House Price Prediction Using Linear Regression

In [8]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pylab
import math
import scipy.stats as stats
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
```

In [9]:
```
pd.set_option("display.max_rows",None)
pd.set_option("display.max_columns",None)
```

SOME INFO RELATED TO DATASET

Price: The price of the house.

Area: The total area of the house in square feet.

Bedrooms: The number of bedrooms in the house.

Bathrooms: The number of bathrooms in the house.

Stories: The number of stories in the house.

Mainroad: Whether the house is connected to the main road (Yes/No).

Guestroom: Whether the house has a guest room (Yes/No).

Basement: Whether the house has a basement (Yes/No).

Hot water heating: Whether the house has a hot water heating system (Yes/No).

Airconditioning: Whether the house has an air conditioning system (Yes/No).

Parking: The number of parking spaces available within the house.

Prefarea: Whether the house is located in a preferred area (Yes/No).

Furnishing status: The furnishing status of the house (Fully Furnished, Semi-Furnished, Unfurnished).

In [12]:
```
df = pd.read_csv("Housing.csv")
```

In [13]:
```
df.head()
```

Out[13]:

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

In [14]:
```
df.shape
```

Out[14]:
```
(545, 13)
```

In [15]:
```
df.columns
```

Out[15]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

In [22]:
```
df.describe()
```

Out[22]:

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

Checking For Null Values

In [114…
```
df.isnull().sum()
```

Out[114…
```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

In [122…
```
# plt.figure(figsize=(6,4))
# sns.heatmap(df.isnull(),cbar=False,linewidths=0.5,cmap="viridis")
# plt.title("Checking For Missing Values")
# plt.show()
```

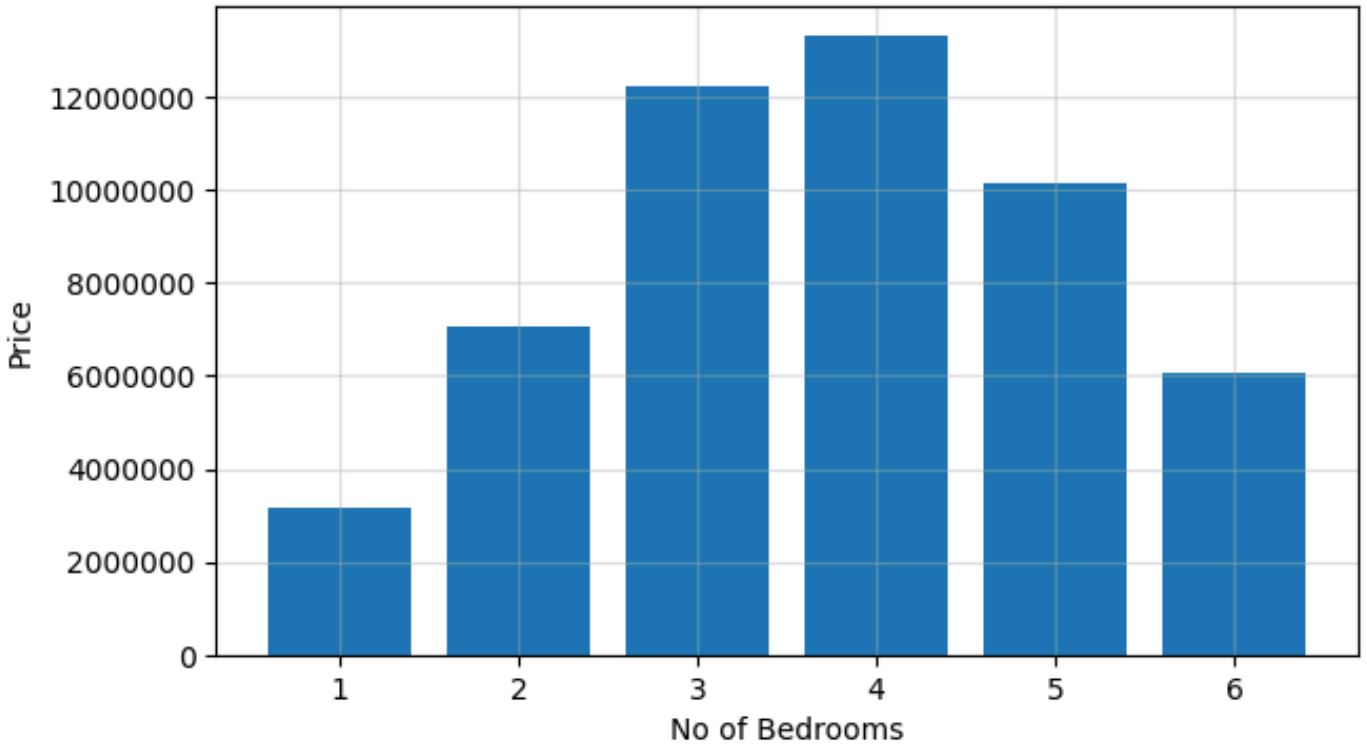
EDA(Exploratory Data Analysis)

price vs bedrooms

In [32]:
```
plt.figure(figsize=(7,4))
plt.bar(df["bedrooms"],df["price"])
plt.ticklabel_format(style="plain",axis="y")
plt.grid(alpha=0.4)
```

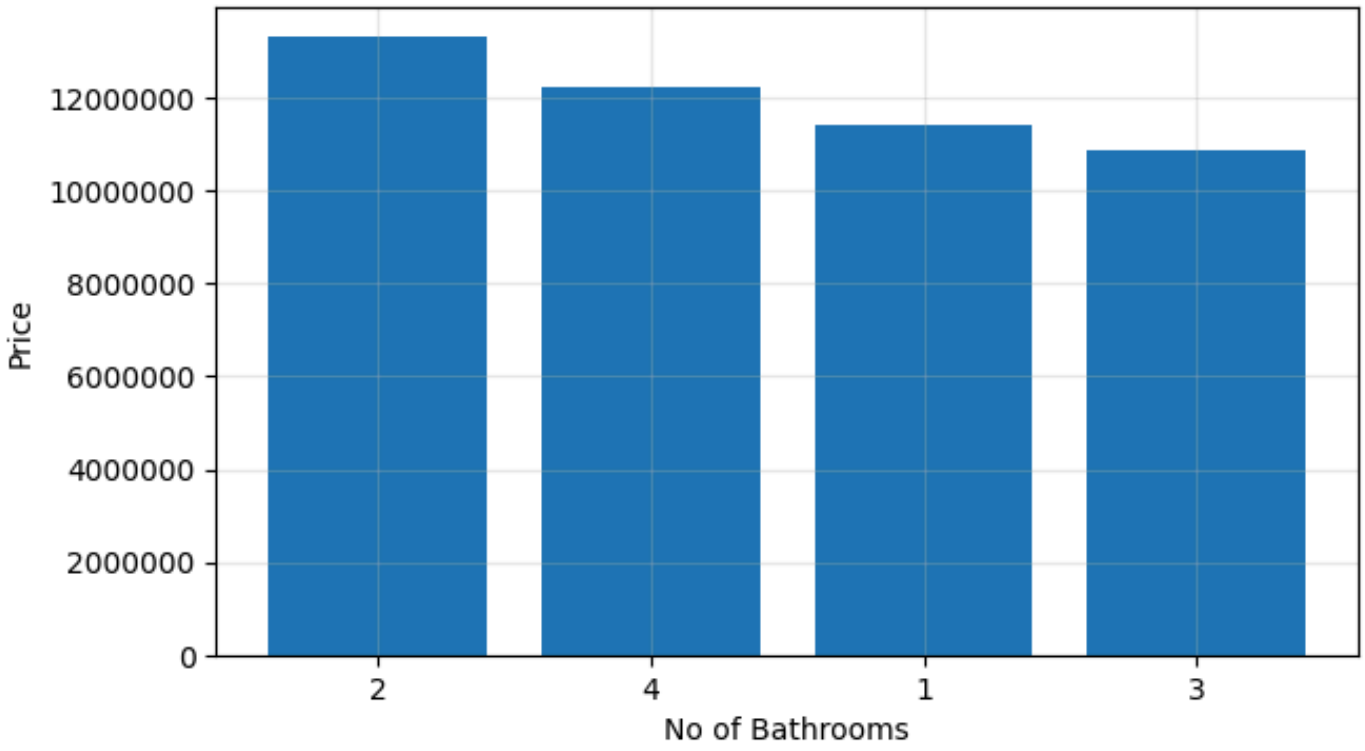
```
plt.ylabel("Price")
plt.xlabel("No of Bedrooms")
```

Out[32]: Text(0.5, 0, 'No of Bedrooms')



price vs bathrooms

In [34]:
```
plt.figure(figsize=(7,4))
plt.bar(df["bathrooms"].astype(str),df["price"])
plt.xlabel("No of Bathrooms")
plt.ylabel("Price")
plt.grid(alpha=0.3)
plt.ticklabel_format(style="plain",axis="y")
```



What is the relationship between the area of a house and its price?

In [123…

```
plt.scatter(df["area"],df["price"],alpha=0.4)
plt.xlabel("Area")
plt.ylabel("Price")
plt.title("Price vs Area")
```

Out[123…

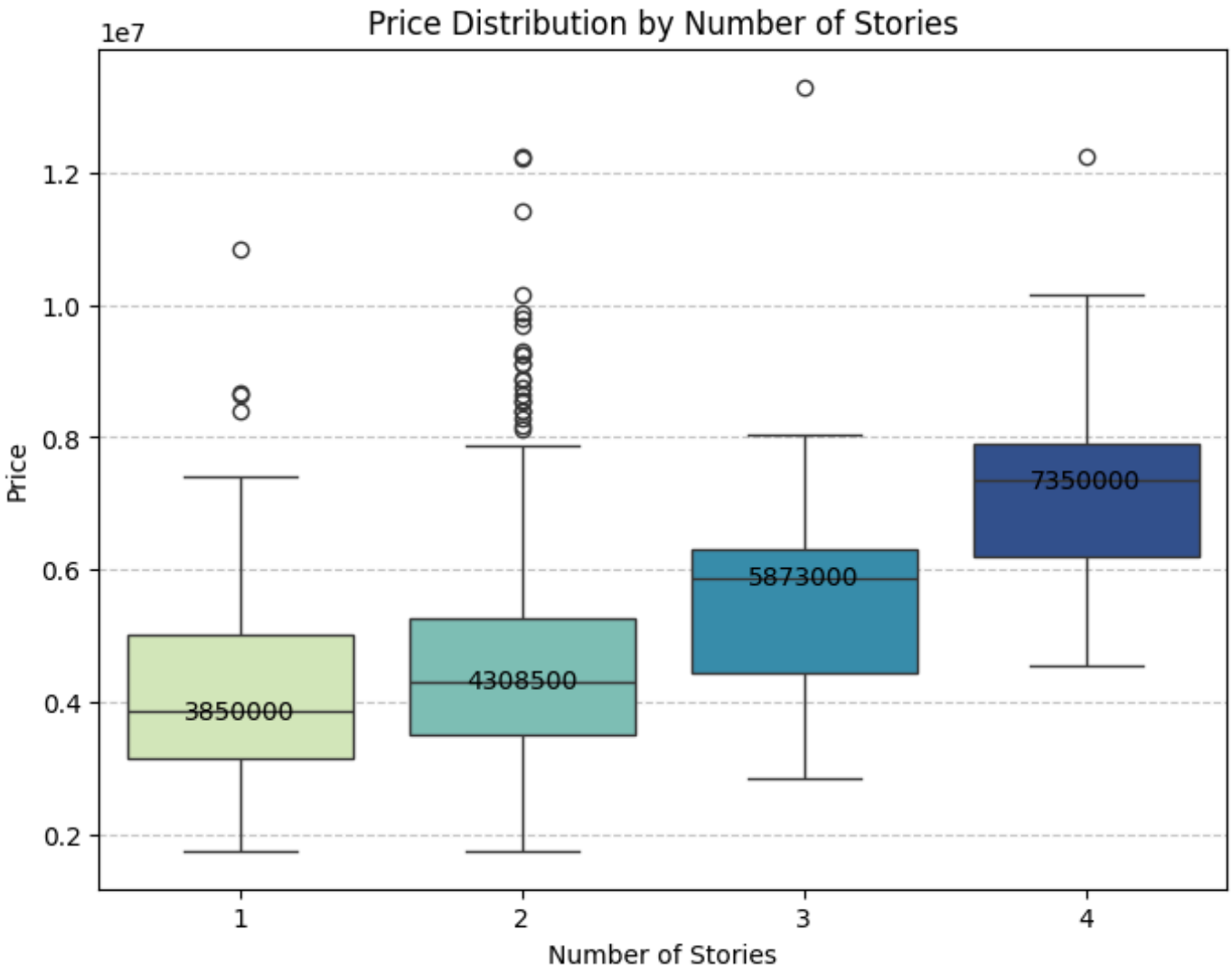
```
Text(0.5, 1.0, 'Price vs Area')
```

Houses with areas between 2000 to 5000 sq ft are somewhat clustered, but there is still noticeable variability in prices. Some houses in this range have higher prices, indicating that factors other than area, such as location or amenities, are influencing the price.

How do the number of stories influence the price of a house?

In [38]:
```
median_prices = df.groupby("stories")["price"].median()
plt.figure(figsize=(8, 6))  # Set figure size
sns.boxplot(x="stories", y="price",data=df, palette="YlGnBu")  # Box plot

for i,median in enumerate(median_prices):
    plt.text(i,median,f"{median:.0f}",ha="center",va="center",color="black",fontsize=1

plt.xlabel("Number of Stories")
plt.ylabel("Price")
plt.title("Price Distribution by Number of Stories")
plt.grid(axis='y', linestyle='--', alpha=0.7)  # Optional grid
plt.show()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\844103065.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="stories", y="price",data=df, palette="YlGnBu")  # Box plot
```



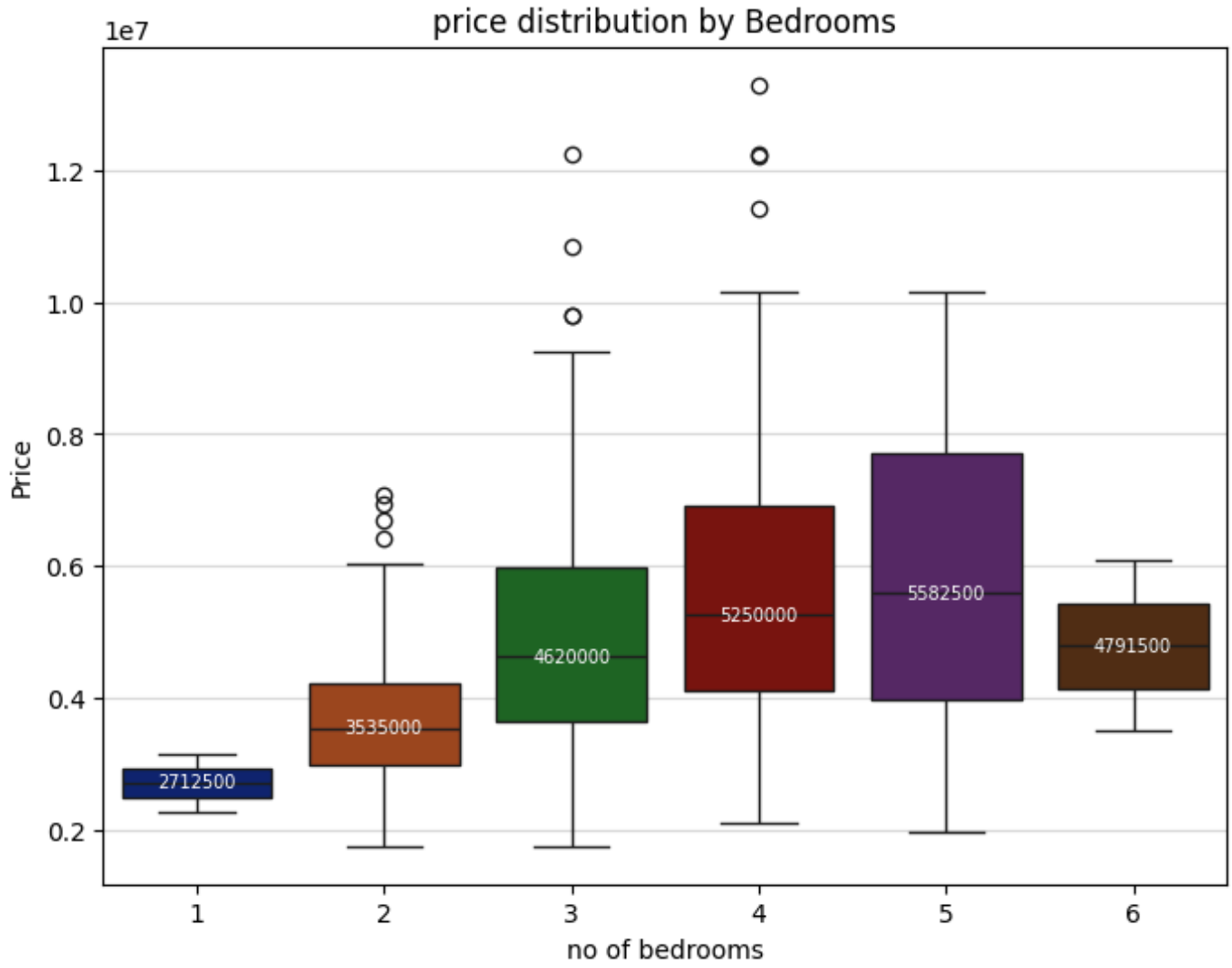
As the number of stories increases, the median price also increases, suggesting that larger or multi-story houses tend to have higher prices. This could indicate that more stories are associated with larger or more luxurious properties.

How does the number of bedrooms and bathrooms affect the price of a house?

price vs No of Bedrooms

In [41]:
```
median_prices = df.groupby("bedrooms")['price'].median()
plt.figure(figsize=(8,6))
sns.boxplot(x="bedrooms",y="price",data=df,palette="dark")

for i,median in enumerate(median_prices):
    plt.text(i,median,f"{median:.0f}",ha="center",va="center",color="white",fontsize=7

plt.xlabel("no of bedrooms")
plt.ylabel("Price")
plt.title("price distribution by Bedrooms")
plt.grid(axis="y",alpha=0.5)
plt.show()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\2835468863.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="bedrooms",y="price",data=df,palette="dark")
```



As the number of bedrooms increases, the price generally rises, but for houses with 6 bedrooms, the price drops below that of houses with 4 or 5 bedrooms. This could indicate that houses with 6 bedrooms may not be as desirable or are in less premium locations, potentially making them less expensive despite the higher bedroom count.

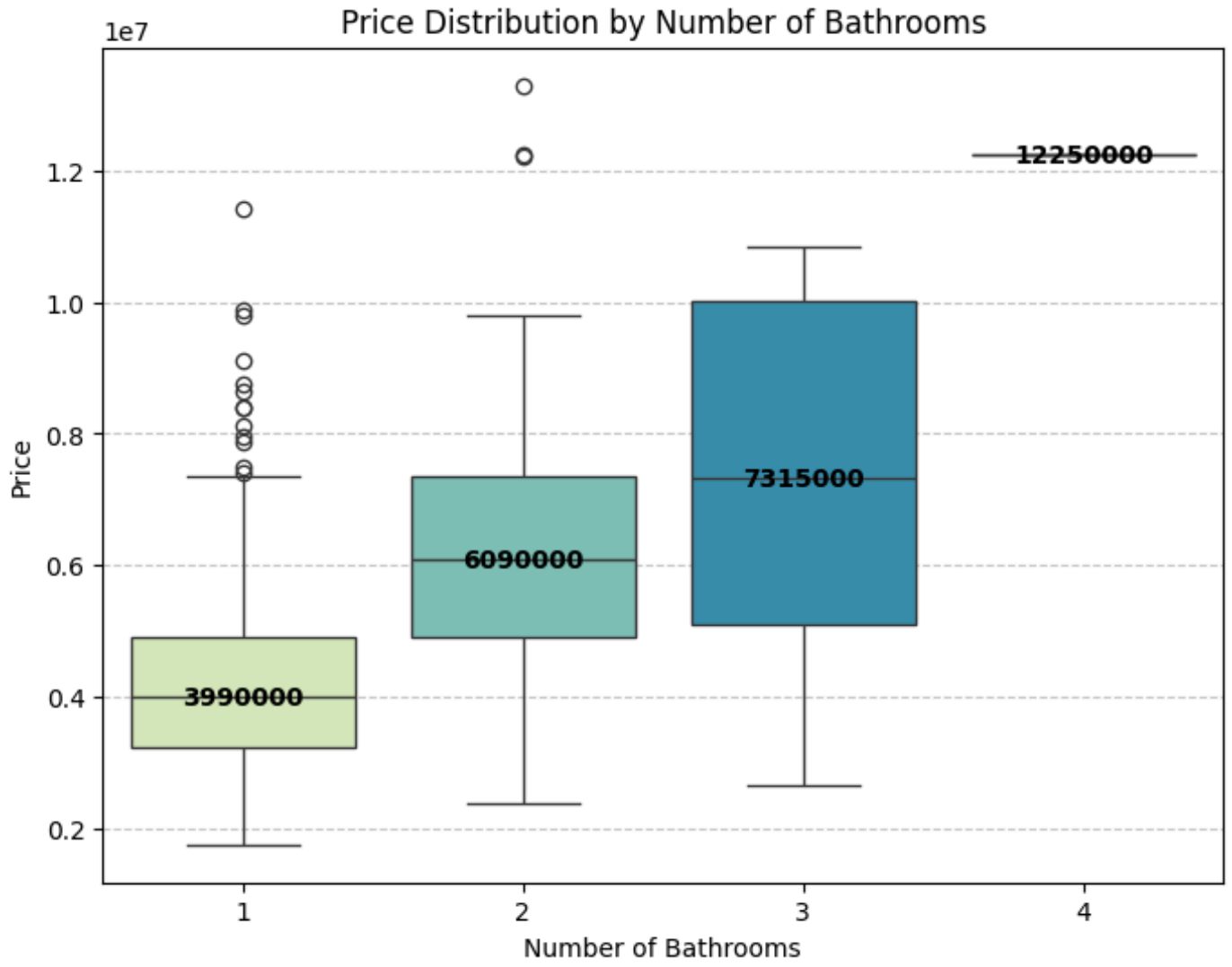
Price vs No of Bathrooms

In [117…

```
median_prices = df.groupby("bathrooms")["price"].median()


plt.figure(figsize=(8, 6))
sns.boxplot(x="bathrooms", y="price", data=df, palette="YlGnBu")

for i, median in enumerate(median_prices):
    plt.text(i, median, f"{median:.0f}", ha="center", va="center", fontsize=10, color=


plt.xlabel("Number of Bathrooms")
```

```
plt.ylabel("Price")
plt.title("Price Distribution by Number of Bathrooms")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\3088255066.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="bathrooms", y="price", data=df, palette="YlGnBu")
```



As the number of bathrooms increases, the price also increases significantly, suggesting that more bathrooms are associated with larger or more luxurious properties, which are priced higher.

In [116…

```
df["bathrooms"].value_counts()
# There is only one house with 4 bathrooms in the dataset. Boxplots require multiple d
# so no box is shown for this category."
```

Out[116…
```
bathrooms
1    401
2    133
3     10
4      1
Name: count, dtype: int64
```

What are the correlations between numerical features like area, bedrooms, bathrooms,stories,parking and price?

In [124…
```
num_df = df[["price","bedrooms","bathrooms","stories","parking","area"]]
plt.figure(figsize=(10,6))
correlation_matrix = num_df.corr()

sns.heatmap(correlation_matrix,annot=True,linewidths=0.5,fmt=".2f",cmap="YlGnBu")
plt.title("Correlation")
```

Out[124…
```
Text(0.5, 1.0, 'Correlation')
```

The correlation between price and area (0.54) is the highest, indicating a moderate positive relationship, where larger houses tend to have higher prices. Price and bathrooms (0.52) also show a moderate positive correlation, suggesting that houses with more bathrooms generally have higher prices. Price and stories (0.42) exhibit a moderate positive relationship, implying that multi-story houses are priced higher. On the other hand, price and bedrooms (0.37) and price and parking (0.38) show weaker positive correlations, indicating that while more bedrooms

and parking spaces are associated with higher prices, their influence is less significant compared to area and bathrooms.

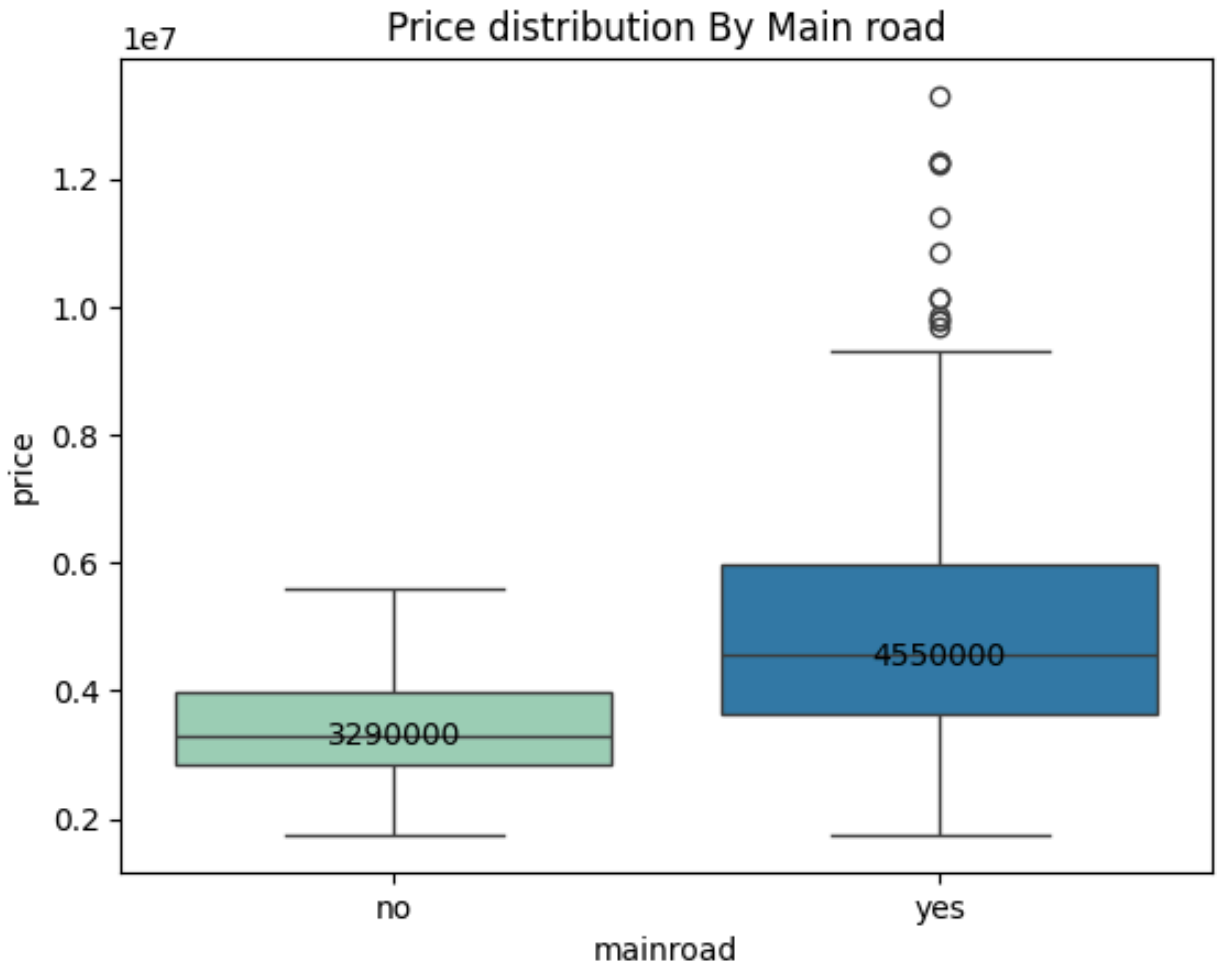
Do houses connected to the main road have higher prices compared to those that are not?

In [50]:
```
with_mainroad = df[df["mainroad"] == "yes"]
print("with_mainroad_price   ",with_mainroad["price"].median())

without_mainroad = df[df["mainroad"] == "no"]
print("without_mainroad_price   ",without_mainroad["price"].median())
```

```
with_mainroad_price    4550000.0
without_mainroad_price    3290000.0
```

In [51]:
```
median_prices = df.groupby("mainroad")["price"].median()
sns.boxplot(x="mainroad",y="price",data=df,palette="YlGnBu",order=["no","yes"])
for i,median in enumerate(median_prices):
    plt.text(i,median,f"{median:.0f}",ha="center",va="center",fontsize=10)

plt.title("Price distribution By Main road")
plt.show()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\2488008257.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="mainroad",y="price",data=df,palette="YlGnBu",order=["no","yes"])
```

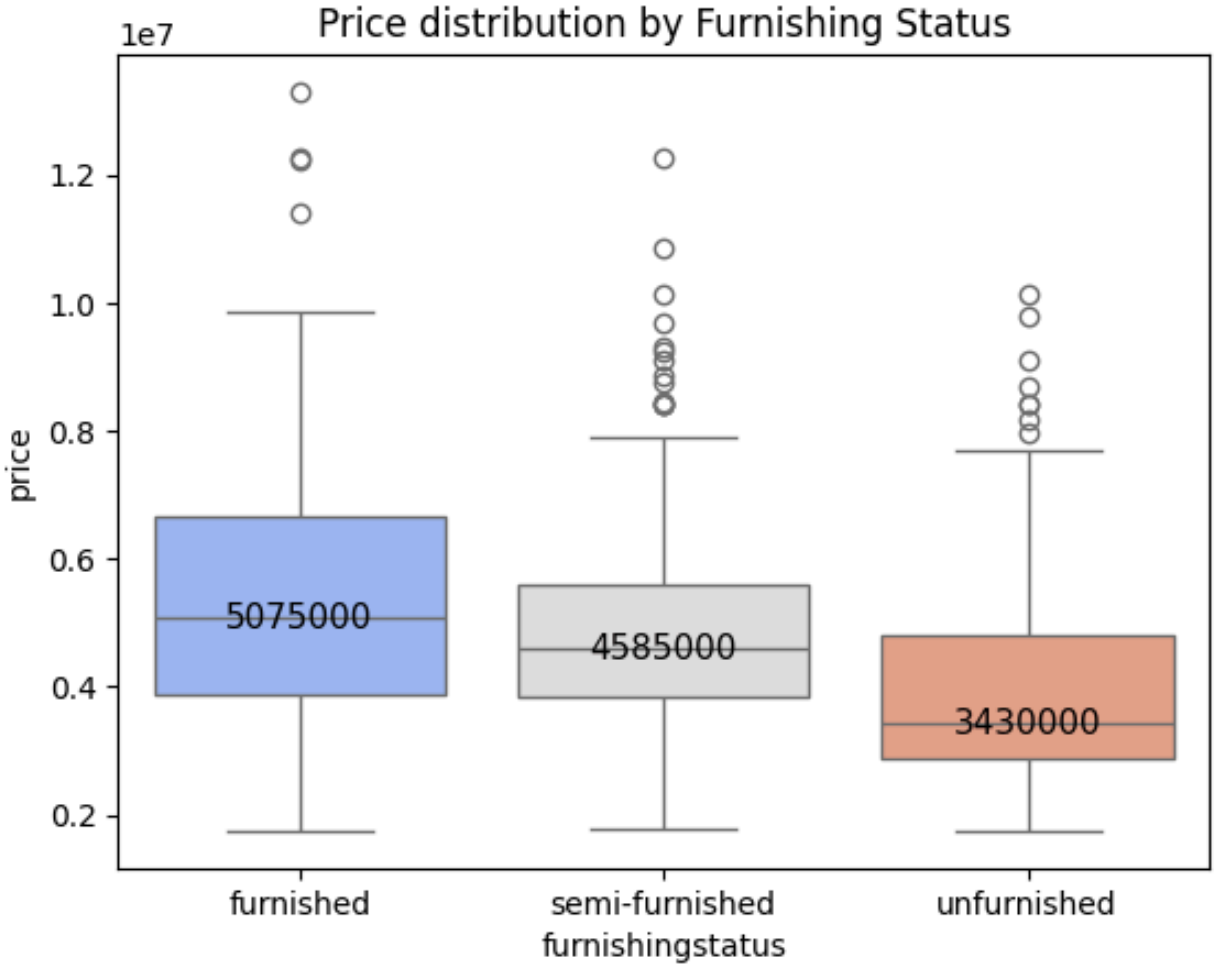


Houses connected to the main road have higher prices compared to those not connected, indicating that location and accessibility significantly influence the property's value.

Is there a significant difference in house prices based on furnishing status (Fully Furnished, Semi-Furnished, Unfurnished)?

In [53]:
```
median_prices = df.groupby("furnishingstatus")["price"].median()
sns.boxplot(x="furnishingstatus",y="price",data=df,palette="coolwarm")
for i,median in enumerate(median_prices):
    plt.text(i,median,f"{median:.0f}",ha="center",va="center",fontsize=11)

plt.title("Price distribution by Furnishing Status")
plt.show()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\2886748074.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="furnishingstatus",y="price",data=df,palette="coolwarm")
```



Furnishing status shows a clear impact on house prices: Fully Furnished houses are priced the highest (5,075,000), followed by Semi-Furnished houses (4,585,000), and Unfurnished houses have the lowest prices (3,430,000), reflecting the added value of furnishings.

In [54]:
```
# Does having a guestroom or basement increase the price of a house?
```

In [55]:
```
# Are houses with air conditioning or hot water heating systems priced higher than tho
```

In [56]:
```
# Do houses in preferred areas (`prefarea`) have significantly higher prices compared
```

What are the characteristics of the most expensive houses in the dataset (e.g., area, stories, amenities)?

In [58]:
```
# high_price = df[df["price"] >= df["price"].quantile(0.9)]
# low_price = df[df["price"] <= df["price"].quantile(0.1)]
# normal_price = df[(df["price"] > df["price"].quantile(0.1)) & (df["price"] < df["pri
```

In [59]:
```
# pd,cut()
```

In [60]:
```
# pd.cut() is a function in Pandas used to segment and sort data values into discrete
# This is often used when you want to categorize continuous numerical data into distin
```

In [61]:
```
low_price = df["price"].quantile(0.1)
high_price = df["price"].quantile(0.9)
```

In [62]:
```
bins = [-float("inf"),low_price,high_price,float("inf")]
```

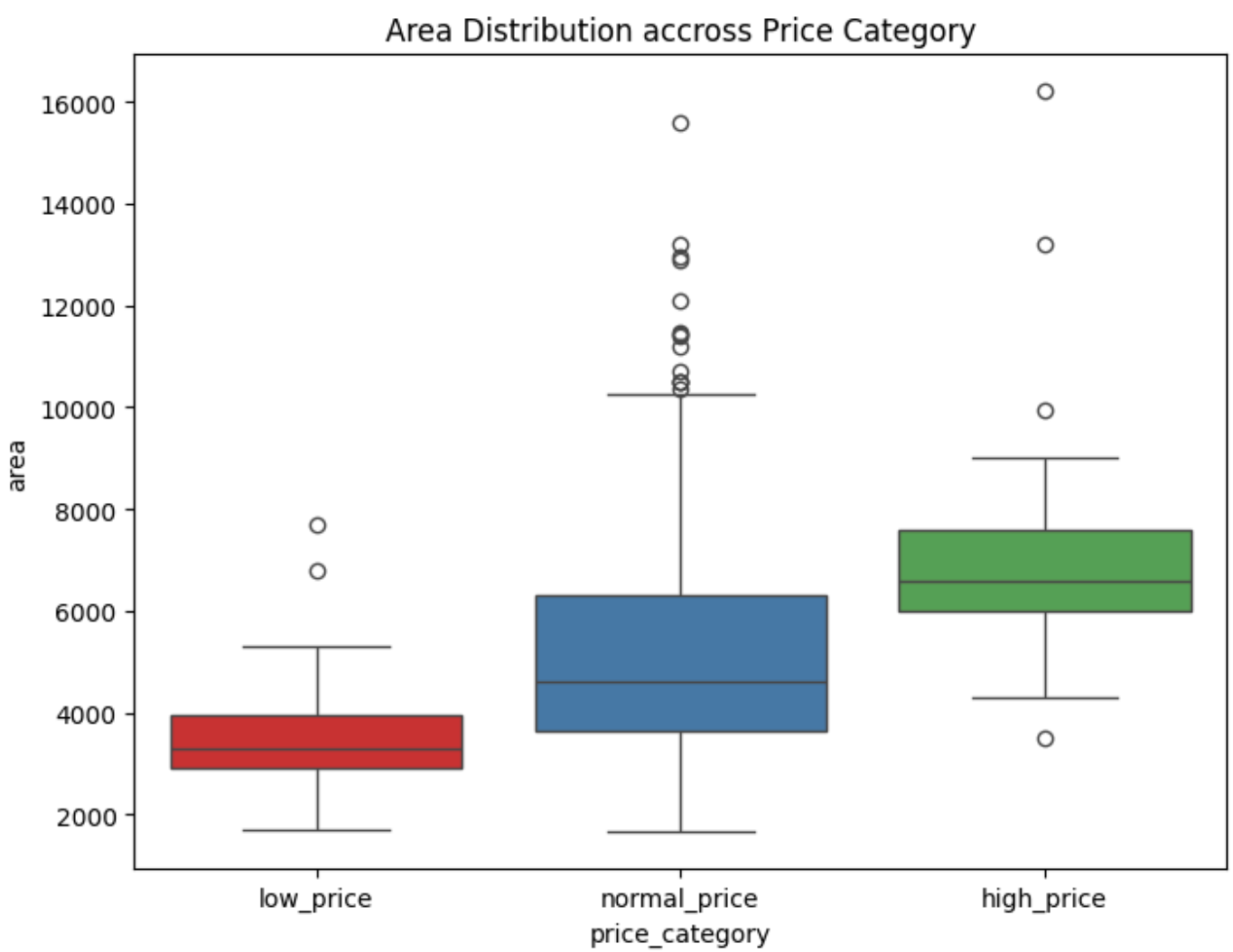
In [63]:
```
df_copy = df.copy()
df_copy["price_category"] = pd.cut(df_copy["price"], bins=bins , labels=["low_price","
df_copy.head()
```

Out[63]:

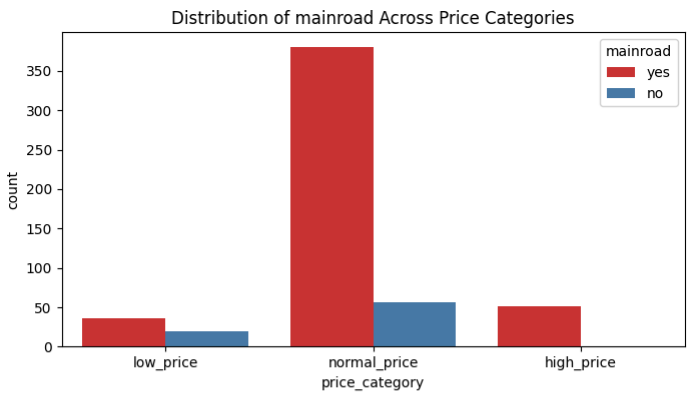
	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwate
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

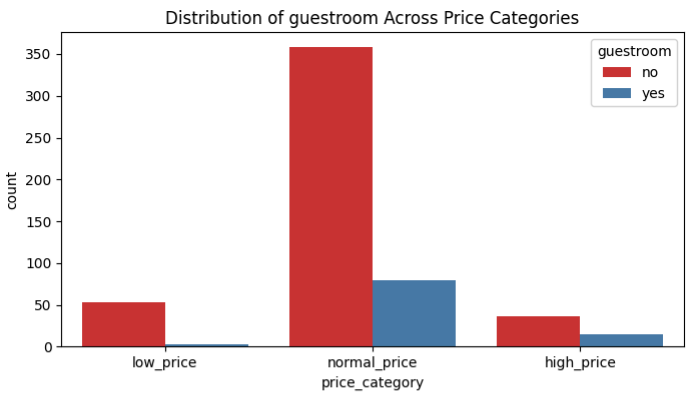
In [64]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x="price_category",y="area",data=df_copy,palette="Set1")
plt.title("Area Distribution accross Price Category")
plt.show()
```

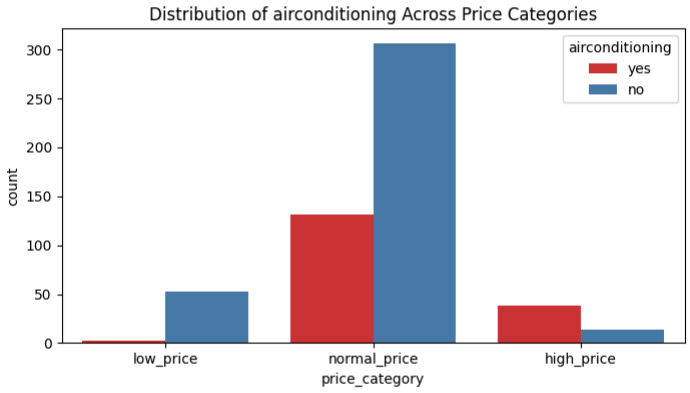
```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\54573014.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="price_category",y="area",data=df_copy,palette="Set1")
```

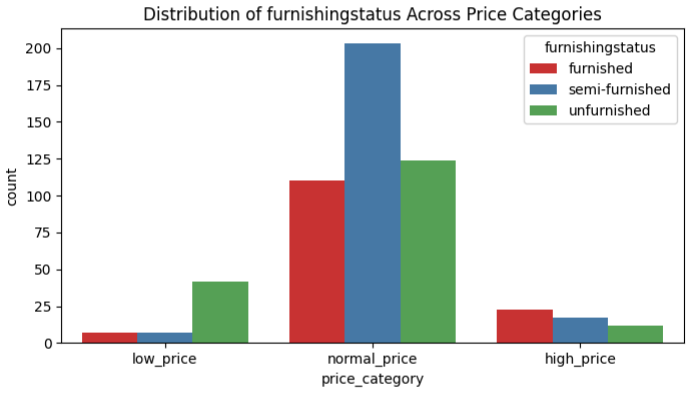


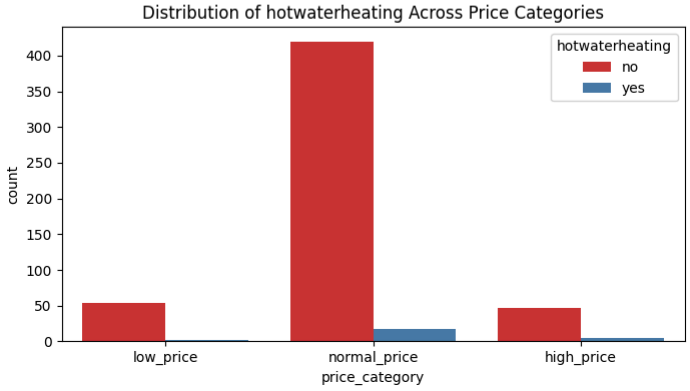
In [65]:
```
amenities = ["mainroad", "guestroom", "airconditioning", "furnishingstatus","hotwaterh
plt.figure(figsize=(14,12))
for i,amenity in enumerate(amenities,start=1):
    plt.subplot(3,2,i)
    sns.countplot(x="price_category", hue=amenity, data=df_copy, palette="Set1")
    plt.title(f"Distribution of {amenity} Across Price Categories")
plt.tight_layout()
plt.show()
```

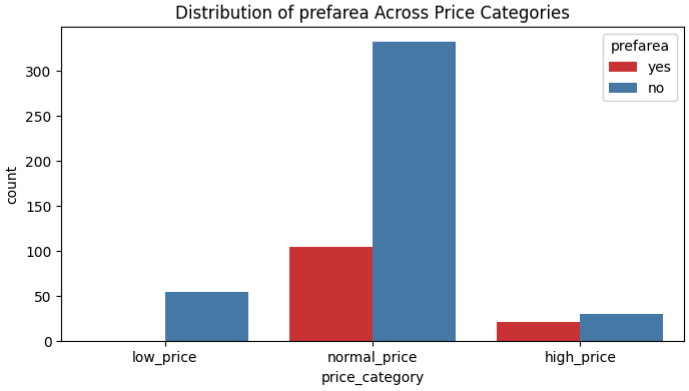








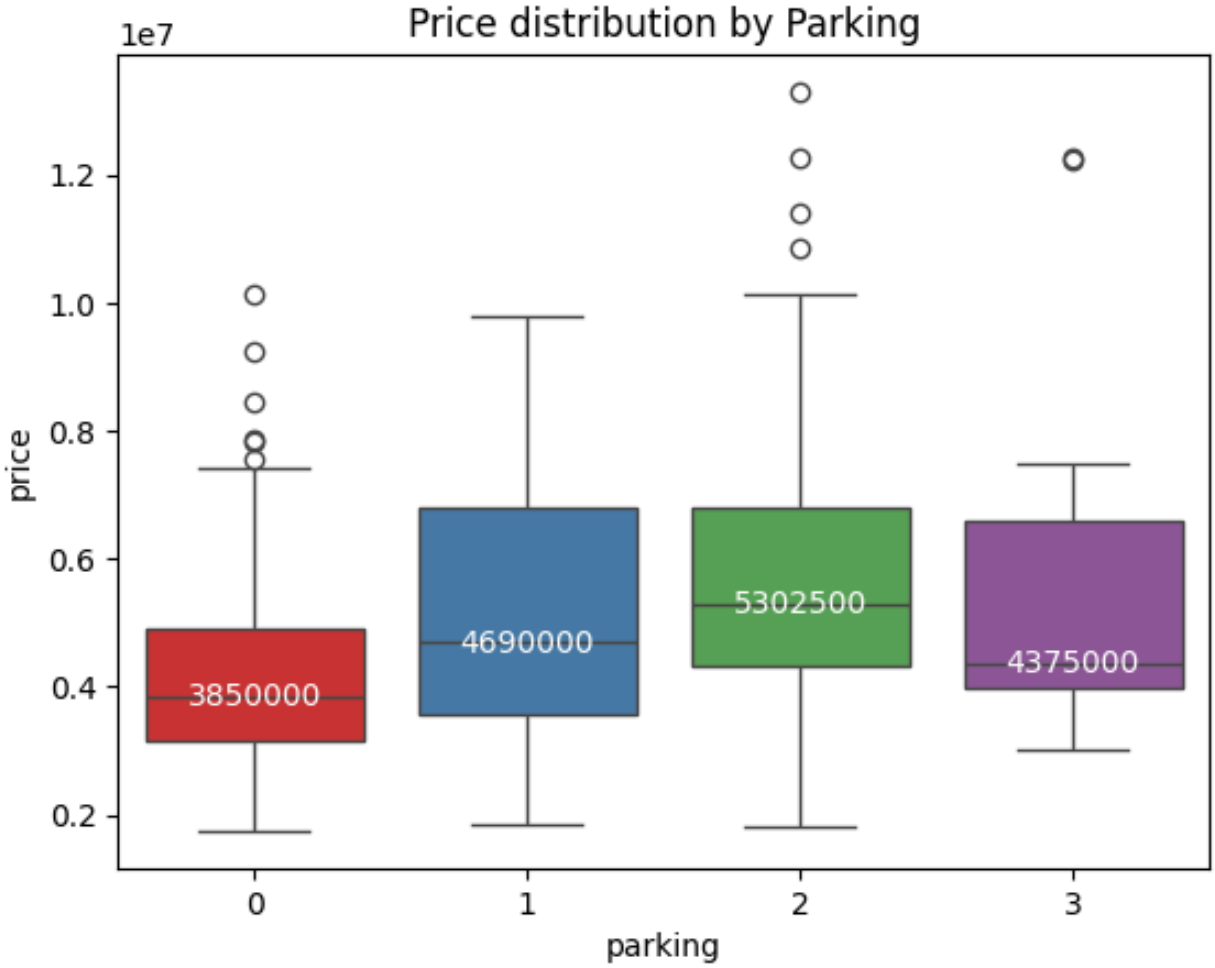




Is there a trend in house prices based on the number of parking spaces available?

In [67]:
```
median_prices = df.groupby("parking")["price"].median()
sns.boxplot(x="parking",y="price",data=df,palette="Set1")
for i,median in enumerate(median_prices):
    plt.text(i,median,f"{median:.0f}",ha="center",va="center",fontsize=10,color="white

plt.title("Price distribution by Parking")
plt.show()
```

```
C:\Users\dell\AppData\Local\Temp\ipykernel_11828\1557225532.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x="parking",y="price",data=df,palette="Set1")
```



Encoding Categorical Data Using Column Transformer

In [69]:
```
df.head(3)
```

Out[69]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwate
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	

In [70]:
```
# Nominal Data
# mainroad,guestroom,basement,airconditioning,hotwaterheating,prefarea

# Ordinal Data
# furnishingstatus
```

In [71]:
```
X_train,X_test,y_train,y_test = train_test_split(df.iloc[:,1:],df["price"],test_size=0
```

In [72]:
```
X_train.shape
```

Out[72]: (436, 12)

In [73]:
```
X_test.shape
```

Out[73]: (109, 12)

In [75]:
```
X_train.head()
```

Out[75]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating
412	2610	3	1	2	yes	no	yes	no
284	7770	2	1	1	yes	no	no	no
504	3185	2	1	1	yes	no	no	no
209	6720	3	1	1	yes	no	no	no
269	3900	3	1	2	yes	no	no	no

In [76]:
```
y_train.head()
```

Out[76]:
```
412    3430000
284    4270000
504    2653000
209    4900000
269    4375000
Name: price, dtype: int64
```

In [77]:
```
X_test.head()
```

Out[77]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating
333	3000	3	1	2	yes	no	no	no
84	3760	3	1	2	yes	no	no	yes
439	3930	2	1	1	no	no	no	no
396	3640	2	1	1	yes	no	no	no
161	6100	3	1	3	yes	yes	no	no

In [78]:
```
transformer = ColumnTransformer(transformers=[
    ("tnf1",OneHotEncoder(drop="first",sparse_output=False),["mainroad","guestroom","b
    ("tnf2",OrdinalEncoder(categories=[["unfurnished","semi-furnished","furnished"]]),
],remainder="passthrough")
```

In [79]:
```
X_train_transformed = transformer.fit_transform(X_train)
```

In [80]:
```
X_test_transformed = transformer.transform(X_test)
```

In [81]:
```
X_train_transformed.shape
```

Out[81]:
```
(436, 12)
```

In [82]:
```
X_test_transformed.shape
```

Out[82]:
```
(109, 12)
```

Model Training

In [83]:
```
lr = LinearRegression()
```

In [84]:
```
lr.fit(X_train_transformed,y_train)
```

Out[84]:
```
LinearRegression
LinearRegression()
```

In [85]:
```
# Coefficients
```

In [86]:
```
lr.coef_
```

Out[86]:
```
array([3.85208365e+05, 3.47165368e+05, 4.00250356e+05, 8.55146057e+05,
       8.19777718e+05, 7.03590958e+05, 2.74547937e+05, 2.47065632e+02,
       6.28684394e+04, 9.56921653e+05, 4.50768846e+05, 2.74140242e+05])
```

In [87]:
```
lr.intercept_
```

Out[87]:
```
np.float64(-211057.4858898064)
```

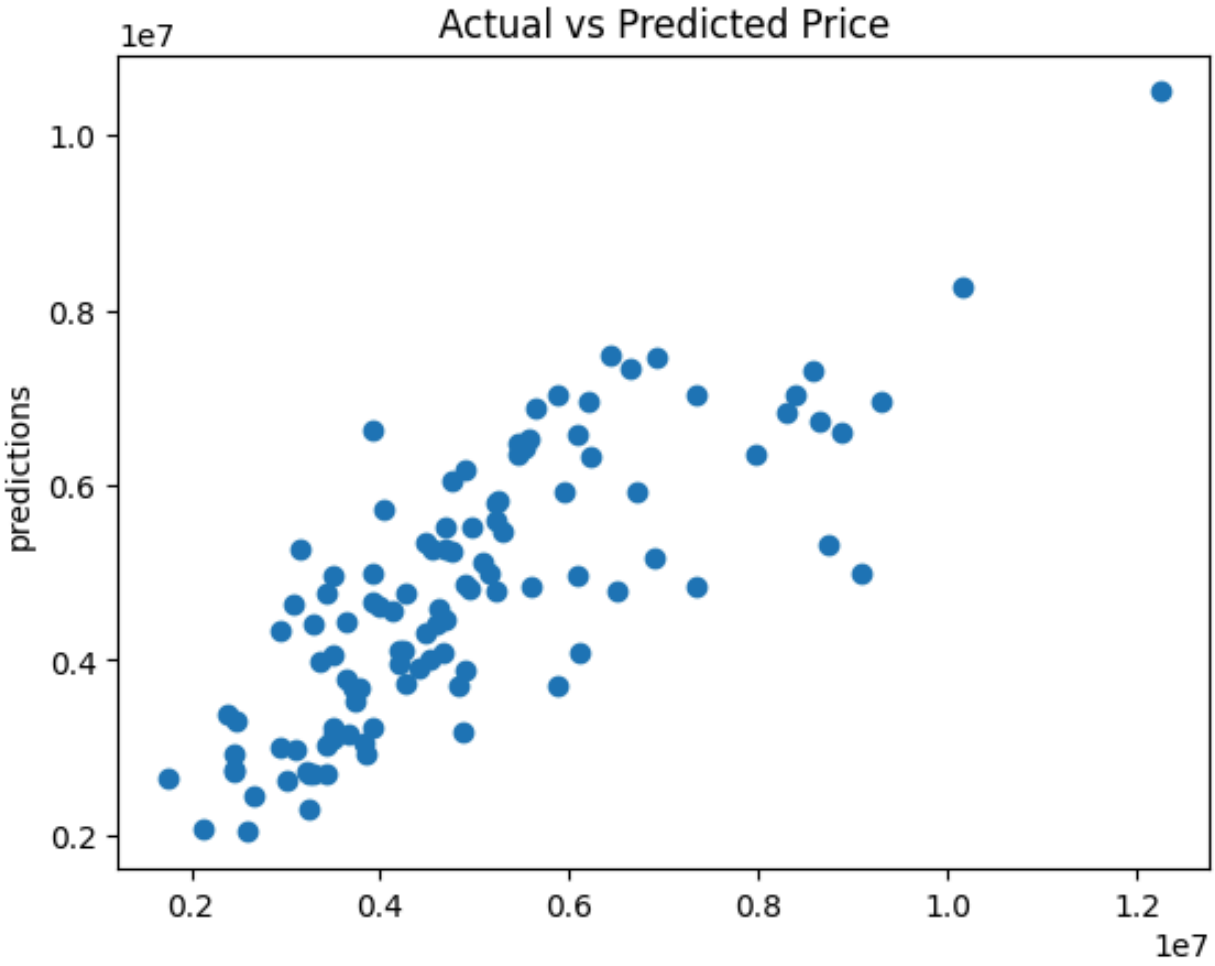
In [88]:
```
predict = lr.predict(X_test_transformed)
```

In [89]:
```
predict
```

Out[89]:
```
array([ 3236960.37582524,  4792788.45772144,  2293337.82629815,
        3155585.33700301,  6359535.06358422,  4072198.83242717,
        3703777.45517467,  3048885.40988129,  3992109.00900103,
        3219453.58460033,  3729684.52334155,  3100172.7877609 ,
        3696251.37583886,  5130299.5328895 ,  4832895.63325195,
        6886157.60569194,  7342665.53866931,  4638608.34456029,
        2919473.33801346,  4422226.70604752,  5478976.29524966,
        4763756.49160806,  4786634.32599728,  6435686.37717437,
        2041330.88146455,  2653295.19531634,  2705723.41111799,
        4306854.42832654,  4968464.23956699,  3381798.85613597,
        5003153.47942992,  4111955.80135169,  3910903.45759475,
        4850455.57382503,  4084033.65407561,  7450712.53753951,
        2462230.46663368,  5178054.20074157,  4994309.57777006,
        2639343.48021869,  7029775.91042757,  3308304.32368058,
        3545505.55193437,  4118607.73544385,  2999583.69420033,
        3778038.14205686,  3042762.77869684,  3182564.11298027,
        4867896.5491543 ,  4664908.61989216, 10496757.35240337,
        5794741.80267215,  2744168.14991571,  2705723.41111799,
        4340940.08778979,  6536355.44746401,  6614326.88571226,
        4470640.41128201,  2063566.78836163,  6639997.16271077,
        2981581.28430979,  3146110.40737524,  4558185.98350715,
        7030037.36901476,  7036354.40740857,  5268604.57858188,
        4022530.50774465,  4820455.91603789,  2726702.10383376,
        5717591.57881352,  6164194.92955443,  5595048.60149722,
        6590976.0570585 ,  5934638.97033909,  5331395.56763116,
        5267634.65206515,  6471306.54767744,  5357548.20617021,
        8263258.96893578,  4978220.6064522 ,  6949180.10448651,
        4095822.56490045,  3954313.03850126,  2722915.39519116,
        2708713.17385671,  4442993.63967217,  7304837.17263145,
        6355738.93416692,  3892786.33251179,  5511134.52521564,
        4771896.0159643 ,  5250795.6467527 ,  4592980.08600322,
        7477995.20917139,  5529776.54528068,  6730554.37475754,
        6966710.71850679,  3699777.41749825,  6050406.67443497,
        4404108.51102906,  6828659.0499602 ,  4609969.59477681,
        5921759.72087412,  3687387.13838065,  6316988.89226452,
        2933328.87821231,  5827968.29679833,  5261159.32429926,
        4989556.55081023])
```

In [90]:
```
plt.scatter(y_test,predict)
plt.ylabel("predictions")
plt.title("Actual vs Predicted Price")
```

Out[90]:
```
Text(0.5, 1.0, 'Actual vs Predicted Price')
```



Model Evaluation

In [121…
```
print("mean_absolute_error",mean_absolute_error(y_test,predict))
print("mean_squared_error",mean_squared_error(y_test,predict))
print("root_mean_squared_error",math.sqrt(mean_squared_error(y_test,predict)))
print("r2_score",r2_score(y_test,predict))
```

```
mean_absolute_error 865891.4864066666
mean_squared_error 1291157100569.138
root_mean_squared_error 1136290.9401069507
r2_score 0.6361990100766235
```

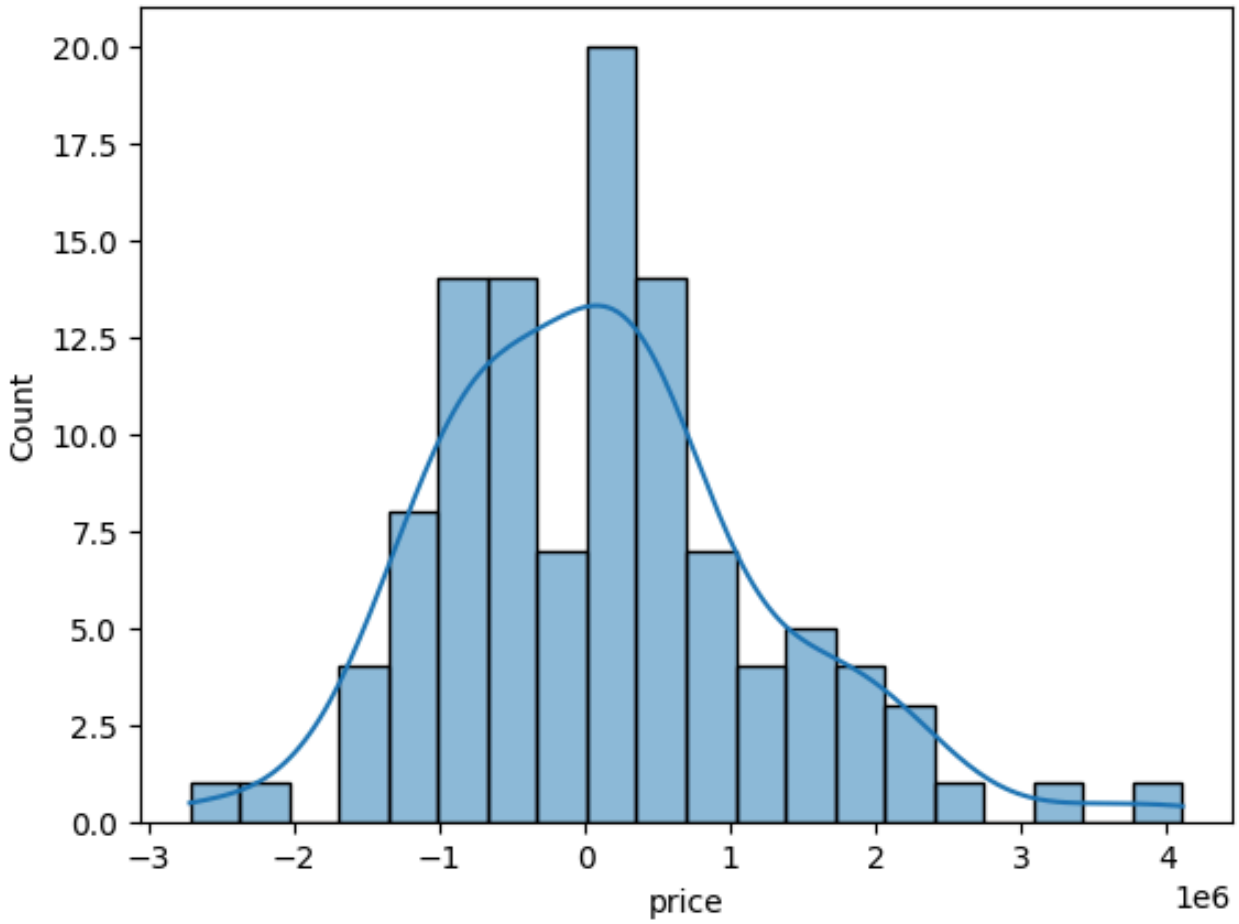
Checking For Normality

In [93]:
```
# Residuals
```

In [94]:
```
Residuals = y_test - predict
```

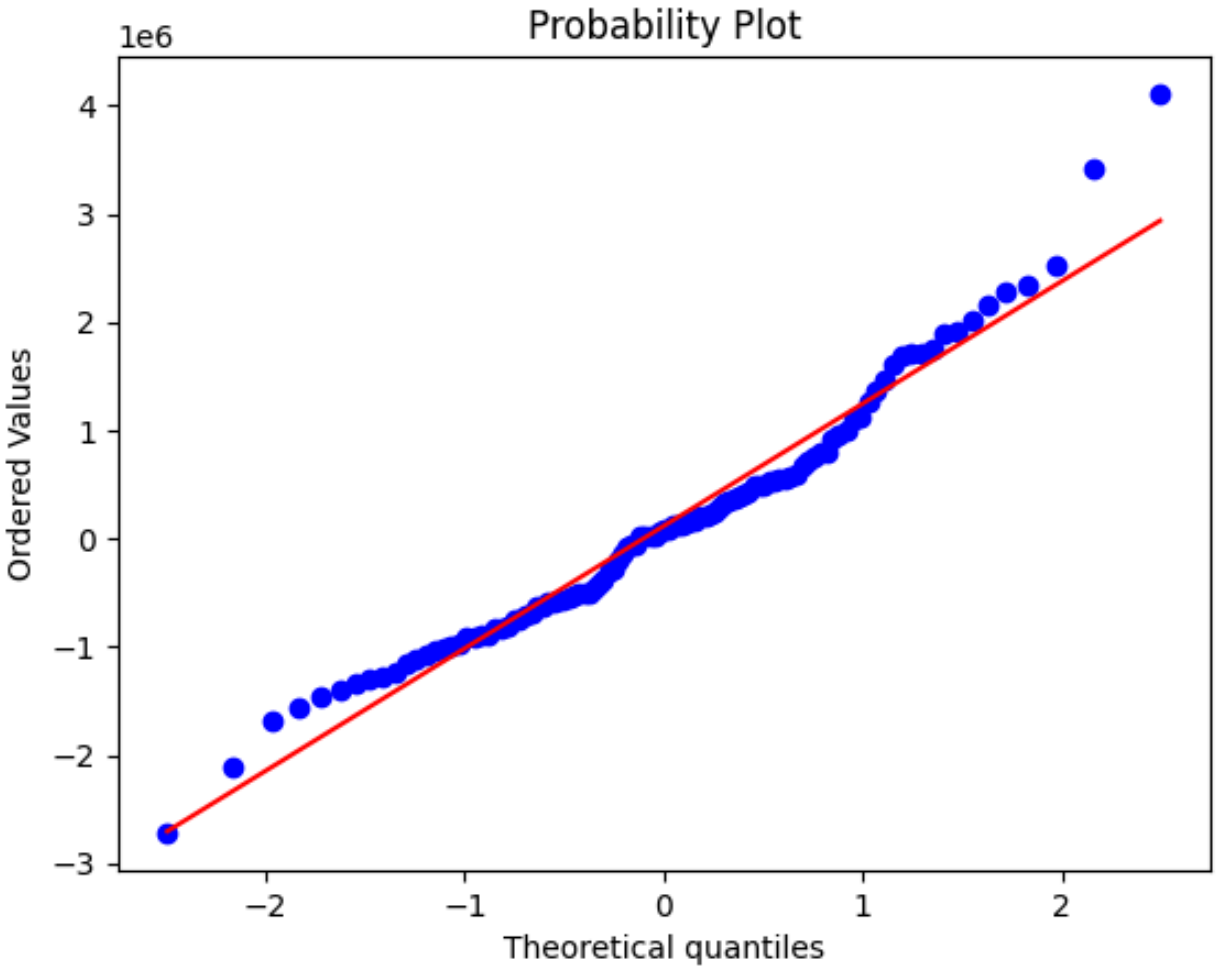
In [95]:
```
sns.histplot(Residuals,kde=True,bins=20)
```

Out[95]:
```
<Axes: xlabel='price', ylabel='Count'>
```



In [96]:
```
# QQPlot
```

In [97]:
```
stats.probplot(Residuals,dist="norm",plot=pylab)
pylab.show()
```



Checking for Overfitting and Underfitting

In [99]:
```
# on test data
```

In [103…
```
lr.score(X_test_transformed,y_test)
```

Out[103…
```
0.6361990100766235
```

In [107…
```
# on training data
```

In [104…
```
lr.score(X_train_transformed,y_train)
```

Out[104…
```
0.6884454072877346
```

In [106…
```
 # 0.688 - 0.636 = 0.052 the differnce is not too big.
```

The model's performance shows a moderate fit with an R^2 score of 0.63 on the test data, indicating that it explains 63% of the variance in house prices. While the model captures some key trends, such as the influence of area and bathrooms on price, there is still room for improvement. The slight difference between training and test scores suggests potential overfitting, and further tuning, such as feature engineering or regularization, could enhance its predictive accuracy.

In [154…
```
rid = Ridge(alpha=0.1)
```

In [155…
```
rid.fit(X_train_transformed,y_train)
```

Out[155…
```
Ridge(alpha=0.1)
```

In [156…
```
rid.score(X_train_transformed,y_train)
```

Out[156… 0.6884449490027233

In [157…
```
rid.score(X_test_transformed,y_test)
```

Out[157… 0.6361885859866439

In [187…
```
ls = Lasso(alpha=1600)
```

In [188…
```
ls.fit(X_train_transformed,y_train)
```

Out[188…
```
Lasso(alpha=1600)
```

In [189…
```
ls.score(X_train_transformed,y_train)
```

Out[189… 0.6884044955024525

In [190…
```
ls.score(X_test_transformed,y_test)
```

Out[190… 0.6362012724110142

In [ ]: