```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,PolynomialFeatures
from sklearn.linear_model import LinearRegression
%matplotlib inline
```

```
df=pd.read_csv("/content/kc_house_data.csv")
```

```
df.head(5)
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	s
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
if 'id' in df.columns:
    df.drop(['id'], axis=1, inplace=True)
if 'Unnamed: 0' in df.columns:
    df.drop(['Unnamed: 0'], axis=1, inplace=True)

df.describe()
```

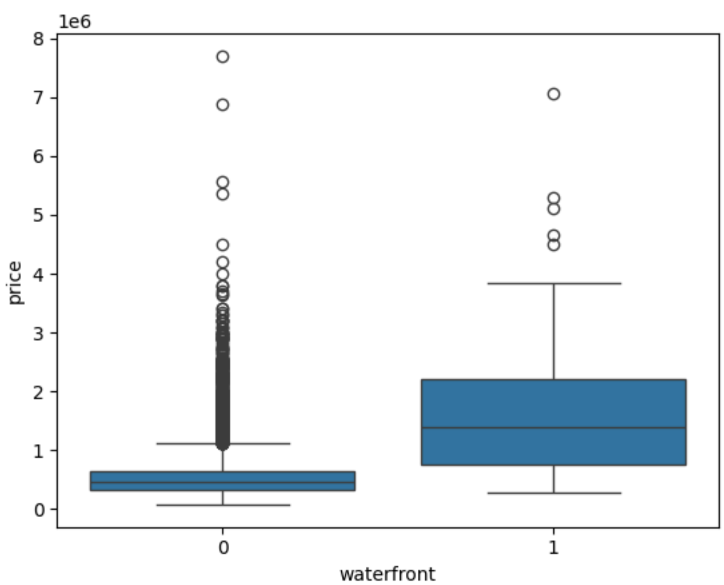
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	
count	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.000000	21613.0
mean	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	3.409430	7.6
std	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	0.650743	1.1
min	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	1.000000	1.0
25%	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	3.000000	7.0
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	3.000000	7.0
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	4.000000	8.0
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	5.000000	13.0

```
# prompt: Use the method value_counts to count the number of houses with unique floor values, and use the method to_frame() to convert it tc

# Count the number of houses with unique floor values and convert to a DataFrame
floors_counts = df['floors'].value_counts().to_frame()
floors_counts
```

floors	count
1.0	10680
2.0	8241
1.5	1910
3.0	613
2.5	161
3.5	8

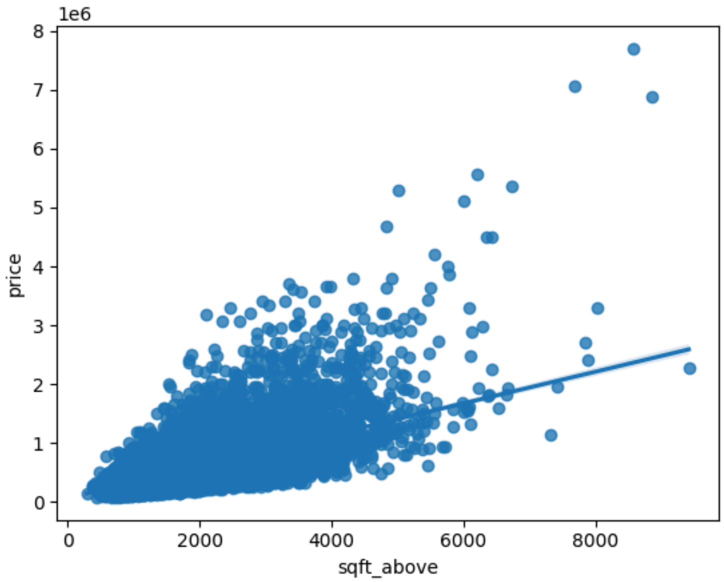
```
# prompt: Use the function boxplot in the seaborn library to produce a plot that can help determine whether houses with a waterfront view or 

sns.boxplot(x="waterfront", y="price", data=df)
```

<Axes: xlabel='waterfront', ylabel='price'>




```
# prompt: Use the function regplot in the seaborn library to determine if the feature sqft_above is negatively or positively correlated with

sns.regplot(x="sqft_above", y="price", data=df)
```

```
<Axes: xlabel='sqft_above', ylabel='price'>
```




```
# prompt: Fit a linear regression model to predict the price using the feature 'sqft_living', then calculate the R^2.

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Define features (X) and target (y)
X = df[['sqft_living']]
y = df['price']

# Split data 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)

# Initialize and fit the linear regression model
lm = LinearRegression()
lm.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lm.predict(X_test)

# Calculate R^2
r2 = r2_score(y_test, y_pred)

print(f"R-squared: {r2}")
```

```
R-squared: 0.49406905389089006
```

```
# prompt: Fit a linear regression model to predict the 'price' using the list of features:
# "floors"
# "waterfront"
# "lat"
# "bedrooms"
# "sqft_basement"
# "view"
# "bathrooms"
# "sqft_living15"
# "sqft_above"
# "grade"
# "sqft_living"

# Define features (X) and target (y)
features = ["floors", "waterfront", "lat", "bedrooms", "sqft_basement", "view", "bathrooms", "sqft_living15", "sqft_above", "grade", "sqft_l
X = df[features]
y = df['price']

# Split data 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)

# Initialize and fit the linear regression model
```

```
lm = LinearRegression()
lm.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lm.predict(X_test)

# Calculate R^2
r2 = r2_score(y_test, y_pred)

print(f"R-squared: {r2}")
```

```
R-squared: 0.6614781405487573
```

```
# prompt: Create a pipeline object that scales the data, performs a polynomial transform, and fits a linear regression model. Fit the object

Input =['floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basement' ,'view' ,'bathrooms','sqft_living15','sqft_above','grade','sqft_living']
# Create a pipeline object
Input=[ 'floors', 'waterfront','lat' ,'bedrooms' ,'sqft_basement' ,'view' ,'bathrooms','sqft_living15','sqft_above','grade','sqft_living']
pipe=Pipeline(steps=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include_bias=False)),('model',LinearRegression())])

# Fit the pipeline object using the features in the question above
pipe.fit(X_train,y_train)

# Make predictions on the test set
y_pred = pipe.predict(X_test)

# Calculate R^2
r2 = r2_score(y_test, y_pred)
print('The R-square is',r2)
```

```
The R-square is 0.7114140982349176
```

```
# prompt: Create and fit a Ridge regression object using the training data, setting the regularization parameter to 0.1, and calculate the R

from sklearn.linear_model import Ridge

# Initialize and fit the Ridge regression model
ridge = Ridge(alpha=0.1)  # alpha is the regularization parameter
ridge.fit(X_train, y_train)

# Make predictions on the test set
y_pred_ridge = ridge.predict(X_test)

# Calculate R^2
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"R-squared (Ridge Regression): {r2_ridge}")
```

```
R-squared (Ridge Regression): 0.6614734596866666
```

```
# prompt: Perform a  transformsecond-order polynomial on both the training data and testing data. Create and fit a Ridge regression object u

# Create polynomial features
pr = PolynomialFeatures(degree=2)
X_train_pr = pr.fit_transform(X_train)
X_test_pr = pr.fit_transform(X_test)

# Initialize and fit the Ridge regression model
ridge = Ridge(alpha=0.1)  # alpha is the regularization parameter
ridge.fit(X_train_pr, y_train)

# Make predictions on the test set
y_pred_ridge = ridge.predict(X_test_pr)

# Calculate R^2
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"R-squared (Ridge Regression with Polynomial Features): {r2_ridge}")
```

```
R-squared (Ridge Regression with Polynomial Features): 0.7003486858533614
```

```
Start coding or generate with AI.
```