EDA: House prices

Cleaning data and exploratory analysis of data related to houses.

In [1]:
```
# First, let's import library we are going to work with

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

In [2]:
```
# Import csv file
df = pd.read_csv('house_data.csv')
df.shape
```

Out[2]:
```
(5000, 16)
```

There are 5000 observations and 16 variables in our dataset. Each observation correspond to a house.

In [3]:
```
# Print the first rows of the table
df.head(5)
```

Out[3]:

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built	bedrooms	bathrooms	sqrt_ft	garage	kitc
0	21530491	5300000.0	85637	-1.103.782	31.356.362	2154.00	5272.00	1941	13	10	10500	0	
1	21529082	4200000.0	85646	-111.045.371	31.594.213	1707.00	10422.36	1997	2	2	7300	0	
2	3054672	4200000.0	85646	-111.040.707	31.594.844	1707.00	10482.00	1997	2	3	None	None	
3	21919321	4500000.0	85646	-111.035.925	31.645.878	636.67	8418.58	1930	7	5	9019	4	

Looking into the first observations, the characteristics of the houses are refered to geolocalization (latitud, longitud, zipcode), furniture and facilities, monetary value (taxes and sold_price), and others elements (HOA and MLS).

Now, let's check missing values.

In [4]:
```
# Count the number of NA's in each column
df.isna().sum()
```

Out[4]:
```
MLS                 0
sold_price          0
zipcode             0
longitude           0
latitude            0
lot_acres          10
taxes               0
year_built          0
bedrooms            0
bathrooms           0
sqrt_ft             0
garage              0
kitchen_features    0
fireplaces         25
floor_covering      0
HOA                 0
dtype: int64
```

In [5]:
```
df.loc[(df['HOA']=='None') | (df['bathrooms']=='None') | (df['sqrt_ft']=='None') | (df['garage']=='None')].shape
```

Out[5]:
```
(600, 16)
```

There are 35 observations with missing values and 600 observations with the word 'None'. A total of 635 observations with missing values representing around 12.7%, which is greater than 5%. It is not possible to drop missing values, so let's check how to deal with them.

In [6]:
```
# Now we are check the type of variables of the table
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   MLS               5000 non-null   int64  
 1   sold_price        5000 non-null   float64
 2   zipcode           5000 non-null   int64  
 3   longitude         5000 non-null   object 
 4   latitude          5000 non-null   object 
 5   lot_acres         4990 non-null   float64
 6   taxes             5000 non-null   float64
 7   year_built        5000 non-null   int64  
 8   bedrooms          5000 non-null   int64  
 9   bathrooms         5000 non-null   object 
 10  sqrt_ft           5000 non-null   object 
 11  garage            5000 non-null   object 
 12  kitchen_features  5000 non-null   object 
 13  fireplaces        4975 non-null   float64
 14  floor_covering    5000 non-null   object 
 15  HOA               5000 non-null   object 
dtypes: float64(4), int64(4), object(8)
memory usage: 625.1+ KB
```

Let's begin with string data. In this case, columns 12 and 14 should be considered.

String variables

Kitchen features

In [7]:
```
df.kitchen_features
```

Out[7]:
```
0                Dishwasher, Freezer, Refrigerator, Oven
1                           Dishwasher, Garbage Disposal
2              Dishwasher, Garbage Disposal, Refrigerator
3       Dishwasher, Double Sink, Pantry: Butler, Refri...
4       Dishwasher, Garbage Disposal, Refrigerator, Mi...
                              ...
4995    Dishwasher, Double Sink, Garbage Disposal, Gas...
4996    Dishwasher, Double Sink, Electric Range, Garba...
4997    Dishwasher, Electric Range, Island, Refrigerat...
4998    Dishwasher, Double Sink, Garbage Disposal, Gas...
4999    Compactor, Dishwasher, Double Sink, Island, Ap...
Name: kitchen_features, Length: 5000, dtype: object
```

All features of the kitchen are together in the same column, so we need to extract all possible information of these features. The idea is to create new columns of these features and fill in with zeros or ones in case the house own the feature.

In [8]:
```
unique_kitchen_features = []

for i in range(1,len(df)):
    temporal_list = df.iloc[i]['kitchen_features'].split(', ')

    for x in temporal_list:
        if x not in unique_kitchen_features:
            unique_kitchen_features.append(x)

print(unique_kitchen_features)
```

```
['Dishwasher', 'Garbage Disposal', 'Refrigerator', 'Double Sink', 'Pantry: Butler', 'Microwave', 'Oven', 'Co
mpactor', 'Freezer', 'Electric Range', 'Island', 'Appliance Color: Stainless', 'Countertops: CORIAN', 'Micro
wave: Built In', 'Oven: Built In', 'Countertops: wood and granite', 'Oven: 2 ovens-Main 1-GH', 'Gas Range',
'Prep Sink', 'Appliance Color: Black', 'Countertops: Granite', 'Desk', 'Warming Drawer', 'Countertops: Marbl
e', 'Lazy Susan', 'Pantry: Walk-In', 'Countertops: granite', 'Countertops: Concrete', 'Microwave: Drawer',
'Oven: Double Wall', 'Microwave: Stainless', 'Oven: Stainless', 'Countertops: Granite & Quartz', 'Indoor Gri
ll', 'Wet Bar', 'Microwave: stainless', 'Oven: stainless', 'Water Purifier', 'Microwave: wolf', 'Oven: wol
f', 'Countertops: quartz', 'Microwave: built in', 'Countertops: Marble & Granite', 'Microwave: yes', 'Oven:
yes', 'Pantry: Closet', 'Appliance Color: Other', 'Countertops: Ceaserstone', 'Microwave: Built In Wall', 'O
ven: Built In Wall', 'Oven: Oven', 'Countertops: slate/honed Granite', 'Microwave: Thermador', 'Oven: Therma
dor', 'Countertops: GRANITE', 'Countertops: Quartz', 'Microwave: Viking', 'Oven: Gas', 'Microwave: Wolf', 'O
ven: Wolf', 'Microwave: Microwave', 'Oven: Double', 'Reverse Osmosis', 'Missing: All Appliances', 'Oven: 2 O
VENS', 'Pantry: Cabinet', 'None', 'Countertops: Maole', 'Countertops: wood', 'Oven: vintage', 'Countertops:
Quartzite', 'Countertops: Graniet', 'Microwave: YES', 'Microwave: Undercabinet', 'Countertops: Granite/Slat
e', 'Oven: GE Monogram', 'Appliance Color: White', 'Countertops: STONE', 'Microwave: STAINLESS', 'Oven: STAI
NELSS', 'Microwave: Kitchen Aid', 'Oven: Kitchen Aid', 'Countertops: corrian', 'Microwave: .', 'Countertops:
Slab Granite', 'Microwave: Included', 'Oven: Viking', 'Countertops: Stainless', 'Microwave: none', 'Microwav
e: -', 'Oven: double', 'Countertops: concrete', 'Microwave: Black', 'Oven: Double/Black', 'Oven: Double Viki
ng Ovens', 'Countertops: Brazillian slate', 'Microwave: Convection/Steam', 'Oven: .', 'Microwave: Built-In',
```

Looking into most common words of this column, I will create a function to detect these words and fill in with zeros or ones in case the column includes that specific word.

In [9]:
```
def IdentifyKeyWord(x, word):
    if word in x.lower():
        return 1
    else:
        return 0

df['KF_Dishwasher'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'dishwasher'))
df['KF_GarbageDisposal'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'garbage disposal'))
df['KF_Refrigerator'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'refrigerator'))
df['KF_DoubleSink'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'double sink'))
df['KF_Microwave'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'microwave'))
df['KF_Oven'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'oven'))
df['KF_Compactor'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'compactor'))
df['KF_Freezer'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'freezer'))
df['KF_ElectricRange'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'electric range'))
df['KF_Island'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'island'))
df['KF_GasRange'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'gas range'))
df['KF_Countertops'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'countertops'))
df['KF_Desk'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'desk'))
```

In [10]:
```
df[['KF_Dishwasher','KF_GarbageDisposal','KF_Refrigerator','KF_DoubleSink','KF_Microwave','KF_Oven','KF_Compactc
```

Out[10]:
```
KF_Dishwasher         4857
KF_GarbageDisposal    4520
KF_Refrigerator       4234
KF_DoubleSink         1164
KF_Microwave          3625
KF_Oven               3977
KF_Compactor           432
KF_Freezer             395
KF_ElectricRange       401
KF_Island             1252
KF_GasRange           1307
KF_Countertops        1482
KF_Desk                327
dtype: int64
```

Most of the houses own a dishwasher or garbage disposal, but just a few own a freezer or desk.

Floor covering

Now, we will do the same analysis as the kitchen features.

In [11]:
```
unique_floor_covering = []

for i in range(1,len(df)):
    temporal_list = df.iloc[i]['floor_covering'].split(', ')

    for x in temporal_list:
        if x not in unique_floor_covering:
            unique_floor_covering.append(x)

print(unique_floor_covering)
```

```
['Natural Stone', 'Other', 'Other: Rock', 'Ceramic Tile', 'Laminate', 'Wood', 'Carpet', 'Concrete', 'Mexican T
ile', 'Other: Porcelain', 'Vinyl', 'Other: Brick', 'Other: Brick Pavers', 'Other: Flagstone', 'Other: Marble-M
aster Bath', 'Other: Marble', 'Granite', 'Other: concrete tile', 'Other: Porcelain Tile', 'Other: Quartzite',
'Other: Porcelyn', 'Other: Tile', 'Other: Porcelain tile', 'Other: CONCRETE TILE', 'Other: Brick Floor', 'Othe
r: Saltillo', 'Other: Travertine Tile', 'Indoor/Outdoor', 'Other: Travertine', 'Other: Lime Stone', 'Other: Li
mestone', 'Other: Multiple Types', 'Other: Refinished Brick', 'Other: studio laminate', 'Other: Porcelain/Engi
neered', 'Other: Brick inlayed', 'Other: Terrazzo', 'Other: Cement tiles/Bamboo', 'Other: Slate', 'Other: gray
saltillo', 'Other: Organic Wool Carpet', 'Other: brick', 'Other: flagstone', 'Other: Egytian sandstone', 'Othe
r: Travertine & slate', 'Other: Cork', 'Other: TBD', 'Other: porcelain tile', 'Other: porclain tile', 'Other:
None', 'Other: 100% Porcelain Tile', 'Other: San Marcos Mex Tile', 'Other: Tile-Other', 'Other: carpet- guest
house', 'Other: travertine/flagstone', 'Other: Recycled Porcelain', 'Other: Slate tile', 'Other: Travertine &
Slate', 'Other: Porclain Tile', 'Other: travertine', 'Other: Mesquite wood floors', 'Other: brick pavers', 'Ot
her: Concrete tile', 'Other: Travertine Accents', 'Other: Luxury Vinyl', 'Other: Throughout home', 'Other: Roj
o Concrete Overla', 'Other: Wood laminate', 'Other: Italian tile', 'Other: Brazilian Pergo', 'Other: engineere
d wood', 'Other: Polished Concrete', 'Other: Talavera Floors', 'Other: Vinyl Plank', 'Other: NEW WOOD PLANK TI
LE', 'Other: saltillo', 'Other: Travertine/Marble', 'Other: Italian Tile', 'Other: Canterra Stone', 'Other: Po
rcelain Plank Tile', 'Other: Porcelain Wood Tile', 'Other: Polished Brick', 'None', 'Other: Tile/Powder Rm',
'Other: Parquet', 'Other: Itailian Porclaine', 'Other: Red Brick', 'Other: High End Laminate', 'Other: Master
Bedroom/ Tile', 'Other: Eng wood', 'Other: cork', 'Other: Travertine tile', 'Other: See remarks', 'Other: PLAN
K TILE', 'Other: Real polishd aggrgt', 'Other: Pergo', 'Other: Brick in Studio', 'Other: Porcelain tile 24x2
4', 'Other: Wood Laminate', 'Other: porcelain', 'Other: Wood Laminate Water', 'Other: WOOD LAMINATE', 'Other:
Stained concrete', 'Other: Custom Saltillo', 'Other: Saltillo tile', 'Other: 20 x 20 on Diagonal', 'Other: Woo
d like', 'Other: Porcelain-wood', 'Other: Polish concrete', 'Other: acrylic overlay', 'Other: Engineered Woo
d', 'Other: NEW Plank Tile', 'Other: Travertine entry', 'Other: Dyed Concrete', 'Other: Hardwood', 'Other: Car
pet bedrooms only', 'Other: vinyl planks', 'Other: UPG Flooring', 'Other: Lux Vinyl', 'Other: scored concret
e', 'Other: Bamboo', 'Other: Saltillo on Patio', 'Other: porcelain wood tile', 'Other: Tile bathrooms', 'Othe
r: Wood Plan Laminate']
```

In [12]:
```
df['FC_Stone'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'natural stone'))
df['FC_Ceramic'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'ceramic tile'))
df['FC_Laminate'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'laminate'))
df['FC_Wood'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'wood'))
df['FC_Carpet'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'carpet'))
df['FC_Concrete'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'concrete'))
df['FC_MexicanTile'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'mexican tile'))
```

In [13]:
```
df[['FC_Stone','FC_Ceramic','FC_Laminate','FC_Wood','FC_Carpet','FC_Concrete','FC_MexicanTile']].sum()
```

Out[13]:
```
FC_Stone          1499
FC_Ceramic        2527
FC_Laminate         86
FC_Wood           1248
FC_Carpet         3509
FC_Concrete        756
FC_MexicanTile     660
dtype: int64
```

Numerical variables

HOA

In [14]:
```
df[df['HOA'] == 'None']
```

Out[14]:

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built	bedrooms	bathrooms	...	KF_GasRange	I
2	3054672	4200000.0	85646	-111.040.707	31.594.844	1707.00	10482.00	1997	2	3	...	0	
3	21919321	4500000.0	85646	-111.035.925	31.645.878	636.67	8418.58	1930	7	5	...	0	
10	21900396	2776518.0	85640	-111.045.441	31.562.121	147.18	7330.36	1935	5	5	...	0	
39	4113243	2200000.0	85640	-11.104.463	31.562.185	147.18	3902.44	1935	5	4.5	...	0	
49	21818418	1800000.0	86024	-111.228.462	34.596.971	59.30	4509.90	2003	2	3	...	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4990	21906583	526710.0	85748	-11.072.839	32.221.871	9.18	3390.67	1960	4	3	...	0	
4995	21810382	495000.0	85641	-110.661.829	31.907.917	4.98	2017.00	2005	5	3	...	1	
4997	21832452	475000.0	85192	-110.755.428	32.964.708	12.06	1000.00	1969	3	2	...	0	
4998	21900515	550000.0	85745	-111.055.528	32.296.871	1.01	5822.93	2009	4	4	...	1	
4999	4111490	450000.0	85621	-110.913.054	31.385.259	4.16	2814.48	1988	4	4	...	0	

562 rows × 36 columns

There are 563 observations with 'None'. We should impute using the mean, median, mode or zeros.

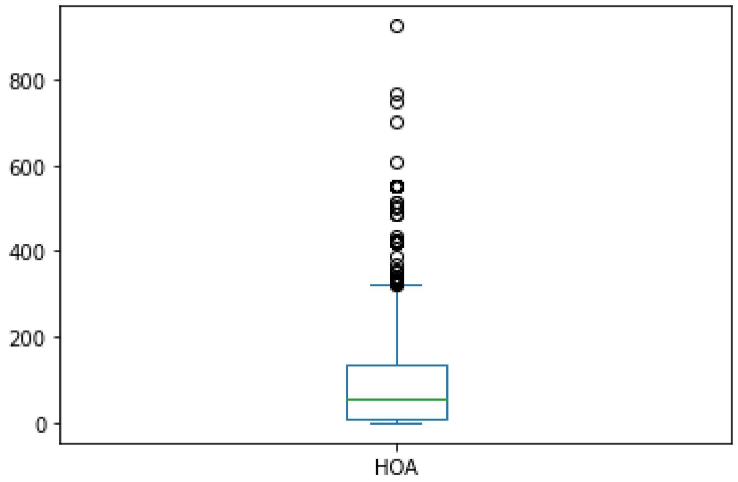
In [16]:
```
df[df['HOA'] != 'None']['HOA'].astype(float)
```

```
-> 1292     new_values = astype_array(values, dtype, copy=copy)
   1293 except (ValueError, TypeError):
   1294     # e.g. astype_nansafe can fail on object-dtype of strings
   1295     #  trying to convert to float
   1296     if errors == "ignore":

File ~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py:1237, in astype_array(values, dtype, copy)
   1234     values = values.astype(dtype, copy=copy)
   1236 else:
-> 1237     values = astype_nansafe(values, dtype, copy=copy)
   1239 # in pandas we don't store numpy str dtypes, so convert to object
   1240 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File ~\anaconda3\lib\site-packages\pandas\core\dtypes\cast.py:1181, in astype_nansafe(arr, dtype, copy, skipna)
   1177     raise ValueError(msg)
   1179 if copy or is_object_dtype(arr.dtype) or is_object_dtype(dtype):
   1180     # Explicit copy, or required since NumPy can't view from / to object.
-> 1181     return arr.astype(dtype, copy=True)
```

Some observations has a comma, so we have to change them to dots.

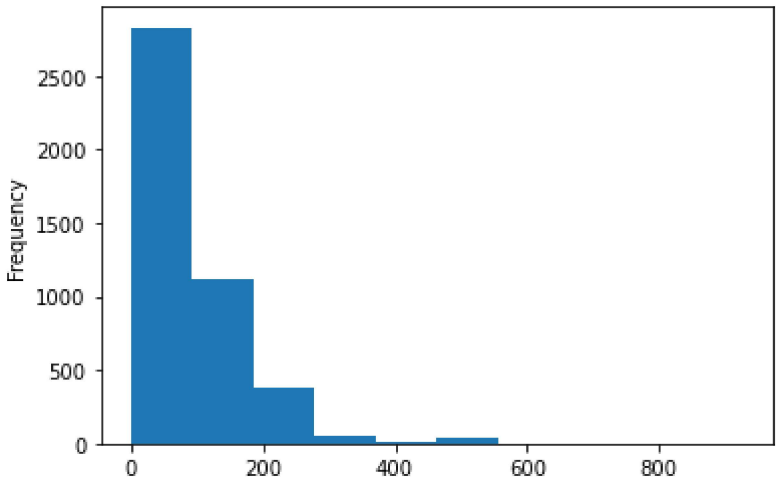
In [17]:
```
df['HOA'] = df.HOA.str.replace(',','.')
df[df['HOA'] != 'None']['HOA'].astype(float).plot.box()
```

Out[17]: <AxesSubplot:>



In [18]:
```
df[df['HOA'] != 'None']['HOA'].astype(float).plot.hist()
```

Out[18]: <AxesSubplot:ylabel='Frequency'>



Due to the most the observations are close to zero, we decide to fill in missing values with zeros.

In [19]:
```
df['HOA'] = df.HOA.str.replace('None','0')
df['HOA'] = df.HOA.astype(float)
```

Some observations has decimals, but should be integers. To finish, we will change to int type.

In [20]:
```
df.HOA = df.HOA.astype(int)
```

Bathrooms

In [21]:
```
df[df['bathrooms'] == 'None']
```

Out[21]:

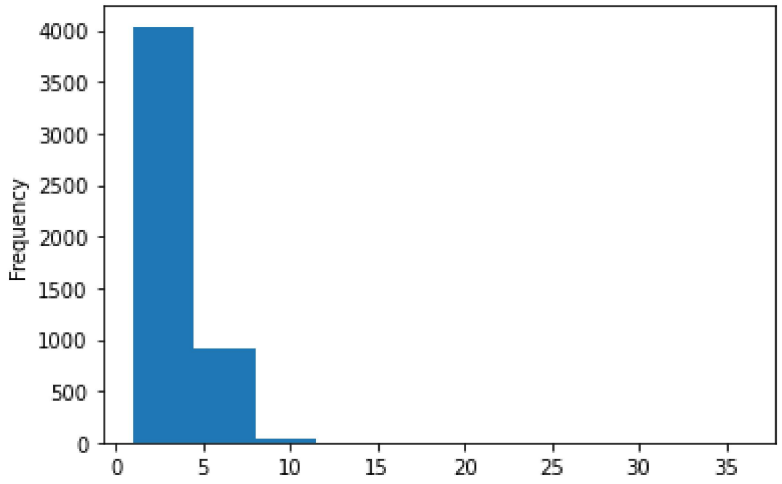
	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built	bedrooms	bathrooms	...	KF_GasRange	KF
2025	3044867	660000.0	85614	-110.969.465	31.836.723	3.60	5526.00	2007	3	None	...	1	
2766	3042851	575000.0	85614	-110.960.497	31.854.446	0.87	4623.05	2002	3	None	...	1	
3108	3047540	610000.0	85614	-111.002.544	31.840.061	1.70	3800.00	2007	3	None	...	0	
3529	3046317	535000.0	85614	-110.986.426	31.806.614	4.27	3826.25	2006	2	None	...	1	
3822	3045347	550000.0	85614	-111.008.754	31.841.141	0.99	3702.07	2007	2	None	...	0	
4812	3046287	500000.0	85646	-111.051.431	31.636.207	1.03	8102.00	1999	4	None	...	0	

6 rows × 36 columns

There are 6 observations with missing values, let's choose a value to fill in.

In [22]:
```
df[df['bathrooms'] != 'None']['bathrooms'].astype(float).plot.hist()
```

Out[22]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [23]:
```
df[df['bathrooms'] != 'None']['bathrooms'].astype(float).describe()
```

Out[23]:
```
count    4994.000000
mean        3.829896
std         1.387063
min         1.000000
25%         3.000000
50%         4.000000
75%         4.000000
max        36.000000
Name: bathrooms, dtype: float64
```

In [24]:
```
df[df['bathrooms'] != 'None']['bathrooms'].astype(float).mode()
```

Out[24]:
```
0    3.0
Name: bathrooms, dtype: float64
```

The mean is 3.82 (4 bathrooms) and the mode is 3. In this case, we opt for the mode due to bias distribution to the left.

In [25]:
```
df['bathrooms'] = df.bathrooms.str.replace('None','3')
df['bathrooms'] = df.bathrooms.astype(float)
```

SQRT_FT

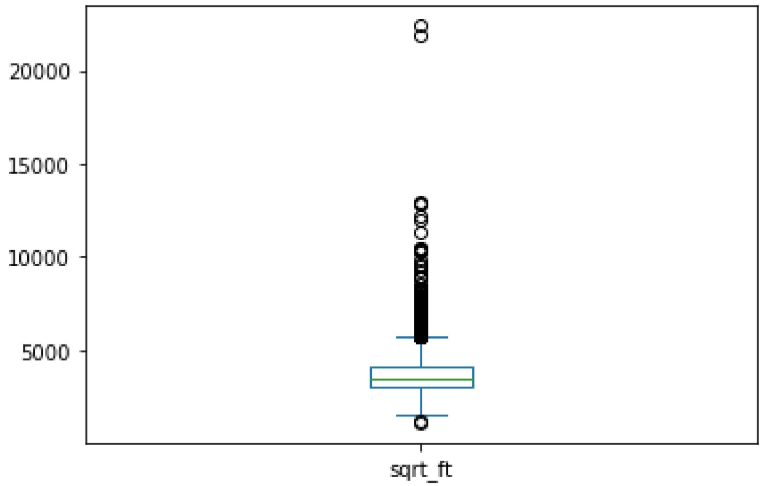
In [26]:
```
df[df['sqrt_ft'] == 'None'].shape
```

Out[26]:
```
(56, 36)
```

There are 56 observations with the term 'None'.

In [27]:
```
df[df['sqrt_ft'] != 'None']['sqrt_ft'].astype(float).plot.box()
```

Out[27]:
```
<AxesSubplot:>
```

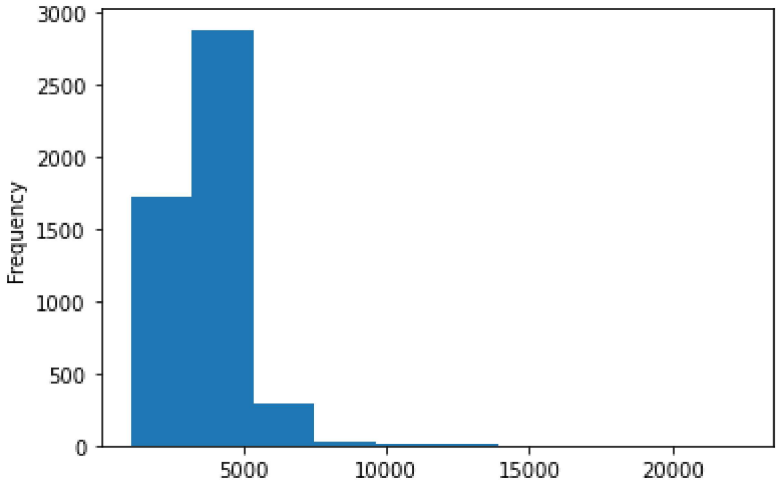


In [28]:
```
df[df['sqrt_ft'] != 'None']['sqrt_ft'].astype(float).describe()
```

Out[28]:
```
count     4944.000000
mean      3716.366828
std       1120.683515
min       1100.000000
25%       3047.000000
50%       3512.000000
75%       4130.250000
max      22408.000000
Name: sqrt_ft, dtype: float64
```

In [29]:
```
df[df['sqrt_ft'] != 'None']['sqrt_ft'].astype(float).plot.hist()
```

Out[29]:
```
<AxesSubplot:ylabel='Frequency'>
```



In this case, we will fill missing values with the median due to the distribution is asymmetric.

In [30]:
```
df['sqrt_ft'] = df.sqrt_ft.str.replace('None','3512')
df['sqrt_ft'] = df.sqrt_ft.astype(float)
```

Garage

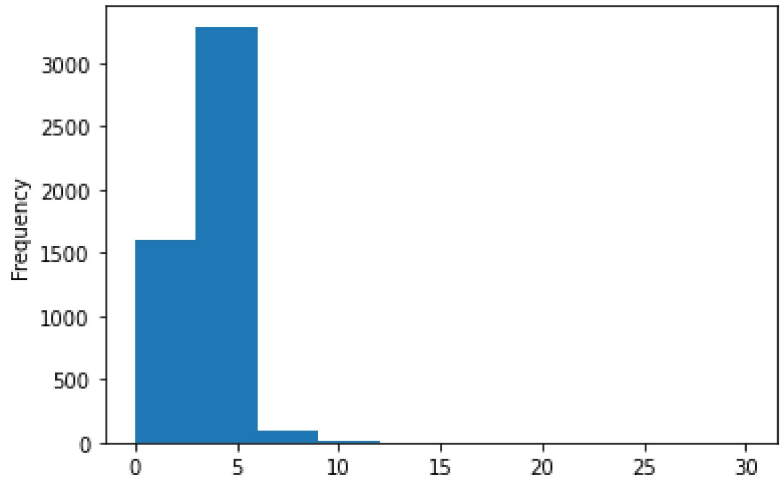
In [31]:
```
df[df['garage'] == 'None'].shape
```

Out[31]:
```
(7, 36)
```

There are only 7 observations with missing values. Let's see the distribution.

In [32]:
```
df[df['garage'] != 'None']['garage'].astype(float).plot.hist()
```

Out[32]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [33]:
```
df[df['garage'] != 'None']['garage'].astype(float).describe()
```

Out[33]:
```
count    4993.000000
mean        2.816143
std         1.192946
min         0.000000
25%         2.000000
50%         3.000000
75%         3.000000
max        30.000000
Name: garage, dtype: float64
```

In [34]:
```
df[df['garage'] != 'None']['garage'].astype(float).mode()
```

Out[34]:
```
0    3.0
Name: garage, dtype: float64
```

The mean, median and mode are equal to 3.

In [35]:
```
df['garage'] = df.garage.str.replace('None','3')
df['garage'] = df.garage.astype(float)
```

Latitude and Longitud

Some of the observations are separated by dots and range is not from -90 to 90 for latitude and -180 to 180 for longitude. To solve this problems we are going to remove the dots and divide all the observations by $1 * 10^6$.

In [36]:
```
df['latitude'] = df.latitude.str.replace('.','').astype(float)
df['longitude'] = df.longitude.str.replace('.','').astype(float)
```

```
C:\Users\Consultant\AppData\Local\Temp\ipykernel_9440\1485290633.py:1: FutureWarning: The default value of reg
ex will change from True to False in a future version. In addition, single character regular expressions will
*not* be treated as literal strings when regex=True.
  df['latitude'] = df.latitude.str.replace('.','').astype(float)
C:\Users\Consultant\AppData\Local\Temp\ipykernel_9440\1485290633.py:2: FutureWarning: The default value of reg
ex will change from True to False in a future version. In addition, single character regular expressions will
*not* be treated as literal strings when regex=True.
  df['longitude'] = df.longitude.str.replace('.','').astype(float)
```

In [37]:
```
df[['latitude','longitude']].head(5)
```

Out[37]:

	latitude	longitude
0	31356362.0	-1103782.0
1	31594213.0	-111045371.0
2	31594844.0	-111040707.0
3	31645878.0	-111035925.0
4	32285162.0	-110813768.0

To finish, we will divide by 10^6 each observation and change type to float.

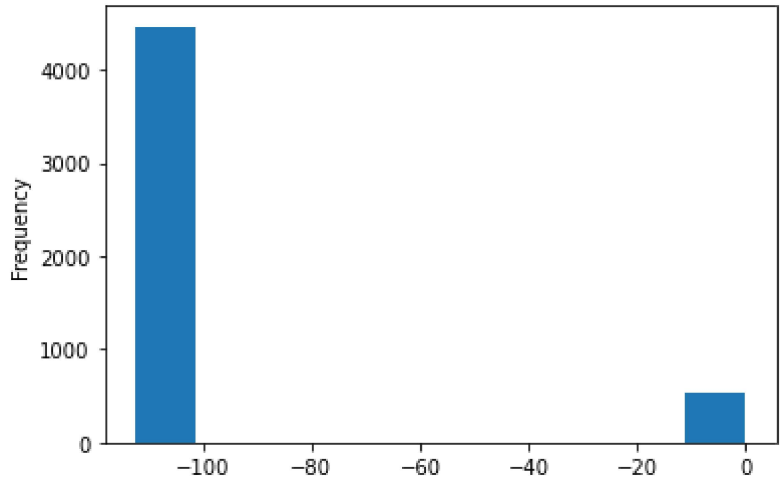
In [38]:
```
df['latitude'] = df['latitude']/10**6
df['longitude'] = df['longitude']/10**6
```

In [39]:
```
df.longitude.describe()
```

Out[39]:
```
count    5000.000000
mean     -100.029450
std        31.296943
min      -112.520168
25%      -110.971372
50%      -110.910850
75%      -110.831354
max        -0.001109
Name: longitude, dtype: float64
```

In [40]:
```
df.longitude.plot.hist()
```

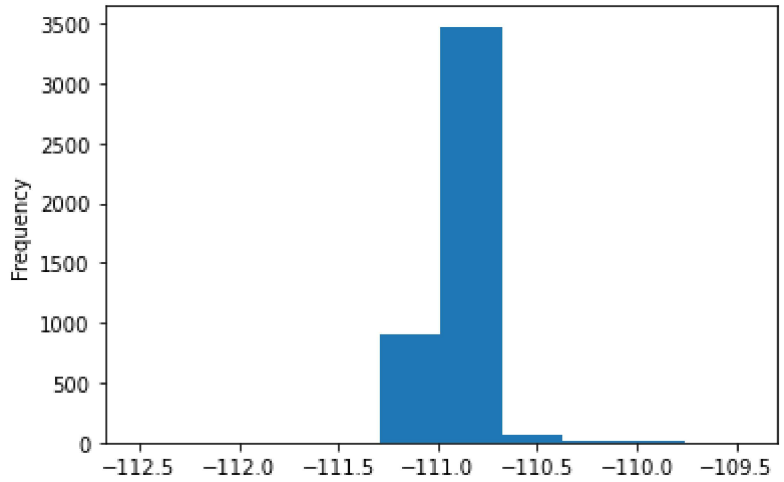
Out[40]:
```
<AxesSubplot:ylabel='Frequency'>
```



Some records have lower values than the majority of the observations. Also, there are no places near to the longitude from -20 to 0, and a latitude of around 32. For this reason, these records will be change to a range similar to their parents.

In [41]:
```
df[df['longitude'] < -100].longitude.plot.hist()
```

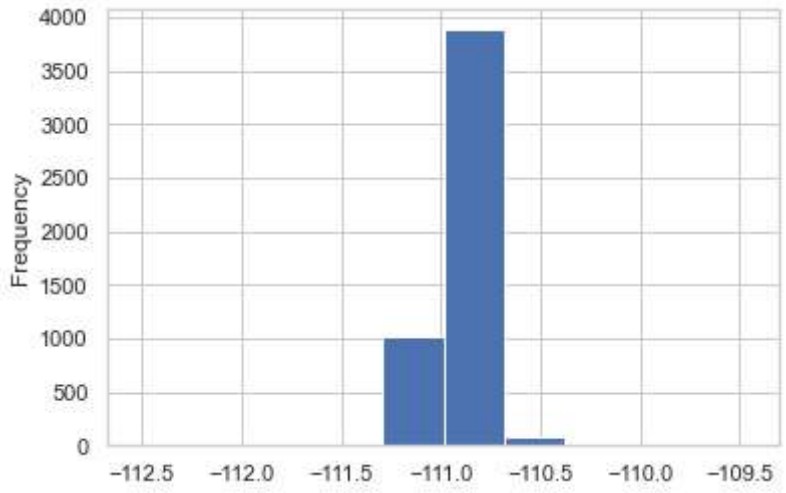
Out[41]: <AxesSubplot:ylabel='Frequency'>



In [78]:
```
def TransLon (x):
    if x > -0.01:
        return(x * 100000)
    elif x > -0.1:
        return(x * 10000)
    elif x > -1:
        return(x * 1000)
    elif x > -10:
        return(x * 100)
    elif x > -100:
        return(x * 10)
    else: return(x)
df.longitude = df.longitude.apply(TransLon)
```

In [79]:
```
df.longitude.plot.hist()
```

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

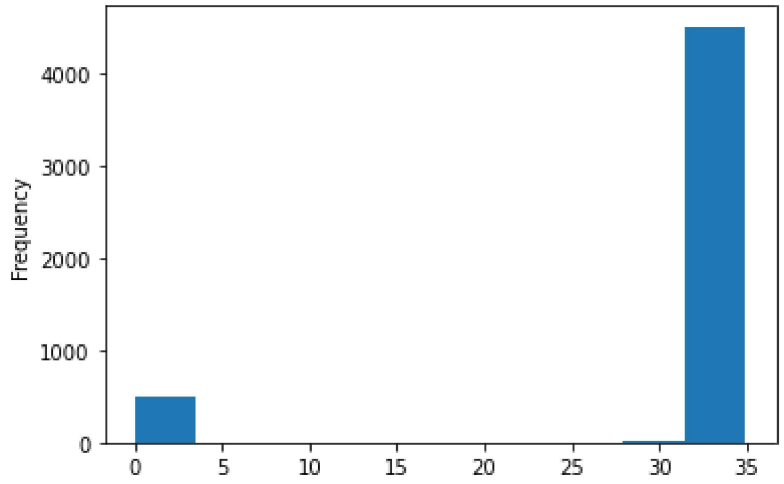


In [44]:
```
df.longitude.describe()
```

Out[44]:
```
count    5000.000000
mean     -109.883725
std        10.136770
min      -112.520168
25%      -110.978818
50%      -110.922538
75%      -110.857760
max        -0.011090
Name: longitude, dtype: float64
```

In [45]:
```
df.latitude.plot.hist()
```

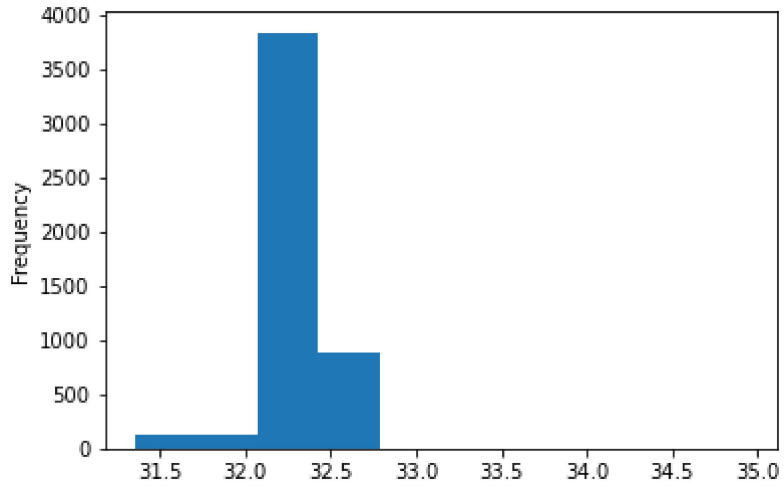
Out[45]: <AxesSubplot:ylabel='Frequency'>



In [46]:
```
def TransLat (x):
    if x < 0.01:
        return(x * 10000)
    elif x < 0.1:
        return(x * 1000)
    elif x < 1:
        return(x * 100)
    elif x < 10:
        return(x * 10)
    else: return(x)
df.latitude = df.latitude.apply(TransLat)
```

In [47]:
```
df.latitude.plot.hist()
```

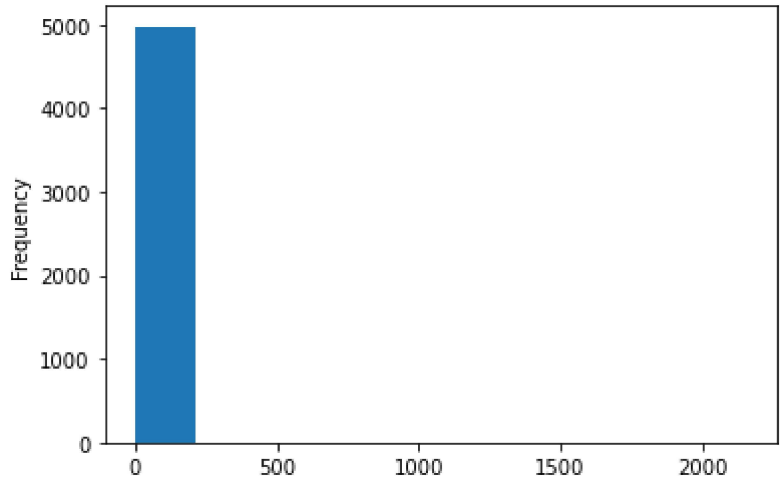
Out[47]:
```
<AxesSubplot:ylabel='Frequency'>
```



Lot_acres

In [48]:
```
df.lot_acres.plot.hist()
```

Out[48]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [49]:
```
df.lot_acres.describe()
```

Out[49]:
```
count    4990.000000
mean        4.661317
std        51.685230
min         0.000000
25%         0.580000
50%         0.990000
75%         1.757500
max      2154.000000
Name: lot_acres, dtype: float64
```

In [50]:
```
df.lot_acres.mode()
```

Out[50]:
```
0    1.0
Name: lot_acres, dtype: float64
```

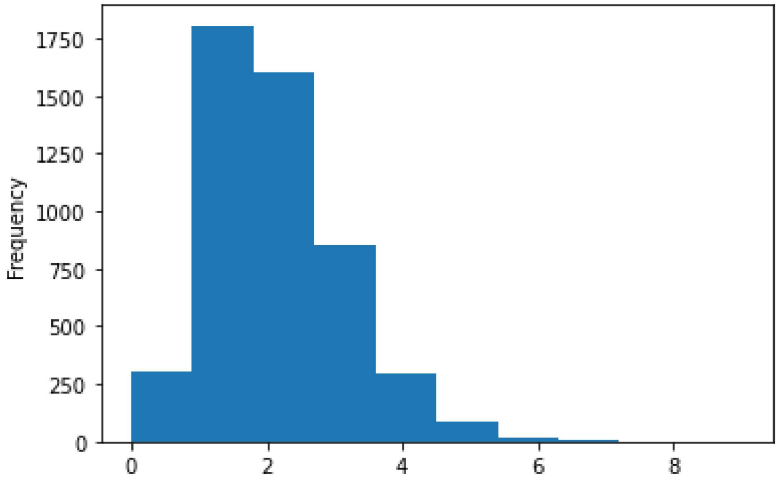
The median and the mode are the same. Also, there are a significant outliers in the distribution, so we will take 1 to fill Na's.

In [51]:
```
df.lot_acres = df.lot_acres.fillna(1)
```

Fireplaces

In [52]:
```
df.fireplaces.plot.hist()
```

Out[52]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [53]:
```
df.fireplaces.describe()
```

Out[53]:
```
count    4975.000000
mean        1.885226
std         1.136578
min         0.000000
25%         1.000000
50%         2.000000
75%         3.000000
max         9.000000
Name: fireplaces, dtype: float64
```

In [54]:
```
df.fireplaces.mode()
```

Out[54]:
```
0    1.0
Name: fireplaces, dtype: float64
```

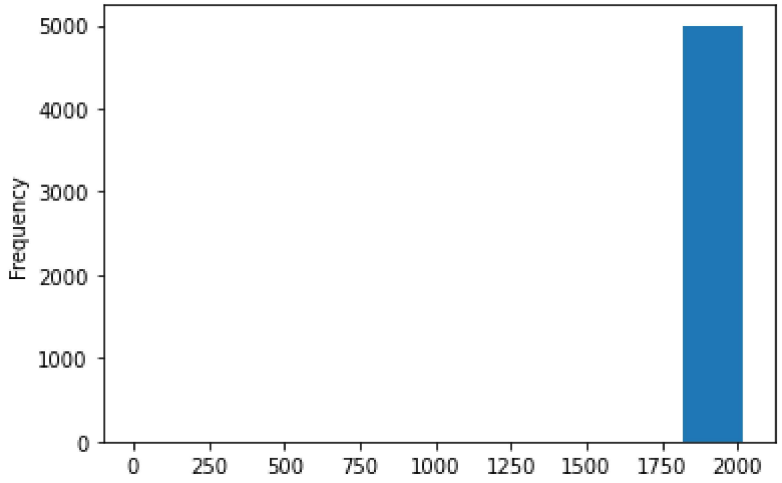
The majority of the elements are closed to the mean and median (2 fireplaces), so we will fill missing values with 2.

In [55]:
```
df.fireplaces = df.fireplaces.fillna(2)
```

Year_built

In [56]:
```
df.year_built.plot.hist()
```

Out[56]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [57]:
```
df.year_built.describe()
```

Out[57]:
```
count    5000.00000
mean     1992.32800
std        65.48614
min         0.00000
25%      1987.00000
50%      1999.00000
75%      2006.00000
max      2019.00000
Name: year_built, dtype: float64
```

There are some observations which has a record of zero. It is not possible, we assume zero is the number of years the house has been built, so we will replace zeros with last year recorded in the database.

In [58]:
```
df.year_built = df.year_built.replace(0, 2019)
```

In [59]:
```
# Let's check the type of the variables
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 36 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   MLS                  5000 non-null   int64  
 1   sold_price           5000 non-null   float64
 2   zipcode              5000 non-null   int64  
 3   longitude            5000 non-null   float64
 4   latitude             5000 non-null   float64
 5   lot_acres            5000 non-null   float64
 6   taxes                5000 non-null   float64
 7   year_built           5000 non-null   int64  
 8   bedrooms             5000 non-null   int64  
 9   bathrooms            5000 non-null   float64
 10  sqrt_ft              5000 non-null   float64
 11  garage               5000 non-null   float64
 12  kitchen_features     5000 non-null   object 
 13  fireplaces           5000 non-null   float64
 14  floor_covering       5000 non-null   object 
 15  HOA                  5000 non-null   int32  
 16  KF_Dishwasher        5000 non-null   int64  
 17  KF_GarbageDisposal   5000 non-null   int64  
 18  KF_Refrigerator      5000 non-null   int64  
 19  KF_DoubleSink        5000 non-null   int64  
 20  KF_Microwave         5000 non-null   int64  
 21  KF_Oven              5000 non-null   int64  
 22  KF_Compactor         5000 non-null   int64  
 23  KF_Freezer           5000 non-null   int64  
 24  KF_ElectricRange     5000 non-null   int64  
 25  KF_Island            5000 non-null   int64  
 26  KF_GasRange          5000 non-null   int64  
 27  KF_Countertops       5000 non-null   int64  
 28  KF_Desk              5000 non-null   int64  
 29  FC_Stone             5000 non-null   int64  
 30  FC_Ceramic           5000 non-null   int64  
 31  FC_Laminate          5000 non-null   int64  
 32  FC_Wood              5000 non-null   int64  
 33  FC_Carpet            5000 non-null   int64  
 34  FC_Concrete          5000 non-null   int64  
```

```
 35  FC_MexicanTile      5000 non-null   int64
dtypes: float64(9), int32(1), int64(24), object(2)
memory usage: 1.4+ MB
```

All the variables are float or integer, so the cleaning process is completed.

Exploratory Analysis

Let's continue finding asociations between the price of the houses and the rest of variables.

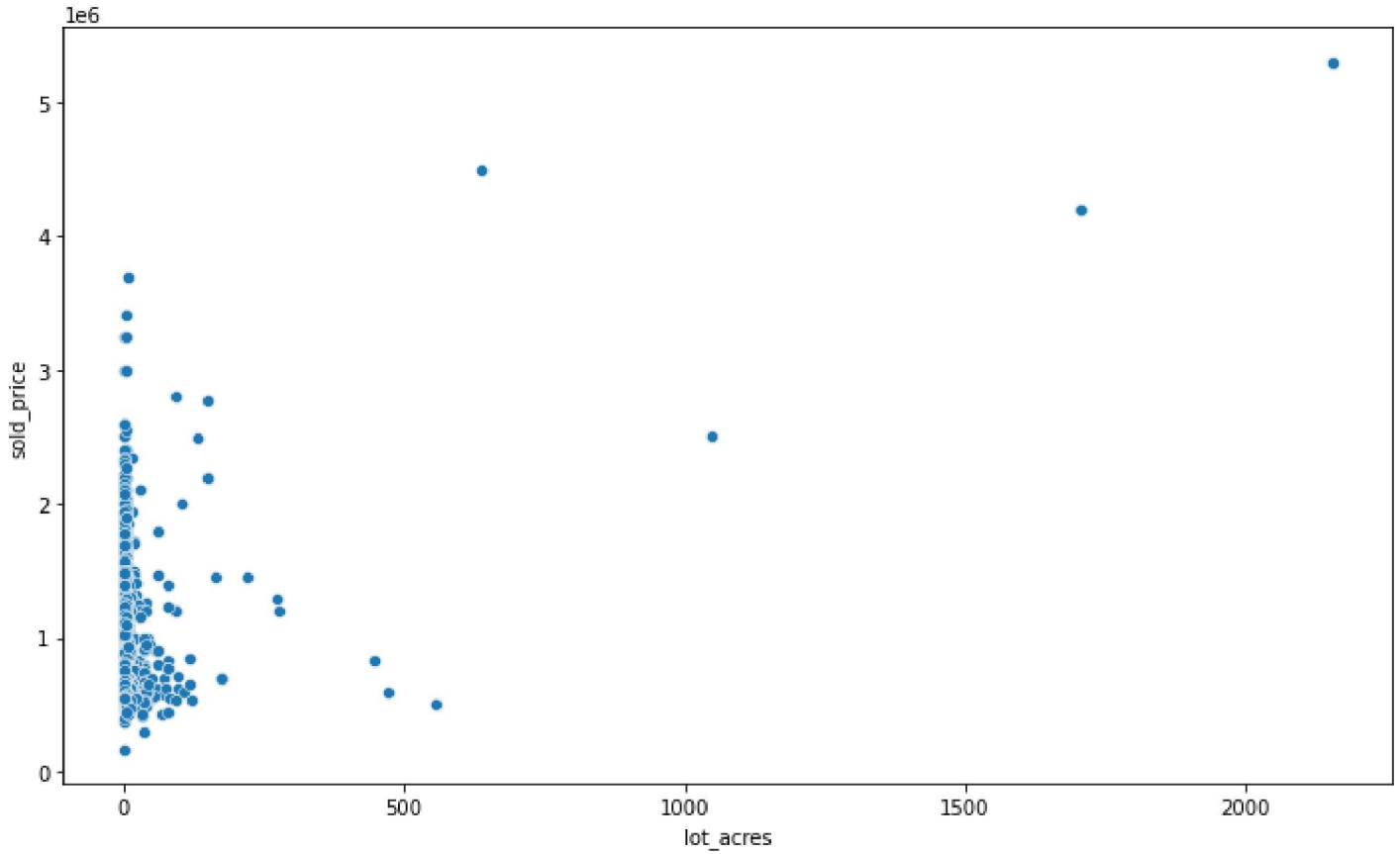
In [60]:
```
corr = df[['MLS','sold_price','zipcode','longitude','latitude','lot_acres','taxes','year_built','bedrooms','bath
corr
```

Out[60]:

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built	bedrooms	bathrooms	sqrt_ft	g
MLS	1.000000	0.031019	0.323165	0.015415	0.441466	-0.078616	0.003306	-0.000294	-0.012254	-0.053332	-0.035502	-0.0
sold_price	0.031019	1.000000	-0.039925	0.011257	0.035206	0.332329	0.023326	0.098461	0.115932	0.327019	0.524581	0.0
zipcode	0.323165	-0.039925	1.000000	-0.002556	0.457102	-0.126539	-0.001265	0.008372	0.049982	-0.050670	-0.002964	0.0
longitude	0.015415	0.011257	-0.002556	1.000000	0.008090	0.077110	-0.001653	0.023038	-0.016469	-0.003250	-0.009524	-0.0
latitude	0.441466	0.035206	0.457102	0.008090	1.000000	-0.199936	0.000823	0.172578	-0.091935	-0.078040	-0.110815	0.0
lot_acres	-0.078616	0.332329	-0.126539	0.077110	-0.199936	1.000000	-0.000567	-0.066292	0.069328	0.055113	0.106292	-0.0
taxes	0.003306	0.023326	-0.001265	-0.001653	0.000823	-0.000567	1.000000	-0.004243	0.005259	0.008996	0.037667	0.0
year_built	-0.000294	0.098461	0.008372	0.023038	0.172578	-0.066292	-0.004243	1.000000	-0.184503	-0.050859	-0.057192	0.3
bedrooms	-0.012254	0.115932	0.049982	-0.016469	-0.091935	0.069328	0.005259	-0.184503	1.000000	0.687375	0.547272	0.0
bathrooms	-0.053332	0.327019	-0.050670	-0.003250	-0.078040	0.055113	0.008996	-0.050859	0.687375	1.000000	0.662861	0.1
sqrt_ft	-0.035502	0.524581	-0.002964	-0.009524	-0.110815	0.106292	0.037667	-0.057192	0.547272	0.662861	1.000000	0.1
garage	-0.008902	0.099582	0.081066	-0.000357	0.060783	-0.052590	0.005576	0.316112	0.049750	0.108609	0.180258	1.0
fireplaces	-0.056511	0.383819	-0.018285	-0.026148	-0.087933	0.086336	0.022540	-0.126997	0.144822	0.225339	0.405575	0.0

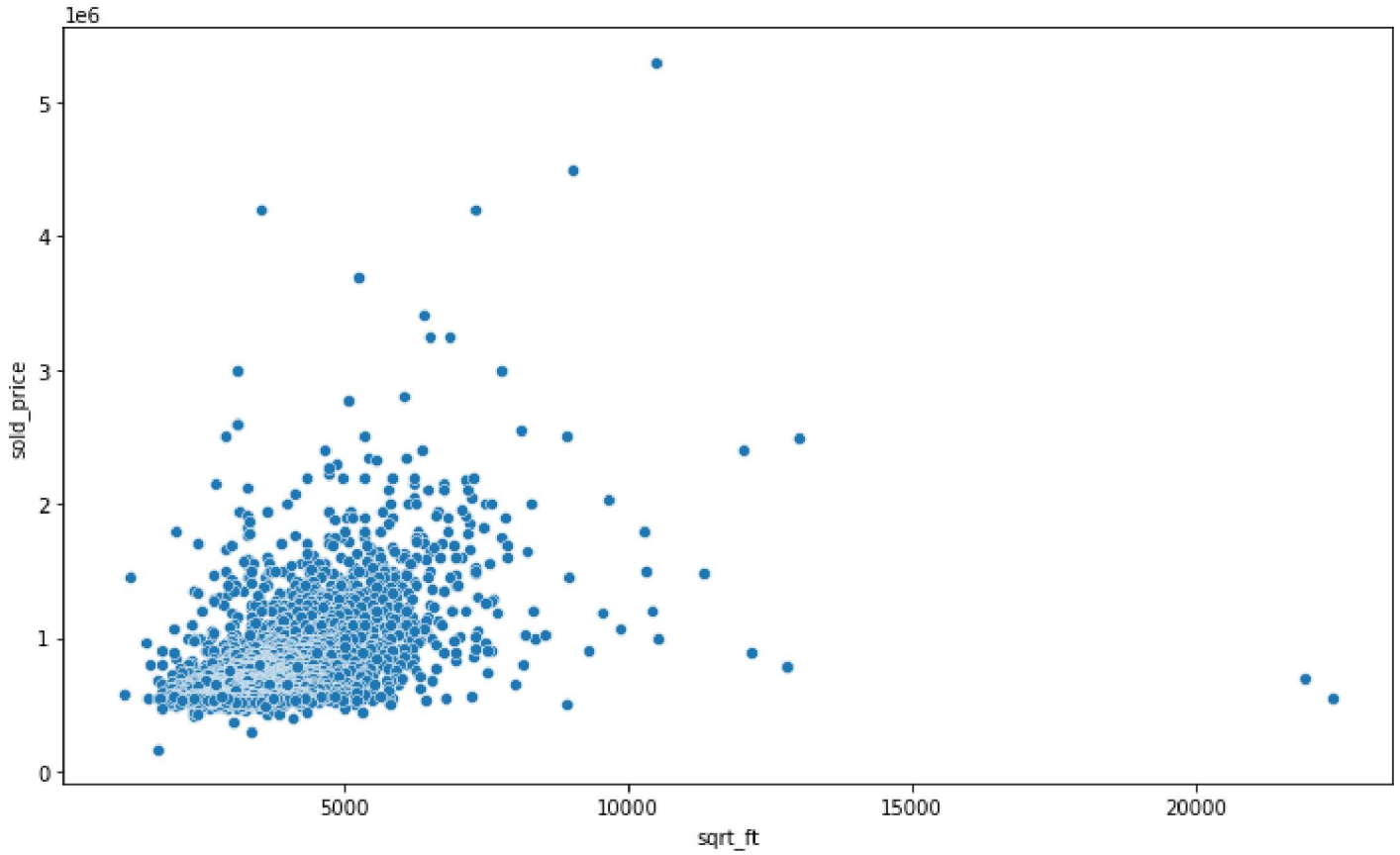
In [61]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df, x='lot_acres', y='sold_price')
```

Out[61]:
```
<AxesSubplot:xlabel='lot_acres', ylabel='sold_price'>
```



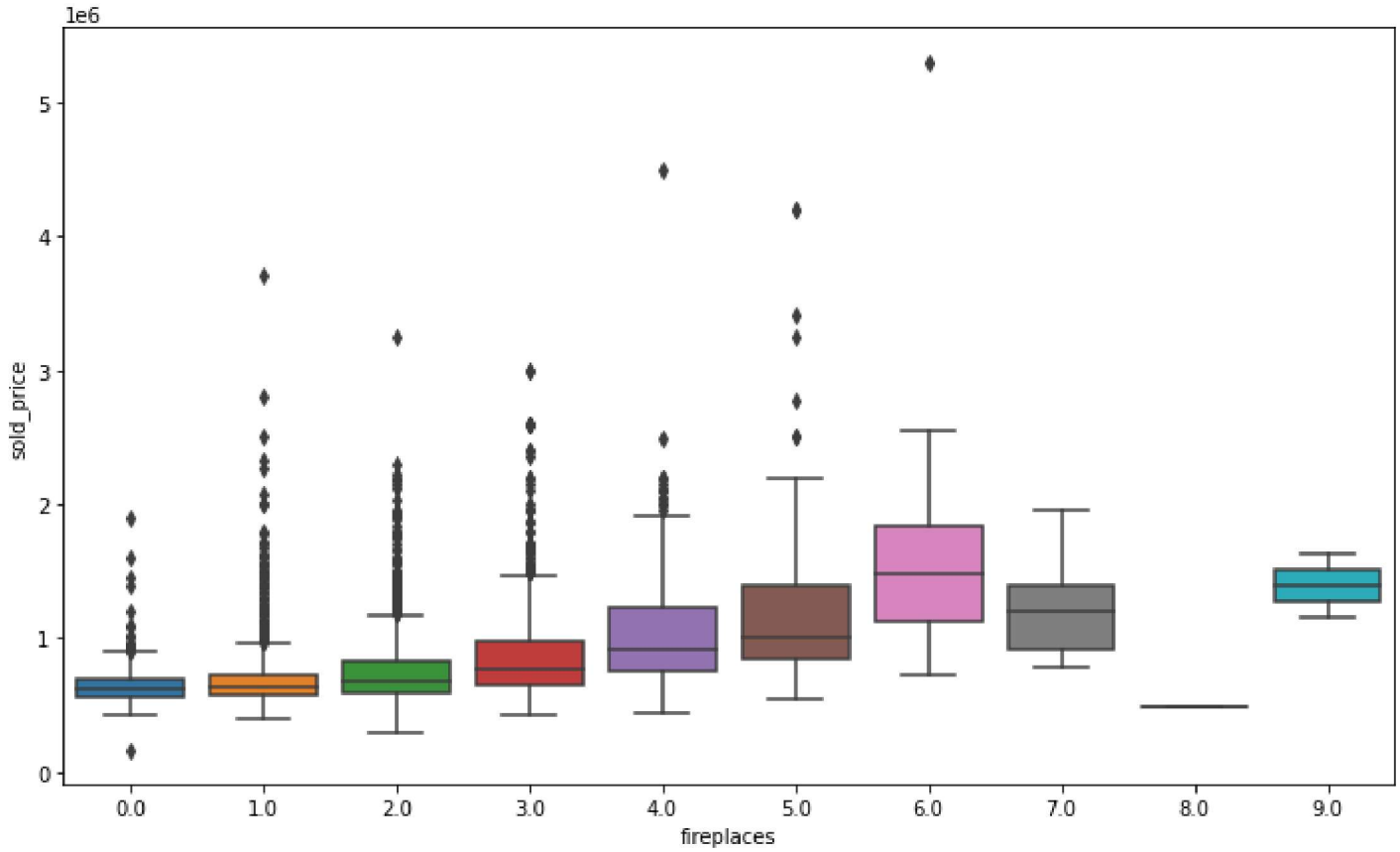
In [62]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df, x='sqrt_ft', y='sold_price')
```

Out[62]: <AxesSubplot:xlabel='sqrt_ft', ylabel='sold_price'>



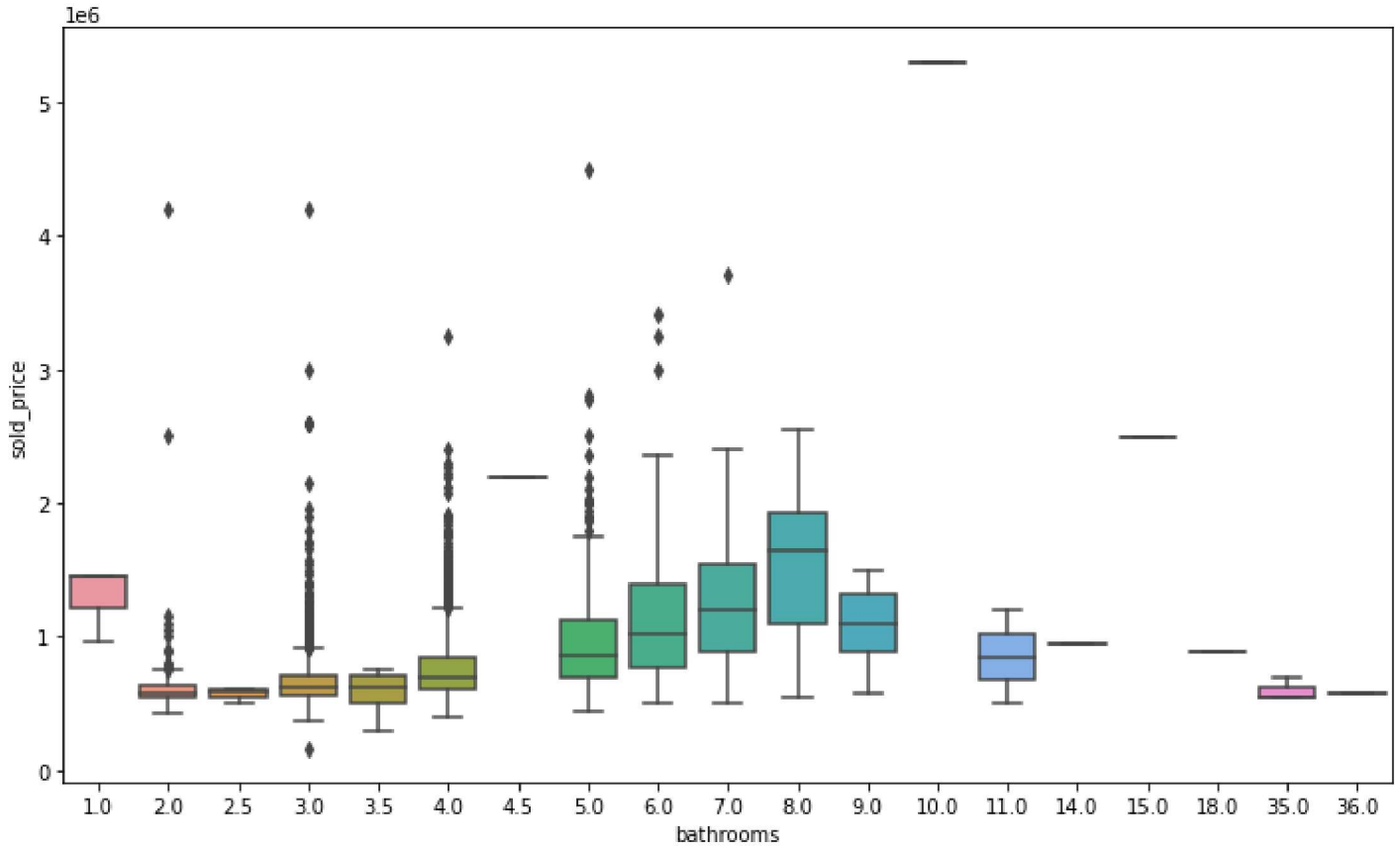
In [63]:
```
plt.figure(figsize = (12,7))
sns.boxplot(x = 'fireplaces', y = 'sold_price', data = df)
```

Out[63]:
```
<AxesSubplot:xlabel='fireplaces', ylabel='sold_price'>
```



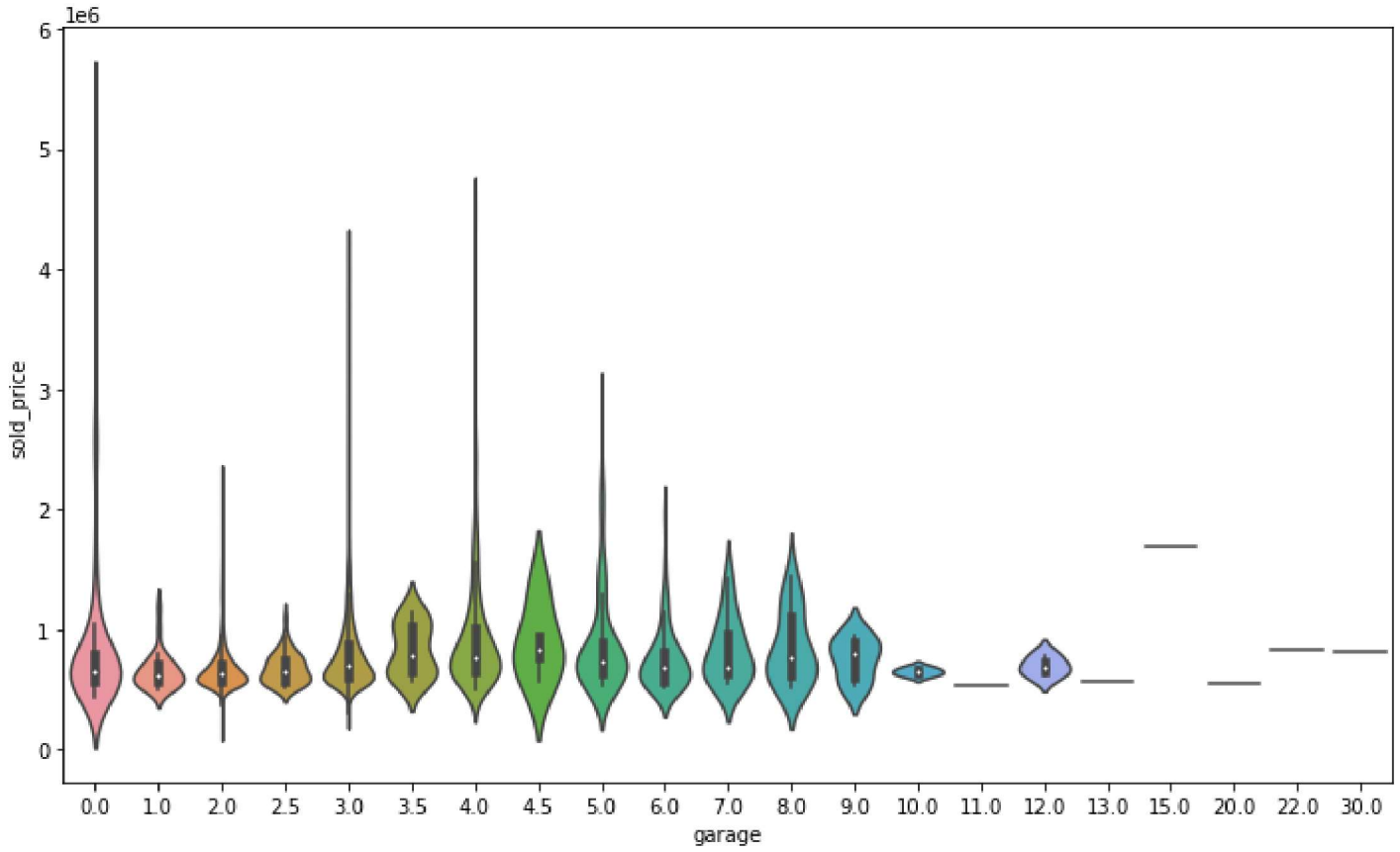
In [64]:
```
plt.figure(figsize = (12,7))
sns.boxplot(x = 'bathrooms', y = 'sold_price', data = df)
```

Out[64]: <AxesSubplot:xlabel='bathrooms', ylabel='sold_price'>



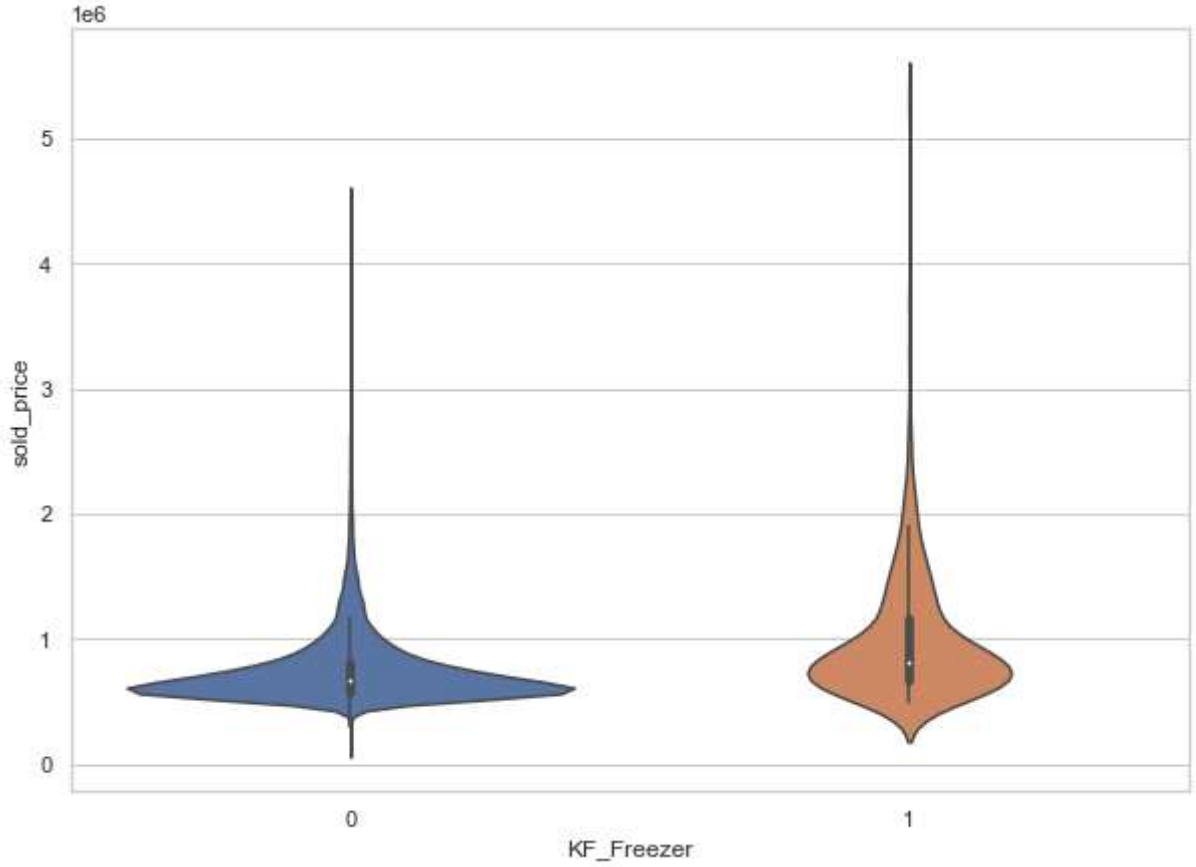
In [65]:
```
plt.figure(figsize = (12,7))
sns.violinplot(x = 'garage', y = 'sold_price', data = df, scale="width")
```

Out[65]: <AxesSubplot:xlabel='garage', ylabel='sold_price'>



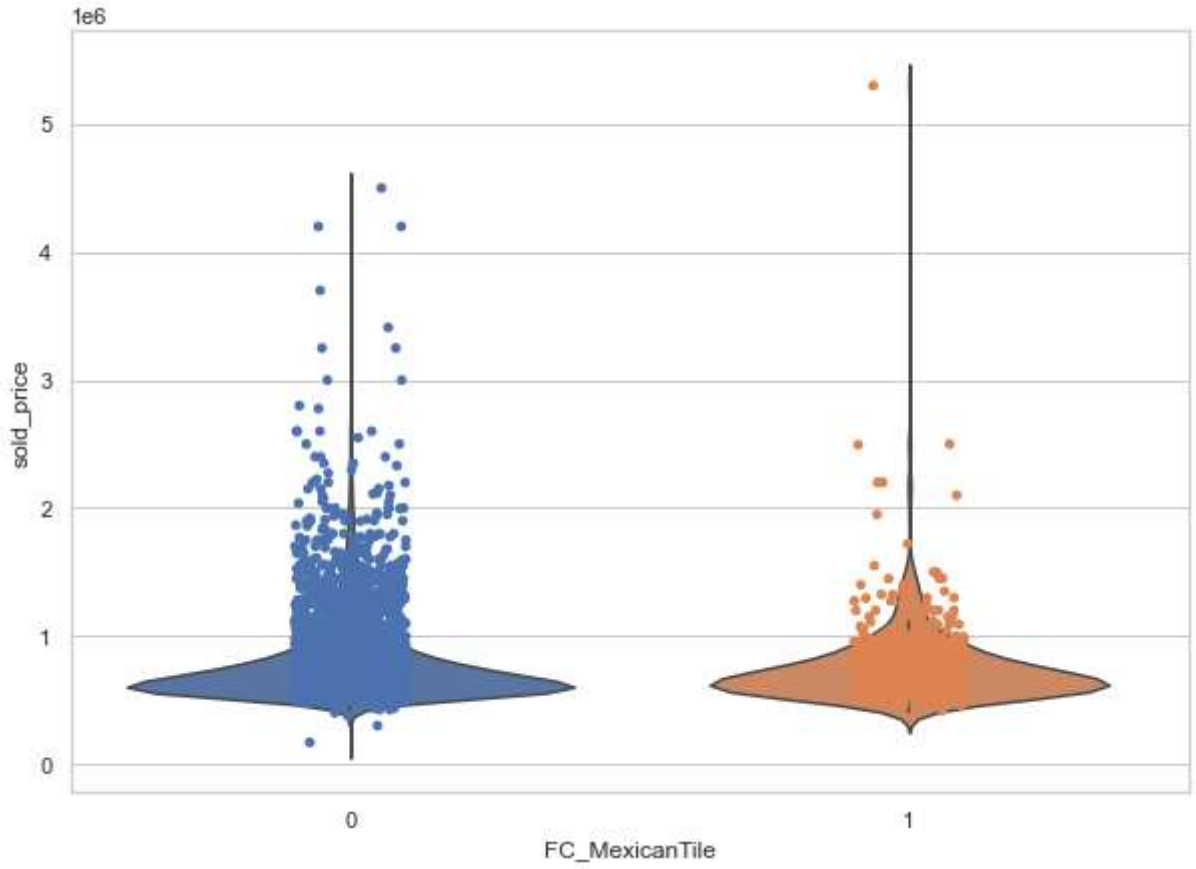
In [76]:
```
plt.figure(figsize = (10,7))
sns.set_theme(style = 'whitegrid')
#sns.stripplot(x = 'KF_Freezer', y = 'sold_price', data = df, jitter=True)
sns.violinplot(x = 'KF_Freezer', y = 'sold_price', data = df)
```

Out[76]: <AxesSubplot:xlabel='KF_Freezer', ylabel='sold_price'>



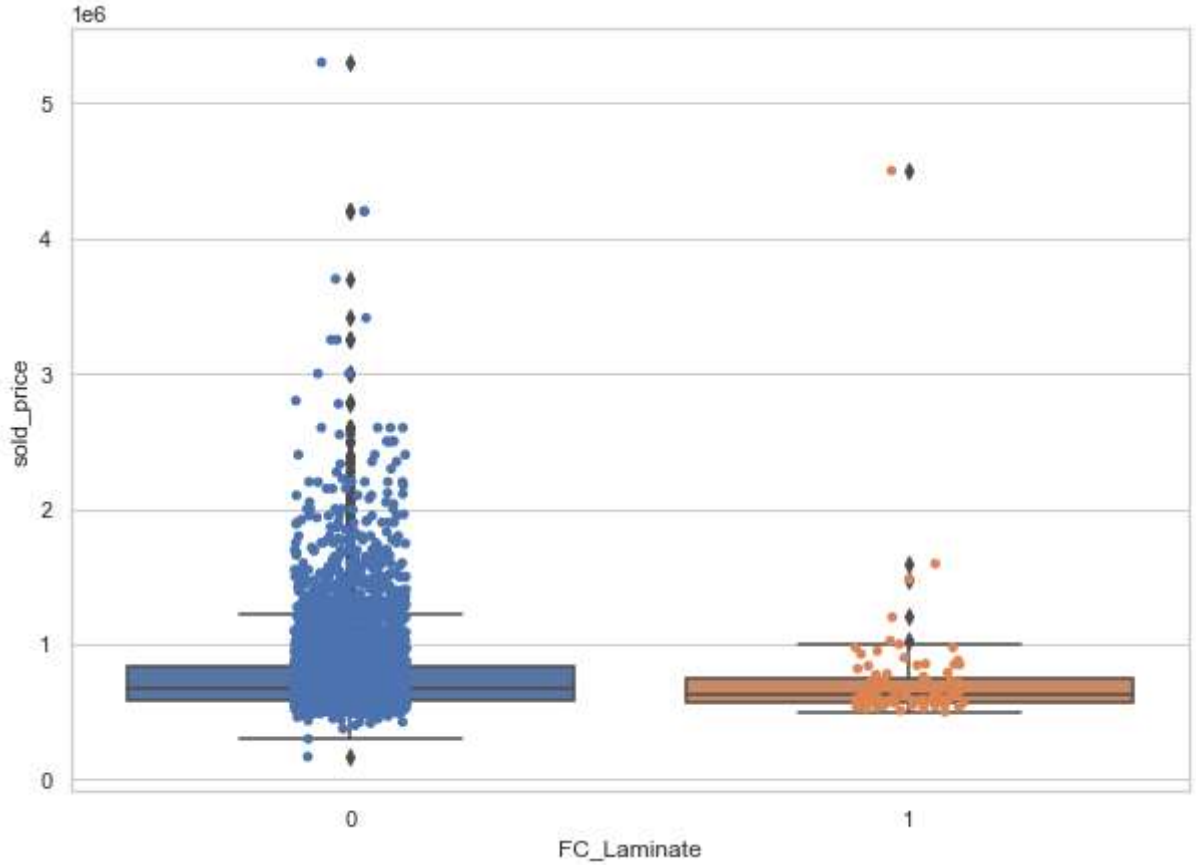
In [67]:
```
plt.figure(figsize = (10,7))
sns.set_theme(style = 'whitegrid')
sns.violinplot(x = 'FC_MexicanTile', y = 'sold_price', data = df)
sns.stripplot(x = 'FC_MexicanTile', y = 'sold_price', data = df, jitter=True)
```

Out[67]: <AxesSubplot:xlabel='FC_MexicanTile', ylabel='sold_price'>



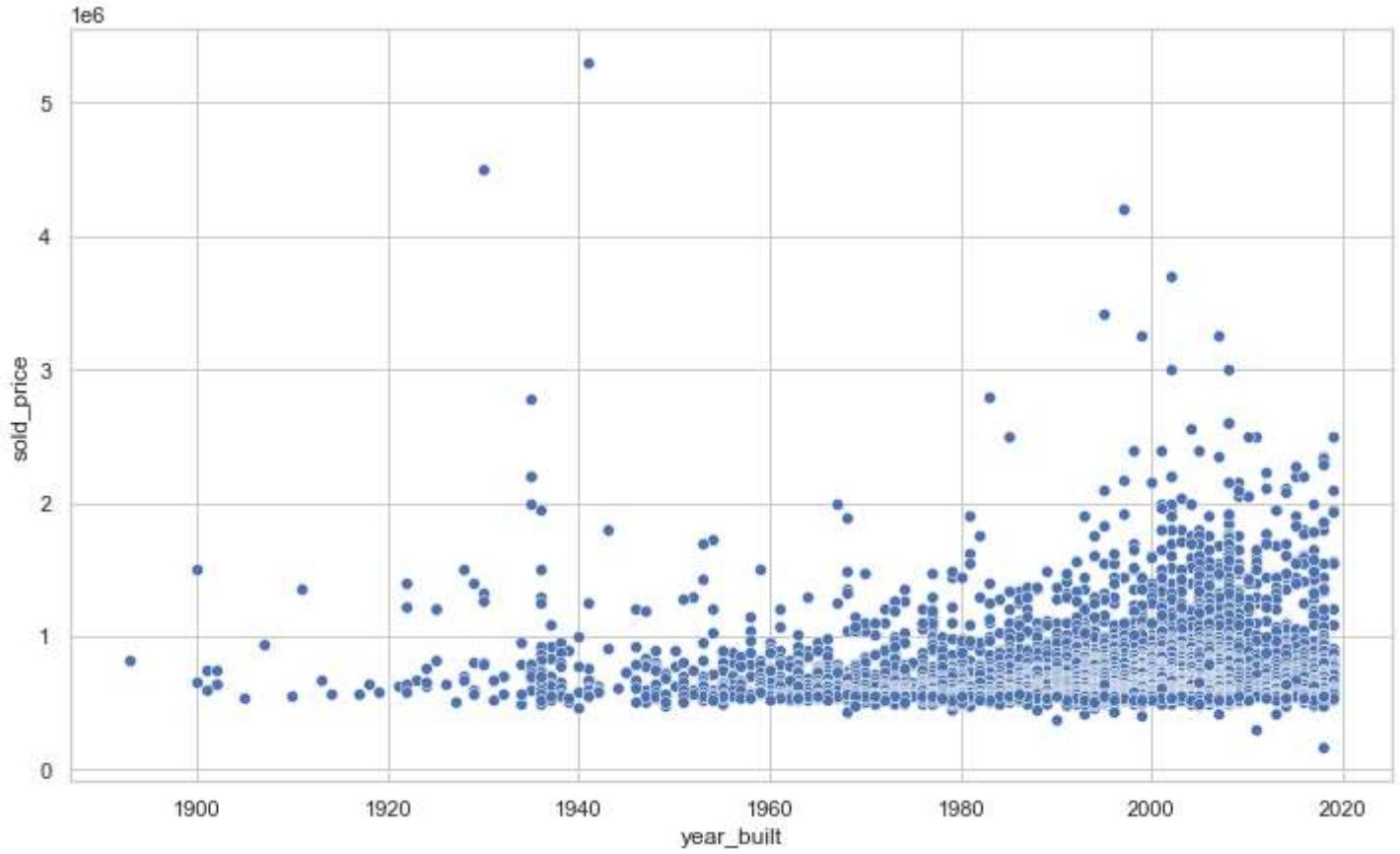
In [68]:
```
plt.figure(figsize = (10,7))
sns.set_theme(style = 'whitegrid')
sns.boxplot(x = 'FC_Laminate', y = 'sold_price', data = df)
sns.stripplot(x = 'FC_Laminate', y = 'sold_price', data = df, jitter=True)
```

Out[68]: <AxesSubplot:xlabel='FC_Laminate', ylabel='sold_price'>



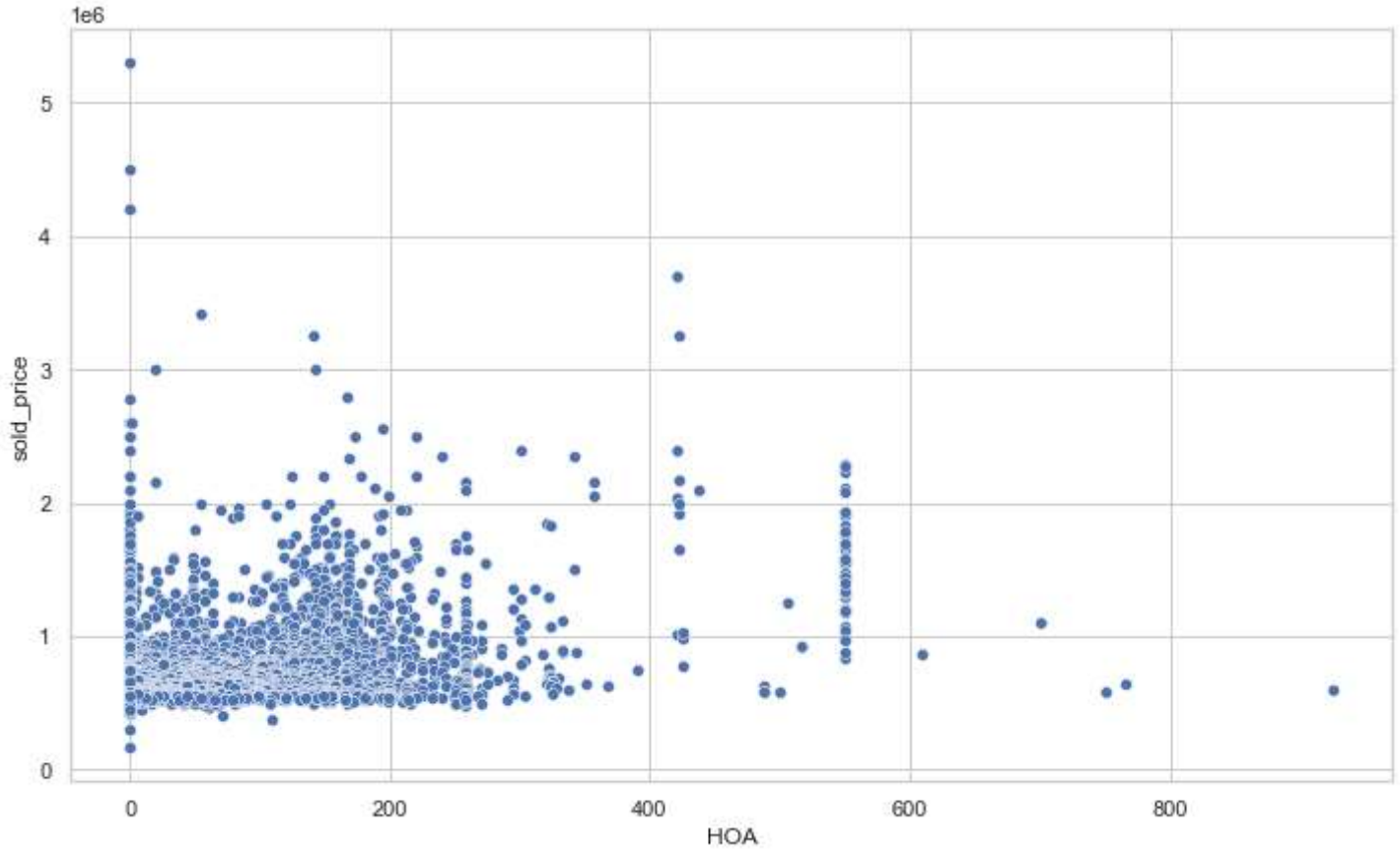
In [69]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df, x='year_built', y='sold_price')
```

Out[69]:
```
<AxesSubplot:xlabel='year_built', ylabel='sold_price'>
```



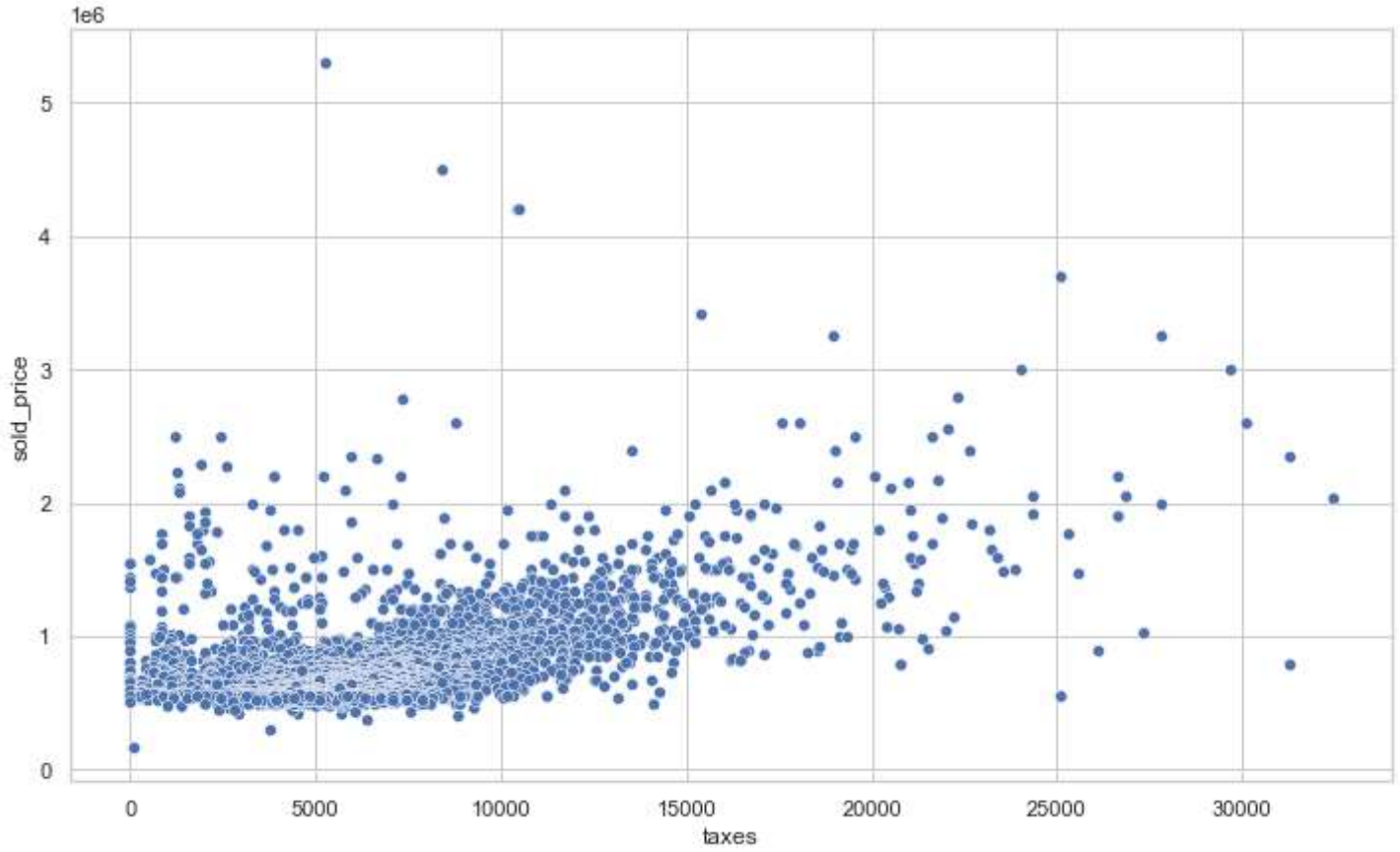
In [70]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df, x='HOA', y='sold_price')
```

Out[70]: <AxesSubplot:xlabel='HOA', ylabel='sold_price'>



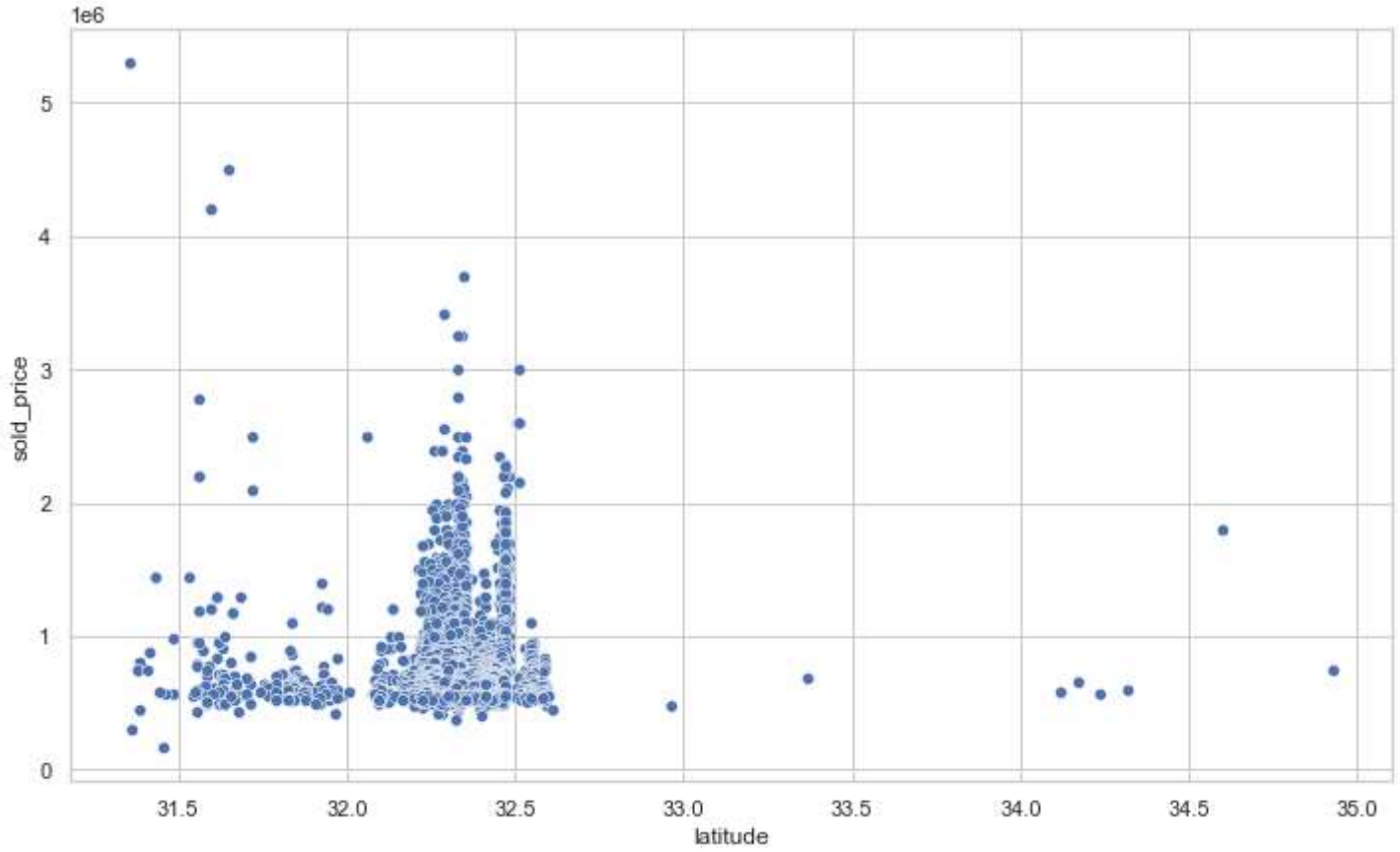
In [71]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df[df['taxes'] < 100000], x='taxes', y='sold_price')
```

Out[71]:
```
<AxesSubplot:xlabel='taxes', ylabel='sold_price'>
```



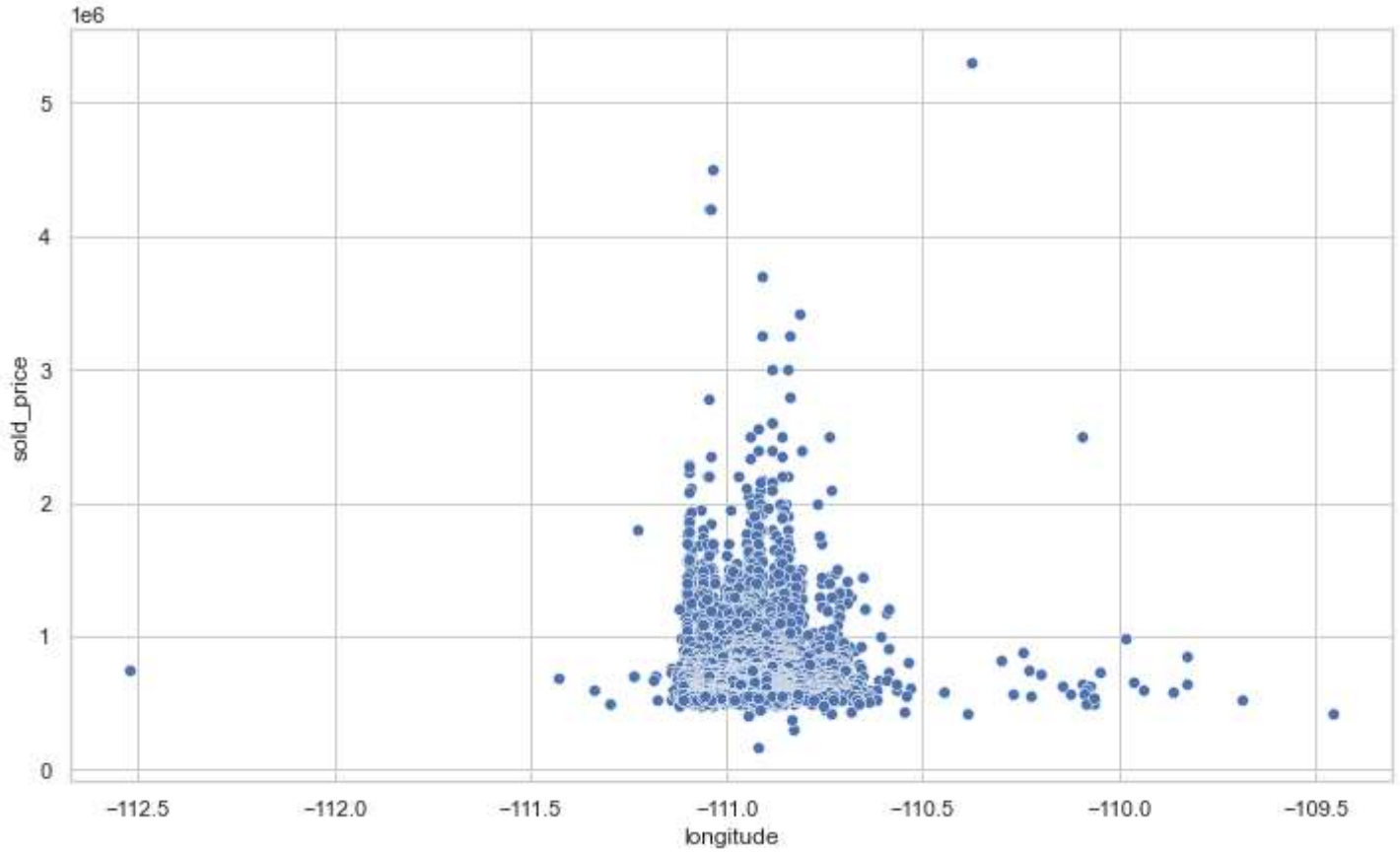
In [72]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df, x='latitude', y='sold_price')
```

Out[72]:
```
<AxesSubplot:xlabel='latitude', ylabel='sold_price'>
```

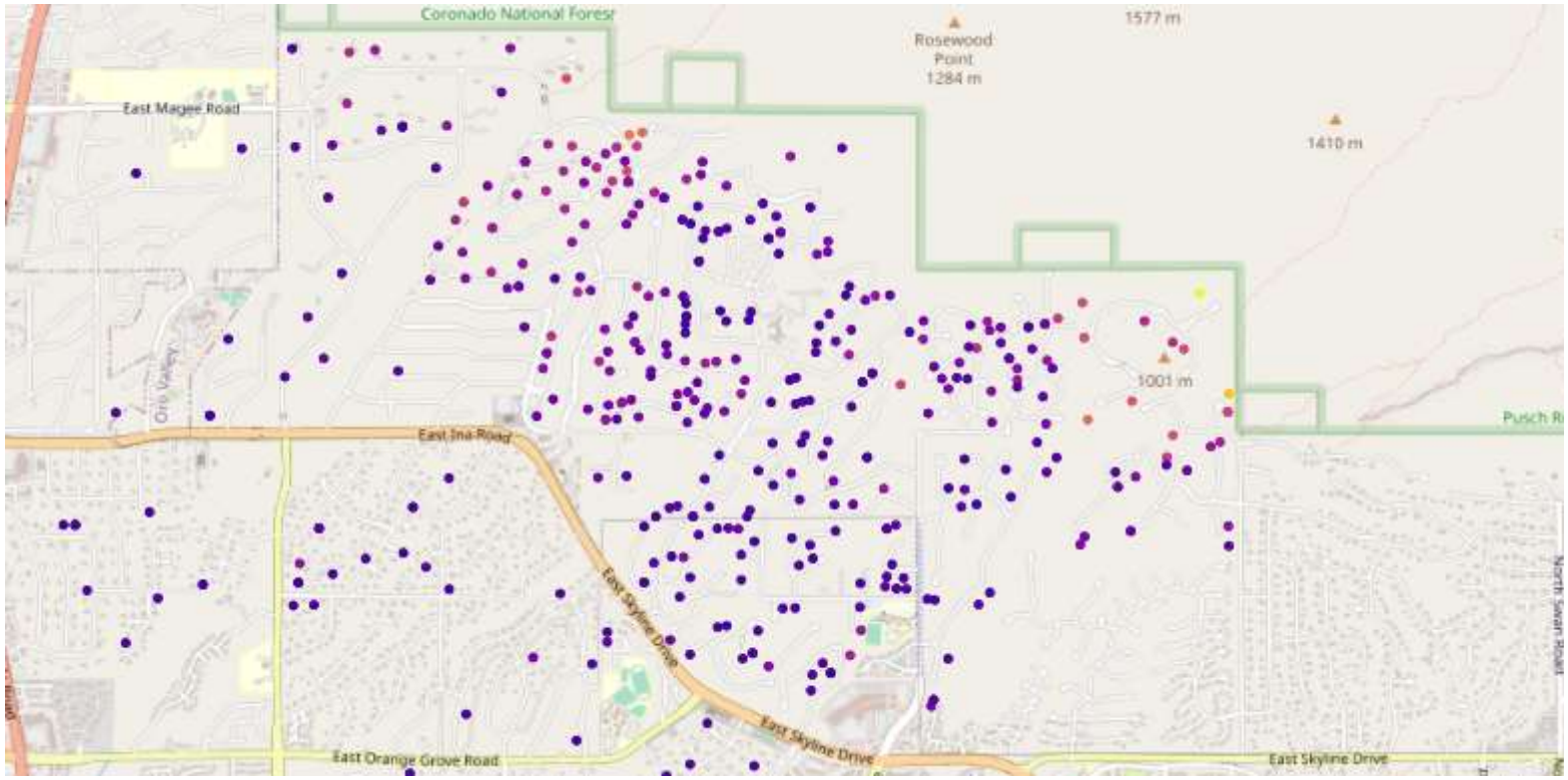


In [80]:
```
plt.figure(figsize = (12,7))
sns.scatterplot(data=df, x='longitude', y='sold_price')
```

Out[80]:
```
<AxesSubplot:xlabel='longitude', ylabel='sold_price'>
```



In [74]:
```
fig = px.scatter_mapbox(df[(df['longitude'] < -110.91) & (df['longitude'] > -111)],
                        lat = 'latitude', lon = 'longitude', color = 'sold_price',
                        center=dict(lon=-111, lat=32.37),
                       zoom = 10, mapbox_style = 'open-street-map')
fig
```



In [75]:
```
df.head(5)
```

Out[75]:

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_built	bedrooms	bathrooms	...	KF_GasRange	KF_C
0	21530491	5300000.0	85637	-11.037820	31.356362	2154.00	5272.00	1941	13	10.0	...	0	
1	21529082	4200000.0	85646	-111.045371	31.594213	1707.00	10422.36	1997	2	2.0	...	0	
2	3054672	4200000.0	85646	-111.040707	31.594844	1707.00	10482.00	1997	2	3.0	...	0	
3	21919321	4500000.0	85646	-111.035925	31.645878	636.67	8418.58	1930	7	5.0	...	0	
4	21306357	3411450.0	85750	-110.813768	32.285162	3.21	15393.00	1995	4	6.0	...	0	

5 rows × 36 columns