MACHINE LEARNING 2017
FINAL PROJECT REPORT

Group Name: NTU_r0592106_武神與祂的子民

GROUP MEMBER:

R05921069 黃武昱

R05921095 曾旻

R05921088 向何鑫

R05921105 李育澤

PROJECT TOPIC

俄羅斯房地產

Sberbank Russian Housing Market - Can you predict realty price fluctuations in Russia's volatile economy?

PROJECT INTRODUCTION

俄羅斯房地產的比賽是由 Sberbank，俄羅斯最老的銀行，所主辦。其目的是希望透過大量過往房屋價格的資訊，以及俄羅斯經濟發展的數據，來預測俄羅斯房地產項目之價格的趨勢，以協助他們的客戶做更好的理財規劃。同時，銀行也希望透過這樣的方式，來增加客戶們進場買房的信心，這樣也能同時增加銀行的利潤。

雖說俄羅斯國內的房屋市場相對穩定，沒有過大的起伏波動，然而國家整體的經濟狀況變化卻極大，這也一定程度上使預測房屋市場的趨勢變得困難。要單純使用例如房間數量，地點等 features 來預測房屋價格走勢本來就已經不太容易，若再加上經濟因素的影響則就更困難。這也是這個比賽最大的難題所在。

這次比賽的 DataSet 就分為了 Housing Data 和 Macroeconomic patterns 兩個部分。前者包含了 train 和 test 兩個檔案，分別記錄了不同時期房屋於不同時期的各種資訊，包含售價、交易日期、總面積、建造年分等共 291 維的 features。至於後者，則是包含了油價、GDP、交易水準、CPI、PPI 等用以衡量經濟發展的 99 種 features。

PREPROCESSING / FEATURES ENGINEERING

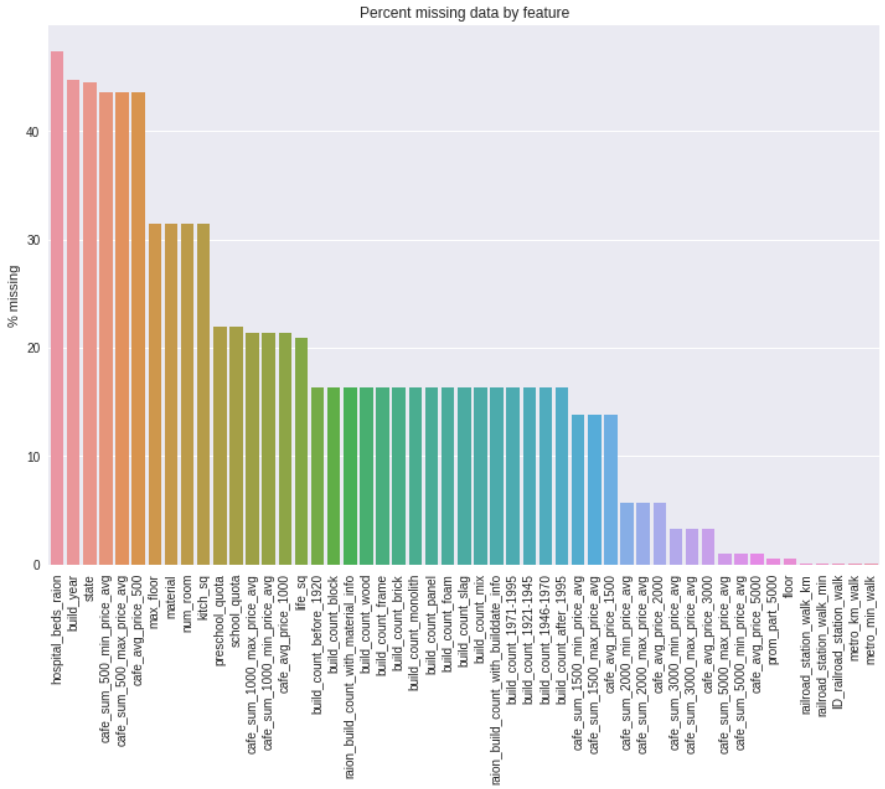
- ### Data Quality Issues

  Training Data 中不難發現有部分的 Data 明顯是誤值。例如 *state* 這個 feature 的值理應為 1~4 的整數，但卻有一筆資料的 state 值是 33。另外 *build_year* 這個 feature 的值理應就是年分，例如 2005、2006。但卻有一筆資料的值為 20052009，因此在此部分的 data preprocessing 就將這類的錯誤資料做調整處理。例如將 *state* 的資料以 mode 做取代，還有將 *build_year* 20052009 以 2007 來取代。

- ### Missing Data

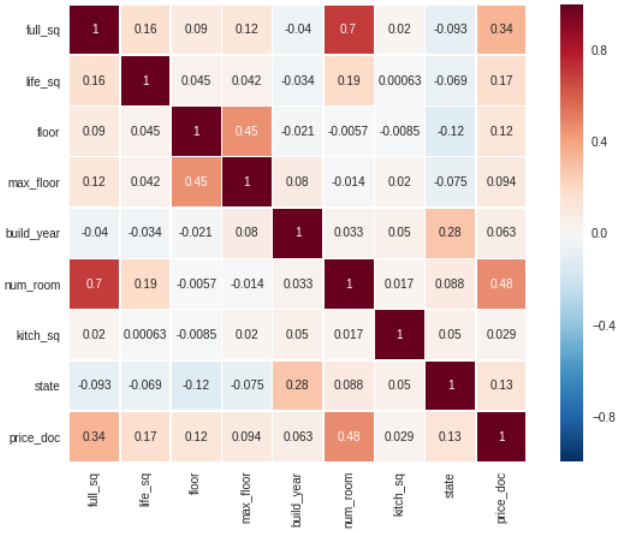
  Training Data 的檔案裏面有一部分的 Data 是 NAN，這些我們把它稱之為 Missing Data，再剛開始做 Data Processing 的時候，我們先 identify 那些 features 的 missing data 比較多，然後試圖將這些 missing data 做處理。另外，針對 missing data 比重最多的幾個 features，我們也有去測試將這些 missing data 做處理後代入值的結果與直接不使用這些

features 做 training 的結果進行比較分析。下圖則是我們將 Percentage of Missing Data of each feature 去做 visualization 後的結果：



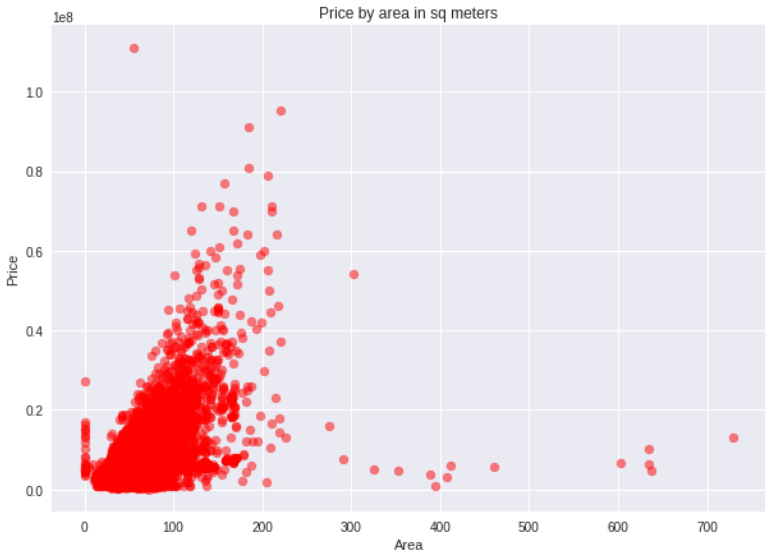
- Housing Internal Characteristics

這個部分的分析是將房屋內部的相關 feature 去做 heatmap，藉以觀察 feature 與 feature 之間的 correlation。這個方法在之前的作業也有實作過，因此就不詳細介紹。根據以下的 heatmap 的結果，我們有嘗試去除看似不好的 feature，例如 state：



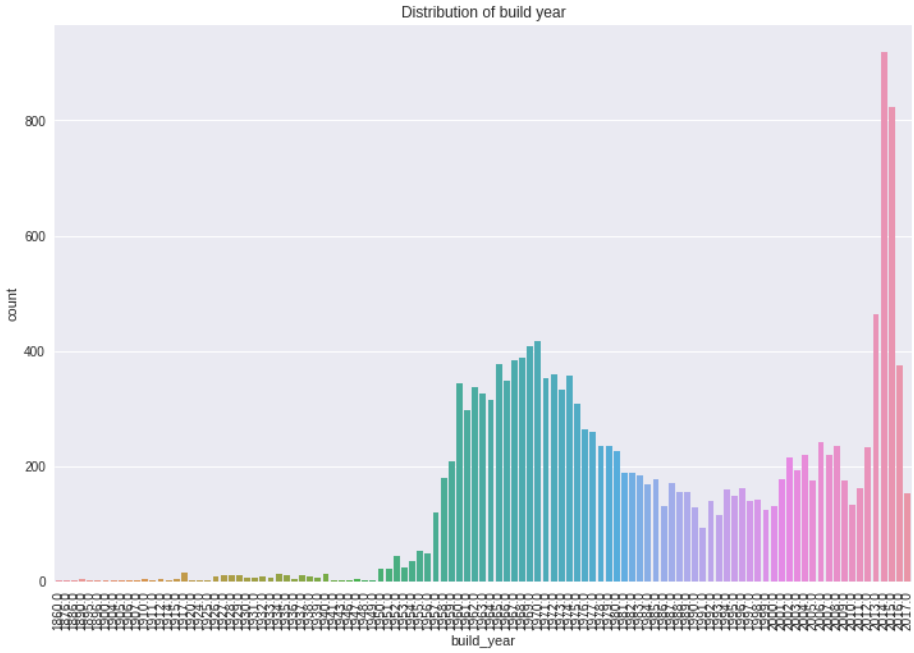
- Price by area in square meters

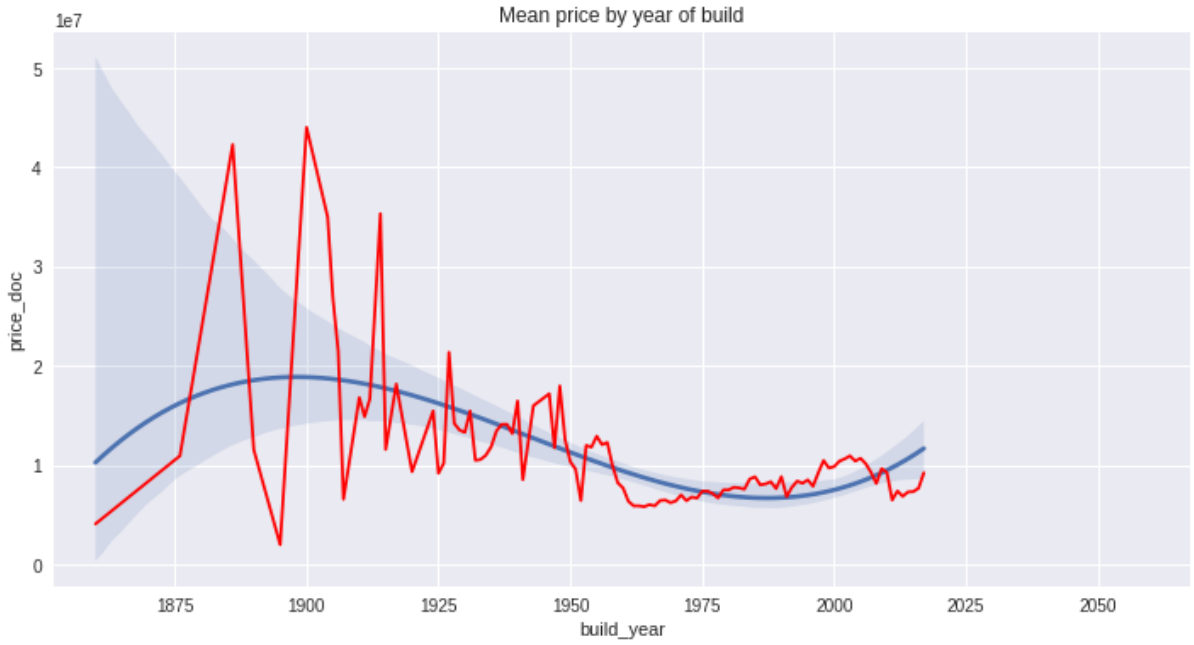
這個部分我們則是畫出了 Price by area in Square Meters 的分布狀況 (如下圖)。我們可以觀察到有部分的點是大幅的偏離主要分布區域，例如 Price 軸高於 1.0 與 Area 軸大於 700 的這兩筆資料，這些資料我們則視為極端資料，再我們的 training 中會將這些資料去除。



- Build Year

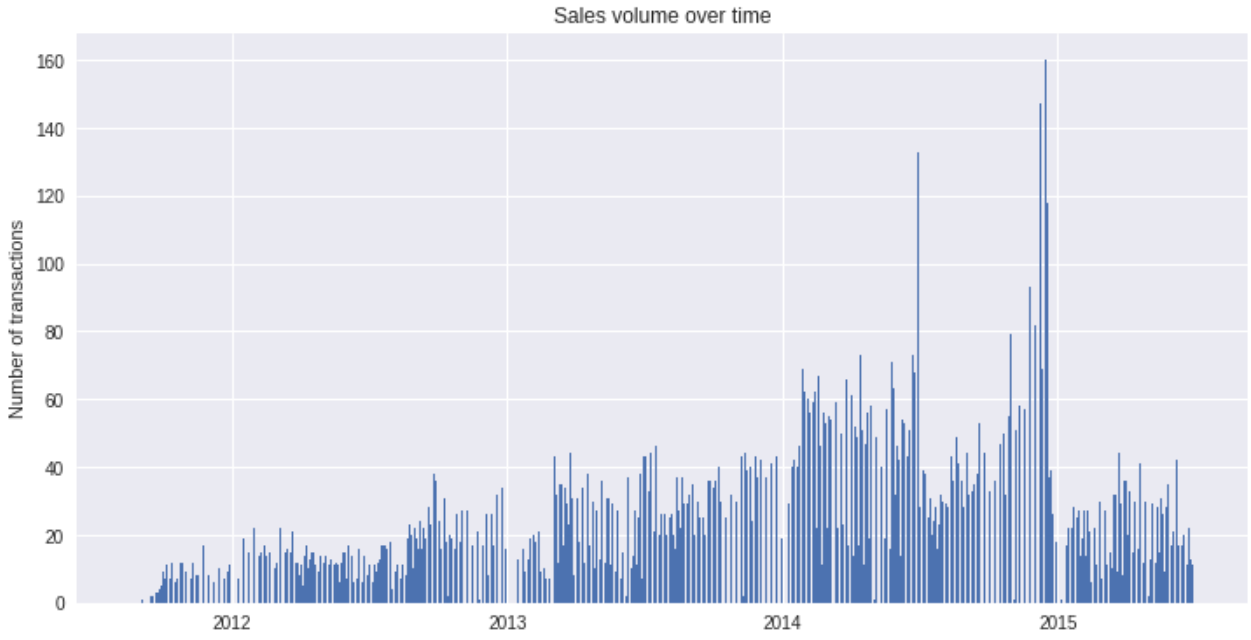
這個部分我們根據 Build Year 來觀察資料，其中包括了 Build Year 的分布狀況(下圖一)，以及畫出 mean price by year of build 的圖(下圖二)來做分析。這樣的分析主要的作用是用來測試後面根據年份而將不同資料去加權/除權的測試。





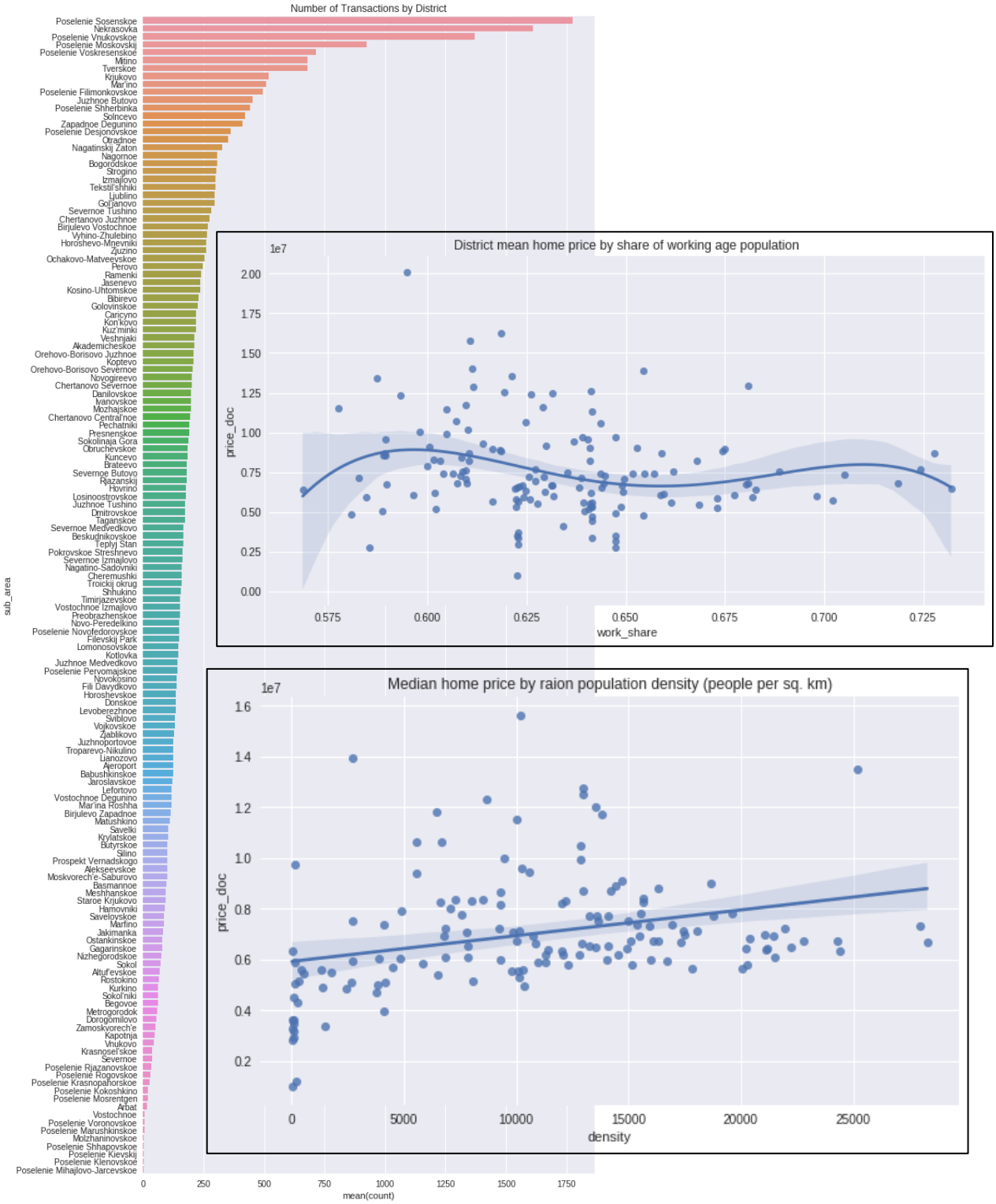
- Timestamp

這個部分是根據 timestamp (date of transaction)這個 feature 來分析資料。當資料被 visualize 後可以發現在某些年份的成交數量特別的大。這樣一來就可以知道大約再哪些時候的成交資料可信度比較高，哪些時候的參考價值比較低等資訊，再針對資料做加權/除權的處理。



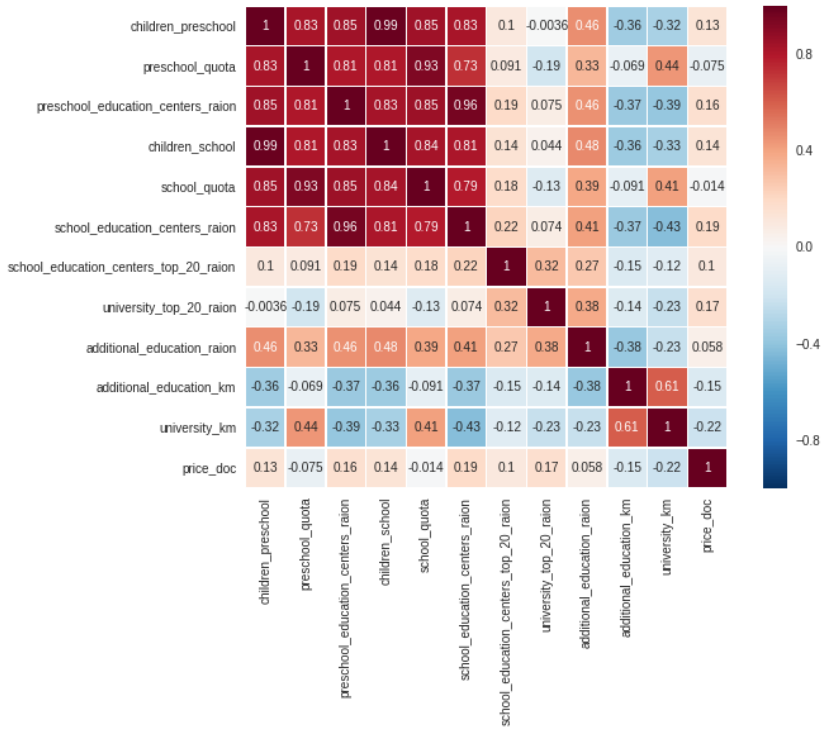
- Demographic Information

這個部分我們透過觀察區域人口分布，成交量分布，以及成交價格三個資訊來進行分析，以觀察這些 feature 的可信度。



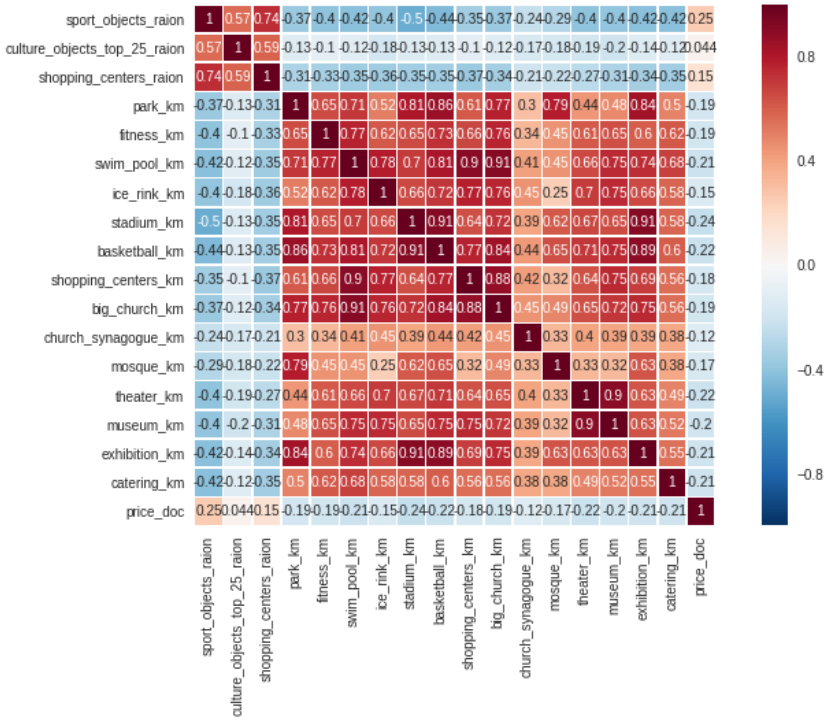
- School Characteristics

此部分為觀察與學校有關的 feature 之間的關聯度。



- Cultural / Recreational Characteristics

此部分則為觀察一些公共建設相關的 feature 之間的關聯度。



- Variable Importance

這個部分利用了 sklearn 裡面的套件，也透過做 label encoding，來做 feature importance 的分析。

```
## rf variable importance
## full_sq                                5.87095874356e-07
## life_sq                                9.04347628546e-08
## floor                                  1.20590421994e-08
## max_floor                              1.20794309636e-07
## material                               7.64088471083e-08
## build_year                             5.56703735125e-05
## num_room                               1.18520322785e-06
## kitch_sq                               5.87408823402e-08
## state                                  1.4491456575e-07
## product_type                           2.21240486991e-08
## sub_area                               5.77751434752e-09
## area_m                                 6.22619361625e-07
## raion_popul                            2.39214582143e-08
## green_zone_part                        2.57185801048e-07
## indust_part                            5.86329500469e-08
## children_preschool                     4.39636702546e-08
## preschool_quota                        3.2159141789e-05
## preschool_education_centers_raion      1.74500515456e-05
## children_school                        1.75531137303e-07
## school_quota                           2.0742244049e-07
```

MODEL DESCRIPTION 1 (DNN)

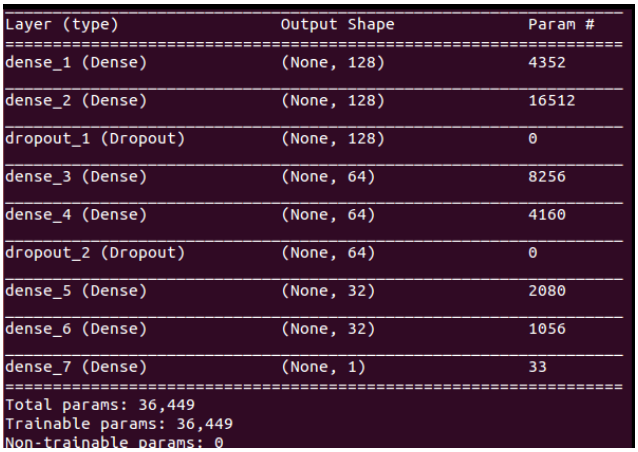
根據上一部分的資料分析，我們預先處理 train data 的部分，只有提取我們所需要的資料如下，在 train 和 test 的部分是一樣的：

```
X_list_num = ['full_sq', 'num_room', 'floor', 'timestamp',
              'preschool_education_centers_raion', 'school_education_centers_raion',
              'children_preschool', 'children_school',
              'shopping_centers_raion', 'healthcare_centers_raion',
              'office_raion', 'sport_objects_raion',
              'metro_min_walk', 'public_transport_station_min_walk',
              'railroad_station_walk_min', 'cafe_count_500',
              'kremlin_km', 'workplaces_km', 'ID_metro',
              'public_healthcare_km', 'kindergarten_km', 'school_km', 'university_km',
              'museum_km', 'fitness_km', 'park_km', 'shopping_centers_km',
              'additional_education_km', 'theater_km',
              'raion_popul', 'work_all', 'young_all', 'ekder_all']
```

然後我們有把 num_room 的 nan 部分，用 pandas 的 dropna 去處理，metro_min_walk 和 railroad_station_walk_min 的 nan 部分則是使用 interpolate 的 linear 方法處理，最後 floor 的 nan 部分是使用 fillna，方法是 floor 的 median。

以下是我們 DNN 的架構:

```
model.add(Dense(128, input_dim=33, kernel_initializer='normal', activation='relu'))
model.add(Dense(128, kernel_initializer='normal', activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(64, kernel_initializer='normal', activation='relu'))
model.add(Dense(64, kernel_initializer='normal', activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(32, kernel_initializer='normal', activation='relu'))
model.add(Dense(32, kernel_initializer='normal', activation='relu'))

model.add(Dense(1, kernel_initializer='normal'))

model.summary()

model.compile(loss='mse', optimizer='rmsprop', metrics=['mae'])
```



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	4352
dense_2 (Dense)	(None, 128)	16512
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
dense_4 (Dense)	(None, 64)	4160
dropout_2 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 32)	2080
dense_6 (Dense)	(None, 32)	1056
dense_7 (Dense)	(None, 1)	33

Total params: 36,449
Trainable params: 36,449
Non-trainable params: 0

用上述這個架構所預測出來的分數是

result_DNNmodel1_1.csv 0.35740
a minute ago by r05921105_cody
DNN1

結果並不是很好，所以我們在嘗試使用另一種 model 的架構。

MODEL DESCRIPTION 2 (BaggingRegressor)

由於使用 DNN 的結果不好，我們決定改採 decision tree 類型的方法，但由於基礎的 RandomForest 方法有 overfitting 的問題，因此我們使用了 sklearn 套件裡面的其中一個方法 BaggingRegressor 來嘗試，這部分我們輸入的 data 跟 DNN 是一樣的。

所預測出來的分數是

result_Bagging_1.csv 0.35686
a minute ago by r05921105_cody
Bagging

結果比 DNN 來的好一些，但還是蠻差的。

MODEL DESCRIPTION 3 (XGB)

在 XGBoost 的方法中，我們也使用三個不同 model 分別進行預測後，將結果 merge and adjust 之後，得到最後的結果。第一個 model 叫 jason model，是參考再 kaggle kernal 上一位名叫 jason 的人所寫的 model。這個 model 的特色是，在 train 之前，會先做 clean data 的動作，將 training data 中的 NAN 用別的值替換掉，而 data clean 的部分內容，也是根據類似於我們所做的 Data preprocess 與分析的結果來進行，下圖為部分的 data clean:

```
#clean data
bad_index = train[train.life_sq > train.full_sq].index
train.ix[bad_index, "life_sq"] = np.NaN
equal_index = [601,1896,2791]
test.ix[equal_index, "life_sq"] = test.ix[equal_index, "full_sq"]
bad_index = test[test.life_sq > test.full_sq].index
test.ix[bad_index, "life_sq"] = np.NaN
bad_index = train[train.life_sq < 5].index
train.ix[bad_index, "life_sq"] = np.NaN
bad_index = test[test.life_sq < 5].index
test.ix[bad_index, "life_sq"] = np.NaN
bad_index = train[train.full_sq < 5].index
train.ix[bad_index, "full_sq"] = np.NaN
bad_index = test[test.full_sq < 5].index
test.ix[bad_index, "full_sq"] = np.NaN
kitch_is_build_year = [13117]
```

同時我們也根據前面部分的 preprocess 分析來去除一些極端的資料。

```
# brings error down a lot by removing extreme price per sqm
train.loc[train.full_sq == 0, 'full_sq'] = 50
train = train[train.price_doc/train.full_sq <= 600000]
train = train[train.price_doc/train.full_sq >= 10000]
```

另外，此方法也加入了一些新的 feature，例如年份、月份、周的 data 進行 training，並針對經濟狀況對每年的房價進行除權的微調。

第二個 model 叫做 Reynaldo's model，顧名思義就是一個名叫 Reynaldo 的 kaggler 所設計的。這個 model 只有在 train price 的部分做了參數微調。

第三個 model 叫做 Bruno's model，也就是一個名叫 Bruno 的 kaggler 所設計的。這個 model 的特色在於，他加入了 macro data (macro.csv)來進行 training。同時它也有類似於 Jason model 中的做法，加入年份、月份、周的 data 進行 training。

三個方法的 xgb training 參數也有所不同，差別在於 learning rate，iteration、tree depth、subsample ratio 等參數。

最後則是將這三個方法的結果 merge 起來。Merge 的方式會在下一個部分 Experiments 中詳述。

EXPERIMENTS AND DISCUSSION

這次 project 中實驗的部分主要就是在於測試不同的方法與調整參數。我們之所以會選擇上一部分所描述的三種方法皆有其原因。

會選擇第一種方法是因為就觀察 training data 我們大概可以得出這一次的 training 需要用到 linear regression 的結論。於是第一個想到的當然是曾經在作業中使用過，利用 keras 的 model 實作 linear regression。但由於這個方法的成效明顯不彰，於是我們便使用了第二個我們熟悉的方法 sklearn。

Sklearn 方法中使用的套件在 model description 已經有描述過，會選擇這個套件是因為從作業經驗還有 sklearn 的 document 內容，判斷認為 bagging regressor 可能會是處理這次比賽資料的一個準確且有效的方法。而且經過了嘗試也發現 sklearn 中其它 regressor (例如 gradient boosting regressor)的效果比 bagging 更差。

但由於第二個方法依舊沒有顯著的成效，於是我們從網路上(kaggle 的 kernal)中找到了第三個方法，xgboost。這個方法的 model 在上面的部分也已經有細述，這邊主要分享從一開始使用 xgboost 到最後成果中間所做的實驗。

一開始我們只有使用一個 model，並且利用了我們自己所做的 data preprocess 來 train。唯一的變化就只是調整 training 的參數。這樣的效果也沒有非常好，從下面的圖我們可以看到調整參數後的結果相差不大：

XgbFollow4.csv 22 days ago by r05921088_whyalwaysme add submission details	0.31284	☐
XgbFollow4.csv 22 days ago by r05921088_whyalwaysme add submission details	0.31285	☐
XgbFollow.csv 23 days ago by r05921088_whyalwaysme add submission details	0.31119	☐
Sub_feat0.31313.csv a month ago by r05921088_whyalwaysme add submission details	0.31337	☐

後來我們試著加入新 feature 來做 training，如下圖：

```
test['area_per_room'] = test['life_sq'] / test['num_room'].astype(float)
test['livArea_ratio'] = test['life_sq'] / test['full_sq'].astype(float)
test['yrs_old'] = 2017 - test['build_year'].astype(float)
test['avgfloor_sq'] = test['life_sq']/test['max_floor'].astype(float) #living area per floor
test['pts_floor_ratio'] = test['public_transport_station_km']/test['max_floor'].astype(float) #apartments
near public t?
test['room_size'] = test['life_sq'] / test['num_room'].astype(float)
test['gender_ratio'] = test['male_f']/test['female_f'].astype(float)
test['kg_park_ratio'] = test['kindergarten_km']/test['park_km'].astype(float)
test['high_ed_extent'] = test['school_km'] / test['kindergarten_km']
test['pts_x_state'] = test['public_transport_station_km'] * test['state'].astype(float) #public trans *
state of listing
test['lifesq_x_state'] = test['life_sq'] * test['state'].astype(float)
test['floor_x_state'] = test['floor'] * test['state'].astype(float)
```

增加這些 feature 後 training 的效果也沒有明顯的改善，但至少足夠我們過 simple baseline。

後來，就是我們在網路上參考到有人利用三個 model 來做單純的加權平均的方法。這樣的方法讓結果有明顯的變好。大約到 0.3109 附近的分數。

最後，我們嘗試使用 macro 的資料來針對 result 做調整。我們先通過 marco data 去計算一個係數。然後計算 Macro data monthly medians，Price data monthly medians，接著把 monthly prices 和 macro data 合在一起。接下來的步驟就是，Prepare data for macro model；Predict with macro model；Merge with test cases and compute mean for macro prediction。由於 Naive macro model 假設了 housing prices will simply follow CPI，於是我們可以利用 Combine naive and substantive macro models 來得到的 macro_mean 取 log 函數，最後代入三個 model 的預測值，加權後得到最後的結果。

使用這樣的方法可以得到我們在 kaggle 上最好的成績：

WORK DISTRIBUTION

Team Members	黃武昱	曾旻	向何鑫	李育澤
資料分析	✓	✓	✓	✓
DNN	✓	✓		✓
XGB		✓	✓	
Presentation		✓	✓	✓
Report	✓	✓	✓	✓

Reference

[1] XGBoost Github, https://github.com/dmlc/xgboost

[2] XGBoost Documents, https://xgboost.readthedocs.io/en/latest/

[3] Kaggle Kernals, https://www.kaggle.com/c/sberbank-russian-housing-market/kernels

[4] T. Chen and C. Guestrin, “XGBoost: Reliable Large-scale Tree Boosting System”, 2015.

[5] J. H. Friedman, “Greedy Function Approximatation: A Gradient Boosting Machine”, IMS 1999.