London Business School

Estimation Engine for London Real Estate Investment

Uzair Zaidi

Summary

This report details the creation and evaluation of an estimation engine designed to predict property prices in London, as well as the selection of 200 potential investment targets. The data originates from a curated 2019 London housing dataset. Four distinct machine learning algorithms—Linear Regression, Random Forest, XGBoost, and Gradient Boosting Machine (GBM)—were trained on a subset of the data. Each algorithm was tuned extensively to enhance predictive capacity on a validation split. An ensemble (stacking) approach was then employed to fuse these models, resulting in improved performance metrics (R^2 and RMSE)

After developing and validating the stacked model, it was applied to the out-of-sample test dataset, which contains only asking prices rather than actual sale prices. The final step involved ranking properties by projected "profit," computed as (predicted_price−asking_price)/asking_price and selecting the top 200 properties. This report addresses the methodology of this estimation engine, highlighting data preparation, model tuning, and integration.

Introduction

London property prices have skyrocketed, especially in recent times, surpassing wage growth by a large margin. The objective is to construct an estimation engine

that guides investment decisions in the London real estate market by predicting house prices. The provided data consists of a training set with actual, finalized prices (to which we have complete access), and an out-of-sample testing set containing only asking prices. Our target is to:

1. Identify which algorithm (or combination thereof) yields the most accurate predictions on unseen data.
2. Use the best model to forecast out-of-sample property prices.
3. Select 200 houses that promise the highest profit margins, assuming the asking price is what one would pay, while the model's predicted price is a stand-in for actual value.

Structure:

1. Data – how it was curated, cleaned, and split for training and validation.
2. Modeling – approaches employed
3. Stacked Algorithm (Ensemble) – rationale, methodology, and gain in performance.
4. Investment Selection – formula for profit, logic for ranking properties, and final picks.
5. Extensions – possible improvements, future directions, and commentary on Crossrail's potential effect.
6. Conclusions and Recommendations – overall takeaways, key results, and recommendations for real estate investment.

Data Details

- The training data included 14,000 properties sold in 2019.
- The testing set includes ~2,000 properties on the market in 2019, with only asking prices available.
- Variables included property type, floor area, number of rooms, energy ratings, distances to transport, latitude/longitude, average income of the area, etc.

The ultimate deliverable is a CSV file (Zaidi_Uzair.csv) with an added 'buy' column, marking exactly 200 properties as good investments.

1. Data Preparation

- **Reading/Cleaning**:
  The "training_data_assignment_with_prices.csv" and "test_data_assignment.csv" files were read in, columns such as address2, town, and date were removed. Missing values in population were imputed using the median (assumed population was missing at random). Factors and numeric transformations were applied to maintain consistency.
- **Split**:
  A 75–25% initial split (via initial_split) was performed on the training data, producing a training_set (75%) and a validation_set (25%). The validation set's original copy was stored (validation_set_original) to preserve the actual prices for final evaluations.
- **Feature Engineering**:
  Four new features were engineered:
  1. rooms_per_area: number_habitable_rooms / (total_floor_area + 1).
  2. distance_to_center: Euclidean distance of each property's lat-lon from central London (approx. 51.5074N, -0.1278E).
  3. log_distance_station: log(distance_to_station+1)
  4. log_energy_consumption_current: log(energy_consumption_current+1)
  5. log_price: log(price) → many "high-price" outliers, used log to diminish skewness of price.

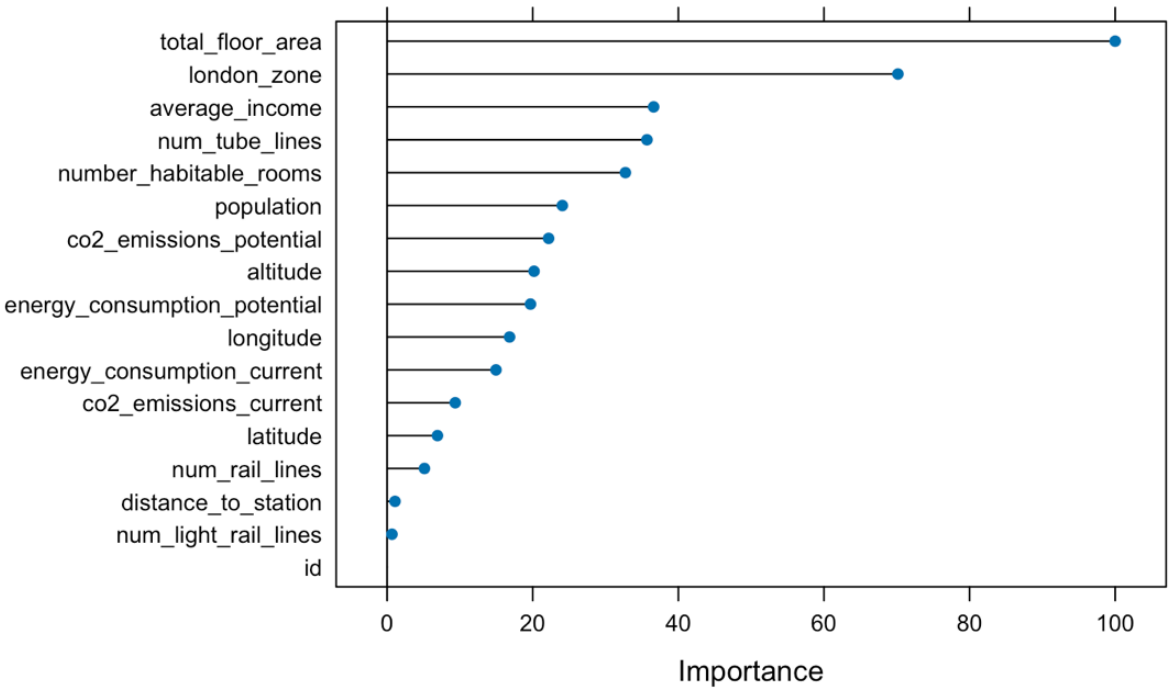
  These transformations aimed to capture non-linearities and scale wide-ranging variables.

Although a linear regression or a single decision tree might label certain variables as "insignificant" statistically, the choice was made to retain all top-correlated features. Non-linear algorithms (Random Forest, XGBoost) often extract hidden interactions or splits that linear models miss. A variable that looks unimportant in a purely linear context could potentially still meaningfully boost performance when combined with other features in tree-based and boosting methods. Keeping all of the top-correlated predictors, we avoid prematurely discarding information that might prove useful once models explore more complex (non-linear) relationships.

2. Modeling

2.1 Linear Regression (Baseline)

- **Method**:
  Built an R lm model using the top correlated variables (from correlation table)
  → top_vars: sorted by absolute correlation in descending order.
  - RMSE on validation ~ 321,100.
  - R^2 ~ 0.63.
  - Although straightforward, linear regression often underfits complex property data.

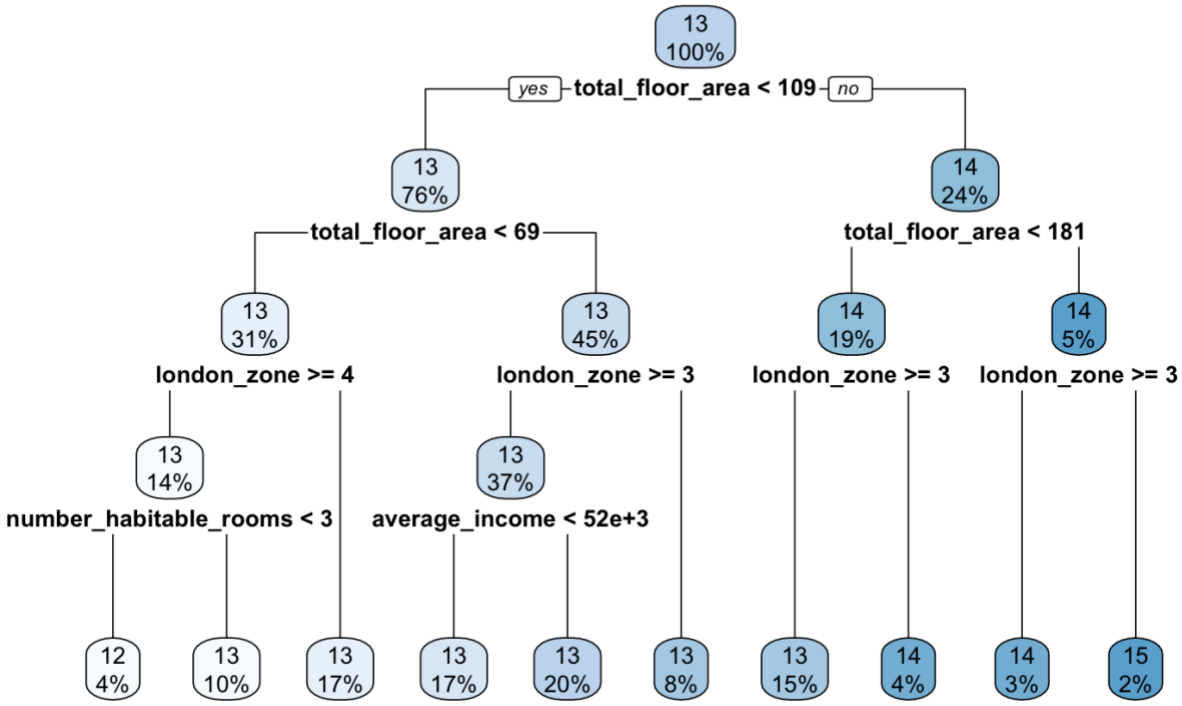


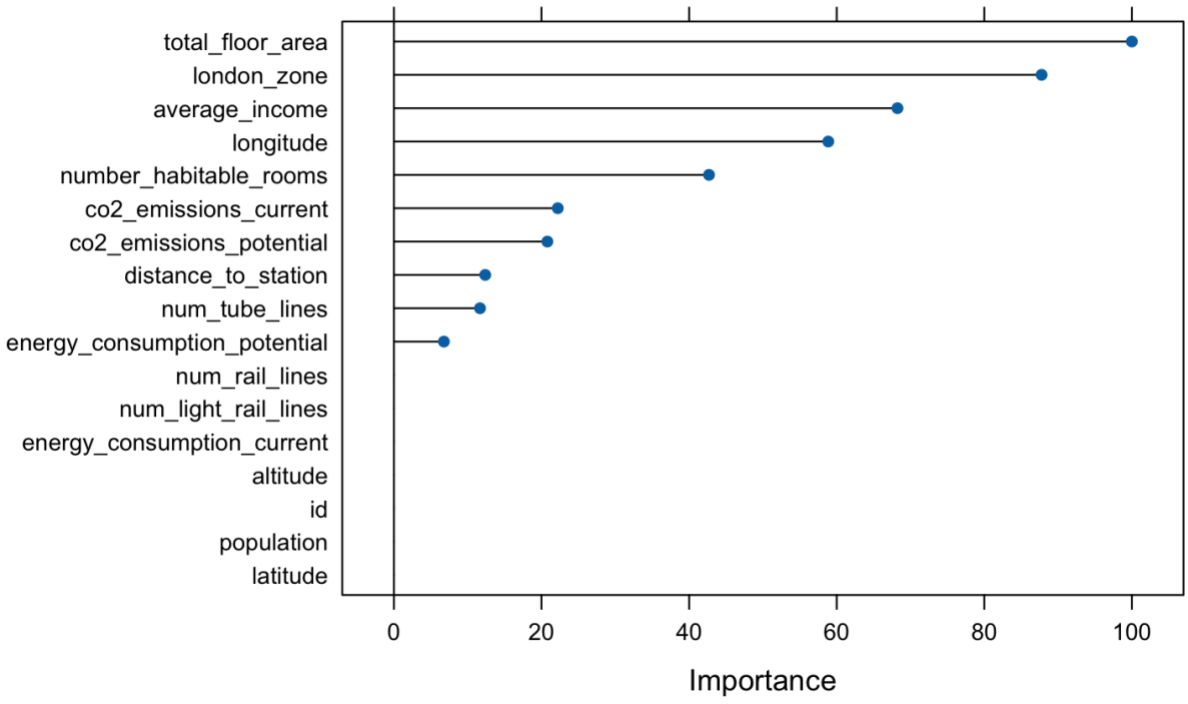
2.2 Decision Tree (rpart)

- **Implementation**:
  - Used caret grid search (tuneLength=10) to pick an optimal complexity parameter (cp).

- Turned each property price into log(price) for training, then exponentiated predictions to revert to original scale.

- **Results**:
  - Considerably simpler than advanced ensembles but captured some non-linearity.
  - However, the high-level RMSE remained above 300k on the raw scale, with moderate R^2 in the 0.60–0.65 range.


13
100%
yes
total_floor_area < 109
no
13
76%
14
24%
total_floor_area < 69
total_floor_area < 181
13
31%
13
45%
14
19%
14
5%
london_zone >= 4
london_zone >= 3
london_zone >= 3
london_zone >= 3
13
14%
13
37%
number_habitable_rooms < 3
average_income < 52e+3
12
4%
13
10%
13
17%
13
17%
13
20%
13
8%
13
15%
14
4%
14
3%
15
2%

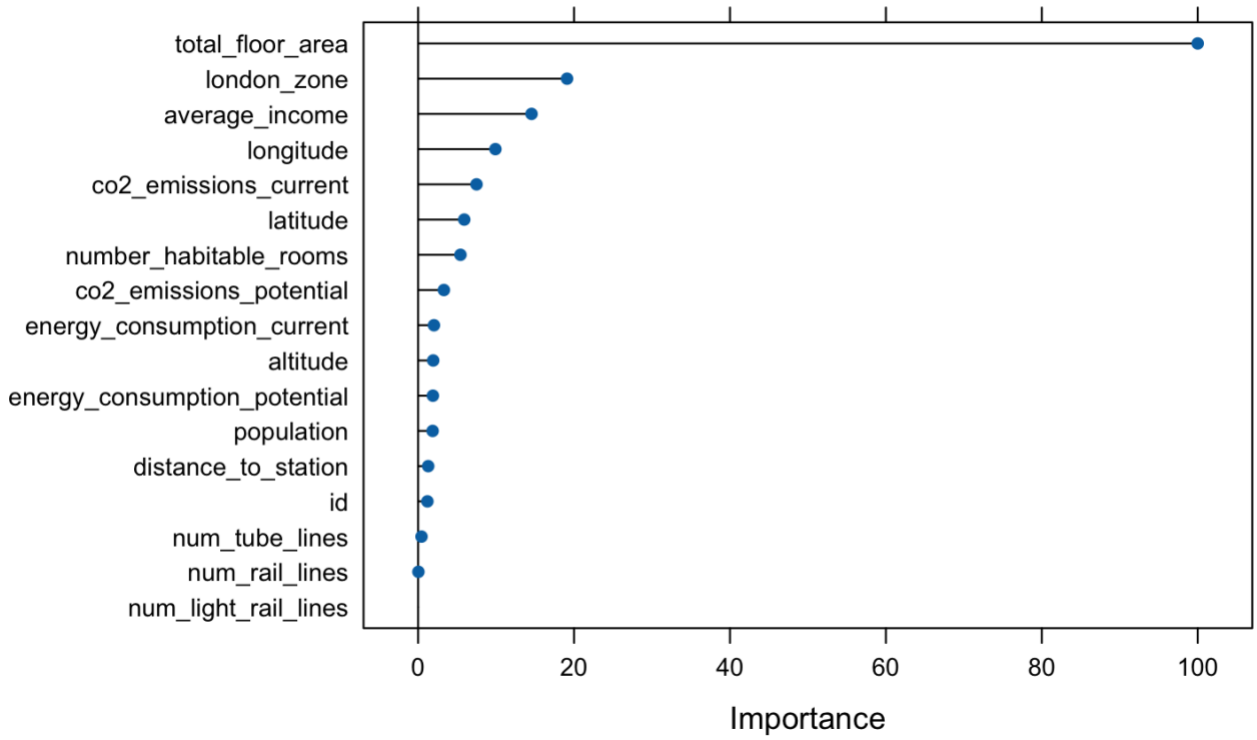

total_floor_area
london_zone
average_income
longitude
number_habitable_rooms
co2_emissions_current
co2_emissions_potential
distance_to_station
num_tube_lines
energy_consumption_potential
num_rail_lines
num_light_rail_lines
energy_consumption_current
altitude
id
population
latitude
0
20
40
60
80
100
Importance

2.3 Random Forest

- **Method**:
  - Used ranger with expanded grid for hyperparameters (mtry, min.node.size, splitrule).
  - Number of trees set to 200.
- **Outcome**:
  - Validation RMSE ~ 192,548.7
  - R^2 ~ 0.856
  - Notable improvement over simpler methods.

2.4 XGBoost

- **Tuning**:
  - 5-fold CV, saving final predictions.
  - Grid: nrounds {100,200}, max_depth {3,5}, eta {0.1,0.05}, gamma {0,1}, etc.
- **Outcome**:
  - The chosen best was nrounds= 200, max_depth= 5, eta= 0.1, gamma = 0.
  - RMSE ~ 178406.3
  - R^2 ~ 0.867



2.5 Gradient Boosting Machine (GBM)

- **Implementation**:
  - 5-fold CV (to limit computation time), (caret + gbm), tuning interaction.depth, n.trees, shrinkage, etc.
- **Outcome**:
  - RMSE ~ 195,694.6
  - R^2 ~ 0.8423.

3. Stacked Ensembling

- **CaretStack**:
  Combined final versions of Random Forest (model_rf_min), XGBoost (model_xgb), and GBM (model_gbm).
- **Stack Performance**:
  - RMSE ~ 176,820.7
  - R^2 ~ 0.87

  - Marked improvement over any individual method alone. (XGBoost was best individual model)
- **Explanation**:
  - The stacked model overcame individual biases by combining predictions, capturing richer dynamics.

4. Choosing 200 Investments

- **Test Data**:
  - Contains only asking_price (no final sale).
  - Applied the same feature engineering (rooms_per_area, distance_to_center, log_distance_station, etc.).
  - Called predict(stacked_model, test_data) to get predictions_log_test → predicted_price.
- **Profit Formula**:

  $$\text{Profit} = (\text{predicted\_price} - \text{asking\_price})/\text{asking\_price}$$

- **Ranking**:
  - Sorted test data by descending profit.
  - Marked the top 200 with buy= 1, the rest buy= 0.
- **Output**:
  - Saved as a CSV named "Zaidi_Uzair.csv", retained all original test-data columns, plus two additional columns: predicted_price for the stacked model's forecast and 'buy' indicating the top 200 picks."
  - Sorted by descending profit and chose top 200

Extensions

---

1. **Expanded Tuning**
   - Increase the number of trees in Random Forest / XGBoost or refine the learning rate, edit parameters.
2. **Additional Data**

- Integrate more macroeconomic or city-planning data (local council projects, infrastructure developments).
- Additional 'house condition' data to analyze maintenance
- Local School Ratings
- Short-Term Rental value

Crossrail (Elizabeth Line) Procedure

Context and Motivation
The Elizabeth line is a major addition to London's transit network, running from west to east at a cost of approximately £20 billion. Properties near new or upgraded stations could potentially see above-average price increases due to reduced commute times, increased connectivity, and heightened neighborhood desirability. The crux is whether these prospective gains have already been "priced in" by the market, or if there remains unrealized price appreciation.

1. Collecting Crossrail Data

1. **Station Locations**: Gather precise geospatial coordinates (latitude/longitude) for each Crossrail station (whether completed or under construction).
2. **Timeline / Status**: Note the opening dates (if the station is already operational, or if future completion is expected). This helps segment properties by their stage of Crossrail readiness.

2. Enhanced Feature Engineering

A new variable could be created for each property in both the training and testing sets to capture Crossrail station proximity:

- **`distance_to_crossrail`**: The Euclidean or road distance from the property's coordinates to the nearest Crossrail station.
- **`within_crossrail_radius`**: Binary feature (1 if the property is within a chosen cutoff—for instance, 1 km or 2 km—of a Crossrail station, 0 otherwise).

3. Modeling Crossrail Impact

1. **Edit Existing Model**:
    - Add the new Crossrail features (`distance_to_crossrail` or `within_crossrail_radius`) to the training data.
    - Retrain the entire modeling pipeline (particularly the stacked model) and compare performance metrics with and without these new features.
    - A significant improvement in R^2 or a shift in variable importance (favoring Crossrail features) could possibly indicate that the market effect of Crossrail is at least partly captured by the new distance and radius features.

4. Procedure to Identify "Unpriced" Opportunities

1. **Generate Predictions on Out-of-Sample**:
    - Produce new `predicted_price_crossrail` for each property in the test data.
    - Compare these predictions to the simple "non-Crossrail-aware" model (stacked_model). If the predicted price from the Crossrail-aware model is consistently higher for certain postcodes, that suggests Crossrail benefits are not fully reflected in the original out-of-sample set.
2. **Compute "Crossrail Profit"**:
    - For each property, compute `profit_crossrail = (predicted_price_crossrail - asking_price) / asking_price`.
    - Sort properties by descending `profit_crossrail`.
3. **Compare / Evaluate**:
    - If the highest ranking properties cluster around new Crossrail stations, that strongly suggests there is still a profit opportunity tied to future or recent improvements in transport connectivity.
    - Compare how "Crossrail Profit" differs from the baseline "Profit" to see if certain neighborhoods show a big jump once Crossrail variables are included.
4. **Filter / Sensitivity Analysis**:
    - Optionally, only highlight properties that are within 1 km (or 2 km) of a Crossrail station and see if the projected profit is significantly higher than properties outside that radius.

- Helps pinpoint which neighborhoods or postcodes could still be undervalued.

5. Limitations

- **Timelines**: If the model uses 2019 data, stations that opened (or partially opened) later might not be fully reflected in the sale prices.
- **Data Availability**: Accurately modeling Crossrail's impact benefits from updated data points (post-2019 sales, partial openings, etc.)… real-time data would be crucial.

Conclusions and Recommendations

- **Conclusions**:
  1. **Stacked Model** is the strongest performer, with an RMSE of ~176,820.7 and R^2 ~0.87 on validation, outperforming each base learner.
  2. **Random Forest** and **XGBoost** individually also offer robust predictions (R^2 in the 0.85+ range), but stacking confers additional gains.
  3. The engineered features (log scales, distance metrics) proved crucial for capturing non-linear relationships and boosting overall performance.

Overall, the stacked ensemble approach delivers a viable solution for guiding investments under the assumption that the difference between "predicted price" and "asking price" is indicative of potential profit.

No additional raw data used; validation sets and final CSV are straightforward. Detailed tuning grids are available in the code.