

**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

**CE/CZ4041: Machine Learning
Group Project**

Topic: Sberbank Russian Housing Market

Best Result: Rank 747 (Top 23%)

Group 19

Team Member	Matriculation Number	Contribution
Chen Wei May	U2020687E	Data Processing and Modelling
Goel Tejas	U1923301G	Exploratory Data Analysis and Modelling
Surawar Sanath Sachin	U1922529K	Feature Engineering, Data Cleaning and Modelling
Gupta Suhana	U1923230B	Modelling
Lumlertluksanachai Pongpakin	U2023344C	Modelling

21 April 2023

Introduction

The Sberbank Russian Housing Market dataset is a comprehensive and extensive source of data that offers valuable insights into the dynamics of the Russian real estate market from 2011 to 2016. Compiled by Sberbank, one of the largest banks in Russia, this dataset contains a wide range of features related to residential properties, including location, size, condition, and various economic indicators.

With over 300 features and many samples, this dataset presents ample opportunities for feature engineering, data visualisation, and machine learning model development. By leveraging this dataset, data scientists can gain valuable insights into the factors that influence housing prices in Russia, build accurate predictive models for estimating property prices, and make informed decisions in the real estate market.

In this report, we will delve into the details of the Sberbank Russian Housing Market dataset, thoroughly analyse its features, perform comprehensive data pre-processing, and feature engineering, and develop machine learning models to estimate housing prices in Russia. The goal of this report is to provide a comprehensive and insightful analysis of the dataset, showcasing the potential of machine learning in predicting property prices and its implications in the real estate market and to evaluate the performance of different machine learning models in predicting the sale price of properties based on the features provided in the dataset.

Exploratory Data Analysis

EDA is a data analysis approach that helps in understanding the data by identifying patterns, trends, relationships, and anomalies. It involves visual and quantitative methods to summarise, inspect and understand the main characteristics of the data.

The goal of EDA is to gain insights and understanding of the data, to identify problems or anomalies that may be present, and to develop an intuition about the underlying structure of the data. EDA helps in making informed decisions about data cleaning, pre-processing, feature selection, model selection, and hyperparameter tuning.

Description of datasets

The dataset contains information on the residential real estate market in Russia from August 2011 to June 2015. It includes a total of 30471 samples, with 292 variables in the training set and 291 variables in the test set. The data is structured into several tables, including a main transactional table with detailed information on each transaction, as well as supplementary tables with macroeconomic indicators, regional statistics, and other relevant data.

The main transactional table contains information about individual residential properties, such as their location, area, number of rooms, floor number, and building type, as well as the transaction date, the price in Russian rubles, and other details. The supplementary tables contain additional information about macroeconomic indicators such as GDP, inflation, and oil prices, as well as regional statistics such as population demographics and employment rates.

EDA for Sberbank Russian Housing Data:

Exploratory data analysis for this project was carried out in the following manner.

Firstly, the dataset was checked for missing data, and it was found that of 292 columns, 51 columns have missing values. The percentage of missing values varies from 0.08% to 47.4% across features. Most notably, 'hospital_beds_raion', 'build_year' and 'state' features had the highest percentages of missing values.

Next, the dataset was checked for invalid values and value counts of the features were analysed. It was observed that there were certain features which had invalid values. For example, where 'floor' (home floor) was larger 'max_floor' (~1500 instances).

Analysis of housing parameters

The median house prices were plotted against time ('yearmonth') from October 2011 until June 2015, and it is observed that there is an overall increasing trend of house prices.

To take into consideration the contribution of size of houses to the house price variation, the median 'price_per_sq' was plotted against time ('yearmonth').

A significant decline in 'price_per_sq' was observed from July 2012 to October 2012. Since then, there has been a generally increasing trend for 'price_per_sq'.

As the housing districts could contribute significantly to the house prices, a bar plot was plotted for the average house price for each district. It is observed that Hamovniki district has the highest average house price and Poselenie Klenovskoe has the lowest.

Additionally, the average selling price of houses on each weekday was analysed via 7 subplots of average house prices of each district on the respective days of the week. It was observed that the highest average selling price was on Sunday at Arbat district. For most of the districts, the average selling price of the house was less on weekends compared to other days.

The house prices were analysed for any relation to the floor (number/level) of the apartment via a bar plot of house prices against 'floor' (floor number). An interesting observation was made that houses located on floor 33 have the highest average house price.

The joint plot between house price ['price_doc'] and the total area ['full_sq'] indicates the general trend that as total area of the house increases, house price also increases.

There are exceptions for cases where the houses are in the outskirts/relatively remote and not city areas, which have house prices that do not seem to justify the large total area. The converse is also true for houses located close to the city centre. For some such houses, even though the total area of the house is relatively small, it will still command a higher price in the market than their size seems to justify. There are a few data instances which represent this behaviour. This indicates that locality has some effect on house prices.

The joint plot between house price ['price_doc'] and the living area ['life_sq'] also exhibits a similar trend as the abovementioned trend between house price and total area.

Data Pre-processing

Before training the models, we performed some data pre-processing steps, including data cleaning and feature engineering. We removed missing values, transformed categorical variables into dummy variables, and created new features by combining existing ones.

Data cleaning

Invalid Data Handling

The column 'state' should have discrete values between 1 and 4. There is one row having 33, which is reasonably assumed to be a data entry error and was probably meant to be 3. Hence, we fix that by replacing it with value 3.

'build_year' has a few erroneous values including: 20052009, 0, 1, 3, 20, 70, 215. It is unclear what each of these are truly supposed to be. 20052009 value was replaced with 2007 (in accordance with Kaggle discussions). The remaining values are replaced with the median of 'build_year' data. We choose not to drop these rows as they are a significant proportion (>40%) of training data.

Missing value imputation

The final dataset that was used for modelling still had missing values which had to be imputed for. For missing value imputation, median and mean.

Median imputation is a simple imputation method that replaces missing values with the median of the non-missing values in the same column. While this method is easy to implement and

computationally efficient, it has some limitations. For example, it can lead to biased estimates of the true distribution of the data, and it does not consider any relationships between the missing values and the other variables in the dataset.

KNN imputation, on the other hand, is a more advanced imputation method that uses a nearest neighbours' algorithm to estimate missing values. It considers the relationships between the missing values and the other variables in the dataset and can lead to more accurate imputations than median imputation, especially when the proportion of missing values is large. However, KNN imputation can be computationally intensive and may require more careful tuning of parameters to achieve optimal performance.

As such, on noting that most columns had missing values below 30%, median imputation was used for replacing missing values. For example:

Column Name	% missing
provision_retail_space_modern_sqm	0.98
provision_retail_space_sqm	0.85
museum_visitis_per_100_cap	0.64
load_of_teachers_preschool_per_teacher	0.64
students_reg_sports_share	0.64
theaters_viewers_per_1000_cap	0.64
hospital_beds_raion	0.47
cafe_avg_price_500	0.43
cafe_sum_500_min_price_avg	0.43
cafe_sum_500_max_price_avg	0.43
build_year	0.38
state	0.37

Data Scaling

Min-Max scaling is a normalisation technique that enables us to scale data in a dataset to a specific range using each feature's minimum and maximum value. MinMaxScaler was used for the dataset as it does not change the shape of data, while keeping the data within the range of 0 and 1.

Feature Engineering

The following section describes the feature selection and feature extraction of the

Feature Selection (choosing subset of features)

Certain prominent features were selected according to the following categories: within house area, geographic locality, remoteness, nearby environmental facilities, lifestyle facilities, school facilities and distance from city center.

within house area

1. 'life_sq',
2. 'full_sq',
3. 'num_room'

geographic locality

4. 'state',

remoteness

5. 'sub_area',

nearby environmental facilities

6. 'nuclear_reactor_km'

lifestyle facilities,

7. 'university_km'
8. 'sport_objects_raion'

distance from city center

9. 'Kremlin_km'

Macroeconomic features considered include: 'cpi', 'gdp_annual_growth', 'usdrub' 'invest_fixed_capital_per_cap' because we assessed that these were the more prominent ones that would likely impact the house prices in the market based on the economic landscape.

Some important features we were able to extract using Recursive Feature Elimination with Cross Val scores were:

'full_sq', 'num_room', 'state', 'culture_objects_top_25',
'build_count_monolith', 'ttk_km', 'railroad_km',
'cafe_count_500_na_price', 'cafe_count_1000_price_high',
'trc_count_1500', 'cafe_count_1500_price_1500', 'cafe_count_2000',
'cafe_sum_2000_max_price_avg', 'cafe_count_2000_price_2500',
'cafe_count_2000_price_4000', 'cafe_count_3000',
'cafe_count_3000_price_2500', 'cafe_count_3000_price_4000',
'leisure_count_3000', 'sport_count_3000', 'cafe_count_5000',
'cafe_count_5000_price_1500', 'cafe_count_5000_price_2500',
'cafe_count_5000_price_high', 'church_count_5000'

Feature Extraction (new processed features)

```
# Creating new features
mergeData["avg_dist_to_amenities"] = (mergeData["park_km"] + mergeData["fitness_km"] + mergeData["swim_pool_km"] + mergeData["stadium_km"]) / 4
mergeData["area_density"] = mergeData.apply(lambda x: x['raion_popul']/x['area_m'], axis=1)
mergeData["area_diff_sq"] = mergeData.apply(lambda x: x['full_sq'] - x['kitch_sq'], axis=1)
mergeData["median_max_floor"] = mergeData.groupby('sub_area').max_floor.transform(lambda x: x.median())
mergeData["max_floor_ratio"] = mergeData.apply(lambda x: x['max_floor']/x['median_max_floor'] if x['median_max_floor']!=0 else 1, axis=1)
mergeData["utility_sq"] = mergeData.apply(lambda x: x['full_sq']-x['life_sq'], axis=1)
mergeData["life_sq_ratio"] = mergeData.apply(lambda x: x['life_sq']/x['full_sq'], axis=1)
mergeData["room_size"] = mergeData.apply(lambda x: x['life_sq']/x['num_room'] if x['num_room']!=0 else 0, axis=1)
mergeData["rel_floor"] = mergeData.apply(lambda x: x['floor']/x['max_floor'] if x['max_floor']!=0 else 0, axis=1)
```

We created new features as can be seen above based on the exploratory data analysis since it revealed to have higher positive correlation with the sale price of the house. In specific, we created:

1. **avg_dist_to_amenities:** This new feature is the average distance of a property to four amenities: park, fitness center, swimming pool, and stadium. This feature is relevant because the proximity of amenities to a property can impact its desirability and value. For example, a property located near a park, fitness center, or swimming pool may be more attractive to potential buyers or renters.
2. **area_density:** This new feature is the population density of the area surrounding the property. It is calculated by dividing the population of the region (raion_popul) by its area (area_m). This feature is relevant because population density can impact property prices, with more densely populated areas potentially leading to higher prices due to increased demand.
3. **area_diff_sq:** This new feature represents the difference between the full area of a property and its kitchen area. This feature is relevant because the ratio of living area to kitchen area can impact the value of a property, with larger living areas relative to the kitchen potentially leading to higher prices.
4. **median_max_floor:** This new feature represents the median maximum floor of properties within a specific sub_area. This feature is relevant because the maximum floor of a property can impact its value and desirability, with higher maximum floors potentially leading to better views and more natural light.
5. **utility_sq:** This new feature represents the difference between the full area of a property and its living area. This feature is relevant because it can impact the perceived value of a property, with larger living areas relative to the total area potentially leading to higher prices.
6. **life_sq_ratio:** This new feature represents the ratio of the living area to the full area of a property. This feature is relevant because it can impact the perceived value of a property, with larger living areas relative to the total area potentially leading to higher prices.
7. **room_size:** This new feature represents the size of each room in a property. It is calculated by dividing the living area by the number of rooms in the property. This feature is relevant because it can provide an indication of the overall layout and functionality of a property, with larger rooms potentially leading to higher prices.

8. **rel_floor:** This new feature represents the floor of a property relative to its maximum floor. This feature is relevant because it can impact the perceived value of a property, with higher floors potentially leading to better views and more natural light, but lower floors potentially being more desirable for some buyers or renters.

Modelling

We trained several regression models on the preprocessed data, including:

Decision Tree, Random Forest, XGBoost, LightGBM, Gradient Boosting Regressor, BaggingRegressor, K Nearest Neighbours, etc

We evaluated the performances of these models using root mean squared error (RMSE) as the metric. The models were trained on a subset of the data (80%) and evaluated on the remaining data (20%).

Stacking Regressor

A Stacking Regressor is a type of ensemble learning algorithm that combines multiple regression models to make a more accurate prediction. It is based on the concept of stacking or meta-learning, where the predictions of multiple base models are combined to generate a more accurate final prediction.

The Stacking Regressor works by training multiple base regression models on the training data. These models can be of different types and have different hyperparameters, allowing for a diverse set of models to be used. The predictions of these models are then combined to create a new feature matrix, which is used as input to a meta-regressor.

The meta-regressor is a final regression model that takes the combined predictions of the base models as input and produces the final prediction. It can be any regression model such as linear regression, random forest, or support vector regression.

The Stacking Regressor uses cross-validation to train the base models and the meta-regressor. During each iteration of the cross-validation process, the training data is split into K folds. K-1 folds are used to train the base models, and the remaining fold is used to generate predictions that are combined to create the new feature matrix. This process is repeated for each fold, and the resulting predictions are used to train the meta-regressor.

Two stacking regressor models were experimented on:

1. 6 stacking regressor model: includes RandomForest Regressor, SVM Regressor, XGBoost, LightGBM, Gradient Boosting Regressor, Linear Regression
2. 8 stacking regressor model: includes AdaBoost Regressor, SVM Regressor, ExtraTrees Regressor, Linear Regression, XGBoost, LightGBM, Gradient Boosting Regressor

Both the above stacking regression models use XGBoost as the meta-regressor to produce final prediction. The models were trained using 5-fold cross validation.

Most models were trained with default parameters. Some noteworthy hyperparameters for the above models are shown below:

- XGBoost Regressor (base model): `colsample_bytree: 0.8`, `eta: 0.05`, `max_depth: 5`, `num_boost_round: 10000`
- LGBM Regressor: `learning_rate: 0.05`, `num_boost_round: 10000`, `bagging_freq: 40`, `bagging_fraction: 0.9`, `max_bin: 20`, `num_leaves: 15`, `feature_fraction: 0.7`
- RandomForest and ExtraTrees regressors: `n_estimators: 500`
- XGBoost Regressor (meta model): `eta: 0.1`, `max_depth=2`, `num_boost_round: 10000`

Note: Each training iteration for the stacking regressor took ~2 hours on our machines; hence, hyperparameter tuning using Grid Search / Randomized Search was not possible. The parameters selected above were based on our intuition and readings from Kaggle discussions.

XGBoost

XGBoost (Extreme Gradient Boosting) is a popular and powerful algorithm for supervised learning problems, particularly for regression and classification tasks. It is an optimised distributed gradient boosting library that uses a combination of tree-based models and gradient boosting algorithms to create accurate and scalable models for large-scale datasets.

XGBoost works by creating an ensemble of decision trees, where each subsequent tree tries to correct the errors of the previous trees. It applies gradient descent to optimise a loss function, which measures the difference between the predicted and actual values, to find the optimal values of model parameters.

The algorithm has several key features that make it popular for data science projects, including its ability to handle missing values and automatically handle feature selection. It is also fast and scalable, which makes it well-suited for large datasets. Additionally, it provides various hyperparameters for model tuning, which makes it a flexible and powerful tool for building accurate machine learning models.

Experimentation:

The `XGBRegressor()` function from `sklearn` is used to construct the XGBoost model. The hyperparameter `colsample_bytree`, `learning_rate`, `max_depth`, `n_estimators` and `subsample` of the XGBoost must be tuned to obtain the best results. Hence `RandomizedSearchCV()` from `sklearn` is

used. It runs through all the different parameter distributions defined in paramDist to determine the most optimal combination of parameters based on the best score.

Optimal hyperparameters: subsample: 0.5, n_estimators: 100, max_depth: 11, learning_rate: 0.07, colsample_bytree: 1

Results:

MSE: 66909194917074.77

RMSE: 8179804.09

MSLE: 244.40

RMSLE: 15.63

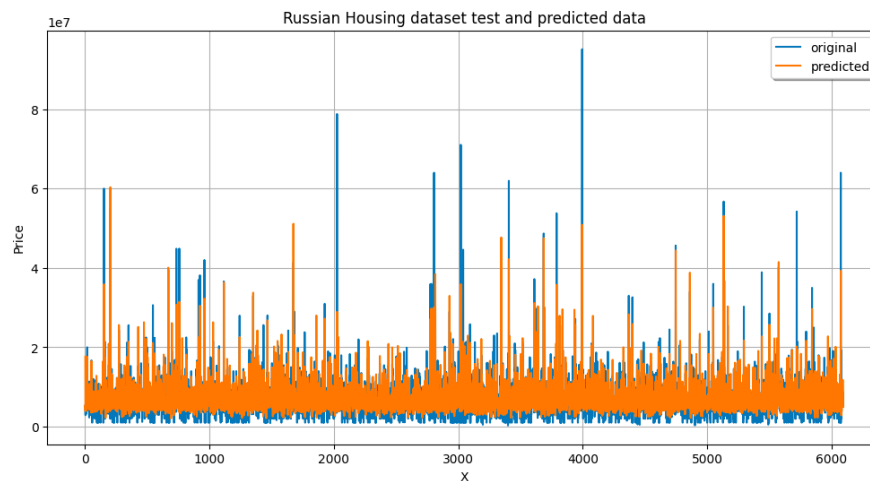


Figure: XGBoost model's train set predicted and original values

SVM Regressor

The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with a few minor differences. Firstly, it is difficult to predict the target value because the expected output is a real number, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which is already set up for the problem. The main idea of SVR is the same as SVM for classification, which is to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

SVM regression is considered a nonparametric technique because it relies on kernel functions. The kernel functions transform the data into a higher dimensional feature space to make it possible to perform the linear separation.

Extra Trees Regressor

ExtraTreesRegressor is a type of ensemble learning algorithm that combines multiple decision tree models to make a more accurate prediction. It can model complex nonlinear relationships between the input features and the target variable in regression problems.

It is a variation of the random forest algorithm, where instead of building a forest of decision trees using bootstrapped samples of the training data, ExtraTreesRegressor selects random subsets of the features and splits each node using random thresholds.

ExtraTreesRegressor can often achieve similar or better performance than the random forest algorithm, although it uses a simpler algorithm to construct the decision trees used as members of the ensemble.

In ExtraTreesRegressor, multiple decision trees are trained on different subsets of the training data and random subsets of the features. The trees are then combined to make a final prediction by averaging their outputs. This averaging helps to reduce the variance of the model and improve its generalization performance.

Adaboost Regressor

AdaBoostRegressor is a type of ensemble learning algorithm that combines multiple weak regression models to make a more accurate prediction. It is an extension of the AdaBoost algorithm, which was originally developed for binary classification problems.

In AdaBoostRegressor, multiple weak regression models are trained on different subsets of the training data, with each model assigned a weight based on its performance. The weights are then used to adjust the contribution of each model to the final prediction.

During each iteration of the algorithm, the training data is reweighted to focus on the examples that were previously misclassified by the weak models, which improves the performance of the subsequent models on these examples, leading to a more accurate prediction.

Hyperparameters that can be tuned to improve its performance include the number of weak models, the maximum depth of each model, and the learning rate.

Decision Tree

Decision Tree is a decision-making tool that uses a flowchart-like tree structure or is a model of decisions and all their possible results. Decision-tree algorithms are a type of supervised learning algorithm. It works for both continuous as well as categorical output variables.

The branches/edges represent the result of the node and the nodes have either:

- Conditions [Decision Nodes], or

- Result [End Nodes]

The branches/edges represent the truth/falsity of the statement and makes a decision based on that. Decision Trees can solve nonlinear problems and work on high-dimensional data with good accuracy. It is also easy to visualise and explain. However, the issue of overfitting may be faced when decision trees are used, which might be resolved by random forest. Also, the structure of the optimal decision tree is susceptible to small changes in the data.

Decision Tree Regression: Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output.

Experimentation:

The `DecisionTreeRegressor()` from `sklearn` is used to construct the decision tree regressor model. The hyperparameter `max_depth` represents the depth of the tree and must be selected by the user. To tune the hyperparameters for the model, `sklearn`'s `GridSearchCV` is utilized. The technique runs through all the different parameters fed by the parameter grid to produce the most optimal combination of parameters based on a scoring metric. The following code shows the list of values for `max_depth` to be passed into the parameter grid for `GridSearchCV`.

Optimal hyperparameters: `max_depth: 5`

Results:

MSE: 64867076486832.85

RMSE: 8054009.96

MSLE: 244.65

RMSLE: 15.64

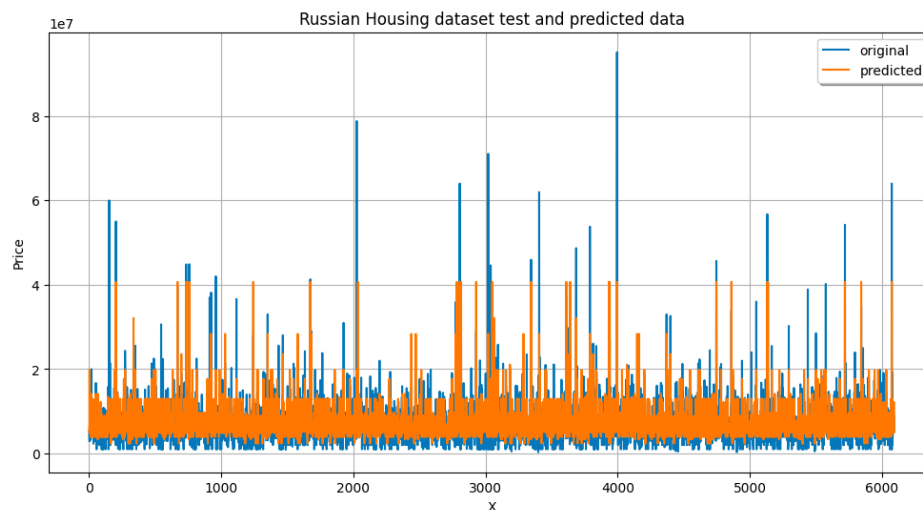


Figure: Decision Tree model's train set predicted and original values

Random Forest

Random Forest is a supervised machine learning algorithm that is commonly used for classification and regression problems. It is an ensemble learning technique that combines multiple decision trees to create a forest. In the case of classification, each tree in the forest predicts the class of an input data point, and the final output is the majority vote of all the trees. In the case of regression, each tree in the forest predicts a numerical value, and the final output is the average of all the trees.

Random Forest is a powerful algorithm due to its ability to handle many features and high-dimensional data. It also performs well on large datasets, is less prone to overfitting compared to a single decision tree and can handle missing data without imputation.

One of the main advantages of Random Forest is its interpretability. It can provide important insights into the relative importance of different features in predicting the target variable. This can be used to identify the most important factors affecting the outcome of the model, and to identify areas where further investigation may be needed.

However, predictions made by random forest are not as easy to interpret as compared to decision tree classifiers. If the data contains groups of correlated features of similar relevance to output, smaller groups are favoured over larger ones. Moreover, they require a significant amount of memory for storage.

Experimentation:

The `RandomForestRegressor()` function from `sklearn` is used to construct the random forest regression model. The hyperparameter `n_estimators` and `max_depth` of the random forest must be tuned to obtain the best result. Hence `RandomizedSearchCV()` from `sklearn` is used. It runs through all the different parameters fed by the `paramDist` to produce the most optimal combination of parameters based on a scoring metric.

Optimal hyperparameters: `n_estimators=300`, `max_depth=15`, `random_state=42`, `n_jobs=-1`

Results:

MSE: 65754349620255.22

RMSE: 8108905.57

MSLE: 244.21

RMSLE: 15.63

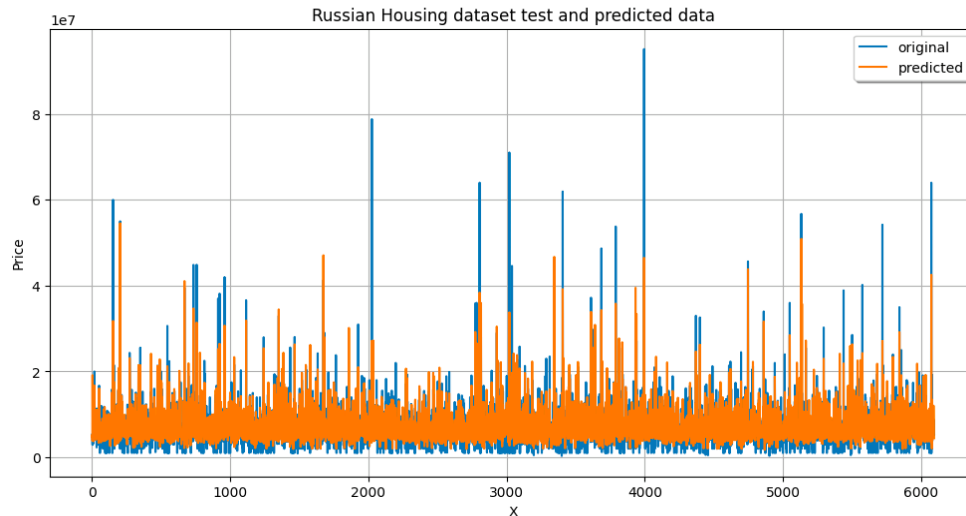


Figure: Random Forest models's train set predicted and original values

LightGBM

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework that uses tree-based learning algorithms. It is efficient, scalable, and accurate, and thus suitable for building models on large-scale datasets. LightGBM uses a gradient-based one-side sampling (GOSS) technique to speed up the training and reduce memory usage. Also, feature values are binned via histogram-based algorithms which improve training speed and model performance.

Some features of LightGBM include its ability to handle large datasets, its fast training and prediction speed, and its high accuracy. It also offers several hyperparameters that can be tuned to optimise model performance, including the learning rate, number of iterations, and maximum depth of the trees.

Experimentation:

The `LGBMRegressor()` function from `lightgbm` is used to construct the LightGBM model. The hyperparameters `subsample`, `reg_lambda`, `reg_alpha`, `num_leaves`, `n_estimators`, `max_depth`, `learning_rate`, `colsample_bytree` of the `lightgbm` must be tuned to obtain the best results. Hence `RandomizedSearchCV()` from `sklearn` is used. It runs through all the different parameter distributions defined in `param_grid` to determine the most optimal combination of parameters based on the best score.

Optimal hyperparameters: `subsample: 0.7`, `reg_lambda: 0.5`, `reg_alpha: 0.1`, `num_leaves: 20`, `n_estimators: 400`, `max_depth: 15`, `learning_rate: 0.05`, `colsample_bytree: 0.5`

Results:

MSE: 67741679408476.59

RMSE: 8230533.36

MSLE: 244.46

RMSLE: 15.64

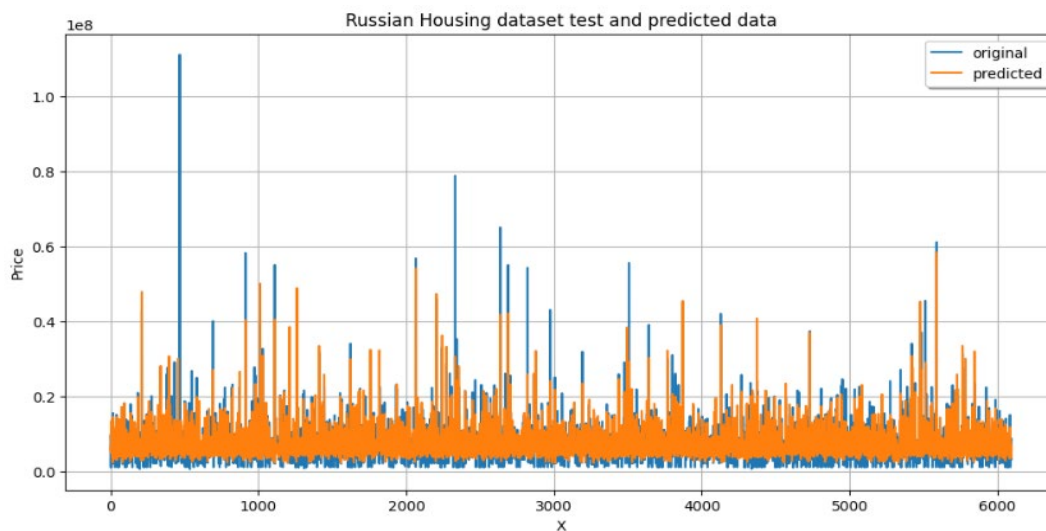


Figure: LightGBM model's train set predicted and original values

Gradient Boosting Regressor

Gradient boosting is one of the variants of ensemble methods where you create multiple weak models and combine them to get better performance. It is a technique for repeatedly adding decision trees so that the next decision tree corrects the previous decision tree error.

Gradient boosting regression is generally used when we want to decrease the Bias error.

It calculates the difference between the current prediction and the known correct target value. This difference is called residual. After that Gradient boosting regression trains a weak model that maps features to that residual. The estimator builds an additive model in a forward stage-wise manner which allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.

The features are always randomly permuted at each split. Therefore, even with the same training data and `max_features=n_features`, the best-found split may vary if the improvement of the criterion is identical for several splits enumerated during the search of the best split. `Random_state` has to be fixed to obtain a deterministic behaviour during fitting.

Optimal hyperparameters: `learning_rate=0.28`, `max_depth=6`, `n_estimators= 266`, `subsample=0.97`

Results:

MSE: 65004994941078.28

RMSE: 8062567.52

MSLE: 244.31

RMSLE: 15.63

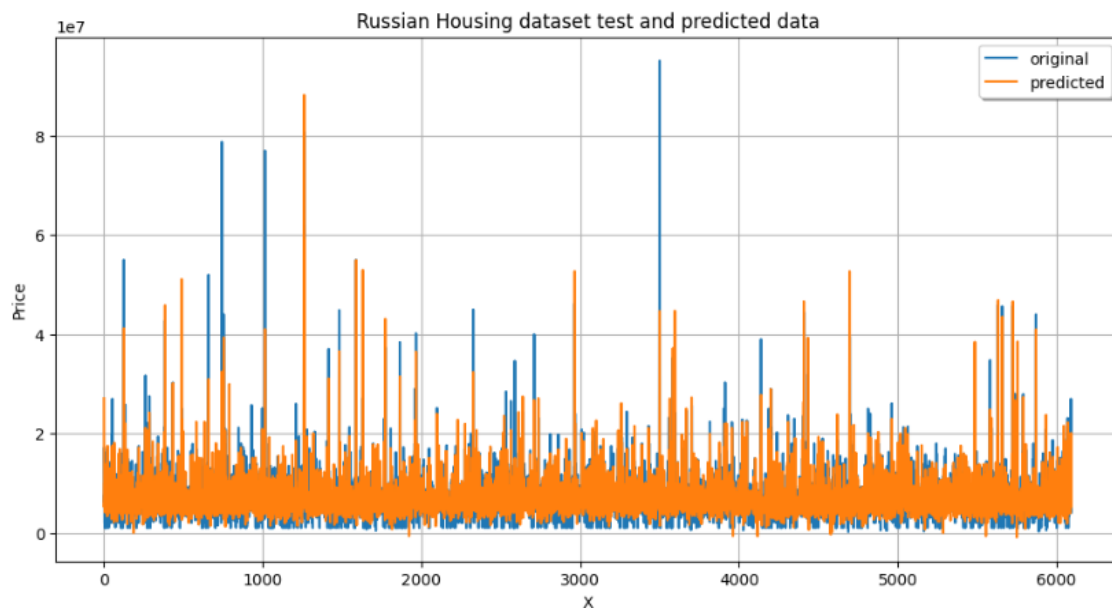


Figure: Gradient Boosting Regressor model's train set predicted and original values

KNearestNeighbours

It is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. The KNN algorithm uses 'feature similarity' to predict the values of any new data points. When using KNearestNeighbours for Regression, we take the k nearest values of the target variable and compute the mean of those values. Those k nearest values act like regressors of linear regression. The average of the values among the K -nearest neighbours to the test instance is the predicted value.

KNearestNeighbours is a lazy learning method where predictions can be made without model training. Its time complexity is $O(n)$ (good efficiency) and can be used for both classification and regression. However, it does not work well with large dataset, is sensitive to noisy data, missing values and outliers. Additionally, there is a need for feature scaling and an important issue for consideration is choosing the correct K value, where K is the number of nearest neighbours whose values are to be taken into consideration when determining the value of the predicted target variable.

Optimal hyperparameters: $K = 7$

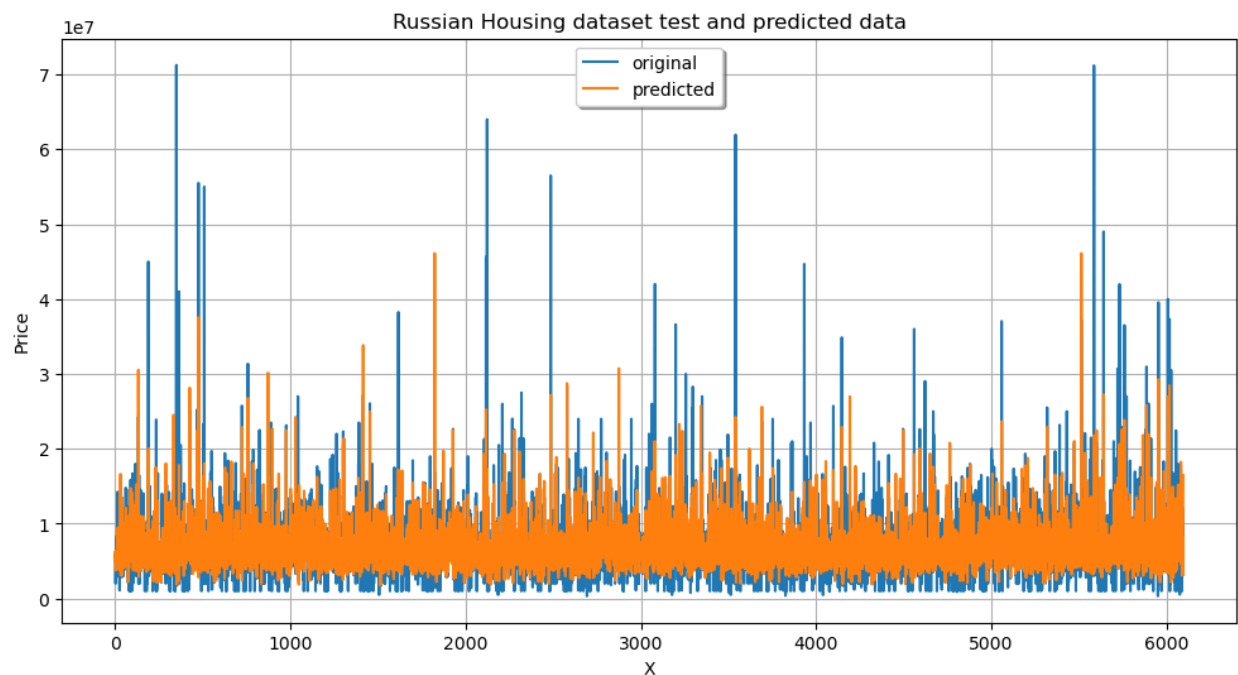
Results:

MSE: 10787243474801.03

RMSE: 3284393.93

MSLE: 0.26

RMSLE: 0.51



KMeans + Linear Regression

KMeans clustering is simple to implement, scales to large data sets, guarantees convergence, easily adopts to new examples and generalises to clusters of different shapes and sizes. However, it is sensitive to outliers and scalability decreases when dimension increases. Additionally, the issue of choosing an appropriate K value (number of cluster centroids) is challenging. Furthermore, KMeans also faces the issue of being sensitive to initial values, as initialization of the K centroids could significantly affect the KMeans clustering result.

After dividing the dataset into the appropriate number of clusters, a simple linear regression model is applied to each cluster for the target value prediction. Linear Regression is a supervised learning algorithm that supports finding the linear correlation among variables. It is an algorithm that provides a linear relationship between an independent variable and a dependent variable to predict the outcome of future events.

Optimal hyperparameters: K = 2

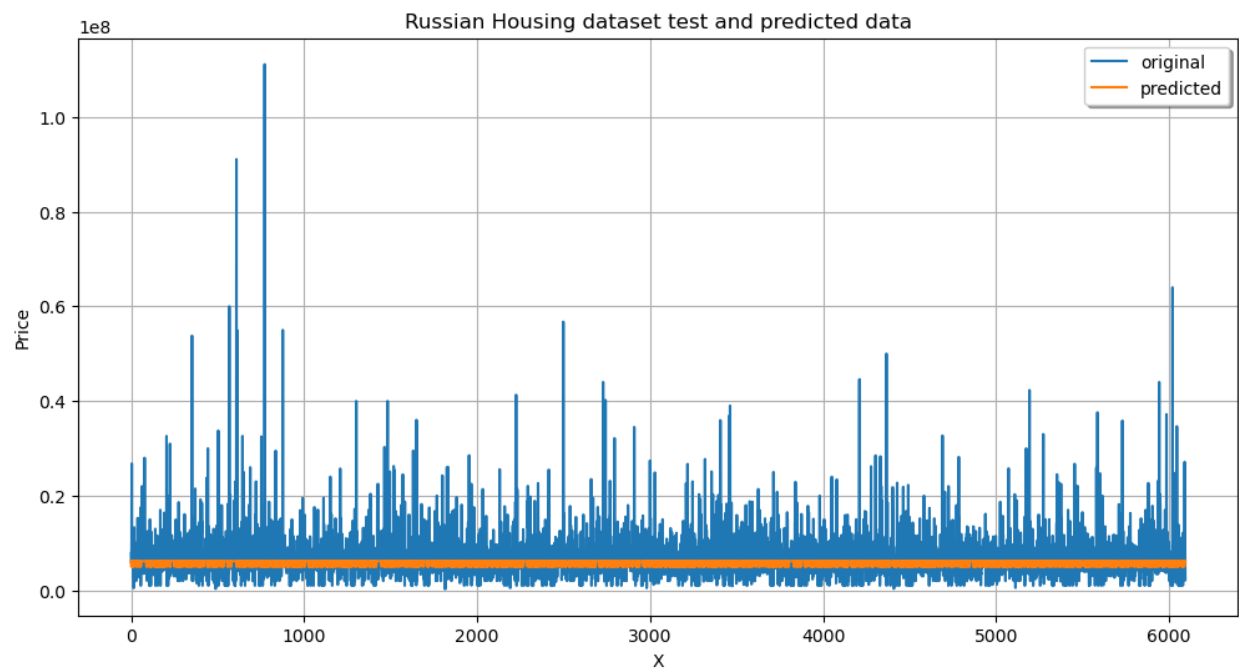
Results:

MSE: 25439724538854.35



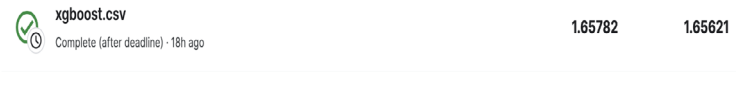


RMSE: 5043780.78





MSLE: 0.35

RMSLE: 0.59



Leaderboard:

Model	Private Score	Public Score	Screenshot
Decision Tree	1.15367	1.16329	
Random Forest	1.77386	1.77086	
XGBoost	1.65782	1.65621	
LightGBM	1.43008	1.42598	
Gradient Boosting Regressor	1.54446	1.54193	

Model	Private Score	Public Score	Screenshot	Rank
KMeans + Linear Regression	0.53329	0.52881		3097/3265 94.85%
K-Nearest Neighbours	0.41785	0.41409		2938/3265 89.98%
8 Stacking regressor model	0.31399	0.31118		747/3265 22.88%
6 Stacking regressor model	0.31969	0.31889		1473/3265 45.11%

Results Discussion

The results show that the Stacking Regressor model (with 8 models comprising of RandomForest Regressor, AdaBoostRegressor, SVM Regressor, ExtraTrees Regressor, Linear Regression, XGBoost, LightGBM, Gradient Boosting Regressor) achieved the best kaggle score of 0.311. The next best performing model was Stacking Regressor model which comprised 6 models gave kaggle score of 0.319. The other models performed comparatively worse.

Conclusion:

In conclusion, we found that the Stacking Regressor ensemble models performed the best in predicting the sale price of properties in the Sberbank Russian Housing Market dataset. These models can be used to provide valuable insights to real estate stakeholders in Russian and other similar markets. Future work can include further feature engineering and exploring more complex models to improve the prediction performance.