

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**  
**SINGAPORE**

CZ4041/SC4000 Group Assignment

**Sberbank Russian Housing Market**

Can you predict realty price fluctuations in Russia's volatile economy?

Name	Matriculation Number	Role
Isabelle Ng	U1922243C	XGBoost
Sethi Aryan	U2123583E	Data Cleaning, Feature Engineering, Lasso Regression, Ensemble Learning (Stacking)
Mukherjee Tathagato	U2120365K	Ridge Regression, Random Forest
Lee Xuanhui	U1921226K	LightGBM
Siddid Khanna	U2023684H	EDA, Data Cleaning, Data Preprocessing

**Group: 53**

## Introduction

Machine learning is increasingly being utilized to optimize and enhance various facets of our lives. Machine Learning is being increasingly used to make predictions about prices in different fields so that customers and lenders are better informed before they make any investments or business decisions. Price prediction involves developing models that can forecast the future prices of financial assets or commodities based on historical data and various features.

## Problem Statement

Sberbank, Russia's oldest and largest bank, helps its customers by making predictions about realty prices so renters, developers, and lenders are more confident when they sign a lease or purchase a building. The problem statement is to develop a model that will use a broad spectrum of features to predict realty prices. Sberbank has provided a dataset that provides housing features and patterns. We need to predict future prices as accurately as possible, 'price\_doc' is the "target" variable that the machine learning model should predict.

## Objectives

- The model could help stakeholders, including buyers and sellers, to make more informed decisions by having a better understanding of the future value of properties.
- Accurate predictive models can help identify potential risks in the market.
- Eventually, such a model can help customers in better financial planning. By having reliable predictions of future property values, customers can make more informed decisions.

## Evaluation

$$\text{RMSLE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\log(p_i + 1) - \log(a_i + 1))^2}$$

Where:

$n$  is the total number of observations in the (public/private) data set,

$p_i$  is your prediction of target, and

$a_i$  is the actual target for  $i$ .

$\log(x)$  is the natural logarithm of  $x$  ( $\log_e(x)$ ).

Submissions are evaluated on the RMSLE between their predicted prices and the actual data. The target variable, called price\_doc in the training set, is the sale price of each property. The lower the RMSLE, the better the predictions.

## Exploratory Data Analysis & Data Pre-Processing

### 1. Train.csv and Test.csv

This file contains information about individual transactions. The rows are indexed by the "id" field, which refers to individual transactions. This file also includes supplementary information about the local area of each property.

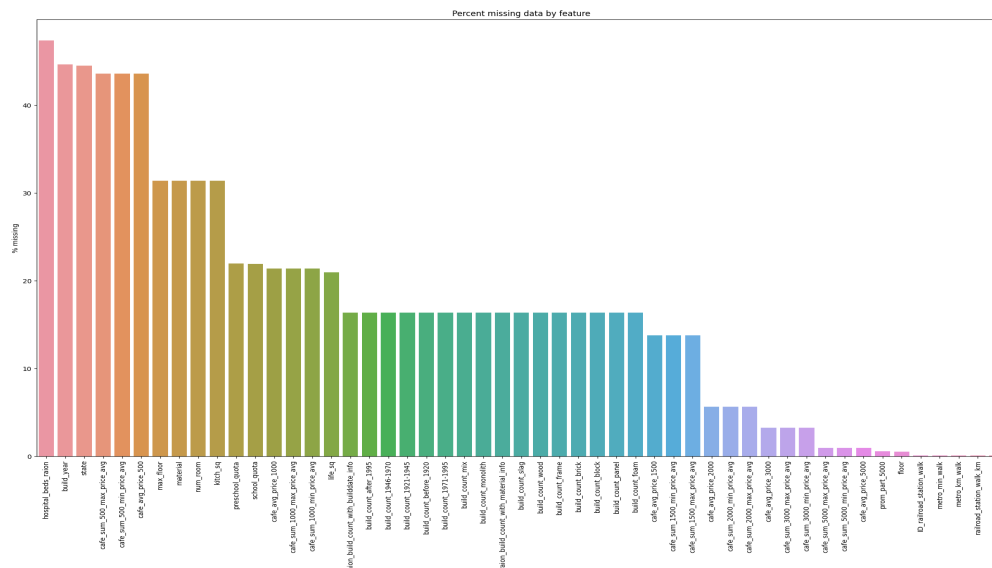
Train.csv contains many features such as:

- ID: A unique identifier for each data entry.
- Timestamp: Time information indicating when the data was recorded.
- Property Features: Details about the real estate properties being considered. This may include information such as the number of rooms, floor area, location, type of property, etc.
- Amenities and Conditions: Information about amenities (e.g., presence of a pool, garage, etc.) and the overall condition of the property.

For Data Cleaning, we have first merged the train and test datasets. We have then removed the features which have a correlation of less than 5% with the target variable. Reducing the number of irrelevant features simplifies the model, making it easier to interpret and understand. This makes the model less prone to overfitting and also increases computational efficiency. Besides this variables that have no variance have also been removed. These variables do not contain any information that helps distinguish between different instances in the dataset and thus do not provide any meaningful insights.

Several irrelevant data points needed to be replaced. These columns include the 'state', 'material', 'number of rooms', and the 'square feet' columns. Removing these irrelevant values helps in improving the data quality which in turn helps in improving model performance and consistency.

There is a large amount of missing data in train.csv and these missing values significantly affect the performance of our model. These missing values need to be dealt with before any further computation. The percentage of missing Data in each column can be seen below.



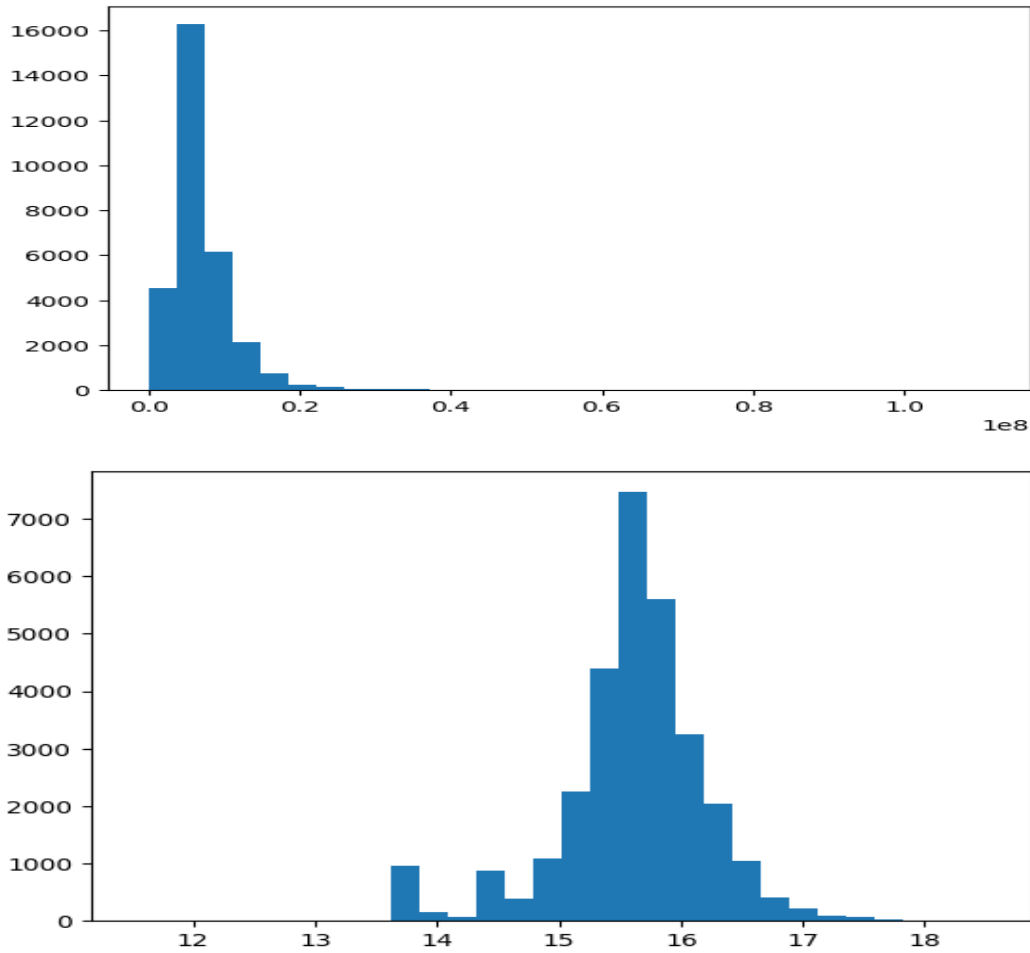
When the percentage of missing values was less than 30% we used 'Median Imputation'. Median imputation is a technique used to fill missing values in a dataset by replacing them with the median value of the variable. Median imputation is particularly useful when dealing with skewed distributions or when the data contains outliers as is the case with our Data. On the other hand, when the percentage of missing values was greater than 30% we used Model-based imputation using the KNN imputer.

To ensure that features with larger magnitudes do not dominate the learning process and lead to suboptimal performance we performed scaling for the numerical features. Scaling ensures that all numerical features contribute equally to the computation of distances and gradients during the training of models. We used the min-max scaling technique for our data.

For the Categorical features, we used Label Encoding. The process of encoding involves converting categorical features into a numerical format which can be used by our algorithm. Label Encoding Assigns a unique integer to each category. This method is suitable when the categories have an ordinal relationship, meaning there is a meaningful order among them.

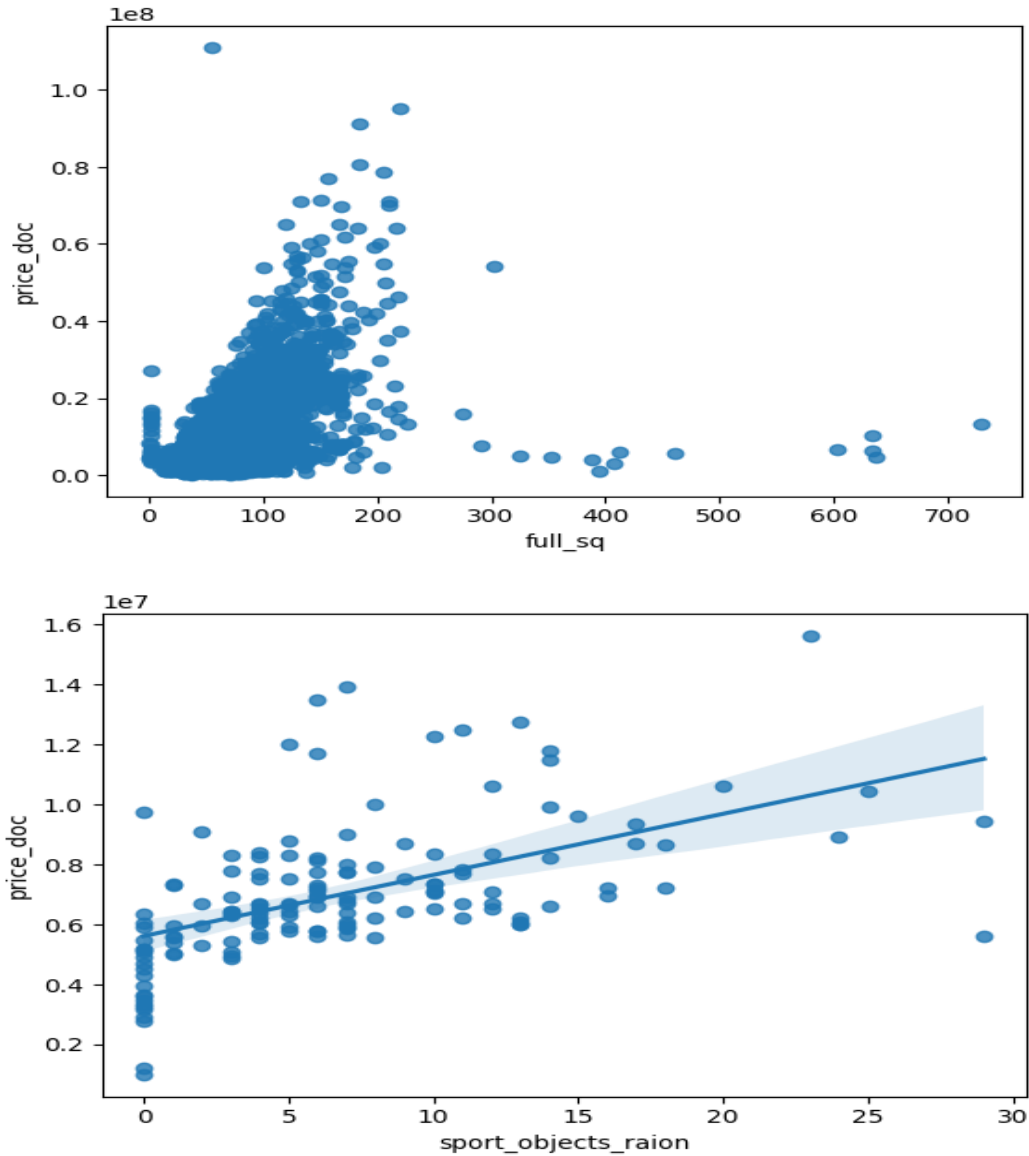
## 2. Target Variable (price\_doc)

The Target Variable is 'price\_doc' which represents the price of the property. This is the variable we need to predict in our model. The Distribution of the Response variable and log of the response variable can be seen below.

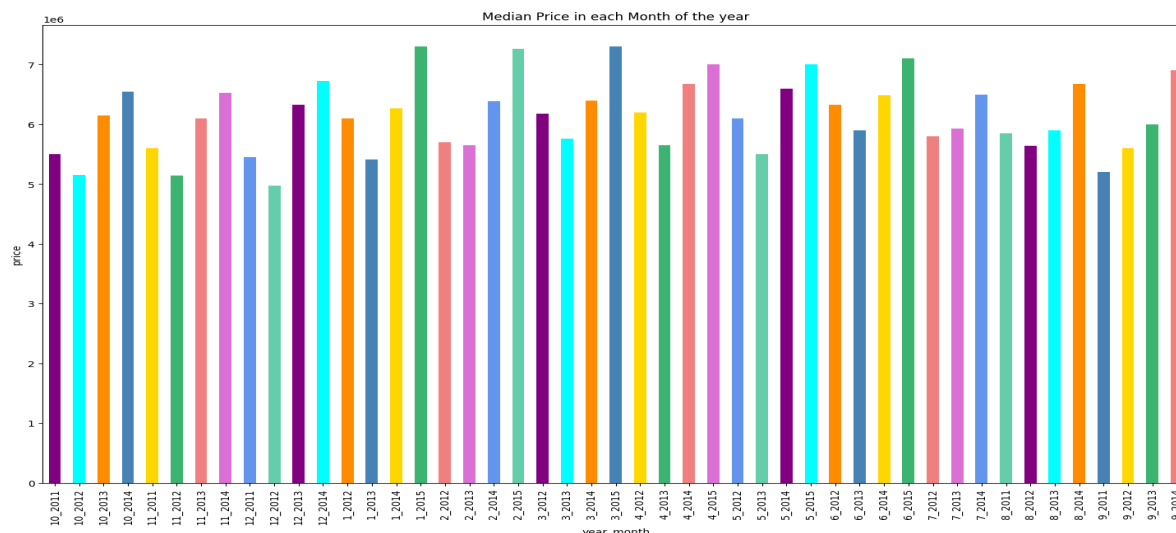


The distribution of the target variable appears to be mostly normal, with a few outlier values.

We also plotted the response of the target variable vs other variables to understand the effect that each variable has on price\_doc. The plot for price vs area and price vs no of sporting areas in the vicinity can be seen below.



We also checked the prices over all the months to see if the prices fell or increased abnormally during any of the months as this could adversely affect the predictions made by our model. From the graph below we can see that although the prices continuously increased and decreased throughout the months, the changes should not be adverse enough to affect the predictions made by our model. Similar trends in train and test data indicate that there is no need for time-based data splitting, ensuring accurate predictions.



## Feature Engineering

Feature engineering is the process of creating new features or modifying existing ones in a dataset to improve the performance of a machine-learning model. The Techniques we used for Feature Engineering are

### 1. Date Time Features

These features enable the model to capture the temporal patterns and dynamics of real estate prices, which are crucial for accurate predictions. By extracting features such as year and month the model can learn to recognize the temporal variations in customer behavior. We did this by splitting the timestamp into year and month.

### 2. Aggregation

Aggregation is the process of clustering existing features to generate new features that give a glimpse into the way customer behaviors change concerning another factor. We did this by creating binary variables.

Some of the features we created are:

- We created two new features, 'year' and 'year\_mo', based on the 'timestamp' column in the DataFrame.
- We created the feature 'resident\_to\_total\_ratio'.
- We created the features 'Avg\_room\_area' and 'extra\_area'.
- Other features include 'apt\_name', 'pct\_labor\_force', and 'floor\_rel\_total'.
- We also created a few binary variables such as Average building height for subarea, and Monthly Sales for the apartments.

## Model Building

### 1. Lasso Regression

#### A) Overview

Lasso regression is a regularization technique. It is used over regression methods for a more accurate prediction. It is a popular technique used in statistical modeling and machine learning to estimate the relationships between variables and make predictions. LASSO stands for Least Absolute Shrinkage and Selection Operator.

#### B) Motivation

LASSO regression was selected for its proficiency in achieving a balance between the simplicity and accuracy of the model. The rationale behind this selection was driven by the necessity of a model that is precise yet easy to interpret. The unique aspect of LASSO, which involves diminishing the influence of lesser important features, plays a pivotal role in simplifying the model without detracting from its effectiveness. Its inherent capability of automatic feature discrimination, favoring more significant variables, contributes to both the model's streamlined efficiency and its comprehensibility.

#### C) Approach

In the conducted analysis, the data was initially partitioned into training and validation sets, to facilitate an effective model evaluation. The cross-validation process was then implemented using the *RepeatedKFold* method, which involved the dataset being divided into ten parts, with the validation process being executed five times with varying data divisions. This approach was instrumental in ensuring a comprehensive evaluation of the model's performance.

The optimization of the *alpha* parameter in the LASSO model was accomplished through *LassoCV*, employing the established cross-validation framework. The absence of predefined *alphas* allowed for an autonomous exploration of potential values, aiming to identify an optimal balance in the model's complexity.

Finally, the training of the LASSO model, incorporating the optimal *alpha* value was executed. This helped develop a model that not only adheres closely to the training data but also maintains a level of simplicity to avoid overfitting, thereby enhancing its applicability in broader contexts.



## D) Final Results

▼	Lasso
	Lasso(alpha=0.23500321021261555)

The most optimized value of alpha was chosen to train the final model. We used 85% of the training dataset and the rest for cross-validation.

The results of the model are as follows:

Train RMSE: 0.5960

Test/Validation RMSE: 0.8028

Kaggle Private Score: 0.8863

Kaggle Public Score: 0.88576

## 2. Ridge Regression

### A) Overview

Ridge Regression, known for its ability to handle multicollinearity by introducing a regularization penalty, has also been implemented in our project. The regularization strength is controlled by the hyperparameter alpha, which aims to enhance the model's generalization capabilities by constraining the magnitude of the coefficients and preventing overfitting.

### B) Motivation

The choice of Ridge Regression was motivated by the nature of our dataset, which exhibited features that were highly intercorrelated. By applying a Ridge model, we aimed to diminish the potential issues caused by multicollinearity and improve the model's prediction accuracy. The regularization aspect of Ridge Regression ensures that while the model remains sensitive to the data, it also maintains a level of robustness against variance.

### C) Approach

Since the initial model with  $\alpha=0.0002$  yielded  $\text{RMSE} = 0.7798$ , Hyperparameter tuning played a pivotal role in refining our Ridge Regression model. We deployed RidgeCV, a cross-validated ridge regression model, to determine the most effective alpha value. Through iterative computation, the model explored a range of alpha values and identified  $\alpha=2.3101297000831603$  as the optimal parameter, which provided a balance between model complexity and performance.

Our model underwent rigorous evaluation, with the Root Mean Squared Error (RMSE) serving as the primary metric. The validation set yielded an RMSE of 0.5837074607691141, which indicated a satisfactory predictive performance. Furthermore, to diagnose the model for potential overfitting, we compared the RMSE scores between the training set (0.4880393715071023) and the validation set, finding a subtle difference that validated the model's generalization ability.

#### **D) Final Results**

Post hyperparameter tuning, the Ridge Regression model was submitted to Kaggle, where it achieved a public score of **0.52837** and a private score of **0.54656**. These scores reflect the model's competitive edge in a real-world scenario, demonstrating its capability to make accurate predictions on unseen data.

### **3. Random Forest**

#### **A) Overview**

The Random Forest algorithm, known for its robustness and high accuracy, was chosen to predict housing prices due to its ensemble method. It constructs multiple decision trees and aggregates their results, thus reducing the risk of overfitting while capturing complex interactions within the data.

#### **B) Motivation**

A common pitfall when dealing with high-dimensional data is overfitting, which Random Forest is also capable of handling robustly. The model's innate mechanism of averaging decisions from multiple trees diminishes the noises and enhances the signal in the predictive process. Moreover, its capacity to uncover nonlinear interactions between variables is crucial for capturing the nuanced dynamics of the real estate market. Importantly, the algorithm's inherent feature selection capabilities enable us to discern the most impactful predictors, optimizing computational efficiency.

#### **C) Approach**

Using the default hyperparameters (100 trees and a random state of 42 for reproducibility), the initial Random Forest model was trained. The model's RMSE was found to be **0.4890**, which was even more promising than the optimized Ridge Regressor. This performance indicated a strong starting point for further optimization.

To improve the model's performance, a hyperparameter tuning process was undertaken using RandomizedSearchCV. The hyperparameters included the number of trees (n\_estimators), the maximum depth of the trees (max\_depth), the minimum number of samples required to split an internal node (min\_samples\_split), and the minimum number of samples required to be at a leaf node (min\_samples\_leaf). The search also varied the number of features considered for each split (max\_features).

The best parameters found from this search were n\_estimators: 100, min\_samples\_split: 2, min\_samples\_leaf: 2, max\_features: 'sqrt', and a max\_depth of 10, suggesting a preference for a more regularized model to prevent overfitting.

#### D) Final Results

The optimized Random Forest model was evaluated on a separate validation set, where it achieved an RMSE of **0.3458** on Kaggle's public leaderboard and **0.3451** on the private leaderboard. These scores reflected an improvement over the initial model, validating the effectiveness of the hyperparameter tuning.

An overfitting check was performed by comparing the RMSE on the training set (0.3869) with that on the validation set. The close performance between these two sets suggested that the model generalized well and was not overfitting.

## 4. XGBoost

### A) Overview

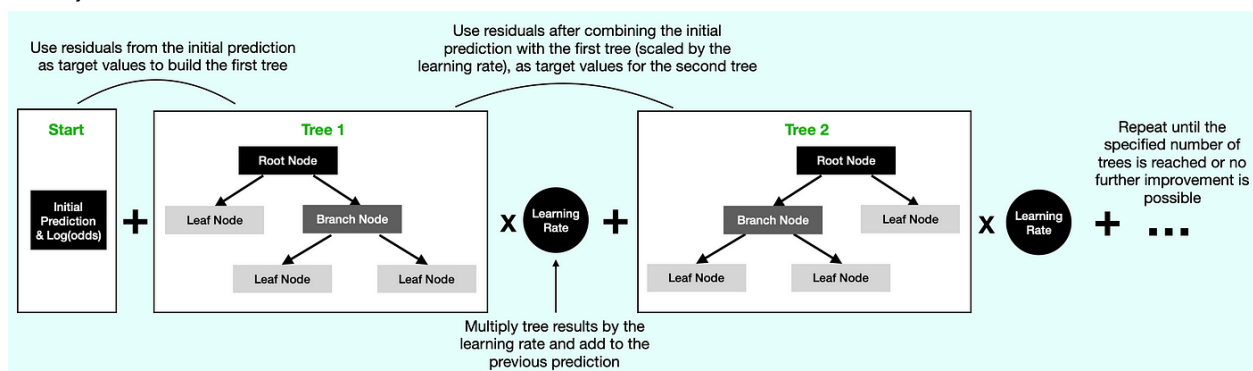


Figure 1: XGBoost Process [1]

XGBoost is an ensemble learning method that builds multiple decision trees in a sequential manner, where each tree tries to correct the errors made by the previous ones. It is used for both regression and classification problems. The

gradient boosting optimization technique constructs a new model to minimize the loss function of an existing ensemble by sequentially adding decision trees that specifically target the residuals or errors of prior trees.

## B) Motivation

XGBoost is particularly well-suited for the dataset because it can handle a mix of categorical and continuous features, it can model complex nonlinear relationships, and it can be regularized to prevent overfitting of which all of these are present in our dataset. XGBoost is also able to handle missing values and outliers.

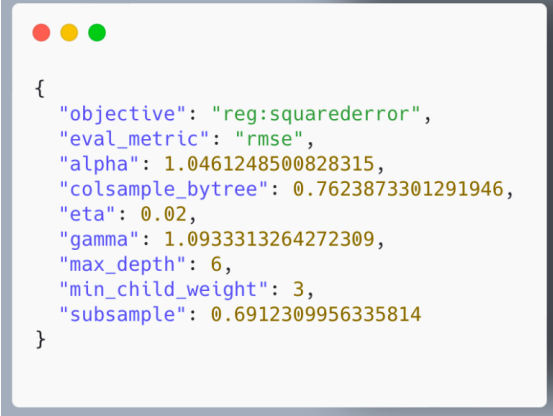
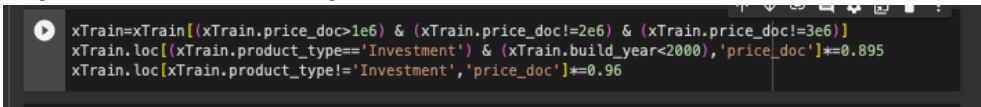
XGBoost also offers a higher level of model performance by allowing the tuning of numerous parameters to fine-tune predictions, which generally yields better results than simple decision trees or random forests. This customization, paired with XGBoost's advanced handling of overfitting through its regularization parameters, often results in a more accurate and robust model compared to simpler decision tree or random forest models, especially in complex datasets like the one provided by Sberbank's Russian Housing Market.

The DMatrix data structure in XGBoost optimizes data handling for improved speed and efficiency, making it faster than many other gradient-boosting models and still giving us good prediction results

## C) Approach

Feature engineering has already been carried out on the data, with missing values filled in according to the methods outlined in Section 4.

Experiments Description	Results
<b>Experiment 1: Baseline Model</b> <ul style="list-style-type: none"> <li>Ran a basic XGBoost model with default hyperparameters</li> <li>Established Performance benchmarking with RMSE</li> </ul>	Private Score: 0.33259 Public Score: 0.32975
<b>Experiment 2: Pruned Feature Set (50 Features)</b> <ul style="list-style-type: none"> <li>Reduce features beyond the top 50</li> <li>Observed RMSE improvement in test data</li> </ul>	Private Score: 0.32616 Public Score: 0.32489
<b>Experiment 3: Aggressive Feature Reduction (20 Features)</b> <ul style="list-style-type: none"> <li>Further reduced the features to beyond the top 50</li> </ul>	Private Score: 0.33095

<ul style="list-style-type: none"> <li>Noticed an increase in RMSE, an indication of potential over-pruning</li> </ul>	Public Score: 0.33026
<p><b>Experiment 4: Full Feature Tuning</b></p> <ul style="list-style-type: none"> <li>Performed RandomizedSearchCV on top 50 on all features</li> <li>Post Optimization, select the top 50 features</li> <li>Apply RandomisedSearchCV again for refined hyperparameter Tuning</li> </ul>	Private Score: 0.33832 Public Score: 0.33285
<p><b>Experiment 5: Optimal Results for Naive XGBoost</b></p>  <pre> {   "objective": "reg:squarederror",   "eval_metric": "rmse",   "alpha": 1.0461248500828315,   "colsample_bytree": 0.7623873301291946,   "eta": 0.02,   "gamma": 1.0933313264272309,   "max_depth": 6,   "min_child_weight": 3,   "subsample": 0.6912309956335814 } </pre> <ul style="list-style-type: none"> <li>Start with the top 50 features from Experiment 2</li> <li>Use RandomisedSearchCV again for refined hyperparameter Tuning</li> </ul>	Private Score: 0.32603 Public Score: 0.32469
<p><b>Experiment 6: Final Optimal Results for XGBoost</b></p>  <pre> xTrain=xTrain[(xTrain.price_doc&gt;1e6) &amp; (xTrain.price_doc!=2e6) &amp; (xTrain.price_doc!=3e6)] xTrain.loc[(xTrain.product_type=='Investment') &amp; (xTrain.build_year&lt;2000), 'price_doc']*=0.895 xTrain.loc[xTrain.product_type!='Investment', 'price_doc']*=0.96 </pre> <ul style="list-style-type: none"> <li>Used RandomizedSearchCV to find the best parameters</li> <li>Pruning features was noted to increase score on the test set after incorporating the magic numbers</li> <li>The model improved after making changes with the help of these magic numbers</li> </ul> <p><b>Rationale for Magic Numbers [2]</b></p> <ul style="list-style-type: none"> <li>Kaggle participants noticed the test set's average price might be lower than the training set's model predictions.</li> <li>The training data likely includes outliers or 'fake' prices, affecting prediction accuracy.</li> <li>Predicting these 'fake' prices was difficult without further research and external data on Russian property market values.</li> </ul>	Public Score: 0.31746 Private Score: 0.31854

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>• Modifying training data by reducing price values was found to enhance results on the Kaggle test dataset.</li> </ul> |  |
|---|--|

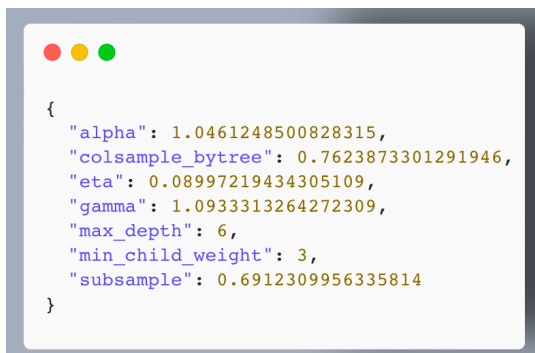
### Hyperparameters tuned in all experiments

In our hyperparameter tuning approach, we employed a randomized search strategy leveraging the RandomizedSearchCV method to refine the performance of an XGBoost model. This method was chosen due to its efficacy in exploring a large hyperparameter space without the exhaustive computation that grid search entails.

The specific hyperparameters selected were:

- eta (learning rate): To allow the model to learn efficiently without overshooting the minimum of the loss function.
- max\_depth: This parameter was bounded to ensure the model was complex enough to learn from the data but not so complex that it would overfit.
- subsample and colsample\_bytree: Both parameters introduce a stochastic element to the model training process, improving generalization by preventing overfitting.
- min\_child\_weight and gamma: These were set to manage the balance between model complexity and regularization.
- alpha: This L1 regularization term helps in feature selection, leading to a simpler, more interpretable model structure.

### D) Final Results



```
{
  "alpha": 1.0461248500828315,
  "colsample_bytree": 0.7623873301291946,
  "eta": 0.08997219434305109,
  "gamma": 1.0933313264272309,
  "max_depth": 6,
  "min_child_weight": 3,
  "subsample": 0.6912309956335814
}
```

The best-performing XGBoost model was chosen as our final result with the following hyperparameters. We used 85% of the training dataset and the rest for cross-validation.

The results of the model are as follows:

- Train RMSE: 0.2290
- Test/Validation RMSE: 0.2507
- Kaggle Private Score: 0.31854
- Kaggle Public Score: 0.31746

## 5. LightGBM

### A. Overview

LightGBM is a high-performance gradient-boosting framework that excels in building decision trees leaf-wise, rather than the traditional level-wise approach. This unique growth strategy allows it to concentrate on the most promising leaf, optimizing for better accuracy and efficiency. It's particularly effective for large datasets, offering fast training speeds and lower memory consumption. LightGBM can automatically handle categorical features and supports parallel as well as GPU learning, enhancing its scalability and speed.

### B. Motivation

LightGBM is particularly suitable for the Sberbank Russian Housing dataset for several reasons, making it an ideal choice for predictive modeling in this context:

- **Handling Diverse Features:** The framework's ability to handle a mix of categorical and numerical features directly aligns with the varied attributes of real estate data.
- **Robustness to Overfitting:** With many features potentially contributing to the noise, LightGBM's robustness helps in building models that generalize well to new data.
- **Feature Importance:** LightGBM provides insights into which features most influence housing prices, crucial for understanding real estate market dynamics.
- **Effective in Regression Tasks:** As a regression-focused problem, predicting housing prices aligns well with LightGBM's strengths in handling such tasks efficiently.

### C. Approach

Experiment Description	Results
<b>Experiment 1: Baseline Model</b> <ul style="list-style-type: none"><li>• Ran a basic LightGBM model with default hyperparameters and no feature engineering</li></ul>	Private Score: 0.33049 Public Score: 0.32824
<b>Experiment 2: Hyperparameter Tuning</b> <ul style="list-style-type: none"><li>• Used Optuna for hyperparameter tuning</li></ul>	Private Score: 0.32651 Public Score: 0.32578

<b>Experiment 3: Magic Numbers without Hyperparameter Tuning</b> <ul style="list-style-type: none"> <li>• Uses baseline model</li> </ul>	Private Score: 0.31652 Public Score: 0.31586
<b>Experiment 4: Magic Numbers with Feature Selection</b> <ul style="list-style-type: none"> <li>• Used Optuna for hyperparameter tuning, 100 rounds</li> <li>• Early stopping</li> <li>• Selected top 50 features for training using a preliminary LightGBM model</li> </ul>	Private Score: 0.31433 (46th Place) Public Score: 0.31163
<b>Experiment 5: K-folds Cross Validation</b> <ul style="list-style-type: none"> <li>• Used Optuna for hyperparameter tuning, 100 rounds</li> <li>• Early stopping</li> <li>• Selected top 40 features for training using a preliminary LightGBM model</li> <li>• Magic Numbers</li> <li>• Number of folds: 7</li> </ul>	Private Score: 0.31539 (Top 2.6%) Public Score: 0.31121 (Top 23%)

**a. Experiment Details and Notes:**

- Cleaned vs Raw Dataset

The performance of LightGBM was better whenever an 'uncleaned' dataset was used, basically using train.csv and test.csv with some cleaning and feature engineering that is separate from cleaned\_train.csv/cleaned\_test.csv and cleaned\_train\_2.csv/cleaned\_test\_2.csv.

LightGBM can handle missing values natively. It finds the best split by considering the missing values or by treating them as a separate category. Imputation might introduce bias or artificial patterns, especially if not done correctly. In some cases, LightGBM's native handling of missing values might be more effective than imputed data.

LightGBM might also be able to handle minor noise and outliers more effectively due to its robustness. Over-cleaning might remove certain variances that the model could have used.

- Hyperparameter Tuning



The library Optuna was used for hyperparameter tuning. After a fixed number of iterations, the trial with the best hyperparameters is then chosen.

- Insert hyperparameters
- K-fold Cross Validation

In K-fold cross-validation, the dataset is divided into 'K' equal-sized folds or subsets. For each fold, the model is trained on 'K-1' folds and validated on the remaining fold. This is repeated 'K' times, with each of the 'K' folds used exactly once as the validation data. It helps the model reduce overfitting, with robust validation, more comprehensive training, and better hyperparameter tuning. It helps in building a model that not only performs well on the training data but also generalizes effectively to new, unseen data, thereby potentially increasing its accuracy and reliability in real-world scenarios.

#### D. Final Results

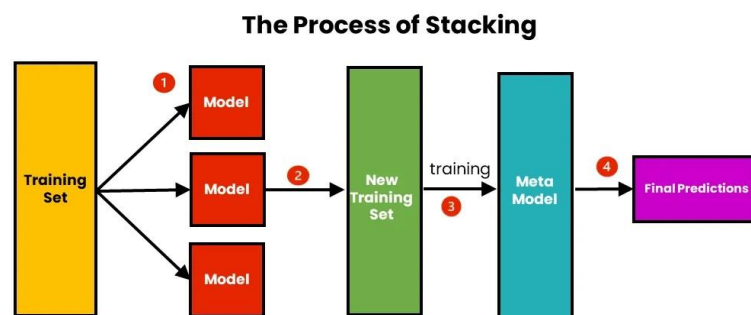
The results of the model are as follows:

- Train RMSE: 0.24374
- Test/Validation RMSE: 0.25363
- Kaggle Private Score: 0.31539
- Kaggle Public Score: 0.31121

### Novelty: Adoption of Ensemble Learning (Stacking)

The prediction of Sberbank Russian Housing Market fluctuations was approached with an innovation that stepped beyond the bounds of conventional single-model methods. Ensemble learning was chosen for its robustness, and a strategy was devised that synthesized the predictions from a variety of models. Each model brought its strengths, contributing to a cultivated collective model with enhanced overall performance.

#### A) Overview



Stacking, a machine learning ensemble technique also called Stacked Generalization, combines multiple models to enhance the overall performance of the model. The main idea behind stacking is to feed the predictions from many base models into a meta-model, also called a blender, a higher-level model that then mixes the predictions to produce the final forecast.

## B) Motivation

The stacking approach was selected because it is known for improving prediction accuracy. The choice of stacking was influenced by its ability to combine various models, taking advantage of their strengths and reducing their weaknesses. This is particularly useful with the Sberbank Russian Housing Dataset, given its complexity with many different features and potential non-linear relationships.

## C) Approach

- i) **Base Models Selection:** Following the data setup, six primary models were selected, namely XGBoost, LightGBM, Random Forest, Extra Trees, AdaBoost, and Gradient Boosting Regressor. Each of these models underwent separate training on the entire dataset. The rationale for choosing a broad array of models was to encompass a wide array of data patterns, utilizing the distinct strengths and data processing methods inherent to each model. For instance, models like Random Forest and Extra Trees, which are tree-based, are known for their proficiency in uncovering non-linear relationships. In contrast, gradient-boosting models like XGBoost and LightGBM are well-suited for handling diverse structured data types. Each model was independently trained to maximize its potential in understanding and processing the data.
- ii) **Predictions and Meta-Model Formation:** Post the training of the primary models, their predictions were compiled. These predictions then served as novel features for the meta-model. To ensure the robustness and general applicability of these predictions, especially to data not previously seen, a five-fold cross-validation method was employed. This phase was crucial in reinforcing the trustworthiness of the entire model ensemble. Subsequently, a meta-model, specifically an XGBoost with a linear booster, was trained using the new dataset composed of the base models' predictions. This meta-model was integral to the ensemble, tasked with amalgamating the predictions from the base models. It adeptly learned to merge these predictions in a way that counterbalanced the weaknesses of

some models with the strengths of others. This extra layer of modeling allowed for a more intricate and refined interpretation of the outputs from the base models.

- iii) **Ensemble Performance:** The concluding ensemble model, which combined the individual base models with the meta-model, exhibited superior performance. This improved performance is credited to the varied modeling techniques and the enhanced predictive capacity of the meta-model. The success of this ensemble underscores the efficacy of employing a layered and comprehensive approach in predictive modeling, particularly in complex datasets such as the Sberbank Russian Housing dataset.

## D) Results

```
params1 = {'eta':0.05, 'max_depth':5, 'subsample':0.8, 'colsample_bytree':0.8, 'min_child_weight':1,
           'gamma':0, 'silent':1, 'objective':'reg:linear', 'eval_metric':'rmse'}
xgb1 = XGBRegressor(params1)

params2 = {'booster':'gblinear', 'alpha':0,
           'eta':0.1, 'max_depth':2, 'subsample':1, 'colsample_bytree':1, 'min_child_weight':1,
           'gamma':0, 'silent':1, 'objective':'reg:linear', 'eval_metric':'rmse'}
xgb2_meta = XGBRegressor(params2)

params_lgb = {'objective':'regression', 'metric':'rmse',
              'learning_rate':0.05, 'max_depth':-1, 'sub_feature':0.7, 'sub_row':1,
              'num_leaves':15, 'min_data':30, 'max_bin':20,
              'bagging_fraction':0.9, 'bagging_freq':40, 'verbosity':0}
lgb1 = LGBRegressor(params_lgb)
```

The following hyperparameters were used for the different models, and the results of the final stacking model are as follows:

- Train RMSE: 0.2467
- Test/Validation RMSE: 0.2438
- Kaggle Private Score: 0.3159
- Kaggle Public Score: 0.31422







## Evaluation:

The evaluation of various models approached is described in the table below:

Model Performance on Train and Cross-Validated Data

Model	Train RMSE	Test/Validation RMSE
Lasso Regression	0.5960	0.8028
Ridge Regression	0.4969	0.5837
Random Forest Regressor	0.3897	0.4709

<b>XGBoost</b>	0.2290	0.2507
<b>LightGBM</b>	0.24374	0.25363
<b>Ensemble Learning</b>	0.2467	0.2438

Model Name	Private Evaluation Score	Public Evaluation Score	Screenshot
<b>Lasso Regression</b>	0.8863	0.88576	 Lasso Regression.csv Complete (after deadline) · now 0.8863 0.88576
<b>Ridge Regression</b>	0.54656	0.52837	 output_file2 (1).csv Complete (after deadline) · 10h ago 0.54656 0.52837
<b>Random Forest Regressor</b>	0.34583	0.34517	 output_file3.csv Complete (after deadline) · 24m ago 0.34583 0.34517
<b>XGBoost</b>	0.31854	0.31746	 Final Magic XGBoost.csv Complete (after deadline) · now 0.31854 0.31746 <input type="checkbox"/>
<b>LightGBM</b>	0.31539	0.31121	 lgb_20170501_optimized_kfold (4).csv Complete (after deadline) · 1h ago 0.31539 0.31121
<b>Ensemble Learning (Stacking)</b>	0.3159	0.31422	 Ensemble-Stacking.csv Complete (after deadline) · now 0.3159 0.31422 <input type="checkbox"/>

## Challenges of the Problem

- **Missing Data:** There is a large no. of missing values in the Dataset. Handling missing data appropriately is crucial for accurate analysis and modeling.
- **Outliers:** Outliers in real estate data may arise from various factors, such as unique properties, errors in data collection, or extreme market conditions. Identifying and handling outliers is important to prevent them from disproportionately influencing the analysis.

- **Model Selection:** Model selection is made difficult because of anonymized features. Therefore, multiple models had to be built and compared to select the best one.

## Conclusion:

This machine learning project has been invaluable in comprehending the intricate steps within the ML workflow. Our key learnings encompass several essential aspects:

Exploratory Data Analysis (EDA) and data cleaning have been foundational. They not only facilitate a deep examination of the data but also aid in understanding inter-feature relationships. Removing null values and scaling significantly enhance data quality, enabling better visualization through EDA and subsequent Feature Engineering. Feature Engineering has been instrumental in uncovering complex patterns, often hidden. Techniques like aggregation and the incorporation of Date Time features were particularly impactful, contributing to a refined prediction model.

We also understood the importance of domain knowledge which can greatly affect our model accuracy scores. In this dataset, there are wrong prices for certain real-estate listings, potentially because of Tax Evasion Schemes in our target columns. We had to deduce a method to eliminate these illegal listings from the train, test, and any future data through the use of magic numbers and that greatly improved our results accuracy. Our understanding of the pivotal role of domain knowledge in influencing model accuracy was a significant revelation. Detecting potentially incorrect prices in real-estate listings—potentially linked to Tax Evasion in the 'price\_doc' column—prompted us to deduce a method to eliminate these problematic entries from our datasets. This strategic application of 'magic numbers' notably enhanced the accuracy of our results.

Our strategic approach involved deploying a diverse range of models—Lasso Regression, Ridge Regression, Random Forest, XGBoost, LightGBM, and a bespoke ensemble model crafted via stacking. Rigorous hyperparameter tuning and comparative analysis unearthed invaluable insights. LightGBM emerged as the standout performer, achieving an exceptional **public score of 0.31121**, positioning us among the **top 23%** of performers.

This Kaggle competition has immensely enriched our team's understanding of the complete machine-learning workflow. Our journey spanned various pivotal stages, commencing with exploratory data analysis and encompassing data preparation, including data preprocessing, feature engineering, hyperparameter tuning, model training, and comprehensive evaluation. This experience has equipped us with enhanced insights into navigating diverse machine-learning challenges effectively.

## References:

- [1] S. Dobilas, 'XGBoost: Extreme Gradient Boosting — How to Improve on Regular Gradient Boosting?', Towards Data Science, 2021.  
<https://towardsdatascience.com/xgboost-extreme-gradient-boosting-how-to-improve-on-regular-gradient-boosting-5c6acf66c70a> (Accessed 7 Nov 2022)
- [2] A. Chavakula, 'What's this "Magic numbers" business?', Kaggle, 2017.  
<https://www.kaggle.com/c/sberbank-russian-housing-market/discussion/35044> (Accessed 7 Nov 2022)
- [3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD.
- [4] Soni, B. (2023, May 2). Stacking to Improve model Performance: A comprehensive guide on ensemble learning in Python. Medium.  
[https://medium.com/@brijesh\\_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28](https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28)