Capstone Project - Real Estate Development

Applied Data Science Capstone by IBM/Coursera

Marcos Geraldo

Table of contents

1. Introduction: Business Problem

This project will provide insights about capital gain on real estate investemnts.

It wil be targeted to landlords who are evaluating the impact of home improvements projects in the selling price of their properties.

It will also provide a model to estimate the listing price that fits the market valuation of a particular house.

It will use current data published for the city of interest, and use it to stablish the relative weights of the key elements that drive the price of a house.

It will use Foursquare Data to evaluate the distance to relevant venues, and evaluate the weight of those elements in the listing price of a property.

2. Data

According to the problem definition, the relevant data to understand price valuation, are the following:

- selling price
- listing price (as a proxy for selling price, that might not be public)
- number and distance of venues

To avoid market variations the data will come from current market conditions. The candidates are real state web sites that publish and share freely properties and listing prices:

- Realtor

- FourSquare

The values

- Year of construction
- Constructed suface
- Bedrooms
- Bathrooms
- Garages
- Stories
- School ratings
- Number of venues by category
- Distance to venues
- others to be found

2.1 Realtor.com

After trying some APIs, I will use Realtor as the main source for data collection, due to its reliability, and simplicity.

Realtor offers multiple API:

- list-for-sale
- detail
- list-sold
- list-similar-homes
- list-for-rent
- list-by-mls
- list-similar-rental-homes

2.1.1 list-for-sale API

This API shows properties for sale inin groups of 200. Here is an example of how I read two pages using the variable Offset:

To read all the data, I will need a Function to read each page of the query Here I defined two functions:

- **read_realtor**: requests a specific page for a city. This query can get up to 200 records
- **list_for_sale**: uses *read_realtor* function to collect all the avalable pages for that city

To explore the data I will get the properties published for San Ramon

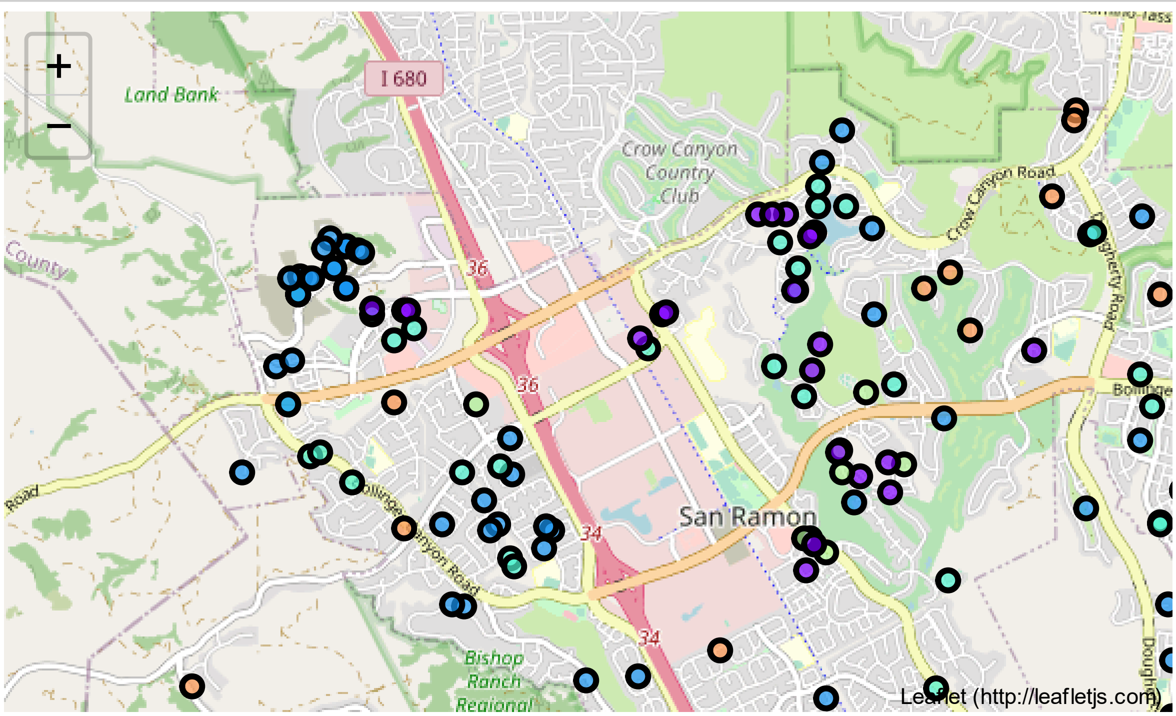
Here is a visual representation of the proprties in currently in the city selected:

In [17]:

```python
import folium
import matplotlib.cm as cm
import matplotlib.colors as colors
#setting colors
number_types = len(resp['beds'].unique())
colors_array = cm.rainbow(np.linspace(0, 1,number_types))
rainbow = pd.DataFrame([colors.rgb2hex(i) for i in colors_array])
rainbow.index = resp['beds'].unique()
# centring the screen:
latitude = (resp['address.lat'].max()+resp['address.lat'].min())/2
longitude = (resp['address.lon'].max()+resp['address.lon'].min())/2
# create map of Toronto using latitude and longitude values
map_tto = folium.Map(location=[latitude, longitude], zoom_start=13)
# add markers to map
for lat, lng, address, neighborhood, beds, price, size in zip(resp['addres
                                                              resp['addres
                                                              resp['beds']
                                                              resp['buildi
    label = '{}, {}, {} beds, ${}, {}sqft, ({})'.format(address, neighborh
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker(
        [lat, lng],
        radius=5,
        popup=label,
        color='black',
        fill=True,
        fill_color=rainbow.loc[beds][0],
        fill_opacity=0.7,
        parse_html=False).add_to(map_tto)

map_tto
```

Out[17]:



2.1.2 Detail API

I will have to use the details API to get details such as

- Schools raitings
- Year Built
- Number of Stories

In [225]:
```
json_normalize(response.json()['properties'][0]['schools'])
```

Out[225]:

	distance_in_miles	education_levels	funding_type	grades.range.high	grades.range.low	gr
0	0.6	[elementary]	public	5	K	
1	0.8	[middle]	public	8	6	
2	0.4	[high]	public	12	9	
3	0.7	[elementary]	public	5	K	
4	1.6	[middle]	public	8	6	

Each property has a list of schools. I will get the average of the raitings as the index of schools quality.

The function **get_details** that gets those three details:

- School rating
- Stories
- Year Built

Different kinsd of properties have different JSON structures, so this function needs to react correctly when the data is not found.

In [35]:
```
resp_detail = get_details('M2269810031')
```

In [36]:
```
resp_detail
```

Out[36]:

	school_rating	stories	year_built
0	8.0	None	1974

2.1.3 the data from the two API

I will select the relevant columns from the full list of properties, and then I will use *get_details* for each property to get these three values

Finally here is an initial data set to work with.

In [44]:

```
train_data.tail()
```

Out[44]:

	property_id	listing_id	address.city	address.county	address.lat	address.lon	addres
217	M9104371519	2913840789	San Ramon	Contra Costa	37.766450	-121.917223	
218	M9154671243	2913839368	San Ramon	NaN	37.766450	-121.917223	
219	M1998497830	2913826667	San Ramon	Contra Costa	37.781273	-121.937082	
220	M1013154486	2913022228	San Ramon	Contra Costa	37.788706	-121.948041	
221	M9048153993	2861209906	San Ramon	NaN	37.766450	-121.917223	

2.2 FourSquare API

API Dcocumentation [here]

First step will be to get the Venues Close to a property

The function **GetNearbyVenues** finds the venues that are in a defined radius (RADIUS) from the location (Latitude and Longitude) of the property. I am using the coordinates prvided by Realtor API.

The function ***dist_coord*** function finds the distance between two sets of coordinated

Obtaining the property ids and geolocation for each of the properties

fs_resp will contain a row per venue for each of the properties in the list.

In [12]:
```
fs_resp.head()
```

Out[12]:

	property_id	address.lat	address.lon	Venue	Venue Latitude	Venue Longitude	Venue Category
0	M1010694934	37.783997	-121.948831	Canyon Lakes Pool	37.785593	-121.949975	Pool
1	M1010694934	37.783997	-121.948831	San Ramon Country Club	37.782490	-121.945542	Tennis Court
2	M1010694934	37.783997	-121.948831	Coyote Crossing Park	37.777168	-121.942039	Park
3	M1010694934	37.783997	-121.948831	Jeff's Kitchens	37.790089	-121.956863	Construction & Landscaping
4	M1010694934	37.783997	-121.948831	Inspiration Point Oakland Hills Ca	37.783339	-121.937559	Trail

Adding the distance as an additional column to the data set

In [14]:
```
fs_resp.tail()
```

Out[14]:

	property_id	address.lat	address.lon	Venue	Venue Latitude	Venue Longitude	Venue Category
2645	M9048153993	37.76645	-121.917223	Raashi Design	37.763336	-121.915154	Furniture / Home Store
2646	M9048153993	37.76645	-121.917223	Bark And Ride	37.771518	-121.917791	Dog Run
2647	M9048153993	37.76645	-121.917223	SAN PETER HANDYMAN	37.762267	-121.912112	Home Service
2648	M9048153993	37.76645	-121.917223	Creekside Park	37.770470	-121.922858	Playground
2649	M9048153993	37.76645	-121.917223	Indian Hotspot	37.774059	-121.922775	Indian Restaurant

Next Step will be to convert the categorical variable "Venue Category" in dummy variables

134 categories are too many to so few data points. The categories need to be simplified by aggregating them by hierarchies. We can get the **hierarchy of categories from Foursquare**

The categories com in JSON format. I wil have to navigate the tree to find the macro category that correcponds to each category. This is a list of the Macro Categories we want to use:

In [18]:
```
json_normalize(categories['categories'])[['name']]
```

Out[18]:

	name
0	Arts & Entertainment
1	College & University
2	Event
3	Food
4	Nightlife Spot
5	Outdoors & Recreation
6	Professional & Other Places
7	Residence
8	Shop & Service
9	Travel & Transport

I will define two functions to navigate the JSON:

- **count_branches** gives the number of leafs in a branch
- **list_leafs** gives a list with all the leafs in a branch

now for each Macro category in the tree, I will get all the child categories

In [21]:
```
table_cat.head()
```

Out[21]:

	0
Amphitheater	Arts & Entertainment
Aquarium	Arts & Entertainment
Arcade	Arts & Entertainment
Art Gallery	Arts & Entertainment
Bowling Alley	Arts & Entertainment

In [22]:
```
table_cat.tail()
```

Out[22]:

	0
Tram Station	Travel & Transport
Transportation Service	Travel & Transport
Travel Lounge	Travel & Transport
Truck Stop	Travel & Transport
Tunnel	Travel & Transport

Now, i find the Macro Cateogry for Catogry in fs_resp

fs_resp dataset contains each venue for each property, with a macro category based on the child category. Here is a view of some of its records:

In [24]:
```
fs_resp
```

Out[24]:

	property_id	address.lat	address.lon	Venue	Venue Latitude	Venue Longitude	Ca
0	M1010694934	37.783997	-121.948831	Canyon Lakes Pool	37.785593	-121.949975	
1	M1010694934	37.783997	-121.948831	San Ramon Country Club	37.782490	-121.945542	Tenni
2	M1010694934	37.783997	-121.948831	Coyote Crossing Park	37.777168	-121.942039	
3	M1010694934	37.783997	-121.948831	Jeff's Kitchens	37.790089	-121.956863	Cons Lands
4	M1010694934	37.783997	-121.948831	Inspiration Point Oakland Hills Ca	37.783339	-121.937559	
5	[illegible]	[illegible]	[illegible]	Jack in the	[illegible]	[illegible]	Fa

To convert this list to a set of useful features I will extend the Macro categories into dummies, and then add them up to have a dataset with a row for each property.

In [28]:

```
macro_cat = fs_dummies.groupby(by=['property_id']).sum()
macro_cat.head()
```

Out[28]:

property_id	Arts & Entertainment	Food	Outdoors & Recreation	Professional & Other Places	Shop & Service	Travel & Transport
M1003675301	0	0	5	0	2	0
M1007260231	0	1	3	0	0	0
M1008272648	0	0	3	0	1	0
M1010204240	0	1	3	0	1	0
M1010694934	0	0	4	0	1	0

In [33]:
```
train_data.head().T
```

Out[33]:

	0	1	2	3	
Unnamed: 0	0	1	2	3	
property_id	M1010694934	M2514786495	M2660320517	M1066182116	M29948
listing_id	2920062379	2920057281	2920056809	2920050827	2920C
address.city	San Ramon	San Ramon	San Ramon	San Ramon	San F
address.county	Contra Costa	Contra Costa	Contra Costa	Contra Costa	Contra
address.lat	37.784	37.7383	37.7439	37.7675	
address.lon	-121.949	-121.953	-121.946	-121.945	-1.
address.neighborhood_name	Dougherty Hills	Westside	Southern San Ramon	Canyon Lakes South	Southe F
address.postal_code	94582	94583	94583	94582	
baths_full	2	2	2	2	
baths_half	NaN	1	NaN	NaN	
beds	2	4	4	2	
building_size.size	1272	1995	1448	1079	
lot_size.size	NaN	2550	8000	NaN	
prop_type	condo	single_family	single_family	condo	single_
prop_status	for_sale	for_sale	for_sale	for_sale	fc
price	699000	998000	895000	615000	8
school_rating	9.16667	9	9.16667	9.16667	9
stories	1	2	1	1	
year_built	1990	1999	1971	1988	
Arts & Entertainment	0	0	0	0	
Food	0	1	1	16	
Outdoors & Recreation	4	3	7	6	
Professional & Other Places	0	0	0	0	
Shop & Service	1	3	2	7	
Travel & Transport	0	0	0	1	

Nulls and sustitutions

baths_half appear as null when the house doesn't have half baths. I will replace the Na by 0

In [97]:
```
train_data.loc[train_data['baths_half'].isna(),['baths_half']]=0
```

address.neighborhood_name To replace NULLs here I will find the closest neighborhood based on the average distances that I have for each neigborhood

In [123]:
```
nbh_geo = train_data.groupby(by=['address.neighborhood_name']).mean()[['ad
nbh_geo
```

Out[123]:

address.neighborhood_name	address.lat	address.lon
Alta Mira	37.784891	-121.924698
Canyon Lakes South	37.766972	-121.943629
Crow Canyon	37.776769	-121.981989
Dougherty Hills	37.782341	-121.948261
Dougherty Valley	37.766741	-121.919309
Norris Canyon Estates	37.747331	-121.994462
Royal Vista	37.740749	-121.930868
Southern San Ramon	37.742544	-121.939478
Twin Creeks	37.764326	-121.978379
Westside	37.740588	-121.956019
Windemere	37.761618	-121.898336

The function ***closer_nbh*** returns the name of the closest neighborhood

In [177]:
```
def closer_nbh(lat,lon):
    dists = dist_coord(nbh_geo[['address.lat']].values,
                       nbh_geo[['address.lon']].values,
                       np.repeat(lat,len(nbh_geo)),
                       np.repeat(lon,len(nbh_geo))
                      )
    nbh_2 = nbh_geo.copy()
    nbh_2['dist'] = dists
    return nbh_2['dist'].idxmin()
```

year_built is NA in many cases. I will find the typical year for the neigborhood to replace the NAN values in this column

In [108]:

```
typical_year = pd.DataFrame(train_data.groupby(by=['address.neighborhood_r
typical_year
```

Out[108]:

	year_built
address.neighborhood_name	
Alta Mira	1994.5
Canyon Lakes South	1988.0
Crow Canyon	1996.0
Dougherty Hills	1989.0
Dougherty Valley	2009.0
Norris Canyon Estates	2005.5
Royal Vista	1977.0
Southern San Ramon	1976.0
Twin Creeks	1979.0
Westside	1998.0
Windemere	2006.0

stories by default I will assume 1

In [220]:

```
train_data.loc[train_data['stories'].isna(),'stories']=1
```

Adding Dummies

the columns ***postal_code*** and ***neighborhood_name*** will require dummy variables

In [292]:

```
print(train_data['address.postal_code'].unique())
print(train_data['address.neighborhood_name'].unique())

dummies_zip = pd.get_dummies(train_data['address.postal_code'], prefix='zi
dummies_nbh = pd.get_dummies(train_data['address.neighborhood_name'], pref
baths = pd.DataFrame(train_data['baths_full'] + train_data['baths_half']/2
```

```
['94582' '94583' '94588']
['Dougherty Hills' 'Westside' 'Southern San Ramon' 'Canyon Lakes South'
 'Dougherty Valley' 'Windemere' 'Twin Creeks' 'Crow Canyon' 'Alta Mira'
 'Royal Vista' 'Norris Canyon Estates']
```

In [335]:

```python
data = pd.concat([train_data[['price','building_size.size','beds','stories
                              'school_rating','year_built',
                              'Arts & Entertainment','Food','Outdoors & Rec
                             ]],
                 dummies_zip,
                 dummies_nbh,
                 baths], axis=1)
data = data[data.columns.tolist()[:3]+ data.columns.tolist()[-1:]+ data.co
data.head().T
```

Out[335]:

	0	1	2	3	4
price	699000.000000	998000.0	895000.000000	615000.000000	819000.000000
building_size.size	1272.000000	1995.0	1448.000000	1079.000000	1500.000000
beds	2.000000	4.0	4.000000	2.000000	3.000000
baths	2.000000	2.5	2.000000	2.000000	2.000000
stories	1.000000	2.0	1.000000	1.000000	1.000000
school_rating	9.166667	9.0	9.166667	9.166667	9.166667
year_built	1990.000000	1999.0	1971.000000	1988.000000	1978.000000
Arts & Entertainment	0.000000	0.0	0.000000	0.000000	0.000000
Food	0.000000	1.0	1.000000	16.000000	0.000000
Outdoors & Recreation	4.000000	3.0	7.000000	6.000000	2.000000
Professional & Other Places	0.000000	0.0	0.000000	0.000000	0.000000
Shop & Service	1.000000	3.0	2.000000	7.000000	2.000000
Travel & Transport	0.000000	0.0	0.000000	1.000000	1.000000
zip_94583	0.000000	1.0	1.000000	0.000000	1.000000
zip_94588	0.000000	0.0	0.000000	0.000000	0.000000
nbh_Canyon Lakes South	0.000000	0.0	0.000000	1.000000	0.000000
nbh_Crow Canyon	0.000000	0.0	0.000000	0.000000	0.000000
nbh_Dougherty Hills	1.000000	0.0	0.000000	0.000000	0.000000
nbh_Dougherty Valley	0.000000	0.0	0.000000	0.000000	0.000000
nbh_Norris Canyon Estates	0.000000	0.0	0.000000	0.000000	0.000000
nbh_Royal Vista	0.000000	0.0	0.000000	0.000000	0.000000

	0	1	2	3	4
nbh_Southern San Ramon	0.000000	0.0	1.000000	0.000000	1.000000
nbh_Twin Creeks	0.000000	0.0	0.000000	0.000000	0.000000
nbh_Westside	0.000000	1.0	0.000000	0.000000	0.000000
nbh_Windemere	0.000000	0.0	0.000000	0.000000	0.000000

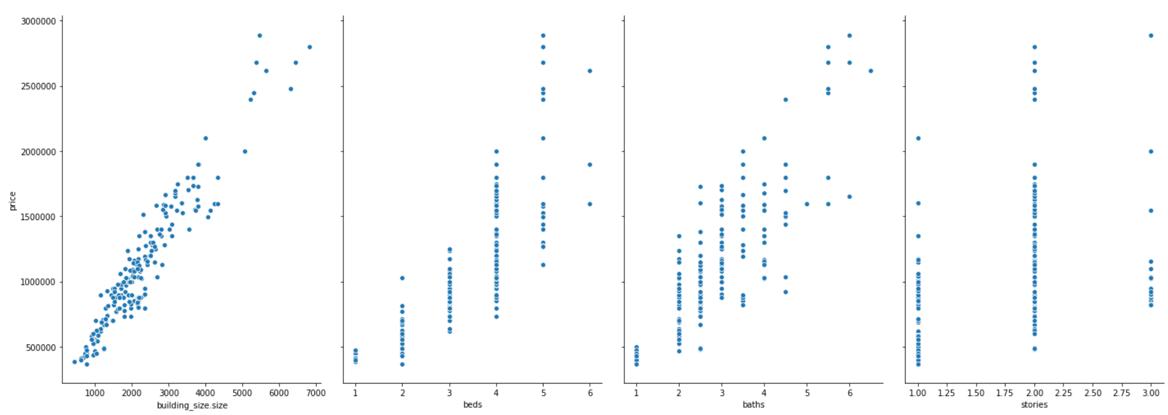
3. Methodology

Now I need to create a model that predicts the listing price. \n Opaque models will not provide the insights about the retaive importance of each feature.\n To have good insights about the contribution of each factor I will use a **linear model**, which means I need to veryfy how the features are correlated among each other

In [343]:
```
features_cols = data.columns.tolist()[1:]
```

In [360]:
```
import seaborn as sns
sns.pairplot(data, x_vars=features_cols[:4], y_vars='price', size=7, aspec
```

Out[360]: <seaborn.axisgrid.PairGrid at 0x1b63eb82d30>



In [372]:

```
data.iloc[:,:7].corr()
```

Out[372]:

	price	building_size.size	beds	baths	stories	school_rating
price	1.000000	0.954159	0.799433	0.821871	0.350076	-0.280514
building_size.size	0.954159	1.000000	0.790917	0.847030	0.427817	-0.339301
beds	0.799433	0.790917	1.000000	0.741374	0.396574	-0.355202
baths	0.821871	0.847030	0.741374	1.000000	0.544320	-0.425073
stories	0.350076	0.427817	0.396574	0.544320	1.000000	-0.437946
school_rating	-0.280514	-0.339301	-0.355202	-0.425073	-0.437946	1.000000
year_built	0.397562	0.412954	0.300495	0.521292	0.581429	-0.649694

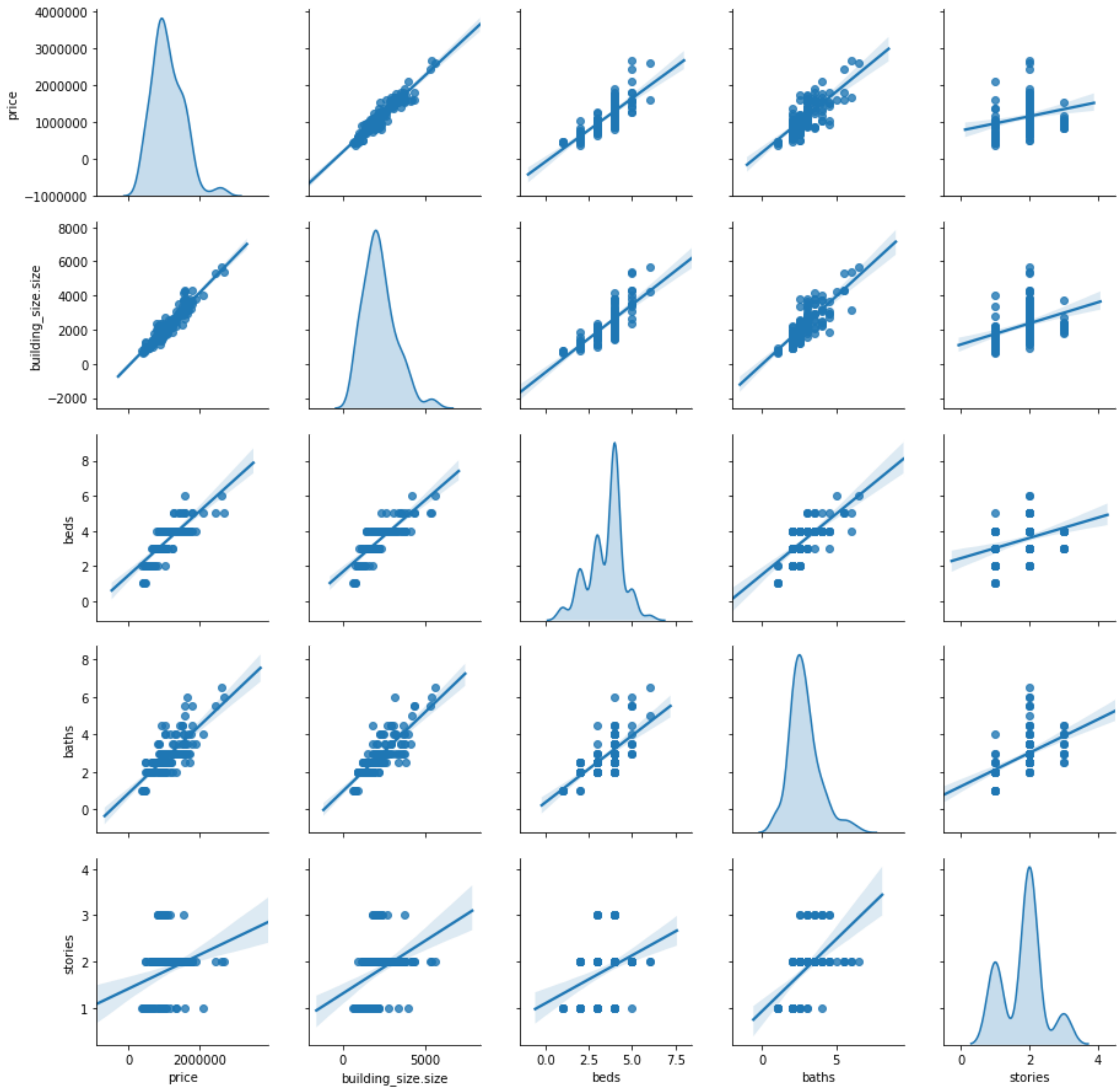
In [350]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data[features_cols], d
```

In [352]:

```
train_dataset = X_train[features_cols[:4]].copy()
train_dataset.insert(0, "price", y_train)
_ = sns.pairplot(train_dataset, kind='reg', diag_kind='kde')
```



Selecting the right combination of features

Several variables are correlated among them. I wan to avoid colinearity in to get meaningful coeficients, and at the same time, I want to have an accurate model. To determine the best set of features I will rum several combinations of features, and compare them based on RMSE and R2.

model 1	model 2	model 3	model 4	model 5
'building_size.size'	'building_size.size'	'building_size.size',	'building_size.size',	'building_size.size',
'beds'	'beds'	'beds'	'beds'	'beds'
'baths'	'baths'	'baths'	'baths'	'baths'
'stories'	'stories'	'stories'	'stories'	'stories'
'school_rating'		'school_rating'	'school_rating'	'school_rating'
'year_built'		'year_built'	'year_built'	'year_built'

model 1	model 2	model 3	model 4	model 5
'Arts & Entertainment'				
'Food'				
'Outdoors & Recreation'				
'Professional & Other Places'				
'Shop & Service'				
'Travel & Transport'				
'zip_94583'			'zip_94583'	'zip_94583'
'zip_94588'			'zip_94588'	'zip_94588'
'nbh_Canyon Lakes South'				'nbh_Canyon Lakes South'
'nbh_Crow Canyon'				'nbh_Crow Canyon'
'nbh_Dougherty Hills'				'nbh_Dougherty Hills'
'nbh_Dougherty Valley'				'nbh_Dougherty Valley'
'nbh_Norris Canyon Estates'				'nbh_Norris Canyon Estates'
'nbh_Royal Vista'				'nbh_Royal Vista'
'nbh_Southern San Ramon'				'nbh_Southern San Ramon'
'nbh_Twin Creeks'				'nbh_Twin Creeks'
'nbh_Westside'				'nbh_Westside'
'nbh_Windemere'				'nbh_Windemere'

Comparison of Performance for the different models

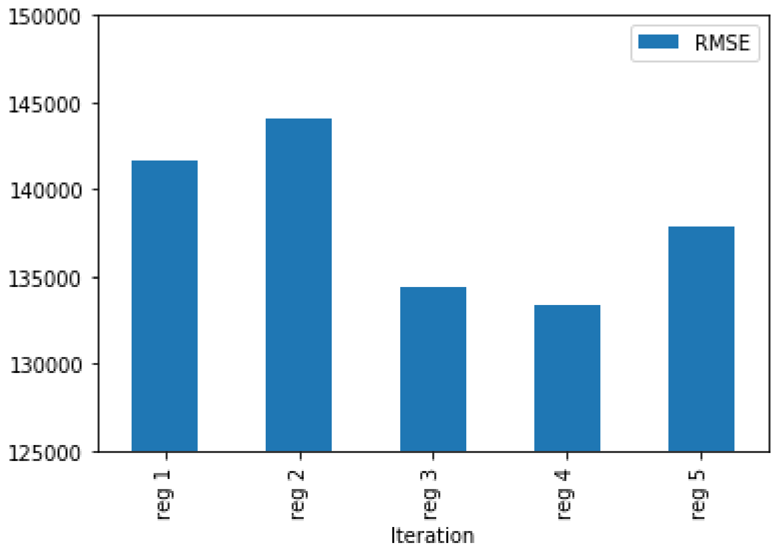
From the 5 sets of features used to solve this problem, model 4 was the one with smalles Root Mean Squared Error. The graph below shows the comparison between models.

In [423]:

```
RMSE = {'Iteration': ['reg 1','reg 2','reg 3','reg 4','reg 5'],
        'RMSE':[np.sqrt(mean_squared_error(y_test, price_pred_1)),
        np.sqrt(mean_squared_error(y_test, price_pred_2)),
        np.sqrt(mean_squared_error(y_test, price_pred_3)),
        np.sqrt(mean_squared_error(y_test, price_pred_4)),
        np.sqrt(mean_squared_error(y_test, price_pred_5))
       ]}
pd.DataFrame(RMSE).plot.bar(x='Iteration',y='RMSE',ylim=(125000,150000))
```

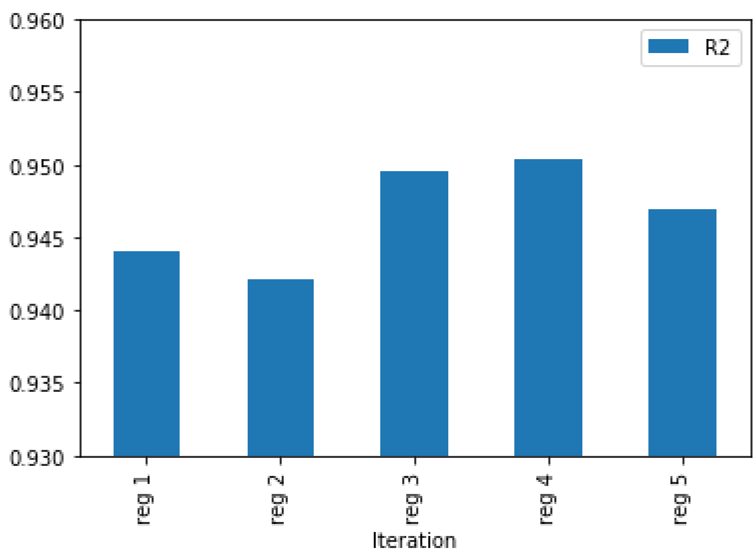
Out[423]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b63fc14320>
```



In [537]:
```
R2 = {'Iteration': ['reg 1','reg 2','reg 3','reg 4','reg 5'],
      'R2':[r2_score(y_test, price_pred_1),
              r2_score(y_test, price_pred_2),
              r2_score(y_test, price_pred_3),
              r2_score(y_test, price_pred_4),
              r2_score(y_test, price_pred_5)
       ]}
pd.DataFrame(R2).plot.bar(x='Iteration',y='R2',ylim=(0.93,.96))
```

Out[537]: <matplotlib.axes._subplots.AxesSubplot at 0x1b6372ef6a0>



The winning model's coeficients

From the 5 sets of features used to solve this problem, model 4 was the one with smalles Root Mean Squared Error. The graph below shows the comparison between models.

The winning combination contains:

- Built Surface
- Number of Bedrooms
- Number of Baths
- School ratings

- Year Built
- Postal code

In [424]:
```
pd.DataFrame(reg_4.coef_,index=data[cols_4].columns, columns=['Coeficient'
```

Out[424]:

	Coeficient
building_size.size	363.142507
beds	61082.726749
baths	17371.953626
stories	-95848.255351
school_rating	91867.688896
year_built	3824.983023
zip_94583	17282.230079
zip_94588	272997.346664

In [504]:
```
price_gain(data.loc[100,cols_4].values,
           [100,1,1,0,0,0,0,0]
           )
```

```
The property gains $114,768.93 in value for adding
100 sqft
1 bedrooms,
1 bathrooms, and
0 floors.
Original estimated price: $1,056,036.28
New estimated price:    $941,267.35
```

Out[504]:
```
114768.93102911115
```

4.Results

[https://www.homeadvisor.com/cost/bathrooms/#half (https://www.homeadvisor.com/cost/bathrooms/#half)](https://www.homeadvisor.com/cost/bathrooms/#half)

The evaluation of the added value to the property can be done by using reference data for the costs and typical sizes of the expansions done to a typical house.

Bath room:

- Area = 40 sqft
- Average cost to convert an existing space = $5,000
- Average cost to build an new space = $22,000

First let's evaluate a bathroom that is added to the house, adding squared feet to the house.

In [548]:
```
gain_bath = price_gain(data.loc[100,cols_4].values,
            [40,0,1,0,0,0,0,0]
            )
```

```
The property gains $31,897.65 in value for adding
40 sqft
0 bedrooms,
1 bathrooms, and
0 floors.
Original estimated price: $973,165.00
New estimated price:    $941,267.35
```

In [549]:
```
cost_bath = 22000
ROI_bath = gain_bath/cost_bath
print('Consideing a cost of {}, the ROI of the investment would be {:.2%}'
```

```
Consideing a cost of 22000, the ROI of the investment would be 144.99%
```

Now let's see what happens if the new bathroom is built reusing space in the house.

In [551]:
```
gain_bath = price_gain(data.loc[100,cols_4].values,
            [0,0,1,0,0,0,0,0]
            )
```

```
The property gains $17,371.95 in value for adding
0 sqft
0 bedrooms,
1 bathrooms, and
0 floors.
Original estimated price: $958,639.30
New estimated price:    $941,267.35
```

In [553]:
```
cost_bath = 17000
ROI_bath = gain_bath/cost_bath
print('Consideing a cost of {}, the ROI of the investment would be {:.2%}'
```

```
Consideing a cost of 17000, the ROI of the investment would be 102.19%
```

Half bathroom:

- Area = 25 sqft
- Average cost = $20,000

In [519]:

```
gain_half= price_gain(data.loc[100,cols_4].values,
           [25,0,.5,0,0,0,0,0]
           )
```

```
The property gains $17,764.54 in value for adding
25 sqft
0 bedrooms,
0.5 bathrooms, and
0 floors.
Original estimated price: $959,031.89
New estimated price:    $941,267.35
```

In [520]:

```
cost_half = 20000
ROI_half = gain_half/cost_half
print('Consideing a cost of {}, the ROI of the investment would be {:.2%}'
```

```
Consideing a cost of 20000, the ROI of the investment would be 88.82%
```

Second Floor:

- Cost = from $100,000 to $150,000
- Area = from 600 sqft to 1500 sqft

In [525]:

```
gain_floor= price_gain(data.loc[100,cols_4].values,
           [700,2,1,1,0,0,0,0]
           )
```

```
The property gains $297,888.91 in value for adding
700 sqft
2 bedrooms,
1 bathrooms, and
1 floors.
Original estimated price: $1,239,156.26
New estimated price:    $941,267.35
```

In [526]:

```
cost_floor = 110000
ROI_floor = gain_floor/cost_floor
print('Consideing a cost of {}, the ROI of the investment would be {:.2%}'
```

```
Consideing a cost of 110000, the ROI of the investment would be 270.81%
```

Adding a new room:

- Area = 200 sqft to 600 sqft
- Cost = $ 45,000 average (from 10k to 125k)

In [531]:
```
gain_room= price_gain(data.loc[100,cols_4].values,
            [400,1,0,0,0,0,0,0]
            )
```

```
The property gains $206,339.73 in value for adding
400 sqft
1 bedrooms,
0 bathrooms, and
0 floors.
Original estimated price: $1,147,607.08
New estimated price:    $941,267.35
```

In [532]:
```
cost_room = 60000
ROI_room = gain_room/cost_room
print('Consideing a cost of {}, the ROI of the investment would be {:.2%}'
```

```
Consideing a cost of 60000, the ROI of the investment would be 343.90%
```

5. Discussion

5.1 Data Issues

Here are the consideration I made to eliminate or replace null values in the data:

- **Lot Size** : Too many NA values to be used. This was the case for the lot size, that presented >30% of missing data.
- **Year Built**: In theory the year of construction of the house is relevant to determine the qualiy of the construction, and might be related to price. That field had 8% of empty values. The most likely
- **School Rating** The missing value in School rating was due to a failure in the API call for the details. I just repeated the call and fixed it manually.
- **County** Not used. Already contained in zipcode.

5.2 Modeing Issues

- **Interactions between variables**: this model considers the variables as independent. Clearly there is a correlation between bath, rooms, and quared feet that are condider in the evaluation of the options. For example, when the house gets a new room, the evaluation considers the increase in the number of rooms and also the increase in the built surface. The interaction that is not captured here is the interaction between zip_codes and the other variables. Clearle a squared meter in one zip code is more valuable than in other. Introduce this interaction would require to include combined varibales such as zip x baths and zip x surface.

6. Conclusion

The listing price of houses depend strongly on the charateristics of the house itself, the postal_code, the rating of schools, but surprisingly not as strongly on the services surrounding them, the concentration of shops, or the closeness to recreation locations.

Other variables that showed up as weak was the neighborhood, or al least less informative that the zip_code. That doesn no't mean necesarily that it is not important but simply correlated to a other more relevant variable.

Another finding was that not all real state investemnt produces postive returs. Here the examples of the half bathroom is clearly returning less that the investment needed, and the bathroom built reusing space in the house barely makes the cut. Analyzing the coeficients we can see that many other combinations give negative retunrs, for example adding a second floor is not worth it unless you add enough surface and rooms to the building.

Clearly, investment in Real Estate depends strongly on the cost of labor, the additioal built surface added, and the rooms added.