NYC PROPERTY SALES RECOMMENDATIONS

ALLA SAI MANI TEJA REDDY(50496173)(saimanit)

, YALLA YASHWANTH(50496403)(yaalla)

FALL 2023

PROBLEM STATEMENT:

The growing real estate presents a challenge for the buyers to buy the property based on their match preference. We propose a NYC rolling sales recommendation to address this problem. As well as for the agents to go upon the customer's area of interest and provide the necessary resources. The goal is to provide accurate and relevant recommendations.

BACKGROUND:

The dataset under consideration is a comprehensive record of property transactions within the New York City real estate market over a 12-month period. It offers valuable insights about the sale of building units, including their location, address, sale price, sale date, and other key attributes. To effectively utilize this dataset for analysis purposes, it is essential to have a clear understanding of the context and the dataset's specific characteristics. BY using feature engineering, we will get the age of each building. which we will use in further model building. This analysis project of NY Property sales can be improved by engineering more features and replacing the outliers by mean and median values when building the models. This could help the property-sales companies and the customers to have basic idea of the property value in particular locations of NY.

DATA CLEANING STEPS:

1. **DATA SUMMARY**:

```
In [110]: import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns
          import numpy as np
          from scipy import stats

In [111]: file_path = 'nyc-rolling-sales.csv'
          data = pd.read_csv(file_path)
```

we are importing the data.

2. **DATA:**

```
In [153]: data.head()
```

Out[153]:

	Unnamed: 0	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDING CLASS AT PRESENT	ADDRESS	...	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	S
0	4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6		C2	153 AVENUE B	...	5	0	5	
1	5	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	26		C7	234 EAST 4TH STREET	...	28	3	31	
2	6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	39		C7	197 EAST 3RD STREET	...	16	1	17	
3	7	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21		C4	154 EAST 7TH STREET	...	10	0	10	
4	8	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55		C2	301 EAST 10TH STREET	...	6	0	6	

5 rows × 22 columns

The data set is displayed with the head of the data. The following data set contains 22 columns and 84549 rows.

3. **DATA TYPE CHECK:**

```
In [401]: data.dtypes

Out[401]: Unnamed: 0                         int64
          BOROUGH                            int64
          NEIGHBORHOOD                      object
          BUILDING CLASS CATEGORY           object
          TAX CLASS AT PRESENT              object
          BLOCK                              int64
          LOT                                int64
          EASE-MENT                        float64
          BUILDING CLASS AT PRESENT         object
          ADDRESS                           object
          APARTMENT NUMBER                  object
          ZIP CODE                           int64
          RESIDENTIAL UNITS                  int64
          COMMERCIAL UNITS                   int64
          TOTAL UNITS                        int64
          LAND SQUARE FEET                  object
          GROSS SQUARE FEET                 object
          YEAR BUILT                         int64
          TAX CLASS AT TIME OF SALE          int64
          BUILDING CLASS AT TIME OF SALE    object
          SALE PRICE                        object
          SALE DATE                         object
          BUILDING AGE                       int64
          SALE_YEAR                          int64
          SALE_MONTH                         int64
          dtype: object
```

4. **DUPLICATE CHECKS:**

```
In [231]: # Check for and remove duplicate rows
          data.drop_duplicates(inplace=True)

In [233]: data.head()

Out[233]:
```

	Unnamed: 0	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDING CLASS AT PRESENT	ADDRESS	...	COMMERCIAL UNITS	TOTAL UNITS	LAND SQUARE FEET	GRO SQUA FE
0	4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6	NaN	C2	153 AVENUE B	...	0	5	1633	6
1	5	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	26	NaN	C7	234 EAST 4TH STREET	...	3	31	4616	18
2	6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	39	NaN	C7	197 EAST 3RD STREET	...	1	17	2212	7
3	7	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	NaN	C4	154 EAST 7TH STREET	...	0	10	2272	6
4	8	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55	NaN	C2	301 EAST 10TH STREET	...	0	6	2369	4

5 rows × 23 columns

We checked whether we have any duplicate values.There are no duplicates in our data which is also good sign.

5.**HANDLING MISSING DATA:**

There are lot of missing values in our data, we replace the missing values with nan and update the data. We can see the number of missing data values in our data area replaced with nan.

```
In [67]: data.replace(' ',np.nan, inplace=True)
```

```
In [68]: data.head()
```

Out[68]:

	Unnamed: 0	BOROUGH	NEIGHBORHOOD	BUILDING CLASS CATEGORY	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDING CLASS AT PRESENT	ADDRESS	...	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	S
0	4	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	392	6	NaN	C2	153 AVENUE B	...	5	0	5	
1	5	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	26	NaN	C7	234 EAST 4TH STREET	...	28	3	31	
2	6	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2	399	39	NaN	C7	197 EAST 3RD STREET	...	16	1	17	
3	7	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2B	402	21	NaN	C4	154 EAST 7TH STREET	...	10	0	10	
4	8	1	ALPHABET CITY	07 RENTALS - WALKUP APARTMENTS	2A	404	55	NaN	C2	301 EAST 10TH STREET	...	6	0	6	

5 rows × 22 columns

6.DATA MISSING COUNT:

```
In [69]: round(data.isna().sum() /len(data) *100,2)
```

```
Out[69]: Unnamed: 0                       0.00
BOROUGH                          0.00
NEIGHBORHOOD                     0.00
BUILDING CLASS CATEGORY          0.00
TAX CLASS AT PRESENT             0.87
BLOCK                            0.00
LOT                              0.00
EASE-MENT                      100.00
BUILDING CLASS AT PRESENT        0.87
ADDRESS                          0.00
APARTMENT NUMBER                77.47
ZIP CODE                         0.00
RESIDENTIAL UNITS                0.00
COMMERCIAL UNITS                 0.00
TOTAL UNITS                      0.00
LAND SQUARE FEET                 0.00
GROSS SQUARE FEET                0.00
YEAR BUILT                       0.00
TAX CLASS AT TIME OF SALE        0.00
BUILDING CLASS AT TIME OF SALE   0.00
SALE PRICE                       0.00
SALE DATE                        0.00
dtype: float64
```

There are almost 100% missing values on EASE-MENT so we remove that as it’s not necessary.

7. CONVERTING TO LOWER CASE LETTERS:

Converting to lower case letters.

```
In [155]: for column in data1.columns:
              if data1[column].dtype == 'O':  # Check if the column contains objects (strings)
                  data1[column] = data1[column].str.lower()

          # Save the modified DataFrame back to a CSV file
          data.to_csv('lowercase_data.csv',index=False)
```

8.DATA DESCRIBE before removing the outliers:

```
In [71]: data.describe()
```

Out[71]:

	Unnamed: 0	BOROUGH	BLOCK	LOT	EASE-MENT	ZIP CODE	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	YEAR BUILT	TAX CLASS AT TIME OF SALE
count	84548.000000	84548.000000	84548.000000	84548.000000	0.0	84548.000000	84548.000000	84548.000000	84548.000000	84548.000000	84548.000000
mean	10344.359878	2.998758	4237.218976	376.224015	NaN	10731.991614	2.025264	0.193559	2.249184	1789.322976	1.657485
std	7151.779436	1.289790	3568.263407	658.136814	NaN	1290.879147	16.721037	8.713183	18.972584	537.344993	0.819341
min	4.000000	1.000000	1.000000	1.000000	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	4231.000000	2.000000	1322.750000	22.000000	NaN	10305.000000	0.000000	0.000000	1.000000	1920.000000	1.000000
50%	8942.000000	3.000000	3311.000000	50.000000	NaN	11209.000000	1.000000	0.000000	1.000000	1940.000000	2.000000
75%	15987.250000	4.000000	6281.000000	1001.000000	NaN	11357.000000	2.000000	0.000000	2.000000	1965.000000	2.000000
max	26739.000000	5.000000	16322.000000	9106.000000	NaN	11694.000000	1844.000000	2261.000000	2261.000000	2017.000000	4.000000

We are plotting data statistics such as count, mean, min, max, etc.. before removing the outliers.

That makes us understand the data very well.

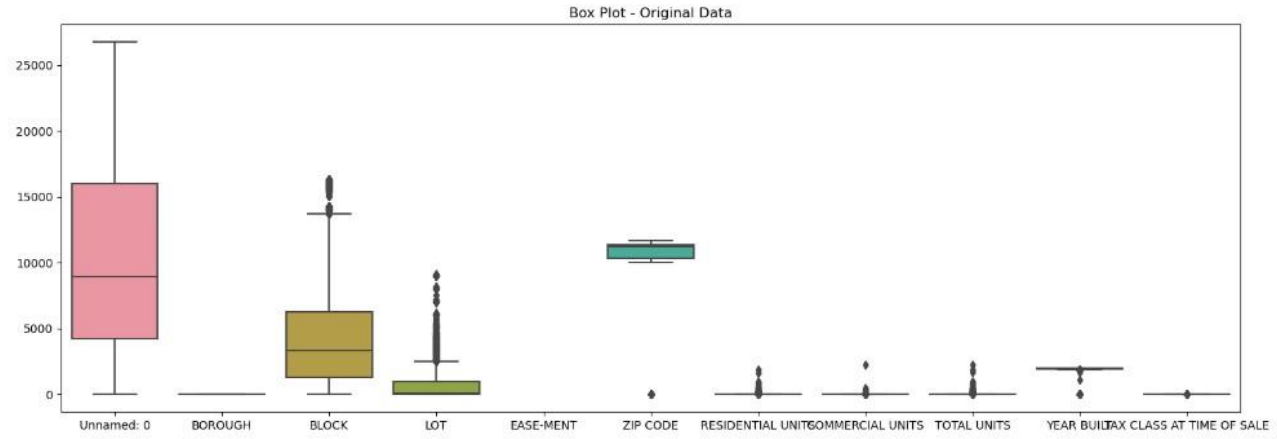
9. OUTLIERS DETECTION:

Outliers are one of the most common issue that alter the data while processing the data.

We can see many number of outliers in our data, so we remove every outlier.

```
In [76]: plt.figure(figsize=(40, 6))
         plt.subplot(1, 2, 1)
         sns.boxplot(data=numeric_columns, orient='vertical')
         plt.title('Box Plot - Original Data')
```

```
Out[76]: Text(0.5, 1.0, 'Box Plot - Original Data')
```

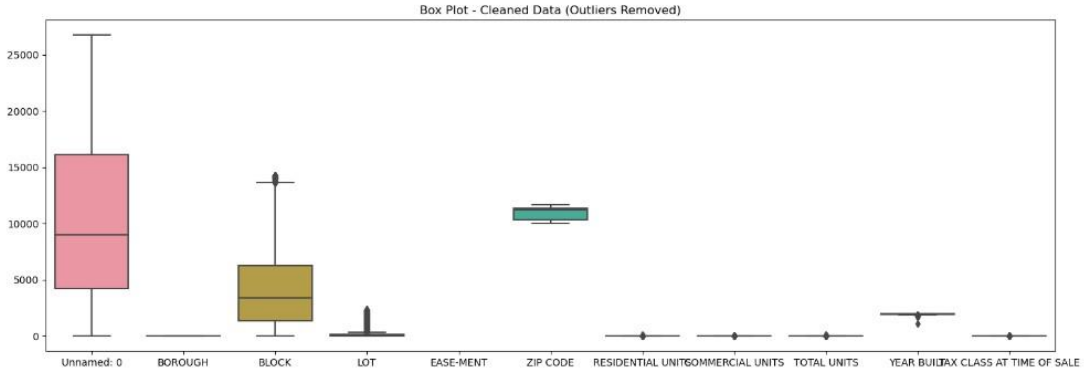


10.REMOVING OUTLIERS:

```
In [77]: def remove_outliers_zscore(data, threshold=3):
             z_scores = np.abs(stats.zscore(data))
             outlier_mask = (z_scores > threshold).any(axis=1)
             cleaned_data = data[~outlier_mask]
             return cleaned_data

         threshold = 3

         numeric_columns = data.select_dtypes(include=['int64', 'float64'])
         cleaned_data = remove_outliers_zscore(numeric_columns, threshold=threshold)

         num_removed_outliers = len(data) - len(cleaned_data)
         print(f"Number of removed outliers: {num_removed_outliers}")

         cleaned_data.to_csv('cleaned_data.csv',index=False)
```

```
Number of removed outliers: 8709
```

```
In [78]:
         plt.figure(figsize=(40, 6))
         plt.subplot(1, 2, 2)
         sns.boxplot(data=cleaned_data, orient='vertical')
         plt.title('Box Plot - Cleaned Data (Outliers Removed)')
         plt.show()
```



11.DATA DESCRIBE after removing the outliers:

We are plotting the data statistics such as count, mean, min, max, etc.. after removing the outliers. We can see there are lot of changes in the count, mean, min, max etc… which indicates that the outliers are removed. We can clearly find the range of min , max values and there mean.

```
In [146]: data1.describe()
```

Out[146]:

	Unnamed: 0	BOROUGH	BLOCK	LOT	EASE-MENT	ZIP CODE	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	YEAR BUILT	TAX CLASS AT TIME OF SALE
count	75839.000000	75839.000000	75839.000000	75839.000000	0.0	75839.000000	75839.000000	75839.000000	75839.000000	75839.000000	75839.000000
mean	10391.623663	3.037646	4194.486860	295.756972	NaN	10863.061762	1.525943	0.114466	1.666082	1949.798204	1.607432
std	7194.547128	1.280755	3364.373901	493.261538	NaN	554.509820	3.088062	0.685618	3.209806	34.283908	0.778533
min	4.000000	1.000000	1.000000	1.000000	NaN	10001.000000	0.000000	0.000000	0.000000	1111.000000	1.000000
25%	4235.500000	2.000000	1345.000000	20.000000	NaN	10306.000000	0.000000	0.000000	1.000000	1925.000000	1.000000
50%	8992.000000	3.000000	3378.000000	46.000000	NaN	11210.000000	1.000000	0.000000	1.000000	1945.000000	1.000000
75%	16113.000000	4.000000	6269.000000	153.000000	NaN	11357.000000	2.000000	0.000000	2.000000	1970.000000	2.000000
max	26739.000000	5.000000	14255.000000	2345.000000	NaN	11693.000000	52.000000	26.000000	57.000000	2017.000000	4.000000

Exploratory Data Analysis (EDA) Steps:

1.FEATURE SELECTION:

So we are selecting the features that are required for the EDA process.

2. FEATURE ENGINEERING:

We come with new features such as building age and sale_year and sale date, Total square feet. So these features can be useful for the further process to show the relation for sale_price etc…

```
In [215]: data1.describe()
```

Out[215]:

	Unnamed: 0	BOROUGH	BLOCK	LOT	EASE-MENT	ZIP CODE	RESIDENTIAL UNITS	COMMERCIAL UNITS	TOTAL UNITS	YEAR BUILT	TAX CLASS AT TIME OF SALE
count	75839.000000	75839.000000	75839.000000	75839.000000	0.0	75839.000000	75839.000000	75839.000000	75839.000000	75839.000000	75839.000000
mean	10391.623663	3.037646	4194.486860	295.756972	NaN	10863.061762	1.525943	0.114466	1.666082	1949.798204	1.607432
std	7194.547128	1.280755	3364.373901	493.261538	NaN	554.509820	3.088062	0.685618	3.209806	34.283908	0.778533
min	4.000000	1.000000	1.000000	1.000000	NaN	10001.000000	0.000000	0.000000	0.000000	1111.000000	1.000000
25%	4235.500000	2.000000	1345.000000	20.000000	NaN	10306.000000	0.000000	0.000000	1.000000	1925.000000	1.000000
50%	8992.000000	3.000000	3378.000000	46.000000	NaN	11210.000000	1.000000	0.000000	1.000000	1945.000000	1.000000
75%	16113.000000	4.000000	6269.000000	153.000000	NaN	11357.000000	2.000000	0.000000	2.000000	1970.000000	2.000000
max	26739.000000	5.000000	14255.000000	2345.000000	NaN	11693.000000	52.000000	26.000000	57.000000	2017.000000	4.000000

```
In [156]: data['BUILDING AGE'] = data['YEAR BUILT'].apply(lambda year_built: 2023 - year_built)
```

```
In [157]: data.head()
```

Out[157]:

	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDING CLASS AT PRESENT	ADDRESS	...	COMMERCIAL UNITS	TOTAL UNITS	LAND SQUARE FEET	GROSS SQUARE FEET	YEAR BUILT	TAX CLASS AT TIME OF SALE	BUILDING CLASS AT TIME OF SALE	SALE PRICE	SALE DATE	BUILDING AGE
	2A	392	6	NaN	C2	153 AVENUE B	...	0	5	1633	6440	1900	2	C2	6625000	2017-07-19 00:00:00	123
	2	399	26	NaN	C7	234 EAST 4TH STREET	...	3	31	4616	18690	1900	2	C7	-	2016-12-14 00:00:00	123
	2	399	39	NaN	C7	197 EAST 3RD STREET	...	1	17	2212	7803	1900	2	C7	-	2016-12-09 00:00:00	123
	2B	402	21	NaN	C4	154 EAST 7TH STREET	...	0	10	2272	6794	1913	2	C4	3936272	2016-09-23 00:00:00	110
	2A	404	55	NaN	C2	301 EAST 10TH STREET	...	0	6	2369	4615	1900	2	C2	8000000	2016-11-17 00:00:00	123

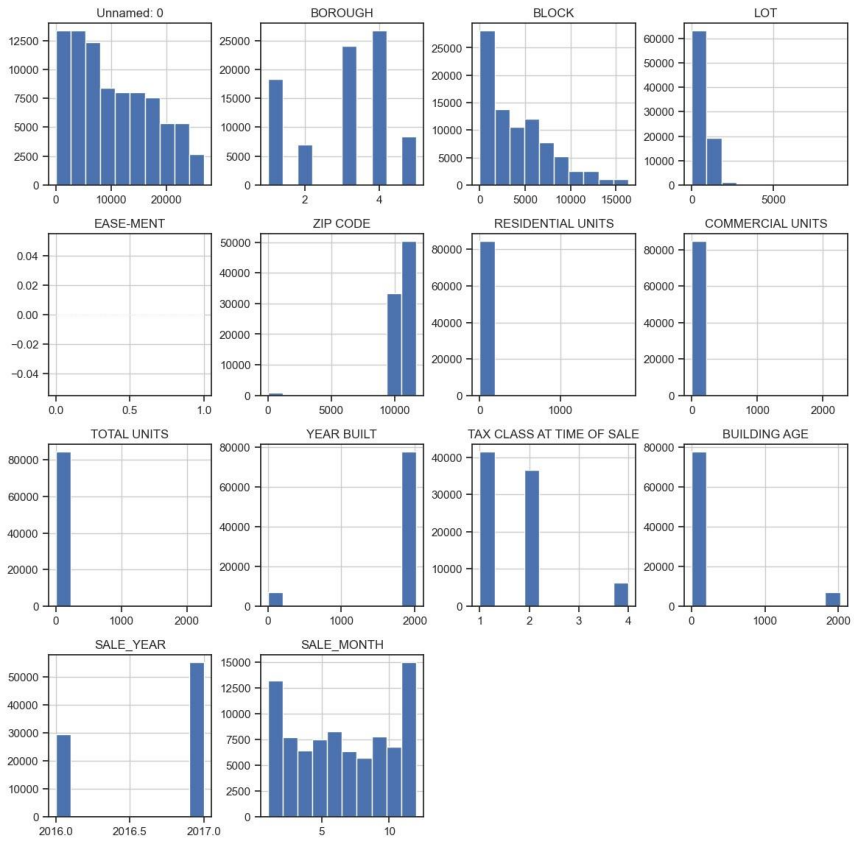
```
In [447]: data["TOTAL SQUARE FEET"] = data["LAND SQUARE FEET"] + data["GROSS SQUARE FEET"]
```

```
In [448]: data.head()
```

Out[448]:

	TAX CLASS AT PRESENT	BLOCK	LOT	EASE-MENT	BUILDING CLASS AT PRESENT	ADDRESS	...	GROSS SQUARE FEET	YEAR BUILT	TAX CLASS AT TIME OF SALE	BUILDING CLASS AT TIME OF SALE	SALE PRICE	SALE DATE	BUILDING AGE	SALE_YEAR	SALE_MONTH	TOTAL SQUARE FEET
	2A	392	6	NaN	C2	153 AVENUE B	...	6440	1900	2	C2	6625000	2017-07-19 00:00:00	123	2017	7	16336440
	2	399	26	NaN	C7	234 EAST 4TH STREET	...	18690	1900	2	C7	-	2016-12-14 00:00:00	123	2016	12	461618690
	2	399	39	NaN	C7	197 EAST 3RD STREET	...	7803	1900	2	C7	-	2016-12-09 00:00:00	123	2016	12	22127803
	2B	402	21	NaN	C4	154 EAST 7TH STREET	...	6794	1913	2	C4	3936272	2016-09-23 00:00:00	110	2016	9	22726794
	2A	404	55	NaN	C2	301 EAST 10TH STREET	...	4615	1900	2	C2	8000000	2016-11-17 00:00:00	123	2016	11	23694615

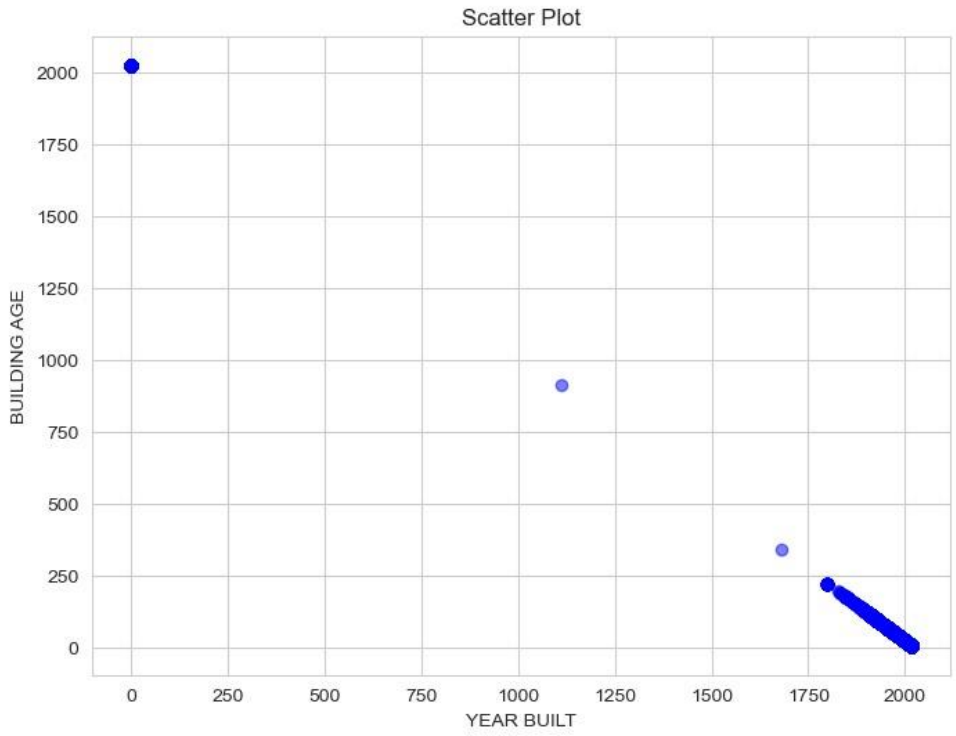
HISTOGRAM PLOT FOR EVERY FEATURE:



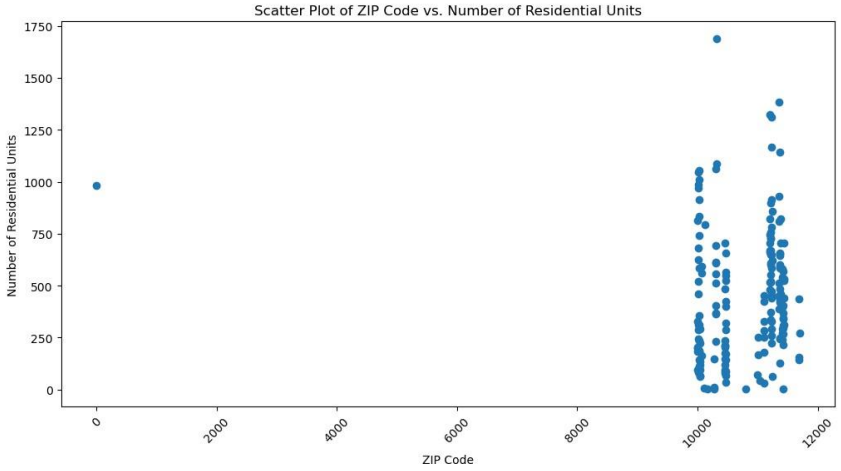
So for each and every feature we are selecting we are plotting the histogram.

- From the above plots we can justify the range and of each and every feature, and can justify the max and min values.
- We mostly require year built and we can clearly see that all are from over the year 2000. o And zip code are even from range (10000-12000).

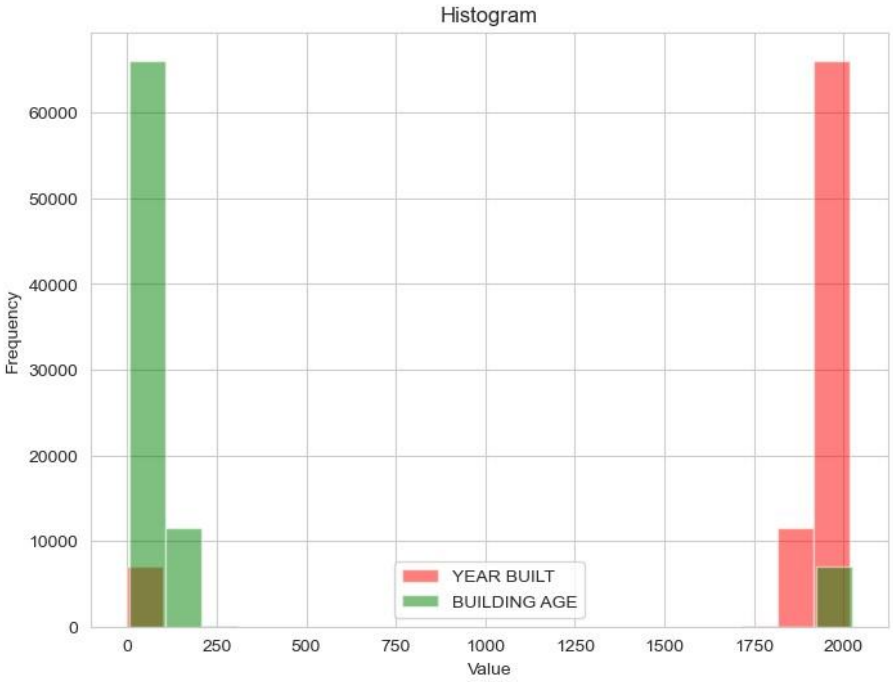
SCATTER PLOT:



The above scatter plot shows the years built of the buildings which re around from 1750 to over 2000 years with the building age on the y-axis clearly. This helps us to clearly identify that the buildings ranges. o Scatter plot for number of residential units and zip code(range from 10000-12000).

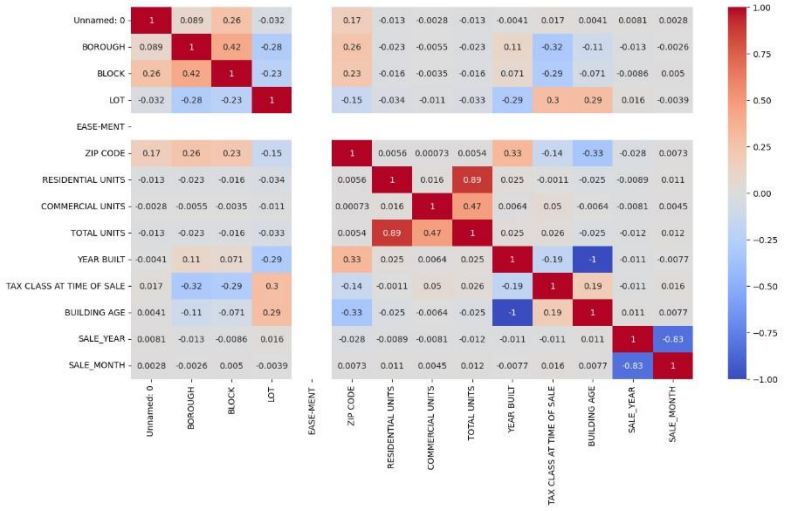


HISTOGRAM:



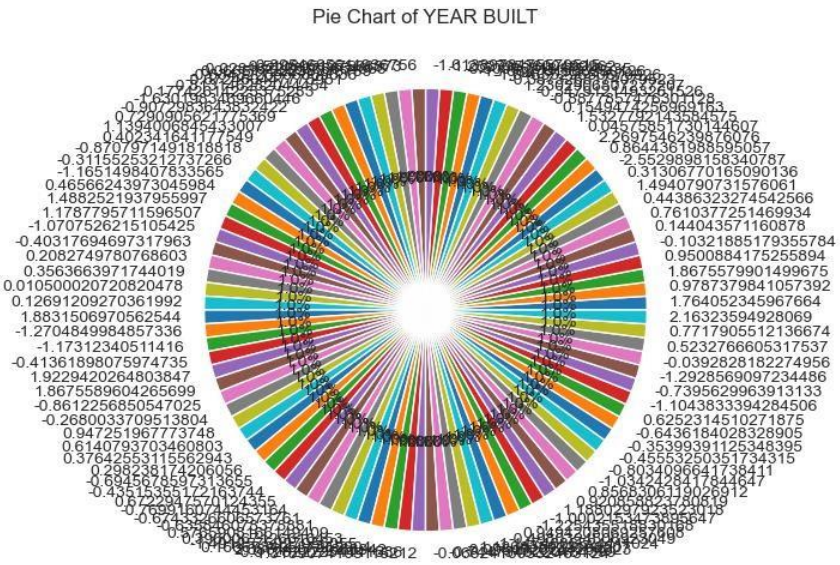
The histogram plot shows the range of year built and building age.

HEAT MAP:



From the above heat map, we say the highly correlated features, Which are very similar to each other.

PIE CHART:



Plotting the pie-chart to ensure the range of building year built. And from chart we can find year model of the building.

DISTRIBUTION PLOT:

The distribution chart clearly shows the Building ages , where can clearly find some of them are built in the same year.

Regression Model Analysis Report:

Overview:

This report presents a comprehensive analysis of various regression models applied to a dataset for predicting property sale prices. The dataset contains several features including residential units, commercial units, total units, land square feet, gross square feet, year built, tax class at the time of sale, building age, and others.

Models Evaluated

The following regression models were chosen for evaluation:

- Linear Regression
- Lasso Regression
- Ridge Regression
- Random Forest Regression
- Decision Tree Regression
- Polynomial Regression

Rationale for Model Selection

- Linear Regression was chosen as the baseline model for its simplicity and interpretability.
- Lasso Regression extends Linear Regression with L1 regularization, yielding sparse models where irrelevant features are automatically ignored (feature selection).
- Ridge Regression also extends Linear Regression but with L2 regularization, which can handle multicollinearity better by shrinking the coefficients.
- Random Forest Regression is an ensemble method that can model non-linear relationships and interactions between features without requiring transformation.
- Decision Tree Regression is a non-linear model is easy to understand. It's good for capturing complex patterns but can overfit easily.
- Polynomial Regression was used to capture non-linearity by considering polynomial features, which allows for a more flexible model.

Reason for Choosing These Models:

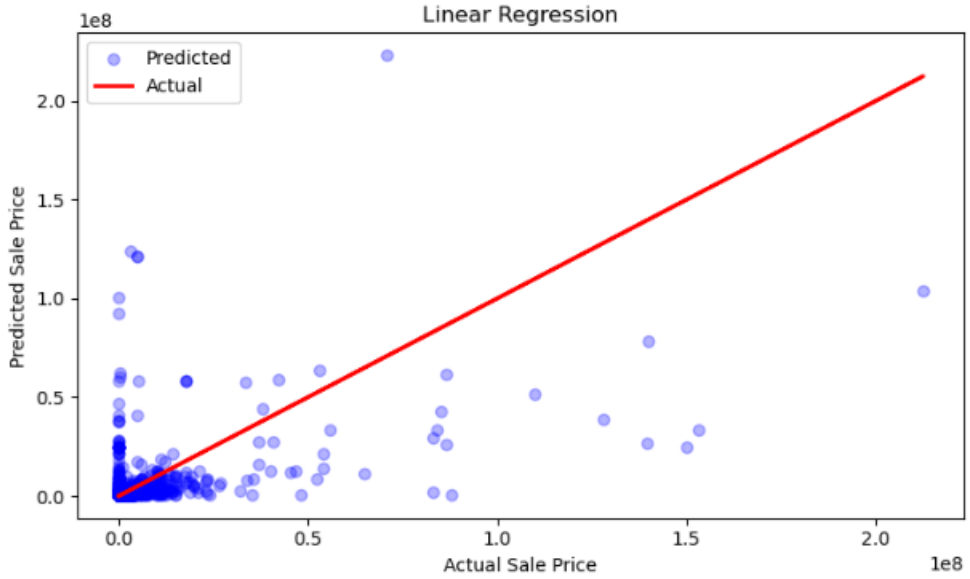
- These models cover a spectrum from simple to complex, which allows us to understand the trade-off between bias and variance.
- The regularization in Lasso and Ridge can provide insights into the importance of different features.
- Random Forest and Decision Tree provide robustness against outliers and can model complex hierarchical relationships.
- Polynomial Regression can model non-linear relationships that other linear models might miss.

Linear Regression:

Linear Regression is a fundamental statistical approach for modeling the relationship between a scalar response and one or more explanatory variables. It assumes a linear relationship between the inputs and the target variable.

Training and Testing: The model was trained using the 'GROSS SQUARE FEET' as the explanatory variable to predict 'SALE PRICE'. The dataset was split into a training set (80%) and a testing set (20%).

Output: The model obtained an MSE of 2.70 x 10^13 and an R^2 of 0.056, suggesting it captured a small fraction of the variance in the sale prices .



This model is the simplest, using a straight-line (linear) approach to predict the sale prices based on the GROSS SQUARE FEET feature.

The blue points should ideally fall on the red line if predictions were perfect.

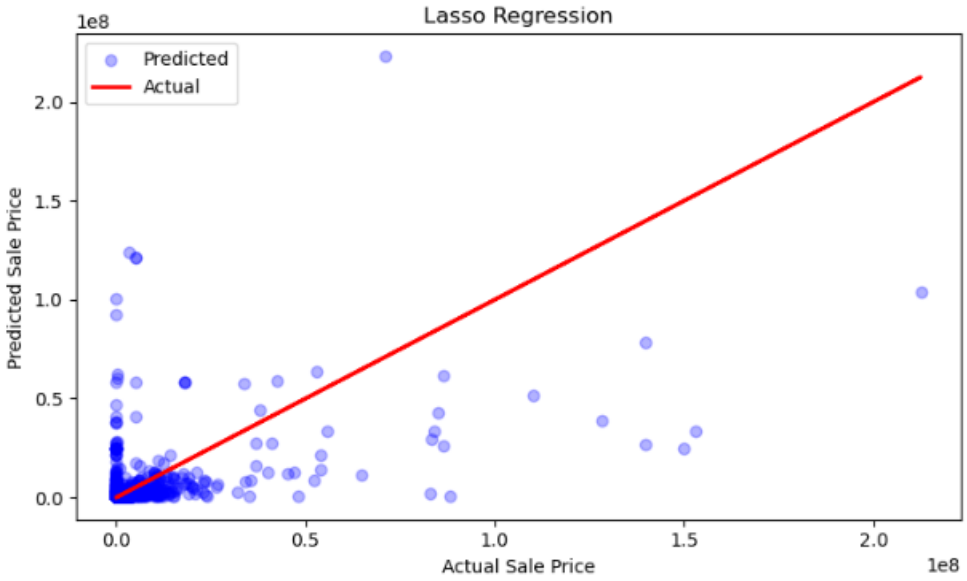
The scatter of points around the red line indicates variance from the actual values, with the model underestimating for higher values and overestimating for lower values.

Lasso Regression:

Lasso Regression extends Linear Regression by adding a penalty term to the loss function, which encourages simple, sparse models (i.e., models with fewer parameters).

Training and Testing: The same split was used as in Linear Regression, and the 'GROSS SQUARE FEET' was used to predict 'SALE PRICE'.

Output: The MSE was nearly identical to Linear Regression, and the R^2 was also 0.056, indicating that adding the L1 penalty did not significantly change the performance.



Lasso is similar to linear regression but includes a penalty term that can shrink less important feature coefficients to zero, effectively performing feature selection.

The plot for Lasso is very similar to linear regression, indicating that the penalty term didn't significantly change the predictions for this particular dataset.

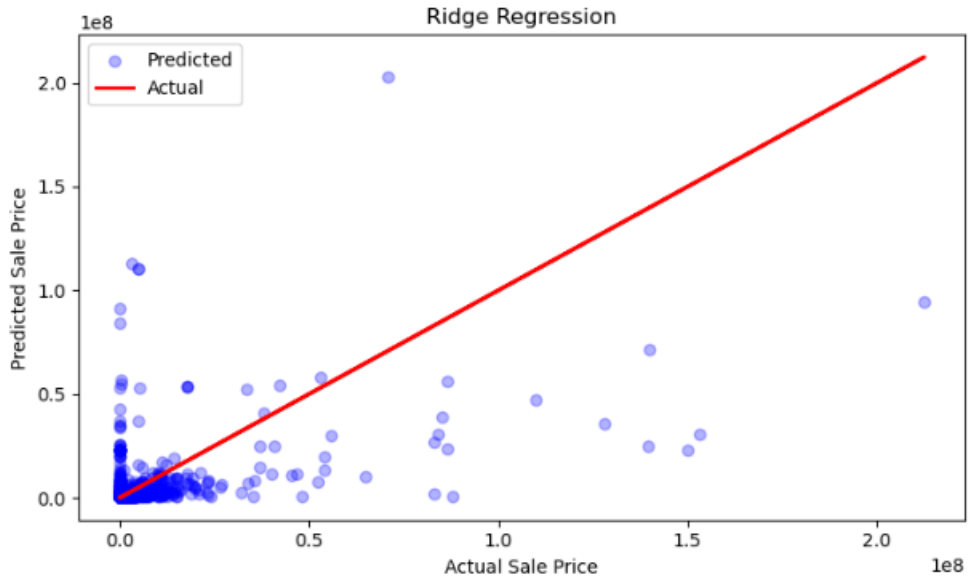
The predictions still show variance from the actual values, with a similar pattern of underestimation and overestimation.

Ridge Regression:

Ridge Regression is similar to Lasso Regression but adds an L2 penalty to the loss function. This penalty term discourages large coefficients and is particularly useful for addressing multicollinearity.

Training and Testing: The model was trained and tested on the same split, using 'GROSS SQUARE FEET' to predict 'SALE PRICE'.

Output: The model's performance improved slightly with an MSE of 2.53 x 10^13 and an R^2 of 0.116, suggesting a better fit than the Linear and Lasso models.



Ridge also adjusts the coefficients like Lasso but uses a different penalty that does not force them to zero, which can be better when there are many small but important effects.

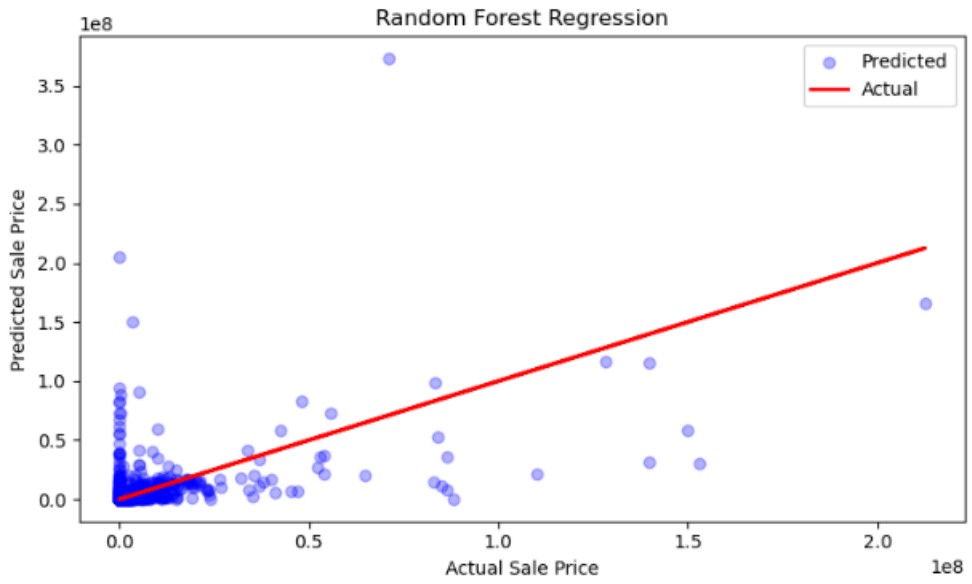
The points are slightly closer to the red line compared to Linear and Lasso, indicating a slightly better fit. There's still a visible deviation, especially for higher actual values, suggesting the model struggles with higher-priced sales.

Random Forest Regression:

Random Forest is an ensemble learning method for regression (and classification) that operates by constructing multiple decision trees during training time and outputting the mean prediction of the individual trees.

Training and Testing: This model used more features from the dataset and was also split into training (80%) and testing (20%) sets.

Output: The model performed worse than the simpler models, with an MSE of 3.62 x 10^13 and a negative R^2 of -0.264, indicating overfitting to the training data.



This model is a type of ensemble learning, where the predictions from multiple decision trees are averaged to improve accuracy.

The plot shows a wide scatter of blue points, indicating that the model's predictions are often far from the actual values.

The model appears to have a particularly hard time with higher-priced sales, with a noticeable spread in predictions.

Decision Tree Regression:

Decision Tree Regression uses a decision tree as a predictive model which maps features (tree branches) to conclusions about the target value.

Training and Testing: It was trained with the same feature set and data split as the Random Forest model.

Output: The model had a high MSE of 1.22 x 10^14 and a very negative R^2 of -3.246, suggesting that it was not capturing the underlying data distribution well and was likely overfitting.

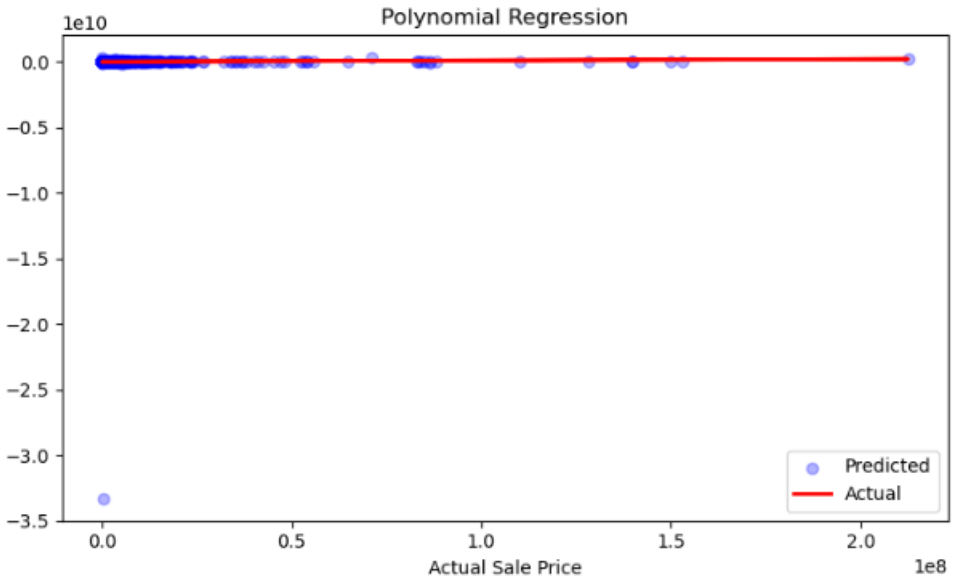
A Decision Tree uses a tree-like model of decisions and their possible consequences. It's a non‑linear approach that can capture complex relationships. The plot reveals a high degree of variance with many points far from the red line, implying that the model is overfitting to the training data. This model seems to have the widest scatter, suggesting a poor fit across the range of sale prices.

Polynomial Regression:

Polynomial Regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial.

Training and Testing: Polynomial features were generated from the same feature set, and the model was split similarly for training and testing.

Output: This model yielded an extremely high MSE of 1.15 x 10^17 and a significantly negative R^2 of -4021.63, indicating significant overfitting to the noise in the training data.



Polynomial regression extends linear regression to include polynomial terms, which allows for a curved line of best fit. This plot shows extreme variance from the actual values, with many points far away from the red line, indicating that the model is dramatically overfitting. The model's predictions are wildly inaccurate, particularly for the mid to higher range of actual values, with some predictions being negative which is non-sensical in the context of sale prices.

In summary, the plots show that while Linear, Lasso, and Ridge regression provide the closest predictions to actual sale prices, they still have considerable errors. The Random Forest and Decision Tree regressions display more variance and poorer fit, with Polynomial regression showing the worst performance due to overfitting.

Values for all the six models:

	Model	Mean Squared Error (MSE)	Coefficient of Determination (R^2)
0	Linear Regression	2.704245e+13	0.055786
1	Lasso	2.704239e+13	0.055788
2	Ridge	2.533110e+13	0.115539
3	Random Forest	3.621230e+13	-0.264389
4	Decision Tree	1.216075e+14	-3.246048
5	Polynomial Regression	1.152089e+17	-4021.632074

Model Performance Comparison:

The performance of the models was evaluated based on the Mean Squared Error (MSE) and the Coefficient of Determination (R^2). Here are the findings:

Linear and Lasso Regressions performed similarly, indicating that feature selection did not play a significant role with the given features.

Ridge Regression showed slightly better performance, likely due to its ability to handle multicollinearity among features.

Random Forest and Decision Tree models had negative R^2 values, which implies that they performed worse than a simple mean model. This could be due to overfitting to the training data.

Polynomial Regression yielded the worst performance with extremely high MSE and negative R^2, indicating significant overfitting.

Model performance Conclusion:

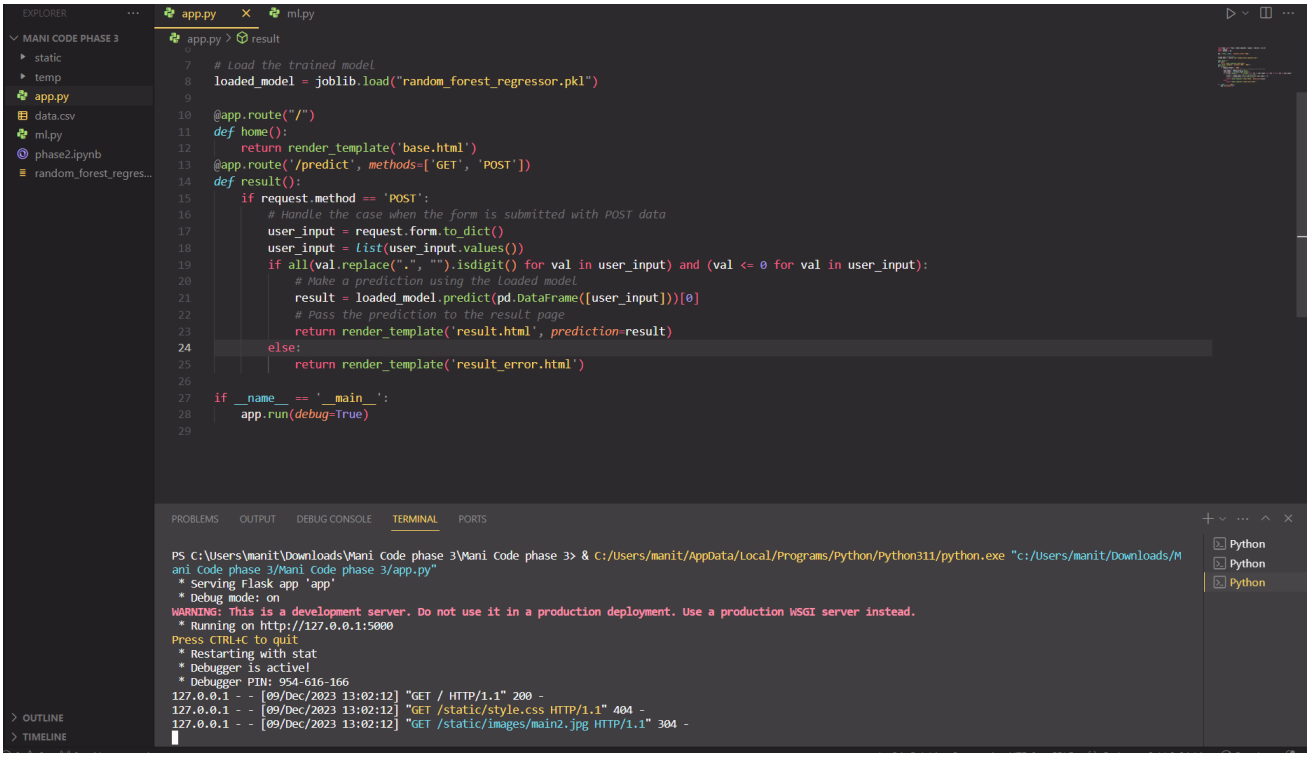
The Ridge Regression model emerged as the best-performing model in terms of MSE and , although it still explained only a small fraction of the variance in sale prices. Given the poor performance across all models, this suggests that the relationships in the dataset are complex and may require more sophisticated feature engineering, alternative modeling approaches, or additional data that captures the underlying patterns affecting sale prices. Further research could

involve hyperparameter tuning, trying different sets of features, or using advanced regression techniques like Support Vector Regression or Neural Networks. The final model choice should balance complexity with predictive power while being mindful of the model's assumptions and the characteristics of the data.

AUTOMATING AND VISUALIZATION THE PROJECT:

Automating the project to predict the sales price of the property based on the land square feet, gross square feet, and building age. For this process, we used the random forest model as the MSE value is more of all the models we have selected. So that is why we are using the random forest model to predict the process.

So, to do this process we used Python. We need to upload the entire automation file into vscode and run the app.py file and in the bottom terminal window we can see the http link to the page. Use CTRL + right click to open the link



```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import joblib
#random forest
from sklearn.ensemble import RandomForestRegressor

file_path = 'data.csv'
df = pd.read_csv(file_path)
selected_features = [
    'LAND SQUARE FEET', 'GROSS SQUARE FEET', 'BUILDING AGE'
]


X = df[selected_features]
y = df['SALE PRICE']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

random_forest_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

random_forest_regressor.fit(X_train, y_train)

joblib.dump(random_forest_regressor, 'random_forest_regressor.pkl')

y_pred_rf = random_forest_regressor.predict(X_test)

mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(mse_rf, r2_rf)
```

Automating the project to predict the sales price of the property based on the land square feet, gross square feet, and building age. This helps the customers to select the desired inputs they need to purchase the property they need.

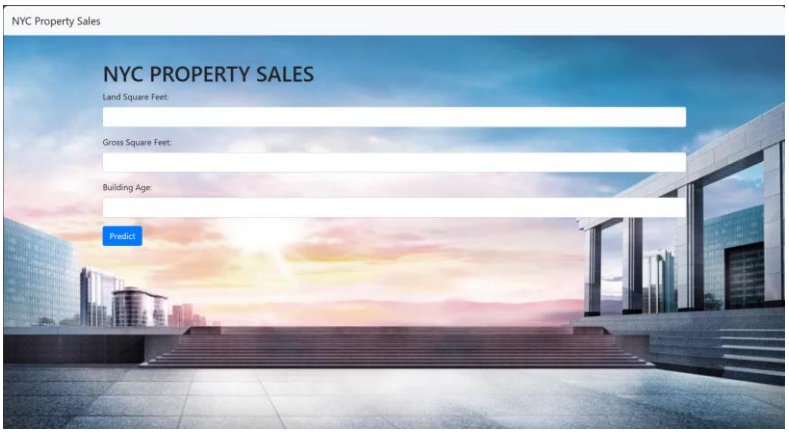


Fig . a

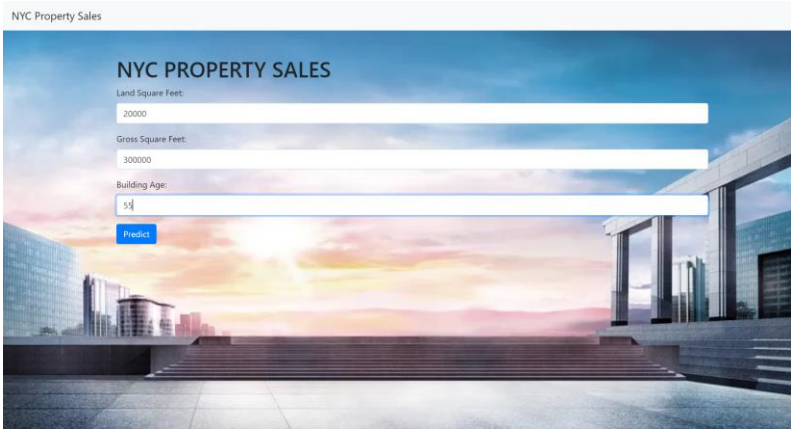


Fig . b

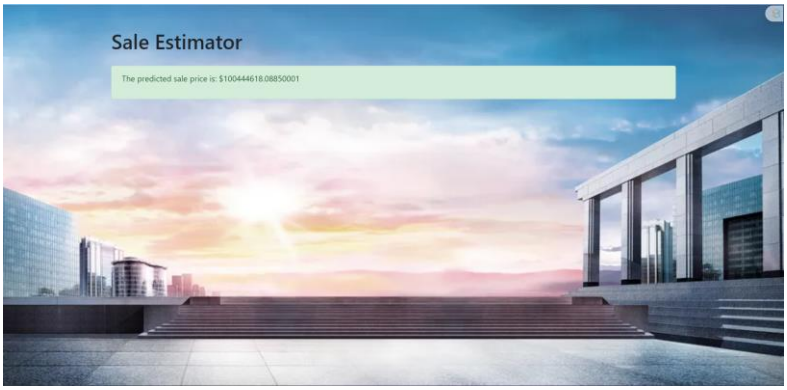
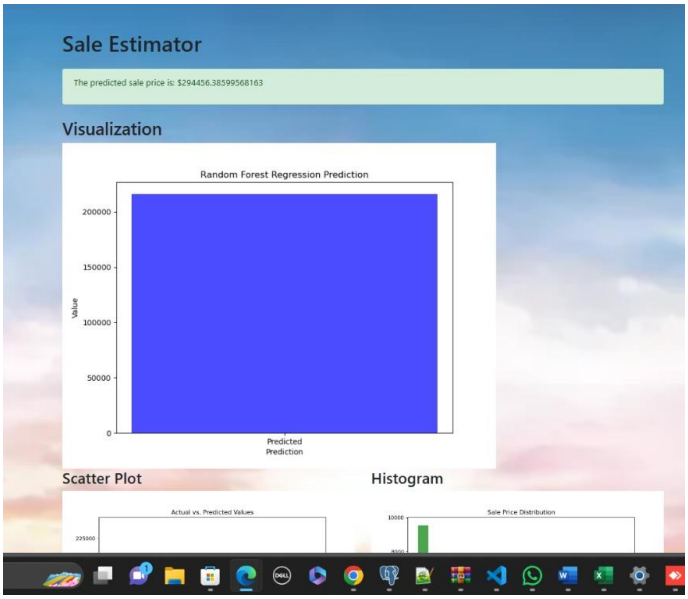


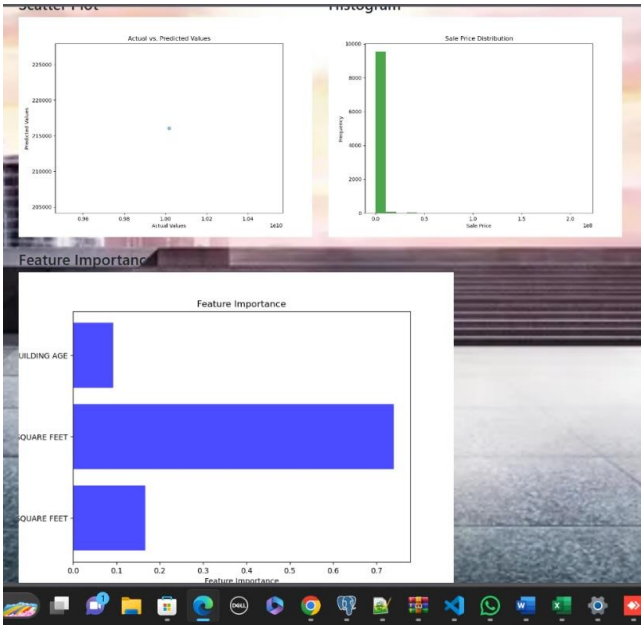
Fig c

From Fig a and Fig b, the users can give the inputs that they are planning to purchase the property based on their needs. After the inputs are done the model(regression model) predicts the prices based on the inputs and makes it easy for the customers

VISUALIZATION:

Not only just going for predicting the prices , we will also do the visualization part so that we it can even be easy to analyze the prices through visualization.





Through visualization we can clearly see the price prediction graph in the random forest model.

REFERENCES:

NYC Property Sales (kaggle.com)

NYC Property Sales (kaggle.com)

End-to-end Machine Learning Regression Project | Kaggle

6 Types of Regression Models in Machine Learning You Should Know About | upGrad blog

User guide and tutorial — seaborn 0.13.0 documentation (pydata.org)

Plotly Python Graphing Library

scikit-learn: machine learning in Python — scikit-learn 1.3.2 documentation