2.Data Preprocessing

1. We drop irrelevant or less important features
2. We need to process datetime, from daily-->monthly

\+ Code + Markdown

```
#Select useful features
train_monthly = lk_train[['date', 'date_block_num', 'shop_id', 'item_category_id', 'item_id', 'item_price', 'item_
train_monthly.head()
```

	date	date_block_num	shop_id	item_category_id	item_id	item_price	item_cnt_day
0	2013-01-02	0	59	37	22154	999.0	1.0
10	2013-01-03	0	25	55	2574	399.0	2.0
11	2013-01-05	0	25	55	2574	399.0	1.0
12	2013-01-07	0	25	55	2574	399.0	1.0
13	2013-01-08	0	25	55	2574	399.0	2.0

]:

```
# Group by month in this case "date_block_num" and aggregate features.
train_monthly = train_monthly.sort_values(['date']).groupby(['date_block_num', 'shop_id', 'item_category_id', 'ite
train_monthly = train_monthly.agg({'item_price':['sum', 'mean'], 'item_cnt_day':['sum', 'mean','count']})
```

]:

```
# Rename features.
train_monthly.columns = ['date_block_num', 'shop_id', 'item_category_id', 'item_id', 'item_price', 'mean_item_pric
```

\+ Code + Markdown

We're using empty_df, we need a dataframe that will have combinations of months, shop_id, and item_id. First create an empty dataframe, then iterate over the existing records to fill it, and finally fill the missing values with 0. We're taking all possible combinations here since we have to tell the model that for those months the item count for a particular shop ID/item ID was zero instead of having any missing records.

```
shop_ids = train_monthly['shop_id'].unique()
item_ids = train_monthly['item_id'].unique()
empty_df = []
for i in range(34):
    for shop in shop_ids:
        for item in item_ids:
            empty_df.append([i, shop, item])
empty_df = pd.DataFrame(empty_df, columns=['date_block_num','shop_id','item_id'])
```

```
#Merge the train set with the complete set (missing records will be filled with 0).
train_monthly = pd.merge(empty_df, train_monthly, on=['date_block_num','shop_id','item_id'], how='left')
train_monthly.fillna(0, inplace=True)
```

```
train_monthly.head()
```

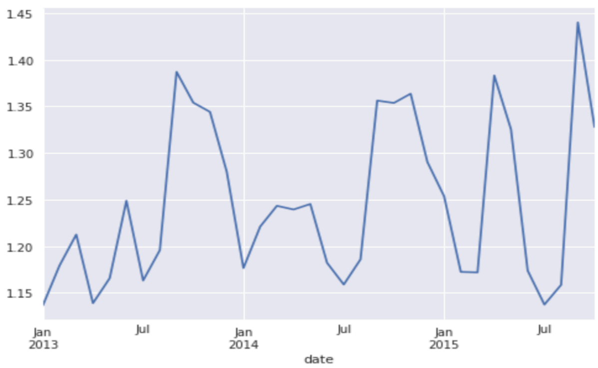
	date_block_num	shop_id	item_id	item_category_id	item_price	mean_item_price	item_cnt	mean_item_cnt	transactions
count	6.734448e+06	6.734448e+06	6.734448e+06	6.734448e+06	6.734448e+06	6.734448e+06	6.734448e+06	6.734448e+06	6.734448e+06
mean	1.650000e+01	3.164286e+01	1.104189e+04	3.786271e+00	1.873922e+02	8.123012e+01	2.402225e-01	9.729913e-02	1.818165e-01
std	9.810709e+00	1.756189e+01	6.210744e+03	1.321296e+01	2.177442e+03	5.347327e+02	3.456639e+00	6.122031e-01	9.047315e-01
min	0.000000e+00	2.000000e+00	3.000000e+01	0.000000e+00	0.000000e+00	0.000000e+00	-4.000000e+00	-2.000000e+00	0.000000e+00
25%	8.000000e+00	1.600000e+01	5.385250e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	1.650000e+01	3.450000e+01	1.126550e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
75%	2.500000e+01	4.700000e+01	1.606825e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
max	3.300000e+01	5.900000e+01	2.216700e+04	8.300000e+01	5.155736e+05	4.299000e+04	2.253000e+03	1.000000e+03	3.100000e+01

+ Code + Markdown

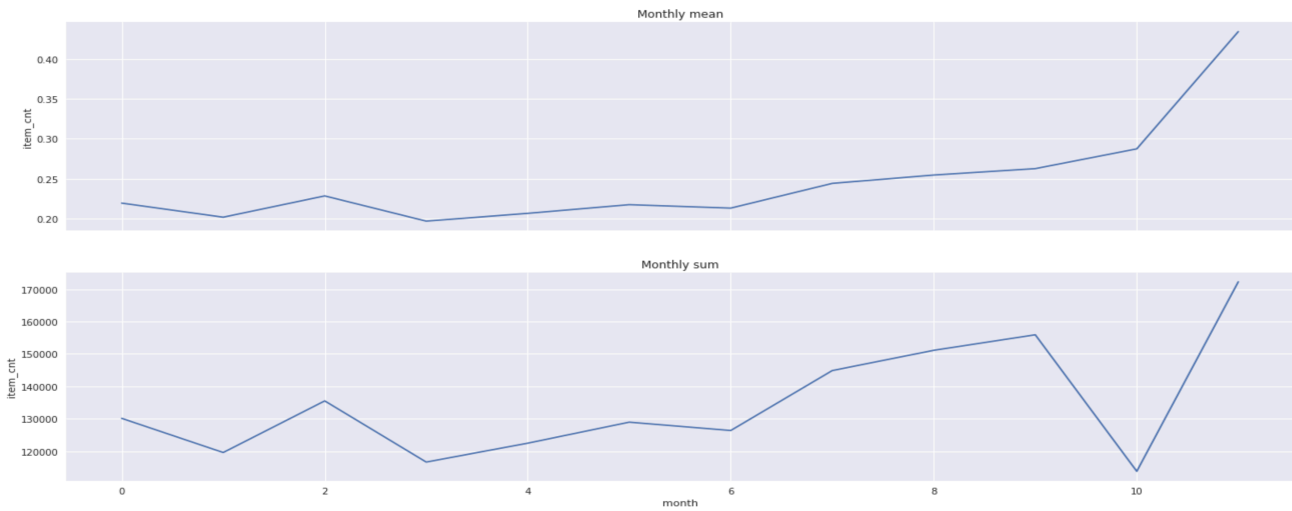
```
# Extract time based features, this will be essential when we group the data based on month/year.
train_monthly['year'] = train_monthly['date_block_num'].apply(lambda x: ((x//12) + 2013))
train_monthly['month'] = train_monthly['date_block_num'].apply(lambda x: (x % 12))
```

```
#date_block_num covers a window of 34 months and takes values from 0 to 33, so to extract some 'monthly' features
gp_month_mean = train_monthly.groupby(['month'], as_index=False)['item_cnt'].mean()
gp_month_sum = train_monthly.groupby(['month'], as_index=False)['item_cnt'].sum()
gp_category_mean = train_monthly.groupby(['item_category_id'], as_index=False)['item_cnt'].mean()
gp_category_sum = train_monthly.groupby(['item_category_id'], as_index=False)['item_cnt'].sum()
gp_shop_mean = train_monthly.groupby(['shop_id'], as_index=False)['item_cnt'].mean()
gp_shop_sum = train_monthly.groupby(['shop_id'], as_index=False)['item_cnt'].sum()
```

```
plt.style.use('seaborn')
train.copy().set_index('date').item_cnt_day.resample('M').mean().plot()
```




```
f, axes = plt.subplots(2, 1, figsize=(22, 10), sharex=True)
sns.lineplot(x="month", y="item_cnt", data=gp_month_mean, ax=axes[0]).set_title("Monthly mean")
sns.lineplot(x="month", y="item_cnt", data=gp_month_sum, ax=axes[1]).set_title("Monthly sum")
plt.show()
```



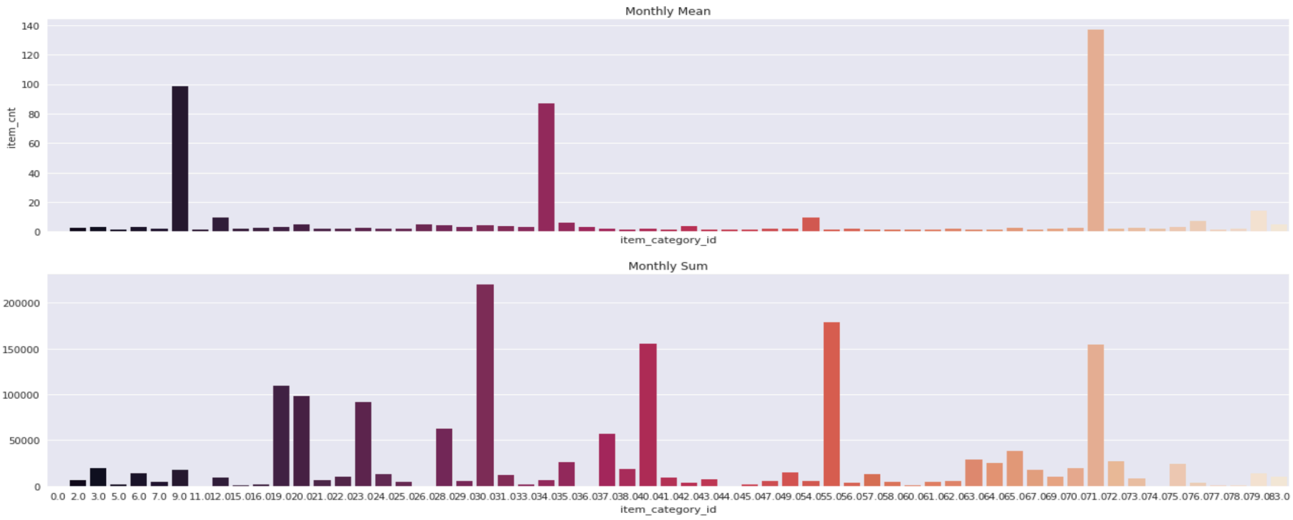
Summary

1. We have a trending increase of item sales count (mean) towards the ending of the year.
2. There are peaks in October, then dips in November, followed by an increase in December and similar item count zig-zag behavior can be seen in June-July-August. This can be due to: vacation time/national holidays.

Item Category Sales Viz

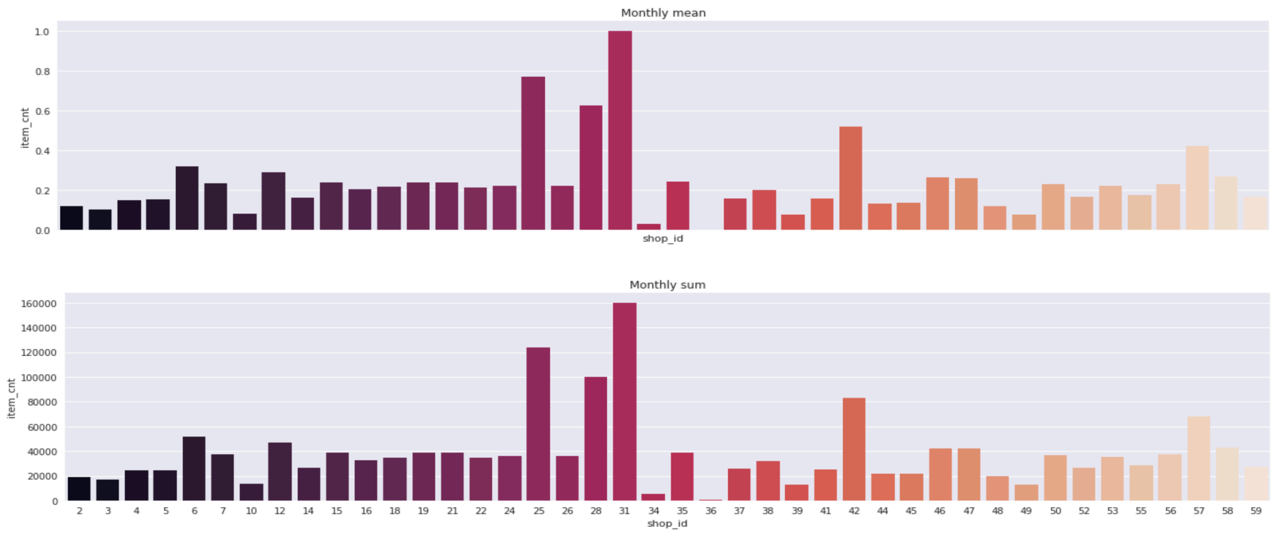
+ Code + Markdown

```
fig, axes = plt.subplots(2, 1, figsize=(22, 10), sharex=True)
sns.barplot(x="item_category_id", y="item_cnt", data=gp_category_mean, ax=axes[0], palette="rocket").set_title("Monthly Mean")
sns.barplot(x="item_category_id", y="item_cnt", data=gp_category_sum, ax=axes[1], palette="rocket").set_title("Monthly Sum")
plt.show()
```



Shops Sales Viz

```
fig, axes = plt.subplots(2, 1, figsize=(22, 10), sharex=True)
sns.barplot(x="shop_id", y="item_cnt", data=gp_shop_mean, ax=axes[0], palette="rocket").set_title("Monthly mean")
sns.barplot(x="shop_id", y="item_cnt", data=gp_shop_sum, ax=axes[1], palette="rocket").set_title("Monthly sum")
plt.show()
```



+ Code + Markdown

Conclusion: Most of the shops have a similar sell rate, but 3 of them have a much higher rate, because of their shop size.

Outlier Identification

```python
plt.figure(figsize=(15,4))
plt.xlim(-100, 3000)
sns.boxplot(x=train['item_cnt_day'])
print('Item Sale outliers:',train['item_id'][train['item_cnt_day']>500].unique())

plt.figure(figsize=(15,4))
plt.xlim(train['item_price'].min(), train['item_price'].max())
sns.boxplot(x=train['item_price'])
print('Item price outliers:',train['item_id'][train['item_price']>50000].unique())
```

```
Item Sale outliers: [ 8057 20949  9242 19437  3731 11373  9249  9248]
Item price outliers: [11365  6066 13199]
```


```python
#drop the outliers
train_monthly = train_monthly.query('item_cnt >= 0 and item_cnt <= 500 and item_price < 50000')
train_monthly
```

	date_block_num	shop_id	item_id	item_category_id	item_price	mean_item_price	item_cnt	mean_item_cnt	transactions	year	month
0	0	2	5572	2.0	10730.00	1532.857143	9.0	1.285714	7.0	2013	0
1	0	2	5643	2.0	4775.21	2387.605000	0.0	0.000000	2.0	2013	0
2	0	2	5583	5.0	1188.30	594.150000	2.0	1.000000	2.0	2013	0
3	0	2	7893	6.0	5970.00	1990.000000	3.0	1.000000	3.0	2013	0
4	0	2	7894	6.0	1490.00	1490.000000	1.0	1.000000	1.0	2013	0
...	...	...	...	...	...	...	...	...	...	...	...
6734443	33	36	9103	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9
6734444	33	36	9107	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9
6734445	33	36	5704	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9
6734446	33	36	12733	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9
6734447	33	36	15925	0.0	0.00	0.000000	0.0	0.000000	0.0	2015	9

6732967 rows × 11 columns