Springboard Data Science Capstone Project I

Analysis of house prices in London 1995-2019

Akiro Semtei

April, 2020

Table of Contents

1. Introduction

The Housing Market in the UK is known for its volatility. Now, as people are confined to their homes, the property chain in many cases has been disrupted as all sales and moves are on hold.

Pre-Coronavirus Economy Housing Market

The housing market was in a fairly good position before Covid-19 called for a frozen market. The head of research at estate agent Knight Frank stated that the housing market was in a "strong position" for the first months of 2020, noticing a general trend of price increases and sales across the UK. For properties in central London, the market was beginning to experience a reversal of the decline caused by Brexit uncertainty. More houses were being built and round 70,000 mortgage approvals were made in February alone, which was the highest monthly figure in six years.

However, due to the wide-scale economic burdens being faced, the housing market must prepare for weaker economic activity than expected. Zoopla has warned that transaction volumes could fall by as much as 80% during the Spring compared to 2019 figures. Now, due to the lockdown, thousands of house moves are on hold with many transactions falling through completely, multiple delayed transactions, and many individuals left with legal costs. Though the situation seems bleak, experts are positive that this slump will be

short-lived due to the finite nature of the crisis. The key objective of this project was to predict house prices in London for any postcode with data from 1995-2019.

2. Data Acquisition and Cleaning

The data set was acquired from various open sources. The first set was retrieved from the UK government HM Land Registry. The dataset dubbed 'Price Paid Data' or PPD includes information on all property sales in England and Wales that are sold for value and are lodged with HRMC for registration; the data contains HM Land Registry data ©Crown copyright and database right 2020. This data is licensed under the Open Government License v3.0.

The second and third datasets were retrieved from the Office for National Statistics, also a part of the UK government. The second file contains the National Statistics Postcode Lookup (NSPL) for the United Kingdom as of February 2019 in Comma Separated Variable (CSV). The NSPL relates both current and terminated postcodes to a range of current statutory geographies via 'best-fit' allocation from the 2011 Census Output Areas. The third and final file contains the digital vector boundaries for Local Authority Districts in the United Kingdom as of 1 April 2019. These last two sets contains Royal Mail, Gridlink, LPS (Northern Ireland), Ordnance Survey and ONS Intellectual Property Rights. They are to be used under the same Open Government License v3.0.

Price Paid Data (PPD)

The first dataset contains all house transactions registered with HRMC across the entirety of the United Kingdom since January 1st, 1995. We thus can assume that most, if not all transactions in real estate have been recorded in the studied dataset. PPD contains 25 million transactions with data stored across 16 columns as outlined below:

Column	Data item	Explanation (where appropriate)
1	Transaction unique identifier	A reference number which is generated automatically recording each published sale. The number is unique and will change each time a sale is recorded.
2	Price	Sale price stated on the transfer deed.
3	Date of Transfer	Date when the sale was completed, as stated on the transfer deed.
4	Postcode	This is the postcode used at the time of the original transaction.
5	Property Type	D = Detached, S = Semi-Detached, T = Terraced, F = Flats/Maisonettes, O = Other Note that: 'Other' is only valid where the transaction relates to a property type that is not covered by existing values; meaning all commercial and office properties fall under this description
6	Old/New	Indicates the age of the property and applies to all price paid transactions, residential and non-residential. Y = a newly built property, N = an established residential building
7	Duration	Relates to the tenure: F = Freehold, L= Leasehold etc.

		Note that HM Land Registry does not record leases of 7 years or less in the Price Paid Dataset.
8	PAON	Primary Addressable Object Name. Typically the house number or name.
9	SAON	Secondary Addressable Object Name. Where a property has been divided into separate units (for example, flats), the PAON (above) will identify the building and a SAON will be specified that identifies the separate unit/flat.
10	Street	
11	Locality	
12	Town/City	
13	District	
14	County	
15	PPDCategory Type	Indicates the type of Price Paid transaction. A = Standard Price Paid entry, includes single residential property sold for value. B = Additional Price Paid entry including transfers under a power of sale/repossessions, buy-to-lets (where they can be identified by a Mortgage) and transfers to non-private individuals. Note that category B does not separately identify the transaction types stated. HM Land Registry has been collecting information on Category A transactions from January 1995. Category B transactions were identified from October 2013.
16	Record Status - monthly file only	Indicates additions, changes and deletions to the records. A = Addition C = Change D = Delete.

The set is far from clean and ready to use in the study. The columns are not completely populated as evidenced by a null analysis. There are 22 million missing points in #9 SAON and 8million in #11 Locality amongst others.

We reduce the columns of no interest or those that are repeated in the second dataset and drop the NAs of those unknown postcodes as seen in figure 2.

```
df0.sina().sum()
```

```
Transaction unique identifier              0
Price                                      0
Date of Transfer                           0
Postcode                               38268
Property Type                              0
Old/New                                    0
Duration                                   0
PAON                                    4206
SAON                                22104100
Street                                384856
Locality                             8065357
Town/City                                  0
District                                 186
County                                     0
PPDCategory Type                           0
Record Status - monthly file only          0
dtype: int64
```

Figure 1 - Initial columns NAs

We proceed to filter out all transactions of Property Type = 'O'. These properties cover all commercial sales and are of no interest for this study. Similarly, PPDCategory Type = 'B' is of no interest to us as it would skew our results. These transactions consider deals that are not strictly private residential purchases. After the cleaning we retain 24.2 million observations.

```
df0.isna().sum()
```

```
Price                   0
Date of Transfer        0
Postcode            38268
Property Type           0
Old/New                 0
Duration                0
Town/City               0
PPDCategory Type        0
dtype: int64
```

Figure 3 - Final columns NAs

```
df0 = df0[df0['PPDCategory Type']=='A']
df0 = df0[df0['Property Type']!='O']
```

Figure 2 - Cleaning unrelated transactions

National Statistics Postcode Lookup (NSPL)

The second dataset has information on the postcodes for all residences in the UK. It comes with a host of information that is interesting to our study. There are 2.6 million postcodes in the UK, thus we have the same amount of observations. The data has 46 columns that enrich our study space, detailed below:

1	Postcode	16	Introduced	31	MSOA
2	In Use?	17	Terminated	32	Middle
3	Latitude	18	Parish	33	Parish
4	Longitude	19	National Park	34	Census
5	Easting	20	Population	35	Constituency
6	Northing	21	Households	36	Index
7	Grid Ref	22	Built	37	Quality
8	County	23	Built	38	User
9	District	24	Lower	39	Last
10	Ward	25	Rural/urban	40	Nearest
11	District Code	26	Region	41	Distance
12	Ward Code	27	Altitude	42	Postcode
13	Country	28	London	43	Postcode
14	County	29	LSOA	44	Police
15	Constituency	30	Local	45	Water
				46	Plus

There are many issues with this dataset that make it far from useful in its current state, columns like #9 National Park which is missing in over 95% of observations, similarly #20 Population; a column that could be of real interest for the analysis, has over 1.1 million missing values. We have to drop several columns due to the NAs leaving us with the ones shown in figure 4.

```
pc.isnull().sum()
```

```
Postcode                0
Latitude                0
Longitude               0
District             9961
Ward                 9961
District Code        9961
Ward Code            9961
Altitude             9896
LSOA Code            9961
MSOA Code            9961
Nearest station     69173
Distance to station 69173
Plus Code            9961
dtype: int64
```

```
pc = pc.dropna()
```

Figure 4 - Postcodes final columns

Local Authority Districts – Digital vector boundaries

The final dataset is a geojson that describes the Local Authority Districts of the UK overlaid on GPS coordinates. This data does not require cleaning for the purposes of this study.

3. Data Exploration

```
result = pd.merge(pc, df0, on='Postcode')
result.to_csv('result.csv', index=False)
```

Figure 5 - Merging PPD with NSPL over Postcodes

3.1 Merging and Outliers

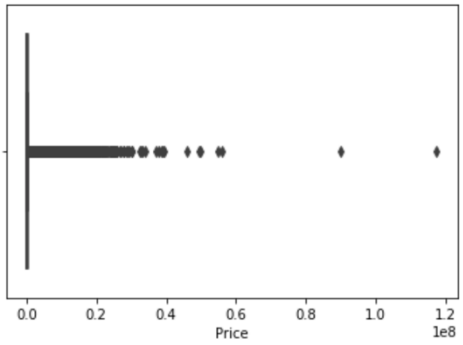


Figure 7 - Boxplot of cleaned data by Price

In order to run our experiment efficiently we want to join the Price Paid Data (PPD) with the National Statistics Postcode Lookup (NSPL) into a single data frame via merge on the 'Postcodes' column. We now have 24.2 million transactions rich in data over 17 feature columns and not a single NA. Observing figure 6, a seaborn boxplot confirms huge outliers in the expensive end of the prices. However a simple interquartile range filter would not suffice as this data is indeed a timeseries. We proceed to order the data by year and then apply an interquartile filter. We remove the outliers because some transactions observed are out of line with market expectations with sums as low as £ 1 and as high as £ 120 million.

```
yearly = {}
for i in range(1995,2020):
    yearly[i] = (result[result.year == i][result.Price>50000])
    Q1 = yearly[i].loc[:, ('Price')].quantile(0.25)
    Q3 = yearly[i].loc[:, ('Price')].quantile(0.75)
    IQR = Q3 - Q1
    yearly[i] = yearly[i].loc[(yearly[i]['Price'] > (Q3 - 1.5 * IQR)) & (yearly[i]['Price'] < (Q3 + 1.5 * IQR))]
```

Figure 6 - IQR outlier cleaning

A standard interquartile range of (0.25-0.75) proved to cut way too many data points; the most expensive property in the data would be reduced to just £625k. We decided to apply a modified version, as seen in figure 7, that would keep more values by eliminating all properties under £50k. London properties in 2019 would hardly be affected by any properties purchased under this price thus it is safe to eliminate these values without introducing significant bias. With this procedure we reduce the study space to all properties sold between 1995-2019 on the price range of £50k - £1.12mill. As such our study will not be applicable or useful for the super prime market in London but will return sufficiently good results to estimate the prices for over 90% of all properties.

Finally we filter by the 13 London boroughs to focus on our target market. We ran an analysis with all UK points and the improvement in the results was less than 5% at a computing cost of over 5,000% in runtime. We decided to keep our focus on the 13 boroughs.

```
boroughs = ['Camden',
'Greenwich',
'Hackney',
'Hammersmith and Fulham',
'Islington',
'City of London',
'Kensington and Chelsea',
'Lambeth',
'Lewisham',
'Southwark',
'Tower Hamlets',
'Wandsworth',
'Westminster',
'Barking and Dagenham',
'Barnet',
'Bexley',
'Brent',
'Bromley',
'Croydon',
'Ealing',
'Enfield',
'Haringey',
'Harrow',
'Havering',
'Hillingdon',
'Hounslow',
'Kingston upon Thames',
'Merton',
'Newham',
'Redbridge',
'Richmond upon Thames',
'Sutton',
'Waltham Forest']
```

Figure 8 - London Boroughs

3.2 Understanding Target Variable

The total number of samples is 2.75 million transactions in the 13 London boroughs. There are no classes and the target variable of continuous nature. We are facing a complex regression problem, we will proceed to test XGBoost and Random Forest Regressor as algorithms with target variable both as 'Price' and the logn of 'Price', cleverly named 'logPrice'

3.3 Visual analysis

Our final data set for modeling is saved as 'cleaned-all.csv', info() on it is shown to the right in figure 9.

```
cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2750409 entries, 0 to 2750485
Data columns (total 19 columns):
Postcode                object
Latitude                float64
Longitude               float64
District                object
Ward                    object
District Code           object
Ward Code               object
Altitude                float64
LSOA Code               object
MSOA Code               object
Nearest station         object
Distance to station     float64
Price                   int64
Date of Transfer        datetime64[ns]
Property Type           object
Old/New                 object
Duration                object
year                    int64
logPrice                float64
dtypes: datetime64[ns](1), float64(5), int64(2), object(11)
memory usage: 419.7+ MB
```

```
cleaned.to_csv('cleaned-all.csv', index=False)
```

Figure 9 - Cleaned dataset

This dataset is organized in an efficient way to run algorithms but require some massaging to show insightful plots that help us understand the information in front of us.

Using GeoPandas and the json boundary file we created a set of interactive plots for each year's transactions. These plots show the difference in price for the different boroughs, the years 2000, 2010, and 2019 are shown in figures 10 and 11 as examples of the

illustrations that are possible with our cleaned data.

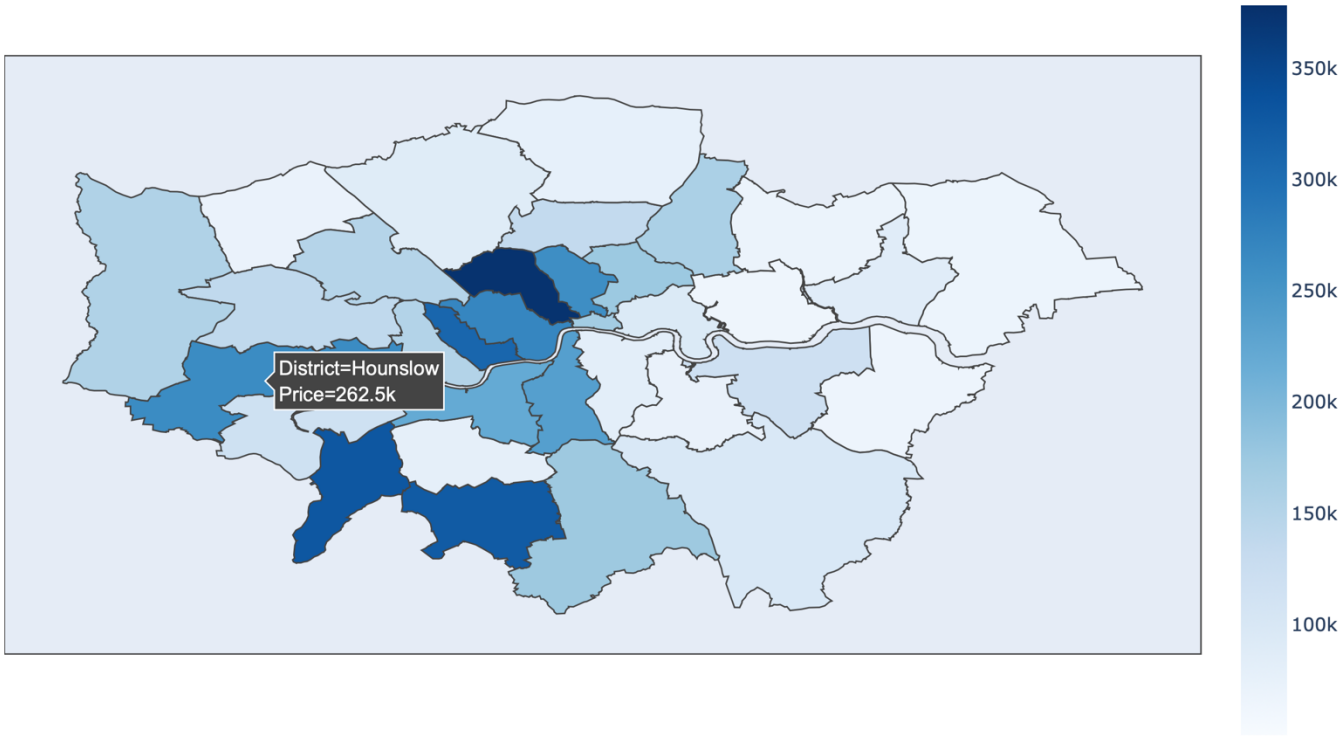


Figure 10 - London Borough Prices in 2000

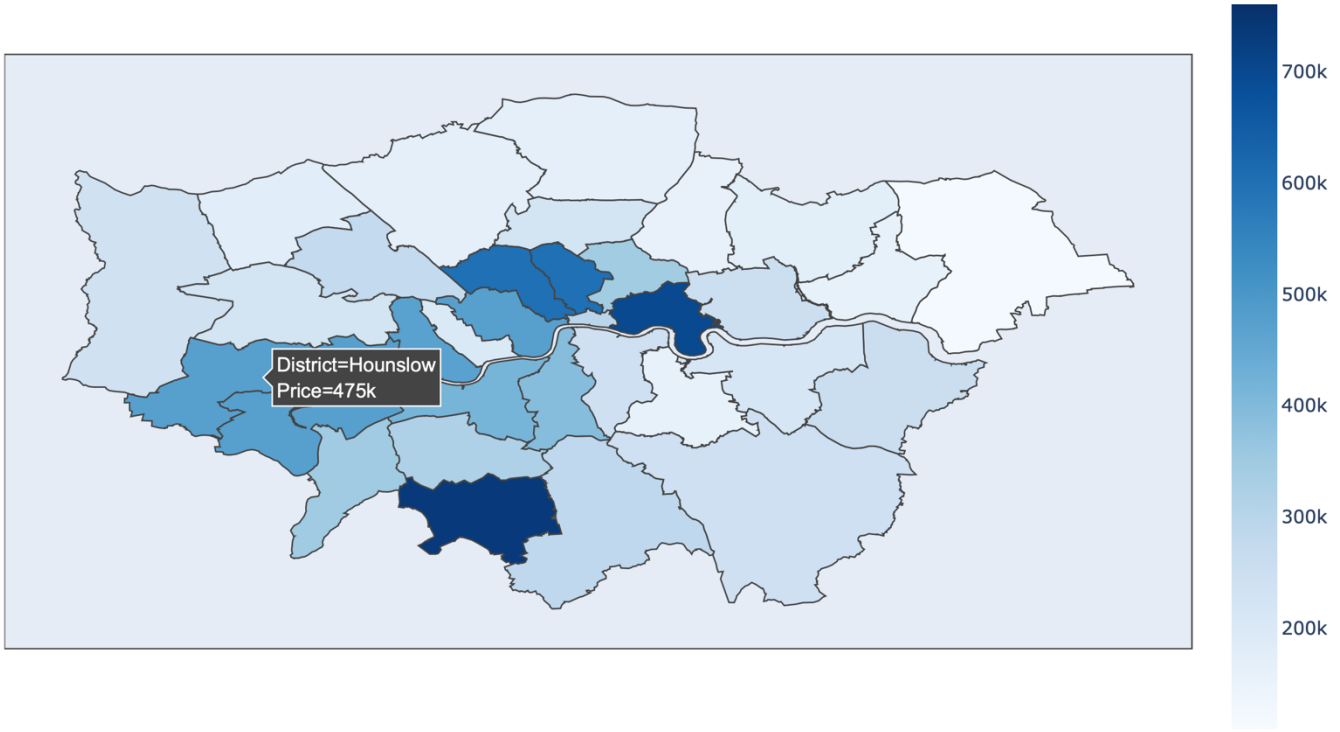


Figure 11 - London Borough Prices in 2010

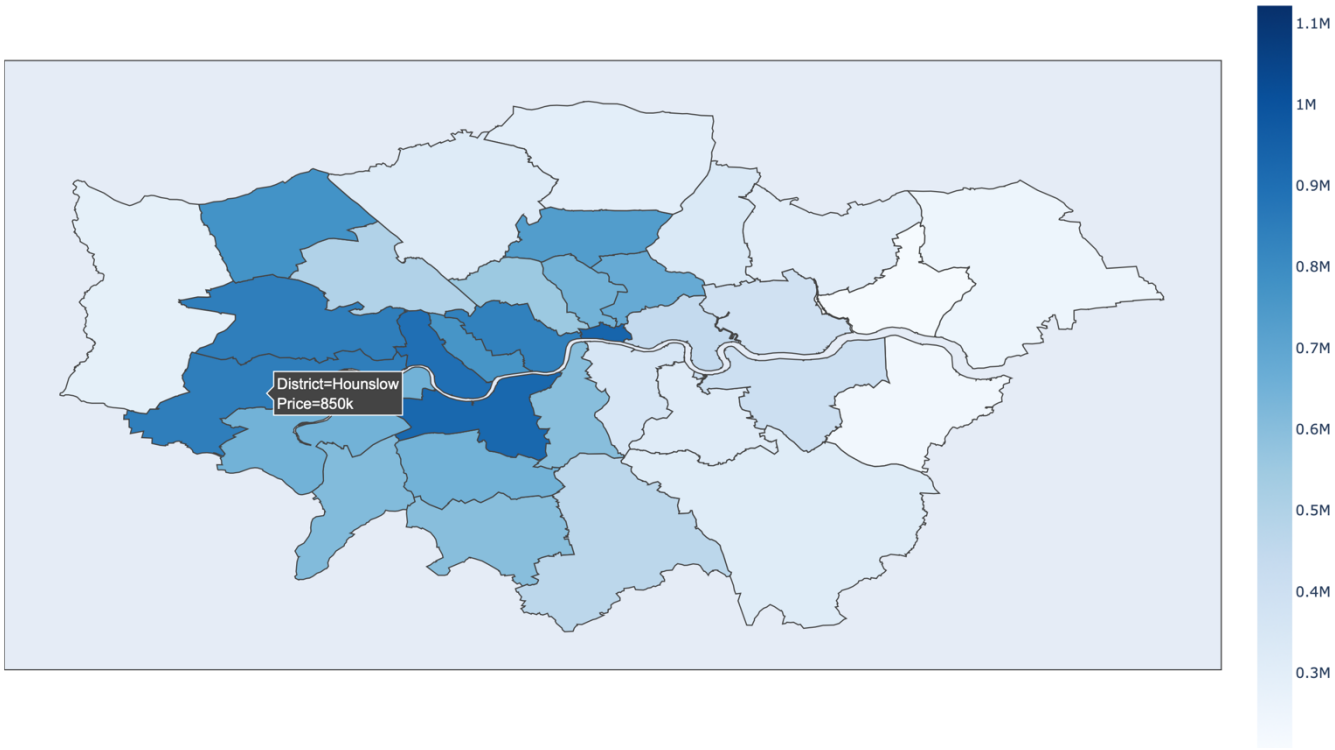


Figure 12 - London Borough Prices in 2019

We can clearly observe how the different boroughs concentrate the attention of the market through time. From the figures above we can deduce that the highest prices for residential properties are slowly shifting west while other boroughs in the east lagging behind.

We proceed to show the mean prices for each district across time in figure 13. From these figures we can observe that some districts had a much bigger impact from the 2008 financial crises while others remained relatively unscathed. However, every district in London seems to have overcome the crisis and are normalizing back to a linear regression.

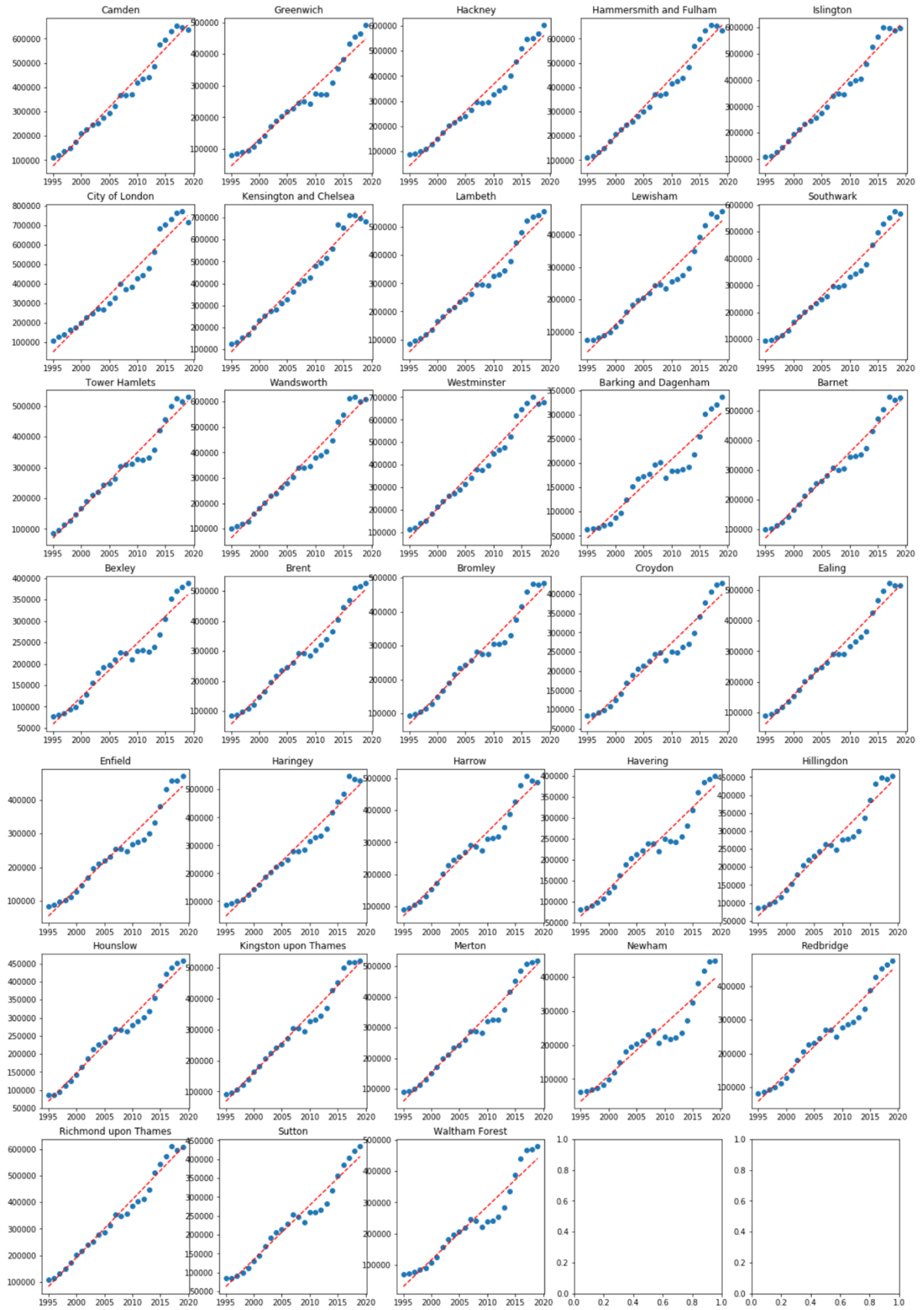


Figure 13 - London Boroughs mean prices per year

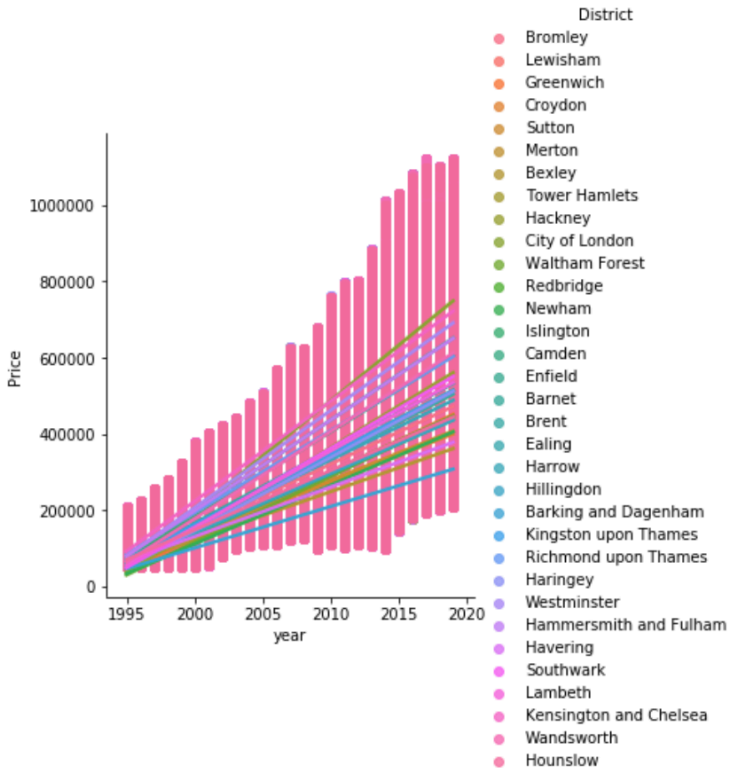


Figure 14 shows us a simple linear regression for the mean prices per year for all boroughs with the interval of values in the background

Figure 14 - Stacked regression plot

Figure 15 shows us the mean price growth per year per district

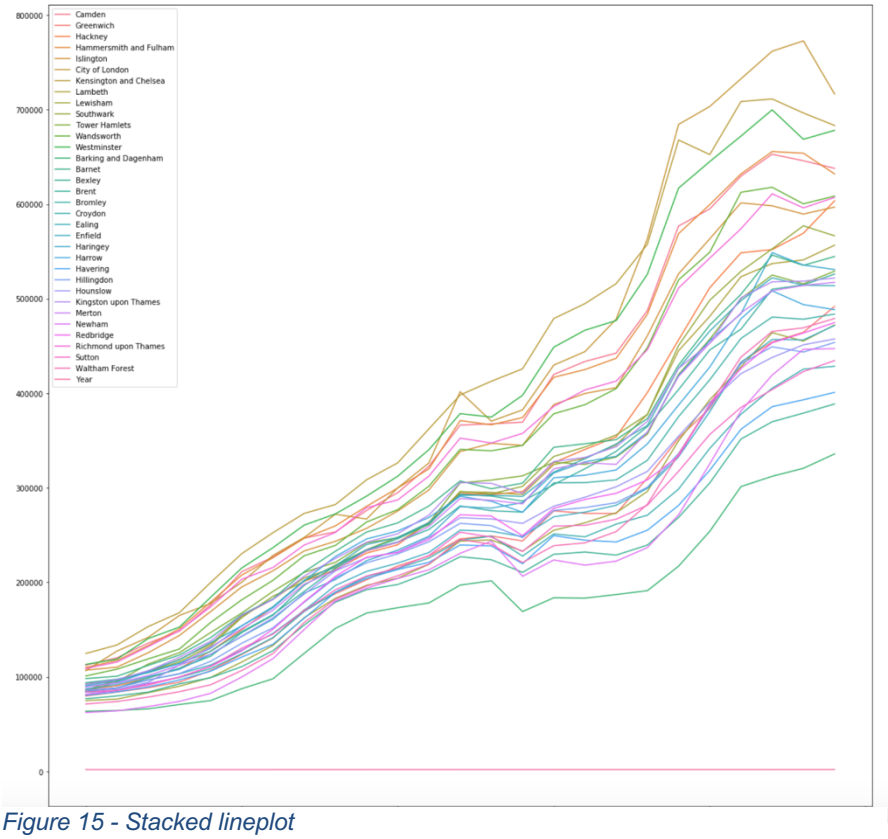


Figure 15 - Stacked lineplot

We clearly can see that prices and locations are correlated in some form as different districts behave differently through time in our data. Our problem may have a solution that fits and can model the pricing for London residences in a covid-free forecast.

4. Modelling

4.1 Data Pre-processing

4.1.1 Scaling:

Feature scaling in machine learning is one of the most important steps during preprocessing of data. Regression models learn a mapping from a set of features to an output variable. As such, the scale and distribution of the data drawn from the domain may be different for each variable. Differences in the scales across input variables may increase the difficulty of the problem being modeled. A model with large weight values is often unstable, meaning that it may suffer from poor performance during. We proceed to scale all values of Latitude, Distance to station, Altitude, and Longitude to (0,1)

4.1.2 Encoding:

Encoding is important to apply when you have categorical features that are better understood as binary variables. One Hot Encoding helps us here by creating a series of dummy variables. We need to encode the feature columns 'Property Type', 'Old/New', and 'Duration'.

4.1.3 XGBoost

We start with a hyperparameter tuning followed by fitting the data to an estimator as seen in figure 16. The sample is too large, and the computing resources required increase exponentially, we have to tune the parameters with 5% of the data. Fitting the estimator is repeated with a new target variable, 'logPrice' to compare results, seen in figure 17.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_state=42)
```

```
gsc = GridSearchCV(estimator=xgboost.XGBRegressor(),
        param_grid={
        'min_child_weight': [12,13,14],
        'gamma': [0.09,0.11,0.12],
        'max_depth': [11,12,13]
        },
        cv=5, scoring='neg_root_mean_squared_error', verbose=3,n_jobs=-1)
```

```
grid_result = gsc.fit(X_train, y_train)
best_params = grid_result.best_params_
print(best_params)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  96 tasks      | elapsed:  8.6min
[Parallel(n_jobs=-1)]: Done 135 out of 135 | elapsed: 12.2min finished
```

```
{'gamma': 0.09, 'max_depth': 12, 'min_child_weight': 13}
```

{'gamma': 0.09, 'max_depth': 12, 'min_child_weight': 13}

Figure 16 - GridSearchCV for hyperparameter tuning

```
%%time
xgb = xgboost.XGBRegressor(random_state =42,objective='reg:squarederror',
        min_child_weight=13,
        gamma=0.09,
        subsample=1,
        colsample_bytree=0.8,
        max_depth=12  ,n_jobs=-1
                        )
xgb.fit(X_train, y_train)
```

```
Wall time: 1min 27s
```

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0.09, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=12,
             min_child_weight=13, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=-1, num_parallel_tree=1,
             objective='reg:squarederror', random_state=42, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

```
%%time
xgblog = xgboost.XGBRegressor(random_state =42,objective='reg:squarederror',
        min_child_weight=13,
        gamma=0.09,
        subsample=1,
        colsample_bytree=0.8,
        max_depth=12  ,n_jobs=-1
                        )
xgblog.fit(Xlog_train, ylog_train)
```

```
Wall time: 1min 27s
```

```
XGBRegressor(base_score=0.5, booster=None, colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.8, gamma=0.09, gpu_id=-1,
             importance_type='gain', interaction_constraints=None,
             learning_rate=0.300000012, max_delta_step=0, max_depth=12,
             min_child_weight=13, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=-1, num_parallel_tree=1,
             objective='reg:squarederror', random_state=42, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None,
             validate_parameters=False, verbosity=None)
```

Figure 17 - XGBoost for Price and logPrice

4.1.4 Random Forest Regressor

Similarly, we tart with a hyperparameter tuning followed by fitting the data to an estimator as seen in figure 18. We also use 5% of the set to estimate the parameters. Fitting the estimator is repeated with a new target variable, 'logPrice' to compare results, seen in figure 19.

```
gsc2 = GridSearchCV(estimator=RandomForestRegressor(),
        param_grid={
    'n_estimators': (100,200),
    'min_samples_leaf': [1,2,5],
    'min_samples_split': [5,10,50]},
    cv=3, scoring='neg_root_mean_squared_error', verbose=3,n_jobs=-1)
```

```
grid_result2 = gsc2.fit(X_train, y_train)
best_params2 = grid_result2.best_params_
print(best_params2)
```

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 out of  54 | elapsed:  3.4min remaining:   59.0s
[Parallel(n_jobs=-1)]: Done  54 out of  54 | elapsed:  4.4min finished
```

```
{'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
```

{'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}

Figure 18 - GridSearchCV for RFR

```
rfr = RandomForestRegressor(
    n_estimators=100,
    min_samples_leaf=1,
    min_samples_split=10,
    bootstrap=True,n_jobs=-1,
    random_state=42)
```

```
rfrlog = RandomForestRegressor(
    n_estimators=200,
    min_samples_leaf=1,
    min_samples_split=10,
    bootstrap=True,n_jobs=-1,
    random_state=42)
```

```
%%time
rfrlog.fit(Xlog_train, ylog_train)
```

```
Wall time: 4min 22s
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=200, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

```
%%time
rfr.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=10, min_weight_fraction_leaf=0.0,
                      n_estimators=200, n_jobs=-1, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```

Figure 19 - Random Forest Regressor for Price and logPrice

4.1.5 Results comparison

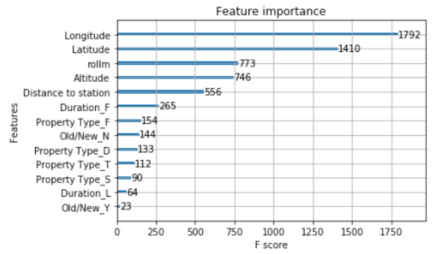
RMSE is the chosen comparison metric for this project. We will compare 4 different models in total, these are: XGBoost-Prices, XGBoost-logPrices, RFR-Prices, and RFR-logPrices. They represent the four combinations of algorithm and target variable possible in our study.

Price - XGBoost

```
RMSE = np.sqrt(mean_squared_error(y_test, xgb_pred))
print(RMSE.round(4))
```

```
65804.8682
```

```
%%time
from xgboost import plot_importance
plot_importance(xgb, max_num_features=15) # top 10 most important features
plt.show()
```



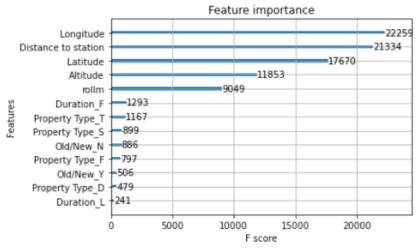

```
CPU times: user 252 ms, sys: 15.7 ms, total: 267 ms
Wall time: 271 ms
```

logPrice - XGBoost

```
RMSE = np.sqrt(mean_squared_error(y_test, xgblog_pred_rebased))
print(RMSE.round(4))
```

```
66764.2592
```

```
%%time
from xgboost import plot_importance
plot_importance(xgblog, max_num_features=15) # top 10 most important features
plt.show()
```




```
Wall time: 1.36 s
```

Price - RFR

```
RMSE = np.sqrt(mean_squared_error(y_test, rfr_pred))
print(RMSE.round(4))
```

```
65850.7396
```

```
RFRfeatures
```

	0
Altitude	0.037493
Distance to station	0.034466
Duration_F	0.016496
Duration_L	0.016109
Latitude	0.128872
Longitude	0.153049
Old/New_N	0.003847
Old/New_Y	0.003839
Property Type_D	0.020829
Property Type_F	0.003152
Property Type_S	0.007017
Property Type_T	0.003940
rollm	0.570892

logPrice - RFR

```
RMSE = np.sqrt(mean_squared_error(y_test, rfr_pred_rebased))
print(RMSE.round(4))
```

```
66149.1629
```

```
RFRfeatureslog
```

	0
Altitude	0.028758
Distance to station	0.026887
Duration_F	0.007587
Duration_L	0.020508
Latitude	0.101324
Longitude	0.122757
Old/New_N	0.003399
Old/New_Y	0.005654
Property Type_D	0.012568
Property Type_F	0.001910
Property Type_S	0.006441
Property Type_T	0.004326
rollm	0.657882

Figure 20 - RMSE comparison

As you can see from the RMSE results RFR and XGBoost present similar performance with an accuracy of about £65k pounds. Tranforming the target variable to logn turns to slightly decrease the performance of the models.

These results meant that these models are able to predict prices for any house in london provided the postcode with a +-£65k confidence interval.

Figure 20 shows a comparison of the 4 RMSE and figure 21 shows a plot comparison between the estimated values and the real ones.

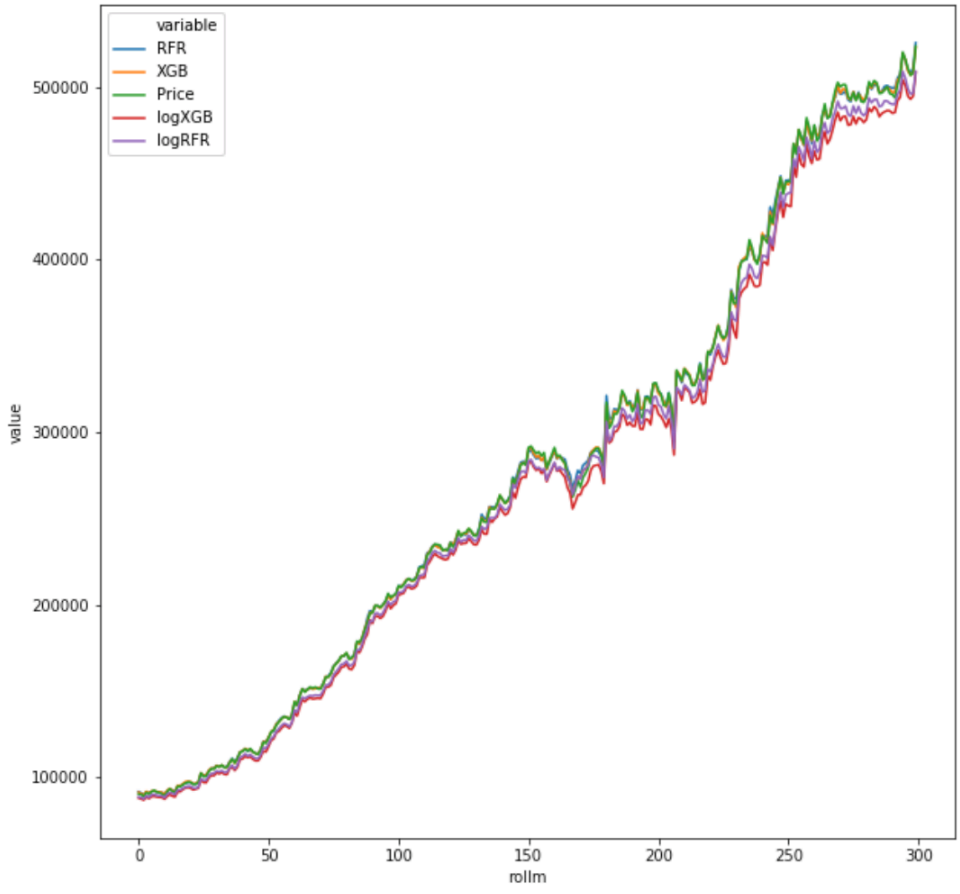


Figure 21 - Monthly regression mean superimposed

5. Using Model and Recommendations

In order to make predictions, one would need to input data for each of the features, scale the input data, and take it into the model. As a result, one can achieve about significant accuracy in predicting the house prices for any postcode in London and at any point in time in the future. However, there are some limitations, the model only works for properties in prices between £50k and £1.1m and with an accuracy of £+-65k.

6. Conclusions

We explored the residential transactions across the UK and London specifically and managed to achieve a an optimized RMSE. Unfortunately, the features were not rich enough to have a more accurate prediction. Nonetheless our result allows us to have a clear comparison between the expected properties prices and the ones offered in the corona-market economy.

The model can be run to predict a new observation dated to the current month and any postcode. For example, figure xx shows an updated entry for postcode 'W4 1PE' to reflect the expected price of the property in 06-2020. The model does not discriminate between number of beds or baths as there is no data to link transactions to flat descriptors. However, our study is accurate enough to help any individual inform themselves better about real estate price projections to a very high level of detail. The XGB model was run to estimate a comparison on the price for a typical property at 'W3 7QN' in 06-2000, the predicted price for the property in 06-2012, and the predicted price for the property and 06-2020. **The predicted prices are £242k in 2000, £484k in 2012 (actual value £568k), and**

£616k in 06-2020. Interesting, huh?