In [43]:
```
 # Surpress warnings:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [44]:
```
# We will require the following libraries for this project:
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
```

In [45]:
```
# We use the Pandas method read_csv() to load the data.

file_name='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clou
df = pd.read_csv(file_name)
```

In [46]:
```
# We use the method head to display the first
# 5 columns of the dataframe.

df.head()
```

Out[46]:

| | Unnamed: 0 | id | date | price | bedrooms | bathrooms | sqft_living |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 7129300520 | 20141013T000000 | 221900.0 | 3.0 | 1.00 | 1180 |
| **1** | 1 | 6414100192 | 20141209T000000 | 538000.0 | 3.0 | 2.25 | 2570 |
| **2** | 2 | 5631500400 | 20150225T000000 | 180000.0 | 2.0 | 1.00 | 770 |
| **3** | 3 | 2487200875 | 20141209T000000 | 604000.0 | 4.0 | 3.00 | 1960 |
| **4** | 4 | 1954400510 | 20150218T000000 | 510000.0 | 3.0 | 2.00 | 1680 |

5 rows × 22 columns

In [47]:
```
# We display the data types of each column using the function dtypes

df.dtypes
```

Out[47]:
```
Unnamed: 0          int64
id                  int64
date               object
price             float64
bedrooms          float64
bathrooms         float64
sqft_living         int64
sqft_lot            int64
floors            float64
waterfront          int64
view                int64
condition           int64
grade               int64
sqft_above          int64
sqft_basement       int64
yr_built            int64
yr_renovated        int64
zipcode             int64
lat               float64
long              float64
sqft_living15       int64
sqft_lot15          int64
dtype: object
```

In [48]:
```
# We use the method describe to obtain a statistical summary of the datafram

df.describe()
```

Out[48]:

| | Unnamed: 0 | id | price | bedrooms | bathrooms | sqft_livi |
|---|---|---|---|---|---|---|
| **count** | 21613.00000 | 2.161300e+04 | 2.161300e+04 | 21600.000000 | 21603.000000 | 21613.0000 |
| **mean** | 10806.00000 | 4.580302e+09 | 5.400881e+05 | 3.372870 | 2.115736 | 2079.8997 |
| **std** | 6239.28002 | 2.876566e+09 | 3.671272e+05 | 0.926657 | 0.768996 | 918.4408 |
| **min** | 0.00000 | 1.000102e+06 | 7.500000e+04 | 1.000000 | 0.500000 | 290.0000 |
| **25%** | 5403.00000 | 2.123049e+09 | 3.219500e+05 | 3.000000 | 1.750000 | 1427.0000 |
| **50%** | 10806.00000 | 3.904930e+09 | 4.500000e+05 | 3.000000 | 2.250000 | 1910.0000 |
| **75%** | 16209.00000 | 7.308900e+09 | 6.450000e+05 | 4.000000 | 2.500000 | 2550.0000 |
| **max** | 21612.00000 | 9.900000e+09 | 7.700000e+06 | 33.000000 | 8.000000 | 13540.0000 |

8 rows × 21 columns

In [49]:
```
# We drop the columns "id" and "Unnamed: 0" from axis 1
# using the method drop(),
# then we use the method describe() to obtain a statistical summary of the c

df.drop('id',axis=1, inplace = True)
df.drop('Unnamed: 0',axis=1, inplace = True)
df.describe()
```

Out[49]:

| | price | bedrooms | bathrooms | sqft_living | sqft_lot | floo |
|---|---|---|---|---|---|---|
| **count** | 2.161300e+04 | 21600.000000 | 21603.000000 | 21613.000000 | 2.161300e+04 | 21613.0000 |
| **mean** | 5.400881e+05 | 3.372870 | 2.115736 | 2079.899736 | 1.510697e+04 | 1.4943 |
| **std** | 3.671272e+05 | 0.926657 | 0.768996 | 918.440897 | 4.142051e+04 | 0.5399 |
| **min** | 7.500000e+04 | 1.000000 | 0.500000 | 290.000000 | 5.200000e+02 | 1.0000 |
| **25%** | 3.219500e+05 | 3.000000 | 1.750000 | 1427.000000 | 5.040000e+03 | 1.0000 |
| **50%** | 4.500000e+05 | 3.000000 | 2.250000 | 1910.000000 | 7.618000e+03 | 1.5000 |
| **75%** | 6.450000e+05 | 4.000000 | 2.500000 | 2550.000000 | 1.068800e+04 | 2.0000 |
| **max** | 7.700000e+06 | 33.000000 | 8.000000 | 13540.000000 | 1.651359e+06 | 3.5000 |

In [50]:
```
# We can see we have missing values for the columns
# bedrooms and  bathrooms

print("number of NaN values for the column bedrooms :", df['bedrooms'].isnul
print("number of NaN values for the column bathrooms :", df['bathrooms'].isn
```

```
number of NaN values for the column bedrooms : 13
number of NaN values for the column bathrooms : 10
```

In [51]:
```
# We can replace the missing values of the column 'bedrooms'
# with the mean of the column 'bedrooms' using the method replace()

mean=df['bedrooms'].mean()
df['bedrooms'].replace(np.nan,mean, inplace=True)
```

In [52]:
```
# We also replace the missing values of the column 'bathrooms'
# with the mean of the column 'bathrooms' using the method replace()

mean=df['bathrooms'].mean()
df['bathrooms'].replace(np.nan,mean, inplace=True)
```

In [53]:
```
print("number of NaN values for the column bedrooms :", df['bedrooms'].isnul
print("number of NaN values for the column bathrooms :", df['bathrooms'].isn
```

```
number of NaN values for the column bedrooms : 0
number of NaN values for the column bathrooms : 0
```

In [54]:
```
# We use the method value_counts to count the number of houses
# with unique floor values, use the method .to_frame()
# to convert it to a dataframe

df['floors'].value_counts().to_frame().reset_index()
```

Out[54]:

| | floors | count |
|---|---|---|
| 0 | 1.0 | 10680 |
| 1 | 2.0 | 8241 |
| 2 | 1.5 | 1910 |
| 3 | 3.0 | 613 |
| 4 | 2.5 | 161 |
| 5 | 3.5 | 8 |

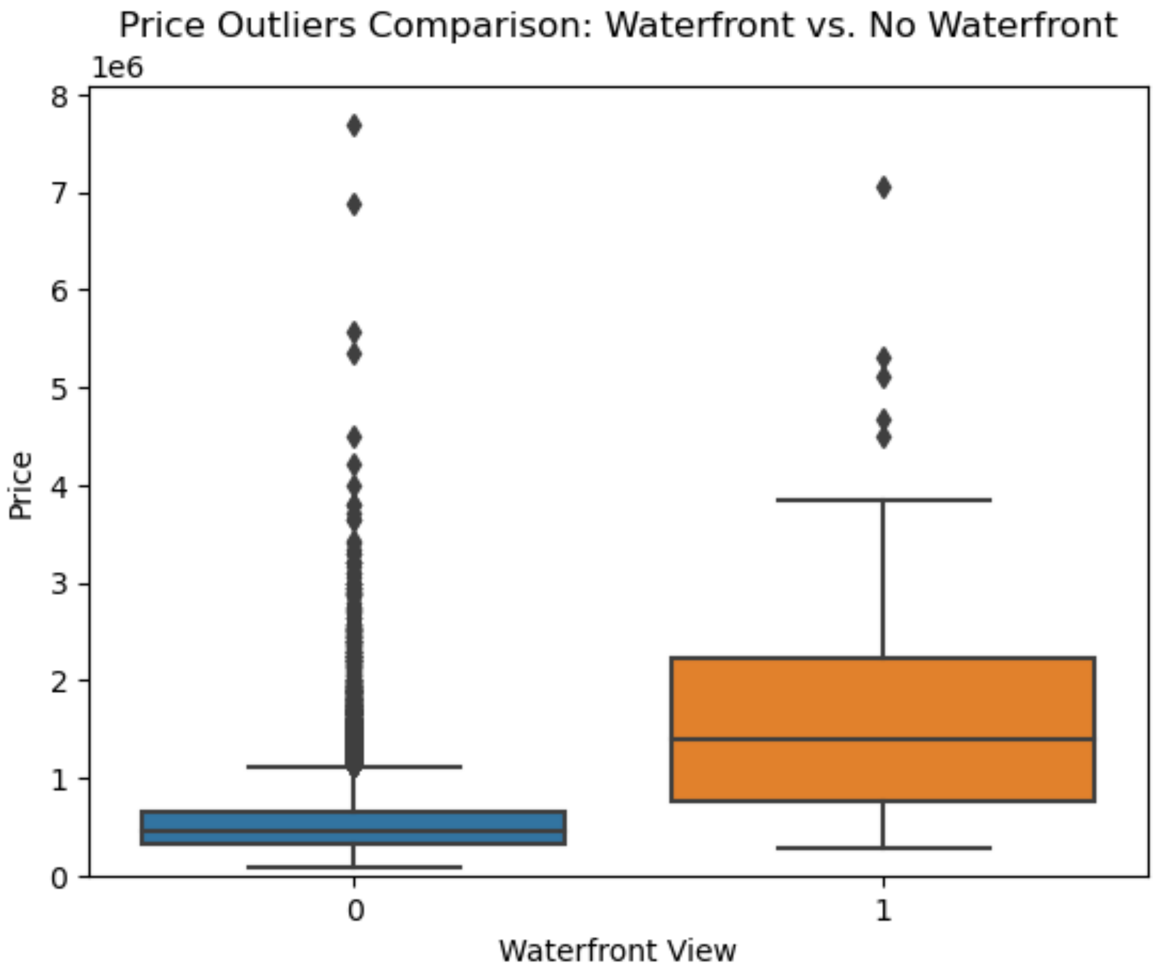
In [55]:
```
# We use the function boxplot in the seaborn library
# to determine whether houses with a waterfront view
# or without a waterfront view have more price outliers.

# Create a box plot
sns.boxplot(x='waterfront', y='price', data=df)

# Add labels and title
plt.xlabel('Waterfront View')
plt.ylabel('Price')
plt.title('Price Outliers Comparison: Waterfront vs. No Waterfront')

# Show the plot
plt.ylim(0,)
```

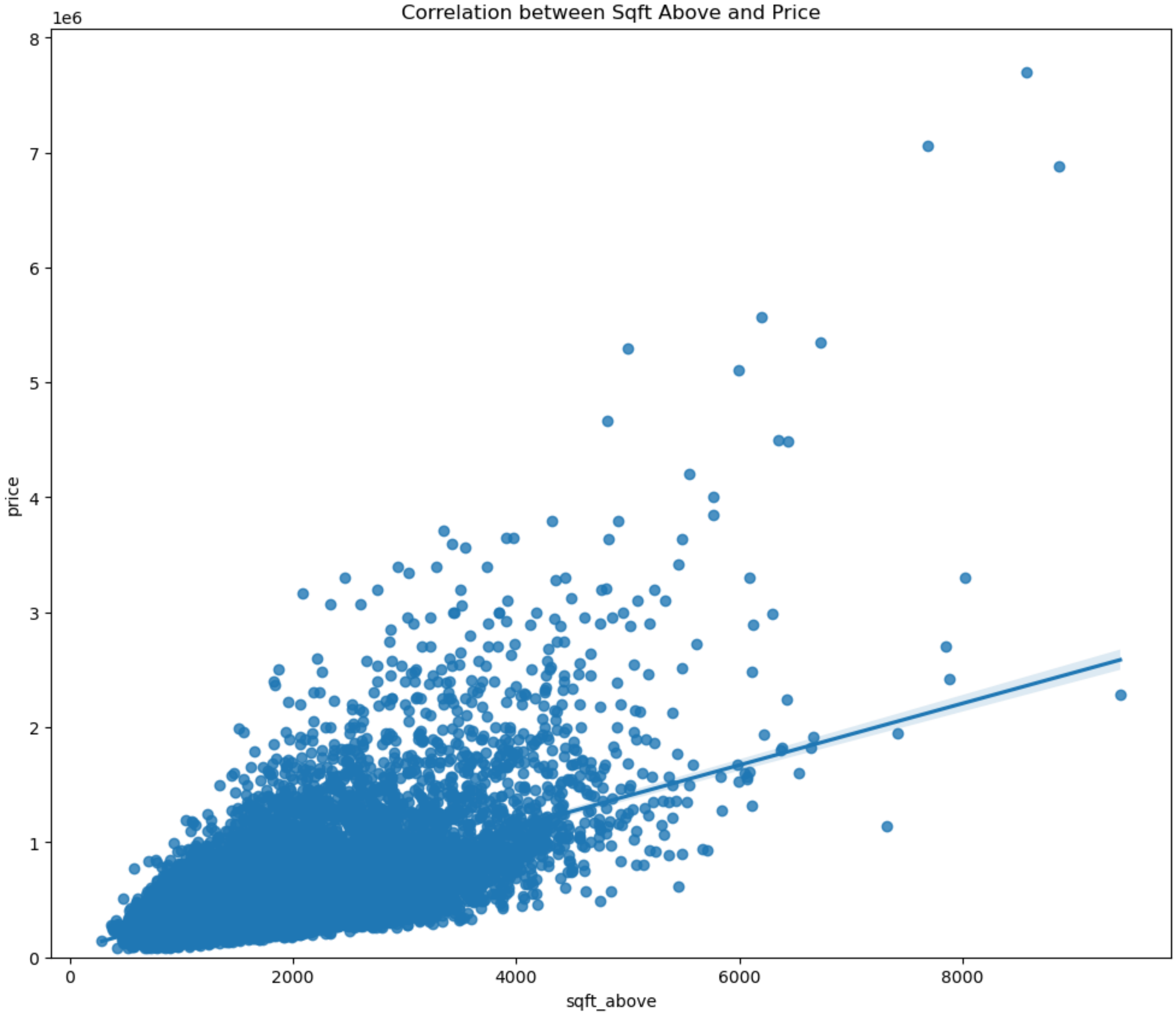
Out[55]:
```
(0.0, 8081250.0)
```




In [56]:
```
# We use the function regplot in the seaborn library
# to determine if the feature sqft_above is negatively
# or positively correlated with price.

width = 12
height = 10
plt.figure(figsize=(width, height))
sns.regplot(x="sqft_above", y="price", data=df)
plt.title('Correlation between Sqft Above and Price')
plt.ylim(0,)
```

Out[56]:
```
(0.0, 8081250.0)
```




In [59]:
```
#Model Development

#We can Fit a linear regression model
#using the longitude feature 'long' and
#caculate the R^2.

X = df[['long']]
Y = df['price']
lm = LinearRegression()
lm.fit(X,Y)
lm.score(X, Y)
```

Out[59]:
```
0.00046769430149007363
```

In [60]:
```
#Fit a linear regression model to predict the 'price'
#using the feature 'sqft_living'.

X = df[['sqft_living']]
Y = df['price']
lm.fit(X,Y)
Yhat=lm.predict(X)
Yhat[0:5]
```

Out[60]:
```
array([287555.06702451, 677621.82640197, 172499.40418656, 506441.44998452,
       427866.85097324])
```

In [61]:
```
#calculate the R^2.

print('The R-square is: ', lm.score(X, Y))
```

```
The R-square is:  0.4928532179037931
```

In [62]:
```
features =["floors", "waterfront","lat" ,"bedrooms" ,
"sqft_basement" ,"view" ,"bathrooms","sqft_living15","sqft_above","grade","s
```

In [63]:
```
X= df[features]
lm1 = LinearRegression()
lm1.fit(X,Y)

r2 =lm1.score(X,Y)
print('The R-square is: ', r2)
```

```
The R-square is:  0.6576000197691028
```

In [64]:
```
Input=[('scale',StandardScaler()),('polynomial', PolynomialFeatures(include_
```

In [65]:
```
pipe=Pipeline(Input)
Z = df[features]
Z = Z.astype(float)
pipe.fit(Z,Y)
ypipe = pipe.predict(Z)

from sklearn.metrics import r2_score
print('coefficient of determination:' , r2_score(Y,ypipe))
```

```
coefficient of determination: 0.7510104260853295
```

In [66]:
```
from sklearn.linear_model import Ridge
```

In [67]:
```
#Model Evaluation and Refinement
#Import the necessary modules:

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
print("done")
```

```
done
```

In [68]:
```
#We will split the data into training and testing sets:

features =["floors", "waterfront","lat" ,"bedrooms" ,"sqft_basement" ,"view"
X = df[features]
Y = df['price']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, ra

print("number of test samples:", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples: 3242
number of training samples: 18371
```

In [69]:
```
#Create and fit a Ridge regression object using the training data,
#set the regularization parameter to 0.1, and calculate the R^2
# using the test data.

RigeModel = Ridge(alpha=0.1)
RigeModel.fit(x_train, y_train)
RigeModel.score(x_test, y_test)
```

Out[69]:
```
0.6478759163939118
```

In [70]:
```
#Perform a second order polynomial transform on
#both the training data and testing data.
#Create and fit a Ridge regression object
#using the training data,
#set the regularisation parameter to 0.1,
#and calculate the R^2 utilising the test data
#provided.
```

In [71]:
```
pr=PolynomialFeatures(degree=2)
x_train=pr.fit_transform(x_train)
x_test=pr.fit_transform(x_test)

RigeModel=Ridge(alpha=0.1)
RigeModel.fit(x_train, y_train)
yhat = RigeModel.predict(x_test)
yhat
RigeModel.score(x_test, y_test)
```

Out[71]:
```
0.7002744260728231
```

In [ ]: