College of Professional Studies
Northeastern University San Jose

MPS Analytics

Course: ALY6020

Assignment:

Module 4 – Project

Submitted on:

October 20, 2023

Submitted to:

Professor: Ahmadi Behzad

Submitted by:

Heejae Roh

Introduction

In this analysis, I will deploy linear regression, decision tree, random forest, and XGBoost models to target the price of real estate and whether it is overvalued. Using linear regression, I want to predict building_value, a numerical variable, and using other models, I will decide which model is the best to predict binary variable 'sale_price_compared_to_value' overvalued or not. In the process of analysis, I will compare decision tree, random forest, and XGBoost and consider how to visualize each model. Based on this, I will derive insights into what model the company can use to analyze the house market.

Abstraction

1. Use proper data cleansing techniques to ensure that you have the highest quality data to model this problem. Detail your process and discuss the decisions you made to clean the data.
The dataset underwent a series of changes, including column format adjustments, removal of missing values, and exclusion of specific columns either due to redundancy or lack of relevance for analysis. Notably, certain columns with unique or automatically assigned values were omitted, and some decisions were made based on the characteristics of specific columns.

1. The sale_date column was reformatted to a date type. Subsequently, this column was split into separate year, month, and day columns, and the original sale_date column was not considered in the analysis.
2. The suite/Condo # column had entirely missing values and was therefore removed. For other columns, where missing values ranged between 1 and 3 for attributes like bedrooms, property address, property city, full bath, foundation type, and finished, they were excluded. Notably, 108 half baths represented 0.47% of the total entries.
3. Columns named unnamed:_0 and parcel_id were disregarded since they contain automatically generated data.
4. The legal_reference column, containing 22,452 distinct values, was omitted as it seemed challenging to utilize either as a categorical or numerical variable.
5. All entries in the state column had a single unique value, TN (Tennessee), resulting in its exclusion.
6. The neighborhood column was identified as potentially representing zip codes and was subsequently decided to be left out from the linear regression analysis.

2. Build a linear regression model to accurately predict housing prices and determine what is driving those prices.

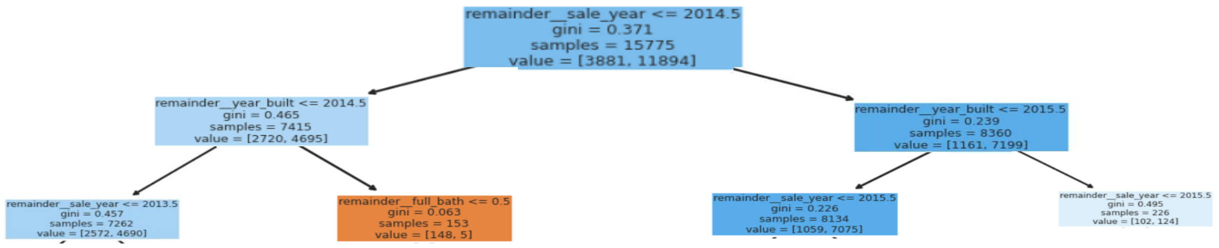
```
                    After Normalization OLS Regression Results
==============================================================================
Dep. Variable:         building_value   R-squared:                       0.735
Model:                            OLS   Adj. R-squared:                  0.735
Method:                 Least Squares   F-statistic:                     8313.
Date:                Thu, 19 Oct 2023   Prob (F-statistic):               0.00
Time:                        21:33:00   Log-Likelihood:            -2.3285e+05
No. Observations:               18028   AIC:                         4.657e+05
Df Residuals:                   18021   BIC:                         4.658e+05
Df Model:                           6
Covariance Type:            nonrobust
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const           1.625e+04   2975.826      5.461      0.000    1.04e+04    2.21e+04
acreage        -5.298e+05   2.32e+04    -22.814      0.000   -5.75e+05   -4.84e+05
land_value       4.27e+05   1.82e+04     23.448      0.000    3.91e+05    4.63e+05
finished_area   2.864e+06   2.24e+04    127.883      0.000    2.82e+06    2.91e+06
bedrooms       -2.594e+05   1.23e+04    -21.039      0.000   -2.84e+05   -2.35e+05
half_bath        9.45e+04   4974.609     18.997      0.000    8.48e+04    1.04e+05
sold_as_vacant  6.133e+04   9807.383      6.253      0.000    4.21e+04    8.05e+04
```

```
==============================================================================
Omnibus:                    25315.703   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         43579468.002
Skew:                           7.564   Prob(JB):                         0.00
Kurtosis:                     243.389   Cond. No.                         37.7
==============================================================================
```

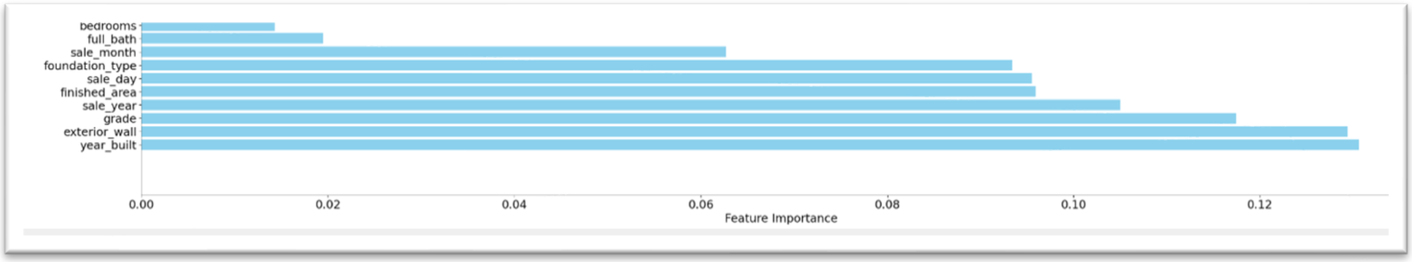
Most negative influence: acreage > bedrooms
Most positive influence: finished_area > land_value
Dependent Variable (Target): The model is trying to predict the building_value.

1. R-squared: The R-squared value is 0.735, which means that approximately 73.5% of the variance in the building value is explained by the model. This indicates a reasonably good fit.
2. Adjusted R-squared: This is very close to the R-squared value, suggesting that most of the variables included in the model are relevant predictors.
3. acreage: For each additional acre, the building value decreases by 529,800 units, holding all else constant.
4. land_value: For each unit increase in land value, the building value increases by 427,000 units, holding all other factors constant.
5. finished_area: For each unit increase in the finished area, the building value increases by 2,864,000 units.
6. bedrooms: For each additional bedroom, the building value decreases by 259,400 units.
7. half_bath: For each additional half bath, the building value increases by 94,500 units.
8. sold_as_vacant: If the property is sold as vacant, the building value increases by 61,330 units compared to when it's not sold as vacant.

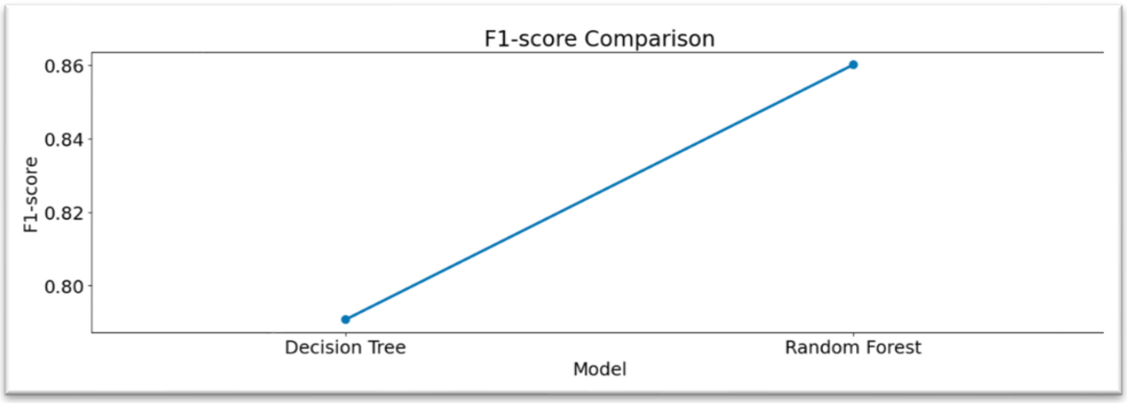
3, 4. Build a decision tree model and compare the results with the results of the previous model. Build a Random Forest model and compare the results with the results of the previous models.



This is the visualization that can be seen when the depth of the decision tree is adjusted to 5. In the first layer, sale_year is divided based on 2014.5 and gini is 0.371. The second node was divided into those lower and higher than 2014.5 and those lower and higher than 2015.5 using year built. In Layer 3, the depth of the decision tree was added using sale year, full_bath, and sale year.

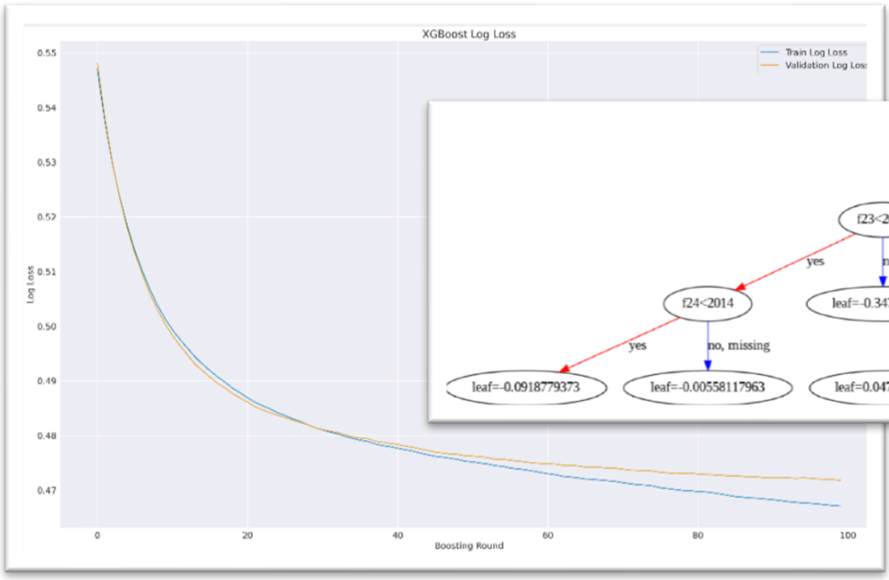


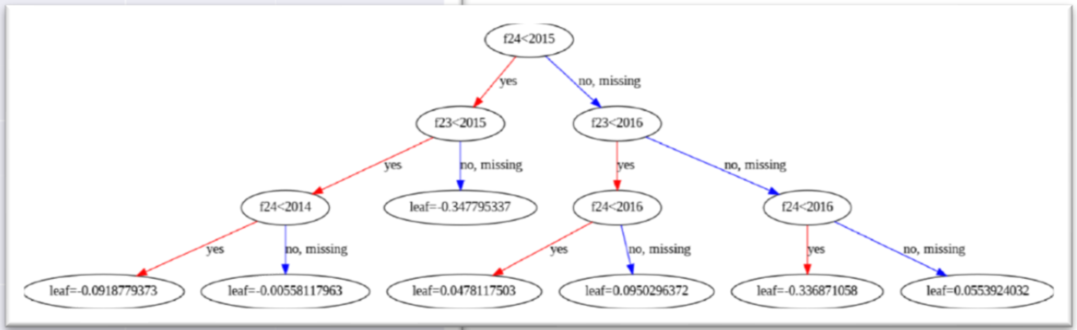
I compared the importance of features in the decision tree. Using the most important ten variables which are 'exterior_wall', 'year_built', 'grade', 'sale_year', 'finished_area', 'foundation_type', 'sale_day', 'sale_month', 'full_bath', 'bedrooms', I decided to predict 'compared_to_value' as Over or Under.



The F1-scores of the two final models using the same features are 79.09% and 86.01%, respectively. You can see that Random Forest made better predictions than Decision tree.

5. Build a Gradient Boost model and compare the results with the results of the previous models.



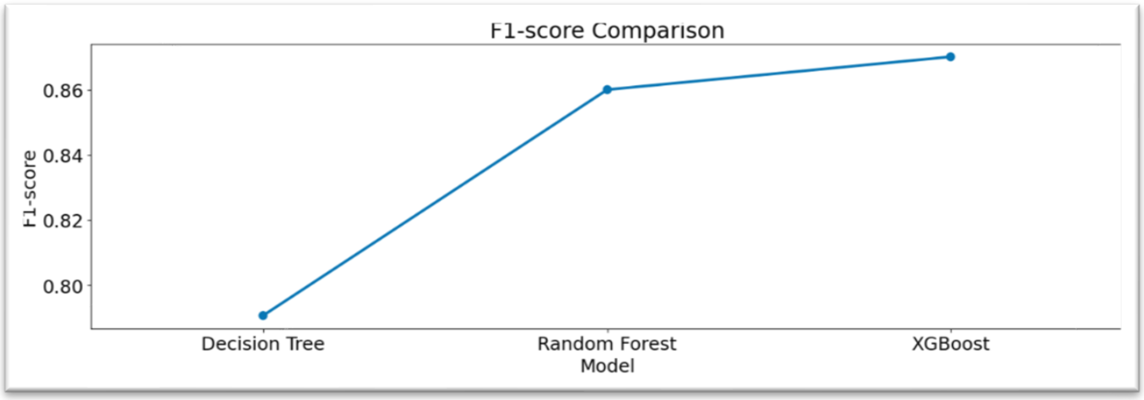


The line graph depicts the XGBoost model's log loss across boosting rounds. After the 100th round, the model's log loss drops, but for validation data, it rises post the 40th round. While initially set to 100 rounds, based on these trends, it might be more optimal to cap rounds at 40, 50, or 60. If the change isn't significant, keeping it at 100 remains an option.

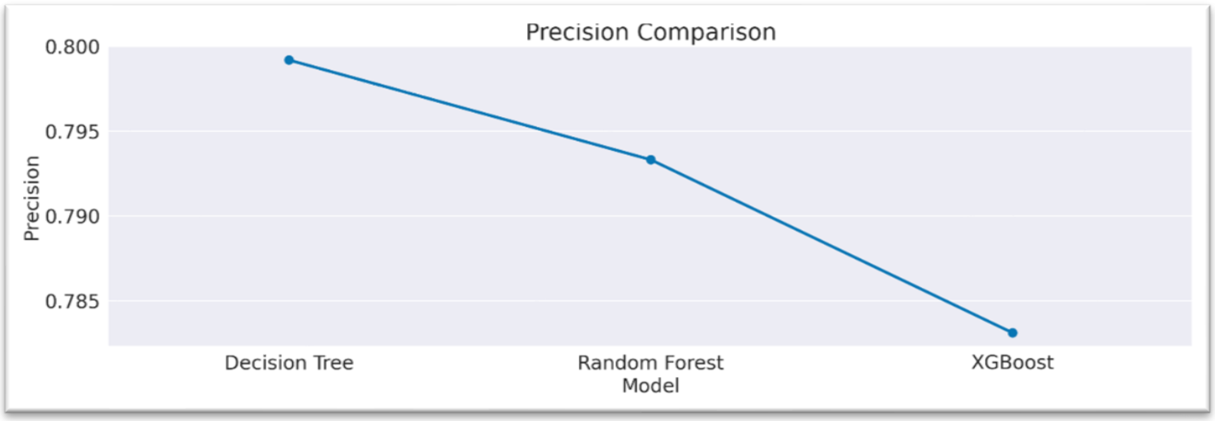
	Default Feature Name	Original Feature Name
0	f23	year_built
1	f24	sale_year
2	f25	finished_area
3	f27	sale_month
4	f26	sale_day

The table right above provides a breakdown of features according to their importance. You'll notice that 'f24' corresponds to 'sale_year' in the initial layer. By the time we get to the second layer, 'year_built' emerges as the most crucial factor.

6. Use multiple benchmarking metrics to compare and contrast the three models.



Given that our target variable, 'Sale Price Compared To Value', represents imbalanced data, the F1-score provides some insightful observations. As we progress through different models, from the decision tree to the random forest and finally to XGBoost, there's a noticeable uptick in the F1-score. This improvement is consistently mirrored in both accuracy and recall metrics.



As the model becomes more sophisticated, a decline in precision indicates an increased likelihood of predicting values as '1' when they may not be. However, the certainty of a predicted '1' truly being '1' diminishes. This phenomenon can be seen as a form of overfitting where the model leans towards classifying more data as 'over', but at the cost of reduced accuracy. While adjusting the features might address this issue, there's an inherent trade-off between boosting accuracy and the overall F-1 score. Given this trade-off, we've decided to conclude our analysis at this point.

7. Based on your findings, provide evidence of which model you believe the real estate company should use.

In linear regression to forecast building value, a couple of key influences are identified. On the negative side, 'acreage' has a stronger impact than 'bedrooms'. Conversely, for positive influence, 'finished_area' stands out more than 'land_value'. While it's crucial to view this from both angles, given that certain assumptions aren't met, a deeper discussion is required about the relationship between 'acreage' and 'finished_area'. Consulting a domain expert on this matter and pondering the exclusion of the feature might be necessary steps moving forward.

From the company's vantage point, the XGBoost model appears most suitable, given its ability to predict 'over' and 'under' values in a balanced manner. Yet, when it comes to precision, if the company's decision-making leans primarily towards the 'over' category, there's merit in exploring methods to enhance precision. This could involve further data processing or incorporating additional features. Fundamentally, it's important to note that this dataset is imbalanced, with a predominant skew towards 'over' values. Such an aspect should be factored into any analysis or decision-making process.

Dataset Understanding

I used building value to measure price using linear regression. In decision tree, random forest, and XGBoost, ‘Sale Price Compared To Value’, which represents 'evaluation of the sale price in comparison to its value', was used as a taget variable. There are 26 columns and 22,651 entries. If you look at the sale dates column, you can see that the data was created after 2016. Sale dates are distributed from 2013 to 2016.

Description of the variables/features in the dataset.

#	**column name**	Description
1	**Unnamed:0**	id number attributed to a buyer
2	**Parcel ID**	code attributed to a land
3	**Land Use**	shows the different uses of land
4	**Property Address**	address of land sold
5	**Suite/ Condo #**	city where the property is located (e.g., NASHVILLE).
6	**Property City**	location of land
7	**Sale Date**	date when the land was sold
8	**Legal Reference**	citation is the practice of crediting and referring to authoritative documents
9	**Sold As Vacant**	whether the property was sold as vacant or not
10	**Multiple Parcels Involved in Sale**	the sale involved multiple parcels.
11	**City**	city associated with the owner's address
12	**State**	state associated with the owner's address
13	**Acreage**	the size of an area of land in acres
14	**Tax District**	tax district to which the property belongs
15	**Neighborhood**	4 digits excluding the first part of the zip code
16	**Land Value**	the worth of the land
17	**Building Value**	worth of a building
18	**Finished Area**	finished area of the property in square feet
19	**Foundation Type**	type of foundation the property has (e.g., PT BSMT, SLAB)
20	**Year Built**	year the building was built
21	**Exterior Wall**	material of the property's exterior wall (e.g., BRICK, BRICK/FRAME)
22	**Grade**	grade or rating for the property (e.g., C, B)
23	**Bedrooms**	The number of bedrooms in the property
24	**Full Bath**	a bathroom that includes a shower, a bathtub, a sink, and a toilet.
25	**Half Bath**	a half bathroom only contains a sink and a toilet
26	**Sale Price Compared To Value**	evaluation of the sale price in comparison to its value (e.g., Over, Under)

Headtail of Dataset 1

Unnamed: 0	Parcel ID	Land Use	Property Address	Suite/ Condo #	Property City	Sale Date	Legal Reference	Sold As Vacant
1	105 11 0 080.00	SINGLE FAMILY	1802 STEWART PL	NaN	NASHVILLE	1/11/2013	20130118-0006337	No
2	118 03 0 130.00	SINGLE FAMILY	2761 ROSEDALE PL	NaN	NASHVILLE	1/18/2013	20130124-0008033	No
3	119 01 0 479.00	SINGLE FAMILY	224 PEACHTREE ST	NaN	NASHVILLE	1/18/2013	20130128-0008863	No
56607	176 09 0 003.00	SINGLE FAMILY	4964 HICKORY WOODS E	NaN	ANTIOCH	10/28/2016	20161031-0114817	No
56614	082 05 0 040.00	SINGLE FAMILY	1625 5TH AVE N	NaN	NASHVILLE	10/28/2016	20161102-0115988	No
56615	082 05 0 058.00	SINGLE FAMILY	1614 5TH AVE N	NaN	NASHVILLE	10/26/2016	20161101-0115366	No

Headtail of Dataset 2

…	Building Value	Finished Area	Foundation Type	Year Built	Exterior Wall	Grade	Bedrooms	Full Bath	Half Bath	Sale Price Compared To Value
…	134400	1149	PT BSMT	1941	BRICK	C	2	1	0	Over
…	157800	2090.82495	SLAB	2000	BRICK/FRAME	C	3	2	1	Over
…	243700	2145.60001	FULL BSMT	1948	BRICK/FRAME	B	4	2	0	Under
…	159300	3117	SLAB	1995	BRICK/FRAME	C	3	3	0	Over
…	204100	1637	CRAWL	2004	FRAME	B	3	2	1	Over
…	295900	2478	CRAWL	2005	FRAME	B	4	3	1	Over

Exploratory Data Analysis

Descriptive Analysis of Dataset 1

	acreage	land_value	building_value	finished_area	year_built	bedrooms	full_bath	half_bath
count	22651	2.27E+04	2.27E+04	22650	22651	22648	22650	22543
mean	0.454705	7.01E+04	1.72E+05	1915.37715	1961.94768	3.10491	1.887285	0.270239
std	0.611818	1.03E+05	1.90E+05	1079.09452	25.843908	0.829287	0.95122	0.480186
min	0.04	9.00E+02	1.40E+03	450	1832	0	0	0
25%	0.2	2.20E+04	8.55E+04	1250	1947	3	1	0
50%	0.28	3.00E+04	1.19E+05	1645.825	1959	3	2	0
75%	0.46	6.03E+04	1.88E+05	2213.375	1977	4	2	1
max	17.5	1.87E+06	5.82E+06	19728.2499	2017	11	10	3

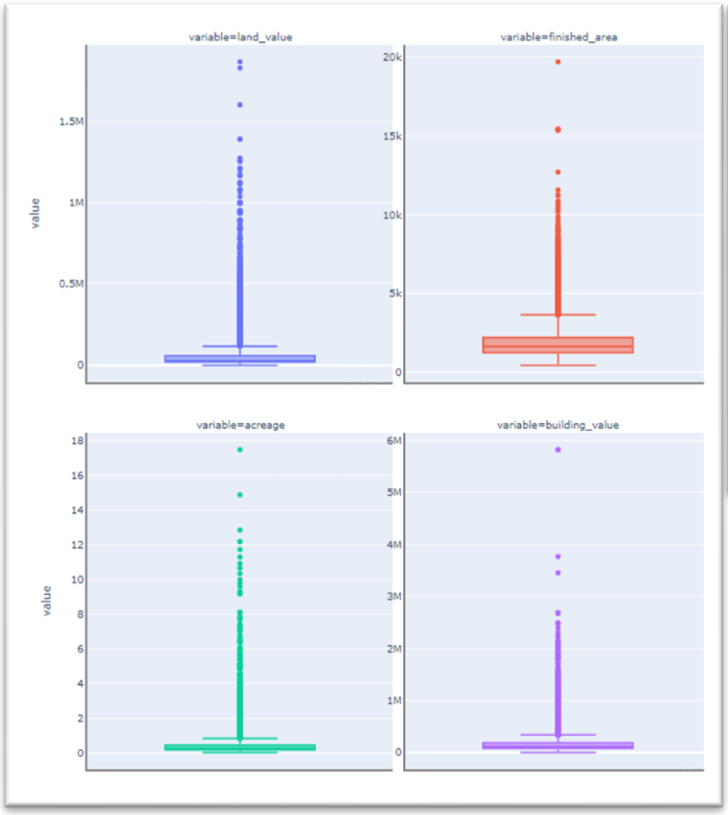
1. The target variable building_value shows a mean of 172,000 and the max value is high at 582,000. In outliers analysis, however, we must decide whether to proceed with the analysis including that value.
2. The average value of year_built is 1961.94. The oldest house was built in 1832, and the most recent house was built in 2017, so you can see that the range is quite wide.
3. Bedrooms are distributed from 0 to 11, but you can see that most of them are distributed between 3 and 4. This also seems to require outlier analysis.
4. finished_area also has a large max value of 19728. Therefore, outlier analysis is necessary.

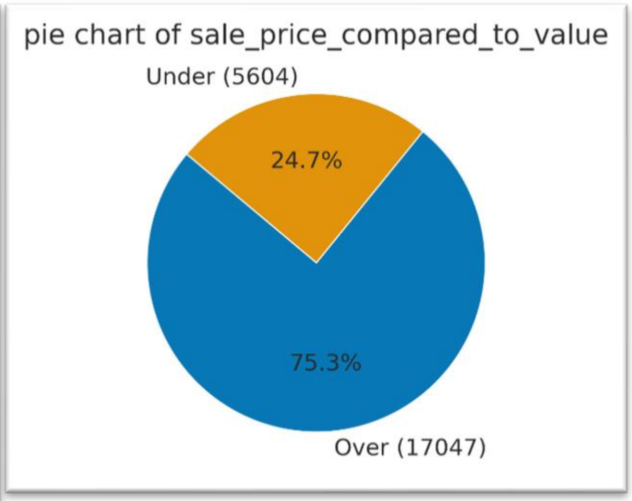
Data Cleansing

1. I first changed the sale_date column to date format and then divided the columns into year, month, and day. The existing sale_date column was excluded from the analysis.
2. After checking for missing values, all values in the suite/Condo # column were confirmed to be all missing values and column was removed. In the remaining columns, there were 108 half baths, or 0.47% of the total, so all missing values between 3 and 1 in bedrooms, property address, property city, full bath, foundation type, and finished were removed.
3. The columns unnamed:_0 and parcel_id were excluded because they are automatically assigned data.
4. legal_reference was excluded because it was judged to be difficult to analyze as a category or numerical variable because it had 22,452 unique values.
5. States were excluded because they all had one unique value, TN (Tennessee).
6. As a result of searching, neighborhood appeared to represent a zip code, so it was decided to exclude it from the linear regression analysis.

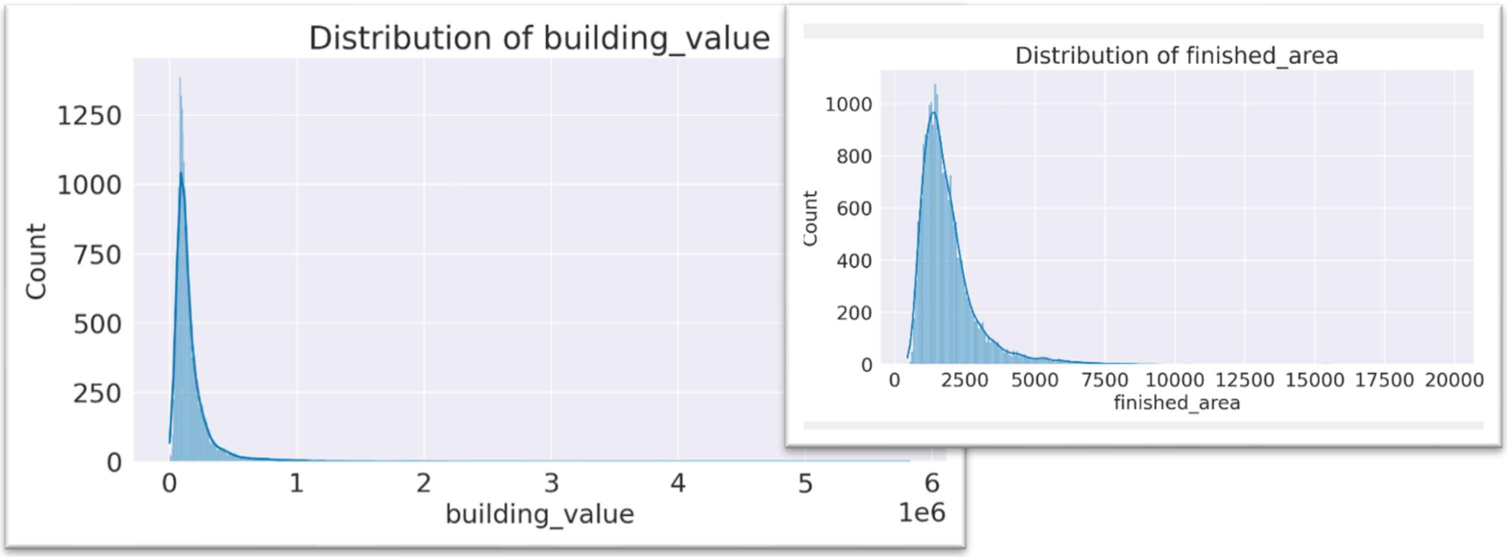
Data Visualizations

BoxPlots of Primary Features & Pie chart of sale_price_compraed_to_value

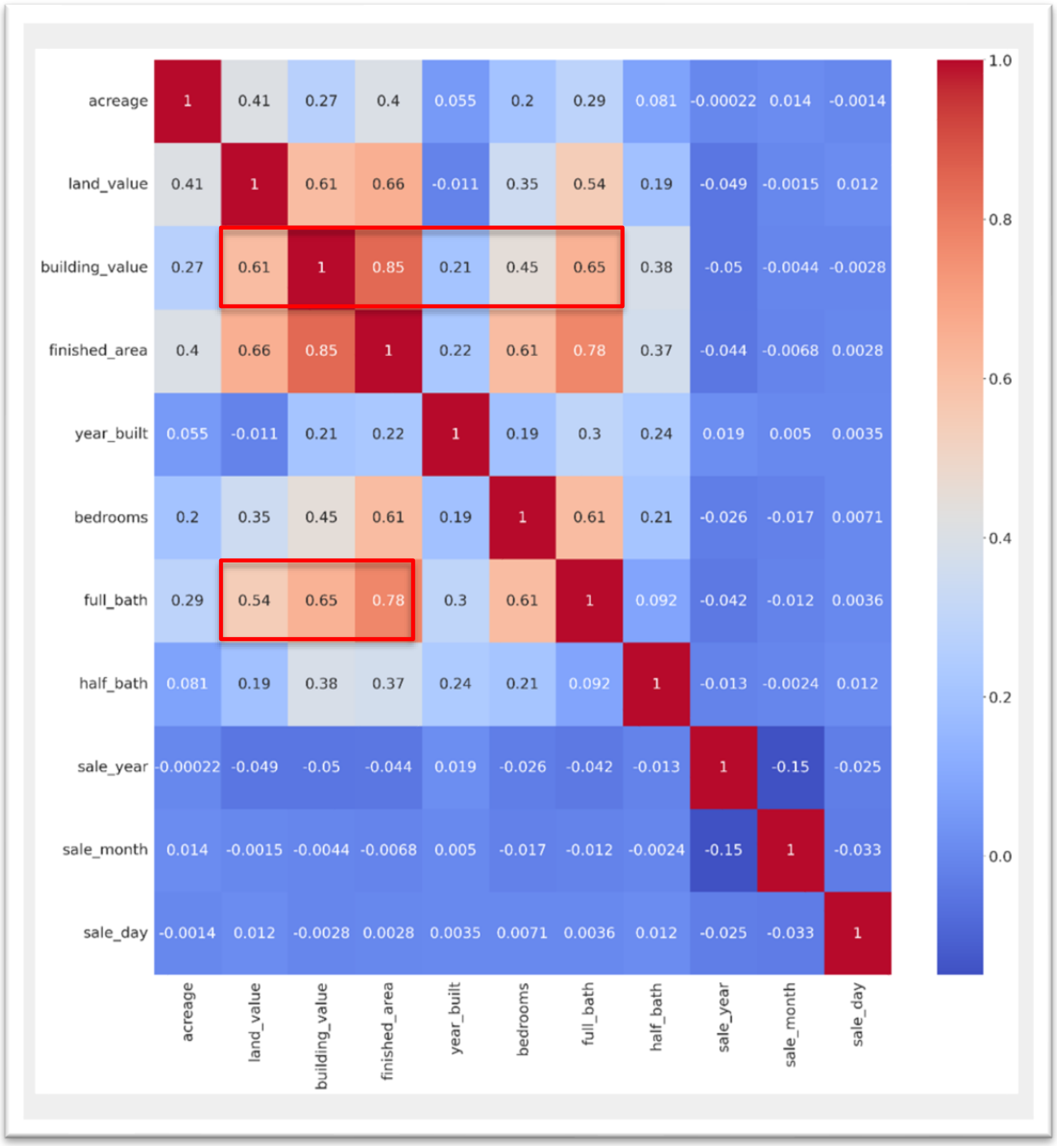




From the boxplot presented, it's evident that several data points stand out as potential outliers. Even though I've already opted to exclude missing values, I've decided to incorporate these points into the analysis. This decision stems from recognizing the nature of housing data, which can display varied prices based on property size. Nonetheless, for a more segmented analysis that distinctly categorizes 'over' and 'under', we can contemplate individually removing these outliers. Transitioning to the pie chart on the right, it illustrates a comparison derived from the decision tree relative to our objective metric. The chart underscores an imbalance, with 'Over' representing a significant 75% of the data.



In the depicted visualization on the left, you can observe the building value, serving as the target variable for our linear regression model. As highlighted by the preceding box plot, this variable showcases several notably high data points, leading to a right-skewed distribution. A similar right-skewed trend is evident in the distribution of the 'finished_area'. Following this preliminary analysis, it would be prudent to engage with the business team to strategize on the approach towards outlier treatment in subsequent assessments.



	acreage	land_value	building_value	finished_area	year_built	bedrooms	full_bath	half_bath	sale_year	sale_month	sale_day
acreage	1	0.41	0.27	0.4	0.055	0.2	0.29	0.081	-0.00022	0.014	-0.0014
land_value	0.41	1	0.61	0.66	-0.011	0.35	0.54	0.19	-0.049	-0.0015	0.012
building_value	0.27	0.61	1	0.85	0.21	0.45	0.65	0.38	-0.05	-0.0044	-0.0028
finished_area	0.4	0.66	0.85	1	0.22	0.61	0.78	0.37	-0.044	-0.0068	0.0028
year_built	0.055	-0.011	0.21	0.22	1	0.19	0.3	0.24	0.019	0.005	0.0035
bedrooms	0.2	0.35	0.45	0.61	0.19	1	0.61	0.21	-0.026	-0.017	0.0071
full_bath	0.29	0.54	0.65	0.78	0.3	0.61	1	0.092	-0.042	-0.012	0.0036
half_bath	0.081	0.19	0.38	0.37	0.24	0.21	0.092	1	-0.013	-0.0024	0.012
sale_year	-0.00022	-0.049	-0.05	-0.044	0.019	-0.026	-0.042	-0.013	1	-0.15	-0.025
sale_month	0.014	-0.0015	-0.0044	-0.0068	0.005	-0.017	-0.012	-0.0024	-0.15	1	-0.033
sale_day	-0.0014	0.012	-0.0028	0.0028	0.0035	0.0071	0.0036	0.012	-0.025	-0.033	1

The correlation matrix provides insights into the factors that strongly influence the 'building_value'. Notably, 'full_bath' is intertwined with other pivotal numerical variables, leading us to contemplate its exclusion from the ultimate linear regression model. Furthermore, I refined our analysis by evaluating the Variance Inflation Factor (VIF) to inspect multi-collinearity issues. Although 'finished_area' exhibited a notable correlation with 'acreage' at 0.4, this wasn't as pronounced as anticipated. As a result, I'll revisit this relationship by assessing it again through VIF.

Result of Linear Regression

Result 1: before feature selection and normalization

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         building_value   R-squared:                       0.735
Model:                            OLS   Adj. R-squared:                  0.735
Method:                 Least Squares   F-statistic:                     4999.
Date:                Thu, 19 Oct 2023   Prob (F-statistic):               0.00
Time:                        21:32:37   Log-Likelihood:            -2.3289e+05
No. Observations:               18028   AIC:                         4.658e+05
Df Residuals:                   18017   BIC:                         4.659e+05
Df Model:                          10
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          3.181e+06   1.39e+06      2.295      0.022    4.64e+05     5.9e+06
acreage       -3.103e+04   1407.094    -22.055      0.000   -3.38e+04   -2.83e+04
land_value        0.2246      0.010     22.398      0.000       0.205       0.244
finished_area   143.1306      1.438     99.569      0.000     140.313     145.948
year_built      182.7186     31.859      5.735      0.000     120.273     245.164
bedrooms      -2.553e+04   1168.102    -21.857      0.000   -2.78e+04   -2.32e+04
full_bath      6821.6508   1425.523      4.785      0.000    4027.489    9615.812
half_bath       3.39e+04   1822.694     18.596      0.000    3.03e+04    3.75e+04
sale_year     -1779.0124    688.062     -2.586      0.010   -3127.679    -430.345
sale_month       25.6320    242.105      0.106      0.916    -448.918     500.182
sale_day       -135.9338     81.563     -1.667      0.096    -295.806      23.938
==============================================================================
Omnibus:                    25570.369   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         46611780.720
Skew:                           7.705   Prob(JB):                         0.00
Kurtosis:                     251.626   Cond. No.                     2.37e+08
==============================================================================
Notes:
 [1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
 [2] The condition number is large, 2.37e+08. This might indicate that there are
 strong multicollinearity or other numerical problems.
```

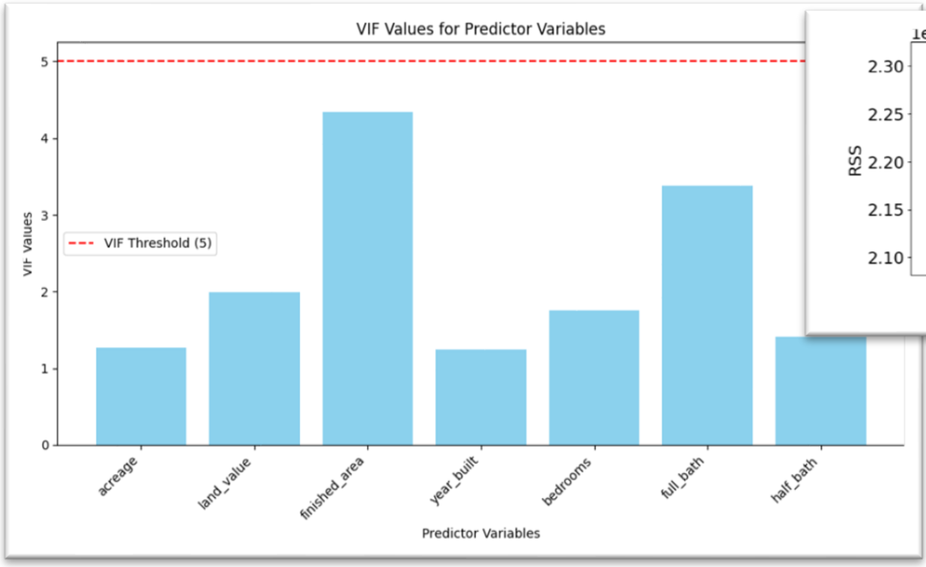
I started off by constructing a linear regression using all the numerical variables. Then, I streamlined the model by removing variables with insignificant p-values. Throughout this process, I also kept a close eye on multicollinearity by regularly checking the Variance Inflation Factor (VIF).

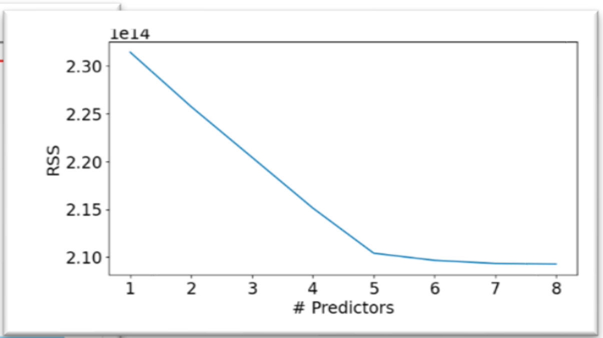
Result 2: Removing low p-value

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         building_value   R-squared:                       0.743
Model:                            OLS   Adj. R-squared:                  0.743
Method:                 Least Squares   F-statistic:                     6510.
Date:                Thu, 19 Oct 2023   Prob (F-statistic):               0.00
Time:                        21:32:37   Log-Likelihood:            -2.3301e+05
No. Observations:               18028   AIC:                         4.660e+05
Df Residuals:                   18019   BIC:                         4.661e+05
Df Model:                           8
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          2.709e+06   1.38e+06      1.962      0.050    2193.618    5.42e+06
```

```
acreage        -3.071e+04   1363.202    -22.525      0.000    -3.34e+04    -2.8e+04
land_value         0.2336      0.010     23.401      0.000        0.214       0.253
finished_area    145.7677      1.420    102.683      0.000      142.985     148.550
year_built       184.8063     32.099      5.757      0.000      121.890     247.723
bedrooms       -2.665e+04   1180.772    -22.567      0.000     -2.9e+04   -2.43e+04
full_bath       5710.7207   1429.137      3.996      0.000     2909.476    8511.966
half_bath       3.312e+04   1830.837     18.089      0.000     2.95e+04    3.67e+04
sale_year      -1547.9043    685.739     -2.257      0.024    -2892.018    -203.791
==============================================================================
Omnibus:                    25072.884   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         40904275.701
Skew:                           7.432   Prob(JB):                         0.00
Kurtosis:                     235.881   Cond. No.                     2.37e+08
==============================================================================
     Notes:
 [1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
 [2] The condition number is large, 2.37e+08. This might indicate that there are
strong multicollinearity or other numerical problems.
Notes:
```

In the initial linear regression, I removed 'sale_month' and 'sale_day' due to their low p-values. I went back and compared both the p-value and VIF once more.





I opted to compare the Residual Sum of Squares (RSS) using the 'best_selection' method. From this, I chose to move forward with the most significant five variables for my final analysis. Among these, 'full_bath' had the least significance, leading to our decision to remove it. I closely examined the variables with VIF values exceeding 3. Given that 'full_bath' and 'half_bath' are semantically related, I decided to remove 'full_bath' due to its elevated VIF.

Result 3: with categorical variables as dummy variables after normalization

```
                            OLS Regression Results
==============================================================================
Dep. Variable:         building_value   R-squared:                       0.735
Model:                            OLS   Adj. R-squared:                  0.735
Method:                 Least Squares   F-statistic:                     8313.
Date:                Thu, 19 Oct 2023   Prob (F-statistic):               0.00
Time:                        21:33:00   Log-Likelihood:            -2.3285e+05
No. Observations:               18028   AIC:                         4.657e+05
Df Residuals:                   18021   BIC:                         4.658e+05
Df Model:                           6
Covariance Type:            nonrobust
```

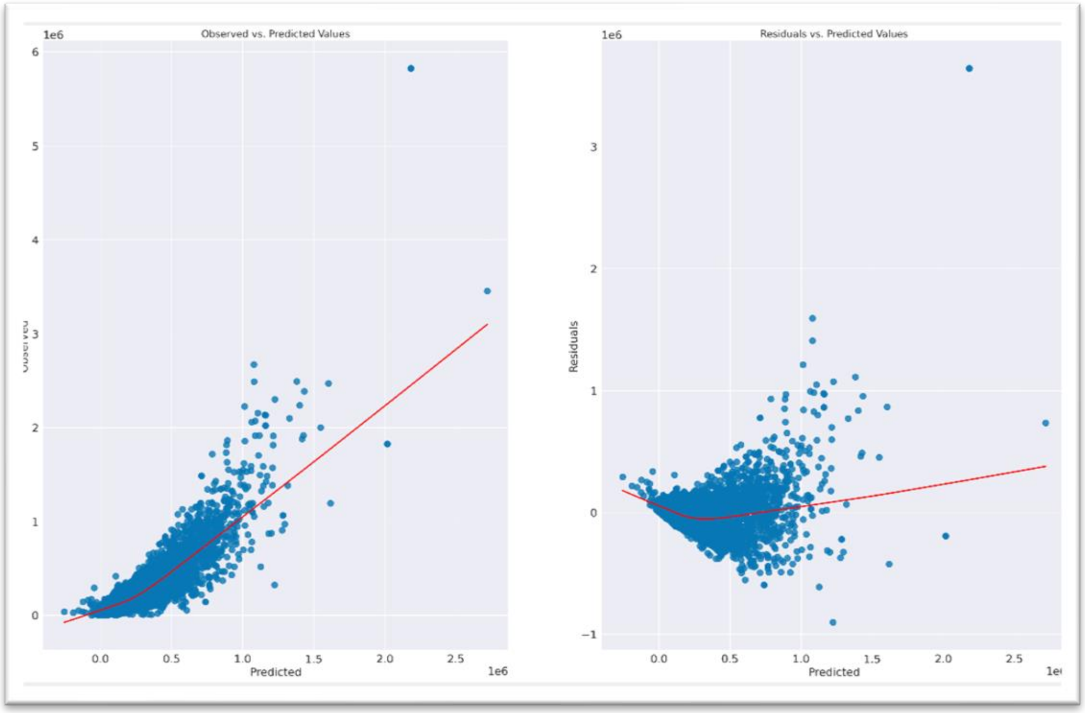
```
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const           1.625e+04   2975.826      5.461      0.000    1.04e+04    2.21e+04
acreage        -5.298e+05   2.32e+04    -22.814      0.000   -5.75e+05   -4.84e+05
land_value       4.27e+05   1.82e+04     23.448      0.000    3.91e+05    4.63e+05
finished_area   2.864e+06   2.24e+04    127.883      0.000    2.82e+06    2.91e+06
bedrooms       -2.594e+05   1.23e+04    -21.039      0.000   -2.84e+05   -2.35e+05
half_bath        9.45e+04   4974.609     18.997      0.000    8.48e+04    1.04e+05
sold_as_vacant  6.133e+04   9807.383      6.253      0.000    4.21e+04    8.05e+04
==============================================================================
Omnibus:                    25315.703   Durbin-Watson:                   2.018
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         43579468.002
Skew:                           7.564   Prob(JB):                         0.00
Kurtosis:                     243.389   Cond. No.                         37.7
==============================================================================
```

In the final linear regression, I've focused on 'sold_as_vacant' due to its significance among the categorical variables. I've also included 'acreage', 'finished_area', 'bedrooms', and 'half_bath' as they are key indicators of a house's size. The model has an R-squared value of 0.735. This suggests that our model can explain roughly 73.5% of the changes in the building's value.

The Adjusted R-squared is closely aligned with the R-squared, indicating that the variables I've included in our model are indeed meaningful and play a significant role in predicting the building value.

A noteworthy observation is that with every extra acre of land, the value of the building drops by 529,800 units, assuming other factors remain unchanged. Similarly, with each unit rise in the land's value, the associated building's value goes up by 427,000 units, given everything else stays constant. An increase in the finished area by a single unit leads to a massive jump of 2,864,000 units in the building value. Interestingly, adding an extra bedroom reduces the building value by 259,400 units.On the flip side, for every added half bath, the building's worth climbs by 94,500 units. If a property is labeled as being sold vacant, it enjoys a premium, with its value getting a boost of 61,330 units compared to properties that aren't sold as vacant.

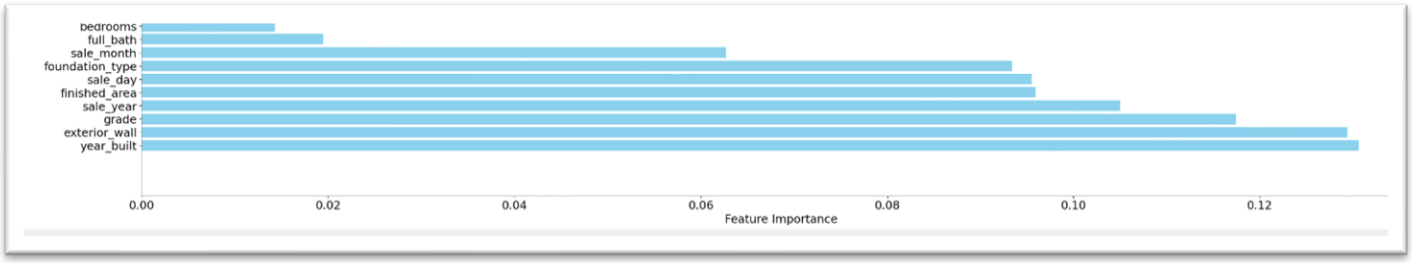
Checking Assumption



By examining the Observed vs. Predicted Value graph, I can assess linearity. But when observing the residuals vs. predicted value, there's a noticeable clustering of points, revealing a distinct pattern. This suggests we might need to further analyze the normality of our data. To optimize our linear regression, it might be beneficial to reconsider our feature selection or to further preprocess the data.

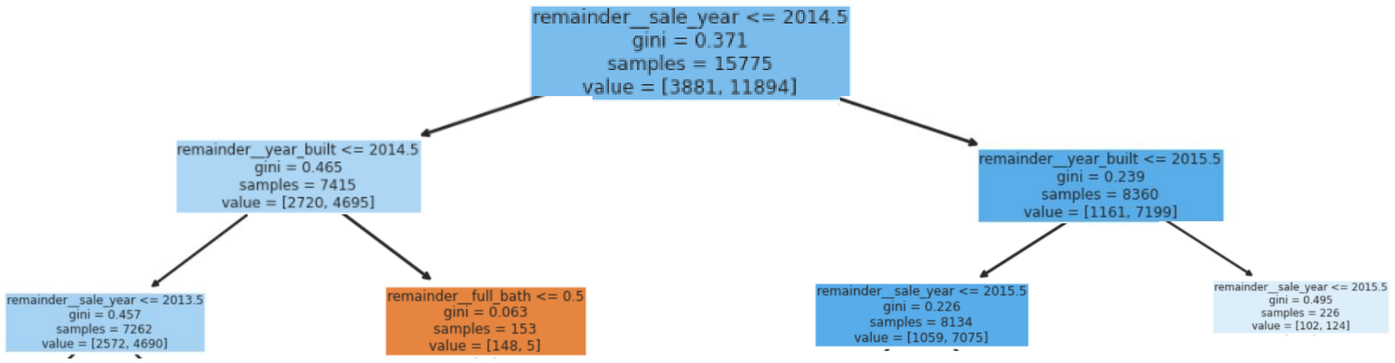
Result of Decision Tree

Features importance and selection

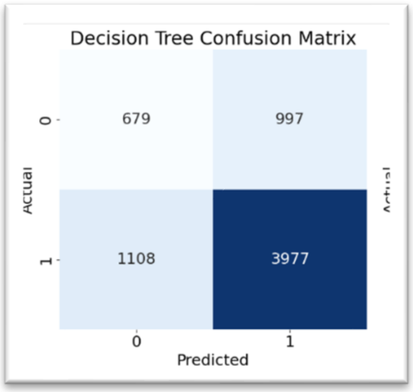


Initially, I built a decision tree incorporating all features. However, I then opted to refine our approach, focusing on a model that utilizes the top 10 most important features. For a comprehensive analysis, I also ran both random forest and XGBoost models, ensuring they were based on these same selected features.

Result: Layer 1 and 2



1. The presented visualization illustrates a decision tree with its depth set to 5. In the initial layer, the tree segments the data based on the sale_year, specifically using a threshold of 2014.5.
2. The resulting gini impurity for this layer is 0.371. At the second layer, nodes further subdivide the data. Here, entries with a sale_year below 2014.5 are distinguished from those above it.
3. Additionally, the year_built attribute is used to further differentiate entries based on whether they fall before or after 2015.5.
4. By the third layer, the tree incorporates additional attributes for segmentation, namely the sale_year (again) and full_bath.

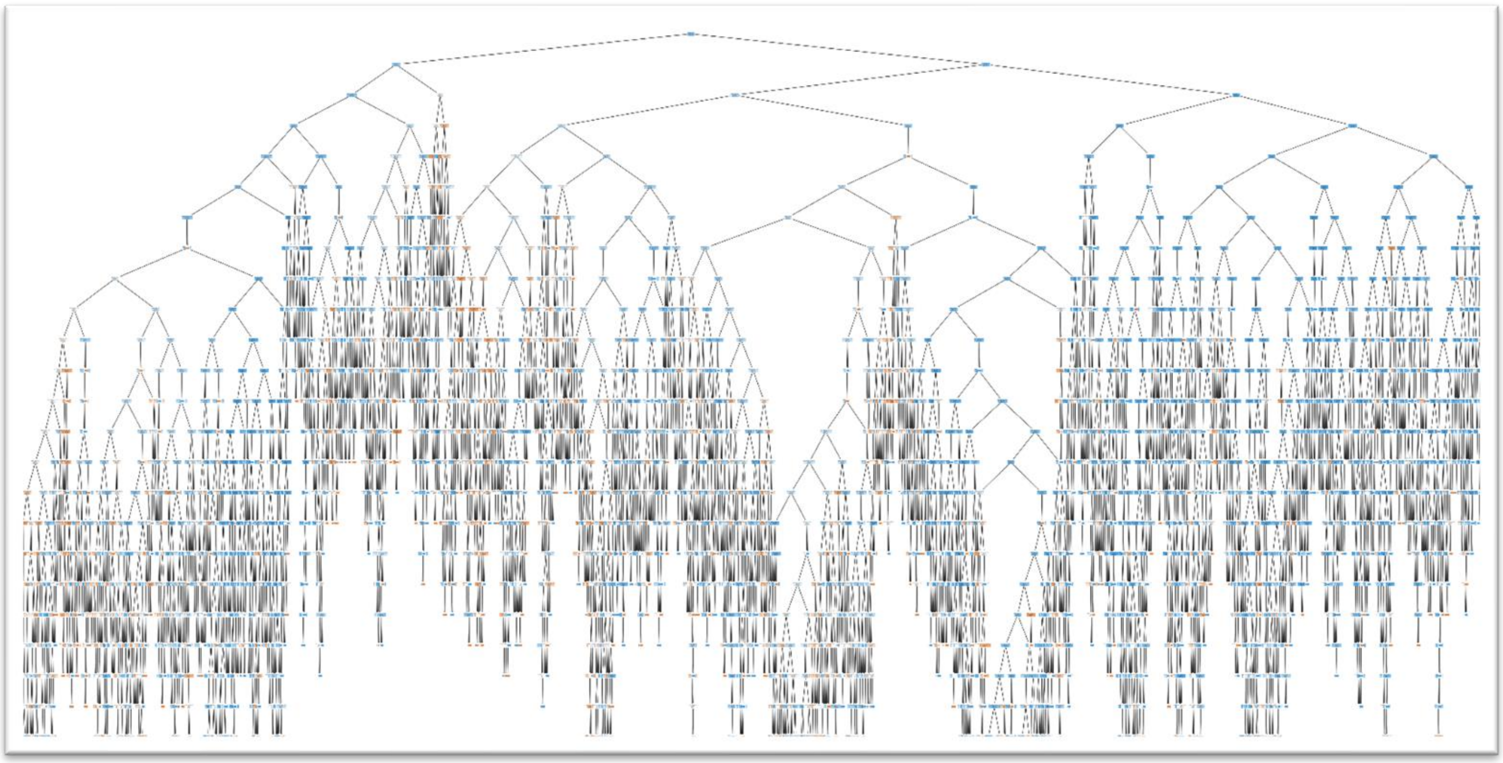


Looking at the confusion matrix of the decision tree model, the accuracy was 0.793, the precision was 0.800, and the recall was 0.785. Ultimately, the F1-score recorded 0.793. In general, all benchmarks were uniformly high.

Accuracy is not a good metric to use when you have class imbalance. One way to solve class imbalance problems is to work on your sample. Another way to solve class imbalance problems is to use better accuracy metrics like the F1 score (Joos, 2001).

Result of Random Forest

Result: Number 0 Tree

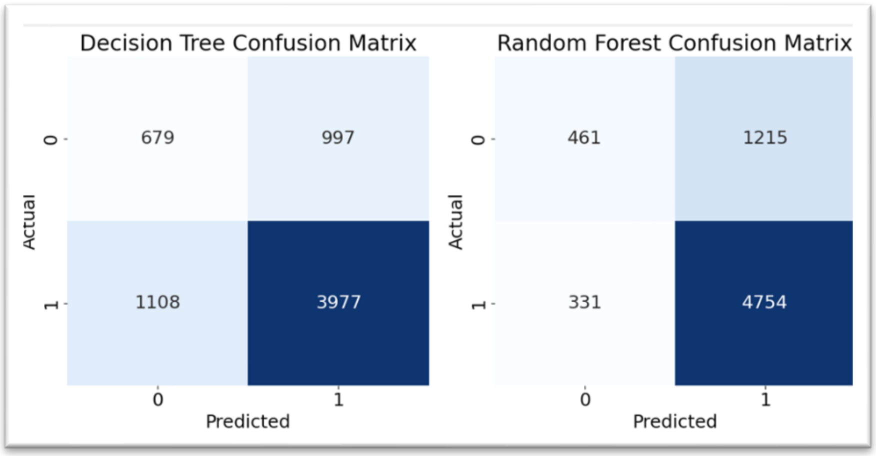


This visualization represents just one of the many decision trees within the random forest model. Given its intricate layers, you'd need to zoom into specific sections to grasp the finer details.

How many trees and what is the depth of each tree

Tree Number	Tree Depth
1	32
2	37
3	32
4	33
5	32
...	...
96	33
97	36
98	33
99	32
100	34

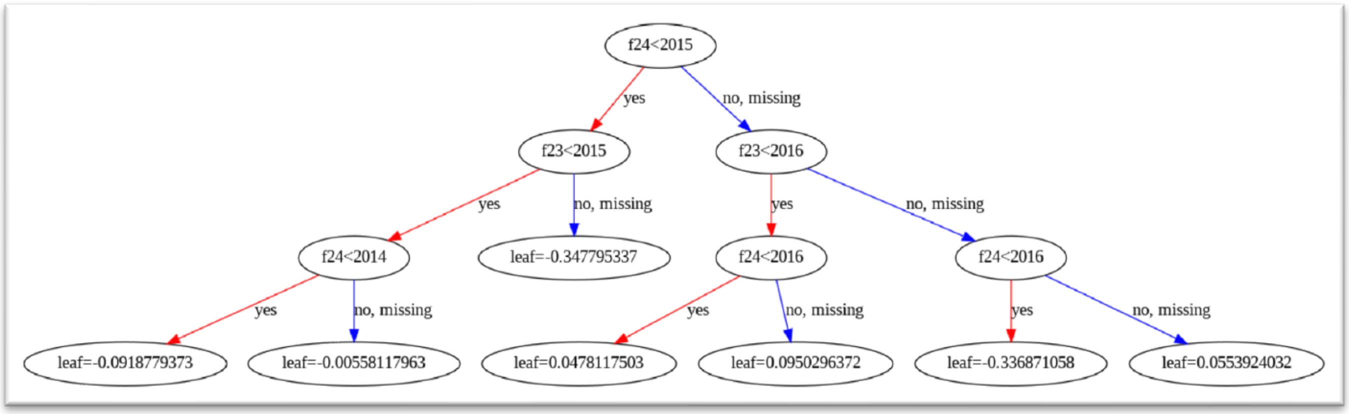
What I have here represents the total count of trees in the random forest, along with the depth of each tree. Classifications are primarily executed through trees that have a depth ranging from 32 to 37.



When comparing the confusion matrices of both the decision tree and random forest, I notice an uptick in the true positive count (predicted: 1, actual: 1) located in the lower right quadrant. Conversely, the true negative count (predicted: 0, actual: 0) in the upper left corner seems to have dropped. While this might suggest an improvement in accuracy or recall, it could potentially indicate a decrease in precision.

Result of XGBoost

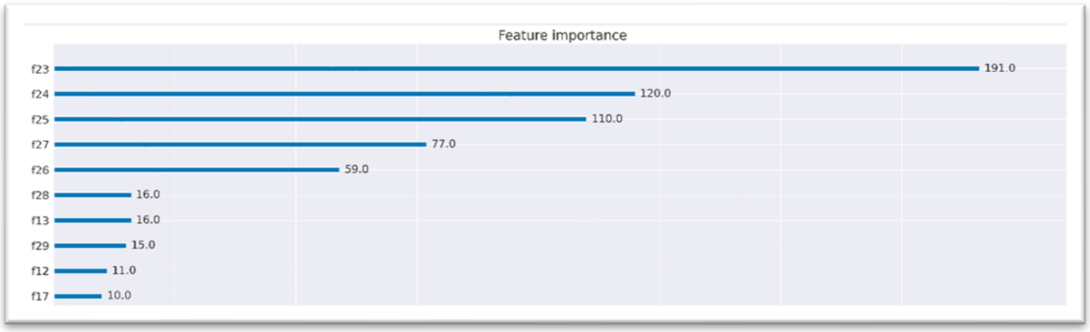
Result: Number 0 Tree



I persisted with the 10 features that were previously employed in the decision tree and max_depth as 3. However, due to the inclusion of dummy variables, the total number of features expands to 29. To identify each feature, one can refer to the re-encoded data. These features are detailed in the table below, ranked by their importance.

Features and Feature Importance of XGBoost

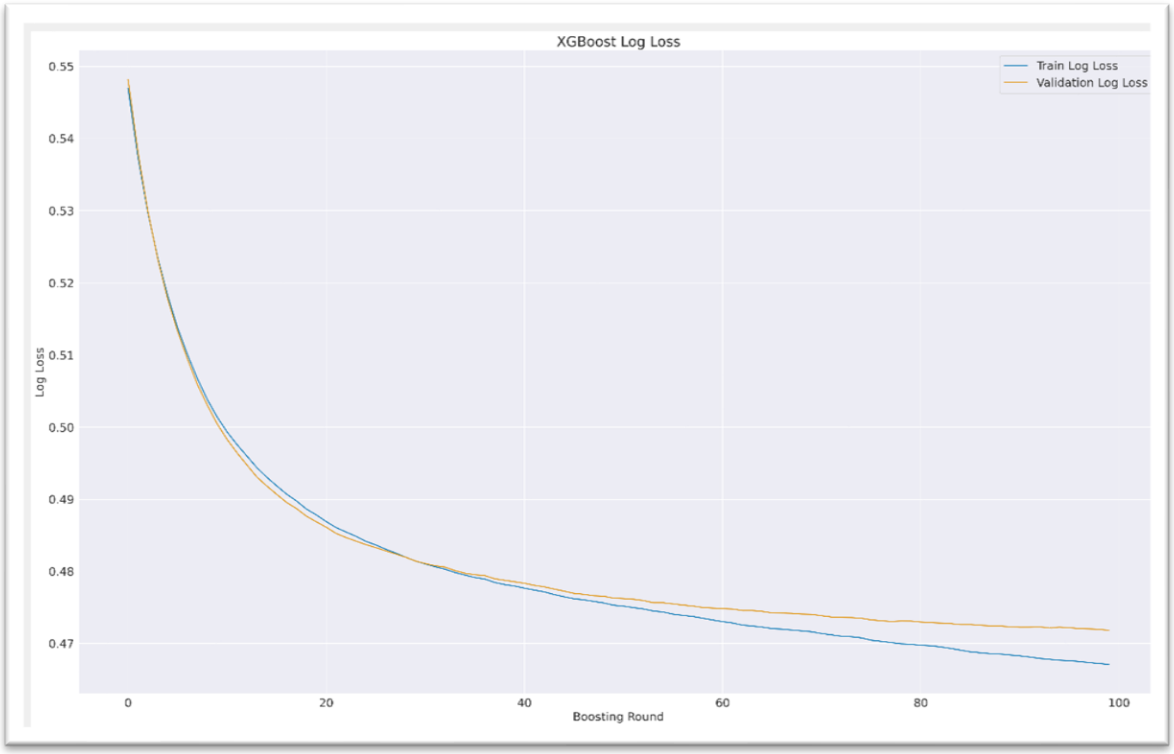
	Default Feature Name	Original Feature Name
0	f23	year_built
1	f24	sale_year
2	f25	finished_area
3	f27	sale_month
4	f26	sale_day



Parameters of XGBoost

1. 'max_depth' = 3: This defines the maximum depth of any given tree within the model. The depth of a tree is essentially the number of steps it can take from the root to the farthest leaf. Here, the trees will be limited to a depth of 3.
2. 'learning_rate' = 0.1: Choose a relatively high learning rate. Generally, a learning rate of 0.1 works. This parameter is used to prevent overfitting. A value of 0.1 means that each tree's contribution to the final prediction will be reduced or "shrunk" by 10% (Jain, 2023)
3. 'n_estimators' = 100: This parameter determines the number of trees, or "boosting rounds", that the model will construct. So, the model will be built using a total of 100 decision trees.
4. 'eval_metric' = 'logloss': The log loss, or logarithmic loss, measures the performance of a classification model where the prediction input is a probability value between 0 and 1. A lower log loss value indicates a better-performing model.

How many trees and what is the depth of each tree



The line graph illustrates the log loss trajectory of the XGBoost model across its boosting rounds. Notably, post the 100th boosting round, there's a discernible decline in the model's log loss. While the training model's loss steadily diminishes, there's an uptick in the log loss of the validation data post the 40th round. Based on this observation, although the rounds have been initially set at 100, there's a rationale to consider capping the rounds at approximately 40, 50, or even 60. However, if the magnitude isn't deemed substantial, retaining it at 100 might be appropriate.

Conclusion

In the scope of this project, I undertook a comprehensive examination of housing prices and their potential overvaluation utilizing methods such as linear regression, decision trees, random forests, and XGBoost. A predominant takeaway from this endeavor was the paramount importance of data cleaning. As an individual who places significant emphasis on data integrity, I believe that ambiguous outliers warrant collaborative evaluation with cross-functional teams. Central to investigation was an emphasis on visualizing insights from XGBoost and Random Forest models. Additionally, I delved into understanding the interconnectedness and nuances among the three modeling techniques, including decision trees.

Reference

Jain, A. (2023, July 11). Complete guide to parameter tuning in XGBoost (with codes in Python). Analytics Vidhya. Retrieved from https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Joos, Korstanje. (2021, August 31). The F1 score. Medium. Retrieved from https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

Esther, Abel. (2023, February 10). Nashville housing data analysis: SQL project. Medium. Retrieved from https://medium.com/@abelesther/nashville-housing-data-analysis-sql-project-78233238eaba

shivapriya1726. (2023, September 29). How to compare two columns in pandas. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/how-to-compare-two-columns-in-pandas/

Kent State University Libraries. (n.d.). SPSS: Chi-Square Test of Independence. Kent State University. Retrieved from https://libguides.library.kent.edu/spss/chisquare#:~:text=The%20Chi%2DSquare%20Test%20of%20Independence%20determines%20whether%20there%20is,Chi%2DSquare%20Test%20of%20Association.

mkln. (2013, December 10). How to reset index in a pandas dataframe. StackOverflow. Retrieved from https://stackoverflow.com/questions/20490274/how-to-reset-index-in-a-pandas-dataframe

Eric, Kleppen. (2023, May 11). How to Find Outliers in Data: A Guide for Beginner Analysts. CareerFoundry. Retrieved from https://careerfoundry.com/en/blog/data-analytics/how-to-find-outliers/

Eligijus, Bujokas. (2022, June 1). Feature importance in Decision Trees. Medium. Retrieved from https://towardsdatascience.com/feature-importance-in-decision-trees-e9450120b445

mljar. (n.d.). How to visualize decision trees. Retrieved from https://mljar.com/blog/visualize-decision-tree/

#000000, (2018). Exploring nashville. Kaggle. Retrieved from https://www.kaggle.com/code/stevenknguyen/exploring-nashville

Sandeep, Ram. (2020, October 18). Mastering random forests: A comprehensive guide. Medium. Retrieved from https://towardsdatascience.com/mastering-random-forests-a-comprehensive-guide-51307c129cb1

Brownlee, J. (2020, August 27). How to visualize gradient boosting decision trees with XGBoost in Python. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/visualize-gradient-boosting-decision-trees-xgboost-python/

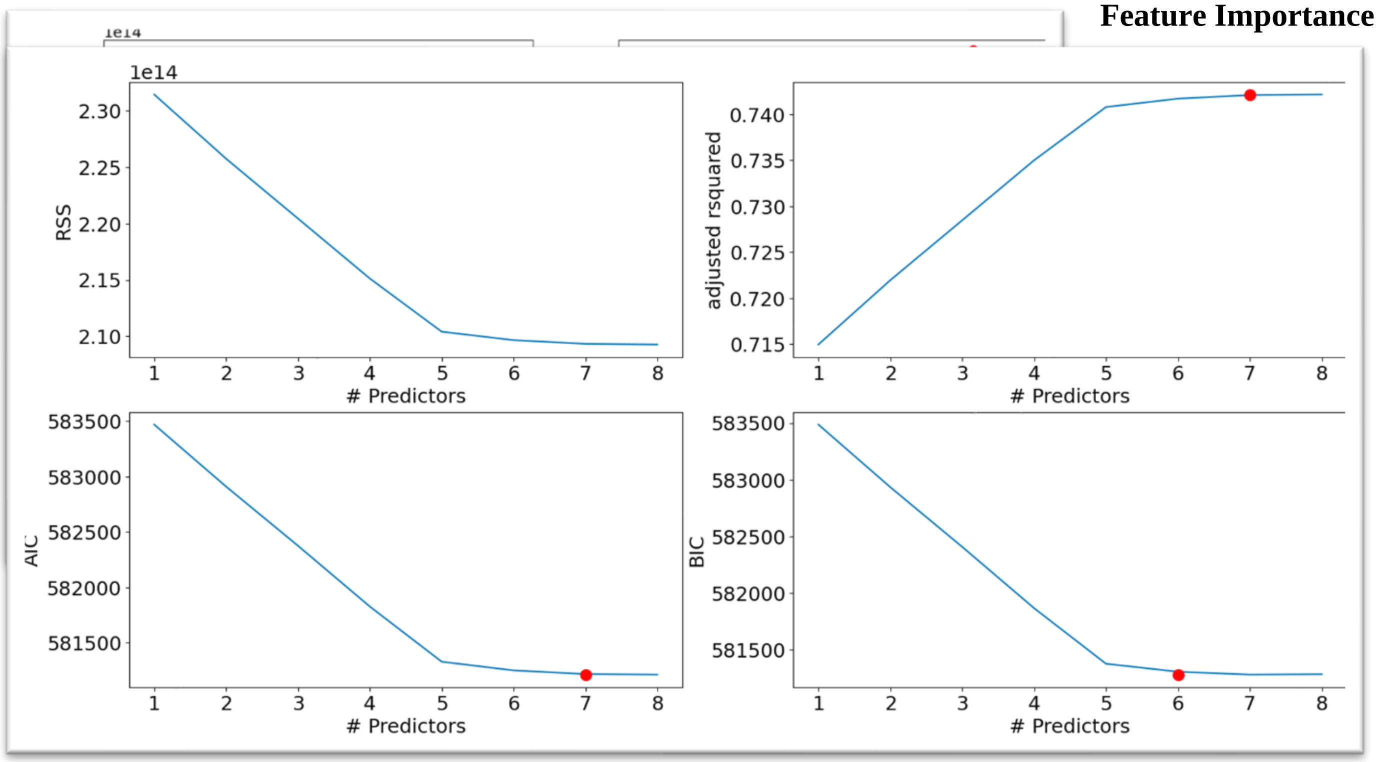
Brownlee, J. (2020, August 27). Feature importance and feature selection with XGBoost in Python. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/

Saha, S. (2018, October 7). Understanding the log-loss function of XGBoost. Medium - Data Driven Investor. Retrieved from https://medium.datadriveninvestor.com/understanding-the-log-loss-function-of-xgboost-8842e99d975d

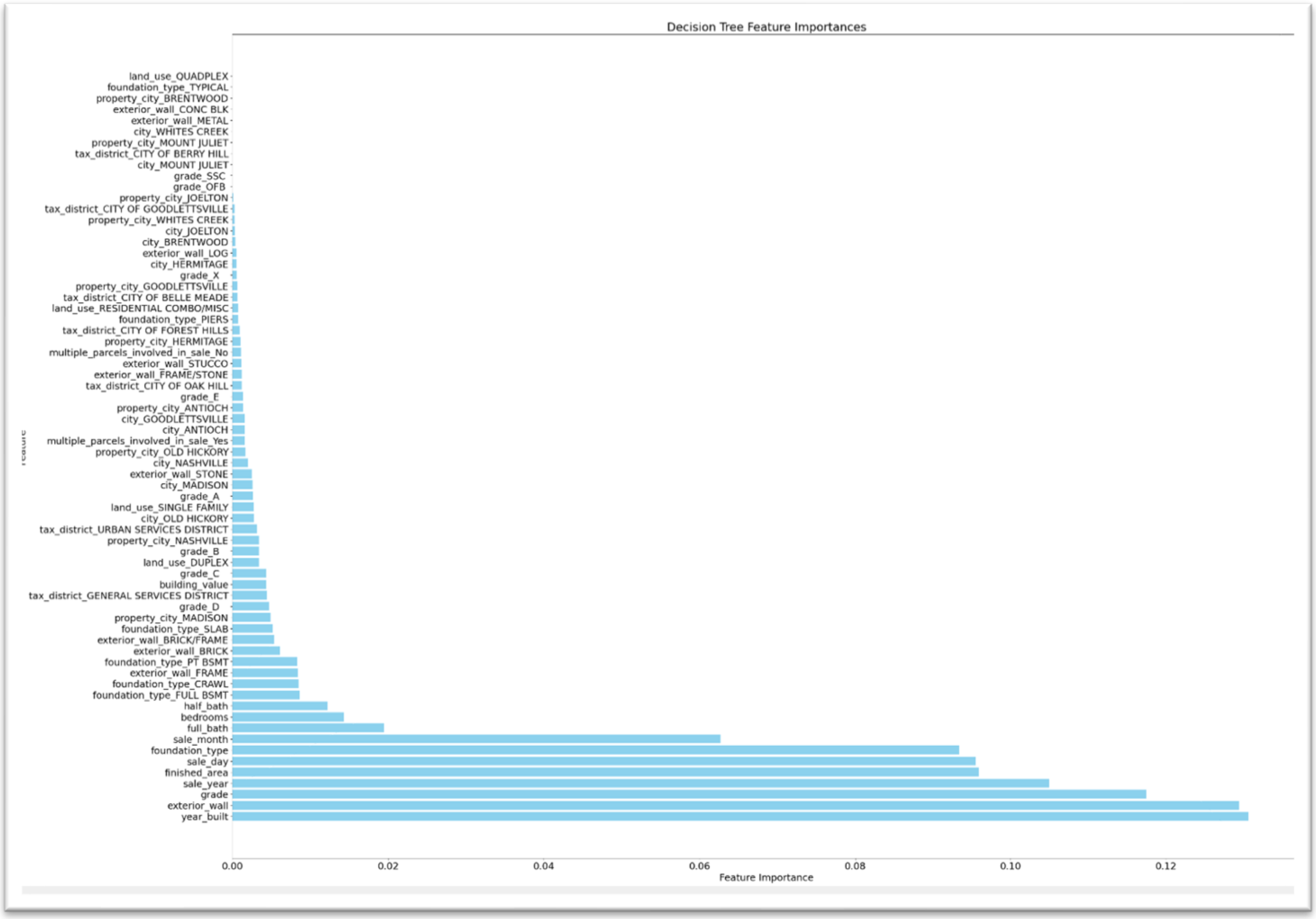
Appendix (Graphs):

Linear Regression best subset method

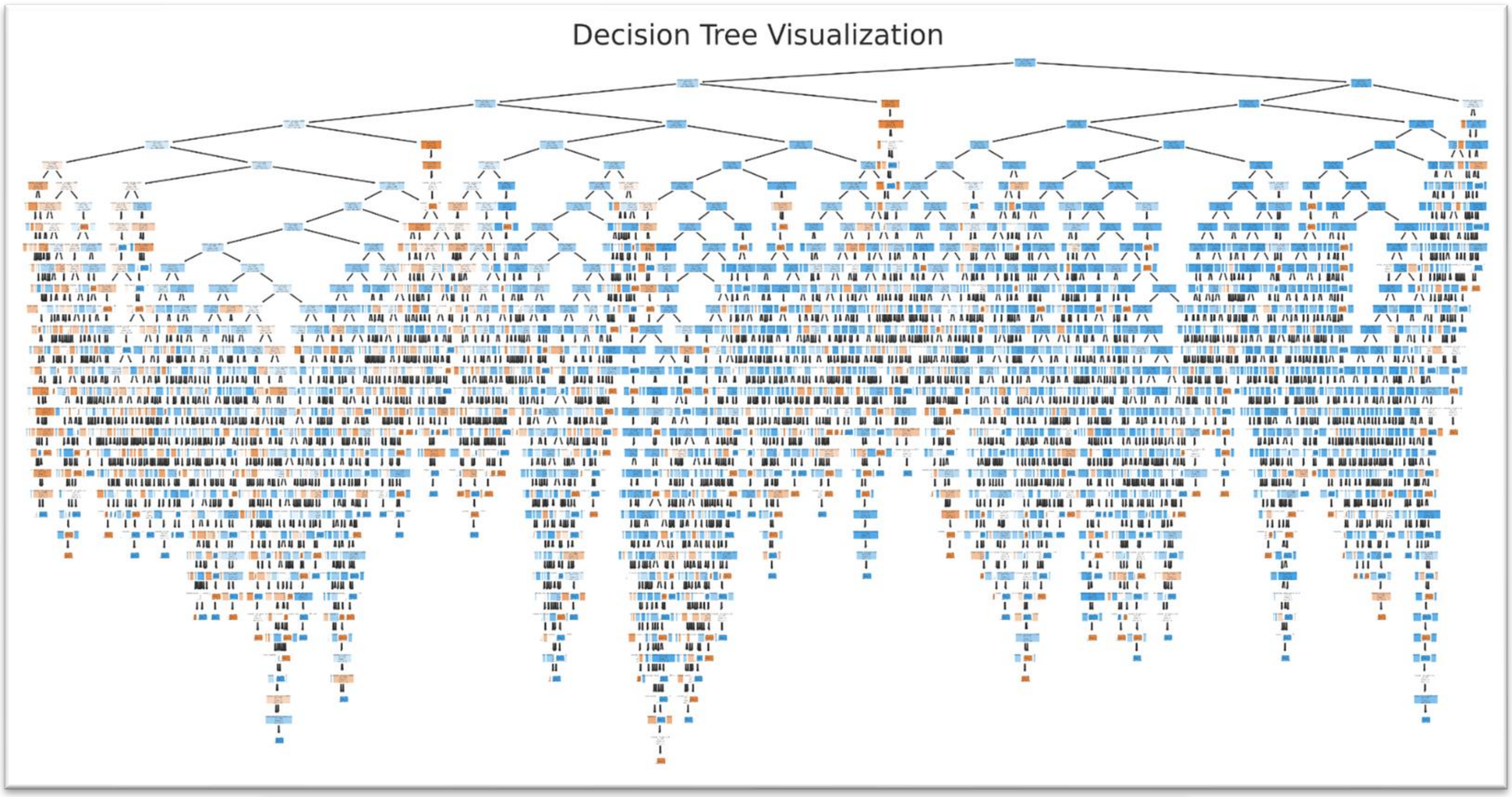
Feature Importance

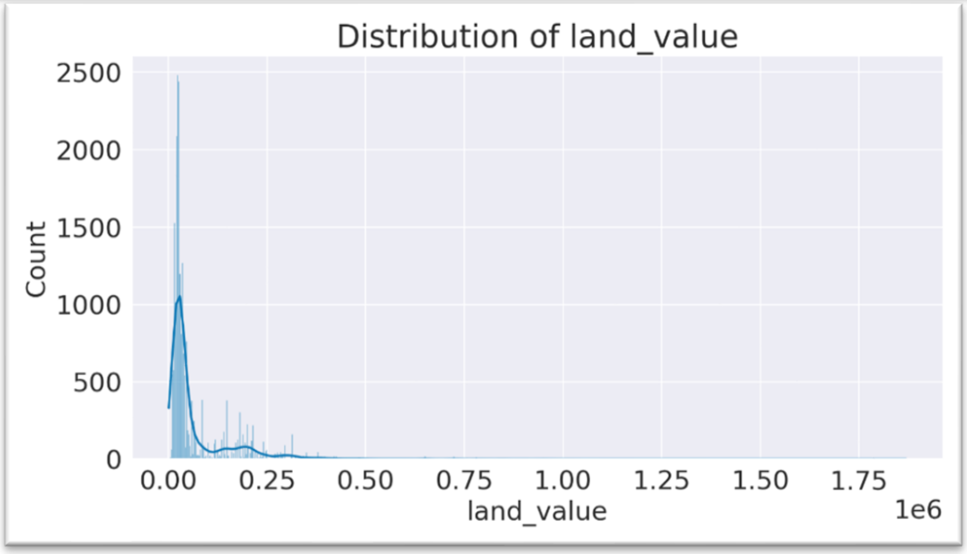


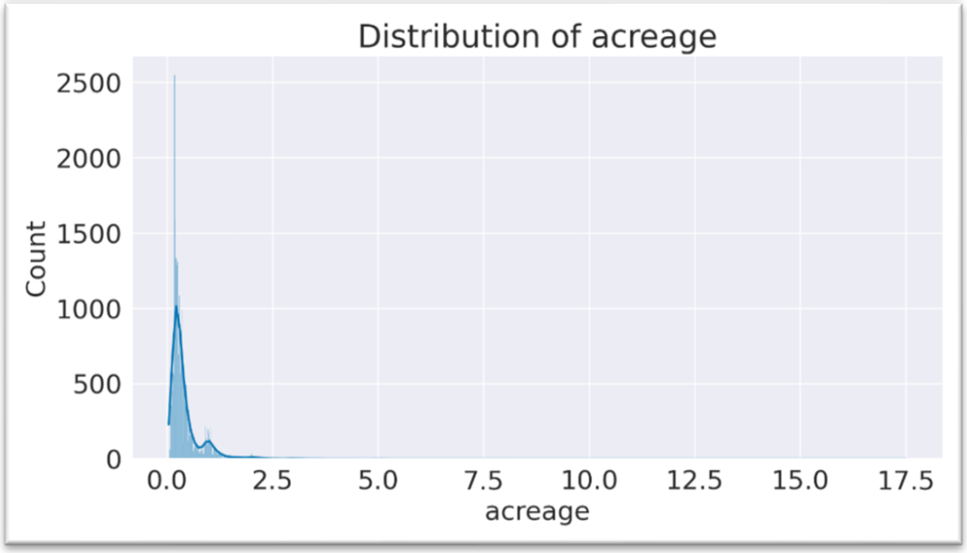
of Decision tree model



Whole Decision Tree model







Appendix (Python code):

```python
import pandas as pd
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from collections import Counter
import seaborn as sns

"""### DATA Import"""

df = pd.read_csv('week 4 - Nashville_housing_data.csv')

df.head()

df.tail()

"""## Visualization & Understanding Dataset"""

def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys=['Missing_Number',
'Missing_Percent'])
    return missing_values[missing_values['Missing_Number']>0]

def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('-'*79, 'red', attrs=['bold']),
          colored("\nInfo:\n", attrs=['bold']), sep='')
    print(df.info(), '\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", attrs=['bold']), df.nunique(),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values:\n", attrs=['bold']), missing_values(df),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", attrs=['bold']), list(df.columns),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')

    df.columns= df.columns.str.lower().str.replace('&', '_').str.replace(' ', '_')

    print(colored("Columns after rename:", attrs=['bold']), list(df.columns),'\n',
              colored('-'*79, 'red', attrs=['bold']), sep='')

pip install colorama

import colorama
from colorama import Fore, Style  # maakes strings colored
from termcolor import colored

missing_values(df)

first_looking(df)

df.describe()

## !pip install -U ydata-profiling[notebook]==4.0.0 matplotlib==3.5.1

## from ydata_profiling import ProfileReport

## df.profile_report()

"""## Data Preperation

"""

df1=df.copy()
```

```python
df

df=df.drop(['suite/_condo___#'], axis=1)

df['sale_date'] = pd.to_datetime(df['sale_date'], format='%m/%d/%Y')
df['sale_year'] = df['sale_date'].dt.year
df['sale_month'] = df['sale_date'].dt.month
df['sale_day'] = df['sale_date'].dt.day

"""### Linear Regression"""

df2=df.copy()
df2

df2=df2.dropna()
df2

df2=df2.drop(['unnamed:_0', 'parcel_id', 'property_address', 'sale_date', 'legal_reference', 'state',
'neighborhood'], axis=1)

df2.info()

cat_vars = [x for x in df2.columns if df2[x].dtype =="object"]
cat_vars

num_vars = [x for x in df2.columns if x not in cat_vars]
num_vars

df_cat = df2[cat_vars]
df_cat = df_cat.drop(['sale_price_compared_to_value'], axis=1)
df_cat.head()

res_equal = df_cat['property_city'].equals(df_cat['city'])
print(res_equal)

df_num=df2[num_vars]
df_num

car_corr_matrix=df_num.corr()

plt.figure(figsize=(20, 20))
sns.heatmap(car_corr_matrix, cmap='coolwarm', annot=True)

sns.set_palette('colorblind')
sns.pairplot(data=df_num)

df_anova=pd.concat([df_cat, df_num['building_value']], axis=1)
df_anova

from scipy.stats import f_oneway

# Running the one-way anova test between CarPrice and FuelTypes
# Assumption(H0) is that FuelType and CarPrices are NOT correlated

# Finds out the Prices data for each FuelType as a list
for i in df_anova.columns:
    if i != 'building_value':
        CategoryGroupLists = df_anova.groupby(i)['building_value'].apply(list).values
        AnovaResults = f_oneway(*CategoryGroupLists)
        print(f'P-Value for Anova with {i} is:', AnovaResults.pvalue)

"""#### Linear Regression right away

"""

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
import statsmodels.api as sm
from scipy import stats

y=df_num[['building_value']]
x=df_num.iloc[:,:]
x=x.drop(['building_value'], axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

X2 = sm.add_constant(x_train)
est = sm.OLS(y_train, X2)
est2 = est.fit()
print(est2.summary())

"""#### Removing low p-value"""

y=df_num[['building_value']]
x=df_num.iloc[:,:]
x=x.drop(['building_value','sale_month', 'sale_day'], axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

X2 = sm.add_constant(x_train)
est = sm.OLS(y_train, X2)
est2 = est.fit()
print(est2.summary())

y=df_num[['building_value']]
x=df_num.iloc[:,:]
x=x.drop(['building_value','sale_month', 'sale_day', 'sale_year'], axis=1)
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

X2 = sm.add_constant(x_train)
est = sm.OLS(y_train, X2)
est2 = est.fit()
print(est2.summary())

"""#### checking VIF values"""

y=df_num[['building_value']]
x=df_num.iloc[:,:]

from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor

#find design matrix for regression model using 'rating' as response variable
y, X = dmatrices('building_value ~ acreage+land_value+finished_area+year_built+bedrooms+full_bath+half_bath',
data=x, return_type='dataframe')

#create DataFrame to hold VIF values
vif_df = pd.DataFrame()
vif_df['variable'] = X.columns

#calculate VIF for each predictor variable
vif_df['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

#view VIF for each predictor variable
print(vif_df)

vif_df = vif_df[vif_df['variable'] != 'Intercept']
vif_df

plt.figure(figsize=(10, 6))
plt.bar(vif_df['variable'], vif_df['VIF'], color='skyblue')
plt.axhline(y=5, color='red', linestyle='--', label='VIF Threshold (5)')

plt.xlabel('Predictor Variables')
plt.ylabel('VIF Values')
plt.title('VIF Values for Predictor Variables')
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.tight_layout()

plt.show()

"""#### best subset method"""

import itertools
import time
import statsmodels.api as sm
import matplotlib.pyplot as plt

y=df_num[['building_value']]
x=df_num.iloc[:,:]

X=df_num.drop(['building_value'], axis=1)
X=X.astype('float64')
X

y=df2.building_value
y

X_=X
X_=pd.concat([X], axis=1)
X_

def processSubset(feature_set):
    # Fit model on feature_set and calculate RSS
    X_subset = sm.add_constant(X[list(feature_set)])
    model = sm.OLS(y,X_subset)
    regr = model.fit()
    RSS = ((regr.predict(X_subset) - y) ** 2).sum()
    return {"model":regr, "RSS":RSS}

def getBest(k):

    tic = time.time()

    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)

    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].argmin()]

    toc = time.time()
    print("Processed", models.shape[0], "models on", k, "predictors in", (toc-tic), "seconds.")

    # Return the best model, along with some other useful information about the model
    return best_model

# Could take quite awhile to complete...

models_best = pd.DataFrame(columns=["RSS", "model"])

tic = time.time()
for i in range(1,9):
    models_best.loc[i] = getBest(i)

toc = time.time()
print("Total elapsed time:", (toc-tic), "seconds.")

# Gets the second element from each row ('model') and pulls out its rsquared attribute
models_best.apply(lambda row: row[1].rsquared, axis=1)

plt.figure(figsize=(20,10))
plt.rcParams.update({'font.size': 18, 'lines.markersize': 10})
```

```python
# Set up a 2x2 grid so we can look at 4 plots at once
plt.subplot(2, 2, 1)

# We will now plot a red dot to indicate the model with the largest adjusted R^2 statistic.
# The argmax() function can be used to identify the location of the maximum point of a vector
plt.plot(models_best["RSS"])
plt.xlabel('# Predictors')
plt.ylabel('RSS')

# We will now plot a red dot to indicate the model with the largest adjusted R^2 statistic.
# The argmax() function can be used to identify the location of the maximum point of a vector

rsquared_adj = models_best.apply(lambda row: row[1].rsquared_adj, axis=1)

plt.subplot(2, 2, 2)
plt.plot(rsquared_adj)
plt.plot(rsquared_adj.argmax(), rsquared_adj.max(), "or")
plt.xlabel('# Predictors')
plt.ylabel('adjusted rsquared')

# We'll do the same for AIC and BIC, this time looking for the models with the SMALLEST statistic
aic = models_best.apply(lambda row: row[1].aic, axis=1)

plt.subplot(2, 2, 3)
plt.plot(aic)
plt.plot(aic.argmin(), aic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('AIC')

bic = models_best.apply(lambda row: row[1].bic, axis=1)

plt.subplot(2, 2, 4)
plt.plot(bic)
plt.plot(bic.argmin(), bic.min(), "or")
plt.xlabel('# Predictors')
plt.ylabel('BIC')

print(models_best.loc[5, "model"].summary())

"""#### VIF check again"""

y=df_num[['building_value']]
x=df_num.iloc[:,:]

from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor

#find design matrix for regression model using 'rating' as response variable
y, X = dmatrices('building_value ~ acreage+land_value+finished_area+bedrooms+half_bath', data=x,
return_type='dataframe')

#create DataFrame to hold VIF values
vif_df = pd.DataFrame()
vif_df['variable'] = X.columns

#calculate VIF for each predictor variable
vif_df['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

#view VIF for each predictor variable
print(vif_df)

vif_df = vif_df[vif_df['variable'] != 'Intercept']
vif_df

plt.figure(figsize=(10, 6))
plt.bar(vif_df['variable'], vif_df['VIF'], color='skyblue')
plt.axhline(y=5, color='red', linestyle='--', label='VIF Threshold (5)')

plt.xlabel('Predictor Variables')
plt.ylabel('VIF Values')
plt.title('VIF Values for Predictor Variables')
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.tight_layout()

plt.show()

"""#### with dummy variables"""

import os as os
from itertools import product
import scipy.stats as ss

df_cat

df_cat_v1 = df_cat.dropna()
df_cat_v1.shape

cat_var1 = ('land_use', 'property_city', 'sold_as_vacant', 'multiple_parcels_involved_in_sale', 'city',
'tax_district', 'foundation_type', 'exterior_wall', 'grade')
cat_var2 = ('land_use', 'property_city', 'sold_as_vacant', 'multiple_parcels_involved_in_sale', 'city',
'tax_district', 'foundation_type', 'exterior_wall', 'grade')

cat_var_prod = list(product(cat_var1,cat_var2, repeat = 1))

result = []

for i in cat_var_prod:
    if i[0] != i[1]:
        result.append((i[0], i[1], list(ss.chi2_contingency(pd.crosstab(
                            df_cat_v1[i[0]], df_cat_v1[i[1]])))[1]))

chi_test_output = pd.DataFrame(result, columns = ['var1', 'var2', 'coeff'])

chi_test_output.pivot(index='var1', columns='var2', values='coeff')



df2

y=df2[['building_value']]
x=pd.concat([df_cat, df2[['acreage', 'land_value', 'finished_area', 'bedrooms', 'half_bath']]], axis=1)
x

#### 'land_use'
lus = pd.get_dummies(x[["land_use"]],drop_first = True)
x = pd.concat([x,lus],axis=1)
### 'property_city'
ppc = pd.get_dummies(x[["property_city"]],drop_first = True)
x = pd.concat([x,ppc],axis=1)
### 'sold_as_vacant'
sav = pd.get_dummies(x[["sold_as_vacant"]],drop_first = True)
x = pd.concat([x,sav],axis=1)
### 'city'
ct =pd.get_dummies(x[["city"]],drop_first = True)
x = pd.concat([x,ct],axis=1)
### 'tax_district'
tdi =pd.get_dummies(x[["tax_district"]],drop_first = True)
x = pd.concat([x,tdi],axis=1)
### 'foundation_type'
fdt = pd.get_dummies(x[["foundation_type"]],drop_first = True)
x = pd.concat([x,fdt],axis=1)
### 'exterior_wall'
exw = pd.get_dummies(x[["exterior_wall"]],drop_first = True)
x = pd.concat([x,exw],axis=1)

x = x.drop(['land_use', 'property_city', 'sold_as_vacant', 'multiple_parcels_involved_in_sale', 'city',
'tax_district', 'foundation_type', 'exterior_wall', 'grade'], axis=1)
```

```
x

y

x = sm.add_constant(x)

result=sm.OLS(y, x).fit()

print(result.summary())

"""#### with normalization"""

y=df_num[['building_value']]
x= df2[['acreage', 'land_value', 'finished_area', 'bedrooms', 'half_bath']]

X=df2[['acreage', 'land_value', 'finished_area', 'bedrooms', 'half_bath']]
X=X.astype('float64')
X

from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler().fit(x)
X_norm = min_max_scaler.transform(x)
X_norm

X_norm_df = pd.DataFrame(X_norm, columns=x.columns)
X_norm_df

df3=df2.copy()

df3

df3=df3.reset_index()

df3

x = X_norm_df
x = pd.concat([x, df3[['sold_as_vacant']]], axis=1)

###Creating a function for converting binary variable
def binary(x):
    if x == "Yes":
        return 1
    else:
        return 0

x["sold_as_vacant"] = x["sold_as_vacant"].apply(binary)

x

y=df3[['building_value']]
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

X2 = sm.add_constant(x_train)
est = sm.OLS(y_train, X2)
est2 = est.fit()
print(est2.summary())



"""### checking assumptions"""

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
# %config InlineBackend.figure_format ='retina'
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.stats.api as sms
sns.set_style('darkgrid')
sns.mpl.rcParams['figure.figsize'] = (30.0, 18.0)
```

```python
def linearity_test(model, y):
    '''
    Function for visually inspecting the assumption of linearity in a linear regression model.
    It plots observed vs. predicted values and residuals vs. predicted values.

    Args:
    * model - fitted OLS model from statsmodels
    * y - observed values
    '''
    fitted_vals = model.predict()
    resids = model.resid

    fig, ax = plt.subplots(1,2)

    sns.regplot(x=fitted_vals, y=y, lowess=True, ax=ax[0], line_kws={'color': 'red'})
    ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
    ax[0].set(xlabel='Predicted', ylabel='Observed')

    sns.regplot(x=fitted_vals, y=resids, lowess=True, ax=ax[1], line_kws={'color': 'red'})
    ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
    ax[1].set(xlabel='Predicted', ylabel='Residuals')

linearity_test(est2, y_train)

"""### Decision tree with all variables"""

df

df6=df.copy()

df6=df6.dropna()
df6=df6.drop(['unnamed:_0', 'parcel_id', 'property_address', 'legal_reference', 'state', 'neighborhood',
'sale_date'], axis = 1)
df6.reset_index(drop=True)
df6

y=df6['sale_price_compared_to_value']

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

y=df6[['sale_price_compared_to_value']]
x=df6.drop(['sale_price_compared_to_value'], axis=1)

x

y

###Creating a function for converting binary variable
def binary(x):
    if x == "Yes":
        return 1
    else:
        return 0

x["sold_as_vacant"] = x["sold_as_vacant"].apply(binary)

###Creating a function for converting binary variable
def binary(y):
    if y == "Over":
        return 1
    else:
        return 0

y["sale_price_compared_to_value"] = y["sale_price_compared_to_value"].apply(binary)

df_cluster = y
```

```python
train_data = x

x_train, x_valid, y_train, y_valid = train_test_split(train_data,df_cluster['sale_price_compared_to_value'],
                                                                      test_size=0.3,
                                                                      random_state=2)

x_valid.info()

"""#### One-Hot Encoding"""

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

x_valid.info()

# Encode categorical variables using one-hot encoding
categorical_features = ['land_use', 'property_city', 'multiple_parcels_involved_in_sale', 'city',
                       'tax_district', 'foundation_type', 'exterior_wall', 'grade']  # Replace with your
categorical variable column names
preprocessor = ColumnTransformer(transformers=[('encoder', OneHotEncoder(handle_unknown='ignore'),
categorical_features)],
                                 remainder='passthrough')
X_train_encoded = preprocessor.fit_transform(x_train)
X_test_encoded = preprocessor.transform(x_valid)

"""#### Decision tree classifier"""

# Create an instance of the decision tree classifier
dt_classifier = DecisionTreeClassifier()

# Fit the model to the training data
dt_classifier.fit(X_train_encoded, y_train)

# Predict the target variable for the test set
y_pred_dt = dt_classifier.predict(X_test_encoded)

from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc_curve, accuracy_score, recall_score,
classification_report, f1_score

from sklearn.metrics import precision_score

# calculating f1 score
dt_f1_score = f1_score(y_valid, y_pred_dt)

dt_f1_score

confusion=confusion_matrix(y_valid, y_pred_dt)

def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):
    '''
    This function will make a pretty plot of an sklearn Confusion Matrix cm using a Seaborn heatmap visualization.

    Arguments
    ---------
    cf:            confusion matrix to be passed in

    group_names:   List of strings that represent the labels row by row to be shown in each square.

    categories:    List of strings containing the categories to be displayed on the x,y axis. Default is 'auto'
```

```
count:         If True, show the raw number in the confusion matrix. Default is True.

normalize:     If True, show the proportions for each category. Default is True.

cbar:          If True, show the color bar. The cbar values are based off the values in the confusion matrix.
               Default is True.

xyticks:       If True, show x and y ticks. Default is True.

xyplotlabels:  If True, show 'True Label' and 'Predicted Label' on the figure. Default is True.

sum_stats:     If True, display summary statistics below the figure. Default is True.

figsize:       Tuple representing the figure size. Default will be the matplotlib rcParams value.

cmap:          Colormap of the values displayed from matplotlib.pyplot.cm. Default is 'Blues'
               See http://matplotlib.org/examples/color/colormaps_reference.html

title:         Title for the heatmap. Default is None.

'''


# CODE TO GENERATE TEXT INSIDE EACH SQUARE
blanks = ['' for i in range(cf.size)]

if group_names and len(group_names)==cf.size:
    group_labels = ["{}\n".format(value) for value in group_names]
else:
    group_labels = blanks

if count:
    group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]
else:
    group_counts = blanks

if percent:
    group_percentages = ["{0:.2%}".format(value) for value in cf.flatten()/np.sum(cf)]
else:
    group_percentages = blanks

box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(group_labels,group_counts,group_percentages)]
box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])


# CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
if sum_stats:
    #Accuracy is sum of diagonal divided by total observations
    accuracy  = np.trace(cf) / float(np.sum(cf))

    #if it is a binary confusion matrix, show some more stats
    if len(cf)==2:
        #Metrics for Binary Confusion Matrices
        precision = cf[1,1] / sum(cf[:,1])
        recall    = cf[1,1] / sum(cf[1,:])
        f1_score  = 2*precision*recall / (precision + recall)
        stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.3f}\nF1 Score={:0.3f}".format(
            accuracy,precision,recall,f1_score)
    else:
        stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
else:
    stats_text = ""


# SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
if figsize==None:
    #Get default figure size if not set
    figsize = plt.rcParams.get('figure.figsize')
```

```python
    if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,yticklabels=categories)

    if xyplotlabels:
        plt.ylabel('True label')
        plt.xlabel('Predicted label' + stats_text)
    else:
        plt.xlabel(stats_text)

    if title:
        plt.title(title)

labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(confusion,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')

# Encode categorical variables using one-hot encoding
categorical_features = ['land_use', 'property_city', 'multiple_parcels_involved_in_sale', 'city',
                       'tax_district', 'foundation_type', 'exterior_wall', 'grade']  # Replace with your
categorical variable column names
preprocessor = ColumnTransformer(transformers=[('encoder', OneHotEncoder(handle_unknown='ignore'),
categorical_features)],
                                 remainder='passthrough')
X_train_encoded = preprocessor.fit_transform(x_train)

# Get the feature names after one-hot encoding
one_hot_encoded_feature_names =
preprocessor.named_transformers_['encoder'].get_feature_names_out(categorical_features)

# Combine one-hot encoded feature names with non-categorical feature names
all_feature_names = list(one_hot_encoded_feature_names) + x_train.columns[len(categorical_features):].tolist()

# Get feature importances from the trained decision tree classifier
feature_importances = dt_classifier.feature_importances_

# Create a DataFrame with feature names and their corresponding importances
feature_importance_df = pd.DataFrame({'Feature': all_feature_names, 'Importance': feature_importances})

# Sort the features by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Visualize the feature importances using a bar chart
plt.figure(figsize=(36, 28))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], color='skyblue')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Decision Tree Feature Importances')
plt.show()



"""### Decision tree with selected features

"""

df

df6=df.copy()

df6=df6.dropna()
df6=df6[['exterior_wall', 'year_built', 'grade', 'sale_year', 'finished_area', 'foundation_type', 'sale_day',
'sale_month','full_bath', 'bedrooms', 'sale_price_compared_to_value']]
df6.reset_index(drop=True)
df6

y=df6['sale_price_compared_to_value']

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

y=df6[['sale_price_compared_to_value']]
x=df6.drop(['sale_price_compared_to_value'], axis=1)

x

y

def binary(y):
    if y == "Over":
        return 1
    else:
        return 0

y["sale_price_compared_to_value"] = y["sale_price_compared_to_value"].apply(binary)

df_cluster = y
train_data = x

x_train, x_valid, y_train, y_valid = train_test_split(train_data,df_cluster['sale_price_compared_to_value'],
                                                                      test_size=0.3,
                                                                      random_state=2)

"""#### One-Hot Encoding"""

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

x_valid.info()

categorical_features = ['exterior_wall', 'grade', 'foundation_type']
preprocessor = ColumnTransformer(transformers=[('encoder', OneHotEncoder(handle_unknown='ignore'),
categorical_features)],
                                 remainder='passthrough')
X_train_encoded = preprocessor.fit_transform(x_train)
X_test_encoded = preprocessor.transform(x_valid)

"""#### Decision tree classifier"""

dt_classifier = DecisionTreeClassifier()

dt_classifier.fit(X_train_encoded, y_train)

y_pred_dt = dt_classifier.predict(X_test_encoded)

from sklearn.metrics import confusion_matrix, precision_recall_curve, auc, roc_curve, accuracy_score, recall_score,
classification_report, f1_score

dt_f1_score = f1_score(y_valid, y_pred_dt)

dt_f1_score

confusion=confusion_matrix(y_valid, y_pred_dt)

labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['0', '1']
make_confusion_matrix(confusion,
                      group_names=labels,
                      categories=categories,
                      cmap='binary')
```

```python
"""#### visualizing decision tree"""

import matplotlib.pyplot as plt
from sklearn import tree

text_representation = tree.export_text(dt_classifier)
print(text_representation)

with open("decistion_tree.log", "w") as fout:
    fout.write(text_representation)

y

from sklearn.tree import plot_tree

plt.figure(figsize=(20,10))
plot_tree(dt_classifier, filled=True,
feature_names=preprocessor.get_feature_names_out(input_features=x_train.columns))
plt.title("Decision Tree Visualization")
plt.show()

dt_classifier2 = DecisionTreeClassifier(max_depth=5)

dt_classifier2.fit(X_train_encoded, y_train)

y_pred_dt2 = dt_classifier2.predict(X_test_encoded)

plt.figure(figsize=(40,20))
plot_tree(dt_classifier2, filled=True,
feature_names=preprocessor.get_feature_names_out(input_features=x_train.columns))
plt.title("Decision Tree Visualization")
plt.show()

"""#### Random Forest Classifier"""

rf_classifier = RandomForestClassifier()

rf_classifier.fit(X_train_encoded, y_train)

y_pred_rf = rf_classifier.predict(X_test_encoded)

rf_f1_score = f1_score(y_valid, y_pred_rf)

rf_f1_score

confusion_matrix(y_valid, y_pred_rf)

rf_classifier = RandomForestClassifier()

rf_classifier.fit(X_train_encoded, y_train)

chosen_tree = rf_classifier.estimators_[0]

plt.figure(figsize=(50, 40))
tree.plot_tree(chosen_tree, feature_names=preprocessor.get_feature_names_out(), filled=True)
plt.show()

rf_classifier = RandomForestClassifier()

rf_classifier.fit(X_train_encoded, y_train)

number_of_trees = rf_classifier.n_estimators
print(f"Number of trees in the random forest: {number_of_trees}")

for idx, est in enumerate(rf_classifier.estimators_):
    print(f"Depth of tree {idx + 1}: {est.tree_.max_depth}")

# Create an instance of the random forest classifier
rf_classifier = RandomForestClassifier()
```

```python
# Fit the model to the training data (assuming you have X_train_encoded and y_train)
rf_classifier.fit(X_train_encoded, y_train)

# Check the number of trees in the random forest
number_of_trees = rf_classifier.n_estimators

# Extract the depth of each tree in the random forest
depths = [est.tree_.max_depth for est in rf_classifier.estimators_]

# Create a DataFrame
df10 = pd.DataFrame({
    'Tree Number': range(1, number_of_trees + 1),
    'Tree Depth': depths
})

df10

# Define the F1-scores
f1_scores = [dt_f1_score, rf_f1_score]

# Define the classifiers
classifiers = ['Decision Tree', 'Random Forest']

# Plot the F1-scores
plt.bar(classifiers, f1_scores)
plt.xlabel('Classifier')
plt.ylabel('F1-score')
plt.title('F1-score Comparison')
plt.show()

# Create a DataFrame with the model names and corresponding F1-scores
compare_1 = pd.DataFrame({'Model': ['Decision Tree', 'Random Forest'],
                          'F1_Score': [dt_f1_score, rf_f1_score]})

# Set the figure size
plt.figure(figsize=(18, 5))

# Create the point plot
sns.pointplot(x='Model', y='F1_Score', data=compare_1)

# Set the title, x-label, and y-label
plt.title('F1-score Comparison')
plt.xlabel('Model')
plt.ylabel('F1-score')

# Display the plot
plt.show()

# Calculate the confusion matrices
cm_dt = confusion_matrix(y_valid, y_pred_dt)
cm_rf = confusion_matrix(y_valid, y_pred_rf)

# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plot the decision tree confusion matrix
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[0])
axes[0].set_title('Decision Tree Confusion Matrix')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')

# Plot the random forest confusion matrix
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[1])
axes[1].set_title('Random Forest Confusion Matrix')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')

# Adjust the spacing between subplots
plt.tight_layout()
```

```python
# Show the plot
plt.show()

"""### XGBoost"""

import xgboost as xgb
from sklearn.metrics import f1_score

X_train_encoded

dtrain = xgb.DMatrix(X_train_encoded, label=y_train)
dvalid = xgb.DMatrix(X_test_encoded, label=y_valid)

params = {
    'objective': 'binary:logistic',
    'max_depth': 3,
    'learning_rate': 0.1,
    'n_estimators': 100,
    'eval_metric': 'logloss'
}

num_round = 100  # Number of boosting rounds
bst = xgb.train(params, dtrain, num_round, evals=[(dvalid, 'eval')], early_stopping_rounds=10, verbose_eval=True)

# Predict using the trained model
y_pred_xgb = bst.predict(dvalid)

# Convert probabilities to binary predictions
y_pred_binary = [1 if pred > 0.5 else 0 for pred in y_pred_xgb]

# Calculate F1 score for evaluation
xgb_f1_score = f1_score(y_valid, y_pred_binary)

# Print F1 score
print("XGBoost F1 Score:", xgb_f1_score)

import xgboost as xgb
from xgboost import plot_importance
import matplotlib.pyplot as plt

# Plotting feature importance
plot_importance(bst)
plt.show()

ohe_feature_names = preprocessor.named_transformers_['encoder'].get_feature_names_out(categorical_features)

# Combine the one-hot encoded names with any non-transformed features
all_feature_names = np.concatenate([ohe_feature_names,
                                    np.array([col for col in x_train.columns if col not in categorical_features])])

# Map default feature names to original feature names
mapping = {f'f{i}': feature for i, feature in enumerate(all_feature_names)}

# Print out the original names for the features you're interested in
features_of_interest = ['f23', 'f24', 'f25', 'f26', 'f27']
for f in features_of_interest:
    print(f"{f} corresponds to {mapping[f]}")

ohe_feature_names = preprocessor.named_transformers_['encoder'].get_feature_names_out(categorical_features)

# Combine the one-hot encoded names with any non-transformed features
all_feature_names = np.concatenate([ohe_feature_names,
                                    np.array([col for col in x_train.columns if col not in categorical_features])])

# Map default feature names to original feature names
mapping = {f'f{i}': feature for i, feature in enumerate(all_feature_names)}

# Extract the original names for the features you're interested in
features_of_interest = ['f23', 'f24', 'f25', 'f27', 'f26']
```

```python
original_names = [mapping[f] for f in features_of_interest]

# Create a DataFrame to display in table format
df11 = pd.DataFrame({
    'Default Feature Name': features_of_interest,
    'Original Feature Name': original_names
})

df11

# Plotting the first tree
xgb.plot_tree(bst, num_trees=0)
plt.show()

evals_result = {}
bst = xgb.train(params, dtrain, num_round, evals=[(dvalid, 'eval')], early_stopping_rounds=10,
evals_result=evals_result, verbose_eval=True)

# Extracting the evaluation results
epochs = len(evals_result['eval']['logloss'])
x_axis = range(0, epochs)

# Plotting log loss
fig, ax = plt.subplots()
ax.plot(x_axis, evals_result['eval']['logloss'], label='Validation Log Loss')
ax.legend()
plt.xlabel('Boosting Round')
plt.ylabel('Log Loss')
plt.title('XGBoost Log Loss')
plt.show()

"""#### with train and validation"""

evals_result = {}
bst = xgb.train(params, dtrain, num_round, evals=[(dtrain, 'train'), (dvalid, 'eval')],
                early_stopping_rounds=10, evals_result=evals_result, verbose_eval=True)

epochs = len(evals_result['eval']['logloss'])
x_axis = range(0, epochs)

fig, ax = plt.subplots()
ax.plot(x_axis, evals_result['train']['logloss'], label='Train Log Loss')
ax.plot(x_axis, evals_result['eval']['logloss'], label='Validation Log Loss')
ax.legend()
plt.xlabel('Boosting Round')
plt.ylabel('Log Loss')
plt.title('XGBoost Log Loss')
plt.show()

"""#### visualization of results"""

compare_2 = compare_1.append({'Model': 'XGBoost', 'F1_Score': xgb_f1_score}, ignore_index=True)

# Set the figure size
plt.figure(figsize=(18, 5))

# Create the bar plot
sns.pointplot(x='Model', y='F1_Score', data=compare_2)

# Set the title, x-label, and y-label
plt.title('F1-score Comparison')
plt.xlabel('Model')
plt.ylabel('F1-score')

# Display the plot
plt.show()

"""#### multiple benchmark"""

# calculating accuracy
```

```python
dt_accuracy = accuracy_score(y_valid, y_pred_dt)

rf_accuracy = accuracy_score(y_valid, y_pred_rf)

xgb_accuracy = accuracy_score(y_valid, y_pred_binary)

# Create a DataFrame with the model names and corresponding accuracy
compare_3 = pd.DataFrame({'Model': ['Decision Tree', 'Random Forest'],
                          'Accuracy': [dt_accuracy, rf_accuracy]})

compare_4 = compare_3.append({'Model': 'XGBoost', 'Accuracy': xgb_accuracy}, ignore_index=True)

# Set the figure size
plt.figure(figsize=(18, 5))

# Create the bar plot
sns.pointplot(x='Model', y='Accuracy', data=compare_4)

# Set the title, x-label, and y-label
plt.title('Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')

# Display the plot
plt.show()

# calculating recall
dt_recall = recall_score(y_valid, y_pred_dt)

rf_recall = recall_score(y_valid, y_pred_rf)

xgb_recall = recall_score(y_valid, y_pred_binary)

# Create a DataFrame with the model names and corresponding accuracy
compare_5 = pd.DataFrame({'Model': ['Decision Tree', 'Random Forest'],
                          'Recall': [dt_recall, rf_recall]})

compare_6 = compare_5.append({'Model': 'XGBoost', 'Recall': xgb_recall}, ignore_index=True)

# Set the figure size
plt.figure(figsize=(18, 5))

# Create the bar plot
sns.pointplot(x='Model', y='Recall', data=compare_6)

# Set the title, x-label, and y-label
plt.title('Recall Comparison')
plt.xlabel('Model')
plt.ylabel('Recall')

# Display the plot
plt.show()

# calculating recall
dt_precision = precision_score(y_valid, y_pred_dt)

rf_precision = precision_score(y_valid, y_pred_rf)

xgb_precision = precision_score(y_valid, y_pred_binary)

# Create a DataFrame with the model names and corresponding accuracy
compare_7 = pd.DataFrame({'Model': ['Decision Tree', 'Random Forest'],
                          'Precision': [dt_precision, rf_precision]})

compare_8 = compare_7.append({'Model': 'XGBoost', 'Precision': xgb_precision}, ignore_index=True)

# Set the figure size
plt.figure(figsize=(18, 5))

# Create the bar plot
```

```python
sns.pointplot(x='Model', y='Precision', data=compare_8)

# Set the title, x-label, and y-label
plt.title('Precision Comparison')
plt.xlabel('Model')
plt.ylabel('Precision')

# Display the plot
plt.show()

"""#### Data Visualization"""

import plotly.express as px

df_v = df2[['land_value', 'finished_area', 'acreage', 'building_value']]

fig = px.box(df_v.melt(), y="value", facet_col="variable",facet_col_wrap=2, boxmode="overlay",
color="variable",height=1000, width=900)
fig.update_yaxes(matches=None)

for i in range(len(fig["data"])):
    yaxis_name = 'yaxis' if i == 0 else f'yaxis{i + 1}'
    fig.layout[yaxis_name].showticklabels = True

fig.update_layout(showlegend=False)
fig.update_xaxes(showline=True, linewidth=2, linecolor='grey')
fig.update_yaxes(showline=True, linewidth=2, linecolor='grey')

fig.show()

plt.figure(figsize=(10, 5))
sns.histplot(x=df2['land_value'], kde=True)

plt.title("Distribution of land_value")

plt.figure(figsize=(10, 5))
sns.histplot(x=df2['finished_area'], kde=True)

plt.title("Distribution of finished_area")

plt.figure(figsize=(10, 5))
sns.histplot(x=df2['acreage'], kde=True)

plt.title("Distribution of acreage")

plt.figure(figsize=(10, 5))
sns.histplot(x=df2['building_value'], kde=True)

plt.title("Distribution of building_value")

counts = df1["sale_price_compared_to_value"].value_counts()

# Extract labels and sizes for the pie chart
labels = counts.index.tolist()
sizes = counts.values

# Create a pie chart
plt.figure(figsize=(6, 6))  # Optional: Set the figure size
plt.pie(sizes, labels=[f'{label} ({count})' for label, count in zip(labels, sizes)], autopct='%.1f%%',
startangle=140)

# Display the pie chart
plt.title('pie chart of sale_price_compared_to_value')
plt.show()
```