BUSINESS UNDERSTANDING

An association of home owners have approached seeking to know what features in a house could be remodelled to increase the house prices and what features they should add as they list their houses in the market.

DATA UNDERSTANDING

The data is collected from the King County House Sales dataset and the column description is as follows:

- `id` - Unique identifier for a house
- `date` - Date house was sold
- `price` - Sale price (prediction target)
- `bedrooms` - Number of bedrooms
- `bathrooms` - Number of bathrooms
- `sqft_living` - Square footage of living space in the home
- `sqft_lot` - Square footage of the lot
- `floors` - Number of floors (levels) in house
- `waterfront` - Whether the house is on a waterfront
    - Includes Duwamish, Elliott Bay, Puget Sound, Lake Union, Ship Canal, Lake Washington, Lake Sammamish, other lake, and river/slough waterfronts
- `view` - Quality of view from house
    - Includes views of Mt. Rainier, Olympics, Cascades, Territorial, Seattle Skyline, Puget Sound, Lake Washington, Lake Sammamish, small lake / river / creek, and other
- `condition` - How good the overall condition of the house is. Related to maintenance of house.
    - Rated 1-5 from poor to very good
- `grade` - Overall grade of the house. Related to the construction and design of the house.
    - Rated 1-13 from poor to excellent
- `sqft_above` - Square footage of house apart from basement
- `sqft_basement` - Square footage of the basement
- `yr_built` - Year when house was built
- `yr_renovated` - Year when house was renovated
- `zipcode` - ZIP Code used by the United States Postal Service
- `lat` - Latitude coordinate

- `long` - Longitude coordinate
- `sqft_living15` - The square footage of interior housing living space for the nearest 15 neighbors
- `sqft_lot15` - The square footage of the land lots of the nearest 15 neighbors

Data Preprocessing

This involves checking the data, renaming some values, cleaning it and handling any outliers

Import all necessary libraries

In [352]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import math
import scipy.stats as stats
import statsmodels.api as sm
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

Retrieve data from the dataset and preview the data

In [353]:
```
houses = pd.read_csv("data/kc_house_data.csv")
houses.head()
```

Out[353]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	NONE	...	7 Average	1180	
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	NONE	...	7 Average	2170	
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	NONE	...	6 Low Average	770	
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	NONE	...	7 Average	1050	
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	NONE	...	8 Good	1680	

5 rows × 21 columns

Get the infomation on the shape of data and columns of the data

In [354]:
```
houses.shape
```

Out[354]:
```
(21597, 21)
```

In [355]:
```
houses.info()
```

```
<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  object 
 9   view           21534 non-null  object 
 10  condition      21597 non-null  object 
 11  grade          21597 non-null  object 
 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(6), int64(9), object(6)
memory usage: 3.5+ MB
```

In [356]:
```
# Check for missing data
houses.isna().sum()
```

Out[356]:
```
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
```

Fill the missing values with modes of their respective columns as they are very few missing values

In [357]:
```
houses.waterfront.fillna(houses["waterfront"].mode().max(),inplace=True)
houses.waterfront.value_counts()
```

Out[357]:
```
NO     21451
YES      146
Name: waterfront, dtype: int64
```

In [358]:
```
houses.view.fillna(houses["view"].mode().max(),inplace=True)
houses.view.value_counts()
```

Out[358]:
```
NONE         19485
AVERAGE        957
GOOD           508
FAIR           330
EXCELLENT      317
Name: view, dtype: int64
```

In [359]:
```
houses.yr_renovated.fillna(houses["yr_renovated"].mode().max(),inplace=True)
houses.yr_renovated.value_counts()
```

Out[359]:
```
0.0       20853
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

Check for missing values

In [360]:
```
houses.isna().sum()
```

Out[360]:
```
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
```

Check individual columns for any irregular data

In [361]:
```
for k,v in houses.items():
    print(f"For {k} the value counts are:\n {houses[k].value_counts()}")
```

```
For id the value counts are:
 795000620     3
1825069031    2
2019200220    2
7129304540    2
1781500435    2
             ..
7812801125    1
4364700875    1
3021059276    1
880000205     1
1777500160    1
Name: id, Length: 21420, dtype: int64
For date the value counts are:
 6/23/2014    142
6/25/2014    131
6/26/2014    131
7/8/2014     127
4/27/2015    126
```

In [362]:
```
# Replace ? in sqft_basement with the mode
houses.sqft_basement.replace("?",0.0,inplace=True)
houses.sqft_basement.value_counts()
```

Out[362]:
```
0.0       12826
0.0         454
600.0       217
500.0       209
700.0       208
          ...
1275.0        1
1913.0        1
225.0         1
2570.0        1
1281.0        1
Name: sqft_basement, Length: 304, dtype: int64
```

In [363]:
```
# Replace the strings in grade with the integers
houses["grade"] = houses.grade.apply(lambda x: x.split()[0])
```

In [364]:
```
# Change the waterfall column to numbers on a scale No - 0 and Yes - 1
houses.waterfront.replace({"NO":0,"YES":1}, inplace=True)
houses.waterfront.value_counts()
```

Out[364]:
```
0    21451
1      146
Name: waterfront, dtype: int64
```

In [365]:
```
# Change the scale quality in condition from 0 - 4
scale_cond = {
    "Poor": 0,
    "Fair": 1,
    "Average": 2,
    "Good": 3,
    "Very Good": 4
}
houses.condition.replace(scale_cond,inplace=True)
houses.condition.value_counts()
```

Out[365]:
```
2    14020
3     5677
4     1701
1      170
0       29
Name: condition, dtype: int64
```

In [366]:
```
# Change the scale quality in view from 0 - 4
scale_view = {
    "NONE": 0,
    "FAIR": 1,
    "AVERAGE": 2,
    "GOOD": 3,
    "EXCELLENT": 4
}
houses.view.replace(scale_view,inplace=True)
houses.view.value_counts()
```

Out[366]:
```
0    19485
2      957
3      508
1      330
4      317
Name: view, dtype: int64
```

In [367]:
```
houses.drop((houses[houses['grade'] == 13].index) | (houses[houses['grade'] == 3].index), inplace = True)
houses.grade.value_counts()
```

Out[367]:
```
7     8974
8     6065
9     2615
6     2038
10    1134
11     399
5      242
12      89
4       27
13      13
3        1
Name: grade, dtype: int64
```

Change some of the data types of specific columns

In [368]:
```
# Convert date column from object to datetime format
houses.date = pd.to_datetime(houses.date)
```

In [369]:
```
# Convert sqft_basement to an integer format
houses["sqft_basement"] = houses.sqft_basement.astype(float)
```

In [370]:
```
houses.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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  datetime64[ns]
 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     21597 non-null  int64
 9   view           21597 non-null  int64
 10  condition      21597 non-null  int64
 11  grade          21597 non-null  object
 12  sqft_above     21597 non-null  int64
 13  sqft_basement  21597 non-null  float64
 14  yr_built       21597 non-null  int64
 15  yr_renovated   21597 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: datetime64[ns](1), float64(7), int64(12), object(1)
memory usage: 3.6+ MB
```

In [371]:
```
# Save to csv
houses.to_csv("data/cleaned_data.csv",index=False)
```

Feature Exploration

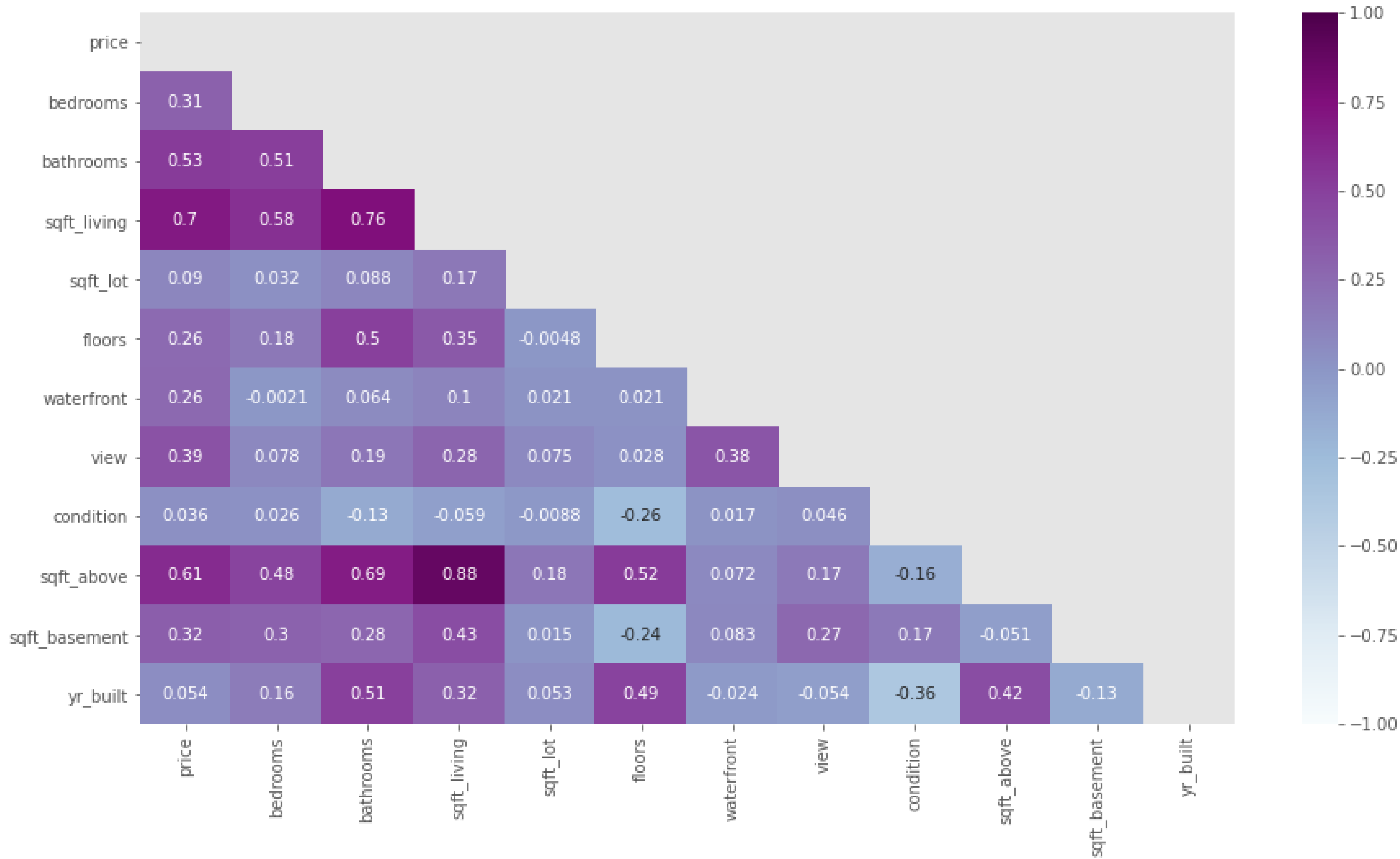
In [372]:
```
# drop unwanted columns
houses.drop(["id","zipcode","long","lat","sqft_living15","sqft_lot15","yr_renovated","date"],axis=1,inplace=True
```

In [373]:
```
houses.describe()
```

Out[373]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.000000	21597.000000	21597.000000	2
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	0.006760	0.233181	2.409825	
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	0.081944	0.764673	0.650546	
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.000000	0.000000	0.000000	
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.000000	0.000000	2.000000	
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	2.000000	
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	0.000000	0.000000	3.000000	
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	4.000000	

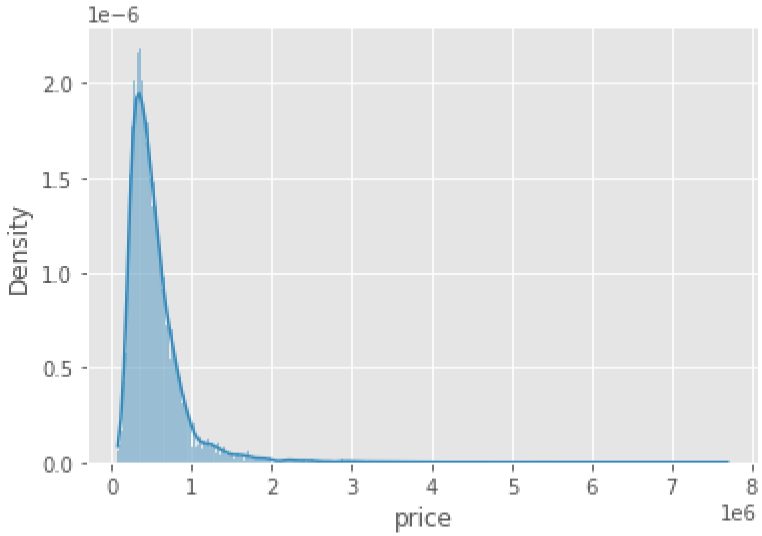
In [374]:
```
# Display the correlation in a heatmap
fig,ax  = plt.subplots(figsize=(15,8))
mask = np.triu(np.ones_like(houses.corr(), dtype=np.bool))
sns.heatmap(data=houses.corr(),center=0,vmin=-1,vmax=1,annot=True,mask=mask,cmap=sns.color_palette("BuPu", as_cm
plt.savefig("images/price heatmap correlation");
```



In [375]:
```
# Correlation between other columns and price
houses.corr()["price"]
```

Out[375]:
```
price            1.000000
bedrooms         0.308787
bathrooms        0.525906
sqft_living      0.701917
sqft_lot         0.089876
floors           0.256804
waterfront       0.264306
view             0.393497
condition        0.036056
sqft_above       0.605368
sqft_basement    0.321108
yr_built         0.053953
Name: price, dtype: float64
```

In [376]:
```
# Display a graph showing the price distribution
sns.histplot(data=houses, x="price",stat="density",kde=True)
plt.savefig("images/price density distribution");
```

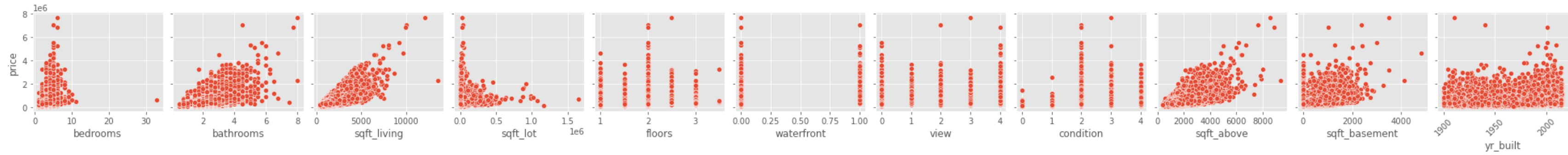


Identifying variables

- Continuous numeric variables
- Discrete numeric variables
- String categorical variables
- Discrete categorical variables

In [377]:
```
num_cols = houses.select_dtypes("number")
```

In [378]:
```
sns.pairplot(data = houses,y_vars="price", x_vars=num_cols.columns.drop("price"),diag_kind = None)
plt.xticks(rotation=45)
plt.savefig("images/scatter plot distribution of numeric colums and price");
```



From the above plot we can see that the types of variables are:

- continuous numeric variables like sqft_living,sqft_lot,sqft_above,sqft_basement
- discrete numeric variables like bedrooms, bathrooms, floors, waterfront, view, condition, grade, yr_bulit, yr_renovated
- There are no string categoricals as all our values are of numerical type
- Discrete categorical variable like grade,floors,waterfront,view,condition

In [379]:
```
houses.corr().abs()["price"]
```

Out[379]:
```
price            1.000000
bedrooms         0.308787
bathrooms        0.525906
sqft_living      0.701917
sqft_lot         0.089876
floors           0.256804
waterfront       0.264306
view             0.393497
condition        0.036056
sqft_above       0.605368
sqft_basement    0.321108
yr_built         0.053953
Name: price, dtype: float64
```

In [380]:
```
# Plot showing the category with the highest linearity with price
sns.scatterplot(x="sqft_living",y="price",data=houses)
plt.title("price vs sqft_living")
plt.savefig("images/price vs sqft_living");
```

Modelling

In [381]:
```
# Identify the selected features and create dummy variables
selected_features = houses.copy()

# Creating dummies
view_dummy = pd.get_dummies(selected_features, columns=["view"],drop_first=True)
grade_dummy = pd.get_dummies(selected_features, columns=["grade"],drop_first=True)
features_df = pd.concat([selected_features,grade_dummy,view_dummy],axis=1)
features_df = features_df.drop(['grade', 'view'], axis=1)
# Remove any duplicated columns
features_df = features_df.loc[:,~features_df.columns.duplicated()]

# Calculating the Mean Absolute Error of the model
def mae(x,y,model):
    y_pred = model.predict(sm.add_constant(x))
    mae = mean_absolute_error(y,y_pred)
    return mae
```

In [382]:
```
# Check for correlation on features_df
features_df.corr().abs()["price"]
```

Out[382]:
```
price            1.000000
bedrooms         0.308787
bathrooms        0.525906
sqft_living      0.701917
sqft_lot         0.089876
floors           0.256804
waterfront       0.264306
condition        0.036056
sqft_above       0.605368
sqft_basement    0.321108
yr_built         0.053953
grade_11         0.357589
grade_12         0.291068
grade_13         0.211806
grade_3          0.005155
grade_4          0.031618
grade_5          0.084549
grade_6          0.209463
grade_7          0.316053
grade_8          0.004576
grade_9          0.235859
view_1           0.092597
view_2           0.147179
view_3           0.182932
view_4           0.303059
Name: price, dtype: float64
```

Baseline model

- sqft_living is selected as the independent column and we want to use as it has the highest correlation with the price

In [383]:
```
# Identify X and y variables that we shall use. sqft_living is selected as it has the highes
X_model_baseline = features_df['sqft_living']
y_model_baseline = features_df['price']

# Create and fit the model
model_baseline = sm.OLS(y_model_baseline, sm.add_constant(X_model_baseline)).fit()
```

In [384]:
```
# Baseline model
model_baseline.summary()
```

Out[384]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.493
Model:	OLS	**Adj. R-squared:**	0.493
Method:	Least Squares	**F-statistic:**	2.097e+04
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	03:50:53	**Log-Likelihood:**	-3.0006e+05
No. Observations:	21597	**AIC:**	6.001e+05
Df Residuals:	21595	**BIC:**	6.001e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-4.399e+04	4410.023	-9.975	0.000	-5.26e+04	-3.53e+04
sqft_living	280.8630	1.939	144.819	0.000	277.062	284.664

Omnibus:	14801.942	**Durbin-Watson:**	1.982
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	542662.604
Skew:	2.820	**Prob(JB):**	0.00
Kurtosis:	26.901	**Cond. No.**	5.63e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [385]:
```
mae(X_model_baseline,y_model_baseline,model_baseline)
```

Out[385]:
```
173824.8874961748
```

The baseline model is interpreted as follows:

- The formula for getting price is written as

```
price = -43990 + 280*sqft_living
```

- The overall model explains about 49.3% of the variance in the prices.
- The sqft_living coefficient is statistically significant.
- The model is off by about $173, 824inprice * Thehousepriceis$43,990 when the sqft_living is 0
- For a one square foot increase in sqft_living, there is a $280 increase in price

Model 1

- We shall perform multiple linear regression
- This will be a model that has 2 independent continuous numerical values

In [386]:
```
X_model1 = features_df.loc[:,["sqft_living", "sqft_basement"]]
y_model1 = features_df["price"]

model1 = sm.OLS(y_model1, sm.add_constant(X_model1)).fit()
```

In [387]:
```
# Model 1
model1.summary()
```

Out[387]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.493
Model:	OLS	**Adj. R-squared:**	0.493
Method:	Least Squares	**F-statistic:**	1.051e+04
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	03:50:53	**Log-Likelihood:**	-3.0005e+05
No. Observations:	21597	**AIC:**	6.001e+05
Df Residuals:	21594	**BIC:**	6.001e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-4.106e+04	4453.261	-9.220	0.000	-4.98e+04	-3.23e+04
sqft_living	276.6134	2.146	128.920	0.000	272.408	280.819
sqft_basement	20.6946	4.479	4.620	0.000	11.916	29.474

Omnibus:	14754.603	**Durbin-Watson:**	1.982
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	538977.524
Skew:	2.807	**Prob(JB):**	0.00
Kurtosis:	26.821	**Cond. No.**	5.75e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.75e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [388]:
```
mae(X_model1,y_model1,model1)
```

Out[388]:
```
173566.09992442088
```

The first model is interpreted as follows:

- The formula for getting price is price = -41060 + 276.61(sqft_living) + 20.70(sqft_basement)
- The overall model explains about 49.3% of the variance in the prices.
- The model is off by about $173,566 inprice * Allthecoefficientsarestatisticallysignificant * Thehousepriceis -$41060 when the sqft_living and sqft_basement are 0
- For a 1 square foot increase in the sqft_living, there is a $276.61 increaseinprice * Fora 1 squarefootincreaseinthesqft_basement, thereisa$20.70 increase in price

Model 2

- Perform multiple linear regression on two other continuous variables

In [389]:
```
X_model2 = features_df.loc[:,["sqft_lot","sqft_above"]]
y_model2 = features_df["price"]

model2 = sm.OLS(y_model2,sm.add_constant(X_model2)).fit()
```

In [390]:
```
# Model 2
model2.summary()
```

Out[390]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.367
Model:	OLS	**Adj. R-squared:**	0.367
Method:	Least Squares	**F-statistic:**	6259.
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	03:50:54	**Log-Likelihood:**	-3.0245e+05
No. Observations:	21597	**AIC:**	6.049e+05
Df Residuals:	21594	**BIC:**	6.049e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	5.948e+04	4736.364	12.559	0.000	5.02e+04	6.88e+04
sqft_lot	-0.1983	0.049	-4.058	0.000	-0.294	-0.103
sqft_above	270.4952	2.445	110.642	0.000	265.703	275.287

Omnibus:	16444.632	**Durbin-Watson:**	1.987
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	722025.818
Skew:	3.252	**Prob(JB):**	0.00
Kurtosis:	30.569	**Cond. No.**	1.05e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.05e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

In [391]:
```
mae(X_model2,y_model2,model2)
```

Out[391]:
```
192035.56368124706
```

The second model is interpreted as follows:

- The formula for getting price is price = 276700 + 1161000(waterfront) + 171100(floors)
- The overall model explains about 0.133% of the variance in the prices.
- The model is off by about $220,007inprice * Allthecoefficientsarestatisticallysignificant * Thehousepriceis$276700 when the waterfront and floors are 0
- For an availability of a waterfront, there is a $1,161,000increaseinprice * Fora1increaseinthefloors, thereisa$171,100 increase in price

Model 3

- The model will drop all the features that has the smallest correlation

In [392]:
```
X_model3 = features_df.drop(["price"],axis=1)
X_model3.drop(["grade_3","grade_8"],axis=1,inplace=True)
y_model3 = features_df["price"]

model3 = sm.OLS(y_model3,sm.add_constant(X_model3)).fit()
```

In [393]:
```
# Model 3
model3.summary()
```

Out[393]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.654
Model:	OLS	**Adj. R-squared:**	0.654
Method:	Least Squares	**F-statistic:**	1856.
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	03:50:54	**Log-Likelihood:**	-2.9592e+05
No. Observations:	21597	**AIC:**	5.919e+05
Df Residuals:	21574	**BIC:**	5.921e+05
Df Model:	22		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.412e+06	1.35e+05	47.515	0.000	6.15e+06	6.68e+06
bedrooms	-3.738e+04	2053.371	-18.203	0.000	-4.14e+04	-3.34e+04
bathrooms	4.954e+04	3491.594	14.189	0.000	4.27e+04	5.64e+04
sqft_living	145.8608	19.383	7.525	0.000	107.869	183.852
sqft_lot	-0.2692	0.037	-7.343	0.000	-0.341	-0.197
floors	3.944e+04	3783.650	10.423	0.000	3.2e+04	4.69e+04
waterfront	5.178e+05	2.19e+04	23.639	0.000	4.75e+05	5.61e+05
condition	2.033e+04	2468.439	8.236	0.000	1.55e+04	2.52e+04
sqft_above	47.6731	19.355	2.463	0.014	9.736	85.610
sqft_basement	40.9728	19.235	2.130	0.033	3.271	78.674
yr_built	-3211.9697	68.223	-47.080	0.000	-3345.692	-3078.247
grade_11	4.301e+05	1.23e+04	35.104	0.000	4.06e+05	4.54e+05
grade_12	8.362e+05	2.44e+04	34.223	0.000	7.88e+05	8.84e+05

grade_13	1.923e+06	6.14e+04	31.307	0.000	1.8e+06	2.04e+06
grade_4	-1.893e+05	4.2e+04	-4.507	0.000	-2.72e+05	-1.07e+05
grade_5	-2.059e+05	1.48e+04	-13.914	0.000	-2.35e+05	-1.77e+05
grade_6	-1.607e+05	6609.464	-24.318	0.000	-1.74e+05	-1.48e+05
grade_7	-1.005e+05	4052.420	-24.795	0.000	-1.08e+05	-9.25e+04
grade_9	6.549e+04	5106.767	12.824	0.000	5.55e+04	7.55e+04
view_1	1.225e+05	1.21e+04	10.108	0.000	9.88e+04	1.46e+05
view_2	6.27e+04	7345.826	8.536	0.000	4.83e+04	7.71e+04
view_3	1.321e+05	1e+04	13.173	0.000	1.12e+05	1.52e+05
view_4	2.865e+05	1.52e+04	18.825	0.000	2.57e+05	3.16e+05

Omnibus:	11446.125	**Durbin-Watson:**	1.977
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	302660.740
Skew:	2.017	**Prob(JB):**	0.00
Kurtosis:	20.890	**Cond. No.**	4.05e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.05e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [394]:
```
mae(X_model3,y_model3,model3)
```

Out[394]: 141119.16094729994

The third model is interpreted as follows:

- The overall model explains about 65.4% of the variance in the prices.
- The model is off by about $141,119 in price
- All coefficients are statistically significant
- Compared to grade 3, grades 4,5,6 and 7 have a price decrease

- Compared to grade 3, grades 9,11,12 and 13 have a price increase with grade 13 having the most increase
- Compared to view 0, views 1,3 and 4 have a price increase with view 4 having the most

Model 4

Check for pairs that are highly correlated and remove some of the features

In [395]:
```
# Checking for independent variables that are highly correlated and dropping them
def correlation(df, threshold):
    """
    The function takes in a dataframe and threshold
    The threshold determines what minimum correlation value you want to check from the dataframe
    It returns the columns that fit the threshold
    """

    corr_cols = set()
    corr_matrix = df.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j]) > threshold:
                new_col = corr_matrix.columns[i]
                corr_cols.add(new_col)
    return corr_cols
```

In [396]:
```
X_model4 = features_df.drop(["price","grade_3","grade_8"],axis=1)
X_model4 = X_model4.drop(correlation(features_df,0.6),axis=1)
y_model4 = features_df["price"]

model4 = sm.OLS(y_model4,sm.add_constant(X_model4)).fit()
```

In [397]:
```python
# Model 4
model4.summary()
```

Out[397]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.600
Model:	OLS	**Adj. R-squared:**	0.600
Method:	Least Squares	**F-statistic:**	1620.
Date:	Sat, 01 Oct 2022	**Prob (F-statistic):**	0.00
Time:	03:50:54	**Log-Likelihood:**	-2.9749e+05
No. Observations:	21597	**AIC:**	5.950e+05
Df Residuals:	21576	**BIC:**	5.952e+05
Df Model:	20		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	6.756e+06	1.45e+05	46.641	0.000	6.47e+06	7.04e+06
bedrooms	5441.2002	2060.320	2.641	0.008	1402.820	9479.580
bathrooms	1.254e+05	3474.270	36.089	0.000	1.19e+05	1.32e+05
sqft_lot	0.1213	0.039	3.131	0.002	0.045	0.197
floors	6.153e+04	4031.065	15.264	0.000	5.36e+04	6.94e+04
waterfront	5.488e+05	2.35e+04	23.307	0.000	5.03e+05	5.95e+05
condition	1.866e+04	2653.117	7.032	0.000	1.35e+04	2.39e+04
sqft_basement	81.6472	4.494	18.170	0.000	72.840	90.455
yr_built	-3358.4918	73.246	-45.852	0.000	-3502.059	-3214.924
grade_11	6.722e+05	1.24e+04	54.262	0.000	6.48e+05	6.96e+05
grade_12	1.204e+06	2.54e+04	47.463	0.000	1.15e+06	1.25e+06
grade_13	2.531e+06	6.51e+04	38.874	0.000	2.4e+06	2.66e+06

grade_4	-2.829e+05	4.51e+04	-6.269	0.000	-3.71e+05	-1.94e+05
grade_5	-2.971e+05	1.58e+04	-18.781	0.000	-3.28e+05	-2.66e+05
grade_6	-2.357e+05	6969.070	-33.826	0.000	-2.49e+05	-2.22e+05
grade_7	-1.627e+05	4202.279	-38.719	0.000	-1.71e+05	-1.54e+05
grade_9	1.276e+05	5368.923	23.768	0.000	1.17e+05	1.38e+05
view_1	1.42e+05	1.3e+04	10.899	0.000	1.16e+05	1.68e+05
view_2	7.985e+04	7886.736	10.124	0.000	6.44e+04	9.53e+04
view_3	1.56e+05	1.08e+04	14.477	0.000	1.35e+05	1.77e+05
view_4	3.25e+05	1.63e+04	19.879	0.000	2.93e+05	3.57e+05

Omnibus:	13378.851	**Durbin-Watson:**	1.969
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	445620.882
Skew:	2.450	**Prob(JB):**	0.00
Kurtosis:	24.707	**Cond. No.**	4.04e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.04e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

In [398]:
```
mae(X_model4,y_model4,model4)
```

Out[398]: 149953.7152730511

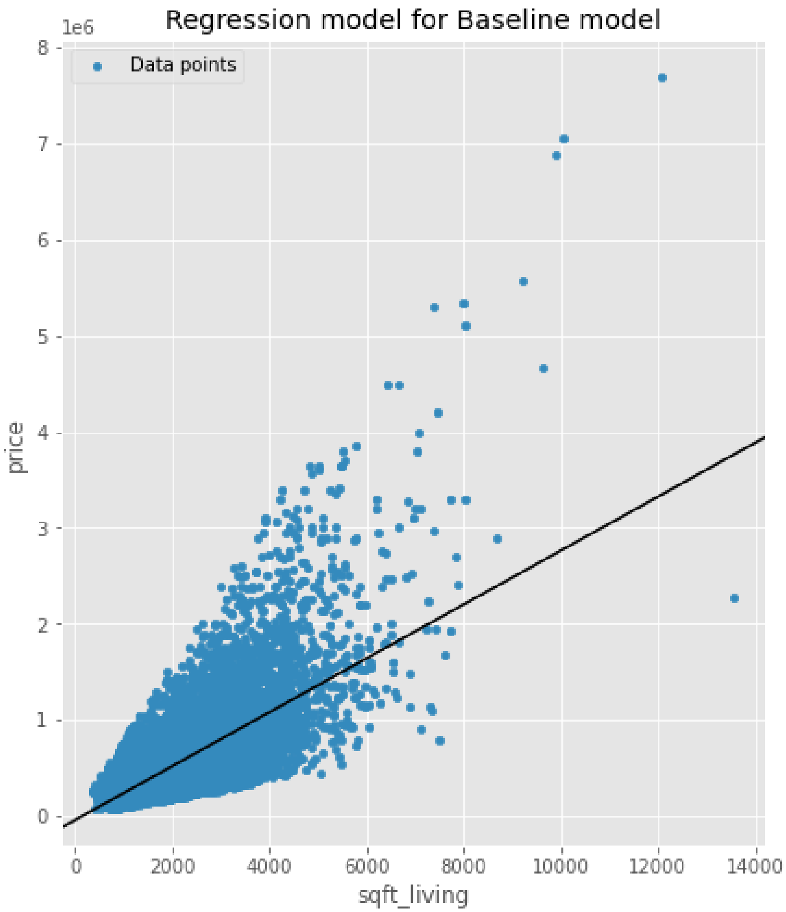
The fourth model is interpreted as follows:

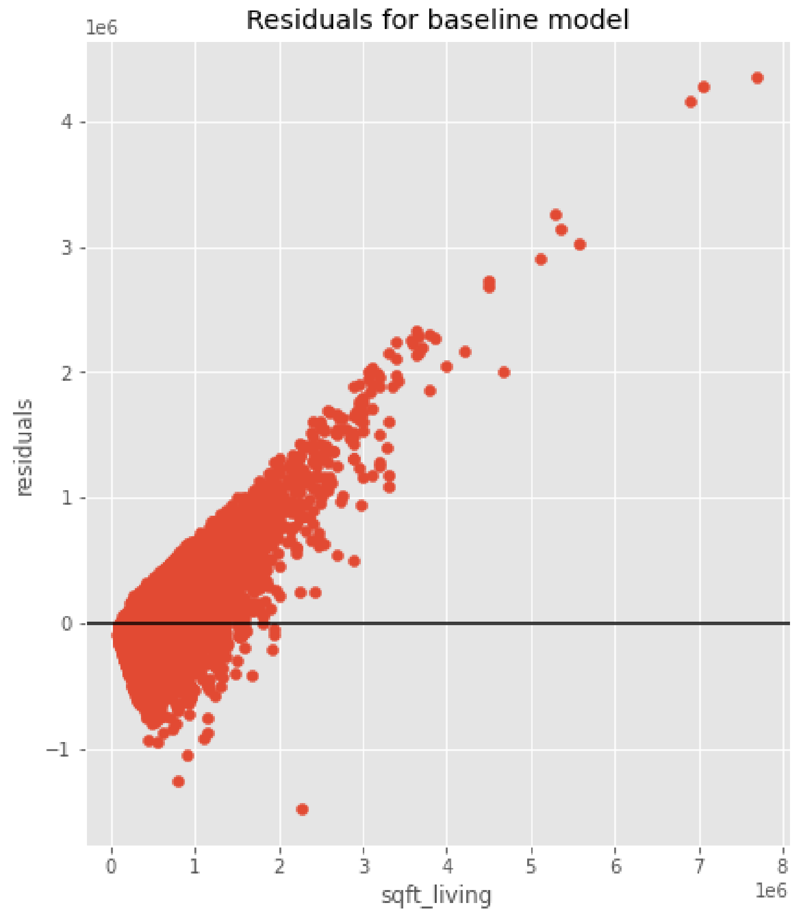
- The overall model explains about 60% of the variance in the prices.
- The model is off by about $149,953 in price
- All cofficients are statistically significant
- Compared to grade 3, grades 4,5,6 and 7 have a price decrease

- Compared to grade 3, grades 9,11,12 and 13 have a price increase with grade 13 having the most increase
- Compared to view 0, all views have a price increase with view 4 having the most then view 1

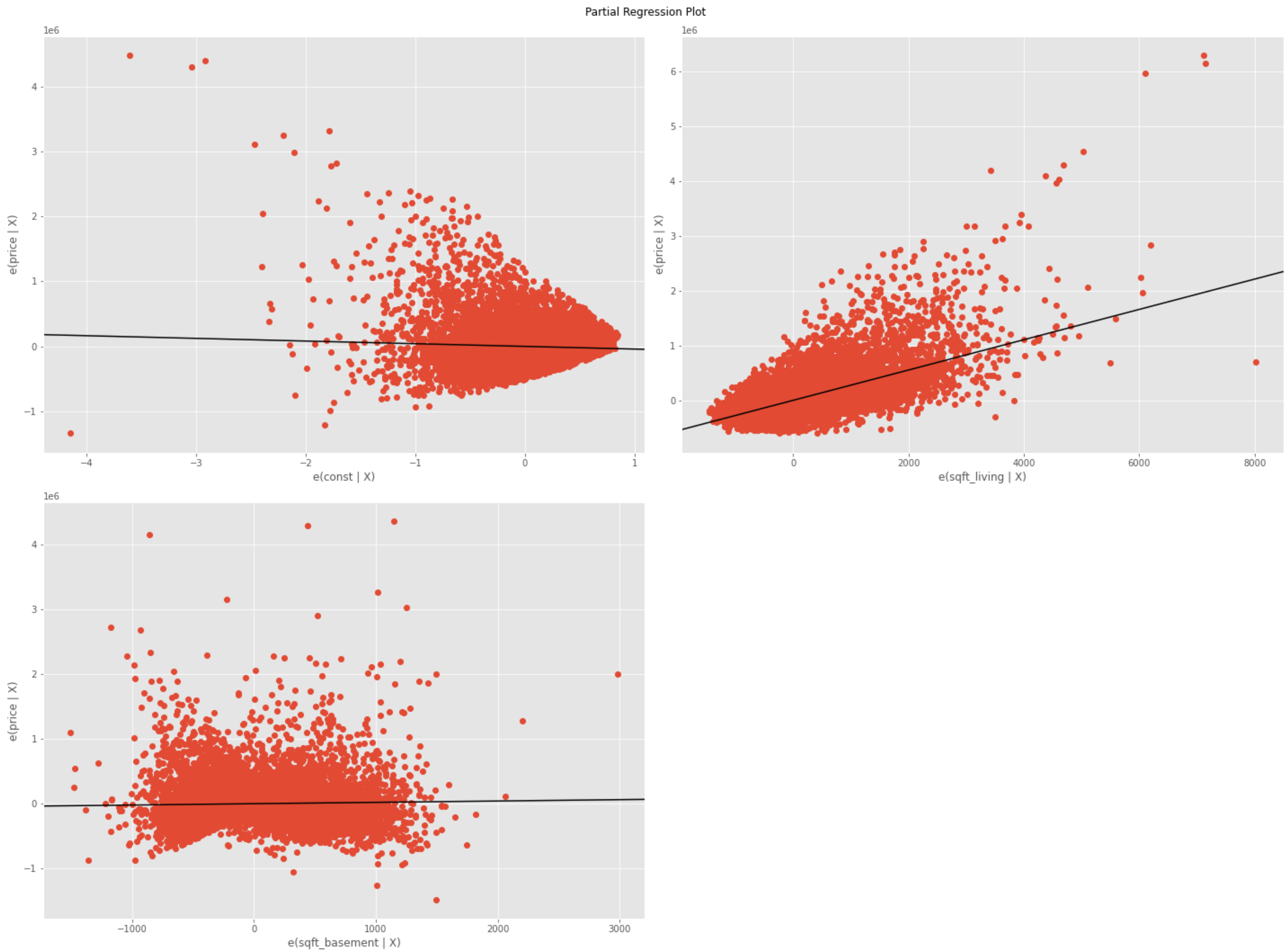
Visualizations of the models

In [400]:
```
# Baseline model
# Plot the regression line
fig, ax = plt.subplots(nrows=1,ncols=2,figsize=(15,8))
features_df.plot.scatter(x="sqft_living", y="price", label="Data points", ax=ax[0])
sm.graphics.abline_plot(model_results=model_baseline, label="Regression line", ax=ax[0], color="black")
ax[0].set_title("Regression model for Baseline model")

#Plot the residuals
ax[1].scatter(features_df["price"], model_baseline.resid)
ax[1].axhline(y=0, color="black")
ax[1].set_xlabel("sqft_living")
ax[1].set_ylabel("residuals")
ax[1].set_title("Residuals for baseline model")
plt.savefig("images/Baseline model plots");
```

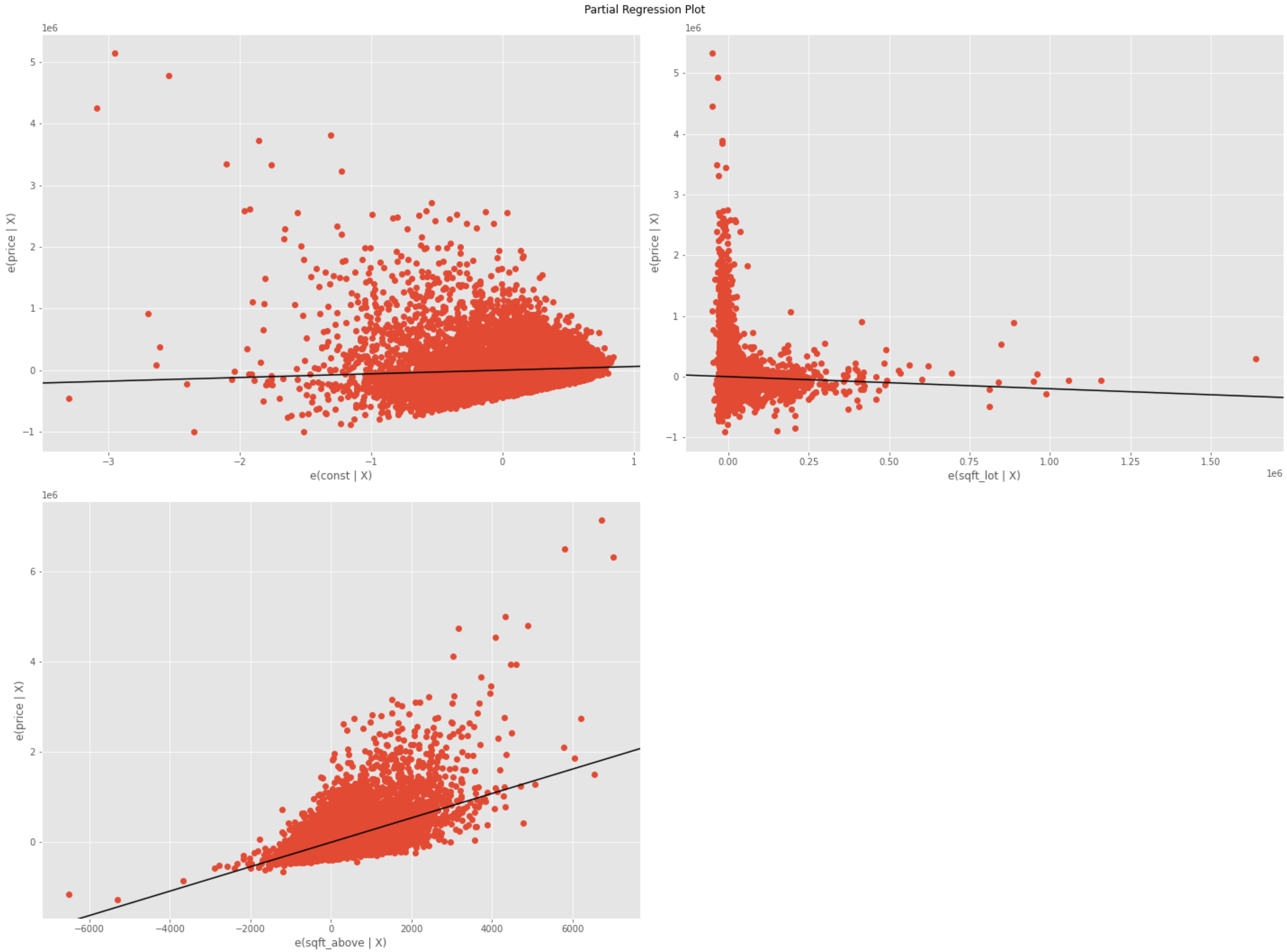

Regression model for Baseline model
1e6
Data points
price
sqft_living
0
1
2
3
4
5
6
7
8
0
2000
4000
6000
8000
10000
12000
14000


Residuals for baseline model
1e6
residuals
sqft_living
1e6
−1
0
1
2
3
4
0
1
2
3
4
5
6
7
8

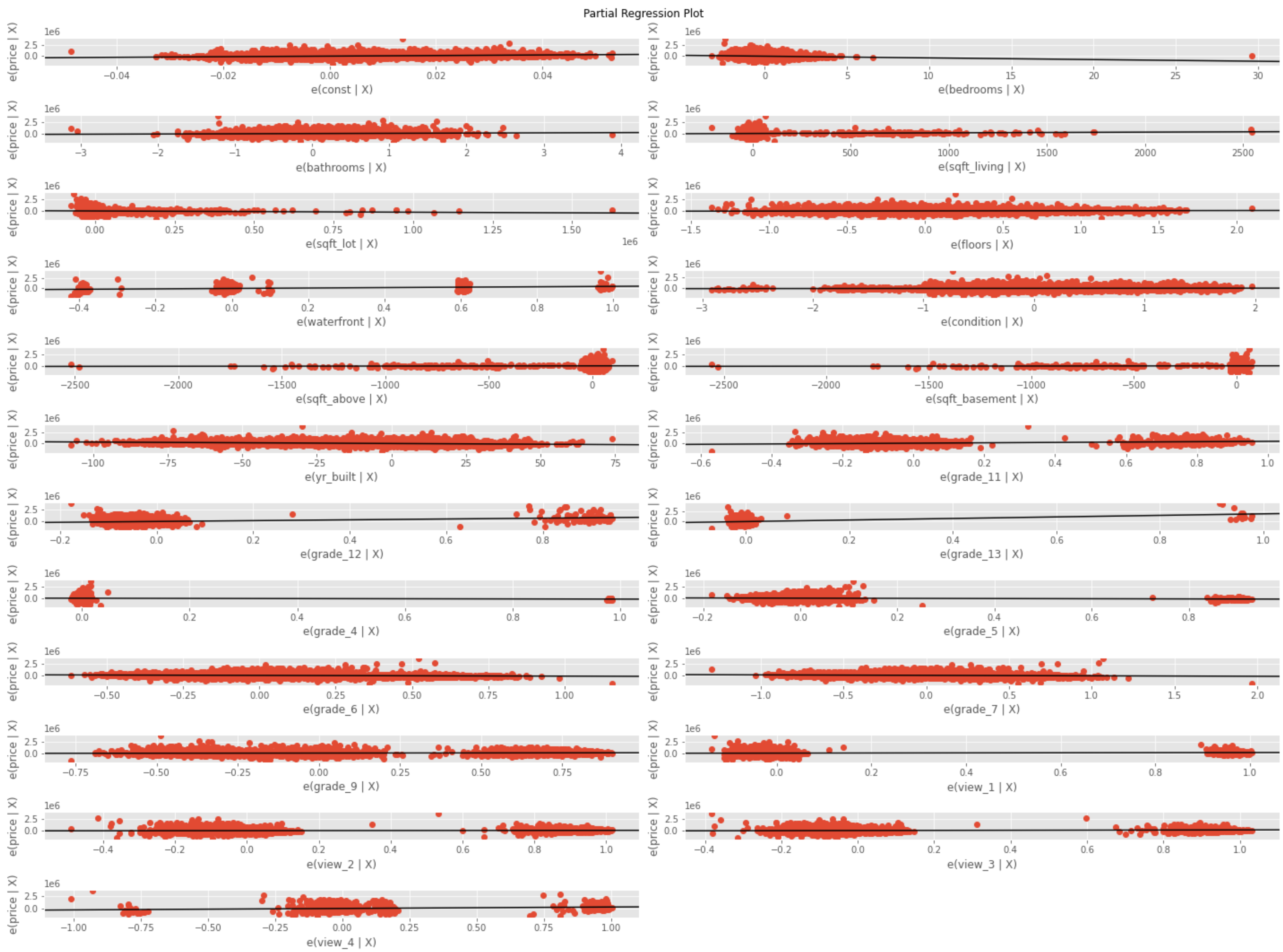
In [414]:
```
# Model 1
fig = plt.figure(figsize=(20,15))
sm.graphics.plot_partregress_grid(model1,fig=fig)
plt.savefig("images/model1 partial regression plot");
```



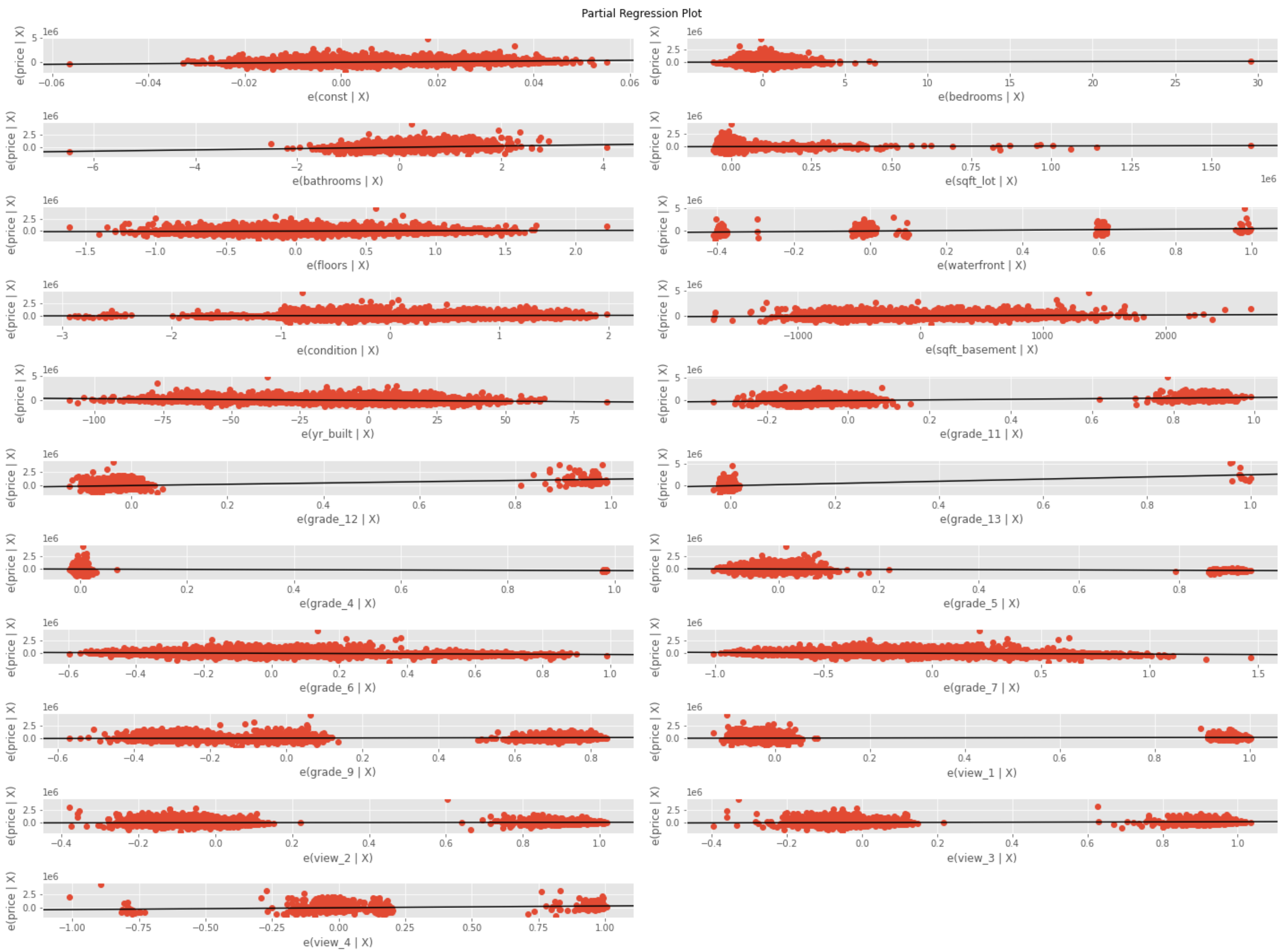
In [415]:
```
# Model 2
fig = plt.figure(figsize=(20,15))
sm.graphics.plot_partregress_grid(model2,fig=fig)
plt.savefig("images/model2 partial regression plot")
```



In [416]:
```
# Model 3
fig = plt.figure(figsize=(20,15))
sm.graphics.plot_partregress_grid(model3,fig=fig)
plt.savefig("images/model3 partial regression plot");
```



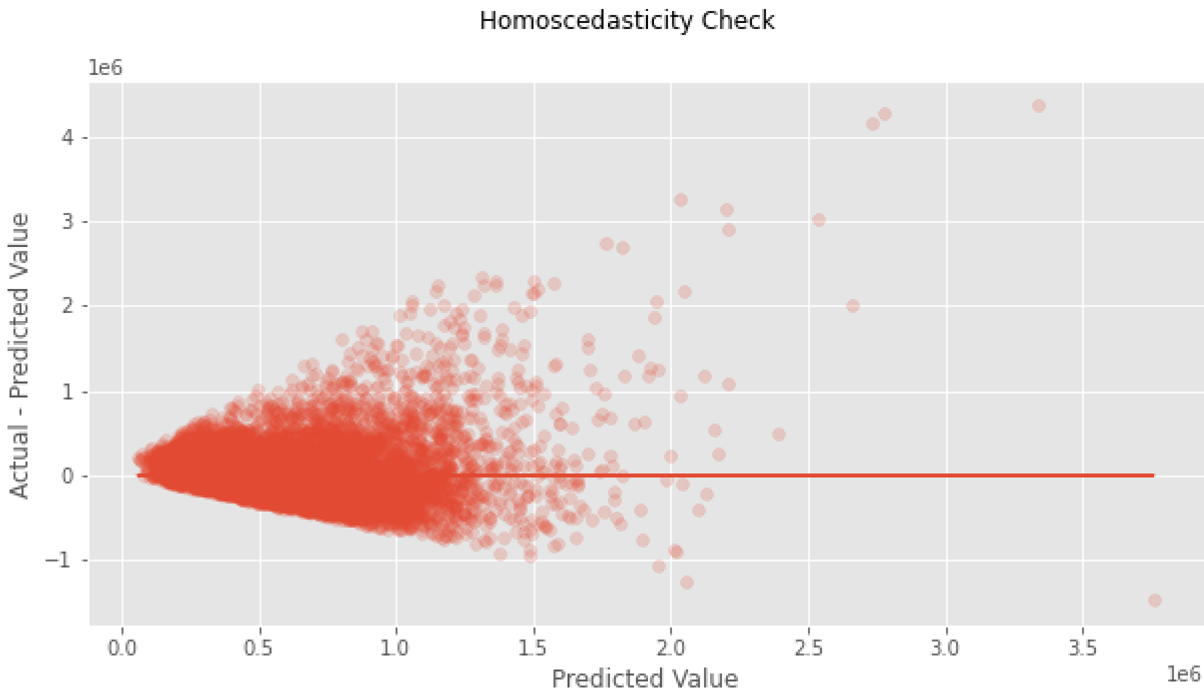
In [418]:
```
# Model 4
fig = plt.figure(figsize=(20,15))
sm.graphics.plot_partregress_grid(model4,fig=fig)
plt.savefig("images/model4 partial regression plot");
```



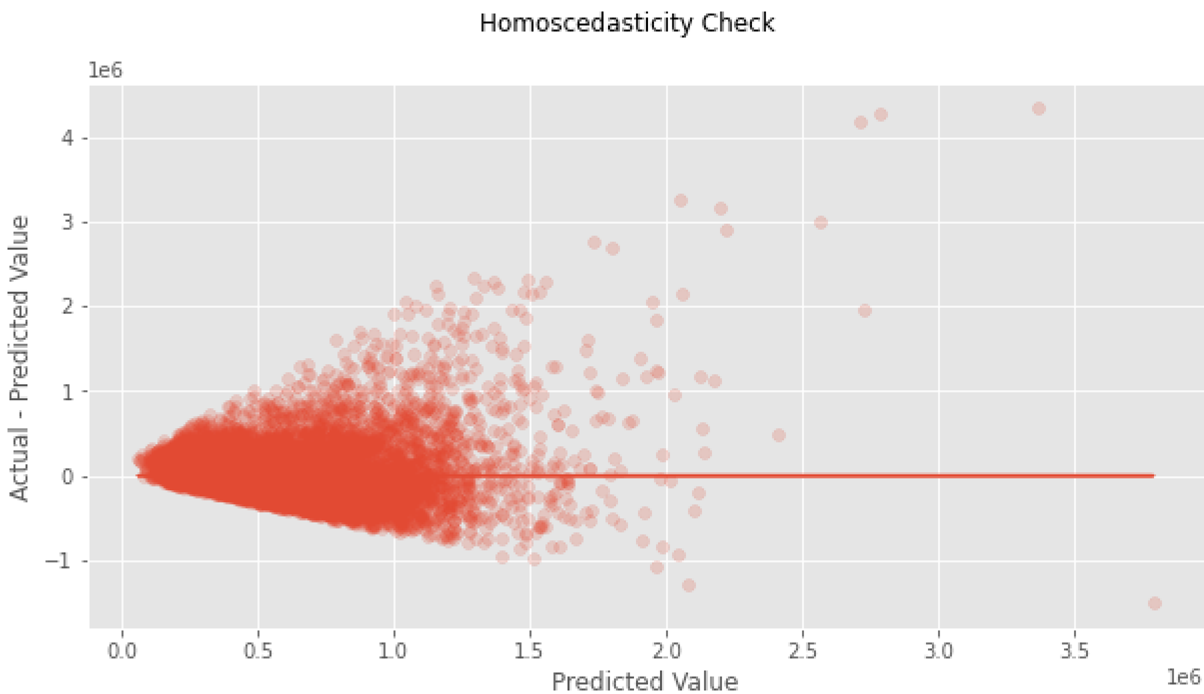
From the above residual plots, it is seen that from the data, the linear regression would be the best fit in comparison to a non-linear one

Homoscedasticity

In [405]:
```
# Baseline Model
preds = model_baseline.predict(sm.add_constant(X_model_baseline))
residuals = model_baseline.resid

fig, ax = plt.subplots(figsize=(10,5))
fig.suptitle('Homoscedasticity Check')
ax.scatter(preds, residuals, alpha=0.2)
ax.plot(preds, [0 for i in range(len(X_model_baseline))])
ax.set_xlabel("Predicted Value")
ax.set_ylabel("Actual - Predicted Value")
plt.savefig("images/Basline Model Homoscedasticity check");
```

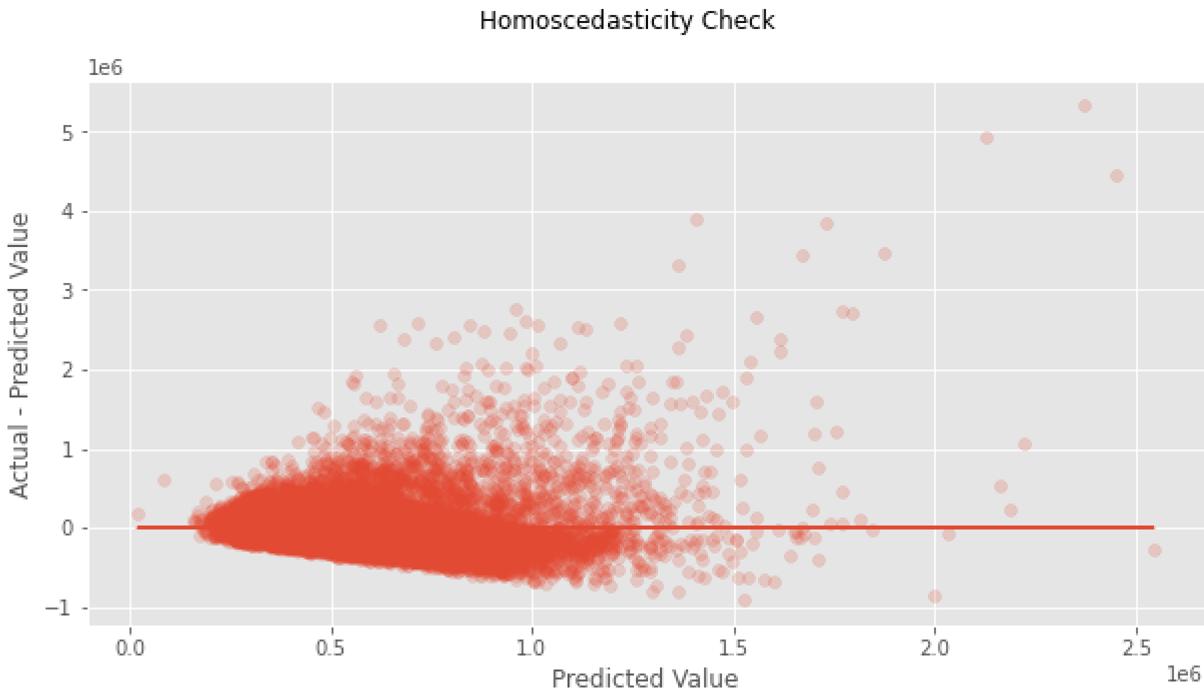


In [406]:
```
# Model 1
preds = model1.predict(sm.add_constant(X_model1))
residuals = model1.resid

fig, ax = plt.subplots(figsize=(10,5))
fig.suptitle('Homoscedasticity Check')
ax.scatter(preds, residuals, alpha=0.2)
ax.plot(preds, [0 for i in range(len(X_model1))])
ax.set_xlabel("Predicted Value")
ax.set_ylabel("Actual - Predicted Value")
plt.savefig("images/Model one Homoscedasticity check");
```

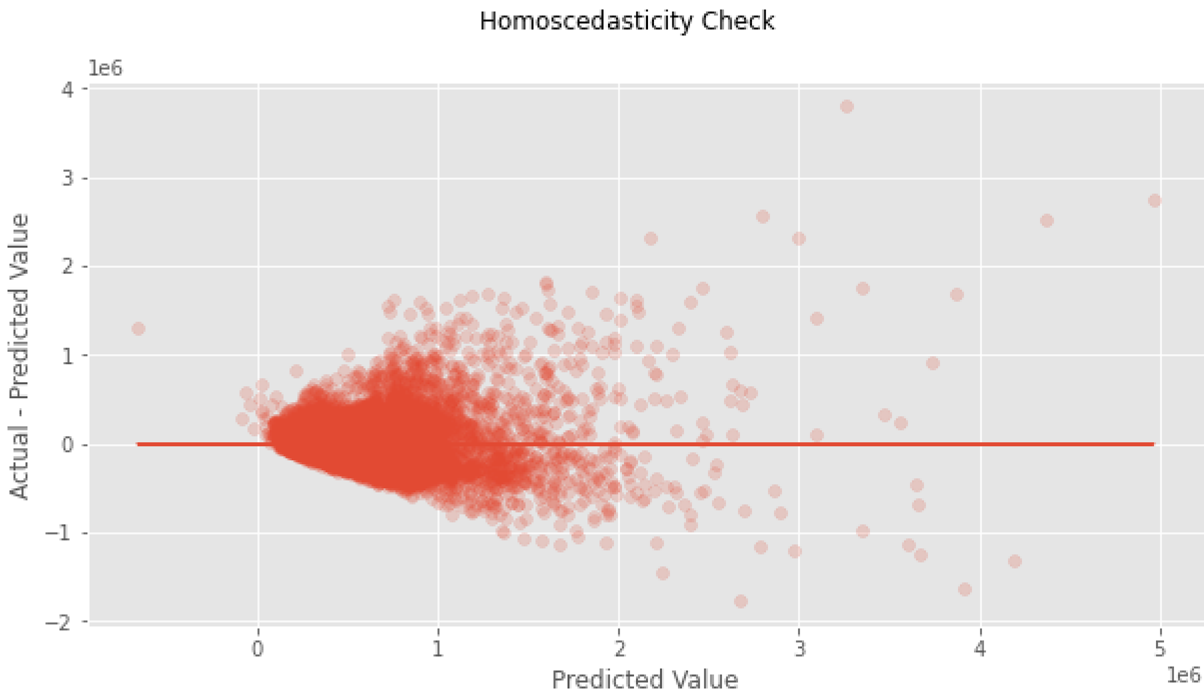


In [407]:

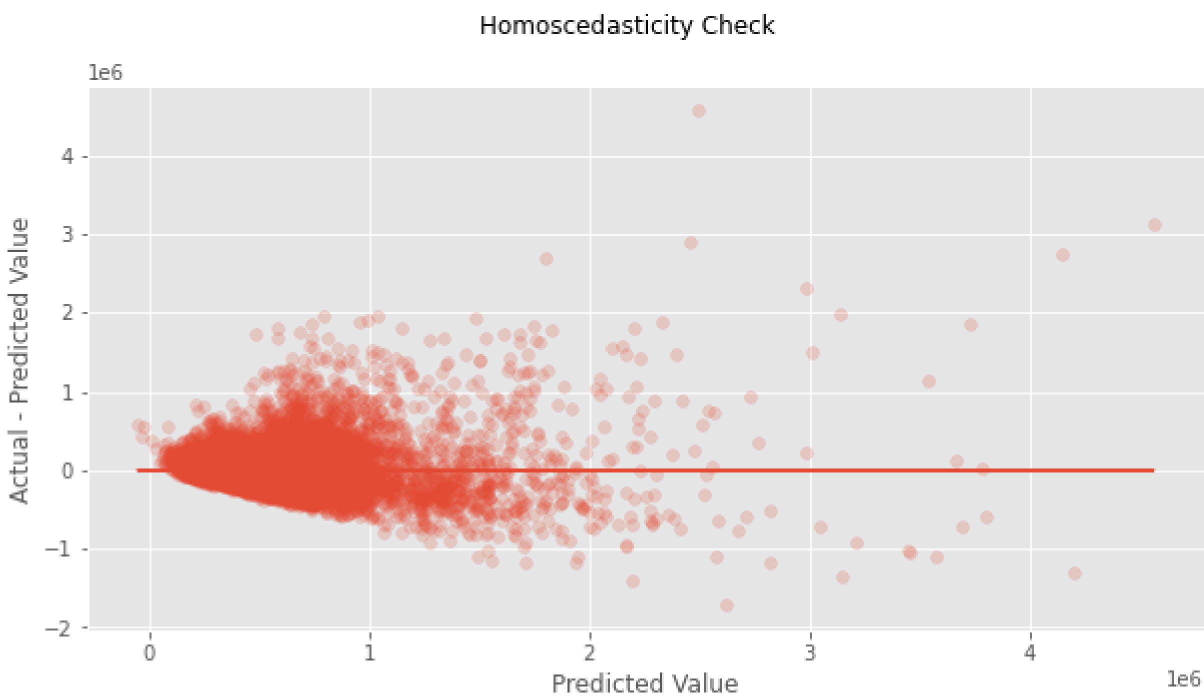
```
# Model 2
preds = model2.predict(sm.add_constant(X_model2))
residuals = model2.resid

fig, ax = plt.subplots(figsize=(10,5))
fig.suptitle('Homoscedasticity Check')
ax.scatter(preds, residuals, alpha=0.2)
ax.plot(preds, [0 for i in range(len(X_model2))])
ax.set_xlabel("Predicted Value")
ax.set_ylabel("Actual - Predicted Value")
plt.savefig("images/Model two Homoscedasticity check");
```



In [408]:
```
# Model 3
preds = model3.predict(sm.add_constant(X_model3))
residuals = model3.resid

fig, ax = plt.subplots(figsize=(10,5))
fig.suptitle('Homoscedasticity Check')
ax.scatter(preds, residuals, alpha=0.2)
ax.plot(preds, [0 for i in range(len(X_model3))])
ax.set_xlabel("Predicted Value")
ax.set_ylabel("Actual - Predicted Value")
plt.savefig("images/Model three Homoscedasticity check");
```



In [409]:
```
# Model 4
preds = model4.predict(sm.add_constant(X_model4))
residuals = model4.resid

fig, ax = plt.subplots(figsize=(10,5))
fig.suptitle('Homoscedasticity Check')
ax.scatter(preds, residuals, alpha=0.2)
ax.plot(preds, [0 for i in range(len(X_model4))])
ax.set_xlabel("Predicted Value")
ax.set_ylabel("Actual - Predicted Value")
plt.savefig("images/Model four Homoscedasticity check");
```



The models fail the Homoscedasticity check

Conclusion

Final model

Model 3 is selected out of all the models as it has a higher r-squared value and a low mean absolute error The model's characteristics are:

- The overall model explains about 65.4% of the variance in the prices.
- The model is off by about $141,119 in price
- All coefficients are statistically significant
- Compared to grade 3, grades 4,5,6 and 7 have a price decrease
- Compared to grade 3, grades 9,11,12 and 13 have a price increase with grade 13 having the most increase
- Compared to view 0, views 1,3 and 4 have a price increase with view 4 having the most

Recommendation

I would recommend the following:

- Houses with a waterfront are set to increase the price by $517,800 * Thetypesofview, comparedtohavingnoviewswouldbehavingahousehavingaviewqualityof 1 whichisconsideredfairasitwouldincreasethepr$ 122,500 and should one want the best view at wuality of 4, they would see an increase in price by $286,500
- When it comes to building materials, one should go with the rating that is more than average to increase the sale as the highest geade would increase price by 1,923,000

Limitations and Next steps

The model fits about 65% of the data and additional tests to check for normality and scaling of features should be considered. Also, for future research, deployment of other models besides the regression would be best and the data needs to be analysed while zoning in on specific zipcodes so as to get more accurate and fine tuned results