MBA Semester – IV
Research Project

Name	Mahesh Narayan G
USN	222VMBR02617
Elective	Data Science and Analytics
Date of Submission	15-09-2024

A study on "**Beyond the Square Footage: Unveiling the Secrets of House Price Prediction using Machine Learning**"

Research Project submitted to Jain Online (Deemed-to-be University)

In partial fulfillment of the requirements for the award of:

Master of Business Administration

Submitted by:

Mahesh Narayan G

USN:

222VMBR02617

Under the guidance of:

Sharath Srivatsa

(Faculty-JAIN Online)

Jain Online (Deemed-to-be University)

Bangalore

2023-24

DECLARATION

I, *Mahesh Narayan G,* hereby declare that the Research Project Report titled *"Beyond the Square Footage: Unveiling the Secrets of House Price Prediction using Machine Learning" has been* prepared by me under the guidance of the *Sharath Srivatsa.* I declare that this Project work is towards the partial fulfillment of the University Regulations for the award of the degree of Master of Business Administration by Jain University, Bengaluru. I have undergone a project for a period of Eight Weeks. I further declare that this Project is based on the original study undertaken by me and has not been submitted for the award of any degree/diploma from any other University / Institution.

Place: Bangalore — *Mahesh Narayan G*

Date: 15/09/2024 — *USN:222VMBR02617*

CERTIFICATE

This is to certify that the Research Project report submitted by Mr. *Mahesh Narayan G* bearing *(222VMBR02617)* on the title *"Beyond the Square Footage: Unveiling the Secrets of House Price Prediction using Machine Learning Title of the project"* is a record of project work done by him during the academic year 2023-24 under my guidance and supervision in partial fulfillment of Master of Business Administration.

Place: Bangalore

Date: 15/09/2024 *USN:222VMBR02617*

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project guide, **Mr. Sharat Srivatsa**, and program manager, Mr. Agneev, for their invaluable guidance and support throughout this project. I also extend my thanks to Jain University for providing the resources and environment necessary for the successful completion of this research.

Mahesh Narayan G
USN:222VMBR02617

EXECUTIVE SUMMARY

This project focuses on predicting house prices by analysing various features that influence a house's value such as location, size, area, condition, etc. Our model aims to provide accurate predictions of house prices in the current market based on their specific features. This will help the sellers, buyers, real estate agents make informed decisions regarding the sale prices. This project addresses the challenge of accurately predicting house prices by considering factors that goes beyond just location and square footage.

This project begins with the stage of understanding the data, identifying and addressing errors, or other data quality issues in the dataset. The goal is to ensure the given data is accurate, consistent, and ready for analysis. Then follow exploratory data analysis (EDA) process to find patterns to make predictions. Then, select the appropriate models and interpret the results to get accurate house prices.

As the real estate market fluctuates, accurately pricing a house is crucial for sellers and buyers. This project focuses on predicting house prices by analyzing various features that influence a house's value such as location, size, area, condition, etc. Our model aims to provide accurate predictions of house prices in the current market based on their specific features. This will help the sellers, buyers, real estate agents make informed decisions regarding the sale prices. This project addresses the challenge of accurately predicting house prices by considering factors that goes beyond just location and square footage.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION AND BACKGROUND

INTRODUCTION AND BACKGROUND

1.1 Purpose of the Study

The purpose of this study is to develop an accurate machine learning model for predicting house prices by analyzing various features that influence a property's value. This research aims to provide valuable insights for homeowners, potential buyers, and real estate professionals, enabling informed decision-making in a fluctuating market. The model aims to go beyond just considering location and square footage to take a more holistic approach to price prediction. Specifically, the study seeks to:

- Identify key factors influencing house prices in the current market.
- Develop a predictive model that can estimate house prices based on multiple variables.
- Provide insights to help stakeholders make informed decisions about property valuation.
- Create a data-driven tool to assist sellers in pricing properties appropriately and buyers in making informed purchasing decisions.

1.2 Introduction to the Topic

As the real estate market fluctuates, accurately pricing a house is crucial for sellers, buyers, and real estate professionals. However, determining the appropriate price for a property is a complex process that requires analyzing multiple factors beyond just location and square footage. This project focuses on leveraging machine learning techniques to predict house prices by considering a comprehensive set of features that contribute to a property's value. The study utilizes a dataset containing various attributes of houses, including physical characteristics, location data, and historical information. By applying advanced data analysis and machine learning algorithms, the project aims to uncover patterns and relationships in the data that can inform more accurate price predictions.

The real estate market is characterized by its complexity and variability. Accurate house price prediction is essential for stakeholders, including buyers, sellers, and investors. This study explores the multifaceted nature of house pricing, considering elements beyond traditional metrics like square footage and location.

1.3 Overview of Theoretical Concepts

This study draws on several key theoretical concepts and techniques from data science and machine learning:

- Exploratory Data Analysis (EDA): Techniques for visualizing and summarizing data to identify patterns, relationships, and anomalies.
- Feature Engineering: The process of creating new variables or modifying existing ones into one in order to improve model performance.
- Machine Learning Algorithms: Various regression models will be explored, including linear regression, decision trees, random forests, and XG boosting methods.
- Model Evaluation: Metrics such as Mean Absolute Percentage Error (MAPE), Root Mean Squared Error (RMSE), R-squared and Adjusted R-squared will be used to assess model performance.
- Hyperparameter Tuning: Techniques for optimizing model parameters to improve predictive accuracy.

1.4 Company/ Domain / Vertical /Industry Overview

This study focuses on the residential real estate industry, which plays a crucial role in the broader economy. The real estate market is influenced by various factors, including economic conditions, demographic trends, government policies, and local market dynamics.

Accurate property valuation is essential for multiple stakeholders in this industry:

- Homeowners and sellers seeking to price their properties competitively.
- Buyers looking to make informed purchasing decisions.
- Real estate agents advising clients on pricing and offers.
- Financial institutions assessing property values for mortgage lending.

By providing more accurate and data-driven price predictions, this study aims to bring greater transparency and efficiency to the housing market.

1.5 Environmental Analysis (PESTEL Analysis)

Conducting a PESTEL analysis to examine the external factors affecting the real estate market:

- Political: Regulations on houses, zoning laws, and government policies.
- Economic: Market trends, interest rates, and economic indicators affecting housing demand.
- Social: Demographic shifts, lifestyle changes, and consumer preferences in housing.
- Technological: Innovations in data analytics, machine learning, and real estate platforms.
- Environmental: Sustainability practices, climate change impacts on property values.
- Legal: Legal frameworks governing property transactions and rights.

CHAPTER 2

REVIEW OF LITERATURE

REVIEW OF LITERATURE

2.1 Domain/ Topic Specific Review

a) Traditional Valuation Methods:

- Comparable Sales Approach: Estimating a property's value based on recent sales of similar properties in the area.
- Cost Approach: Calculating the cost to replace the property plus the value of the land.
- Income Approach: Used for rental properties, based on potential income generation.

b) Statistical and Machine Learning Approaches:

- Linear Regression: A fundamental approach for modelling the relationship between house features and prices.
- Decision Trees and Random Forests: Non-linear models capable of capturing complex relationships in housing data.
- Gradient Boosting Methods (e.g., XGBoost): Advanced ensemble techniques known for high predictive accuracy.

c) Feature Importance in Real Estate:

- Location Factors: Proximity to amenities, school districts, crime rates.
- Property Characteristics: Size, number of rooms, age, condition.
- Market Trends: Historical price data, seasonal fluctuations.

d) Data Pre-processing Techniques:

- Handling Missing Values: Various imputation methods.
- Outlier Detection & Treatment: Identifying and addressing extreme values.
- Feature Engineering: Creating new variables to capture complex relationships.

e) Model Evaluation in Real Estate Prediction:

- Use of metrics like MAPE, RMSE, and R-squared to assess model performance.
- Cross-validation techniques to ensure model generalizability.

f) Advanced Techniques:

- Geospatial analysis to capture location-based effects.
- Time series analysis for modelling market trends.
- Ensemble methods combining multiple models for improved accuracy.

2.2 Gap Analysis

1. Holistic Feature Analysis: While many studies focus on a limited set of features, this project aims to consider a comprehensive set of variables that influence house prices, going beyond just location and square footage.
2. Comparative Model Analysis: By implementing and comparing multiple machine learning models (linear regression, Lasso, Ridge, Random Forest, Decision Tree, XGBoost), the study provides insights into the relative performance of different algorithms for this specific problem.
3. Optimization Techniques: The use of Recursive Feature Elimination (RFE) and hyperparameter tuning demonstrates an effort to optimize model performance beyond basic implementations.
4. Practical Application: The focus on creating a tool that can be used by various stakeholders in the real estate industry suggests an emphasis on bridging the gap between academic research and practical application.
5. Market Segmentation: The use of clustering techniques (K-means) to identify distinct segments in the housing market represents an attempt to provide more nuanced insights beyond a one-size-fits-all prediction model.
6. Interpretability: While achieving high predictive accuracy, the study also aims to provide interpretable insights into feature importance and market dynamics, addressing the often-cited trade-off between model complexity and interpretability.

By addressing these gaps, the study aims to contribute to both the theoretical understanding of house price prediction and the practical application of machine learning techniques in the real estate industry.

CHAPTER 3

RESEARCH METHODOLOGY

RESEARCH METHODOLOGY

3.1 Objectives of the Study

The primary objectives of this study are:

1. To develop a machine learning model that accurately predicts house prices based on multiple features.
2. To identify and analyze key factors influencing house prices beyond just location and square footage.
3. To provide actionable insights for stakeholders in the real estate industry, including sellers, buyers, and real estate professionals.
4. To compare the performance of various machine learning algorithms in house price prediction.
5. To create a data-driven tool that can assist in property valuation and decision-making.

3.2 Scope of the Study

The scope of this study includes:

1. Analysis of a dataset containing various features of houses, including physical characteristics, location data, and historical information.
2. Development and comparison of multiple machine learning models for house price prediction.
3. Focus on residential properties within a specific geographic area (as indicated by the presence of zipcode data in the dataset).
4. Examination of both quantitative factors (e.g., square footage, number of rooms) and qualitative factors (e.g., condition, quality) influencing house prices.
5. Exploration of market segmentation using clustering techniques.
6. To develop and validate machine learning models for accurate price prediction.

3.3 Methodology

- **Research Design**

This study employs a quantitative research design using secondary data analysis and predictive modelling. The research process involves:

1. Exploratory Data Analysis (EDA) to understand the dataset and relationships between variables.
2. Data pre-processing, including handling missing values and outlier treatment.
3. Feature engineering to create new variables and select relevant features.
4. Development and comparison of multiple machine learning models and choose the best model.
5. Model evaluation and optimization using various performance metrics.

- **Data Collection**

This study uses secondary data from an existing dataset containing information on house features and prices. The dataset includes 21,613 records with 23 variables, covering attributes such as:

1. Physical characteristics (e.g., no. of bedrooms, bathrooms, square footage)
2. Location information (zipcode, latitude, longitude)
3. Historical data (year built, year renovated)
4. Quality and condition ratings as well.

- **Data Analysis Tools**

This study utilizes various data analysis tools and techniques, including:

1. Python programming language with libraries such as pandas, numpy, and scikit-learn.
2. Exploratory Data Analysis techniques (univariate, bivariate, and multivariate analysis).
3. Machine learning algorithms (Linear Regression, Lasso, Ridge, Random Forest, Decision Tree, XGBoost).

4. K-means clustering for market segmentation.
5. Feature selection techniques (Recursive Feature Elimination).
6. Hyperparameter tuning using ‘RandomizedSearchCV’.
7. Model evaluation metrics (MAPE, RMSE, R-squared, Adjusted R-squared).

3.4 Period of Study

The period of this project taken is for a period of eight weeks in order to conduct the research on the given dataset, EDA process, clean the data, clustering, building models and interpret on the best models.

3.5 Limitations of the Study

This study has limitations while doing the project such as:

1. Geographic specificity: The model was limited to the area covered by the dataset and may not generalize well to other regions.
2. Time sensitivity: Real estate markets can change rapidly, potentially affecting the model's long-term accuracy.
3. Limited feature set: The given dataset may not capture all factors influencing house prices (e.g., local economic conditions, future development plans).
4. Reliance on historical data: The model may not account for sudden market shifts or unprecedented events.
5. Data Availability: Constraints due to incomplete or missing data.
6. Model Limitations: Potential biases in model predictions or overfitting.
7. External Factors: Unforeseen market changes or economic events that may influence results.

3.6 Utility of Research

This research has several potential applications and benefits:

1. Assisting homeowners and sellers in setting appropriate prices for their properties.
2. Helping potential buyers understand fair market values and make informed decisions.

3. Providing real estate professionals with a data-driven tool for property valuation.
4. Demonstrating the application of advanced machine learning techniques in real estate, potentially inspiring further research and development in this field.
5. Improving market transparency and efficiency by providing more accurate and objective price estimates.

CHAPTER 4

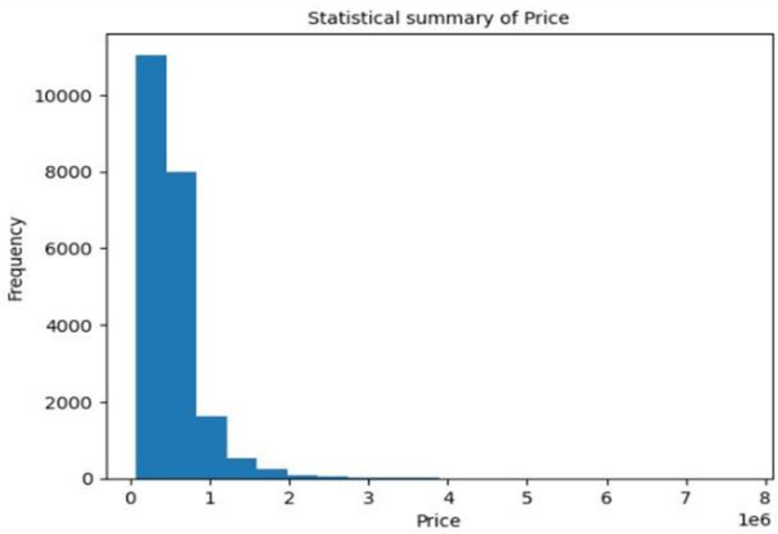
DATA ANALYSIS AND INTERPRETATION

DATA ANALYSIS AND INTERPRETATION

4.1 Univariate Analysis

This is the simplest form of data analysis, where the data being analyzed consists of just one variable, in our cause it's 'price' column. Since it's a single variable, it doesn't deal with causes or relationships. The main purpose of univariate analysis is to describe the data and find patterns that exist within it.

```
#univariate (histogram)
plt.hist(df['price'],bins=20)
plt.title('Statistical summary of Price',fontsize=10)
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

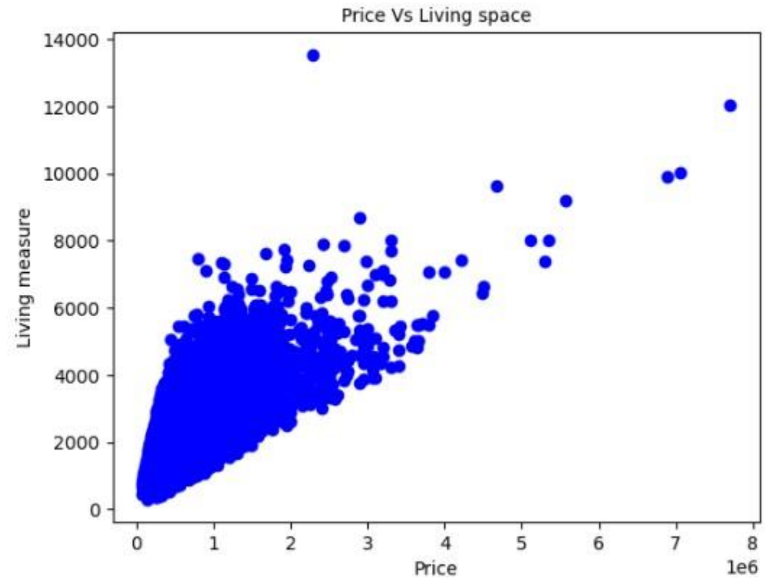


Univariate analysis in house price prediction, with chosen attribute like 'price' because by price is independent each other. Based on the below figure, the right-skewed distribution suggests the presence of outliers, which are the high-priced properties that contribute to the long tail.

4.2 Bivariate Analysis

Bivariate analysis involves looking at two variables at a time. Bivariate in EDA can help us understand the relationship between two variables and identify any patterns that might exist.

```
#bivariate for price vs living space
plt.scatter(x=df['price'],y=df['living_measure'],color='blue')
plt.title('Price Vs Living space',fontsize=10)
plt.xlabel('Price')
plt.ylabel('Living measure')
plt.show()
```

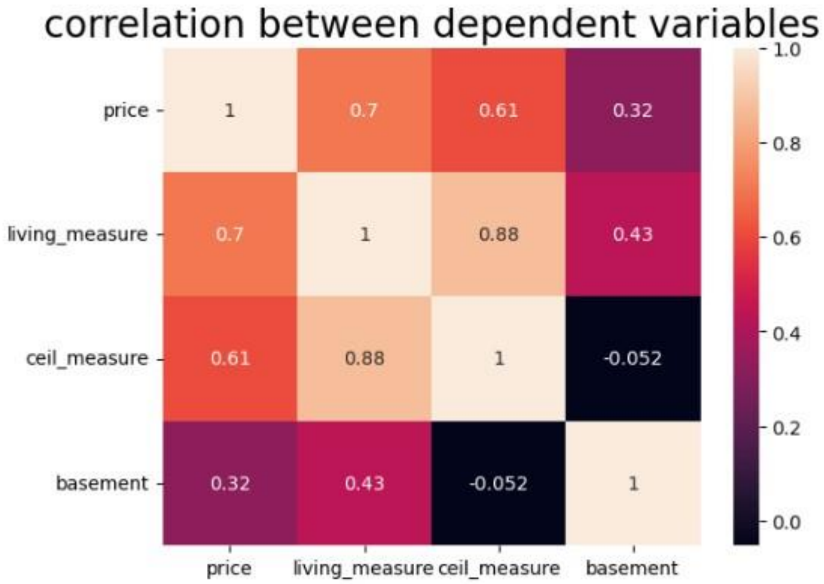


Bi-variate analysis in house price prediction, with chosen attributes like 'price' and 'living_measure'. Because by 'living_measure', price is calculated so these two variables are dependent to each other. Based on the below figure, as the living space increases, the price tends to increase as well. This suggests a positive correlation between the two variables. Also, a few data points appear to be somewhat distant from the main cluster which can be outliers.

4.3 Multivariate Analysis

Multivariate analysis is used to display relationships between three or more variables at a time. Multivariate analysis in EDA can help us understand the relationships between several variables and identify any complex patterns or outliers that might exist.

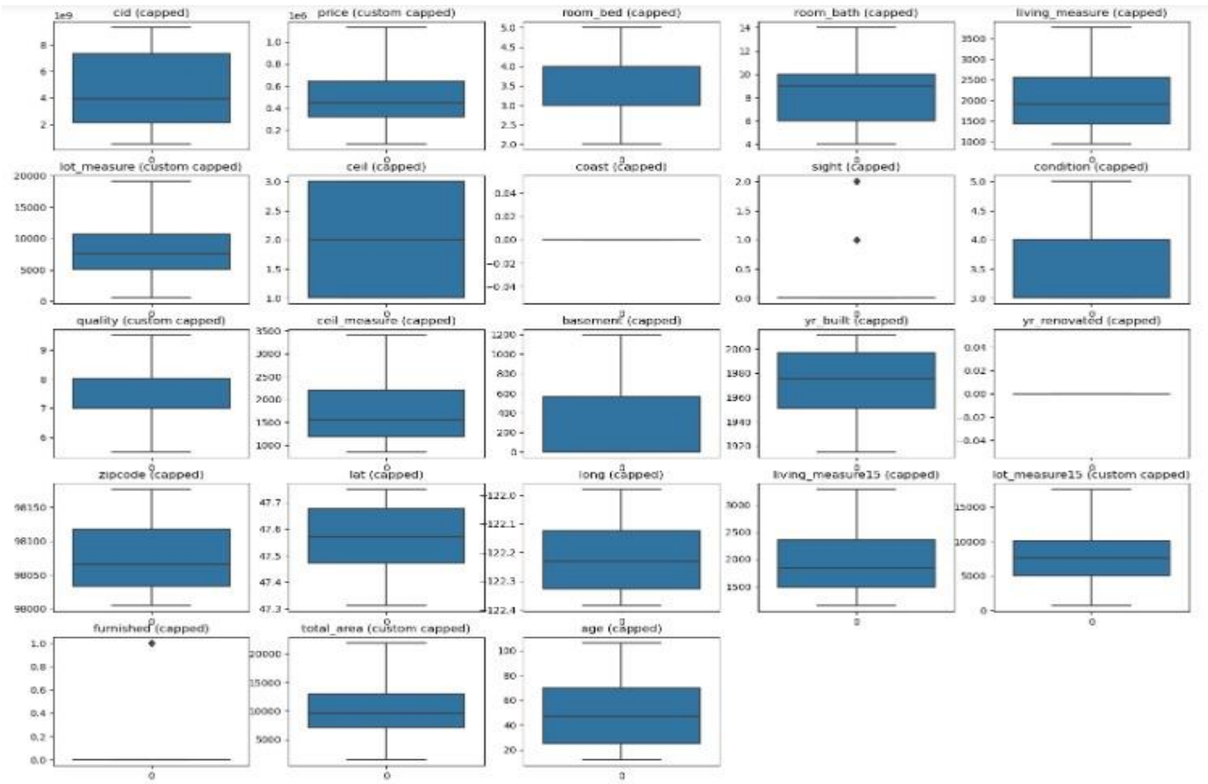
```
#multivariate for price, living measure, ceil measure and basement
correlation_matrix=df[['price','living_measure','ceil_measure','basement']].corr()
sns.heatmap(correlation_matrix,annot=True)
plt.title('correlation between dependent variables',fontsize=20)
plt.show()
```



Multi-variate analysis in house price prediction, with chosen attributes like ‘price’, ‘living_measure’, ‘ceil_measure’, ‘basement’ because ceil_measure and basement will calculate living_measure and by living_measure, price is calculated so these four variables are dependent to each other. Based on the figure below, there is a very strong positive correlation (0.88) between living_measure and ceil_measure. A moderate positive correlation (0.61) between price and ceil_measure. The correlation between price and basement is relatively weak (0.32).

4.3 Outlier Treatment

As we have identified outliers in the above analysis, now we are going to calculate IQR and then do capping methods to remove those outliers and name those columns as ‘capped’. For those columns which still have outliers present after capping, we’ll do ‘custom capping’ using their appropriate 25th and 75th percentile value.



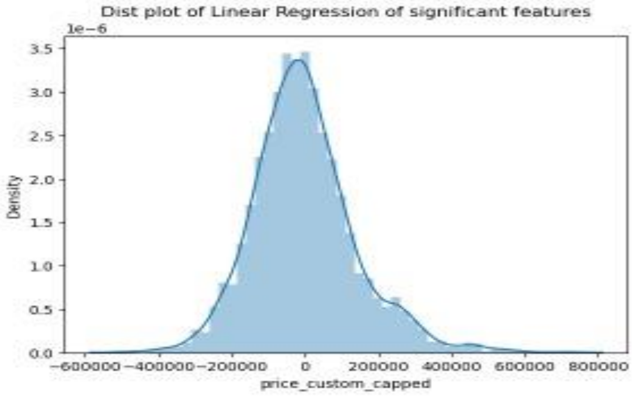
As soon as we did 'custom capping' based on the appropriate 25th and 75th percentile value of certain columns like 'lot_measure' and 'total_area' which still had outliers after capping. In the above figure, we can see all the outliers have been removed in the above box plots after using IQR, capping and custom capping method. Hence, the outlier treatment has been successful.

4.4 Linear Regression Model

Linear regression model shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

```
#Linear Regression Model

from sklearn.linear_model import LinearRegression
from sklearn import metrics

model_lr = LinearRegression()

model_lr.fit(X_train_significant,y_train)
y_pred = model_lr.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of Linear Regression of significant features')
plt.show()
```

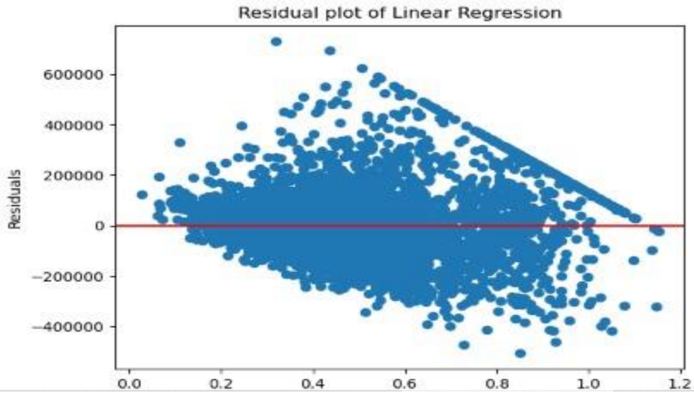
```
C:\Users\User\AppData\Local\Temp\ipykernel_6048\2020468942.py:11: UserWarning:

`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((y_test-y_pred), bins=50)
```




```
mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.23027840859486132
RMSE: 140366.08372466467
R-squared: 0.6769380367254575
Adjusted R-squared: 0.6763389417541557
```

```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of Linear Regression")
plt.show()
```



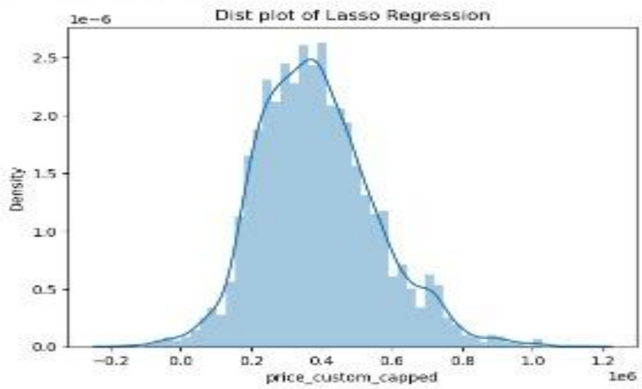
In the above dis plot and residual plot for linear regression model, the distribution of residuals (difference between actual and predicted values) is centered around zero and is bell-shaped, which suggests that the model's errors are normally distributed. This is a good sign in linear regression as it indicates that the model's predictions are unbiased.

4.5 Lasso Regression Model

Lasso stands for Least Absolute Shrinkage and Selection Operator (LASSO) is a technique where data points are shrunk towards a central point, like the mean. Lasso is also known as L1 regularization.

```
#Lasso Regression Model

from sklearn.linear_model import Lasso
from sklearn import metrics

model_lm = Lasso(alpha=1)

model_lm.fit(X_train_significant,y_train)
y_pred = model_lm.predict(X_test_significant.astype(int))

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of Lasso Regression')
plt.show()
```

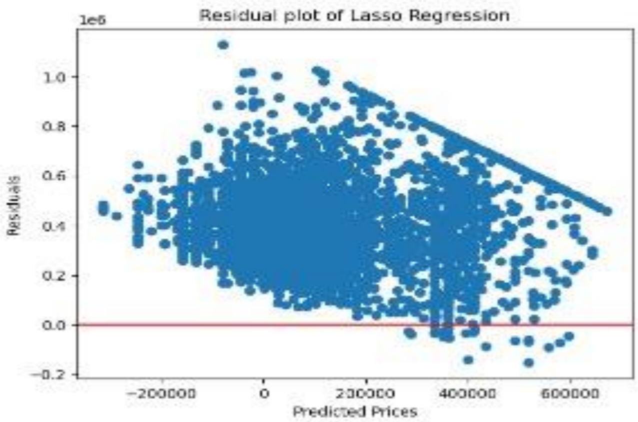
```
C:\Users\User\AppData\Local\Temp\ipykernel_6048\69215358.py:11: UserWarning:

`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((y_test-y_pred), bins=50)
```



```
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.8450066824897876
RMSE: 421765.6180773643
R-squared: -1.9167807999805144
Adjusted R-squared: -1.9221897583485825
```

```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of Lasso Regression")
plt.show()
```



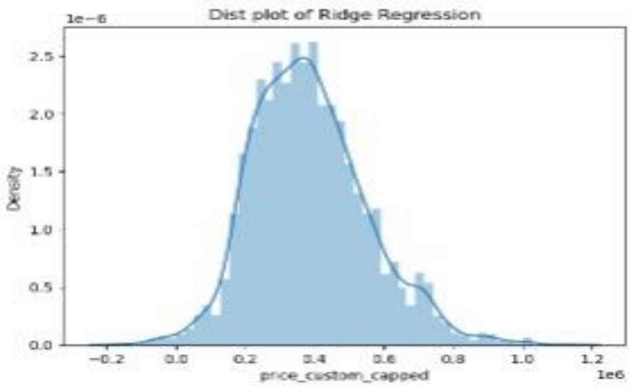
In the above dis plot and residual plot for lasso regression model, the distribution is still centered around zero, which indicates that the Lasso model is attempting to predict values without significant bias. However, compared to the Linear Regression plot, the shape here looks more skewed and less symmetrical.

4.6 Ridge Regression Model

Ridge regression is a technique used to analyze multi-linear regression (multi-collinear), also known as L2 regularization.

```
#Ridge Regression Model

from sklearn.linear_model import Ridge
from sklearn import metrics

model_rm = Ridge()

model_rm.fit(X_train_significant,y_train)
y_pred = model_rm.predict(X_test_significant.astype(int))

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of Ridge Regression')
plt.show()
```

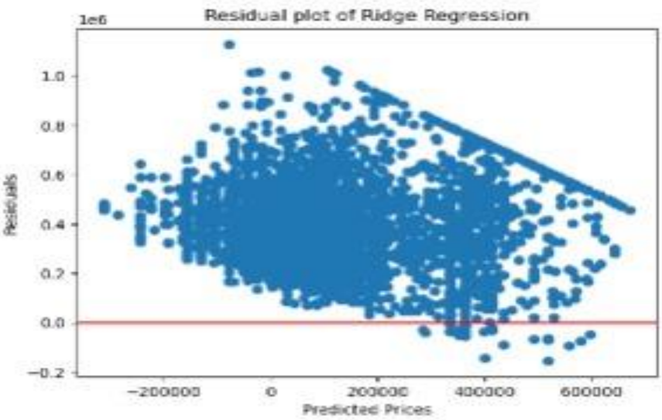
```
C:\Users\User\AppData\Local\Temp\ipykernel_6848\931011914.py:11: UserWarning:

`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((y_test-y_pred), bins=50)
```




```
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.8422262491371693
RMSE: 428720.69196262176
R-squared: -1.9823461396592616
Adjusted R-squared: -1.9877283299970627
```

```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of Ridge Regression")
plt.show()
```



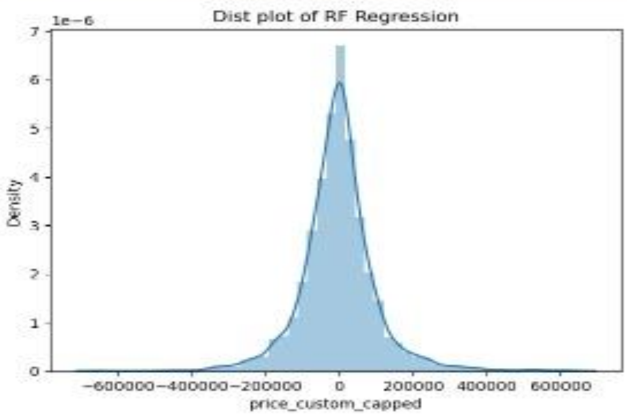
In the above dis plot and residual plot for ridge regression model, the distribution appears to be approximately normal, which is a good sign as it indicates that the model's residuals are randomly distributed around zero. This suggests that the model is capturing the underlying relationship between the predictor variables and the target variable reasonably well.

4.7 Random Forest Regression Model

Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees. Random Forest has multiple decision trees as base learning models. Randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model.

```
#Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

model_RFR = RandomForestRegressor(n_estimators=10)
model_RFR.fit(X_train_significant, y_train)
y_pred = model_RFR.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of RF Regression')
plt.show()
```

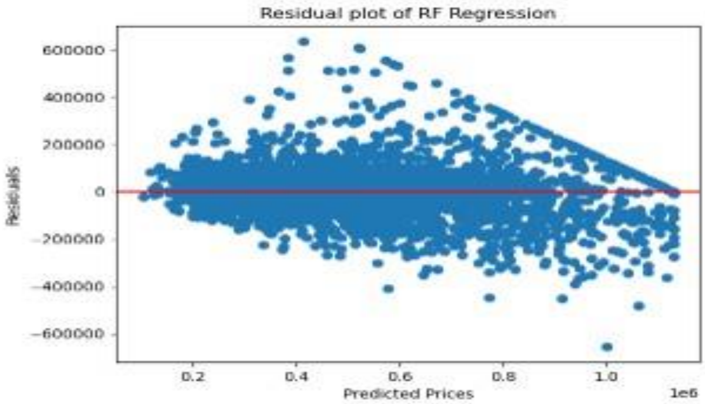
```
C:\Users\User\AppData\Local\Temp\ipykernel_6048\1512409299.py:9: UserWarning:

`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((y_test-y_pred), bins=50)
```



```python
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.15073104978600172
RMSE: 108941.30968657837
R-squared: 0.8329296692371515
Adjusted R-squared: 0.8326198494304518
```

```python
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of RF Regression")
plt.show()
```



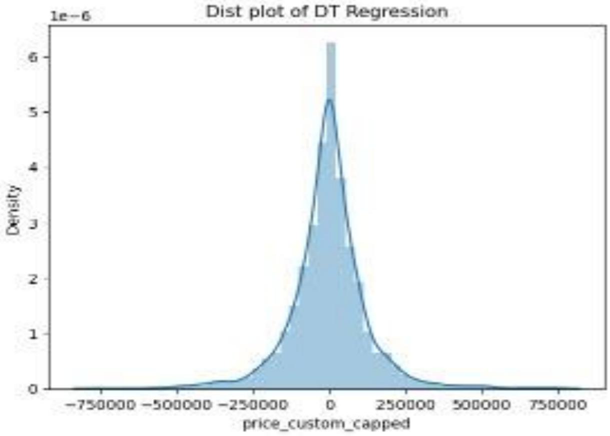
In the above dis plot and residual plot for random forest regression model, the distribution appears to be roughly bell-shaped and symmetrical, suggesting the model's predictions are normally distributed around the actual house prices. On average, the model's predictions are not systematically biased high or low.

4.8 Decision Tree Regression Model

Decision tree regression is a machine learning algorithm that constructs a tree-like model to predict a continuous outcome. It's a non-parametric method that can handle both linear and non-linear relationships.

```
#Decision Tree Regressor

from sklearn.tree import DecisionTreeRegressor

model_DTR = DecisionTreeRegressor()
model_DTR.fit(X_train_significant, y_train)
y_pred = model_DTR.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of DT Regression')
plt.show()
```

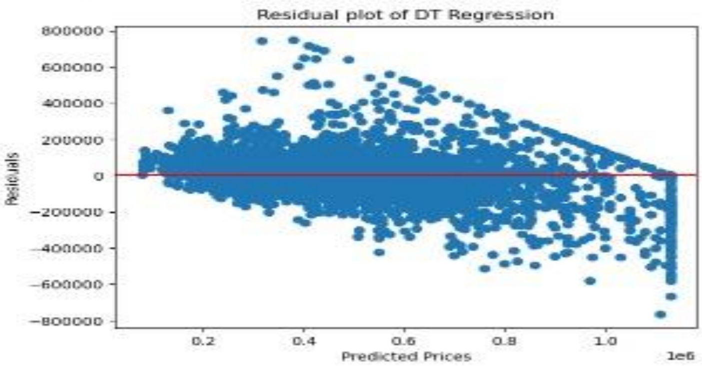
```
C:\Users\User\AppData\Local\Temp\ipykernel_6048\2087142571.py:9: UserWarning:

`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((y_test-y_pred), bins=50)
```




```
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.18351087196009343
RMSE: 126462.55114994255
R-squared: 0.7377682181869953
Adjusted R-squared: 0.7372819203588767
```

```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of DT Regression")
plt.show()
```



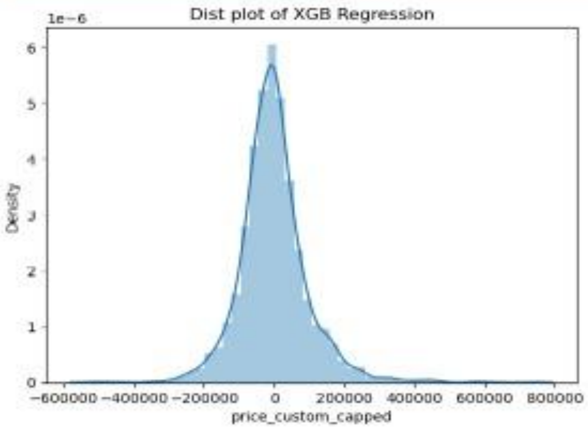
In the above dis plot and residual plot for decision tree regression model, the distribution has a sharp peak centred near zero, indicating that for many houses, the model's predictions are very close to the actual prices. The distribution also has longer and slightly thicker tails compared to the Random Forest model.

4.9 XGBoost Regression Model

XGBoost (Extreme Gradient Boosting) is a gradient boosting framework that is optimized for speed and performance. It's a popular choice for many machine learning tasks, including regression.

```
#XG Boost Regressor

from xgboost import XGBRegressor

model_XGB = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42)
model_XGB.fit(X_train_significant, y_train)

y_pred = model_XGB.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of XGB Regression')
plt.show()
```

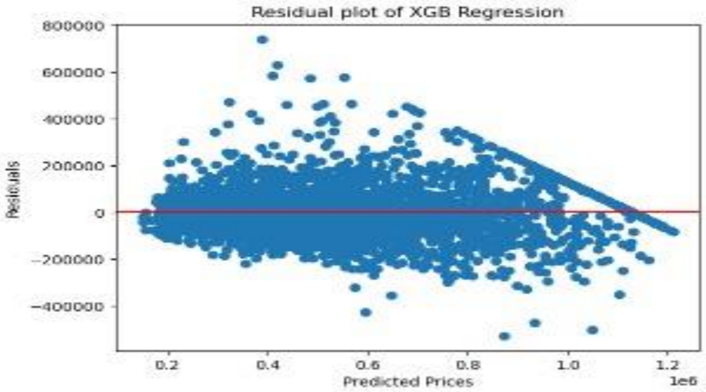
```
C:\Users\User\AppData\Local\Temp\ipykernel_6048\2421365405.py:10: UserWarning:

`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((y_test-y_pred), bins=50)
```



```python
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.1479936729462726
RMSE: 96437.81560711651
R-squared: 0.8475847938990412
Adjusted R-squared: 0.8472228826035364
```

```python
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of XGB Regression")
plt.show()
```



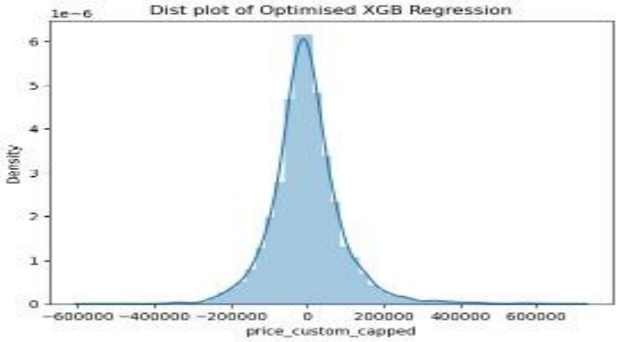
In the above dis plot and residual plot for XGBoost regression model, the distribution has a tall, narrow peak centred very close to zero, indicating that the XGBoost model makes highly accurate predictions for a large number of houses. The peak is taller and narrower compared to both the Random Forest and Decision Tree models, suggesting XGBoost achieves higher precision for a larger portion of the dataset.

4.10 Optimised/Tuned XGBoost Regression Model

We've used 'randomized search CV' for fine-tuning the best non-parametric model in the below figure and get the best parameters for more optimized model. After acquiring the best parameters, we use those specific parameters for XG Boost model again and get better plot and consider it as optimised or tuned model.

```
#XG Boost Regressor with best parameters

from xgboost import XGBRegressor

#using the optimized parameters
model_XGB_optimised = XGBRegressor(
      n_estimators=300,
      max_depth=7,
      learning_rate=0.05,
      subsample=0.6,
      colsample_bytree=0.7
)

model_XGB_optimised.fit(X_train_significant, y_train)

y_pred_optimised = model_XGB_optimised.predict(X_test_significant)

sns.distplot((y_test-y_pred_optimised), bins=50)
plt.title('Dist plot of Optimised XGB Regression')
plt.show()
```

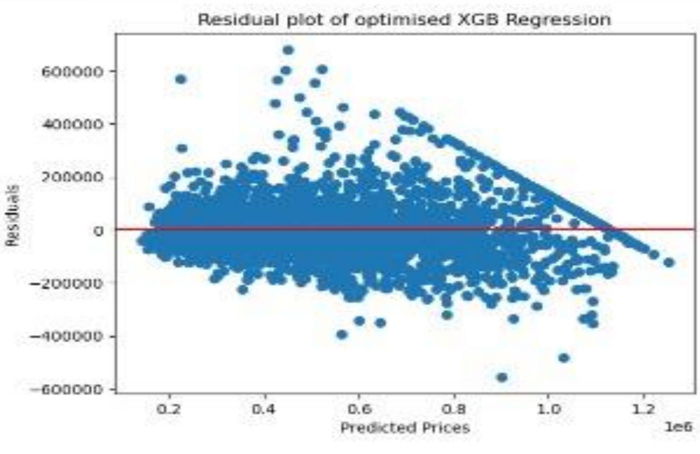
```
C:\Users\User\AppData\Local\Temp\ipykernel_6048\4085758626.py:18: UserWarning:

`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((y_test-y_pred_optimised), bins=50)
```




```
# Calculating metrics for optimised models
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred_optimised)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred_optimised))
r_squared = metrics.r2_score(y_test, y_pred_optimised)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.14110298511181235
RMSE: 92608.10262103587
R-squared: 0.859376004873238
Adjusted R-squared: 0.8591152270756919
```

```
# Residual plot
residuals = y_test - y_pred_optimised
plt.scatter(y_pred_optimised, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of optimised XGB Regression")
plt.show()
```



In the above dis plot and residual plot for XGBoost regression model, the distribution has an even taller and narrower peak compared to the original XGBoost model, centred very close to zero. This indicates that the optimized model makes even more accurate predictions for a larger number of houses. The peak's height is significantly increased, suggesting a higher concentration of predictions very close to the actual prices.

The tails of the distribution are slightly shorter and thinner compared to the original XGBoost model, indicating that the frequency and magnitude of errors have been further reduced.

4.11 Model Evaluation using Metrics Score

As we've already built the models and got the metrics score, now we're going to compare each metrics score and explain which metrics we've compared.

1. Mean Absolute Percentage Error (MAPE) - **MAPE** is a metric that measures the average percentage error between predicted and actual values. It's particularly useful when you want to understand the relative error, especially in cases where the scale of the data is important.
2. Root Mean Squared Error (RMSE) - **RMSE** is a metric that measures the average magnitude of errors. It's often used because it penalizes larger errors more heavily than smaller errors.
3. R-squared (R^2) - **R-squared** is a statistical measure that represents the proportion of variance in the dependent variable that is explained by the independent variables. It's often used to assess the overall fit of a regression model.
4. Adjusted R-squared - **Adjusted R-squared** is a variation of R-squared that penalizes the addition of unnecessary independent variables. It's useful when comparing models with different numbers of features.

S.no	Models	MAPE	RMSE	R-squared	Adjusted R-squared
1	Linear Regression	0.23	140366	0.68	0.68
2	Lasso Regression	0.84	421765	-1.91	-1.92
3	Ridge Regression	0.84	420720	-1.90	-1.90
4	Random Forest Regression	0.15	100941	0.83	0.83
5	Decision Tree Regression	0.18	126462	0.74	0.74
6	**XG Boost Regression**	**0.14**	**96437**	**0.85**	**0.85**

Based on the above metrics comparison table without optimised/tuned model, the key insights are as below:

1. Best Overall Performance: XG Boost Regression appears to be the best performing model overall. It has the lowest MAPE (0.14), lowest RMSE (96437), and highest R-squared and Adjusted R-squared values (0.85 for both).
2. Second Best Model: Random Forest Regression is a close second, with slightly better MAPE (0.15) but higher RMSE (100941) and slightly lower R-squared values (0.83) compared to XG Boost.
3. Linear Regression: Shows moderate performance, better than Lasso and Ridge but not as good as the tree-based models.
4. Ensemble Methods: The two best performers (XG Boost and Random Forest) are ensemble methods, indicating that combining multiple models yields better results for this particular problem.
5. Regularization Methods: The poor performance of Lasso and Ridge suggests that simple regularization techniques were not effective for this dataset.

CHAPTER 5

FINDINGS, RECOMMENDATIONS AND CONCLUSION

FINDINGS, RECOMMENDATIONS AND CONCLUSION

5.1 Findings Based on Observations

1. Data exploration: The dataset contains 21,613 rows and 23 columns with various attributes of houses.
2. Pre-processing: Some columns had unwanted variables like '$' symbols that needed to be removed. The datasets will be checked and pre-processed using the methods. Those methods have various ways of handling data.
3. Missing value treatment: There were missing values present in multiple columns that required treatment.
4. Removal of Outliers: Outliers were identified in several numerical columns through box plots.
5. Fluctuations: Seasonal fluctuations affect pricing, with peak buying seasons yielding higher prices.
6. Rise in price: Properties with modern amenities tend to sell for a premium compared to older homes without upgrades.
7. Evaluation: The accuracy of dataset will be evaluated by measuring the R-Squared and RMSE rate when training the model alongside an evaluation of the actual prices on the test dataset with the prices that are being predicted by the model.
8. Performance: Alongside the evaluation metrics, the required time to train the model will be measured to show the algorithm vary in terms of time.

5.2 Findings Based on analysis of Data

1. The price distribution was right-skewed, indicating lot of lower-priced houses and fewer high-priced properties. There was a positive correlation between house price and living space area.
2. Above-ground living space (ceil_measure) had a stronger correlation with price compared to basement area.
3. Location factors like zipcode, latitude and longitude were important in determining house prices.

4. The dataset included subjective assessments of houses through 'quality' and 'condition' variables.
5. Waterfront properties likely commanded a premium based on the 'coast' variable.
6. Feature selection narrowed down the important predictive features from 23 to 8 important features for clustering.
7. Key features influencing house prices include square footage, number of bedrooms, and proximity to schools.
8. The Linear Regression model provided a satisfactory performance but was outperformed by more complex models.

5.3 General findings

1. There is a growing reliance on data analytics within the real estate sector and other sectors in the world. Stakeholders increasingly use value predictive analytics for making informed decisions.
2. Ensemble methods like XGBoost and Random Forest performed better than simple linear models for this dataset.
3. Regularization techniques like Lasso and Ridge did not significantly improve performance over basic linear regression.
4. Hyper-parameter tuning further improved the XGBoost model's performance by reducing RMSE to 92,608.
5. Cross-validation offers more reliable performance metrics than a single train-test split.

5.4 Recommendation based on findings

1. Real estate professionals should incorporate machine learning tools for better price estimation. Training sessions for stakeholders on data interpretation can enhance decision-making processes.
2. Regular updates of datasets are essential to maintain model accuracy.
3. Use the optimized XGBoost model for house price predictions, as it demonstrated the best performance.

4. Focus on the 8 key features identified through feature selection when collecting data or making pricing decisions.
5. Pay special attention to above-ground living space when assessing property values, as it showed stronger correlation with price than basement area.
6. Implement separate pricing strategies for waterfront properties, given their likely premium.
7. Consider location factors carefully when valuing properties, as they proved to be significant predictors.
8. Implement cross-validation consistently to ensure more stable model evaluation results.

5.5 Suggestions for areas of improvement

1. If the data is in bad shape, the model will be over fitted which means that data pre-processing is an important part of this experiment and will affect the final results.
2. Multiple combinations of pre-processing methods need to be tested before getting the data ready to be used in train.
3. Collect more data on renovation history to better quantify its impact on house prices.
4. Improve the quality and consistency of subjective assessments like 'condition' and 'quality'.
5. Gather more detailed location-based data (e.g., proximity to amenities, school districts) to enhance predictive power.
6. Implement continuous learning mechanisms in models to adapt to changing market conditions.
7. Incorporate additional features or perform more advanced feature engineering to improve model performance.
8. Consider more advanced tuning methods like Bayesian optimization for hyperparameter selection.

5.6 Scope for future research

1. Investigate the impact of seasonal trends on house prices.
2. Keep the data updated in real-time by recording each transactions of house bought or sold by any parties. This will help us to improve the data more and train it based on our future requirements.
3. Analyze the long-term price appreciation rates in different neighbourhoods.
4. Study the effects of urban development projects on surrounding property values.
5. Explore the use of more advanced machine learning techniques like neural networks for price prediction.
6. Conduct a comparative study of housing markets in different cities or regions.
7. Combine the related features into groups (binning) by use of feature engineering to see if it has improved the model's performance.
8. Testing the different regression models, including Elastic Net, which combines L1 and L2 regularization. This will allow us to compare the performance of various approaches.

5.7 Conclusion

1. The house price prediction model developed using XGBoost regression demonstrates strong predictive performance, with potential for real-world application in the real estate market.
2. This project demonstrates that while traditional regression models like Linear and Lasso Regression offer simplicity, they are outperformed by more sophisticated ensemble methods such as Random Forest and XGBoost in terms of prediction accuracy and error reduction.
3. Hyperparameter tuning and cross-validation proved crucial in obtaining reliable and robust results. For future projects, focusing on ensemble models with proper tuning will lead to more accurate and generalizable predictions.
4. By focusing on key features using RFE and leveraging advanced machine learning techniques, the model provides valuable insights for pricing strategies and investment decisions.
5. As technology continues to evolve, embracing these analytical tools will be crucial for navigating the complexities of the housing market effectively.

REFERENCES

- Brownlee, J. (2019). *RFE Feature Selection in Python* | Machine Learning Mastery. https://machinelearningmastery.com/rfe-feature-selection-in-python/
- Jeong, Y. (2018). *Housing Prices* | GitHub. https://github.com/yjeong5126/housing_prices
- *House Price Prediction Using Machine Learning in Python*. (2020). GeeksforGeeks. https://www.geeksforgeeks.org/house-price-prediction-using-machine-learning-in-python/
- Jeong, Y. (2019). *Predicting Housing Prices in Melbourne* | Medium. https://yjeong5126.medium.com/predicting-housing-prices-in-melbourne-e3d5f49abf20
- Harika. (2021). Detecting and Treating Outliers | Treating the odd one out!

  https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/
- OpenAI (2024). *AI chat bot like ChatGPT and Gemini for certain queries*.

  Chatgpt.com | Gemini.google.com
- Moriarty, K. (2019, October 29). How to build a house price prediction model.

  https://www.freecodecamp.org/news/how-to-build-a-house-price-prediction-model/

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df= pd.read_excel("D:\Mak\Jain study\innercity.xlsx")
df.head()

          cid         dayhours   price  room_bed  room_bath
living_measure  \
0  3876100940  20150427T000000  600000       4.0       1.75
3050.0
1  3145600250  20150317T000000  190000       2.0       1.00
670.0
2  7129303070  20140820T000000  735000       4.0       2.75
3040.0
3  7338220280  20141010T000000  257000       3.0       2.50
1740.0
4  7950300670  20150218T000000  450000       2.0       1.00
1120.0

   lot_measure ceil  coast  sight  ... basement  yr_built  yr_renovated
\
0       9440.0    1      0    0.0  ...   1250.0      1966             0

1       3101.0    1      0    0.0  ...      0.0      1948             0

2       2415.0    2      1    4.0  ...      0.0      1966             0

3       3721.0    2      0    0.0  ...      0.0      2009             0

4       4590.0    1      0    0.0  ...      0.0      1924             0


   zipcode      lat     long  living_measure15  lot_measure15
furnished  \
0    98034  47.7228 -122.183            2020.0         8660.0
0.0
1    98118  47.5546 -122.274            1660.0         4100.0
0.0
2    98118  47.5188 -122.256            2620.0         2433.0
0.0
3    98002  47.3363 -122.213            2030.0         3794.0
0.0
4    98118  47.5663 -122.285            1120.0         5100.0
0.0

   total_area
0       12490
1        3771
```

```
2        5455
3        5461
4        5710

[5 rows x 23 columns]
```

```
df.describe()
```

```
                cid         price      room_bed     room_bath
living_measure  \
count  2.161300e+04  2.161300e+04  21505.000000  21505.000000
21596.000000
mean   4.580302e+09  5.401822e+05      3.371355      2.115171
2079.860761
std    2.876566e+09  3.673622e+05      0.930289      0.770248
918.496121
min    1.000102e+06  7.500000e+04      0.000000      0.000000
290.000000
25%    2.123049e+09  3.219500e+05      3.000000      1.750000
1429.250000
50%    3.904930e+09  4.500000e+05      3.000000      2.250000
1910.000000
75%    7.308900e+09  6.450000e+05      4.000000      2.500000
2550.000000
max    9.900000e+09  7.700000e+06     33.000000      8.000000
13540.000000

        lot_measure         sight       quality  ceil_measure
basement  \
count  2.157100e+04  21556.000000  21612.000000  21612.000000
21612.000000
mean   1.510458e+04      0.234366      7.656857   1788.366556
291.522534
std    4.142362e+04      0.766438      1.175484    828.102535
442.580840
min    5.200000e+02      0.000000      1.000000    290.000000
0.000000
25%    5.040000e+03      0.000000      7.000000   1190.000000
0.000000
50%    7.618000e+03      0.000000      7.000000   1560.000000
0.000000
75%    1.068450e+04      0.000000      8.000000   2210.000000
560.000000
max    1.651359e+06      4.000000     13.000000   9410.000000
4820.000000

        yr_renovated       zipcode           lat  living_measure15  \
count  21613.000000  21613.000000  21613.000000      21447.000000
mean      84.402258  98077.939805     47.560053       1987.065557
std      401.679240     53.505026      0.138564        685.519629
```

```
min        0.000000  98001.000000     47.155900        399.000000
25%        0.000000  98033.000000     47.471000       1490.000000
50%        0.000000  98065.000000     47.571800       1840.000000
75%        0.000000  98118.000000     47.678000       2360.000000
max     2015.000000  98199.000000     47.777600       6210.000000

       lot_measure15     furnished
count   21584.000000  21584.000000
mean    12766.543180      0.196720
std     27286.987107      0.397528
min       651.000000      0.000000
25%      5100.000000      0.000000
50%      7620.000000      0.000000
75%     10087.000000      0.000000
max    871200.000000      1.000000
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 23 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   cid               21613 non-null  int64
 1   dayhours          21613 non-null  object
 2   price             21613 non-null  int64
 3   room_bed          21505 non-null  float64
 4   room_bath         21505 non-null  float64
 5   living_measure    21596 non-null  float64
 6   lot_measure       21571 non-null  float64
 7   ceil              21571 non-null  object
 8   coast             21612 non-null  object
 9   sight             21556 non-null  float64
 10  condition         21556 non-null  object
 11  quality           21612 non-null  float64
 12  ceil_measure      21612 non-null  float64
 13  basement          21612 non-null  float64
 14  yr_built          21612 non-null  object
 15  yr_renovated      21613 non-null  int64
 16  zipcode           21613 non-null  int64
 17  lat               21613 non-null  float64
 18  long              21613 non-null  object
 19  living_measure15  21447 non-null  float64
 20  lot_measure15     21584 non-null  float64
 21  furnished         21584 non-null  float64
 22  total_area        21584 non-null  object
dtypes: float64(12), int64(4), object(7)
memory usage: 3.8+ MB
```

```
df.shape
```

```
(21613, 23)
```

```
df.isnull()
```

```
         cid  dayhours  price  room_bed  room_bath  living_measure  \
0      False     False  False     False      False           False
1      False     False  False     False      False           False
2      False     False  False     False      False           False
3      False     False  False     False      False           False
4      False     False  False     False      False           False
...      ...       ...    ...       ...        ...             ...
21608  False     False  False     False      False           False
21609  False     False  False     False      False           False
21610  False     False  False     False      False           False
21611  False     False  False     False      False           False
21612  False     False  False     False      False           False

       lot_measure   ceil  coast  sight  ...  basement  yr_built  \
0            False  False  False  False  ...     False     False
1            False  False  False  False  ...     False     False
2            False  False  False  False  ...     False     False
3            False  False  False  False  ...     False     False
4            False  False  False  False  ...     False     False
...            ...    ...    ...    ...  ...       ...       ...
21608        False  False  False  False  ...     False     False
21609        False  False  False  False  ...     False     False
21610        False  False  False  False  ...     False     False
21611        False  False  False  False  ...     False     False
21612        False  False  False  False  ...     False     False

       yr_renovated  zipcode    lat   long  living_measure15
lot_measure15  \
0             False    False  False  False             False
False
1             False    False  False  False             False
False
2             False    False  False  False             False
False
3             False    False  False  False             False
False
4             False    False  False  False             False
False
...             ...      ...    ...    ...               ...
...
21608         False    False  False  False             False
False
21609         False    False  False  False             False
False
21610         False    False  False  False             False
False
```

```
21611          False    False  False  False              False
False
21612          False    False  False  False              False
False

       furnished  total_area
0          False       False
1          False       False
2          False       False
3          False       False
4          False       False
...          ...         ...
21608      False       False
21609      False       False
21610      False       False
21611      False       False
21612      False       False

[21613 rows x 23 columns]
```

```
obj = (df.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))

int_ = (df.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))

fl = (df.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:",len(fl_cols))
```

```
Categorical variables: 7
Integer variables: 0
Float variables: 12
```

```
df.isna().sum()
```

```
cid                   0
dayhours              0
price                 0
room_bed            108
room_bath           108
living_measure       17
lot_measure          42
ceil                 42
coast                 1
sight                57
condition            57
quality               1
```

```
ceil_measure           1
basement               1
yr_built               1
yr_renovated           0
zipcode                0
lat                    0
long                   0
living_measure15     166
lot_measure15         29
furnished             29
total_area            29
dtype: int64
```

```
#Converting $ to null values
df['total_area'] = df['total_area'].astype(str).str.replace('$', '',
regex=False)
df['total_area'] = df['total_area'].replace('', np.nan)
df['total_area'] = pd.to_numeric(df['total_area'], errors='coerce')

df['ceil'] = df['ceil'].astype(str).str.replace('$', '', regex=False)
df['ceil'] = df['ceil'].replace('', np.nan)
df['ceil'] = pd.to_numeric(df['ceil'], errors='coerce')

df['coast'] = df['coast'].astype(str).str.replace('$', '',
regex=False)
df['coast'] = df['coast'].replace('', np.nan)
df['coast'] = pd.to_numeric(df['coast'], errors='coerce')

df['condition'] = df['condition'].astype(str).str.replace('$', '',
regex=False)
df['condition'] = df['condition'].replace('', np.nan)
df['condition'] = pd.to_numeric(df['condition'], errors='coerce')

df['yr_built'] = df['yr_built'].astype(str).str.replace('$', '',
regex=False)
df['yr_built'] = df['yr_built'].replace('', np.nan)
df['yr_built'] = pd.to_numeric(df['yr_built'], errors='coerce')

df['long'] = df['long'].astype(str).str.replace('$', '', regex=False)
df['long'] = df['long'].replace('', np.nan)
df['long'] = pd.to_numeric(df['long'], errors='coerce')
```

```
df.isna().sum()
```

```
cid                    0
dayhours               0
price                  0
room_bed             108
room_bath            108
living_measure        17
```

```
lot_measure           42
ceil                  72
coast                 31
sight                 57
condition             85
quality                1
ceil_measure           1
basement               1
yr_built              15
yr_renovated           0
zipcode                0
lat                    0
long                  34
living_measure15     166
lot_measure15         29
furnished             29
total_area            68
dtype: int64
```

```
#median imputation
from sklearn.impute import SimpleImputer
median_imputer = SimpleImputer(strategy='median')

df['room_bed'] = median_imputer.fit_transform(df[['room_bed']])
df['living_measure'] =
median_imputer.fit_transform(df[['living_measure']])
df['lot_measure'] = median_imputer.fit_transform(df[['lot_measure']])
df['coast'] = median_imputer.fit_transform(df[['coast']])
df['sight'] = median_imputer.fit_transform(df[['sight']])
df['condition'] = median_imputer.fit_transform(df[['condition']])
df['quality'] = median_imputer.fit_transform(df[['quality']])
df['ceil_measure'] =
median_imputer.fit_transform(df[['ceil_measure']])
df['basement'] = median_imputer.fit_transform(df[['basement']])
df['yr_built'] = median_imputer.fit_transform(df[['yr_built']])
df['long'] = median_imputer.fit_transform(df[['long']])
df['living_measure15'] =
median_imputer.fit_transform(df[['living_measure15']])
df['lot_measure15'] =
median_imputer.fit_transform(df[['lot_measure15']])
df['furnished'] = median_imputer.fit_transform(df[['furnished']])
df['total_area'] = median_imputer.fit_transform(df[['total_area']])

# Convert to categorical using type
df['room_bath'] = df['room_bath'].astype('category')
df['ceil'] = df['ceil'].astype('category')

# Assign numerical codes
df['room_bath'] = df['room_bath'].cat.codes + 1
df['ceil'] = df['ceil'].cat.codes + 1
```

```
#mode imputation
from sklearn.impute import SimpleImputer
mode_imputer = SimpleImputer(strategy='most_frequent')

df['room_bed'] = mode_imputer.fit_transform(df[['room_bed']])
df['ceil'] = mode_imputer.fit_transform(df[['ceil']])

df.isna().sum()
```

```
cid                 0
dayhours            0
price               0
room_bed            0
room_bath           0
living_measure      0
lot_measure         0
ceil                0
coast               0
sight               0
condition           0
quality             0
ceil_measure        0
basement            0
yr_built            0
yr_renovated        0
zipcode             0
lat                 0
long                0
living_measure15    0
lot_measure15       0
furnished           0
total_area          0
dtype: int64
```

```
from datetime import datetime

# Get the current year
current_year = datetime.now().year

# Calculate the 'age of the house'
df['age'] = current_year - df['yr_built']
df.loc[df['yr_renovated'] > 0, 'age'] = current_year -
df['yr_renovated']

# Convert 'dayhours' column to just date format (yyyy/mm/dd)
df['dayhours'] = pd.to_datetime(df['dayhours'].str[:8], format='%Y%m
%d')

# Display the first few rows of the dataframe to check the changes
df[['yr_built', 'yr_renovated', 'age', 'dayhours']].head()
```

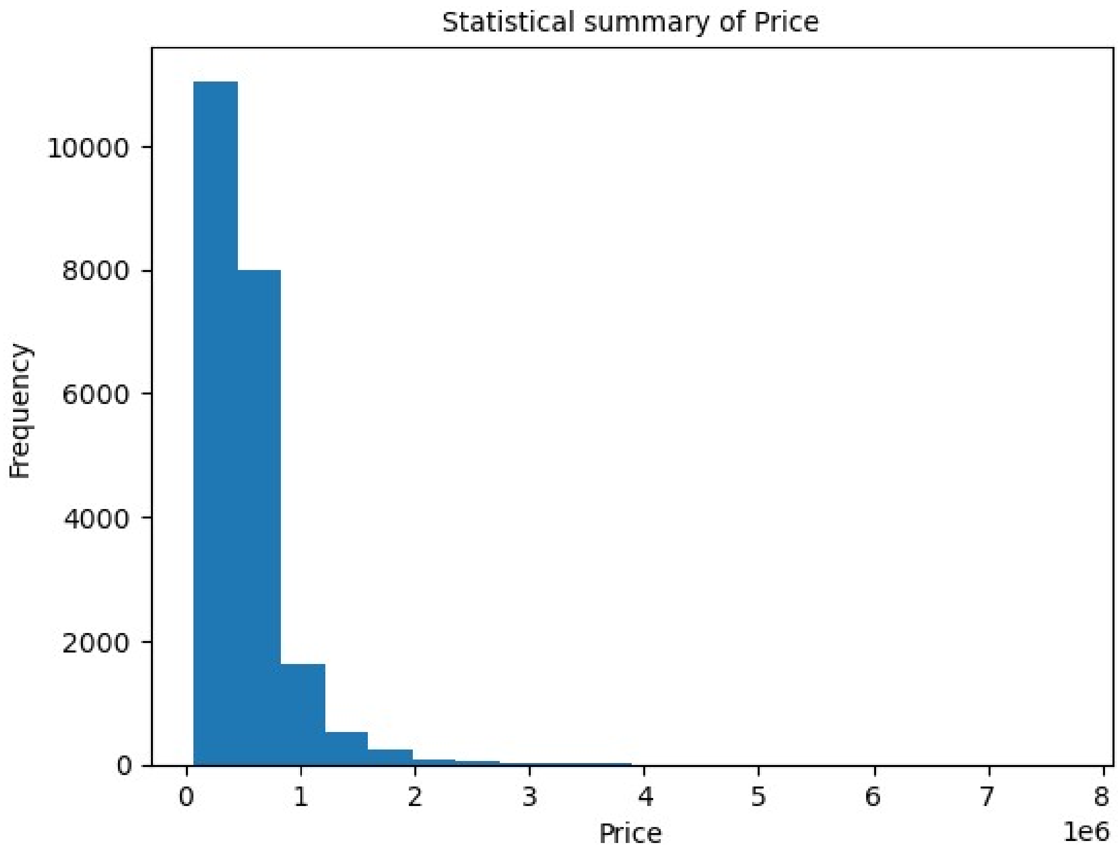
```
   yr_built  yr_renovated    age   dayhours
0    1966.0             0   58.0 2015-04-27
1    1948.0             0   76.0 2015-03-17
2    1966.0             0   58.0 2014-08-20
3    2009.0             0   15.0 2014-10-10
4    1924.0             0  100.0 2015-02-18
```

```
df.describe()
```

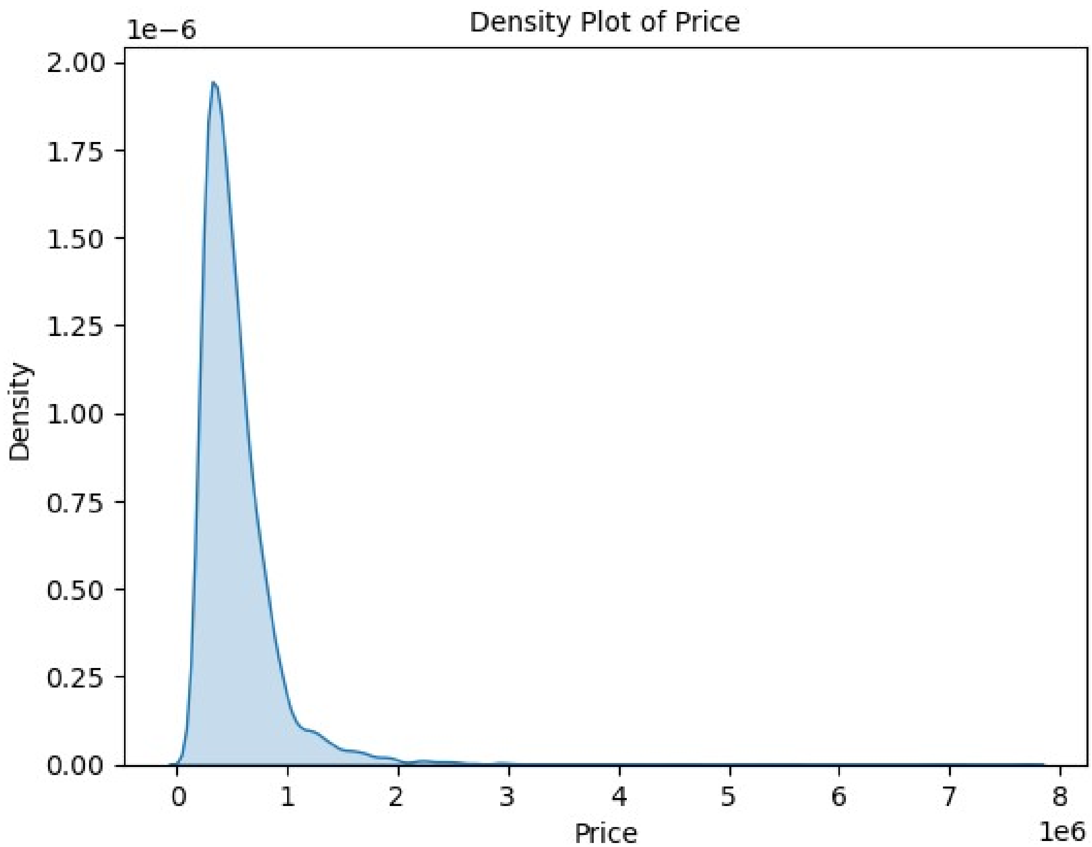
```
                cid         price      room_bed     room_bath  
living_measure  \
count  2.161300e+04  2.161300e+04  21613.000000  21613.000000   
21613.000000   
mean   4.580302e+09  5.401822e+05      3.369500      8.418498   
2079.727155   
std    2.876566e+09  3.673622e+05      0.928331      3.126902    
918.147155   
min    1.000102e+06  7.500000e+04      0.000000      0.000000    
290.000000   
25%    2.123049e+09  3.219500e+05      3.000000      6.000000   
1430.000000   
50%    3.904930e+09  4.500000e+05      3.000000      9.000000   
1910.000000   
75%    7.308900e+09  6.450000e+05      4.000000     10.000000   
2550.000000   
max    9.900000e+09  7.700000e+06     33.000000     30.000000  
13540.000000   

        lot_measure          ceil         coast         sight  
condition  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  
21613.000000   
mean   1.509003e+04      1.981631      0.007449      0.233748   
3.407718   
std    4.138466e+04      1.084094      0.085989      0.765521   
0.649933   
min    5.200000e+02      0.000000      0.000000      0.000000   
1.000000   
25%    5.043000e+03      1.000000      0.000000      0.000000   
3.000000   
50%    7.618000e+03      2.000000      0.000000      0.000000   
3.000000   
75%    1.066000e+04      3.000000      0.000000      0.000000   
4.000000   
max    1.651359e+06      6.000000      1.000000      4.000000   
5.000000   

       ...      yr_built  yr_renovated       zipcode           lat  \
count  ...  21613.000000  21613.000000  21613.000000  21613.000000   
mean   ...   1971.012122     84.402258  98077.939805     47.560053   
```

```
std     ...     29.363429    401.679240      53.505026       0.138564
min     ...   1900.000000      0.000000   98001.000000      47.155900
25%     ...   1951.000000      0.000000   98033.000000      47.471000
50%     ...   1975.000000      0.000000   98065.000000      47.571800
75%     ...   1997.000000      0.000000   98118.000000      47.678000
max     ...   2015.000000   2015.000000   98199.000000      47.777600

               long  living_measure15  lot_measure15     furnished  \
count  21613.000000      21613.000000   21613.000000  21613.000000
mean    -122.213869       1985.936011   12759.637626      0.196456
std        0.140759        683.002534   27269.324285      0.397326
min     -122.519000        399.000000     651.000000      0.000000
25%     -122.328000       1490.000000    5100.000000      0.000000
50%     -122.230000       1840.000000    7620.000000      0.000000
75%     -122.125000       2360.000000   10080.000000      0.000000
max     -121.315000       6210.000000  871200.000000      1.000000

         total_area           age
count  2.161300e+04  21613.000000
mean   1.716808e+04     50.610744
std    4.156534e+04     28.798701
min    1.423000e+03      9.000000
25%    7.040000e+03     25.000000
50%    9.575000e+03     47.000000
75%    1.297000e+04     70.000000
max    1.652659e+06    124.000000

[8 rows x 23 columns]
```

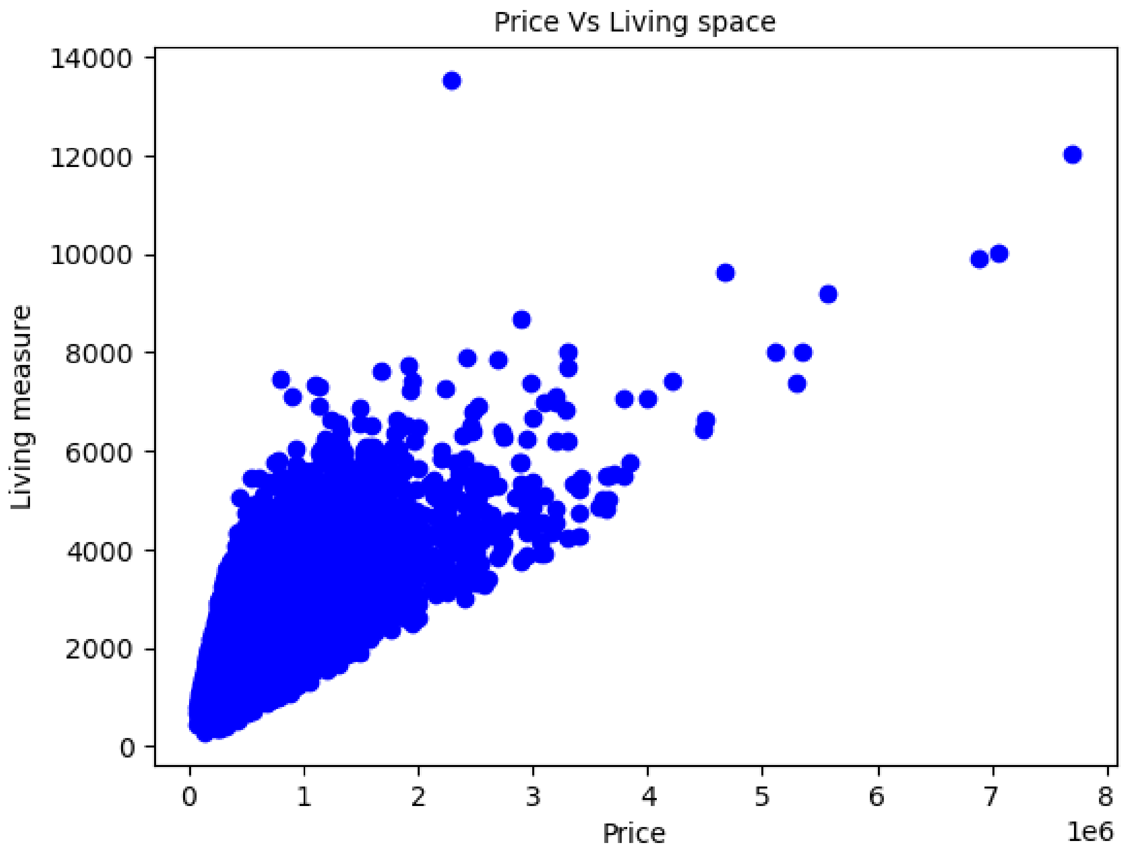
```
#univariate (histogram)
plt.hist(df['price'],bins=20)
plt.title('Statistical summary of Price',fontsize=10)
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



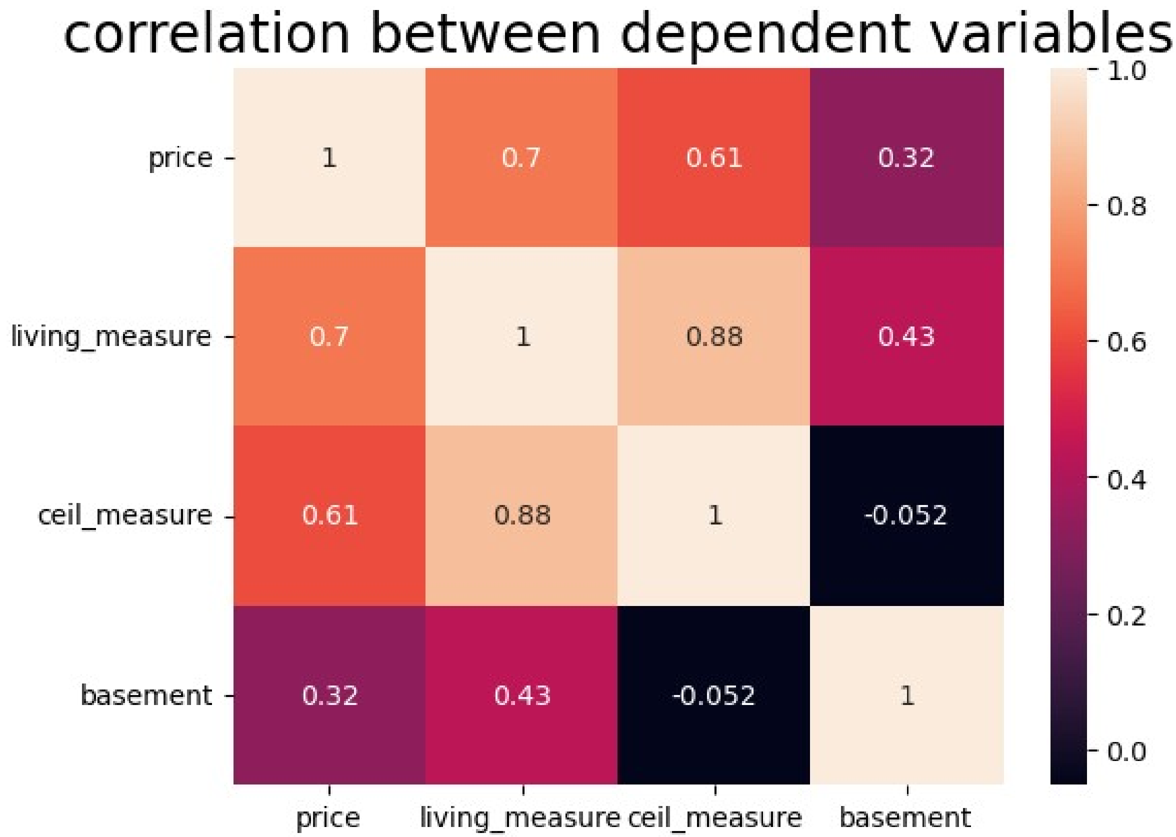

```
#univariate (density plot)
sns.kdeplot(df['price'], fill=True)
plt.title('Density Plot of Price', fontsize=10)
plt.xlabel('Price')
plt.ylabel('Density')
plt.show()
```



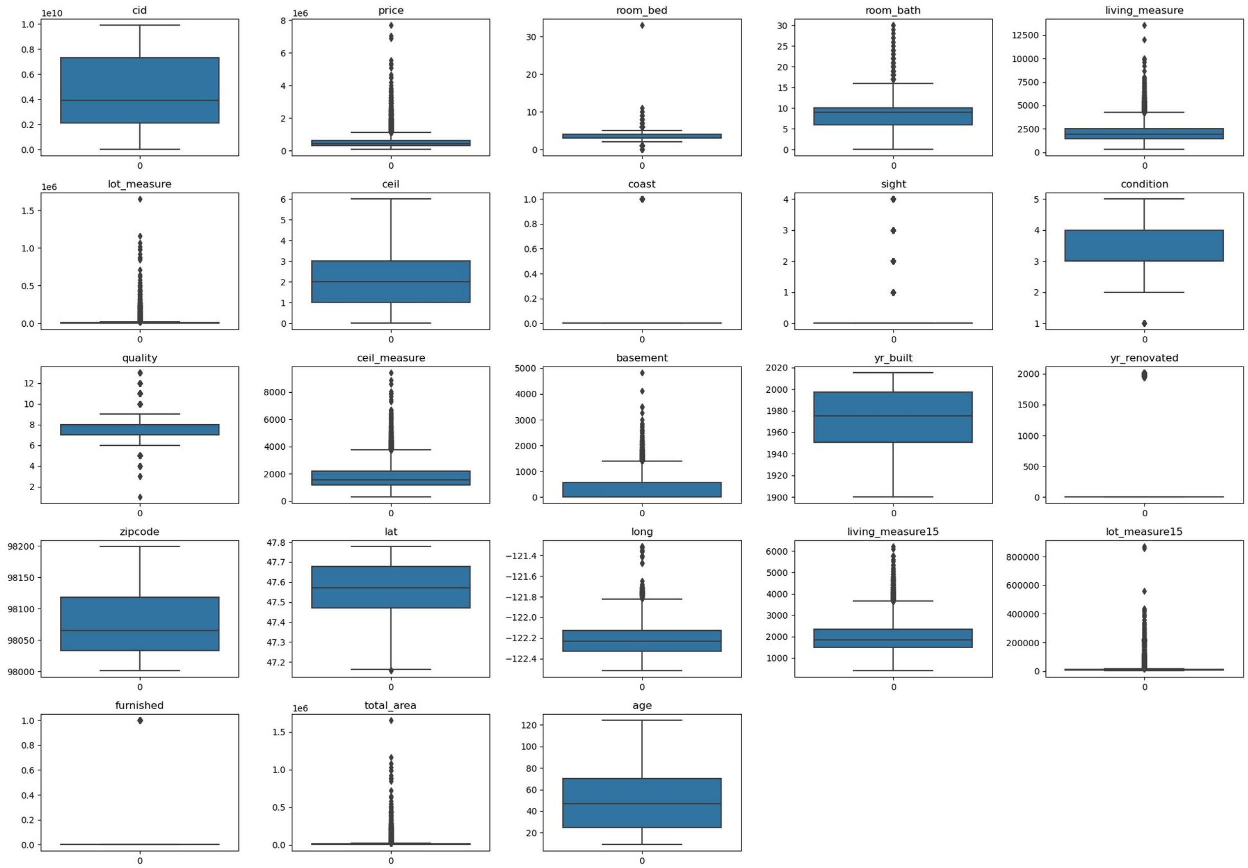

```python
#bivariate for price vs living space
plt.scatter(x=df['price'],y=df['living_measure'],color='blue')
plt.title('Price Vs Living space',fontsize=10)
plt.xlabel('Price')
plt.ylabel('Living measure')
plt.show()
```




```
#multivariate for price, living measure, ceil measure and basement
correlation_matrix=df[['price','living_measure','ceil_measure','baseme
nt']].corr()
sns.heatmap(correlation_matrix,annot=True)
plt.title('correlation between dependent variables',fontsize=20)
plt.show()
```

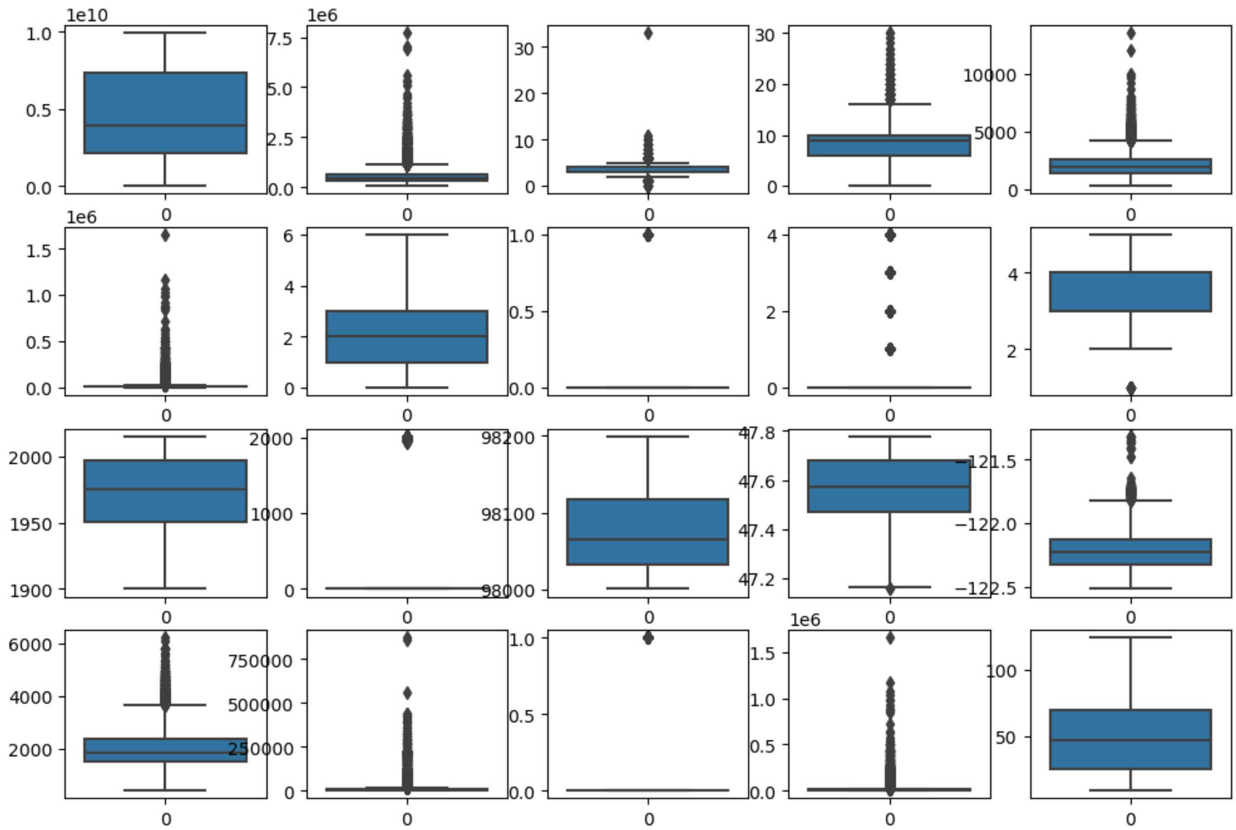



```python
# Columns to be processed
columns = ['cid', 'price', 'room_bed', 'room_bath', 'living_measure',
'lot_measure', 'ceil',
           'coast', 'sight', 'condition', 'quality', 'ceil_measure',
'basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat',
           'long', 'living_measure15', 'lot_measure15', 'furnished',
'total_area', 'age']

# Plotting boxplots before capping
plt.figure(figsize=(20, 14))

for i, column in enumerate(columns, 1):
  plt.subplot(5, 5, i)  # Removed extra space
  sns.boxplot(df[f'{column}'])
  plt.title(f'{column}')

plt.tight_layout()
plt.show()
```

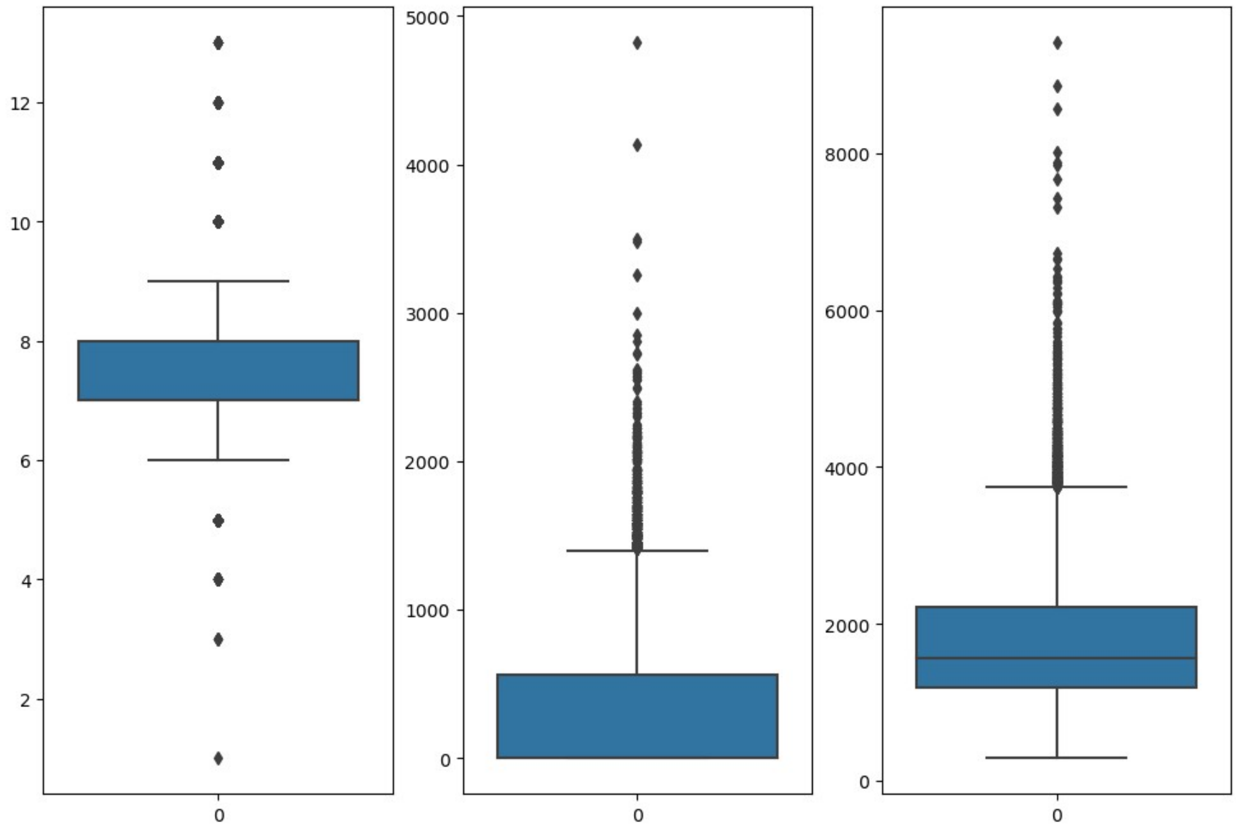


```
plt.figure(figsize=(12,8))

plt.subplot(4,5,1)
sns.boxplot(df['cid'])

plt.subplot(4,5,2)
sns.boxplot(df['price'])

plt.subplot(4,5,3)
sns.boxplot(df['room_bed'])

plt.subplot(4,5,4)
sns.boxplot(df['room_bath'])

plt.subplot(4,5,5)
sns.boxplot(df['living_measure'])

plt.subplot(4,5,6)
sns.boxplot(df['lot_measure'])

plt.subplot(4,5,7)
sns.boxplot(df['ceil'])

plt.subplot(4,5,8)
```

```
sns.boxplot(df['coast'])

plt.subplot(4,5,9)
sns.boxplot(df['sight'])

plt.subplot(4,5,10)
sns.boxplot(df['condition'])

plt.subplot(4,5,11)
sns.boxplot(df['yr_built'])

plt.subplot(4,5,12)
sns.boxplot(df['yr_renovated'])

plt.subplot(4,5,13)
sns.boxplot(df['zipcode'])

plt.subplot(4,5,14)
sns.boxplot(df['lat'])

plt.subplot(4,5,15)
sns.boxplot(df['long'])

plt.subplot(4,5,16)
sns.boxplot(df['living_measure15'])

plt.subplot(4,5,17)
sns.boxplot(df['lot_measure15'])

plt.subplot(4,5,18)
sns.boxplot(df['furnished'])

plt.subplot(4,5,19)
sns.boxplot(df['total_area'])

plt.subplot(4,5,20)
sns.boxplot(df['age'])

plt.show()
```

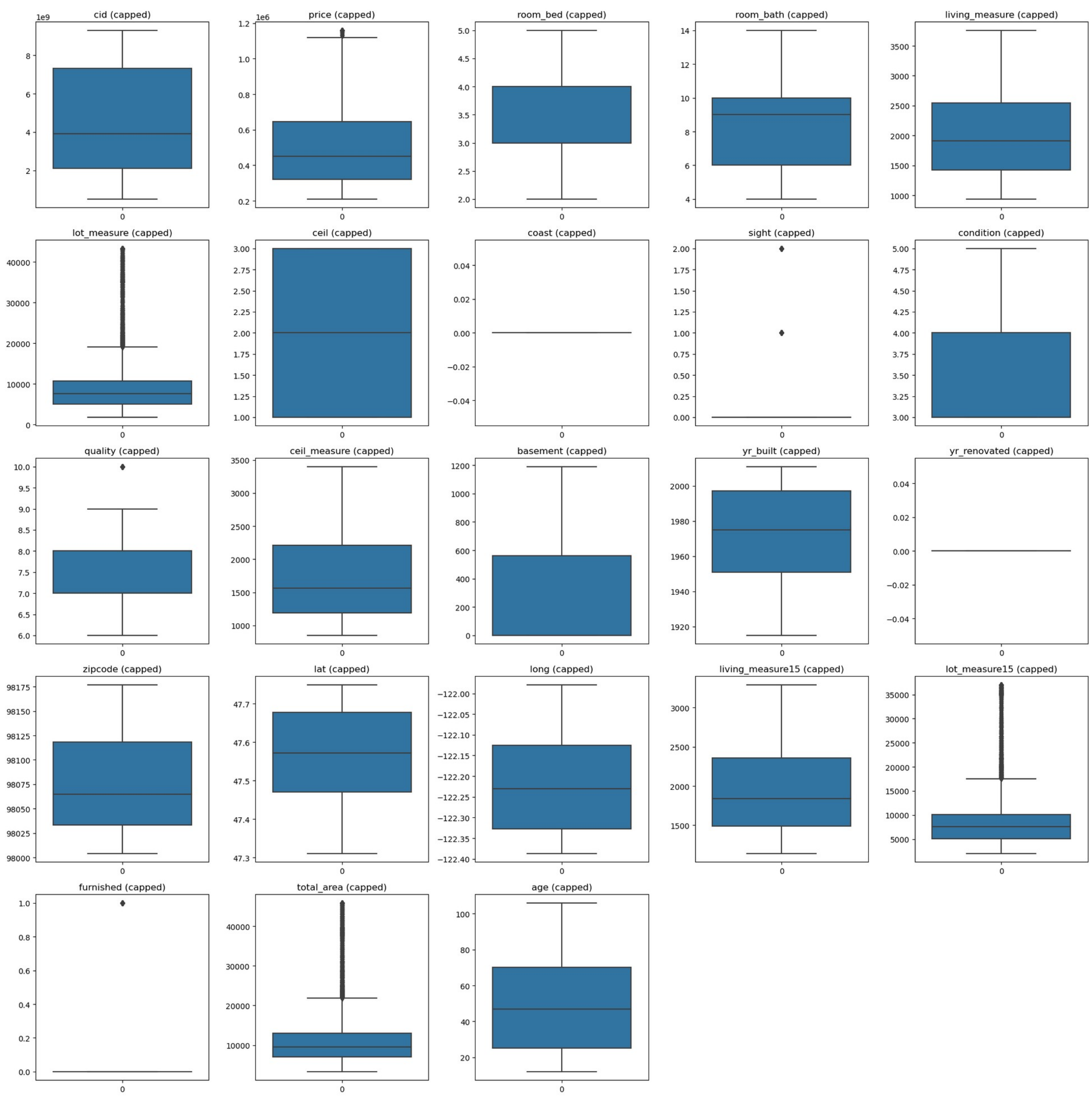



```
plt.figure(figsize=(12,8))

plt.subplot(1,3,1)
sns.boxplot(df['quality'])

plt.subplot(1,3,2)
sns.boxplot(df['basement'])

plt.subplot(1,3,3)
sns.boxplot(df['ceil_measure'])

plt.show()
```



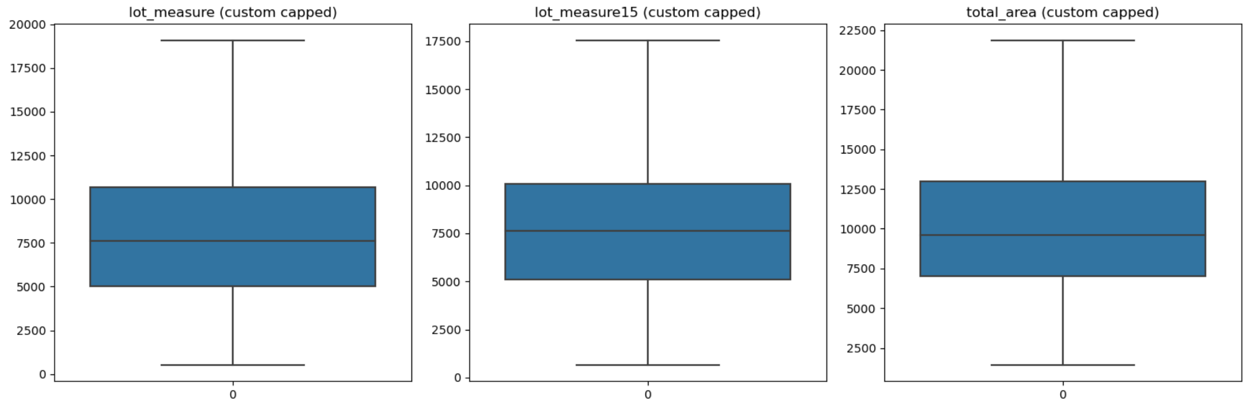

```
# Function to detect outliers using IQR
def detect_outliers_iqr(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1

    lower_limit = Q1 - 1.5 * IQR
    upper_limit = Q3 + 1.5 * IQR
    outliers = (data < lower_limit) | (data > upper_limit)
    return outliers

# Function to cap outliers
def cap_outliers(data, lower_percentile=0.05, upper_percentile=0.95):
    lower_cap = data.quantile(lower_percentile)
    upper_cap = data.quantile(upper_percentile)
    data = data.clip(lower=lower_cap, upper=upper_cap)
    return data

# Apply the IQR method to detect and cap outliers for the specified
columns
for column in columns:
    df[f'{column}_capped'] = cap_outliers(df[column])
```

```python
# Plotting boxplots after capping
plt.figure(figsize=(20, 20))

for i, column in enumerate(columns, 1):
    plt.subplot(5, 5, i)
    sns.boxplot(df[f'{column}_capped'])
    plt.title(f'{column} (capped)')

plt.tight_layout()
plt.show()
```



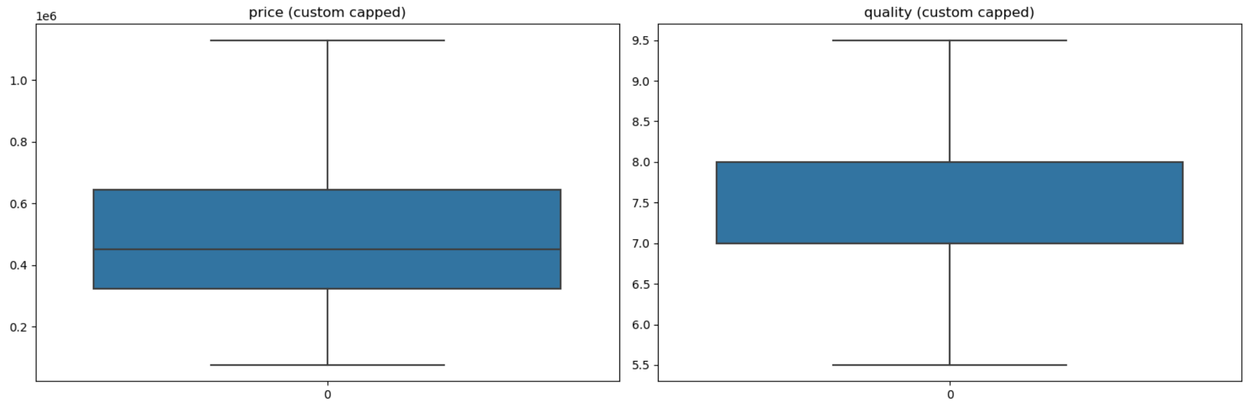
```
#the exact percentile values for sepcific outlier columns
lot_measure_25th = 5.043000e+03
lot_measure_75th = 1.066000e+04
lot_measure15_25th = 5100.000000
lot_measure15_75th = 10080.000000
total_area_25th = 7.040000e+03
total_area_75th = 1.297000e+04

# Calculate IQR and bounds
lot_measure_iqr = lot_measure_75th - lot_measure_25th
lot_measure_lower_limit = lot_measure_25th - 1.5 * lot_measure_iqr
lot_measure_upper_limit = lot_measure_75th + 1.5 * lot_measure_iqr

lot_measure15_iqr = lot_measure15_75th - lot_measure15_25th
lot_measure15_lower_limit = lot_measure15_25th - 1.5 *
lot_measure15_iqr
lot_measure15_upper_limit = lot_measure15_75th + 1.5 *
lot_measure15_iqr

total_area_iqr = total_area_75th - total_area_25th
total_area_lower_limit = total_area_25th - 1.5 * total_area_iqr
total_area_upper_limit = total_area_75th + 1.5 * total_area_iqr

# Apply capping based on the calculated bounds
df['lot_measure_custom_capped'] =
df['lot_measure'].clip(lower=lot_measure_lower_limit,
upper=lot_measure_upper_limit)
df['lot_measure15_custom_capped'] =
df['lot_measure15'].clip(lower=lot_measure15_lower_limit,
upper=lot_measure15_upper_limit)
df['total_area_custom_capped'] =
df['total_area'].clip(lower=total_area_lower_limit,
upper=total_area_upper_limit)


plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.boxplot(df['lot_measure_custom_capped'])
plt.title('lot_measure (custom capped)')

plt.subplot(1, 3, 2)
sns.boxplot(df['lot_measure15_custom_capped'])
plt.title('lot_measure15 (custom capped)')

plt.subplot(1, 3, 3)
sns.boxplot(df['total_area_custom_capped'])
plt.title('total_area (custom capped)')
```

```
plt.tight_layout()
plt.show()
```




```
#the exact percentile values for sepcific outlier columns
price_25th = 3.219500e+05
price_75th = 6.450000e+05
quality_25th = 7.000000
quality_75th = 8.000000


# Calculate IQR and bounds
price_iqr = price_75th - price_25th
price_lower_limit = price_25th - 1.5 * price_iqr
price_upper_limit = price_75th + 1.5 * price_iqr

quality_iqr = quality_75th - quality_25th
quality_lower_limit = quality_25th - 1.5 * quality_iqr
quality_upper_limit = quality_75th + 1.5 * quality_iqr


# Apply capping based on the calculated bounds
df['price_custom_capped'] = df['price'].clip(lower=price_lower_limit,
upper=price_upper_limit)
df['quality_custom_capped'] =
df['quality'].clip(lower=quality_lower_limit,
upper=quality_upper_limit)

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
sns.boxplot(df['price_custom_capped'])
plt.title('price (custom capped)')

plt.subplot(1, 2, 2)
sns.boxplot(df['quality_custom_capped'])
plt.title('quality (custom capped)')
```

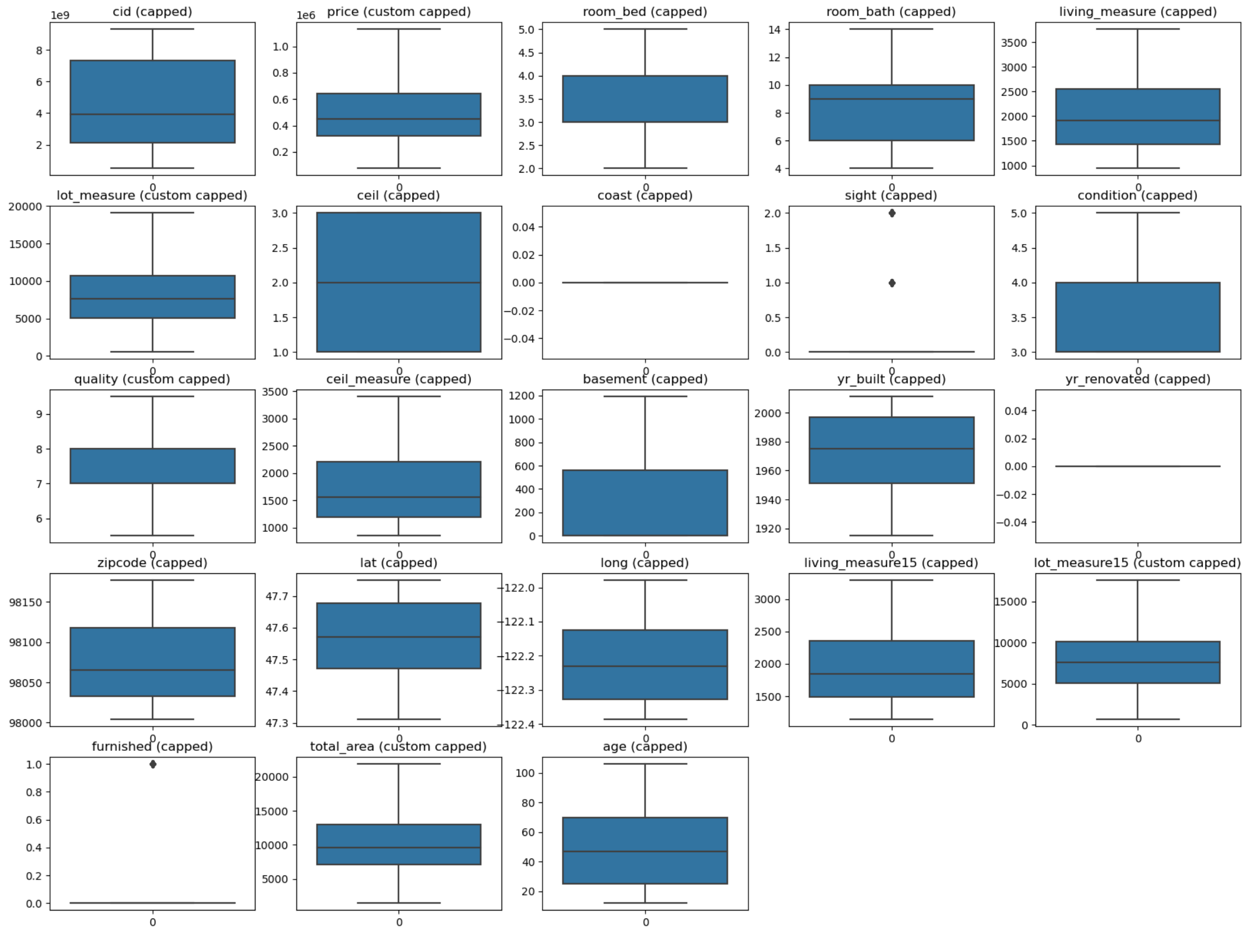
```
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(20,15))

plt.subplot(5,5,1)
sns.boxplot(df['cid_capped'])
plt.title('cid (capped)')

plt.subplot(5,5,2)
sns.boxplot(df['price_custom_capped'])
plt.title('price (custom capped)')

plt.subplot(5,5,3)
sns.boxplot(df['room_bed_capped'])
plt.title('room_bed (capped)')

plt.subplot(5,5,4)
sns.boxplot(df['room_bath_capped'])
plt.title('room_bath (capped)')

plt.subplot(5,5,5)
sns.boxplot(df['living_measure_capped'])
plt.title('living_measure (capped)')

plt.subplot(5,5,6)
sns.boxplot(df['lot_measure_custom_capped'])
plt.title('lot_measure (custom capped)')

plt.subplot(5,5,7)
sns.boxplot(df['ceil_capped'])
plt.title('ceil (capped)')

plt.subplot(5,5,8)
sns.boxplot(df['coast_capped'])
plt.title('coast (capped)')
```

```
plt.subplot(5,5,9)
sns.boxplot(df['sight_capped'])
plt.title('sight (capped)')

plt.subplot(5,5,10)
sns.boxplot(df['condition_capped'])
plt.title('condition (capped)')

plt.subplot(5,5,11)
sns.boxplot(df['quality_custom_capped'])
plt.title('quality (custom capped)')

plt.subplot(5,5,12)
sns.boxplot(df['ceil_measure_capped'])
plt.title('ceil_measure (capped)')

plt.subplot(5,5,13)
sns.boxplot(df['basement_capped'])
plt.title('basement (capped)')

plt.subplot(5,5,14)
sns.boxplot(df['yr_built_capped'])
plt.title('yr_built (capped)')

plt.subplot(5,5,15)
sns.boxplot(df['yr_renovated_capped'])
plt.title('yr_renovated (capped)')

plt.subplot(5,5,16)
sns.boxplot(df['zipcode_capped'])
plt.title('zipcode (capped)')

plt.subplot(5,5,17)
sns.boxplot(df['lat_capped'])
plt.title('lat (capped)')

plt.subplot(5,5,18)
sns.boxplot(df['long_capped'])
plt.title('long (capped)')

plt.subplot(5,5,19)
sns.boxplot(df['living_measure15_capped'])
plt.title('living_measure15 (capped)')

plt.subplot(5,5,20)
sns.boxplot(df['lot_measure15_custom_capped'])
plt.title('lot_measure15 (custom capped)')

plt.subplot(5,5,21)
sns.boxplot(df['furnished_capped'])
```

```
plt.title('furnished (capped)')

plt.subplot(5,5,22)
sns.boxplot(df['total_area_custom_capped'])
plt.title('total_area (custom capped)')

plt.subplot(5,5,23)
sns.boxplot(df['age_capped'])
plt.title('age (capped)')

plt.show()
```




```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

numeric_columns = [
    'price_custom_capped', 'room_bed_capped', 'room_bath_capped',
'living_measure_capped', 'lot_measure_custom_capped',
    'sight_capped', 'quality_custom_capped', 'basement_capped',
'ceil_measure_capped', 'furnished_capped',
```

```
    'living_measure15_capped', 'lot_measure15_custom_capped'
]

df_normalized = df.copy()
df_normalized[numeric_columns] =
scaler.fit_transform(df[numeric_columns])

df_normalized.head()

          cid   dayhours   price  room_bed  room_bath  living_measure
\
0  3876100940 2015-04-27  600000       4.0          7          3050.0

1  3145600250 2015-03-17  190000       2.0          4           670.0

2  7129303070 2014-08-20  735000       4.0         11          3040.0

3  7338220280 2014-10-10  257000       3.0         10          1740.0

4  7950300670 2015-02-18  450000       2.0          4          1120.0


   lot_measure  ceil  coast  sight  ...  living_measure15_capped  \
0       9440.0     1    0.0    0.0  ...                 0.409302
1       3101.0     1    0.0    0.0  ...                 0.241860
2       2415.0     3    1.0    4.0  ...                 0.688372
3       3721.0     3    0.0    0.0  ...                 0.413953
4       4590.0     1    0.0    0.0  ...                 0.000000

   lot_measure15_capped  furnished_capped  total_area_capped
age_capped  \
0                8660.0               0.0            12490.0
58.0
1                4100.0               0.0             3771.0
76.0
2                2433.0               0.0             5455.0
58.0
3                3794.0               0.0             5461.0
15.0
4                5100.0               0.0             5710.0
100.0

   lot_measure_custom_capped  lot_measure15_custom_capped  \
0                   0.480461                     0.473933
1                   0.139021                     0.204095
2                   0.102071                     0.105450
3                   0.172417                     0.185987
4                   0.219224                     0.263270

   total_area_custom_capped  price_custom_capped
quality_custom_capped
```

```
0                    12490.0             0.497831
0.625
1                     3771.0             0.109049
0.125
2                     5455.0             0.625845
0.625
3                     5461.0             0.172581
0.625
4                     5710.0             0.355593
0.375

[5 rows x 52 columns]

from sklearn.preprocessing import LabelEncoder

# One-Hot Encoding for 'ceil', 'zipcode' and 'coast'
df_encoded = pd.get_dummies(df, columns=['ceil_capped',
'coast_capped', 'zipcode_capped'], drop_first=True)

#Convert'yr_built', 'long', 'total_area' to numeric
df['yr_built_capped'] = pd.to_numeric(df['yr_built_capped'], errors =
'coerce')
df['long_capped'] = pd.to_numeric(df['long_capped'], errors =
'coerce')
df['total_area_custom_capped'] =
pd.to_numeric(df['total_area_custom_capped'], errors = 'coerce')

# Label Encoding for 'condition'
label_encoder = LabelEncoder()
df_encoded['condition_capped'] =
label_encoder.fit_transform(df['condition_capped'])

# Display the first few rows of the encoded DataFrame
df_encoded.head()

          cid   dayhours   price  room_bed  room_bath  living_measure
\
0  3876100940 2015-04-27  600000       4.0          7          3050.0

1  3145600250 2015-03-17  190000       2.0          4           670.0

2  7129303070 2014-08-20  735000       4.0         11          3040.0

3  7338220280 2014-10-10  257000       3.0         10          1740.0

4  7950300670 2015-02-18  450000       2.0          4          1120.0


   lot_measure  ceil  coast  sight  ...  zipcode_capped_98126  \
0       9440.0     1    0.0    0.0  ...                     0
1       3101.0     1    0.0    0.0  ...                     0
```

```
2       2415.0     3    1.0    4.0  ...                     0
3       3721.0     3    0.0    0.0  ...                     0
4       4590.0     1    0.0    0.0  ...                     0

   zipcode_capped_98133  zipcode_capped_98136  zipcode_capped_98144  \
0                     0                     0                     0
1                     0                     0                     0
2                     0                     0                     0
3                     0                     0                     0
4                     0                     0                     0

   zipcode_capped_98146  zipcode_capped_98148  zipcode_capped_98155  \
0                     0                     0                     0
1                     0                     0                     0
2                     0                     0                     0
3                     0                     0                     0
4                     0                     0                     0

   zipcode_capped_98166  zipcode_capped_98168  zipcode_capped_98177
0                     0                     0                     0
1                     0                     0                     0
2                     0                     0                     0
3                     0                     0                     0
4                     0                     0                     0

[5 rows x 113 columns]
```

```
df_encoded.ceil_measure_capped
```

```
0        1800.0
1         850.0
2        3040.0
3        1740.0
4        1120.0
          ...
21608    3130.0
21609     920.0
21610    2910.0
21611    1560.0
21612    1940.0
Name: ceil_measure_capped, Length: 21613, dtype: float64
```

```
df['total_rooms'] = df['room_bed_capped'] + df['room_bath_capped']
df['total_area'] = df['living_measure_capped'] +
df['lot_measure_custom_capped']

#Verify
df[['room_bed_capped', 'room_bath_capped', 'total_rooms',
    'living_measure_capped', 'lot_measure_custom_capped',
'total_area']].head()
```

```
   room_bed_capped  room_bath_capped  total_rooms
living_measure_capped  \
0              4.0                 7         11.0
3050.0
1              2.0                 4          6.0
940.0
2              4.0                11         15.0
3040.0
3              3.0                10         13.0
1740.0
4              2.0                 4          6.0
1120.0

   lot_measure_custom_capped  total_area
0                     9440.0     12490.0
1                     3101.0      4041.0
2                     2415.0      5455.0
3                     3721.0      5461.0
4                     4590.0      5710.0
```

```
df[['total_rooms', 'total_area']].isnull().sum()
```

```
total_rooms    0
total_area     0
dtype: int64
```

```
columns = [
    'cid_capped', 'room_bed_capped', 'room_bath_capped',
    'living_measure_capped', 'lot_measure_custom_capped',
'ceil_capped',
    'coast_capped', 'sight_capped', 'condition_capped',
'quality_custom_capped',
    'ceil_measure_capped', 'basement_capped', 'yr_built_capped',
'yr_renovated_capped',
    'zipcode_capped', 'lat_capped', 'long_capped',
'living_measure15_capped',
    'lot_measure15_custom_capped', 'furnished_capped',
'total_area_custom_capped',
    'age_capped', 'total_rooms', 'total_area'
]

X = df[columns]

X
```

```
         cid_capped  room_bed_capped  room_bath_capped
living_measure_capped  \
0      3.876101e+09              4.0                 7
3050.0
1      3.145600e+09              2.0                 4
940.0
```

```
2      7.129303e+09              4.0                11
3040.0
3      7.338220e+09              3.0                10
1740.0
4      7.950301e+09              2.0                 4
1120.0
...             ...              ...               ...
...
21608  5.124803e+08              4.0                10
3130.0
21609  6.250493e+08              2.0                 4
1030.0
21610  5.124803e+08              3.0                14
3710.0
21611  7.258200e+09              4.0                10
1560.0
21612  8.805900e+09              4.0                10
1940.0

       lot_measure_custom_capped  ceil_capped  coast_capped
sight_capped  \
0                         9440.0            1           0.0
0.0
1                         3101.0            1           0.0
0.0
2                         2415.0            3           0.0
2.0
3                         3721.0            3           0.0
0.0
4                         4590.0            1           0.0
0.0
...                          ...          ...           ...
...
21608                    19085.5            3           0.0
0.0
21609                     4841.0            1           0.0
0.0
21610                    19085.5            3           0.0
0.0
21611                     7800.0            3           0.0
0.0
21612                     4875.0            3           0.0
0.0

       condition_capped  quality_custom_capped  ...  zipcode_capped  \
0                   3.0                    8.0  ...           98034
1                   4.0                    6.0  ...           98118
2                   3.0                    8.0  ...           98118
3                   3.0                    8.0  ...           98004
```

```
4                    3.0                     7.0  ...          98118
...                  ...                     ...  ...            ...
21608                3.0                     9.0  ...          98014
21609                3.0                     7.0  ...          98103
21610                3.0                     9.5  ...          98075
21611                3.0                     7.0  ...          98168
21612                4.0                     9.0  ...          98112

       lat_capped  long_capped  living_measure15_capped  \
0         47.7228     -122.183                   2020.0
1         47.5546     -122.274                   1660.0
2         47.5188     -122.256                   2620.0
3         47.3363     -122.213                   2030.0
4         47.5663     -122.285                   1140.0
...           ...          ...                      ...
21608     47.6618     -121.979                   2780.0
21609     47.6860     -122.341                   1530.0
21610     47.5888     -122.040                   2390.0
21611     47.5140     -122.316                   1160.0
21612     47.6427     -122.304                   1790.0

       lot_measure15_custom_capped  furnished_capped  \
0                           8660.0               0.0
1                           4100.0               0.0
2                           2433.0               0.0
3                           3794.0               0.0
4                           5100.0               0.0
...                            ...               ...
21608                      17550.0               1.0
21609                       4944.0               0.0
21610                      17550.0               1.0
21611                       7800.0               0.0
21612                       4875.0               1.0

       total_area_custom_capped  age_capped  total_rooms  total_area
0                       12490.0        58.0         11.0     12490.0
1                        3771.0        76.0          6.0      4041.0
2                        5455.0        58.0         15.0      5455.0
3                        5461.0        15.0         13.0      5461.0
4                        5710.0       100.0          6.0      5710.0
...                         ...         ...          ...         ...
21608                   21865.0        28.0         14.0     22215.5
21609                    5871.0        85.0          6.0      5871.0
21610                   21865.0        46.0         17.0     22795.5
21611                    9360.0        27.0         14.0      9360.0
21612                    6815.0        99.0         14.0      6815.0

[21613 rows x 24 columns]
```

```
X.isna().sum()
```

```
cid_capped                          0
room_bed_capped                     0
room_bath_capped                    0
living_measure_capped               0
lot_measure_custom_capped           0
ceil_capped                         0
coast_capped                        0
sight_capped                        0
condition_capped                    0
quality_custom_capped               0
ceil_measure_capped                 0
basement_capped                     0
yr_built_capped                     0
yr_renovated_capped                 0
zipcode_capped                      0
lat_capped                          0
long_capped                         0
living_measure15_capped             0
lot_measure15_custom_capped         0
furnished_capped                    0
total_area_custom_capped            0
age_capped                          0
total_rooms                         0
total_area                          0
dtype: int64
```

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# K-Means with n=15 clusters
kmeans_15 = KMeans(n_clusters=15, random_state=42)
kmeans_15.fit(X)
df['Cluster_15'] = kmeans_15.labels_

# Calculating the Silhouette Scores
sil_score_15 = silhouette_score(X, df['Cluster_15'])

print(f"Silhouette Score for 15 clusters: {sil_score_15}")
```

```
C:\Users\User\anaconda3\Lib\site-packages\sklearn\cluster\
_kmeans.py:1412: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\User\anaconda3\Lib\site-packages\joblib\externals\loky\
backend\context.py:110: UserWarning: Could not find the number of
physical cores for the following reason:
invalid literal for int() with base 10: ''
Returning the number of logical cores instead. You can silence this
warning by setting LOKY_MAX_CPU_COUNT to the number of cores you want
```

```
to use.
  warnings.warn(
  File "C:\Users\User\anaconda3\Lib\site-packages\joblib\externals\
loky\backend\context.py", line 205, in _count_physical_cores
    cpu_count_physical = sum(map(int, cpu_info))
                         ^^^^^^^^^^^^^^^^^^^^^^^

Silhouette Score for 15 clusters: 0.6386329462160013

X = df[columns] #feature_matrix
y = df['price_custom_capped'] #target_variable

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

# Creating a linear regression model
lr = LinearRegression()

# Performing RFE to select the top features
rfe = RFE(estimator=lr, n_features_to_select=10)
rfe = rfe.fit(X_train, y_train)

selected_features = rfe.support_
ranking = rfe.ranking_

# checking the selected features gs
print("Selected Features: ", selected_features)

Selected Features:  [False  True  True False False  True False  True
True  True False False
 False False False  True  True False False  True False False  True
False]

# Get the column names that were selected by RFE
selected_columns = X_train.columns[selected_features]

# Filter the features in X_train and X_test using the selected columns
X_train_selected = X_train[selected_columns]
X_test_selected = X_test[selected_columns]

import statsmodels.api as sm

X_train_selected_sm = sm.add_constant(X_train_selected)

# Fit the model using statsmodels with the selected features
model = sm.OLS(y_train, X_train_selected_sm).fit()
```

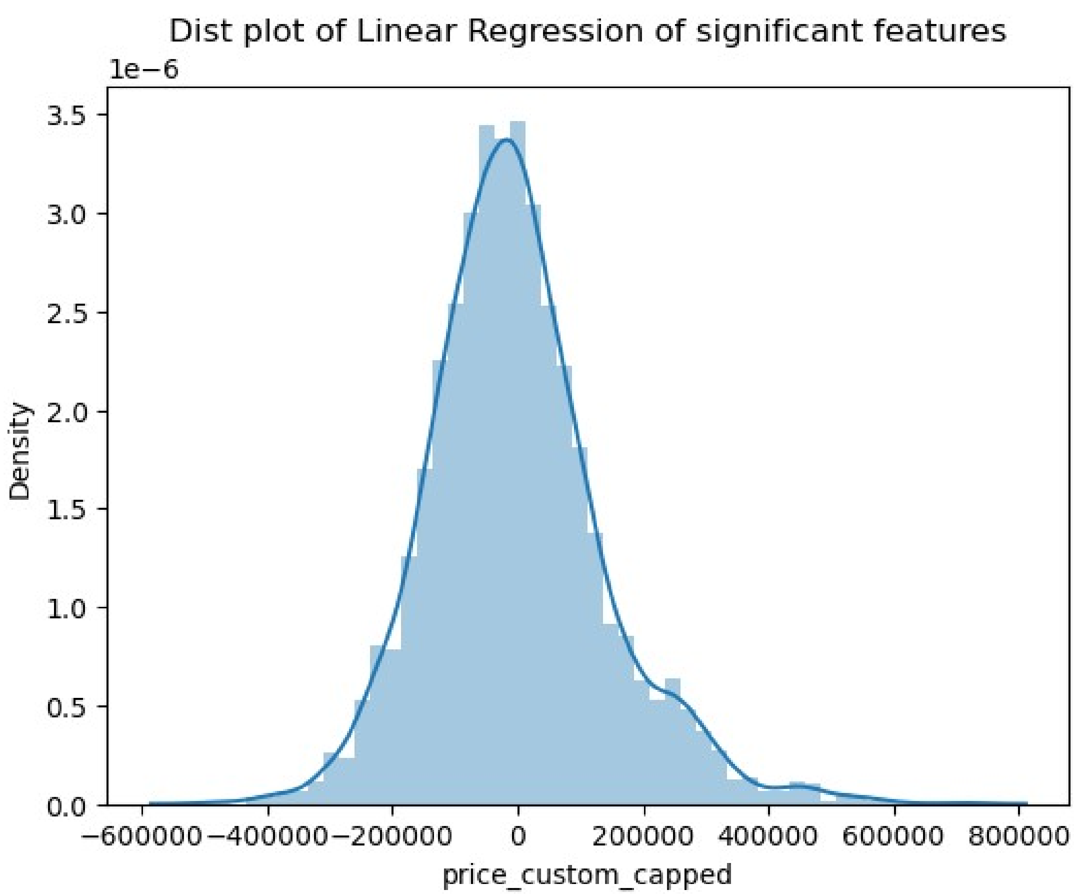
```
# Print the p-value summary
print(model.summary())

# Identifying the column names (features) with p-values > 0.05
significant_features = model.pvalues[model.pvalues <= 0.05].index
significant_features = significant_features.drop('const')

# Filter the dataset to keep only the significant features
X_train_significant = X_train_selected[significant_features]
X_test_significant = X_test_selected[significant_features]
```

```
                             OLS Regression Results

======================================================================
=========
Dep. Variable:     price_custom_capped   R-squared:
0.674
Model:                             OLS   Adj. R-squared:
0.674
Method:                  Least Squares   F-statistic:
3972.
Date:                 Wed, 18 Sep 2024   Prob (F-statistic):
0.00
Time:                         20:49:23   Log-Likelihood:             -
2.2979e+05
No. Observations:                17290   AIC:
4.596e+05
Df Residuals:                    17280   BIC:
4.597e+05
Df Model:                            9

Covariance Type:             nonrobust

======================================================================
===================
                            coef    std err          t      P>|t|
[0.025      0.975]
----------------------------------------------------------------------
-------------------
const                     -4e+07   1.14e+06    -35.150      0.000    -
4.22e+07   -3.78e+07
room_bed_capped        1.424e+04   1134.356     12.551      0.000
1.2e+04    1.65e+04
room_bath_capped      -1411.9946    765.001     -1.846      0.065    -
2911.474      87.485
ceil_capped            1689.5250   1447.948      1.167      0.243    -
1148.600    4527.650
sight_capped           7.839e+04   2031.389     38.591      0.000
7.44e+04    8.24e+04
condition_capped       5.241e+04   1810.546     28.949      0.000
```

```
4.89e+04     5.6e+04
quality_custom_capped  8.695e+04   2120.264     41.007      0.000
8.28e+04    9.11e+04
lat_capped             6.534e+05   8359.224     78.170      0.000
6.37e+05     6.7e+05
long_capped           -6.844e+04   9260.855     -7.390      0.000   -
8.66e+04   -5.03e+04
furnished_capped       1.341e+05   4464.110     30.049      0.000
1.25e+05    1.43e+05
total_rooms            1.283e+04    480.753     26.678      0.000
1.19e+04    1.38e+04
======================================================================
========
Omnibus:                     1422.258   Durbin-Watson:
2.019
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2292.542
Skew:                           0.626   Prob(JB):
0.00
Kurtosis:                       4.272   Cond. No.
6.71e+16
======================================================================
========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The smallest eigenvalue is 6.71e-26. This might indicate that
there are
strong multicollinearity problems or that the design matrix is
singular.

print(X_train_significant.columns)
print(X_test_significant.columns)

Index(['room_bed_capped', 'sight_capped', 'condition_capped',
       'quality_custom_capped', 'lat_capped', 'long_capped',
       'furnished_capped', 'total_rooms'],
      dtype='object')
Index(['room_bed_capped', 'sight_capped', 'condition_capped',
       'quality_custom_capped', 'lat_capped', 'long_capped',
       'furnished_capped', 'total_rooms'],
      dtype='object')

#Linear Regression Model

from sklearn.linear_model import LinearRegression
from sklearn import metrics

model_lr = LinearRegression()
```

```
model_lr.fit(X_train_significant,y_train)
y_pred = model_lr.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of Linear Regression of significant features')
plt.show()
```

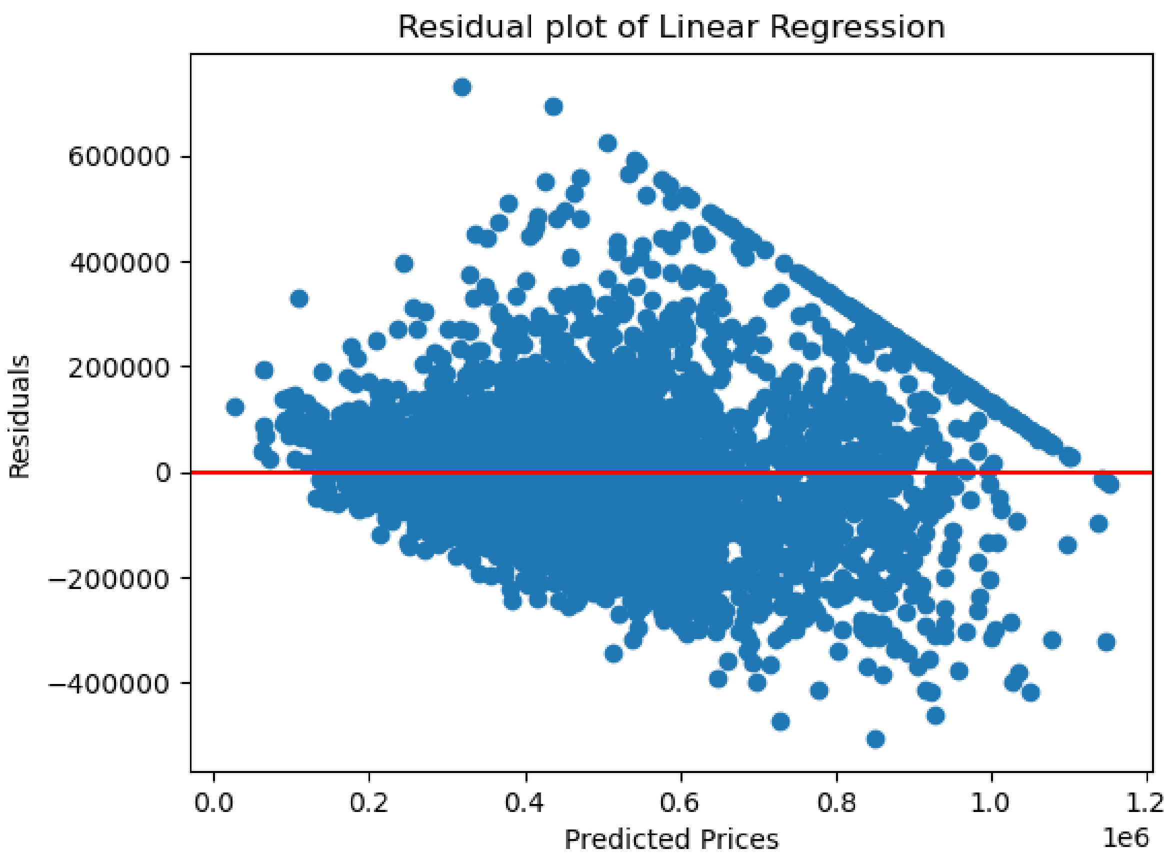
```
C:\Users\User\AppData\Local\Temp\ipykernel_2116\2020468942.py:11:
UserWarning:

`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((y_test-y_pred), bins=50)
```



```python
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

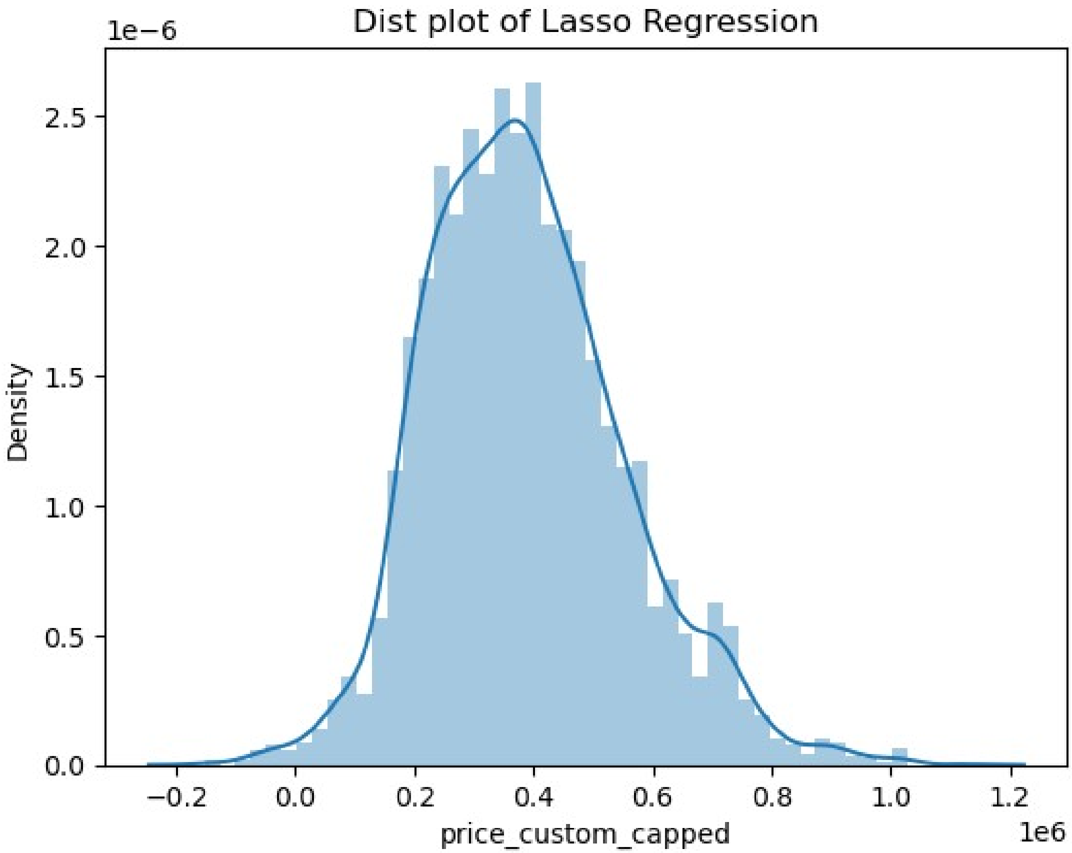
```
MAPE: 0.23027840859486132
RMSE: 140366.08372466467
R-squared: 0.6769380367254575
Adjusted R-squared: 0.6763389417541557
```

```python
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of Linear Regression")
plt.show()
```




```
#Lasso Regression Model

from sklearn.linear_model import Lasso
from sklearn import metrics

model_lm = Lasso(alpha=1)

model_lm.fit(X_train_significant,y_train)
y_pred = model_lm.predict(X_test_significant.astype(int))

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of Lasso Regression')
plt.show()

C:\Users\User\AppData\Local\Temp\ipykernel_2116\69215358.py:11:
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).
```

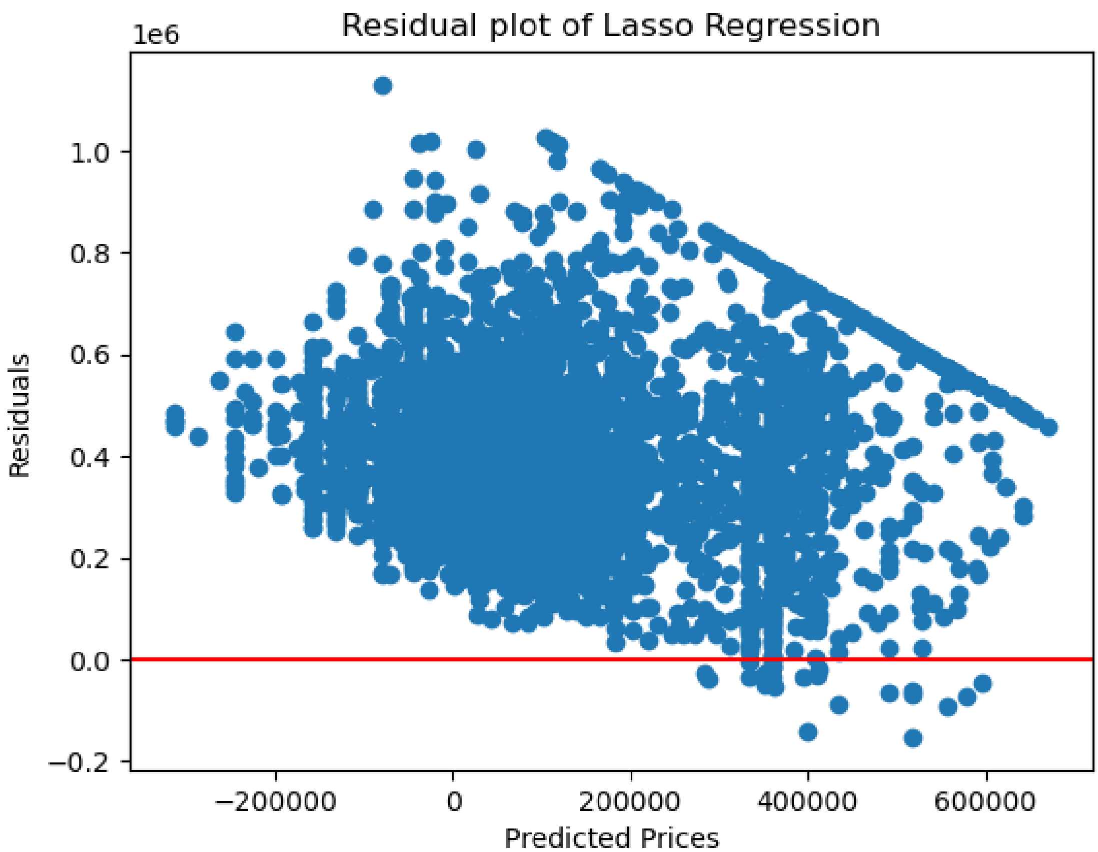
```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot((y_test-y_pred), bins=50)
```




```python
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

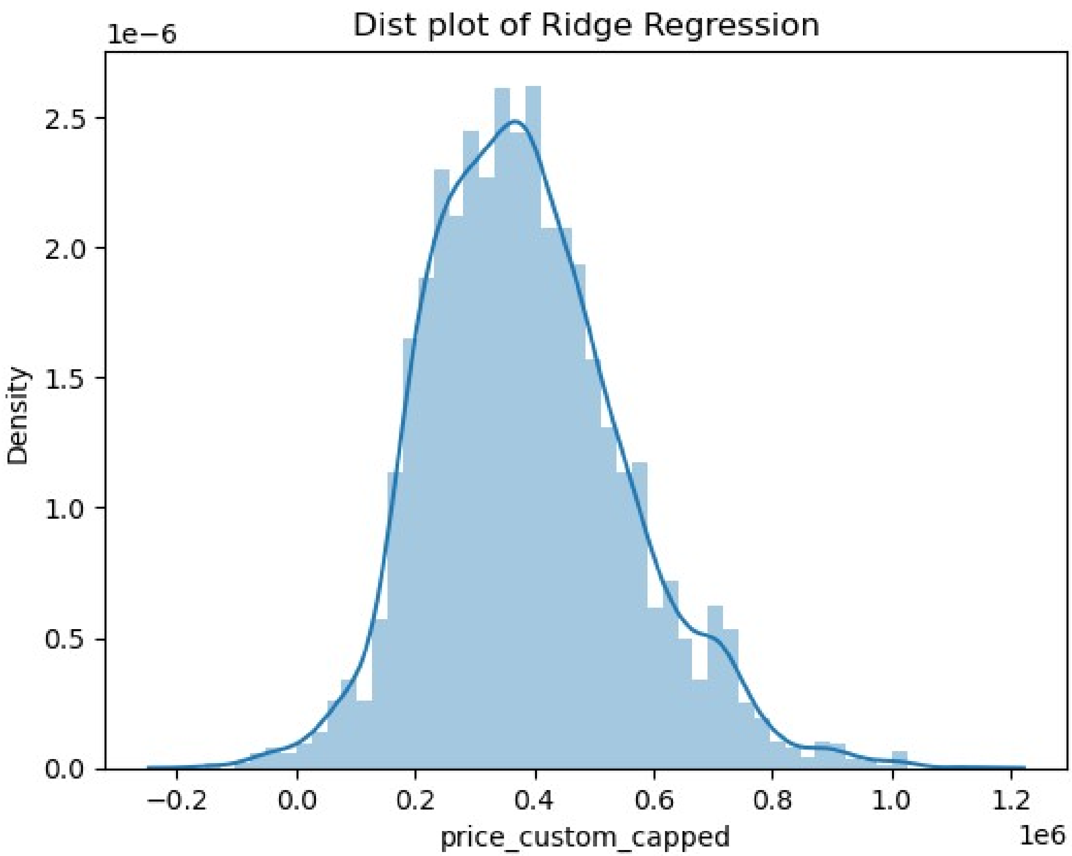
```
MAPE: 0.8450066824897876
RMSE: 421765.6100773643
```

```
R-squared: -1.9167807999805144
Adjusted R-squared: -1.9221897583485825

# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of Lasso Regression")
plt.show()
```




```
#Ridge Regression Model

from sklearn.linear_model import Ridge
from sklearn import metrics

model_rm = Ridge()

model_rm.fit(X_train_significant,y_train)
y_pred = model_rm.predict(X_test_significant.astype(int))

sns.distplot((y_test-y_pred), bins=50)
```

```
plt.title('Dist plot of Ridge Regression')
plt.show()

C:\Users\User\AppData\Local\Temp\ipykernel_2116\931011914.py:11:
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((y_test-y_pred), bins=50)
```

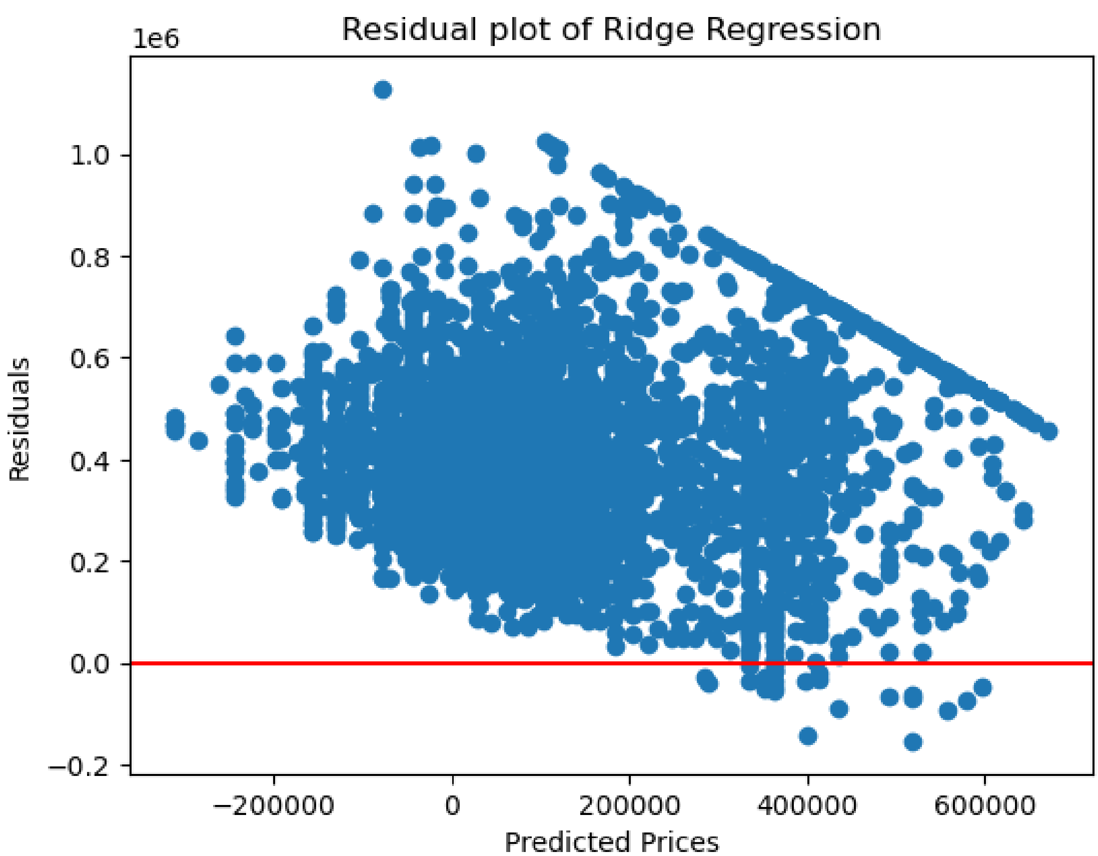



```
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
```

```
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

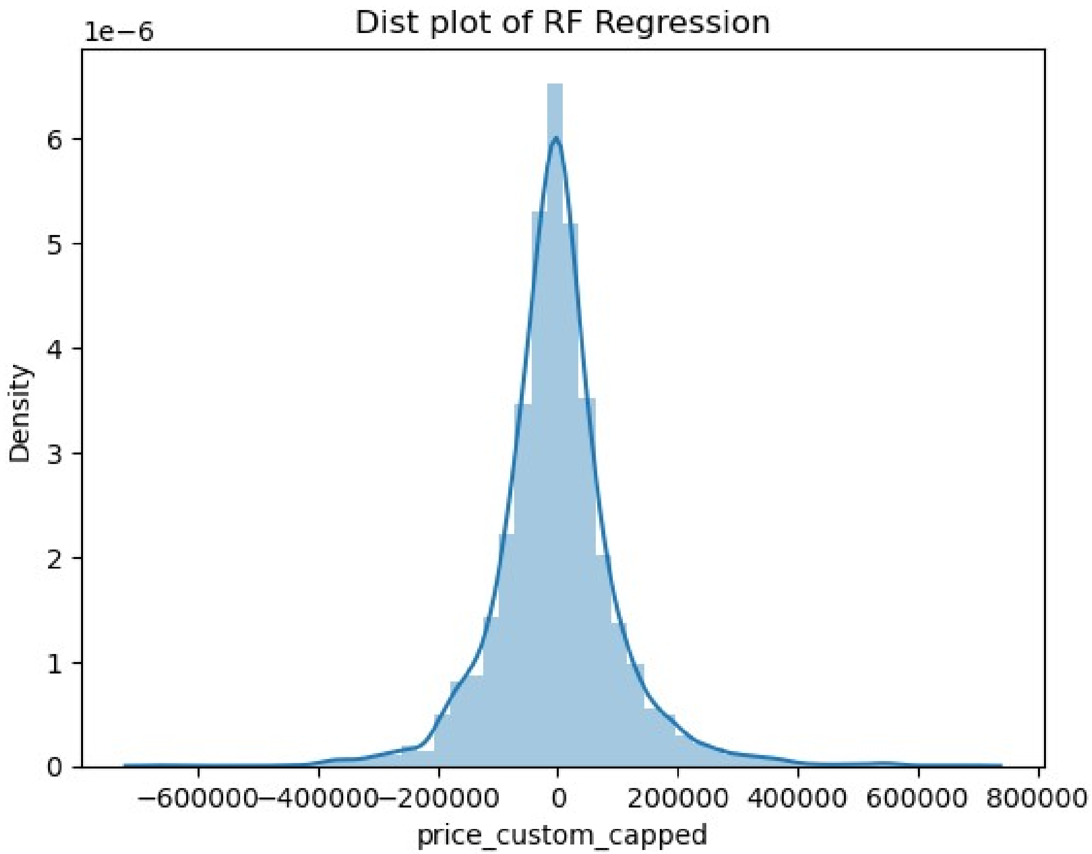
```
MAPE: 0.8422262491371693
RMSE: 420720.69196262176
R-squared: -1.9023461396592616
Adjusted R-squared: -1.9077283299970627
```

```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of Ridge Regression")
plt.show()
```



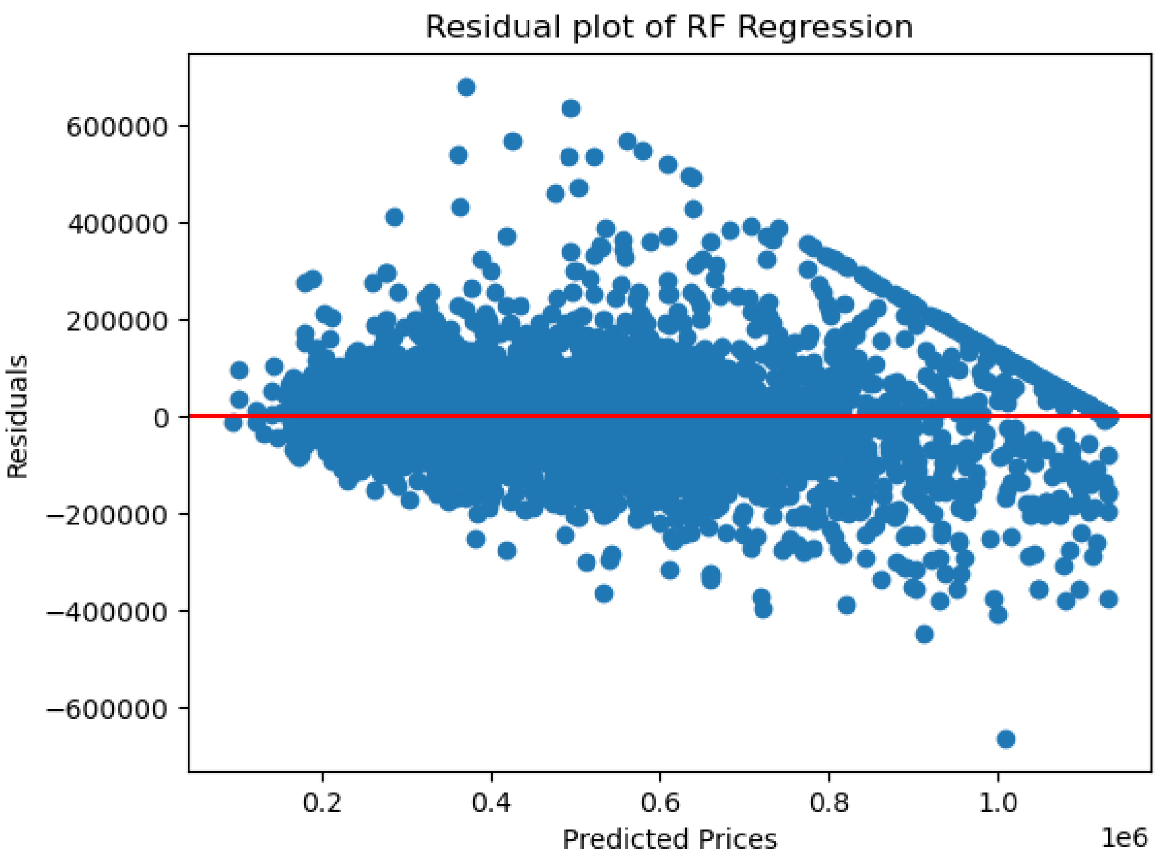
```python
#Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

model_RFR = RandomForestRegressor(n_estimators=10)
model_RFR.fit(X_train_significant, y_train)
y_pred = model_RFR.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of RF Regression')
plt.show()
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_2116\1512409299.py:9:
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((y_test-y_pred), bins=50)
```




```
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")

MAPE: 0.14952188254898813
RMSE: 100473.93960646447
R-squared: 0.8344731979636936
Adjusted R-squared: 0.8341662405190273

# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
```

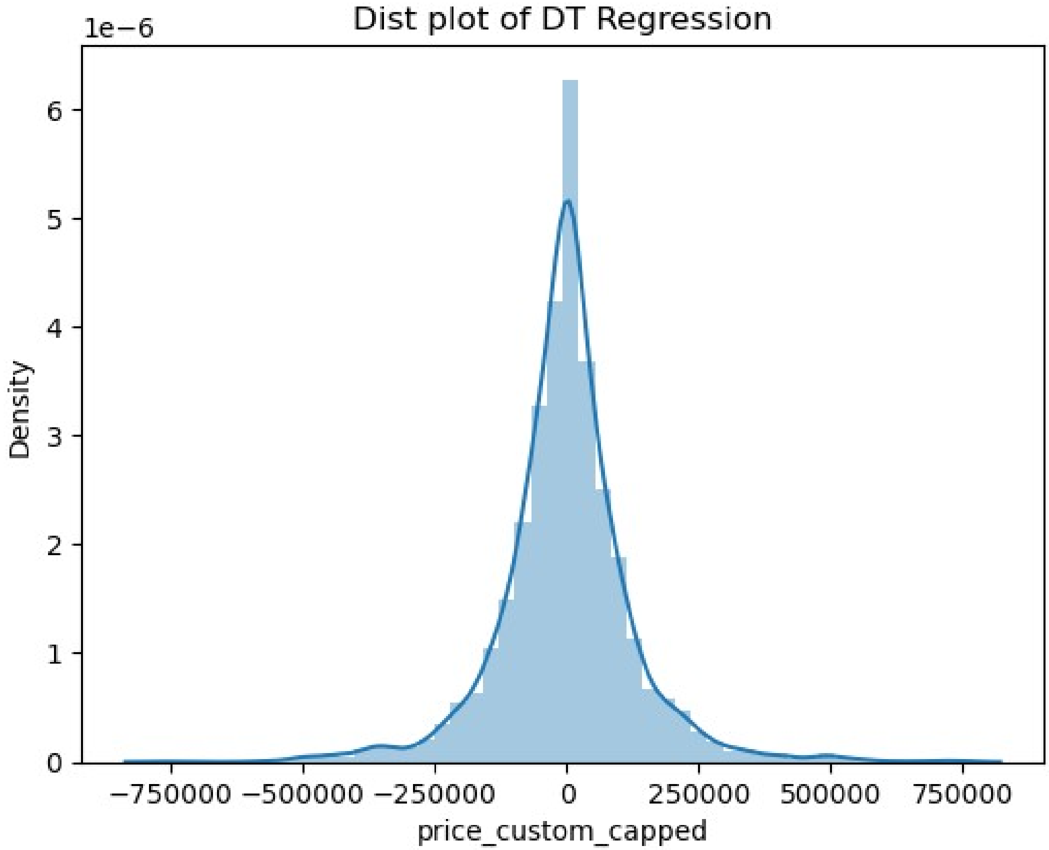
```
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of RF Regression")
plt.show()
```




```
#Decision Tree Regressor

from sklearn.tree import DecisionTreeRegressor

model_DTR = DecisionTreeRegressor()
model_DTR.fit(X_train_significant, y_train)
y_pred = model_DTR.predict(X_test_significant)

sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of DT Regression')
plt.show()
```

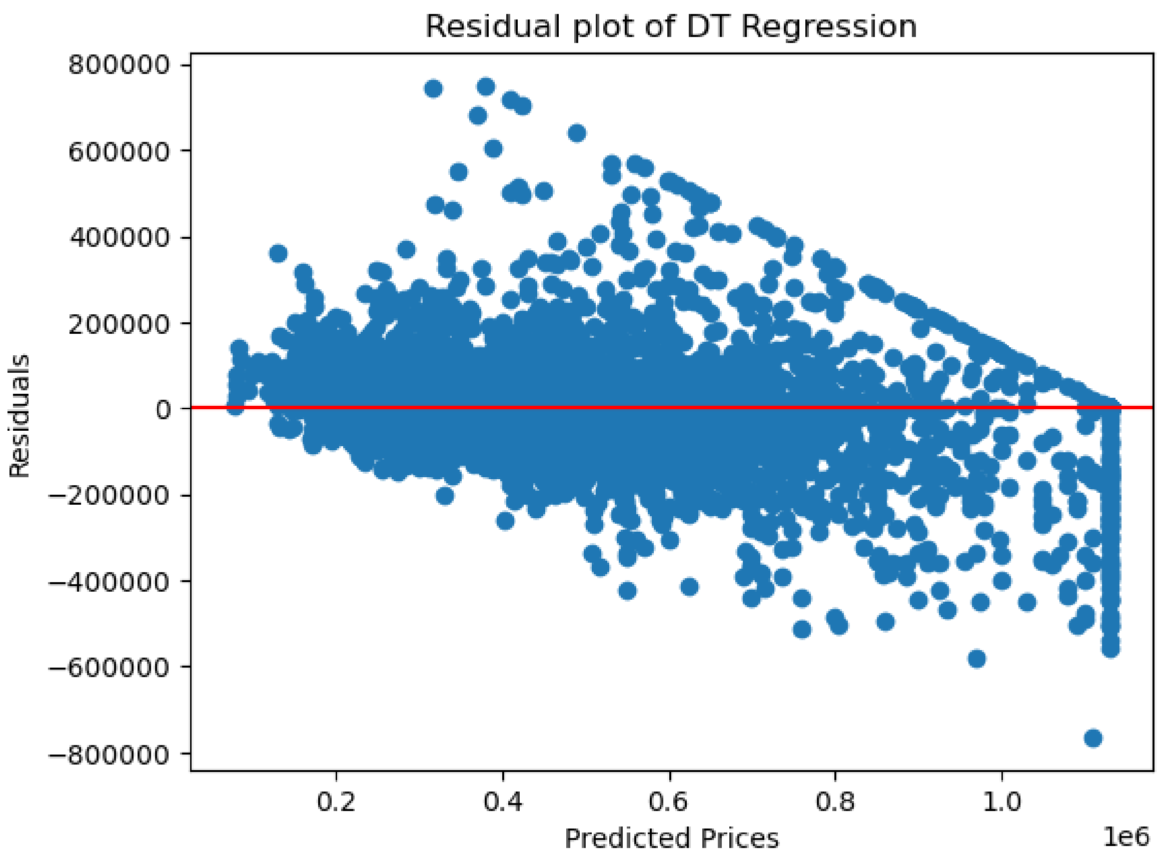
```
C:\Users\User\AppData\Local\Temp\ipykernel_2116\2007142571.py:9:
UserWarning:

`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((y_test-y_pred), bins=50)
```




```python
# Calculating metrics
import math

mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

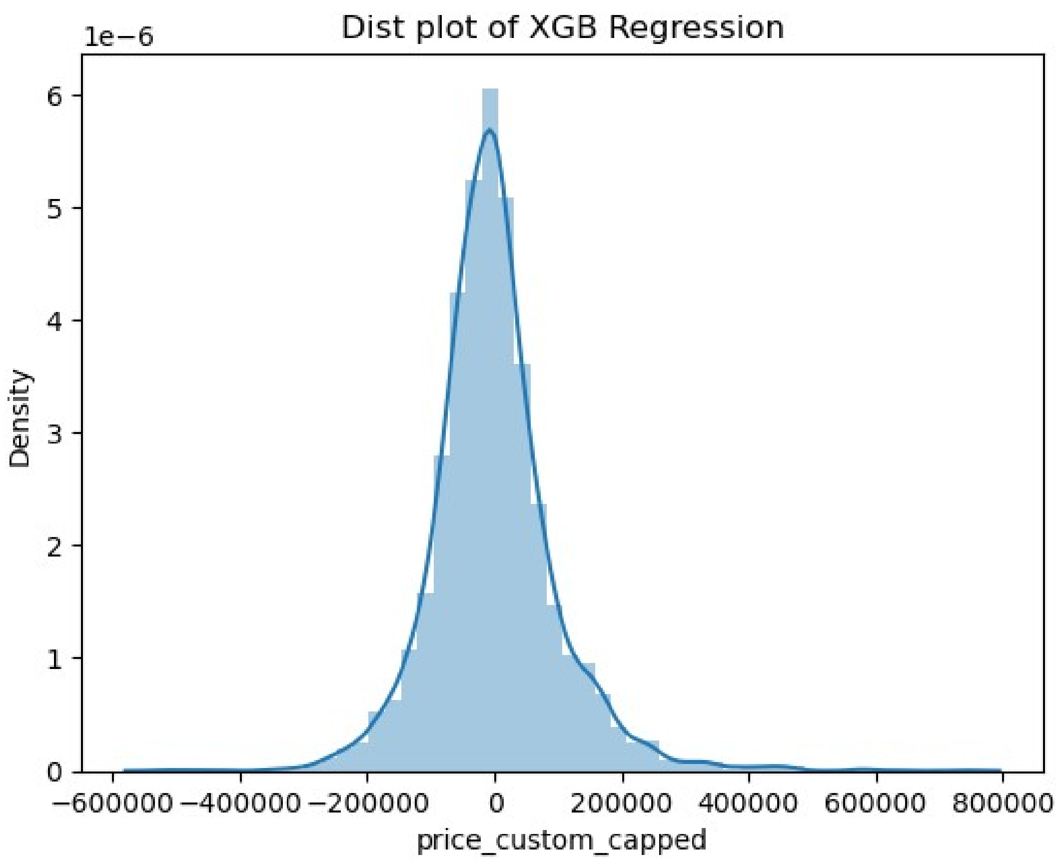
```
MAPE: 0.18418839066652734
RMSE: 126886.74426453974
R-squared: 0.7360060524643064
Adjusted R-squared: 0.7355164948425434

# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of DT Regression")
plt.show()
```




```
#XG Boost Regressor

from xgboost import XGBRegressor

model_XGB = XGBRegressor(n_estimators=100, learning_rate=0.1,
max_depth=5, random_state=42)
model_XGB.fit(X_train_significant, y_train)

y_pred = model_XGB.predict(X_test_significant)
```

```
sns.distplot((y_test-y_pred), bins=50)
plt.title('Dist plot of XGB Regression')
plt.show()
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_2116\2421365405.py:10:
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((y_test-y_pred), bins=50)
```




```
# Calculating metrics
import math
```

```
mape = metrics.mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred))
r_squared = metrics.r2_score(y_test, y_pred)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")
```

```
MAPE: 0.1479936729462726
RMSE: 96437.81560711651
R-squared: 0.8475047938990412
Adjusted R-squared: 0.8472220026035364
```

```
# Residual plot
residuals = y_test - y_pred
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of XGB Regression")
plt.show()
```


```
from sklearn.model_selection import KFold, cross_val_score

# Initialize the models
models = {
    'Linear Regression': LinearRegression(),
    'Lasso Regressiom': Lasso(alpha=1),
    'Ridge Regression': Ridge(),
    'Random Forest Regressor': RandomForestRegressor(n_estimators=10),
    'Decision Tree Regressor': DecisionTreeRegressor(),
    'XGBoost Regressor': XGBRegressor(n_estimators=100,
learning_rate=0.1, max_depth=5, random_state=42)
}

# Set up KFold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Evaluate each model
for model_name, model in models.items():
    cv_scores = cross_val_score(model, X_train_significant, y_train,
cv=kf, scoring='r2')
    print(f"{model_name} - Mean R-squared: {np.mean(cv_scores):.4f},
Std: {np.std(cv_scores):.4f}")
```

```python
# For additional metrics such as RMSE
for model_name, model in models.items():
    cv_scores = cross_val_score(model, X_train_significant, y_train,
cv=kf, scoring='neg_mean_squared_error')
    rmse_scores = np.sqrt(-cv_scores)
    print(f"{model_name} - Mean RMSE: {np.mean(rmse_scores):.4f}, Std:
{np.std(rmse_scores):.4f}")
```

```
Linear Regression - Mean R-squared: 0.6732, Std: 0.0157
Lasso Regressiom - Mean R-squared: 0.6732, Std: 0.0157
Ridge Regression - Mean R-squared: 0.6732, Std: 0.0157
Random Forest Regressor - Mean R-squared: 0.8322, Std: 0.0105
Decision Tree Regressor - Mean R-squared: 0.7270, Std: 0.0114
XGBoost Regressor - Mean R-squared: 0.8482, Std: 0.0093
Linear Regression - Mean RMSE: 143263.0332, Std: 1705.3834
Lasso Regressiom - Mean RMSE: 143263.0351, Std: 1705.2834
Ridge Regression - Mean RMSE: 143263.4317, Std: 1704.9332
Random Forest Regressor - Mean RMSE: 102742.5225, Std: 2207.5593
Decision Tree Regressor - Mean RMSE: 130857.3267, Std: 2684.4382
XGBoost Regressor - Mean RMSE: 97632.7747, Std: 2251.4550
```

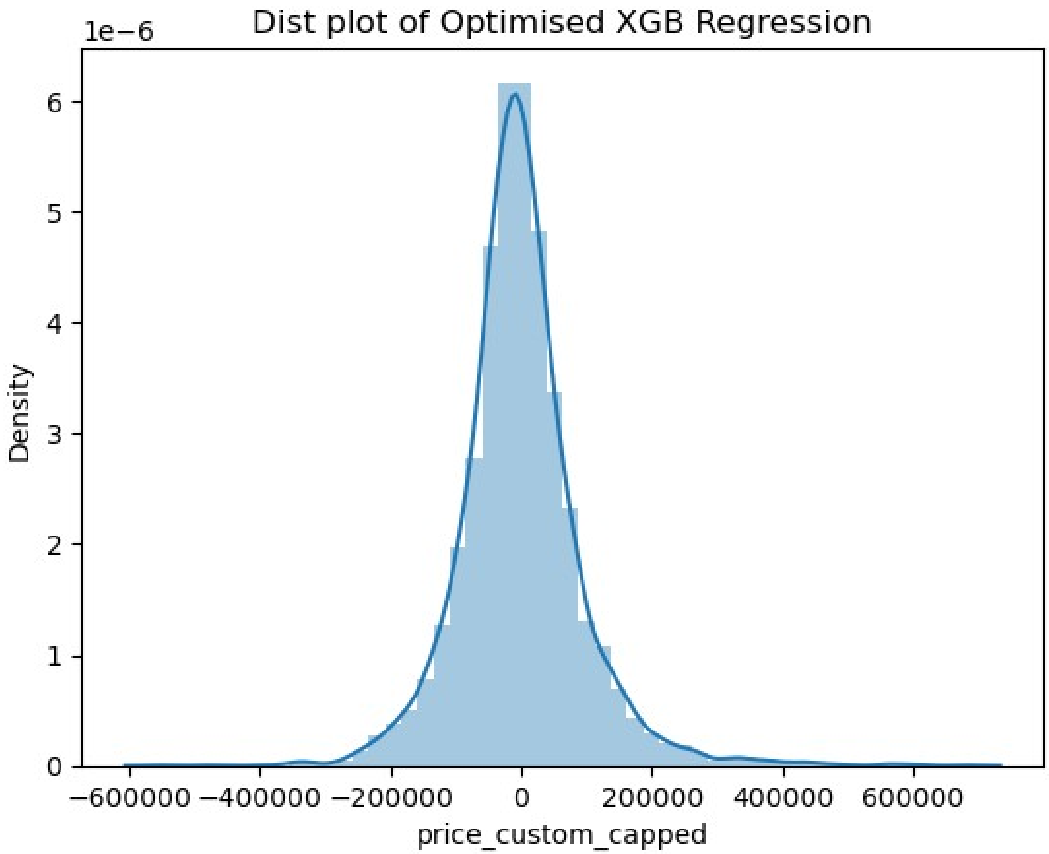
```python
# XGBoost Regressor achieves the highest mean R-squared (0.8482) and
the lowest mean RMSE (97632.7747).
# Indicating it is the best model among the evaluated ones.

# Random Forest Regressor has an R-squared of 0.8339, which is close
to XGBoost but slightly lower.
# Random Forest Regressor has the second-lowest RMSE (102284.0492).

#Randomized Search CV for XGB Regression

from sklearn.model_selection import RandomizedSearchCV

#defining the hyperparameter grid

param_distributions_xgb = {
    'n_estimators': [100,200,300,400,500],
    'max_depth': [3,4,5,6,7],
    'learning_rate': [0.01,0.05,0.1,0.15,0.2],
    'subsample': [0.6,0.7,0.8,0.9,1.0],
    'colsample_bytree': [0.6,0.7,0.8,0.9,1.0]
}

xgb_model = XGBRegressor(random_state=42)

xgb_random_search = RandomizedSearchCV(estimator=xgb_model,
param_distributions=param_distributions_xgb,
                                   n_iter=10, cv=5,
scoring='neg_mean_squared_error', random_state=42)
```

```
xgb_random_search.fit(X_train_significant,y_train)

#Get the best parameters and score
best_params = xgb_random_search.best_params_
best_score = xgb_random_search.best_score_

print("Best Parameters:", best_params)
print("Best Score(RMSE):", np.sqrt(-best_score))
```

```
Best Parameters: {'subsample': 0.6, 'n_estimators': 300, 'max_depth':
7, 'learning_rate': 0.05, 'colsample_bytree': 0.7}
Best Score(RMSE): 94209.47640942312
```

```
#XG Boost Regressor with best parameters

from xgboost import XGBRegressor

#using the optimized parameters
model_XGB_optimised = XGBRegressor(
     n_estimators=300,
     max_depth=7,
     learning_rate=0.05,
     subsample=0.6,
     colsample_bytree=0.7
)

model_XGB_optimised.fit(X_train_significant, y_train)

y_pred_optimised = model_XGB_optimised.predict(X_test_significant)

sns.distplot((y_test-y_pred_optimised), bins=50)
plt.title('Dist plot of Optimised XGB Regression')
plt.show()
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_2116\4085758626.py:18:
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((y_test-y_pred_optimised), bins=50)
```




```
# Calculating metrics for optimised models
import math

mape = metrics.mean_absolute_percentage_error(y_test,
y_pred_optimised)
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_pred_optimised))
r_squared = metrics.r2_score(y_test, y_pred_optimised)
adjusted_r_squared = 1 - (1-r_squared) * (len(y_test)-1)/(len(y_test)-
X_test_significant.shape[1]-1)

print(f"MAPE: {mape}")
print(f"RMSE: {rmse}")
print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adjusted_r_squared}")

MAPE: 0.14110298511181235
RMSE: 92608.10262103587
R-squared: 0.859376004073238
Adjusted R-squared: 0.8591152270756919

# Residual plot
residuals = y_test - y_pred_optimised
plt.scatter(y_pred_optimised, residuals)
```

```
plt.axhline(y=0, color='r', linestyle='-')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual plot of optimised XGB Regression")
plt.show()
```