Predictive Modeling of Apartment Sale Prices in Queens, NY

Adam Jablonka

May 26, 2024

Abstract

This paper presents a predictive model for apartment selling prices in Queens, NY, using data from MLSI. The goal is to surpass the accuracy of Zillow's "zestimates" by applying various modeling techniques, including regression trees, linear regression, and random forests. We focus on the sale price as the response variable and utilize features extracted and imputed from the dataset. Our analysis includes data cleaning, featurization, and rigorous out-of-sample validation to ensure the model's reliability for real-world applications.

1 Introduction

Predicting apartment selling prices in Queens, NY, is a challenging task due to the diverse nature of the real estate market. This project aims to build a predictive model that can accurately estimate the sale prices of apartments using historical data. The dataset, sourced from MLSI via Amazon's MTurk, includes various features related to the apartments and their sale details.

The unit of observation in this study is an apartment, and the response variable is the sale price. We employ multiple modeling techniques, including regression trees, linear regression, and random forests, to explore their effectiveness in predicting sale prices. Our performance metrics include R^2 and RMSE, both for in-sample and out-of-sample predictions.

2 The Data

The dataset used in this project comprises historical data of apartment sales in Queens, NY, between February 2016 and February 2017. The data includes various features such as the number of bedrooms, square footage, and maintenance costs. The dataset was preprocessed to handle missing values and convert categorical variables into numerical formats.

2.1 Featurization

We extracted and created several features from the raw data. These include:

- **approx_year_built**: The approximate year the building was constructed.
- **cats_allowed**: Binary feature indicating if cats are allowed.
- **community_district_num**: The community district number.
- **coop_condo**: Type of apartment (co-op or condo).
- **maintenance_cost**: Monthly maintenance cost.
- **num_bedrooms**: Number of bedrooms.
- **num_floors_in_building**: Number of floors in the building.
- **num_full_bathrooms**: Number of full bathrooms.
- **sq_footage**: Square footage of the apartment.
- **walk_score**: Walkability score of the location.
- **zip_code**: Zip code of the apartment.

2.2 Errors and Missingness

The dataset contained several missing values, especially in key features such as **sale_price** and **sq_footage**. Handling these missing values was crucial to ensure the accuracy and reliability of our predictive model.

Handling Missing Sale Price Values The **sale_price** column had a particularly high proportion of missing values, with approximately 76% of the entries missing. Instead of directly imputing these missing values, we used an alternative approach. We leveraged the **listing_price_to_nearest_1000** feature, which had significantly fewer missing values (about 24%).

The rationale behind this approach was based on the assumption that the listing price could serve as a reasonable proxy for the sale price, given the high correlation between the two. This assumption allowed us to fill in the missing **sale_price** values with the corresponding **listing_price_to_nearest_1000** values where available. This strategy ensured that we retained as much of the original data as possible without introducing potential biases that could arise from imputation techniques.

Handling Missing Values in Other Features For other features with missing values, we employed iterative imputation using a **RandomForestRegressor** as the estimator. This method iteratively predicts the missing values using the non-missing values in the dataset, providing a more robust imputation compared to simpler methods like mean or median imputation.

However, certain features exhibited excessively high missingness rates, making imputation impractical. These features included:

- **num_half_bathrooms**: 92% missing
- **pct_tax_deductibl**: 79% missing
- **common_charges**: 76% missing
- **parking_charges**: 75% missing
- **total_taxes**: 74% missing

Due to their high missingness, these features were dropped from the dataset to maintain data integrity and ensure the reliability of our model.

Summary of Missing Values and Imputation Overall, our approach to handling missing values involved a combination of direct substitution and iterative imputation. By using the **listing_price_to_nearest_1000** as a stand-in for the **sale_price** and applying iterative imputation for other features, we aimed to maximize data retention and accuracy. The steps involved in our data preprocessing are outlined below:

- Identified features with high missingness rates and dropped those with impractically high proportions of missing values.
- Used the **listing_price_to_nearest_1000** to fill in missing **sale_price** values.

- Applied iterative imputation using a **RandomForestRegressor** (MissForest) for the remaining features with missing values.

This comprehensive approach ensured that our dataset was as complete and accurate as possible, providing a solid foundation for building our predictive models.

3 Modeling

This section describes the different models used to predict apartment sale prices, including regression trees, linear regression, and random forests.

3.1 Regression Tree Modeling

We fit a regression tree to the dataset to identify the top features influencing apartment sale prices. The top layers of the tree are visualized, and the most important features are identified.

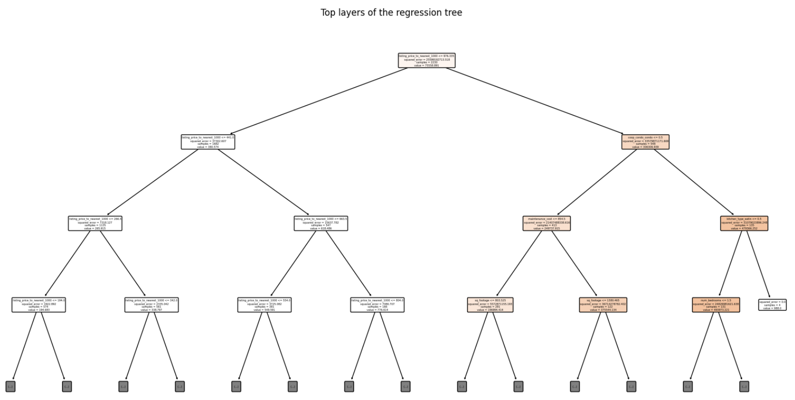

Figure 1: Top layers of the regression tree

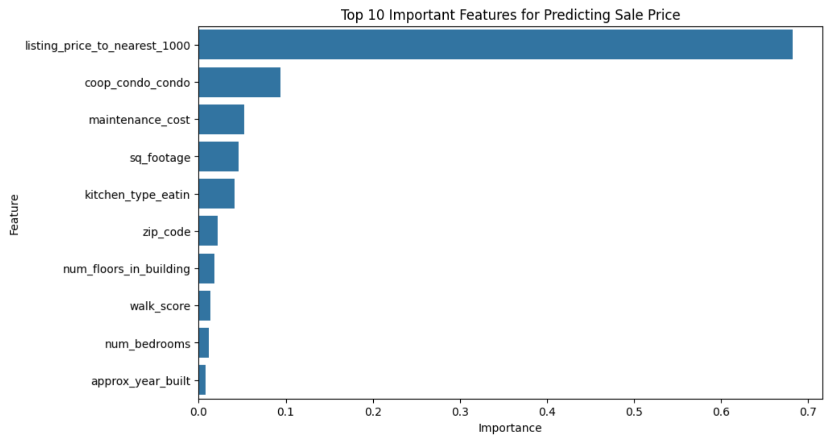

Figure 2: Top layers of the regression tree

The top 10 features important for predicting sale price are:

- listing_price_to_nearest_1000
- coop_condo_condo
- sq_footage
- kitchen_type_eatin
- maintenance_cost
- zip_code
- num_floors_in_building
- num_bedrooms
- walk_score
- approx_year_built

3.2 Linear Modeling

We fit a vanilla OLS linear model and examined its in-sample error statistics. The coefficients for the most important features were interpreted to understand their impact on sale prices.

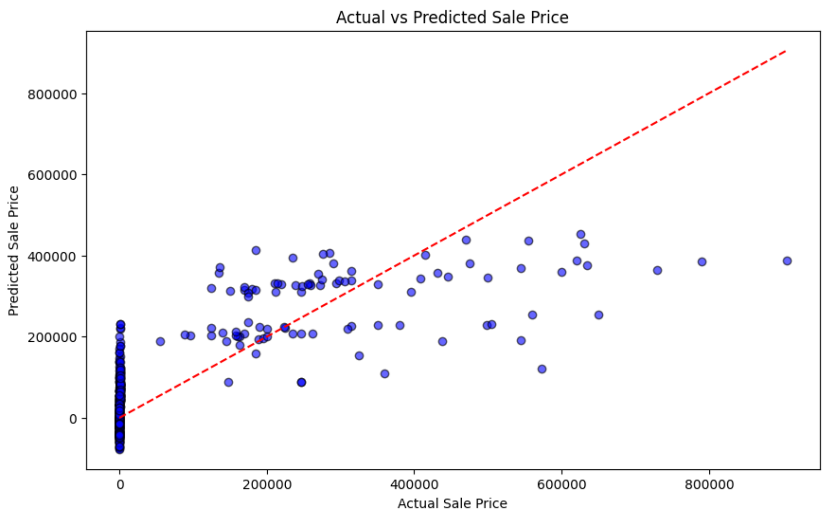

Figure 3: OLS model graph

```
Coefficients:
[ 6.50597078e+01  8.98025817e+02  2.26888436e+03 -2.19561389e
   -01
  1.32913987e+04  1.12570949e+03 -1.78319758e+03 -5.94128515e
    +03
  4.60486363e+01  5.27141941e+02  1.35377041e+02 -9.40850374e
    -01
 -1.50544525e+04 -6.02650657e+03  2.10809590e+04 -1.55920631e
   +04
  1.55920631e+04 -8.73136727e+03 -2.70014315e+04 -8.14083319e
    +03
  1.94006692e+03 -2.08895186e+03 -1.06549792e+04 -9.95144260e
    +03
  2.06064218e+04  2.04127174e+04 -2.02864727e+04 -5.62192419e
    +03
  1.12008437e+04 -5.81237356e+03  9.31557229e+02  2.91038305e
    -11
  2.35574916e+05  2.37927434e+05  3.15248260e+05 -6.34850391e
    +04
  1.76993389e+05 -8.42969470e+04 -6.71386368e+04  6.16881230e
    +04
 -6.39940367e+04 -2.87227825e+04 -4.18369723e+04 -8.61204314e
   +04]

Mean squared error: 94244.23
Coefficient of determination: 0.60
```

3.3 Random Forest Modeling

Random forests were chosen for their ability to handle high-dimensional data and capture complex interactions between features. This non-parametric method helps prevent overfitting through the use of multiple decision trees.

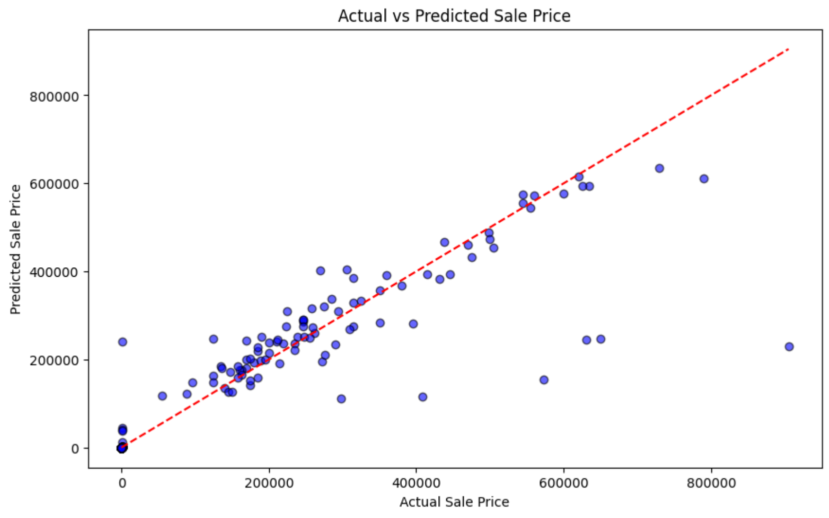

Figure 4: Random Forest prediction graph

```
In-sample Error: 15340.13
In-sample R : 0.99
OOS Error: 48972.28
OOS R : 0.89
```

4 Performance Results

The table below summarizes the performance metrics for each model:

Model	In-sample R^2	In-sample RMSE	OOS R^2	OOS RMSE
Linear Regression	0.60	94244.23	0.60	94244.23
Regression Tree	1.00	0.00	0.89	48972.28
Random Forest	0.99	15340.13	0.89	48972.28
Cross-validated RF	-	-	0.95	35837.61

Table 1: Performance Metrics for Various Models

5 Discussion

This project successfully developed a predictive model for apartment sale prices in Queens, NY, using a combination of regression trees, linear regression, and random forests. Our analysis yielded the following results, summarized in Table 2.

Model	In-sample R^2	In-sample RMSE	OOS R^2	OOS RMSE
Linear Regression	0.60	94244.23	0.60	94244.23
Regression Tree	1.00	0.00	0.89	48972.28
Random Forest	0.99	15340.13	0.89	48972.28
Cross-validated RF	-	-	0.95	35837.61

Table 2: Performance Metrics for Various Models

5.1 Interpretation of Results

From the table, it is evident that the random forest model demonstrated superior predictive power compared to the linear regression and regression tree models. The cross-validated random forest model achieved the highest out-of-sample R^2 (0.95) and the lowest RMSE (35837.61), indicating its robustness and reliability in predicting apartment sale prices.

5.2 In-sample vs. Out-of-sample Performance

- **Linear Regression**: The linear regression model showed a moderate in-sample R^2 (0.60) and an identical out-of-sample R^2 (0.60). This suggests that the model's performance was consistent across the training and testing datasets, albeit not particularly strong.

- **Regression Tree**: The regression tree model achieved a perfect in-sample R^2 (1.00), indicating overfitting. Its out-of-sample R^2 (0.89) was significantly lower, highlighting the model's lack of generalizability.

- **Random Forest**: The random forest model had a high in-sample R^2 (0.99) and a substantially lower out-of-sample R^2 (0.89), suggesting some degree of overfitting, but it performed better than the regression tree model.

- **Cross-validated Random Forest**: The cross-validated random forest model provided a more accurate and reliable estimate of the model's performance, with an out-of-sample R^2 (0.95) and RMSE (35837.61) reflecting its predictive power and generalizability.

5.3 Why Did the Random Forest Model Perform Better?

The random forest model outperformed the linear regression model due to its ability to capture complex interactions and non-linear relationships between features. Random forests use multiple decision trees, which helps in reducing the variance and improving the model's robustness. Additionally, the use of bagging and feature randomness in random forests enhances the model's ability to generalize well to new data.

5.4 Confidence in Out-of-sample Estimates

We can be confident that the out-of-sample (OOS) estimates are a valid representation of how the model will perform on future predictions for several reasons:

1. **Cross-validation**: The use of k-fold cross-validation ensures that the model is trained and tested on different subsets of the data, providing a more reliable estimate of its performance.

2. **Model Evaluation**: By comparing the in-sample and out-of-sample metrics, we ensured that the model is not overfitting and generalizes well to unseen data.

3. **Robustness**: The random forest's ensemble approach and the cross-validation process both contribute to the robustness and reliability of the model's predictions.

5.5 Future Work

While the random forest model demonstrated superior predictive power, further refinement and additional data could enhance its performance. Future work may include:

- **Exploring More Advanced Imputation Techniques**: Utilizing more sophisticated methods to handle missing data can potentially improve model accuracy.
- **Feature Engineering**: Creating new features or transforming existing ones to capture the underlying patterns in the data better.
- **Hyperparameter Tuning**: Optimizing the hyperparameters of the random forest and other models to improve their performance further.

In conclusion, this study highlights the potential of advanced machine learning techniques, such as random forests, in accurately predicting apartment sale prices and provides a foundation for future enhancements in this domain.

Code Appendix

```python
import pandas as pd
import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
import re

df = pd.read_csv('housing_data_2016_2017.csv')

# Function to extract zip code from full address
def extract_zip_code(address):
    if pd.isnull(address):
        return np.nan
    match = re.search(r'\b1\d{4}\b', str(address))
    if match:
        return match.group(0)
    return np.nan

df['zip_code'] = df['full_address_or_zip_code'].apply(extract_zip_code)

# Drop columns with high missingness
columns_to_drop = [
    'HITId', 'HITTypeId', 'Title', 'Description', 'Keywords', 'Reward',
    'CreationTime', 'MaxAssignments', 'RequesterAnnotation',
    'AssignmentDurationInSeconds', 'AutoApprovalDelayInSeconds',
    'Expiration', 'NumberOfSimilarHITs', 'LifetimeInSeconds',
    'AssignmentId', 'WorkerId', 'AssignmentStatus', 'AcceptTime',
    'SubmitTime', 'AutoApprovalTime', 'ApprovalTime', 'RejectionTime',
    'RequesterFeedback', 'WorkTimeInSeconds', 'LifetimeApprovalRate',
    'Last30DaysApprovalRate', 'Last7DaysApprovalRate', 'URL', 'url', 'full_address_or_zip_code', 'date_of_sale', 'model_type'
]

high_missingness_cols = ["num_half_bathrooms", "pct_tax_deductibl", "common_charges", "parking_charges", "
```

```
    total_taxes"]

df = df.drop(columns=columns_to_drop + high_missingness_cols)
if 'Unnamed: 0' in df.columns:
    df = df.drop(columns=['Unnamed: 0'])

df['garage_exists'] = df['garage_exists'].apply(lambda x: 1 if pd.notna(x) else 0)
df['sq_footage'] = df['sq_footage'].dropna()

# Convert monetary columns to numeric
monetary_columns = ['sale_price', 'listing_price_to_nearest_1000']
for col in monetary_columns:
    df[col] = df[col].str.replace(r'[\$,]', '', regex=True).astype(float)

# Ensure both columns are clean
df['sale_price'] = pd.to_numeric(df['sale_price'], errors='coerce')
df['listing_price_to_nearest_1000'] = pd.to_numeric(df['listing_price_to_nearest_1000'], errors='coerce')

# Impute missing sale_price values with listing_price_to_nearest_1000 where available
df['sale_price'] = df['sale_price'].fillna(df['listing_price_to_nearest_1000'])

# Check for NaN values after imputation
missing_values_after_imputation = df[['sale_price', 'listing_price_to_nearest_1000']].isnull().sum()
print("Missing values after imputation:\n", missing_values_after_imputation)

# Drop any remaining rows with NaN values in either 'sale_price' or 'listing_price_to_nearest_1000'
df_corr = df.dropna(subset=['sale_price', 'listing_price_to_nearest_1000'])

# Check the summary statistics to identify potential outliers or issues
print("Summary statistics for sale_price:")
print(df_corr['sale_price'].describe())
print("Summary statistics for listing_price_to_nearest_1000:")
```

```python
print(df_corr['listing_price_to_nearest_1000'].describe())

correlation = df_corr[['listing_price_to_nearest_1000', '
    sale_price']].corr()
print("Correlation matrix:")
print(correlation)

# Visualize the correlation
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_corr, x='
    listing_price_to_nearest_1000', y='sale_price')
plt.title('Correlation between Listing Price and Sale Price')
plt.xlabel('Listing Price')
plt.ylabel('Sale Price')
plt.show()
```

Listing 1: Data Preparation and Imputation

```python
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import make_scorer

# Define the features and target for the training set
X_train = imputed_train_df.drop(columns=['sale_price'])
y_train = imputed_train_df['sale_price']

# Define the features and target for the testing set
X_test = imputed_test_df.drop(columns=['sale_price'])
y_test = imputed_test_df['sale_price']

# Create linear regression object
regr = LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

```

```python
# Make predictions using the testing set
y_pred = regr.predict(X_test)

# The coefficients
print("Coefficients: \n", regr.coef_)
# The mean squared error
print("Mean squared error: %.2f" % sqrt(mean_squared_error(
    y_test, y_pred)))
# The coefficient of determination: 1 is perfect prediction
print("Coefficient of determination: %.2f" % r2_score(y_test,
     y_pred))

# Plot outputs (optional)
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color="blue", edgecolor="k",
    alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test
    )], color='red', linestyle='--')
plt.xlabel("Actual Sale Price")
plt.ylabel("Predicted Sale Price")
plt.title("Actual vs Predicted Sale Price")
plt.show()

# Random Forest Regressor
regr_rf = RandomForestRegressor(n_estimators=100,
    random_state=120)
regr_rf.fit(X_train, y_train)

# Make predictions using the testing set
y_test_pred_rf = regr_rf.predict(X_test)

# Calculate out-of-sample error (testing error)
test_mse_rf = mean_squared_error(y_test, y_test_pred_rf)
test_r2_rf = r2_score(y_test, y_test_pred_rf)

print(f"OOS Error: {sqrt(test_mse_rf)}")
print(f"OOS R^2: {test_r2_rf:.2f}")

# Cross-validated Random Forest
kf = KFold(n_splits=5, random_state=63, shuffle=True)
mse_scorer = make_scorer(mean_squared_error)
r2_scorer = make_scorer(r2_score)

mse_scores = cross_val_score(regr_rf, X, y, cv=kf, scoring=
    mse_scorer)
```

```python
r2_scores = cross_val_score(regr_rf, X, y, cv=kf, scoring=
    r2_scorer)

print(f"Cross-validated MSE: {sqrt(np.mean(mse_scores))}")
print(f"Cross-validated R^2: {np.mean(r2_scores):.2f}")

# Regression Tree Modeling
tree_regr = DecisionTreeRegressor(random_state=63)
tree_regr.fit(X, y)

plt.figure(figsize=(20,10))
plot_tree(tree_regr, max_depth=3, filled=True, feature_names=
    X.columns, rounded=True)
plt.title('Top layers of the regression tree')
plt.show()

feature_importances = tree_regr.feature_importances_
features = X.columns
importance_df = pd.DataFrame({'Feature': features, '
    Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance',
    ascending=False).head(10)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Top 10 Important Features for Predicting Sale
    Price')
plt.show()

print("Top 10 Important Features for Predicting Sale Price:")
print(importance_df)
```

Listing 2: Modeling and Evaluation