In [1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler, LabelEnc
from sklearn.preprocessing import OrdinalEncoder
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
from scipy import stats as stats
import seaborn as sns
from matplotlib import pyplot as plt
import statsmodels.api as sm
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import svm
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

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

In [3]:
```
df.head()
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	v
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN	N
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	NO	N
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	NO	N
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	NO	N
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	NO	N

5 rows × 21 columns

In [4]:
```
df.columns
```

Out[4]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

In [5]:
```
df.describe()
```

Out[5]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floc

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floc
count	2.159700e+04	2.159700e+04	21597.000000	21597.000000	21597.000000	2.159700e+04	21597.0000
mean	4.580474e+09	5.402966e+05	3.373200	2.115826	2080.321850	1.509941e+04	1.4940
std	2.876736e+09	3.673681e+05	0.926299	0.768984	918.106125	4.141264e+04	0.5396
min	1.000102e+06	7.800000e+04	1.000000	0.500000	370.000000	5.200000e+02	1.0000
25%	2.123049e+09	3.220000e+05	3.000000	1.750000	1430.000000	5.040000e+03	1.0000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.5000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068500e+04	2.0000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.5000

In [6]:
```
df.shape
```

Out[6]: (21597, 21)

In [7]:
```
df.corr()['price']['yr_renovated']
```

Out[7]: 0.129599275906393

In [8]:
```
#Dropping columns tht are objects or that are not necessary to one hot encode.
df.drop(columns=['waterfront','date','view','yr_renovated','lat', 'long', 'sqft_living1
```

In [9]:
```
#zipped together the columns that were highly correlated after dropping a few columns
# this helps me see what columns are directly and closely correlated.


df_new=df.corr().abs().stack().reset_index().sort_values(0, ascending=False)
df_new['pairs']= list(zip(df_new.level_0, df_new.level_1))
df_new.set_index(['pairs'], inplace = True)
df_new.drop(columns=['level_1', 'level_0'], inplace = True)
df_new.columns = ['cc']
```

In [10]:
```
df_new.head(50)
```

Out[10]:

	cc
pairs	
(id, id)	1.000000
(price, price)	1.000000
(yr_built, yr_built)	1.000000
(sqft_above, sqft_above)	1.000000

	cc
pairs	
(floors, floors)	1.000000
(sqft_lot, sqft_lot)	1.000000
(sqft_living, sqft_living)	1.000000
(bathrooms, bathrooms)	1.000000
(bedrooms, bedrooms)	1.000000
(zipcode, zipcode)	1.000000
(sqft_living, sqft_above)	0.876448
(sqft_above, sqft_living)	0.876448
(bathrooms, sqft_living)	0.755758
(sqft_living, bathrooms)	0.755758
(sqft_living, price)	0.701917
(price, sqft_living)	0.701917
(bathrooms, sqft_above)	0.686668
(sqft_above, bathrooms)	0.686668
(sqft_above, price)	0.605368
(price, sqft_above)	0.605368
(sqft_living, bedrooms)	0.578212
(bedrooms, sqft_living)	0.578212
(bathrooms, price)	0.525906
(price, bathrooms)	0.525906
(sqft_above, floors)	0.523989
(floors, sqft_above)	0.523989
(bedrooms, bathrooms)	0.514508
(bathrooms, bedrooms)	0.514508
(bathrooms, yr_built)	0.507173
(yr_built, bathrooms)	0.507173
(bathrooms, floors)	0.502582
(floors, bathrooms)	0.502582
(floors, yr_built)	0.489193
(yr_built, floors)	0.489193
(bedrooms, sqft_above)	0.479386
(sqft_above, bedrooms)	0.479386

	cc
pairs	
(sqft_above, yr_built)	0.424037
(yr_built, sqft_above)	0.424037
(sqft_living, floors)	0.353953
(floors, sqft_living)	0.353953
(zipcode, yr_built)	0.347210
(yr_built, zipcode)	0.347210
(yr_built, sqft_living)	0.318152
(sqft_living, yr_built)	0.318152
(bedrooms, price)	0.308787
(price, bedrooms)	0.308787
(zipcode, sqft_above)	0.261570
(sqft_above, zipcode)	0.261570
(floors, price)	0.256804
(price, floors)	0.256804

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 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   condition      21597 non-null  object 
 8   grade          21597 non-null  object 
 9   sqft_above     21597 non-null  int64  
 10  sqft_basement  21597 non-null  object 
 11  yr_built       21597 non-null  int64  
 12  zipcode        21597 non-null  int64  
dtypes: float64(3), int64(7), object(3)
memory usage: 2.1+ MB
```

In [12]:
```
#Grabbing a random sample point.
sample_point2= df.sample(1)
sample_price = sample_point2.iloc[0,0]
fin_sample_point = sample_point2.drop('price', axis = 1)
print(f'Price of Sample: $ {sample_price}')
fin_sample_point
```

```
Price of Sample: $ 7750500120
```

Out[12]:

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above
17748	7750500120	3	1.0	950	4760	1.5	Average	6 Low Average	950

In [13]:

```
def train_lr_randomly(data, sample_point=None, ntimes=100):
    r2= []
    rmse = []
    point_preds = [] if (sample_point is not None) else None

    for i in range(ntimes):
        df_sample = data.sample(5000, replace = True)
        y = df_sample.price
        x = df_sample.drop('price', axis = 1)

        lr = LinearRegression()
        lr.fit(x,y)

        y_hat = lr.predict(x)
        rmse.append(np.sqrt(mean_squared_error(y, y_hat)))
        r2.append(lr.sccore(x,y))

        if sample_point is not None:
            y_hat_point = lr.predict(sample_point)

            point_preds.append(y_hat_point[0])

    return r2, rmse, point_preds
```

In [14]:

```
#OneHot Encoding condition colun to convert it from a string

condition_train = df[['condition']]
condition_ohe = OneHotEncoder(categories='auto', sparse=False, handle_unknown="ignore")
condition_ohe.fit(condition_train)
condition_ohe.categories_
condition_encoded_train = condition_ohe.transform(condition_train)
condition_encoded_train = pd.DataFrame(condition_encoded_train, columns=condition_ohe.c
condition_encoded_train
df.drop('condition', axis=1, inplace=True)
df = pd.concat([df, condition_encoded_train], axis=1)
df
```

Out[14]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above
0	7129300520	221900.0	3	1.00	1180	5650	1.0	7 Average	1180
1	6414100192	538000.0	3	2.25	2570	7242	2.0	7 Average	2170
2	5631500400	180000.0	2	1.00	770	10000	1.0	6 Low Average	770

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above
3	2487200875	604000.0	4	3.00	1960	5000	1.0	7 Average	1050
4	1954400510	510000.0	3	2.00	1680	8080	1.0	8 Good	1680
...	...	...	...	...	...	...	...	...	...
21592	263000018	360000.0	3	2.50	1530	1131	3.0	8 Good	1530
21593	6600060120	400000.0	4	2.50	2310	5813	2.0	8 Good	2310
21594	1523300141	402101.0	2	0.75	1020	1350	2.0	7 Average	1020
21595	291310100	400000.0	3	2.50	1600	2388	2.0	8 Good	1600
21596	1523300157	325000.0	2	0.75	1020	1076	2.0	7 Average	1020

21597 rows × 17 columns

In [15]:
```
#Borrowed from Ashley but utilized to change my grade values into integers

df['grade'].replace(to_replace="3 Poor", value=3, inplace=True)
df['grade'].replace(to_replace="4 Low", value=4, inplace=True)
df['grade'].replace(to_replace="5 Fair", value=5, inplace=True)
df['grade'].replace(to_replace="6 Low Average", value=6, inplace=True)
df['grade'].replace(to_replace="7 Average", value=7, inplace=True)
df['grade'].replace(to_replace="8 Good", value=8, inplace=True)
df['grade'].replace(to_replace="9 Better", value=9, inplace=True)
df['grade'].replace(to_replace="10 Very Good", value=10, inplace=True)
df['grade'].replace(to_replace="11 Excellent", value=11, inplace=True)
df['grade'].replace(to_replace="12 Luxury", value=12, inplace=True)
df['grade'].replace(to_replace="13 Mansion", value=13, inplace=True)
```

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 17 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 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   grade          21597 non-null  int64  
 8   sqft_above     21597 non-null  int64  
 9   sqft_basement  21597 non-null  object 
 10  yr_built       21597 non-null  int64  
 11  zipcode        21597 non-null  int64  
 12  Average        21597 non-null  float64
```

```
 13  Fair           21597 non-null  float64
 14  Good           21597 non-null  float64
 15  Poor           21597 non-null  float64
 16  Very Good      21597 non-null  float64
dtypes: float64(8), int64(8), object(1)
memory usage: 2.8+ MB
```

In [17]:
```
#going to place my grade of house into bins to make them appear easier to understand
bins = (2.0,4.0,6.0,8.0,10.0,12.0)
group_name = ['bad','approaching_average','below_average','Average', 'Great' 'Excellent
df['grade'] = pd.cut(df['grade'], bins = bins, labels = group_name)
df['grade'].unique()
```

Out[17]:
```
['below_average', 'approaching_average', 'GreatExcellent', 'Average', 'bad', NaN]
Categories (5, object): ['bad' < 'approaching_average' < 'below_average' < 'Average' <
'GreatExcellent']
```

In [18]:
```
#Now I am running a label encoder on the categories of grade these bins to make sure th
label_quality = LabelEncoder()
df['grade'] = label_quality.fit_transform(df['grade'])
```

In [19]:
```
#now if we run our dataframe you can see that grades are labeled 0-5
df
```

Out[19]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above	sc
0	7129300520	221900.0	3	1.00	1180	5650	1.0	4	1180	
1	6414100192	538000.0	3	2.25	2570	7242	2.0	4	2170	
2	5631500400	180000.0	2	1.00	770	10000	1.0	2	770	
3	2487200875	604000.0	4	3.00	1960	5000	1.0	4	1050	
4	1954400510	510000.0	3	2.00	1680	8080	1.0	4	1680	
...	...	...	...	...	...	...	...	...	...	...
21592	263000018	360000.0	3	2.50	1530	1131	3.0	4	1530	
21593	6600060120	400000.0	4	2.50	2310	5813	2.0	4	2310	
21594	1523300141	402101.0	2	0.75	1020	1350	2.0	4	1020	
21595	291310100	400000.0	3	2.50	1600	2388	2.0	4	1600	
21596	1523300157	325000.0	2	0.75	1020	1076	2.0	4	1020	

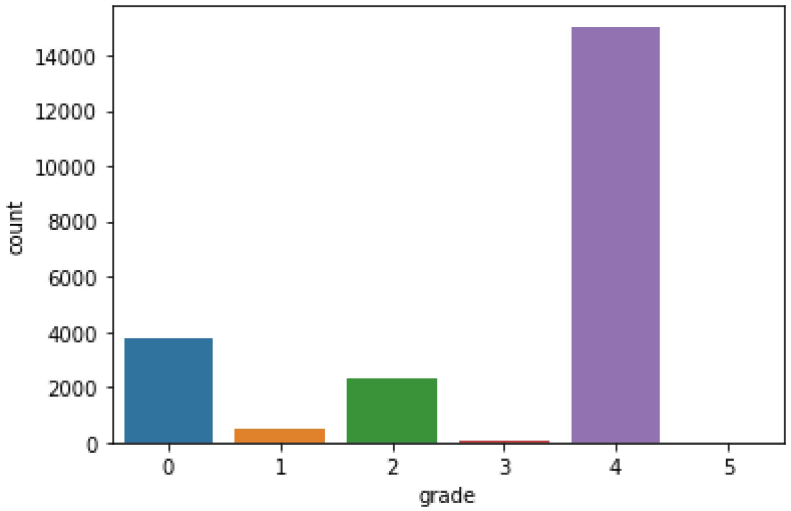
21597 rows × 17 columns

In [20]:
```
#ou ccan see just how much average fixtures in a home will influence our model
sns.countplot(df['grade'])
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
```

```
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

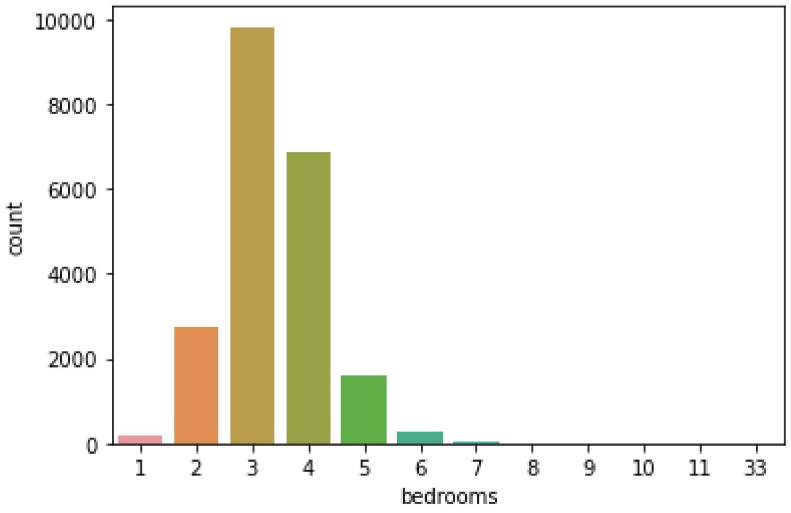
Out[20]: <AxesSubplot:xlabel='grade', ylabel='count'>



In [21]:
```
#Lets do this for all columns
sns.countplot(df['bedrooms'])
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

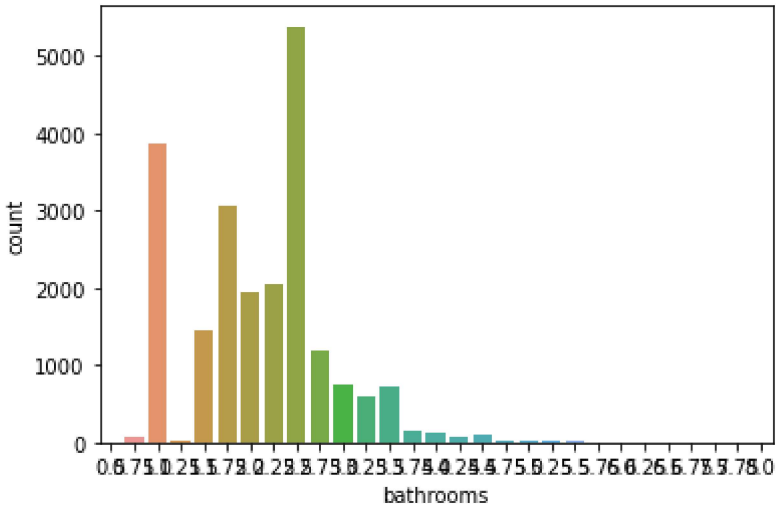
Out[21]: <AxesSubplot:xlabel='bedrooms', ylabel='count'>



In [22]:
```
sns.countplot(df['bathrooms'])
#bathrooms looks a bit weird lets look at that one some more
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[22]: <AxesSubplot:xlabel='bathrooms', ylabel='count'>



In [23]:
```
df['bathrooms'].isna().sum()
```

Out[23]:
```
0
```

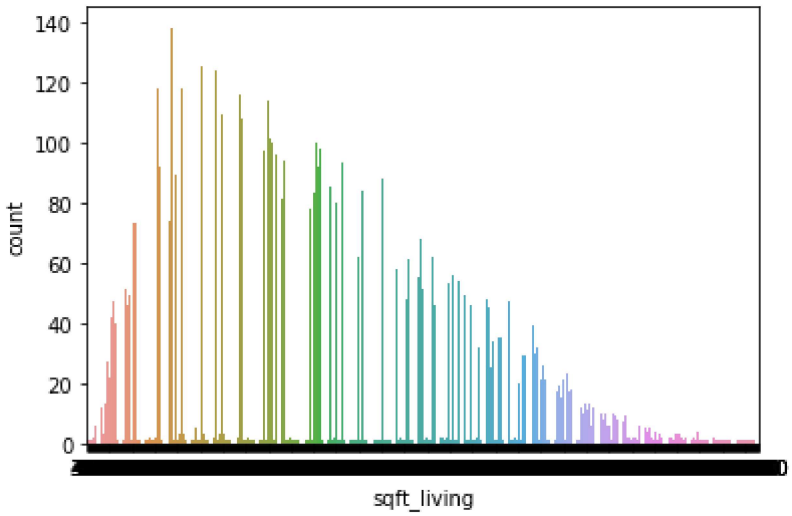
In [24]:
```
df['bathrooms'].value_counts()
#nothing out oof the ordinary numbers appeared congested due to 2 decimal floats
#or there are 4 homes with only a half bath but whatever
```

Out[24]:
```
2.50    5377
1.00    3851
1.75    3048
2.25    2047
2.00    1930
1.50    1445
2.75    1185
3.00     753
3.50     731
3.25     589
3.75     155
4.00     136
4.50     100
4.25      79
0.75      71
4.75      23
5.00      21
5.25      13
5.50      10
1.25       9
6.00       6
5.75       4
0.50       4
6.25       2
8.00       2
6.75       2
6.50       2
7.75       1
7.50       1
Name: bathrooms, dtype: int64
```

In [25]:
```
sns.countplot(df['sqft_living'])
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

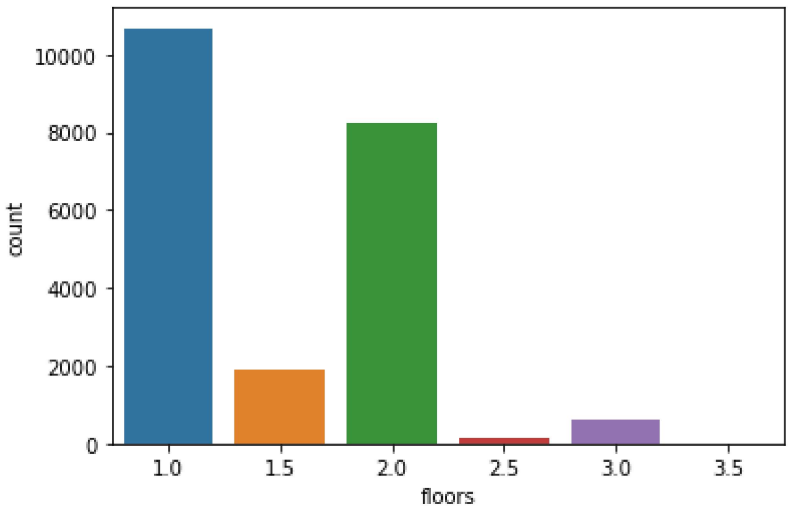
Out[25]: <AxesSubplot:xlabel='sqft_living', ylabel='count'>



In [26]:
```
sns.countplot(df['floors'])
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

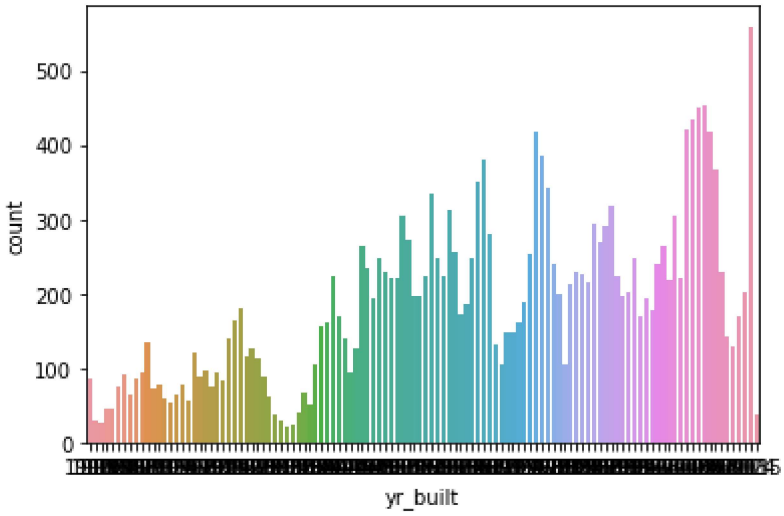
Out[26]: <AxesSubplot:xlabel='floors', ylabel='count'>



In [27]:
```
sns.countplot(df['yr_built'])
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[27]: <AxesSubplot:xlabel='yr_built', ylabel='count'>



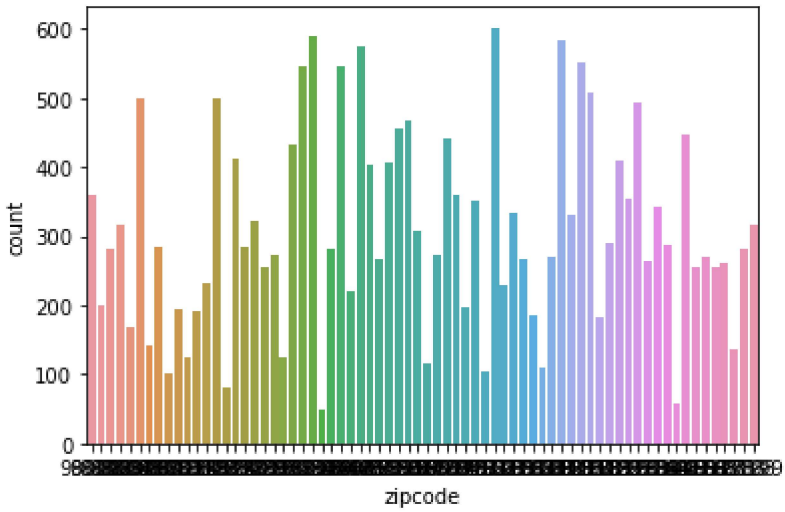
In [28]:
```
df['yr_built'].value_counts()
```

Out[28]:
```
2014    559
2006    453
2005    450
2004    433
2003    420
       ...
1933     30
1901     29
1902     27
1935     24
1934     21
Name: yr_built, Length: 116, dtype: int64
```

In [29]:
```
sns.countplot(df['zipcode'])
```

```
C:\Users\metropolitanparkapts\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fut
ureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an explici
t keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[29]:
```
<AxesSubplot:xlabel='zipcode', ylabel='count'>
```



In [30]:

```
#for the most part there isn't an extreme correlation in an column having one outier th
#the entire column with the exception of grade.
```

In [44]:
```
df.drop(columns = ['sqft_above','sqft_basement'], axis = 1,  inplace = True)
#forgot to drop these two.
df
```

Out[44]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	yr_built	zipco
0	7129300520	221900.0	3	1.00	1180	5650	1.0	4	1955	981
1	6414100192	538000.0	3	2.25	2570	7242	2.0	4	1951	981
2	5631500400	180000.0	2	1.00	770	10000	1.0	2	1933	980
3	2487200875	604000.0	4	3.00	1960	5000	1.0	4	1965	981
4	1954400510	510000.0	3	2.00	1680	8080	1.0	4	1987	980
...	...	...	...	...	...	...	...	...	...	
21592	263000018	360000.0	3	2.50	1530	1131	3.0	4	2009	981
21593	6600060120	400000.0	4	2.50	2310	5813	2.0	4	2014	981
21594	1523300141	402101.0	2	0.75	1020	1350	2.0	4	2009	981
21595	291310100	400000.0	3	2.50	1600	2388	2.0	4	2004	980
21596	1523300157	325000.0	2	0.75	1020	1076	2.0	4	2008	981

21597 rows × 15 columns

In [45]:
```
X = df.drop('price', axis = 1)
y = df['price']
```

In [46]:
```
#Train And Test and Split
X_train, X_test, y_train, y_test =train_test_split(X,y, test_size = .3, random_state= 4
```

In [47]:
```
#applying a standard scaler
#you want to use the same scaling on the test and train data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [48]:
```
#Lets look at our trained data
X_train
```

Out[48]:
```
array([[ 1.41471075, -1.5135143 ,  0.49344162, ..., -0.59403093,
        -0.03355336, -0.29364175],
       [ 1.20380828,  0.68636029,  2.11638165, ..., -0.59403093,
        -0.03355336, -0.29364175],
```

```
       [ 0.90601827, -0.413577  ,  0.49344162, ..., -0.59403093,
        -0.03355336, -0.29364175],
       ...,
       [ 1.40640328,  0.68636029, -0.8049104 , ...,  1.68341404,
        -0.03355336, -0.29364175],
       [-0.99415616, -2.61345159, -1.77867442, ..., -0.59403093,
        -0.03355336, -0.29364175],
       [ 1.37495643,  0.68636029,  0.49344162, ..., -0.59403093,
        -0.03355336, -0.29364175]])
```

These are other model types

Random Forrest Classifier

import our model

rfc = RandomForestClassifier(n_estimators=50)

programming our training data

rfc.fit(X_train, y_train)

prediction

pred_rfc = rfc.predict(X_test)

clf = svm.SVC()

programming our training data

clf.fit(X_train, y_train)

prediction

pred_clf = clf.predict(X_test

In [62]:
```
model = sm.formula.ols(formula='y ~ + X', data = df).fit().summary()
```

```
model
#coefficients seem to be all over the place need to make sure there is  scaler
```

Out[62]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.568
Model:	OLS	**Adj. R-squared:**	0.568
Method:	Least Squares	**F-statistic:**	2184.
Date:	Thu, 06 Jan 2022	**Prob (F-statistic):**	0.00
Time:	06:01:26	**Log-Likelihood:**	-2.9832e+05
No. Observations:	21597	**AIC:**	5.967e+05
Df Residuals:	21583	**BIC:**	5.968e+05
Df Model:	13		
Covariance Type:	nonrobust		

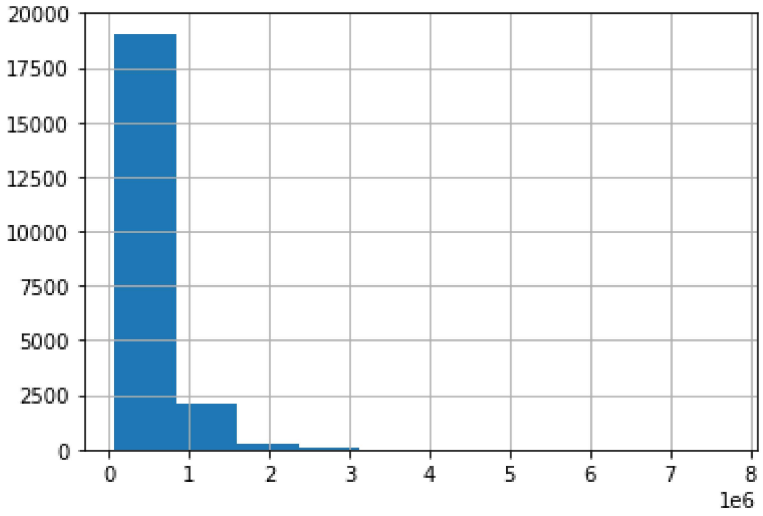
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1.451e+06	2.83e+06	-0.513	0.608	-6.99e+06	4.09e+06
X[0]	-8.518e-07	5.77e-07	-1.477	0.140	-1.98e-06	2.79e-07
X[1]	-6.189e+04	2254.129	-27.456	0.000	-6.63e+04	-5.75e+04
X[2]	7.341e+04	3832.744	19.154	0.000	6.59e+04	8.09e+04
X[3]	275.1661	3.223	85.372	0.000	268.848	281.484
X[4]	-0.3152	0.041	-7.649	0.000	-0.396	-0.234
X[5]	4.61e+04	3830.877	12.034	0.000	3.86e+04	5.36e+04
X[6]	-2.801e+04	1208.708	-23.174	0.000	-3.04e+04	-2.56e+04
X[7]	-3168.5486	77.824	-40.714	0.000	-3321.089	-3016.008
X[8]	81.6759	34.010	2.402	0.016	15.013	148.338
X[9]	-2.762e+05	5.66e+05	-0.488	0.625	-1.39e+06	8.33e+05
X[10]	-3.342e+05	5.66e+05	-0.591	0.555	-1.44e+06	7.75e+05
X[11]	-2.596e+05	5.65e+05	-0.459	0.646	-1.37e+06	8.48e+05
X[12]	-3.48e+05	5.66e+05	-0.615	0.539	-1.46e+06	7.62e+05
X[13]	-2.326e+05	5.65e+05	-0.411	0.681	-1.34e+06	8.76e+05

Omnibus:	15371.635	**Durbin-Watson:**	1.985
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	716902.678
Skew:	2.902	**Prob(JB):**	0.00
Kurtosis:	30.622	**Cond. No.**	4.92e+25

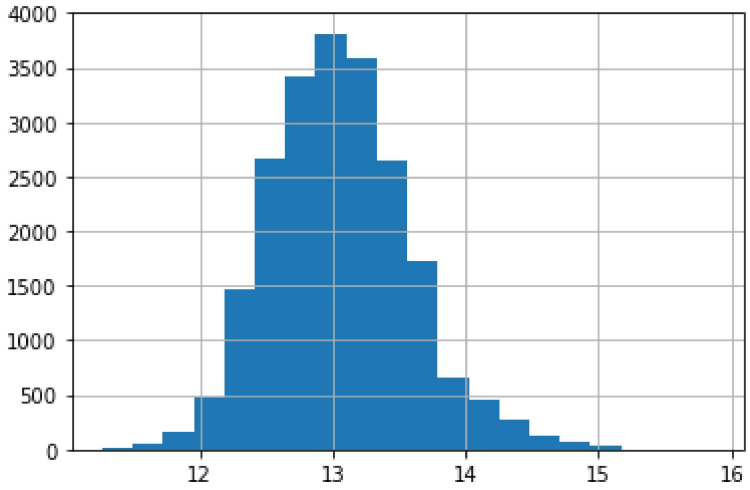
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.61e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [66]:
```
#Target is not normally distrubuted

y.hist();
```



In [69]:
```
log_y = np.log(y)
log_y.hist(bins = 20);
```



In [84]:
```
#added a constant do to plot
X = sm.add_constant(X)
model2 = sm.OLS(log_y, X).fit()
model2.summary()
```

Out[84]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.555
Model:	OLS	**Adj. R-squared:**	0.555

Method:	Least Squares	**F-statistic:**	2070.
Date:	Thu, 06 Jan 2022	**Prob (F-statistic):**	0.00
Time:	06:46:13	**Log-Likelihood:**	-8051.4
No. Observations:	21597	**AIC:**	1.613e+04
Df Residuals:	21583	**BIC:**	1.624e+04
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-11.2608	4.114	-2.737	0.006	-19.324	-3.197
id	1.336e-12	8.39e-13	1.591	0.112	-3.09e-13	2.98e-12
bedrooms	-0.0561	0.003	-17.095	0.000	-0.063	-0.050
bathrooms	0.1220	0.006	21.871	0.000	0.111	0.133
sqft_living	0.0003	4.69e-06	74.516	0.000	0.000	0.000
sqft_lot	-1.072e-07	6e-08	-1.789	0.074	-2.25e-07	1.03e-08
floors	0.1234	0.006	22.129	0.000	0.112	0.134
grade	-0.0356	0.002	-20.238	0.000	-0.039	-0.032
yr_built	-0.0043	0.000	-38.340	0.000	-0.005	-0.004
zipcode	0.0003	4.95e-05	7.032	0.000	0.000	0.000
Average	-2.1554	0.824	-2.617	0.009	-3.770	-0.541
Fair	-2.4282	0.823	-2.949	0.003	-4.042	-0.814
Good	-2.1244	0.823	-2.583	0.010	-3.737	-0.512
Poor	-2.4795	0.824	-3.009	0.003	-4.095	-0.864
Very Good	-2.0733	0.823	-2.520	0.012	-3.686	-0.460

Omnibus:	176.545	**Durbin-Watson:**	1.976
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	245.653
Skew:	-0.106	**Prob(JB):**	4.54e-54
Kurtosis:	3.477	**Cond. No.**	4.92e+25

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.61e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [73]:
```
# More Scaling
constant_x = sm.add_constant(X)
```

In [89]:
```
scaled_x = constant_x - np.mean(constant_x) / np.std(constant_x)
scaled_x.drop(columns = 'const', inplace = True)
scaled_x
```

Out[89]:

	id	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	yr_built
0	7.129301e+09	-0.641673	-1.751519	1177.734064	5649.635383	-1.768536	2.069238	1887.90112
1	6.414100e+09	-0.641673	-0.501519	2567.734064	7241.635383	-0.768536	2.069238	1883.90112
2	5.631500e+09	-1.641673	-1.751519	767.734064	9999.635383	-1.768536	0.069238	1865.90112
3	2.487201e+09	0.358327	0.248481	1957.734064	4999.635383	-1.768536	2.069238	1897.90112
4	1.954401e+09	-0.641673	-0.751519	1677.734064	8079.635383	-1.768536	2.069238	1919.90112
...	...	...	...	...	...	...	...	...
21592	2.630000e+08	-0.641673	-0.251519	1527.734064	1130.635383	0.231464	2.069238	1941.90112
21593	6.600060e+09	0.358327	-0.251519	2307.734064	5812.635383	-0.768536	2.069238	1946.90112
21594	1.523300e+09	-1.641673	-2.001519	1017.734064	1349.635383	-0.768536	2.069238	1941.90112
21595	2.913101e+08	-0.641673	-0.251519	1597.734064	2387.635383	-0.768536	2.069238	1936.90112
21596	1.523300e+09	-1.641673	-2.001519	1017.734064	1075.635383	-0.768536	2.069238	1940.90112

21597 rows × 14 columns

In [80]:
```

```

Out[80]:
```
const          0
id             0
bedrooms       0
bathrooms      0
sqft_living    0
sqft_lot       0
floors         0
grade          0
yr_built       0
zipcode        0
Average        0
Fair           0
Good           0
Poor           0
Very Good      0
dtype: int64
```

In [95]:
```
scaled_x = sm.add_constant(scaled_x)
model_3 =sm.OLS(y, scaled_x).fit()
model_3.summary()
```

Out[95]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.568

Model:	OLS	**Adj. R-squared:**	0.568
Method:	Least Squares	**F-statistic:**	2184.
Date:	Thu, 06 Jan 2022	**Prob (F-statistic):**	0.00
Time:	06:52:12	**Log-Likelihood:**	-2.9832e+05
No. Observations:	21597	**AIC:**	5.967e+05
Df Residuals:	21583	**BIC:**	5.968e+05
Df Model:	13		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1.237e+06	2.42e+06	-0.512	0.609	-5.97e+06	3.5e+06
id	-8.518e-07	5.77e-07	-1.477	0.140	-1.98e-06	2.79e-07
bedrooms	-6.189e+04	2254.129	-27.456	0.000	-6.63e+04	-5.75e+04
bathrooms	7.341e+04	3832.744	19.154	0.000	6.59e+04	8.09e+04
sqft_living	275.1661	3.223	85.372	0.000	268.848	281.484
sqft_lot	-0.3152	0.041	-7.649	0.000	-0.396	-0.234
floors	4.61e+04	3830.877	12.034	0.000	3.86e+04	5.36e+04
grade	-2.801e+04	1208.708	-23.174	0.000	-3.04e+04	-2.56e+04
yr_built	-3168.5486	77.824	-40.714	0.000	-3321.089	-3016.008
zipcode	81.6759	34.010	2.402	0.016	15.013	148.338
Average	3.542e+05	6.64e+05	0.533	0.594	-9.48e+05	1.66e+06
Fair	2.962e+05	6.65e+05	0.446	0.656	-1.01e+06	1.6e+06
Good	3.708e+05	6.65e+05	0.558	0.577	-9.32e+05	1.67e+06
Poor	2.824e+05	6.66e+05	0.424	0.672	-1.02e+06	1.59e+06
Very Good	3.978e+05	6.65e+05	0.599	0.549	-9.05e+05	1.7e+06

Omnibus:	15371.635	**Durbin-Watson:**	1.985
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	716902.678
Skew:	2.902	**Prob(JB):**	0.00
Kurtosis:	30.622	**Cond. No.**	1.50e+24

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 2.8e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [99]:
```
# One More Scale
```

In [ ]:
```

```