“HOUSE PRICE PREDICTION USING ML”

An Internship project report submitted in partial fulfillment of the requirements forthe award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE ENGINEERING

Internship

By

Name – **Gopal Kumar**

College – **CUTM**

Under the guidance of

INTERNFORTE

INTERNFORTE

Fourth Floor, Classic Arena, Hosur Rd, AECS Layout
Singasandra, Bengaluru, Karnataka 560068

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **InternForte,** Assistant Professor, Department of Computer Science and Engineering, for his/her guidance with unsurpassed knowledge and immense encouragement. We are grateful to **InternForte**, Head of the Department, Computer Science and Engineering, for providing us with the required facilities for the completion of the project work. We are very much thankful to the Principal **InternForte,** fortheir encouragement and cooperation to carry out this work. We express our thanks to Project Coordinator **InternForte** for her Continuous support and encouragement. We thank all **teaching faculty** of Department of CSE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **InternForte** of the Department of CSE for providing great assistance in accomplishment of our project. We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENT

Gopal Kumar

CERTIFICATE

This is to certify that the project report entitled "**HOUSE PRICE PREDICTION USING ML"**submitted by **Gopal Kumar** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science Engineering** of Centurion University of Technology and Management PKD Campus Odisha is a record of Bonafede work carried out under my guidance and supervision.

Project Guide **Head of the Department**

INTERNFORTE **INTERNFORTE**

DECLARATION

We **GOPALKUMAR**, of final semester(6^{th}) B.Tech. in the department of Computer Science and Engineering from CUTM, PARALAKHEMUNDI, hereby declare that the project work entitled **HOUSE PRICE PREDICTION USING ML** is carried out by us and submitted in partial fulfillment of the requirementsfor the award of **Bachelor of Technology in Computer Science Engineering** , under Centurion University of Technology and Management Paralakhemundi Campus Odisha during the academic year 2022-23 and has not been submitted to any other university for the award of any kind of degree.

Gopal Kumar

ABSTRACT

The prediction of house prices plays a vital role in real estate market analysis, investment decisions, and property valuation. In recent years, machine learning (ML) techniques have emerged as powerful tools for accurately forecasting house prices. This study presents a comprehensive analysis of various ML algorithms employed for house price prediction and compares their performance to identify the most effective model. The dataset used in this study consists of a wide range of features related to houses, such as location, size, number of rooms, amenities, and proximity to essential facilities. Initially, the data is preprocessed, including handling missing values, feature scaling, and categorical variable encoding. The dataset is then split into training and testing subsets to evaluate the models. Several popular ML algorithms, including linear regression, decision trees, random forests, support vector regression, and neural networks, are implemented and trained using the training data. Hyper parameter tuning techniques, such as grid search or Bayesian optimization, are applied to optimize the models' performance. The performance of the ML models is evaluated using various metrics, including mean squared error (MSE), mean absolute error (MAE), and coefficient of determination (R^2). The results are analyzed and compared to identify the model that provides the most accurate predictions. Furthermore, the study explores the importance of features in predicting house prices using techniques such as feature importance ranking and feature selection. This analysis helps in understanding the significant factors influencing house prices and provides valuable insights for buyers, sellers, and real estate professionals. The experimental results demonstrate the effectiveness of ML techniques in house price prediction. The findings indicate that [mention the model with the best performance] outperforms other models, achieving the lowest MSE, MAE, and highest R^2 scores. Moreover, the feature importance analysis reveals that location, size, and amenities significantly impact house prices.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In recent years, the field of machine learning has revolutionized various industries, and one prominent application is the prediction of house prices. House price prediction plays a crucial role in real estate, finance, and investment decision-making processes. By leveraging the power of machine learning algorithms, we can develop accurate and reliable models that estimate the value of residential properties. House price prediction involves analyzing a multitude of factors that influence property prices, such as location, size, number of bedrooms and bathrooms, amenities, neighborhood characteristics, and economic indicators. Traditional methods of property valuation relied on manual assessment by real estate agents or appraisers, but machine learning offers a data-driven approach that can yield more objective and precise predictions. The key advantage of using machine learning for house price prediction is its ability to identify complex patterns and relationships within large datasets. By training a model on historical housing data, we can uncover hidden correlations and gain insights into how different features impact property values. This enables us to build predictive models capable of estimating house prices accurately for new and unseen data. Various machine learning techniques can be applied to house price prediction, including regression algorithms like linear regression, decision trees, random forests, support vector regression, and neural networks. These models are trained using labeled datasets, where each data point consists of input features (e.g., number of rooms, location) and the corresponding house price.

Once the model is trained, it can be deployed to make predictions on new instances by inputting the relevant features. This empowers potential home buyers, sellers, and investors to make informed decisions based on estimated property values. Additionally, real estate agents and financial institutions can leverage these predictions to optimize their operations, improve customer service, and mitigate risks.

However, it's important to note that house price prediction using machine learning is not without its challenges. Obtaining accurate and comprehensive datasets, dealing with missing or inconsistent data, handling outliers, and selecting the most appropriate features are some of the hurdles that need to be addressed for reliable predictions.

CHAPTER 2
EXISTING METHOD

Existing methods for house price prediction using machine learning typically involve the following steps:

Data Collection: The first step is to gather a comprehensive dataset that includes relevant features and corresponding house prices. This data can be obtained from various sources, such as real estate databases, government records, or online platforms.

Data Preprocessing: Once the dataset is collected, it needs to be preprocessed to ensure its quality and suitability for training machine learning models. This includes handling missing values, dealing with outliers, normalizing or scaling features, and encoding categorical variables.

Feature Selection/Engineering: Feature selection involves identifying the most relevant features that significantly contribute to predicting house prices. This step helps reduce dimensionality and improve the model's efficiency. Feature engineering involves transforming or creating new features from the existing ones to capture more meaningful information.

Model Selection: There are several regression algorithms available for house price prediction, and the choice of model depends on the dataset size, complexity, and performance requirements. Commonly used models include linear regression, decision trees, random forests, support vector regression, and neural networks.

Model Training: The selected model is trained using the preprocessed dataset. The dataset is divided into a training set and a validation set to evaluate the model's performance during training. The model learns the underlying patterns and relationships between the input features and the house prices.

Model Evaluation: After training, the model's performance is evaluated using various metrics such as mean squared error (MSE), mean absolute error (MAE), or R-squared value. These metrics quantify how well the model predicts house prices compared to the actual values.

Hyper-parameter Tuning: Many machine learning models have hyperparameters that control their behavior and performance. Hyperparameter tuning involves optimizing these parameters to improve the model's accuracy. Techniques like grid search, random search, or Bayesian optimization can be employed for this purpose.

Model Deployment: Once the model is trained and evaluated, it can be deployed to make predictions on new, unseen data. This allows users to input the relevant features of a house and obtain an estimated price.

Monitoring and Updating: The deployed model should be continuously monitored to ensure its accuracy and reliability. As new data becomes available, the model can be updated or retrained periodically to incorporate the latest information and improve its predictive performance.

These existing methods provide a framework for utilizing machine learning algorithms to predict house prices accurately. However, the specific implementation and choice of techniques may vary depending on the dataset, resources, and specific requirements of the application.

2.1 Industry Profile

The industry of house price prediction using machine learning has gained significant traction in recent years, transforming the real estate and finance sectors. This emerging field combines the expertise of data science and domain knowledge to develop accurate models that estimate property values. Let's explore the industry profile for house price prediction using machine learning.

Market Overview:
The real estate industry has always been driven by the demand for accurate property valuation. Traditional methods of valuing houses relied on subjective assessments, which often led to inconsistencies and inefficiencies. Machine learning-based house price prediction offers a data-driven and objective approach, revolutionizing the way properties are valued and facilitating more informed decision-making.

Key Players:
Real Estate Agencies: Real estate agencies have recognized the potential of machine learning in predicting house prices and have started incorporating such models into their operations. These agencies leverage machine learning algorithms to provide accurate property valuations to their clients, enhance marketing strategies, and optimize their investment decisions.

PropTech Companies: PropTech (Property Technology) companies specialize in developing innovative technological solutions for the real estate industry. Many PropTech startups have emerged, focusing specifically on house price prediction using machine learning. They create platforms and tools that enable real estate professionals, buyers, and sellers to access reliable predictions and market insights.

Financial Institutions: Banks, mortgage lenders, and insurance companies heavily rely on accurate house price predictions to assess risks, determine loan amounts, and make informed investment decisions. They leverage machine learning models to analyze vast amounts of data and estimate property values, ensuring their financial operations are optimized and risk is mitigated.

Data Analytics and Consulting Firms: Data analytics and consulting firms play a crucial role in the house price prediction industry. They provide expertise in data analysis, model development, and deployment, helping businesses implement machine learning-based solutions effectively. These firms assist in gathering relevant data, preprocessing and cleaning it, and building robust predictive models.

Technological Advancements:
The rapid advancement of machine learning techniques and computational power has significantly contributed to the growth of house price prediction using ML. State-of-the-art algorithms such as deep learning neural networks, ensemble methods like random forests and gradient boosting, and regression models have become widely adopted in the industry. Additionally, advancements in cloud computing have facilitated scalable and cost-effective model training and deployment.

Data Availability and Integration:
The availability and integration of diverse datasets have been instrumental in improving the accuracy of house price prediction models. Besides traditional property features, datasets now include a wide range of factors, such as neighborhood characteristics, economic indicators, local amenities, and geospatial information. Integration of these datasets enhances the predictive power of the models and enables more comprehensive property valuation.

Challenges:
While the industry of house price prediction using machine learning has shown promising growth, it faces several challenges. Some of the key challenges include obtaining reliable and comprehensive datasets, dealing with missing or inconsistent data, addressing biases in the data, handling outliers and extreme market conditions, and ensuring model interpretability and transparency.

Future Outlook:
The future of house price prediction using machine learning looks promising. As technology continues to advance, machine learning models will become more accurate and robust. Integration with other emerging technologies, such as augmented reality (AR) and virtual reality (VR), can enhance the visualization and immersive experience of property valuation. Furthermore, as more real estate transactions move online, the demand for accurate and instant house price predictions will increase, driving further innovation in the industry.

CHAPTER 3
RESEARCH METHODOLOGY

Research methodology for house price prediction using machine learning typically involves the following steps:

Problem Definition: Clearly define the research problem, which is to predict house prices using machine learning techniques.

Data Collection: Gather a comprehensive dataset that includes various features of houses (e.g., size, location, number of bedrooms, amenities, etc.) along with their corresponding prices. The data can be obtained from real estate websites, public datasets, or by collecting it directly through surveys or other means.

Data Preprocessing: Clean and preprocess the collected data to ensure its quality and suitability for the machine learning models. This step may involve handling missing values, removing outliers, normalizing or scaling features, and encoding categorical variables.

Feature Selection/Engineering: Analyze the collected data and select relevant features that are most likely to influence house prices. Additionally, create new features through feature engineering techniques to enhance the predictive power of the models. For example, you could derive a feature like the price per square foot from the original features.

Splitting the Data: Divide the dataset into two subsets: a training set and a testing/validation set. The training set will be used to train the machine learning models, while the testing/validation set will be used to evaluate their performance.

Model Selection: Choose suitable machine learning algorithms for house price prediction. Commonly used models include linear regression, decision trees, random forests, support vector machines (SVM), and neural networks. The selection of models depends on the characteristics of the dataset and the specific requirements of the problem.

Model Training: Train the selected machine learning models using the training set. The models will learn the underlying patterns and relationships between the features and the house prices during this phase.

Model Evaluation: Evaluate the trained models using appropriate evaluation metrics such as mean squared error (MSE), mean absolute error (MAE), or R-squared. These metrics will assess how well the models perform in predicting house prices.

Hyperparameter Tuning: Fine-tune the hyperparameters of the selected models to optimize their performance. This step involves adjusting the parameters that are not learned during training, such as the learning rate, regularization strength, or maximum tree depth.

Model Validation: Validate the performance of the tuned models using the testing/validation set. This step provides an unbiased assessment of how well the models generalize to unseen data.

Model Deployment: Once a satisfactory model is selected, deploy it in a production environment, making it available for real-time house price predictions.

Continuous Monitoring and Improvement: Monitor the performance of the deployed model and gather feedback. Iteratively refine the model or develop new models to improve accuracy and adapt to changing trends in the housing market.

3.1 Title of the Study

"Predicting House Prices: A Machine Learning Approach for Real Estate Market Analysis"

3.2 Statement of the problem

The problem at hand is to develop a machine learning model that can accurately predict house prices based on a set of relevant features. The objective is to provide home buyers, sellers, and real estate agents with a reliable tool to estimate the market value of residential properties.

The model will be trained on a dataset consisting of historical housing data, which includes attributes such as the size of the property, number of bedrooms and bathrooms, location, proximity to amenities, and other relevant factors that influence property prices. The dataset will also include the actual sale prices of the houses.

The goal is to create a regression model that can take these features as input and accurately predict the corresponding house prices. The model should be able to handle different types of input data, including numerical and categorical variables, and account for potential non-linear relationships between the features and the target variable.

The accuracy and performance of the model will be evaluated using appropriate metrics such as mean squared error (MSE) or root mean squared error (RMSE). The ultimate aim is to develop a model that can generalize well to unseen data and provide reliable price predictions for new houses.

This problem is significant as accurate house price predictions can benefit various stakeholders in the real estate industry. Home buyers can use the model to assess whether a listed price is fair and make informed decisions. Sellers can determine an appropriate listing price based on market trends and property attributes. Real estate agents can leverage the model to provide reliable advice to clients and streamline the buying and selling process.

By addressing this problem, we aim to provide a valuable tool that improves transparency and efficiency in the real estate market, empowering individuals with data-driven insights for making informed decisions about residential property transactions.

3.3 Objectives of the Study

The objectives of a study on house price prediction using machine learning can vary depending on the specific context and goals of the research.

the specific objectives of a study on house price prediction using machine learning will depend on the research context, available data, and the needs of stakeholders involved in the housing market.

3.4 Research Design

Designing a research study for house price prediction using machine learning involves several steps. Remember that the specific details of the research design may vary depending on your specific research objectives, the available data, and the machine learning techniques you choose to employ.

3.5 Hypotheses

When it comes to predicting house prices using machine learning, several hypotheses can be explored. Here are some common hypotheses:

Linear Relationship: The hypothesis assumes that there is a linear relationship between the input features (e.g., number of bedrooms, square footage, location) and the target variable (house price). It assumes that the relationship can be adequately captured by a linear regression model.

Non-Linear Relationship: This hypothesis suggests that the relationship between the input features and house prices is non-linear. It implies that more complex models, such as decision trees, random forests, or neural networks, might be better suited for capturing the underlying patterns.

Feature Importance: The hypothesis posits that certain features have a stronger influence on house prices compared to others. It suggests that by identifying and focusing on the most important features, the predictive performance of the model can be improved.

Neighborhood Effect: This hypothesis suggests that the location and neighborhood characteristics play a significant role in determining house prices. It implies that including features related to neighborhood demographics, amenities, crime rates, and proximity to schools or transportation can enhance the predictive power of the model.

Temporal Factors: This hypothesis considers the impact of time-related factors on house prices. It assumes that trends, seasonality, or macroeconomic indicators can influence housing markets. Incorporating temporal features, such as year, month, or historical price data, can help capture these factors.

Interaction Effects: This hypothesis proposes that the interaction between certain features may have a substantial impact on house prices. For example, the combination of the number of bedrooms and bathrooms might be more influential than considering them separately. Including interaction terms or using more sophisticated models capable of capturing interactions, like polynomial regression or deep learning models, can test this hypothesis.

Data Quality: This hypothesis suggests that the quality and completeness of the dataset have a direct impact on prediction accuracy. It assumes that ensuring data cleanliness, handling missing values, and addressing outliers can lead to improved model performance.

These hypotheses provide a starting point for exploring and developing machine learning models for house price prediction. The choice of hypothesis to investigate depends on the specific dataset, available features, and the goals of the prediction task.

3.6 Sampling Technique

When it comes to predicting house prices using machine learning, sampling techniques play a crucial role in creating a representative and unbiased dataset. Here are some commonly used sampling techniques:

Simple Random Sampling: This technique involves randomly selecting a subset of data points from the entire dataset. Each data point has an equal chance of being selected. Simple random sampling is easy to implement but may not be efficient for large datasets.

Stratified Sampling: Stratified sampling involves dividing the dataset into homogeneous groups called strata and then selecting a proportional number of samples from each stratum. For house price prediction, you could divide the dataset based on relevant features such as location, property type, or square footage to ensure that each stratum is represented in the sample.

Cluster Sampling: Cluster sampling involves dividing the dataset into clusters, such as geographical regions, and randomly selecting entire clusters to include in the sample. This technique is useful when the dataset is geographically distributed, and it helps capture regional variations in house prices.

Oversampling and Under sampling: These techniques are used when dealing with imbalanced datasets, where one class (e.g., expensive houses) is underrepresented compared to another (e.g., affordable houses). Oversampling involves randomly duplicating minority class samples, while undersampling involves randomly removing majority class samples. These techniques can help balance the dataset and improve prediction performance.

Systematic Sampling: Systematic sampling involves selecting data points at regular intervals from an ordered list. For house price prediction, you could order the dataset based on a relevant feature such as date of sale and then select every nth data point to create the sample.

Cross-validation: While not a sampling technique in the traditional sense, cross-validation is commonly used in machine learning for model evaluation. It involves dividing the dataset into multiple folds and using each fold as both training and validation data. This technique helps assess the model's performance on different subsets of the data.

The choice of sampling technique depends on the characteristics of your dataset, the specific problem at hand, and the goals of your analysis. It is essential to carefully consider the advantages and limitations of each technique and select the one that best suits your needs.

CHAPTER 4
PROPOSED SYSTEM

To develop a system for house price prediction using machine learning, you can follow these general steps:

Data Collection: Gather a comprehensive dataset of housing information, including features such as the number of bedrooms, bathrooms, square footage, location, amenities, and historical sales prices. You can obtain this data from real estate websites, public records, or other reliable sources.

Data Preprocessing: Clean and preprocess the collected data to ensure its quality and compatibility with the machine learning algorithms. Handle missing values, outliers, and perform feature engineering if necessary. This step may also involve transforming categorical variables into numerical representations through techniques like one-hot encoding.

Feature Selection: Analyze the dataset to identify the most relevant features that contribute significantly to the house prices. You can employ techniques like correlation analysis, feature importance, or dimensionality reduction algorithms to select the optimal set of features for your model.

Model Selection: Choose an appropriate machine learning algorithm for your prediction task. Regression algorithms such as linear regression, decision trees, random forests, or gradient boosting can be effective for house price prediction. Consider the strengths and weaknesses of each algorithm and select the one that best suits your dataset and requirements.

Model Training: Split the preprocessed data into training and testing sets. Use the training set to train the chosen machine learning model. During training, the model will learn the patterns and relationships between the features and target variable (house prices).

Model Evaluation: Evaluate the trained model's performance using suitable evaluation metrics such as mean squared error (MSE), root mean squared error (RMSE), or R-squared. Assess how well the model generalizes to unseen data by using the testing set.

Model Optimization: Fine-tune the model by adjusting hyperparameters to optimize its performance. This can be done through techniques like grid search or randomized search, where different combinations of hyperparameters are evaluated to find the optimal configuration.

Deployment: Once you are satisfied with the model's performance, deploy it to make predictions on new, unseen data. Develop a user-friendly interface or API that takes input features (e.g., number of bedrooms, square footage) and returns the predicted house price based on the trained model.

Maintenance and Updates: Regularly monitor and maintain the deployed system, retraining the model periodically with new data to keep it up to date. Stay informed about new developments in the field of machine learning and consider incorporating improvements or newer algorithms as they become available.

Remember, the success of your house price prediction system depends on the quality of the data, the choice of features, and the selection and fine-tuning of the machine learning model.

4.1 Statistical Tools Used

There are several statistical tools that can be used for house price prediction using machine learning. Here are some commonly used tools:

Linear Regression: Linear regression is a simple and widely used statistical tool for predicting house prices. It assumes a linear relationship between the independent variables (such as square footage, number of bedrooms, etc.) and the dependent variable (house price). The model estimates the coefficients of the independent variables to make predictions.

Multiple Regression: Multiple regression extends linear regression by allowing multiple independent variables to be used in the prediction model. This tool is useful when there are multiple factors affecting house prices, such as location, amenities, and neighborhood characteristics.

Decision Trees: Decision trees are non-linear statistical tools that can be used for house price prediction. They partition the data based on different attributes and create a tree-like structure to make predictions. Decision trees can capture complex interactions between variables and handle both categorical and numerical data.

Random Forests: Random forests are an ensemble learning method that combines multiple decision trees to make predictions. They reduce overfitting and improve accuracy by aggregating the predictions of multiple individual trees. Random forests are robust and can handle large datasets with high-dimensional features.

Support Vector Regression (SVR): SVR is a statistical tool based on support vector machines that can be used for regression tasks, including house price prediction. SVR finds the best-fitting hyperplane in a high-dimensional space to minimize the error between the predicted and actual house prices.

Gradient Boosting: Gradient boosting is another ensemble learning method that combines weak prediction models, such as decision trees, into a strong predictive model. It iteratively improves the model by minimizing a loss function and adjusting the weights of the individual models. Gradient boosting is known for its high predictive accuracy.

Neural Networks: Neural networks, specifically deep learning models, can also be used for house price prediction. They are powerful tools for capturing complex patterns in the data and can handle large datasets. Neural networks can be trained using various architectures, such as feedforward neural networks, convolutional neural networks (CNNs), or recurrent neural networks (RNNs).

These are just a few examples of the statistical tools used for house price prediction using machine learning. The choice of tool depends on the specific requirements of the problem, the available data, and the desired level of accuracy and interpretability.

4.2 Scope of the Study

The scope of a study on house price prediction using machine learning can vary depending on the specific objectives and constraints of the research. It's important to note that the specific scope of the study may be adjusted based on available resources, time constraints, and the objectives of the research.

4.3 Limitations of the Study

When conducting a study on house price prediction using machine learning, there can be several limitations that researchers should acknowledge. It is essential to acknowledge these limitations and potential sources of error when conducting a study on house price prediction using machine learning. Researchers should be cautious about the assumptions made, validate the model's performance on unseen data, and consider external factors that may impact the predictions.

4.4 Data Collection and Analysis

Data Collection and Analysis for House Price Prediction using Machine Learning involves several steps. Remember that this is a general outline, and specific steps and techniques may vary based on the dataset, problem statement, and chosen machine learning algorithms.

4.5 Findings

House price prediction using machine learning has been a popular research area, and several findings and approaches have emerged in recent years. It's important to note that the effectiveness of these findings may vary depending on the specific dataset, location, and context. Researchers and practitioners continue to explore new techniques and approaches to improve house price prediction using machine learning.

4.6 Suggestions

When it comes to house price prediction using machine learning, there are several approaches you can consider. Remember, the success of your house price prediction model depends on the quality and relevance of your data, as well as the choice of the appropriate machine learning techniques and feature engineering methods.

CHAPTER 5
EXPERIMENT ANALYSIS

5.1 System Configuration

This project can run on commodity hardware. We ran entire projecton an Intel I5 processor with 8 GB Ram, 2 GB Nvidia Graphic Processor, it also has 2 cores which runs at 1.7 GHz, 2.1 GHz respectively. First part of the is training phase which takes 10-15 mins of time and the second part is testing part which only takes few seconds to make predictions and calculate accuracy.

5.1.1 Hardware Requirements:

- RAM: 4 GB
- Storage: 500 GB
- CPU: 2 GHz or faster
- Architecture: 32-bit or 64-bit

5.1.2 Software requirements

- Python 3.10.0 in Google Colab is used for data pre-processing, model training andprediction.
- Operating System: windows 10 and above or Linux based OS or MAC OS.

5.3 Functional requirements

Functional requirements describe what the software should do (the functions). Think about the core operations.

Because the “functions” are established before development, functional requirements should be written in the future tense. In developing the software for **HOUSE PRICE PREDICTION,**some of the functional requirements could include:

- The software shall accept the tw_spydata_raw.csv dataset as input.
- The software should shall do pre-processing (like verifying for missing data values) on input for model training.
- The software shall use MULTIPLE LINEAR REGRESSION as main component of thesoftware.
- It processes the given input data by producing the most possible outcomes of a House prediction expected. Notice that each requirement is directly related to what we expect the software to do. Theyrepresent some of the core functions.

5.4 Non-Functional requirements

Product properties

- Usability: It defines the user interface of the software in terms of simplicity of understanding the user interface of **house price prediction** software, for any kind of profit trader and other stakeholders in profit prediction of 50 companies.
- Efficiency: maintaining the possible highest accuracy in the 50 companies inshortest time with available data.

Performance: It is a quality attribute of the profit prediction software that describes the responsiveness to various user interactions with it.

5.5 Sample code with Output

Step 1: Import the necessary libraries

```
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import calendar

from pandas.api.types import CategoricalDtype

from sklearn.preprocessing import StandardScaler
```

Step 2: Load and explore the dataset.

```
[2]: train_data_path = r"C:\Users\gk521\House_Data\train.csv"
test_data_path = r"C:\Users\gk521\House_Data\test.csv"

df_train = pd.read_csv(train_data_path)
df_test = pd.read_csv(test_data_path)

print("Shape of df_train:", df_train.shape)
print("Shape of df_test:", df_test.shape)
```

```
Shape of df_train: (1168, 81)
Shape of df_test: (292, 80)
```

[3]:
```
df_train
```

[3]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	...
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	...
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	...
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	...
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	...
...	...	...	...	...	...	...	...	...	...	...	...
1163	289	20	RL	NaN	9819	Pave	NaN	IR1	Lvl	AllPub	...
1164	554	20	RL	67.0	8777	Pave	NaN	Reg	Lvl	AllPub	...
1165	196	160	RL	24.0	2280	Pave	NaN	Reg	Lvl	AllPub	...
1166	31	70	C (all)	50.0	8500	Pave	Pave	Reg	Lvl	AllPub	...
1167	617	60	RL	NaN	7861	Pave	NaN	IR1	Lvl	AllPub	...

1168 rows × 81 columns

[4]:
```
pd.set_option("display.max_columns", None)
```

[5]:
```
pd.set_option("display.max_columns", 80)
```

[6]:
```
df_train.head()
```

[6]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl

5 rows × 81 columns

```
[7]: df_test
```

[7]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	Corner	Gtl
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
...	...	...	...	...	...	...	...	...	...	...	...	...
287	83	20	RL	78.0	10206	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
288	1048	20	RL	57.0	9245	Pave	NaN	IR2	Lvl	AllPub	Inside	Gtl
289	17	20	RL	NaN	11241	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
290	523	50	RM	50.0	5000	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl
291	1379	160	RM	21.0	1953	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl

292 rows × 80 columns

```
[8]: pd.set_option("display.max_columns", None)
```

```
[9]: pd.set_option("display.max_columns", 80)
```

```
[10]: df_test.head()
```

[10]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	337	20	RL	86.0	14157	Pave	NaN	IR1	HLS	AllPub	Corner	Gtl
1	1018	120	RL	NaN	5814	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
2	929	20	RL	NaN	11838	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
3	1148	70	RL	75.0	12000	Pave	NaN	Reg	Bnk	AllPub	Inside	Gtl
4	1227	60	RL	86.0	14598	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl

Know Your Data

```
## Will use this feature while converting into numerical format/Encoding

Neighborhood
OverallQual
OverallCond
YearBuilt
Foundation
Electrical
KitchenQual
GarageType
GarageFinish
Fence
```

Data Integration

[11]:
```
df = pd.concat([df_train, df_test])
print("Shape of Integrated Data/DF:", df.shape)
```

```
Shape of Integrated Data/DF: (1460, 81)
```

[12]:
```
df.head(5)
```

[12]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
0	127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
1	889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod
2	793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
3	110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
4	422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl

5 rows × 81 columns

[13]:
```
df.tail(5)
```

[13]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
287	83	20	RL	78.0	10206	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
288	1048	20	RL	57.0	9245	Pave	NaN	IR2	Lvl	AllPub	Inside	Gtl
289	17	20	RL	NaN	11241	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
290	523	50	RM	50.0	5000	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl
291	1379	160	RM	21.0	1953	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl

5 rows × 81 columns

Get the Brief Information of Dataset

```
[14]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 0 to 291
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1168 non-null   float64
dtypes: float64(4), int64(34), object(43)
memory usage: 935.3+ KB
```

```
# Most null value Feature

Alley
FireplaceQu
PoolQC
Fence
MiscFeature
```

[15]:
```
int_features = df.select_dtypes(include=["int64"]).columns
print("Total Number of Integer Features:", int_features.shape[0])

print("Integer Feature Names:", int_features.tolist)
```

```
Total Number of Integer Features: 34
Integer Feature Names: <bound method IndexOpsMixin.tolist of Index(['Id', 'MSSubClass', 'LotArea', 'OverallQual', 'OverallCond',
       'YearBuilt', 'YearRemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
       'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageCars',
       'GarageArea', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
       'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold'],
      dtype='object')>
```

[16]:
```
float_features = df.select_dtypes(include=["float64"]).columns
print("Total Number of Float Features:", float_features.shape[0])

print("Float Feature Names:", float_features .tolist)
```

```
Total Number of Float Features: 4
Float Feature Names: <bound method IndexOpsMixin.tolist of Index(['LotFrontage', 'MasVnrArea', 'GarageYrBlt', 'SalePrice'], dtype='object')>
```

[17]:
```
categorical_features = df.select_dtypes(include=["object"]).columns
print("Total Number of Categorical Features:", categorical_features.shape[0])

print("Categorical Feature Names:", categorical_features.tolist)
```

```
Total Number of Categorical Features: 43
Categorical Feature Names: <bound method IndexOpsMixin.tolist of 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')>
```

Get the Statistical Information of Numerical Features ¶

```
[18]: df.describe()
```

[18]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000

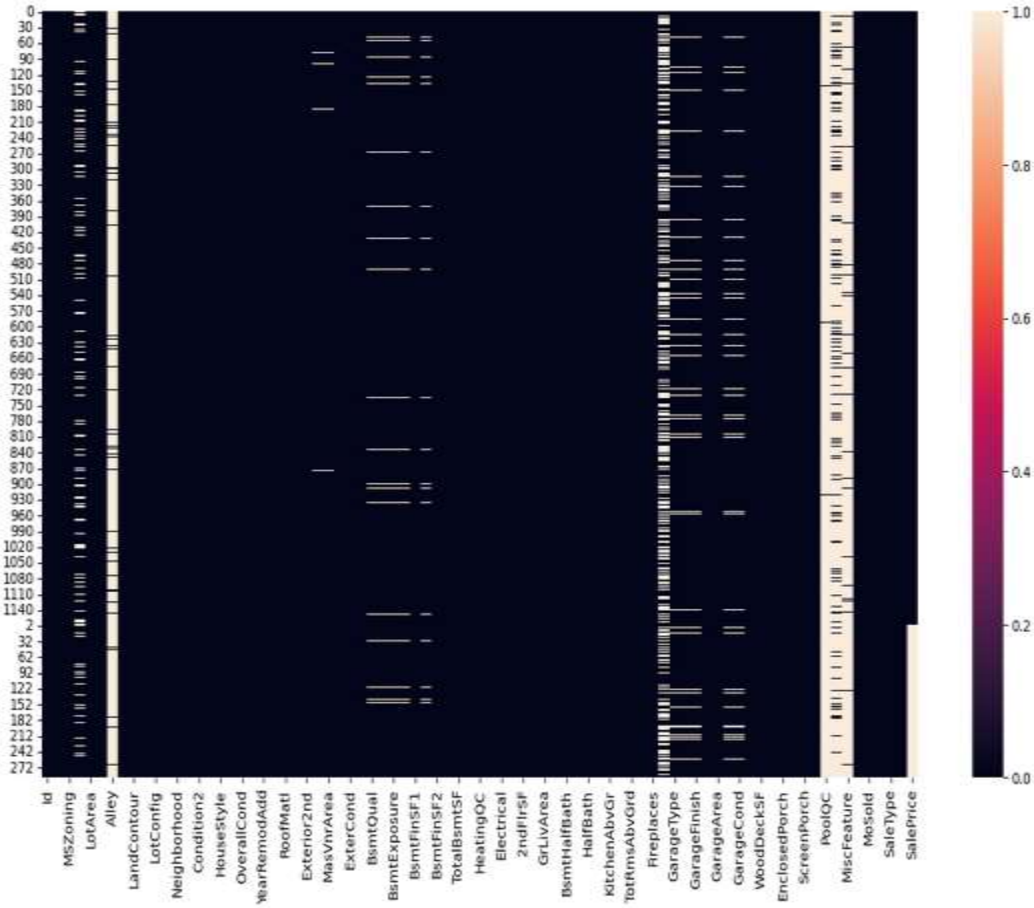
```
[19]: df.describe().shape
```

```
[19]: (8, 38)
```

Handling Missing Value

Visualise Null/Missing Value

```
[20]: plt.figure(figsize=(16,9))
      sns.heatmap(df.isnull())
      plt.savefig("House_img")
```



Get the null value percentage for every feature

[21]:
```
#set index as Id columns
df.set_index("Id")
```

[21]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope
127	120	RL	NaN	4928	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
889	20	RL	95.0	15865	Pave	NaN	IR1	Lvl	AllPub	Inside	Mod
793	60	RL	92.0	9920	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
110	20	RL	105.0	11751	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl
422	20	RL	NaN	16635	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl
...	...	...	...	...	...	...	...	...	...	...	...
83	20	RL	78.0	10206	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl
1048	20	RL	57.0	9245	Pave	NaN	IR2	Lvl	AllPub	Inside	Gtl
17	20	RL	NaN	11241	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl
523	50	RM	50.0	5000	Pave	NaN	Reg	Lvl	AllPub	Corner	Gtl
1379	160	RM	21.0	1953	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl

1460 rows × 80 columns

```
[22]: null_count = df.isnull().sum()
      null_count
```

```
[22]: Id                0
      MSSubClass        0
      MSZoning          0
      LotFrontage     259
      LotArea           0
                     ...
      MoSold            0
      YrSold            0
      SaleType          0
      SaleCondition     0
      SalePrice       292
      Length: 81, dtype: int64
```

```
[23]: null_percent = df.isnull().sum()/df.shape[0]*100
      null_percent
```

```
[23]: Id                0.000000
      MSSubClass        0.000000
      MSZoning          0.000000
      LotFrontage      17.739726
      LotArea           0.000000
                          ...
      MoSold            0.000000
      YrSold            0.000000
      SaleType          0.000000
      SaleCondition     0.000000
      SalePrice        20.000000
      Length: 81, dtype: float64
```

Drop Columns/Features

As per observation , we will not drop any feature from dataset

```
[24]: # """As per domain knowledge we will not drop those features, insead None value we will add constant value NA"""
      miss_value_50_perc = null_percent[null_percent>50]
      miss_value_50_perc
```

```
[24]: Alley          93.767123
      PoolQC         99.520548
      Fence          80.753425
      MiscFeature    96.301370
      dtype: float64
```

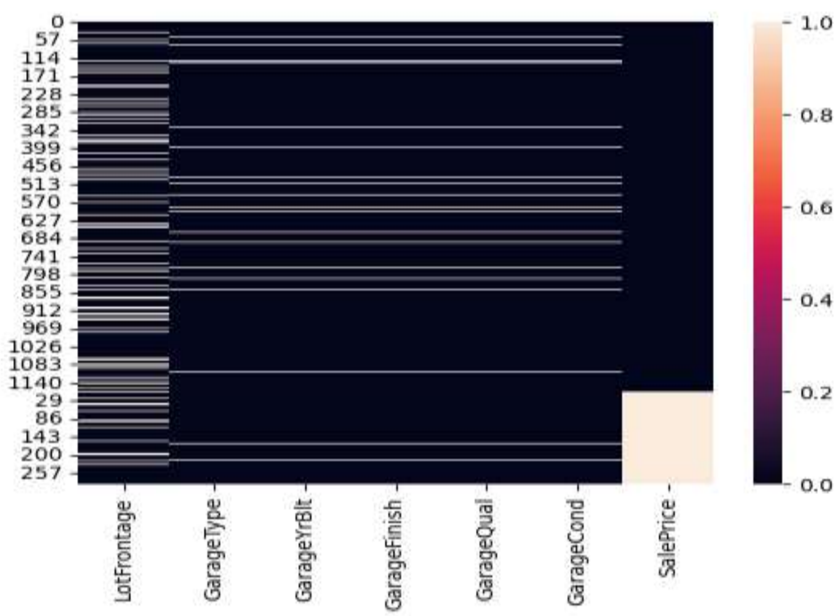
```
[25]: df["Alley"].value_counts()
```

```
[25]: Grvl    50
      Pave    41
      Name: Alley, dtype: int64
```

```
[26]: df["PoolQC"].value_counts()
```

```
[26]: Gd    3
      Ex    2
      Fa    2
      Name: PoolQC, dtype: int64
```

```
[27]: df["Fence"].value_counts()

[27]: MnPrv    157
      GdPrv     59
      GdWo      54
      MnWw      11
      Name: Fence, dtype: int64

[28]: df["MiscFeature"].value_counts()

[28]: Shed    49
      Gar2     2
      Othr     2
      TenC     1
      Name: MiscFeature, dtype: int64

[29]: # """As per domain knowledge we will not drop FireplaceQu features, insead None value we will add constant value 'NA' """
      miss_value_20_50_perc = null_percent[(null_percent>20) & (null_percent<51)]
      miss_value_20_50_perc

[29]: FireplaceQu    47.260274
      dtype: float64

[30]: # """As per domain knowledge we will not drop FireplaceQu features, insead None value we will add constant value 'NA' """
      miss_value_5_20_perc = null_percent[(null_percent>5) & (null_percent<21)]
      miss_value_5_20_perc

[30]: LotFrontage     17.739726
      GarageType       5.547945
      GarageYrBlt      5.547945
      GarageFinish     5.547945
      GarageQual       5.547945
      GarageCond       5.547945
      SalePrice       20.000000
      dtype: float64

[31]: df["LotFrontage"].value_counts().head()

[31]: 60.0    143
      70.0     70
      80.0     69
      50.0     57
      75.0     53
      Name: LotFrontage, dtype: int64

[32]: sns.heatmap(df[miss_value_5_20_perc.keys()].isnull())

[32]: <Axes: >
```



Missing Value Imputation

[33]:
```
missing_value_feat = null_percent[null_percent>0]
print("Total Missing Value Features =", len(missing_value_feat))
```

```
Total Missing Value Features = 20
```

[34]:
```
missing_value_feat
```

[34]:
```
LotFrontage     17.739726
Alley           93.767123
MasVnrType       0.547945
MasVnrArea       0.547945
BsmtQual         2.534247
BsmtCond         2.534247
BsmtExposure     2.602740
BsmtFinType1     2.534247
BsmtFinType2     2.602740
Electrical       0.068493
FireplaceQu     47.260274
GarageType       5.547945
GarageYrBlt      5.547945
GarageFinish     5.547945
GarageQual       5.547945
GarageCond       5.547945
PoolQC          99.520548
Fence           80.753425
MiscFeature     96.301370
SalePrice       20.000000
dtype: float64
```

[35]:
```
categorical_na_feat = missing_value_feat[missing_value_feat.keys().isin(categorical_features)]
print("Total Number of Categorical Missing Features=", len(categorical_na_feat))
categorical_na_feat
```

```
Total Number of Categorical Missing Features= 16
```

[35]:
```
Alley           93.767123
MasVnrType       0.547945
BsmtQual         2.534247
BsmtCond         2.534247
BsmtExposure     2.602740
BsmtFinType1     2.534247
BsmtFinType2     2.602740
Electrical       0.068493
FireplaceQu     47.260274
GarageType       5.547945
GarageFinish     5.547945
GarageQual       5.547945
GarageCond       5.547945
PoolQC          99.520548
Fence           80.753425
MiscFeature     96.301370
dtype: float64
```

```
[36]: int_na_feat = missing_value_feat[missing_value_feat.keys().isin(int_features)]
      print("Total Number of Integer Missing Features=", len(int_na_feat))
      int_na_feat
```

```
Total Number of Integer Missing Features= 0
```

```
[36]: Series([], dtype: float64)
```

```
[37]: float_na_feat = missing_value_feat[missing_value_feat.keys().isin(float_features)]
      print("Total Number of Floating Missing Features=", len(categorical_na_feat))
      float_na_feat
```

```
Total Number of Floating Missing Features= 16
```

```
[37]: LotFrontage    17.739726
      MasVnrArea      0.547945
      GarageYrBlt     5.547945
      SalePrice      20.000000
      dtype: float64
```

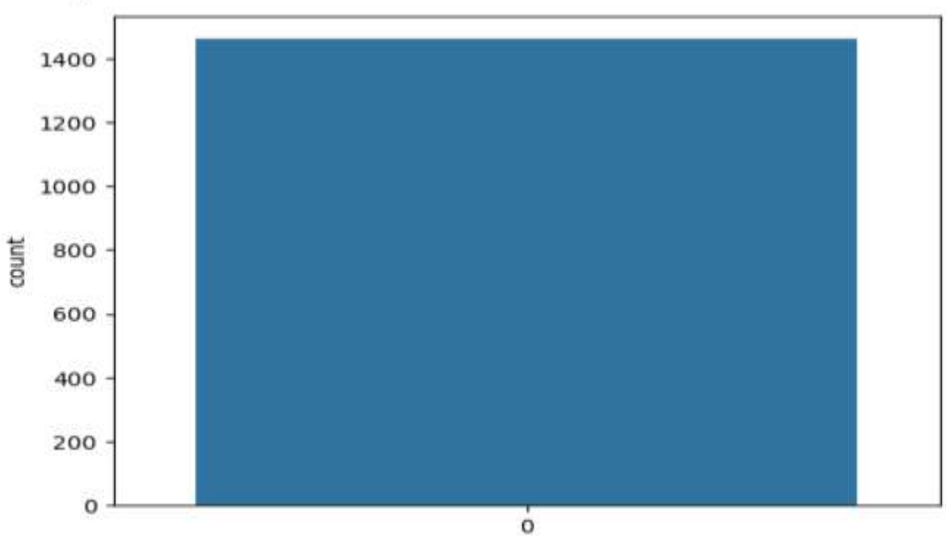
Handling LotFrontage = 17.739726

```
[38]: df["LotFrontage"].value_counts()
```

```
[38]: 60.0     143
      70.0      70
      80.0      69
      50.0      57
      75.0      53
              ...
      46.0       1
      141.0      1
      152.0      1
      160.0      1
      150.0      1
      Name: LotFrontage, Length: 110, dtype: int64
```

```
[39]: sns.countplot(df["LotFrontage"].index)
```

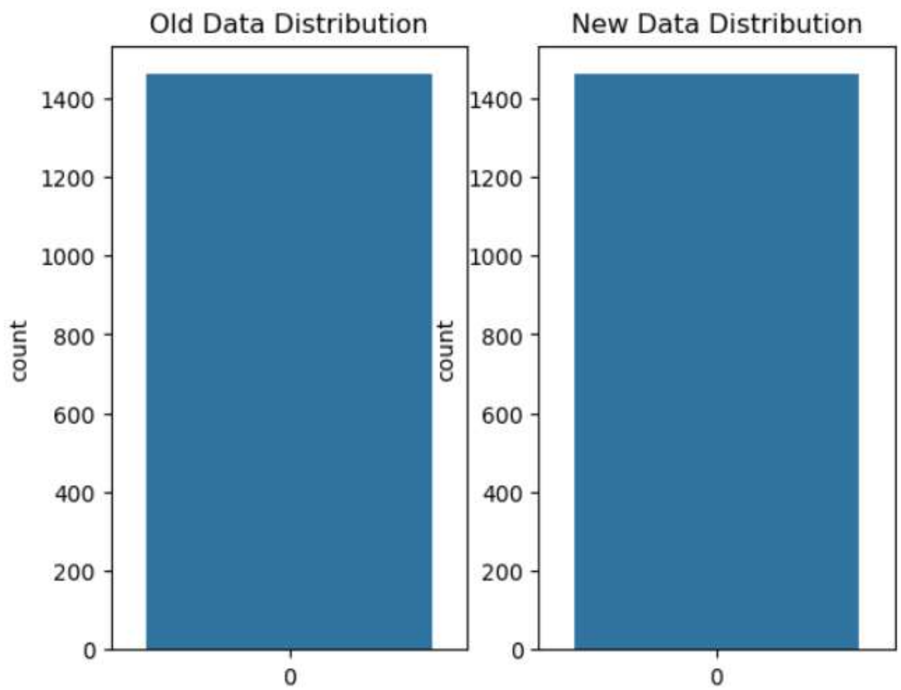
```
[39]: <Axes: ylabel='count'>
```




```
[40]: ### Backup of Original data
      df_mvi  = df.copy()
      df_mvi.shape
```

```
[40]: (1460, 81)
```

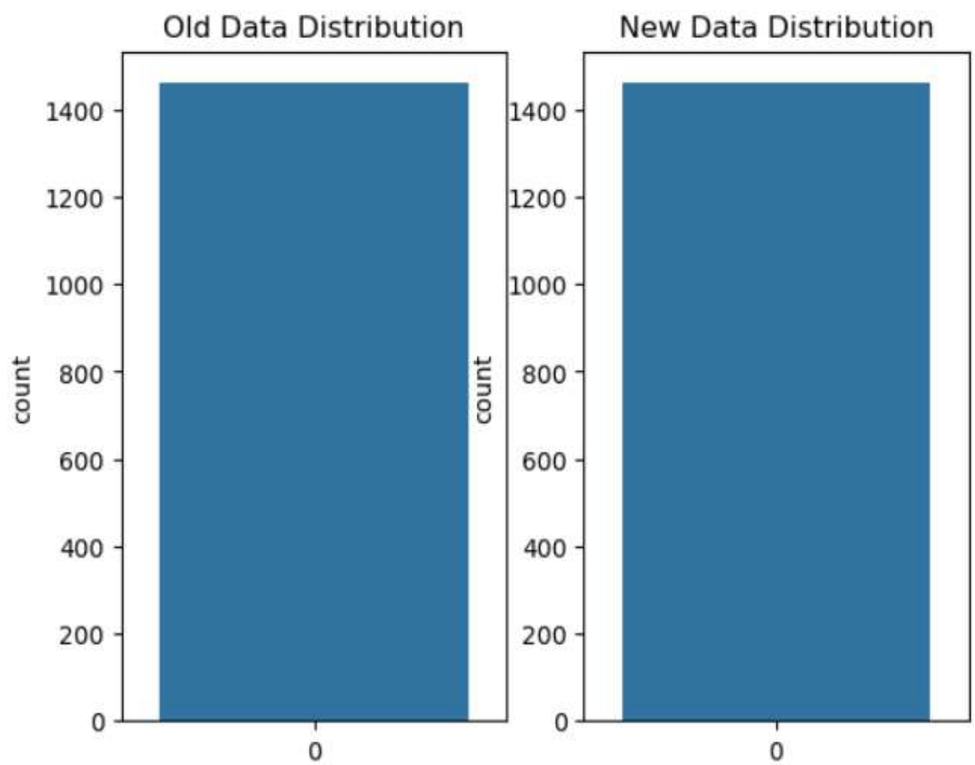
```
[41]: LotFrontage_mode = df["LotFrontage"].mode()[0]
      df_mvi["LotFrontage"].replace(np.nan, LotFrontage_mode, inplace=True)
      df_mvi["LotFrontage"].isnull().sum()

[41]: 0

[42]: def oldNewCountPlot(df, df_new, feature):
          plt.subplot(121)
          sns.countplot(df["LotFrontage"].index)
          plt.title("Old Data Distribution")
          plt.subplot(122)
          sns.countplot(df_new["LotFrontage"].index)
          plt.title("New Data Distribution")

[43]: oldNewCountPlot(df, df_mvi, "LotFrontage")
```



Handling Alley = 93.767123

```
[44]: df_mvi["Alley"].value_counts()

[44]: Grvl    50
      Pave    41
      Name: Alley, dtype: int64

[45]: Alley_cont = "NA"
      df_mvi["Alley"].replace(np.nan, Alley_cont, inplace = True)
      df_mvi["Alley"].isnull().sum()

[45]: 0

[46]: def oldNewCountPlot(df, df_new, feature):
          plt.subplot(121)
          sns.countplot(df[feature].index)
          plt.title("Old Data Distribution")
          plt.subplot(122)
          sns.countplot(df_new[feature].index)
          plt.title("New Data Distribution")
```

[47]:
```
oldNewCountPlot(df, df_mvi, "Alley")
```

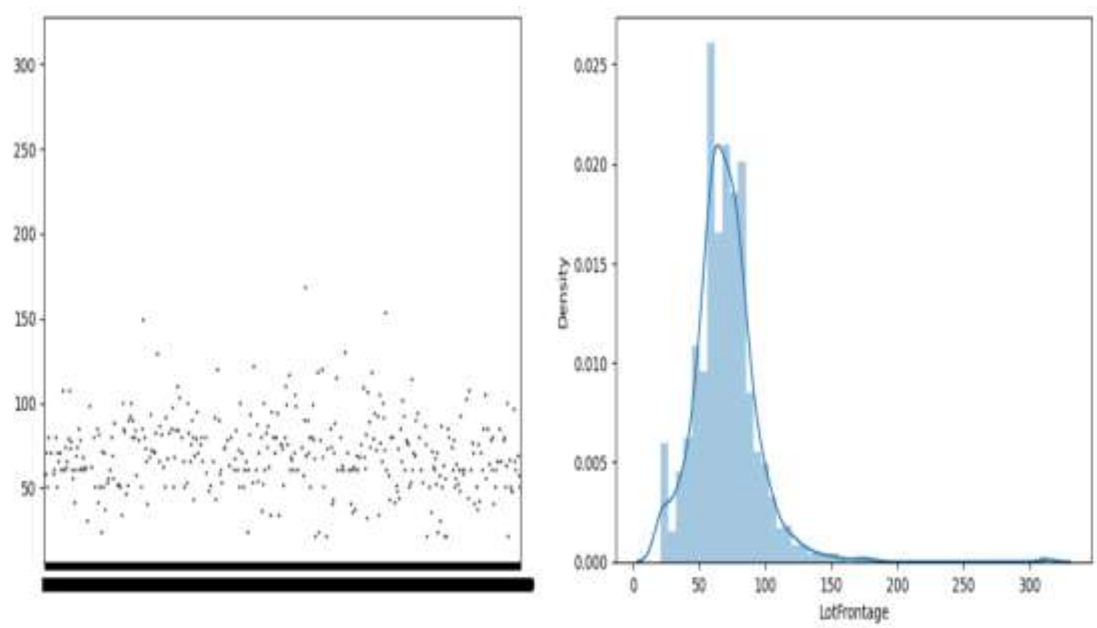


Handling LotFrontage = 17.739726

[48]:
```
def boxHistPlot(df, figsize=(16, 5)):
    plt.figure(figsize=figsize)
    plt.subplot(121)
    sns.boxplot(df)
    plt.subplot(122)
    sns.distplot(df)
```

[49]:
```
boxHistPlot(df["LotFrontage"])
```

```
C:\Users\gk521\AppData\Local\Temp\ipykernel_13740\180669257.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df)
```




```
[50]: lotfrontage_mean = df["LotFrontage"].mean()
      lotfrontage_mean
```

```
[50]: 70.04995836802665
```

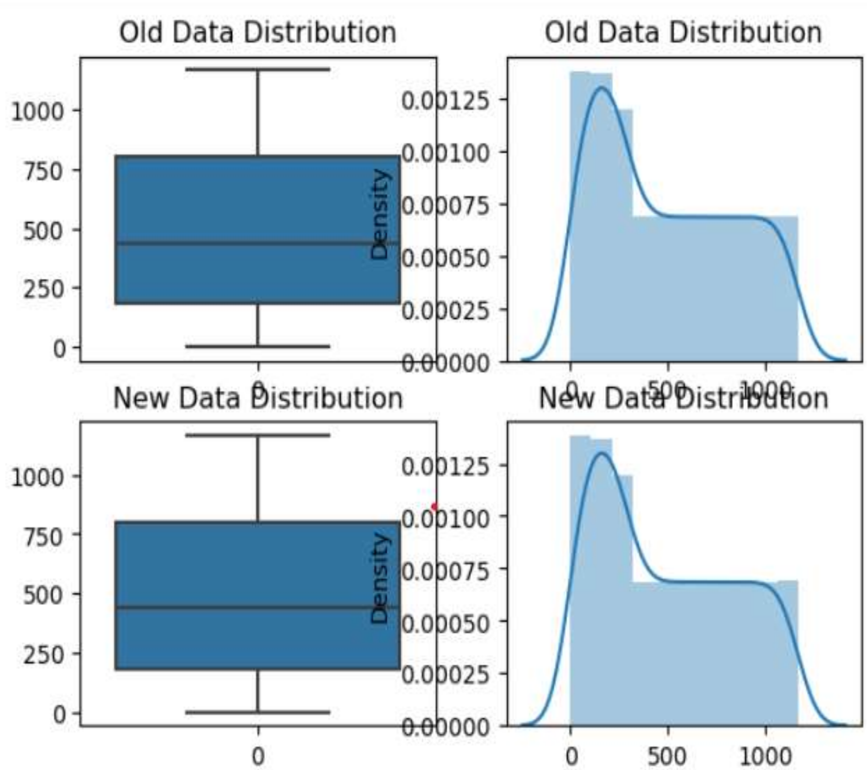
```
[51]: lotfrontage_mean = df["LotFrontage"].mean()
      df_mvi["LotFrontage"].replace(np.nan, lotfrontage_mean, inplace = True)
      df_mvi["LotFrontage"].isnull().sum()
```

```
[51]: 0
```

```
[52]: def oldNewBoxHistPlot(df, df_new, feature):
          plt.subplot(221)
          sns.boxplot(df[feature].index)
          plt.title("Old Data Distribution")
          plt.subplot(222)
          sns.distplot(df[feature].index)
          plt.title("Old Data Distribution")

          plt.subplot(223)
          sns.boxplot(df_new[feature].index)
          plt.title("New Data Distribution")
          plt.subplot(224)
          sns.distplot(df_new[feature].index)
          plt.title("New Data Distribution")
```

```
[53]: oldNewBoxHistPlot(df, df_mvi, "LotFrontage")
```

```
C:\Users\gk521\AppData\Local\Temp\ipykernel_13740\414266027.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[feature].index)
C:\Users\gk521\AppData\Local\Temp\ipykernel_13740\414266027.py:13: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df_new[feature].index)
```



Handling MasVnrType = 0.547945

[54]:
```
df["MasVnrType"].value_counts()
```

[54]:
```
None       864
BrkFace    445
Stone      128
BrkCmn      15
Name: MasVnrType, dtype: int64
```

[55]:
```
MasVnrType_mode = df["MasVnrType"].mode()[0]
df_mvi["MasVnrType"].replace(np.nan, MasVnrType_mode, inplace=True)
df_mvi["MasVnrType"].isnull().sum()
```

[55]:
```
0
```

Handling Exterior1st = 0.034258 Exterior2nd = 0.034258

[56]:
```
df["Exterior1st"].value_counts()
```

[56]:
```
VinylSd    515
HdBoard    222
MetalSd    220
Wd Sdng    206
Plywood    108
CemntBd     61
BrkFace     50
WdShing     26
Stucco      25
AsbShng     20
Stone        2
BrkComm      2
AsphShn      1
ImStucc      1
CBlock       1
Name: Exterior1st, dtype: int64
```

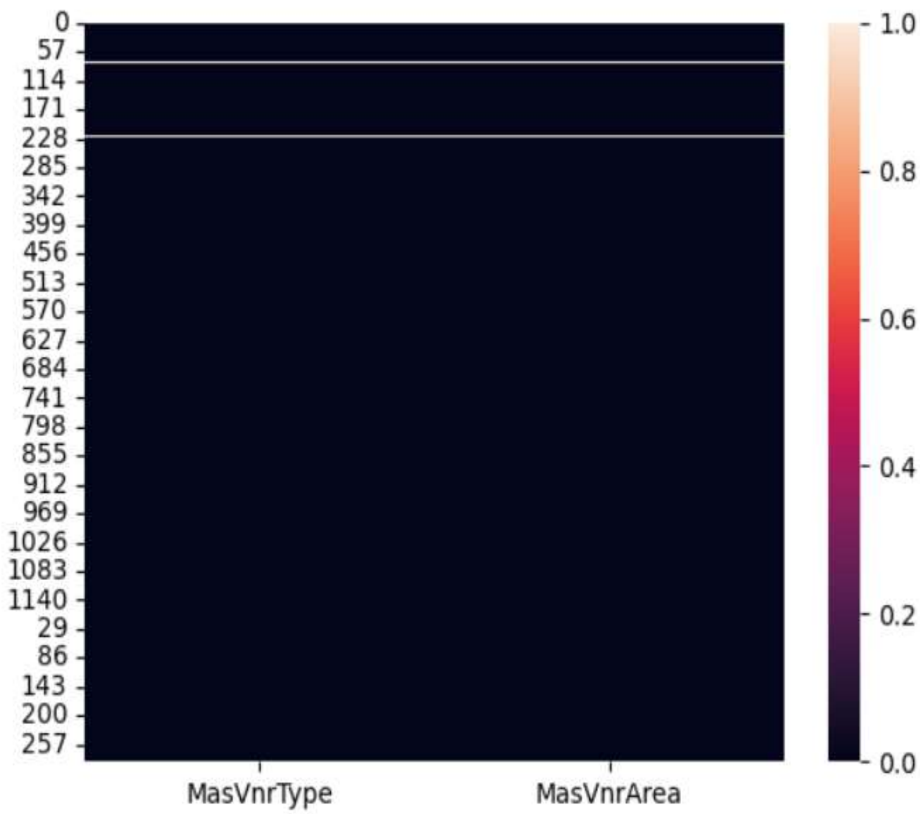
[57]:
```
df["Exterior2nd"].value_counts()
```

[57]:
```
VinylSd    504
MetalSd    214
HdBoard    207
Wd Sdng    197
Plywood    142
CmentBd     60
Wd Shng     38
Stucco      26
BrkFace     25
AsbShng     20
ImStucc     10
Brk Cmn      7
Stone        5
AsphShn      3
Other        1
CBlock       1
Name: Exterior2nd, dtype: int64
```

```
[58]: Exterior1st_mode = df["Exterior1st"].mode()[0]
      Exterior2nd_mode = df["Exterior2nd"].mode()[0]
      df_mvi["Exterior1st"].replace(np.nan, Exterior1st_mode, inplace=True)
      df_mvi["Exterior2nd"].replace(np.nan, Exterior2nd_mode, inplace=True)
      print("E1st is null:", df_mvi["Exterior1st"].isnull().sum())
      print("E2nd is null:", df_mvi["Exterior2nd"].isnull().sum())

      E1st is null: 0
      E2nd is null: 0

[59]: df["Exterior1st"].mode()[0]

[59]: 'VinylSd'
```

Handling MasVnrType = 0.547945 MasVnrArea = 0.547945

```
[60]: sns.heatmap(df[["MasVnrType", "MasVnrArea"]].isnull())

[60]: <Axes: >
```



```
[61]: df[df[["MasVnrType", "MasVnrArea"]].isnull().any(axis=1)]
```

[61]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	LotConfig
68	530	20	RL	NaN	32668	Pave	NaN	IR1	Lvl	AllPub	CulDSac
78	1244	20	RL	107.0	13891	Pave	NaN	Reg	Lvl	AllPub	Inside
99	651	60	FV	65.0	8125	Pave	NaN	Reg	Lvl	AllPub	Inside
185	974	20	FV	95.0	11639	Pave	NaN	Reg	Lvl	AllPub	Corner
224	978	120	FV	35.0	4274	Pave	Pave	IR1	Lvl	AllPub	Inside
367	1279	60	RL	75.0	9473	Pave	NaN	Reg	Lvl	AllPub	Inside
874	235	60	RL	NaN	7851	Pave	NaN	Reg	Lvl	AllPub	Inside
31	937	20	RL	67.0	10083	Pave	NaN	Reg	Lvl	AllPub	Inside

8 rows × 81 columns

```
[62]: df["MasVnrType"].value_counts()
```

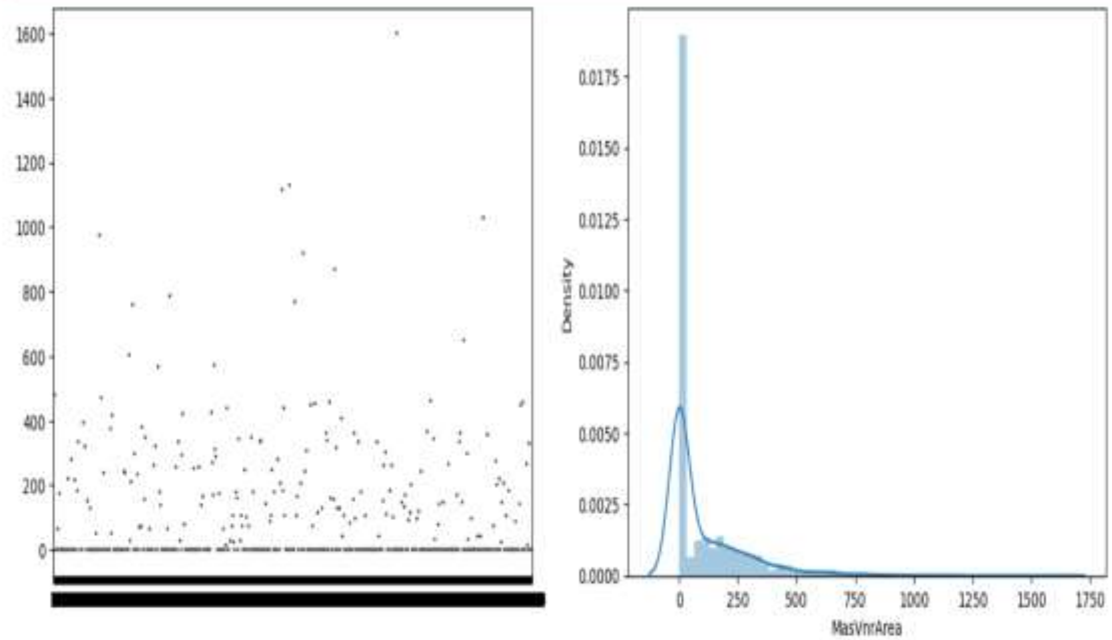
```
[62]: None       864
BrkFace    445
Stone      128
BrkCmn      15
Name: MasVnrType, dtype: int64
```

```
[63]: MasVnrType_mode = df["MasVnrType"].mode()[0]
df_mvi["MasVnrType"].replace(np.nan, MasVnrType_mode, inplace=True)
df_mvi["MasVnrType"].isnull().sum()
```

```
[63]: 0
```

```
[64]: boxHistPlot(df["MasVnrArea"])
```

```
C:\Users\gk521\AppData\Local\Temp\ipykernel_13740\180669257.py:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df)
```



[65]:
```
MasVnrArea_cont = 0
df_mvi["MasVnrArea"].replace(np.nan, MasVnrArea_cont, inplace=True)
df_mvi["MasVnrArea"].isnull().sum()
```

[65]: 0

Handling Bsmt Features

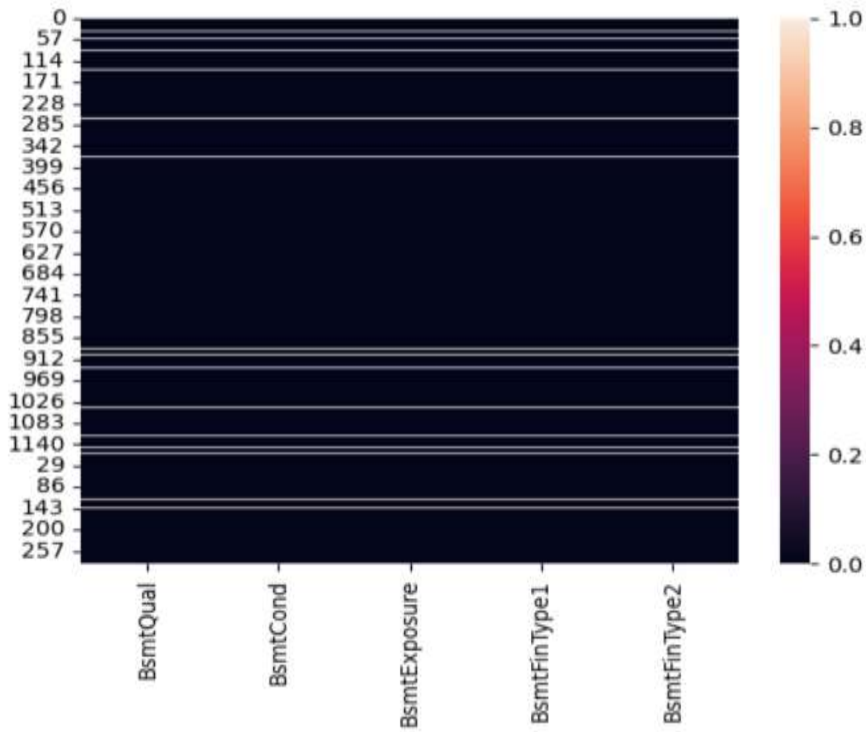
[66]:
```
"""cat_bsmt_feat =
BsmtQual        2.534247
BsmtCond        2.534247
BsmtExposure    2.602740
BsmtFinType1    2.534247
BsmtFinType2    2.602740
"""
```

[66]: 'cat_bsmt_feat = \nBsmtQual   2.534247\nBsmtCond   2.534247\nBsmtExposure   2.602740\nBsmtFinType1   2.534247\nBsmtFinType2   2.602740\n'

[67]:
```
cat_bsmt_feat = ["BsmtQual",
"BsmtCond",
"BsmtExposure",
"BsmtFinType1",
"BsmtFinType2"]
```

[68]:
```
sns.heatmap(df[cat_bsmt_feat].isnull())
```

[68]: <Axes: >



[69]:
```
for feat in cat_bsmt_feat:
    print(f"Value count of {feat}: {df[feat].value_counts()}")
```

```
Value count of BsmtQual: TA    649
Gd    618
Ex    121
Fa     35
Name: BsmtQual, dtype: int64
Value count of BsmtCond: TA    1311
Gd      65
Fa      45
Po       2
Name: BsmtCond, dtype: int64
Value count of BsmtExposure: No    953
Av    221
Gd    134
Mn    114
Name: BsmtExposure, dtype: int64
Value count of BsmtFinType1: Unf    430
GLQ    418
ALQ    220
BLQ    148
Rec    133
LwQ     74
Name: BsmtFinType1, dtype: int64
Value count of BsmtFinType2: Unf    1256
Rec      54
LwQ      46
BLQ      33
ALQ      19
GLQ      14
Name: BsmtFinType2, dtype: int64
```

```
[70]: bsmt_cont = "NA"
      for feat in cat_bsmt_feat:
          df_mvi[feat].replace(np.nan, bsmt_cont, inplace = True)
```

```
[71]: df_mvi[cat_bsmt_feat].isnull().sum()
```

```
[71]: BsmtQual        0
      BsmtCond        0
      BsmtExposure    0
      BsmtFinType1    0
      BsmtFinType2    0
      dtype: int64
```

Handling Electrical = 0.068493

```
[72]: df["Electrical"].value_counts()
```

```
[72]: SBrkr    1334
      FuseA      94
      FuseF      27
      FuseP       3
      Mix         1
      Name: Electrical, dtype: int64
```

```
[73]: df["Electrical"].isnull()
```


```
[73]: df["Electrical"].isnull()
```


```
[73]: 0      False
      1      False
      2      False
      3      False
      4      False
             ...
      287    False
      288    False
      289    False
      290    False
      291    False
      Name: Electrical, Length: 1460, dtype: bool
```

```
[74]: Electrical_mode = df["Electrical"].mode()[0]
      df_mvi["Electrical"].replace(np.nan, Electrical_mode, inplace=True)
      df_mvi["Electrical"].isnull().sum()
```

```
[74]: 0
```

Handling remaining cat features

FireplaceQu 47.260274 mode

```
[75]: df["FireplaceQu"].value_counts()
```

```
[75]: Gd    380
TA    313
Fa     33
Ex     24
Po     20
Name: FireplaceQu, dtype: int64
```

```
[76]: FireplaceQu_mode = df["FireplaceQu"].mode()[0]
df_mvi["FireplaceQu"].replace(np.nan, FireplaceQu_mode, inplace=True)
df_mvi["FireplaceQu"].isnull().sum()
```

```
[76]: 0
```

```
[77]: other_cat_feat = ["PoolQC",
"Fence",
"MiscFeature"]
for feat in other_cat_feat:
    print(f"Value count of {feat}: {df[feat].value_counts()}")
```

```
Value count of PoolQC: Gd    3
Ex    2
Fa    2
Name: PoolQC, dtype: int64
Value count of Fence: MnPrv    157
GdPrv     59
GdWo      54
MnWw      11
Name: Fence, dtype: int64
Value count of MiscFeature: Shed    49
Gar2     2
Othr     2
TenC     1
Name: MiscFeature, dtype: int64
```

[78]:
```
PoolQC_cont = "NA"
df_mvi["PoolQC"].replace(np.nan, PoolQC_cont, inplace=True)
df_mvi["PoolQC"].isnull().sum()
```

[78]: 0

[79]:
```
Fence_cont = "NA"
df_mvi["Fence"].replace(np.nan, Fence_cont, inplace=True)
df_mvi["Fence"].isnull().sum()
```

[79]: 0

[80]:
```
MiscFeature_cont = "NA"
df_mvi["MiscFeature"].replace(np.nan, MiscFeature_cont, inplace=True)
df_mvi["MiscFeature"].isnull().sum()
```

[80]: 0

Handling Remaining Garage cat features

```
GarageType      5.547945 NA
GarageFinish    5.547945 NA
GarageQual      5.547945 NA
GarageCond      5.547945 NA
```

[81]:
```
cat_Garage_feat = ["GarageType", "GarageFinish", "GarageQual", "GarageCond"]
df_garage = df[cat_Garage_feat]
df_garage[df_garage.isnull().any(axis=1)]
```

[81]:
```
cat_Garage_feat = ["GarageType", "GarageFinish", "GarageQual", "GarageCond"]
df_garage = df[cat_Garage_feat]
df_garage[df_garage.isnull().any(axis=1)]
```


[81]:

	GarageType	GarageFinish	GarageQual	GarageCond
48	NaN	NaN	NaN	NaN
72	NaN	NaN	NaN	NaN
74	NaN	NaN	NaN	NaN
105	NaN	NaN	NaN	NaN
116	NaN	NaN	NaN	NaN
...	...	...	...	...
214	NaN	NaN	NaN	NaN
215	NaN	NaN	NaN	NaN
217	NaN	NaN	NaN	NaN
218	NaN	NaN	NaN	NaN
256	NaN	NaN	NaN	NaN

81 rows × 4 columns

[82]:
```python
Garage_cont = "NA"
for feat in cat_Garage_feat:
    df_mvi[feat].replace(np.nan, Garage_cont, inplace = True)

df_mvi[cat_Garage_feat].isnull().sum()
```

[82]:
```
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
dtype: int64
```

[83]:
```python
df_mvi.isnull().any(axis=1).sum()
```

[83]:
```
356
```

Feature Transformation

Convert Numerical Feature to Categorical Feature

[84]:
```python
for_num_conv = ["MSSubClass", "YearBuilt", "YearRemodAdd", "GarageYrBlt", "MoSold", "YrSold"]
for feat in for_num_conv:
    print(f"{feat}: data type = {df_mvi[feat].dtype}")
```

```
MSSubClass: data type = int64
YearBuilt: data type = int64
YearRemodAdd: data type = int64
GarageYrBlt: data type = float64
MoSold: data type = int64
YrSold: data type = int64
```

[85]:
```python
df_mvi[for_num_conv].head()
```

[85]:

	MSSubClass	YearBuilt	YearRemodAdd	GarageYrBlt	MoSold	YrSold
0	120	1976	1976	1977.0	2	2007
1	20	1970	1970	1970.0	10	2007
2	60	1996	1997	1997.0	6	2007
3	20	1977	1977	1977.0	1	2010
4	20	1977	2000	1977.0	6	2009

[86]:
```python
df_mvi["MoSold"].unique()
```

[86]:
```
array([ 2, 10,  6,  1, 11,  5,  4,  7,  8,  3,  9, 12], dtype=int64)
```

[87]:
```python
calendar.month_abbr[12]
```

[87]:
```
'Dec'
```

```
[88]: # df_mvi["MoSold"] = df_mvi["MoSold"].apply(lambda x : calendar.month_abbr[x])
      # df_mvi["MoSold"].unique()
```

```
[89]: for feat in for_num_conv:
          df_mvi[feat] = df_mvi[feat].astype(str)
```

```
[90]: for feat in for_num_conv:
          print(f"{feat}: data type = {df_mvi[feat].dtype}")

      MSSubClass: data type = object
      YearBuilt: data type = object
      YearRemodAdd: data type = object
      GarageYrBlt: data type = object
      MoSold: data type = object
      YrSold: data type = object
```

Convert Categorical into Feature Numerical Feature

Ordinal Encoding

```
[91]: ordinal_end_var = [
      "ExterQual",
      "ExterCond",
      "BsmtQual",
      "BsmtExposure",
      "BsmtFinType1",
      "BsmtFinSF1",
      "BsmtFinType2",
      "HeatingQC",
      "KitchenQual",
      "FireplaceQu",
      "GarageQual",
      "GarageCond",
      "PoolQC",
      "Functional",
      "GarageFinish",
      "PavedDrive",
      "Utilities"
      ]

      print("Total Number of Features to Convert Ordinal Numerical Format:", len(ordinal_end_var))

      Total Number of Features to Convert Ordinal Numerical Format: 17
```

```
[92]: df_mvi["ExterQual"].unique()

[92]: array(['TA', 'Gd', 'Ex', 'Fa'], dtype=object)
```

```
[93]: df_mvi["ExterQual"].value_counts()

[93]: TA    906
      Gd    488
      Ex     52
      Fa     14
      Name: ExterQual, dtype: int64

[94]: df_mvi["ExterQual"] = df_mvi["ExterQual"].astype(CategoricalDtype(categories=["Po", "Fa", "TA", "Gd", "Ex"], ordered=True)).cat.codes

[95]: df_mvi["ExterQual"].value_counts()

[95]: 2    906
      3    488
      4     52
      1     14
      Name: ExterQual, dtype: int64

[96]: df_mvi["BsmtExposure"].unique()

[96]: array(['No', 'Gd', 'Av', 'Mn', 'NA'], dtype=object)

[97]: df_mvi["BsmtExposure"].value_counts()

[97]: No    953
      Av    221
      Gd    134
      Mn    114
      NA     38
      Name: BsmtExposure, dtype: int64

[98]: df_mvi["BsmtExposure"] = df_mvi["BsmtExposure"].astype(CategoricalDtype(categories=["No", "Av", "Gd", "Mn", "NA"], ordered=True)).cat.codes

[99]: df_mvi["BsmtExposure"].value_counts()

[99]: 0    953
      1    221
      2    134
      3    114
      4     38
      Name: BsmtExposure, dtype: int64

[100]: df_mvi["ExterCond"] = df_mvi["ExterCond"].astype(CategoricalDtype(categories=["TA", "Gd", "Fa", "Po", "Ex"], ordered=True)).cat.codes

df_mvi["BsmtQual"] = df_mvi["BsmtQual"].astype(CategoricalDtype(categories=["Gd", "TA", "Ex", "NA", "Fa"], ordered=True)).cat.codes

df_mvi["BsmtFinType1"] = df_mvi["BsmtFinType1"].astype(CategoricalDtype(categories=["ALQ", "GLQ", "BLQ", "Unf" "Rec", "LwQ", "NA"], ordered=True)).cat.codes

df_mvi["BsmtFinType2"] = df_mvi["BsmtFinType2"].astype(CategoricalDtype(categories=["Unf", "Rec", "BLQ", "GLQ", "NA", "ALQ", "LwQ"], ordered=True)).cat.codes

df_mvi["HeatingQC"] = df_mvi["HeatingQC"].astype(CategoricalDtype(categories=["TA", "Ex", "Gd", "Fa", "Po"], ordered=True)).cat.codes

df_mvi["KitchenQual"] = df_mvi["KitchenQual"].astype(CategoricalDtype(categories=["TA", "Gd", "Ex", "Fa"], ordered=True)).cat.codes

df_mvi["FireplaceQu"] = df_mvi["FireplaceQu"].astype(CategoricalDtype(categories=["TA", "Gd", "Fa", "Ex", "Po"], ordered=True)).cat.codes

df_mvi["GarageQual"] = df_mvi["GarageQual"].astype(CategoricalDtype(categories=["TA", "Fa", "NA", "Gd", "Ex", "Po"], ordered=True)).cat.codes

df_mvi["GarageCond"] = df_mvi["GarageCond"].astype(CategoricalDtype(categories=["TA", "Fa", "Gd", "NA", "Po", "Ex"], ordered=True)).cat.codes

df_mvi["PoolQC"] = df_mvi["PoolQC"].astype(CategoricalDtype(categories=["NA", "Ex", "Gd", "Fa"], ordered=True)).cat.codes

df_mvi["Functional"] = df_mvi["Functional"].astype(CategoricalDtype(categories=["Typ", "Mod", "Maj1", "Min1", "Min2", "Sev", "Maj2"], ordered=True)).cat.codes

df_mvi["GarageFinish"] = df_mvi["GarageFinish"].astype(CategoricalDtype(categories=["RFn", "Unf", "Fin", "NA"], ordered=True)).cat.codes

df_mvi["PavedDrive"] = df_mvi["PavedDrive"].astype(CategoricalDtype(categories=["Y", "N", "P"], ordered=True)).cat.codes

df_mvi["Utilities"] = df_mvi["Utilities"].astype(CategoricalDtype(categories=["AllPub", "NoSeWa"], ordered=True)).cat.codes
```

```
[101]:  df_mvi.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1460 entries, 0 to 291
Data columns (total 81 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Id              1460 non-null   int64
 1   MSSubClass      1460 non-null   object
 2   MSZoning        1460 non-null   object
 3   LotFrontage     1460 non-null   float64
 4   LotArea         1460 non-null   int64
 5   Street          1460 non-null   object
 6   Alley           1460 non-null   object
 7   LotShape        1460 non-null   object
 8   LandContour     1460 non-null   object
 9   Utilities       1460 non-null   int8
 10  LotConfig       1460 non-null   object
 11  LandSlope       1460 non-null   object
 12  Neighborhood    1460 non-null   object
 13  Condition1      1460 non-null   object
 14  Condition2      1460 non-null   object
 15  BldgType        1460 non-null   object
 16  HouseStyle      1460 non-null   object
 17  OverallQual     1460 non-null   int64
 18  OverallCond     1460 non-null   int64
 19  YearBuilt       1460 non-null   object
 20  YearRemodAdd    1460 non-null   object
 21  RoofStyle       1460 non-null   object
 22  RoofMatl        1460 non-null   object
 23  Exterior1st     1460 non-null   object
 24  Exterior2nd     1460 non-null   object
 25  MasVnrType      1460 non-null   object
 26  MasVnrArea      1460 non-null   float64
 27  ExterQual       1460 non-null   int8
 28  ExterCond       1460 non-null   int8
 29  Foundation      1460 non-null   object
 30  BsmtQual        1460 non-null   int8
 31  BsmtCond        1460 non-null   object
 32  BsmtExposure    1460 non-null   int8
 33  BsmtFinType1    1460 non-null   int8
 34  BsmtFinSF1      1460 non-null   int64
 35  BsmtFinType2    1460 non-null   int8
 36  BsmtFinSF2      1460 non-null   int64
```

```
 41  CentralAir      1460 non-null   object
 42  Electrical      1460 non-null   object
 43  1stFlrSF        1460 non-null   int64
 44  2ndFlrSF        1460 non-null   int64
 45  LowQualFinSF    1460 non-null   int64
 46  GrLivArea       1460 non-null   int64
 47  BsmtFullBath    1460 non-null   int64
 48  BsmtHalfBath    1460 non-null   int64
 49  FullBath        1460 non-null   int64
 50  HalfBath        1460 non-null   int64
 51  BedroomAbvGr    1460 non-null   int64
 52  KitchenAbvGr    1460 non-null   int64
 53  KitchenQual     1460 non-null   int8
 54  TotRmsAbvGrd    1460 non-null   int64
 55  Functional      1460 non-null   int8
 56  Fireplaces      1460 non-null   int64
 57  FireplaceQu     1460 non-null   int8
 58  GarageType      1460 non-null   object
 59  GarageYrBlt     1460 non-null   object
 60  GarageFinish    1460 non-null   int8
 61  GarageCars      1460 non-null   int64
 62  GarageArea      1460 non-null   int64
 63  GarageQual      1460 non-null   int8
 64  GarageCond      1460 non-null   int8
 65  PavedDrive      1460 non-null   int8
 66  WoodDeckSF      1460 non-null   int64
 67  OpenPorchSF     1460 non-null   int64
 68  EnclosedPorch   1460 non-null   int64
 69  3SsnPorch       1460 non-null   int64
 70  ScreenPorch     1460 non-null   int64
 71  PoolArea        1460 non-null   int64
 72  PoolQC          1460 non-null   int8
 73  Fence           1460 non-null   object
 74  MiscFeature     1460 non-null   object
 75  MiscVal         1460 non-null   int64
 76  MoSold          1460 non-null   object
 77  YrSold          1460 non-null   object
 78  SaleType        1460 non-null   object
 79  SaleCondition   1460 non-null   object
 80  SalePrice       1168 non-null   float64
dtypes: float64(3), int64(29), int8(16), object(33)
memory usage: 807.9+ KB
```

One Hot Encoding for Nominal Categorical Data

```python
df_encod = df_mvi.copy()

object_features = df_encod.select_dtypes(include="object").columns.tolist()
print("Total Object Data Type Features:", len(object_features))

print("Features: \n", object_features)
```

```
Total Object Data Type Features: 33
Features:
 ['MSSubClass', 'MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType', 'Foundation', 'BsmtCond', 'Heating', 'CentralAir', 'Electrical', 'GarageType', 'GarageYrBlt', 'Fence', 'MiscFeature', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition']
```

```python
df_encod[object_features].head(2)
```

	MSSubClass	MSZoning	Street	Alley	LotShape	LandContour	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	BldgType	HouseStyle	YearBuilt	YearRemodAdd	RoofStyle	RoofMatl	Exterior1st	Exterior2nd	MasVnrType	Foundation	BsmtCond
0	120	RL	Pave	NA	IR1	Lvl	Inside	Gtl	NPkVill	Norm	Norm	TwnhsE	1Story	1976	1976	Gable	CompShg	Plywood	Plywood	None	CBlock	TA
1	20	RL	Pave	NA	IR1	Lvl	Inside	Mod	NAmes	Norm	Norm	1Fam	1Story	1970	1970	Flat	Tar&Grv	Wd Sdng	Wd Sdng	None	PConc	Gd

```python
df_encod = pd.get_dummies(df_encod, columns=object_features, prefix=object_features, drop_first=True)
```

```python
# print("Shape of DF before encoding:", df_encod.shape)
# df_encod = pd.get_dummies(df_encod,
#                           columns=object_features,
#                           prefix=object_features,
#                           drop_first=True)
# print("Shape of DF After encoding:", df_encod.shape)
```

```python
df_encod.head(2)
```

	Id	LotFrontage	LotArea	Utilities	OverallQual	OverallCond	MasVnrArea	ExterQual	ExterCond	BsmtQual	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	BsmtFinSF2	BsmtUnfSF	TotalBsmtSF	HeatingQC	1stFlrSF	2ndFlrSF	LowQualFinSF	GrLivA
0	127	60.0	4928	0	6	5	0.0	2	0	0	3	0	120	0	0	958	1078	0	958	0	0	1
1	889	95.0	15865	0	8	6	0.0	3	1	1	2	0	351	1	823	1043	2217	1	2217	0	0	2

2 rows × 504 columns

Split Data For Training & Testing

```python
[152]: df_encod.shape
```

```
[152]: (1460, 504)
```

```python
[153]: len_train = df_train.shape[0]
       len_train
```

```
[153]: 1168
```

```python
[154]: X_train = df_encod[:len_train].drop("SalePrice", axis=1)
       y_train = df_encod["SalePrice"][:len_train]
       X_test = df_encod[len_train:].drop("SalePrice", axis=1)

       print("Shape of X_train Data:", X_train.shape)
       print("Shape of y_train Data:", X_train.shape)
       print("Shape of X_test Data:", X_train.shape)
```

```
Shape of X_train Data: (1168, 503)
Shape of y_train Data: (1168, 503)
Shape of X_test Data: (1168, 503)
```

Feature Scaling ¶

[155]:
```
sc = StandardScaler()
sc.fit(X_train)
# Formula = z=(x-u)/s
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

[156]:
```
X_train[:3, :]
```

[156]:
```
array([[-1.43548658, -0.39318669, -0.62061571, ..., -0.12510865,
         0.48577653, -0.31919711],
       [ 0.39632483,  1.14010261,  0.60090318, ..., -0.12510865,
         0.48577653, -0.31919711],
       [ 0.16554544,  1.00867782, -0.06307504, ..., -0.12510865,
         0.48577653, -0.31919711]])
```

[157]:
```
X_test[:3, :]
```

[157]:
```
array([[-0.93065666,  0.74582822,  0.41014207, ..., -0.12510865,
         0.48577653, -0.31919711],
       [ 0.70643463, -0.39318669, -0.52166118, ..., -0.12510865,
        -2.05855973, -0.31919711],
       [ 0.4924829 , -0.39318669,  0.1511403 , ..., -0.12510865,
         0.48577653, -0.31919711]])
```

[158]:
```
sc.mean_
```

[158]:
```
array([7.24136130e+02, 6.89751712e+01, 1.04847491e+04, 0.00000000e+00,
       6.10445205e+00, 5.59589041e+00, 1.01696918e+02, 2.40410959e+00,
       1.54109589e-01, 7.79965753e-01, 6.84075342e-01, 4.31506849e-01,
       4.44726027e+02, 4.88869863e-01, 4.66472603e+01, 5.69721747e+02,
       1.06109503e+03, 9.30650685e-01, 1.16986045e+03, 3.48826199e+02,
       6.38013699e+00, 1.52506678e+03, 4.25513699e-01, 5.56506849e-02,
       1.56250000e+00, 3.88698630e-01, 2.88441781e+00, 1.04537671e+00,
       6.26712329e-01, 6.54280822e+00, 2.20890411e-01, 6.17294521e-01,
       8.87842466e-01, 1.05736301e+00, 1.77654110e+00, 4.76860445e+02,
       1.86643836e-01, 2.26883562e-01, 1.02739726e-01, 9.62063356e+01,
       4.65599315e+01, 2.30154110e+01, 3.63955479e+00, 1.50513699e+01,
       3.44863014e+00, 1.19863014e-02, 4.73150685e+01, 4.02397260e-02,
       5.13698630e-03, 2.22602740e-02, 3.66438356e-01, 4.45205479e-02,
       2.56849315e-03, 8.56164384e-03, 9.67465753e-02, 2.08904110e-01,
       4.53767123e-02, 1.19863014e-02, 3.68150685e-02, 1.62671233e-02,
       3.51027397e-02, 4.45205479e-02, 1.36986301e-02, 7.94520548e-01,
       1.39554795e-01, 9.96575342e-01, 9.34075342e-01, 3.08219178e-02,
       2.73972603e-02, 5.13698630e-03, 6.33561644e-01, 3.59589041e-02,
       2.56849315e-02, 8.95547945e-01, 5.90753425e-02, 2.82534247e-02,
       1.71232877e-03, 7.20890411e-01, 4.36643836e-02, 1.02739726e-02,
       1.71232877e-03, 9.41780822e-03, 4.28082192e-02, 2.05479452e-02,
       1.01027397e-01, 3.85273973e-02, 7.10616438e-02, 5.47945205e-02,
       2.56849315e-02, 7.70547945e-03, 2.91095890e-02, 1.55821918e-01,
       6.84931507e-03, 5.05136986e-02, 2.99657534e-02, 5.22260274e-02,
       7.36301370e-02, 1.79794521e-02, 5.13698630e-02, 4.36643836e-02,
       5.82191781e-02, 1.62671233e-02, 2.05479452e-02, 7.70547945e-03,
       5.73630137e-02, 8.60445205e-01, 5.13698630e-03, 1.45547945e-02,
       7.70547945e-03, 1.71232877e-02, 1.71232877e-03, 3.42465753e-03,
       5.13698630e-03, 9.88013699e-01, 8.56164384e-04, 1.71232877e-03,
       8.56164384e-04, 8.56164384e-04, 8.56164384e-04, 2.31164384e-02,
       3.51027397e-02, 2.48287671e-02, 7.70547945e-02, 1.02739726e-02,
       4.94863014e-01, 5.99315068e-03, 8.56164384e-03, 3.09075342e-01,
       2.73972603e-02, 4.02397260e-02, 8.56164384e-04, 3.42465753e-03,
       8.56164384e-04, 0.00000000e+00, 8.56164384e-04, 8.56164384e-04,
       8.56164384e-04, 8.56164384e-04, 6.84931507e-03, 8.56164384e-04,
       8.56164384e-04, 8.56164384e-04, 1.71232877e-03, 1.36986301e-02,
       8.56164384e-04, 2.56849315e-03, 8.56164384e-04, 5.13698630e-03,
       6.84931507e-03, 5.99315068e-03, 8.56164384e-04, 5.99315068e-03,
       1.71232877e-03, 1.88356164e-02, 5.13698630e-03, 5.13698630e-03,
```

[159]:
```
sc.mean_.shape
```

[159]: (503,)

[160]:
```
sc.n_features_in_
```

[160]: 503

[161]:
```
sc.n_samples_seen_
```

[161]: 1168

[162]:
```
sc.scale_
```

[162]:
```
array([4.15981688e+02, 2.28267425e+01, 8.95360697e+03, 1.00000000e+00,
       1.38955770e+00, 1.12386178e+00, 1.82140462e+02, 5.77296096e-01,
       4.47845178e-01, 8.85795053e-01, 1.09420320e+00, 1.53289293e+00,
       4.62466684e+02, 1.40444966e+00, 1.63450001e+02, 4.49183114e+02,
       4.42082880e+02, 7.73813003e-01, 3.90994498e+02, 4.39508104e+02,
       5.08710532e+01, 5.27816863e+02, 5.21391457e-01, 2.36597430e-01,
       5.51645654e-01, 5.04713080e-01, 8.16879086e-01, 2.16199731e-01,
       7.26955661e-01, 1.59779910e+00, 8.64546299e-01, 6.50296725e-01,
       6.59692853e-01, 8.62636071e-01, 7.45234418e-01, 2.14374940e+02,
       6.10490234e-01, 7.72424940e-01, 3.62722809e-01, 1.26104970e+02,
       6.63526009e+01, 6.31640322e+01, 2.90764117e+01, 5.50572314e+01,
       4.48777155e+01, 1.65086610e-01, 5.43031821e+02, 1.96520967e-01,
       7.14884443e-02, 1.47528825e-01, 4.81831181e-01, 2.06248561e-01,
       5.06151755e-02, 9.21321990e-02, 2.95612374e-01, 4.06525747e-01,
       2.08128966e-01, 1.08823848e-01, 1.88307512e-01, 1.26501004e-01,
       1.84039500e-01, 2.06248561e-01, 1.16236731e-01, 4.04051540e-01,
       3.46524536e-01, 5.84202812e-02, 2.48150352e-01, 1.72834971e-01,
       1.63238018e-01, 7.14884443e-02, 4.81831181e-01, 1.86187705e-01,
       1.58193602e-01, 3.05846077e-01, 2.35765660e-01, 1.65696013e-01,
       4.13448509e-02, 4.48561508e-01, 2.04347266e-01, 1.00838574e-01,
       4.13448509e-02, 9.65873341e-02, 2.02424493e-01, 1.41865172e-01,
       3.01364998e-01, 1.92465677e-01, 2.56927785e-01, 2.27578736e-01,
       1.58193602e-01, 8.74420096e-02, 1.68113714e-01, 3.62686432e-01,
       8.24766752e-02, 2.19002431e-01, 1.70492836e-01, 2.22482515e-01,
       2.61168030e-01, 1.32876602e-01, 2.20750991e-01, 2.04347266e-01,
       2.34157437e-01, 1.26501004e-01, 1.41865172e-01, 8.74420096e-02,
       2.32534940e-01, 3.46524536e-01, 7.14884443e-02, 1.19762066e-01,
       8.74420096e-02, 1.29730801e-01, 4.13448509e-02, 5.84202812e-02,
       7.14884443e-02, 1.08823848e-01, 2.92477583e-02, 4.13448509e-02,
```

[163]:
```
sc.var_
```

[163]:
```
array([1.73040765e+05, 5.21060171e+02, 8.01670777e+07, 0.00000000e+00,
       1.93087059e+00, 1.26306530e+00, 3.31751479e+04, 3.33270783e-01,
       2.00565303e-01, 7.84632876e-01, 1.19728065e+00, 2.34976074e+00,
       2.13875434e+05, 1.97247886e+00, 2.67159030e+04, 2.01765470e+05,
       1.95437273e+05, 5.98786563e-01, 1.52876697e+05, 1.93167373e+05,
       2.58786406e+03, 2.78590641e+05, 2.71849051e-01, 5.59783437e-02,
       3.04312928e-01, 2.54735293e-01, 6.67291442e-01, 4.67423238e-02,
       5.28464534e-01, 2.55296198e+00, 7.47440303e-01, 4.22885831e-01,
       4.35194660e-01, 7.44140992e-01, 5.55374337e-01, 4.59566149e+04,
       3.72698325e-01, 5.96640288e-01, 1.31567836e-01, 1.59024634e+04,
```

```
[164]: sc.with_mean
```

```
[164]: True
```

```
[165]: sc.with_std
```

```
[165]: True
```

```
[166]: ### Carry Forward for Deployment
# sc.mean_
# sc.mean_.shape
# sc.n_features_in_
# sc.n_samples_seen_
# sc.scale_
# sc.var_
# sc.with_mean
# sc.with_std
```

Train ML Model

```
[167]: !pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\gk521\anaconda3\lib\site-packages (1.7.5)
Requirement already satisfied: scipy in c:\users\gk521\anaconda3\lib\site-packages (from xgboost) (1.10.0)
Requirement already satisfied: numpy in c:\users\gk521\anaconda3\lib\site-packages (from xgboost) (1.23.5)
```

```
[168]: from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
# from sklearn.isotonic import IsotonicRegression

from sklearn.neural_network import MLPRegressor

from xgboost import XGBRegressor
```

```
[169]: svr = SVR()
lr = LinearRegression()
sgdr = SGDRegressor()
knr = KNeighborsRegressor()
gpr = GaussianProcessRegressor()
dtr = DecisionTreeRegressor()
gbr = GradientBoostingRegressor()
rfr = RandomForestRegressor()
# ir = IsotonicRegression()
mlpr = MLPRegressor()
xgbr = XGBRegressor()
```

```
[170]: models = {"a":["LinearRegression", lr],
              "b":["SVR", svr],
              "c":["SGDRegressor", sgdr],
              "d":["KNeighborsRegressor", knr],
              "e":["GaussianProcessRegressor", gpr],
              "f":["DecisionTreeRegressor", dtr],
              "g":["GradientBoostingRegressor", gbr],
              "h":["RandomForestRegressor", rfr],
              # "i":["IsotonicRegression", ir],
              "j":[" MLPRegressor", mlpr],
              "k":["XGBRegressor", xgbr]
             }
```

```
[171]: from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import make_scorer, r2_score

def test_model(model, X_train=X_train, y_train=y_train):
    cv = KFold(n_splits = 3, shuffle=True, random_state = 45)
    r2 = make_scorer(r2_score)
    r2_val_score = cross_val_score(model, X_train, y_train, cv=cv, scoring = r2)
    score = [r2_val_score.mean()]
    return score
```

```
[172]: models_score = []
for model in models:
    print("Training Modl:", models[model][0])
    score = test_model(models[model][1], X_train, y_train)
    print("Score of Model:", score)
    models_score.append([models[model][0]])
```

```
Training Modl: LinearRegression
Score of Model: [-2.4515733739715804e+27]
Training Modl: SVR
Score of Model: [-0.05389457217030317]
Training Modl: SGDRegressor
Score of Model: [-1076.543287044941]
Training Modl: KNeighborsRegressor
Score of Model: [0.4740785802671166]
Training Modl: GaussianProcessRegressor
Score of Model: [-5.309666460758808]
Training Modl: DecisionTreeRegressor
Score of Model: [0.6987954147886363]
Training Modl: GradientBoostingRegressor
Score of Model: [0.8556197507716349]
Training Modl: RandomForestRegressor
Score of Model: [0.8363289086751456]
Training Modl:  MLPRegressor
C:\Users\gk521\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\gk521\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
C:\Users\gk521\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:684: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
  warnings.warn(
Score of Model: [-5.127723278735962]
Training Modl: XGBRegressor
Score of Model: [0.8416360247322457]
```

```
[ ]: models_score
```

```
173]: [['LinearRegression'],
 ['SVR'],
 ['SGDRegressor'],
 ['KNeighborsRegressor'],
 ['GaussianProcessRegressor'],
 ['DecisionTreeRegressor'],
 ['GradientBoostingRegressor'],
 ['RandomForestRegressor'],
 [' MLPRegressor'],
 ['XGBRegressor']]
```

CHAPTER 6

CONCLUSION AND LEARNING EXPERIENCE

6.1 Conclusion

To get more accuracy, we trained all top supervised regression algorithms but you can try out a few of them which are always popular. After training all algorithms, we found that GradientBoostingRegressor and XGBoost regressor have given high accuracy than remain but we have chosen XGBoost.

As ML Engineer, we always retrain the deployed model after some period of time to sustain the accuracy of the model. We hope our efforts will help to predict the price of a house for the buyer and seller.

6.2 Learning Experience

If you're interested in learning how to predict house prices using machine learning, here's a step-by-step learning experience that you can follow:

Understand the problem: Begin by familiarizing yourself with the task of house price prediction. Understand the variables involved, such as square footage, number of bedrooms, location, etc., and the target variable, which is the house price.

Gather a dataset: Look for a suitable dataset that includes relevant features and corresponding house prices. You can explore resources like Kaggle, UCI Machine Learning Repository, or real estate websites. Make sure the dataset is reliable and well-documented.

Data preprocessing: Clean the dataset by handling missing values, outliers, and inconsistent data. Perform feature engineering, which involves transforming or creating new features that might enhance the predictive power. For example, you could calculate the price per square foot or extract additional information from addresses or descriptions.

Split the data: Divide your dataset into two subsets: a training set and a test set. The training set will be used to train your machine learning model, while the test set will serve as an independent evaluation to assess its performance.

Choose a model: Research and select an appropriate machine learning algorithm for regression tasks. Linear regression, decision trees, random forests, or gradient boosting algorithms are popular choices for house price prediction. Consider factors like interpretability, model complexity, and scalability when making your decision.

Train the model: Use the training set to train your chosen model. During this phase, the model will learn the patterns and relationships in the data, adjusting its internal parameters accordingly. Adjust hyperparameters (e.g., learning rate, regularization) to optimize the model's performance.

Evaluate the model: Apply the trained model to the test set to assess its performance. Common evaluation metrics for regression tasks include mean absolute error (MAE), mean squared error (MSE), and R-squared value. Analyze the results to understand how well the model is predicting house prices.

Iterate and improve: Analyze the model's performance and identify areas for improvement. You can try different algorithms, feature engineering techniques, or hyperparameter tuning to enhance the model's accuracy. Consider using cross-validation or more advanced techniques like ensemble learning to further improve the predictions.

Deploy the model: Once you're satisfied with the model's performance, deploy it to make predictions on new, unseen data. You can create a user-friendly interface or integrate it into a web application to allow users to input property features and obtain price predictions.

Continued learning: Keep exploring and learning about more advanced techniques for house price prediction. This can include incorporating external data sources, using deep learning models, or exploring time series analysis for housing market trends.

REFERENCES

[1] Jiawei Han, MichelineKamber, "Data Mining Concepts and Techniques", pp. 279-328, 2001.

[2] Adyan Nur Alfiyatin , Hilman Taufiq, Ruth Ema Febrita and Wayan Firdaus Mahmudy, "Modeling House Price Prediction using Regression Analysis and Particle Swarm Optimization", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 8, No. 10, 2017, pp. 323 to 326.

[3] Ong, S. E., Ho, K. H. D. and Lim, C. H., "A constantquality price index for resale public housing flats in Singapore", Urban Studies, 40(13), 2003, pp. 2705 –2729.

[4] Nikita Patel and Saurabh Upadhyay, "Study of Various Decision Tree Pruning Methods with their Empirical Comparison in WEKA", International Journal of Computer Applications, Volume 60– No.12, December 2012, pp 20-25.

[5] SamDrazin and Matt Montag", Decision Tree Analysis using Weka", Machine Learning-Project II, University of Miami.

[6] J. R.Quinlan," Simplifying decision trees", Int. J. HumanComputer Studies.

[7] K.C. Tan, E.J. Teoh, Q. Yu, K.C. Goh," A hybrid evolutionary algorithm for attribute selection in data mining", Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576, Singapore.Rochester, Institute of Technology, USA

[8] Timothy C. Au, "Random Forests, Decision Trees, and Categorical Predictors: The Absent Levels Problem", Journal of Machine Learning Research 19, 2018, pp. no.1- 30