

Renting a Room of One's Own: Analyzing Airbnb Listings in DC and the Factors Fetching Top Prices

Team Zero

Devin Brande, Domenico Fieni, and Brittany Gregerson

Georgetown SCS Data Science Program

Cohort 6 - Summer 2016

Github: github.com/georgetown-analytics/team-zero/

Introduction

At its core, Airbnb is an online marketplace that connects travelers with locals who host guests in their available space - from a shared room to entire house - to meet people and earn extra income. Travelers benefit from the opportunity to book unique accommodations and to interact with locals, often for less than the price of a hotel room. Founded August 2008 in San Francisco, CA, Airbnb is now valued at over \$30 billion¹ and is present in more than 34,000 cities in 190+ countries worldwide.²

Problem

Airbnb provides the listing, booking, and payments platform as well as insurance and assistance with things like cleaning and professional listing photos. However, Airbnb provides only limited help with pricing, a difficult and persistent challenge for current and would-be hosts, who must ask themselves; what is the best price; which aspects of a listing have the biggest impact on market value; when should prices fluctuate, if at all, and in response to what conditions? These are only some of the questions existing or potential hosts wrestle.

Analysis

Our analysis delves into Airbnb data for Washington, D.C. in an attempt to begin answering some of these questions. We had three major goals: 1) explore and characterize Airbnb housing inventory in Washington, D.C.; 2) determine the factors with the greatest influence on listing price; and 3) provide current and potential Airbnb hosts with suggestions for pricing their listing(s) based on its key features, which encompass both physical property characteristics and host behavioral characteristics.

The outcome of our analysis as presented in this report is a deeper understanding to help hosts price their listing(s) within the D.C. market. While there may be no ideal price, there are key features, both in terms of host behavior and property characteristics, that can help a guide a host towards methods and behaviors to improve their pricing.

¹<https://techcrunch.com/2016/08/06/airbnb-raising-a-reported-850m-at-a-30b-valuation/>

² <https://www.airbnb.com/about/about-us>

Our Data

Our data is a single snapshot of Airbnb listings information for Washington, D.C. taken on 3 October 2015 by Murray Cox, a Brooklyn-based community activist unaffiliated with Airbnb, and provided under Creative Commons CC0 1.0 Universal license through his website <http://insideairbnb.com>.³ The dataset consists of four CSV files, one each for listing data, listing reviews, listing calendars, and neighborhood descriptions, plus a GeoJSON file containing multipolygons corresponding to each neighborhood. Of those, our principal dataset and the focus of our analysis is the detailed listing data (titled on the Inside Airbnb website as `listings.csv`), consisting of 3,724 rows, each one corresponding to a unique Airbnb listing, and 92 columns of features describing each listing.

In addition, we made use of a dataset provided by the Washington Metropolitan Area Transit Authority (WMATA), through their website, which provides D.C. Metro station locations based on latitude and longitude (among other information).⁴ Metro station location information was used to enrich our analysis through visualization and pricing analysis based on proximity to a key element of public transit in D.C.

Hypothesis Development

After settling on the domain and dataset for our project, we approached the problem, that is, the lack of robust information and resources to help Airbnb hosts inform their pricing, from multiple avenues in developing a hypothesis:

- Predicting the optimal price for listings, perhaps through development of a price calculator app for hosts.
- Rationalizing real estate cost and Airbnb nightly rate, in essence identifying the best opportunities for would-be real estate investors.
- Understanding the impact of price as it relates to Metro station proximity, reviews per listing host, and potential information revealed through studying hosts with multiple listings
- Identifying which listing features are more predictive of price
- Exploration of the data using various machine learning models and visualization tools

Eventually, we developed the following hypothesis:

We hypothesize that listing price is determined by certain key features, which can be identified, and which may or may not be under the host's control. Further, we suspect listings that fetch the highest prices and occupancy rates are 1) in desirable locations, as defined by proximity to

³ Inside Airbnb: Get the Data (<http://insideairbnb.com/get-the-data.html>)

⁴<https://developer.wmata.com/docs/services/5476364f031f590f38092507/operations/5476364f031f5909e4fe3311>

desirable features of Washington, D.C., 2) good values based on price per occupant, and 3) posted by superior hosts as measured by listing reviews.

Our goal is to explore and analyze the available data to provide enhanced, granular insights into Washington D.C. Airbnb listings, in particular the characteristics that have the greatest impact on price, to allow potential hosts to better price their listings.

Methodology

NOTE: All code discussed in this section is available for review or download on the Team Zero github repo: <https://github.com/georgetown-analytics/team-zero>

Data Science Pipeline

The starting point of our methodology was the data science pipeline, as shown in Figure 1

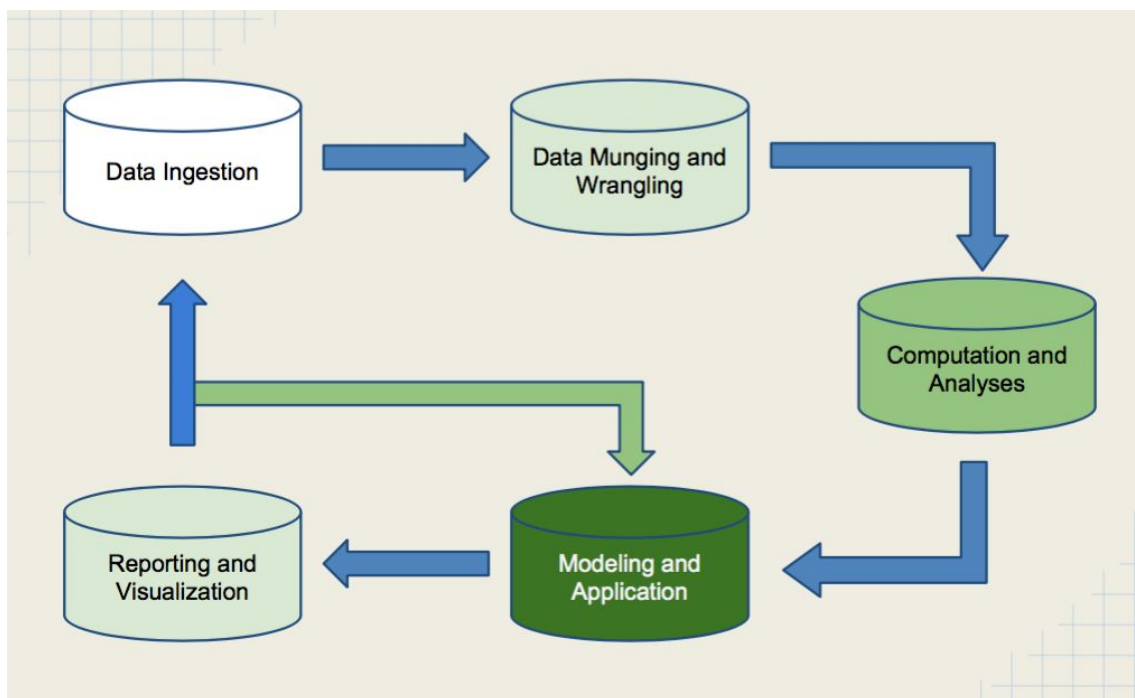


Figure 1: Data Science Pipeline

Project Architecture

Building from the data science pipeline, we developed a project architecture, shown in Figure 2, illustrating key components of the methodology and identifying analysis features or branches we anticipated to be primary development components.

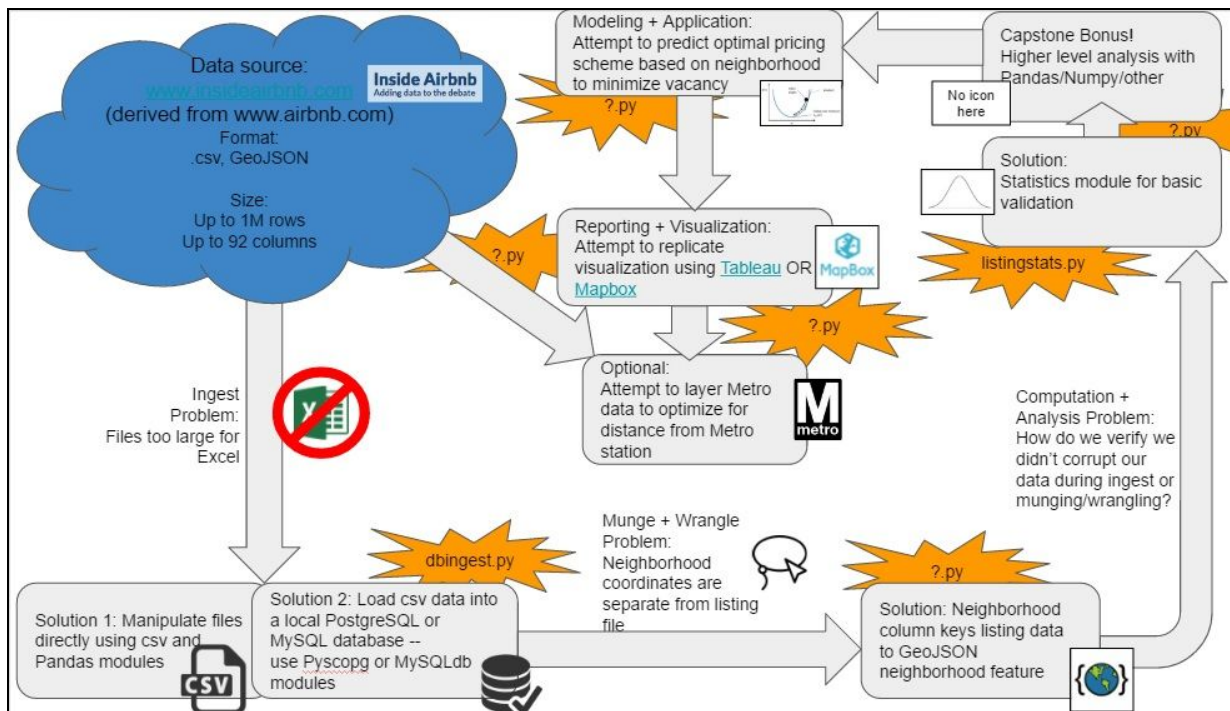


Figure 2: Team Zero Capstone Architecture

Each development component of our architecture roughly maps to a component of the data science pipeline.

Data Ingestion

Initial exploration of the CSV files provided by Inside Airbnb led us to believe that a database solution might help improve computational performance when performing downstream analysis -- even opening the listings.csv file using Microsoft Excel can be time consuming. To that end, we developed and implemented a rudimentary PostgreSQL database. Using the pycpg2 Python module we developed a short script, dbIngest.py, that ingests the CSV data from the local, working directory into a series of database tables in a locally hosted database.

Unfortunately, either the pycpg2 module or some component of our methodology is low performing, causing the ingest process to take nearly 4.5 minutes. Ultimately, we discarded our database solution and manipulated the data directly from the CSV files as stored in the user's working directory. This solution proved to be convenient and high performing, providing no indication that a database would have been a necessary intermediate ingestion step for our project.

Data Munging and Wrangling

Data munging and wrangling took place in several stages of the workflow, supporting key components of our analysis. First and foremost, focusing on the core listings.csv data that made up our principal data set, we had to perform careful data cleansing/preparation in support of

later analysis. This data preparation was performed largely using a single script, `csvMLCleanse.py`. The key functions of that script are to:

1. Capture the 43 columns/features we identified to be most relevant for our analysis
 - a. Free text fields such as host descriptions and reviewer comments were discarded since our analysis was not focused on language processing.
 - b. Non-descriptive fields, such as scrape id and scrape date (provided by Inside Airbnb) were discarded
 - c. Certain duplicative fields were removed, such as duplicate neighborhood information
2. Convert all features to string, float, or integer type, as appropriate.
3. Provide a mechanism to impute missing values where necessary
4. Provide a spot check on data integrity, through calculating the median price per night, to error check the ingestion process
5. Output the cleansed data to a new CSV for downstream analysis,

Further, our data ingestion script is easily tailored to support ingestion of other city data on Inside Airbnb, providing a convenient mechanism for expanding analysis.

Second, building from the cleansed data, we munged together WMATA Metro station latitude and longitude data in support of our geographic visualizations and public transit proximity analysis, discussed later, using latitude and longitude to orient listings to Metro stations.

Computation and analysis

Fundamentally, our modeling effort revolved around regression analysis using listing price as the target. In support of that, a key component of our computation and analysis effort was feature selection and feature engineering -- going from our original 92 listing features to a more manageable set of features we could use to perform modeling. Part of our feature selection effort was accomplished during data ingestion using qualitative analysis to reduce the overall feature count to 43.

Following data ingestion, we conducted exploratory analysis to see if key features related to host behavior and experience, reviews per month and number of host listings, were predictive of price. In this case, we used an engineered feature, price per accommodation, in an attempt to normalize listings that accommodate a wide range of guest quantities. However, as illustrated in Figure 3, our analysis indicates these particular features are in fact not highly correlated to price, leading us to the conclusion that a more holistic feature selection effort less dependant on subjective analysis was likely necessary.

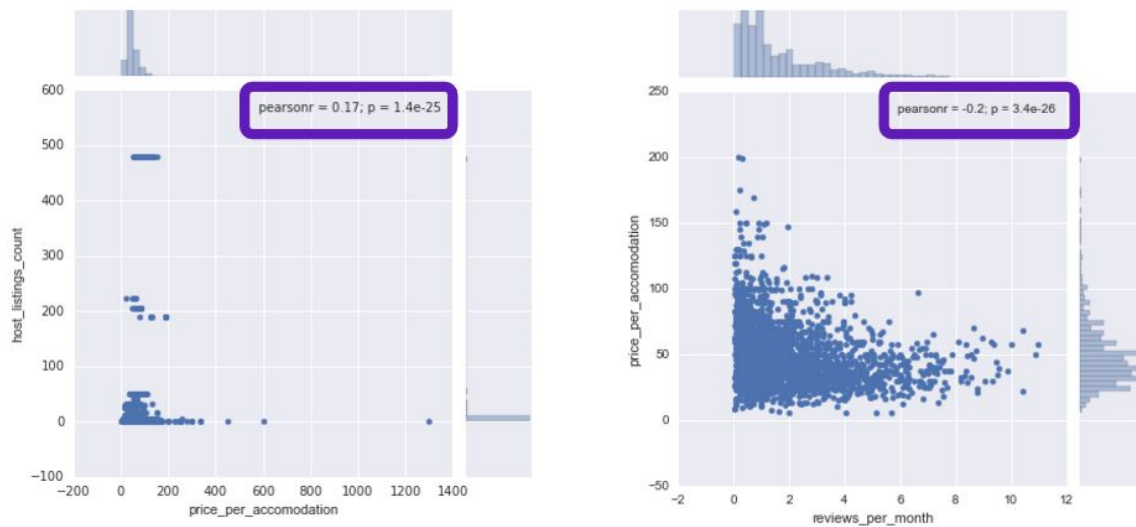


Figure 3: Host Behavior Feature Correlation to Price Per Accommodation

As a result, we explored regularization techniques and transformer methods to help identify the most predictive features. Using our feature selection script, `feature_selection_tz.py`, our first pass analysis explored only integer and float type features, bringing the feature count down from 43 to 34, using LASSO, Ridge, and ElasticNet regression. A truncated matrix of feature weights and selections, shown in Figure 4, illustrates the widely varying coefficients.

	LASSO forces weak features to have zeroes as coefficients, effectively dropping the least predictive features	Ridge assigns every feature a weight, but spreads the coefficient values out more equally, shrinking but still maintaining less predictive features.	ElasticNet is a linear combination of L1 and L2 regularization, meaning it combines Ridge and LASSO and essentially splits the difference.	Transformer Methods Returns		
	LASSO	Ridge	ElasticNet	LASSO	Ridge	ElasticNet
'id'	2.752825742384942e-06]	2.768985932826149e-06]	2.655600750707135e-06]			
'accommodates'	16.298005233698582]	15.632328489854224]	15.539058398612748]	x	x	x
'host_response_rate'	-35.16153328241112]	-42.47624341222212]	-5.610031404959538]	x	x	x
'host_acceptance_rate'	-20.53873767963941]	-24.39368545889045]	-5.634094755346993]	x	x	x
'host_listings_count'	0.29325050896114835]	0.29083200508493283]	0.3062138278881312]	x		
'bathrooms'	38.74285585144712]	41.70725189548679]	11.574983628271516]	x	x	x
'bedrooms'	26.97048164932825]	27.397951877323226]	13.092572124126134]	x	x	x
'beds'	0.058020952811236176]	0.5289062664788309]	7.691414765163531]	x		x
'square_feet'	0.024932090757464817]	0.023780588908641664]	0.033252845516885086]	x		
'security_deposit'	0.04659531872483361]	0.04591030494619727]	0.054556023841733345]	x		
'cleaning_fee'	0.1369447761600625]	0.13228634902371187]	0.22424864394555319]	x		
'guests_included'	1.4339095815721788]	2.1695730259376003]	3.50497283485178]	x		
'extra_people'	0.10774258171749762]	0.09726030275074472]	0.1030350836716738]	x		
'minimum_nights'	-0.3874730416822602]	-0.4782389629670918]	-0.28119158238470825]	x		
'number_of_reviews'	0.005873518750461996]	0.03339473912396296]	-0.11136953169679621]	x		
'maximum_nights'	-7.0136329590674635e-06]	-6.7089631318171156e-06]	-6.422460906814133e-06]			
'availability_30'	0.7085697685587218]	0.6485241624288319]	1.092739248275227]	x		
'availability_60'	0.15146623298985212]	0.1789025516878528]	0.3759561353985393]	x		
'availability_90'	-0.051975493944668095]	-0.037305689030084516]	-0.38631958447427384]	x		
'availability_365'	-0.0724174176185342]	-0.07236835211489115]	-0.06963132772378088]	x		

Figure 4: LASSO, Ridge, and ElasticNet Driven Feature Selection Matrix

The complete list of features identified by the transformer methods is provided in Figure 5

LASSO Results								
host_response_rate'	host_acceptance_rate'	host_listings_count'	zipcode'	accommodates'	bathrooms'	bedrooms'	beds'	square_foot'
weekly_price'	monthly_price'	security_deposit'	cleaning_fee'	extra_people'	minimum_n'	availability_30'	availability_60'	availability_90'
availability_365'	number_of_reviews'	review_scores_rating'	review_scores_location'	review_scores_communication'	review_scores_location'	review_scores_value'		
reviews_per_month'								
Ridge Results								
host_response_rate'	latitude'	longitude'	bathrooms'	bedrooms'				
ElasticNet Results								
host_response_rate'	host_acceptance_rate'	accommodates'	bathrooms'	bedrooms'	beds'	review_scores_communication'	review_scores_location'	reviews_per_month'

Figure 5: Transformer Method Feature List

Although some agreement is shown between the three methods, the combination of weak correlations between price and certain key features along with the widely varying regression coefficients led us to some uncertainty with respect to which features we should actually choose for modeling, not to mention the identification of a minimum viable feature set. As a result, the outcome of our feature selection effort was the creation of several feature lists, which we used during our modeling efforts, as follows:

1. Full feature list -- 43 features
2. Modified full feature list, to remove target leakage (discussed later) -- 41 features
3. Numeric features -- 33 features
4. Host controlled features -- 20 features
5. Static listing features -- 13 features
6. Regularization derived features -- 13 features

Progressing from our feature selection effort, we pursued model selection, using the scikit-learn model cheat sheet, shown in Figure 6, as a starting point.

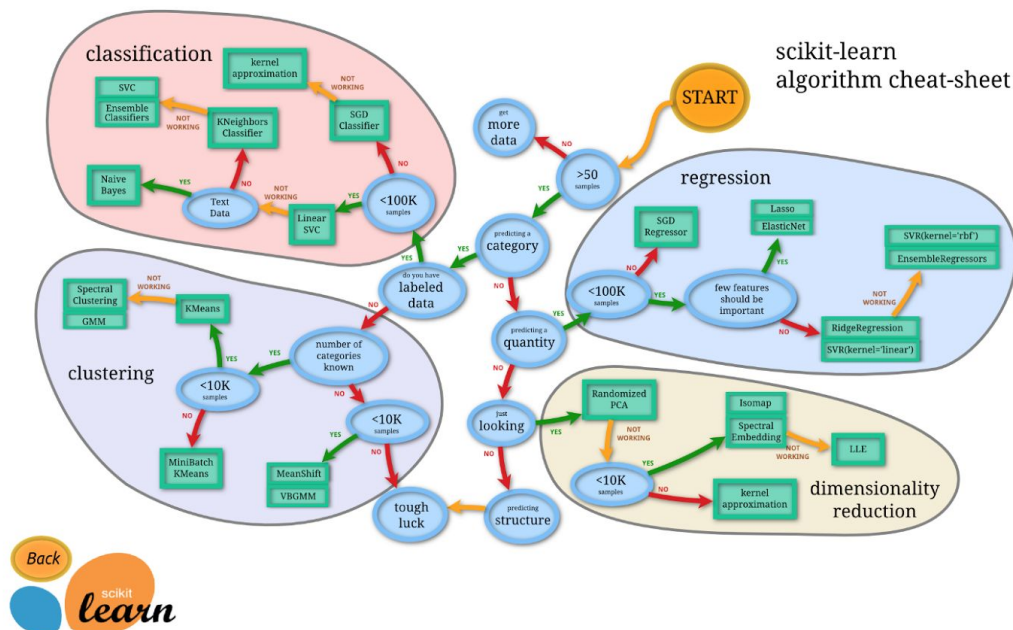


Figure 6: scikit-learn Model Cheat Sheet⁵

In concert with the model cheat sheet, we also depended on the regression model evaluation metrics, show in Figure 7 and as recommended by the scikit-learn documentation, to help us quantify model performance.

⁵ http://scikit-learn.org/stable/tutorial/machine_learning_map/

Classification metrics

See the [Classification metrics](#) section of the user guide for further details.

<code>metrics.accuracy_score(y_true, y_pred[, ...])</code>	Accuracy classification score.
<code>metrics.auc(x, y[, reorder])</code>	Compute Area Under the Curve (AUC) using the trapezoidal rule
<code>metrics.average_precision_score(y_true, y_score)</code>	Compute average precision (AP) from prediction scores
<code>metrics.brier_score_loss(y_true, y_prob[, ...])</code>	Compute the Brier score.
<code>metrics.classification_report(y_true, y_pred)</code>	Build a text report showing the main classification metrics
<code>metrics.confusion_matrix(y_true, y_pred[, ...])</code>	Compute confusion matrix to evaluate the accuracy of a classification
<code>metrics.f1_score(y_true, y_pred[, labels, ...])</code>	Compute the F1 score, also known as balanced F-score or F-measure
<code>metrics.fbeta_score(y_true, y_pred, beta[, ...])</code>	Compute the F-beta score
<code>metrics.hamming_loss(y_true, y_pred[, classes])</code>	Compute the average Hamming loss.
<code>metrics.hinge_loss(y_true, pred_decision[, ...])</code>	Average hinge loss (non-regularized)
<code>metrics.jaccard_similarity_score(y_true, y_pred)</code>	Jaccard similarity coefficient score
<code>metrics.log_loss(y_true, y_pred[, eps, ...])</code>	Log loss, aka logistic loss or cross-entropy loss.
<code>metrics.matthews_corrcoef(y_true, y_pred)</code>	Compute the Matthews correlation coefficient (MCC) for binary classes
<code>metrics.precision_recall_curve(y_true, ...)</code>	Compute precision-recall pairs for different probability thresholds
<code>metrics.precision_recall_fscore_support(...)</code>	Compute precision, recall, F-measure and support for each class
<code>metrics.precision_score(y_true, y_pred[, ...])</code>	Compute the precision
<code>metrics.recall_score(y_true, y_pred[, ...])</code>	Compute the recall
<code>metrics.roc_auc_score(y_true, y_score[, ...])</code>	Compute Area Under the Curve (AUC) from prediction scores
<code>metrics.roc_curve(y_true, y_score[, ...])</code>	Compute Receiver operating characteristic (ROC)
<code>metrics.zero_one_loss(y_true, y_pred[, ...])</code>	Zero-one classification loss.
<code>metrics.brier_score_loss(y_true, y_prob[, ...])</code>	Compute the Brier score.

Regression metrics

See the [Regression metrics](#) section of the user guide for further details.

<code>metrics.explained_variance_score(y_true, y_pred)</code>	Explained variance regression score function
<code>metrics.mean_absolute_error(y_true, y_pred)</code>	Mean absolute error regression loss
<code>metrics.mean_squared_error(y_true, y_pred[, ...])</code>	Mean squared error regression loss
<code>metrics.median_absolute_error(y_true, y_pred)</code>	Median absolute error regression loss
<code>metrics.r2_score(y_true, y_pred[, ...])</code>	R ² (coefficient of determination) regression score function.

Figure 7: Regression Model Performance Metrics⁶

Modeling and Application

Our baseline modeling efforts focused on building simple models we could use to verify our development efforts, begin to quantify performance, and develop a deeper understanding of our features. Unfortunately, our first pass through revealed a key error -- target leakage. Using R-squared as our basic measure of model comparison, our initial linear regression model showed suspiciously strong performance. What we quickly realized was that, by overlooking price per week and price per month, we had heavily influenced our model with information that is simply a derivation of the target. Figure 8 illustrates a comparison using the regularization derived features with and without the weekly and monthly price features, showing that R-squared decreases from 0.78 to 0.29 when removing just those two features.

⁶ http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

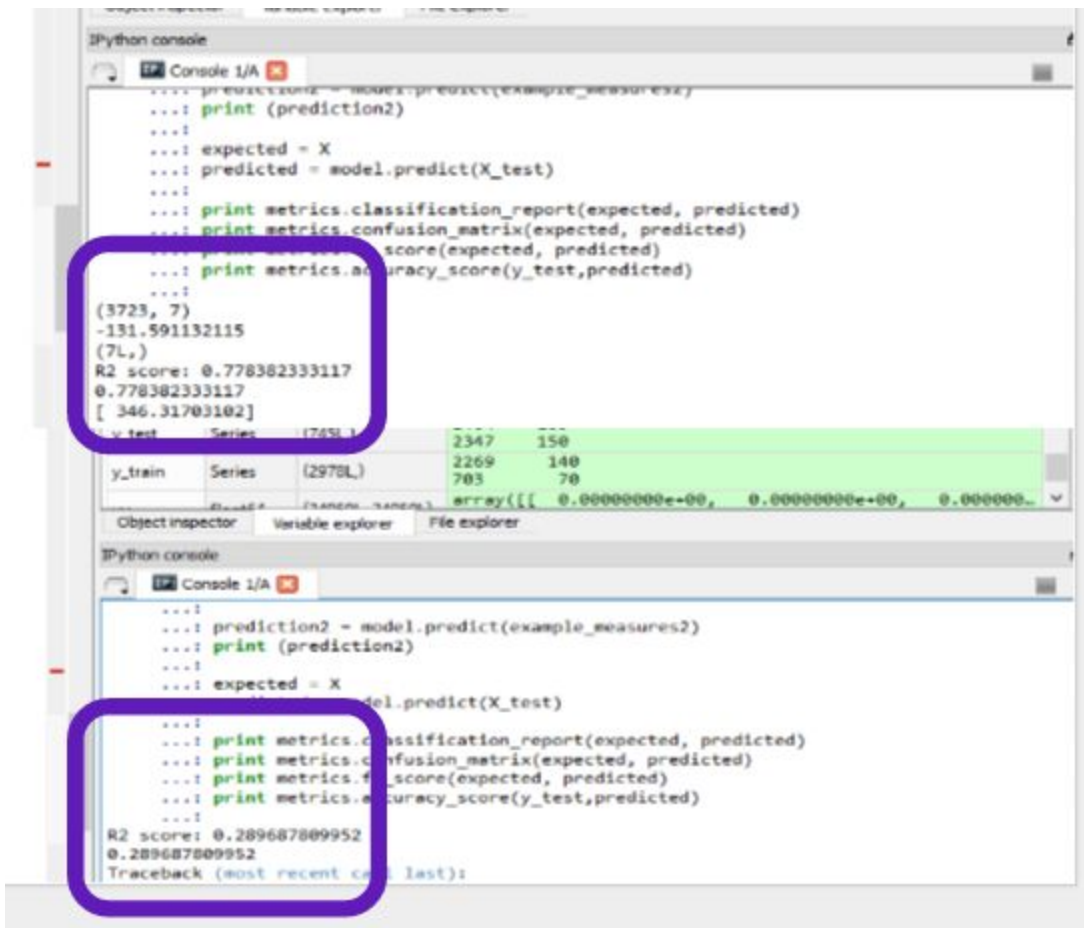


Figure 8: Model Comparison Revealing Target Leakage

Using the smaller feature set, corrected for target leakage, performance appeared to be poor. In response, we stepped up to a larger feature set, this time focusing on numeric features corrected for target leakage, and applied three regression models using demonstration code from class: linear, ridge, and random forest, using the Jupyter Notebook `airbnb_energy_efficiency.ipynb`, adapted from the energy efficiency notebook explored in class. As shown in Figure 9, R-squared performance improved but remained relatively low, with a somewhat limited change in performance across the three models.

```

model = Ridge(alpha=0.1)
model.fit(X_train, y_train)

expected = y_test
predicted = model.predict(X_test)

print "Ridge Regression model"
print "Mean Squared Error: %0.3f" % mse(expected, predicted)
print "Coefficient of Determination: %0.3f" % r2_score(expected, predicted)
print "R2 score = %0.3f" % r2_score(expected, predicted)

```

```

Ridge Regression model
Mean Squared Error: 6034.548
Coefficient of Determination: 0.428
R2 score = 0.428

```

```

model = LinearRegression()
model.fit(X_train, y_train)

expected = y_test
predicted = model.predict(X_test)

print "Linear Regression model"
print "Mean Squared Error: %0.3f" % mse(expected, predicted)
print "Coefficient of Determination: %0.3f" % r2_score(expected, predicted)
print "R2 score = %0.3f" % r2_score(expected, predicted)

```

```

Linear Regression model
Mean Squared Error: 6037.467
Coefficient of Determination: 0.427
R2 score = 0.427

```

```

model = RandomForestRegressor()
model.fit(X_train, y_train)

expected = y_test
predicted = model.predict(X_test)

print "Random Forest model"
print "Mean squared error = %0.3f" % mse(expected, predicted)
print "R2 score = %0.3f" % r2_score(expected, predicted)

```

```

Random Forest model
Mean squared error = 5169.077
R2 score = 0.510

```

Figure 9: Second Pass Regression Modeling Results, Corrected for Target Leakage

If we had stopped at this point, one explanation for the observed model performance could be that Airbnb pricing is more related to human behavior than true market behavior, making price a somewhat harder target to model. However, while human decision making certainly plays a part in pricing, the shape of the data would indicate that is not likely the most important factor. For

example, Figure 10 illustrates the shape of listing prices, showing a fairly smooth distribution with slight positive skew.

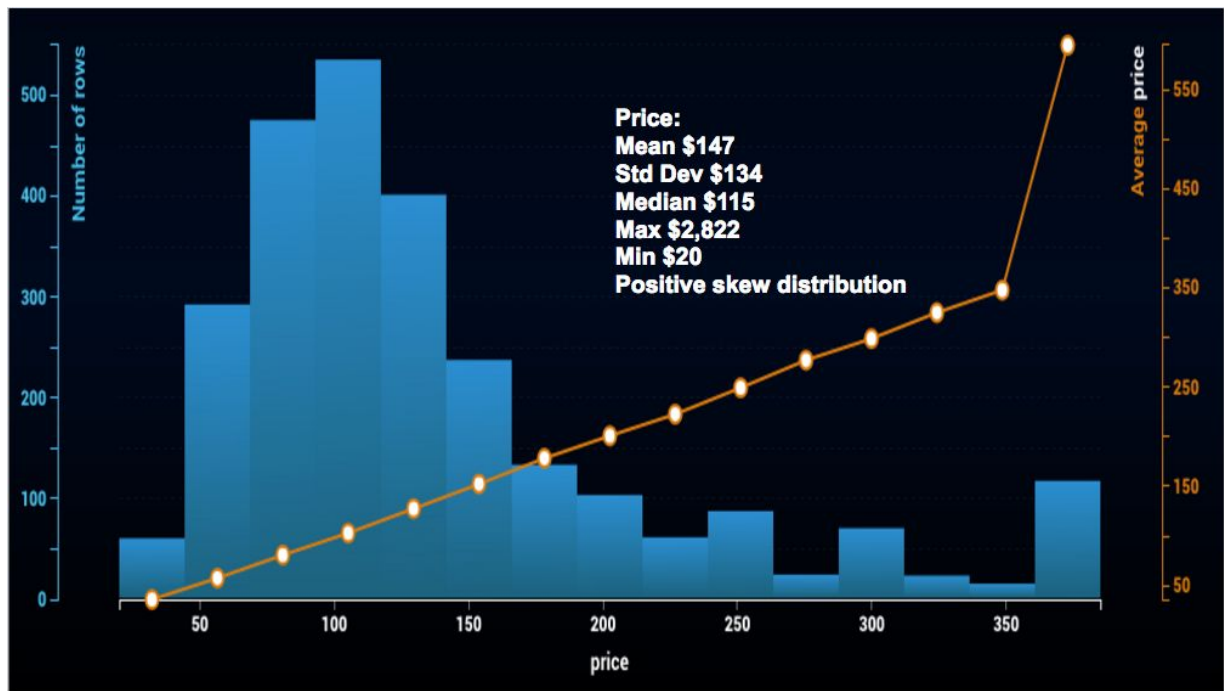


Figure 10: Listing Price Distribution

Our next logical step was to explore more sophisticated regression models while continuing to exercise our feature lists. Fortunately, we were able to access the DataRobot machine learning platform to help us accelerate our iteration process. Beginning with the full feature list, corrected for leakage, we explore model performance across a wide range of exotic models, again using R-squared in order to compare to our earlier efforts. Figure 11 shows the model performance as ranked by R-squared for model validation, in descending order.




Menu Search + Add New Model			Metric R Squared		
Model Name and Description	Feature List	Sample Size	Validation	Cross Validation	Hold Out
 eXtreme Gradient Boosted Trees Regressor BP114 M200 Tree-based Algorithm Preprocessing v1 Log Transformed Response	Price Isolation	64.01 % +	0.7245	0.5790	🔒
 eXtreme Gradient Boosted Trees Regressor (Gamma Loss) BP115 M207 Tree-based Algorithm Preprocessing v20 Log Transformed Response	Price Isolation	64.01 % +	0.7244	0.6006	🔒
 eXtreme Gradient Boosted Trees Regressor (Gamma Loss) BP350 M495 Tree-based Algorithm Preprocessing v20 Log Transformed Response	Numeric Only	64.01 % +	0.7240	0.5558	🔒

Figure 11: Modified Full Feature List Modeling and Comparison

Though R-squared performance is dramatically improved by using the full feature list and a relatively exotic gradient boosted tree model, we can see from the deviation between validation and cross validation performance that our model is likely overfit to the training data. If we instead sort of the models by cross validation performance, as shown in Figure 12, we can see more well matched performance between validation and cross validation (in this case we're performing five-fold cross validation to reduce computation time), though still some indication of overfit.




Menu Search + Add New Model			Metric R Squared		
Model Name and Description	Feature List	Sample Size	Validation	Cross Validation	Hold Out
 eXtreme Gradient Boosted Trees Regressor (Gamma Loss) BP115 M207 Tree-based Algorithm Preprocessing v20 Log Transformed Response	Price Isolation	64.01 % +	0.7244	0.6006	🔒
 RuleFit Regressor BP334 M492 Missing Values Imputed Log Transformed Response RuleFit Regressor Calibrate predictions	Numeric Only	64.01 % +	0.6723	0.5862	🔒
 eXtreme Gradient Boosted Trees Regressor (Gamma Loss) BP103 M206 Tree-based Algorithm Preprocessing v1	Price Isolation	64.01 % +	0.6808	0.5843	🔒

Figure 12: Model Comparison Based on Five-Fold Cross Validation

Attempts at investigating simpler models using the more sparse features lists we developed showed exceptionally poor performance. Figure 12a shows a snapshot of models using the regularization derived feature list, again sorted by descending order of cross-validation performance. We can see here that using smaller feature lists is probably not viable for creating a robust model.

Menu Search + Add New Model			Metric Gamma Deviance			
Model Name and Description		Feature List	Sample Size	Validation	Cross Validation	Hold Out
<div> <div>XGBoost</div> <div>BP67 M105</div> </div> <div>eXtreme Gradient Boosted Trees Regressor</div> <div>Tree-based Algorithm Preprocessing v1 Log Transformed Response</div>		TZ Feature List 1	64.01 % +	0.1246	0.1350	
<div> <div>RuleFit</div> <div>BP40 M111</div> </div> <div>RuleFit Regressor</div> <div>One-Hot Encoding Missing Values Imputed Log Transformed Response RuleFit Regressor Calibrate predictions</div>		TZ Feature List 1	64.01 % +	0.1335	0.1372	
<div> <div>RandomForest</div> <div>BP43 M109</div> </div> <div>RandomForest Regressor</div> <div>Tree-based Algorithm Preprocessing v1</div>		TZ Feature List 1	64.01 % +	0.1444	0.1377	

Figure 12a: Sparse Feature List Model Performance

Inspection of relative feature impact, shown in Figure 13, for our high performing gradient boosted trees regressor reveals that certain features one might subjectively select are indeed predictive of price, but that there are also key host behaviors that influence price, in particular host response rate, the cleaning fee (which is at the discretion of the host), the date of first review (indicating host experience), and several others.

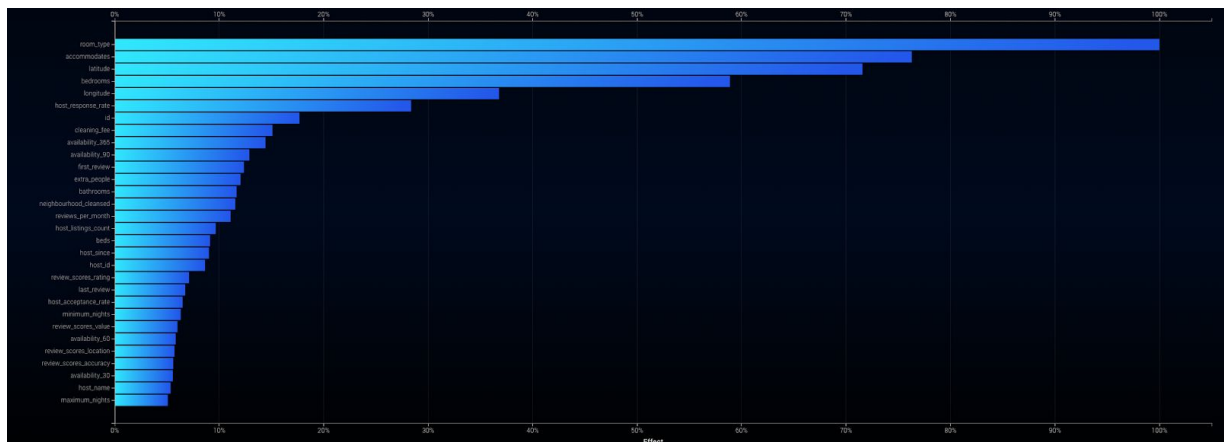


Figure 13: Relative Feature Impact For Gradient Boosted Trees Regressor

Unfortunately, by the time we reached this point in our efforts, the data hosted on Inside Airbnb changed, reducing from 92 features down to 16, removing many of the key features we identified as predictive of price and thwarting our desire to apply our models to other city data.

Reporting and Visualization

Despite some question about the ability of our models to predict listing price, we were successful in performing geographic modeling and visualization of listing data to help expand our understanding and provide more qualitative assistance to the prospective Airbnb host. Our

key efforts for geographic visualization were conducted using Tableau and focused on two features we expected to be important: 1) neighborhood and 2) proximity the Metro stations.

Figure 14 illustrates each one of our listing instances plotted on a map of D.C., using their respective latitude and longitude features, and colored based on neighborhood, as defined by the neighborhood listing feature.

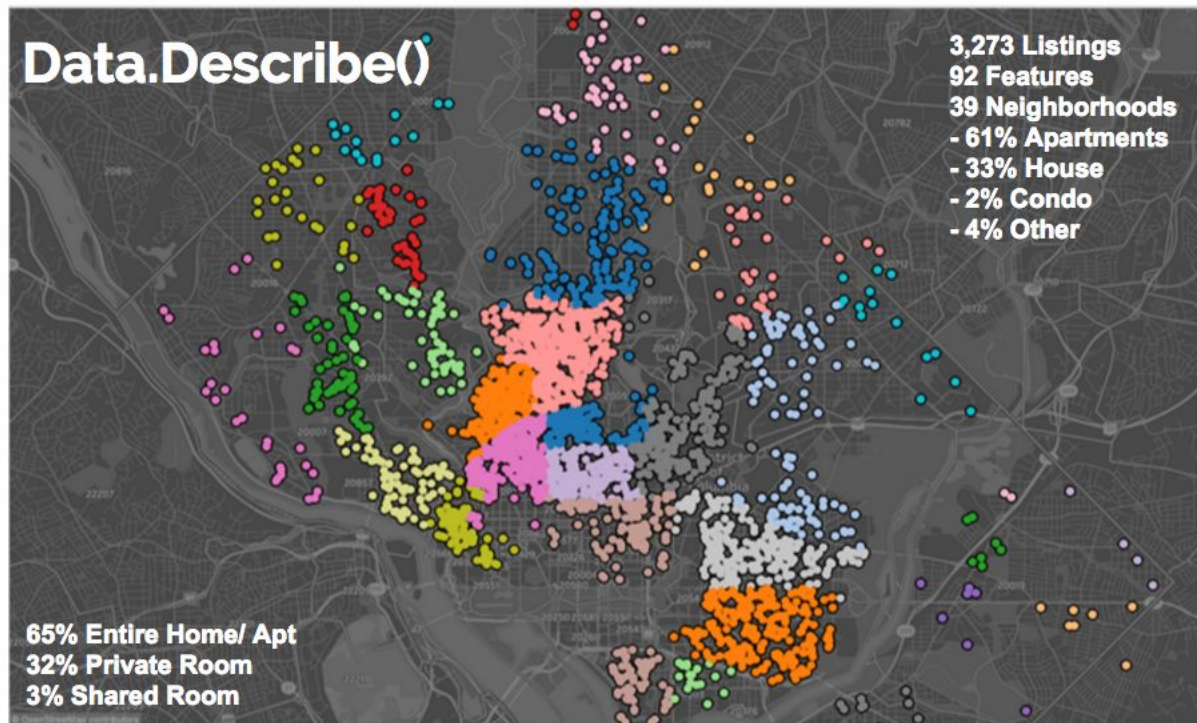


Figure 14: Listings Plotted by Neighborhood

Unsurprisingly, we can observe key clusters that indicate certain neighborhoods contain the preponderance of Airbnb listings. Perhaps more surprisingly, it appears that certain high cost neighborhoods, Georgetown in particular, have a much lower density of listings.

Pursuing this clustering further, we discovered that 80% of listings fall into only 13 neighborhoods, of a total 39 neighborhoods defined. Figure 15 illustrates the disparity, overlaid with Metro station locations. Not only can we observe that 33% of neighborhoods dominate the available listings, but that proximity to a Metro station may be worth an approximately 36% premium on price per night.

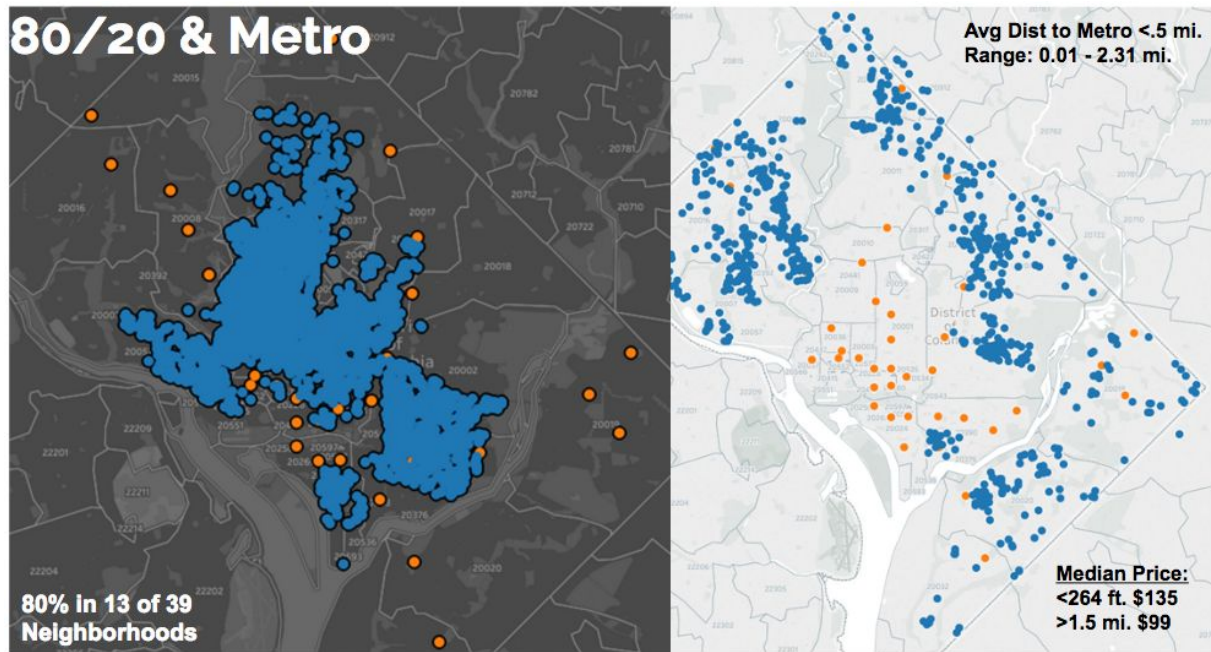


Figure 15: Neighborhood Concentration and Metro Station Proximity

At this point, we felt confident that, even without the ability to perfectly predict price, we could assist a potential Airbnb host in more accurately and strategically pricing their listing.

Quality of Methodology

Based on the dramatic improvement we achieved in our own understanding of Airbnb listings in Washington, D.C., we feel confident the overall architecture of our methodology is sound. The depth of iteration is not discussed in detail in this report, but our process was successful in identifying and correcting certain key errors, particularly related to feature selection and target leakage. In addition, there's a high degree of separability in our analysis, allowing key elements to be deployed independent of others, in particular the mapping visualizations, and/or against other city data.

The most significant improvements to methodology are likely to be enjoyed in modeling and application. Using R-squared as the singular metric for model performance is likely too rudimentary to draw robust conclusions about our models and were we to move forward a more sophisticated understanding of performance would likely need to be gained, potentially through improved model visualizations.

Conclusions

Summarizing the efforts presented in this report, the key findings we would communicate to a potential Washington, D.C. Airbnb host are as follows:

- Airbnb listings in D.C. are concentrated in a handful of neighborhoods, and along Metro lines, with a potentially substantial premium for proximity to a Metro station.
- Many of the features you would expect to affect price do indeed, but other things matter as well.
 - Unsurprisingly, people care about the number of beds and the value of a listing (cost per person), rather than simply absolute price.
 - Host behavior, in particular responsiveness, experience, and decisions about add-on fees, appear to play a role in allowing hosts to set their price.

A host seeking to maximize price would be well served to focus on customer service and broadcasting easy access to public transit. Adding a couple of air mattresses might also improve perceived value.

Areas for Future Exploration

During the course of our analysis, we revealed several key areas of future exploration. First, perhaps most interestingly, is the positive skew in listing price, around a median list price of \$115/night. If we compare to the Government Services Administration (GSA) fiscal year 2016 max lodging rate for hotels in Washington, D.C., we can see the GSA lists \$222/night as a lodging rate for October 2015,⁷ 93% above the median Airbnb listing price. Aside from the fact that a government traveler can probably save a lot of money staying at an Airbnb, we might hypothesize that the median Airbnb guest either 1) is income limited and therefore cannot select more expensive properties or 2) believes Airbnb to be a discount brand and thus does not select more expensive properties. While those hypotheses are not mutually exclusive, they do lead to very different corrective behavior and improved understanding of either should lead to dramatic improvements in host income based on increased listing prices.

Second, the strong geographic clustering of listings along with the calculated impact of Metro station proximity on listing price leads us to believe that other key geographic features may have a profound impact on price. Drawing in other data sources to expand public transit modeling (e.g., bus lines), or entertainment modeling (e.g., restaurants and landmarks) is likely to provide rich insight for hosts in the way they represent key and differentiating features of their listings.

⁷ <http://www.gsa.gov/portal/category/100120>