Georgetown University, Data Science Certificate-Fall 2016 Cohort

Team: Housing Risk

Members: Neal Humphrey, Khan Kashif (Coordinator), Ashish Lal, and Peter Mattingly

21 January 2017

Report: Housing Risk Under Section 8

1. ABSTRACT/DOMAIN

Under the U.S. Department of Housing and Urban Development, the Housing Choice Voucher Program, known as "Section 8," provides affordable, safe, and sanitary residential housing in the private sector. Participants include those in the metropolitan area whose income does not exceed 50% of the median income in the surrounding region; this provision would allow for low-income families, particularly the elderly, the disabled, as well as anyone else who may qualify according to local Public Housing Agency (PHA) regulations.

Escalating rates of gentrification, however, pose credible threats to this system, particularly those efforts to preserve adequate, affordable housing rentals throughout the metropolitan region of D.C. and across the United States. According to the Joint Center for Housing Studies of Harvard University (2011), while the demand for such rentals continues to increase, the supply inversely continues to shrink. Keely et al. (2012) in their report, "The Shifting Nature of U.S. Housing Demand," propose that demographic shifts, such as the adult millennial population, immigrant population, and aging baby-boomer population, will bear great impact on this trend, especially as these specific demographic groups will require more housing rentals over time. It is estimated that the rate of those seeking such living arrangements under Section 8 will increase from 360,000 to 470,000 every year from 2010 to 2020; and it is predicted that this rate will reach a total of more than 3.6 million of new renter households by 2020 (Joint Center for Housing Studies of Harvard University, 2011).

Currently, no available application allows for a user to evaluate whether or not a property under Section 8 will continue to remain under contract. When their contracts expire, private owners of buildings participating in the Section 8 program can choose to renew or terminate their contracts. If they choose to terminate, existing residents will be displaced and there will be less available affordable housing. Knowing which buildings are most likely to opt out of the Section 8 program is of interest to affordable housing advocates, residents, and local HUD offices, as there are often options for keeping this housing affordable by refinancing and tenant organizing. This research project seeks to build a predictive model based on existing data that can assess this risk.

2. HYPOTHESIS

We hypothesize that certain variables significantly influence the risk of a property loss under Section 8; the most predominant and conclusive include neighborhood housing prices, demographic shifts in population, and median household incomes. Moreover, the percentage of larger room types (e.g. properties with two bedrooms or more), ownership type, and other property-specific variables also serves as factors. By enlisting the available data on these variables using a computational analytic methodology, we can build a predictive tool that will estimate the risk of losing a property under Section 8.

Previous analysis by HUD in 2006 and again in 2015 compared the outcomes of all Section 8 properties and verified that there were correlations to factors such as neighborhood rent, building size and ownership type. We used this as in indicator that there was enough information to suggest it was possible to build a predictive model.

a. TOOLS & PROGRAMS

To ensure a predictive model that holistically integrates all extraneous variables which might determine the risk of losing a property, we determined several applications to be the most apposite to our investigation. Our tools for data ingestion include Amazon S3, PostgreSQL, and Python. The resultant code was stored in our GitHub repository, Other tools for data munging, wrangling, modeling, and visualization include MatPlotLib, scikit-learn, R, and Tableau.

3. METHODOLOGY

The team leveraged the data science pipeline to ingest, wrangle, model and visualize the data. A description of the specifics used during each phase of the pipeline follows.

a. DATA SETS

- **Section 8 Contracts Database** from the US Department of Housing and Urban Development. HUD distributes the current status of contracts in the Section 8 program through their website. We obtained historical copies from the Urban Institute of the Microsoft Access database from each time it was published by HUD since 2001; historical copies of the database were not available from HUD itself. Each year's data was imported into the PostgreSQL database as individual tables.
- **Population and Housing Data**: retrieved and compiled the 5-year survey estimates (from 2007-2014) in Population and Housing from American Community Survey based on zip code and county level.
- **Fair Market Rent (FMR):** obtained from HUD
- **Geocoded Identifiers**: Obtained from Census Geocoder

b. INGESTION

All datasets used by our team were stored on Amazon S3. This provided for a central location for all data files and allowed us to upload and download data files as needed. Each team member created their Amazon S3 account and used their account to upload / download data.

The primary mechanism for ingesting data into a PostgreSQL database was loading the CSV files obtained from HUD, American Community Survey (ACS) and other data sources via Python scripts. The decision was made to use PostgreSQL due to its ease of use, free availability and robust RDBMS features. The datasets were ingested locally on local installations of PostgreSQL databases on each team member's laptop, as well as remotely on an Amazon PostgreSQL database. A generic, multipurpose Python script was created that could take any CSV file and create a table based off that data. The script was made repeatable by providing a mechanism to skip over tables / data sets that had previously been uploaded successfully.

c. MUNGING & WRANGLING

Once the data was ingested into PostgreSQL, the next step in the pipeline was to wrangle the data to make it suitable for the machine learning exercise. This involved several steps:

- ***Relating and joining data***: all contract, property, FMR, and ACS records were joined together using SQL queries joins and uploaded into a series of intermediate database tables. The joins were based on contract or property ID, or geocode identifier. Obtaining the geocode identifier for all the properties being analyzed was a key challenge we had to overcome, in order to join data obtained from ACS (which had geocode information only), and the contracts and properties data from HUD. We uploaded all property addresses to the Census Geocoder site to obtain the relevant geocodes. Once the geocodes were obtained, they were uploaded into a separate table in the database. These identifiers were then used to join to the remaining ACS and FMR data.
- ***Removing duplicate and incomplete records***: Duplicate and incomplete records were removed from the joined dataset. This allowed us to work off a clean data set. There were also several years worth of data that were dropped off in the final analysis, since there was uncertainty about the completeness or veracity of the data obtained. The team ultimately chose to focus on data from **2009** to **2014** only. This data available for this range of years was complete, and importantly, other data sources such as the ACS data was available for these years, making our analysis more complete.
- ***Creating a single instance (decision) record***: Once the records were joined together, they had to be converted into a single instance that could then be passed to the SciKit Learn library in Python for purposes of machine learning. We determined that the instance that was needed was a decision of whether a property chose to stay in or leave the Section 8 program. As such, the joined data was converted into a single record per year per property, with a flag for whether the decision was made by the property to stay in, opt out, or something else (i.e. suspicious data). These decisions were made by looking at and comparing the contracts data year over year. For example, if it was determined that contract end date or contract term had changed from one year to another, then it was very likely that the property in question had chosen to stay in with a modified contract, and this record was marked as an 'IN' record. Similarly, if a property disappeared from the records, it was assumed that the property had decided not to renew, and the prior record was marked as an OUT record. To make the analysis of contract data across multiple years, we employed the use of SQL LAG / LEAD functions, which allowed us to look at partitions of data across multiple rows by contract or property, and avoid the need for joins to multiple copies of the same table, thus improving performance run times.
- ***Removing no change and churn records***: during the wrangling stage, we explored the datasets by using the Bokeh library to visualize the data. In the visualization below, each column represents one copy of the HUD contracts data from a different point in time (a "snapshot"). Each row is a single

contract, and a random sample of all the contracts in our data are included in this visualization. Cells in the visualization are coded as follows:

- Grey = no change from the previous snapshot for this contract
- Red = “out” decision, due to a change in contract status
- Light green = “in” decision
- Dark green = “in” decision that follows an “out” decision, indicating churn
- Purple = first observation
- Blue = suspicious or conflicting decision from different data fields.

This visualization revealed some important details. For example, it became immediately visible that there were some properties that were opting out and then opting back into the Section 8 program from one year to another. This event, which we referred to as churn, seemed suspicious, and we were unsure whether this data should be considered as part of our analysis. We also determined that this could be an issue with how the records were being reported and stored in the databases. To remove the impact of these churn records in our analysis, we made a decision to exclude churn records from our analysis. Utilizing data visualizations during this stage of the pipeline allowed us to make identify this anomaly and make this decision. We also decided to remove records from the analysis that provided no additional value or potential signal, i.e. records where there was no change to the status of a property.

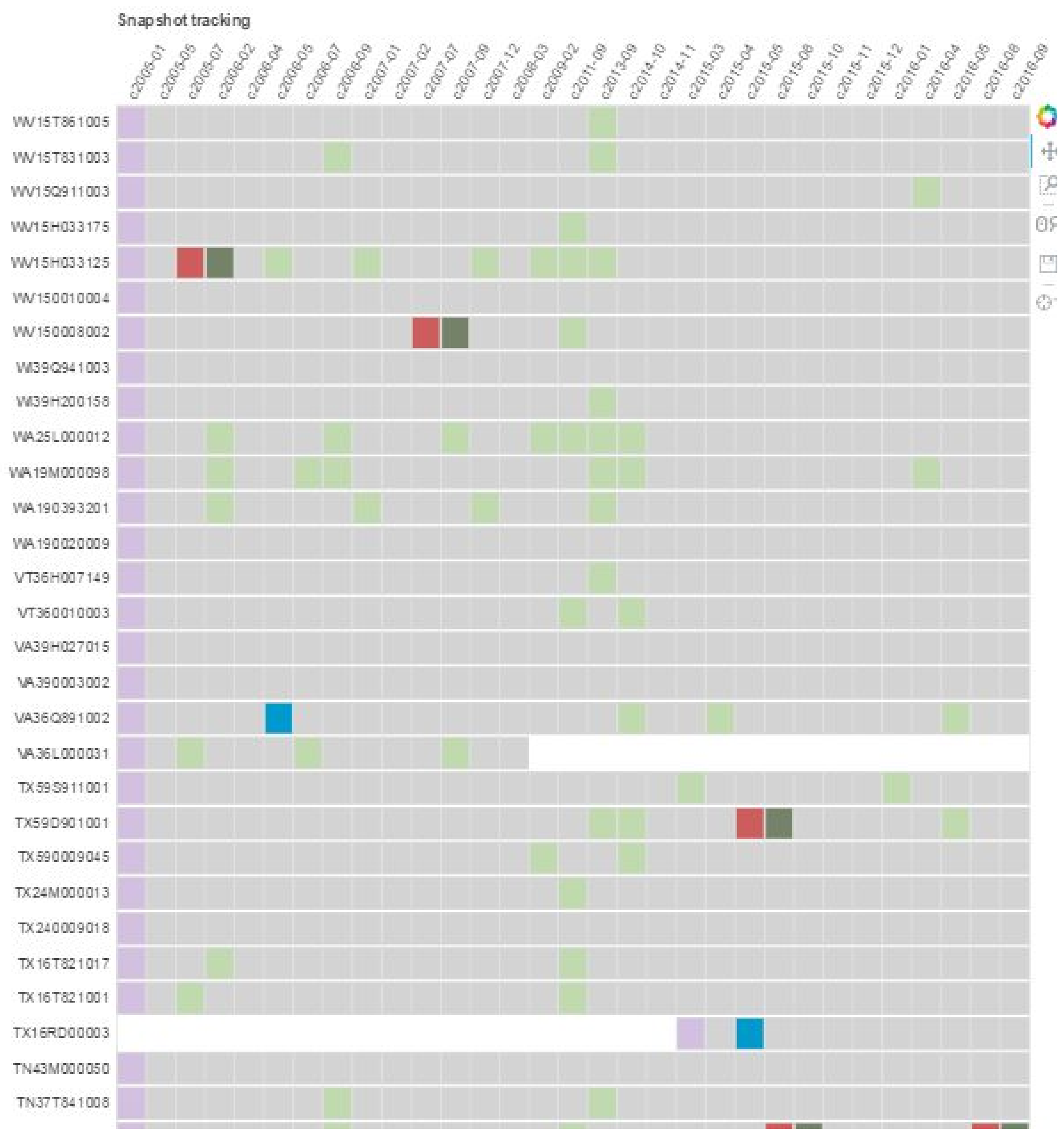


Figure 1 - Bokeh Plot of Raw Contracts data

- ***Finalizing decision instances***: After removing all records that were not of interest for the analysis, the team decided to finalize the records for analysis. The final number of records for the analysis were as follows:

Item Type	**Number of Records**
Original # of samples	400,633
# of IN records	29,361

# of OUT records	3,831
# of No Change / Churn records (excluded)	367,441

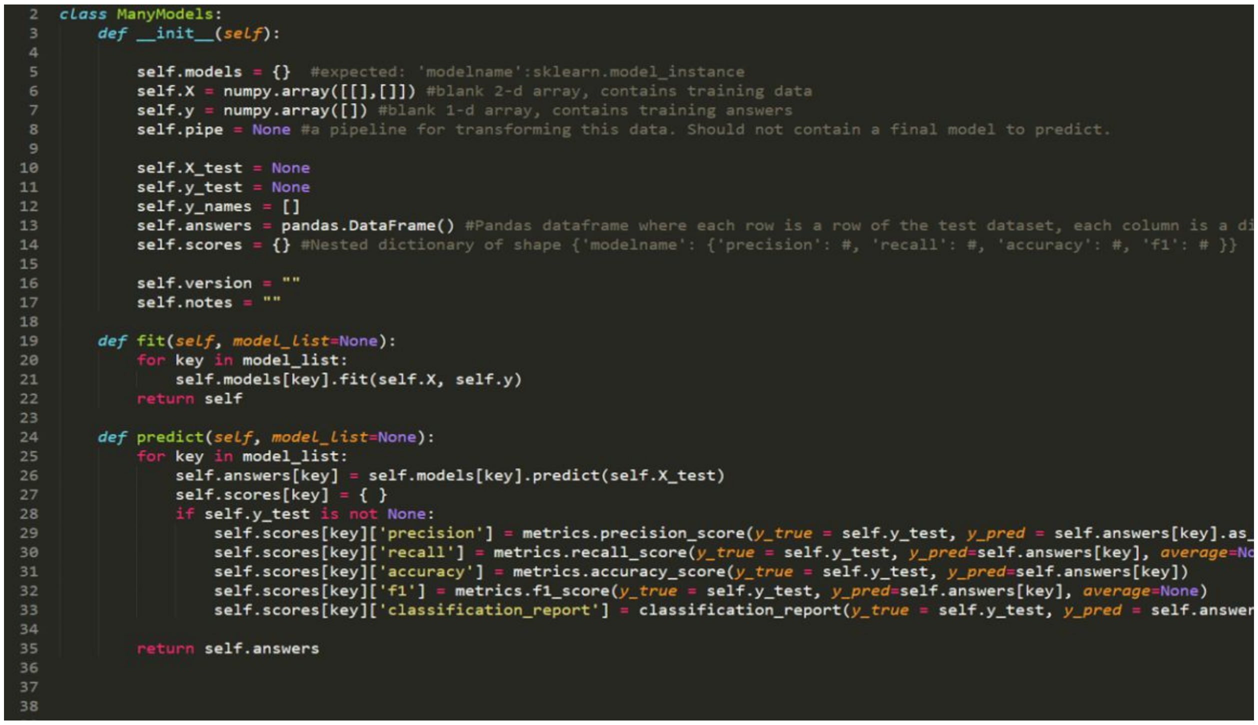
Table 1 - Records Analyzed for Machine Learning

d. COMPUTATION AND MODELING

Per the methods covered in the data science certificate program, we set up our modeling code to implement the machine learning models provided in the scikit-learn Python package. Our approach to structuring our training and testing of these models was designed for flexibility and comparability of multiple models.

Our primary method training and testing models revolved around a custom class to store each model instance (before and after fitting), the same single set of data that was used for training all the model instances, one set of data for testing all the models, the predicted answers for each test record and model, and summary scores for each model including precision, recall, and a classification report. Note that this is in contrast to using a KFolds approach or similar method for shuffling the data among different attempts at fitting the model, which would be more appropriate when finalizing the model training. All summaries shown here use the same 30% training split on our full data set, stratified to obtain a proportional balance in the training and testing data of “in” and “out” decisions.

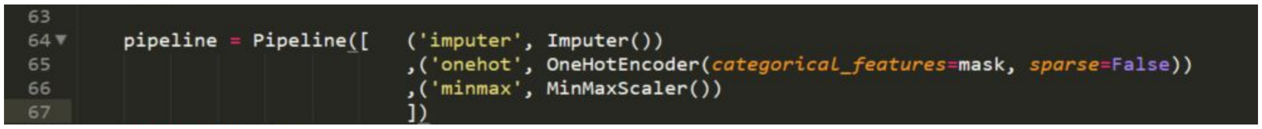
This custom class was implemented as follows (abbreviated for clarity). The full ManyModels class is available in the accompanying GitHub repository. By attaching our training and testing data to the ManyModels instance and then using the fit() and predict() methods of this class directly, we were able to be sure that all our models used the same data.



```python
class ManyModels:
    def __init__(self):

        self.models = {}  #expected: 'modelname':sklearn.model_instance
        self.X = numpy.array([[],[]]) #blank 2-d array, contains training data
        self.y = numpy.array([]) #blank 1-d array, contains training answers
        self.pipe = None #a pipeline for transforming this data. Should not contain a final model to predict.

        self.X_test = None
        self.y_test = None
        self.y_names = []
        self.answers = pandas.DataFrame() #Pandas dataframe where each row is a row of the test dataset, each column is a di
        self.scores = {} #Nested dictionary of shape {'modelname': {'precision': #, 'recall': #, 'accuracy': #, 'f1': # }}

        self.version = ""
        self.notes = ""

    def fit(self, model_list=None):
        for key in model_list:
            self.models[key].fit(self.X, self.y)
        return self

    def predict(self, model_list=None):
        for key in model_list:
            self.answers[key] = self.models[key].predict(self.X_test)
            self.scores[key] = { }
            if self.y_test is not None:
                self.scores[key]['precision'] = metrics.precision_score(y_true = self.y_test, y_pred = self.answers[key].as_
                self.scores[key]['recall'] = metrics.recall_score(y_true = self.y_test, y_pred=self.answers[key], average=No
                self.scores[key]['accuracy'] = metrics.accuracy_score(y_true = self.y_test, y_pred=self.answers[key])
                self.scores[key]['f1'] = metrics.f1_score(y_true = self.y_test, y_pred=self.answers[key], average=None)
                self.scores[key]['classification_report'] = classification_report(y_true = self.y_test, y_pred = self.answer

        return self.answers
```

Figure 2 - ManyModels class

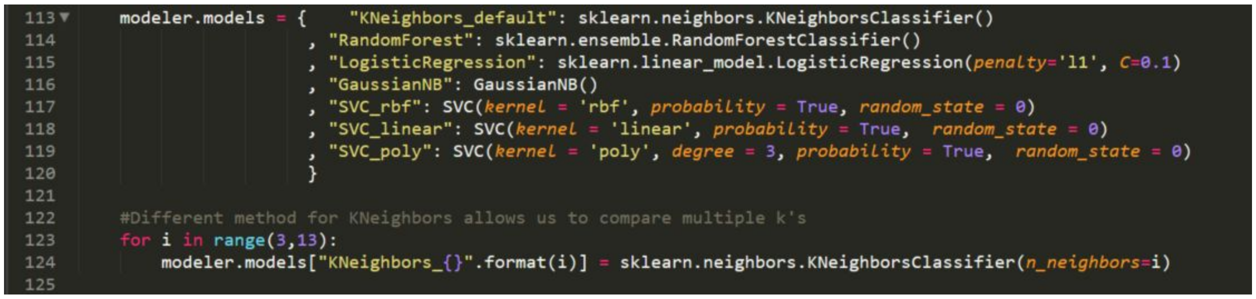
To make our data cleaning and transformation repeatable, we used a combination of custom categorical encoding using a meta data file, as well as a scikit-learn Pipeline. Any records with missing values were imputed, categorical features were encoded using OneHotEncoder, and all values were scaled to between 0 and 1 using the MinMaxScaler.



```python

    pipeline = Pipeline([   ('imputer', Imputer())
                           ,('onehot', OneHotEncoder(categorical_features=mask, sparse=False))
                           ,('minmax', MinMaxScaler())
                           ])
```

Figure 3 - Pipeline

For modeling, we selected a variety of supervised classification models, under the assumption that it was best to compare the model performance under the "try them all" approach. For our use of K-Nearest Neighbors, we specified 10 model versions using values of 'k' from 3 to 12; for most other models the hyperparameters were largely left to the defaults of the scikit-learn library. The full list of our models and their parameters are shown below, and are also available in the accompanying run_models.py file.



```python
    modeler.models = {    "KNeighbors_default": sklearn.neighbors.KNeighborsClassifier()
                        , "RandomForest": sklearn.ensemble.RandomForestClassifier()
                        , "LogisticRegression": sklearn.linear_model.LogisticRegression(penalty='l1', C=0.1)
                        , "GaussianNB": GaussianNB()
                        , "SVC_rbf": SVC(kernel = 'rbf', probability = True, random_state = 0)
                        , "SVC_linear": SVC(kernel = 'linear', probability = True,  random_state = 0)
                        , "SVC_poly": SVC(kernel = 'poly', degree = 3, probability = True,  random_state = 0)
                        }

    #Different method for KNeighbors allows us to compare multiple k's
    for i in range(3,13):
        modeler.models["KNeighbors_{}".format(i)] = sklearn.neighbors.KNeighborsClassifier(n_neighbors=i)

```

Figure 4 - List of Models and Parameters

Prediction

Given the premise of our project, the most salient statistics are the precision and recall scores for the identification of "out" decisions. Our first set of models, trained on the full set of validated 'in' and 'out' decisions are presented in the figure below. These are sorted from highest to lowest based on the precision of the 'out' decisions. The 5 bar charts are:

- In: Precision - percent of 'in' predictions that were actually 'in' for the test data set
- In: Recall - fraction of the 'in' decisions contained in the testing data set that were identified as 'in' by the predictive model.
- Out: Precision - as above, for 'out' decisions
- Out: Recall - as above, for 'out' decisions
- Out freq frac - the fraction of 'out' predictions out of all predictions. For reference, the test data contained 11.5% of samples that were actually 'out' decisions.

As can be seen, some models had quite good precision at 0.85; however, this came at a tremendous cost of poor recall. Looking at the Out freq frac, it can be seen that the models are biasing toward almost never predicting an 'out' decision; the high recall and precision for 'in' decisions come from, essentially, predicting 'in' for every building. The one exception to this is the Gaussian

Naive-Bayes model, which although it had higher recall for the 'out' decisions this came at the cost of poor recall of 'in' decisions.

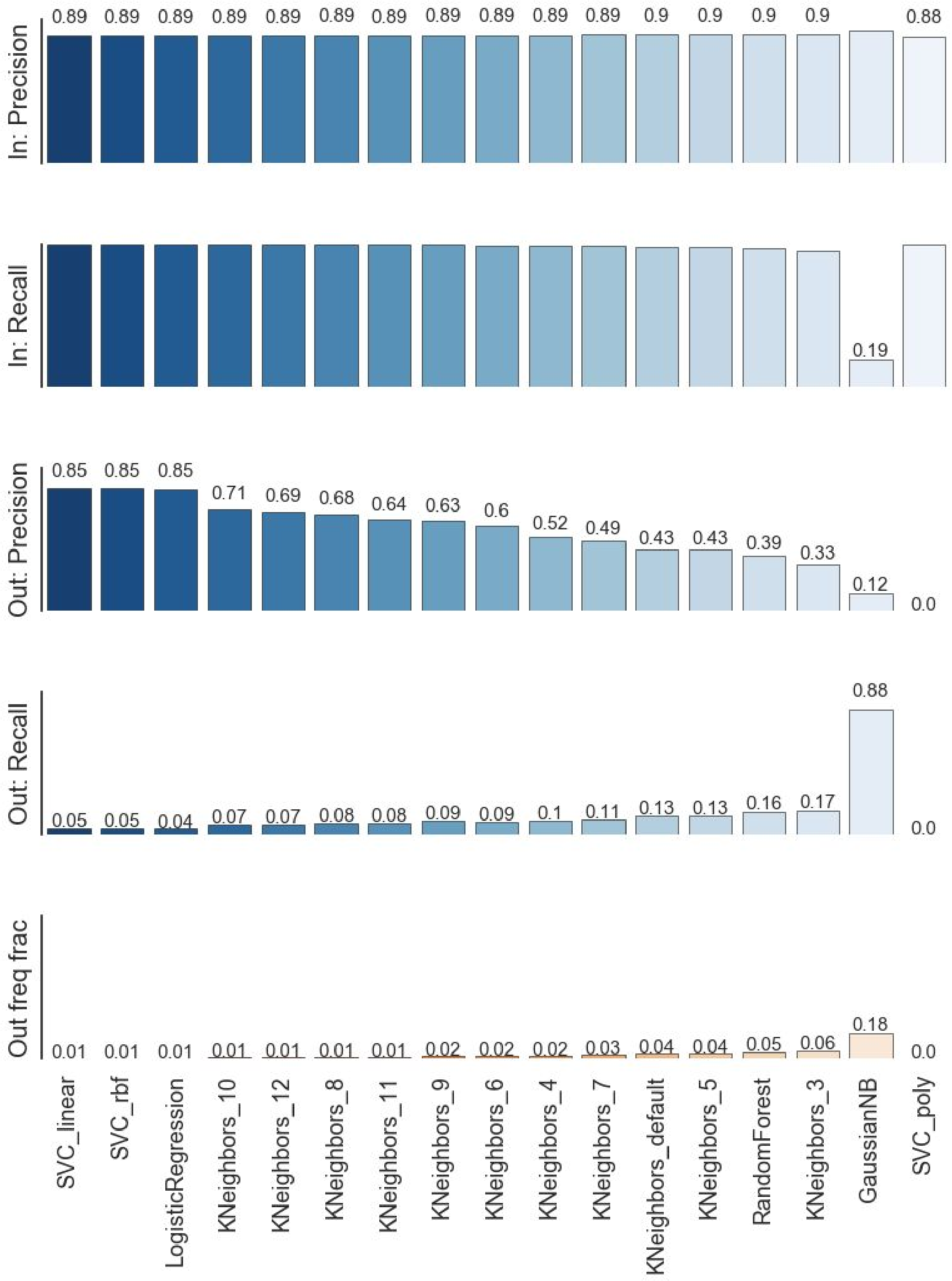
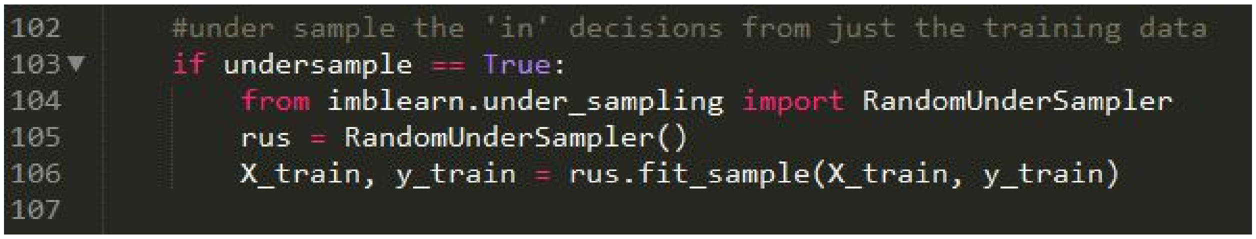


Figure 5 - Modeling Results: Full Training Data Set

The biggest issue with this first training attempt was clearly the class imbalance, which encouraged the models to predict ‘in’ at all instances. As our primary attempt to deal with this issue, we repeated our analysis but with a balanced class. We opted to use an undersampling technique to achieve this balance, as we had enough samples to feel this was preferable to other resampling options like sampling with replacement or synthetic samples. To perform the undersampling, we first split the data (using the same 30% testing split), and then randomly removed additional samples from only the training data, preserving the imbalanced classes in our testing data. We used the imbalanced-learn package in Python to achieve this undersampling, as shown in the code snippet below.



```
    #under sample the 'in' decisions from just the training data
    if undersample == True:
        from imblearn.under_sampling import RandomUnderSampler
        rus = RandomUnderSampler()
        X_train, y_train = rus.fit_sample(X_train, y_train)

```

Figure 6 - Imbalanced Learning Code

The results of this analysis can be seen in the figure below. The format of this figure is the same as the one above, although results are re-sorted based on the Out Precision of this new modeling result.

Undersampling did achieve the desired effect of encouraging the models to predict ‘out’ decisions more often - this can be seen in the Out Frequency Fraction (Out freq frac) graph. This increased recall (due to predicting ‘out’ more often) but drove down precision.

Model Selection

While the models trained on the two versions of training data performed very differently with respect to their predictions of ‘out’ decisions, neither approach was able to provide the quality of results we expected or hoped for. We concluded that, for the data and parameters that we chose, we weren’t able to achieve a model with sufficient accuracy for reliable predictions of which buildings would opt out of their Section 8 contracts.

If we were to identify one model that was best suited to providing a flag for advocates to consider, we determined that the RandomForest model that was trained on the undersampled data set would be the most useful for predicting ‘out’ decisions. To make this decision, we considered the context that such a predictive model might be used. Policy advocates looking to intervene early with a building do not need to know for sure whether a building will opt out - they just need to know whether a building is worth investigating further. Therefore it’s important that the model not be too selective - it’s better to have many false positives rather than missing buildings that should have been flagged. We do, of course, want as high a precision as possible among the buildings flagged. The RandomForest model seemed to strike the best balance between these competing concerns.

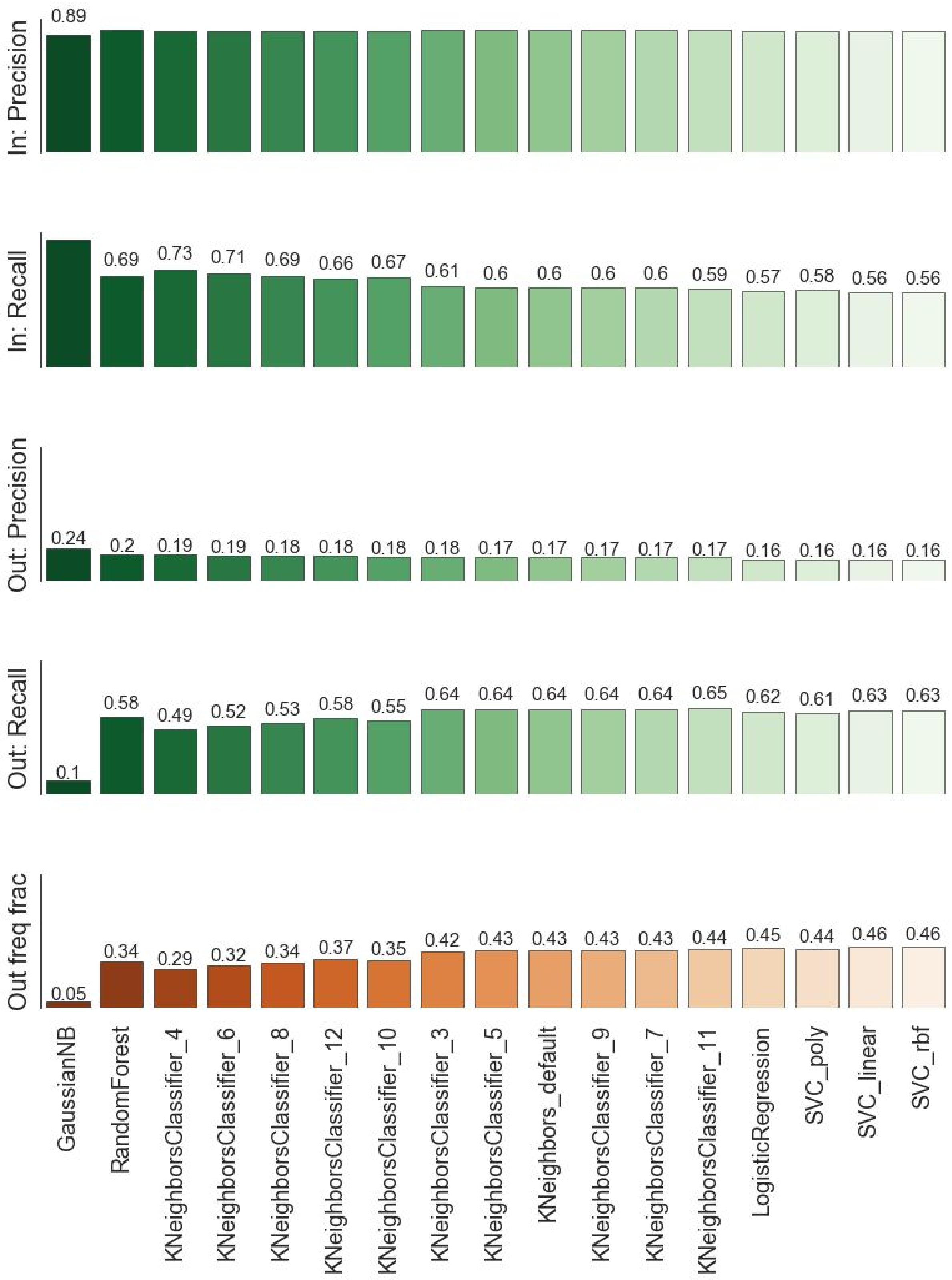


Figure 7 - Modeling Results: Undersampled Training Data Set

Methods of improving future model performance

Despite our initial somewhat disappointing results in model performance, there are additional techniques that would be worth pursuing to attempt to improve the predictive power of a model. Some things we think could improve the performance of our analysis include:

- **Obtain more data.** We did not have as many data sources in our final analysis as we had originally intended, due to the amount of time required to transform our source data into trainable 'decision' instances.
- **Ensemble Classifiers.** By combining the results of multiple models, we could see if we could improve our predictions.
- **Simplify the problem.** Another option would be to train our model on a simpler identification problem that is still useful and relevant to the affordable housing advocates. For example, rather than trying to predict decisions immediately, we could create instances that indicate whether a building will opt out within the next 5 years.
- **Additional imbalanced sampling options.** We did not pursue additional options outside of the undersampling approach; this could improve results.
- **Hyperparameter tuning.** We only attempted a small amount of hyperparameter tuning, in the form of the 'k' value in our K Nearest Neighbors models.

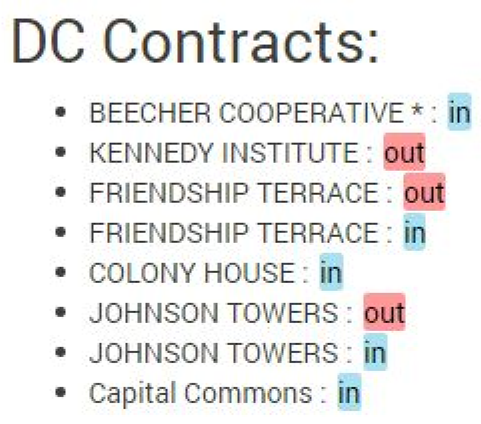
4. VISUALIZATION APPLICATION (DEMO WEBSITE)

In the real world, a model such as the one we created could be integrated into a website such as the Housing Insights website currently under development at Code for DC. The value of this model would be to update the predictions for buildings as information about them changes over time - changes in neighborhood demographics, market rate rent levels, or building ownership status. Since these changes occur gradually over time, we elected to demonstrate the ability of our predictive algorithm in a live data product by using a scenario approach - allowing the users to alter the values of a few selected parameters and then rerunning the model using the combination of the real data and the altered parameters.

For this model, we constructed a simple Flask-based website. This website contains the current (2016) data for all the 110 Section 8 buildings in Washington, D.C. When the website is loaded, the fitted RandomForest model (from the undersampled analysis) calculates predictions for all DC buildings as an 'initial' case. Users can then enter new values for "neighborhood median market rent" and for "contract duration", the duration of the most recent contract extension. The Flask code then recalculates predictions for all buildings by substituting the single user-entered value for all buildings - i.e., if the original median rent was $1,200 for one building and $2,000 for another, the user entered value of $2,500 would be used for both buildings.

The website shows the predictions for each building in the current scenario (in or out). After running the analysis, the new in/out predictions are shown, with the addition of a "changed" flag if the new prediction is different from the baseline prediction. The first few buildings from this analysis are shown in the two screenshots below. Note that some buildings have multiple contracts and so the building names are shown twice in this simple website format. A live demo is also shown below as an animation in the online version of this report.

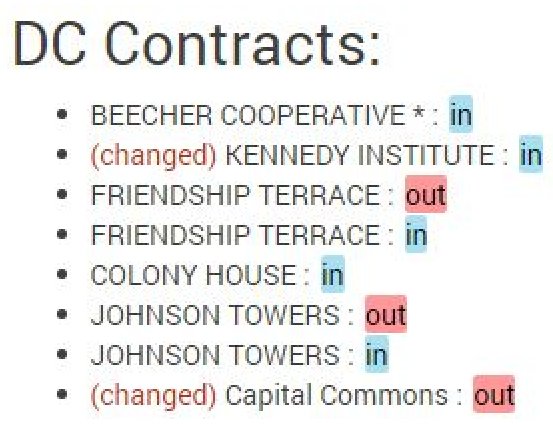
Base predictions:



DC Contracts:

- BEECHER COOPERATIVE * : in
- KENNEDY INSTITUTE : out
- FRIENDSHIP TERRACE : out
- FRIENDSHIP TERRACE : in
- COLONY HOUSE : in
- JOHNSON TOWERS : out
- JOHNSON TOWERS : in
- Capital Commons : in

Updated predictions: (rent = $2,500, duration = 12)



DC Contracts:

- BEECHER COOPERATIVE * : in
- (changed) KENNEDY INSTITUTE : in
- FRIENDSHIP TERRACE : out
- FRIENDSHIP TERRACE : in
- COLONY HOUSE : in
- JOHNSON TOWERS : out
- JOHNSON TOWERS : in
- (changed) Capital Commons : out

Live Website Demo

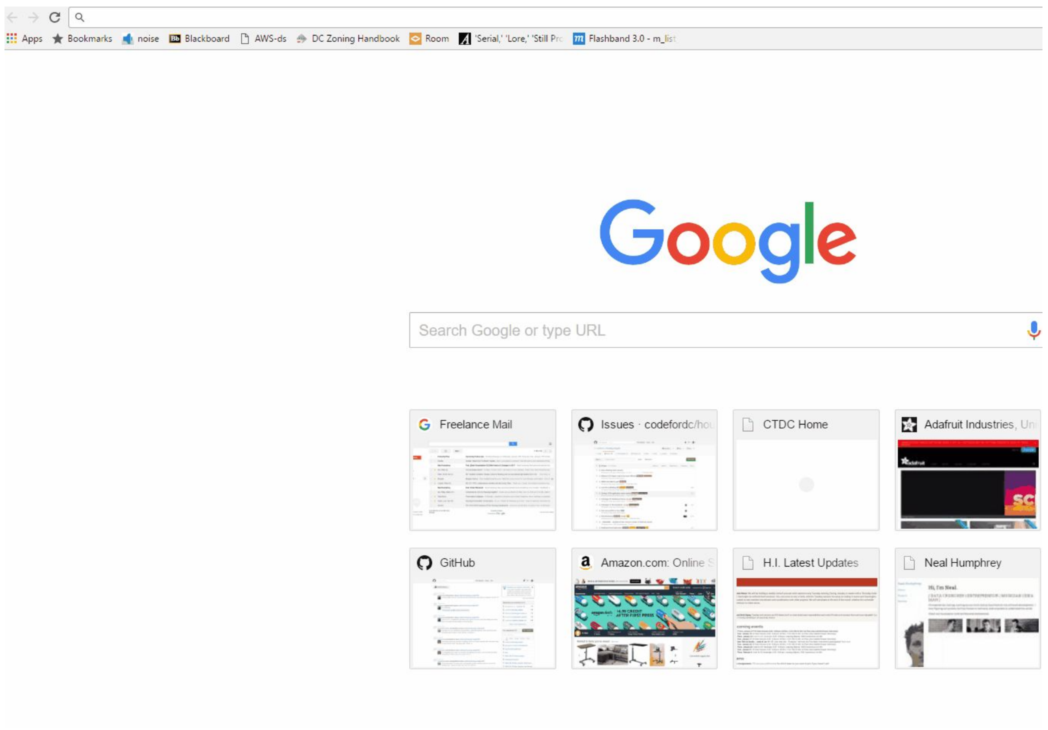


Figure 8 - Website Demo

While overall we did not feel any of our models were sufficiently predictive to provide significant value to the affordable housing advocates, this site demonstrates the type of capability that could be created if a sufficiently predictive model were identified.

We also investigated the use of this data for a simple visualization of areas within each neighborhood in DC where there is a risk of a property opting out of a section 8 program. This could indicate certain neighborhoods that are more susceptible to property owners choosing to opt out of the Section 8 program. This visualization is depicted below:

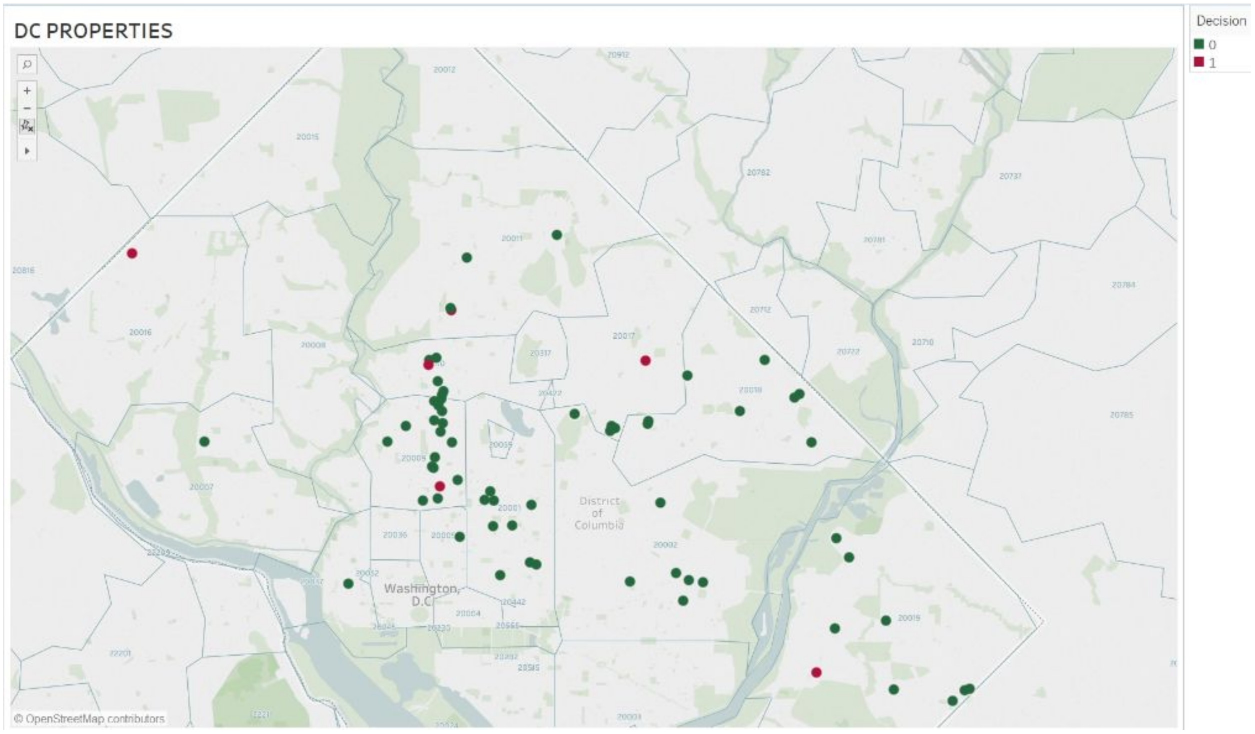


Figure 9 - Neighborhood Data Visualization

5. CONCLUSION

    a. LESSONS LEARNED

- ***Advanced acumen and subject matter domain knowledge necessary***: At the outset of this project, we were optimistic and, overall, centered on the concept of applying our skills from the course to such a unique endeavor. However, as we initiated the first stages of research and ingestion, we soon realized how naive we (or, most of us) were. Domain expertise in computer programming and specifically in the chosen programming language (Python) proved critical for the success of this project. Knowledge of the subject matter, too, proved essential; otherwise, most if not all but one member of the group would not have known how to empirically approach the goals for this research. The team was fortunate to have a member on the team who had an extraordinary familiarity with the subject-and, incidentally, who was working on a similar problem at work. Therefore, it is imperative that experience in various abilities and skills is realistically addressed and collaboratively evaluated.

- ***Strategizing effectively, productively, and constantly***: No cautious, creative researcher starts a project without a backup strategy. Nonetheless, given the limited timeframe of this project, the team needed to ensure its completion (and, ultimately, its success) at any costs. For this project to be "successful," members needed to continuously learn from mistakes and apply innovative technique to previous error. A more rigorous application of the Agile methodology could have benefited the team's productivity and allowed for more efficient accomplishment of objectives. Had our objectives been more defined at the outset of the project, we could have engaged with additional levels of analysis and encompass a greater variety of skills. Moreover, our group underestimated the extent to which we would be able to wrangle the data on schedule. While it is admirable to maintain deadlines, it is even more noble to admit when certain tasks will take much, much, much longer than others. Planning for the worst is what contributes to a data scientist's success.

- ***Interplay of significant variables***: The model which we produced would be more holistic if variables, such as race, gender, crime, and other factors that could influence gentrification rates and the housing market, had been more available and easily accessible. However, such data proved either to be too complicated to incorporate or structurally lacked relevancy. To capture a correlational relationship of any kind with these factors (crime particularly) would have required an advanced level of insight in the munging process. To plot a factor such as crime would have posed a plethora of questions and ethical compromises: i.e. what kind of crime? Is all relevant crime recorded accurately? What types of crime impact these decisions the most? What groups of people are forced into crime and how do racial demographics vary by crime committed? Such an endeavor would have been a capstone project on its own. Though we were unable to incorporate these variables, they still bear significance in such a way that might influence later model-making.

- ***There is (or needs to be) a Python programmer in all of us!***: The paramount decision that future cohorts must take is determining a research project that can incorporate the strength of all team members. Since our team’s project required an extensive knowledge of Python, progress was hindered due to a lack of expertise with this language amongst all team members, and tasks could not be completed independently without additional assistance from the more seasoned Python programmers in the group. Overall, the only way to ensure that all members are able to partake in such a project equally and fairly, is to ensure that (1) the scope of the project be precise and that its focus is transparent and understandable; (2) and each member should ideally focus on at least one software or programming language to use for analyses. Otherwise, it is safe to assume another topic for the group’s project which can facilitate all members’ background interests and programming expertise.

6. Github Location of Project Code
    a. Housing Risk Github project

7. Link to Capstone presentation
    a. Housing Risk presentation

8. REFERENCES

Joint Center for Housing Studies of Harvard University. 2012. “America’s Rental Housing Meeting Challenges, Building on Opportunities.”

Louise Keely, Bart van Ark, Gad Levanon, and Jeremy Burbank. 2012. “The shifting nature of U.S. housing demand,” The Demand Institute, 20.