NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4000/CZ4041 - Machine Learning

Project Report: Sberbank Russian Housing Market

Group No: 8

Name	Matric No.	Contribution
Dang Huy Phuong	U2120380G	Models experiments, report writing, and presentation
Hoang Minh Trung	U2020430G	Hyperparameter tuning, report writing, and presentation.
Nguyen Tung Bach	U2120390E	Data exploration, report writing, and presentation.
Pham Minh Quan	U2140810C	Models experiments, report writing, and presentation
Vu Duc Anh	U2020241B	Models experiments, report writing, and presentation

Table of Contents

1. Problem Statement & Challenges

The Russian housing market is one of the most dynamic and volatile in the world, influenced by various economic, political, and social factors. Accurately predicting the realty prices in Russia can help investors, home buyers, and policymakers make informed decisions and optimize their outcomes. However, predicting the realty prices in Russia is challenging due to the lack of reliable and comprehensive data, as well as the complexity and diversity of the market conditions.

To achieve this goal, they organized a competition on Kaggle, inviting participants to predict accurate realty prices based on housing characteristics. The training dataset encompassed both real prices and housing attributes spanning from August 2011 to June 2015, while the test set only included housing features from July 2015 to May 2016, along with corresponding macroeconomic data. The assessment of predictions was conducted using the Root Mean Square Log Error (RMSLE) to measure the disparity between predicted and actual prices.

One of the primary challenges in this competition pertains to data. There is a vast number of available features, and some of them, particularly those in the macro dataset, may not be pertinent to the problem at hand. Additionally, some features contain problematic values that must be addressed before we can begin modeling. Another significant hurdle involves determining the most appropriate model type to employ. Given that this is a regression problem, there are numerous techniques at our disposal, and it's initially uncertain which one would be the most suitable for solving this problem.

2. Data Exploration

2.1. Dataset Overview

The dataset consists of the training and testing set, *train.csv* and *test.csv* respectively, which include our target variable *price_doc* - the sale price of the units and the features of each unit such as living area, build year, and general features about the location of the unit and its neighborhood. In total, there are 30471 and 7662 rows in the training and testing set respectively, with 291 starting features.

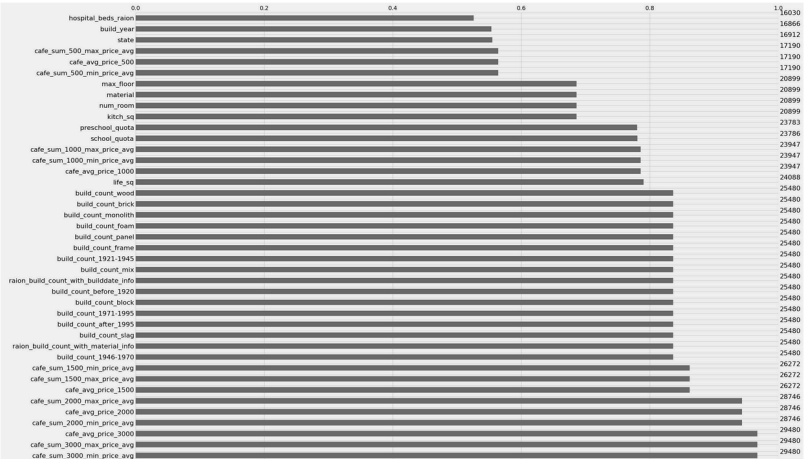


Figure 1: Columns with missing values in the Sberbank dataset.

Among the 292 columns, we identify 51 columns that contain missing values. The quantity of missing values in these columns varies, spanning from 25 to 14,441. This discovery will have an impact on the feature engineering process outlined in section 3.1.

2.2. Data exploration of the features

We conduct correction analysis to detect the correlation between feature columns, resulting in the decision to perform methods such as feature extraction or feature engineering. As shown in Figure 2, the correlation analysis chart reveals a diverse range of relationships between variables, with a predominant pattern of correlations clustering around zero. This implies that changes in one variable are not consistently associated with proportional changes in another.

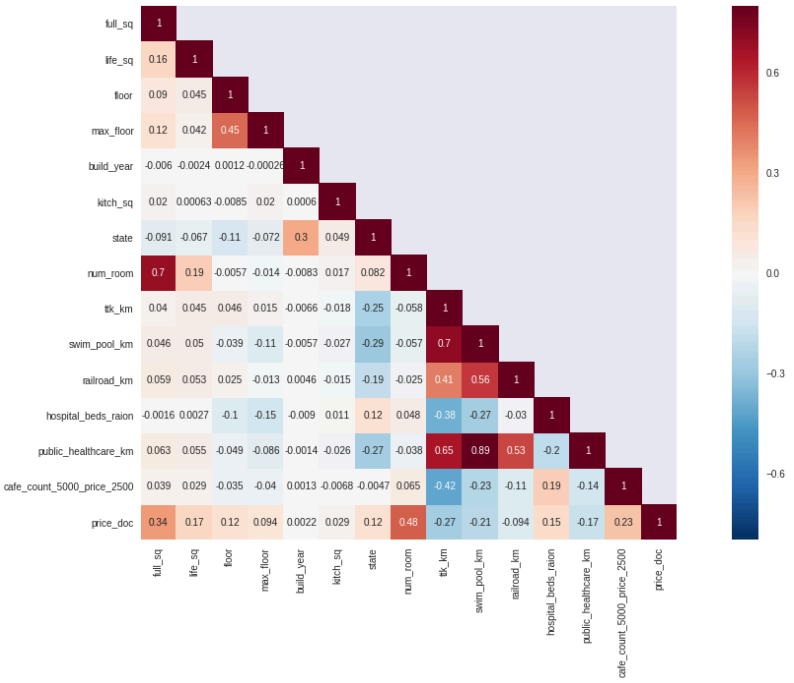


Figure 2: Correlation between the first 14 columns and the target column

We also conduct experiments to identify important features using different simple machine learning models such as XGBoost and Random Forest. Empirical experiment shows that "full_sq" and "railroad_km" are often crucial contributors to the models' predictive performance. This gave us a better insight to enhance our approach in the preprocessing and model proposition stages.

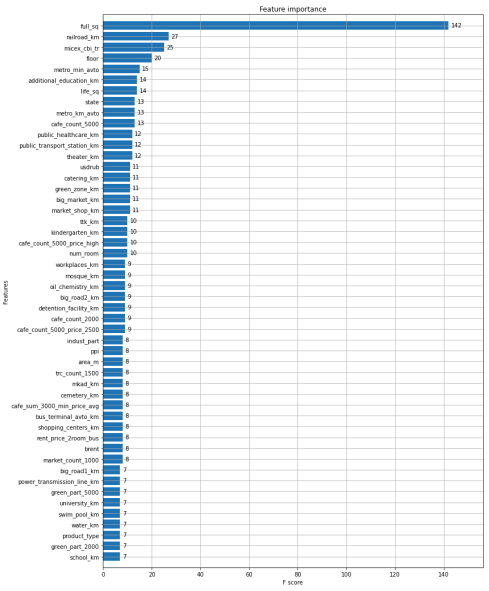


Figure 3: Feature importance using XGBoost model

2.3. Data exploration of the target variable

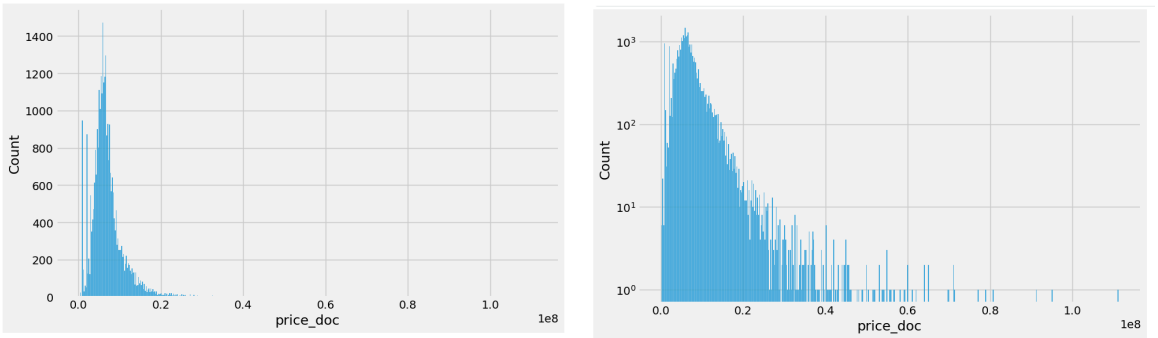


Figure 4: Distribution of target variable - *price_doc*

As shown in Figure 4, we notice that while most of the data is concentrated in the 0.0 to 0.2e8 range, there is a significant presence of *price_doc* outliers toward the higher end of the value spectrum. The central tendency in this specific range may result in some issues such as outliers and skewed distribution when training models.

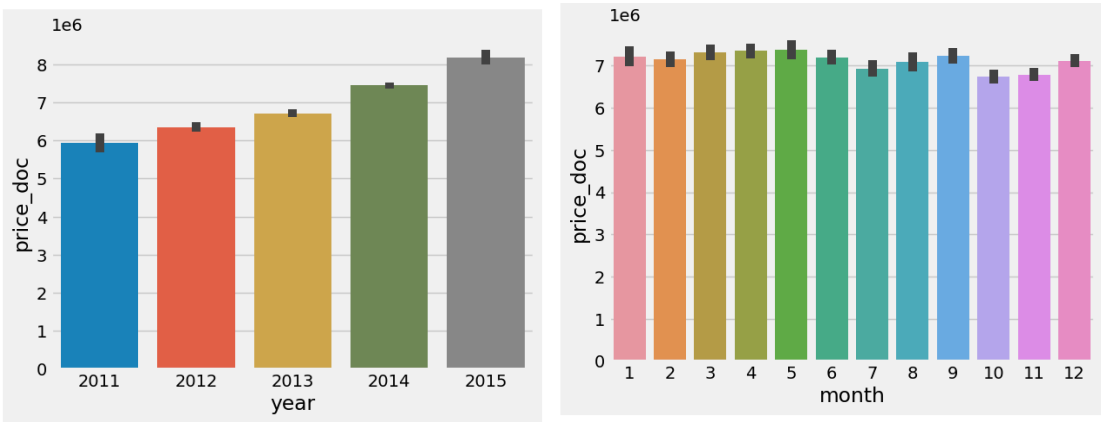


Figure 5: Effect of year and month on *price_doc*

As shown in Figure 5, the mean price tends to go up as the year goes on. However, when we look at the monthly data for price_doc, there is not a clear and steady pattern. The month-to-month changes in price_doc seem a bit unpredictable, indicating there might be some factors at play that need a closer look. It's worth digging deeper to figure out what might be causing these fluctuations.

2.3. To *macro* or not *macro*

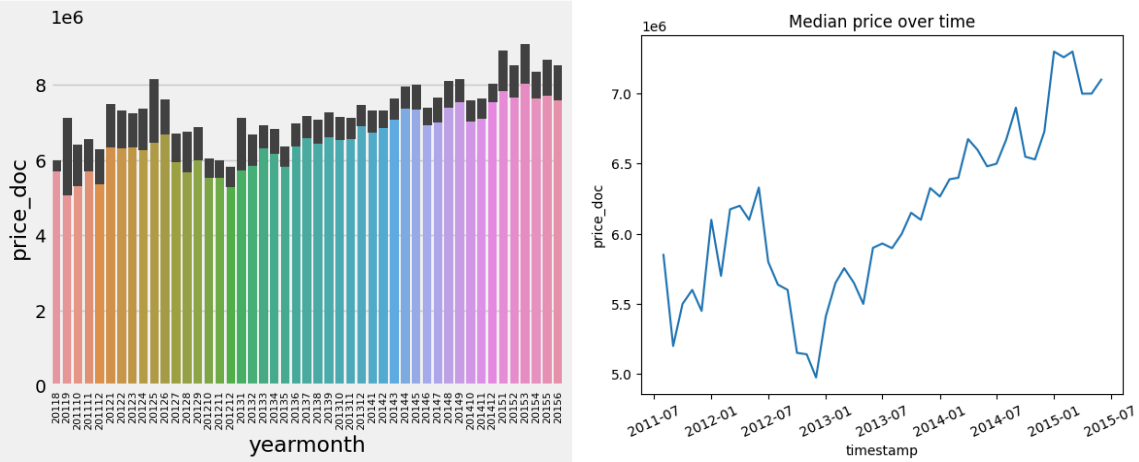


Figure 6: Mean and median price over time

We attempt to utilize macroeconomic data, which contains information about the macroeconomy of Russia in different time periods, to enhance our predictive models. We recognized a connection between GDP growth and the real estate market. Consequently, we experiment with adjusting the predicted prices by multiplying them with the factor (100 + gdp_quart_growth) / 100, with gdp_quart_growth representing the GDP growth data sourced from the *macro.csv* file.

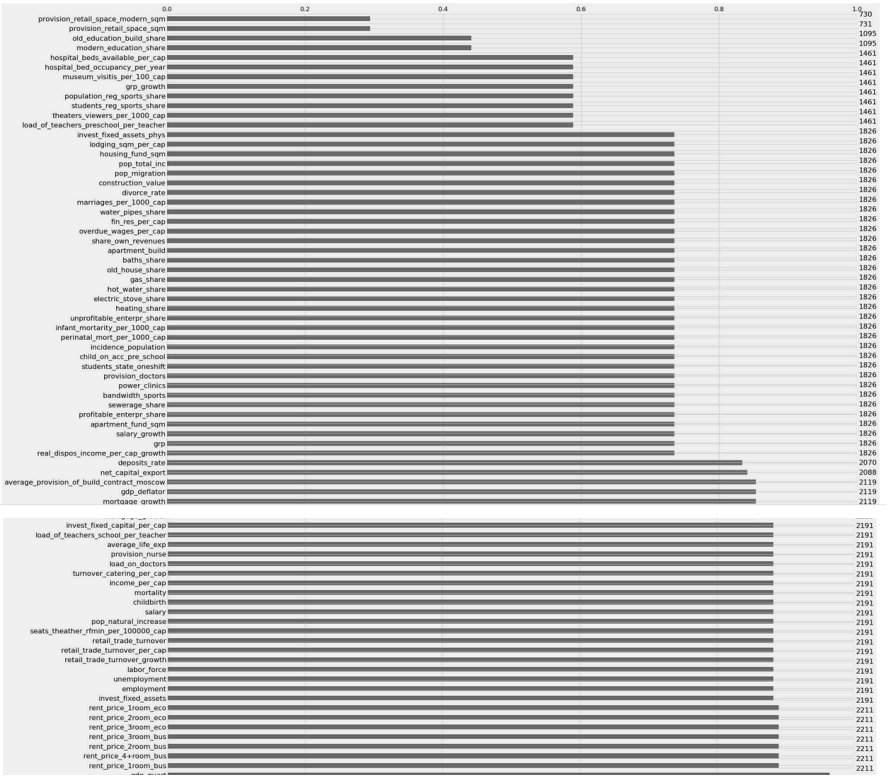


Figure 7: Missing values of macro_economy

Unfortunately, our empirical experiments indicate that integrating macro information does not yield a substantial improvement in the accuracy of our selected models. This may be because a lot of the columns are missing a lot of data, and usually represent data not available at the time of collection. The missing data also occurs temporally and in conjunction with each other, mainly due to the statistics not being available at the time, such as quarterly GDP not being available as the quarter has not finished yet. Imputing it with statistics like mean or forward/backfilling would be inaccurate, and more complicated methods would lack the data from other attributes.

3. Our proposed method

3.1. Preprocessing

3.1.1. Column Adjustments

The general approach for eliminating columns in our dataset is to remove less meaningful or high collinearity columns. While less meaningful columns play a minimal role in describing real-estate prices, high collinearity columns are often mutually associated and cause the predictions to be overestimated. The detailed set of criteria is as below.

1. Index column: This column is just for identification purposes and has no relation with predicting the target price. Therefore, we decided to drop this column.

2. A large percentage of missing values: We will remove those columns with missing values proportion greater than 0.4. After running summary statistics, about 10 columns are removed with respect to this condition.
    - df.isnull().mean() > 0.4

3. Obsoleted distance columns: These columns are highly associated with the time measurement to reach certain destinations within the dataset, hence we can proceed to remove them.
    - metro_km_walk
    - railroad_station_walk_km
    - public_transport_station_km

4. Counting of destinations between an area: These columns are often in the form of x_count_y or x_part_y, meaning there are a number of counts or shares of x within an area of y surrounding the instance. They are almost repetitive and do not enhance the prediction ability. Some examples are illustrated below.
    - cafe_count_3000_price_1500
    - cafe_count_3000_price_2500
    - cafe_count_3000_price_4000
    - green_part_2000
    - prom_part_4000
5. Variance threshold:
    We try to remove the columns that have a lower variance than 0.25, as these low-variance features might contribute not much to the final estimator and usually introduce noises to the data. We found that by using this, the evaluation score is better.

On the other hand, we also create some new columns which are more related to prediction power. Details are shown below.

1. Population per area:
    - df['population_per_area'] = df['raion_popul'] / df['area_m']
    - df['pop_per_mall'] = df['shopping_centers_raion'] / df['raion_popul']
    - df['pop_per_office'] = df['office_raion'] / df['raion_popul']

2. Educational opportunity and capacity (Pre-school and School age):
    - df['preschool_fill'] = df['preschool_quota'] / df['children_preschool']
    - df['preschool_capacity'] = df['preschool_education_centers_raion'] / df['children_preschool']
    - df['school_fill'] = df['school_quota'] / df['children_school']
    - df['school_capacity'] = df['school_education_centers_raion'] / df['children_school']

3. Age distribution of the local population (working and retirement age):
    - df['percent_working'] = df['work_all'] / df['full_all']
    - df['percent_old'] = df['ekder_all'] / df['full_all']

3.1.2. Outlier Removal

In our data preprocessing phase, we employ a data-centric approach using human knowledge to curate the training dataset. This process is rooted in the understanding that the quality and precision of data significantly impact the performance of machine-learning models. We establish a set of rule-based criteria to identify and subsequently eliminate samples that do not adhere to the expected data quality standards.

The criteria for data exclusion are as follows:

1. Inconsistent Area Values: We exclude data samples where the total area is less than the living area or the kitchen area is smaller than 2 square meters. Specifically:
    - total_sq < life_sq
    - life_sq < kitch_sq
    - kitch_sq < 2

2. Minimum Area Threshold: To ensure the inclusion of meaningful data, we exclude samples where the total area and living area are both less than 5 square meters. This step helps in filtering out data points with minimal or erroneous area values.
    - total_sq < 5
    - life_sq < 5
3. Outliers in Full Area Ratio: Samples with a full area greater than 210 square meters and a life area that accounts for less than 30% of the full area are removed to prevent outliers from affecting the model's performance.
    - full_sq > 210
    - life_sq / full_sq < 0.3
4. Excessive Living Area: Samples with living areas exceeding 300 square meters are considered outliers and are excluded from the dataset.
    - life_sq > 300

5. Implausible Building Years: Data points with building years earlier than 1500 or beyond 2023 are dropped due to their implausibility.
    - build_year < 1500
    - build_year > 2023
6. Negative Room Counts: Entries with non-positive room counts (num_room <= 0) are omitted as they do not conform to typical property configurations.
7. Invalid Floor Information: We filter out samples where the maximum floor or floor number is zero or the floor number exceeds the maximum floor number. This ensures that floor information is logical and reliable.
    - max_floor <= 0
    - floor <= 0
    - floor > max_floor
8. Kitchen Area Threshold: We also remove data points with unrealistically small kitchen areas, which may be indicative of data errors.
    - kitch_sq < threshold

3.2 Relevant libraries and tools for data preprocessing

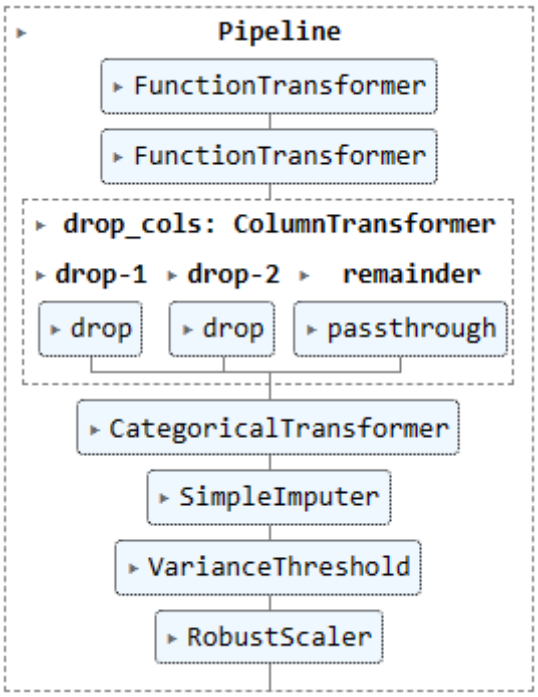


Figure 4. Our pipeline for data preprocessing

FunctionTransformer: A custom-defined function or a function object is employed to generate the outcome of the said function, which can be particularly beneficial for stateless transformations. Therefore, we can create custom transformations for categorical columns, time columns, and categorical columns that have custom orders, namely the *"ecology"* column where it is in this order [*'poor', 'satisfactory', 'good', 'excellent'*]

Imputer: In this project, we implement ***SimpleImputer*** from the *scikit-learn* library, with *strategy='median'* for **numeric** columns and *'most_frequent'* for **categorical** columns, empirical experiment shows that this is the best approach for a large-scale dataset in this project.

Scaler: In this project, the dataset is highly skewed and contains a lot of outliers. Therefore, we implement different data scaling methods, namely StandardScaler and RobustScaler. These techniques are often used to prepare data for machine learning algorithms, ensuring that features are on a consistent scale and thereby improving the performance of the models.

- **StandardScaler**: StandardScaler rescales the data by shifting the mean to 0 and scaling the variance to 1. It assumes that the data follows a normal distribution, which makes it sensitive to outliers. In cases where your dataset contains outliers or non-normally distributed data, StandardScaler may not be the best choice, as it can distort the feature scaling.
- **RobustScaler**: A scaling technique that is more robust to outliers and non-normally distributed data. It scales the data by replacing the mean with the median and the standard deviation with the interquartile range. This makes it a suitable choice when dealing with data that has outliers or does not follow a normal distribution.

Empirical experiment shows that RobustScaler consistently outperforms StandardScaler in multiple model settings. This may be due to that our dataset contains outliers and is not normally distributed. By using RobustScaler, we can ensure that the scaling process is less affected by these anomalies, resulting in a more accurate representation of the data and ultimately better model performance.

3.3 Models

Given that forecasting property values fall under the category of regression tasks, we opt to conduct experiments employing widely used regression methods. To address this particular challenge, we utilize the regression algorithms provided by the sklearn library. Initially, the following techniques were chosen:

- Gradient Boosting Machine
- XGBoost
- LightGBM
- CatBoost

3.3.1. Gradient Boosting Machine (GBM)

Gradient Boosting utilizes the Boosting method to convert weak tree learners into strong tree learners by training many of them in an additive and sequential manner. Gradient Boosting identifies weak points of the previous model by utilizing the gradient with respect to the loss function. This gives the model the ability to target a specific loss function.

We use Optuna with a 5-fold split to tune the parameters of GBM. We also utilize early stopping at 50 iterations to save running time.

Parameters	Explanation	Search Values	Best Value
learning_rate	The learning rate controls how much the model's parameters should be adjusted in response to the loss after each iteration.	[0.001, 0.3]	0.00526

n_estimators	Number of boosting stages to perform	[100, 5000]	4300
subsample	Fraction of sample to be used for the base learners	[0.5, 0.9]	0.9
max_features	How the algorithm decide the features for splitting	[auto, sqrt, log2]	log2
max_depth	How deep one tree should be	[3, 9]	7

After optimising, our model obtained a score of 0.32647 on the testing dataset.

3.3.2. XGBoost

The XGBoost is an implementation of gradient boosted decision trees. It makes use of multiple decision trees to improve the predictions while optimizing a cost function, similar to the Gradient boosting regressor. However, XGBoost seems to generally have a better performance than other algorithms like Gradient boosting regressor due to it having an implementation that creates a more regularized model, which helps with overfitting. Furthermore, it is able to handle any missing data without the need for preprocessing.

For hyperparameters tuning, first the *learning_rate* is set to high (0.1) and *n_estimators* is tuned to find the optimised *n_estimators*, then *max_depth* and *min_child_weight* are tuned together as these 2 hyperparams are related to each other. Then *subsample* and *colsample_bytree* are tuned, these two hyperparams turn out to be best at their default value (0.8). Finally lower the learning_rate and train the final XGBoost model.

Parameters	Explanation	Search Values	Best Value
learning_rate	The learning rate controls how much the model's parameters should be adjusted in response to the loss after each iteration.	[0.01, 0.05, 0.1]	0.01
max_depth	The maximum depth of a tree. Used to control over-fitting.	[5, 6, 7]	6
n_estimators	The number of trees used to estimate. Less trees ~ less complexity ~ less overfitting	[500, 1000, 2000]	1000
min_child_weight	Stop trying to split once your sample size in a node goes below a given	[4, 6, 8]	8

	threshold. Prevent very extreme regression when node might contain only 1 sample.		
subsample	Fraction of the training samples (randomly selected) that will be used to train each tree	[0.5, 0.8, 1.0]	0.8
colsample_bytree	Fraction of features (randomly selected) that will be used to train each tree	[0.5, 0.8, 1.0]	0.8

3.3.3. CatBoost

CatBoost leverages decision trees with a boosting method as its base learners, purposefully designed to handle scenarios involving categorical features. It also demonstrates adeptness in effectively managing missing values within the data. The algorithm further employs symmetric trees to optimize both training time and accuracy. We apply grid search on the parameters of CatBoost.

Parameters	Explanation	Search Values	Best Value
learning_rate	The learning rate controls how much the model's parameters should be adjusted in response to the loss after each iteration.	[0.1, 0.2, 0.5, 1]	0.1
iteration	Iteration represents the number of times the CatBoost algorithm will run to minimize the loss function.	[150, 200, 300, 2000]	150
depth	It is the pre-set value for the depth of the trained tree	[2, 5, 10, 15]	5

Using the optimal parameters, our Catboost model attains a Root Mean Squared Logarithmic Error of 0.32172 on the testing dataset.

3.3.4. LightGBM

LightGBM is another gradient boosting framework that utilizes tree-based learning algorithms. Its major advantages can be described as:

- Fast training speed with better efficiency
- Less memory usage
- Accuracy improvement
- Allow parallel, distributed and GPU learning
- Able to handle large-scale data

With all of the above advantages, LightGBM is often the winning solution of machine learning competitions, together with much lower resource consumption. The GridSearch in our model testing is shown below.

Parameters	Explanation	Search Values	Best Value
n_estimators	Number of boosted trees to fit in model	[300, 350, 400, 450, 500]	400
learning rate	The learning rate controls how much the model's parameters should be adjusted in response to the loss after each iteration.	[0.05, 0.1, 0.15]	0.1
max_depth	Maximum of tree depth for based learners	[6, 7, 8, 9]	7
num_leaves	Maximum of tree leaves for based learner	[30, 35, 40, 50]	35
boosting	Boosting type	[‘gbdt’, ‘dart’]	‘dart’

3.4. Ensemble Learning

In our project, we have utilized the power of ensemble learning to enhance the performance and reliability of our predictive model. Ensemble learning has proven to be an indispensable tool for improving predictive accuracy, reducing overfitting, and achieving robustness in machine learning tasks. By combining the predictions of multiple base models, we can effectively exploit the strengths of various algorithms and mitigate their weaknesses, resulting in a more accurate and stable overall prediction.

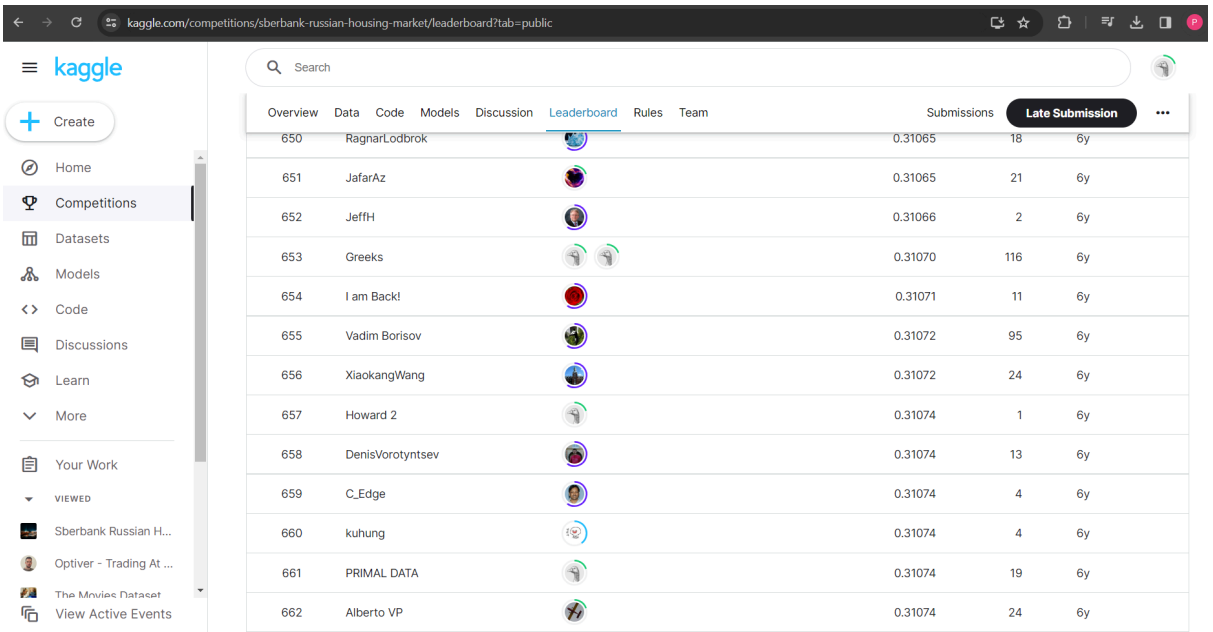
After evaluating the performance of the candidate models, we chose four LightGBM models and one XGBoost model for ensemble learning. We have then utilized an aggregate scoring mechanism to combine the outputs of these models. The empirical experiment shows that the average scoring mechanism outperforms other mechanisms such as median or weighted voting. This approach has enabled us to maximize the strengths of machine learning models while minimizing the impact of potential model-specific weaknesses, leading to a more robust and accurate prediction model for our application.

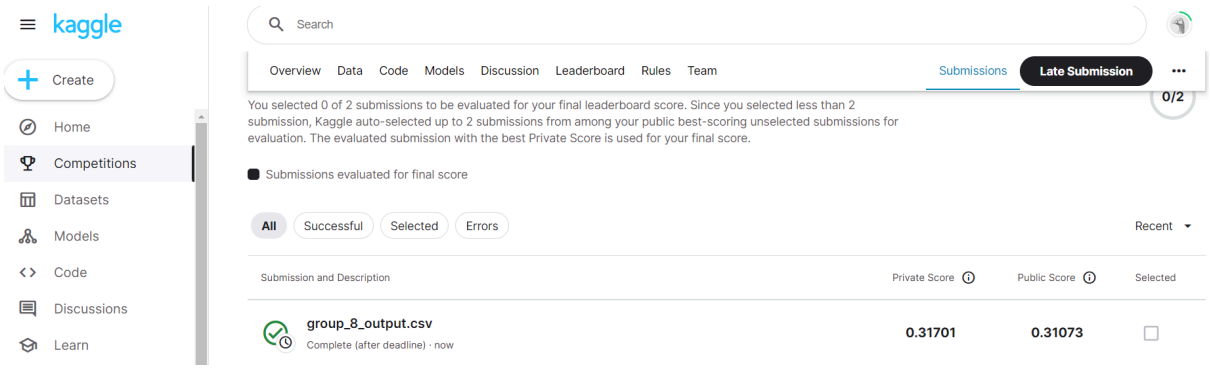
4. Result & Discussion

4.1. Result

Model	Public Score
Final model (Unweighted Average of 4 LightGBM and 1 XGBoost models)	0.31073 (Top 20% Leaderboard)
LightGBM	0.31345
XGBoost	0.31548
GBM	0.32647
CatBoost	0.32172

As from the above table, XGBoost and LightGBM alone are the two best models with a score of 0.31548 and 0.31345. The remainder of GBM and CatBoost are not performing that good, with a score of above 0.32. Therefore we approach aggregating models by weighted average of LightGBM and XGBoost. We compare all combinations of (a, b) such that the sum of a and b is 1, then apply on model output to get the respective scores. It is indicated that the pair (0.8, 0.2) works the best and produces a score of 0.31073, reaching top 20% of Kaggle public leaderboard. While tuning closer around a of 0.8 and b of 0.2, we do not see any further improvements. Therefore, we complete our modeling. We observe that LightGBM is the most suitable for Sberbank housing dataset, but if it is needed to add a portion of XGBoost so that we can enhance the generalization ability on unseen data.





Public Leaderboard ranking: 656/3264 (Top 20%)

4.2. Discussion

A. Importance of proper data cleaning

Through this project, we discover the significance of effective data cleaning in facilitating the extraction of meaningful information and patterns by models. Failure to conduct thorough data cleaning could significantly hinder the model's performance, and even extensive hyperparameter tuning or model selection may prove insufficient to enhance its effectiveness.

B. Importance of choosing the right model and the limitations

Optimizing hyperparameters is crucial for identifying an optimal combination of values that allows accurate predictions on the given dataset. It is acknowledged that achieving perfection in hyperparameter selection is impractical due to the time-intensive nature of cross-validation across various parameter combinations. Consequently, there are instances where a trade-off between model performance and timely results becomes necessary. Interestingly, we observe that empirical hyperparameter tuning, as opposed to systematic exploration, can still yield a model with commendable performance.

Moreover, our exploration revealed that solely adjusting models and hyperparameters has its limitations in enhancing prediction accuracy. To achieve further improvements, additional efforts in data refinement and feature engineering may be essential.

Furthermore, we dive into the efficacy of ensemble learning by employing ensemble-based algorithms from sklearn and developing custom ensemble-based predictions.

5. Conclusion

In conclusion, our project aimed to address the challenges posed by predicting real estate prices using the Sberbank dataset through the application of machine learning models. We began by thoroughly exploring the dataset, gaining insights into its features and assessing the macroeconomic factors' impact. Our proposed method involved a systematic preprocessing approach, including column adjustments and outlier removal, to enhance the quality of our

data. Leveraging relevant libraries and tools, we implemented a variety of machine learning models, including Gradient Boosting Machine, XGBoost, CatBoost, and LightGBM, to comprehensively analyze their performance.

Ensemble learning further improved our predictive capabilities, demonstrating the power of combining multiple models. Despite the challenges inherent in real-world datasets, our project contributes valuable insights into the application of machine learning in the real estate domain. The success of our models suggests their potential for practical implementation in real-world scenarios, aiding stakeholders in making informed decisions. This project not only provides a solid foundation for predicting real estate prices but also underscores the significance of meticulous data exploration and preprocessing in enhancing the performance of machine learning models.