DS 675-854 Machine Learning
Jesse Hilario, Sumanth Kota, Kalline Tong
Spring 2024

NYC Housing Prices Project Report

Project video:
https://drive.google.com/file/d/1OT1zGl6UNIk5A8pKDCqJBH5Xn8dQPbIU/view?usp=drive_link

Link to Milestone 2 Proposal: Milestone 2 - DS675854-1.pdf: Kalline Tong (instructure.com)

Our work is based on the New York Housing dataset on Kaggle (https://www.kaggle.com/datasets/nelgiriyewithana/new-york-housing-market). We were interested particularly by the work done in the notebook by Scott Pitcher (https://www.kaggle.com/code/scottpitcher/nyc-housing-project) and found that his results regarding the New York Housing Market were the most informative. He used the XGBRegressor.

The steps that were the most interesting were how he preprocessed his data as well as how he designed his models. We were interested in two questions:

1) Can we find a different model that provides a better result?
2) Will changing the steps for preprocessing the data improve the result?

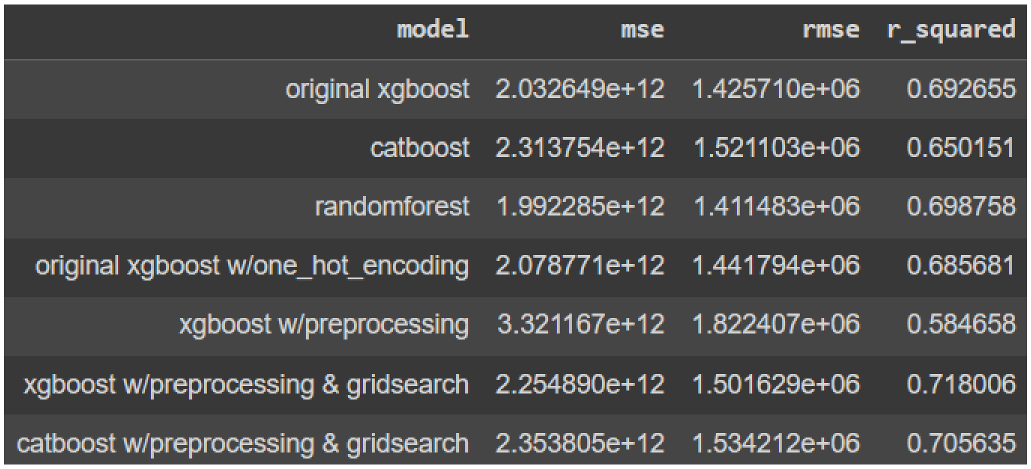
We hoped to answer these questions experimentally in two parts corresponding to each question. For question 1, we would hold each part of the process constant except for the model itself. All data cleaning, preprocessing, feature engineering, and other additional steps would follow exactly Scott Pitcher's approach to provide the most reliable comparison between our models and his. For question 2, we would do essentially the opposite: we would have to hold the model constant in order to compare our group's preprocessing method with Pitcher's.

First, we replicated the model used in a previous notebook (Scott Pitcher's) and were able to replicate his results, getting an identical r-squared value of 0.693. Using the same preprocessing method, we tried different machine learning models to find the best fit on the data. Previous Kaggle notebooks used XGBClassifier, RandomForestClassifier, BaggingClassifier, KNeighborsClassifier, SVC, and LogisticRegression. The untuned models had accuracy ranging from 0.273 for LogisticRegression to 0.492 for XGBClassifier. Since the XGBClassifier had the highest accuracy, it was selected for further hyperparameter tuning by previous notebook authors.

Question 1 is approached from the beginning of the notebook until the section "XGBRegressor using the onehot encoded values." We attempt to answer Question 2 in the section "Keeping the Models constant but changing the preprocessing."

Part 1 - Can we find a different model that provides a better result?

As stated, we copied the notebook from Pitcher from preprocessing to modeling the first xgboost regressor. This ensured that every line of code and every transformation of the data was identical. We do not enter data until the section "Experimenting with other models," where we compare XGBoost to other models.



model	mse	rmse	r_squared
original xgboost	2.032649e+12	1.425710e+06	0.692655
catboost	2.313754e+12	1.521103e+06	0.650151
randomforest	1.992285e+12	1.411483e+06	0.698758
original xgboost w/one_hot_encoding	2.078771e+12	1.441794e+06	0.685681
xgboost w/preprocessing	3.321167e+12	1.822407e+06	0.584658
xgboost w/preprocessing & gridsearch	2.254890e+12	1.501629e+06	0.718006
catboost w/preprocessing & gridsearch	2.353805e+12	1.534212e+06	0.705635

Figure 1. Results from different models. Note: Part 1 uses the results for the first 4 rows. Part 2 uses the final 3 rows.

We experimented with different combinations of hyperparameters, preprocessing methods, and machine learning models, i.e., CatBoost, XGBoost, and Random Forest models. In Figure 1, the *original xgboost* model refers to the model and data processing methods published in Scott Pitcher's notebook, which resulted in an r-squared value of 0.693. *Catboost* (r-squared = 0.650) and *randomforest* (r-squared=0.699) use the same preprocessing methods as the *original* XGBoost model. However, because *randomforest* required the use of onehot-encoding (because Sci-Kit Learn requires numeric inputs, not categorical), we had to first one-hot encode *randomforest*. Because of this, we also modified the *original* XGBoost model, which used label-encoded categorical features, to instead use onehot-encoded categorical features, creating *original xgboost w/one_hot_encoding* and allowing us to compare the two. This increased the original model's r-squared to 0.686. In total, we experimented with 4 different models using Scott Pitcher's preprocessing methods. Out of these models, the *randomforest* model resulted in the highest r-squared value at 0.699.

While this result was higher than both the Catboost and XGBoost regressors, the difference did not appear very significant between the three models. The proportion of variance in price explained by the data for each of these models is between 0.65 and 0.70. Future projects could focus on either tuning the hyperparameters differently or using models that may better generalize with this data.

Part 2 - Will changing the steps for preprocessing the data improve the result?

This section of the notebook starts at “Keeping the Models constant but changing the preprocessing.” In this section, we sought to see if different preprocessing steps could improve the result. There were many other features not used by Scott Pitcher’s notebook that we believed could be used to make predictions. In particular, we noticed that the location data was very messy. The only location columns that appeared reliable were the longitude, latitude, and formatted_address. So, we extracted the zip code from formatted_address. We also used onehot encoding on the categorical features.

We mapped feature correlation in order to experiment with removing highly correlated features. We found that eliminating redundant features improved the r-squared value for all models. Multicollinearity occurs when two or more predictor variables are highly correlated, and can lead to coefficients that are poorly estimated and that exhibit high variance. This housing dataset included separate features for “Address”, “Main address”, “Formatted address”, “Locality”, “Sublocality”, “Street name”, “Long name”, latitude and longitude. By removing redundant features, we significantly improved the accuracy and generalization capabilities of our model. However, we left in place other features that showed lower values of correlation. These features included bedroom and bathroom counts as well as property square footage.

In order to make our results as experimentally sound as we could, similar to Part 1, we had to leave the model constant in order to compare our different approach in preprocessing. This ensures that it is not the model, but the preprocessing itself that is affecting the results. So, we used the same model architecture of xgboost as used by Scott Pitcher. Namely, we used the xgbregressor with the same hyperparameters as our *original xgboost* model, resulting in *xgboost w/preprocessing* (see Figure 1; the parameters in question are held by the variable xg_original_best_params). Following our preprocessing steps, this resulted in a lower r-squared value of .585 compared to the original model. It appears the preprocessing may have made the result worse.

This made sense as the hyperparameters of that model may not have been as good as well-fitting to the results of this newly transformed data. Important to note, this transformed data has many more columns than the original preprocessing by Scott Pitcher, which may also have affected the result. Pitcher only had 4 columns, while the columns from our new dataset included his columns as well as onehot-encoded categorical variables (which result in many columns due to the large number of possible values), zipcode, and broker title.

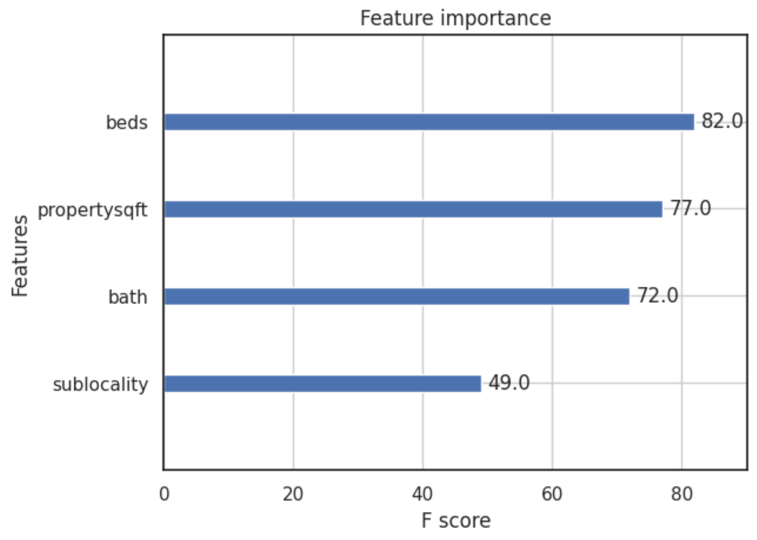
If we were to conduct this experiment again, we may have been extra tedious by comparing the result of these multiple changes only one at a time, which would incrementally show the difference in performance for the model. This would show which changes had the biggest impact.

Additional Results

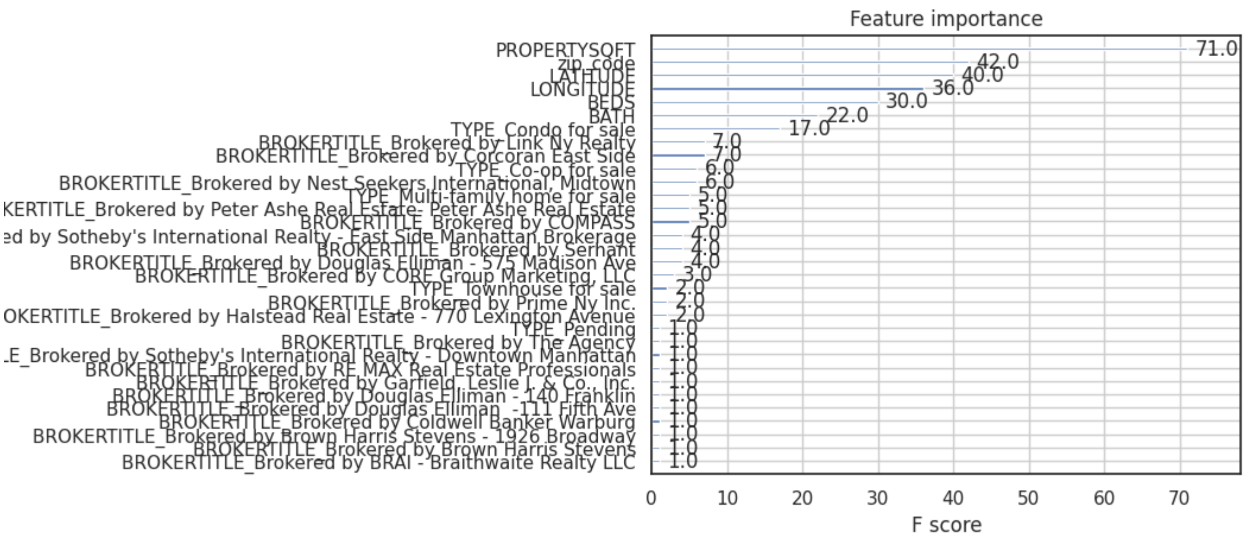
For a sanity check, we included an xgboost model that was allowed to have its hyperparameters tuned, as well as a catboost with similar flexibility. Additionally, we used gridsearch to fine tune hyperparameters for our XGBoost and CatBoost models with redundant features removed. This resulted in the model *xgboost w/preprocessing & gridsearch* which had the highest r-squared value of 0.718 out of all the models. Similarly, the model *catboost w/preprocessing & gridsearch* had an improved r-squared value of 0.706, the second-highest of all the models.

The first challenge was processing and organizing the data. There were mistakes in the data where some values from the .csv file were incorrectly extracted from the original data. We cleaned the original data and extracted the correct values for all features. Additionally, the data had 4444 separate entries and 4240 unique entries. 204 duplicate entries had to be removed. In addition, the dataset contained two separate addresses that had duplicate entries but mismatched valuation. We removed the two duplicate addresses that were listed at an unusual price as error.

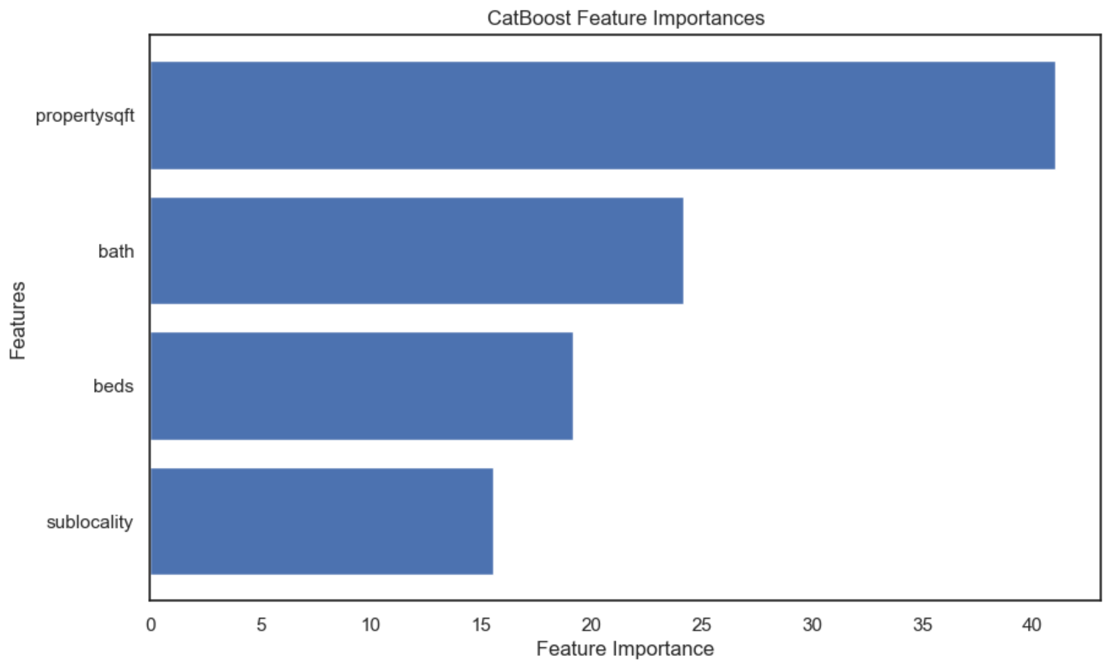
In order to ensure that key features were preserved when eliminating redundant variables, we analyzed feature importance to identify essential features. Different models yielded different results when graphing feature importance.



When analyzed using XGBoost both with and without onehot encoding, the number of bedrooms was the most important feature, followed by property square footage, the number of bathrooms, and sublocality.



When using XGBoost to analyze the preprocessed data with redundant features removed, property square footage was the most important feature, followed by zip code, and latitude and longitude.



The CatBoost model placed the highest importance on property square footage, followed by bathrooms, bedrooms, and sublocality.

The use of tree-based learners allowed us to have these very interesting additional pieces of information from the feature importances. Some interesting patterns were that the square footage of a property (propertysqft) consistently scored high in feature importance among the models, along with the number of bedrooms in some models. In the models where we did feature engineering, the location information proved to be very important; latitude, longitude, and zip code were always among the top 4 most important features. This seems to suggest that location and property square footage are two of the most important factors when it comes to home pricing in New York City.

Additional Thoughts

We attempted to enhance our dataset by extracting the floor number from apartment and condo listings, using the first digit of the apartment number as a potential indicator. However, this method proved ineffective due to the inconsistent representation of floor numbers by the first digit in apartment numbers.

Member Contributions

Kalline Tong

- Experimented with models (XGBRegression, Random Forest Classifier, SVC, Logistic Regression Classifier, k-Nearest Neighbors Classifier)
- Fine tuning hyperparameters using gridsearch
- Created project report

Jesse Hilario

- Assisted with hyperparameter tuning selection for CatBoost and RandomForest using gridsearch, keeping models consistent by tuning similar parameters (e.g., learning_rate in CatBoost and XGBRegressor)
- Zip Code extraction from formatted_addresses, preprocessing, Onehot encoding of variables, cleaning categorical values
- Graphing feature importances and creating results table to track metrics across models

Sumanth Kota

- Helped in data analysis by locating correlated and redundant features, and removing those features.
- Price feature incorrect for a number of rows was rectified.

- Feature extraction through the address column and modeling on the same models for comparison.

An interesting result of using different versions.

When running the same code on Google Colab vs Jupyter notebook on the local machine, the code produced dramatically different results. After taking the professor's suggestion on the issue and changing all the versions to the same version as in Jupyter, the code produced the same result as in Jupyter. It shows that a small difference in the version produced a huge change in the output. The encoding technique also had an impact, only using OneHotEncoder did not produce the result as using LabelEncoder followed by OneHotEncoder.
Result:
Even after changing the **scikit-learn version to the same as in Jupyter which was 1.3.0,** the code produced different results. Realizing that there was some difference in the way the data was being handled, by changing the **pandas version to 2.2.1 and numpy version to 1.26.4** we saw that the results changed to the same as Jupyter. We first had to install the same version as Jupyter, then restart the session, then check if the install was correct using **!pip show <package>** and once the correct version was installed the proper results were revealed. While our video was recorded before we were able to implement this change, it does represent all the work we have done, but not our results. By changing the versions, one will be able to see the difference in result, and it is quite shocking the difference then what is present in our video!