housing

June 25, 2017

1 Predict House Prices in King Country

Note: gmaps plugin neeeds to be installed:
conda install -c conda-forge gmaps

```
In [32]: %matplotlib inline
         import numpy as np # linear algebra
         import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

1.1 Loading the dataset

```
In [33]: data = "kc_house_data.csv"

         df = pd.read_csv(data, header = 0)
```

1.2 Getting first information about the data

To get a first impression about the data, will try display useful information about every attribute.

```
In [34]: df.describe()
```

Out[34]:

	id	price	bedrooms	bathrooms	sqft_livir
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.00000
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.89973
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.44089
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.00000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.00000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.00000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.00000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.00000

	sqft_lot	floors	waterfront	view	conditio
count	2.161300e+04	21613.000000	21613.000000	21613.000000	21613.00000
mean	1.510697e+04	1.494309	0.007542	0.234303	3.40943
std	4.142051e+04	0.539989	0.086517	0.766318	0.65074
min	5.200000e+02	1.000000	0.000000	0.000000	1.00000
25%	5.040000e+03	1.000000	0.000000	0.000000	3.00000

50%	7.618000e+03	1.500000	0.000000	0.000000	3.00000
75%	1.068800e+04	2.000000	0.000000	0.000000	4.00000
max	1.651359e+06	3.500000	1.000000	4.000000	5.00000

	grade	sqft_above	sqft_basement	yr_built	yr_renovat
count	21613.000000	21613.000000	21613.000000	21613.000000	21613.0000
mean	7.656873	1788.390691	291.509045	1971.005136	84.4022
std	1.175459	828.090978	442.575043	29.373411	401.6792
min	1.000000	290.000000	0.000000	1900.000000	0.0000
25%	7.000000	1190.000000	0.000000	1951.000000	0.0000
50%	7.000000	1560.000000	0.000000	1975.000000	0.0000
75%	8.000000	2210.000000	560.000000	1997.000000	0.0000
max	13.000000	9410.000000	4820.000000	2015.000000	2015.0000

	zipcode	lat	long	sqft_living15	sqft_lo
count	21613.000000	21613.000000	21613.000000	21613.000000	21613.000
mean	98077.939805	47.560053	-122.213896	1986.552492	12768.455
std	53.505026	0.138564	0.140828	685.391304	27304.179
min	98001.000000	47.155900	-122.519000	399.000000	651.000
25%	98033.000000	47.471000	-122.328000	1490.000000	5100.000
50%	98065.000000	47.571800	-122.230000	1840.000000	7620.000
75%	98118.000000	47.678000	-122.125000	2360.000000	10083.000
max	98199.000000	47.777600	-121.315000	6210.000000	871200.000

Price The first values I will have a closer look at will be the price.

We have prices ranging from roughly $75,000 to $7,700,000. 50% of the prices is around $450,000. The standard deviation is $360,000. The mean of the prices is $540,000. If we look at the quarters we have: - first 25% span over $235,000 - second 25% span over $130,000 - third 25% span over $195,000 - top 25% span $7,125,000

This clearly shows that the first 3/4 of all houses lie in a pretty similiar range and are at least somewhat evenly distributed in their quarters. The top 25% are really far out regarding their prices which might be caused by a lot of expensive houses mixed with some extremely expensive houses.

Living Sqft. The second value I will examine is the living space (in square feet). This value spans in between 290 to 13,540 sqft. The standard deviation is 918 and the mean around 2080 sqft. This points that most of the houses should be in between 1000 to 3000 sqft. When looking at the quarter distribution you get the following picture: - first 25% is 1137 sqft. in distance - second 25% spans 483 sqft. - third 25% is 640 sqft. - the last 25% is 9990 sqft.

This huge difference proves the previous estimation as at least 75% of the houses are in between 290 to 2550 sqft. in living space. This also kind of resembles the discoveries of looking at the house prices.

1.3 Simple Regression model

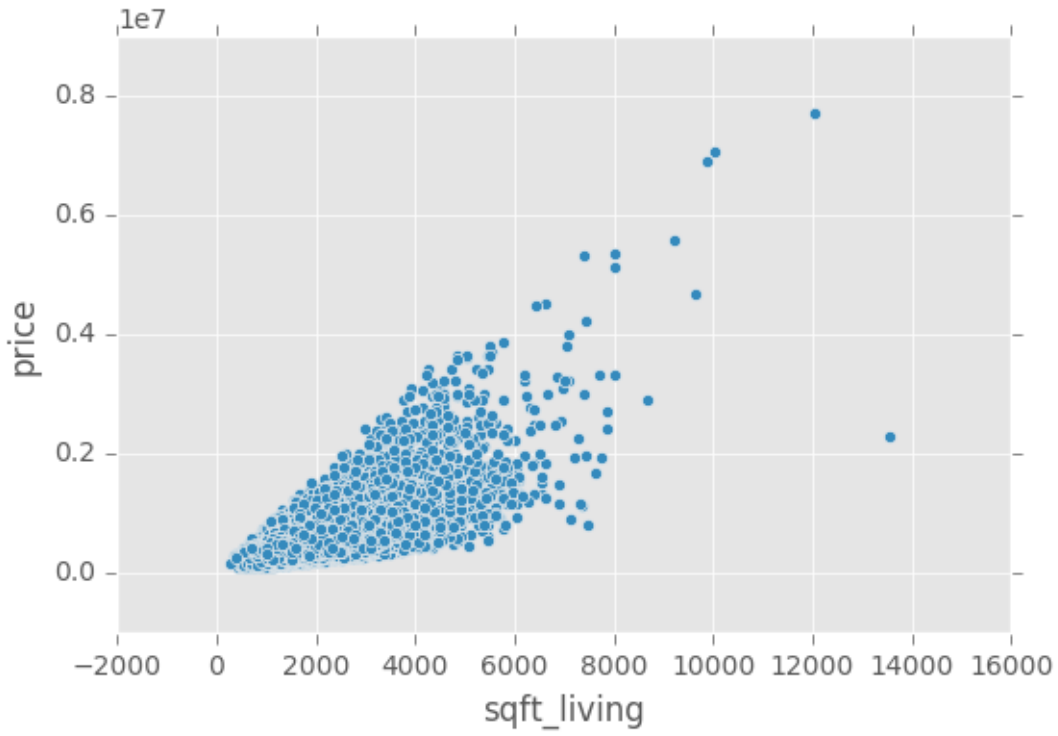
To create a benchmark model we will first do a simple linear regression on the relation between living space and price of the house. ### Relationship between price and living space ### First, I

will print the relationship between price and living space.

```
In [35]: import matplotlib.pyplot as plt
         import matplotlib
         matplotlib.style.use('ggplot')

         df.plot.scatter(x='sqft_living', y='price')
```

```
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1189905d0>
```



1.3.1 Linear Regression

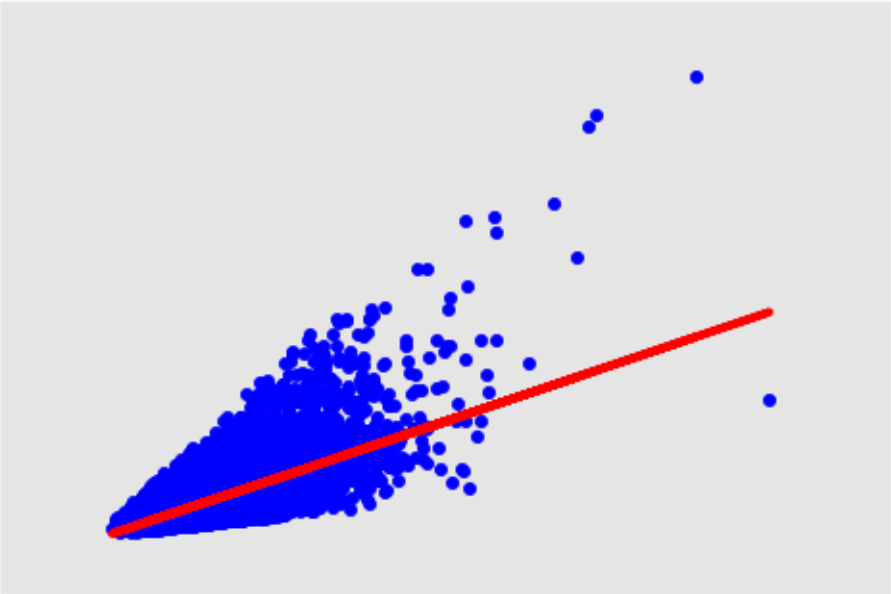
As seen here, most of the house are forming some kind of triangle on the graph which looks like it will work good with a linear regression.

```
In [36]: from sklearn import linear_model

         regression = linear_model.LinearRegression()
         regr_X = df['sqft_living'].reshape(21613,-1)
         regr_y = df['price']

         regression.fit(regr_X, regr_y)
         plt.scatter(regr_X, regr_y,  color='blue')
         plt.plot(regr_X, regression.predict(regr_X), color='red', linewidth=3)
```

```
plt.xticks(())
plt.yticks(())
plt.show()
```



1.3.2 R^2 Score of the simple regression

For comparision the R^2 Score of this linear regression will be calculated in the following.

```
In [37]: from sklearn.metrics import r2_score

         print "For the Simple Regression the R2 Score is: %.4f" % r2_score(regr_y,
```

```
For the Simple Regression the R2 Score is: 0.4929
```

1.4 Improving the Predicition model

This part is about finding a better metric for predicting future house sales regarding their price.

First, I will try to create a price heatmap to examine corelation between geo location and prices. Then I will take a look at redundant features and outliers.

1.4.1 Location based prices

House prices don't only depend on the size of the house or amount of rooms, but are also really dependant on the location of said house. To get an idea how the position might impact my data I analyse the relationship between location and price in my dataset.

```
In [38]: import gmaps
         gmaps.configure(api_key="AIzaSyDPWAl8lcrK9q-tOkrl64sGkxDnbWz47Ko")

         locations = df[["lat", "long"]]
         prices = df["price"]

         heatmap_layer = gmaps.heatmap_layer(locations, weights=prices)
         heatmap_layer.max_intensity = 7200000
         heatmap_layer.point_radius = 4

         fig = gmaps.figure()
         fig.add_layer(heatmap_layer)
         fig
```

This graph shows that there seems to be a real relationship between location and price. Especially in the center of Seattle the prices are much higher. There is also the town of Snoqualmie which is known for having a lot of highly educated inhabitants. That circumstance leads to the people of Snoqualmie having a substantially higher household income than the average.

Therefore, the housing prices are also higher in this area. The same reasons for higher living costs can also be applied to Seattle, which is much more attractive to live in for wealthy people.

1.4.2 Feature Relevance

To try to find out which features might be redundant, we will compare the regression score for each feature. This will give the score we get for predicting a feature just by all the other features. If a feature has a regressor score > 0.95 then this is highly likely to be redundant. If a feature has a regressor score of < 0 then it is not really well described by the rest of the data and might therefore be a important one.

```
In [39]: data = df.drop(labels = ["price","date"], axis=1)
         redundant = []
         significant = []
         for feature in data.keys():
             print 'Checking Feature "',feature,'"'
             # Make a copy of the DataFrame, using the 'drop' function to drop the
             X = data.drop(labels = feature, axis=1)

             from sklearn.cross_validation import train_test_split
             # Split the data into training and testing sets using the given featu
             y = data[feature]
             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

             from sklearn.tree import DecisionTreeRegressor
             # Create a decision tree regressor and fit it to the training set
             regressor = DecisionTreeRegressor(random_state = 42)
             regressor.fit(X_train, y_train)

             # Report the score of the prediction using the testing set
```

```
            score = regressor.score(X_test, y_test)
            print "The Regressor score is: ",score,"\n"
            if score > 0.95:
                redundant.append(feature)
            if score < 0:
                significant.append(feature)

        print "Redundant features(>0) : ",redundant,"\n"
        print "Significant features(<0) : ",significant,"\n"
```

```
Checking Feature " id "
The Regressor score is:  -0.13338979727

Checking Feature " bedrooms "
The Regressor score is:  -0.160352462696

Checking Feature " bathrooms "
The Regressor score is:  0.50795150982

Checking Feature " sqft_living "
The Regressor score is:  0.996110554063

Checking Feature " sqft_lot "
The Regressor score is:  0.269109294583

Checking Feature " floors "
The Regressor score is:  0.592701773203

Checking Feature " waterfront "
The Regressor score is:  0.293462831528

Checking Feature " view "
The Regressor score is:  -0.0166225785212

Checking Feature " condition "
The Regressor score is:  -0.333728697699

Checking Feature " grade "
The Regressor score is:  0.588520208236

Checking Feature " sqft_above "
The Regressor score is:  0.98419834569

Checking Feature " sqft_basement "
The Regressor score is:  0.973475041586

Checking Feature " yr_built "
The Regressor score is:  0.639089125308
```

```
Checking Feature " yr_renovated "
The Regressor score is:  -0.417516672101

Checking Feature " zipcode "
The Regressor score is:  0.976556891755

Checking Feature " lat "
The Regressor score is:  0.990196948347

Checking Feature " long "
The Regressor score is:  0.95619613868

Checking Feature " sqft_living15 "
The Regressor score is:  0.568577429132

Checking Feature " sqft_lot15 "
The Regressor score is:  0.182621375198

Redundant features(>0) :  ['sqft_living', 'sqft_above', 'sqft_basement', 'zipcode',

Significant features(<0) :  ['id', 'bedrooms', 'view', 'condition', 'yr_renovated']
```

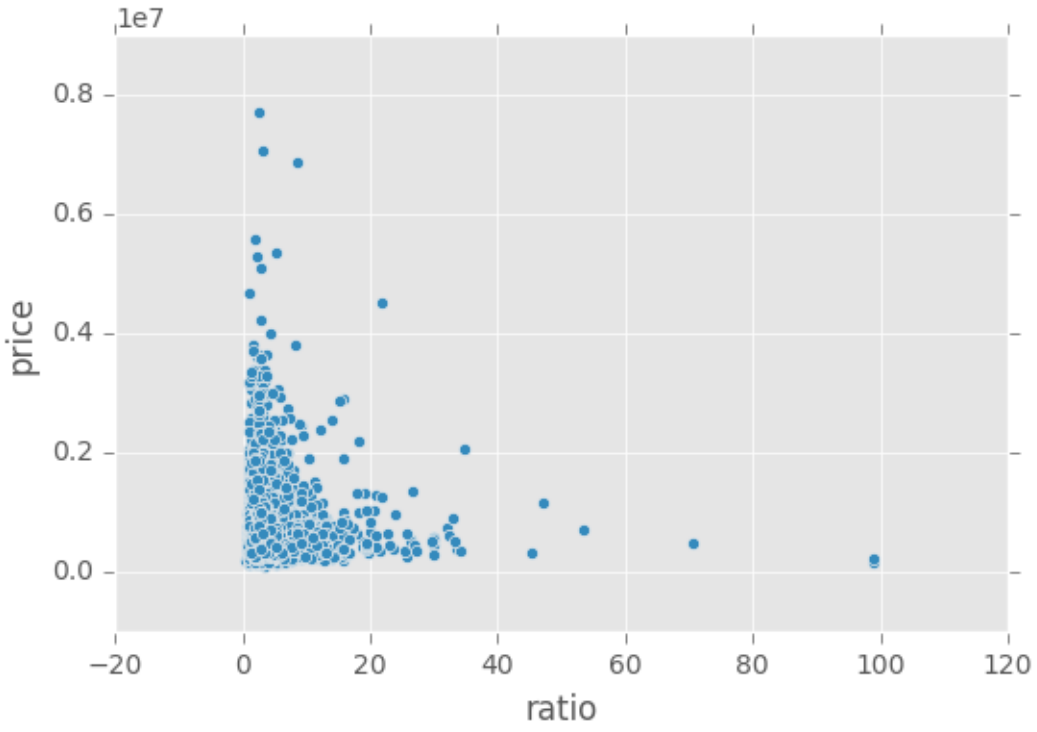
The result shows that sqft_living, sqft_above, sqft_basement, zipcode, latitude and longitude can be very good described by the the rest of the data when you leave them out. I think this might be explained, that all the square foot params describe the house and can therefore be explained by the number of rooms and other square foot values. Zipcode can explan the latitude and logitude and vice versa. Bedrooms, view, condition and year renovated are all features which are hard to be described but just looking at all other features.

sqft above and basement are both already included in sqft_living which is just the sum of it. To remove the above and basement feature I will check if the ratio of above to basement really has an impact on the price.

```
In [40]: basement = df['sqft_basement']
         no_basemet = basement[basement==0].keys()
         has_basement = df.drop(no_basemet).reset_index()
         above = has_basement['sqft_above']
         has_basement['ratio'] = above.div(has_basement['sqft_basement'], fill_valu

         has_basement.plot.scatter(x='ratio', y='price')

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0x118809850>
```



I can't really make out a big correlation for these 2 values and will therefore drop the sqft_above and basement from the dataset.

```
In [41]: data = data.drop('sqft_above', axis=1).drop('sqft_basement', axis=1)
```

1.5 pd.scatter_matrix(data, alpha = 0.3, figsize = (14,8), diagonal = 'kde');

1.5.1 Detecting Outliers

The second step to improve our learning behaviour is to find outliers and then remove them from the data set if needed. To detect outliers I will compare us Tukeys Method to compare the values of each feature for a entry to the difference between the 25th and 75th percentile for that feature. If the difference is more than 3 times higher, we consider a entry to be an extreme outlier. These extreme outliers will then be removed from the dataset

```
In [42]: # For each feature find the data points with extreme high or low values
         _out = []
         duplicate = []
         for feature in data.keys():

             # Calculate Q1 (25th percentile of the data) for the given feature
             Q1 = np.percentile(data[feature],25)

             # Calculate Q3 (75th percentile of the data) for the given feature
             Q3 = np.percentile(data[feature],75)
```

```
            # Use the interquartile range to calculate an outlier step (1.5 times
            step = 1.5*(Q3-Q1)
            extreme_step = 3*(Q3-Q1)

            # Display the outliers
            print "Calculating Data points considered outliers for the feature '{}
            _outliers = data[~((data[feature] >= Q1 - step) & (data[feature] <= Q3
            _extreme_outliers = data[~((data[feature] >= Q1 - extreme_step) & (dat
            for index in _outliers.index.values:
                if index in _out:
                    duplicate.append(index)
                else:
                    _out.append(index)
            #print _outliers[feature]

        # Print the datapoints which are displayed in more than 1 feature
        #print "duplicates: ",duplicate

        # OPTIONAL: Select the indices for data points you wish to remove
        print "extreme outliers: ",_extreme_outliers.index.values
        outliers  = _extreme_outliers.index.values

        # Remove the outliers, if any were specified
        good_data = data.drop(data.index[outliers]).reset_index(drop = True)
        prices = df["price"].drop(data.index[outliers]).reset_index(drop = True)
```

```
Calculating Data points considered outliers for the feature 'id'...
Calculating Data points considered outliers for the feature 'bedrooms'...
Calculating Data points considered outliers for the feature 'bathrooms'...
Calculating Data points considered outliers for the feature 'sqft_living'...
Calculating Data points considered outliers for the feature 'sqft_lot'...
Calculating Data points considered outliers for the feature 'floors'...
Calculating Data points considered outliers for the feature 'waterfront'...
Calculating Data points considered outliers for the feature 'view'...
Calculating Data points considered outliers for the feature 'condition'...
Calculating Data points considered outliers for the feature 'grade'...
Calculating Data points considered outliers for the feature 'yr_built'...
Calculating Data points considered outliers for the feature 'yr_renovated'...
Calculating Data points considered outliers for the feature 'zipcode'...
Calculating Data points considered outliers for the feature 'lat'...
Calculating Data points considered outliers for the feature 'long'...
Calculating Data points considered outliers for the feature 'sqft_living15'...
Calculating Data points considered outliers for the feature 'sqft_lot15'...
extreme outliers:  [    5    41    49 ..., 21525 21532 21548]
```

1.5.2 Using DecisionTreeRegressor

We will use the DecisionTreeRegressor. Then we will use cross validation to test the classifier.

In [43]:
```
from sklearn.metrics import make_scorer
from sklearn.grid_search import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.cross_validation import ShuffleSplit

def fit_model(X, y):
    """ Performs grid search over the 'max_depth' parameter for a
        decision tree regressor trained on the input data [X, y]. """

    cv_sets = ShuffleSplit(X.shape[0], n_iter = 10, test_size = 0.20, rand
    regressor = DecisionTreeRegressor()
    params = {'max_depth':[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}
    scoring_fnc = make_scorer(r2_score)
    grid = GridSearchCV(regressor,params,cv=cv_sets,scoring=scoring_fnc)
    grid = grid.fit(X, y)

    print "best r2 score: %s " % grid.best_score_
    # Return the optimal model after fitting the data
    return grid.best_estimator_
```

In [44]:
```
from sklearn import cross_validation
clf = linear_model.SGDRegressor()
X = good_data.drop("id",axis=1)
y = prices

fit_model(X,y)
```

```
best r2 score: 0.785861065443
```

Out[44]:
```
DecisionTreeRegressor(criterion='mse', max_depth=9, max_features=None,
                      max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                      splitter='best')
```

1.6 Conclusion

Comparing the Regression we made with the improved model to the easy linear regression we could improve the r2 score by .29 from .49 to .79. Therefore this new model can be used to make better estimations about the housing prices in the Kings Country area.

In [ ]: