# Final Project-3

August 11, 2025

[218]:
```
"""
Created on Sun Jul 27 20:28:42 2025

@author: James Shoenhair
"""
```

[218]: '\nCreated on Sun Jul 27 20:28:42 2025\n\n@author: James Shoenhair\n'

[219]:
```
#Import House Sales Data

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from datetime import datetime

myseries = pd.read_csv('house_sales.csv')

print(myseries)

#Find Missing Data Values

missing_summary = myseries.isnull().sum()
print("Missing values per column:")
for col, count in missing_summary.items():
    if count > 0:
        print(f"{col}: {count} ({count/len(myseries)*100}%)")

#Missing Value Per Room Feature

myseries['bedrooms'] = myseries['bedrooms'].fillna(myseries['bedrooms'].
 ↪median())
myseries['bathrooms'] = myseries['bathrooms'].fillna(myseries['bathrooms'].
 ↪median())
```

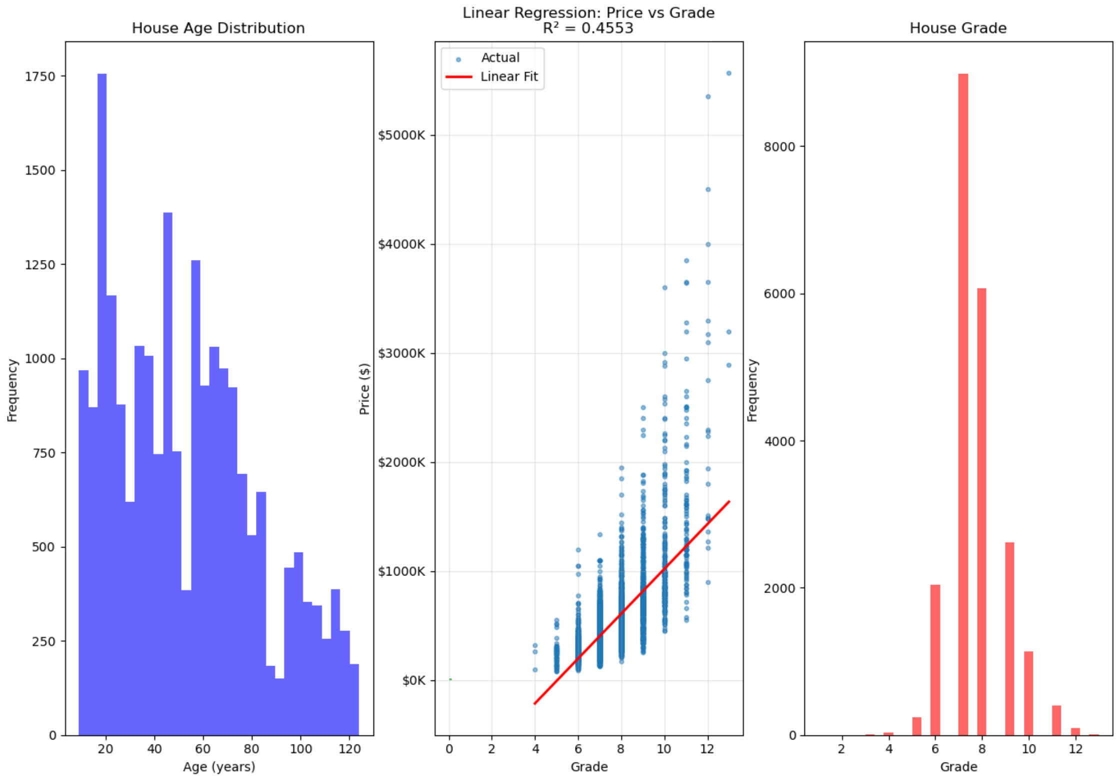
```python
myseries['sqft_living'] = myseries['sqft_living'].
    ↪fillna(myseries['sqft_living'].median())
myseries['sqft_lot'] = myseries['sqft_lot'].fillna(myseries['sqft_lot'].
    ↪median())

print(f"Missing values after cleaning: {myseries.isnull().sum().sum()}")

#Data Type Conversions

myseries['date'] = pd.to_datetime(myseries['date'], format='%Y%m%dT%H%M%S')


#Outliers For Time and Location Features

outlier_columns = ['year_built', 'year_renovated', 'grade', 'view',␣
    ↪'waterfront']

#House Age and condition

myseries['price_per_sqft'] = myseries['price'] / myseries['sqft_living']
myseries['house_age'] = 2024 - myseries['yr_built']
myseries['years_since_renovation'] = np.where(myseries['yr_renovated'] == 0,
                                       myseries['house_age'],
                                       2024 - myseries['yr_renovated'])

myseries['is_renovated'] = (myseries['yr_renovated'] > 0).astype(int)
myseries['basement_ratio'] = myseries['sqft_basement'] / myseries['sqft_living']


#Visualizations

plt.style.use('default')
fig, axes = plt.subplots(1, 3, figsize=(15, 10))

axes[0].hist(myseries['house_age'], bins=30, alpha=0.6, color='blue')
axes[0].set_title('House Age Distribution')
axes[0].set_xlabel('Age (years)')
axes[0].set_ylabel('Frequency')

axes[1].hist(myseries['view'], bins=30, alpha=0.6, color='green')
axes[1].set_title('House View')
axes[1].set_xlabel('View Score')
axes[1].set_ylabel('Frequency')

axes[2].hist(myseries['grade'], bins=30, alpha=0.6, color='red')
axes[2].set_title('House Grade')
axes[2].set_xlabel('Grade')
```

```
axes[2].set_ylabel('Frequency')


#Initial Linear Regression Model

grade = myseries[['grade']].copy()
price = myseries['price'].copy()

# def train_grade_model(grade_values, price_values):
grade_train, grade_test, price_train, price_test = train_test_split(grade,␣
 ↪price, test_size=0.2, random_state=42)


linear_model = LinearRegression()
linear_model.fit(grade_train, price_train)

price_train_pred_linear = linear_model.predict(grade_train)
price_test_pred_linear = linear_model.predict(grade_test)


# R2 Score, Squared Error, and Absolute Error Metrics

train_r2_linear = r2_score(price_train, price_train_pred_linear)
test_r2_linear = r2_score(price_test, price_test_pred_linear)
train_rmse_linear = np.sqrt(mean_squared_error(price_train,␣
 ↪price_train_pred_linear))
test_rmse_linear = np.sqrt(mean_squared_error(price_test,␣
 ↪price_test_pred_linear))
train_mae_linear = mean_absolute_error(price_train, price_train_pred_linear)
test_mae_linear = mean_absolute_error(price_test, price_test_pred_linear)

#Linear Regression Equation

slope = linear_model.coef_[0]
intercept = linear_model.intercept_
print(f"\nLinear Regression Equation:")
print(f"Price = {intercept:,.0f} + {slope:,.0f} × Grade")
print(f"For each grade increase, price increases by ${slope:,.0f}")


#Linear Regression Scatterplot

axes[1].scatter(grade_test, price_test, alpha=0.5, s=10, label='Actual')
axes[1].plot(grade_test.sort_values('grade'),
                linear_model.predict(grade_test.sort_values('grade')),
                'r-', linewidth=2, label='Linear Fit')
axes[1].set_xlabel('Grade')
```

```python
axes[1].set_ylabel('Price ($)')
axes[1].set_title(f'Linear Regression: Price vs Grade\nR² = {test_r2_linear:.4f}')
axes[1].legend()
axes[1].grid(True, alpha=0.3)
axes[1].yaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f'${x/1000:.0f}K'))
```

```
               id             date     price  bedrooms  bathrooms  \
0      7129300520  20141013T000000  221900.0       3.0       1.00
1      6414100192  20141209T000000  538000.0       3.0       2.25
2      5631500400  20150225T000000  180000.0       2.0       1.00
3      2487200875  20141209T000000  604000.0       4.0       3.00
4      1954400510  20150218T000000  510000.0       3.0       2.00
…             …                …        …        …         …
21608   263000018  20140521T000000  360000.0       3.0       2.50
21609  6600060120  20150223T000000  400000.0       4.0       2.50
21610  1523300141  20140623T000000  402101.0       2.0       0.75
21611   291310100  20150116T000000  400000.0       3.0       2.50
21612  1523300157  20141015T000000  325000.0       2.0       0.75

       sqft_living  sqft_lot  floors  waterfront  view  …  grade  \
0           1180.0    5650.0     1.0           0     0  …      7
1           2570.0    7242.0     2.0           0     0  …      7
2            770.0   10000.0     1.0           0     0  …      6
3           1960.0    5000.0     1.0           0     0  …      7
4           1680.0    8080.0     1.0           0     0  …      8
…              …        …      …          …    …  …    …
21608       1530.0    1131.0     3.0           0     0  …      8
21609       2310.0    5813.0     2.0           0     0  …      8
21610       1020.0    1350.0     2.0           0     0  …      7
21611       1600.0    2388.0     2.0           0     0  …      8
21612       1020.0    1076.0     2.0           0     0  …      7

       sqft_above  sqft_basement  yr_built  yr_renovated  zipcode      lat  \
0            1180              0      1955             0    98178  47.5112
1            2170            400      1951          1991    98125  47.7210
2             770              0      1933             0    98028  47.7379
3            1050            910      1965             0    98136  47.5208
4            1680              0      1987             0    98074  47.6168
…             …            …       …           …      …      …
21608        1530              0      2009             0    98103  47.6993
21609        2310              0      2014             0    98146  47.5107
21610        1020              0      2009             0    98144  47.5944
21611        1600              0      2004             0    98027  47.5345
21612        1020              0      2008             0    98144  47.5941

          long  sqft_living15  sqft_lot15
```

```
0      -122.257          1340         5650
1      -122.319          1690         7639
2      -122.233          2720         8062
3      -122.393          1360         5000
4      -122.045          1800         7503
...       ...        ...          ...
21608 -122.346          1530         1509
21609 -122.362          1830         7200
21610 -122.299          1020         2007
21611 -122.069          1410         1287
21612 -122.299          1020         1357

[21613 rows x 21 columns]
Missing values per column:
bedrooms: 1134 (5.246842178318604%)
bathrooms: 1068 (4.941470411326517%)
sqft_living: 1110 (5.135797899412391%)
sqft_lot: 1044 (4.830426132420302%)
Missing values after cleaning: 0

Linear Regression Equation:
Price = -1,034,439 + 205,414 × Grade
For each grade increase, price increases by $205,414
```



[220]:
```
"""
Created on Sun Jul 27 20:28:42 2025

@author: BereketTarekegn
"""
```

[220]: '\nCreated on Sun Jul 27 20:28:42 2025\n\n@author: BereketTarekegn\n'

[221]:
```
# Step 1: Import basic libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

[222]:
```
# Show plots in Spyder's plot pane
%matplotlib inline
```

[223]:
```
# Step 2: upload file
#os.chdir(r"C:\Users\bezaa\OneDrive\Desktop\USD\Data Science Programming␣
 ↪(ADS-500B-01\Group Assignment\My Project\Dataset 2 (House Sales)")
```

[224]:
```
# Step 2.1: Load the dataset
df = pd.read_csv("house_sales.csv")
```

[225]:
```
# Step 2.2: Show the first few rows
print(df.head())
```

```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0       3.0       1.00       1180.0
1  6414100192  20141209T000000  538000.0       3.0       2.25       2570.0
2  5631500400  20150225T000000  180000.0       2.0       1.00        770.0
3  2487200875  20141209T000000  604000.0       4.0       3.00       1960.0
4  1954400510  20150218T000000  510000.0       3.0       2.00       1680.0

   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0    5650.0     1.0           0     0  …      7        1180              0
1    7242.0     2.0           0     0  …      7        2170            400
2   10000.0     1.0           0     0  …      6         770              0
3    5000.0     1.0           0     0  …      7        1050            910
4    8080.0     1.0           0     0  …      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
```

```
3      1965             0     98136  47.5208 -122.393           1360
4      1987             0     98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

[226]:
```python
# Step 3: location-based features and price
location_df = df[['zipcode', 'lat', 'long', 'price']]
```

[227]:
```python
# See data types and if anything is missing
print(location_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   zipcode  21613 non-null  int64
 1   lat      21613 non-null  float64
 2   long     21613 non-null  float64
 3   price    21613 non-null  float64
dtypes: float64(3), int64(1)
memory usage: 675.5 KB
None
```

[228]:
```python
# Basic statistics
print(location_df.describe())
```

```
            zipcode           lat          long         price
count  21613.000000  21613.000000  21613.000000  2.161300e+04
mean   98077.939805     47.560053   -122.213896  5.400881e+05
std       53.505026      0.138564      0.140828  3.671272e+05
min    98001.000000     47.155900   -122.519000  7.500000e+04
25%    98033.000000     47.471000   -122.328000  3.219500e+05
50%    98065.000000     47.571800   -122.230000  4.500000e+05
75%    98118.000000     47.678000   -122.125000  6.450000e+05
max    98199.000000     47.777600   -121.315000  7.700000e+06
```

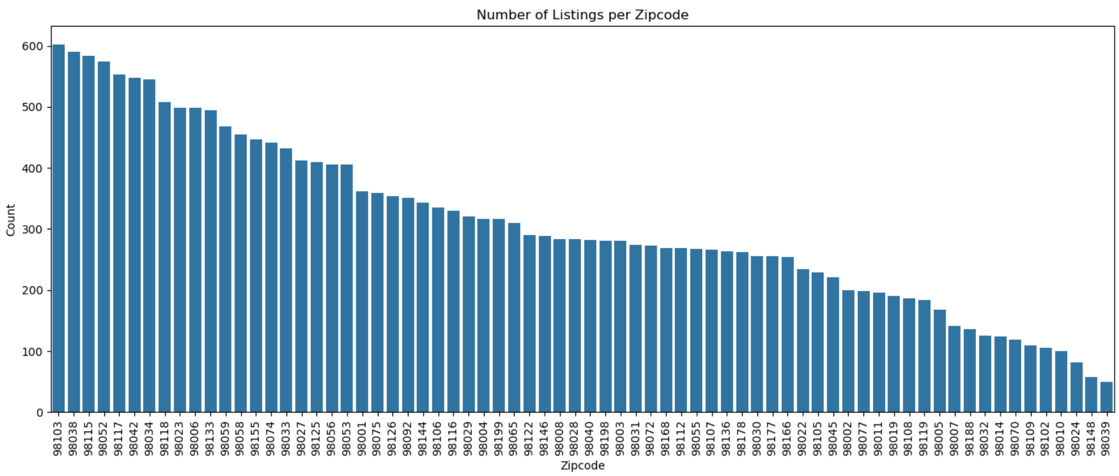
[229]:
```python
# Count missing values
print(location_df.isnull().sum())
```

```
zipcode    0
lat        0
```

```
long       0
price      0
dtype: int64
```

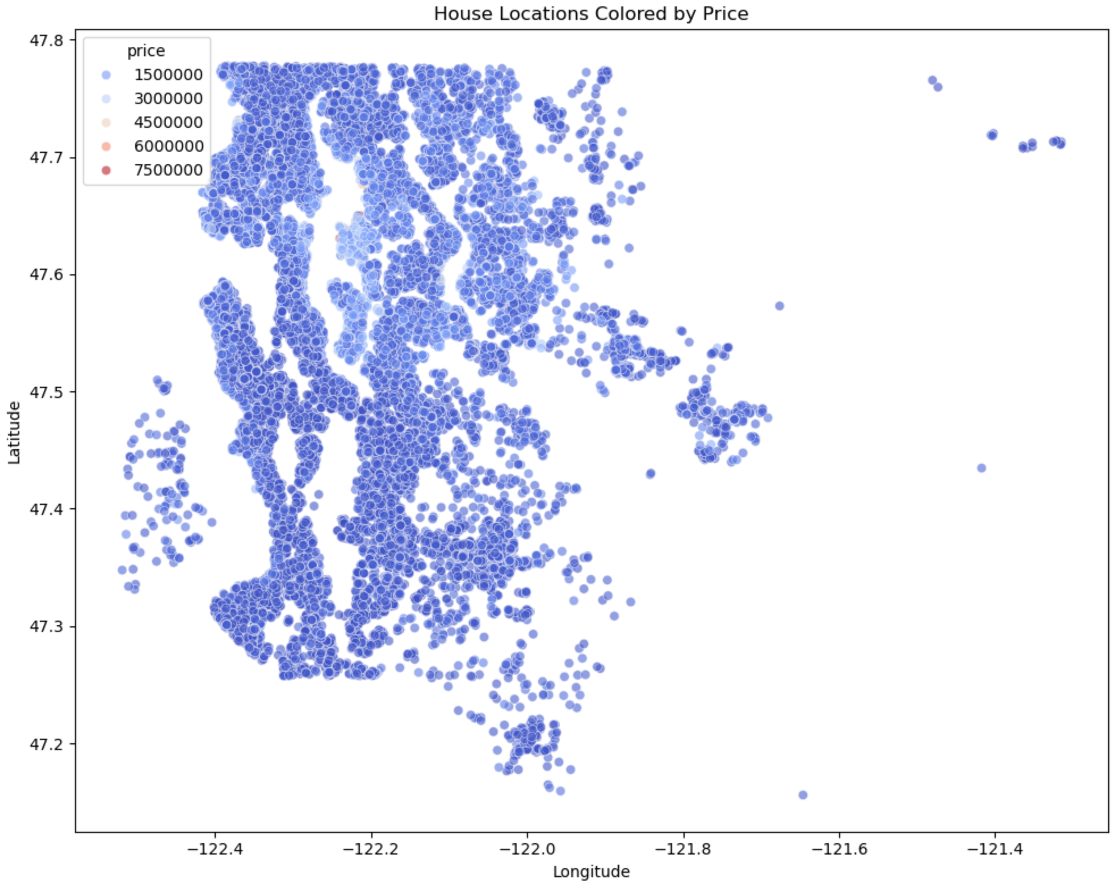
[230]:

```
#4.1. Listings per Zipcode
plt.figure(figsize=(14, 6))
sns.countplot(data=df, x='zipcode', order=df['zipcode'].value_counts().index)
plt.xticks(rotation=90)
plt.title("Number of Listings per Zipcode")
plt.xlabel("Zipcode")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```




[231]:

```
#4.2. Map of House Locations Colored by Price
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df, x='long', y='lat', hue='price', palette='coolwarm',␣
 ↪alpha=0.6)
plt.title("House Locations Colored by Price")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.tight_layout()
plt.show()
```




[232]:
```python
#4.3. Correlation Between Location and Price
# Correlation matrix for lat, long, and price
print(location_df.corr())
```

```
          zipcode       lat      long     price
zipcode  1.000000  0.267048 -0.564072 -0.053203
lat      0.267048  1.000000 -0.135512  0.307003
long    -0.564072 -0.135512  1.000000  0.021626
price   -0.053203  0.307003  0.021626  1.000000
```

[233]:
```python
#Step 5: Regression with Location Features
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

[234]:
```python
# Use only lat and long to predict price
X = df[['lat', 'long']]
y = df['price']
```

[235]:
```
# Split the data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[236]:
```
# Create and train model
model = LinearRegression()
model.fit(X_train, y_train)
```

[236]:
```
LinearRegression()
```

[237]:
```
# Check R² score (how well location explains price)
r2_score = model.score(X_test, y_test)
print(f"R² Score using lat/long to predict price: {r2_score:.4f}")
```

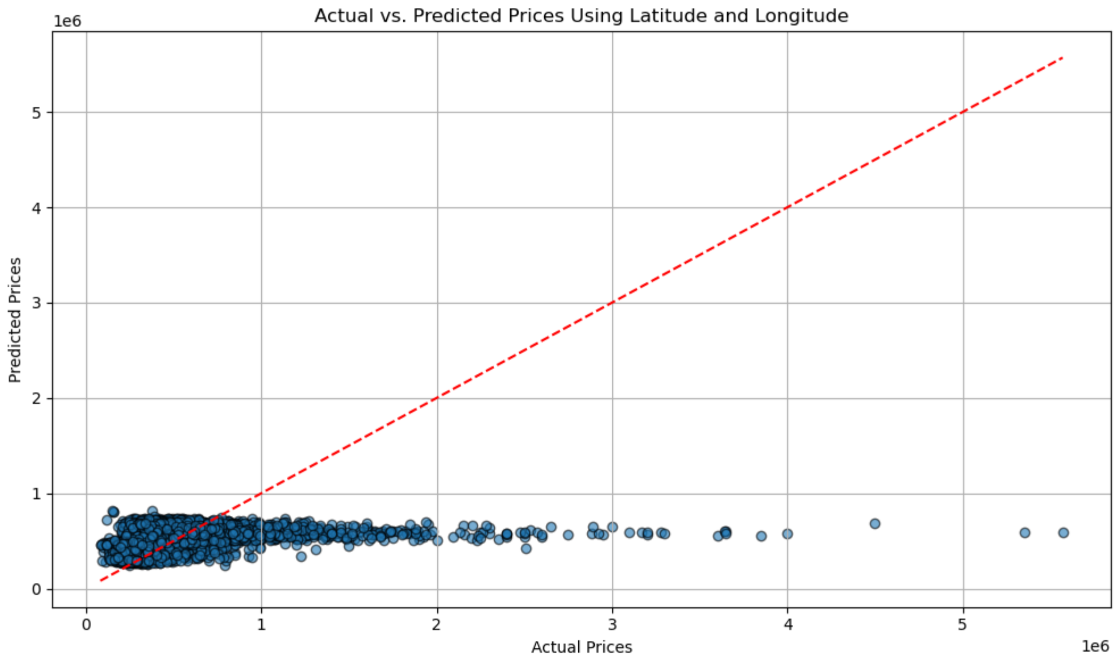
```
R² Score using lat/long to predict price: 0.0881
```

[238]:
```
# Step 6: Evaluate the model with Cross-Validation and MAE
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_absolute_error

# Predict on test set
y_pred = model.predict(X_test)

# Mean Absolute Error on test set
mae = mean_absolute_error(y_test, y_pred)
print(f"\nMean Absolute Error (MAE) on Test Set: {mae:.2f}")

# Cross-validation setup (5-fold)
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# R² from cross-validation
cv_r2_scores = cross_val_score(model, X, y, cv=kf, scoring='r2')
print(f"Cross-Validated R² Scores: {cv_r2_scores}")
print(f"Average Cross-Validated R²: {cv_r2_scores.mean():.4f}")

# MAE from cross-validation (using negative MAE scoring, so we take negative of␣
↪result)
cv_mae_scores = -cross_val_score(model, X, y, cv=kf,␣
↪scoring='neg_mean_absolute_error')
print(f"Cross-Validated MAE Scores: {cv_mae_scores}")
print(f"Average Cross-Validated MAE: {cv_mae_scores.mean():.2f}")
```

```
Mean Absolute Error (MAE) on Test Set: 216870.57
Cross-Validated R² Scores: [0.08810933 0.09080477 0.09857414 0.10250905
0.11410444]
Average Cross-Validated R²: 0.0988
Cross-Validated MAE Scores: [216870.5744618  212711.68503546 204549.55253278
```

```
209304.68014327
 208154.24531562]
Average Cross-Validated MAE: 210318.15
```

[239]:
```python
# Step 7: Visualization - Actual vs. Predicted Prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.6, edgecolors='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  #␣
 ↪Perfect prediction line
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs. Predicted Prices Using Latitude and Longitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```




[240]:
```python
print("\nInterpretation:")
print("The scatter plot above compares the actual house prices to the prices␣
 ↪predicted by the regression model")
print("using only latitude and longitude features. Ideally, data points should␣
 ↪align closely along the red dashed")
print("line, which represents perfect prediction (i.e., predicted = actual).␣
 ↪However, we observe that most predictions")
print("cluster around lower price values regardless of the actual prices,␣
 ↪especially for higher-priced homes.")
```

```
print("This pattern highlights the model’s inability to capture variability in␣
  ↪housing prices using location data alone,")
print("confirming the low R² score of approximately 0.0881. This reinforces the␣
  ↪conclusion that latitude and longitude")
print("alone are weak predictors of house prices in this dataset.")
```

```
Interpretation:
The scatter plot above compares the actual house prices to the prices predicted
by the regression model
using only latitude and longitude features. Ideally, data points should align
closely along the red dashed
line, which represents perfect prediction (i.e., predicted = actual). However,
we observe that most predictions
cluster around lower price values regardless of the actual prices, especially
for higher-priced homes.
This pattern highlights the model’s inability to capture variability in housing
prices using location data alone,
confirming the low R² score of approximately 0.0881. This reinforces the
conclusion that latitude and longitude
alone are weak predictors of house prices in this dataset.
```

[241]:
```
"""
Structure-Related Features
@author: Jameel Saccoh
"""
```

[241]: '\nStructure-Related Features\n@author: Jameel Saccoh\n'

[7]:
```
#install libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from statsmodels.api import OLS
import statsmodels.api as sm
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import absolute
```

[8]:
```
# Load and preview housing dataset
housing = pd.read_csv('house_sales.csv')
housing.head()
```

[8]:
```
           id             date     price  bedrooms  bathrooms  sqft_living  \
0  7129300520  20141013T000000  221900.0       3.0       1.00       1180.0
1  6414100192  20141209T000000  538000.0       3.0       2.25       2570.0
2  5631500400  20150225T000000  180000.0       2.0       1.00        770.0
3  2487200875  20141209T000000  604000.0       4.0       3.00       1960.0
4  1954400510  20150218T000000  510000.0       3.0       2.00       1680.0

   sqft_lot  floors  waterfront  view  …  grade  sqft_above  sqft_basement  \
0    5650.0     1.0           0     0  …      7        1180              0
1    7242.0     2.0           0     0  …      7        2170            400
2   10000.0     1.0           0     0  …      6         770              0
3    5000.0     1.0           0     0  …      7        1050            910
4    8080.0     1.0           0     0  …      8        1680              0

   yr_built  yr_renovated  zipcode      lat     long  sqft_living15  \
0      1955             0    98178  47.5112 -122.257           1340
1      1951          1991    98125  47.7210 -122.319           1690
2      1933             0    98028  47.7379 -122.233           2720
3      1965             0    98136  47.5208 -122.393           1360
4      1987             0    98074  47.6168 -122.045           1800

   sqft_lot15
0        5650
1        7639
2        8062
3        5000
4        7503

[5 rows x 21 columns]
```

[9]:
```python
#Find the number of records within the dataset
print("Number of records:", housing.shape[0],"\n")

#Find all the null values within the dataset
print(housing.isna().sum())
```

```
Number of records: 21613

id                  0
date                0
price               0
bedrooms         1134
bathrooms        1068
sqft_living      1110
sqft_lot         1044
floors              0
waterfront          0
view                0
```

```
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated       0
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

[10]:

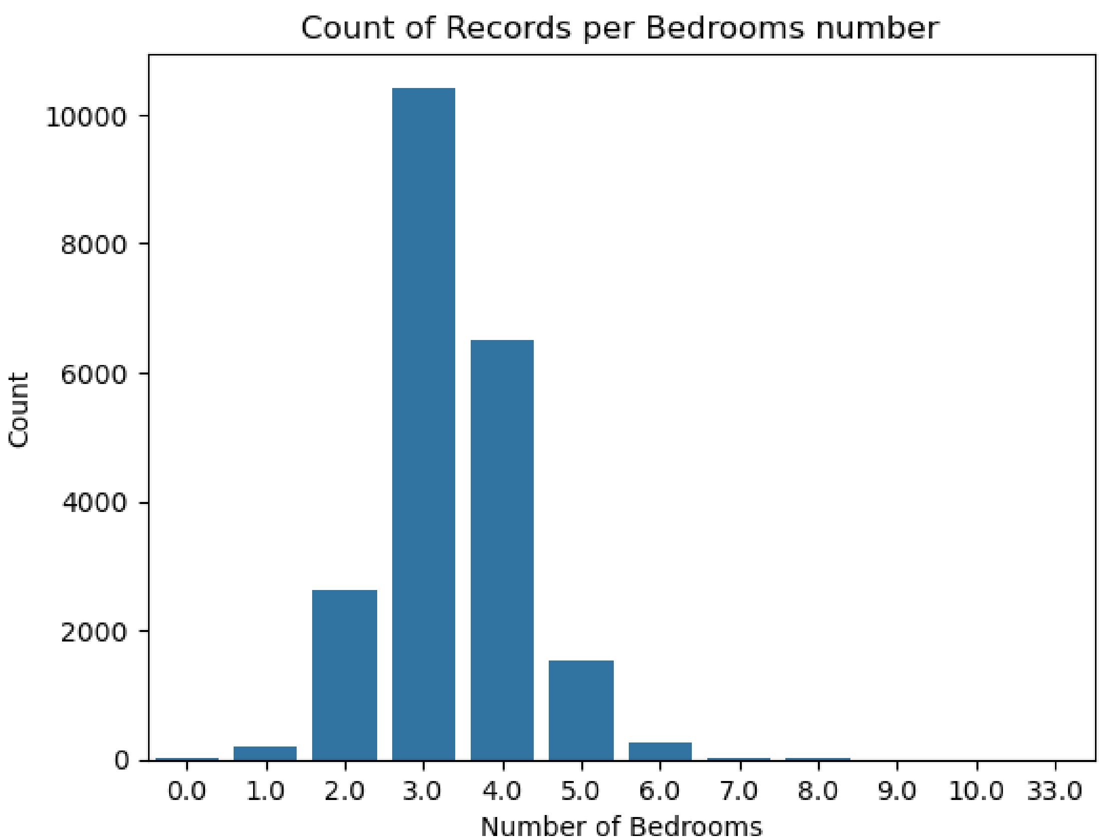
```
#Handle missing values for bedroom and bathroom columns with median imputation
housing['bedrooms'].fillna(housing['bedrooms'].median(), inplace=True)
housing['bathrooms'].fillna(housing['bathrooms'].median(), inplace=True)

#Handle missing values for sqft_living and sqft_lot columns with mean␣
↪imputation rounded to the nearest integer
housing['sqft_living'].fillna(round(housing['sqft_living'].mean(),0),␣
↪inplace=True)
housing['sqft_lot'].fillna(round(housing['sqft_lot'].mean(),0), inplace=True)

#Check for any remaining null values
print(housing.isna().sum())
```

```
id                0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
view              0
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated      0
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

```
C:\Users\jamee\AppData\Local\Temp\ipykernel_31660\3364273193.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  housing['bedrooms'].fillna(housing['bedrooms'].median(), inplace=True)
C:\Users\jamee\AppData\Local\Temp\ipykernel_31660\3364273193.py:3:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  housing['bathrooms'].fillna(housing['bathrooms'].median(), inplace=True)
C:\Users\jamee\AppData\Local\Temp\ipykernel_31660\3364273193.py:6:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  housing['sqft_living'].fillna(round(housing['sqft_living'].mean(),0),
inplace=True)
C:\Users\jamee\AppData\Local\Temp\ipykernel_31660\3364273193.py:7:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
```

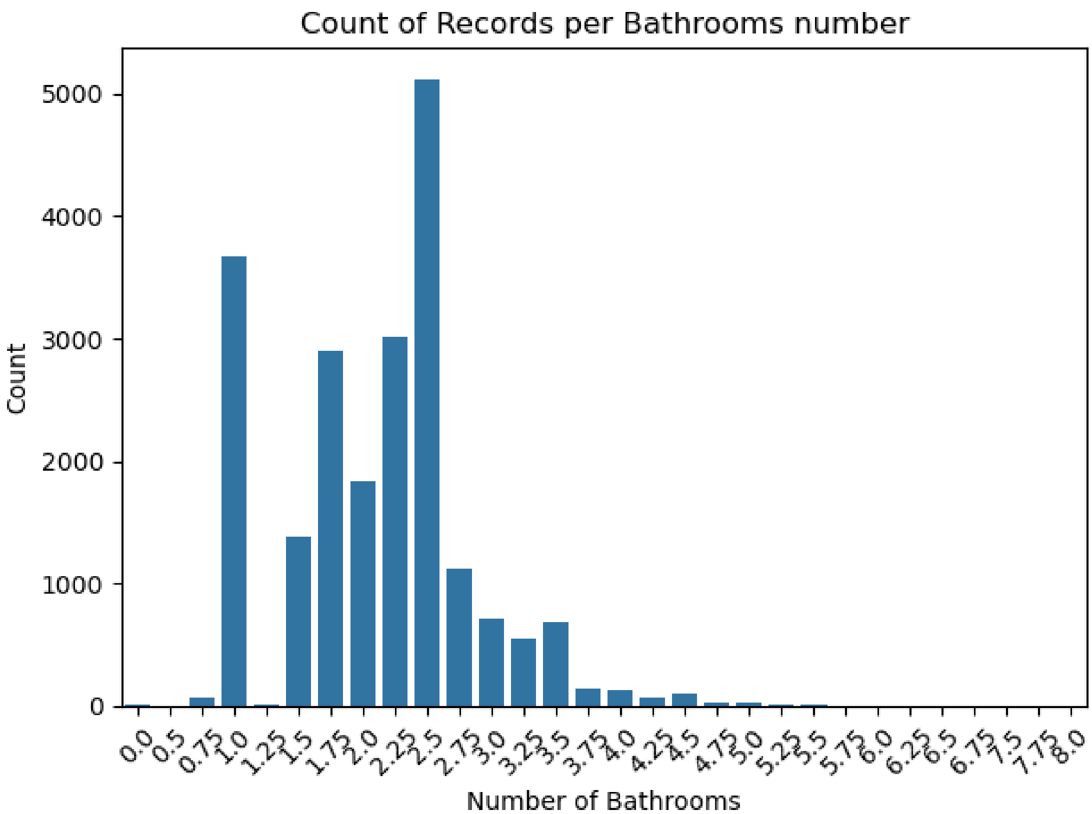
```
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  housing['sqft_lot'].fillna(round(housing['sqft_lot'].mean(),0), inplace=True)
```

[11]:
```python
#Plot count of records per number of bedrooms
sns.countplot(x='bedrooms', data=housing)
plt.title('Count of Records per Bedrooms number')
plt.xlabel('Number of Bedrooms')
plt.ylabel('Count')
plt.show()
```

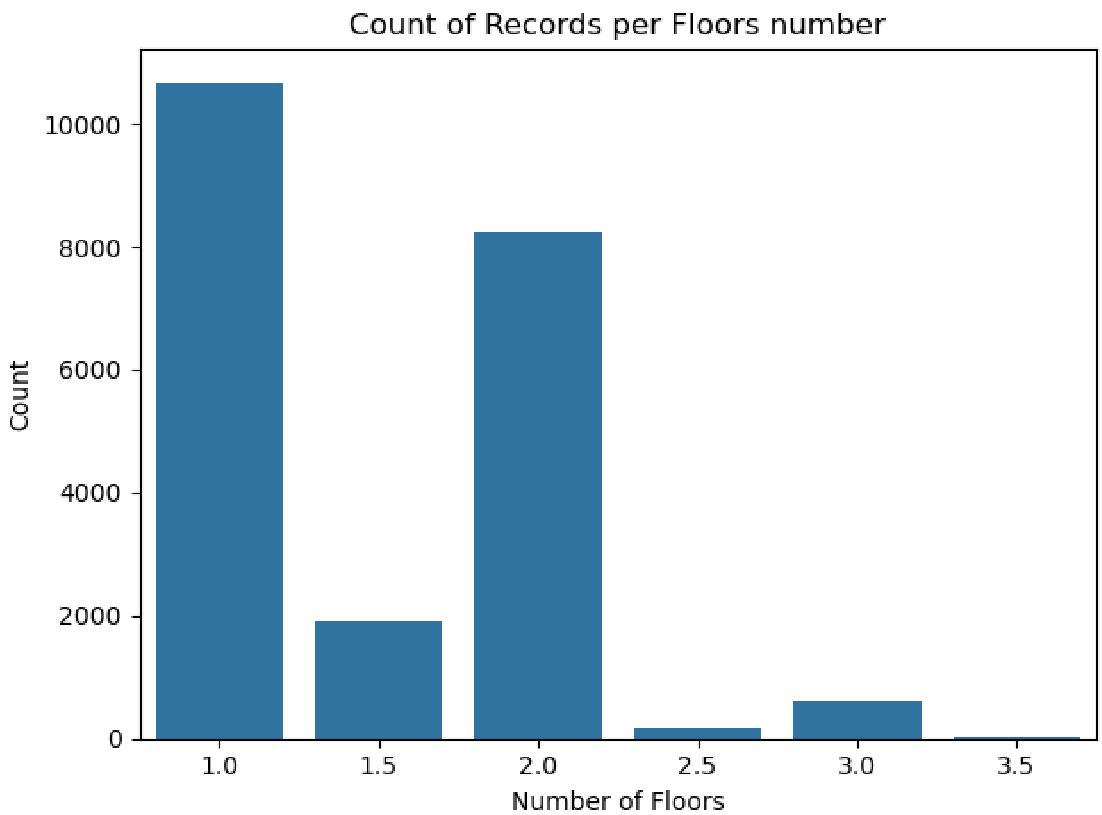



[12]:
```python
#Plot count of records per number of bathrooms
sns.countplot(x='bathrooms', data=housing)
plt.title('Count of Records per Bathrooms number')
plt.xlabel('Number of Bathrooms')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
```

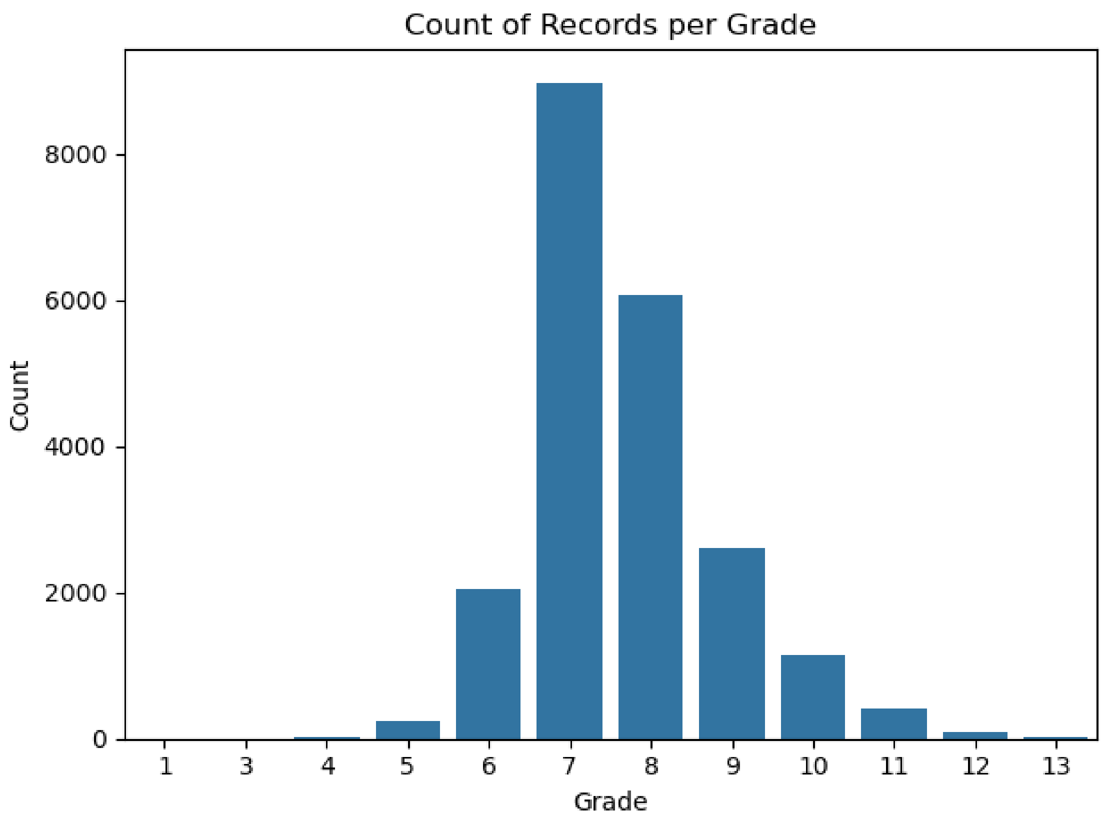
```
plt.show()
```




[13]:
```
#Plot count of records per number of floors
sns.countplot(x='floors', data=housing)
plt.title('Count of Records per Floors number')
plt.xlabel('Number of Floors')
plt.ylabel('Count')
plt.tight_layout()
```




[14]:
```
#Plot count of records per grade
sns.countplot(x='grade', data=housing)
plt.title('Count of Records per Grade')
plt.xlabel('Grade')
plt.ylabel('Count')
plt.tight_layout()
```




[15]:
```python
#Create subset of data with price, bedrooms, bathrooms, sqft_living, sqft_lot,␣
↪floors, and grade
struct_df = housing[['price', 'bedrooms', 'bathrooms', 'sqft_living',␣
↪'sqft_lot', 'floors', 'grade','sqft_above','sqft_basement']]

#Calculate correlation matrix
correlation_matrix = struct_df.corr()

#print correlation matrix
print("Correlation Matrix:\n", correlation_matrix)
```

```
Correlation Matrix:
                   price  bedrooms  bathrooms  sqft_living  sqft_lot    floors  \
price          1.000000  0.302493   0.515365     0.681806  0.086295  0.256794
bedrooms       0.302493  1.000000   0.487280     0.549012  0.027169  0.172168
bathrooms      0.515365  0.487280   1.000000     0.714817  0.083221  0.487859
sqft_living    0.681806  0.549012   0.714817     1.000000  0.160199  0.345740
sqft_lot       0.086295  0.027169   0.083221     0.160199  1.000000 -0.005540
floors         0.256794  0.172168   0.487859     0.345740 -0.005540  1.000000
grade          0.667434  0.348556   0.648745     0.744523  0.109002  0.458183
```

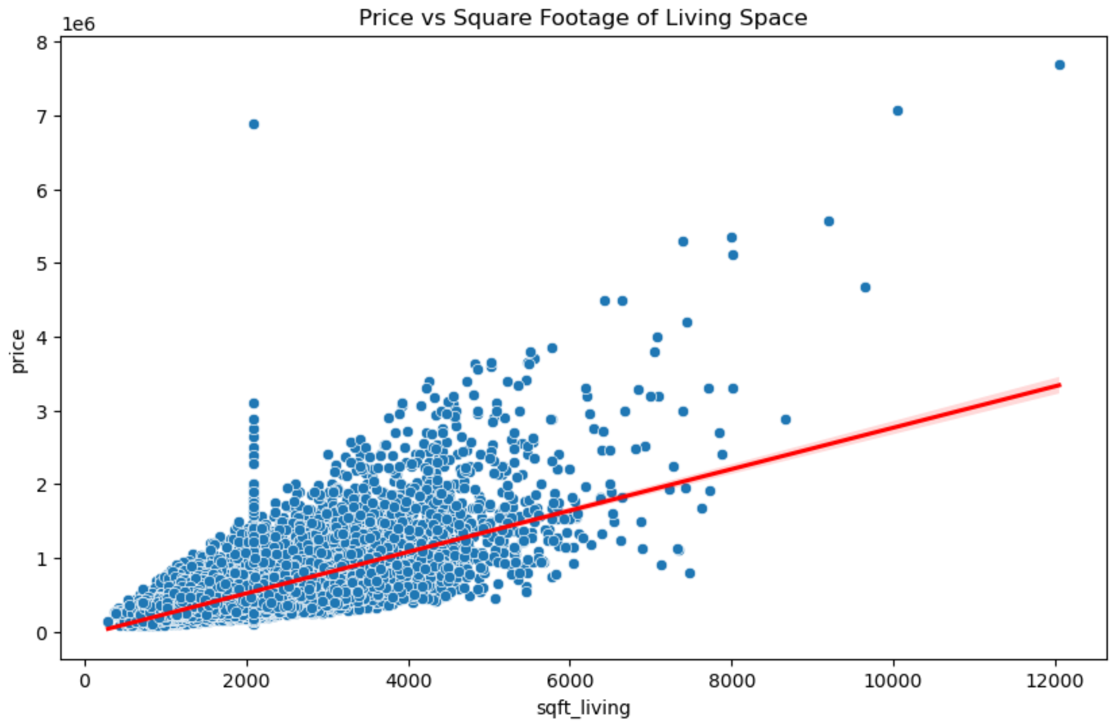
```
sqft_above     0.605567  0.465769   0.667757     0.851347  0.176906  0.523885
sqft_basement  0.323816  0.291689   0.276989     0.420816  0.015212 -0.245705

                  grade  sqft_above  sqft_basement
price          0.667434    0.605567       0.323816
bedrooms       0.348556    0.465769       0.291689
bathrooms      0.648745    0.667757       0.276989
sqft_living    0.744523    0.851347       0.420816
sqft_lot       0.109002    0.176906       0.015212
floors         0.458183    0.523885      -0.245705
grade          1.000000    0.755923       0.168392
sqft_above     0.755923    1.000000      -0.051943
sqft_basement  0.168392   -0.051943       1.000000
```

[16]:
```
struct_df.describe()
```

[16]:
```
              price     bedrooms     bathrooms   sqft_living      sqft_lot  \
count  2.161300e+04  21613.00000  21613.000000  21613.000000  2.161300e+04
mean   5.400881e+05      3.35326      2.120252   2081.069912  1.517983e+04
std    3.671272e+05      0.90977      0.750257    891.234976  4.047174e+04
min    7.500000e+04      0.00000      0.000000    290.000000  5.200000e+02
25%    3.219500e+05      3.00000      1.750000   1450.000000  5.140000e+03
50%    4.500000e+05      3.00000      2.250000   1980.000000  7.830000e+03
75%    6.450000e+05      4.00000      2.500000   2510.000000  1.186700e+04
max    7.700000e+06     33.00000      8.000000  12050.000000  1.651359e+06

             floors         grade    sqft_above  sqft_basement
count  21613.000000  21613.000000  21613.000000   21613.000000
mean       1.494309      7.656873   1788.390691     291.509045
std        0.539989      1.175459    828.090978     442.575043
min        1.000000      1.000000    290.000000       0.000000
25%        1.000000      7.000000   1190.000000       0.000000
50%        1.500000      7.000000   1560.000000       0.000000
75%        2.000000      8.000000   2210.000000     560.000000
max        3.500000     13.000000   9410.000000    4820.000000
```

[17]:
```
#Plot price vs sqft_living with line of best fit
plt.figure(figsize=(10, 6))
sns.scatterplot(x='sqft_living', y='price', data=housing)
plt.title('Price vs Square Footage of Living Space')
plt.xlabel('Square Footage of Living Space')
plt.ylabel('Price')
sns.regplot(x='sqft_living', y='price', data=housing, scatter=False,␣
 ↪color='red')
```

[17]:
```
<Axes: title={'center': 'Price vs Square Footage of Living Space'},
xlabel='sqft_living', ylabel='price'>
```




[18]:
```
#Split the data between training and testing sets
X = struct_df.drop('price', axis=1)
y = struct_df['price']

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

#Create a linear regression model
model = LinearRegression()

#Fit the model to the training data
model.fit(X_train, y_train)
```

[18]: LinearRegression()

[19]:
```
#Check r-squared value of the model
struct_model = sm.add_constant(X_train)
struct_model_res = OLS(y_train,X_train).fit()
struct_model_res.summary()
```

[19]:

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared (uncentered):** | 0.852 |
| **Model:** | OLS | **Adj. R-squared (uncentered):** | 0.852 |
| **Method:** | Least Squares | **F-statistic:** | 1.245e+04 |
| **Date:** | Mon, 11 Aug 2025 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 17:13:37 | **Log-Likelihood:** | -2.3938e+05 |
| **No. Observations:** | 17290 | **AIC:** | 4.788e+05 |
| **Df Residuals:** | 17282 | **BIC:** | 4.788e+05 |
| **Df Model:** | 8 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **bedrooms** | -7.386e+04 | 2305.837 | -32.030 | 0.000 | -7.84e+04 | -6.93e+04 |
| **bathrooms** | -6975.0981 | 4136.424 | -1.686 | 0.092 | -1.51e+04 | 1132.712 |
| **sqft_living** | -36.1311 | 9.294 | -3.888 | 0.000 | -54.347 | -17.915 |
| **sqft_lot** | -0.4249 | 0.051 | -8.257 | 0.000 | -0.526 | -0.324 |
| **floors** | -2.543e+04 | 4635.012 | -5.486 | 0.000 | -3.45e+04 | -1.63e+04 |
| **grade** | 3.572e+04 | 1360.819 | 26.246 | 0.000 | 3.3e+04 | 3.84e+04 |
| **sqft_above** | 309.4438 | 9.333 | 33.155 | 0.000 | 291.150 | 327.738 |
| **sqft_basement** | 346.4075 | 10.261 | 33.760 | 0.000 | 326.295 | 366.520 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 12831.065 | **Durbin-Watson:** | 2.016 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 600346.035 |
| **Skew:** | 3.096 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 31.195 | **Cond. No.** | 1.10e+05 |

Notes:
[1] $R^2$ is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 1.1e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

[20]:
```
# Test the model on the test set
y_pred = model.predict(X_test)
y_pred
```

[20]:
```
array([ 520304.51688853,  723515.5726481 , 1167295.49821263, …,
        589363.67177804,  578289.07455117,  614042.72904717])
```

[21]:
```
#Mean absolute error
mean_absolute_error(y_test, y_pred)
```

[21]:
```
164302.3359401914
```

[22]:
```
#Cross validation method
cv = KFold(n_splits = 5, random_state=222, shuffle=True)

#Use the method to evaluate model
scores = cross_val_score(model, X_train, y_train, scoring =␣
 ↪'neg_mean_absolute_error', cv=cv, n_jobs = -1)
```

```
#Check mean absolute error
mae_scores = mean(absolute(scores))
mae_scores
```

[22]: 159529.66172237677

[51]:
```
print("Interpretation:")
print("As we can see from the data, sqft_living has the strongest correlation␣
↪out of the structure related features\n and grade comes in close second.␣
↪Looking at our correlation plot,\n we can see that as sqft_living goes up,␣
↪price goes up to a certain degree.\n We can also see a good R^2 score from␣
↪our regression model by using these structure related features. \n We have␣
↪alot of variation between the actual prices and the predicted prices, with␣
↪high mean absolute error too.")
```

```
Interpretation:
As we can see from the data, sqft_living has the strongest correlation out of
the structure related features
 and grade comes in close second. Looking at our correlation plot,
 we can see that as sqft_living goes up, price goes up to a certain degree.
 We can also see a good R^2 score from our regression model by using these
structure related features.
 We have alot of variation between the actual prices and the predicted prices,
with high mean absolute error too.
```

[23]:
```
#Create dataframe using every feature but price, binary features, and other␣
↪insignificant features to price
num_feat = housing.drop(['price', 'id', 'date', 'zipcode', 'lat', 'long'],␣
↪axis=1)

#Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(num_feat, y, test_size=0.2,␣
↪random_state=42)

#Fit the model to the training data
model.fit(X_train, y_train)
```

[23]: LinearRegression()

[24]:
```
#Check r-squared value of the model
struct_model = sm.add_constant(X_train)
struct_model_res = OLS(y_train,X_train).fit()
struct_model_res.summary()
```

[24]:

| | | | |
|---|---|---|---|
| **Dep. Variable:** | price | **R-squared (uncentered):** | 0.881 |
| **Model:** | OLS | **Adj. R-squared (uncentered):** | 0.881 |
| **Method:** | Least Squares | **F-statistic:** | 8557. |
| **Date:** | Mon, 11 Aug 2025 | **Prob (F-statistic):** | 0.00 |
| **Time:** | 17:13:40 | **Log-Likelihood:** | -2.3747e+05 |
| **No. Observations:** | 17290 | **AIC:** | 4.750e+05 |
| **Df Residuals:** | 17275 | **BIC:** | 4.751e+05 |
| **Df Model:** | 15 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **bedrooms** | -2.814e+04 | 2331.374 | -12.069 | 0.000 | -3.27e+04 | -2.36e+04 |
| **bathrooms** | -5754.6857 | 3720.244 | -1.547 | 0.122 | -1.3e+04 | 1537.370 |
| **sqft_living** | -26.1473 | 8.335 | -3.137 | 0.002 | -42.485 | -9.809 |
| **sqft_lot** | 0.0285 | 0.067 | 0.427 | 0.670 | -0.103 | 0.160 |
| **floors** | 3554.9151 | 4278.323 | 0.831 | 0.406 | -4831.031 | 1.19e+04 |
| **waterfront** | 5.541e+05 | 2.2e+04 | 25.237 | 0.000 | 5.11e+05 | 5.97e+05 |
| **view** | 5.531e+04 | 2608.754 | 21.203 | 0.000 | 5.02e+04 | 6.04e+04 |
| **condition** | 5.816e+04 | 2686.859 | 21.647 | 0.000 | 5.29e+04 | 6.34e+04 |
| **grade** | 1.11e+05 | 2584.427 | 42.967 | 0.000 | 1.06e+05 | 1.16e+05 |
| **sqft_above** | 190.4379 | 8.988 | 21.187 | 0.000 | 172.820 | 208.056 |
| **sqft_basement** | 218.9183 | 9.494 | 23.060 | 0.000 | 200.310 | 237.527 |
| **yr_built** | -409.8080 | 9.716 | -42.178 | 0.000 | -428.852 | -390.763 |
| **yr_renovated** | 69.7425 | 4.309 | 16.184 | 0.000 | 61.296 | 78.189 |
| **sqft_living15** | 17.2615 | 4.160 | 4.150 | 0.000 | 9.108 | 25.415 |
| **sqft_lot15** | -0.6515 | 0.092 | -7.062 | 0.000 | -0.832 | -0.471 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 12922.694 | **Durbin-Watson:** | 2.009 |
| **Prob(Omnibus):** | 0.000 | **Jarque-Bera (JB):** | 914805.516 |
| **Skew:** | 2.985 | **Prob(JB):** | 0.00 |
| **Kurtosis:** | 38.131 | **Cond. No.** | 6.22e+05 |

Notes:
[1] $R^2$ is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 6.22e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

[25]:
```
# Test the model on the test set
y_pred = model.predict(X_test)
y_pred
```

[25]:
```
array([ 528528.46770355,  684231.4757155 , 1227421.21542145, …,
        507056.85939439,  562061.59288962,  523706.72411786])
```

[26]:
```
#Mean absolute error
mean_absolute_error(y_test, y_pred)
```

[26]:
```
143433.17178940342
```

[27]:
```python
#Cross validation method
cv = KFold(n_splits = 5, random_state=222, shuffle=True)

#Use the method to evaluate model
scores = cross_val_score(model, X_train, y_train, scoring =␣
↪'neg_mean_absolute_error', cv=cv, n_jobs = -1)

#Check mean absolute error
mae_scores = mean(absolute(scores))
mae_scores
```

[27]:
```
139202.8829235144
```

[55]:
```python
print("Interpretation:")
print("The final model is picked based off of the accuracy of the R^2 score. We␣
↪have a score of 0.881, which is the highest within our models.\nThe features␣
↪make sense in predicting a increase in price as well. Our mean absolute␣
↪error decreased with this model and predicted values\nwill be closer to the␣
↪actual prices. Overall, this model seems to be the best for predicting the␣
↪price for housing out of all models tested.")
```

```
Interpretation:
The final model is picked based off of the accuracy of the R^2 score. We have a
score of 0.881, which is the highest within our models.
The features make sense in predicting a increase in price as well. Our mean
absolute error decreased with this model and predicted values
will be closer to the actual prices. Overall, this model seems to be the best
for predicting the price for housing out of all models tested.
```

[ ]:
```python

```