King County Housing Prices Modeling Project

Overview

In today's competitive real estate market, homeowners seek ways to maximize the value of their properties through renovations. This project aims to leverage linear regression modeling to provide insights into how different types of home renovations can affect the estimated value of homes. By analyzing historical sales data and applying regression techniques, we will quantify the impact of specific renovation projects on home prices, helping homeowners make data-driven decisions and real estate agents provide expert advice to their clients.

Business Understanding

The primary objective of this project is to provide actionable insights to a real estate agency that assists homeowners in buying and selling properties. The agency's clients often inquire about the potential increase in home value resulting from various renovation projects. Therefore, this project aims to address the following key objectives:

1. Develop a Predictive Model: Create and validate a linear regression model that predicts the increase in home value based on the type and extent of renovations undertaken, ensuring its reliability and applicability to various property types and market conditions.
2. Quantify the Impact of Renovations: Determine how different types of home renovations contribute to the overall increase in property value by analyzing historical sales data and identifying the renovations that provide the highest return on investment.

Data Understanding

This project utilizes the King County House Sales dataset, contained in the file named kc_house_data.csv. This dataset includes various features related to house sales, such as square footage, number of bedrooms and bathrooms, presence of a waterfront, view quality, year built, and renovation status. The dataset also provides the sale price of each property, which serves as the dependent variable in our regression modeling. Detailed descriptions of the column names can be found in the accompanying column_names.md file. The aim is to leverage this rich dataset to understand and quantify the impact of various home renovations on property values.

In [1]:
```
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_

# Load the dataset
df = pd.read_csv('./data/kc_house_data.csv')

# Reading the column descriptions
with open('./data/column_names.md', 'r') as file:
    column_descriptions = file.read()

print(column_descriptions)
```

```
# Column Names and descriptions for Kings County Data Set
* **id** - unique identified for a house
* **dateDate** - house was sold
* **pricePrice** -  is prediction target
* **bedroomsNumber** -  of Bedrooms/House
* **bathroomsNumber** -  of bathrooms/bedrooms
* **sqft_livingsquare** -  footage of the home
* **sqft_lotsquare** -  footage of the lot
* **floorsTotal** -  floors (levels) in house
* **waterfront** - House which has a view to a waterfront
* **view** - Has been viewed
* **condition** - How good the condition is ( Overall )
* **grade** - overall grade given to the housing unit, based on King Co
unty grading system
* **sqft_above** - square footage of house apart from basement
* **sqft_basement** - square footage of the basement
* **yr_built** - Built Year
* **yr_renovated** - Year when house was renovated
* **zipcode** - zip
* **lat** - Latitude coordinate
* **long** - Longitude coordinate
* **sqft_living15** - The square footage of interior housing living spa
ce for the nearest 15 neighbors
* **sqft_lot15** - The square footage of the land lots of the nearest 1
5 neighbors
```

Column Names and Descriptions:

Based on the column descriptions, below are further comments on some of them based on relevance for modelling or predicting house prices.

- **Id and Date:** These columns can be useful for identifying records and time-based analysis. However, they may not be directly useful for modeling.

- **Price:** This is the target variable we aim to predict.
- **Bedrooms and Bathrooms:** These are essential features representing the size and functionality of the house.
- **Square Footage:** The living area (sqft_living) and lot size (sqft_lot) are crucial features for predicting house prices.
- **Floors, Waterfront, View, Condition, and Grade:** These categorical features can significantly influence the house price and need to be encoded appropriately.
- **Year Built and Year Renovated:** These features can indicate the age and modernity of the house.
- **Location (Latitude, Longitude, and ZIP code):** Location features are often critical in real estate price prediction due to the influence of neighborhood and geographical factors.
- **Square Footage of Neighbors:** The living area and lot size of neighboring houses can provide context on the neighborhood's characteristics.

In [2]:
```
# Display basic information about the dataframe
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
```

Out[2]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+

Dataset Information:

- **Number of Rows:** 21,597
- **Number of Columns:** 21

Descriptive Statistics:

- **Price:** The average house price is approximately $540,000.
- **Bedrooms:** On average, houses have around 3.37 bedrooms.
- **Bathrooms:** The average number of bathrooms is 2.11.
- **Square Footage (Living Area):** The average living area is about 2,080 square feet.
- **Square Footage (Lot):** The average lot size is about 15,090 square feet.

EDA & Data Cleaning

In [3]:
```
# Inspect the Data
print(df.shape)
df.head()
```

```
(21597, 21)
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	\
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [4]:
```
# Check for missing values
print(df.isnull().sum())
```

```
id                 0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2376
view              63
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated    3842
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

In [5]:
```
# Check if there are any duplicates in the entire DataFrame
any_duplicates = df.duplicated().any()
print(f"Any duplicates: {any_duplicates}")
```

```
Any duplicates: False
```

Duplicates & Missing Values Observation:

- There are no duplicates.
- Only three columns have missing values, that is, waterfront(2,376), view(63) and yr_renovated(3,842).
- Waterfont and view have categorical data and therefore will use mode to fill the missing values while mean is used for the yr renovated

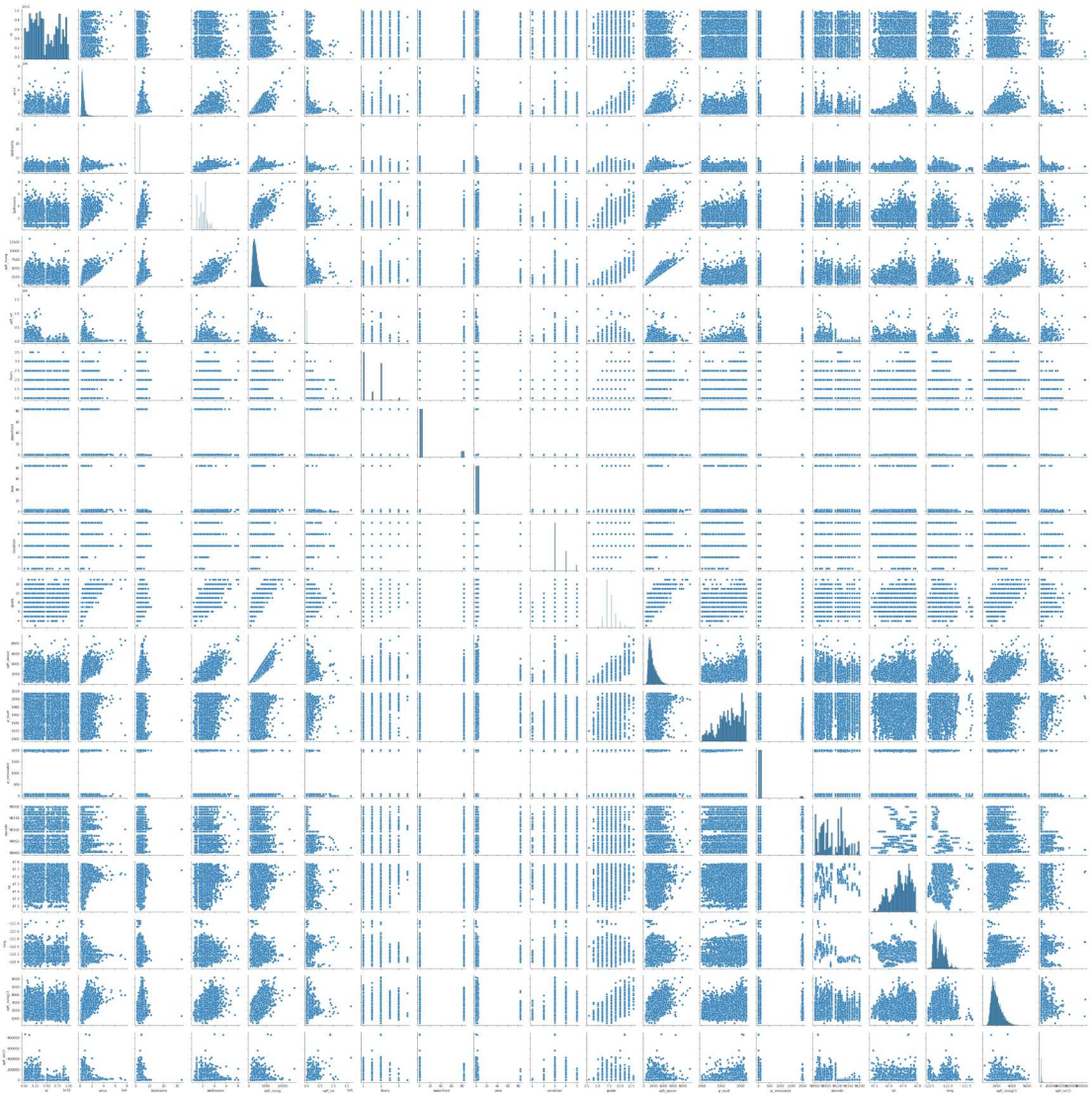
In [6]:
```
# Address missing values
df = df.fillna(df['yr_renovated'].mean())
df = df.fillna(df['waterfront'].mode()) # categorical data
df = df.fillna(df['view'].mode()) # categorical data
df.isnull().sum()
```

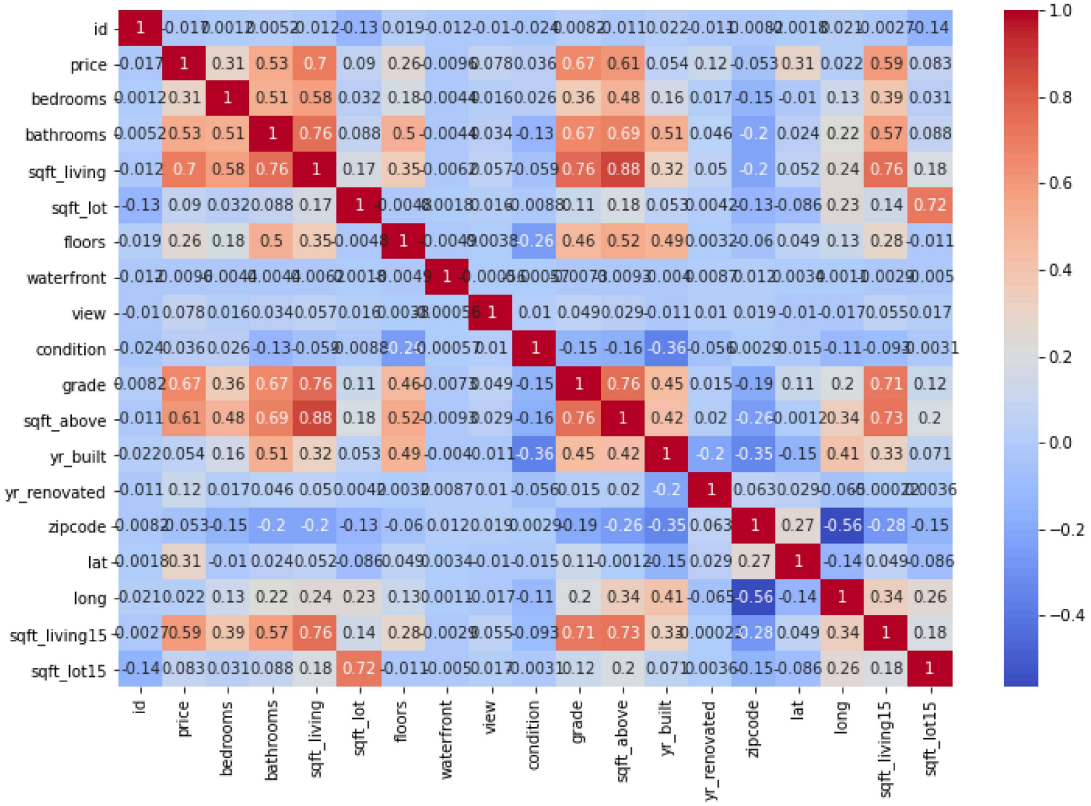
Out[6]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

Data Preparation

In [7]:

```
# Pairplot to see relationships
sns.pairplot(df)
plt.show()

# Correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```





Since the pairplot and heatmap above are difficult to read, a correlation table rounded off to two decimal places is generated below for readability.

In [8]:
```
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Convert the correlation matrix to a readable format
correlation_table = correlation_matrix.round(2)
correlation_table
```

Out[8]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
id	1.00	-0.02	0.00	0.01	-0.01	-0.13	0.02	-0.01
price	-0.02	1.00	0.31	0.53	0.70	0.09	0.26	-0.01
bedrooms	0.00	0.31	1.00	0.51	0.58	0.03	0.18	-0.00
bathrooms	0.01	0.53	0.51	1.00	0.76	0.09	0.50	-0.00
sqft_living	-0.01	0.70	0.58	0.76	1.00	0.17	0.35	-0.01
sqft_lot	-0.13	0.09	0.03	0.09	0.17	1.00	-0.00	0.00
floors	0.02	0.26	0.18	0.50	0.35	-0.00	1.00	-0.00
waterfront	-0.01	-0.01	-0.00	-0.00	-0.01	0.00	-0.00	1.00
view	-0.01	0.08	0.02	0.03	0.06	0.02	0.00	-0.00
condition	-0.02	0.04	0.03	-0.13	-0.06	-0.01	-0.26	-0.00
grade	0.01	0.67	0.36	0.67	0.76	0.11	0.46	-0.01
sqft_above	-0.01	0.61	0.48	0.69	0.88	0.18	0.52	-0.01
yr_built	0.02	0.05	0.16	0.51	0.32	0.05	0.49	-0.00
yr_renovated	-0.01	0.12	0.02	0.05	0.05	0.00	0.00	0.01
zipcode	-0.01	-0.05	-0.15	-0.20	-0.20	-0.13	-0.06	0.01
lat	-0.00	0.31	-0.01	0.02	0.05	-0.09	0.05	0.00
long	0.02	0.02	0.13	0.22	0.24	0.23	0.13	0.00
sqft_living15	-0.00	0.59	0.39	0.57	0.76	0.14	0.28	-0.00
sqft_lot15	-0.14	0.08	0.03	0.09	0.18	0.72	-0.01	-0.00

Feature Selection:

Based on correlation analysis and domain knowledge, below features are recommended for modelling due to their potential impact on the target variable (price) and their practical relevance in real estate valuation.

1. sqft_living (Square Footage of Living Area):

- **Reason for Selection:** The sqft_living has the highest correlation with price at 0.7.
- **Expected Impact:** With a positive correlation of 0.70 with price, larger living areas are expected to lead to higher house prices.

2. grade (Overall Grade Given to the Housing Unit):

- **Reason for Selection:** Grade equally has a high correlation with Price at 0.67. The grade reflects the quality of construction and finishes in the house. Higher grades indicate better quality, which can be a significant selling point.
- **Expected Impact:** With a positive correlation of 0.67 with price, higher grades would be expected to result in higher property values.

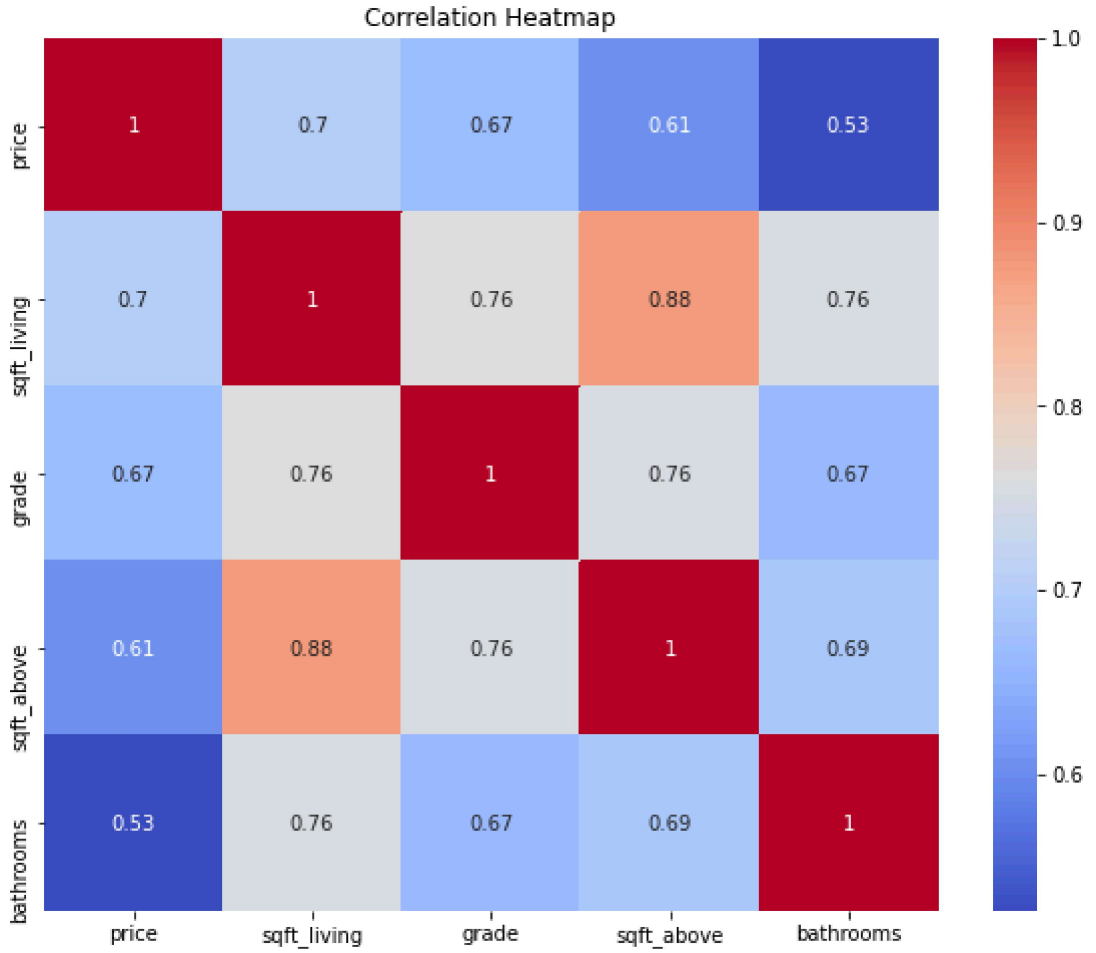
3. sqft_above (Square Footage of the House Apart from the Basement):

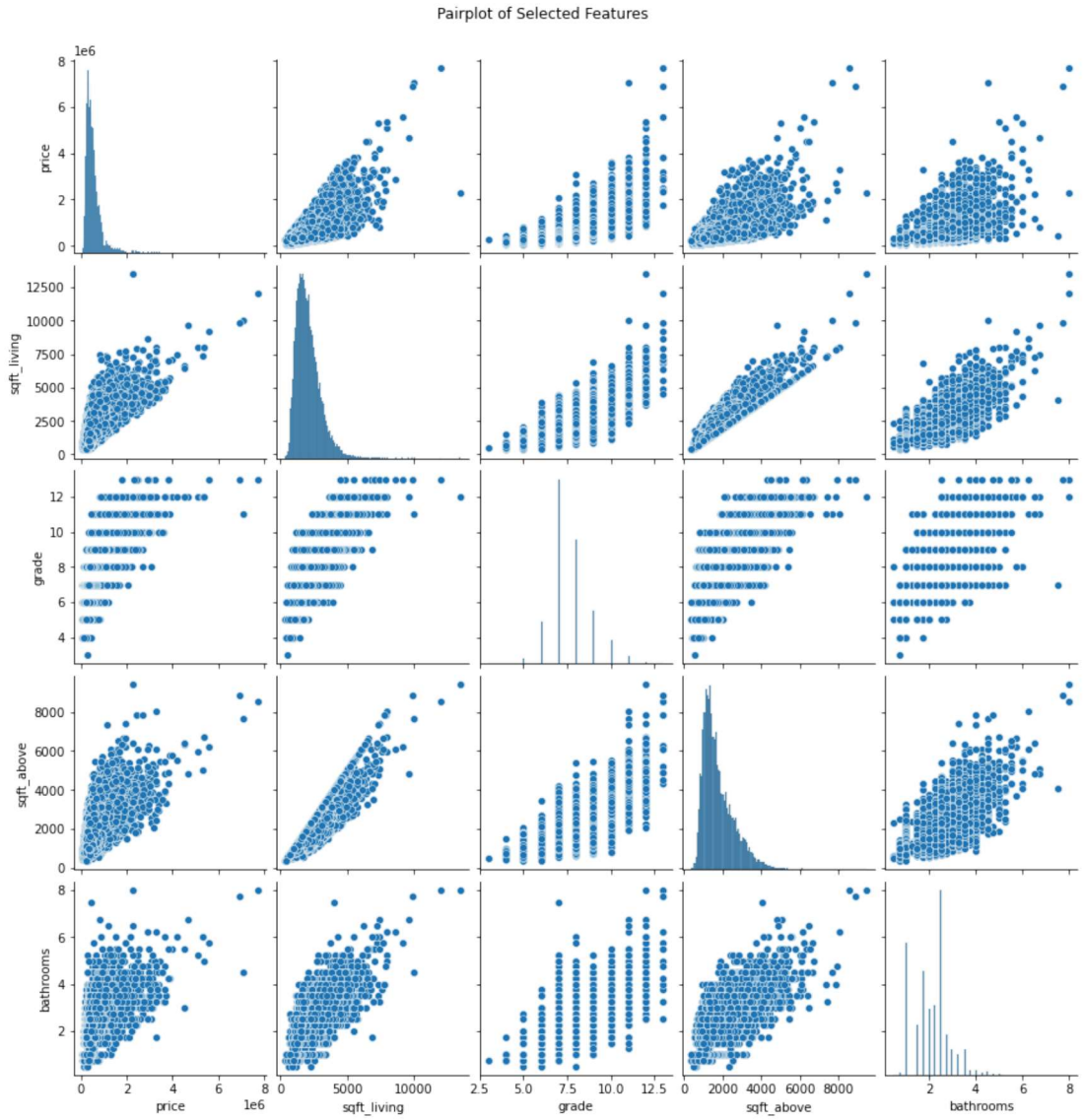
- **Reason for Selection:** This feature represents the above-ground living area, which is often more valuable than basement space. It also has a moderately strong positive correlation of 0.61 with price.
- **Expected Impact:** With a positive correlation of 0.61 with price, more above-ground living space would generally be expected to increases the house's value.

4. bathrooms (Number of Bathrooms):

- **Reason for Selection:** The number of bathrooms is a crucial factor for buyers, as it impacts the functionality and convenience of the house. It also has a moderately strong positive correlation of 0.53 with price.
- **Expected Impact:** With the positive correlation of 0.53 with price, houses with more bathrooms would be expected to be valued higher.

In [9]:
```
# Select a subset of features for pairplot to avoid clutter
sub_features = ['price', 'sqft_living', 'grade', 'sqft_above', 'bathrooms

# Correlation heatmap
plt.figure(figsize=(10, 8))
correlation_matrix = df[sub_features].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()

# Pairplot
sns.pairplot(df[sub_features])
plt.suptitle("Pairplot of Selected Features", y=1.02)
plt.show()
```





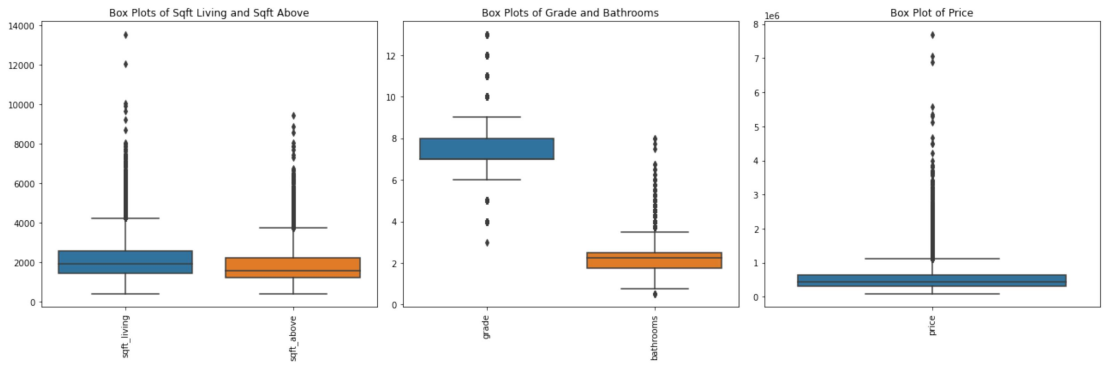
Selected Features Correlation Observations:

The correlation heatmap and pairplot for the selected features are shown above to better understand the relationships between the features and the target variable price. Here are some key observations that further support the potential of the selected features in predicting house prices:

- **Correlation Heatmap:** sqft_living and grade have a strong positive correlation with price. This indicates that larger living spaces and higher grades are associated with higher house prices.
- **Pairplot:** The scatter plots in the pairplot show linear relationships between price and the selected features, particularly for sqft_living and grade.
- **Multicollinearity:** High correlation between sqft_living and sqft_above suggests multicollinearity, which needs to be addressed in modeling.

Checking for Outliers:

In [10]:
```
# Visualize distributions and identify potential outliers
# Define the feature groups
sub1_features = ['sqft_living', 'sqft_above']
sub2_features = ['grade', 'bathrooms']
sub3_features = ['price']

# Create a figure with subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=False)

# Plot for sqft_living and sqft_above
sns.boxplot(data=df[sub1_features], ax=axes[0])
axes[0].set_title('Box Plots of Sqft Living and Sqft Above')
axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=90)

# Plot for grade and bathrooms
sns.boxplot(data=df[sub2_features], ax=axes[1])
axes[1].set_title('Box Plots of Grade and Bathrooms')
axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=90)

# Plot for price
sns.boxplot(data=df[sub3_features], ax=axes[2])
axes[2].set_title('Box Plot of Price')
axes[2].set_xticklabels(axes[2].get_xticklabels(), rotation=90)

# Display the plots
plt.tight_layout()
plt.show()
```



Outliers Observations:

The box plots indicate the presence of outliers in all examined features (sqft_living, sqft_above, grade, bathrooms, and price). These outliers, particularly on the higher end, can skew the data distribution and potentially impact the model's accuracy and reliability. It is recommended to handle these outliers by either capping them at a certain percentile or removing them to help in reducing their influence on the model and improving the overall robustness and generalizability of the predictions.

Modeling & Validation

Key Metrics for Evaluation:

- **Mean Absolute Error (MAE):** Measures the average magnitude of the errors in a set of predictions, without considering their direction. Lower MAE is better.
- **Mean Squared Error (MSE):** Measures the average of the squares of the errors, giving more weight to larger errors. Lower MSE is better.
- **R-Squared (R^2):** Indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. Higher R^2 is better, with a maximum value of 1.

Iteration 1: Without addressing Outliers, Multicollinearity and with No Scaling

In [11]:
```
# Inspect data
df[sub_features].head()
```

Out[11]:

	price	sqft_living	grade	sqft_above	bathrooms
0	221900.0	1180	7	1180	1.00
1	538000.0	2570	7	2170	2.25
2	180000.0	770	6	770	1.00
3	604000.0	1960	7	1050	3.00
4	510000.0	1680	8	1680	2.00

In [12]:

```
# Separate features and target
# Define X1 and y1
X1 = df[sub_features].drop('price', axis=1)
y1 = df['price']

# Split the data into train and test sets
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_siz

# Model 1: Simple Linear Regression with one feature (sqft_living)
model1 = LinearRegression()
model1.fit(X1_train[['sqft_living']], y1_train)
y_pred1 = model1.predict(X1_test[['sqft_living']])

# Evaluate Model 1
mae1 = mean_absolute_error(y1_test, y_pred1)
mse1 = mean_squared_error(y1_test, y_pred1)
r2_1 = r2_score(y1_test, y_pred1)

# Model 2: Linear Regression with two features (sqft_living and grade)
model2 = LinearRegression()
model2.fit(X1_train[['sqft_living', 'grade']], y1_train)
y_pred2 = model2.predict(X1_test[['sqft_living', 'grade']])

# Evaluate Model 2
mae2 = mean_absolute_error(y1_test, y_pred2)
mse2 = mean_squared_error(y1_test, y_pred2)
r2_2 = r2_score(y1_test, y_pred2)

# Model 3: Linear Regression with three features (sqft_living, grade, an
model3 = LinearRegression()
model3.fit(X1_train[['sqft_living', 'grade', 'sqft_above']], y1_train)
y_pred3 = model3.predict(X1_test[['sqft_living', 'grade', 'sqft_above']]

# Evaluate Model 3
mae3 = mean_absolute_error(y1_test, y_pred3)
mse3 = mean_squared_error(y1_test, y_pred3)
r2_3 = r2_score(y1_test, y_pred3)

# Model 4: Linear Regression with four features (e.g., sqft_living, grad
model4 = LinearRegression()
model4.fit(X1_train, y1_train)
y_pred4 = model4.predict(X1_test)

# Evaluate Model 4
mae4 = mean_absolute_error(y1_test, y_pred4)
mse4 = mean_squared_error(y1_test, y_pred4)
r2_4 = r2_score(y1_test, y_pred4)

print(f"Model 1 - MAE: {mae1}, MSE: {mse1}, R2: {r2_1}")
print(f"Model 2 - MAE: {mae2}, MSE: {mse2}, R2: {r2_2}")
print(f"Model 3 - MAE: {mae3}, MSE: {mse3}, R2: {r2_3}")
print(f"Model 4 - MAE: {mae4}, MSE: {mse4}, R2: {r2_4}")
```

```
Model 1 - MAE: 170982.92465955476, MSE: 65977373783.61759, R2: 0.493324
69237979504
Model 2 - MAE: 162840.69782208806, MSE: 61550769137.98829, R2: 0.527318
9413460091
Model 3 - MAE: 161005.30381084685, MSE: 61204919012.7009, R2: 0.5299749
082111829
Model 4 - MAE: 159888.3637660468, MSE: 60729065072.41195, R2: 0.5336292
434438783
```

Model 4 (with features sqft_living, grade, bathrooms, and sqft_above) is the best performing model in this iteration. It has the lowest errors (MAE & MSE) and highest explanatory power (R2). The results also indicate that including additional relevant features continued to improve the model's performance.

Iteration 2: With Outliers Addressed

Capped outliers to retain all data points and reduce the influence of extreme values and preserve overall data structure. Based on the model performance, removing outliers might be considered in further model iterations.

In [13]:
```
# Cap outliers using IQR method
def cap_outliers(data, columns):
    for feature in columns:
        Q1 = data[feature].quantile(0.25)
        Q3 = data[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        data[feature] = np.where(data[feature] < lower_bound, lower_boun
        data[feature] = np.where(data[feature] > upper_bound, upper_boun
    return data

# Cap outliers in the selected features
df_capped = cap_outliers(df, sub_features)

# Display the capped features
df_capped[sub_features].describe()
```

Out[13]:

	price	sqft_living	grade	sqft_above	bathrooms
count	2.159700e+04	21597.000000	21597.000000	21597.000000	21597.000000
mean	5.117047e+05	2058.392184	7.599134	1769.804788	2.099244
std	2.499734e+05	838.660736	1.000943	763.788290	0.721473
min	7.800000e+04	370.000000	5.500000	370.000000	0.625000
25%	3.220000e+05	1430.000000	7.000000	1190.000000	1.750000
50%	4.500000e+05	1910.000000	7.000000	1560.000000	2.250000
75%	6.450000e+05	2550.000000	8.000000	2210.000000	2.500000
max	1.129500e+06	4230.000000	9.500000	3740.000000	3.625000

In [14]:

```
# Separate features and target
# Define X2 and y2
X2 = df_capped[sub_features].drop('price', axis=1)
y2 = df_capped['price']

# Split the data into train and test sets
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_siz

# Model 1: Simple Linear Regression with one feature (sqft_living)
model1 = LinearRegression()
model1.fit(X2_train[['sqft_living']], y2_train)
y_pred1 = model1.predict(X2_test[['sqft_living']])

# Evaluate Model 1
mae1 = mean_absolute_error(y2_test, y_pred1)
mse1 = mean_squared_error(y2_test, y_pred1)
r2_1 = r2_score(y2_test, y_pred1)

# Model 2: Linear Regression with two features (sqft_living and grade)
model2 = LinearRegression()
model2.fit(X2_train[['sqft_living', 'grade']], y2_train)
y_pred2 = model2.predict(X2_test[['sqft_living', 'grade']])

# Evaluate Model 2
mae2 = mean_absolute_error(y2_test, y_pred2)
mse2 = mean_squared_error(y2_test, y_pred2)
r2_2 = r2_score(y2_test, y_pred2)

# Model 3: Linear Regression with three features (sqft_living, grade, an
model3 = LinearRegression()
model3.fit(X2_train[['sqft_living', 'grade', 'sqft_above']], y2_train)
y_pred3 = model3.predict(X2_test[['sqft_living', 'grade', 'sqft_above']]

# Evaluate Model 3
mae3 = mean_absolute_error(y2_test, y_pred3)
mse3 = mean_squared_error(y2_test, y_pred3)
r2_3 = r2_score(y2_test, y_pred3)

# Model 4: Linear Regression with four features (sqft_living, grade, sqf
model4 = LinearRegression()
model4.fit(X2_train, y2_train)
y_pred4 = model4.predict(X2_test)

# Evaluate Model 4
mae4 = mean_absolute_error(y2_test, y_pred4)
mse4 = mean_squared_error(y2_test, y_pred4)
r2_4 = r2_score(y2_test, y_pred4)

print(f"Model 1 - MAE: {mae1}, MSE: {mse1}, R2: {r2_1}")
print(f"Model 2 - MAE: {mae2}, MSE: {mse2}, R2: {r2_2}")
print(f"Model 3 - MAE: {mae3}, MSE: {mse3}, R2: {r2_3}")
print(f"Model 4 - MAE: {mae4}, MSE: {mse4}, R2: {r2_4}")
```

```
Model 1 - MAE: 140758.1831601869, MSE: 31532711726.328255, R2: 0.491034
96929645996
Model 2 - MAE: 131120.88073440106, MSE: 27770344490.892174, R2: 0.55176
28055866342
Model 3 - MAE: 129803.11149984438, MSE: 27432949834.652058, R2: 0.55720
86449125644
Model 4 - MAE: 129160.6885549101, MSE: 27242617861.51876, R2: 0.5602807
66314314
```

Capping outliers led to a improvement in all models, particularly in reducing MAE and MSE, and further improving R2 for the best performing model (Model 4).

Iteration 3: With Multicollinearity Addressed

Performed this iteration on Model 4 only since it emerged as the best performing model from the previous iterations.

In [15]:
```
### Dropped sqft_abv to address high multicollinearity between sqft_livi
df2 = df_capped[sub_features].drop('sqft_above', axis=1)
df2.head()
```

Out[15]:

	price	sqft_living	grade	bathrooms
0	221900.0	1180.0	7.0	1.00
1	538000.0	2570.0	7.0	2.25
2	180000.0	770.0	6.0	1.00
3	604000.0	1960.0	7.0	3.00
4	510000.0	1680.0	8.0	2.00

In [16]:
```
# Separate features and target
# Define X3 and y3
X3 = df2.drop('price', axis=1)
y3 = df2['price']

# Split the data into train and test sets
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_siz

# Model 4: Linear Regression with sqft_above dropped
model4 = LinearRegression()
model4.fit(X3_train, y3_train)
y_pred4 = model4.predict(X3_test)

# Evaluate Model 4
mae4 = mean_absolute_error(y3_test, y_pred4)
mse4 = mean_squared_error(y3_test, y_pred4)
r2_4 = r2_score(y3_test, y_pred4)

print(f"Model 4 - MAE: {mae4}, MSE: {mse4}, R2: {r2_4}")
```

```
Model 4 - MAE: 130430.56277187045, MSE: 27575351995.251114, R2: 0.55491
01518216251
```

The results indicate that dropping sqft_above did not improve the best perfoming model (Model 4). It resulted in a slight increase in both MAE and MSE and a slight decrease in R2. This suggests that the model's predictive performance slightly worsened after dropping one feature.

Iteration 4: With Scaled Predictors

Continued iterating on the best performing model, Model 4 under iteration 2 to check if scaling predictors would improved the model further.

In [17]:
```
# Inspect data
df3 = df_capped[sub_features]
df3.head()
```

Out[17]:

	price	sqft_living	grade	sqft_above	bathrooms
0	221900.0	1180.0	7.0	1180.0	1.00
1	538000.0	2570.0	7.0	2170.0	2.25
2	180000.0	770.0	6.0	770.0	1.00
3	604000.0	1960.0	7.0	1050.0	3.00
4	510000.0	1680.0	8.0	1680.0	2.00

In [18]:
```
# Separate features and target
# Define X4 and y4
X4 = df3.drop('price', axis=1)
y4 = df3['price']

# Split the data into train and test sets
X4_train, X4_test, y4_train, y4_test = train_test_split(X4, y4, test_siz

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data
scaler.fit(X4_train)

# Transform the training data
X_train_scaled = scaler.transform(X4_train)

# Transform the test data
X_test_scaled = scaler.transform(X4_test)

# Model 4: Linear Regression with scaled predictors
model4 = LinearRegression()
model4.fit(X_train_scaled, y4_train)
y_pred4 = model4.predict(X_test_scaled)

# Evaluate Model 4
mae4 = mean_absolute_error(y4_test, y_pred4)
mse4 = mean_squared_error(y4_test, y_pred4)
r2_4 = r2_score(y4_test, y_pred4)

print(f"Model 4 - MAE: {mae4}, MSE: {mse4}, R2: {r2_4}")
```

```
Model 4 - MAE: 129160.68855490991, MSE: 27242617861.51887, R2: 0.560280
7663143121
```

Results indicate that scaling the predictors did not improve the model's performance. The metrics (MAE, MSE, and R2) are exactly the same before and after scaling. This could be because Linear regression models are not affected by the absolute scales of the input features because the optimization of the model parameters inherently adjusts for these scales. Therefore, the performance metrics remain unchanged.

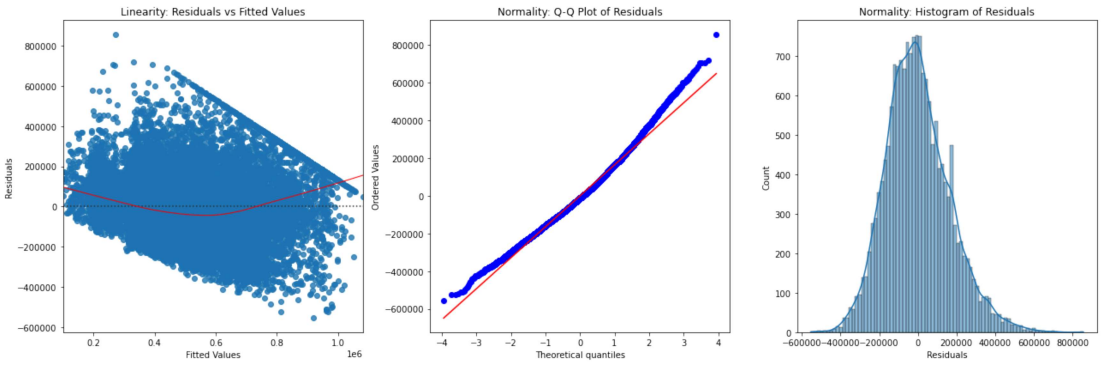
Model 4 Diagnosis:

Checked whether the best performing model (Model 4, under iteration 2) conforms to common assumption of linear regression (Linearity & Normality) to ensure the model is valid and reliable.

In [19]:
```
# Model 4: Linear Regression with four features (sqft_living, grade, sqft
model4 = LinearRegression()
model4.fit(X2_train, y2_train)
y_pred4 = model4.predict(X2_test)

# Evaluate Model 4
mae4 = mean_absolute_error(y2_test, y_pred4)
mse4 = mean_squared_error(y2_test, y_pred4)
r2_4 = r2_score(y2_test, y_pred4)

print(f"Model 4 - MAE: {mae4}, MSE: {mse4}, R2: {r2_4}")
```

```
Model 4 - MAE: 129160.6885549101, MSE: 27242617861.51876, R2: 0.5602807
66314314
```

In [20]:
```
# Residuals vs. Fitted values plot
fitted_values = model4.predict(X2_train)
residuals = y2_train - fitted_values

# Check on conformity with Assumptions of Linear Regression
# Plot Residuals vs Fitted Values, Q-Q Plot of Residuals, and Histogram o
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Residuals vs Fitted Values (Linearity Assumption)
sns.residplot(x=fitted_values, y=residuals, lowess=True, line_kws={'colo
axs[0].set_xlabel('Fitted Values')
axs[0].set_ylabel('Residuals')
axs[0].set_title('Linearity: Residuals vs Fitted Values')

# Q-Q Plot of Residuals (Normality Assumption)
stats.probplot(residuals, dist="norm", plot=axs[1])
axs[1].get_lines()[1].set_color('red')
axs[1].set_title('Normality: Q-Q Plot of Residuals')

# Histogram of Residuals (Normality Assumption)
sns.histplot(x=residuals, kde=True, ax=axs[2])
axs[2].set_xlabel('Residuals')
axs[2].set_title('Normality: Histogram of Residuals')

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



Model 4 Diagnosis Observations:

- **Linearity:** For the linearity assumption to hold, the residuals should be randomly scattered around the horizontal axis, with no clear pattern. The plot shows a curved pattern, indicating that the residuals are not randomly scattered around the horizontal axis and hence assumption of linearity is violated.
- **Normality:** For the normality assumption to hold, the points in the Q-Q plot should lie approximately along the red line and the histogram of the residuals should resemble a bell-shaped curve, symmetric around zero. Both the Q-Q plot and the Histogram of Residuals suggest that the residuals are not perfectly normally distributed. There are deviations from normality, particularly in the tails, indicating potential presence of outliers or skewness.
- **Next Steps:** We will consider log transforming the dependent variable and scaling predictors to stabilize variance and achieve linearity. At the same time, we will consider removing outliers to improve normality. And since previous iterations suggested that adding relevant additional features improves model performance, bedrooms with a 0.31 correlation with price is included.

Iteration 5: With additional Feature, Outliers Removed, Predictors Scaled and Price Transformed

In [21]:
```
# Inspect data
df.head()
```

Out[21]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	7129300520	10/13/2014	221900.0	3	1.00	1180.0	5650	1.0
1	6414100192	12/9/2014	538000.0	3	2.25	2570.0	7242	2.0
2	5631500400	2/25/2015	180000.0	2	1.00	770.0	10000	1.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960.0	5000	1.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680.0	8080	1.0

5 rows × 21 columns

In [22]:

```
# Function to remove outliers
def remove_outliers(df, columns):
    for feature in columns:
        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df = df[(df[feature] >= lower_bound) & (df[feature] <= upper_bou
    return df

# Define the features to clean
features_to_clean = ['sqft_living', 'grade', 'sqft_above', 'bathrooms',

# Remove outliers in the specified features
df = remove_outliers(df, features_to_clean)

# Log-transform the dependent variable
df['log_price'] = np.log(df['price'])

# Define the features and the log-transformed target variable
features = ['sqft_living', 'grade', 'sqft_above', 'bathrooms', 'bedrooms
X = df[features]
y = df['log_price']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Scale the predictors
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit the linear regression model
model = LinearRegression()
model.fit(X_train_scaled, y_train)

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

# Evaluate the model performance
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"MAE: {mae}, MSE: {mse}, R2: {r2}")
```

```
MAE: 0.26327816040957763, MSE: 0.10870258064816198, R2: 0.5327746405197
726
```

In [23]:
```
# Create a DataFrame to compare true values and predicted values
df = pd.DataFrame({"true":y_test,"pred":y_pred})
df.head()
```

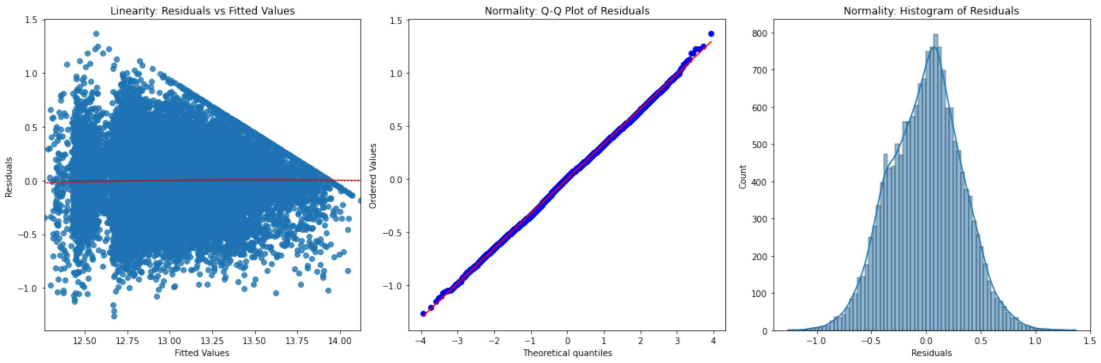
Out[23]:

	true	pred
10869	13.235603	12.880685
13410	13.487006	13.580520
17212	13.171154	12.852520
16399	13.541074	13.687352
6878	13.204865	12.814871

Model 5 Diagnosis:

In [24]:

```
# Check on conformity with Assumptions of Linear Regression

# Residuals vs. Fitted values plot
fitted_values = model.predict(X_train_scaled)
residuals = y_train - fitted_values

# Plot Residuals vs Fitted Values, Q-Q Plot of Residuals, and Histogram
fig, axs = plt.subplots(1, 3, figsize=(18, 6))

# Residuals vs Fitted Values (Linearity Assumption)
sns.residplot(x=fitted_values, y=residuals, lowess=True, line_kws={'colo
axs[0].set_xlabel('Fitted Values')
axs[0].set_ylabel('Residuals')
axs[0].set_title('Linearity: Residuals vs Fitted Values')

# Q-Q Plot of Residuals (Normality Assumption)
stats.probplot(residuals, dist="norm", plot=axs[1])
axs[1].get_lines()[1].set_color('red')
axs[1].set_title('Normality: Q-Q Plot of Residuals')

# Histogram of Residuals (Normality Assumption)
sns.histplot(x=residuals, kde=True, ax=axs[2])
axs[2].set_xlabel('Residuals')
axs[2].set_title('Normality: Histogram of Residuals')

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



Model 5 Diagnosis Observations:

Linearity: The plot shows a more randomly scattered pattern with no clear systematic structure, which suggests that the linearity assumption is better satisfied compared to the previous plot that exhibited a clear pattern.

Normality: The points in the Q-Q plot closely follow the red line, with slight deviations at the tails. This indicates that the residuals are approximately normally distributed, and the normality assumption is better met than in the previous plot, which showed larger deviations.The histogram is bell-shaped and shows less skewness and kurtosis compared to the previous histogram. This suggests a better approximation of normality for the residuals.

In [25]:
```
# Transform predictions back to the original scale to interpret the resu
y_pred_original_scale = np.exp(y_pred)

# Evaluate the performance on the original scale
y_test_original_scale = np.exp(y_test)
mae_original_scale = mean_absolute_error(y_test_original_scale, y_pred_o
mse_original_scale = mean_squared_error(y_test_original_scale, y_pred_or
r2_original_scale = r2_score(y_test_original_scale, y_pred_original_scal

print(f"Original Scale MAE: {mae_original_scale}, MSE: {mse_original_sca
```

```
Original Scale MAE: 123967.2465110259, MSE: 26571843532.431545, R2: 0.5
684781371801043
```

Model 5 with sqft_living, grade, sqft_above, bathrooms and bedrooms features; removed outliers; log transformed price and scaled predictors return the best linear regression metrics yet, that is, lowest MAE & MSE and highest R2.

In [26]:
```
# Create a DataFrame to compare true values and predicted values
df = pd.DataFrame({"true":y_test_original_scale,"pred":y_pred_original_s
df.head()
```

Out[26]:

	true	pred
10869	559950.0	392654.511487
13410	720000.0	790578.319941
17212	525000.0	381749.486285
16399	760000.0	879714.018504
6878	543000.0	367644.257636

Conclusion

The iterative modeling process culminated in the development of a robust predictive model (Model 5) that estimates the increase in home value based on sqft_living, grade, sqft_above, bathrooms, and bedrooms as key features. The models R2 of 0.57 on the test set indictates that 57% of the variance in housing prices can be explained by these key features. This strong explanatory power between key features and home prices suggest that strategic renovations of the features can substantially increase a property's value.

Recommendations:

1. **Focus on High-Impact Renovations:** The agency should advise homeowners to prioritize renovations that significantly increase the living area, improve the quality of construction, and add more functional spaces such as bathrooms and bedrooms. These improvements are shown to have the most substantial impact on home value.

2. **Use the Predictive Model for Client Consultations:** Incorporate the predictive model into client consultations to provide data-driven estimates of potential home value increases from specific renovations. This will help homeowners make informed decisions about their renovation projects.
3. **Market Insights:** Utilize the insights from the model to identify market trends and property features that are most desirable to buyers. This can help the agency better position properties in the market and tailor marketing strategies to highlight key selling points.

By leveraging the predictive capabilities of the model, the agency can offer precise, actionable advice to homeowners, helping them maximize their return on investment from renovation projects. This not only enhances client satisfaction but also strengthens the