Build model based on nearest properties that predict your property value

It might be usefull to know your neighbourhood avg price in order to predic the value of the existing property. So the question is How does that work ?.

Table of Contents

In [1]:
```
import housing_data as hd
import pandas as pd
from statsmodels.formula.api import ols
import numpy as np

%load_ext autoreload
%autoreload 2
```

Getting our nearest properties

It might be usefull to know your neighbourhood avg price in order to predic the value of the existing property. For that we need to come up with an algorithm that retrieves those nearest properties. see housing_data module for more details.

In [2]:
```
# Load dataset
data = hd.load_housing_data(with_cat_columns=False)
```

In [3]:
```
# Try to get Avg price per sqft base on nearest neighbors within radius (in
property_ds = data.iloc[0]  # Selected property
radius = 1  # 1 mile around selected property.

# Retrieve nearest properties
closet_properties_df = hd.get_closest_properties(data, property_ds, radius)

# Calculate Avg price per sqft and compare it against global Avg price/sqft
print('Average price/sqft_living : ', closet_properties_df['price'].mean()/
print('Global Average price/sqft_living : ', data['price'].mean()/data['sqf
```

```
Average price/sqft_living :  194.5573950948879
Global Average price/sqft_living :  259.7177804188506
```

Conclusion : Now that we are able to calculate the Avg price around the property, we can generalise the calculus to the entire dataset.

In [4]:
```
# ## Running nearest properties avg price per sqft against dataset
# Run the prediction on a smaller dataset as the process take ages
# (~20mins depending on your machine).
enriched_data = data.copy()
sample_data = enriched_data[:1000]
```

In [6]:
```
# Enriched the data by adding the avg sqft price of neighbours
sample_data['price_sqft'] = hd.get_price_per_sqft_living(sample_data)
sample_data.head()
```

```
--- 62.601375102996826 seconds ---
```

```
/Users/flatironstudentaccount/anaconda3/lib/python3.7/site-packages/ipyke
rnel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydata.or
g/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
```

Out[6]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view
0	7129300520	2014-10-13	221900.0	3	1.00	1180	5650	1.0	False	0.0
1	6414100192	2014-12-09	538000.0	3	2.25	2570	7242	2.0	False	0.0
2	5631500400	2015-02-25	180000.0	2	1.00	770	10000	1.0	False	0.0
3	2487200875	2014-12-09	604000.0	4	3.00	1960	5000	1.0	False	0.0
4	1954400510	2015-02-18	510000.0	3	2.00	1680	8080	1.0	False	0.0

5 rows × 22 columns

Convert categorical variables

In [7]:
```
# We are now going to try to run a simple Regression against our dataset
cat_variables = ['grade', 'condition']
cleaned_data = hd.convert_categorical_variables(sample_data, cat_variables,
```

Scale and normalise variables

In [ ]:
```
# Plotting variables to see distribution and skewness
x_cols = ['price', 'sqft_living', 'price_sqft']
pd.plotting.scatter_matrix(cleaned_data[x_cols], figsize=(10,12));
```

Conclusion : We can see that these variables are not normally distriburted Some log normalisation is needed in order to remove skewness.

In [8]:
```
# Scale Variables data
log_sqft_living = np.log(cleaned_data['sqft_living'])
log_price_sqft = np.log(cleaned_data['price_sqft'])

# Scaling the variables
scaled_sqft_living = (log_sqft_living-min(log_sqft_living))/(max(log_sqft_l
scaled_price_sqft = (log_price_sqft-min(log_price_sqft))/(max(log_price_sqf

data_fin = pd.DataFrame([])
data_fin['sqft_living'] = scaled_sqft_living
data_fin['price_sqft'] = scaled_price_sqft

scaled_data = cleaned_data.drop(['sqft_living', 'price_sqft'], axis=1)
scaled_data = pd.concat([scaled_data, data_fin], axis=1)
```

Building the model

We are now going to try to run a simple Regression against our dataset

In [9]:
```
# Build formula
# Notes that we are especting a corrolation between sqft_living and price_s
formula = 'price ~ sqft_living * price_sqft -1'
```

In [10]:
```
# Run simple prediction
model = ols(formula=formula, data=scaled_data).fit()
model.summary()
```

Out[10]:

OLS Regression Results

Dep. Variable:	price	**R-squared:**	0.893
Model:	OLS	**Adj. R-squared:**	0.893
Method:	Least Squares	**F-statistic:**	2773.
Date:	Wed, 08 May 2019	**Prob (F-statistic):**	0.00
Time:	16:28:48	**Log-Likelihood:**	-13640.
No. Observations:	1000	**AIC:**	2.729e+04
Df Residuals:	997	**BIC:**	2.730e+04
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
sqft_living	2.771e+05	2.71e+04	10.222	0.000	2.24e+05	3.3e+05
price_sqft	-5.208e+05	5.43e+04	-9.589	0.000	-6.27e+05	-4.14e+05
sqft_living:price_sqft	2.356e+06	1.02e+05	23.206	0.000	2.16e+06	2.56e+06

Omnibus:	988.904	**Durbin-Watson:**	1.908
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	66120.301
Skew:	4.466	**Prob(JB):**	0.00
Kurtosis:	41.821	**Cond. No.**	13.7

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

Observations : The Adj. R-squared is pretty high and our variables coef p-values are low. This looks much better than the zipcode model.

Conclusion

We saw that the model built based on lat/lon proximity is more accurate than the one based on zipcode. This might be true because of price variation within a specific zipcode. We are now going to try to run a simple Regression against our dataset