House Price Prediction

Case Study: House Price Prediction

Background

Lagos is one of the fastest-growing cities in Africa, with a rapidly expanding population and a booming real estate market. The city's housing market is highly competitive, with a wide range of properties available at varying prices. However, with the high demand for housing, it can be challenging for buyers and sellers to accurately determine the fair market value of a property.

Objective

The objective of this case study is to help Cressida Homes, a new entrant into the Real Estate Market, develop a machine learning model that can accurately predict the price of a house in Lagos based on its features, such as size, number of bedrooms, and amenities.

Data

The data used in this case study is a publicly available dataset from Kaggle, which contains information on various properties in Lagos, including their location, size, number of bedrooms, and price. The dataset contains over 545 records and 13 variables.

Methodology

The methodology used in this case study involves the following steps:

Data cleaning and preprocessing: The first step is to clean and preprocess the data, including handling missing values, removing outliers, and transforming variables as necessary.

Exploratory data analysis: Next, we will perform exploratory data analysis to gain insights into the data, such as identifying trends and patterns, and identifying correlations between variables.

Feature engineering: Based on the insights gained from the exploratory data analysis, we will perform feature engineering to select the most relevant features for predicting house prices and transform them as necessary.

Model selection and training: We will then select a suitable machine learning algorithm for predicting house prices, such as linear regression or a decision tree, and train the model on the preprocessed data.

Model evaluation and fine-tuning: We will evaluate the performance of the model using various metrics, such as mean absolute error and mean squared error, and fine-tune the model as necessary to improve its accuracy.

In [13]:

```python
# Import necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings("ignore")
from scipy.stats import skew, kurtosis
```

Load the Data

In [2]:

```python
# Load dataset

df = pd.read_csv(r"C:\Users\HENRY OKEOMA\Downloads\Cressida_Housing_Data (1).csv")
df
```

Out[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	pre
0	2835000	4350	3	1	2	no	no	no	yes	no	1	
1	5250000	9800	4	2	2	yes	yes	no	no	no	2	
2	4543000	4100	2	2	1	yes	yes	yes	no	no	0	
3	4200000	4600	3	2	2	yes	no	no	no	yes	1	
4	2975000	4352	4	1	2	no	no	no	no	no	1	
...	...	...	...	...	...	...	...	...	...	...	...	
540	7420000	6325	3	1	4	yes	no	no	no	yes	1	
541	4480000	4510	4	1	2	yes	no	no	no	yes	2	
542	3570000	4500	4	2	2	yes	no	yes	no	no	2	
543	3850000	5300	5	2	2	yes	no	no	no	no	0	
544	7455000	4300	3	2	2	yes	no	yes	no	no	1	

545 rows × 13 columns

Data Inspection and Cleaning

In [4]:

```python
# Descriptive statistics

df.describe(include='all').T
```

Out[4]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	
price	545.0	NaN	NaN	NaN	4766729.247706	1870439.615657	1750000.0	3430000.0	4340000.0	5740000.0	13300
area	545.0	NaN	NaN	NaN	5150.541284	2170.141023	1650.0	3600.0	4600.0	6360.0	16
bedrooms	545.0	NaN	NaN	NaN	2.965138	0.738064	1.0	2.0	3.0	3.0	
bathrooms	545.0	NaN	NaN	NaN	1.286239	0.50247	1.0	1.0	1.0	2.0	
stories	545.0	NaN	NaN	NaN	1.805505	0.867492	1.0	1.0	2.0	2.0	
mainroad	545	2	yes	468	NaN	NaN	NaN	NaN	NaN	NaN	
guestroom	545	2	no	448	NaN	NaN	NaN	NaN	NaN	NaN	
basement	545	2	no	354	NaN	NaN	NaN	NaN	NaN	NaN	
hotwaterheating	545	2	no	520	NaN	NaN	NaN	NaN	NaN	NaN	
airconditioning	545	2	no	373	NaN	NaN	NaN	NaN	NaN	NaN	
parking	545.0	NaN	NaN	NaN	0.693578	0.861586	0.0	0.0	0.0	1.0	
prefarea	545	2	no	417	NaN	NaN	NaN	NaN	NaN	NaN	
furnishingstatus	545	3	semi-furnished	227	NaN	NaN	NaN	NaN	NaN	NaN	

In [6]:

```
# Information about our columns

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 [7]:

```
# Check for missing values

df.isnull().sum()
```

Out[7]:

```
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 [8]:

```
# Check for duplicates


df.duplicated().sum()
```

Out[8]:

```
0
```

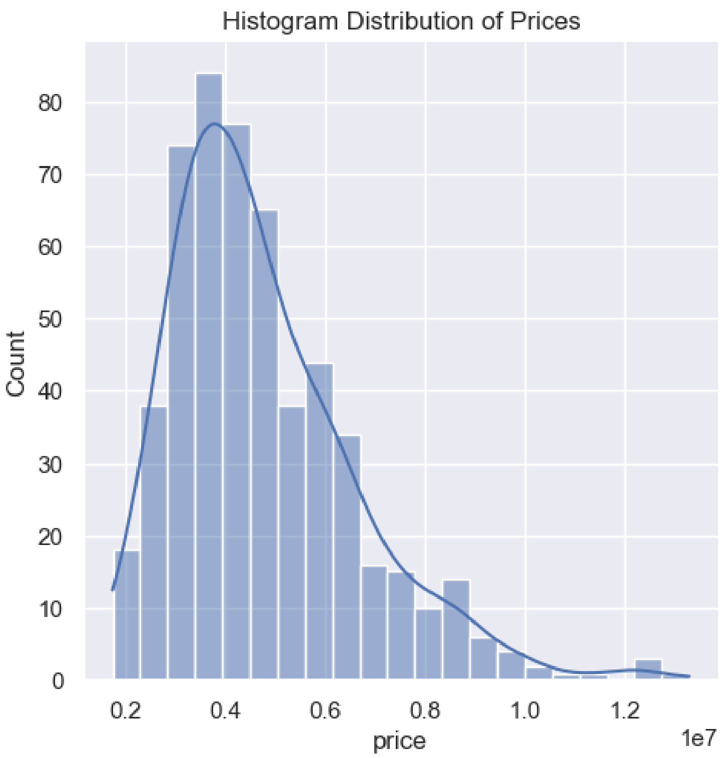
Dataset is clean for Exploratory Data Analysis

EDA

In [55]:

```
# Check Price distribution

plt.figure(figsize=(40,20))
this_plot = sns.displot(df['price'], kde=True)
plt.title('Histogram Distribution of Prices');
```

```
<Figure size 4000x2000 with 0 Axes>
```



In [14]:

```
print(df['price'].skew())
print(df['price'].kurtosis())
```

```
1.2122388370279804
1.9601302314152003
```

```
Insight: We have a positively skewed graph with outlsiers present on the right. Further we hava a platykurtic distribution with few tailing on the right.
```

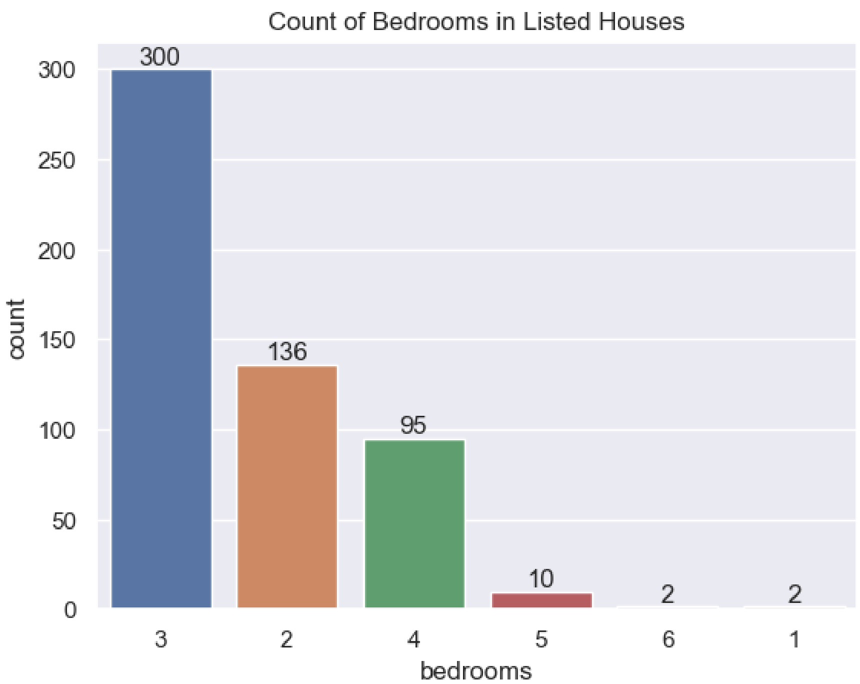
In [54]:

```
# Check for relationship between area and price

sns.scatterplot(x='area',y='price', data=df)
plt.ticklabel_format(style='plain')
plt.title('Relationship between Price and Area of the House');
```


```
Insight: We have a positive correlation between area of the house and the price. Although the correlation is weak.
```

In [52]:

```
# Check for spread of rooms

ax = sns.countplot(x='bedrooms', data=df, order=df['bedrooms'].value_counts().index)
values = df['bedrooms'].value_counts().values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of Bedrooms in Listed Houses');
```



Insights: Three bedrooms are predominant in our dataset followed by 2 beds

In [17]:

```
df.head(3)
```

Out[17]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefar
0	2835000	4350	3	1	2	no	no	no	yes	no	1	
1	5250000	9800	4	2	2	yes	yes	no	no	no	2	
2	4543000	4100	2	2	1	yes	yes	yes	no	no	0	

In [21]:

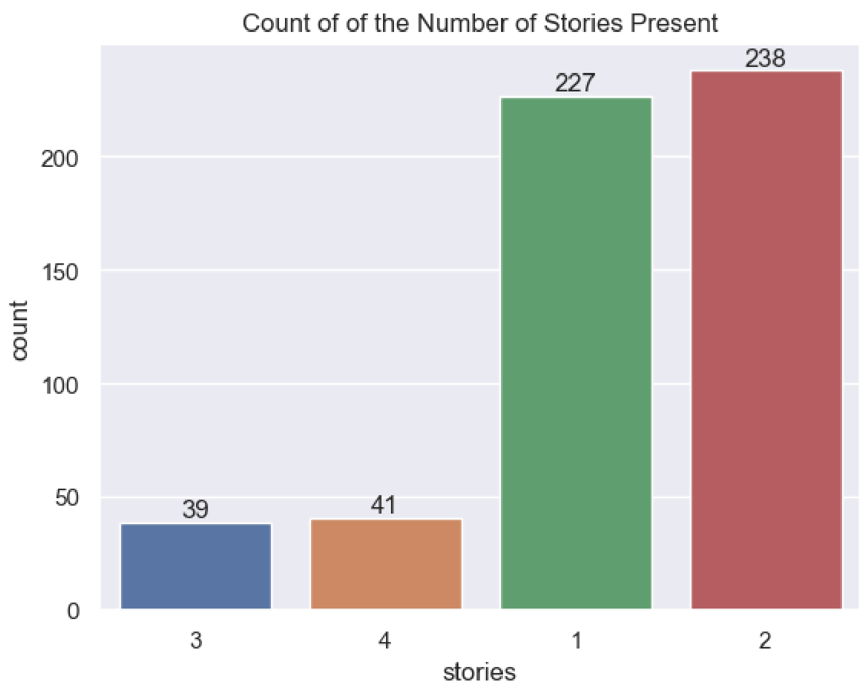
```
df['bedrooms'].value_counts(ascending=False)
```

Out[21]:

```
3    300
2    136
4     95
5     10
6      2
1      2
Name: bedrooms, dtype: int64
```

In [56]:

```
# Check for spread of floors/stories

ax = sns.countplot(x='stories', data=df, order=df['stories'].value_counts(ascending=True).index)
values = df['stories'].value_counts(ascending=True).values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of of the Number of Stories Present');
```



Insight: We have more 2 and 1 stories buildings than the others.

In [27]:

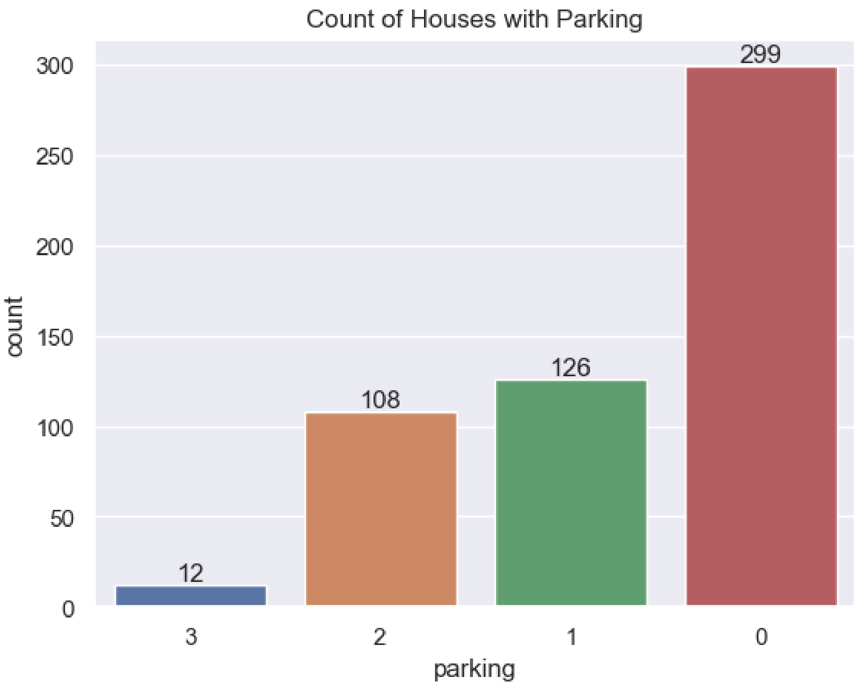
```
df['stories'].value_counts()
```

Out[27]:

```
2    238
1    227
4     41
3     39
Name: stories, dtype: int64
```

In [51]:

```
# Check for spread of parking spaces

ax = sns.countplot(x='parking', data=df, order=df['parking'].value_counts(ascending=True).index)
values = df['parking'].value_counts(ascending=True).values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of Houses with Parking');
```




```
Insight: Most of the houses dont have parking spaces.
```

In [30]:

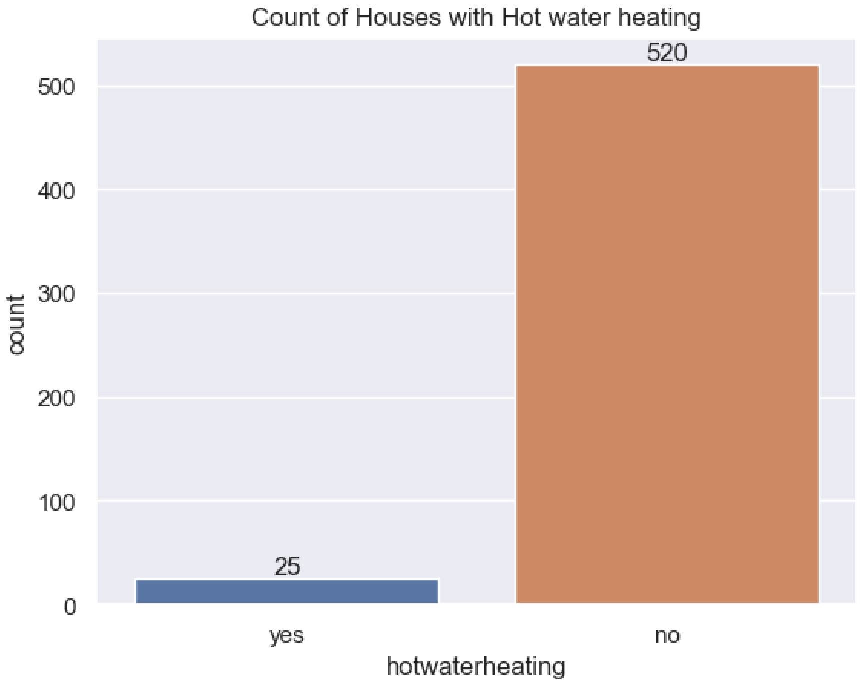
```
df['parking'].value_counts()
```

Out[30]:

```
0    299
1    126
2    108
3     12
Name: parking, dtype: int64
```

In [50]:

```
# How many houses have hot water heating?

ax = sns.countplot(x='hotwaterheating', data=df, order=df['hotwaterheating'].value_counts(ascending=True).index)
values = df['hotwaterheating'].value_counts(ascending=True).values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of Houses with Hot water heating');
```



In [33]:

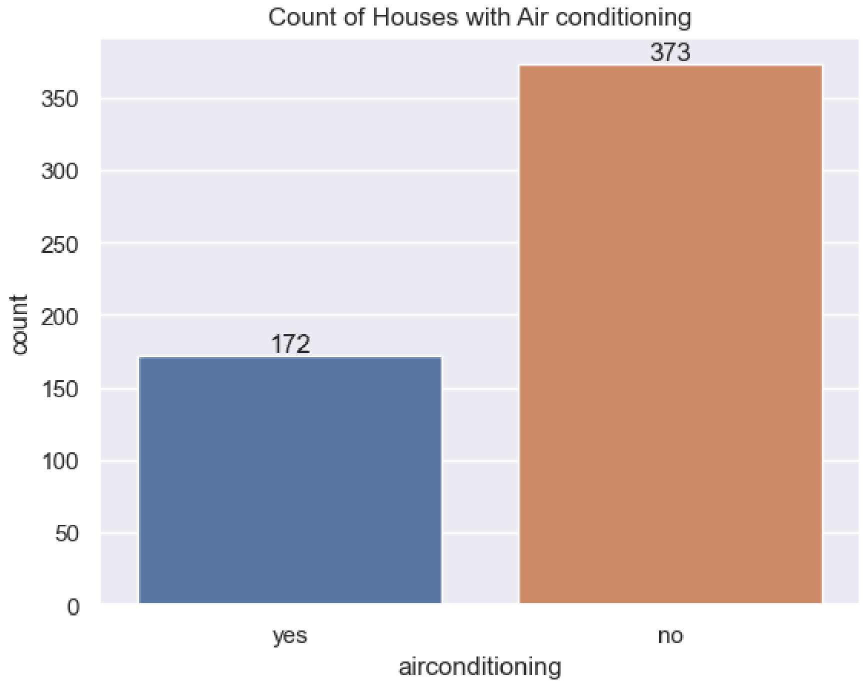
```
df['hotwaterheating'].value_counts()
```

Out[33]:

```
no     520
yes     25
Name: hotwaterheating, dtype: int64
```

In [49]:

```
# How many houses have airconditioning?

ax = sns.countplot(x='airconditioning', data=df, order=df['airconditioning'].value_counts(ascending=True).index)
values = df['airconditioning'].value_counts(ascending=True).values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of Houses with Air conditioning');
```



In [36]:

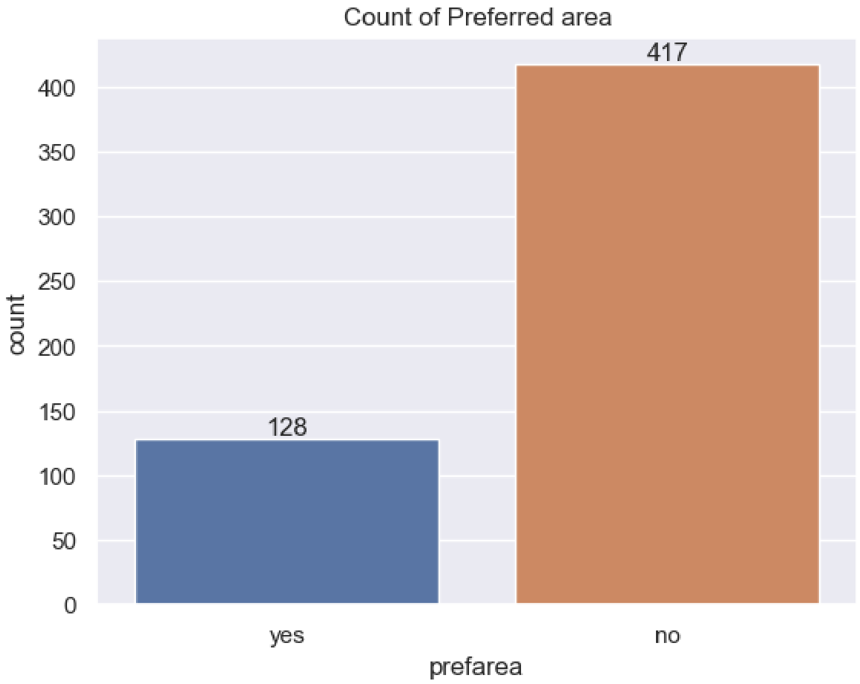
```
df['airconditioning'].value_counts()
```

Out[36]:

```
no     373
yes    172
Name: airconditioning, dtype: int64
```

In [48]:

```
# How many houses are in preferred areas?

ax = sns.countplot(x='prefarea', data=df, order=df['prefarea'].value_counts(ascending=True).index)
values = df['prefarea'].value_counts(ascending=True).values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of Preferred area');
```




```
Insight: Few Houses are in the preferred area
```

In [39]:

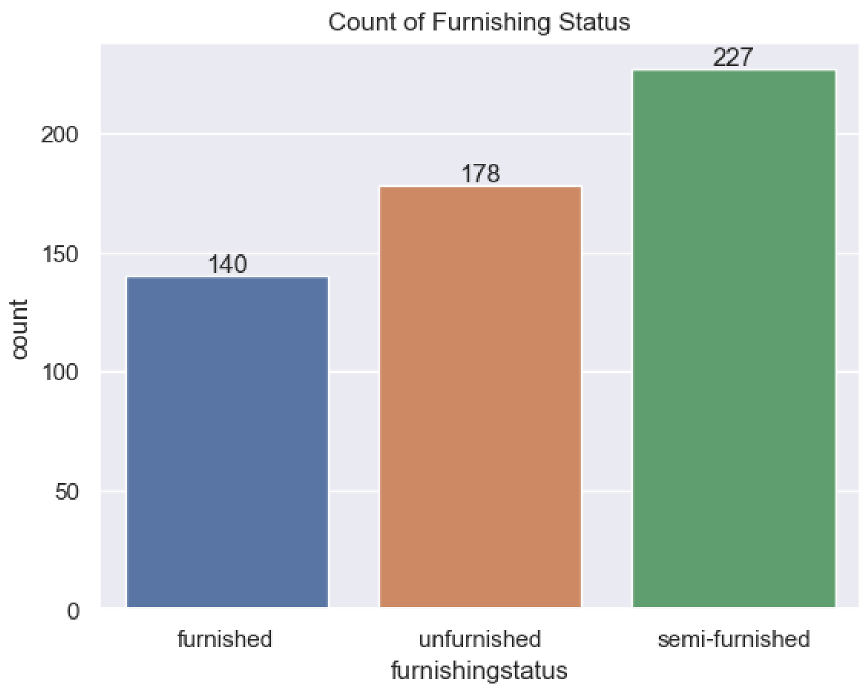
```
df['prefarea'].value_counts()
```

Out[39]:

```
no     417
yes    128
Name: prefarea, dtype: int64
```

In [47]:

```python
# What is the furnishing status?

ax = sns.countplot(x='furnishingstatus', data=df, order=df['furnishingstatus'].value_counts(ascending=True).index)
values = df['furnishingstatus'].value_counts(ascending=True).values
ax.bar_label(container=ax.containers[0], labels=values)
plt.title('Count of Furnishing Status');
```



Insight: Semi-Furnished houses are greater in number than both furnished. A good number of the houses are also furnished

In [44]:

```python
df['furnishingstatus'].value_counts()
```

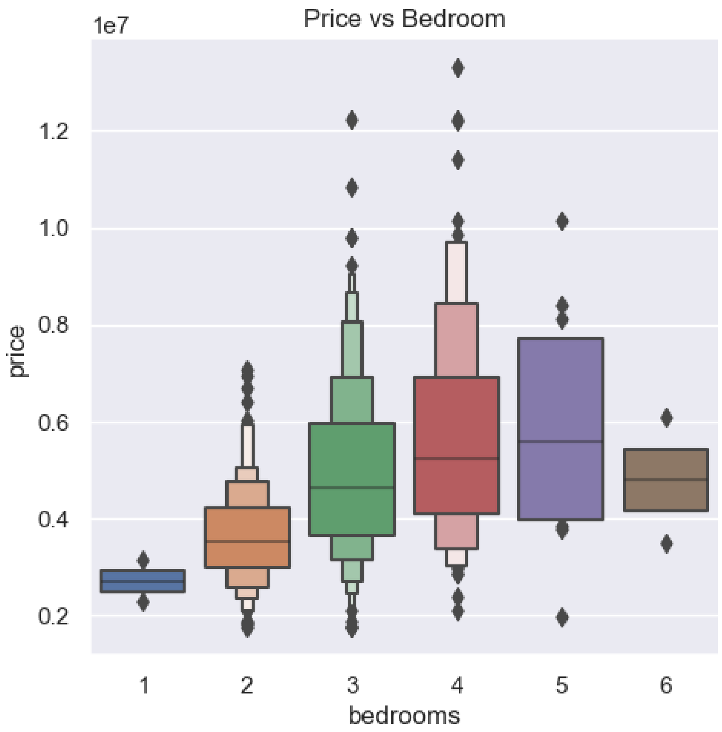
Out[44]:

```
semi-furnished    227
unfurnished       178
furnished         140
Name: furnishingstatus, dtype: int64
```

BIVARIATE ANALYSIS

In [60]:

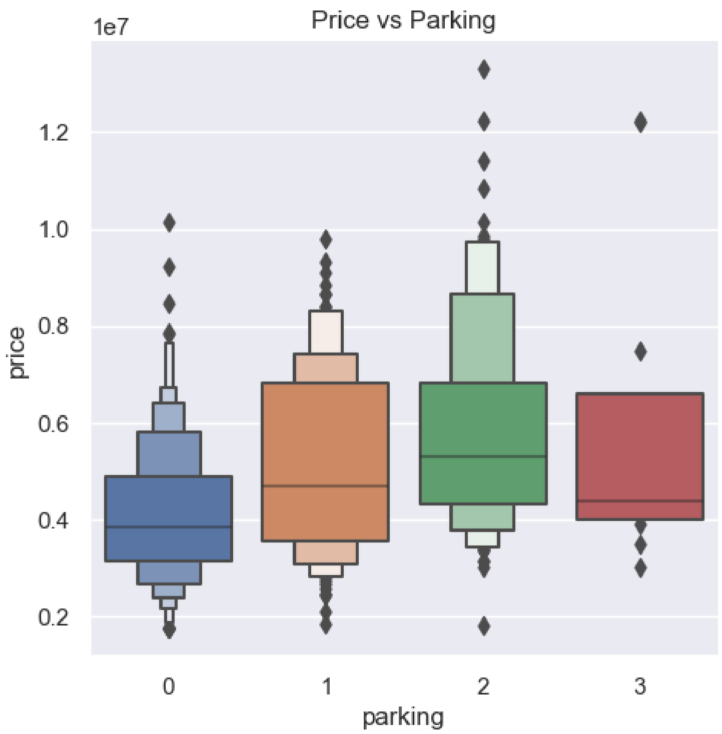
```
# Distribution of bedrooms and price

sns.catplot(data=df, x='bedrooms', y='price', kind="boxen")
plt.title('Price vs Bedroom');
```




```
Insights: 6 bed houses are lower in prices than 5, 4 and 3 bedroom. this could be because of less demand.
```

In [59]:

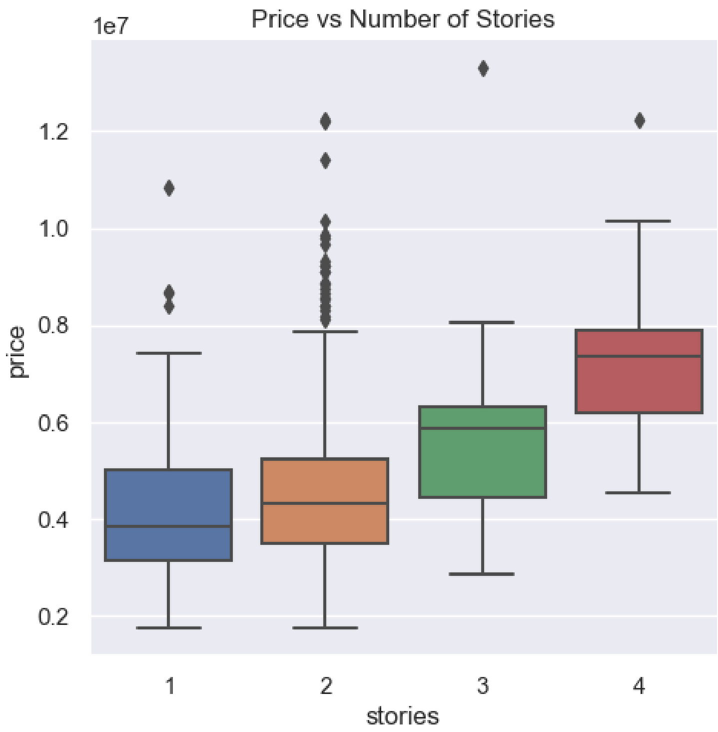
```
# Distribution of stories and price

sns.catplot(data=df, x='parking', y='price', kind="boxen")
plt.title('Price vs Parking');
```



Insight: Hhouses with 2 or 1 parking are more values than house with no parking

In [62]:

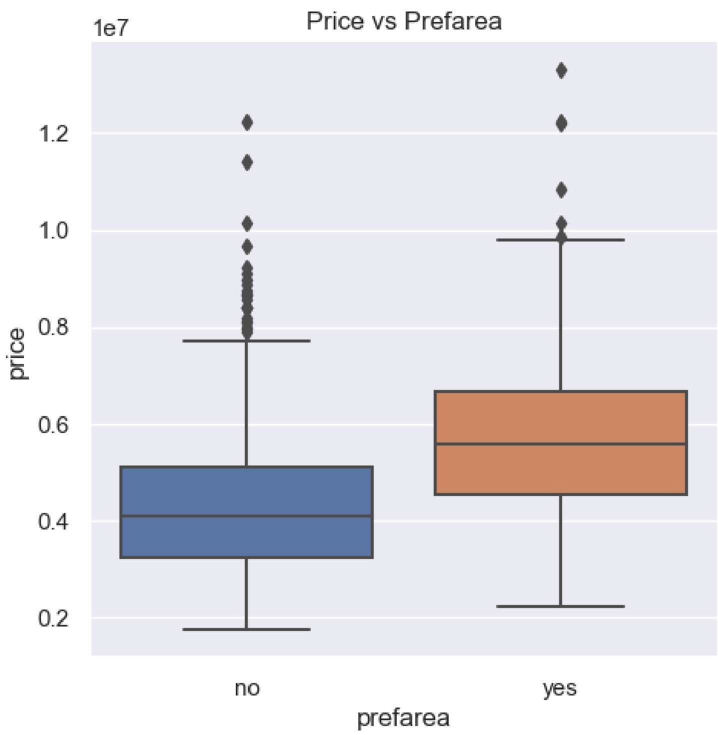
```
# Distribution of parking and price

sns.catplot(data=df, x='stories', y='price', kind="box")
plt.title('Price vs Number of Stories');
```



Insights:4 and 3 stories are most valued in price. No much different between 1 and 2 stories houses. Also 2 stories houses have lots price outliers in the data set

In [64]:

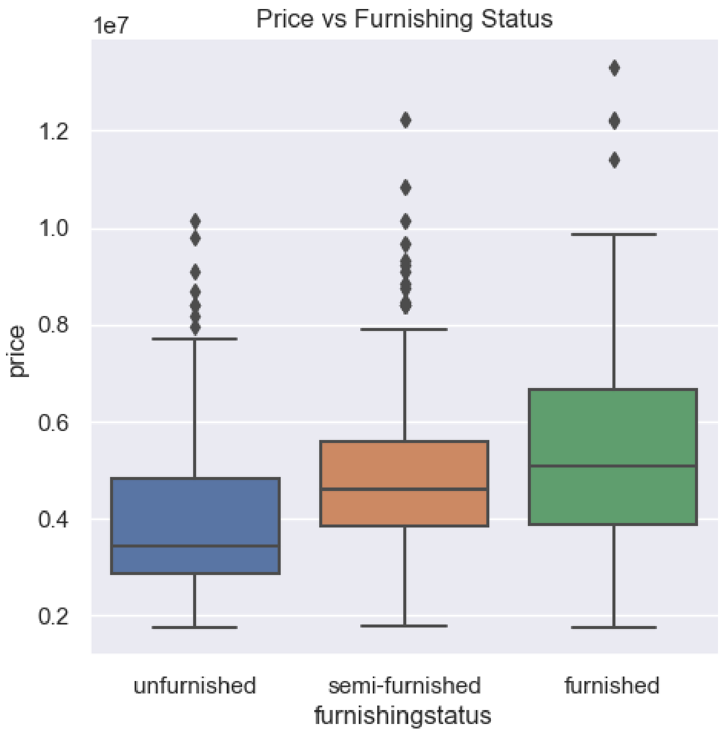
```
# Distribution of prefarea and price

sns.catplot(data=df, x='prefarea', y='price', kind="box")
plt.title('Price vs Prefarea');
```



```
Insight: Price are higher in the preferred area of the city
```

In [65]:

```
# Distribution of furnishing status and price

sns.catplot(data=df, x='furnishingstatus', y='price', kind="box")
plt.title('Price vs Furnishing Status');
```




```
Insight: Prices of furnished houses are highest then semi-furnished andunfurnished is the least
```

In [67]:

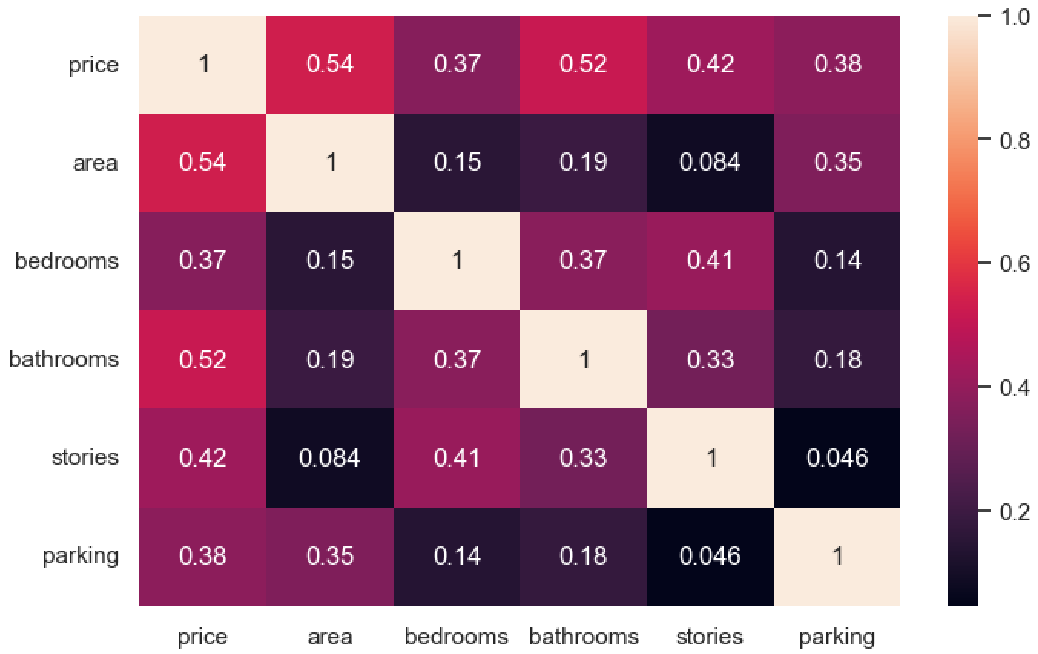
```
sns.pairplot(df);
```

In [71]:

```
# Check for correlation in our features


plt.figure(figsize=(8,5))
sns.heatmap(df.corr(),annot=True)
```

Out[71]:

```
<AxesSubplot:>
```




```
Insight: Number of bathroom and area of the property have a correlation with the price. Further correlation exist
between stories, bedrooms and parking.
```

Splitting the data for Machine Learnig

In [91]:

```
# We have to drop the price column, which the variable which we shall try to predict
X = df.drop('price', axis=1)
X
```

Out[91]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	fur
0	4350	3	1	2	no	no	no	yes	no	1	no	
1	9800	4	2	2	yes	yes	no	no	no	2	no	s
2	4100	2	2	1	yes	yes	yes	no	no	0	no	s
3	4600	3	2	2	yes	no	no	no	yes	1	no	s
4	4352	4	1	2	no	no	no	no	no	1	no	
...	...	...	...	...	...	...	...	...	...	...	...	
540	6325	3	1	4	yes	no	no	no	yes	1	no	
541	4510	4	1	2	yes	no	no	no	yes	2	no	s
542	4500	4	2	2	yes	no	yes	no	no	2	no	
543	5300	5	2	2	yes	no	no	no	no	0	no	s
544	4300	3	2	2	yes	no	yes	no	no	1	no	

545 rows × 12 columns

We have dropped the price column from our data set

In [103]:

```
# We need to change our categorical columns to dummies
# (a variable that takes values of 0 and 1, where the values indicate the absence or presence of something
# in this case 0 = No and 1 = Yes)

X = pd.get_dummies(X)
X
```

Out[103]:

	area	bedrooms	bathrooms	stories	parking	mainroad_no	mainroad_yes	guestroom_no	guestroom_yes	basement_no	base
0	4350	3	1	2	1	1	0	1	0	1	
1	9800	4	2	2	2	0	1	0	1	1	
2	4100	2	2	1	0	0	1	0	1	0	
3	4600	3	2	2	1	0	1	1	0	1	
4	4352	4	1	2	1	1	0	1	0	1	
...	...	...	...	...	...	...	...	...	...	...	
540	6325	3	1	4	1	0	1	1	0	1	
541	4510	4	1	2	2	0	1	1	0	1	
542	4500	4	2	2	2	0	1	1	0	0	
543	5300	5	2	2	0	0	1	1	0	1	
544	4300	3	2	2	1	0	1	1	0	0	

545 rows × 20 columns

All categorical variables have taken the form of 0 & 1s for easier predictions

In [104]:

```
y = df['price']
y
```

Out[104]:

```
0      2835000
1      5250000
2      4543000
3      4200000
4      2975000
        ...
540    7420000
541    4480000
542    3570000
543    3850000
544    7455000
Name: price, Length: 545, dtype: int64
```

Train-Test Split

In [105]:

```
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)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(436, 20)
(109, 20)
(436,)
(109,)
```

The data has now been splitted into train and test data for both the X and the y.

In [106]:

```
# Import algorithms


from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

Linear Regression Model

In [117]:

```
# Initialise/Create and instance of Linear Regression model

lr = LinearRegression()

# Fit your model
lr.fit(X_train, y_train)


# Make predictions
lr_pred = lr.predict(X_test)
```

In [109]:

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [120]:

```
mae = mean_absolute_error(y_test, lr_pred)
mse = mean_squared_error(y_test, lr_pred)
r_square = r2_score(y_test, lr_pred)

print(mae)
print(mse)
print(r_square)
```

```
733894.4525559898
1121668389130.9573
0.6334772718187527
```

Using other regression models

In [121]:

```
dr = DecisionTreeRegressor()
rr = RandomForestRegressor()


#create list of your model names
models = [lr, dr, rr]
```

In [122]:

```
#create function to train a model and evaluate metrics
def trainer(model,X_train,y_train,X_test,y_test):
    #fit your model
    model.fit(X_train,y_train)
    #predict on the fitted model
    prediction = model.predict(X_test)
    #print evaluation metric
    print('\nFor {}, Mean Absolute Error is {} \n'.format(model.__class__.__name__,mean_absolute_error(prediction,y_tes
    print('\nFor {}, Mean Squared Error is {} \n'.format(model.__class__.__name__,mean_squared_error(prediction,y_test)
    print('\nFor {}, R_Square is {} \n'.format(model.__class__.__name__,r2_score(prediction,y_test)))
    print('----------------------------------------------------------------------------------------------------')
    #print(classification_report(prediction,y_valid)) #use this later
```

In [134]:

```
#loop through each model, training in the process
for model in models:
    trainer(model,X_train,y_train,X_test,y_test)
```

```
For LinearRegression, Mean Absolute Error is 733894.4525559898


For LinearRegression, Mean Squared Error is 1121668389130.9573


For LinearRegression, R_Square is 0.40119374523585194

----------------------------------------------------------------------------------------------------

For DecisionTreeRegressor, Mean Absolute Error is 1058512.385321101


For DecisionTreeRegressor, Mean Squared Error is 2185063249747.7065


For DecisionTreeRegressor, R_Square is 0.21660741174236287

----------------------------------------------------------------------------------------------------

For RandomForestRegressor, Mean Absolute Error is 744220.3058103976


For RandomForestRegressor, Mean Squared Error is 1135598438382.0518


For RandomForestRegressor, R_Square is 0.4001944348111123

----------------------------------------------------------------------------------------------------
```

In [135]:

```
# Linear Regression has the best metrics from the lot
```

Cross Validation

In [136]:

```
from sklearn.model_selection import cross_val_score
```

In [139]:

```
models = [lr, dr, rr]




#create function to train a model and evaluate r2
def trainer_with_cv(model,X_train,y_train,X_test,y_test):
    scores = cross_val_score(model, X_train, y_train, scoring='r2', cv=5)
    #print evaluation metric
    print('\nFor {}, Cross-Validation Scores are  {} \n'.format(model.__class__.__name__,scores))
    print('----------------------------------------------------------------------------------------------------')
```

In [140]:

```
for model in models:
    trainer_with_cv(model,X_train,y_train,X_test,y_test)
```

```

For LinearRegression, Cross-Validation Scores are  [0.60655378 0.59677981 0.64522106 0.72765523 0.7371974
1]

----------------------------------------------------------------------------------------------------

For DecisionTreeRegressor, Cross-Validation Scores are  [0.43789247 0.2870949  0.20380123 0.46466467 0.29
961546]

----------------------------------------------------------------------------------------------------

For RandomForestRegressor, Cross-Validation Scores are  [0.61760379 0.60774089 0.50779901 0.72127124 0.59
115608]

----------------------------------------------------------------------------------------------------
```

Using K-Fold Cross Validation

In [141]:

```
from sklearn.model_selection import KFold
from numpy import mean
from numpy import std


# Perform a 10-Fold split and evaluate mean cross evaluation score
folds = KFold(n_splits=10, random_state=1, shuffle=True)

def trainer_with_kfold_cv(model,X_train,y_train,X_test,y_test):
    '''Cross validation function. Expects a model,'''
    # evaluate model
    scores = cross_val_score(model, X_train, y_train, scoring='r2', cv=folds, n_jobs=-1)
    # cross validation scores
    print('\nFor {}, Cross-Validation Scores are  {} \n'.format(model.__class__.__name__,scores))
    # report performance
    print('R_Square: %.3f' % (mean(scores)))
```

In [142]:

```
for model in models:
    trainer_with_kfold_cv(model,X_train,y_train,X_test,y_test)
```

```
For LinearRegression, Cross-Validation Scores are  [0.74080717 0.51459248 0.77382824 0.57659263 0.6873129
6 0.68356068
 0.53122189 0.62940424 0.65690655 0.64251035]

R_Square: 0.644

For DecisionTreeRegressor, Cross-Validation Scores are  [-0.03863353 -0.29755248  0.62246922  0.19780549
0.16388794  0.18033796
  0.16659575 -0.01588439 -0.33135362  0.39426552]

R_Square: 0.104

For RandomForestRegressor, Cross-Validation Scores are  [0.74257597 0.31852473 0.65955285 0.58115094 0.57
830797 0.61426959
 0.49365676 0.4535574  0.53301079 0.62250516]

R_Square: 0.560
```