Task 1

Regression

Importing Nicessary Libraries

In [1]:
```
# Importing all libraries that would be needed throughout the experiment
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.linear_model import Ridge
```

In [2]:
```
# Importing the Houseprice data
data = pd.read_csv("/content/drive/MyDrive/Regression,clustering,ANNproject/Houseprice_data.csv")
```

Data Inspection/ Cleaning

In [3]:
```
data.head()
```

Out[3]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	yr_built
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1955
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1951
2	180000.0	2	1.00	770	10000	1.0	0	0	3	6	770	0	1933
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	7	1050	910	1965
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	8	1680	0	1987

In [4]:
```
data.info(), data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   price          21613 non-null  float64
 1   bedrooms       21613 non-null  int64  
 2   bathrooms      21613 non-null  float64
 3   sqft_living    21613 non-null  int64  
 4   sqft_lot       21613 non-null  int64  
 5   floors         21613 non-null  float64
 6   waterfront     21613 non-null  int64  
 7   view           21613 non-null  int64  
 8   condition      21613 non-null  int64  
 9   grade          21613 non-null  int64  
 10  sqft_above     21613 non-null  int64  
 11  sqft_basement  21613 non-null  int64  
 12  yr_built       21613 non-null  int64  
 13  yr_renovated   21613 non-null  int64  
 14  zipcode        21613 non-null  int64  
 15  lat            21613 non-null  float64
 16  long           21613 non-null  float64
 17  sqft_living15  21613 non-null  int64  
 18  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(14)
memory usage: 3.1 MB
```

Out[4]:
```
(None,
              price      bedrooms     bathrooms   sqft_living      sqft_lot  \
count  2.161300e+04  21613.000000  21613.000000  21613.000000  2.161300e+04
mean   5.401822e+05      3.370842      2.114757   2079.899736  1.510697e+04
std    3.673622e+05      0.930062      0.770163    918.440897  4.142051e+04
min    7.500000e+04      0.000000      0.000000    290.000000  5.200000e+02
25%    3.219500e+05      3.000000      1.750000   1427.000000  5.040000e+03
50%    4.500000e+05      3.000000      2.250000   1910.000000  7.618000e+03
75%    6.450000e+05      4.000000      2.500000   2550.000000  1.068800e+04
max    7.700000e+06     33.000000      8.000000  13540.000000  1.651359e+06

             floors    waterfront          view     condition         grade  \
count  21613.000000  21613.000000  21613.000000  21613.000000  21613.000000
mean       1.494309      0.007542      0.234303      3.409430      7.656873
std        0.539989      0.086517      0.766318      0.650743      1.175459
min        1.000000      0.000000      0.000000      1.000000      1.000000
25%        1.000000      0.000000      0.000000      3.000000      7.000000
50%        1.500000      0.000000      0.000000      3.000000      7.000000
75%        2.000000      0.000000      0.000000      4.000000      8.000000
max        3.500000      1.000000      4.000000      5.000000     13.000000

         sqft_above  sqft_basement      yr_built  yr_renovated       zipcode  \
count  21613.000000   21613.000000  21613.000000  21613.000000  21613.000000
mean    1788.390691     291.509045   1971.005136     84.402258  98077.939805
std      828.090978     442.575043     29.373411    401.679240     53.505026
min      290.000000       0.000000   1900.000000      0.000000  98001.000000
25%     1190.000000       0.000000   1951.000000      0.000000  98033.000000
50%     1560.000000       0.000000   1975.000000      0.000000  98065.000000
75%     2210.000000     560.000000   1997.000000      0.000000  98118.000000
max     9410.000000    4820.000000   2015.000000   2015.000000  98199.000000

                lat          long  sqft_living15     sqft_lot15
count  21613.000000  21613.000000   21613.000000   21613.000000
mean      47.560053   -122.213896    1986.552492   12768.455652
std        0.138564      0.140828     685.391304   27304.179631
min       47.155900   -122.519000     399.000000     651.000000
25%       47.471000   -122.328000    1490.000000    5100.000000
50%       47.571800   -122.230000    1840.000000    7620.000000
75%       47.678000   -122.125000    2360.000000   10083.000000
max       47.777600   -121.315000    6210.000000  871200.000000  )
```

In [5]:
```
# Checking for null values
data.isnull().sum()
```

Out[5]:
```
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
```

In [6]:
```
# After checking for duplicates, there was 5 duplicates which was then dropped
data.duplicated().sum()
data = data.drop_duplicates()
```

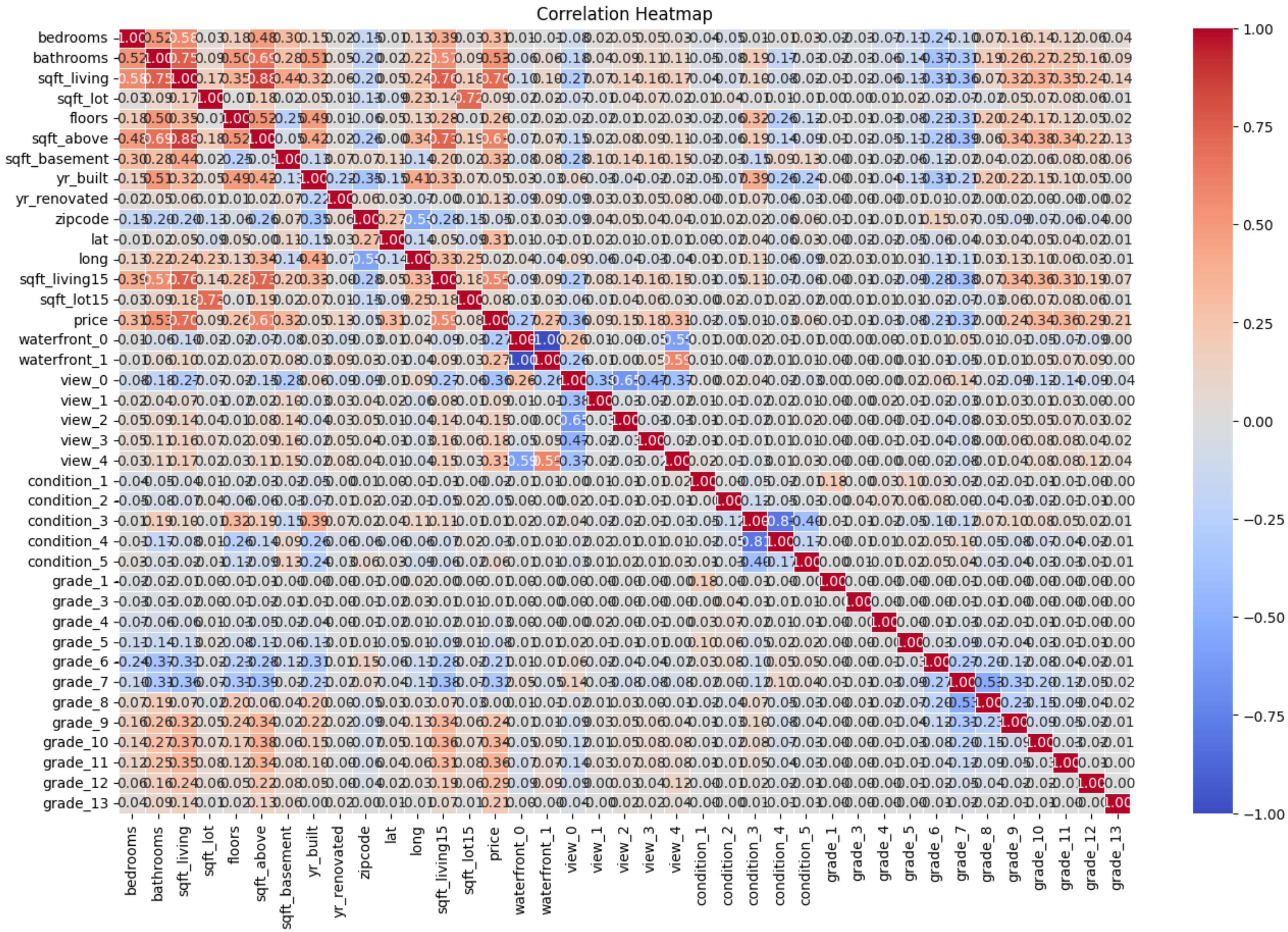
Data Preprocessing

In [7]:
```
selected_features = ['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'view', 'conc
data = data[selected_features]
```

In [8]:
```
# Turining specific columns to categorical  columns
data = pd.get_dummies(data, columns=['waterfront', 'view', 'condition', 'grade'], prefix=['waterfront', 'view'
```

In [9]:
```
# Scaling the data
scaler = StandardScaler()
data[['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'sqft_above', 'sqft_basement', 'yr_built',
```

In [10]:
```
# Calculating the correlation matrix
correlation_matrix = data.corr()

# Plotting the correlation heatmap
plt.figure(figsize=(16, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
bedrooms
bathrooms
sqft_living
sqft_lot
floors
sqft_above
sqft_basement
yr_built
yr_renovated
zipcode
lat
long
sqft_living15
sqft_lot15
price
waterfront_0
waterfront_1
view_0
view_1
view_2
view_3
view_4
condition_1
condition_2
condition_3
condition_4
condition_5
grade_1
grade_3
grade_4
grade_5
grade_6
grade_7
grade_8
grade_9
grade_10
grade_11
grade_12
grade_13
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

In [11]:

```
# Extracting the independent variables
X = data.drop('price', axis=1)

# Calculating VIF for each variable
vif_data = pd.DataFrame()
vif_data["Variable"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display the VIF values
print(vif_data)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide
by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

	Variable	VIF
0	bedrooms	1.727662
1	bathrooms	3.396057
2	sqft_living	inf
3	sqft_lot	2.108188
4	floors	2.082902
5	sqft_above	inf
6	sqft_basement	inf
7	yr_built	2.551770
8	yr_renovated	1.165371
9	zipcode	1.684440
10	lat	1.187669
11	long	1.838005
12	sqft_living15	3.047669
13	sqft_lot15	2.137032
14	waterfront_0	inf
15	waterfront_1	inf
16	view_0	inf
17	view_1	inf
18	view_2	inf
19	view_3	inf
20	view_4	inf
21	condition_1	inf
22	condition_2	inf
23	condition_3	inf
24	condition_4	inf
25	condition_5	inf
26	grade_1	inf
27	grade_3	inf
28	grade_4	inf
29	grade_5	inf
30	grade_6	inf
31	grade_7	inf
32	grade_8	inf
33	grade_9	inf
34	grade_10	inf
35	grade_11	inf
36	grade_12	inf
37	grade_13	inf

In [12]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(data_frame):
    # Calculate VIF for each variable
    vif_data = pd.DataFrame()
    vif_data["Variable"] = data_frame.columns
    vif_data["VIF"] = [variance_inflation_factor(data_frame.values, i) for i in range(data_frame.shape[1])]
    return vif_data

# Your original dataframe
X = data.drop('price', axis=1)

# Loop to iteratively drop variables with high VIF
while True:
    vif_data = calculate_vif(X)
    max_vif = vif_data['VIF'].max()

    if max_vif > 5:
        # Drop the variable with the highest VIF
        variable_to_drop = vif_data[vif_data['VIF'] == max_vif]['Variable'].values[0]
        X = X.drop(variable_to_drop, axis=1)
    else:
        break

# Displaying the final dataframe with reduced multicollinearity
print(X)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide
by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide
by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide
by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide
by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
```

```
       bedrooms  bathrooms  sqft_lot    floors  sqft_basement  yr_built  \
0     -0.398812  -1.447297 -0.228361 -0.915258      -0.658704 -0.544756
1     -0.398812   0.175615 -0.189929  0.936944       0.245041 -0.680946
2     -1.473987  -1.447297 -0.123349 -0.915258      -0.658704 -1.293800
3      0.676363   1.149362 -0.244052 -0.915258       1.397317 -0.204281
4     -0.398812  -0.148967 -0.169699 -0.915258      -0.658704  0.544762
...         ...        ...       ...       ...            ...       ...
21608 -0.398812   0.500197 -0.337452  2.789147      -0.658704  1.293806
21609  0.676363   0.500197 -0.224426  0.936944      -0.658704  1.464043
21610 -1.473987  -1.771879 -0.332166  0.936944      -0.658704  1.293806
21611 -0.398812   0.500197 -0.307108  0.936944      -0.658704  1.123569
21612 -1.473987  -1.771879 -0.338780  0.936944      -0.658704  1.259758

       yr_renovated   zipcode       lat      long  ...  grade_3  grade_4  \
0         -0.210034  1.870034 -0.352515 -0.306089  ...        0        0
1          4.748775  0.879485  1.161469 -0.746346  ...        0        0
2         -0.210034 -0.933408  1.283425 -0.135667  ...        0        0
3         -0.210034  1.085070 -0.283238 -1.271814  ...        0        0
4         -0.210034 -0.073686  0.409528  1.199305  ...        0        0
...             ...       ...       ...       ...  ...      ...      ...
21608     -0.210034  0.468313  1.004875 -0.938070  ...        0        0
21609     -0.210034  1.271966 -0.356123 -1.051685  ...        0        0
21610     -0.210034  1.234587  0.247883 -0.604327  ...        0        0
21611     -0.210034 -0.952098 -0.184375  1.028883  ...        0        0
21612     -0.210034  1.234587  0.245718 -0.604327  ...        0        0

       grade_5  grade_6  grade_8  grade_9  grade_10  grade_11  grade_12  \
0            0        0        0        0         0         0         0
1            0        0        0        0         0         0         0
2            0        1        0        0         0         0         0
3            0        0        0        0         0         0         0
4            0        0        1        0         0         0         0
...        ...      ...      ...      ...       ...       ...       ...
21608        0        0        1        0         0         0         0
21609        0        0        1        0         0         0         0
21610        0        0        0        0         0         0         0
21611        0        0        1        0         0         0         0
21612        0        0        0        0         0         0         0

       grade_13
0             0
1             0
2             0
```

```
3            0
4            0
...        ...
21608        0
21609        0
21610        0
21611        0
21612        0

[21608 rows x 32 columns]
```

This are the Columns that was dropped sqft_living, sqft_above, sqft_basement, waterfront_0, waterfront_1, view_0, view_1, view_2, view_3, view_4, condition_1, condition_2, condition_3, condition_4, condition_5, grade_1, grade_3, grade_4, grade_5, grade_6, grade_7, grade_8, grade_9, grade_10, grade_11, grade_12, grade_13

These features were highly correlated with other features in the dataset, making them redundant and causing issues like infinite VIF values. Dropping them helps to address multicollinearity.

In [14]:
```
# Slitting to test and train data
y = data['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

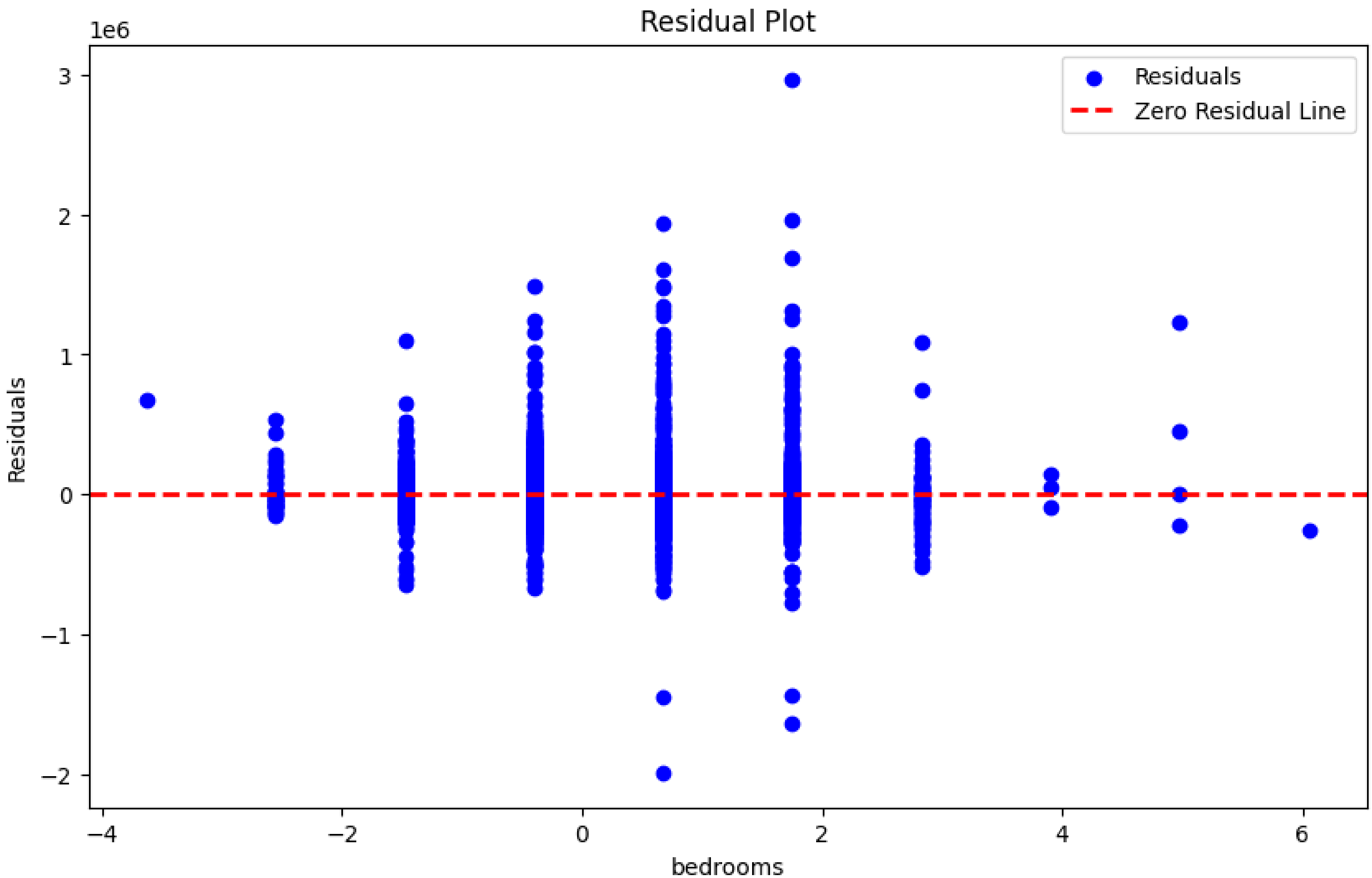
Training the Model

In [15]:
```
# Using Linear Regression Model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[15]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [17]:
```
# Making predictions on the test set
y_pred = model.predict(X_test)

# Plotting residuals against one input (bedrooms)
plt.figure(figsize=(10, 6))
plt.scatter(X_test['bedrooms'], y_test - y_pred, c='blue', marker='o', label='Residuals')
plt.axhline(y=0, color='red', linestyle='--', linewidth=2, label='Zero Residual Line')
plt.xlabel('bedrooms')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend()
plt.show()
```

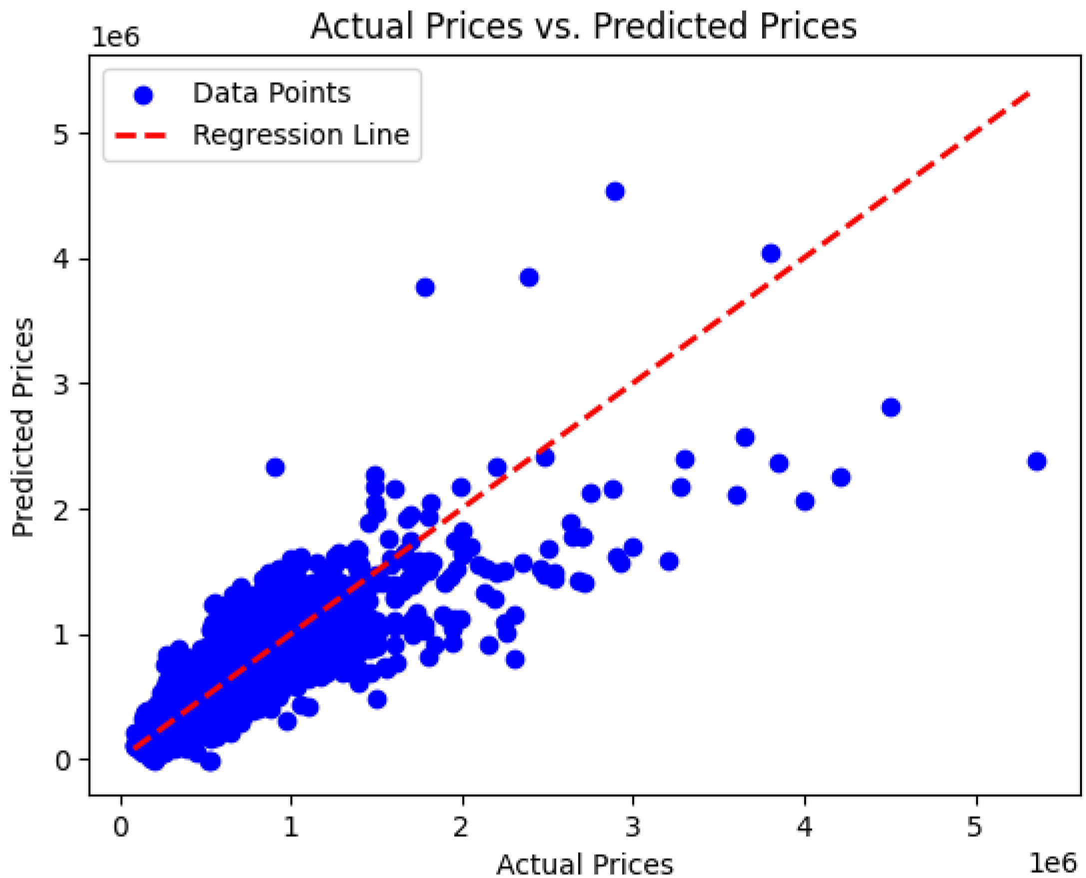


In [18]:
```
# Measuring the effectiveness of the model by using Mse and R2 score
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error: ", mse)
print("R-squared (R2) Score: ", r2)
```

```
Mean Squared Error:  41684779886.43211
R-squared (R2) Score:  0.7075361790482855
```

Visualising the predicted model

In [19]:
```
# Creating a scatter plot of actual vs. predicted prices and showing the regression line
plt.scatter(y_test, y_pred, color='blue', label='Data Points')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', linewidth=2, lab

plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.legend()

plt.show()
```



Improving the Linear Regression algorithm

Using GridSearchCv to get the best parameters

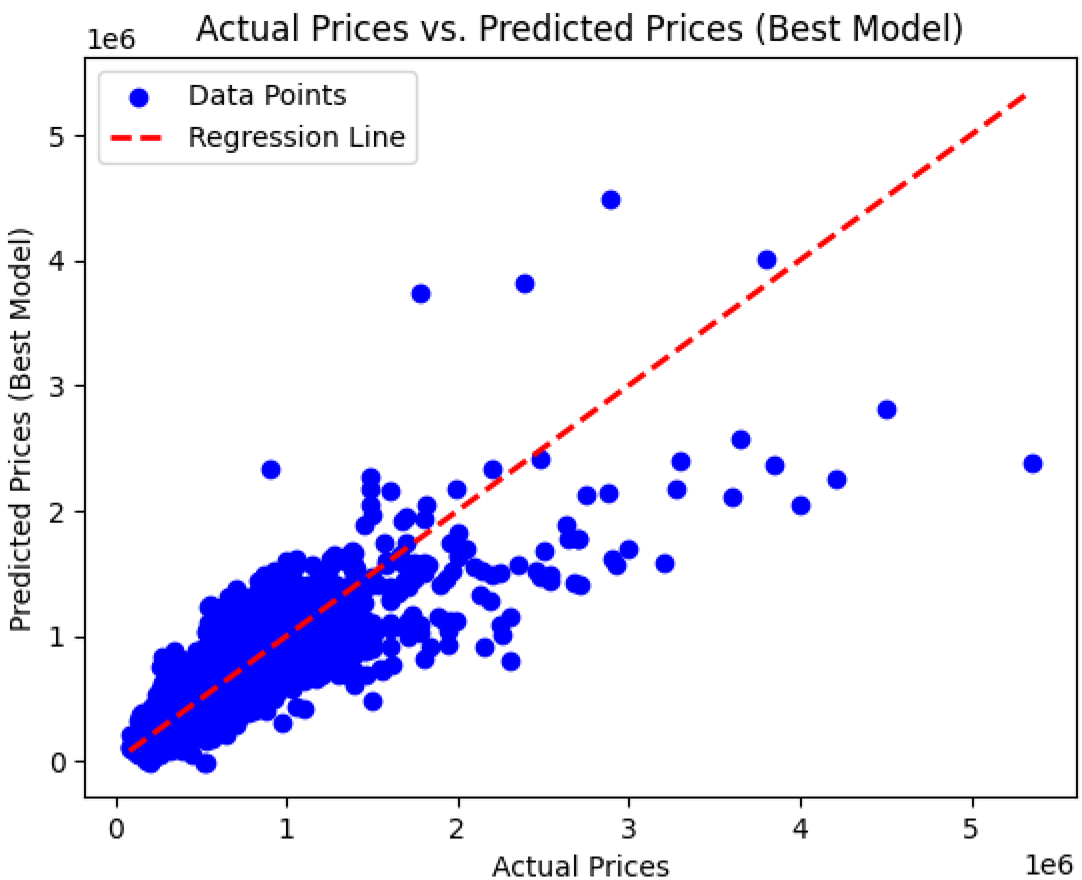
In [22]:
```
# Defining the parameter grid to search over
param_grid = {
    'fit_intercept': [True, False],
    'positive': [True, False],
    'alpha': [0.1, 0.5, 1.0, 5.0]  #regularization strength values
}

# Creating a GridSearchCV object for Ridge regression
ridge_grid_search = GridSearchCV(
    estimator=Ridge(),
    param_grid=param_grid,
    scoring='neg_mean_squared_error',
    cv=5
)
```

In [23]:
```
# Fitting the GridSearchCV object to your data
ridge_grid_search.fit(X_train, y_train)

# Getting the best parameters and the best estimator
best_params_ridge = ridge_grid_search.best_params_
best_ridge_model = ridge_grid_search.best_estimator_
```

In [25]:
```
# Using the best estimator for predictions
y_pred2 = best_ridge_model.predict(X_test)

# Measuring the effectiveness of the Improved model by using Mse and R2 score
mse = mean_squared_error(y_test, y_pred2)
r2 = r2_score(y_test, y_pred2)

print("Best Parameters: ", best_params_ridge)
print("Mean Squared Error (Best Model): ", mse)
print("R-squared (R2) Score (Best Model): ", r2)
```

```
Best Parameters:  {'alpha': 0.1, 'fit_intercept': True, 'positive': False}
Mean Squared Error (Best Model):  41602418451.733
R-squared (R2) Score (Best Model):  0.7081140336023168
```

In [27]:
```
# Creating a scatter plot of actual vs. predicted prices for the best parameter model and showing the regressi
plt.scatter(y_test, y_pred2, color='blue', label='Data Points')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', linewidth=2, lab

plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices (Best Model)")
plt.title("Actual Prices vs. Predicted Prices (Best Model)")
plt.legend()

plt.show()
```



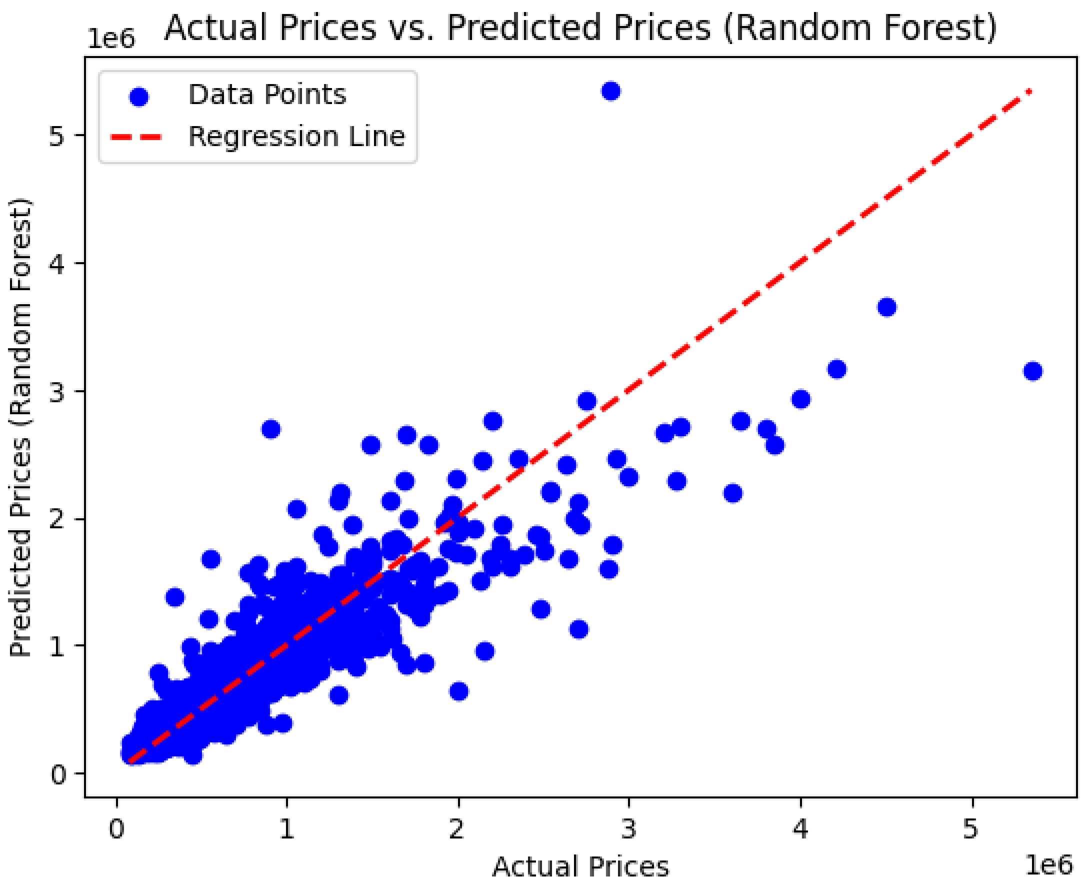
Exploring Advanced Regression models

Using RandomForest Regression

In [28]:
```
# Creating a Random Forest Regressor
rf_regressor = RandomForestRegressor(random_state=42)

rf_regressor.fit(X_train, y_train)
y_pred_rf = rf_regressor.predict(X_test)

# E# Measuring the effectiveness of the Random Forest model by using Mse and R2 score
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("Random Forest Regression - Mean Squared Error: ", mse_rf)
print("Random Forest Regression - R-squared (R2) Score: ", r2_rf)
```

```
Random Forest Regression - Mean Squared Error:  23741866400.706726
Random Forest Regression - R-squared (R2) Score:  0.833425125837455
```

In [29]:
```
# Create a scatter plot of actual vs. predicted prices and showing the regression line
plt.scatter(y_test, y_pred_rf, color='blue', label='Data Points')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--', linewidth=2, lab

plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices (Random Forest)")
plt.title("Actual Prices vs. Predicted Prices (Random Forest)")
plt.legend()

plt.show()
```



In [ ]:

```

```