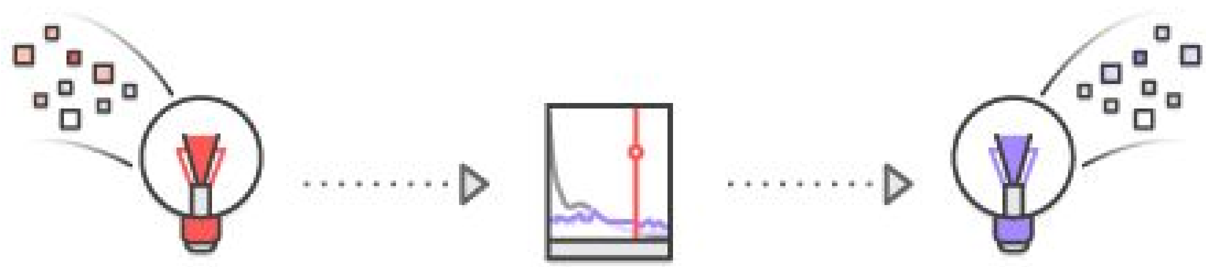
Predict Future Sales

- Kaggle Competition



Stakeholders

- Upeksha Liyanage 15000771
- Kshithija Liyanage 15000788
- Panduka Liyanathanthri 15000796
- Basuru Amaradiwakara 15000087

Content

1. Problem Definition
    a. Challenge Description 03
    b. Data Description 03
    c. File Description 04
2. Data Analysis
    a. Daily Sales Analysis 05
    b. Monthly Sales Analysis 07
3. Model Selection 09
    a. Supervised Model
        i. Rolling WIndows-Based Regression..11
        ii. Time Series Model
            1. ARIMA 09

2. SARIMA ……………………………………………………………………10
iii. Models Comparisons
1. XGBoost vs LightGBM……………………………………………………10
2. Boosting Methods vs Networks…………………………………..10
b. Model Development ……………………………………………………………………11
4. Data Preprocessing
a. Feature Engineering
i. Downcasting Variables……………………………………………………………12
ii. Merging Features ……………………………………………………………………13
iii. Adding Windows Features…………………………………………………………14
iv. Adding Missing Values……………………………………………………………16
v. Add Rolling Windows Features (Sum and Mean)……………………16
1. Preprocessed Data…………………………………………………………19

vi. Add Lag Features ……..……………………………………………………….20
vii. Add Seasonal Components / Holidays……………..………………….…20
b. Cross Validation and Hyperparameter Tuning……………………………………21
i. Hyperparameters For Light GBM……………………………………………21
c. Regularization ………………………….……………………………….…22
5. References ………………..………………………………….………23

Problem Definition

Challenge Description

This competition consist of challenging time-series dataset consisting of daily sales data, provided by one of the largest Russian software firms - 1C Company. Challenge is to predict total sales for every product a store will sell in the next month.

Data Description

Daily historical sales data of 1C Company is provided as training data. The task is to forecast the total amount of products sold per month in every shop for the test set. List of shops and products slightly changes every month. Data fields

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

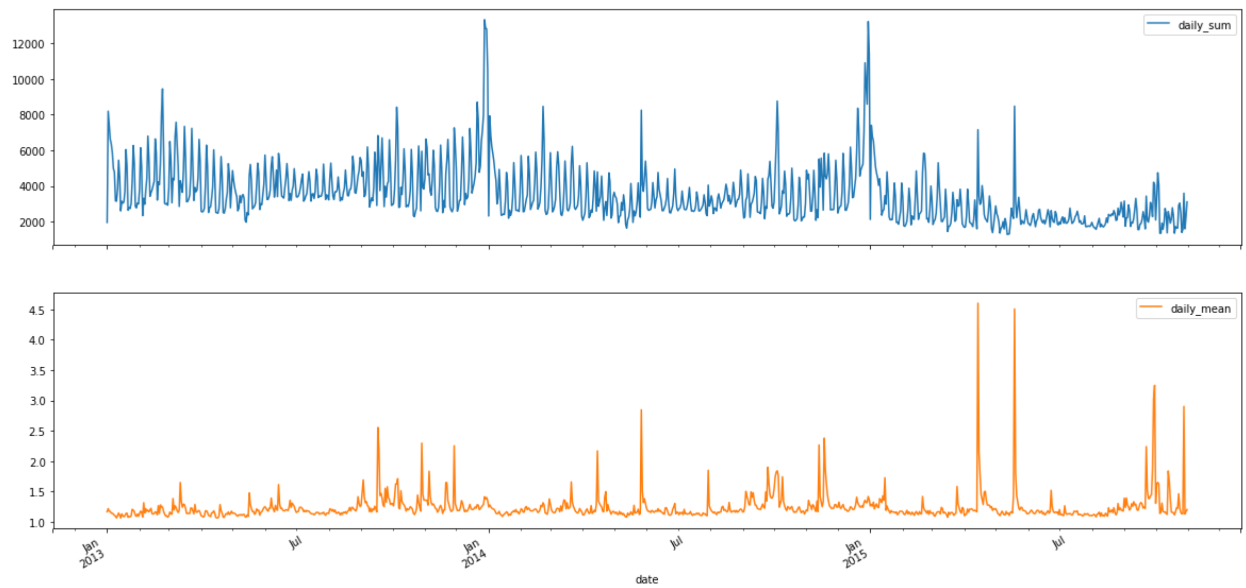
File Descriptions

- sales_train.csv - the training set. Daily historical data from January 2013 to October 2015.
- test.csv - the test set. 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.
- As we analysed the dataset we found out the following facts
- Item_cnt_data in sales_train.csv file contains both positive and negative values and it does not have any zero values. This lead us to conclusion that this file contains only sales and returns per day for a particular item at a particular shop. Also we found out that shops, items and item categories are not consistent throughout the time period. They does not appear for some months.

Data Analysis

Daily Sales Analysis

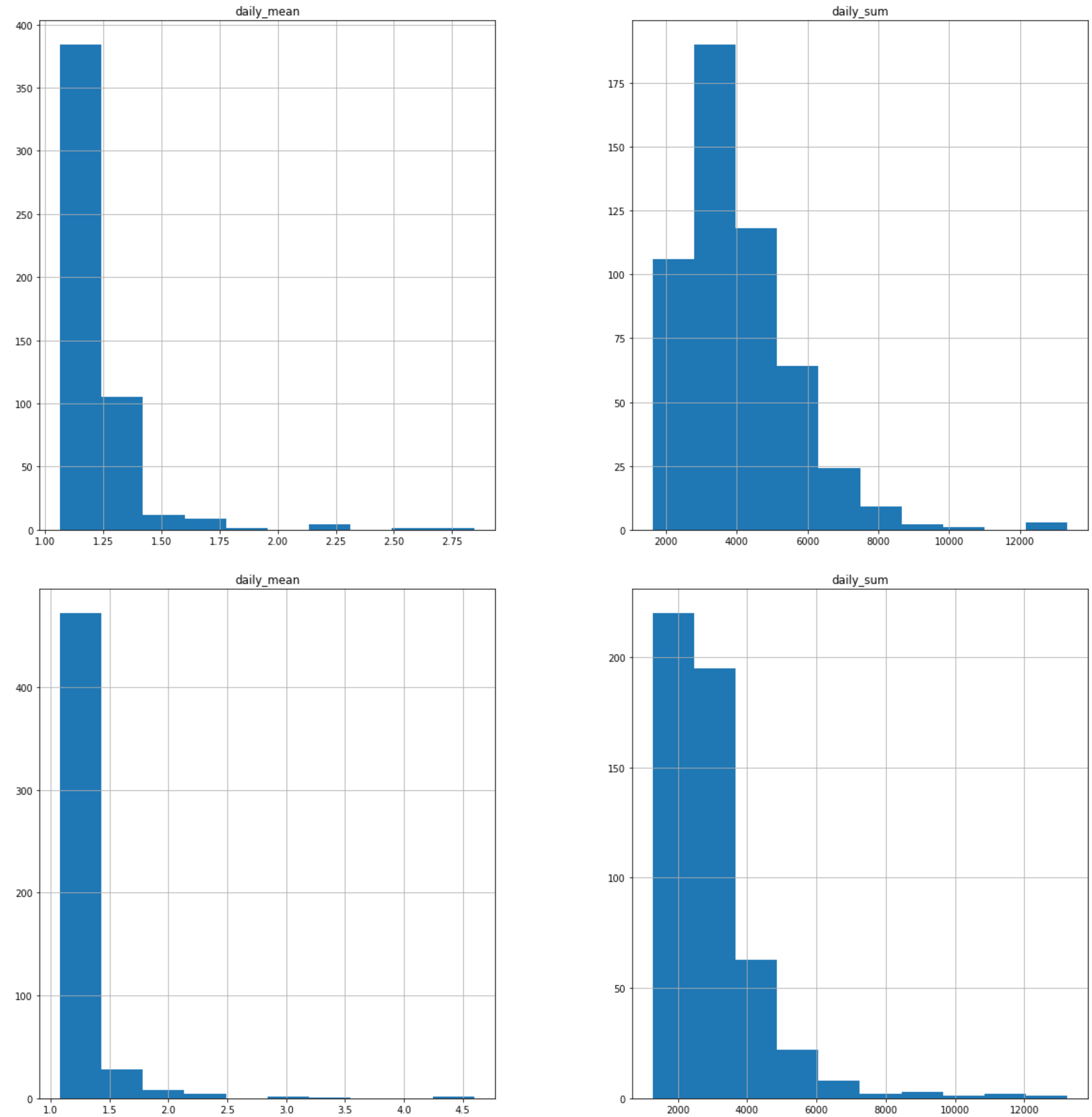
The training set is available as daily sales of the company. Even though the final prediction has to be made in monthly sales, since daily sales data was available data analysis process was carried out on daily sales also in order to identify any increase in demand because of special days such as Valentines Day and Women's Day and to get better insight of the dataset.



Stationarity

Stationarity of a time series is the single most important aspect when considering using statistical models for time series analysis such as ARIMA. Basic meaning of stationarity is when all statistical properties of a time series such as mean, median and variance are constant over time.

In order to check the stationarity, the data set was split into two sections and the mean and variance was calculated for each section and compared.



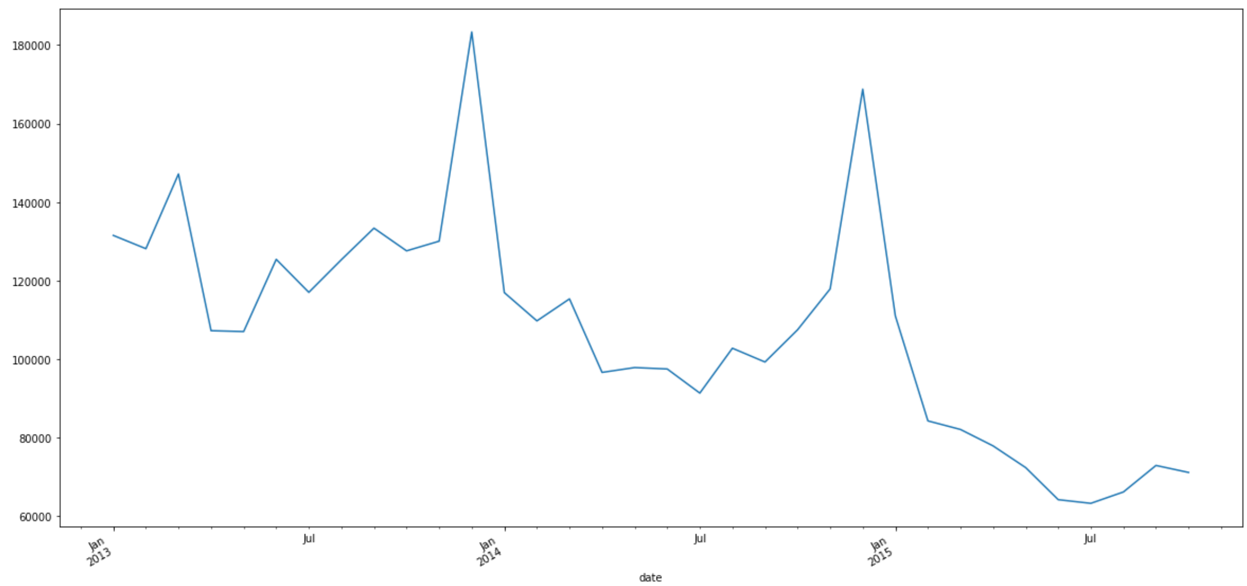

```
Mean Difference:
daily_sum      1080.212766
daily_mean        0.035069
dtype: float64
```

```
Variance Difference:
daily_sum     426388.895266
daily_mean         0.060627
dtype: float64
```

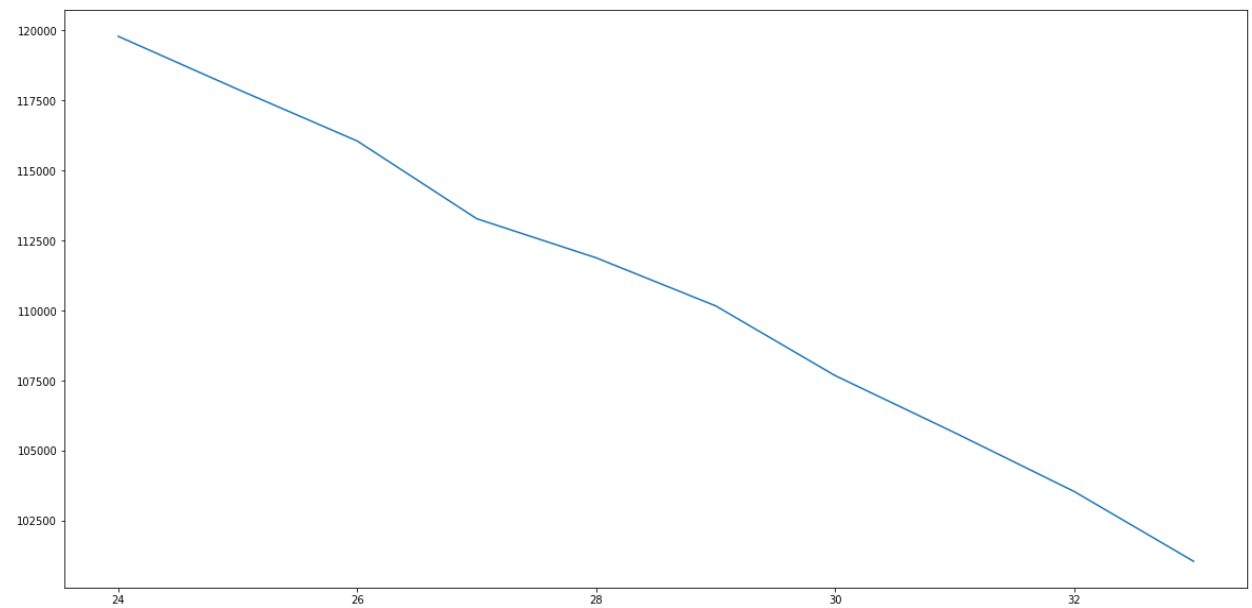
Which provided above results where the conclusion was that the time series is not stationary when looking at daily sales.

Monthly Sales Analysis

The data set was grouped monthly and the sum of item counts per each month was plotted against the number of month in order to identify any seasonality in the data. As seen below, the sales of the company are clearly affected around the November and December months because of the festival season.



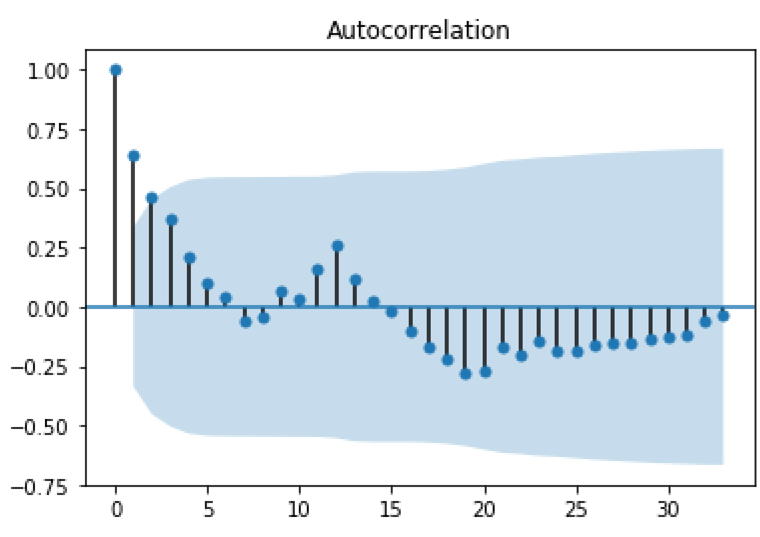
Rolling mean of the monthly sales was plotted against the number of the month in order ro identify the overall trend of sales. The rolling mean window has been set to a higher value in order to produce a smoother curve to remove monthly fluctuations of sales. By analysing the graph, it can be concluded that the sales of the company has a decreasing trend.



Autocorrelation of Monthly Sales

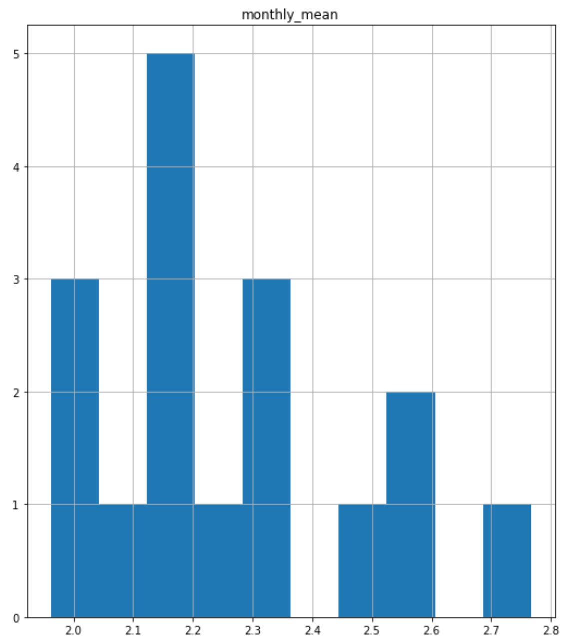
Autocorrelation quantifies internal association of data with fixed time periods apart. For analysing monthly sales, monthly autocorrelation was plotted.

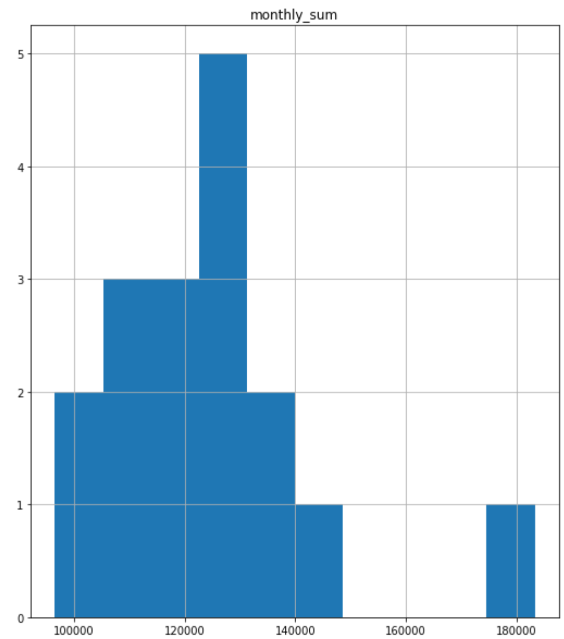
A positive value depicts positive relationship among neighbouring data, a negative value depicts negative relationship and 0 is where no relationship exists among neighbouring data.

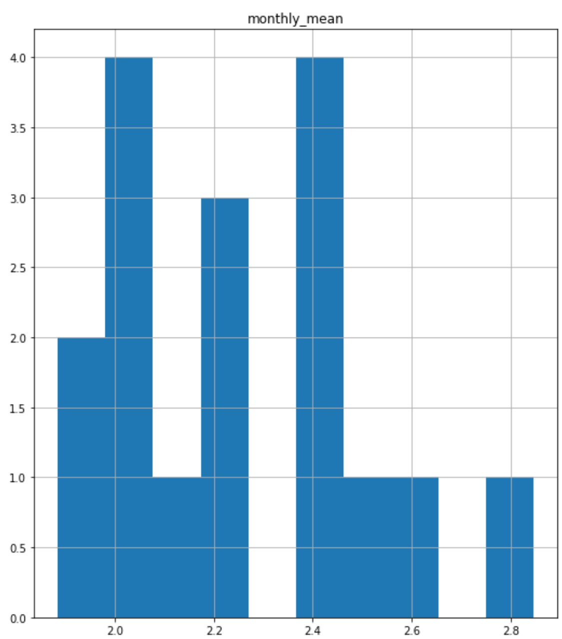


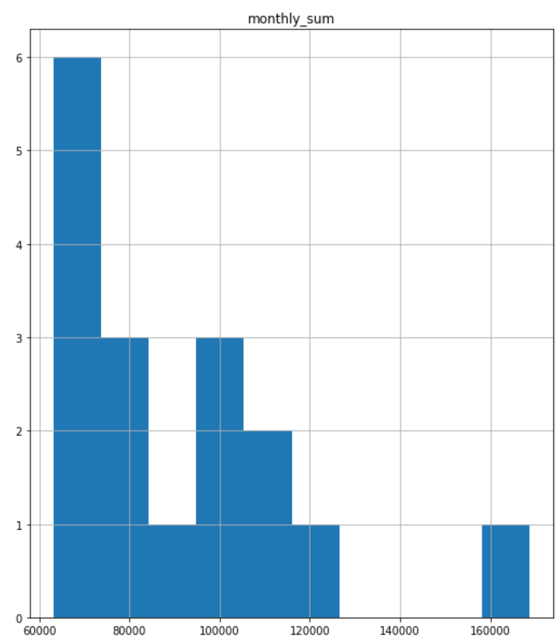
By analysing the plot above, correlation difference of time periods 12 months apart are greater than 0.7, which indicates high correlation of data 12 months apart.

Stationarity









```
Mean:

mean1:
monthly_sum     123468.352941
monthly_mean         2.258329
dtype: float64

mean2:
monthly_sum     91132.000000
monthly_mean        2.254215
dtype: float64
```

```
Variance:

var11:
monthly_sum     4.160894e+08
monthly_mean    4.998568e-02
dtype: float64

var2:
monthly_sum     6.985138e+08
monthly_mean    6.892915e-02
dtype: float64
```

Monthly sales data was also checked for stationarity using the same method as daily sales data. Mean and variance differs significantly between the two time splits considered. Therefore the data is not stationary looking at monthly sales.

Augmented Dicky Fuller test was also carried out to identify the non stationarity of the data.

```
ADF Statistic: -2.395704
p-value: 0.142953
Critical Values:
        1%: -3.646
        5%: -2.954
        10%: -2.616
```

The result was obtained after carrying out ADF for the monthly sum of items for each month, where seasonality and trend was present. The p-value of the test is higher than the accepted range of 5%, which means the dataset is non stationary.

Model Selection

Since our dataset is not ordered by the time (has a composite index), we could not directly apply time series models. It was not possible to create our own ensemble model using time series models due to resource and time constraints.

Even most time series models consider the dataset is stationary where it removes seasonality and trend component from the data to predict the next value. By using ANN we could save the time depended components. But it would consume lots of resources.

Because of this we thought of converting the dataset, so that we could apply supervised learning algorithms with gradient boosting models.

Supervised Models

Rolling Windows-Based Regression

Basically in Rolling windows based regression, if we want to predict X(t+1), next value in a time series, we feed not only X(t), but X(t-1), X(t-2) etc to the model and use a regression algorithm to predict the X(t+1) value.

$$y_t = w\,\Phi(x_t + x[n]_{t-1} + x[n]_{t-2} + x[n]_{t-3} + x[n]_{t-5} + x[n]_{t-12})$$

Time Series models

ARIMA

In statistics and econometrics, and in particular in time series analysis, an autoregressive integrated moving average (ARIMA) model is a generalization of an autoregressive moving average (ARMA) model. Both of these models are fitted to time series data either to better understand the data or to predict future points in the series (forecasting). ARIMA models are applied in some cases where data show evidence of non-stationarity, where an initial differencing step (corresponding to the "integrated" part of the model) can be applied one or more times to eliminate the non-stationarity.

The AR part of ARIMA indicates that the evolving variable of interest is regressed on its own lagged values. The MA part indicates that the regression error is actually a linear combination of error terms whose values occurred contemporaneously and at various times in the past. The

I (for "integrated") indicates that the data values have been replaced with the difference between their values and the previous values (and this differencing process may have been performed more than once). The purpose of each of these features is to make the model fit the data as well as possible.

Non-seasonal ARIMA models are generally denoted ARIMA(p,d,q) where parameters p, d, and q are non-negative integers, p is the order (number of time lags) of the autoregressive model, d is the degree of differencing (the number of times the data have had past values subtracted), and q is the order of the moving-average model. Seasonal ARIMA models are usually denoted ARIMA(p,d,q)(P,D,Q)m, where m refers to the number of periods in each season, and the uppercase P,D,Q refer to the autoregressive, differencing, and moving average terms for the seasonal part of the ARIMA model

SARIMA

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component.

It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality.

Models Comparisons

XGBoost vs LightGBM

- XGBoost is a pre sort based algorithm and has time complexity of O(data)
- LightGBM uses a histogram based model which has a lower time complexity than XGBoost.
- LGBM also requires less memory compared to XGBoost.
- Since we used standard laptop computers in training the model, LGBM was a better fit than XGB.

Boosting Methods vs Neural Networks

- Model generation using neural networks requires huge amount of data points.
- Requires more time and computing resources compared to boosting methods.
- Even though NNs could provide better precision. Boosting was selected because of ease of use and time efficiency.

Model Development

We have used LightGBM model which is a Tree base gradient boosting algorithm.

Gradient boosting involves three elements:

1. A loss function to be optimized.

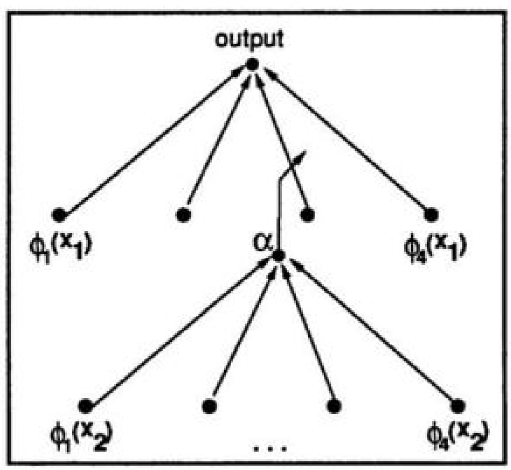
   Loss Function: $L(y,F_m(x)) = \sqrt[n]{\frac{1}{n}\sum_{i}^{n}(y - Fm(x))^2}$

2. A weak learner to make predictions.

   We preprocess the dataset according to Rolling Windows-based Regression,

   Where, $y_t = w\,\Phi(x_t + x[n]_{t-1} + x[n]_{t-2} + x[n]_{t-3} + x[n]_{t-5} + x[n]_{t-12})$

   $y' = w\,\Phi(x) = F_m(x)$



3. An additive model to add weak learners to minimize the loss function.

   for $m=1 \rightarrow M$

   $F_m(x) = F_{m-1}(x) + \alpha\, h_m(x)$

Data Preprocessing

Feature Engineering

Downcasting Variables

Sales data

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day
0	2013-01-02	0	59	22154	999.00	1
1	2013-01-03	0	25	2552	899.00	1
2	2013-01-05	0	25	2552	899.00	-1
3	2013-01-06	0	25	2554	1709.05	1
4	2013-01-15	0	25	2555	1099.00	1

Step 1 - Downcast data (in sales) to manipulate the memory productively.

```
sales.info()
```

```
<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
```

```
sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2935849 entries, 0 to 2935848
Data columns (total 6 columns):
date              datetime64[ns]
date_block_num    int8
shop_id           int8
item_id           int16
item_price        float64
item_cnt_day      int16
dtypes: datetime64[ns](1), float64(1), int16(2), int8(2)
memory usage: 61.6 MB
```

Item Data

	item_name	item_id	item_category_id
0	! Ð□Ð□ Ð□Ð□Ð□Ð¡Ð¢Ð□ Ð□Ð□Ð□Ð□Ð□Ð□Ð□Ð□Ð¯ (Ð□Ð□...	0	40
1	!ABBYY FineReader 12 Professional Edition Full...	1	76
2	***Ð□ Ð□Ð£Ð§Ð□Ð¥ Ð¡Ð□Ð□Ð□Ð« (UNV) ...	2	40
3	***Ð□Ð□Ð□Ð£Ð□Ð□Ð¯ Ð□Ð□Ð□Ð□Ð□ (Univ) ...	3	40
4	***Ð□Ð□Ð Ð□Ð□Ð□Ð□ (Ð¡Ð¢Ð□Ð□Ð□Ð□) ...	4	40

Step 2 - Downcast data(in items)

```
items.info()
```

```
<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
```

```
items.info()
```

```
<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 int16
item_category_id      22170 non-null int8
dtypes: int16(1), int8(1), object(1)
memory usage: 238.2+ KB
```

Merging Features

Step 3 - merging sales data and item data. (training day data set)

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_category_id
0	2013-01-02	0	59	22154	999.00	1	37
1	2013-01-03	0	25	2552	899.00	1	58
2	2013-01-05	0	25	2552	899.00	-1	58
3	2013-01-06	0	25	2554	1709.05	1	58
4	2013-01-15	0	25	2555	1099.00	1	56

Add Window Features

Converting Daily Sales into Monthly sales.

Step 4 - data group by month

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	item_category_id
0	2013-01-01	0	59	22154	999.00	1	37
1	2013-01-01	0	25	2552	899.00	1	58
2	2013-01-01	0	25	2552	899.00	-1	58
3	2013-01-01	0	25	2554	1709.05	1	58
4	2013-01-01	0	25	2555	1099.00	1	56

Step 5 - Groupby data to analyse the item count for a month (training month data set)

	date_block_num	shop_id	item_id	item_price	item_cnt_month
0	0	0	32	221.0	6
1	0	0	33	347.0	3
2	0	0	35	247.0	1
3	0	0	43	221.0	1
4	0	0	51	128.5	2

Step 6 - merging training day data set with training month data set

	shop_id	item_id	date_block_num	date	item_category_id	item_price	item_cnt_month
0	59	22154	0	2013-01-01	37	999.00	1
1	25	2552	0	2013-01-01	58	899.00	0
2	25	2552	0	2013-01-01	58	899.00	0
3	25	2554	0	2013-01-01	58	1709.05	1
4	25	2555	0	2013-01-01	56	1099.00	1

Step 7 - drop duplicates on training month data set using month, item and shop fields.

	shop_id	item_id	date_block_num	date	item_category_id	item_price	item_cnt_month
0	59	22154	0	2013-01-01	37	999.00	1
1	25	2552	0	2013-01-01	58	899.00	0
3	25	2554	0	2013-01-01	58	1709.05	1
4	25	2555	0	2013-01-01	56	1099.00	1
5	25	2564	0	2013-01-01	59	349.00	1

Step 8 - creating zero sales for every item which did not sale

	shop_id	item_id	date_block_num
0	59	22154	0
1	59	2552	0
2	59	2554	0
3	59	2555	0
4	59	2564	0

Step 9 - merging zero sales with training month data set and dropping category and price fields(all data)

	shop_id	item_id	date_block_num	date	item_cnt_month
0	59	22154	0	2013-01-01 00:00:00	1.0
1	59	2552	0	0	0.0
2	59	2554	0	0	0.0
3	59	2555	0	0	0.0
4	59	2564	0	0	0.0

Add Missing Values

Step 14 - Assume item price is differed from the date. Add prices to Zero sales. Merging all data with training month data.

	shop_id	item_id	date_block_num	date	item_cnt_month	item_price
0	59	22154	0	2013-01-01	1	999.0
1	59	22154	0	2013-01-01	1	999.0
2	59	22154	0	2013-01-01	1	999.0
3	59	22154	0	2013-01-01	1	999.0
4	59	22154	0	2013-01-01	1	999.0

```
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75218848 entries, 0 to 75218847
Data columns (total 6 columns):
shop_id           int8
item_id           int16
date_block_num    int8
date              datetime64[ns]
item_cnt_month    int16
item_price        float16
dtypes: datetime64[ns](1), float16(1), int16(2), int8(2)
memory usage: 1.7 GB
```

Add Rolling Window Features (Sum and Mean)

Step 10 - Get shop wise data for a month by grouping date and shop

	date_block_num	shop_id	shop_block_target_sum	shop_block_target_mean
0	0	0	5578	0.687369
1	0	1	2947	0.363155
2	0	2	1146	0.141220
3	0	3	767	0.094516
4	0	4	2114	0.260505

Step 11 - Get item wise data for a month by grouping date and item

	date_block_num	item_id	item_block_target_sum	item_block_target_mean
0	0	19	1.0	0.022222
1	0	27	7.0	0.155556
2	0	28	8.0	0.177778
3	0	29	4.0	0.088889
4	0	32	299.0	6.644444

Step 12 - merging item wise data with all data

	shop_id	item_id	date_block_num	date	item_cnt_month	item_category_id
0	59	22154	0	2013-01-01	1.0	37
1	59	2552	0	2013-01-01	0.0	58
2	59	2554	0	2013-01-01	0.0	58
3	59	2555	0	2013-01-01	0.0	56
4	59	2564	0	2013-01-01	0.0	59

Step 13 - Get item-category wise data for a month by grouping date and item-category

	date_block_num	item_category_id	item_cat_block_target_sum	item_cat_block_target_mean
0	0	0	1.0	0.022222
1	0	1	1.0	0.022222
2	0	2	1390.0	0.834835
3	0	3	440.0	4.888889
4	0	4	251.0	0.507071

Step 15 - merging all data with items

	shop_id	item_id	date_block_num	date	item_cnt_month	item_price	item_category_id
0	59	22154	0	2013-01-01	1	999.0	37
1	59	22154	0	2013-01-01	1	999.0	37
2	59	22154	0	2013-01-01	1	999.0	37
3	59	22154	0	2013-01-01	1	999.0	37
4	59	22154	0	2013-01-01	1	999.0	37

Step 16 - merging all data with item-category

	shop_id	item_id	date_block_num	date	item_cnt_month	item_price	item_category_id	item_cat_block_target_sum	item_cat_block_target_m
0	59	22154	0	2013-01-01	1	999.0	37	6094.0	0.199738
1	59	22154	0	2013-01-01	1	999.0	37	6094.0	0.199738
2	59	22154	0	2013-01-01	1	999.0	37	6094.0	0.199738
3	59	22154	0	2013-01-01	1	999.0	37	6094.0	0.199738
4	59	22154	0	2013-01-01	1	999.0	37	6094.0	0.199738

Step 17 - merging all data with item.

	shop_id	item_id	date_block_num	date	item_cnt_month	item_price	item_category_id	item_cat_block_target_sum	item_cat_block_target_m
0	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
1	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
2	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
3	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
4	59	22154	0	2013-01-01	1	999.0	37	37	0.199707

Step 18 - merging all data with shop data.

	shop_id	item_id	date_block_num	date	item_cnt_month	item_price	item_category_id	item_cat_block_target_sum	item_cat_block_target_m
0	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
1	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
2	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
3	59	22154	0	2013-01-01	1	999.0	37	37	0.199707
4	59	22154	0	2013-01-01	1	999.0	37	37	0.199707

Preprocessed Data

```
all_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 75218848 entries, 0 to 75218847
Data columns (total 13 columns):
shop_id                       int8
item_id                       int16
date_block_num                int8
date                          datetime64[ns]
item_cnt_month                int16
item_price                    float16
item_category_id              int8
item_cat_block_target_sum     int8
item_cat_block_target_mean    float16
item_block_target_sum         int16
item_block_target_mean        float16
shop_block_target_sum         int16
shop_block_target_mean        float16
dtypes: datetime64[ns](1), float16(4), int16(4), int8(4)
memory usage: 2.5 GB
```

	shop_id	item_id	date_block_num	date	item_cnt_month	item_price	item_category_id	item_cat_block_target_sum
0	59	22154	0	2013-01-01	1	999.0	37	37
1	59	22154	0	2013-01-01	1	999.0	37	37
2	59	22154	0	2013-01-01	1	999.0	37	37
3	59	22154	0	2013-01-01	1	999.0	37	37
4	59	22154	0	2013-01-01	1	999.0	37	37

item_cat_block_target_mean	item_block_target_sum	item_block_target_mean	shop_block_target_sum	shop_block_target_mean
0.199707	18	0.399902	2017	0.248535
0.199707	18	0.399902	2017	0.248535
0.199707	18	0.399902	2017	0.248535
0.199707	18	0.399902	2017	0.248535
0.199707	18	0.399902	2017	0.248535

Add Lag Features

1, 2, 3, 5 months was selected in order to reflect the overall decreasing trend of the sales of the company and to reflect the effect of recent events in the most recent quarter to the month being considered.
12th month lag feature was selected to reflect the annual seasonality of the sales.

Add Seasonal Components/ Holidays

Cross Validation and Hyperparameter Tuning

We created 6 folds of validation sets using last 6 months 28 to 33 and calculated the root mean squared error per each fold. We used the mean of all the 6 folds as the metric to improve hyperparameters.

Hyperparameter tuning is done by using hyperopt python library which we tried to tune the following parameter

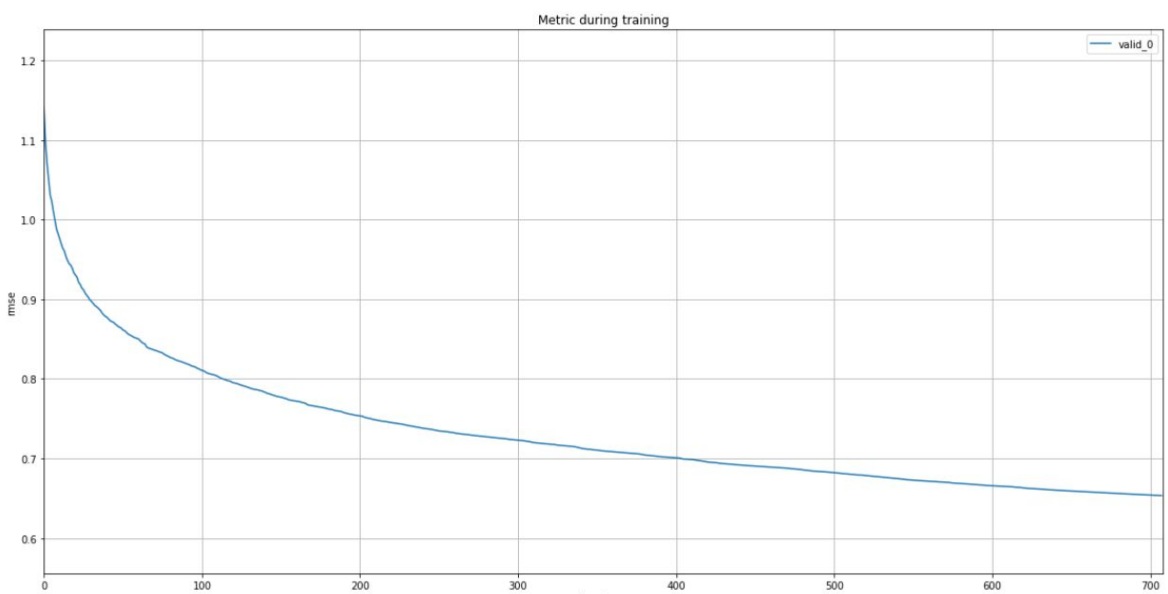
Hyperparameters for LightGBM

- 'colsample_bytree': 0.8
  - Percentage of columns used per iteration.
- 'metric': 'rmse'
  - Evaluation metrics to be monitored through iterations.
- 'min_data_in_leaf': 21
  - It is the minimum number of the records a leaf may have.
- 'subsample': 0.55
  - Subsample ratio of the training instance.
- 'learning_rate': 0.225
  - List of learning rates for each boosting round or a customized function that calculates.

- 'objective': 'regression'
    - Final objective of what method used for prediction.
- 'bagging_seed': 128
    - Random seed for bagging.
- 'num_leaves': 7
    - Maximum tree leaves for base learners.
- 'max_depth': 3
    - It describes the maximum depth of tree. This parameter is used to handle model overfitting. Any time you feel that your model is overfitted.
- 'bagging_freq': 1
    - Turning on bagging.
- 'seed': 1204
    - Seed used to generate the weights

Regularization

We have not used a specific regularization term in the model but we controlled overfitting by tuning hyperparameters such as reducing the depth of the tree, number of leaves, subsampling and choosing set of columns per iteration. Some of these values are taken from recommendations such as number of leaves and depth and some of the values are taken from hyper tuning the parameters such as subsample size. By addressing the overfitting issue we were able to reduce the 'root mean square error' (rmse) of testing dataset from 4.2rmse to 2.1rmse.



References

[1]"anhquan0412/Predict_Future_Sales", *GitHub*, 2018. [Online]. Available: https://github.com/anhquan0412/Predict_Future_Sales. [Accessed: 19- Sep- 2018]

[2]J. Brownlee, "7 Time Series Datasets for Machine Learning", *Machine Learning Mastery*, 2018. [Online]. Available: https://machinelearningmastery.com/time-series-datasets-for-machine-learning/. [Accessed: 19- Sep- 2018]

[3]T. Sanger, *Basis-Function Trees as a Generalization of Local Variable Selection Methods for Function Approximation*. 2018 [Online]. Available: https://papers.nips.cc/paper/340-basis-function-trees-as-a-generalization-of-local-variable-selection-methods-for-function-approximation.pdf. [Accessed: 19- Sep- 2018]

[4]"ScienceDirect.com | Science, health and medical journals, full text articles and books.", *Sciencedirect.com*, 2018. [Online]. Available: https://www.sciencedirect.com/. [Accessed: 19- Sep- 2018]

[5]G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree", 2018 [Online]. Available: https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf. [Accessed: 19- Sep- 2018]