King County, WA, U.S.A.

1 Housing Guidance for King County, WA, U.S.A

- Student name: Vi Bui
- Student pace: Part-Time
- Scheduled project review date/time: 10/xx/21
- Instructor name: Claude Fried
- Blog post URL: [https://datasciish.com/ (https://datasciish.com/)](https://datasciish.com/)

1.1 Overview [...]

Client: New WA state home buyers needing consultation on WA real estate market and expectations (price, size, location)

Data, Methodology, and Analysis: King County (WA, U.S.A.) housing data from 2014-2015

Results & Recommendations: After analyzing data and building models assessing relationships between price and square feet; price and bedrooms; and price to zip code, we've modeled the expectations for price range depending on square feet of living space, number of bedrooms, and number of bathrooms

2 Data Exploration, Cleansing, Visualization, and Preparation

Data Exploration
Explore King County, WA, U.S.A. data from years 2014-2015

Data Cleansing
Check for duplicates (none); drop NaN values and unnecessary columns; continuously clean data as necessary

Data Visualization
Use visualizations to explore the data and determine how to further refine the dataset in order to prepare for modeling

Data Preparation

2.1 Data Exploration and Cleansing

Import data and all packages needed for data exploration and modeling

In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

import scipy.stats as stats

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

import os
import warnings
```

executed in 1.48s, finished 02:56:47 2021-10-25

Explore: columns, shape, info

In [2]:
```
df = pd.read_csv('data/kc_house_data.csv', index_col=0)
df.head()
```

executed in 56ms, finished 02:56:47 2021-10-25

Out[2]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vie
7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	C
6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	C
5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	C
2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	C
1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	C

In [3]:
```
# Explore number of entries; which columns have missing data; and data type

df.info()
```

executed in 9ms, finished 02:56:47 2021-10-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 1   price          21597 non-null  float64
 2   bedrooms       21597 non-null  int64  
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64  
 5   sqft_lot       21597 non-null  int64  
 6   floors         21597 non-null  float64
 7   waterfront     19221 non-null  float64
 8   view           21534 non-null  float64
 9   condition      21597 non-null  int64  
 10  grade          21597 non-null  int64  
 11  sqft_above     21597 non-null  int64  
 12  sqft_basement  21597 non-null  object 
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   17755 non-null  float64
 15  zipcode        21597 non-null  int64  
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(10), object(2)
memory usage: 3.5+ MB
```

In [4]:
```
# Check for duplicates

df.duplicated(keep='first').sum()
```
executed in 15ms, finished 02:56:47 2021-10-25

Out[4]:
```
0
```

In [5]:
```
# Check for NaN values

df.isna().sum()

# Columns and number of respective NaN values
# waterfront       2376
# view               63
# yr_renovated     3842
```
executed in 6ms, finished 02:56:47 2021-10-25

Out[5]:
```
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront      2376
view              63
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated    3842
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
dtype: int64
```

In [6]:
```
# Explore columns

df.columns
```
executed in 2ms, finished 02:56:47 2021-10-25

Out[6]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lo
t',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abov
e',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lo
ng',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

2.1.1 Understand Column Names and Descriptions for King

County's Data Set

- **id** - unique identified for a house
- **dateDate** - house was sold
- **pricePrice** - is prediction target
- **bedroomsNumber** - of Bedrooms/House
- **bathroomsNumber** - of bathrooms/bedrooms
- **sqft_livingsquare** - footage of the home
- **sqft_lotsquare** - footage of the lot
- **floorsTotal** - floors (levels) in house
- **waterfront** - House which has a view to a waterfront
- **view** - Has been viewed
- **condition** - How good the condition is (Overall)
- **grade** - overall grade given to the housing unit, based on King County grading system
- **sqft_above** - square footage of house apart from basement
- **sqft_basement** - square footage of the basement
- **yr_built** - Built Year
- **yr_renovated** - Year when house was renovated
- **zipcode** - zip
- **lat** - Latitude coordinate
- **long** - Longitude coordinate
- **sqft_living15** - The square footage of interior housing living space for the nearest 15 neighbors
- **sqft_lot15** - The square footage of the land lots of the nearest 15 neighbors

In [7]:
```
# Calculate the percentage of NaN

df['waterfront'].value_counts()

# Ask Claude how to do this "smarter": (19075/(19075+146))
# (2376+19075)/(2376+19075+146)
```
executed in 4ms, finished 02:56:47 2021-10-25

Out[7]:
```
0.0    19075
1.0      146
Name: waterfront, dtype: int64
```

Observations after exploring waterfront data:

- 99.2% of houses (146 out of 19,221) do not have a waterfront view
- With 2376 entries with NaN values, imputing the NaN values to 0 makes no material difference
- Clean data: impute waterfront NaN values to 0 (represents no waterfront view)
- Resulting data: 99.3% of houses (21,451 out of 21,597) do not have a waterfront view

In [8]:
```
# Impute waterfront NaN values to 0

df['waterfront'] = df['waterfront'].fillna(0)
df['waterfront'].value_counts()
```
executed in 5ms, finished 02:56:47 2021-10-25

Out[8]:
```
0.0    21451
1.0      146
Name: waterfront, dtype: int64
```

In [9]:
```
# Double check for NaN values left

df['waterfront'].isna().sum()
```
executed in 3ms, finished 02:56:47 2021-10-25

Out[9]:
```
0
```

In [10]:
```
# Continue exploring other data that needs to be cleansed

df.info()
```
executed in 10ms, finished 02:56:47 2021-10-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 1   price          21597 non-null  float64
 2   bedrooms       21597 non-null  int64  
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64  
 5   sqft_lot       21597 non-null  int64  
 6   floors         21597 non-null  float64
 7   waterfront     21597 non-null  float64
 8   view           21534 non-null  float64
 9   condition      21597 non-null  int64  
 10  grade          21597 non-null  int64  
 11  sqft_above     21597 non-null  int64  
 12  sqft_basement  21597 non-null  object 
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   17755 non-null  float64
 15  zipcode        21597 non-null  int64  
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(10), object(2)
memory usage: 3.5+ MB
```

```
Write raw LaTeX or other formats here, for use with nbconvert. It will not
be rendered in the notebook. When passing through nbconvert, a Raw Cell's
content is added to the output unmodified.
```

In [11]:
```
df['yr_renovated'].value_counts()
```
executed in 6ms, finished 02:56:47 2021-10-25

Out[11]:
```
0.0       17011
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

'yr_renovated' data needs to be cleansed. Observations about 'yr_renovated':

- 'yr_renovated' has 3842 NaN values
- About the data: if house has been renovated, the year is entered. If not, 0 has been entered
- 95.8% of current data set (17,011 of 17,755 houses) have not been renovated
- Imputing the 3842 NaN values to 0 (not renovated) does not make a substantial difference
- Resulting data: 96.6% of new data set (20,853 of 21,597 houses) have not been renovated

In [12]:
```
df['yr_renovated'] = df['yr_renovated'].fillna(0)
df['yr_renovated'].value_counts()
```
executed in 5ms, finished 02:56:47 2021-10-25

Out[12]:
```
0.0       20853
2014.0       73
2003.0       31
2013.0       31
2007.0       30
          ...
1946.0        1
1959.0        1
1971.0        1
1951.0        1
1954.0        1
Name: yr_renovated, Length: 70, dtype: int64
```

In [13]:
```
# ask Claude how you would get the sum of the value counts
# df['yr_renovated'].value_counts('0')
20853/21597
```
executed in 3ms, finished 02:56:47 2021-10-25

Out[13]:
```
0.9655507709404084
```

In [14]:
```
df.head()
```
executed in 13ms, finished 02:56:47 2021-10-25

Out[14]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vie
id									
7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	C
6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	C
5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	C
2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	C
1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	C

'sqft_basement' data needs to be cleansed. Observations about 'sqft_basement':

- Ran into an error with 'sqft_basement' data
- Found there are 454 entries with '?' symbols in the data
- 60.7% of current data set (12,826 of 21,143 houses) have 0 as entered for sqft_basement
- Imputing the 454 '?' entries to 0 does not make a substantial difference
- Resulting data: 61.5% of new data set (13,280 of 21,597 houses) have 0 sqft_basement

In [15]:
```
# Check how many entries for 'sqft_basement' are '?'
# Ask Claude how to view the entire data set for sqft_basement

df['sqft_basement'].value_counts()
```
executed in 6ms, finished 02:56:47 2021-10-25

Out[15]:
```
0.0       12826
?           454
600.0       217
500.0       209
700.0       208
          ...
2180.0        1
2490.0        1
1880.0        1
508.0         1
143.0         1
Name: sqft_basement, Length: 304, dtype: int64
```

In [16]:
```
# Impute 454 '?' entries to 0 values
# Transform data type from object to float

df['sqft_basement'] = df['sqft_basement'].apply(lambda x: 0 if x == '?' els
```
executed in 7ms, finished 02:56:47 2021-10-25

In [17]:
```
df.info()
```
executed in 10ms, finished 02:56:47 2021-10-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 1   price          21597 non-null  float64
 2   bedrooms       21597 non-null  int64  
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64  
 5   sqft_lot       21597 non-null  int64  
 6   floors         21597 non-null  float64
 7   waterfront     21597 non-null  float64
 8   view           21534 non-null  float64
 9   condition      21597 non-null  int64  
 10  grade          21597 non-null  int64  
 11  sqft_above     21597 non-null  int64  
 12  sqft_basement  21597 non-null  float64
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   21597 non-null  float64
 15  zipcode        21597 non-null  int64  
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(9), int64(10), object(1)
memory usage: 3.5+ MB
```

Continue cleaning data/transform data types:

- Transform data types
- Most importantly, convert zipcode from integer to string

In [18]:
```
# yr_renovated from float to integer (preference)
# zipcode from integer to string

df['yr_renovated'] = (df['yr_renovated'].astype(int))
df['zipcode'] = (df['zipcode'].astype(str))
```
executed in 18ms, finished 02:56:47 2021-10-25

In [19]:
```
df.info()
```
executed in 10ms, finished 02:56:47 2021-10-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           21597 non-null  object 
 1   price          21597 non-null  float64
 2   bedrooms       21597 non-null  int64  
 3   bathrooms      21597 non-null  float64
 4   sqft_living    21597 non-null  int64  
 5   sqft_lot       21597 non-null  int64  
 6   floors         21597 non-null  float64
 7   waterfront     21597 non-null  float64
 8   view           21534 non-null  float64
 9   condition      21597 non-null  int64  
 10  grade          21597 non-null  int64  
 11  sqft_above     21597 non-null  int64  
 12  sqft_basement  21597 non-null  float64
 13  yr_built       21597 non-null  int64  
 14  yr_renovated   21597 non-null  int64  
 15  zipcode        21597 non-null  object 
 16  lat            21597 non-null  float64
 17  long           21597 non-null  float64
 18  sqft_living15  21597 non-null  int64  
 19  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(10), object(2)
memory usage: 3.5+ MB
```

2.2 Create New Features

1. Year Sold (from date column)
2. Renovated (make renovated a binary value: renovated = 1; not renovated = 0)
3. Basement Present (make basement a binary value: renovated = 1; not renovated = 0)
4. Actual Age of Property (year sold - year built)
5. Bathrooms Per Bedroom (bathrooms/bedrooms)
6. Square Feet Living to Square Foot Lot (sqft_living/sqft_lot)

In [20]:
```
# Create new features

df['yr_sold'] = (df['date'].str[-4:].astype(int))
df['renovated'] = np.where(df['yr_renovated']!=0, 1,0)
df['basement_present'] = np.where(df['sqft_basement']!=0, 1,0)
df['actual_age_of_property'] = df['yr_sold']-df['yr_built']
df['bathrooms_per_bedroom'] = df['bathrooms']/df['bedrooms']
df['sqft_living_to_sqft_lot'] = df['sqft_living']/df['sqft_lot']
df.head()
```

executed in 33ms, finished 02:56:47 2021-10-25

Out[20]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	vie
id									
7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	0.0	C
6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0	C
5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0	C
2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0	C
1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0	C

5 rows × 26 columns

In [21]:
```
# Check: data types
# Check: all value counts match

df.info()
```
executed in 15ms, finished 02:56:47 2021-10-25

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21597 entries, 7129300520 to 1523300157
Data columns (total 26 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   date                       21597 non-null  object 
 1   price                      21597 non-null  float64
 2   bedrooms                   21597 non-null  int64  
 3   bathrooms                  21597 non-null  float64
 4   sqft_living                21597 non-null  int64  
 5   sqft_lot                   21597 non-null  int64  
 6   floors                     21597 non-null  float64
 7   waterfront                 21597 non-null  float64
 8   view                       21534 non-null  float64
 9   condition                  21597 non-null  int64  
 10  grade                      21597 non-null  int64  
 11  sqft_above                 21597 non-null  int64  
 12  sqft_basement              21597 non-null  float64
 13  yr_built                   21597 non-null  int64  
 14  yr_renovated               21597 non-null  int64  
 15  zipcode                    21597 non-null  object 
 16  lat                        21597 non-null  float64
 17  long                       21597 non-null  float64
 18  sqft_living15              21597 non-null  int64  
 19  sqft_lot15                 21597 non-null  int64  
 20  yr_sold                    21597 non-null  int64  
 21  renovated                  21597 non-null  int64  
 22  basement_present           21597 non-null  int64  
 23  actual_age_of_property     21597 non-null  int64  
 24  bathrooms_per_bedroom      21597 non-null  float64
 25  sqft_living_to_sqft_lot    21597 non-null  float64
dtypes: float64(10), int64(14), object(2)
memory usage: 4.4+ MB
```

In [22]:
```
df.columns
```
executed in 3ms, finished 02:56:47 2021-10-25

Out[22]:
```
Index(['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lo
t',
       'floors', 'waterfront', 'view', 'condition', 'grade', 'sqft_abov
e',
       'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode', 'lat', 'lo
ng',
       'sqft_living15', 'sqft_lot15', 'yr_sold', 'renovated',
       'basement_present', 'actual_age_of_property', 'bathrooms_per_bedro
om',
       'sqft_living_to_sqft_lot'],
      dtype='object')
```

In [23]:
```
# Explore correlation for numerical values with .corr()

df.corr()
```
executed in 43ms, finished 02:56:47 2021-10-25

Out[23]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
price	1.000000	0.308787	0.525906	0.701917	0.089876	0.256804	0.2643
bedrooms	0.308787	1.000000	0.514508	0.578212	0.032471	0.177944	-0.0021
bathrooms	0.525906	0.514508	1.000000	0.755758	0.088373	0.502582	0.0636
sqft_living	0.701917	0.578212	0.755758	1.000000	0.173453	0.353953	0.1046
sqft_lot	0.089876	0.032471	0.088373	0.173453	1.000000	-0.004814	0.0214
floors	0.256804	0.177944	0.502582	0.353953	-0.004814	1.000000	0.0207
waterfront	0.264306	-0.002127	0.063629	0.104637	0.021459	0.020797	1.0000
view	0.395734	0.078523	0.186451	0.282532	0.075298	0.028436	0.3820
condition	0.036056	0.026496	-0.126479	-0.059445	-0.008830	-0.264075	0.0166
grade	0.667951	0.356563	0.665838	0.762779	0.114731	0.458794	0.0828
sqft_above	0.605368	0.479386	0.686668	0.876448	0.184139	0.523989	0.0717
sqft_basement	0.321108	0.297229	0.278485	0.428660	0.015031	-0.241866	0.0830
yr_built	0.053953	0.155670	0.507173	0.318152	0.052946	0.489193	-0.0244
yr_renovated	0.117855	0.017900	0.047177	0.051060	0.004979	0.003793	0.0739
lat	0.306692	-0.009951	0.024280	0.052155	-0.085514	0.049239	-0.0121
long	0.022036	0.132054	0.224903	0.241214	0.230227	0.125943	-0.0376
sqft_living15	0.585241	0.393406	0.569884	0.756402	0.144763	0.280102	0.0838
sqft_lot15	0.082845	0.030690	0.088303	0.184342	0.718204	-0.010722	0.0306
yr_sold	0.003727	-0.009949	-0.026577	-0.029014	0.005628	-0.022352	-0.0050
renovated	0.117543	0.017635	0.046742	0.050829	0.005091	0.003713	0.0742
basement_present	0.178264	0.158412	0.159863	0.201198	-0.034889	-0.252465	0.0392
actual_age_of_property	-0.053890	-0.155817	-0.507561	-0.318592	-0.052853	-0.489514	0.0244
bathrooms_per_bedroom	0.281227	-0.236129	0.652668	0.310690	0.063306	0.421169	0.0737
sqft_living_to_sqft_lot	0.123063	0.026798	0.287015	0.076988	-0.252601	0.556700	-0.0298

24 rows × 24 columns

In [24]:
```
# Explore descriptive statistics with .describe()
# Summarizes central tendency (mean), dispersion and shape of a dataset's d

df.describe()
```
executed in 65ms, finished 02:56:47 2021-10-25

Out[24]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	0.
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	0.
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	0.
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.

8 rows × 24 columns

In [25]:
```
# Explore distribution (value_counts) of bedroom data

df['bedrooms'].value_counts()
```
executed in 4ms, finished 02:56:47 2021-10-25

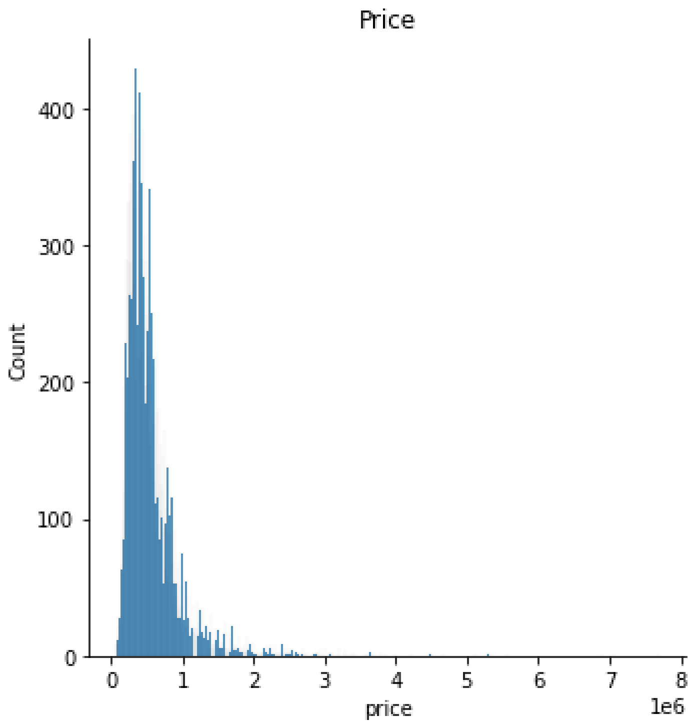
Out[25]:
```
3     9824
4     6882
2     2760
5     1601
6      272
1      196
7       38
8       13
9        6
10       3
11       1
33       1
Name: bedrooms, dtype: int64
```

2.3 Data Visualization

In [26]:
```
plt.figure(figsize=(12,8))
sns.displot(df['price'],bins=1000)
plt.title('Price')
plt.show();
```

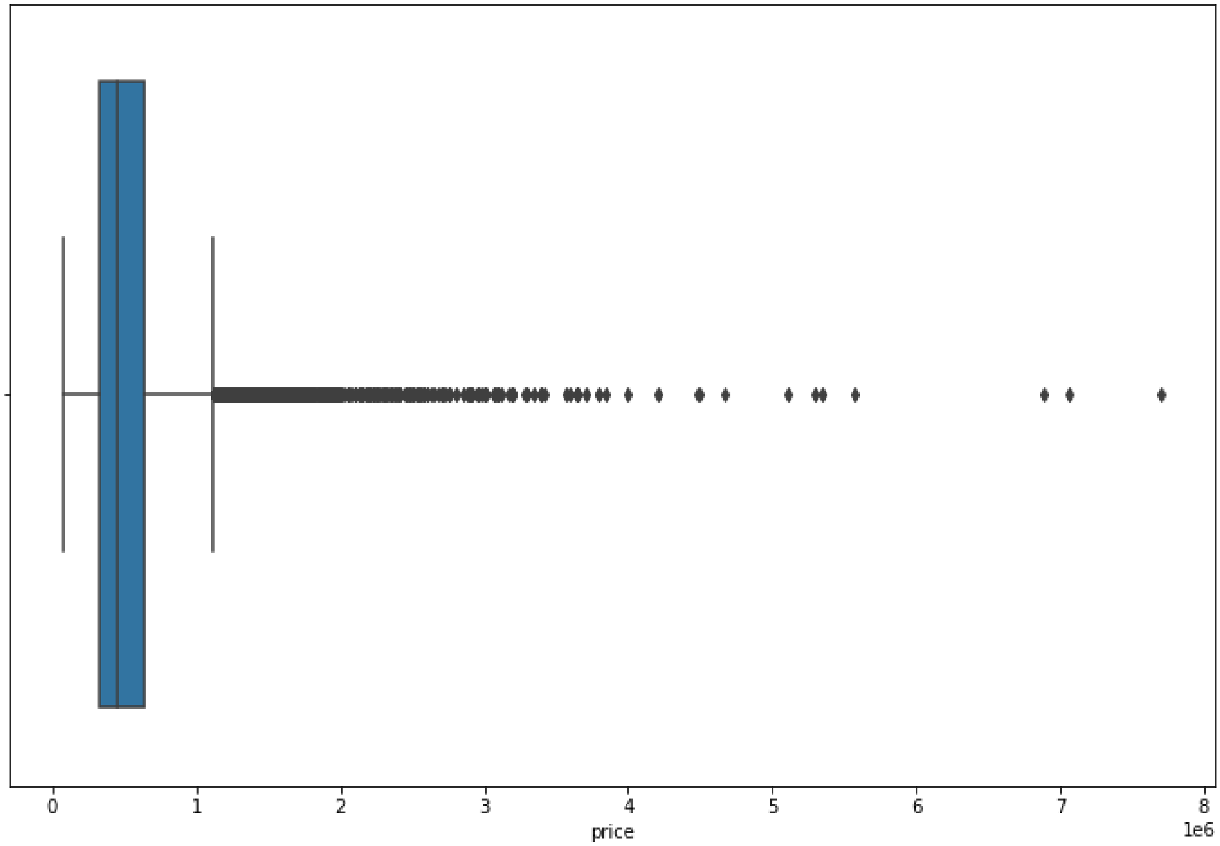
executed in 1.33s, finished 02:56:49 2021-10-25

```
<Figure size 864x576 with 0 Axes>
```



In [27]:
```
fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(x='price', data=df, ax=ax)
```
executed in 176ms, finished 02:56:49 2021-10-25

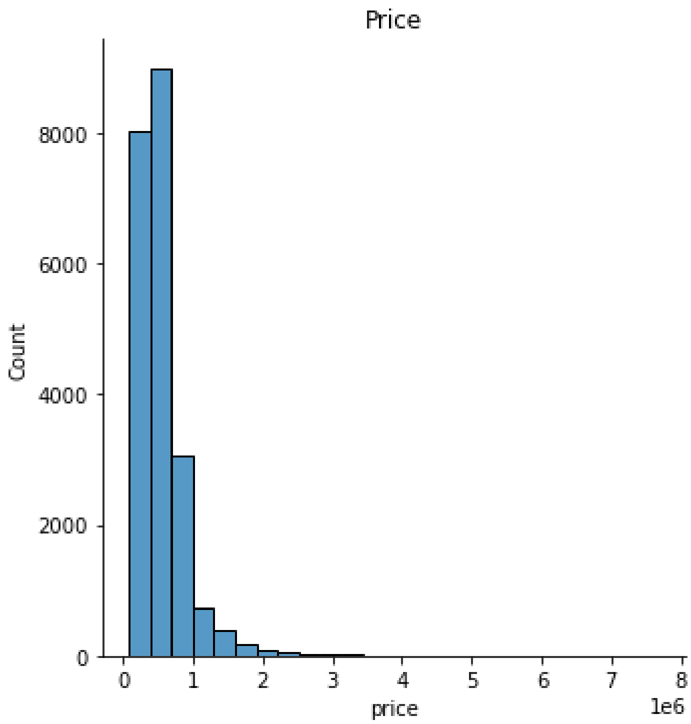
Out[27]: <AxesSubplot:xlabel='price'>



In [28]:
```
plt.figure(figsize=(12,8))
sns.displot(df['price'],bins=25)
plt.title('Price')
plt.show();
```

executed in 208ms, finished 02:56:49 2021-10-25

```
<Figure size 864x576 with 0 Axes>
```



In [29]:
```
df.describe()
```
executed in 72ms, finished 02:56:49 2021-10-25

Out[29]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.000000	21597.
mean	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.494096	0.
std	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.539683	0.
min	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.000000	0.
25%	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.000000	0.
50%	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.
75%	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.000000	0.
max	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.

8 rows × 24 columns

Narrow price range to $175,000-$650,000

In [30]:
```
# Ask Claude how to find where the "majority" of thd prices fall

df = df[df['price'].between(175_000,650_000)]
```
executed in 6ms, finished 02:56:49 2021-10-25

In [31]:
```
df.info()
```
executed in 9ms, finished 02:56:49 2021-10-25

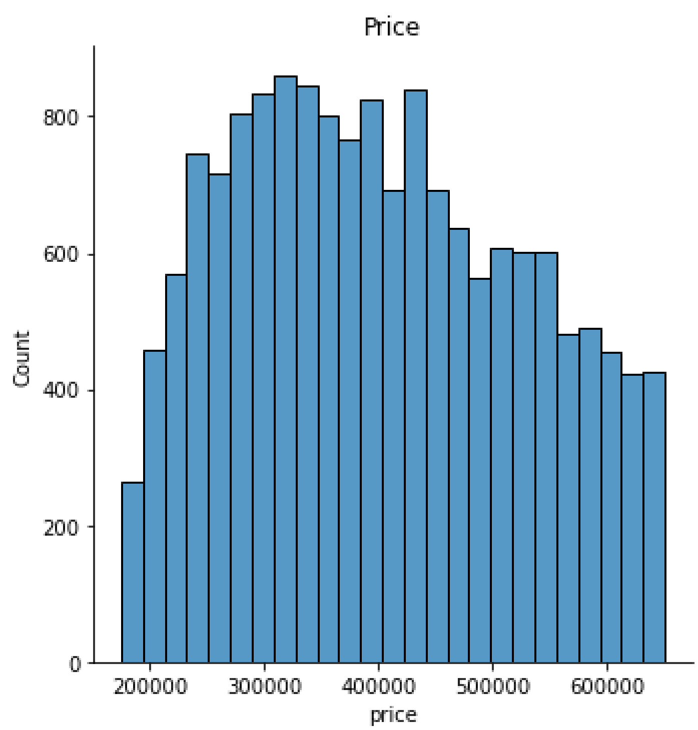
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 15993 entries, 7129300520 to 1523300157
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   date                     15993 non-null  object
 1   price                    15993 non-null  float64
 2   bedrooms                 15993 non-null  int64
 3   bathrooms                15993 non-null  float64
 4   sqft_living              15993 non-null  int64
 5   sqft_lot                 15993 non-null  int64
 6   floors                   15993 non-null  float64
 7   waterfront               15993 non-null  float64
 8   view                     15949 non-null  float64
 9   condition                15993 non-null  int64
 10  grade                    15993 non-null  int64
 11  sqft_above               15993 non-null  int64
 12  sqft_basement            15993 non-null  float64
 13  yr_built                 15993 non-null  int64
```

In [32]:
```
# Percentage of data that will be used

15_993/21_597
```
executed in 2ms, finished 02:56:49 2021-10-25

Out[32]: 0.7405195165995277

In [33]:
```
plt.figure(figsize=(12,8))
sns.displot(df['price'],bins=25)
plt.title('Price')
plt.show();
```
executed in 187ms, finished 02:56:49 2021-10-25

<Figure size 864x576 with 0 Axes>



In [34]:
```
# Explore the data - specifically at bedrooms, bathrooms, and sqft_living

df.describe()
```
executed in 65ms, finished 02:56:49 2021-10-25

Out[34]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	15993.000000	15993.000000	15993.000000	15993.000000	1.599300e+04	15993.000000	1599
mean	400987.591009	3.244982	1.953667	1800.124742	1.317614e+04	1.431533	(
std	123893.841894	0.886582	0.660056	631.746037	3.310214e+04	0.537112	(
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.000000	(
25%	299950.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.000000	(
50%	392000.000000	3.000000	2.000000	1720.000000	7.439000e+03	1.000000	(
75%	499990.000000	4.000000	2.500000	2180.000000	9.968000e+03	2.000000	(
max	650000.000000	33.000000	7.500000	5461.000000	1.164794e+06	3.500000	

8 rows × 24 columns

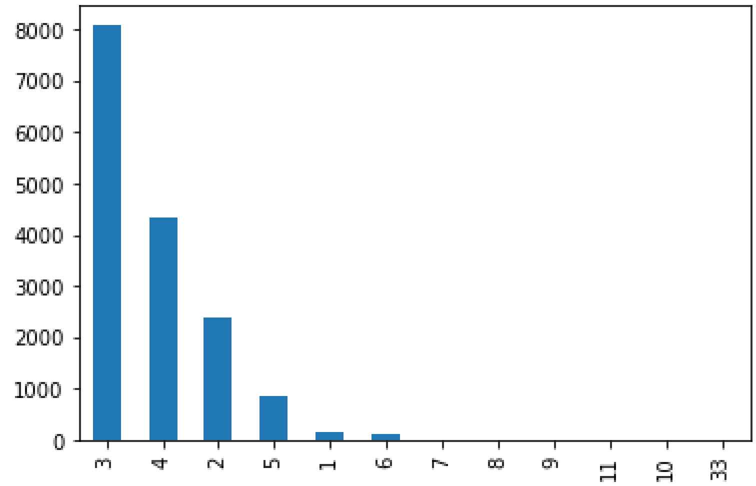
In [35]:
```
# Explore correlation after narrowing data set

df.corr()
```
executed in 37ms, finished 02:56:49 2021-10-25

Out[35]:

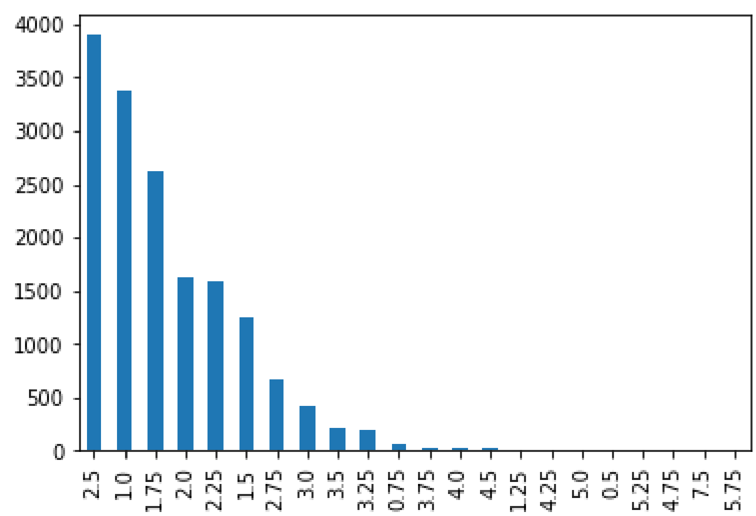
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfro
price	1.000000	0.176433	0.315515	0.417273	0.071915	0.200133	0.0227
bedrooms	0.176433	1.000000	0.456624	0.589010	0.020603	0.102590	-0.0384
bathrooms	0.315515	0.456624	1.000000	0.666310	0.023810	0.492458	-0.0292
sqft_living	0.417273	0.589010	0.666310	1.000000	0.132353	0.268110	-0.0116
sqft_lot	0.071915	0.020603	0.023810	0.132353	1.000000	-0.052962	0.0210
floors	0.200133	0.102590	0.492458	0.268110	-0.052962	1.000000	-0.0151
waterfront	0.022750	-0.038404	-0.029209	-0.011641	0.021022	-0.015131	1.0000
view	0.126625	0.008418	0.038959	0.109977	0.105113	-0.030150	0.2654
condition	-0.003818	0.020562	-0.161222	-0.076913	0.015127	-0.291827	0.0204
grade	0.451436	0.256539	0.557766	0.586257	0.034926	0.424865	-0.0216
sqft_above	0.318006	0.449544	0.589683	0.811530	0.127287	0.493237	-0.0176
sqft_basement	0.196807	0.276462	0.188410	0.397274	0.020780	-0.319435	0.0087
yr_built	0.040321	0.162717	0.593987	0.353469	0.005088	0.547670	-0.0366
yr_renovated	0.027449	-0.008372	-0.012700	-0.001082	0.018037	-0.021295	0.0477
lat	0.479098	-0.107569	-0.112134	-0.143029	-0.107369	-0.017537	-0.0362
long	0.054300	0.132547	0.232980	0.256092	0.217447	0.105360	-0.0555
sqft_living15	0.382913	0.342631	0.478078	0.682108	0.152715	0.204379	0.0002
sqft_lot15	0.067769	0.015451	0.024263	0.143713	0.712409	-0.057095	0.0464
yr_sold	0.007087	-0.009127	-0.027191	-0.024446	-0.006834	-0.018975	-0.0050
renovated	0.027333	-0.008531	-0.013034	-0.001096	0.018080	-0.021294	0.0477
basement_present	0.174533	0.138967	0.121556	0.200849	-0.020799	-0.292454	0.0093
actual_age_of_property	-0.040202	-0.162862	-0.594414	-0.353860	-0.005201	-0.547963	0.0365
bathrooms_per_bedroom	0.184907	-0.306894	0.638663	0.181322	0.005793	0.435363	0.0068
sqft_living_to_sqft_lot	0.188304	-0.024121	0.328013	0.050878	-0.257394	0.610488	-0.0319

24 rows × 24 columns

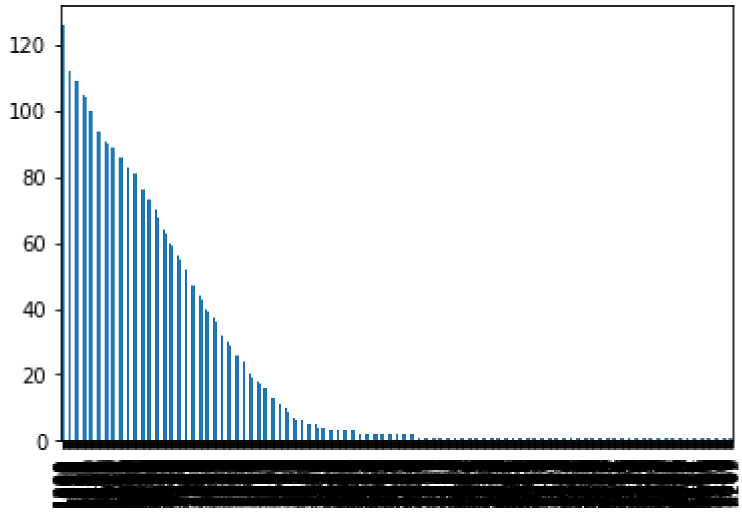
In [36]:
```
df['bedrooms'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 144ms, finished 02:56:50 2021-10-25



In [37]:
```
df['bathrooms'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 193ms, finished 02:56:50 2021-10-25

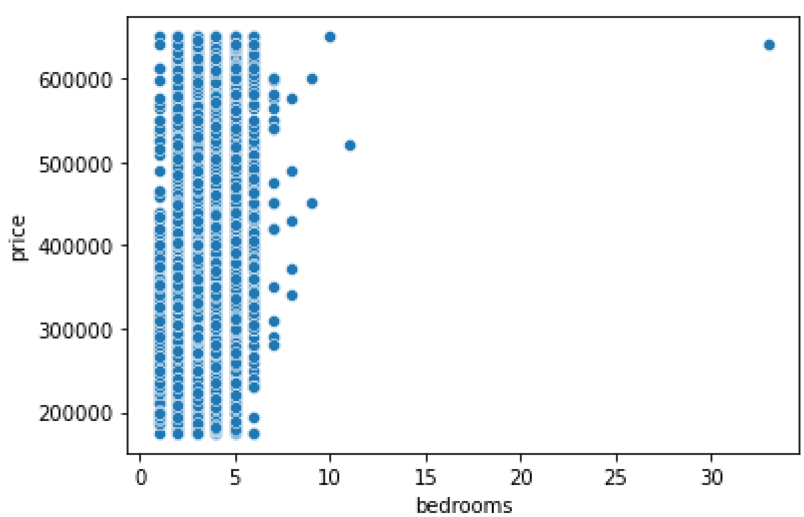


In [38]:
```
df['sqft_living'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 8.51s, finished 02:56:58 2021-10-25



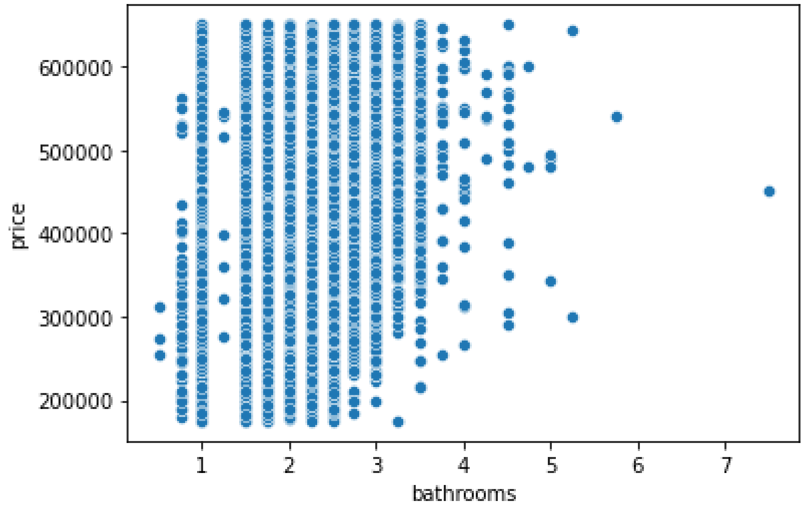
In [39]:
```
sns.scatterplot(data=df, x='bedrooms', y='price')
```
executed in 134ms, finished 02:56:58 2021-10-25

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



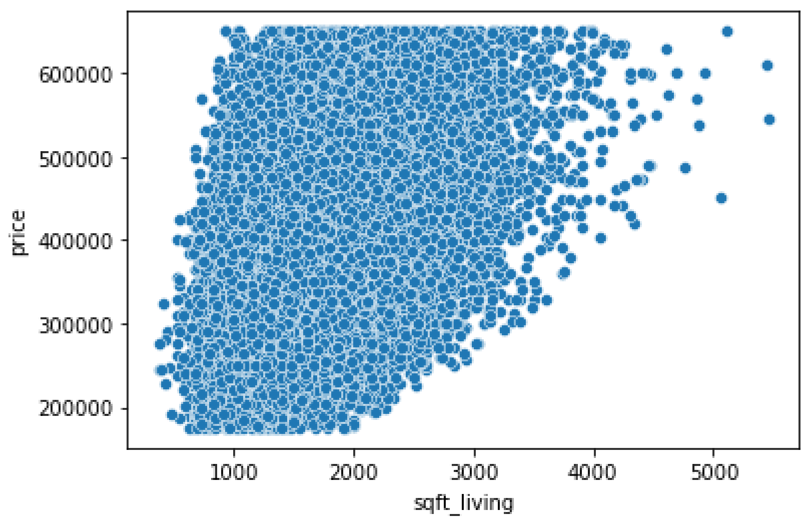
In [40]:
```
sns.scatterplot(data=df, x='bathrooms', y='price')
```
executed in 137ms, finished 02:56:59 2021-10-25

Out[40]: <AxesSubplot:xlabel='bathrooms', ylabel='price'>



In [41]:
```
sns.scatterplot(data=df, x='sqft_living', y='price')
```
executed in 135ms, finished 02:56:59 2021-10-25

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



After seeing outliers in the data, refine data set to:

1. bedrooms to 6 or less
2. sqft_living to 4000 or less

In [42]:
```
df.describe()
```
executed in 65ms, finished 02:56:59 2021-10-25

Out[42]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	15993.000000	15993.000000	15993.000000	15993.000000	1.599300e+04	15993.000000	1599
mean	400987.591009	3.244982	1.953667	1800.124742	1.317614e+04	1.431533	(
std	123893.841894	0.886582	0.660056	631.746037	3.310214e+04	0.537112	(
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.000000	(
25%	299950.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.000000	(
50%	392000.000000	3.000000	2.000000	1720.000000	7.439000e+03	1.000000	(
75%	499990.000000	4.000000	2.500000	2180.000000	9.968000e+03	2.000000	(
max	650000.000000	33.000000	7.500000	5461.000000	1.164794e+06	3.500000	[illegible]

8 rows × 24 columns

In [43]:
```
df = df[df['bedrooms'] <= 6]
df.describe()
```
executed in 67ms, finished 02:56:59 2021-10-25

Out[43]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wat
count	15965.000000	15965.000000	15965.000000	15965.000000	1.596500e+04	15965.00000	15965.
mean	400835.351519	3.235766	1.951425	1797.901973	1.318139e+04	1.43135	0.
std	123867.266044	0.835528	0.656015	629.416330	3.312958e+04	0.53716	0.
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.00000	0.
25%	299900.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.00000	0.
50%	392000.000000	3.000000	2.000000	1720.000000	7.434000e+03	1.00000	0.
75%	499950.000000	4.000000	2.500000	2180.000000	9.966000e+03	2.00000	0.
max	650000.000000	6.000000	5.250000	5461.000000	1.164794e+06	3.50000	1.

8 rows × 24 columns

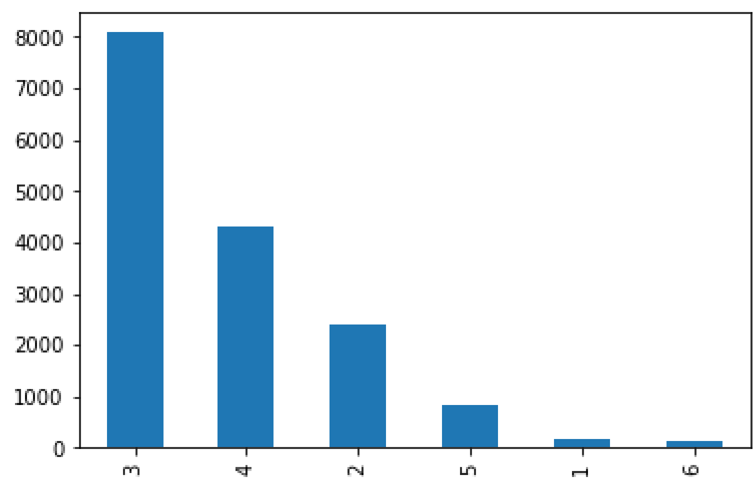
In [44]:
```
df = df[df['sqft_living'] <= 4000]
df.describe()
```
executed in 67ms, finished 02:56:59 2021-10-25

Out[44]:

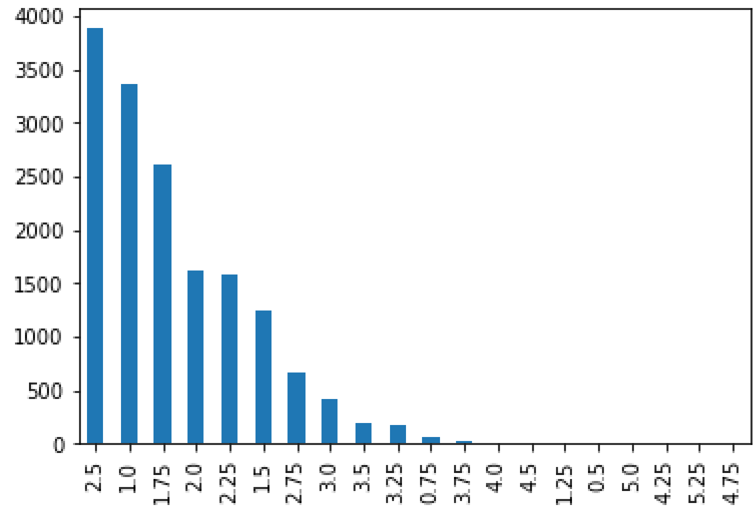
	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
count	15916.000000	15916.000000	15916.000000	15916.000000	1.591600e+04	15916.000000	1591
mean	400358.968145	3.232093	1.947427	1789.922971	1.310122e+04	1.430227	(
std	123707.148968	0.833128	0.652116	613.395670	3.301546e+04	0.536934	(
min	175000.000000	1.000000	0.500000	370.000000	5.720000e+02	1.000000	(
25%	299725.000000	3.000000	1.500000	1330.000000	5.000000e+03	1.000000	(
50%	390000.000000	3.000000	2.000000	1720.000000	7.420000e+03	1.000000	(
75%	499900.000000	4.000000	2.500000	2180.000000	9.936000e+03	2.000000	(
max	650000.000000	6.000000	5.250000	4000.000000	1.164794e+06	3.500000	-

8 rows × 24 columns

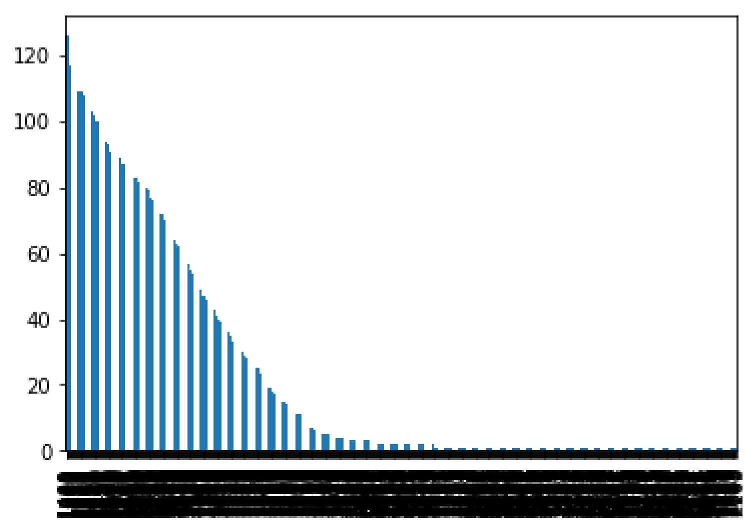
In [45]:
```
df['bedrooms'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 113ms, finished 02:56:59 2021-10-25



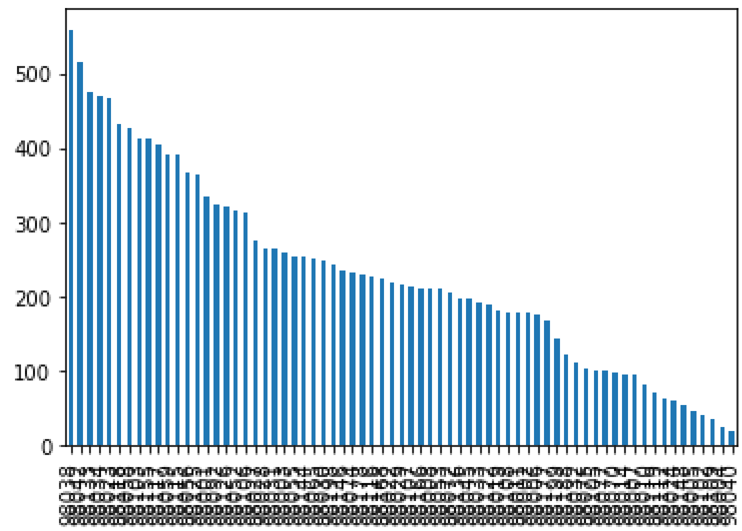
In [46]:
```
df['bathrooms'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 184ms, finished 02:56:59 2021-10-25



In [47]:
```
df['sqft_living'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 8.17s, finished 02:57:07 2021-10-25



In [48]:
```
df['zipcode'].value_counts().plot(kind='bar')
sns.despine;
```
executed in 1.03s, finished 02:57:08 2021-10-25



In [49]:
```
df['zipcode'].value_counts(ascending=True)
```
executed in 6ms, finished 02:57:08 2021-10-25

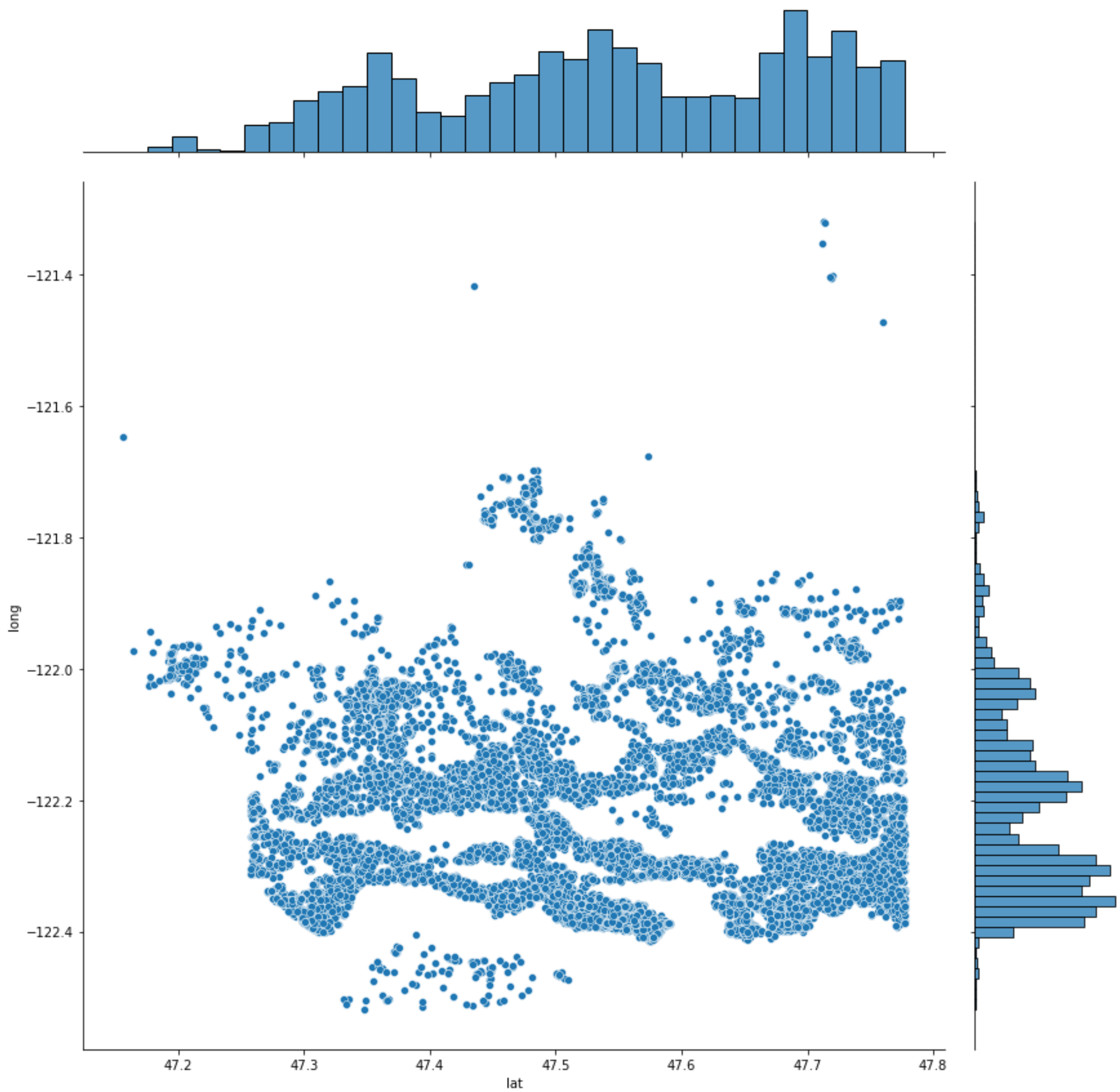
Out[49]:
```
98040     19
98004     23
98109     35
98102     40
98005     47
        ...
98023    466
98034    469
98133    475
98042    516
98038    559
Name: zipcode, Length: 69, dtype: int64
```

In [50]:
```
# For Future Work: explore correlation between these zip codes and price
# least_houses_zip = df[df['zipcode' == '98004', '98109', '98112', '98102',
# most_houses_zips = 98023, 98034, 98133, 98042, 98038
```
executed in 1ms, finished 02:57:08 2021-10-25

In [51]:
```
# For future work on zipcodes

plt.figure(figsize=(12,12))
sns.jointplot(x=df['lat'], y=df['long'], size=12)
plt.xlabel('Latitude', fontsize=11)
plt.ylabel('Longitude', fontsize=11)
plt.show()
sns.despine;
```

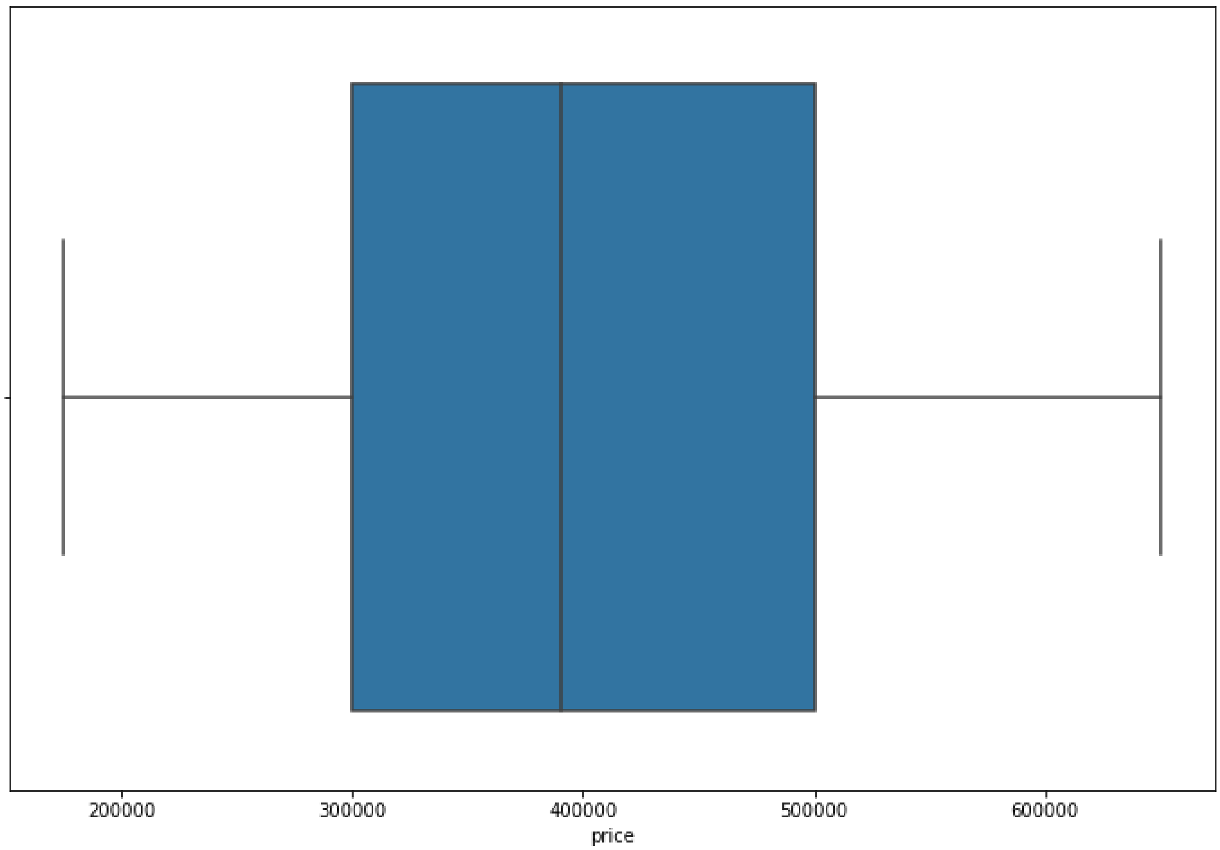
executed in 567ms, finished 02:57:09 2021-10-25

```
/Users/v/opt/anaconda3/envs/learn-env/lib/python3.8/site-packages/seabor
n/axisgrid.py:2015: UserWarning: The `size` parameter has been renamed to
`height`; please update your code.
  warnings.warn(msg, UserWarning)

<Figure size 864x864 with 0 Axes>
```



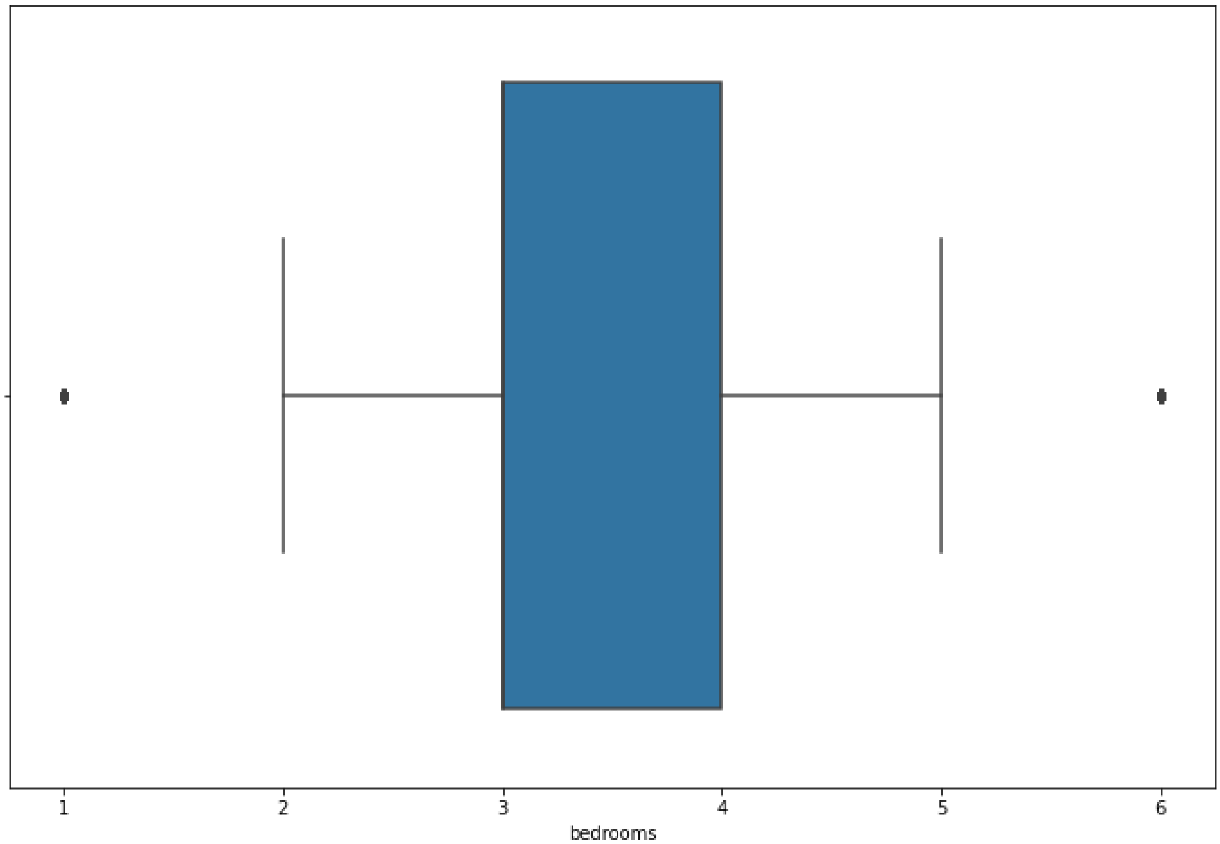
In [52]:
```
# Boxplot for price

fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(x='price', data=df, ax=ax)
```
executed in 87ms, finished 02:57:09 2021-10-25

Out[52]:
```
<AxesSubplot:xlabel='price'>
```



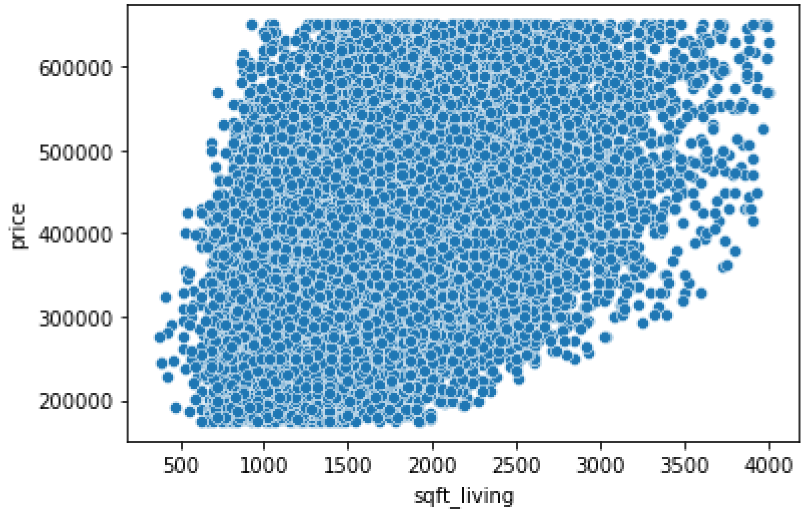
In [53]:
```
# Boxplot for bedrooms

fig, ax = plt.subplots(figsize=(12,8))
sns.boxplot(x='bedrooms', data=df, ax=ax)
```
executed in 250ms, finished 02:57:09 2021-10-25

Out[53]: <AxesSubplot:xlabel='bedrooms'>



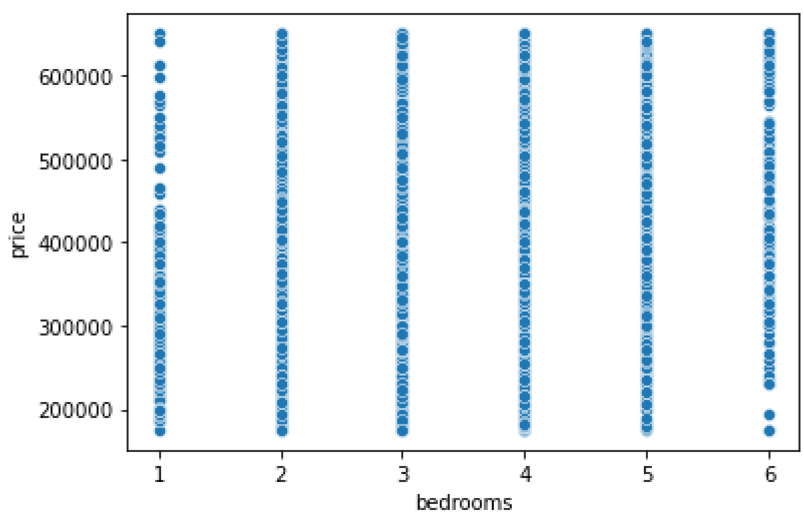
In [54]:
```
# Scatterplot for sqft_living and price

sns.scatterplot(data=df, x='sqft_living', y='price')
```
executed in 176ms, finished 02:57:10 2021-10-25

Out[54]: <AxesSubplot:xlabel='sqft_living', ylabel='price'>



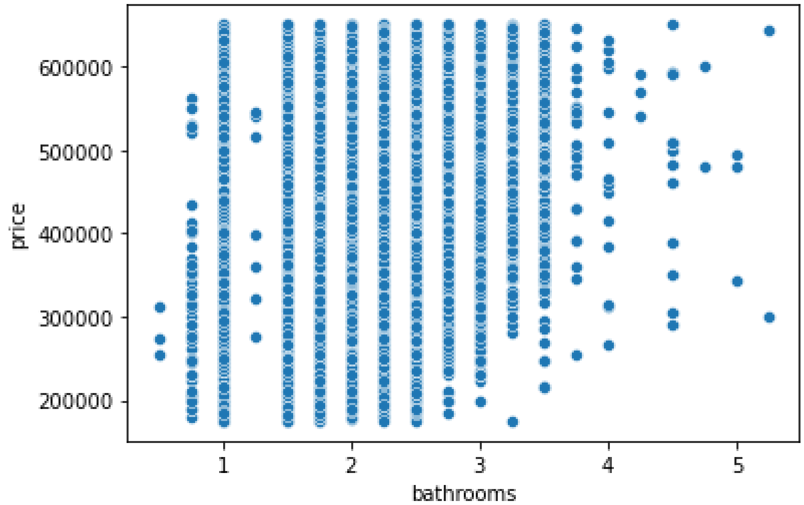
In [55]:
```
# Scatterplot for bedrooms and price

sns.scatterplot(data=df, x='bedrooms', y='price')
```
executed in 153ms, finished 02:57:10 2021-10-25

Out[55]: <AxesSubplot:xlabel='bedrooms', ylabel='price'>



In [56]:
```
# Scatterplot for bathrooms and price

sns.scatterplot(data=df, x='bathrooms', y='price')
```
executed in 147ms, finished 02:57:10 2021-10-25

Out[56]: <AxesSubplot:xlabel='bathrooms', ylabel='price'>



2.4 Data Preparation

Start dropping columns that will not be used

In [57]:
```
# Will not use 'view' (# of times the house has been viewed) for analysis

if 'view' in df.columns:
    df.drop('view', axis=1, inplace=True)
```
executed in 5ms, finished 02:57:10 2021-10-25

In [58]:
```
# Drop date

if 'date' in df.columns:
    df.drop('date', axis=1, inplace=True)
```
executed in 5ms, finished 02:57:10 2021-10-25

2.4.1 Create Target and Explore Data with More Visualizations

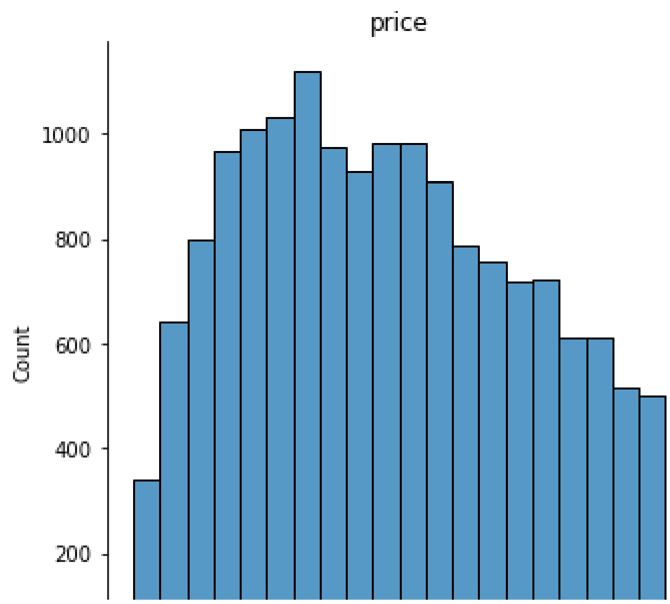
TARGET is price

In [59]:
```
TARGET = 'price'
X_VALS = [c for c in df.columns if c != TARGET]
TARGET in X_VALS
```
executed in 3ms, finished 02:57:10 2021-10-25

Out[59]: False

In [*]:
```
for col in df.columns:
    plt.figure(figsize=(12,8))
    sns.displot(df[col],bins=20)
    plt.title(col)
    plt.show();
```
execution queued 02:56:46 2021-10-25

<Figure size 864x576 with 0 Axes>



In [*]:
```
for col in df.columns:
    plt.scatter(df[col], df[TARGET])
    plt.title(col)
    plt.show()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
for col in df.columns:
    plt.hist(df[col])
    plt.title(col)
    plt.show()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
for col in df.select_dtypes('number').columns:
    plt.boxplot(df[col], vert=False)
    plt.title(col)
    plt.show()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
corr = df.corr().abs()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap="GnBu", vmin=0, vmax=1.0, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})
```
execution queued 02:56:46 2021-10-25

In [*]:
```
plt.figure(figsize=(14, 14))
corr_matrix = df.corr().abs().round(2)
sns.heatmap(data=corr_matrix,cmap="GnBu",annot=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
df.head()
```
execution queued 02:56:46 2021-10-25

2.4.2 Clean Data Before Modeling

- drop sqft_basement, sqft_basement15, sqft_lot, sqft_lot15, yr_renovated
- drop longitude, latitude,

In [*]:
```
if 'sqft_basement' in df.columns:
    df.drop('sqft_basement', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'sqft_basement15' in df.columns:
    df.drop('sqft_basement15', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'sqft_lot' in df.columns:
    df.drop('sqft_lot', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'sqft_lot15' in df.columns:
    df.drop('sqft_lot15', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'sqft_living15' in df.columns:
    df.drop('sqft_living15', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'yr_renovated' in df.columns:
    df.drop('yr_renovated', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'lat' in df.columns:
    df.drop('lat', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
if 'long' in df.columns:
    df.drop('long', axis=1, inplace=True)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
df.head()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
corr = df.corr().abs()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap="GnBu", vmin=0, vmax=1.0, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .75})
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Check data one more time

for col in df.columns:
    plt.scatter(df[col], df[TARGET])
    plt.title(col)
    plt.show()
```
execution queued 02:56:46 2021-10-25

3 Start Building Model

- Create dependent(y) and independent(x) variables
- Create Train and Test data subsets

3.1 Create TARGET and Independent Variables

In [*]:
```
# Dependent variable(y) is price as previously defined as TARGET
# Independent variables(X) are all variables that are not price

y = df[TARGET]
X = df.drop(columns=[TARGET])
X
```
execution queued 02:56:46 2021-10-25

3.2 Create Train and Test Data Subsets

In [*]:
```
# Create Train and Test data subsets using train_test_split
# Check shape of each data set

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=100)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Check percentage of data that is Train data
# Train data is 75% of data; Test data is 25% of data

11_937/(11_937+3979)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Reset index on Train and Test data sets

X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```
execution queued 02:56:46 2021-10-25

Create Number and Category Column Variables

In [*]:
```
# Create variable for "Number" columns (integers, floats)
# Create variable for "Category" columns (objects, strings)
# Check Category Columns

NUMBER_COLS = X_train.select_dtypes('number').columns
CATEGORY_COLS = X_train.select_dtypes('object').columns
CATEGORY_COLS
```
execution queued 02:56:46 2021-10-25

3.3 One Hot Encode Category Columns (zipcode)

In [*]:
```
# ONE HOT ENCODE

ohe = OneHotEncoder(drop='first', sparse=False)
X_train_ohe = ohe.fit_transform(X_train[CATEGORY_COLS])
X_test_ohe = ohe.transform(X_test[CATEGORY_COLS])

X_train_ohe = pd.DataFrame(X_train_ohe, columns=ohe.get_feature_names(CATEG
X_test_ohe = pd.DataFrame(X_test_ohe, columns=ohe.get_feature_names(CATEGOR

X_train_ohe.columns = [c.lower() for c in X_train_ohe]
X_test_ohe.columns = [c.lower() for c in X_test_ohe]
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Check one hot encoding of zipcodes

X_train_ohe.head()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Concatenate Number Columns with One Hot Encoded Columns

X_train_raw = pd.concat([X_train[NUMBER_COLS],
                         X_train_ohe],
                        axis=1)
X_test_raw = pd.concat([X_test[NUMBER_COLS],
                        X_test_ohe],
                       axis=1)
```
execution queued 02:56:46 2021-10-25

3.4 Scale the Data

In [*]:
```
# Scale the Data using StandardScaler()

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train[NUMBER_COLS])
X_test_scaled = scaler.transform(X_test[NUMBER_COLS])

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train[NUMBER_COLS].
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test[NUMBER_COLS].col
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Check the shape of the data

X_train_scaled.shape, X_test_scaled.shape
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Concatenate Scaled data with One Hot Encoded data

X_train_scaled = pd.concat([X_train_scaled,
                            X_train_ohe],
                           axis=1)
X_test_scaled = pd.concat([X_test_scaled,
                           X_test_ohe],
                          axis=1)
```

execution queued 02:56:46 2021-10-25

In [*]:
```
# Check X_train_scaled data

X_train_scaled
```

execution queued 02:56:46 2021-10-25

4 MODELS

4.1 Model 1: Everything

- Used for exploration

In [*]:
```
model1 = sm.OLS(y_train, X_train_scaled).fit()
model1.summary()
```

execution queued 02:56:46 2021-10-25

In [*]:
```
# Results sorted by coefficients descending

results1_as_html = model1.summary().tables[1].as_html()
results1 = pd.read_html(results1_as_html, header=0, index_col=0)[0]
results1.sort_values('coef', ascending=False)#.set_option('display.max_rows
```

execution queued 02:56:46 2021-10-25

*****Check Linear Model Assumptions**

1. Linearity

2. Residual Normality
sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True)
Omnibus Value

3. Homoskedasticity
Durbin-Watson: range of 1.5 to 2.5 is relatively normal

4. Multicollinearity
VIF (variance_inflation_factor())

Also check p-value
A p-value less than 0.05 (typically ≤ 0.05) is statistically significant. It indicates strong evidence against the null hypothesis, as there is less than a 5% probability the null is correct (and the results

are random).

In [*]:
```
sm.graphics.qqplot(model1.resid, dist=stats.norm, line='45', fit=True);
```
execution queued 02:56:46 2021-10-25

4.2 Model Without Zipcode

In [*]:
```
# Model without zipcodes

[c for c in X_train_scaled.columns if not c.startswith('zipcode')]
```
execution queued 02:56:46 2021-10-25

In [*]:
```
model = sm.OLS(y_train, X_train_scaled[[c for c in X_train_scaled.columns i
model.summary2()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
model = sm.OLS(y_train, X_train_scaled).fit()

results_as_html = model.summary().tables[1].as_html()
results = pd.read_html(results_as_html, header=0, index_col=0)[0]
results.sort_values('coef', ascending=False)
```
execution queued 02:56:46 2021-10-25

In [*]:
```
sm.graphics.qqplot(model.resid, dist=stats.norm, line='45', fit=True);
```
execution queued 02:56:46 2021-10-25

In [*]:
```
# Check for Multicollinearity
# Revisit

def create_vif_dct(dataframe, const_col_name='const'):

    if const_col_name not in dataframe.columns:
        dataframe = sm.add_constant(dataframe)

    # Dummy-checking.
    df = dataframe.select_dtypes('number')
    if df.shape != dataframe.shape:
        warnings.warn('\n\nThere are non-numerical columns trying to be pas
    if df.isna().sum().any():
        raise ValueError('There may not be any missing values in the datafr

    # Creating VIF Dictionary.
    vif_dct = {}

    # Loop through each row and set the variable name to the VIF.
    for i in range(len(df.columns)):
        vif = variance_inflation_factor(df.values, i)
        v = df.columns[i]
        vif_dct[v] = vif

    return vif_dct
```

execution queued 02:56:46 2021-10-25

4.3 Model 2: Price and Square Feet Living

In [*]:
```
# Model 2: Price and Square Ft Living

model2 = sm.OLS(y_train, X_train['sqft_living']).fit()
model2.summary()
```

execution queued 02:56:46 2021-10-25

In [*]:
```
sm.graphics.qqplot(model2.resid, dist=stats.norm, line='45', fit=True);
```

execution queued 02:56:46 2021-10-25

4.4 Model 3: Price and Bedrooms

In [*]:
```
# Model 3: Price and Bedrooms

model3 = sm.OLS(y_train, X_train['bedrooms']).fit()
model3.summary()
```

execution queued 02:56:46 2021-10-25

In [*]:
```
sm.graphics.qqplot(model3.resid, dist=stats.norm, line='45', fit=True);
```

execution queued 02:56:46 2021-10-25

4.5 Model 4: Price and Bathrooms

In [*]:
```
# Model 4: Price and Bathrooms

model4 = sm.OLS(y_train, X_train['bathrooms']).fit()
model4.summary()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
sm.graphics.qqplot(model4.resid, dist=stats.norm, line='45', fit=True);
```
execution queued 02:56:46 2021-10-25

4.6 Model 5: Price and Renovated

In [*]:
```
# Model 5: Price and Renovated

model5 = sm.OLS(y_train, X_train['renovated']).fit()
model5.summary()
```
execution queued 02:56:46 2021-10-25

In [*]:
```
sm.graphics.qqplot(model5.resid, dist=stats.norm, line='45', fit=True);
```
execution queued 02:56:46 2021-10-25

5 Evaluation and Conclusions

After building models to evaluate the relationship between price and square feet, bedrooms, bathrooms, and renovation status, we can offer guidance to new home buyers in WA State about the expectation of price relative to square feet of living, bedrooms, and bathrooms.

**** Important note: the results are best suited for home buyers seeking homes with a maximum of 6 bedrooms,4000 square feet, and a budget of ranging from $175,000 to $650,000

Conclusions

- With the highest correlation (r-squared: 0.891) of our models, our model for price to square feet shows: **in King County, WA, every additional square feet of space costs approximately $209**
- Model for price to bedrooms shows: **every additional bedroom costs approximately $117,500**
- Model for price to bathrooms shows: **every additional bathroom costs approximately $190,800**

6 Future Work

Future work:

- Refine existing models and expand dataset for different types of home buyers
- Explore relationship of price to zip code

- Build models for Suburbs (Medina, WA) vs. City (Seattle, WA)
- Build more comprehensive models considering other factors such as location, renovations, waterfront view