DCS 550 Data Mining (DSC550-T302 2227-1)

Bellevue University

Term Final Project

Author: Jake Meyer

Date: 8/13/2022

Project Introduction - Prediction of House Prices

This project focuses on the prediction of house prices utilizing several different types of regression analysis. Homeowners, realtors, and real-estate companies are often interested in the important features that impact the price of a home. A homeowner may want to understand how a particular home improvement will enhance the value of their home. Realtors and real-estate companies will want to ensure the listing value for a home is accurate prior to engaging with their clients. This project will involve the following tasks:

1. Overview the dataset.
2. Perform exploratory data analysis.
3. Prepare the data for the model.
4. Identify which features are most influential for house prices.
5. Randomly split the data into 80% training and 20% test data, target is house price.
6. Build a few different models to predict house prices based on the most influential features included in the dataset.
7. Describe which type of model is recommended for the house price predictions.

The Housing Prices Dataset utilized for this project can be found in the cell below. The shape of the dataset is initially 13 columns and 545 rows. The features included in this dataset are house price, area of home, number of bedrooms, number of bathrooms, number of stories, connections to a main road, guest room included, basement included, hot water heater included, air conditioning included, number of parking spots, preferred location of the house, and furnishing status. The intent for this project will be to predict the price of a home based on the features included within this dataset. Other factors (such as location, home age, lot size, market conditions, etc...) would be recommended to include in future analysis.

Link to Housing Prices Dataset from Kaggle

Download the Housing.csv dataset from: Housing Dataset.

Milestone 1: Data Selection and Exploratory Data Analysis (EDA)

The first milestone will involve Tasks 1 and 2 listed in the project introduction section. Based on the features included in the dataset, here are some preliminary question(s) and/or topics of interest:

- Understand the distribution of House Prices.
- Understand the count of values for each categorical feature.
- Do any of the categorical features appear to have a relationship with house pricing?
- Which features appear to be correlated with house prices?
- Are there any missing values or outliers within the dataset?

In [165…

```
'''
Import the necessary libraries.
'''
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy
from collections import OrderedDict
from scipy.stats import skew
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import VarianceThreshold
from sklearn.decomposition import PCA
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score, accuracy
%matplotlib inline
```

In [5]:

```
'''
Check the versions of the packages.
'''
print('numpy version:', np.__version__)
print('pandas version:', pd.__version__)
print('seaborn version:', sns.__version__)
print('matplotlib version:', mpl.__version__)
print('scipy:', scipy.__version__)
```

```
numpy version: 1.20.3
pandas version: 1.3.4
seaborn version: 0.11.2
matplotlib version: 3.4.3
scipy: 1.7.1
```

Task 1 - Overview the dataset.

In [6]:

```
'''
Import the housing data with read_csv() with the file Housing.csv.
```

```
Note: A copy of the CSV file was placed into the same directory as this notebook.
'''
df = pd.read_csv('Housing.csv')
```

In [7]:
```
'''
Show the data has been loaded successfully into the data frame
by printing the first 5 rows with head().
'''
df.head()
```

Out[7]:

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

In [8]:
```
'''
Understand the shape of the dataframe initially.
'''
print('There are {} rows and {} columns in this data frame.'.format(df.shape[0], df.sha
```

```
There are 545 rows and 13 columns in this data frame.
```

In [9]:
```
'''
Display the total size of this data frame initially.
'''
print('This data frame contains {} records.'.format(df.size))
```

```
This data frame contains 7085 records.
```

In [10]:
```
'''
Understand if there are any missing values in the data frame.
'''
df.isna().sum()
```

Out[10]:
```
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 [11]:
```
'''
Find the type of data within each column initially.
'''
df.dtypes
```

Out[11]:
```
price              int64
area               int64
bedrooms           int64
bathrooms          int64
stories            int64
mainroad          object
guestroom         object
basement          object
hotwaterheating   object
airconditioning   object
parking            int64
prefarea          object
furnishingstatus  object
dtype: object
```

In [12]:
```
'''
Understand the number of unique values in each column.
'''
df.nunique()
```

Out[12]:
```
price             219
area              284
bedrooms            6
bathrooms           4
stories             4
mainroad            2
guestroom           2
basement            2
hotwaterheating     2
airconditioning     2
parking             4
prefarea            2
furnishingstatus    3
dtype: int64
```

Observations - Overview of the Data

- The dataset initially contains 545 rows and 13 columns.
- Some of the columns will need to be converted over to dummy variables during the data preparation step.
- There is a mix of numerical and categorical data.
- The target of the model will be 'House Price'.
- There are no missing values upon initial review of the dataset.
- Number of unique values for each column is defined above and will be reviewed during the EDA section.

Task 2 - Perform Exploratory Data Analysis (EDA).

In [13]:
```
'''
Utilize the describe() function to get a preliminary understanding of the dataset.
```

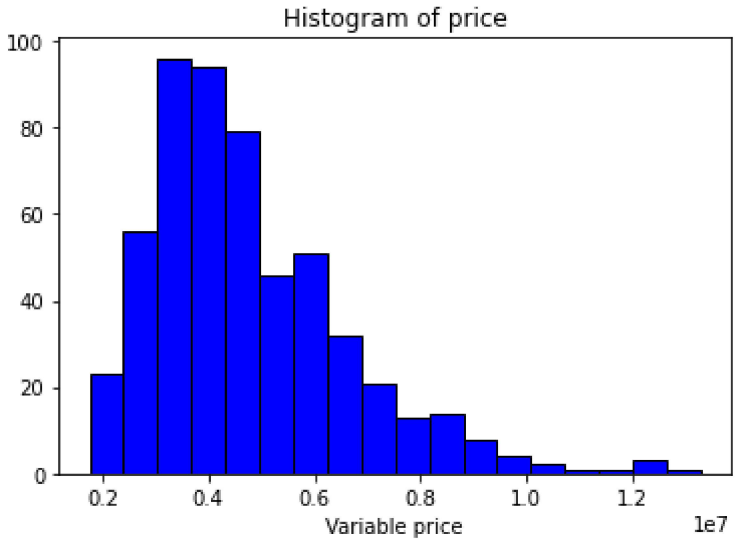
```
Note: This will only consider the numeric data based on the current data types.
'''
df.describe()
```

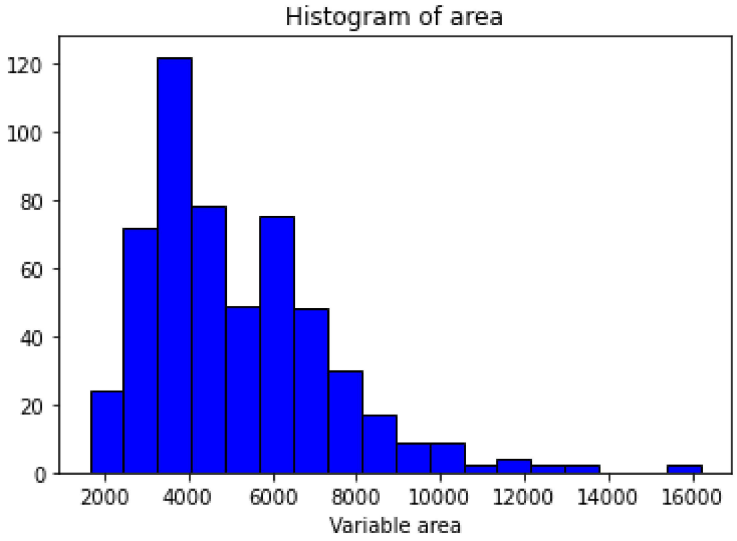
Out[13]:

	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

In [14]:

```
'''
This section of the code will utilize a for loop to plot the histograms for the
continuous numeric columns using the plt.hist() function.
'''
df_num = df[['price','area']]

# For loop generation of histogram plot across all numeric columns in Housing.csv.
for col in df_num.columns:
    # Iterate across columns in df_reduced to plot a histogram of each col.
    plt.hist(df_num[col], color = 'blue', edgecolor='k', bins = 18) # plt.hist() functi
    plt.title("Histogram of {}".format(col), fontsize=12) # Title added to each histogr
    plt.xlabel("Variable " +col, fontsize=10) # Xlabel added to each histogram plot.
    plt.xticks(fontsize=10) # Formatting of x tick marks for each histogram plot.
    plt.yticks(fontsize=10) # Formatting of y tick marks for each histogram plot
    plt.show() # plt.show() will provide the output histogram plots.
```





In [15]:
```
"""
Check the Fisher_Pearson correlation of skew for House Price with skew().
"""
print('Skewness for data:', skew(df['price']))
```

```
Skewness for data: 1.2088998457878217
```

In [16]:
```
"""
Check the Fisher_Pearson correlation of skew for House Area with skew().
"""
print('Skewness for data:', skew(df['area']))
```

```
Skewness for data: 1.3175492613408553
```

In [17]:
```
'''
Calculate the difference between the mean and median for House Price.
'''
mean_price = round(df['price'].mean(),0)
median_price = round(df['price'].median(),0)
difference = round((mean_price) - (median_price),0)
print('Mean House Price: ${}\n'
      'Median House Price: ${}\n'
      'Difference of Mean and Median House Price: ${}'.format(mean_price, median_price,
```

```
Mean House Price: $4766729.0
Median House Price: $4340000.0
Difference of Mean and Median House Price: $426729.0
```

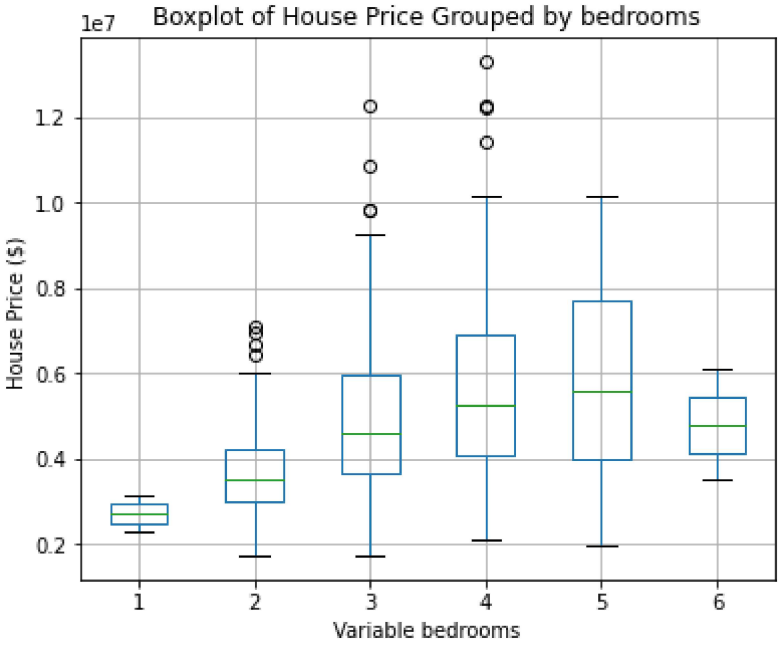
Observations - Histograms of House Price and Area

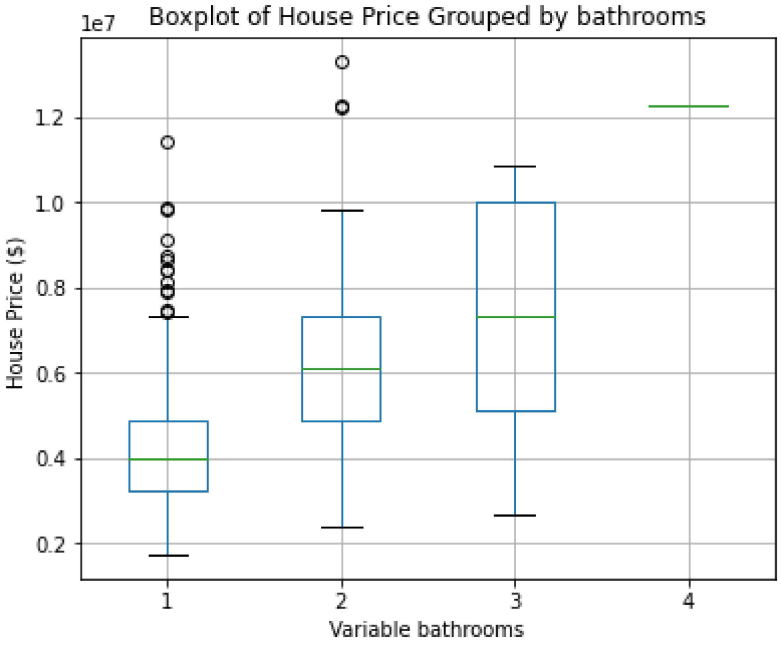
● The distribution for the target variable, House Price, appears to be slightly skewed (positive) with a Fisher_Pearson correlation of 1.21.

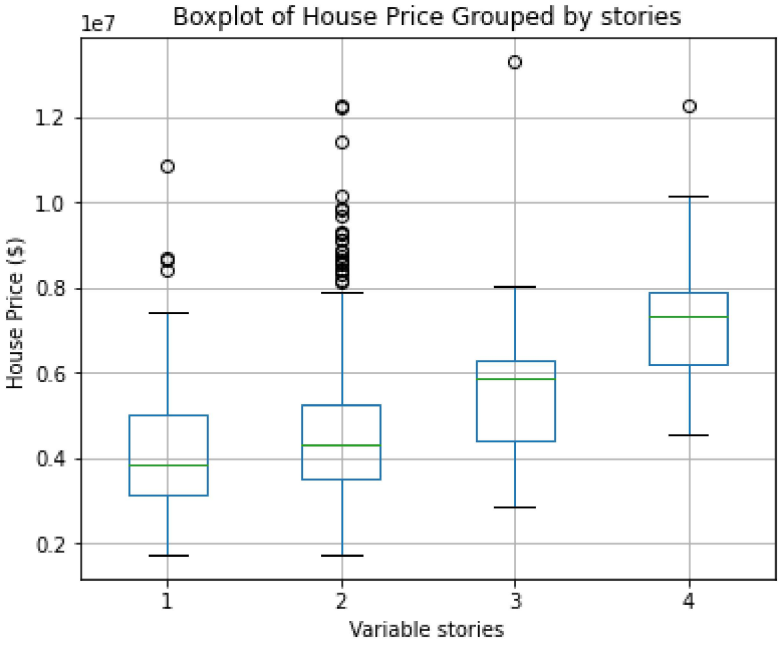
● The median House Price is lower than the mean by $426729.
 ● The distribution of Home Area is also slightly positvely skewed with a Fisher_Pearson correlation of 1.32.

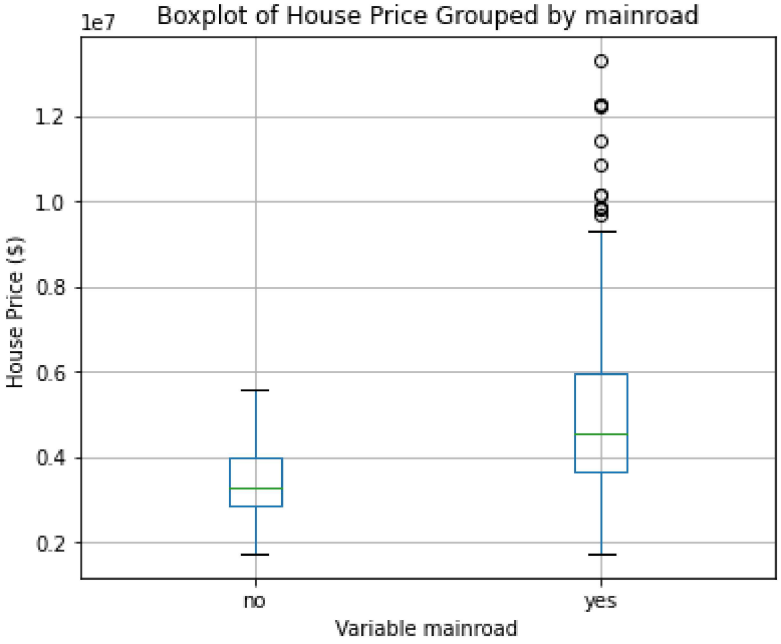
In [18]:
```
'''
This section of the code will utilize a for loop to plot box plots for the
```

```
categorical columns using the boxplot() function. The target response will
be home price per the scope of the project.
'''
df_cat = df[['bedrooms', 'bathrooms', 'stories', 'mainroad',
             'guestroom', 'basement','hotwaterheating','airconditioning',
             'parking', 'prefarea', 'furnishingstatus']]

for col in df_cat.columns:
    df.boxplot(column = 'price', by = col, figsize=(6,5))
    plt.title("Boxplot of House Price Grouped by {}".format(col), fontsize=12)
    plt.xlabel("Variable " +col, fontsize=10)
    plt.ylabel("House Price ($) ", fontsize=10)
    plt.suptitle('')
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()
```

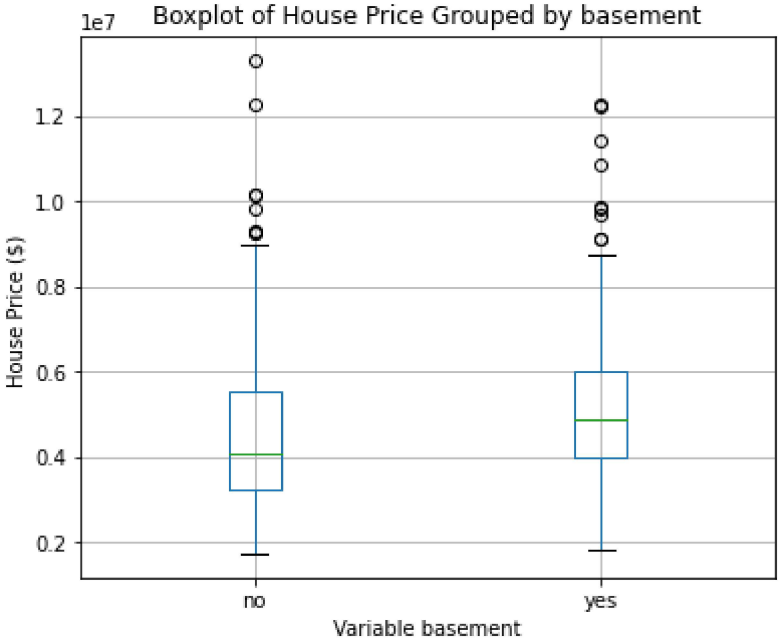


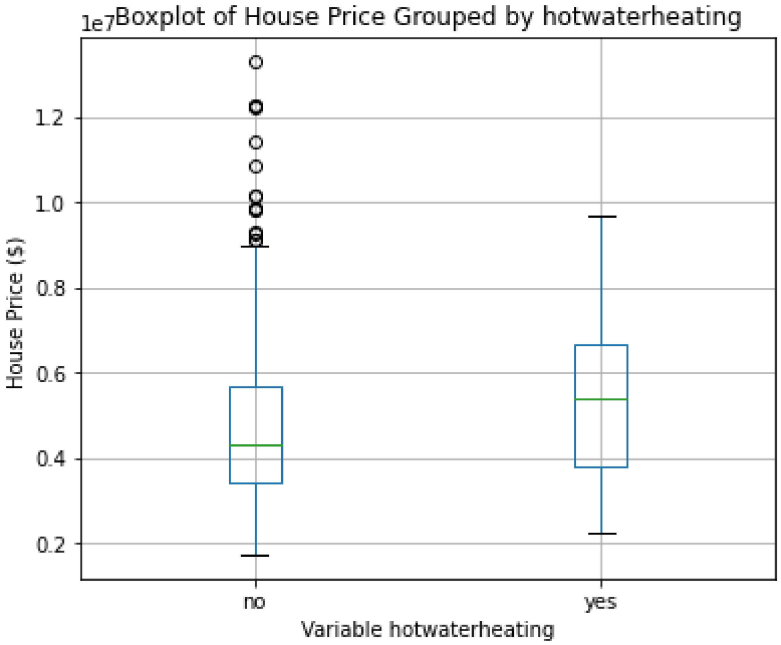


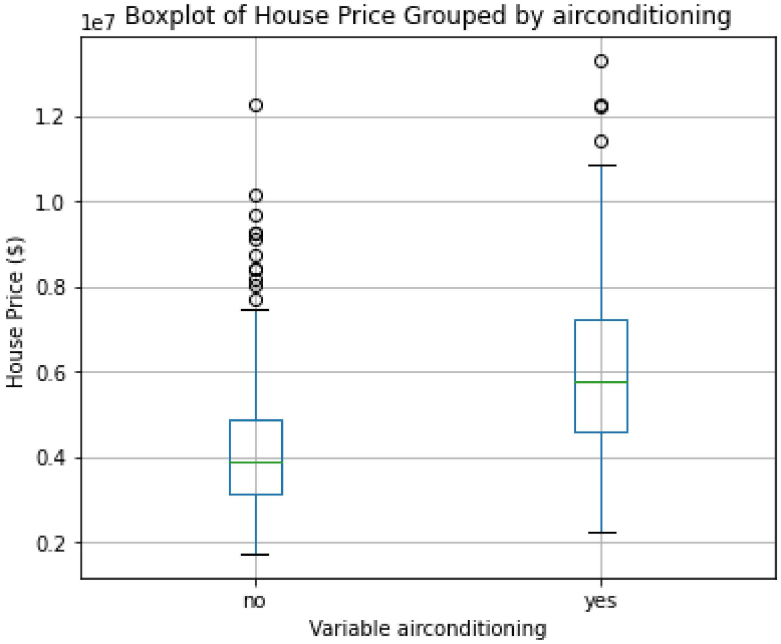

Boxplot of House Price Grouped by stories
1e7
1.2
1.0
0.8
0.6
0.4
0.2
House Price ($)
1
2
3
4
Variable stories


Boxplot of House Price Grouped by mainroad
1e7
1.2
1.0
0.8
0.6
0.4
0.2
House Price ($)
no
yes
Variable mainroad

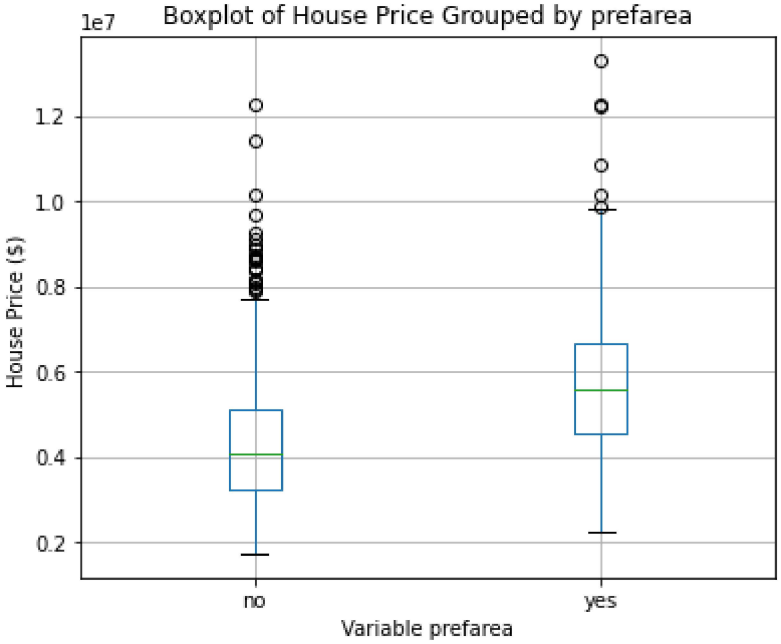

Boxplot of House Price Grouped by guestroom
1e7
1.2
1.0
0.8
0.6
0.4
0.2
House Price ($)
no
yes
Variable guestroom

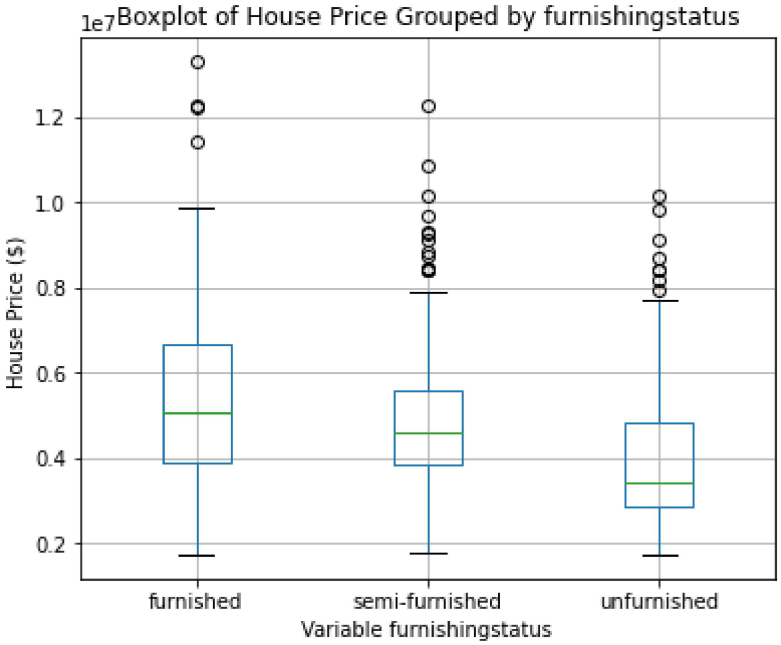

Boxplot of House Price Grouped by basement
1e7
1.2
1.0
0.8
0.6
0.4
0.2
House Price ($)
no
yes
Variable basement


1e7
Boxplot of House Price Grouped by hotwaterheating
House Price ($)
1.2
1.0
0.8
0.6
0.4
0.2
no
yes
Variable hotwaterheating


1e7
Boxplot of House Price Grouped by airconditioning
House Price ($)
1.2
1.0
0.8
0.6
0.4
0.2
no
yes
Variable airconditioning


Boxplot of House Price Grouped by parking
1e7
House Price ($)
Variable parking
0
1
2
3


Boxplot of House Price Grouped by prefarea
1e7
House Price ($)
Variable prefarea
no
yes

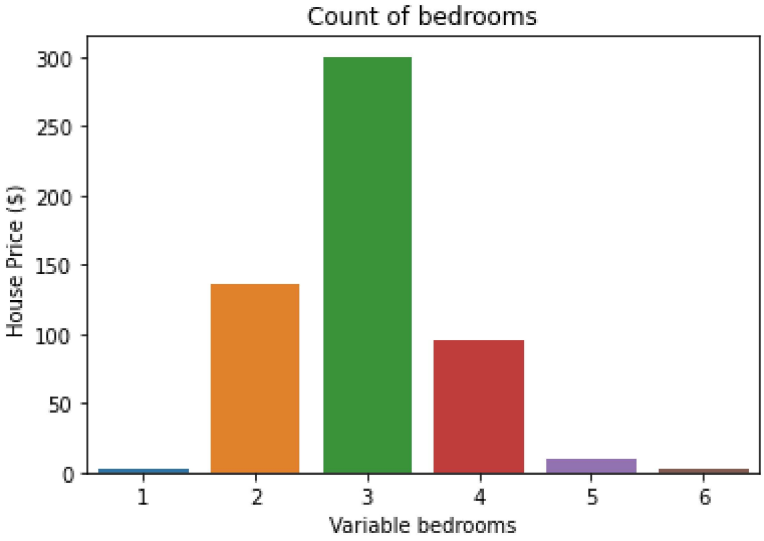


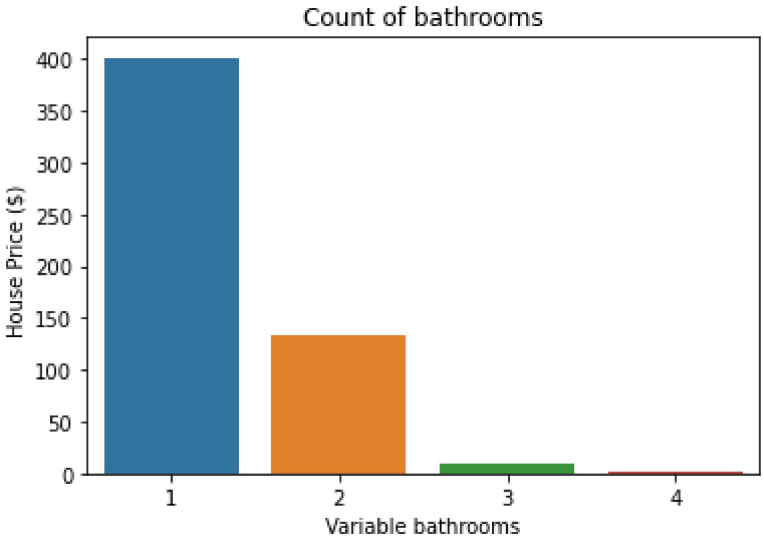
Observations - Box Plots of Categorical Features with Respect to House Price

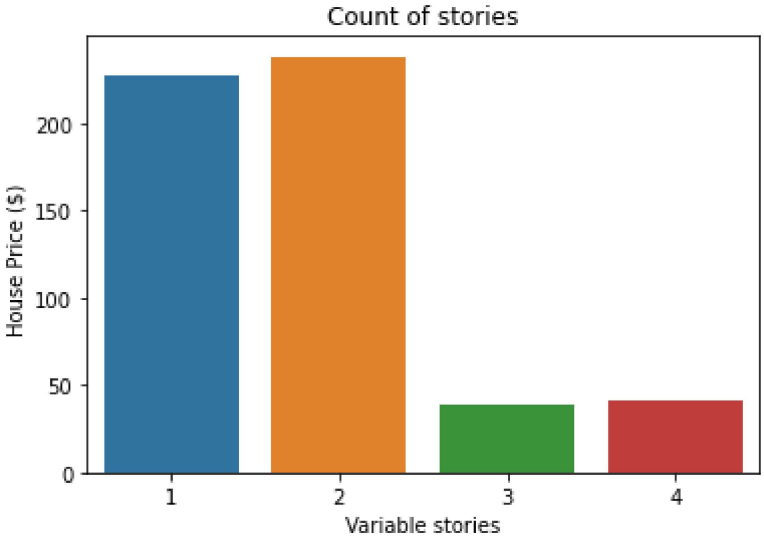
- House Price tended to increase as the number of bedrooms increased up until 5 bedrooms.
- House Price tended to increase as the number of bathrooms increased.
- House Price tended to increase as the number of stories increased.
- House Prices for homes connecting to a mainroad tended to be higher than homes that did not.
- House Prices for homes that included a guestroom tended to be higher than homes that did not.
- House Price was not influenced much by whether a basement was included. Homes that incldued a basement tended to be slightly higher than homes that did not.
- House prices for homes with hot water heaters tended to be slightly higher than those homes that did not.
- House Prices for homes that included air conditioning tended to be higher than homes that did not.
- House Prices for homes that included parking tended to be higher than homes that did not have any parking. However, the number of parking spaced did not relate to an increase in house price.
- House Prices for homes in a preferred location tended to be higher than homes that were not.
- Furnished and semi-furnished homes had higher House Prices compared to unfurnished homes.
- There appears to be some outliers whithin the dataset that may need to be cleaned up during the data preparation step.

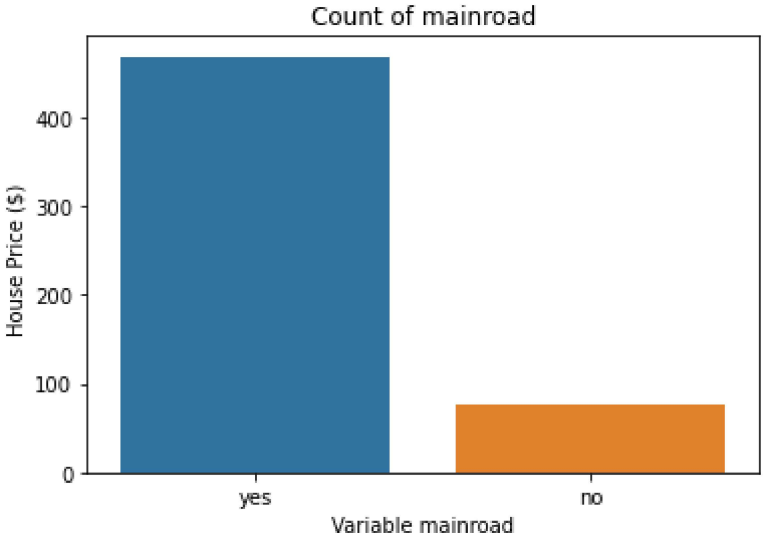
In [19]:
```
'''
This section of the code will utilize a for loop to create count plots for the
categorical columns using the countplot() function.
'''
df_cat = df[['bedrooms', 'bathrooms', 'stories', 'mainroad',
             'guestroom', 'basement','hotwaterheating','airconditioning',
             'parking', 'prefarea', 'furnishingstatus']]

for col in df_cat.columns:
    data = df[col]
```

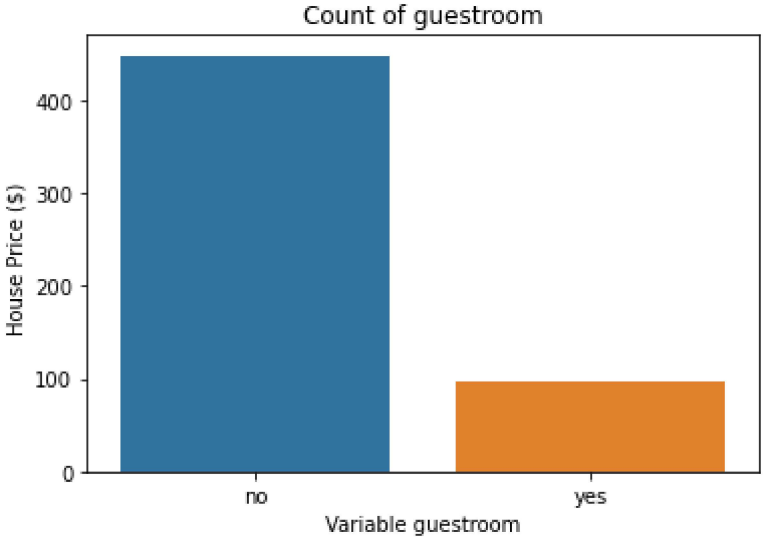
```
    sns.countplot(x = data)
    plt.title("Count of {}".format(col), fontsize=12)
    plt.xlabel("Variable " +col, fontsize=10)
    plt.ylabel("House Price ($) ", fontsize=10)
    plt.suptitle('')
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    plt.show()
```

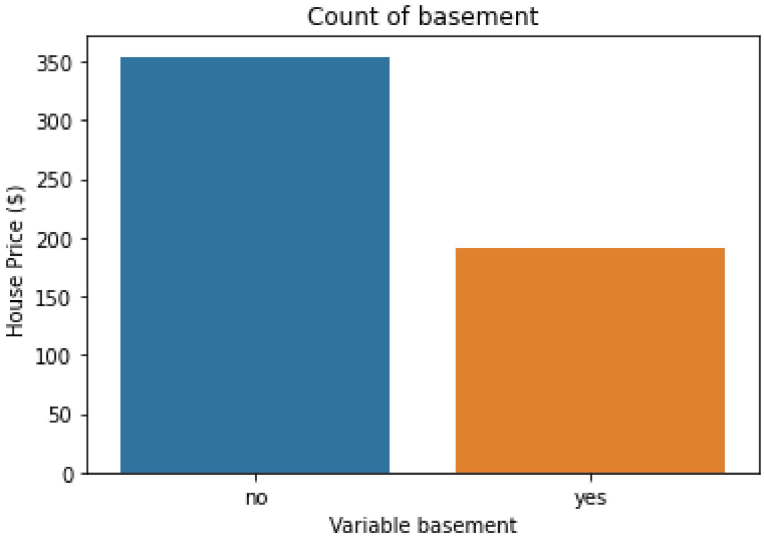


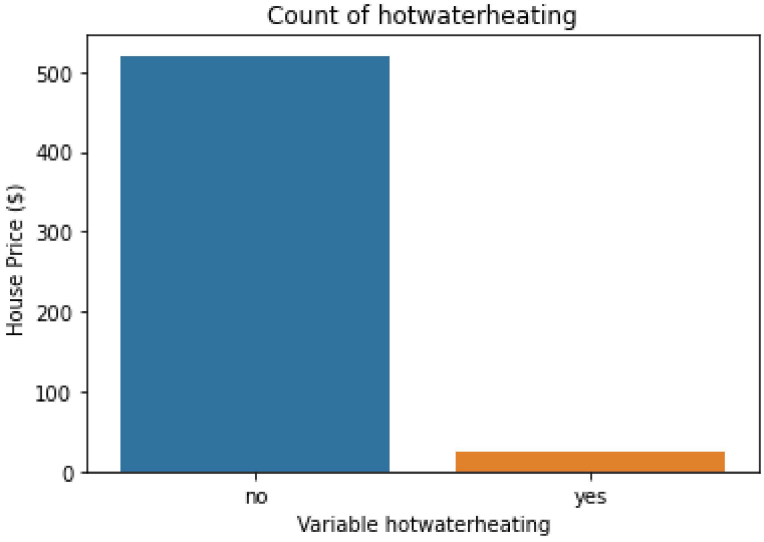


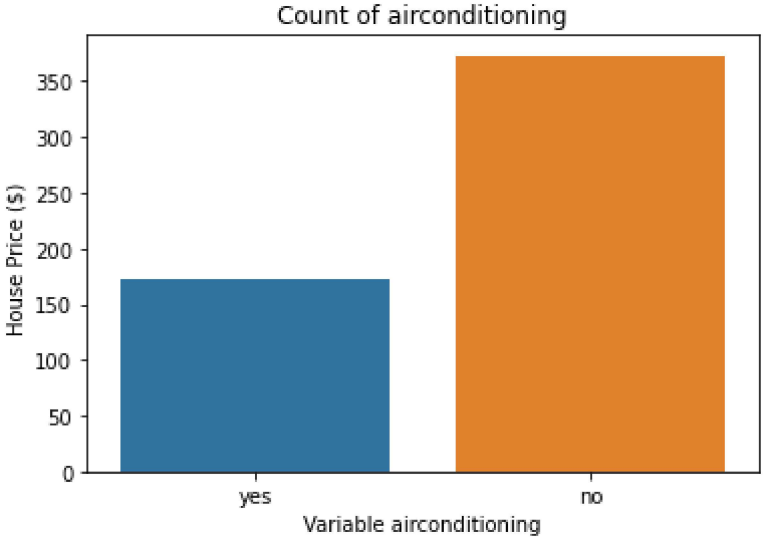

Count of stories
House Price ($)
0
50
100
150
200
1
2
3
4
Variable stories

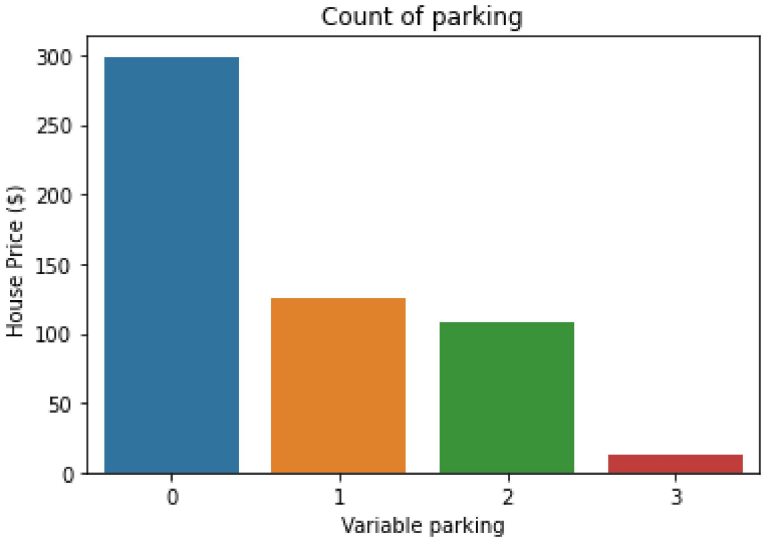

Count of mainroad
House Price ($)
0
100
200
300
400
yes
no
Variable mainroad

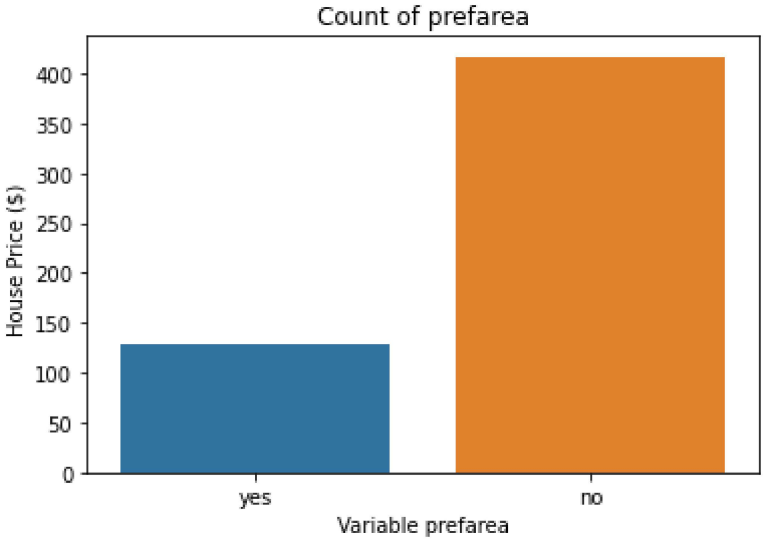

Count of guestroom
House Price ($)
0
100
200
300
400
no
yes
Variable guestroom

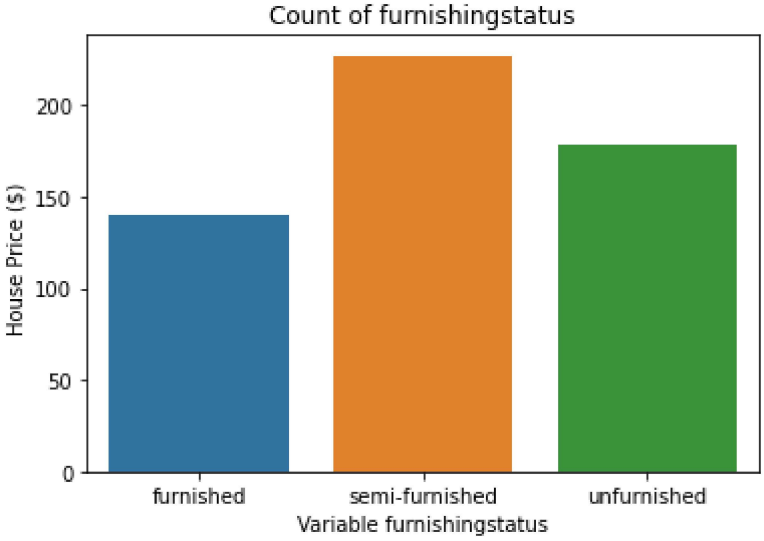

Count of basement
House Price ($)
0
50
100
150
200
250
300
350
no
yes
Variable basement


Count of hotwaterheating
House Price ($)
0
100
200
300
400
500
no
yes
Variable hotwaterheating


Count of airconditioning
House Price ($)
0
50
100
150
200
250
300
350
yes
no
Variable airconditioning







In [20]:
```
'''
Check value count of number of bedrooms.
'''
df['bedrooms'].value_counts()
```

Out[20]:
```
3    300
```

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

In [21]:
```
'''
Check value count of number of bathrooms.
'''
df['bathrooms'].value_counts()
```

Out[21]:
```
1    401
2    133
3     10
4      1
Name: bathrooms, dtype: int64
```

In [22]:
```
'''
Check value count of number of stories.
'''
df['stories'].value_counts()
```

Out[22]:
```
2    238
1    227
4     41
3     39
Name: stories, dtype: int64
```

In [23]:
```
'''
Check value count of number homes connected to a mainroad.
'''
df['mainroad'].value_counts()
```

Out[23]:
```
yes    468
no      77
Name: mainroad, dtype: int64
```

In [24]:
```
'''
Check value count of number homes that include a guestroom.
'''
df['guestroom'].value_counts()
```

Out[24]:
```
no     448
yes     97
Name: guestroom, dtype: int64
```

In [25]:
```
'''
Check value count of number homes that include a basement.
'''
df['basement'].value_counts()
```

Out[25]:
```
no     354
yes    191
Name: basement, dtype: int64
```

In [26]:

```
'''
Check value count of number homes that include a hot water heater.
'''
df['hotwaterheating'].value_counts()
```

Out[26]:
```
no     520
yes     25
Name: hotwaterheating, dtype: int64
```

In [27]:
```
'''
Check value count of number homes that include air conditioning.
'''
df['airconditioning'].value_counts()
```

Out[27]:
```
no     373
yes    172
Name: airconditioning, dtype: int64
```

In [28]:
```
'''
Check value count of number homes with parking spots.
'''
df['parking'].value_counts()
```

Out[28]:
```
0    299
1    126
2    108
3     12
Name: parking, dtype: int64
```

In [29]:
```
'''
Check value count of number homes in a preferred area.
'''
df['prefarea'].value_counts()
```

Out[29]:
```
no     417
yes    128
Name: prefarea, dtype: int64
```

In [30]:
```
'''
Check value count of furnishing status.
'''
df['furnishingstatus'].value_counts()
```

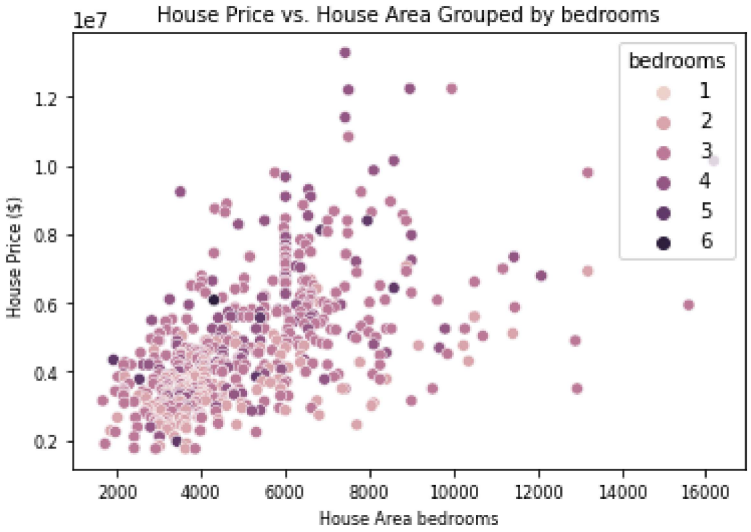
Out[30]:
```
semi-furnished    227
unfurnished       178
furnished         140
Name: furnishingstatus, dtype: int64
```

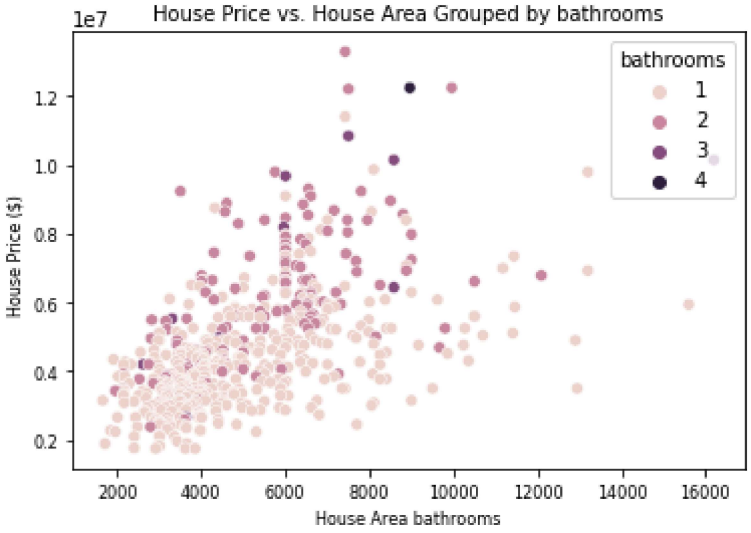
Observations - Countplots of Categorical Features

- Most houses had three bedrooms (QTY 300 homes).
- Most houses had one bathroom (QTY 401 homes).
- Most houses had one (QTY 227 homes) or two (QTY 238) stories.
- Majority of houses are connected to a mainroad. (QTY 468 Yes, QTY 77 No)
- Majority of houses do not include a guest bedroom. (QTY 97 Yes, QTY 448 No)

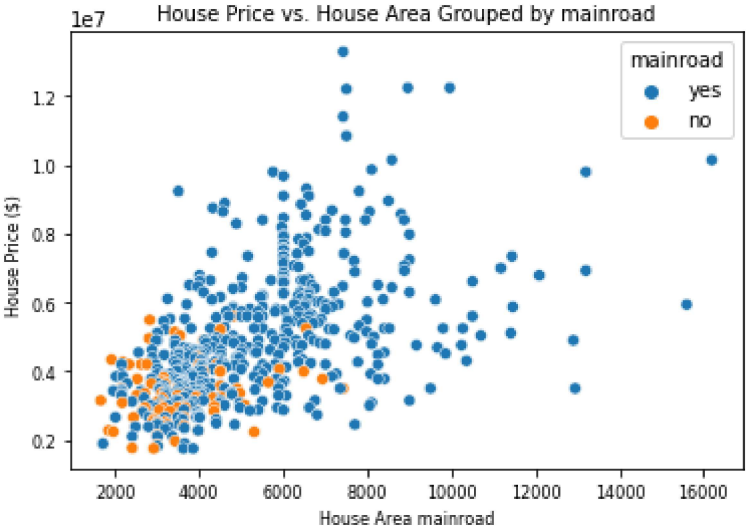
- Majority of houses do not include a basement. (QTY 191 Yes, QTY 354 No)
- Majority of houses do not include hot water heaters. (QTY 25 Yes, QTY 520 No)
- Majority of houses do not include air conditioning. (QTY 172 Yes, QTY 373 No)
- Majority of houses do not have a parking spot. (QTY 246 Yes, QTY 299 No)
- Majority of houses are not in a preferred area. (QTY 128 Yes, QTY 417 No)
- Majority of houses are either semi-furnished or furnished. (QTY 367 Yes, QTY 178 No)

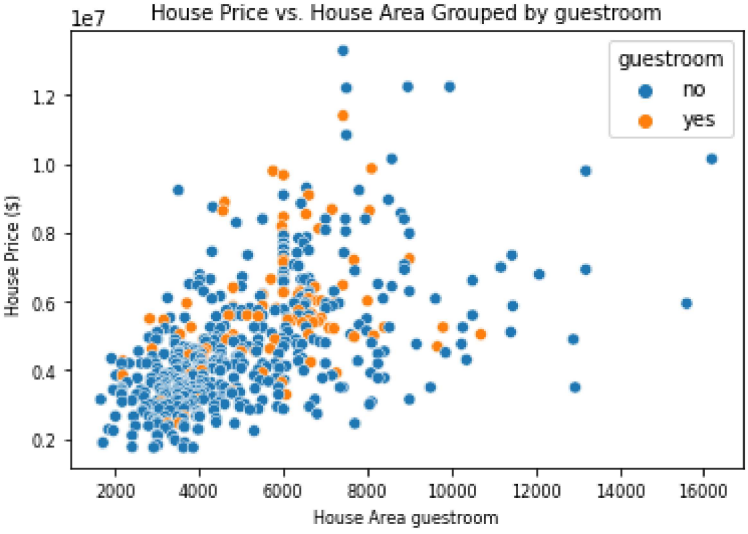
In [31]:
```
'''
Plot the house price vs home area on a scatter plot.
Also, group out the categorical features within the plots.
'''
for col in df_cat.columns:
    sns.scatterplot(x='area', y='price', data=df, hue = df[col])
    plt.title("House Price vs. House Area Grouped by {}".format(col), fontsize=10)
    plt.xlabel("House Area " +col, fontsize=8)
    plt.ylabel("House Price ($) ", fontsize=8)
    plt.suptitle('')
    plt.xticks(fontsize=8)
    plt.yticks(fontsize=8)
    plt.show()
```

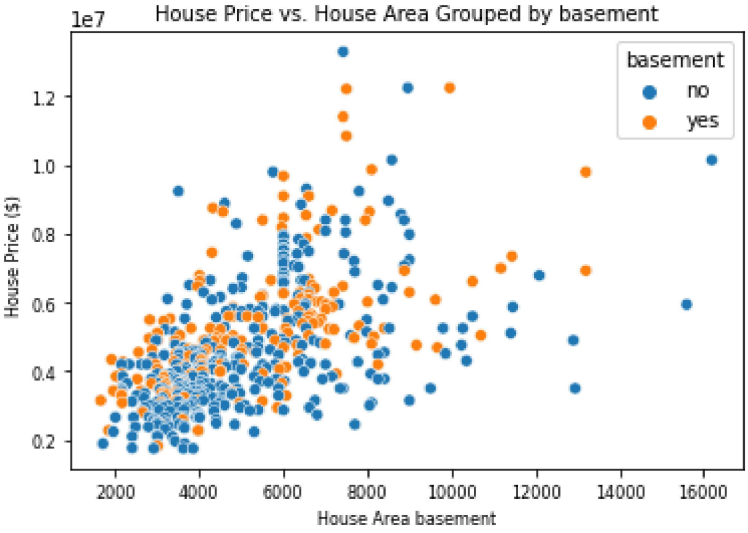


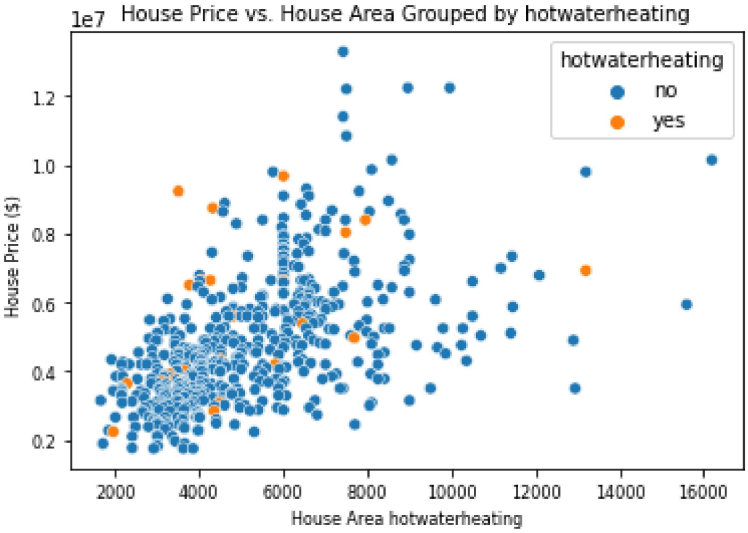


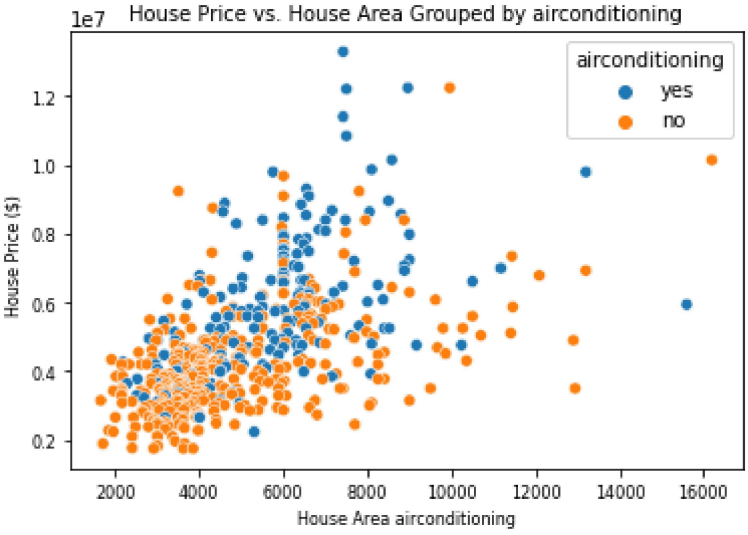

House Price vs. House Area Grouped by stories
1e7
stories
1
2
3
4
House Price ($)
House Area stories

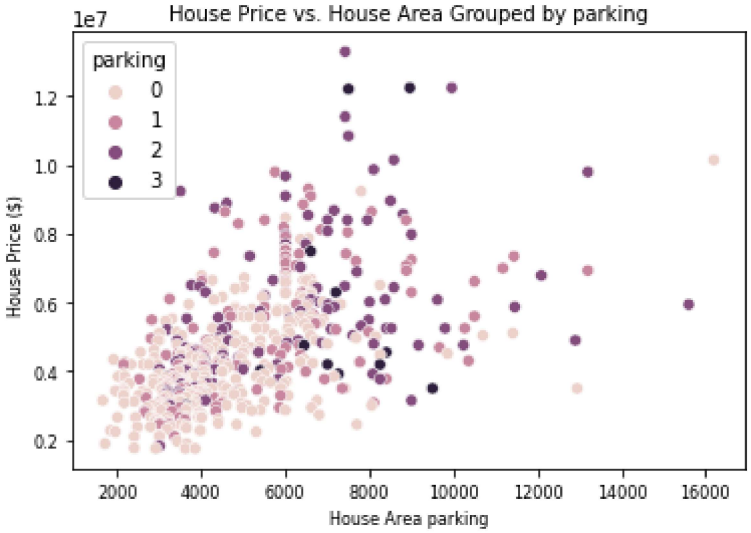

House Price vs. House Area Grouped by mainroad
1e7
mainroad
yes
no
House Price ($)
House Area mainroad

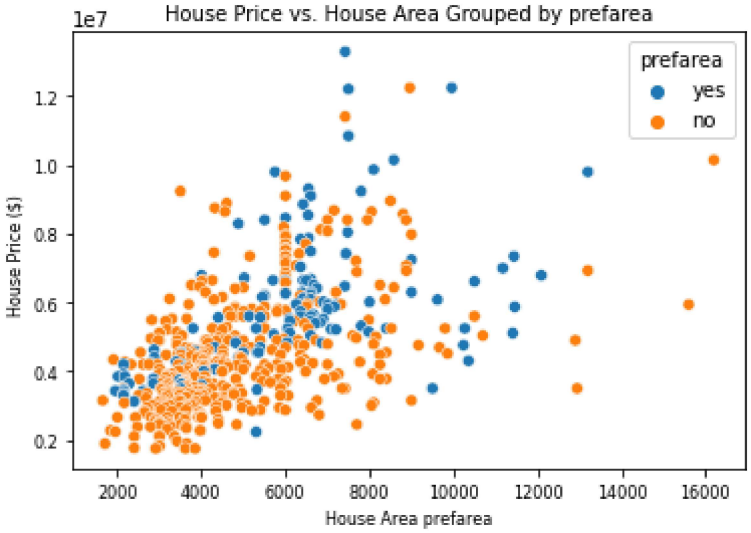

House Price vs. House Area Grouped by guestroom
1e7
guestroom
no
yes
House Price ($)
House Area guestroom

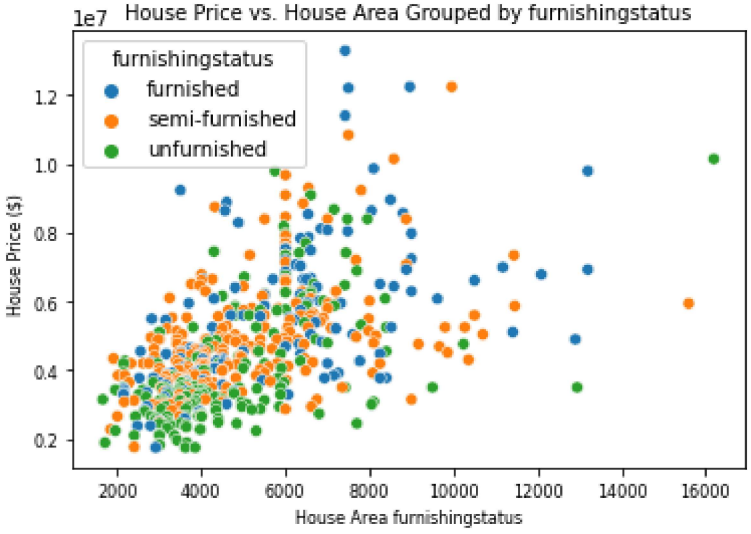

House Price vs. House Area Grouped by basement
1e7
basement
no
yes
House Price ($)
House Area basement
1.2
1.0
0.8
0.6
0.4
0.2
2000
4000
6000
8000
10000
12000
14000
16000


House Price vs. House Area Grouped by hotwaterheating
1e7
hotwaterheating
no
yes
House Price ($)
House Area hotwaterheating
1.2
1.0
0.8
0.6
0.4
0.2
2000
4000
6000
8000
10000
12000
14000
16000


House Price vs. House Area Grouped by airconditioning
1e7
airconditioning
yes
no
House Price ($)
House Area airconditioning
1.2
1.0
0.8
0.6
0.4
0.2
2000
4000
6000
8000
10000
12000
14000
16000







In [32]:
```
'''
Utilize the pairplot() function from seaborn to understand the relationship between all
'''
sns.pairplot(df).map_upper(sns.kdeplot, levels = 4, color = ".7")
```

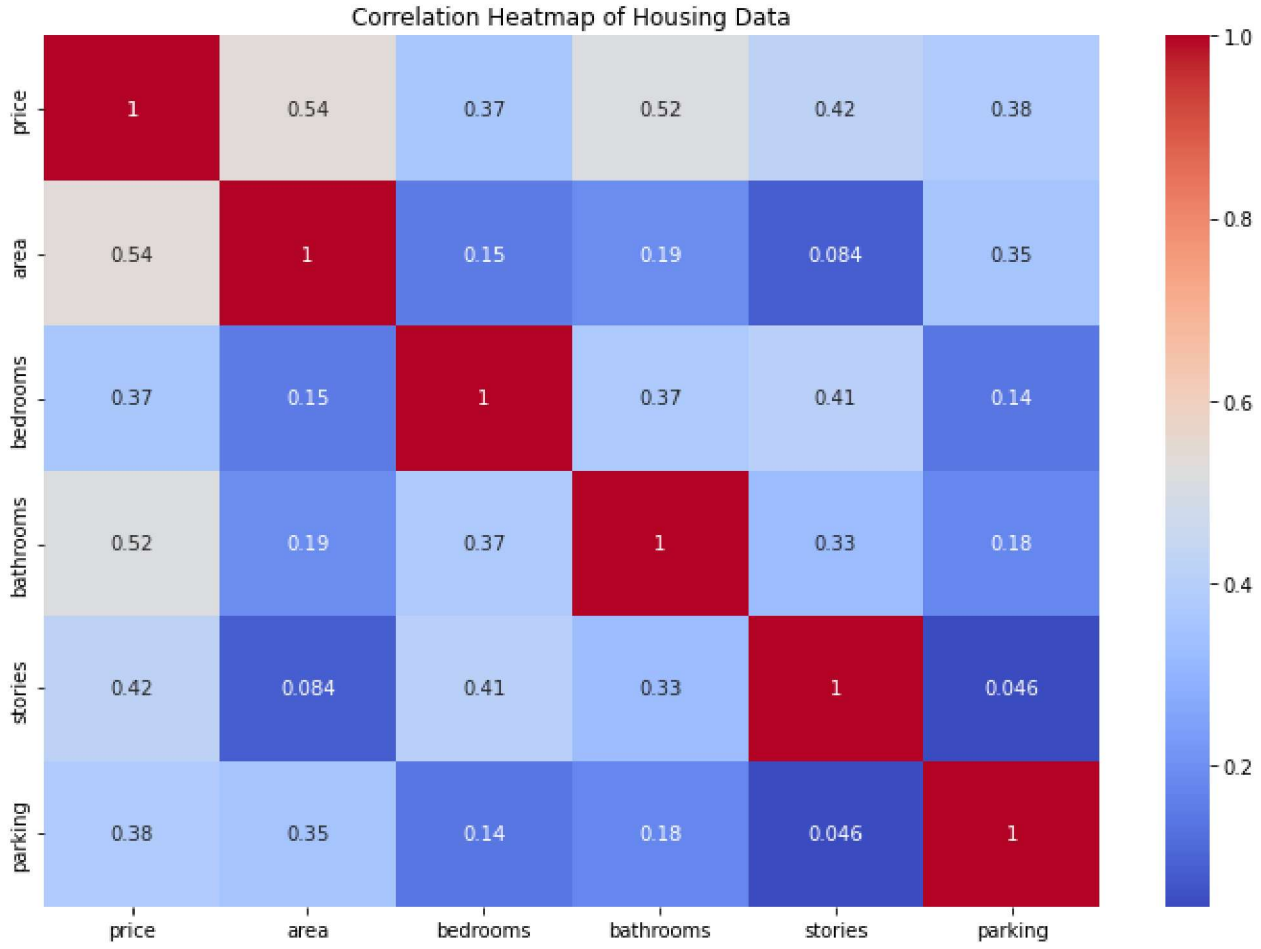
Out[32]:
```
<seaborn.axisgrid.PairGrid at 0x1e9efc8b670>
```

In [33]:

```
'''
Display a correlation heatmap. Utilize sns.heatmap() to generate the figure.
'''
# Calculate the correlation coefficient with corr().
correlation_number = df.corr()

# Create the heatmap for the correlation coefficients calculated above.
fig, ax = plt.subplots(1, 1, figsize=(10,7), tight_layout = True)
sns.heatmap(correlation_number, annot = True, cmap = 'coolwarm')
plt.title('Correlation Heatmap of Housing Data')
```

Out[33]:

```
Text(0.5, 1.0, 'Correlation Heatmap of Housing Data')
```



Observations - Scatterplots, Pairplot, and Correlation Heat Map

● All the features are positively correlated with price, however the strength of the correlation seems to be moderate to low.
● Scatterplots provide an alternative way to illustrate the findings from the box plots previously observed.
● It is recommended to create dummy variables for the remaining categorical variables:'mainroad','guestroom', 'basement','hotwaterheating','airconditioning','prefarea', and 'furnishingstatus'. This can be completed after the data preparation step and the correlation matrix can be regenerated.
● The features with the strongest correlation with House Price are Home Area, Number of Bathrooms, and Number of Stories respectively.
● Collinearity does not appear to exist based on the correlation heatmap, however this should be checked again after the dummy variables are created for the additional categorical variables mentioned above.

Conclusions from Milestone 1 Questions

- The distribution of House Prices is positvely skewed. There is a difference of $\$426729$ between the mean and median values. This suggests that there may be some data points significantly influencing the mean of the Housing Price data.

- There is not a great balance for the categorical features within this dataset. Majority of homes had three bedrooms, one bathroom, contained one or two stories, were connected to a mainroad, did not include a guest bedroom, did not include a basement, did not have hot water heating, did not have a parking spot, were not located in a preferred location, and were either fully furnished or semi-furnished. The countplots illustrate the inbalances with value_counts listed in the preceeding section.
- There is a direct relationship between House Price and House Area as seen in the scatterplots. As Home Area increased, the House Price tended to increase. Logically, the visualizations from the box plots make sense to explain the relationship between the categorical variables and House Price. Homes with more bedrooms, bathrooms, stories, connections to mainroads, guestrooms included, basements included, hot water heating included, air conditioning included, parking spots included, in a preferred area, and furnished (to some degree) tended to have higher House Prices compared to homes that did not. The degree at which the House Prices increase did vary for each feature suggesting the relationship was stronger for some features compared to others.
- The features with the strongest correlation with House Price are Home Area, Number of Bathrooms, and Number of Stories respectively. All of the features with numeric data currently indicate a positive correlation with House Price, however the correlation strength appears to be moderate to low.
- There are no missing values in this dataset. The dataset initially contains 545 rows and 13 columns. During the data preparation step (next phase), it is recommended to create dummy variables for the remaining categorical variables. Recheck collinearity with a correlation heatmap once further with the data preparation phase.
- Ethical Considerations - The steps for overviewing and exploring the dataset are outlined in this Milestone 1 report. The original dataset was found through Kaggle from the poster/author M Yasser H. The purpose of this project is to construct various regression models from this dataset in order to predict House Prices. The sole intent for this report is to satisfy the requirements for Milestone 1 for the Term Project for DSC 550. Any alternative uses of this dataset or model shall require addtional ethical considerations for the end purpose or scope.

Milestone 2: Data Preparation

The second milestone will involve Tasks 3, 4, and 5 listed in the project introduction section. The data preparation steps will be outlined throughout this section. The end goal for this section is to ensure the data is ready for the model building/evaluation phase. During the previous section, the data set was found to have no missing values. The high-level data preparation steps that will be followed for this section are:

- Handling Outliers
- Eliminate Nonessential Features for the Model
- Split into Training and Test Data Sets (80% Training, 20% Test)
- Convert Categorical Features to Dummy Variables
- Perform Feature Selection (Use for alternative model with reduced features)

In [34]:
```
'''
Remove the data points that fall outside the inner quartile range of the data set for t
Use the interquartile range method to remove the outliers.
Begin by identifying the Q1, Q3, and InterQuartile Range for the pricing data.
'''
q1 = np.percentile(df['price'], 25, interpolation = 'midpoint')
q3 = np.percentile(df['price'], 75, interpolation = 'midpoint')
iqr = q3-q1
print('Q1:${}\n'
      'Q3:${}\n'
      'IQR:${}'.format(q1,q3,iqr))
```

```
Q1:$3430000.0
Q3:$5740000.0
IQR:$2310000.0
```

In [35]:
```
'''
Identify the  upper and lower bounds to locate the outliers.
'''
upper = np.where(df['price'] >= (q3 + 1.5*(iqr)))
lower = np.where(df['price'] <= (q1 - 1.5*(iqr)))
```

In [36]:
```
print('Upper Outliers Identified:{}'.format(len(upper[0])))
print('Lower Outliers Identified: {}'.format(len(lower[0])))
```

```
Upper Outliers Identified:15
Lower Outliers Identified: 0
```

In [37]:
```
'''
Remove the outliers from the data frame and store under new data frame df1.
Original Data Frame with outliers is df.
New Data Frame with outliers removed is df1.
'''
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)
```

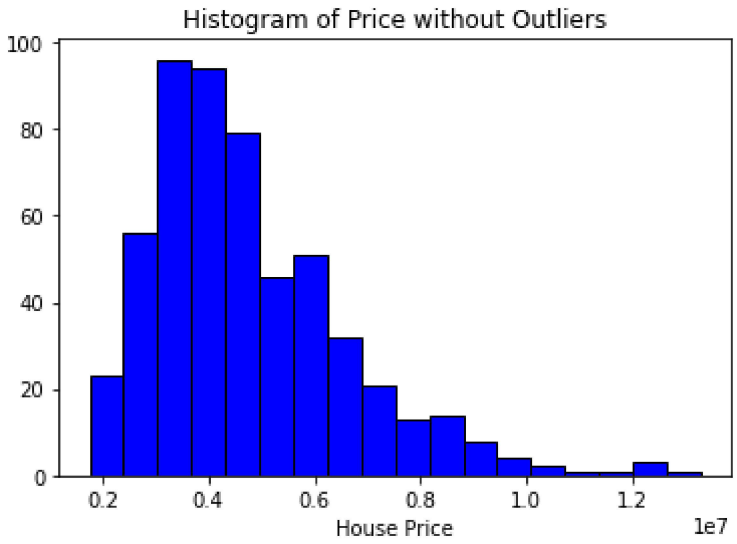
In [38]:
```
'''
Understand the shape of the revised data frame.
'''
print('There are {} rows and {} columns in the original data frame.'.format(df.shape[0]
```

```
There are 530 rows and 13 columns in the original data frame.
```

There were 15 outliers removed from the original data set. Revisit the price distribution for the housing data set as well as the correlation heatmap to understand if this may have changed any of the initial observations.

In [39]:
```
'''
Show a visual of the House Price distribution after the outliers were removed.
'''
plt.hist(df_num['price'], color = 'blue', edgecolor='k', bins = 18)
plt.title("Histogram of Price without Outliers".format(col), fontsize=12)
plt.xlabel("House Price", fontsize=10)
plt.xticks(fontsize=10)
```

```
plt.yticks(fontsize=10)
plt.show() # plt.show()
```



In [40]:
```
'''
Calculate the difference between the mean and median for House Price post outlier remov
'''
mean_price = round(df['price'].mean(),0)
median_price = round(df['price'].median(),0)
difference = round((mean_price) - (median_price),0)
print('Mean House Price: ${}\n'
      'Median House Price: ${}\n'
      'Difference of Mean and Median House Price: ${}'.format(mean_price, median_price,
```

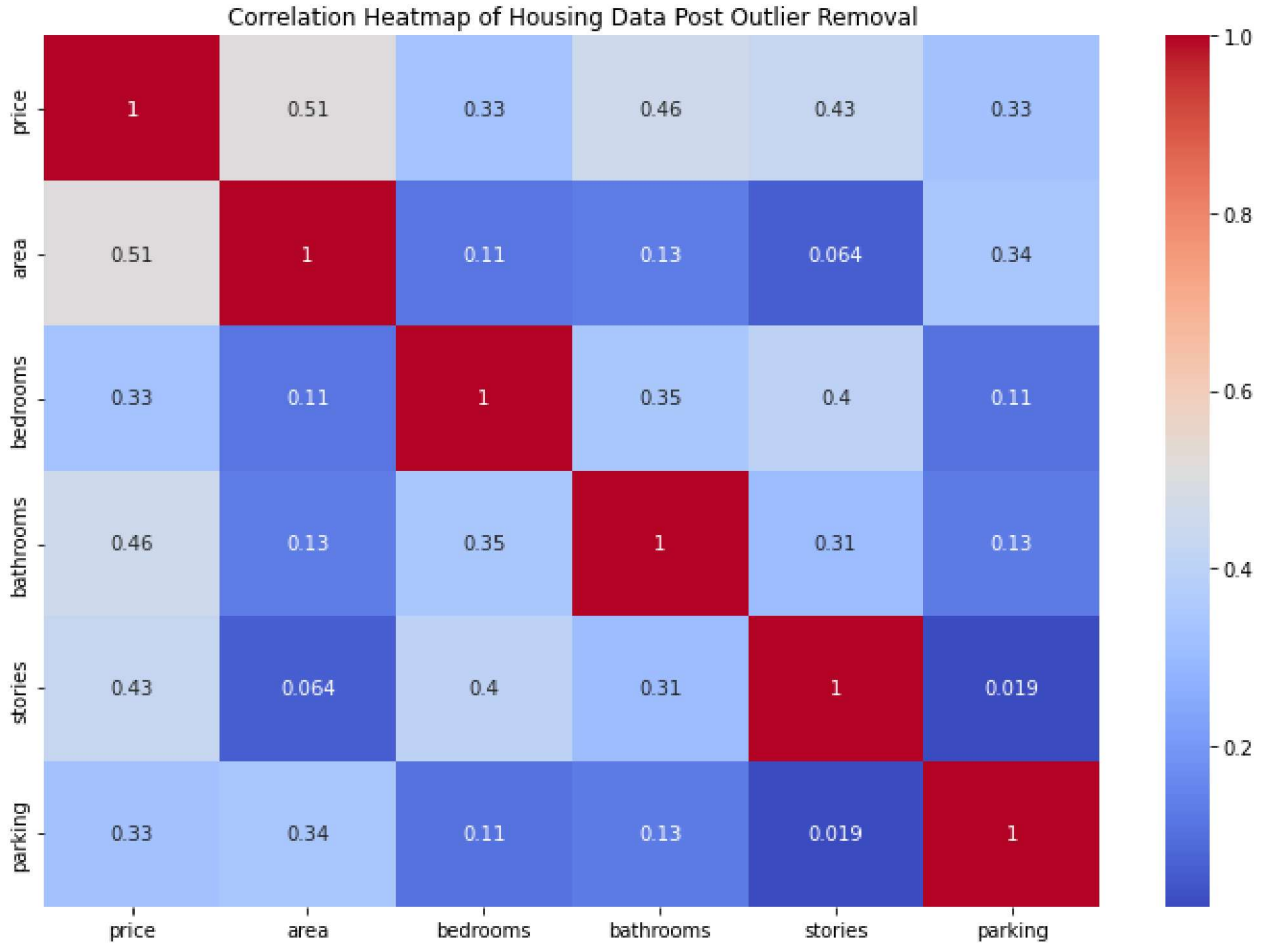
```
Mean House Price: $4600663.0
Median House Price: $4270000.0
Difference of Mean and Median House Price: $330663.0
```

In [41]:
```
"""
Check the Fisher_Pearson correlation of skew for House Price with skew() after the outl
"""
print('Skewness for data after outliers were removed:', skew(df['price']))
```

```
Skewness for data after outliers were removed: 0.6937839875453022
```

In [42]:
```
'''
Display a correlation heatmap after outliers were removed. Utilize sns.heatmap() to gen
'''
# Calculate the correlation coefficient with corr().
correlation_number_2 = df.corr()

# Create the heatmap for the correlation coefficients calculated above.
fig, ax = plt.subplots(1, 1, figsize=(10,7), tight_layout = True)
sns.heatmap(correlation_number_2, annot = True, cmap = 'coolwarm')
plt.title('Correlation Heatmap of Housing Data Post Outlier Removal')
```

Out[42]:
```
Text(0.5, 1.0, 'Correlation Heatmap of Housing Data Post Outlier Removal')
```



Observations - Outlier Removal

● There were 15 rows removed from the data set. These rows contained outliers above the upper bound (Q3 + 1.5*IQR) for House Price.

● The difference between the mean and median House Price was reduced to $330663. The difference prior to the outlier removal was $426729.

● The positive skew for the House Price distribution was reduced down to a Fisher_Pearson correlation value of 0.694. Prior to the outliers being removed, the Fisher_Pearson correlation was 1.21.

● The revised correlation matrix is shown in the heatmap above. Price and House Area still appear to have the highest correlation based on the matrix coefficients.

In [43]:
```
'''
Remove the nonessential features from the data frame prior to splitting the data into t
'''
df.drop('hotwaterheating', inplace = True, axis = 1)
```

In [44]:
```
'''
Understand the shape of the revised data frame.
'''
print('There are {} rows and {} columns in the original data frame.'.format(df.shape[0]
```

```
There are 530 rows and 12 columns in the original data frame.
```

Observations - Removal of Nonessential Features

- Removed the 'hotwaterheating' column since the balance was poor for homes with a hot water heater. Only 4 percent of homes in this data set actually had a hot water heater.
- All other features will be retained until the feature selection phase.

In [45]:
```
'''
Prior to performing any additional data preparation tasks, the data will be split into
a training and test set. Start by defining the features and target variables as X and y
The target variable will be price and the remaining columns will be the initial feature
'''
X = df.drop('price', axis = 1)
y = df['price']
```

In [46]:
```
'''
Utilize the train_test_split() function to split data into a training and test data set
I'll be splitting the entire data set into 80% Training and 20% Test.
'''
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state
```

In [47]:
```
'''
Understand/Verify the shape of the training and test data sets just created.
'''
print("The X_train shape is {} rows and {} columns.".format(X_train.shape[0],X_train.sh
print("The y_train shape is {} rows.".format(y_train.shape[0]))
print("The X_test shape is {} rows and {} columns.".format(X_test.shape[0],X_test.shape
print("The y_test shape is {} rows.".format(y_test.shape[0]))
```

```
The X_train shape is 424 rows and 11 columns.
The y_train shape is 424 rows.
The X_test shape is 106 rows and 11 columns.
The y_test shape is 106 rows.
```

In [48]:
```
'''
Create Dummy Variables for the categorical columns with first dummy variable dropped to
'''
X_train = pd.get_dummies(X_train, drop_first = True)
X_test = pd.get_dummies(X_test, drop_first = True)
```

In [49]:
```
'''
Understand/Verify the shape of the training and test feature data sets with the dummy v
'''
print("The X_train shape is {} rows and {} columns.".format(X_train.shape[0],X_train.sh
print("The X_test shape is {} rows and {} columns.".format(X_test.shape[0],X_test.shape
```

```
The X_train shape is 424 rows and 12 columns.
The X_test shape is 106 rows and 12 columns.
```

In [50]:
```
'''
View the first 5 rows of the X_train data set.
'''
X_train.head()
```

Out[50]:

	area	bedrooms	bathrooms	stories	parking	mainroad_yes	guestroom_yes	basement_yes	airco
152	5400	5	1	2	0	1	1	1	
541	2400	3	1	1	0	0	0	0	
432	6060	3	1	1	0	1	1	1	
532	3000	2	1	1	0	0	0	0	
84	3760	3	1	2	2	1	0	0	

In [51]:
```
'''
View the first 5 rows of the X_test data set.These should match without any issues.
'''
X_test.head()
```

Out[51]:

	area	bedrooms	bathrooms	stories	parking	mainroad_yes	guestroom_yes	basement_yes	airco
155	6100	3	2	1	2	1	0	1	
413	1950	3	2	2	0	1	0	1	
21	7155	3	2	1	2	1	1	1	
349	4820	3	1	2	0	1	0	0	
337	2145	4	2	1	0	1	0	1	

Observations - Train and Test Data Set Split (80% Train, 20% Test)

- The feature training data set (X_train) contains 424 rows and 12 columns.
- The feature test data set (X_test) contains 106 rows and 12 columns.
- The categorical features were converted to dummy variables and the first column was dropped to prevent collinearity between terms.

In [52]:
```
'''
Perform PCA so that 90% of the variance is retained.
Standardize the X_train and X_test datasets.
'''
sc = StandardScaler()
X_train_standardized = sc.fit_transform(X_train)
X_test_standardized = sc.transform(X_test)
```

In [53]:
```
'''
Create a PCA that will retain 90% of the variance.
'''
pca = PCA(n_components=0.90, whiten = True)
```

In [54]:
```
'''
```

```
Conduct PCA on the X_train_standardized datasets.
'''
X_train_pca = pca.fit_transform(X_train_standardized)
```

In [55]:
```
print('Original number of features:', X_train.shape[1])
print('Reduced number of features in PCA-transformed matrix:', X_train_pca.shape[1])
```

```
Original number of features: 12
Reduced number of features in PCA-transformed matrix: 10
```

There are 10 features in the PCA-transformed matrix.

In [56]:
```
'''
Use the transform() method only on the X_test features with the PCA that will retain 90
Transform but DO NOT fit the test features with the same PCA.
'''
X_test_pca = pca.transform(X_test_standardized)
```

In [57]:
```
'''
Apply a min-max scaler on the X_train dataset (original training features).
'''
scaler = MinMaxScaler()
X_train_mms = scaler.fit_transform(X_train)
```

In [58]:
```
'''
Create a thresholder with VarianceThreshold(). Use this to find the min-max scaled feat
variance above 0.1.
'''
thresholder = VarianceThreshold(threshold = 0.1)
```

In [59]:
```
'''
Create a high variance feature matrix based on X_train_mms data.
'''
train_features_high_variance = thresholder.fit_transform(X_train_mms)
```

In [60]:
```
'''
Display the matrix of features with high variance from training set.
'''
train_features_high_variance[0:5]
```

Out[60]:
```
array([[1., 1., 1., 1., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [1., 1., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 1., 0.]])
```

In [61]:
```
print('Original number of features:', X_train.shape[1])
print('Reduced number of high variance features:', train_features_high_variance.shape[1
```

```
Original number of features: 12
Reduced number of high variance features: 7
```

In [62]:
```
'''
Use the transform() method only on the X_test features with the thresholder set at 0.1.
Start with the min-max scaler on X_test.
'''
X_test_mms = scaler.transform(X_test)
```

In [63]:
```
'''
Use the transform() method only on the X_test features with the thresholder set at 0.1.
Find the min-max features in the X_test dataset.
Create a high variance feature matrix based on X_test_mms.
'''
test_features_high_variance = thresholder.transform(X_test_mms)
```

In [64]:
```
'''
Display the matrix of features with high variance from test set.
'''
test_features_high_variance[0:5]
```

Out[64]:
```
array([[1., 0., 1., 0., 1., 0., 0.],
       [1., 0., 1., 0., 1., 0., 1.],
       [1., 1., 1., 1., 0., 0., 1.],
       [1., 0., 0., 0., 0., 1., 0.],
       [1., 0., 1., 0., 1., 0., 1.]])
```

Observations - Feature Selection with PCA and Variance Threshold

● The original training and test feature set will consider 12 features.

● A second training and test data set was generated based on Principal Component Analysis (PCA). These training and test data sets will be labeled as X_train_pca and X_test_pca. These data sets were reduced to 10 features based on 90% of the variance being retained.

● A third training and test data set were generated based on Variance Threshold. The features with higher variance were stored under train_features_high_variance and test_features_of_high_variance respectively. The number of features was reduced from 12 down to 7 for this feature matrix.

Summary of Data Preparation Steps.

● Removed outliers from the data set using interquartile range method. Checked to see how the removal of the outliers influenced the House Price distribution. The removal of the outliers reduced the skew and difference between the mean and median values.

● Removed nonessential column of data, hot water heating, since only ~4% of houses actually do have hot water heaters.

● Split data into training and test sets (80% Training, 20% Test)

● Created dummy variables for the categorical data. The first dummy variable that was created was dropped to eliminate collinearity between dummy variables.

● Performed feature selection with training and test data sets. First training and test data set include all the features after the dummy variables were created. The second training and test data set had PCA applied to reduce the features down to 10 based on 90% of variance being retained. The third training and test data set included a Varianve Threshold to reduce the features down to 7 based on a threshold set to 0.1.

● The data preparation performed in this section allows for several options to be performed during the model building and evaluation phase.

Milestone 3: Model Building and Evaluation

Milestone 3 will involve Tasks 6 and 7 outlined in the project introduction section. This section will provide a step-by-step outline of four models to predict house prices utilizing the above training and test data sets. The four models will be a Linear Regression, Decision Tree, Random Forest Classiffication, and Lasso Regression. The problem being addressed in this case involves supervised learning with a continuous target variable. Once the models are created, trained, and validated, then metrics will be set to select the best model for this problem. The known metrics at this point in time will be R-Squared, Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error. Each of the models will be evaluated with the original, PCA, and Variance Threshold training and test data. Lastly, a short conclusion will summarize all the results drawn from this part of the project.

Linear Regression Model (Min-Max Scaler, No PCA or Variance Threshold Performed)

In [126…

```
'''
Create a Linear Regression Model.
Fit the model to the training data sets (without PCA or Variance Threshold Performed)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
model_lr_prediction = model_lr.predict(X_test)
model_lr_prediction_train = model_lr.predict(X_train)
```

In [164…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
lr_mae = mean_absolute_error(y_test, model_lr_prediction)
lr_mse = mean_squared_error(y_test, model_lr_prediction)
lr_rmse = np.sqrt(lr_mse)
lr_r2 = r2_score(y_test, model_lr_prediction)
lr_r2_train = r2_score(y_train, model_lr_prediction_train)
print("MAE of the Linear Regression Model is:", round(lr_mae,4))
print("MSE of the Linear Regression Model is:", round(lr_mse,4))
print("RMSE of the Linear Regression Model is:", round(lr_rmse,4))
print("R2 Score of the Linear Regression Model on Test Data is:", round(lr_r2,4))
print("R2 Score of the Linear Regression Model on Train Data is:", round(lr_r2_train,4)
```

```
MAE of the Linear Regression Model is: 776999.581
MSE of the Linear Regression Model is: 1136826208405.7507
RMSE of the Linear Regression Model is: 1066220.5252
R2 Score of the Linear Regression Model on Test Data is: 0.6717
R2 Score of the Linear Regression Model on Train Data is: 0.6457
```

Linear Regression Model (Standard Scaler, PCA)

In [128…
```
'''
Create a Linear Regression Model.
Fit the model to the training data sets (PCA)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_lr_pca = LinearRegression()
model_lr_pca.fit(X_train_pca, y_train)
model_lr_pca_prediction = model_lr_pca.predict(X_test_pca)
model_lr_pca_prediction_train = model_lr_pca.predict(X_train_pca)
```

In [163…
```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
lr_mae_pca = mean_absolute_error(y_test, model_lr_pca_prediction)
lr_mse_pca = mean_squared_error(y_test, model_lr_pca_prediction)
lr_rmse_pca = np.sqrt(lr_mse_pca)
lr_r2_pca = r2_score(y_test, model_lr_pca_prediction)
lr_r2_pca_train = r2_score(y_train, model_lr_pca_prediction_train)
print("MAE of the Linear Regression Model (PCA) is:", round(lr_mae_pca,4))
print("MSE of the Linear Regression Model (PCA) is:", round(lr_mse_pca,4))
print("RMSE of the Linear Regression Model (PCA) is:", round(lr_rmse_pca,4))
print("R2 Score of the Linear Regression Model (PCA) on Test Data is:", round(lr_r2_pca
print("R2 Score of the Linear Regression Model (PCA) on Train Data is:", round(lr_r2_pc
```

```
MAE of the Linear Regression Model (PCA) is: 798568.4759
MSE of the Linear Regression Model (PCA) is: 1190875255533.897
RMSE of the Linear Regression Model (PCA) is: 1091272.3104
R2 Score of the Linear Regression Model (PCA) on Test Data is: 0.6561
R2 Score of the Linear Regression Model (PCA) on Train Data is: 0.6396
```

Linear Regression Model (Min-Max Scaler, Variance Threshold Performed)

In [130…
```
'''
Create a Linear Regression Model.
Fit the model to the training data sets (Variance Threshold Performed)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_lr_vt = LinearRegression()
model_lr_vt.fit(train_features_high_variance, y_train)
model_lr_vt_prediction = model_lr_vt.predict(test_features_high_variance)
model_lr_vt_prediction_train = model_lr_vt.predict(train_features_high_variance)
```

In [162…
```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
lr_mae_vt = mean_absolute_error(y_test, model_lr_vt_prediction)
lr_mse_vt = mean_squared_error(y_test, model_lr_vt_prediction)
lr_rmse_vt = np.sqrt(lr_mse_vt)
lr_r2_vt = r2_score(y_test, model_lr_vt_prediction)
lr_r2_vt_train = r2_score(y_train, model_lr_vt_prediction_train)
print("MAE of the Linear Regression Model (Variance Threshold) is:", round(lr_mae_vt,4)
```

```
print("MSE of the Linear Regression Model (Variance Threshold) is:", round(lr_mse_vt,4)
print("RMSE of the Linear Regression Model (Variance Threshold) is:", round(lr_rmse_vt,
print("R2 Score of the Linear Regression Model (Variance Threshold) on Test Data is:",
print("R2 Score of the Linear Regression Model (Variance Threshold) on Train Data is:",
```

```
MAE of the Linear Regression Model (Variance Threshold) is: 1095657.6968
MSE of the Linear Regression Model (Variance Threshold) is: 2042634081908.9043
RMSE of the Linear Regression Model (Variance Threshold) is: 1429207.5013
R2 Score of the Linear Regression Model (Variance Threshold) on Test Data is: 0.4101
R2 Score of the Linear Regression Model (Variance Threshold) on Train Data is: 0.3906
```

Decision Tree Model (Min-Max Scaler, No PCA or Variance Threshold Performed)

In [132…

```
'''
Create a Decision Tree Model.
Fit the model to the training data sets (without PCA or Variance Threshold Performed)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_dt = DecisionTreeRegressor()
model_dt.fit(X_train, y_train)
model_dt_prediction = model_dt.predict(X_test)
model_dt_prediction_train = model_dt.predict(X_train)
```

In [161…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
dt_mae = mean_absolute_error(y_test, model_dt_prediction)
dt_mse = mean_squared_error(y_test, model_dt_prediction)
dt_rmse = np.sqrt(dt_mse)
dt_r2 = r2_score(y_test, model_dt_prediction)
dt_r2_train = r2_score(y_train, model_dt_prediction_train)
print("MAE of the Decision Tree Model is:", round(dt_mae,4))
print("MSE of the Decision Tree Model is:", round(dt_mse,4))
print("RMSE of the Decision Tree Model is:", round(dt_rmse,4))
print("R2 Score of the Decision Tree Model on Test Data is:", round(dt_r2,4))
print("R2 Score of the Decision Tree Model on Train Data is:", round(dt_r2_train,4))
```

```
MAE of the Decision Tree Model is: 1059848.2075
MSE of the Decision Tree Model is: 1819252913899.057
RMSE of the Decision Tree Model is: 1348796.8394
R2 Score of the Decision Tree Model on Test Data is: 0.4746
R2 Score of the Decision Tree Model on Train Data is: 0.9997
```

Decision Tree Model (Standard Scaler, PCA)

In [134…

```
'''
Create a Decision Tree Model.
Fit the model to the training data sets (PCA)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_dt_pca = DecisionTreeRegressor()
model_dt_pca.fit(X_train_pca, y_train)
```

```
model_dt_pca_prediction = model_dt_pca.predict(X_test_pca)
model_dt_pca_prediction_train = model_dt_pca.predict(X_train_pca)
```

In [160…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
dt_mae_pca = mean_absolute_error(y_test, model_dt_pca_prediction)
dt_mse_pca = mean_squared_error(y_test, model_dt_pca_prediction)
dt_rmse_pca = np.sqrt(dt_mse_pca)
dt_r2_pca = r2_score(y_test, model_dt_pca_prediction)
dt_r2_pca_train = r2_score(y_train, model_dt_pca_prediction_train)
print("MAE of the Decision Tree Model (PCA) is:", round(dt_mae_pca,4))
print("MSE of the Decision Tree Model (PCA) is:", round(dt_mse_pca,4))
print("RMSE of the Decision Tree Model (PCA) is:", round(dt_rmse_pca,4))
print("R2 Score of the Decision Tree Model (PCA) on Test Data is:", round(dt_r2_pca,4))
print("R2 Score of the Decision Tree Model (PCA) on Train Data is:", round(dt_r2_pca_tr
```

```
MAE of the Decision Tree Model (PCA) is: 1164933.3962
MSE of the Decision Tree Model (PCA) is: 2659062630977.3584
RMSE of the Decision Tree Model (PCA) is: 1630663.2488
R2 Score of the Decision Tree Model (PCA) on Test Data is: 0.2321
R2 Score of the Decision Tree Model (PCA) on Train Data is: 0.9997
```

Decision Tree Model (Min-Max Scaler, Variance Threshold Performed)

In [136…

```
'''
Create a Decision Tree Model.
Fit the model to the training data sets (Variance Threshold)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_dt_vt = DecisionTreeRegressor()
model_dt_vt.fit(train_features_high_variance, y_train)
model_dt_vt_prediction = model_dt_vt.predict(test_features_high_variance)
model_dt_vt_prediction_train = model_dt_vt.predict(train_features_high_variance)
```

In [159…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
dt_mae_vt = mean_absolute_error(y_test, model_dt_vt_prediction)
dt_mse_vt = mean_squared_error(y_test, model_dt_vt_prediction)
dt_rmse_vt = np.sqrt(dt_mse_vt)
dt_r2_vt = r2_score(y_test, model_dt_vt_prediction)
dt_r2_vt_train = r2_score(y_train, model_dt_vt_prediction_train)
print("MAE of the Decision Tree Model (Variance Threshold) is:", round(dt_mae_vt,4))
print("MSE of the Decision Tree Model (Variance Threshold) is:", round(dt_mse_vt,4))
print("RMSE of the Decision Tree Model (Variance Threshold) is:", round(dt_rmse_vt,4))
print("R2 Score of the Decision Tree Model (Variance Threshold) on Test Data is:", roun
print("R2 Score of the Decision Tree Model (Variance Threshold) on Train Data is:", rou
```

```
MAE of the Decision Tree Model (Variance Threshold) is: 1138005.2365
MSE of the Decision Tree Model (Variance Threshold) is: 2190322280312.8955
RMSE of the Decision Tree Model (Variance Threshold) is: 1479973.7431
```

```
R2 Score of the Decision Tree Model (Variance Threshold) on Test Data is: 0.3674
R2 Score of the Decision Tree Model (Variance Threshold) on Train Data is: 0.4698
```

Random Forest Model (Min-Max Scaler, No PCA or Variance Threshold Performed)

In [139…

```
'''
Create a Random Forest Model.
Fit the model to the training data sets (without PCA or Variance Threshold Performed)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_rf = RandomForestRegressor()
model_rf.fit(X_train, y_train)
model_rf_prediction = model_rf.predict(X_test)
model_rf_prediction_train = model_rf.predict(X_train)
```

In [157…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
rf_mae = mean_absolute_error(y_test, model_rf_prediction)
rf_mse = mean_squared_error(y_test, model_rf_prediction)
rf_rmse = np.sqrt(rf_mse)
rf_r2 = r2_score(y_test, model_rf_prediction)
rf_r2_train = r2_score(y_train, model_rf_prediction_train)
print("MAE of the Random Forest Model is:", round(rf_mae,4))
print("MSE of the Random Forest Model is:", round(rf_mse,4))
print("RMSE of the Random Forest Model is:", round(rf_rmse,4))
print("R2 Score of the Random Forest Model on Test Data is:", round(rf_r2,4))
print("R2 Score of the Random Forest Model on Train Data is:", round(rf_r2_train,4))
```

```
MAE of the Random Forest Model is: 856667.4623
MSE of the Random Forest Model is: 1250280255817.8052
RMSE of the Random Forest Model is: 1118159.3159
R2 Score of the Random Forest Model on Test Data is: 0.6389
R2 Score of the Random Forest Model on Train Data is: 0.9469
```

Random Forest Model (Standard Scaler, PCA)

In [142…

```
'''
Create a Random Forest Model.
Fit the model to the training data sets (PCA)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_rf_pca = RandomForestRegressor()
model_rf_pca.fit(X_train_pca, y_train)
model_rf_pca_prediction = model_rf_pca.predict(X_test_pca)
model_rf_pca_prediction_train = model_rf_pca.predict(X_train_pca)
```

In [156…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
```

```
rf_mae_pca = mean_absolute_error(y_test, model_rf_pca_prediction)
rf_mse_pca = mean_squared_error(y_test, model_rf_pca_prediction)
rf_rmse_pca = np.sqrt(rf_mse_pca)
rf_r2_pca = r2_score(y_test, model_rf_pca_prediction)
rf_r2_pca_train = r2_score(y_train, model_rf_pca_prediction_train)
print("MAE of the Random Forest Model (PCA) is:", round(rf_mae_pca,4))
print("MSE of the Random Forest Model (PCA) is:", round(rf_mse_pca,4))
print("RMSE of the Random Forest Model (PCA) is:", round(rf_rmse_pca,4))
print("R2 Score of the Random Forest Model (PCA) on Test Data is:", round(rf_r2_pca,4))
print("R2 Score of the Random Forest Model (PCA) on Train Data is:", round(rf_r2_pca_tr
```

```
MAE of the Random Forest Model (PCA) is: 819063.4821
MSE of the Random Forest Model (PCA) is: 1114654252692.1802
RMSE of the Random Forest Model (PCA) is: 1055771.8753
R2 Score of the Random Forest Model (PCA) on Test Data is: 0.6781
R2 Score of the Random Forest Model (PCA) on Train Data is: 0.9436
```

Random Forest Model (Min-Max Scaler, Variance Threshold Performed)

In [144…

```
'''
Create a Random Forest Model.
Fit the model to the training data sets (Variance Threshold)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_rf_vt = RandomForestRegressor()
model_rf_vt.fit(train_features_high_variance, y_train)
model_rf_vt_prediction = model_rf_vt.predict(test_features_high_variance)
model_rf_vt_prediction_train = model_rf_vt.predict(train_features_high_variance)
```

In [155…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
rf_mae_vt = mean_absolute_error(y_test, model_rf_vt_prediction)
rf_mse_vt = mean_squared_error(y_test, model_rf_vt_prediction)
rf_rmse_vt = np.sqrt(rf_mse_vt)
rf_r2_vt = r2_score(y_test, model_rf_vt_prediction)
rf_r2_vt_train = r2_score(y_train, model_rf_vt_prediction_train)
print("MAE of the Random Forest Model (Variance Threshold) is:", round(rf_mae_vt,4))
print("MSE of the Random Forest Model (Variance Threshold) is:", round(rf_mse_vt,4))
print("RMSE of the Random Forest Model (Variance Threshold) is:", round(rf_rmse_vt,4))
print("R2 Score of the Random Forest Model (Variance Threshold) on Test Data is:", roun
print("R2 Score of the Random Forest Model (Variance Threshold) on Train Data is:", rou
```

```
MAE of the Random Forest Model (Variance Threshold) is: 1118317.4681
MSE of the Random Forest Model (Variance Threshold) is: 2054511430695.474
RMSE of the Random Forest Model (Variance Threshold) is: 1433356.7004
R2 Score of the Random Forest Model (Variance Threshold) on Test Data is: 0.4067
R2 Score of the Random Forest Model (Variance Threshold) on Train Data is: 0.4664
```

Lasso Regression Model (Min-Max Scaler, No PCA or Variance Threshold Performed)

In [146…

```
'''
```

```
Create a Lasso Regression Model.
Fit the model to the training data sets (without PCA or Variance Threshold Performed)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_lasso = Lasso()
model_lasso.fit(X_train, y_train)
model_lasso_prediction = model_lasso.predict(X_test)
model_lasso_prediction_train = model_lasso.predict(X_train)
```

In [154…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
lasso_mae = mean_absolute_error(y_test, model_lasso_prediction)
lasso_mse = mean_squared_error(y_test, model_lasso_prediction)
lasso_rmse = np.sqrt(lasso_mse)
lasso_r2 = r2_score(y_test, model_lasso_prediction)
lasso_r2_train = r2_score(y_train, model_lasso_prediction_train)
print("MAE of the Lasso Regression Model is:", round(lasso_mae,4))
print("MSE of the Lasso Regression Model is:", round(lasso_mse,4))
print("RMSE of the Lasso Regression Model is:", round(lasso_rmse,4))
print("R2 Score of the Lasso Regression Model on Test Data is:", round(lasso_r2,4))
print("R2 Score of the Lasso Regression Model on Train Data is:", round(lasso_r2_train,
```

```
MAE of the Lasso Regression Model is: 776999.7922
MSE of the Lasso Regression Model is: 1136825852703.4912
RMSE of the Lasso Regression Model is: 1066220.3584
R2 Score of the Lasso Regression Model on Test Data is: 0.6717
R2 Score of the Lasso Regression Model on Train Data is: 0.6457
```

Lasso Regression Model (Standard Scaler, PCA)

In [148…

```
'''
Create a Lasso Regression Model.
Fit the model to the training data sets (PCA)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_lasso_pca = Lasso()
model_lasso_pca.fit(X_train_pca, y_train)
model_lasso_pca_prediction = model_lasso_pca.predict(X_test_pca)
model_lasso_pca_prediction_train = model_lasso_pca.predict(X_train_pca)
```

In [153…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
lasso_mae_pca = mean_absolute_error(y_test, model_lasso_pca_prediction)
lasso_mse_pca = mean_squared_error(y_test, model_lasso_pca_prediction)
lasso_rmse_pca = np.sqrt(lasso_mse_pca)
lasso_r2_pca = r2_score(y_test, model_lasso_pca_prediction)
lasso_r2_pca_train = r2_score(y_train, model_lasso_pca_prediction_train)
print("MAE of the Lasso Regression Model (PCA) is:", round(lasso_mae_pca,4))
print("MSE of the Lasso Regression Model (PCA) is:", round(lasso_mse_pca,4))
print("RMSE of the Lasso Regression Model (PCA) is:", round(lasso_rmse_pca,4))
```

```
print("R2 Score of the Lasso Regression Model (PCA) on Test Data is:", round(lasso_r2_p
print("R2 Score of the Lasso Regression Model (PCA) on Train Data is:", round(lasso_r2_
```

```
MAE of the Lasso Regression Model (PCA) is: 798568.652
MSE of the Lasso Regression Model (PCA) is: 1190875024224.9265
RMSE of the Lasso Regression Model (PCA) is: 1091272.2045
R2 Score of the Lasso Regression Model (PCA) on Test Data is: 0.6561
R2 Score of the Lasso Regression Model (PCA) on Train Data is: 0.6396
```

Lasso Regression Model (Min-Max Scaler, Variance Threshold Performed)

In [150…

```
'''
Create a Lasso Regression Model.
Fit the model to the training data sets (Variance Threshold)
Create predictions to validate the model performance on "unseen data".
Create predictions to validate the model performance on trained data.
'''
model_lasso_vt = Lasso()
model_lasso_vt.fit(train_features_high_variance, y_train)
model_lasso_vt_prediction = model_lasso_vt.predict(test_features_high_variance)
model_lasso_vt_prediction_train = model_lasso_vt.predict(train_features_high_variance)
```

In [152…

```
'''
Calcualte the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Er
and R2 Score for the model on the train and test data.
'''
lasso_mae_vt = mean_absolute_error(y_test, model_lasso_vt_prediction)
lasso_mse_vt = mean_squared_error(y_test, model_lasso_vt_prediction)
lasso_rmse_vt = np.sqrt(lasso_mse_vt)
lasso_r2_vt = r2_score(y_test, model_lasso_vt_prediction)
lasso_r2_vt_train = r2_score(y_train, model_lasso_vt_prediction_train)
print("MAE of the Lasso Regression Model (Variance Threshold) is:", round(lasso_mae_vt,
print("MSE of the Lasso Regression Model (Variance Threshold) is:", round(lasso_mse_vt,
print("RMSE of the Lasso Regression Model (Variance Threshold) is:", round(lasso_rmse_v
print("R2 Score of the Lasso Regression Model (Variance Threshold) on Test Data is:", r
print("R2 Score of the Lasso Regression Model (Variance Threshold) on Test Data is:", r
```

```
MAE of the Lasso Regression Model (Variance Threshold) is: 1095657.921
MSE of the Lasso Regression Model (Variance Threshold) is: 2042634424321.0947
RMSE of the Lasso Regression Model (Variance Threshold) is: 1429207.6211
R2 Score of the Lasso Regression Model (Variance Threshold) on Test Data is: 0.4101
R2 Score of the Lasso Regression Model (Variance Threshold) on Test Data is: 0.3906
```

In [179…

```
'''
Create a dictionary summarizing the evaluation criteria for the four models.
'''
summary_data = OrderedDict([('PCA', ['No', 'Yes', 'No','No', 'Yes', 'No','No', 'Yes', '
                             ('Variance_Threshold', ['No', 'No', 'Yes', 'No', 'No', 'Yes'
                             ('MAE', [lr_mae, lr_mae_pca, lr_mae_vt,dt_mae,dt_mae_pca,dt_
                                    rf_mae,rf_mae_pca,rf_mae_vt,lasso_mae,lasso_mae_pca,l
                             ('MSE', [lr_mse, lr_mse_pca, lr_mse_vt,dt_mse,dt_mse_pca,dt_
                                    rf_mse,rf_mse_pca,rf_mse_vt,lasso_mse,lasso_mse_pca,l
                             ('RMSE', [lr_rmse, lr_rmse_pca, lr_rmse_vt,dt_rmse,dt_rmse_p
                                    rf_rmse,rf_rmse_pca,rf_rmse_vt,lasso_rmse,lasso_rmse_
                             ('R2_Score_Train', [lr_r2_train, lr_r2_pca_train, lr_r2_vt_t
```

```
                                        dt_r2_vt_train, rf_r2_train,rf_r2_pca_tr
                                        lasso_r2_train,lasso_r2_pca_train,lasso_
                        ('R2_Score_Test', [lr_r2, lr_r2_pca, lr_r2_vt,dt_r2,dt_r2_pc
                                        dt_r2_vt, rf_r2,rf_r2_pca,rf_r2_vt,
                                        lasso_r2,lasso_r2_pca,lasso_r2_vt]),
                        ('Model', ['LinearRegression', 'LinearRegression_PCA', 'Line
                                   'DecisionTree', 'DecisionTree_PCA', 'DecisionTree
                                   'RandomForest', 'RandomForest_PCA', 'RandomForest
                                   'Lasso', 'Lasso_PCA', 'Lasso_VT']),])
```

In [180…

```
'''
Display the summary evaluation metrics for the four models in a pandas dataframe.
'''
summary_df = pd.DataFrame(summary_data, index = summary_data['Model'])
display(summary_df)
```

	PCA	Variance_Threshold	MAE	MSE	RMSE	R2_Score_Tra
LinearRegression	No	No	7.769996e+05	1.136826e+12	1.066221e+06	0.6456
LinearRegression_PCA	Yes	No	7.985685e+05	1.190875e+12	1.091272e+06	0.6396
LinearRegression_VT	No	Yes	1.095658e+06	2.042634e+12	1.429208e+06	0.3905
DecisionTree	No	No	1.059848e+06	1.819253e+12	1.348797e+06	0.9997
DecisionTree_PCA	Yes	No	1.164933e+06	2.659063e+12	1.630663e+06	0.9997
DecisionTree_VT	No	Yes	1.138005e+06	2.190322e+12	1.479974e+06	0.4698
RandomForest	No	No	8.566675e+05	1.250280e+12	1.118159e+06	0.9468
RandomForest_PCA	Yes	No	8.190635e+05	1.114654e+12	1.055772e+06	0.9436
RandomForest_VT	No	Yes	1.118317e+06	2.054511e+12	1.433357e+06	0.4664
Lasso	No	No	7.769998e+05	1.136826e+12	1.066220e+06	0.6456
Lasso_PCA	Yes	No	7.985687e+05	1.190875e+12	1.091272e+06	0.6396
Lasso_VT	No	Yes	1.095658e+06	2.042634e+12	1.429208e+06	0.3905

In [181…

```
'''
Sort the values based on the highest R-Squared Score on the test data.
'''
summary_df.sort_values(by = "R2_Score_Test", ascending = False)
```

Out[181…

	PCA	Variance_Threshold	MAE	MSE	RMSE	R2_Score_Tra
RandomForest_PCA	Yes	No	8.190635e+05	1.114654e+12	1.055772e+06	0.9436
Lasso	No	No	7.769998e+05	1.136826e+12	1.066220e+06	0.6456
LinearRegression	No	No	7.769996e+05	1.136826e+12	1.066221e+06	0.6456
Lasso_PCA	Yes	No	7.985687e+05	1.190875e+12	1.091272e+06	0.6396
LinearRegression_PCA	Yes	No	7.985685e+05	1.190875e+12	1.091272e+06	0.6396

	PCA	Variance_Threshold	MAE	MSE	RMSE	R2_Score_Tra
RandomForest	No	No	8.566675e+05	1.250280e+12	1.118159e+06	0.9468
DecisionTree	No	No	1.059848e+06	1.819253e+12	1.348797e+06	0.9997
LinearRegression_VT	No	Yes	1.095658e+06	2.042634e+12	1.429208e+06	0.3905
Lasso_VT	No	Yes	1.095658e+06	2.042634e+12	1.429208e+06	0.3905
RandomForest_VT	No	Yes	1.118317e+06	2.054511e+12	1.433357e+06	0.4664
DecisionTree_VT	No	Yes	1.138005e+06	2.190322e+12	1.479974e+06	0.4698
DecisionTree_PCA	Yes	No	1.164933e+06	2.659063e+12	1.630663e+06	0.9997

In [213…

```
'''
Check the order of importance for the RandomForest_PCA Model based on the included feat
'''
feature_labels = ['area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'airconditioning', 'parking', 'prefarea',
       'furnishingstatus']
for feature in zip(feature_labels, model_rf_pca.feature_importances_):
    print(feature)
```

```
('area', 0.6760747438943454)
('bedrooms', 0.0785057433374234)
('bathrooms', 0.035554352793511365)
('stories', 0.029023381942248853)
('mainroad', 0.027150074458280486)
('guestroom', 0.02196132770246257)
('basement', 0.022758205758111466)
('airconditioning', 0.03335733510319122)
('parking', 0.04503791568155605)
('prefarea', 0.03057691932886914)
```

In [210…

```
'''
Find the order of importance for the Lasso Regression Model Features.
'''
importance = model_lasso.coef_

for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f'% (i,v))
```

```
Feature: 0, Score: 201.87485
Feature: 1, Score: 104138.82818
Feature: 2, Score: 834978.96980
Feature: 3, Score: 423531.98319
Feature: 4, Score: 184260.74112
Feature: 5, Score: 391929.16715
Feature: 6, Score: 370930.95073
Feature: 7, Score: 333408.75922
Feature: 8, Score: 701223.67829
Feature: 9, Score: 559444.12471
Feature: 10, Score: -77242.72043
Feature: 11, Score: -335958.86252
```

Milestone 3 Summary

● During the data preparation phase, the data was separated into three versions of training (80%) and test (20%) data sets. The first version contains the original data without any alterations. The second version of training data involved Principal Component Analysis (PCA) to reduce the features based on retaining 90% of the variance. This reduced the features down to 10. The third version of training data involved setting a Variance Threshold (above 0.1). The Variance Threshold reduced the features down to 7.

● The four machine learning model types evaluated in this section were Linear Regression, Decision Tree, Random Forest, and Lasso Regression. Each model was trained on the three different types of training data. The models were then evaluated based on R-Squared for test data set, R-Squared for training data set, Mean Absolute Error, Mean Square Error, and Root Mean Square Error. The summary of the metrics are shown in the above data frame.

● The best performance on the test data set was from the Random Forest Model with PCA performed. The R-Squared Value for this model turned out as 0.6781 when evaulated on the test data. Although this performed the best based on the R-Squared Value, this is still not an ideal model to use for predicting home prices. The R-Squared value indicates that there may be some additional features influencing the home price that are not included in this data set. In addition, this model shows signs of being overfit to the training data due to the high R-Squared Value on the trained data set. The most influential feature for this model is home area.

● The next two models that performed the best were the Lasso Regression and Linear Regression models trained without any PCA or High Variance Threshold. Both of these models performed similarly with R-Squared Values of 0.6712. The R-Squared Values were similar on the trained data set for both models as well. These models did not have the same overfit problem as the Random Forest Model.

● Recommendations for future improvements for this project would be to perform hyperparameter tuning on the models to see if any additional improvements can be made for the model performance. In addition, the code can be consolidated through the use of Pipelines and or functions to improve readability. Cross-validation may be a good idea to include in the analysis to evaluate the selected model's performance. Lastly, additional data such as more samples or new features may help understand some of the unexplained variance left out of the data.