Question 1. Data Exploration

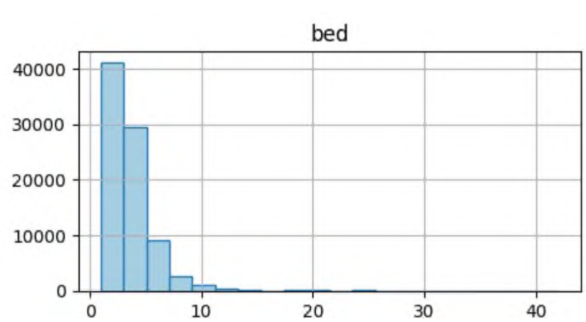
a. Description and Data Cleaning

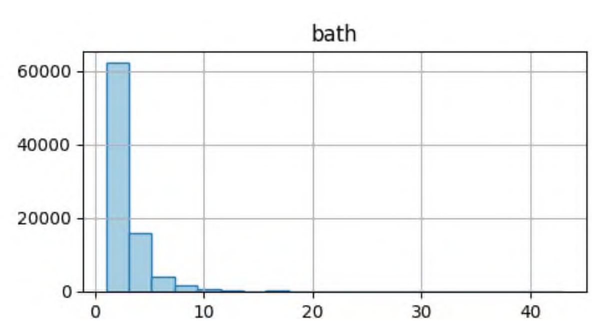
The subject of this report is a list of properties that are currently listed for sale in the state of New York.

	bed	bath	acre_lot	zip_code	house_size	price	is_prev_sold	log_price
count	84040.000000	83946.000000	84040.000000	84036.000000	84040.000000	8.404000e+04	388925.000000	84040.000000
mean	3.930200	2.980690	10.624121	10983.073409	2472.279938	1.274604e+06	0.145542	13.612734
std	2.062923	1.756449	849.058032	688.870753	2326.090138	2.312462e+06	0.352647	0.836738
min	1.000000	1.000000	0.000000	6390.000000	122.000000	2.000000e+04	0.000000	9.903488
25%	3.000000	2.000000	0.060000	10514.000000	1370.000000	5.280000e+05	0.000000	13.176852
50%	4.000000	3.000000	0.140000	10916.000000	2000.000000	7.545000e+05	0.000000	13.533811
75%	5.000000	4.000000	0.570000	11233.000000	2880.000000	1.220000e+06	0.000000	14.014361
max	42.000000	43.000000	100000.000000	14534.000000	112714.000000	1.690000e+08	1.000000	18.945409

Figure 1: Summary Statistics of DataFrame (df)

From the summary statistics in Figure 1, we found that 84,040 entries are present in the dataset. It includes information on the number of bedrooms and bathrooms, lot size, zip code, house size, sale price, whether the property was previously sold, and the natural logarithm of the sale price. Some missing values were observed under numerous categories, such as "bath", the number of bathrooms in a property, and "prev_sold_date", the last recorded date of property sale. The number of bedrooms ranged from 1 to an outlier of 42 with an average of 3-4 bedrooms, and the number of bathrooms ranged from 1 to an outlier of 43 with an average of 2-3 bathrooms in the property. The distribution of the "acre_lot" is highly skewed, with a mean of 10.62 acres, but a median of only 0.14, suggesting there are outliers present in the data. Similarly, there is considerable variation under "house_size", with an average of 2,472 square feet, standard deviation of 2,326, but a maximum of 112,714. The average property price sits over $1.27 million, with its standard deviation at over $2.31 million, ranging from $20,000 to $169 million. Notably, the mean of the added variable, log-transformed price, "log_price" is $13.61, with a much tighter range of $9.90 to $18.95, highlighting how the added variable helped in normalizing the distribution of house prices, making the data more comparable to produce a more meaningful result. Lastly, from the other added variable "is_prev_sold", derived from the "prev_sold_date" variable, we found that 14.55% of properties have a recorded previous sale.





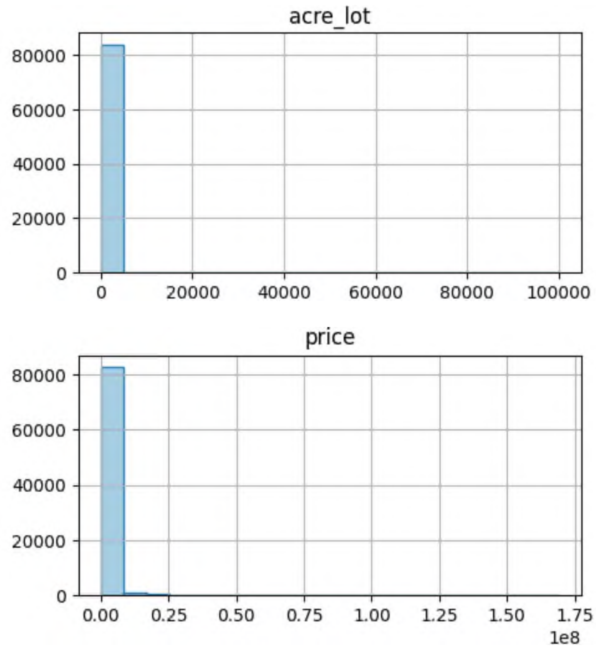
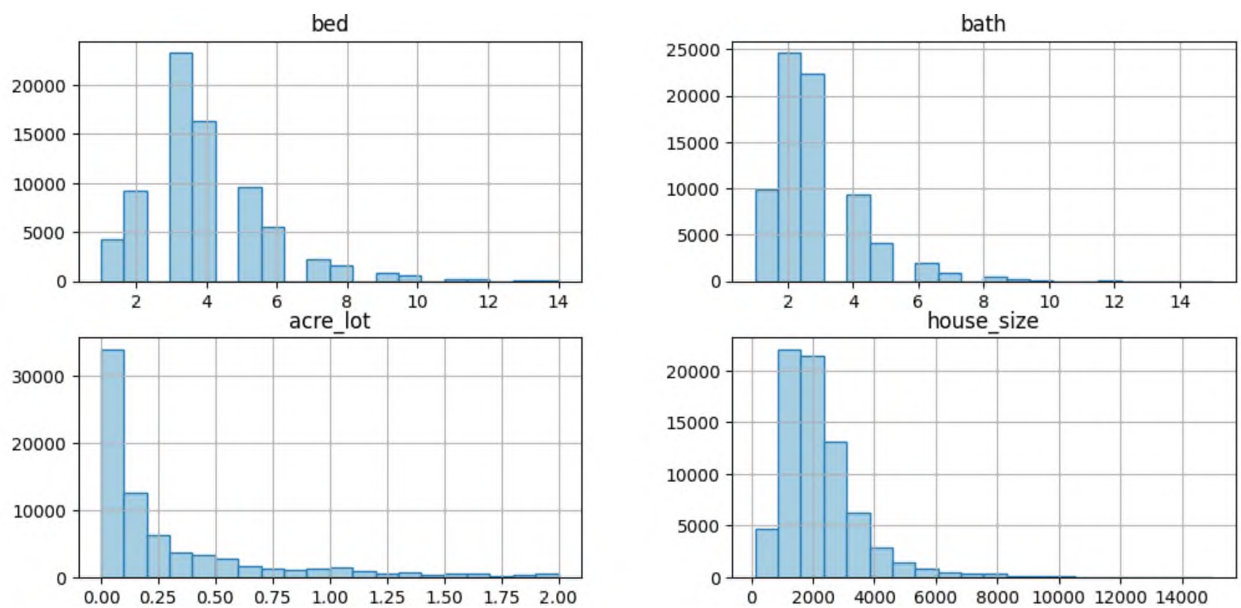


Figure 2.1: Visual representation of the distribution of numerical features

The histograms in Figure 2.1 each display numeric variables "bed", "bath", "acre_lot", "house_size", "price", and "log_price" found in the dataset, showing that most variables are heavily skewed to the right, with a concentration of smaller values. Unsurprisingly, "log_price" appears to be more normally distributed than other variables.



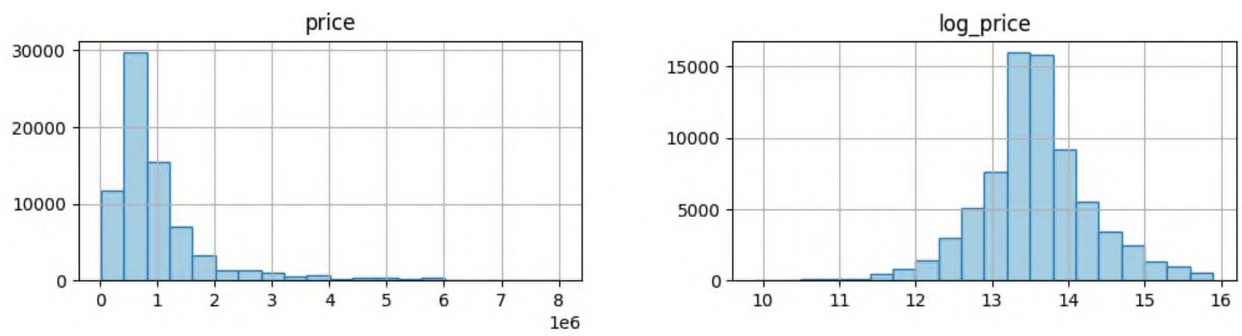


Figure 2.2: Visual representation of the distribution of numerical features (filtered)

After applying filters to remove extreme outliers, the updated histograms in Figure 2.2 for numeric variables "bed", "bath", "acre_lot", "house_size", "price", and "log_price" now all exhibit more focused ranges, with their skewness reduced. "log_price" remains largely unchanged.The filters applied were: removing entries with more than 15 bedrooms; removing entries with more than 15 bathrooms; removing entries with lot size larger than 2; removing entries with house size larger than 20,000 square feet; removing entries with house price higher than $8 million.

b. House Features - Price

Figure 3 is a pairplot with histograms and scatter plots. The diagonals are histograms showing distribution of bed, bath, price and log_price. Histograms for bed, bath, and house_size represent unimodal distribution, with clear peaks. Most properties have 3 to 6 bedrooms, with a mean of 4.5; numbers for bedrooms are rare for lower 3 and higher than 6 bedrooms. Most properties have 2 to 4 bathrooms, decreasing significantly beyond 5. House_size histogram shows most houses are under 5,000 square feet.

Scatterplot in bed and bath reveal a positive correlation, as more bedrooms correspond to more bathrooms. Similarly larger houses will have more bedrooms and bathrooms; the ‘bath and house_size’ figure shows more slope than ‘bed and house_size’, indicating bathrooms influence house_size more.

Using log_price reduces price outliers, creates a near-normal distribution in histograms, and enhances linear correlations with bed, bath, and house_size. As shown in the bottom row, log_price scatter plots reveal: 1. bedrooms and log_price show a slight positive correlation; more bedrooms will have high log_price. However, with 10 bedrooms, log_price varies widely (11-16). Although the bed number is the same, the log_price does not increase within a small bound (between maximum log_price and minimum log_price) for a stable bedroom number, indicating other factors like bathroom number also affect price.

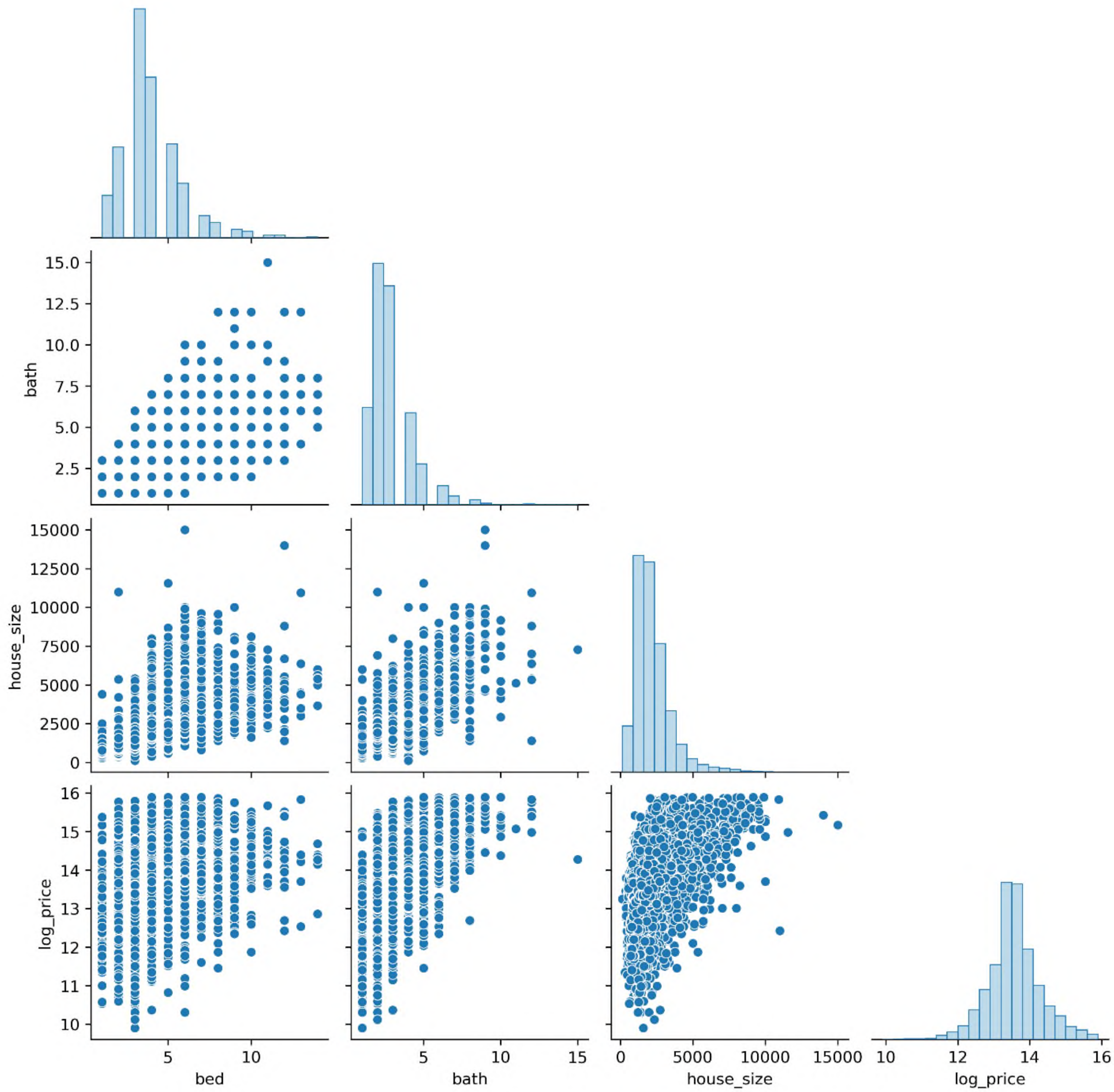


Figure 3: relation between bed number, bath number, and price.

2. The bed and log_price figure shows as the number of bathrooms increases, the log_price of properties increases. This correlation is stronger compared to bedrooms. They form a moderate positive correlation.
3. The scatterplot shows a strong positive correlation between house_size and log_price, with closely clustered points indicating a stable relationship less influenced by other factors. Overall, house_size has the strongest correlation with log_price.

c. Geo Location - Price

c.1. Distribution

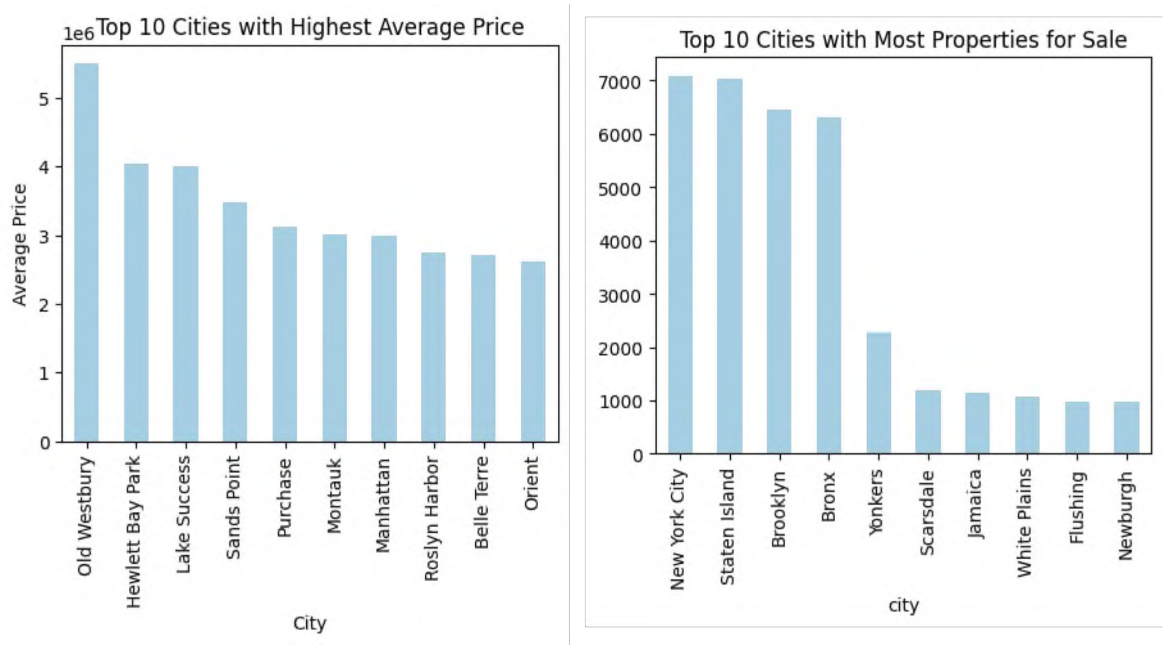


Figure 4: Top 10 cities with highest average price & most properties.

The right side of Figure 4 shows New York City and Staten Island hold the most properties for sale than other cities', with both cities achieving over 7,000. Brooklyn and Bronx are the second tier city group containing the most properties for sale around 6,500 after New York City and Staten Island. After the fifth place, Yonkers, the number of properties listed for sale drastically decreased, in places such as Scarsdale, Jamaica, and White Plains, each holding around 1,000 properties respectively. This suggests that the top 4 cities have the highest liquidity in terms of property trading. The left side of the figure shows Old Westbury has the highest average price of properties for sale at over $5.5, far surpassing all other cities.

c.2. In-NYC vs Non-NYC

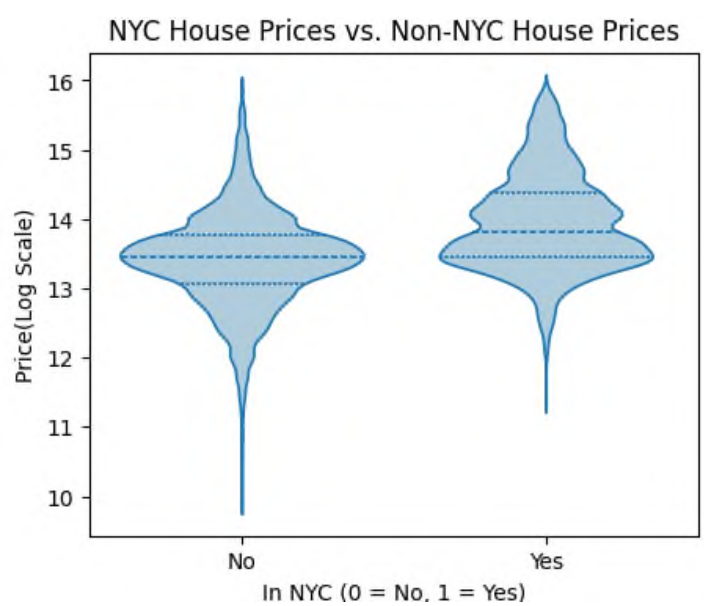


Figure 5: NYC House Prices vs. Non-NYC House Prices

Figure 5 shows a violin plot that compares the log-transformed house prices in New York City (NYC) and those outside of NYC. The X-axis puts the properties that are located in NYC (1 = Yes) against those that are not (0 = No), while the Y-axis displays the log-transformed prices, ranging from 10 to 16. The plot shows that the log-transformed prices of properties in NYC are generally higher than those outside of NYC. The price distribution of non-NYC houses is wider than that of NYC houses, suggesting a bigger difference between the price floor and ceiling. The median price of NYC houses is higher than the median price of non-NYC houses, as shown by the thicker part of the violin plot being in the higher price range of houses in NYC.

d. Previous Sale Status - Price

d.1. Number of Properties Sold Over Time

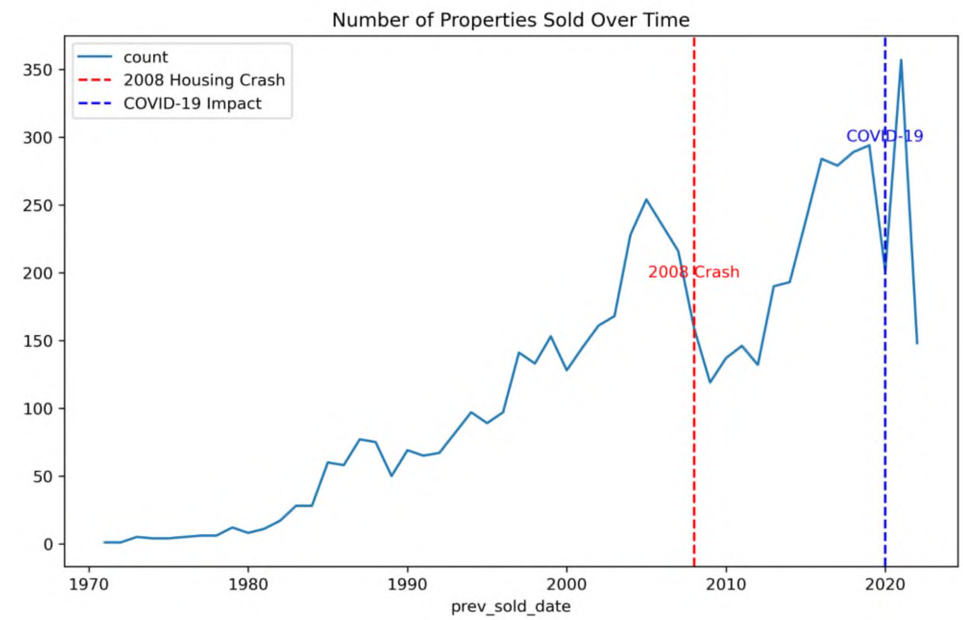


Figure 6: Number of Properties Sold Over Time

Figure 6 shows a line plot displaying the trend in the number of properties sold from the 1970s to the 2020s. Over time, the real estate market grew steadily, with an increasing number of properties being sold. However, two significant declines stand out. The first drop occurred around 2008, which aligns with the global financial crisis, a period that severely disrupted housing markets. The second decline took place around 2020, likely due to the effects of the COVID-19 pandemic. Both events highlight how external shocks can impact property sales.

d.2. Is_Prev_Sold vs Non_Prev_Sold

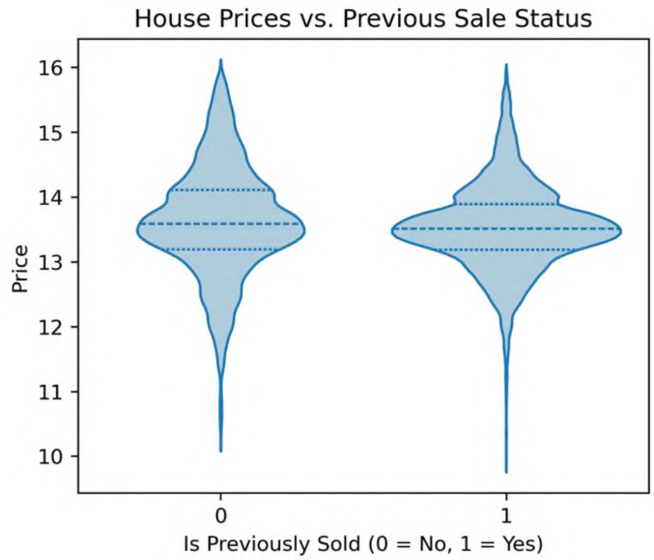


Figure 7: Violin Plot of House Prices by Previous Sales Status

In Figure 7 a violin plot was used to compare house price distributions based on whether the property had been sold before (is_prev_sold). The results indicate that properties with a previous sale history (is_prev_sold = 1) have a higher concentration of data points.

e. Summary Correlation Matrix

Interestingly, the average prices for the two groups appear to be nearly the same. However, the upper quartile (75th percentile) of prices for properties that have not been sold before (is_prev_sold = 0) is slightly higher. This observation might suggest that new or unsold properties tend to achieve higher price ceilings compared to those with a prior sales history.

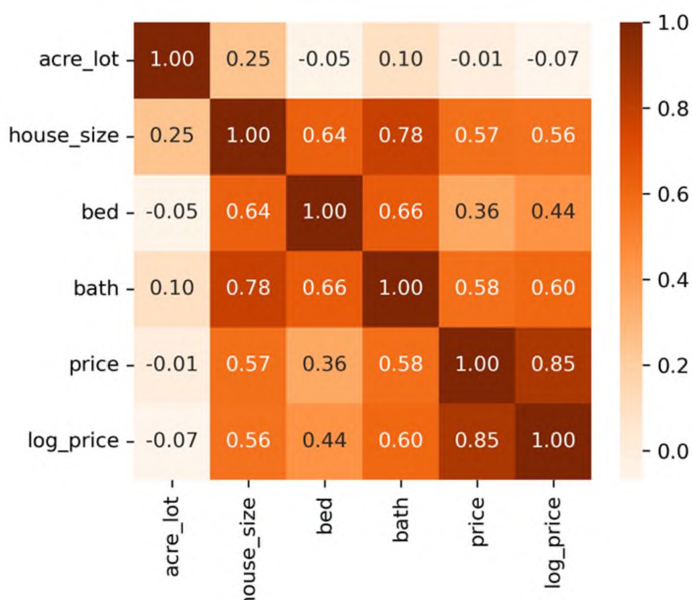


Figure 8: Correlations Between Numeric Variables

For Figure 8 we created a correlation heatmap. It provides insight into how different property features relate to house prices, particularly using log_price for a more normalized representation. The correlation coefficients between log_price and key variables are as follows: 0.44 for bed (number of bedrooms), 0.60 for bath (number of bathrooms), and 0.56 for house_size. These moderate positive correlations suggest that larger properties with more rooms and bathrooms generally command higher prices.

It is also worth noting that the correlation between price and log_price is almost identical. Moving forward, we will primarily focus on log_price as it scales down the variation, and tends to enhance the accuracy of further analysis.

Interestingly, the correlation between lot size (acre_lot) and price is very weak at **-0.07**, indicating that lot size has little impact on house prices in the dataset. This could be due to the influence of other factors, such as location, property features, or demand in the housing market. Additionally, inconsistencies are observed with acre_lot during further analysis, so we decided to prioritize house_size as a more reliable feature for modeling purposes.

Finally, we observe a strong positive correlation between house_size and both bed and bath (0.64 and 0.78, respectively). This aligns with expectations, as larger homes tend to accommodate more bedrooms and bathrooms.

Overall, this analysis highlights key property features that influence prices, providing a foundation for further exploration and modeling.

Question 2. Regression Model

a. Log Transformation

We used two variations of the dataset—one cleaned (duplicates and outliers removed) and one uncleaned (only outliers removed)—to analyze how the data sets impact model performance. This allowed us to compare the effects of both log transformation and cleaning on the accuracy and reliability of the models. We applied a log transformation to scale down the price data, reducing skewness. Alongside this, we retained an untransformed version of the price data to build a separate model, enabling a thorough comparison of the impact of transformations and data cleaning on predictive accuracy.

b. Variable Selection

The selected variables encapsulate key factors influencing property prices. price is the target variable, while house size, bed, and bath represent core features affecting functionality and appeal. acre_lot reflects land size, and nyc accounts for the unique pricing dynamics of the New York City location. This selection balances property attributes and location for a comprehensive analysis.

c. Model Selection

We used the OLS linear regression model because it is a simpler and powerful model for predicting a continuous target variable like price. It assumes a linear relationship between the dependent variable (price) and the independent variables (house_size, bed, bath, acre_lot, and nyc). This allows us to interpret the impact of each feature on property prices, providing clear insights into how changes in features influence the price.

d. Model Performance

The performance of the OLS linear regression model was evaluated using key metrics. The R-squared and adjusted R-squared values of the selected model demonstrated that the model effectively captured a significant portion of the variance in property prices while accounting for the number of predictors. The p-values confirmed which key variables were statistically significant contributors to the model. Additionally, the coefficients provided clear and interpretable insights into the magnitude and direction of each variable's impact on property prices showing us which had strongest relation and which were weakest. The goodness of fit is explained in the findings below. Overall, the model performed well, delivering reliable predictions and valuable insights.

e. Findings

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.487
Model:                            OLS   Adj. R-squared:                  0.487
Method:                 Least Squares   F-statistic:                 1.404e+04
Date:                Sun, 15 Dec 2024   Prob (F-statistic):               0.00
Time:                        17:38:18   Log-Likelihood:            -1.1001e+06
No. Observations:               74005   AIC:                         2.200e+06
Df Residuals:                   73999   BIC:                         2.200e+06
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept  -1.803e+05   6917.582    -26.060      0.000   -1.94e+05   -1.67e+05
nyc[T.Yes]  6.477e+05   5898.716    109.796      0.000    6.36e+05    6.59e+05
house_size   340.5055      3.465     98.263      0.000     333.714     347.297
bed         -7.48e+04   1952.312    -38.311      0.000   -7.86e+04    -7.1e+04
bath        2.201e+05   3055.031     72.052      0.000    2.14e+05    2.26e+05
acre_lot   -1.928e+05   7210.347    -26.741      0.000   -2.07e+05   -1.79e+05
==============================================================================
Omnibus:                    41683.164   Durbin-Watson:                   0.132
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           548442.073
Skew:                           2.455   Prob(JB):                         0.00
Kurtosis:                      15.400   Cond. No.                     8.78e+03
==============================================================================
```

Figure 1 : Original Data (Excluding Outliers)

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.509
Model:                            OLS   Adj. R-squared:                  0.509
Method:                 Least Squares   F-statistic:                 1.535e+04
Date:                Sat, 14 Dec 2024   Prob (F-statistic):               0.00
Time:                        21:44:58   Log-Likelihood:                -55540.
No. Observations:               74005   AIC:                         1.111e+05
Df Residuals:                   73999   BIC:                         1.111e+05
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     12.5448      0.005   2444.450      0.000      12.535      12.555
nyc[T.Yes]     0.5208      0.004    119.008      0.000       0.512       0.529
house_size     0.0002   2.57e-06     77.802      0.000       0.000       0.000
bed            0.0021      0.001      1.430      0.153      -0.001       0.005
bath           0.1699      0.002     74.953      0.000       0.165       0.174
acre_lot      -0.1896      0.005    -35.454      0.000      -0.200      -0.179
==============================================================================
Omnibus:                     5057.576   Durbin-Watson:                   0.103
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            13929.687
Skew:                          -0.380   Prob(JB):                         0.00
Kurtosis:                       4.985   Cond. No.                     8.78e+03
==============================================================================
```

Figure 2 : Original Data with Logged Prices (Excluding Outliers)

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.494
Model:                            OLS   Adj. R-squared:                  0.493
Method:                 Least Squares   F-statistic:                     1545.
Date:                Sat, 14 Dec 2024   Prob (F-statistic):               0.00
Time:                        21:49:43   Log-Likelihood:                -6693.3
No. Observations:                7928   AIC:                         1.340e+04
Df Residuals:                    7922   BIC:                         1.344e+04
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     12.4321      0.018    694.631      0.000      12.397      12.467
nyc[T.Yes]     0.5362      0.014     38.149      0.000       0.509       0.564
house_size     0.0002   8.22e-06     24.041      0.000       0.000       0.000
bed           -0.0109      0.005     -2.210      0.027      -0.021      -0.001
bath           0.2007      0.007     26.769      0.000       0.186       0.215
acre_lot      -0.2094      0.017    -12.109      0.000      -0.243      -0.175
==============================================================================
Omnibus:                      483.617   Durbin-Watson:                   0.799
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1168.148
Skew:                          -0.371   Prob(JB):                    2.19e-254
Kurtosis:                       4.728   Cond. No.                     8.87e+03
==============================================================================
```

Figure 3 : Cleaned Data with Logged Prices

The analysis compares three regression models to identify which one performs best at predicting property prices. Each model approached the dataset differently, with variations in data preprocessing and transformations of the price variable, aiming to address potential issues like skewness, outliers, and duplicate records.

The first model was based on the original dataset with outliers in property prices removed but without any transformations applied to the price variable. This model achieved an R-squared value of 48.7%, meaning that 48.7% of the variation in property prices was explained by the selected predictors (house_size, bed, bath, acre_lot, and whether the property was in NYC). While this is a reasonable level of explanatory power, the lack of transformation may have left issues like skewed prices and heteroscedasticity unaddressed, which can limit the model's effectiveness and stability.

In the second model, the same dataset was used, but the price variable was log-transformed. This transformation helped scale down the values. As expected, the R-squared value increased to 50.9%, and the adjusted R-squared also stood at 50.9%. This adjustment reflects the proportion of variance explained by the model while accounting for the number of predictors, confirming that the improvement is meaningful. By log-transforming the price, the model captured a stronger and more linear relationship between the predictors and the target variable (price), resulting in better predictive accuracy.

In the third model we removed duplicate rows from the dataset in addition to applying the log transformation to prices. While this additional cleaning step improved the integrity of the data, the R-squared value for this model dropped slightly to 49.4%. This suggests that the removal of duplicates reduced some of the variability in the dataset that the model could use, leading to a marginally lower explanatory power. While this model still performed well, it did not surpass the second model in terms of fit or clarity.

Based on this comparison, the second model proves as the best-performing model, with an R-squared of 50.9%. This implies that the log transformation effectively addressed potential issues with the price variable, allowing for a better fit. The adjusted R-squared being identical to the R-squared confirms that the predictors used in the model were highly relevant and contribute meaningfully to explaining the variance in property prices.

As we have chosen the second model for further analysis, a deeper observation reveals that most predictors - house_size, bath, acre_lot, and the NYC indicator - demonstrate strong statistical significance, with p-values consistently reported as 0.00. This means these variables have a significant impact on property prices, allowing us to confidently reject the null hypothesis that they have no effect. However, one important detail stands out: the predictor bed does not meet the same level of significance. Its p-value is 0.13, which is above the conventional threshold of 0.05, indicating that the number of bedrooms does not independently contribute to explaining property prices in this model.

In examining the coefficients from the chosen model (second model), the strongest predictor of property prices is the NYC indicator, with a coefficient of approximately 0.52 in the log-transformed model. This suggests that properties in the NYC area are associated with a price increase of about 52% compared to non-NYC properties, highlighting the premium of urban locations. Bathrooms also play a significant role, with a coefficient of 0.17, indicating a 17% increase in price for each additional bathroom. Meanwhile, house size contributes positively to price, though its influence is comparatively weaker, with a coefficient of 0.0002. Bedrooms and lot size have even smaller impacts, with coefficients of 0.0021 and -0.19, respectively. Interestingly, the negative coefficient for lot size suggests that larger lots might slightly reduce property prices after controlling for other factors, which could reflect market preferences for smaller, more centrally located lots in certain areas.

In conclusion, the second model, which applied a log transformation to the price variable, was selected as the most effective and reliable model for predicting property prices. It demonstrated the highest explanatory power, with an R-squared of 50.9%, and showcased a strong relationship between predictors and property prices. The findings emphasize the importance of location, amenities like bathrooms, and house size as key drivers of property value.

In conclusion, we developed and evaluated three regression models to predict property prices, each using different approaches to data preparation and transformation. After analyzing the goodness of fit, model performance, and the significance of predictors, we found that the log-transformed model using the uncleaned dataset provided the best results. This model effectively captured the relationships between property prices and key factors like location, house size, and amenities. By integrating data preprocessing with statistical analysis, we were able to build a model that offers clear insights into the drivers of property value and reliable predictions for real estate pricing.

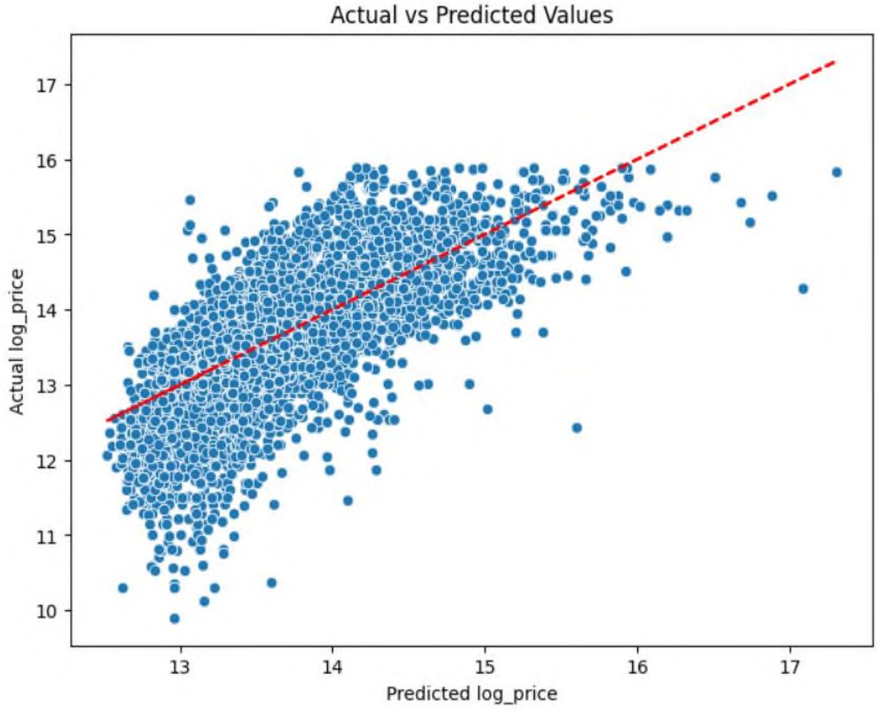


Figure 4: Comparison of Actual and Predicted Log-Prices

The scatterplot compares the actual log-transformed property prices (y-axis) with the predicted values (x-axis) from our regression model, with the red dashed line representing perfect predictions (y = x). The plot shows a clear upward trend, indicating that the model captures the overall relationship between the predictors and log prices. However, there are notable deviations from this ideal line, particularly for higher predicted values.

The scatter widens as the predicted log prices increase, suggesting the presence of heteroscedasticity. This means that the model's accuracy decreases for higher-priced properties. Points below the red line reflect underpredictions, where the actual values are higher than predicted, while points above indicate overpredictions.

While the model provides a good basis and successfully captures the general trend, the heteroscedasticity observed in the residuals suggests areas for improvement. Addressing this issue could involve refining the feature set, or exploring more advanced regression techniques to stabilize the variance and improve overall prediction accuracy,

Question 3. Classification Model

a.Objective:

The goal of the project was to develop and assess different classification models like Logistic Regression, Decision Tree and Random Forest model that can predict whether a property is located in NYC or not, given that its characteristics are provided in the framework of the assignment.
As per the requirement, all geographical predictors such as "city" and "zip code" were excluded from the

analysis. The approach has been focused on structured feature selection, experimentation with different models, and thorough performance evaluation of the models. A systematic approach focusing on feature selection, model experimentation, and performance evaluation was adopted.

b.Methodology:

The data cleaning process ensured that the information was relevant to the project and cleaned which was required for the comprehensive data reprocessing.
Geographical dictators like city, zip_code, state, status, and prev_sold_date were removed to ensure a reliable dataset was produced for the analysis through different predicting models.
Rows with missing values in the target variable nyc were dropped and the missing values in numeric predictors - bed, bath, acre_lot, house_size, price were filled using their respective medians.

To determine the most suitable model for the task, three machine learning algorithms were tested:
1. Logistic Regression; 2. Decision tree; 3. Random forest

c. Data Preprocessing:

Missing Values Handling:
Missing value handling was one of the crucial steps involved in building the models.
In this analysis, the following steps were executed:
Missing values in nyc variables were removed for data integrity and to avoid any potential biases.
The median values for each column were used for imputing the missing values in bed, bath, acre_lot, house_size, and price numerical columns. Due to this approach, the central tendency is maintained while limiting the distortion from outliers.

Encoding Target Variable:
Firstly the variable targets 'nyc' were converted to binary numeric for better modelling.
Feature Scaling was used to make variables more consistent and to allow the best performance of gradient-based models:Numerical predictors (bed, bath, acre_lot, house_size, and price) were standardized using StandardScaler.

d. Model Development:

Below are the three machine learning models that were used in the analysis to predict whether a property is in NYC or not:
1. Logistic Regression; 2. Decision tree; 3. Random forest

Cross-Validation:
Stratified K-Fold Cross-Validation with 5 folds was used to make a fair and balanced splits between the NYC and non-NYC properties. By using this method, the same proportion of NYC and non-NYC properties was produced in each fold, hence making more accurate results when testing these models.

Model Performance Evaluation Metrics:

To assess the performance of the classification models, the following evaluation methods were used:

1. Confusion Matrices:
Confusion matrices gave a detailed breakdown of the performance of the different models by showing the counts of true positives, true negatives, false positives, and false negatives. This analysis helped evaluate how well the models performed across both NYC and non-NYC property classifications.

2. ROC Curves:
ROC curves were plotted to visually show the trade-off between the true positive rate (TPR) and the false positive rate (FPR). These plots showed a clear view of how well the models were classified by the two classes at different decision thresholds, thus making the task of comparing their effectiveness easier.
These metrics allowed gaining valuable insight into the strengths and weaknesses of each model regarding its classification capabilities.

3. ROC-AUC:
It is defined as the ability of the model to distinguish between NYC and non-NYC properties across various thresholds.

4. Accuracy: The overall number of correct predictions made by the model.

e. Results:

The different classification models developed for predicting whether a property is in NYC showed varied performances.
The strengths and limitations of each of the models are discussed below.

1. Logistic Regression:
Overall the Logistic Regression performed well, establishing itself as a good baseline model. However, some limitations in the robustness were seen. As it was sensitive to class imbalances, which affected its generalization ability.

2. Decision Tree:
The Decision Tree model was highly comprehensible with its straightforward and transparent structure.
It helped in shedding light on how each feature contributed to the prediction in an understandable manner, as the relationship between different variables was recognisable.
However, it was very prone to overfitting because its high variance resulted in poor generalization between different data splits.

3. Random Forest:
Random Forest proved to be the best performance model during this analysis. Being an ensemble method, it was also resistant to overfitting, with the highest accuracy and ROC-AUC score when compared to other models.
The Random Forest was the best predicting model at classifying NYC versus non-NYC properties. This fact is further supported by its confusion matrix, which has high classification

accuracy for both true positives and true negatives. The Random forest model performed the best as it handled the scaled and the balanced data well.

Key Insights:

The results show that without the direct location data, the variable set of property size, price, and room count was sufficient to effectively classify properties in NYC.
Of all the models tested, the Random Forest was the most accurate and reliable choice for this task.
Its robustness and ability to leverage key features like house size and price make it the most suitable model for predicting the location of properties of NYC.

Visualizations:

ROC Curves:

The ROC curves below illustrate the trade-off between sensitivity and specificity for each model:

a. Logistic Regression

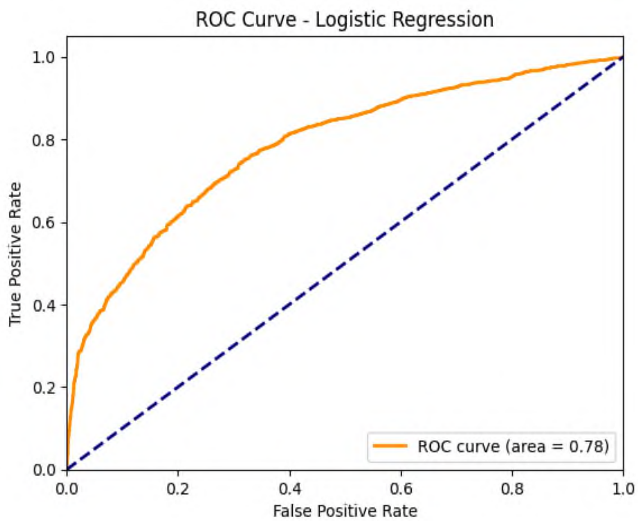


Figure 1: Logistic Regression ROC Curve

The ROC curve for the Logistic Regression model represents a trade-off between the sensitivity (True Positive Rate) and specificity (False Positive Rate). The ROC curve is above the diagonal baseline, meaning that the model does perform better than random guessing. Area under the ROC curve is 0.78, reflecting moderate predictive performance.

The Logistic Regression model did a decent job distinguishing between NYC and non-NYC properties and an AUC of 0.78 suggests there is further work to be done to achieve even higher classification performance.
This indicates the model is reliable; however it may not be as robust as the other models.

b. Decision Tree

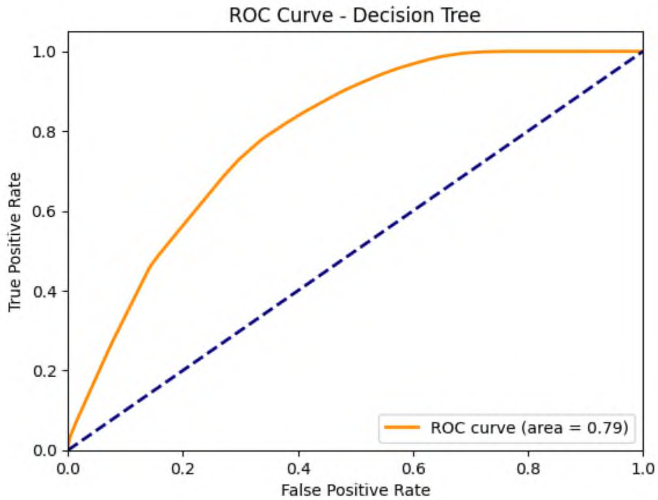


Figure 2: Decision Tree ROC Curve

The ROC curve for the Decision Tree model illustrates the balance between True Positive Rate(sensitivity) and False Positive Rate(specificity). The ROC curve is well above the baseline, which confirms that the model is doing better than random guessing. The AUC is 0.79, which is better compared to Logistic Regression and moderate effectiveness in distinguishing between NYC and non-NYC properties.
The Decision Tree model showed better predictive performance with an AUC of 0.79. That means this model was pretty effective at classifying properties, but it is also understandable.
While the curve is indicative of great performance, the model is probably prone to overfitting; this decreases its generalization capability when compared to a robust ensemble method.

c. Random Forest

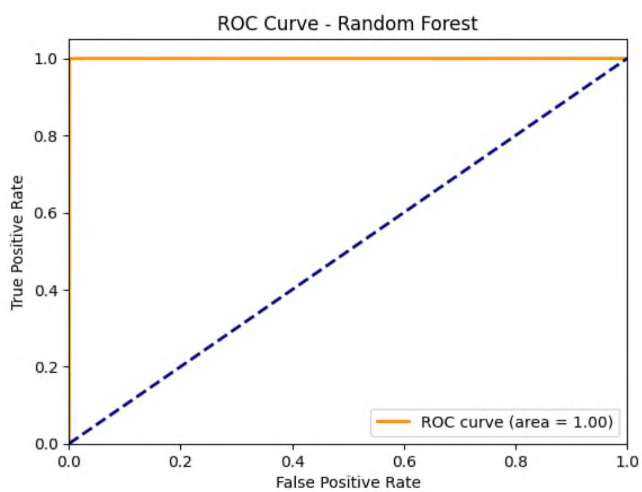


Figure 3: Random Forest ROC Curve

The ROC curve for the Random Forest model speaks very highly for its excellent prediction capability. The ROC curve touches the upper boundary, which indicates perfect separation by the model between the two classes (NYC versus non-NYC properties).
Area under the curve equals 1.00, meaning perfect classification and no compromise between True Positive Rate and False Positive Rate.An AUC score of 1.00 emphasizes the high performance of the

Random Forest model, with no single error in the separation of NYC properties from the rest. This result underscores the reliability, strength, and suitability of the model for this kind of classification, thus making it optimal for the problem at hand.

d. Confusion Matrix (Random Forest):

Below is the confusion matrix of the Random Forest model:

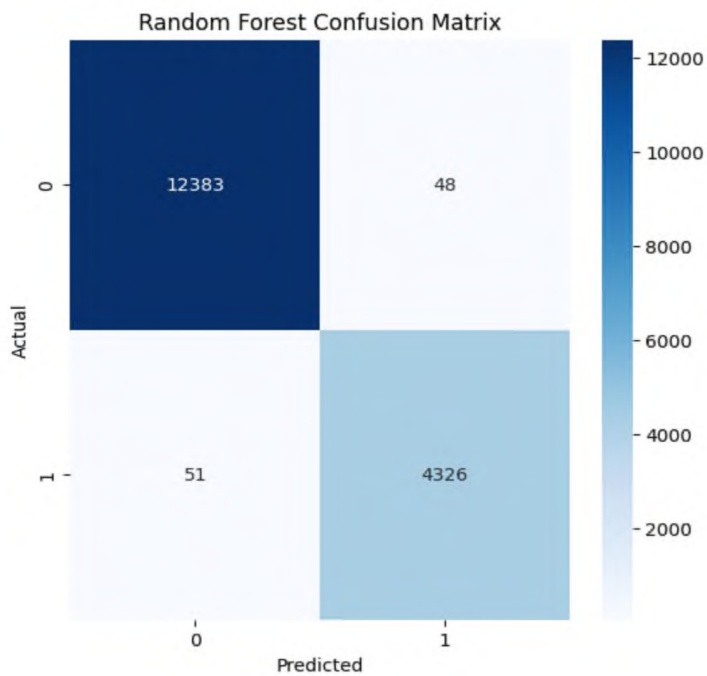


Figure 4: Confusion Matrix Random Forest Model

The confusion matrix for the Random Forest model provides valuable insights into its predictive performance of the model.

Key results are as follows:

1. True Negatives: 12,383 properties correctly classified as not being in NYC.
2. False Positives: 48 properties incorrectly classified as not being in NYC.
3. False Negatives: 51 properties incorrectly classified as being in NYC.
4. True Positives: 4,326 properties accurately classified as being in NYC.

The Random Forest model is exhibiting very good performance, meaning that the model is correctly classifying NYC and non-NYC properties, which showcases its efficiency in handling the classification task.

The low number of misclassified properties is indicative of the robustness and reliability of the model. In general, these results confirm that the Random Forest model is the best choice for this classification task, since it presents high predictive accuracy with reliable performance.

e. Conclusions:

The analysis of different classification models shows that the Random Forest is the best model for classifying NYC properties. It attained the highest accuracy and ROC-AUC score, which indicates a better capability in distinguishing between NYC and non-NYC properties. The ensemble-based approach followed by Random Forest made it resistant to overfitting, hence helping it to extract meaningful patterns from the dataset. Its consistent performance across all evaluation metrics makes it the most reliable choice for this classification task.

1. Key Features:
The features house_size and price showed up as significant predictors throughout all the models as per the assignments framework. These variables were very vital in differentiating NYC properties because they were strongly correlated with the value of the property.

2. Final Takeaway:
Among all the predicting models, the Random Forest model was best suited for considering the problems of performance and handling diverse relationships within the dataset.
The model gives high importance to features like house_size and price to capture unique aspects of NYC properties, producing results with better accuracy in each subsequent trial.
The model can have enhanced precision and robustness for wider applications by tuning parameter adjustments and addressing class imbalances.
The results show that even in the absence of direct geographical information, property attributes carry sufficient information to classify NYC properties effectively.

f. Recommendations:

To improve the models of the Random Forest, Logistic Regression and the Decision Tree the class imbalance can be dealt with by using different techniques as follows:

1. Hyperparameter Tuning:
Further optimization of the Random Forest model, by tuning parameters such as the number of trees or maximum depth, may boost its performance.

2. Handling Imbalanced Classes:
Techniques such as oversampling the minority class or applying weighted loss functions enhance the performance of Logistic Regression and Decision Tree models, respectively, by handling class imbalance efficiently.

3. Refining the model:
The mentioned steps will help in improving the models to make those models more accurate and robust in their predictions.

g. Business Implications:

The developed model provides a strong foundation for predicting the location of NYC properties with key features such as house size and price. The insights derived will help to identify the most influential factors that determine whether a property is in NYC.

These key factors will enable businesses and investors to focus their investments in properties that house these particular characteristics.
This helps businesses in making more data-driven decisions for targeting properties in NYC while optimizing their strategies.
The fact that the model can differentiate between NYC properties without the explicit use of location data demonstrates its strength and practical application in real estate decision-making.
In general, these can lead to wiser investments, better resource allocation, and increased profitability in the competitive NYC property market.

AI Declaration: AI tools were used to refine language for the report.

Import libraries

```
######## To read and manipulate data
import pandas as pd
import numpy as np

######## To run regressions
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.graphics.api as smg

####### Various sci-kit learn functions
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score,auc
from itertools import combinations
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, RocCurveDisplay

# Import accuracy_score
from sklearn.metrics import accuracy_score

######## For plotting
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import scipy.stats as stats

%pylab inline
```

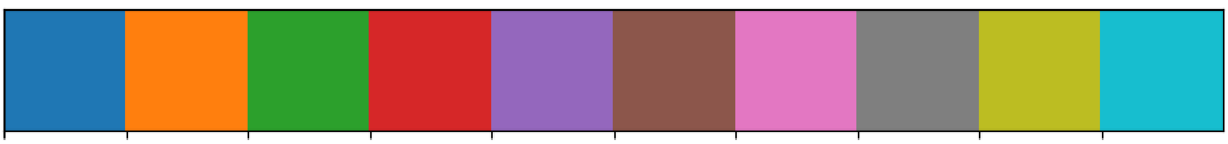
```
%pylab is deprecated, use %matplotlib inline and import the required
libraries.
Populating the interactive namespace from numpy and matplotlib
```

Color Settings

```
palette = sns.color_palette() # Default color palette
print(palette)  # Prints the RGB tuples that make up this color
palette
sns.palplot(palette) # Plotting your palette!
pairpalette = sns.color_palette('Paired')
```

```
sns.palplot(pairpalette) # Seaborn color palette, with 10 colors
sns.color_palette("Oranges", as_cmap=True) # Get a CMap
```

```
[(0.12156862745098039, 0.4666666666666667, 0.7058823529411765), (1.0,
0.4980392156862745, 0.054901960784313725), (0.17254901960784313,
0.6274509803921569, 0.17254901960784313), (0.8392156862745098,
0.15294117647058825, 0.1568627450980392), (0.5803921568627451,
0.403921568627451, 0.7411764705882353), (0.5490196078431373,
0.33725490196078434, 0.29411764705882354), (0.8901960784313725,
0.4666666666666667, 0.7607843137254902), (0.4980392156862745,
0.4980392156862745, 0.4980392156862745), (0.7372549019607844,
0.7411764705882353, 0.13333333333333333), (0.09019607843137255,
0.7450980392156863, 0.8117647058823529)]
```



```
# Convert RGB tuples to hex
hex_colors = [mcolors.to_hex(color) for color in
sns.color_palette('Paired')]
print(hex_colors)
```

```
['#a6cee3', '#1f78b4', '#b2df8a', '#33a02c', '#fb9a99', '#e31a1c',
'#fdbf6f', '#ff7f00', '#cab2d6', '#6a3d9a', '#ffff99', '#b15928']
```

Question 1: Summary Statistics

Read Files

```
# The file path starts from the same location of this notebook
file_path = 'realtor-data-ny.csv'
df = pd.read_csv(file_path)
```

1. Data Cleaning

First, look at the data

```python
# Create the 'prev_sold' variable as a explanation for entries with
their 'prev_sold_date' variable missing. This shows whether a house
had been previously sold.
df['is_prev_sold'] = df['prev_sold_date'].notnull().astype(int)

# Create the log transformed 'price' variable, 'log_price' for better
relative comparison, since entires are spread across different
geographic locations, with exceptional anomalies displayed in New York
City.
df['log_price'] = np.log(df['price'])

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84040 entries, 0 to 84039
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   status          84040 non-null  object
 1   bed             84040 non-null  int64
 2   bath            83946 non-null  float64
 3   acre_lot        84040 non-null  float64
 4   city            84038 non-null  object
 5   state           84040 non-null  object
 6   zip_code        84036 non-null  float64
 7   house_size      84040 non-null  int64
 8   prev_sold_date  56605 non-null  object
 9   price           84040 non-null  int64
 10  nyc             84040 non-null  object
 11  is_prev_sold    84040 non-null  int32
 12  log_price       84040 non-null  float64
dtypes: float64(4), int32(1), int64(3), object(5)
memory usage: 8.0+ MB
```

```python
df.head()
```

```
     status  bed  bath  acre_lot    city     state  zip_code
house_size  \
0  for_sale    3   1.0      0.37  Accord  New York   12404.0
960
1  for_sale    3   2.0      0.38  Accord  New York   12404.0
1936
2  for_sale    2   1.0      0.41  Accord  New York   12404.0
832
3  for_sale    3   1.0      5.50  Accord  New York   12404.0
1900
4  for_sale    3   3.0      6.50  Accord  New York   12404.0
4000

  prev_sold_date   price nyc  is_prev_sold  log_price
```

```
0     21/03/2022  249900  No             1  12.428816
1     06/01/1989  319000  No             1  12.672946
2     10/09/2015  169500  No             1  12.040608
3            NaN  695000  No             0  13.451667
4     30/07/2021  250000  No             1  12.429216
```

```
df.tail()
```

```
         status  bed  bath  acre_lot   city     state  zip_code
house_size  \
84035  for_sale    3   3.0      0.65  Yulan  New York   12792.0
1480
84036  for_sale    4   1.0      1.64  Yulan  New York   12792.0
1692
84037  for_sale    4   1.0      1.64  Yulan  New York   12792.0
1692
84038  for_sale    5   2.0      0.13    NaN  New York       NaN
1925
84039  for_sale    2   1.0    110.00    NaN  New York   12523.0
1177

      prev_sold_date   price nyc  is_prev_sold  log_price
84035     23/05/2006  425000  No             1  12.959844
84036     20/01/2012  188500  No             1  12.146853
84037     20/01/2012  188500  No             1  12.146853
84038            NaN  710000  No             0  13.473020
84039            NaN  495000  No             0  13.112313
```

```
df.describe()
```

```
                bed          bath       acre_lot      zip_code
house_size  \
count  84040.000000  83946.000000   84040.000000  84036.000000
84040.000000
mean       3.930200      2.980690      10.624121  10983.073409
2472.279938
std        2.062923      1.756449     849.058032    688.870753
2326.090138
min        1.000000      1.000000       0.000000   6390.000000
122.000000
25%        3.000000      2.000000       0.060000  10514.000000
1370.000000
50%        4.000000      3.000000       0.140000  10916.000000
2000.000000
75%        5.000000      4.000000       0.570000  11233.000000
2880.000000
max       42.000000     43.000000  100000.000000  14534.000000
112714.000000

              price  is_prev_sold     log_price
```

```
count   8.404000e+04  84040.000000  84040.000000
mean    1.274604e+06      0.673548     13.612734
std     2.312462e+06      0.468917      0.836738
min     2.000000e+04      0.000000      9.903488
25%     5.280000e+05      0.000000     13.176852
50%     7.545000e+05      1.000000     13.533811
75%     1.220000e+06      1.000000     14.014361
max     1.690000e+08      1.000000     18.945409
```

```
df.isnull().sum()
```

```
status                0
bed                   0
bath                 94
acre_lot              0
city                  2
state                 0
zip_code              4
house_size            0
prev_sold_date    27435
price                 0
nyc                   0
is_prev_sold          0
log_price             0
dtype: int64
```

For the records that are missing 'Prev_sold_data', we decided that it just means there's no record that the property's been sold. It's still valid data, so we kept them.

1.1 distribution of categorical variables

```
# List of variables to analyze
variables = ['status', 'city', 'state', 'prev_sold_date',
'is_prev_sold', 'nyc']
# Loop through variables and display value counts
for col in variables:
    print(f"Distribution of values for {col}:")
    print(df[col].value_counts())
    print("\n")
```

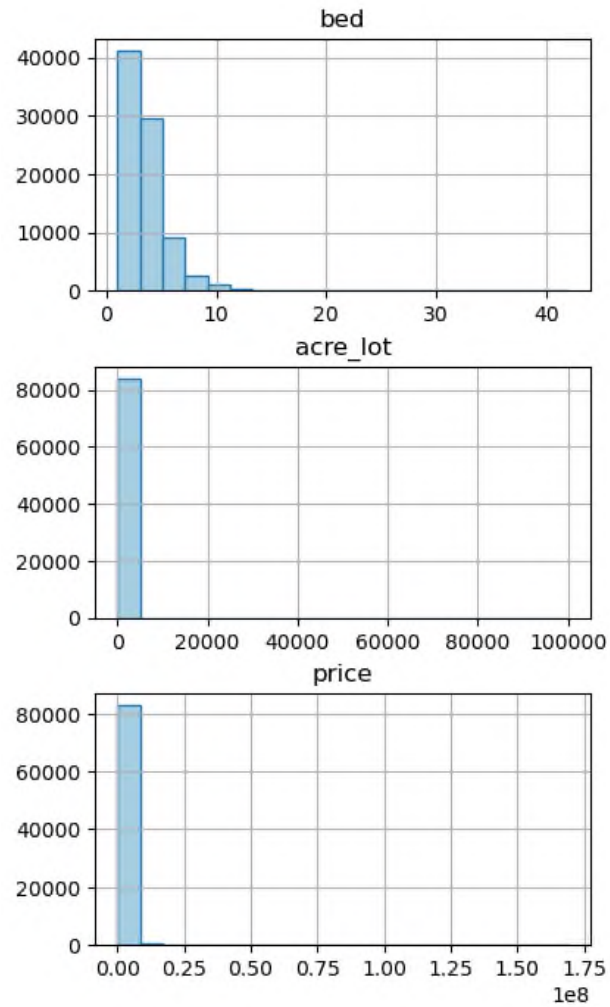
```
Distribution of values for status:
status
for_sale    84040
Name: count, dtype: int64


Distribution of values for city:
city
New York City     8338
Staten Island     7073
```

```
Brooklyn          6535
Bronx             6424
Yonkers           2321
                  ...
Godeffroy            1
Grahamsville         1
Northport            1
Hewlett Harbor       1
Staatsburg           1
Name: count, Length: 457, dtype: int64


Distribution of values for state:
state
New York    84040
Name: count, dtype: int64


Distribution of values for prev_sold_date:
prev_sold_date
04/11/2003    134
05/11/2021    104
21/06/2017     81
03/10/2013     77
06/09/2005     73
             ...
20/03/2008      1
10/07/1998      1
12/07/2017      1
11/07/2006      1
21/03/2022      1
Name: count, Length: 4267, dtype: int64


Distribution of values for is_prev_sold:
is_prev_sold
1    56605
0    27435
Name: count, dtype: int64


Distribution of values for nyc:
nyc
No     61936
Yes    22104
Name: count, dtype: int64
```

1.2 Distribution of continuous variables

```
# List of columns for which to plot histograms
columns_to_plot = ['bed', 'bath', 'acre_lot', 'house_size', 'price',
'log_price']
print(df[columns_to_plot].describe())
# Plot histograms for the specified columns
df[columns_to_plot].hist(figsize=(10,8), bins=20,
color=pairpalette[0],edgecolor=pairpalette[1])
```

```
                bed          bath       acre_lot     house_size
price  \
count  84040.000000  83946.000000   84040.000000   84040.000000
8.404000e+04
mean       3.930200      2.980690      10.624121    2472.279938
1.274604e+06
std        2.062923      1.756449     849.058032    2326.090138
2.312462e+06
min        1.000000      1.000000       0.000000     122.000000
2.000000e+04
25%        3.000000      2.000000       0.060000    1370.000000
5.280000e+05
50%        4.000000      3.000000       0.140000    2000.000000
7.545000e+05
75%        5.000000      4.000000       0.570000    2880.000000
1.220000e+06
max       42.000000     43.000000  100000.000000  112714.000000
1.690000e+08

          log_price
count  84040.000000
mean      13.612734
std        0.836738
min        9.903488
25%       13.176852
50%       13.533811
75%       14.014361
max       18.945409

array([[<Axes: title={'center': 'bed'}>,
        <Axes: title={'center': 'bath'}>],
       [<Axes: title={'center': 'acre_lot'}>,
        <Axes: title={'center': 'house_size'}>],
       [<Axes: title={'center': 'price'}>,
        <Axes: title={'center': 'log_price'}>]], dtype=object)
```



Remove Outliers

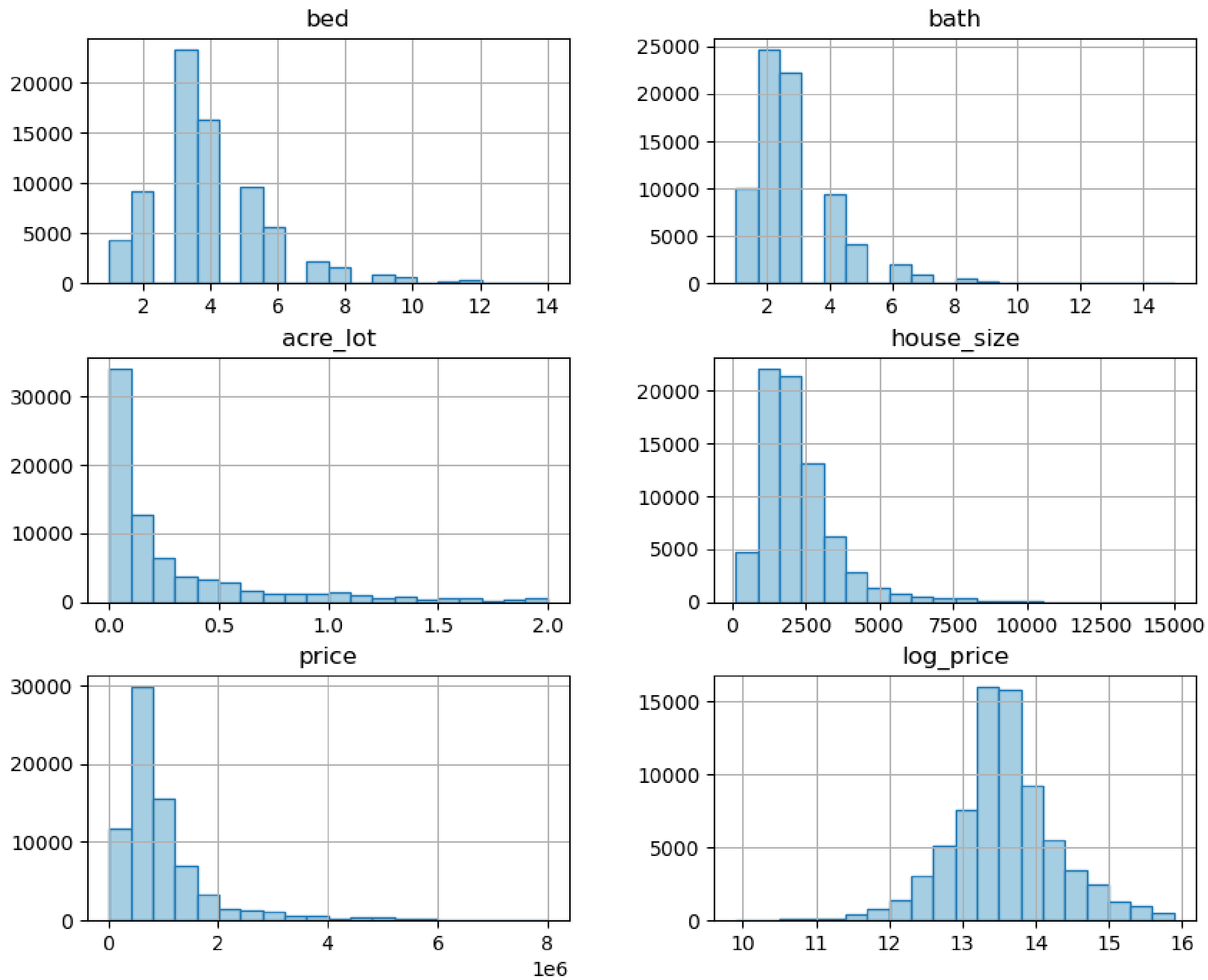
```
# Apply all filtering conditions at once
cdf = df[
    (df['bed'] <= 15) &
    (df['bath'] <= 15) &
    (df['acre_lot'] <= 2) &
    (df['house_size'] <= 20000) &
    (df['price'] <= 8000000)
]

# Check the resulting data
print(cdf.describe())
```

```
                bed          bath      acre_lot      zip_code
house_size  \
count  74005.000000  74005.000000  74005.000000  74001.000000
74005.000000
mean       3.882724      2.829836      0.294322  10968.352806
```

```
2221.820107
std          1.849067      1.418526      0.392463    654.365575
1283.951529
min          1.000000      1.000000      0.000000   6390.000000
122.000000
25%          3.000000      2.000000      0.060000  10512.000000
1360.000000
50%          4.000000      3.000000      0.110000  10923.000000
1950.000000
75%          5.000000      3.000000      0.360000  11233.000000
2713.000000
max         14.000000     15.000000      2.000000  14534.000000
15000.000000

              price  is_prev_sold     log_price
count  7.400500e+04  74005.000000  74005.000000
mean   1.033275e+06      0.682805     13.567891
std    9.643704e+05      0.465387      0.731436
min    2.000000e+04      0.000000      9.903488
25%    5.390000e+05      0.000000     13.197471
50%    7.500000e+05      1.000000     13.527828
75%    1.178000e+06      1.000000     13.979329
max    7.999000e+06      1.000000     15.894827
```

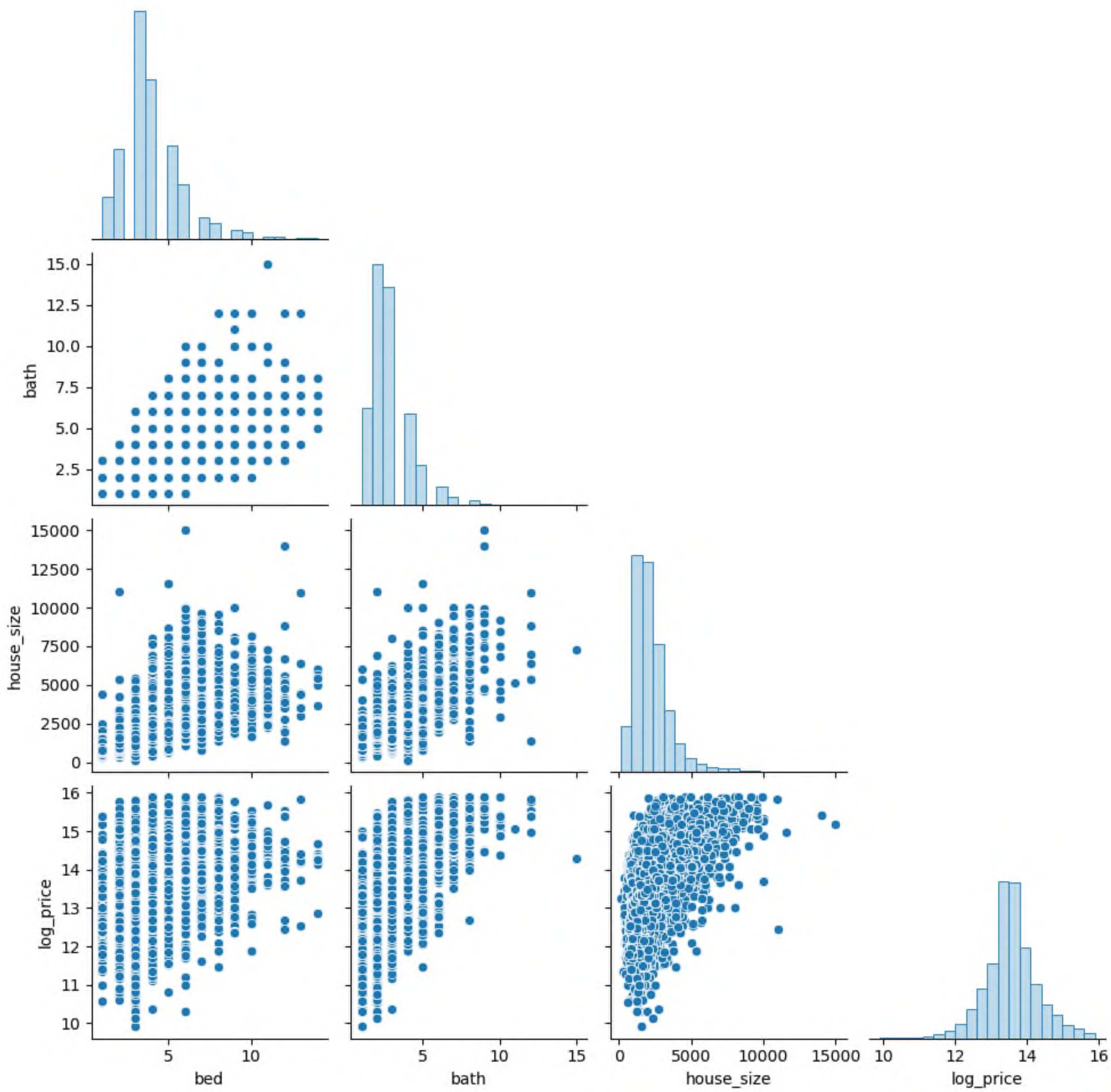
```python
# Plot cleaned data
cdf[columns_to_plot].hist(figsize=(10,8), color=pairpalette[0],
bins=20, edgecolor=pairpalette[1])
```

```
array([[<Axes: title={'center': 'bed'}>,
        <Axes: title={'center': 'bath'}>],
       [<Axes: title={'center': 'acre_lot'}>,
        <Axes: title={'center': 'house_size'}>],
       [<Axes: title={'center': 'price'}>,
        <Axes: title={'center': 'log_price'}>]], dtype=object)
```



2. House Features - Price

```
sns.pairplot(cdf[['bed', 'bath','house_size', 'log_price']],
corner=True, diag_kind='hist',
             diag_kws={'bins': 20, 'edgecolor': pairpalette[1],
'color': pairpalette[0]})
plt.show()
```



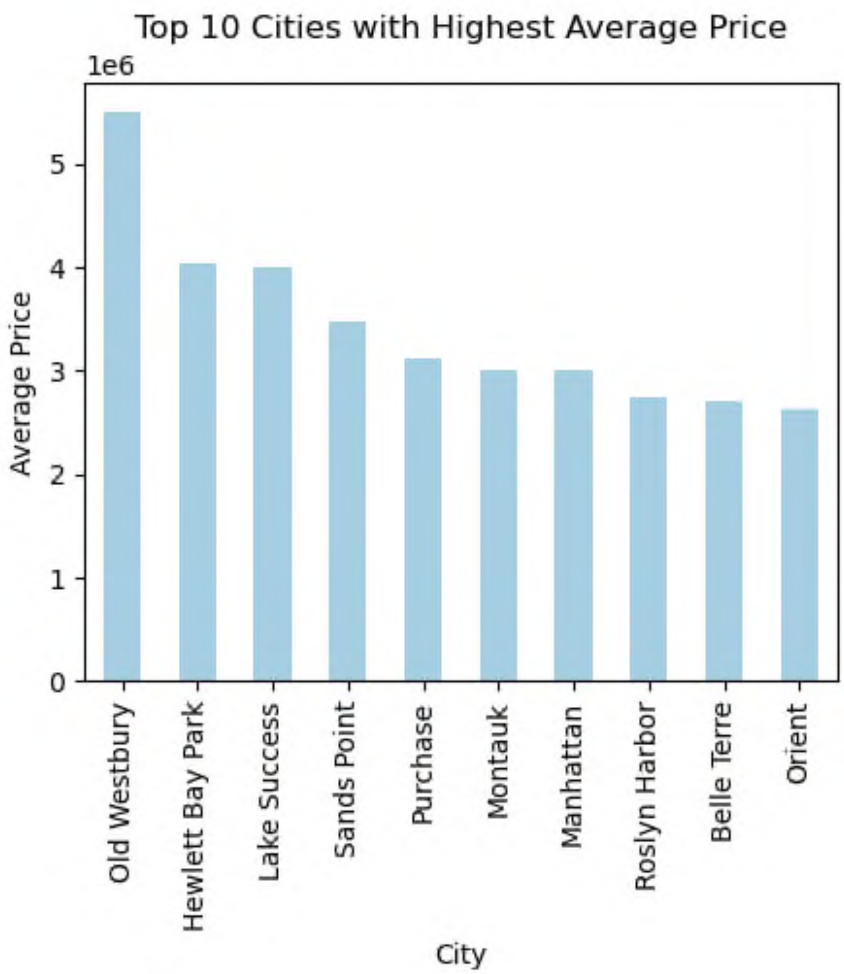
3. Geo Location vs. Price

3.1 Top 10 Cities with highest average price

```
# Group by 'city' and calculate the mean 'price'"
avg_price_by_city = cdf.groupby('city')['price'].mean()

# Sort the results by average price, in descending order
sorted_avg_price_by_city =
avg_price_by_city.sort_values(ascending=False)

# Select the top 10 cities
top_10_avg_price_by_city = sorted_avg_price_by_city.head(10)
```

```
top_10_avg_price_by_city.plot(kind='bar', figsize=(5, 4),
color=pairpalette[0], title='Top 10 Cities with Highest Average
Price', xlabel='City', ylabel='Average Price')
```

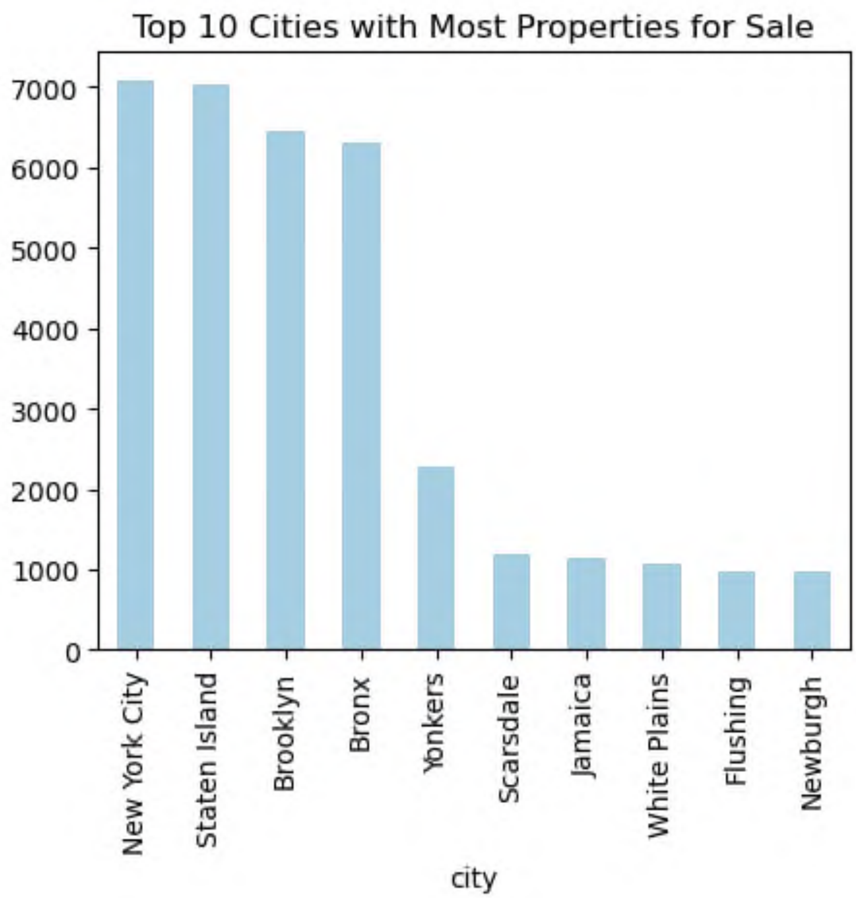
```
<Axes: title={'center': 'Top 10 Cities with Highest Average Price'},
xlabel='City', ylabel='Average Price'>
```



3.2 Top 10 Cities with Most Properties for Sale

```
cdf['city'].value_counts().head(10).plot(kind='bar', figsize=(5,4),
color=pairpalette[0], title='Top 10 Cities with Most Properties for
Sale')
```

```
<Axes: title={'center': 'Top 10 Cities with Most Properties for
Sale'}, xlabel='city'>
```



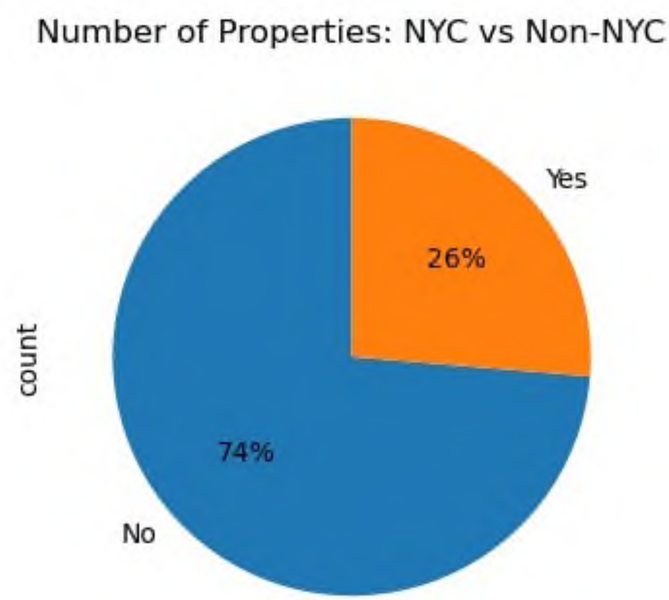
3.3 NYC vs. Non-NYC

```
print(cdf['nyc'].value_counts())
```

```
nyc
No     53294
Yes    20711
Name: count, dtype: int64
```

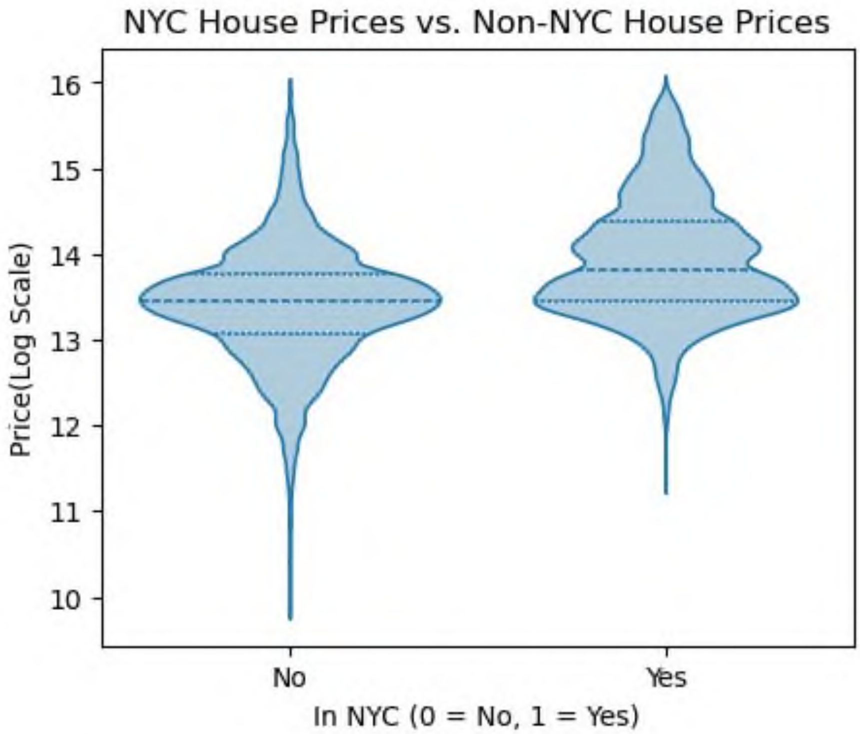
```
df['nyc'].value_counts().plot(kind='pie', figsize=(5, 4),
autopct='%1.0f%%', startangle=90, color=[pairpalette[1],
pairpalette[7]], title='Number of Properties: NYC vs Non-NYC')
```

```
<Axes: title={'center': 'Number of Properties: NYC vs Non-NYC'},
ylabel='count'>
```

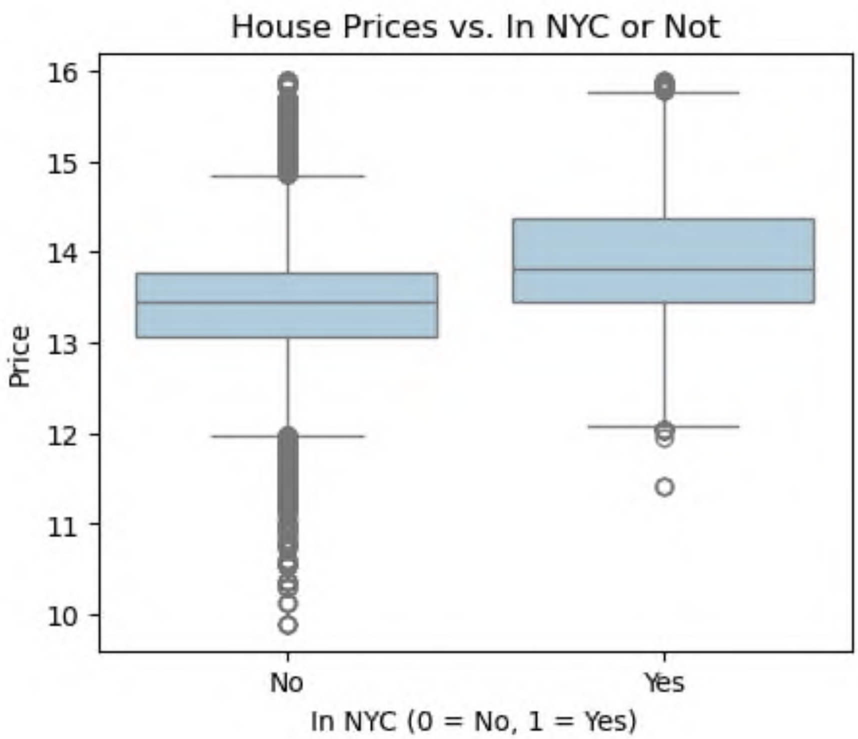



```
# Create a violin plot
plt.figure(figsize=(5,4))
sns.violinplot(x='nyc', y='log_price', data=cdf,
inner='quartile',edgecolor=pairpalette[1],color=pairpalette[0])
plt.title('NYC House Prices vs. Non-NYC House Prices')
plt.xlabel('In NYC (0 = No, 1 = Yes)')
plt.ylabel('Price(Log Scale)')
```

```
Text(0, 0.5, 'Price(Log Scale)')
```




```
# Create a box plot
plt.figure(figsize=(5,4))
sns.boxplot(x='nyc', y='log_price', data=cdf,color=pairpalette[0])
plt.title('House Prices vs. In NYC or Not')
plt.xlabel('In NYC (0 = No, 1 = Yes)')
plt.ylabel('Price')
plt.show()
```



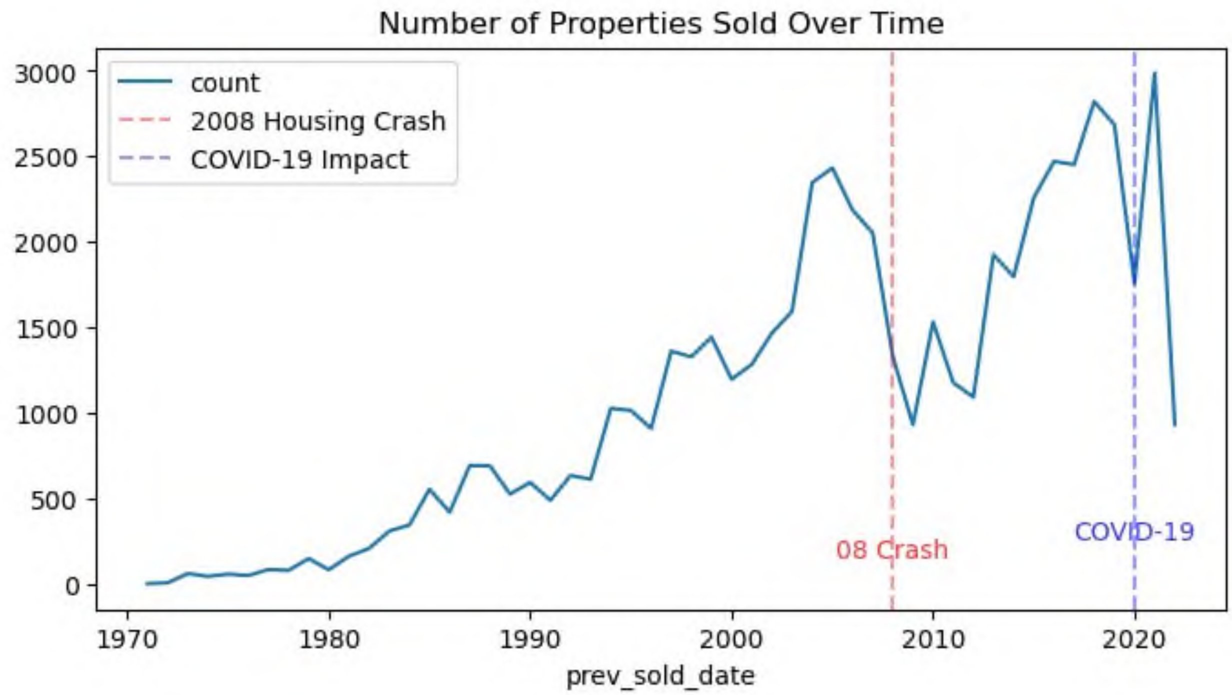
4. Status - Price

4.1 Properties Sold Over Time

```
df['prev_sold_date'] = pd.to_datetime(df['prev_sold_date'],
errors='coerce')
df['prev_sold_date'].dt.year.value_counts().sort_index().plot(kind='li
ne', figsize=(8,4), title='Number of Properties Sold Over Time')

# Annotations: Housing Market Crash (2008) and COVID-19 (2020)
plt.axvline(x=2008, color='red', linestyle='--', label='2008 Housing
Crash', alpha=0.4)
plt.axvline(x=2020, color='blue', linestyle='--', label='COVID-19
Impact', alpha=0.4)
plt.legend()

# Adding text annotations
plt.text(2008, 200, '08 Crash', color='red', ha='center', va='center',
fontsize=10, alpha=0.8)
plt.text(2020, 300, 'COVID-19', color='blue', ha='center',
va='center', fontsize=10, alpha=0.8)
plt.show()
```

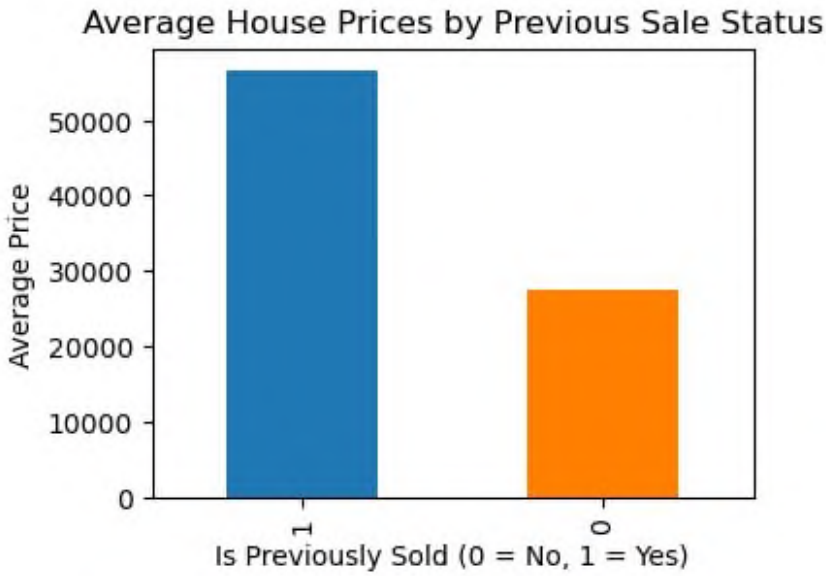
```
C:\Users\gsh\AppData\Local\Temp\ipykernel_12056\464379496.py:1:
UserWarning: Parsing dates in %d/%m/%Y format when dayfirst=False (the
default) was specified. Pass `dayfirst=True` or specify a format to
silence this warning.
```

```
  df['prev_sold_date'] = pd.to_datetime(df['prev_sold_date'],
errors='coerce')
```



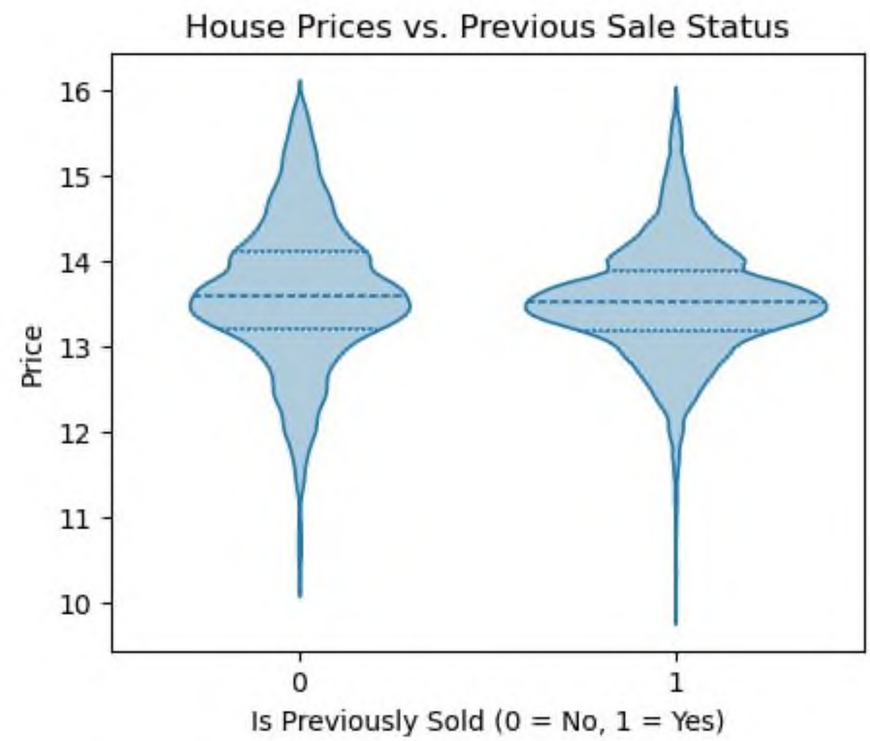
4.2 Is_pre_sold vs on_pre_sold

```
print(cdf['is_prev_sold'].value_counts())

is_prev_sold
1    50531
0    23474
Name: count, dtype: int64

df['is_prev_sold'].value_counts().plot(kind='bar', figsize=(4, 3),
color=[pairpalette[1], pairpalette[7]])
plt.title('Average House Prices by Previous Sale Status')
plt.xlabel('Is Previously Sold (0 = No, 1 = Yes)')
plt.ylabel('Average Price')
plt.show()
```

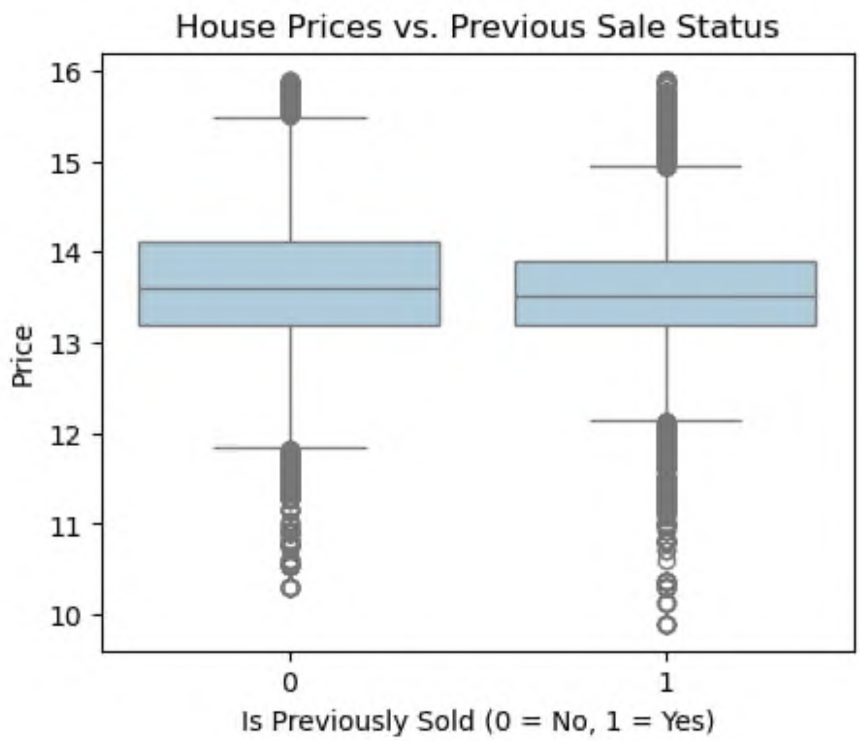



```python
# Create a violin plot
plt.figure(figsize=(5,4))
sns.violinplot(x='is_prev_sold', y='log_price', data=cdf,
inner='quartile',edgecolor=pairpalette[1],color=pairpalette[0])
plt.title('House Prices vs. Previous Sale Status')
plt.xlabel('Is Previously Sold (0 = No, 1 = Yes)')
plt.ylabel('Price')
```

```
Text(0, 0.5, 'Price')
```




```
# Create a box plot
plt.figure(figsize=(5,4))
sns.boxplot(x='is_prev_sold', y='log_price',
data=cdf,color=pairpalette[0])
plt.title('House Prices vs. Previous Sale Status')
plt.xlabel('Is Previously Sold (0 = No, 1 = Yes)')
plt.ylabel('Price')
plt.show()
```

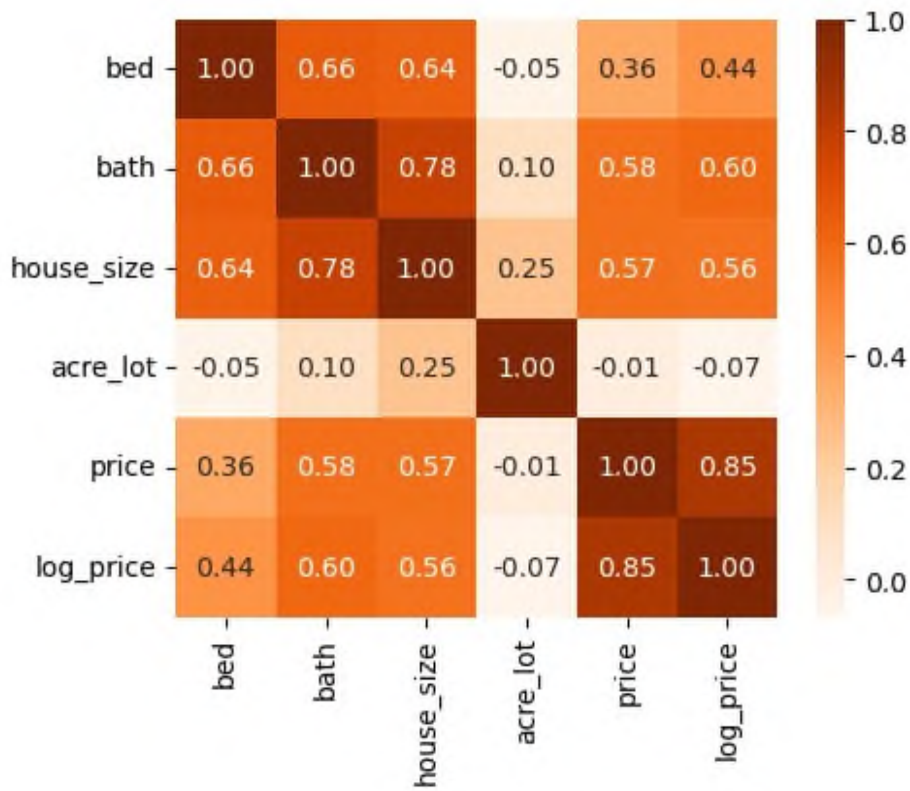


5. Summary & Others

5.1 Summary Correlation Heatmap

```
plt.figure(figsize=(5,4))
sns.heatmap(cdf[['bed','bath','house_size','acre_lot','price','log_price']].corr(), annot=True, cmap='Oranges', fmt='.2f')
```

```
<Axes: >
```



Question 2: Regression Model

```
model = smf.ols(formula= 'price ~ house_size+bed+bath+acre_lot+nyc',
data=cdf).fit()
print(model.summary())

                            OLS Regression Results

==============================================================================
========
Dep. Variable:                  price   R-squared:
0.487
Model:                            OLS   Adj. R-squared:
0.487
Method:                 Least Squares   F-statistic:
1.404e+04
Date:                Mon, 16 Dec 2024   Prob (F-statistic):
0.00
Time:                        11:30:08   Log-Likelihood:            -
1.1001e+06
No. Observations:               74005   AIC:
2.200e+06
Df Residuals:                   73999   BIC:
2.200e+06
Df Model:                           5
```

```
Covariance Type:            nonrobust

==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
Intercept  -1.803e+05   6917.582    -26.060      0.000   -1.94e+05   -
1.67e+05
nyc[T.Yes]  6.477e+05   5898.716    109.796      0.000    6.36e+05
6.59e+05
house_size   340.5055      3.465     98.263      0.000     333.714
347.297
bed         -7.48e+04   1952.312    -38.311      0.000   -7.86e+04
-7.1e+04
bath        2.201e+05   3055.031     72.052      0.000    2.14e+05
2.26e+05
acre_lot   -1.928e+05   7210.347    -26.741      0.000   -2.07e+05   -
1.79e+05
==============================================================================
========
Omnibus:                    41683.164   Durbin-Watson:
0.132
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
548442.073
Skew:                           2.455   Prob(JB):
0.00
Kurtosis:                      15.400   Cond. No.
8.78e+03
==============================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 8.78e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

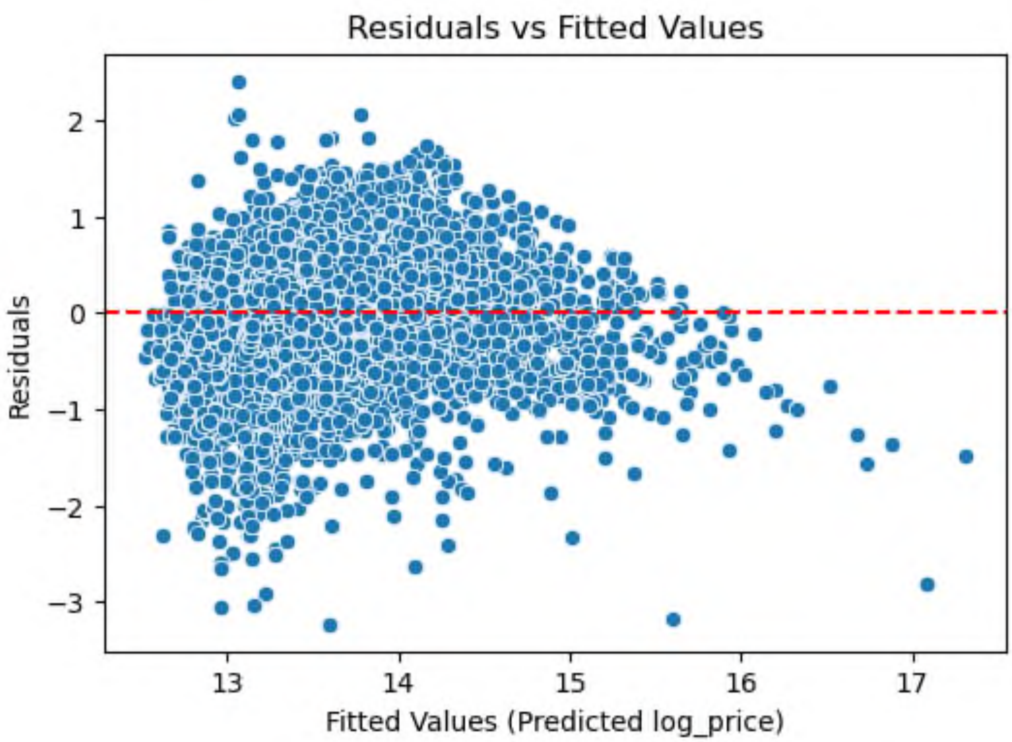
```
model = smf.ols(formula= 'log_price ~
house_size+bed+bath+acre_lot+nyc', data=cdf).fit()
print(model.summary())
```

```
                            OLS Regression Results

==============================================================================
========
Dep. Variable:              log_price   R-squared:
```

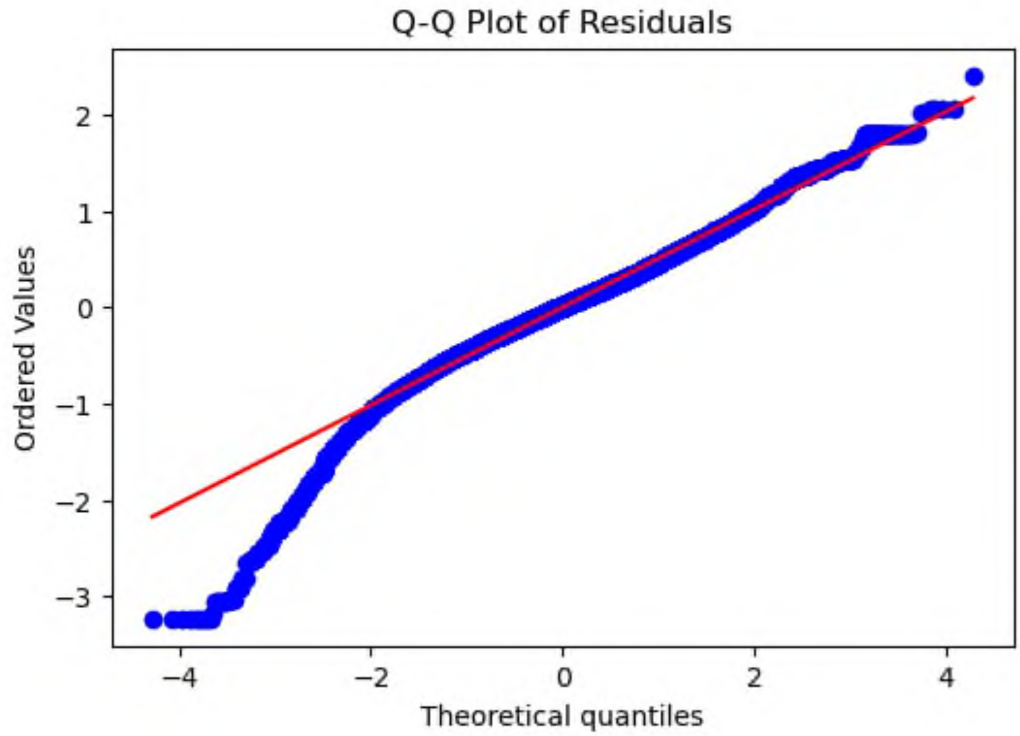
```
0.509
Model:                            OLS   Adj. R-squared:
0.509
Method:                 Least Squares   F-statistic:
1.535e+04
Date:                Mon, 16 Dec 2024   Prob (F-statistic):
0.00
Time:                        11:30:08   Log-Likelihood:
-55540.
No. Observations:               74005   AIC:
1.111e+05
Df Residuals:                   73999   BIC:
1.111e+05
Df Model:                           5

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
----------------------------------------------------------------------
--------
Intercept     12.5448      0.005   2444.450      0.000      12.535
12.555
nyc[T.Yes]     0.5208      0.004    119.008      0.000       0.512
0.529
house_size     0.0002   2.57e-06     77.802      0.000       0.000
0.000
bed            0.0021      0.001      1.430      0.153      -0.001
0.005
bath           0.1699      0.002     74.953      0.000       0.165
0.174
acre_lot      -0.1896      0.005    -35.454      0.000      -0.200
-0.179
======================================================================
========
Omnibus:                     5057.576   Durbin-Watson:
0.103
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
13929.687
Skew:                          -0.380   Prob(JB):
0.00
Kurtosis:                       4.985   Cond. No.
8.78e+03
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
```

```
correctly specified.
[2] The condition number is large, 8.78e+03. This might indicate that
there are
strong multicollinearity or other numerical problems.
```

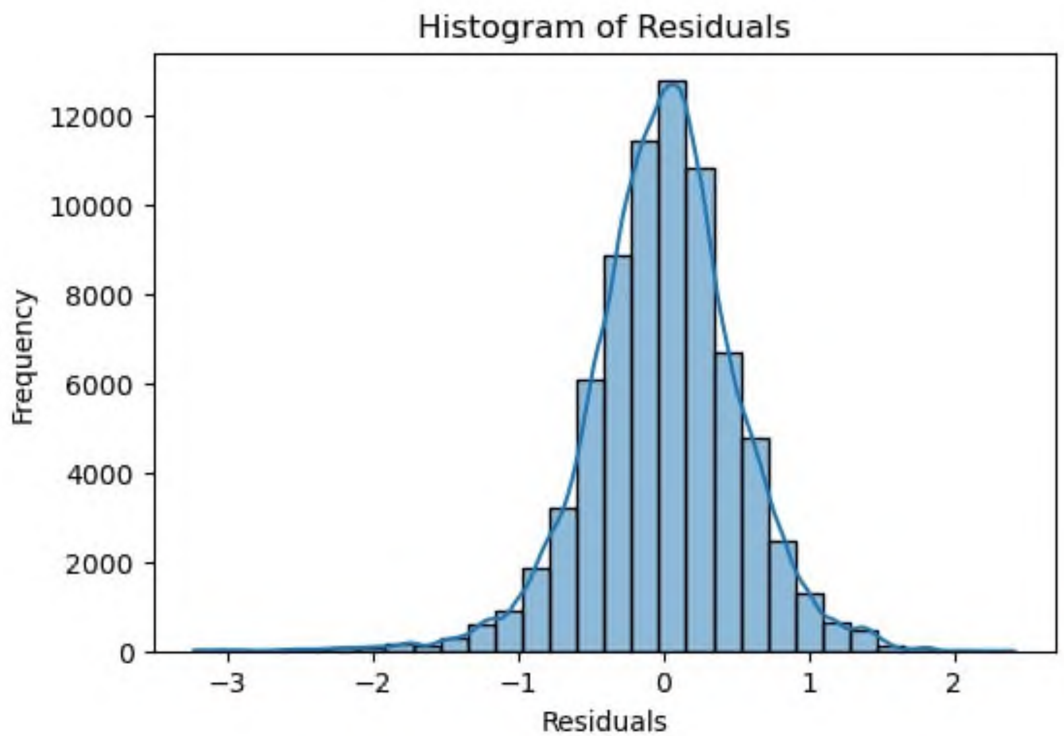
```
# Plot residuals vs fitted values
fitted_values = model.fittedvalues  # Predicted values
residuals = model.resid  # Residuals (actual - predicted)

plt.figure(figsize=(6,4))
sns.scatterplot(x=fitted_values, y=residuals)
plt.axhline(0, color='red', linestyle='--')  # Reference line at y=0
plt.title('Residuals vs Fitted Values')
plt.xlabel('Fitted Values (Predicted log_price)')
plt.ylabel('Residuals')
plt.show()
```



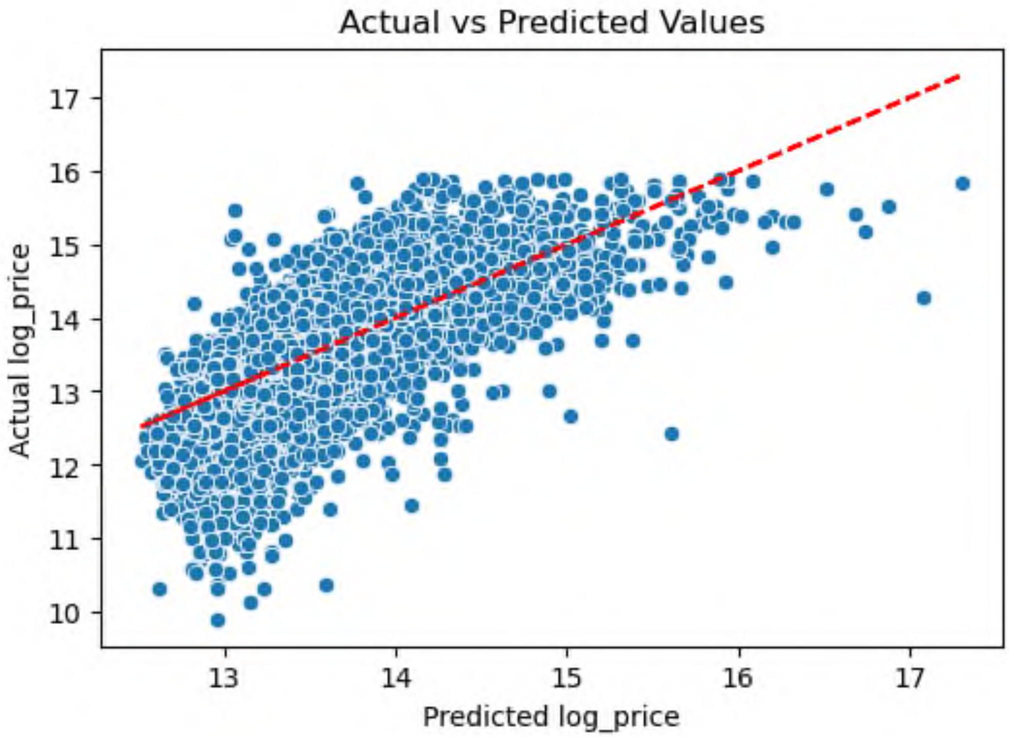

```
# Q-Q plot
plt.figure(figsize=(6,4))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title('Q-Q Plot of Residuals')
plt.show()
```




```
plt.figure(figsize=(6,4))
sns.histplot(residuals, kde=True, bins=30)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()
```




```
# Actual vs Predicted
plt.figure(figsize=(6,4))
sns.scatterplot(x=model.fittedvalues, y=cdf['log_price'])
plt.plot(model.fittedvalues, model.fittedvalues, color='red',
linestyle='--')  # Ideal line (y = x)
plt.title('Actual vs Predicted Values')
plt.xlabel('Predicted log_price')
plt.ylabel('Actual log_price')
plt.show()
```




```
drop_dup_cdf = cdf.drop_duplicates()
model = smf.ols(formula= 'log_price ~
house_size+bed+bath+acre_lot+nyc', data=drop_dup_cdf).fit()
print(model.summary())

                            OLS Regression Results

======================================================================
========
Dep. Variable:              log_price   R-squared:
0.494
Model:                            OLS   Adj. R-squared:
0.493
Method:                 Least Squares   F-statistic:
1545.
Date:                Mon, 16 Dec 2024   Prob (F-statistic):
0.00
Time:                        11:30:10   Log-Likelihood:
-6693.3
No. Observations:                7928   AIC:
1.340e+04
Df Residuals:                    7922   BIC:
1.344e+04
Df Model:                           5

Covariance Type:            nonrobust
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     12.4321      0.018    694.631      0.000      12.397      12.467
nyc[T.Yes]     0.5362      0.014     38.149      0.000       0.509       0.564
house_size     0.0002   8.22e-06     24.041      0.000       0.000       0.000
bed           -0.0109      0.005     -2.210      0.027      -0.021      -0.001
bath           0.2007      0.007     26.769      0.000       0.186       0.215
acre_lot      -0.2094      0.017    -12.109      0.000      -0.243      -0.175
==============================================================================
Omnibus:                      483.617   Durbin-Watson:                   0.799
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1168.148
Skew:                          -0.371   Prob(JB):                    2.19e-254
Kurtosis:                       4.728   Cond. No.                     8.87e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.87e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Question 3: Classification Model

```python
# Drop rows with missing 'nyc' values
df = df.dropna(subset=['nyc'])

# Save the cleaned DataFrame to a new CSV file
df.head()
```

```
     status  bed  bath  acre_lot    city     state  zip_code  house_size  \
0  for_sale    3   1.0      0.37  Accord  New York   12404.0  960
```

```
1  for_sale    3   2.0      0.38  Accord  New York   12404.0
1936
2  for_sale    2   1.0      0.41  Accord  New York   12404.0
832
3  for_sale    3   1.0      5.50  Accord  New York   12404.0
1900
4  for_sale    3   3.0      6.50  Accord  New York   12404.0
4000

  prev_sold_date   price nyc  is_prev_sold  log_price
0     2022-03-21  249900  No             1  12.428816
1     1989-01-06  319000  No             1  12.672946
2     2015-09-10  169500  No             1  12.040608
3            NaT  695000  No             0  13.451667
4     2021-07-30  250000  No             1  12.429216

df.tail()

         status  bed  bath  acre_lot   city     state  zip_code
house_size  \
84035  for_sale    3   3.0      0.65  Yulan  New York   12792.0
1480
84036  for_sale    4   1.0      1.64  Yulan  New York   12792.0
1692
84037  for_sale    4   1.0      1.64  Yulan  New York   12792.0
1692
84038  for_sale    5   2.0      0.13    NaN  New York       NaN
1925
84039  for_sale    2   1.0    110.00    NaN  New York   12523.0
1177

      prev_sold_date   price nyc  is_prev_sold  log_price
84035     2006-05-23  425000  No             1  12.959844
84036     2012-01-20  188500  No             1  12.146853
84037     2012-01-20  188500  No             1  12.146853
84038            NaT  710000  No             0  13.473020
84039            NaT  495000  No             0  13.112313

# Handle missing values in numeric columns by filling with the median
numeric_columns = ['bed', 'bath', 'acre_lot', 'house_size', 'price']
df[numeric_columns] = df[numeric_columns].apply(lambda x:
x.fillna(x.median()))

# Encode the target variable 'nyc' (Yes -> 1, No -> 0)
le_nyc = LabelEncoder()
df['nyc'] = le_nyc.fit_transform(df['nyc'])
df.head()

     status  bed  bath  acre_lot    city     state  zip_code
house_size  \
0  for_sale    3   1.0      0.37  Accord  New York   12404.0
```

```
960
1  for_sale    3   2.0       0.38  Accord  New York   12404.0
1936
2  for_sale    2   1.0       0.41  Accord  New York   12404.0
832
3  for_sale    3   1.0       5.50  Accord  New York   12404.0
1900
4  for_sale    3   3.0       6.50  Accord  New York   12404.0
4000

  prev_sold_date   price  nyc  is_prev_sold  log_price
0     2022-03-21  249900    0             1  12.428816
1     1989-01-06  319000    0             1  12.672946
2     2015-09-10  169500    0             1  12.040608
3            NaT  695000    0             0  13.451667
4     2021-07-30  250000    0             1  12.429216

# Define the feature set
features = ['bed', 'bath', 'acre_lot', 'house_size', 'price']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[features])
y = df['nyc']

# Initialize classifiers
log_reg = LogisticRegression(random_state=42)
decision_tree = DecisionTreeClassifier(random_state=42)
rf_model = RandomForestClassifier(random_state=42)

# Define Stratified K-Fold cross-validation (5 folds)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Perform Stratified K-Fold cross-validation for Logistic Regression
log_reg_scores = cross_val_score(log_reg, X_scaled, y, cv=skf,
scoring='accuracy')
print("Logistic Regression Stratified K-Fold Accuracy Scores:",
log_reg_scores)
print(f"Logistic Regression Mean Accuracy:
{np.mean(log_reg_scores):.4f}")

# Perform Stratified K-Fold cross-validation for Decision Tree
decision_tree_scores = cross_val_score(decision_tree, X_scaled, y,
cv=skf, scoring='accuracy')
print("\nDecision Tree Stratified K-Fold Accuracy Scores:",
decision_tree_scores)
print(f"Decision Tree Mean Accuracy:
{np.mean(decision_tree_scores):.4f}")

# Perform Stratified K-Fold cross-validation for Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
rf_scores = cross_val_score(rf_model, X_scaled, y, cv=skf,
scoring='accuracy')

print("\nRandom Forest Stratified K-Fold Accuracy Scores:", rf_scores)
print(f"Random Forest Mean Accuracy: {np.mean(rf_scores):.4f}")
```

```
Logistic Regression Stratified K-Fold Accuracy Scores: [0.78807711
0.78873156 0.78825559 0.78891004 0.78908853]
Logistic Regression Mean Accuracy: 0.7886

Decision Tree Stratified K-Fold Accuracy Scores: [0.99280105 0.993396
0.99303903 0.99262256 0.99113517]
Decision Tree Mean Accuracy: 0.9926

Random Forest Stratified K-Fold Accuracy Scores: [0.99369348
0.99375297 0.99393146 0.99327701 0.99440743]
Random Forest Mean Accuracy: 0.9938
```

```
# Placeholder function for evaluating feature combinations
def evaluate_feature_combinations(model, X, y, features):
    best_combo = None
    best_score = 0
    for r in range(1, len(features) + 1):
        for combo in combinations(features, r):
            X_combo = X[list(combo)]
            scores = cross_val_score(model, X_combo, y, cv=5,
scoring='roc_auc')
            mean_score = np.mean(scores)
            if mean_score > best_score:
                best_score = mean_score
                best_combo = combo
    best_model = model.fit(X[list(best_combo)], y)
    return best_combo, best_score, best_model

# Convert the scaled data back to a DataFrame for easy feature
selection
X_scaled_df = pd.DataFrame(X_scaled, columns=features)


# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y,
test_size=0.2, random_state=42)

# Train Random Forest model
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Test Accuracy: {rf_accuracy:.2f}")
```

```
Random Forest Test Accuracy: 0.99
```

```python
# Placeholder for best_rf_combo and best_rf_score
best_rf_combo = features
best_rf_score = rf_accuracy
best_rf_model = rf_model
```

```python
# Evaluate Logistic Regression
print("Evaluating Logistic Regression...")
best_log_reg_combo, best_log_reg_score, best_log_reg_model =
evaluate_feature_combinations(
    log_reg, X_scaled_df, y, features
)
```

```
Evaluating Logistic Regression...
```

```python
# Evaluate Decision Tree
print("\nEvaluating Decision Tree...")
best_tree_combo, best_tree_score, best_tree_model =
evaluate_feature_combinations(
    decision_tree, X_scaled_df, y, features
)
```

```
Evaluating Decision Tree...
```

```python
# Evaluate Random Forest model
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy:.2f}")
```

```
Random Forest Accuracy: 0.99
```

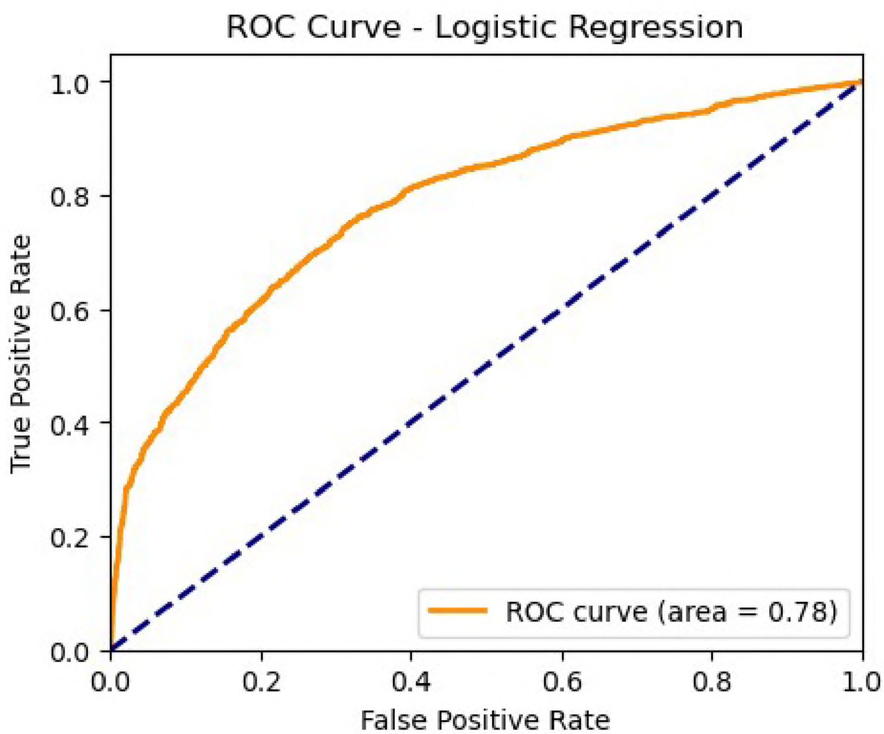
```python
# Display the best results
print("\nBest Logistic Regression Feature Combination:",
best_log_reg_combo, "with ROC AUC:", best_log_reg_score)
print("Best Decision Tree Feature Combination:", best_tree_combo,
"with ROC AUC:", best_tree_score)
print("Best Random Forest Feature Combination:", best_rf_combo, "with
ROC AUC:", best_rf_score)
```

```
Best Logistic Regression Feature Combination: ('bed', 'acre_lot',
'house_size', 'price') with ROC AUC: 0.7677090297237096
Best Decision Tree Feature Combination: ('acre_lot',) with ROC AUC:
0.6814030705289911
Best Random Forest Feature Combination: ['bed', 'bath', 'acre_lot',
'house_size', 'price'] with ROC AUC: 0.9941099476439791
```

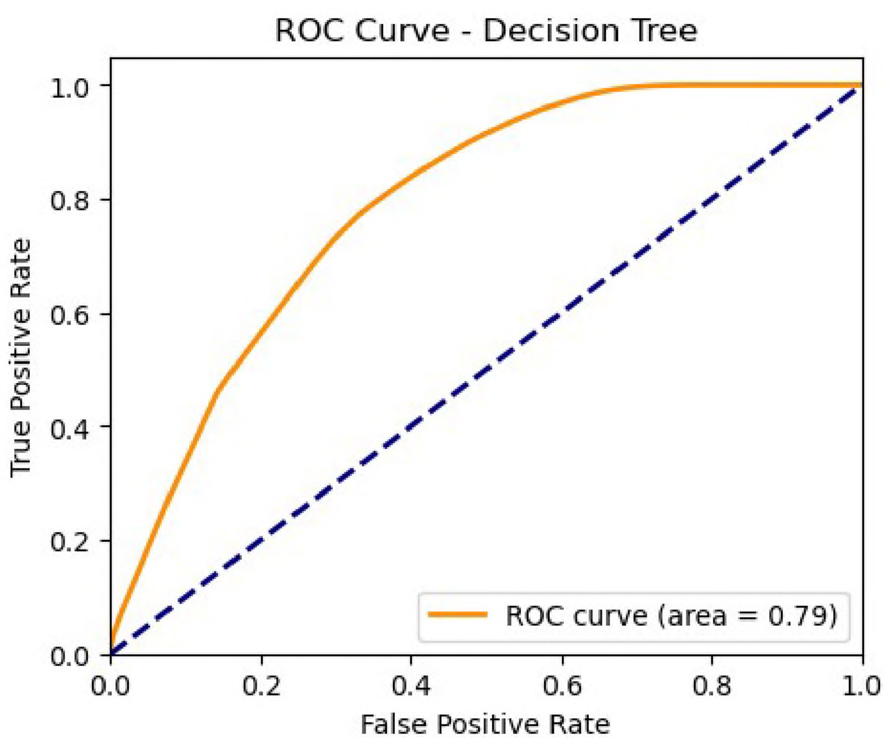
```python
def plot_roc_curve(model, X, y, model_name, feature_combo):

    # Ensure feature_combo is a list, not a tuple
    feature_combo = list(feature_combo)
```

```
    # Predict probabilities for the positive class
    y_pred_prob = model.predict_proba(X[feature_combo])[:, 1]

    # Calculate the ROC curve
    fpr, tpr, thresholds = roc_curve(y, y_pred_prob)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.figure()
    plt.figure(figsize=(5, 4))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve
(area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {model_name}')
    plt.legend(loc="lower right")
    plt.show()

# Plot ROC curves for both models
plot_roc_curve(best_log_reg_model, X_scaled_df, y, "Logistic
Regression", best_log_reg_combo)
plot_roc_curve(best_tree_model, X_scaled_df, y, "Decision Tree",
best_tree_combo)
plot_roc_curve(best_rf_model, X_scaled_df, y, "Random Forest",
best_rf_combo)
```

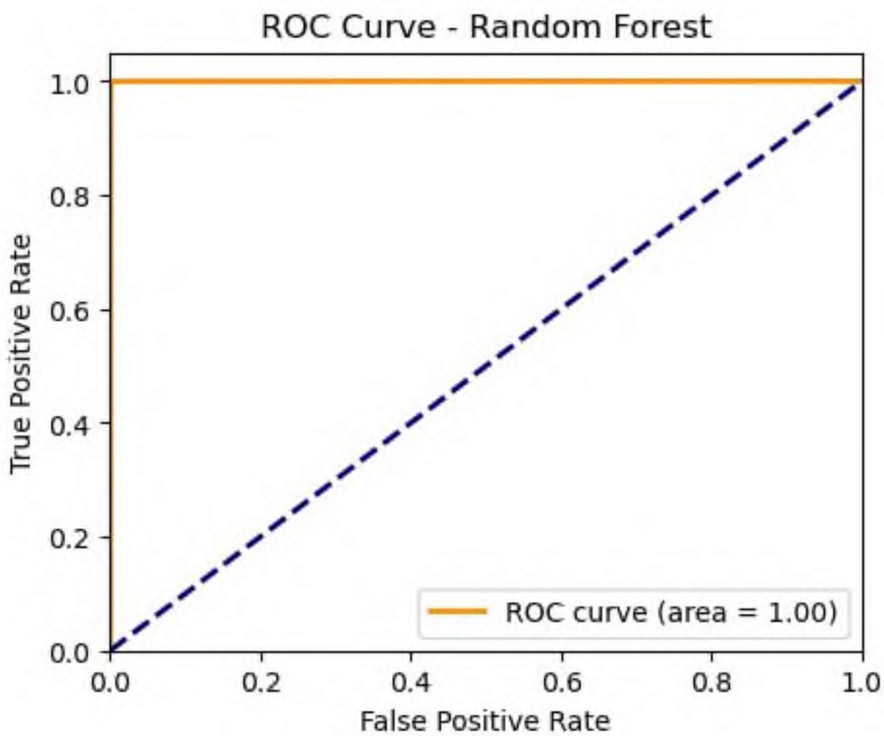
```
<Figure size 640x480 with 0 Axes>
```



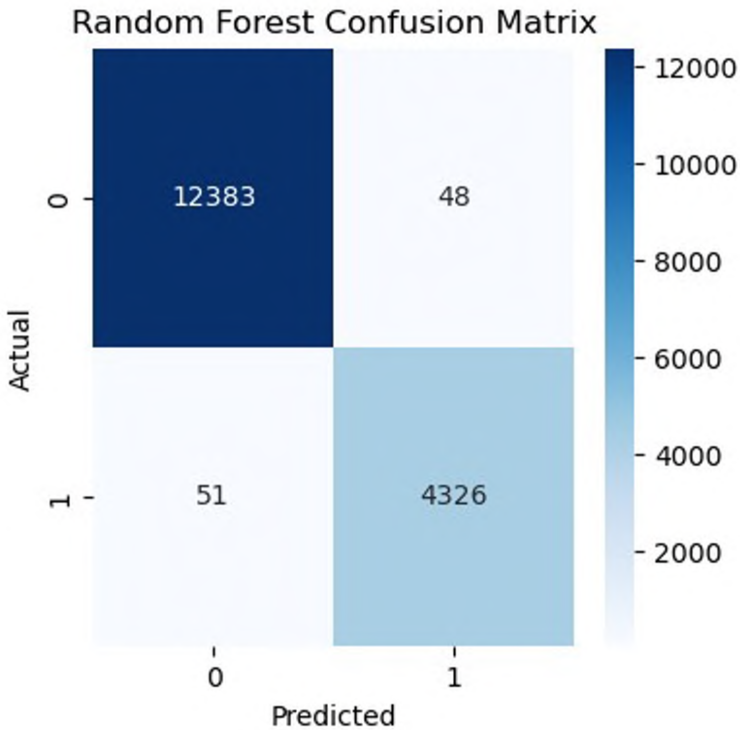

```
&lt;Figure size 640x480 with 0 Axes&gt;
```




```
&lt;Figure size 640x480 with 0 Axes&gt;
```




```python
# Confusion Matrix for Random Forest
rf_cm = confusion_matrix(y_test, rf_predictions)
plt.figure(figsize=(4, 4))
sns.heatmap(rf_cm, annot=True, fmt='d', cmap='Blues')
plt.title("Random Forest Confusion Matrix")
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.show()
```


Random Forest Confusion Matrix
12383
48
51
4326
Actual
Predicted
0
1
12000
10000
8000
6000
4000
2000