```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import warnings
warnings.filterwarnings('ignore')


data = pd.read_csv('kc_house_data.csv')


data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_abov
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	118
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	217
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	77
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	105
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	168

5 rows × 21 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

```
#checking the null values
data.isnull().sum()
```

```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

```
#checking the duplicated values
data.duplicated().sum()
```

```
0
```

```
#checking the datatypes
data.dtypes
```

```
id                int64
date             object
price           float64
bedrooms          int64
bathrooms       float64
sqft_living       int64
sqft_lot          int64
floors          float64
waterfront        int64
view              int64
condition         int64
grade             int64
sqft_above        int64
sqft_basement     int64
yr_built          int64
yr_renovated      int64
zipcode           int64
lat             float64
long            float64
sqft_living15     int64
sqft_lot15        int64
dtype: object
```

```
#unique values in floor column
x=data['floors'].unique()
xlist = x.tolist()
xlist
```

```
[1.0, 2.0, 1.5, 3.0, 2.5, 3.5]
```

```
#the value count for each unique floor value
count=data.floors.value_counts(normalize=True)*100
count
```

```
1.0    49.414704
2.0    38.129829
1.5     8.837274
3.0     2.836256
2.5     0.744922
3.5     0.037015
Name: floors, dtype: float64
```

```
rounded_percentage = pd.DataFrame({'Percentage': count.round(2)})
rounded_percentage
```

	Percentage
1.0	49.41
2.0	38.13
1.5	8.84
3.0	2.84
2.5	0.74
3.5	0.04

```
data['grade'].unique()
```

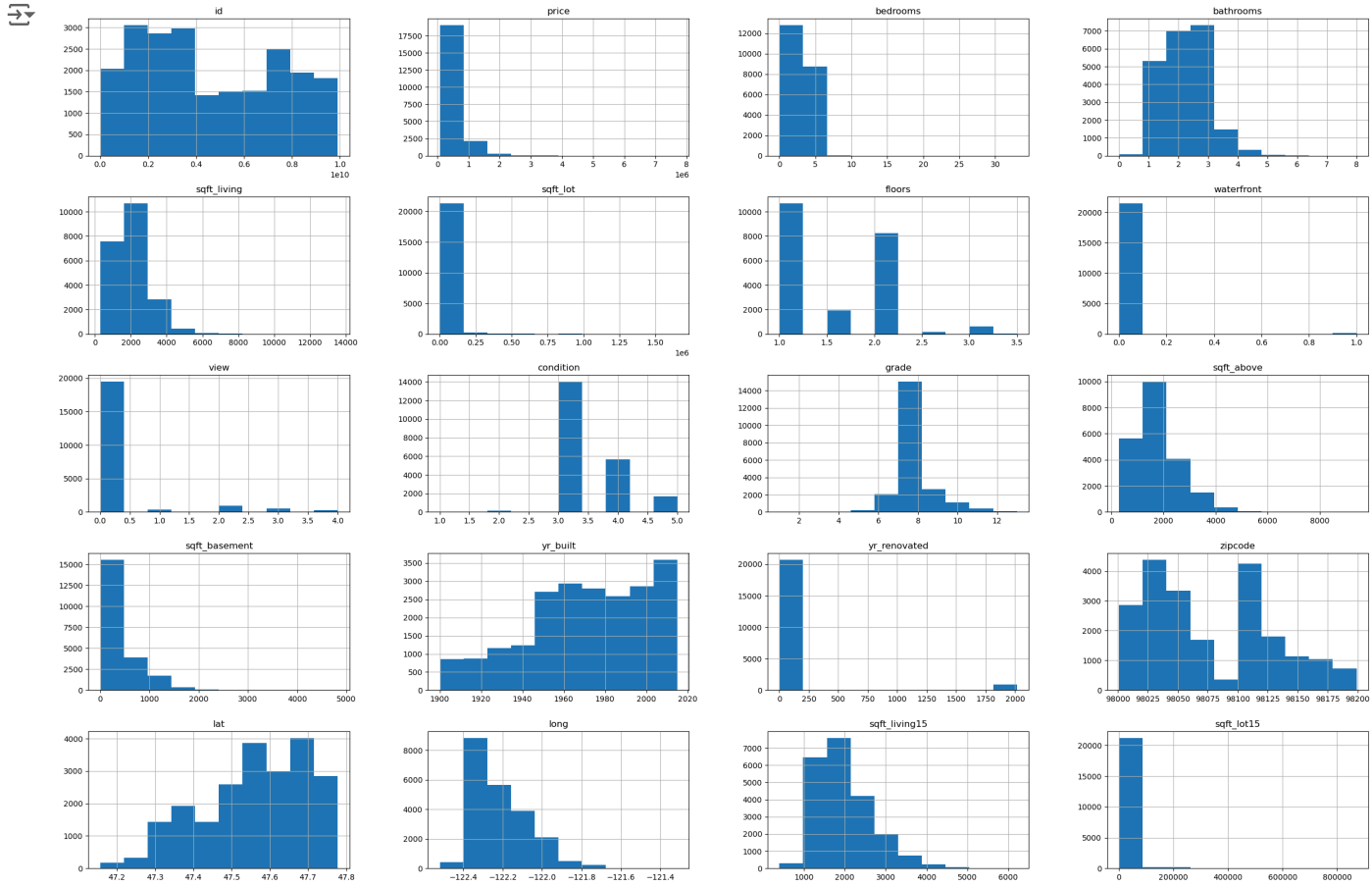
```
array([ 7,  6,  8, 11,  9,  5, 10, 12,  4,  3, 13,  1], dtype=int64)
```

```
data.grade.value_counts()
```

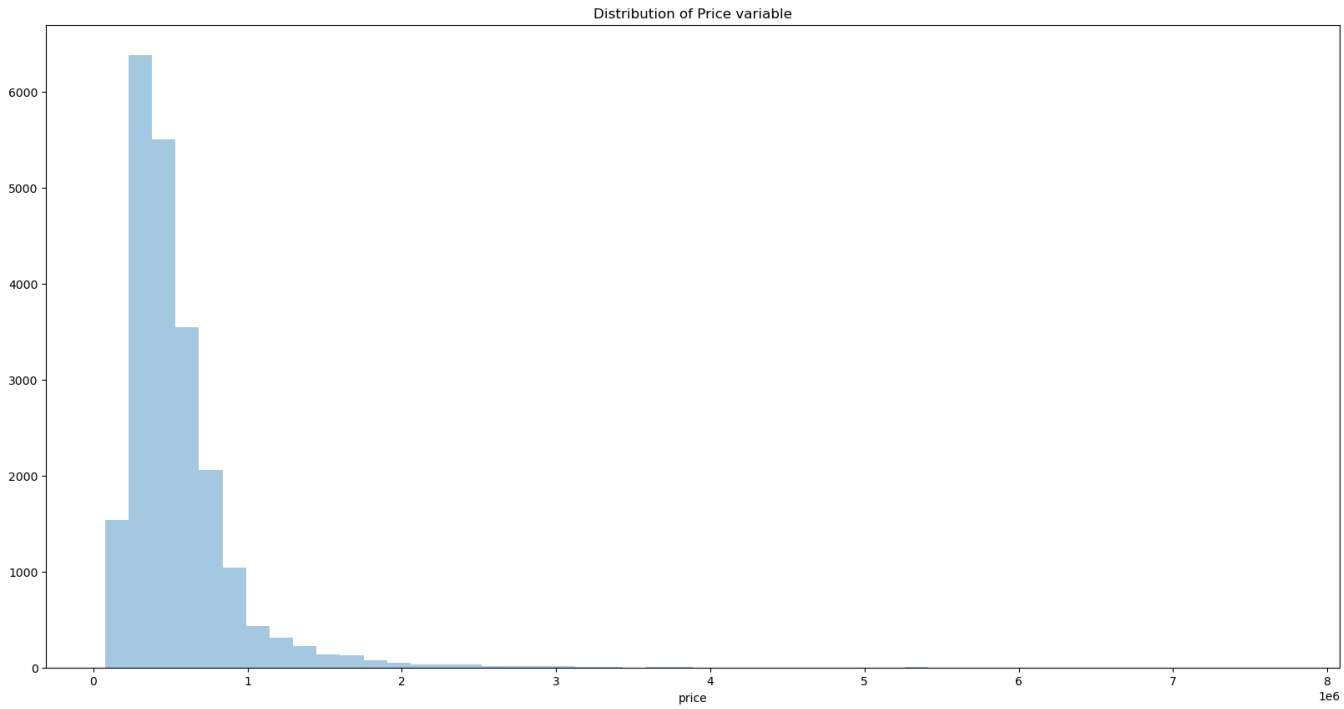
```
7     8981
8     6068
9     2615
6     2038
```

```
10    1134
11     399
5      242
12      90
4       29
13      13
3        3
1        1
Name: grade, dtype: int64
```

```
#data distribution of columns
data.hist(figsize=(30,20))
plt.show()
```

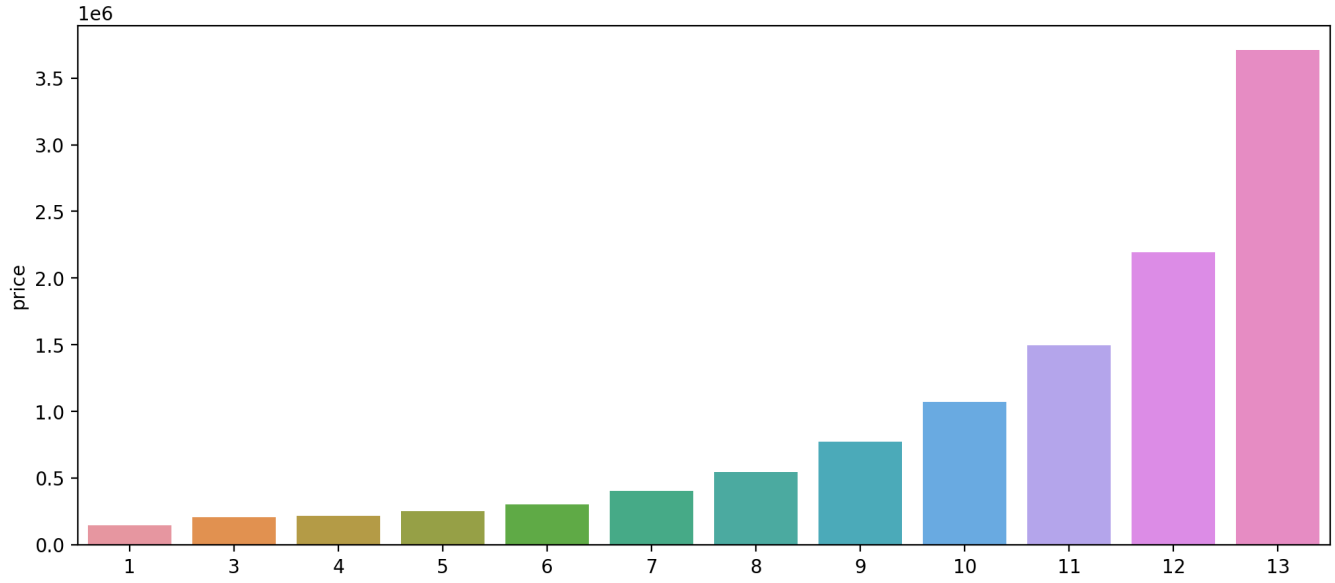



```
# visualizing the price distribution
plt.figure(figsize=(20,10))
sns.distplot(data.price, kde=False)
plt.title('Distribution of Price variable')
plt.show()
```




```
#lets see if the grades affect the price of the house
grade_price_data=({'grades' : sorted(data.grade.unique()),'Average_price' : data.groupby('grade')['price'].mean()})
plt.figure(figsize = (12,5),dpi=200)
sns.barplot(x='grades',y='Average_price',data=grade_price_data)
```

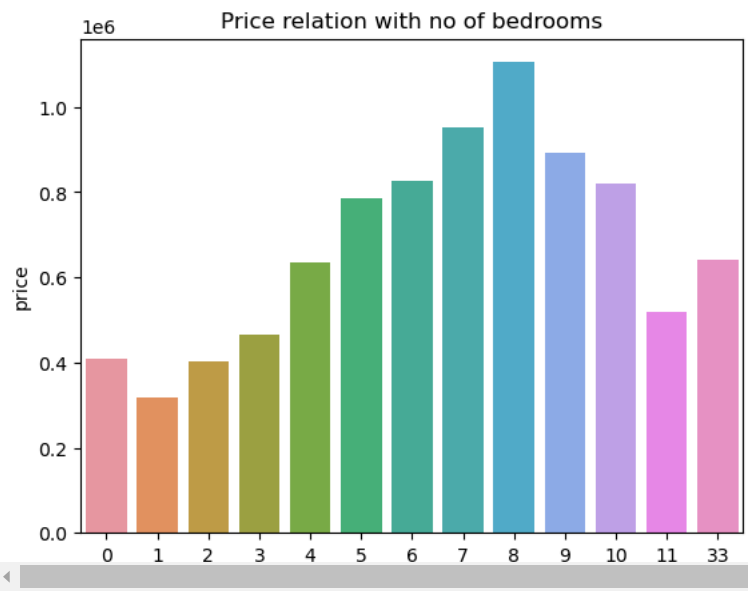
```
<Axes: ylabel='price'>
```



we can clearly see that the grades affect the prcicing of house

```
bedroom_price = ({'Bedrooms': sorted(data.bedrooms.unique()),'Average_price': data.groupby('bedrooms')['price'].mean()})
```

```
sns.barplot(x='Bedrooms',y='Average_price',data=bedroom_price)
plt.title('Price relation with no of bedrooms')
plt.show()
```



we can clearly see that the no of bedrooms doesnt affect the price of houses, but houses with 8 bedrooms is the most expensive one

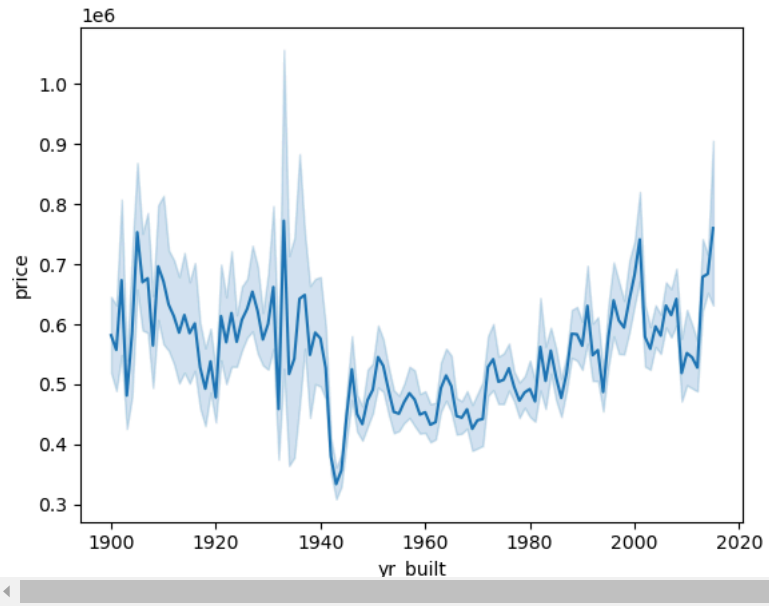
```
data.yr_built.unique()
```

```
array([1955, 1951, 1933, 1965, 1987, 2001, 1995, 1963, 1960, 2003, 1942,
       1927, 1977, 1900, 1979, 1994, 1916, 1921, 1969, 1947, 1968, 1985,
       1941, 1915, 1909, 1948, 2005, 1929, 1981, 1930, 1904, 1996, 2000,
       1984, 2014, 1922, 1959, 1966, 1953, 1950, 2008, 1991, 1954, 1973,
       1925, 1989, 1972, 1986, 1956, 2002, 1992, 1964, 1952, 1961, 2006,
       1988, 1962, 1939, 1946, 1967, 1975, 1980, 1910, 1983, 1978, 1905,
       1971, 2010, 1945, 1924, 1990, 1914, 1926, 2004, 1923, 2007, 1976,
       1949, 1999, 1901, 1993, 1920, 1997, 1943, 1957, 1940, 1918, 1928,
```

```
       1974, 1911, 1936, 1937, 1982, 1908, 1931, 1998, 1913, 2013, 1907,
       1958, 2012, 1912, 2011, 1917, 1932, 1944, 1902, 2009, 1903, 1970,
       2015, 1934, 1938, 1919, 1906, 1935], dtype=int64)
```

```
#checking if the year in which the house was built has any affect on the price
sns.lineplot(x='yr_built',y='price',data=data)
```

```
<Axes: xlabel='yr_built', ylabel='price'>
```



By visualizing the graph we can see that there is not much diiference in the price for houses which were built in 1900 and 2020, so from this we can say that the build year does not have a major impact on the house prices

```
data["date"]= pd.to_datetime(data.date)
```

```
data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_bas
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

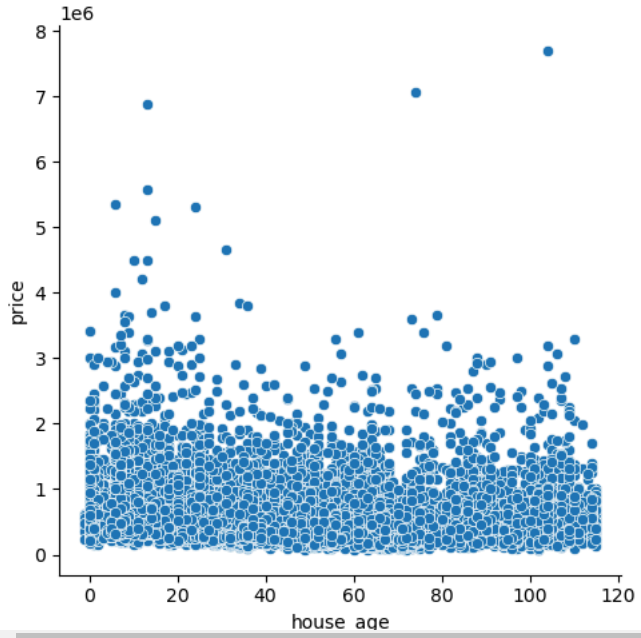
5 rows × 21 columns

```
age_of_house = [data['date'][index].year - data['yr_built'][index] for index in range(data.shape[0])]
```

```
data["house_age"] = age_of_house
```

```
sns.relplot(x='house_age',y='price',data=data
           )
```

```
<seaborn.axisgrid.FacetGrid at 0x1e35e174e50>
```

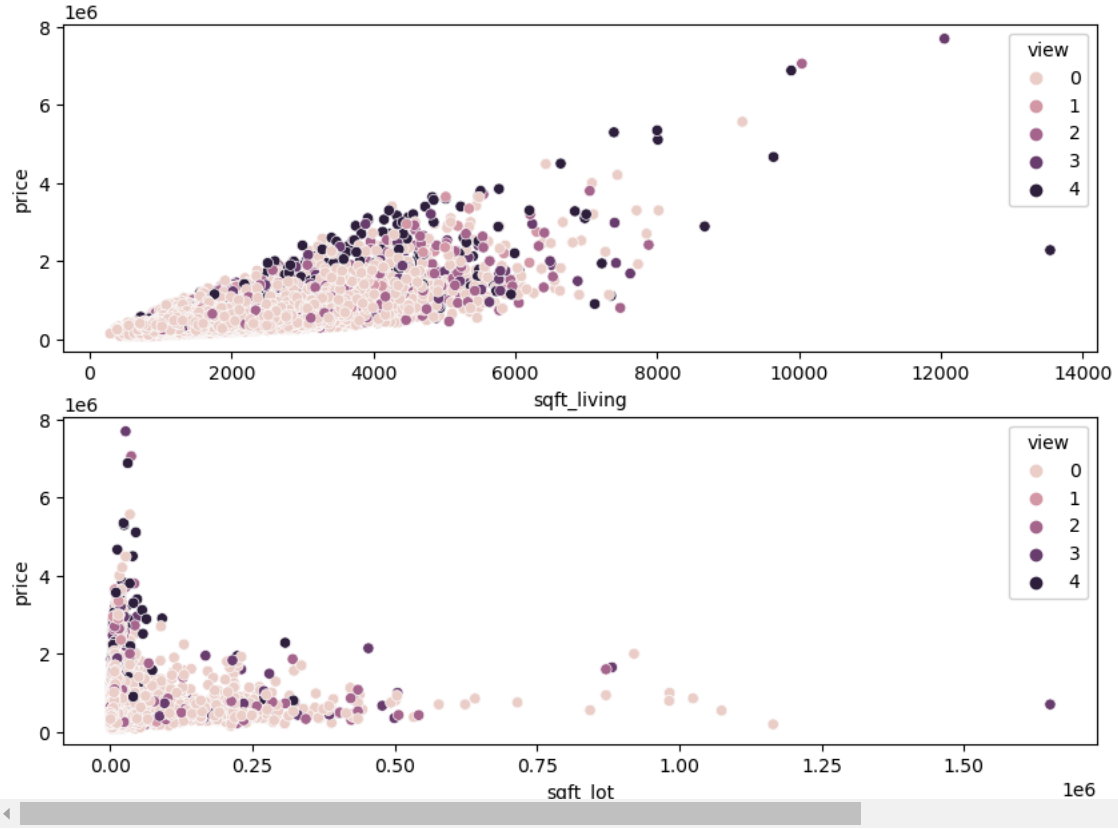



```
plt.figure(figsize =(10 , 7))

plt.subplot(2,1,1)
sns.scatterplot(data =data,x = 'sqft_living',y= 'price', hue ="view")

plt.subplot(2,1,2)
sns.scatterplot(data =data,x = 'sqft_lot',y= 'price', hue ="view")
```

```
<Axes: xlabel='sqft_lot', ylabel='price'>
```



its given that more the living are the price will be higher

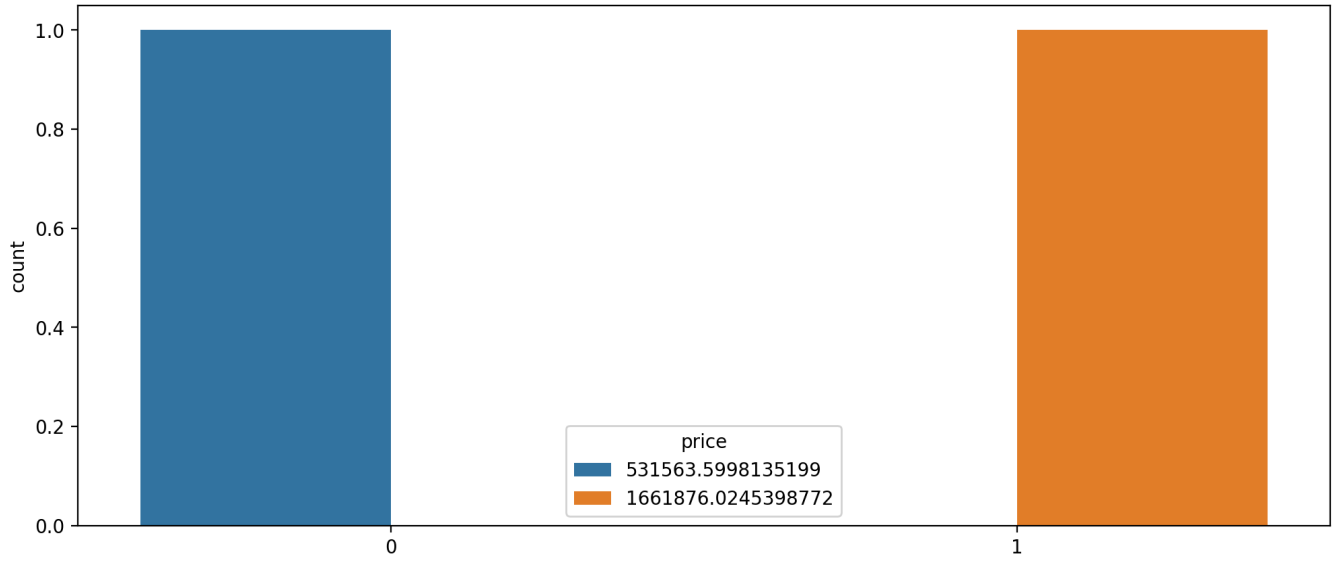
```
data.waterfront.unique()
```

```
array([0, 1], dtype=int64)
```

```
water_front_prices = ({'waterfront':sorted(data.waterfront.unique()),'Average': data.groupby('waterfront')['price'].mean()})
```

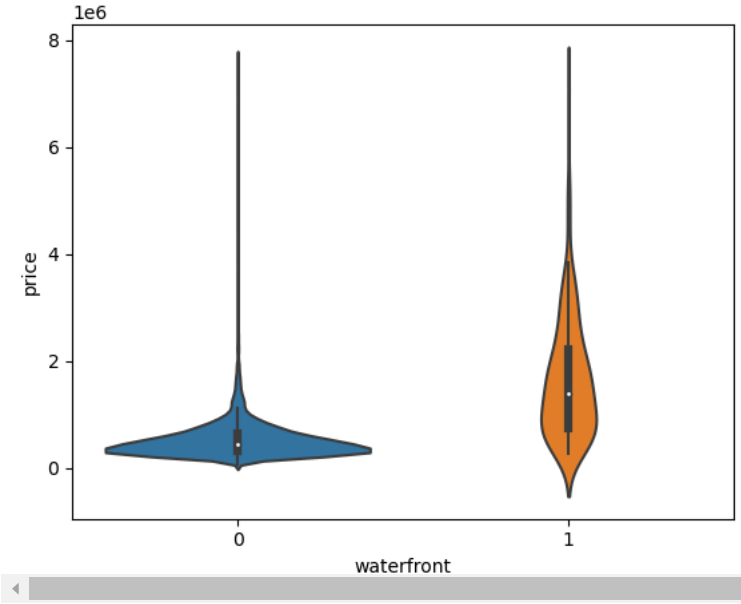
```
#checking if the watefront affects the price of houses
plt.figure(figsize=(12,5),dpi=200)
sns.countplot(x='waterfront',hue='Average',data=water_front_prices)
```

```
<Axes: ylabel='count'>
```



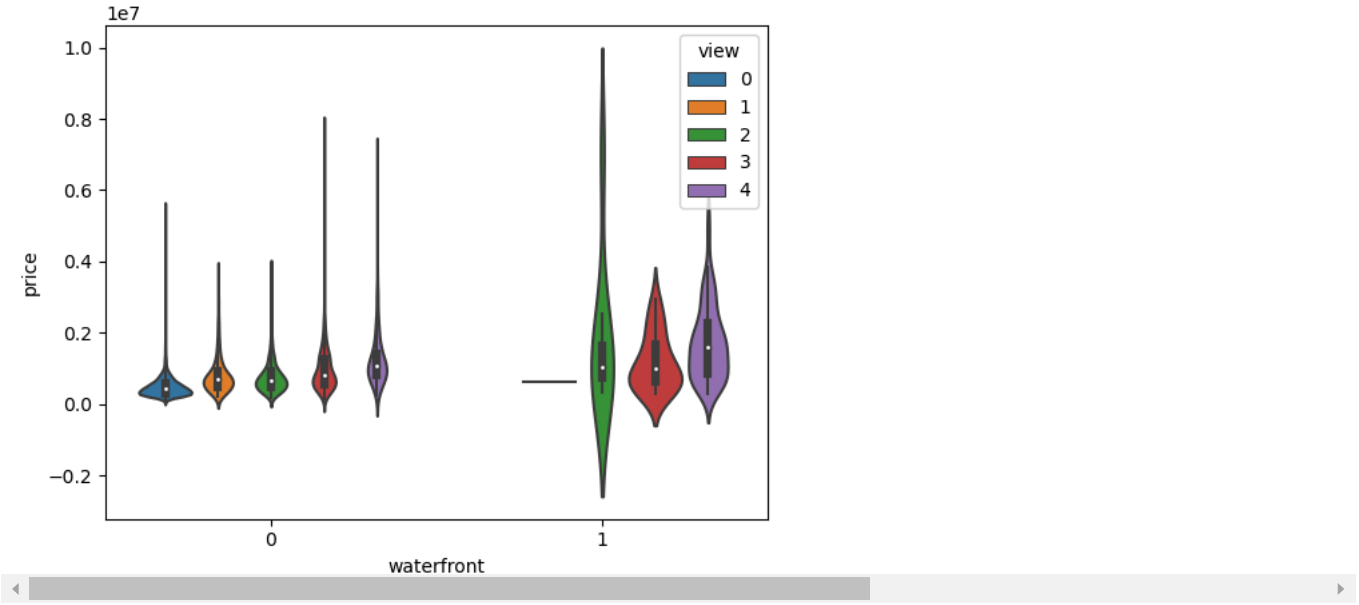

```
sns.violinplot(data =data,x = "waterfront" ,y ="price")
```

```
<Axes: xlabel='waterfront', ylabel='price'>
```



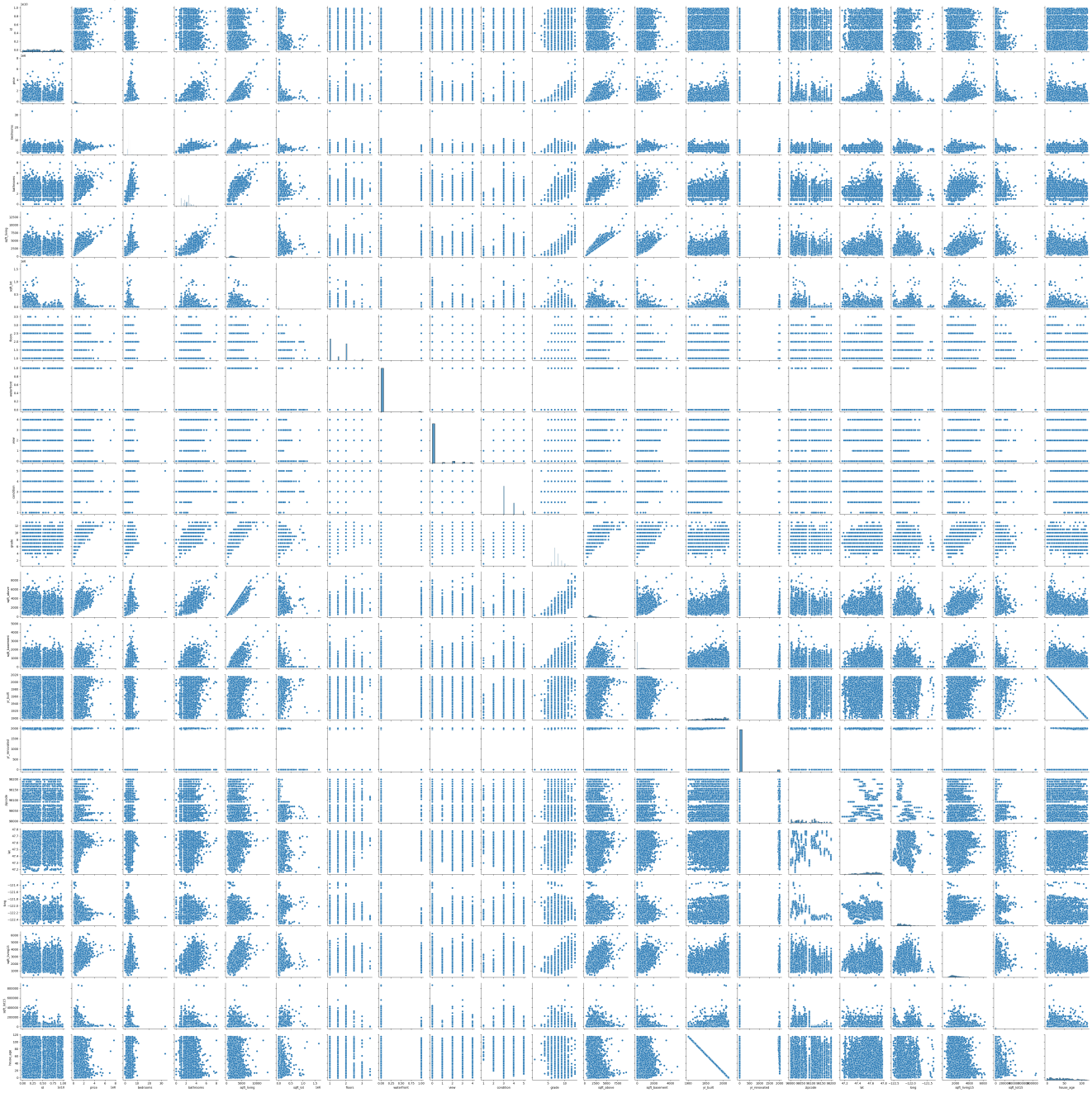

```
sns.violinplot(data =data,x = "waterfront" ,y ="price" , hue ="view")
```

```
<Axes: xlabel='waterfront', ylabel='price'>
```



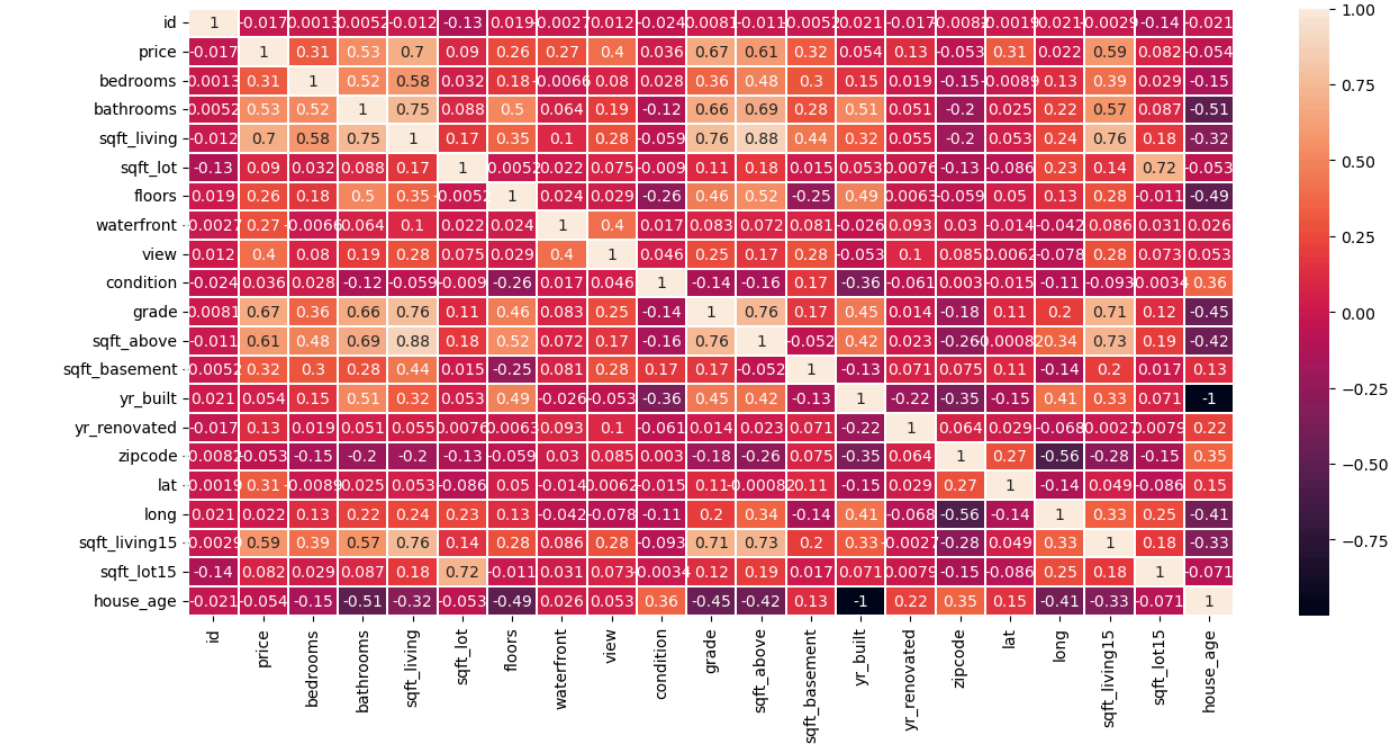

```
sns.pairplot(data=data)
```

```
<seaborn.axisgrid.PairGrid at 0x1e35eed72d0>
```



```
plt.figure(figsize =(15,7))
sns.heatmap(data.corr() , annot =True , linewidth =0.2)
```

```
<Axes: >
```




```
X = data[['bedrooms','bathrooms','sqft_living','sqft_lot','floors','waterfront','view','condition',
       'grade','sqft_above','sqft_basement','sqft_living15','sqft_lot15']]
Y = data[['price']]
```

```
print(X.shape)
print(Y.shape)
```

```
(21613, 13)
(21613, 1)
```

```
X.head()
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_basement	sqft_living15
0	3	1.00	1180	5650	1.0	0	0	3	7	1180	0	1340
1	3	2.25	2570	7242	2.0	0	0	3	7	2170	400	1690
2	2	1.00	770	10000	1.0	0	0	3	6	770	0	2720
3	4	3.00	1960	5000	1.0	0	0	5	7	1050	910	1360
4	3	2.00	1680	8080	1.0	0	0	3	8	1680	0	1800

```
Y.head()
```

	price
0	221900.0
1	538000.0
2	180000.0
3	604000.0
4	510000.0

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test = train_test_split(X,Y , test_size = 0.2 , random_state = 5)
```

```
print('Training data shape:  ',x_train.shape)
print('Training label sahpe: ',y_train.shape)
print('Testing data shape: ',x_test.shape)
print('Testing label shape: ',y_test.shape)
```

```
Training data shape:   (17290, 13)
Training label sahpe:  (17290, 1)
Testing data shape:  (4323, 13)
Testing label shape:  (4323, 1)
```

```
from sklearn.preprocessing import StandardScaler

std = StandardScaler()
X_scaled = std.fit_transform(X)
```

model training and evaluation

1) linear regression

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train,y_train)
```

```
LinearRegression
LinearRegression()
```

```
print('Training Set Score : ', regressor.score(x_train, y_train))
print('Testing Set Score  : ', regressor.score(x_test, y_test))
```

```
Training Set Score :  0.6059397767708085
Testing Set Score  :  0.6111480391176347
```

```
from sklearn.metrics import r2_score

pred_train = regressor.predict(x_train)
pred_test  = regressor.predict(x_test)

print('Training Set Score : ', r2_score(y_train, pred_train))
print('Testing Set Score  : ', r2_score(y_test, pred_test))
```

```
Training Set Score :  0.6059397767708085
Testing Set Score  :  0.6111480391176347
```

```
from sklearn.metrics import mean_squared_error,mean_absolute_error


mse_lr = mean_squared_error(y_test,pred_test)
mae_lr = mean_absolute_error(y_test,pred_test)
rmse_lr = np.sqrt(mean_squared_error(y_test,pred_test))


print('MSE: ',mse_lr)
print('MAE: ',mae_lr)
print('RMSE: ',rmse_lr)
```

```
MSE:  53051681887.01648
MAE:  152478.9605315535
RMSE:  230329.50720004694
```

2)polynomial regression

```
from sklearn.preprocessing import PolynomialFeatures


poly = PolynomialFeatures(degree = 4)
x_train_poly = poly.fit_transform(x_train)
x_test_poly = poly.transform(x_test)


from sklearn.linear_model import LinearRegression
regressor2 = LinearRegression()
regressor2.fit(x_train_poly,y_train)
```