1. Why do you want work in Japan?
日本で働きたい理由（日本語)

私は自分のキャリア価値を最大化したいと思います。機械学習の知識を活かし、日本の商業活動や文化をもっと深く理解するため、日本で就職したいです。私は機械学習モデルどうやって商業の運用に役立つことがすごく興味が持っています。なぜなら、商業のデータは人の活動で構築することです。私は人とのコミュニケーションが好き、日本の文化が惹きつけられ、日本で機械学習の仕事をしたいです。そのために、大学院の頃英語と日本語を勉強し、機械学習の数学も把握して、Python プログラムで基本的な ML や DL のプロジェクトを構築でした。企業の本番で使える機械学習プロジェクトは困難なことがわかりますが、私は難しいことを恐れずチャレンジして、なるべく早めに役立てる人材になりたいです。

2. What is your major or what are you studying?
大学でどのような勉強をしたか？ あなたの final year project か 研究テーマを簡単に話せるようにしてください。
a. Motive of your research paper. What do you want to achieve by writing the paper? Is it novel piece of research or a something that is already being done by other researchers? What kind of insights do you aim to find?
目的: 何を達成するのか 、何を明らかにするのか

私の研究テーマは械機学習の運用することです。中で一番気に入っているのは kaggle の上で参加する最大のロシアソフトウェア企業の販売数量を予測するコンペティションです。

コンペティションの目標は 2013 年一月から 2015 年十月までのソフトウェア

販売数量を収集し、2015 年十一月のソフトウェア販売数量を予測するという

時系列のデータ問題です。

b. Process. In order to achieve your motive, how did you go about conducting the research? Surveys, interviews, data analysis?
実施内容 ： 目的を達成するために行った実験、実施したこと

このコンペティションの未加工のデータは、特徴が店 ID と商品 ID と価格と商品種類 ID と商品を売れる時間と販売数量、六つしかありません。ML モデルには、これだけの特徴が少ないから予測することが難しいです。

販売数量を予測する問題は ML モデルの精度を高めるため、未加工のデータと専門分野に関する知識を組み合わせて、適切に動かす特徴（フィーチャー）を構築することになります。

より良い特徴を構築するために、この特徴がこの問題になぜ関連していると思われるか合理的な仮定が必要です。

まずは、データを前処理のあと、同じ月同じ商品 ID 違い店 ID の販売数量を加えて、特徴としてデータセットに合わせました。そして、同じ月違い商品 ID 同じ店 ID の販売数量を加えて、また、同じ月違い商品種類 ID の販売数量を加えて、特徴としてデータセットに合わせました。

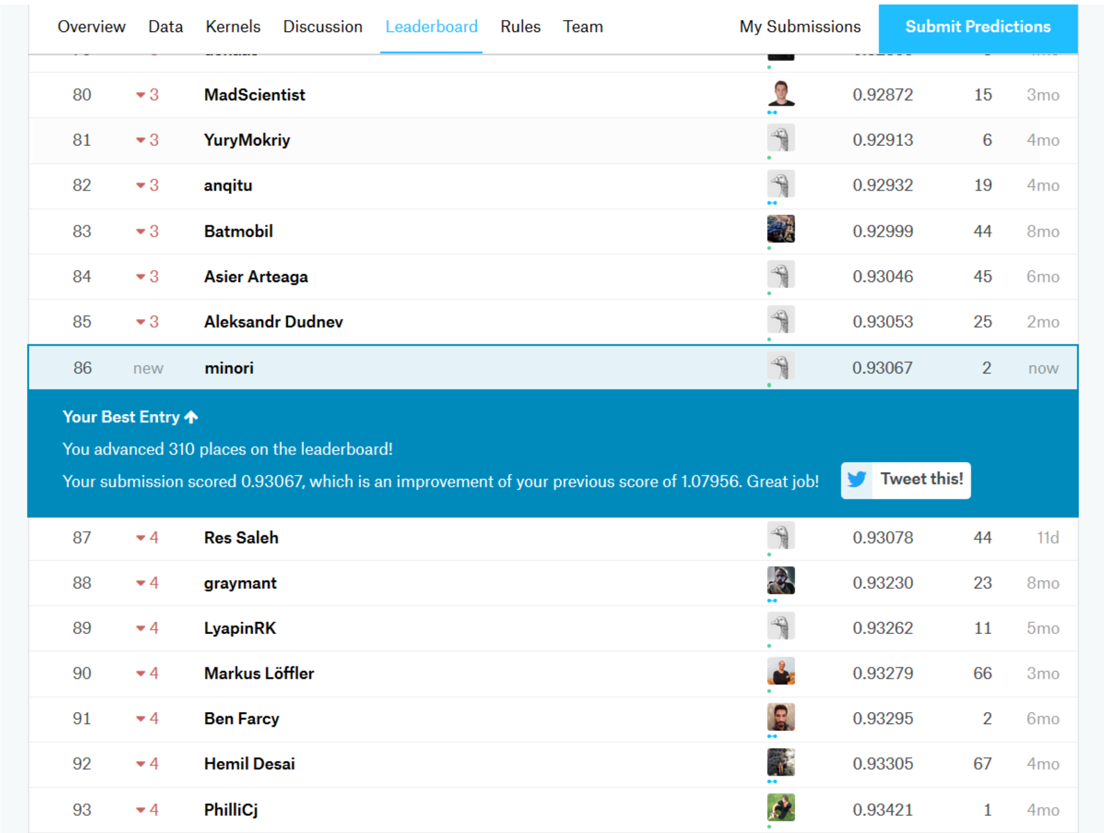
これは商品と店と商品種類が販売数量に影響を与えると論理的に推測することです。

後は色々な豊かな特徴をつくりつつ、ランダムフォレストの機械学習のアル

ゴリズムで回帰分析の精度を観察しつつ、調整していました。

c. Results. What are the results? If possible, give the actual outcome in numbers or figures.
結果 :実験や実施したことの結果および目的に対して達成できたのか

最後に、コンペティションのランキングが 86 位をとったことです。


Overview Data Kernels Discussion Leaderboard Rules Team My Submissions Submit Predictions

80	▾3	MadScientist	0.92872	15	3mo
81	▾3	YuryMokriy	0.92913	6	4mo
82	▾3	anqitu	0.92932	19	4mo
83	▾3	Batmobil	0.92999	44	8mo
84	▾3	Asier Arteaga	0.93046	45	6mo
85	▾3	Aleksandr Dudnev	0.93053	25	2mo
86	new	minori	0.93067	2	now

Your Best Entry ↑
You advanced 310 places on the leaderboard!
Your submission scored 0.93067, which is an improvement of your previous score of 1.07956. Great job!
Tweet this!

87	▾4	Res Saleh	0.93078	44	11d
88	▾4	graymant	0.93230	23	8mo
89	▾4	LyapinRK	0.93262	11	5mo
90	▾4	Markus Löffler	0.93279	66	3mo
91	▾4	Ben Farcy	0.93295	2	6mo
92	▾4	Hemil Desai	0.93305	67	4mo
93	▾4	PhilliCj	0.93421	1	4mo

d. Reflections. What have you learned? If your research went well, why? If it did not, why? How can you improve?

このコンペティションには特徴の作成することが非常に手のかかる作業です。もう一つ重要なのはトレーニング用と評価用のデータセットを分割することです。

時系列のデータだから、単純なランダムのサンプリングは使えません。それ

で、2013 年一月から 2015 年九月までのデータがトレーニング用のデータにして、2015 年十月のデータは評価用のデータにすることになりました。

時系列のデータにはノイズ変数が大変なので、非常に困難な問題と思います。このモデルの精度はまだ上げられると思いました。いま時系列の問題は強化学習のアルゴリズムで解決したいとの研究も進んでいます。私もすごく興味が持って、いつも新しい論文とプロジェクトを勉強します。

3. What kind of company do you want to work in? Why?
どのような会社で働きたいか？なぜそのように思ったか？

技術の人材を重視する会社で働きたいです。どのような分野の技術でも、ひとつずつ積み上げていく以外に、上達する術がないことがわかります。私は少しストイックな人で、地味にアルゴリズムの反復練習を黙々と続けているタイプです。だから、焦ったり、AI に過度な期待をしたりもしなく、人材育成と技術を戦略的に推進する会社で働きたいです。

1. Competition Overview

In this competition we will work with a challenging **time-series** dataset consisting of daily sales data, kindly provided by one of the largest Russian software firms - 1C Company.

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.

1.1 Loading Libraries

In [1]:

```
import pandas as pd
import numpy as np
import gc
import matplotlib.pyplot as plt
%matplotlib inline

pd.set_option('display.max_rows', 600)
pd.set_option('display.max_columns', 50)

import lightgbm as lgb
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import seaborn as sns # for making plots with seaborn
from tqdm import tqdm_notebook
# color = sns.color_palette()
# sns.set()
sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
plt.style.use('ggplot')

from itertools import product

Validation = False
```

1.2 Load data

Submissions are evaluated by **root mean squared error (RMSE)**. True target values are clipped into [0,20] range.

In [2]:

```
train = pd.read_csv('sales_train.csv.gz')
shops = pd.read_csv('shops.csv')
items = pd.read_csv('items.csv')
item_cats = pd.read_csv('item_categories.csv')
test = pd.read_csv('test.csv.gz')
submission = pd.read_csv('sample_submission.csv.gz')
```

Let's check data shapes.

In [3]:

```
print('Size of train :', train.shape)
print('Size of test :', test.shape)
print('Size of shops :', shops.shape)
print('Size of items :', items.shape)
print('Size of item_cats :', item_cats.shape)
```

```
Size of train : (2935849, 6)
Size of test : (214200, 3)
Size of shops : (60, 2)
Size of items : (22170, 3)
Size of item_cats : (84, 2)
```

2. Exploratory Data Analysis

- **item_cnt_day** - number of products sold. You are predicting a monthly amount of this measure
- **date_block_num** - a consecutive month number, used for convenience. January 2013 is 0, February 2013 is 1,..., October 2015 is 33

In [4]:

```
print('Train Unique shops :', len(train['shop_id'].unique()))
print('Test Unique shops :', len(test['shop_id'].unique()))
print('Train Unique items:', len(train['item_id'].unique()))
print('Test Unique items :', len(test['item_id'].unique()))
print('min item prize :', min(train['item_price']))
print('max item prize :', max(train['item_price']))
```

```
Train Unique shops : 60
Test Unique shops : 42
Train Unique items: 21807
Test Unique items : 5100
min item prize : -1.0
max item prize : 307980.0
```

In [5]:

```
def eda(data):
    print("----------Top-5- Record----------")
    print(data.head(5))
    print("-----------Information-----------")
    print(data.info())
    print("-----------Data Types-----------")
    print(data.dtypes)
    print("----------Missing value-----------")
    print(data.isnull().sum())
    print("----------Null value-----------")
    print(data.isna().sum())
    print("----------Shape of Data----------")
    print(data.shape)

def graph_insight(data, bins=34, figsize=(16, 16), xlabelsize=8, ylabelsize=8):
    print(set(data.dtypes.tolist()))
    df_num = data.select_dtypes(include = ['float64', 'int64'])
    df_num.hist(figsize=figsize, bins=bins, xlabelsize=xlabelsize, ylabelsize=ylabelsize);

def drop_duplicate(data, subset):
    print('Before drop shape:', data.shape)
    before = data.shape[0]
    data.drop_duplicates(subset,keep='first', inplace=True) #subset is list where you have to put
all column for duplicate check
    data.reset_index(drop=True, inplace=True)
    print('After drop shape:', data.shape)
    after = data.shape[0]
    print('Total Duplicate:', before-after)
```

In [6]:

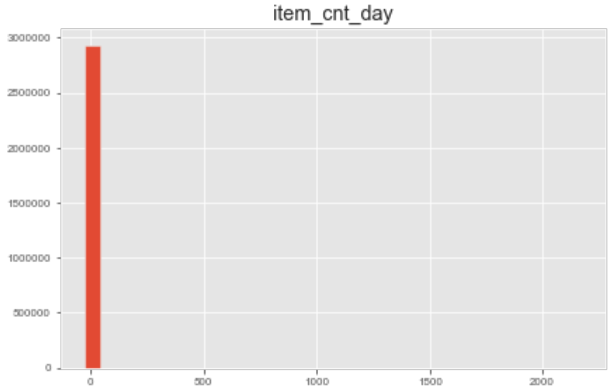
```
# sales train insights
eda(train)
graph_insight(train)
```

```
----------Top-5- Record----------
         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
```

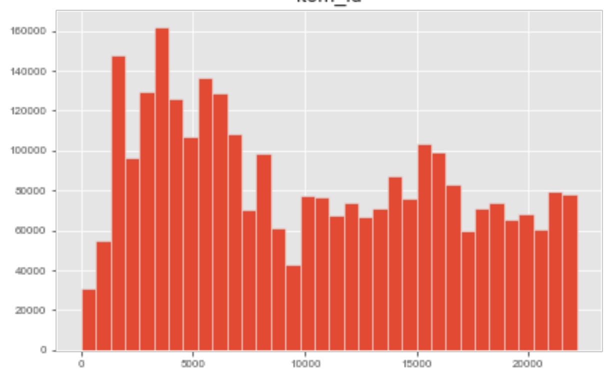
```
4  15.01.2013                 0        25      2555      1099.00            1.0
-----------Information-----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2935849 entries, 0 to 2935848
Data columns (total 6 columns):
date              object
date_block_num    int64
shop_id           int64
item_id           int64
item_price        float64
item_cnt_day      float64
dtypes: float64(2), int64(3), object(1)
memory usage: 134.4+ MB
None
-----------Data Types-----------
date               object
date_block_num      int64
shop_id             int64
item_id             int64
item_price        float64
item_cnt_day      float64
dtype: object
----------Missing value-----------
date              0
date_block_num    0
shop_id           0
item_id           0
item_price        0
item_cnt_day      0
dtype: int64
----------Null value-----------
date              0
date_block_num    0
shop_id           0
item_id           0
item_price        0
item_cnt_day      0
dtype: int64
----------Shape of Data----------
(2935849, 6)
{dtype('int64'), dtype('O'), dtype('float64')}
```

date_block_num

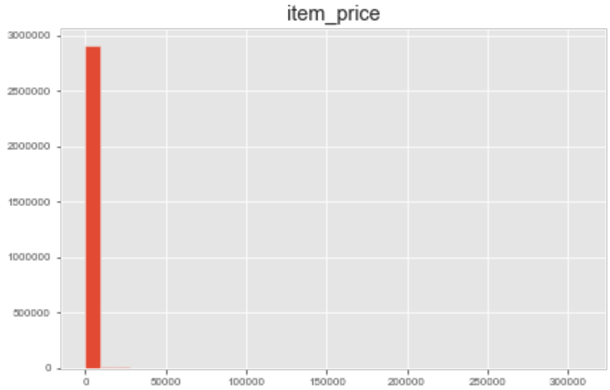
item_cnt_day



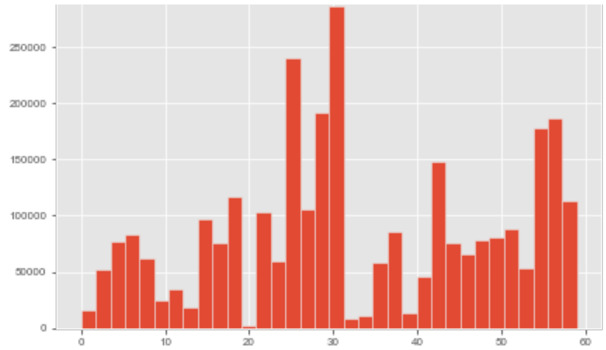
item_id



item_price



shop_id



In [7]:

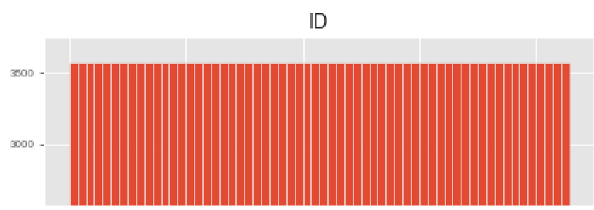
```
# Drop Duplicate Data
subset = ['date', 'date_block_num', 'shop_id', 'item_id','item_cnt_day']
drop_duplicate(train, subset = subset)
```

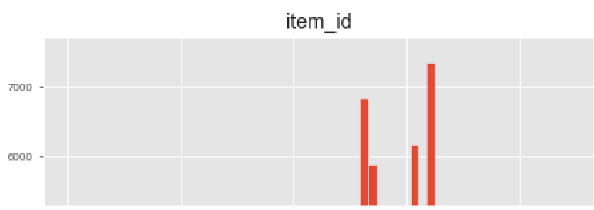
```
Before drop shape: (2935849, 6)
After drop shape: (2935825, 6)
Total Duplicate: 24
```

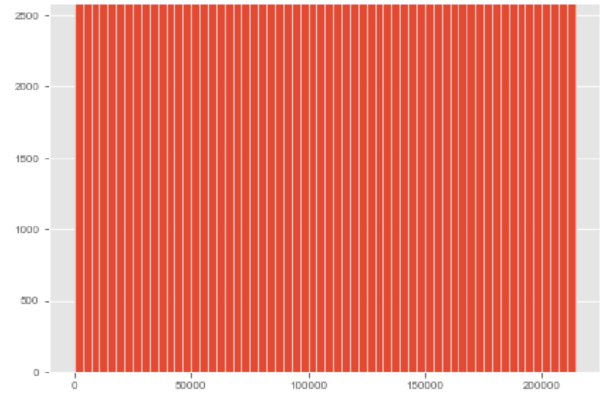
In [8]:

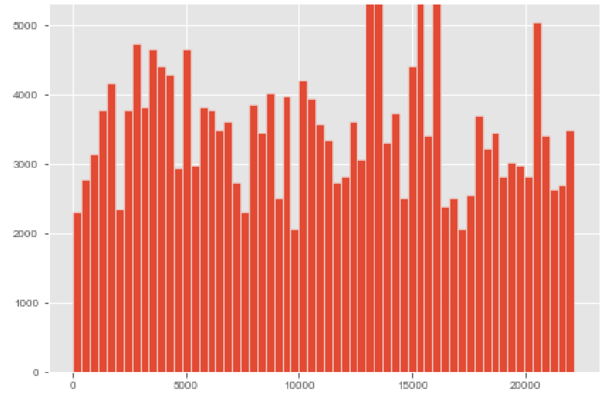
```
# test insight
eda(test)
graph_insight(test, bins=60)
```

```
----------Top-5- Record----------
   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
-----------Information-----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214200 entries, 0 to 214199
Data columns (total 3 columns):
ID         214200 non-null int64
shop_id    214200 non-null int64
item_id    214200 non-null int64
dtypes: int64(3)
memory usage: 4.9 MB
None
-----------Data Types-----------
ID         int64
shop_id    int64
item_id    int64
dtype: object
----------Missing value-----------
ID         0
shop_id    0
item_id    0
dtype: int64
----------Null value-----------
ID         0
shop_id    0
item_id    0
dtype: int64
----------Shape of Data----------
(214200, 3)
{dtype('int64')}
```

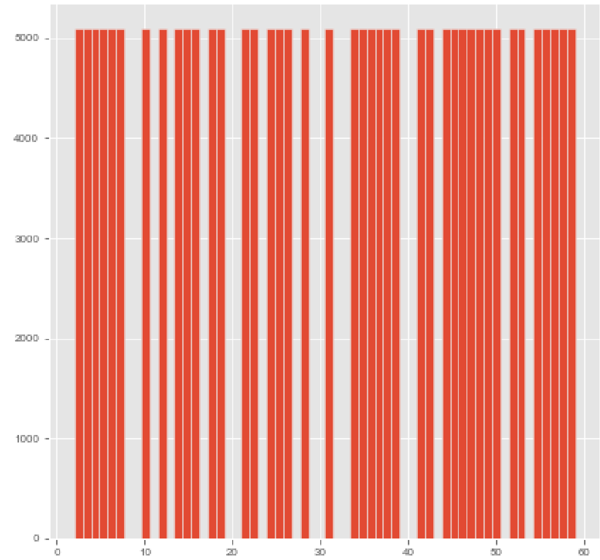








shop_id

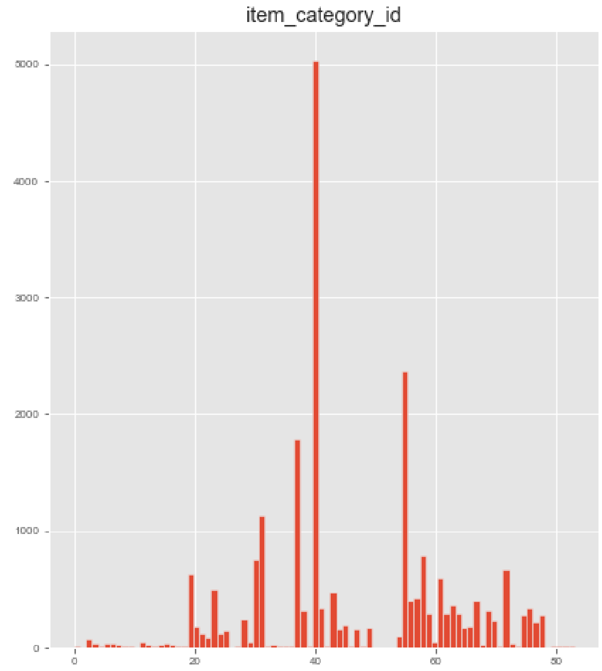


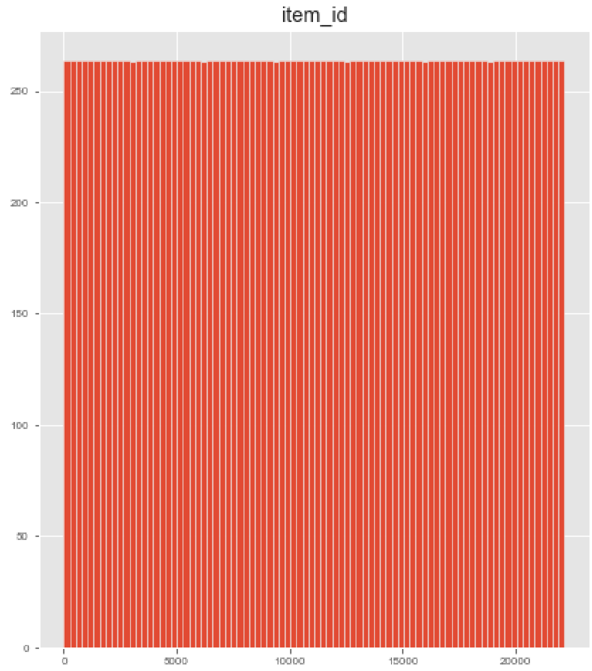
In [9]:

```
eda(items)
graph_insight(items, bins=84, figsize=(16,8))
```

```
----------Top-5- Record----------
                                               item_name  item_id  \
0          ! ВО ВЛАСТИ НАВАЖДЕНИЯ (ПЛАСТ.)         D        0
1  !ABBYY FineReader 12 Professional Edition Full...        1
2      ***В ЛУЧАХ СЛАВЫ   (UNV)                    D        2
3    ***ГОЛУБАЯ ВОЛНА  (Univ)                      D        3
4        ***КОРОБКА (СТЕКЛО)                       D        4

   item_category_id
0                40
1                76
2                40
3                40
4                40
-----------Information-----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22170 entries, 0 to 22169
Data columns (total 3 columns):
item_name           22170 non-null object
item_id             22170 non-null int64
item_category_id    22170 non-null int64
dtypes: int64(2), object(1)
memory usage: 519.7+ KB
None
-----------Data Types-----------
item_name           object
item_id              int64
item_category_id     int64
dtype: object
```

```
----------Missing value-----------
item_name           0
item_id             0
item_category_id    0
dtype: int64
----------Null value-----------
item_name           0
item_id             0
item_category_id    0
dtype: int64
----------Shape of Data----------
(22170, 3)
{dtype('int64'), dtype('O')}
```





In [10]:

```
eda(item_cats)
# graph_insight(item_cats)
```

```
----------Top-5- Record----------
        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
-----------Information-----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84 entries, 0 to 83
Data columns (total 2 columns):
item_category_name    84 non-null object
item_category_id      84 non-null int64
dtypes: int64(1), object(1)
memory usage: 1.4+ KB
None
-----------Data Types-----------
item_category_name    object
item_category_id       int64
dtype: object
----------Missing value-----------
item_category_name    0
item_category_id      0
dtype: int64
----------Null value-----------
item_category_name    0
item_category_id      0
dtype: int64
----------Shape of Data----------
(84, 2)
```

In [11]:

```
eda(shops)
# graph_insight(shops)
```

```
----------Top-5- Record----------
                        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
-----------Information-----------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60 entries, 0 to 59
Data columns (total 2 columns):
shop_name    60 non-null object
shop_id      60 non-null int64
dtypes: int64(1), object(1)
memory usage: 1.0+ KB
None
-----------Data Types-----------
shop_name    object
shop_id       int64
dtype: object
----------Missing value-----------
shop_name    0
shop_id      0
dtype: int64
----------Null value-----------
shop_name    0
shop_id      0
dtype: int64
----------Shape of Data----------
(60, 2)
```

In [12]:

```
def unresanable_data(data):
    print("Min Value:",data.min())
    print("Max Value:",data.max())
    print("Average Value:",data.mean())
    print("Center Point of Data:",data.median())
```

In [13]:

```
unresanable_data(train['item_price'])
```

```
Min Value: -1.0
Max Value: 307980.0
Average Value: 890.8557861463909
Center Point of Data: 399.0
```

In [14]:

```
# -1 and 307980 looks like outliers, let's delete them
print('before train shape:', train.shape)
train = train[(train.item_price > 0) & (train.item_price < 300000)]
print('after train shape:', train.shape)
```

```
before train shape: (2935825, 6)
after train shape: (2935823, 6)
```

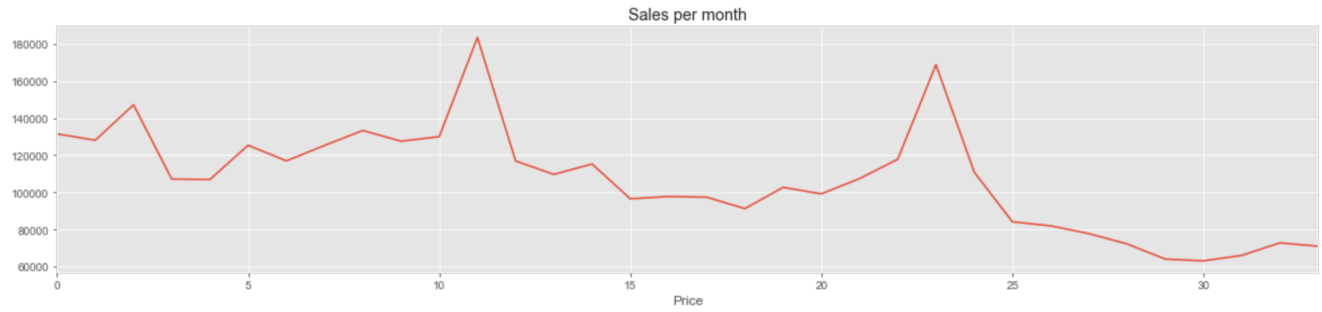
In [34]:

```
plt.figure(figsize = (20,4))
sns.tsplot(train.groupby('date_block_num').sum()['item_cnt_day'])
plt.title('Sales per month')
plt.xlabel('Price')
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\seaborn\timeseries.py:183:
UserWarning: The `tsplot` function is deprecated and will be removed in a future release. Please u
pdate your code to use the new `lineplot` function.
  warnings.warn(msg, UserWarning)
```

Out[34]:

```
Text(0.5,0,'Price')
```

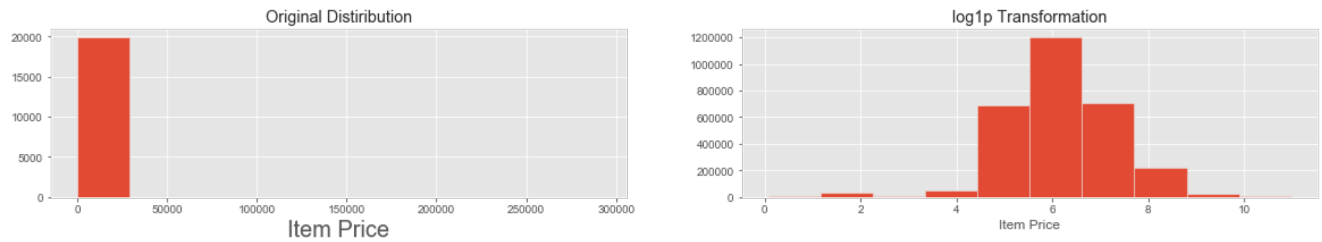


Distribution Checking

In [16]:

```python
unresanable_data(train['item_price'])
count_price = train.item_price.value_counts().sort_index(ascending=False)
plt.subplot(221)
count_price.hist(figsize=(20,6))
plt.xlabel('Item Price', fontsize=20);
plt.title('Original Distiribution')

plt.subplot(222)
train.item_price.map(np.log1p).hist(figsize=(20,6))
plt.xlabel('Item Price');
plt.title('log1p Transformation')
train.loc[:,'item_price'] = train.item_price.map(np.log1p)
unresanable_data(train['item_price'])
```

```
Min Value: 0.07
Max Value: 59200.0
Average Value: 890.7514892291624
Center Point of Data: 399.0
Min Value: 0.06765864847381481
Max Value: 10.988693712621323
Average Value: 6.163376384893391
Center Point of Data: 5.991464547107982
```



In [17]:

```python
train.sort_values(['date_block_num','shop_id','item_id'],inplace=True)
train.head()
```

Out[17]:

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
40084	03.01.2013	0	0	32	5.402677	2.0

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
40085	21.01.2013	0	0	32	5.402677	2.0
40086	25.01.2013	0	0	32	5.402677	1.0
40087	31.01.2013	0	0	32	5.402677	1.0
40088	03.01.2013	0	0	33	5.852202	1.0

In [66]:

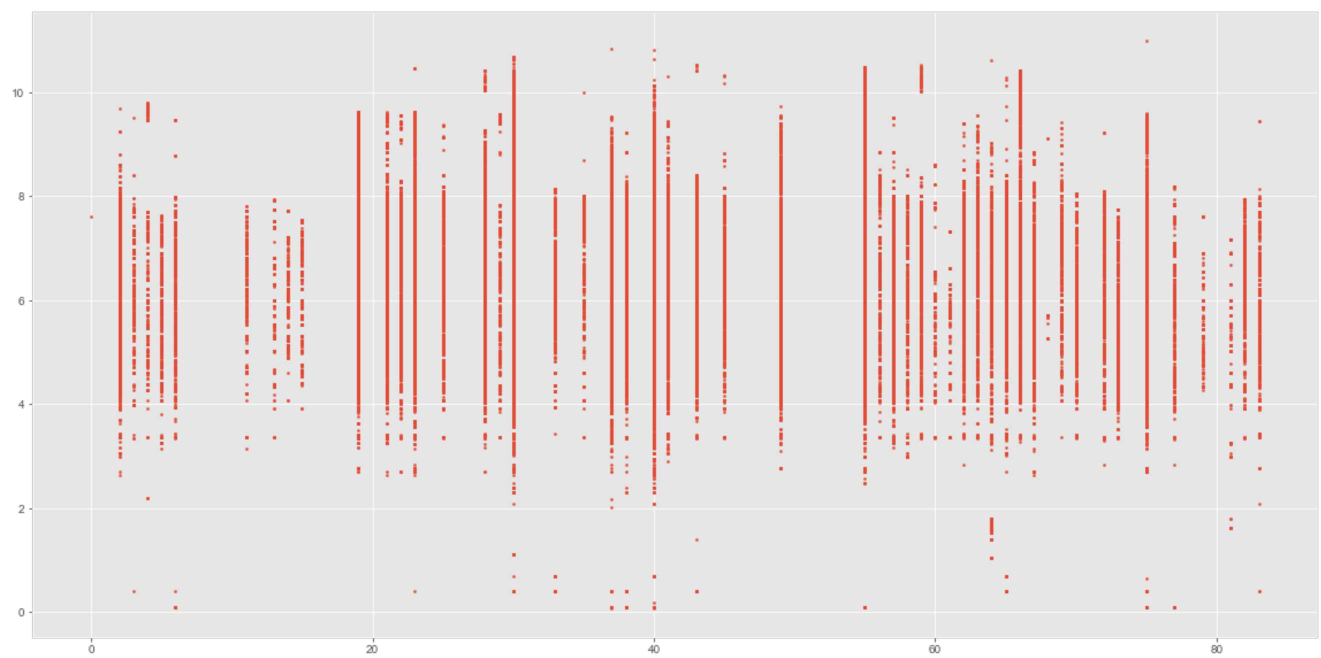
```
l_cat = list(item_cats.item_category_name)

for ind in range(0,1):l_cat[ind] = 'PC Headsets / Headphones'
for ind in range(1,8):l_cat[ind] = 'Access'
l_cat[8] = 'Tickets (figure)'
l_cat[9] = 'Delivery of goods'
for ind in range(10,18):l_cat[ind] = 'Consoles'
for ind in range(18,25):l_cat[ind] = 'Consoles Games'
l_cat[25] = 'Accessories for games'
for ind in range(26,28):l_cat[ind] = 'phone games'
for ind in range(28,32):l_cat[ind] = 'CD games'
for ind in range(32,37):l_cat[ind] = 'Card'
for ind in range(37,43):l_cat[ind] = 'Movie'
for ind in range(43,55):l_cat[ind] = 'Books'
for ind in range(55,61):l_cat[ind] = 'Music'
for ind in range(61,73):l_cat[ind] = 'Gifts'
for ind in range(73,79):l_cat[ind] = 'Soft'
for ind in range(79,81):l_cat[ind] = 'Office'
for ind in range(81,83):l_cat[ind] = 'Clean'
l_cat[83] = 'Elements of a food'
```

In [29]:

```
plt.figure(figsize = (20,10))
plt.scatter(train['item_category_id'], train['item_price'], s=4)
```

Out[29]:

```
<matplotlib.collections.PathCollection at 0x1bb081b3a90>
```



Convert the `item['date']` to pandas date format

In [25]:

```
train['date'] = pd.to_datetime(train['date'], format = '%d.%m.%Y')
```

Add **item_category_id** to sales as a feature

Add **item_category_id** to **sales** as a feature

In [26]:

```
train['item_category_id'] = train['item_id'].map(train['item_id'].map(items['item_category_id']))
train.head()
```

Out[26]:

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_category_id
40084	2013-01-03	0	0	32	5.402677	2.0	23
40085	2013-01-21	0	0	32	5.402677	2.0	23
40086	2013-01-25	0	0	32	5.402677	1.0	23
40087	2013-01-31	0	0	32	5.402677	1.0	23
40088	2013-01-03	0	0	33	5.852202	1.0	19

In [60]:

```
# Correct sale_train values
sales['item_price'][2909818] = np.nan
sales['item_cnt_day'][2909818] = np.nan
sales['item_price'][2909818] = sales[(sales['shop_id'] ==12) & (sales['item_id'] == 11373) & (sales
['date_block_num'] == 33)]['item_price'].median()
sales['item_cnt_day'][2909818] = round(sales[(sales['shop_id'] ==12) & (sales['item_id'] == 11373)
& (sales['date_block_num'] == 33)]['item_cnt_day'].median())
sales['item_price'][885138] = np.nan
sales['item_price'][885138] = sales[(sales['item_id'] == 11365) & (sales['shop_id'] ==12) & (sales[
'date_block_num'] == 8)]['item_price'].median()
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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

C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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

C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  import sys
```

```
import sys
```

3. Feature Engineering

In [61]:

```
sales = sales.merge(test[['shop_id']].drop_duplicates(), how = 'inner')
sales['date'] = pd.to_datetime(sales['date'], format='%d.%m.%Y')
```

3.1 Get a feature matrix

In [62]:

```
# Create "grid" with columns
index_cols = ['shop_id', 'item_id', 'date_block_num']

# For every month we create a grid from all shops/items combinations from that month
grid = []
for block_num in sales['date_block_num'].unique():
    cur_shops = sales.loc[sales['date_block_num'] == block_num, 'shop_id'].unique()
    cur_items = sales.loc[sales['date_block_num'] == block_num, 'item_id'].unique()
    grid.append(np.array(list(product(*[cur_shops, cur_items, [block_num]])),dtype='int32'))

# Turn the grid into a dataframe
grid = pd.DataFrame(np.vstack(grid), columns = index_cols,dtype=np.int32)

# Groupby data to get shop-item-month aggregates
sales['item_cnt_day'] = sales['item_cnt_day'].clip(0,20)
gb = sales.groupby(index_cols,as_index=False)['item_cnt_day'].agg('sum').rename(columns = {'item_cn
t_day': 'item_cnt_month'})
gb['item_cnt_month'] = gb['item_cnt_month'].clip(0,20).astype(np.int)
# Join it to the grid
all_data = pd.merge(grid, gb, how='left', on=index_cols).fillna(0)

# Same as above but with shop-month aggregates
gb = sales.groupby(['shop_id', 'date_block_num'],as_index=False)['item_cnt_day'].agg('sum').rename(
columns = {'item_cnt_day': 'target_shop'})
gb['target_shop'] = gb['target_shop'].clip(0,20).astype(np.int)
# Join it to the grid
all_data = pd.merge(all_data, gb, how='left', on=['shop_id', 'date_block_num']).fillna(0)

# Same as above but with item-month aggregates
gb = sales.groupby(['item_id', 'date_block_num'],as_index=False)['item_cnt_day'].agg('sum').rename(
columns = {'item_cnt_day': 'target_item'})
gb['target_item'] = gb['target_item'].clip(0,20).astype(np.int)
# Join it to the grid
all_data = pd.merge(all_data, gb, how='left', on=['item_id', 'date_block_num']).fillna(0)
```

In [63]:

```
def downcast_dtypes(df):
    '''
        Changes column types in the dataframe:

                `float64` type to `float32`
                `int64`   type to `int32`
    '''

    # Select columns to downcast
    float_cols = [c for c in df if df[c].dtype == "float64"]
    int_cols =   [c for c in df if df[c].dtype == "int64"]

    # Downcast
    df[float_cols] = df[float_cols].astype(np.float32)
    df[int_cols]   = df[int_cols].astype(np.int32)

    return df

# Downcast dtypes from 64 to 32 bit to save memory
all_data = downcast_dtypes(all_data)
del grid, gb
```

```
gc.collect();
```

In [64]:

```
all_data.sort_values(['date_block_num','shop_id','item_id'],inplace=True)
all_data.head()
```

Out[64]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item
84203	2	19	0	0.0	20	1
86531	2	27	0	1.0	20	6
88912	2	28	0	0.0	20	8
87693	2	29	0	0.0	20	4
83623	2	32	0	0.0	20	20

In [30]:

```
# Sanity check
print(sales['item_cnt_day'].sum())
print(all_data['item_cnt_month'].sum())
```

```
3582120.0
3261311.0
```

Add **item_category_id** to **sales** as a feature

In [65]:

```
all_data = all_data.merge(items[['item_id', 'item_category_id']], on = ['item_id'], how = 'left')
test = test.merge(items[['item_id', 'item_category_id']], on = ['item_id'], how = 'left')
```

In [20]:

```
all_data.head()
```

Out[20]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id
0	2	19	0	0.0	20	1	40
1	2	27	0	1.0	20	6	19
2	2	28	0	0.0	20	8	30
3	2	29	0	0.0	20	4	23
4	2	32	0	0.0	20	20	40

In [21]:

```
item_cats.shape
```

Out[21]:

```
(84, 2)
```

In [66]:

```
l_cat = list(item_cats.item_category_name)

for ind in range(0,1):l_cat[ind] = 'PC Headsets / Headphones'
for ind in range(1,8):l_cat[ind] = 'Access'
```

```python
l_cat[8] = 'Tickets (figure)'
l_cat[9] = 'Delivery of goods'
for ind in range(10,18):l_cat[ind] = 'Consoles'
for ind in range(18,25):l_cat[ind] = 'Consoles Games'
l_cat[25] = 'Accessories for games'
for ind in range(26,28):l_cat[ind] = 'phone games'
for ind in range(28,32):l_cat[ind] = 'CD games'
for ind in range(32,37):l_cat[ind] = 'Card'
for ind in range(37,43):l_cat[ind] = 'Movie'
for ind in range(43,55):l_cat[ind] = 'Books'
for ind in range(55,61):l_cat[ind] = 'Music'
for ind in range(61,73):l_cat[ind] = 'Gifts'
for ind in range(73,79):l_cat[ind] = 'Soft'
for ind in range(79,81):l_cat[ind] = 'Office'
for ind in range(81,83):l_cat[ind] = 'Clean'
l_cat[83] = 'Elements of a food'
```

In [67]:

```python
from sklearn import preprocessing

lb = preprocessing.LabelEncoder()
item_cats['item_cat_id_fix'] = lb.fit_transform(l_cat)

all_data = all_data.merge(item_cats[['item_cat_id_fix', 'item_category_id']], on =
['item_category_id'], how = 'left')
test = test.merge(item_cats[['item_cat_id_fix', 'item_category_id']], on = ['item_category_id'], h
ow = 'left')
all_data.head()
```

Out[67]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix
0	2	19	0	0.0	20	1	40	11
1	2	27	0	1.0	20	6	19	7
2	2	28	0	0.0	20	8	30	3
3	2	29	0	0.0	20	4	23	7
4	2	32	0	0.0	20	20	40	11

In [68]:

```python
del items, item_cats
gc.collect();
```

3.3 Mean encodings features

In [23]:

```python
all_data.head()
```

Out[23]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix
0	0	19	0	0.0	20	1	40	11
1	0	27	0	0.0	20	7	19	7
2	0	28	0	0.0	20	8	30	3
3	0	29	0	0.0	20	5	23	7
4	0	32	0	6.0	20	20	40	11

3.3.1 KFold scheme regularization

In [69]:

```
from sklearn.model_selection import KFold

mean_encoded_col = ['shop_id', 'item_id', 'item_category_id', 'item_cat_id_fix']
global_mean =  all_data['item_cnt_month'].mean()

for col in tqdm_notebook(mean_encoded_col):
    kf = KFold(n_splits=5,shuffle=False, random_state = 0)
    all_data[col+'_enc_kf'] = np.nan

    for train_index , test_index in kf.split(all_data['item_cnt_month'].values):
        x_tr, x_val = all_data.iloc[train_index], all_data.iloc[test_index]
        means = x_val[col].map(x_tr.groupby(col).item_cnt_month.mean())
        all_data[col+'_enc_kf'].iloc[test_index] = means

    # Fill NaNs
    all_data[col+'_enc_kf'].fillna(global_mean, inplace=True)
    corr = np.corrcoef(all_data['item_cnt_month'].values, all_data[col+'_enc_kf'])[0][1]
    print(col+'_enc_kf',corr)

    # Drop if correlation < 0.3
#     if corr < 0.3:
#         all_data.drop(columns=[col+'_enc_kf'], inplace=True)
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\pandas\core\indexing.py:189:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
```

```
shop_id_enc_kf 0.17337045864056277
item_id_enc_kf 0.31586214733336654
item_category_id_enc_kf 0.27407179596715203
item_cat_id_fix_enc_kf 0.15732343270016874
```

In [25]:

```
all_data.head()
```

Out[25]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix	item_id
0	0	19	0	0.0	20	1	40	11	0.29882
1	0	27	0	0.0	20	7	19	7	0.04852
2	0	28	0	0.0	20	8	30	3	0.14242
3	0	29	0	0.0	20	5	23	7	0.03030
4	0	32	0	6.0	20	20	40	11	0.89553

3.3.2 Leave-one-out scheme regularization

In [70]:

```
for col in tqdm_notebook(mean_encoded_col):
    all_data[col+'_enc_loo'] = np.nan

    all_data[col+'_enc_loo'] = all_data.groupby(col)['item_cnt_month'].transform('sum') - all_data[
'item_cnt_month']
    all_data[col+'_enc_loo'] /= (all_data.groupby(col)['item_cnt_month'].transform('count')-1)

    # Fill NaNs
    all_data[col+'_enc_loo'].fillna(global_mean, inplace=True)
```

```
    corr = np.corrcoef(all_data['item_cnt_month'].values, all_data[col+'_enc_loo'])[0][1]
    print(col+'_enc_loo',corr)

    # Drop if correlation < 0.3
#     if corr < 0.3:
#         all_data.drop(columns=[col+'_enc_loo'], inplace=True)
```

```
shop_id_enc_loo 0.17554681313306436
item_id_enc_loo 0.4819368416124951
item_category_id_enc_loo 0.2927779834798382
item_cat_id_fix_enc_loo 0.17159342731824384
```

In [27]:

```
all_data.head()
```

Out[27]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix	item_id
0	0	19	0	0.0	20	1	40	11	0.29882
1	0	27	0	0.0	20	7	19	7	0.04852
2	0	28	0	0.0	20	8	30	3	0.14242
3	0	29	0	0.0	20	5	23	7	0.03030
4	0	32	0	6.0	20	20	40	11	0.89553

3.3.3 Smoothing regularization

In [71]:

```
for col in tqdm_notebook(mean_encoded_col):
    all_data[col+'_enc_smoo'] = np.nan

    alpha = 100
    mean_target = all_data.groupby(col)['item_cnt_month'].transform('mean')
    nrow = all_data.groupby(col)['item_cnt_month'].transform('count')
    all_data[col+'_enc_smoo'] = (mean_target*nrow + 0.3343*alpha)/(nrow+alpha)

    # Fill NaNs
    all_data[col+'_enc_smoo'].fillna(global_mean, inplace=True)
    corr = np.corrcoef(all_data['item_cnt_month'].values, all_data[col+'_enc_smoo'])[0][1]
    print(col+'_enc_smoo',corr)

    # Drop if correlation < 0.3
#     if corr < 0.3:
#         all_data.drop(columns=[col+'_enc_smoo'], inplace=True)
```

```
shop_id_enc_smoo 0.17557235554506093
item_id_enc_smoo 0.47976679311611814
item_category_id_enc_smoo 0.2927316906354779
item_cat_id_fix_enc_smoo 0.1716387693172654
```

In [29]:

```
all_data.head()
```

Out[29]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix	item_id
0	0	19	0	0.0	20	1	40	11	0.29882
1	0	27	0	0.0	20	7	19	7	0.04852

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix	item_id
2	0	28	0	0.0	20	8	30	3	0.14242
3	0	29	0	0.0	20	5	23	7	0.03030
4	0	32	0	6.0	20	20	40	11	0.89553

3.3.4 Expanding mean scheme regularization

In [72]:

```
for col in tqdm_notebook(mean_encoded_col):
    all_data[col+'_enc_expan'] = np.nan

    cumsum = all_data.groupby(col)['item_cnt_month'].cumsum()-all_data['item_cnt_month']
    cumcnt = all_data.groupby(col)['item_cnt_month'].cumcount()
    all_data[col+'_enc_expan'] = cumsum/cumcnt

    # Fill NaNs
    all_data[col+'_enc_expan'].fillna(global_mean, inplace=True)
    corr = np.corrcoef(all_data['item_cnt_month'].values, all_data[col+'_enc_expan'])[0][1]
    print(col+'_enc_expan',corr)

    # Drop if correlation < 0.3
#     if corr < 0.3:
#         all_data.drop(columns=[col+'_enc_expan'], inplace=True)
```

```
shop_id_enc_expan 0.17574555214870394
item_id_enc_expan 0.5656456891458477
item_category_id_enc_expan 0.2961041783359048
item_cat_id_fix_enc_expan 0.1768452301105094
```

In [31]:

```
all_data.head()
```

Out[31]:

	shop_id	item_id	date_block_num	item_cnt_month	target_shop	target_item	item_category_id	item_cat_id_fix	item_id
0	0	19	0	0.0	20	1	40	11	0.29882
1	0	27	0	0.0	20	7	19	7	0.04852
2	0	28	0	0.0	20	8	30	3	0.14242
3	0	29	0	0.0	20	5	23	7	0.03030
4	0	32	0	6.0	20	20	40	11	0.89553

3.3 Lag-based features

We generate both train and test lag-based features together.

In [73]:

```
# Uncomment following when submitting
if Validation == False:
    test['date_block_num'] = 34
    all_data = pd.concat([all_data, test], axis = 0)
    all_data = all_data.drop(columns = ['ID'])
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:4:
FutureWarning: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.
```

```
after removing the cwd from sys.path.
```

In [35]:

```
all_data
```

Out[35]:

	date_block_num	item_cat_id_fix	item_category_id	item_cnt_month	item_id	item_id_enc_expan	item_id_enc_kf
0	0	11	40	0.0	19	0.298823	0.298823
1	0	7	19	0.0	27	0.298823	0.048523
2	0	3	30	0.0	28	0.298823	0.142424
3	0	7	23	0.0	29	0.298823	0.030303
4	0	11	40	6.0	32	0.298823	0.895534
5	0	11	37	3.0	33	0.298823	0.487509
6	0	11	40	0.0	34	0.298823	0.107018
7	0	11	40	1.0	35	0.298823	0.020408
8	0	12	57	0.0	40	0.298823	0.054717
9	0	12	57	0.0	41	0.298823	0.038136
10	0	12	57	0.0	42	0.298823	0.063331
11	0	11	40	1.0	43	0.298823	0.040000
12	0	12	57	0.0	44	0.298823	0.078091
13	0	12	57	0.0	45	0.298823	0.106737
14	0	12	57	0.0	46	0.298823	0.049808
15	0	12	57	0.0	47	0.298823	0.043981
16	0	12	57	0.0	48	0.298823	0.040388
17	0	12	57	0.0	49	0.298823	0.099167
18	0	12	57	0.0	50	0.298823	0.298823
19	0	12	57	2.0	51	0.298823	0.117335
20	0	12	57	0.0	52	0.298823	0.046131
21	0	12	57	0.0	53	0.298823	0.117335
22	0	12	57	0.0	54	0.298823	0.056738
23	0	2	43	0.0	55	0.298823	0.124561
24	0	12	57	0.0	56	0.298823	0.042463
25	0	12	57	0.0	57	0.298823	0.098410
26	0	12	57	0.0	59	0.298823	0.131455
27	0	2	43	0.0	60	0.298823	0.067138
28	0	2	43	1.0	61	0.298823	0.000000
29	0	11	40	0.0	63	0.298823	0.036364
30	0	11	40	0.0	72	0.298823	0.048170
31	0	11	40	0.0	73	0.298823	0.018182
32	0	11	40	1.0	75	0.298823	0.054545
33	0	11	40	0.0	84	0.298823	0.021978
34	0	11	37	0.0	85	0.298823	0.065020
35	0	11	40	0.0	86	0.298823	0.298823
36	0	11	40	1.0	88	0.298823	0.037815
37	0	11	40	0.0	89	0.298823	0.040268

	date_block_num	item_cat_id_fix	item_cat_id_fix_enc_expan	item_cat_id_fix_enc_kf	item_cat_id_fix_enc_loo	ite
8812232	34	11	0.000000	0.000000	0.000000	0.(
8812233	34	10	0.000000	0.000000	0.000000	0.(
8812234	34	12	0.000000	0.000000	0.000000	0.(
8812235	34	7	0.000000	0.000000	0.000000	0.(
8812236	34	7	0.000000	0.000000	0.000000	0.(
8812237	34	10	0.000000	0.000000	0.000000	0.(
8812238	34	10	0.000000	0.000000	0.000000	0.(
8812239	34	12	0.000000	0.000000	0.000000	0.(
8812240	34	10	0.000000	0.000000	0.000000	0.(
8812241	34	12	0.000000	0.000000	0.000000	0.(
8812242	34	11	0.000000	0.000000	0.000000	0.(
8812243	34	11	0.000000	0.000000	0.000000	0.(

5459310 rows × 157 columns

In [76]:

```
lag_cols = [col for col in all_data.columns if col[-1] in [str(item) for item in shift_range]]
all_data = downcast_dtypes(all_data)
```

3.4 Date features

In [77]:

```
# Get dates from *sales dataframe*
sales['date'] = pd.to_datetime(sales['date'], format='%d.%m.%Y')
dates_train = sales[['date', 'date_block_num']].drop_duplicates()
dates_test = dates_train[dates_train['date_block_num'] == 34-12]
dates_test['date_block_num'] = 34
dates_test['date'] = dates_test['date'] + pd.DateOffset(years=1)
dates_all = pd.concat([dates_train, dates_test])


# Generate date features
dates_all['dow'] = dates_all['date'].dt.dayofweek
dates_all['year'] = dates_all['date'].dt.year
dates_all['month'] = dates_all['date'].dt.month

# Convert categorical variable into dummy/indicator variables
dates_all = pd.get_dummies(dates_all, columns=['dow'])
dow_col = ['dow_' + str(x) for x in range(7)]
date_features = dates_all.groupby(['year', 'month', 'date_block_num'])[dow_col].agg('sum').reset_in
dex()
date_features['days_of_month'] = date_features[dow_col].sum(axis=1)
date_features['year'] = date_features['year'] - 2013

# Merge date_features to all_data
#date_features = date_features[['month', 'year', 'days_of_month', 'date_block_num']]
all_data = all_data.merge(date_features, on = 'date_block_num', how = 'left')
date_columns = date_features.columns.difference(set(index_cols))
all_data.head()
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:5:
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: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  """
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:6:
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
```

```
Try using .loc[row_indexer,col_indexer] = value instead

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

Out[77]:

	date_block_num	item_cat_id_fix	item_cat_id_fix_enc_expan	item_cat_id_fix_enc_kf	item_cat_id_fix_enc_loo	item_cat_
0	12	11	0.240344	0.223743	0.227192	0.227196
1	12	11	0.240344	0.223743	0.227192	0.227196
2	12	11	0.240344	0.223743	0.227192	0.227196
3	12	11	0.240344	0.223743	0.227192	0.227196
4	12	11	0.240345	0.223743	0.227192	0.227196

5 rows × 167 columns

3.5 Scale feature columns

In [78]:

```
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler and split data to assure the same distribution
sc = StandardScaler()
train = all_data[all_data['date_block_num']!= all_data['date_block_num'].max()]
test = all_data[all_data['date_block_num']== all_data['date_block_num'].max()]

# Select columns to be standardized
to_drop_cols = ['date_block_num']
feature_columns = list(set(lag_cols + index_cols + list(date_columns)).difference(to_drop_cols))

# fit_transform
train[feature_columns] = sc.fit_transform(train[feature_columns])
test[feature_columns] = sc.transform(test[feature_columns])
all_data = pd.concat([train, test], axis = 0)

# Downcast for standardized data
all_data = downcast_dtypes(all_data)
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:13:
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: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  del sys.path[0]
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\pandas\core\indexing.py:543:
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: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self.obj[item] = s
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:14:
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: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

In [41]:

```
all_data.head()
```

Out[41]:

	date_block_num	item_cat_id_fix	item_category_id	item_cnt_month	item_id	item_id_enc_expan	item_id_enc_kf	item
0	12	-0.651193	-1.650131	0.0	-1.792746	0.065814	0.067542	0.05
1	12	0.471781	-0.312874	0.0	-1.792268	2.735060	1.220146	1.08
2	12	0.471781	-0.503911	0.0	-1.792108	1.958167	0.995126	0.86
3	12	0.471781	-0.312874	1.0	-1.791949	2.404022	1.370690	1.28
4	12	0.471781	-0.503911	1.0	-1.791789	0.767824	0.525862	0.52

4. Validation

The **two-level stacking** model is used to take advantages from different models(linear, Xgboost, NN, etc).

Save `date_block_num`, as we can't use them as features, but will need them to split the dataset into parts.

In [79]:

```
dates = all_data['date_block_num']
last_block = dates.max()
print('Test `date_block_num` is %d' % last_block)
print('Number of features; ' ,len(feature_columns))
```

```
Test `date_block_num` is 34
Number of features;  147
```

4.1 First-level model

Here the **time-series validation** scheme is used.

- 1. Split the train data into chunks of duration **T**. Select first **M** chunks.
- 1. Fit **N** diverse models on those **M** chunks and predict for the chunk **M+1**. Then fit those models on first **M+1** chunks and predict for chunk **M+2** and so on, until you hit the end. After that use all train data to fit models and get predictions for test. Now we will have **meta-features** for the chunks starting from number **M+1** as well as **meta-features** for the test.
- 1. Now we can use **meta-features** from first **K** chunks **[M+1,M+2,..,M+K]** to fit level 2 models and validate them on chunk **M+K+1**. Essentially we are back to step 1. with the lesser amount of chunks and **meta-features** instead of features.

In [80]:

```
from sklearn.metrics import mean_squared_error
from math import sqrt

num_first_level_models = 3
scoringMethod = 'r2'

# Train meta-features M = 15 (12 + 15 = 27)
months_to_generate_meta_features = range(27,last_block +1)
mask = dates.isin(months_to_generate_meta_features)
Target = 'item_cnt_month'
y_all_level2 = all_data[Target][mask].values
X_all_level2 = np.zeros([y_all_level2.shape[0], num_first_level_models])
```

Now fill `X_train_level2` with **meta-features**

In [82]:

```
slice_start = 0
SEED = 0

for cur_block_num in tqdm_notebook(months_to_generate_meta_features):
    # Use M chunks to predict for the chunk M+1
    cur_X_train = all_data.loc[dates <  cur_block_num][feature_columns]
    cur_X_test =  all_data.loc[dates == cur_block_num][feature_columns]
    cur_y_train = all_data.loc[dates <  cur_block_num, Target].values
    cur_y_test =  all_data.loc[dates == cur_block_num, Target].values

    # Create Numpy arrays of train, test and target dataframes to feed into models
    train_x = cur_X_train.values
    train_y = cur_y_train.ravel()
    test_x = cur_X_test.values
    test_y = cur_y_test.ravel()
    preds = []

    from sklearn.linear_model import (LinearRegression, SGDRegressor)
    import lightgbm as lgb

    sgdr= SGDRegressor(
        penalty = 'l2' ,
        random_state = SEED )
    lgb_params = {'feature_fraction': 0.75,
                  'metric': 'rmse',
                  'nthread':16,
                  'min_data_in_leaf': 2**7,
                  'bagging_fraction': 0.75,
                  'learning_rate': 0.03,
                  'objective': 'mse',
                  'bagging_seed': 2**7,
                  'num_leaves': 2**7,
                  'bagging_freq':1,
                  'verbose':0}
    estimators = [sgdr]

    for estimator in estimators:
        estimator.fit(train_x, train_y)
        pred_test = estimator.predict(test_x)
        preds.append(pred_test)

        # pred_train = estimator.predict(train_x)
        # print('Train RMSE for %s is %f' % (estimator.__class__.__name__,
sqrt(mean_squared_error(cur_y_train, pred_train))))
        # print('Test RMSE for %s is %f' % (estimator.__class__.__name__,
sqrt(mean_squared_error(cur_y_test, pred_test))))

    print('Training Model %d: %s'%(len(preds), 'lightgbm'))
    estimator = lgb.train(lgb_params, lgb.Dataset(train_x, label=train_y), 300)
    pred_test = estimator.predict(test_x)
    preds.append(pred_test)

    # pred_train = estimator.predict(train_x)
    # print('Train RMSE for %s is %f' % ('lightgbm', sqrt(mean_squared_error(cur_y_train,
pred_train))))
    # print('Test RMSE for %s is %f' % ('lightgbm', sqrt(mean_squared_error(cur_y_test,
pred_test))))

    print('Training Model %d: %s'%(len(preds), 'keras'))
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.wrappers.scikit_learn import KerasRegressor

    def baseline_model():
        # create model
        model = Sequential()
        model.add(Dense(50, input_dim=train_x.shape[1], kernel_initializer='glorot_normal', activat
ion='softplus'))
        model.add(Dropout(0.2))
        model.add(Dense(50, kernel_initializer='glorot_normal', activation = 'relu'))
        model.add(Dropout(0.2))
        model.add(Dense(20, kernel_initializer='glorot_normal', activation = 'relu'))
        model.add(Dropout(0.2))
        model.add(Dense(20, kernel_initializer='glorot_normal', activation = 'relu'))
```

```
        model.add(Dropout(0.2))
        model.add(Dense(1, kernel_initializer='glorot_normal', activation = 'relu'))

        # Compile model
        model.compile(loss='mse', optimizer='adam', metrics=['mse'])
        # model.compile(loss='mean_squared_error', optimizer='adam')
        return model

    estimator = KerasRegressor(build_fn=baseline_model, verbose=1, epochs=10, batch_size = 55000)
    estimator.fit(train_x, train_y)
    pred_test = estimator.predict(test_x)
    preds.append(pred_test)

    slice_end = slice_start + cur_X_test.shape[0]
    X_all_level2[ slice_start : slice_end , :] = np.c_[preds].transpose()
    slice_start = slice_end
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-
packages\sklearn\linear_model\stochastic_gradient.py:128: FutureWarning: max_iter and tol
parameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDRegressor'> in 0
.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter
defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

```
Training Model 1: lightgbm
Training Model 2: keras
Epoch 1/10
3746814/3746814 [==============================] - ETA: 1:42 - loss: 1.5894 - mean_squared_error:
1.58 - ETA: 1:02 - loss: 1.6221 - mean_squared_error: 1.62 - ETA: 48s - loss: 1.6603 -
mean_squared_error: 1.6603 - ETA: 41s - loss: 1.6325 - mean_squared_error: 1.632 - ETA: 36s - loss
: 1.6161 - mean_squared_error: 1.616 - ETA: 33s - loss: 1.6064 - mean_squared_error: 1.606 - ETA:
31s - loss: 1.5872 - mean_squared_error: 1.587 - ETA: 29s - loss: 1.5786 - mean_squared_error: 1.5
78 - ETA: 28s - loss: 1.5775 - mean_squared_error: 1.577 - ETA: 26s - loss: 1.5683 -
mean_squared_error: 1.568 - ETA: 25s - loss: 1.5622 - mean_squared_error: 1.562 - ETA: 24s - loss:
1.5534 - mean_squared_error: 1.553 - ETA: 24s - loss: 1.5413 - mean_squared_error: 1.541 - ETA: 23
s - loss: 1.5340 - mean_squared_error: 1.534 - ETA: 22s - loss: 1.5255 - mean_squared_error: 1.525
- ETA: 21s - loss: 1.5213 - mean_squared_error: 1.521 - ETA: 21s - loss: 1.5073 -
mean_squared_error: 1.507 - ETA: 20s - loss: 1.4986 - mean_squared_error: 1.498 - ETA: 20s - loss:
1.4912 - mean_squared_error: 1.491 - ETA: 19s - loss: 1.4800 - mean_squared_error: 1.480 - ETA: 18
s - loss: 1.4713 - mean_squared_error: 1.471 - ETA: 18s - loss: 1.4629 - mean_squared_error: 1.462
- ETA: 17s - loss: 1.4538 - mean_squared_error: 1.453 - ETA: 17s - loss: 1.4449 -
mean_squared_error: 1.444 - ETA: 16s - loss: 1.4365 - mean_squared_error: 1.436 - ETA: 16s - loss:
1.4308 - mean_squared_error: 1.430 - ETA: 16s - loss: 1.4226 - mean_squared_error: 1.422 - ETA: 15
s - loss: 1.4136 - mean_squared_error: 1.413 - ETA: 15s - loss: 1.4046 - mean_squared_error: 1.404
- ETA: 14s - loss: 1.3962 - mean_squared_error: 1.396 - ETA: 14s - loss: 1.3907 -
mean_squared_error: 1.390 - ETA: 13s - loss: 1.3847 - mean_squared_error: 1.384 - ETA: 13s - loss:
1.3783 - mean_squared_error: 1.378 - ETA: 13s - loss: 1.3752 - mean_squared_error: 1.375 - ETA: 12
s - loss: 1.3662 - mean_squared_error: 1.366 - ETA: 12s - loss: 1.3636 - mean_squared_error: 1.363
- ETA: 11s - loss: 1.3565 - mean_squared_error: 1.356 - ETA: 11s - loss: 1.3491 -
mean_squared_error: 1.349 - ETA: 10s - loss: 1.3431 - mean_squared_error: 1.343 - ETA: 10s - loss:
1.3343 - mean_squared_error: 1.334 - ETA: 10s - loss: 1.3286 - mean_squared_error: 1.328 - ETA: 9s
- loss: 1.3230 - mean_squared_error: 1.323 - ETA: 9s - loss: 1.3201 - mean_squared_error: 1.32 - E
TA: 9s - loss: 1.3170 - mean_squared_error: 1.31 - ETA: 8s - loss: 1.3132 - mean_squared_error: 1.
31 - ETA: 8s - loss: 1.3095 - mean_squared_error: 1.30 - ETA: 7s - loss: 1.3041 -
mean_squared_error: 1.30 - ETA: 7s - loss: 1.2997 - mean_squared_error: 1.29 - ETA: 7s - loss: 1.2
950 - mean_squared_error: 1.29 - ETA: 6s - loss: 1.2889 - mean_squared_error: 1.28 - ETA: 6s - los
s: 1.2847 - mean_squared_error: 1.28 - ETA: 5s - loss: 1.2814 - mean_squared_error: 1.28 - ETA: 5s
- loss: 1.2762 - mean_squared_error: 1.27 - ETA: 5s - loss: 1.2717 - mean_squared_error: 1.27 - ET
A: 4s - loss: 1.2674 - mean_squared_error: 1.26 - ETA: 4s - loss: 1.2637 - mean_squared_error: 1.2
6 - ETA: 4s - loss: 1.2592 - mean_squared_error: 1.25 - ETA: 3s - loss: 1.2547 -
mean_squared_error: 1.25 - ETA: 3s - loss: 1.2521 - mean_squared_error: 1.25 - ETA: 2s - loss: 1.2
482 - mean_squared_error: 1.24 - ETA: 2s - loss: 1.2441 - mean_squared_error: 1.24 - ETA: 2s - los
s: 1.2422 - mean_squared_error: 1.24 - ETA: 1s - loss: 1.2388 - mean_squared_error: 1.23 - ETA: 1s
- loss: 1.2350 - mean_squared_error: 1.23 - ETA: 1s - loss: 1.2312 - mean_squared_error: 1.23 - ET
A: 0s - loss: 1.2266 - mean_squared_error: 1.22 - ETA: 0s - loss: 1.2230 - mean_squared_error: 1.2
2 - ETA: 0s - loss: 1.2198 - mean_squared_error: 1.21 - 25s 7us/step - loss: 1.2197 -
mean_squared_error: 1.2197
Epoch 2/10
3746814/3746814 [==============================] - ETA: 33s - loss: 1.0734 - mean_squared_error: 1
.073 - ETA: 28s - loss: 1.0517 - mean_squared_error: 1.051 - ETA: 26s - loss: 1.0620 -
mean_squared_error: 1.062 - ETA: 25s - loss: 1.0423 - mean_squared_error: 1.042 - ETA: 24s - loss:
1.0498 - mean_squared_error: 1.049 - ETA: 23s - loss: 1.0363 - mean_squared_error: 1.036 - ETA: 23
s - loss: 1.0272 - mean_squared_error: 1.027 - ETA: 22s - loss: 1.0341 - mean_squared_error: 1.034
- ETA: 22s - loss: 1.0251 - mean_squared_error: 1.025 - ETA: 21s - loss: 1.0218 -
mean_squared_error: 1.021 - ETA: 21s - loss: 1.0127 - mean_squared_error: 1.012 - ETA: 20s - loss:
1.0118 - mean_squared_error: 1.011 - ETA: 20s - loss: 1.0171 - mean_squared_error: 1.017 - ETA: 19
```

```
.995 - ETA: 5s - loss: 0.8498 - mean_squared_error: 0.849 - ETA: 4s - loss: 0.7902 -
mean_squared_error: 0.79 - ETA: 3s - loss: 0.7587 - mean_squared_error: 0.75 - ETA: 3s - loss: 0.7
515 - mean_squared_error: 0.75 - ETA: 3s - loss: 0.7501 - mean_squared_error: 0.75 - ETA: 3s - los
s: 0.7619 - mean_squared_error: 0.76 - ETA: 3s - loss: 0.7615 - mean_squared_error: 0.76 - ETA: 3s
- loss: 0.7592 - mean_squared_error: 0.75 - ETA: 3s - loss: 0.7548 - mean_squared_error: 0.75 - ET
A: 2s - loss: 0.7516 - mean_squared_error: 0.75 - ETA: 2s - loss: 0.7479 - mean_squared_error: 0.7
4 - ETA: 2s - loss: 0.7459 - mean_squared_error: 0.74 - ETA: 2s - loss: 0.7459 -
mean_squared_error: 0.74 - ETA: 2s - loss: 0.7397 - mean_squared_error: 0.73 - ETA: 2s - loss: 0.7
383 - mean_squared_error: 0.73 - ETA: 2s - loss: 0.7375 - mean_squared_error: 0.73 - ETA: 2s - los
s: 0.7351 - mean_squared_error: 0.73 - ETA: 2s - loss: 0.7373 - mean_squared_error: 0.73 - ETA: 2s
- loss: 0.7344 - mean_squared_error: 0.73 - ETA: 2s - loss: 0.7343 - mean_squared_error: 0.73 - ET
A: 2s - loss: 0.7374 - mean_squared_error: 0.73 - ETA: 2s - loss: 0.7400 - mean_squared_error: 0.7
4 - ETA: 2s - loss: 0.7453 - mean_squared_error: 0.74 - ETA: 1s - loss: 0.7474 -
mean_squared_error: 0.74 - ETA: 1s - loss: 0.7496 - mean_squared_error: 0.74 - ETA: 1s - loss: 0.7
514 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7546 - mean_squared_error: 0.75 - ETA: 1s - los
s: 0.7544 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7568 - mean_squared_error: 0.75 - ETA: 1s
- loss: 0.7573 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7586 - mean_squared_error: 0.75 - ET
A: 1s - loss: 0.7579 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7566 - mean_squared_error: 0.7
5 - ETA: 1s - loss: 0.7571 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7552 -
mean_squared_error: 0.75 - ETA: 1s - loss: 0.7564 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7
572 - mean_squared_error: 0.75 - ETA: 1s - loss: 0.7586 - mean_squared_error: 0.75 - ETA: 1s - los
s: 0.7611 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7617 - mean_squared_error: 0.76 - ETA: 0s
- loss: 0.7648 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7641 - mean_squared_error: 0.76 - ET
A: 0s - loss: 0.7641 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7637 - mean_squared_error: 0.7
6 - ETA: 0s - loss: 0.7625 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7618 -
mean_squared_error: 0.76 - ETA: 0s - loss: 0.7636 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7
616 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7636 - mean_squared_error: 0.76 - ETA: 0s - los
s: 0.7629 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7644 - mean_squared_error: 0.76 - ETA: 0s
- loss: 0.7650 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7653 - mean_squared_error: 0.76 - ET
A: 0s - loss: 0.7647 - mean_squared_error: 0.76 - ETA: 0s - loss: 0.7651 - mean_squared_error: 0.7
6 - ETA: 0s - loss: 0.7641 - mean_squared_error: 0.76 - 3s 2us/step - loss: 0.7645 -
mean_squared_error: 0.7645
214200/214200 [==============================] - ETA: 21 - ETA: 0 - ETA:  - 1s 3us/step
1498296/1498296 [==============================] - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  -
ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - ETA:  - 1s 0us/step
Train R-squared for train_preds_NN_stacking is 0.946909
```

4.2.1 Second level learning model via linear regression

In [50]:

```python
from sklearn.linear_model import (LinearRegression, SGDRegressor)

print('Training Second level learning model via linear regression')
lr = LinearRegression()
lr.fit(X_train_level2, y_train_level2)

# Compute R-squared on the train and test sets.
# print('Train R-squared for %s is %f' %('test_preds_lr_stacking',
sqrt(mean_squared_error(y_train_level2, lr.predict(X_train_level2)))))
test_preds_lr_stacking = lr.predict(X_test_level2)
train_preds_lr_stacking = lr.predict(X_train_level2)
print('Train R-squared for %s is %f' %('train_preds_lr_stacking',
sqrt(mean_squared_error(y_train_level2, train_preds_lr_stacking))))
pred_list['test_preds_lr_stacking'] = test_preds_lr_stacking

if Validation:
    print('Test R-squared for %s is %f' %('test_preds_lr_stacking', sqrt(mean_squared_error(y_test_
level2, test_preds_lr_stacking))))
```

```
Training Second level learning model via linear regression
Train R-squared for train_preds_lr_stacking is 0.793253
Test R-squared for test_preds_lr_stacking is 0.919241
```

In [51]:

```python
print('Training Second level learning model via SGDRegressor')
sgdr= SGDRegressor(
    penalty = 'l2' ,
    random_state = SEED )
sgdr.fit(X_train_level2, y_train_level2)

# Compute R-squared on the train and test sets.
```

```
# print('Train R-squared for %s is %f' %('test_preds_lr_stacking', sqrt(mean_squared_error(y_train_level2, lr.predict(X_train_level2)))))
test_preds_sgdr_stacking = sgdr.predict(X_test_level2)
train_preds_sgdr_stacking = sgdr.predict(X_train_level2)
print('Train R-squared for %s is %f' %('train_preds_lr_stacking', sqrt(mean_squared_error(y_train_level2, train_preds_sgdr_stacking))))
pred_list['test_preds_sgdr_stacking'] = test_preds_sgdr_stacking

if Validation:
    print('Test R-squared for %s is %f' %('test_preds_sgdr_stacking', sqrt(mean_squared_error(y_test_level2, test_preds_sgdr_stacking))))
```

```
Training Second level learning model via SGDRegressor
```

```
C:\Users\minori\Anaconda3\envs\tensorflow\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:128: FutureWarning: max_iter and tol parameters have been added in <class 'sklearn.linear_model.stochastic_gradient.SGDRegressor'> in 0.19. If both are left unset, they default to max_iter=5 and tol=None. If tol is not None, max_iter defaults to max_iter=1000. From 0.21, default max_iter will be 1000, and default tol will be 1e-3.
  "and default tol will be 1e-3." % type(self), FutureWarning)
```

```
Train R-squared for train_preds_lr_stacking is 0.793911
Test R-squared for test_preds_sgdr_stacking is 0.917305
```

5. Submission

In [86]:

```
if not Validation:
    submission = pd.read_csv('sample_submission.csv.gz')
    ver = 2
    submission['item_cnt_month'] = pred_list['test_preds_NN_stacking'].clip(0,20)
    submission[['ID', 'item_cnt_month']].to_csv('ver%d.csv' % ( ver), index = False)
```

In [50]:

```
if not Validation:
    submission = pd.read_csv('sample_submission.csv.gz')
    ver = 1
    for pred_ver in ['lr_stacking', 'sgdr_stacking']:
        print(pred_list['test_preds_' + pred_ver].clip(0,20).mean())
        submission['item_cnt_month'] = pred_list['test_preds_' + pred_ver].clip(0,20)
        submission[['ID', 'item_cnt_month']].to_csv('ver%d.csv' % ( ver), index = False)
```

```
0.29133925941839706
0.2741408713025763
```

In [95]:

```
from catboost import CatBoostRegressor
from sklearn import metrics
```

In [96]:

```
#CatBoost
cb_model = CatBoostRegressor(iterations=100, learning_rate=0.2, depth=7, loss_function='RMSE', eval_metric='RMSE', random_seed=18, od_type='Iter', od_wait=20)
cb_model.fit(X_train, y_train, eval_set=(X_test, y_test), use_best_model=True, verbose=False)
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test.clip(0.,20.), cb_model.predict(X_test).clip(0.,20.))))
```

```
RMSE: 1.004505352578051
```

In [97]:

```
#CatBoost
cb_model = CatBoostRegressor(iterations=1000, learning_rate=0.2, depth=7, loss_function='RMSE', eval_metric='RMSE', random_seed=18, od_type='Iter', od_wait=20)
```

```
l_metric='RMSE', random_seed=10, od_type='Iter', od_wait=20)
cb_model.fit(X_train, y_train, eval_set=(X_test, y_test), use_best_model=True, verbose=False)
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test.clip(0.,20.), cb_model.predict(X_test).cli
p(0.,20.))))
```

```
RMSE: 1.004505352578051
```

In [98]:

```
#CatBoost
cb_model = CatBoostRegressor(iterations=5000, learning_rate=0.05, depth=7, loss_function='RMSE', ev
al_metric='RMSE', random_seed=18, od_type='Iter', od_wait=20)
cb_model.fit(X_train, y_train, eval_set=(X_test, y_test), use_best_model=True, verbose=False)
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test.clip(0.,20.), cb_model.predict(X_test).cli
p(0.,20.))))
```

```
RMSE: 1.0027329909297533
```

In [91]:

```
all_data.head(10)
```

Out[91]:

	shop_id	item_id	date_block_num	target	target_shop	target_item	target_lag_1	target_item_lag_1	target_shop_lag_1
0	54	10297	12	4.0	8198.0	23.0	3.0	42.0	10055.0
1	54	10296	12	3.0	8198.0	17.0	0.0	24.0	10055.0
2	54	10298	12	14.0	8198.0	182.0	21.0	369.0	10055.0
3	54	10300	12	3.0	8198.0	26.0	1.0	54.0	10055.0
4	54	10284	12	1.0	8198.0	3.0	0.0	4.0	10055.0
5	54	10292	12	9.0	8198.0	93.0	8.0	156.0	10055.0
6	54	10109	12	2.0	8198.0	17.0	1.0	19.0	10055.0
7	54	10107	12	1.0	8198.0	26.0	2.0	23.0	10055.0
8	54	10121	12	1.0	8198.0	1.0	0.0	0.0	0.0
9	54	10143	12	1.0	8198.0	12.0	1.0	18.0	10055.0