Utilization of Machine Learning (ML) in Prediction of House Pricing at the Real Estate Industry

Under the Supervision of:

Dr. Nik Hadiyan Binti Nik Azman

Prepared By:

Kermina Rafik Abd El Sayed

P-EM0248/23

Academic Year 2023/2024

School of Management

Universiti Sains Malaysia

Abstract

This study elucidates the transformative impact of the Fourth Industrial Revolution, resulting in the emergence of the property technology concept. This phenomenon has significantly affected and revolutionized traditional practices in the real estate industry, especially in the aftermath of the COVID-19 pandemic, with the integration of advanced technologies, including but not limited to artificial intelligence and machine learning. In the real estate industry, predicting house prices is an essential yet complex task. Achieving accurate predictions is made possible through machine learning technology, offering benefits to various stakeholders, including buyers, sellers, investors, and companies. The study explains and examines the significance of utilizing machine learning for accurate house price predictions, exemplifying the concept of property technology. This is demonstrated through an in-depth analysis of the Ames Housing dataset sourced from Kaggle. This involves detailed explanations of data exploration, data pre-processing, model building, and evaluation of models. The objective is to identify the model with the highest performance in predicting house prices. The study employs and compares multiple machine learning regression techniques using three main metrics which are R-squared, Mean Square Error and Root Meen Square Error, showcasing the practical implementation of house price prediction including Lasso, K-Nearest Neighbor Regressor, Support Vector Regressor, Decision Tree Regressor, Random Forest Regressor, Gradient Booster Regressor, eXtreme Gradient Boosting, LightGBM Regressor, and Voting Regressor. Our Champion Model was XGBoost regressor achieving highest R^2 squared, lowest MAE and lowest RMSE while the worst performing model was Decision Tree model with lowest R^2 squared, highest MAE and RMSE.

Key Words: Fourth Industrial Revolution, Property technology, Machine Learning, Real Estate, House Prices, Regression techniques, Lasso, KNN, SVR, DTR, RFR, GBR, XGBoost, LGBM, Voting Regressor.

Table of Contents

List Of Tables

List of Figures

Acknowledgment

First, I want to express my deepest gratitude for God presence, guidance, and support he offered me throughout my research work to complete successfully. Without God's help and support I would not be able to achieve anything. God led me through it all. I would like to thank Dr. Nik Hadiyan Binti Nik Azman , I was honored to work under her supervision. I am also extremely grateful for the unconditional love, support, encouragement and understanding that my parents, brother, and friends showed me throughout my master's journey. They played a huge part; without them I wouldn't be here. Finally, a special thanks to Eng. Basma for guiding and helping me whenever I needed throughout my project, she was always there to support me.

Chapter 1

1.0 Introduction

This chapter explains in detail the background of the study including the evolution of real estate industry, the importance of machine learning in house price prediction, followed by the problem statement discussed throughout this research paper, the research questions, the research objectives and finally the significance of the study, Scope of project and organization.

1.1 Background of study

Industry 4.0 (I4.0) also known as the fourth industrial revolution, which is initiated by digital technologies advancement, it transformed the economy and how industries operate through the integration of artificial intelligence (AI), machine learning , internet of things (IOT), large data, 3D printers, cloud computing and other advanced technology, leading to increasing automation, efficiency, and productivity, also encouraging data-driven decisions. Industry 4.0 is affecting how people live, work, and think. The usage of information technology and services became vital in the world of industries in order to be able to acquire a competitive advantage to differentiate a business from its competitors. (Machkour & Abriane, 2020)

1.1.1 Property technology

The real estate industry could not avoid the reshaping force of industry 4.0. (Starr et al., 2020) Especially after the covid-19 pandemic that led to changing the rules of real estate due to the need for remote and limited interactions where real estate companies and industry needed to quickly adapt to modern technologies. Property technology or prop-tech concept has emerged as a subset of industry 4.0 which simply means the use of digital technology in the real estate industry. Prop-tech is disrupting the practices of traditional real estate market in terms of the way properties are bought, sold, and financed through the presence of technological innovation applications that are transforming the real estate industry which include new category of products, websites, and mobile apps designed to streamline the process of browsing, buying, and building real estate. (cycles & Text, n.d.)

Prop-tech includes but not limited to artificial intelligence and machine learning that are used in predicting house prices, blockchain technology that reduces fraud and enhances the security and transparency of real estate transactions, and data analytics that aids professionals to gain insights from large real estate datasets enabling for better investments decisions and risk assessments.

These technologies are transforming and enhancing various aspects of the real estate industry. (Putatunda, 2019) (Saiz, 2020). Prop-tech investments are continuously growing where most of the investment is concentrated in the United States, China, and Asia specific (APAC) where USA investments reached $ 61.1 billion in the first half of 2022. while China and India reached 12.5 billion and 9.1 billion, respectively. In Europe, the top investors are in Spain, the UK and Germany. Prop-tech investments are expected to grow and consider as an opportunity for real estate companies (cycles & Text, n.d.).

1.1.2 Zillow Real Estate Website

Zillow is well-known in the real estate online market. It is one of the most visited real estate websites in the United States. Zillow company is a successful example of a prop-tech company that utilizes modern technology in their processes. Zillow offers various on-demand services for their customers such as property price listing, home valuation tools and market data to help them in selling, buying, renting, and financing properties with transparency. One of the most important tools on their platform is "Zestimate" which is an automated home valuation model that forecast home's market value considering home characteristics, amenities, location, and market trends such as supply and demand, interest rates and economic conditions. Zillow company is challenging the traditional real estate market by providing more options and information to the buyers and sellers giving them the opportunity to make better informed decisions. (*Zillow*, n.d.)

1.1.3 House Price Prediction

In this study the main focus will be on utilization of machine learning in prediction of house prices in the real estate industry as an application of property technology with highlighting that property technology and machine learning as an opportunity for companies and real estate stake holders. Housing is fundamental for everyone whether for investing or for living in, and house price prediction is a very complex task. house prices are affected by countless number of features including external features such as the physical characteristics of the building, its location, and the neighborhood and the internal features such as the square feet , number of rooms, condition of kitchen, and living area (Kumar & Syed, 2023). Experts in real estate such as agents rely on years of experience for the valuation of houses and properties (Geerts et al., 2023). On the other hand, investors and buyers have multiple concerns when buying a house such as the correct cost for the houses and whether the features of these houses properly synchronize with the price. Thus, it becomes essential to accurately determine the prices of the house or property.

House prices can be predicted using machine leaning models through analyzing datasets of historical sales data that contain house features, demographic information, and other indicators on which the price depends in order to uncover hidden patterns and relationships that may not be obvious to traditional methods. By performing predictive analysis of future trends potential scenarios can be predicted and used to take important strategic decisions. Building a model for houses prediction can benefit both the house sellers and buyers in making well-informed decisions, lowering costs, and increase profit. (Kansal et al., 2023)
In house price prediction, regression techniques are often used among other options where a single target is used as the output which in our case is the price and one or more predictors are used as input indicators which are the house features (Yadav et al., 2023). There are various regression models that can be used including linear regression, random forest regressor, support vector machine regressor, k-nearest neighbor and others as everyday machine learning is evolving to find better and more efficient solutions.

1.1.4 main goal

This study aims to build several machine learning regression techniques to explain the process of house price prediction through machine learning by utilizing a comprehensive Ames housing dataset from Kaggle website. The study will compare these models' performance to select the champion model that can achieve higher performance in Ames, Iowa house price predictions with considering that there is no one size fits all model, best machine learning model to use depends on the size and complexity of the data.

1.1.5 Ames, Iowa City

Before diving into this study, it is important to know more about Ames, Iowa city. Ames is in the state of Iowa, which is a growing city located in united states of America with a population of more than 65,000, this number includes students attending Iowa State University who reside in Ames during the school year or longer. As a growing city, Ames continues to focus on building a strong community filled with opportunities for all citizens. also, the city is expanding housing opportunities and diversifying housing choices. The city has its own assessor's office, where the assessor's primary duty is to assess all real property. The real property is revalued every two years. As on-going process of gathering and reviewing information, measuring, and listing new construction and investigating sales of real estate to determine the fair market value of property (*City of Ames, IA | Home*, n.d.).

1.2 Problem Statement

It is a critical issue for Ames, Iowa real estate stakeholders including agencies, and local government officials in the real estate industry to accurately predict fair house prices as it is a growing population city and evaluation of houses is essential. However, house prices prediction is not an easy task especially when using traditional methods of comparing different houses with various different features that affect prices. These traditional methods are time consuming and often inaccurate as house prices are influenced by several factors such as location, size, condition, , amenities and many others. False price prediction may lead to several problems. If real estate agencies price houses too high this could challenge the selling process and cost them huge monetary loss, while pricing houses too low means not getting the full value, also buyers might end up paying too much or lose out the opportunity of good deals. On the other hand, investors could make wrong investing decisions. Accordingly, false prices can disrupt the real estate market and damage public trust. Furthermore, Banks can face credit risk such as risk related to mortgage loans also the government represented in tax authority may face losses during collecting real estate taxes. So, ensuring accurate house pricing is essential to avoid all the pervious issues in the real estate market. Addressing previous issues, machine learning (ML) can be used to solve this problem. Specifically tailored to the Ames housing market, a machine learning model can be trained on Ames housing dataset to learn the relationship between the factors that affect house prices. Once trained, these machine learning models can be used to predict the prices of new houses with a high degree of accuracy, saving effort, cost, time, and can help real estate stakeholders in Ames, Iowa city to predict house prices and to avoid previous mentioned issues leading to saving resources. As well as making more informed decisions regarding house prices.

1.3 Study questions

I. Which regression model achieves the best performance in Ames house price prediction?
II. What are the major features that significantly influence prediction of house prices?
III. What are the most effective data preprocessing techniques for Ames house price prediction?
IV. Which of the feature engineering techniques can be utilized to enhance models' prediction powers?
V. What are possible future work and limitations?

1.4 Study objectives

I. To evaluate the performance of different regression models in predicting house prices in Ames housing dataset from Kaggle.
II. To understand different features that influence house prices prediction.
III. To identify effective preprocessing techniques for Ames house prices prediction.
IV. To identify the most suitable feature engineering technique to enhance models' prediction.
V. To highlight possible future work and limitations.

1.5 Significance of the study

Nowadays, with the presence of modern technology, accurately predicting house prices has become crucial in the real estate market for many stakeholders, specifically in Ames, Iowa. Successfully implementing house price prediction models has several benefits. Accurate predictions that can help individuals to make more informed decisions concerning property investments, purchases, and sales. Also, reliable price estimates can encourage transparency in the real estate market and reduce uncertainty. It can help in assessing and managing real estate investment risks. Finally, the insights gained from this project can encourage the development of new products and services tailored to Ames, Iowa real estate market.

1.6 Scope of the study

The study focuses on building and comparing multiple regression machine learning techniques for predicting house prices in Ames , Iowa city. In addition to comparing the performance of these models using multiple evaluation techniques. This study does not include any further deployment of machine learning.

1.7 Outline

This project paper consists of five main chapters which are **Chapter 1** the introduction: this chapter is an overview of the project topic which is " Utilization of machine learning (ML) in prediction of house price at the real estate industry.". It starts with an overview of fourth industrial revolution and its impact on the real estate industry through the emergence of Property technology concept and the importance of house price prediction using machine learning, highlighting the problem statements, the project objectives, project questions, and significance of the study. **Chapter 2** the literature review, this chapter reviews and discusses the previous work regarding the topic of house price predictions using machine learning. **Chapter 3** the methodology, which will include explaining the data collection and source, the data preprocessing, the data exploratory and the machine learning model used. **Chapter 4** the findings and discussion from the data analysis and applied machine learning models. Finally, **Chapter 5** will discuss the limitations, recommendations, and conclusions reached.

Chapter 2

2.0 Literature Review

This literature review builds a strong foundation to support this study which is achieved through reviewing past work regarding real estate industry including the explanation of fourth industrial revolution, property technology and its application, in addition to emphasizing the role of machine learning in house price prediction which can benefits stakeholders such as house buyers , seller and investors while describing various used machine learning supervised models and techniques.

2.1 Fourth industrial revolution (Industry 4.0)

Fourth industrial revolution, what is known as industry 4.0 was first defined by Klaus Schwab, founder of the world economic forum which described the industry 4.0, showing a society where people transition through technology between digital and physical world to facilitate their daily activities (Xu et al., 2018). Moreover, as the fourth industrial revolution supports digital technology it is also described as "The Digital Revolution". This technology includes the combination of artificial intelligence, robotics, 3D printing, internet of systems (IOS), internet of things (IOT), cloud computing and large data. The main goal of industry 4.0 is to improve peoples' standard of living and to increase profits through being more faster, effective, and efficient leading to better industrial transformation (Dogaru, 2020) (Machkour & Abriane, 2020). The adoption of industry 4.0 technologies in companies and industry became important as it provides them with innovation, competitiveness, and growth powers (Bai et al., 2020).

2.2 Real estate, Industry 4.0, and Property technology

Real estate as a valuable asset and industry, couldn't avoid the huge impact made by the fourth industrial revolution (industry 4.0) innovations, and prop-tech has emerged, short for property technology which means the digital transformation in real estate industry (Baum et al., 2020). Prop-tech is a movement changing the shape and business model of the real estate industry including innovative technology and new products such as virtual reality (VR), Artificial intelligence (AI) and machine learning, building information modeling (BIM), internet of things (IoT), Blockchain, in addition to smart cities and homes which revolutionized the real estate industry. Property technology led to the development of smart real state which is the technology platforms for operating and managing real estate processes (GÓRSKA et al., 2022).

Prop-tech proved to be a key point for real estate companies to acquire competitive advantage in the industry through the use of these innovative technologies. key players real estate companies such as Zillow, Airbnb and WeWork proved that the impact of such business models can be huge (Bittini et al., 2022), these companies have gained the market share from traditional real estate agents by providing straightforward information aggregation proposition in addition to facilitating contractual offers for properties on their platforms. According to a research study the global prop-tech market share in terms of revenue was valued at $19.5 billion in 2022 and it is expected to reach around $32.2 billion by 2030 with a growth rate of 6.5% between 2023 to 2030 (Research, n.d.)(Yahoo Finance, 2023). The valuation process of properties and houses can be time consuming leading to several drawbacks such as wasting time, increase of uncertainty and delays in the sale processes. Prop-tech companies solved these problems through the utilization of machine learning for prediction individual property price which helps in reducing property selling and purchasing needed time in addition to increasing market transparency, and decreasing transaction costs which benefits both buyers, sellers and investors (Baum et al., 2020).

2.3 Utilizing machine learning in house price predictions.

Machine learning as subfield of artificial intelligence and an example of property technology proved to be a successful method for predicting house prices accurately as the traditional methods are complex for estimating house prices due to the presence of various of factors that can affect the valuation of house prices such as amenities, number of rooms, house conditions, number of bathrooms, condition of kitchen, and garages. Machine learning algorithm approved to be able to find hidden patterns and relationship addressing the most important house factors that affect house prices leading to accurate prediction of house prices. Accurate prediction of house prices has various benefits including helping buyers and sellers to make an informed decision regarding house prices (Kansal et al., 2023).

2.3.1 Supervised machine learning.

In supervised machine learning the training data set contains a target or desired solution which needs to be predicted according to understanding different features patterns. The supervised machine learning is classified into classification type such as prediction of emails whether it belongs to spam class or ham class, the other supervised type is called regression, and it is used when the target which needed to be predicted is numeric value. In this project regression techniques are used where the target is house prices and predictors are different house features (Géron, 2022).

2.3.2 Related work to house price prediction.

In (Truong et al., 2020) research, the authors applied both traditional and advanced machine learning techniques to examine the difference between several advanced models. Using dataset called "Housing Price in Beijing" containing 300,000 observations and 26 variables, they built and evaluated the following models including Random Forest, XGBoost, LightGBM, hyprid regression and stacked Genralization Regression. After data preprocessing, data splitting, model building and evaluation of models using Root Mean Square Logarithmic Error (RMSLE), the result showed that all of the method showed a desired outcome but each method has its own pros and cons, random forest had high time complexity and was prone to overfitting, XGBoost and LightGBM performed well in terms of time complexity, finally hybrid regression and stacked generalization showed high accuracy however they had high time complexity. The study suggests considering fast ways for complex models.

In (Abdul-Rahman et al., 2021) research they utilized advanced machine learning models for house prices prediction comparing two recent models, LightGBM and XGBoost using combined data set called "Property Listing in Kuala Lumpur" from Kaggle and Google Map with 21984 observation and 11 feature, the study used root mean square error (RMSE), mean absolute error (MAE),and adjusted r_squared value. They found that XGBoost outperformed LightGBM. They proven the reliability of the model by deploying it using another sample of data and the results showed a small variance between the actual and predicted price.

In (Ho et al., 2021) paper, authors used three machine learning techniques for house price prediction including support vector machine (SVM), random forest (RF), and gradient boosting machine (GBM). The models were applied on Hong Kong dataset. The result showed that (RMSE), absolute percentage error (MAPE), of Rf and GBM have achieved better performance than SVM model. However, the study admitted that SVM is useful in data fitting as it produces accurate prediction in tight time constraints. In conclusion the study agreed on the promising opportunity of machine learning property valuation, it also highlighted the importance of feature selection.

In (Hanuma Reddy & Sriramya, 2022) this study used and predicted real estate prices comparing performance of two model which are voting regressor and linear regression models. For the voting regression model the author combined three regression models to obtain final result including linear regression, decision tree, and random forest regression. After evaluating and training the models, the results showed that the voting regression outperformed the linear regression and achieved higher accuracy.

The authors (Zhou et al., 2023), focused on building effective models by comparing several models such as linear regression, random forest regressor, XGBoost, support vector machine (SVM) regressor, K-Nearest Neighbor (KNN), and linear regression. The results showed that Random Forest Regressor outperformed other models with R2 score 99.97% while the least performing model was SVM with R2 score -4.11%, the champion model for house prices prediction was Random Forest.

In (Kumar & Syed, 2023) research, they highlighted the importance of price prediction using machine learning and how a predicting model can benefit buyers to find fair purchase price as well as sellers to find the suitable selling house prices. They also used dataset of six different cities from Kaggle, consisting of 40 features for house price prediction. They developed and compared six machine learning models based on their accuracy including linear regression, random forest , decision tree, KNN , XGBoost, and SVM. The result showed that XGBoost outperformed the other five models with the highest accuracy.

In (Yadav et al., 2023), the paper aim was to examine the main characteristics of real estate property affect price and provide insights for each characteristics. In this study several machine learning models were built such as linear regression, Lasso regression, Decision tree, and GridSearchCV to predict costs of houses using Banglore dataset from Kaggle. The study focused on Linear regression results, which showed a score of 85% which was okay, then K-fold cross validation was used for optimality giving score 80%. The study results showed that linear regression had the highest prediction accuracy than other methods and recommended applying other methods in the future such as ensemble techniques for better accuracy.

In (El Mouna et al., 2023) study, they addressed house price predictions by applying three machine learning method including linear regression (LN), Random Forest (RF) and GradientBoosting (GB) on Melbourne real estate dataset which includes 34,857 sales and 21 house features. The models were evaluated using (RMSE), (MAE) and R- square. The results showed that Gradient Boosting outperformed the other two models. It calculated maximum R-square explaining most of variability in the dataset with 0.828. in addition, it calculated lower error values for RMSE and MAE.

In (Sibindi et al., 2023) study, as Boosting ensemble techniques are widely used in house price predictions, they focused on developing hybrid LGBM and XGBoost model to prevent overfitting through minimizing variance and improving accuracy the hybrid model was compared to LGBM, XGBoost, Adaboost, and GBM models and the models were evaluated using Mean Square Error (MSE), Mean Average Error (MAE), and Mean Absolute Percentage Error (MAPE). The results showed that the hybrid model outperformed other models achieving 0.193, 0.285 and 0.156 respectively.

The above literature review gives an overview of different machine learning algorithms and techniques utilized in prediction of house prices in the real estate industry on various datasets with different sizes and features. In addition to explaining the evaluation methods used for evaluating the performance of these models.

2.4 Regression machine learning models

There are various types of machine learning models, but this study focuses on the following machine learning regression models which are widely used in house price predictions.

Least absolute shrinkage and selection operator regression (LASSO):

LASSO is regularized version of linear regression. A unique characteristic of LASSO is that it eliminates the weight of less important features. It automatically performs features selection (Géron, 2022).

K-nearest neighbor regressor (KNN):

KNN was first invented by Evelyn Fix and Joseph Hodges in 1951. It is a supervised machine learning technique and one of the easiest algorithms. KNN uses similarity measures, it simply assumes that similar data points are near to each other. K in KNN refers to the number of neighbors of the new data points. It is essential to choose the accurate value of K as low value of K can lead to noisy results while high value can create confusion, although there is no fixed value for K, the standard is 5. In KNN The distance between data points is often measured using Euclidean distance. From KNN pros, is that it is easy and fast to apply and it is resistant to noise in the training data, however it is tricky to decide the accurate value of K and due to the calculation for the distance between all data points, it leads for high cost of computation (Bansal et al., 2022).

Support Vector Regressor (SVR):

SVR is a powerful model which has the ability to perform linear and nonlinear regressions. It works best with small to medium sized nonlinear datasets (hundreds to thousands of observations). However, it doesn't fit well with very large datasets. The main aim of SVR is to find the most suitable decision boundaries which is known as hyperplane that separates the data points while limiting margin violations (Géron, 2022) (Bansal et al., 2022).

Decision Tree Regressor (DTR):

Decision Tree is a powerful algorithm that is capable of fitting complex datasets. It is also a fundamental component to other machine learning algorithms such as random forest and XGBoost models. It has a tree-like structure where internal nodes represent a decision based on a feature and each leaf is a predicted outcome (Kansal et al., 2023) (Géron, 2022) (Vemuri, 2020).

Gradient Boosting Regressor (GBR):

It is a well-known boosting algorithm which is one of ensemble approaches that involves combining and training several weak learners (simple models) and capitalizing them into a strong learner (stronger model). Gradient Boosting works in a sequential manner in which each added predictor to the ensemble correct its predecessor through trying to fit the new predictor to the residual errors made by the previous predictor (Géron, 2022) (Vemuri, 2020).

Random Forest Regressor (RFR):

RFR is another ensemble algorithm that trains and combines multiple decision trees which resemble a forest. Each decision tree is trained on a unique dataset. Furthermore at each decision tree node, different subset of features are used for splitting which result in minimizing the influence of any single feature leading to the overall robustness of the model and achieving a high performance (Alyami et al., 2024) (Breiman, 2001) (Géron, 2022).

Extreme gradient boosting (XGBoost):

XGBoost is a tree boosting system. It is an ensemble learning technique that combines several decision trees. It is also known for its speed and performance when working with large datasets that consist of complex features. The model is designed to improve accuracy and strength of the model (Chen & Guestrin, 2016).

Light Gradient Boosting (LightGBM):

LightGBM is a gradient boosting decision tree model designed for large datasets and high dimensional features. It is designed to have the following advantages including faster training and prediction speed, higher efficiency, low memory usage, and higher accuracy (Ke et al., 2017).

Voting Regressor (VR):

Voting Regressor is another ensemble technique which combine and train numerous regression models for the purposes of aggregating the predictions of each one of the base models and produces a final prediction based on vote majority that results in an improved performance (Hanuma Reddy & Sriramya, 2022).

2.5 Model Evaluation Techniques

There are multiple performance measures for evaluating machine learning models that can be used to evaluate models, which are:

Root Mean Square Error (RMSE)

It is the most preferred performance measure for regression problems. It gives a good idea of how much error is represented in the model predictions by giving higher weights to large errors. However, if there are many outliers it's preferable to use another measure technique (Géron, 2022).

Mean Absolute Error (MAE)

It is a more suitable performance measure when there are many outliers as it is more sensitive than RMSE. It measures the difference between actual and predicted values (Géron, 2022).

R-Squared

It returns the coefficient of determination; it explains how much variability in the target variable is explained by the features variables. R-squared range between 0 to 1, the closer R^2 to 1 the more of the variance in the target variable is explained by the features (Albon, 2018).

Chapter 3

3.0 Methodology

The study's methodology and steps are explained in this chapter. The steps were performed using 'Google Collab' which is a tool that offers free access to Jupyter Notebook environment on cloud. In this study the whole process was conducted using python programming language. The methodology process starts from data retrieval, data exploration ,data pre-processing, feature engineering and finally, building and training models to figure out the champion model for predicting house prices including Least Absolute Shrinkage and Selection Operator (Lasso), K-Nearest Neighbor Regressor (KNN) , Support Vector Regressor (SVR) , Decision Tree Regressor (DTR) , Random Forest Regressor (RFR), Gradient Boosting Regressor (GBR), eXtreme Gradient Boosting (XGBoost) , Light Gradient Boosting Machine Regressor (LGBM), Voting Regressor.

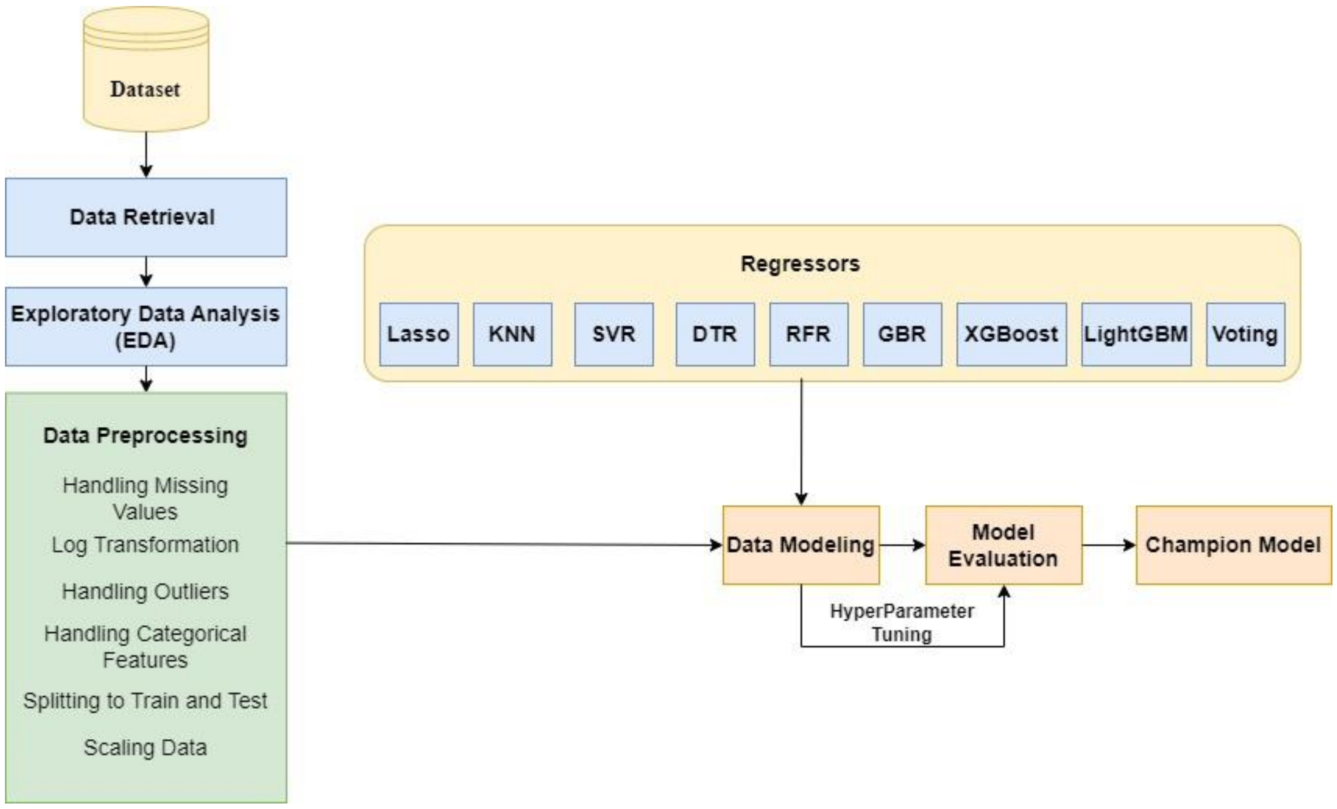


Figure 3. 1 Methodology workflow.

3.1 Data Retrieval

The dataset used for this study is called 'Ames Housing dataset' that was downloaded from Kaggle website in the form of comma-separated value (CSV) file format. The data set contains home sales that occurred in Ames, Iowa between 2006 and 2010 with 1460 observations and 79 explanatory features (categorical and numerical) describing residential houses in Ames, Iowa along with each house's sale price as the target variable. The data was imported to python using **read_csv( )** which is used to read csv files. The following table contains descriptions for each feature included in the dataset.

Tabel 3. 1 Description of dataset features.

Feature	Description	Feature	Description
• **MSSubClass**	Identifies the type of dwelling involved in the sale.	• **MSZoning**	Identifies the general zoning classification of the sale.
• **LotFrontage**	Linear feet of street connected to property.	• **LotArea**	Lot size in square feet.
• **Street**	Type of road access to property.	• **Alley**	Type of alley access to property.
• **LotShape**	General shape of property.	• **LandContour**	Flatness of the property.
• **Utilities**	Type of utilities available.	• **LotConfig**	Lot configuration.
• **LandSlope**	Slope of property.	• **Neighborhood**	Physical locations within Ames's city limits.
• **Condition1**	Proximity to various conditions.	• **Condition2**	Proximity to various conditions.
• **BldgType**	Type of dwelling.	• **HouseStyle**	Style of dwelling.
• **OverallQual**	Rates the overall material and finish of the house.	• **OverallCond**	Rates the overall condition of the house.
• **YearBuilt**	Original construction date.	• **YearRemodAdd**	Remodel date.
• **RoofMatl**	Roof material.	• **Exterior1st**	Exterior covering on house.
• **Exterior2nd**	Exterior covering on house.	• **MasVnrType**	Masonry veneer type.
• **MasVnrArea**	Masonry veneer area in square feet.	• **ExterQual**	Evaluates the quality of the material on the exterior.
• **ExterCond**	Evaluates the present condition of the material on the exterior.	• **Foundation**	Type of foundation.
• **BsmtQual**	Evaluates the height of the basement.	• **BsmtCond**	Evaluates the general condition of the basement.
• **BsmtExposure**	Refers to walkout or garden level walls.	• **BsmtFinType1**	Rating of basement finished area.
• **BsmtFinSF1**	Type 1 finished square feet.	• **BsmtFinType2**	Rating of basement finished area.
• **BsmtFinSF2**	Type 2 finished square feet.	• **BsmtUnfSF**	Unfinished square feet of basement area.

• **TotalBsmtSF**	Total square feet of basement area.	• **Heating**	Type of heating.
• **HeatingQC**	Heating quality and condition.	• **CentralAir**	Central air conditioning.
• **Electrical**	Electrical system.	• **1stFlrSF**	First Floor square feet.
• **2ndFlrSF**	Second floor square feet.	• **LowQualFinSF**	Low quality finished square feet (all floors).
• **GrLivArea**	Above grade (ground) living area square feet.	• **BsmtFullBath**	Basement full bathrooms.
• **BsmtHalfBath**	Basement half bathrooms.	• **FullBath**	Full bathrooms above grade.
• **HalfBath**	Half baths above grade.	• **Bedroom**	Bedrooms above grade.
• **Kitchen**	Kitchens above grade.	• **KitchenQual**	Kitchen quality.
• **TotRmsAbvGrd**	Total rooms above grade.	• **Functional**	Home functionality.
• **Fireplaces**	Number of fireplaces.	• **FireplaceQu**	Fireplace quality.
• **GarageType**	Garage location.	• **GarageYrBlt**	Year garage was built.
• **GarageFinish**	Interior finish of the garage.	• **GarageCars**	Size of garage in car capacity.
• **GarageArea**	Size of garage in square feet.	• **GarageQual**	Garage quality.
• **GarageCond**	Garage condition.	• **PavedDrive**	Paved driveway.
• **WoodDeckSF**	Wood deck area in square feet.	• **OpenPorchSF**	Open porch area in square feet.
• **EnclosedPorch**	Enclosed porch area in square feet.	• **3SsnPorch**	Three season porch area in square feet.
• **ScreenPorch**	Screen porch area in square feet.	• **PoolArea**	Pool area in square feet.
• **PoolQC**	Pool quality.	• **Fence**	Fence quality.
• **MiscFeature**	Miscellaneous feature not covered in other categories.	• **MiscVal**	Value of miscellaneous feature.
• **MoSold**	Month Sold (MM).	• **YrSold**	Year Sold (YYYY).
• **SaleType**	Type of sale.	• **SaleCondition**	Condition of sale.

3.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis is an essential step for every project and study. It was applied for exploring, investigating, and summarizing data to uncover patterns and trends to understand data fundamentals. It is the step preceding advanced analysis and modelling which gives a basic view about the data. (*Pandas Documentation — Pandas 2.1.4 Documentation*, n.d.)

3.2.1 Basic Overview About the Data

In order to have a quick understanding of the dataset we implemented the following steps:

We used **house_df.head( )** function to retrieve the first five rows of the data , **house_df.columns** to give the complete lists of column names, **house_df.shape** function to retrieve the number of rows and columns of the data, we used all the previous to have a quick overview of the data columns and observations. Then, we used **house_df.duplicated( ).sum( )** function to check for possible duplicates represented in the data.

The Id column is useless for our study, we dropped it using the following function **house_df.drop(['Id'], axis = 1, inplace = True).**

To further explore the data, we used **house_df.info( )** function to return comprehensive information about the data such as names, non-null count, and data type of each column. Additionally, we used **house_df.describe( ).T** that generates descriptive analysis for the data including count of observations, mean, standard deviation, minimum, maximum, 25% quartile, 50% quartile, and 75% quartile for each feature in the dataset to understand central of tendency, dispersion, and shape of data. In addition to using the describe( ) function for the whole dataset, it was used to focus on our target variable 'SalePrice' as well **house_df['SalePrice'].describe( )**. The mean (180921.195890) resulted from SalePrice descriptive analysis is higher than the median (163000.000000) which suggests a right skewed distribution, so we needed to visualize it later in this chapter. The following schedule explains all descriptive analysis values for SalePrice.

Tabel 3. 2 Sale Price Descriptive Analysis

Measure	values
Count	1460.000000
Mean	180921.195890
STD	79442.502883
Min	34900.000000
25%	129975.000000
50%	163000.000000
75%	214000.000000
max	755000.000000

Furthermore, to understand our target distribution, we visualized it using histogram plot function **sns.histplot(house_df['SalePrice'])** to reveal its skewness pattern that was right skewed and will be altered later in this chapter, it is represented by figure 3.2. We also measured its variance using **house_df['SalePrice'].var( )** function to understand how much variability exists in the target variable.

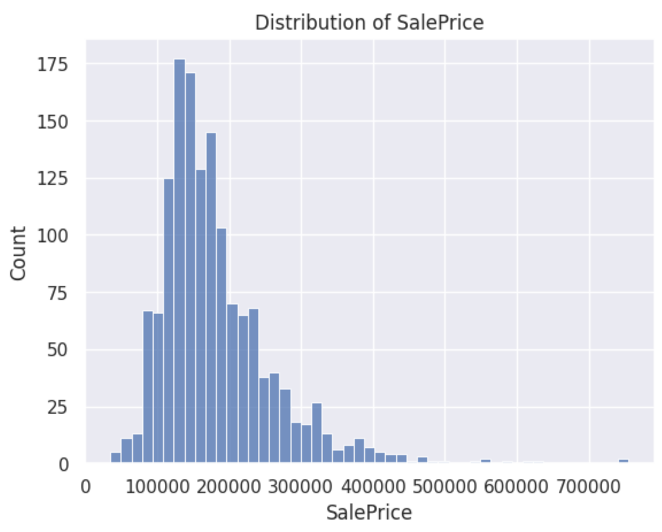


Figure 3. 2 Sale Price Histogram.

Then, we retrieved the numerical and categorical features names, they were 37 numerical features and 43 categorical variables. We used horizonal bar plot **sns.barplot( )** to visualize all categorical features against SalePrice to clearly understand their values and their unique classes. We also visualized numerical features using pair plot function **sns.pairplot( )** that showed features outliers that ought to be removed in addition to their relationship with each other and with the SalePrice. We used heatmap **sns.heatmap( )** function to understand numerical features and their relationship and correlations with SalePrice feature, as well as calculating correlation matrix using **corr( )** function, correlation ranges between 1 and -1 the closest the correlation to 1 the strongest. The following table contains the top 10 positive correlated variables with SalePrice feature including OverallQual, GrLivArea, GarageCars, GarageArea, TotalBsmtSF, 1stFlrSF, FullBath, TotRmsAbvGrd, YearBuilt, and YearRemodAdd which are highly correlated with SalePrice. We can find that overall house quality has the strongest correlation with house sale price.

Tabel 3. 3 Top 10 Positive Correlated Features

Feature	Positive Correlation
OverallQual	0.790982
GrLivArea	0.708624
GarageCars	0.640409
GarageArea	0.623431

TotalBsmtSF	0.613581
1stFlrSF	0.605852
FullBath	0.560664
TotRmsAbvGrd	0.533723
YearBuilt	0.522897
YearRemodAdd	0.507101

3.3 Data Preprocessing

After retrieving the data and performing exploratory data analysis to get initial insights, a data preprocessing was performed to prepare the data to be in a suitable format for the machine learning models.

3.3.1 Handling Missing values

Most machine learning models cannot work with features that contain missing values so, they need to be handled, in order for machine learning models to work accurately. We calculated the null values for every feature, with their percentages and their data type in order to be able to see the full picture. We found that 19 features out of 79 contained null values, and they were handled differently according to the number of null values, their data type, and the data description, we didn’t want to remove any information as the data is relatively small and according to real estate every information should be considered. The following table represents features that had null values and how they were handled.

Tabel 3. 4 Features Missing Values

Feature	No. Nulls	Replaced with	Feature	No. Nulls	Replaced with
PoolQC	1453	None (no pool)	GarageType	81	None (no garage)
MiscFeature	1406	None (no extra feature)	GarageFinish	81	None
Alley	1369	None (no alley)	GarageQual	81	None
Fence	1179	None (no fence)	BsmtExposure	38	None (No basement)
FireplaceQu	690	None	BsmtFinType2	38	None

		(no fireplace)			
LotFrontage	259	Median	BsmtCond	37	None
GarageYrBlt	81	Median	BsmtQual	37	None
GarageCond	81	None	BsmtFinType1	37	None
MasVnrArea	8	Median	MasVnrType	8	Mode
Electrical	1	Mode			

3.3.2 Log Transformation

In our EDA the target variable (SalePrice) was heavily skewed to the right and many machine learning models assume normal distribution of the target variable. Accordingly, the target variable was log transformed using **np.log( )** NumPy function, so that it can be normally distributed or close to normal. We can use **np.exp( )** later after prediction to reverse the transformation for predicted values. The following figures represent target variable before and after transformation.

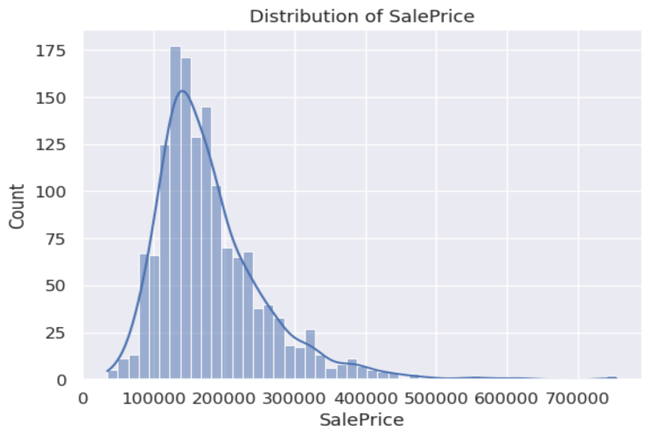


Figure 3. 3 Sale Price Before Log Transformation.

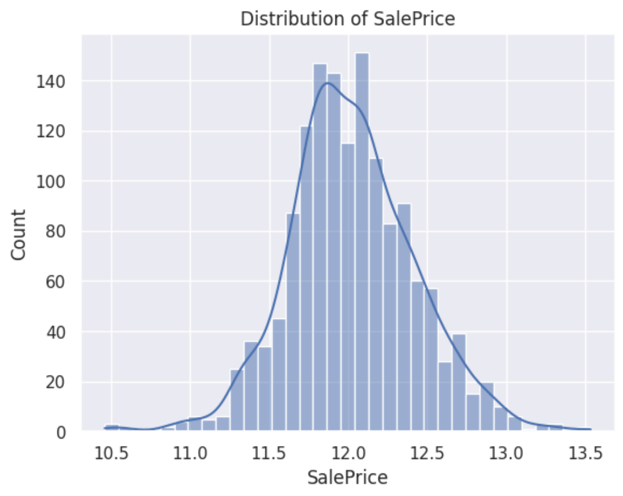


Figure 3. 4 Sale Price After Log Transformation.

3.3.3 Handling Outliers

Outliers are data points which are extremely high or low and deviate from other data points that result from data entry error, computation error or natural outliers and can lead to skewed and inaccurate predictions. We defined a range of acceptable values between the 1st and 3rd quartile by using interquartile range (IQR) value. The values that fall outside the range are considered outliers and they can be treated either by removing or using a capping method. In this study we used capping methods. The capping method was used to handle outliers through setting threshold value and any outlier that is beyond the threshold is replaced with a specific value, as a result it preserved information of the outlier instead of removing them. We visualized each numerical feature using Boxplot before and after using capping method, for checking the capping effect on features that contained outliers. The following Figures represent outliers before and after handling them.

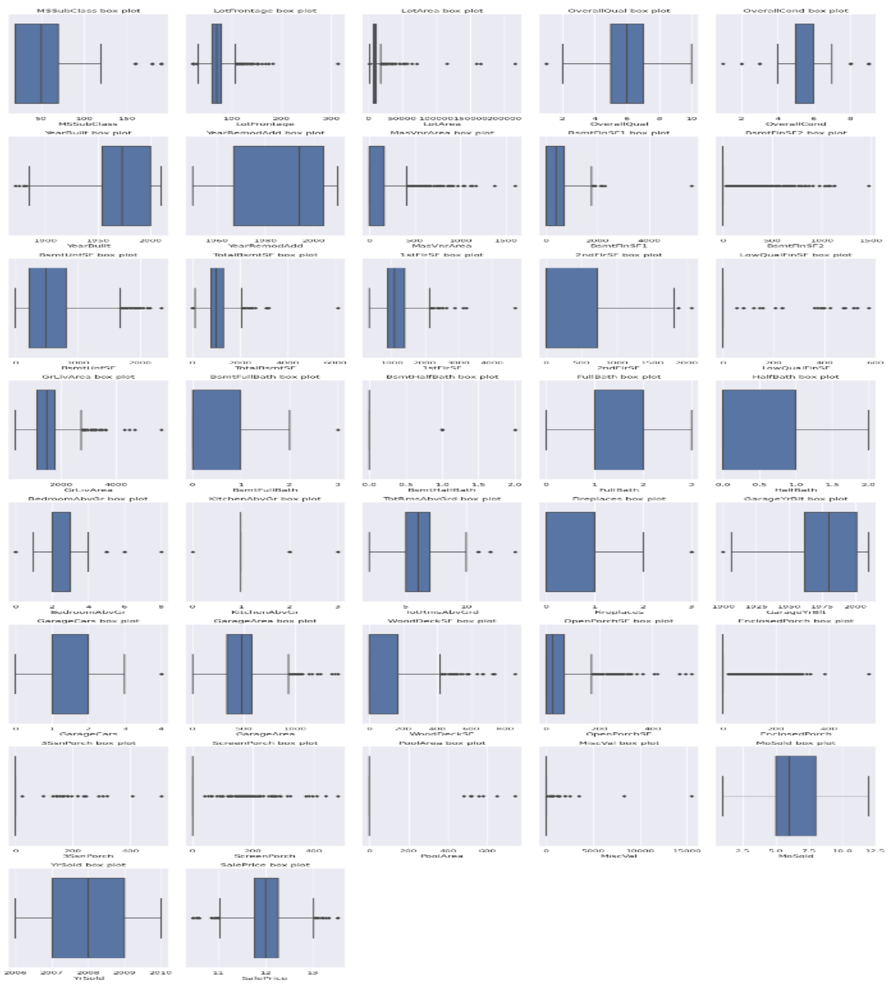


Figure 3. 5 Features with Outliers.

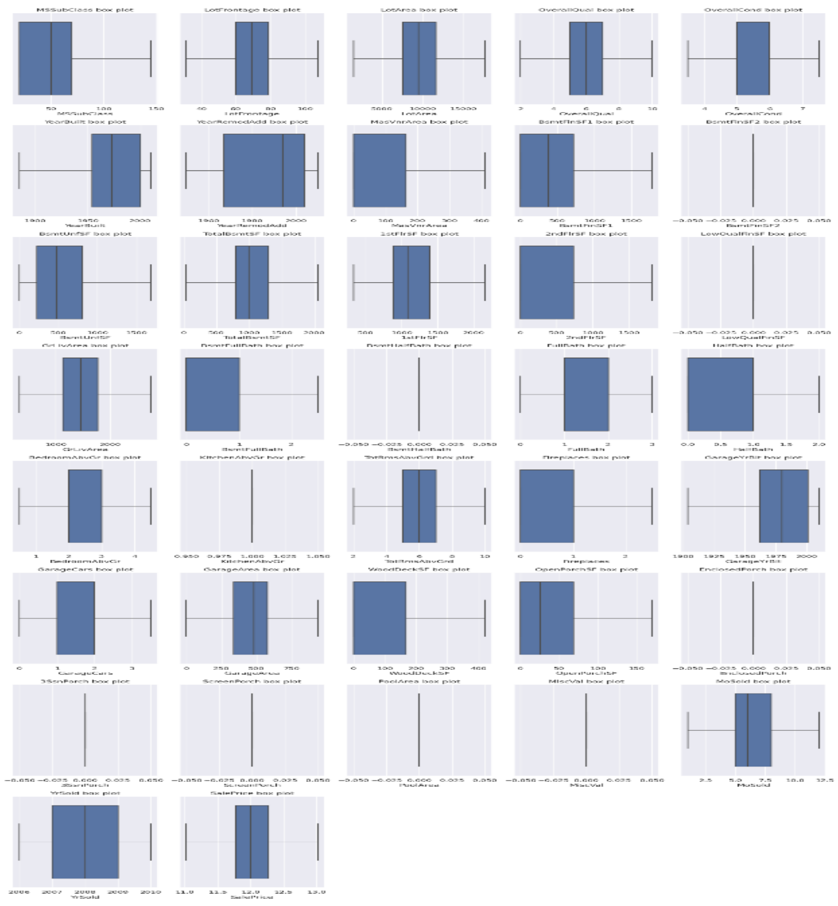


Figure 3. 6 Features Without Outliers.

3.3.4 Handling Categorical Features

Since, our dataset contains a lot of categorical features and most machine learning models can only take numerical values as input , they must be encoded into numerical values before being used to test and train the models. We used **LabelEncoder( )** from sklearn.preprocessing library to encode categorical features where each category within the feature is assigned to a unique integer starting from 0 to number of classes.

3.3.5 Splitting the Data

Before building models, the dataset needs to be split into train set and test set. The train set is used to train and fit the model while test set is used to evaluate the model performance. In this study the dataset was split into 80% for model training and 20% for model testing. We split the data using **train_test_split( )** to split the data into X_train, X_test, y_triain, and y_test where X was the features with dropping the target and y was the target.

3.3.6 Scaling and Splitting the Data

Feature Scaling is one of the most important transformations that need to be applied to our data. When the data have very different scales, machine learning models cannot perform well so they need to be transformed. In this study we used **StandardScaler( )** for scaling our features. It is important to fit StrandardScaler to the training data only and not the testing data in order to prevent data leakage which is the transferring of information from training set to testing set.

3.4 Modelling and Hyperparameter Tuning

In this study, we built nine different machine learning models in order to compare them and choose the highest performance model to be used for house price predictions including Lasso, KNN, SVR, DTR , RFR, GBR, XGBoost, LGBM, Voting Regressor. We used **cross_val_score ( )** function on each model in order to evaluate models' performance through cross validation which divide training data into multiple folds usually 5 or 10 to be trained on each of the fold and evaluates model performance on the held-out fold to be able to see how our model would perform on different datasets through providing evaluation score such as root mean square error. We also used Randomized Search cross validation for hyperparameter tuning of models using **RandomizedSearchCV( )** which select random value for each hyperparameter at every iteration which can improve our model's performance and has reduced computational costs than other techniques. Different values were used for each hyperparameter. For model evaluation performance we used three techniques which are Mean Average Error, Root Mean Square Error, and R^2 square.

Model building steps that were applied for each model:

Model = model( ) – to create instance of model.

Model.fit(X_train_scaled, y_train) – to fit the model on training data.

Model_y_predict = model.predict(X_test_scaled) – to use model for prediction.

The following functions were used for each model:

- Lasso( )
- KNeighborsRegressor( )
- SVR( )
- DecisionTreeRegressor( )
- RandomForestRegressor( )
- GradientBoostingRegressor( )
- XGBRegressor( )
- lgb.LGBMRegressor( )
- VotingRegressor( )

Chapter 4

4.0 Findings

This chapter contains the final result regarding the previously built models for Ames Housing dataset house price predictions using the parameters represented in the following table. The models were evaluated based on three key performance metrics which are R² squared, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) that are widely used in various regressions as they provide comprehensive view of model's performance.

4.1 Hyperparameter tuning results

We used the highlighted parameters in the following table which resulted from Randomized Search cross validation technique to enhance models' performance while for some models such as SVR and RFR hyperparameter tuning decreased their performance so, we used default parameters for each of them.

Tabel 4. 1 Hyperparameter Tuning Values.

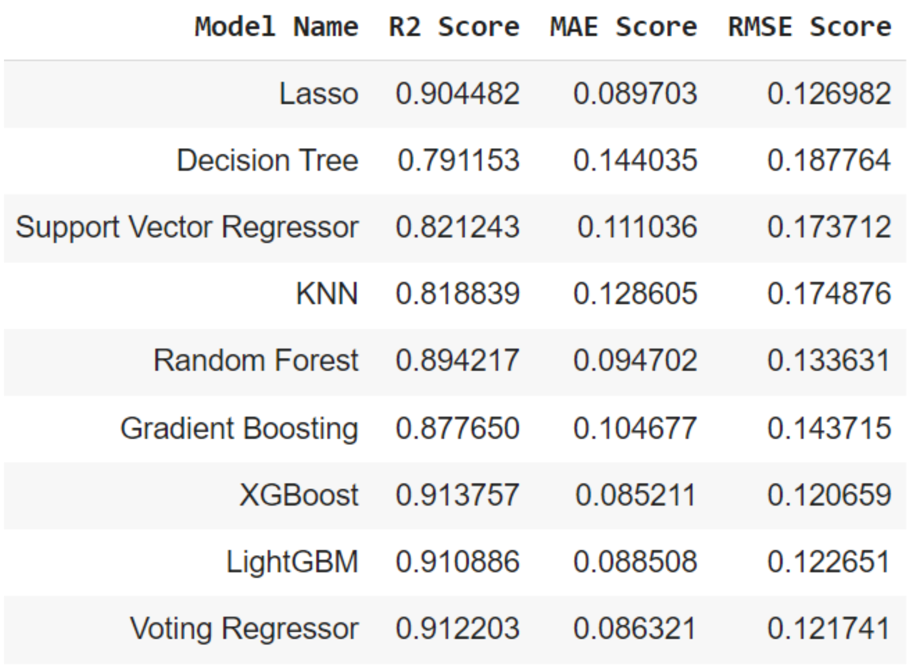
Model	Hyperparameter	Values
Lasso	alpha	[**0.001**,0.01,0.1,0.000391,1,10,100,1000]
	random_state	[1,5,**10**,24,42]
KNN	n_neigbors	[5,6**,7,**8,9,10,11,12,13,14…20]
SVR	Default parameter	Default values
DTR	max_depth	[1,2,3,4,4,6,7,8….**20**,21]
	min_samples_split	[2,3,4,4,6,7,8,9,**10**,11]
	min_samples_leaf	[1,2,3,4,4,6,7,8,**9**,10,11]
	max_features	[20,30,**50**]
RFR	Default parameter	Default values
GBR	n_estimators	[2,3,4,5,6,7,20,**30**,50]
	alpha	[0.001,0.1,**0.5**,0.2]
XGBoost	n_estimators	[100,200,300,500,2200,**220**]
	learning_rate	[0.01,0.03,**0.05**,0.1,0.001]
	max_depth	[3,4,**5**,6]
	min_child_weight	[1,2,3,4,**1.7817**,0.178,0.0178]

	gamma	[**0**,0.1,0.2,0.3,0.005,0.0045,0.35,0.5]
	colsample_bytree	[0.6,**0.7**,0.8,0.9]
	alpha	[0,0.1,**0.5**,1]
	random_state	**42**
LGBM	learning_rate	[0.7,0.8,0.1,0.001,**0.2**,0.0003,0.3,0.5]
	max_depth	[3,4,5,**6**,50]
	n_estimators	[6,7,10,11,**50**,60]
	num_leaves	[5,**6**,7]
	reg_alpha	[**0**,0.1,0.5,1]
Voting	Weights:	[0.2,0.6,0.2]
	[gbr_model,xgb_model,lgb_model]	[0.6,0.2,0.2][0.40,0.40,0.40]**[0.20,0.40,0.50]**

4.2 Model Performance

The following table and figures summarize each model performance:

Tabel 4. 2 Model Evaluation Comparison.



Model Name	R2 Score	MAE Score	RMSE Score
Lasso	0.904482	0.089703	0.126982
Decision Tree	0.791153	0.144035	0.187764
Support Vector Regressor	0.821243	0.111036	0.173712
KNN	0.818839	0.128605	0.174876
Random Forest	0.894217	0.094702	0.133631
Gradient Boosting	0.877650	0.104677	0.143715
XGBoost	0.913757	0.085211	0.120659
LightGBM	0.910886	0.088508	0.122651
Voting Regressor	0.912203	0.086321	0.121741

The result showed that XGBoost model comes in the first place as the top performing model achieving the highest R² squared score (0.913757), lowest MAE score (0.085211) and lowest RMSE score (0.120659). The Second-best performing model is the voting regressor which combines Gradient boosting, XGBoost, and LightGBM models achieving high R² squared score (0.912203), low MAE score (0.086321) and low RMSE (0.121741) . Also, LightGBM, and Lasso showed a strong performance achieving R² squared score above 0.90.

while Decision Tree model considered to be the lowest performing model achieving lowest R² squared score (0.791153), highest MAE score (0.144035) and lowest RMSE score (0.187764). For further investigation of the most features that were used by XGBoost that contributed the most for predicting house prices, we used the following code xgb_model.feateaure_importances_ our champion model**,** as a result we found that top 5 most important features that contributed the most in our model predictive power which are OverallQual (30%) which is overall quality of the house, ExterQual (14%) which is exterior quality, GarageCars (11%) which is size of garage in terms of number of cars, GrLivArea (5%) which is above ground area and KitchenQual (5%) which is kitchen quality.

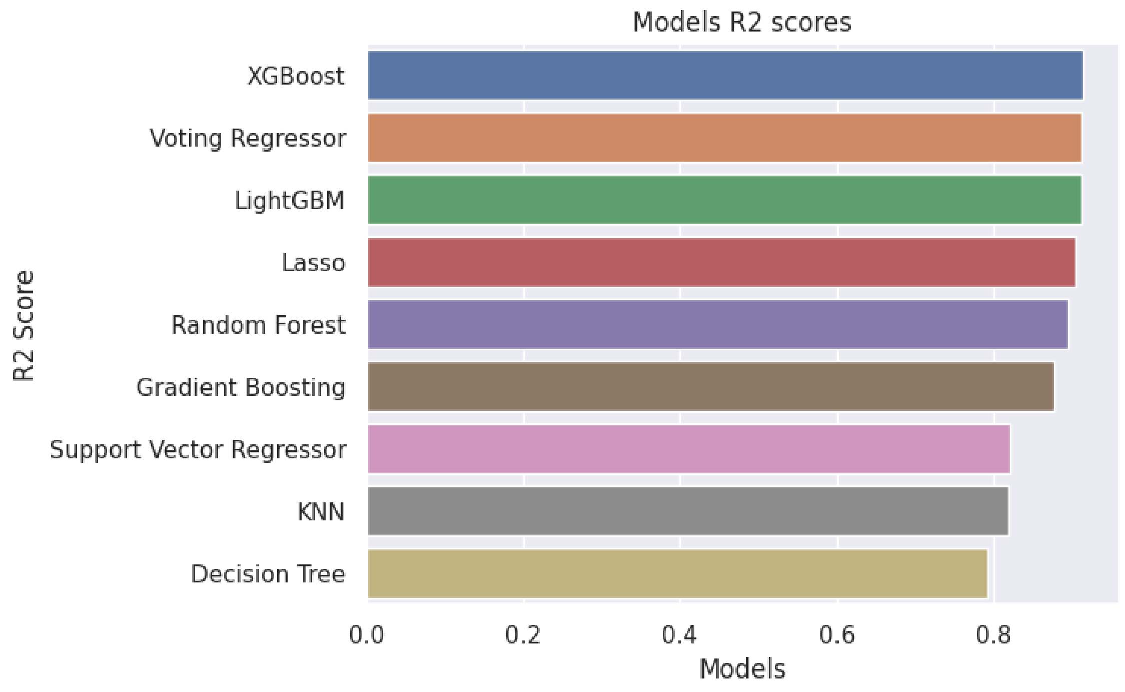


Figure 4. 1 Models R2 Scores.

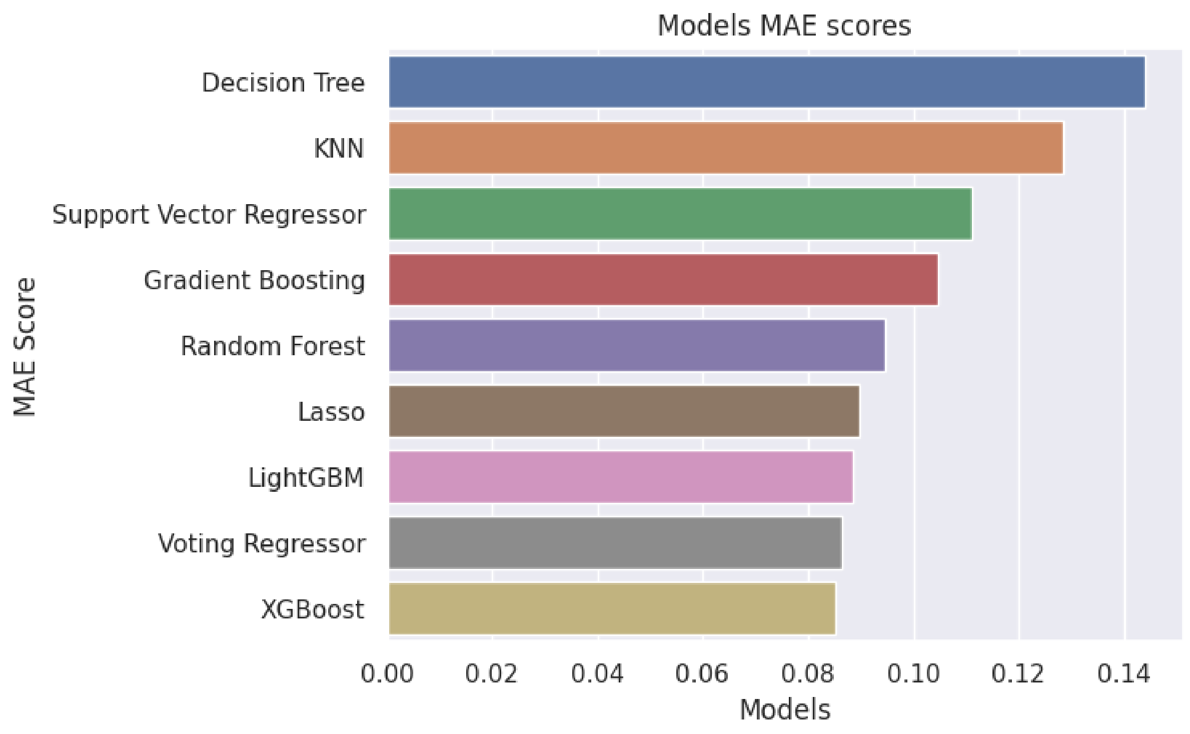


Figure 4. 2 Models MAE Scores.

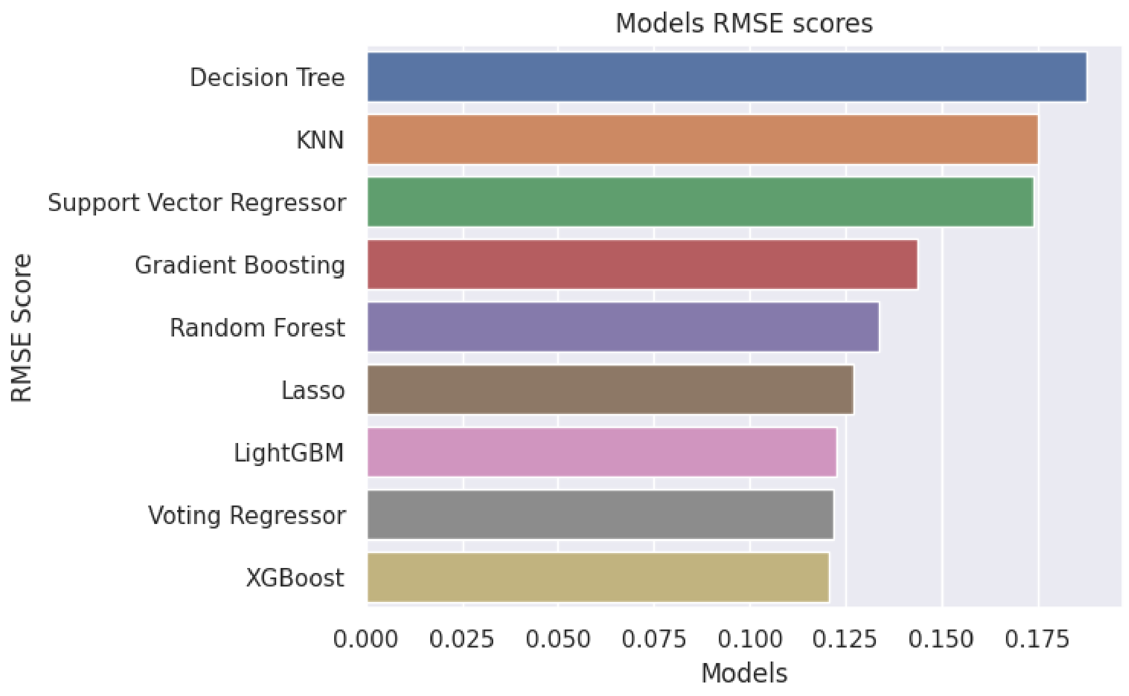


Figure 4. 3 Models RMSE Scores.

4.4 Discussion

The XGBoost is the champion model, its performance results suggest that it is a powerful ensemble model that was effective in capturing the underlying patterns within the Ames Housing dataset. Furthermore, the strong performance of other advanced models including Voting Regressor which combined between (gradient boosting, XGboost and LightGBM) models, as well as LightGBM performance. All the previous mentioned models concludes that the house price predictions can be quite complicated, and advanced regression methods can be used as they combine the advantages of various models for predictions.

Chapter 5

5.0 Conclusion

In this study we covered a lot of house price predictions aspects. We gave a brief overlook to fourth industrial revolution and property technology impact on Real Estate industry. In addition, to mentioning previous research about house price prediction. We also walked through the importance and process of house price prediction using Ames' housing dataset from Kaggle.

We retrieved the data and preformed exploratory data analysis to have a basic understand to it, then we conducted necessary preprocessing techniques including missing values imputation, feature engineering, log transformation the target variable, handling outliers, splitting, and standardizing data to prepare it in a suitable format that can be used for modelling.

Then, we built nine popular models including linear, non-linear and ensemble models for house price predictions, we performed cross validation for every model to see how the models will perform on unseen data, in addition to hyperparameter tuning using randomized search cross validation technique to enhance models' performance.

After constructing the models, we evaluated them using R^2 squared, MAE, and RMSE metrics. According to our final model evaluation XGBoost was our champion model through achieving highest R^2, lowest MAE and RMSE followed by Voting model achieving second best performance. While the lowest performing model was Decision Tree model the achieved lowest R^2 and highest MAE and RMSE. It is important to understand that that there is no single model that can work for every problem case, the selection of appropriate model depends on several factors including data quality, and problem complexity.

Finally, Ames, Iowa city can benefit from using the XGBoost model in the process of house price predictions as it will make the process more efficient and save lots of resources including time, effort and cost.

5.1 Limitations

This study contains some limitations. First, the small size of data which only contains 1460 observations which can result in lack of capturing the diversity and variability of house market trends that could affect predictions. Furthermore, small dataset can affect models' ability to generalize well on unseen data , as well as inability to fully capture complex patterns.

Second, the data contains a large number of subjective features including but not limited to OverallOual, ExterQual, KitchenQual, and BsmtQual which are rated poor to excellent which can be exposed to bias. Moreover, the dataset lacks necessary features that may also affect the house prices such as environmental conditions, population, and proximity to desired amenities (transportation, schools, parks, and markets).

Third, the same data structure and features are necessary in order to be able to use the model built in this study for future predictions.

Fourth, as house price prediction is important, however it's not enough and needs the support of experience in the real estate field, which will be complementary to the prediction.

Lastly, that Real estate market is a dynamic industry and the dataset used in this study doesn't cover wide range of years, using a dataset that covers longer timeframe can be more effective and efficient in understanding house prices.

5.2 Recommendations

There are a number of recommendations that could be made. First, more feature engineering can be done to enhance model performance, in addition to trying different hyperparameter tuning techniques that can work better.

Second, this study can be repeated using more advanced models such as Neural Network and deep learning, which mainly wasn't applied due to small size of data.

Third, more study can be performed to enhance house price predictions process to be recognized as a successful opportunity for real-estate companies and to encourage them to use property technology.

Lastly, for the future a website or mobile application for house price predictions can be built and used in order to help buyers, investors and agencies to better understand the real estate market and not to be scammed with incorrect property prices or costs.

References

Abdul-Rahman, S., Zulkifley, N. H., Ismail, I., & Mutalib, S. (2021). Advanced machine learning algorithms for house price prediction: Case study in Kuala Lumpur. *International Journal of Advanced Computer Science and Applications*, *12*(12). https://search.proquest.com/openview/a0ba078716308553044e6f39d9131222/1?pq-origsite=gscholar&cbl=5444811

Albon, C. (2018). *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning*. O'Reilly Media, Inc. https://books.google.com/books?hl=en&lr=&id=kIhQDwAAQBAJ&oi=fnd&pg=PT107&dq=Machine+Learning+with+Python+Cookbook+:+Practical+Solutions+&ots=OoQt0KhhJO&sig=_T6fmYAUyvdCxPYZalOTFgg-P_o

Alyami, M., Khan, M., Fawad, M., Nawaz, R., Hammad, A. W. A., Najeh, T., & Gamil, Y. (2024). Predictive modeling for compressive strength of 3D printed fiber-reinforced concrete using machine learning algorithms. *Case Studies in Construction Materials*, *20*, e02728. https://doi.org/10.1016/j.cscm.2023.e02728

Bai, C., Dallasega, P., Orzes, G., & Sarkis, J. (2020). Industry 4.0 technologies assessment: A sustainability perspective. *International Journal of Production Economics*, *229*, 107776. https://doi.org/10.1016/j.ijpe.2020.107776

Bansal, M., Goyal, A., & Choudhary, A. (2022). A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning. *Decision Analytics Journal*, *3*, 100071. https://doi.org/10.1016/j.dajour.2022.100071

Baum, A., Saull, A., & Braesemann, F. (2020). PropTech 2020: The future of real estate. *Said Business School. University of Oxford Research. Search In*.

Bittini, J. S., Rambaud, S. C., Pascual, J. L., & Moro-Visconti, R. (2022). Business Models and Sustainability Plans in the FinTech, InsurTech, and PropTech Industry: Evidence from Spain. *Sustainability (Switzerland)*, *14*(19). Scopus. https://doi.org/10.3390/su141912088

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. https://doi.org/10.1145/2939672.2939785

City of Ames, IA | Home. (n.d.). Retrieved January 24, 2024, from https://www.cityofames.org/home

cycles, T. text provides general information S. assumes no liability for the information given being complete or correct D. to varying update, & Text, S. C. D. M. up-to-D. D. T. R. in the. (n.d.). *Topic: Proptech*. Statista. Retrieved December 3, 2023, from https://www.statista.com/topics/5230/proptech/

Dogaru, L. (2020). The Main Goals of the Fourth Industrial Revolution. Renewable Energy Perspectives. *Procedia Manufacturing*, *46*, 397–401. https://doi.org/10.1016/j.promfg.2020.03.058

El Mouna, L., Silkan, H., Haynf, Y., Nann, M. F., & Tekouabou, S. C. K. (2023). *A Comparative Study of Urban House Price Prediction using Machine Learning Algorithms*. *418*. Scopus. https://doi.org/10.1051/e3sconf/202341803001

Geerts, M., vanden Broucke, S., & De Weerdt, J. (2023). A Survey of Methods and Input Data Types for House Price Prediction. *ISPRS International Journal of Geo-Information*, *12*(5), Article 5. https://doi.org/10.3390/ijgi12050200

Géron, A. (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow.* O'Reilly Media, Inc.

GÓRSKA, A., MAZURCZAK, A., & STRĄCZKOWSKI, \Lukasz. (2022). IMPLEMENTATION OF MODERN TECHNOLOGIES (PROPTECH) BY DEVELOPERS ON THE LOCAL HOUSING MARKET. *Scientific Papers of Silesian University of Technology. Organization & Management/Zeszyty Naukowe Politechniki Slaskiej. Seria Organizacji i Zarzadzanie*, *162*. https://managementpapers.polsl.pl/wp-content/uploads/2022/12/162-G%C3%B3rska-Mazurczak-Str%C4%85czkowski-2.pdf

Hanuma Reddy, G., & Sriramya, P. (2022). Real Estate Price Prediction and Analysis Using Voting Regression Compared with Linear Regression. In *Advances in Parallel Computing Algorithms, Tools and Paradigms* (pp. 625–630). IOS Press. https://doi.org/10.3233/APC220089

Ho, W. K. O., Tang, B.-S., & Wong, S. W. (2021). Predicting property prices with machine learning algorithms. *Journal of Property Research*, *38*(1), 48–70. https://doi.org/10.1080/09599916.2020.1832558

Kansal, M., Singh, P., Shukla, S., & Srivastava, S. (2023). A Comparative Study of Machine Learning Models for House Price Prediction and Analysis in Smart Cities. In F. Ortiz-Rodríguez, S. Tiwari, P. Usoro Usip, & R. Palma (Eds.), *Electronic Governance with Emerging Technologies* (pp. 168–184). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-43940-7_14

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 3149–3157.

Kumar, S., & Syed, M. H. (2023). House Pricing Prediction Based on Composite Facility Score Using Machine Learning Algorithms. In P. K. Singh, S. T. Wierzchoń, S. Tanwar, J. J. P. C. Rodrigues, & M. Ganzha (Eds.), *Proceedings of Third International Conference on Computing, Communications, and Cyber-Security* (pp. 235–248). Springer Nature. https://doi.org/10.1007/978-981-19-1142-2_18

Machkour, B., & Abriane, A. (2020). Industry 4.0 and its Implications for the Financial Sector. *Procedia Computer Science*, *177*, 496–502. https://doi.org/10.1016/j.procs.2020.10.068

pandas documentation—Pandas 2.1.4 documentation. (n.d.). Retrieved January 8, 2024, from https://pandas.pydata.org/docs/index.html

Putatunda, S. (2019). *PropTech for Proactive Pricing of Houses in Classified Advertisements in the Indian Real Estate Market* (arXiv:1904.05328). arXiv. https://doi.org/10.48550/arXiv.1904.05328

Research, Z. M. (n.d.). *PropTech Market Size, Share, Trends, Demand, Applications 2023-2030*. Zion Market Research. Retrieved January 11, 2024, from https://www.zionmarketresearch.com/report/proptech-market

Saiz, A. (2020). Bricks, mortar, and proptech: The economics of IT in brokerage, space utilization and commercial real estate finance. *Journal of Property Investment & Finance*, *38*(4), 327–347. https://doi.org/10.1108/JPIF-10-2019-0139

Sibindi, R., Mwangi, R. W., & Waititu, A. G. (2023). A boosting ensemble learning based hybrid light gradient boosting machine and extreme gradient boosting model for predicting house prices. *Engineering Reports*, *5*(4), e12599. https://doi.org/10.1002/eng2.12599

Starr, C. W., Saginor, J., & Worzala, E. (2020). The rise of PropTech: Emerging industrial technologies and their impact on real estate. *Journal of Property Investment & Finance*, *39*(2), 157–169. https://doi.org/10.1108/JPIF-08-2020-0090

Truong, Q., Nguyen, M., Dang, H., & Mei, B. (2020). Housing price prediction via improved machine learning techniques. *Procedia Computer Science*, *174*, 433–442.

Vemuri, V. K. (2020). The Hundred-Page Machine Learning Book: By Andriy Burkov, Quebec City, Canada, 2019, 160 pp., $49.99 (Hardcover); $29.00 (paperback); $25.43 (Kindle Edition), (Alternatively, can purchase at leanpub.com at a minimum price of $20.00), ISBN 978-1999579517. *Journal of Information Technology Case and Application Research*, *22*(2), 136–138. https://doi.org/10.1080/15228053.2020.1766224

Xu, M., David, J. M., & Kim, S. H. (2018). The fourth industrial revolution: Opportunities and challenges. *International Journal of Financial Research*, *9*(2), 90–95.

Yadav, S., Dhanda, N., Sahai, A., Verma, R., & Pandey, S. (2023). Real Estate Price Prediction Using Machine Learning. In D. K. Sarkar, P. K. Sadhu, S. Bhunia, J. Samanta, & S. Paul (Eds.), *Proceedings of the 4th International Conference on Communication, Devices and Computing* (pp. 95–111). Springer Nature. https://doi.org/10.1007/978-981-99-2710-4_9

Zhou, J., Hai, T., Maxwell-Chigozie, E. C., Adedayo, A., Chen, Y., Iwendi, C., & Boulouard, Z. (2023). Effective House Price Prediction Using Machine Learning. In C. Iwendi, Z. Boulouard, & N. Kryvinska (Eds.), *Proceedings of ICACTCE'23—The International Conference on Advances in Communication Technology and Computer Engineering* (Vol.

735, pp. 425–436). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-37164-6_32

Zillow: Real Estate, Apartments, Mortgages & Home Values. (n.d.). Retrieved December 3, 2023, from https://www.zillow.com/

Appendix

Libraries Imported

```
#Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Lasso, LinearRegression
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import RandomizedSearchCV
import lightgbm as lgb
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import VotingRegressor
```

Data Description

```
# Open and print data describtion file to understand data.
with open('/content/data_description.txt', 'r') as file:
    data_describtion = file.read()
print(data_describtion)

MSSubClass: Identifies the type of dwelling involved in the sale.

        20      1-STORY 1946 & NEWER ALL STYLES
        30      1-STORY 1945 & OLDER
        40      1-STORY W/FINISHED ATTIC ALL AGES
        45      1-1/2 STORY - UNFINISHED ALL AGES
        50      1-1/2 STORY FINISHED ALL AGES
        60      2-STORY 1946 & NEWER
        70      2-STORY 1945 & OLDER
        75      2-1/2 STORY ALL AGES
        80      SPLIT OR MULTI-LEVEL
        85      SPLIT FOYER
        90      DUPLEX - ALL STYLES AND AGES
       120      1-STORY PUD (Planned Unit Development) - 1946 & NEWER
       150      1-1/2 STORY PUD - ALL AGES
       160      2-STORY PUD - 1946 & NEWER
       180      PUD - MULTILEVEL - INCL SPLIT LEV/FOYER
       190      2 FAMILY CONVERSION - ALL STYLES AND AGES

MSZoning: Identifies the general zoning classification of the sale.

       A        Agriculture
       C        Commercial
       FV       Floating Village Residential
       I        Industrial
       RH       Residential High Density
       RL       Residential Low Density
       RP       Residential Low Density Park
       RM      Residential Medium Density

LotFrontage: Linear feet of street connected to property

LotArea: Lot size in square feet

Street: Type of road access to property

       Grvl    Gravel
       Pave    Paved

Alley: Type of alley access to property

       Grvl    Gravel
       Pave    Paved
       NA      No alley access

LotShape: General shape of property

       Reg     Regular
       IR1     Slightly irregular
       IR2     Moderately Irregular
       IR3     Irregular

LandContour: Flatness of the property

       Lvl     Near Flat/Level
       Bnk     Banked - Quick and significant rise from street grade to building
       HLS     Hillside - Significant slope from side to side
       Low     Depression

Utilities: Type of utilities available

       AllPub  All public Utilities (E,G,W,& S)
       NoSewr  Electricity, Gas, and Water (Septic Tank)
       NoSeWa  Electricity and Gas Only
       ELO     Electricity only

LotConfig: Lot configuration

       Inside  Inside lot
       Corner  Corner lot
       CulDSac Cul-de-sac
       FR2     Frontage on 2 sides of property
       FR3     Frontage on 3 sides of property

LandSlope: Slope of property

       Gtl     Gentle slope
       Mod     Moderate Slope
       Sev     Severe Slope

Neighborhood: Physical locations within Ames city limits

       Blmngtn Bloomington Heights
       Blueste Bluestem
       BrDale  Briardale
```

BrkSide	Brookside
ClearCr	Clear Creek
CollgCr	College Creek
Crawfor	Crawford
Edwards	Edwards
Gilbert	Gilbert
IDOTRR	Iowa DOT and Rail Road
MeadowV	Meadow Village
Mitchel	Mitchell
Names	North Ames
NoRidge	Northridge
NPkVill	Northpark Villa
NridgHt	Northridge Heights
NWAmes	Northwest Ames
OldTown	Old Town
SWISU	South & West of Iowa State University
Sawyer	Sawyer
SawyerW	Sawyer West
Somerst	Somerset
StoneBr	Stone Brook
Timber	Timberland
Veenker	Veenker

Condition1: Proximity to various conditions

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

Condition2: Proximity to various conditions (if more than one is present)

Artery	Adjacent to arterial street
Feedr	Adjacent to feeder street
Norm	Normal
RRNn	Within 200' of North-South Railroad
RRAn	Adjacent to North-South Railroad
PosN	Near positive off-site feature--park, greenbelt, etc.
PosA	Adjacent to postive off-site feature
RRNe	Within 200' of East-West Railroad
RRAe	Adjacent to East-West Railroad

BldgType: Type of dwelling

1Fam	Single-family Detached
2FmCon	Two-family Conversion; originally built as one-family dwelling
Duplx	Duplex
TwnhsE	Townhouse End Unit
TwnhsI	Townhouse Inside Unit

HouseStyle: Style of dwelling

1Story	One story
1.5Fin	One and one-half story: 2nd level finished

1.5Unf	One and one-half story: 2nd level unfinished
2Story	Two story
2.5Fin	Two and one-half story: 2nd level finished
2.5Unf	Two and one-half story: 2nd level unfinished
SFoyer	Split Foyer
SLvl	Split Level

OverallQual: Rates the overall material and finish of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average
3	Fair
2	Poor
1	Very Poor

OverallCond: Rates the overall condition of the house

10	Very Excellent
9	Excellent
8	Very Good
7	Good
6	Above Average
5	Average
4	Below Average
3	Fair
2	Poor
1	Very Poor

YearBuilt: Original construction date

YearRemodAdd: Remodel date (same as construction date if no remodeling or additions)

RoofStyle: Type of roof

Flat	Flat
Gable	Gable
Gambrel	Gabrel (Barn)
Hip	Hip
Mansard	Mansard
Shed	Shed

RoofMatl: Roof material

ClyTile	Clay or Tile
CompShg	Standard (Composite) Shingle
Membran	Membrane
Metal	Metal
Roll	Roll
Tar&Grv	Gravel & Tar
WdShake	Wood Shakes
WdShngl	Wood Shingles

Exterior1st: Exterior covering on house

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles
BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

Exterior2nd: Exterior covering on house (if more than one material)

AsbShng	Asbestos Shingles
AsphShn	Asphalt Shingles
BrkComm	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
CemntBd	Cement Board
HdBoard	Hard Board
ImStucc	Imitation Stucco
MetalSd	Metal Siding
Other	Other
Plywood	Plywood
PreCast	PreCast
Stone	Stone
Stucco	Stucco
VinylSd	Vinyl Siding
Wd Sdng	Wood Siding
WdShing	Wood Shingles

MasVnrType: Masonry veneer type

BrkCmn	Brick Common
BrkFace	Brick Face
CBlock	Cinder Block
None	None
Stone	Stone

MasVnrArea: Masonry veneer area in square feet

ExterQual: Evaluates the quality of the material on the exterior

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

ExterCond: Evaluates the present condition of the material on the exterior

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

Foundation: Type of foundation

BrkTil	Brick & Tile
CBlock	Cinder Block
PConc	Poured Contrete
Slab	Slab
Stone	Stone
Wood	Wood

BsmtQual: Evaluates the height of the basement

Ex	Excellent (100+ inches)
Gd	Good (90-99 inches)
TA	Typical (80-89 inches)
Fa	Fair (70-79 inches)
Po	Poor (<70 inches
NA	No Basement

BsmtCond: Evaluates the general condition of the basement

Ex	Excellent
Gd	Good
TA	Typical - slight dampness allowed
Fa	Fair - dampness or some cracking or settling
Po	Poor - Severe cracking, settling, or wetness
NA	No Basement

BsmtExposure: Refers to walkout or garden level walls

Gd	Good Exposure
Av	Average Exposure (split levels or foyers typically score average or above)
Mn	Mimimum Exposure
No	No Exposure
NA	No Basement

BsmtFinType1: Rating of basement finished area

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters

Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Heating: Type of heating

Floor	Floor Furnace
GasA	Gas forced warm air furnace
GasW	Gas hot water or steam heat
Grav	Gravity furnace
OthW	Hot water or steam heat other than gas
Wall	Wall furnace

HeatingQC: Heating quality and condition

Ex	Excellent
Gd	Good
TA	Average/Typical
Fa	Fair
Po	Poor

CentralAir: Central air conditioning

N	No
Y	Yes

Electrical: Electrical system

SBrkr	Standard Circuit Breakers & Romex
FuseA	Fuse Box over 60 AMP and all Romex wiring (Average)
FuseF	60 AMP Fuse Box and mostly Romex wiring (Fair)
FuseP	60 AMP Fuse Box and mostly knob & tube wiring (poor)
Mix	Mixed

1stFlrSF: First Floor square feet

2ndFlrSF: Second floor square feet

LowQualFinSF: Low quality finished square feet (all floors)

GrLivArea: Above grade (ground) living area square feet

BsmtFullBath: Basement full bathrooms

BsmtHalfBath: Basement half bathrooms

FullBath: Full bathrooms above grade

HalfBath: Half baths above grade

```
Bedroom: Bedrooms above grade (does NOT include basement bedrooms)

Kitchen: Kitchens above grade

KitchenQual: Kitchen quality

       Ex      Excellent
       Gd      Good
       TA      Typical/Average
       Fa      Fair
       Po      Poor

TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)

Functional: Home functionality (Assume typical unless deductions are warranted)

       Typ     Typical Functionality
       Min1    Minor Deductions 1
       Min2    Minor Deductions 2
       Mod     Moderate Deductions
       Maj1    Major Deductions 1
       Maj2    Major Deductions 2
       Sev     Severely Damaged
       Sal     Salvage only

Fireplaces: Number of fireplaces

FireplaceQu: Fireplace quality

       Ex      Excellent - Exceptional Masonry Fireplace
       Gd      Good - Masonry Fireplace in main level
       TA      Average - Prefabricated Fireplace in main living area or Masonry Fireplace in ba
sement
       Fa      Fair - Prefabricated Fireplace in basement
       Po      Poor - Ben Franklin Stove
       NA      No Fireplace

GarageType: Garage location

       2Types  More than one type of garage
       Attchd  Attached to home
       Basment Basement Garage
       BuiltIn Built-In (Garage part of house - typically has room above garage)
       CarPort Car Port
       Detchd  Detached from home
       NA      No Garage

GarageYrBlt: Year garage was built

GarageFinish: Interior finish of the garage

       Fin     Finished
       RFn     Rough Finished
       Unf     Unfinished
       NA      No Garage

GarageCars: Size of garage in car capacity
```

```
GarageArea: Size of garage in square feet

GarageQual: Garage quality

       Ex      Excellent
       Gd      Good
       TA      Typical/Average
       Fa      Fair
       Po      Poor
       NA      No Garage

GarageCond: Garage condition

       Ex      Excellent
       Gd      Good
       TA      Typical/Average
       Fa      Fair
       Po      Poor
       NA      No Garage

PavedDrive: Paved driveway

       Y       Paved
       P       Partial Pavement
       N       Dirt/Gravel

WoodDeckSF: Wood deck area in square feet

OpenPorchSF: Open porch area in square feet

EnclosedPorch: Enclosed porch area in square feet

3SsnPorch: Three season porch area in square feet

ScreenPorch: Screen porch area in square feet

PoolArea: Pool area in square feet

PoolQC: Pool quality

       Ex      Excellent
       Gd      Good
       TA      Average/Typical
       Fa      Fair
       NA      No Pool

Fence: Fence quality

       GdPrv   Good Privacy
       MnPrv   Minimum Privacy
       GdWo    Good Wood
       MnWw    Minimum Wood/Wire
       NA      No Fence

MiscFeature: Miscellaneous feature not covered in other categories
```

```
       Elev    Elevator
       Gar2    2nd Garage (if not described in garage section)
       Othr    Other
       Shed    Shed (over 100 SF)
       TenC    Tennis Court
       NA      None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

       WD      Warranty Deed - Conventional
       CWD     Warranty Deed - Cash
       VWD     Warranty Deed - VA Loan
       New     Home just constructed and sold
       COD     Court Officer Deed/Estate
       Con     Contract 15% Down payment regular terms
       ConLw   Contract Low Down payment and low interest
       ConLI   Contract Low Interest
       ConLD   Contract Low Down
       Oth     Other

SaleCondition: Condition of sale

       Normal  Normal Sale
       Abnorml Abnormal Sale -  trade, foreclosure, short sale
       AdjLand Adjoining Land Purchase
       Alloca  Allocation - two linked properties with separate deeds, typically condo with a g
arage unit
       Family  Sale between family members
       Partial Home was not completed when last assessed (associated with New Homes)
```

###Data Acquisisiton

```
#upload the training data.
house_df = pd.read_csv('/content/train.csv')
```

###Data Exploration

```
#Check head of data
house_df.head()

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

  LandContour Utilities  ... PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  ...        0    NaN   NaN         NaN       0      2
```

```
4         Lvl    AllPub  ...        0    NaN   NaN         NaN       0     12

  YrSold  SaleType  SaleCondition  SalePrice
0   2008        WD         Normal     208500
1   2007        WD         Normal     181500
2   2008        WD         Normal     223500
3   2006        WD        Abnorml     140000
4   2008        WD         Normal     250000

[5 rows x 81 columns]
```

```
house_df.columns
```

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

```
# Understand the shape of training data.
house_df.shape
```

```
(1460, 81)
```

```
# check for duplicates
house_df.duplicated().sum()
```

```
0
```

```
# Drop Id column
house_df.drop(['Id'], axis = 1, inplace = True)
house_df.head()
```

```
   MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0          60       RL         65.0     8450   Pave   NaN      Reg
1          20       RL         80.0     9600   Pave   NaN      Reg
2          60       RL         68.0    11250   Pave   NaN      IR1
3          70       RL         60.0     9550   Pave   NaN      IR1
4          60       RL         84.0    14260   Pave   NaN      IR1

  LandContour Utilities LotConfig  ... PoolArea PoolQC Fence MiscFeature  \
0         Lvl    AllPub    Inside  ...        0    NaN   NaN         NaN
1         Lvl    AllPub       FR2  ...        0    NaN   NaN         NaN
2         Lvl    AllPub    Inside  ...        0    NaN   NaN         NaN
3         Lvl    AllPub    Corner  ...        0    NaN   NaN         NaN
4         Lvl    AllPub       FR2  ...        0    NaN   NaN         NaN

  MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
```

```
0       0      2    2008        WD         Normal      208500
1       0      5    2007        WD         Normal      181500
2       0      9    2008        WD         Normal      223500
3       0      2    2006        WD        Abnorml      140000
4       0     12    2008        WD         Normal      250000

[5 rows x 80 columns]
```

```
# house training data info.
house_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 80 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MSSubClass     1460 non-null   int64
 1   MSZoning       1460 non-null   object
 2   LotFrontage    1201 non-null   float64
 3   LotArea        1460 non-null   int64
 4   Street         1460 non-null   object
 5   Alley          91 non-null     object
 6   LotShape       1460 non-null   object
 7   LandContour    1460 non-null   object
 8   Utilities      1460 non-null   object
 9   LotConfig      1460 non-null   object
 10  LandSlope      1460 non-null   object
 11  Neighborhood   1460 non-null   object
 12  Condition1     1460 non-null   object
 13  Condition2     1460 non-null   object
 14  BldgType       1460 non-null   object
 15  HouseStyle     1460 non-null   object
 16  OverallQual    1460 non-null   int64
 17  OverallCond    1460 non-null   int64
 18  YearBuilt      1460 non-null   int64
 19  YearRemodAdd   1460 non-null   int64
 20  RoofStyle      1460 non-null   object
 21  RoofMatl       1460 non-null   object
 22  Exterior1st    1460 non-null   object
 23  Exterior2nd    1460 non-null   object
 24  MasVnrType     1452 non-null   object
 25  MasVnrArea     1452 non-null   float64
 26  ExterQual      1460 non-null   object
 27  ExterCond      1460 non-null   object
 28  Foundation     1460 non-null   object
 29  BsmtQual       1423 non-null   object
 30  BsmtCond       1423 non-null   object
 31  BsmtExposure   1422 non-null   object
 32  BsmtFinType1   1423 non-null   object
 33  BsmtFinSF1     1460 non-null   int64
 34  BsmtFinType2   1422 non-null   object
 35  BsmtFinSF2     1460 non-null   int64
 36  BsmtUnfSF      1460 non-null   int64
 37  TotalBsmtSF    1460 non-null   int64
 38  Heating        1460 non-null   object
 39  HeatingQC      1460 non-null   object
 40  CentralAir     1460 non-null   object
 41  Electrical     1459 non-null   object
 42  1stFlrSF       1460 non-null   int64
 43  2ndFlrSF       1460 non-null   int64
 44  LowQualFinSF   1460 non-null   int64
 45  GrLivArea      1460 non-null   int64
 46  BsmtFullBath   1460 non-null   int64
 47  BsmtHalfBath   1460 non-null   int64
 48  FullBath       1460 non-null   int64
 49  HalfBath       1460 non-null   int64
 50  BedroomAbvGr   1460 non-null   int64
 51  KitchenAbvGr   1460 non-null   int64
 52  KitchenQual    1460 non-null   object
 53  TotRmsAbvGrd   1460 non-null   int64
 54  Functional     1460 non-null   object
 55  Fireplaces     1460 non-null   int64
 56  FireplaceQu    770 non-null    object
 57  GarageType     1379 non-null   object
 58  GarageYrBlt    1379 non-null   float64
 59  GarageFinish   1379 non-null   object
 60  GarageCars     1460 non-null   int64
 61  GarageArea     1460 non-null   int64
 62  GarageQual     1379 non-null   object
 63  GarageCond     1379 non-null   object
 64  PavedDrive     1460 non-null   object
 65  WoodDeckSF     1460 non-null   int64
 66  OpenPorchSF    1460 non-null   int64
 67  EnclosedPorch  1460 non-null   int64
 68  3SsnPorch      1460 non-null   int64
 69  ScreenPorch    1460 non-null   int64
 70  PoolArea       1460 non-null   int64
 71  PoolQC         7 non-null      object
 72  Fence          281 non-null    object
 73  MiscFeature    54 non-null     object
 74  MiscVal        1460 non-null   int64
 75  MoSold         1460 non-null   int64
 76  YrSold         1460 non-null   int64
 77  SaleType       1460 non-null   object
 78  SaleCondition  1460 non-null   object
 79  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(34), object(43)
memory usage: 912.6+ KB
```

```
# Describe the training data.
house_df.describe().T
```

```
                count          mean          std      min       25%  \
MSSubClass     1460.0     56.897260    42.300571     20.0     20.00
LotFrontage    1201.0     70.049958    24.284752     21.0     59.00
LotArea        1460.0  10516.828082  9981.264932   1300.0   7553.50
OverallQual    1460.0      6.099315     1.382997      1.0      5.00
OverallCond    1460.0      5.575342     1.112799      1.0      5.00
YearBuilt      1460.0   1971.267808    30.202904   1872.0   1954.00
YearRemodAdd   1460.0   1984.865753    20.645407   1950.0   1967.00
MasVnrArea     1452.0    103.685262   181.066207      0.0      0.00
BsmtFinSF1     1460.0    443.639726   456.098091      0.0      0.00
BsmtFinSF2     1460.0     46.549315   161.319273      0.0      0.00
BsmtUnfSF      1460.0    567.240411   441.866955      0.0    223.00
TotalBsmtSF    1460.0   1057.429452   438.705324      0.0    795.75
1stFlrSF       1460.0   1162.626712   386.587738    334.0    882.00
2ndFlrSF       1460.0    346.992466   436.528436      0.0      0.00
LowQualFinSF   1460.0      5.844521    48.623081      0.0      0.00
GrLivArea      1460.0   1515.463699   525.480383    334.0   1129.50
BsmtFullBath   1460.0      0.425342     0.518911      0.0      0.00
BsmtHalfBath   1460.0      0.057534     0.238753      0.0      0.00
FullBath       1460.0      1.565068     0.550916      0.0      1.00
```

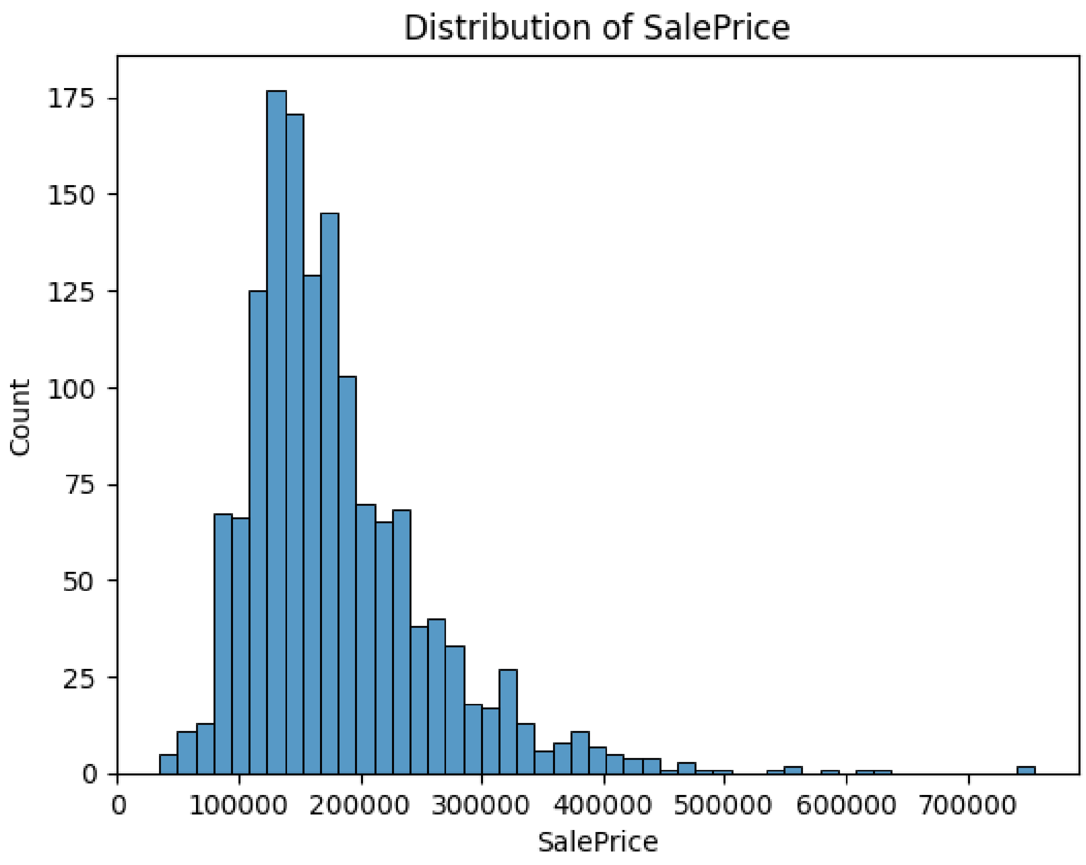
```
HalfBath       1460.0       0.382877      0.502885     0.0       0.00
BedroomAbvGr   1460.0       2.866438      0.815778     0.0       2.00
KitchenAbvGr   1460.0       1.046575      0.220338     0.0       1.00
TotRmsAbvGrd   1460.0       6.517808      1.625393     2.0       5.00
Fireplaces     1460.0       0.613014      0.644666     0.0       0.00
GarageYrBlt    1379.0    1978.506164     24.689725  1900.0    1961.00
GarageCars     1460.0       1.767123      0.747315     0.0       1.00
GarageArea     1460.0     472.980137    213.804841     0.0     334.50
WoodDeckSF     1460.0      94.244521    125.338794     0.0       0.00
OpenPorchSF    1460.0      46.660274     66.256028     0.0       0.00
EnclosedPorch  1460.0      21.954110     61.119149     0.0       0.00
3SsnPorch      1460.0       3.409589     29.317331     0.0       0.00
ScreenPorch    1460.0      15.060959     55.757415     0.0       0.00
PoolArea       1460.0       2.758904     40.177307     0.0       0.00
MiscVal        1460.0      43.489041    496.123024     0.0       0.00
MoSold         1460.0       6.321918      2.703626     1.0       5.00
YrSold         1460.0    2007.815753      1.328095  2006.0    2007.00
SalePrice      1460.0  180921.195890  79442.502883 34900.0  129975.00

                    50%        75%       max
MSSubClass         50.0      70.00     190.0
LotFrontage        69.0      80.00     313.0
LotArea          9478.5   11601.50  215245.0
OverallQual         6.0       7.00      10.0
OverallCond         5.0       6.00       9.0
YearBuilt        1973.0    2000.00    2010.0
YearRemodAdd     1994.0    2004.00    2010.0
MasVnrArea          0.0     166.00    1600.0
BsmtFinSF1        383.5     712.25    5644.0
BsmtFinSF2          0.0       0.00    1474.0
BsmtUnfSF         477.5     808.00    2336.0
TotalBsmtSF       991.5    1298.25    6110.0
1stFlrSF         1087.0    1391.25    4692.0
2ndFlrSF            0.0     728.00    2065.0
LowQualFinSF        0.0       0.00     572.0
GrLivArea        1464.0    1776.75    5642.0
BsmtFullBath        0.0       1.00       3.0
BsmtHalfBath        0.0       0.00       2.0
FullBath            2.0       2.00       3.0
HalfBath            0.0       1.00       2.0
BedroomAbvGr        3.0       3.00       8.0
KitchenAbvGr        1.0       1.00       3.0
TotRmsAbvGrd        6.0       7.00      14.0
Fireplaces          1.0       1.00       3.0
GarageYrBlt      1980.0    2002.00    2010.0
GarageCars          2.0       2.00       4.0
GarageArea        480.0     576.00    1418.0
WoodDeckSF          0.0     168.00     857.0
OpenPorchSF        25.0      68.00     547.0
EnclosedPorch       0.0       0.00     552.0
3SsnPorch           0.0       0.00     508.0
ScreenPorch         0.0       0.00     480.0
PoolArea            0.0       0.00     738.0
MiscVal             0.0       0.00   15500.0
MoSold              6.0       8.00      12.0
YrSold           2008.0    2009.00    2010.0
SalePrice      163000.0  214000.00  755000.0
```

```
# describe target column 'SalePrice'.
house_df['SalePrice'].describe()
```

```
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
```

```
# Distribution of target column 'SalePrice'.
sns.histplot(house_df['SalePrice'])
plt.title("Distribution of SalePrice")
```

```
Text(0.5, 1.0, 'Distribution of SalePrice')
```



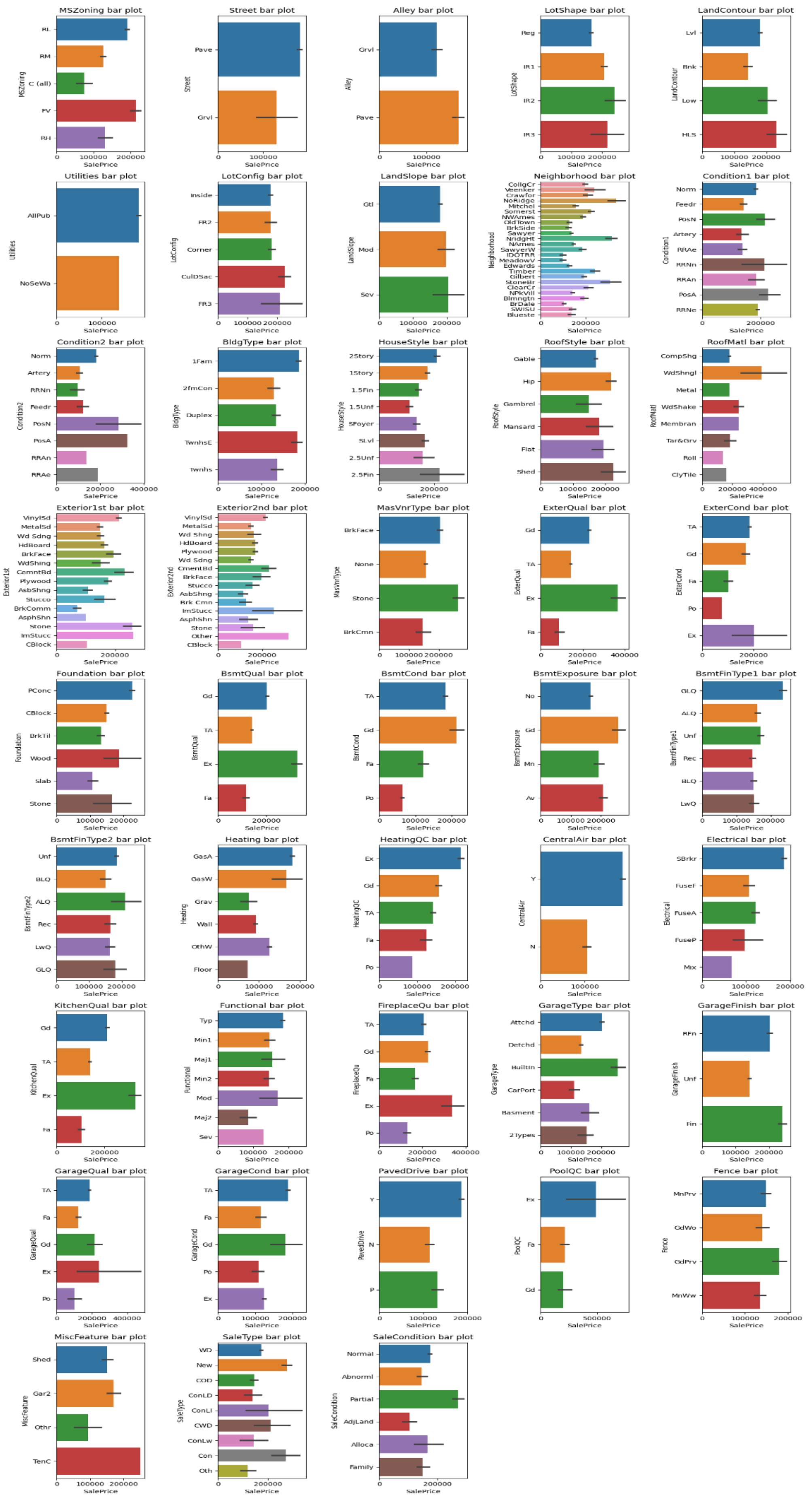
```
# Check variance of SalePrice
house_df['SalePrice'].var()
```

```
6311111264.297448
```

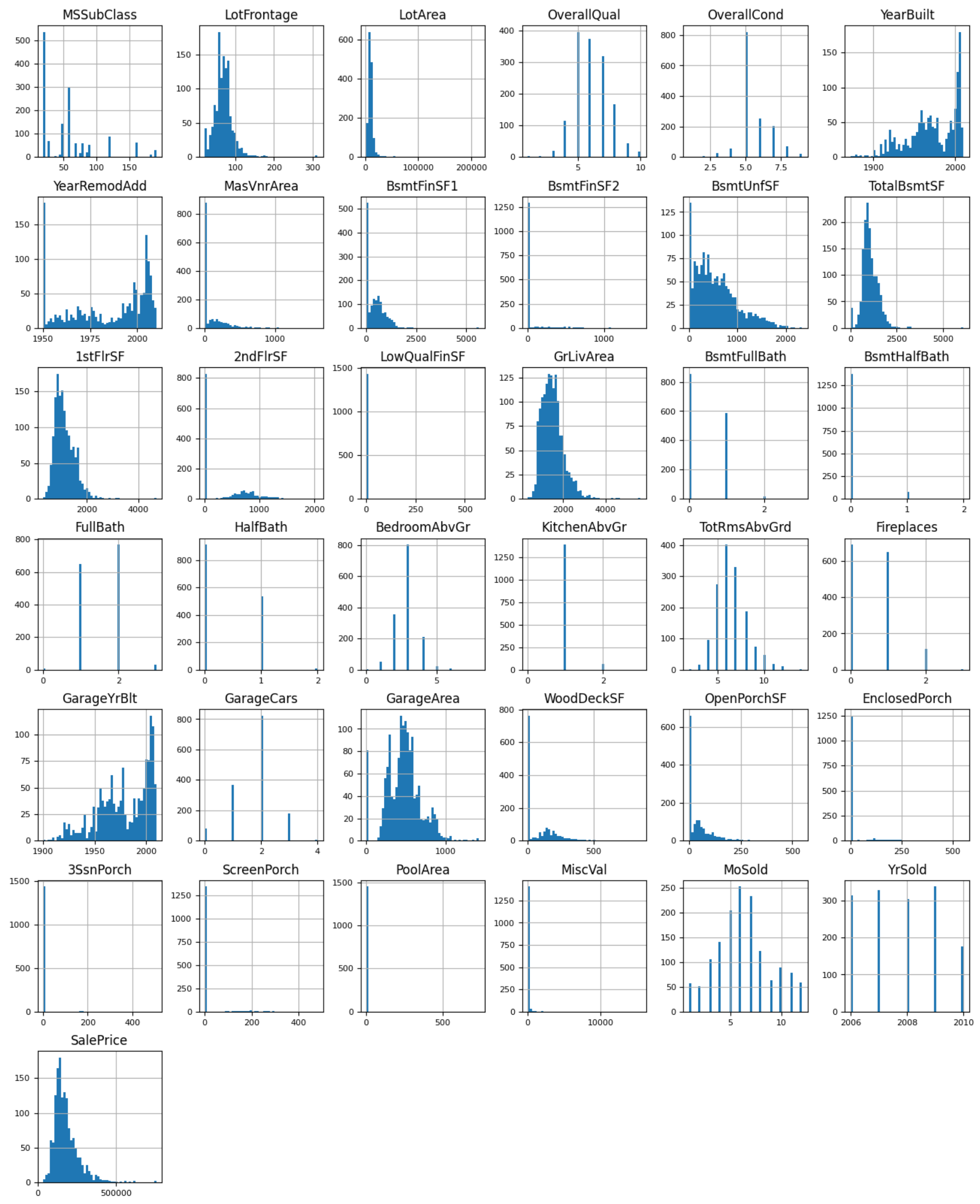
```
# Separate numerical and categorical features.
numerical_house_df = house_df.select_dtypes(include=['int64', 'float64'])
categorical_house_df = house_df.select_dtypes(exclude=['int64','float64'])
```

```
numerical_house_df.columns
```

```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')

categorical_house_df.head(3)

  MSZoning Street Alley LotShape LandContour Utilities LotConfig LandSlope  \
0       RL   Pave   NaN      Reg         Lvl    AllPub    Inside       Gtl
1       RL   Pave   NaN      Reg         Lvl    AllPub       FR2       Gtl
2       RL   Pave   NaN      IR1         Lvl    AllPub    Inside       Gtl

  Neighborhood Condition1  ... GarageType GarageFinish GarageQual GarageCond  \
0      CollgCr       Norm  ...     Attchd          RFn         TA         TA
1      Veenker      Feedr  ...     Attchd          RFn         TA         TA
2      CollgCr       Norm  ...     Attchd          RFn         TA         TA

  PavedDrive PoolQC Fence MiscFeature SaleType SaleCondition
0          Y    NaN   NaN         NaN       WD        Normal
1          Y    NaN   NaN         NaN       WD        Normal
2          Y    NaN   NaN         NaN       WD        Normal

[3 rows x 43 columns]

# Bar plot for exploring categorical features
cat_features = len(categorical_house_df.columns)
# Calculate the figure size
fig, axes = plt.subplots(nrows= 9, ncols= 5, figsize=(15, 40))
# Flatten the axes array to simplify indexing
axes = axes.flatten()
# Iterate over categorical features and create individual bar plot in subplots
for i, column in enumerate(categorical_house_df.columns):
    if i < cat_features:
        sns.barplot(x=house_df['SalePrice'],y=column ,data= categorical_house_df,orient = 'h',
ax=axes[i])
        axes[i].set_title(f'{column} bar plot')
# Minimize to only 43 plot not 45
for i in range(len(categorical_house_df.columns), len(axes)):
    fig.delaxes(axes[i])
#prevent overlapping title.
plt.tight_layout()
plt.show()
```

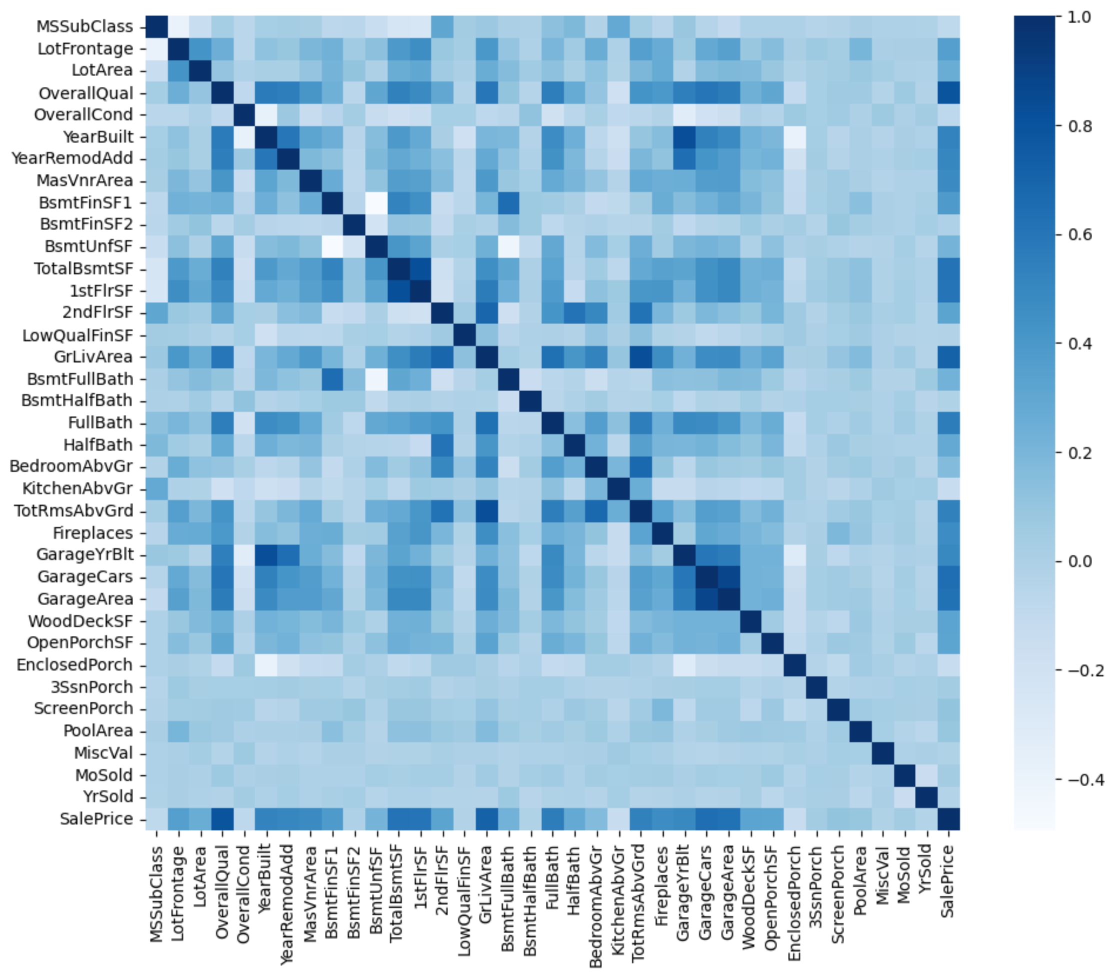

MSZoning bar plot
RL
RM
C (all)
FV
RH
Street bar plot
Pave
Grvl
Alley bar plot
Grvl
Pave
LotShape bar plot
Reg
IR1
IR2
IR3
LandContour bar plot
Lvl
Bnk
Low
HLS
Utilities bar plot
AllPub
NoSeWa
LotConfig bar plot
Inside
FR2
Corner
CulDSac
FR3
LandSlope bar plot
Gtl
Mod
Sev
Neighborhood bar plot
CollgCr
Veenker
Crawfor
NoRidge
Mitchel
Somerst
NWAmes
OldTown
BrkSide
Sawyer
NridgHt
NAmes
SawyerW
IDOTRR
MeadowV
Edwards
Timber
Gilbert
StoneBr
ClearCr
NPkVill
Blmngtn
BrDale
SWISU
Blueste
Condition1 bar plot
Norm
Feedr
PosN
Artery
RRAe
RRNn
RRAn
PosA
RRNe
Condition2 bar plot
Norm
Artery
RRNn
Feedr
PosN
PosA
RRAn
RRAe
BldgType bar plot
1Fam
2fmCon
Duplex
TwnhsE
Twnhs
HouseStyle bar plot
2Story
1Story
1.5Fin
1.5Unf
SFoyer
SLvl
2.5Unf
2.5Fin
RoofStyle bar plot
Gable
Hip
Gambrel
Mansard
Flat
Shed
RoofMatl bar plot
CompShg
WdShngl
Metal
WdShake
Membran
Tar&Grv
Roll
ClyTile
Exterior1st bar plot
VinylSd
MetalSd
Wd Sdng
HdBoard
BrkFace
WdShing
CemntBd
Plywood
AsbShng
Stucco
BrkComm
AsphShn
Stone
ImStucc
CBlock
Exterior2nd bar plot
VinylSd
MetalSd
Wd Shng
HdBoard
Plywood
Wd Sdng
CmentBd
BrkFace
Stucco
AsbShng
Brk Cmn
ImStucc
AsphShn
Stone
Other
CBlock
MasVnrType bar plot
BrkFace
None
Stone
BrkCmn
ExterQual bar plot
Gd
TA
Ex
Fa
ExterCond bar plot
TA
Gd
Fa
Po
Ex
Foundation bar plot
PConc
CBlock
BrkTil
Wood
Slab
Stone
BsmtQual bar plot
Gd
TA
Ex
Fa
BsmtCond bar plot
TA
Gd
Fa
Po
BsmtExposure bar plot
No
Gd
Mn
Av
BsmtFinType1 bar plot
GLQ
ALQ
Unf
Rec
BLQ
LwQ
BsmtFinType2 bar plot
Unf
BLQ
ALQ
Rec
LwQ
GLQ
Heating bar plot
GasA
GasW
Grav
Wall
OthW
Floor
HeatingQC bar plot
Ex
Gd
TA
Fa
Po
CentralAir bar plot
Y
N
Electrical bar plot
SBrkr
FuseF
FuseA
FuseP
Mix
KitchenQual bar plot
Gd
TA
Ex
Fa
Functional bar plot
Typ
Min1
Maj1
Min2
Mod
Maj2
Sev
FireplaceQu bar plot
TA
Gd
Fa
Ex
Po
GarageType bar plot
Attchd
Detchd
BuiltIn
CarPort
Basment
2Types
GarageFinish bar plot
RFn
Unf
Fin
GarageQual bar plot
TA
Fa
Gd
Ex
Po
GarageCond bar plot
TA
Fa
Gd
Po
Ex
PavedDrive bar plot
Y
N
P
PoolQC bar plot
Ex
Fa
Gd
Fence bar plot
MnPrv
GdWo
GdPrv
MnWw
MiscFeature bar plot
Shed
Gar2
Othr
TenC
SaleType bar plot
WD
New
COD
ConLD
ConLI
CWD
ConLw
Con
Oth
SaleCondition bar plot
Normal
Abnorml
Partial
AdjLand
Alloca
Family
SalePrice

```python
# Numerical features distribution.
numerical_house_df.hist(figsize=(16,20), bins=50, xlabelsize=8, ylabelsize=8)
plt.title('Distribution of Numerical features')
plt.show()
```



```
# Heatmap of numerical features.
corrmat = numerical_house_df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat,square=True,cmap='Blues')
```

```
<Axes: >
```

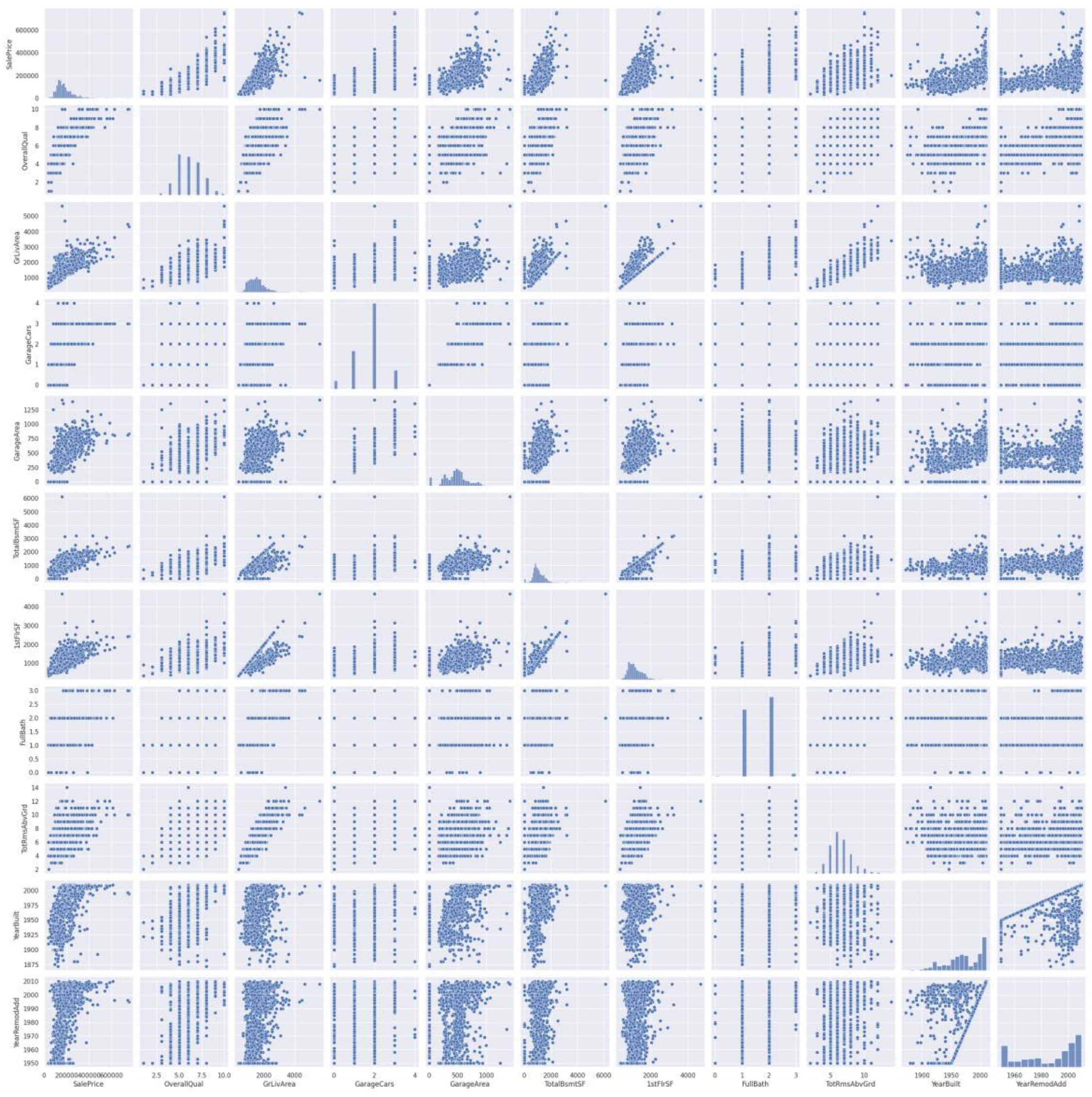


```
corr_matrix = numerical_house_df.corr()
corr_matrix['SalePrice'].sort_values(ascending=False)
```

```
SalePrice        1.000000
OverallQual      0.790982
GrLivArea        0.708624
GarageCars       0.640409
GarageArea       0.623431
TotalBsmtSF      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
BsmtFinSF1       0.386420
```

```
LotFrontage      0.351799
WoodDeckSF       0.324413
2ndFlrSF         0.319334
OpenPorchSF      0.315856
HalfBath         0.284108
LotArea          0.263843
BsmtFullBath     0.227122
BsmtUnfSF        0.214479
BedroomAbvGr     0.168213
ScreenPorch      0.111447
PoolArea         0.092404
MoSold           0.046432
3SsnPorch        0.044584
BsmtFinSF2      -0.011378
BsmtHalfBath    -0.016844
MiscVal         -0.021190
LowQualFinSF    -0.025606
YrSold          -0.028923
OverallCond     -0.077856
MSSubClass      -0.084284
EnclosedPorch   -0.128578
KitchenAbvGr    -0.135907
Name: SalePrice, dtype: float64
```

```
# Scatter plot for numerical features that have high correlation with target.
sns.set()
numerical_features = ['SalePrice','OverallQual','GrLivArea','GarageCars','GarageArea','TotalBs
mtSF','1stFlrSF','FullBath','TotRmsAbvGrd','YearBuilt','YearRemodAdd']
sns.pairplot(numerical_house_df[numerical_features],height= 2.5)
plt.show()
```


SalePrice
OverallQual
GrLivArea
GarageCars
GarageArea
TotalBsmtSF
1stFlrSF
FullBath
TotRmsAbvGrd
YearBuilt
YearRemodAdd

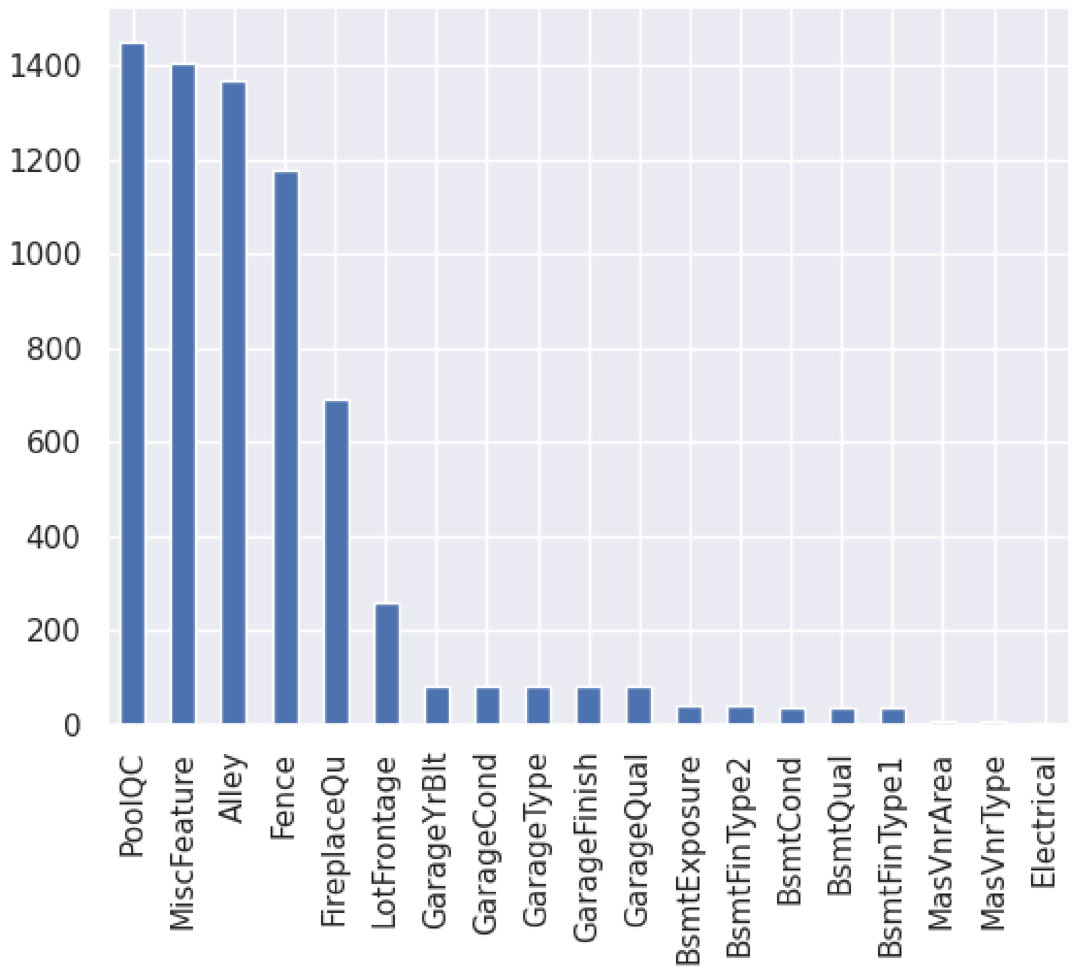
###Data Pre-Processing

#####Handling Missing Values "Imputation"

```
# Calculate missing data
missing_values = house_df.isnull().sum().sort_values(ascending=False)
missing_percent = (house_df.isnull().sum()/house_df.isnull().count()).sort_values(ascending=Fa
lse)
missing_data = pd.concat([missing_values, missing_percent], axis=1, keys=['missing_values', 'm
issing_percent'])
missing_data.head(30)
```

```
              missing_values  missing_percent
PoolQC                  1453         0.995205
MiscFeature             1406         0.963014
Alley                   1369         0.937671
Fence                   1179         0.807534
FireplaceQu              690         0.472603
LotFrontage              259         0.177397
GarageYrBlt               81         0.055479
GarageCond                81         0.055479
GarageType                81         0.055479
GarageFinish              81         0.055479
GarageQual                81         0.055479
BsmtExposure              38         0.026027
BsmtFinType2              38         0.026027
BsmtCond                  37         0.025342
BsmtQual                  37         0.025342
BsmtFinType1              37         0.025342
MasVnrArea                 8         0.005479
MasVnrType                 8         0.005479
Electrical                 1         0.000685
MSSubClass                 0         0.000000
Fireplaces                 0         0.000000
Functional                 0         0.000000
KitchenQual                0         0.000000
KitchenAbvGr               0         0.000000
BedroomAbvGr               0         0.000000
HalfBath                   0         0.000000
FullBath                   0         0.000000
BsmtHalfBath               0         0.000000
TotRmsAbvGrd               0         0.000000
GarageCars                 0         0.000000
```

```
# bar chart for missing values
missing_values = missing_values[missing_values > 0]
missing_values.plot.bar()
plt.show()
```



```
# data type of missing values in each feature.
house_df.loc[:, missing_values.index].dtypes
```

```
PoolQC           object
MiscFeature      object
Alley            object
Fence            object
FireplaceQu      object
LotFrontage     float64
GarageYrBlt     float64
GarageCond       object
GarageType       object
GarageFinish     object
GarageQual       object
BsmtExposure     object
BsmtFinType2     object
BsmtCond         object
BsmtQual         object
BsmtFinType1     object
MasVnrArea      float64
MasVnrType       object
Electrical       object
dtype: object
```

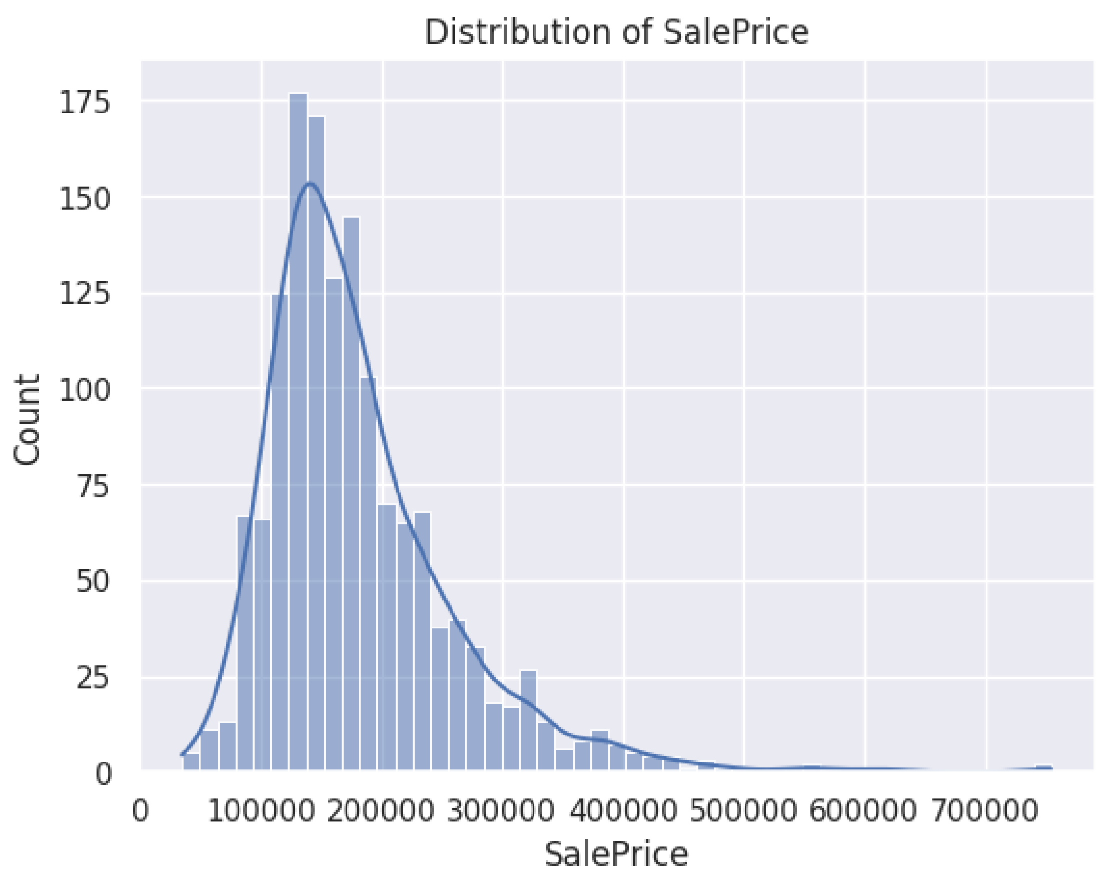
```
# Replace missing values in Fire place quality with None.
house_df['FireplaceQu'].fillna('None',inplace = True)# No fireplace
house_df['GarageCond'].fillna('None',inplace = True)# No Garage
house_df['GarageType'].fillna('None',inplace = True) # No Garage
house_df['GarageFinish'].fillna('None',inplace = True)# No Garage
house_df['GarageQual'].fillna('None',inplace = True)# No Garage
house_df['PoolQC'].fillna('None',inplace = True) # No Pool
house_df['MiscFeature'].fillna('None',inplace = True)# No fireplace
house_df['Alley'].fillna('None',inplace = True)# No Alley
house_df['Fence'].fillna('None',inplace = True)# No Fence
house_df['BsmtExposure'].fillna('None',inplace = True)# No Basement
house_df['BsmtFinType2'].fillna('None',inplace = True)# No Basement
house_df['BsmtCond'].fillna('None',inplace = True)# No Basement
house_df['BsmtQual'].fillna('None',inplace = True)# No Basement
house_df['BsmtFinType1'].fillna('None',inplace = True)# No Basement

# Replace missing values with mode.
house_df['MasVnrType']=house_df['MasVnrType'].fillna(house_df['MasVnrType'].mode()[0])
house_df['Electrical']=house_df['Electrical'].fillna(house_df['Electrical'].mode()[0])

# Replace missing values from numerical columns.
house_df['LotFrontage']=house_df['LotFrontage'].fillna(house_df['LotFrontage'].median())
house_df['GarageYrBlt']= house_df['GarageYrBlt'].fillna(house_df['GarageYrBlt'].median())
house_df['MasVnrArea'] = house_df['MasVnrArea'].fillna(house_df['MasVnrArea'].median())

house_df.isnull().sum().sort_values(ascending = False).head(20)
```

```
MSSubClass      0
MSZoning        0
GarageYrBlt     0
GarageType      0
FireplaceQu     0
Fireplaces      0
Functional      0
TotRmsAbvGrd    0
KitchenQual     0
KitchenAbvGr    0
BedroomAbvGr    0
HalfBath        0
FullBath        0
BsmtHalfBath    0
BsmtFullBath    0
GrLivArea       0
LowQualFinSF    0
2ndFlrSF        0
1stFlrSF        0
GarageFinish    0
dtype: int64
```

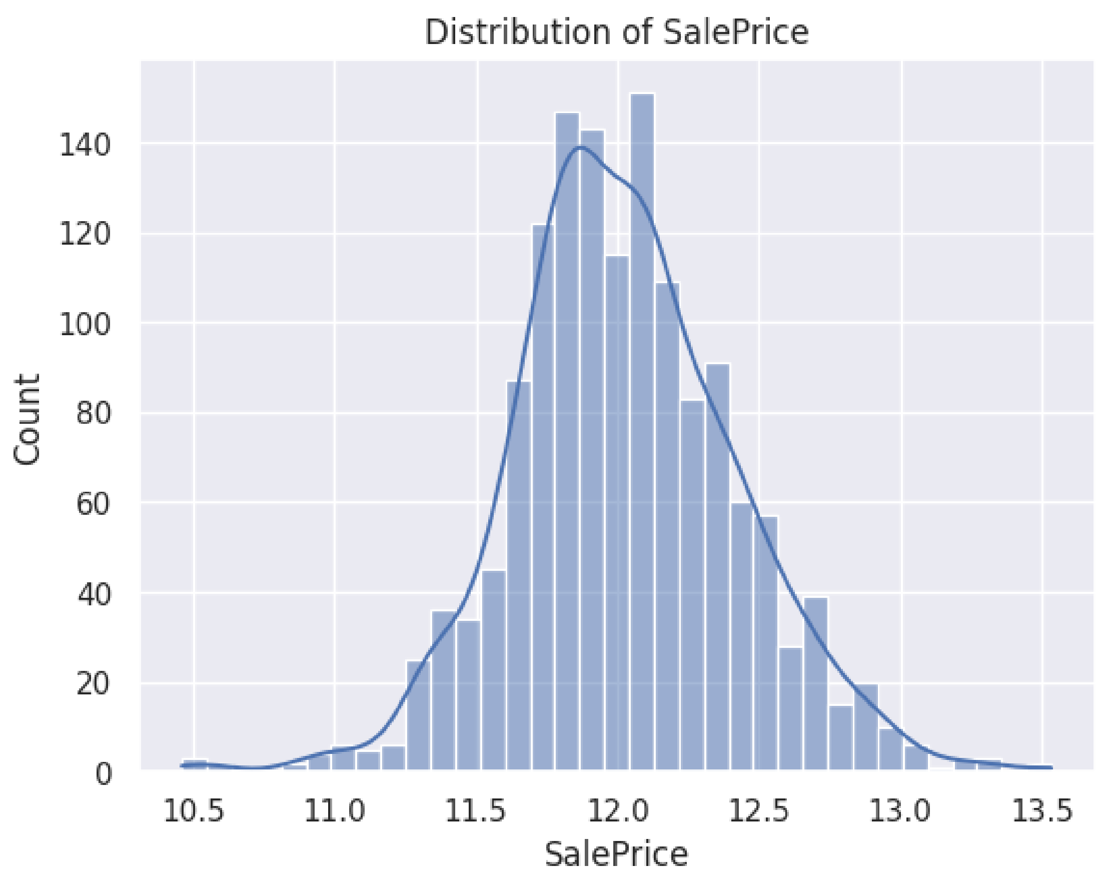
#####Log transofmation

```
# Sale Price before log transformation
sns.histplot(house_df['SalePrice'], kde = True)
plt.title("Distribution of SalePrice")
```

```
Text(0.5, 1.0, 'Distribution of SalePrice')
```



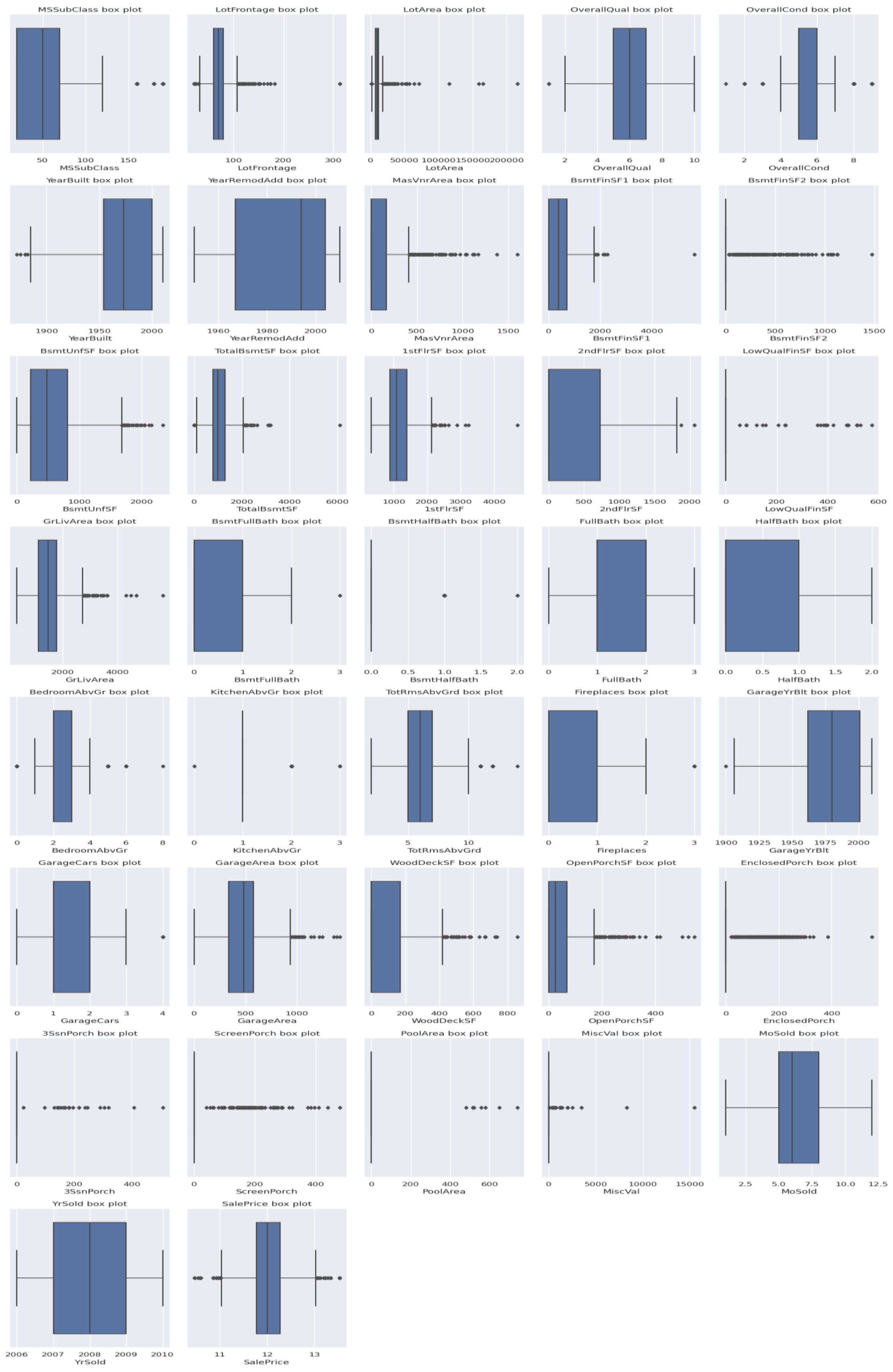
```
# Target Log transformation
house_df['SalePrice'] = np.log(house_df['SalePrice'])

# SalePrice variance after log transformation
house_df['SalePrice'].var()

0.15956179505733453

# SalePrice after transformation
sns.histplot(house_df['SalePrice'], kde = True)
plt.title("Distribution of SalePrice")

Text(0.5, 1.0, 'Distribution of SalePrice')
```



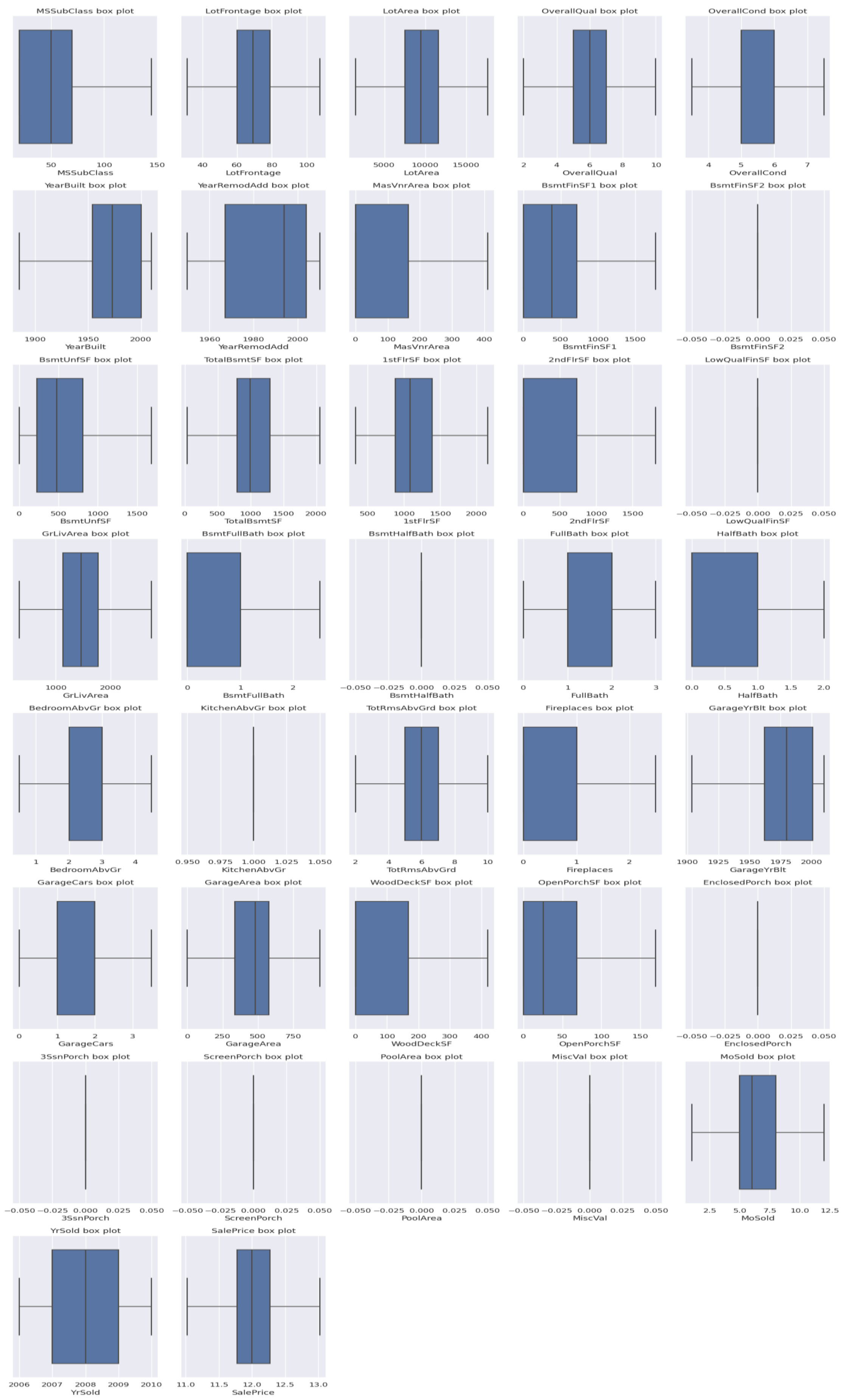
Handling Outliers

```
#numerica columns
numerical_house_df.columns

Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')

# boxplot for numerical features before deleting outliers
num_features = len(numerical_house_df.columns)
# Calculate the figure size
fig, axes = plt.subplots(nrows= 8, ncols= 5, figsize=(15, 40))
# Flatten the axes array to simplify indexing
axes = axes.flatten()
# Iterate over numerical features and create individual box plots in subplots
for i, column in enumerate(numerical_house_df.columns):
    if i < num_features:
        sns.boxplot(x=house_df[column], ax=axes[i])
        axes[i].set_title(f'{column} box plot')
# Minimize to only 37 plot
for i in range(len(numerical_house_df.columns), len(axes)):
    fig.delaxes(axes[i])
```

```
#prevent overlapping title.
plt.tight_layout()

plt.show()
```


MSSubClass box plot
MSSubClass
LotFrontage box plot
LotFrontage
LotArea box plot
LotArea
OverallQual box plot
OverallQual
OverallCond box plot
OverallCond
YearBuilt box plot
YearBuilt
YearRemodAdd box plot
YearRemodAdd
MasVnrArea box plot
MasVnrArea
BsmtFinSF1 box plot
BsmtFinSF1
BsmtFinSF2 box plot
BsmtFinSF2
BsmtUnfSF box plot
BsmtUnfSF
TotalBsmtSF box plot
TotalBsmtSF
1stFlrSF box plot
1stFlrSF
2ndFlrSF box plot
2ndFlrSF
LowQualFinSF box plot
LowQualFinSF
GrLivArea box plot
GrLivArea
BsmtFullBath box plot
BsmtFullBath
BsmtHalfBath box plot
BsmtHalfBath
FullBath box plot
FullBath
HalfBath box plot
HalfBath
BedroomAbvGr box plot
BedroomAbvGr
KitchenAbvGr box plot
KitchenAbvGr
TotRmsAbvGrd box plot
TotRmsAbvGrd
Fireplaces box plot
Fireplaces
GarageYrBlt box plot
GarageYrBlt
GarageCars box plot
GarageCars
GarageArea box plot
GarageArea
WoodDeckSF box plot
WoodDeckSF
OpenPorchSF box plot
OpenPorchSF
EnclosedPorch box plot
EnclosedPorch
3SsnPorch box plot
3SsnPorch
ScreenPorch box plot
ScreenPorch
PoolArea box plot
PoolArea
MiscVal box plot
MiscVal
MoSold box plot
MoSold
YrSold box plot
YrSold
SalePrice box plot
SalePrice

```
# Create function to calculate upper and lower whisker for each numerical feature.
def capping(df, cols, factor):
    for col in cols:
        Q1 = df[col].quantile(0.25) # Calculate first quartile
        Q3 = df[col].quantile(0.75) # calculate second quartile
        IQR = Q3 - Q1 # Interquartile
        upper = Q3 + (factor*IQR) # upper_whisker
        lower = Q1 - (factor*IQR) # lower_whisker
        df[col] = np.where(df[col]>upper, upper,np.where(df[col]<lower, lower, df[col])) # cha
nge outliers

# Handle outliers with capping method.
capping(house_df,house_df.select_dtypes(include=['int64', 'float64']), 1.5)

# boxplot after deleting outliers
num_features = len(numerical_house_df.columns)
# figure size
fig, axes = plt.subplots(nrows= 8, ncols= 5, figsize=(15, 40))
# Flatten axes to simplify indexing
axes = axes.flatten()
# create individual box plots for each feature in subplots
for i, column in enumerate(numerical_house_df.columns):
    if i < num_features:
        sns.boxplot(x=house_df[column], ax=axes[i])
        axes[i].set_title(f'{column} box plot')
#minimize to only 37 plot
for i in range(len(numerical_house_df.columns), len(axes)):
    fig.delaxes(axes[i])
#prevent overlapping title.
plt.tight_layout()
plt.show()
```


MSSubClass box plot
LotFrontage box plot
LotArea box plot
OverallQual box plot
OverallCond box plot
YearBuilt box plot
YearRemodAdd box plot
MasVnrArea box plot
BsmtFinSF1 box plot
BsmtFinSF2 box plot
BsmtUnfSF box plot
TotalBsmtSF box plot
1stFlrSF box plot
2ndFlrSF box plot
LowQualFinSF box plot
GrLivArea box plot
BsmtFullBath box plot
BsmtHalfBath box plot
FullBath box plot
HalfBath box plot
BedroomAbvGr box plot
KitchenAbvGr box plot
TotRmsAbvGrd box plot
Fireplaces box plot
GarageYrBlt box plot
GarageCars box plot
GarageArea box plot
WoodDeckSF box plot
OpenPorchSF box plot
EnclosedPorch box plot
3SsnPorch box plot
ScreenPorch box plot
PoolArea box plot
MiscVal box plot
MoSold box plot
YrSold box plot
SalePrice box plot

####**Handling Categorical Features**

```
# Categorical columns
categorical_house_df.columns

Index(['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilities',
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'FireplaceQu', 'GarageType', 'GarageFinish', 'GarageQual',
       'GarageCond', 'PavedDrive', 'PoolQC', 'Fence', 'MiscFeature',
       'SaleType', 'SaleCondition'],
      dtype='object')

# Numerical columns
numerical_house_df.columns

Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')

# Categorical encoding
label = LabelEncoder()
for col in house_df.columns:
  if house_df[col].dtype == 'object':
    house_df[col] = label.fit_transform(house_df[col])

house_df.shape

(1460, 80)

house_df.head()

   MSSubClass  MSZoning  LotFrontage  LotArea  Street  Alley  LotShape  \
0        60.0         3         65.0   8450.0       1      1         3
1        20.0         3         80.0   9600.0       1      1         3
2        60.0         3         68.0  11250.0       1      1         0
3        70.0         3         60.0   9550.0       1      1         0
4        60.0         3         84.0  14260.0       1      1         0

   LandContour  Utilities  LotConfig  ...  PoolArea  PoolQC  Fence  \
0            3          0          4  ...       0.0       3      4
1            3          0          2  ...       0.0       3      4
2            3          0          4  ...       0.0       3      4
3            3          0          0  ...       0.0       3      4
4            3          0          2  ...       0.0       3      4

   MiscFeature  MiscVal  MoSold  YrSold  SaleType  SaleCondition  SalePrice
0            1      0.0     2.0  2008.0         8              4  12.247694
1            1      0.0     5.0  2007.0         8              4  12.109011
2            1      0.0     9.0  2008.0         8              4  12.317167
3            1      0.0     2.0  2006.0         8              0  11.849398
4            1      0.0    12.0  2008.0         8              4  12.429216
```

```
[5 rows x 80 columns]
```

###Splitting to train and test

```
X = house_df.drop('SalePrice', axis = 1)
y = house_df['SalePrice']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

X_train
```

```
      MSSubClass  MSZoning  LotFrontage  LotArea  Street  Alley  LotShape  \
254         20.0         3         70.0   8400.0       1      1         3
1066        60.0         3         59.0   7837.0       1      1         0
638         30.0         3         67.0   8777.0       1      1         3
799         50.0         3         60.0   7200.0       1      1         3
380         50.0         3         50.0   5000.0       1      2         3
...          ...       ...          ...      ...     ...    ...       ...
1095        20.0         3         78.0   9317.0       1      1         0
1130        50.0         3         65.0   7804.0       1      1         3
1294        20.0         3         60.0   8172.0       1      1         3
860         50.0         3         55.0   7642.0       1      1         3
1126       120.0         3         53.0   3684.0       1      1         3

      LandContour  Utilities  LotConfig  ...  ScreenPorch  PoolArea  PoolQC  \
254             3          0          4  ...          0.0       0.0       3
1066            3          0          4  ...          0.0       0.0       3
638             3          0          4  ...          0.0       0.0       3
799             3          0          0  ...          0.0       0.0       3
380             3          0          4  ...          0.0       0.0       3
...           ...        ...        ...  ...          ...       ...     ...
1095            3          0          4  ...          0.0       0.0       3
1130            3          0          4  ...          0.0       0.0       3
1294            3          0          4  ...          0.0       0.0       3
860             3          0          0  ...          0.0       0.0       3
1126            3          0          4  ...          0.0       0.0       3

      Fence  MiscFeature  MiscVal  MoSold  YrSold  SaleType  SaleCondition
254       4            1      0.0     6.0  2010.0         8              4
1066      4            1      0.0     5.0  2009.0         8              4
638       2            1      0.0     5.0  2008.0         8              4
799       2            1      0.0     6.0  2007.0         8              4
380       4            1      0.0     5.0  2010.0         8              4
...     ...          ...      ...     ...     ...       ...            ...
1095      4            1      0.0     3.0  2007.0         8              4
1130      2            1      0.0    12.0  2009.0         8              4
1294      4            1      0.0     4.0  2006.0         8              4
860       0            1      0.0     6.0  2007.0         8              4
1126      4            1      0.0     6.0  2009.0         8              4

[1168 rows x 79 columns]
```

#####Scaling the Data

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_train_scaled

      MSSubClass  MSZoning  LotFrontage   LotArea    Street     Alley  \
0      -0.933485 -0.054796     0.044442 -0.369609  0.058621  0.047674
1       0.135447 -0.054796    -0.592466 -0.527046  0.058621  0.047674
2      -0.666252 -0.054796    -0.129260 -0.264185  0.058621  0.047674
3      -0.131786 -0.054796    -0.534566 -0.705176  0.058621  0.047674
4      -0.131786 -0.054796    -1.113573 -1.320382  0.058621  4.025068
...          ...       ...          ...       ...       ...       ...
1163   -0.933485 -0.054796     0.507648 -0.113180  0.058621  0.047674
1164   -0.131786 -0.054796    -0.245062 -0.536274  0.058621  0.047674
1165   -0.933485 -0.054796    -0.534566 -0.433367  0.058621  0.047674
1166   -0.131786 -0.054796    -0.824069 -0.581576  0.058621  0.047674
1167    1.738845 -0.054796    -0.939871 -1.688388  0.058621  0.047674

      LotShape  LandContour  Utilities  LotConfig  ...  ScreenPorch  PoolArea  \
0     0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0
1    -1.356644     0.299798  -0.029273   0.630128  ...          0.0       0.0
2     0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0
3     0.765535     0.299798  -0.029273  -1.792884  ...          0.0       0.0
4     0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0
...        ...          ...        ...        ...  ...          ...       ...
1163 -1.356644     0.299798  -0.029273   0.630128  ...          0.0       0.0
1164  0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0
1165  0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0
1166  0.765535     0.299798  -0.029273  -1.792884  ...          0.0       0.0
1167  0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0

        PoolQC     Fence  MiscFeature  MiscVal    MoSold    YrSold  SaleType  \
0     0.066503  0.467615    -0.190299      0.0 -0.133417  1.650065  0.316662
1     0.066503  0.467615    -0.190299      0.0 -0.508010  0.893677  0.316662
2     0.066503 -1.343907    -0.190299      0.0 -0.508010  0.137290  0.316662
3     0.066503 -1.343907    -0.190299      0.0 -0.133417 -0.619098  0.316662
4     0.066503  0.467615    -0.190299      0.0 -0.508010  1.650065  0.316662
...        ...       ...          ...      ...       ...       ...       ...
1163  0.066503  0.467615    -0.190299      0.0 -1.257196 -0.619098  0.316662
1164  0.066503 -1.343907    -0.190299      0.0  2.114141  0.893677  0.316662
1165  0.066503  0.467615    -0.190299      0.0 -0.882603 -1.375486  0.316662
1166  0.066503 -3.155429    -0.190299      0.0 -0.133417 -0.619098  0.316662
1167  0.066503  0.467615    -0.190299      0.0 -0.133417  0.893677  0.316662

      SaleCondition
0          0.201772
1          0.201772
2          0.201772
3          0.201772
4          0.201772
...             ...
1163       0.201772
1164       0.201772
1165       0.201772
1166       0.201772
1167       0.201772

[1168 rows x 79 columns]

X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_test_scaled
```

```
     MSSubClass  MSZoning  LotFrontage   LotArea    Street     Alley  \
0     -0.933485 -0.054796     0.044442 -0.365694  0.058621  0.047674   
1      0.135447 -0.054796     1.665663  0.708680  0.058621  0.047674   
2     -0.666252  1.586277    -0.766169 -0.213011  0.058621 -3.929719   
3     -0.131786  1.586277    -1.113573 -1.320382  0.058621  0.047674   
4     -0.933485 -0.054796     1.144556  0.888208  0.058621  0.047674   
..          ...       ...          ...       ...       ...       ...   
287   -0.666252  1.586277    -1.113573 -1.061716  0.058621  0.047674   
288   -0.933485 -0.054796     2.215720  1.799832  0.058621  0.047674   
289    0.135447 -0.054796    -0.360863 -0.425817  0.058621  0.047674   
290    0.402680 -0.054796    -0.534566 -0.178336  0.058621  0.047674   
291   -0.933485 -0.054796     0.044442 -0.447908  0.058621  0.047674   

     LotShape  LandContour  Utilities  LotConfig  ...  ScreenPorch  PoolArea  \
0    0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0   
1   -1.356644     0.299798  -0.029273  -1.792884  ...          0.0       0.0   
2    0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0   
3    0.765535     0.299798  -0.029273  -1.792884  ...          0.0       0.0   
4   -1.356644    -2.618233  -0.029273   0.630128  ...          0.0       0.0   
..        ...          ...        ...        ...  ...          ...       ...   
287  0.765535    -4.077249  -0.029273   0.630128  ...          0.0       0.0   
288 -1.356644    -1.159218  -0.029273   0.630128  ...          0.0       0.0   
289  0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0   
290  0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0   
291  0.765535     0.299798  -0.029273   0.630128  ...          0.0       0.0   

       PoolQC     Fence  MiscFeature  MiscVal    MoSold    YrSold  SaleType  \
0    0.066503 -1.343907    -0.190299      0.0 -1.631789 -1.375486  0.316662   
1    0.066503  0.467615    -0.190299      0.0 -0.882603  1.650065  0.316662   
2    0.066503  0.467615    -0.190299      0.0 -1.257196  1.650065  0.316662   
3    0.066503  0.467615    -0.190299      0.0  1.364955 -1.375486  0.316662   
4    0.066503  0.467615    -0.190299      0.0  0.990362  0.893677  0.316662   
..        ...       ...          ...      ...       ...       ...       ...   
287  0.066503 -1.343907    -0.190299      0.0 -1.257196 -0.619098  0.316662   
288  0.066503  0.467615    -0.190299      0.0 -0.133417  0.893677  0.316662   
289  0.066503  0.467615    -0.190299      0.0  1.364955  0.137290  0.316662   
290  0.066503 -1.343907    -0.190299      0.0  1.364955  0.893677  0.316662   
291  0.066503  0.467615    -0.190299      0.0  0.241176  0.893677  0.316662   

     SaleCondition  
0         0.201772  
1         0.201772  
2         0.201772  
3         0.201772  
4         0.201772  
..             ...  
287      -1.653892  
288       0.201772  
289       0.201772  
290       0.201772  
291       0.201772  

[292 rows x 79 columns]
```

#####Modeling and Hyperparameter Tuning

######LASSO

```
# create model instance
lasso_model = Lasso(alpha = 0.001, random_state = 10) #create model instance
```

```
# Use RandomizedSearchCV
lasso_parameters = {'alpha': [0.001, 0.01, 0.1,0.000391,0.000491,1, 10, 100, 1000]
                    ,'random_state':[1,5,10,24,42]}
lasso_model_random = RandomizedSearchCV(lasso_model, lasso_parameters, n_iter=10, cv=5,scoring
= 'neg_mean_absolute_error',n_jobs = -1)
lasso_model_random.fit(X_train_scaled,y_train)

RandomizedSearchCV(cv=5, estimator=Lasso(alpha=0.001, random_state=10),
                   n_jobs=-1,
                   param_distributions={'alpha': [0.001, 0.01, 0.1, 0.000391,
                                                  0.000491, 1, 10, 100, 1000],
                                        'random_state': [1, 5, 10, 24, 42]},
                   scoring='neg_mean_absolute_error')

# Lasso validation
lasso_r2_cv = cross_val_score(lasso_model,X_train_scaled,y_train, scoring="r2",cv = 5)
lasso_mae_cv = -cross_val_score(lasso_model,X_train_scaled,y_train, scoring="neg_mean_absolute
_error",cv = 5)
lasso_rmse_cv = -cross_val_score(lasso_model,X_train_scaled,y_train, scoring="neg_root_mean_sq
uared_error",cv = 5)
# Print score and scoring means
print("R2 scores:", lasso_r2_cv)
print("Mean R2:", np.mean(lasso_r2_cv))
print("MAE scores:", lasso_mae_cv)
print("Mean MAE:", np.mean(lasso_mae_cv))
print("RSME scores:", lasso_rmse_cv)
print("Mean RSME:", np.mean(lasso_rmse_cv))

R2 scores: [0.91514172 0.86114004 0.88331953 0.90540684 0.90613434]
Mean R2: 0.8942284939023724
MAE scores: [0.0812163  0.0917392  0.09074863 0.08588829 0.07771177]
Mean MAE: 0.0854608399317327
RSME scores: [0.10970036 0.13920932 0.13993263 0.11722237 0.1083503 ]
Mean RSME: 0.12288299421975364

# figure out best hyperparameters
print(lasso_model_random.best_estimator_)

Lasso(alpha=0.000391, random_state=10)

# Training LinearRegression model on training set.
lasso_model.fit(X_train_scaled,y_train)

Lasso(alpha=0.001, random_state=10)

# Model Prediction
lasso_y_pred = lasso_model.predict(X_test_scaled)

# Calculate R2, MAE, RMSE
lasso_r2_score = r2_score(y_test, lasso_y_pred)
lasso_mae_score = mean_absolute_error(y_test, lasso_y_pred)
lasso_rmse_score = np.sqrt(mean_squared_error(y_test, lasso_y_pred))
# Print scores
print("R2 score:", lasso_r2_score)
print("MAE scores:", lasso_mae_score)
print("RMSE score:", lasso_rmse_score)

R2 score: 0.9044818230200362
MAE scores: 0.08970264159372936
RMSE score: 0.12698196744804813
```

######KNN Regressor

```
# KNN hyperparameter tuning.
knn_model = KNeighborsRegressor()
knn_parameters = {
    'n_neighbors': np.arange(5,20)
}
knn_model_random = RandomizedSearchCV(knn_model, knn_parameters, n_iter=10, cv=5, random_state
=42,scoring = 'neg_mean_absolute_error',n_jobs = -1)
knn_model_random.fit(X_train_scaled,y_train)

RandomizedSearchCV(cv=5, estimator=KNeighborsRegressor(), n_jobs=-1,
                   param_distributions={'n_neighbors': array([ 5,  6,  7,  8,  9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19])},
                   random_state=42, scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(knn_model_random.best_estimator_)

KNeighborsRegressor(n_neighbors=7)

# KNN Cross-Validation
knn_model = KNeighborsRegressor(7)
knn_r2_cv = cross_val_score(knn_model,X_train_scaled,y_train, scoring="r2",cv = 5)
knn_mae_cv = -cross_val_score(knn_model,X_train_scaled,y_train, scoring="neg_mean_absolute_err
or",cv = 5)
knn_rmse_cv = -cross_val_score(knn_model,X_train_scaled,y_train, scoring="neg_root_mean_square
d_error",cv = 5)
# Print score and scoring means
print("R2 scores:", knn_r2_cv)
print("Mean R2:", np.mean(knn_r2_cv))
print("MAE scores:", knn_mae_cv)
print("Mean MAE:", np.mean(knn_mae_cv))
print("RSME scores:", knn_rmse_cv)
print("Mean RSME:", np.mean(knn_rmse_cv))

R2 scores: [0.82942321 0.80857873 0.78376055 0.80545169 0.81547797]
Mean R2: 0.8085384293636986
MAE scores: [0.11217005 0.11960553 0.13581658 0.1229724  0.11066458]
Mean MAE: 0.12024582799950663
RSME scores: [0.1555324  0.16344623 0.19049666 0.16811032 0.15191507]
Mean RSME: 0.1659001348252332

# KNN fitting to data.
knn_model.fit(X_train_scaled, y_train)

KNeighborsRegressor(n_neighbors=7)

# KNN prediction
knn_y_pred = knn_model.predict(X_test_scaled)

# Calculate and print scores
knn_r2_score = r2_score(y_test, knn_y_pred)
knn_mae_score = mean_absolute_error(y_test, knn_y_pred)
knn_rmse_score = np.sqrt(mean_squared_error(y_test, knn_y_pred))
# Print scores
print("R2 score:", knn_r2_score)
print("MAE scores:", knn_mae_score)
print("RMSE score:", knn_rmse_score)
```

```
R2 score: 0.8188392441996817
MAE scores: 0.1286051517763571
RMSE score: 0.17487635843662072
```

######SVR

```
# SVR cross_validation
svr_model = SVR( )# create model instance
svr_r2_cv = cross_val_score(svr_model,X_train_scaled,y_train, scoring="r2",cv = 5)
svr_mae_cv = -cross_val_score(svr_model,X_train_scaled,y_train, scoring="neg_mean_absolute_err
or",cv = 5)
svr_rmse_cv = -cross_val_score(svr_model,X_train_scaled,y_train, scoring="neg_root_mean_square
d_error",cv = 5)
# Print score and scoring means
print("R2 scores:", svr_r2_cv)
print("Mean R2:", np.mean(svr_r2_cv))
print("MAE scores:", svr_mae_cv)
print("Mean MAE:", np.mean(svr_mae_cv))
print("RSME scores:", svr_rmse_cv)
print("Mean RSME:", np.mean(svr_rmse_cv))
```

```
R2 scores: [0.85125548 0.85978534 0.84081795 0.82259625 0.84384907]
Mean R2: 0.8436608180793096
MAE scores: [0.09320769 0.10188274 0.11181347 0.10880024 0.09787004]
Mean MAE: 0.10271483449661556
RSME scores: [0.14523838 0.13988673 0.16344323 0.16053216 0.1397491 ]
Mean RSME: 0.1497699199056675
```

```
# fit SVR model
svr_model.fit(X_train_scaled, y_train)
```

```
SVR()
```

```
# Make predictions on the test data.
svr_y_pred = svr_model.predict(X_test_scaled)

# Calculate and print evaluations scores
svr_r2_score = r2_score(y_test, svr_y_pred)
svr_mae_score = mean_absolute_error(y_test, svr_y_pred)
svr_rmse_score = np.sqrt(mean_squared_error(y_test, svr_y_pred))
# Print the evaluation metrics.
print("R2 score:", svr_r2_score)
print("MAE scores:", svr_mae_score)
print("RMSE score:", svr_rmse_score)
```

```
R2 score: 0.821242610879984
MAE scores: 0.1110363341623632
RMSE score: 0.17371248802626518
```

######Decision Tree Regressor

```
# DTR hyperparameter tuning using RandomizedSearch cv
dtr_model = DecisionTreeRegressor() #create model instance

dtr_parameters = {
    'max_depth': np.arange(1, 21),
    'min_samples_split': np.arange(2, 11),
    'min_samples_leaf': np.arange(1, 11),
    'max_features':[20,30,50]
}
dtr_model_random = RandomizedSearchCV(dtr_model, dtr_parameters, n_iter=10, cv=5, random_state
=42,scoring = 'neg_mean_absolute_error',n_jobs = -1)
dtr_model_random.fit(X_train_scaled,y_train)

RandomizedSearchCV(cv=5, estimator=DecisionTreeRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9
, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20]),
                                        'max_features': [20, 30, 50],
                                        'min_samples_leaf': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10]),
                                        'min_samples_split': array([ 2,  3,  4,  5,  6,  7,  8
,  9, 10])},
                   random_state=42, scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(dtr_model_random.best_estimator_)

DecisionTreeRegressor(max_depth=20, max_features=50, min_samples_leaf=9,
                      min_samples_split=10)

# DTR validation
dtr_model = DecisionTreeRegressor(max_depth=11, min_samples_leaf=7, min_samples_split=4)
dtr_r2_cv = cross_val_score(dtr_model,X_train_scaled,y_train, scoring="r2",cv = 5)
dtr_mae_cv = -cross_val_score(dtr_model,X_train_scaled,y_train, scoring="neg_mean_absolute_err
or",cv = 5)
dtr_rmse_cv = -cross_val_score(dtr_model,X_train_scaled,y_train, scoring="neg_root_mean_square
d_error",cv = 5)
# Print score and scoring means
print("R2 scores:", dtr_r2_cv)
print("Mean R2:", np.mean(dtr_r2_cv))
print("MAE scores:", dtr_mae_cv)
print("Mean MAE:", np.mean(dtr_mae_cv))
print("RSME scores:", dtr_rmse_cv)
print("Mean RSME:", np.mean(dtr_rmse_cv))

R2 scores: [0.79393244 0.73275859 0.81215354 0.785475   0.76992605]
Mean R2: 0.778849125071273
MAE scores: [0.1257418  0.13655795 0.13387828 0.13613404 0.12706427]
Mean MAE: 0.13187526633460778
RSME scores: [0.17219127 0.19365437 0.1775503  0.17646496 0.16934539]
Mean RSME: 0.1778412568497989

# fitting dtr model.
dtr_model.fit(X_train_scaled,y_train)

DecisionTreeRegressor(max_depth=11, min_samples_leaf=7, min_samples_split=4)

# dtr prediction
dtr_y_pred = dtr_model.predict(X_test_scaled)

# calculate and print dtr scores
dtr_r2_score = r2_score(y_test, dtr_y_pred)
dtr_mae_score = mean_absolute_error(y_test, dtr_y_pred)
dtr_rmse_score = np.sqrt(mean_squared_error(y_test,dtr_y_pred))
# Print scores
print("R2 score:", dtr_r2_score)
print("MAE scores:", dtr_mae_score)
print("RMSE score:", dtr_rmse_score)

R2 score: 0.7911529039464633
MAE scores: 0.14403474799395483
RMSE score: 0.18776440251810755
```

######Random Forest Regressor

```
# Rf hyperparameter tuning.
rf_model = RandomForestRegressor()
rf_parameters = {
    'min_samples_leaf':[10,15,20],
    'min_samples_split':[10,15,20],
    'max_depth': [4,6,8,10],
}
rf_model_random = RandomizedSearchCV(rf_model, rf_parameters, n_iter=10, cv=5, random_state=42
,scoring = 'neg_mean_absolute_error',n_jobs = -1)
rf_model_random.fit(X_train_scaled,y_train)

RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
                   param_distributions={'max_depth': [4, 6, 8, 10],
                                        'min_samples_leaf': [10, 15, 20],
                                        'min_samples_split': [10, 15, 20]},
                   random_state=42, scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(rf_model_random.best_estimator_)

RandomForestRegressor(max_depth=8, min_samples_leaf=15, min_samples_split=10)

# cross validation to random forest
rf_model = RandomForestRegressor()
random_forest_r2_cv = cross_val_score(rf_model,X_train_scaled,y_train, scoring="r2",cv = 5)
random_forest_mae_cv = -cross_val_score(rf_model,X_train_scaled,y_train, scoring="neg_mean_abs
olute_error",cv = 5)
random_forest_rmse_cv = -cross_val_score(rf_model,X_train_scaled,y_train, scoring="neg_root_me
an_squared_error",cv = 5)
# Print score and scoring means
print("R2 scores:", random_forest_r2_cv)
print("Mean R2:", np.mean(random_forest_r2_cv))
print("MAE scores:", random_forest_mae_cv)
print("Mean MAE:", np.mean(random_forest_mae_cv))
print("RSME scores:", random_forest_rmse_cv)
print("Mean RSME:", np.mean(random_forest_rmse_cv))

R2 scores: [0.89803974 0.82701911 0.87129568 0.86519458 0.88786007]
Mean R2: 0.8698818338757828
MAE scores: [0.08374276 0.10587958 0.10187934 0.10281325 0.08199719]
Mean MAE: 0.09526242445464875
RSME scores: [0.12076353 0.15440703 0.14723576 0.13856128 0.11617691]
Mean RSME: 0.1354289046572923

#Random Forest fit
rf_model.fit(X_train_scaled,y_train)

RandomForestRegressor()

# random forest prediction
rf_y_pred = rf_model.predict(X_test_scaled)

# calculate and print scores
rf_r2_score = r2_score(y_test, rf_y_pred)
rf_mae_score = mean_absolute_error(y_test, rf_y_pred)
rf_rmse_score = np.sqrt(mean_squared_error(y_test, rf_y_pred))
# Print scores
print("R2 score:", rf_r2_score)
print("MAE scores:", rf_mae_score)
print("RMSE score:", rf_rmse_score)

R2 score: 0.8899633870185536
MAE scores: 0.09728684639777124
RMSE score: 0.13629114689424793
```

######Gradient Booster Regressor

```
# Gradient Boosting Hyperparameter tuning
gbr_model = GradientBoostingRegressor()
gbr_parameters = {
    'n_estimators':[2,3,4,5,6,7,20,30,50],
    'learning_rate':[0.01, 0.03, 0.05, 0.1,0.001],
    'max_depth':[3, 4, 5, 6],
    'alpha':[0.001,0.1,0.5,0.2]
}
gbr_model_random = RandomizedSearchCV(gbr_model,gbr_parameters, n_iter=10, cv=5,scoring = 'neg
_mean_absolute_error',n_jobs = -1,error_score='raise')
gbr_model_random.fit(X_train_scaled,y_train)

RandomizedSearchCV(cv=5, error_score='raise',
                   estimator=GradientBoostingRegressor(), n_jobs=-1,
                   param_distributions={'alpha': [0.001, 0.1, 0.5, 0.2],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.001],
                                        'max_depth': [3, 4, 5, 6],
                                        'n_estimators': [2, 3, 4, 5, 6, 7, 20,
                                                         30, 50]},
                   scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(gbr_model_random.best_estimator_)

GradientBoostingRegressor(alpha=0.5, learning_rate=0.05, max_depth=5,
                          n_estimators=30)

# Gradient Booster cross validation
gbr_model = GradientBoostingRegressor(alpha=0.5, n_estimators=30) # create model instance
gbr_r2_cv = cross_val_score(gbr_model,X_train_scaled,y_train, scoring="r2",cv = 5)
gbr_mae_cv = -cross_val_score(gbr_model,X_train_scaled,y_train, scoring="neg_mean_absolute_err
or",cv = 5)
gbr_rmse_cv = -cross_val_score(gbr_model,X_train_scaled,y_train, scoring="neg_root_mean_square
d_error",cv = 5)
# Print score and scoring means
print("R2 scores:", gbr_r2_cv)
print("Mean R2:", np.mean(gbr_r2_cv))
print("MAE scores:", gbr_mae_cv)
print("Mean MAE:", np.mean(gbr_mae_cv))
print("RSME scores:", gbr_rmse_cv)
print("Mean RSME:", np.mean(gbr_rmse_cv))

R2 scores: [0.8673904  0.83720537 0.83951086 0.8478362  0.87889375]
Mean R2: 0.8541673164693911
MAE scores: [0.09652216 0.11075329 0.11661752 0.10888996 0.08685095]
Mean MAE: 0.10392677621492344
RSME scores: [0.13704732 0.1519922  0.1641129  0.14898419 0.12283623]
Mean RSME: 0.1449945684809948

# Fit gbr model to data
gbr_model.fit(X_train_scaled, y_train)
```

```
GradientBoostingRegressor(alpha=0.5, n_estimators=30)

# gbr prediction
gbr_y_pred = gbr_model.predict(X_test_scaled)

# Calculate and print evaluation scores
gbr_r2_score = r2_score(y_test, gbr_y_pred)
gbr_mae_score = mean_absolute_error(y_test, gbr_y_pred)
gbr_rmse_score = np.sqrt(mean_squared_error(y_test, gbr_y_pred))
# Print the evaluation metrics.
print("R2 score:", gbr_r2_score)
print("MAE scores:", gbr_mae_score)
print("RMSE score:", gbr_rmse_score)

R2 score: 0.8776502352477635
MAE scores: 0.10467667222559117
RMSE score: 0.14371450669300415
```

######XGBoost

```
# XGBoost hyperparameter tuning
xgb_model = XGBRegressor()
xgb_parameters = {
    'n_estimators': [100, 200, 300, 500,2200,220],
    'learning_rate': [0.01, 0.03, 0.05, 0.1,0.001],
    'max_depth': [3, 4, 5, 6],
    'min_child_weight': [1, 2, 3, 4,1.7817,0.178,0.0178],
    'gamma': [0, 0.1, 0.2, 0.3,0.005, 0.0045,0.35,0.5],
    'subsample': [0.6, 0.7, 0.8, 0.9],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9],
    'alpha': [0, 0.1, 0.5, 1]
}
xgb_model_random = RandomizedSearchCV(xgb_model,xgb_parameters, n_iter=10, cv=5,scoring = 'neg
_mean_absolute_error',n_jobs = -1)
xgb_model_random.fit(X_train_scaled,y_train)

RandomizedSearchCV(cv=5,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          callbacks=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, device=None,
                                          early_stopping_rounds=None,
                                          enable_categorical=False,
                                          eval_metric=None, feature_types=None,
                                          gamma=None, grow_policy=None,
                                          importance_type=None,
                                          interaction_constraints=None,
                                          learning_rate=...
                   n_jobs=-1,
                   param_distributions={'alpha': [0, 0.1, 0.5, 1],
                                        'colsample_bytree': [0.6, 0.7, 0.8,
                                                             0.9],
                                        'gamma': [0, 0.1, 0.2, 0.3, 0.005,
                                                  0.0045, 0.35, 0.5],
                                        'learning_rate': [0.01, 0.03, 0.05, 0.1,
                                                          0.001],
                                        'max_depth': [3, 4, 5, 6],
                                        'min_child_weight': [1, 2, 3, 4, 1.7817,
                                                             0.178, 0.0178],
                                        'n_estimators': [100, 200, 300, 500,
                                             2200, 220],
                                  'subsample': [0.6, 0.7, 0.8, 0.9]},
                   scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(xgb_model_random.best_estimator_)

XGBRegressor(alpha=0.1, base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.0045, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.03, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=1.7817, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=200, n_jobs=None,
             num_parallel_tree=None, ...)

# XGBoost validation
xgb_model = XGBRegressor( gamma = 0,n_estimators=220,learning_rate=0.05,random_state=42,max_de
pth=5,colsample_bytree=0.7,min_child_weight=1.7817,alpha=0.5)# create model instance
xgb_r2_cv = cross_val_score(xgb_model,X_train_scaled,y_train, scoring="r2",cv = 5)
xgb_mae_cv = -cross_val_score(xgb_model,X_train_scaled,y_train, scoring="neg_mean_absolute_err
or",cv = 5)
xgb_rmse_cv = -cross_val_score(xgb_model,X_train_scaled,y_train, scoring="neg_root_mean_square
d_error",cv = 5)
# Print score and scoring means
print("R2 scores:", xgb_r2_cv)
print("Mean R2:", np.mean(xgb_r2_cv))
print("MAE scores:", xgb_mae_cv)
print("Mean MAE:", np.mean(xgb_mae_cv))
print("RSME scores:", xgb_rmse_cv)
print("Mean RSME:", np.mean(xgb_rmse_cv))

R2 scores: [0.90824463 0.86600988 0.89993456 0.8837685  0.90858287]
Mean R2: 0.8933080880133396
MAE scores: [0.07781259 0.09374167 0.09168381 0.09632367 0.07593342]
Mean MAE: 0.08709903241328876
RSME scores: [0.11407137 0.13674649 0.12958716 0.12993991 0.10692778]
Mean RSME: 0.12345453994917192

# XGBoost fit to data.
xgb_model.fit(X_train_scaled,y_train)

XGBRegressor(alpha=0.5, base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.7, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=5, max_leaves=None,
             min_child_weight=1.7817, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=220, n_jobs=None,
             num_parallel_tree=None, ...)

#XGBoost prediction
xgb_y_pred = xgb_model.predict(X_test_scaled)
```

```
# calculate and print scores.
xgb_r2_score = r2_score(y_test, xgb_y_pred)
xgb_mae_score = mean_absolute_error(y_test, xgb_y_pred)
xgb_rmse_score = np.sqrt(mean_squared_error(y_test, xgb_y_pred))
# Print scores
print("R2 score:", xgb_r2_score)
print("MAE scores:", xgb_mae_score)
print("RMSE score:", xgb_rmse_score)

R2 score: 0.9137570160601971
MAE scores: 0.08521061986653618
RMSE score: 0.12065933469866713
```

######LightGBM

```
# LGB Hyperparameter
lgb_model = lgb.LGBMRegressor()
lgb_parameters = {
    'n_estimators': [6,7,10,11,50,60],
    'learning_rate': [0.7,0.8,0.1,0.001,0.2,0.0003,0.3,0.5],
    'num_leaves': [5,6,7],
    'max_depth': [3, 4, 5, 6,50],
    'reg_alpha': [0, 0.1, 0.5, 1],


}
lgb_model_random = RandomizedSearchCV(lgb_model,lgb_parameters,n_iter=10, cv=5,scoring = 'neg_
mean_absolute_error',n_jobs = -1)
lgb_model_random.fit(X_train_scaled,y_train)

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000574
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2926
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 65
[LightGBM] [Info] Start training from score 12.031035

RandomizedSearchCV(cv=5, estimator=LGBMRegressor(), n_jobs=-1,
                   param_distributions={'learning_rate': [0.7, 0.8, 0.1, 0.001,
                                                          0.2, 0.0003, 0.3,
                                                          0.5],
                                        'max_depth': [3, 4, 5, 6, 50],
                                        'n_estimators': [6, 7, 10, 11, 50, 60],
                                        'num_leaves': [5, 6, 7],
                                        'reg_alpha': [0, 0.1, 0.5, 1]},
                   scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(lgb_model_random.best_estimator_)

LGBMRegressor(learning_rate=0.3, max_depth=6, n_estimators=50, num_leaves=7,
              reg_alpha=1)

# LightGBM validation
lgb_model = lgb.LGBMRegressor(learning_rate=0.2, max_depth=6, n_estimators=50, num_leaves=6,re
g_alpha=0)# create model instance
lgb_r2_cv = cross_val_score(lgb_model,X_train_scaled,y_train, scoring="r2",cv = 5)
lgb_mae_cv = -cross_val_score(lgb_model,X_train_scaled,y_train, scoring="neg_mean_absolute_err
or",cv = 5)
lgb_rmse_cv = -cross_val_score(lgb_model,X_train_scaled,y_train, scoring="neg_root_mean_square
d_error",cv = 5)
```

```
print("R2 scores:", lgb_r2_cv)
print("Mean R2:", np.mean(lgb_r2_cv))
print("MAE scores:", lgb_mae_cv)
print("Mean MAE:", np.mean(lgb_mae_cv))
print("RSME scores:", lgb_rmse_cv)
print("Mean RSME:", np.mean(lgb_rmse_cv))
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000357
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2828
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.030302
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000405
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.020671
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000375
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2822
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 64
[LightGBM] [Info] Start training from score 12.041320
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000423
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.034734
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000386
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2835
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.028148
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000417
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2828
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.030302
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000577
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.020671
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000382
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2822
```

```
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 64
[LightGBM] [Info] Start training from score 12.041320
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000416
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.034734
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000405
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2835
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.028148
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000383
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2828
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.030302
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000374
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.020671
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000399
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2822
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 64
[LightGBM] [Info] Start training from score 12.041320
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000379
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.034734
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000438
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2835
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.028148
R2 scores: [0.90362198 0.86798016 0.88655032 0.88880053 0.89486506]
Mean R2: 0.8883636114462181
MAE scores: [0.08291353 0.08959073 0.09415874 0.09406981 0.07951395]
Mean MAE: 0.08804935321431087
RSME scores: [0.11690953 0.13573736 0.13798171 0.12709603 0.11467012]
Mean RSME: 0.12647895119428948
```

```
#fit LGBmodel
lgb_model.fit(X_train_scaled,y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000501
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2926
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 65
[LightGBM] [Info] Start training from score 12.031035

LGBMRegressor(learning_rate=0.2, max_depth=6, n_estimators=50, num_leaves=6,
              reg_alpha=0)

# LGBmodel prediction
lgb_y_pred = lgb_model.predict(X_test_scaled)

# Calculate and print scores
lgb_r2_score = r2_score(y_test, lgb_y_pred)
lgb_mae_score = mean_absolute_error(y_test, lgb_y_pred)
lgb_rmse_score = np.sqrt(mean_squared_error(y_test, lgb_y_pred))
# Print scores
print("R2 score:", lgb_r2_score)
print("MAE scores:", lgb_mae_score)
print("RMSE score:", lgb_rmse_score)

R2 score: 0.9108859444420738
MAE scores: 0.08850765663252229
RMSE score: 0.1226512965590393
```

######Voting Regressor

```
# Voting Model
voting_model = VotingRegressor(estimators=[('Gradient Boost',gbr_model),('xgboost', xgb_model)
, ('lightgbm', lgb_model)])

# voting Hyperparameter
voting_parameters = {
'weights': [[0.2,0.6,0.2], [0.6,0.2,0.2],[0.40,0.40,0.40], [0.20,0.40,0.50]]
}
voting_model_random = RandomizedSearchCV(voting_model,voting_parameters,n_iter=10, cv=5,scorin
g = 'neg_mean_absolute_error',n_jobs = -1)
voting_model_random.fit(X_train_scaled,y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:305: UserWarning: T
he total space of parameters 4 is smaller than n_iter=10. Running 4 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000465
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2926
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 65
[LightGBM] [Info] Start training from score 12.031035

RandomizedSearchCV(cv=5,
                   estimator=VotingRegressor(estimators=[('Gradient                    Boost',
                                                          GradientBoostingRegressor(alpha=0.5,
                                                                                    n_estimato
rs=30)),
                                                         ('xgboost',
                                                          XGBRegressor(alpha=0.5,
                                                                       base_score=None,
                                                                       booster=None,
                                                                       callbacks=None,
                                                                       colsample_bylevel=None,
                                                                       colsample_bynode=None,
                                                                       colsample_bytree=0.7,
                                                                       device=None,
                                                                       early_stopping_rounds=N
one,
                                                                       enable_categorical=Fals
e,
                                                                       eval_metr...
                                                                       missing=nan,
                                                                       monotone_constraints=No
ne,
                                                                       multi_strategy=None,
                                                                       n_estimators=220,
                                                                       n_jobs=None,
                                                                       num_parallel_tree=None,
...)),
                                                         ('lightgbm',
                                                          LGBMRegressor(learning_rate=0.2,
                                                                        max_depth=6,
                                                                        n_estimators=50,
                                                                        num_leaves=6,
                                                                        reg_alpha=0))]),
                   n_jobs=-1,
                   param_distributions={'weights':         [[0.2,          0.6,         0.2],
                                                     [0.6,          0.2,          0.2],
                                                     [0.4,          0.4,          0.4],
                                                     [0.2,         0.4,          0.5]]},
                   scoring='neg_mean_absolute_error')

# figure out best hyperparameters
print(voting_model_random.best_estimator_)

VotingRegressor(estimators=[('Gradient Boost',
                             GradientBoostingRegressor(alpha=0.5,
                                                       n_estimators=30)),
                            ('xgboost',
                             XGBRegressor(alpha=0.5, base_score=None,
                                          booster=None, callbacks=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=0.7, device=None,
                                          early_stopping_rounds=None,
                                          enable_categorical=False,
                                          eval_metric=None, feature_types=None,
                                          gamma=0...
                                          max_cat_threshold=None,
                                          max_cat_to_onehot=None,
                                          max_delta_step=None, max_depth=5,
                                          max_leaves=None,
                                          min_child_weight=1.7817, missing=nan,
                                          monotone_constraints=None,
                                          multi_strategy=None, n_estimators=220,
                                          n_jobs=None, num_parallel_tree=None, ...)),
                            ('lightgbm',
                             LGBMRegressor(learning_rate=0.2, max_depth=6,
                                           n_estimators=50, num_leaves=6,
                                       reg_alpha=0))],
                weights=[0.2, 0.4, 0.5])

# create model instance after hyperparameters tuning
voting_model = VotingRegressor(estimators=[('Gradient Boost',gbr_model),('xgboost', xgb_model)
, ('lightgbm', lgb_model)],weights=[0.2, 0.4, 0.5])

# Validation of Voting Regressor
voting_r2_cv = cross_val_score(voting_model,X_train_scaled,y_train, scoring="r2",cv = 5)
voting_mae_cv = -cross_val_score(voting_model,X_train_scaled,y_train, scoring="neg_mean_absolu
te_error",cv = 5)
voting_rmse_cv = -cross_val_score(voting_model,X_train_scaled,y_train, scoring="neg_root_mean_
squared_error",cv = 5)
print("R2 scores:", voting_r2_cv)
print("Mean R2:", np.mean(voting_r2_cv))
print("MAE scores:", voting_mae_cv)
print("Mean MAE:", np.mean(voting_mae_cv))
print("RSME scores:", voting_rmse_cv)
print("Mean RSME:", np.mean(voting_rmse_cv))
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000476
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2828
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.030302
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000446
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.020671
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000471
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2822
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 64
[LightGBM] [Info] Start training from score 12.041320
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000441
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.034734
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000467
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2835
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.028148
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000550
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2828
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
```

```
[LightGBM] [Info] Start training from score 12.030302
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000615
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.020671
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000497
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2822
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 64
[LightGBM] [Info] Start training from score 12.041320
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000459
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.034734
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000485
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2835
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.028148
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000463
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2828
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.030302
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000467
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 65
[LightGBM] [Info] Start training from score 12.020671
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000452
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2822
[LightGBM] [Info] Number of data points in the train set: 934, number of used features: 64
[LightGBM] [Info] Start training from score 12.041320
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000469
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2824
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.034734
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000511
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
```

```
[LightGBM] [Info] Total Bins 2835
[LightGBM] [Info] Number of data points in the train set: 935, number of used features: 65
[LightGBM] [Info] Start training from score 12.028148
R2 scores: [0.90759188 0.87050146 0.89093514 0.88825343 0.90530335]
Mean R2: 0.892517051729099
MAE scores: [0.0799881  0.09111986 0.09358461 0.09280279 0.07613774]
Mean MAE: 0.08672662128709613
RSME scores: [0.11446415 0.13477808 0.13525746 0.12729988 0.10882885]
Mean RSME: 0.1241256842819197
```

```
# Fit on data
voting_model.fit(X_train_scaled, y_train)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000527
seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2926
[LightGBM] [Info] Number of data points in the train set: 1168, number of used features: 65
[LightGBM] [Info] Start training from score 12.031035

VotingRegressor(estimators=[('Gradient Boost',
                             GradientBoostingRegressor(alpha=0.5,
                                                       n_estimators=30)),
                            ('xgboost',
                             XGBRegressor(alpha=0.5, base_score=None,
                                          booster=None, callbacks=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=0.7, device=None,
                                          early_stopping_rounds=None,
                                          enable_categorical=False,
                                          eval_metric=None, feature_types=None,
                                          gamma=0...
                                          max_cat_threshold=None,
                                          max_cat_to_onehot=None,
                                          max_delta_step=None, max_depth=5,
                                          max_leaves=None,
                                          min_child_weight=1.7817, missing=nan,
                                          monotone_constraints=None,
                                          multi_strategy=None, n_estimators=220,
                                          n_jobs=None, num_parallel_tree=None, ...)),
                            ('lightgbm',
                             LGBMRegressor(learning_rate=0.2, max_depth=6,
                                           n_estimators=50, num_leaves=6,
                                           reg_alpha=0))],
                weights=[0.2, 0.4, 0.5])
```

```
# Voting prediction
voting_model_y_pred = voting_model.predict(X_test_scaled)

# Calculate and print evaluation score
voting_r2_score = r2_score(y_test, voting_model_y_pred)
voting_mae_score = mean_absolute_error(y_test, voting_model_y_pred)
voting_rmse_score = np.sqrt(mean_squared_error(y_test, voting_model_y_pred))
# Print scores
print("R2 score:", voting_r2_score)
print("MAE scores:", voting_mae_score)
print("RMSE score:", voting_rmse_score)
```

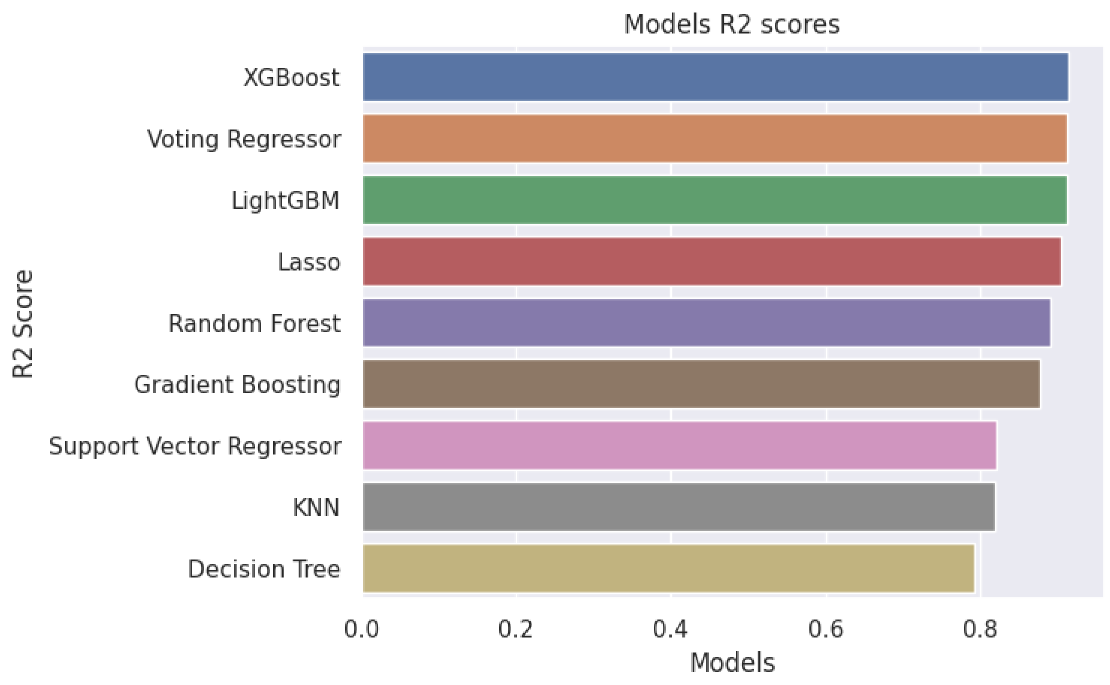
```
R2 score: 0.9122033750178448
MAE scores: 0.08632119223256815
RMSE score: 0.12174130416172735
```

###Summary

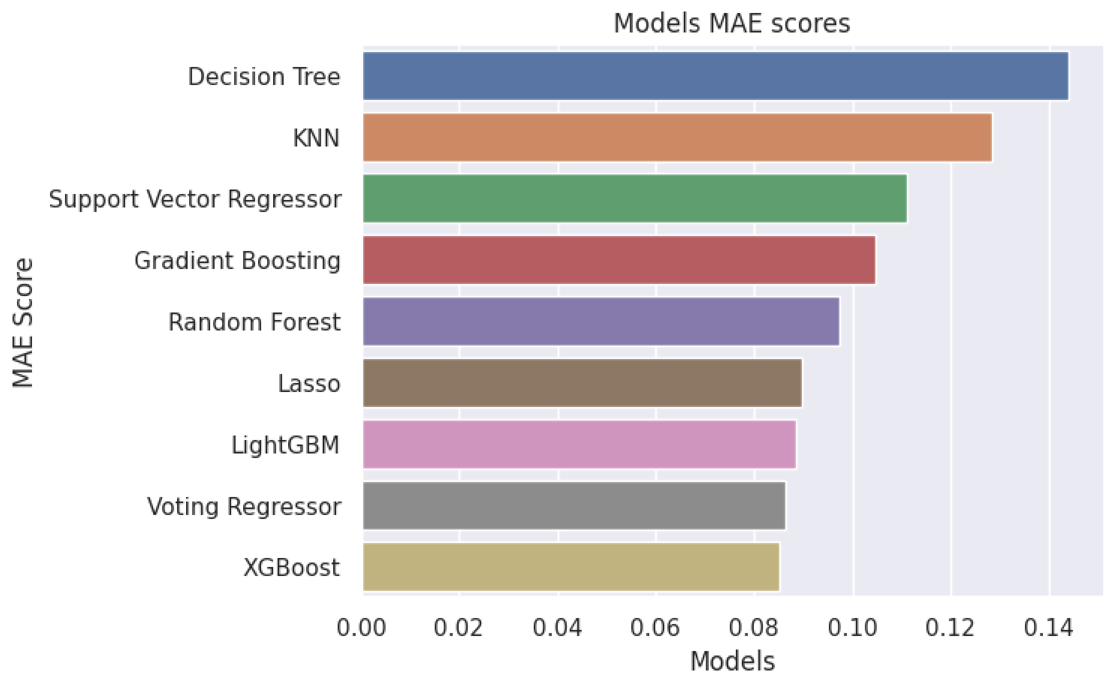
```
# summary result table
summary = pd.DataFrame({
    'Model Name': ['Lasso', 'Decision Tree', 'Support Vector Regressor', 'KNN','Random Forest'
,'Gradient Boosting','XGBoost', 'LightGBM','Voting Regressor'],
    'R2 Score': [lasso_r2_score, dtr_r2_score, svr_r2_score, knn_r2_score,rf_r2_score,gbr_r2_s
core,xgb_r2_score, lgb_r2_score,voting_r2_score],
    'MAE Score': [lasso_mae_score, dtr_mae_score, svr_mae_score, knn_mae_score,rf_mae_score,gb
r_mae_score,xgb_mae_score, lgb_mae_score,voting_mae_score],
    'RMSE Score': [lasso_rmse_score, dtr_rmse_score, svr_rmse_score, knn_rmse_score,rf_rmse_sc
ore,gbr_rmse_score,xgb_rmse_score, lgb_rmse_score,voting_rmse_score]
})
df_summary = pd.DataFrame(summary)
df_summary
```

```
                 Model Name  R2 Score  MAE Score  RMSE Score
0                     Lasso  0.904482   0.089703    0.126982
1             Decision Tree  0.791153   0.144035    0.187764
2  Support Vector Regressor  0.821243   0.111036    0.173712
3                       KNN  0.818839   0.128605    0.174876
4             Random Forest  0.889963   0.097287    0.136291
5         Gradient Boosting  0.877650   0.104677    0.143715
6                   XGBoost  0.913757   0.085211    0.120659
7                  LightGBM  0.910886   0.088508    0.122651
8          Voting Regressor  0.912203   0.086321    0.121741
```

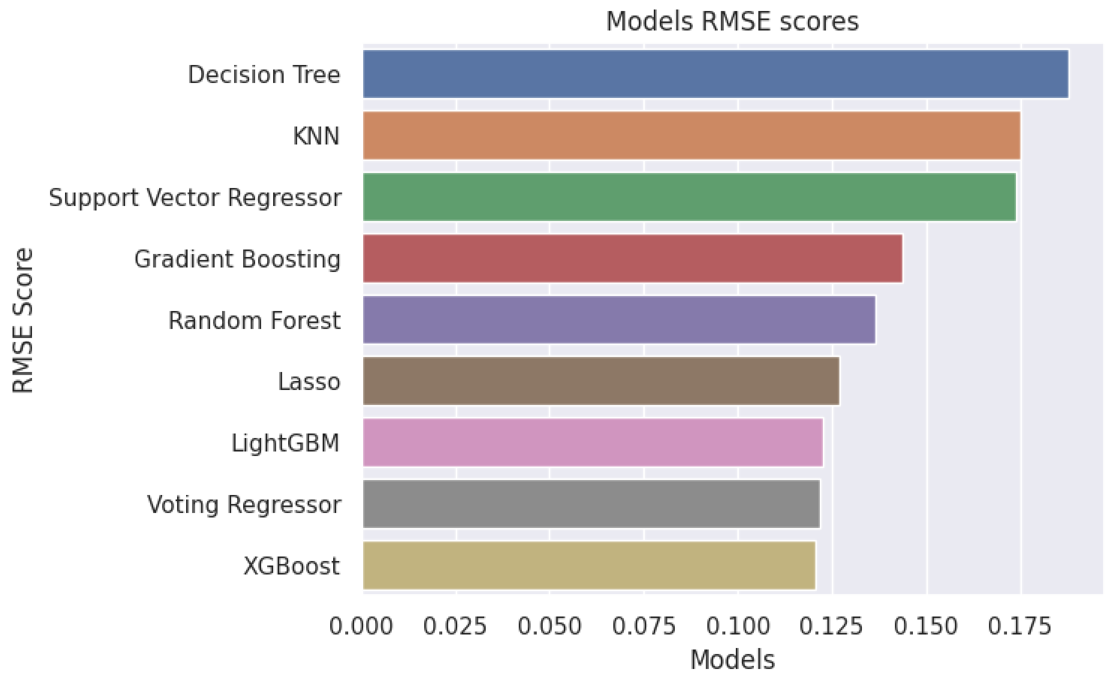
```
# Create Bar plot for models according to R2.
df_summary = df_summary.sort_values(by='R2 Score', ascending=False)
sns.barplot(y = df_summary['Model Name'],x = df_summary['R2 Score'] ,data= df_summary, orient
= 'h')
plt.title(f"Models R2 scores")
plt.xlabel('Models')
plt.ylabel("R2 Score")
plt.show()
```



```
# Create Bar plot for models according to MAE.
df_summary = df_summary.sort_values(by='MAE Score', ascending=False)
sns.barplot(y = df_summary['Model Name'],x = df_summary['MAE Score'] ,data= df_summary, orient
= 'h')
plt.title(f"Models MAE scores")
plt.xlabel('Models')
plt.ylabel("MAE Score")
plt.show()
```



```
# Create Bar plot for models according to RMSE.
df_summary = df_summary.sort_values(by='RMSE Score', ascending=False)
sns.barplot(y = df_summary['Model Name'],x = df_summary['RMSE Score'] ,data= df_summary, orien
t = 'h')
plt.title(f"Models RMSE scores")
plt.xlabel('Models')
plt.ylabel("RMSE Score")
plt.show()
```



```
#XGBoost Feature importance
for score, name in zip(xgb_model.feature_importances_,X_train_scaled):
  print(round(score,2),name)
```

```
0.0 MSSubClass
0.01 MSZoning
0.0 LotFrontage
0.0 LotArea
0.0 Street
0.0 Alley
0.0 LotShape
0.01 LandContour
0.0 Utilities
0.0 LotConfig
0.0 LandSlope
0.0 Neighborhood
0.0 Condition1
0.0 Condition2
0.0 BldgType
0.0 HouseStyle
0.33 OverallQual
0.01 OverallCond
0.03 YearBuilt
0.01 YearRemodAdd
0.0 RoofStyle
0.0 RoofMatl
0.0 Exterior1st
```

```
0.0 Exterior2nd
0.0 MasVnrType
0.0 MasVnrArea
0.14 ExterQual
0.0 ExterCond
0.0 Foundation
0.01 BsmtQual
0.0 BsmtCond
0.0 BsmtExposure
0.0 BsmtFinType1
0.01 BsmtFinSF1
0.0 BsmtFinType2
0.0 BsmtFinSF2
0.0 BsmtUnfSF
0.02 TotalBsmtSF
0.0 Heating
0.0 HeatingQC
0.03 CentralAir
0.0 Electrical
0.01 1stFlrSF
0.01 2ndFlrSF
0.0 LowQualFinSF
0.05 GrLivArea
0.0 BsmtFullBath
0.0 BsmtHalfBath
0.02 FullBath
0.01 HalfBath
0.0 BedroomAbvGr
0.0 KitchenAbvGr
0.05 KitchenQual
0.0 TotRmsAbvGrd
0.0 Functional
0.03 Fireplaces
0.0 FireplaceQu
0.02 GarageType
0.0 GarageYrBlt
0.01 GarageFinish
0.11 GarageCars
0.01 GarageArea
0.01 GarageQual
0.02 GarageCond
0.0 PavedDrive
0.0 WoodDeckSF
0.0 OpenPorchSF
0.0 EnclosedPorch
0.0 3SsnPorch
0.0 ScreenPorch
0.0 PoolArea
0.0 PoolQC
0.0 Fence
0.0 MiscFeature
0.0 MiscVal
0.0 MoSold
0.0 YrSold
0.0 SaleType
0.0 SaleCondition
```