lambda x: x[1].strip()
                         if len(x) > 1
                         else x[0].strip())
item_cats['subtype'] = item_cats['split'].map(lambda x: x[0].strip())
item_cats.drop(['split'], axis=1, inplace=True)
item_cats.head(5)
```

	item_category_name	item_category_id	type	subtype
0	PC - Гарнитуры/Наушники	0	Гарнитуры/Наушники	PC
1	Аксессуары - PS2	1	PS2	Аксессуары
2	Аксессуары - PS3	2	PS3	Аксессуары
3	Аксессуары - PS4	3	PS4	Аксессуары
4	Аксессуары - PSP	4	PSP	Аксессуары

[item_category_name]

- ‘ - ‘ 기준 분리 -> 뒤쪽=type -> 앞쪽=subtype
- type / subtype 칼럼 생성

3-2) item_categories.csv

```
item_cats[item_cats['type'] == item_cats['subtype']]
```

	item_category_name	item_category_id	type	subtype
8	Билеты (Цифра)	8	Билеты (Цифра)	Билеты (Цифра)
9	Доставка товара	9	Доставка товара	Доставка товара
32	Карты оплаты (Кино, Музыка, Игры)	32	Карты оплаты (Кино, Музыка, Игры)	Карты оплаты (Кино, Музыка, Игры)
79	Служебные	79	Служебные	Служебные
81	Чистые носители (шпиль)	81	Чистые носители (шпиль)	Чистые носители (шпиль)
82	Чистые носители (штучные)	82	Чистые носители (штучные)	Чистые носители (штучные)
83	Элементы питания	83	Элементы питания	Элементы питания

```
import re
pt = r'\([^)]*\)'
type_chk = item_cats['type'] == item_cats['subtype']
fix_index = item_cats[type_chk].index
item_cats.loc[fix_index, 'type'] = item_cats.loc[fix_index, 'type']\
                                   .apply(lambda x: re.sub(pattern=pt,
                                                           repl='',
                                                           string=x))
```

[item_category_name]

- '-' 가 존재하지 않는 데이터 -> type / subtype 구분 불가
- 괄호 제거 후 ' '(공백) 기준으로 구분

3-3) shops.csv

[shops.head]

	shop_name	shop_id
0	!Якутск Орджоникидзе, 56 фран	0
1	!Якутск ТЦ "Центральный" фран	1
2	Адыгея ТЦ "Мега"	2
3	Балашиха ТРК "Октябрь-Киномир"	3
4	Волжский ТЦ "Волга Молл"	4
5	Вологда ТРЦ "Мармелад"	5
6	Воронеж (Плехановская, 13)	6
7	Воронеж ТРЦ "Максимир"	7
8	Воронеж ТРЦ Сити-Парк "Град"	8
9	Выездная Торговля	9

[nunique]

	nunique
shop_name	60
shop_id	60

shop_type / some code

[shops.csv]

- Columns : shop_name, shop_id + **(city, shop_type 생성 가능성)**
- shop_name : shop_id = 1 : 1
- 60 unique shop_name, id

3-4) sales_train.csv

[sales_train.head]

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	02.01.2013	0	59	22154	999.00	1.0
1	03.01.2013	0	25	2552	899.00	1.0
2	05.01.2013	0	25	2552	899.00	-1.0
3	06.01.2013	0	25	2554	1709.05	1.0
4	15.01.2013	0	25	2555	1099.00	1.0
5	10.01.2013	0	25	2564	349.00	1.0
6	02.01.2013	0	25	2565	549.00	1.0
7	04.01.2013	0	25	2572	239.00	1.0
8	11.01.2013	0	25	2572	299.00	1.0
9	03.01.2013	0	25	2573	299.00	3.0

[sales_train.csv]

- Columns : 7
- (2935849, 6) 크기의 데이터
- date와 shop_id 에 따라 구분된다.

3-4) sales_train - date

```
sales_train.date = pd.to_datetime(sales_train.date, format='%d.%m.%Y')
```

```
print('[*]date')
min_date = sales_train.date.min()
max_date = sales_train.date.max()
period_days = (max_date - min_date).days
print('>>date nunique : {}'.format(sales_train.date.nunique()))
print('>>min date : {}'.format(min_date))
print('>>max date : {}'.format(max_date))
print('>>total period days : {} days'.format(period_days))
```

```
[*]date
>>date nunique : 1034
>>min date : 2013-01-01 00:00:00
>>max date : 2015-10-31 00:00:00
>>total period days : 1033 days
```

[sales_train - date]

- 2013. 01. 01 ~ 2015. 10. 31 데이터 존재
- 1034일의 데이터 (약 34달)

3-4) sales_train - date_block_num

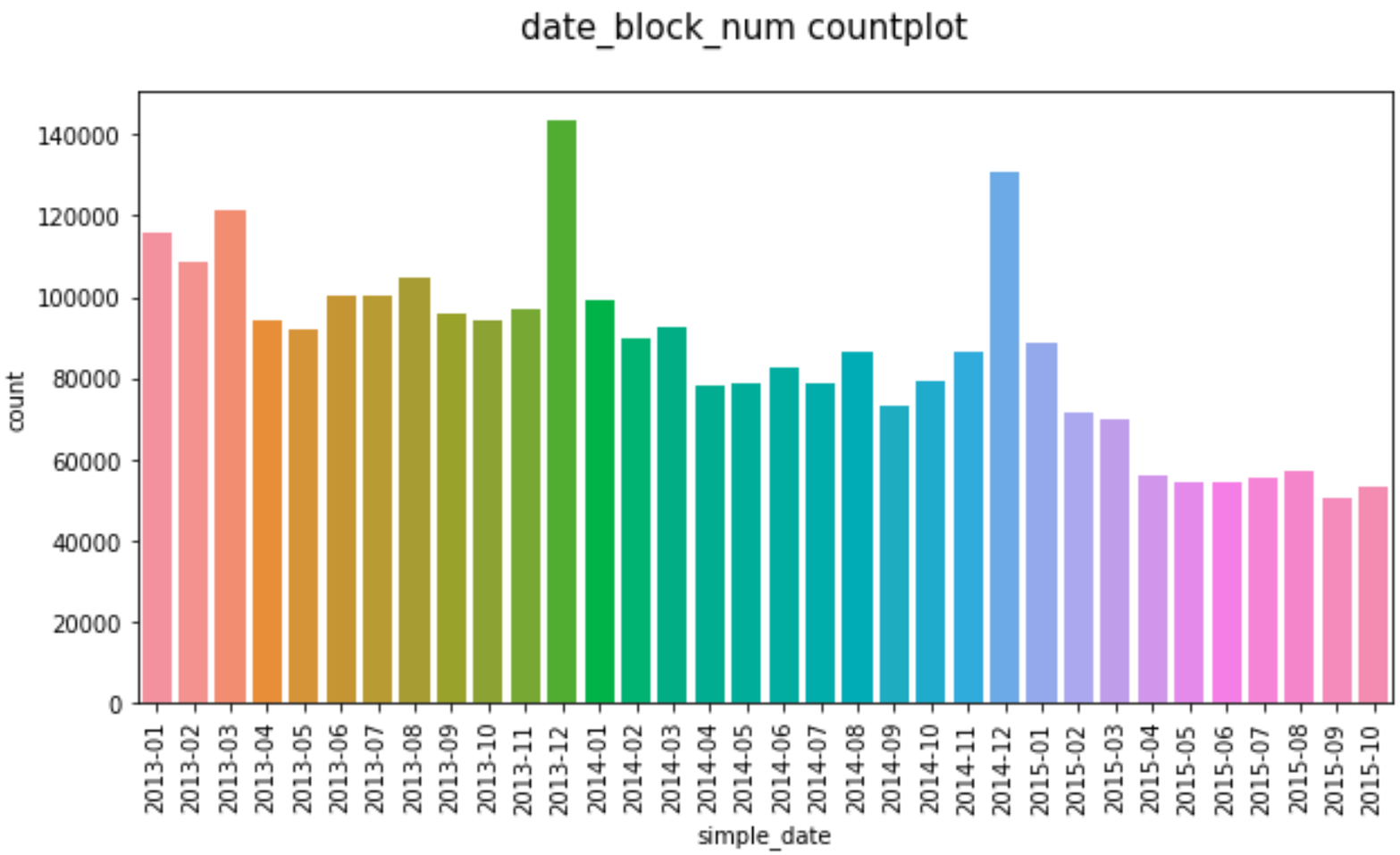
```
sales_train['simple_date'] = sales_train\
                            .date.apply(lambda x: x.strftime('%Y-%m'))
simple_date = [(a, b) for a, b\
               in zip(sales_train.simple_date,
                      sales_train.date_block_num)]
print('[*]nunique (simple_date & date_block_num) : {}'\
      .format(len(set(simple_date))))
```

```
[*]nunique (simple_date & date_block_num) : 34
```

[sales_train - date_block_num]

- date -> 년 / 월만 분리하여 simple_date 생성
- simple_date와 date_block_num = 1 : 1 대응
- date_block_num은 1달 기준 인 것을 확인

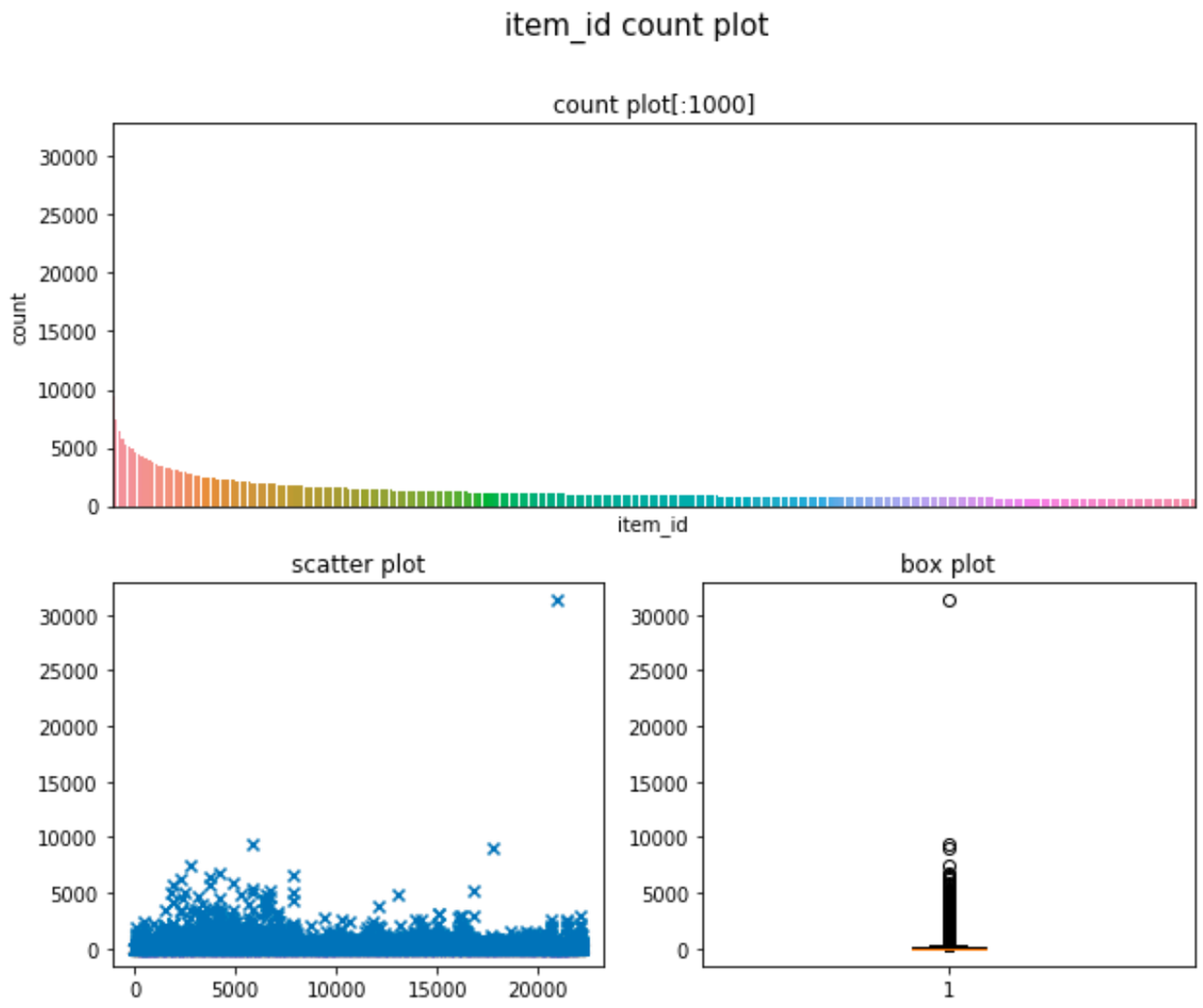
3-4) sales_train - date_block_num



[sales_train - date_block_num]

- 연말에 피크를 보인다.
- 총 판매량은 감소 추세이다.

3-4) sales_train - item_id



[sales_train - item_id]

- unique : 21807
- min : 0 / max : 22169
- max count : 31340 (item_id = 20949)

3-4) sales_train - item_id

```
sales_train.query('item_id==20949').drop_duplicates().head(10)
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	simple_date
352379	2013-04-30	3	22	20949	5.000000	12.0	2013-04
352399	2013-04-29	3	22	20949	4.923077	13.0	2013-04
352400	2013-04-28	3	22	20949	5.000000	4.0	2013-04
352401	2013-04-27	3	22	20949	5.000000	4.0	2013-04
352455	2013-04-24	3	22	20949	5.000000	2.0	2013-04
352456	2013-04-25	3	22	20949	4.875000	16.0	2013-04
352457	2013-04-26	3	22	20949	5.000000	15.0	2013-04
359513	2013-04-29	3	28	20949	5.000000	2.0	2013-04
359534	2013-04-30	3	28	20949	5.000000	5.0	2013-04
362755	2013-04-24	3	25	20949	5.000000	1.0	2013-04

[sales_train - item_id == 20949]

- 같은 item_id 이더라도 item_price는 다를 수 있다.
- outliers가 존재한다.

3-4) sales_train - item_price

```
max_size_item_price = sales_train.groupby('item_price').size()
max_size_item_price = max_size_item_price.sort_values().tail(1)
print(max_size_item_price)
max_size_item_price = max_size_item_price.values[0]
print('>>max size of item_price : {}'.format(max_size_item_price)
      , end=' ')
print('(percentage : {:.2f}%)'.format(max_size_item_price\
                                      /len(sales_train)\
                                      *100))
```

```
item_price
299.0    291352
dtype: int64
>>max size of item_price : 291352 (percentage : 9.92%)
```

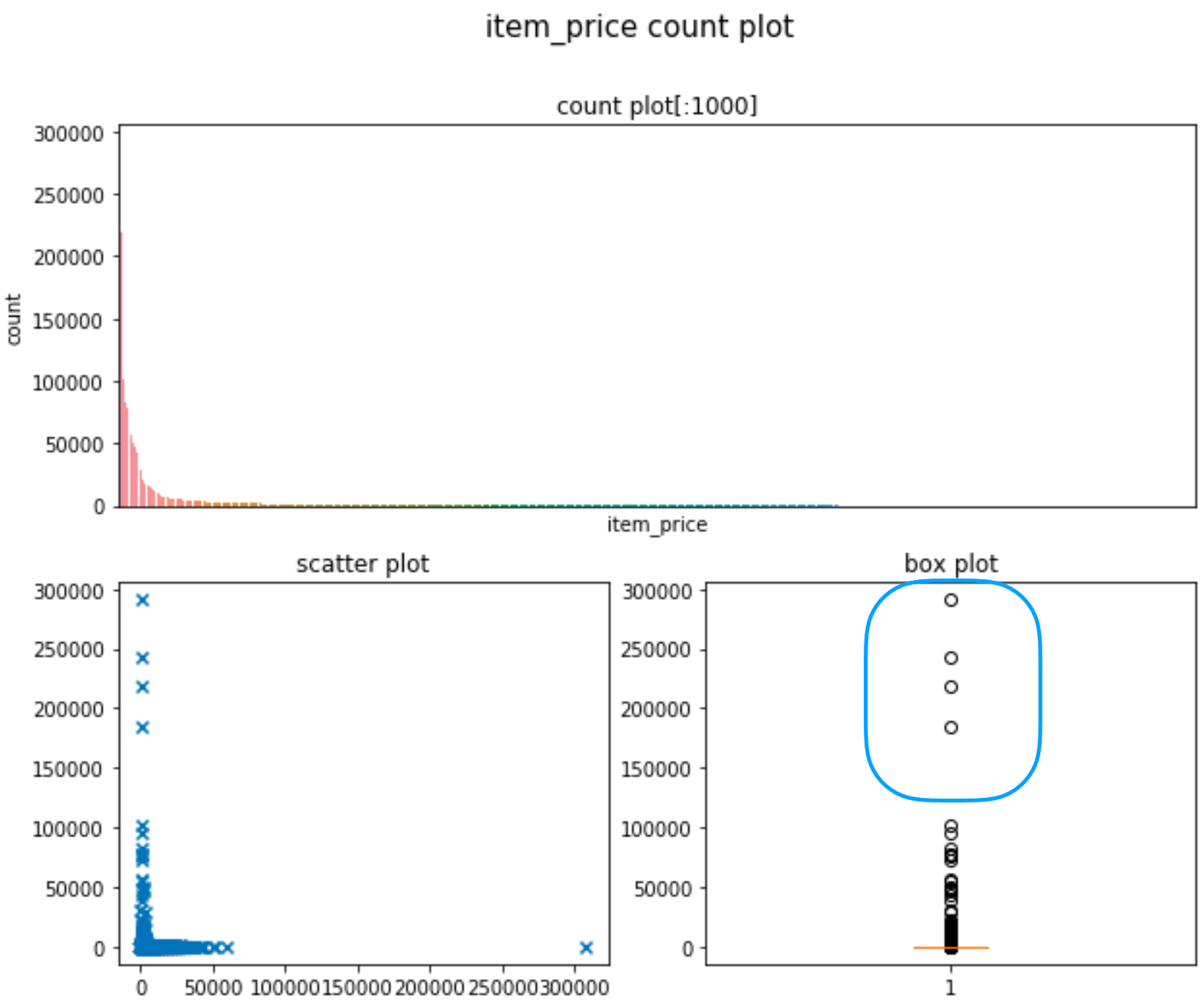
```
price_299_num = sales_train.query('item_price==299.0').item_id.nunique()
print('>>299.9 price item_id nunique : {}'.format(price_299_num))
```

```
>>299.9 price item_id nunique : 2749
```

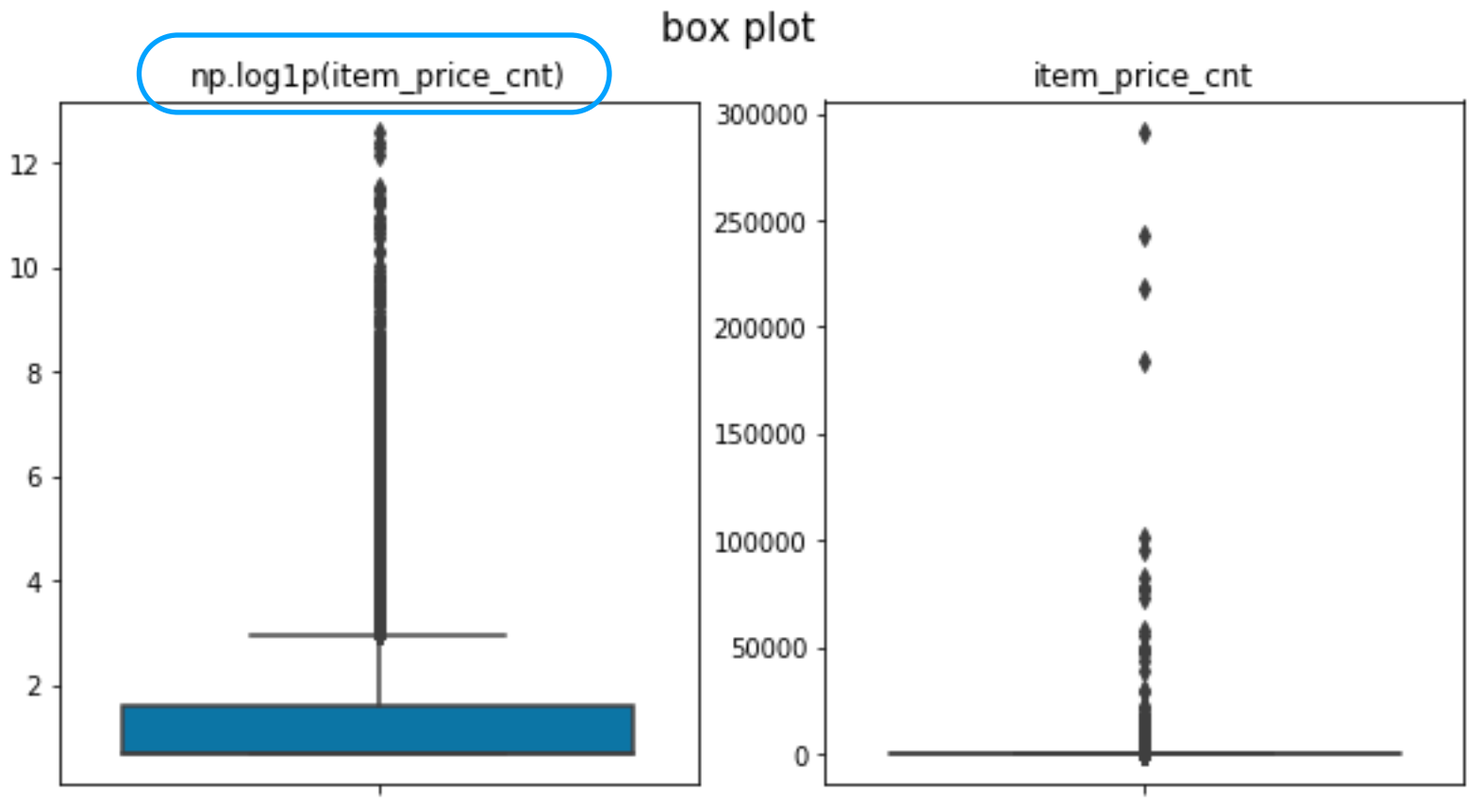
[sales_train - item_price]

- unique : 19993
- min : -1.0 / max : 307980.0

3-4) sales_train - item_price



3-4) sales_train - item_price



[sales_train - item_price]

- 몇몇 값의 차이가 매우 크다.
- 10% 가까이 같은 값을 가진다.
- -1 value가 존재한다.

3-4) sales_train - item_cnt_day

```
sales_train.item_cnt_day.describe()
```

```
count    2.935849e+06
mean     1.242641e+00
std      2.618834e+00
min     -2.200000e+01
25%      1.000000e+00
50%      1.000000e+00
75%      1.000000e+00
max      2.169000e+03
Name: item_cnt_day, dtype: float64
```

```
sales_train.query('item_cnt_day==-22')  #s Only 1
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	simple_date
1057907	2013-11-02	10	12	8023	15.0	-22.0	2013-11

```
sales_train.query('item_id==8023 and item_cnt_day < 0')
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	simple_date
1057907	2013-11-02	10	12	8023	15.0	-22.0	2013-11

```
sales_train.query('item_id==8023 and item_cnt_day > 10')
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
885224	25.09.2013	8	12	8023	15.0	22.0
1330814	27.01.2014	12	12	8023	15.0	20.0

3-4) sales_train - item_cnt_day

```
sales_train.query('item_cnt_day < 0').groupby('item_cnt_day').size()
```

```
item_cnt_day
-22.0       1
-16.0       1
-9.0        1
-6.0        2
-5.0        4
-4.0        3
-3.0       14
-2.0       78
-1.0     7252
dtype: int64
```

[sales_train - item_cnt_day]

- 음수 값의 경우 대부분 -1.0 값이다 (98.59%)
- 데이터는 대부분 1.0 값이다. (89.56%)
- 음수인지 아닌지 binary 형식으로 표현하는 것도 좋은 방법일 수 있다.

3-4) sales_train - item_cnt_day

```
print('>>The day the goods were not sold was not recorded.')
sales_train.groupby(['item_id', 'date_block_num']).count().head(5)
```

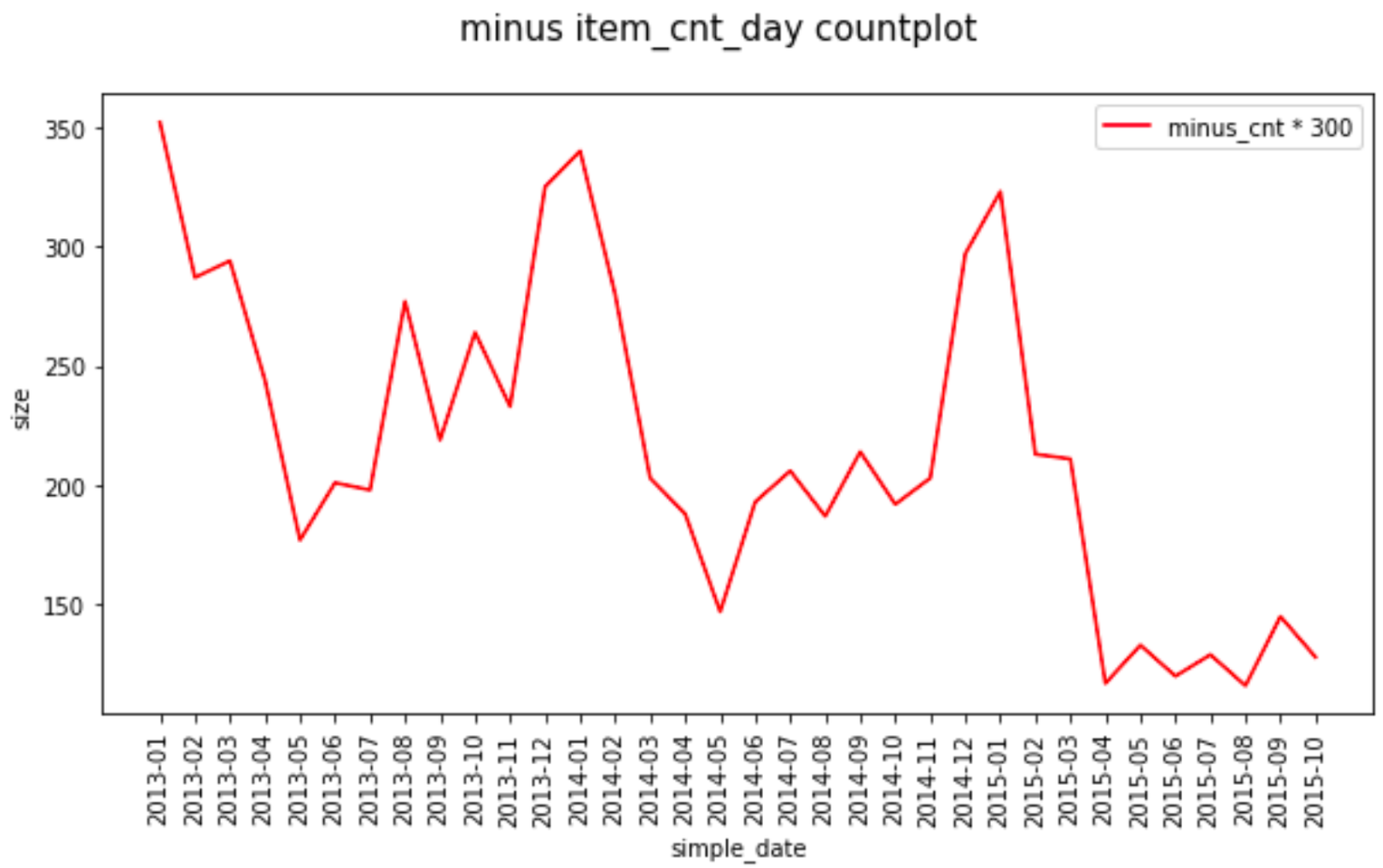
```
>>The day the goods were not sold was not recorded.
```

item_id	date_block_num	date	shop_id	item_price	item_cnt_day	simple_date
0	20	1	1	1	1	1
1	15	2	2	2	2	2
	18	1	1	1	1	1
	19	1	1	1	1	1
	20	1	1	1	1	1

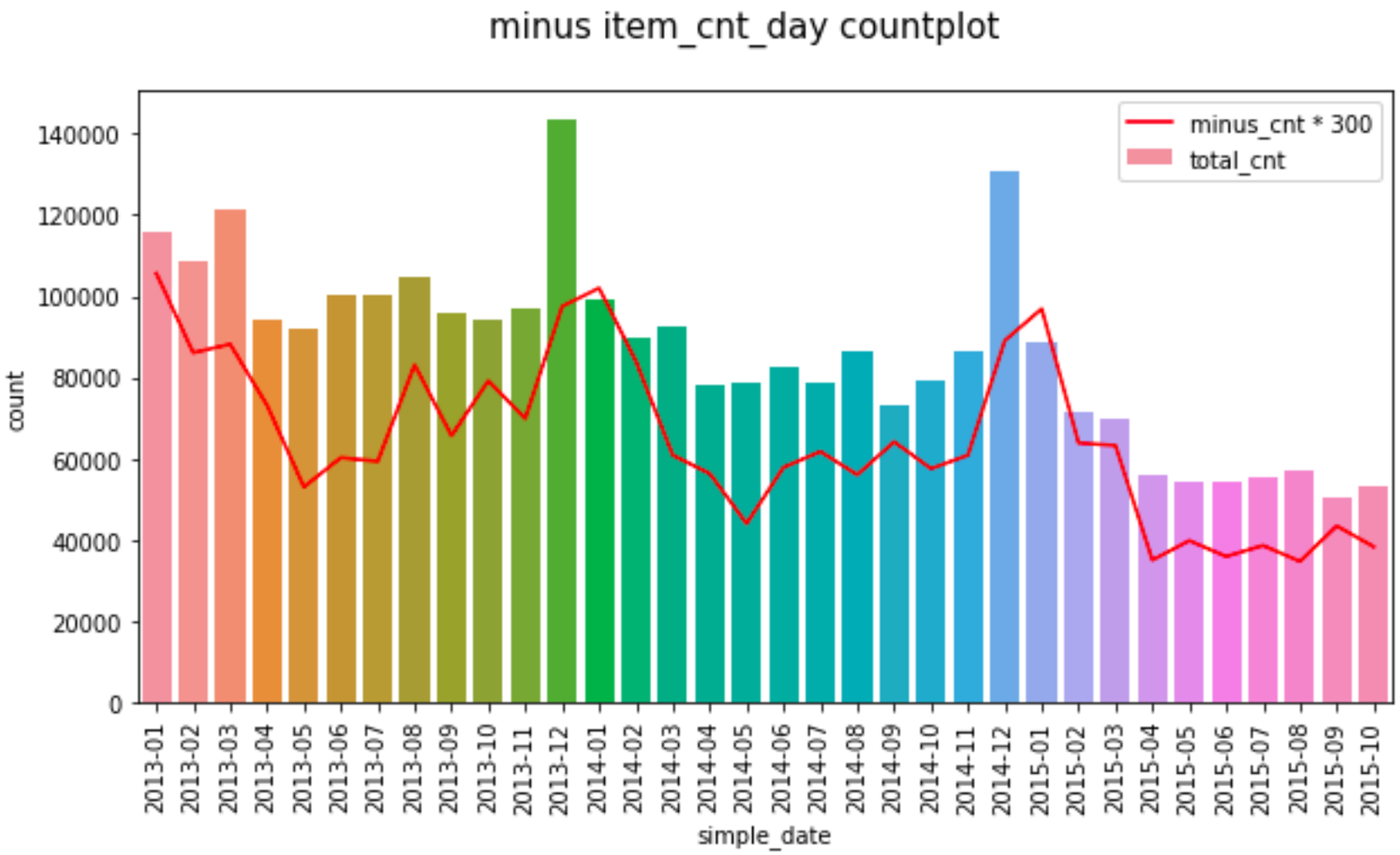
[sales_train - item_cnt_day]

- 해당 날짜에 판매가 이루어지지 않았을 경우 데이터 존재하지 않는다.

3-4) sales_train - item_cnt_day



3-4) sales_train - item_cnt_day



3-4) sales_train - item_cnt_day

```
date_format = '["2013-01-{}", "2014-01-{}"]'
test_format = 'date in {} and item_cnt_day<0'
print('[*]minus item_cny_day size in special date')
for date in ['01', '02', '03', '04', '05']:
    query_date = date_format.format(date, date)
    query = test_format.format(query_date)
    rs = sales_train.query(query).shape[0]
    print('>>01/{} is : {}'.format(date, rs))
```

```
[*]minus item_cny_day size in special date
>>01/01 is : 9
>>01/02 is : 51
>>01/03 is : 33
>>01/04 is : 34
>>01/05 is : 29
```

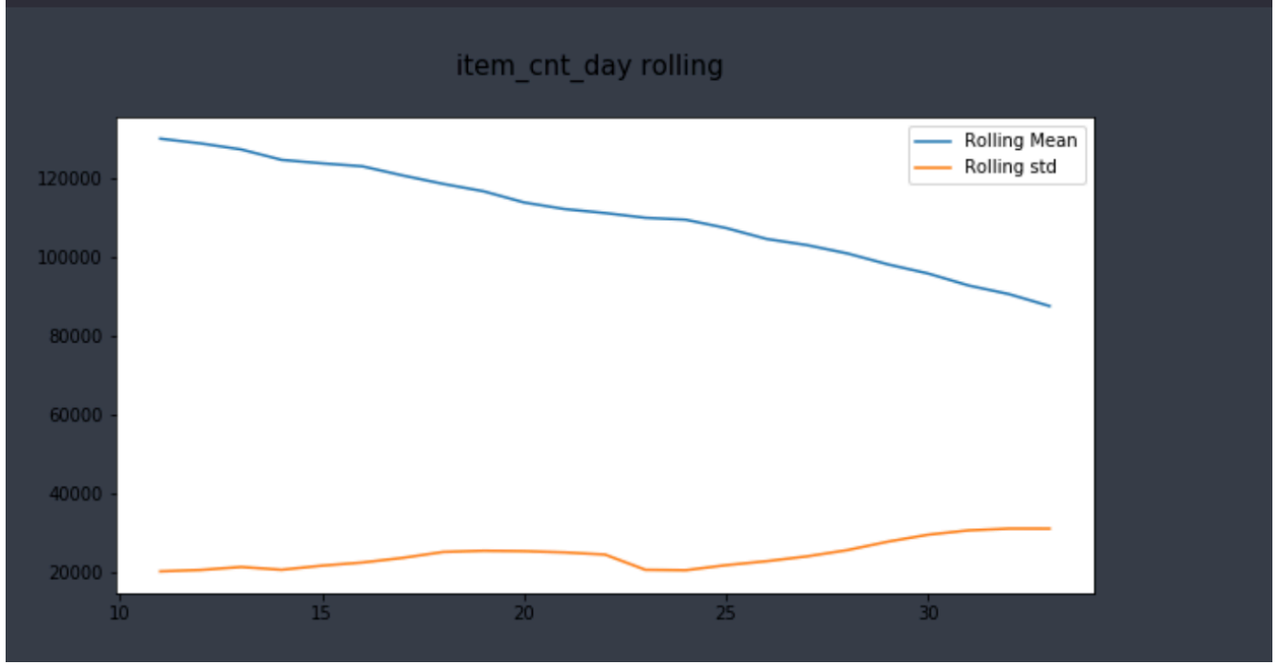
```
[*]minus item_cny_day size in special date
>>05/01 is : 14
>>05/02 is : 16
>>05/03 is : 10
>>05/04 is : 9
>>05/05 is : 7
```

[sales_train - item_cnt_day]

- 연말에 과소비를 하고 새해에 환불을 하는 경우가 많은 것으로 간주된다.

3-4) sales_train - item_cnt_day

```
rolling_mean = sales_train.groupby(["date_block_num"])
rolling_mean = rolling_mean["item_cnt_day"].sum()
plt.figure(figsize=(10, 5))
plt.suptitle('item_cnt_day rolling', fontsize=15)
plt.plot(rolling_mean.rolling(window=12,center=False).mean(),
         label='Rolling Mean')
plt.plot(rolling_mean.rolling(window=12,center=False).std(),
         label='Rolling std')
plt.legend()
plt.show()
```



[sales_train - item_cnt_day]

- **편차는 점점 증가하며, 이동평균은 감소하는 추세이다.**
- **(-> 추후 AR 모델에 필요한 정보)**

3-4) sales_train - item_cnt_day

[Summary]

- 1.0의 값이 90% 정도를 차지한다.
- 0.0의 값은 없다.
- - 0.25% 의 음수 값 존재 (98%의 -1.0 값을 가진다.)
- item_cnt_day, minus 값을 가진 item_cnt_day 모두 피크를 가진다.
- 이동평균은 부드러운 하향 추세이다 (window size = 12)

4) items.csv

```
items.head(10)
```

	item_name	item_id	item_category_id
0	! ВО ВЛАСТИ НАВАЖДЕНИЯ (ПЛАСТ.) D	0	40
1	!ABBYY FineReader 12 Professional Edition Full...	1	76
2	***В ЛУЧАХ СЛАВЫ (UNV) D	2	40
3	***ГОЛУБАЯ ВОЛНА (Univ) D	3	40
4	***КОРОБКА (СТЕКЛО) D	4	40
5	***НОВЫЕ АМЕРИКАНСКИЕ ГРАФФИТИ (UNI) ...	5	40
6	***УДАР ПО ВОРОТАМ (UNI) D	6	40
7	***УДАР ПО ВОРОТАМ-2 (UNI) D	7	40
8	***ЧАЙ С МУССОЛИНИ D	8	40
9	***ШУГАРЛЭНДСКИЙ ЭКСПРЕСС (UNI) D	9	40

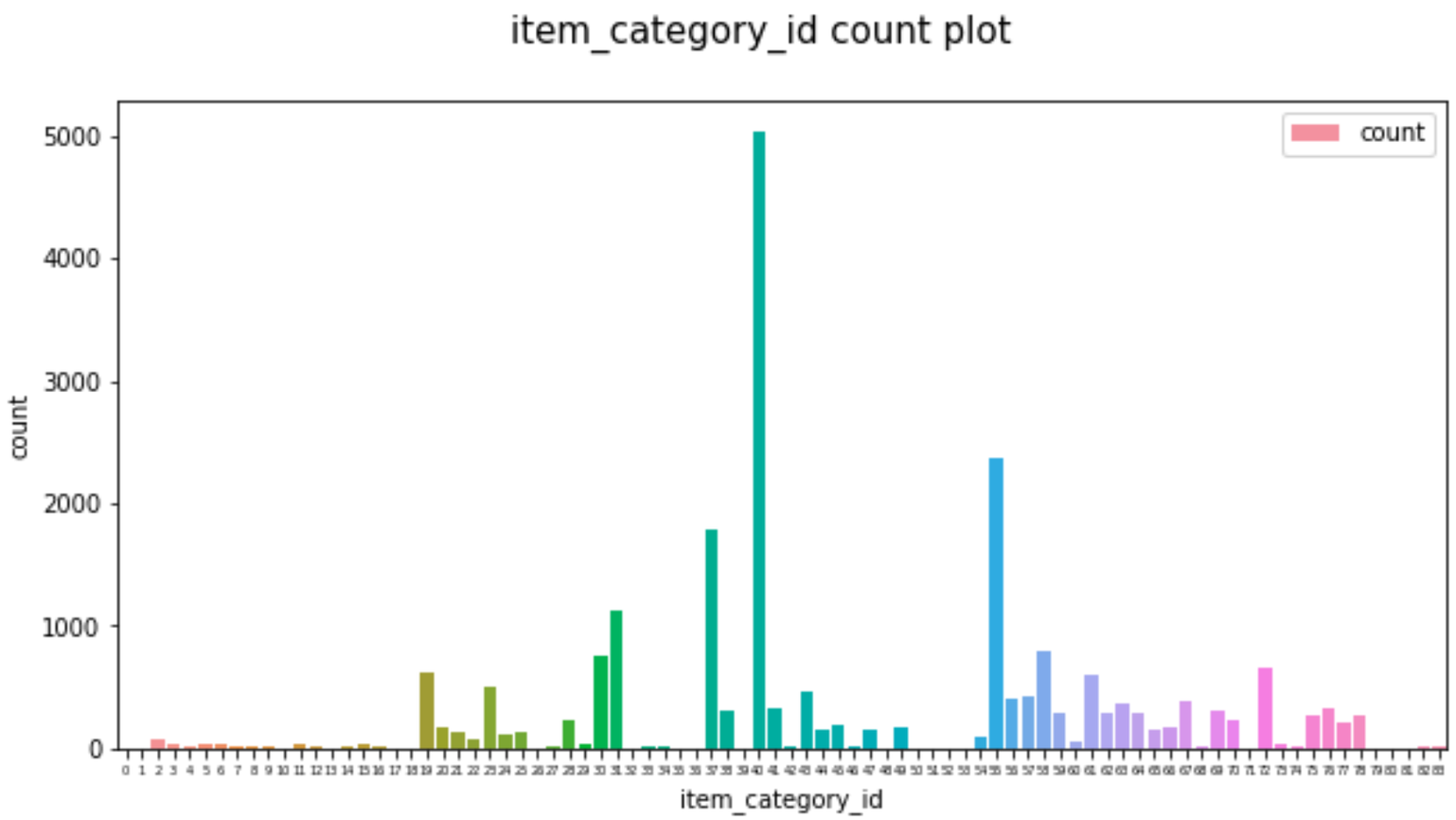
```
print('[*]items.shape : {}'.format(items.shape))
```

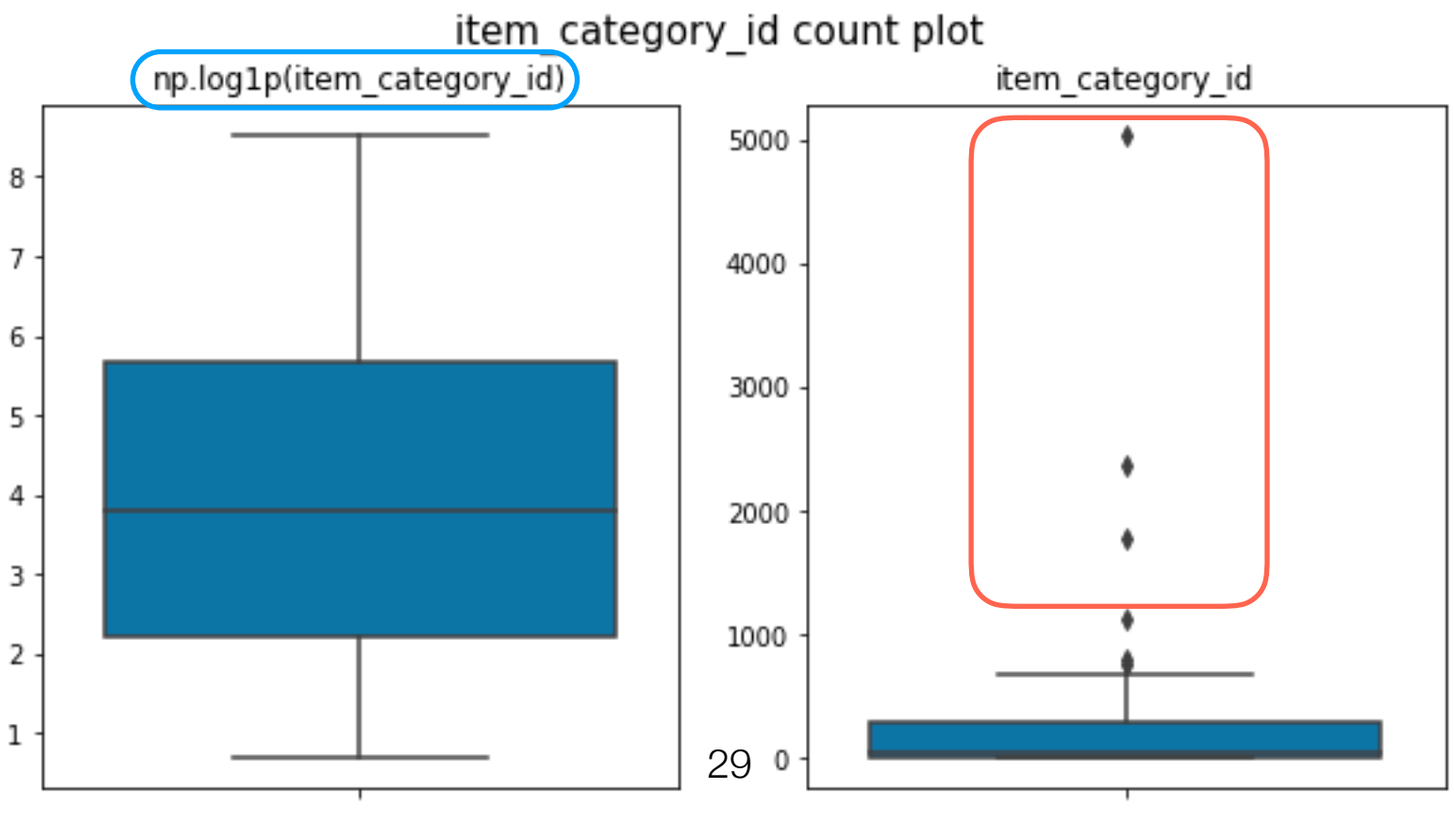
```
[*]items.shape : (22170, 3)
```

[item_name]

- item_category_name 과 마찬가지로 어떠한 조합으로 이루어져있다. (작은 의미)

4) items.csv


item_category_id count plot
count
5000
4000
3000
2000
1000
0
item_category_id


item_category_id count plot
np.log1p(item_category_id)
item_category_id
8
7
6
5
4
3
2
1
5000
4000
3000
2000
1000
0

4. kernel - EDA

1) Import and prepare data

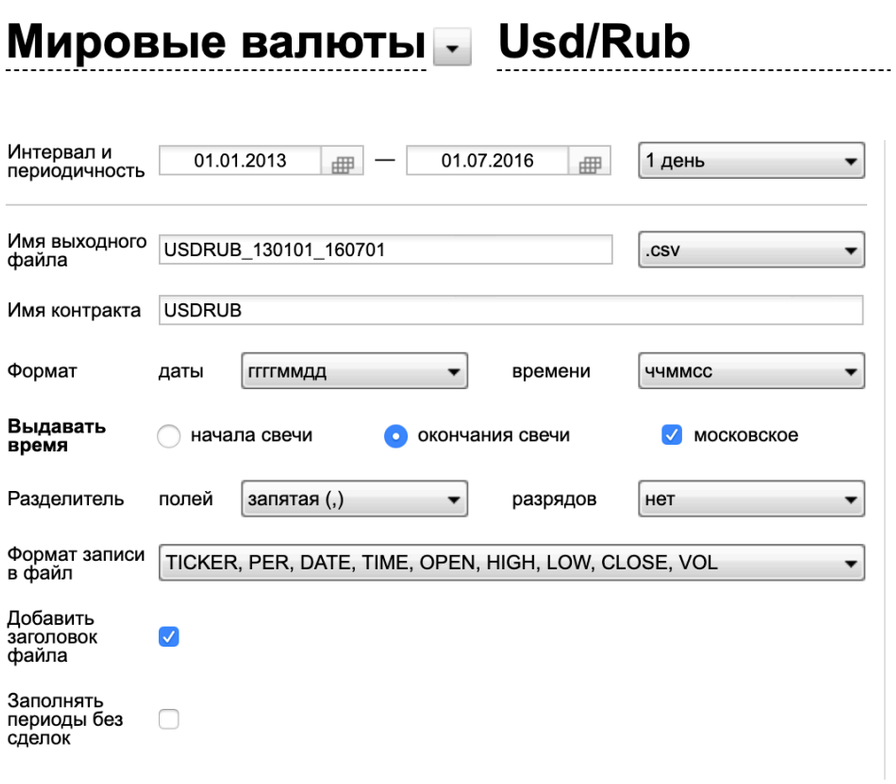
```
train_df = pd.read_csv('../input/sales_train.csv')
test_df = pd.read_csv('../input/test.csv')
item_df = pd.read_csv('../input/items.csv')
item_cat_df = pd.read_csv('../input/item_categories.csv')
shop_df = pd.read_csv('../input/shops.csv')
```

USD/RUB exchange rate taken from https://www.finam.ru/profile/forex/usd-rub/export/ :

```
exch_rate = pd.read_csv('../input/USDRUB_130101_160701.csv',
                        sep=',', parse_dates=['<DATE>'])
exch_rate = exch_rate.drop(['<TICKER>', '<PER>', '<TIME>'], axis=1)
exch_rate = exch_rate.rename(columns={'<DATE>':'date', '<CLOSE>':'rate'})
```

```
exch_rate.head(3)
```

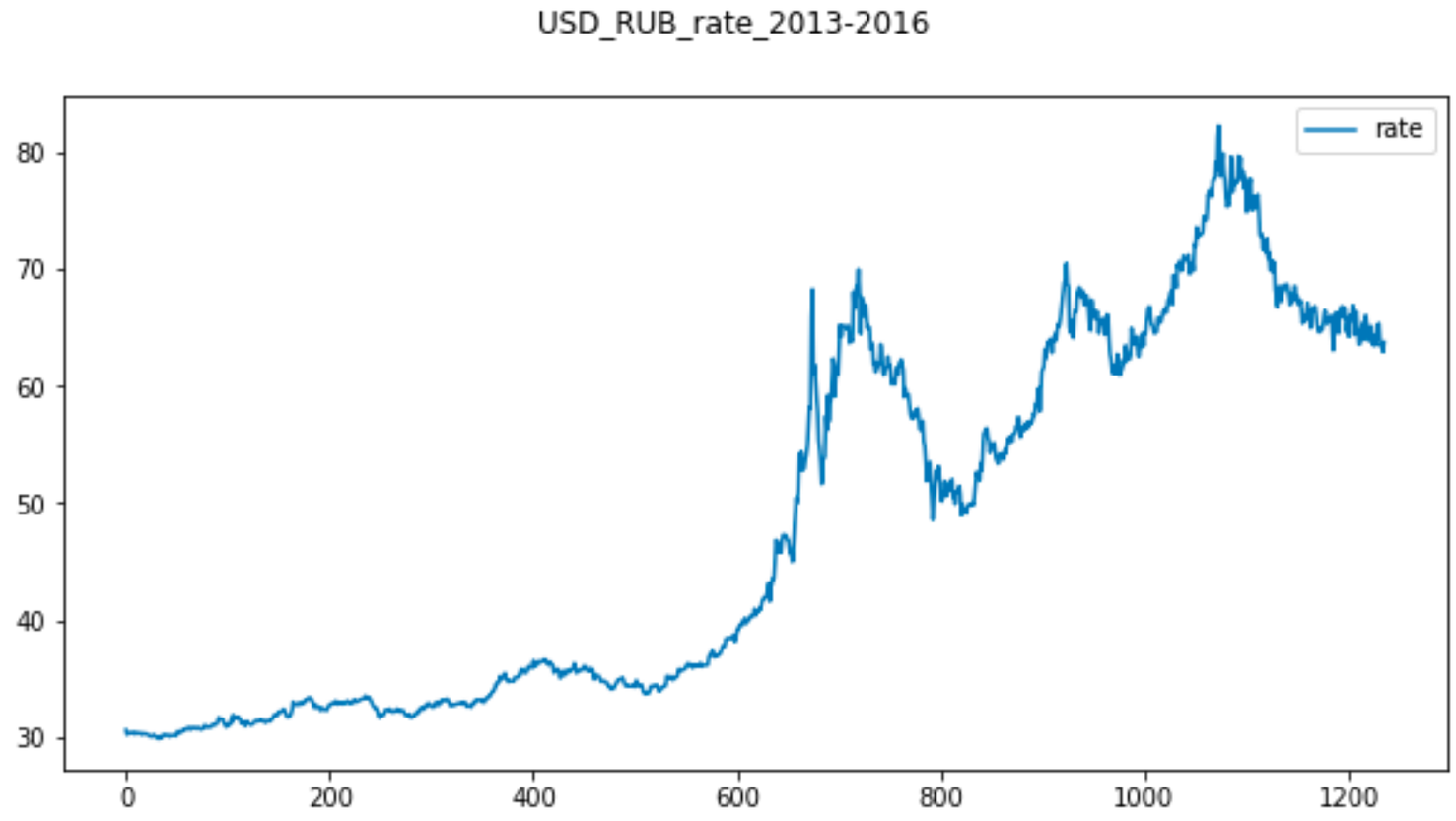
	date	<OPEN>	<HIGH>	<LOW>	rate	<VOL>
0	2013-01-01	30.56	30.61	30.49	30.57	0
1	2013-01-02	30.56	30.59	30.09	30.18	0
2	2013-01-03	30.19	30.39	30.12	30.25	0



[Code]

- 데이터 읽어오기
- 루블 - 달러 환율 파일 추가 (https://www.finam.ru/profile/forex/usd-rub/export/#)

1) Import and prepare data



1) Import and prepare data

```
start = time.time()
train_df['date_test'] = pd.to_datetime(train_df.date, format='%d.%m.%Y')
print(time.time() - start)
```

```
3.9262948036193848
```

```
start = time.time()
train_df['date'] = train_df['date']\
                   .parallel_apply(dateutil.parser.parse, dayfirst=True)
print(time.time() - start)
```

```
14.398569345474243
```

[parallel_apply]

- datetime format을 자동으로 parse 가능하나, 속도가 비교적 느린 편.

2) items data

```python
def name_correction(x):
    x = x.lower()
    x = x.partition('[')[0]
    x = x.partition('(')[0]
    x = re.sub('[^A-Za-z0-9А-Яа-я]+', ' ', x)
    x = x.replace('  ', ' ')
    x = x.strip()
    return x
```

```python
item_df['item_name'] = item_df['item_name'].apply(name_correction)
```

[Code]

- https://www.kaggle.com/kyakovlev/1st-place-solution-part-1-hands-on-data
- 소문자 치환 >> '[' 앞 부분 선택 >> '(' 앞 부분 선택 >> 문자 선택 >> 공백 제거
- 22170 > 18210 unique 감소

2) items data

[partition function]

```
a = 'abc[def]gh'
print('partition : ', a.partition('['))
print('[0] index : ', a.partition('[')[0])
```

```
partition :  ('abc', '[', 'def]gh')
[0] index :  abc
```

```
a = 'abc[def][gh]'
print('partition : ', a.partition('['))
print('[0] index : ', a.partition('[')[0])
```

```
partition :  ('abc', '[', 'def][gh]')
[0] index :  abc
```

2) item categories data

```
item_cat_df['new_cats'] = (item_cat_df['item_category_name']
                           .str.split(' - ').apply(lambda x: x[0]))
```

```
item_cat_df['new_subcats'] = (item_cat_df['item_category_name']
                           .str.split(' - ').apply(lambda x: x[-1]))
```

[Code]

- '-' 기준으로 split 하여 간단히 구별 >> new_cats, new_subcats 생성

3) shops data

```
shop_df.head(10)
```

	shop_name	shop_id
0	!Якутск Орджоникидзе, 56 фран	0
1	!Якутск ТЦ "Центральный" фран	1
2	Адыгея ТЦ "Мега"	2
3	Балашиха ТРК "Октябрь-Киномир"	3
4	Волжский ТЦ "Волга Молл"	4
5	Вологда ТРЦ "Мармелад"	5
6	Воронеж (Плехановская, 13)	6
7	Воронеж ТРЦ "Максимир"	7
8	Воронеж ТРЦ Сити-Парк "Град"	8
9	Выездная Торговля	9

[shop_name]

Балашиха ТРК “Октябрь-Киномир"

>> 지역 : Балашиха

>> 타입 : ТРК

>> 이름 : Октябрь-Киномир

공백 또는 괄호로 구분되어 진다.

3) shops data

```
shop_df['city'] = (shop_df['shop_name'].apply(name_correction)
                   .str.split(' ').apply(lambda x: x[0]))
```

```
shop_df['city'].value_counts()
```

```
москва          13
якутск           4
воронеж          3
тюмень           3
ростовнадону     2
н                2
казань           2
спб              2
новосибирск      2
жуковский        2
красноярск       2
уфа              2
самара           2
адыгея           1
ярославль        1
цифровой         1
омск             1
чехов            1
волжский         1
сергиев          1
томск            1
балашиха         1
сургут           1
коломна          1
курск            1
интернет         1
выездная         1
калуга           1
мытищи           1
вологда          1
химки            1
Name: city, dtype: int64
```

```
shop_df.loc[shop_df['city']=='н', 'city']
```

```
34    н
35    н
Name: city, dtype: object
```

```
shop_df.loc[shop_df['city']=='н', 'city'] = 'нижний_новгород'
```

[Code]

- nunique를 줄이려는 목적보다 알맞은 값으로 치환.

3) shops data

```
shop_df['shop_type'] = (shop_df['shop_name'].apply(name_correction)
                        .str.split(' ')\
                        .apply(lambda x: x[1] if (len(x)>1) else 'other'))
```

```
shop_df['shop_type'].value_counts()
```

```
тц              27
трц              9
тк               5
трк              5
орджоникидзе     2
магазин          2
новгород         2
ул               2
торговля         1
распродажа       1
мтрц             1
посад            1
other            1
склад            1
Name: shop_type, dtype: int64
```

[Code]

- ‘ ‘ (공백)을 기준으로 shop_type 생성 (결측값 처리 > ‘other’)

3) shops data

```
shop_df.loc[(shop_df['shop_type']=='новгород')|
            (shop_df['shop_type']=='мтрц'),:]
```

	shop_name	shop_id	city	shop_type
21	Москва МТРЦ "Афи Молл"	21	москва	мтрц
34	Н.Новгород ТРЦ "РИО"	34	н	новгород
35	Н.Новгород ТРЦ "Фантастика"	35	н	новгород

```
shop_df.loc[(shop_df['shop_type']=='новгород')|
            (shop_df['shop_type']=='мтрц'), 'shop_type'] = 'трц'
```

```
shop_df.loc[(shop_df['shop_type']=='посад'), :]
```

	shop_name	shop_id	city	shop_type
46	Сергиев Посад ТЦ "7Я"	46	сергиев	посад

```
shop_df.loc[(shop_df['shop_type']=='тц'), :].head(5)
```

	shop_name	shop_id	city	shop_type
1	!Якутск ТЦ "Центральный" фран	1	якутск	тц
2	Адыгея ТЦ "Мега"	2	адыгея	тц
4	Волжский ТЦ "Волга Молл"	4	волжский	тц
13	Казань ТЦ "Бехетле"	13	казань	тц
14	Казань ТЦ "ПаркХаус" II	14	казань	тц

```
shop_df.loc[(shop_df['shop_type']=='посад'),
            'shop_type'] = 'тц'
```

[Code]

- nunique를 줄이려는 목적보다 알맞은 값으로 치환.

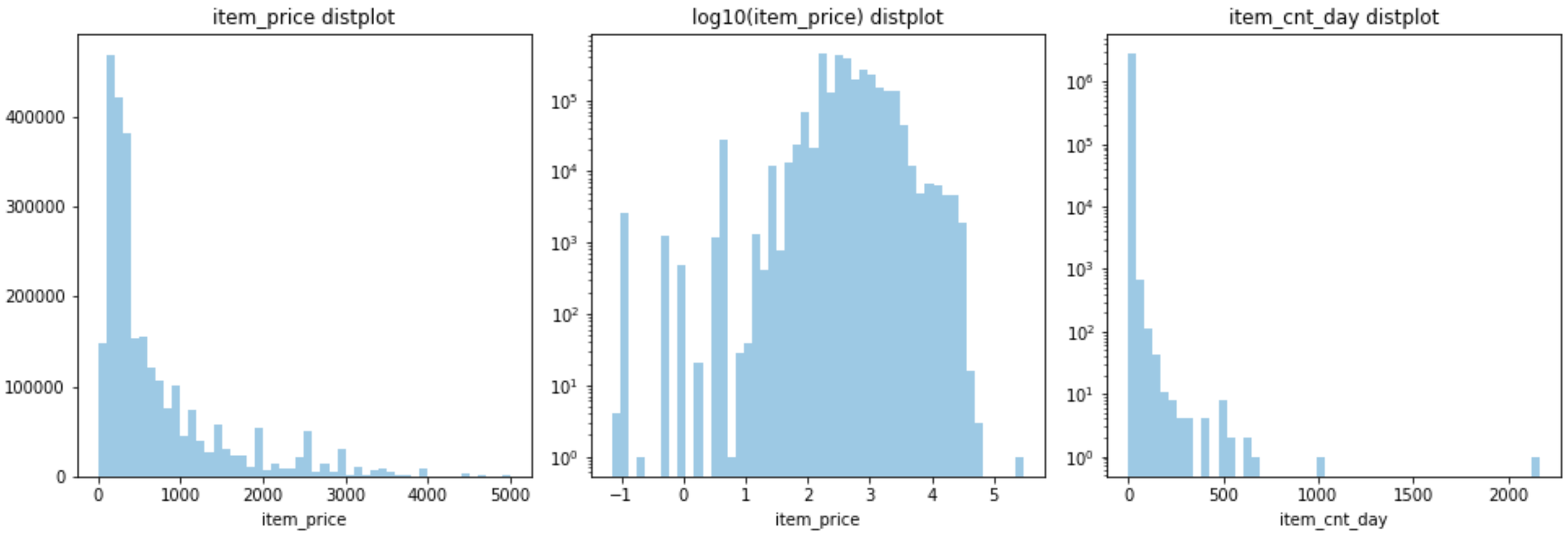
4) Merge everything

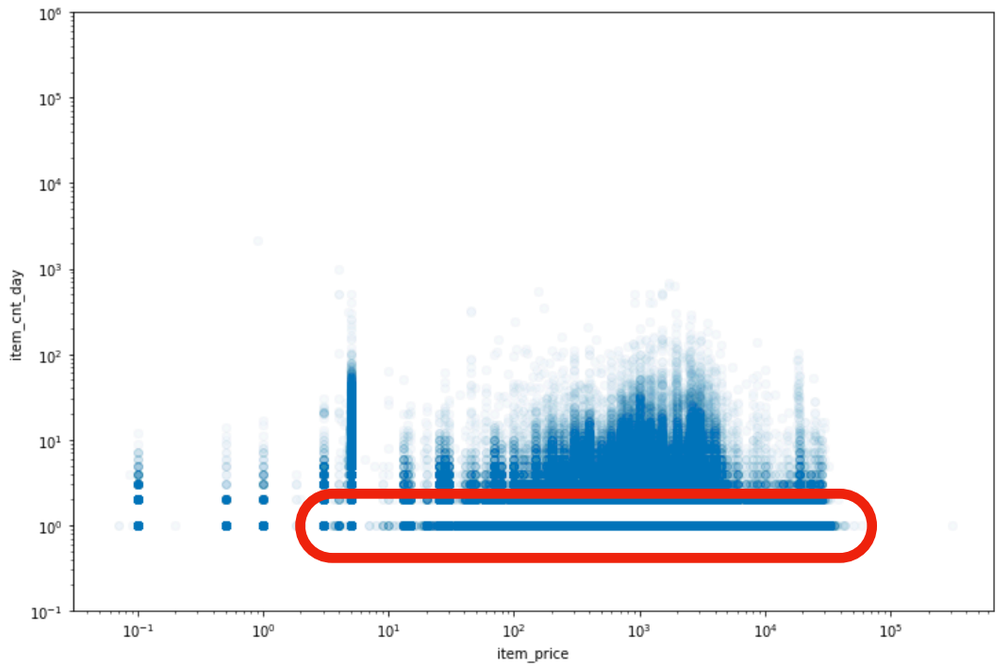
```
train_df['price_usd'] = train_df['item_price'] / train_df['rate']
train_df['revenue'] = train_df['item_price'] * train_df['item_cnt_day']
train_df['revenue_usd'] = train_df['price_usd']*train_df['item_cnt_day']
```

```
train_df.head()
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	date_test	item_name	item_category	item_subcategory	city	shop_type	rate	price_usd	revenue	revenue_usd
0	2013-01-02	0	59	22154	999.0	1.0	2013-01-02	явление 2012	Кино	Blu-Ray	ярославль	тц	30.18	33.101392	999.0	33.101392
1	2013-01-02	0	59	2746	299.0	2.0	2013-01-02	dead space 2	Игры PC	Стандартные издания	ярославль	тц	30.18	9.907223	598.0	19.814447
2	2013-01-02	0	59	2847	1699.0	1.0	2013-01-02	disney история игрушек парк развлечений	Игры	XBOX 360	ярославль	тц	30.18	56.295560	1699.0	56.295560
3	2013-01-02	0	59	2848	99.0	1.0	2013-01-02	disney мир героев	Игры PC	Стандартные издания	ярославль	тц	30.18	3.280318	99.0	3.280318
4	2013-01-02	0	59	2854	449.0	1.0	2013-01-02	disney храбрая сердцем	Игры PC	Стандартные издания	ярославль	тц	30.18	14.877402	449.0	14.877402

4) Distribution





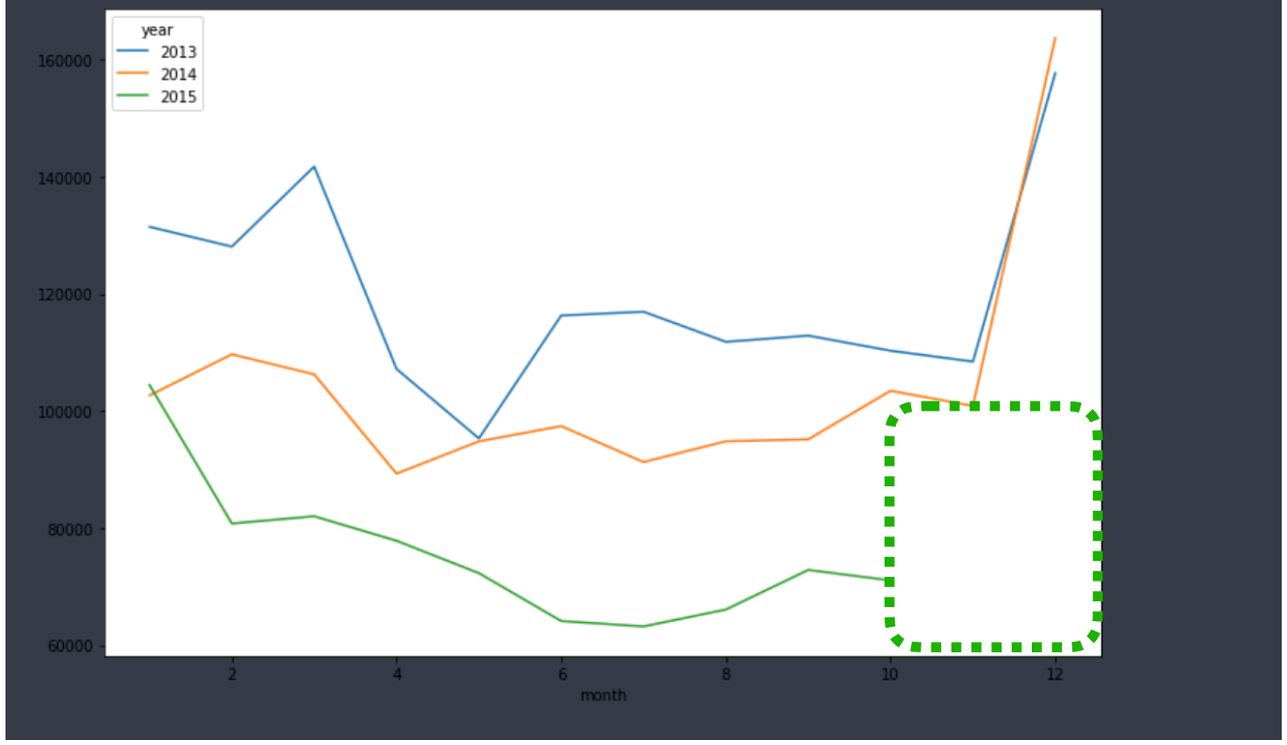
4) Total sales behavior

```
t_df.groupby(
    ['year','month']).agg({'item_cnt_day': np.sum}).reset_index().head(5
```

	year	month	item_cnt_day
0	2013	1	131479.0
1	2013	2	128090.0
2	2013	3	141773.0
3	2013	4	107190.0
4	2013	5	95315.0

```
data = t_df.groupby(
    ['year','month']).agg({'item_cnt_day': np.sum}).reset_index()\
    .pivot(index='month', columns='year', values='item_cnt_day')
data.plot(figsize=(12, 8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b61129b70>
```



[진행 상황]

- item category -> main_type, sub_type
- shop name -> city, shop_type, shop_name
- item_cnt_day -> -1이 존재. 대부분 1
- item_cnt_day groupby > 평균 감소, 편차 증가
- price -> 달러 ~ 루블 환율 칼럼

[해야 할 일]

- AR, MA 공식을 활용한 칼럼 생성
- lag_feature의 생성
- ML

4. kernel - Feature Engineering

- Target Variable / Feature Engineering

1) Target variable

```
test_df.head()
```

	ID	shop_id	item_id
0	0	5	5037
1	1	5	5320
2	2	5	5233
3	3	5	5232
4	4	5	5268

```
test_df.shape
```

```
(214200, 3)
```

```
pd.DataFrame(product(*[test_df['shop_id'].unique(),
                       test_df['item_id'].unique()])).shape
```

```
(214200, 2)
```

- ID / shop_id / item_id 로 이루어져 있다.
- product 연산으로 확인 결과 중복값은 없다.
- 대회 방식에따라 적절한 변환이 필요하다 (대회 : 총 판매량 예측)

*product : https://programmers.co.kr/learn/courses/4008/lessons/12835

1) Target variable

```
gr_date = train_df.groupby('date_block_num')
```

```
t_d = gr_date.apply(lambda x: product(*[[x.name],
                                        x['shop_id'].unique(),
                                        x['item_id'].unique()]))
```

```
all_vals = pd.DataFrame()
for t in tqdm(t_d.values):
    all_vals = pd.concat([all_vals, pd.DataFrame(t)], axis=0)
```

```
100%|██████████| 34/34 [00:06<00:00,  5.15it/s]
```

```
all_vals.columns = ['date_block_num', 'shop_id', 'item_id']
all_vals.head()
```

	date_block_num	shop_id	item_id
0	0	59	22154
1	0	59	2746
2	0	59	2847
3	0	59	2848
4	0	59	2854

- date_block_num 으로 groupby
- date_block_num -> shop_id -> item_id 순서로 순회하며 DataFrame 생성
- columns 수정

[작업 현황]

- test_df
- train_df -> gr_date -> t_d -> all_vals

1) Target variable

```
test_vals = pd.DataFrame()
test_vals['shop_id'] = test_df['shop_id'].copy()
test_vals['item_id'] = test_df['item_id'].copy()
test_vals['date_block_num'] = 34
```

```
all_vals = pd.concat([all_vals, test_vals], axis=0)
```

```
all_vals.head(3)
```

	date_block_num	item_id	shop_id
0	0	22154	59
1	0	2746	59
2	0	2847	59

- Target ‘date_block_num’ = 34
- train_df를 통해 만들었던 all_vals 에 target 추가

[작업 현황]

- test_df
- train_df -> gr_date -> t_d -> all_vals

1) Target variable

```
month_sum = train_df.groupby(['shop_id',
                              'item_id',
                              'date_block_num']).sum()
```

```
month_sum.shape
```

```
(1549415, 6)
```

```
month_sum = month_sum.rename(columns={'item_cnt_day':'item_cnt_month'})
```

```
month_sum.head()
```

shop_id	item_id	date_block_num	item_price	item_cnt_month	rate	price_usd	revenue	reve
0	30	1	2385.0	31.0	272.60	78.744539	8215.0	271.6
	31	1	3038.0	11.0	212.17	100.234346	4774.0	157.8
	32	0	884.0	6.0	120.53	29.337544	1326.0	43.94
		1	1547.0	10.0	211.06	51.309233	2210.0	73.28
	33	0	1041.0	3.0	90.60	34.470247	1041.0	34.47

- Basic form >
  shop_id, item_id, date_block_num

- Basic form으로 groupby 하여 sum()

[작업 현황]

- test_df
- train_df -> gr_date -> t_d -> all_vals
- train_df -> month_sum

1) Target variable

```
month_sum = pd.merge(all_vals, month_sum,
                     on=['shop_id','item_id', 'date_block_num'],
                     how='left').fillna(0)
```

```
month_sum.head()
```

	date_block_num	item_id	shop_id	item_cnt_month	revenue	revenue_usd
0	0	22154	59	1.0	999.0	33.101392
1	0	2746	59	9.0	2571.0	84.993986
2	0	2847	59	2.0	3398.0	112.853483
3	0	2848	59	2.0	198.0	6.575924
4	0	2854	59	3.0	1347.0	44.543739

- all_vals (basic_forms) 에 merge

[작업 현황]

- test_df -> all_vals
- train_df -> gr_date -> t_d -> all_vals
- train_df -> month_sum -> all_vals

1) Target variable

```
alles = month_sum.merge(shop_df[['shop_id', 'city', 'shop_type']],
                        how='left', on=['shop_id'])
```

```
alles = alles.merge(item_df, how='left', on=['item_id'])
```

```
alles.head()
```

	date_block_num	item_id	shop_id	item_cnt_month	revenue	revenue_usd	city	shop_typ
0	0	22154	59	1.0	999.0	33.101391	ярославль	тц
1	0	2746	59	9.0	2571.0	84.993988	ярославль	тц
2	0	2847	59	2.0	3398.0	112.853485	ярославль	тц
3	0	2848	59	2.0	198.0	6.575924	ярославль	тц
4	0	2854	59	3.0	1347.0	44.543739	ярославль	тц

- month_sum (월별 매출)에 item_df, shop_df merge >> alles 생성

[작업 현황]
- test_df -> all_vals
- train_df -> gr_date -> t_d -> all_vals -> month_sum -> alles
- train_df -> month_sum -> alley
- shop_id -> alles
- item_id -> alles

1) Target variable

```
exch_rate['month'] = exch_rate['date'].dt.month
exch_rate['year'] = exch_rate['date'].dt.year
```

```
temp_mean = exch_rate.groupby(['year', 'month']).mean().reset_index()
```

```
temp_mean.shape
```

```
(43, 3)
```

```
temp_mean['date_block_num'] = range(43)
alles = alles.merge(temp_mean, how='left', on=['date_block_num'])
```

- USE-RUB 환율 데이터 월 평균 생성
  >>alles에 merge

[작업 현황]

- test_df -> all_vals
- train_df -> gr_date -> t_d -> all_vals -> month_sum -> alles
- train_df -> month_sum -> alley
- shop_id -> alles
- item_id -> alles
- exchange -> alles

1) Target variable

```
alles.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10985992 entries, 0 to 10985991
Data columns (total 15 columns):
date_block_num      int8
item_id             int16
shop_id             int8
item_cnt_month      float16
revenue             float32
revenue_usd         float32
city                object
shop_type           object
item_name           object
item_category_id    int8
item_category       object
item_subcategory    object
year                int16
month               int8
rate                float16
dtypes: float16(2), float32(2), int16(2), int8(4), object(5)
memory usage: 712.4+ MB
```

- Metadata 생성

2) Feature Engineering

- lag_features
- projection (투영)

2) Feature Engineering

```python
def lag_feature(df, lags, col):
    tmp = df[['date_block_num','shop_id','item_id',col]]
    for i in lags:
        shifted = tmp.copy()
        shifted.columns = ['date_block_num',
                           'shop_id','item_id', col+'_lag_'+str(i)]
        shifted['date_block_num'] += i
        df = pd.merge(df, shifted, on=['date_block_num',
                                       'shop_id','item_id'], how='left')
    return df
```

[date_block_num / item_cnt_day]

2) Feature Engineering

```
gr = train_df.groupby(['item_id', 'shop_id', 'date_block_num'])
```

```
m_price = gr.mean()
```

```
m_price.head(3)
```

			item_price	item_cnt_day	rate	price_usd	revenue	revenue
item_id	shop_id	date_block_num						
0	54	20	58.0	1.0	37.3183	1.554197	58.0	1.554197
1	55	15	4490.0	1.0	35.3850	126.890495	4490.0	126.8904
		18	4490.0	1.0	34.3913	130.556274	4490.0	130.5562

```
m_price = m_price.rename(columns={'revenue': 'av_revenue',
                                  'revenue_usd': 'av_revenue_usd',
                                  'item_cnt_day': 'av_item_cnt_day',
                                  'item_price': 'av_item_price',
                                  'price_usd': 'av_price_usd'})
```

```
m_price['item_purchase_days'] = gr.count()['date']
```

```
m_price['av_item_cnt_day'] /= m_price['item_purchase_days']
```

- Base_form을 기준으로 월별 평균을 구함
- 월별 평균 판매 일수를 구함

[작업 현황]

- train_df -> gr
- gr -> m_price

2) Feature Engineering

```
m_group = m_price.groupby(['item_id'],
                          as_index=False)['av_item_price'].mean()
```

```
m_group = m_group.rename(columns={'av_item_price': 'mean_price'})
```

```
m_price = pd.merge(all_vals, m_price,
                   on=['shop_id','item_id', 'date_block_num'],
                   how='left').fillna(0)
```

```
m_price = pd.merge(m_price, m_group, on=['item_id'], how='left')
```

```
m_price['price_dev'] = (m_price['av_item_price'] - m_price['mean_price'])
                       / m_price['mean_price']
```

- m_price
  현재 월별로 평균이 구해져있는 상태

- m_group
  월별 평균을 item_id로 그룹 평균

- price_dev : 월별 전체 평균에서 item_id 그룹별 평균의 차이

[작업 현황]

- train_df -> gr
- gr -> m_price -> m_group

2) Feature Engineering

Lag features

```
m_price = lag_feature(m_price, [1, 2, 3, 6, 12], 'item_purchase_days')
m_price = lag_feature(m_price, [1, 2, 3, 6, 12], 'av_item_cnt_day')
m_price = lag_feature(m_price, [1, 2, 3], 'av_item_price')
m_price = lag_feature(m_price, [1, 2, 3], 'av_price_usd')
m_price = lag_feature(m_price, [1, 2, 3], 'price_dev')
m_price = lag_feature(m_price, [1, 2, 3], 'av_revenue')
m_price = lag_feature(m_price, [1, 2, 3], 'av_revenue_usd')
```

```
m_price.head()
```

ısd	item_purchase_days	mean_price	...	av_price_usd_lag_3	price_dev_lag_1	price_dev_lag_2	price_d
	1.0	690.5000	...	NaN	NaN	NaN	NaN
	7.0	265.7500	...	NaN	NaN	NaN	NaN
	2.0	1414.0000	...	NaN	NaN	NaN	NaN
	2.0	96.4375	...	NaN	NaN	NaN	NaN
	2.0	237.2500	...	NaN	NaN	NaN	NaN

[작업 현황]

- train_df -> gr
- gr -> m_price -> m_group

2) Feature Engineering

Projection

```
m_price['proj_av_revenue'] = 2 * (m_price['av_revenue_lag_1'] -
                                  m_price['av_revenue_lag_2'])\
                                  + m_price['av_revenue_lag_2']

m_price['proj_av_revenue_2'] = 3 * (m_price['av_revenue_lag_1'] -
                                    m_price['av_revenue_lag_3']) / 2\
                                    + m_price['av_revenue_lag_3']
```

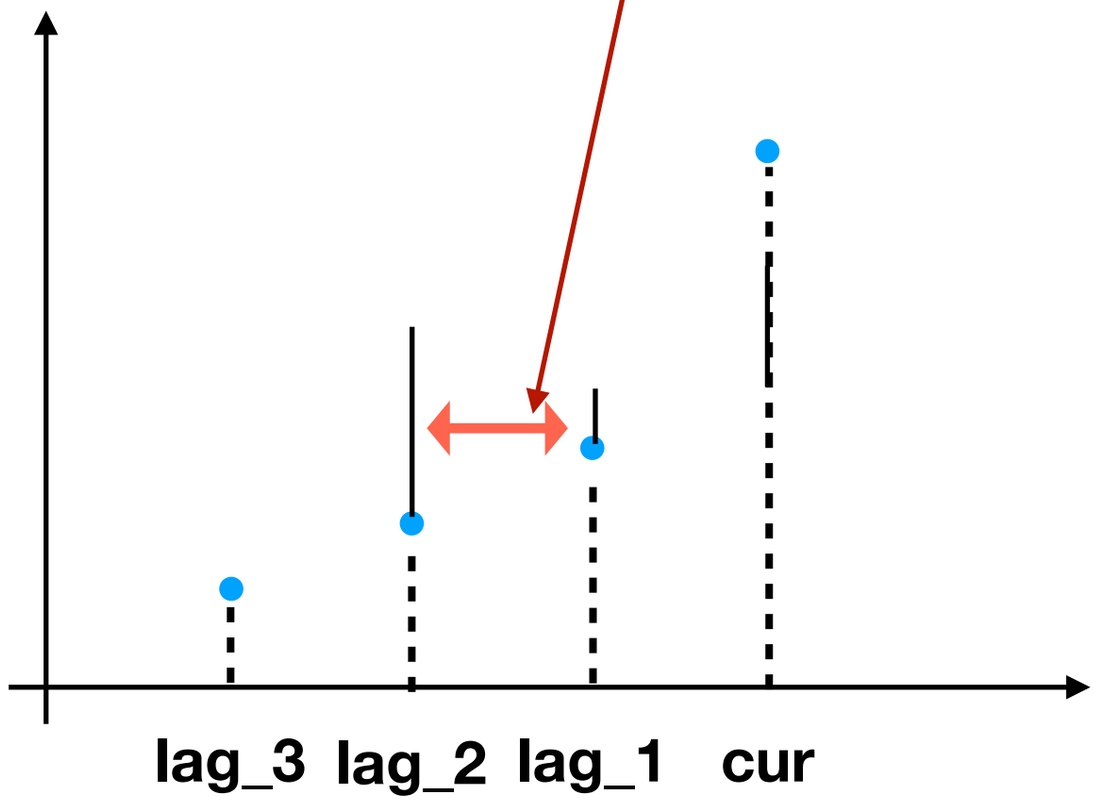
```
m_price['proj_price'] = 2 * (m_price['av_item_price_lag_1'] -
                             m_price['av_item_price_lag_2'])\
                           + m_price['av_item_price_lag_2']

m_price['proj_price_2'] = 3 * (m_price['av_item_price_lag_1'] -
                               m_price['av_item_price_lag_3'])\
                             / 2 + m_price['av_item_price_lag_3']
```

[작업 현황]

- train_df -> gr
- gr -> m_price -> m_group

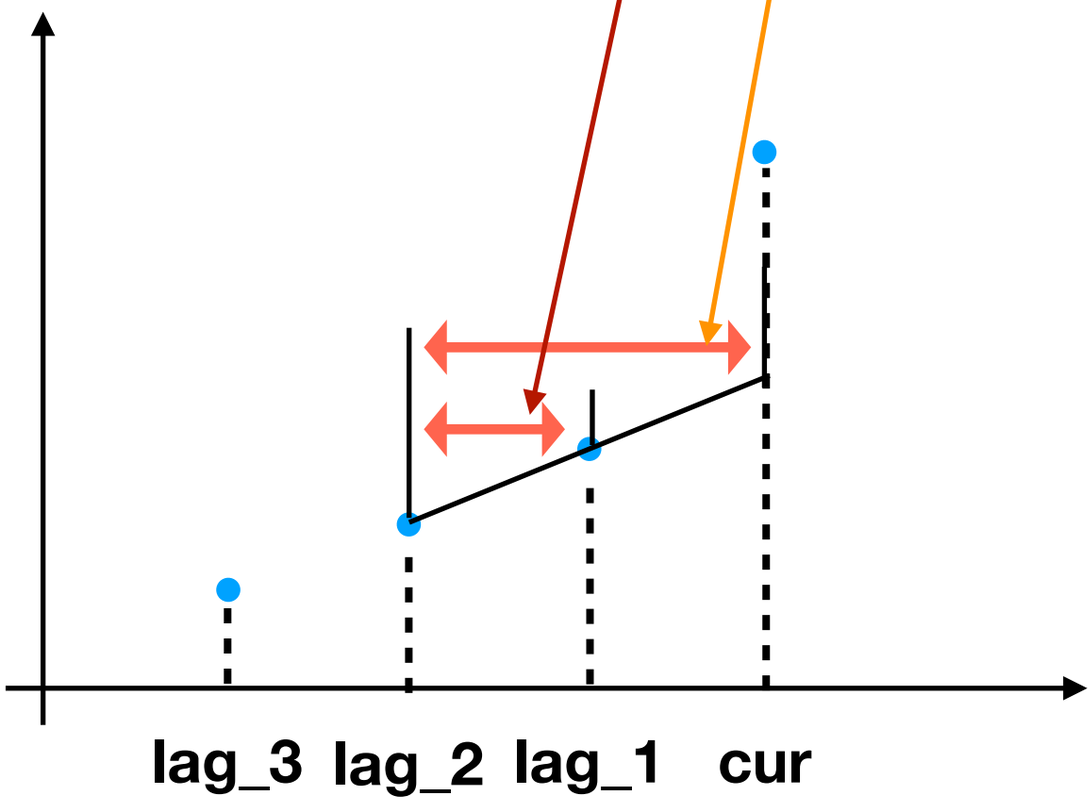
2) Feature Engineering

```
m_price['proj_av_revenue'] = 2 * (m_price['av_revenue_lag_1'] -
                                  m_price['av_revenue_lag_2'])\
                                  + m_price['av_revenue_lag_2']
```



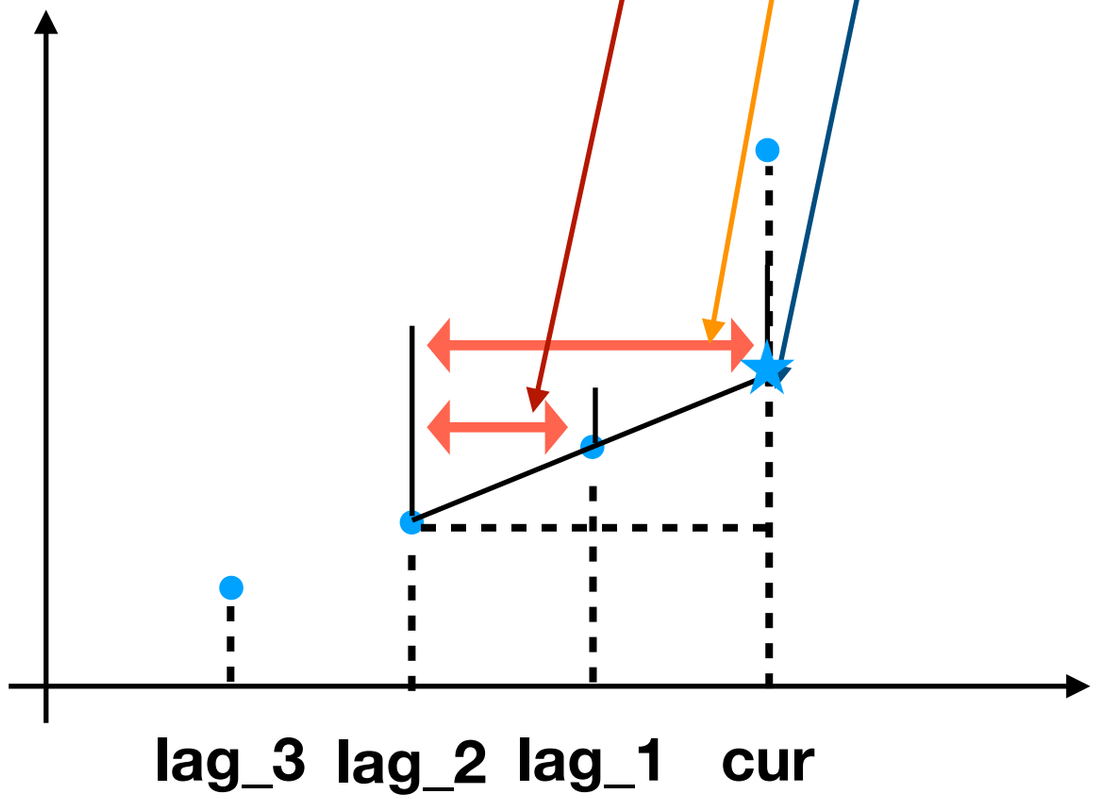
2) Feature Engineering

```
m_price['proj_av_revenue'] = 2 * (m_price['av_revenue_lag_1'] -
                                  m_price['av_revenue_lag_2'])\
                                  + m_price['av_revenue_lag_2']
```



2) Feature Engineering

```
m_price['proj_av_revenue'] = 2 * (m_price['av_revenue_lag_1'] -
                                  m_price['av_revenue_lag_2'])\
                             + m_price['av_revenue_lag_2']
```



2) Feature Engineering

Merge

```
m_price = reduce_mem_usage(m_price.loc[:, cols])

alles = pd.merge(alles, m_price,
         on=['item_id', 'date_block_num', 'shop_id'], how='left')
```

[작업 현황]

- train_df -> gr
- gr -> m_price -> m_group -> m_price -> alles

5. Model

1) Input data

```
alles = pd.read_pickle('data/alles_final.pkl')
#alles = alles.drop(['mean_price', 'av_shop_item'], axis=1)
print('[*]alles shape : {}'.format(alles.shape))
```

```
[*]alles shape : (10985992, 83)
```

- EDA 후 생성한 DataFrame 로드

item_purchase_days_lag_()

item_id, shop_id, date_block_num 을 그룹하여 purchase_date count

av_item_cnt_day_lag_()

하루 평균 판매량

av_price_usd_lag_()

판매 평균 금액

av_revenue_lag_()

수익 평균 금액

av_revenue_usd_lag_()

수익 평균 금액(USD)

proj_av_revenue_()

수익 평균 금액 -> projection

item_cnt_month_lag_()

한달간 판매량

revenue_lag_()

수익

revenue_usd_lag_()

수익(USD)

[*] lag : date_block_num, shop_id, item_id groupby -> shift -> merge

2) Encoding

```
alles[['item_name', 'item_category',
       'item_subcategory', 'city',
       'shop_type']] = \
alles[['item_name', 'item_category',
       'item_subcategory', 'city',
       'shop_type']].apply(LabelEncoder().fit_transform)
```

- Categorical -> LabelEncoder

3) select row

```
print('[*](date block num ==0).isnull().sum() : ',
      alles.iloc[0, :].isnull().sum())
```

```
[*](data block num ==0).isnull().sum() :  62
```

```
print('[*](data block num ==11).isnull().sum() : ',
      alles[alles.date_block_num==11].iloc[0, :].isnull().sum())
```

```
[*](data block num ==0).isnull().sum() :  9
```

- lag feature : 1 ~ 12 의 값
- date_block_num == 0의 경우 : Null - 62/83
- date_block_num == 11의 경우 : Null - 9/83

```
rows = alles['date_block_num'] > 11
data = alles.loc[rows, :]
```

```
print('[*](data block num ==0).isnull().sum() : ',
      data.iloc[0, :].isnull().sum())
```

```
[*](data block num ==0).isnull().sum() :  0
```

- **date_block_num <= 11 의 row는 버린다**

4) train

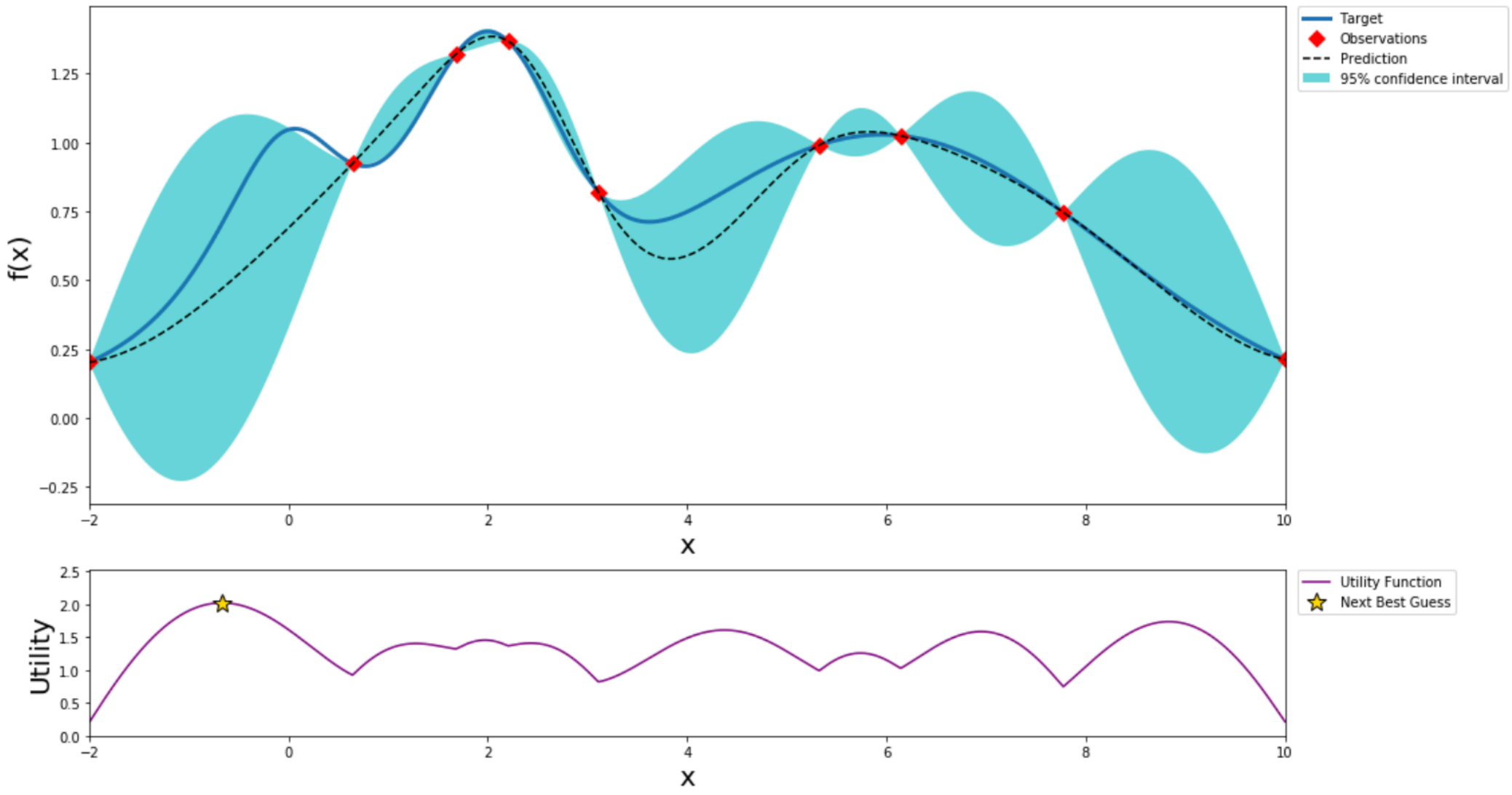
```
for train_index, test_index in list(kf.split(X_train, y_train)):
    X_train2 = X_train.iloc[train_index, :]
    y_train2 = y_train.iloc[train_index]
    X_test2 = X_train.iloc[test_index, :]
    y_test2 = y_train.iloc[test_index]

    d_training = lgb.Dataset(X_train2, label=y_train2,
                             categorical_feature=categorical_features,
                             free_raw_data=False)
    d_test = lgb.Dataset(X_test2, label=y_test2,
                         categorical_feature=categorical_features,
                         free_raw_data=False)

    clf = lgb.train(params, train_set=d_training, num_boost_round=1000,
                    valid_sets=[d_training, d_test], verbose_eval=25,
                    early_stopping_rounds=50
                    )

    models.append(clf)
    temp_predict = clf.predict(X_test2, num_iteration=clf.best_iteration)
    y_final_test += clf.predict(X_final_test, num_iteration=clf.best_iteration)/5
```

- Kfold(5) -> iterator 5
- train/test -> lgb.Dataset -> train -> predict
- lgb.Dataset
  categorical_features 자동 처리
- fold 별로 5개의 모델이 정답을 각각 예측한 뒤 이를 평균내서 제출한다

6. Tuning

- https://www.kaggle.com/corochann/optuna-tutorial-for-hyperparameter-optimization

1) BayesianOptimization



$$f(x) = e^{-(x-2)^2} + e^{-\frac{(x-6)^2}{10}} + \frac{1}{x^2+1},$$

2) example

```python
import optuna
def func(trail):
    layers = []
    n_layers = trail.suggest_int('n_layers', 1, 4)
    for i in range(n_layers):
        layers.append(trail.suggest_int(str(i), 1, 128)
    mnist = fetch_mldata('MNIST original')
    x_train, x_test, y_train, y_test = \
            train_test_split(mnist.data, mnist.target)
    clf = MLPClassifier(hidden_layer_sizes=tuple(layers))
    clf.fit(x_train, y_train)
    return clf.score(x_test, y_test)

study = optuna.create_study()
study.optimize(func, n_trials=100)
```

- **trail.suggest_int -> 지정된 범위 자동 탐색**

- **optima object 생성**

- **optimize**

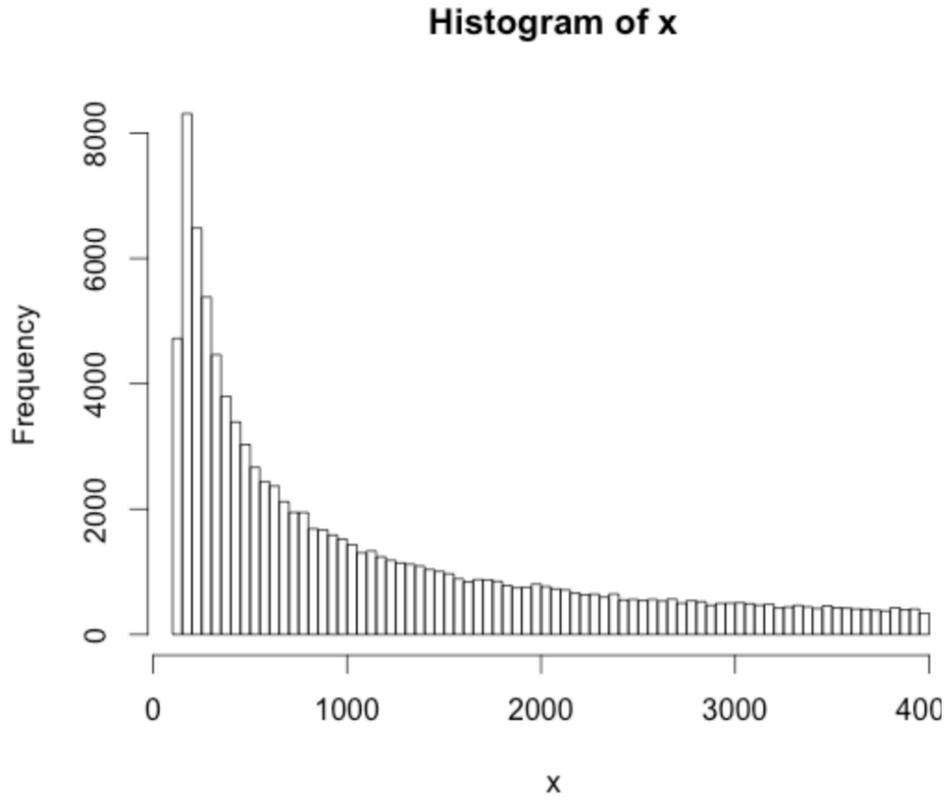
3) suggest_(type)

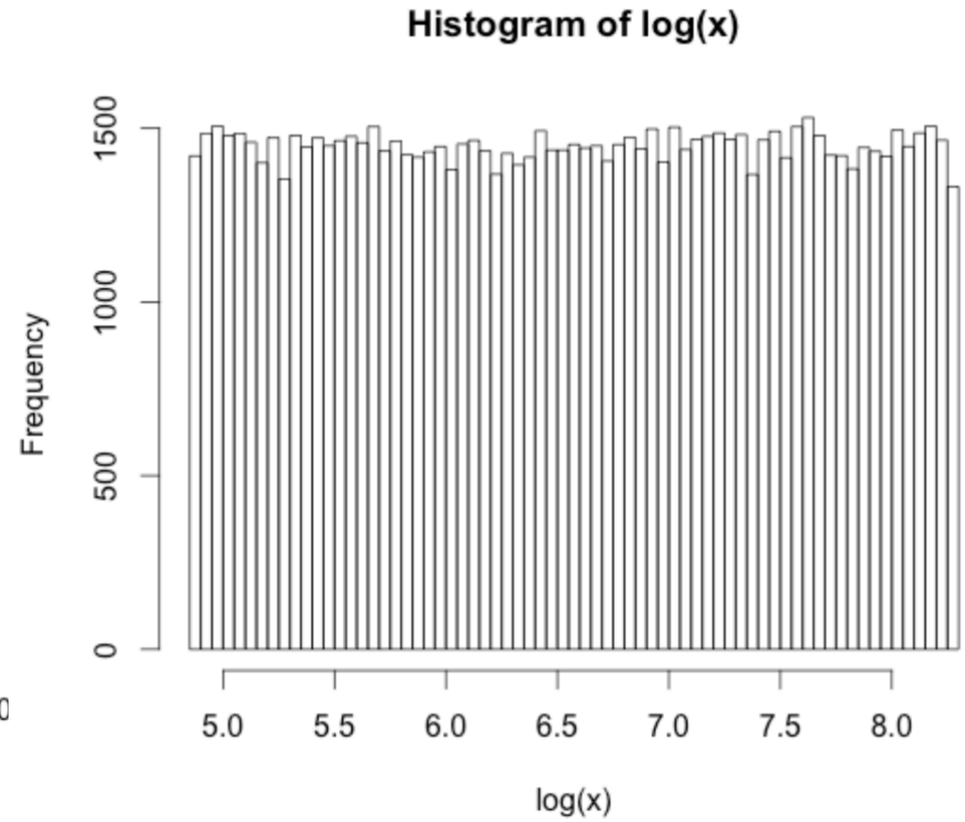
```
Categorical parameter ¶
optimizer = trial.suggest_categorical('optimizer', ['MomentumSGD', 'Adam'])
Int parameter
num_layers = trial.suggest_int('num_layers', 1, 3)
Uniform parameter
dropout_rate = trial.suggest_uniform('dropout_rate', 0.0, 1.0)
Loguniform parameter
learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e-2)
Discrete-uniform parameter
drop_path_rate = trial.suggest_discrete_uniform('drop_path_rate', 0.0, 1.0, 0.1)
```

- **Categorical**
- **Int**
- **Uniform**
- **Log-uniform**
- **Discrete-uniform**

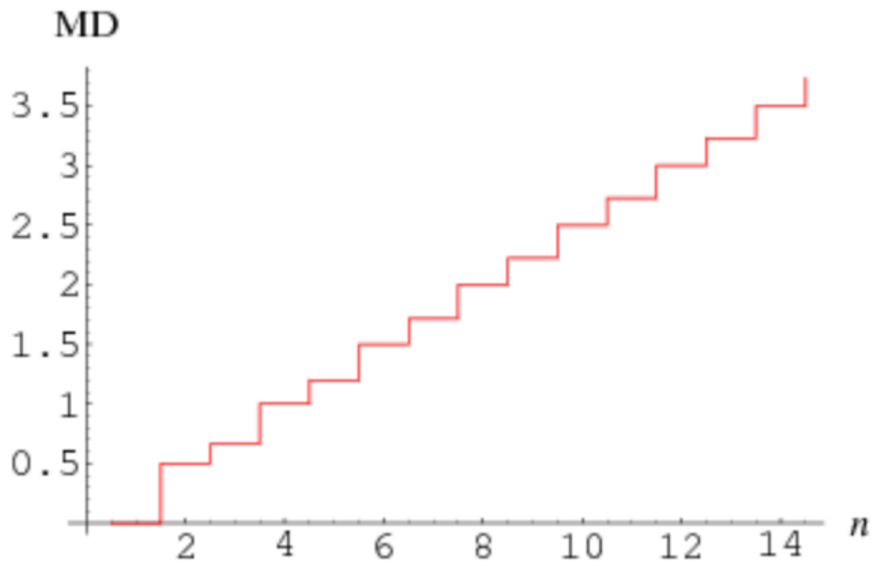
3) suggest_type

Log - uniform





Descrete - uniform



4) GridSearch style

```
params = {
    'num_leaves': [4, 8, 16, 32, 64],
    'objective': 'regression',
    'learning_rate': [0.1, 0.01, 0.001],
    "boosting": ["gbdt", "rf"],
    'lambda_l1': [0.01, 0.05, 0.1],
    'lambda_l2': [0.01, 0.05, 0.1],
    "bagging_freq": 5,
    "bagging_fraction": [0.1, 0.3, 0.6, 0.9],
    "feature_fraction": [0.1, 0.3, 0.6, 0.9],
    "metric": metric,
    "verbosity": -1,
}
```

- 범위를 좁혀가는 것이 상당히 어렵다.
- CV를 이용할 경우 Runtime이 배로 증가한다

```
def get_gs_model(model, params, cv, verbose=10):
    model = GridSearchCV(model, params, cv=kfolds,
```

5) optuna style

```
def fit_model(....):
    params = {
            'num_leaves': trial.suggest_int('num_leaves', 2, 256),
            'objective': 'regression',
            'learning_rate': 0.1,
            "boosting": "gbdt",
            'lambda_l1': trial.suggest_loguniform('lambda_l1', 1e-8, 10.0),
            'lambda_l2': trial.suggest_loguniform('lambda_l2', 1e-8, 10.0),
            "bagging_freq": 5,
            "bagging_fraction": trial.suggest_uniform('bagging_fraction', 0.1, 1.0),
            "feature_fraction": trial.suggest_uniform('feature_fraction', 0.4, 1.0),
            "metric": metric,
            "verbosity": -1,
        }
    ...
    model = lgb.train(params,
                      train_set=d_train,
                      num_boost_round=num_rounds,
                      valid_sets=watchlist,
                      verbose_eval=verbose_eval,
                      early_stopping_rounds=early_stop)

    ...
    return model
```

```
study = optuna.create_study()
study.optimize(objective, n_trials=10)
#study.optimize(lambda trial: objective(trial, arg0=1, arg1=2),
#n_trials=100)
```

- fit / train 코드를 실행하는 함수 작성 후, optimize 파라미터로 함수 전달

- 함수에서 parameter가 필요할 경우 전달 가능

5) optuna result

```
print('Best trial: score {}, params {}'.format(study.best_trial.value, study.best_trial.params))

# It has alias, below is also same.
# print('Best trial: score {}, params {}'.format(study.best_value, study.best_params))
```

->study = optuna.create_study()

```
Best trial: score 0.391769546475642, params {'num_leaves': 17, 'lambda_l1': 0.03280599745495849, 'lambda_l2': 0.1609433676848143, 'bagging_fraction': 0.907447663961603, 'feature_fraction': 0.9526656120616738}
```

->optuna.optimize(study…)

->study.best_params

```
valid_score, models0, y_pred_valid, y_train = objective(optuna.trial.FixedTrial(study.best_params), fast_check=False, target_meter=0, return_info=True)
```

->optuna.trial.FixedTrial(study.best_params)

5) optuna result

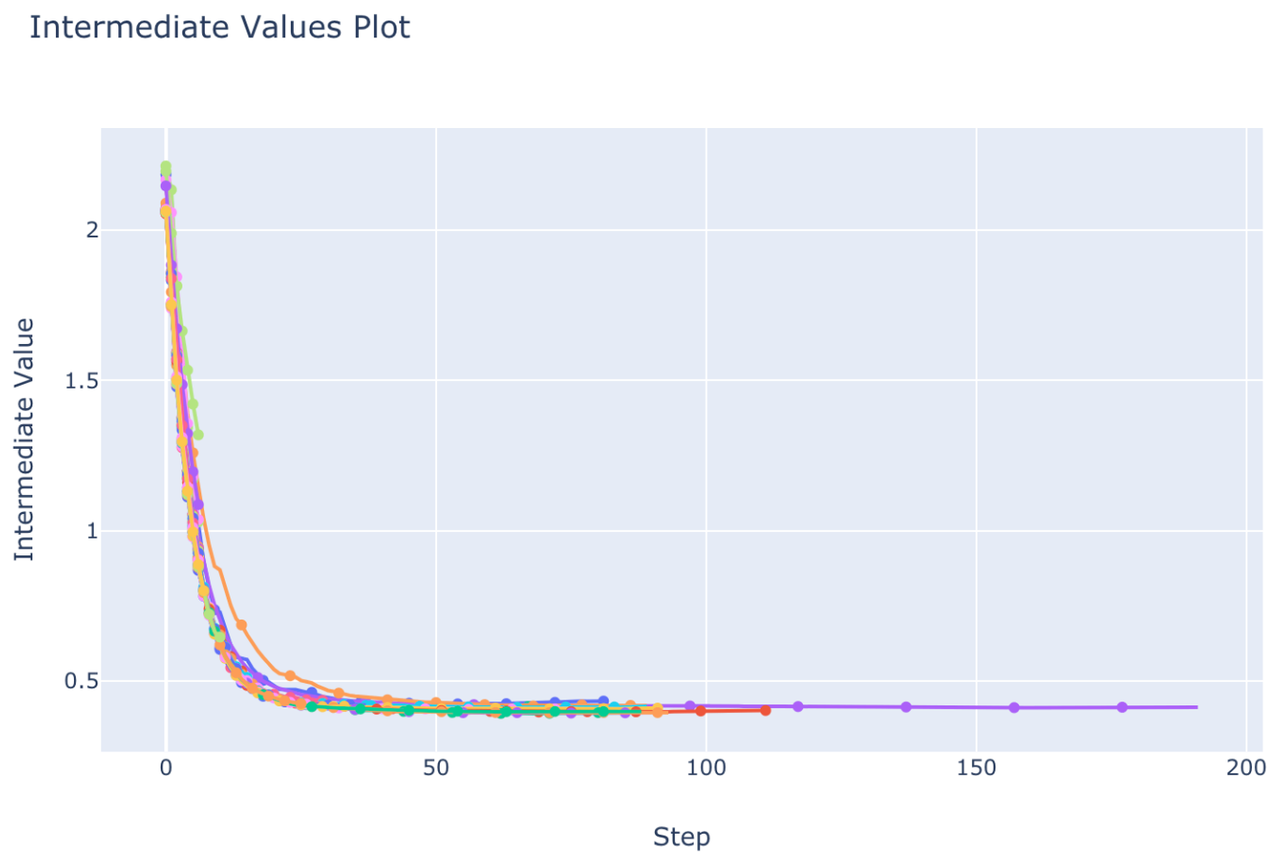
```
trials_df = study.trials_dataframe()
trials_df
```

:

	number	state	value	datetime_start	datetime_complete	params		
						bagging_fraction	feature_fraction	lam
0	0	TrialState.COMPLETE	0.422976	2019-11-09 09:48:58.126685	2019-11-09 09:48:59.387033	0.163128	0.535436	3.7
1	1	TrialState.COMPLETE	0.397111	2019-11-09 09:48:59.391329	2019-11-09 09:49:00.576226	0.672162	0.971740	1.3
2	2	TrialState.COMPLETE	0.405595	2019-11-09 09:49:00.580185	2019-11-09 09:49:02.325606	0.342223	0.740275	4.8
3	3	TrialState.COMPLETE	0.410640	2019-11-09 09:49:02.329879	2019-11-09 09:49:04.036126	0.962719	0.924633	6.8
4	4	TrialState.COMPLETE	0.416890	2019-11-09 09:49:04.040552	2019-11-09 09:49:05.200182	0.567689	0.727386	2.1

5) optuna result

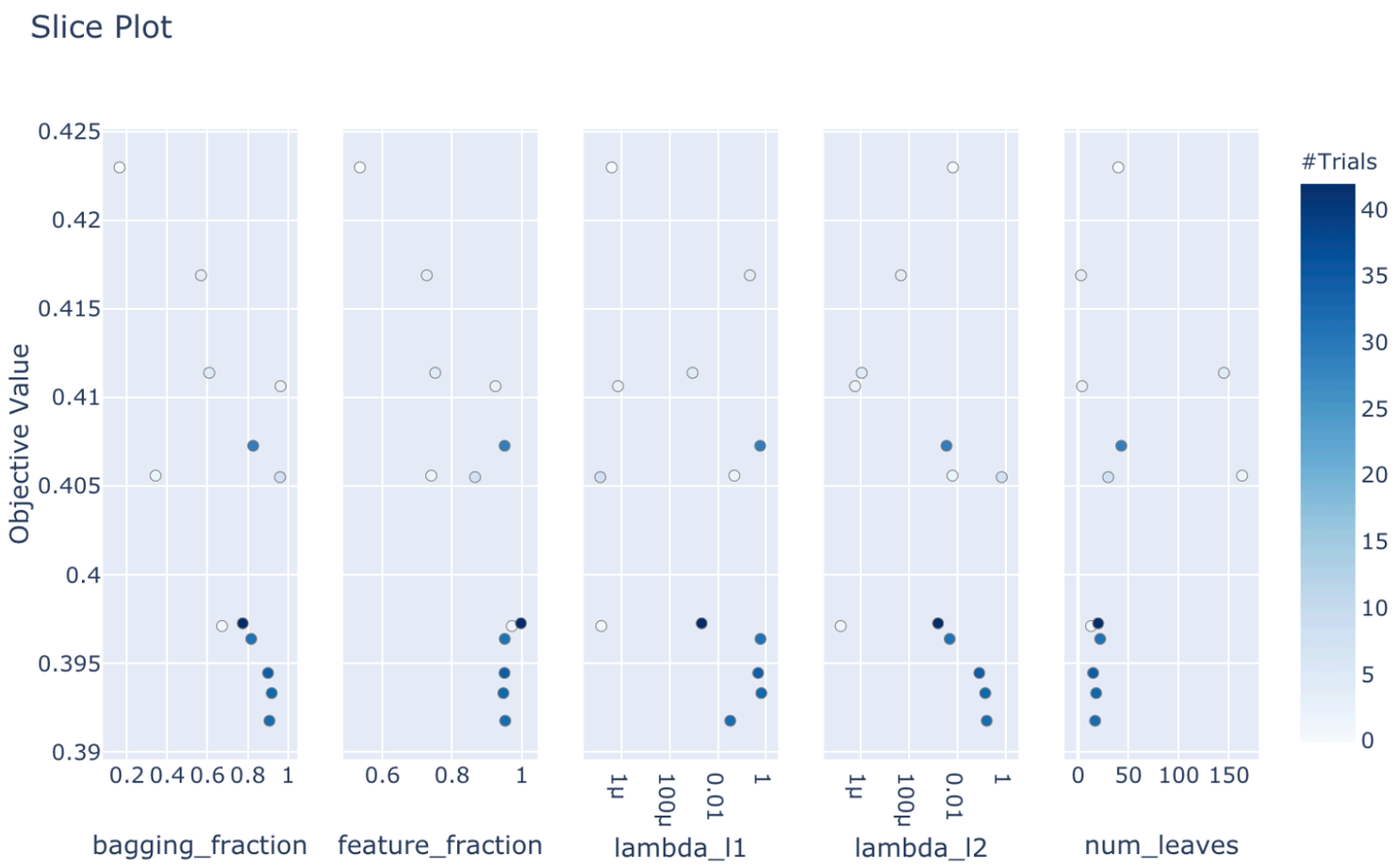
```
optuna.visualization.plot_intermediate_values(study)
```



- **Trial, step 별로 모델 score 확인 가능**

5) optuna result

```
optuna.visualization.plot_slice(study)
```



- **parameter 값 별로 score 확인 가능**

-> 분포 위주로 확인

7. Permutation importance

1) Permutation importance

```
import eli5
from eli5.sklearn import PermutationImportance
from lightgbm import LGBMClassifier
kf = KFold(n_splits = 2, random_state=42)
for train_index, test_index in list(kf.split(X_train, y_train)):
    X_train2 = X_train.iloc[train_index, :]
    y_train2 = y_train.iloc[train_index]
    X_test2 = X_train.iloc[test_index, :]
    y_test2 = y_train.iloc[test_index]
    clf = LGBMClassifier(random_state=2020, n_threads=-1,  n_estimators=2, num_boost_round=5)
    print('>>clf fit start')
    clf.fit(X_train2, y_train2,
            verbose=1,
            eval_set=[(X_test2, y_test2)])
    print('>>clf fit finish')
    permutation_importance = PermutationImportance(clf, random_state=2020)
    print('>>perm fit start')
    permutation_importance.fit(X_train2, y_train2)
    print('>>perm fit finish')
```

- **.fit 이 실행된 clf에 대하여 수행가능.**
- **lightgbm.Dataset 클래스에 대해서는 수행하지 않음.**