Final Project Submission

Please fill out:

- Student name: Jon Marks
- Student pace: self paced
- Scheduled project review date/time: 5/22/23 3 PM
- Instructor name: Morgan Jones
- Blog post URL: https://www.blogger.com/blog/posts/4766829666294336498 (https://www.blogger.com/blog/posts/4766829666294336498)

Housing Analysis Project

Business Understanding

The client is a housing planner that must set prices and wants to use market data to do so. It is necessary to know the impact on the housing price of various real estate metrics, so that a price can be estimated.

Data Understanding

The data is housing data from a Northwestern county and comes from the county government. Key data categories include price, number of rooms, various square footage metrics, and age of the house. Each row of data represents a different house sold.

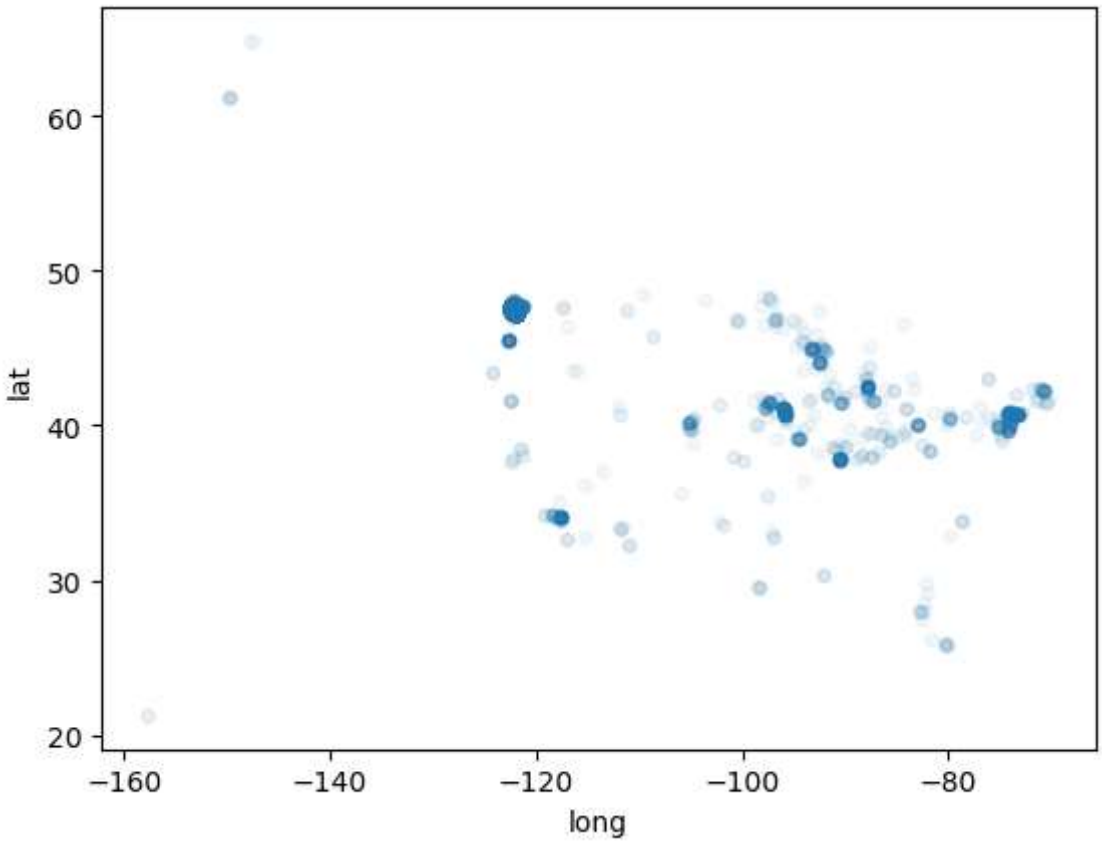
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:
```
data = pd.read_csv('./data/kc_house_data.csv')
```

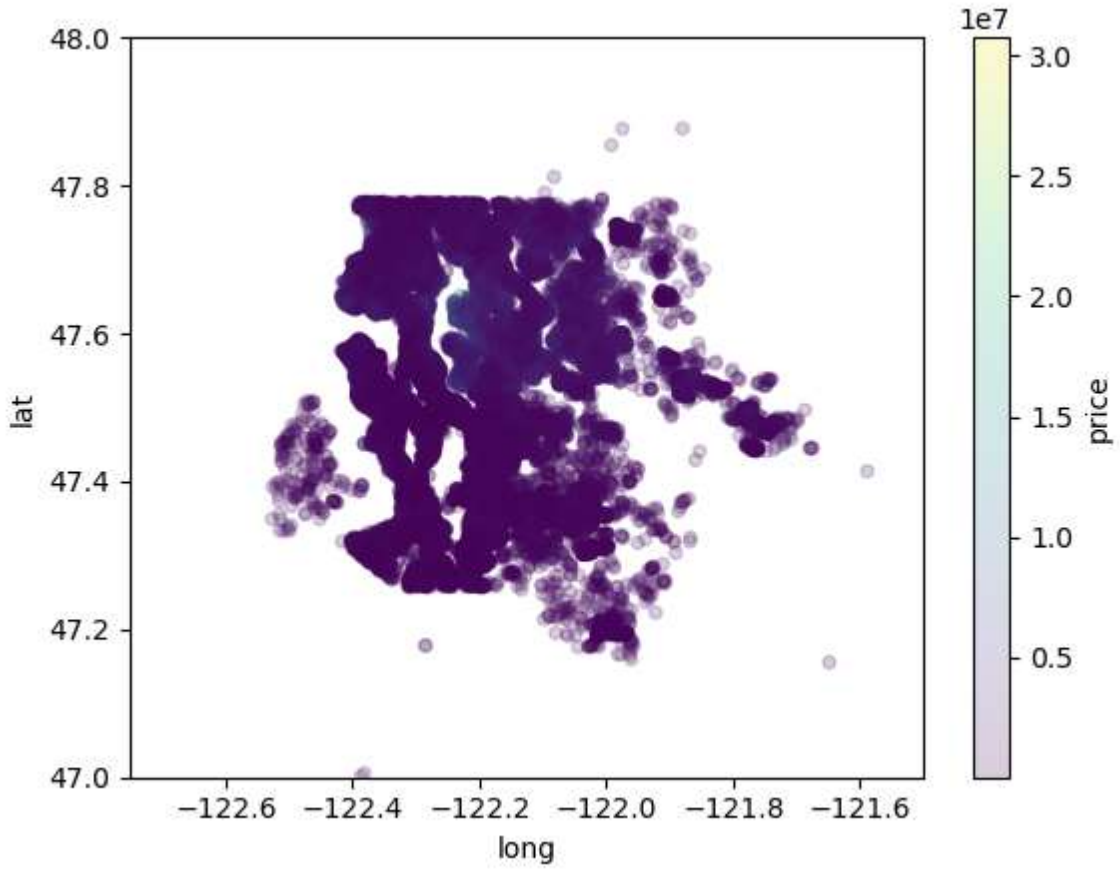
The distribution of the houses is such that nearly all observations are within the Greater Seattle Area (King County) while the rest are distributed over the entire United States:

In [3]:
```
data.plot(kind = 'scatter', x = "long", y="lat", alpha=.05)
```

Out[3]: <AxesSubplot:xlabel='long', ylabel='lat'>



In [4]:
```
fig, ax = (15,5)
data.plot.scatter('long', 'lat',
                  c = 'price',
                  colormap = 'viridis', alpha=.2)
plt.ylim(47,48)
plt.xlim(-122.75,-121.5);
```



Above, there appear to be some patterns/dividing line in King County (non-outlier) map data, and an area near the center of the map with higher price houses. This will be further analyzed in the data analysis section.

In [ ]:
```
```

Remove outlier data:

In [5]:
```
dataCut=data.drop(data.loc[data['lat']<47.12].index).copy()
```

In [6]:
```
dataCut=dataCut.drop(dataCut.loc[47.8<dataCut['lat']].index).copy()
```

In [7]:
```
dataCut=dataCut.drop(dataCut.loc[dataCut['long']<-122.56].index).copy()
```

In [8]:
```
dataCut=dataCut.drop(dataCut.loc[-121.75<dataCut['long']].index).copy()
```

In [9]:
```
dataCut.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             29022 non-null  int64  
 1   date           29022 non-null  object 
 2   price          29022 non-null  float64
 3   bedrooms       29022 non-null  int64  
 4   bathrooms      29022 non-null  float64
 5   sqft_living    29022 non-null  int64  
 6   sqft_lot       29022 non-null  int64  
 7   floors         29022 non-null  float64
 8   waterfront     29022 non-null  object 
 9   greenbelt      29022 non-null  object 
 10  nuisance       29022 non-null  object 
 11  view           29022 non-null  object 
 12  condition      29022 non-null  object 
 13  grade          29022 non-null  object 
 14  heat_source    29001 non-null  object 
 15  sewer_system   29011 non-null  object 
 16  sqft_above     29022 non-null  int64  
 17  sqft_basement  29022 non-null  int64  
 18  sqft_garage    29022 non-null  int64  
 19  sqft_patio     29022 non-null  int64  
 20  yr_built       29022 non-null  int64  
 21  yr_renovated   29022 non-null  int64  
 22  address        29022 non-null  object 
 23  lat            29022 non-null  float64
 24  long           29022 non-null  float64
dtypes: float64(5), int64(10), object(10)
memory usage: 5.8+ MB
```

In [10]:
```
dataCut.price.mean()
```

Out[10]: 1113289.6514712977

In [11]:
```
dataCut.price.max()
```

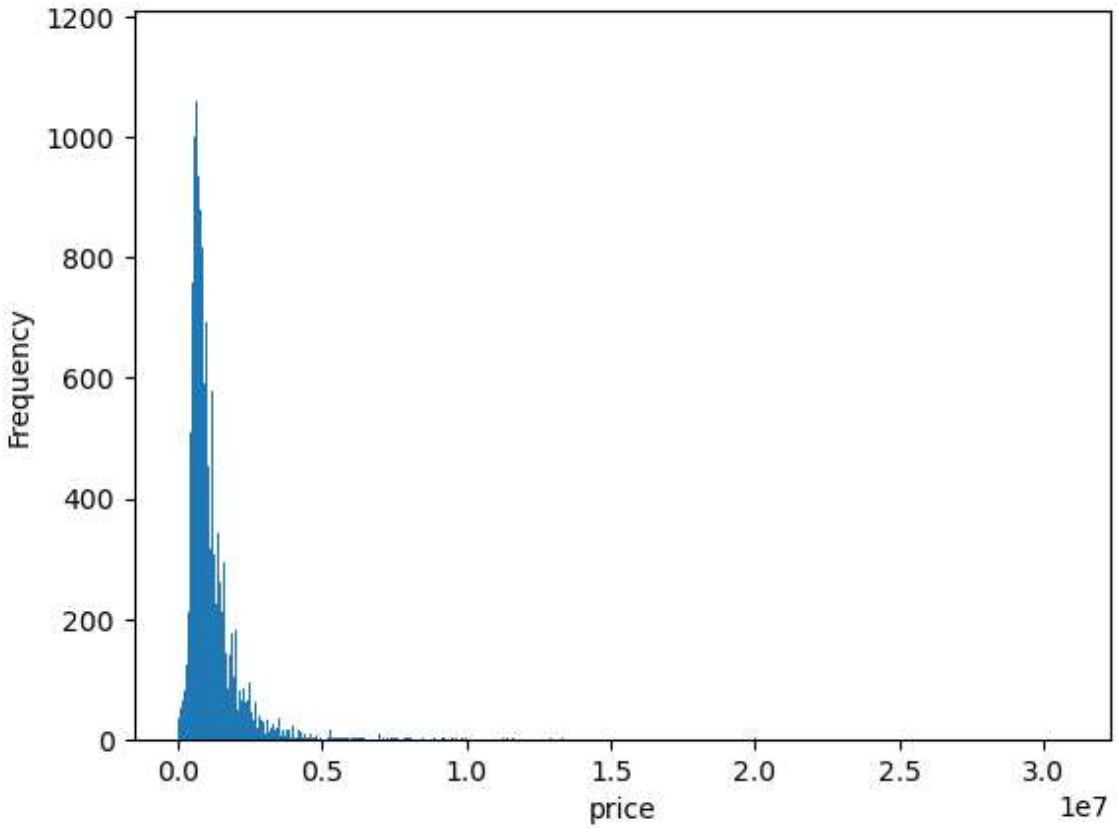
Out[11]: 30750000.0

In [12]:
```
dataCut.price.median()
```

Out[12]: 865000.0

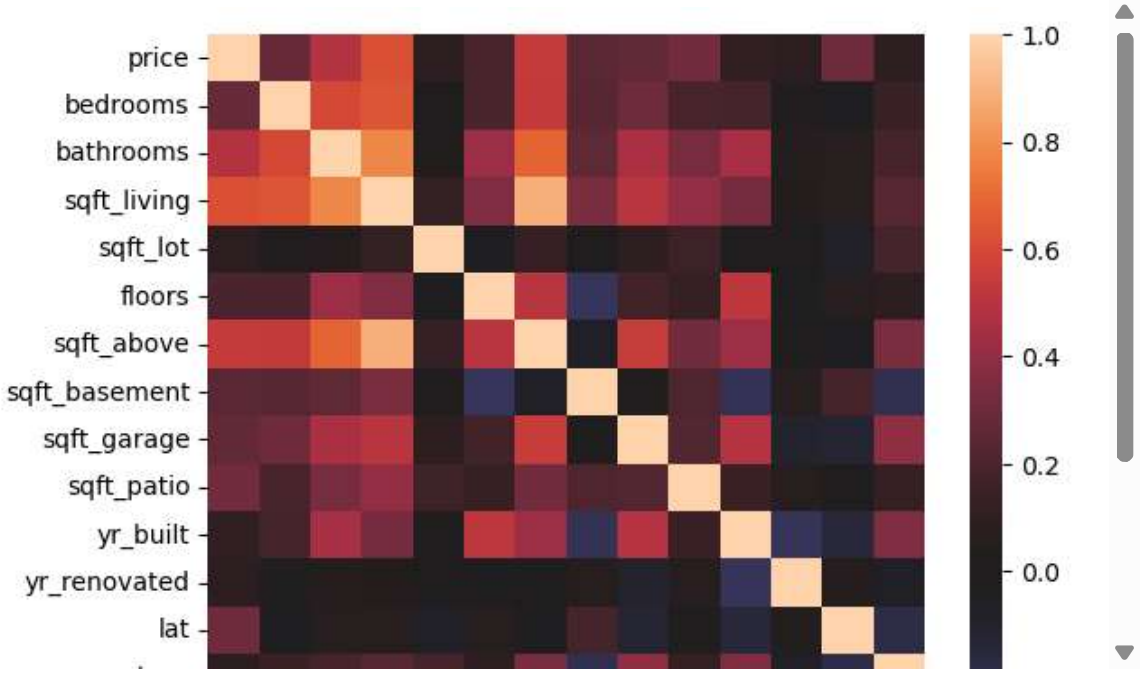
In [13]:

```
fig, ax = plt.subplots()
dataCut.price.plot(kind='hist', bins=1000)

ax.set_xlabel("price");
```



Above, prices have a non-normal, right skewed distribution, with potential outliers. The median is 865,000. The mean is 1,113,290.

In [14]:
```
sns.heatmap(dataCut.iloc[:,2:].corr(), center=0);
```



Above, sqft_living most correlated with price. Also highly correlated to other potential predictors.

In [15]:
```
abs(dataCut.corr()) > 0.5
```

Out[15]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	sqft_above
id	True	False	False	False	False	False	False	False
price	False	True	False	False	True	False	False	True
bedrooms	False	False	True	True	True	False	False	True
bathrooms	False	False	True	True	True	False	False	True
sqft_living	False	True	True	True	True	False	False	True
sqft_lot	False	False	False	False	False	True	False	False
floors	False	False	False	False	False	False	True	True
sqft_above	False	True	True	True	True	False	True	True
sqft_basement	False	False	False	False	False	False	False	False
sqft_garage	False	False	False	False	True	False	False	True
sqft_patio	False	False	False	False	False	False	False	False
yr_built	False	False	False	False	False	False	True	False
yr_renovated	False	False	False	False	False	False	False	False
lat	False	False	False	False	False	False	False	False
long	False	False	False	False	False	False	False	False

Data Analysis

In [176]:
```
dataCut.head(5)
```

Out[176]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
0	7399300360	5/24/2022	675000.0	4	1.0	1180	7140	1.0	
1	8910500230	12/13/2021	920000.0	5	2.5	2770	6703	1.0	
2	1180000275	9/29/2021	311000.0	6	2.0	2880	6156	1.0	
3	1604601802	12/14/2021	775000.0	3	3.0	2160	1400	2.0	
4	8562780790	8/24/2021	592500.0	2	2.0	1120	758	2.0	

5 rows × 28 columns

In [17]:
```
dataCut.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 25 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             29022 non-null  int64  
 1   date           29022 non-null  object 
 2   price          29022 non-null  float64
 3   bedrooms       29022 non-null  int64  
 4   bathrooms      29022 non-null  float64
 5   sqft_living    29022 non-null  int64  
 6   sqft_lot       29022 non-null  int64  
 7   floors         29022 non-null  float64
 8   waterfront     29022 non-null  object 
 9   greenbelt      29022 non-null  object 
 10  nuisance       29022 non-null  object 
 11  view           29022 non-null  object 
 12  condition      29022 non-null  object 
 13  grade          29022 non-null  object 
 14  heat_source    29001 non-null  object 
 15  sewer_system   29011 non-null  object 
 16  sqft_above     29022 non-null  int64  
 17  sqft_basement  29022 non-null  int64  
 18  sqft_garage    29022 non-null  int64  
 19  sqft_patio     29022 non-null  int64  
 20  yr_built       29022 non-null  int64  
 21  yr_renovated   29022 non-null  int64  
 22  address        29022 non-null  object 
 23  lat            29022 non-null  float64
 24  long           29022 non-null  float64
dtypes: float64(5), int64(10), object(10)
memory usage: 5.8+ MB
```

In [18]:
```
dataCut.heat_source.unique()
```

Out[18]:
```
array(['Gas', 'Oil', 'Electricity', 'Gas/Solar', 'Electricity/Solar',
       'Other', nan, 'Oil/Solar'], dtype=object)
```

In [19]:
```
dataCut.sewer_system.unique()
```

Out[19]:
```
array(['PUBLIC', 'PRIVATE', 'PRIVATE RESTRICTED', nan,
       'PUBLIC RESTRICTED'], dtype=object)
```

In [20]:
```
dataCut.yr_built.value_counts().sort_index()
```

Out[20]:
```
1900     117
1901      39
1902      36
1903      56
1904      73
         ...
2018     331
2019     259
2020     343
2021    1318
2022     251
Name: yr_built, Length: 123, dtype: int64
```

In [21]:
```
dataCut.grade.unique()
```

Out[21]:
```
array(['7 Average', '9 Better', '8 Good', '6 Low Average', '10 Very Good',
       '5 Fair', '11 Excellent', '12 Luxury', '4 Low', '13 Mansion',
       '3 Poor', '1 Cabin', '2 Substandard'], dtype=object)
```

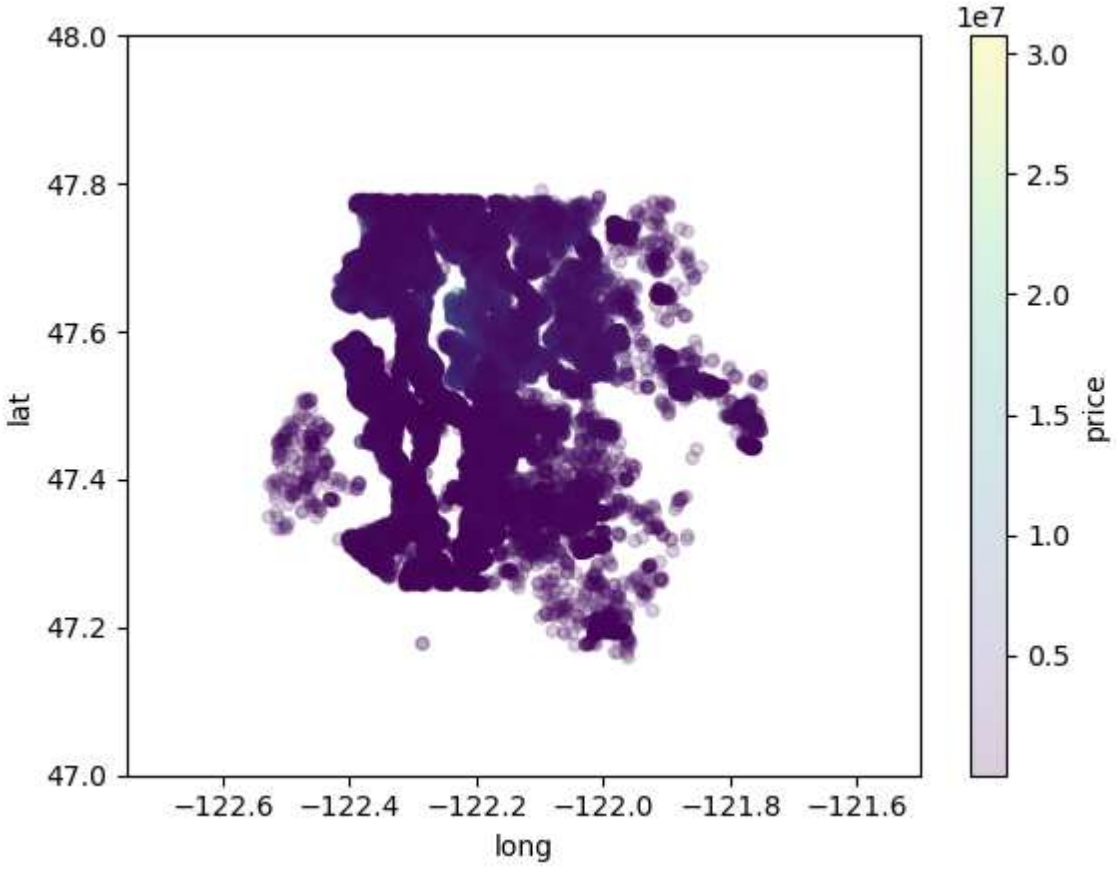
In [22]:
```
dataCut.condition.unique()
```

Out[22]:
```
array(['Good', 'Average', 'Very Good', 'Fair', 'Poor'], dtype=object)
```

In [ ]:

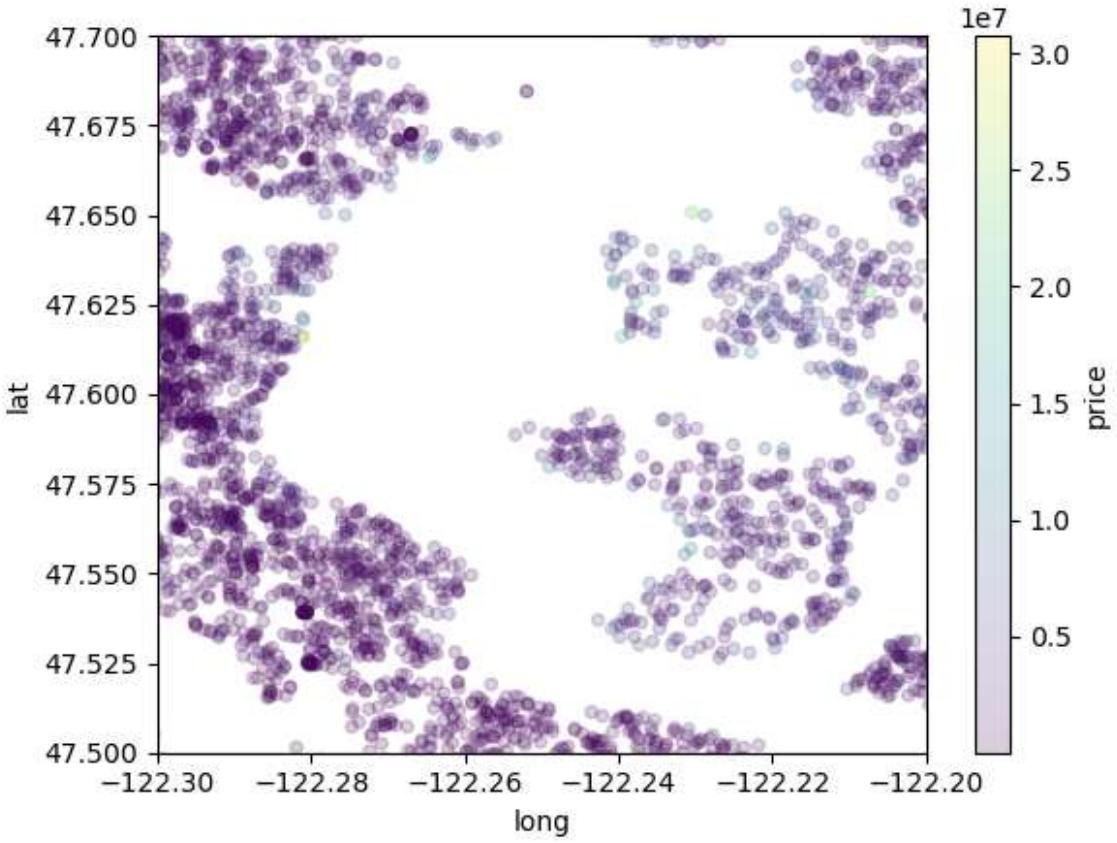
Further analyze the house price map in order to create categorical variable(s) which correspond to different prize areas.

In [23]:
```
fig, ax = (15,5)
dataCut.plot.scatter('long', 'lat',
                     c = 'price',
                     colormap = 'viridis', alpha=.2)
plt.ylim(47,48)
plt.xlim(-122.75,-121.5);
```



Above, there appear to be some patterns/dividing line in King County (non-outlier) map data, and an area near the center of the map with higher price houses. Below, is zoomed in map of the this area.

In [24]:
```
fig, ax = (15,5)
dataCut.plot.scatter('long', 'lat',
                     c = 'price',
                     colormap = 'viridis', alpha=.2)
plt.ylim(47.5,47.7)
plt.xlim(-122.3,-122.2);
```



Further analyze this high price area:

In [25]:
```
data1=dataCut.drop(dataCut.loc[dataCut['lat']<47.525].index)
```

In [26]:
```
data2=data1.drop(data1.loc[47.65<data1['lat']].index)
```

In [27]:
```
data3=data2.drop(data2.loc[data2['long']<-122.25].index)
```

In [28]:
```
data4=data3.drop(data3.loc[-122.20<data3['long']].index)
```

In [177]:
```
data4.head(5)
```

Out[177]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
27	5424700190	2/26/2022	4500000.0	4	3.0	2760	13150	1.5
36	1925059107	6/29/2021	2450000.0	4	3.5	2300	8370	2.0
74	5453600300	9/29/2021	1825000.0	5	3.0	3470	10893	1.0
118	5425700191	12/15/2021	3000000.0	3	1.5	2040	14284	1.0
140	1651600020	9/7/2021	6438000.0	5	4.5	6110	21562	2.0

5 rows × 25 columns

In [30]:
```
len(data4)
```

Out[30]: 646

The Jumbo area is largely comprised of the Clyde Hill area and the whole area is 646 houses.

Create categorical variable for this Jumbo area:

In [31]:
```
data['Jumbo']=0
for i in range(len(data)):
    if (data['lat'][i]<47.525) | (data['long'][i]<-122.25) | (47.65<data['l
        data['Jumbo'][i]= 0
    else:
        data['Jumbo'][i]= 1
```

```
C:\Users\jmark\AppData\Local\Temp\ipykernel_29944\109765181.py:4: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  data['Jumbo'][i]= 0
C:\Users\jmark\AppData\Local\Temp\ipykernel_29944\109765181.py:6: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  data['Jumbo'][i]= 1
```

Convert grade data to numeric

In [32]:
```
grade_array = list(dataCut['grade'])
```

In [33]:
```
grade_array_split =[]
for i in grade_array:
    split = i.split(" ")
    grade_array_split.append(split[0])
```

In [34]:
```
grade_num= grade_array_split
```

In [35]:
```
dataCut['grade_num']=grade_num
```

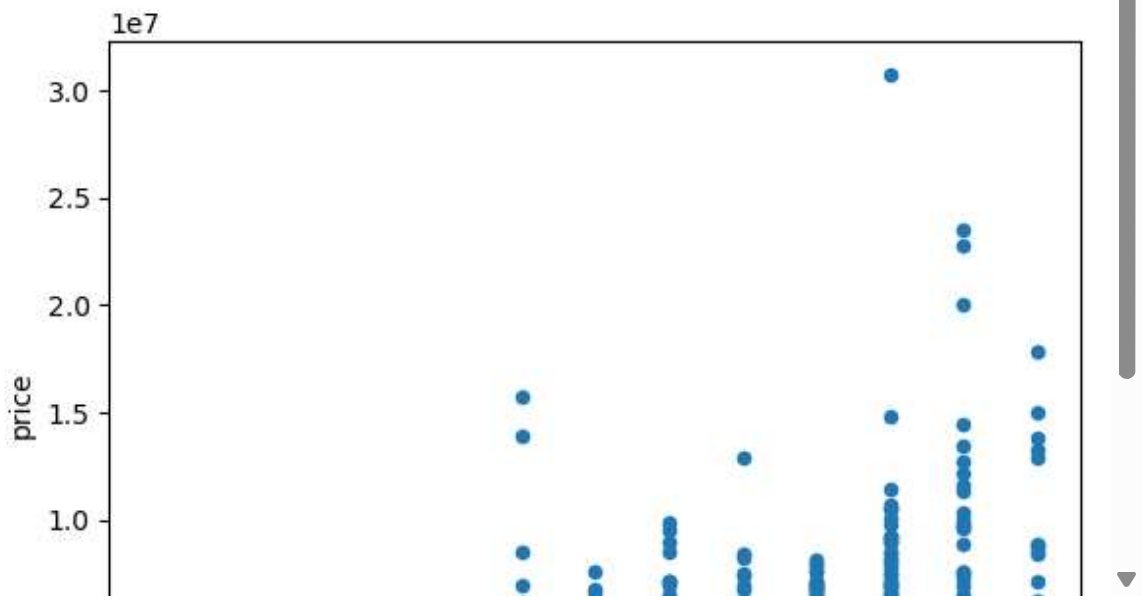
In [36]:
```
dataCut['grade_num']=dataCut['grade_num'].astype(int)
```

In [37]:
```
dataCut['grade_num']
```

Out[37]:
```
0        7
1        7
2        7
3        9
4        7
        ..
30150    8
30151    7
30152    7
30153    8
30154    7
Name: grade_num, Length: 29022, dtype: int32
```

In [38]:
```
dataCut.plot(kind = 'scatter', x = 'grade_num', y='price')
```

Out[38]:
```
<AxesSubplot:xlabel='grade_num', ylabel='price'>
```



Above, apparent correlation between grade and price.

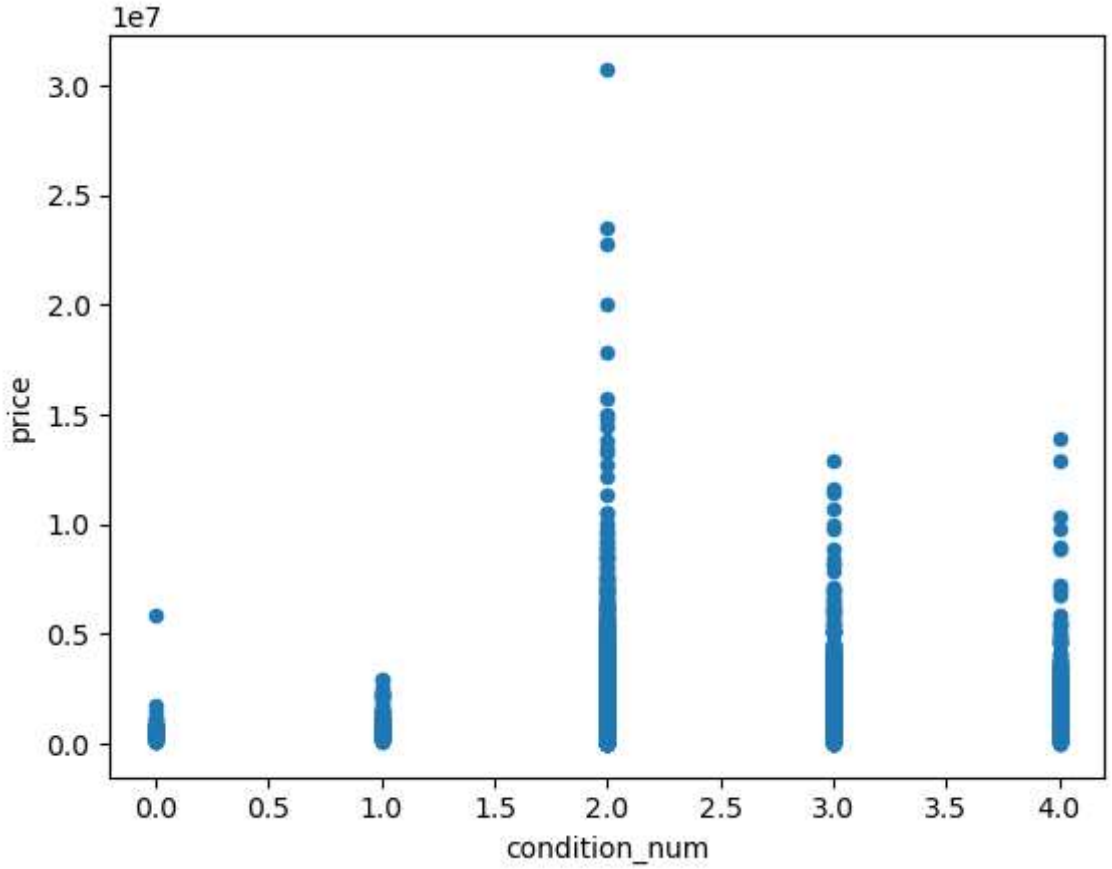
Convert condition data to numeric

In [39]:
```
condition_num = dataCut['condition'].replace({'Good': 3, 'Average':2, 'Very
```

In [40]:
```
dataCut['condition_num'] =condition_num
```

In [41]:
```
dataCut.plot(kind = 'scatter', x = 'condition_num', y='price')
```

Out[41]: <AxesSubplot:xlabel='condition_num', ylabel='price'>



Above, some correlation between condition and price.

In [42]:
```
dataCut.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 27 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             29022 non-null  int64  
 1   date           29022 non-null  object 
 2   price          29022 non-null  float64
 3   bedrooms       29022 non-null  int64  
 4   bathrooms      29022 non-null  float64
 5   sqft_living    29022 non-null  int64  
 6   sqft_lot       29022 non-null  int64  
 7   floors         29022 non-null  float64
 8   waterfront     29022 non-null  object 
 9   greenbelt      29022 non-null  object 
 10  nuisance       29022 non-null  object 
 11  view           29022 non-null  object 
 12  condition      29022 non-null  object 
 13  grade          29022 non-null  object 
 14  heat_source    29001 non-null  object 
 15  sewer_system   29011 non-null  object 
 16  sqft_above     29022 non-null  int64  
 17  sqft_basement  29022 non-null  int64  
 18  sqft_garage    29022 non-null  int64  
 19  sqft_patio     29022 non-null  int64  
 20  yr_built       29022 non-null  int64  
 21  yr_renovated   29022 non-null  int64  
 22  address        29022 non-null  object 
 23  lat            29022 non-null  float64
 24  long           29022 non-null  float64
 25  grade_num      29022 non-null  int32  
 26  condition_num  29022 non-null  int64  
dtypes: float64(5), int32(1), int64(11), object(10)
memory usage: 7.1+ MB
```

Convert view variable to numeric

In [43]:
```
view_num = dataCut['view'].replace({'NONE': 0, 'FAIR':1, 'AVERAGE':2, 'GOOD
```

In [44]:
```
dataCut['view_num']=view_num
```

In [45]:
```
dataCut.corr()
```

Out[45]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
id	1.000000	-0.029777	-0.003910	-0.009854	-0.026248	-0.121002	0.036233
price	-0.029777	1.000000	0.290882	0.487924	0.617183	0.085703	0.199624
bedrooms	-0.003910	0.290882	1.000000	0.593953	0.631284	-0.000379	0.194789
bathrooms	-0.009854	0.487924	0.593953	1.000000	0.780259	0.036096	0.427116
sqft_living	-0.026248	0.617183	0.631284	0.780259	1.000000	0.115622	0.354304
sqft_lot	-0.121002	0.085703	-0.000379	0.036096	0.115622	1.000000	-0.022163
floors	0.036233	0.199624	0.194789	0.427116	0.354304	-0.022163	1.000000
sqft_above	-0.020607	0.547395	0.537776	0.681415	0.881426	0.126223	0.504484
sqft_basement	-0.015847	0.246291	0.237436	0.262344	0.338553	0.000687	-0.242824
sqft_garage	-0.006668	0.266950	0.305357	0.461469	0.502945	0.083983	0.178207
sqft_patio	-0.041931	0.317956	0.192921	0.333951	0.405950	0.153948	0.122778
yr_built	0.024931	0.105748	0.180289	0.455002	0.325989	0.011554	0.520803
yr_renovated	-0.028768	0.086514	0.012580	0.043859	0.037263	0.007585	-0.015489
lat	-0.000410	0.299084	-0.016751	0.048819	0.049595	-0.074523	0.049505
long	0.013862	0.089415	0.137826	0.184829	0.237786	0.180904	0.084809
grade_num	0.005402	0.577762	0.388521	0.652401	0.738025	0.055215	0.471058
condition_num	-0.010836	-0.012346	0.021841	-0.063770	-0.066313	-0.002972	-0.264362
view_num	-0.007610	0.321968	0.051333	0.159689	0.241858	0.089262	0.006164

Create dummy variables

In [46]:
```
data_wdum= pd.get_dummies(dataCut, columns = ['waterfront','heat_source','g
```

In [47]:
```
data_wdum.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 35 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   id                                29022 non-null  int64  
 1   date                              29022 non-null  object 
 2   price                             29022 non-null  float64
 3   bedrooms                          29022 non-null  int64  
 4   bathrooms                         29022 non-null  float64
 5   sqft_living                       29022 non-null  int64  
 6   sqft_lot                          29022 non-null  int64  
 7   floors                            29022 non-null  float64
 8   view                              29022 non-null  object 
 9   condition                         29022 non-null  object 
 10  grade                             29022 non-null  object 
 11  sqft_above                        29022 non-null  int64  
 12  sqft_basement                     29022 non-null  int64  
 13  sqft_garage                       29022 non-null  int64  
 14  sqft_patio                        29022 non-null  int64  
 15  yr_built                          29022 non-null  int64  
 16  yr_renovated                      29022 non-null  int64  
 17  address                           29022 non-null  object 
 18  lat                               29022 non-null  float64
 19  long                              29022 non-null  float64
 20  grade_num                         29022 non-null  int32  
 21  condition_num                     29022 non-null  int64  
 22  view_num                          29022 non-null  int64  
 23  waterfront_YES                    29022 non-null  uint8  
 24  heat_source_Electricity/Solar     29022 non-null  uint8  
 25  heat_source_Gas                   29022 non-null  uint8  
 26  heat_source_Gas/Solar             29022 non-null  uint8  
 27  heat_source_Oil                   29022 non-null  uint8  
 28  heat_source_Oil/Solar             29022 non-null  uint8  
 29  heat_source_Other                 29022 non-null  uint8  
 30  greenbelt_YES                     29022 non-null  uint8  
 31  nuisance_YES                      29022 non-null  uint8  
 32  sewer_system_PRIVATE RESTRICTED   29022 non-null  uint8  
 33  sewer_system_PUBLIC               29022 non-null  uint8  
 34  sewer_system_PUBLIC RESTRICTED    29022 non-null  uint8  
dtypes: float64(5), int32(1), int64(12), object(5), uint8(12)
memory usage: 6.5+ MB
```

Type *Markdown* and LaTeX: α^2

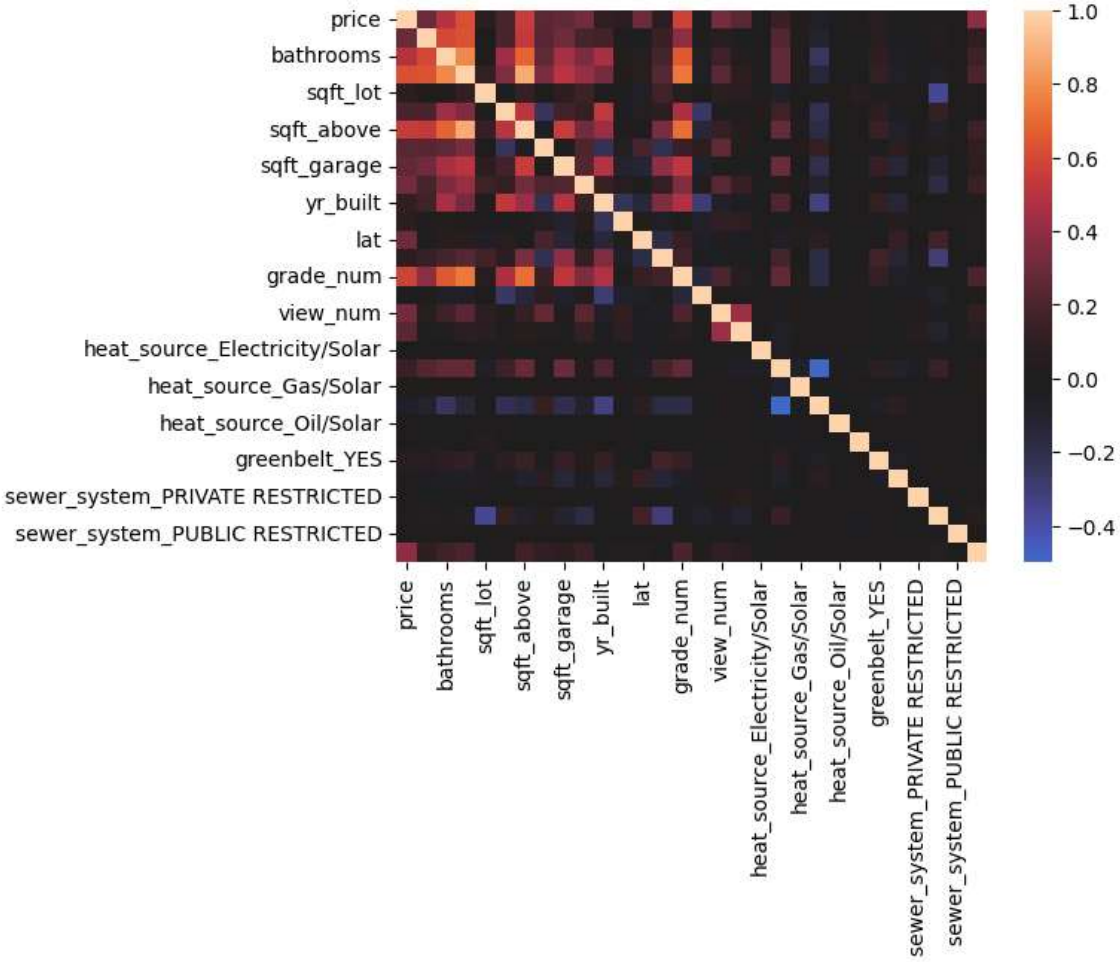
In [48]:
```
data_wdum['Jumbo']=data['Jumbo']
```

In [49]:
```
df = data_wdum
```

Check correlations with new dummy variables

In [50]:

```
sns.heatmap(df.iloc[:,1:].corr(), center=0);
```



In [51]:
```
df.corr()
```

Out[51]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_l
id	1.000000	-0.029777	-0.003910	-0.009854	-0.026248	-0.1210
price	-0.029777	1.000000	0.290882	0.487924	0.617183	0.0857
bedrooms	-0.003910	0.290882	1.000000	0.593953	0.631284	-0.0003
bathrooms	-0.009854	0.487924	0.593953	1.000000	0.780259	0.0360
sqft_living	-0.026248	0.617183	0.631284	0.780259	1.000000	0.1156
sqft_lot	-0.121002	0.085703	-0.000379	0.036096	0.115622	1.0000
floors	0.036233	0.199624	0.194789	0.427116	0.354304	-0.0221
sqft_above	-0.020607	0.547395	0.537776	0.681415	0.881426	0.1262
sqft_basement	-0.015847	0.246291	0.237436	0.262344	0.338553	0.0006
sqft_garage	-0.006668	0.266950	0.305357	0.461469	0.502945	0.0839
sqft_patio	-0.041931	0.317956	0.192921	0.333951	0.405950	0.1539
yr_built	0.024931	0.105748	0.180289	0.455002	0.325989	0.0115
yr_renovated	-0.028768	0.086514	0.012580	0.043859	0.037263	0.0075
lat	-0.000410	0.299084	-0.016751	0.048819	0.049595	-0.0745
long	0.013862	0.089415	0.137826	0.184829	0.237786	0.1809
grade_num	0.005402	0.577762	0.388521	0.652401	0.738025	0.0552
condition_num	-0.010836	-0.012346	0.021841	-0.063770	-0.066313	-0.0029
view_num	-0.007610	0.321968	0.051333	0.159689	0.241858	0.0892
waterfront_YES	-0.035859	0.242572	-0.032478	0.046179	0.080684	0.0700
heat_source_Electricity/Solar	0.008837	-0.008305	0.000536	0.003018	-0.006277	-0.0026
heat_source_Gas	0.072319	0.145362	0.216746	0.272899	0.263925	-0.0771
heat_source_Gas/Solar	0.000648	0.037038	0.013456	0.033124	0.031484	0.0010
heat_source_Oil	-0.019038	-0.071064	-0.100314	-0.254048	-0.150764	0.0086
heat_source_Oil/Solar	-0.003037	-0.002369	-0.008215	-0.007617	-0.003196	-0.0014
heat_source_Other	-0.012518	0.000569	-0.018138	-0.013846	-0.007569	0.0578
greenbelt_YES	0.057002	0.068618	0.062557	0.095076	0.114615	-0.0136
nuisance_YES	-0.052889	0.006468	-0.043141	-0.051060	-0.055115	0.0092
sewer_system_PRIVATE RESTRICTED	-0.009042	-0.005387	-0.016186	-0.006655	-0.009742	0.0020
sewer_system_PUBLIC	0.152617	0.022527	0.041912	0.031075	-0.056398	-0.3596
sewer_system_PUBLIC RESTRICTED	-0.008608	-0.000653	0.004799	0.003879	0.002989	-0.0015
Jumbo	-0.014611	0.391320	0.096769	0.158856	0.202239	-0.0044

31 rows × 31 columns

In [52]:
```
df_c = df.loc[:,['price','sqft_living','heat_source_Electricity/Solar','wat
```

In [53]:
```
df_c.corr()
```

Out[53]:

	price	sqft_living	heat_source_Electricity/Solar	waterfront_Y
price	1.000000	0.617183	-0.008305	0.242
sqft_living	0.617183	1.000000	-0.006277	0.080
heat_source_Electricity/Solar	-0.008305	-0.006277	1.000000	-0.005
waterfront_YES	0.242572	0.080684	-0.005798	1.000
view_num	0.321968	0.241858	0.012878	0.423
yr_built	0.105748	0.325989	0.003748	-0.038
condition_num	-0.012346	-0.066313	0.015107	0.001
grade_num	0.577762	0.738025	-0.005095	0.050
yr_renovated	0.086514	0.037263	0.016314	0.098

Sqft_living most correlated with price. Also highly correlated are bathrooms, sqft_above, sqft_garage, grade, view. Only categorical variables materially correlated with price are waterfront properties and Jumbo area.

Data Modeling

Run baseline model

Start with one variable, the most correlated with price: Sqft_living

In [54]:
```
import statsmodels.api as sm
```

In [55]:
```
baseline_model = sm.OLS(df['price'], sm.add_constant(df['sqft_living']))
baseline_results = baseline_model.fit()
```

In [56]:
```
baseline_results.summary()
```

Out[56]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.381
Model:	OLS	**Adj. R-squared:**	0.381
Method:	Least Squares	**F-statistic:**	1.786e+04
Date:	Fri, 19 May 2023	**Prob (F-statistic):**	0.00
Time:	10:24:41	**Log-Likelihood:**	-4.3205e+05
No. Observations:	29022	**AIC:**	8.641e+05
Df Residuals:	29020	**BIC:**	8.641e+05
Df Model:	1		
Covariance Type:	nonrobust		

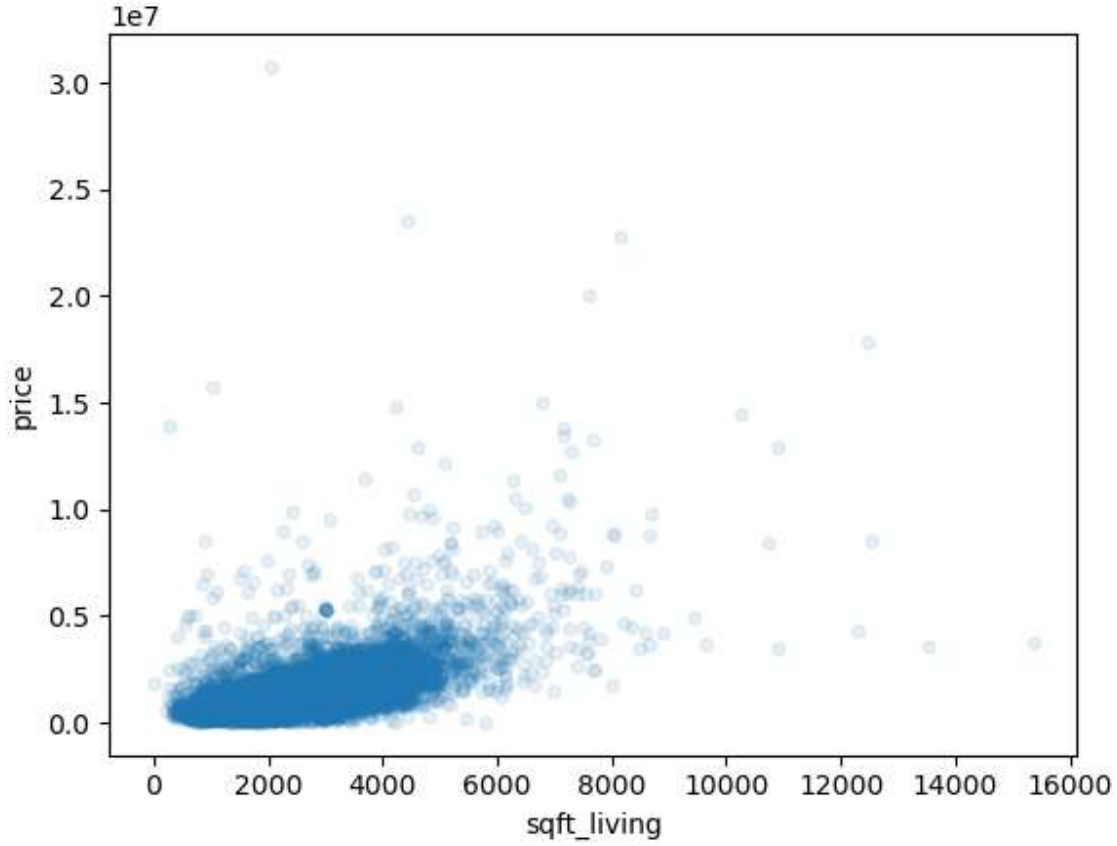
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-9.416e+04	9942.048	-9.471	0.000	-1.14e+05	-7.47e+04
sqft_living	567.3132	4.246	133.625	0.000	558.992	575.635

Omnibus:	41953.578	**Durbin-Watson:**	1.939
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	49427411.165
Skew:	8.217	**Prob(JB):**	0.00
Kurtosis:	204.505	**Cond. No.**	5.61e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.61e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [57]:
```
df.plot(kind = 'scatter',x='sqft_living',y= 'price', alpha = .1)
```

Out[57]:
```
<AxesSubplot:xlabel='sqft_living', ylabel='price'>
```



Above, there appears to be some curvature and hence non-linearity in this relationship.

Next model: Add to previous, variables with price correlation greater than .25 and sqft_living correlation less than .75 to increase rsquared.

In [58]:
```
df_X2 = df.loc[:,['sqft_living','bedrooms','sqft_garage', 'sqft_patio', 'gr
```

In [59]:
```
df_X2.head()
```

Out[59]:

	sqft_living	bedrooms	sqft_garage	sqft_patio	grade_num	view_num
0	1180	4	0	40	7	0
1	2770	5	0	240	7	2
2	2880	6	0	0	7	2
3	2160	3	200	270	9	2
4	1120	2	550	30	7	0

In [60]:
```
model_2575 = sm.OLS(df['price'], sm.add_constant(df_X2))
model_2575_results = model_2575.fit()
model_2575_results.summary()
```

Out[60]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.456
Model:	OLS	**Adj. R-squared:**	0.456
Method:	Least Squares	**F-statistic:**	4050.
Date:	Fri, 19 May 2023	**Prob (F-statistic):**	0.00
Time:	10:24:41	**Log-Likelihood:**	-4.3018e+05
No. Observations:	29022	**AIC:**	8.604e+05
Df Residuals:	29015	**BIC:**	8.604e+05
Df Model:	6		
Covariance Type:	nonrobust		

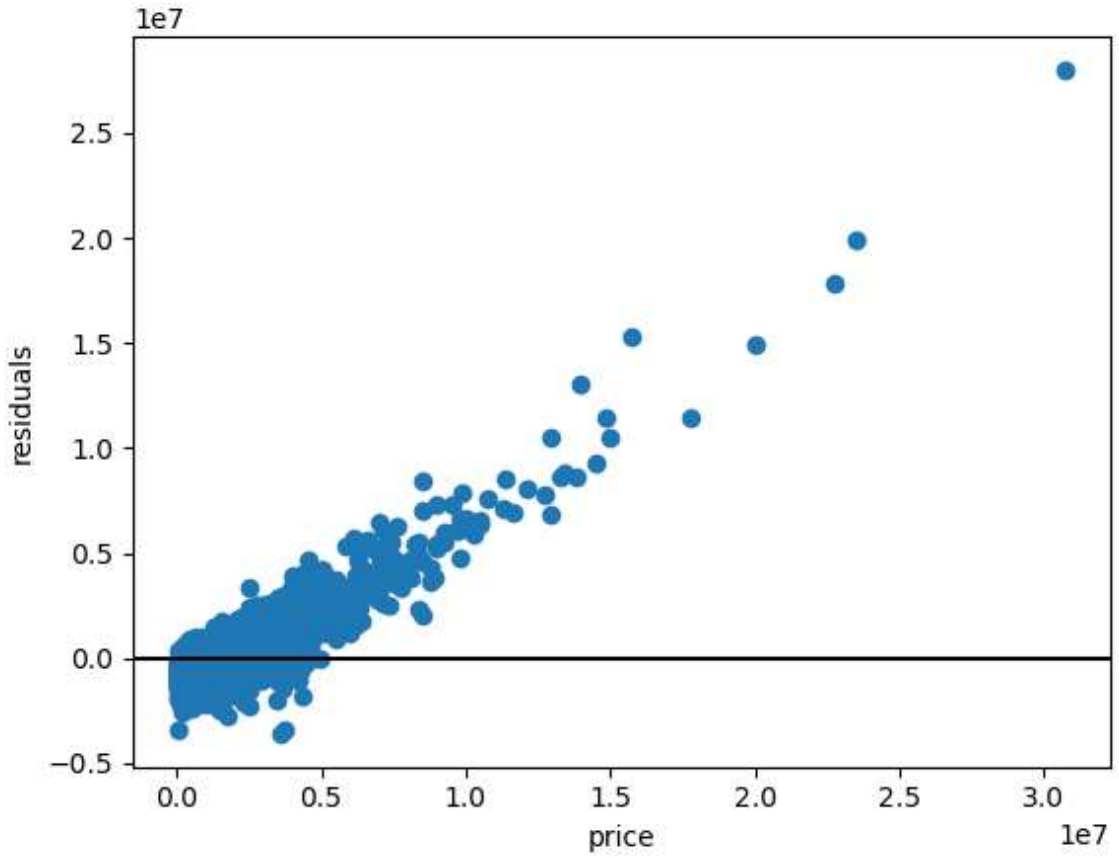
	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.067e+06	3.47e+04	-30.768	0.000	-1.14e+06	-9.99e+05
sqft_living	443.6263	7.490	59.227	0.000	428.945	458.308
bedrooms	-9.46e+04	5234.424	-18.074	0.000	-1.05e+05	-8.43e+04
sqft_garage	-291.8322	16.386	-17.810	0.000	-323.950	-259.715
sqft_patio	116.0772	17.604	6.594	0.000	81.573	150.582
grade_num	2.08e+05	5227.664	39.780	0.000	1.98e+05	2.18e+05
view_num	1.556e+05	4791.625	32.475	0.000	1.46e+05	1.65e+05

Omnibus:	42327.701	**Durbin-Watson:**	1.924
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	53877696.717
Skew:	8.341	**Prob(JB):**	0.00
Kurtosis:	213.419	**Cond. No.**	2.15e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.15e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [61]:

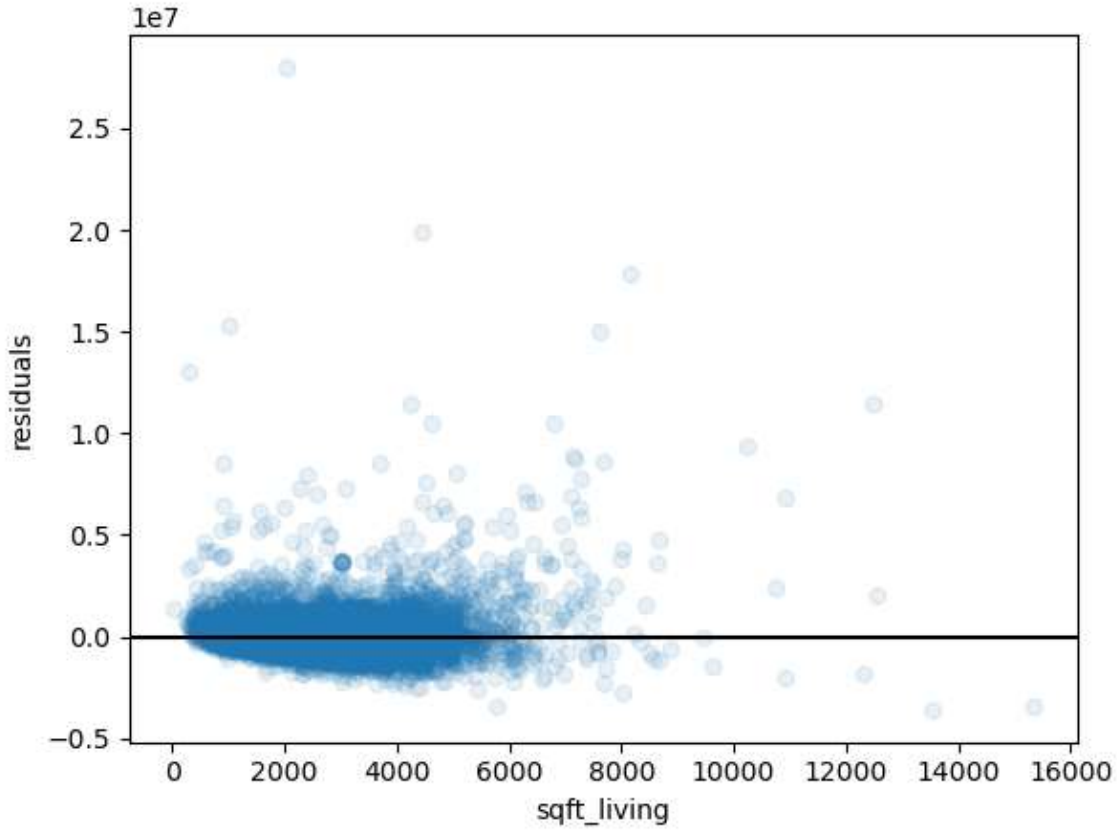
```
fig, ax = plt.subplots()

ax.scatter(df["price"], model_2575_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("price")
ax.set_ylabel("residuals");
```



Above, errors are but should not be increasing with the dependent variable. There are potential linearity and heteroskedaticity issues.

In [62]:

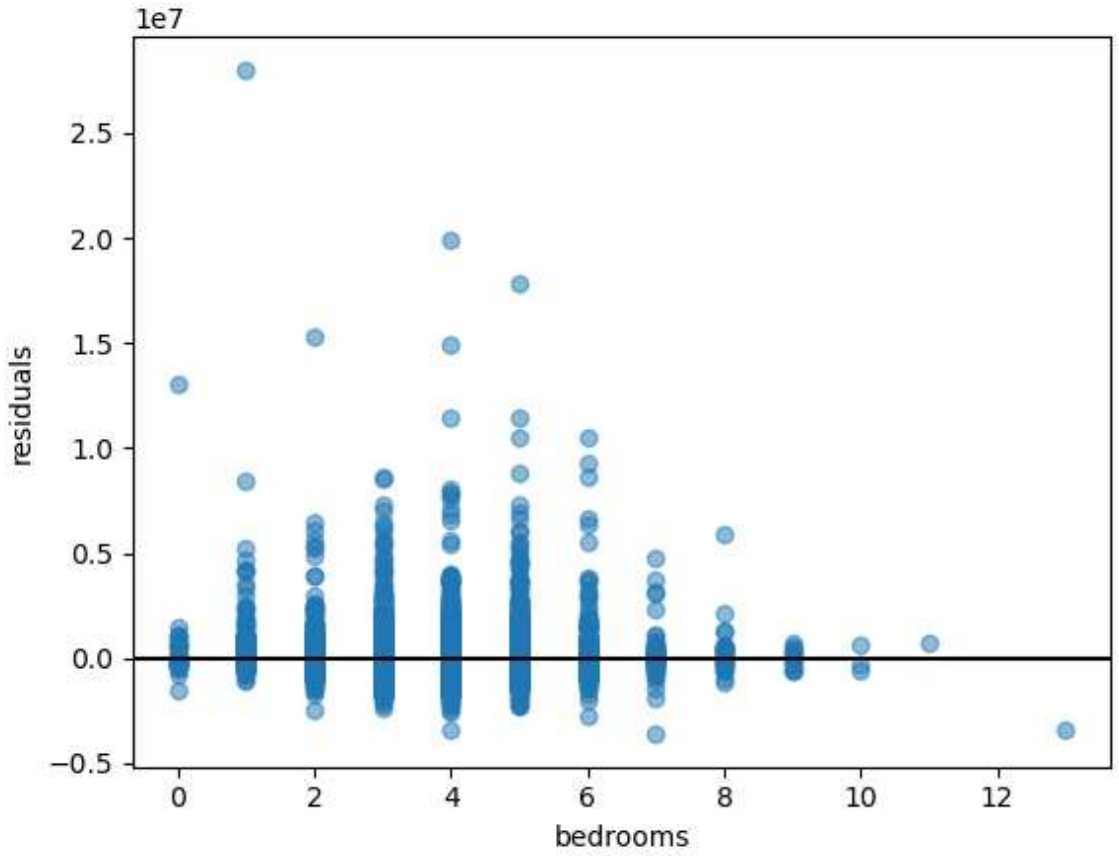
```
fig, ax = plt.subplots()

ax.scatter(df_X2["sqft_living"], model_2575_results.resid, alpha =.1)
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_living")
ax.set_ylabel("residuals");
```



Above, variance in residuals is increasing with sqft_living variabe, suggesting some heteroskedaticity.

In [63]:

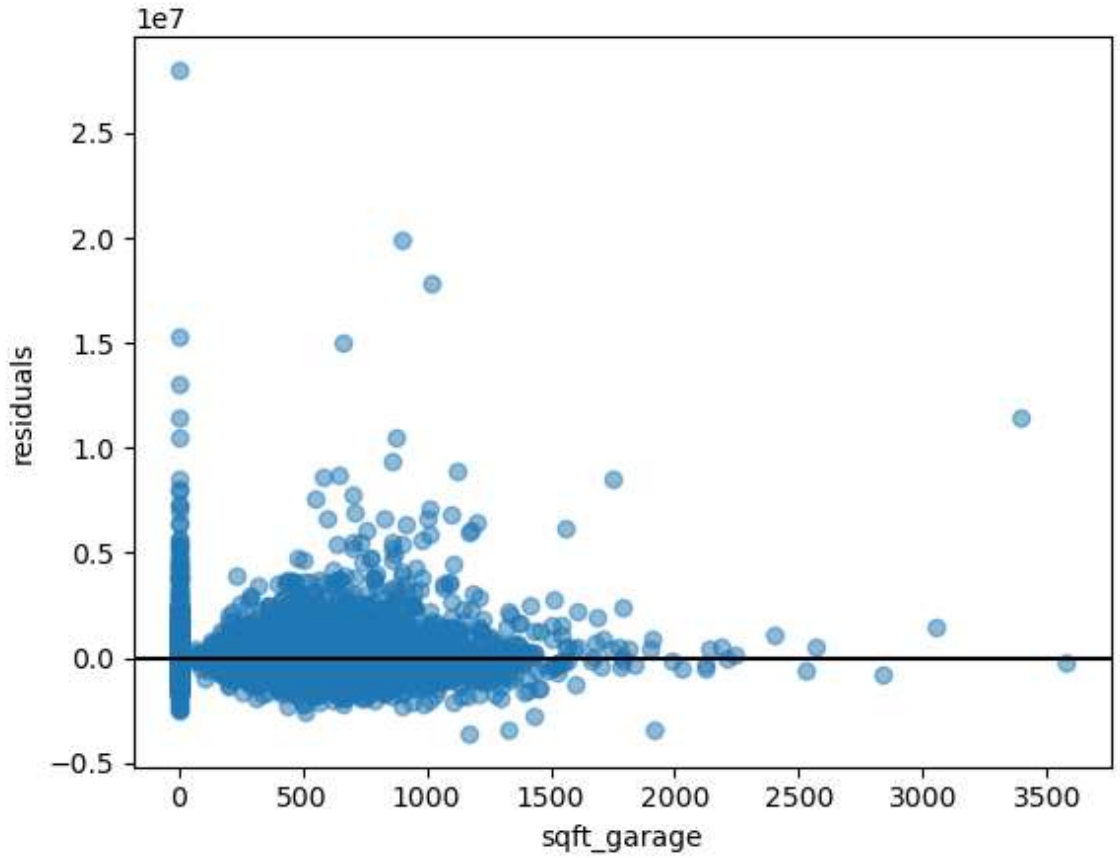
```
fig, ax = plt.subplots()

ax.scatter(df_X2["bedrooms"], model_2575_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("bedrooms")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, suggesting some heteroskedaticity.

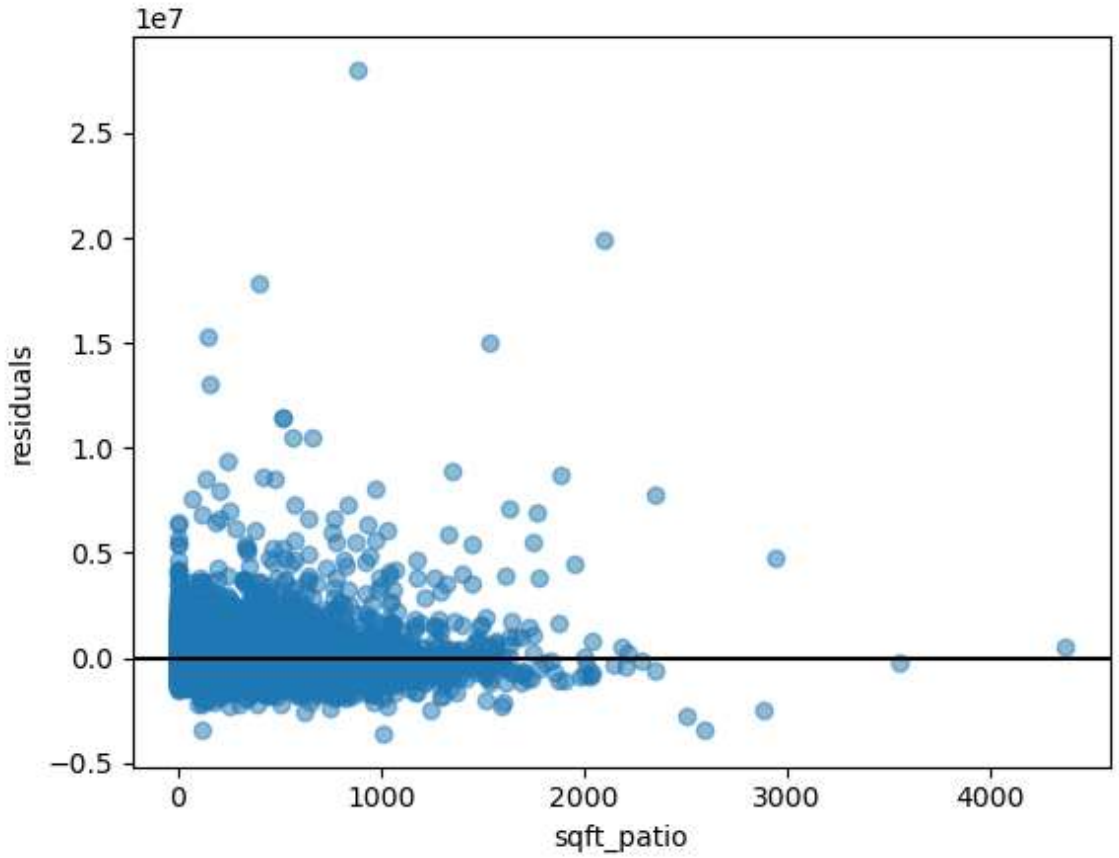
In [64]:

```
fig, ax = plt.subplots()

ax.scatter(df_X2["sqft_garage"], model_2575_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_garage")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, generally decreasing, suggesting some heteroskedaticity.

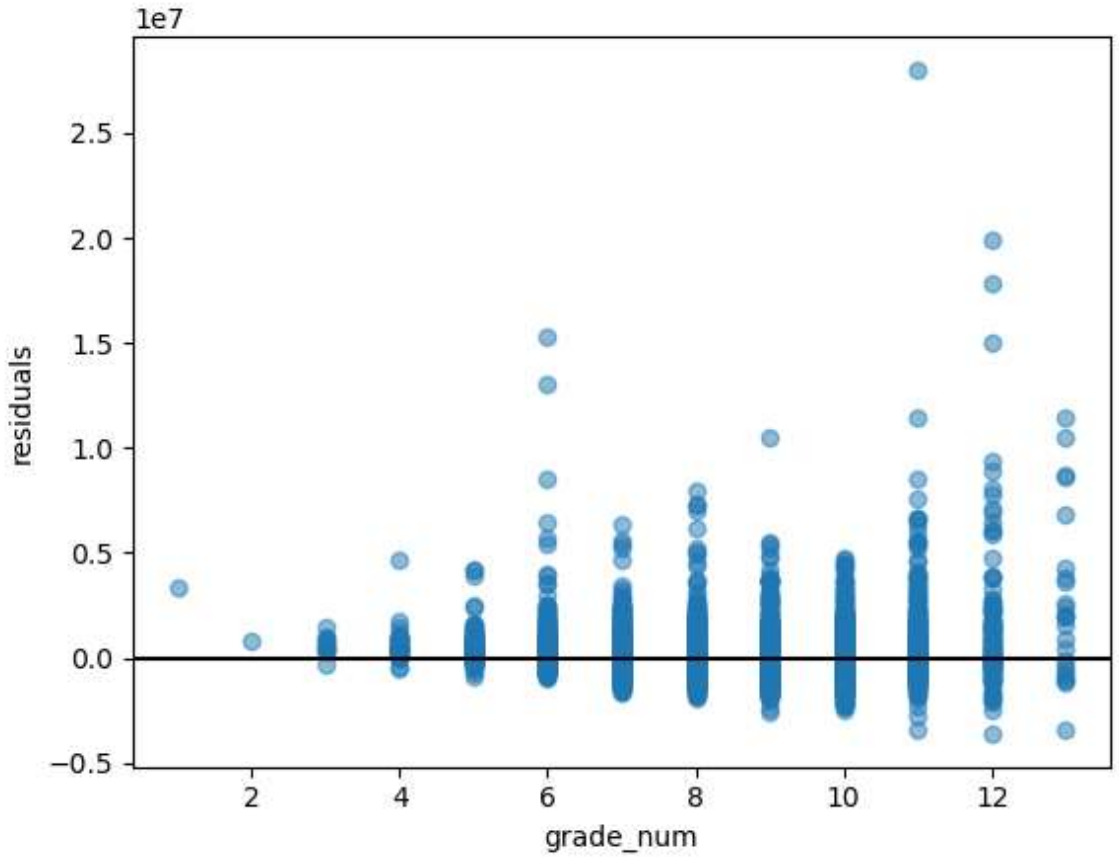
In [65]:
```
fig, ax = plt.subplots()

ax.scatter(df_X2["sqft_patio"], model_2575_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_patio")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, generally decreasing, suggesting some heteroskedaticity.

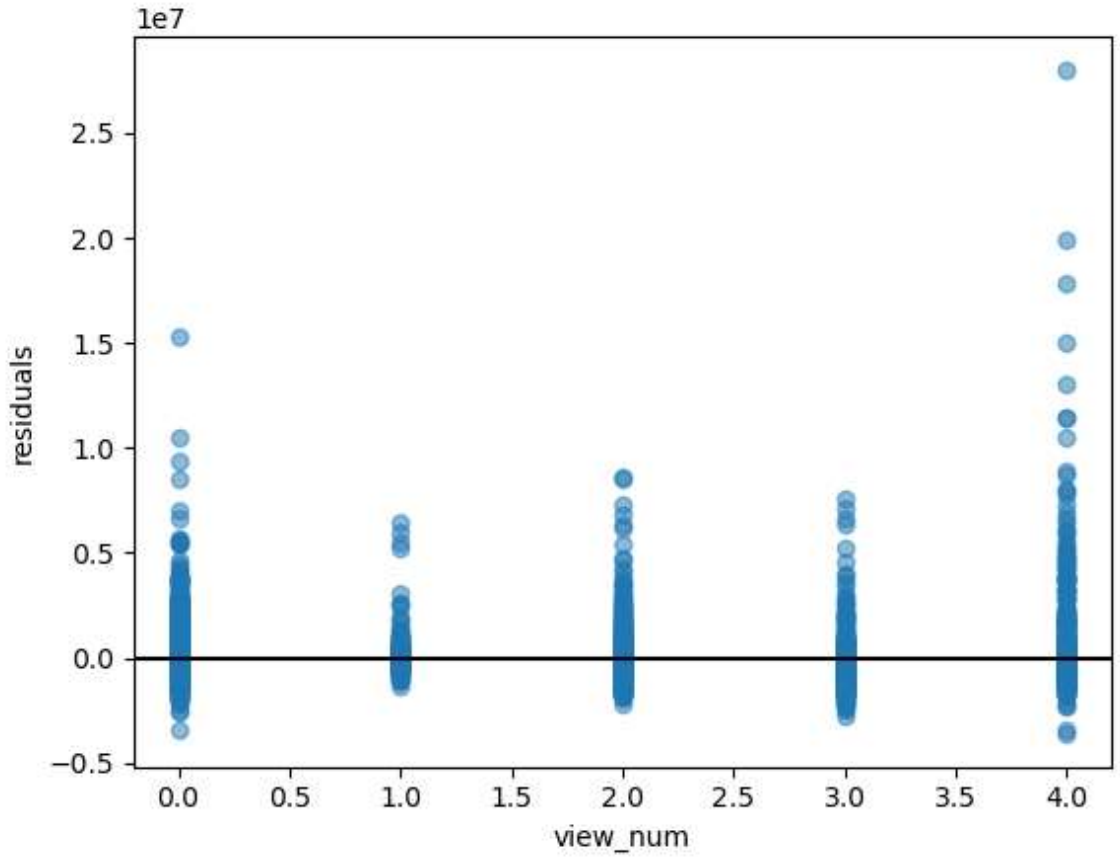
In [66]:

```
fig, ax = plt.subplots()

ax.scatter(df_X2["grade_num"], model_2575_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("grade_num")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, generally increasing, suggesting some heteroskedaticity.

In [67]:
```
fig, ax = plt.subplots()

ax.scatter(df_X2["view_num"], model_2575_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("view_num")
ax.set_ylabel("residuals");
```

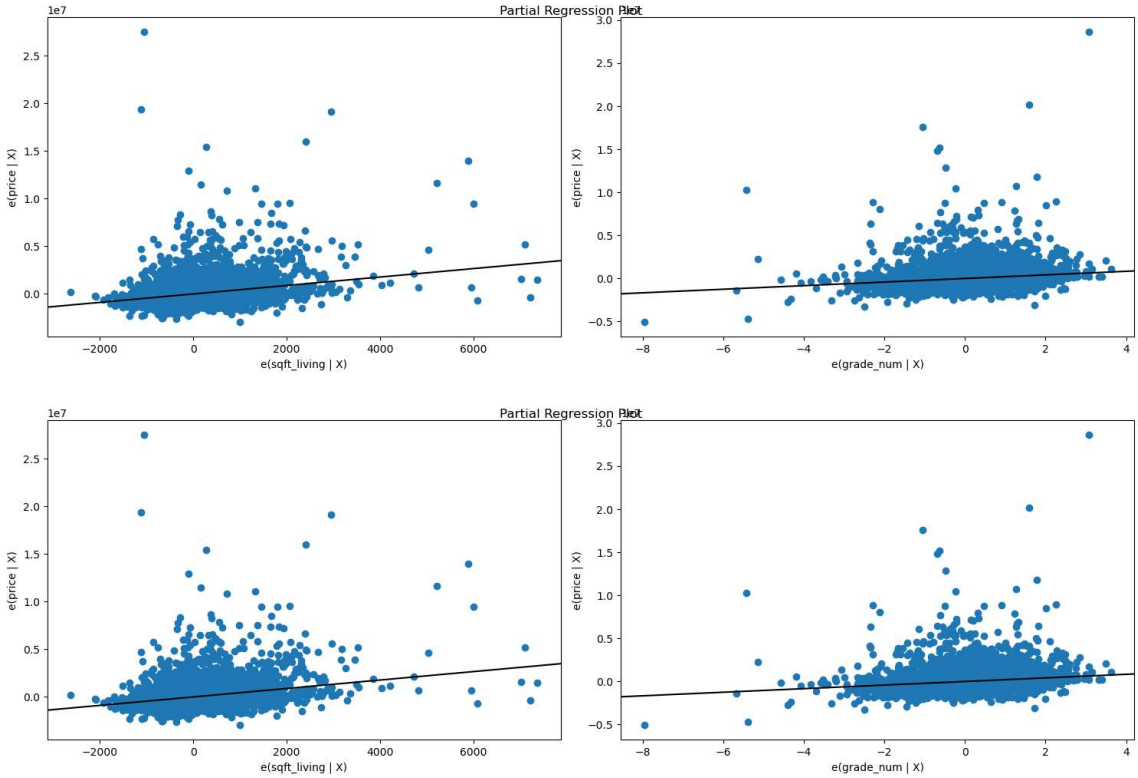


Above, there may be a pattern in the variance of residuals suggesting some heteroskedaticity.

In [68]:
```
fig = plt.figure(figsize=(15,5))
sm.graphics.plot_partregress_grid(model_2575_results, exog_idx=["sqft_livin
```

```
eval_env: 1
eval_env: 1
```

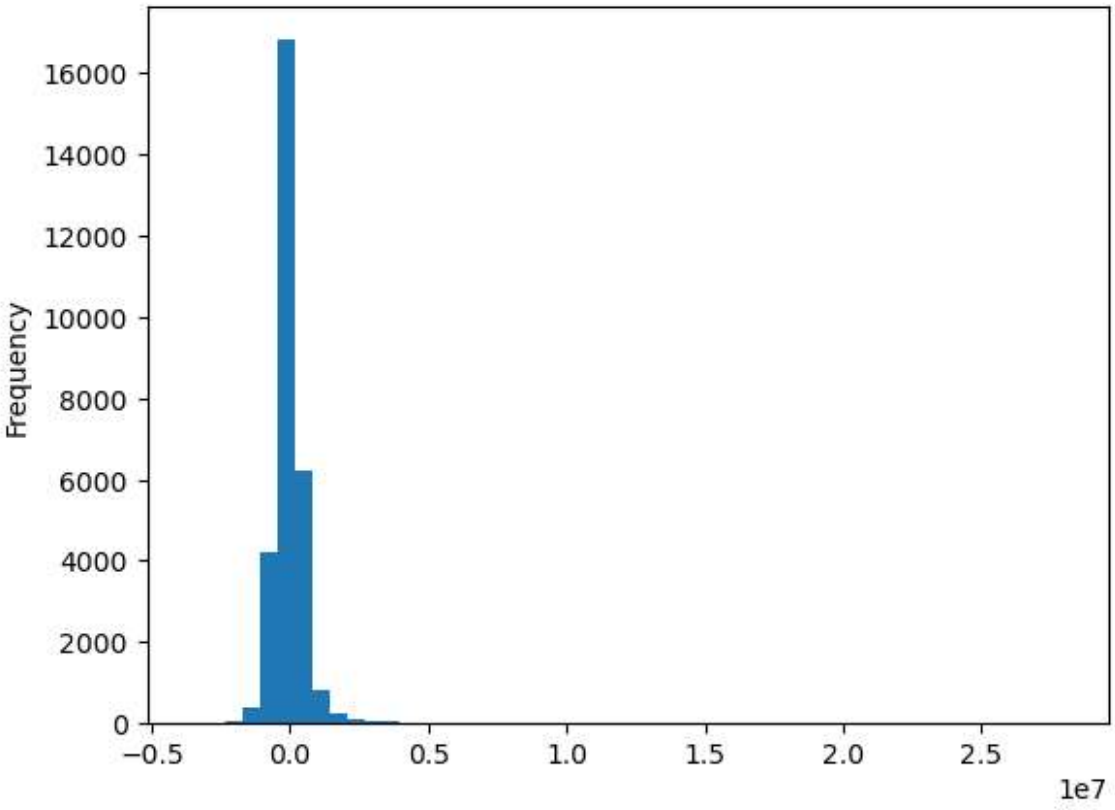
Out[68]:



Above, there is a pattern in the partial regression plot, more for sqft_living than grade_num, suggesting some non-linearity or heteroskedaticity.

In [69]:
```
model_2575_results.resid.plot(kind ='hist', bins = 50)
```

Out[69]: <AxesSubplot:ylabel='Frequency'>



Above, the residuals appear fairly normal, however the Jarque-Bera null hypothesis of normality is rejected.

Below, check for non-normality in variables based on non-linearity issues.

In [70]:
```
dataCut.price.plot(kind='hist', bins=1000)
```

Out[70]: <AxesSubplot:ylabel='Frequency'>

In [71]:
```
from scipy.stats import kurtosis, skew
```

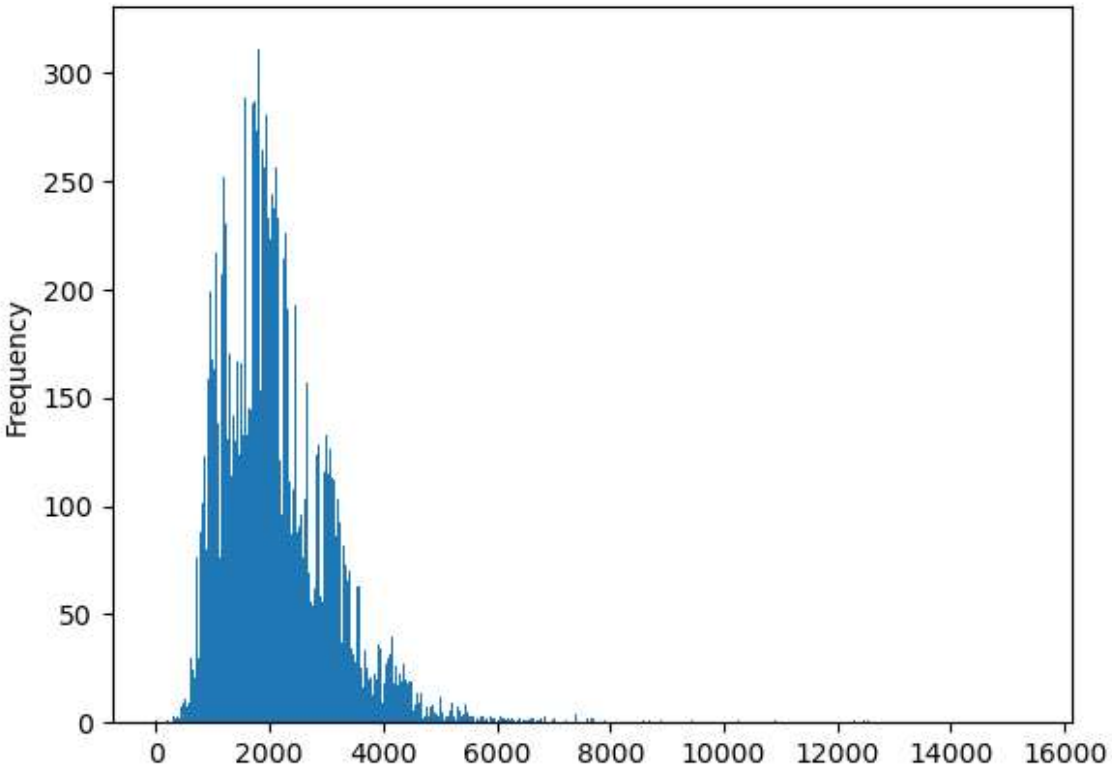
In [72]:
```
skew(dataCut['price'])
```

Out[72]: 6.6193687802407615

Above, price is somewhat skewed to the right and so not normally distributed.

In [73]:
```
dataCut.sqft_living.plot(kind='hist', bins=1000)
```

Out[73]: <AxesSubplot:ylabel='Frequency'>



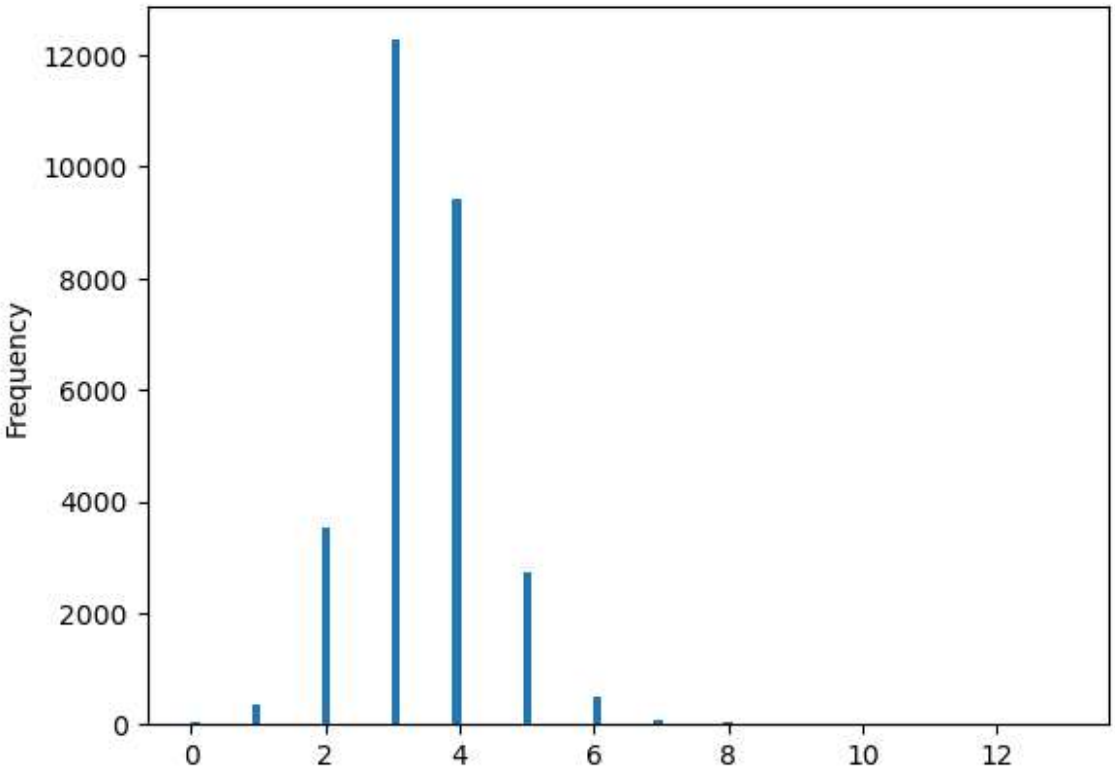
In [74]:
```
skew(dataCut['sqft_living'])
```

Out[74]: 1.6110785161352401

Above, sqft_living is somewhat skewed to the right and so not normally distributed.

In [75]:
```
dataCut.bedrooms.plot(kind='hist', bins=100)
```

Out[75]: <AxesSubplot:ylabel='Frequency'>



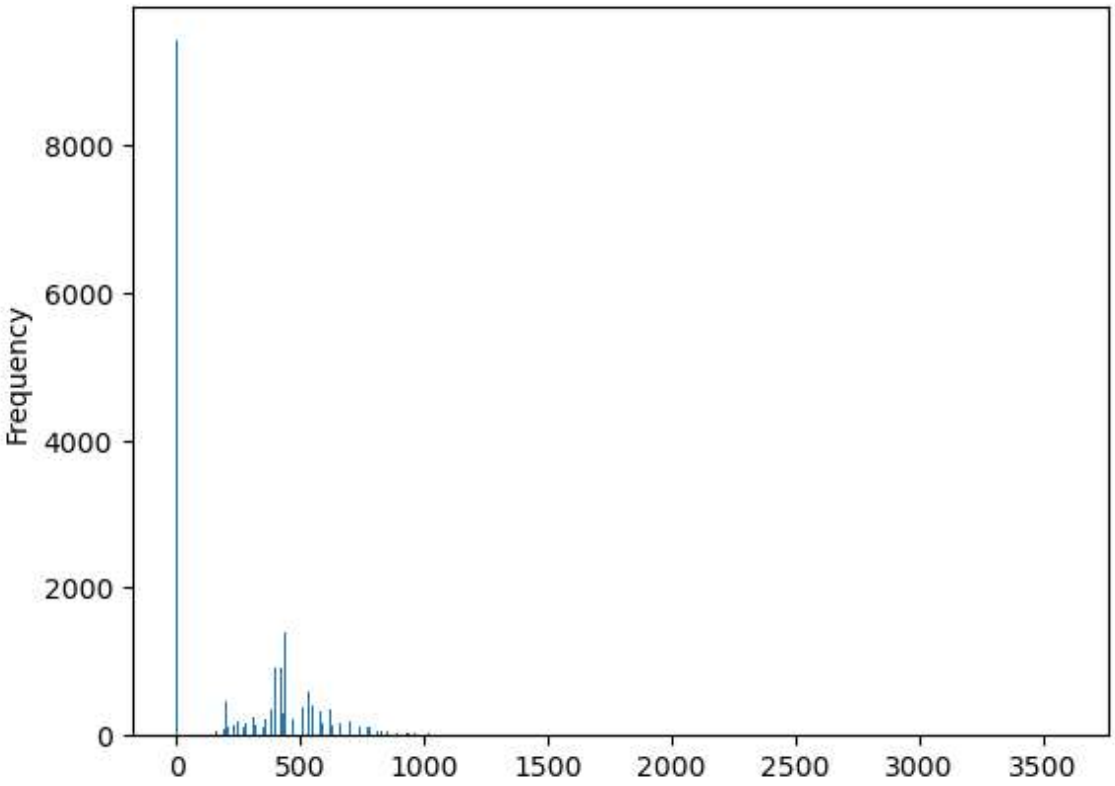
In [76]:
```
skew(dataCut['bedrooms'])
```

Out[76]: 0.5060726439825238

Above, bedrooms is less skewed to the right and so somewhat not normally distributed.

In [77]:
```
dataCut.sqft_garage.plot(kind='hist', bins=1000)
```

Out[77]: <AxesSubplot:ylabel='Frequency'>



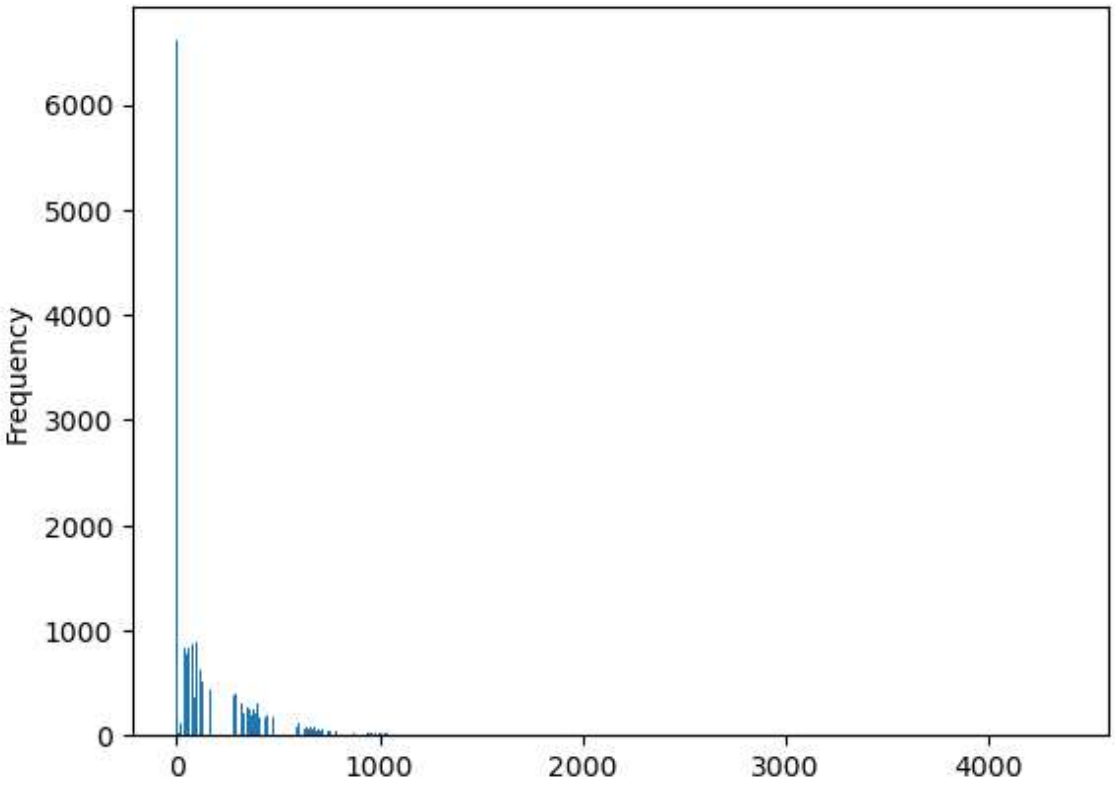
In [78]:
```
skew(dataCut['sqft_garage'])
```

Out[78]: 0.6418671177202289

Above, sqft_garage is also less skewed to the right and so somewhat not normally distributed.

In [79]:
```
dataCut.sqft_patio.plot(kind='hist', bins=1000)
```

Out[79]: <AxesSubplot:ylabel='Frequency'>



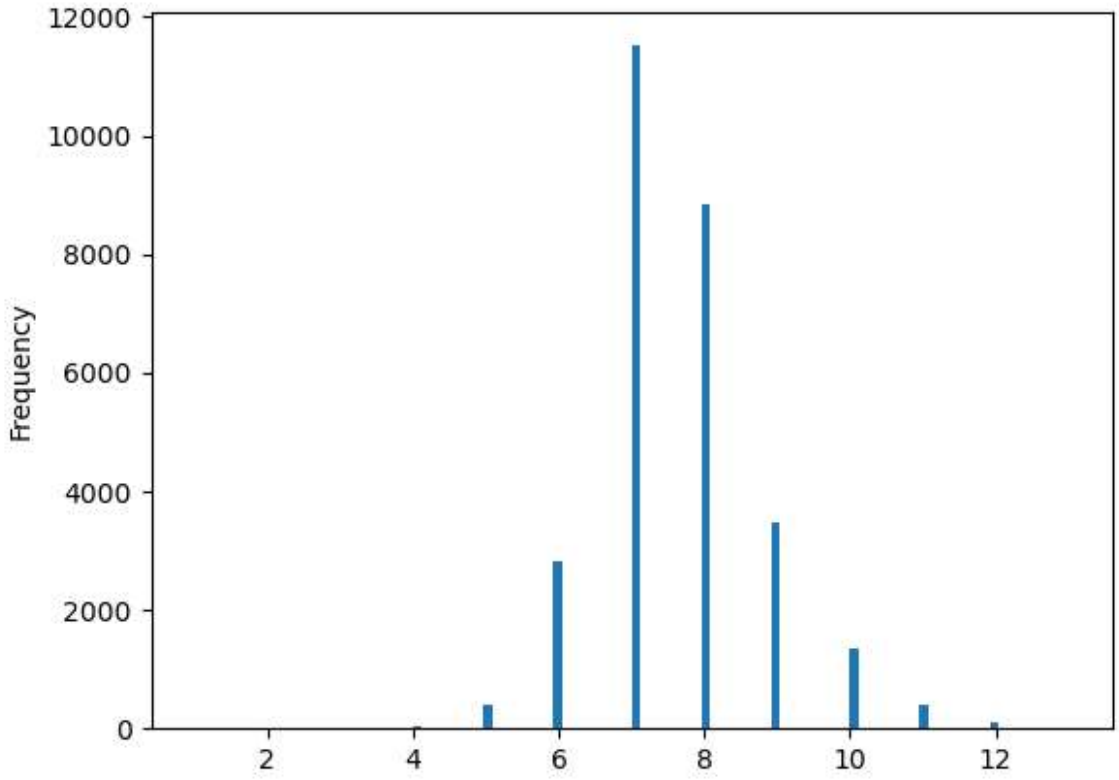
In [80]:
```
skew(dataCut['sqft_patio'])
```

Out[80]: 2.3589906964522265

Above, sqft_patio is somewhat skewed to the right and so not normally distributed.

In [81]:
```
dataCut.grade_num.plot(kind='hist', bins=100)
```

Out[81]: <AxesSubplot:ylabel='Frequency'>



In [82]:
```
skew(dataCut['grade_num'])
```

Out[82]: 0.669007645530176

Above, grade_num is also less skewed to the right and so somewhat not normally distributed.

In [83]:
```
dataCut.view_num.plot(kind='hist', bins=1000)
```

Out[83]: <AxesSubplot:ylabel='Frequency'>

In [84]:
```
skew(dataCut['view_num'])
```

Out[84]: 2.8579656493984533

Above, view_num is somewhat skewed to the right and so not normally distributed.

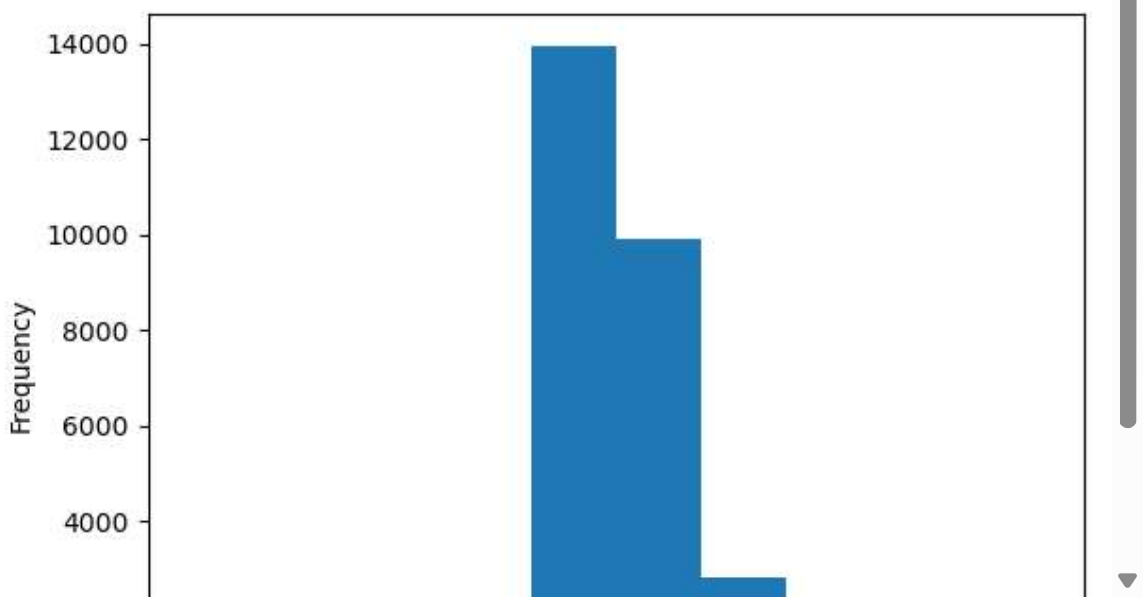
Start building log price/sqft living model based on non-linearity issues (in residual plots and partial regression plots) and non-normal issues in histograms.

In [85]:
```
y = df['price']
```

In [86]:
```
y_log = np.log(y)
```

In [87]:
```
y_log.plot(kind ='hist')
```

Out[87]: <AxesSubplot:ylabel='Frequency'>



Above, Y_log distribution is more normal than y distribution.

In [88]:
```
df_X2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sqft_living  29022 non-null  int64
 1   bedrooms     29022 non-null  int64
 2   sqft_garage  29022 non-null  int64
 3   sqft_patio   29022 non-null  int64
 4   grade_num    29022 non-null  int32
 5   view_num     29022 non-null  int64
dtypes: int32(1), int64(5)
memory usage: 2.4 MB
```

In [89]:
```
df_X2_log_living = df_X2.copy()
```

In [90]:
```
df_X2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sqft_living  29022 non-null  int64
 1   bedrooms     29022 non-null  int64
 2   sqft_garage  29022 non-null  int64
 3   sqft_patio   29022 non-null  int64
 4   grade_num    29022 non-null  int32
 5   view_num     29022 non-null  int64
dtypes: int32(1), int64(5)
memory usage: 2.4 MB
```

In [91]:
```
df_X2_log_living['sqft_living_log']= np.log(df_X2_log_living['sqft_living']
```

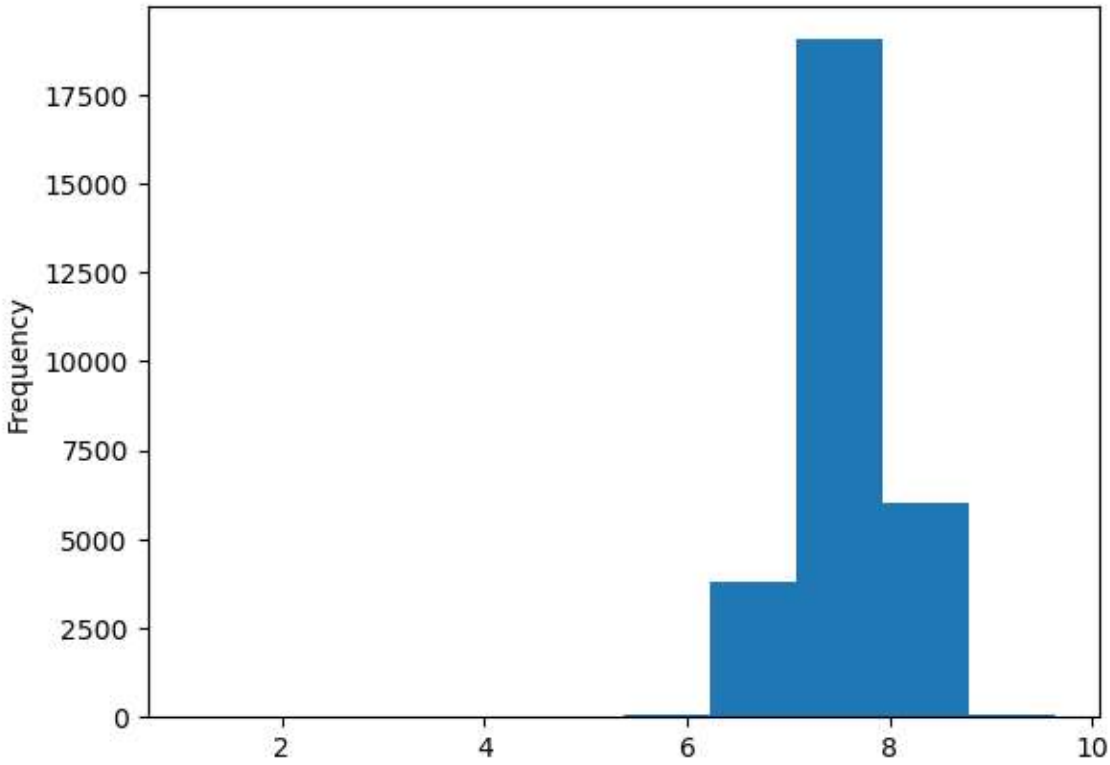
In [92]:
```
df_X2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29022 entries, 0 to 30154
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sqft_living  29022 non-null  int64
 1   bedrooms     29022 non-null  int64
 2   sqft_garage  29022 non-null  int64
 3   sqft_patio   29022 non-null  int64
 4   grade_num    29022 non-null  int32
 5   view_num     29022 non-null  int64
dtypes: int32(1), int64(5)
memory usage: 2.4 MB
```

In [93]:
```
df_X2_log_living= df_X2_log_living.drop("sqft_living", axis=1)
```

In [94]:
```
df_X2_log_living['sqft_living_log'].plot(kind='hist', bins=10)
```

Out[94]: <AxesSubplot:ylabel='Frequency'>



Above, sqft_living_log is more normally distributed than sqft_living distribution.

In [95]:
```
model_log_pr_liv = sm.OLS(y_log, sm.add_constant(df_X2_log_living))
model_log_pr_liv_results = model_log_pr_liv.fit()
```

In [96]:
```
model_log_pr_liv_results.summary()
```

Out[96]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.454
Model:	OLS	**Adj. R-squared:**	0.454
Method:	Least Squares	**F-statistic:**	4025.
Date:	Fri, 19 May 2023	**Prob (F-statistic):**	0.00
Time:	10:24:49	**Log-Likelihood:**	-16894.
No. Observations:	29022	**AIC:**	3.380e+04
Df Residuals:	29015	**BIC:**	3.386e+04
Df Model:	6		
Covariance Type:	nonrobust		

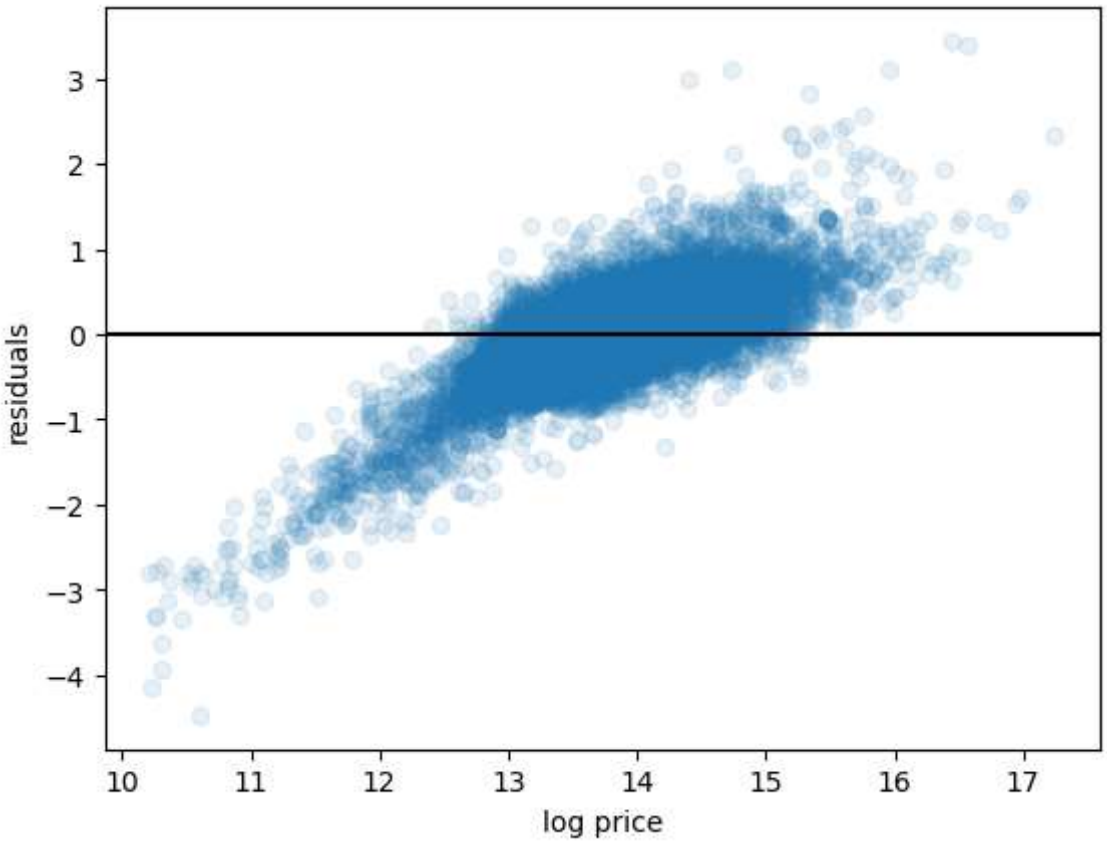
	coef	std err	t	P>|t|	[0.025	0.975]
const	9.1121	0.063	144.504	0.000	8.988	9.236
bedrooms	-0.0046	0.004	-1.282	0.200	-0.012	0.002
sqft_garage	-0.0002	1.07e-05	-17.434	0.000	-0.000	-0.000
sqft_patio	0.0001	1.14e-05	8.891	0.000	7.93e-05	0.000
grade_num	0.2077	0.003	61.647	0.000	0.201	0.214
view_num	0.0833	0.003	26.768	0.000	0.077	0.089
sqft_living_log	0.4060	0.011	36.986	0.000	0.384	0.427

Omnibus:	6288.015	**Durbin-Watson:**	1.992
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	62913.505
Skew:	-0.757	**Prob(JB):**	0.00
Kurtosis:	10.052	**Cond. No.**	1.26e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.26e+04. This might indicate that there are strong multicollinearity or other numerical problems.

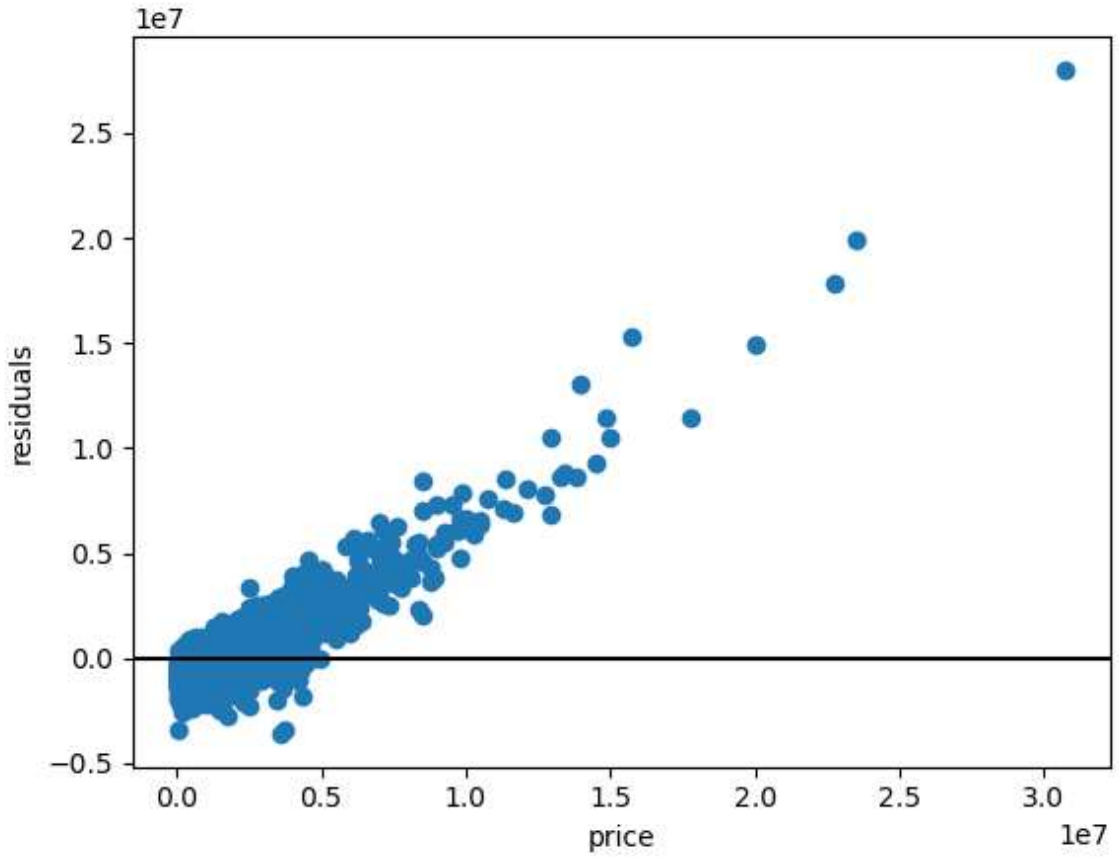
In [97]:

```
fig, ax = plt.subplots()

ax.scatter(y_log, model_log_pr_liv_results.resid, alpha=.1)
ax.axhline(y=0, color="black")
ax.set_xlabel("log price")
ax.set_ylabel("residuals");
```



In [98]:

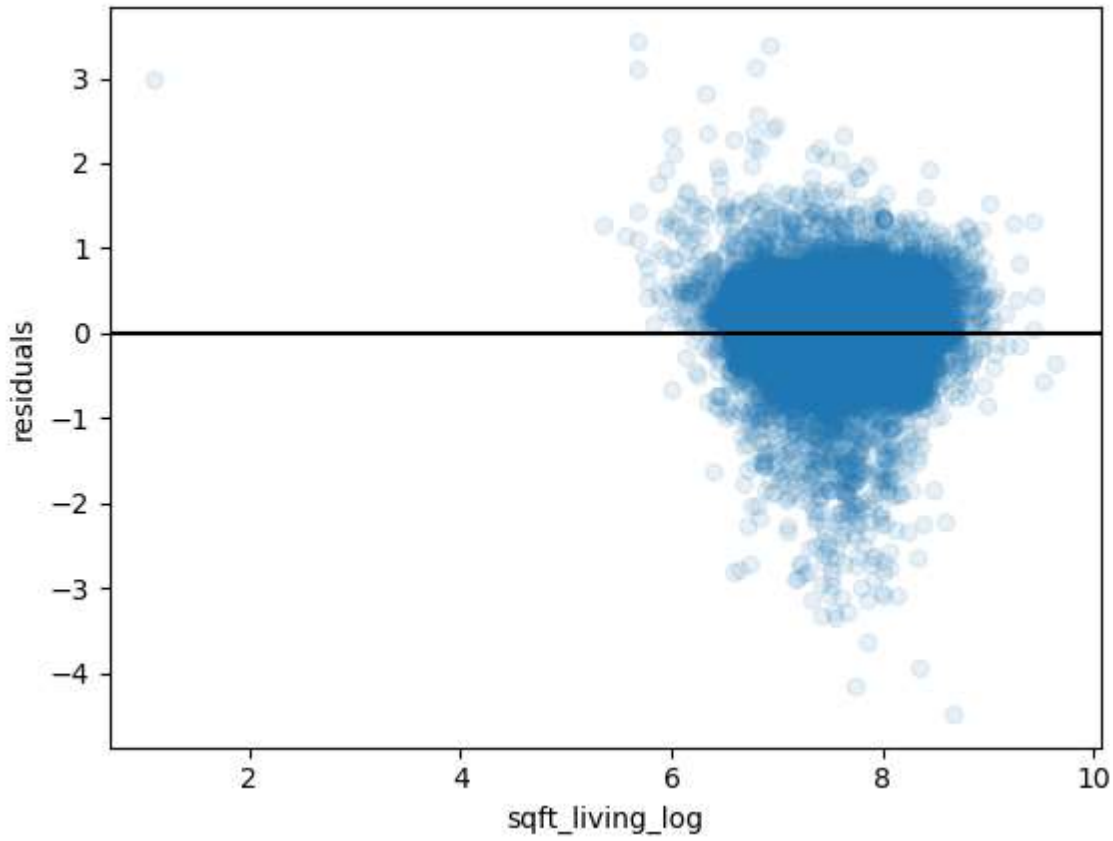
```
fig, ax = plt.subplots()

ax.scatter(df["price"], model_2575_results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("price")
ax.set_ylabel("residuals");
```



Above, modest improvement in randomness of residuals, (from top to bottom chart). There still are non-linearity issues, and maybe heterskedaticity.

In [99]:

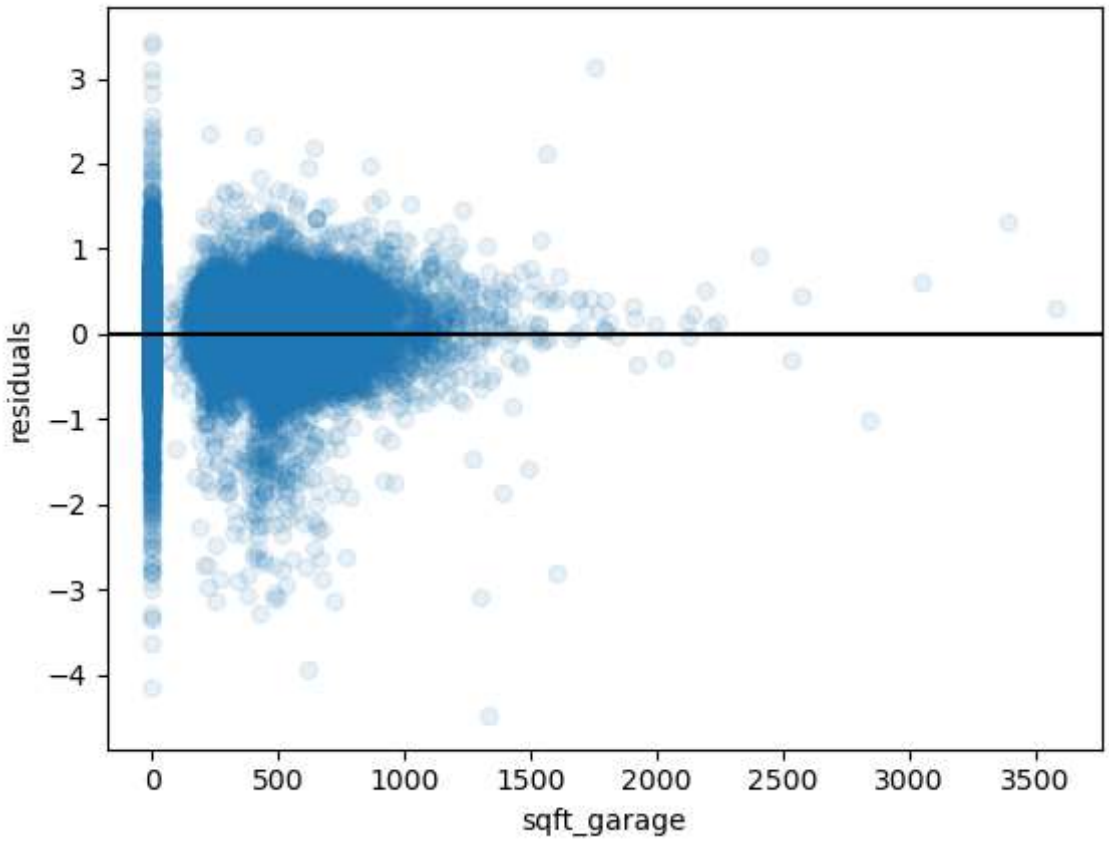
```
fig, ax = plt.subplots()

ax.scatter(df_X2_log_living['sqft_living_log'], model_log_pr_liv_results.re
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_living_log")
ax.set_ylabel("residuals");
```



Above, less heteroskedaticity than unlogged plot but residuals are somewhat larger in the middle suggesting some heteroskedaticity.

In [100]:

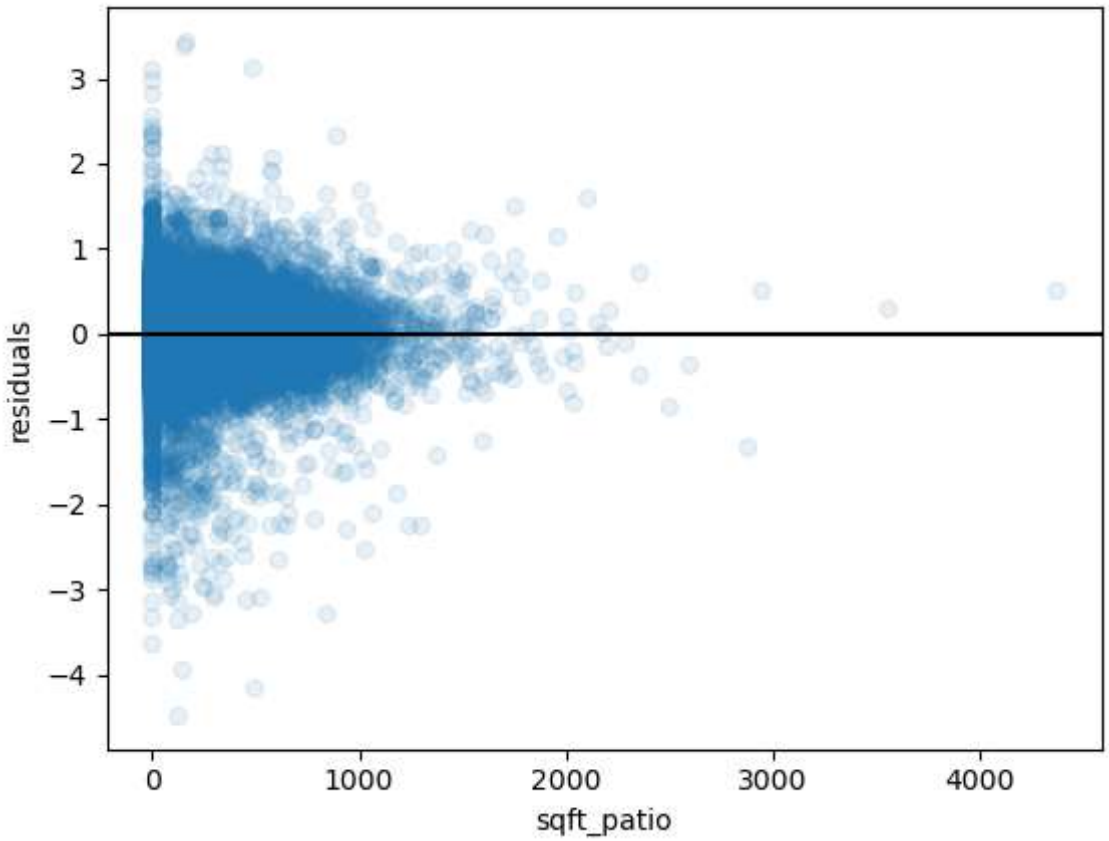
```
fig, ax = plt.subplots()

ax.scatter(df_X2_log_living['sqft_garage'], model_log_pr_liv_results.resid,
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_garage")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, decreasing, suggesting some heteroskedaticity.

In [101]:

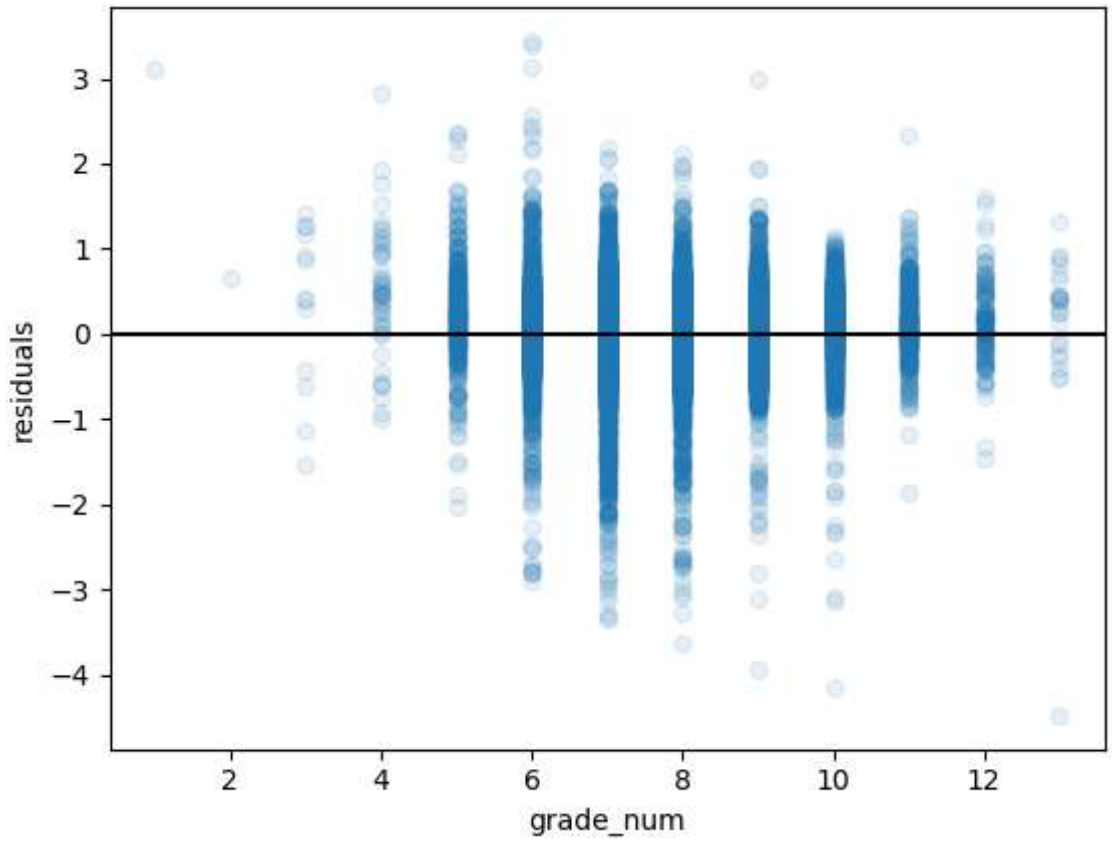
```
fig, ax = plt.subplots()

ax.scatter(df_X2_log_living['sqft_patio'], model_log_pr_liv_results.resid,
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_patio")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, decreasing, suggesting some heteroskedaticity.

In [102]:

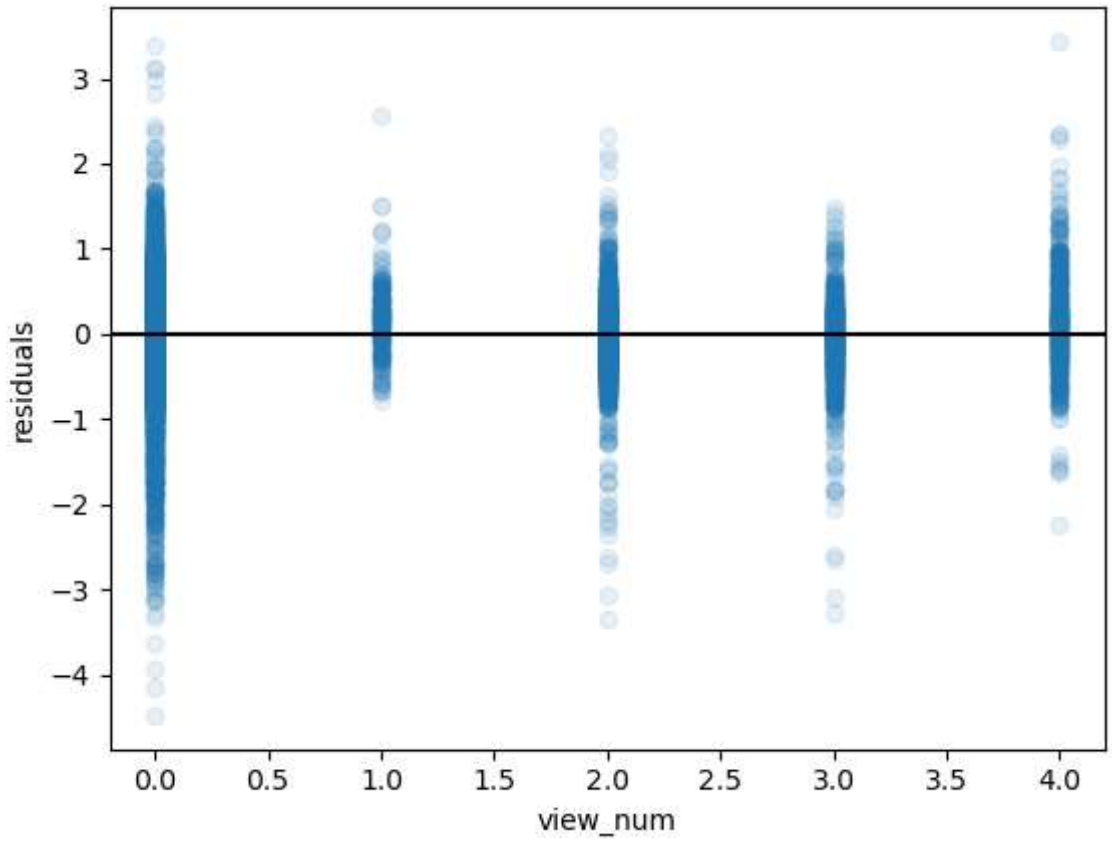
```
fig, ax = plt.subplots()

ax.scatter(df_X2_log_living['grade_num'], model_log_pr_liv_results.resid, a
ax.axhline(y=0, color="black")
ax.set_xlabel("grade_num")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals suggesting some heteroskedaticity.

In [103]:

```
fig, ax = plt.subplots()

ax.scatter(df_X2_log_living['view_num'], model_log_pr_liv_results.resid, al
ax.axhline(y=0, color="black")
ax.set_xlabel("view_num")
ax.set_ylabel("residuals");
```

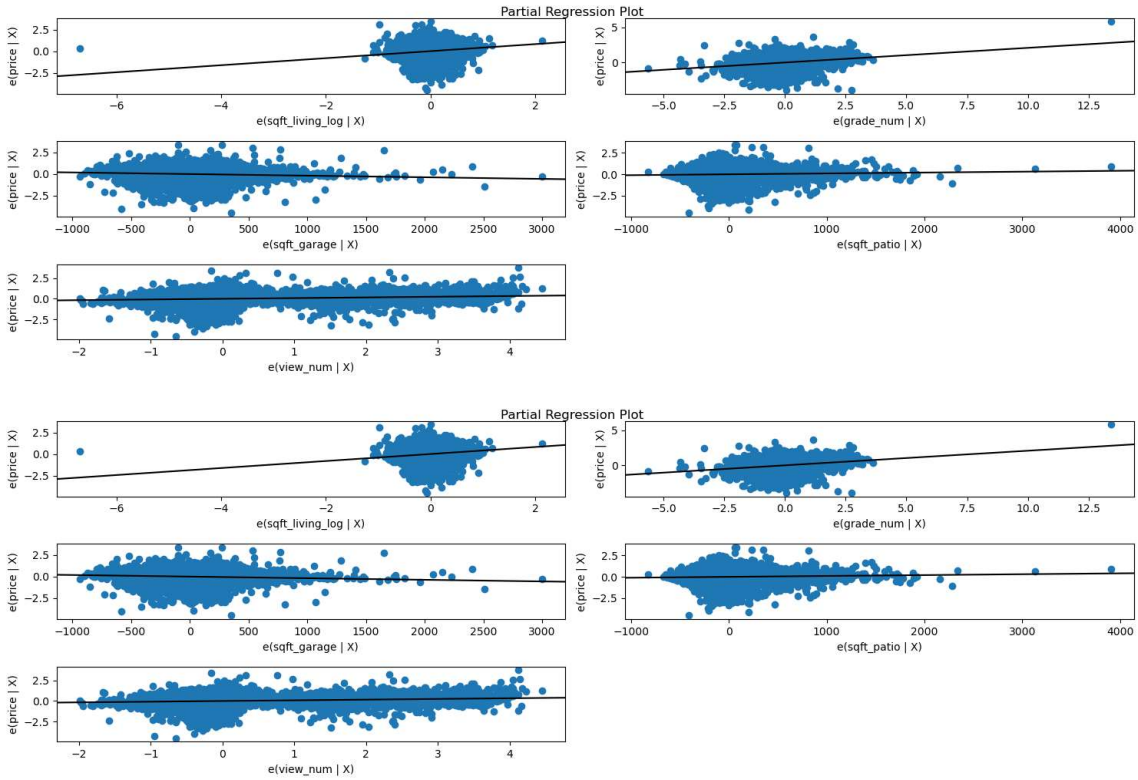


Above, variance of residuals are somewhat consistent, suggesting homoskedaticity.

In [104]:
```
fig = plt.figure(figsize=(15,5))
sm.graphics.plot_partregress_grid(model_log_pr_liv_results, exog_idx=["sqft
```

```
eval_env: 1
eval_env: 1
eval_env: 1
eval_env: 1
eval_env: 1
```

Out[104]:



Above, there are some non-linear patterns suggesting some non-linearity or heteroskedaticity in patiosqft and garagesqft.

Analyze potential outliers of 0 (and heteroskedaticity, inflamed by 0 values) in sqftgarage and patio (and bedrooms) and looking 0 values for data entry errors:

In [105]:
```
dataCut.sqft_garage.value_counts().sort_index()
```

Out[105]:
```
0       9411
1          1
40         1
70         2
80         5
          ...
2570       1
2840       1
3050       1
3390       1
3580       1
Name: sqft_garage, Length: 403, dtype: int64
```

In [106]:
```
dataCut.sqft_patio.value_counts().sort_index()
```

Out[106]:
```
0       6605
8          1
10        20
12         1
14         1
         ...
2590       1
2880       1
2940       1
3550       1
4370       1
Name: sqft_patio, Length: 521, dtype: int64
```

In [107]:
```
dataCut.bedrooms.value_counts().sort_index()
```

Out[107]:
```
0        41
1       375
2      3535
3     12272
4      9429
5      2741
6       493
7        80
8        37
9        14
10        3
11        1
13        1
Name: bedrooms, dtype: int64
```

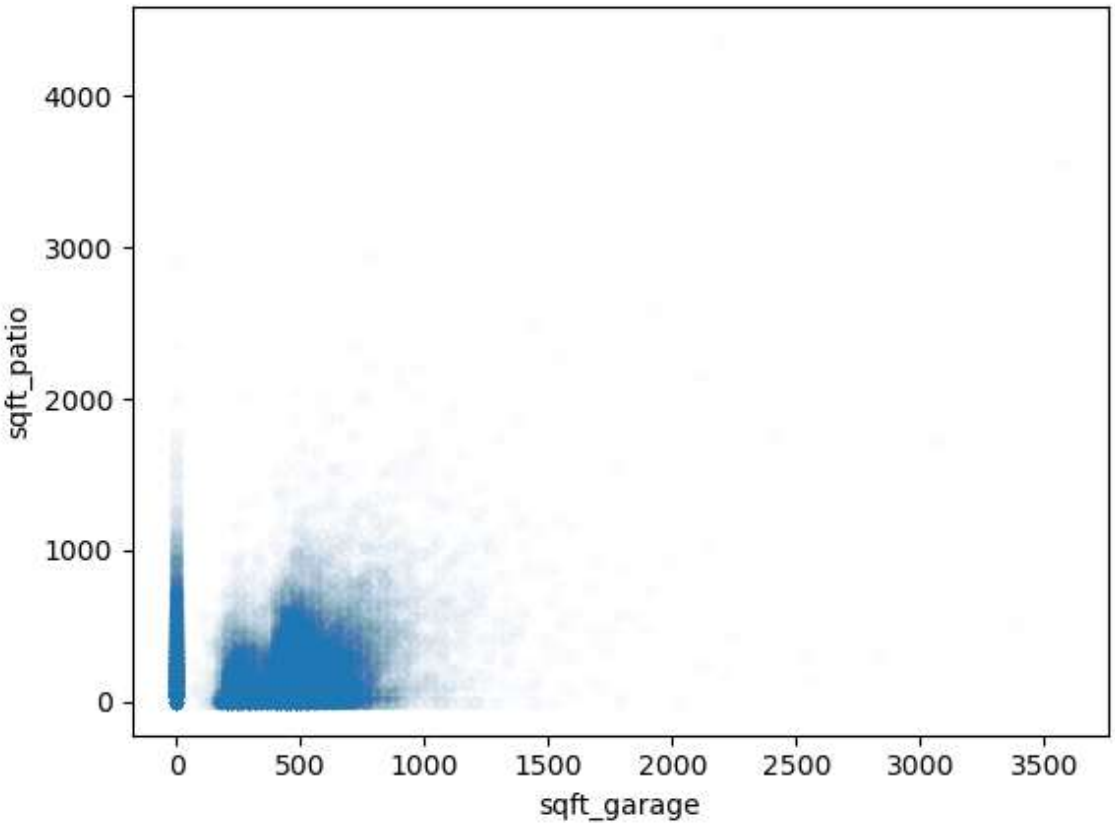
In [108]:
```
dataCut.sqft_living.mean()
```

Out[108]:
```
2128.358934601337
```

0 bedrooms dont make sense because houses have significant sqfootage (larger than studios). Zero values in garage and patio make sense as many houses don't have these, but they still may be outliers.

In [109]:
```
df_X2_log_living.plot(kind = 'scatter', x = 'sqft_garage', y='sqft_patio',
```

Out[109]: <AxesSubplot:xlabel='sqft_garage', ylabel='sqft_patio'>

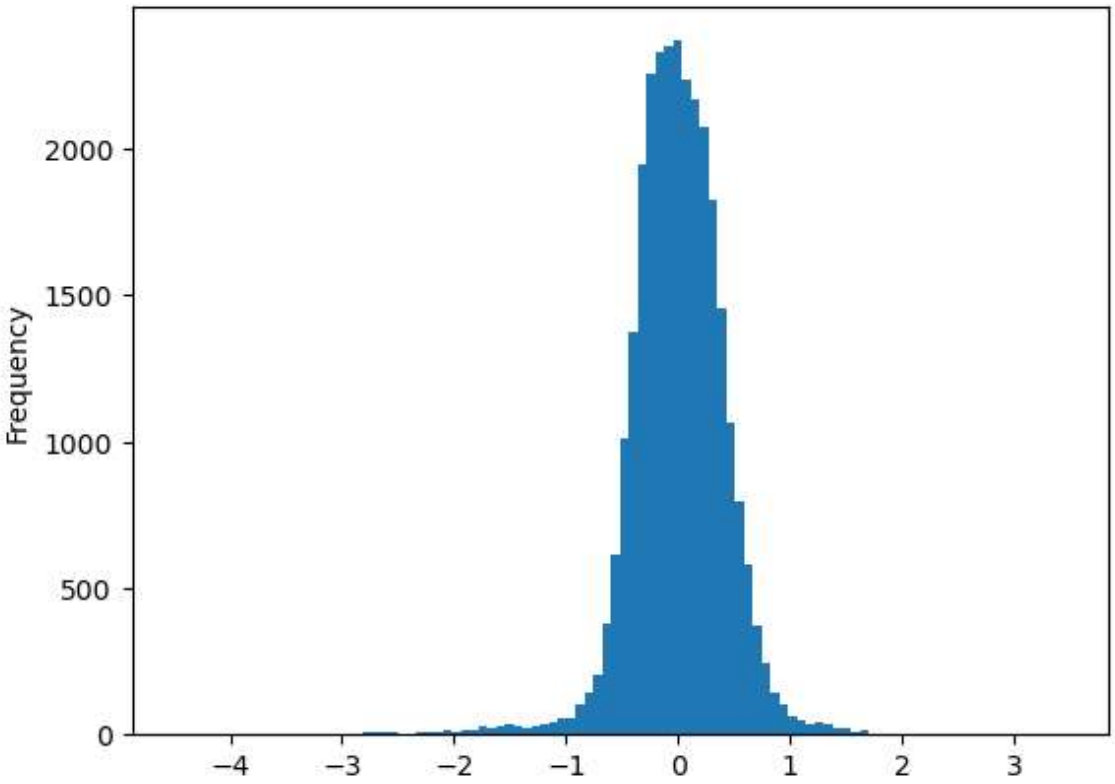


Above, fairly low crossover between 0's , thus dropping them would eliminate large part of dataset.

Check normality of residuals:

In [110]:
```
model_log_pr_liv_results.resid.plot(kind ='hist', bins = 100)
```

Out[110]: <AxesSubplot:ylabel='Frequency'>



Above, residuals appear fairly normal for log price&sqftliving model and are a large improvement from non-log model. However, the Jarque-Bera null hypothesis of normality is rejected. Skewness is slightly negative and improved from high positives in previous model. Kurtosis is high.

New model with more log tranformed variables (sqft_patio,sqft_garage) and 0 bedroom houses removed and remove bedrooms as predictor. Also create dummy var's for whether house has garage or patio. The aim is to improve on non-linearity issue, heteroskedaticity issues and non-normality issues and to improve rsquared.

initialize X3 df and remove 0 bedroom houses.

In [111]:
```
df_X3_logs = df_X2_log_living
```

In [112]:
```
df_X3_logs['y_log']=y_log
```

In [113]:
```
df_X3_logs = df_X3_logs.drop(df_X3_logs.loc[df_X3_logs['bedrooms']==0].inde
```

In [114]:
```
df_X3_logs.describe()
```

Out[114]:

	bedrooms	sqft_garage	sqft_patio	grade_num	view_num	sqft_living_log
count	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000
mean	3.439357	336.211587	216.121252	7.635382	0.300576	7.567615
std	0.970416	285.854606	246.563226	1.152907	0.861905	0.441677
min	1.000000	0.000000	0.000000	1.000000	0.000000	1.098612
25%	3.000000	0.000000	40.000000	7.000000	0.000000	7.272398
50%	3.000000	400.000000	140.000000	7.000000	0.000000	7.570443
75%	4.000000	510.000000	310.000000	8.000000	0.000000	7.874739
max	13.000000	3580.000000	4370.000000	13.000000	4.000000	9.639522

In [115]:
```
df_X3_logs= df_X3_logs.drop("bedrooms", axis=1)
```

Create two dummy variables for whether house has garage and patio.

In [116]:
```
garage_yes = df_X3_logs['sqft_garage'].map(lambda x: 0 if x==0 else 1)
```

In [117]:
```
patio_yes = df_X3_logs['sqft_patio'].map(lambda x: 0 if x==0 else 1)
```

In [118]:
```
df_X3_logs['garage_yes']=garage_yes
```

In [119]:
```
df_X3_logs['patio_yes']=patio_yes
```

Substitute .1 in for 0 values in sqft gar & patio in order to log transform. .1 is equivalent to 0 as both mean essentially no garage or patio.

In [120]:
```
df_X3_logs['sqft_garage'] = df_X3_logs['sqft_garage'].map(lambda x: .1 if x
```

In [121]:
```
df_X3_logs['sqft_patio'] = df_X3_logs['sqft_patio'].map(lambda x: .1 if x==
```

In [122]:
```
#df_X3_logs.describe()
```

In [123]:
```
df_X3_logs['sqft_garage_log']= np.log(df_X3_logs['sqft_garage'])

df_X3_logs= df_X3_logs.drop("sqft_garage", axis=1)
```

In [124]:
```
df_X3_logs['sqft_patio_log']= np.log(df_X3_logs['sqft_patio'])

df_X3_logs= df_X3_logs.drop("sqft_patio", axis=1)
```

In [125]:
```
df_X3_logs.describe()
```

Out[125]:

	grade_num	view_num	sqft_living_log	y_log	garage_yes	patio_yes
count	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000	28981.000000
mean	7.635382	0.300576	7.567615	13.738000	0.676512	0.773162
std	1.152907	0.861905	0.441677	0.586109	0.467815	0.418794
min	1.000000	0.000000	1.098612	10.216837	0.000000	0.000000
25%	7.000000	0.000000	7.272398	13.377006	0.000000	1.000000
50%	7.000000	0.000000	7.570443	13.670485	1.000000	1.000000
75%	8.000000	0.000000	7.874739	14.086682	1.000000	1.000000
max	13.000000	4.000000	9.639522	17.241401	1.000000	1.000000

In [126]:
```
df_X3_logs.corr()
```

Out[126]:

	grade_num	view_num	sqft_living_log	y_log	garage_yes	patio_yes	sq
grade_num	1.000000	0.209331	0.720184	0.621400	0.353333	0.254709	
view_num	0.209331	1.000000	0.208744	0.278210	-0.035780	0.103912	
sqft_living_log	0.720184	0.208744	1.000000	0.594424	0.335307	0.309592	
y_log	0.621400	0.278210	0.594424	1.000000	0.148768	0.198133	
garage_yes	0.353333	-0.035780	0.335307	0.148768	1.000000	0.118955	
patio_yes	0.254709	0.103912	0.309592	0.198133	0.118955	1.000000	
sqft_garage_log	0.382501	-0.029911	0.367679	0.169316	0.996556	0.131214	
sqft_patio_log	0.301046	0.148537	0.359002	0.244905	0.118802	0.972272	

Above, high correlations between dummies and sqft_garage_log sqft_patio_log may suggest multicollinearity/confounding variable issues, but domain knowledge says that both the existense of garage/patio and the size should have seperate effects on price.

seperating out dep. var.:

In [127]:
```
y_log_X3 = df_X3_logs['y_log']
```

In [128]:
```
df_X3_logs= df_X3_logs.drop("y_log", axis=1)
```

In [129]:
```
model_X3_logs = sm.OLS(y_log_X3, sm.add_constant(df_X3_logs))
model_X3_logs_results = model_X3_logs.fit()
```

In [130]:
```
model_X3_logs_results.summary()
```

Out[130]:

OLS Regression Results

Dep. Variable:	y_log	**R-squared:**	0.458
Model:	OLS	**Adj. R-squared:**	0.458
Method:	Least Squares	**F-statistic:**	3504.
Date:	Fri, 19 May 2023	**Prob (F-statistic):**	0.00
Time:	10:24:52	**Log-Likelihood:**	-16752.
No. Observations:	28981	**AIC:**	3.352e+04
Df Residuals:	28973	**BIC:**	3.359e+04
Df Model:	7		
Covariance Type:	nonrobust		

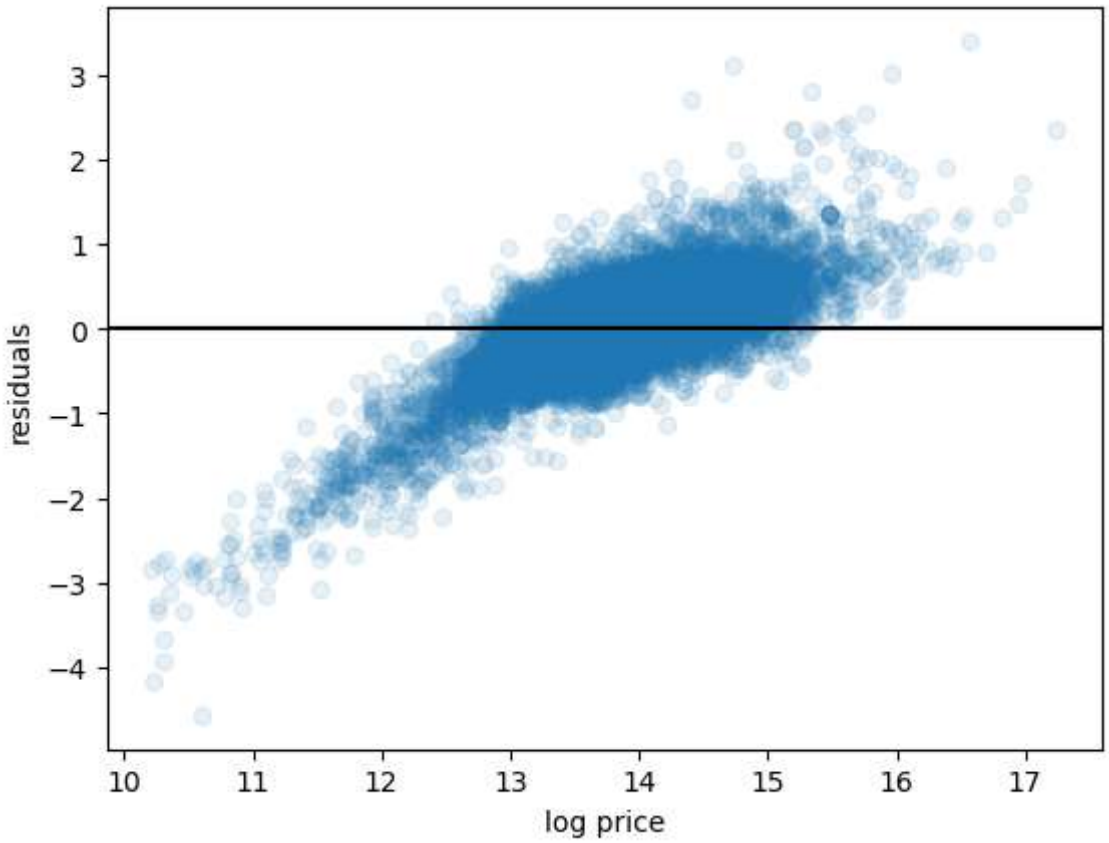
	coef	std err	t	P>|t|	[0.025	0.975]
const	8.9733	0.065	137.825	0.000	8.846	9.101
grade_num	0.2094	0.003	63.784	0.000	0.203	0.216
view_num	0.0806	0.003	26.092	0.000	0.075	0.087
sqft_living_log	0.4136	0.009	46.881	0.000	0.396	0.431
garage_yes	0.4536	0.072	6.261	0.000	0.312	0.596
patio_yes	-0.2099	0.027	-7.835	0.000	-0.262	-0.157
sqft_garage_log	-0.0683	0.009	-7.859	0.000	-0.085	-0.051
sqft_patio_log	0.0280	0.004	7.979	0.000	0.021	0.035

Omnibus:	6507.496	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	63411.590
Skew:	-0.803	**Prob(JB):**	0.00
Kurtosis:	10.067	**Cond. No.**	415.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [131]:

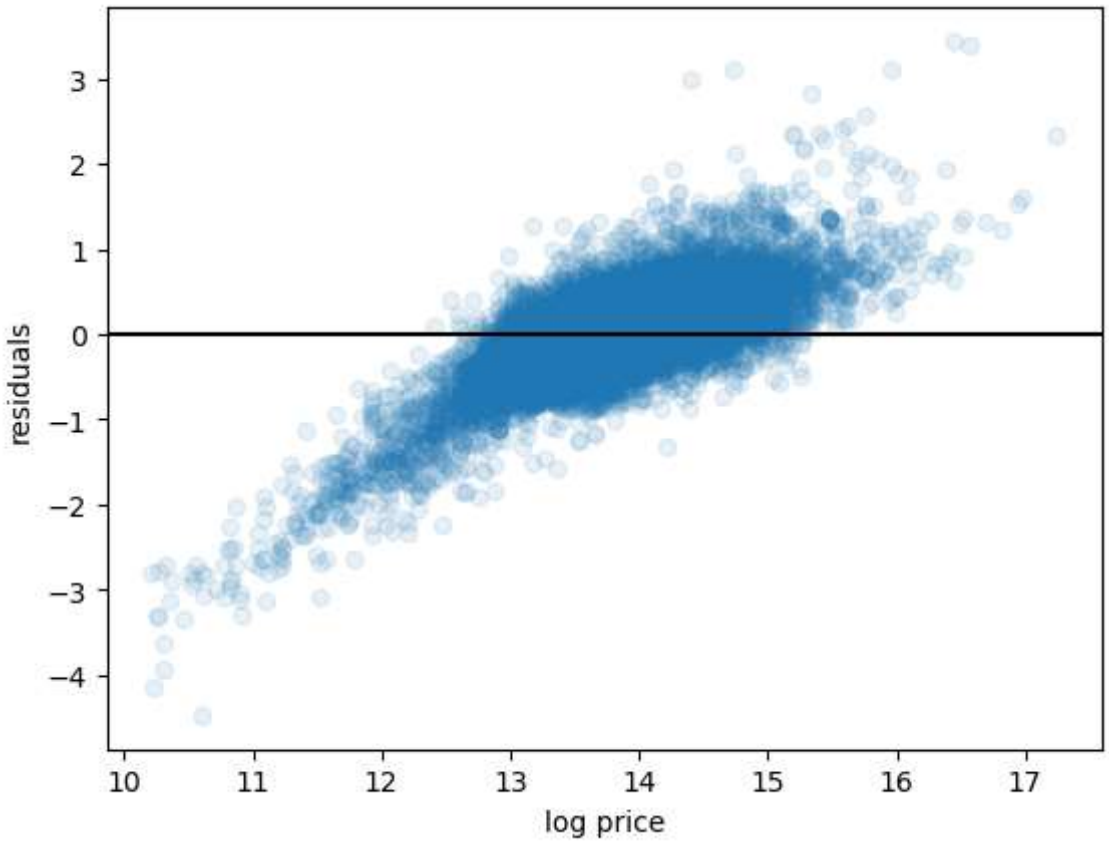
```
fig, ax = plt.subplots()

ax.scatter(y_log_X3, model_X3_logs_results.resid, alpha=.1)
ax.axhline(y=0, color="black")
ax.set_xlabel("log price")
ax.set_ylabel("residuals");
```



Above, previous residual scatter

In [132]:

```
fig, ax = plt.subplots()

ax.scatter(y_log, model_log_pr_liv_results.resid, alpha=.1)
ax.axhline(y=0, color="black")
ax.set_xlabel("log price")
ax.set_ylabel("residuals");
```

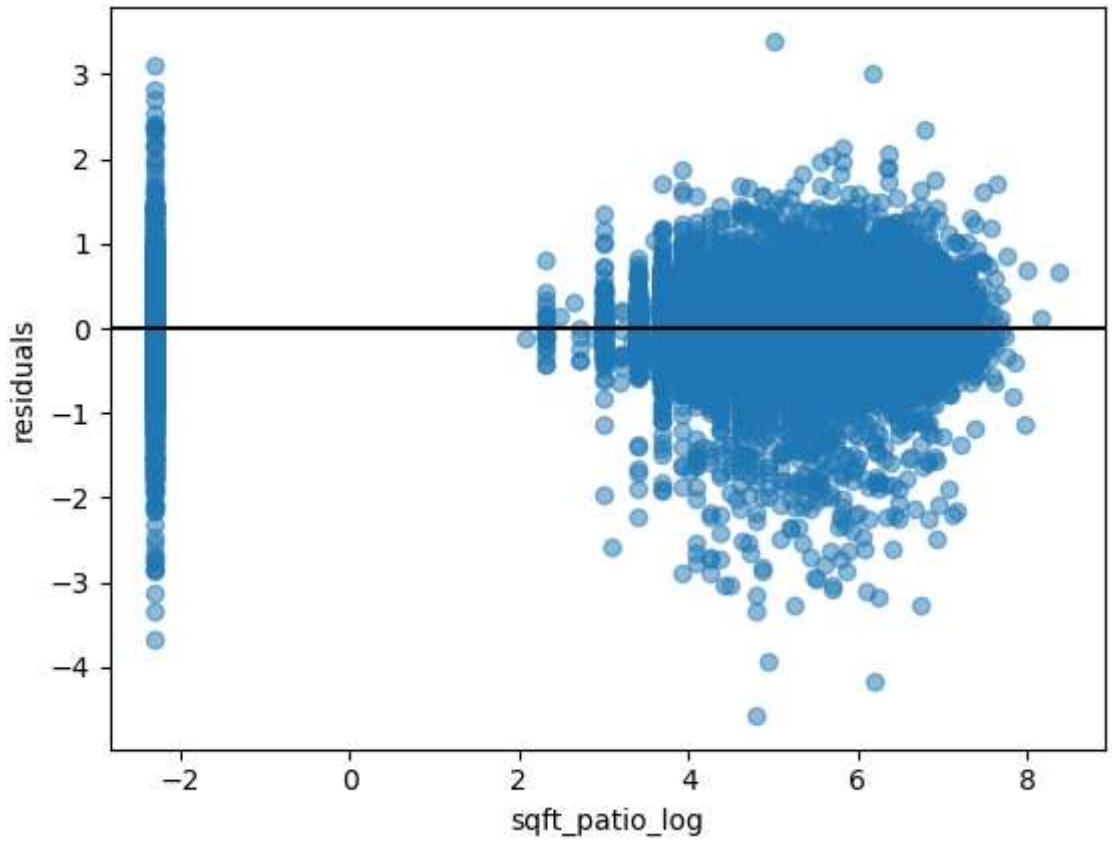


Above, very modest if any improvement in lowering the upward trend in residuals. Thus there still are non-linearity issues, and maybe heterskedaticity.

Below, check for heterskedaticity, normality of residuals, non linearity.

In [133]:

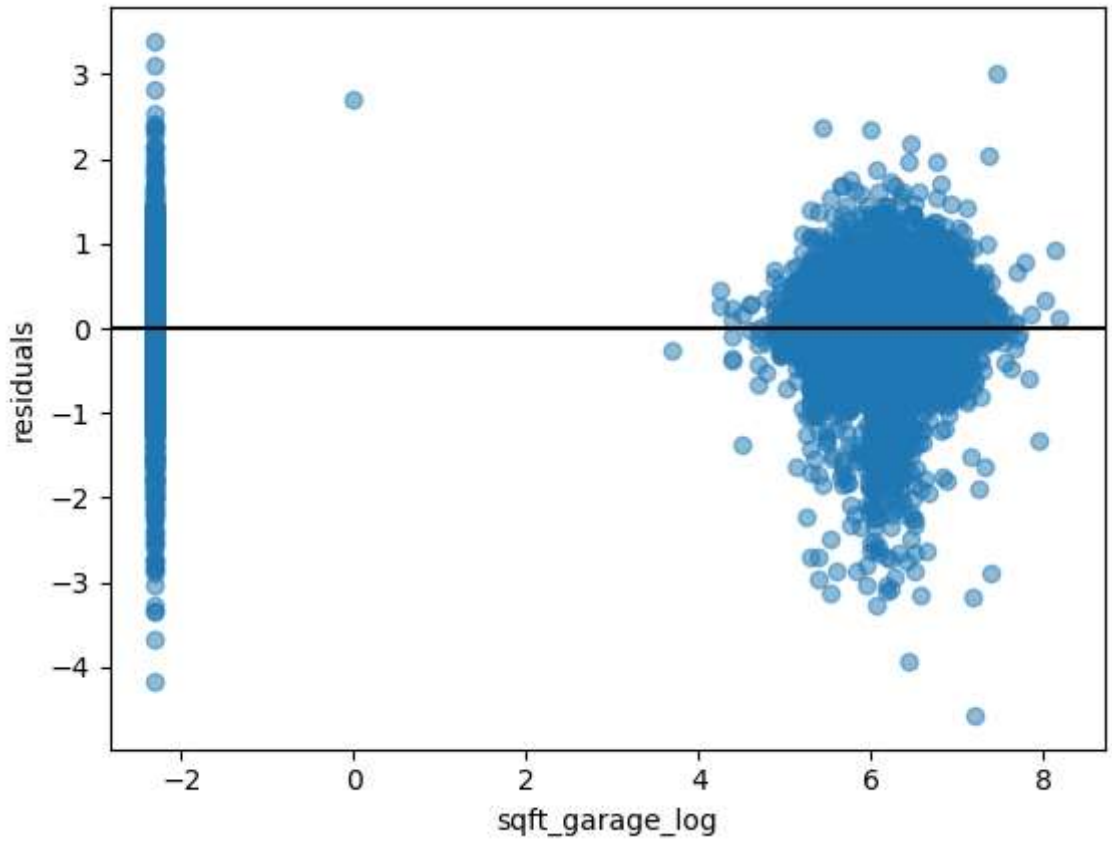
```
fig, ax = plt.subplots()

ax.scatter(df_X3_logs["sqft_patio_log"], model_X3_logs_results.resid, alpha
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_patio_log")
ax.set_ylabel("residuals");
```



Above, there is a decreasing trend in residuals, cause by 0 values, suggesting possible heterosedaticity.

In [134]:

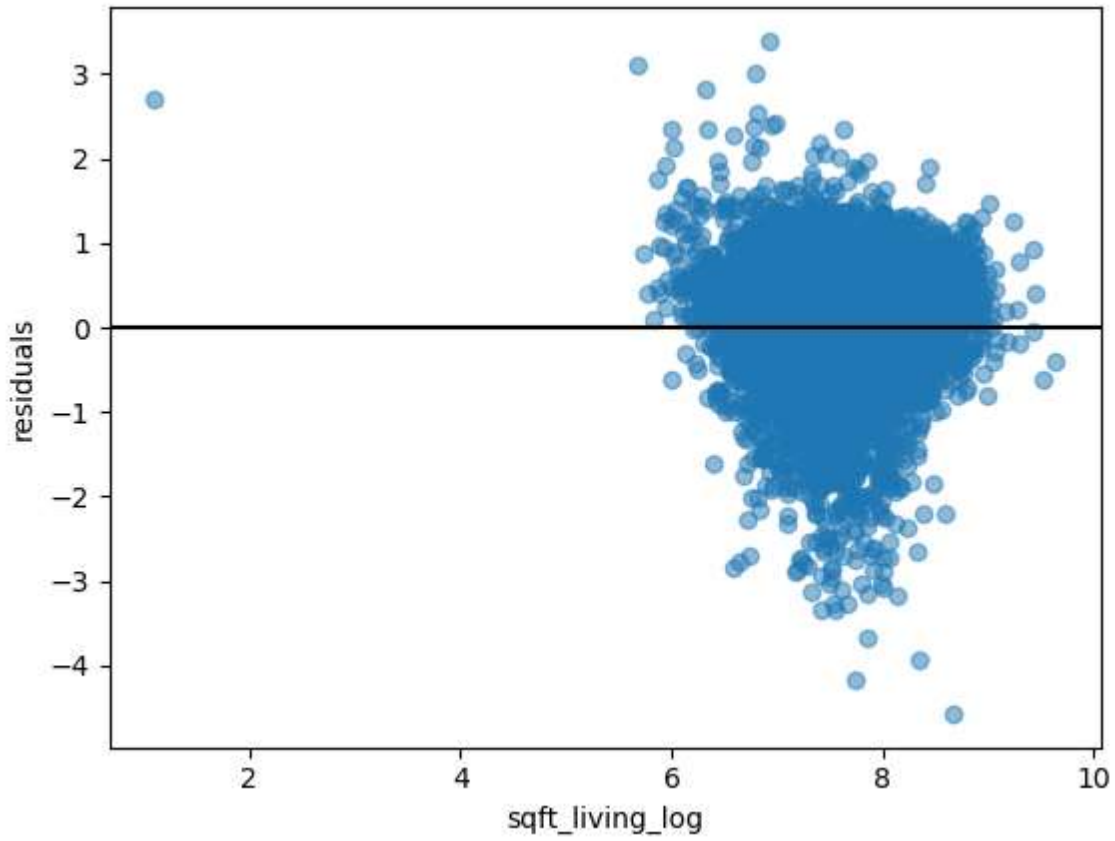
```
fig, ax = plt.subplots()

ax.scatter(df_X3_logs["sqft_garage_log"], model_X3_logs_results.resid, alph
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_garage_log")
ax.set_ylabel("residuals");
```



Above, there is a decreasing trend in residuals, cause by 0 values, suggesting possible heterosedaticity.

In [135]:

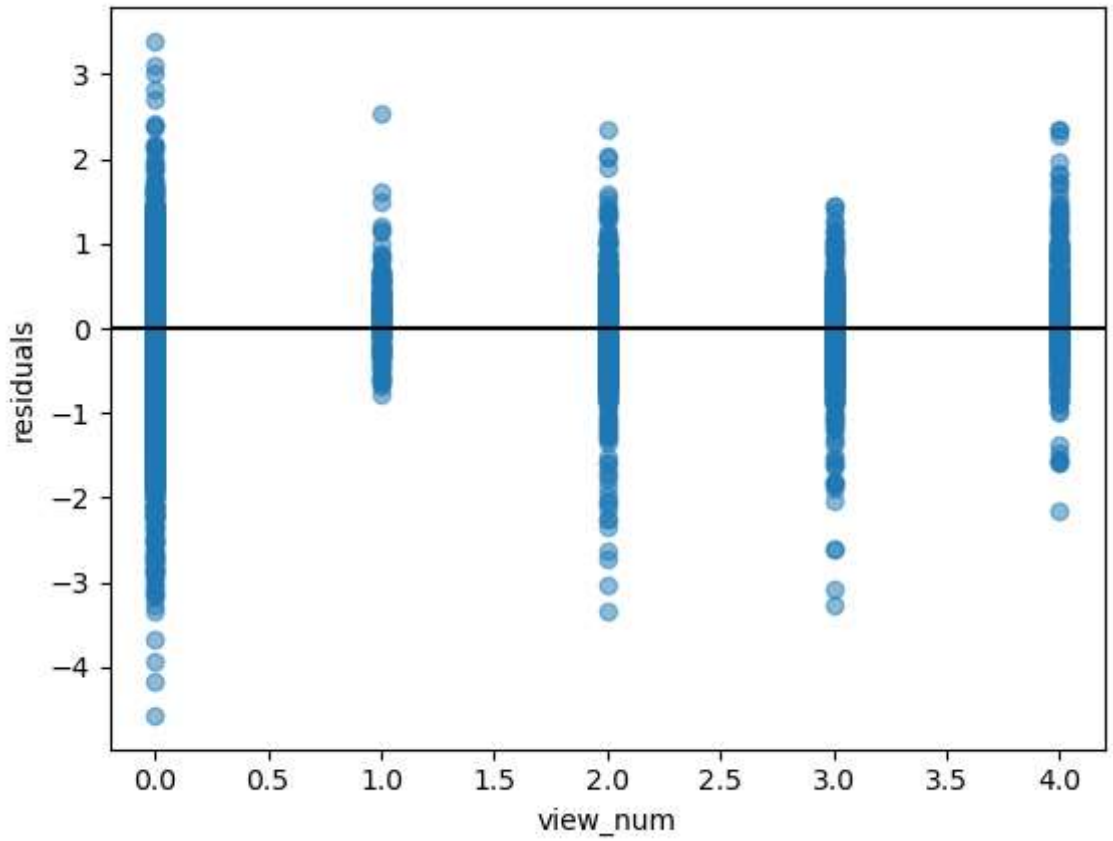
```
fig, ax = plt.subplots()

ax.scatter(df_X3_logs["sqft_living_log"], model_X3_logs_results.resid, alph
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_living_log")
ax.set_ylabel("residuals");
```



Above, there may be a pattern in the variance of residuals suggesting some heteroskedaticity.

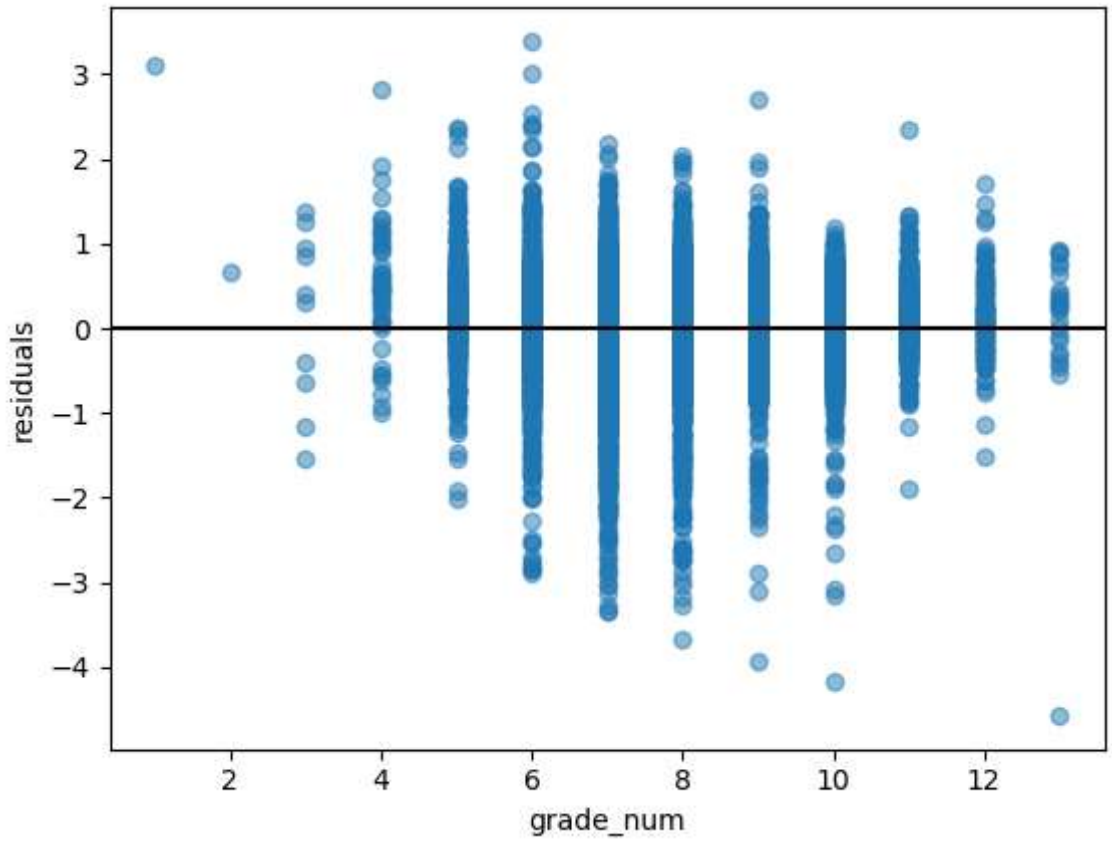
In [136]:

```
fig, ax = plt.subplots()

ax.scatter(df_X3_logs["view_num"], model_X3_logs_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("view_num")
ax.set_ylabel("residuals");
```



Above, variance of residuals are somewhat consistent, suggesting homoskedaticity.

In [137]:
```
fig, ax = plt.subplots()

ax.scatter(df_X3_logs["grade_num"], model_X3_logs_results.resid, alpha =.5)
ax.axhline(y=0, color="black")
ax.set_xlabel("grade_num")
ax.set_ylabel("residuals");
```



Above, there may be a pattern in the variance of residuals suggesting some heteroskedaticity.

In [ ]:
```
```

model_X3_logs has similar issues of non-linearity, non-normality (Jarque-Beta test) heteroskedacities, although heteroskedacities are improved. But 0 values in garage and patio variables are still problematic.

Final model:

Log grade and view due to non-normality(despite improvement) and some heteroskedaticity.

Add categorical variables Waterfront and Jumbo to increase rsquared.

Also drop outliers in sqft_gar and sqft_pat variables to decrease heteroskedaticity. (And drop associated dummy variables as 0 patio/no garage houses removed.)

In [ ]:
```
```

In [138]:
```
df_X4_drop = df_X3_logs.copy()
```

In [139]:
```
y_drop_X4=y_log_X3.copy()
```

In [140]:
```
df_X4_drop['y_drop_X4']=y_drop_X4
```

In [141]:
```
df_X4_drop['WaterFront_Yes']= df['waterfront_YES']
```

remove 0 outliers.

In [142]:
```
df_X4_drop = df_X4_drop.drop(df_X4_drop.loc[df_X4_drop['sqft_garage_log']<-
```

In [143]:
```
df_X4_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19606 entries, 3 to 30154
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   grade_num        19606 non-null  int32  
 1   view_num         19606 non-null  int64  
 2   sqft_living_log  19606 non-null  float64
 3   garage_yes       19606 non-null  int64  
 4   patio_yes        19606 non-null  int64  
 5   sqft_garage_log  19606 non-null  float64
 6   sqft_patio_log   19606 non-null  float64
 7   y_drop_X4        19606 non-null  float64
 8   WaterFront_Yes   19606 non-null  uint8  
dtypes: float64(4), int32(1), int64(3), uint8(1)
memory usage: 1.3 MB
```

In [144]:
```
df_X4_drop = df_X4_drop.drop(df_X4_drop.loc[df_X4_drop['sqft_patio_log']<-2
```

In [145]:
```
df_X4_drop.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15834 entries, 3 to 30153
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   grade_num        15834 non-null  int32  
 1   view_num         15834 non-null  int64  
 2   sqft_living_log  15834 non-null  float64
 3   garage_yes       15834 non-null  int64  
 4   patio_yes        15834 non-null  int64  
 5   sqft_garage_log  15834 non-null  float64
 6   sqft_patio_log   15834 non-null  float64
 7   y_drop_X4        15834 non-null  float64
 8   WaterFront_Yes   15834 non-null  uint8  
dtypes: float64(4), int32(1), int64(3), uint8(1)
memory usage: 1.0 MB
```

In [146]:
```
y_X4_drops =df_X4_drop['y_drop_X4'].copy()
```

In [147]:
```
df_X4_drop= df_X4_drop.drop("garage_yes", axis=1)
```

In [148]:
```
df_X4_drop= df_X4_drop.drop("patio_yes", axis=1)
```

In [149]:
```
df_X4_drop['grade_num_log']= np.log(df_X4_drop['grade_num'])

df_X4_drop= df_X4_drop.drop("grade_num", axis=1)
```

add 1 to view_num so don't have to log zero value

In [150]:
```
df_X4_drop['view_num'] = df_X4_drop['view_num'].map(lambda x: x+1)

df_X4_drop['view_num_log']= np.log(df_X4_drop['view_num'])
df_X4_drop= df_X4_drop.drop("view_num", axis=1)
```

In [151]:
```
df_X4_drop= df_X4_drop.drop("y_drop_X4", axis=1)
```

In [152]:
```
y_drop_X4
```

Out[152]:
```
0        13.422468
1        13.732129
2        12.647548
3        13.560618
4        13.292106
           ...
30150    14.256986
30151    14.087825
30152    13.592367
30153    13.560618
30154    13.122363
Name: y_log, Length: 28981, dtype: float64
```

In [153]:
```
df_X4_drop
```

Out[153]:

	sqft_living_log	sqft_garage_log	sqft_patio_log	WaterFront_Yes	grade_num_log	view
3	7.677864	5.298317	5.598422	0	2.197225	
4	7.021084	6.309918	3.401197	0	1.945910	
7	7.702556	6.086775	5.327876	0	2.079442	
8	7.757906	6.086775	4.248495	0	2.079442	
9	7.999679	6.291569	5.135798	0	2.079442	
...	...	...	...	...	...	
30147	7.649693	6.086775	3.688879	0	1.945910	
30148	8.039157	6.565265	4.700480	0	2.079442	
30149	7.146772	5.298317	4.094345	0	2.079442	
30152	7.390181	5.480639	4.700480	0	1.945910	
30153	7.851661	6.173786	4.605170	0	2.079442	

15834 rows × 6 columns

Add engineered Jumbo area variable:

In [154]:
```
df_X4_drop['Jumbo']=data['Jumbo']
```

In [155]:
```
model_X4_drop = sm.OLS(y_X4_drops, sm.add_constant(df_X4_drop))
model_X4_drop_results = model_X4_drop.fit()
```

In [156]:
```
model_X4_drop_results.summary()
```

Out[156]:

OLS Regression Results

Dep. Variable:	y_drop_X4	**R-squared:**	0.514
Model:	OLS	**Adj. R-squared:**	0.514
Method:	Least Squares	**F-statistic:**	2396.
Date:	Fri, 19 May 2023	**Prob (F-statistic):**	0.00
Time:	10:24:54	**Log-Likelihood:**	-8348.9
No. Observations:	15834	**AIC:**	1.671e+04
Df Residuals:	15826	**BIC:**	1.678e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.0719	0.071	99.122	0.000	6.932	7.212
sqft_living_log	0.4832	0.013	37.195	0.000	0.458	0.509
sqft_garage_log	-0.1035	0.010	-10.201	0.000	-0.123	-0.084
sqft_patio_log	0.0355	0.004	8.876	0.000	0.028	0.043
WaterFront_Yes	0.3019	0.030	10.034	0.000	0.243	0.361
grade_num_log	1.6657	0.036	45.754	0.000	1.594	1.737
view_num_log	0.1033	0.009	11.941	0.000	0.086	0.120
Jumbo	0.5701	0.020	28.430	0.000	0.531	0.609

Omnibus:	4378.249	**Durbin-Watson:**	2.015
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	37618.359
Skew:	-1.084	**Prob(JB):**	0.00
Kurtosis:	10.233	**Cond. No.**	253.

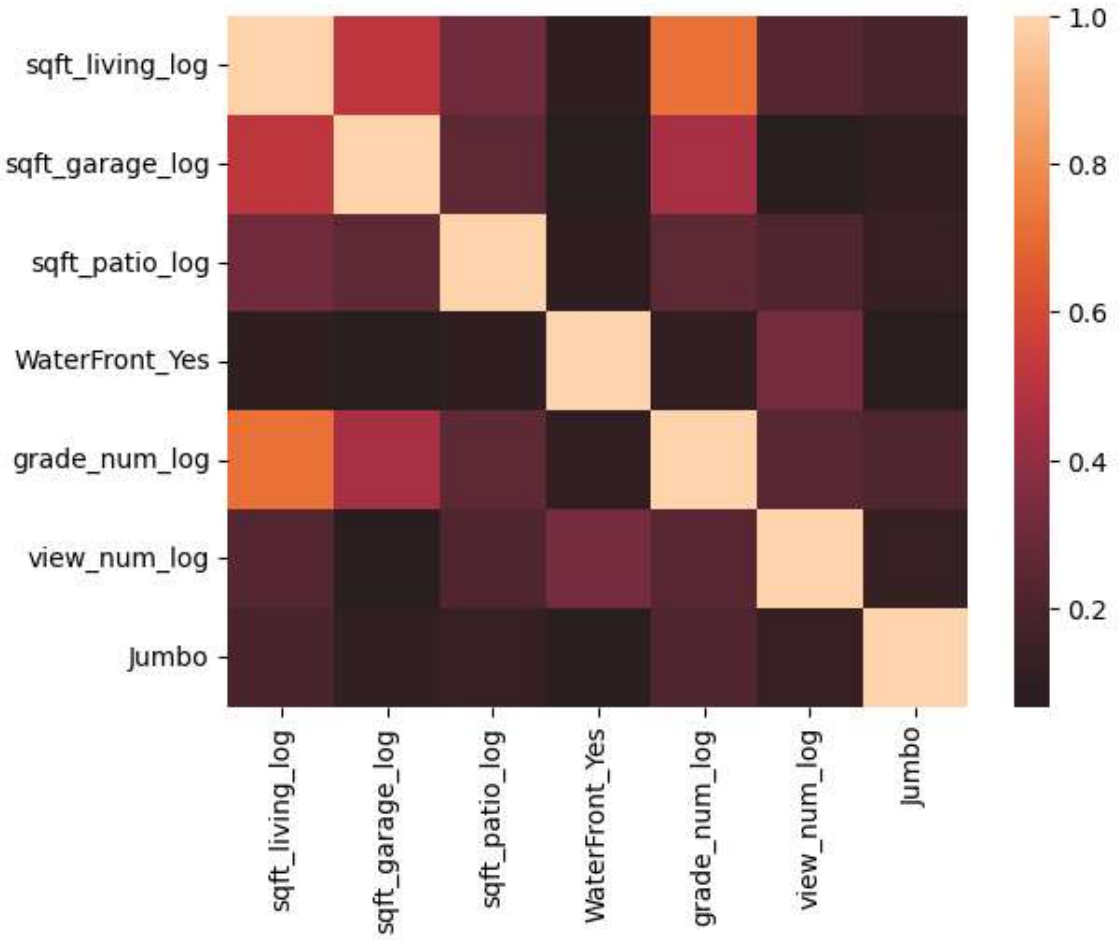
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [157]:
```
df_X4_drop.corr()
```

Out[157]:

	sqft_living_log	sqft_garage_log	sqft_patio_log	WaterFront_Yes	grade_num_
sqft_living_log	1.000000	0.516483	0.313917	0.102699	0.720
sqft_garage_log	0.516483	1.000000	0.253390	0.070055	0.459
sqft_patio_log	0.313917	0.253390	1.000000	0.096536	0.250
WaterFront_Yes	0.102699	0.070055	0.096536	1.000000	0.109
grade_num_log	0.720475	0.459446	0.250268	0.109460	1.000
view_num_log	0.232556	0.067089	0.216588	0.328209	0.240
Jumbo	0.194114	0.109539	0.128863	0.081534	0.218

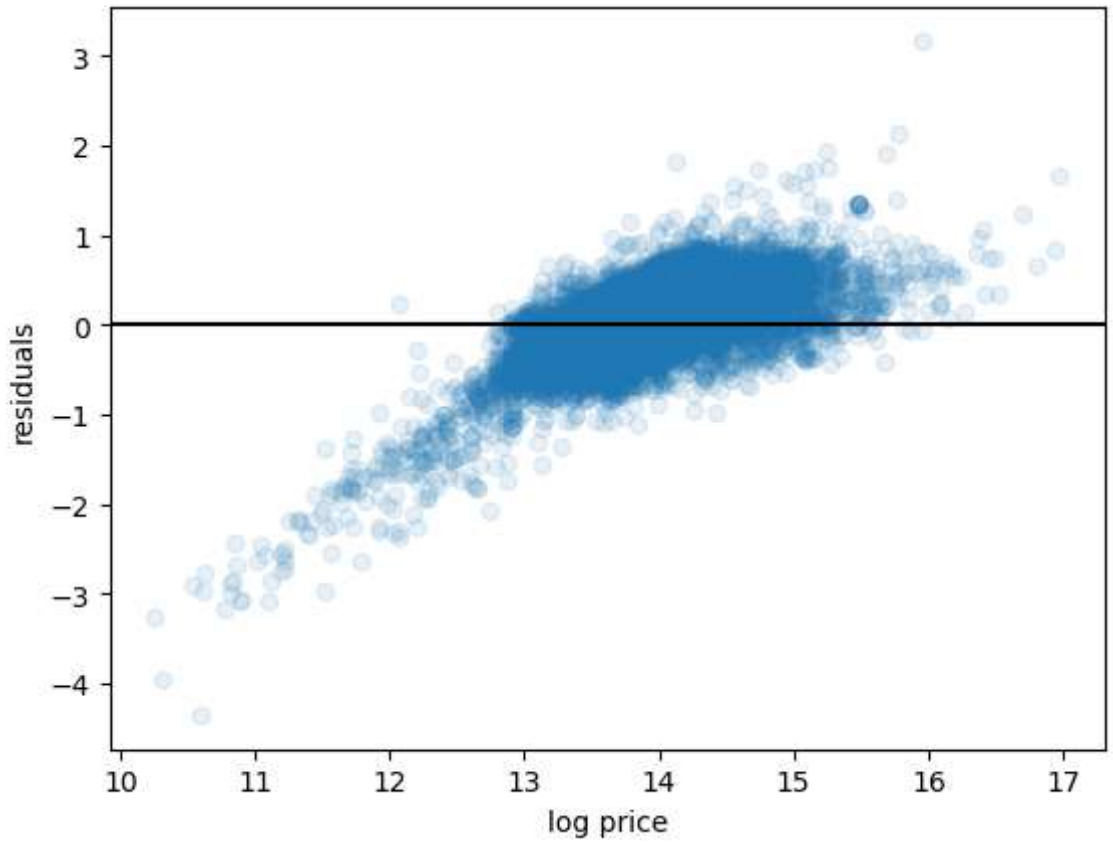
In [158]:
```
sns.heatmap(df_X4_drop.corr(), center=0);
```



Above, multicollinearity is low, all correlations below .75.

In [159]:

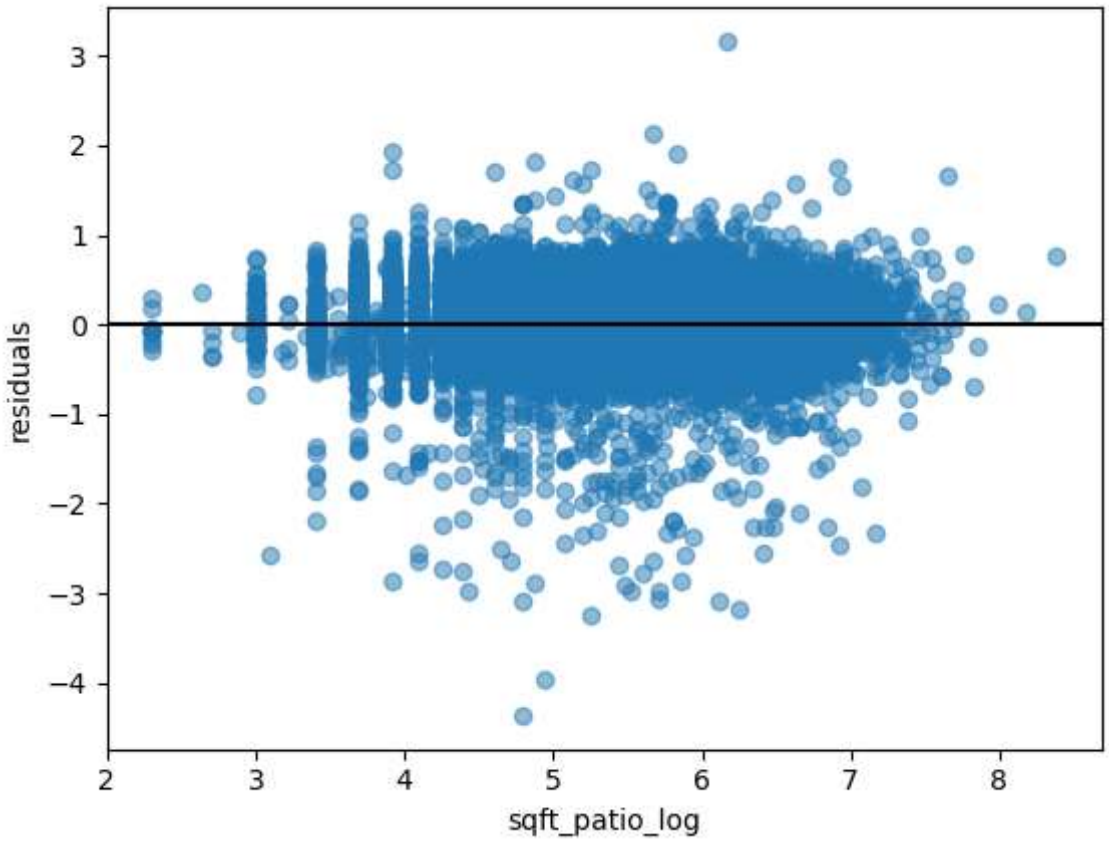
```
fig, ax = plt.subplots()

ax.scatter(y_X4_drops, model_X4_drop_results.resid, alpha=.1)
ax.axhline(y=0, color="black")
ax.set_xlabel("log price")
ax.set_ylabel("residuals");
```



Above, this residual plot is similar to the previous model residual plot, (increasing), thus there are still some heteroskedaticity and non-linearity issues.

In [160]:

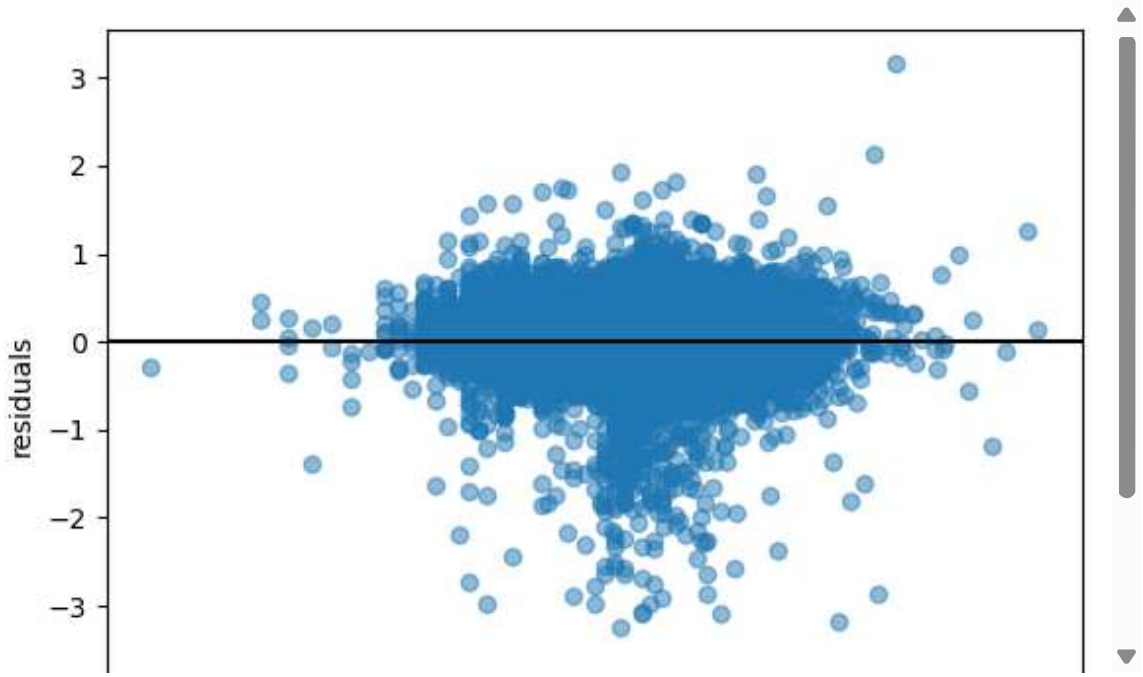
```
fig, ax = plt.subplots()

ax.scatter(df_X4_drop["sqft_patio_log"], model_X4_drop_results.resid, alpha
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_patio_log")
ax.set_ylabel("residuals");
```



Above is homoskedastic after drops of outliers.

In [161]:

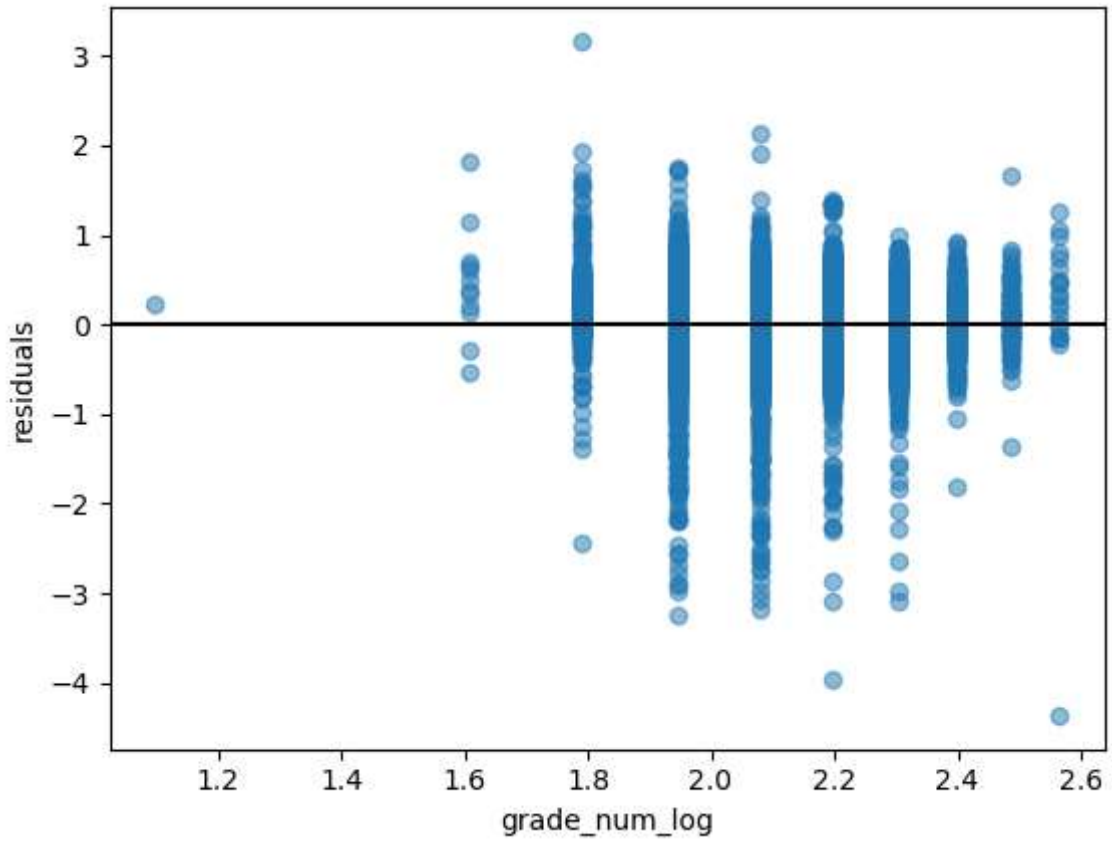
```
fig, ax = plt.subplots()

ax.scatter(df_X4_drop["sqft_garage_log"], model_X4_drop_results.resid, alph
ax.axhline(y=0, color="black")
ax.set_xlabel("sqft_garage_log")
ax.set_ylabel("residuals");
```



Above is homoskedastic after drops of outliers.

In [162]:

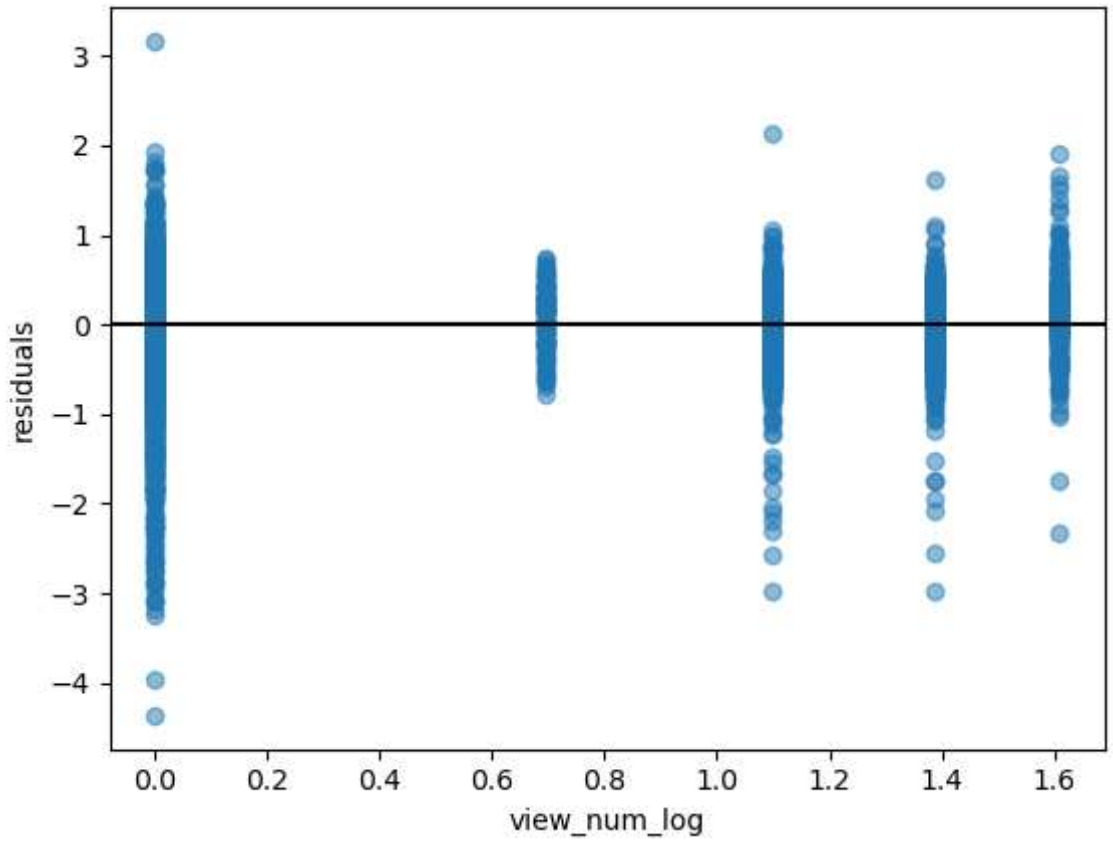
```
fig, ax = plt.subplots()

ax.scatter(df_X4_drop["grade_num_log"], model_X4_drop_results.resid, alpha
ax.axhline(y=0, color="black")
ax.set_xlabel("grade_num_log")
ax.set_ylabel("residuals");
```



Above, there is a pattern in the variance of residuals, suggesting some heteroskedaticity.

In [163]:

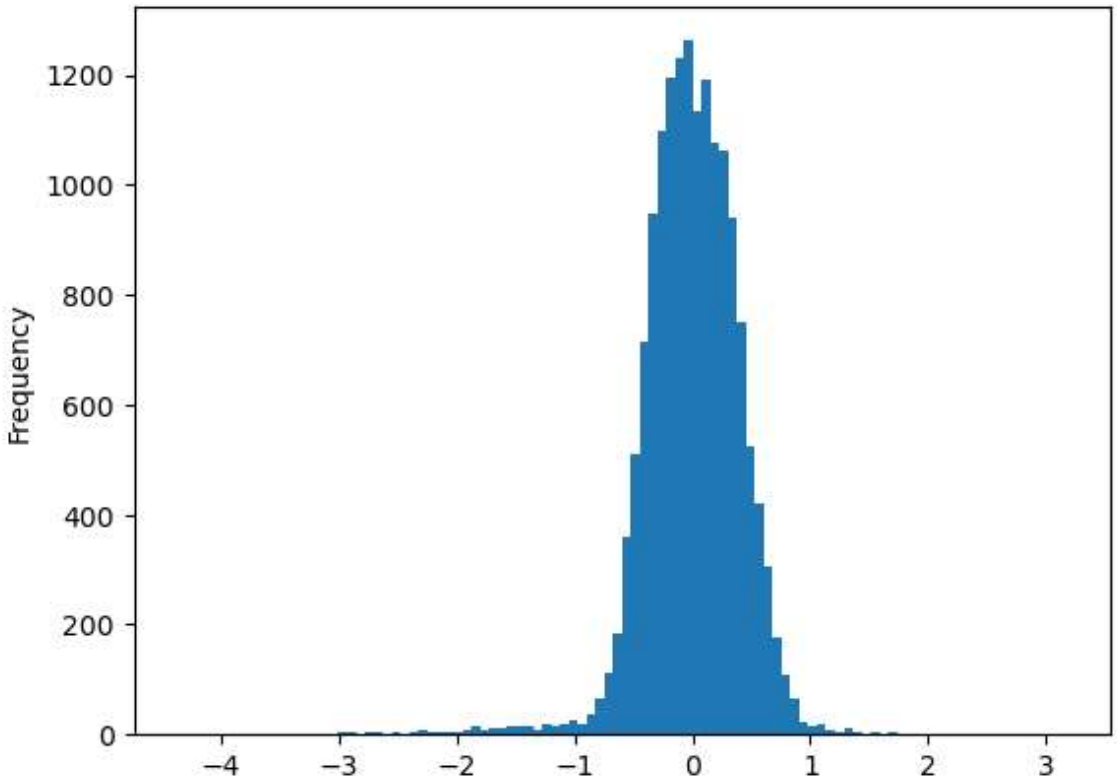
```
fig, ax = plt.subplots()

ax.scatter(df_X4_drop["view_num_log"], model_X4_drop_results.resid, alpha =
ax.axhline(y=0, color="black")
ax.set_xlabel("view_num_log")
ax.set_ylabel("residuals");
```



Above, variance of residuals are somewhat consistent, suggesting homoskedaticity.

In [164]:
```
model_X4_drop_results.resid.plot(kind ='hist', bins = 100)
```

Out[164]: <AxesSubplot:ylabel='Frequency'>



Above, residual plot appears normally distributed, and is improved from previous model, however J-B test still failed suggesting non-normality.

Results/Recommendations/Next steps

Model Evaluation

In [165]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
y_pred = model_X4_drop_results.predict(sm.add_constant(df_X4_drop))

MSE = mean_squared_error(y_pred, y_X4_drops, squared=False)
MSE
```

Out[165]: 0.4099747157167996

This is the mean squared error in terms of log(Y). This is a measure of how far off the predictions of log(price) are from the actual log(price). Ideally, one would convert to terms of change in Y (but this may not be as simple as taking the exponent of the mse of log(Y)).

In [166]:
```
RMSE = MSE**.5
RMSE
```

Out[166]: 0.6402926797307615

This is the root mean squared error in terms of log(Y). This is about the average of how far off the predictions of log(price) are from the actual log(price). Ideally, one would convert to terms of change in Y (but this may not be as simple as taking the exponent of the mse of log(Y)).

Interpretation of coefficients

Formula for interpreting raw predictor vs log target:

For each increase of 1 unit in x we see an associated change of ((e^B-1)*100)% in y

Formula for interpreting log predictor vs log target:

For each increase of p% in x we see an associated change of ((e^(B* log((100+p)/100))-1)*100)% in y

In [172]:
```
RawPredictors = ['WaterFront_Yes', 'Jumbo']
LogPredictors = ['sqft_living', 'sqft_garage', 'sqft_patio', 'grade_num', '
RawBetas = [0.3019, 0.5701]
LogBetas =[0.4832, -0.1035, 0.0355, 1.6657, 0.1033]


def RawDLogI(RawPredictors, RawBetas):
    for name, beta in zip(RawPredictors, RawBetas):
        PerChange = ((np.exp(beta)-1)*100)
        print ('For each increase of 1 unit in {} we see an associated chan
```

In [173]:
```
RawDLogI(RawPredictors, RawBetas)
```

```
For each increase of 1 unit in WaterFront_Yes we see an associated change
of 35.24259773493921 % in price
For each increase of 1 unit in Jumbo we see an associated change of 76.844
38869805086 % in price
```

In [178]:
```
def LogDLogI(LogPredictors, LogBetas):
    for name, beta in zip(LogPredictors, LogBetas):
        PerChangeforP= '((np.exp({}* log((100+p)/100))-1)*100)'.format(beta
        PerChangefor1= ((np.exp(beta* np.log((100+1)/100))-1)*100)
        print ('For each increase of p% in {} we see an associated change o
```

In [175]:
```
LogDLogI(LogPredictors, LogBetas)
```

```
For each increase of p% in sqft_living we see an associated change of ((n
p.exp(0.4832* log((100+p)/100))-1)*100)% in price. So for each 1% increase
in sqft_living we see an associated change of 0.4819576846217144% in price
For each increase of p% in sqft_garage we see an associated change of ((n
p.exp(-0.1035* log((100+p)/100))-1)*100)% in price. So for each 1% increas
e in sqft_garage we see an associated change of -0.10293291202720933% in p
rice
For each increase of p% in sqft_patio we see an associated change of ((np.
exp(0.0355* log((100+p)/100))-1)*100)% in price. So for each 1% increase i
n sqft_patio we see an associated change of 0.035329914073312096% in price
For each increase of p% in grade_num we see an associated change of ((np.e
xp(1.6657* log((100+p)/100))-1)*100)% in price. So for each 1% increase in
grade_num we see an associated change of 1.6712381247844244% in price
For each increase of p% in view_num we see an associated change of ((np.ex
p(0.1033* log((100+p)/100))-1)*100)% in price. So for each 1% increase in
view_num we see an associated change of 0.10283976156946206% in price
```

Interpretation of coefficients table

Categorical Variables	% Effect of its presence on price
WaterFront	35.24
Jumbo Area	76.84
Numeric Variables	**% Effect of its 1% increase on price**
sqft_living	0.48
sqft_garage	0.10
sqft_patio	0.04
grade_num	1.67
view_num	0.10

Recommendations

- To determine a price of a house with a waterfront, take a similar house without a waterfront and add 35.24% to the non-waterfront price.
  - To determine a price of a house in the Jumbo area, take a similar house not in the Jumbo area and add 76.84% to the non-Jumbo price.

In [179]:
```
((np.exp(0.4832* np.log((100+10)/100))-1)*100)
```

Out[179]: 4.713082767481547

- To determine a price of a house, take a similar house with about 10% less sqft of living area and add 4.71% to that price.

Next steps

- Establish a better interpretation of the root mean squared error.
- Further analyze the negative coefficient of garage size variable.
- Also testing interaction variables (e.g. differing lot sizes and house sizes for different geographic areas.)