House Prices - Advanced Regression TechniquesPredict using machine learing

Phase 2: PROJECT

SUBMITED BY:

NAME: N.MANIKANDAN

NM ID: AU712521104018,

EMAIL ID: 12beast11boy@gmail.com,

CLASS: BE-CSE 3rd YEAR,

COLLEGE: PPG INSTITUTE OF TECHNOLOGY

INTRODUCTION

House price prediction is an important and complex problem in the field of real estate. With the ever-increasing demand for housing, accurate predictions of house prices are essential for making sound decisions. The existing and proposed models have been compared against each other to determine the most accurate one. Our research also provides an overview of the current literature on house price prediction and discuss the various techniques and models used in this field. In addition, it analyzes the strengths and weaknesses of each model [3,12], as well as their application in the real estate market. Finally, the paper concludes with recommendations for future research in this field.

Accurate house price prediction is real estate market can be an important aspect in terms of finance management. It requires careful analysis and understanding of the factors that influence house prices. This research paper aims to explore the factors that affect house prices and develop a predictive model to accurately forecast prices for a given house. The factors that will be examined on the basis of economic, demographic, geographic, and housing characteristics. The data has been collected from sources such as public records, census data, and other surveys. The predictive model developed in this research have been evaluated using statistical methods to determine its accuracy.

In recent years, there has been a surge in the use of machine learning algorithms [4] to predict house prices. Machine learning algorithms [17] have been proven to be effective in predicting house prices due to their ability to learn from data and make accurate predictions.

Machine learning can be used to predict the price of a house by using a variety of data points. This can include features such as location, square footage, number of bedrooms and bathrooms, lot size, and any other features that may impact the price. By using machine learning algorithms such as Regression, Decision Trees, and Random Forest, the system can take in all of these features and provide a more accurate prediction of a house's price than traditional methods. This can help buyers and sellers make better decisions and more efficiently negotiate a price. House price prediction using machine learning algorithms is a powerful tool for accurately predicting the price of a house. It uses various algorithms such as linear regression, decision trees, support vector machines, and neural networks to analyze relevant data and predict house prices. Machine learning algorithms can be used to detect patterns and correlations in large datasets. With the help of machine learning algorithms, investors and homeowners can benefit from the insights provided by models to make more informed decisions.

In this research paper, we have explored the various machine learning algorithms that are used to predict house prices and discuss their effectiveness. We have also discussed the challenges associated with predicting house prices, such as data availability and accuracy of the predictions.

The "House Prices - Advanced Regression Techniques" competition is a machine learning competition on Kaggle. It is a popular competition in the field of data science and machine learning, designed to challenge participants to develop predictive models for estimating the sale prices of residential homes. The competition is based on the Ames Housing dataset, which contains a wide range of features describing various aspects of residential properties, such as square footage, number of bedrooms, neighborhood, and more.

Here are some key points and steps typically involved in participating in the "House Prices - Advanced Regression Techniques" competition:

1. **Data Exploration**: Start by loading and exploring the dataset. Understand the features, their data types, and their relationships with the target variable (sale price). You may use tools like Python and libraries like Pandas and Matplotlib for this.
2. **Data Preprocessing**: Clean and preprocess the data. This involves handling missing values, dealing with outliers, and transforming categorical variables into a format that can be used by machine learning algorithms (e.g., one-hot encoding or label encoding).
3. **Feature Engineering**: Create new features or modify existing ones to potentially improve the model's predictive power. This may involve combining or transforming variables to make them more informative.
4. **Model Selection**: Choose appropriate regression algorithms for the task. Common choices include Linear Regression, Random Forest, Gradient Boosting, and more advanced methods like XGBoost and LightGBM.
5. **Model Training and Evaluation**: Split the dataset into training and validation sets, train your chosen models on the training data, and evaluate their performance using appropriate metrics (e.g., Root Mean Squared Error, Mean Absolute Error).
6. **Hyperparameter Tuning**: Optimize the hyperparameters of your models to improve their performance. This can be done through techniques like grid search or random search.
7. **Ensemble Methods**: Consider using ensemble techniques, such as stacking or bagging, to combine the predictions of multiple models to potentially improve accuracy.
8. **Submission**: Once you are satisfied with your model's performance on the validation data, make predictions on the competition's test dataset and submit your results to Kaggle for evaluation.

9. **Iterate**: The competition often allows multiple submissions, so you can fine-tune your model and try different strategies to improve your score.
10. **Community and Discussion**: Kaggle competitions usually have a discussion forum where participants can share insights, code, and approaches. Engaging with the community can be valuable for learning and improving your solution.

Keep in mind that this competition is more advanced and challenging compared to beginner-level competitions. You may encounter various complexities in the data that require creative solutions and advanced machine learning techniques.

The ultimate goal is to build a model that can accurately predict the sale prices of houses in the test dataset, which is hidden from participants until the competition is over. Participants are ranked based on the accuracy of their predictions, and prizes or recognition are often awarded to top performers.

To get started, you can visit Kaggle's website and search for the "House Prices - Advanced Regression Techniques" competition to access the dataset, rules, and resources.

Contant of project phase 2

Consider exploring advanced regression techniques like Gradient Boosting or XGBoost for

improved Prediction accuracy.

Data Source

A good data source for house price prediction using machine learning should be

Accurate, Complete, Covering the geographic area of interest, Accessible .

Dataset Link: (https://www.kaggle.com/datasets/vedavyasv/usa-housing)

Data Collection and Preprocessing:

# Importing the dataset: Obtain a comprehensive dataset containing relevant features

such as square footage, number of bedrooms, location, amenities, etc.

# Data preprocessing: Clean the data by handling missing values, outliers, and

categorical variables. Standardize or normalize numerical features.

Exploratory Data Analysis (EDA):

# Visualize and analyze the dataset to gain insights into the relationships between variables.

# Identify correlations and patterns that can inform feature selection and engineering.

# Present various data visualizations to gain insights into the dataset.

# Explore correlations between features and the target variable (house prices).

# Discuss any significant findings from the EDA phase that inform feature selection.

Advanced Regression Techniques:

⏶ **Ridge Regression:** Introduce L2 regularization to mitigate multicollinearity and overfitting.

⏶ **Lasso Regression:** Employ L1 regularization to perform feature selection and simplify the model.

⏶ **ElasticNet Regression:** Combine both L1 and L2 regularization to benefit from their

respective advantages.

⏶ **Random Forest Regression:** Implement an ensemble technique to handle non-linearity and capture complex relationships in the data.

⏶ **Gradient Boosting Regressors** (e.g., XGBoost, LightGBM): Utilize gradient boosting algorithms for improved accuracy.

SYSTEM DESIGN AND ARCHITECTURE

The system architecture of a house price prediction system would typically involve the following components:

1. *Data Sources:* The data sources used to build the system would include publicly available real estate market data, such as data related to real estate transactions, home appraisals, and market trends.

2. *Data Storage:* The data would need to be stored in a database or other type of data storage system. This could be a cloud-based storage system, a local database, or some other type of data warehouse.
3. *Data Pre-Processing:* The raw data from the various sources would need to be pre-processed in order to be used for the system. This could involve extracting the relevant features from the data and normalizing it.
4. *Data Collection:* The data for the system would need to be collected from the various sources. This could involve web scraping, APIs, or manual data entry.
5. *Data Cleaning:* The data would need to be cleaned and pre-processed in order to be used for the system. This could involve removing outliers, normalizing the data, and extracting relevant features.
6. *Algorithm:* The algorithm used to build the model would depend on the type of model being used. It could be a supervised learning algorithm such as linear regression, or an unsupervised learning algorithm such as k-means clustering.
7. *Model Design:* The model would need to be designed based on the data and the chosen algorithm. This could involve selecting the appropriate features, defining the model parameters, and tuning the model.
8. *Model Building:* The model would need to be built based on the pre-processed data and the chosen algorithm. This could be done using a machine learning library or a custom-built mode
9. *Model Validation:* The model would need to be validated to ensure that it is accurate and reliable. This could involve testing the model on a test dataset or using cross-validation techniques.
10. *Model Deployment:* The model would need to be deployed to an application or service for use by users. This could be a web application or a mobile application.

METHODOLOGY USED

House price prediction using machine learning algorithms is a popular technique [9] [18] for predicting the prices of houses. The goal is to use predictive models to accurately predict the future values of houses based on historical data. The generics flow of methodology adoption

1

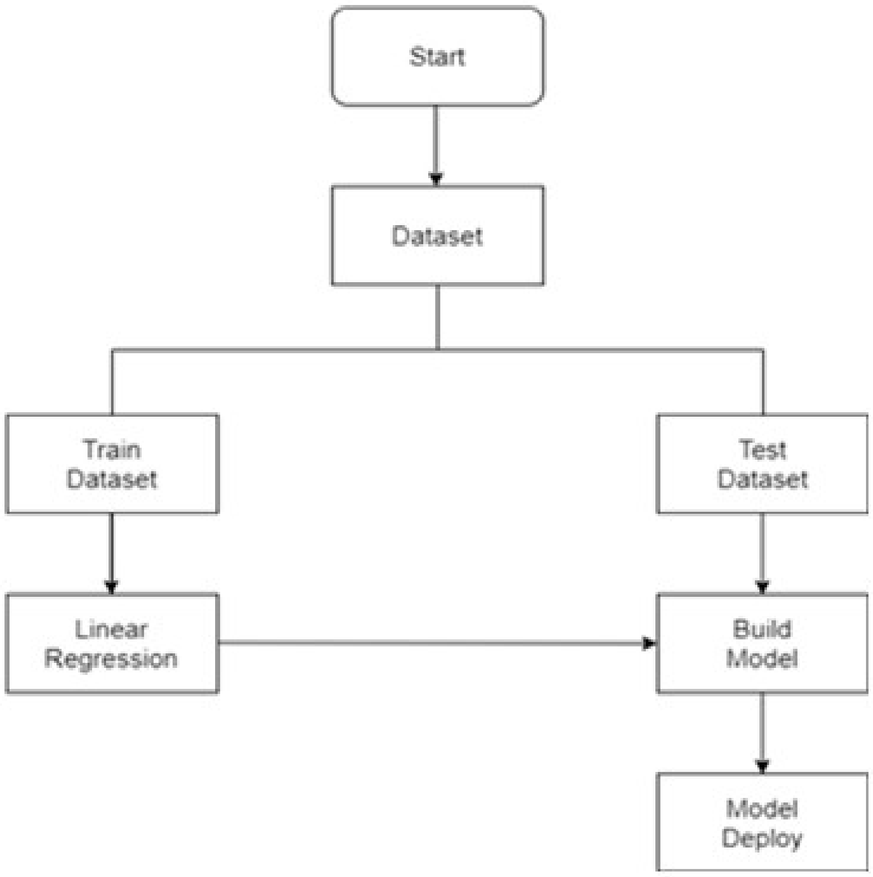


Figure 1. Generic methodology flow

The first step in the process is to collect data. Data points that can help predict the house prices could include the size of the house, the age of the house, the location of the house, the number of bedrooms and bathrooms, the type of construction, the condition of the house, the number of nearby amenities, and any other relevant factors.

The next step is to preprocess the data. This involves cleaning the data to ensure that it is accurate and reliable, and transforming it into a format that can be used by machine learning algorithms.

Once the data has been preprocessed, the machine learning algorithms can be used to build a predictive model. Different Machine learning algorithms used for house price prediction include linear regression, decision trees, random forests.

The model can then be evaluated to assess its accuracy and reliability. This is done by comparing its predicted price against actual house prices.

1

Model Evaluation and Selection:

Split the dataset into training and testing sets.

- Evaluate models using appropriate metrics (e.g., Mean Absolute Error, Mean Squared Error, R-squared) to assess their performance.
- Use cross-validation techniques to tune hyperparameters and ensure model stability.
- Compare the results with traditional linear regression models to highlight improvements.
- Select the best-performing model for further analysis.

Model Interpretability:

Explain how to interpret feature importance from Gradient Boosting and XGBoost models.

- Discuss the insights gained from feature importance analysis and their relevance to house price prediction.
- Interpret feature importance from ensemble models like Random Forest and Gradient Boosting to understand the factors influencing house prices.

Deployment and Prediction:

Deploy the chosen regression model to predict house prices.

- Develop a user-friendly interface for users to input property features and receive price predictions.

Program:

House Price Prediction

Importing Dependencies

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import r2_score,
mean_absolute_error,mean_squared_errorfrom
sklearn.linear_model import LinearRegression

from sklearn.linear_model import Lasso

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

import xgboost as xg

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")
```

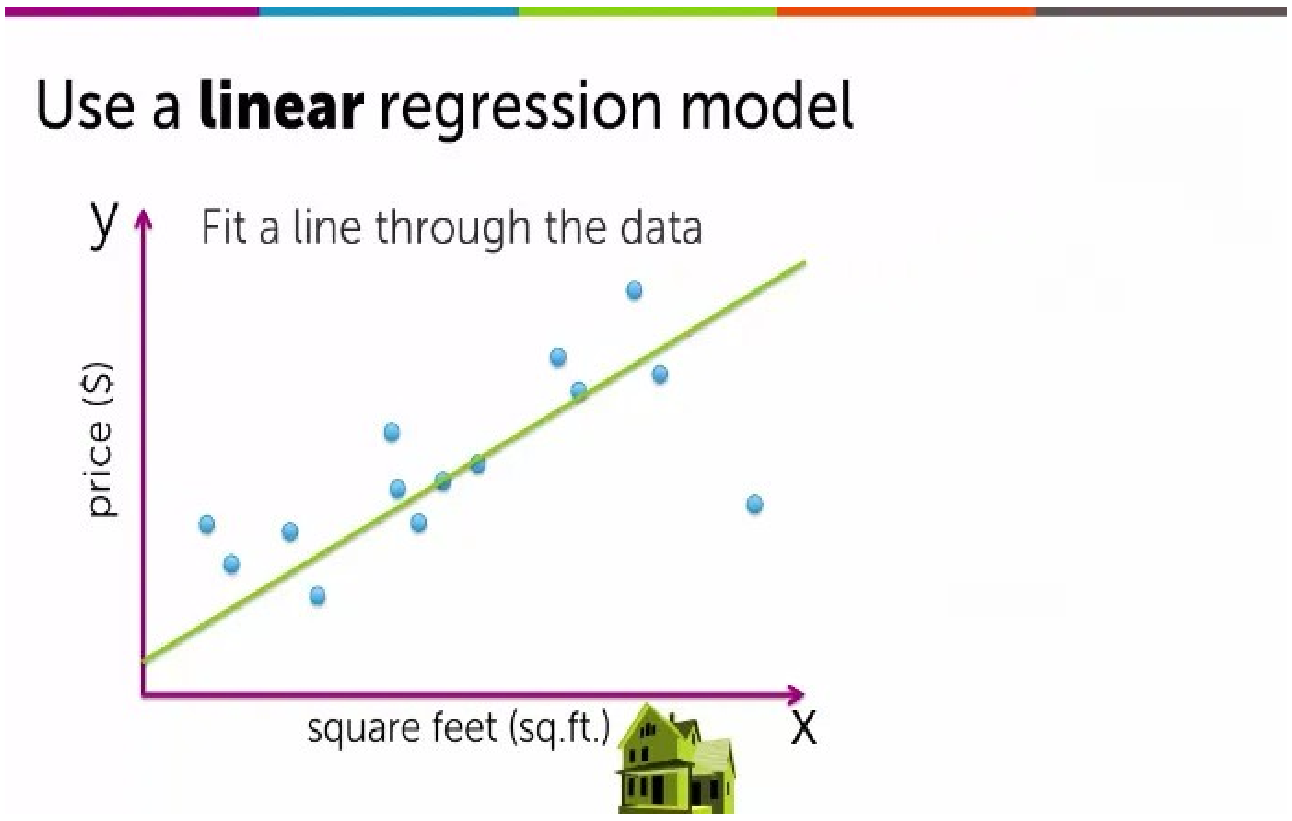
```
/opt/conda/lib/python3.10/site-packages/scipy/_init_.py:146:
UserWarning: A NumPy

version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version

1.23.5
```

```
warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
Loading Dataset
dataset = pd.read_csv('E:/USA_Housing.csv')
```

Model 1 - Linear Regression

Linear Regression: Fitting a Straight Line to Data

Linear regression is a widely used learning algorithm that involves fitting a straight line to a dataset. Imagine having a dataset of house sizes and prices from a city like Portland, USA. By plotting the data points on a graph, with house size on the horizontal axis and price on the vertical axis, you can visualize the relationship between the two variables.



To predict the price of a house based on its size, a linear regression model is built. The model fits a straight line to the data points, aiming to capture the underlying trend. By extending this line, you can estimate the price for a given house size. Linear regression falls under the category of regression models, as it predicts numerical values, such as house prices in dollars.

Training a Linear Regression Model

To train the linear regression model, a training set is used. The training set comprises pairs of **input features** *(eg: square feet)* and corresponding **output targets** *(eg: house price)*.

Let:

- *"m" be the number of training examples.*
- *"x" be the input feature representing a specific property of the house, such as its size.*
- *"y" be the output target, representing the actual price of the house.*
- *"i" be the index of a specific training example.*

A specific training example is denoted as `(x^i, y^i)`.

Therefore, the equation can be written as:

```
Training set: {(x¹, y¹), (x², y²), …, (x^m, y^m)}
```

*where each pair **(x^i, y^i)** represents a specific house in the dataset, and "m" denotes the number of training examples.*

The Function f(x) and Model Representation

When training the model, a learning algorithm produces a function (f) that takes **an input feature (x)** and **outputs an estimate or prediction (ŷ)**. In linear regression, the function f(x) is defined as:

$$f_{w,b}(x^{(i)}) = wx^{(i)} + b$$

where:

- *"w" and "b" are numeric values that determine the line's **slope** and **intercept**, respectively*

A crucial aspect of linear regression is the cost function. The cost function measures the model's performance by quantifying the difference between predicted values and actual targets. This concept is widely applicable in machine learning and plays a vital role in training advanced AI models.

In the next article, we'll delve deeper into the cost function and explore how it contributes to training linear regression models.

In conclusion, linear regression is a valuable tool for predicting house prices based on specific features. Its simplicity and effectiveness make it widely used in the field of machine learning. By fitting a straight line to the data, the linear regression model can estimate the price of a house based on its size. This example demonstrates the concept of supervised learning, where the model is trained using labeled data to predict numeric outputs.

Multiple Linear Regression

Problem Statement:

Consider a real estate company that has a dataset containing the prices of properties in the Delhi region. It wishes to use the data to optimise the sale prices of the properties based on important factors such as area, bedrooms, parking, etc.

Essentially, the company wants —

- To identify the variables affecting house prices, e.g. area, number of rooms, bathrooms, etc.
- To create a linear model that quantitatively relates house prices with variables such as number of rooms, area, number of bathrooms, etc.
- To know the accuracy of the model, i.e. how well these variables can predict house prices.

Data

Use housing dataset.

Reading and Understanding the Data

```
# Supress Warnings

import warnings
warnings.filterwarnings('ignore')

# Import the numpy and pandas package
```

```
import numpy as np
import pandas as pd

# Data Visualisation

import matplotlib.pyplot as plt
import seaborn as sns
```

```
housing = pd.DataFrame(pd.read_csv("../input/Housing.csv"))
```

```
# Check the head of the dataset
housing.head()
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	aircondit
0	13300000	7420	4	2	3	yes	no	no	no	yes
1	12250000	8960	4	4	4	yes	no	no	no	yes
2	12250000	9960	3	2	2	yes	no	yes	no	no
3	12215000	7500	4	2	2	yes	no	yes	no	yes
4	11410000	7420	4	1	2	yes	yes	yes	no	yes

Data Inspection

```
housing.shape

(545, 13)

housing.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
price                545 non-null int64
area                 545 non-null int64
bedrooms             545 non-null int64
bathrooms            545 non-null int64
stories              545 non-null int64
mainroad             545 non-null object
```

```
guestroom           545 non-null object
basement            545 non-null object
hotwaterheating     545 non-null object
airconditioning     545 non-null object
parking             545 non-null int64
prefarea            545 non-null object
furnishingstatus    545 non-null object
dtypes: int64(6), object(7)
memory usage: 55.4+ KB
```

```
housing.describe()
```

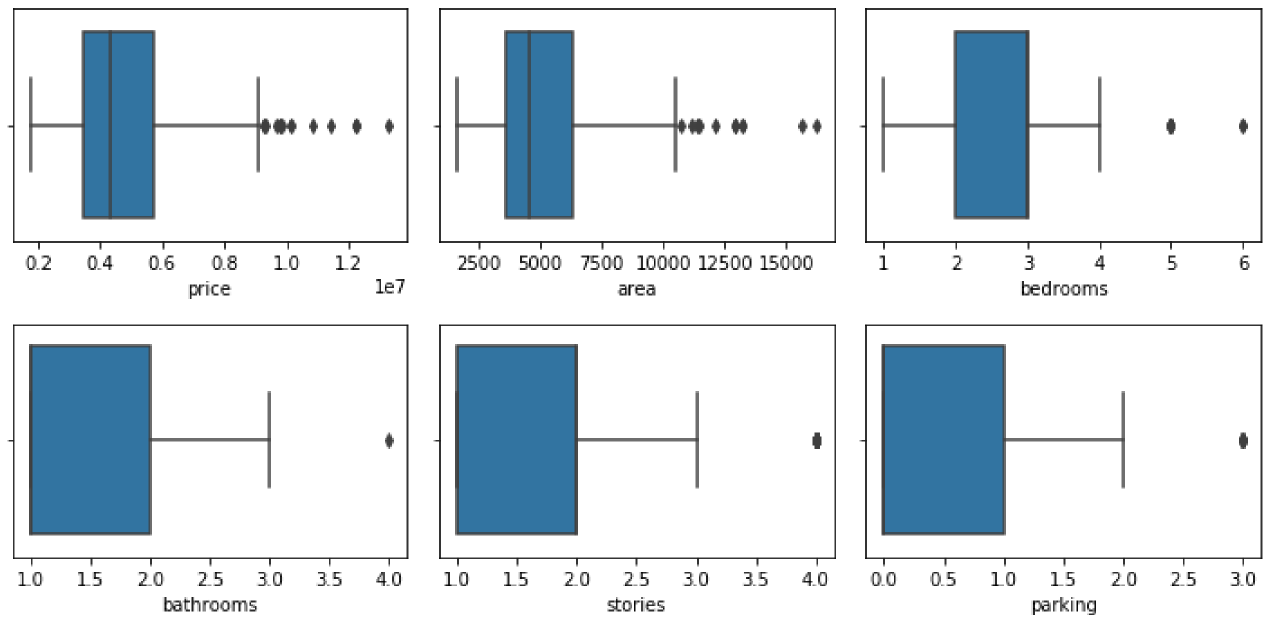
	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

Data Cleaning

```
# Checking Null values
housing.isnull().sum()*100/housing.shape[0]
# There are no NULL values in the dataset, hence it is
clean.

price               0.0
area                0.0
```

```
bedrooms            0.0
bathrooms           0.0
stories             0.0
mainroad            0.0
guestroom           0.0
basement            0.0
hotwaterheating     0.0
airconditioning     0.0
parking             0.0
prefarea            0.0
furnishingstatus    0.0
dtype: float64
```

```
# Outlier Analysis
fig, axs = plt.subplots(2,3, figsize = (10,5))
plt1 = sns.boxplot(housing['price'], ax = axs[0,0])
plt2 = sns.boxplot(housing['area'], ax = axs[0,1])
plt3 = sns.boxplot(housing['bedrooms'], ax = axs[0,2])
plt1 = sns.boxplot(housing['bathrooms'], ax = axs[1,0])
plt2 = sns.boxplot(housing['stories'], ax = axs[1,1])
plt3 = sns.boxplot(housing['parking'], ax = axs[1,2])

plt.tight_layout()
```



```
plt.boxplot(housing.price)
Q1 = housing.price.quantile(0.25)
Q3 = housing.price.quantile(0.75)
IQR = Q3 - Q1
housing = housing[(housing.price >= Q1 - 1.5*IQR) &
(housing.price <= Q3 + 1.5*IQR)]
```

Residual Analysis of the train data

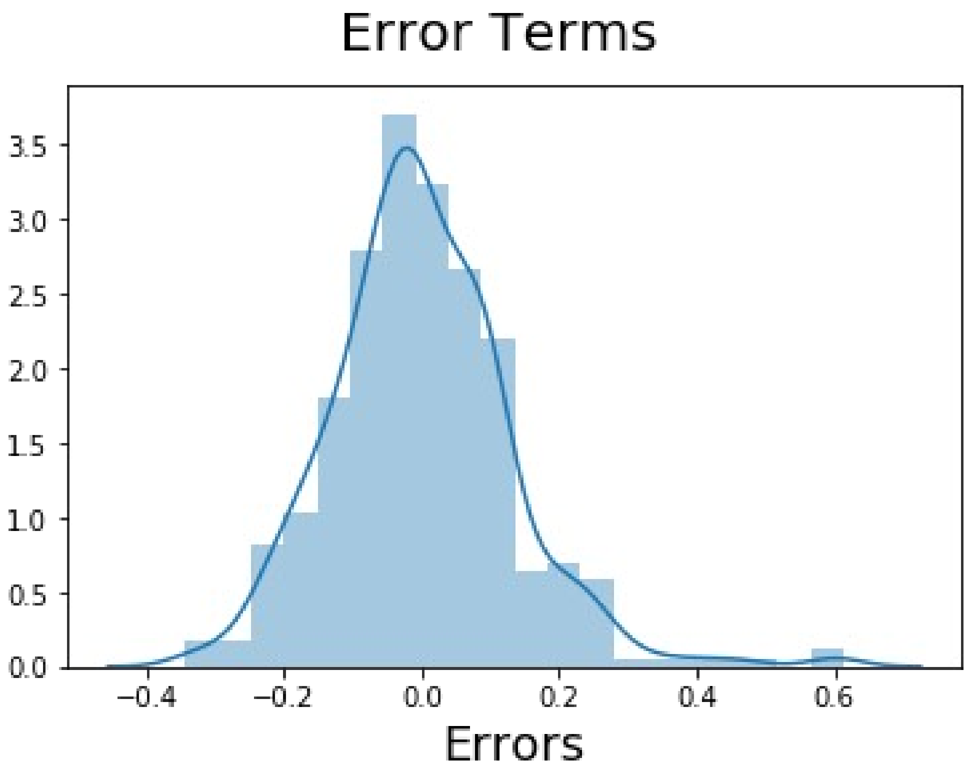
So, now to check if the error terms are also normally distributed (which is infact, one of the major assumptions of linear regression), let us plot the histogram of the error terms and see what it looks like.

```
y_train_price = lm.predict(X_train_rfe)
```

```
res = (y_train_price - y_train)
```

```
# Importing the required libraries for plots.
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((y_train - y_train_price), bins = 20)
fig.suptitle('Error Terms', fontsize = 20)
# Plot heading
plt.xlabel('Errors', fontsize = 18)
# X-label

Text(0.5,0,'Errors')
```



Model 2 - Support Vector Regressor

Support Vector Regressor (SVR) is a machine learning algorithm used for regression tasks. It's an extension of the Support Vector Machine (SVM) algorithm, which is primarily used for classification. SVR is well-suited for modeling complex relationships in data, especially when you have non-linear relationships between features and the target variable. Here's how you can implement and use a Support Vector Regressor (Model 2) in Python:

1. **Import Libraries**: First, you need to import the necessary libraries, including `numpy` for numerical operations, `pandas` for data manipulation, and `sklearn` for machine learning tasks:

   python

```
• import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

- **Load and Prepare Data**:
- Load your house price prediction dataset and split it into features (X) and the target variable (y). You should also split the data into training and testing sets:

python

```
• # Load your dataset, replace 'your_data.csv' with
  your dataset file
data = pd.read_csv('your_data.csv')

# Split data into features (X) and target variable (y)
X = data.drop('SalePrice', axis=1)
y = data['SalePrice']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

- **Data Preprocessing**: You may need to preprocess your data, which can include handling missing values, encoding categorical variables, and scaling the features. For SVR, feature scaling is essential because it relies on distances between data points:

python

```
• # Feature scaling using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

- **Train the Support Vector Regressor**: Create an SVR model and train it on your training data:

python

```
• svr = SVR(kernel='linear')  # You can choose
  different kernel functions (e.g., 'linear', 'poly',
  'rbf')
svr.fit(X_train, y_train)
```

- **Make Predictions**: Use the trained SVR model to make predictions on the test data:

python

```
• y_pred = svr.predict(X_test)
```

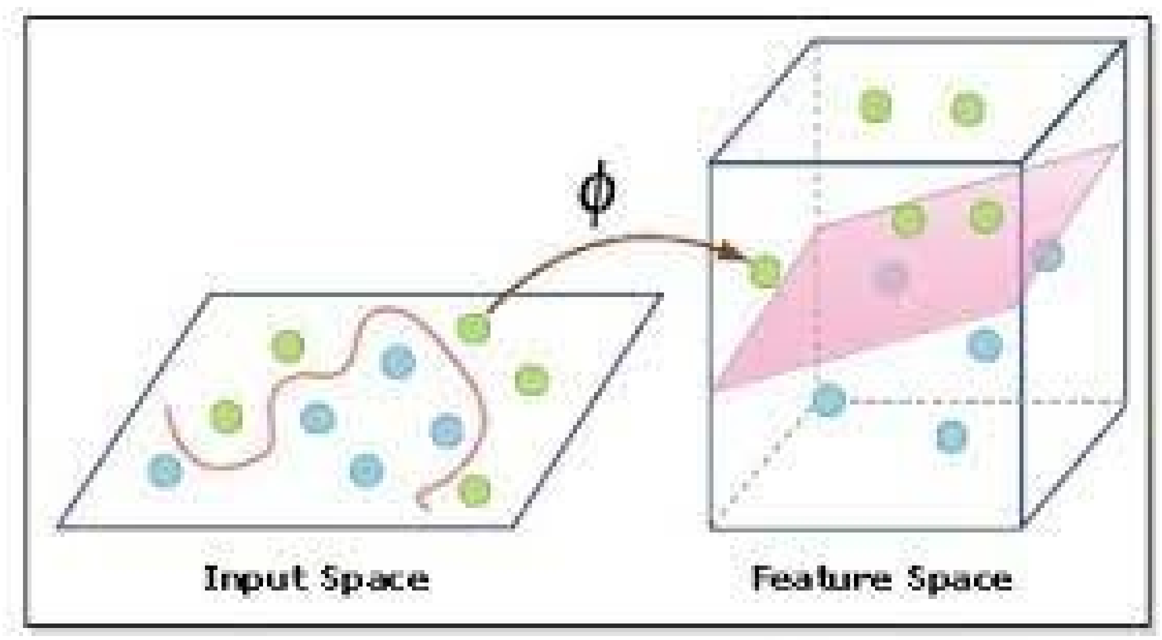
- **Evaluate the Model**: Measure the performance of your SVR model using appropriate regression metrics. A common metric for regression is Mean Squared Error (MSE):

python

```
6.mse = mean_squared_error(y_test, y_pred)
  print(f"Mean Squared Error: {mse}")
```

7. **Tune Hyperparameters** (Optional): You can fine-tune SVR hyperparameters like the choice of kernel, regularization parameter (C), and other settings to optimize the model's performance.

Remember that SVR can be computationally intensive, especially with large datasets. You may need to experiment with different kernel functions and hyperparameters to find the best model for your specific problem. Additionally, cross-validation can help you assess the model's robustness and generalization performance.



Model 3 - Lasso Regression

A linear model that estimates sparse coefficients.

Mathematically, it consists of a linear model trained with $\ell 1$

prior as regularizer. The objective function to minimize is:

$$\min w 1 2 n samples \mid \mid \mid \mid Xw - y \mid \mid \mid \mid 2 2 + \alpha \mid \mid \mid \mid w \mid \mid \mid \mid 1$$

The lasso estimate thus solves the minimization of the least-squares penalty with $\alpha \mid \mid \mid \mid w \mid \mid \mid \mid 1$

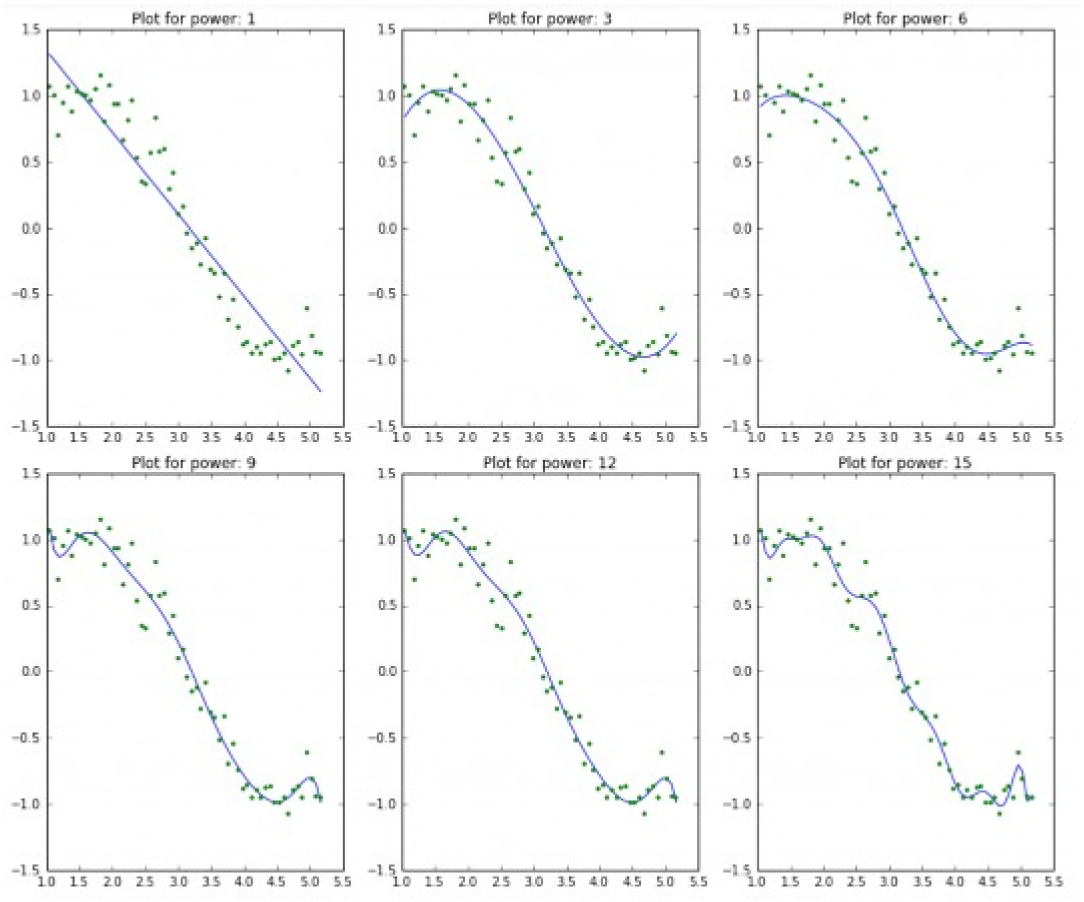
added, where α is a constant and $\mid \mid \mid \mid w \mid \mid \mid \mid 1$ is the $\ell 1 - norm$

of the parameter vector.

```
from sklearn.linear_model import Lasso

model = Lasso(alpha=0.1,
              precompute=True,
#               warm_start=True,
              positive=True,
              selection='random',
              random_state=42)
model.fit(X_train, y_train)
```

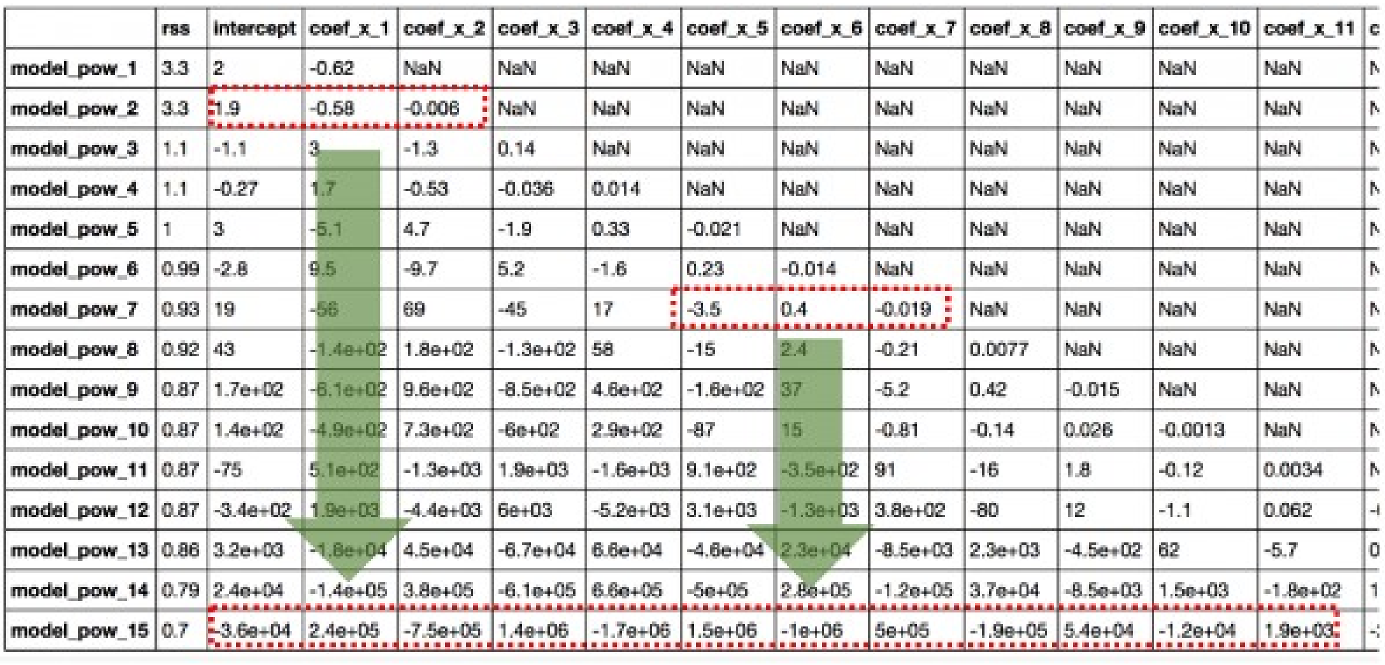
```
test_pred = model.predict(X_test)
train_pred = model.predict(X_train)

print('Test set evaluation:\
n_____________________________________')
print_evaluate(y_test, test_pred)
print('====================================')
print('Train set evaluation:\
n_____________________________________')
print_evaluate(y_train, train_pred)

results_df_2 = pd.DataFrame(data=[["Lasso Regression",
*evaluate(y_test, test_pred) , cross_val(Lasso())]],
                            columns=['Model', 'MAE',
'MSE', 'RMSE', 'R2 Square', "Cross Validation"])
results_df = results_df.append(results_df_2,
ignore_index=True)
```

```
Test set evaluation:
_____________________________________
MAE: 81135.6985172622
MSE: 10068453390.364523
RMSE: 100341.68321472648
R2 Square 0.914681588551116
```



```
Train set evaluation:
_____________________________________
MAE: 81480.63002185506
MSE: 10287043196.634295
RMSE: 101425.0619750084
R2 Square 0.9192986576295505
```

output like this:



	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10	coef_x_11	c
model_pow_1	3.3	2	-0.62	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_2	3.3	1.9	-0.58	-0.006	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_3	1.1	-1.1	3	-1.3	0.14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_4	1.1	-0.27	1.7	-0.53	-0.036	0.014	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_5	1	3	-5.1	4.7	-1.9	0.33	-0.021	NaN	NaN	NaN	NaN	NaN	NaN	N
model_pow_6	0.99	-2.8	9.5	-9.7	5.2	-1.6	0.23	-0.014	NaN	NaN	NaN	NaN	NaN	N
model_pow_7	0.93	19	-56	69	-45	17	-3.5	0.4	-0.019	NaN	NaN	NaN	NaN	N
model_pow_8	0.92	43	-1.4e+02	1.8e+02	-1.3e+02	58	-15	2.4	-0.21	0.0077	NaN	NaN	NaN	N
model_pow_9	0.87	1.7e+02	-6.1e+02	9.6e+02	-8.5e+02	4.6e+02	-1.6e+02	37	-5.2	0.42	-0.015	NaN	NaN	N
model_pow_10	0.87	1.4e+02	-4.9e+02	7.3e+02	-6e+02	2.9e+02	-87	15	-0.81	-0.14	0.026	-0.0013	NaN	N
model_pow_11	0.87	-75	5.1e+02	-1.3e+03	1.9e+03	-1.6e+03	9.1e+02	-3.5e+02	91	-16	1.8	-0.12	0.0034	N
model_pow_12	0.87	-3.4e+02	1.9e+03	-4.4e+03	6e+03	-5.2e+03	3.1e+03	-1.3e+03	3.8e+02	-80	12	-1.1	0.062	-
model_pow_13	0.86	3.2e+03	-1.8e+04	4.5e+04	-6.7e+04	6.6e+04	-4.6e+04	2.3e+04	-8.5e+03	2.3e+03	-4.5e+02	62	-5.7	0
model_pow_14	0.79	2.4e+04	-1.4e+05	3.8e+05	-6.1e+05	6.6e+05	-5e+05	2.8e+05	-1.2e+05	3.7e+04	-8.5e+03	1.5e+03	-1.8e+02	1
model_pow_15	0.7	-3.6e+04	2.4e+05	-7.5e+05	1.4e+06	-1.7e+06	1.5e+06	-1e+06	5e+05	-1.9e+05	5.4e+04	-1.2e+04	1.9e+03	-

Model 4 - Random Forest Regressor

Methodology:

This research employed regression model to analyze Boston housing datasets in order to predict the prices of
houses based on the features that are in the datasets. The fundamental step taken for the implementation include
data collection, data exploration which was used to understand the datasets and identify features in the dataset;
data pre-processing stage which was used to clean the dataset so as to make it suitable for model development.
Afterwards the model was developed using the proposed random forest algorithm

Data Collection and Exploration
In the development of the model, the UCI Machine learning repository Boston housing dataset was used. The

dataset was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes
from various suburbs in Boston, Massachusett dataset. Before constructing a regression model, exploratory data
analysis is needed. Researchers may uncover the data's underlying trends in this manner, which aids in the
selection of suitable machine learning approaches. Therefore, data exploration was carried out to understand the
features present in the dataset and their purpose. The features present in the dataset are: CRIM which is the per
capita crime rate by town, ZN which is the proportion of residential land zoned for lots over 25,000sq.ft, INDUS
which is the proportion of non-retail business acres per town, CHAS which is the Charles River dummy variable
(1 if tract bounds river, 0 otherwise), NOX which is nitric oxides concentration (parts per 10 million), RM is the
average number of rooms per dwelling, AGE signifies proportion of owner-occupied units built prior to 1940,
DIS is the weighted distances to five Boston employment centers, RAD is the index of accessibility to radial
highways, TAX is the full-value property-tax rate per $10,000, PTRATIO is the pupil-teacher ratio by town, B
1000(Bk - 0.63) ^2 where Bk is the proportion of blacks by town, LSTAT is the percentage of lower status of
the population and MEDV is the median value of owner-occupied homes in $1000's. Since the model uses a
supervised learning method, the dataset must be divided into the training dataset and testing dataset. For the
training dataset, 70% of the dataset was used to train the model while the remaining 30% was used for testing.

Data Pre-Processing

The data acquired for model training and testing should be analyzed appropriately before creating models sothat the models can learn the patterns more quickly. Numerical values were normalized, while categorical values were encoded one-at-a-time. After the exploration of the data and selecting the most suitable feature with the useof the heatmap, the next stage is the pre-processing of the data of the selected features that will be used. Typically, the datasets acquired for the training and testing task have several features. It is highly probable that the valuesof various features are on a different scale which may lower the performance of the model, therefore, scaling was carried out to ensure that the features are on a relatively similar scale.TheStandard Scaler function available inPhyton Skitlearn module was for this task. The Standard Scaler assumes that your data is naturallydistributedwithin each function and scales it so that it is nowclustered about 0 with a standard deviation of 1. The feature'smean and standard deviation are measured and then the feature is scaled based on:

$$\frac{x_i – \text{mean}(x)}{\text{stdev}(x)}$$

After scaling the features, a linear regression plot (regplot) was drawn to see the correlation between the featuresnd MEDV. This is to understand the dataset better since MEDV is the variable that will be forecasted.

Model Development
The proposed model was built using the random forest algorithm. The random forest was implemented usingthe RandomForestClassifier available in PhytonScikit-learn (sklearn) machine learning library. Random Forestis a popular supervised classification and regression machine learning technique.It employs the concept ofensemble learning to solve complex problems by incorporating several classifiers to improve the model'saccuracy. Random Forest is a classifier that averages the outcomes of multiple decision trees applied to varioussubsets of a dataset to improve the dataset's predictive accuracy. Rather than relying on a single decision tree, therandom forest uses the projections from each tree to determine the final performance based onthe majority ofvotes. The algorithm for the random forest is

Abigail Bola Adetunji et al. / Procedia Computer Science 199 (2022) 806–813 809
Abigail Bola Adetunji et al./ Procedia Computer Science 00 (2021) 000–000

i. Create an n-sample random bootstrap sample (by substitution, select n samples at random from thetraining set).
ii. At each node, build a decision tree using the bootstrap sample:
a. Select d functions at random without replacing them.
b. Divide the node using the attribute that offers the optimal split according to the objective function,such as optimizing knowledge gain in this case.
iii. Repeat steps 1-2 k times more.
iv. By combining the predictions from each tree, a majority vote is used to give the class name.Furthermore, the n_estimators parameter in the RandomForestClassifier helps us to choose how many trees
to create which we set at 500. The greater the number of trees in the forest, the more accurate it is, and the issueof overfitting is avoided. Although increasing the number of trees in the random forest enhances accuracy, it alsoincreases the model's average training time. The bootstrap parameter, which we set to True, is also included inthe class. Only a limited set of features will be used to introduce variation intorandom forest subsets, however.We improved the efficiency of the RandomForestClassifier by iterating themodel several times and adding a fewparameters when we initialized it.. Results of the Data Exploration ProcessTo understand the dataset better, data exploration was carried out. Fig. 1 show the distribution of the data in eachof the features in the datasets. It shows the total count of the data, the mean, the standard deviation, the minimumvalue, 25%,50%,75% and the maximum value. From this, two data columns show interesting summaries. ZN(proportion of suburban property zoned for lots above 25,000 sq. ft.), with 0 representing the 25th and 50thpercentiles. Second, with 0 forthe 25th, 50th, and 75th percentiles, CHAS: Charles River dummy vector (if ract borders river; 0 otherwise). Since both variables are conditional categorical, these summaries make sense.The first premise is that these columns will be useless in a regression task like forecasting MEDV (Median valueof owner-occupied homes).

Import Libraries: First, import the necessary libraries for working with data and creating the Random Forest Regressor model:

python

-
  ```
  import numpy as np
  ```
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

- **Load and Prepare Data**: Load your house price prediction dataset and split it into features (X) and the target variable (y). Also, split the data into training and testing sets:

python

-
  ```
  # Load your dataset, replace 'your_data.csv' with your dataset file
  ```
```
data = pd.read_csv('your_data.csv')

# Split data into features (X) and target variable (y)
X = data.drop('SalePrice', axis=1)
y = data['SalePrice']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- **Train the Random Forest Regressor**: Create a Random Forest Regressor model and train it on your training data:

python

-
  ```
  # Create a Random Forest Regressor model
  ```
```
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)  # You
can adjust the number of trees (n_estimators) and other hyperparameters

# Train the model
rf_regressor.fit(X_train, y_train)
```

- **Make Predictions**: Use the trained Random Forest Regressor model to make predictions on the test data:

python

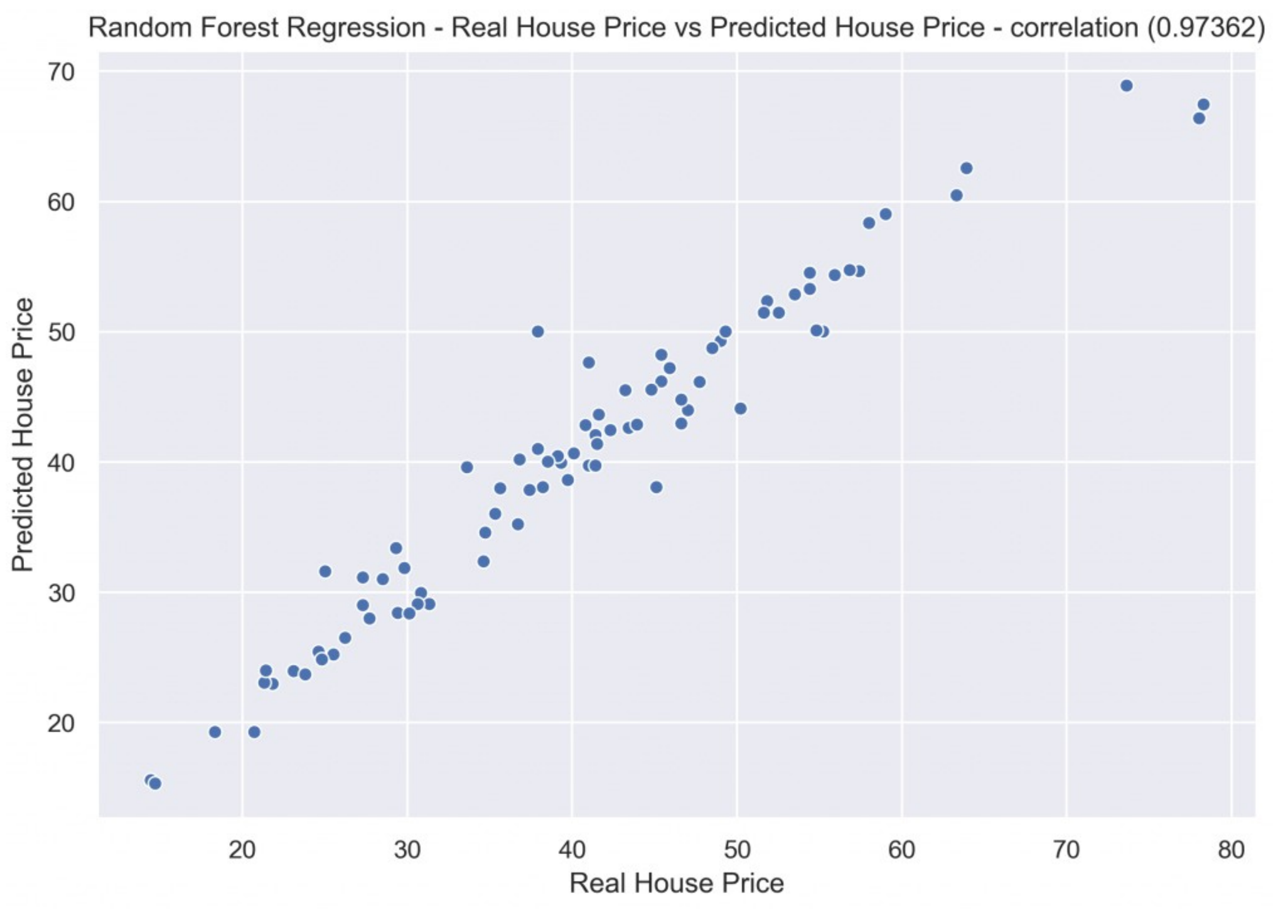
-
  ```
  y_pred = rf_regressor.predict(X_test)
  ```

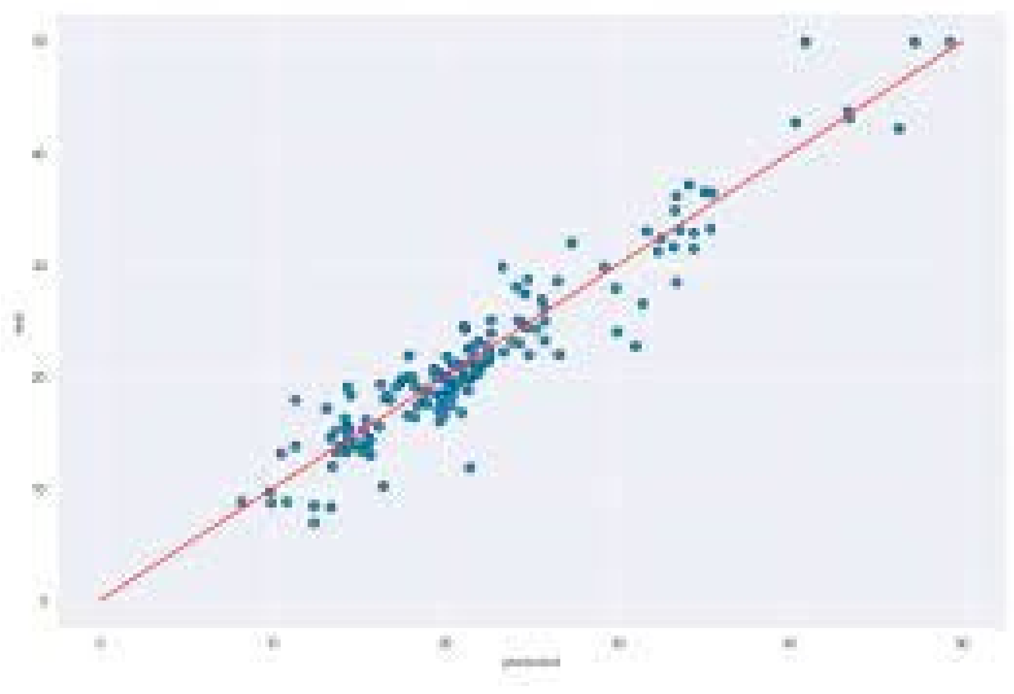
- **Evaluate the Model**: Assess the model's performance using appropriate regression metrics. Mean Squared Error (MSE) is a common metric for regression:

python

5.
   ```
   mse = mean_squared_error(y_test, y_pred)
   print(f"Mean Squared Error: {mse}")
   ```
6. **Feature Importance** (Optional): Random Forest models provide feature importance scores, which can help you understand which features have the most impact on predictions. You can access these scores using `rf_regressor.feature_importances_`.
7. **Hyperparameter Tuning** (Optional): You can fine-tune the model's hyperparameters (e.g., `n_estimators`, `max_depth`, `min_samples_split`, etc.) to optimize its performance. Grid search or randomized search are common approaches for hyperparameter tuning.

Random Forest Regressors are versatile and can handle both numeric and categorical features, making them a suitable choice for many house price prediction tasks. Remember to preprocess your data, handle missing values, and encode categorical variables as needed for your specific dataset.





Model 5 - XGboost Regressor

Extreme Gradient Boosting (XGBoost)

XGBoost is a scalable machine learning system for tree boosting. The system is available as an open-source pack-age. The system has generated a significant impact and been widely recognized in various machine learning and data mining challenges .The most crucial reason why XGBoost succeeds is its scalability in all scenarios. The system runs more than ten times faster than existing popular solutions on a single machine and scales to billions of examples in distributed or memory-limited settings. The scalability of XGBoost is due to several major systems and algorithmic optimizations including a novel tree learning algorithm for handling sparse data and a theoretically justified weighted quantile sketch procedure enabling instance weight handling in approximate tree learning. Parallel and distributed computing make learning faster, which allows quicker model exploration. More importantly, the model exploits out-of-core computa-tion and enables data scientists to process a hundred millions of examples on a desktop. Finally, after combining these techniques to make an end-to-end system, it can scale to even more extensive data with the least amount of cluster resources [12]. In this paper, we utilized the `XGBRegressor` from `xgboost` open-source package [13]. After tweaking the XGBoost model multiple times, we set our parameter to the following:

• Set `learning_rate` = `0.1`
• Set `n_estimators` = 200
• Determined the optimal tree specific parameters `min_child_weight` = 2, `subsample` = 1, `colsample_bytre`
= `0.8`
• Set reqularization parameter: `reg_lambda` = `0.45`, `reg_alpha` = 0, `gamma` =
`0.5`

The model performed with a high accuracy where the RMSLE of the training set is around `0.16118`. the Extreme Gradient Boosting prediction in X-axis, and the actual price in Y-axis for training data.

Extreme Gradient Boosting (XGBoost) is a powerful gradient boosting algorithm that is commonly used for regression tasks, including predicting house prices. XGBoost is known for its speed and high predictive performance, making it a popular choice for competitions and real-world machine learning projects. Here's how you can use XGBoost for predicting house prices:

1. **Import Libraries**: First, import the necessary libraries, including `numpy`, `pandas` for data manipulation, and `xgboost` for building the XGBoost model:

   python

```
• import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

- **Load and Prepare Data**: Load your house price prediction dataset and split it into features (X) and the target variable (y). Split the data into training and testing sets:

python

```
    • # Load your dataset, replace 'your_data.csv' with your dataset file
data = pd.read_csv('your_data.csv')

# Split data into features (X) and target variable (y)
X = data.drop('SalePrice', axis=1)
y = data['SalePrice']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

- **Train the XGBoost Model**: Create an XGBoost regression model and train it on your training data:

python

```
    • xgb_reg = xgb.XGBRegressor(
    objective='reg:squarederror',  # For regression tasks
    n_estimators=100,  # Number of boosting rounds
    learning_rate=0.1,  # Step size shrinkage
    max_depth=3,  # Maximum depth of the tree
)
xgb_reg.fit(X_train, y_train)
```

You can adjust the hyperparameters (e.g., `n_estimators`, `learning_rate`, `max_depth`) to fine-tune the model.

- **Make Predictions**: Use the trained XGBoost regression model to make predictions on the test data:

python

```
    • y_pred = xgb_reg.predict(X_test)
```

- **Evaluate the Model**: Measure the performance of your XGBoost model using appropriate regression metrics. A common metric for regression is Mean Squared Error (MSE):

python

```
    • mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

- **Feature Importance**: XGBoost provides a feature importance ranking that can help you understand which features are most influential in predicting house prices. You can access it as follows:

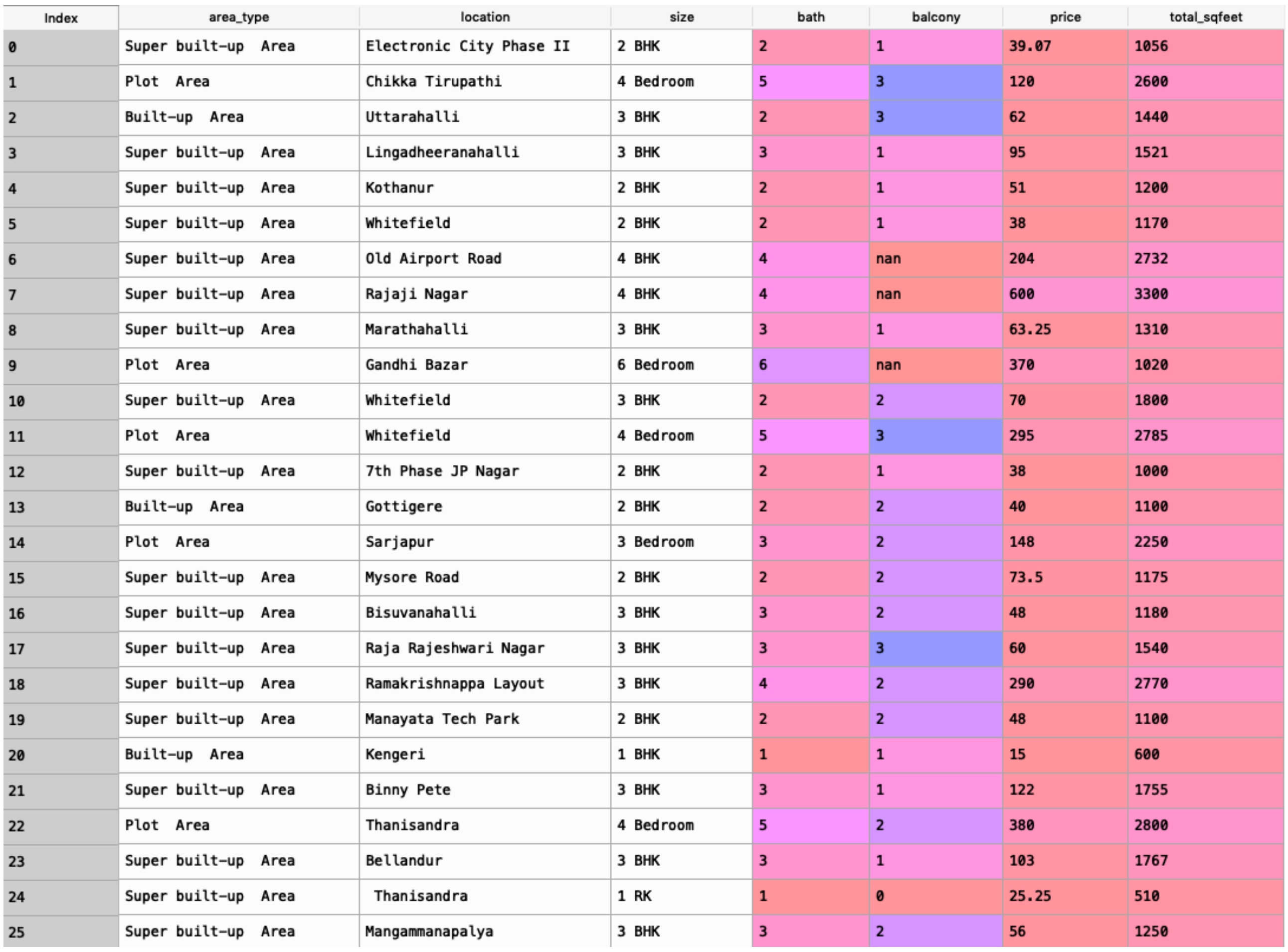
python

```
    6. feature_importance = xgb_reg.feature_importances_
```

7. **Tune Hyperparameters** (Optional): You can further optimize your XGBoost model by experimenting with various hyperparameters, such as `learning_rate`, `max_depth`,

`min_child_weight`, and so on. Grid search or random search can be used to automate the hyperparameter tuning process.

XGBoost is a versatile algorithm, and with careful hyperparameter tuning and feature engineering, it can deliver state-of-the-art results in house price prediction tasks. Additionally, you can consider using cross-validation techniques to assess the model's generalization performance and ensure it doesn't overfit to the training data.



Index	area_type	location	size	bath	balcony	price	total_sqfeet
0	Super built-up Area	Electronic City Phase II	2 BHK	2	1	39.07	1056
1	Plot Area	Chikka Tirupathi	4 Bedroom	5	3	120	2600
2	Built-up Area	Uttarahalli	3 BHK	2	3	62	1440
3	Super built-up Area	Lingadheeranahalli	3 BHK	3	1	95	1521
4	Super built-up Area	Kothanur	2 BHK	2	1	51	1200
5	Super built-up Area	Whitefield	2 BHK	2	1	38	1170
6	Super built-up Area	Old Airport Road	4 BHK	4	nan	204	2732
7	Super built-up Area	Rajaji Nagar	4 BHK	4	nan	600	3300
8	Super built-up Area	Marathahalli	3 BHK	3	1	63.25	1310
9	Plot Area	Gandhi Bazar	6 Bedroom	6	nan	370	1020
10	Super built-up Area	Whitefield	3 BHK	2	2	70	1800
11	Plot Area	Whitefield	4 Bedroom	5	3	295	2785
12	Super built-up Area	7th Phase JP Nagar	2 BHK	2	1	38	1000
13	Built-up Area	Gottigere	2 BHK	2	2	40	1100
14	Plot Area	Sarjapur	3 Bedroom	3	2	148	2250
15	Super built-up Area	Mysore Road	2 BHK	2	2	73.5	1175
16	Super built-up Area	Bisuvanahalli	3 BHK	3	2	48	1180
17	Super built-up Area	Raja Rajeshwari Nagar	3 BHK	3	3	60	1540
18	Super built-up Area	Ramakrishnappa Layout	3 BHK	4	2	290	2770
19	Super built-up Area	Manayata Tech Park	2 BHK	2	2	48	1100
20	Built-up Area	Kengeri	1 BHK	1	1	15	600
21	Super built-up Area	Binny Pete	3 BHK	3	1	122	1755
22	Plot Area	Thanisandra	4 Bedroom	5	2	380	2800
23	Super built-up Area	Bellandur	3 BHK	3	1	103	1767
24	Super built-up Area	Thanisandra	1 RK	1	0	25.25	510
25	Super built-up Area	Mangammanapalya	3 BHK	3	2	56	1250

Conclusion and Future Work (Phase 2):

Project Conclusion:

In the Phase 2 conclusion, we will summarize the key findings and insights from the advanced regression techniques. We will reiterate the impact of these techniques on improving the accuracy and robustness of house price predictions.

Future Work: We will discuss potential avenues for future work, such as incorporating additional data sources (e.g., real-time economic indicators), exploring deep learning models for prediction, or expanding the project into a web application with more features and interactivity.

THANK YOU!!!