Predicting Future Sales

Cedric Yu[♮]

Abstract

We summarise our findings on the (expired) Kaggle competition. In this project, we work with a time-series dataset consisting of daily sales data of one of the largest Russian software firms— 1C Company. We are tasked with predicting the total sales for every product and store in a coming month.

1 Problem Statement

In this competition, we are given the sales records of the items sold in the shops of IC Company, during the time period January 2013 – October 2015. We are tasked with predicting the total sales *by number* for every product and store in the coming month, i.e. November 2015. Submissions are evaluated by the root-mean-squared error (RMSE), and the true target values are clipped into the $[0, 20]$ range.

1.1 Datasets: A Quick Overview

We are given a few datasets: `shops.csv`, `items.csv` and `item_categories`, as well as `sales_train.csv` and `test.csv`. We first introduce the former three. Since this is a Russian company, data are stored in Russian, not English. We make use of Google Translate when necessary.

1.1.1 Shops, Items and Item Categories

The data file `shops.csv` contains the `shop_name` of each of the 60 shops, and an assignment of a unique `shop_id` to each of them.

`items.csv` similarly consists of the `item_name` of 22,170 items, and the assignment of an `item_id`, and an `item_category_id`. `item_categories` maps each of the 84 `item_category_id` to its `item_category_name`.

1.1.2 Training and Test Sets

In the training set, named `sales_train.csv`, we are given 2,935,849 instances of daily sales (by number) records of each item in each shop, from January 2013 to October 2014. It contains no missing values. The feature columns are `date`, `date_block_num`, `shop_id`, `item_id`, `item_price`, and `item_cnt_day`. `date_block_num` is a number that counts the number of months since 2013-01; it is 0 for 2013-01, 1 for 2013-02, 2 for 2013-03, etc. The set contains 60 unique `shop_id` and 21,807 unique `item_id`. `item_price` is given in ruble. `item_cnt_day` is the daily sales count of that item at that particular shop. This is not exactly the target; we want to predict *monthly* sales.

The test set contains simply 214,200 pairs of (`shop_id`, `item_id`), each assigned to an `ID`. In particular, the `item_price` feature is not provided.

[♮]cedric.yu@nyu.edu. Last updated: September 17, 2021.

1.2 A Time-Series Problem?

While this can be regarded as a time series problem of the 214, 200 pairs of (`shop_id`, `item_id`), the large number of pairs could make training the individual time series models impractical to achieve, not to mention we should also take into account interactions between the series, e.g. Granger “causality". As such, when it comes to selecting machine learning models, we will employ the standard tools, such as tree-based algorithms and deep neural networks. However, we will perform feature engineering in a time-conscious manner. This includes the engineering and use of lag features, as well as performing train-validation split in a time-ordered way.

2 Exploratory Data Analysis

We make plots in `dataset_study.py`.

2.1 Shops, Items and Item Categories

We investigate the supplementary data files `shops.csv`, `items.csv` and `item_categories` in `about_shops_items.py`. There we also extract some features from the shop and item names.

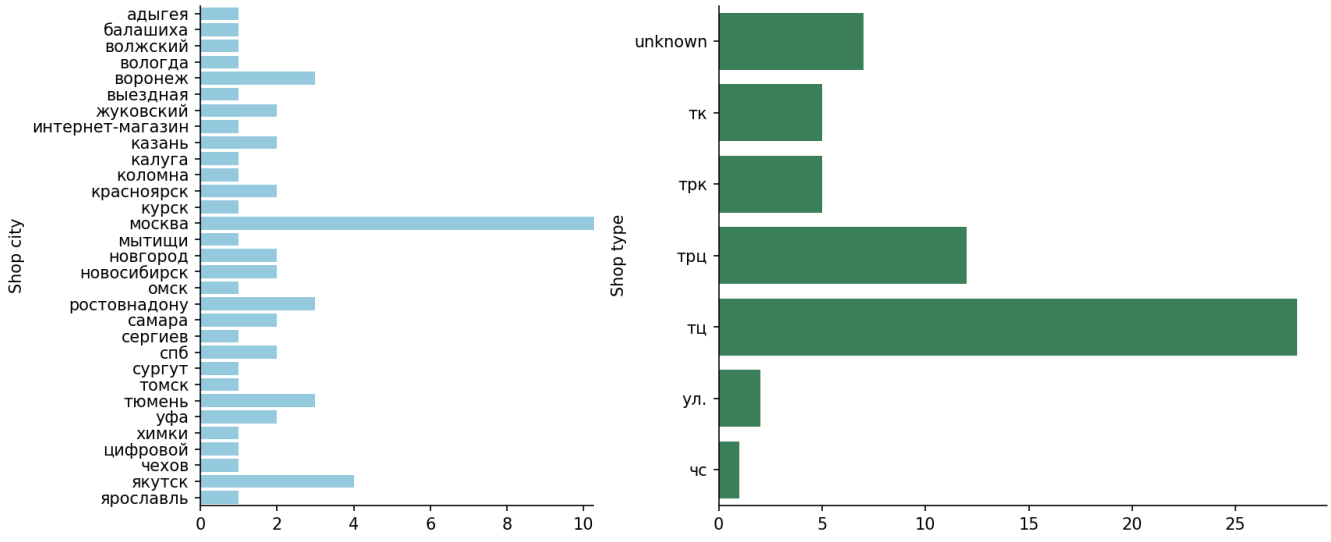
We first look at the shops. By a quick inspection of the `shop_name`, with the help of Google Translate, we find a few pairs of shops whose names are almost identical; each such pair should correspond to the same shop. Using `fuzzywuzzy.fuzz`, we construct a pairwise fuzzy match matrix on the shop names using `token_sort_ratio` as the metric, and pick out the pairs that score higher than 80 as possible duplicate pairs. They are:
`[(10, 11), (23, 24), (30, 31), (39, 40), (0, 57), (1, 58)]`. But not all of them are actual duplicate pairs. For example, `(30, 31)` are located in different places, as their names suggest, and `(23, 24)` are different stores. Looking at their monthly sales, as we will do later, also gives extra hints. In the end, we arrive at the true duplicate pairs: `[(10, 11), (57, 0), (58, 1)]`. The first, but not the second, `shop_id` of each tuple appears in the test set.

The `shop_name` contains information about each shop in a systematic way: the first word is the city, followed by the shop type. The city names are translated to [1]

```
['adygea', 'balashikha', 'Volzhsky', 'Vologda', 'voronezh',
'exit', 'zhukovsky', 'online store', 'kazan', 'kaluga', 'Kolomna',
'Krasnoyarsk', 'Kursk', 'Moscow', 'mytischi', 'novgorod',
'novosibirsk', 'Omsk', 'Rostovnadon', 'samara', 'sergiev',
'spb', 'surgut', 'tomsk', 'tyumen', 'ufa', 'khimki',
'digital', 'chekhov', 'yakutsk', 'Yaroslavl']
```

Not all of them make sense, but I would take that. We store the new features in `items_cat_platform.csv` for future use. The number of shops in each city and type are plotted in Figure 1.

[1] I have not figured out how to type Russian in LaTeX.



(a) City

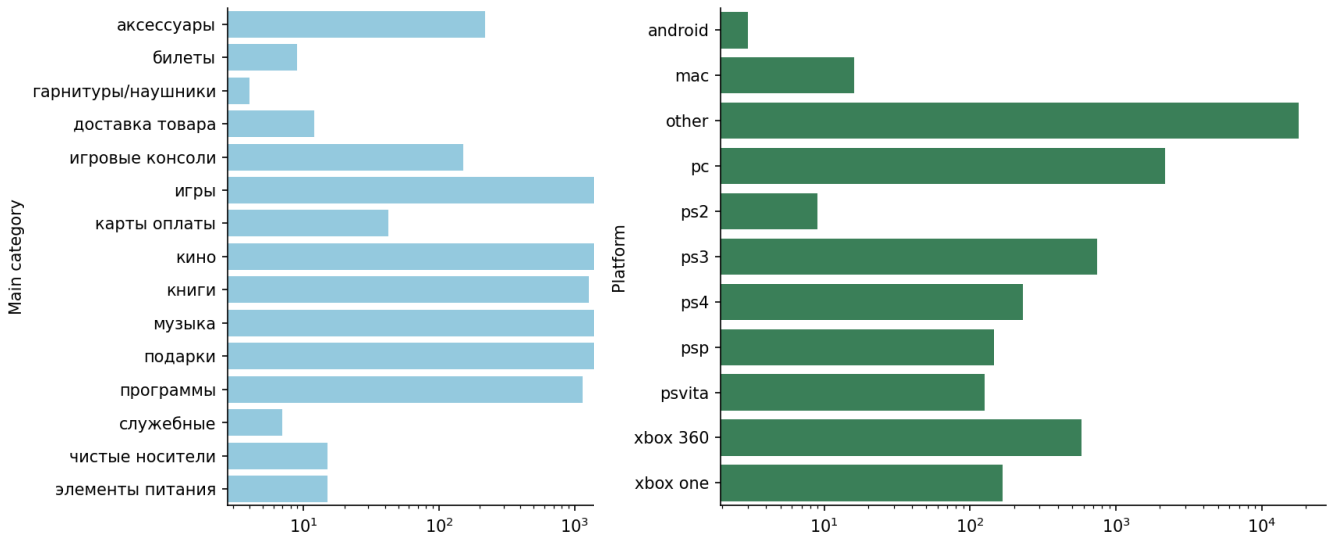
(b) Shop type

Figure 1: Number of shops in each city and type.

Most of the shops are located in Moscow, and the shops mostly belong to shopping centres.

Next, we turn to the items and categories. From the `item_category_name`, we can extract, using `regex`, features such as the `item_main_category` and `platform` (android, mac, etc.) if applicable. The `item_main_category` are

```
['accessories', 'tickets', 'headsets / headphones', 'delivery of goods',
 'game consoles', 'games', 'payment cards', 'cinema', 'books', 'music',
 'present', 'programs', 'service', 'blank media', 'batteries']
```



(a) Main category

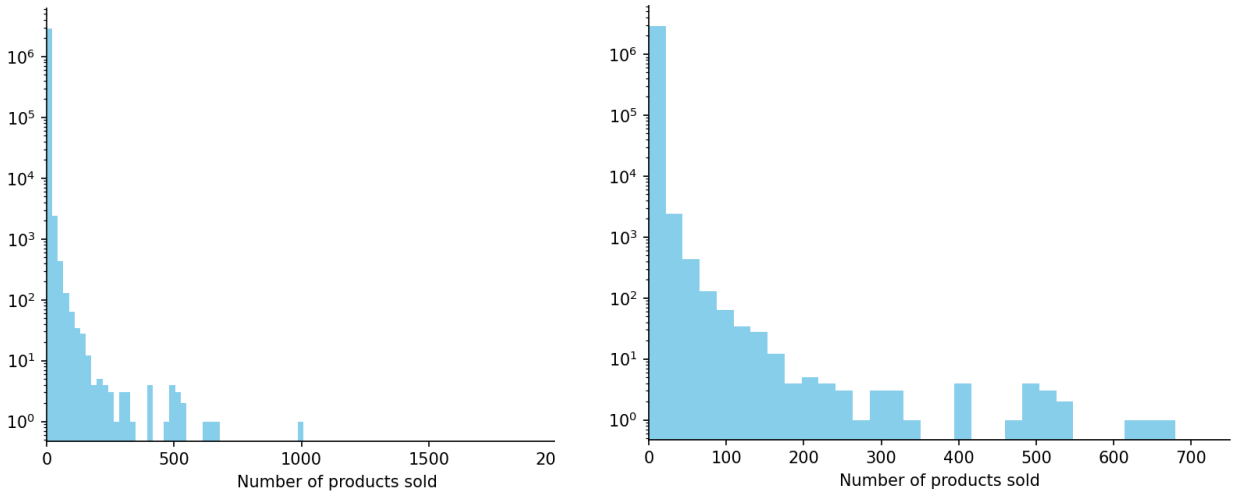
(b) Platform

Figure 2: Number of items in each main category and platform, in log scale.

We store the new features in `item_cat_count_in_platform.csv` for future use. The number of items in each `item_main_category` and `platform`, in log scale, are plotted in Figure 2.

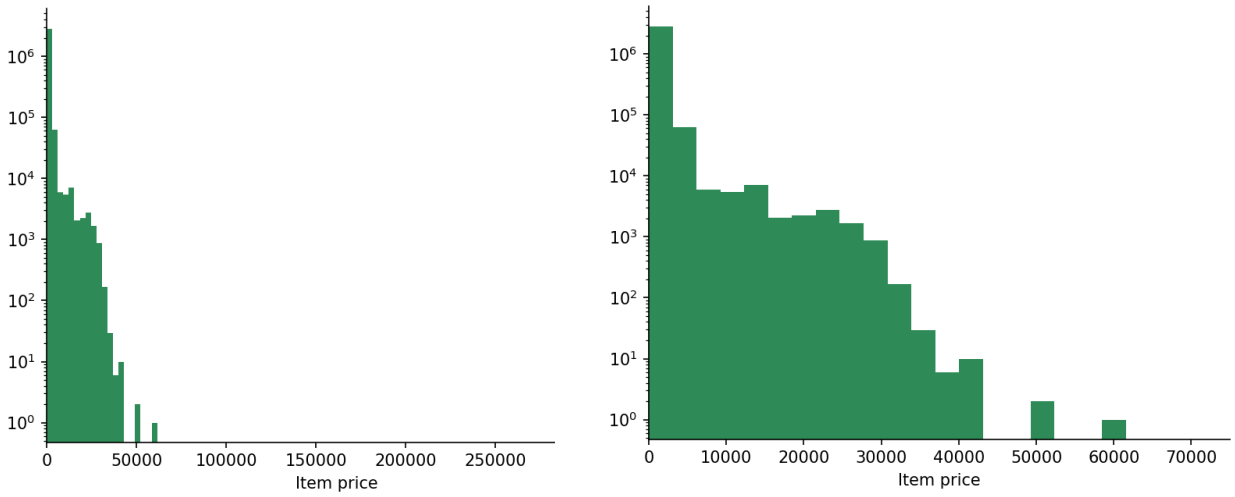
2.2 Outliers

We now investigate the training set. First, we inspect the daily item sales count `item_cnt_day` and `item_price`; see Figure 3 and Figure 4.



(a) Full range (b) Truncated

Figure 3: Histograms of daily item sales count, in log scale.



(a) Full range (b) Truncated

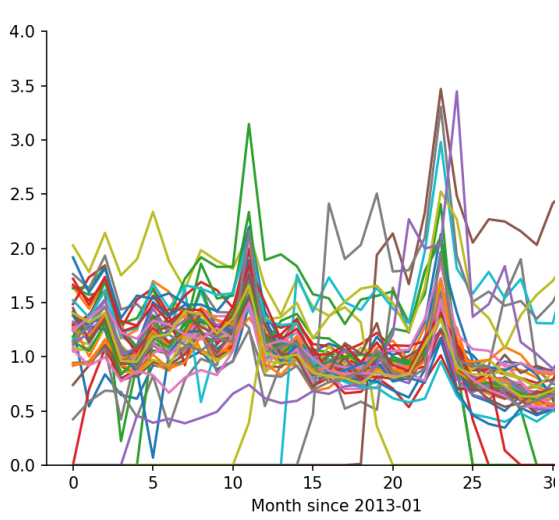
Figure 4: Histograms of item price, in log scale.

Informed by the histograms, we hereafter only keep instances with $0 < \texttt{item\_cnt\_day} < 750$, and $0.01 < \texttt{item\_price} < 75000$. The latter is in ruble; to give some perspective, a PS5

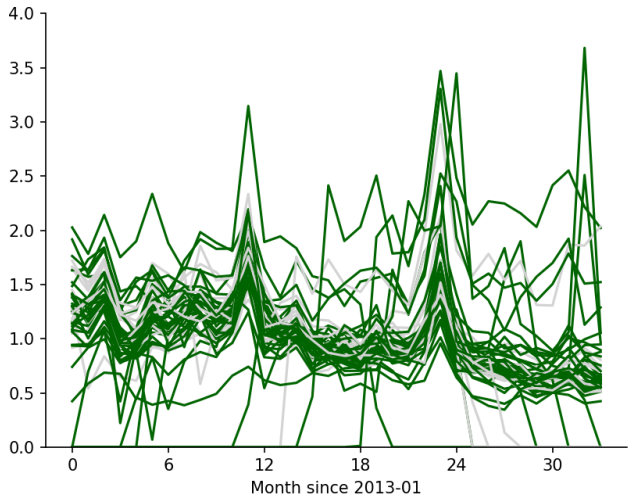
costs about 47, 000 rubles at the time of writing. Moreover, we merge duplicate `shop_id` pairs as discussed in the last subsection.

2.2.1 Sales Count by Shop

For each `shop_id`, we aggregate the total number of sales of *all* items in each month. The resulting monthly sales for each shop in the training set duration, rescaled by the respectively means of each shop, is given in Figure 5a. We have discarded those with low sales counts to reduce noise in the plot. From this, we found that some shops are no longer open by the end of the training set period; we define “open" by having any sales in at least 3 months during the period *and* in the last month. In Figure 5b we highlight those that are open in green.



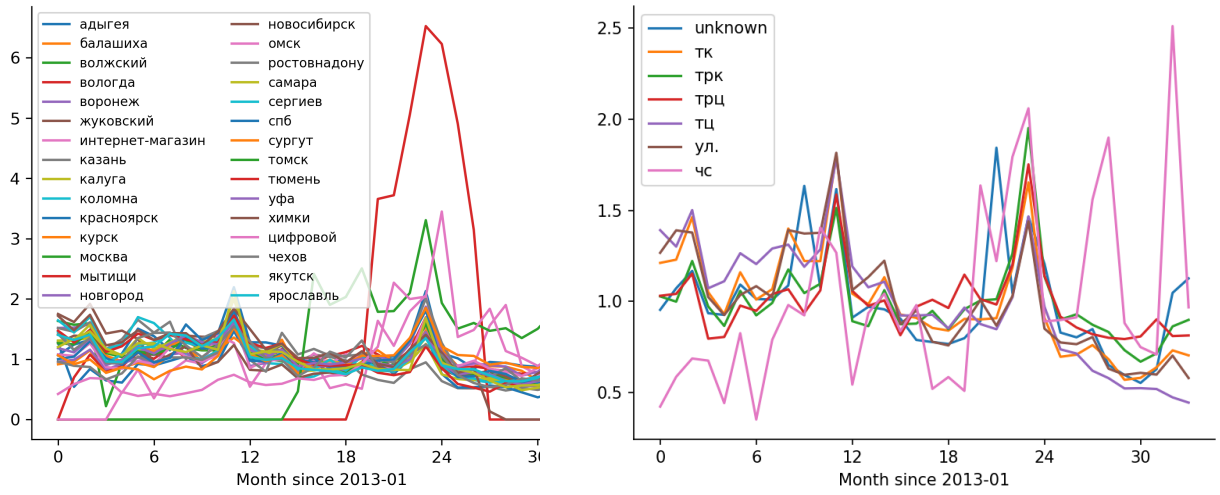
(a) All shops



(b) Green(grey): shops that are still open (closed) by the end of training set period.

Figure 5: Monthly total item sales of each shop, scaled by the respective means.

Next, we plot monthly sales of shops grouped by city and shop type respectively, in Figure 6.



(a) By City

(b) By shop type

Figure 6: Monthly total item sales grouped by city and shop type, scaled by the respective means.

We also plot the total monthly sales of all shops combined, in Figure 7.

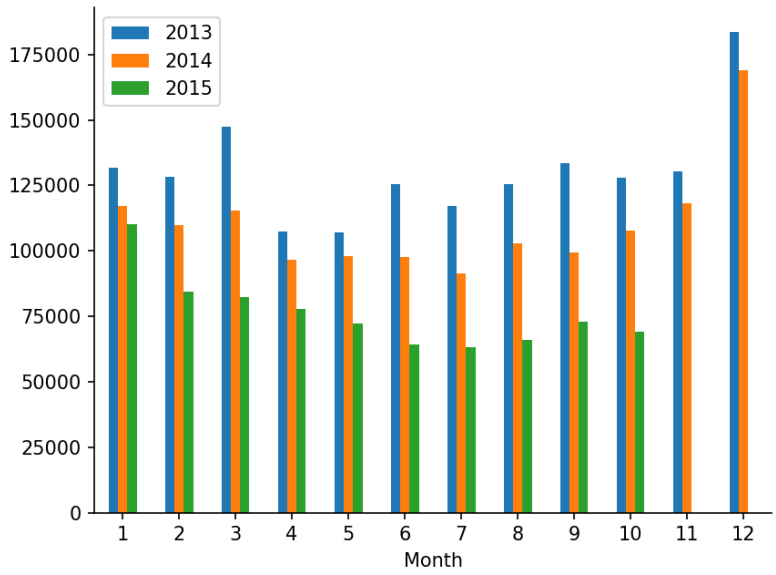


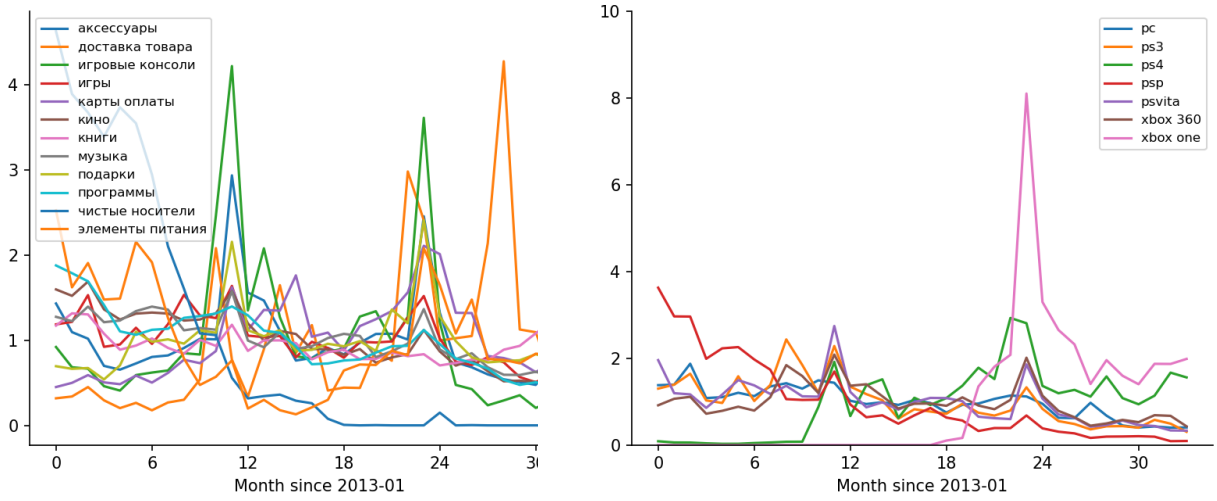
Figure 7: Total monthly sales of all shops and items combined.

Figure 5 and Figure 6 look somewhat messy, with high fluctuations. But in both Figure 5, Figure 6 and Figure 7, we see the lines in the respective plots follow roughly similar patterns modulo fluctuations. We also see seasonality: higher sales during November and December (holiday seasons), as well as generally declining sales over the years.

In the Figure 14 and Figure 15, we also included the auto-correlation and partial auto-correlation plots of the monthly sales of each shop. Roughly, we see that most of the shops have large partial auto-correlations at the first 2-3 month lags, as well as at 12-14 months. This suggests that we engineer lag features in monthly sales count with these lags.

2.2.2 Sales Count by Item

For each `item_id`, we aggregate the total number of sales in *all* shops in each month. We furthermore respectively group them into the `item_main_category` and `platform`. The resulting monthly sales plots in the training set duration, rescaled by the respectively means, are given in Figure 8. We have discarded those with low sales counts to reduce noise in the plots.



(a) Main category

(b) Platform.

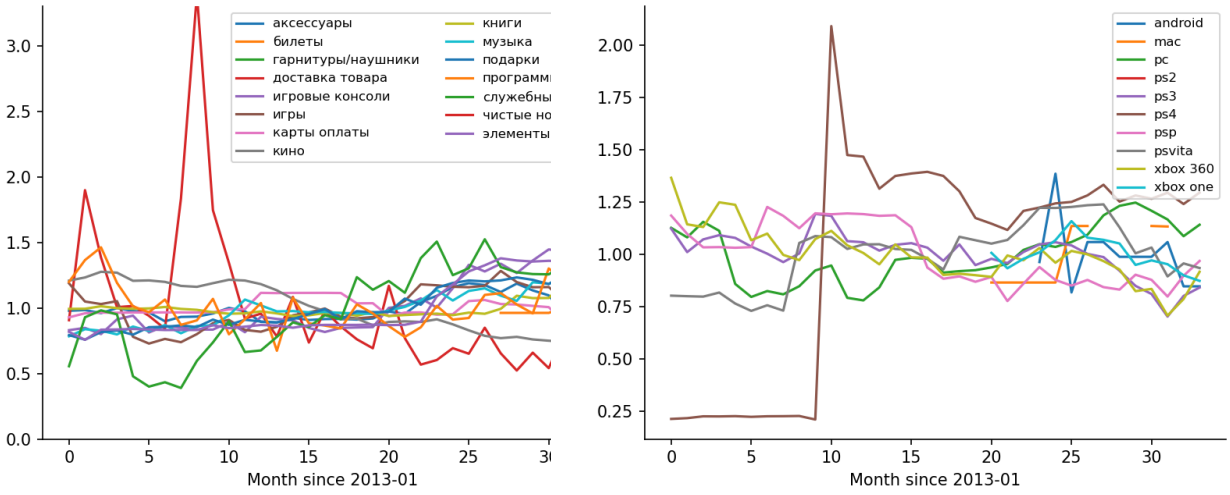
Figure 8: Monthly total item sales of items in each main category and platform, scaled by the respective means.

The plots in Figure 8 appear quite messy, but we do again see some general trends: peaks at the end of each year, and generally declining sales over time.

In the Figure 16 and Figure 17, we made the auto-correlation and partial auto-correlation plots of the monthly sales of each `item_main_category`. We found that some of them have significant partial auto-correlations at the first 1 month lag, as well as at 11-12 months, similar to the case of grouping by shop. This suggests that we engineer lag features in monthly sales count with these lags.

2.2.3 Item Prices

We now study the item prices `item_price`, which were only provided in the training set. We only consider the items that were still sold for at least three months, and in last three months of the training period. The mean prices by `item_main_category` and `platform` are plotted in Figure 9.



(a) Main category

(b) Platform.

Figure 9: Monthly mean item prices of each main category and platform, scaled by the respective means.

In Figure 9a, we find that the mean prices were mostly close to their respectively means, except for the main category “delivery of goods". I do not know exactly what it stands for, but I guess it refers to the delivery fees of sending the goods to customers’ addresses. Thus it makes sense that it fluctuates more. The same can be said to Figure 9b, except for the platform `ps4`. PS4 was launched in November 2013, so as expected we see a spike in prices on that platform at launch, which them gradually fall down and approach the mean value.

In the Figure 18 and Figure 19, we made the auto-correlation and partial auto-correlation plots of the monthly mean price of each `item_main_category`. Once again, we found that most of them have significant partial auto-correlations at the first 1 month lag, and some at 11-12 months too. This suggests that we engineer lag features in monthly sales count with these lags.

2.3 Summary of Preliminary Findings

Let us summarise what we gathered so far. We can group the shops by city and shop type, and items by main category and platform. We dropped outliers in daily sales number and item price, and cleaned up the duplicate `shop_id`. After that, we found that some shops appeared to have ceased operations by the end of the training period. In terms of both the shops and items, the monthly sales numbers follow a rough trend: peaks at the end of the year, and generally declining sales over the years. The mean prices of items, on the other hand, are mostly roughly constant. In both monthly sales count and mean price, we found significant partial auto-correlations at 1 month lag, as well as at around 12 month lag.

3 Feature Pre-processing and Engineering

Informed by the findings in the exploratory data analysis, we now describe the feature pre-processing and engineering procedure, implemented in `preprocessing.py` and `preprocessing_utils.py`. Most steps involve just the use of standard methods in pandas.

3.1 Workflow

We first describe our workflow:

1. Load training and test datasets
2. Drop outliers: keep instances with $0 <$ `item_cnt_day` < 750, and $0.01 <$ `item_price` < 75000
3. Merge duplicate `shop_id`
4. Aggregate monthly sales count into `item_cnt_month` and mean item prices `item_price_mean` in training set; the former is our target
5. Add `month` and `year` columns to both training and test sets
6. Concatenate training and test sets into a single dataframe
7. Append features `shop_city`, `shop_type`, `item_main_category`, `platform`
8. Create 1-4, 12-14-month lag features for `item_cnt_month` and `item_price_mean`
9. Time-ordered train-validation split: train set spans March 2014 – July 2015, and validation set spans Aug – Oct 2015
10. Find and only keep shops in the train set that are still open (open for at least 3 months, open last month), and items that are still sold (sold for at least 3 months, and sold in last 3 months)
11. Encode categorical features, and fill missing values by the means
12. Select features
13. Apply `MinMaxScaler`

3.2 Creating Lag Features

To create lag features, we shift the provided `date_block_num` feature, which counts the month since January 2013, once each time. In `preprocessing_util.py`, we implement the following function

```
def lags(df, col, n=3) :
    dfo = df.copy()

    for i in np.arange(1, n + 1) :
        df_lag = dfo[['shop_id', 'item_id', 'date_block_num', col]].copy()
        df_lag['date_block_num'] = df_lag['date_block_num'] + 1
        df_lag.rename(columns = {col: col + '_lag_' + str(i)}, inplace = True)
        dfo = pd.merge(dfo, df_lag, how = 'left', on = ['shop_id', 'item_id',
        'date_block_num'])
        dfo[col + '_lag_' + str(i)].fillna(0., inplace = True)

    return dfo
```

which returns a new dataframe with lag features of consecutive months, up to the given parameter *n*.

We experimented with how many lag features to include for `item_cnt_month` and `item_price_mean`. Initially we used only 1-month and 12-month lags, informed by our exploratory data analysis. But we got better results by including 1-3 month and 12-14 month lags features.

3.3 Encoding Categorical Variables

We use frequency encoding for all the categorical features, which are

```
['first_open_month_num', 'month', 'year',
'shop_city', 'shop_type', 'item_main_category', 'platform']
```

(Read the last section to see what these features mean.)

3.4 Feature Selection

We use the following features:

```
['item_cnt_month_lag_1', 'item_cnt_month_lag_2', 'item_cnt_month_lag_3',
       'item_cnt_month_lag_4', 'item_cnt_month_lag_12',
       'item_cnt_month_lag_13', 'item_cnt_month_lag_14',
       'item_price_mean_lag_1', 'item_price_mean_lag_2',
       'item_price_mean_lag_3', 'item_price_mean_lag_4',
       'item_price_mean_lag_12', 'item_price_mean_lag_13',
       'item_price_mean_lag_14', 'first_open_month_num_freq_encoded',
       'month_freq_encoded', 'year_freq_encoded', 'shop_city_freq_encoded',
       'shop_type_freq_encoded', 'item_main_category_freq_encoded',
       'platform_freq_encoded']
```

4 Models

We performed modeling training in `master.py`. We use `xgboost.XGBRegressor`, `sklearn.ensemble.RandomForestRegressor`, and `lightgbm.LGBMRegressor`. Past experience tells us that deep neural networks do not generally perform better than these tree-based models, so we will not use them this time. We train the models with the objective of minimising the RMSE. We use `RandomizedSearchCV` to find the optimal hyperparameters.

For `xgboost.XGBRegressor`, we use the following set of optimal hyperparameters:

```
XGBR_model = XGBRegressor(eval_metric = "rmse",
                          learning_rate = 0.05,
                          max_depth = 8,
                          n_estimators = 650,
                          reg_lambda = 0.9,
                          n_jobs = 6)
```

together with `early_stopping_round=50`.

For `RandomForestRegressor`, we used

```
RFR = RandomForestRegressor(n_estimators = 1000,
                            max_depth = 60,
                            max_features= 'sqrt',
                            min_samples_leaf = 4,
                            min_samples_split = 128,
                            random_state = 0, n_jobs = -1)
```

For `LGBMRegressor`,

```
LGBMreg = lgb.LGBMRegressor(boosting_type = 'gbdt',
                            learning_rate = 0.02,
                            num_leaves = 800,
                            n_estimators = 1000,
                            num_iterations = 5000,
                            max_bin = 500,
                            feature_fraction = 0.7,
                            bagging_fraction = 0.7,
                            lambda_l2 = 0.5,
                            max_depth = 25,
                            silent = False)
```

with `early_stopping_rounds=50`.

5 Results

Our `XGBRegressor` gives RMSE of 1.88953 on the training set, and 1.99079 on the validation set. Our `RandomForestRegressor` gives RMSE of 1.94239 on the training set, and 1.97356 on the validation set. Our `LGBMRegressor` gives RMSE of 1.83872 on the training set, and 1.96716 on the validation set.

Judging by the validation RMSE, it appears that `LGBMRegressor` performs the best. We simply make predictions from all three models, submit to Kaggle and see which one does the best on the test set.

In the end, the public scores (RMSE) of the models are respectively 1.67870, 1.72399 and 1.73834. So it turns out it was `XGBRegressor` that gave the best score.

Finally, we look at the feature importances of the best performing models, `XGBRegressor`, `RandomForestRegressor` and `LGBMRegressor`.

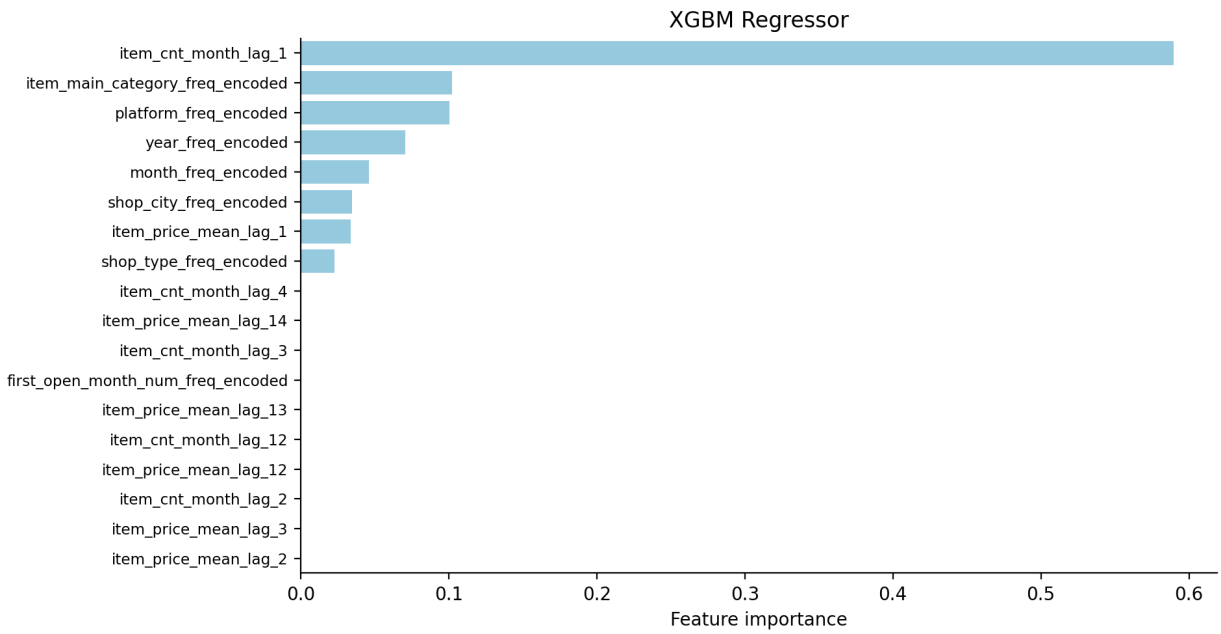


Figure 10: Feature importances in `XGBRegressor`.

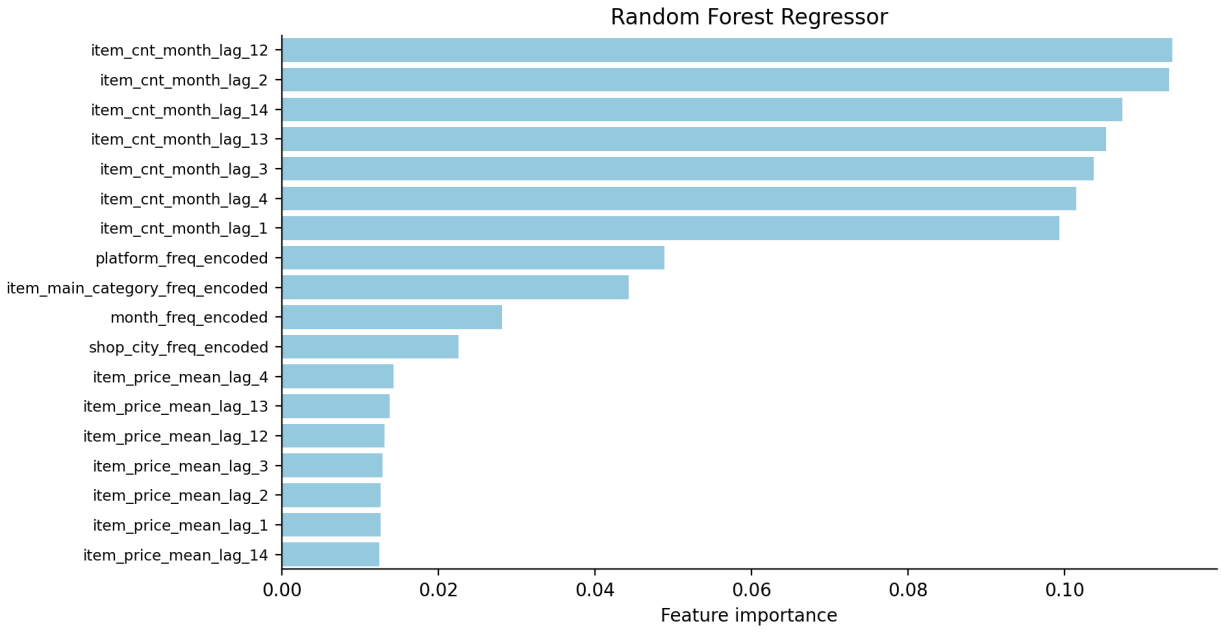


Figure 11: Feature importances in `RandomForestRegressor`.

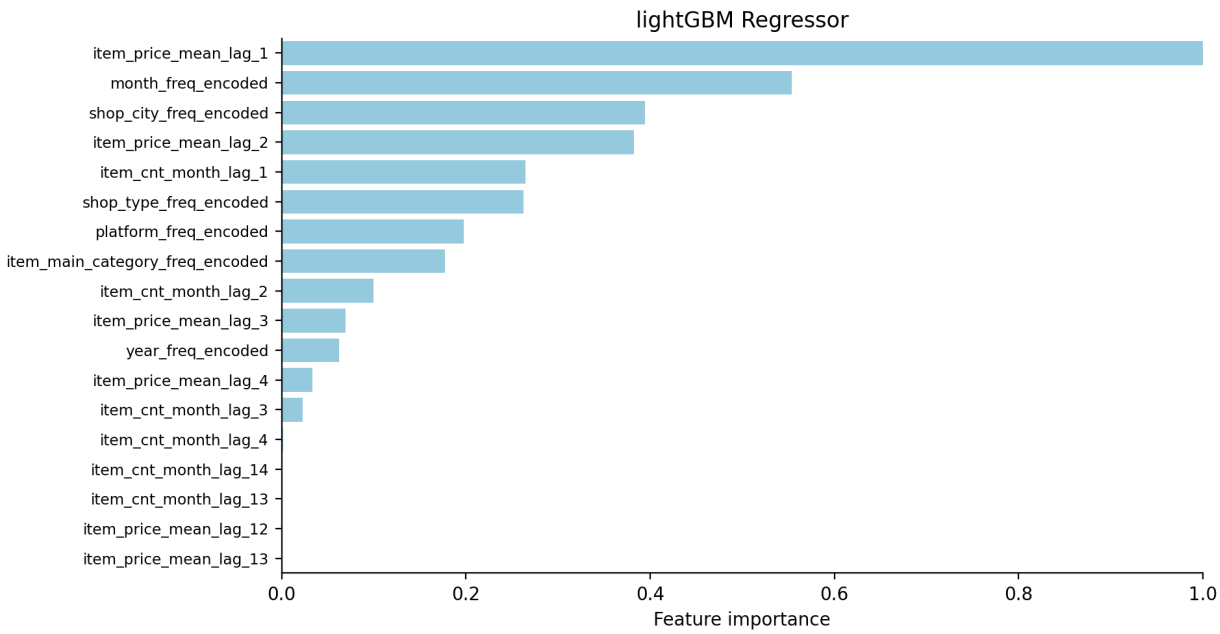


Figure 12: Feature importances in `LGBMRegressor`.

The feature importances vary greatly among the models. For example, `LGBMRegressor` puts virtually no importance on the large lag features, despite our intuition was that sales had seasonality, while `XGBRegressor` and `RandomForestRegressor` have basically the opposite. The `item_price_mean` lag features are important in `LGBMRegressor`, but totally not in `XGBRegressor` and `RandomForestRegressor`. Intuitively, we might expect prices to play some roles, e.g. higher sales when the price of an item drops. The other categorical features are moderately important in `RandomForestRegressor` and `LGBMRegressor`, and are very important in `XGBRegressor`.

Lastly, we note that our best public (RMSE) score was achieved— 1.50672— in an earlier attempt. There we used lag features of 1,2,3,4 and 12 months for the monthly item sales `item_cnt_month`, and only 1 to 4 month lags for the mean item prices `item_price_mean`, frequency encoding of categorical features (we did not encode `year` and `month` at that time), and `RandomForestRegressor` with optimised hyperparameters from `RandomizedSearchCV`. The training and validation RMSE were respectively 2.02259 and 1.90444. The feature importances are plotted below:

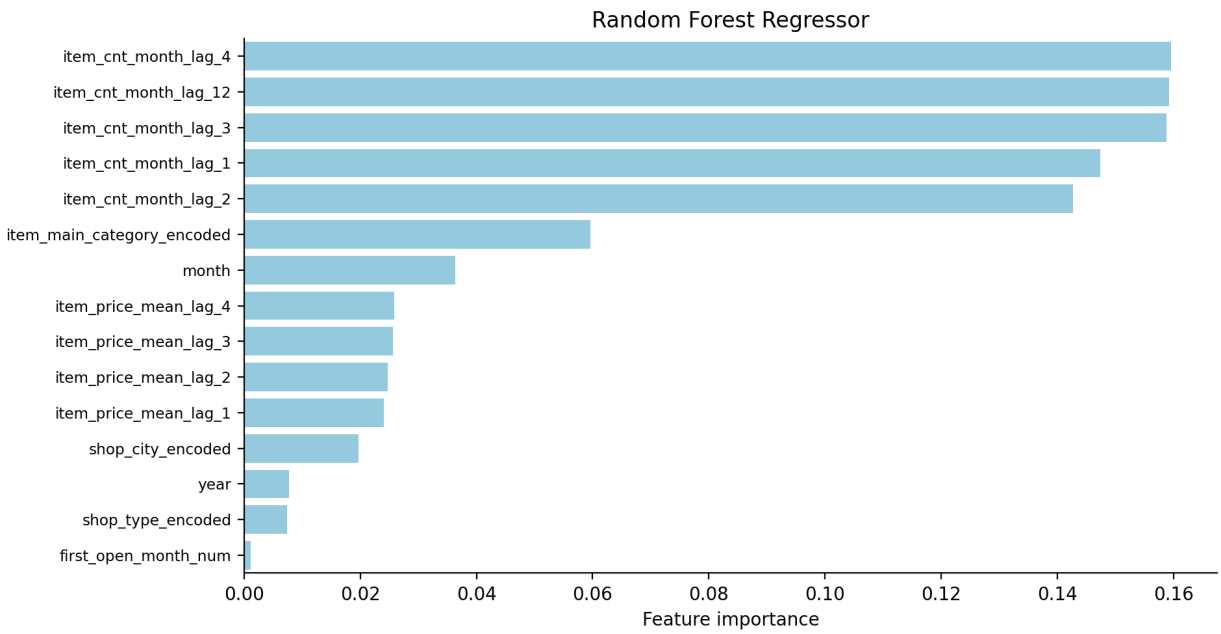


Figure 13: Feature importances in the best performing trial using `RandomForestRegressor`.

Comparing this plot with that of our current attempt Figure 11, we see that they are similar qualitatively: strong emphasis on lag features of monthly sales, and then some categorical features, and less on mean item price lag features. The choice of which months' lag features to include appeared to affect significantly the model performance. By inspection, we knew there was seasonality in the monthly sales count, so it was intuitive to include 12-month lag features. It turned out that including also 13- and 14-month lags for `item_cnt_month` as well as more than 4-month lags for `item_price_mean`, "just to be safe", was detrimental to our goal.

For all the above models, it is curious that the test RMSE were lower than the respective validation RMSE. It may be that these models were all better in predicting the target label in November than in August-October (validation period). This may be because November is closer to the peak holiday season (December), where the seasonal trend is more manifest. This is just my wild guess. Apparently, further study on time series techniques is necessary.

6 Acknowledgment

We acknowledge the authors from various sources online, whose tools and techniques were borrowed and implemented in our codes. (I didn't keep track of the references.)

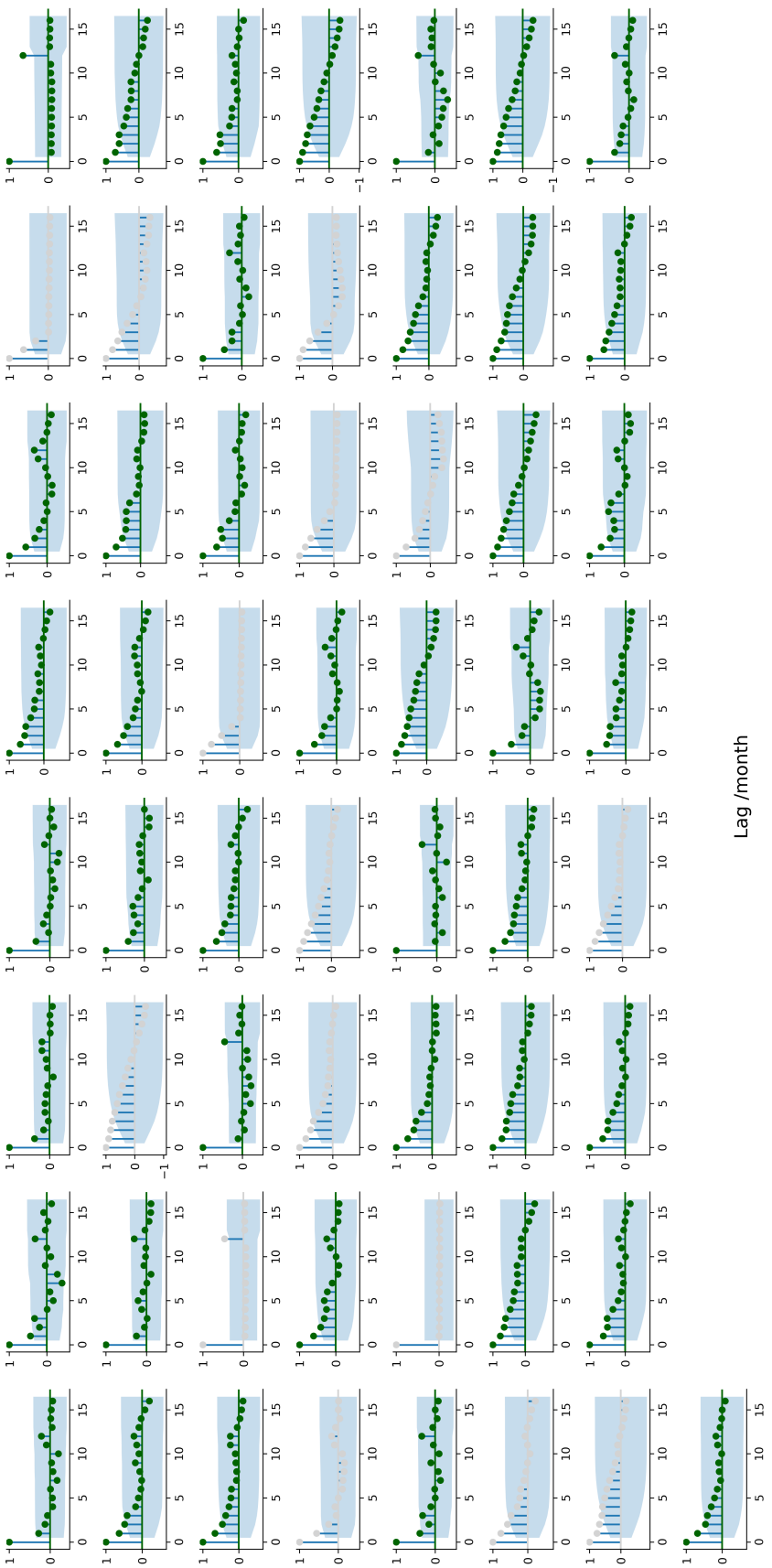


Figure 14: Auto-correlation of monthly sales of each shop. Closed shops are greyed out.

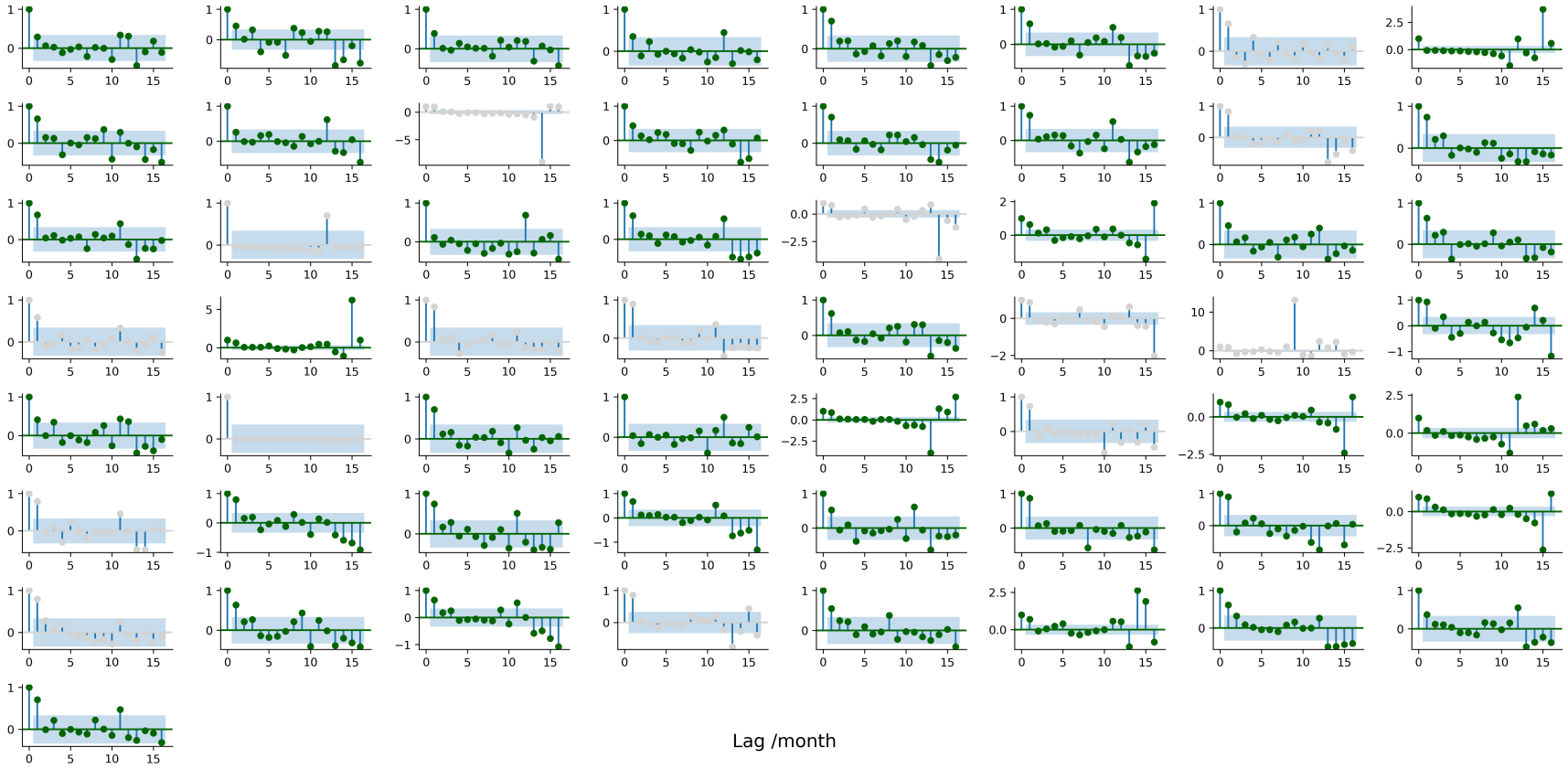


Figure 15: Partial auto-correlation of monthly sales of each shop. Closed shops are greyed out.

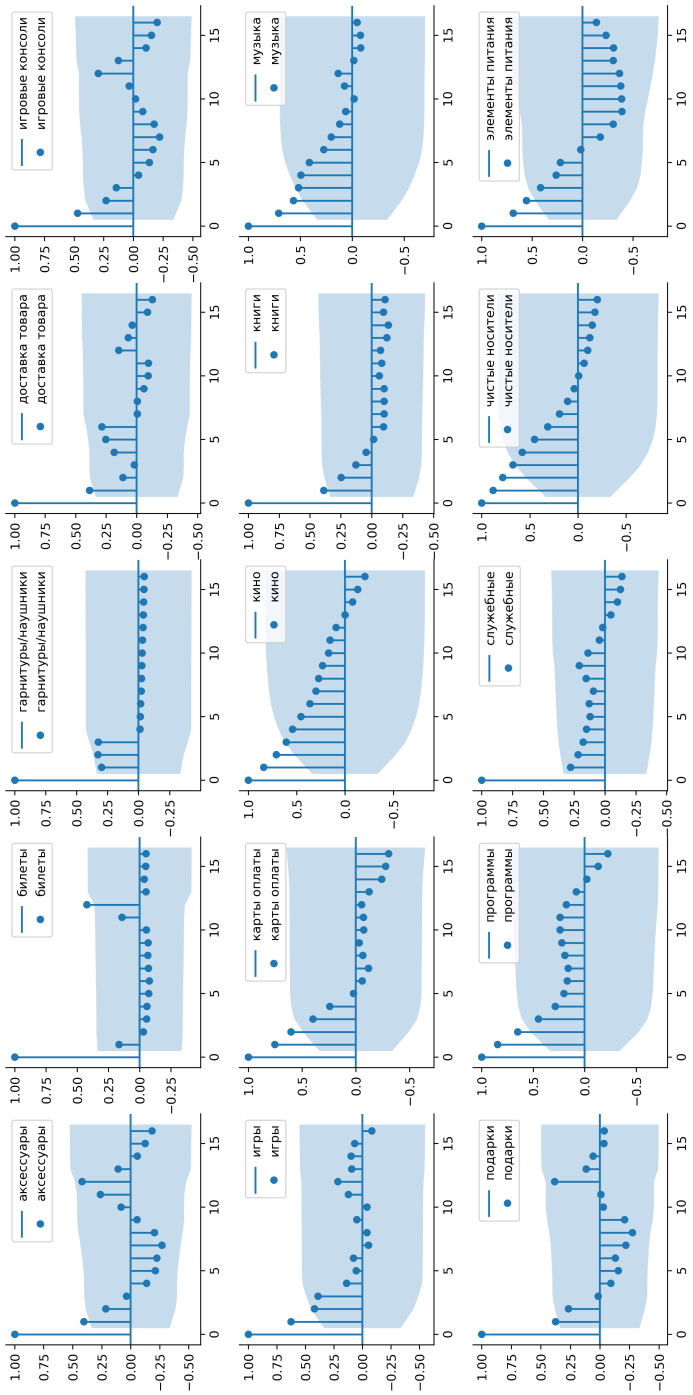


Figure 16: Auto-correlation of monthly sales of each main category.

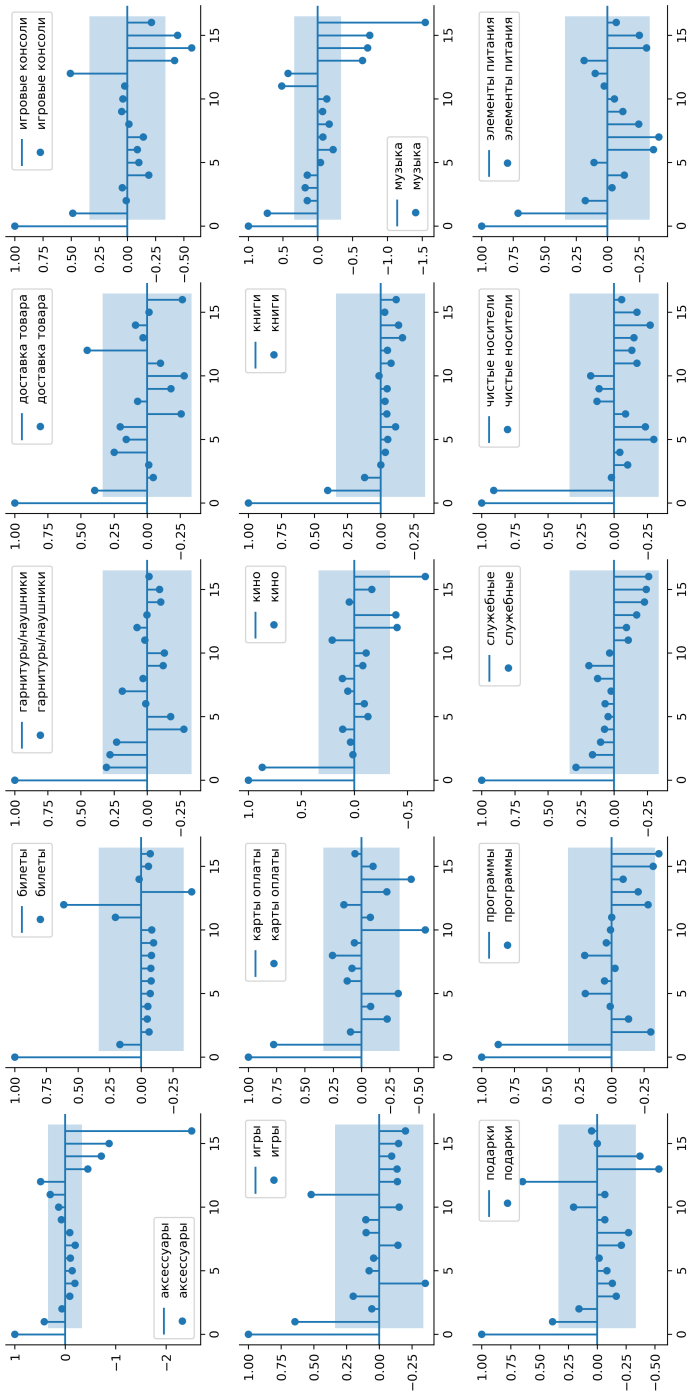


Figure 17: Partial auto-correlation of monthly sales of each main category.

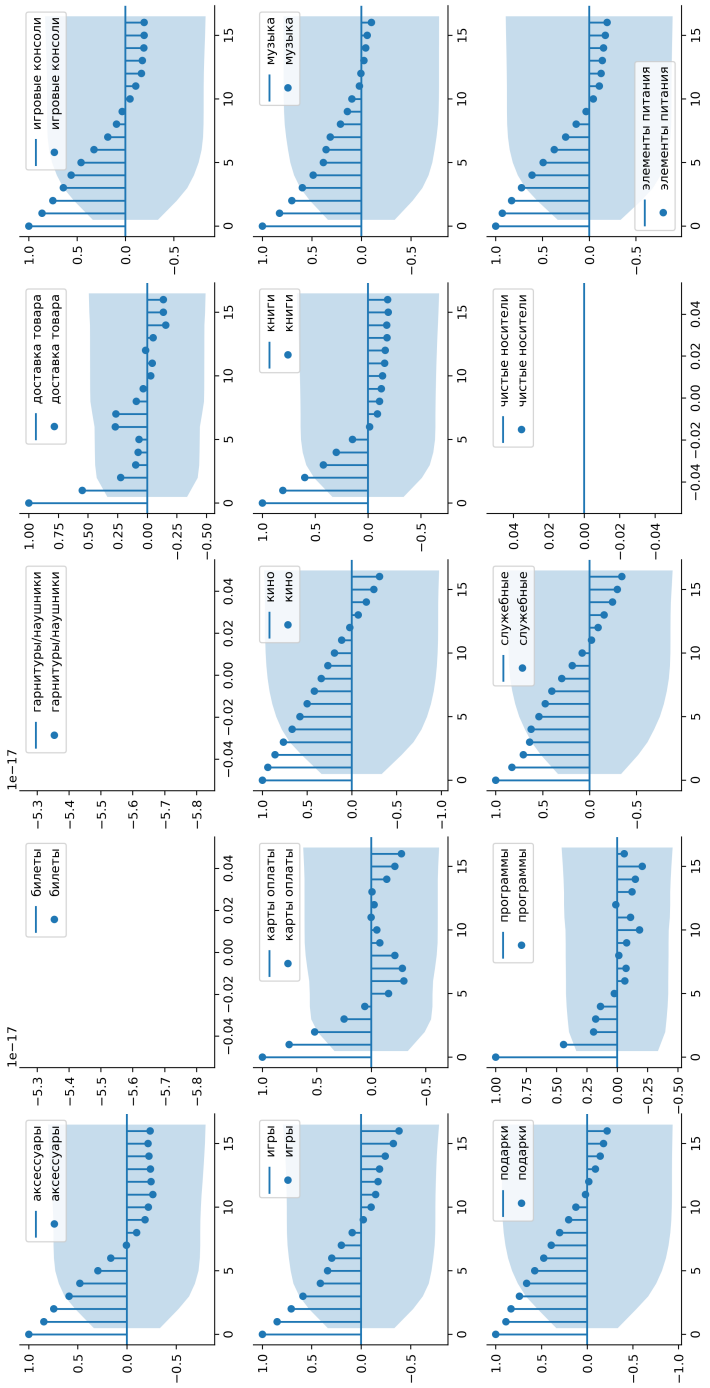


Figure 18: Auto-correlation of monthly mean prices of items in each main category.

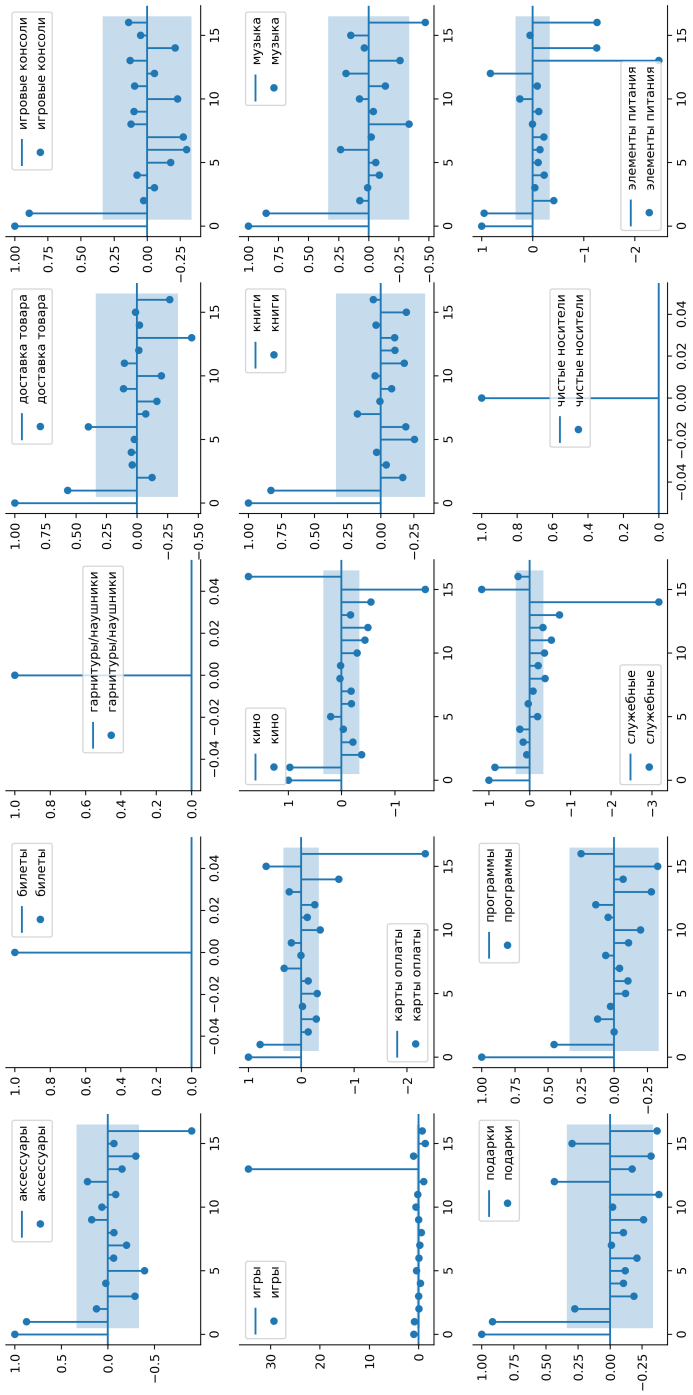


Figure 19: Partial auto-correlation of mean prices of items in each main category.