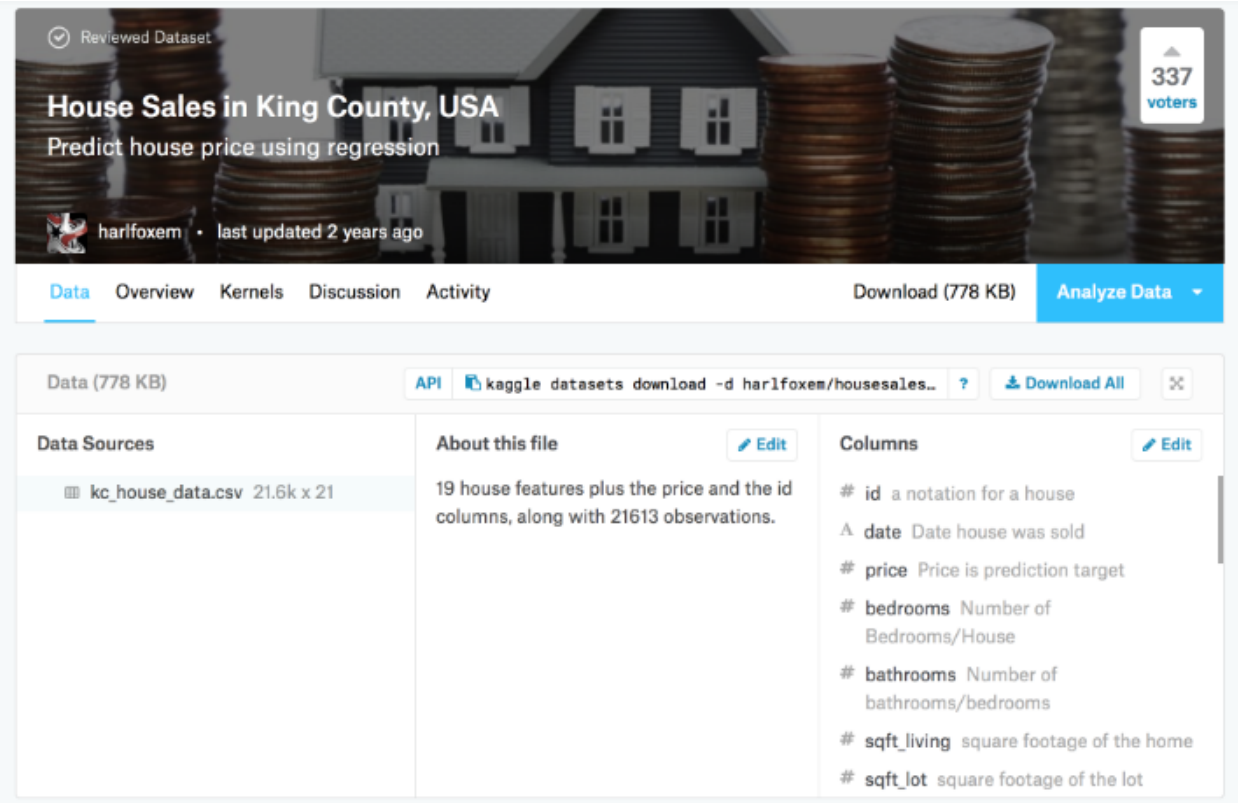
DC竞赛

(https://www.dcxueyuan.com/classDetail/classIntroduce/25/page.html) DC学院 (https://class.pkbigdata.com)

数据介绍

数据主要包括2014年5月至2015年5月美国King County的房屋销售价格以及房屋的基本信息。
数据分为训练数据和测试数据，分别保存在kc_train.csv和kc_test.csv两个文件中。

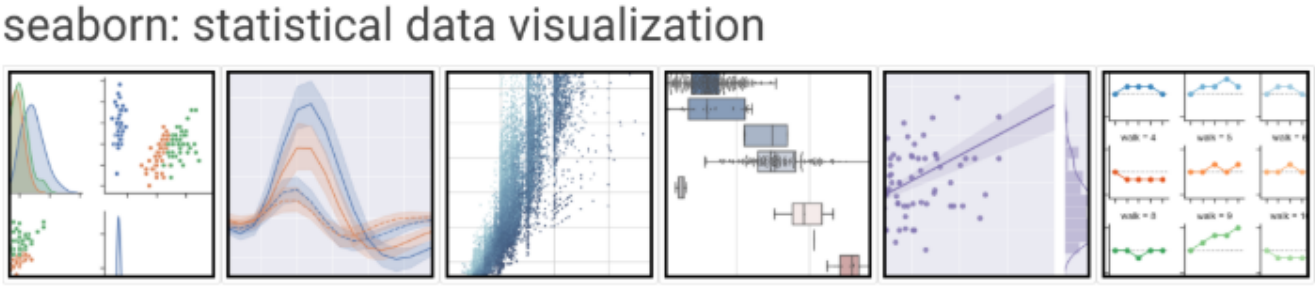
- 其中训练数据主要包括10000条记录，14个字段，主要字段说明如下：
- 第一列“销售日期”（date）：2014年5月到2015年5月房屋出售时的日期；
- **第二列“销售价格”（price）：房屋交易价格，单位为美元，是目标预测值；**
- 第三列“卧室数”（bedrooms）：房屋中的卧室数目；
- 第四列“浴室数”（bathrooms）：房屋中的浴室数目；
- 第五列“房屋面积”（sqft_living）：房屋里的生活面积；
- 第六列“停车面积”（sqft_lot）：停车坪的面积；
- 第七列“楼层数”（floors）：房屋的楼层数；
- 第八列“房屋评分”（grade）：King County房屋评分系统对房屋的总体评分；
- 第九列“建筑面积”（sqft_above）：除了地下室之外的房屋建筑面积；
- 第十列“地下室面积”（sqft_basement）：地下室的面积；
- 第十一列“建筑年份”（yr_built）：房屋建成的年份；
- 第十二列“修复年份”（yr_renovated）：房屋上次修复的年份；
- 第十三列"纬度"（lat）：房屋所在纬度；
- 第十四列“经度”（long）：房屋所在经度。
  （注：比赛所用到的数据取自于kaggle datasets，由@harlfoxem提供并分享。我们只选取了其中的子集，并对数据做了一些预处理使数据更加符合回归分析比赛的要求。）



网址：kaggle原始数据 (https://www.kaggle.com/harlfoxem/housesalesprediction)

数据准备

Seaborn在matplotlib的基础上进行了更高级的API封装，从而使得作图更加容易，在大多数情况下使用seaborn就能做出很具有吸引力的图，而使用matplotlib能制作具有更多特色的图。我们可以将seaborn视为matplotlib的补充来帮助我们更好的作图。



网址：http://seaborn.pydata.org/ (http://seaborn.pydata.org/)

In [1]:

```
# 导入相关 Python 库
import warnings
warnings.filterwarnings('ignore')#忽略一些警告

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
# 读取数据
columns = ['date', 'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
           'floors', 'grade', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_ren
ovated',
           'lat', 'long']

kc_train = pd.read_csv('./kc_train.csv', names=columns)#路径根据具体情况而定
```

In [3]:

```
# 删除日期（date）特征
kc_train.drop('date', axis=1, inplace=True)
```

In [4]:

```
# 查看数据集中是否有NaN
print(kc_train.isnull().sum())
```

```
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
lat              0
long             0
dtype: int64
```

In [5]:

```
print(kc_train.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 13 columns):
price            10000 non-null int64
bedrooms         10000 non-null int64
bathrooms        10000 non-null float64
sqft_living      10000 non-null int64
sqft_lot         10000 non-null int64
floors           10000 non-null float64
grade            10000 non-null int64
sqft_above       10000 non-null int64
sqft_basement    10000 non-null int64
yr_built         10000 non-null int64
yr_renovated     10000 non-null int64
lat              10000 non-null float64
long             10000 non-null float64
dtypes: float64(4), int64(9)
memory usage: 1015.7 KB
None
```

In [6]:

```
# 查看数据集前五行数据
kc_train.head(5)
```

Out[6]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	grade	sqft_above	sqft_
0	545000	3	2.25	1670	6240	1.0	8	1240	430
1	785000	4	2.50	3300	10514	2.0	10	3300	0
2	765000	3	3.25	3190	5283	2.0	9	3190	0
3	720000	5	2.50	2900	9525	2.0	9	2900	0
4	449500	5	2.75	2040	7488	1.0	7	1200	840

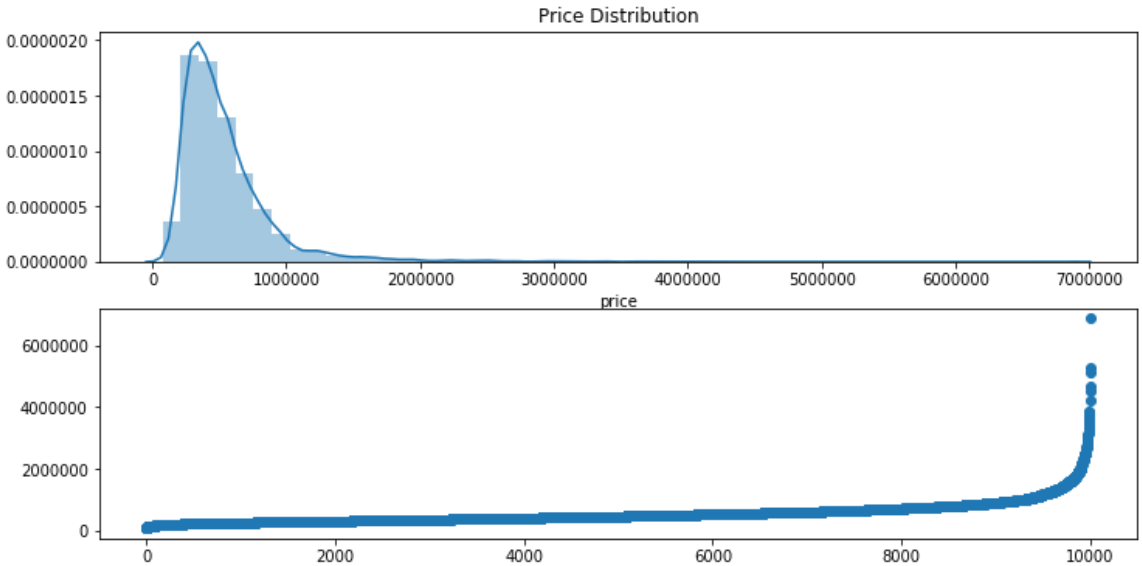
数据挖掘和特征工程

先查看我们需要预测的房屋价格的相关信息。

matplotlib.pyplot各参数API文档：https://matplotlib.org/api/pyplot_summary.html (https://matplotlib.org/api/pyplot_summary.html)

In [7]:

```
plt.figure(figsize=(12, 6))
# 绘制价格的数据分布曲线
plt.subplot(211)
plt.title('Price Distribution')
sns.distplot(kc_train['price'])
# 绘制价格曲线
plt.subplot(212)
plt.scatter(range(kc_train.shape[0]), np.sort(kc_train['price'].values))
plt.show()
```



In [8]:

```
# 查看特征的统计信息
kc_train.describe()
```

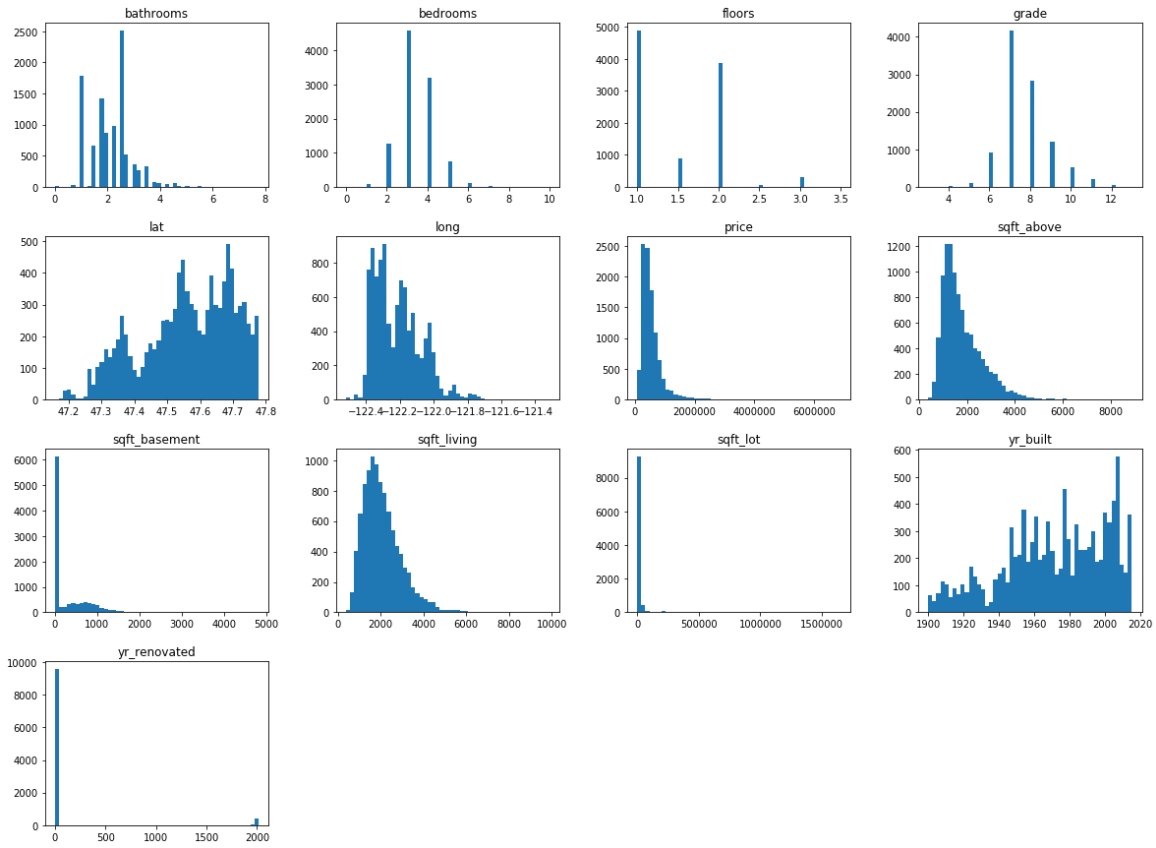
Out[8]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	f
count	1.000000e+04	10000.000000	10000.0000	10000.000000	1.000000e+04	10000.0(
mean	5.428749e+05	3.367600	2.1168	2082.488400	1.535273e+04	1.50285(
std	3.729258e+05	0.893169	0.7741	922.878916	4.577623e+04	0.54364:
min	7.500000e+04	0.000000	0.0000	390.000000	5.720000e+02	1.00000(
25%	3.225000e+05	3.000000	1.7500	1430.000000	5.023250e+03	1.00000(
50%	4.507000e+05	3.000000	2.2500	1910.000000	7.590000e+03	1.50000(
75%	6.450000e+05	4.000000	2.5000	2550.000000	1.071700e+04	2.00000(
max	6.885000e+06	10.000000	7.7500	9890.000000	1.651359e+06	3.50000(

绘制出各个特征的分布曲线来了解下它们的数据类型。

In [9]:

```
# 绘制各个特征的分布柱状图
kc_train.hist(figsize=(20, 15), bins=50, grid=False)
plt.show()
```



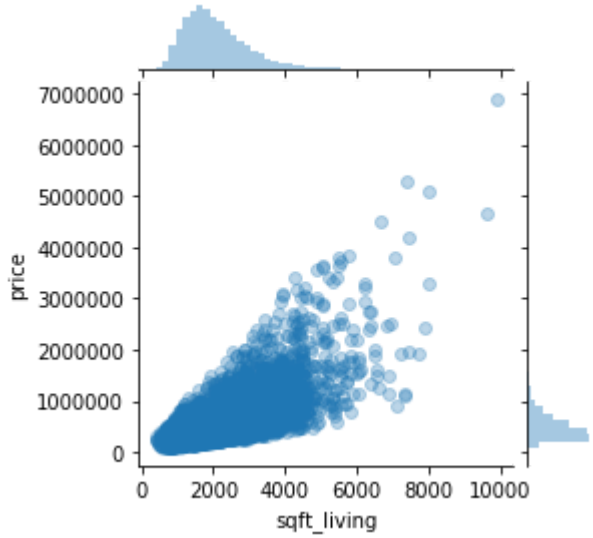
我们可以注意到以下几种特征都是连续变量：

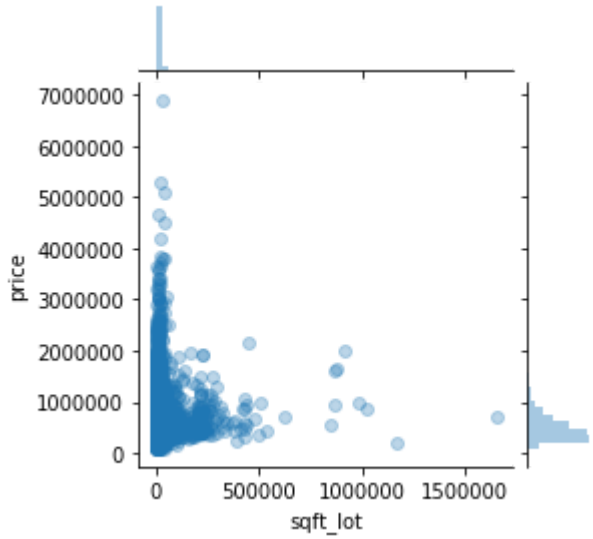
- lat
- long
- sqft_above
- sqft_basement
- sqft_living
- sqft_lot
- yr_built
- yr_renovated

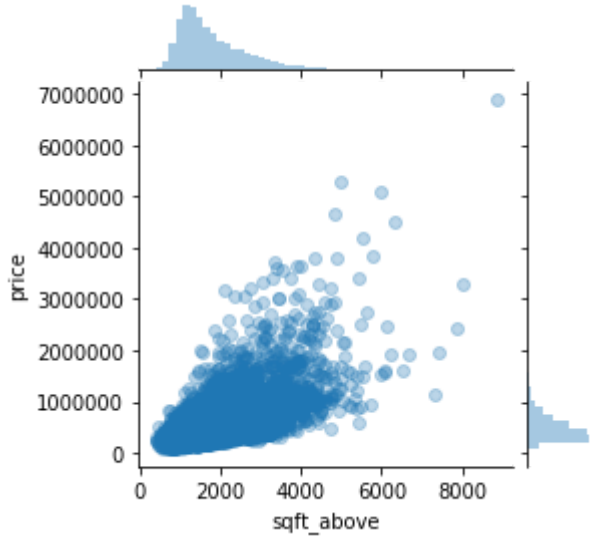
我们再来分析下几种连续变量与房价之间的相关关系，由于它们都是连续的，所以我们采用Pearson相关系数（相关系数的绝对值越大，相关性越强：相关系数越接近于1或-1，相关度越强，相关系数越接近于0，相关度越弱）来衡量它们的相关程度。

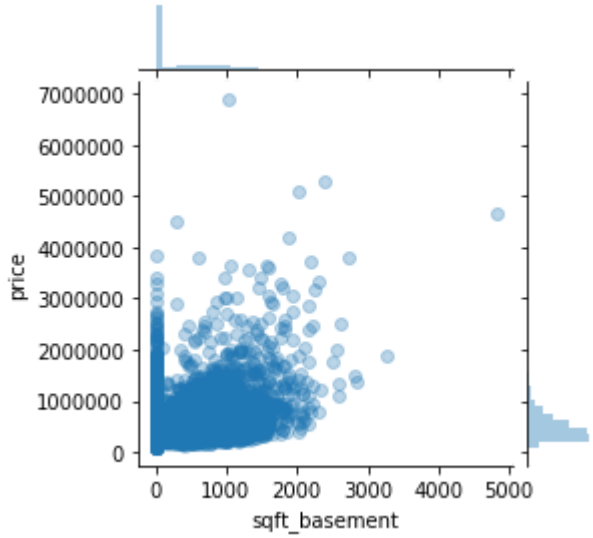
In [10]:

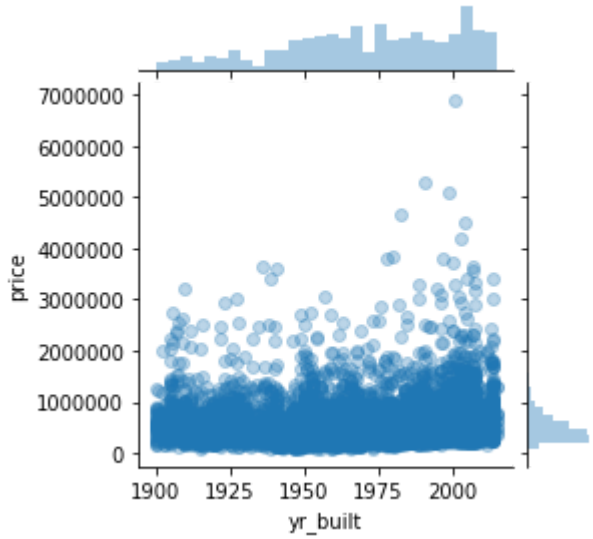
```
continuous_cols = ['sqft_living', 'sqft_lot', 'sqft_above', 'sqft_basement',
       'yr_built', 'yr_renovated', 'lat', 'long']

for col in continuous_cols:
    sns.jointplot(x=col, y="price", data=kc_train, alpha=0.3, size=4)
```

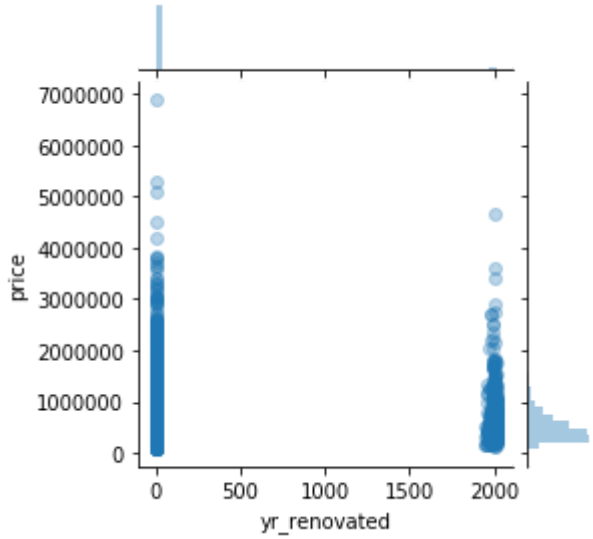

price
sqft_living
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
0
2000
4000
6000
8000
10000

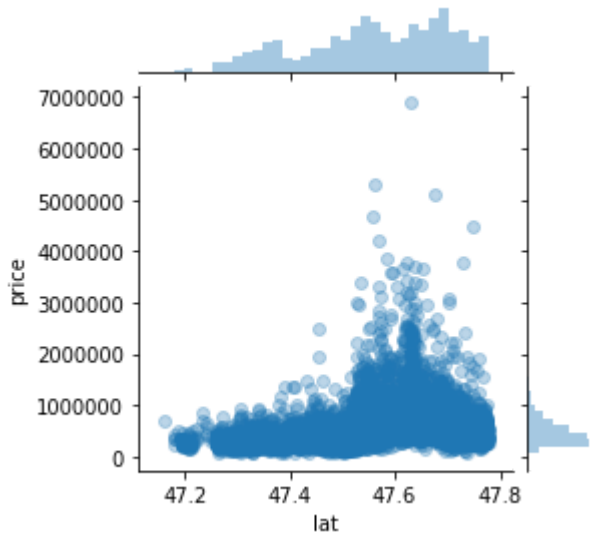

price
sqft_lot
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
0
500000
1000000
1500000

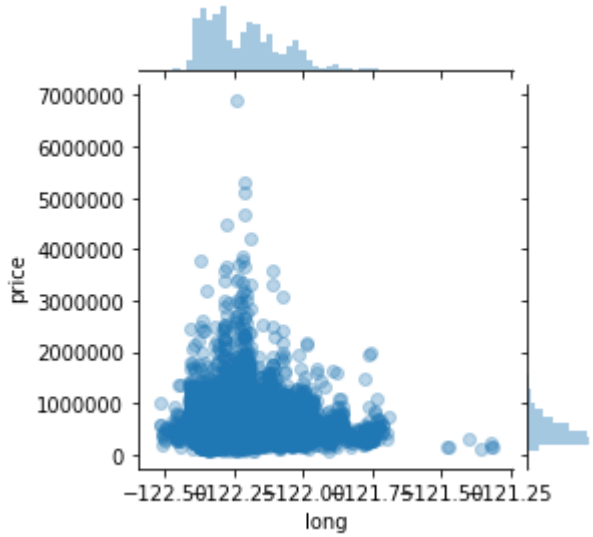

price
sqft_above
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
0
2000
4000
6000
8000


price
sqft_basement
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
0
1000
2000
3000
4000
5000


price
yr_built
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
1900
1925
1950
1975
2000


price
yr_renovated
0
1000000
2000000
3000000
4000000
5000000
6000000
7000000
0
500
1000
1500
2000





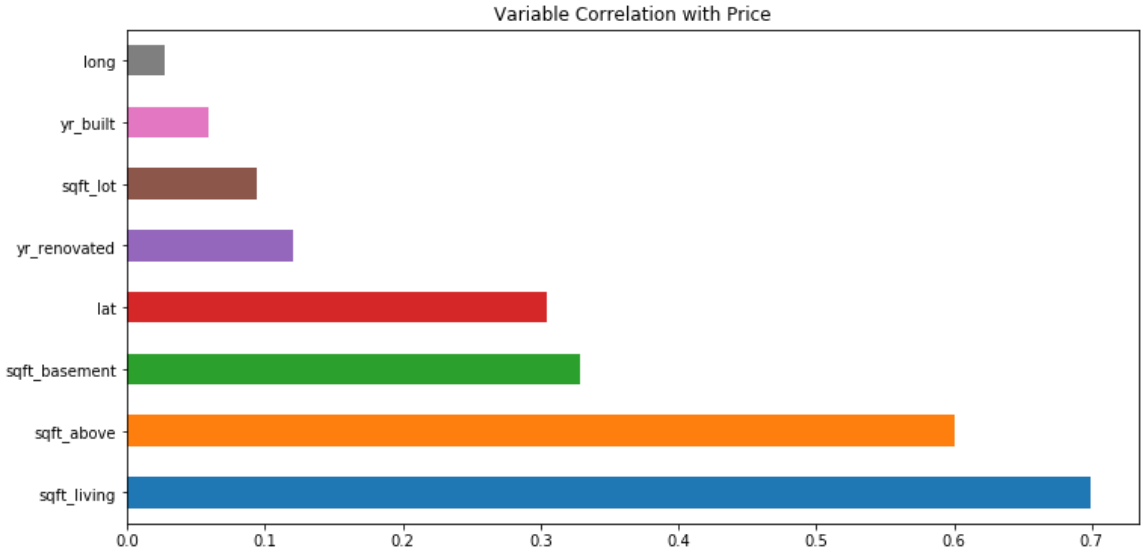
我们可以将这些连续变量与房屋售价之间的相关关系按照Pearson系数进行排序。

In [11]:

```
plt.figure(figsize=(12, 6))
kc_train.corr()['price'][continuous_cols].sort_values(ascending=False).plot(
    'barh', figsize=(12, 6), title='Variable Correlation with Price'
)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21755946b70>
```



可以注意到sqft_basement、yr_renovated都有存在很多为0的数据，所以我们可以考虑构建新的二值特征分别用来表示是否有地下室、是否曾翻新过。

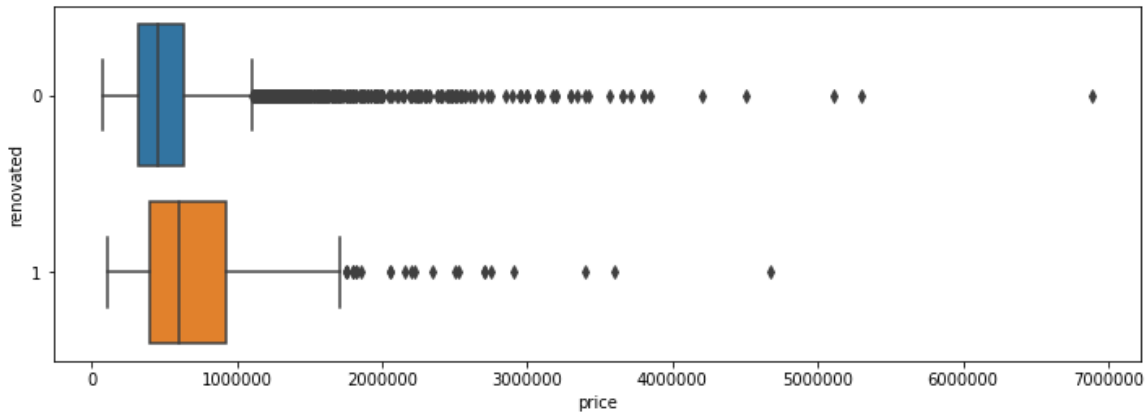
In [12]:

```
kc_train['basement_present'] = kc_train['sqft_basement'].apply(lambda x: 1 if x
> 0 else 0)
kc_train['renovated'] = kc_train['yr_renovated'].apply(lambda x: 1 if x > 0 else
 0)
```

在上面我们构建的basement_present、renovated都是属于分类变量（categorical variable），我们可以使用点二列相关系数来计算两个变量之间的关系。

In [13]:

```
from scipy.stats import pointbiserialr

# 绘制箱形图
plt.figure(figsize=(12, 4))
sns.boxplot(y='renovated', x='price', data=kc_train, orient='h')
plt.show()

# 计算点二列相关系数
r, p = pointbiserialr(kc_train['renovated'], kc_train['price'])
print ('renovated 与 price 的点二列相关系数中 r = %s, p = %s' %(r, p))
r, p = pointbiserialr(kc_train['basement_present'], kc_train['price'])
print ('basement_present 与 price 的点二列相关系数中 r = %s, p = %s' %(r, p))
```




```
renovated 与 price 的点二列相关系数中 r = 0.12001849968412436, p = 2.088
308248694044e-33
basement_present 与 price 的点二列相关系数中 r = 0.18330716277977743, p
= 2.7139191120161128e-76
```

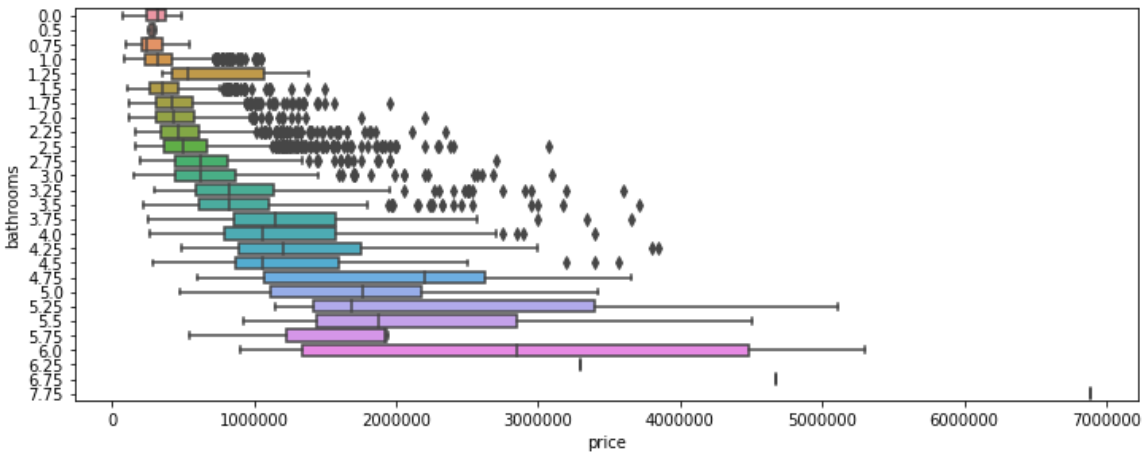
可以看出来：

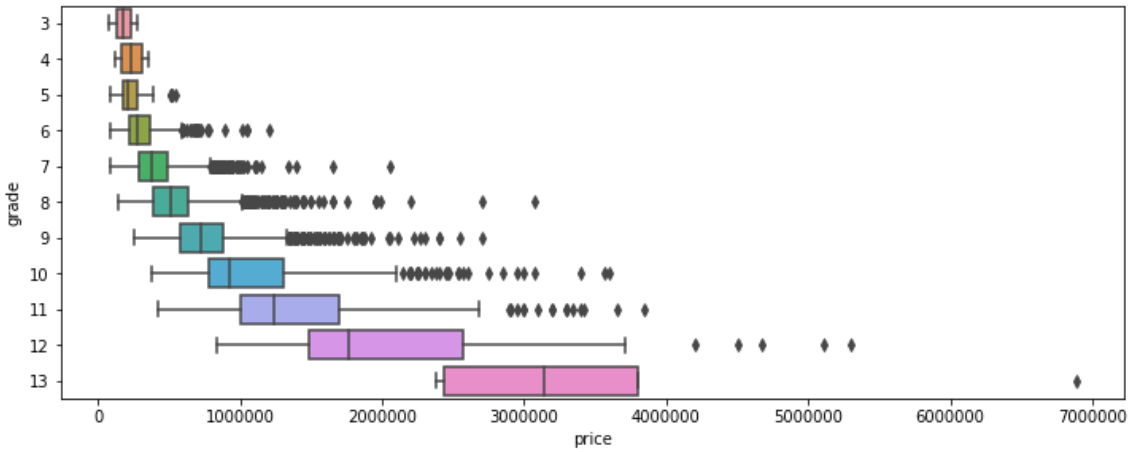
- 没有翻新过的箱形图比较窄，这表明整体而言这组房价非常接近
- 有无翻新对于房价高低没有太大的影响，一般而言翻新后房价可能会高一点
- renovated、basement_present和price变量之间的相关性都较小

接下来我们再探究各个顺序变量（ordinal variable）和price之间的关系，用斯皮尔曼等级相关系数（Spearman's rank-order correlation）来计算相关性

In [14]:

```
plt.figure(figsize=(12, 10))
plt.subplot(211)
sns.boxplot(y='bathrooms', x='price', data=kc_train, orient='h')
plt.subplot(212)
sns.boxplot(y='grade', x='price', data=kc_train, orient='h')
plt.show()
```





In [15]:

```
from scipy.stats import spearmanr

r, p = spearmanr(kc_train['floors'], kc_train['price'])
print ('floors 和 price 斯皮尔曼相关系数为 %s，其中 p = %s' %(r,p))
r, p = spearmanr(kc_train['bedrooms'], kc_train['price'])
print ('bedrooms 和 price 斯皮尔曼相关系数为 %s，其中 p = %s' %(r,p))
r, p = spearmanr(kc_train['bathrooms'], kc_train['price'])
print ('bathrooms 和 price 斯皮尔曼相关系数为 %s，其中 p = %s' %(r,p))
r, p = spearmanr(kc_train['grade'], kc_train['price'])
print ('grade 和 price 斯皮尔曼相关系数为 %s，其中 p = %s' %(r,p))
```

```
floors 和 price 斯皮尔曼相关系数为 0.3127190036471108，其中 p = 9.5863331
25228778e-226
bedrooms 和 price 斯皮尔曼相关系数为 0.3458697748314727，其中 p = 5.68012
69872902106e-279
bathrooms 和 price 斯皮尔曼相关系数为 0.5012034892956143，其中 p = 0.0
grade 和 price 斯皮尔曼相关系数为 0.6603554146361819，其中 p = 0.0
```

scipy.stats.spearmanr(a, b=None, axis=0, nan_policy='propagate')

参数解释：

- a,b：是一维或二维数组，b是可选的。
- axis：整型或者空值，如果axis = 0（默认值），则每列代表一个变量，并在行中进行观察。如果axis = 1，则转换关系：每行代表一个变量，而列包含观察值。
- nan_policy : {‘propagate’, ‘raise’, ‘omit’}, 都是可选的，定义输入包含nan时的处理方式。'propagate'返回nan，'raise'抛出错误，'omit'执行忽略nan值的计算。默认为'propagate'。

返回值：

- correlation：Spearman相关矩阵或相关系数
- pvalue：浮点数，假设检验的双侧p值，其零假设是两组数据不相关，具有与rho相同的维度。

我们可以得出以下结论：

- 连续变量中sqft_living、sqft_above、sqft_basement与price之间存在极强的相关关系
- 二元变量basement_present、renovated与price之间存在一定的相关关系，但是关联度较小
- 几个顺序变量（floors、bedrooms、bathrooms、grade）都与price之间存在相关关系

数据预处理

虚拟变量（dummy variable）是指用0或1来表示某个特定的分类是否存在的人工变量，这里我们对floors、grade做编码处理，将bedrooms和bathrooms看作是连续变量。

In [16]:

```
ordinal_cols = ['floors', 'grade']

for col in ordinal_cols:
    dummies = pd.get_dummies(kc_train[col], drop_first=False)
    dummies = dummies.add_prefix("{}#".format(col))
    kc_train.drop(col, axis=1, inplace=True)
    kc_train = kc_train.join(dummies)
```

In [17]:

```
kc_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 30 columns):
price                10000 non-null int64
bedrooms             10000 non-null int64
bathrooms            10000 non-null float64
sqft_living          10000 non-null int64
sqft_lot             10000 non-null int64
sqft_above           10000 non-null int64
sqft_basement        10000 non-null int64
yr_built             10000 non-null int64
yr_renovated         10000 non-null int64
lat                  10000 non-null float64
long                 10000 non-null float64
basement_present     10000 non-null int64
renovated            10000 non-null int64
floors#1.0           10000 non-null uint8
floors#1.5           10000 non-null uint8
floors#2.0           10000 non-null uint8
floors#2.5           10000 non-null uint8
floors#3.0           10000 non-null uint8
floors#3.5           10000 non-null uint8
grade#3              10000 non-null uint8
grade#4              10000 non-null uint8
grade#5              10000 non-null uint8
grade#6              10000 non-null uint8
grade#7              10000 non-null uint8
grade#8              10000 non-null uint8
grade#9              10000 non-null uint8
grade#10             10000 non-null uint8
grade#11             10000 non-null uint8
grade#12             10000 non-null uint8
grade#13             10000 non-null uint8
dtypes: float64(3), int64(10), uint8(17)
memory usage: 1.2 MB
```

由于连续变量太多还有包含yr_renovated这类比较难以处理的变量，所以这里我们暂不做数据归一化处理。接下来我们再分割训练集。

In [18]:

```
from sklearn.cross_validation import train_test_split
np.random.seed(21)
target = kc_train['price']
kc_train.drop('price', axis=1, inplace=True)
train_data, val_data, train_y, val_y = train_test_split(
    kc_train, target, train_size=0.8, random_state=21
)
```

```
e:\anaconda3\envs\fake_1\lib\site-packages\sklearn\cross_validation.
py:41: DeprecationWarning: This module was deprecated in version 0.1
8 in favor of the model_selection module into which all the refactor
ed classes and functions are moved. Also note that the interface of
the new CV iterators are different from that of this module. This mo
dule will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

sklearn.cross_validation.train_test_split(*arrays, **options)

参数：

- *arrays:具有相同长度/形状的可索引序列。
- train_size:训练集所占的比例，在(0，1)之间
- random_state :随机种子数，可选int，RandomState instance或者none。默认值是none，如果是int，则random_state是随机数生成器使用的种子; 如果是RandomState实例，则random_state是随机数生成器; 如果没有，随机数生成器所使用的RandomState实例np.random。

模型训练与特征选择

简单线性回归

我们先选择使用一个简单的线性回归模型来对单一特征进行训练。

In [19]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

def simple_linear_model(train_data, train_y, val_data, val_y, input_feature):
    reg = LinearRegression()
    reg.fit(train_data.as_matrix(columns=[input_feature]), train_y)
    RMSE = mean_squared_error(
        val_y, reg.predict(val_data.as_matrix(columns=[input_feature]))
    ) ** 0.5
    return RMSE
```

In [20]:

```
np.random.seed(21)
RMSE = simple_linear_model(train_data, train_y, val_data, val_y, 'sqft_living')
print('Validation RMSE for sqft_living is: %s' % RMSE)
```

```
Validation RMSE for sqft_living is: 240277.31977171925
```

mean_squared_error：均方误差

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\mathrm{samples}}} \sum_{i=0}^{n_{\mathrm{samples}}-1} (y_i - \hat{y}_i)^2.$$

同样的，我们可以对数据中的所有特征进行相同的训练，使用上面的简单线性回归模型来评估哪个特征更能拟合模型。

In [21]:

```
estimate_result = pd.DataFrame(columns=['feature', 'Val_RMSE'])

np.random.seed(21)

for feature in train_data.columns:
    RMSE = simple_linear_model(train_data, train_y, val_data, val_y, feature)
    estimate_result = estimate_result.append({'feature': feature, 'Val_RMSE': RM
SE}, ignore_index=True)
```

In [22]:

```
estimate_result.sort_values(by='Val_RMSE').head(10)
```

Out[22]:

	feature	Val_RMSE
2	sqft_living	240277.319772
4	sqft_above	272600.850888
1	bathrooms	282566.848238
25	grade#10	300019.451698
5	sqft_basement	305349.135337
22	grade#7	305904.054259
8	lat	308012.254495
24	grade#9	310292.442529
0	bedrooms	311763.024435
26	grade#11	315187.693311

从拟合结果中我们可以看到，所有特征中sqft_living拟合效果最好，其次sqft_above、bathrooms、grade#10、sqft_basement也能较好的拟合房价曲线。

多元线性回归

接下来我们再简单的修改下线性回归函数，在上面的基础上将多个特征作为输入用来训练和拟合房价数据。

In [23]:

```
def multiple_regression(train_data, train_y, val_data, val_y, input_feature):
    reg = LinearRegression()
    reg.fit(train_data.as_matrix(columns=input_feature), train_y)
    RMSE = mean_squared_error(
        val_y, reg.predict(val_data.as_matrix(columns=input_feature))
    ) ** 0.5
    return RMSE
```

我们随机尝试几种不同的组合。

In [24]:

```
feature_combinations = [
    ['sqft_living', 'bathrooms', 'grade#10'],
    ['sqft_living', 'bathrooms', 'bedrooms'],
    ['sqft_above', 'grade#10', 'bathrooms'],
    ['sqft_basement', 'bathrooms', 'bedrooms']
]

np.random.seed(21)

for combination in feature_combinations:
    RMSE = multiple_regression(train_data, train_y, val_data, val_y, combination
)
    print('RMSE for %s is %s' % (', '.join(combination), RMSE))
```

```
RMSE for sqft_living, bathrooms, grade#10 is 235642.8838123593
RMSE for sqft_living, bathrooms, bedrooms is 236238.8651737115
RMSE for sqft_above, grade#10, bathrooms is 259075.6916844574
RMSE for sqft_basement, bathrooms, bedrooms is 273350.88502554636
```

我们还可以选择拟合效果较好的特征用于构建多项式特征，比如选择sqft_living和sqft_above构建一个二次项特征。

In [25]:

```
train_data['sqft_living_squared'] = train_data['sqft_living'].apply(lambda x: x*
*2)
val_data['sqft_living_squared'] = val_data['sqft_living'].apply(lambda x: x**2)
train_data['sqft_above_squared'] = train_data['sqft_above'].apply(lambda x: x**2
)
val_data['sqft_above_squared'] = val_data['sqft_above'].apply(lambda x: x**2)

squared_combinations = [
    ['sqft_living', 'sqft_living_squared'],
    ['sqft_above', 'sqft_above_squared']
]

np.random.seed(21)

for squared_combination in squared_combinations:
    RMSE = multiple_regression(train_data, train_y, val_data, val_y, squared_com
bination)
    print('RMSE for %s is %s' % (', '.join(squared_combination), RMSE))
```

```
RMSE for sqft_living, sqft_living_squared is 231045.66700240257
RMSE for sqft_above, sqft_above_squared is 267872.41923152807
```

特征选择

通过选择不同的组合和构造多项式特征我们可以获得性能更好的模型，但是仍然存在许多问题，我们并不知道该选择哪些特征来进行组合或者构建多项式特征能够在验证集上得到更小的RMSE，通常如果将所有特征都加入到模型训练中很容易导致过拟合，所以我们需要进行特征提取。这里我们采用随机森林（random forest）来做特征选择。

我们先使用所有特征来训练模型查看效果。

In [26]:

```
np.random.seed(21)

RMSE = multiple_regression(train_data, train_y, val_data, val_y, train_data.colu
mns)
print('RMSE for all features before construction is %s' % RMSE)
```

```
RMSE for all features before construction is 185777.37051205485
```

然后我们再构造其他多项式特征使得构建的特征能够使模型达到过拟合的效果。

In [27]:

```
train_data['sqft_living_cubed'] = train_data['sqft_living'].apply(lambda x: x**3
)
val_data['sqft_living_cubed'] = val_data['sqft_living'].apply(lambda x: x**3)

train_data['bedrooms_squared'] = train_data['bedrooms'].apply(lambda x: x**2)
val_data['bedrooms_squared'] = val_data['bedrooms'].apply(lambda x: x**2)

train_data['bed_bath_rooms'] = train_data['bedrooms'] * train_data['bathrooms']
val_data['bed_bath_rooms'] = val_data['bedrooms'] * val_data['bathrooms']

train_data['log_sqft_living'] = train_data['sqft_living'].apply(lambda x: np.log
(x))
val_data['log_sqft_living'] = val_data['sqft_living'].apply(lambda x: np.log(x))
```

使用随机森林模型来做特征提取。

In [28]:

```
from sklearn.ensemble import RandomForestRegressor

np.random.seed(21)

rf_reg = RandomForestRegressor(n_estimators=50, verbose=1)
rf_reg.fit(train_data, train_y)

combine_lists = lambda item: [item[0], item[1]]
feature_importance = list(map(combine_lists, zip(train_data.columns, rf_reg.feat
ure_importances_)))
feature_importance = pd.DataFrame(
    feature_importance, columns=['feature', 'importance']
).sort_values(by='importance', ascending=False)
```

```
[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    3.9s finishe
d
```

sklearn.ensemble.RandomForestRegressor()

参数：

- n_estimators：森林里树的数量，通常数量越大，效果越好，但是计算时间也会随之增加。 此外要注意，当树的数量超过一个临界值之后，算法的效果并不会很显著地变好。
- max_features：分割节点时考虑的特征的随机子集的大小。 这个值越低，方差减小得越多，但是偏差的增大也越多。 根据经验，回归问题中使用 max_features = n_features ， 分类问题使用 max_features = sqrt（n_features （其中 n_features 是特征的个数）是比较好的默认值。
- verbose: (default=0) 是否显示任务进程。
- max_depth = None 和 min_samples_split = 2 结合通常会有不错的效果（即生成完全的树）。

In [29]:

```
feature_importance.head(10)
```

Out[29]:

	feature	importance
8	lat	0.185722
34	log_sqft_living	0.168064
31	sqft_living_cubed	0.136864
2	sqft_living	0.124488
29	sqft_living_squared	0.124146
9	long	0.090469
3	sqft_lot	0.034986
6	yr_built	0.019514
33	bed_bath_rooms	0.017337
4	sqft_above	0.013256

过滤重要性小于0.001的所有特征包含已构造的特征在内。

In [30]:

```
filter_feature = feature_importance[feature_importance['importance'] > 0.001]['f
eature'].tolist()
```

In [31]:

```
np.random.seed(21)

RMSE = multiple_regression(train_data, train_y, val_data, val_y, filter_feature)
print('RMSE for filtered features after construction is %s' % RMSE)
RMSE = multiple_regression(train_data, train_y, val_data, val_y, train_data.colu
mns)
print('RMSE for all features after construction is %s' % RMSE)
```

```
RMSE for filtered features after construction is 185292.7233928708
RMSE for all features after construction is 185862.69738691175
```

尝试其他模型

上述训练中我们都是采用的简单的线性回归模型，还有很多其他的回归模型我们可以选择，比如岭回归模型、随机森林回归模型等等可以尝试，通常可以得到不同的结果。

In [32]:

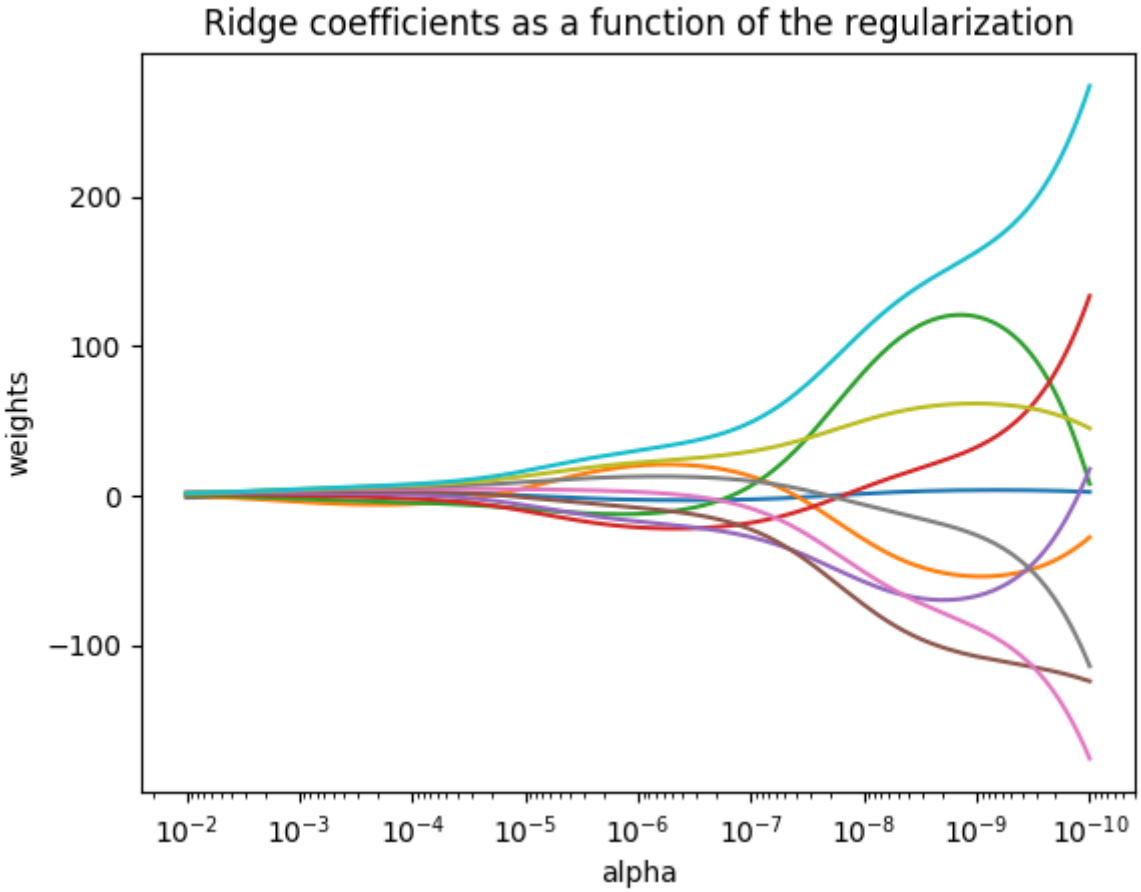
```
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

np.random.seed(21)

def other_regression_model(reg, train_data, train_y, val_data, val_y, input_feat
ure):
    reg.fit(train_data.as_matrix(columns=input_feature), train_y)
    RMSE = mean_squared_error(
        val_y, reg.predict(val_data.as_matrix(columns=input_feature))
    ) ** 0.5
    return reg, RMSE

ridge = Ridge()
dt_gre = DecisionTreeRegressor()
rf_reg = RandomForestRegressor()
ridge, RMSE = other_regression_model(ridge, train_data, train_y, val_data, val_y
, filter_feature)
print('RMSE for ridge_regression is %s' % RMSE)
dt_gre, RMSE = other_regression_model(dt_gre, train_data, train_y, val_data, val
_y, filter_feature)
print('RMSE for DecisionTreeRegressor is %s' % RMSE)
rf_reg, RMSE = other_regression_model(rf_reg, train_data, train_y, val_data, val
_y, filter_feature)
print('RMSE for RandomForestRegressor is %s' % RMSE)
```

```
RMSE for ridge_regression is 185192.9195556569
RMSE for DecisionTreeRegressor is 192512.00182354866
RMSE for RandomForestRegressor is 144223.17002847893
```

岭回归：回归通过对系数的大小施加惩罚来解决 普通最小二乘法 的一些问题。 岭系数最小化的是带罚项的残差平方和。

$$\min_{w} ||Xw - y||_2^2 + \alpha ||w||_2^2$$

其中， α>= 0 是控制系数收缩量的复杂性参数：α的值越大，收缩量越大，这样系数对共线性的鲁棒性也更强。



调节超参数

In [56]:

```
from sklearn.model_selection import GridSearchCV

np.random.seed(21)

tune_params = {'n_estimators': [10, 20 ,30],
               'max_features': ['auto', 'sqrt'],
               'max_depth': [10, 30, 50]}

rf_reg = RandomForestRegressor()
grid_search = GridSearchCV(estimator=rf_reg, param_grid=tune_params, verbose=1,
n_jobs = -1)
grid_search.fit(train_data, train_y)
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits

[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   13.2s
[Parallel(n_jobs=-1)]: Done  54 out of  54 | elapsed:   15.6s finish
ed
```

Out[56]:

```
GridSearchCV(cv=None, error_score='raise',
       estimator=RandomForestRegressor(bootstrap=True, criterion='ms
e', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start
=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid={'n_estimators': [10, 20, 30], 'max_features': ['a
uto', 'sqrt'], 'max_depth': [10, 30, 50]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='war
n',
       scoring=None, verbose=1)
```

In [57]:

```
grid_search.best_params_
```

Out[57]:

```
{'max_depth': 30, 'max_features': 'auto', 'n_estimators': 30}
```

In [58]:

```
np.random.seed(21)

rf_reg = RandomForestRegressor(n_estimators=50, max_features='auto', max_depth=3
0)

rf_reg, RMSE = other_regression_model(rf_reg, train_data, train_y, val_data, val
_y, filter_feature)
print('RMSE for RandomForestRegressor is %s' % RMSE)
```

```
RMSE for RandomForestRegressor is 134234.98988259918
```

结论

在这个案例里我们给出了数据科学竞赛的基本流程，从了解数据集、数据挖掘和可视化、数据预处理和构造数据再到模型训练这几个方面，能够帮助大家很好的入门数据科学竞赛。案例中仍有很多待完善的部分，通过解决这些不足能够帮助大家在竞赛中获得更好的成绩。

同时这里也有一些给新手的建议：

- 选择一个自己感兴趣的或者已经了解的方向来参加相关的比赛；
- 可以按照我上述提到的流程对比赛数据进行初步的处理和训练，然后有意识的针对不足的部分加以改进；
- 通过公开的代码去学习和理解别人的想法；
- 在分数已经不错的时候可以尝试去做集成学习模型；
- 尽量都去尝试一些不同的方法，不要满足于调参。