Kings County Real Estate Analysis

Please fill out:

- Student name: Andrew Bennett
- Student pace: self paced / **part time** / full time
- Scheduled project review date/time: **Sept. 7th 2:00pm**
- Instructor name: Morgan Jones
- Blog post URL: https://dev.to/bennettandrewm/false-myths-of-false-positives-3kmd

1. Project Overview

A Seattle real estate brokerage wants to expand their services to developers. They're offering "state of the art" data analysis to new developers in the area on where and what to build, as it relates to price. They want to partner with you to serve as their in-house data engineer to build, operate, and interface with the model.

This analysis would include, at a minimum, a linear regression model to examine the relationship between square footage and zip code on price. They'd also like to see how other factors affect the price, if at all.

2. Business Understanding

The Seattle real estate market is always competitive. To stay ahead of the competition, a brokerage firm must attract new clients and keep them. To do this, KRG Realty is providing a data analytics package to lure buyers and sellers and developers to their business. This new package would provide linear regression modeling to analyze the relationship between square footage and zipcode at a minimum.

3. Data Import and Inspection

To perform this analysis, we're utilizing data from the Kings County House Sales dataset in the form of a csv file (`data/kc_house_data.csv`).

Step 1 - Import Data

Let's import the data and see what it looks like.

In [1]:
```
# import data using read_csv
import pandas as pd

kc = pd.read_csv('data/kc_house_data.csv')
kc
```

Out[1]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.0	N
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.0	N
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.0	N
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.0	N
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.0	N
...	...	...	...	...	...	...	...	...	
30150	7834800180	11/30/2021	1555000.0	5	2.0	1910	4000	1.5	N
30151	194000695	6/16/2021	1313000.0	3	2.0	2020	5800	2.0	N
30152	7960100080	5/27/2022	800000.0	3	2.0	1620	3600	1.0	N
30153	2781280080	2/24/2022	775000.0	3	2.5	2570	2889	2.0	N
30154	9557800100	4/29/2022	500000.0	3	1.5	1200	11058	1.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror

30155 rows × 25 columns

We've successfully imported the CSV file into a data frame. We can see that there are 25 columns in this dataframe. Let's go ahead and input those columns name.

Step 2 - Data Inspection

As we inspect the data, we're going to start with the General Information and work our way into classifying the data at numeric or categorical. Within numeric, will breakdown to discrete and

General Information

In [2]:

```
#print column names
columns = kc.columns
columns
```

Out[2]:

```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'greenbelt', 'nuisance', 'view',
       'condition', 'grade', 'heat_source', 'sewer_system', 'sqft_above',
       'sqft_basement', 'sqft_garage', 'sqft_patio', 'yr_built',
       'yr_renovated', 'address', 'lat', 'long'],
      dtype='object')
```

We have the column names, stored here. It looks like we have a lot of standard real estate information, coupled with other address data. Let's see which information is numeric vs which is categoric.

In [3]:

```
#get data info
kc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30155 entries, 0 to 30154
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             30155 non-null  int64
 1   date           30155 non-null  object
 2   price          30155 non-null  float64
 3   bedrooms       30155 non-null  int64
 4   bathrooms      30155 non-null  float64
 5   sqft_living    30155 non-null  int64
 6   sqft_lot       30155 non-null  int64
 7   floors         30155 non-null  float64
 8   waterfront     30155 non-null  object
 9   greenbelt      30155 non-null  object
 10  nuisance       30155 non-null  object
 11  view           30155 non-null  object
 12  condition      30155 non-null  object
 13  grade          30155 non-null  object
 14  heat_source    30123 non-null  object
 15  sewer_system   30141 non-null  object
 16  sqft_above     30155 non-null  int64
```

```
 17  sqft_basement  30155 non-null  int64
 18  sqft_garage    30155 non-null  int64
 19  sqft_patio     30155 non-null  int64
 20  yr_built       30155 non-null  int64
 21  yr_renovated   30155 non-null  int64
 22  address        30155 non-null  object
 23  lat            30155 non-null  float64
 24  long           30155 non-null  float64
dtypes: float64(5), int64(10), object(10)
memory usage: 5.8+ MB
```

In [4]:

```
#group variable column names by data type

numeric_cont = ['date', 'price', 'sqft_living', 'sqft_lot', 'sqft_above',
       'sqft_basement', 'sqft_garage', 'sqft_patio', 'yr_built',
       'yr_renovated', 'lat', 'long']
numeric_disc = ['bedrooms', 'bathrooms', 'floors']
categorical = ['waterfront', 'greenbelt', 'nuisance', 'view',
       'condition', 'grade', 'heat_source', 'sewer_system', 'address']
```

Great, so we've got our our columns saved AND we have them saved by variable type. We're in good shape, so far. Let's take a look at each of variable types and see if we can see anything, interesting.

Numeric Continuous

In [5]:

```
#inspect the numeric continuous data
kc[numeric_cont].describe()
```

Out[5]:

	price	sqft_living	sqft_lot	sqft_above	sqft_basement	sqft_garage	sqft_pa
count	3.015500e+04	30155.000000	3.015500e+04	30155.000000	30155.000000	30155.000000	30155.000(
mean	1.108536e+06	2112.424739	1.672360e+04	1809.826098	476.039396	330.211142	217.412(
std	8.963857e+05	974.044318	6.038260e+04	878.306131	579.631302	285.770536	245.3027
min	2.736000e+04	3.000000	4.020000e+02	2.000000	0.000000	0.000000	0.0000(
25%	6.480000e+05	1420.000000	4.850000e+03	1180.000000	0.000000	0.000000	40.0000(
50%	8.600000e+05	1920.000000	7.480000e+03	1560.000000	0.000000	400.000000	150.0000(
75%	1.300000e+06	2619.500000	1.057900e+04	2270.000000	940.000000	510.000000	320.0000(
max	3.075000e+07	15360.000000	3.253932e+06	12660.000000	8020.000000	3580.000000	4370.0000(

Okay, so a few things interesting here: `price` - looks okay, min is 27000, max is 30,000,000, mean is 1.1M which is higher than median 860,000. 30M seems high, but can't tell if it will skew anything. `sqft` - min of 3.000 looks small. Perhaps we should clean that up. Other dats looks okay. Max is quite high. `sqft_lot` - min of 402 sq.ft lot is small. other data looks okay. `sqft_above` - min of 2 sq.ft living space is small. other data looks okay. `sqft_basement` - max of 8020 sq.ft living space is probably too large. nothing else obviously off. `yr_built` - min year is 1900. That means subtracting by 1900 would be a good way to adjust the year. `yr_renovated` - most homes haven't been remodeled. Nothing to adjust here.

Numeric Discrete

In [6]:
```python
#inspect the numeric discrete data
kc[numeric_disc].describe()
```

Out[6]:

	bedrooms	bathrooms	floors
count	30155.000000	30155.000000	30155.000000
mean	3.413530	2.334737	1.543492
std	0.981612	0.889556	0.567717
min	0.000000	0.000000	1.000000
25%	3.000000	2.000000	1.000000
50%	3.000000	2.500000	1.500000
75%	4.000000	3.000000	2.000000
max	13.000000	10.500000	4.000000

Some things that stand out: `bedrooms` - min of bedrooms looks unrealistic, unless we're talking about studio apartments. Perhaps we should clean that up. Max is quite high. `bathrooms` - min of 0 is small and doesn't really quantify and new construction. `floors` - min of 2 sq.ft living space is small. other data looks okay. Although, we're not really sure what floors mean.

In [7]:
```python
##Lets print the categorical data
kc[categorical]
```

Out[7]:

	waterfront	greenbelt	nuisance	view	condition	grade	heat_source	sewer_system	a
0	NO	NO	NO	NONE	Good	7 Average	Gas	PUBLIC	So 21s Wasl
1	NO	NO	YES	AVERAGE	Average	7 Average	Oil	PUBLIC	Gree Wa
2	NO	NO	NO	AVERAGE	Average	7 Average	Gas	PUBLIC	850. 113tl Wasl
3	NO	NO	NO	AVERAGE	Average	9 Better	Gas	PUBLIC	407 Washi
4	NO	NO	YES	NONE	Average	7 Average	Electricity	PUBLIC	No Talu

	waterfront	greenbelt	nuisance	view	condition	grade	heat_source	sewer_system	a
									Is Wa
...	...	...	...	...	...	...	...	...	
30150	NO	NO	NO	NONE	Good	8 Good	Oil	PUBLIC	4673 Washi
30151	NO	NO	NO	FAIR	Average	7 Average	Gas	PUBLIC	41 Sou Wasl
30152	NO	NO	YES	NONE	Average	7 Average	Gas	PUBLIC	910 Luth V
30153	NO	NO	NO	NONE	Average	8 Good	Gas	PUBLIC	1712 Sou Wa
30154	NO	NO	NO	NONE	Average	7 Average	Oil	PUBLIC	18 Wasl

30155 rows × 9 columns

Some things that stand out: `price` , `waterfront` , and `nuisance` - binary categories of yes, no, except water appears to include the water it faces `view` - has different descriptions of the quality of the view. `condition` - string description of the quality `grade` - min of 2 sq.ft living space is small. other data looks okay. `heat_source` , `sewer_system` - describe the systems with a string. `address` - includes the full address as a string.

In [8]:
```
#print the value counts of condition
kc['condition'].value_counts()
```

Out[8]:
```
Average      18547
Good          8054
Very Good     3259
Fair           230
Poor            65
Name: condition, dtype: int64
```

The `condition` category has 5 different values, with "average" being the most popular. This is a tricky category to factor because the "Average" vlaue is roughly 60% of the data. That's not to say it couldn't be used.

In [9]:
```
#print the value counts of grade
kc['grade'].value_counts()
```

Out[9]:
```
7 Average        11697
8 Good            9410
9 Better          3806
6 Low Average     2858
10 Very Good      1371
11 Excellent       406
5 Fair             393
12 Luxury          122
4 Low               51
13 Mansion          24
3 Poor              13
1 Cabin              2
2 Substandard        2
Name: grade, dtype: int64
```

Okay, this looks like a promising category. It appears that we have a range of numbers 1-13. We can convert these to numbers we'll have a continuous numeric value.

In [10]:
```
#print the value counts of view
kc['view'].value_counts()
```

Out[10]:
```
NONE         26589
AVERAGE       1915
GOOD           878
EXCELLENT      553
FAIR           220
Name: view, dtype: int64
```

In [11]:
```
#print the value counts of waterfront
kc['waterfront'].value_counts()
```

Out[11]:
```
NO     29636
YES      519
Name: waterfront, dtype: int64
```

In [12]:
```
#print the value counts of greenbelt
kc['greenbelt'].value_counts()
```

Out[12]:
```
NO     29382
YES      773
Name: greenbelt, dtype: int64
```

4. Data Cleaning

Null Values

We'll start with any information that's empty or missing. We had 30,155 entries previously

In [13]:
```
#drop null values using dropna
kc.dropna(how = "any", inplace = True)
```

In [14]:
```
#verify successful completion
kc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 30111 entries, 0 to 30154
```

```
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             30111 non-null  int64  
 1   date           30111 non-null  object 
 2   price          30111 non-null  float64
 3   bedrooms       30111 non-null  int64  
 4   bathrooms      30111 non-null  float64
 5   sqft_living    30111 non-null  int64  
 6   sqft_lot       30111 non-null  int64  
 7   floors         30111 non-null  float64
 8   waterfront     30111 non-null  object 
 9   greenbelt      30111 non-null  object 
 10  nuisance       30111 non-null  object 
 11  view           30111 non-null  object 
 12  condition      30111 non-null  object 
 13  grade          30111 non-null  object 
 14  heat_source    30111 non-null  object 
 15  sewer_system   30111 non-null  object 
 16  sqft_above     30111 non-null  int64  
 17  sqft_basement  30111 non-null  int64  
 18  sqft_garage    30111 non-null  int64  
 19  sqft_patio     30111 non-null  int64  
 20  yr_built       30111 non-null  int64  
 21  yr_renovated   30111 non-null  int64  
 22  address        30111 non-null  object 
 23  lat            30111 non-null  float64
 24  long           30111 non-null  float64
dtypes: float64(5), int64(10), object(10)
memory usage: 6.0+ MB
```

... And now we're down to 30111. We eliminated ~40 entries. Solid!

Numeric Continuous Data

Let's address some of our concerns with numeric_continuous.

square footage

Let's start with the square footage. We've noticed that one entry had 3 sq.ft. This is unrealistic. In fact, any house with less than 100 sq.ft. may require further inspection. Let's go ahead and take a look

In [15]:
```
#find the number of entries with sqft less than 100
kc[kc['sqft_living']<100]
```

Out[15]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
14977	1549500215	12/17/2021	1803000.0	4	4.0	3	326701	2.0	N

1 rows × 25 columns

Okay so we have one entry, and it looks like something is very off with this sqft_living, as well as sqft garage and sqft_basement. Let's go ahead and get rid of that.

In [16]:
```
#drop these entries
kc.drop(kc[kc['sqft_living']<100].index, inplace=True)
```

lot square footage

Let's start with the lot square footage. We've noticed that a few entries are less than 500 sq.ft. This is tight, so let's take a closer look.

In [17]:
```
#find the number of entries with sqft_lot less than 500
kc[kc['sqft_lot']<500]
```

Out[17]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
711	7625701309	10/29/2021	504950.0	1	2.0	840	487	3.0	N
1798	1498302991	6/22/2021	550000.0	2	2.0	1080	468	2.0	N
2376	9828701815	6/30/2021	749000.0	2	2.0	1030	487	3.0	N
2410	7228501003	2/27/2022	799950.0	2	3.0	1270	474	3.0	N
3035	9297300934	10/5/2021	550000.0	3	2.0	1140	492	3.0	N
3065	3300701084	12/16/2021	599950.0	2	2.0	1010	499	2.0	N
3070	3574300188	1/19/2022	635950.0	3	3.0	1320	435	3.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
3739	9834201384	2/7/2022	509950.0	2	2.0	780	486	2.0	N
4139	7234600295	11/21/2021	1085000.0	2	2.0	1380	479	3.0	N
4310	7228501004	2/16/2022	799950.0	2	3.0	1270	474	3.0	N
5322	9839300500	10/4/2021	550000.0	2	2.0	1050	497	3.0	N
6178	9839300499	10/4/2021	550000.0	2	2.0	1050	497	3.0	N
8788	7217400014	10/13/2021	599950.0	2	2.0	960	412	3.5	N
8878	9839300502	9/10/2021	550000.0	2	2.0	1050	497	3.0	N
8977	9834201386	1/3/2022	479000.0	2	2.0	780	478	2.0	N
9472	546000164	6/24/2021	560000.0	2	2.0	930	498	3.0	N
9547	546000165	6/24/2021	565000.0	2	2.0	930	498	3.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
10627	9834201385	11/23/2021	479000.0	2	2.0	780	478	2.0	N
11303	9834201387	2/7/2022	500000.0	2	2.0	780	487	2.0	N
11360	4083301622	10/6/2021	694950.0	2	2.0	1020	475	3.0	N
12287	9297300937	11/17/2021	550000.0	3	2.0	1140	492	3.0	N
12400	6450300380	2/17/2022	579950.0	2	2.0	1020	420	3.0	N
12864	4449800002	5/26/2022	700000.0	0	0.0	1215	486	3.0	N
13329	9839300503	10/4/2021	550000.0	2	2.0	1050	497	3.0	N
14571	4083301618	10/6/2021	675000.0	2	2.0	1020	488	3.0	N
14830	7625701294	10/13/2021	489950.0	1	2.0	840	480	3.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
16171	9839300501	10/4/2021	550000.0	2	2.0	1050	497	3.0	N
16655	9297300947	9/24/2021	525000.0	3	2.0	1140	492	3.0	N
16934	7518508947	7/7/2021	565000.0	2	2.0	680	402	2.0	N
17564	546000166	6/24/2021	565000.0	2	2.0	930	498	3.0	N
17692	9828701814	9/20/2021	739950.0	2	2.0	1030	482	3.0	N
17920	9297300946	11/14/2021	525000.0	3	2.0	1140	492	3.0	N
19687	7628700519	8/5/2021	515000.0	1	2.0	1080	485	2.0	N
20288	9297300933	10/7/2021	550000.0	3	2.0	1140	492	3.0	N
20888	9297300936	7/29/2021	576300.0	3	2.0	1140	492	3.0	N
21401	7625701306	10/20/2021	499950.0	1	2.0	840	487	3.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
22211	7228501002	2/15/2022	815000.0	2	3.0	1270	474	3.0	N
22752	7234600291	11/21/2021	815000.0	2	2.0	1040	493	3.0	N
24071	4083301623	10/6/2021	680000.0	2	2.0	1020	475	3.0	N
25082	7628700518	7/6/2021	499950.0	1	2.0	1080	485	2.0	N

40 rows × 25 columns

It looks like there's plenty of multistory houses here on small lots. Perhaps this is condos or apartment with a narrow footprint. These look okay so we'll leave this alone.

But! We also notice a few entries NOT located in Kings County, with different addresses and lattitudes and longitudes. We'll save this note for later.

above square footage

Let's start with the lot square footage. We've noticed that a few entries are less than 300 sq.ft. This is tight, so let's take a closer look.

In [18]:

```
#find the number of entries with sqft_above less than 300
kc[kc['sqft_above']<300]
```

Out[18]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
5811	2424049035	8/19/2021	13950000.0	0	1.0	290	178017	1.0	Y
8391	9178601015	11/30/2021	1625000.0	0	1.0	290	4000	1.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
8694	476000125	5/24/2022	1160000.0	0	1.0	260	3500	1.0	N
23622	2872102320	9/1/2021	960000.0	0	1.0	290	5000	1.0	N

4 rows × 25 columns

So... these entries looks small, maybe consistent with a small house or studio arrangement. No bedroom, 1 bath, just under 300 sq. ft. SO, we'll leave them alone.

Numeric Discrete Data

Let's address some of our concerns with numeric_discrete data. This includes the bathroom, bedroom, and floors. I'm not really sure what floors mean, so for now, we'll ignore it.

bedrooms and bathrooms

While we saw earlier there were entries with 0 bedroom, 1 bathroom, that corresponded with maybe a small, studio dwelling. We should double check listings that have 0 bedrooms and 0 bathrooms to see if they resemble empty lots. If they don't look like lots, we should delete them instantly.

In [19]:
```
#find the number of entries with beds and baths less than 1
kc[(kc['bedrooms'] < 1) & (kc['bathrooms'] < 1)]
```

Out[19]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
573	3920030050	5/19/2022	930000.0	0	0.0	1617	2156	3.0	N
1289	2768301406	3/2/2022	1090000.0	0	0.0	1500	1262	3.0	N
1310	3462800015	11/10/2021	360000.0	0	0.0	910	19000	1.0	N
1952	2020069042	9/27/2021	399990.0	0	0.0	1677	43264	1.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
4835	9523101492	1/27/2022	830000.0	0	0.0	1255	983	3.0	N
7545	4318200415	12/17/2021	1225000.0	0	0.0	1940	8893	2.0	N
8338	9265400150	7/20/2021	550000.0	0	0.0	1370	8169	2.0	N
8445	4447300012	9/27/2021	841000.0	0	0.0	1327	875	3.0	N
8749	3920030080	5/25/2022	685000.0	0	0.0	1336	888	3.0	N
12864	4449800002	5/26/2022	700000.0	0	0.0	1215	486	3.0	N
14827	1728800145	9/29/2021	2500000.0	0	0.0	7710	7182	2.5	N
16787	2767701526	7/14/2021	649950.0	0	0.0	1290	715	3.0	N
17536	3920030100	7/26/2021	900000.0	0	0.0	1768	771	3.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
18916	7550800737	7/21/2021	800000.0	0	0.0	1455	646	3.0	N
19920	3920030040	7/7/2021	690000.0	0	0.0	1312	739	3.0	N
20643	1232001191	3/30/2022	1060000.0	0	0.0	1541	649	4.0	N
23914	2767701527	12/1/2021	660000.0	0	0.0	1296	778	3.0	N
25994	3920030010	1/9/2022	939000.0	0	0.0	1768	772	3.0	N
27540	3920030060	12/20/2021	850000.0	0	0.0	1738	2143	3.0	N
28508	4083301619	10/6/2021	654950.0	0	0.0	900	575	3.0	N
29574	2827100079	5/4/2022	610000.0	0	0.0	1530	1392	1.0	N

21 rows × 25 columns

It appears as though there are 21 entries with no bedrooms or bathrooms. These are a little suspicious. One entry, is listed as Missouri. We'll have to delete that one in the addresses, later. The majority of these listings were built in the last 30, 40 years, which would have code requirements about bathrooms for such big spaces. The other data around is looks okay, maybe they are cabins. Let's check

In [20]:
```
#find the grades of entries with beds and baths less than 1
kc[(kc['bedrooms'] < 1) & (kc['bathrooms'] < 1)]['grade']
```

Out[20]:
```
573              8 Good
1289             8 Good
1310      6 Low Average
1952          7 Average
4835          7 Average
7545             8 Good
8338          7 Average
8445          7 Average
8749          7 Average
12864         7 Average
14827            8 Good
16787            8 Good
17536         7 Average
18916            8 Good
19920         7 Average
20643         7 Average
23914            8 Good
25994         7 Average
27540         7 Average
28508         7 Average
29574         7 Average
Name: grade, dtype: object
```

So, we don't see any cabins here, or any below grade places. I'm inclined to delete these as it's hard to tell if they're real.

In [21]:
```
#drop entries with beds and baths less than 1
kc.drop(kc[(kc['bedrooms'] < 1) & (kc['bathrooms'] < 1)].index, inplace=True)
```

Categorical Data

Let's address some of our concerns. We'd like to convert the `grade` column to a number. We'd also like to verify that we only have Kings County addresses in are analysis.

Addresses

Earlier, we found an address that was listed as Missouri, which isn't great. Let's verify that all of addresses appear in Washington state.

In [22]:
```
#create function to to return true if Washington is in the address and false if not
def isWashington (address):
    return ('Washington' in address)

#add column of "Address_Washington" which is True or False
kc['address_washington'] = kc['address'].apply(isWashington)

#inspect entries in column "Address_Washington" that are false
kc[kc['address_washington'] == False]
```

Out[22]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
12	1797501124	6/25/2021	750000.0	3	2.0	1280	964	3.0	N
53	7548300606	5/3/2022	960000.0	3	2.0	1280	1221	2.0	N
62	1934800106	8/24/2021	740000.0	2	2.0	1120	734	3.0	N
159	856000595	7/8/2021	3730000.0	4	4.5	4820	10800	2.0	N
172	1336300219	2/9/2022	759900.0	2	2.0	960	591	3.0	N
...	...	...	...	...	...	...	...	...	
30029	1978200468	10/28/2021	1480000.0	3	2.0	2050	1090	3.5	N
30044	9834201391	2/17/2022	520000.0	2	2.0	790	597	2.0	N
30116	2768100152	1/1/2022	710000.0	1	2.0	1180	616	3.0	N
30129	8584800130	11/18/2021	940000.0	2	2.0	1550	1026	2.5	N
30144	2267000442	12/1/2021	729950.0	2	2.0	1290	720	3.0	N

902 rows × 26 columns

Ookay, we can see about 902 entries do not contain the word, Washington. If we spot check a few of these, we see that nearly all of them are in different states. Sooo.... let's delete them entirely from our dataset. I'm also going to drop the extra column `address_Washington` so we don't have extraneous columns.

In [23]:
```
#drop bad entries from KC
kc.drop(kc[kc['address_washington'] == False].index, inplace=True)

#drop entire column 'Address Washington' because bad entries have been removed
kc.drop('address_washington',axis=1, inplace=True)
```

In [24]:
```
kc['sqft_living'].describe()
```

Out[24]:
```
count    29187.000000
mean      2131.765649
std        976.219778
min        260.000000
25%       1440.000000
50%       1940.000000
75%       2640.000000
max      15360.000000
Name: sqft_living, dtype: float64
```

5: Data Engineering

ZipCode

Let's see if we can extract the zipcode from the address. It looks as if all of our zipcodes belong to Kings County. I'm going to see if we can extract the zipcode from the address.

I'll make a function to take the zipcode from the address provided.

In [25]:
```
#create function to extract 5 digit zipcode from address
def zip98 (address):
    index = 0

#check towards the end of the string to avoid address numbers
    backword = address[-25:]

#check for location of 5 digit zipcode
    if '980' in backword:
        index = backword.find('980')
    if '981' in backword:
        index = backword.find('981')
    if '98224' in backword:
        index = backword.find('98224')
    if '98288' in backword:
        index = backword.find('98288')
    if '98354' in backword:
        index = backword.find('98354')

#return 5 digit zip after locating
    if backword[index:index+5].isdigit():
```

```
        return backword[:40][index:index+5]

#add new column 'zipcode' with five digit integer
kc['zipcode'] = kc['address'].apply(zip98)
kc
```

Out[25]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.0	N
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.0	N
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.0	N
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.0	N
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.0	N
...	...	...	...	...	...	...	...	...	
30150	7834800180	11/30/2021	1555000.0	5	2.0	1910	4000	1.5	N
30151	194000695	6/16/2021	1313000.0	3	2.0	2020	5800	2.0	N
30152	7960100080	5/27/2022	800000.0	3	2.0	1620	3600	1.0	N
30153	2781280080	2/24/2022	775000.0	3	2.5	2570	2889	2.0	N

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfror
30154	9557800100	4/29/2022	500000.0	3	1.5	1200	11058	1.0	N

29187 rows × 26 columns

That looks good. Let see the zipcode counts to see how many we're working with.

In [26]:
```
#let's count values of the new column
kc['zipcode'].value_counts()
```

Out[26]:
```
98042    992
98038    857
98103    759
98115    754
98117    745
        ... 
98039     59
98354     23
98288     16
98224      3
98050      2
Name: zipcode, Length: 77, dtype: int64
```

Great, it looks like we have 77 zipcodes here. That's a lot. We'll... see where this goes, from 992 addresses in one zip code to 2, in the smallest zipcode.

'Grade'

Let's convert grade to a numeric value that can be used as continuous data. From our value_counts previously, we see that all of the grades have consistent formatting, with the number first, followed by a space, and then characters. Let's convert this to a string.

In [27]:
```
#create function to convert string to integer with one number
def intconvert (grade):
    return int(grade[0:2].strip(' '))

#create new column `intgrade` which had grade column converted to integers
kc['intgrade'] = kc['grade'].apply(intconvert)
kc['intgrade']
```

Out[27]:
```
0        7
1        7
2        7
3        9
4        7
        ..
30150    8
30151    7
30152    7
30153    8
```

```
30154    7
Name: intgrade, Length: 29187, dtype: int64
```

'Year'

Let's convert the year so we could measure it with more accuracy. For instance, it looks like the oldest house was built in 1900, so we could subtract our `yr_built` column by 1900 so that we can compare the houses with eacother.

In [28]:
```
#make new column by subtracting 1900 from `yr-built`
kc['yr_built_transform'] = kc['yr_built'] - 1900
kc['yr_built_transform']
```

Out[28]:
```
0         69
1         50
2         56
3        110
4        112
        ...
30150     21
30151    111
30152     95
30153    106
30154     65
Name: yr_built_transform, Length: 29187, dtype: int64
```

Perfect. I think we're good here.

'Date'

So, we know that the market varies from year to year and season to season. This means that time - year and month - probably do have some effect on the price. So let's investigate.

In [29]:
```
#create function to convert sales date to a year
def year_convert (salesdate):
    return salesdate[-4:]

#create column with the year of the salesdate
kc['sales_year'] = kc['date'].apply(year_convert)
kc['sales_year'].value_counts()
```

Out[29]:
```
2021    18642
2022    10545
Name: sales_year, dtype: int64
```

OKay, so these is just 2021 to 2022 data. Let's see what the month's reveal.

In [30]:
```
#create function that returns the month as an integer of the salesdate
def month_convert (salesdate):
    return int(salesdate[0:2].strip('/'))

#create column with the month of the sales date as an integer
kc['sales_month'] = kc['date'].apply(month_convert)
kc['sales_month'].value_counts()
```

Out[30]:
```
7     3173
8     3163
9     2828
10    2735
```

```
6      2686
4      2560
3      2528
5      2484
11     2451
12     1891
2      1557
1      1131
Name: sales_month, dtype: int64
```

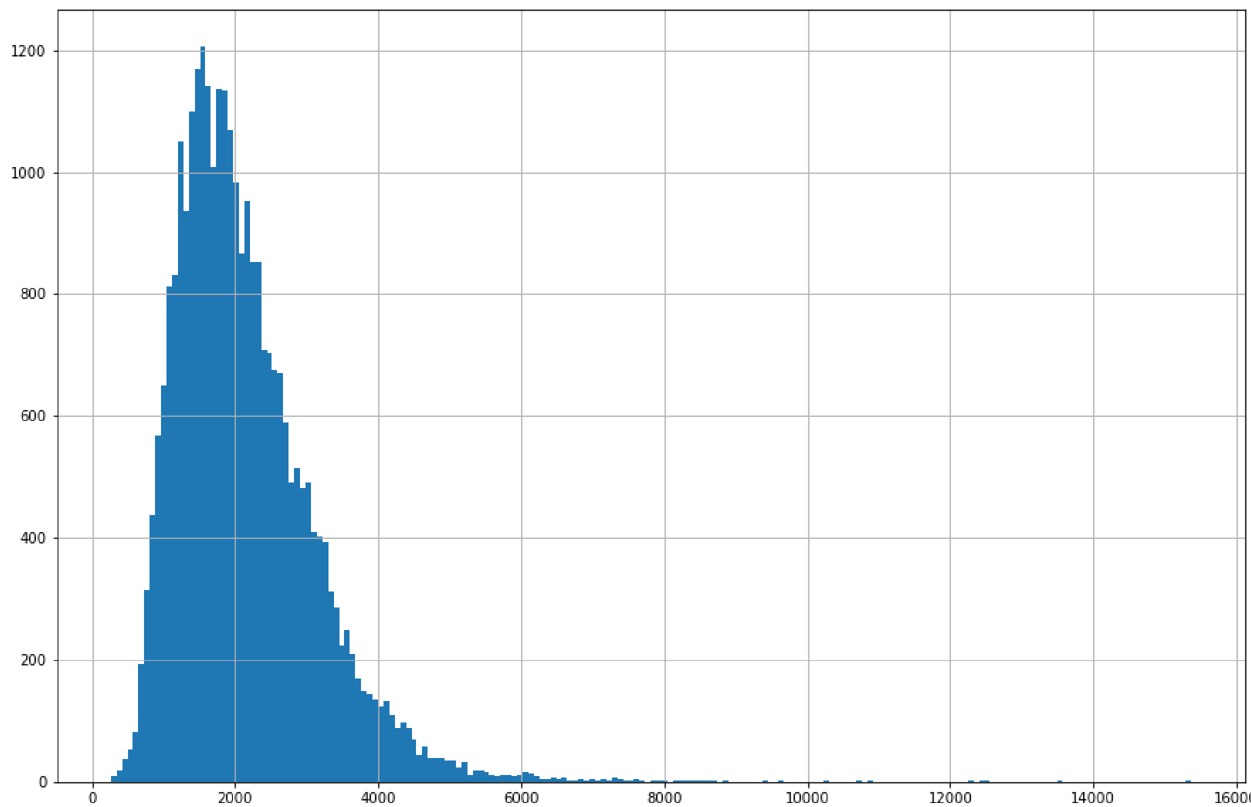
Good, I think we successfully cleaned the date to get months and years.

Mansion Outlier

We've tried to eliminate entries that we didn't think were real. Now, we're going to look at outlier data.

Generally, the highest 3% are considered outliers. But let's look at plots of both square footage and price.

In [31]:
```
#create histogram of the square footage of all homes sold
kc['sqft_living'].hist(figsize=(15,10), bins="auto");
```



As we can see above, it looks like normal curve with a slight left skew. There's also a long tail, where it appears to contain some exceptionally large size homes.

In [32]:
```
#look at stats of square footage
kc['sqft_living'].describe()
```

Out[32]:
```
count    29187.000000
mean      2131.765649
std        976.219778
min        260.000000
```

```
25%       1440.000000
50%       1940.000000
75%       2640.000000
max      15360.000000
Name: sqft_living, dtype: float64
```

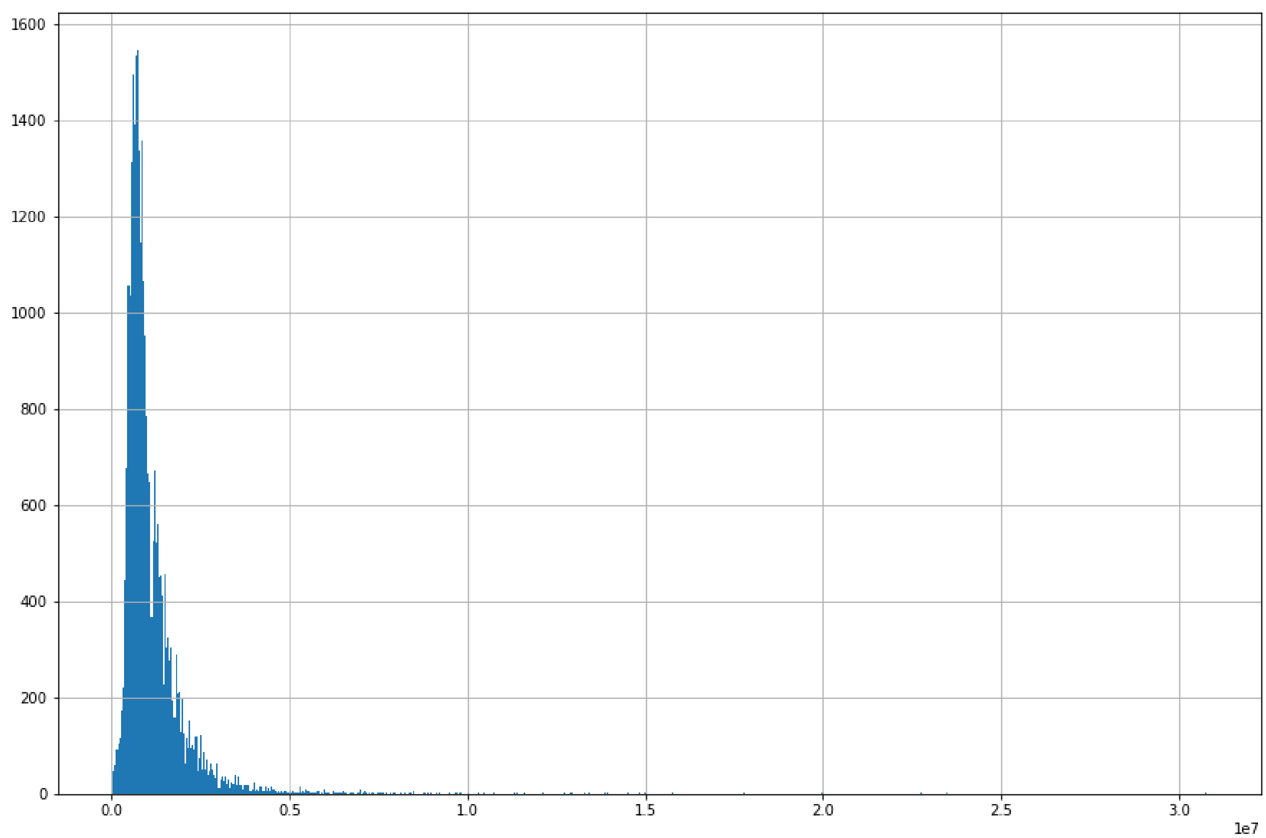
We can see here the maximum is quite high compared to the 75% percentile. And we have quite a tail. Let's see what the 99th percentile is.

In [33]:
```python
#import numpy for stats packages
import numpy as np

#determine outlier
sqft_outlier = np.percentile(kc['sqft_living'], 99)
sqft_outlier
```

Out[33]:
```
5190.0
```

Now, let's took a look at the price.

In [34]:
```python
#create histogram of the prices of all homes sold
kc['price'].hist(figsize=(15,10), bins="auto");
```



As we can see above, it looks like normal curve with a slight left skew. There's also a long tail, where it appears to contain some exceptionally expensive homes.

In [35]:
```python
#look at stats of price
kc['price'].describe()
```

Out[35]:
```
count    2.918700e+04
mean     1.113111e+06
std      8.955589e+05
min      2.736000e+04
```

```
25%        6.450000e+05
50%        8.680000e+05
75%        1.310000e+06
max        3.075000e+07
Name: price, dtype: float64
```

AS with `sqft_living`, the maximum is quite high compared to the 75% percentile. And we have quite a tail. Let's see what the 98th percentile is.

In [36]:
```
#find 99th percentile home
price_outlier = np.percentile(kc['price'], 99)
price_outlier
```

Out[36]:
```
4300000.0
```

The 99th percentile for homes is 4.3M. This is about 14% of our maximum, which was 30M.

Let's go ahead and get rid of the outliers above 99th percentile for both for each variable

In [37]:
```
#drop 99th percentile home by sqft
kc.drop(kc[kc['sqft_living'] > sqft_outlier].index, inplace=True)

#show updates stats
kc['sqft_living'].describe()
```

Out[37]:
```
count    28899.000000
mean      2089.673276
std        874.068860
min        260.000000
25%       1440.000000
50%       1940.000000
75%       2610.000000
max       5190.000000
Name: sqft_living, dtype: float64
```

We went from 29,187 entries to 28,899, with our max being 5,190 sq. ft. This looks reasonable and we will note to our audience that our analysis only includes houses less than 5200 sq. ft.

In [38]:
```
#drop 99th percentile home by price
kc.drop(kc[kc['price'] > price_outlier].index, inplace=True)

#show updates stats
kc['price'].describe()
```

Out[38]:
```
count    2.872600e+04
mean     1.046535e+06
std      6.156467e+05
min      2.736000e+04
25%      6.400000e+05
50%      8.600000e+05
75%      1.295000e+06
max      4.300000e+06
Name: price, dtype: float64
```

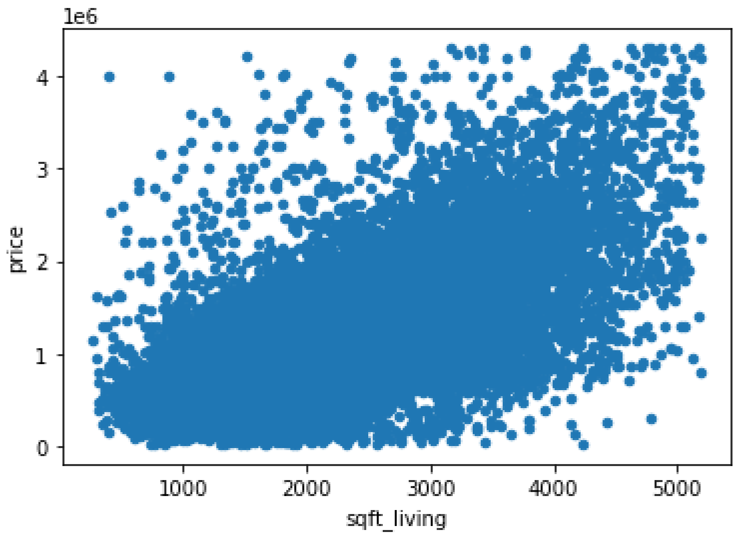
This looks good. We went from 28,899 entries to 28,726, with our max being 3.45M. We will note to our audience that our analysis doesn't include houses more than 3.45

6 - Start Modeling the Baseline

Let's plot sqft vs price to see if it's a good candidate for linear regression.

In [39]:
```
#show scatter plot of price vs. sq.ft
kc.plot.scatter("sqft_living","price")
```

Out[39]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



So, it looks like we can see some linear relationship - as sqft_living goes up, so does the price. It's clearly not a 1 to 1, but let's do a deep dive into the data and see what's what.

So, I'm going to run a linear regression to see what we get.

In [40]:
```
#import stats model functions for linear regression creation
import statsmodels.api as sm

#specify X and Y
X = kc['sqft_living']
y = kc['price']

#create model
baseline_model = sm.OLS(y, sm.add_constant(X))
baseline_results = baseline_model.fit()

#print results
print(baseline_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.393
Model:                            OLS   Adj. R-squared:                  0.393
Method:                 Least Squares   F-statistic:                 1.862e+04
Date:                Thu, 07 Sep 2023   Prob (F-statistic):               0.00
Time:                        11:58:55   Log-Likelihood:            -4.1651e+05
No. Observations:               28726   AIC:                         8.330e+05
Df Residuals:                   28724   BIC:                         8.330e+05
Df Model:                           1
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
const        1.155e+05   7386.375     15.636      0.000    1.01e+05     1.3e+05
sqft_living   447.5556      3.280    136.456      0.000     441.127     453.984
==============================================================================
Omnibus:                     7462.167   Durbin-Watson:                   1.984
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            29178.588
```

```
Skew:                           1.250   Prob(JB):                        0.00
Kurtosis:                       7.257   Cond. No.                    5.88e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 5.88e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Results Discussion

The model overall explains about 39% of the variance in sale price, which is low.

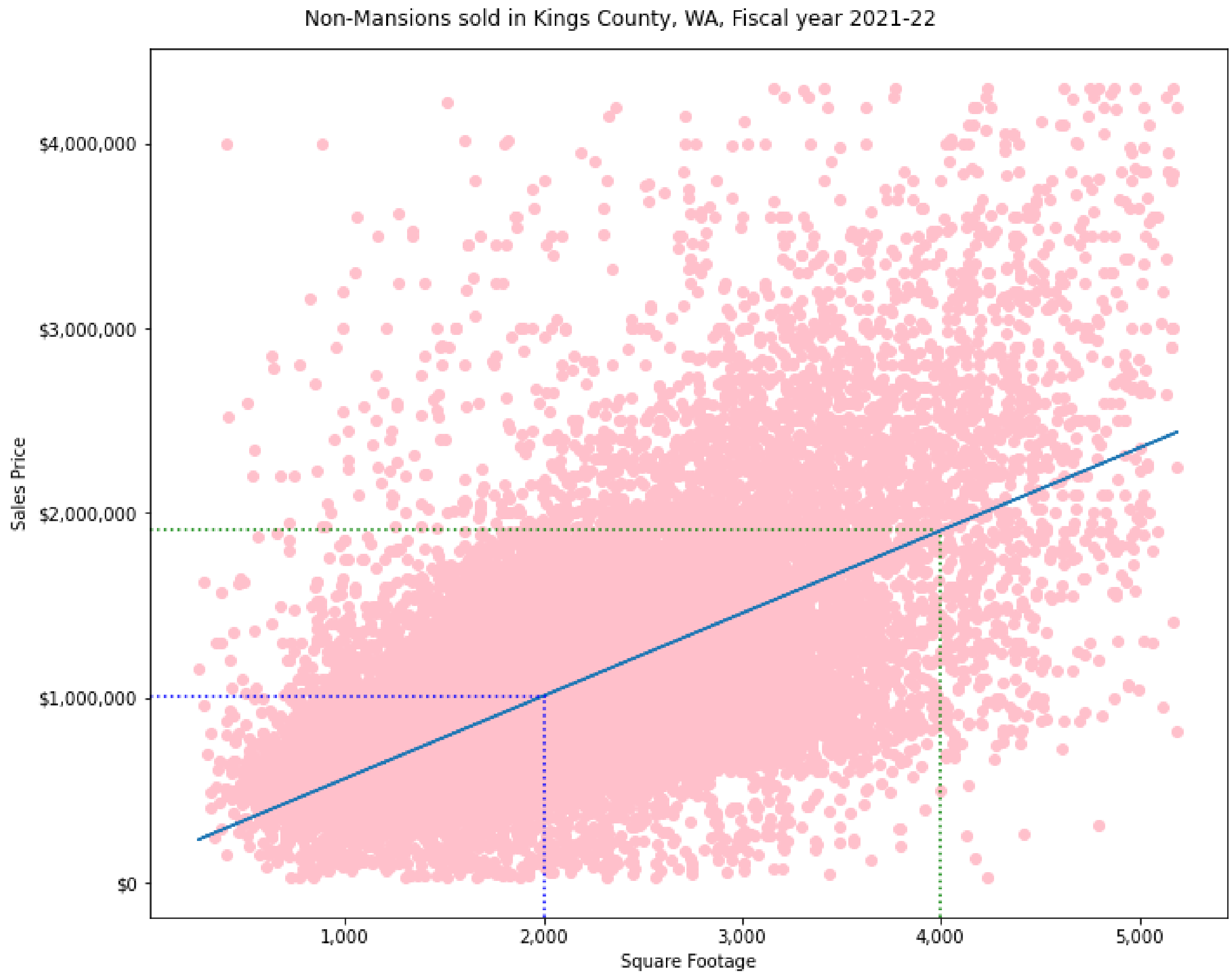
Using the standard alpha of 0.05 to evaluate statistical significance:

Coefficients for the intercept as well sqft_living are all statistically significant. According to the model, houses are selling at approximately $447/sq. ft.

The coefficients for the intercept is positive. What does that mean? Well, for a house with 0 sq ft, it would sell for $115,000. Let's see how this looks on the plot.

In [41]:

```python
#import matplotlib to plot
import matplotlib.pyplot as plt

#setup plot
fig, ax = plt.subplots(figsize = (10,8))

#show the best-fit line from the model overlaid on the scatter plot
ax.scatter(X, y, color='pink')
ax.plot(X, baseline_results.predict(sm.add_constant(X)))

#create vertical and horizontal lines showing 2,000 and 4,000 sq ft home
ax.axvline (x = 2000, color = "blue", linestyle = ":", ymax=.25)
ax.axhline (y = 1009000, color = "blue", linestyle = ":", xmax = .37)
ax.axvline (x = 4000, color = "green", linestyle = ":", ymax=.44)
ax.axhline (y = 1910000, color = "green", linestyle = ":", xmax = .73)

#format plot
ax.set_xlabel('Square Footage')
ax.set_ylabel('Sales Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('{x:,.0f}')
fig.suptitle("Non-Mansions sold in Kings County, WA, Fiscal year 2021-22");
fig.tight_layout()
```



We can see with our data that we have a reasonable plot, but there's still quite a bit of variance. With houses are selling at approximately USD 447/sq. ft, we can see that a 2,000 sq. ft home would go for nearly 1M. Wowsers. A 4,000 sq. ft home would sell for approximately 2M.

7. Refining and Iterating for Additional Factors, like ZipCode.

Let's add our categorical data. Our primary goal with this analysis is to include zipcode. We know that

Zipcode analysis.

To create the one-hot encoding with the categorical data for zipcode. We'll combine the square footage and zipcode columns and do one-hot encoding with dummy values.

In [42]:
```
#create data set for one hot encoding of zipcode
zip_sq_ft = ['sqft_living', 'zipcode']

X_zip_sq_ft = kc[zip_sq_ft]

#one hot encode using .getdummies
X_zip_sq_ft_dummy = pd.get_dummies(X_zip_sq_ft, columns=['zipcode'])
```

```python
#let's drop zipcode 98070 as it is the most common zipcode
X_zip_sq_ft_dummy.drop('zipcode_98070', axis = 1, inplace = True)

#let's run our model
sq_ft_zip_model = sm.OLS(y, sm.add_constant(X_zip_sq_ft_dummy))
sq_ft_zip_results = sq_ft_zip_model.fit()

print(sq_ft_zip_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.685
Model:                            OLS   Adj. R-squared:                  0.684
Method:                 Least Squares   F-statistic:                     809.2
Date:                Thu, 07 Sep 2023   Prob (F-statistic):               0.00
Time:                        11:59:02   Log-Likelihood:            -4.0710e+05
No. Observations:               28726   AIC:                         8.143e+05
Df Residuals:                   28648   BIC:                         8.150e+05
Df Model:                          77
Covariance Type:            nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const          2.952e+05   2.35e+04     12.549      0.000    2.49e+05    3.41e+05
sqft_living     374.5289      2.603    143.888      0.000     369.427     379.631
zipcode_98001 -4.403e+05   2.69e+04    -16.387      0.000   -4.93e+05   -3.88e+05
zipcode_98002 -3.978e+05   2.93e+04    -13.565      0.000   -4.55e+05    -3.4e+05
zipcode_98003 -4.353e+05   2.83e+04    -15.362      0.000   -4.91e+05    -3.8e+05
zipcode_98004  1.257e+06   3.28e+04     38.350      0.000    1.19e+06    1.32e+06
zipcode_98005  7.265e+05   3.55e+04     20.471      0.000    6.57e+05    7.96e+05
zipcode_98006  4.341e+05   2.78e+04     15.589      0.000     3.8e+05    4.89e+05
zipcode_98007  2.899e+05   3.62e+04      8.016      0.000    2.19e+05    3.61e+05
zipcode_98008  3.684e+05   2.89e+04     12.745      0.000    3.12e+05    4.25e+05
zipcode_98010 -4.246e+05   3.03e+04    -14.034      0.000   -4.84e+05   -3.65e+05
zipcode_98011   2.44e+04   3.15e+04      0.775      0.438   -3.73e+04    8.61e+04
zipcode_98014 -1.773e+05   3.62e+04     -4.903      0.000   -2.48e+05   -1.06e+05
zipcode_98019 -1.922e+05    3.2e+04     -6.000      0.000   -2.55e+05   -1.29e+05
zipcode_98022 -3.754e+05   2.82e+04    -13.329      0.000   -4.31e+05    -3.2e+05
zipcode_98023 -4.449e+05   2.65e+04    -16.775      0.000   -4.97e+05   -3.93e+05
zipcode_98024  3.452e+04   4.09e+04      0.844      0.399   -4.56e+04    1.15e+05
zipcode_98027  9.511e+04   2.92e+04      3.261      0.001    3.79e+04    1.52e+05
zipcode_98028  -7.23e+04   2.96e+04     -2.445      0.014    -1.3e+05   -1.43e+04
zipcode_98029  2.491e+05   3.03e+04      8.234      0.000     1.9e+05    3.08e+05
zipcode_98030 -4.243e+05   2.91e+04    -14.570      0.000   -4.81e+05   -3.67e+05
zipcode_98031 -4.062e+05   2.75e+04    -14.777      0.000    -4.6e+05   -3.52e+05
zipcode_98032   -3.7e+05   3.44e+04    -10.754      0.000   -4.37e+05   -3.03e+05
zipcode_98033  7.131e+05   2.71e+04     26.295      0.000     6.6e+05    7.66e+05
zipcode_98034  1.626e+05   2.66e+04      6.118      0.000     1.1e+05    2.15e+05
zipcode_98038 -3.218e+05   2.59e+04    -12.426      0.000   -3.73e+05   -2.71e+05
zipcode_98039  1.873e+06   6.72e+04     27.858      0.000    1.74e+06    2.01e+06
zipcode_98040  8.653e+05   3.05e+04     28.409      0.000    8.06e+05    9.25e+05
zipcode_98042 -4.244e+05   2.55e+04    -16.635      0.000   -4.74e+05   -3.74e+05
zipcode_98045 -1.473e+05   2.81e+04     -5.240      0.000   -2.02e+05   -9.22e+04
zipcode_98047 -4.064e+05   4.59e+04     -8.859      0.000   -4.96e+05   -3.17e+05
zipcode_98050  2.604e+05   2.46e+05      1.060      0.289   -2.21e+05    7.42e+05
zipcode_98051 -2.545e+05   4.87e+04     -5.225      0.000    -3.5e+05   -1.59e+05
zipcode_98052  3.728e+05   2.73e+04     13.677      0.000    3.19e+05    4.26e+05
zipcode_98053  2.368e+05   2.91e+04      8.130      0.000     1.8e+05    2.94e+05
zipcode_98055 -3.662e+05   3.28e+04    -11.149      0.000   -4.31e+05   -3.02e+05
zipcode_98056 -1.288e+05   2.76e+04     -4.666      0.000   -1.83e+05   -7.47e+04
zipcode_98057 -2.912e+05   3.85e+04     -7.570      0.000   -3.67e+05   -2.16e+05
zipcode_98058 -3.352e+05   2.66e+04    -12.616      0.000   -3.87e+05   -2.83e+05
zipcode_98059 -1.513e+05   2.72e+04     -5.566      0.000   -2.05e+05    -9.8e+04
zipcode_98065 -4.889e+04   3.07e+04     -1.591      0.112   -1.09e+05    1.14e+04
```

```
zipcode_98072  1.127e+05      3e+04      3.763      0.000     5.4e+04    1.71e+05
zipcode_98074  3.522e+05   2.87e+04     12.292      0.000    2.96e+05    4.08e+05
zipcode_98075  3.941e+05   2.89e+04     13.616      0.000    3.37e+05    4.51e+05
zipcode_98077  1.917e+05   3.24e+04      5.925      0.000    1.28e+05    2.55e+05
zipcode_98092 -4.506e+05    2.7e+04    -16.694      0.000   -5.04e+05   -3.98e+05
zipcode_98102  3.953e+05   3.71e+04     10.641      0.000    3.22e+05    4.68e+05
zipcode_98103  1.546e+05   2.62e+04      5.897      0.000    1.03e+05    2.06e+05
zipcode_98105  2.555e+05   3.01e+04      8.492      0.000    1.97e+05    3.14e+05
zipcode_98106 -1.978e+05   2.74e+04     -7.220      0.000   -2.52e+05   -1.44e+05
zipcode_98107  1.479e+05   2.82e+04      5.252      0.000    9.27e+04    2.03e+05
zipcode_98108 -1.972e+05   3.04e+04     -6.487      0.000   -2.57e+05   -1.38e+05
zipcode_98109  3.912e+05   3.85e+04     10.167      0.000    3.16e+05    4.67e+05
zipcode_98112  4.709e+05   3.08e+04     15.293      0.000    4.11e+05    5.31e+05
zipcode_98115  1.703e+05   2.63e+04      6.486      0.000    1.19e+05    2.22e+05
zipcode_98116  1.256e+05    2.9e+04      4.332      0.000    6.88e+04    1.82e+05
zipcode_98117  1.275e+05   2.63e+04      4.854      0.000     7.6e+04    1.79e+05
zipcode_98118 -1.131e+05    2.7e+04     -4.185      0.000   -1.66e+05   -6.01e+04
zipcode_98119    3.5e+05   3.22e+04     10.878      0.000    2.87e+05    4.13e+05
zipcode_98122   1.73e+05   2.83e+04      6.101      0.000    1.17e+05    2.29e+05
zipcode_98125 -1.785e+04   2.78e+04     -0.641      0.521   -7.24e+04    3.67e+04
zipcode_98126 -8.999e+04   2.83e+04     -3.176      0.001   -1.46e+05   -3.45e+04
zipcode_98133 -1.238e+05   2.67e+04     -4.643      0.000   -1.76e+05   -7.15e+04
zipcode_98136  6.512e+04   3.05e+04      2.134      0.033    5304.153    1.25e+05
zipcode_98144  6.359e+04   2.84e+04      2.240      0.025    7938.400    1.19e+05
zipcode_98146 -1.927e+05   2.85e+04     -6.763      0.000   -2.49e+05   -1.37e+05
zipcode_98148 -3.671e+05   4.26e+04     -8.614      0.000   -4.51e+05   -2.84e+05
zipcode_98155 -6.843e+04   2.76e+04     -2.476      0.013   -1.23e+05   -1.43e+04
zipcode_98166 -1.903e+05   2.97e+04     -6.415      0.000   -2.48e+05   -1.32e+05
zipcode_98168 -3.332e+05    2.9e+04    -11.476      0.000    -3.9e+05   -2.76e+05
zipcode_98177  8.856e+04   3.06e+04      2.892      0.004    2.85e+04    1.49e+05
zipcode_98178 -2.914e+05    2.9e+04    -10.035      0.000   -3.48e+05   -2.34e+05
zipcode_98188 -3.506e+05   3.36e+04    -10.440      0.000   -4.16e+05   -2.85e+05
zipcode_98198 -3.476e+05   2.85e+04    -12.195      0.000   -4.04e+05   -2.92e+05
zipcode_98199  3.388e+05   2.88e+04     11.768      0.000    2.82e+05    3.95e+05
zipcode_98224 -3.738e+05   2.01e+05     -1.859      0.063   -7.68e+05    2.03e+04
zipcode_98288 -3.695e+05   8.95e+04     -4.128      0.000   -5.45e+05   -1.94e+05
zipcode_98354 -3.969e+05   7.57e+04     -5.241      0.000   -5.45e+05   -2.48e+05
==============================================================================
Omnibus:                     9383.691   Durbin-Watson:                   1.958
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            96184.727
Skew:                           1.274   Prob(JB):                         0.00
Kurtosis:                      11.595   Cond. No.                     2.74e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 2.74e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

OKay, so what does this model show? We've got improvements in our Adjusted R-Square, which is now showing a value of 68.4% of all variance accounted for. This is better.

Using the standard alpha of 0.05 to evaluate statistical significance:

Coefficients for sqft_living and most of our zipcodes are statistically significant. Our baseline zipcode is 98070. It seems that relative to our baseline, the zipcodes do have a statistically significant effect on price, except for zipcodes 98224, 98113, 98118, 98027, and 98011). So...

According to the model, houses are selling at approximately $374/sq. ft.

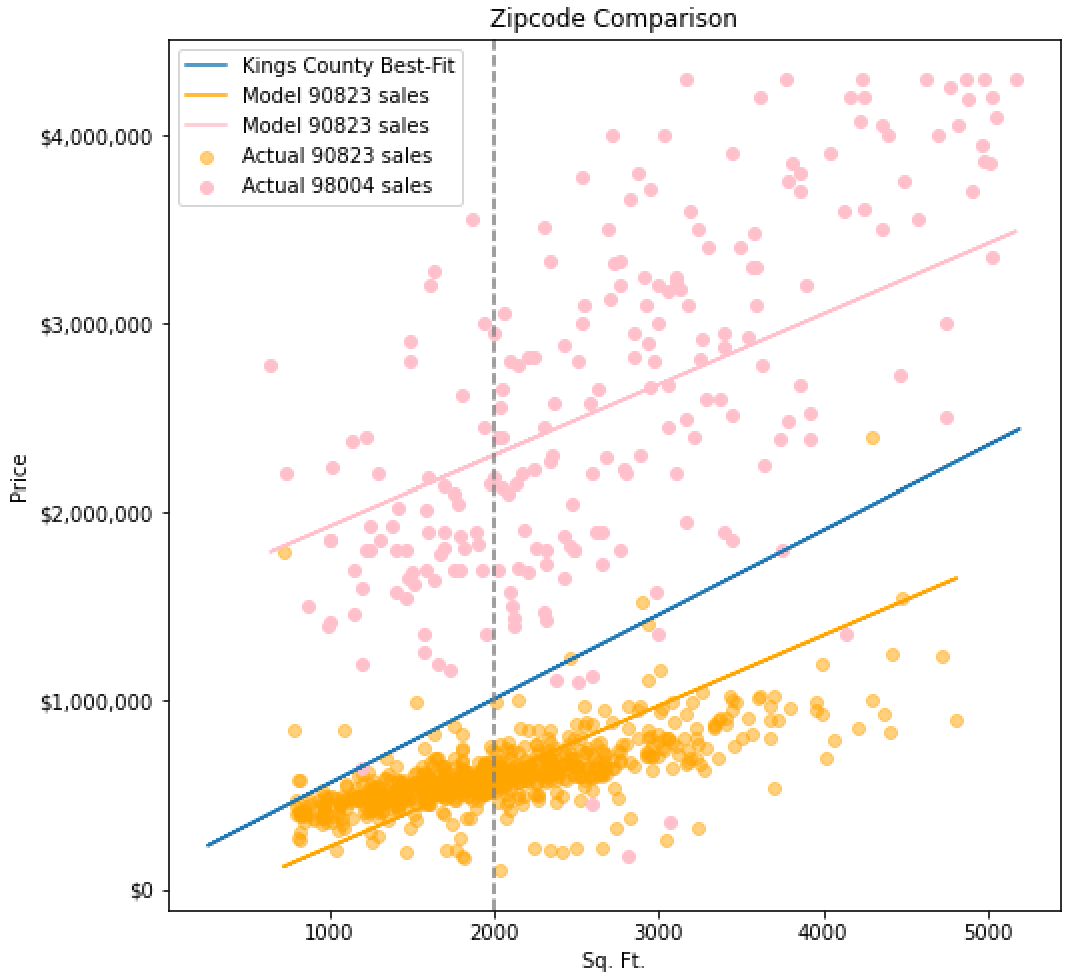
The coefficients for the intercept is 295,200. That means that, when not accounting for square footage or zipcode, you could assume a house will sell for 295,200. That's quite a premium.

The zipcode with the largest effect is zipcode 98004, 98005, 98039, and 98040. Zipcode 98065, 98092, 98001, 98003, 98023, have the most negative effect on pricing.

Let's go ahead an plot an example to show the effect. For example, I'll go ahead and pick two zipcodes, 98004 and 98023, that have a positive and negative impact, respectively.

In [43]:
```
#create plot to show two separate zipcodes compared to our original best fit

fig, ax = plt.subplots(figsize = (8,8))

#plot original best fit-line from first linear regression
ax.plot(X, baseline_results.predict(sm.add_constant(X)), label = "Kings County Best-Fit

#and now, scatter plot one of the lest expensive zipcodes
X_graph = X_zip_sq_ft_dummy[X_zip_sq_ft_dummy['zipcode_98023'] ==1]['sqft_living']
Y_graph = X_graph.to_frame().join(kc['price'],how="inner")
ax.scatter(X_graph, Y_graph['price'], color='orange', label = 'Actual 90823 sales', alp

#and now, model plot one of the lest expensive zipcodes
slope = sq_ft_zip_results.params["sqft_living"]
intercept = sq_ft_zip_results.params["const"]
zipeffect_90023 = sq_ft_zip_results.params["zipcode_98023"]
fit_line_90023 = slope * X_graph + intercept + zipeffect_90023
ax.plot(X_graph, fit_line_90023, color = 'orange', label = 'Model 90823 sales')

#and now, one of the more expensive zipcodes
X_graph = X_zip_sq_ft_dummy[X_zip_sq_ft_dummy['zipcode_98004'] ==1]['sqft_living']
Y_graph = X_graph.to_frame().join(kc['price'],how="inner")
ax.scatter(X_graph, Y_graph['price'], color='pink', label = 'Actual 98004 sales')
#ax.plot(X_graph, sq_ft_zip_results.predict(sm.add_constant(X_graph)), color='pink')

#and now, model plot one of the lest expensive zipcodes
slope = sq_ft_zip_results.params["sqft_living"]
intercept = sq_ft_zip_results.params["const"]
zipeffect_98004 = sq_ft_zip_results.params["zipcode_98004"]
fit_line_90004 = slope * X_graph + intercept + zipeffect_98004
ax.plot(X_graph, fit_line_90004, color = 'pink', label = 'Model 90823 sales')

#make vertical and horizontal lines showing 2,000 sq ft homes in each zipcode using har
ax.axvline (x = 2000, color = "gray", linestyle = "dashed")

#format graph
ax.set_xlabel('Sq. Ft.')
ax.set_ylabel('Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.set_title("Zipcode Comparison");
ax.legend()
```

Out[43]:
```
<matplotlib.legend.Legend at 0x1207088be80>
```

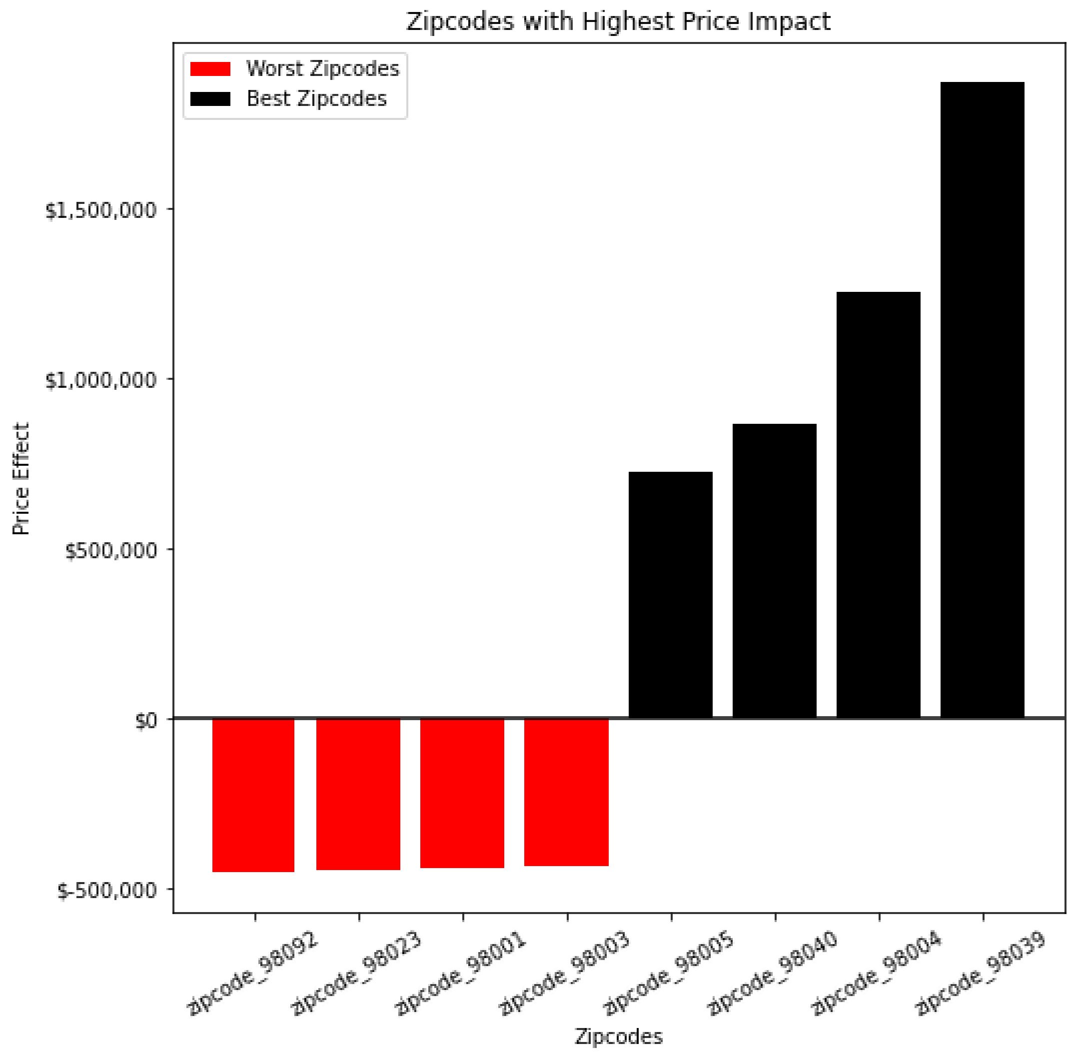


Let's see a list of the most impactful zicodes.

In [44]:
```
#first, establish lists of the 4 most impactful zipcodes, both negative and positive

#from the zipcode list, create
best_worst_zips = sq_ft_zip_results.params
zipslist = best_worst_zips.sort_values(ascending=True)
zipslist1 = zipslist.drop(zipslist.index[4:78])
zipslist2 = zipslist.drop(zipslist.index[0:74])
```

In [45]:
```
#establish plot
fig, ax = plt.subplots(figsize = (8,8))

#create bar graphs with negative impacts in red and positive in black
ax.bar(x=zipslist1.index, height=zipslist1.values, color='red', label = 'Worst Zipcodes
ax.bar(x=zipslist2.index, height=zipslist2.values, color='black', label = 'Best Zipcode

#format graphs
ax.set_xlabel('Zipcodes')
ax.set_ylabel('Price Effect')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.set_title("Zipcodes with Highest Price Impact");
ax.tick_params(axis='x', labelrotation = 30)
ax.axhline (y = 0, color = "black")
ax.legend()
```

Out[45]: <matplotlib.legend.Legend at 0x120707e6c10>



So we can see here how the zipcode does effect whether it's above or below the "mean" square footage line that we spoke of earlier.

For zipcode 90004, the prices are on average 1.25M higher than the mean, which for now we assume to be Zipcode 98070. For the 2,000 sq. ft home example, or "mean" home would be 1M, but our price, according to the model, would be 2.25M.

Vice Versa, for zipcode 908023, the prices are on average are 449,000 less than our typical zipcode. As you can see, our 2,000 sq.ft. home would now cost 550,000

9. Refining and Iterating for Additional Factors.

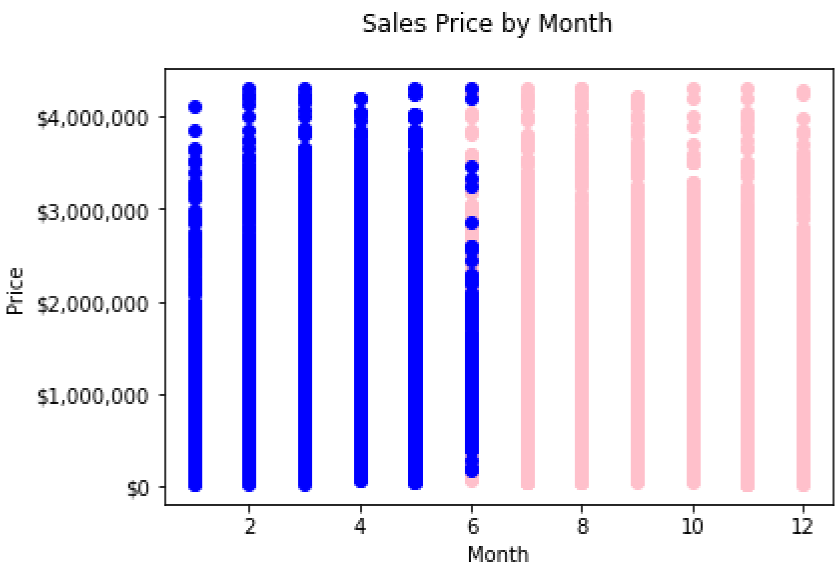
Closing Date.

Another important factor is the time of year that a house gets sold. We know informally that houses don't sell as well when the weather is bad and that prices change with the year. Let's see how much they differ.

I'm going to plot the prices overall and see what emerges

In [46]:

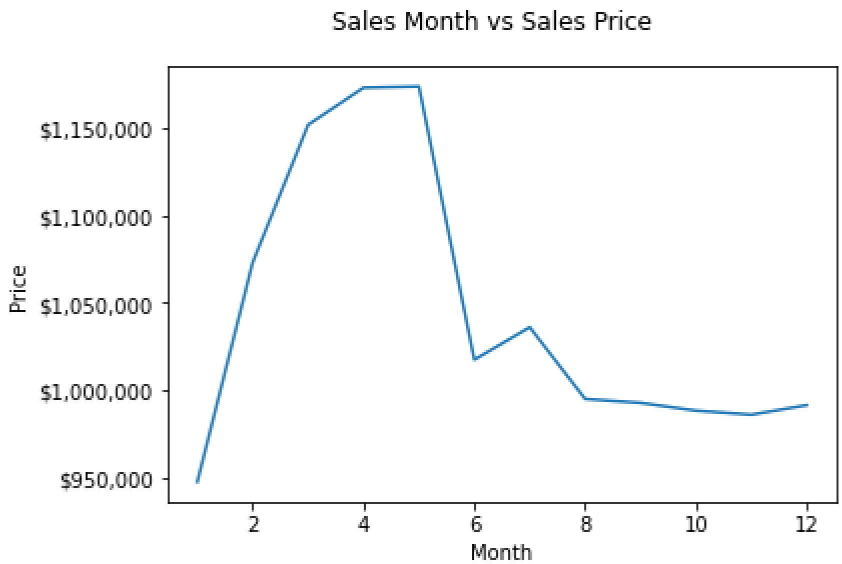
```
#establish plot to observe the sales months
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

#create datasets for each year
year2021 = kc[kc['sales_year']=='2021']
year2022 = kc[kc['sales_year']=='2022']

#scatter plot for each year by month
ax.scatter(year2021['sales_month'], year2021['price'], color='pink', label='2021')
ax.scatter(year2022['sales_month'], year2022['price'], color='blue', label='2022')

#format graph
ax.set_xlabel('Month')
ax.set_ylabel('Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
fig.suptitle("Sales Price by Month");
```



Okay, it looks as though we only have sales data for 1 year. That means the sales year won't be a factor in our analysis. Let's take a look at monthly mean data and see if that reveals anything.

In [47]:

```
#establish line plot for median sales by month
fig, ax = plt.subplots()

#create line plot for median sales by month
ax.plot(kc.groupby(['sales_month'])['price'].mean())

#format graph
ax.set_xlabel('Month')
ax.set_ylabel('Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
fig.suptitle("Sales Month vs Sales Price");
```



Okay, so we can see a clear trend here. With our peak season occuring in May and our low happening in January. This makes sense. This is also going to be a good candidate for our one-hot encoding, because while there is a trend, there's no LINEAR RELATIONSHIP here.

So, let's un an analysis to factor the month of the year. We'll get rid of the month of June, as it appears to occur sort of in the middle.

In [48]:
```
#establish data set and one hot encode sales month
X_combined_sales_month = X_zip_sq_ft_dummy.join(kc['sales_month'], how="inner")
X_combined_sales_month = pd.get_dummies(X_combined_sales_month, columns=['sales_month']

#let's drop zipcode 'sales_month_6' as it seems to have the average prices
X_combined_sales_month.drop('sales_month_7', axis = 1, inplace = True)

#create linear regression model
X_combined_sales_month_model = sm.OLS(y, sm.add_constant(X_combined_sales_month))
X_combined_sales_month_results = X_combined_sales_month_model.fit()
```

Additional Numeric Factors

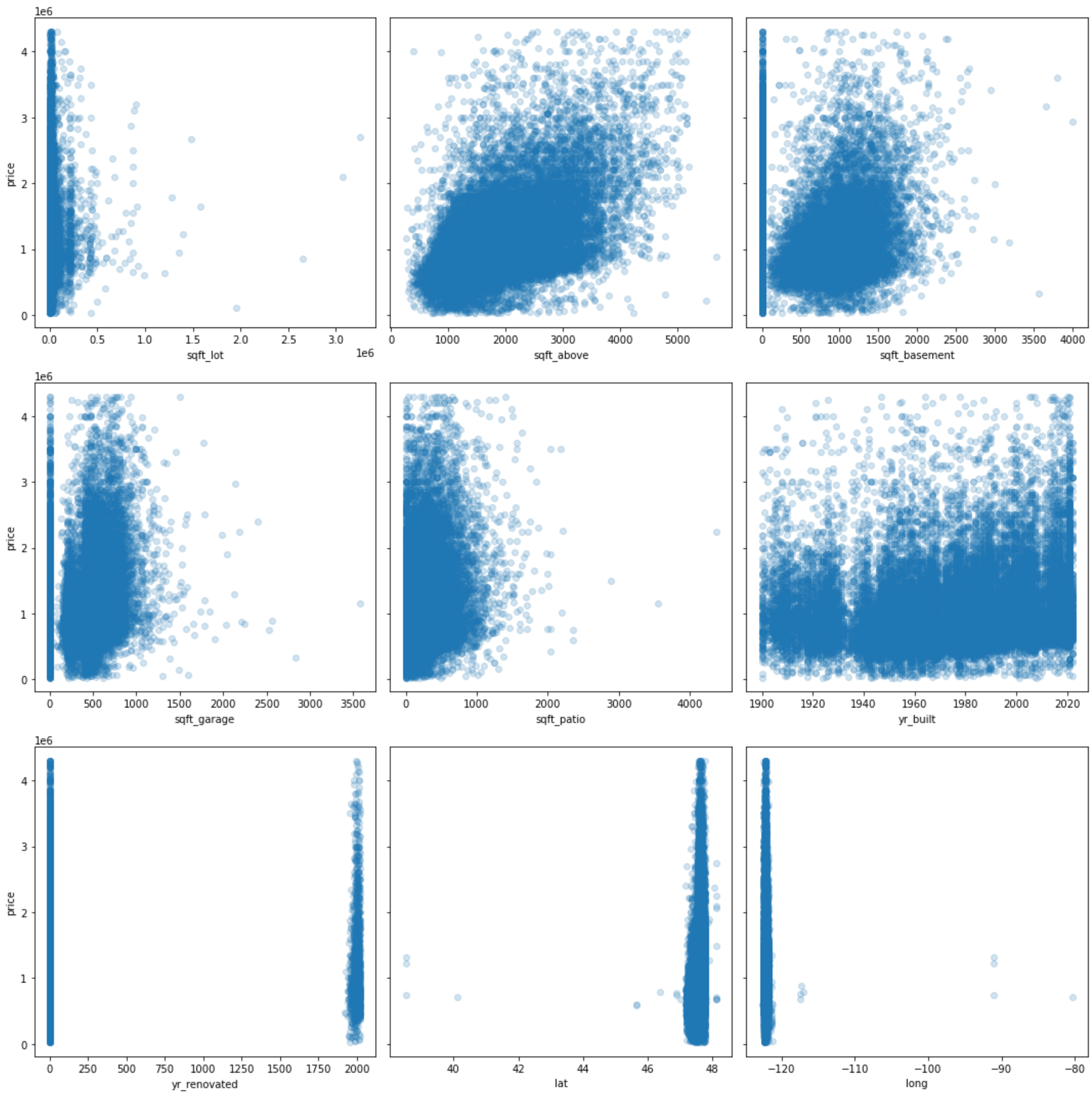
So, we have a decent model with regard to Adjusted R-square, but let's see if we can do better. Let's examine...

Linearity

I'm going to plot some independent variables and see if they have any effect on our target, or dependent variable.

In [49]:
```
#establish plot to compare various variables for linearity
import matplotlib.pyplot as plt
import numpy as np

#create datasets
X = kc[numeric_cont].drop("date", axis=1)
X = X.drop("price", axis=1)
X = X.drop("sqft_living", axis=1)
```

```python
#setup plot
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15,15), sharey=True)

#iteratively plot each column vs price and plot as scatter
for i, column in enumerate(X.columns):
    # Locate applicable axes
    row = i // 3
    col = i % 3
    ax = axes[row][col]

    # Plot feature vs. y and label axes
    ax.scatter(X[column], y, alpha=0.2)
    ax.set_xlabel(column)
    if col == 0:
        ax.set_ylabel("price")

#format plots
fig.tight_layout()
```

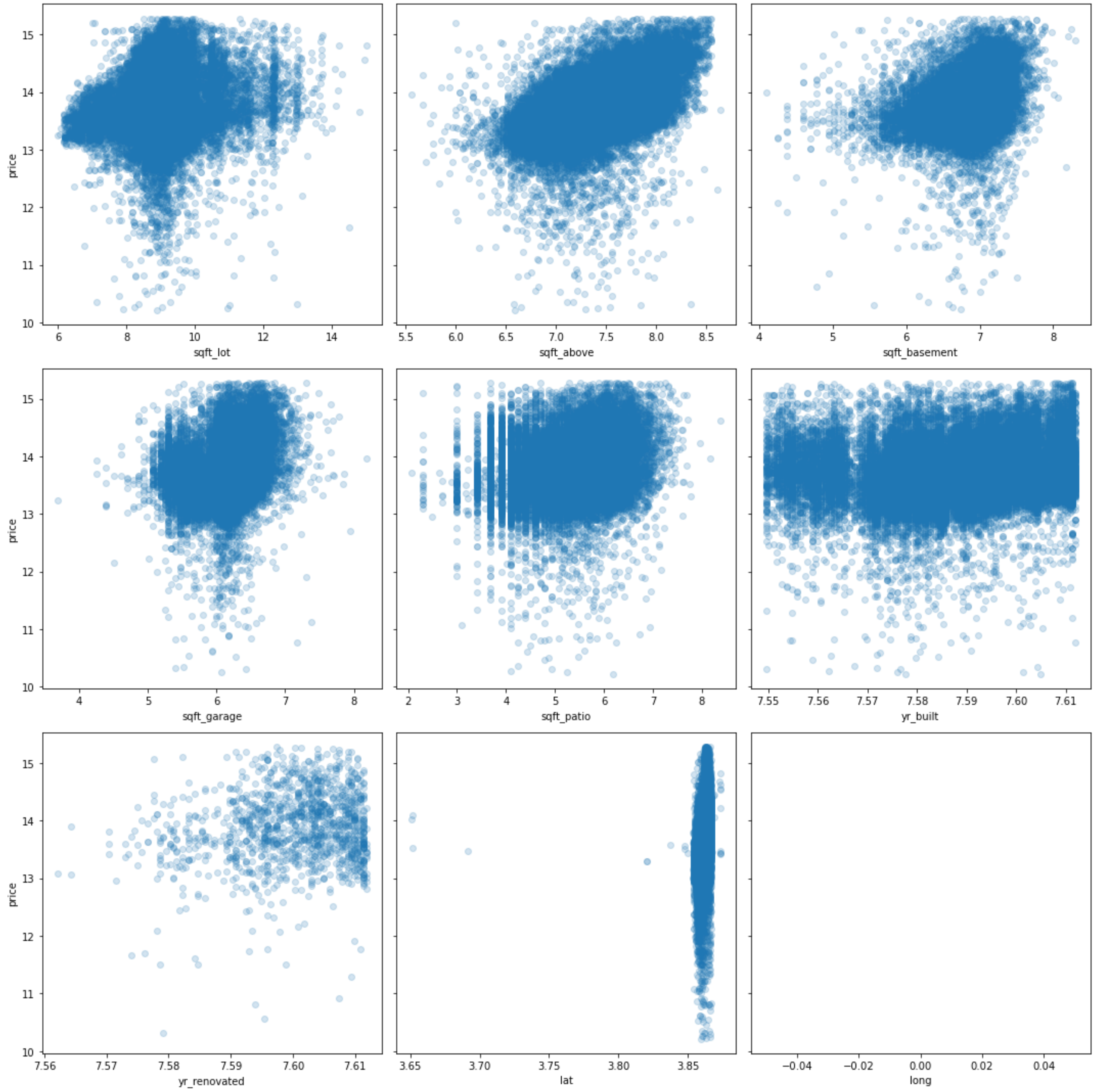


Interesting... we don't have any great linear relationships here that wouldn't already be collinear with. For example, there are a a few sqft variables that could effect the price, but we know.

Perhaps we can try log relationships. There may be some good candidates. This time, we'll try with logs for both the target and variables.

In [50]:
```
import matplotlib.pyplot as plt
import numpy as np

X = kc[numeric_cont].drop("date", axis=1)
X = X.drop("price", axis=1)
X = X.drop("sqft_living", axis=1)

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15,15), sharey=True)

for i, column in enumerate(X.columns):
    # Locate applicable axes
    row = i // 3
    col = i % 3
    ax = axes[row][col]

    # Plot feature vs. y and label axes
    ax.scatter(np.log(X[column]), np.log(y), alpha=0.2)
    ax.set_xlabel(column)
    if col == 0:
        ax.set_ylabel("price")

fig.tight_layout()
```

```
C:\Users\benne\anaconda3\envs\learn-env\lib\site-packages\pandas\core\series.py:726: Run
timeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\benne\anaconda3\envs\learn-env\lib\site-packages\pandas\core\series.py:726: Run
timeWarning: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

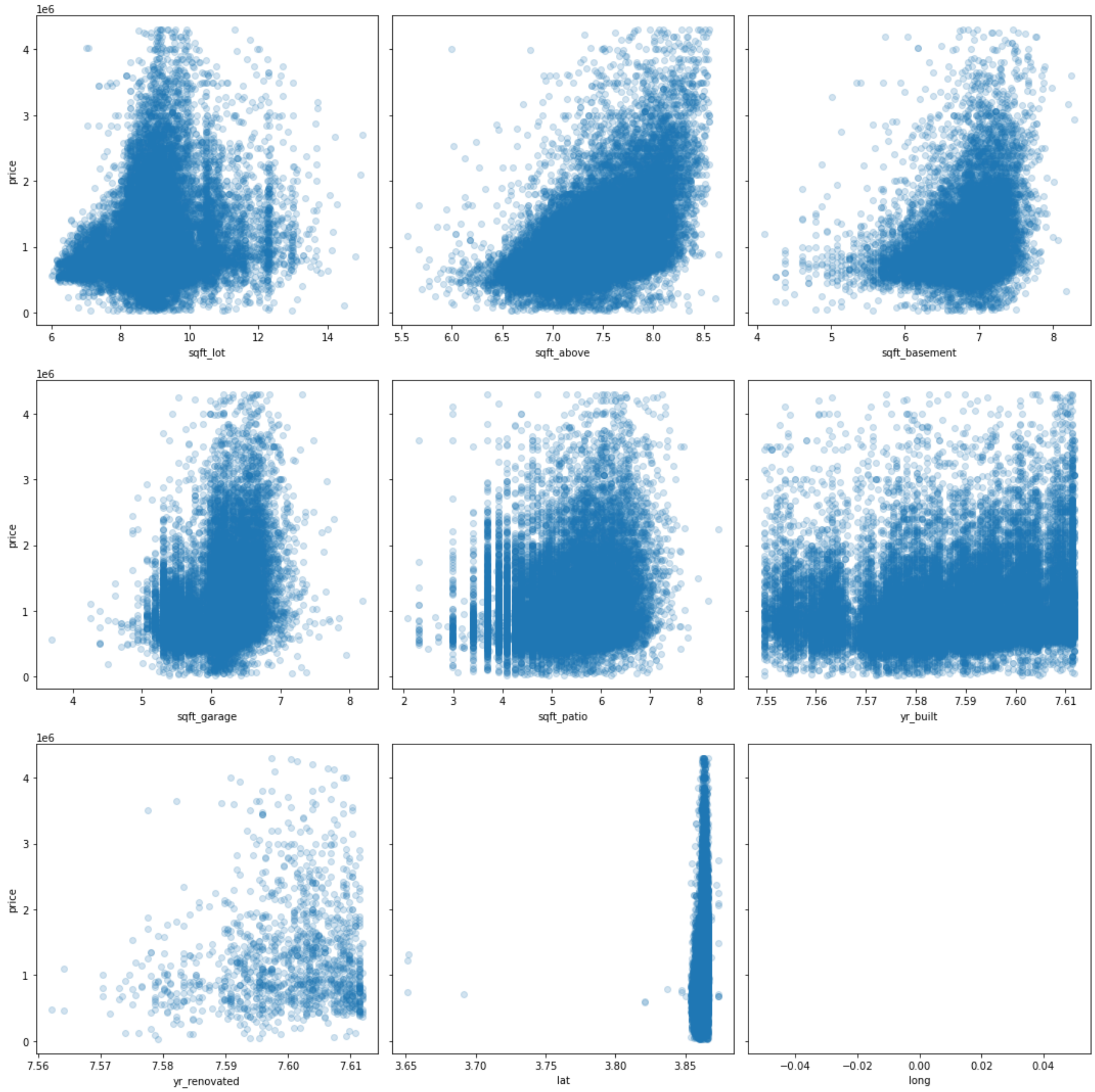


Wow! nothing is really standing out. Let's look at just the log of the variables and leave the y as is.

In [51]:

```
X = kc[numeric_cont].drop("date", axis=1)
X = X.drop("price", axis=1)
X = X.drop("sqft_living", axis=1)

fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15,15), sharey=True)

for i, column in enumerate(X.columns):
    # Locate applicable axes
    row = i // 3
    col = i % 3
    ax = axes[row][col]

    # Plot feature vs. y and label axes
    ax.scatter(np.log(X[column]), y, alpha=0.2)
    ax.set_xlabel(column)
    if col == 0:
        ax.set_ylabel("price")
```

```
fig.tight_layout()
```



I see nothing that really jumps out. Based on these graphs for the numeric columns, I feel comfortable that we've already located linear relationships without incuring major collinearity.

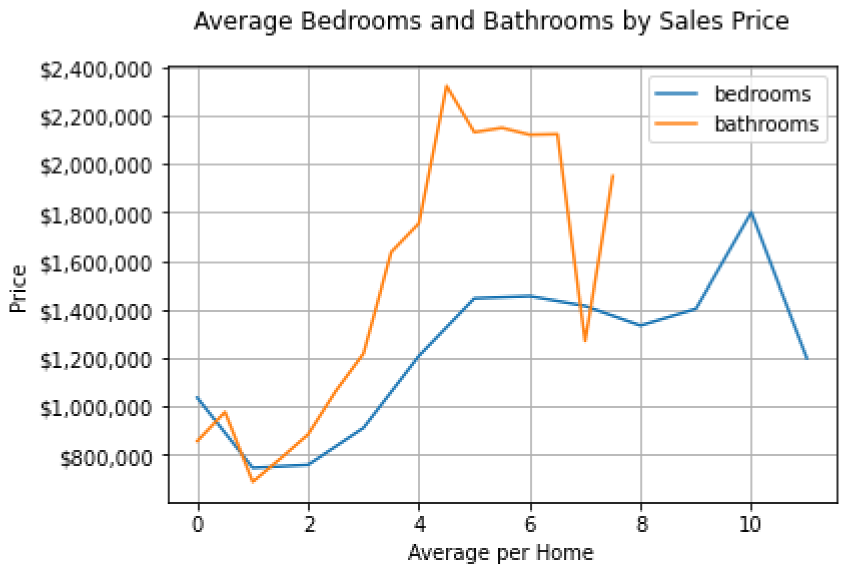
Discrete Numeric Variables.

We've got multiple discrete numeric functions which will have an effect of the saleprice. To get a high level view of this, let's look at all of the numeric variables, besides square footage.

In [52]:
```
fig, ax = plt.subplots()

ax.plot(kc.groupby(['bedrooms'])['price'].mean(), label = "bedrooms")
ax.plot(kc.groupby(['bathrooms'])['price'].mean(), label = "bathrooms")

ax.set_xlabel('Average per Home')
ax.set_ylabel('Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
```

```
fig.suptitle("Average Bedrooms and Bathrooms by Sales Price");

ax.grid('on', which='minor', axis='x' )
ax.grid('off', which='major', axis='x' )
ax.grid('on', which='minor', axis='y' )
ax.grid('off', which='major', axis='y' )

ax.legend()
```

Out[52]: <matplotlib.legend.Legend at 0x120708f2af0>



Interesting, so... a few things jump out. 1. Bathrooms is not really that linear, with peak prices occurring at 4.5 baths. 2. Bedrooms could be interpreted as linear, but really it's showing something similar to bathrooms, namely a peak or flattening around 5 bathrooms. In fact, there appears to be some errant data around 10 bathrooms, which does seem like an outlier. Let's look at the stats.

In [53]:
```
kc[numeric_disc].describe()
```

Out[53]:

	bedrooms	bathrooms	floors
count	28726.000000	28726.000000	28726.000000
mean	3.420769	2.300442	1.511523
std	0.957244	0.837446	0.547395
min	0.000000	0.000000	1.000000
25%	3.000000	2.000000	1.000000
50%	3.000000	2.500000	1.500000
75%	4.000000	3.000000	2.000000
max	11.000000	7.500000	4.000000

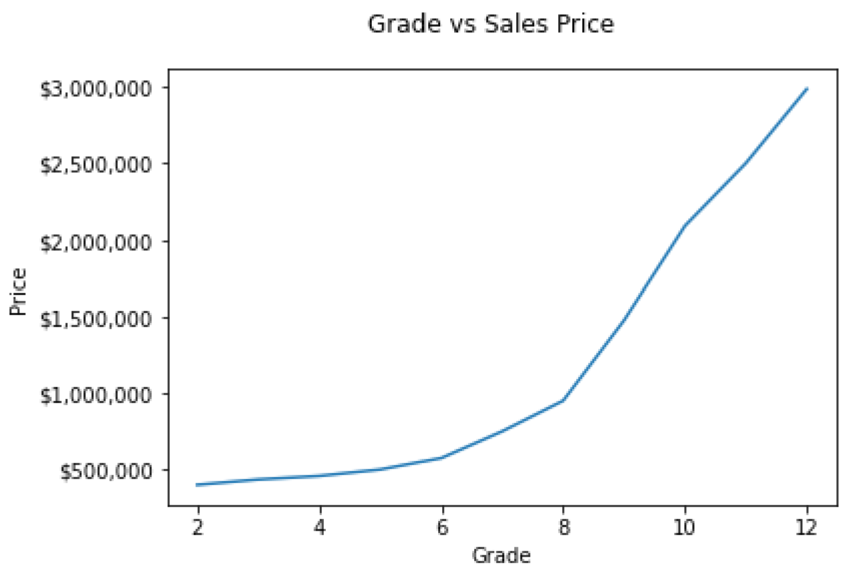
This is interesting, we the median home is a 3 bedroom, 2.5 bath. When we graphs the medians of bathroom and bedrooms, we see a peek around in price around 4 bedrooms.

Grade as a Category

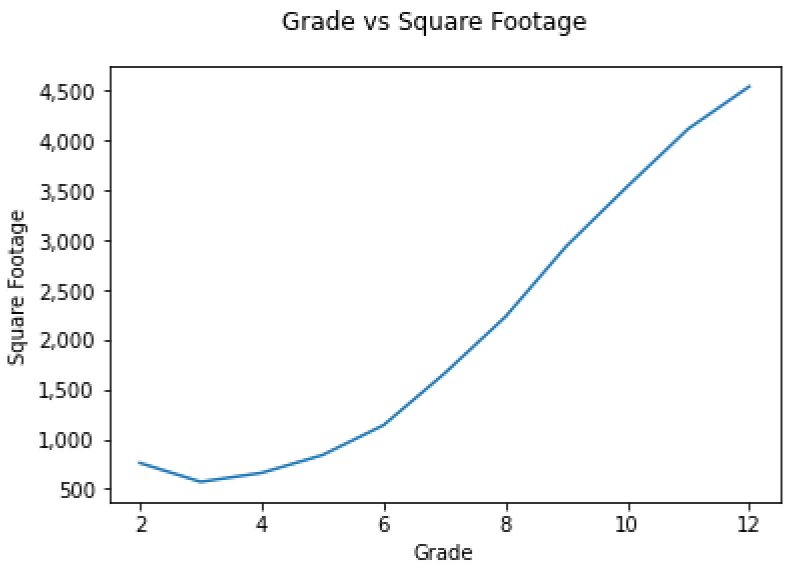
One column we haven't addressed yet, is the the grade column. Remember, this is an overall grade on the home based on the design and construction and quality of the house.

First, let's see an effect that intgrade would have on priceand sq.ft .

In [54]:
```
fig, ax = plt.subplots()

ax.plot(kc.groupby(['intgrade'])['price'].median())

ax.set_xlabel('Grade')
ax.set_ylabel('Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
fig.suptitle("Grade vs Sales Price");
```



In [55]:
```
fig, ax = plt.subplots()

ax.plot(kc.groupby(['intgrade'])['sqft_living'].median())

ax.set_xlabel('Grade')
ax.set_ylabel('Square Footage')
ax.yaxis.set_major_formatter('{x:,.0f}')
fig.suptitle("Grade vs Square Footage");
```



Yup, that look's sort of linearish for both price and sq. footage. This isn't a great sign, and we should probably avoid using this to avoid collinearity.

Add Year to the Model

In [56]:
```
#let's join year_built_transform to our exciting data set of independent variables
X_combined_year = X_combined_sales_month.join(kc['yr_built_transform'], how="inner")

#let's create the model
X_combined_year_model = sm.OLS(y, sm.add_constant(X_combined_year))
X_combined_year_results = X_combined_year_model.fit()
```

Condition as a Category

Let's try to add the condition as well. This variable represents an assessment of the overall condition of the house, with regard to maintenance. Let's add it to the model

In [57]:
```
#let's join `condition` to our data set of independent variables
X_combined_condition = X_combined_year.join(kc['condition'], how="inner")
X_combined_condition = pd.get_dummies(X_combined_condition, columns=['condition'])
```

In [58]:
```
#let's drop 'condition_Average' as it is the average rating
X_combined_condition.drop('condition_Average', axis = 1, inplace = True)

#let's create the model
X_combined_condition_model = sm.OLS(y, sm.add_constant(X_combined_condition))
X_combined_condition_results = X_combined_condition_model.fit()
```

View as a Category

Let's look at two more categories that may help us out. View and Waterfront. We always here how much a view might be worth, or how nice something is on water. So let's see.

In [59]:
```
#let's join the variable we want and then one-hot encode it.
X_combined_view = X_combined_condition.join(kc['view'], how="inner")
X_combined_view = pd.get_dummies(X_combined_view, columns=['view'])
```

```
#Let's drop zipcode 'intgrade_7' as it is the most common zipcode
X_combined_view.drop('view_NONE', axis = 1, inplace = True)

#Let's create the model
X_combined_view_model = sm.OLS(y, sm.add_constant(X_combined_view))
X_combined_view_results = X_combined_view_model.fit()
```

Waterfront as a Category

Aha! We do see that adding the View does appear to have some value. In fact, anywhere from 578k to 155k, compared to a home with no view at all. So let's check out the water.

In [60]:
```
#Let's join the variable we want and then one-hot encode it.
X_combined_water = X_combined_view.join(kc['waterfront'], how="inner")
X_combined_water = pd.get_dummies(X_combined_water, columns=['waterfront'])

#Let's drop zipcode 'intgrade_7' as it is the most common zipcode
X_combined_water.drop('waterfront_NO', axis = 1, inplace = True)

#Let's create the model
X_combined_water_model = sm.OLS(y, sm.add_constant(X_combined_water))
X_combined_water_results = X_combined_water_model.fit()
```

Greenbelt as a Category

Let's include Greenbelt as well. Greenbelt, just means undeveloped property adjacent to the property. We will one-hot encode it and drop the `greenbelt_NO`, which is the typical condition.

In [61]:
```
X_combined_greenbelt = X_combined_water.join(kc['greenbelt'], how="inner")
X_combined_greenbelt = pd.get_dummies(X_combined_greenbelt, columns=['greenbelt'])

#Let's drop zipcode 'intgrade_7' as it is the most common zipcode
X_combined_greenbelt.drop('greenbelt_NO', axis = 1, inplace = True)

X_combined_greenbelt_model = sm.OLS(y, sm.add_constant(X_combined_greenbelt))
X_combined_greenbelt_results = X_combined_greenbelt_model.fit()

print(X_combined_greenbelt_results.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                  price   R-squared:                       0.727
Model:                            OLS   Adj. R-squared:                  0.726
Method:                 Least Squares   F-statistic:                     770.0
Date:                Thu, 07 Sep 2023   Prob (F-statistic):               0.00
Time:                        12:00:05   Log-Likelihood:            -4.0504e+05
No. Observations:               28726   AIC:                         8.103e+05
Df Residuals:                   28626   BIC:                         8.111e+05
Df Model:                          99                                         
Covariance Type:            nonrobust                                         
==================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------
const           5.193e+04   2.36e+04      2.203      0.028    5734.287    9.81e+04
sqft_living      355.6009      2.611    136.209      0.000     350.484     360.718
zipcode_98001   -2.91e+05   2.53e+04    -11.498      0.000   -3.41e+05   -2.41e+05
zipcode_98002  -2.568e+05   2.76e+04     -9.292      0.000   -3.11e+05   -2.03e+05
zipcode_98003  -3.012e+05   2.67e+04    -11.285      0.000   -3.53e+05   -2.49e+05
zipcode_98004   1.422e+06   3.08e+04     46.191      0.000    1.36e+06    1.48e+06
```

zipcode_98005	8.481e+05	3.33e+04	25.435	0.000	7.83e+05	9.13e+05
zipcode_98006	5.5e+05	2.63e+04	20.908	0.000	4.98e+05	6.02e+05
zipcode_98007	4.606e+05	3.4e+04	13.563	0.000	3.94e+05	5.27e+05
zipcode_98008	4.794e+05	2.72e+04	17.634	0.000	4.26e+05	5.33e+05
zipcode_98010	-3.053e+05	2.84e+04	-10.742	0.000	-3.61e+05	-2.5e+05
zipcode_98011	1.838e+05	2.96e+04	6.212	0.000	1.26e+05	2.42e+05
zipcode_98014	-4.685e+04	3.38e+04	-1.385	0.166	-1.13e+05	1.95e+04
zipcode_98019	-3.494e+04	3.01e+04	-1.160	0.246	-9.4e+04	2.41e+04
zipcode_98022	-2.604e+05	2.65e+04	-9.821	0.000	-3.12e+05	-2.08e+05
zipcode_98023	-3.053e+05	2.5e+04	-12.208	0.000	-3.54e+05	-2.56e+05
zipcode_98024	1.268e+05	3.83e+04	3.314	0.001	5.18e+04	2.02e+05
zipcode_98027	2.242e+05	2.75e+04	8.144	0.000	1.7e+05	2.78e+05
zipcode_98028	8.474e+04	2.78e+04	3.045	0.002	3.02e+04	1.39e+05
zipcode_98029	4.066e+05	2.85e+04	14.257	0.000	3.51e+05	4.62e+05
zipcode_98030	-2.69e+05	2.75e+04	-9.791	0.000	-3.23e+05	-2.15e+05
zipcode_98031	-2.417e+05	2.59e+04	-9.315	0.000	-2.93e+05	-1.91e+05
zipcode_98032	-2.398e+05	3.22e+04	-7.437	0.000	-3.03e+05	-1.77e+05
zipcode_98033	8.741e+05	2.56e+04	34.164	0.000	8.24e+05	9.24e+05
zipcode_98034	3.005e+05	2.51e+04	11.984	0.000	2.51e+05	3.5e+05
zipcode_98038	-1.804e+05	2.44e+04	-7.379	0.000	-2.28e+05	-1.32e+05
zipcode_98039	2.044e+06	6.28e+04	32.562	0.000	1.92e+06	2.17e+06
zipcode_98040	9.805e+05	2.87e+04	34.201	0.000	9.24e+05	1.04e+06
zipcode_98042	-2.826e+05	2.41e+04	-11.709	0.000	-3.3e+05	-2.35e+05
zipcode_98045	-1.38e+04	2.64e+04	-0.522	0.602	-6.56e+04	3.8e+04
zipcode_98047	-2.373e+05	4.29e+04	-5.527	0.000	-3.21e+05	-1.53e+05
zipcode_98050	2.641e+05	2.29e+05	1.153	0.249	-1.85e+05	7.13e+05
zipcode_98051	-1.468e+05	4.55e+04	-3.226	0.001	-2.36e+05	-5.76e+04
zipcode_98052	5.331e+05	2.57e+04	20.704	0.000	4.83e+05	5.84e+05
zipcode_98053	3.973e+05	2.75e+04	14.447	0.000	3.43e+05	4.51e+05
zipcode_98055	-2.012e+05	3.09e+04	-6.519	0.000	-2.62e+05	-1.41e+05
zipcode_98056	-759.1345	2.6e+04	-0.029	0.977	-5.17e+04	5.02e+04
zipcode_98057	-1.76e+05	3.61e+04	-4.875	0.000	-2.47e+05	-1.05e+05
zipcode_98058	-1.836e+05	2.5e+04	-7.334	0.000	-2.33e+05	-1.35e+05
zipcode_98059	-3824.5670	2.57e+04	-0.149	0.882	-5.42e+04	4.65e+04
zipcode_98065	6.902e+04	2.89e+04	2.386	0.017	1.23e+04	1.26e+05
zipcode_98072	2.841e+05	2.82e+04	10.070	0.000	2.29e+05	3.39e+05
zipcode_98074	5.134e+05	2.7e+04	19.003	0.000	4.6e+05	5.66e+05
zipcode_98075	5.461e+05	2.72e+04	20.054	0.000	4.93e+05	5.99e+05
zipcode_98077	3.663e+05	3.05e+04	12.022	0.000	3.07e+05	4.26e+05
zipcode_98092	-3.169e+05	2.54e+04	-12.454	0.000	-3.67e+05	-2.67e+05
zipcode_98102	5.306e+05	3.48e+04	15.230	0.000	4.62e+05	5.99e+05
zipcode_98103	3.14e+05	2.48e+04	12.672	0.000	2.65e+05	3.63e+05
zipcode_98105	4.263e+05	2.84e+04	14.997	0.000	3.71e+05	4.82e+05
zipcode_98106	-4.948e+04	2.58e+04	-1.916	0.055	-1e+05	1132.198
zipcode_98107	2.858e+05	2.65e+04	10.780	0.000	2.34e+05	3.38e+05
zipcode_98108	-3.029e+04	2.86e+04	-1.059	0.290	-8.63e+04	2.58e+04
zipcode_98109	5.065e+05	3.61e+04	14.029	0.000	4.36e+05	5.77e+05
zipcode_98112	6.423e+05	2.91e+04	22.086	0.000	5.85e+05	6.99e+05
zipcode_98115	3.273e+05	2.48e+04	13.186	0.000	2.79e+05	3.76e+05
zipcode_98116	2.201e+05	2.73e+04	8.068	0.000	1.67e+05	2.74e+05
zipcode_98117	2.891e+05	2.48e+04	11.639	0.000	2.4e+05	3.38e+05
zipcode_98118	3.434e+04	2.55e+04	1.349	0.177	-1.56e+04	8.42e+04
zipcode_98119	4.802e+05	3.03e+04	15.842	0.000	4.21e+05	5.4e+05
zipcode_98122	3.253e+05	2.67e+04	12.181	0.000	2.73e+05	3.78e+05
zipcode_98125	1.107e+05	2.62e+04	4.226	0.000	5.93e+04	1.62e+05
zipcode_98126	4.238e+04	2.67e+04	1.589	0.112	-9894.333	9.47e+04
zipcode_98133	3.134e+04	2.51e+04	1.247	0.212	-1.79e+04	8.06e+04
zipcode_98136	1.622e+05	2.87e+04	5.648	0.000	1.06e+05	2.18e+05
zipcode_98144	1.97e+05	2.67e+04	7.372	0.000	1.45e+05	2.49e+05
zipcode_98146	-5.707e+04	2.68e+04	-2.127	0.033	-1.1e+05	-4491.081
zipcode_98148	-1.827e+05	3.99e+04	-4.576	0.000	-2.61e+05	-1.04e+05
zipcode_98155	8.026e+04	2.6e+04	3.082	0.002	2.92e+04	1.31e+05
zipcode_98166	-1.021e+05	2.78e+04	-3.666	0.000	-1.57e+05	-4.75e+04
zipcode_98168	-1.712e+05	2.74e+04	-6.257	0.000	-2.25e+05	-1.18e+05
zipcode_98177	1.979e+05	2.88e+04	6.871	0.000	1.41e+05	2.54e+05

```
zipcode_98178        -1.64e+05   2.73e+04     -6.007      0.000   -2.17e+05    -1.1e+05
zipcode_98188       -2.112e+05   3.15e+04     -6.710      0.000   -2.73e+05   -1.49e+05
zipcode_98198       -2.217e+05   2.68e+04     -8.278      0.000   -2.74e+05   -1.69e+05
zipcode_98199        4.385e+05   2.71e+04     16.211      0.000    3.86e+05    4.92e+05
zipcode_98224       -1.786e+05   1.88e+05     -0.953      0.341   -5.46e+05    1.89e+05
zipcode_98288       -2.647e+05   8.34e+04     -3.173      0.002   -4.28e+05   -1.01e+05
zipcode_98354       -2.481e+05   7.07e+04     -3.511      0.000   -3.87e+05    -1.1e+05
sales_month_1        5.479e+04   1.13e+04      4.838      0.000    3.26e+04     7.7e+04
sales_month_2        1.403e+05   1.01e+04     13.915      0.000    1.21e+05     1.6e+05
sales_month_3        1.958e+05   8703.118     22.493      0.000    1.79e+05    2.13e+05
sales_month_4            2e+05   8656.906     23.103      0.000    1.83e+05    2.17e+05
sales_month_5        1.862e+05   8715.990     21.362      0.000    1.69e+05    2.03e+05
sales_month_6        4418.7184   8534.787      0.518      0.605   -1.23e+04    2.11e+04
sales_month_8       -6059.9879   8178.909     -0.741      0.459   -2.21e+04    9971.057
sales_month_9          50.5415   8414.276      0.006      0.995   -1.64e+04    1.65e+04
sales_month_10        1.62e+04   8496.073      1.907      0.057    -453.138    3.29e+04
sales_month_11       2.734e+04   8754.131      3.123      0.002    1.02e+04    4.45e+04
sales_month_12       3.773e+04   9483.976      3.978      0.000    1.91e+04    5.63e+04
yr_built_transform    401.3477     77.604      5.172      0.000     249.240     553.456
condition_Fair       -7.91e+04   2.19e+04     -3.609      0.000   -1.22e+05   -3.61e+04
condition_Good        3.38e+04   4730.032      7.145      0.000    2.45e+04    4.31e+04
condition_Poor      -6.677e+04   4.19e+04     -1.593      0.111   -1.49e+05    1.54e+04
condition_Very Good  8.712e+04   6563.881     13.272      0.000    7.43e+04       1e+05
view_AVERAGE         1.046e+05   8147.422     12.836      0.000    8.86e+04    1.21e+05
view_EXCELLENT       5.552e+05   1.81e+04     30.613      0.000     5.2e+05    5.91e+05
view_FAIR            1.793e+05    2.3e+04      7.809      0.000    1.34e+05    2.24e+05
view_GOOD            2.343e+05    1.2e+04     19.584      0.000    2.11e+05    2.58e+05
waterfront_YES       2.349e+05   1.88e+04     12.500      0.000    1.98e+05    2.72e+05
greenbelt_YES        6.607e+04   1.24e+04      5.307      0.000    4.17e+04    9.05e+04
==============================================================================
Omnibus:                     8052.632   Durbin-Watson:                   1.945
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            99987.925
Skew:                           0.996   Prob(JB):                         0.00
Kurtosis:                      11.920   Cond. No.                     2.74e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 2.74e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Final Model Results

Okay, this feels like a reasonable model. Let's review some of the statistics.

Our Adjusted R-Square is now showing a value of 72.7% of all variance accounted for. This is better, but one wonders if 5% increase when accounting for all of these models is really worth it. But... anyways.

Using the standard alpha of 0.05 to evaluate statistical significance:

Coefficients for sqft_living and most of our zipcodes are statistically significant. Our baseline zipcode is 98070. It seems that relative to our baseline, the zipcodes do have a statistically significant effect on price, except for zipcodes 98014, 98019, 98045, 98050, 98056, 98059, 98108, 98118, 98126, 98133, 982242). So...

Our coefficient for the intercept is not significanty significant for an alpha of .05.

According to the model, houses are selling at approximately $355/sq. ft.

The coefficients for the intercept is 51,930. That means that, when not accounting for square footage or zipcode, you could assume a house will sell for 51,930.

The zipcode with the largest effect is zipcode 98004, 98005, 98039, and 98040. Zipcode 98010, 98001, 98003, 98023, have the most negative effect on pricing.

Now that we have this model, let's double check for errors.

10. Checking for Errors and Accuracy of the Model

To do this, I'm going to check the assumptions of Linear Regression. First, let's make sure there's no collinearity.

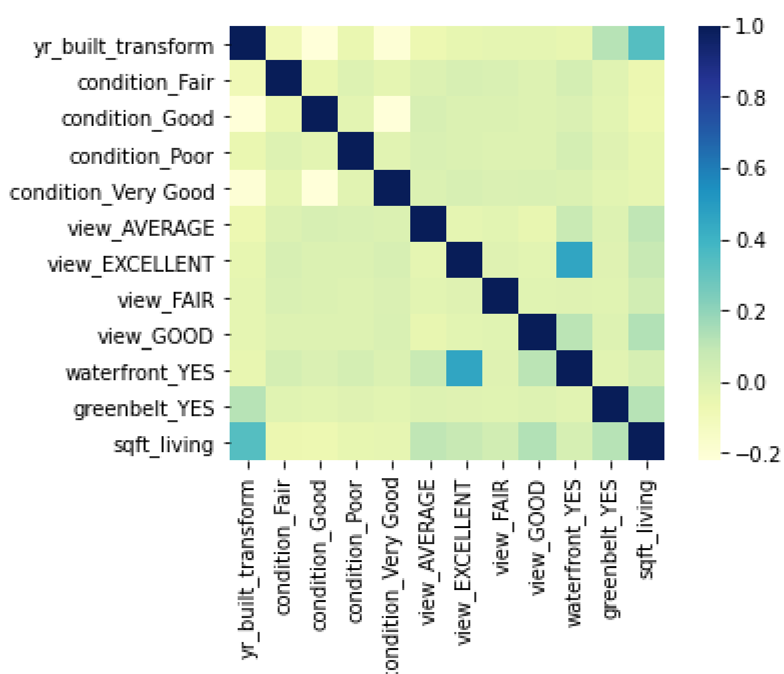
Check for Collinearity

To do this, we're going to create a correlation matrix, and view it on a heat map. Basically, we're looking for anything that is very dark blue or higher

In [62]:

```
import seaborn as sns

corr_check = X_combined_greenbelt.loc[:,'yr_built_transform':'greenbelt_YES']
corr_check = corr_check.join(kc['sqft_living'], how="inner")


sns.heatmap(corr_check.corr(), cmap='YlGnBu', square=True)
```

Out[62]: <AxesSubplot:>



It looks as though there isn't much correlation, which is good. There some deep bluish-greenish between `view_EXCELLENT` and `waterfront_YES` , which makes sense. I'll remove `view_EXCELLENT` .

Also, there seems to be some issues with the `yr_built_transform` variable. I see some correlation between `yr_built_transform` and `intgrades` . I'll remove `yr_built_transform` as it seems to have a small effect at $401/yr.

So I'm going to go ahead and drop those items, and let's see where we land.

In [63]:
```
X_combined_greenbelt.drop('waterfront_YES', axis = 1, inplace = True)
X_combined_greenbelt.drop('yr_built_transform', axis = 1, inplace = True)

X_combined_greenbelt_model = sm.OLS(y, sm.add_constant(X_combined_greenbelt))
X_combined_greenbelt_results = X_combined_greenbelt_model.fit()
```

Let's continue to check a few of these things. To make sure we have a model that's nice an linear, let's go ahead and check a few more assumptions of linearity.

First, let's do some error testing.

Error Testing

MAE

I'm going to investigate the residuals and see how it looks. First, let's check the Mean Asbolute Error (MAE)

In [64]:
```
y_pred = X_combined_greenbelt_results.fittedvalues
y = kc["price"]

mae_resid = np.mean(np.abs(y - y_pred))
mae_resid
```

Out[64]:
```
212706.98225099346
```

The model informs us that we have a mean average error of about $215,000 (200,259).

RMSE

Let's see what the Root Mean Squared Error (RMSE) is:

In [65]:
```
rmse_residuals = np.sqrt(X_combined_greenbelt_results.mse_resid)
rmse_residuals
```

Out[65]:
```
323211.54707118793
```

So here, the RMSE is about $323,000. Because the RMSE > MAE, there may be more outliers in our data even though we eliminated the top 1% for square footage and price.

Linearity

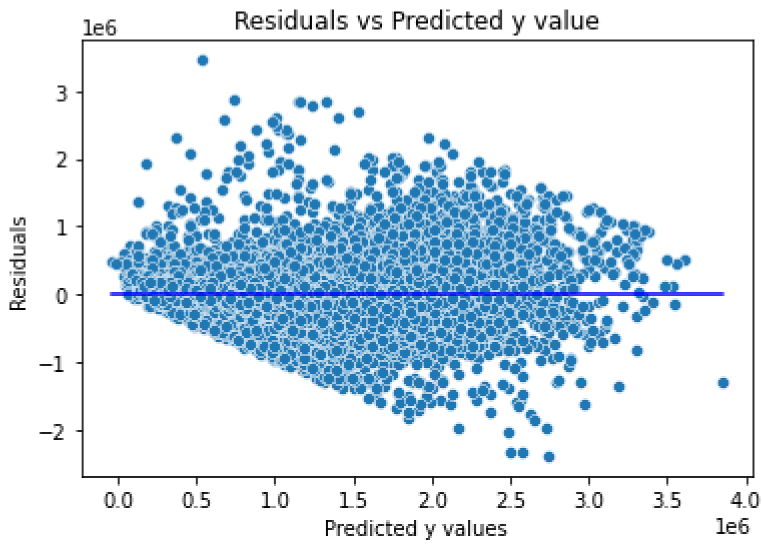
We covered this in previous sections. I feel comfortable that we've included most of the variables with some linear properties while avoiding collinearity.

Homoscedasticity

Homoscedasticity is the observation that the magnitude of the errors (or residuals) is the same no matter what the input, or independent variable is. The most effective way to observe this is by plotting the residuals against the predicted values. Homoscedasticity will result in a straight line around the max residuals. Heteroscedasticity will result in a curved line around the max residuals

In [66]:
```
residuals = X_combined_greenbelt_results.resid

p = sns.scatterplot(y_pred,residuals)
plt.xlabel('Predicted y values')
plt.ylabel('Residuals')
#plt.xlim(70,100)
p = sns.lineplot([y_pred.min(),y_pred.max()],[0,0],color='blue')
p = plt.title('Residuals vs Predicted y value')
```

```
C:\Users\benne\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\benne\anaconda3\envs\learn-env\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an exp
licit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Okay, this... doesn't look flat, but we do see consistency in the errors of the model. It appears we are getting a blob shape here, meaning we got consistent errors as our predicted value get higher. Let's do another test. The dreaded, Goldfeld Quandt Test.

Goldfeld Quandt Test

In [68]:
```
# run Goldfeld Quandt Test
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
name = ['F statistic', 'p-value']

test = sms.het_goldfeldquandt(residuals, sm.add_constant(X_combined_greenbelt))
lzip(name, test)
```

Out[68]:
```
[('F statistic', 1.0237944110966875), ('p-value', 0.08011863682380574)]
```

Here our p-value is 0.08 greater than 0.05. This means we are not able to reject the null hypothesis which states our error terms are homoscedastic.

This further validates the assumption of homoscedasticity of our residuals. Even though it ain't the prettiest thing.

So... we're comfortable with the model, let's discuss the results.

11. Discussion of Results

If you recally, we discussed previously how, in general, Kings County had on a price of 447/sq.ft County wide.

After we accounted for zipcode, we saw quite a shift depending on where our home was located.

But what about the other features that we added. Let's run our model one more time to determine all of our coefficients.

In [69]:
```
X_combined_greenbelt_model = sm.OLS(y, sm.add_constant(X_combined_greenbelt))
X_combined_greenbelt_results = X_combined_greenbelt_model.fit()

print(X_combined_greenbelt_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.725
Model:                            OLS   Adj. R-squared:                  0.724
Method:                 Least Squares   F-statistic:                     779.3
Date:                Thu, 07 Sep 2023   Prob (F-statistic):               0.00
Time:                        12:00:47   Log-Likelihood:            -4.0513e+05
No. Observations:               28726   AIC:                         8.105e+05
Df Residuals:                   28628   BIC:                         8.113e+05
Df Model:                          97
Covariance Type:            nonrobust
=======================================================================================
                          coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------------
const                 1.12e+05   2.29e+04      4.892      0.000    6.71e+04    1.57e+05
sqft_living           357.4552      2.521    141.775      0.000     352.513     362.397
zipcode_98001       -3.138e+05   2.53e+04    -12.403      0.000   -3.63e+05   -2.64e+05
zipcode_98002       -2.848e+05   2.76e+04    -10.304      0.000   -3.39e+05   -2.31e+05
zipcode_98003       -3.278e+05   2.67e+04    -12.292      0.000    -3.8e+05   -2.76e+05
zipcode_98004        1.389e+06   3.08e+04     45.132      0.000    1.33e+06    1.45e+06
zipcode_98005        8.126e+05   3.33e+04     24.378      0.000    7.47e+05    8.78e+05
zipcode_98006        5.164e+05   2.62e+04     19.677      0.000    4.65e+05    5.68e+05
zipcode_98007        4.294e+05    3.4e+04     12.637      0.000    3.63e+05    4.96e+05
zipcode_98008        4.496e+05   2.72e+04     16.546      0.000    3.96e+05    5.03e+05
zipcode_98010       -3.153e+05   2.84e+04    -11.092      0.000   -3.71e+05    -2.6e+05
zipcode_98011        1.583e+05   2.96e+04      5.348      0.000       1e+05    2.16e+05
zipcode_98014       -5.978e+04   3.39e+04     -1.763      0.078   -1.26e+05    6681.056
zipcode_98019       -5.541e+04   3.01e+04     -1.841      0.066   -1.14e+05    3595.502
zipcode_98022       -2.901e+05   2.65e+04    -10.960      0.000   -3.42e+05   -2.38e+05
zipcode_98023       -3.275e+05    2.5e+04    -13.100      0.000   -3.76e+05   -2.78e+05
zipcode_98024        1.156e+05   3.84e+04      3.014      0.003    4.04e+04    1.91e+05
zipcode_98027        1.984e+05   2.75e+04      7.209      0.000    1.44e+05    2.52e+05
zipcode_98028        5.707e+04   2.78e+04      2.052      0.040    2545.940    1.12e+05
zipcode_98029        3.822e+05   2.85e+04     13.420      0.000    3.26e+05    4.38e+05
zipcode_98030       -2.938e+05   2.74e+04    -10.707      0.000   -3.48e+05    -2.4e+05
zipcode_98031       -2.665e+05   2.59e+04    -10.286      0.000   -3.17e+05   -2.16e+05
```

zipcode_98032	-2.626e+05	3.23e+04	-8.129	0.000	-3.26e+05	-1.99e+05
zipcode_98033	8.468e+05	2.56e+04	33.134	0.000	7.97e+05	8.97e+05
zipcode_98034	2.722e+05	2.5e+04	10.872	0.000	2.23e+05	3.21e+05
zipcode_98038	-1.963e+05	2.44e+04	-8.041	0.000	-2.44e+05	-1.48e+05
zipcode_98039	2.007e+06	6.29e+04	31.898	0.000	1.88e+06	2.13e+06
zipcode_98040	9.519e+05	2.87e+04	33.199	0.000	8.96e+05	1.01e+06
zipcode_98042	-3.048e+05	2.41e+04	-12.659	0.000	-3.52e+05	-2.58e+05
zipcode_98045	-2.75e+04	2.64e+04	-1.040	0.298	-7.93e+04	2.43e+04
zipcode_98047	-2.634e+05	4.3e+04	-6.127	0.000	-3.48e+05	-1.79e+05
zipcode_98050	2.19e+05	2.3e+05	0.953	0.341	-2.31e+05	6.69e+05
zipcode_98051	-1.74e+05	4.56e+04	-3.817	0.000	-2.63e+05	-8.46e+04
zipcode_98052	5.066e+05	2.57e+04	19.700	0.000	4.56e+05	5.57e+05
zipcode_98053	3.753e+05	2.75e+04	13.649	0.000	3.21e+05	4.29e+05
zipcode_98055	-2.292e+05	3.09e+04	-7.425	0.000	-2.9e+05	-1.69e+05
zipcode_98056	-2.71e+04	2.6e+04	-1.043	0.297	-7.8e+04	2.38e+04
zipcode_98057	-2.211e+05	3.6e+04	-6.136	0.000	-2.92e+05	-1.5e+05
zipcode_98058	-2.034e+05	2.51e+04	-8.118	0.000	-2.52e+05	-1.54e+05
zipcode_98059	-2.767e+04	2.57e+04	-1.078	0.281	-7.8e+04	2.26e+04
zipcode_98065	4.419e+04	2.89e+04	1.528	0.127	-1.25e+04	1.01e+05
zipcode_98072	2.608e+05	2.82e+04	9.242	0.000	2.05e+05	3.16e+05
zipcode_98074	4.893e+05	2.7e+04	18.124	0.000	4.36e+05	5.42e+05
zipcode_98075	5.227e+05	2.72e+04	19.200	0.000	4.69e+05	5.76e+05
zipcode_98077	3.406e+05	3.05e+04	11.175	0.000	2.81e+05	4e+05
zipcode_98092	-3.404e+05	2.54e+04	-13.404	0.000	-3.9e+05	-2.91e+05
zipcode_98102	4.92e+05	3.48e+04	14.129	0.000	4.24e+05	5.6e+05
zipcode_98103	2.766e+05	2.47e+04	11.202	0.000	2.28e+05	3.25e+05
zipcode_98105	3.827e+05	2.83e+04	13.529	0.000	3.27e+05	4.38e+05
zipcode_98106	-7.863e+04	2.58e+04	-3.051	0.002	-1.29e+05	-2.81e+04
zipcode_98107	2.52e+05	2.65e+04	9.520	0.000	2e+05	3.04e+05
zipcode_98108	-6.418e+04	2.86e+04	-2.247	0.025	-1.2e+05	-8188.556
zipcode_98109	4.571e+05	3.6e+04	12.694	0.000	3.87e+05	5.28e+05
zipcode_98112	5.99e+05	2.9e+04	20.689	0.000	5.42e+05	6.56e+05
zipcode_98115	2.875e+05	2.47e+04	11.638	0.000	2.39e+05	3.36e+05
zipcode_98116	1.74e+05	2.71e+04	6.411	0.000	1.21e+05	2.27e+05
zipcode_98117	2.512e+05	2.47e+04	10.150	0.000	2.03e+05	3e+05
zipcode_98118	-2768.7300	2.54e+04	-0.109	0.913	-5.25e+04	4.7e+04
zipcode_98119	4.328e+05	3.02e+04	14.340	0.000	3.74e+05	4.92e+05
zipcode_98122	2.902e+05	2.67e+04	10.886	0.000	2.38e+05	3.42e+05
zipcode_98125	7.705e+04	2.61e+04	2.947	0.003	2.58e+04	1.28e+05
zipcode_98126	5276.3497	2.66e+04	0.198	0.843	-4.69e+04	5.74e+04
zipcode_98133	-989.9415	2.51e+04	-0.039	0.969	-5.02e+04	4.82e+04
zipcode_98136	1.2e+05	2.86e+04	4.191	0.000	6.39e+04	1.76e+05
zipcode_98144	1.6e+05	2.67e+04	6.001	0.000	1.08e+05	2.12e+05
zipcode_98146	-9.386e+04	2.68e+04	-3.507	0.000	-1.46e+05	-4.14e+04
zipcode_98148	-2.146e+05	4e+04	-5.368	0.000	-2.93e+05	-1.36e+05
zipcode_98155	4.571e+04	2.6e+04	1.759	0.079	-5234.486	9.67e+04
zipcode_98166	-1.353e+05	2.78e+04	-4.866	0.000	-1.9e+05	-8.08e+04
zipcode_98168	-2.085e+05	2.73e+04	-7.636	0.000	-2.62e+05	-1.55e+05
zipcode_98177	1.556e+05	2.87e+04	5.420	0.000	9.94e+04	2.12e+05
zipcode_98178	-2.01e+05	2.72e+04	-7.378	0.000	-2.54e+05	-1.48e+05
zipcode_98188	-2.401e+05	3.15e+04	-7.623	0.000	-3.02e+05	-1.78e+05
zipcode_98198	-2.527e+05	2.68e+04	-9.445	0.000	-3.05e+05	-2e+05
zipcode_98199	4.032e+05	2.7e+04	14.932	0.000	3.5e+05	4.56e+05
zipcode_98224	-2.299e+05	1.88e+05	-1.223	0.221	-5.99e+05	1.39e+05
zipcode_98288	-2.467e+05	8.37e+04	-2.948	0.003	-4.11e+05	-8.27e+04
zipcode_98354	-2.716e+05	7.08e+04	-3.834	0.000	-4.1e+05	-1.33e+05
sales_month_1	5.187e+04	1.14e+04	4.567	0.000	2.96e+04	7.41e+04
sales_month_2	1.375e+05	1.01e+04	13.606	0.000	1.18e+05	1.57e+05
sales_month_3	1.946e+05	8729.138	22.295	0.000	1.78e+05	2.12e+05
sales_month_4	1.984e+05	8681.770	22.857	0.000	1.81e+05	2.15e+05
sales_month_5	1.844e+05	8740.389	21.093	0.000	1.67e+05	2.01e+05
sales_month_6	3819.1434	8560.850	0.446	0.656	-1.3e+04	2.06e+04
sales_month_8	-6051.7972	8203.958	-0.738	0.461	-2.21e+04	1e+04
sales_month_9	-426.9064	8440.014	-0.051	0.960	-1.7e+04	1.61e+04
sales_month_10	1.455e+04	8520.724	1.708	0.088	-2151.475	3.13e+04

```
sales_month_11       2.536e+04   8779.666      2.889      0.004    8156.415    4.26e+04
sales_month_12       3.582e+04   9510.345      3.767      0.000    1.72e+04    5.45e+04
condition_Fair      -8.469e+04   2.19e+04     -3.874      0.000   -1.28e+05   -4.18e+04
condition_Good       2.638e+04   4507.574      5.851      0.000    1.75e+04    3.52e+04
condition_Poor      -6.624e+04   4.19e+04     -1.580      0.114   -1.48e+05    1.59e+04
condition_Very Good  7.721e+04   6309.289     12.238      0.000    6.48e+04    8.96e+04
view_AVERAGE         1.144e+05   8086.098     14.148      0.000    9.86e+04     1.3e+05
view_EXCELLENT       6.559e+05   1.61e+04     40.771      0.000    6.24e+05    6.87e+05
view_FAIR            1.787e+05    2.3e+04      7.761      0.000    1.34e+05    2.24e+05
view_GOOD             2.53e+05   1.18e+04     21.349      0.000     2.3e+05    2.76e+05
greenbelt_YES        6.576e+04   1.25e+04      5.269      0.000    4.13e+04    9.02e+04
==============================================================================
Omnibus:                     8176.956   Durbin-Watson:                   1.948
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            99281.915
Skew:                           1.023   Prob(JB):                         0.00
Kurtosis:                      11.875   Cond. No.                     2.74e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specifi
ed.
[2] The condition number is large, 2.74e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Okay, this feels like a reasonable model. Let's review some of the statistics.

We also discovered that our model has an error of approximately $212,000

Our Adjusted R-Square is now showing a value of 72.4% of all variance accounted for. This is better, but one wonders if 4% increase when accounting for all of these models is really worth it. But... anyways.

Using the standard alpha of 0.05 to evaluate statistical significance:

Coefficients for sqft_living and most of our zipcodes are statistically significant. Our baseline zipcode is 98070. It seems that relative to our baseline, the zipcodes do have a statistically significant effect on price, except for zipcodes 98014, 98019, 98045, 98050, 98056, 98059, 98065, 98118, 98126, 98133, 98155, 98224). So...

Our coefficient for the intercept is not significanty significant for an alpha of .05.

According to the model, houses are selling at approximately $357/sq. ft.

The coefficients for the intercept is 112,000. That means that, when not accounting for square footage or zipcode, you could assume a house will sell for 112,000.

The zipcodes with the largest effect are 98004, 98005, 98033, 98039, and 98040. Zipcode 98010, 98001, 98003, 98023, and 98092 have the most negative effect on pricing.

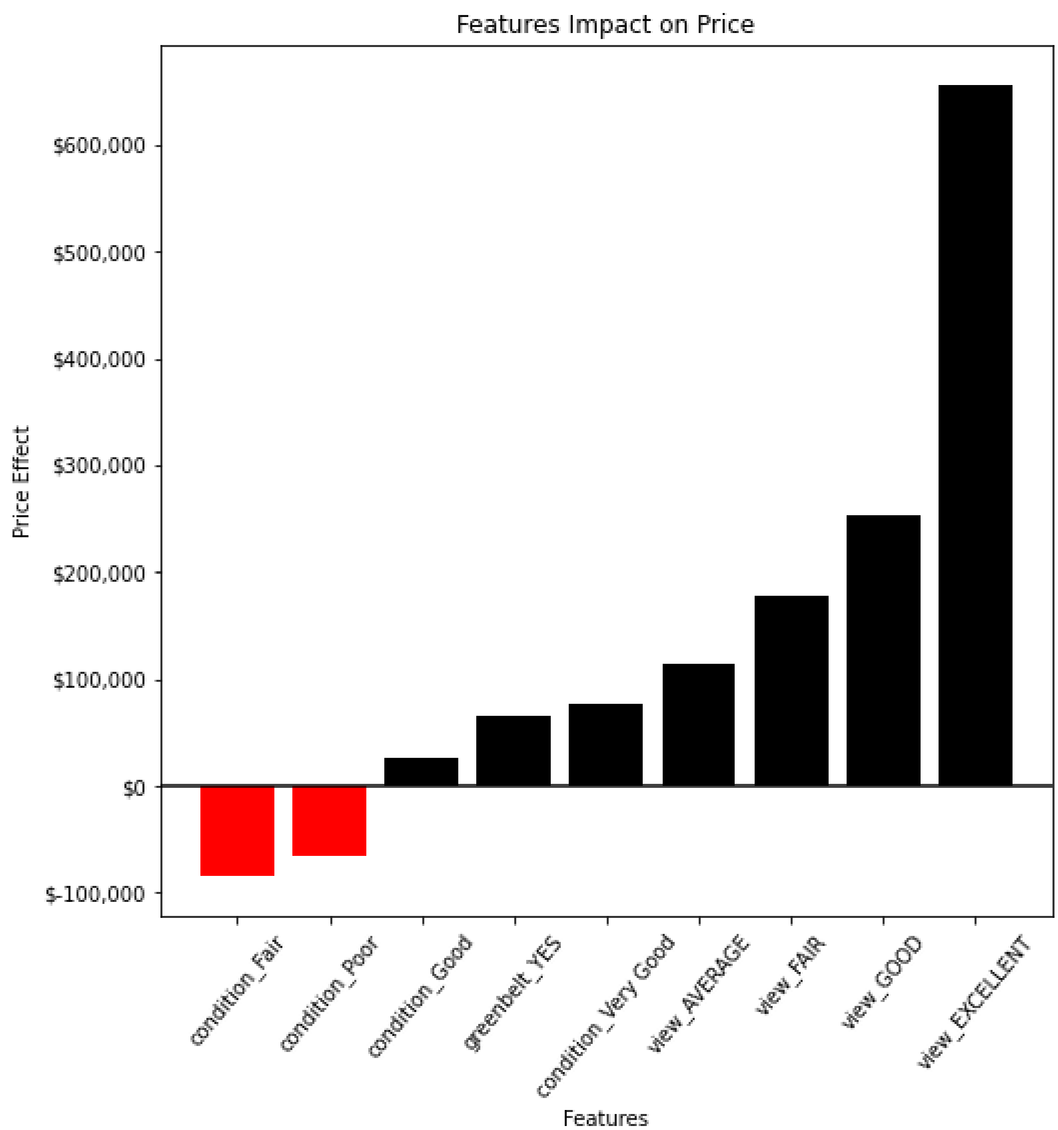
Quantifying additional features besides square footage and zipcode

In [70]:
```
features = X_combined_greenbelt_results.params

features = features.drop(features.index[0:89])
```

```
features.sort_values(ascending = True, inplace = True)

features_red = features.drop(features.index[2:10])
features_black = features.drop(features.index[0:2])
```

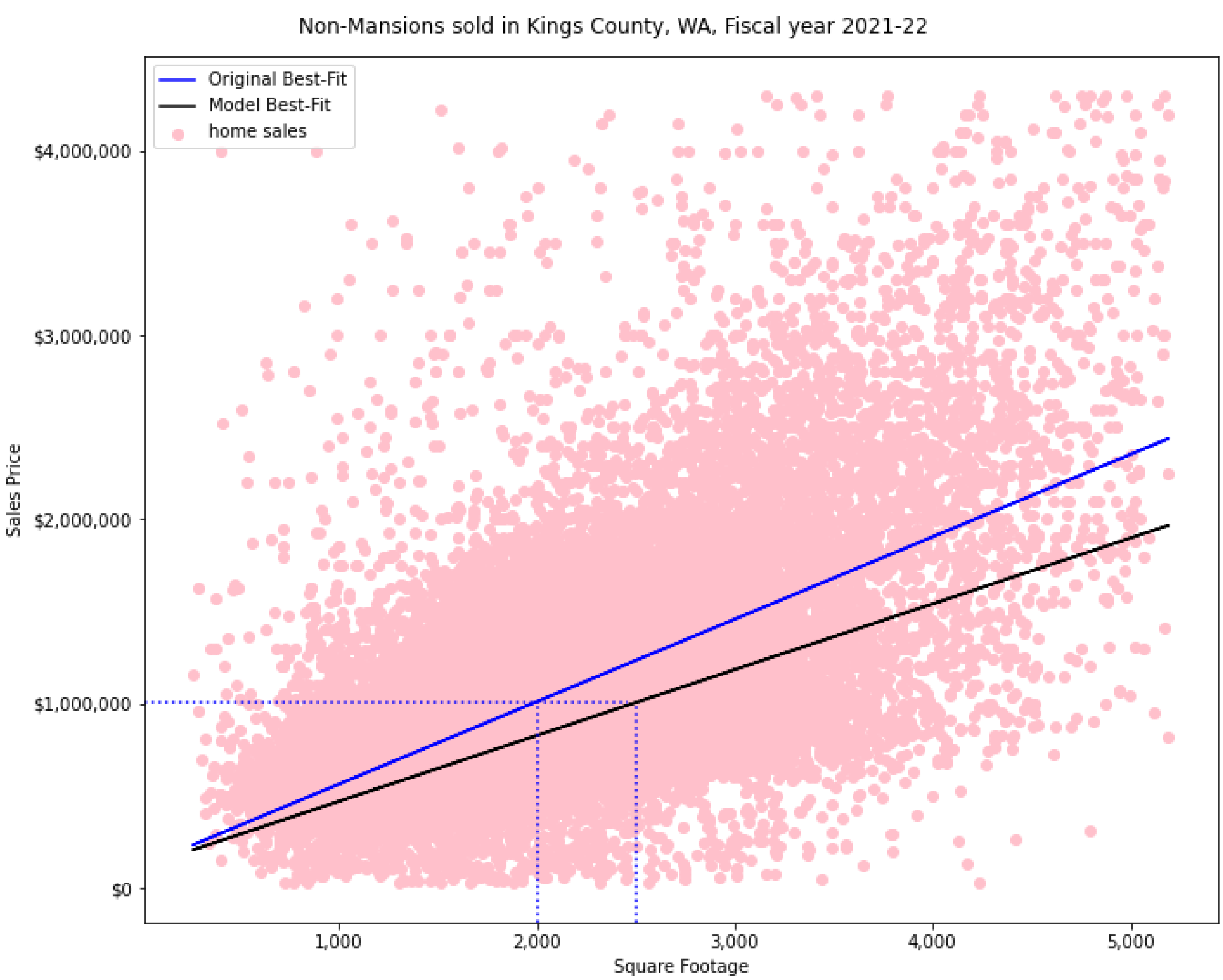
In [71]:
```
fig, ax = plt.subplots(figsize = (8,8))

ax.bar(x=features_red.index, height=features_red.values, color='red')
ax.bar(x=features_black.index, height=features_black.values, color='black')

ax.set_xlabel('Features')
ax.set_ylabel('Price Effect')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.set_title("Features Impact on Price");
ax.tick_params(axis='x', labelrotation = 50)
ax.axhline (y = 0, color = "black")
```

Out[71]: `<matplotlib.lines.Line2D at 0x12002682340>`



In [72]:
```
#let's make our plot
fig, ax = plt.subplots(figsize = (10,8))

#plot all of the home sales
X = kc['sqft_living']
```

```
y = kc['price']

ax.scatter(X, y, color='pink', label = 'home sales')

#plot best fit line of homesales using our very initial model
ax.plot(X, baseline_results.predict(sm.add_constant(X)), color = 'blue', label = "Origi

#we're going to create our "new" best fit line of sqft vs price based on our model
coeff = X_combined_greenbelt_results.params['const']
slope = X_combined_greenbelt_results.params['sqft_living']
adjusted_best_fit = X * slope + coeff
ax.plot(X, adjusted_best_fit, color = 'black',label = "Model Best-Fit")

#let's plot some vertical lines to show the square footage for a 1.009M home from origi
ax.axvline (x = 2000, color = "blue", linestyle = ":", ymax=.25)
ax.axhline (y = 1009000, color = "blue", linestyle = ":", xmax = .45)
ax.axvline (x = 2500, color = "blue", linestyle = ":", ymax=.25)
#ax.axhline (y = 1910000, color = "green", linestyle = ":", xmax = .73)

#plot characteristics
ax.set_xlabel('Square Footage')
ax.set_ylabel('Sales Price')
ax.yaxis.set_major_formatter('${x:,.0f}')
ax.xaxis.set_major_formatter('{x:,.0f}')
fig.suptitle("Non-Mansions sold in Kings County, WA, Fiscal year 2021-22");
ax.legend()
fig.tight_layout()
```



We can see here, that home would have to be about 2500 sq. ft. to sell for the same amount that a 2,000 sq.ft. home would have to in our previous analyis. Why? Because now that we account for other factors (zipcode, additional features) the value of square footage has gone down.

11. Conclusion

We were able to create a model that account for a model with reasonabale accuracy.

We confirmed that sqaure footage and zip code are the two largest factors when pricing a home.

A linear regression model built iteratively was able to account for 72% of the variance in the housing price, with an average error of approximately $212,000.

Real estate developers should build houses that can accomodate an everage number of bedrooms and bathrooms in desirable neighborhoods, and no more.