PREDICT FUTURE SALES

Capstone project

Abstract

For Udacity's Machine Learning Engineer Nanodegree project I have developed the solution for the Kaggle competition to challenge in the project of time-series forecasting to predict sales in the stores of Russian software 1C Company

Galieva, Zarina

dulic11221Gmail.com

Project proposal

Project Overview, Problem Statement, Metrics & Benchmark

Kaggle playground competition:

This challenge serves as final project for the "How to win a data science competition" Coursera course.
The time-series dataset consisting of daily sales data, kindly provided by one of the largest Russian software firms - 1C Company.
The dataset and additional information can be found here:
(https://www.kaggle.com/competitions/competitive-data-science-predict-future-sales)

The main goal is to predict total sales for every product and store in the next month. In context of marketing it's very important as soon as possible know to make business decisions that can bring more profits and see insights in customer behavior for effective investing and optimizing of store. Nowadays there are a plenty of tools that can enable forecast in market or open-source, depending on the level of customers and budget. By using of new techniques and approaches of machine learning libraries it's possible easy and fast to analyze data, create statistics and make predictions.

Motivation: Time-series forecasting is one of machine learning tasks and as my next activity in my job I have project with time-series forecasting. This usecase provides a good opportunity to practice received skills and knowledge from the course and my background to try it out and get a new experience.

Problem statement:

To predict the total amount of products sold in every shop and a total number of sales for each id product based on historical sales data.

Evaluation metrics:

As a metrics for evaluation of predicted target values and actual target values in test data was used root mean squared error (RMSE).

Solution statement:

The provided code will be written on Python3 in Jupyter notebook with using classic data science libraries such as pandas, scikit-learn, matplotlib for plotting and visualization, catboost, xgboost, lightgbm. The type of problem is regression, that means that we need to estimate output and decide which features can impact to target.

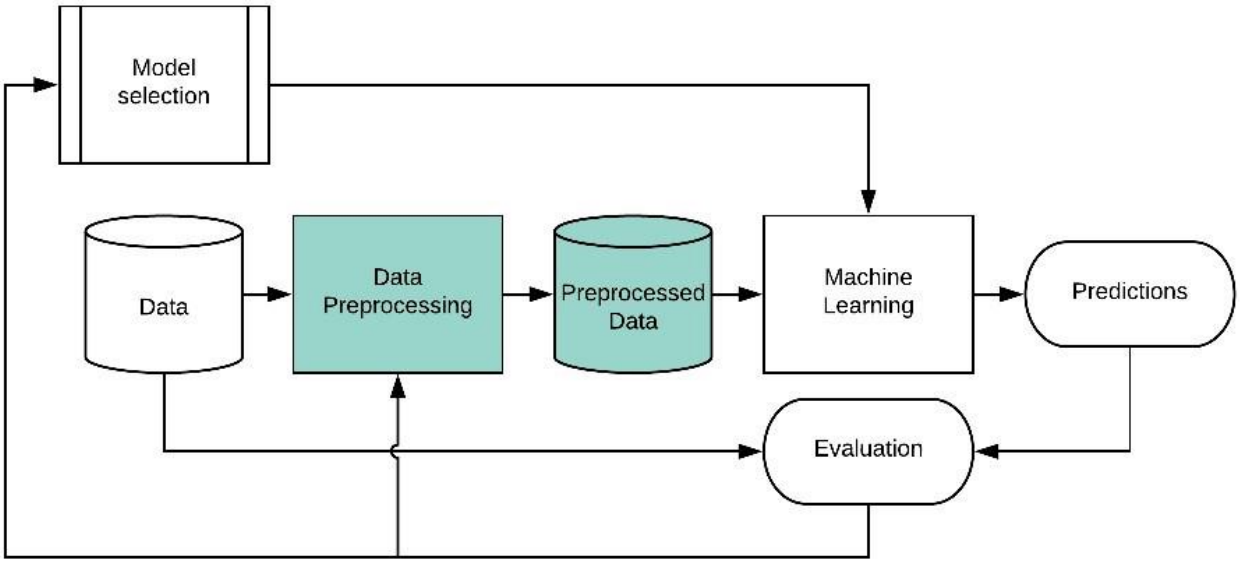
Project design:

As required part firstly is ought to do explorational data analysis for the best understanding of nature of data. Then make feature engineering to preprocess data, visualize to detect anomalies,

outliers in the dataset, look at statistical variables such as mean, minimum and maximum for each feature from tables of data, plotting correlation matrix, graphs, scatter plots, histograms, data distributions), specify types of columns whether its numerical value or categorical consider within on context. In the case of missing values(blanks), typos (wrong unreal entries) in data may be applied imputation (filling in by mean or median value in column) or removing of the rows. It's also need to check for duplicates in rows, often contents of some rows is the same but differs a few letters from each other.

The model selection I will do baseline models to choose the best one or two for next tuning and hyperparameter optimization.

There is the workflow of the process is visualized below on the scheme:



Data description:

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

Data fields

- ID - an Id that represents a (Shop, Item) tuple within the test set
- shop_id - unique identifier of a shop
- item_id - unique identifier of a product
- item_category_id - unique identifier of item category
- item_cnt_day - number of products sold.
- item_price - current price of an item
- date - date in format dd/mm/yyyy

- date_block_num - a consecutive month number, used for convenience. January 2013 is 0, February 2013 is 1,..., October 2015 is 33
- item_name - name of item
- shop_name - name of shop
- item_category_name - name of item category

The inputs variables can be reduced as may affect on target, depending on their importance to model performance.
The target is item_cnt_day, predicting a monthly amount of this measure.

Benchmark Model

There are a plenty of projects implemented solutions in open notebooks of project on kaggle. I have found that some of them are really helpful for beginners and I guess I will use a few approaches and apply techniques for feature engineering, instead that reinvention the wheel.

- https://www.kaggle.com/code/dimitreoliveira/model-stacking-feature-engineering-and-eda/notebook
- https://www.kaggle.com/code/kyakovlev/1st-place-solution-part-1-hands-on-data/notebook
- https://www.kaggle.com/code/dlarionov/feature-engineering-xgboost/notebook

In these workbooks as model were used various models like LSTM, CNN, multilayer perceptron, XGBoost model. The result of the solution(RMSE=1.2) I will use as a benchmark: https://www.kaggle.com/minhtriet/a-beginner-guide-for-sale-data-prediction.

From the guide below for dealing with a big dataset I found using of datatable library for quicker loading, reading massive dataset and reducing of memory size and a lot of hints to manipulate with big data here:

https://www.kaggle.com/code/bextuychiev/how-to-work-w-million-row-datasets-like-a-pro

Model data:

I plan to create a pipeline as a common practice after baseline modeling and cross validation. I will try all of them and then decide what will tune. As I have from my experience tree-based models make impact on performance, mostly I use CatBoost, because it handles categorical variables and takes less time to run, faster 10x than XGBoost. I will try all baseline and then decide Mainly I will concentrate more on doing pipeline as part of Data Engineer activity, than

in exploratory part of data as Data Scientist, because of concept of the nanodegree course is applying skills in building and deploying ML models in production using Amazon Sagemaker.

Project Solution

Exploratory data analysis

There are multiple CSV files with sales data of stores and items sold with multiple data fields. It was performed exploratory data analysis on the datasets with purpose to investigate (and visualise) the data, identify anomalies and deal with them, as well as engineer features for modelling. As a goal is to predict future sales numbers for items sold in store.

The Kaggle dataset consists of daily historical data from the 1st January 2013 to 10th December 2015. It has 2935849 data rows, and 5 variables including one response variable, the number of products sold (item_cnt_day). We are predicting a monthly amount of this measure. To prepare the data, we converted the data variable to date format and the number variables into factor variables.

```
# Look at time period of data
print('Min date from train set: %s' % train['date'].min().date())
print('Max date from train set: %s' % train['date'].max().date())
```

```
Min date from train set: 2013-01-01
Max date from train set: 2015-12-10
```

Data is from 1st January 2013 to 10th Decemer 2015, as we expect

There are no any missing or null values:

```
----------Missing value-----------
date                0
date_block_num      0
shop_id             0
item_id             0
item_price          0
item_cnt_day        0
item_name           0
item_category_id    0
shop_name           0
item_category_name  0
dtype: int64
----------Null value-----------
date                0
date_block_num      0
shop_id             0
item_id             0
item_price          0
item_cnt_day        0
item_name           0
item_category_id    0
shop_name           0
item_category_name  0
dtype: int64
```

Before merging of data sets, good approach is to reduce their huge weights. Because of massive data sets size it was required to downcast types of features by specifying of their types for less memory usage and reducing of their allocation on RAM (inspired by notebook: https://www.kaggle.com/code/bextuychiev/how-to-work-w-million-row-datasets-like-a-pro)

```
# Load in dataset (cast float64 -> float32 and int32 -> int16 to save memory)
items = pd.read_csv('../input/items.csv',
                    dtype={'item_name': 'str', 'item_id': 'int16', 'item_category_id': 'int16'}
                   )
shops = pd.read_csv('../input/shops.csv',
                    dtype={'shop_name': 'str', 'shop_id': 'int16'}
                   )
categories = pd.read_csv('../input/item_categories.csv',
                         dtype={'item_category_name': 'str', 'item_category_id': 'int16'}
                        )
train = pd.read_csv('../input/sales_train.csv',
                    dtype={
                        'date': 'str',
                        'date_block_num': 'int16',
                        'shop_id': 'int16',
                        'item_id': 'int16',
                        'item_price': 'float32',
                        'item_cnt_day': 'int16'}
                   )
# set index to ID to avoid dropping it later
test  = pd.read_csv('../input/test.csv',
                    dtype={'ID': 'int16', 'shop_id': 'int16', 'item_id': 'int16'}
                   ).set_index('ID')
```

The lack of memory also compels to focus only on rows which are presented on test set, so it means removing rows in training set that don't match with test set.

```
test_shop_ids = test['shop_id'].unique()
test_item_ids = test['item_id'].unique()

# Only shops that exist in test set.
corrlate_train = train[train['shop_id'].isin(test_shop_ids)]
# Only items that exist in test set.
correlate_train = corrlate_train[corrlate_train['item_id'].isin(test_item_ids)]
```

```
print('Initial data set size :', train.shape[0])
print('Data set size after matching crossovers between train and test:', correlate_train.shape[0])
```

```
Initial data set size : 2935849
Data set size after matching crossovers between train and test: 1224439
```

After reducing of datasets we will check for duplicates. The number of such records =5 not so meaningful with comparison of the whole data, so we will leave them for now.

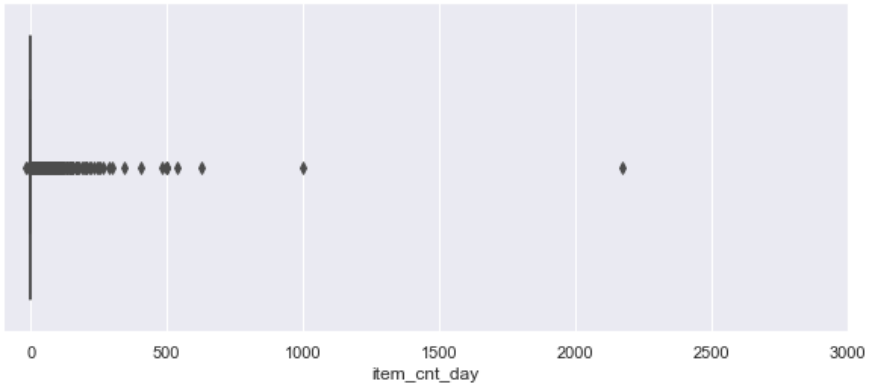
```
train[train.duplicated()]
```

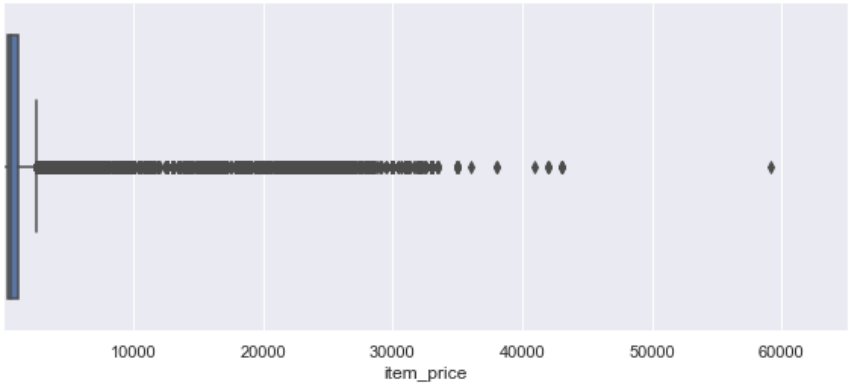
	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_name	item_category_id	shop_name	item_category_name
1435367	2014-02-23	13	50	3423	999.00	1	Far Cry 3 (Classics) [Xbox 360, русская версия]	23	Тюмень ТЦ "Гудвин"	Игры - XBOX 360
1496766	2014-03-23	14	21	3423	999.00	1	Far Cry 3 (Classics) [Xbox 360, русская версия]	23	Москва МТРЦ "Афи Молл"	Игры - XBOX 360
1671873	2014-01-05	16	50	3423	999.00	1	Far Cry 3 (Classics) [Xbox 360, русская версия]	23	Тюмень ТЦ "Гудвин"	Игры - XBOX 360
1866340	2014-12-07	18	25	3423	999.00	1	Far Cry 3 (Classics) [Xbox 360, русская версия]	23	Москва ТРК "Атриум"	Игры - XBOX 360
2198566	2014-12-31	23	42	21619	499.00	1	ЧЕЛОВЕК ДОЖДЯ (BD)	37	СПб ТК "Невский Центр"	Кино - Blu-Ray

Let’s glance on plots of outliers on columns ‘item_cnt_day’ and ‘item_price’, it looks like there are a few of items are out of distributions:

```
plt.figure(figsize=(10,4))
plt.xlim(-100, 3000)
sns.boxplot(x=train.item_cnt_day)

plt.figure(figsize=(10,4))
plt.xlim(train.item_price.min(), train.item_price.max()*1.1)
sns.boxplot(x=train.item_price)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18881fac640>
```






```
train = train[train.item_price<100000]
train = train[train.item_cnt_day<1000]
```

By looking through columns in feature we can see that some information might be useful, especially for creation of new categorical columns. Let's divide shop_name column to separate shop_city and shop_name.

```
# Encode the city name into a code column
train['city_code'] = LabelEncoder().fit_transform(train['city'])
train.head()
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_name	item_category_id	shop_name	item_category_name	city	city_code
0	2013-02-01	0	59	22154	999.00	1	ЯВЛЕНИЕ 2012 (BD)	37	Ярославль ТЦ "Альтаир"	Кино - Blu-Ray	Ярославль	27
10	2013-03-01	0	25	2574	399.00	2	DEL REY LANA Born To Die The Paradise Editio...	55	Москва ТРК "Атриум"	Музыка - CD локального производства	Москва	12
11	2013-05-01	0	25	2574	399.00	1	DEL REY LANA Born To Die The Paradise Editio...	55	Москва ТРК "Атриум"	Музыка - CD локального производства	Москва	12
12	2013-07-01	0	25	2574	399.00	1	DEL REY LANA Born To Die The Paradise Editio...	55	Москва ТРК "Атриум"	Музыка - CD локального производства	Москва	12

Alternatively, for item_category_name let's do new features by splitting then dividing to item_category_type and item_subcategory_type as well. Also applied encoding transformation created columns by Label Encoder.

```
# Create separate column with split category name
train['split_category_name'] = train['item_category_name'].str.split('-')
train.head()
```

```
# Make column for category type and encode
train['item_category_type'] = train['split_category_name'].map(lambda x : x[0].strip())
train['item_category_type_code'] = LabelEncoder().fit_transform(train['item_category_type'])
```

After that we can drop original columns corresponding to new created.

```
train = train.drop(['shop_name',
                    'item_category_name',
                    'split_category_name',
                    'item_category_type',
                    'item_category_subtype',
                   ], axis = 1)
train.head()
```

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_name	item_category_id	city	city_code	item_category_type_code	item_category_subtype_code
0	2013-02-01	0	59	22154	999.00	1	ЯВЛЕНИЕ 2012 (BD)	37	Ярославль	27	7	1
10	2013-03-01	0	25	2574	399.00	2	DEL REY LANA Born To Die The Paradise Editio...	55	Москва	12	9	2
11	2013-05-01	0	25	2574	399.00	1	DEL REY LANA Born To Die The Paradise Editio...	55	Москва	12	9	2
12	2013-07-01	0	25	2574	399.00	1	DEL REY LANA Born To Die The Paradise Editio...	55	Москва	12	9	2
13	2013-08-01	0	25	2574	399.00	2	DEL REY LANA Born To Die The Paradise Editio...	55	Москва	12	9	2

As the next step for grouping our dataset we will group rows by aggregation of date_block_num, so we can predict the monthly data sets

```
train_by_month = train.sort_values('date').groupby([
                                                    'date_block_num',
                                                    'item_category_type_code',
                                                    'item_category_subtype_code',
                                                    'item_name_code',
                                                    'city_code',
                                                    'shop_id',
                                                    'item_category_id',
                                                    'item_id',
    # Keep simple; will just use the above columns
], as_index=False)
train_by_month.size()
```

```
date_block_num  item_category_type_code  item_category_subtype_code  item_name_code  city_code  shop_id  item_category_id  item_id
0               0                       9                           365             0          2        2                 5572       7
                                                                                    1          3        2                 5572       3
                                                                                    2          4        2                 5572       3
                                                                                    4          6        2                 5572       7
                                                                                               7        2                 5572       7
                                                                                                                                    ..
33              13                      47                          1579            26         58       83                22088      2
                                                                                                                          22091      1
                                                                                    27         59       83                22087      3
                                                                                                                          22088      2
                                                                                                                          22091      1
Length: 600158, dtype: int64
```

Some item prices have different values in shops and during season, these changes make influence on trend and count of sales, let's create new features like sum, mean, and additional count for each row in train.

```
# Aggregate item price and item count
train_by_month = train_by_month.agg({'item_price':['sum', 'mean'], 'item_cnt_day':['sum', 'mean','count']})
```

```
train_by_month.head()
```

	date_block_num	item_category_type_code	item_category_subtype_code	item_name_code	city_code	shop_id	item_category_id	item_id	item_price		item_cnt_day		
									sum	mean	sum	mean	count
0	0	0	9	365	0	2	2	5572	10730.00	1532.86	9	1.29	7
1	0	0	9	365	1	3	2	5572	4770.00	1590.00	3	1.00	3
2	0	0	9	365	2	4	2	5572	4570.00	1523.33	3	1.00	3
3	0	0	9	365	4	6	2	5572	11130.00	1590.00	8	1.14	7
4	0	0	9	365	4	7	2	5572	11030.00	1575.71	8	1.14	7

As we have to apply predictions to the test set, we must ensure all possible combinations of "shop_id" and "item_id" are covered.

To do this, we will loop through all possible combinations in our test set and append to an empty dataframe. Then we will merge that empty dataframe to our main dataframe and fill in missing na values with 0.

Also need to check for nan values(isna()), if they are fill the gaps by 0 (fillna())

```
# Get all unique shop id's and item id's
shop_ids = test['shop_id'].unique()
item_ids = test['item_id'].unique()
# Initialise empty df
empty_df = []
# Loop through months and append to dataframe
for i in range(34):
    for item in item_ids:
        for shop in shop_ids:
          empty_df.append([i, shop, item])
# Turn into dataframe
empty_df = pd.DataFrame(empty_df, columns=['date_block_num','shop_id','item_id'])
```

```
# Merge monthly train set with the complete set (missing records will be filled with 0).
train_by_month = pd.merge(train_by_month, empty_df, on=['date_block_num','shop_id','item_id'], how='outer')
len(train_by_month)
```

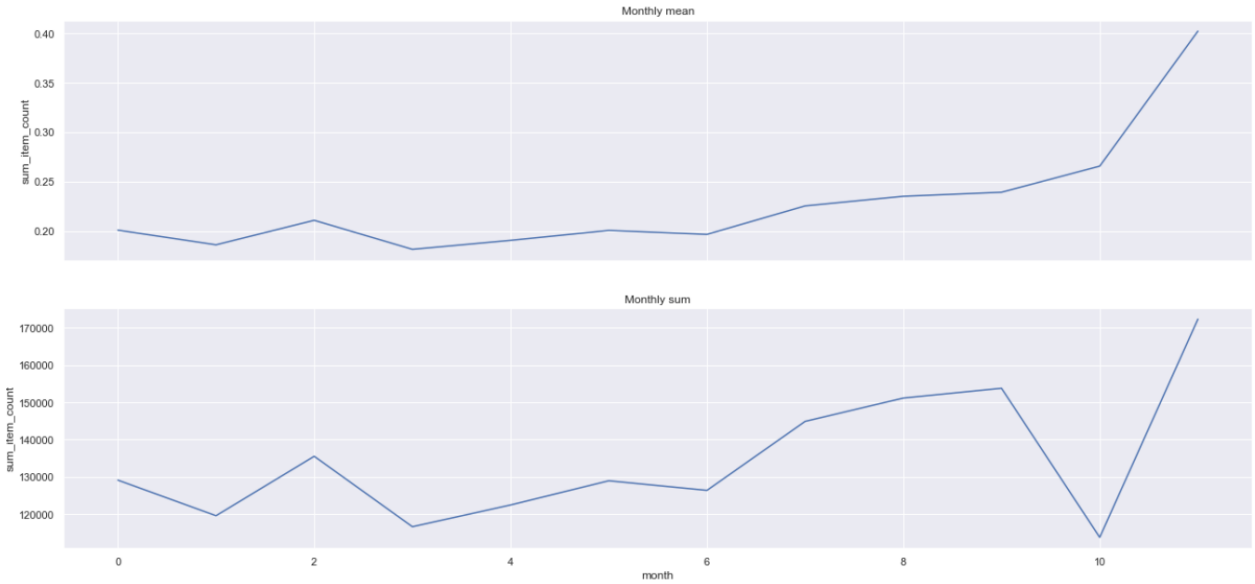
```
7282800
```

Graphs and plots are important parts in EDA for finding insights in data, so next groupings of dataframes shows us how sales behave in each year (created as new features by splitting of datetime). We do this to investigate the seasonality of our data, and find out how sales may change year by year. These could therefore turn out to be important features in modeling our data. The figures directly below are a series of plots in investigating more trends of seasonality and item correlation.

```
# Grouping data to plot

# Group by month
gp_month_mean = train.groupby(['month'], as_index=False)['sum_item_count'].mean()
gp_month_sum = train.groupby(['month'], as_index=False)['sum_item_count'].sum()

# Group by item_category
gp_category_mean = train.groupby(['item_category_id'], as_index=False)['sum_item_count'].mean()
gp_category_sum = train.groupby(['item_category_id'], as_index=False)['sum_item_count'].sum()

# Group by shop_id
gp_shop_mean = train.groupby(['shop_id'], as_index=False)['sum_item_count'].mean()
gp_shop_sum = train.groupby(['shop_id'], as_index=False)['sum_item_count'].sum()
```

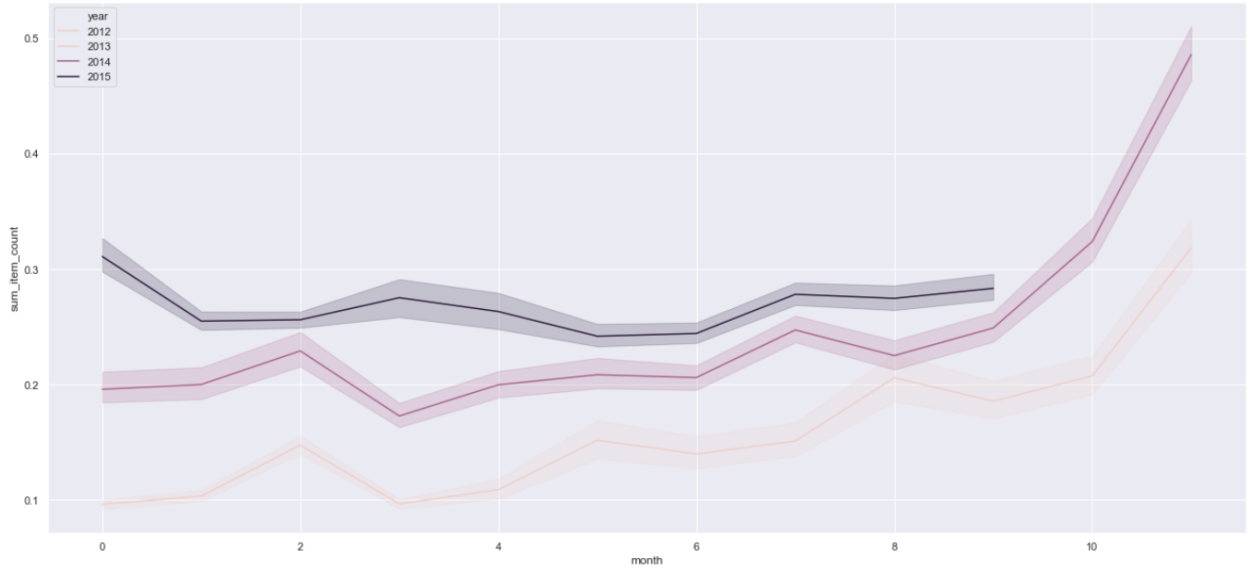
```
# Group by item name code
gp_item_name_code_mean = train.groupby(['item_name_code'], as_index=False)['sum_item_count'].mean()
gp_item_name_code_sum =  train.groupby(['item_name_code'], as_index=False)['sum_item_count'].sum()

# Group by city code
gp_city_code_mean = train.groupby(['city_code'], as_index=False)['sum_item_count'].mean()
gp_city_code_sum =  train.groupby(['city_code'], as_index=False)['sum_item_count'].sum()
```

Vizualization plots



Overall item count mean average and sum by month. We can see that there is an increase towards the end of the year which could correlate with the festive period and a lull in the mid months. In the lower graph, the monthly sum, we can see there is a dip on the 10th month and 3rd which also corresponds with the monthly mean.

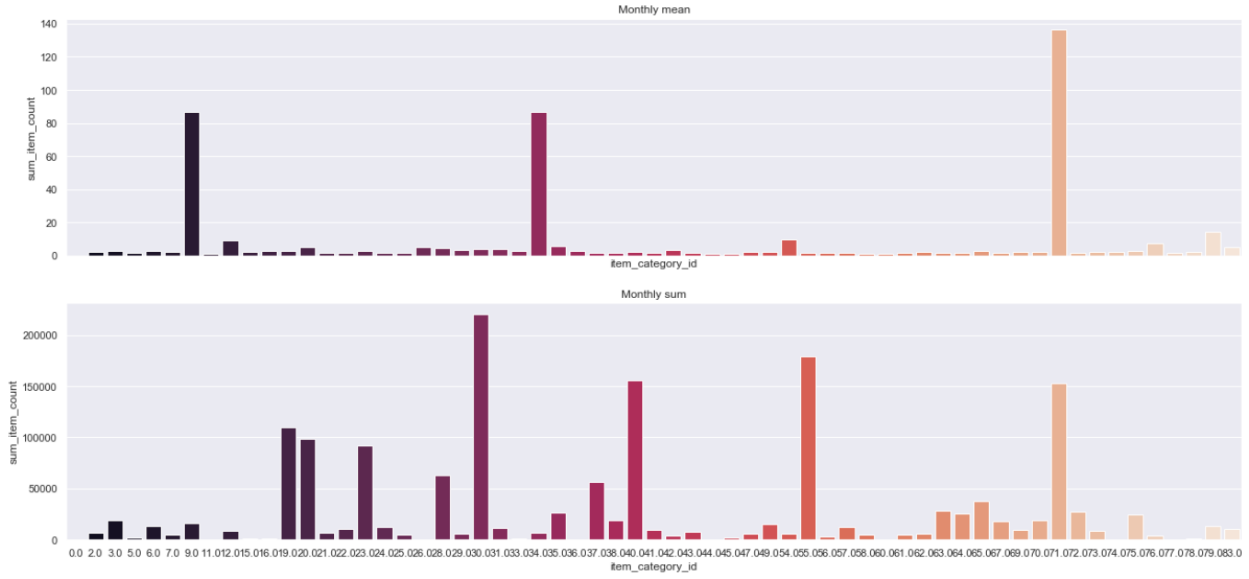
```
# Also plot the item counts by year
f, axes = plt.subplots(1, 1, figsize=(22, 10), sharex=True)
sns.lineplot(x="month", y="sum_item_count", hue="year",data=train)
plt.show()
```



Item count mean sales by year. It appears sales improve year on year; with 2014 outperforming 2013 and 2015 outperforming 2014. Here we can also see seasonal trends in our datta, observing dips in

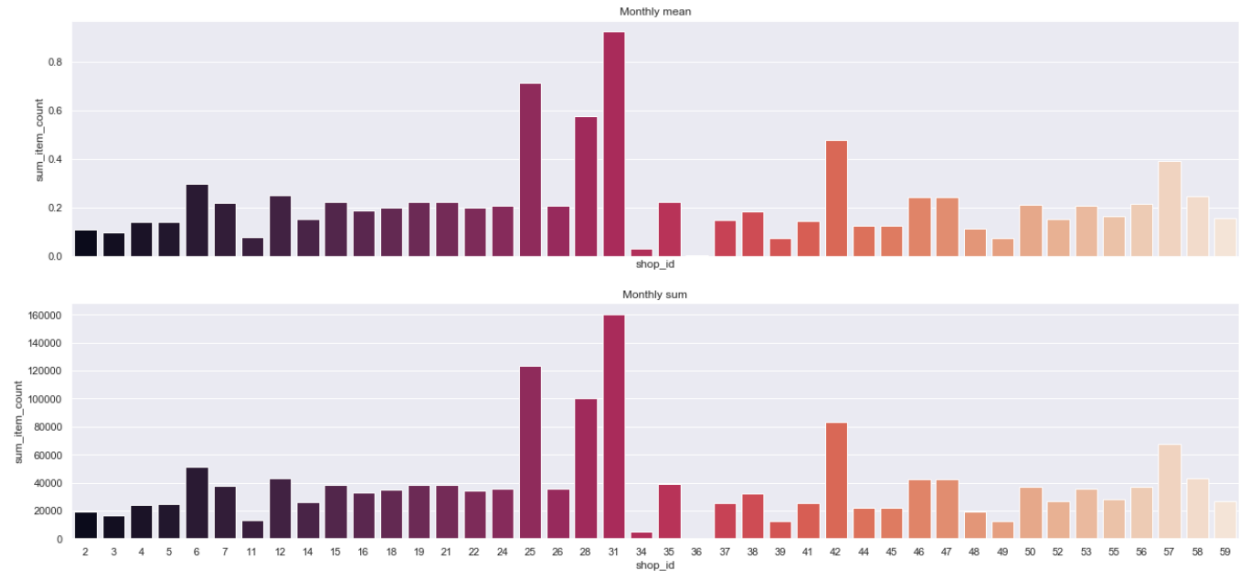
month 3, which does not occur in 2015. Notably, the data for 2015 is incomplete, so stops at month 9. We can, however, make inferences from m2013 and 2014 and so expect the value to rise for later months in accordance with 2015.

```
# We can also plot by item category id
f, axes = plt.subplots(2, 1, figsize=(22, 10), sharex=True)
sns.barplot(x="item_category_id", y="sum_item_count", data=gp_category_mean, ax=axes[0], palette="rocket").set_title("Monthly mean")
sns.barplot(x="item_category_id", y="sum_item_count", data=gp_category_sum, ax=axes[1], palette="rocket").set_title("Monthly sum")
plt.show()
```



Item count sales for category ids. We can see that a few items sell very well and others very little especially in the monthly mean. On the other hand, the monthly sum bar plot shows more variability in distribution of item sales by category id.

```
# We can also plot by shop id
f, axes = plt.subplots(2, 1, figsize=(22, 10), sharex=True)
sns.barplot(x="shop_id", y="sum_item_count", data=gp_shop_mean, ax=axes[0], palette="rocket").set_title("Monthly mean")
sns.barplot(x="shop_id", y="sum_item_count", data=gp_shop_sum, ax=axes[1], palette="rocket").set_title("Monthly sum")
plt.show()
```



Item count sales according to shop ids. There appears to be about 4 shops that sell far above the others, which have a similar amount of sales. It is possible there are other reasons for this such as shop location or size which could be a future factor to investigate.

Feature Engineering(continuous)

As soon as we will predict sum item count of items for month for doing this will approach this by creating a target (Y) column which is the item count sales of the next month for a given item, and is done for each row, also will create a column of item count for the next month for each item id.

```
# Create a column with item price per unit
train['item_price_unit'] = train['sum_item_price'] // train['sum_item_count']

# Replace inf with Nan (occurs when division by zero)
train['item_price_unit'] = train['item_price_unit'].replace([np.inf, -np.inf], np.nan)
# fillna witth 0
train['item_price_unit'].fillna(0, inplace=True)

print(train['item_price_unit'].max()) # Sanity check
train.head()
```

```
94970.0
```

de	shop_id	item_category_id	item_id	sum_item_price	mean_item_price	sum_item_count	mean_item_count	transactions	year	month	sum_item_cnt_next_month	item_price_unit
.00	2	2.00	5572	10730.00	1532.86	9.00	1.29	7.00	2013	0	9.00	1192.00
.00	3	2.00	5572	4770.00	1590.00	3.00	1.00	3.00	2013	0	3.00	1590.00
.00	4	2.00	5572	4570.00	1523.33	3.00	1.00	3.00	2013	0	3.00	1523.00
.00	6	2.00	5572	11130.00	1590.00	8.00	1.14	7.00	2013	0	8.00	1391.00
.00	7	2.00	5572	11030.00	1575.71	8.00	1.14	7.00	2013	0	8.00	1378.00

We also make the maximum and minimum item price a feature of our dataset. With that we can have additional columns in each row of how much the value deviates from the maximum and minimum values accordingly (price increase and price decrease)

```
# Aggregate how a given item price has changed from it's minimum and maximum prices
train['price_increase'] = train['sum_item_price'] - train['hist_min_item_price']
train['price_decrease'] = train['hist_max_item_price'] - train['sum_item_price']
train.head()
```

ean_item_price	...	mean_item_count	transactions	year	month	sum_item_cnt_next_month	item_price_unit	hist_min_item_price	hist_max_item_price	price_increase	price_decrease
1532.86	...	1.29	7.00	2013	0	9.00	1192.00	0.00	18979.50	10730.00	8249.50
1590.00	...	1.00	3.00	2013	0	3.00	1590.00	0.00	18979.50	4770.00	14209.50
1523.33	...	1.00	3.00	2013	0	3.00	1523.00	0.00	18979.50	4570.00	14409.50
1590.00	...	1.14	7.00	2013	0	8.00	1391.00	0.00	18979.50	11130.00	7849.50
1575.71	...	1.14	7.00	2013	0	8.00	1378.00	0.00	18979.50	11030.00	7949.50

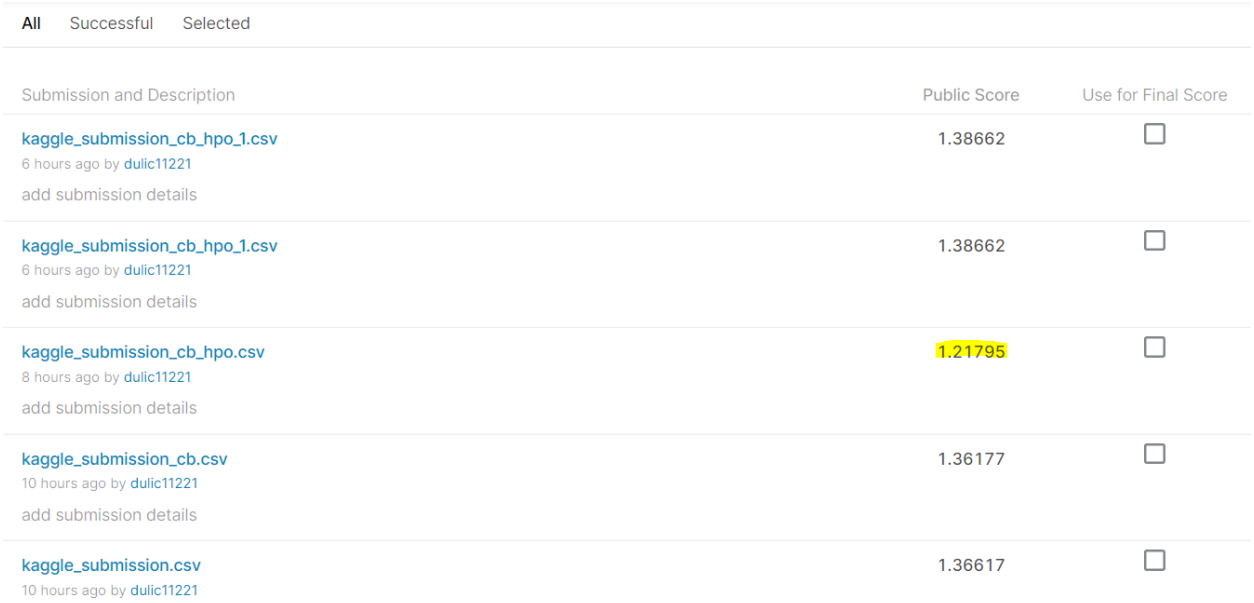
In time series prediction, a powerful technique is using a rolling window. We create a 3 month rolling window for our features and calculate summary statistics (max, min, mean and standard deviation) of our target metric, the sum_item_count and append these as columns to our dataframe. 3 months is chosen as it is long enough to capture a season, hoping to capture seasonal trends without being too large making computation detrimental. If we had more historical data and greater access to computing resources, we could date a rolling window back further to capture longer term trends. This has the advantages of also helping to assess and gauge the instabilities of our data in our model by comparing to past data, which may be useful (Reference: https://www.mathworks.com/help/econ/rolling-window-estimation-of-state-space-models.html). Because of limited memory, we only group by primary

categories of shop_id, item_category_id and item_id, filling in empty features with 0. Another powerful feature in time series analysis is having lag features, these are variables carried over from prior time steps that may be useful to determine future time steps.

Modeling, baseline models, validation and HPO

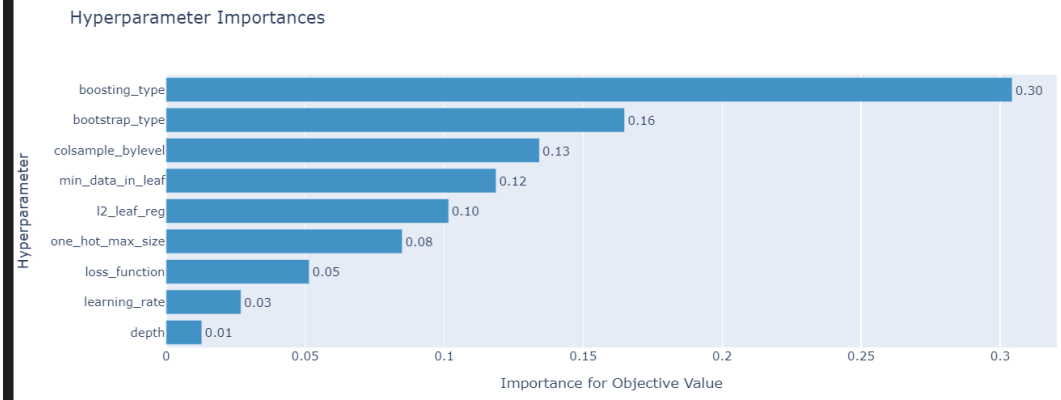
I have tried list of baseline models, such as Linear Regression, Lasso, Random Forest Regressor, Extra Trees Regressor, Gradient Boosting Regressor, but the favorite one is CatBoost. It has the best result from all of them.

I have used hyperparameter optimization library optuna, where was created study with a few trials. By defining of distribution of values for each parameter in task was run number of experiments during them were explored dependencies between attributes, so the compliance of trial is more as more the depth of built trees, it means that need to more explore categorical features for creation of new one for getting bigger p-value importance of feature, for now with playing of around values on attributes on searching space in parameters I was able to get and reach the value of benchmark metrics (RMSE =1.2)in notebook (https://www.kaggle.com/minhtriet/a-beginner-guide-for-sale-data-prediction.)

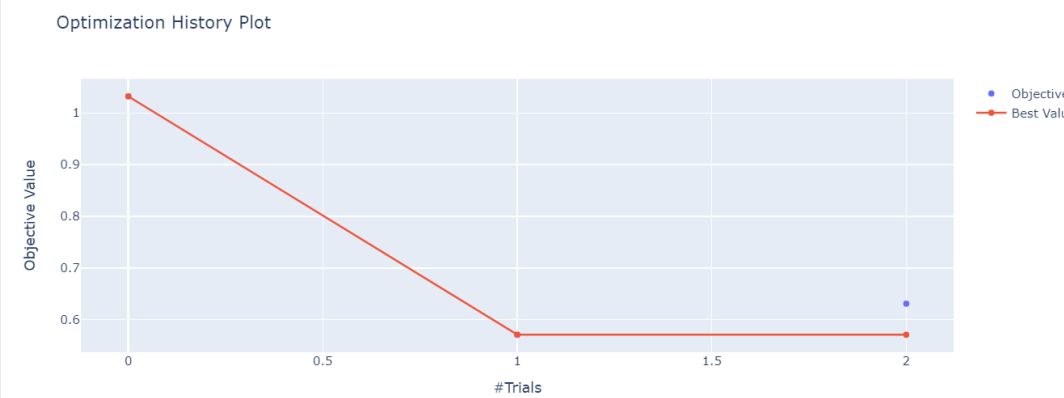


Appendix

```
optuna.visualization.plot_param_importances(study)
```




```
optuna.visualization.plot_optimization_history(study)
```




```
optuna.visualization.plot_slice(study, params=['depth', 'learning_rate', 'bootstrap_type'])
```