



Module Code & Module Title
CC6057NI Applied Machine Learning

Assessment Type: Coursework
Semester: 1
2024/25 Autumn

Student Name: Dipawoli Malla
London Met ID: 22085768
College ID: NP01AI4S230016

Assignment Due Date: Thursday, January 16, 2025
Assignment Submission Date: Thursday, January 16, 2025
Submitted To: Mahotsav Bhattarai
Word Count (Where Required):8009





*I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked.
I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded*






39% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

-  **157** Not Cited or Quoted 38%
Matches with neither in-text citation nor quotation marks
-  **5** Missing Quotations 1%
Matches that are still very similar to source material
-  **0** Missing Citation 0%
Matches that have quotation marks, but no in-text citation
-  **0** Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

- 8%  Internet sources
- 5%  Publications
- 38%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Table of Contents

1. Introduction	7
1.1. Project Overview	1
1.2. Machine Learning Concepts	1
2. Problem Domain	2
2.1. Problem Definition.....	2
2.2. Previous Dataset Overview.....	3
2.2.1. Source	3
2.2.2. Reasons for not choosing the Dataset.....	3
2.3. New Dataset Overview	5
2.3.1. Source	5
2.3.2. Reasons for choosing the dataset	5
2.3.3. Features Description Table	5
2.3.4. Challenges and Considerations	8
2.4. Background Research	9
3. Solution Approach	10
3.1. Solution Overview	10
3.1.1. Data Preprocessing	10
3.1.2. Feature engineering.....	10
3.1.3. Model development	10
3.1.4. Model Evaluation	10
3.2. Machine Learning Algorithms.....	10
3.2.1. Algorithms Explanation and Justification.....	11
3.2.2. Pseudocode.....	13
3.2.3. Project Flowchart.....	17
3.3. Evaluation Metrics Selection	18
4. Development Tools and Technologies.....	20
4.1. Programming Language	20
4.2. Development environment	20
4.3. Libraries	20
5. Development.....	22
5.1. Exploratory Data Analysis	22
5.2. Data Preprocessing	35

5.2.1.	Handling Missing values	35
5.2.2.	Data Encoding	36
5.3.	Further Preprocessing	38
6.	Results	41
6.1.	Model Development	41
6.1.1.	Split data	41
6.1.2.	Linear Regression	42
6.1.3.	Random Forest Regressor	42
6.1.4.	Support Vector Regression	43
6.1.5.	Lasso Regression	44
6.1.6.	Retraining of the models after further preprocessing	44
6.2.	Model Results	46
6.2.1.	Linear Regression	46
6.2.2.	Random Forest Regression	47
6.2.3.	Support Vector Regression	48
6.2.4.	Lasso Regression	49
6.3.	Hyperparameter tuning	50
6.4.	Model Comparison and Analysis	51
6.5.	Residual Plots	51
6.6.	Actual vs Predicted values plot	54
6.7.	Feature Importance Table	56
7.	Conclusion	58
7.1.	Analysis of work done	58
7.2.	Effectiveness and accuracy	58
7.3.	Addressing real world problems	58
7.4.	Suggestions for future work	Error! Bookmark not defined.
8.	References	59

Table of Figures

Figure 1: House Price rise in US history (Buchholz, 2023).....	2
Figure 2: Nepali Housing Price Dataset	3
Figure 3: Flowchart.....	17
Figure 4: Loading train.csv	22
Figure 5: Loading test.csv	22
Figure 6: Columns information	23
Figure 7: Target value description	23
Figure 8: Histogram of SalePrice	24
Figure 9: Boxplot of SalePrice	25
Figure 10: Correlation heatmap of numerical features	25
Figure 11: Correlation heatmap of most correlated features with SalePrice	26
Figure 12: Correlation values of most correlated features with SalePrice	27
Figure 13: Correlation values of least correlated feature.....	27
Figure 14: Histogram of Yearbuilt and Scatterplot of Yearbuilt and SalePrice	28
Figure 15: Histogram of OverallQual and Scatterplot of OverallQual and SalePrice	28
Figure 16: Histogram of GLiveArea and Scatterplot of GrivArea and SalePrice.....	29
Figure 17: Histogram of GarageCars and Scatterplot of GarageCars and SalePrice	29
Figure 18: Histogram of GarageArea and Scatterplot of GarageArea and SalePrice	30
Figure 19: Histogram of YearBuilt and Scatterplot of SalePrice and YearBuilt	30
Figure 20: EnclosedPorch histogram and Scatterplot with SalePrice.....	31
Figure 21: KitchenAbvGr histogram and Scatterplot with SalePrice.....	31
Figure 22: Histogram of MSSubClass and Scatterplot of MSSubClass and SalePrice	32
Figure 23: Histogram of OverallCond and Scatterplot of OverallCond and SalePrice	32
Figure 24: Boxplot of OverallQual	33
Figure 25: Boxplot of YearBuilt	33
Figure 26: Features with Missing value	34
Figure 27: Dropping of columns	35
Figure 28: Shape of dataset after dropping columns.....	35
Figure 29: Treating missing values for each columns	36
Figure 30: Treating missing values for each columns pt2.....	36
Figure 31: Identification of Ordinal Columns	37
Figure 32: Label encoding on ordinal columns.....	37
Figure 33: Label encoding on categorical features.....	37
Figure 34: OneHotEncoding on categorical features.....	38
Figure 35: List of columns after OneHotEncoding.....	38
Figure 36: Log transformation to y_train_linear.....	38
Figure 37: Original histogram of the SalePrice.....	39
Figure 38: Log transformed histogram of SalePrice	39
Figure 39: Boxplot of SalePrice log transformation	40

Figure 40: Boxplot of SalePrice without log transformation	40
Figure 41: OneHotEncoded columns saved as linear_df	41
Figure 42: LabelEncoded columns saved as tree_df	41
Figure 43: Splitting the data into train and test data set	41
Figure 44: Linear Regression model	42
Figure 45: Intercept and c of linear regression	42
Figure 46: Random Forest Regressor model	43
Figure 47: SVR model	43
Figure 48: Lasso Regression model	44
Figure 49: Retraining of Linear Model after log transformation	45
Figure 50: Retraining of Lasso Model after log transformation	45
Figure 51: Retraining of Lasso Model after log transformation	45
Figure 52: Retraining of Lasso Model after log transformation	45
Figure 53: Evaluation score of linear regression	46
Figure 54: Evaluation Score after retraining Linear Regression	46
Figure 55: Prediction for random forest	47
Figure 56: Evaluation score of Support Vector Regressor	47
Figure 57: Evaluation Score after retraining RandomForestRegression Model	47
Figure 58: Evaluation Score after retraining SVR Model after retraining	48
Figure 59: Evaluation Score after retraining SVR Model after retraining	48
Figure 60: Evaluation score of Lasso Regression	49
Figure 61: Evaluation Score after retraining Lasso Model after retraining	49
Figure 62: Residual plot of linear regression	52
Figure 63: Residual plot of SVR	52
Figure 64: Residual plots of Random Forest regression	53
Figure 65: Residual plots of Lasso Model	53
Figure 66: Actual vs Predicted Values Plots of Linear Regression model	54
Figure 67: Actual vs Predicted Values Plots of SVR model	55
Figure 68: Actual vs Predicted Values Plots of Random Forest Regression model ..	55
Figure 69: Actual vs Predicted Values Plots of Lasso Regression model	56

Table of Tables

Table 1: Model Evaluation score for previous dataset	4
Table 2: Features Description table for new dataset.....	8
Table 3: Model Evaluation score comparison	51
Table 4: Feature Importance Table	57

1. Introduction

1.1. Project Overview

House Price Prediction Model project is a machine learning model that focuses on predicting house prices with house price dataset. The goal of this project is to build machine learning models that will estimate the price of a house based on various factors, such as its location, size, and other features as well as evaluate these models to achieve better accuracy. This can be treated as market analysis tool which can be helpful for buyers, sellers, and real estate professionals to make informed economic decisions with the help of this model.

The aim of this project is to predict the house price using machine learning concepts.

The objective of this project are as follows:

- To research about the problem domain of house price predictions and research done on the machine learning methods of predicting house price.
- To apply machine learning concepts on the development of models and evaluation of models
- To build machine learning models for house price predictions.

1.2. Machine Learning Concepts

This project uses supervised machine learning with regression algorithms and will apply key machine learning concepts which comprises of data preprocessing, feature selection, model selection, model development and model evaluation.

Supervised Learning

Regression models are used to predict continuous outputs (house prices).

The target variable (SalePrice) is labeled, enabling training on historical data.

Regression Techniques

House price prediction is a regression problem because it involves predicting a continuous numerical value (the house price) based on various input features, such as size, location, and quality, rather than categorizing data into discrete classes.

2. Problem Domain

2.1. Problem Definition

The real estate industry is a dynamic and complex market which means it is a rapidly changing business environment that depends on lots of factors. It can be challenging and difficult to predict house prices with proper accuracy. (Li, 2024) There are traditional house price prediction methods that are based on cost and sale price comparison or analyzing transaction volumes that may not fully capture the market's dynamics.

It is important to accurately predict the market to help people make the best strategy and economic decision to buy or sell the house. Using machine learning techniques can help predict house prices with better accuracy by analyzing diverse data points, property features, location specifics, and historical price trends. It also can help us to understand the effects of other factors on pricing.

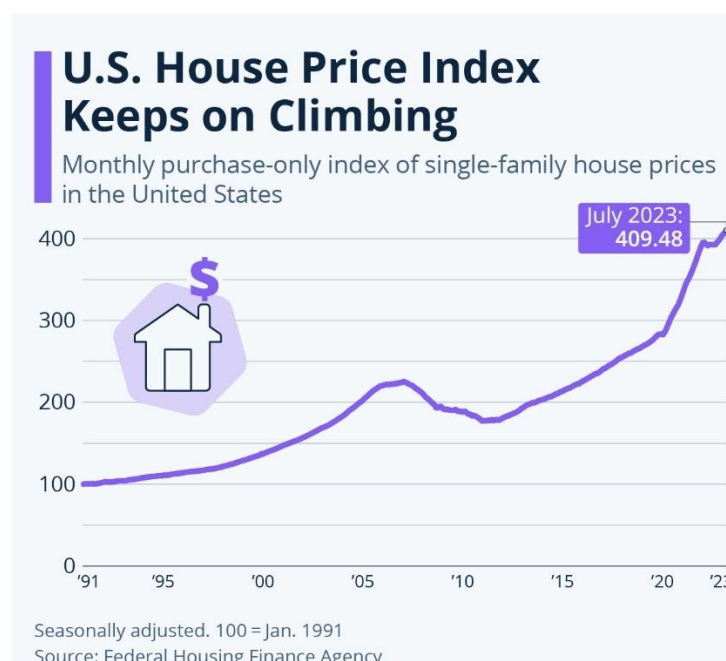


Figure 1: House Price rise in US history (Buchholz, 2023)

2.2. Previous Dataset Overview

2.2.1. Source

The dataset for this project is the Nepali Housing Price Dataset from a popular platform for machine learning datasets named Kaggle. (Montoya & DataCanary, 2016)

The data was extracted from property listing pages of basobaas.com on 2020-04-27, a popular Nepali real estate platform.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Title	Address	City	Price	Bedroom	Bathroom	Floors	Parking	Facet	Year	Views	Area	Road	Road Width	Road Type	Build Area	Posted	Amenities				
2	Flat Syste	Budhanik	Kathmandu	9E+07	6	3	2	10	West	2073	17	1.0-0.0-Az	20 Feet / E 20 Feet									
3	21 Aana B	Paikot, B	Kathmandu	8E+07	5	3	2	9	East	2073	26	0-21-0-0-Az	20 Feet / E 20 Feet									
4	Beautiful	Dhappi	Kathmandu	7E+07	5	3	2	12	East	2071	154	1-1-0-0-Az	20 Feet / E 20 Feet									
5	Modern A	baluwatar	Kathmandu	1.3E+08	6	4	3	9	West	2017	17	0-10-1-0-Az	20 Feet / E 20 Feet									
6	Modern B	Golufutar	Kathmandu	1E+08	6	3	2	10	East	2070	59	0-19-2-0-Az	20 Feet / E 20 Feet									
7	Beautiful	Banasath	Kathmandu	1E+08	5	6	3	9	East	2074	92	0-12-3-1-Az	13 Feet / E 13 Feet									
8	House For	Dadathok	Lalitpur	1.1E+07	3	4	3	1	South East	2019	317	3-1-1-Aani	13 Feet / E 13 Feet									
9	Modern H	Hepati	B. Kathmandu	5E+07	5	4	2	9	East	2073	618	0-8-5-0-Az	20 Feet / E 20 Feet									
10	Flat Syste	baluwatar	Kathmandu	1E+08	10	8	3.5	5	North	2017	110	0-12-3-2-Az	20 Feet / E 20 Feet									
11	Resident	Nearby N	Kathmandu	3.3E+07	6	4	2.5	1	South East	2005	2.6K	0-4-0-0-Az	12 Feet / E 12 Feet									
12	Beautiful	L4, Lekhn	Pokhara	3.2E+07	4	5	3	3	South West	2075	922	14 Aana	15 Feet / E 15 Feet									
13	House Sal	Budhanik	Kathmandu	4E+07	5	4	3	1	North West	2075	592	0-4-2-0-Az	14 Feet / E 14 Feet									
14	Budhanik	Na, Budh	Kathmandu	1.5E+07	2	2	1	1	South	2074	766	5 Aana	12 Feet / E 12 Feet									
15	House For	Na, Budh	Kathmandu	5.3E+07	5	5	3	3	South East	2075	872	8 Aana	20 Feet / E 20 Feet									
16	Tokha	Gra Na, Budh	Kathmandu	2.7E+07	6	6	3	1	East	2076	1.1K	4 Aana	13 Feet / E 13 Feet									
17	Sitapali	INA, Sitap	Kathmandu	2.6E+07	6	6	3	1	East	2076	1.6K	4 Aana	13 Feet / E 13 Feet									
18	2.5 Storey	Nearby C	Kathmandu	3.3E+07	6	4	2.5	1	East	2066	2.8K	0-6-0-0-Az	20 Feet / E 20 Feet									
19	2.5 Storey	Daura	Dip	Kathmandu	1.9E+07	4	4	N/A	3	South	N/A	404	6 Aana	13 Feet / E 13 Feet								
20	House For	Khasi	Baz	Kathmandu	6.2E+07	7	4	N/A	3	East	N/A	164	10 Aana	0 Feet / E 0 Feet								
21	House Sal	Lonag	heji	Kathmandu	1.1E+07	0	0	N/A	0	North	N/A	460	0-4-0-0-Az	10 Feet / E 10 Feet								
22	House For	Sipadi	S. Bhaktap	2.1E+07	7	3	3	1	East	2076	3.2K	0-3-2-0-Az	10 Feet / E 10 Feet									
23	House For	Radhe	Rai	Bhaktap	2.2E+07	0	0	N/A	0	East	N/A	720	0-3-2-0-Az	20 Feet / E 20 Feet								
24	Balaju	Co Na, Balaj	Kathmandu	1.5E+08	95	40	5	7	East	2073	169	21 Aana	20 Feet / E 20 Feet									
25	Budhanik	Na, Budh	Kathmandu	5.5E+07	5	6	3	4	South	2071	253	19 Aana	20 Feet / E 20 Feet									
26	Semi	Furr	Ramkot, S	Kathmandu	2.4E+07	4	4	2.5	1	East	2075	510	0-4-0-0-Az	13 Feet / E 13 Feet								
27	Dhappi	3 Na, Dhapi	Kathmandu	3.7E+07	6	6	3	3	South East	N/A	178	6.1 Aana	32 Feet / E 32 Feet									
28	Golufutar	Na, Goluf	Kathmandu	1E+08	6	6	3	4	East	2070	208	19 Aana	13 Feet / E 13 Feet									
29	Newly	Col	Sital	heji	Lalitpur	2.2E+07	7	4	3.5	2	South	2076	981	0-4-2-0-Az	13 Feet / E 13 Feet							
30	House For	Na, Pepsi	Kathmandu	5.9E+07	6	6	3	2	East	2076	318	6.5 Aana	16 Feet / E 16 Feet									
31	House For	Bojepokhi	Lalitpur	1.9E+07	4	3	2.5	1	East	2076	596	0-3-0-0-Az	13 Feet / E 13 Feet									
32	Newly	Bul	Bojepokhi	Lalitpur	2.1E+07	4	3	2.5	1	East	2076	466	0-3-2-0-Az	13 Feet / E 13 Feet								
33	Imadol	Sy Swastik	T. Lalitpur	1.8E+07	5	3	3	1	South	N/A	157	3-0-2-Aani	13 Feet / E 13 Feet									
34	Baluwatar	Na, Baluw	Kathmandu	1.3E+08	5	6	3	2	East	2074	566	19 Aana	15 Feet / E 15 Feet									
35	Budhanik	Na, Budh	Kathmandu	1.1E+08	8	8	4	4	East	2076	96	21 Aana	15 Feet / E 15 Feet									
36	4 Ropani	F Na, Budh	Kathmandu	1.1E+08	5	5	3	4	South	2070	345	4 Ropani	15 Feet / E 15 Feet									
37	Kapan	Khe Na, Kapan	Kathmandu	2.8E+07	6	5	3	1	East	2076	971	4 Aana	16 Feet / E 16 Feet									
38	Dhumbur	Na, Mandi	Kathmandu	4E+07	7	7	4	2	East	2074	151	7 Aana	15 Feet / E 15 Feet									
39	Hattigowd	Na, Hattig	Kathmandu	2.7E+07	6	6	3	2	West	2075	282	5 Aana	14 Feet / E 14 Feet									
40	Office Spa	Nearby B	Bhaktap	2.3E+07	50	0	2	3.5	2	West	2076	1.1K	2500 Sq. F	20 Feet / E 20 Feet								
41	Grandy	Co Na, Budh	Kathmandu	2.3E+07	6	6	3	1	North East	2070	66	3.1 Aana	18 Feet / E 18 Feet									

Figure 2: Nepali Housing Price Dataset

2.2.2. Reasons for not choosing the Dataset

- The dataset includes null values in essential features such as price, area, no. of bedrooms, bathroom, location, which are key determinants of real estate pricing.
- The dataset includes multiple address lines containing duplicate entries, typos or irrelevant components.
- The dataset includes area measurements in multiple units such as Anna, Ropani, Kattha, and Square Feet which was hard to standardized into a single measurement.
- The evaluation metrics were very low for this model which was below the requirement for this project.

Model Name	RMSE	MSE	MAE	R ² Score
Linear Regression	1.6773×10^7	2.8134×10^{14}	1.2513×10^7	0.19167
Random Forest	1.6719×10^7	2.7952×10^{14}	1.1408×10^7	0.196883
Support Vector Regressor	2.0092×10^7	4.0370×10^{14}	1.4266×10^7	0.159877
Lasso Regression	1.6773×10^7	2.8134×10^{14}	1.2513×10^7	0.19167

Table 1: Model Evaluation score for previous dataset

2.3. New Dataset Overview

2.3.1. Source

The dataset for this project is the Housing Price – Advanced Regression Techniques Dataset from a popular platform for machine learning datasets named Kaggle. (Montoya & DataCanary, 2016)

This dataset contains information about residential homes in Ames, Iowa, with the goal of predicting the final sale price of each house. The dataset provides rich feature engineering opportunities and is often used to explore advanced regression techniques. There are three dataset components for the dataset that are training, test and sample submission file since it is listed for an ongoing competition.

2.3.2. Reasons for choosing the dataset

- The dataset includes a wide range of features, such as OverallQual, GrLivArea, YearBuilt, Neighborhood, and LotArea, which are key determinants of house pricing in real estate.
- It contains both numerical features (e.g., LotFrontage, GarageArea) and categorical features (e.g., MSZoning, Street), allowing for extensive feature engineering and preprocessing opportunities.
- The dataset is suitable for supervised learning since it already has “Price” as a clear target variable.
- The dataset aligns well with real-world real estate problems, particularly for predicting house prices in diverse housing markets.

2.3.3. Features Description Table

The data is explained below in a tabular format with attribute names, data types, and descriptions.

S.N.	Attribute Name	Data Type	Description
1	Id	int64	Unique identifier for each row
2	MSSubClass	int64	Code representing the specific type of dwelling involved in the sale (e.g., 1-story, 2-story, etc.).

3	MSZoning	object	General zoning classification of the property, indicating its land use (e.g., residential, commercial).
4	LotFrontage	float64	Linear feet of street frontage directly connected to the property.
5	LotArea	int64	Total lot size measured in square feet.
6	Street	object	Type of road access available to the property (e.g., paved or gravel).
7	Alley	object	Type of alley access to property
8	LotShape	object	Shape of the property lot (e.g., regular, slightly irregular).
9	LandContour	object	Flatness or elevation contours of the property (e.g., level, hillside).
10	Utilities	object	Type of utilities available
11	LotConfig	object	Lot configuration
12	LandSlope	object	Slope of the property
13	Neighborhood	object	Physical locations within Ames city
14	Condition1	object	Proximity to main road or railway
15	Condition2	object	Proximity to main road or railway (2nd)
16	BldgType	object	Type of dwelling
17	HouseStyle	object	Style of dwelling
18	OverallQual	int64	Rates overall material and finish
19	OverallCond	int64	Rates overall condition
20	YearBuilt	int64	Original construction date
21	YearRemodAdd	int64	Remodel date
22	RoofStyle	object	Type of roof
23	RoofMatl	object	Roof material
24	Exterior1st	object	Exterior covering on house
25	Exterior2nd	object	Exterior covering on house (if multiple)
26	MasVnrType	object	Masonry veneer type
27	MasVnrArea	float64	Masonry veneer area in square feet
28	ExterQual	object	Evaluates exterior material quality
29	ExterCond	object	Evaluates exterior condition

30	Foundation	object	Type of foundation
31	BsmtQual	object	Evaluates basement height
32	BsmtCond	object	Evaluates basement condition
33	BsmtExposure	object	Walkout or garden level basement walls
34	BsmtFinType1	object	Basement finished area rating
35	BsmtFinSF1	int64	Type 1 finished square feet in basement
36	BsmtFinType2	object	Type 2 basement finished area rating
37	BsmtFinSF2	int64	Type 2 finished square feet in basement
38	BsmtUnfSF	int64	Unfinished square feet in basement
39	TotalBsmtSF	int64	Total square feet of basement area
40	Heating	object	Type of heating
41	HeatingQC	object	Heating quality and condition
42	CentralAir	object	Central air conditioning system (Y/N)
43	Electrical	object	Electrical system
44	1stFlrSF	int64	First-floor square feet
45	2ndFlrSF	int64	Second-floor square feet
46	LowQualFinSF	int64	Low quality finished square feet
47	GrLivArea	int64	Above ground living area in square feet
48	BsmtFullBath	int64	Basement full bathrooms
49	BsmtHalfBath	int64	Basement half bathrooms
50	FullBath	int64	Full bathrooms above grade
51	HalfBath	int64	Half bathrooms above grade
52	BedroomAbvGr	int64	Bedrooms above grade
53	KitchenAbvGr	int64	Kitchens above grade
54	KitchenQual	object	Kitchen quality
55	TotRmsAbvGrd	int64	Total rooms above grade (excluding bathrooms)
56	Functional	object	Home functionality rating
57	Fireplaces	int64	Number of fireplaces
58	FireplaceQu	object	Fireplace quality
59	GarageType	object	Garage location
60	GarageYrBlt	float64	Year garage was built

61	GarageFinish	object	Interior finish of the garage
62	GarageCars	int64	Garage capacity in car spaces
63	GarageArea	int64	Size of garage in square feet
64	GarageQual	object	Garage quality
65	GarageCond	object	Garage condition
66	PavedDrive	object	Paved driveway (Y/N)
67	WoodDeckSF	int64	Wood deck area in square feet
68	OpenPorchSF	int64	Open porch area in square feet
69	EnclosedPorch	int64	Enclosed porch area in square feet
70	3SsnPorch	int64	Three season porch area in square feet
71	ScreenPorch	int64	Screen porch area in square feet
72	PoolArea	int64	Pool area in square feet
73	PoolQC	object	Pool quality
74	Fence	object	Fence quality
75	MiscFeature	object	Miscellaneous feature not covered
76	MiscVal	int64	Value of miscellaneous feature
77	MoSold	int64	Month sold
78	YrSold	int64	Year sold
79	SaleType	object	Type of sale
80	SaleCondition	object	Condition of sale
81	SalePrice	int64	Sale price of the property

Table 2: Features Description table for new dataset

2.3.4. Challenges and Considerations

- There are missing data in features like LotFrontage, GarageYrBlt, and MasVnrArea that need imputation or removal.
- There is data imbalance in some categorical features, such as GarageType and MSZoning, that may require balancing.
- There are extreme values in GrLivArea and SalePrice that may need to be addressed.
- Many features may interact non-linearly, requiring careful exploration during feature engineering.

- There are features like GarageArea and GarageCars may have multicollinearity issues that need to be managed.

2.4. Background Research

A study by Angad Gupta et al. (ANGAD GUPTA, 2017) on real estate analysis and prediction tools for Kathmandu Valley focused on analyzing the real estate data to predict land values in Kathmandu Valley using different types of Linear Regression including Lasso and Multi Linear Regression, K-Nearest Neighbors, and Artificial Neural Networks, to estimate commercial land prices. The research provides insights into the effectiveness of different models in capturing the dynamics of Kathmandu's real estate market. This study was very relevant to the project because of the geographical alignment as well as the valuable insights into the local real estate market dynamics of Kathmandu Valley.

Another important study was done by Chenxi Li (Li, 2024) similar to this project that emphasized on development of machine learning model using Linear Regression. It clearly documented the data preprocessing, data analysis and additional emphasis on feature engineering and validation that aligned with my project's objectives.

A comparative study was done by Gauri Chandrasekar and Prof. Syanma Krishna S. (GAURI CHANDRASEKAR, 2024) on machine learning algorithms for predicting house prices in India. The study identifies critical features that affect the house price, along with a comparative analysis of different machine learning algorithms. In previous studies, they focused on a type of machine learning algorithm whereas in this study, they have discussed about three major machine learning algorithms and have done comprehensive study with Support Vector Regression which is very relevant to this project.

3. Solution Approach

3.1. Solution Overview

The solution is to use machine learning model development cycle to predict expected house prices with proper accuracy and check how other features impact the pricing of the house. It can be broken down into the following stages.

3.1.1. Data Preprocessing

The first and crucial step after the data collection is data preprocessing where raw data is processed to make it usable for analysis and modelling. This step includes handling missing values, cleaning data, normalization of data, correcting data types and handling outliers.

3.1.2. Feature engineering

This step involves creating and selecting the most important features for the model by encoding categorical variables like City, Face, as well as identifying which features are more important for predicting house prices using correlation analysis. Techniques like PCA are also involved in this step.

3.1.3. Model development

This step involves building and training machine learning models by first splitting the dataset into train and test set. Then, selecting suitable algorithm, in this case, linear regression or SVR and fitting the selected model to the training data to learn patterns and relationships.

3.1.4. Model Evaluation

This step involves evaluation of model after training by calculating evaluation metrics. In this case, metrics like Mean Absolute Error, Mean Squared Error, and R squared score are used.

3.2. Machine Learning Algorithms

These algorithms are considered for the project from which three algorithms will be used.

- Linear Regression
- Support Vector Regression

- Lasso Regression
- Random Forest Regression

3.2.1. Algorithms Explanation and Justification

3.2.1.1. Linear Regression

Linear Regression (GeeksforGeeks, 2024) is a type of supervised machine learning algorithm that models the linear relationship between the dependent variable and one or more independent variables as a linear function. In the dataset, there is more than one feature, this type of regression is known as Multiple Linear Regression. It assumes that the target variable i.e. house price can be expressed as weighted sum of the input features. The goal is to find the best fit line equation that can predict the values based on the independent variables.

The reasons for choosing linear regression are

- It is suitable for this project because it can help to interpret the coefficients from the dataset.
- It provides direct insights into how features like Area or Bedroom influence house prices.
- It is easy to implement and serves as a benchmark for evaluating more complex models.

3.2.1.2. Support Vector Regression

Support vector regression (SVR) (geeksforgeeks, 2024) is a type of support vector machine (SVM) that is used for regression tasks. It tries to find a function that best predicts the continuous output value for a given input value within a specified margin. It uses linear and non-linear kernels.

The reasons for choosing this model are as follows:

- This model can be suitable for this project because it can capture non-linear relationship i.e. complex dependencies between the features like area, road width and price.
- It also handles outliers and variability effectively especially when pricing trends can be inconsistent.
- It is also effective for small dataset as it focuses on finding a hyperplane with maximum margin and has focus on support vectors.

3.2.1.3. Lasso Regression

Lasso Regression (ibm, 2024), also known as Least Absolute Shrinkage and Selection Operator regression, is a type of regression that uses L1 regularization and variable selection to improve accuracy and interpretability of a model. R1 regularization adds a penalty term to the residual sum of squares (RSS).

The reasons for using lasso regression for this project are

- It uses a statistical method called regularization that can reduce errors caused by overfitting on training data.
- It improves accuracy and interpretability of a model

3.2.1.4. Random Forest Regression

Random Forest Regression (geeksforgeeks, 2024) is an ensemble technique of performing regression with use of multiple decision trees and techniques called Bagging. In Bagging, each tree is trained on a random subset of the data, and the final prediction is obtained by averaging the prediction of all trees.

The reasons for choosing random forest regression are as follows:

- It handles outliers and noise within the dataset
- It captures complex interactions between features without requiring explicit specification.

3.2.2. Pseudocode of Linear Regression

START

IMPORT required libraries

IMPORT dataset

EXPLORE dataset

PREPROCESS dataset

DIVIDE dataset into Training Set and Test Set

INITIALIZE learning rate, number of iterations, and weights (coefficients)

FOR iteration in range(1, number_of_iterations):

CALCULATE predictions using current weights

COMPUTE cost function (Mean Squared Error)

CALCULATE gradients of cost function with respect to weights

UPDATE weights using gradient descent formula

IF convergence criteria met

BREAK loop

END IF

END FOR

EVALUATE the model performance on Test Set using performance metrics (e.g., R-squared, Mean Absolute Error)

SAVE the final model

STOP

3.2.3. Pseudocode of Random Forest Regression

START

IMPORT required libraries

IMPORT dataset

EXPLORE dataset

PREPROCESS dataset

DIVIDE dataset into Training Set and Test Set

INITIALIZE Random Forest parameters (e.g., number of trees, max depth)

FOR each tree in range(1, n_trees):

CREATE a bootstrap sample from the training set

INITIALIZE root node of the tree

WHILE tree depth \leq max_depth:

SELECT a random subset of features

CALCULATE Mean Squared Error (MSE) for all possible splits

CHOOSE the split with the minimum MSE

SPLIT the node into left and right child nodes

END WHILE

ADD the tree to the forest

END FOR

AGGREGATE predictions from all trees

EVALUATE model performance on the Test Set

IF performance criteria not met

ADJUST Random Forest parameters

REPEAT Training and Evaluation

END IF

SAVE the final model

STOP

3.2.4. Pseudocode of Support Vector Regression

START

IMPORT required libraries

IMPORT dataset

EXPLORE dataset

PREPROCESS dataset

DIVIDE dataset into Training Set and Test Set

INITIALIZE SVM parameters:

- Kernel type (e.g., RBF)
- Regularization parameter (C)
- Gamma for RBF kernel (controls smoothness of decision boundary)

TRAIN SVM model using the Training Set:

FOR each data point (x_i , y_i) in the Training Set:

CALCULATE similarity between x_i and all other points using the RBF kernel:

- **MAP** input features into a higher-dimensional space

FIND the hyperplane that maximizes the margin while minimizing misclassification:

- **SOLVE** optimization problem to balance margin size and classification errors

END FOR

EVALUATE the trained model on the Test Set:

FOR each test point (x_j):

COMPUTE decision function using the support vectors and RBF kernel

ASSIGN label based on the sign of the decision function

END FOR

SAVE the final model

STOP

3.2.5. Pseudocode of Lasso Regression

START

IMPORT required libraries

IMPORT dataset

EXPLORE dataset

PREPROCESS dataset

DIVIDE dataset into Training Set and Test Set

INITIALIZE Lasso Regression parameters:

- Regularization strength (alpha)
- Learning rate (if gradient-based optimization is used)

TRAIN Lasso Regression model using the Training Set:

INITIALIZE weights (coefficients) for all features

FOR each iteration:

CALCULATE predictions using the current weights

COMPUTE the cost function:

- Includes Mean Squared Error (MSE) term
- Adds a penalty proportional to the absolute values of the weights

CALCULATE gradients of the cost function with respect to each weight

UPDATE weights to minimize the cost function using gradient descent or coordinate descent

END FOR

EVALUATE the trained model on the Test Set:

CALCULATE predictions for the test data

COMPUTE performance metrics (e.g., Mean Squared Error, R-squared)

IF model performance criteria are not met:

ADJUST Lasso parameters (e.g., alpha, number of iterations)

REPEAT Training and Evaluation

END IF

SAVE the final model

STOP

3.2.6. Project Flowchart

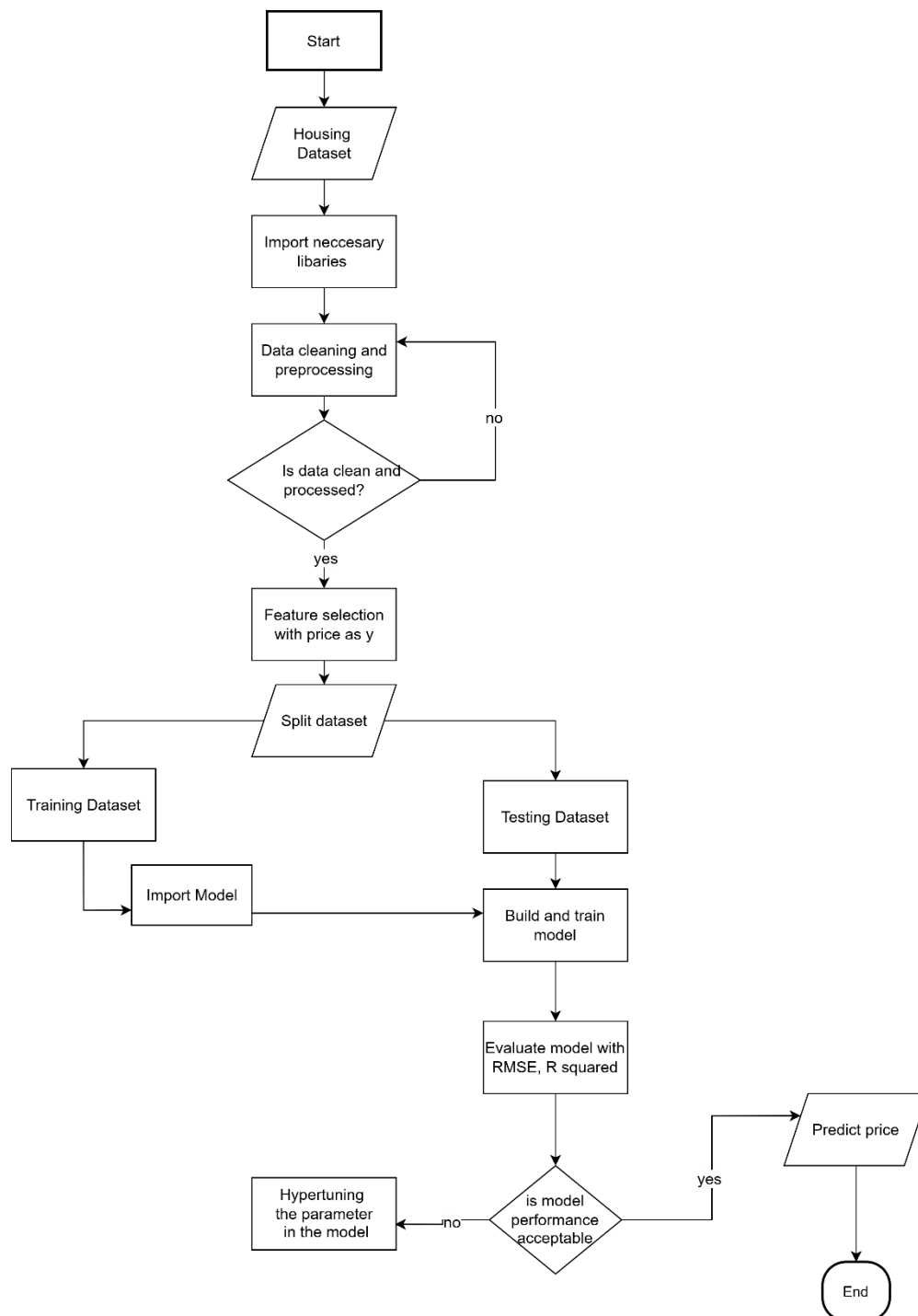


Figure 3: Flowchart

3.3. Evaluation Metrics Selection

The performance of the machine learning will be evaluated using regression-based evaluation metrics to measure the prediction made by the machine learning models.

The selected evaluation metrics that are as follows:

- Mean Squared Error (MSE)

Mean Squared Error measures the average squared differences between the predicted and actual values by squaring the residuals. This metric gives higher weight to larger errors due to squaring operation.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- Mean Absolute Error (MAE)

Mean Absolute Error is an evaluation metric that measures the average magnitude of errors between predicted and actual values. It is also calculated as the average of absolute differences between predictions and actual observations.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- Root Mean Squared Error (RMSE)

The square root of MSE, known as Root Mean Squared Error is a measure of the standard deviation of prediction errors. This metric maintains the same units as the target variable and is used to assess regression model accuracy.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

- R-Squared Error (R^2)

The value of r-squared shows how well the data fits the regression model. It means the amount of variation in dependent variables that can be predicted from independent variables. R^2 of 0.2 means it only explains 20% of the patterns.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

Where y_i = actual value

\hat{y}_i = predicted value

\bar{y} = mean of actual values

n = number of observations

4. Development Tools and Technologies

4.1. Programming Language

- Python

Python (Python, 2024) is a programming language that supports libraries for data analysis and development of machine learning models. It is simple to use and scalable.

4.2. Development environment

- Google Colab

Google Colaboratory (Google Colab, 2024) is a cloud-based platform by Google that provides Jupyter Notebook-like interface. It also simplifies collaboration by allowing code and results to be shared easily.

4.3. Libraries

- Pandas

Pandas is a python library that supports data manipulation and cleaning. It is built on top of NumPy library that offers data structures and operations for manipulating numerical data and time series. It is used for importing and analyzing data much easier. (geeksforgeeks, 2024)

- NumPy

Numpy, also known as numerical python, is a python library for numerical computing on data that can be in the form of large arrays and multi-dimensional matrices. It supports high level mathematical functions to operate on these arrays.

- Matplotlib

Matplotlib is a python library that supports numerical computations and handling arrays. It is used for creating static, manipulated and interactive data visualizations. (geeksforgeeks, 2024) In this coursework, Matplotlib is used for bar graphs, scatterplots, box plot, and histograms.

- Scikit-Learn

Scikit-learn (scikit-learn, 2024) is a python library that supports data preprocessing tasks like scaling, encoding and splitting datasets. It also has functions for Linear regression, support vector regression and other machine learning models. It also helps to calculate the metrics like RMSE that evaluates the machine learning model.

5. Development

5.1. Exploratory Data Analysis

Exploratory data analysis (EDA) is an essential and important initial step to understand the key pattern and identify relationships between the data. This step helps to understand the dataset, its relationship with other features, correlation with target variable and summary statistics of the data. In this project, EDA is performed following ways.

Understanding the dataset

To understand the dataset and learn basic information about the dataset, python line of codes was run as seen in the figure.

Load Training Dataset

```
df_training = pd.read_csv('../ML/data/train.csv')
df_training.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	Pr
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	

Figure 4: Loading train.csv

Load Testing Dataset

```
In [9]:
df_test = pd.read_csv('../ML/data/test.csv')
df_test.head()

Out[9]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandC
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	

Figure 5: Loading test.csv

There are two types of dataset files found in the source called train.csv and test.csv. The “train.csv” file contains the training data and “test.csv” contains the testing data. The training data contains data for 1460 rows which corresponds to 1460 house’s data and 80 columns which correspond to the feature of those houses. Similarly, the testing data contains data of 1461 houses and their 79 attributes.

```

dtype= object )
'9]: df_training.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Id                     1460 non-null   int64
1   MSSubClass             1460 non-null   int64
2   MSZoning               1460 non-null   object
3   LotFrontage            1201 non-null   float64
4   LotArea                1460 non-null   int64
5   Street                 1460 non-null   object
6   Alley                  91 non-null     object
7   LotShape               1460 non-null   object
8   LandContour            1460 non-null   object
9   Utilities              1460 non-null   object
10  LotConfig              1460 non-null   object
11  LandSlope              1460 non-null   object
12  Neighborhood           1460 non-null   object
13  Condition1             1460 non-null   object
..

```

Figure 6: Columns information

After exploring both the dataset, the train.csv is used for the entire project because the test.csv is missing the target variable and cannot be used to build and train the model. Since we are using supervised learning, the target variable is required, and thus train.csv is used.

Understanding target variable: SalePrice

Since the aim of this project is to understand the target variable: SalePrice, understanding the range, basic statistics, distribution of the data of SalePrice is important.

Target Variable: SalePrice that needs to be predicted

```

'1]: # Range of target value 'SalePrice'

print("Lowest value of the target SalePrice:", df_training['SalePrice'].min(), "\n"
      "Maximum value of the target SalePrice:", df_training['SalePrice'].max())

Lowest value of the target SalePrice: 34900
Maximum value of the target SalePrice: 755000

'2]: df_training['SalePrice'].describe()

'3]: count      1460.000000
     mean    180921.195890
     std     79442.502883
     min      34900.000000
     25%    129975.000000
     50%    163000.000000
     75%    214000.000000
     max     755000.000000
     Name: SalePrice, dtype: float64

```

Figure 7: Target value description

From the figure above, basic explanation of statistics can be observed about SalePrice.

- Lowest value (min): The smallest sale price in the data is 34,900.
- Highest value (max): The largest sale price in the data is 755,000.
- Mean (average): The average sale price is about 180,921.
- Standard deviation (std): This tells us how spread out the prices are, and here it is 79,442.
- 25% (1st quartile): 25% of the houses sold for less than 129,975.
- 50% (median): Half of the houses sold for less than 163,000.
- 75% (3rd quartile): 75% of the houses sold for less than 214,000.

```
sns.histplot(df_training['SalePrice'], bins = 30, kde=True)
```

<Axes: xlabel='SalePrice', ylabel='Count'>

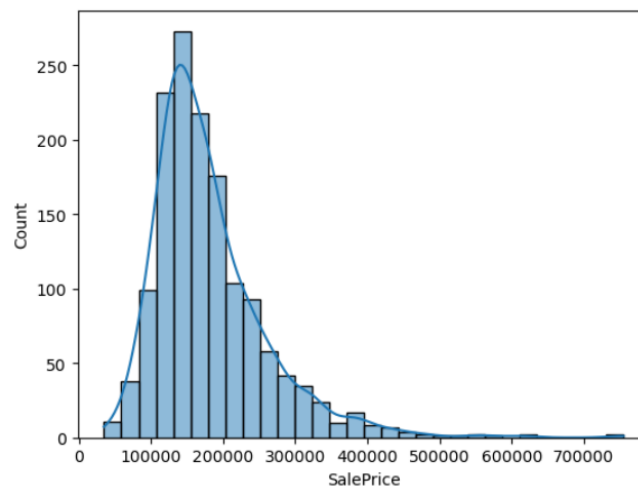


Figure 8: Histogram of SalePrice

The above figure is a histogram of the SalePrice variable, showing how house prices are distributed. It can be observed that each bar represents a range of house prices. The height of the bar shows how many houses fall into that price range.

The smooth curve also known as KDE shows the overall shape of the distribution. It helps us see trends more clearly, like where the data clusters or spreads out.

The distribution is right-skewed, meaning:

- Most houses are sold for lower prices (under \$200,000).
- Fewer houses are sold for higher prices (over \$500,000).

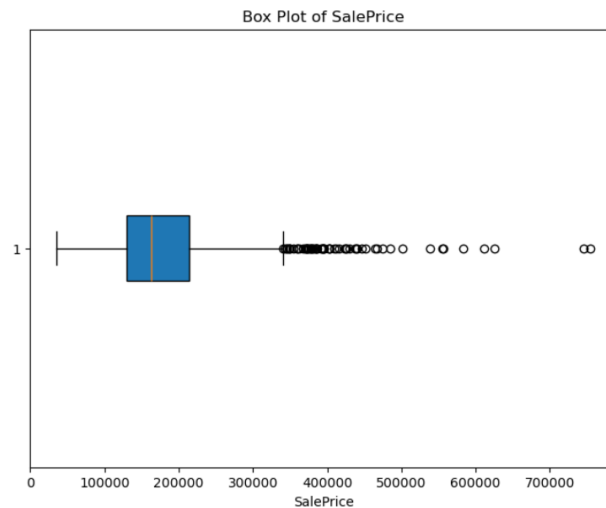


Figure 9: Boxplot of SalePrice

The above box plot of SalePrice shows that most house prices are between 25th percentile and 75th percentile, with some outliers (dots) representing very expensive houses beyond \$400,000.

Correlation Heatmap of numerical features

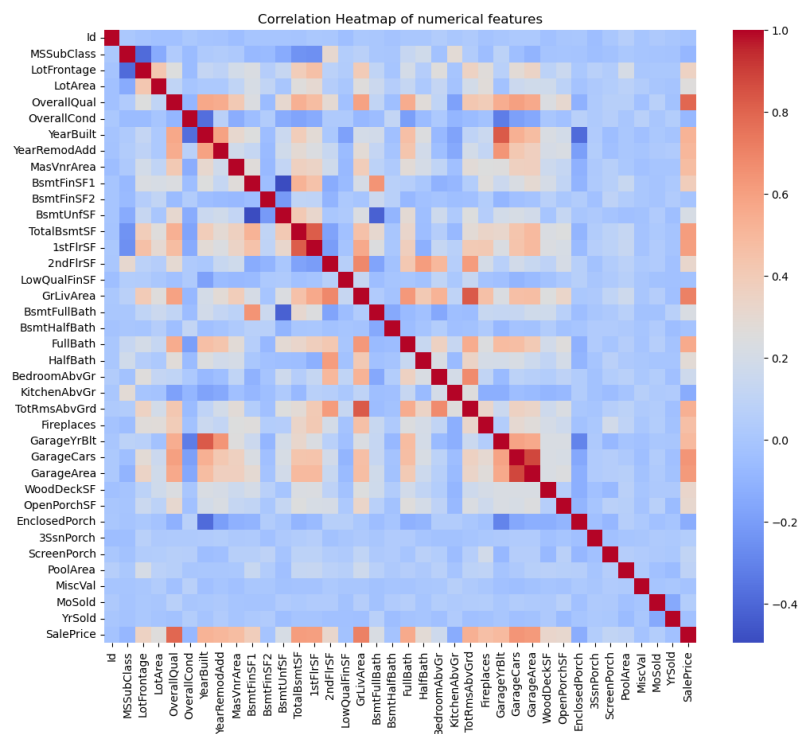


Figure 10: Correlation heatmap of numerical features

The above correlation heatmap shows the relationships between numerical features. Features like OverallQual, GrLivArea, and TotalBsmtSF are strongly correlated with SalePrice, indicating they are important predictors, while other features with weak correlations may have less impact.

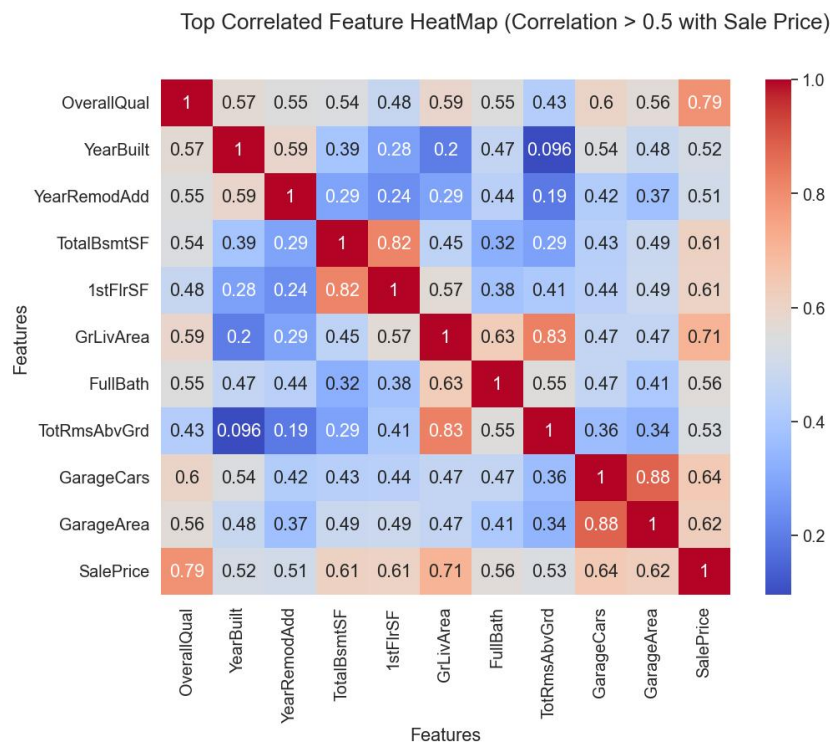


Figure 11: Correlation heatmap of most correlated features with SalePrice

The correlation heatmap in figure 11 focuses on features with strong correlations (above 0.5) to SalePrice. It highlights that OverallQual, GrLivArea, GarageCars, and TotalBsmtSF are the most important predictors for house prices, making them key features to focus on in further analysis.

Correlation Values	
OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
GarageArea	0.623431
TotalBsmstSF	0.613581
1stFlrSF	0.605852
FullBath	0.560664
TotRmsAbvGrd	0.533723
YearBuilt	0.522897
YearRemodAdd	0.507101
GarageYrBlt	0.486362
MasVnrArea	0.477493
Fireplaces	0.466929
BsmstFinSF1	0.386420
LotFrontage	0.351799
WoodDeckSF	0.324413
2ndFlrSF	0.319334
OpenPorchSF	0.315856
HalfBath	0.284108
LotArea	0.263843
BsmstFullBath	0.227122

Figure 12: Correlation values of most correlated features with SalePrice

This table shows the correlation values of different features with SalePrice.

Features like OverallQual, GrLivArea, and GarageCars have the highest positive correlations, meaning they are strong predictors of house prices and should be prioritized in analysis and modelling.

BsmstUnfSF	0.214479
BedroomAbvGr	0.168213
ScreenPorch	0.111447
PoolArea	0.092404
MoSold	0.046432
3SsnPorch	0.044584
BsmstFinSF2	-0.011378
BsmstHalfBath	-0.016844
MiscVal	-0.021190
Id	-0.021917
LowQualFinSF	-0.025606
YrSold	-0.028923
OverallCond	-0.077856
MSSubClass	-0.084284
EnclosedPorch	-0.128578
KitchenAbvGr	-0.135907

Name: SalePrice, dtype: float64

Figure 13: Correlation values of least correlated feature

This table shows features with low or negative correlations to SalePrice, such as KitchenAbvGr, EnclosedPorch, and YrSold. These features are weak predictors of house prices and may have minimal impact on the target variable, suggesting they could be less important for analysis or modelling.

Feature Distribution and Scatter Plot of numerical features with High Correlation to SalePrice

- YearBuilt

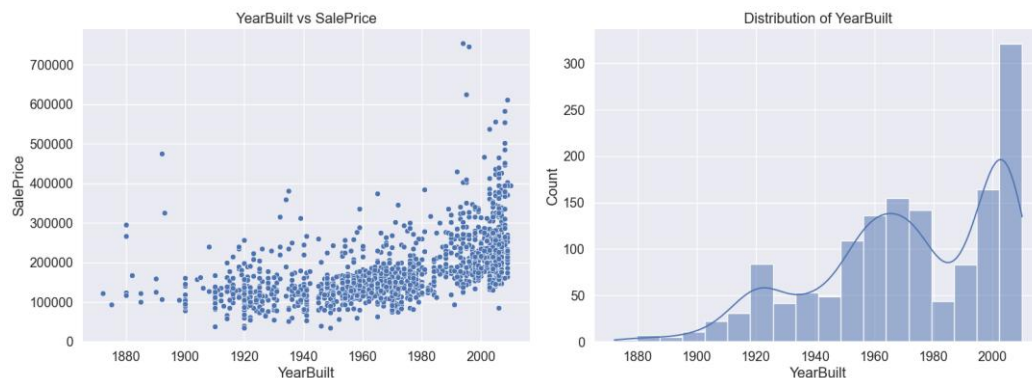


Figure 14: Histogram of Yearbuilt and Scatterplot of Yearbuilt and SalePrice

The left graph shows a positive relationship between YearBuilt and SalePrice, meaning newer houses generally sell for higher prices. The right graph highlights the distribution of houses built over time, with most construction occurring after 1970, peaking around 2000. This helps identify trends in housing prices and construction periods.

- OverallQual

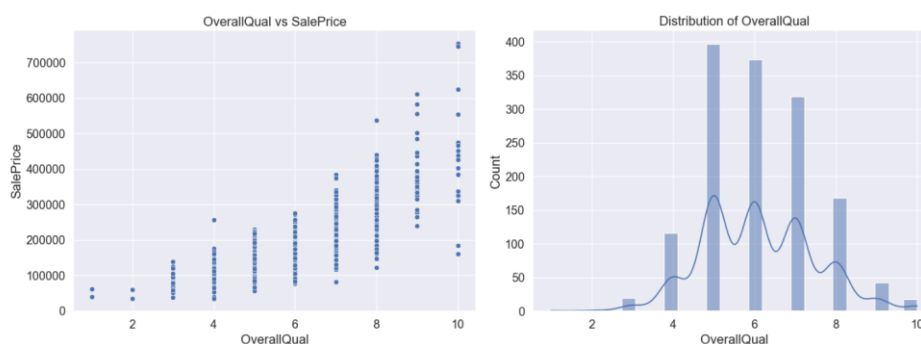


Figure 15: Histogram of OverallQual and Scatterplot of OverallQual and SalePrice

The left graph shows a strong positive relationship between OverallQual (overall quality) and SalePrice, indicating higher quality homes sell for higher prices. The right graph shows the distribution of OverallQual, with most homes having average quality (5-7), which can help focus analysis on common quality levels.

- GrLivArea

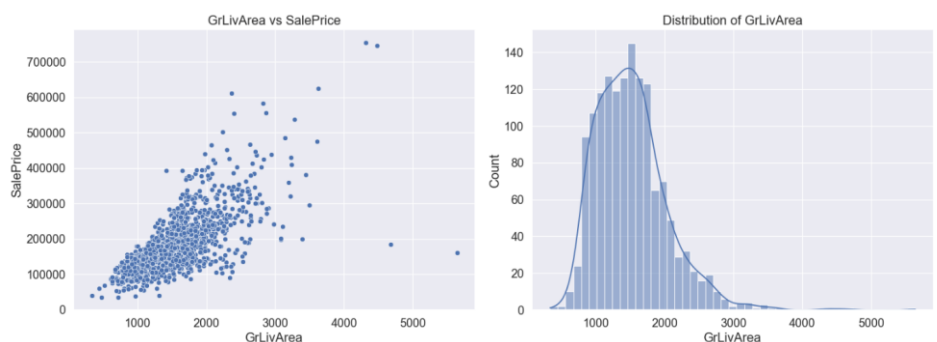


Figure 16: Histogram of GrLivArea and Scatterplot of GrLivArea and SalePrice

GrLivArea refers to the above-ground living area in square feet. It indicates the total livable space, excluding basements. The left graph shows a positive relationship between GrLivArea and SalePrice, where larger homes tend to have higher sale prices. The right graph shows that most homes have living areas between 1000 and 2000 square feet, with very few large homes above 3000 square feet.

- GarageCars

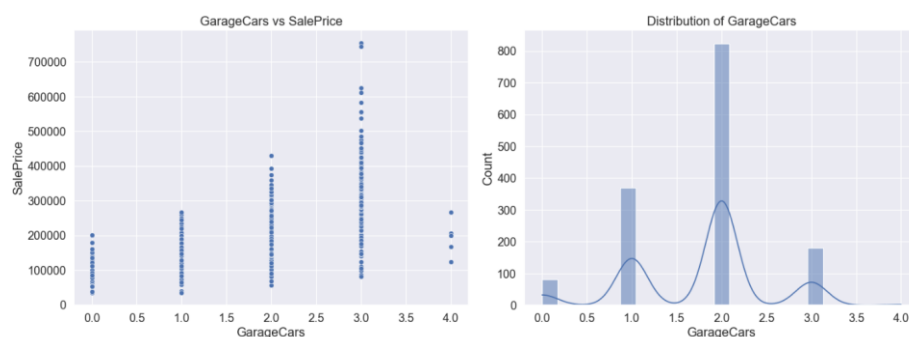


Figure 17: Histogram of GarageCars and Scatterplot of GarageCars and SalePrice

GarageCars represents the size of the garage in terms of the number of cars it can hold. The left graph shows a positive trend between GarageCars and SalePrice, where houses with larger garages tend to have higher prices. The right graph indicates most houses have garages that can hold 1-2 cars, with very few accommodating 3 or more cars.

- GarageArea



Figure 18: Histogram of GarageArea and Scatterplot of GarageArea and SalePrice

GarageArea represents the total area of the garage in square feet. The left graph shows a positive correlation between GarageArea and SalePrice, indicating larger garages are generally associated with higher house prices. The right graph shows that most garages have areas between 400 and 600 square feet, with very few exceeding 800 square feet.

- YearBuilt

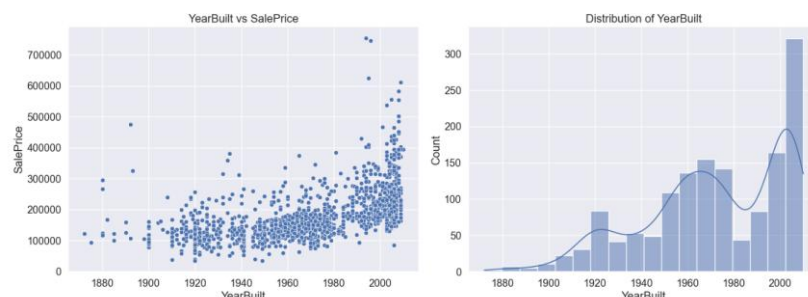


Figure 19: Histogram of YearBuilt and Scatterplot of SalePrice and YearBuilt

YearBuilt refers to the year the house was originally constructed. The left graph shows that newer houses tend to have higher SalePrice, indicating a positive correlation between construction year and price. The right graph shows a steady increase in house construction after 1940, peaking around the 2000s, highlighting a trend of more recent construction.

Histograms and Scatter Plots of 5 least correlated features

- EnclosedPorch

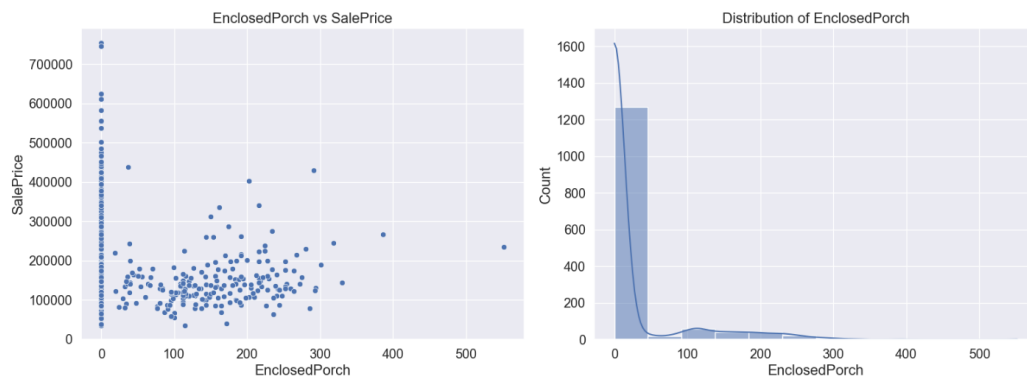


Figure 20: EnclosedPorch histogram and Scatterplot with SalePrice

EnclosedPorch represents the area of the enclosed porch in square feet. The left graph shows no strong relationship between EnclosedPorch and SalePrice, as houses with larger enclosed porches don't consistently have higher prices. The right graph indicates that most houses have little to no enclosed porch area, with very few exceeding 100 square feet.

- KitchenAbvGr



Figure 21: KitchenAbvGr histogram and Scatterplot with SalePrice

KitchenAbvGr represents the number of kitchens above ground in a house. The left graph shows no clear trend between KitchenAbvGr and SalePrice, as most houses have one kitchen regardless of price. The right graph shows that nearly all houses have exactly one kitchen, with very few having more than one.

- MSSubclass

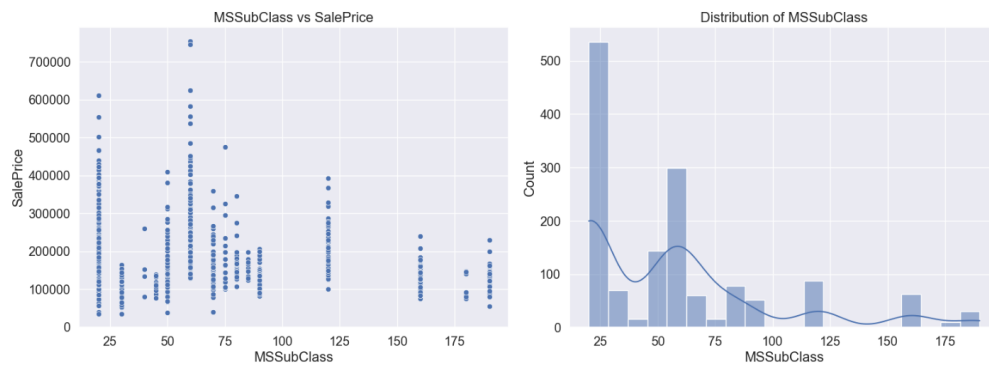


Figure 22: Histogram of MSSubClass and Scatterplot of MSSubClass and SalePrice

MSSubClass identifies the type of dwelling (e.g., 1-story, 2-story, duplex) involved in the sale. The left graph shows varying SalePrice ranges across different MSSubClass types, with some types (like 60 and 120) having higher-priced houses. The right graph highlights that certain dwelling types (like 20, 60, and 50) are more common, while others are less frequent.

- OverallCond

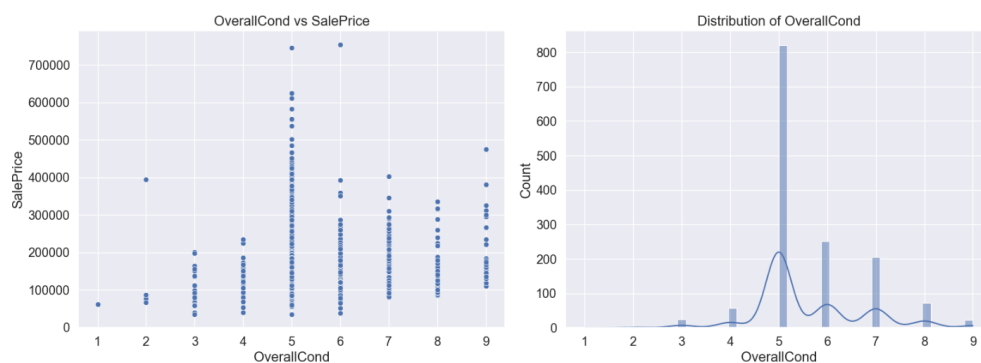


Figure 23: Histogram of OverallCond and Scatterplot of OverallCond and SalePrice

OverallCond rates the overall condition of the house on a scale from 1 (Very Poor) to 10 (Very Excellent). The left graph shows that OverallCond does not have a strong relationship with SalePrice, as houses with average conditions (around 5) vary widely in price. The right graph indicates most houses have an average condition rating of 5, with fewer houses in extreme conditions (very low or very high ratings).

Box Plot of categorical feature: Overall Quality

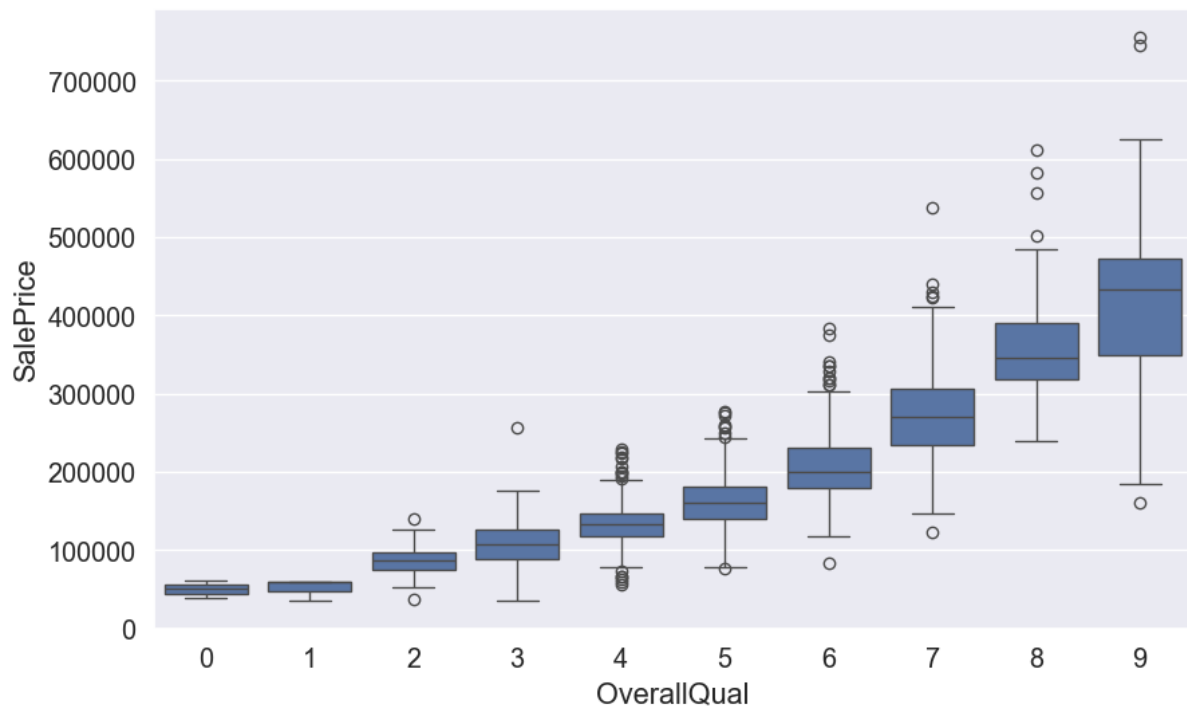


Figure 24: Boxplot of OverallQual

OverallQual rates the overall quality of the house materials and finish, from 1 (Very Poor) to 10 (Very Excellent). The boxplot shows a clear positive relationship between OverallQual and SalePrice; higher-quality houses tend to have higher prices, with increasing variability in price for higher quality ratings.

Box Plot of categorical feature: YearBuilt

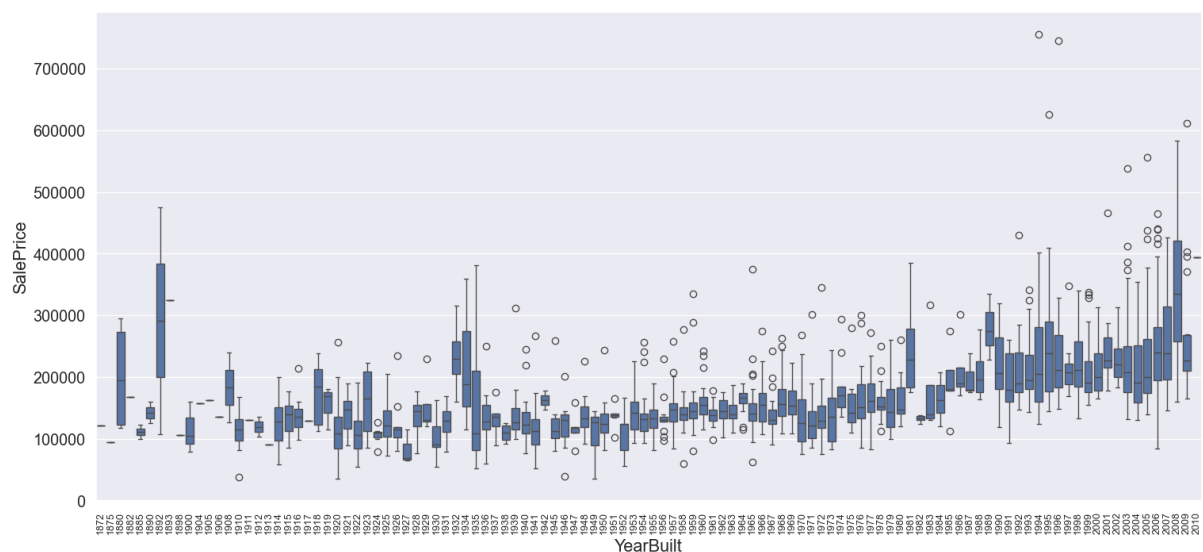


Figure 25: Boxplot of YearBuilt

YearBuilt indicates the year a house was originally constructed. The boxplot shows that houses built in more recent years generally have higher SalePrice, with greater price variability, indicating newer houses tend to be more expensive and diverse in value.

Missing value analysis

Number of features with missing values: 19

	Total Missing	% Missing	Unique Values	Data Type
PoolQC	1453	99.520548	3	object
MiscFeature	1406	96.301370	4	object
Alley	1369	93.767123	2	object
Fence	1179	80.753425	4	object
MasVnrType	872	59.726027	3	object
FireplaceQu	690	47.260274	5	object
LotFrontage	259	17.739726	110	float64
GarageYrBlt	81	5.547945	97	float64
GarageCond	81	5.547945	5	object
GarageType	81	5.547945	6	object
GarageFinish	81	5.547945	3	object
GarageQual	81	5.547945	5	object
BsmtFinType2	38	2.602740	6	object
BsmtExposure	38	2.602740	4	object
BsmtQual	37	2.534247	4	object
BsmtCond	37	2.534247	4	object
BsmtFinType1	37	2.534247	6	object
MasVnrArea	8	0.547945	327	float64
Electrical	1	0.068493	5	object
Id	0	0.000000	1460	int64

Figure 26: Features with Missing value

The above figure shows 19 features with missing values, where PoolQC, MiscFeature, and Alley have the highest percentage of missing data (>90%). Features like GarageYrBlt and MasVnrArea have relatively few missing values (<6%), which may require different handling strategies during data cleaning.

5.2. Data Preprocessing

Data preprocessing involves steps for cleaning and preparing data by handling missing values, removing irrelevant features, and transforming data to make it suitable for analysis or modeling. It ensures the dataset is clean and consistent for accurate predictions.

5.2.1. Handling Missing values

Dropping columns

The code below drops columns with more than 45% missing values (e.g., PoolQC, MiscFeature, Alley) to reduce data sparsity and ensure better model performance. The Id column is also dropped as it is not useful for analysis.

Dropping Columns

```
] : #Dropping any features that have more than 45% of the missing values  
df_training = df_training.drop(['PoolQC', 'MiscFeature', 'Alley', 'MasVnrType', 'FireplaceQu', 'Fence', 'Id'], axis=1)
```

Figure 27: Dropping of columns

The shape of the dataframe after dropping the columns is 1460 rows and 74 columns.

```
df_training.shape  
  
(1460, 74)
```

Figure 28: Shape of dataset after dropping columns

Handling missing values in categorical features

Missing values in categorical features like GarageType and BsmtQual are filled with "None" to indicate absence, such as no garage or basement. For the Electrical feature, the missing value is replaced with the most frequent category (mode) to maintain consistency. This approach ensures the dataset is complete without introducing bias.

```
# Null value likely means No Garage build in the house, so filled as "None" (since these are categorical features)
for col in ('GarageType', 'GarageFinish', 'GarageQual', 'GarageCond'):
    df_training[col] = df_training[col].fillna('None')

# Null value likely means No Basement build in the house, so fill as "None" (since these are categorical features)
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    df_training[col] = df_training[col].fillna('None')

# Only one null value so fill as the most frequent value(mode)
df_training['Electrical'] = df_training['Electrical'].fillna(df_training['Electrical'].mode()[0])
```

Figure 29: Treating missing values for each columns

Handling missing values in numerical features

The code in the figure shown below fills missing values in MasVnrArea and GarageYrBlt with 0, indicating no masonry veneer or garage year. For LotFrontage, the missing values are replaced with the median value, as it likely reflects similar neighborhood properties. This approach ensures numerical features are handled appropriately without introducing bias.

```
# List of numerical features
num_features = ['LotFrontage', 'GarageYrBlt', 'MasVnrArea']

df_training["MasVnrArea"] = df_training["MasVnrArea"].fillna(0) #so fill as 0

# Lot frontage is the feet of street connected to property, which is likely similar to the neighbourhood houses, so fill Median value
df_training["LotFrontage"] = df_training["LotFrontage"].fillna(df_training["LotFrontage"].median())

# Null value likely means No Garage
df_training["GarageYrBlt"] = df_training["GarageYrBlt"].fillna(0) #so fill as 0
```

Figure 30: Treating missing values for each columns pt2

5.2.2. Data Encoding

Data encoding transforms categorical variables into numerical formats for machine learning models. Common methods include Label Encoding (assigning unique numbers to categories) and One-Hot Encoding (creating binary columns for each category). This step ensures models can process and understand non-numerical data effectively.

Label encoding to Ordinal Features

Ordinal features are variables that have a natural order or ranking but the difference between the values may not be equal. For example, OverallQual (house quality rated from 1 to 10).

The code shows that ordinal columns (e.g., OverallQual, ExterQual) are in datatype of numerical features but they represent ranked categories stored as integers. This step is crucial to distinguish between pure numerical data and ordinal data for appropriate transformations or analysis.

```
#Update numerical features after dropping table
numerical_features = df_training.select_dtypes(include='number')

ordinal_cols = [ "OverallQual", "OverallCond", "ExterQual", "ExterCond", "BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2", "HeatingQC",
                 "KitchenQual", "GarageFinish", "GarageQual", "GarageCond", "MSSubClass"]

# Check the data types of the columns in the list
print(df_training[ordinal_cols].dtypes)

OverallQual      int64
OverallCond      int64
ExterQual        int64
ExterCond        int64
BsmtQual         int64
BsmtCond         int64
BsmtExposure     int64
BsmtFinType1     int64
BsmtFinType2     int64
HeatingQC        int64
KitchenQual      int64
GarageFinish     int64
GarageQual       int64
GarageCond       int64
MSSubClass       int64
dtype: object
```

Figure 31: Identification of Ordinal Columns

The code applies Label Encoding to ordinal features, converting categorical rankings (e.g., Excellent, Good, Average) into numerical values (e.g., 3, 2, 1). This preserves the natural order of the categories while transforming them into a format suitable for machine learning models.

```
# Initialize LabelEncoder
label_encoder = LabelEncoder()

for col in ordinal_cols:
    df_training[col] = label_encoder.fit_transform(df_training[col])
    # Print the mapping of categories to numbers
    label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
    print(f"Label Encoding for {col}: {label_mapping}")

Label Encoding for OverallQual: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9)
Label Encoding for OverallCond: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8)
Label Encoding for ExterQual: (0: 0, 1: 1, 2: 2, 3: 3)
Label Encoding for ExterCond: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4)
Label Encoding for BsmtQual: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4)
Label Encoding for BsmtCond: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4)
Label Encoding for BsmtExposure: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4)
Label Encoding for BsmtFinType1: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6)
Label Encoding for BsmtFinType2: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6)
Label Encoding for HeatingQC: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4)
Label Encoding for KitchenQual: (0: 0, 1: 1, 2: 2, 3: 3)
Label Encoding for GarageFinish: (0: 0, 1: 1, 2: 2, 3: 3)
Label Encoding for GarageQual: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5)
Label Encoding for GarageCond: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5)
Label Encoding for MSSubClass: (0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9, 10: 10, 11: 11, 12: 12, 13: 13, 14: 14)
```

Figure 32: Label encoding on ordinal columns

Label encoding to Categorical Features

The label encoding is used for tree models like Random Forest Regression. It is saved in a new dataframe called tree_df. Tree models like Random Forest Regression are not sensitive to the numerical order of encoded labels, making Label Encoding suitable for handling categorical features. By saving the encoded data into tree_df, the dataset is tailored for tree-based algorithms, which can efficiently split and interpret categorical data during training.

```
1: ##Updating categorical columns after dropping the columns
categorical_columns = df_training.select_dtypes(include='object')

2: # Initialize LabelEncoder
label_encoder = LabelEncoder()

3: # Apply Label Encoding to each categorical column
for col in categorical_columns:
    # Fit and transform the column
    df_training[col] = label_encoder.fit_transform(df_training[col])
    # Fit transform() learns the mapping from unique string values to numeric labels.
    # Then, transform() applies this mapping to the column data, converting it into numeric labels.

    # Print the mapping of categories to numbers
    label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
    print(f"Label Encoding for {col}: {label_mapping}")

4: # Save the cleaned dataset with encoded columns
cleaned_data_path = "cleaned_data_label_encoded.csv"
df_training.to_csv(cleaned_data_path, index=False)

print(f"The cleaned dataset with label-encoded categorical columns has been saved to {cleaned_data_path}.")
The cleaned dataset with label-encoded categorical columns has been saved to cleaned_data_label_encoded.csv.
```

Figure 33: Label encoding on categorical features

OneHotEncoding to Categorical Features

The OneHotEncoding is used for tree models like Linear Regression. It is saved in a new dataframe called linear_df. OneHotEncoding is used for linear models like Linear Regression because these models assume no ordinal relationship between categories. It creates binary columns for each category, ensuring the model treats them independently. This prevents incorrect assumptions about the order or hierarchy in categorical data.

```
# Instantiate the OneHotEncoder
ohe = OneHotEncoder(sparse_output=False, drop=None)

# Fit and transform
X_ohe = ohe.fit_transform(df_training[categorical_columns])

# Option 1: Omit the argument (recommended)
ohe_feature_names = ohe.get_feature_names_out()
print("Feature names without passing columns:", ohe_feature_names)

# Convert the array to a DataFrame
df_ohe = pd.DataFrame(X_ohe, columns=ohe_feature_names)

# Combine with the original numeric columns
df_final = pd.concat([df_training.drop(columns=categorical_columns), df_ohe], axis=1)

# Save the cleaned dataset
cleaned_data_path = 'data_one_hot_encoded.csv'
df_ohe_combined.to_csv(cleaned_data_path, index=False)
print(f"The cleaned dataset with one-hot encoded categorical columns has been saved to {cleaned_data_path}.")
```

Figure 34: OneHotEncoding on categorical features

Figure 35: List of columns after OneHotEncoding

5.3. Further Preprocessing

Log transformation of SalePrice

After training of the model and evaluation, transformation of SalePrice and addressing the skewness of the target variable was needed in order to check if the model will perform properly.

```
]: # Apply Log transformation to y_train which is a pandas Series
y_train_log = np.log1p(y_train_linear) # log(1 + SalesPrice)
```

Figure 36: Log transformation to y_train_linear

Log transformation is applied to the target variable (SalePrice) using log1p to address skewness and stabilize variance. This helps linear models perform better by reducing the impact of extreme values and making the data distribution closer to normal, improving prediction accuracy.

The original SalePrice distribution is positively skewed, with a long tail of high prices as seen in the figure below.



Figure 37: Original histogram of the SalePrice

After applying log transformation, the distribution becomes more symmetric and normal-like. This transformation helps improve model performance by stabilizing variance and reducing the influence of extreme values.

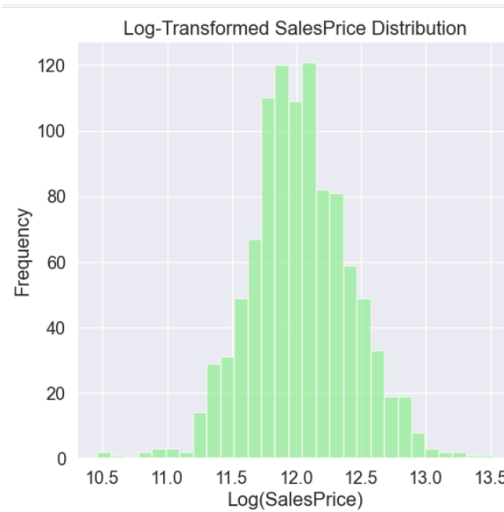


Figure 38: Log transformed histogram of SalePrice

The original SalePrice boxplot shows a significant number of outliers with high values, which can skew the model's predictions.



Figure 39: Boxplot of SalePrice log transformation

After log transformation, the boxplot becomes more compact, reducing the influence of extreme outliers and improving the stability and performance of the model.

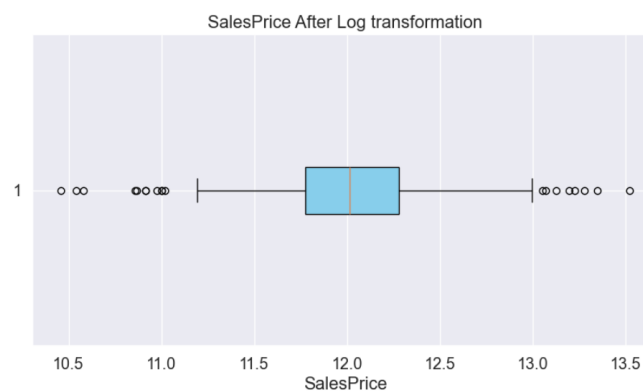


Figure 40: Boxplot of SalePrice without log transformation

6. Results

6.1. Model Development

6.1.1. Split data

After encoding the data with label encoder and OneHotEncoder, the data is saved into two dataframes called `tree_df` and `linear_df` respectively. The data then is checked for its shape as seen in the figure below.

```
linear_df = pd.read_csv('./data_one_hot_encoded.csv')
print(f'The shape of linear df is {linear_df.shape}')

The shape of linear df is (1460, 74)
```

Figure 41: OneHotEncoded columns saved as `linear_df`

In figure 39, we can see that the shape of `linear_df` is 1460 rows and 74 columns or features.

```
tree_df = pd.read_csv('./cleaned_data_label_encoded.csv')
print(f'The shape of tree df {tree_df.shape}')

The shape of tree df(1460, 74)
```

Figure 42: LabelEncoded columns saved as `tree_df`

In figure 40, we can see that the shape of `tree_df` is 1460 rows and 74 columns or features same as `linear_df`.

The `linear_df` and `tree_df` is used to train linear and tree models and thus, splitting is done for training and testing of the data with `test_size = 0.3` which means that 70% of the dataset is used for training set and remaining 30% is used for training set. As seen in figure 41, the shape of testing and training set of both dataframes remain the same.

```
: # Define the target variable and features for linear
X_linear = linear_df.drop(['SalePrice'],axis=1)
y_linear = linear_df['SalePrice']

X_tree = tree_df.drop(['SalePrice'],axis=1)
y_tree = tree_df['SalePrice']

# Split into train and test for linear models
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(X_linear, y_linear, test_size=0.3, random_state=42)

# Split into train and test for tree models
X_train_tree, X_test_tree, y_train_tree, y_test_tree = train_test_split(X_tree, y_tree, test_size=0.3, random_state=42)

# Print the sizes of each dataset
print(f"Training set size for linear: {X_train_linear.shape[0]}")
print(f"Training set size for tree: {X_train_tree.shape[0]}")
print(f"Test set size for linear: {y_test_linear.shape[0]}")
print(f"Test set size for tree: {y_test_linear.shape[0]}")

Training set size for linear: 1022
Training set size for tree: 1022
Test set size for linear: 438
Test set size for tree: 438
```

Figure 43: Splitting the data into train and test data set

6.1.2. Linear Regression

To build linear regression models, there is a class called LinearRegression model from sklearn that fits a linear model to the training dataset. It defines the linear relationship between the target and independent variables.

Model training of Linear Regression

```
# --- Linear Regression ---
linear_model = LinearRegression()
linear_model.fit(X_train_linear, y_train_linear)
```

LinearRegression

LinearRegression()

Figure 44: Linear Regression model

In figure 42, LinearRegression() is fitted to X_train_linear and y_train_linear, indicating that they use one hot encoding data.

```
# Intercept of the linear regression model
print("Intercept of the Linear Regression model:", linear_model.intercept_)
# Coefficients of the linear regression model
print("Coefficients of the Linear Regression model:", linear_model.coef_)

Intercept of the Linear Regression model: 307873.36590922764
Coefficients of the Linear Regression model: [-1.88467791e+03 -2.23642412e+03 -1.84200153e+02  2.96378692e-01
 2.00936327e+04 -1.12803824e+03  4.02149312e+03 -5.82345631e+04
 6.60818841e+01  1.26726445e+04  4.84105390e+02 -4.99049111e+02
-1.44706788e+04 -1.37271236e+03 -1.89055189e+03  1.26430719e+04
 3.86022874e+03  1.67984770e+02  3.56746562e+01  3.00096029e+03
 3.80460054e+03 -1.43252097e+03  6.00533971e+02  1.59075612e+01
-8.16070092e+03  1.03489889e+03  1.35758983e+03 -5.83407885e+03
 2.60050521e+03 -4.29857901e+03 -8.07971662e+02  1.18152337e+00
 3.41527340e+02  2.68330614e+00 -4.43838593e+00 -5.73556509e-01
-3.32737082e+03 -4.06929286e+02  1.23364874e+03 -4.78371018e+02
 1.55019719e+01  1.79848277e+01 -6.17778959e+00  2.73090100e+01
 9.69457413e+03  1.25556687e+03  2.86538446e+03 -8.77310292e+02
-2.87489873e+03 -1.14675319e+04 -9.80647204e+03  4.45222464e+03
 4.41743221e+03  6.25013114e+03 -1.11752545e+02 -1.02459488e+01
 8.04753279e+01  1.57200154e+04 -3.66054318e-02 -5.14721437e+02
 1.13985731e+03  1.53268250e+03  1.05160566e+01 -2.99021069e+01
 2.64673773e+00  4.52373648e+01  6.27052130e+01  1.73217531e+00
-1.98832540e+00 -4.12424748e+01 -3.25872781e+02 -7.72628115e+02
 1.61151425e+03]
```

Figure 45: Intercept and c of linear regression

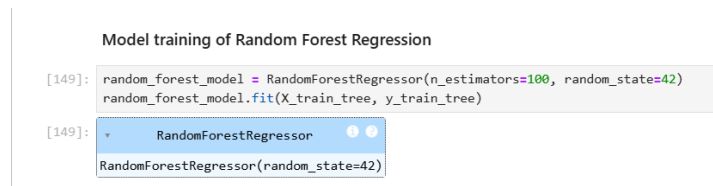
In figure 43, the value of intercept and the value of c can be extracted. These are the weights and biases of linear regression which helps to learn the function of $y=mx+c$. The number of c depends on the number of features which is why there are many of them and one value of intercept.

6.1.3. Random Forest Regressor

To build random forest regressor, RandomForestRegressor class from sklearn.ensemble is imported and fitted to the training dataset. It is ensemble learning of the data.

The most important hyperparameters of this model are n_estimators, max_depth, min_samples_split, random_state, min_samples_leaf, and bootstrap which can be tuned for better results

In figure 44, the RandomForestRegressor model is fitted with `X_train_tree` and `y_train_tree` which indicates that it uses label encoded training set. The hyperparameter used for the current model is `random_state` which is equal to 42. It ensures that the same results can be achieved every time the code is run by controlling the randomness in processes like data shuffling, feature selection and tree construction.



The screenshot shows a Jupyter Notebook cell with the following code and output:

```

Model training of Random Forest Regression

[149]: random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)
       random_forest_model.fit(X_train_tree, y_train_tree)

[149]: RandomForestRegressor
       RandomForestRegressor(random_state=42)

```

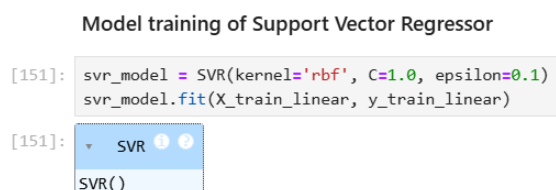
Figure 46: Random Forest Regressor model

6.1.4. Support Vector Regression

To build a support vector regressor (SVR), the SVR class from `sklearn.svm` is imported and fitted to the training dataset. It utilizes the principles of Support Vector Machines for regression tasks, to find the best hyperplane that fits the data within a specified margin of tolerance.

The most important hyperparameters of this model are `C`, `kernel`, `epsilon`, `gamma`, `degree`, and `random_state` which can be tuned for better results.

In figure 45, SVR model is fitted to `X_train_linear` and `y_train_linear`. The hyperparameters used to train the `svr_model` is `kernel`, `c` and `epsilon`. The `kernel='rbf'` means it uses the radial basis function to model complex relationships, `C=1.0` controls how much the model avoids errors versus keeping it simple, and `epsilon=0.1` sets the margin of error where predictions are considered "good enough."



The screenshot shows a Jupyter Notebook cell with the following code and output:

```

Model training of Support Vector Regressor

[151]: svr_model = SVR(kernel='rbf', C=1.0, epsilon=0.1)
       svr_model.fit(X_train_linear, y_train_linear)

[151]: SVR
       SVR()

```

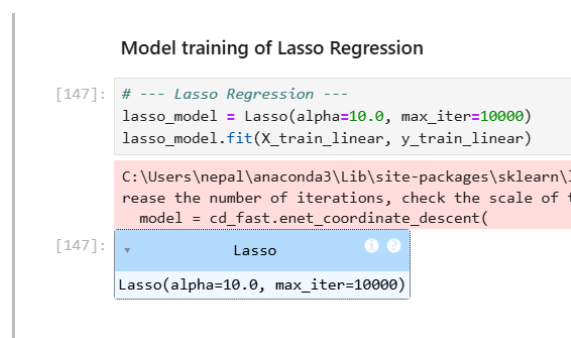
Figure 47: SVR model

6.1.5. Lasso Regression

To build a lasso regression model, the Lasso class from `sklearn.linear_model` is imported and fitted to the training dataset. Lasso, or Least Absolute Shrinkage and Selection Operator, is a type of linear regression that includes a regularization term to the loss function which can lead to sparse models by forcing some coefficients to be exactly zero.

The most important hyperparameters of the model are `alpha`, `fit_intercept`, `normalize`, `precompute`, `copy_X`, `max_iter`, `tol`, `random_state` and `selection` which can be tuned for better results.

In figure 46, `X_train_linear` and `y_train_linear` is used to fit the model. The hyperparameters used for lasso regression is `alpha` and `max_iter`. The `alpha=10.0` controls how strongly the model reduces less important features (higher values mean more regularization), and `max_iter=10000` sets the maximum number of iterations the algorithm will take to find the best solution.



```

Model training of Lasso Regression

[147]: # --- Lasso Regression ---
lasso_model = Lasso(alpha=10.0, max_iter=10000)
lasso_model.fit(X_train_linear, y_train_linear)

C:\Users\nepal\anaconda3\Lib\site-packages\sklearn\
rease the number of iterations, check the scale of t
model = cd_fast.enet_coordinate_descent(

[147]: Lasso
Lasso(alpha=10.0, max_iter=10000)

```

Figure 48: Lasso Regression model

6.1.6. Retraining of the models after further preprocessing

After applying log transformation to the target variable (SalePrice), models like Linear Regression, Lasso Regression, and Random Forest Regressor are retrained. This step ensures the models can better handle the reduced skewness and normalized data, leading to improved performance and more accurate predictions.

Retraining Linear Model after log transformation

```

]: # Train the model
linear_model2 = LinearRegression()
linear_model2.fit(X_train_linear, y_train_log)

```

```

]: LinearRegression
LinearRegression()

```

Figure 49: Retraining of Linear Model after log transformation

Retraining Lasso Model after log transformation

```

: # Initialize the Lasso Regression model
lasso_model2 = Lasso(alpha=10, max_iter=10000)

# Fit the model to the training data
lasso_model2.fit(X_train_linear, y_train_log)

```

```

: Lasso
Lasso(alpha=10, max_iter=10000)

```

Figure 50: Retraining of Lasso Model after log transformation

Retraining Random Forest Regressor Model after log transformation

```

: # Transform target
y_train_log = np.log1p(y_train_tree)
y_test_log = np.log1p(y_test_tree)

rf_model_log = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model_log.fit(X_train_linear, y_train_log)

```

```

: RandomForestRegressor
RandomForestRegressor(random_state=42)

```

Figure 51: Retraining of Lasso Model after log transformation

Retraining Lasso Model after log transformation

```

: # Initialize the Lasso Regression model
lasso_model2 = Lasso(alpha=10, max_iter=10000)

# Fit the model to the training data
lasso_model2.fit(X_train_linear, y_train_log)

```

```

: Lasso
Lasso(alpha=10, max_iter=10000)

```

Figure 52: Retraining of Lasso Model after log transformation

6.2. Model Results

6.2.1. Linear Regression

Before log transformation, the model had an R^2 score of 0.8443, explaining 84.43% of the variance, with an RMSE of 32,961 and an MAE of 22,054, showing decent performance but some large errors.

Model Prediction of Linear Regression

```
[123]: # Predict and evaluate
y_train_pred = linear_model.predict(X_train_linear)
y_test_pred = linear_model.predict(X_test_linear)

print("Linear Regression:")
print(f"Train RMSE: {root_mean_squared_error(y_train_linear, y_train_pred):.4f}")
print(f"Test RMSE: {root_mean_squared_error(y_test_linear, y_test_pred):.4f}")
print(f"Test MSE: {mean_squared_error(y_test_linear, y_test_pred):.4f}")
print(f"Test MAE: {mean_absolute_error(y_test_linear, y_test_pred):.4f}")
print(f"Test R^2: {r2_score(y_test_linear, y_test_pred):.4f}\n")

Linear Regression:
Train RMSE: 31007.7493
Test RMSE: 32961.3567
Test MSE: 1086451037.3922
Test MAE: 22054.8814
Test R^2: 0.8443
```

Figure 53: Evaluation score of linear regression

After log transformation, the R^2 score improved to 0.8939, explaining more variance, and both RMSE (27,213) and MAE (18,633) decreased, indicating better predictions. Log transformation helped the model handle skewness in the target variable and reduced errors significantly.

Evaluation of Linear Regression model after retraining

```
[192]: # Predict in log scale
y_val_pred_log_linear = linear_model2.predict(X_test_linear)

# Reverse log transformation to get predictions in original scale
y_val_pred_linear = np.expml(y_val_pred_log_linear)

# Evaluate using original target values
mse_linear_log = mean_squared_error(y_test_linear, y_val_pred_linear)
mae_linear_log = mean_absolute_error(y_test_linear, y_val_pred_linear)
rmse_linear_log = np.sqrt(mse_linear_log)
r2_linear_log = r2_score(y_test_linear, y_val_pred_linear)

print("Linear Regression (Log-Transformed Target) Metrics:")
print(f" MSE: {mse_linear_log:.4f}") #.4f means 4 decimal points
print(f" RMSE: {rmse_linear_log:.4f}")
print(f"MAE: {mae_linear_log:.4f}")
print(f" R^2: {r2_linear_log:.4f}")

Linear Regression (Log-Transformed Target) Metrics:
MSE: 740585396.3791
RMSE: 27213.6987
MAE: 18633.3623
R^2: 0.8939
```

Figure 54: Evaluation Score after retraining Linear Regression

6.2.2. Random Forest Regression

Before log transformation, the Random Forest model had an R^2 score of 0.9011, showing strong performance, but the RMSE (26,269) and MAE (16,725) indicated room for improvement in handling skewness.

```
# Predict and evaluate
y_train_pred = random_forest_model.predict(X_train_tree)
y_test_pred = random_forest_model.predict(X_test_tree)
```

Figure 55: Prediction for random forest

```
# Evaluate on the training set
train_rmse = root_mean_squared_error(y_train_tree, y_train_pred)
train_mae = mean_absolute_error(y_train_tree, y_train_pred)
train_mse = mean_squared_error(y_train_tree, y_train_pred)

# Evaluate on the test set
test_rmse = root_mean_squared_error(y_test_tree, y_test_pred)
test_mae = mean_absolute_error(y_test_tree, y_test_pred)
test_mse = mean_squared_error(y_test_tree, y_test_pred)

# Print results
print("Random Forest Model:")
print(f"Train RMSE: {train_rmse:.4f}")
print(f"Train MAE: {train_mae:.4f}")
print(f"Train MSE: {train_mse:.4f}")

print(f"Test RMSE: {test_rmse:.4f}")
print(f"Test MAE: {test_mae:.4f}")
print(f"Test MSE: {test_mse:.4f}")
print(f"Test R^2: {r2_score(y_test_tree, y_test_pred):.4f}\n")

Random Forest Model:
Train RMSE: 11826.8296
Train MAE: 6803.1734
Train MSE: 139873898.3053
Test RMSE: 26269.1130
Test MAE: 16725.2574
Test MSE: 690066300.3967
Test R^2: 0.9011
```

Figure 56: Evaluation score of Support Vector Regressor

After log transformation, the R^2 score improved slightly to 0.9006, with reduced RMSE (26,342) and MAE (16,371), showing better error handling. The log transformation helped stabilize predictions and improve the model's reliability for skewed data. This demonstrates effective model development through transformation and tuning.

Evaluation of Random Forest Regression model after retraining

```
[198]: # Predict in log scale
y_val_pred_log_rf = rf_model_log.predict(X_test_tree)

# Invert predictions back to original scale
y_val_pred_rf = np.exp1(y_val_pred_log_rf)

# Evaluate
mse_rf_log = mean_squared_error(y_test_tree, y_val_pred_rf)
mae_rf_log = mean_absolute_error(y_test_tree, y_val_pred_rf)
rmse_rf_log = np.sqrt(mse_rf_log)
r2_rf_log = r2_score(y_test_tree, y_val_pred_rf)

print("Random Forest (Log-Transformed Target) Metrics:")
print(f" MSE: {mse_rf_log:.4f}")
print(f" RMSE: {rmse_rf_log:.4f}")
print(f" MAE: {mae_rf_log:.4f}")
print(f" R^2: {r2_rf_log:.4f}")

Random Forest (Log-Transformed Target) Metrics:
MSE: 693908155.6876
RMSE: 26342.1365
MAE: 16371.6189
R^2: 0.9006
```

Figure 57: Evaluation Score after retraining RandomForestRegression Model

6.2.3. Support Vector Regression

Before log transformation, the SVR model performed poorly, with a negative R^2 score (-0.0320), indicating it could not explain the variance in the data. The RMSE (84,860) and MAE (57,037) were very high, reflecting large prediction errors.

```
# Predict and evaluate
y_train_pred = svr_model.predict(X_train_linear)
y_test_pred = svr_model.predict(X_test_linear)

print("Linear Regression:")
print(f"Train RMSE: {root_mean_squared_error(y_train_linear, y_train_pred):.4f}")
print(f"Test RMSE: {root_mean_squared_error(y_test_linear, y_test_pred):.4f}")
print(f"Test MSE: {mean_squared_error(y_test_linear, y_test_pred):.4f}")
print(f"Test MAE: {mean_absolute_error(y_test_linear, y_test_pred):.4f}")
print(f"Test R^2: {r2_score(y_test_linear, y_test_pred):.4f}\n")

Linear Regression:
Train RMSE: 79262.9393
Test RMSE: 84860.1749
Test MSE: 7201249278.7153
Test MAE: 57037.4543
Test R^2: -0.0320
```

Figure 58: Evaluation Score after retraining SVR Model after retraining

After log transformation, the model's R^2 score improved to 0.7806, and the errors reduced significantly (RMSE: 39,125; MAE: 18,633). This demonstrates that log transformation helped stabilize the data, enabling the SVR model to make more accurate predictions and perform better.

```
# Predict in log scale
y_val_pred_log_svr = svr2.predict(X_test_linear)

# Invert predictions to original scale
y_val_pred_svr = np.exp1(y_val_pred_log_svr)

# Evaluate
mse_svr_log = mean_squared_error(y_test_linear, y_val_pred_svr)
rmse_svr_log = np.sqrt(mse_svr_log)
r2_svr_log = r2_score(y_test_linear, y_val_pred_svr)
mae_linear_log = mean_absolute_error(y_test_linear, y_val_pred_linear)

print("SVR (Log-Transformed Target) Metrics:")
print(f" MSE: {mse_svr_log:.4f}")
print(f" RMSE: {rmse_svr_log:.4f}")
print(f" MAE: {mae_linear_log:.4f}")
print(f" R^2: {r2_svr_log:.4f}")

SVR (Log-Transformed Target) Metrics:
MSE: 1530843404.2172
RMSE: 39125.9940
MAE: 18633.3623
R^2: 0.7806
```

Figure 59: Evaluation Score after retraining SVR Model after retraining

6.2.4. Lasso Regression

Before log transformation, the Lasso Regression model had an R^2 score of 0.8444, explaining 84.44% of the variance, with an RMSE of 32,954 and an MAE of 22,026, showing decent performance.

```
[125]: # Predict and evaluate
y_train_pred = lasso_model.predict(X_train_linear)
y_test_pred = lasso_model.predict(X_test_linear)

print("Linear Regression:")
print(f"Train RMSE: {root_mean_squared_error(y_train_linear, y_train_pred):.4f}")
print(f"Test RMSE: {root_mean_squared_error(y_test_linear, y_test_pred):.4f}")
print(f"Test MSE: {mean_squared_error(y_test_linear, y_test_pred):.4f}")
print(f"Test MAE: {mean_absolute_error(y_test_linear, y_test_pred):.4f}")
print(f"Test R^2: {r2_score(y_test_linear, y_test_pred):.4f}\n")

Linear Regression:
Train RMSE: 31010.6249
Test RMSE: 32954.0659
Test MSE: 1085970460.2742
Test MAE: 22026.6301
Test R^2: 0.8444
```

Figure 60: Evaluation score of Lasso Regression

After log transformation, the R^2 score dropped to 0.7100, but the MAE improved to 18,633, indicating better handling of smaller errors. This suggests that while log transformation slightly reduced the variance explained, it helped the model produce more accurate predictions for individual cases by addressing skewness.

Evaluation of Lasso Regression model after retraining

```
: # Predict in log scale
y_val_pred_log_lasso = lasso_model2.predict(X_test_linear)

# Convert predictions back to original scale
y_val_pred_lasso = np.expml(y_val_pred_log_lasso)

# Evaluate on original target scale
mse_lasso_log = mean_squared_error(y_test_linear, y_val_pred_lasso)
mae_linear_log = mean_absolute_error(y_test_linear, y_val_pred_linear)
rmse_lasso_log = np.sqrt(mse_lasso_log)
r2_lasso_log = r2_score(y_test_linear, y_val_pred_lasso)

print("Lasso (Log-Transformed Target) Metrics:")
print(f" MSE: {mse_lasso_log:.4f}")
print(f" RMSE: {rmse_lasso_log:.4f}")
print(f"MAE: {mae_linear_log:.4f}")
print(f" R^2: {r2_lasso_log:.4f}")

Lasso (Log-Transformed Target) Metrics:
MSE: 2023529889.3737
RMSE: 44983.6625
MAE: 18633.3623
R^2: 0.7100
```

Figure 61: Evaluation Score after retraining Lasso Model after retraining

6.3. Hyperparameter tuning

Since the evaluation score after log transformation wasn't satisfactory, hyperparameter tuning was applied to check the better performance of Lasso Regression model as seen in the figure below.

```
# Data & Parameter Settings

X_train_sample = X_train_linear # Features for training
y_train_sample = y_train_linear # Target (if this is Log-transformed, rename accordingly.)

# Define parameter grids
param_grids = {
    'Lasso': {
        'alpha': [0.01, 0.1, 1, 10, 100], # Regularization strength
        'max_iter': [1000, 5000, 10000] # Number of iterations
    }
}

# Define models to tune
models = {
    'Lasso': Lasso(random_state=42)
}

best_models = {} # Dictionary to store best models for each algorithm
```

Figure 62: Hyperparameter tuning of lasso

This code above in the figure performs hyperparameter tuning using GridSearchCV to optimize the Lasso regression model. The parameter grid specifies values for alpha (regularization strength) and max_iter (iterations), which are tested in a 5-fold cross-validation process. The best parameters and cross-validation score are printed, and the optimal model is saved for further use.

```
# Hyperparameter Tuning Loop
for name, model in models.items():
    print(f"Hyperparameter tuning for {name}")

    # Create the GridSearchCV object
    grid_search = GridSearchCV(
        estimator=model,
        param_grid=param_grids[name],
        scoring='r2', # Or "neg_mean_squared_error", etc.
        cv=5, # 5-fold cross-validation
        verbose=1,
        n_jobs=-1 # Use all available CPU cores
    )

    # Fit on training data
    grid_search.fit(X_train_sample, y_train_sample)

    # Print results
    print(f"Best parameters for {name}: {grid_search.best_params_}")
    print(f"Best cross-validation score (R²): {grid_search.best_score_:.4f}\n")

    # Store the best estimator
    best_models[name] = grid_search.best_estimator_
```

Figure 63: Hyperparameter tuning of lasso

```
Hyperparameter tuning for Lasso
Fitting 5 folds for each of 15 candidates, totalling 75 fits
Best parameters for Lasso: {'alpha': 100, 'max_iter': 1000}
Best cross-validation score (R²): 0.7296
```

Figure 64: Result of hyperparameter tuning

The R2 score of lasso regression model has improved in the log transformed y_train from 0.71 to 0.7296.

6.4. Model Comparison and Analysis

Model	RMSE	MSE	MAE	R2
Linear Regression (Original)	32961.36	1086451037.3922	22054.88	0.8443
Linear Regression (Log-Transformed)	27213.64	740585396.3791	18633.36	0.8939
Random Forest (Original)	26269.11	690066030.3967	16725.26	0.9011
Random Forest (Log-Transformed)	26342.14	693908155.6876	16371.62	0.9006
SVR (Original)	84860.17	7201429278.7153	57037.45	-0.032
SVR (Log-Transformed)	39125.99	1530843440.2172	18633.36	0.7806
Lasso Regression (Original)	32954.07	1089570460.2742	22026.63	0.8444
Lasso Regression (Log-Transformed)	44983.66	2023522989.3737	18633.36	0.71

Table 3: Model Evaluation score comparison

From the table above, follow key insights can be observed

- Random Forest is the most robust model across both original and log-transformed data.
- Log transformation generally improves performance, especially for Linear Regression and SVR but Lasso doesn't improve with log transformation.
- SVR is the least effective model even after transformation.
- Model choice depends on balancing accuracy (R^2) and error metrics (RMSE, MAE) based on the application's needs.

6.5. Residual Plots

Residual Plots of Linear Regression model

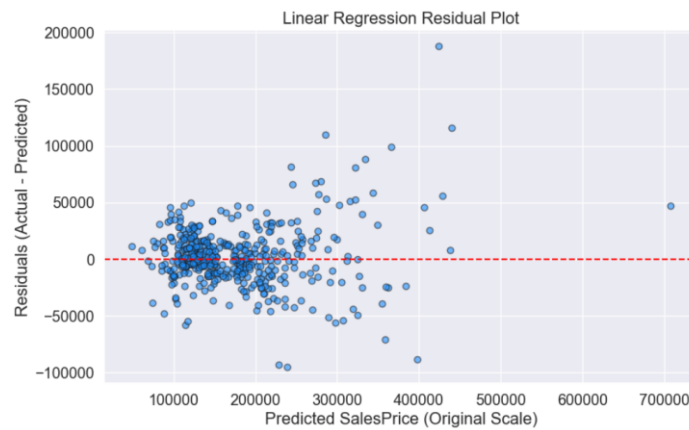


Figure 65: Residual plot of linear regression

The above residual plot shows the difference between actual and predicted house prices (residuals) for the Linear Regression model. Ideally, residuals should be randomly scattered around the red line at 0, indicating accurate predictions. However, the plot shows some patterns and a wider spread of residuals at higher price ranges, suggesting the model struggles to predict very expensive houses accurately. This indicates potential underfitting or the need for further feature adjustments.

Residual Plots of SVR model

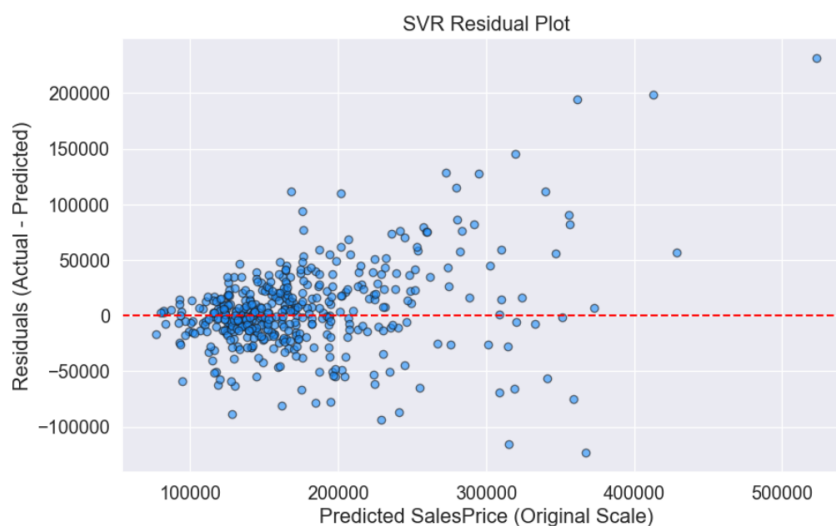


Figure 66: Residual plot of SVR

The above plot shows the residuals (differences between actual and predicted prices) for the SVR model. The residuals are scattered around the red line, but there are noticeable patterns, especially for higher price ranges, where errors are larger. This indicates that the SVR model struggles to predict high-priced houses

accurately. Overall, its performance is less consistent compared to Random Forest or Linear Regression.

Residual Plots of Random Forest Regression model

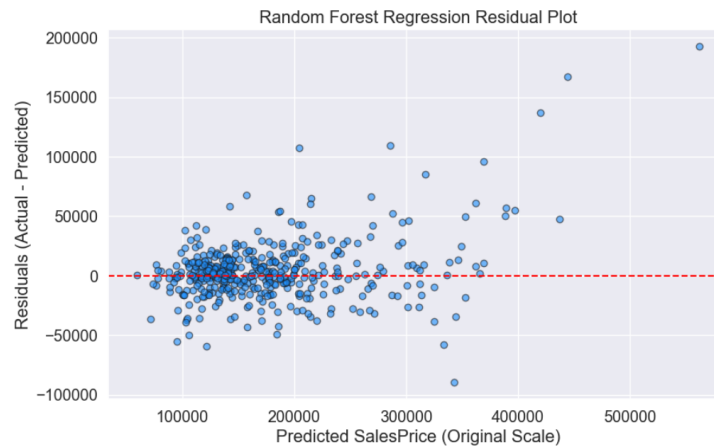


Figure 67: Residual plots of Random Forest regression

The above plot shows the residuals (actual minus predicted prices) for the Random Forest model. The residuals are more evenly scattered around the red line at 0 compared to the Linear Regression plot, indicating better predictions overall. However, there are still some larger residuals at higher price ranges, showing the model struggles slightly with very expensive houses. Overall, the Random Forest model performs well with fewer large errors.

Residual Plots of Lasso Regression model



Figure 68: Residual plots of Lasso Model

The above plot shows the residuals (actual prices minus predicted prices) for the Lasso Regression model. Most residuals are clustered around the red line at 0, but

there are larger errors for higher predicted prices, showing the model struggles with expensive houses. The spread of residuals indicates the model's predictions are less consistent for certain price ranges. Overall, Lasso Regression works well for mid-range prices but struggles with extreme values.

6.6. Actual vs Predicted values plot

Actual vs Predicted Values Plots of Linear Regression model

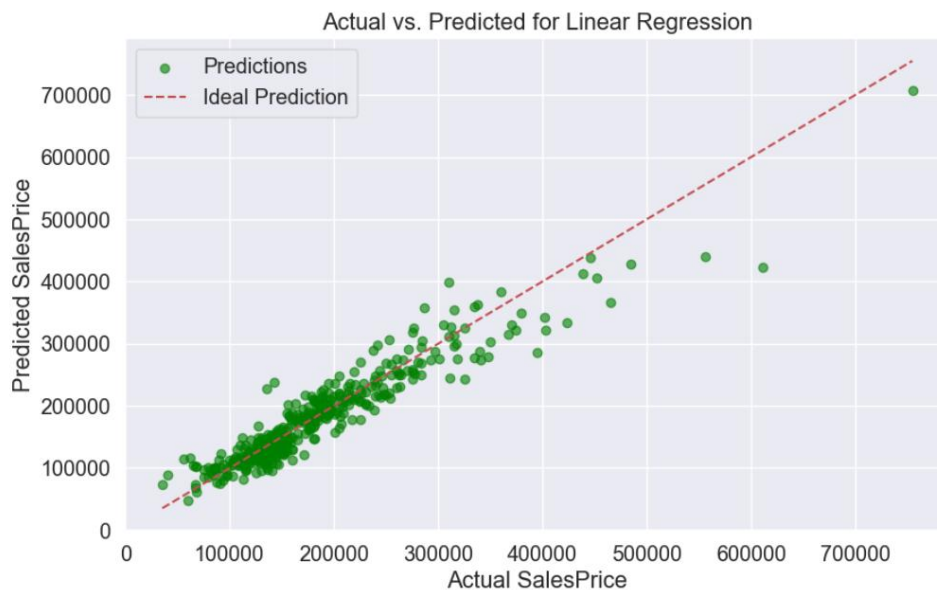


Figure 69: Actual vs Predicted Values Plots of Linear Regression model

The above plot compares the predicted house prices (y-axis) with the actual prices (x-axis). Ideally, all points should lie on the red dashed line, which represents perfect predictions. The points are close to the line for lower and mid-range prices, showing the model predicts these values well. However, some points deviate for higher prices, indicating the model struggles to predict expensive houses accurately.

Actual vs Predicted Values Plots of SVR model

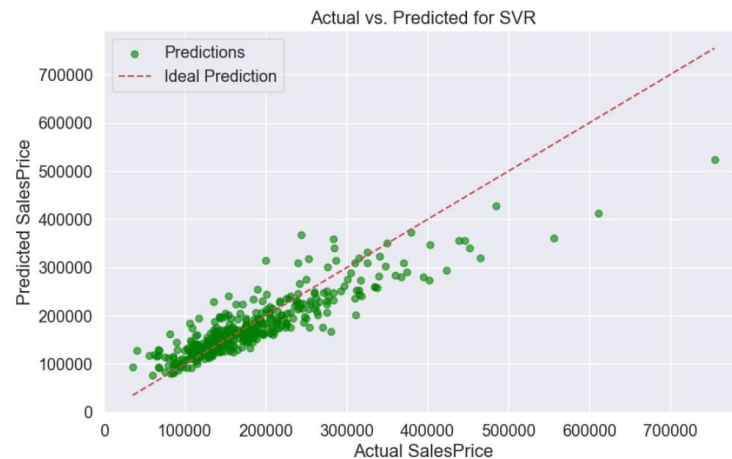


Figure 70: Actual vs Predicted Values Plots of SVR model

The above plot shows how well the SVR model predicts house prices. The red dashed line represents perfect predictions. For lower and mid-range prices, many points are close to the line, showing good predictions. However, for higher prices, the points deviate more, indicating the model struggles to accurately predict expensive houses. Overall, the model performs decently but has room for improvement.

Actual vs Predicted Values Plots of Random Forest Regression model

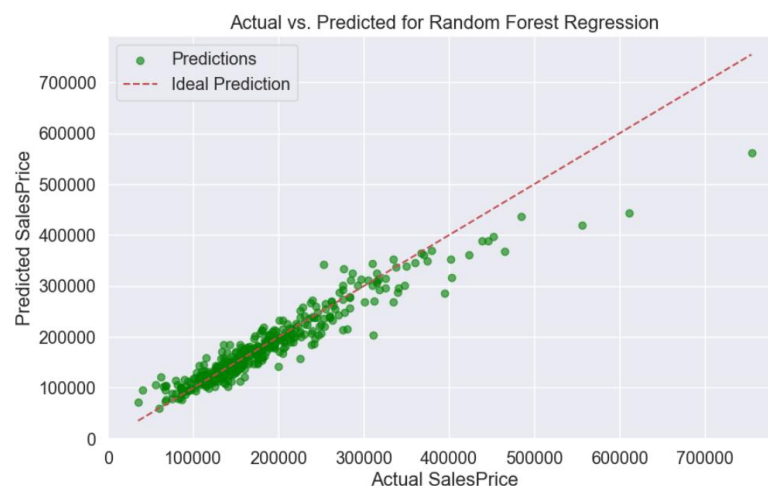


Figure 71: Actual vs Predicted Values Plots of Random Forest Regression model

The above plot shows how well the Random Forest model predicts house prices. The red dashed line represents perfect predictions, and most points are very close to this line, indicating accurate predictions across all price ranges. The model performs especially well for lower and mid-range prices, with only a few deviations for higher prices. Overall, this demonstrates that Random Forest is a strong and reliable model for predicting house prices.

Actual vs Predicted Values Plots of Lasso Regression model

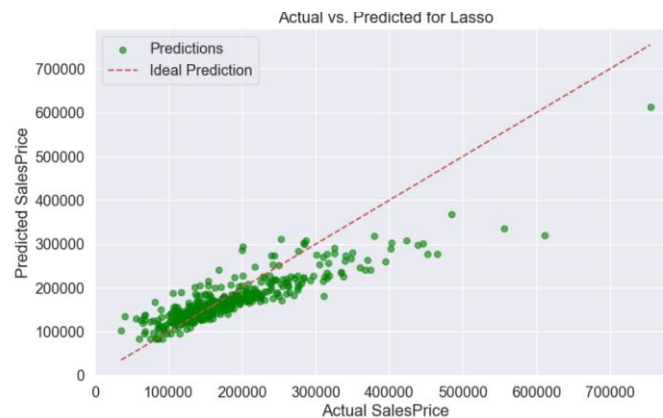


Figure 72: Actual vs Predicted Values Plots of Lasso Regression model

The above plot shows how the Lasso Regression model predicts house prices. The red dashed line represents perfect predictions, and most points are close to it, indicating good performance for lower and mid-range prices. However, there are more deviations at higher prices, showing the model struggles with expensive houses. Overall, the Lasso model performs well but is less accurate for extreme price values.

6.7. Feature Importance Table

The Random Forest model highlights OverallQual as the most influential feature, contributing significantly to predicting SalePrice, followed by features like GrLivArea and GarageCars. These indicate that house quality, living area size, and garage capacity are key factors in determining the price. Understanding feature importance helps identify the most impactful predictors, improving the model's interpretability and focus.

The unique insights gained from the table below which are as follows:

- OverallQual has the highest impact, showing that the material and finish quality of the house is the strongest determinant of price.
- GrLivArea (above-ground living space) significantly influences price, highlighting that larger homes are valued higher.
- Features like GarageCars and GarageArea emphasize that garage size and capacity are important selling points.
- TotalBsmtSF and BsmtFinSF1 reveal that a larger, finished basement contributes meaningfully to home value.

- While less impactful, Neighborhood still plays a role, suggesting that location factors like amenities or reputation affect property prices.

Feature	Importance	Indication
OverallQual	0.552242	Overall material and finish quality
GrLivArea	0.097957	Above ground living area size
GarageCars	0.045197	Number of cars the garage can hold
TotalBsmtSF	0.033791	Total basement square footage
GarageArea	0.029555	Size of the garage in square feet
1stFlrSF	0.022083	First floor square footage
BsmtFinSF1	0.020129	Type 1 finished basement area
LotArea	0.016044	Total lot size in square feet
GarageType	0.014215	Type of garage
YearRemodAdd	0.012461	Year of remodel or addition
YearBuilt	0.011398	Original construction year
OverallCond	0.009387	Overall condition of the house
2ndFlrSF	0.008865	Second floor square footage
GarageYrBltn	0.008709	Year the garage was built
BsmtQual	0.008448	Basement quality
CentralAir	0.008039	Central air conditioning availability
LotFrontage	0.006413	Linear feet of street connected to the property
Neighborhood	0.006106	Neighborhood location
OpenPorchSF	0.005544	Open porch square footage
BsmtUnfSF	0.005473	Unfinished basement square footage

Table 4: Feature Importance Table

7. Conclusion

In this coursework, we explored data in more detailed learning about its patterns, missing values. Furthermore, visualization techniques were used to understand outliers and correlation of variables.

Various regression models predict house prices, utilizing both numerical and categorical features after preprocessing steps like encoding and feature extraction. The evaluation metrics (RMSE, MSE, MAE, and R^2 Score) for Linear Regression, Random Forest, Support Vector Regressor, and Lasso Regression models were analyzed, revealing that while the Random Forest model performed relatively better, all models showed room for improvement due to the high error values and low R^2 scores.

7.1. Analysis of work done

- The data preprocessing and data exploration was setup and performed on the dataset to improve the data quality.
- The correlation graphs help to learn the key predictors of the target variable Price.
- Different techniques like Label encoding, One Hot encoding and were used.
- Model were trained on numerical features.

7.2. Effectiveness

- Random Forest Regression was found to be the highest and effective model in comparison to others with R^2 score of 0.90 achieved.

7.3. Addressing real world problems

- The project directly addresses the real-world issue of predicting house prices in U.S. in global market, aiding in decision-making for buyers, sellers, real estate agents, and policymakers.
- Importance of improved data quality through preprocessing can ensure more reliable predictions, reflecting actual market conditions more accurately.

8. References

ANGAD GUPTA, A. K. A. B. G. S. K. A., 2017. *REAL ESTATE PRICE ANALYSIS AND PREDICTION TOOLS FOR KATHMANDU VALLEY*, Lalitpur: Tribhuvan University, Institute of Engineering, Pulchowk.

Buchholz, K., 2023. *Statista*. [Online]
Available at: <https://www.statista.com/chart/26901/development-of-single-family-home-prices-in-the-us-by-index-points/>
[Accessed 13 January 2025].

GAURI CHANDRASEKAR, P. S. K. S., 2024. HOUSE PRICE PREDICTION MODEL USING MACHINE LEARNING: A COMPARATIVE. *Journal of Emerging Technologies and Innovative Research (JETIR)*, 11(9), pp. 1-6.

geeksforgeeks, 2024. *geeksforgeeks*. [Online]
Available at: <https://www.geeksforgeeks.org/support-vector-regression-svr-using-linear-and-non-linear-kernels-in-scikit-learn/>

GeeksforGeeks, 2024. *Linear Regression*. [Online]
Available at: <https://www.geeksforgeeks.org/ml-linear-regression/#what-is-linear-regression>

geeksforgeeks, 2024. *matplotlib-tutorial*. [Online]
Available at: <https://www.geeksforgeeks.org/matplotlib-tutorial/>

geeksforgeeks, 2024. *pandas-tutorial*. [Online]
Available at: <https://www.geeksforgeeks.org/pandas-tutorial/>

geeksforgeeks, 2024. *Random Forest Regression in python*. [Online]
Available at: <https://www.geeksforgeeks.org/random-forest-regression-in-python/>
[Accessed 22 December 2024].

Google Colab, 2024. *Google Colab*. [Online]
Available at: <https://colab.research.google.com/>
[Accessed 22 December 2024].

ibm, 2024. *Lasso Regression*. [Online]
Available at: <https://www.ibm.com/think/topics/lasso-regression#:~:text=Link%20copied,the%20amount%20of%20regularization%20applied.>
[Accessed 23 December 2024].

Li, C., 2024. *House price prediction using machine learning*. Guangzhou, China, ResearchGate, pp. 1-13.

Montoya, A. & DataCanary, 2016. *House Prices - Advanced Regression Techniques*. [Online]

Available at: <https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview>

Python, 2024. *Python*. [Online]

Available at: <https://www.python.org/>

[Accessed 22 December 2024].

scikit-learn, 2024. *scikit-learn*. [Online]

Available at: <https://scikit-learn.org/stable/>

[Accessed 22 December 2024].