MTH781P Data Analysis – Group 7 Project Report

Real estate has consistently been a hot topic, with property sales data frequently sparking discussion. By investigating a historical dataset containing data on property sales, this coursework aims to explore the key characteristics of data, and develop predictive models. Specifically, it includes a regression model to predict price and a classification model to predict whether a property is located in the New York City area. This report is structured into three sections, each addressing one of these objectives.

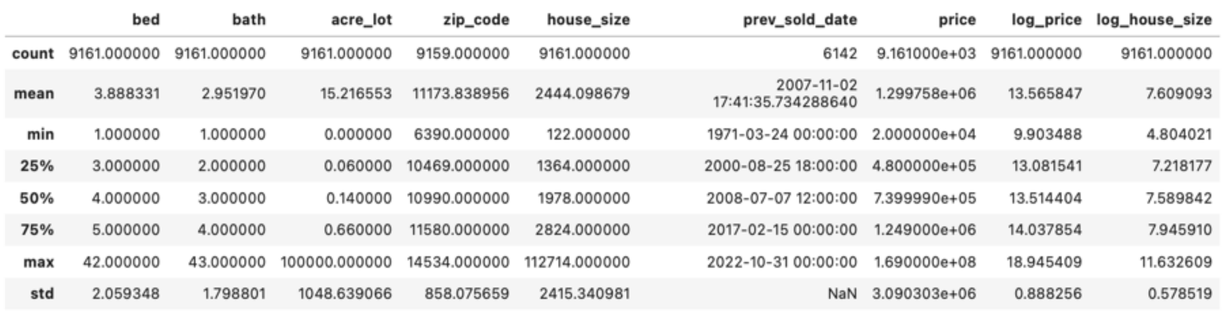
Q1. Data exploration

The primary objective of data cleaning is to ensure that the dataset is prepared for analysis. This encompasses correcting data types, resolving missing values, removing duplicate entries, and preserving consistency in critical columns, including 'price', 'bed', 'bath', 'house_size', and 'city'.

To begin with, by identifying the data type and understanding the basic information of the data, we determined that the key columns are 'price', 'bed', 'bath', and 'house_size' because they may have a high relationship with house price based on common sense. Then we adjusted all the data types such as converting critical data into numerical data. Columns, such as 'price' and 'house_size', were converted into log formats to enable reliable computations. After adjusting all data types, we removed rows with missing values in critical fields to ensure the reliability of calculations, particularly for metrics such as average house prices. Retaining incomplete data would have introduced biases or errors in the results, compromising the integrity of the insights drawn from the analysis. Duplicate values were also eliminated to prevent redundancy and avoid skewing analyses, as duplicates can inflate certain metrics and lead to misleading conclusions. Finally, the dataset was validated to ensure that all inconsistencies had been resolved after the cleansing steps were finished, resulting in a structured and dependable dataset that could be used for further analysis.

After data cleaning, we can find that there are 9161 properties ready for sale in New York, among which 6142 have been sold before. These houses are distributed in 457 different areas of New York, with 3056 located in NYC and 6105 located outside of NYC. To identify the average characteristics of data, we found that the average house price was $1,299,758, the average house size was 2444.09, and each house had an average of nearly 4 beds and 3 bathrooms. The cheapest house is located in Ticonderoga, New York, with a total price of $20,000 and 1584 square feet. It has 3 beds and 1 bathroom and was previously sold on March 28, 2012. However, the most expensive house is located in New York City, with a house size of 8255 square feet but a high price of $169 million. It has 6 beds and 9 bathrooms.

General information of final_data



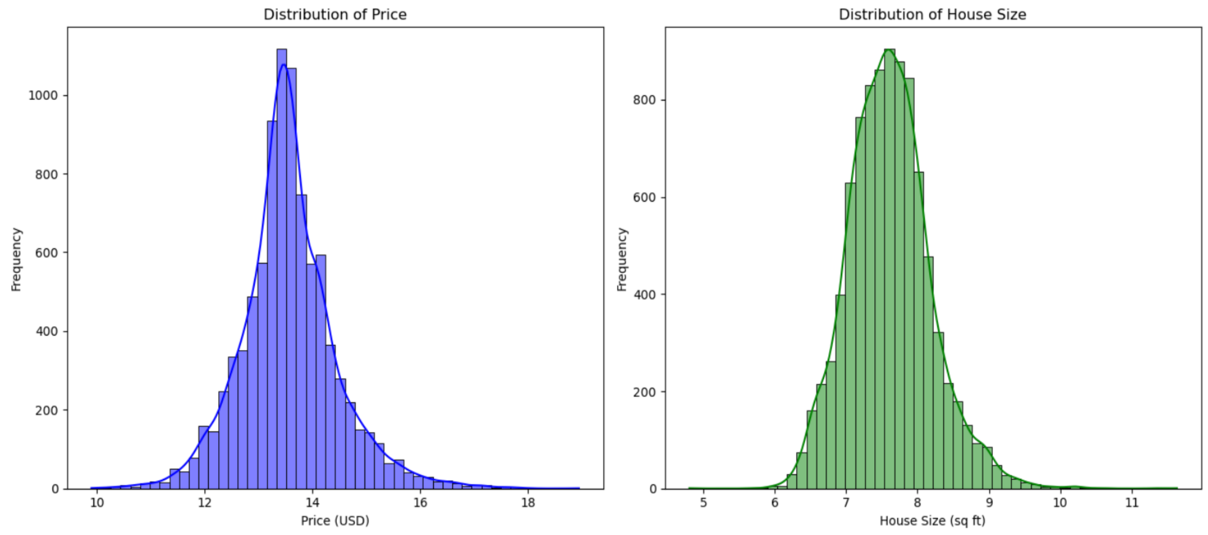
	bed	bath	acre_lot	zip_code	house_size	prev_sold_date	price	log_price	log_house_size
count	9161.000000	9161.000000	9161.000000	9159.000000	9161.000000	6142	9.161000e+03	9161.000000	9161.000000
mean	3.888331	2.951970	15.216553	11173.838956	2444.098679	2007-11-02 17:41:35.734288640	1.299758e+06	13.565847	7.609093
min	1.000000	1.000000	0.000000	6390.000000	122.000000	1971-03-24 00:00:00	2.000000e+04	9.903488	4.804021
25%	3.000000	2.000000	0.060000	10469.000000	1364.000000	2000-08-25 18:00:00	4.800000e+05	13.081541	7.218177
50%	4.000000	3.000000	0.140000	10990.000000	1978.000000	2008-07-07 12:00:00	7.399990e+05	13.514404	7.589842
75%	5.000000	4.000000	0.660000	11580.000000	2824.000000	2017-02-15 00:00:00	1.249000e+06	14.037854	7.945910
max	42.000000	43.000000	100000.000000	14534.000000	112714.000000	2022-10-31 00:00:00	1.690000e+08	18.945409	11.632609
std	2.059348	1.798801	1048.639066	858.075659	2415.340981	NaN	3.090303e+06	0.888256	0.578519

Since all houses are located in New York State, whether a house is in NYC or not will have a certain impact on housing prices. We found through group surveys that the average housing price in NYC is $1,998,642 and the average house size is 2253.48 square feet, while housing prices outside NYC are relatively cheaper with an average of $949,914.7 and a house size of 2539.52 square feet.

Meanwhile, housing prices vary in different cities. According to the data, the top five cities in terms of housing prices are Thompson Ridge ($15,224,500), Rochdale Village($9,800,000), Waccabuc($9,425,000), Old Westbury($5,500,000) and Sands Point($5,035,706). The bottom five house price rankings are Clintonville ($49,500), Witherbee($59,000), Mineville($93,266.67), Port Henry($117,457.12), and North Hudson($129,900). From this, it can be inferred that the maximum difference in average housing prices between different cities is $15,175,000.

Now, we will further show the specific characteristics of the data through specific charts.

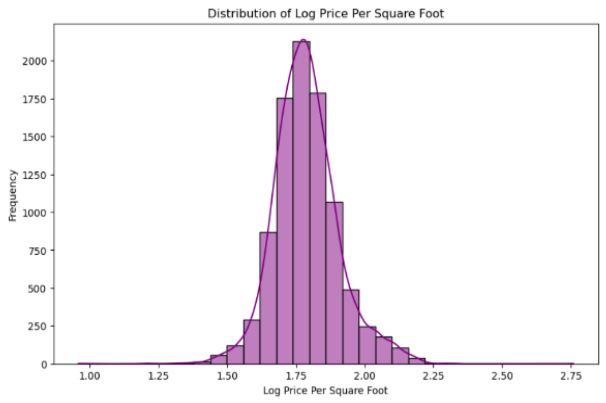
1. Distribution of Price and House Size



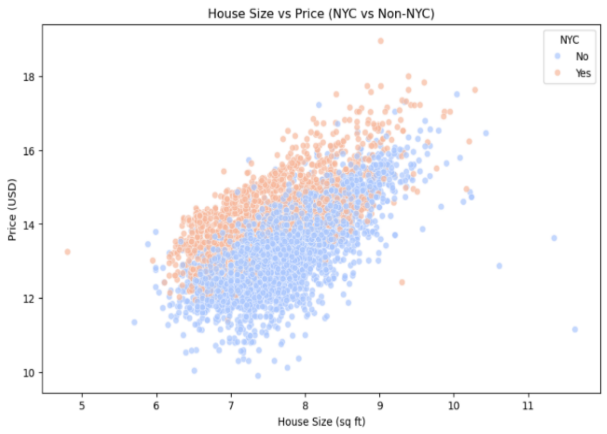
This visualization consists of two histograms side by side: the first shows the distribution of property prices (log-transformed for normalization), and the second depicts the distribution of house sizes (in square feet). Property prices exhibit a right-skewed distribution, with a higher concentration of properties in the mid-to-lower price range and a smaller proportion of high-value properties extending the tail. In contrast, house sizes follow a more normal distribution, with a clear central peak suggesting that most houses fall within a specific size range, around 1,500 to 2,500 square feet. These distributions provide a foundational understanding of the dataset's composition and highlight areas of interest for deeper analysis, such as high-value outliers.

2. Distribution of Log Price Per Square Foot

This histogram visualizes the distribution of log price per square foot for all properties, offering insights into pricing efficiency across the dataset. The distribution is roughly normal, peaking around 1.75 log price per square foot, indicating a balanced spread of pricing. This normalization supports comparative analyses and highlights areas of potential interest, such as outliers on either end of the spectrum.

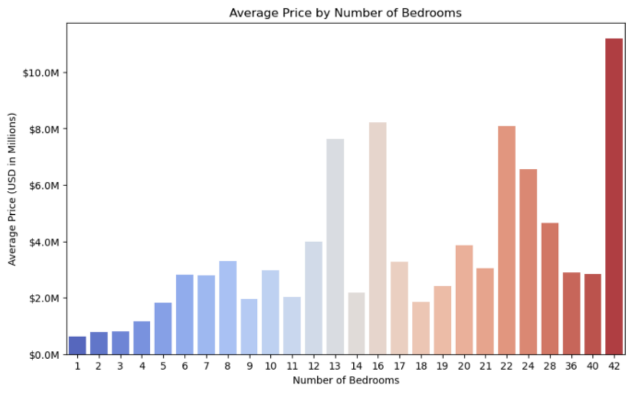


3. House Size vs. Price (NYC vs. Non-NYC Scatter Plot)
This scatter plot compares house size (x-axis) and price (y-axis) with color-coded points distinguishing NYC and non-NYC properties. A positive correlation exists between house size and price across both categories, as expected, with larger properties generally commanding higher prices.

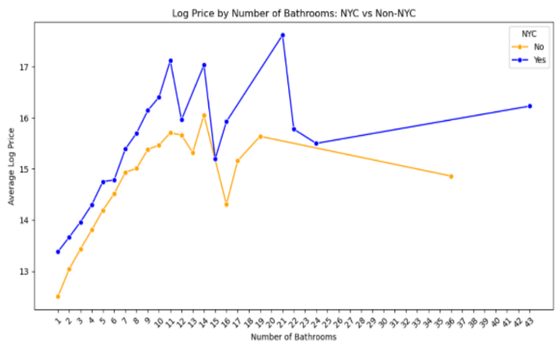


However, NYC properties consistently command significantly higher prices for similar-sized properties compared to non-NYC ones, reflecting the premium nature of NYC's real estate market. Additionally, outliers in larger house sizes, especially in the non-NYC category, suggest the presence of luxury or estate properties skewing the distribution. This plot underscores the impact of location on property valuation.

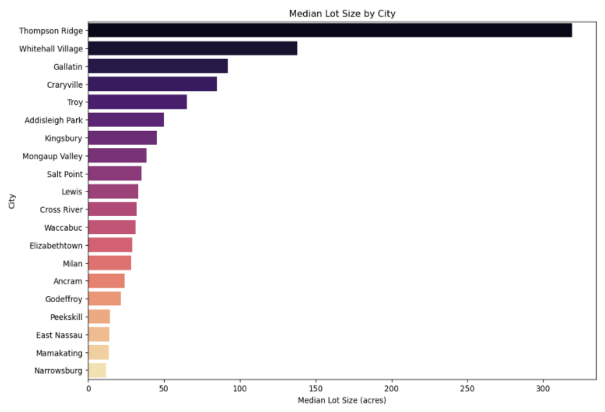
4. Average Price by Number of Bedrooms
This bar plot presents the average price (in USD) for properties grouped by the number of bedrooms. Prices tend to increase steadily with the number of bedrooms, reflecting the added value larger homes provide to buyers. The trend peaks significantly for properties with more than 20 bedrooms, which are likely luxury estates or outliers rather than representative properties. The consistent upward trajectory underscores the general correlation between the number of bedrooms and property value. However, the steep price increases at higher bedroom counts may warrant additional exploration to assess their representativeness.



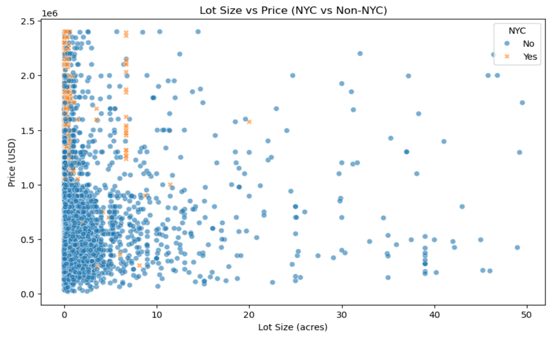
5. Log Price by Number of Bathrooms (NYC vs. Non-NYC Line Plot)
This line plot highlights the average log price for properties with varying numbers of bathrooms, separated into NYC and non-NYC categories. Prices rise as the number of bathrooms increases, aligning with expectations of higher property value for greater amenities. NYC properties consistently maintain higher prices across all bathroom counts, reflecting their premium real estate market. Interestingly, NYC shows greater fluctuations in higher bathroom counts, which could stem from limited data points or the influence of unique luxury properties. Non-NYC properties follow a smoother trend, suggesting more uniform pricing for additional bathrooms outside NYC.



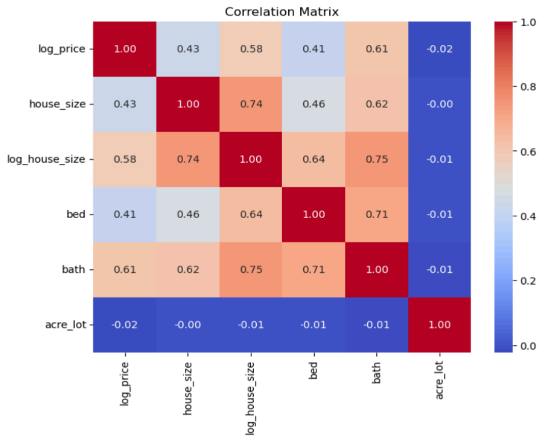
6. Median Lot Size by City
A bar plot displaying the median lot size for the cities with the largest lot sizes. Cities like Thompson Ridge and Whitehall Village stand out with notably larger median lot sizes compared to others, reflecting the presence of expansive rural or suburban properties. The stark disparity between these cities and urban areas highlights the influence of geographic location and zoning policies on property characteristics. This visualization underscores the variability in land availability and usage across different regions.



7. Lot Size vs. Log Price
This scatter plot compares lot size and log price, providing a granular view of their relationship. Most properties cluster at smaller lot sizes, exhibiting a wide range of log prices. Larger lot sizes show greater price variability, suggesting that additional factors such as location, house size, or amenities significantly influence pricing. This plot emphasizes that lot size alone is not a definitive predictor of property value, warranting further exploration of other contributing features.



8. Correlation Matrix
A heatmap showing the correlation coefficients among numerical features, including log_price, house_size, log_house_size, bedrooms, bathrooms, and acre_lot size. Bathrooms and log_house_size exhibit strong positive correlations with log price, reaffirming their importance in property valuation. Conversely, acre_lot size shows a weak correlation, suggesting its limited direct impact on pricing. This visualization provides a comprehensive overview of feature interactions and aids in prioritizing variables for predictive modeling.



	log_price	house_size	log_house_size	bed	bath	acre_lot
log_price	1.00	0.43	0.58	0.41	0.61	-0.02
house_size	0.43	1.00	0.74	0.46	0.62	-0.00
log_house_size	0.58	0.74	1.00	0.64	0.75	-0.01
bed	0.41	0.46	0.64	1.00	0.71	-0.01
bath	0.61	0.62	0.75	0.71	1.00	-0.01
acre_lot	-0.02	-0.00	-0.01	-0.01	-0.01	1.00

Q2. Regression model to predict price

Based on the data exploration above, we select variables like bed, bath, house size, and the location (whether the property is in NYC) to develop the regression model because we can see in the figure Correlation Matrix, that bath, bed, and log_house_size have a high and positive correlation with log_prize. Meanwhile, the house prices in NYC are higher than those outside NYC. Therefore, we build the model based on these variables. The "nyc" column, which indicates whether a property is in New York City, is converted into a numeric format, with "Yes" assigned a value of 1 and "No" a value of 0.

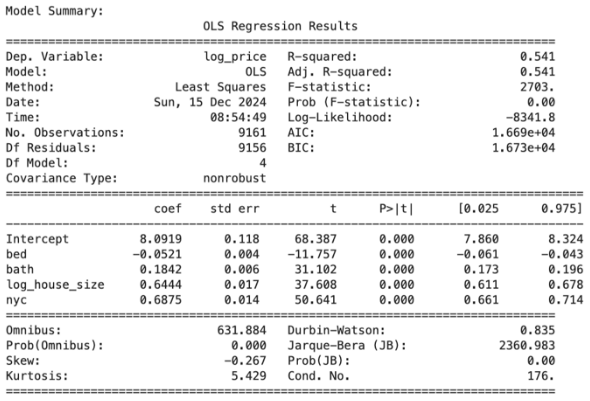
The models tested are simple linear regression and polynomial regression. First, we divide the data set into a training set and a test set, and perform basic OLS regression on the training set. This model using the base variables achieves an R-squared value of 0.538, indicating that approximately 53.8% of the variability in log_price could be explained by the predictors. All variables in the model (bed, bath, log_house_size, and nyc) are statistically significant with p-values < 0.05. The RMSE of the model is 0.5931 and the MAE is 0.4599, which means that the average error between the model prediction and the true value is about 0.46 to 0.6 units. Since our target variable log_price has a wide range of values, we believe that this model performs well. We can add interaction terms or use polynomial regression to make predictions on this basis.

Therefore, based on this model, we conducted the following regressions.

Model 1: Simple Linear Regression
Formula: log_price ~ bed + bath + log_house_size + NYC

This model 1 regression achieves an R-squared value of 0.541, indicating that approximately 54.1% of the variability in log_price could be explained by the predictors. All variables in the model (bed, bath, log_house_size, and nyc) are statistically significant with p-values < 0.005. The coefficients indicated that for every additional bathroom, log_price increases by 0.1842 units, holding other variables constant.

Properties in New York City have a higher log_price by 0.6875 units compared to properties outside the city. Every time log_house_size increases by 1, log_price will increase by 0.6444. But an increase in the number of bedrooms will have a negative impact on house prices. This shows that the size of the house, whether the house is located in NYC and the number of bathrooms have a significant positive impact on housing prices.



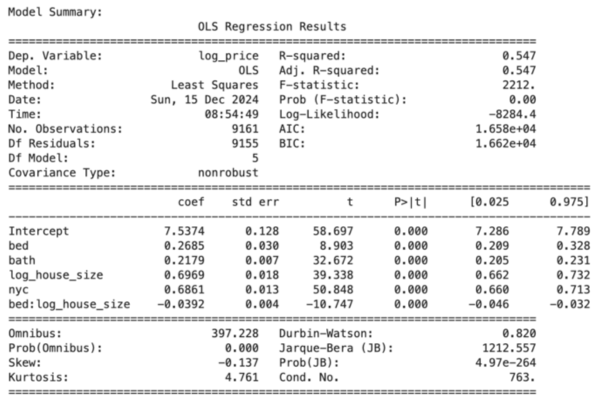
```
Model Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.541
Model:                            OLS   Adj. R-squared:                  0.541
Method:                 Least Squares   F-statistic:                     2703.
Date:                Sun, 15 Dec 2024   Prob (F-statistic):               0.00
Time:                        08:54:49   Log-Likelihood:                -8341.8
No. Observations:                9161   AIC:                         1.669e+04
Df Residuals:                    9156   BIC:                         1.673e+04
Df Model:                           4
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept          8.0919      0.118     68.387      0.000       7.860       8.324
bed               -0.0521      0.004    -11.757      0.000      -0.061      -0.043
bath               0.1842      0.006     31.102      0.000       0.173       0.196
log_house_size     0.6444      0.017     37.608      0.000       0.611       0.678
nyc                0.6875      0.014     50.641      0.000       0.661       0.714
==============================================================================
Omnibus:                      631.884   Durbin-Watson:                   0.835
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2360.983
Skew:                          -0.267   Prob(JB):                         0.00
Kurtosis:                       5.429   Cond. No.                         176.
==============================================================================
```

In many practical cases, the impact of independent variables on dependent variables may not only be a single linear effect, but may also be determined by the combined effect of two or more independent variables. Adding interaction terms to regression analysis can capture the impact of the interaction between independent variables on the dependent variable. Thus, interaction terms are added between correlated variables, such as "bed:bath", "bath:log_house_size", and "bed:log_house_size". After several tests, we finally chose to add bed:log_house_size into the model because this model has a better fitting effect and coefficients of all the variable are significant.

Model 2: Interaction Terms

Formula: log_price ~ bed + bath + log_house_size + nyc + bed:log_house_size

This regression achieves an R-squared value of 0.547, indicating that approximately 54.7% of the variability in log_price could be explained by the predictors. All variables in the model (bed, bath, log_house_size, and nyc) are statistically significant with p-values < 0.005. The coefficients indicated that for every additional bathroom, log_price increases by 0.2179 units. Properties in New York City have a higher log_price by 0.6861 units compared to properties outside the city. Every time log_house_size increases by 1, log_price will increase by 0.6969.



```
Model Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.547
Model:                            OLS   Adj. R-squared:                  0.547
Method:                 Least Squares   F-statistic:                     2212.
Date:                Sun, 15 Dec 2024   Prob (F-statistic):               0.00
Time:                        08:54:49   Log-Likelihood:                -8284.4
No. Observations:                9161   AIC:                         1.658e+04
Df Residuals:                    9155   BIC:                         1.662e+04
Df Model:                           5
Covariance Type:            nonrobust
======================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
Intercept              7.5374      0.128     58.697      0.000       7.286       7.789
bed                    0.2685      0.030      8.903      0.000       0.209       0.328
bath                   0.2179      0.007     32.672      0.000       0.205       0.231
log_house_size         0.6969      0.018     39.338      0.000       0.662       0.732
nyc                    0.6861      0.013     50.848      0.000       0.660       0.713
bed:log_house_size    -0.0392      0.004    -10.747      0.000      -0.046      -0.032
==============================================================================
Omnibus:                      397.228   Durbin-Watson:                   0.820
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1212.557
Skew:                          -0.137   Prob(JB):                    4.97e-264
Kurtosis:                       4.761   Cond. No.                         763.
==============================================================================
```

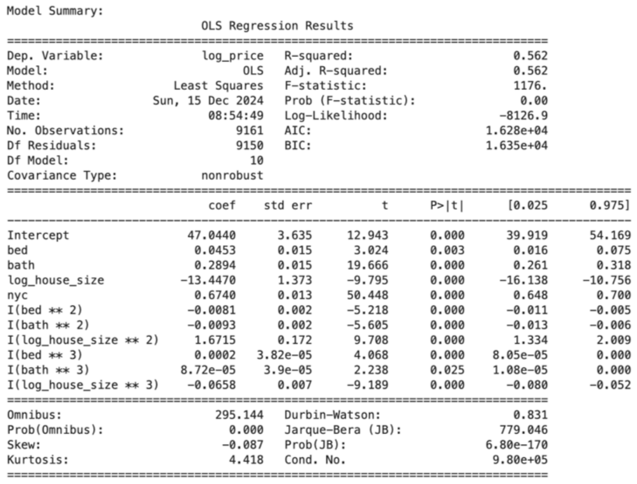
Interestingly, since we added the variable bed:log_house_size, the coefficient of bed changed from negative to positive. For every additional unit of room, the house price increases by 0.2685 units. However, the coefficient of bed:log_house_size indicates that there is a negative interaction between the log_house_size and the number of bedrooms. The impact of the number of bedrooms on the house price will decrease by 0.0392 units with every increase of 1 unit in the log_house_size. The larger the house area, the smaller the increase in house price brought by adding one bedroom. The house size, whether the house is located in NYC and the number of bathrooms still have a significant positive impact on housing prices.

In addition, we also want to use polynomial regression to capture potential nonlinear relationships, so we write the model for any given degree and set the degree to 2 and 3 to see the differences.

The quadratic model (R-squared = 0.556) introduces squared terms for the predictors, capturing the non-linear relationships in the data. The quadratic term for "log_house_size" is positive and significant, indicating that house size has an accelerating impact on log_price. When the degree becomes 3, the R-squared increases to 0.562, suggesting a better fit to the data.

Model 3: Polynomial Regression (Degree 3)

Formula: log_price ~ bed + bath + log_house_size + nyc + I(bed**2) + I(bath**2) + I(log_house_size**2) + I(bed**3) + I(bath**3) + I(log_house_size**3)



```
Model Summary:
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.562
Model:                            OLS   Adj. R-squared:                  0.562
Method:                 Least Squares   F-statistic:                     1176.
Date:                Sun, 15 Dec 2024   Prob (F-statistic):               0.00
Time:                        08:54:49   Log-Likelihood:                -8126.9
No. Observations:                9161   AIC:                         1.628e+04
Df Residuals:                    9150   BIC:                         1.635e+04
Df Model:                          10
Covariance Type:            nonrobust
==========================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------
Intercept                 47.0440      3.635     12.943      0.000      39.919      54.169
bed                        0.0453      0.015      3.024      0.003       0.016       0.075
bath                       0.2894      0.015     19.666      0.000       0.261       0.318
log_house_size           -13.4470      1.373     -9.795      0.000     -16.138     -10.756
nyc                        0.6740      0.013     50.448      0.000       0.648       0.700
I(bed ** 2)               -0.0081      0.002     -5.218      0.000      -0.011      -0.005
I(bath ** 2)              -0.0093      0.002     -5.605      0.000      -0.013      -0.006
I(log_house_size ** 2)     1.6715      0.172      9.708      0.000       1.334       2.009
I(bed ** 3)                0.0002   3.82e-05      4.068      0.000    8.05e-05       0.000
I(bath ** 3)             8.72e-05    3.9e-05      2.238      0.025    1.08e-05       0.000
I(log_house_size ** 3)    -0.0658      0.007     -9.189      0.000      -0.080      -0.052
==============================================================================
Omnibus:                      295.144   Durbin-Watson:                   0.831
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              779.046
Skew:                          -0.087   Prob(JB):                    6.80e-170
Kurtosis:                       4.418   Cond. No.                     9.80e+05
==============================================================================
```

In this model, the R-squared value of is 0.562. We can see the intercept becomes very high, and NYC still maintains a significant impact on housing prices. Although the coefficients of most variables are significant, there are differences in the coefficients of the same variable at different degrees. For example, the coefficient of log_house_size becomes -13.4470, while the coefficient of log_house_size**2 is 1.6715, and the coefficient of log_house_size**3 is -0.0658.

This shows that when the house size is small, an increase in size may lead to diminishing marginal returns or even a decrease in house prices (such as the effects of poorer housing quality or crowding). When the house size exceeds a certain critical value, the positive effect of the square term dominates. The larger house size begins to reflect the value-added effect, and house prices rise as the house size increases. When the housing area further increases to a larger value, the negative effect of the cubic term appears, causing the growth of housing prices to slow down or even fall back. This may reflect diminishing marginal returns on oversized homes, or an excess of market demand. The same is true for other variables such as bed and bath. The coefficient of bath is 0.2894, while the coefficient of bath**2 is -0.0093. This shows that when the number of baths reaches a certain number, increasing the investment in bathrooms no longer brings obvious effects.

The model is statistically significant, emphasizing its non-linear influence on log_price. However, the higher-degree terms for other predictors have mixed significance, potentially reflecting overfitting to the data. Meanwhile, the condition number is large, 9.80e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In conclusion, all models explain a moderate proportion of the variance in prices. The polynomial model shows slightly better predictive accuracy but at the cost of increased complexity. Although model 3 better reflects the complexity and diversity of the data, we still think that model 2 can better predict the price due to the existence of multicollinearity and overfitting of the data. According to the study, the key drivers of price are house size, bathrooms, bedrooms, and NYC premium. The house size and, more importantly, especially location, being in NYC, are the key determinants of prices in the real estate market. The larger houses command higher prices as reflected by the positive coefficient of log_house_size. The number of bathrooms has a stronger impact on price compared to the bedrooms.

Q3. Classification model to predict property location

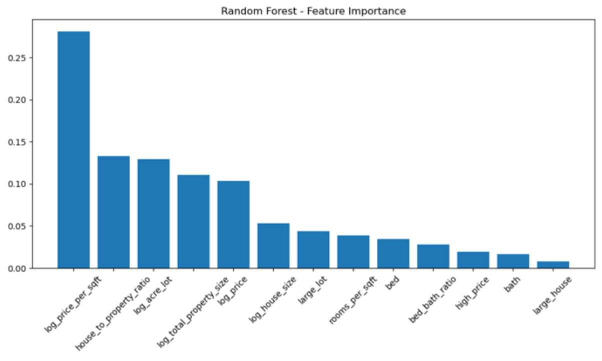
Question 3 explores the New York housing market, focusing on identifying key characteristics that distinguish NYC properties from those in other parts of the state. Using data from approximately 9,000 property listings, we develop machine learning models to understand and predict whether a property is located in NYC based on its features.

The methodology involves three different machine learning approaches: Random Forest, Logistic Regression, and Decision Tree models. To ensure the holistic evaluation, we split our data into training, validation, and test sets, and engineer additional features such as price per square foot and property size ratios to improve model performance. Moreover, to ensure the stability and generalization of our model, we perform 5-fold cross-validation on the training set.

First, the dataset is divided into three subsets to facilitate different stages of the modeling process. The training set, comprising 60% of the data, is used to train the models. A validation set, accounting for 20% of the data, is employed for hyperparameter tuning and intermediate evaluations. Finally, the test set, consisting of the remaining 20%, is reserved for assessing the final performance of the models.

Feature engineering plays a critical role in improving the predictive accuracy of the models. We divide these variables into different features. Basic Features include variables such as log_price, log_house_size, bed and bath. While these features provide foundational information, their predictive power is relatively limited. Property Features encompass metrics like property size and ratios, such as the house-to-property ratio. These features contribute significantly to the model's performance, capturing more nuanced aspects of the data. Engineered Features involve advanced metrics such as price per square foot, rooms per square foot and bed-bath ratios. These features dominate the feature importance rankings, offering the most discriminative power for the classification task.

The variable importance analysis demonstrates several key findings. Log_price_per_sqft emerges as the most influential variable, accounting for 27% of the overall importance. The house-to-property ratio is identified as the second most critical feature, contributing around 13% to the model's predictive strength. Additionally, the analysis highlights the marginal utility of basic features when compared to the engineered ones, emphasizing the value of complex and derived metrics in enhancing model performance.



Based on these features, we use three models (Random Forest, Logistic Regression, and Decision Tree) to develop the classification. To evaluate the performance of the model, we use metrics such as accuracy, precision, recall, and F1 score. Results of the test model with all features are shown below:

Model Performance Metrics (%)

	feature_sets	**model**	**accuracy**	**precision**	**recall**	**f1**
1	All Features	Random Forest	80.3	65.7	89.0	75.6
2	All Features	Logistic Regression	77.6	62.5	87.1	72.8
3	All Features	Decision Tree	76.8	60.5	93	73.3

Under all feature sets, the performance of the Random Forest model is superior to other models, achieving the highest accuracy and F1 score. Especially when all engineering features are included, the accuracy of the random forest model reaches 80.3%, and the F1 score is 75.6%. This indicates that the random forest model has a significant advantage in predicting whether a property is located in New York City.

In conclusion, our analysis reveals that non-geographical features can effectively predict the location of properties in NYC. Among the models tested, the Random Forest model, with its ability to handle feature interactions and robustness to outliers, emerged as the best choice. The model's high recall value for NYC properties indicates its effectiveness in identifying properties located within the city. While precision values were slightly lower, the overall balance between precision and recall, as reflected in the F1 score, makes it the most effective model for predictive tasks in this study. Logistic Regression provides a simpler but still effective alternative, with 77.6% accuracy. The Decision Tree model, while providing clear decision rules, shows lower accuracy and tends to overfit the training data.

Notably, price per square foot emerges as the most discriminative feature, underscoring the distinct pricing patterns of NYC properties compared to those outside the city.

Additionally, all models demonstrate higher recall than precision, indicating a stronger ability to correctly identify NYC properties, albeit with a higher likelihood of false positives.

Codes:

1. Import libraries/modules

```
######## To read and manupulate 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
from sklearn.model_selection import train_test_split
#split data to training and for validation
######## For plotting
import seaborn as sns
import matplotlib.pyplot as plt
# this is needed to see the plots inside the notebook
%pylab inline
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.metrics import roc_auc_score,auc, roc_curve
```

2. Read Data

```
file_path='data/realtor-data-ny.csv'
data = pd.read_csv(file_path, low_memory=False)
```

#Q1

3. Data Cleaning

```
# Check the data types of all columns
print("Column data types before conversion:")
print(data.dtypes)

# Define columns to clean
critical_columns = ['price', 'bed', 'bath', 'house_size']
additional_columns = ['acre_lot', 'prev_sold_date']

# Convert critical columns to numeric
for col in critical_columns:
    data[col] = pd.to_numeric(data[col], errors='coerce')

# Convert additional columns if needed
data['acre_lot'] = pd.to_numeric(data['acre_lot'], errors='coerce')
data['prev_sold_date'] = pd.to_datetime(data['prev_sold_date'], errors='coerce', format='%d/%m/%Y')
```

```
# Remove rows with missing values in critical columns
cleaned_data = data.dropna(subset=critical_columns)

# Display summary after cleaning
print("Summary of dataset after fixing mixed types:")
print(cleaned_data.info())

# check duplicates
duplicates = cleaned_data[cleaned_data.duplicated()]

# count the numbers of duplicate rows
duplicate_count = cleaned_data.duplicated().sum()
print(f"number of duplicates: {duplicate_count}")

# drop duplicates
clean_data = cleaned_data.drop_duplicates()
clean_data.reset_index(drop= True, inplace=True)

# Make data more readable
final_data = clean_data[clean_data['price'] > 0].copy() # Filter out non-positive prices
final_data['log_price'] = np.log(clean_data['price'])
final_data['log_house_size'] = np.log(clean_data['house_size'])
```

4. Summary Statistics

```
# We start exploring the data
# First, we get the basic information
final_data.info()

# We can get some general info about the data (columns)
final_data.describe()

# We can start looking at the first lines of the file
sorted_final_data = final_data.sort_values(by='price', ascending=True) #sort data by price
sorted_final_data.head()

# We can look also at the last part of the data
sorted_final_data.tail()

# Calculate the median and mode of 'price', 'bed', 'bath', 'house_size'
for col in critical_columns:
    median = final_data[col].median()
    mode = final_data[col].mode()
    print(median, mode)

# Check the details of 'status', 'city', 'state' and 'nyc'
unique_values = final_data[['status', 'city', 'state', 'nyc']].apply(pd.Series.unique)
print(unique_values)

# Check how many different cities in New York state
unique_counts = final_data['city'].value_counts()
print(unique_counts)

# Check how many houses in nyc
```

```
nyc_yes_count = (final_data['nyc'] == 'Yes').sum()
print(nyc_yes_count)
nyc_no_count = (final_data['nyc'] == 'No').sum()
print(nyc_no_count)

# Check the critical column in nyc
house_nyc_yes=final_data[final_data['nyc'] == 'Yes']
house_nyc_yes.describe()

# check the critical column outside nyc
house_nyc_no=final_data[clean_data['nyc'] == 'No']
house_nyc_no.describe()

# Comparion between house information based on nyc
nyc_group = final_data.groupby('nyc')[['price', 'bed', 'bath', 'house_size']].agg(['mean', 'median', 'std'])
print(nyc_group)

# Comparion between house information based on city
city_group = final_data.groupby('city')[['price', 'bed', 'bath', 'house_size']].agg(['mean', 'median', 'std'])
city_group.describe()
sorted_city_group = city_group.sort_values(by=('price', 'mean'), ascending=True)
sorted_city_group.describe()

# Check the city with lowerest house price
sorted_city_group.head()

# Check the city with highest house price
sorted_city_group.tail()
```

5. Data visualization

```
# 1. Distribution of Price and House Size
plt.figure(figsize=(14, 6))

# Price Distribution
plt.subplot(1, 2, 1)
sns.histplot(final_data['log_price'], bins=50, kde=True, color='blue')
plt.title('Distribution of Price')
plt.xlabel('Price (USD)')
plt.ylabel('Frequency')

# 2. House Size Distribution
plt.subplot(1, 2, 2)
sns.histplot(final_data['log_house_size'], bins=50, kde=True, color='green')
plt.title('Distribution of House Size')
plt.xlabel('House Size (sq ft)')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# 3. Relationship Between House Size and Price
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(data=final_data, x='log_house_size', y='log_price', hue='nyc', alpha=0.7,
palette='coolwarm')
plt.title('House Size vs Price (NYC vs Non-NYC)')
plt.xlabel('House Size (sq ft)')
plt.ylabel('Price (USD)')
plt.legend(title='NYC')
plt.show()

# Group data by number of bedrooms and calculate the average price
avg_price_bedrooms = final_data.groupby('bed')['price'].mean().reset_index()

# 4. Plot Average Price by Number of Bedrooms
plt.figure(figsize=(10,6))

sns.barplot(x='bed', y='price', data=avg_price_bedrooms, hue='bed', palette='coolwarm', legend=False)

# Adjust y-axis to USD in Millions
plt.gca().get_yaxis().set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x / 1e6:.1f}M'))

plt.title('Average Price by Number of Bedrooms')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Average Price (USD in Millions)')
plt.show()

# 5. Calculate average price by city
avg_price_city = final_data.groupby('city')['price'].mean().sort_values(ascending=False).reset_index()

# Plot average prices by city
plt.figure(figsize=(12, 8))
sns.barplot(x='price', y='city', data=avg_price_city.head(20), hue='city', palette='viridis', legend=False)  #
Top 20 cities by price
plt.title('Average Price by City (Top 20)')
plt.xlabel('Average Price (USD)')
plt.ylabel('City')
plt.gca().get_xaxis().set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x / 1e6:.1f}M'))
plt.show()

# Calculate log price per square foot
final_data['log_price_per_sqft'] = final_data['log_price'] / final_data['log_house_size']

# Plot distribution of log price per square foot
plt.figure(figsize=(10, 6))
sns.histplot(final_data['log_price_per_sqft'], kde=True, bins=30, color='purple')
plt.title('Distribution of Log Price Per Square Foot')
plt.xlabel('Log Price Per Square Foot')
plt.ylabel('Frequency')
plt.show()

# 6. Exclude outliers based on price using the IQR method
Q1 = final_data['price'].quantile(0.25)
Q3 = final_data['price'].quantile(0.75)
IQR = Q3 - Q1
filtered_data = final_data[(final_data['price'] >= Q1 - 1.5 * IQR) & (final_data['price'] <= Q3 + 1.5 * IQR)]
```

```
# Filter out extreme lot sizes for clarity
filtered_data = filtered_data[filtered_data['acre_lot'] < 50]

# Scatter plot of lot size vs price, separated by NYC and Non-NYC
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x=filtered_data['acre_lot'],
    y=filtered_data['price'],
    hue=filtered_data['nyc'],
    style=filtered_data['nyc'],
    alpha=0.6
)
plt.title('Lot Size vs Price (NYC vs Non-NYC)')
plt.xlabel('Lot Size (acres)')
plt.ylabel('Price (USD)')
plt.legend(title='NYC')
plt.show()

# 7. Calculate median lot size by city
median_lot_size_city                                                                =
final_data.groupby('city')['acre_lot'].median().sort_values(ascending=False).reset_index()

# Bar plot of median lot size by city
plt.figure(figsize=(12, 8))
sns.barplot(x='acre_lot', y='city', hue='city', data=median_lot_size_city.head(20), palette='magma')
plt.title('Median Lot Size by City')
plt.xlabel('Median Lot Size (acres)')
plt.ylabel('City')
plt.show()

# 8. Ensure the 'bath' column is treated as integers for proper ordering
final_data['bath'] = final_data['bath'].astype(int)

# Group data by bath and NYC status, and calculate the mean log_price
line_data = final_data.groupby(['bath', 'nyc'])['log_price'].mean().reset_index()

# Line plot
plt.figure(figsize=(12, 6))
sns.lineplot(
    x='bath',
    y='log_price',
    hue='nyc',
    data=line_data,
    palette={'Yes': 'blue', 'No': 'orange'},
    marker='o'
)

# Titles and labels
plt.title('Log Price by Number of Bathrooms: NYC vs Non-NYC')
plt.xlabel('Number of Bathrooms')
plt.ylabel('Average Log Price')
```

```
# Adjust x-axis to display integers only
plt.xticks(ticks=range(final_data['bath'].min(), final_data['bath'].max() + 1), rotation=45)

plt.legend(title='NYC')
plt.show()

# 9. Correlation Heatmap
corr_matrix = final_data[['log_price', 'house_size', 'log_house_size', 'bed', 'bath', 'acre_lot']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

#Q2

```
# look at this, acre_lot has very low correlation to log_price ---- remove it out!
# so now we have 4 independent variables: bed, bath, log_house_size, nyc
# bed negative not make sense---- not works even high rsquare and all variables significant ----> try interaction term

# change 'nyc' to numeric, yes=1 and no=0
final_data["nyc"] = final_data["nyc"].astype(str)
final_data["nyc"] = final_data["nyc"].replace({"Yes": 1, "No": 0}).astype(int)

# Define features and target variable'
X = final_data[['bed', 'bath', 'log_house_size', 'nyc']]
y = final_data['log_price']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Add constant for OLS model
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Train the OLS Regression Model
model = sm.OLS(y_train, X_train_const).fit()

# Print OLS Summary
print("OLS Regression Summary:")
print(model.summary())

# Predict using the model
y_pred = model.predict(X_test_const)

# Evaluate the Model
rmse = mean_squared_error(y_test, y_pred, squared=False)
mae = mean_absolute_error(y_test, y_pred)

print("\nLinear Regression Metrics:")
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
```

```
# Predict using the model
y_pred = model.predict(X_test_const)
print(y_pred)
```

#Model 1 linear regression

```
def get_formula():
    # Base formula
    model_formula = 'log_price ~ bed + bath + log_house_size + nyc'
    return model_formula

model_formula = get_formula()
model = smf.ols(formula=model_formula, data=final_data).fit()

# Print the formula and summary
print("Model Formula:")
print(model_formula)
print("\nModel Summary:")
print(model.summary())

# #bath has high correlation with others so make it interaction term!
#creating regression model with interaction terms
def get_formula():
    # Base formula
    model_formula = 'log_price ~ bed + bath + log_house_size + nyc'

    # Add interaction terms based on correlation analysis
    interaction_terms = [
        'bed:bath',
        'bath:log_house_size',
        'bed:log_house_size'
    ]
    for term in interaction_terms:
        model_formula += f' + {term}'

    return model_formula

# Generate formula and fit the model
model_formula = get_formula()
model = smf.ols(formula=model_formula, data=final_data).fit()

# Print the formula and summary
print("Model Formula:")
print(model_formula)
print("\nModel Summary:")
print(model.summary())
```

#**Model 2: Interaction Terms**

```
def get_formula():
    # Base formula
    model_formula = 'log_price ~ bed + bath + log_house_size + nyc'

    # Add interaction terms based on correlation analysis
    interaction_terms = [
        'bed:log_house_size'
    ]
    for term in interaction_terms:
        model_formula += f' + {term}'

    return model_formula

# Generate formula and fit the model
model_formula = get_formula()
model = smf.ols(formula=model_formula, data=final_data).fit()

# Print the formula and summary
print("Model Formula:")
print(model_formula)
print("\nModel Summary:")
print(model.summary())

# creating degree 2 polynomial regression
def get_formula(degree):
    model_formula='log_price ~ bed + bath + log_house_size + nyc'
    for i in range(1,degree):
        model_formula+= '+I(bed**'+str(i+1)+')'+'+I(bath**'+str(i+1)+')'+'+I(log_house_size**'+str(i+1)+')'
    return model_formula

model_formula = get_formula(degree=2)

model = smf.ols(formula=model_formula, data=final_data).fit()

# Print formula and summary
print("Model Formula:")
print(model_formula)
print("\nModel Summary:")
print(model.summary())

# creating degree 3 polynomial regression
```

# Model 3 polynomial regression (degree=3)

```
model_formula = get_formula(degree=3)

model = smf.ols(formula=model_formula, data=final_data).fit()

# Print formula and summary
print("Model Formula:")
print(model_formula)
print("\nModel Summary:")
print(model.summary())

# Generate predictions and residuals
final_data['Prediction'] = model.predict(final_data)
final_data['Residual'] = final_data['log_price'] - final_data['Prediction']

# Calculate MAE and MSE
mae = mean_absolute_error(final_data['log_price'], final_data['Prediction'])
```

```
mse = mean_squared_error(final_data['log_price'], final_data['Prediction'])

# Print the formula, summary, and error metrics
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")

# Plot Actual vs Predicted values
plt.figure(figsize=(10, 6))
plt.scatter(final_data['log_price'], final_data['Prediction'], alpha=0.7, label='Predicted vs Actual')
plt.plot([final_data['log_price'].min(), final_data['log_price'].max()],
        [final_data['log_price'].min(), final_data['log_price'].max()],
        '--r', linewidth=2, label='Perfect Prediction')
plt.title("Actual vs Predicted Values")
plt.xlabel("Actual log_price")
plt.ylabel("Predicted log_price")
plt.legend()
plt.show()

# Plot residuals
plt.figure(figsize=(10, 6))
plt.scatter(final_data['Prediction'], final_data['Residual'], alpha=0.7, label='Residuals')
plt.axhline(y=0, color='r', linestyle='--', label='Zero Residual Line')
plt.title("Residual Plot")
plt.xlabel("Predicted log_price")
plt.ylabel("Residuals")
plt.legend()
plt.show()
```

#Q3

```
from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (classification_report, confusion_matrix, roc_curve, auc,
                        make_scorer, accuracy_score, precision_score, recall_score, f1_score)

# Set random seeds for reproducibility
import random
np.random.seed(42)
random.seed(42)

# 1. Data Loading and Initial Preprocessing
   def load_and_preprocess_data(file_path):
      """Load and preprocess the dataset"""
      data = pd.read_csv(file_path, low_memory=False)

      # Convert critical columns to numeric
      critical_columns = ['price', 'bed', 'bath', 'house_size', 'acre_lot']
      for col in critical_columns:
         data[col] = pd.to_numeric(data[col], errors='coerce')
```

```
        # Remove rows with missing values in critical columns
        data = data.dropna(subset=critical_columns)

        # Drop duplicates
        data = data.drop_duplicates()

        # Create basic log transformations
        data['log_price'] = np.log(data['price'])
        data['log_house_size'] = np.log(data['house_size'])
        data['log_acre_lot'] = np.log(data['acre_lot'] + 1)

        return data


# 2. Feature Engineering
def create_features(df):
    """Create engineered features"""
    features = df.copy()

    # Price per square foot
    features['price_per_sqft'] = features['price'] / features['house_size']
    features['log_price_per_sqft'] = np.log(features['price_per_sqft'])

    # Total property size
    features['total_property_size'] = features['house_size'] + (features['acre_lot'] * 43560)
    features['log_total_property_size'] = np.log(features['total_property_size'])
    features['house_to_property_ratio'] = features['house_size'] / features['total_property_size']

    # Room-related features
    features['rooms_per_sqft'] = (features['bed'] + features['bath']) / features['house_size']
    features['bed_bath_ratio'] = features['bed'] / features['bath']

    # Binary indicators
    features['high_price'] = (features['price'] > features['price'].median()).astype(int)
    features['large_house'] = (features['house_size'] > features['house_size'].median()).astype(int)
    features['large_lot'] = (features['acre_lot'] > features['acre_lot'].median()).astype(int)

    return features


# 3. Model Training and Evaluation
def train_and_evaluate_models(X_train, X_val, y_train, y_val, feature_set_name):
    """Train and evaluate multiple models with detailed metrics"""
    models = {
        'Logistic Regression': LogisticRegression(random_state=42),
        'Decision Tree': DecisionTreeClassifier(random_state=42),
        'Random Forest': RandomForestClassifier(random_state=42)
    }

    param_grids = {
        'Logistic Regression': {
            'C': [0.1, 1.0, 10.0],
            'class_weight': [None, 'balanced']
        },
        'Decision Tree': {
            'max_depth': [3, 5, 7, None],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4],
            'class_weight': [None, 'balanced']
        },
        'Random Forest': {
            'n_estimators': [100, 200],
            'max_depth': [5, 10, None],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4],
            'class_weight': [None, 'balanced']
        }
    }

    results = {}

    for model_name, model in models.items():
        print(f"\n{'-'*50}")
        print(f"Training {model_name} with {feature_set_name} features")
        print(f"{'-'*50}")

        # GridSearchCV
        grid_search = GridSearchCV(
            estimator=model,
            param_grid=param_grids[model_name],
            cv=5,
            scoring=['accuracy', 'precision', 'recall', 'f1'],
            refit='f1',
            n_jobs=-1,
            verbose=1,
            return_train_score=True
        )

        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_

        # Calculate metrics
        y_train_pred = best_model.predict(X_train)
        y_val_pred = best_model.predict(X_val)

        train_metrics = {
            'accuracy': accuracy_score(y_train, y_train_pred),
            'precision': precision_score(y_train, y_train_pred),
            'recall': recall_score(y_train, y_train_pred),
            'f1': f1_score(y_train, y_train_pred)
        }

        val_metrics = {
            'accuracy': accuracy_score(y_val, y_val_pred),
            'precision': precision_score(y_val, y_val_pred),
            'recall': recall_score(y_val, y_val_pred),
            'f1': f1_score(y_val, y_val_pred)
        }

        # Print results
        print("\nBest Parameters:", grid_search.best_params_)
        print("\nTraining Metrics:")
        for metric, value in train_metrics.items():
            print(f"{metric}: {value:.3f}")

        print("\nValidation Metrics:")
        for metric, value in val_metrics.items():
            print(f"{metric}: {value:.3f}")

        # Plot confusion matrices
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

        sns.heatmap(confusion_matrix(y_train, y_train_pred),
                    annot=True, fmt='d', ax=ax1, cmap='Blues')
        ax1.set_title(f'{model_name} - Training Confusion Matrix')
        ax1.set_ylabel('True Label')
        ax1.set_xlabel('Predicted Label')

        sns.heatmap(confusion_matrix(y_val, y_val_pred),
                    annot=True, fmt='d', ax=ax2, cmap='Blues')
        ax2.set_title(f'{model_name} - Validation Confusion Matrix')
        ax2.set_ylabel('True Label')
        ax2.set_xlabel('Predicted Label')

        plt.tight_layout()
        plt.show()

        # Feature importance for tree-based models
        if model_name in ['Random Forest', 'Decision Tree']:
            plt.figure(figsize=(10, 6))
            importances = pd.DataFrame({
                'feature': features,
                'importance': best_model.feature_importances_
            }).sort_values('importance', ascending=False)

            plt.bar(range(len(importances)), importances['importance'])
            plt.xticks(range(len(importances)), importances['feature'], rotation=45)
            plt.title(f'{model_name} - Feature Importance')
            plt.tight_layout()
            plt.show()

        results[model_name] = {
            'model': best_model,
            'train_metrics': train_metrics,
            'val_metrics': val_metrics,
            'best_params': grid_search.best_params_
        }

    return results
```

```
# Main execution
# Load and preprocess data
file_path = 'data/realtor-data-ny.csv'
data = load_and_preprocess_data(file_path)
features_df = create_features(data)

# Define feature sets
feature_sets = {
    'basic': ['bed', 'bath', 'log_house_size', 'log_price'],
    'property': ['bed', 'bath', 'log_house_size', 'log_price', 'log_acre_lot',
                'house_to_property_ratio', 'log_total_property_size'],
    'all': ['bed', 'bath', 'log_house_size', 'log_price', 'log_acre_lot',
            'house_to_property_ratio', 'log_total_property_size',
            'log_price_per_sqft', 'rooms_per_sqft', 'bed_bath_ratio',
            'high_price', 'large_house', 'large_lot']
}

# Prepare target variable and split data
X = features_df[feature_sets['all']]
y = (features_df['nyc'] == 'Yes').astype(int)

# First split: separate test set
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Second split: separate train and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.2, random_state=42
)

# Train and evaluate models for each feature set
results = {}
scaler = StandardScaler()

for set_name, features in feature_sets.items():
    print(f"\n{'='*80}")
    print(f"Evaluating feature set: {set_name}")
    print(f"{'='*80}")

    # Get features for current set
    X_train_current = X_train[features]
    X_val_current = X_val[features]
    X_test_current = X_test[features]

    # Scale features
    X_train_scaled = scaler.fit_transform(X_train_current)
    X_val_scaled = scaler.transform(X_val_current)
    X_test_scaled = scaler.transform(X_test_current)

    # Train and evaluate on train/validation sets
    set_results = train_and_evaluate_models(
        X_train_scaled, X_val_scaled,
        y_train, y_val,
        set_name
    )

    # Evaluate best models on test set
    print("\nTest Set Evaluation:")
    print("-" * 50)
    for model_name, model_results in set_results.items():
        best_model = model_results['model']
        y_test_pred = best_model.predict(X_test_scaled)

        # Calculate test metrics
        test_metrics = {
            'accuracy': accuracy_score(y_test, y_test_pred),
            'precision': precision_score(y_test, y_test_pred),
            'recall': recall_score(y_test, y_test_pred),
            'f1': f1_score(y_test, y_test_pred)
        }

        print(f"\n{model_name} Test Results:")
        print(classification_report(y_test, y_test_pred))

        # Plot test confusion matrix
        plt.figure(figsize=(8, 6))
        cm_test = confusion_matrix(y_test, y_test_pred)
        sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues')
        plt.title(f'{model_name} - Test Set Confusion Matrix')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.show()

        # Store test results
        model_results['test_metrics'] = test_metrics

    results[set_name] = set_results

# Print final comparison including test results
print("\nFinal Model Comparison (Including Test Results):")
for set_name, models in results.items():
    print(f"\nFeature set: {set_name}")
    print("-" * 50)
    for model_name, metrics in models.items():
        print(f"\n{model_name}:")
        print(f"Best parameters: {metrics['best_params']}")
        print("Training metrics:", metrics['train_metrics'])
        print("Validation metrics:", metrics['val_metrics'])
        print("Test metrics:", metrics['test_metrics'])
```