1 Real estate market analyze at King County with Regression modeling.

Jiajie Xu, 07Nov21

2 Overview

2.1 Business Problem

Data analysis for King County's house sale record, using 2014-2015 house sales data from online. In order to invest new real estate, what features should future owner consider most? and what should owner don't care too much? Should owner buy a larger room than he/she actually needs?

3 Data

3.1 Data Loading/ Understanding

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import copy
import seaborn as sns
# import statsmodels.api as sm
# from statsmodels.formula.api import ols
import statsmodels.api as sm
import scipy.stats as stats
from statsmodels.formula.api import ols
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn import datasets, linear_model
from statsmodels.stats.outliers_influence import variance_inflation_factor
linreg = LinearRegression()
from sklearn.model_selection import KFold
from itertools import combinations
import matplotlib.pyplot as plt
from matplotlib.pyplot import hist
from matplotlib import style
style.use('ggplot')
%matplotlib inline
```

In [2]:
```
ls
```

```
 Volume in drive D is New Volume
 Volume Serial Number is C649-A6AD

 Directory of D:\Flatiron\Project2\dsc-phase-2-project

11/08/2021  08:58 PM    <DIR>          .
10/05/2021  05:13 PM    <DIR>          ..
10/05/2021  05:13 PM               152 .canvas
10/05/2021  05:13 PM                70 .gitignore
10/28/2021  12:34 PM    <DIR>          .ipynb_checkpoints
10/05/2021  05:13 PM             1,846 CONTRIBUTING.md
10/05/2021  07:17 PM    <DIR>          data
10/05/2021  05:13 PM         2,930,391 halfway-there.gif
10/05/2021  05:13 PM             1,354 LICENSE.md
11/08/2021  08:58 PM         1,265,203 Phase_2_project.ipynb
10/05/2021  05:13 PM             4,151 README.md
               7 File(s)      4,203,167 bytes
               4 Dir(s)  1,999,968,514,048 bytes free
```

In [3]:
```
cd data/
```

```
D:\Flatiron\Project2\dsc-phase-2-project\data
```

In [4]:
```
df = pd.read_csv('kc_house_data.csv')
```

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

Out[5]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront
0	7129300520	10/13/2014	221900.0	3	1.00	1180	5650	1.0	NaN
1	6414100192	12/9/2014	538000.0	3	2.25	2570	7242	2.0	0.0
2	5631500400	2/25/2015	180000.0	2	1.00	770	10000	1.0	0.0
3	2487200875	12/9/2014	604000.0	4	3.00	1960	5000	1.0	0.0
4	1954400510	2/18/2015	510000.0	3	2.00	1680	8080	1.0	0.0

5 rows × 21 columns

3.2 Data Preparation

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

Out[6]:
```
id                 int64
date              object
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront       float64
view             float64
condition          int64
grade              int64
sqft_above         int64
sqft_basement     object
yr_built           int64
yr_renovated     float64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [7]:
```
df.isna().sum().sum()
```

Out[7]:
```
6281
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21597 entries, 0 to 21596
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   id             21597 non-null  int64  
 1   date           21597 non-null  object 
 2   price          21597 non-null  float64
 3   bedrooms       21597 non-null  int64  
 4   bathrooms      21597 non-null  float64
 5   sqft_living    21597 non-null  int64  
 6   sqft_lot       21597 non-null  int64  
 7   floors         21597 non-null  float64
 8   waterfront     19221 non-null  float64
 9   view           21534 non-null  float64
 10  condition      21597 non-null  int64  
 11  grade          21597 non-null  int64  
 12  sqft_above     21597 non-null  int64  
 13  sqft_basement  21597 non-null  object 
 14  yr_built       21597 non-null  int64  
 15  yr_renovated   17755 non-null  float64
 16  zipcode        21597 non-null  int64  
 17  lat            21597 non-null  float64
 18  long           21597 non-null  float64
 19  sqft_living15  21597 non-null  int64  
 20  sqft_lot15     21597 non-null  int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 3.5+ MB
```

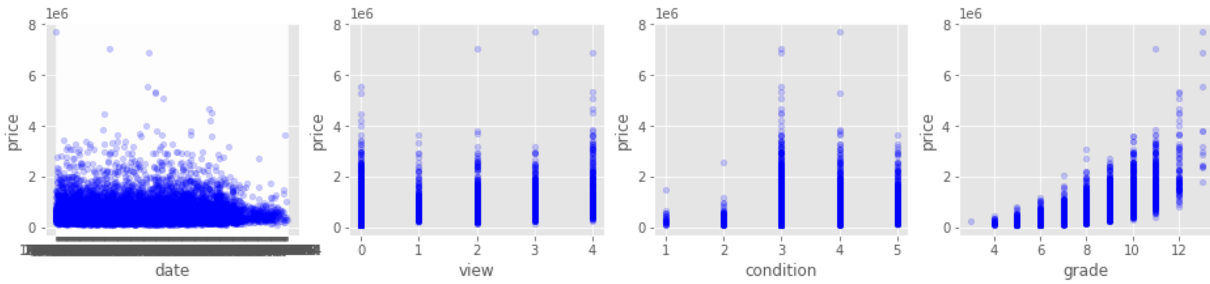
Missing value in waterfront and yr_renovated. Usually, value in such features would not be forgotten since it infulence the sale price. Therefore, use 0 replace the NaN should be fine to modify the data set.

In [9]:
```
# For the whole DataFrame using pandas to change NaN to 0
df = df.fillna(0)
```

3.2.1 Review each colunm and clean data

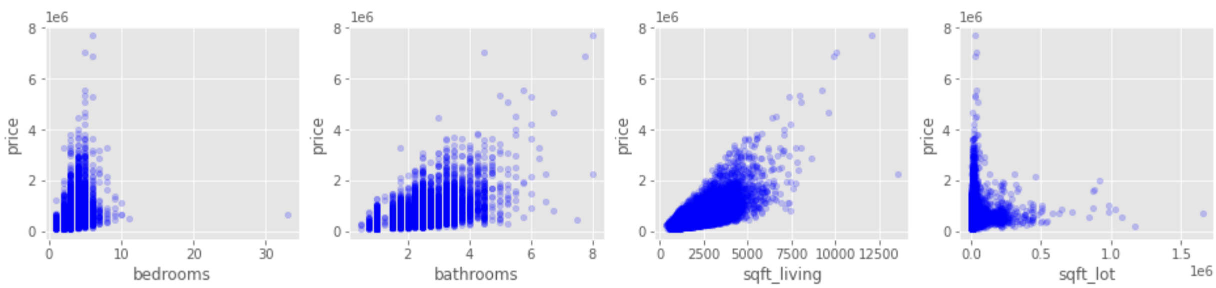
In [10]:
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(['date', 'view', 'condition', 'grade' ], axes):
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```



In [11]:
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot'], axes)
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```



In [12]:
```
df['bedrooms'].describe()
```

Out[12]:
```
count    21597.000000
mean         3.373200
std          0.926299
min          1.000000
25%          3.000000
50%          3.000000
75%          4.000000
max         33.000000
Name: bedrooms, dtype: float64
```

In [13]:
```
df[df['bedrooms'] == df['bedrooms'].max()]
```

Out[13]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
15856	2402100895	6/25/2014	640000.0	33	1.75	1620	6000	1.0	

1 rows × 21 columns

The row number 15856's bedroom is too high to be counted, so delete this line is better for the whole model.

In [14]:
```
df = df.drop(df.index[15856])
```

In [15]:
```
df[df['sqft_lot'] == df['sqft_lot'].max()]
```

Out[15]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
1717	1020069017	3/27/2015	700000.0	4	1.0	1300	1651359	1.0	

1 rows × 21 columns

In [16]:
```
# row 1717's sqft_lot(1651359) is too hign as well, drop this line.
df = df.drop(df.index[1717])
```

In [17]:
```
df[df['sqft_living'] == df['sqft_living'].max()]
```

Out[17]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfr
12764	1225069038	5/5/2014	2280000.0	7	8.0	13540	307752	3.0	

1 rows × 21 columns

In [18]:
```
# row 12764's price(1651359) is too low as well, drop this line.
df.drop([12764], axis = 0, inplace=True)
```

In [19]:
```
df[df['sqft_living'] == df['sqft_living'].max()]
```

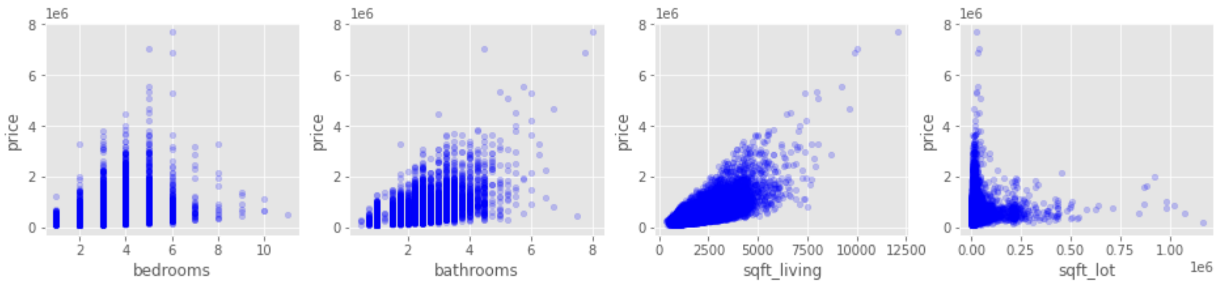
Out[19]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
7245	6762700020	10/13/2014	7700000.0	6	8.0	12050	27600	2.5	

1 rows × 21 columns

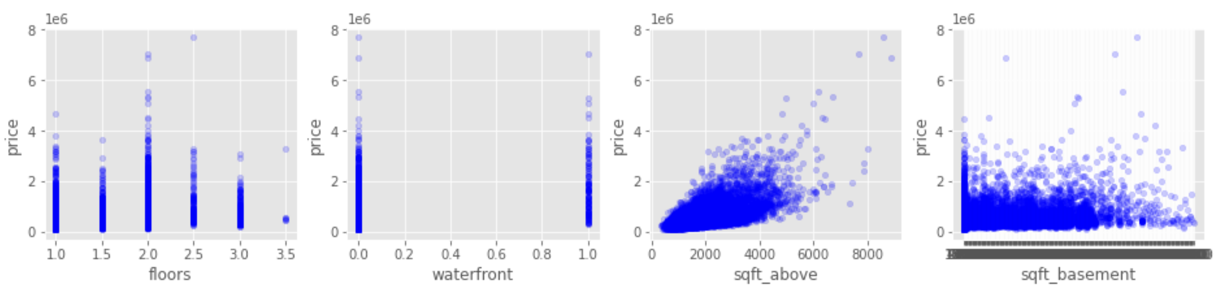
In [20]:
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot'], axes)
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```



In [21]:
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(['floors', 'waterfront', 'sqft_above', 'sqft_basement'], ax
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```



Preprocessing for 'sqft_basement'

In [22]:
```
(df['sqft_basement'] == '?').sum()
```

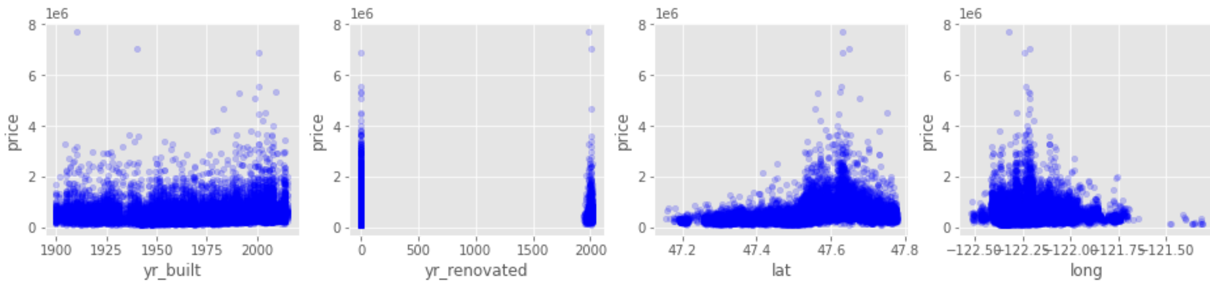
Out[22]: 454

In [23]:
```
# Using replace() function for a single column, change '?' to 0
df['sqft_basement'] = df['sqft_basement'].replace('?', 0)
```

In [24]:
```
df['sqft_basement'] = df['sqft_basement'].fillna(0)
df['sqft_basement'] = pd.to_numeric(df['sqft_basement']).astype(np.int64)
```

In [25]:
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(16,3))

for xcol, ax in zip(['yr_built', 'yr_renovated', 'lat', 'long'], axes):
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```



3.2.2 Data scalling.

In [26]:
```
#rescaling 'yr_built'
df['yr_built'] = np.log(df['yr_built'])
```

In [27]:
```
# preprocessing for rescaling
x_cols = ['sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condition', 'sqft
for col in x_cols:
    df[col] = (df[col] - df[col].mean())/df[col].std()
df.head()
```

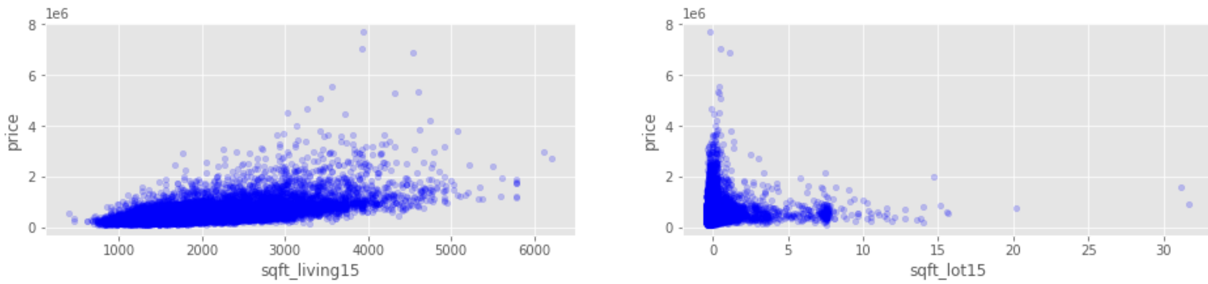
Out[27]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wate
0	7129300520	10/13/2014	221900.0	3	1.00	-0.983623	-0.234949	-0.915624	-0.0
1	6414100192	12/9/2014	538000.0	3	2.25	0.535783	-0.194990	0.937594	-0.0
2	5631500400	2/25/2015	180000.0	2	1.00	-1.431793	-0.125764	-0.915624	-0.0
3	2487200875	12/9/2014	604000.0	4	3.00	-0.131006	-0.251264	-0.915624	-0.0
4	1954400510	2/18/2015	510000.0	3	2.00	-0.437074	-0.173956	-0.915624	-0.0

5 rows × 21 columns

In [28]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16,3))

for xcol, ax in zip(['sqft_living15', 'sqft_lot15' ], axes):
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```



In [29]:
```
df[df['sqft_lot15'] == df['sqft_lot15'].max()]
```

Out[29]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	water
9705	225079036	1/7/2015	937500.0	4	4.0	3.78775	21.490364	0.937594	-0.08

1 rows × 21 columns

3.2.3 Data filtering

In [30]:
```
df['date']
```

Out[30]:
```
0        10/13/2014
1         12/9/2014
2         2/25/2015
3         12/9/2014
4         2/18/2015
            ...
21592     5/21/2014
21593     2/23/2015
21594     6/23/2014
21595     1/16/2015
21596    10/15/2014
Name: date, Length: 21594, dtype: object
```

Sale date group all within one year, which doesn't influence much. Thus, it would be appropriate to drop 'date' column.

In [31]:
```
# So does 'view'
print(df['view'].describe())
```

```
count    21594.000000
mean         0.232889
std          0.764062
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          4.000000
Name: view, dtype: float64
```

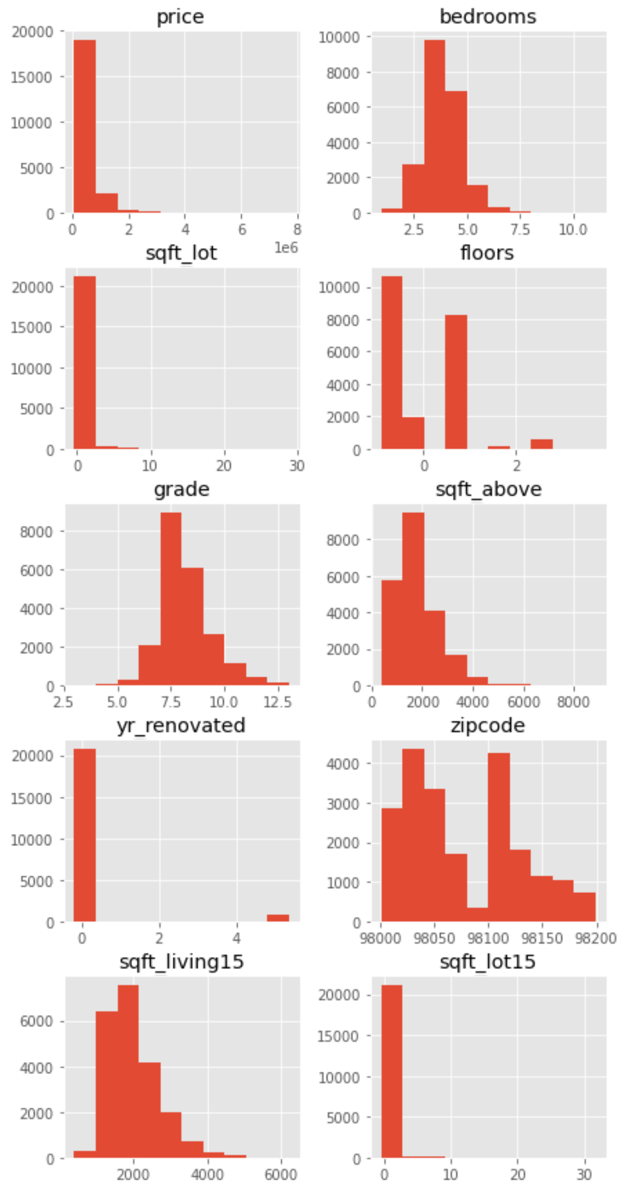
In [32]:
```
df = df.drop(['id', 'date', 'view'], axis=1)
```

In [33]:
```
df.head()
```

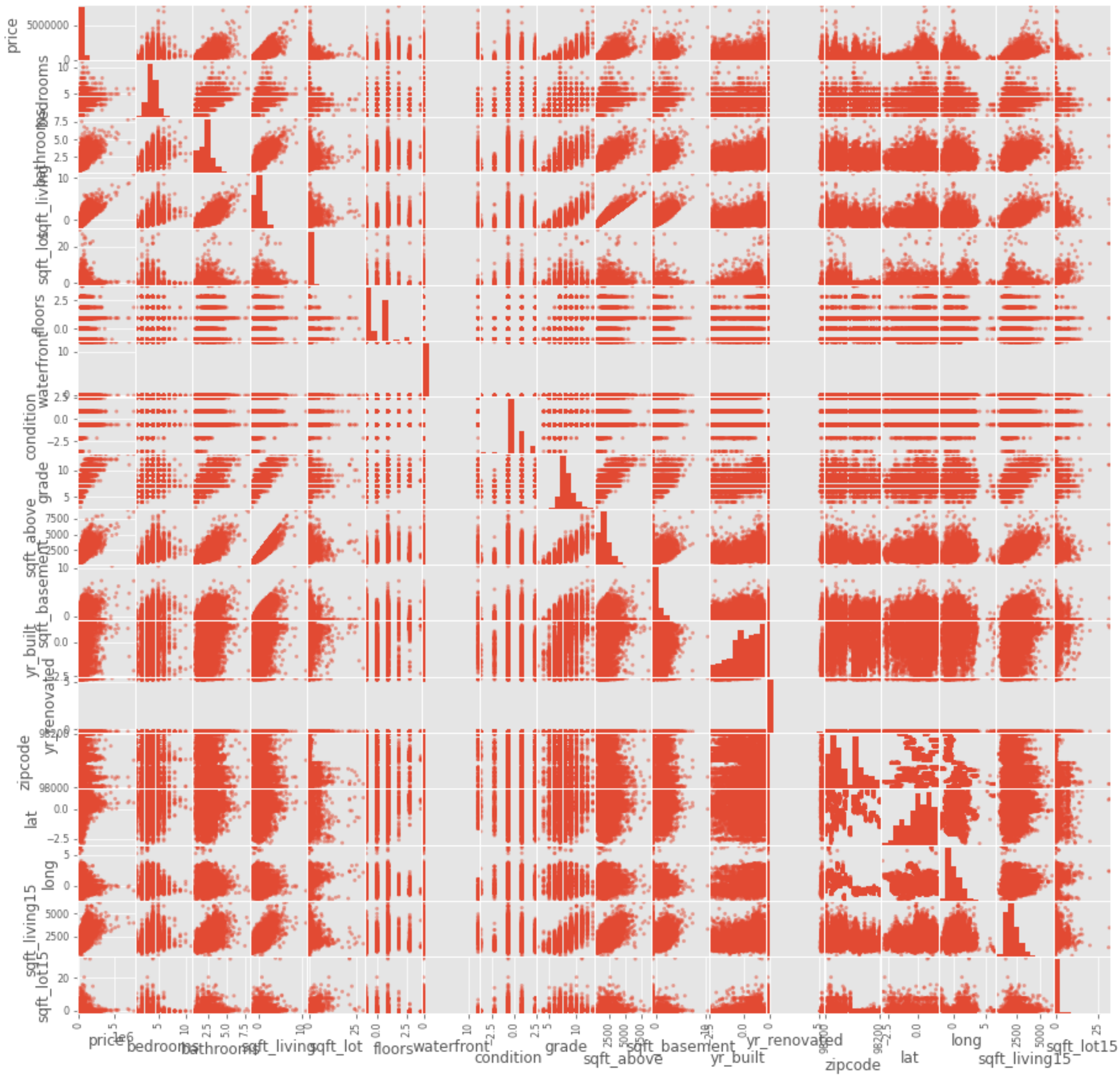
Out[33]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	grade
0	221900.0	3	1.00	-0.983623	-0.234949	-0.915624	-0.082504	-0.629907	7
1	538000.0	3	2.25	0.535783	-0.194990	0.937594	-0.082504	-0.629907	7
2	180000.0	2	1.00	-1.431793	-0.125764	-0.915624	-0.082504	-0.629907	6
3	604000.0	4	3.00	-0.131006	-0.251264	-0.915624	-0.082504	2.444734	7
4	510000.0	3	2.00	-0.437074	-0.173956	-0.915624	-0.082504	-0.629907	8

In [34]:
```
fig = plt.figure(figsize = (15,15))
ax = fig.gca()
df.hist(ax = ax);
```



In [35]:
```
pd.plotting.scatter_matrix(df,figsize  = [15, 15]);
plt.show()
```



In [36]:

```
df.corr()
```

Out[36]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	cond
price	1.000000	0.315353	0.525279	0.702124	0.091042	0.256412	0.264466	0.03
bedrooms	0.315353	1.000000	0.527438	0.593267	0.032256	0.183325	-0.002037	0.02
bathrooms	0.525279	0.527438	1.000000	0.755074	0.092147	0.502330	0.063741	-0.12
sqft_living	0.702124	0.593267	0.755074	1.000000	0.178322	0.353646	0.105061	-0.05
sqft_lot	0.091042	0.032256	0.092147	0.178322	1.000000	-0.004223	0.022491	-0.01
floors	0.256412	0.183325	0.502330	0.353646	-0.004223	1.000000	0.020805	-0.26
waterfront	0.264466	-0.002037	0.063741	0.105061	0.022491	0.020805	1.000000	0.01
condition	0.036171	0.023538	-0.126340	-0.059213	-0.010666	-0.263954	0.016661	1.00
grade	0.667773	0.365805	0.665604	0.763630	0.120733	0.458503	0.082856	-0.14
sqft_above	0.604894	0.492010	0.685677	0.876006	0.189777	0.523894	0.071950	-0.15
sqft_basement	0.319936	0.302569	0.276242	0.425907	0.013919	-0.243484	0.083232	0.16
yr_built	0.052578	0.160747	0.505896	0.317840	0.058527	0.485390	-0.024367	-0.36
yr_renovated	0.117965	0.018674	0.047294	0.051344	0.005597	0.003803	0.073937	-0.05
zipcode	-0.053315	-0.158534	-0.204978	-0.200298	-0.132561	-0.059520	0.028920	0.00
lat	0.306770	-0.011669	0.023904	0.051832	-0.084649	0.049096	-0.012162	-0.01
long	0.021678	0.136283	0.224745	0.241218	0.236214	0.125799	-0.037624	-0.10
sqft_living15	0.584886	0.404053	0.569493	0.757083	0.147527	0.279741	0.083876	-0.09
sqft_lot15	0.081479	0.030220	0.087351	0.182456	0.720024	-0.011140	0.030949	-0.00

We set 0.75 as a cut-off range for correlations between variables.

In [37]:
```
abs(df.corr()) > 0.75
```

Out[37]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	condition	gra
price	True	False	False	False	False	False	False	False	Fa
bedrooms	False	True	False	False	False	False	False	False	Fa
bathrooms	False	False	True	True	False	False	False	False	Fa
sqft_living	False	False	True	True	False	False	False	False	T
sqft_lot	False	False	False	False	True	False	False	False	Fa
floors	False	False	False	False	False	True	False	False	Fa
waterfront	False	False	False	False	False	False	True	False	Fa
condition	False	False	False	False	False	False	False	True	Fa
grade	False	False	False	True	False	False	False	False	T
sqft_above	False	False	False	True	False	False	False	False	T
sqft_basement	False	False	False	False	False	False	False	False	Fa
yr_built	False	False	False	False	False	False	False	False	Fa
yr_renovated	False	False	False	False	False	False	False	False	Fa
zipcode	False	False	False	False	False	False	False	False	Fa
lat	False	False	False	False	False	False	False	False	Fa
long	False	False	False	False	False	False	False	False	Fa
sqft_living15	False	False	False	True	False	False	False	False	Fa
sqft_lot15	False	False	False	False	False	False	False	False	Fa

It seems like the column 'sqft_living', 'bathrooms', 'grade', 'sqft_above', and 'sqft_living15'are all pretty highly correlated among each other.

In [38]:
```
# save absolute value of correlation matrix as a data frame
# converts all values to absolute value
# stacks the row:column pairs into a multindex
# reset the index to set the multindex to seperate columns
# sort values. 0 is the column automatically generated by the stacking

df2 = df.corr().abs().stack().reset_index().sort_values(0, ascending=False)

# zip the variable name columns (Which were only named level_0 and level_1 by d
df2['pairs'] = list(zip(df2.level_0, df2.level_1))

# set index to pairs
df2.set_index(['pairs'], inplace = True)

#d rop level columns
df2.drop(columns=['level_1', 'level_0'], inplace = True)

# rename correlation column as cc rather than 0
df2.columns = ['cc']

# drop duplicates. This could be dangerous if you have variables perfectly corr
# for the sake of exercise, kept it in.
df2.drop_duplicates(inplace=True)
```

In [39]:
```
df2[(df2.cc>.75) & (df2.cc <1)]
```

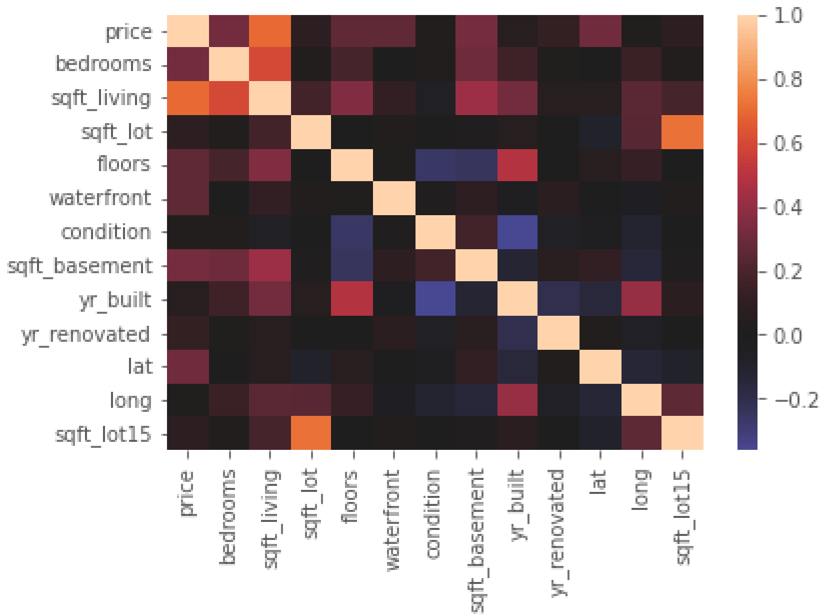
Out[39]:

	cc
pairs	
(sqft_living, sqft_above)	0.876006
(sqft_living, grade)	0.763630
(sqft_living15, sqft_living)	0.757083
(sqft_above, grade)	0.756219
(bathrooms, sqft_living)	0.755074

'bathrooms', 'grade', 'sqft_above', 'sqft_living15' have strong corralation with 'sqft_living'. Therefore, drop these four. Besidea, 'zipcode'provide less info compares to 'long' and 'lat', drop 'Zipcode' as well.

In [40]:
```
df = df.drop(['bathrooms', 'grade', 'sqft_above', 'sqft_living15', 'zipcode'],
```

In [41]:
```
# Heatmap to render the correlation matrix as a visualization.
sns.heatmap(df.corr(), center=0);
```



In [42]:
```
df.head()
```

Out[42]:

	price	bedrooms	sqft_living	sqft_lot	floors	waterfront	condition	sqft_basement	y
0	221900.0	3	-0.983623	-0.234949	-0.915624	-0.082504	-0.629907	-0.650336	-0.5
1	538000.0	3	0.535783	-0.194990	0.937594	-0.082504	-0.629907	0.260696	-0.6
2	180000.0	2	-1.431793	-0.125764	-0.915624	-0.082504	-0.629907	-0.650336	-1.2
3	604000.0	4	-0.131006	-0.251264	-0.915624	-0.082504	2.444734	1.422261	-0.1
4	510000.0	3	-0.437074	-0.173956	-0.915624	-0.082504	-0.629907	-0.650336	0.5

A Model Using the Raw Features

In [43]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21594 entries, 0 to 21596
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   price          21594 non-null  float64
 1   bedrooms       21594 non-null  int64
 2   sqft_living    21594 non-null  float64
 3   sqft_lot       21594 non-null  float64
 4   floors         21594 non-null  float64
 5   waterfront     21594 non-null  float64
 6   condition      21594 non-null  float64
 7   sqft_basement  21594 non-null  float64
 8   yr_built       21594 non-null  float64
 9   yr_renovated   21594 non-null  float64
 10  lat            21594 non-null  float64
 11  long           21594 non-null  float64
 12  sqft_lot15     21594 non-null  float64
dtypes: float64(12), int64(1)
memory usage: 2.3 MB
```

Regression Model Validation.

In order to get a good sense of how well your model will be doing on new instances, you'll have to perform a so-called "train-test-split". What you'll be doing here, is take a sample of the data that serves as input to "train" our model - fit a linear regression and compute the parameter estimates for our variables, and calculate how well our predictive performance is doing comparing the actual targets y and the fitted $\hat{y}$ obtained by our model.

3.2.4 Selecting Features Based on p-values

In [44]:
```
outcome = 'price'
x_cols = ['bedrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condi
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=df).fit()
model.summary()
```

Out[44]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.643
Model:	OLS	**Adj. R-squared:**	0.643
Method:	Least Squares	**F-statistic:**	3237.
Date:	Mon, 08 Nov 2021	**Prob (F-statistic):**	0.00
Time:	22:33:02	**Log-Likelihood:**	-2.9622e+05
No. Observations:	21594	**AIC:**	5.925e+05
Df Residuals:	21581	**BIC:**	5.926e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.175e+05	7190.066	99.789	0.000	7.03e+05	7.32e+05
bedrooms	-5.258e+04	2085.989	-25.207	0.000	-5.67e+04	-4.85e+04
sqft_living	2.991e+05	2394.575	124.921	0.000	2.94e+05	3.04e+05
sqft_lot	4552.2779	2167.842	2.100	0.036	303.147	8801.409
floors	1.551e+04	1996.851	7.769	0.000	1.16e+04	1.94e+04
waterfront	6.488e+04	1515.988	42.794	0.000	6.19e+04	6.78e+04
condition	2.322e+04	1648.033	14.089	0.000	2e+04	2.64e+04
sqft_basement	-2.074e+04	1975.313	-10.500	0.000	-2.46e+04	-1.69e+04
yr_built	-4.099e+04	2063.489	-19.866	0.000	-4.5e+04	-3.69e+04
yr_renovated	1.417e+04	1564.342	9.055	0.000	1.11e+04	1.72e+04
lat	8.864e+04	1554.596	57.016	0.000	8.56e+04	9.17e+04
long	-2.634e+04	1771.361	-14.869	0.000	-2.98e+04	-2.29e+04
sqft_lot15	-1.075e+04	2180.951	-4.931	0.000	-1.5e+04	-6479.600

Omnibus:	14997.948	**Durbin-Watson:**	1.994
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	702513.587
Skew:	2.791	**Prob(JB):**	0.00
Kurtosis:	30.379	**Cond. No.**	18.4

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

All features are significant, no p value above 0.05

3.3 Data seperation

3.3.1 Data split into train set and test set

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

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
```

In [47]:
```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
17275 4319 17275 4319
```

In [48]:
```
linreg = LinearRegression()
linreg.fit(X_train, y_train)

y_hat_train = linreg.predict(X_train)
y_hat_test = linreg.predict(X_test)
```

In [49]:
```
y_hat_train
```

Out[49]:
```
array([[ 556846.87720935],
       [ 657800.25341133],
       [ 262017.07969051],
       ...,
       [ 469236.60541336],
       [1102653.84970539],
       [ 689355.08669649]])
```

In [50]:
```
y_hat_test
```

Out[50]:
```
array([[ 491297.61566077],
       [2432373.83509963],
       [ 418184.47775568],
       ...,
       [ 864883.8594524 ],
       [ 488687.90520885],
       [ 579590.95462438]])
```

Look at the residuals and calculate the MSE for training and test sets:

In [51]:
```
train_residuals = y_hat_train - y_train
test_residuals = y_hat_test - y_test
```

In [52]:
```
mse_train = np.sum((y_train-y_hat_train)**2)/len(y_train)
mse_test = np.sum((y_test-y_hat_test)**2)/len(y_test)
print('Train Mean Squarred Error:', mse_train)
print('Test Mean Squarred Error:', mse_test)
```

```
Train Mean Squarred Error: price    4.860660e+10
dtype: float64
Test Mean Squarred Error: price    4.661225e+10
dtype: float64
```

3.3.2 Cross-Validation

The code below repeats a train-test split creation 20 times, using a test_size of 0.33. So what happens is, each time a new (random) train-test split is created. See how training and testing MSEs swing around by just taking another sample!

In [53]:
```
num = 20
train_err = []
test_err = []
for i in range(num):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
    linreg.fit(X_train, y_train)
    y_hat_train = linreg.predict(X_train)
    y_hat_test = linreg.predict(X_test)
    train_err.append(mean_squared_error(y_train, y_hat_train))
    test_err.append(mean_squared_error(y_test, y_hat_test))
plt.scatter(list(range(num)), train_err, label='Training Error')
plt.scatter(list(range(num)), test_err, label='Testing Error')
plt.legend();
```

In [54]:
```
cv_5_results  = np.mean(cross_val_score(linreg, X, y, cv=5,  scoring='neg_mean_
cv_10_results = np.mean(cross_val_score(linreg, X, y, cv=10, scoring='neg_mean_
cv_20_results = np.mean(cross_val_score(linreg, X, y, cv=20, scoring='neg_mean_
```

In [55]:
```
cv_5_results
```

Out[55]: -48615169153.44209

In [56]:
```
cv_10_results
```

Out[56]: -48524218146.92438

In [57]:
```
cv_20_results
```

Out[57]: -48372694517.184746

A Model Using the Raw Features

In [58]:
```
y = df[['price']]
X = df.drop(['price'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
```

In [59]:
```
train = y_train.join(X_train)
```

In [60]:
```
test = y_test.join(X_test)
```

In [61]:
```
outcome = 'price'
x_cols = ['bedrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condi
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=train).fit()
model.summary()
```

Out[61]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.645
Model:	OLS	**Adj. R-squared:**	0.644
Method:	Least Squares	**F-statistic:**	2608.
Date:	Mon, 08 Nov 2021	**Prob (F-statistic):**	0.00
Time:	22:33:03	**Log-Likelihood:**	-2.3713e+05
No. Observations:	17275	**AIC:**	4.743e+05
Df Residuals:	17262	**BIC:**	4.744e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.273e+05	8123.363	89.534	0.000	7.11e+05	7.43e+05
bedrooms	-5.507e+04	2355.343	-23.381	0.000	-5.97e+04	-5.05e+04
sqft_living	3.008e+05	2688.626	111.895	0.000	2.96e+05	3.06e+05
sqft_lot	4710.1073	2442.934	1.928	0.054	-78.291	9498.506
floors	1.638e+04	2247.447	7.290	0.000	1.2e+04	2.08e+04
waterfront	6.778e+04	1709.595	39.649	0.000	6.44e+04	7.11e+04
condition	2.254e+04	1861.698	12.105	0.000	1.89e+04	2.62e+04
sqft_basement	-1.772e+04	2233.688	-7.935	0.000	-2.21e+04	-1.33e+04
yr_built	-4.09e+04	2336.677	-17.503	0.000	-4.55e+04	-3.63e+04
yr_renovated	1.495e+04	1757.013	8.509	0.000	1.15e+04	1.84e+04
lat	8.91e+04	1756.213	50.733	0.000	8.57e+04	9.25e+04
long	-2.594e+04	1997.640	-12.984	0.000	-2.99e+04	-2.2e+04
sqft_lot15	-1.002e+04	2473.966	-4.050	0.000	-1.49e+04	-5171.427

Omnibus:	12202.438	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	610660.952
Skew:	2.840	**Prob(JB):**	0.00
Kurtosis:	31.568	**Cond. No.**	18.4

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

3.3.3 Model validation by test set

In [62]:
```
# fit a model
lm = linear_model.LinearRegression()
model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```

In [63]:
```
predictions[0:5]
```

Out[63]:
```
array([[261577.38294138],
       [312053.72309381],
       [612762.5994134 ],
       [385518.01249028],
       [342843.70332692]])
```

In [64]:
```
## The line / model
plt.scatter(y_test, predictions)
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[64]:
```
Text(0, 0.5, 'Predictions')
```

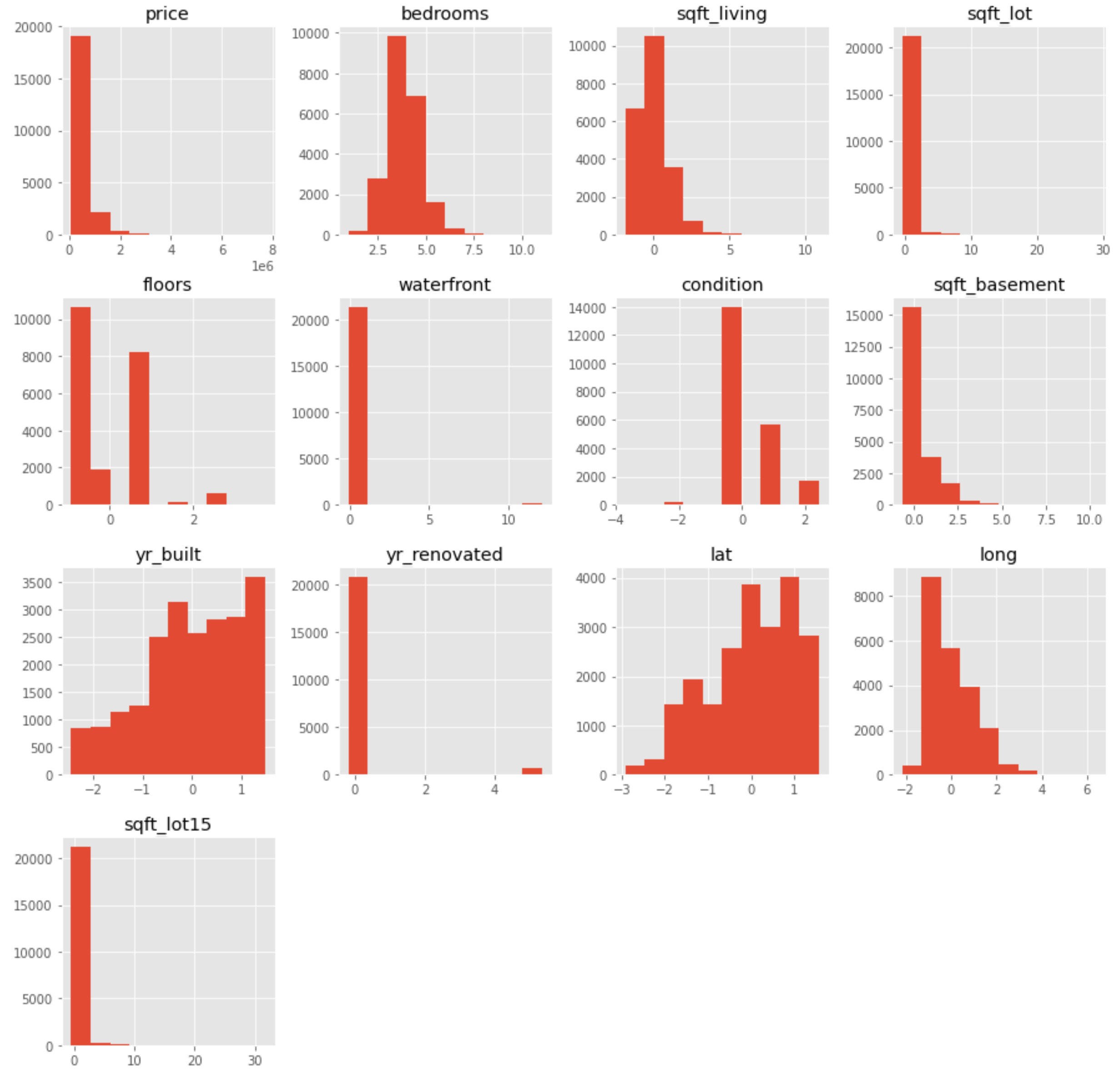
In [65]:
```
print ("Score:", model.score(X_test, y_test))
```

```
Score: 0.6338072955864396
```

In [66]:
```
# pd.plotting.scatter_matrix(df[x_cols], figsize=(20,20));
```

Transforming Non-Normal Features

In [67]:
```
fig = plt.figure(figsize = (15,15))
ax = fig.gca()
df.hist(ax = ax);
```



In [68]:
```
df.head()
```

Out[68]:

	price	bedrooms	sqft_living	sqft_lot	floors	waterfront	condition	sqft_basement	y
0	221900.0	3	-0.983623	-0.234949	-0.915624	-0.082504	-0.629907	-0.650336	-0.5
1	538000.0	3	0.535783	-0.194990	0.937594	-0.082504	-0.629907	0.260696	-0.6
2	180000.0	2	-1.431793	-0.125764	-0.915624	-0.082504	-0.629907	-0.650336	-1.2
3	604000.0	4	-0.131006	-0.251264	-0.915624	-0.082504	2.444734	1.422261	-0.1
4	510000.0	3	-0.437074	-0.173956	-0.915624	-0.082504	-0.629907	-0.650336	0.5

4 Additional Assessments and Refinement

4.1 Checking for variance factor

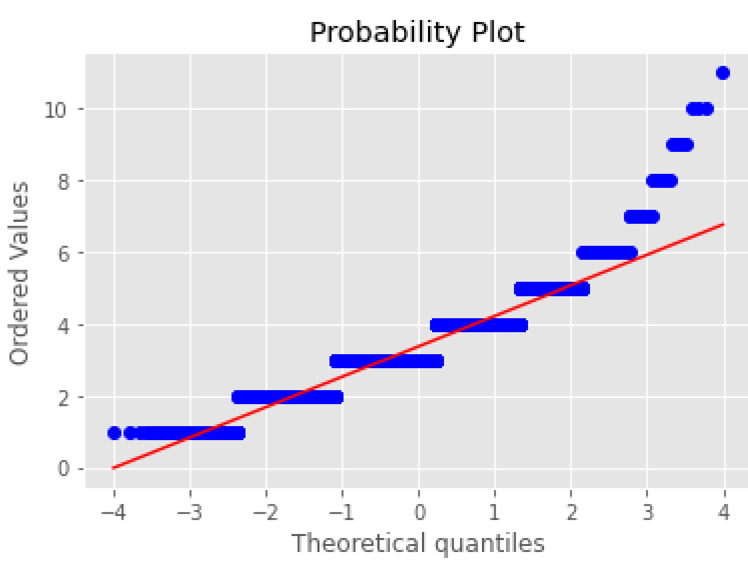
In [69]:
```
X = df[x_cols]
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
list(zip(x_cols, vif))
```

Out[69]:
```
[('bedrooms', 1.0255910825323424),
 ('sqft_living', 2.0240984880288875),
 ('sqft_lot', 2.1033827045220654),
 ('floors', 1.7867226945161656),
 ('waterfront', 1.0229815808627332),
 ('condition', 1.2142406253496107),
 ('sqft_basement', 1.7435395347965563),
 ('yr_built', 1.9068557798548587),
 ('yr_renovated', 1.0965108337555798),
 ('lat', 1.0778669646430472),
 ('long', 1.4048627017712443),
 ('sqft_lot15', 2.126586722281541)]
```

4.2 Checking for single feature Normality

In [70]:
```
import pylab
import scipy.stats as stats

stats.probplot(df['bedrooms'], dist = "norm", plot = pylab)
pylab.show()
```



4.3 Investigating Linearity

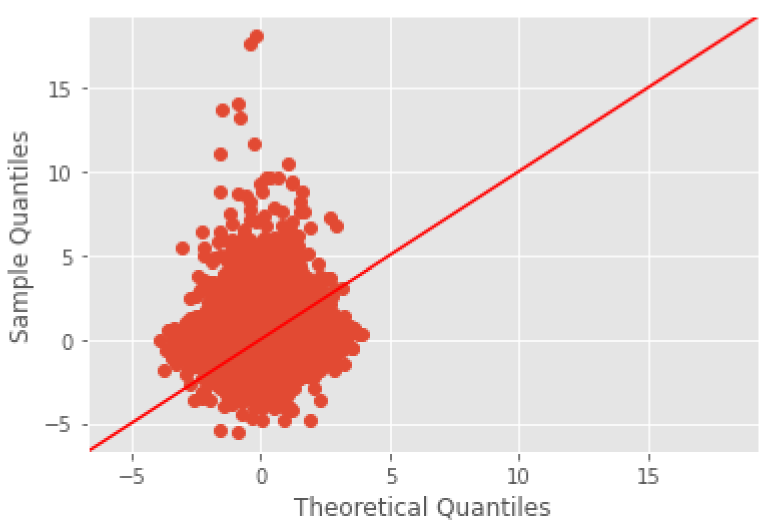
In [71]:
```
preds = model.predict(X)
fig, ax = plt.subplots()

perfect_line = np.arange(y.min().min(), y.max().max())
ax.plot(perfect_line, linestyle="--", color="orange", label="Perfect Fit")
ax.scatter(y, preds, alpha=0.5)
ax.set_xlabel("Actual Price")
ax.set_ylabel("Predicted Price")
ax.legend();
```

4.4 Investigating Normality

In [72]:
```
import scipy.stats as stats

residuals = (y - preds)
sm.graphics.qqplot(residuals, dist=stats.norm, line='45', fit=True);
```



4.5 Investigating Multicollinearity (Independence Assumption)

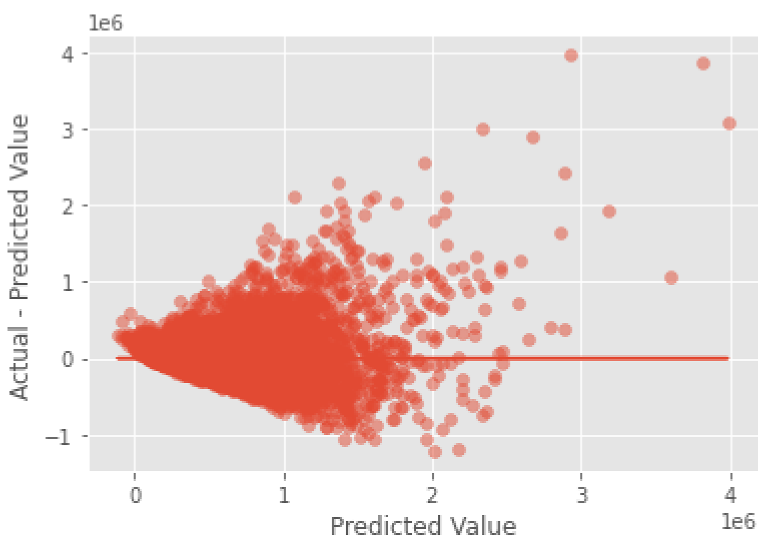
In [73]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
pd.Series(vif, index=X.columns, name="Variance Inflation Factor")
```

Out[73]:
```
bedrooms         1.025591
sqft_living      2.024098
sqft_lot         2.103383
floors           1.786723
waterfront       1.022982
condition        1.214241
sqft_basement    1.743540
yr_built         1.906856
yr_renovated     1.096511
lat              1.077867
long             1.404863
sqft_lot15       2.126587
Name: Variance Inflation Factor, dtype: float64
```

4.6 Investigating Homoscedasticity

In [74]:
```
# Run this cell without changes
fig, ax = plt.subplots()

ax.scatter(preds, residuals, alpha=0.5)
ax.plot(preds, [0 for i in range(len(X))])
ax.set_xlabel("Predicted Value")
ax.set_ylabel("Actual - Predicted Value");
```



5 Questions

5.1 Q1 what features should future owner consider most?

In [75]:
```
outcome = 'price'
x_cols = ['bedrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront', 'condi
predictors = '+'.join(x_cols)
formula = outcome + '~' + predictors
model = ols(formula=formula, data=train).fit()
model.summary()
```

Out[75]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.645
Model:	OLS	**Adj. R-squared:**	0.644
Method:	Least Squares	**F-statistic:**	2608.
Date:	Mon, 08 Nov 2021	**Prob (F-statistic):**	0.00
Time:	22:33:09	**Log-Likelihood:**	-2.3713e+05
No. Observations:	17275	**AIC:**	4.743e+05
Df Residuals:	17262	**BIC:**	4.744e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.273e+05	8123.363	89.534	0.000	7.11e+05	7.43e+05
bedrooms	-5.507e+04	2355.343	-23.381	0.000	-5.97e+04	-5.05e+04
sqft_living	3.008e+05	2688.626	111.895	0.000	2.96e+05	3.06e+05
sqft_lot	4710.1073	2442.934	1.928	0.054	-78.291	9498.506
floors	1.638e+04	2247.447	7.290	0.000	1.2e+04	2.08e+04
waterfront	6.778e+04	1709.595	39.649	0.000	6.44e+04	7.11e+04
condition	2.254e+04	1861.698	12.105	0.000	1.89e+04	2.62e+04
sqft_basement	-1.772e+04	2233.688	-7.935	0.000	-2.21e+04	-1.33e+04
yr_built	-4.09e+04	2336.677	-17.503	0.000	-4.55e+04	-3.63e+04
yr_renovated	1.495e+04	1757.013	8.509	0.000	1.15e+04	1.84e+04
lat	8.91e+04	1756.213	50.733	0.000	8.57e+04	9.25e+04
long	-2.594e+04	1997.640	-12.984	0.000	-2.99e+04	-2.2e+04
sqft_lot15	-1.002e+04	2473.966	-4.050	0.000	-1.49e+04	-5171.427

Omnibus:	12202.438	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	610660.952
Skew:	2.840	**Prob(JB):**	0.00
Kurtosis:	31.568	**Cond. No.**	18.4

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

According to the summary, if owner want to invest a new house. Bigger living room, waterfront, and south area of King County should be considered. Those three are the most important features when market valuate the property.

5.2 Q2 what should owner don't care too much?

In [78]:
```
df_year_long_bedrooms = df[['price', 'bedrooms', 'yr_built', 'long']]
```

In [81]:
```
model.summary()
```

Out[81]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.645
Model:	OLS	**Adj. R-squared:**	0.644
Method:	Least Squares	**F-statistic:**	2608.
Date:	Mon, 08 Nov 2021	**Prob (F-statistic):**	0.00
Time:	22:34:54	**Log-Likelihood:**	-2.3713e+05
No. Observations:	17275	**AIC:**	4.743e+05
Df Residuals:	17262	**BIC:**	4.744e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	7.273e+05	8123.363	89.534	0.000	7.11e+05	7.43e+05
bedrooms	-5.507e+04	2355.343	-23.381	0.000	-5.97e+04	-5.05e+04
sqft_living	3.008e+05	2688.626	111.895	0.000	2.96e+05	3.06e+05
sqft_lot	4710.1073	2442.934	1.928	0.054	-78.291	9498.506
floors	1.638e+04	2247.447	7.290	0.000	1.2e+04	2.08e+04
waterfront	6.778e+04	1709.595	39.649	0.000	6.44e+04	7.11e+04
condition	2.254e+04	1861.698	12.105	0.000	1.89e+04	2.62e+04
sqft_basement	-1.772e+04	2233.688	-7.935	0.000	-2.21e+04	-1.33e+04
yr_built	-4.09e+04	2336.677	-17.503	0.000	-4.55e+04	-3.63e+04
yr_renovated	1.495e+04	1757.013	8.509	0.000	1.15e+04	1.84e+04
lat	8.91e+04	1756.213	50.733	0.000	8.57e+04	9.25e+04
long	-2.594e+04	1997.640	-12.984	0.000	-2.99e+04	-2.2e+04
sqft_lot15	-1.002e+04	2473.966	-4.050	0.000	-1.49e+04	-5171.427

Omnibus:	12202.438	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	610660.952
Skew:	2.840	**Prob(JB):**	0.00
Kurtosis:	31.568	**Cond. No.**	18.4

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

According to the summary and seatle map, King county downtown is at west side, which means more close to downtown and US west coast is more experience. Meanwhile, older houses are usually more experience.

According to the model.summary(), bedrooms, square feet of basement and year of build are the top 3 negative features.

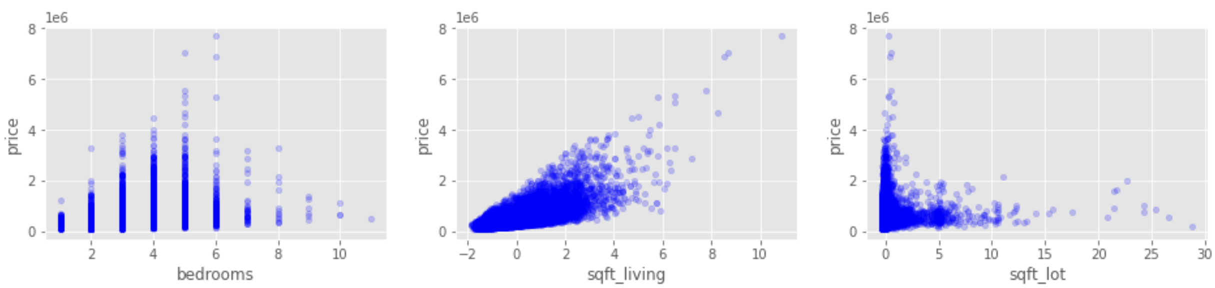
Reason:

1. House with too many bedrooms are not as hot as house with bedrooms less than 5.
2. Seattle is a place less likely to snow but rich of rain. Basement is damp and dark. Besides, basement like such contains more Radon gas, and Radon is a naturally-occurring radioactive gas that can cause lung cancer.
3. New build house are usually far from waterfront, downtown, coastline and school, those area usually been occupied with older houses.

5.3 Q3 Should owner buy a house with mores room than he/she actually needs?

In [82]:
```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(16,3))

for xcol, ax in zip(['bedrooms', 'sqft_living', 'sqft_lot'], axes):
    df.plot(kind='scatter', x=xcol, y='price', ax=ax, alpha=0.2, color='b')
```




```
From the histogram and model summary, we conclude that less bedrooms, more
living room sqft and more lot sqft would increase the price. If actual
bedrooms need is more than 5, more bedrooms usually get a less price house. We
speculate this may because those big house are less popular and hot than
bedrooms number less than 5.
```

6 Conclusion

After analysis, my team believe this is the best time getting into valuable house.

We would suggest:

1. Bigger living room, waterfront, and south area of King County should be considered.
2. For King county, downtown and US west coast is more experience.
3. Bedrooms no more than 5 are more popular and hot to trade.

7 Future work

1. With more time, I would like to look into 'zip' column see if I can find more valuable information.
2. For house sale in different months, I want to check if sale price and deal made amount change accordingly. I may need to search for more sales record in different year to get a convincing trend for that.
3. There are more features might influence price like world economic environment, Nasdaq Index, pandemic, etc. I want to merge more features to get a better model for the price prediction.