# Detecting Misleading Listings through Airbnb Reviews

Team Indubitably: Gregory Glatzer, Charlie Lu

# Table of Contents

# Misleading Listings through Airbnb Reviews

## 1. Introduction

In the presentation by Dr. Petersen about Airbnb review data, he explained that the Airbnb rating system is inflated, meaning that almost all reviews of both customers and properties are 4-5 stars. The reasoning behind this is because users want high ratings on their profile so that property owners are more likely to host them and vice versa. So if users leave a high rating on a listing then the host is likely to leave a high rating for the user, creating a symbiotic relationship where almost all reviews are rated 5 stars.

Due to this review number inflation, the “true” rating of a property is found in between the lines within the written review. Through these written reviews, one can learn if a listing is as good as it appears, and if there are discrepancies between what is claimed to be offered by a listing, and what is actually offered. For our capstone project, we aim to use a build a model to detect reviews that indicate a misleading property listing.

### Motivations, Benefits, and Problem Formulation (Problem and Goals)

One of the main motivations for this capstone project is providing an accurate and reliable source of information for users of Airbnb. Renting a property poses a large risk: If the property is not as advertised and someone rented for 10+ days they might end up in a very uncomfortable situation for the entire duration, or cost them large sums of money since they would have to move to a different location. An issue like this is caused by a misleading listing, for example, if a listing claims to have air conditioning but doesn’t and it is mid summer with temperatures in the 90s. This is the type of situation we want to avoid. For a problem like this a machine learning solution is very suitable given the quantity of data. Desired data for this approach is Airbnb review data, specifically reviews that indicate misleading listings. The procedure is to extract reviews that indicate misleading listings, and determine the best features that help predict the label of if a review is “misleading” (ie the corresponding listing is misleading), then construct a ML model that uses these features to predict from existing user reviews whether or not a listing has high potential to be misleading. The benefits of a solution like this is that the process is automated and can be future-proofed for new reviews, as well as applied to different geographical locations. Our final goal for the project is to apply the machine learning model trained on geographic locations in the training data, and examine how this model can perform on transfer data from a different geographic location.

NOTE: for the rest of this report, we will use the phrase “misleading review” as shorthand for a review that indicates that the corresponding listing was misleading compared to their experience. The review itself is not misleading.

# 2. Data

Our dataset consists of information related to Airbnb listings and reviews. For reviews, we have access to the review itself, the reviewer name, and a reference to which listing it is reviewing. For the listings, the dataset contains lots of information, such as number of summary, description, number of beds, availability, number of people accommodated, square footage, etc, per listing. With all this information, we decided to only consider the “description” and “amenities” columns, which together encapsulate lots of the other columns, and are probably the main parts of the listing that people consider. It is also where information that could eventually be misleading could be, such as saying it has a renovated kitchen when it does not, or the amenities listing TV that a review says is not present/working.

The dataset is split into three geographical locations: Austin, Texas; Fort Lauderdale, Florida; and San Francisco, California. The reason for these cities out of the many available cities with Airbnb data is that these cities have a very large quantity of listings. The first two datasets, Texas and Florida, will be used for our training and test data, while California will be used to evaluate the effectiveness of transfer learning for our model. The datasets were provided by Professor J. Andrew Petersen. This section includes our exploratory data analysis, as well as our labeling methodology.

## 2.1 Exploratory Data Analysis (EDA)

First let us look at the sizes of the datasets. Table 1 shows the respective sizes of these datasets.

| **Airbnb Geo** | **# reviews** | **# potential Misleading** | **% potential Misleading** |
|---|---|---|---|
| Austin, Texas | 332,098 | 1,063 | 0.288 |
| Fort Lauderdale, Florida | 195,857 | 1,065 | 0.545 |
| San Francisco, CA | 366,643 | 1,075 | 0.296 |

Table 1. Size of the data, per geography.

Initially when we looked at the data there were three fields that stood out in particular, description of property, reviews, and advertised amenities. To get an idea of how many misleading reviews there are, we first took a very crude approach during our EDA phase: How many reviews are there that include a misleading keyword (such as “missing”, “not as described”, etc.), and/or a negative keyword (such as “shouldn’t”, “hadn’t”), and also mentions an amenity that was listed as being included with the rental? For the full list of misleading and negative words used in this technique, see the Appendix “Misleading and Negative Keywords, and Common Phrases”.

To answer this question, we can intelligently filter the data based on the presence of certain keywords, which were based on the keywords identified in Devarakonda, 2022. Our list of keywords, which can be found in Section 2.2.2, is an extension of the keywords listed in Devarakonda, 2022. After removing stop words using NLTK

(excluding negative stop words), we obtain a set of unique tokens for each review. We then filter the reviews down to only ones that have at least one listed amenity in the set of review tokens (i.e. a customer mentioned an amenity in their review). This could either be a mention of a discrepancy or positive praise, for example “There is no TV” and “There was a great TV”. To remedy this, we further filter down to only include reviews with both a negative keyword and a misleading word. By doing this we can see how many reviews mention an amenity as well as a misleading/negation word that could indicate an inconsistency between the listing and the review. This gives us an idea of how many inconsistent reviews we could potentially have. For Florida, this ended up being only 0.545% of the dataset. For Texas, it was 0.288%. This is a very small amount of reviews possibly indicating misleading information but that falls within our expectations.

## 2.2 Labeling Methodology

### 2.2.1 Labels, and nuance in labeling

The goal of our model is to identify misleading reviews, but oftentimes the phrases and sentences that indicate a misleading review are nested within a negative review. This poses a major issue: while misleading reviews might indicate negative reviews, a negative review does not necessarily indicate a misleading review. If the model flags reviews as misleading simply because they are negative we will have a major problem. To solve this we need to distinguish a negative review from a misleading one. This leads us to our final labeling methodology, which consists of 6 labels:

- “MBad”: misleading and a bad review
- “Bad”: Bad experience but not misleading
- “MGood”: misleading and a good review
- “Good”: Good experience but not misleading
- “Maybe” *
- “Other” *

*see following paragraph

There are 2 more labels which are for special cases: “Maybe” and “Other”. “Maybe” is a label for potentially misleading reviews, because the quantity of misleading reviews is small, this label serves as a flag for reviews that have potential to be misleading and therefore can be added to the misleading category in the future if the quantity is insufficient. The final label - “Other” - is for reviews that are either not english, or for whatever reason the characters in the review get messed up in the text file encoding that the review is unreadable (we do not find many of these). They represent outliers that will not be considered during training.

The task of deciding these labels is not always straightforward, which makes this a challenging task for making training data, as well as training a model. There are several different kinds of “Bad” reviews, each of which have varying degrees of how much you need to “read between the lines” to understand if the listing is misleading, or if there was something else that made it a negative experience.

With the subtlety in types of reviews, it was challenging to have consistency between the labellers (2 of us). While we did not calculate an inter-labeler consistency measure, we did discuss what is considered misleading or not as we worked through the labeling process. From reading through reviews to generate labels, we noticed several common themes on how people complain/list cons of their stay at an Airbnb listing:

- Unacceptable utilities (A/C unit is broken/bad, bad WiFi, etc)
- Missing utilities/features (wrong size bed, no pool, no TV, etc.)
- Dirty and/or unsafe property (cockroaches, dirt, hair, doors don't lock, etc.)
- Complaints about the neighborhood (homelessness, feeling of safety, etc.)
- Complaints about neighbors (loud parties, nosey neighbors, etc.)
- Personal preferences not met (couches lower/harder than they like, niche preferences not met, etc.)

A missing utility almost always means a misleading listing that did not deliver on its promises. This observation motivated the development of the "Amenity Inclusion" feature, described in Section 3.2.3. The other types of complaints, however, are hard to always categorize as "misleading". Is a complaint about the loud neighbors misleading? Only if the property explicitly described a quiet neighborhood. These other categories of complaints involve more nuance and more context, it becomes a spot the difference between review and listing description. To give the mode the context of the review and listing description, we decided to create word embeddings of these texts as features, as described in Section 3.2.2.

### 2.2.2 Subsetting the data for labeling

Manually labeling all of the training, testing, and transfer (California) datasets would be infeasible due to the size. To give us a smaller subset of the data to label that will hopefully contain a good amount of misleading reviews, we chose to filter the data using (1) keywords and (2) sentiment scores.

(1) **Subsetting on keywords**. Our list of keywords is an extension of the list of keywords described in the Penn State Schreyer Honors Thesis on finding misleading Airbnb listings by Vaishali Devarakonda (Devarakonda, 2022).We identified additional keywords by first applying her list, and then while reading through the filtered reviews, we found common appearing phrases that were similar to our existing list. The goal of these keywords is to find reviews that may be talking about misleading listings. If any of these keywords are in a review, we include it in our subset to label. See the list of misleading words in Appendix "Misleading and Negative Keywords, and Common Phrases" for our full list of keywords.

(2) **Subsetting on sentiment scores.** For generating sentiment scores, we used the BERT sentiment model "nlptown/bert-base-multilingual-uncased-sentiment" from the *transformers* Python library (Jacob et. al, 2018). BERT can only process a maximum of 512 characters at a time, so for reviews bigger than 512 characters, we split the review into multiple substrings, applied BERT to each, and then averaged the scores. BERT returns a sentiment score of 1 (overly negative) to 5 (overly positive). For reviews over 512 characters, the averaged sentiment score was often decimal. Applying this model to all three datasets took several days of processing. To speed this up, we divided this task

across both of our personal computers. This expensive computational task, however, allowed us to intelligently create a new sentiment label to assist us in scaling down the data. Once we have our sentiment scores, we decided to include any reviews with a score below, but not equal to, 5. As illustrated in Figure 1, this excludes the majority of the reviews, which are overwhelmingly positive.

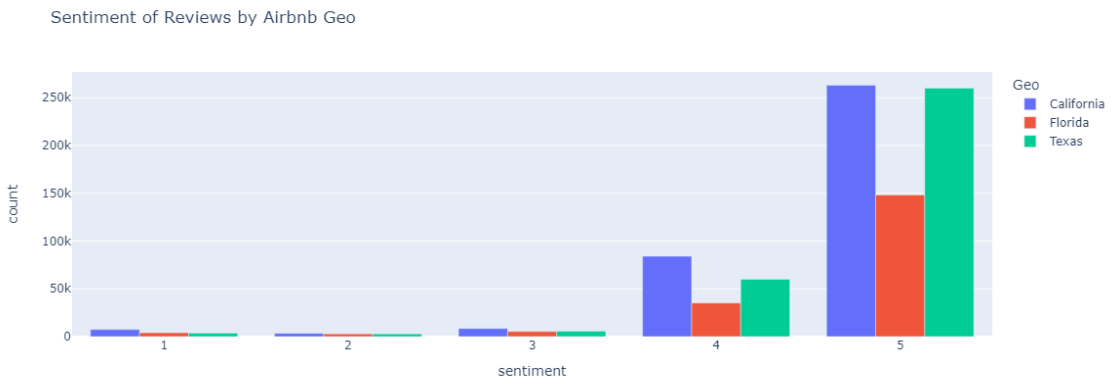



Figure 1. Distribution of sentiment scores for all three datasets. The majority of the scores are 5.

After applying both of these filters - keywords and sentiment scores - we generated the subsets of the three datasets that we needed to manually label. The reasoning behind this decision is that a sentiment of 5 which means an overwhelmingly positive review likely does not contain any contradictory information and thus can be ignored for the purposes of our project goal.

### 2.2.3 The labeling interface

With several thousands reviews to label, we needed a streamlined way of reading the reviews, choosing the label, and saving our work. To do so, we used the Python library tortus, a “python package that makes it easy to add labels to text data within a Jupyter Notebook”. This library allowed us to easily label our combined dataset of reviews and listing descriptions. Once we had our dataset prepared to be labeled, we could generate a dynamic html template that would be displayed for us to review, along with buttons to select a label, as well as a Skip button, if necessary. Figure 2 shows this labeling setup.

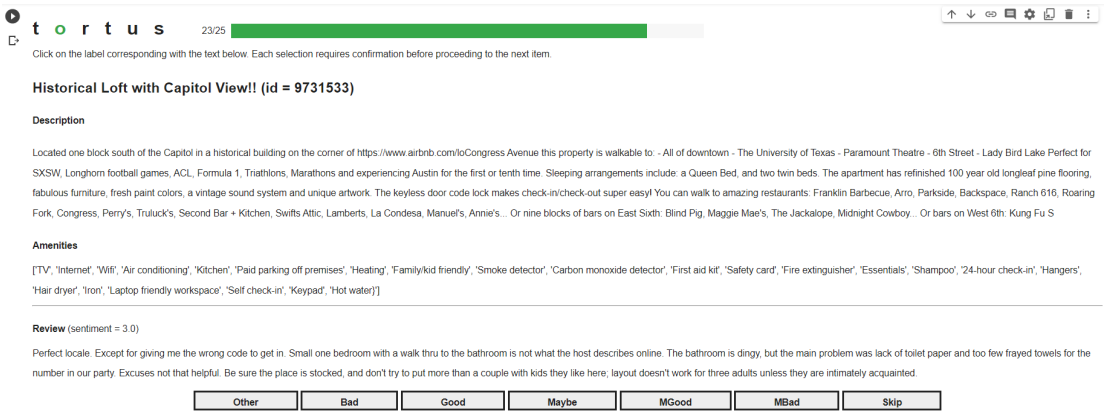



Figure 2. The tortus interface for labeling data, displayed in a Jupyter notebook.

### 2.2.4 Automatic Labeling

After labeling a combined 1700+ reviews for the Texas and Florida datasets, we only had about 200 labels that were misleading. To supplement the low number of misleading labels, we created a technique to automatically label more reviews that undoubtedly indicate a misleading listing. To automate the labeling process we created a basic list of words and phrases, which if present, always indicate a misleading review. For example ['not as pictured', 'is very misleading']. The full list of phrases can be found in Appendix "Automatic Labeling Phrases". These types of phrases are used to filter reviews. Since the misleading labels are divided into misleading bad and misleading good, the filtered reviews need to be separated into a good and bad category. This is done through the sentiment scores obtained earlier. If the sentiment score is less than or equal to 3, the assigned label is misleading bad, otherwise the label is misleading good. When comparing the manually labeled reviews with the automatically labeled reviews, the accuracy is almost 100%. This gives us a good way to obtain reviews that are most certainly misleading, while our manually labeling handles the more subtle fringe cases.

### 2.2.5 Distribution of Labels

With a combination of our manually labeled data and the automatic labels, we come up with 14.88% of our training data containing misleading labels (either Misleading Good (mgood) or Misleading Bad (mbad)). Figure 3 shows the distribution of labels.

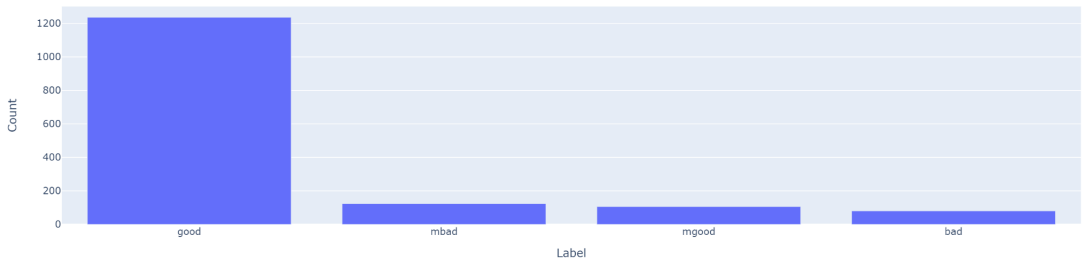



Figure 3. Distribution of labels in training data (Texas and Florida).

# 3. Data Preprocessing and Feature Engineering (Preprocessing and Design)

## 3.1 Data Preprocessing

Raw text can provide several challenges when it comes to processing the data for machine learning consumption. The field of Natural Language Processing (NLP) addresses the complexity of natural language, and how to best represent human language in numbers. A common NLP data processing pipeline usually consists of lowercasing letters, removing punctuation and stopwords, and lemmatization and/or stemming. Our data preprocessing pipeline for text fields is the following:

1. Make all text lowercase
2. Tokenize words (using nltk's RegexpTokenizer)
3. Remove stopwords (using nltk's wordnet stopwords, excluding negative stopwords)
4. Lemmatize text (using nltk's WordNetLemmatizer)

This pipeline was applied to the reviews, as well as the name and description fields of the listing.

For our task of identifying misleading reviews, we must carefully consider stopword removal in particular. In the context of the problem, negative stopwords, such as "doesn't", "won't", "not", "hasn't", etc. can have an important impact on the interpretation of the review. For example, if someone says "I would definitely recommend this place to my friends" versus "I would definitely not recommend this place to my friends", the meaning switches. Most importantly, we have seen several reviews saying "the place is as pictured" (not misleading) versus "the place is not as pictured" (misleading).

## 3.2 Feature Engineering

We tried a variety of feature engineering techniques for extracting out what we thought would be important features for identifying misleading reviews. In this section, we will examine these various techniques, and elaborate on what worked, what didn't, and why we believe this is so.

### 3.2.1 N-Grams Feature

Reviews may have certain phrases, or groups of phrases, that are common across certain types of reviews. Examples could be "would not recommend", "was very misleading", or "not as pictured" for 3-word phrases. This technique of grouping words together into one unit is often called generating "N-grams". For this feature, we looked at various size N-grams to see which value for N yielded the most unique N-grams for misleading reviews. Figure 4 illustrates the number of unique n-grams for different labels in the Texas dataset for different values of N. In this figure we exclude "good" reviews, as due to the sheer number of good reviews, there are a huge (40,000+) number of unique n-grams for this label, regardless of the value of N.

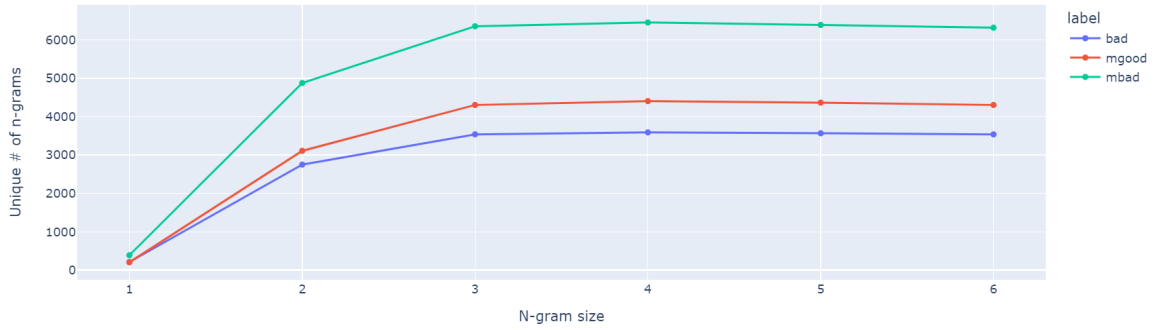



Figure 4. Unique number of texas n-grams for each label (excluding “good” labels, which have a huge number of unique n-grams due to the number of good reviews).

This chart shows that after N=3, there is not much difference in the number of unique n-grams for any label. Because of this, we believe N=3 may be an appropriate size N-gram.

After generating these N-grams, one-hot encoding them into features, and trying to train a classifier on them, we found that this was not a very good feature. There are many N-grams that appear in all types of reviews, such as “within walking distance”, that have nothing to do with indicating a misleading listing. While there are N-grams unique to misleading reviews, such as “not as pictured”, there are many N-grams that exist across all types of reviews. Without a method to distinguish which N-grams represent misleading reviews and isolate them, there is simply too much noise and obtaining a correct classification is too difficult with this feature alone. We later found that less than 50% of the top performing feature sets ever contain this feature, which indicates this this feature is not very useful.

### 3.2.2 Word Embeddings Feature

The next (and currently most successful) feature we created is a vector representation of each review and property description. This was done using the *gensim* Python package’s implementation of Word2Vec. Word2Vec generates a vector representation of each word in the body of text. By averaging all the vectors in one review or property description, we can obtain a single vector representation for that body of text. The parameters for the Word2Vec model are as follows: min count of 1, a window of 3, and a skip-gram algorithm. We have not extensively tested different parameters, however these appear to be fairly common parameters for Word2Vec. For our embedding vector size, we try 50, 100, 200, 300, 500, and 1000.

A large word embedding vector size allows the embedding to capture more context in the text. To be able to keep this high dimension word embedding, but still tackle the problem of high dimensionality, we then apply Principal Component Analysis (PCA) to the embeddings. We consider the target dimension as a design parameter, trying a variety of target dimensions, including 2, 3, 5, 10, and 20. This addresses the high-dimensionality problem. We have also found that this step of abstracting the word embeddings helps reduce overfitting.

### 3.2.3 Amenity Inclusion Feature

In each property listing, there is a set of amenities that are advertised as being included in the property. Examples of amenities include TV, extra linens, towels, pool, washer/dryer, gated property, etc. In a review, an amenity is either mentioned if it is included/exceptional, or not included, i.e. the listing was misleading in claiming it was included and/or functional. For each listed amenity, we generate a binary feature that determines if the amenity was mentioned in the review. This includes some basic cleaning of amenities so they are more easily matched in the review text. This cleaning includes removing quotation marks, text in parenthesis or brackets, and lowercasing all text. We also split up amenities with a slash (/), “and”, or “or” in them into multiple amenities. Examples of this include “washer/dryer”, “cats and dogs”, etc. It appears that the amenity list is a controlled field for Airbnb hosts, as the amenities are listed the same way in every listing (no typos, TV vs. television, etc.). A difficulty in matching amenities to their review counterparts, however, is that the reviewer is free to call a “TV” a “television”, or a “washer/dryer” combo a “washing and drying machine”, etc.

Once we calculated these features, we calculated the amenities with the highest correlation to the label. The top 5 amenities based on correlation to the label are shown in Table 3.

| Amenity | Correlation with label |
|---|---|
| lockbox | 0.119924 |
| dishwasher | 0.091862 |
| pool | 0.084724 |
| stove | 0.075858 |
| keypad | 0.071675 |

Table 3. Top 5 amenities with respect to correlation with the label of if a review indicates a misleading listing.

Without any subsetting of the amenities, we get 33 additional features from this technique. We tried to limit this (as low-correlated amenities may simply be noise) by only including amenity features that are above a certain correlation threshold with the label. We tried to include amenities above a 5% correlation, as well as including all amenities. The results of these amenity subsets are explored in 4.2.

# 4. Model Training

## 4.1 Dealing with class imbalance

One of the difficulties of working with this data is that the percentage of misleading reviews is extremely small, less than 1% of all reviews (from original raw data, as seen in EDA). You can see the labeling distribution in Figure 3. Due to this, if a trained model

were to never label a misleading review it could theoretically still achieve very high accuracy which is great on paper, but ultimately fails in achieving the goal of finding misleading listings. There are several approaches to remedy this, such as oversampling (making additional instances of the minority class using nearest neighbor-driven techniques), or undersampling (picking only the most representative samples from the majority classes). Currently, we are using the *imblearn* package's implementation of the random oversampling to supplement for the small number of misleading reviews. We were originally using SMOTE (Devlin et al., 2018), which is a synthetic oversampling technique, however this resulted in models which were more severely overfit on the training data and did not offer any improvements in any categories over training with no synthetic data.

## 4.2 Evaluation on different feature sets

To find the best model, we tried a combination of the different features we engineered to see which yields the highest performance. We calculated our features using our combined labeled data from both the Austin, Texas and Fort Lauderdale, Florida datasets. We used accuracy to measure model performance. The different feature combination types we tried are:

- "**Amenities and PCA**": Amenities Inclusion + PCA of review embeddings and/or PCA of description embeddings
- "**Amenities and embeddings**": Amenities Inclusion + review embeddings and/or description embeddings (no PCA)
- "**PCA over amenities and embeddings**": PCA of combined list of amenities and review embeddings + PCA of combined list of amenities and description embeddings

The goal of these different combinations is to give us an idea of how PCA affects the performance, as well as if PCA should be applied on the embeddings alone, or over a combination feature space of both the embeddings and list of Amenity Inclusion features.

Within each combination of features, we tried different parameters for the PCA model, as well as whether to have all Amenity Inclusion features, or only Amenity Inclusion features that are above a 5% correlation with the label (as described in Section 3.2.3). For example, we may try the following two set of feature combinations of type "Amenities and PCA", with respective parameters:

**(1) Amenities and PCA:**
amenities (amenities) with parameters {'corr_thresh': 0.05 }
pca (comments) with parameters: {'n_components': 3, 'vector_size': 50},
pca (description) with parameters: {'n_components': 3, 'vector_size': 50}

**(2) Amenities and PCA:**
amenities (amenities) with parameters {'corr_thresh': None }
pca (comments) with parameters: {'n_components': 3, 'vector_size': 50},

pca (description) with parameters: {'n_components': 5, 'vector_size': 100}

**(3) and so on …**

Notice the change in parameters for the PCA (description) model in the second set of features, and the amenity correlation threshold of none (meaning include all amenities, regardless of correlation with the label). Once we try all of the “Amenities and PCA” feature sets with the different parameters we want to try, we then calculate the accuracy of each feature set. This is done for “Amenities and embeddings” and “PCA over amenities and embeddings” feature sets as well, until we have exhausted the search. The accuracy of a KNearestNeighbors classifier is evaluated by being trained on all these feature sets, and the best feature set, with the best parameters for the respective PCA models and Amenity Inclusion correlation threshold, is chosen. Figure 5 shows the best accuracy for each feature combination type, resulting in “Amenities and PCA” with e best performance.

Specifically, our best feature set includes Amenity Inclusion features above a 5% correlation threshold, and both PCAs for comments and description embedding a Word2Vec model with a vector size of 50 down to 5 PCA components.

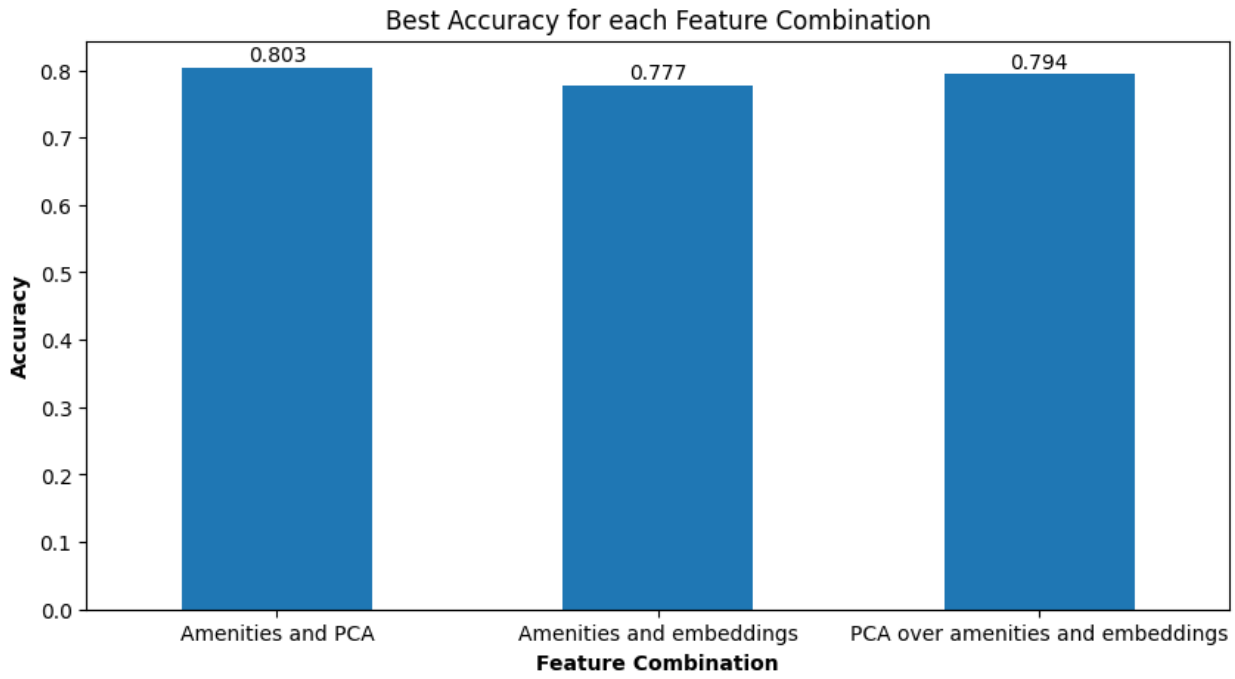


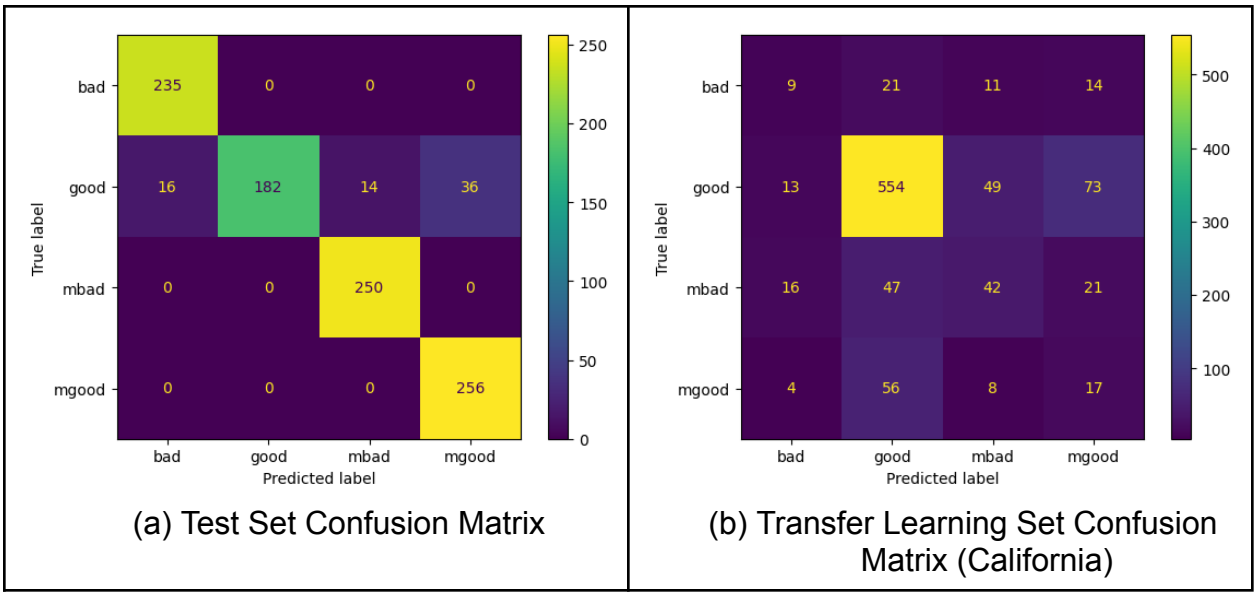

Figure 5. Best accuracy for each feature combination type using a KNN classifier with *leaf size* = *10* and *# neighbors* = *3*. The top feature set comes from Amenities and PCA.

## 4.3 Top Model and Demo

Using our best feature set, we trained several models on the Texas and Florida datasets. Using random oversampling to oversample, and then running a Grid Search CV with 5 folds, we were able to produce a model with 93.33% accuracy on the test set. Figure 6(a) shows the confusion matrix for the test set. We see that this model still struggles with labels that are “good”. This makes sense, since good labels are very prevalent, and what distinguishes good reviews from other reviews can be very subtle, such as a couple words or sentences in an otherwise positive review.

This model performs with 53.93% accuracy on the California transfer learning set. The confusion matrix for this model is shown in Figure 6(b). From this matrix we see that there is no strong preference for a particular label when evaluated on California. However, it is not good it correctly identifying the “bad” or “mgood” labels.

To test out this model against our own reviews and descriptions that we want to write ourselves, we used the Python package “streamlit” to create a webapp where we can write out a description, review, and amenities list and get a classification. This demo is shown in Figure 7.




(a) Test Set Confusion Matrix

(b) Transfer Learning Set Confusion Matrix (California)

Figure 6. Confusion Matrices for our current top model (KNeighborsClassifier) on test and transfer learning sets. It struggles with truly “good” reviews.

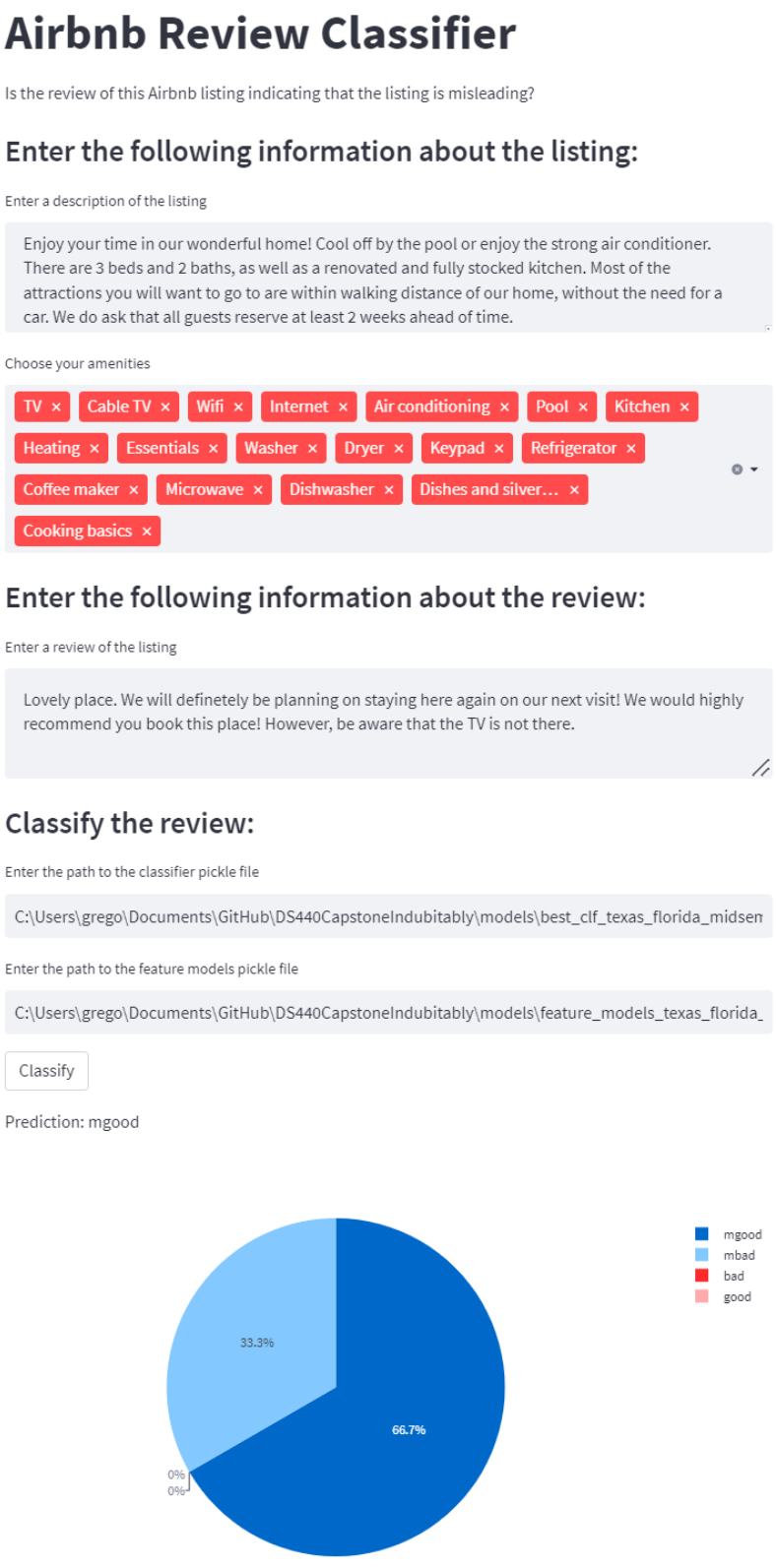



Figure 7. Our current demo application to test unseen values.

# 5. Discussion

Given the complex nature of this project and the subtlety of the problem, a multitude of issues is almost guaranteed to arise. This section will cover the most significant of these, as well as the limitations of our current approach, what we learned from the project, and insights for future human designers.

## 5.1 Challenges

The first major challenge was encountered during the labeling process. The initial labeling strategy was to focus only on misleading reviews and ignore all others. This means that originally there were only two labels, “Yes” for misleading, and “No” for all other reviews. This posed an issue as there was a significant amount of information left on the table that could aid in distinguishing between misleading reviews and non-misleading reviews. If the data was going to be manually reviewed already, then there was no reason to purposefully leave out potentially valuable information, which leads us to the label set we have now - good, bad, good but indicating misleading, and bad but indicating misleading reviews. This problem was caused by poor planning in the initial stage of the project. Luckily, it was not a large task to resolve as the issue was caught early on in the labeling process. Simply relabeling the existing labels was enough.

The second challenge encountered was the fact that there was not enough misleading data to sufficiently train a model after our initial labeling sessions. Given the tedious nature of manually labeling and factoring in human error, and the fact that the standards for what categorized as a misleading label shifted as time passed, there was not a rigid standard that was consistent throughout the labeling process. Since this was the case, a couple strategies were implemented to provide additional labels/address the imbalance data problem. The first was the use of oversampling techniques (See Section 4.1). The second (See Section 2.2.4) was through a labeling algorithm that automatically labels misleading reviews based on a word bank that contained words that if present always imply a misleading listing, and the sentiment score that would sort the review into the good or bad category. This solution generated over 300 extra misleading reviews that were consistent with our standards.

The third and most significant challenge was the models were overfitting. When training on a RandomForest model, training accuracy reached nearly 100% - even for different hyperparameters - which indicates that the model was simply memorizing the misleading reviews. This was causing abysmal testing accuracy, labeling reviews “good” almost 100% of the time. This problem was discovered when evaluating on never-before-seen user-generated reviews where it was discovered that any review entered would result in a “good” label, no matter how negative it was. We fixed this by replacing our original dimensionality reduction technique (t-SNE) with our current technique (PCA), as well as trying different classifiers that did not overfit. We found that many tree-based classifiers (ExtraTreesClassifier, RandomForestClassifier, ExtraTreeClassifier), which we were originally using, were overfitting. We also tried some simpler models, such as LogisticRegression, that severely underperformed (<40% accuracy). These modifications to our methodology prevented a 100% training accuracy and allowed our current model to make predictions on unseen data.

## 5.2 Design Choices

There are several design choices we made that have introduced some limitations to our model, as well as makes the performance of the model harder to evaluate.

One such limitation is our choice to subset the data for labeling using sentiment analysis. By subsetting the data to label to only reviews that are below a sentiment score of 5, we have less work for us as labelers, however there is a tradeoff of there potentially being more false positive “good” labels solely based off of the sentiment score generated by BERT. This “chaining” of models means that our classifier is inherently influenced by the performance of BERT in correctly assigning sentiment scores. This dependence on multiple models’ performance makes the overall performance of our classifier harder to understand, but acts as a tradeoff on how much labeling there is to do.

## 5.3 Varying Model Tradeoffs

While higher accuracy is usually a sign of a better model, in the case of this project, there are some additional metrics that need to be accounted for. Because of the imbalanced nature of the training data, if a model were to predict the majority label (good) 100% of the time, accuracy would still be respectable but contribute nothing towards solving the problem. Therefore the measurement of model performance should be weighted heavily on how well it performs on the misleading label.

This creates a trade-off between whether or not a model is giving mostly false negatives or mostly false positives. In the context of the problem, this means that either a listing is misleading and the model predicts that the listing is not misleading. Or a listing is not misleading but the model says that it is misleading, respectively. Based on these two interpretations, a model which provides mostly false negatives is preferred over a model which gives mostly false positives. The reasoning for this is because from the perspective of a customer, it is preferred to have a listing labeled as misleading, prompting them to look into the reviews themselves, versus a listing labeled as not misleading but in reality is misleading, which could result in a customer renting it out without being prompted to research further. The prior is the preferred result. Therefore a model which provides more false negatives is valued higher than one which provides more false positives, even though the models which provide more false majority labels (good) results in higher overall accuracies due to the imbalance of data.

## 5.4 Lessons Learned

Throughout this project, we learned several lessons, especially about considering design decisions which can affect where the project is going. These design considerations were made in all steps of the project, from deciding on labels to model training:

- Take time to consider what your labels should be. The labeling task turned out to be much more nuanced and complex than we thought, which affected how to best build a model, and how to interpret the results.
- Deciding what feature engineering you want to do and what subset of the data's columns you will use go hand-in-hand. Especially when working on a team of multiple data scientists, thinking several steps ahead helps you not need to backtrack as much later on in the project.
- If you are building a ML model/pipeline for production/deployment, consider how you are building your features and processing the data. Tools such as sklearn's pipeline can help in reproducing transformations of the data, especially as you look towards deploying the model on data that may look very different than the training data.

## 5.5 Insights for Humans

Model complexity has a large role on accuracy of predictions on the test set. Training strategies used to combat an overfitting model included reducing the dimensions of features through PCA, however applying PCA across all features did not produce the desired results. Only through applying dimension reduction on certain features produced improved results. Closely related features, such as our combination of Amenity Inclusion and Embedding features, can be combined in a PCA application. Domain knowledge and deep understanding of the problem can help in recognizing these closely related features.

With respect to detecting misleading Airbnb listings, context matters. Our attempts at creating features based solely on the existence of amenities, let alone, N-grams, in reviews/descriptions was not nearly as successful as word embeddings. This leads us to believe that the greater context of words in a sentence is more telling of if a review is talking about a listing being misleading, versus the binary presence of specific words/N-grams. We did not find any specific words or phrases that were strong indicators of misleading listings.

The subtlety of the difference between a misleading review and a non-misleading review can be VERY small. We found that most often the part of the review that indicates something being misleading is most often very short phrases, often hidden in large swathes of positive review. Of course there are exceptions, however in a dataset dominated by "good" reviews, this insight is important.

# 6. Conclusions

## 6.1 Final project goal reflection

This section is dedicated to reviewing the goals proposed in the mid-semester report and evaluating their status. We had 3 main goals: (1) improve our current model's performance (previously 88% test accuracy), (2) evaluate the best model on the California dataset, and (3) use our final model to rank listings based on a "misleading score" derived from our model's predictions across the listing's respective reviews.

(1), Our current top model is a KNeighborsClassifier, our new test accuracy is 93.33% beating the previous 88% by a significant margin.

(2) The California dataset has been manually and automatically labeled and predicted using the training data from Texas and Florida using our best model KNN with the corresponding best feature set.

(3) For each listing in the California dataset, we have predicted on its associated reviews, and generated a score for what percentage of the reviews are "misleading" (either misleading-good or misleading-bad). This percentage acts as a final "score" that a Airbnb user could use to inform their choice of whether to rent or not.

## 6.2 Risk Assessment, Decision Points, and Mitigation Strategies

A significant risk in this project is the lack of inconsistent review data. Assessment of this can be done by focusing the analysis of model performance on the accuracy of predictions made on inconsistent reviews. Mitigation strategies for this have been discussed in the imbalance data section, 4.1. We are currently using random oversampling, after trying out the synthetic oversampling technique, SMOTE, which resulted in worse overfitting.

Another risk is high dimensionality. We are currently using PCA for this. We are creating 3-D word embeddings, and then applying 3-D PCA to reduce the dimensionality. In the future, we may experiment with the performance of other PCA dimensions.

# References

Devarakonda, Vaishali. "Examining Cases of Property Misinformation in Airbnb." EHT Explore, 2022, https://honors.libraries.psu.edu/catalog/7627vvd5116.

Jacob Devlin, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *CoRR* abs/1810.04805. (2018).

Kevin W. Bowyer, et al. "SMOTE: Synthetic Minority Over-sampling Technique". *CoRR* abs/1106.1813. (2011).

Pedregosa, F. et al. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* 12. (2011): 2825–2830.

# Appendices

## Misleading and Negative Keywords, and Common Phrases

For our Exploratory Data Analysis, we generated very crude labels based on the presence of “misleading” and “negative” words, as described in Section 2.1. Note that these words intentionally have no capitalization or punctuation. The full list if misleading words are:

- deceiving
- decieving (to catch typos)
- disappointing
- horrible
- terrible
- awful
- bad
- misleading
- inaccurate
- incorrect
- missing
- not as described
- not there
- wrong
- not as pictured
- lied
- lie
- liar
- lying
- fraud
- fraudulent
- scam
- scammer
- scamming
- scammed
- unsatisfactory
- unacceptable
- wasn't there
- was not there
- wasnt there
- photoshopped

The full list of negative words are:

- no
- not
- nor
- neither

- never
- none
- doesnt
- couldnt
- shouldnt
- wouldnt
- cant
- cannot
- wont
- isnt
- arent
- wasnt
- werent
- hasnt
- havent
- hadnt
- dont
- didnt
- neednt

We also removed some special phrases that are unique to Airbnb reviews. These commonly seen phrases are simply noise and do not help solve our problem. They are:

- airbnb
- austin
- texas
- home
- house
- florida
- lauderdale
- (hidden by airbnb) - Includes parenthesis

## Automatic Labeling Phrases

For our automatic labeling method, as described in Section 2.2.4, we used a set of phrases that we believe directly indicate a misleading review without question. Reviews containing these words were labeled as misleading, with a sentiment score dictating if the label is “mgood” or “mbad”. The phrases are:

- not as pictured
- is broken
- was broken
- are misleading
- is misleading
- was misleading
- very misleading
- were misled

- misleading description
- nothing like the pictures
- they are liars
- not as advertised
- not as described
- not as listed
- we were lied to
- lied to us
- does not work
- doesnt work

## Implementation Details

For transparency, here we list all the specific implementation details in one section. Specifics of some of these steps are explained in more detail throughout the paper.

- Our entire project was done in Python using Interactive Python notebooks (with a combinations of Anaconda and VSCode IDE)
- We used GitHub for version control and file sharing, as well as Google Drive for data storage.
- We used NLTK for data preprocessing, and BERT from the *transformers* Python package.
- We used the *tortus* Python package for labeling.
- We used *sklearn* for our machine learning model implementations, and *pickle* to save and load the models between steps.
- We used *pandas* for basic data transformations and working with CSVs.
- We used the *streamlit* Python package to build our demo application.

## Reproducibility

The folders you need to have in the data folder are as follows:

- `raw` - raw data from Airbnb. Also contains _raw_processed.csv, which is processed raw data, which was only used for EDA.
- `filtered` - subsets of the raw review data that we will label.
- `labels` - the labels for the filtered data. (can be joined with the filtered data using the `id` column, corresponding to the `listing_id` in the raw data)
- `sentiment` - the sentiment scores for the raw data. Used to generate the filtered subsets.
- `processed` - the processed data, ready to be used for training, testing, and transfer learning. Includes labels.

Following these steps should allow you to reproduce our work. There are no special requirements beyond Python 3 and using the *requirements.txt* file.

1. Download the code.
2. Install the requirements: `pip install -r requirements.txt`
3. Make sure you have the folders listed above in the `data` directory. If not, create them.

4. Extract the raw data and put it in the `data/raw` folder. This requires two steps:
    1. Export the listings worksheet from the Excel files, and save it using the naming convention `<GEO>_listings.csv`. For example, `data/raw/texas_listings.csv`.
    2. Place the `.xlsx` files in the `data/raw` folder, and run the `notebooks/data processing/load data.ipynb` notebook. This will extract the reviews from the Excel files and save them in the `data/raw` folder. There is preprocessing needed to correctly load the reviews, so this notebook is necessary.

       **NOTE:** If you don't want to extract out the raw data/get frustrated with Excel like we did, we provide the CSVs made from this step in `data/raw`.
5. Generate the sentiment scores used for subsetting the data for labeling. You can do this by running the `notebooks/data processing/generate_sentiment.ipynb` notebook. This will save the sentiment scores in the `data/sentiment` folder. This notebook must be run three times, changing the `GEO` variable in the notebook each time. The three values for `GEO` are `texas`, `california`, and `florida`. **NOTE**: This notebook will take hours to run for each geo. The data generated from this step is in `data/sentiment`.
6. Generate the filtered data for labeling. You can do this by running the `notebooks/data processing/generate_subset.ipynb` notebook. This will save the filtered data in the `data/filtered` folder. You only need to run this once.
7. Label the data. This is done manually, however we have a notebook to help with the task, using the data annotation library `tortus`. The tool we wrote to help us read the reviews and label them is in `notebooks/data processing/label_subsets.ipynb`. **NOTE:** We provide the labels for you in `data/labels`
8. Generate the autolabels. This is done using the `notebooks/AutoLabeling.ipynb` notebook. Make sure to change the GEO variable for each of texas and florida.
9. Once you have labels, you can generate the processed data. This is where stopword removal, joining labels with the data, etc. happens. This is done by running the `notebooks/data processing/generate_processed.ipynb` notebook. This will save the processed data in the `data/processed` folder. This will be run for each geography, changing the `GEO` variable in the notebook each time.
10. You are now ready to do feature engineering and modeling. Our entire feature engineering process and generation of our final features for training models is in `notebooks/modelling/feature_engineering.ipynb`. This notebook will generate the final features for each geography, and save them in the `data/processed` folder as `features_<GEO>.csv`. If multiple GEOs are provided it will create the combined features file with both geos in the name. For

example, for our training data, we ran this notebook with `GEO = ['texas', 'florida']`, and it saved the features in `data/processed/features_texas_florida.csv`.

11. Modeling! The `notebooks/modelling/models.ipynb` notebook contains the code for generating models and trying different feature combinations. The best model is saved as a pickle file at the end.
12. The final step is to evaluate on California dataset. This is done with `notebooks/modelling/cali_eval.ipynb`. Set the “MODEL_FP” variable at the beginning of the notebook to point to the saved model pickle you want (generated from models.ipynb), and you will get the performance on the california dataset, as well as a confusion matrix.