House Prices Regression

Jack Gorton

Introduction

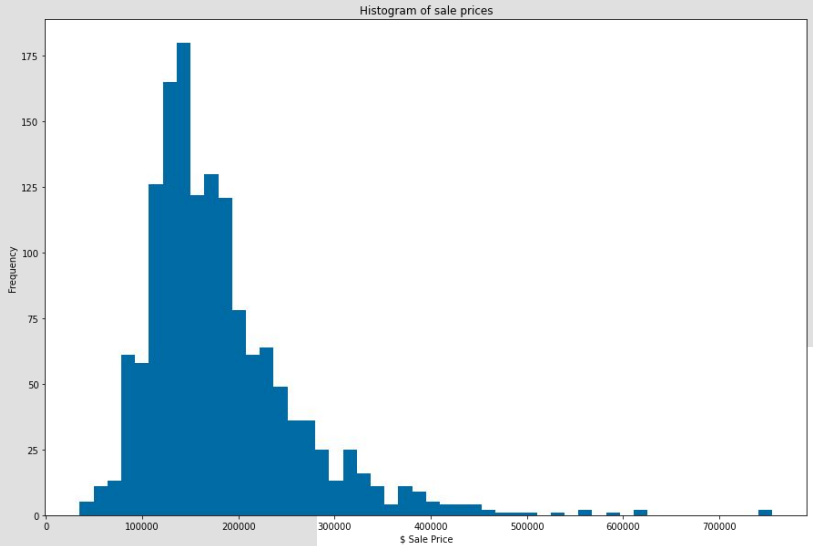
- The goal of this project is to use data describing various attributes of a house to predict the sale value of the house.
- The following slides will describe the process used to arrive at a model:
  - Data Exploration
  - Feature Engineering
  - Feature Selection/ Elimination
  - Model Building
  - Model Evaluation

Exploring the Data

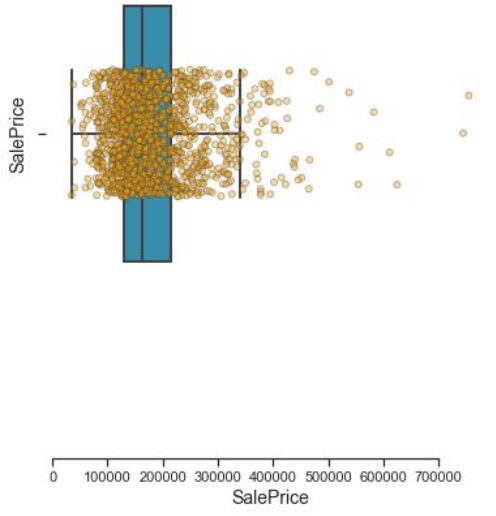
In order for us to generate meaningful results, we need to understand the data that we are using.

Sale Price- Overview

- We have a training set of 1460 records.
- An average house was sold for $180k, with an interquartile range of ~ $84k.
- Despite the majority of houses falling within a relatively small range, we can see a number of houses sold for significantly higher values --> consider excluding these, and some analysis on what caused these extreme sale values.
- Sale price is not a uniform distribution, and displays a long tail. This may impact some of our conclusions later.

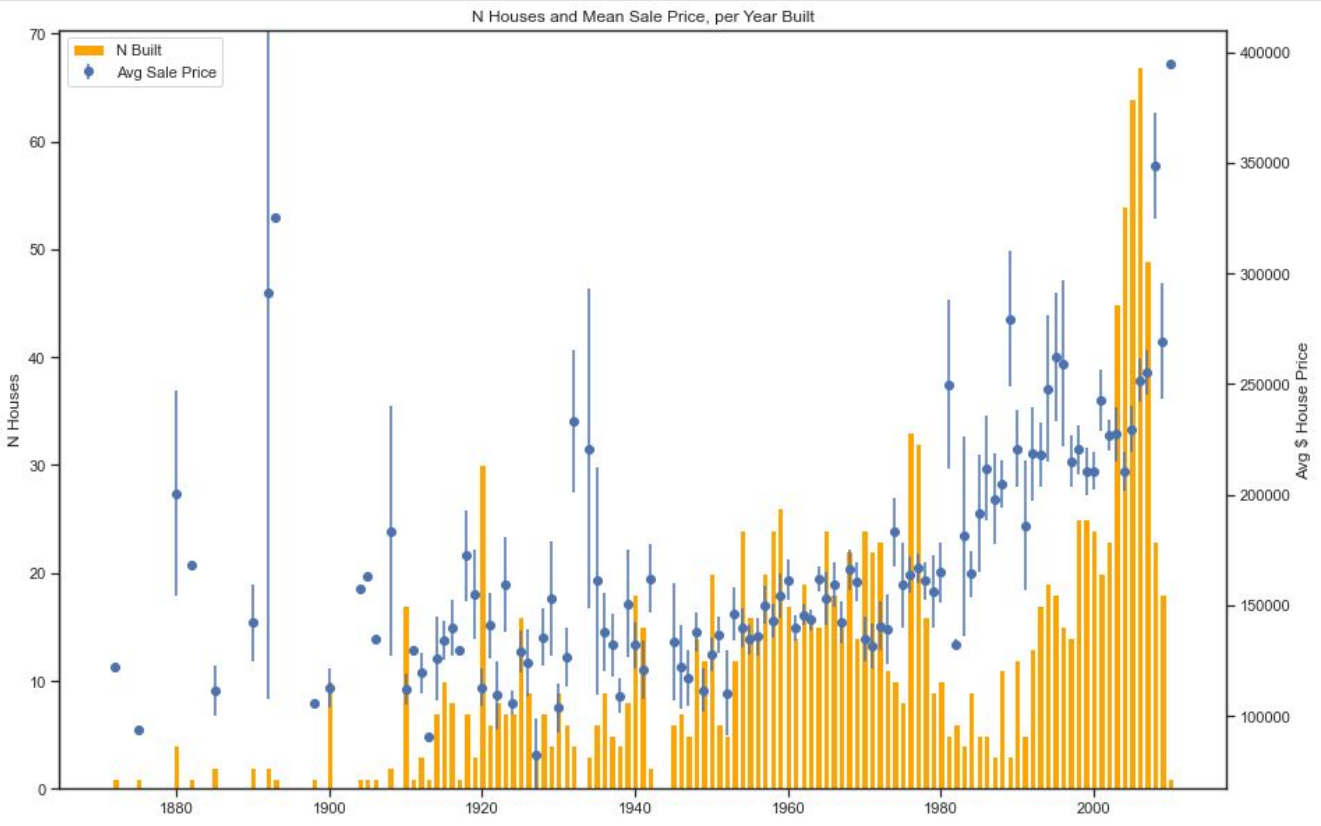


	value
count	1460.0
mean	180921.2
std	79442.5
min	34900.0
25%	129975.0
50%	163000.0
75%	214000.0
max	755000.0



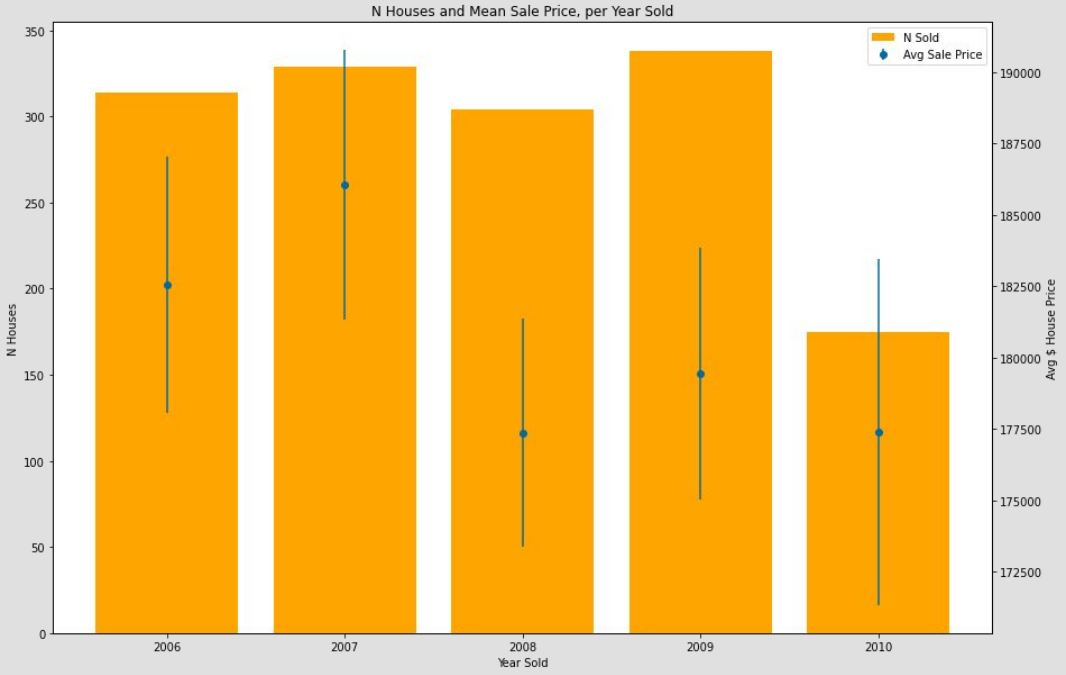
Sale Price and Year Built

- Sold houses were built across a large timeframe.
- Upward trend of house prices- houses built more recently were sold for a higher price.
- Large dip in houses sold that were built in the 80's-
  - Indicative of dearth of house building in the overall market?
  - Are people that bought a new house in the 80's more likely to stay put?



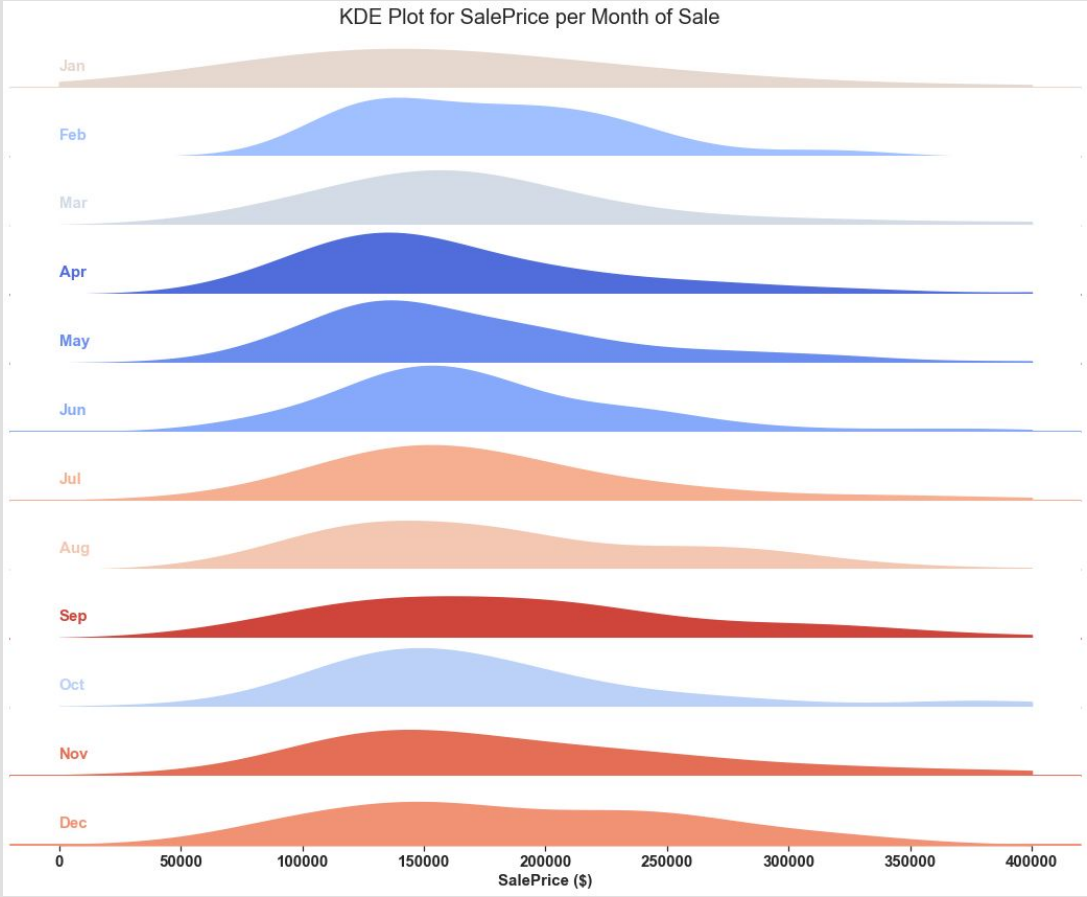
Sale Price and Year Sold

- Houses were sold across a 5 year range.
- We can clearly see the effects of the 2008 market crash here:
    - In 2006-2007 houses were selling for $182-186k on average
    - In 2008 onwards, house sales dropped to $177-179k on average



Sale Price and Month Sold

- There is not a clear best month to maximise sale prices.
- we can however see that houses sold in latter parts of the year are selling for more money on average.

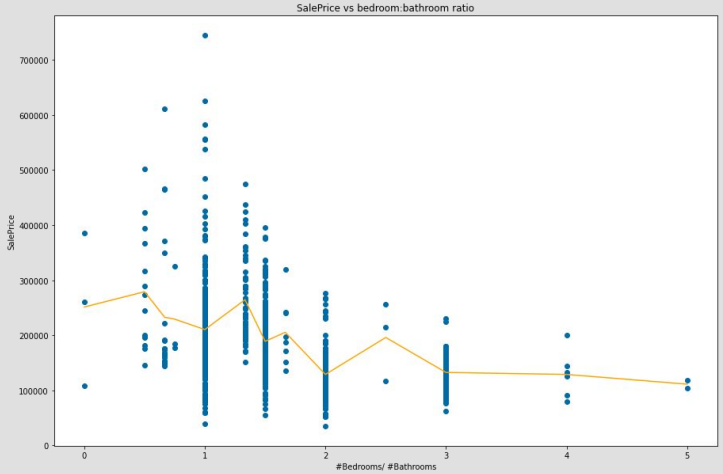


Feature Engineering

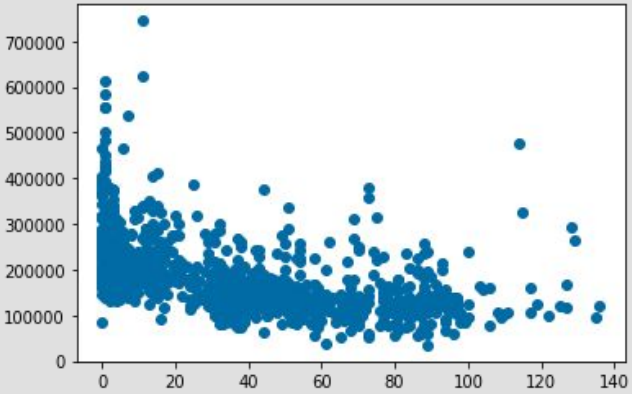
It is important to use industry knowledge to generate features from a dataset, which may provide a new dimension of parameter space that was not covered by the base features. An example would be using a reported income and balances of lines of credit to calculate someone's debt-to-income ratio.

Feature engineering

- Example 1: hypothesis: A house that has a higher bathroom to bedroom ratio will be worth less..
- Result: True (although relationship is relatively weak)



- Example 2: Calculating House Age will be better than using the year it is built.
- Year Built does not take into account when it was sold- we may have two houses from the same year that are 7 years older compared to their sale date



Feature engineering

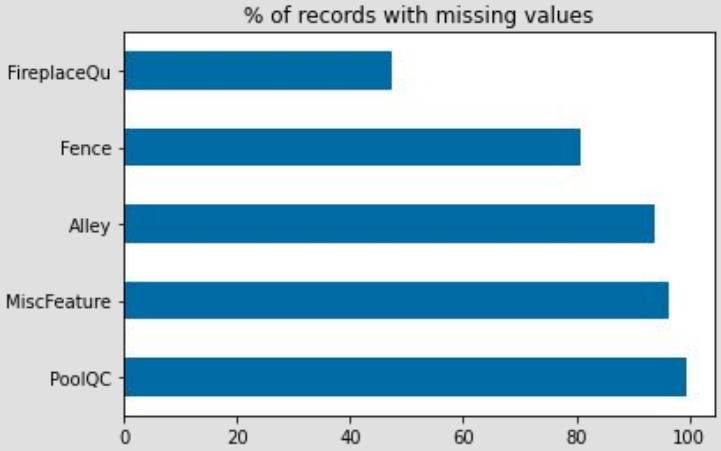
- What % of lot area is covered by house (perhaps someone with a lot of land is worth more)
- What % of basement is converted into livable space (having extra basement room that can be lived in is likely to increase price)
- Has the house ever been remodelled? A remodelling date was provided, but a lot of houses have not received remodelling, so this would make more sense as a flag.

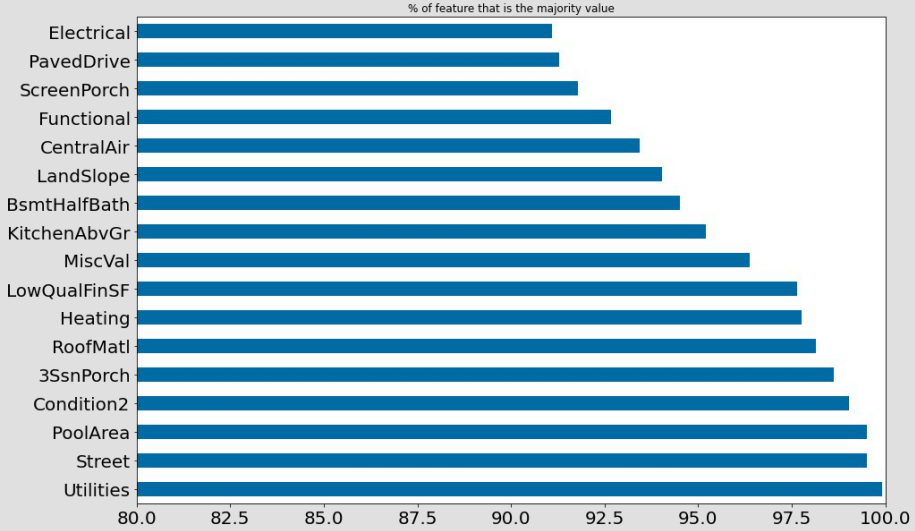
Selecting and Eliminating Features

A model may have worse performance, or be more subject to overfitting if we include all features. This may be due to including highly correlated features, or features that show some mild predictivity due to chance in the training set but not the test set.

Missing (and Repeated) Values

- We are unlikely to want to use any features that are all one value- they are not providing any way of differentiating the target of a record.
- Here we can see that PoolQC, MiscFeature and Alley all have >90% missing values, therefore will likely be excluded.
- As we can see, finding only nan values is half the story. If we check features for the % of entries that are the most common value, we can see that there are many features that are almost entirely one value.





Imputation, Scaling, Encoding

- Two methods were used to impute missing values:
  - For continuous data, it was imputed with the column mean
  - For categorical data, a new ‘MISSING’ group was added.

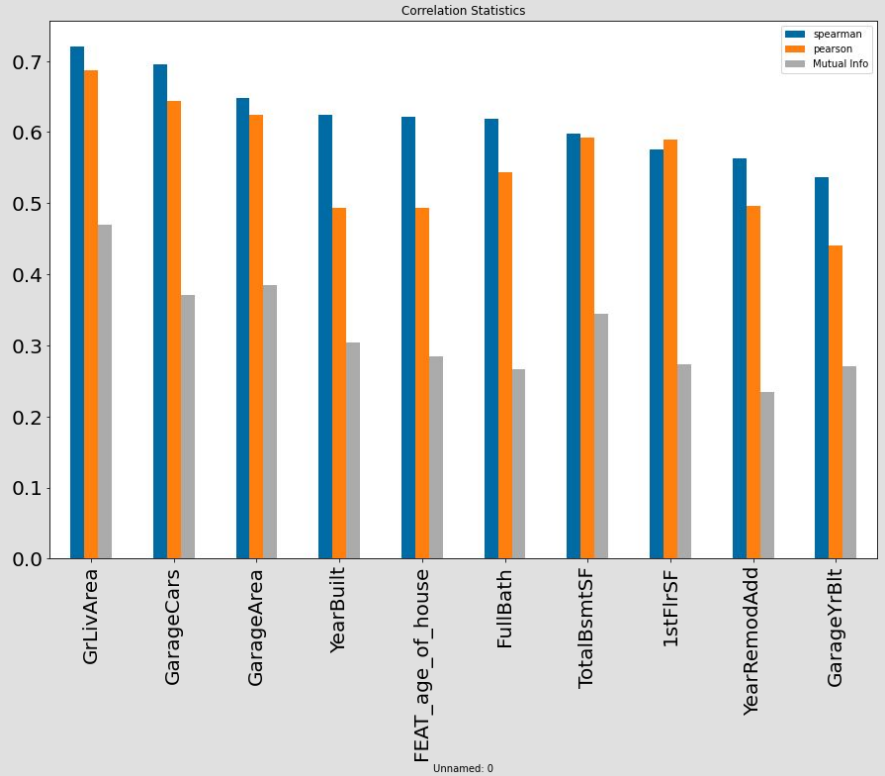
- These are very rudimentary imputations, and were chosen due to time constraints.

- For categorical data, three methods were used to encode the data:
  - For categories with numeric tags, they were left as they are
  - For categories with a known scale (e.g. Excellent, good, poor), ascending numerical values were assigned
  - For all others, an OrdinalEncoder was used.

- The data was all scaled using a StandardScaler class. This is to ensure features are not having a larger impact on the model due to having a larger scale to other features.

Assessing Feature Predictivity

- Typically when modelling we would want to exclude features for a number of reasons:
  - They do not hold any information on the target
  - The information they hold is already covered by another variable (that does it better).
- Here i have calculated various correlation statistics between features and the sale price, to get an idea of how well one follows the other.
- The plot shows the top 10 features according to spearman rank. Having a larger living area, a garage, and a newer house are all indications of a higher sale price.



Model Results

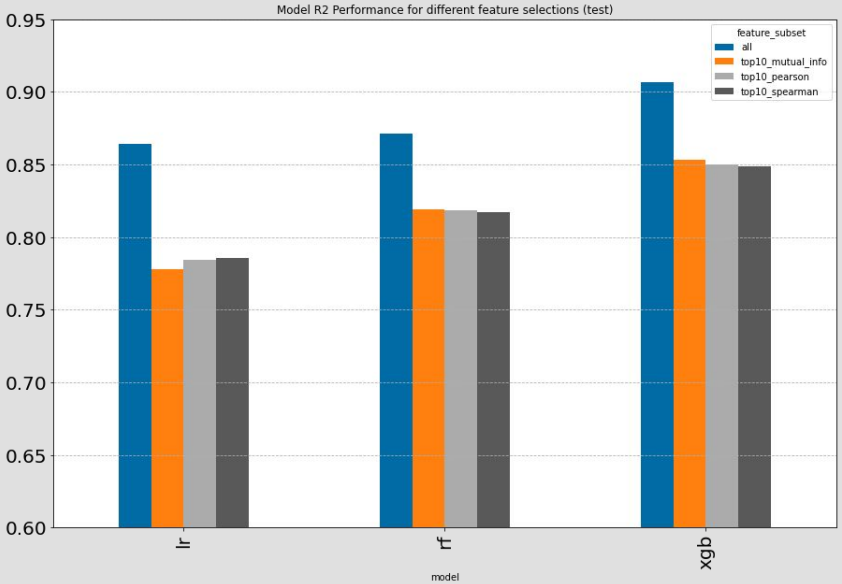
Modelling

- Three different classes of models were assessed:
  - XGBRegressor
  - RandomForestRegressor
  - LogisticRegression

- Additionally, different subsets of features were assessed.

- A comparison of results can be seen in the grouped bar chart- the R2 score for the test set is presented for each subset, and each regressor.

- XGBoost is the clear favourite here- However this is likely due to how i have preprocessed the data.



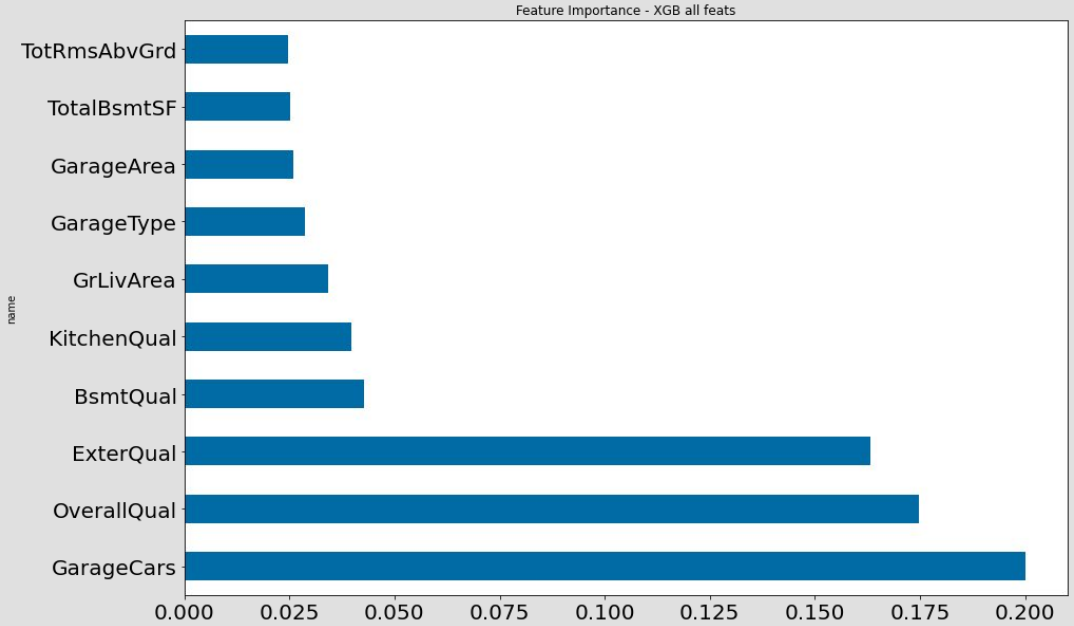
Results

- It is clear that using all data is yielding the best results across all regressors. This is likely due to only including 10 features for the subset models.
- Additionally by only including the top ten per metric, we are never going to be combining categorical with continuous.
- It is extremely strange that we are seeing better performance in the test set compared to the train for logistic regression- more investigation needed

model	train_test	r2	rmse	mse	mae	r2_adj
lr	train	0.82	33,690	1,135,022,429	20,691	0.81
lr	test	0.86	28,864	833,158,211	20,027	0.85
xgb	train	0.95	18,313	335,368,696	12,910	0.94
xgb	test	0.91	23,940	573,141,625	15,087	0.89
rf	train	0.9	25,703	660,634,889	16,881	0.89
rf	test	0.87	28,070	787,941,804	18,120	0.85

Importance of Features

- We see many of the top features from the correlation appearing in the top 10 most important to the final result.
- There are a number of reasons to explain why those that appear here but not in the correlations:
  - Our spearman correlation did not pick up on the relationship (possible that it is non-linear)
  - The XGB regressor has picked up on some combination of features (our correlation is taking one feature only)
  - There is a mistake somewhere in the code



Improvements, Next Steps

- Did not exclude the extreme values of sale price from the training data.
- Did not scale the sale price values when modelling- possible factor in poor Logistic Regression performance.
- When tackling categorical data, we have simply used a LabelEncoder. We may instead achieve better results with a OneHotEncoder.
- When imputing missing values we have simply taken the mean- other options may yield better results (e.g. attempt to predict the missing values with a regression)
- We have not taken into account any correlation/colinearity between model features- this may be contributing to overfitting/ underperformance.
- We have done very limited feature selection in general.
- We have not performed any hyperparameter tuning for the xgb or rf models. Ideally i would use a nested cross validation here, with stepwise feature additions.