## Table of Contents



## Introduction

The primary objective of this project is to provide students of the Coursera course "How to win a data science competition" with a practical experience of applying machine learning algorithms. The challenge, which serves as the final project for the course, entails predicting the total

sales for every store and product in the next month. The dataset provided for this competition is a challenging time-series dataset that contains daily sales information. This dataset was generously made available by one of the largest software firms in Russia, 1C Company.

The main task for participants of this competition is to develop a machine learning model that can accurately predict the total sales for each store and product in the upcoming month. This task requires applying advanced machine learning techniques to the time-series data and making accurate predictions based on patterns observed in historical sales data. It also involves identifying relevant features in the dataset that can be used to train the machine learning model effectively.

Overall, this project presents a unique opportunity for students to apply their knowledge of machine learning algorithms to a real-world problem and gain valuable experience in data science competitions.

## Data Gathering

This dataset has two set of data given. One is originally given by the company, and it is there on Kaggle. The names of cities, item

categories, product categories are in Russian in this dataset and hence it becomes difficult for people who don't know Russian to understand the data.

Hence, there is another dataset given where Russian names are translated to English.

The links of the dataset are given below.

**Dataset with Russian Names**

Predict Future Sales | Kaggle

**Dataset with English Translated Names**


**This is the code snippet of the files read from both the sources of the data.**

```
item_categories = pd.read_csv("/content,
items = pd.read_csv("/content/predict-fu
train_data = pd.read_csv("/content/compe
test_data = pd.read_csv("/content/compe
shops = pd.read_csv("/content/predict-fu
submission = pd.read_csv("/content/compe
```

This is a Russian dataset of a Walmart type mega store which has its outlet in almost all cities of Russia.

**The dataset is divided into various parts.**

1. **train_data.csv:** It has feature names like date, date_block_num, shop_id, item_id, item_price and item_count
2. **item_categories.csv:** It has feature names like item_category_name, item_category_id
3. **items.csv:** It has feature names like item_name, item_id, item_category_id
4. **shops.csv:** It has feature names like shop_name, shop_id
5. **test_data.csv:** It has feature names like shop_id,item_id
6. **submission.csv:** It has the information for the format in which the data should be submitted.

**Column Description**

- 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_name — name of item
- shop_name — name of shop
- item_category_name — name of item category

## Data Preparation

For preparation of the data, different csv files are to be merged to form one single data frame which can be used for training.

```
a = len(train_data['shop_id'].unique())
b = len(test_data['shop_id'].unique())

print(f'The number of unique shops in Train Data are: {a}')
print(f'The number of unique shops in Test Data are: {b}')
```

```
The number of unique shops in Train Data are: 60
The number of unique shops in Test Data are: 42
```

As seen from the above code snippet, the number of shops in train data are more than the number of shops in the test data. So, we will train only those shops which are present in Test Data.

### City Names

City names are appended at the initial part of shops names. Below is the code to extract the city names from the shops and hence we can get additional feature names.

| | shop_name | shop_id | city_name | city_id |
|---|---|---|---|---|
| 0 | ! Yakutsk Ordzhonikidze, 56 Franc | 0 | Yakutsk | 28 |
| 1 | ! Yakutsk TC "Central" Franc | 1 | Yakutsk | 28 |
| 2 | Adygea TC "Mega" | 2 | Adygea | 0 |
| 3 | Balashikha TRC "October-Kinomir" | 3 | Balashikha | 1 |
| 4 | Volzhsky mall "Volga Mall" | 4 | Volzhsky | 26 |

City names after extraction from Shop Names

### Product Categories

Product Categories can be extracted from item category names.

| | item_category_name | item_category_id | Product_Category | Product_Category_id |
|---|---|---|---|---|
| 0 | PC - Headsets / Headphones | 0 | PC | 10 |
| 1 | Accessories - PS2 | 1 | Accessories | 0 |
| 2 | Accessories - PS3 | 2 | Accessories | 0 |
| 3 | Accessories - PS4 | 3 | Accessories | 0 |
| 4 | Accessories - PSP | 4 | Accessories | 0 |

**Finally, all the CSV files should be merged.**

1. item and **item_category_id** CSVs are merged.
2. **train_data** and shops CSVs are merged.

And above two final CSVs are merged to form 1 single data frame which we will use for training.

## Evaluation Criteria

### RMSE

Root mean squared error (RMSE) is a widely

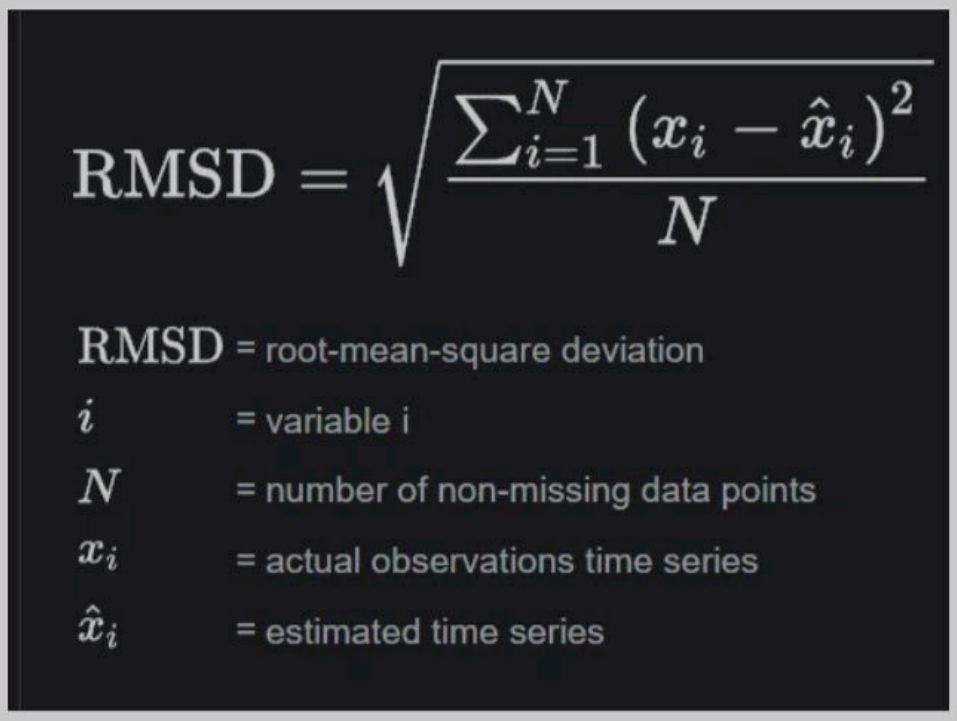


used statistical measure that is used to evaluate the accuracy of a predictive model. It is a measure of the difference between predicted and actual values, also called residuals. The RMSE formula calculates the simple standard deviation of these residuals by taking the square root of the average of the squared differences between predicted and actual values.

The RMSE is useful because it provides a single metric that summarizes the overall error of a model. The lower the RMSE, the better the model's predictive accuracy. The RMSE is commonly used in regression analysis, time series forecasting, and machine learning applications.

One advantage of RMSE is that it punishes large errors more severely than small errors. This means that a model with a few large errors will have a higher RMSE than a model with many small errors, even if the total error is the same. Another advantage is that it is easy to interpret and communicate to non-technical stakeholders.


However, the RMSE has some limitations. It assumes that errors are normally distributed and independent, which may not be the case in all situations. Additionally, it can be sensitive to outliers and may not capture all aspects of model performance. Therefore, it is important to use the RMSE in combination with other metrics and visualizations to gain a complete understanding of a model's strengths and weaknesses.



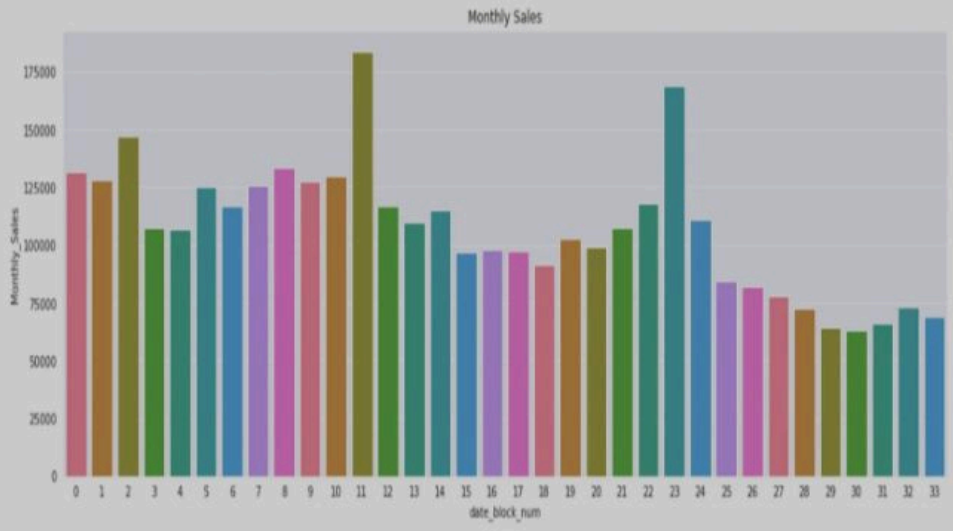

Wikipedia

Note:- RMSD is nothing but another name for RMSE

## Exploratory Data Analysis

The data given is for 33 months. We have to predict the sales of 34th month.

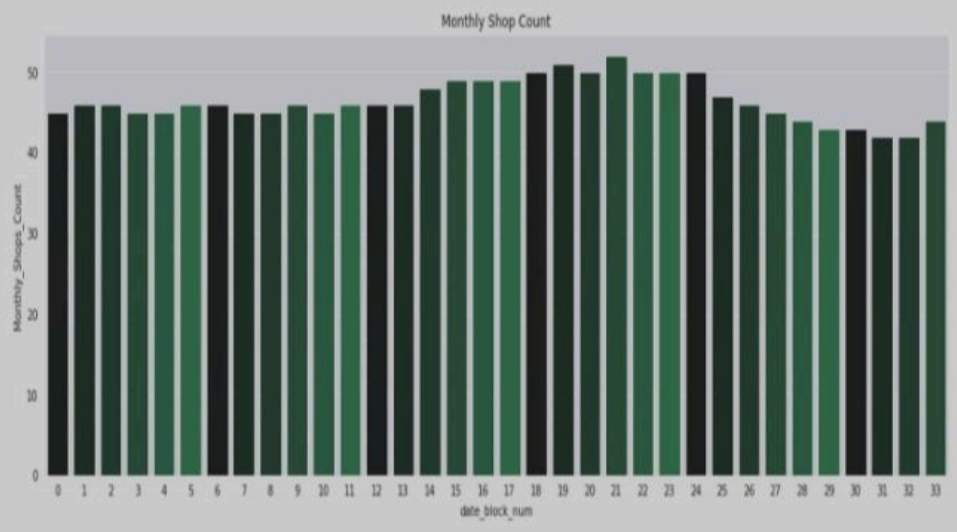
**Total sales per month for all the items taken together.**




Total Sales per month

For monthly sales data, as we can see in the above graph, there are two sales peaks in monthly sales.

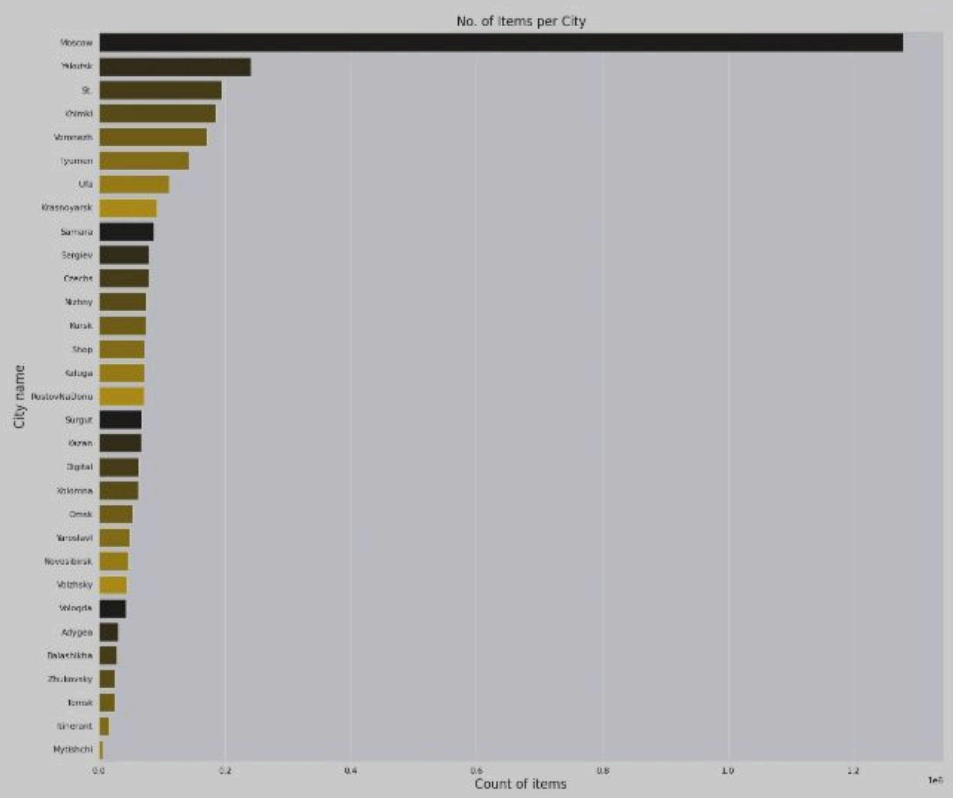
**Total Number of Shops Active per month**




As you can see in the above graph, where number of shops active per month hovers around 40–50. Number of shops peaked around the month 19,20,21. Lowest count of

active shops is in month 31,32.

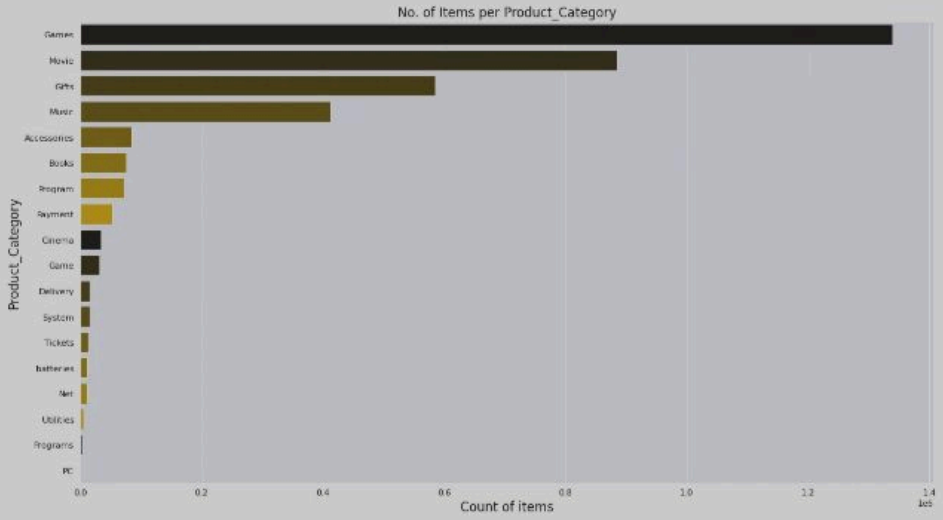
**Total Number of items sold per city are.**




Total items sold per city.

**Moscow** being the biggest city of Russia has the highest items sold and it is more than thrice the sales of the city names **Yakutsk.**

**Number of items sold per Product Category**



responsible for 90% of the sales.

## Feature Engineering

Total number of item counts sold every month for every shop differs. But for training we will need constant and same number of item counts for every shop every month. So, we take help of **itertools** and we create same number of items for every shop every month.

This number of items for every shop may vary for another month, but for same month total number of items for every shop has to be same.

For that we create a new data frame and for every month the respect active shops as shown in the train data and for every shop, we take total items sold and make them same for all the shops of that particular month.

**Hence our final data frame becomes something like this.**

responsible for 90% of the sales.

## Feature Engineering

Total number of item counts sold every month for every shop differs. But for training we will need constant and same number of item counts for every shop every month. So, we take help of **itertools** and we create same number of items for every shop every month.

This number of items for every shop may vary for another month, but for same month total number of items for every shop has to be same.

For that we create a new data frame and for every month the respect active shops as shown in the train data and for every shop, we take total items sold and make them same for all the shops of that particular month.

**Hence our final data frame becomes something like this.**

- Average item count per month for every item id and every shop

2. Add Mean Features

- Create Item means
- Create Shop means
- Create Item and Shop means

3. Add Lag Features

- Adding 3 months of lag to Mean Features and Average Count Features

**In total we have created 41 features.**

## Model Development:

Following models are used to test the accuracy:-

1. Random Forest Regressor
2. Linear Regression
3. XGBoost
4. Bagging Regressor
5. Light GBM
6. XGBoost on Stacking
7. Light GBM on Stacking

**gamma**: the minimum loss reduction required to make a split. In this case, it is set to 0.1.
**min_child_weight**: the minimum sum of instance weight needed in a child. In this case, it is set to 7.
**silent**: whether to print messages during training. In this case, it is set to False, meaning that messages will be printed.
**validate_parameters**: whether to validate the input parameters. In this case, it is set to True.
**max_depth**: the maximum depth of the tree. In this case, it is set to 3.

```
RMSE value is: 0.8690382993662482
```

**Bagging Regressor:**

- It is a powerful ensemble learning technique that combines multiple models to improve prediction accuracy and reduce variance.
- It works by training each model on a random subset of the data, then combining their outputs to make a final prediction.
- It can be used with a variety of models, including decision trees, support vector

machines, and neural networks.

```
from sklearn.ensemble import BaggingRegr

bag_regressor = BaggingRegressor(n_estir

bag_regressor = bag_regressor.fit(X_tra
```

The parameters are as follows:

**n_estimators**: This parameter specifies the number of base estimators in the ensemble. Each estimator will be trained on a random subset of the data. A higher number of estimators will generally improve the stability and accuracy of the model, but also increase the training time and memory usage.

**random_state**: This parameter initializes the random number generator used to select the subsets of the data for each estimator. By fixing the random state, the results of the model will be reproducible.

**max_samples**: This parameter specifies the maximum number of samples to draw from the training set for each base estimator. Each estimator will be trained on a different subset of the data, randomly drawn with

control the amount of randomness in the model, with smaller values leading to more variance in the ensemble.

Bagging Regressor is an ensemble method that uses bootstrap sampling to create multiple base estimators, fits each estimator on a different random subset of the training data, and then combines the predictions of each estimator to form the final prediction. This results in a more robust and less prone to overfitting model compared to using a single estimator.

```
RMSE value is: 0.9447815211138822
```

**Light GBM:**

- It is a fast and efficient gradient boosting algorithm that uses a tree-based approach to handle large datasets with high dimensionality.
- It offers several advanced features such as categorical feature handling, early stopping, and GPU acceleration.

```
import lightgbm as lgb

params = {'metric': 'rmse',
          'num_leaves': 255,
          'learning_rate': 0.005
```

```
                'bagging_freq': 5,
                'force_col_wise' : True,
                'random_state': 10,
               'num_rounds':600,
               'early_stopping':150}

    lgb_train = lgb.Dataset(X_train, y_trai

    lgb_val = lgb.Dataset(X_val, y_val)

    model = lgb.train(params=params, train_
```

The detailed explanation of each parameter is as mentioned below:

**metric**: This parameter specifies the evaluation metric to be used during training. In this case, it is set to 'rmse', which stands for root mean squared error.

**num_leaves**: This parameter controls the complexity of the model by setting the maximum number of leaves in each tree. A higher value will lead to a more complex model with higher accuracy but may also result in overfitting. In this case, it is set to 255.

**learning_rate**: This parameter controls the step size during training. A smaller value will result in a slower learning rate but may improve accuracy and generalization. In this case, it is set to 0.005.

**learning_rate**: This parameter controls the step size during training. A smaller value will result in a slower learning rate but may improve accuracy and generalization. In this case, it is set to 0.005.

**feature_fraction**: This parameter controls the fraction of features to be used in each tree. A smaller value will lead to a simpler model and may reduce overfitting. In this case, it is set to 0.75.

**bagging_fraction**: This parameter specifies the fraction of data to be randomly sampled for each tree. A smaller value will lead to a more robust model with less overfitting. In this case, it is set to 0.75.

**bagging_freq**: This parameter specifies how often to perform bagging. A higher value will lead to more randomness in the model, which may improve generalization. In this case, it is set to 5.

**force_col_wise**: This parameter specifies whether to use column-wise tree building, which can improve efficiency and reduce memory usage. In this case, it is set to True.

**random_state**: This parameter initializes the random number generator used by the

model, which can be used to ensure reproducibility of results.

**num_rounds**: This parameter specifies the number of boosting rounds, which corresponds to the number of trees in the model. A higher value will lead to a more complex model with higher accuracy but may also result in overfitting. In this case, it is set to 600.

**early_stopping**: This parameter specifies the number of rounds without improvement in the validation metric to trigger early stopping. This can be used to prevent overfitting and reduce training time. In this case, it is set to 150.

```
valid_1's rmse: 0.857906
```

## Kaggle Performance

| # | Team | Members | Score | Entries | Last | Join |
|---|---|---|---|---|---|---|
| 1068 | Shivang Shrivastav | | 0.85608 | 22 | 3mo | |
| 1092 | CS567_TheMeanSquares | | 0.85832 | 69 | 3y | |

1093 - 16160 ⇕ See 15068 More

Kaggle Screenshot

As you can see, the sequence number of my accuracy is 1068 and there is total 16160 entries.
Hence, 1068/16160 = 6.60%

## Conclusion and Future Research

LightGBM's Gradient-based One-Side Sampling (GOSS) technique speeds up training by selectively keeping data instances with large gradients and randomly sampling those with small gradients. XGBoost, which uses a pre-sorted or histogram-based algorithm to determine the best feature split, grows level-wise and creates larger models than LightGBM's leaf-wise growth. LightGBM's selective tree growth produces smaller and faster models compared to XGBoost.

While both are asymmetric trees, LightGBM's selective tree growth and GOSS technique make it a faster and more efficient choice and also helps in giving good RMSE Score.

### Future Research

Since this is a time series data, LSTM and Encoder based Transformer can be used and the RMSE values can be checked.

### GitHub Link

Shivang-

Srivastav/colab_notebook_predict_future_sales: Colab notebooks of the competition Predict Future Sales (github.com)

## References

Feature engineering, xgboost | Kaggle


https://github.com/Ksyuwish/PredictFutureSales_EDA/blob/main/project/predict-future-sales-eda.ipynb

Xgboost Lgbm Kaggle

Random Forest Time Series Analysis