BUSN 41204: Machine Learning

Prof. Mladen Kolar

Winter 2023

Final Project Writeup

Alex Bue
Christian Gregorich
Conrad Liu
Ujjwal Sehrawat

We pledge our honor that we have not violated the Chicago Booth Honor Code in this assignment.

1. Executive Summary

Our project uses logistic regression, random forests, boosted trees, and neural networks to predict the failing results of food inspections in Chicago. All businesses in Chicago are inspected for compliance with City sanitation code at least once a year. This project is important because identifying businesses that are likely to fail inspections can mitigate public health risks. Conversely, identifying businesses that are not likely to fail can reduce fiscal waste.

Food inspection data is publicly available but limited in terms of features. For this project, we extensively augment existing food inspection data using a variety of other variables, including Zillow home valuations, crime data, other violations of municipal code, assorted urban data, and historical records of businesses' previous food inspection types, violations, and results.

After augmenting the dataset and selecting variables, we find boosted trees yield the best model with an accuracy of 64.5 percent, the lowest false negative rate, and an uplift value of greater than 2.5 for the top decile of observation. While this is not an especially strong predictive output, we believe these results compare favorably with the Chicago Department of Public Health's current targeting methods.

2. Introduction

Our project uses data from the 250,203 food inspections conducted in Chicago since 2010 and attempts to predict their 48,590 failures. This data has 18 variables, each of which takes a range of unique values.

Variable	Unique Levels	Description
Inspection.ID	250,203	Unique ID for each inspection of each business
DBA Name	31,076	"Doing Business As" Name
AKA Name	29,576	"Also Known As" Name
License	42,793	Unique license for each business
Facility Type	510	Unstandardized (e.g., "Ice Cream", "Pool", watermelon house") data that takes many values
Risk	3	An ordered factor that takes three levels: Risk 3 (Low), Risk 2 (Medium), Risk 1 (High).
Inspection Date	3,328	Presented by month, day, year, hour and minute.

Inspection Type	110	Values include “complaint” and “re-complaint” and show less variation than Facility Type.
Results	7	A factor variable that takes six variables: “Fail, No Entry, Not Ready, Out of Business, Pass, Pass w/ Conditions”
Violations	180,919	Extremely diffuse as each “violation” lists all violations from a list of more than 50 potential violations.
Latitude	17,905	Locational data.
Longitude	17,908	Locational data.
Location	17,916	Presents longitude and latitude together.

52 full-time employees and $6,213,547 of Chicago’s FY2021 budget were dedicated to food inspections in FY2021. The Department of Public Health assigns risk ratings of high, medium, or low to food businesses. Approximately 23% of recorded food inspection observations have resulted in failure. The percentage of failures is uniform across risk assessments, meaning that approximately 23% of high risk and 23% of low risk inspections have resulted in failure. This does not mean, however, that high risk businesses fail just as often as low or medium risk ones. Risk categories prescribe different inspection frequencies. High risk businesses should be inspected twice annually according to municipal code. In practice, however, high risk businesses are inspected nearly as often as the low and medium risk businesses according to an audit in 2016, which found only 49% of high risk businesses had received the prescribed two inspections in the last year. We address the implications of inspection heterogeneity later.

3. Data

We use data from the following sources:

1. Chicago food inspection data
    a. https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5
2. Chicago crime data
    a. https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2
3. Chicago ShotSpotter data
    a. https://data.cityofchicago.org/Public-Safety/Violence-Reduction-Shotspotter-Alerts/3h7q-7mdb
4. Chicago business licenses data
    a. https://data.cityofchicago.org/Community-Economic-Development/Business-Licenses-Current-Active/uupf-x98q

5. Chicago grocery stores
    a. https://data.cityofchicago.org/Community-Economic-Development/Grocery-Stores-2013/53t8-wyrc
6. Chicago Library data
    a. https://data.cityofchicago.org/Education/Libraries-2023-Holds-Placed-by-Location/826i-g5qe
7. Zillow Home Value Index (ZHVI) data
    a. https://www.zillow.com/research/data/

3.1 Inspection Data Limitations

The original inspection data has many observations but few variables. The immediate challenge is that most of these variables are not especially explanatory. They are either idiosyncratic, taking many unique values, or generalized, taking few unique values. "Facility Type" seems descriptive but suffers from a lack of standardization. Locational data such as zip-codes may be extracted and used to predict inspection results but are not informative. They provide information about where businesses are more likely to receive certain results without explaining why. This explanatory element is critical to our project; without it, and without a degree of understanding, our results will not be actionable from a policy perspective. Our project therefore seeks to balance prediction with interpretability or "relationship-type" questions.

Many of the other variables in the inspections data are not features but response variables. Violations, while presumably highly predictive of failing results, is a consequence of the inspection and not its antecedent.

We also clean and transform many variables in our dataset as follows:

1. To get more accurate representations at each inspection level (row), we perform data cleaning and data transformation for our 'inspection_type' and 'inspection_results' columns. For 'inspection_type' which had 100+ categories in the original dataset, we spell-corrected and collated each original category into 6 collective buckets: 'Canvass', 'Complaint', 'Consultation', 'License', 'Task Force', and 'Other'. 'Canvass' inspections are the most common type of inspection performed at a frequency relative to the risk of the establishment. We use a similar procedure to clean and transform the 'Facility Type' column and also clean the 'City' column for spelling errors and missing values. We clean all other columns in a similar way wherever necessary.
2. Generally speaking, we handle missing values by imputing a 'nan' or 'OTHER' string in case of a categorical column and a numerical float 0.0 in case of numerical columns. For Zillow prices, we impute missing values per zip code with an average for all zip codes for a particular city.
3. We use the 'License #' as our unique identifier for each facility. However, this column has some missing values. We impute these values with an ASCII value generated from the unique string

combination derived from the columns 'DBA Name' and 'Street Address'. We believe that in case of a missing or in the absence of a valid license #, the combination of 'DBA Name' and 'Street Address' comes closest to being a placeholder of a unique id for a facility. However, it is an approximation and can be wrong in some instances. For example, we would be counting two Subway restaurants as the same restaurant if they were on the same street and their DBA Names are just 'Subway' (which is likely after looking at the dataset). However, we still go forward with this imputation as the subset of rows with missing values for License # are very few and chances of the above scenario emerging are all the more reduced. In the end, we use a modified version of 'License #' to uniquely identify our facilities.

3.2 Inspection Data Augmentation

We looked to augment the Inspections data with additional features in order to better predict failing results. First, we considered plausible hypotheses about which businesses are most likely to fail inspections.

On the one hand, it is sensible to assume that businesses in low-income areas are more likely to fail inspections. They are often less wealthy, which means they have fewer resources with which to preemptively address those complaints through improved sanitation and cleanliness. Clientele are also likely to be poorer, which might express itself through reduced mobility. Because switching for these patrons is relatively more costly than it is for affluent patrons, businesses in low-income areas have fewer incentives to improve services. There are also considerations of equity which, while hard to explicate quantitatively, might play an important role. It is precisely because businesses in low-income areas are expected to fail inspections more often that they are inspected more often, which results in more failing results.

On the other hand, it seems plausible that businesses in higher-income areas may fail inspections more often. The clientele could be highly discerning and more likely to complain. Foot traffic is perhaps greater, leading to increased wear and tear on facilities.

These hypotheses make apparent that a variety of variables related to socioeconomic status and foot-traffic could be helpful for predicting failing food inspection results. We augment our Inspections data with housing value data from Zillow, which is a proxy for an area's socioeconomic development, and City of Chicago crime reporting, which is a proxy for foot-traffic.

3.2.1 Temporal Dependence

We note the longitudinal nature of our data. Our basic observations at the inspection level (representing a single row in our dataset) are temporally clustered within each facility. Given this temporal dependence, we expect each facility's inspection results over time to be serially/auto correlated to some degree. To leverage this potentially meaningful information for better predictions, we create 28 new time series/historical

features that capture additional information about 'frequency of inspections', 'inspection types', 'inspection results', and 'number of violations' from the previous available time point(s) for each restaurant.

Our time series columns can be broadly classified into two main categories:

1. Previous immediate time point data,
2. Previous cumulative time points data.

Previous immediate time point data: We expect observations that are closer in time to be more correlated with each other relative to ones that are further apart. We thus pay special attention to the 'inspection type' and 'inspection result' of the previous available time point for each restaurant. In case an inspection is the first event for a given restaurant in our dataset, we impute both the 'prev_inspection_type' and 'prev_inspection_result' columns with a string value of 'nan'.

Previous cumulative time points data: In our dataset, the 'Risk' variable (high/medium/low) corresponds to the number of inspections performed for a given restaurant in the past. The CANVASS inspection type is the most common inspection done directly based on a restaurant's assigned risk value. However, we find that the value of the 'Risk' variable for each restaurant in our dataset is a constant, thus, appearing to be time invariant. We create a time-varying version of this variable called 'prev_#_inspections' which keeps track of the cumulative sum of # of previous inspections for a restaurant and is updated with each row. We apply the same logic and create a time series variable called 'prev_#_violations' that reflects the cumulative time varying number of violations made by a restaurant. We also create 18 time series columns to capture the cumulative history of pass, fail, and other result outcomes for each of the 6 inspection types for each facility. In a similar way, we create 6 time series columns to capture the cumulative history of the number of violations for each of the 6 inspection types for each facility.

Our hypothesis is that businesses' past inspection results and their associated data (both single and cumulative time points) will be predictive of future results. Also, we are aware that our approach of including time series columns rests on a simplistic assumption of stationarity as we do not consider seasonality in our time series data.

3.2.2 Zillow

Zillow estimates housing value using the Zillow Home Value Index (ZHVI), which is "built from the ground up by measuring monthly changes in property level Zestimates, captures both the level and home values across a wide variety of geographies and housing types" (link). The Index is available at many levels of geography, from city to state.

We apply the Index to zip codes in Chicago. Index estimates are available on a monthly basis in each zip code from 2000.

We use Dplyr to create a “long” data-table by converting the columns of monthly Index prices into rows for each zip code for each reported month. We then join the prices to the Inspections data. The prices are staggered with 1-month lag, meaning that last month’s Zillow Index estimate will be used to predict this month’s inspections results.

3.2.3 Crime

Chicago's crime reporting is high-quality and standardized (link). More than 7 million crimes have been reported since 2001. Granular data is available at the level of location, type of crime, and census tracts, among other variables. We focus for our purposes on “primary type of crime”, which is standardized at the State level and takes 35 values including, for example, “burglary” and “narcotics” (link).

We create 35 columns in which each column corresponds to a primary type of crime. We then group observations by zip code, month, and year. Sums of the total instances of each type of crime are segmented by zip code and a combined value of month and year. These values are then joined to the Inspections data and staggered by one-month for predictions. As a result, we have grown our feature variables by 35.

3.2.4 ShotSpotter

Like other public safety data, the City of Chicago keeps high-quality data captured by its ShotSpotter system (link) since 2017. ShotSpotter is an automated system that uses audio data to detect gunshots via a series of sensors placed throughout the city. Many columns exist in the ShotSpotter dataset, but we chose the two we thought might be most useful: total ShotSpotter alerts and total number of rounds fired. We aggregated these columns at the zip code level and then joined the zip code-aggregated figures on the inspection data observations. We created features for both the raw totals since 2017 and the per-capita equivalent. ShotSpotter sensor placement and activity are likely to be highly correlated with other public safety-related metrics, but our hope is that there would be additional predictive power.

3.2.5 Population

We used Census Bureau data to find the 2021 population of the various zip codes contained in the inspections dataset. Our thought was that restaurants located in more densely-populated areas of the city may behave differently than less densely populated ones (either lower density housing, zoning that skews toward commercial/industrial, or simply further from the city core). Population data was used both as a standalone feature and to create per-capita versions of several other features aggregated at the zip code level.

3.2.6 Business Licenses

The City keeps an up-to-date registry of all businesses currently operating in the city and some of the surrounding areas. The data was last updated in early March, 2023. Similar to our thinking on including

population data, we hypothesized that restaurants operating in areas that are dense in business establishments might differ from those that are not in some important way that allows us to get predictive information about restaurants' ability to pass inspections in a given area. We included both aggregate and per-capita versions of business licenses as features in the final dataset.

3.2.7 Grocery Stores

Another interesting dataset in the city archives is a dataset on grocery stores across the city. Unfortunately, it has not been updated since 2013, but we figured it might be interesting to include given that supply of grocery stores is likely to be fairly inelastic due to high capital intensity of building new grocery stores. This could be an interesting feature for at least two reasons. First, it is similar to the argument for population and business density. Perhaps there's some underlying behavior about restaurants in areas that are denser overall or denser in retail that allow us to predict inspection failure. Furthermore, grocery stores are specifically interesting because they might compete for wallet share with restaurants in the same area. One potential hypothesis for how this feature could be useful is if in areas with fewer grocery stores, restaurants face less competition and therefore have less incentive to compete on safety inspections. Similar to some other features, the count of grocery stores was aggregated by zip code and then joined on the inspection observations. We included both grocery stores and grocery stores per capita.

3.2.8 Library Activity

Finally, we added City of Chicago data on library holds from January 2023. While we did not anticipate this to be a strong predictor, we figured it would be advantageous to bring in data from a diversity of domains. Our hypothesis is that restaurants that operate in areas with high library utilization may be embedded in active communities, and restaurants in those communities may on average feel some higher level of responsibility to patrons and therefore be less likely to fail safety inspections. Library data was aggregated at the zip code level and then joined on the inspections data. We included both the number of library holds and the per-capita version of the same.

4. Models

Because the goal of our project is predicting the unsuccessful results of food inspections, we start by removing results variables for observations which were not inspected – "not ready" and "out of business". Then we create a column taking binary values in which "1" means "fail" and "0" means "pass" or "pass with conditions".

We run logistic regression, random forest, boosted trees, and neural networks on the original inspections data un-augmented by additional datasets. The best model is boosted trees, but our results are poor with an error rate of approximately 40%. This is only slightly better than guess in terms of our binary "fail" variable.

We then set about incorporating our augmented data. There are 140 columns in this data. In order to remove variables, we attempt to run the Boruta algorithm. However, the process is interminable. Therefore we proceed by eliminating variables manually. Later, with a model selected, we will re-un Boruta.

Initially we choose to remove variables that are either redundant because they are intimated elsewhere or are superfluous because they provide response information or excessive granularity. Ultimately, 50 variables are removed. The full list is available in the appendix code. Below we summarize notable excisions:

- We remove many variables that provide information about the number and type of violations because these are response variables. Including them helps our model "cheat" by telegraphing the pass/fail outcome through highly correlated variables.
- We remove Inspection.ID, DBA.Name, AKA.Name" because while idiosyncratic variables can be useful, embedding predictive information not explicitly incorporated into our model, these variables are redundant to or less useful than other unique identifiers such as "license" (for business license). "Inspection.ID", for example, which takes as many values as there are observations in the dataset, provides superfluous detail.
- We remove businesses addresses because our datasets were joined by zip code and by month-and-year. Address is a level of geography not helpful to our model.

Finally, we convert into factors many of the variables (see appendix) and then run logistic regression, random forests, boosted trees, and neural networks. We evaluate models using five metrics: confusion matrices, ROC, AUC, lift curve, and variable importance.

4.1 Logistic Regression

The GLM model has accuracy of 58.1%. The ROC Curve for training set, ROC Curve for validation set, Gain/lift training set table, Gain/lift validation set table, the Confusion matrix, and the Variable Importance Tables have been provided in the appendix.

4.2 Random Forest

The random forest model with 3000 trees has accuracy of 64.5%. The ROC Curve for training set, ROC Curve for validation set, Gain/lift training set table, Gain/lift validation set table, the Confusion matrix, and the Variable Importance Tables have been provided in the appendix.

4.3 Boosted Trees

The boosted trees model, which we let run for 30 minutes, has accuracy of 64.5%. The ROC Curve for training set, ROC Curve for validation set, Gain/lift training set table, Gain/lift validation set table, the Confusion matrix, and the Variable Importance Tables have been provided in the appendix.

4.4 Neural Nets

The neural net model has accuracy of 62.3%. The ROC Curve for training set, ROC Curve for validation set, Gain/lift training set table, Gain/lift validation set table, the Confusion matrix, and the Variable Importance Tables have been provided in the appendix.

5. Discussion, Interpretation, and Limitations

5.1 Evaluation of Models

Boosted trees and random forests models perform best with accuracy of 64.5% and uplift of more than 2.5 in the top 10 percent of observations. Neural net performs next-best, and logistic regression performs worst.

The boosted trees and random forests models each ascribe the greatest importance to different variables. Boosted trees' most important variables are previous failing results, month and year, the number of past violations, and the Zillow price index. Random forests' most important variables are license, zip code, the previous inspection's result, and facility type.

Notably, the boosted trees and random forests models have different confusion matrices. Boosted trees' false negative rate is lower at .345913, but its false positive rate is higher at .357213. Random forest meanwhile has virtually identical false positive and negative rates at .354742. These differences are small but important. We believe false negatives are more damaging than false positives in this context. It is worse for a business overlooked by our model to purvey unsanitary food than it is for a business to be redundantly inspected. The best model for the purposes of this project is the boosted trees model.

5.2 Limitations

While we are pleased that our models show some ability to predict inspection results, there are several limitations of our data, methods, and resources that are important to consider.

First of all, the machine learning methods we used are powerful tools for prediction but often struggle with interpretability. For example, though the Zillow Housing Index variable is important to both boosted trees and random forests, it is unclear which Index values – smaller or larger – are predictive of an

increased likelihood of failing a food inspection. Ultimately, we have created models which are better than guesses at predicting food inspection failure. But the usefulness of these models is bound up in their implicit and explicit assumptions.

Our explicit assumptions relate mostly to how we joined data. Our data was aggregated over zip code, year, and month. If we had joined upon different values – days of the week or wards – our predictive output may have been different. We thought monthly aggregation to be the most helpful for policy-makers because we assumed that inspections are assigned to personnel on something like a monthly basis. Further work could be done modeling inspection predictions by day, or days from last inspection, or year, which might create an even more agile and accurate prediction tool for the City of Chicago.

Similarly, certain data sources had different temporal aspects than others. Inspections were tied to a date over the last ~10 years while ShotSpotter only existed starting in 2017, and the grocery data is only a point-in-time snapshot from 2013. To deal with these practical challenges, we had to make a series of judgment calls for which data to include and how to aggregate it.

A further limitation is the selection bias of observations. While every business in Chicago is inspected annually, the timing of this inspection may be arbitrary but influential. For example, while month and year are found to be important variables in our models, it is unclear whether these time periods are predictive because certain businesses inherently more likely to fail were inspected at those times, or because an overzealous inspector happened to be on the job, or because (improbably) some month and year for some other reason predicted inspection failures. Additionally, a significant portion of our observations is triggered by a complaint rather than a canvass inspection, meaning that we may have more information for those establishments than we do for establishments that do not get inspected after complaints.

The selection bias in the data also creates an unbalanced data problem. The City already uses a predictive risk model to prioritize establishments for inspection. Establishments that are judged to be “high risk” are supposed to be inspected twice a year instead of once (though many are not, possibly due to limited resources), meaning that observations are much more likely to be classified as fails than if the inspections were targeted randomly, and thus the distributions of the various features we used will differ in the observed data, potentially in some meaningful way, from their true distribution. With additional time and resources, we would want to carefully consider how to address this for both prediction and inference purposes through some kind of oversampling methodology.

Another limitation of our data is the use of license numbers as the unique business identifier. We noticed that some businesses with similar or identical names ceased to exist under an old license number and then reappeared under a new one. This may be something that happens in the normal course of operations for these businesses, or it could be that businesses try to “dodge” their inspection track record by operating under a new license. Either way, this presents a challenge for prediction. Being able to correctly link these instances would likely allow us to improve our models.

5.4 Lessons Learned

In terms of predicting food inspections, we learned valuable lessons that might be applicable to the City of Chicago's targeting in the future. Because our models assign different levels of importance to different variables, we find variously that the following features are predictive: past inspection results, zip codes, Zillow prices, thefts, business licenses, facility type, and numbers of past violations.

It is difficult to aggregate data from disparate sources. Technological constraints are sometimes intractable. Having gathered 150 predictors, for example, we attempted to run Boruta but found the process unwieldy and interminable. Later, after winnowing down our predictors, we ran it again to further refine our variable selection. The results are interesting but not especially informative because some of the variables removed – human trafficking crimes, for example – have fewer few recorded instances. Ultimately, Boruta uses 66 of our chosen variables, removes 13, and regards as ambiguous the influence of 11 others.

However, arguably the most important lesson we learned in conducting this project was about using real-world data for machine learning problems. Unlike in a classroom setting in which the data is already cleaned and prepared for use, the real world is not so kind. We spent a great deal of time and effort searching for, choosing, cleaning, and engineering new features in our data just to get it to the point where we could run our models. We now have a greater appreciation for these issues, and we understand that when we apply these methods after we graduate, a substantial portion – the vast majority even – of the time spent on these projects will be invested in acquiring, cleaning, and augmenting our data.

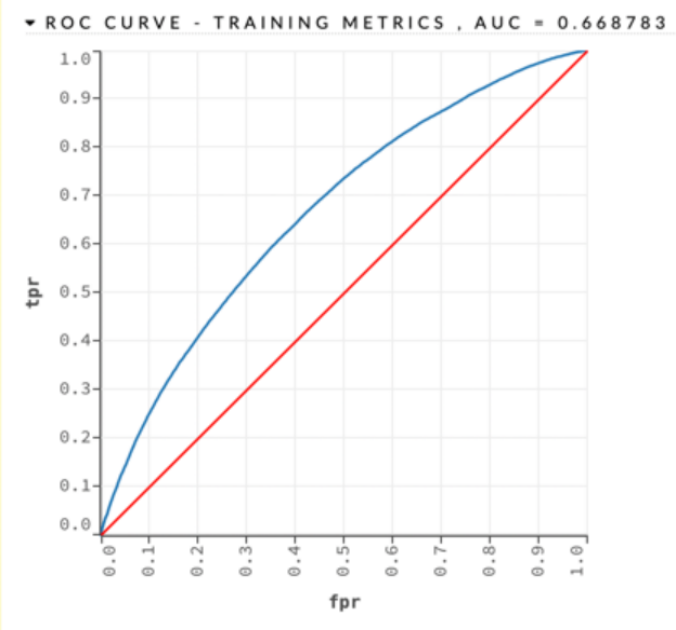
6. Conclusion

Our model predicts failing food inspections with nearly 65% accuracy. For the top decile of observations, our model improves over random targeting by a lift value of 2.5 and greater. We consider this a success with applications beyond the course.

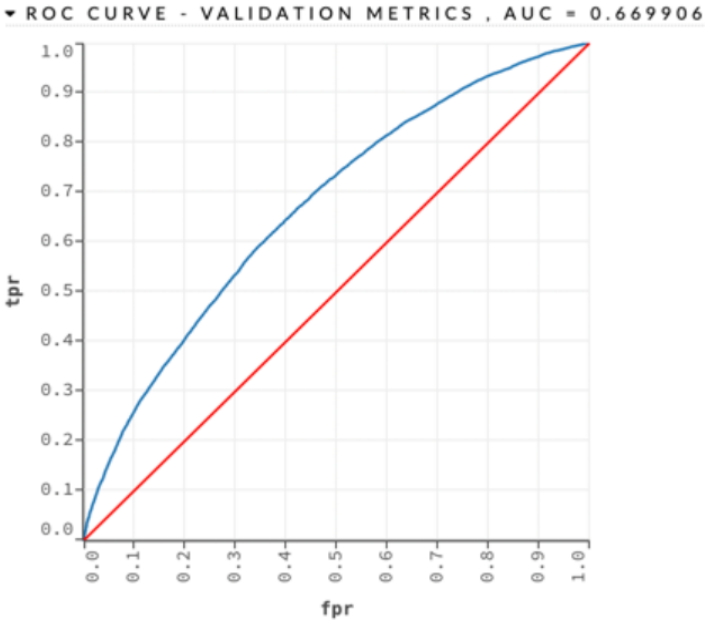
It is difficult to bench-mark the rate at which the City of Chicago successfully predicts failing food inspections, but given only 23% of high-risk observations have received failing inspection results according to public data, we optimistically hope our model can improve upon the City's current method of targeting. In FY2021, Chicago budgeted $6 million for food inspections. If our model achieves even a marginal efficiency improvement of 10%, it would save taxpayer money and improve public health.

Appendix

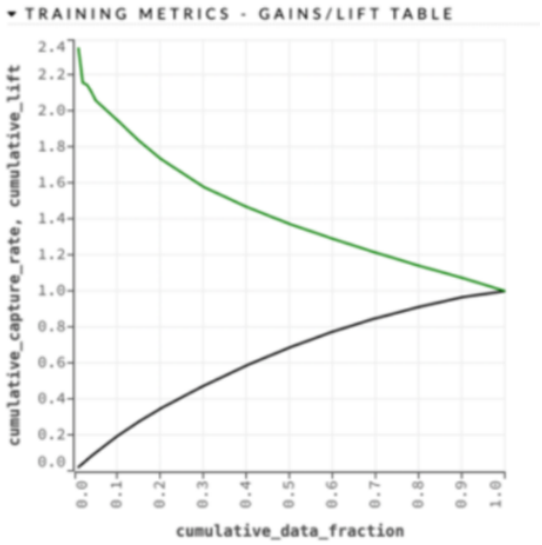
1.1 ROC Curve training set (Logistic Regression):



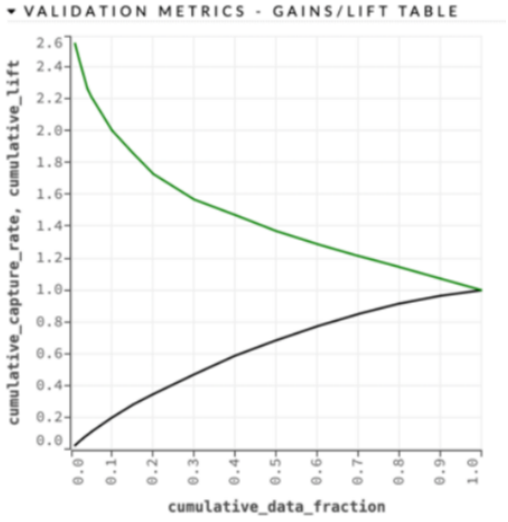
1.2 ROC Curve validation set (Logistic Regression):



1.3 Gain/lift training set table (Logistic Regression):



1.4 Gain/lift validation set table (Logistic Regression):



1.5 Confusion matrix (Logistic Regression):

Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.21926967024011:

	0	1	Error	Rate
0	18492	15093	0.449397	=15093/33585
1	2990	6602	0.311718	=2990/9592
Totals	21482	21695	0.418811	=18083/43177

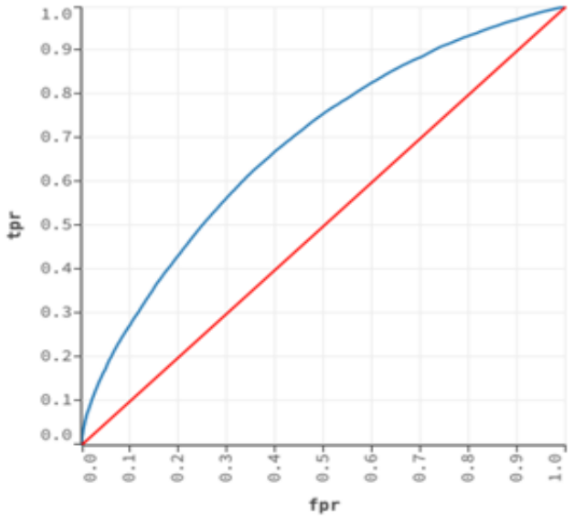
1.6 Variable importance (Logistic Regression):

▾ OUTPUT - VARIABLE IMPORTANCES

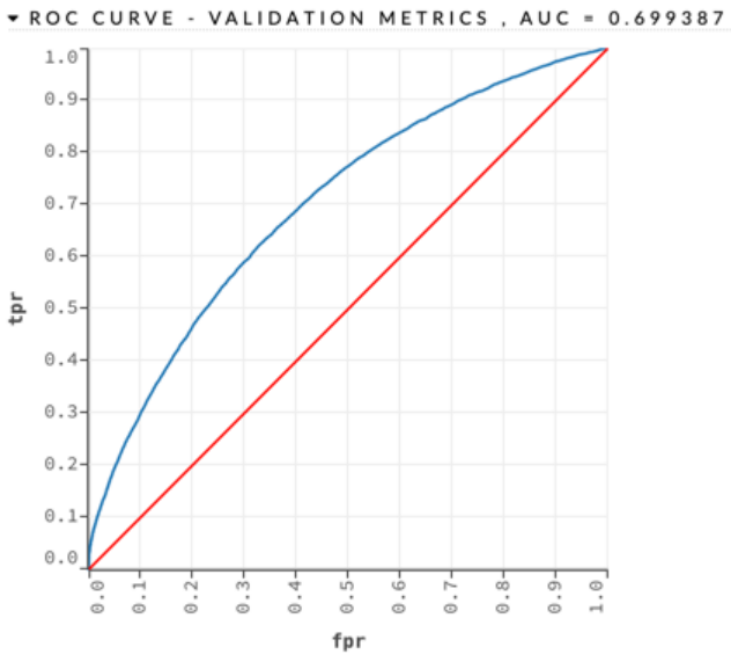
variable	relative_importance	scaled_importance	percentag
prev_result.Fail	0.9079	1.0	0.031
prev_result.Pass w/ Conditions	0.3476	0.3829	0.012
prev_._violations	0.2632	0.2899	0.009
prev_result.nan	0.2578	0.2840	0.009
prev_inspection_type.nan	0.2578	0.2840	0.009
Facility.Type.Childrens_Services	0.2073	0.2283	0.007
prev_result.Pass	0.1916	0.2110	0.006
Inspection.Type_CANVASS.1	0.1728	0.1904	0.006
prev_inspection_type.CANVASS	0.1678	0.1849	0.005
Inspection.Type_CANVASS.0	0.1666	0.1835	0.005
Success_Canvass_prev	0.1571	0.1730	0.005
Inspection.Type_LICENSE.1	0.1568	0.1727	0.005
Success_Complaint_prev	0.1226	0.1350	0.004
Fail_Canvass_prev	0.1145	0.1261	0.004
Risk.Risk 3 (Low)	0.1125	0.1239	0.003
Inspection.Type_TASKFORCE.1	0.1085	0.1195	0.003
Zip.60657	0.1064	0.1171	0.003
Zip.60618	0.1040	0.1146	0.003
Inspection.Type_LICENSE.0	0.1022	0.1126	0.003
Zip.60634	0.0997	0.1098	0.003
Zip.60614	0.0993	0.1094	0.003
Zip.60619	0.0985	0.1085	0.003

2.1 ROC Curve Training Set (Random Forest):

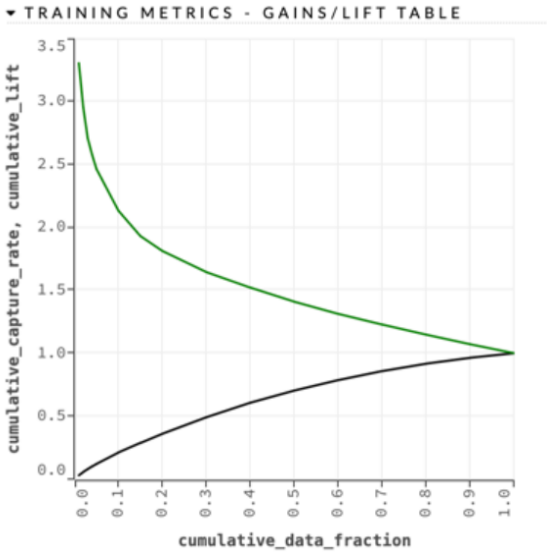
▾ ROC CURVE - TRAINING METRICS , AUC = 0.684713



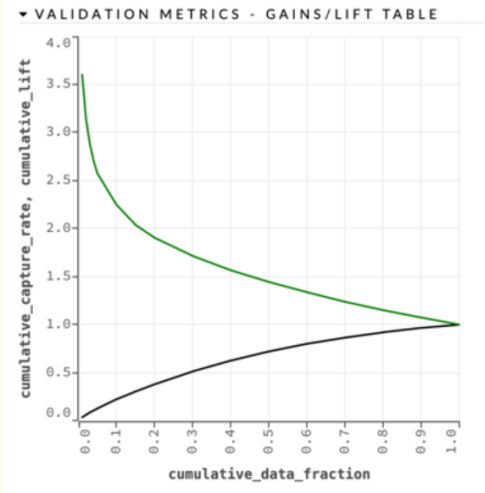
2.2 ROC Curve Validation Set (Random Forest):



2.3 Gain/lift Training Set Table (Random Forest):



2.4 Gain/lift Validation Set Table (Random Forest):

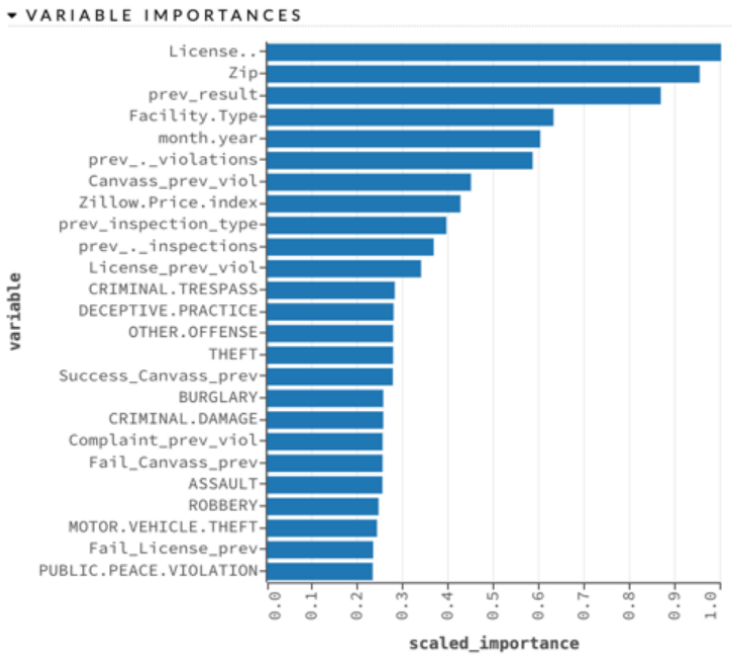


2.5 Confusion Matrix (Random Forest):

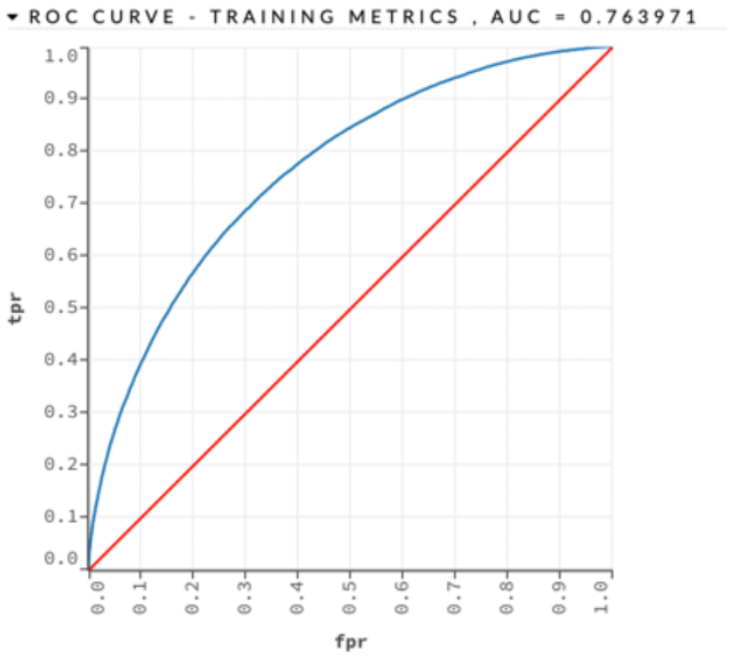
Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.230179127646882:

	0	1	Error	Rate
0	21671	11914	0.354742	=11914/33585
1	3398	6194	0.354254	=3398/9592
Totals	25069	18108	0.354633	=15312/43177

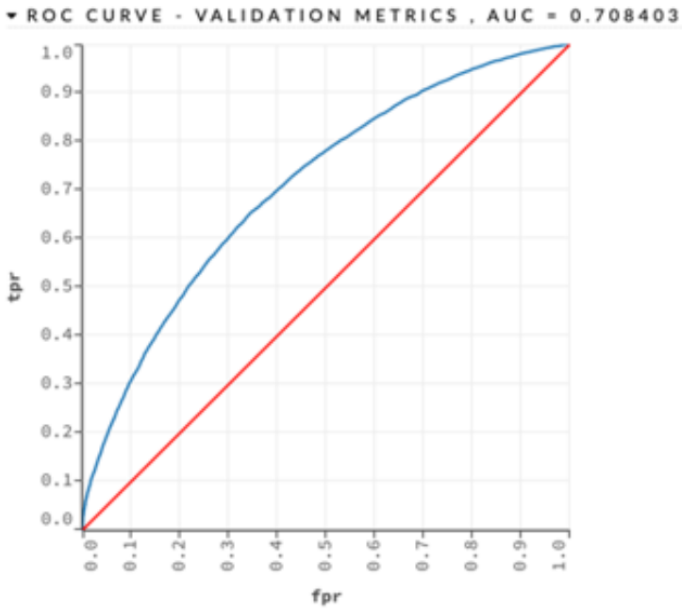
2.6 Variable Importance (Random Forest):



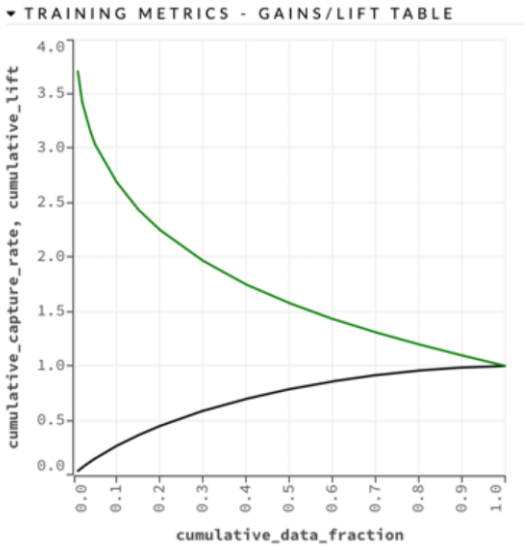
3.1 ROC Curve Training Set (Boosted Tree):



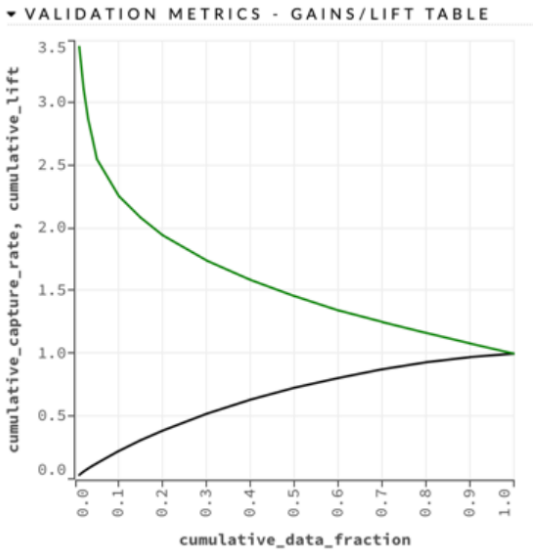
3.2 ROC Curve Validation Set (Boosted Tree):



3.3 Gain/lift Training Set Table (Boosted Tree):



3.4 Gain/lift Validation Set Table (Boosted Tree):

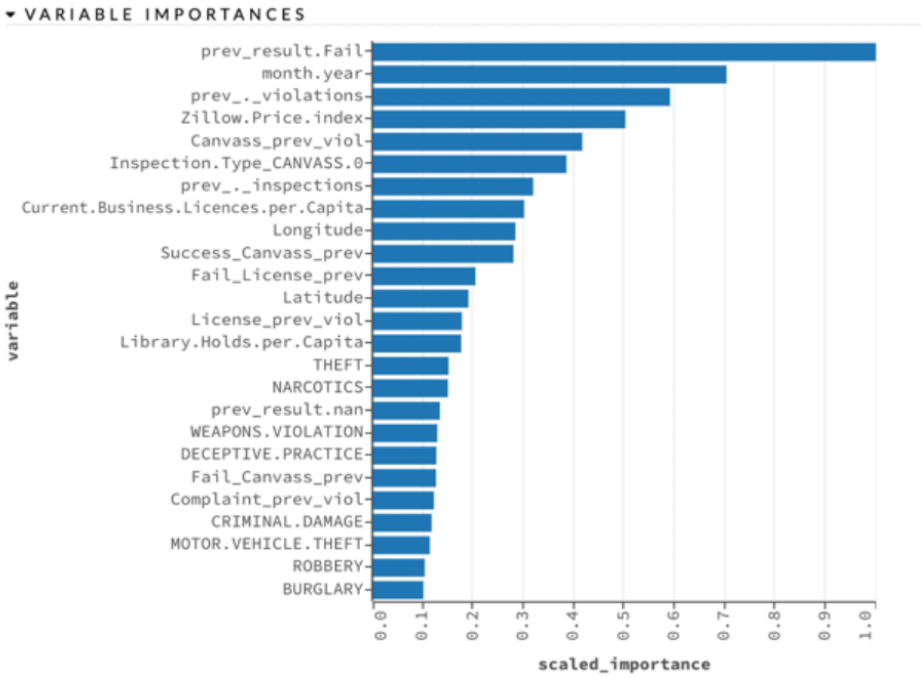


3.5 Confusion Matrix (Boosted Tree):

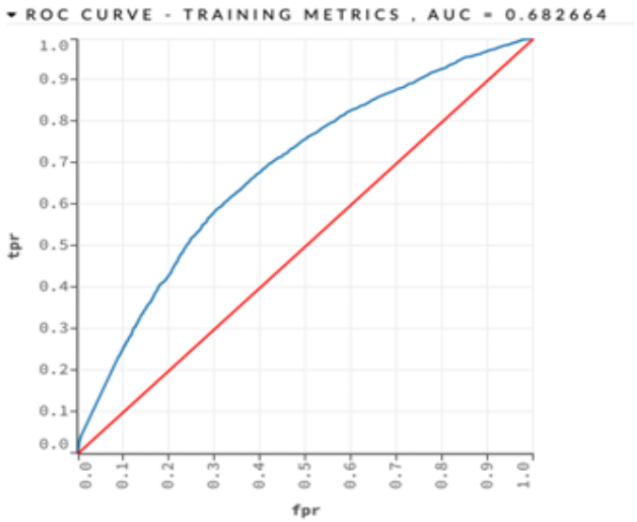
```
Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.224711013708528:
```

	0	1	Error	Rate
0	21588	11997	0.357213	=11997/33585
1	3318	6274	0.345913	=3318/9592
Totals	24906	18271	0.354703	=15315/43177

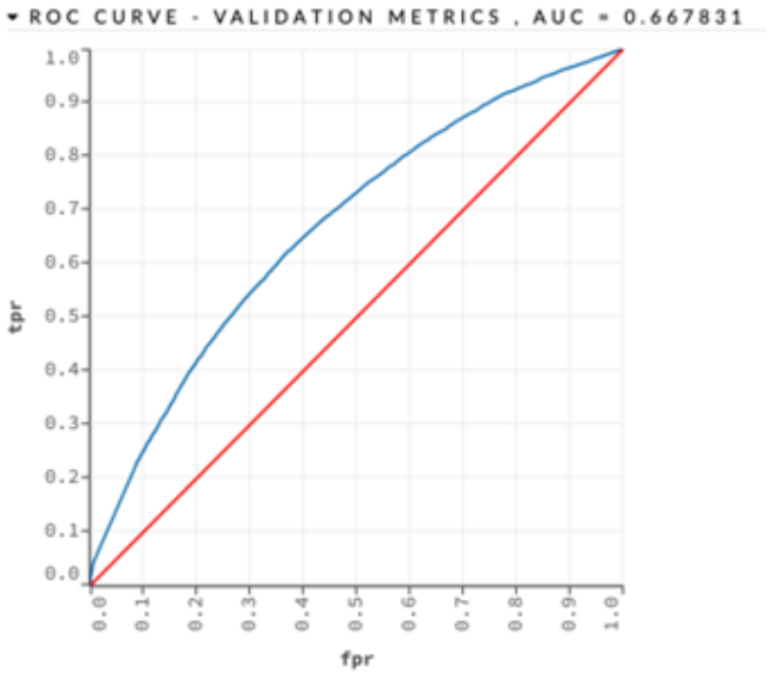
3.6 Variable Importance (Boosted Tree):



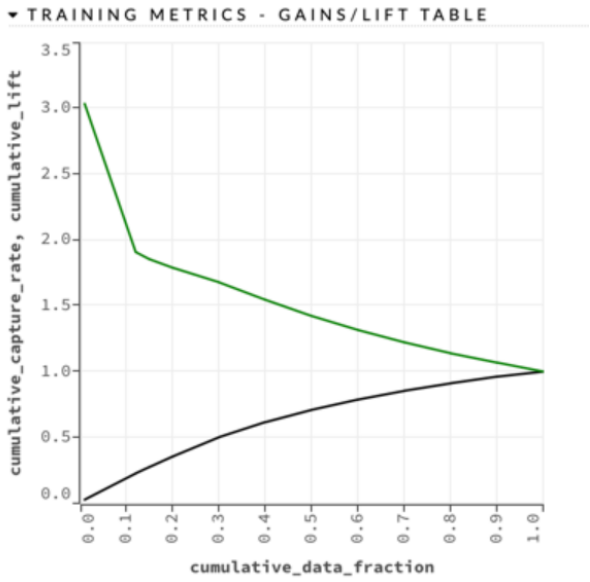
4.1 ROC Curve Training Set (Neural Nets):



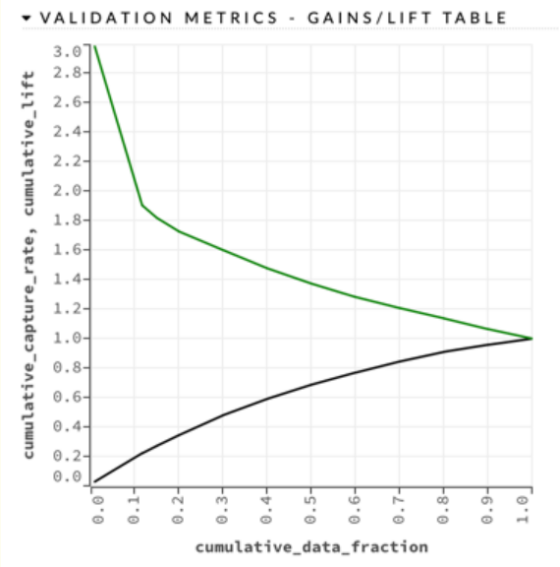
4.2 ROC Curve Validation Set (Neural Nets):



4.3 Gain/lift Training Set Table (Neural Nets):



4.4 Gain/lift Validation Set Table (Neural Nets):

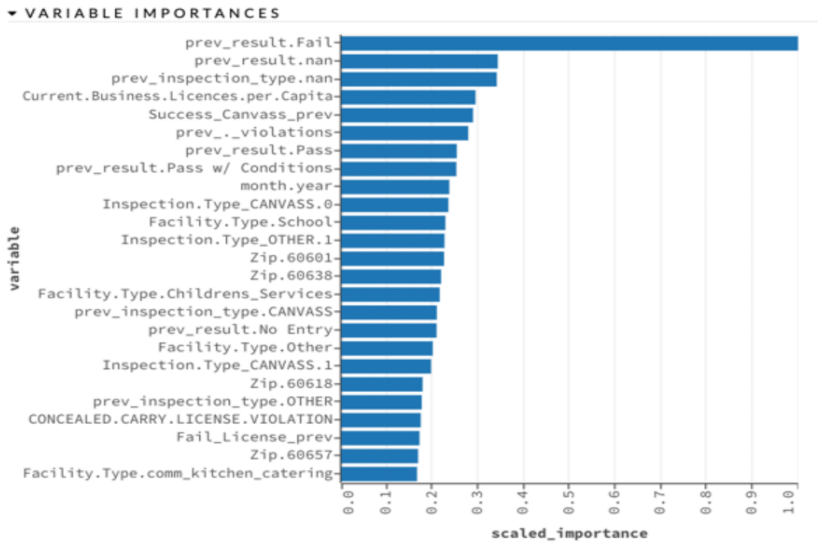


4.5 Confusion Matrix (Neural Nets):

Confusion Matrix (vertical: actual; across: predicted) for max f1 @ threshold = 0.341372565047091:

	0	1	Error	Rate
0	20939	12646	0.376537	=12646/33585
1	3636	5956	0.379066	=3636/9592
Totals	24575	18602	0.377099	=16282/43177

4.6 Variable Importance (Neural Nets):



Final Project

Food Inspections

Alex Bue Christian Gregorich Conrad Liu Ujjwal Sehrawat

Winter 2023

Contents

BUSN 41204-02 | Machine Learning

Final Project: Food Inspections

Winter 2023

Professor Mladen Kolar

Alex Bue

Christian Gregorich

Conrad Liu

Ujjwal Sehrawat

We pledge our honor that we have not violated the honor code during this assignment

```
library(h2o)
library(data.table)
library(Boruta)
```

```
# use all threads by default
h2oServer <- h2o.init()
```

```
H2O is not running yet, starting it now...

Note:  In case of errors look at the following log files:
    /var/folders/wl/5tng8yp94jz569jtkq5s75pm0000gn/T//RtmpeiZt68/file143e92d12b527/h2o_Conrad_started_f
    /var/folders/wl/5tng8yp94jz569jtkq5s75pm0000gn/T//RtmpeiZt68/file143e927332a8f/h2o_Conrad_started_f


Starting H2O JVM and connecting: ........ Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         9 seconds 337 milliseconds
    H2O cluster timezone:       America/Chicago
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.40.0.1
    H2O cluster version age:    1 month and 1 day
    H2O cluster name:           H2O_started_from_R_Conrad_lqj474
    H2O cluster total nodes:    1
    H2O cluster total memory:   3.54 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    R Version:                  R version 4.2.2 (2022-10-31)
```

```
h2o.show_progress()
```

```
new_df = data.table(read.csv("finalmlproj.csv"))

# only keep rows that are either passes or failures, not
# non-inspections.
new_df = new_df[Results_Pass == 1 | Results_Fail == 1 | Results_Pass.w..Conditions ==
  1]
```

```
# create single 1/0 Results variable.  Results == 1 means a
# failure (a 'positive' result) Results == 0 means the
# inspection passed (a 'negative' result)
new_df[Results_Pass == 1 | Results_Pass.w..Conditions == 1, `:=`(Results,
  0)]
new_df[Results_Fail == 1, `:=`(Results, 1)]

# substitute missing values
columnsWithMissingValues = c(
  "Zillow.Price.index", "BATTERY", "CRIMINAL.DAMAGE", "MOTOR.VEHICLE.THEFT",
  "ROBBERY", "SEX.OFFENSE", "ASSAULT", "BURGLARY", "CRIM.SEXUAL.ASSAULT",
  "CRIMINAL.TRESPASS", "DECEPTIVE.PRACTICE", "NARCOTICS", "OTHER.OFFENSE",
  "PUBLIC.PEACE.VIOLATION", "THEFT", "ARSON", "HOMICIDE", "INTERFERENCE.WITH.PUBLIC.OFFICER",
  "INTIMIDATION", "LIQUOR.LAW.VIOLATION", "OFFENSE.INVOLVING.CHILDREN",
  "PROSTITUTION", "STALKING", "WEAPONS.VIOLATION", "KIDNAPPING",
  "GAMBLING", "OBSCENITY", "OTHER.NARCOTIC.VIOLATION", "CRIMINAL.SEXUAL.ASSAULT",
  "PUBLIC.INDECENCY", "NON.CRIMINAL", "HUMAN.TRAFFICKING",
  "NON.CRIMINAL..SUBJECT.SPECIFIED.", "NON...CRIMINAL", "CONCEALED.CARRY.LICENSE.VIOLATION",
  "RITUALISM", "X2020.Population", "Current.Business.Licences",
  "Current.Business.Licences.per.Capita", "X2013.Grocery.Stores",
  "X2013.Grocery.Stores.per.Capita", "Library.Holds", "Library.Holds.per.Capita",
  "Shotspotter.Alerts.per.Capita", "Shotspotter.Alerts", "ShotSpotter.Shots",
  "ShotSpotter.Shots.per.Capita"
)

for (column in columnsWithMissingValues) {
  meanValue = new_df[, mean(
    get(column),
    na.rm = TRUE
  )]
  new_df[is.na(get(column)),
    `:=`(c(column),
      meanValue)]
}

# drop columns that are really response variables, or other
# duplicate columns
columnsToDrop = c(
  "pass", "fail", "other", "canvass_pass", "canvass_fail",
  "canvass_other", "complaint_pass", "complaint_fail", "complaint_other",
  "consultation_pass", "consultation_fail", "consultation_other",
  "license_pass", "license_fail", "license_other", "taskforce_pass",
  "taskforce_fail", "taskforce_other", "other_itype_pass",
  "other_itype_fail", "other_itype_other", "X.violations",
```

```
  "canvass_viol", "complaint_viol", "consultation_viol", "license_viol",
  "taskforce_viol", "other_viol", "Violations", "Results_Business.Not.Located",
  "Results_Fail", "Results_No.Entry", "Results_Not.Ready",
  "Results_Out.of.Business", "Results_Pass", "Results_Pass.w..Conditions",
  "Facility.Type_Old", "X", "Unnamed..0", "Inspection.ID",
  "DBA.Name", "AKA.Name", "Address", "Zipcode", "Inspection.Date",
  "Location", "previous_month_year", "Zillow.Date", "dbaname.address",
  "ascii"
)

for (column in columnsToDrop) {
  new_df[, `:=`(c(column),
    NULL)]
}

# make these columns into factors
columnsToFactorize = c(
  "Results", "License..", "Facility.Type", "Risk", "City",
  "State", "Zip", "prev_inspection_type", "prev_result", "Inspection.Type_CANVASS",
  "Inspection.Type_COMPLAINT", "Inspection.Type_CONSULTATION",
  "Inspection.Type_LICENSE", "Inspection.Type_OTHER", "Inspection.Type_TASKFORCE"
)

for (column in columnsToFactorize) {
  new_df[, `:=`(c(column),
    factor(get(column)))]
}
```

```
food_inspect_df = as.h2o(new_df, destination_frame = "Food_Inspections")
```

```
  |                                                                              |
```

```
food_inspect_split = h2o.splitFrame(
  data = food_inspect_df, destination_frames = c(
    "Food_Inspections_train", "Food_Inspections_validate",
    "Food_Inspections_test"
  ),
  ratios = c(0.6, 0.2),
  seed = 41204
)

response = "Results"
predictors = setdiff(
  names(food_inspect_df),
  response
```

```
)

food_inspect_train = food_inspect_split[[1]]
food_inspect_validate = food_inspect_split[[2]]
food_inspect_test = food_inspect_split[[3]]
```

```
set.seed(41204)
inVars = sample(
  nrow(food_inspect_train),
  nrow(food_inspect_train) *
    0.3
)
borutaData = as.data.table(food_inspect_train)[inVars,
  ]
(boruta = Boruta(Results ~ ., data = borutaData, maxRuns = 15, doTrace = 0))
```

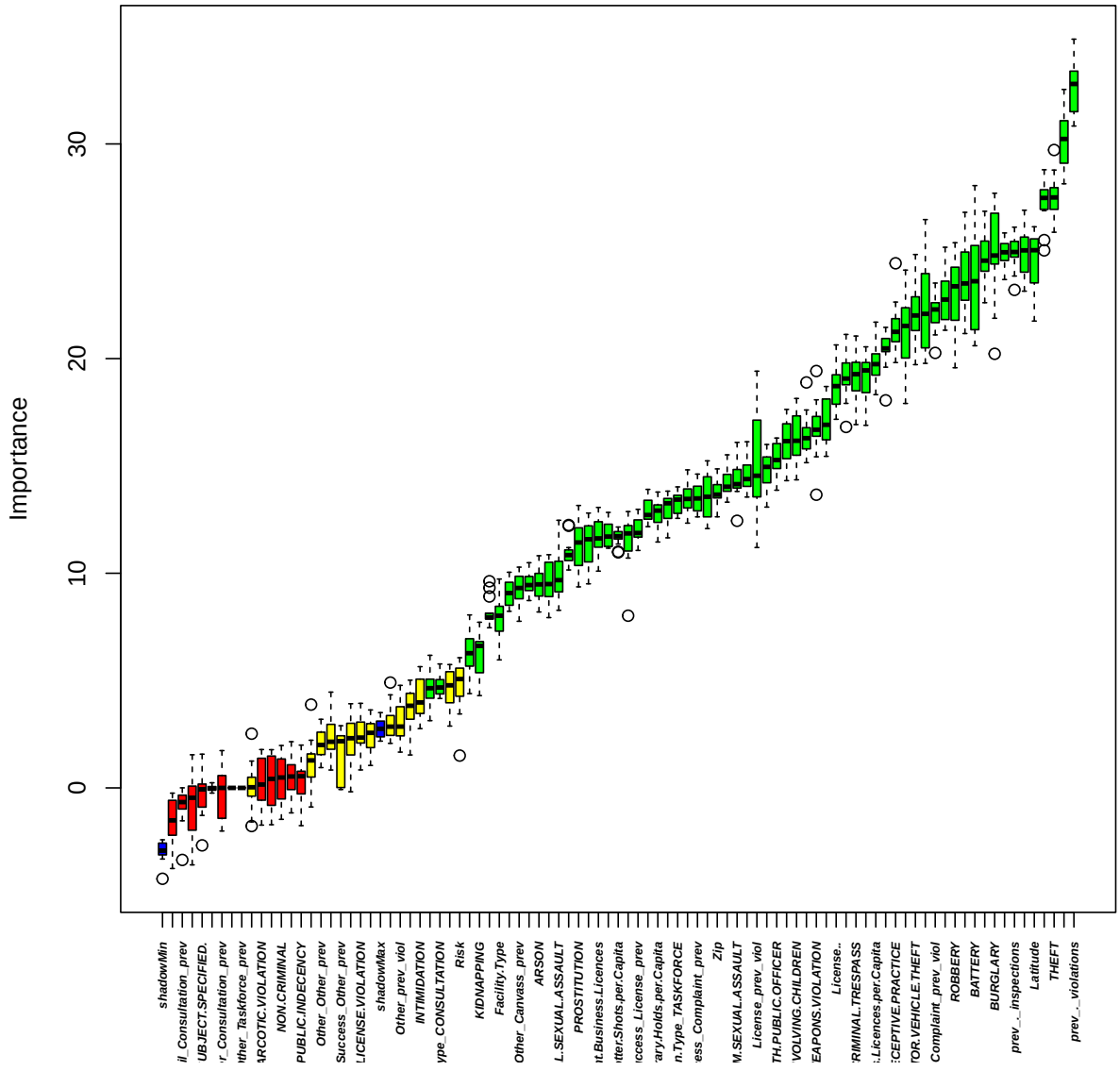
```
Computing permutation importance.. Progress: 18%. Estimated remaining time: 2 minutes, 23 seconds.
Computing permutation importance.. Progress: 39%. Estimated remaining time: 1 minute, 38 seconds.
Computing permutation importance.. Progress: 58%. Estimated remaining time: 1 minute, 7 seconds.
Computing permutation importance.. Progress: 77%. Estimated remaining time: 37 seconds.
Computing permutation importance.. Progress: 95%. Estimated remaining time: 8 seconds.
Computing permutation importance.. Progress: 15%. Estimated remaining time: 3 minutes, 7 seconds.
Computing permutation importance.. Progress: 33%. Estimated remaining time: 2 minutes, 6 seconds.
Computing permutation importance.. Progress: 52%. Estimated remaining time: 1 minute, 27 seconds.
Computing permutation importance.. Progress: 73%. Estimated remaining time: 47 seconds.
Computing permutation importance.. Progress: 94%. Estimated remaining time: 10 seconds.
Computing permutation importance.. Progress: 19%. Estimated remaining time: 2 minutes, 8 seconds.
Computing permutation importance.. Progress: 39%. Estimated remaining time: 1 minute, 39 seconds.
Computing permutation importance.. Progress: 58%. Estimated remaining time: 1 minute, 9 seconds.
Computing permutation importance.. Progress: 77%. Estimated remaining time: 38 seconds.
Computing permutation importance.. Progress: 96%. Estimated remaining time: 5 seconds.
Computing permutation importance.. Progress: 19%. Estimated remaining time: 2 minutes, 8 seconds.
Computing permutation importance.. Progress: 41%. Estimated remaining time: 1 minute, 27 seconds.
Computing permutation importance.. Progress: 63%. Estimated remaining time: 55 seconds.
Computing permutation importance.. Progress: 85%. Estimated remaining time: 22 seconds.
Computing permutation importance.. Progress: 20%. Estimated remaining time: 2 minutes, 4 seconds.
Computing permutation importance.. Progress: 41%. Estimated remaining time: 1 minute, 30 seconds.
Computing permutation importance.. Progress: 54%. Estimated remaining time: 1 minute, 17 seconds.
Computing permutation importance.. Progress: 61%. Estimated remaining time: 1 minute, 20 seconds.
Computing permutation importance.. Progress: 69%. Estimated remaining time: 1 minute, 11 seconds.
Computing permutation importance.. Progress: 76%. Estimated remaining time: 1 minute, 0 seconds.
Computing permutation importance.. Progress: 83%. Estimated remaining time: 43 seconds.
Computing permutation importance.. Progress: 91%. Estimated remaining time: 23 seconds.
Growing trees.. Progress: 60%. Estimated remaining time: 20 seconds.
Computing permutation importance.. Progress: 7%. Estimated remaining time: 7 minutes, 18 seconds.
```

```
Computing permutation importance.. Progress: 15%. Estimated remaining time: 6 minutes, 14 seconds.
Computing permutation importance.. Progress: 23%. Estimated remaining time: 5 minutes, 25 seconds.
Computing permutation importance.. Progress: 32%. Estimated remaining time: 4 minutes, 33 seconds.
Computing permutation importance.. Progress: 42%. Estimated remaining time: 3 minutes, 47 seconds.
Computing permutation importance.. Progress: 50%. Estimated remaining time: 3 minutes, 13 seconds.
Computing permutation importance.. Progress: 58%. Estimated remaining time: 2 minutes, 42 seconds.
Computing permutation importance.. Progress: 67%. Estimated remaining time: 2 minutes, 8 seconds.
Computing permutation importance.. Progress: 75%. Estimated remaining time: 1 minute, 35 seconds.
Computing permutation importance.. Progress: 84%. Estimated remaining time: 1 minute, 2 seconds.
Computing permutation importance.. Progress: 92%. Estimated remaining time: 32 seconds.
Computing permutation importance.. Progress: 21%. Estimated remaining time: 1 minute, 56 seconds.
Computing permutation importance.. Progress: 43%. Estimated remaining time: 1 minute, 20 seconds.
Computing permutation importance.. Progress: 67%. Estimated remaining time: 46 seconds.
Computing permutation importance.. Progress: 90%. Estimated remaining time: 14 seconds.
Computing permutation importance.. Progress: 22%. Estimated remaining time: 1 minute, 47 seconds.
Computing permutation importance.. Progress: 44%. Estimated remaining time: 1 minute, 17 seconds.
Computing permutation importance.. Progress: 67%. Estimated remaining time: 46 seconds.
Computing permutation importance.. Progress: 89%. Estimated remaining time: 15 seconds.
Computing permutation importance.. Progress: 23%. Estimated remaining time: 1 minute, 44 seconds.
Computing permutation importance.. Progress: 46%. Estimated remaining time: 1 minute, 12 seconds.
Computing permutation importance.. Progress: 69%. Estimated remaining time: 41 seconds.
Computing permutation importance.. Progress: 93%. Estimated remaining time: 9 seconds.
Computing permutation importance.. Progress: 23%. Estimated remaining time: 1 minute, 46 seconds.
Computing permutation importance.. Progress: 47%. Estimated remaining time: 1 minute, 12 seconds.
Computing permutation importance.. Progress: 71%. Estimated remaining time: 39 seconds.
Computing permutation importance.. Progress: 93%. Estimated remaining time: 9 seconds.
Computing permutation importance.. Progress: 23%. Estimated remaining time: 1 minute, 46 seconds.
Computing permutation importance.. Progress: 46%. Estimated remaining time: 1 minute, 11 seconds.
Computing permutation importance.. Progress: 70%. Estimated remaining time: 40 seconds.
Computing permutation importance.. Progress: 93%. Estimated remaining time: 9 seconds.
Computing permutation importance.. Progress: 22%. Estimated remaining time: 1 minute, 51 seconds.
Computing permutation importance.. Progress: 45%. Estimated remaining time: 1 minute, 17 seconds.
Computing permutation importance.. Progress: 68%. Estimated remaining time: 44 seconds.
Computing permutation importance.. Progress: 90%. Estimated remaining time: 14 seconds.
Computing permutation importance.. Progress: 23%. Estimated remaining time: 1 minute, 49 seconds.
Computing permutation importance.. Progress: 46%. Estimated remaining time: 1 minute, 12 seconds.
Computing permutation importance.. Progress: 66%. Estimated remaining time: 48 seconds.
Computing permutation importance.. Progress: 85%. Estimated remaining time: 22 seconds.
Computing permutation importance.. Progress: 19%. Estimated remaining time: 2 minutes, 8 seconds.
Computing permutation importance.. Progress: 39%. Estimated remaining time: 1 minute, 36 seconds.
Computing permutation importance.. Progress: 57%. Estimated remaining time: 1 minute, 10 seconds.
Computing permutation importance.. Progress: 77%. Estimated remaining time: 36 seconds.
Computing permutation importance.. Progress: 98%. Estimated remaining time: 3 seconds.

Boruta performed 14 iterations in 46.13862 mins.
```

```
 64 attributes confirmed important: ARSON, ASSAULT, BATTERY, BURGLARY,
Canvass_prev_viol and 59 more;
 12 attributes confirmed unimportant: City, Fail_Consultation_prev,
Fail_Other_prev, HUMAN.TRAFFICKING, NON.CRIMINAL and 7 more;
 14 tentative attributes left: CONCEALED.CARRY.LICENSE.VIOLATION,
Consultation_prev_viol, INTIMIDATION, NON...CRIMINAL, OBSCENITY and 9
more;
```

```r
plot(boruta, xlab = "", xaxt = "n")
lz = lapply(
  1:ncol(boruta$ImpHistory),
  function(i) boruta$ImpHistory[is.finite(boruta$ImpHistory[, i]),
    i]
)
names(lz) = colnames(boruta$ImpHistory)
lb = sort(sapply(lz, median))
axis(
  side = 1, las = 2, labels = names(lb),
  at = 1:ncol(boruta$ImpHistory),
  cex.axis = 0.5, font = 4
)
```




```
confirmed_vars = names(boruta$finalDecision)[boruta$finalDecision %in%
  c("Confirmed")]
print(confirmed_vars)
```

```
 [1] "License.."
 [2] "Facility.Type"
 [3] "Latitude"
 [4] "Longitude"
 [5] "month.year"
 [6] "Zillow.Price.index"
 [7] "BATTERY"
```

```
 [8] "CRIMINAL.DAMAGE"
 [9] "MOTOR.VEHICLE.THEFT"
[10] "ROBBERY"
[11] "SEX.OFFENSE"
[12] "ASSAULT"
[13] "BURGLARY"
[14] "CRIM.SEXUAL.ASSAULT"
[15] "CRIMINAL.TRESPASS"
[16] "DECEPTIVE.PRACTICE"
[17] "NARCOTICS"
[18] "OTHER.OFFENSE"
[19] "PUBLIC.PEACE.VIOLATION"
[20] "THEFT"
[21] "ARSON"
[22] "HOMICIDE"
[23] "INTERFERENCE.WITH.PUBLIC.OFFICER"
[24] "LIQUOR.LAW.VIOLATION"
[25] "OFFENSE.INVOLVING.CHILDREN"
[26] "PROSTITUTION"
[27] "STALKING"
[28] "WEAPONS.VIOLATION"
[29] "KIDNAPPING"
[30] "GAMBLING"
[31] "CRIMINAL.SEXUAL.ASSAULT"
[32] "Zip"
[33] "X2020.Population"
[34] "Current.Business.Licences"
[35] "Current.Business.Licences.per.Capita"
[36] "X2013.Grocery.Stores"
[37] "X2013.Grocery.Stores.per.Capita"
[38] "Library.Holds"
[39] "Library.Holds.per.Capita"
[40] "Shotspotter.Alerts.per.Capita"
[41] "Shotspotter.Alerts"
[42] "ShotSpotter.Shots"
[43] "ShotSpotter.Shots.per.Capita"
[44] "prev_inspection_type"
[45] "prev_result"
[46] "prev_._inspections"
[47] "prev_._violations"
[48] "Inspection.Type_CANVASS"
[49] "Inspection.Type_COMPLAINT"
[50] "Inspection.Type_CONSULTATION"
[51] "Inspection.Type_LICENSE"
[52] "Inspection.Type_OTHER"
```

```
[53] "Inspection.Type_TASKFORCE"
[54] "Success_Canvass_prev"
[55] "Fail_Canvass_prev"
[56] "Other_Canvass_prev"
[57] "Success_Complaint_prev"
[58] "Fail_Complaint_prev"
[59] "Success_License_prev"
[60] "Fail_License_prev"
[61] "Fail_Taskforce_prev"
[62] "Canvass_prev_viol"
[63] "Complaint_prev_viol"
[64] "License_prev_viol"
```

```
length(confirmed_vars)
```

```
[1] 64
```

```
tentative_vars = names(boruta$finalDecision)[boruta$finalDecision %in%
  c("Tentative")]
print(tentative_vars)
```

```
 [1] "Risk"                                "INTIMIDATION"
 [3] "OBSCENITY"                           "NON...CRIMINAL"
 [5] "CONCEALED.CARRY.LICENSE.VIOLATION" "Other_Complaint_prev"
 [7] "Success_Consultation_prev"           "Other_License_prev"
 [9] "Success_Taskforce_prev"              "Success_Other_prev"
[11] "Other_Other_prev"                    "Consultation_prev_viol"
[13] "Taskforce_prev_viol"                 "Other_prev_viol"
```

```
length(tentative_vars)
```

```
[1] 14
```

```
search_criteria = list(
  strategy = "RandomDiscrete", max_runtime_secs = 1800, max_models = 100,
  seed = 1, stopping_rounds = 5, stopping_tolerance = 0.01
)
```

1 glm logistic

```
hyper_params_glm <- list(
  alpha = seq(0, 1, by = .1)
```

```
)

dl_random_grid_glm <- h2o.grid(
  algorithm="glm",
  grid_id = "grid_random_glm",
  training_frame=food_inspect_train,
  validation_frame=food_inspect_validate,
  x=predictors,
  y=response,
  stopping_metric="logloss",
  stopping_tolerance=1e-2,        # stop when logloss does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  hyper_params = hyper_params_glm,
  search_criteria = search_criteria,
  seed = 41204
)
```

```
  |                                                                              |

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!
```

```
Warning in doTryCatch(return(expr), name, parentenv, handler): Reached maximum
number of iterations 50!

Warning in h2o.getGrid(grid_id = grid_id): Adding alpha array to hyperparameter
runs slower with gridsearch. This is due to the fact that the algo has to run
initialization for every alpha value. Setting the alpha array as a model
parameter will skip the initialization and run faster overall.
```

```
grid_glm <- h2o.getGrid("grid_random_glm", sort_by = "logloss", decreasing = FALSE)
```

```
Warning in h2o.getGrid("grid_random_glm", sort_by = "logloss", decreasing =
FALSE): Adding alpha array to hyperparameter runs slower with gridsearch. This
is due to the fact that the algo has to run initialization for every alpha
value. Setting the alpha array as a model parameter will skip the
initialization and run faster overall.
```

```
grid_glm
```

```
H2O Grid Details
================

Grid ID: grid_random_glm
Used hyper parameters:
  -  alpha
Number of models: 11
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss
   alpha                model_ids logloss
1    0.0  grid_random_glm_model_8 0.50241
2    1.0 grid_random_glm_model_11 0.50525
3    0.9  grid_random_glm_model_4 0.50534
4    0.8  grid_random_glm_model_3 0.50537
5    0.7  grid_random_glm_model_9 0.50551
6    0.6  grid_random_glm_model_2 0.50564
7    0.5 grid_random_glm_model_10 0.50588
8    0.4  grid_random_glm_model_1 0.50626
9    0.3  grid_random_glm_model_6 0.50688
10   0.2  grid_random_glm_model_5 0.50803
11   0.1  grid_random_glm_model_7 0.51081
```

```
grid_glm@summary_table[1, ]
```

```
Hyper-Parameter Search Summary: ordered by increasing logloss
  alpha               model_ids logloss
1   0.0 grid_random_glm_model_8 0.50241
```

```
best_model_glm <- h2o.getModel(grid_glm@model_ids[[1]])  # model with lowest logloss
best_model_glm
```

```
Model Details:
==============

H2OBinomialModel: glm
Model ID:  grid_random_glm_model_8
GLM Model: summary
    family  link              regularization number_of_predictors_total
1 binomial logit Ridge ( lambda = 0.002365 )                      38190
  number_of_active_predictors number_of_iterations         training_frame
1                       32856                  23 Food_Inspections_train

Coefficients: glm coefficients
              names coefficients standardized_coefficients
1         Intercept     9.504354                 -1.282446
2       License...1    -0.000489                 -0.000489
3    License...1000    -0.000474                 -0.000474
4 License...1000049     0.001871                  0.001871
5 License...1000200     0.000866                  0.000866

---
                      names coefficients standardized_coefficients
38186      Canvass_prev_viol     0.002502                  0.049712
38187    Complaint_prev_viol    -0.002159                 -0.034183
38188 Consultation_prev_viol     0.005438                  0.003833
38189      License_prev_viol     0.010842                  0.044189
38190    Taskforce_prev_viol    -0.005304                 -0.003277
38191        Other_prev_viol    -0.003555                 -0.003739

H2OBinomialMetrics: glm
** Reported on training data. **

MSE:  0.1638384
RMSE:  0.4047695
LogLoss:  0.5029829
Mean Per-Class Error:  0.3775972
AUC:  0.6695239
AUCPR:  0.3610234
Gini:  0.3390478
```

```
R^2:  0.06200231
Residual Deviance:  130595.5
AIC:  196309.5

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error           Rate
0      58798 41744 0.415190  =41744/100542
1       9955 19324 0.340005    =9955/29279
Totals 68753 61068 0.398233  =51699/129821

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold         value idx
1                       max f1  0.235716      0.427773 231
2                       max f2  0.128896      0.605917 335
3                 max f0point5  0.311025      0.382172 162
4                 max accuracy  0.531718      0.775267  40
5                max precision  0.824868      0.857143   2
6                   max recall  0.022038      1.000000 397
7              max specificity  0.898726      0.999990   0
8             max absolute_mcc  0.261028      0.209404 206
9   max min_per_class_accuracy  0.245393      0.620650 222
10 max mean_per_class_accuracy  0.240431      0.622805 227
11                      max tns  0.898726 100541.000000   0
12                      max fns  0.898726  29278.000000   0
13                      max fps  0.008325 100542.000000 399
14                      max tps  0.022038  29279.000000 397
15                      max tnr  0.898726      0.999990   0
16                      max fnr  0.898726      0.999966   0
17                      max fpr  0.008325      1.000000 399
18                      max tpr  0.022038      1.000000 397

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
H2OBinomialMetrics: glm
** Reported on validation data. **

MSE:  0.1635114
RMSE:  0.4043654
LogLoss:  0.5024096
Mean Per-Class Error:  0.3771122
AUC:  0.6701563
AUCPR:  0.368834
Gini:  0.3403125
R^2:  0.0644496
Residual Deviance:  43263.5
AIC:  108977.5
```

```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
0      19866 13471 0.404086  =13471/33337
1       3403  6316 0.350139    =3403/9719
Totals 23269 19787 0.391908  =16874/43056

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                    max f1  0.237927     0.428116 228
2                    max f2  0.148085     0.607606 318
3                  max f0point5  0.312670     0.381060 158
4                  max accuracy  0.464210     0.776918  56
5                 max precision  0.681918     0.666667   4
6                    max recall  0.045082     1.000000 394
7               max specificity  0.884905     0.999970   0
8              max absolute_mcc  0.264169     0.209644 202
9    max min_per_class_accuracy  0.244645     0.620640 221
10 max mean_per_class_accuracy  0.240366     0.623093 225
11                   max tns  0.884905 33336.000000   0
12                   max fns  0.884905  9719.000000   0
13                   max fps  0.007767 33337.000000 399
14                   max tps  0.045082  9719.000000 394
15                   max tnr  0.884905     0.999970   0
16                   max fnr  0.884905     1.000000   0
17                   max fpr  0.007767     1.000000 399
18                   max tpr  0.045082     1.000000 394

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
h2o.performance(best_model_glm, newdata = food_inspect_test)
```

```
H2OBinomialMetrics: glm

MSE:  0.1624349
RMSE:  0.4030321
LogLoss:  0.5000257
Mean Per-Class Error:  0.3792041
AUC:  0.6678676
AUCPR:  0.3558686
Gini:  0.3357353
R^2:  0.05999602
Residual Deviance:  43179.22
AIC:  108893.2
```

```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
0      20092 13493 0.401757  =13493/33585
1       3421  6171 0.356651    =3421/9592
Totals 23513 19664 0.391736  =16914/43177

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                       max f1  0.238436     0.421862 231
2                       max f2  0.147627     0.602190 320
3                 max f0point5  0.312860     0.377140 162
4                 max accuracy  0.501591     0.778864  45
5                max precision  0.654826     0.600000  10
6                   max recall  0.009720     1.000000 399
7              max specificity  0.815114     0.999970   0
8             max absolute_mcc  0.285516     0.203157 184
9   max min_per_class_accuracy  0.243445     0.617031 226
10 max mean_per_class_accuracy  0.238436     0.620796 231
11                      max tns  0.815114 33584.000000   0
12                      max fns  0.815114  9591.000000   0
13                      max fps  0.009720 33585.000000 399
14                      max tps  0.009720  9592.000000 399
15                      max tnr  0.815114     0.999970   0
16                      max fnr  0.815114     0.999896   0
17                      max fpr  0.009720     1.000000 399
18                      max tpr  0.009720     1.000000 399

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
(confusionMatrix_glm = h2o.confusionMatrix(best_model_glm, food_inspect_test))
```

```
Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.238436417286681:
           0     1    Error          Rate
0      20092 13493 0.401757  =13493/33585
1       3421  6171 0.356651    =3421/9592
Totals 23513 19664 0.391736  =16914/43177
```

```
(totalError_glm = confusionMatrix_glm$Error[length(confusionMatrix_glm$Error)])
```

```
[1] 0.3917363
```

2 random forest

```
hyper_params_rf <- list(
  # Number of features to split on
  mtries       = seq(2, 10, by = 2),

  # Minimal node size
  min_rows  = c(1, 25, 50, 100, 150, 200),

  # Fraction of observation to sample - bootstrapping
  sample_rate = c(.55, .632, .70, .80)
)

random_grid_rf = h2o.grid(
  algorithm="randomForest",
  grid_id = "grid_random_rf",
  training_frame=food_inspect_train,
  validation_frame=food_inspect_validate,
  x=predictors,
  y=response,
  ntrees = 3000,
  stopping_metric = "logloss",
  stopping_tolerance=1e-2,
  stopping_rounds=2,
  hyper_params = hyper_params_rf,
  search_criteria = search_criteria,
  seed = 41204
)
```

```
  |                                                                              |
```

```
grid_rf = h2o.getGrid("grid_random_rf", sort_by = "logloss", decreasing = FALSE)
grid_rf
```

```
H2O Grid Details
================

Grid ID: grid_random_rf
Used hyper parameters:
  -  min_rows
  -  mtries
  -  sample_rate
Number of models: 22
Number of failed models: 0
```

```
Hyper-Parameter Search Summary: ordered by increasing logloss
  min_rows   mtries sample_rate               model_ids logloss
1  1.00000 10.00000     0.63200  grid_random_rf_model_7 0.48753
2 25.00000 10.00000     0.70000 grid_random_rf_model_11 0.49070
3 25.00000 10.00000     0.63200 grid_random_rf_model_20 0.49215
4 25.00000  8.00000     0.80000  grid_random_rf_model_5 0.49241
5 50.00000 10.00000     0.70000  grid_random_rf_model_9 0.49409

---
    min_rows  mtries sample_rate               model_ids logloss
17 100.00000 4.00000     0.55000 grid_random_rf_model_18 0.50696
18   1.00000 2.00000     0.55000 grid_random_rf_model_16 0.50834
19   1.00000 2.00000     0.70000 grid_random_rf_model_19 0.51066
20  50.00000 2.00000     0.63200 grid_random_rf_model_14 0.51232
21  50.00000 2.00000     0.55000  grid_random_rf_model_8 0.51290
22 100.00000 2.00000     0.55000 grid_random_rf_model_13 0.51449
```

```
grid_rf@summary_table[1, ]
```

```
Hyper-Parameter Search Summary: ordered by increasing logloss
  min_rows   mtries sample_rate               model_ids logloss
1  1.00000 10.00000     0.63200 grid_random_rf_model_7 0.48753
```

```
best_model_rf <- h2o.getModel(grid_rf@model_ids[[1]])  # model with lowest logloss
best_model_rf
```

```
Model Details:
==============

H2OBinomialModel: drf
Model ID:  grid_random_rf_model_7
Model Summary:
  number_of_trees number_of_internal_trees model_size_in_bytes min_depth
1              52                       52             6081398        20
  max_depth mean_depth min_leaves max_leaves mean_leaves
1        20   20.00000       7580       9848  8731.50000


H2OBinomialMetrics: drf
** Reported on training data. **
** Metrics reported on Out-Of-Bag training samples **

MSE:  0.160479
RMSE:  0.4005983
```

```
LogLoss:  0.4960614
Mean Per-Class Error:  0.3660408
AUC:  0.6852975
AUCPR:  0.4025905
Gini:  0.370595
R^2:  0.08123542

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error           Rate
0      61242 39300 0.390881  =39300/100542
1       9990 19289 0.341200    =9990/29279
Totals 71232 58589 0.379677  =49290/129821

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold         value idx
1                       max f1  0.219504      0.439045 254
2                       max f2  0.107035      0.607060 338
3                 max f0point5  0.327465      0.405401 180
4                 max accuracy  0.564954      0.781414  78
5                max precision  0.987531      1.000000   0
6                   max recall  0.002088      1.000000 399
7              max specificity  0.987531      1.000000   0
8             max absolute_mcc  0.291563      0.230936 203
9   max min_per_class_accuracy  0.227032      0.631796 248
10 max mean_per_class_accuracy  0.222088      0.634049 252
11                      max tns  0.987531 100542.000000   0
12                      max fns  0.987531  29274.000000   0
13                      max fps  0.002088 100542.000000 399
14                      max tps  0.002088  29279.000000 399
15                      max tnr  0.987531      1.000000   0
16                      max fnr  0.987531      0.999829   0
17                      max fpr  0.002088      1.000000 399
18                      max tpr  0.002088      1.000000 399

Gains/Lift Table: Extract with ‘h2o.gainsLift(<model>, <data>)‘ or ‘h2o.gainsLift(<model>, valid=<T/F>,
H2OBinomialMetrics: drf
** Reported on validation data. **

MSE:  0.1576429
RMSE:  0.3970427
LogLoss:  0.4875346
Mean Per-Class Error:  0.3527347
AUC:  0.7000912
AUCPR:  0.422512
Gini:  0.4001825
```

```
R^2:  0.09802664

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
0      20699 12638 0.379098  =12638/33337
1       3172  6547 0.326371    =3172/9719
Totals 23871 19185 0.367196  =15810/43056

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                       max f1  0.223238     0.453017 247
2                       max f2  0.123235     0.613282 326
3                 max f0point5  0.337100     0.418418 167
4                 max accuracy  0.511131     0.783700  85
5                max precision  0.968387     1.000000   0
6                   max recall  0.017008     1.000000 397
7               max specificity  0.968387     1.000000   0
8             max absolute_mcc  0.266273     0.253133 215
9   max min_per_class_accuracy  0.231852     0.644305 240
10 max mean_per_class_accuracy  0.223238     0.647265 247
11                      max tns  0.968387 33337.000000   0
12                      max fns  0.968387  9718.000000   0
13                      max fps  0.005506 33337.000000 399
14                      max tps  0.017008  9719.000000 397
15                      max tnr  0.968387     1.000000   0
16                      max fnr  0.968387     0.999897   0
17                      max fpr  0.005506     1.000000 399
18                      max tpr  0.017008     1.000000 397

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
h2o.performance(best_model_rf, newdata = food_inspect_test)
```

```
H2OBinomialMetrics: drf

MSE:  0.1566001
RMSE:  0.3957273
LogLoss:  0.4852721
Mean Per-Class Error:  0.3546516
AUC:  0.6972107
AUCPR:  0.4131463
Gini:  0.3944214
R^2:  0.09376149

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
```

```
           0     1    Error          Rate
0      21251 12334 0.367247  =12334/33585
1       3281  6311 0.342056    =3281/9592
Totals 24532 18645 0.361651  =15615/43177
```

```
Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                       max f1  0.226282     0.447002 244
2                       max f2  0.146049     0.608469 310
3                 max f0point5  0.336692     0.418304 167
4                 max accuracy  0.542636     0.786113  80
5                max precision  0.852721     0.875000  11
6                   max recall  0.016242     1.000000 397
7               max specificity  0.974423     0.999970   0
8              max absolute_mcc  0.242650     0.247701 232
9    max min_per_class_accuracy  0.230196     0.644782 241
10 max mean_per_class_accuracy  0.226282     0.645348 244
11                      max tns  0.974423 33584.000000   0
12                      max fns  0.974423  9592.000000   0
13                      max fps  0.006187 33585.000000 399
14                      max tps  0.016242  9592.000000 397
15                      max tnr  0.974423     0.999970   0
16                      max fnr  0.974423     1.000000   0
17                      max fpr  0.006187     1.000000 399
18                      max tpr  0.016242     1.000000 397
```

```
Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
(confusionMatrix_rf = h2o.confusionMatrix(best_model_rf, food_inspect_test))
```

```
Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.226281823456968:
           0     1    Error          Rate
0      21251 12334 0.367247  =12334/33585
1       3281  6311 0.342056    =3281/9592
Totals 24532 18645 0.361651  =15615/43177
```

```
(totalError_rf = confusionMatrix_rf$Error[length(confusionMatrix_rf$Error)])
```

```
[1] 0.3616509
```

3 xgboost

```
hyper_params_xgb <- list(
  max_depth = c(1,3,5,7,9, 11, 13),
  min_rows = c(10, 30, 50),
  eta = c(.01, .1, .3),
  subsample = c(.5, .65, .8)
)

dl_random_grid_xgb <- h2o.grid(
  algorithm="xgboost",
  grid_id = "grid_random_xgb",
  training_frame=food_inspect_train,
  validation_frame=food_inspect_validate,
  x=predictors,
  y=response,
  ntrees = 3000,
  stopping_metric="logloss",
  stopping_tolerance=1e-2,        # stop when logloss does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  hyper_params = hyper_params_xgb,
  search_criteria = search_criteria,
  seed = 41204
)
```

```
  |                                                                              |
```

```
grid_xgb <- h2o.getGrid("grid_random_xgb", sort_by = "logloss", decreasing = FALSE)
grid_xgb
```

```
H2O Grid Details
================

Grid ID: grid_random_xgb
Used hyper parameters:
  -  eta
  -  max_depth
  -  min_rows
  -  subsample
Number of models: 16
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss
       eta max_depth min_rows subsample                model_ids logloss
1  0.10000  13.00000 30.00000   0.65000  grid_random_xgb_model_3 0.48482
2  0.30000  13.00000 30.00000   0.80000  grid_random_xgb_model_6 0.48489
```

```
3  0.30000  13.00000 50.00000   0.80000 grid_random_xgb_model_15 0.48494
4  0.10000  11.00000 30.00000   0.50000 grid_random_xgb_model_13 0.48613
5  0.10000  11.00000 10.00000   0.65000  grid_random_xgb_model_4 0.48654
6  0.10000  13.00000 30.00000   0.50000 grid_random_xgb_model_11 0.48662
7  0.30000  13.00000 50.00000   0.65000  grid_random_xgb_model_2 0.48713
8  0.30000   5.00000 30.00000   0.80000  grid_random_xgb_model_7 0.48862
9  0.30000   3.00000 10.00000   0.65000 grid_random_xgb_model_14 0.49259
10 0.01000   7.00000 10.00000   0.50000 grid_random_xgb_model_12 0.50294
11 0.01000  13.00000 10.00000   0.80000 grid_random_xgb_model_10 0.50476
12 0.01000   9.00000 30.00000   0.50000  grid_random_xgb_model_1 0.50809
13 0.01000   7.00000 50.00000   0.50000  grid_random_xgb_model_9 0.50826
14 0.30000   1.00000 10.00000   0.80000  grid_random_xgb_model_5 0.51698
15 0.01000   5.00000 30.00000   0.65000  grid_random_xgb_model_8 0.52085
16 0.01000  11.00000 30.00000   0.50000 grid_random_xgb_model_16 0.56971
```

```
grid_xgb@summary_table[1, ]
```

```
Hyper-Parameter Search Summary: ordered by increasing logloss
      eta max_depth min_rows subsample               model_ids logloss
1 0.10000  13.00000 30.00000   0.65000 grid_random_xgb_model_3 0.48482
```

```
best_model_xgb <- h2o.getModel(grid_xgb@model_ids[[1]])  # model with lowest logloss
best_model_xgb
```

```
Model Details:
==============

H2OBinomialModel: xgboost
Model ID:  grid_random_xgb_model_3
Model Summary:
  number_of_trees
1              28


H2OBinomialMetrics: xgboost
** Reported on training data. **

MSE:  0.14603
RMSE:  0.3821387
LogLoss:  0.4562847
Mean Per-Class Error:  0.2959699
AUC:  0.7805944
AUCPR:  0.5269409
Gini:  0.5611887
R^2:  0.163958
```

```
Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error           Rate
0      75521 25021 0.248861  =25021/100542
1      10045 19234 0.343079   =10045/29279
Totals 85566 44255 0.270110  =35066/129821

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold         value idx
1                       max f1  0.271216      0.523132 218
2                       max f2  0.197086      0.659102 281
3                 max f0point5  0.352206      0.523283 156
4                 max accuracy  0.397023      0.801118 127
5                max precision  0.846191      1.000000   0
6                   max recall  0.061577      1.000000 395
7              max specificity  0.846191      1.000000   0
8             max absolute_mcc  0.312377      0.364162 186
9   max min_per_class_accuracy  0.255132      0.703905 232
10 max mean_per_class_accuracy  0.253835      0.705647 233
11                      max tns  0.846191 100542.000000   0
12                      max fns  0.846191  29276.000000   0
13                      max fps  0.049133 100542.000000 399
14                      max tps  0.061577  29279.000000 395
15                      max tnr  0.846191      1.000000   0
16                      max fnr  0.846191      0.999898   0
17                      max fpr  0.049133      1.000000 399
18                      max tpr  0.061577      1.000000 395

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
H2OBinomialMetrics: xgboost
** Reported on validation data. **

MSE:  0.15683
RMSE:  0.3960177
LogLoss:  0.4848192
Mean Per-Class Error:  0.3456481
AUC:  0.7107403
AUCPR:  0.4295608
Gini:  0.4214806
R^2:  0.1026779

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
0      22005 11332 0.339923  =11332/33337
1       3415  6304 0.351374    =3415/9719
```

```
Totals 25420 17636 0.342507  =14747/43056

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                        max f1  0.248431     0.460903 233
2                        max f2  0.150063     0.620290 324
3                  max f0point5  0.354122     0.428252 151
4                  max accuracy  0.479437     0.783979  84
5                 max precision  0.827645     1.000000   0
6                    max recall  0.056478     1.000000 398
7               max specificity  0.827645     1.000000   0
8              max absolute_mcc  0.270482     0.263808 215
9    max min_per_class_accuracy  0.246290     0.653358 235
10 max mean_per_class_accuracy  0.248431     0.654352 233
11                      max tns  0.827645 33337.000000   0
12                      max fns  0.827645  9716.000000   0
13                      max fps  0.052701 33337.000000 399
14                      max tps  0.056478  9719.000000 398
15                      max tnr  0.827645     1.000000   0
16                      max fnr  0.827645     0.999691   0
17                      max fpr  0.052701     1.000000 399
18                      max tpr  0.056478     1.000000 398

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
h2o.performance(best_model_xgb, newdata = food_inspect_test)
```

```
H2OBinomialMetrics: xgboost

MSE:  0.1562592
RMSE:  0.3952964
LogLoss:  0.4836031
Mean Per-Class Error:  0.3522141
AUC:  0.7049079
AUCPR:  0.4141288
Gini:  0.4098159
R^2:  0.0957343

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
0      22129 11456 0.341105  =11456/33585
1       3485  6107 0.363324    =3485/9592
Totals 25614 17563 0.346041  =14941/43177

Maximum Metrics: Maximum metrics at their respective thresholds
```

```
                      metric threshold        value idx
1                     max f1  0.247993     0.449788 236
2                     max f2  0.155872     0.615149 317
3                max max f0point5  0.355647     0.415777 150
4                max accuracy  0.505170     0.785858  70
5               max precision  0.816768     1.000000   0
6                  max recall  0.061827     1.000000 395
7              max specificity  0.816768     1.000000   0
8             max absolute_mcc  0.290032     0.251800 201
9   max min_per_class_accuracy  0.245363     0.645642 238
10 max mean_per_class_accuracy  0.234511     0.647884 247
11                    max tns  0.816768 33585.000000   0
12                    max fns  0.816768  9591.000000   0
13                    max fps  0.050233 33585.000000 399
14                    max tps  0.061827  9592.000000 395
15                    max tnr  0.816768     1.000000   0
16                    max fnr  0.816768     0.999896   0
17                    max fpr  0.050233     1.000000 399
18                    max tpr  0.061827     1.000000 395

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
(confusionMatrix_xgb = h2o.confusionMatrix(best_model_xgb, food_inspect_test))
```

```
Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.247992592263807:
           0     1    Error          Rate
0      22129 11456 0.341105  =11456/33585
1       3485  6107 0.363324    =3485/9592
Totals 25614 17563 0.346041  =14941/43177
```

```
(totalError_xgb = confusionMatrix_xgb$Error[length(confusionMatrix_xgb$Error)])
```

```
[1] 0.3460407
```

4 Neural Network

```
hyper_params_nn <- list(
  activation=c("Rectifier","Tanh","RectifierWithDropout","TanhWithDropout"),
  hidden=list(c(20,20),c(50,50),c(30,30,30),c(25,25,25,25),c(64,64,64,64)),
  input_dropout_ratio=c(0,0.05),
  l1=seq(0,1e-4,1e-6),
  l2=seq(0,1e-4,1e-6),
```

```
  max_w2=c(5,10,15)
)



dl_random_grid <- h2o.grid(
  algorithm="deeplearning",
  grid_id = "dl_grid_random",
  training_frame=food_inspect_train,
  validation_frame=food_inspect_validate,
  x=predictors,
  y=response,
  epochs=1000,
  stopping_metric="logloss",
  stopping_tolerance=1e-2,        # stop when logloss does not improve by >=1% for 2 scoring events
  stopping_rounds=2,
  score_duty_cycle=0.025,         # don't score more than 2.5% of the wall time
  hyper_params = hyper_params_nn,
  search_criteria = search_criteria,
  seed = 41204
)
```

```
  |                                                                              |
```

```
dl_grid <- h2o.getGrid("dl_grid_random", sort_by = "logloss", decreasing = FALSE)
dl_grid
```

```
H2O Grid Details
================

Grid ID: dl_grid_random
Used hyper parameters:
  -  activation
  -  hidden
  -  input_dropout_ratio
  -  l1
  -  l2
  -  max_w2
Number of models: 1
Number of failed models: 0

Hyper-Parameter Search Summary: ordered by increasing logloss
            activation           hidden input_dropout_ratio      l1      l2
1 RectifierWithDropout [25, 25, 25, 25]             0.00000 0.00004 0.00008
```

```
    max_w2              model_ids logloss
1 10.00000 dl_grid_random_model_1 0.55243
```

```
dl_grid@summary_table[1, ]
```

```
Hyper-Parameter Search Summary: ordered by increasing logloss
            activation           hidden input_dropout_ratio      l1      l2
1 RectifierWithDropout [25, 25, 25, 25]             0.00000 0.00004 0.00008
    max_w2              model_ids logloss
1 10.00000 dl_grid_random_model_1 0.55243
```

```
best_model_nn <- h2o.getModel(dl_grid@model_ids[[1]])  ## model with lowest logloss
best_model_nn
```

```
Model Details:
==============

H2OBinomialModel: deeplearning
Model ID:  dl_grid_random_model_1
Status of Neuron Layers: predicting Results, 2-class classification, bernoulli distribution, CrossEntro
  layer units             type dropout       l1       l2 mean_rate rate_rms
1     1 38204            Input  0.00 %       NA       NA        NA       NA
2     2    25 RectifierDropout 50.00 % 0.000041 0.000077  0.327390 0.437493
3     3    25 RectifierDropout 50.00 % 0.000041 0.000077  0.006981 0.006940
4     4    25 RectifierDropout 50.00 % 0.000041 0.000077  0.020360 0.015627
5     5    25 RectifierDropout 50.00 % 0.000041 0.000077  0.018908 0.025670
6     6     2          Softmax      NA 0.000041 0.000077  0.005020 0.007090
  momentum mean_weight weight_rms mean_bias bias_rms
1       NA          NA         NA        NA       NA
2 0.000000   -0.000932   0.015172 -0.686783 0.395287
3 0.000000   -0.079430   0.312392  0.131087 0.296159
4 0.000000   -0.009698   0.410960  0.599390 0.772611
5 0.000000   -0.158753   0.331830  0.073621 0.444034
6 0.000000    0.069607   0.470081 -0.028806 0.246707


H2OBinomialMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 9910 samples **

MSE:  0.1811545
RMSE:  0.4256225
LogLoss:  0.5501769
Mean Per-Class Error:  0.3539917
AUC:  0.6805491
```

```
AUCPR:  0.3599286
Gini:  0.3610982

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
          0    1    Error        Rate
0      5220 2466 0.320843  =2466/7686
1       861 1363 0.387140   =861/2224
Totals 6081 3829 0.335721  =3327/9910

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold       value idx
1                       max f1  0.334290    0.450355 169
2                       max f2  0.313399    0.599822 290
3                 max f0point5  0.362918    0.400836  55
4                 max accuracy  0.371998    0.776287   5
5                max precision  0.389298    1.000000   0
6                   max recall  0.265425    1.000000 397
7              max specificity  0.389298    1.000000   0
8             max absolute_mcc  0.334290    0.250204 169
9   max min_per_class_accuracy  0.330481    0.644335 186
10 max mean_per_class_accuracy  0.334290    0.646008 169
11                      max tns  0.389298 7686.000000   0
12                      max fns  0.389298 2223.000000   0
13                      max fps  0.250354 7686.000000 399
14                      max tps  0.265425 2224.000000 397
15                      max tnr  0.389298    1.000000   0
16                      max fnr  0.389298    0.999550   0
17                      max fpr  0.250354    1.000000 399
18                      max tpr  0.265425    1.000000 397

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
H2OBinomialMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on full validation frame **

MSE:  0.1821906
RMSE:  0.426838
LogLoss:  0.552429
Mean Per-Class Error:  0.376995
AUC:  0.6610358
AUCPR:  0.3450631
Gini:  0.3220715

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
```

```
0      20944 12393 0.371749  =12393/33337
1       3715  6004 0.382241    =3715/9719
Totals 24659 18397 0.374117  =16108/43056

Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                       max f1  0.329841     0.427088 186
2                       max f2  0.309219     0.600848 307
3                 max f0point5  0.354937     0.385563  84
4                 max accuracy  0.373324     0.774735   6
5                max precision  0.388483     1.000000   0
6                   max recall  0.262341     1.000000 396
7              max specificity  0.388483     1.000000   0
8             max absolute_mcc  0.337809     0.213972 152
9   max min_per_class_accuracy  0.329411     0.622183 188
10 max mean_per_class_accuracy  0.330644     0.623117 182
11                      max tns  0.388483 33337.000000   0
12                      max fns  0.388483  9717.000000   0
13                      max fps  0.249313 33337.000000 399
14                      max tps  0.262341  9719.000000 396
15                      max tnr  0.388483     1.000000   0
16                      max fnr  0.388483     0.999794   0
17                      max fpr  0.249313     1.000000 399
18                      max tpr  0.262341     1.000000 396

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
h2o.performance(best_model_nn, newdata = food_inspect_test)
```

```
H2OBinomialMetrics: deeplearning

MSE:  0.1824333
RMSE:  0.4271222
LogLoss:  0.5532143
Mean Per-Class Error:  0.3847291
AUC:  0.6438968
AUCPR:  0.3332859
Gini:  0.2877936

Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
           0     1    Error          Rate
0      21321 12264 0.365163  =12264/33585
1       3878  5714 0.404295    =3878/9592
Totals 25199 17978 0.373856  =16142/43177
```

```
Maximum Metrics: Maximum metrics at their respective thresholds
                        metric threshold        value idx
1                       max f1  0.340098     0.414509 168
2                       max f2  0.272003     0.588181 392
3                 max f0point5  0.364582     0.377638  54
4                 max accuracy  0.371440     0.778308  10
5                max precision  0.391766     1.000000   0
6                   max recall  0.251978     1.000000 398
7               max specificity  0.391766     1.000000   0
8              max absolute_mcc  0.351914     0.200946 112
9    max min_per_class_accuracy  0.337958     0.612385 179
10 max mean_per_class_accuracy  0.341181     0.615582 163
11                      max tns  0.391766 33585.000000   0
12                      max fns  0.391766  9591.000000   0
13                      max fps  0.249572 33585.000000 399
14                      max tps  0.251978  9592.000000 398
15                      max tnr  0.391766     1.000000   0
16                      max fnr  0.391766     0.999896   0
17                      max fpr  0.249572     1.000000 399
18                      max tpr  0.251978     1.000000 398

Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLift(<model>, valid=<T/F>,
```

```
(confusionMatrix_nn = h2o.confusionMatrix(best_model_nn, food_inspect_test))
```

```
Confusion Matrix (vertical: actual; across: predicted)  for max f1 @ threshold = 0.340097576898958:
           0     1    Error          Rate
0      21321 12264 0.365163  =12264/33585
1       3878  5714 0.404295    =3878/9592
Totals 25199 17978 0.373856  =16142/43177
```

```
(totalError_nn = confusionMatrix_nn$Error[length(confusionMatrix_nn$Error)])
```

```
[1] 0.3738565
```

```
## import relevant libraries
import pandas as pd
import numpy as np

## read data into a dataframe with 'Inspection Date' Column's datatype

## converted from string to date-type
data = pd.read_csv('cleaned_data_AB_CG.csv',
                   parse_dates=['Inspection.Date'],encoding='latin-1')

data=data.rename(columns={"Inspection.ID": "Inspection ID",
                      "DBA.Name": "DBA Name",
                      "AKA.Name": "AKA Name",
                      "License..":"License #",
                      "Facility.Type": "Facility Type",
                      "Facility.Type_new":"Facility Type_new",
                      "Inspection.Type": "Inspection Type",
                      "Inspection.Date": "Inspection Date"})
data.columns
```

```
Index(['Unnamed: 0', 'Inspection ID', 'DBA Name', 'AKA Name', 'License
#',
       'Facility Type', 'Facility Type_new', 'Risk', 'Address',
'City',
       'State', 'Zipcode', 'Inspection Date', 'Inspection Type',
'Results',
       'Violations', 'Latitude', 'Longitude', 'Location',
'month.year',
       'previous_month_year', 'Zillow.Price.index', 'Zillow.Date',
'BATTERY',
       'CRIMINAL.DAMAGE', 'MOTOR.VEHICLE.THEFT', 'ROBBERY',
'SEX.OFFENSE',
       'ASSAULT', 'BURGLARY', 'CRIM.SEXUAL.ASSAULT',
'CRIMINAL.TRESPASS',
       'DECEPTIVE.PRACTICE', 'NARCOTICS', 'OTHER.OFFENSE',
       'PUBLIC.PEACE.VIOLATION', 'THEFT', 'ARSON', 'HOMICIDE',
       'INTERFERENCE.WITH.PUBLIC.OFFICER', 'INTIMIDATION',
       'LIQUOR.LAW.VIOLATION', 'OFFENSE.INVOLVING.CHILDREN',
'PROSTITUTION',
       'STALKING', 'WEAPONS.VIOLATION', 'KIDNAPPING', 'GAMBLING',
'OBSCENITY',
       'OTHER.NARCOTIC.VIOLATION', 'CRIMINAL.SEXUAL.ASSAULT',
       'PUBLIC.INDECENCY', 'NON.CRIMINAL', 'HUMAN.TRAFFICKING',
       'NON.CRIMINAL..SUBJECT.SPECIFIED.', 'NON...CRIMINAL',
       'CONCEALED.CARRY.LICENSE.VIOLATION', 'RITUALISM', 'Zip',
       'X2020.Population', 'Current.Business.Licences',
       'Current.Business.Licences.per.Capita', 'X2013.Grocery.Stores',
       'X2013.Grocery.Stores.per.Capita', 'Library.Holds',
       'Library.Holds.per.Capita', 'Shotspotter.Alerts.per.Capita',
       'Shotspotter.Alerts', 'ShotSpotter.Shots',
```

```
       'ShotSpotter.Shots.per.Capita'],
      dtype='object')
```

#DATA EXPLORATION + CLEANING + WRANGLING

```
print("COLUMN DATATYPES::")
print()
print(data.dtypes)
print()
print("duplicated rows", len(data[data.duplicated("Inspection ID")]))
print("unique dba names", len(data['DBA Name'].unique()))
print("unque aka names",len(data['AKA Name'].unique()))
print("unique addresses", len(data['Address'].unique()))
print("no. of unique establishments License # ",
      len(data['License #'].unique()))
print("Total no. of rows", len(data))

## create a temporary copy of dataframe 'x' to work with and trasnform
x = data.copy()
print()
print("MISSING VALUES::")
print(x.isna().sum())
print()
# x
```

```
COLUMN DATATYPES::

Unnamed: 0                          int64
Inspection ID                       int64
DBA Name                           object
AKA Name                           object
License #                         float64
                                   ...
Library.Holds.per.Capita          float64
Shotspotter.Alerts.per.Capita     float64
Shotspotter.Alerts                float64
ShotSpotter.Shots                 float64
ShotSpotter.Shots.per.Capita      float64
Length: 70, dtype: object

duplicated rows 0
unique dba names 31076
unque aka names 29576
unique addresses 19295
no. of unique establishments License #  42793
Total no. of rows 250203

MISSING VALUES::
Unnamed: 0                            0
Inspection ID                         0
```

```
DBA Name                              0
AKA Name                           2491
License #                             8
                                    ...
Library.Holds.per.Capita             79
Shotspotter.Alerts.per.Capita        79
Shotspotter.Alerts                   79
ShotSpotter.Shots                    79
ShotSpotter.Shots.per.Capita         79
Length: 70, dtype: int64
```

data

```
        Unnamed: 0  Inspection ID                         DBA Name  \
0                1        2568436  SCHWAB REHAB HOSPITAL&CARE NET
1                2        2567065                      PARK MANOR
2                3        2564901          CHEEZE AND THANK YOU
3                4        2561538              PACINO'S RC, LLC
4                5        2555843              MC DONALD'S 6039
...            ...            ...                             ...
250198      250199          67769               MCDONALDS #5471
250199      250200          67757  DUNKIN DONUTS/BASKIN-ROBBINS
250200      250201         154220                DUNKIN DONUTS
250201      250202          67764                   DONA TORTA
250202      250203         118304    THREE HAPPINESS RESTAURANT

                              AKA Name  License # Facility Type  \
0       SCHWAB REHAB HOSPITAL&CARE NET  2205586.0      Hospital
1                 PARK MANOR ELEMENTARY    24841.0        School
2                  CHEEZE AND THANK YOU  2872888.0           NaN
3                              PACINOS  2683667.0    Restaurant
4                          MC DONALD'S  2341076.0    Restaurant
...                                ...        ...           ...
250198                      MCDONALD'S    54687.0    Restaurant
250199    DUNKIN DONUTS/BASKIN-ROBBINS  1380279.0    Restaurant
250200                   DUNKIN DONUTS    31590.0    Restaurant
250201                      DONA TORTA  1354422.0    Restaurant
250202      THREE HAPPINESS RESTAURANT    39807.0    Restaurant

          Facility Type_new             Risk
Address  \
0       healthcare_provider    Risk 1 (High)     1401 S CALIFORNIA AVE

1                   School    Risk 1 (High)  7037 S Rhodes (532E) AVE

2                      NaN              NaN          2046 E GRAND AVE

3                Restaurant    Risk 1 (High)          1010 S DELANO CT
```

```
4                Restaurant  Risk 2 (Medium)     5656 W IRVING PARK RD

...                     ...              ...                       ..
.
250198           Restaurant  Risk 2 (Medium)          4338 W NORTH AVE

250199           Restaurant  Risk 2 (Medium)         100 W RANDOLPH ST

250200           Restaurant  Risk 2 (Medium)      1927 W FULLERTON AVE

250201           Restaurant    Risk 1 (High)        3057 N ASHLAND AVE

250202           Restaurant    Risk 1 (High)           209 W CERMAK RD


           City  ... Current.Business.Licences  \
0       CHICAGO  ...                    1656.0
1       CHICAGO  ...                     475.0
2       CHICAGO  ...                       NaN
3       CHICAGO  ...                     840.0
4       CHICAGO  ...                     997.0
...         ...  ...                       ...
250198  CHICAGO  ...                    1502.0
250199  CHICAGO  ...                    1022.0
250200  CHICAGO  ...                    1871.0
250201  CHICAGO  ...                    1356.0
250202  CHICAGO  ...                    1176.0

        Current.Business.Licences.per.Capita X2013.Grocery.Stores  \
0                                   0.020697                 18.0
1                                   0.010189                 11.0
2                                        NaN                  NaN
3                                   0.027141                  2.0
4                                   0.013187                 13.0
...                                      ...                  ...
250198                              0.016969                 32.0
250199                              0.070420                  3.0
250200                              0.025846                 11.0
250201                              0.019237                  5.0
250202                              0.022376                  8.0

       X2013.Grocery.Stores.per.Capita Library.Holds
Library.Holds.per.Capita  \
0                             0.000225          2.0
0.000025
1                             0.000236          1.0
0.000021
2                                  NaN          NaN
NaN
```

```
3                              0.000065           1.0
0.000032
4                              0.000172           3.0
0.000040
...                                 ...           ...
...
250198                         0.000362           2.0
0.000023
250199                         0.000207           0.0
0.000000
250200                         0.000152           1.0
0.000014
250201                         0.000071           2.0
0.000028
250202                         0.000152           2.0
0.000038

        Shotspotter.Alerts.per.Capita  Shotspotter.Alerts
ShotSpotter.Shots  \
0                                 0.0            0.000000
0.0
1                                 7.0            0.000150
28.0
2                                 NaN                 NaN
NaN
3                                 0.0            0.000000
0.0
4                                 0.0            0.000000
0.0
...                               ...                 ...
...
250198                            3.0            0.000034
6.0
250199                            0.0            0.000000
0.0
250200                            0.0            0.000000
0.0
250201                            0.0            0.000000
0.0
250202                            0.0            0.000000
0.0

        ShotSpotter.Shots.per.Capita
0                           0.000000
1                           0.000601
2                                NaN
3                           0.000000
4                           0.000000
...                              ...
250198                      0.000068
```

```
250199                        0.000000
250200                        0.000000
250201                        0.000000
250202                        0.000000

[250203 rows x 70 columns]
```

Inspection ID / DBA Name / AKA Name

```
## Missing values of Inspection ID
print("# of missing IDs", len(x[x['Inspection ID'].isnull()==True]))

## DBA Name, AKA Name
print("number of facilities with different DBA and AKA names",
      len(x.loc[(x["DBA Name"] != x["AKA Name"])]["DBA
Name"].unique()))

print("# of missing DBA Names", len(x[x['DBA Name'].isnull()==True]))
print("# of missing AKA Names", len(x[x['AKA Name'].isnull()==True]))
```

```
# of missing IDs 0
number of facilities with different DBA and AKA names 8335
# of missing DBA Names 0
# of missing AKA Names 2491
```

Handle Missing AKA Names (we have no missing Inspection ID or DBA Names) by imputing missing AKA names with DBA Name + " "

```
x['AKA Name']=np.where(x['AKA Name'].isnull()==False,x['AKA
Name'],x['DBA Name']+" ")
```

New Column 'dbaname&address' by combining DBA Name and Address (helpful for future needs for facility identification in absence of License #)

```
## create new column for facility identifier (we need this as in some
cases
## license # is not sufficient as it is missing or equals 0.0) Also,
DBA Name
## is never missing and we have reason to believe that DBA Name +
Address is a
## good facility identifier in absence of license number
x['dbaname&address'] = x['DBA Name'] + x['Address']
## visualize this new column
x['dbaname&address'][0:10]
```

```
0    SCHWAB REHAB HOSPITAL&CARE NET1401 S CALIFORNI...
1                  PARK MANOR7037 S Rhodes (532E) AVE
2                CHEEZE AND THANK YOU2046 E GRAND AVE
3                    PACINO'S RC, LLC1010 S DELANO CT
4               MC DONALD'S 60395656 W IRVING PARK RD
5                          KYO MATCHA2167 S CHINA PL
6            STARBUCKS COFFEE #2689175 W JACKSON BLVD
7             TONY'S ITALIAN DELI6708 N NORTHWEST HWY
8           TRUMP INTERNATIONAL TOWER401 N WABASH AVE
9                 ANDYS FRUIT RANCH4733 N KEDZIE AVE
Name: dbaname&address, dtype: object
```

New Column 'ascii' added : Basically float value corresponding to each dbaname&address value : These will be imputed for License #s that are missing or equal 0.

```
## create a new column that coverts dbaname&address to a float using
ascii value
x['ascii'] = list(map(lambda a: (''.join(str(ord(c)) for c in a)),
                      x['dbaname&address']))
x['ascii'] = x['ascii'].astype(float)

## visualize this new column
x['ascii'][0:10]
```

```
0    8.367729e+103
1     8.065828e+74
2     6.772697e+73
3     8.065677e+65
4     7.767327e+75
5     7.589793e+51
6     8.384658e+81
7     8.479789e+79
8     8.482858e+83
9     6.578689e+69
Name: ascii, dtype: float64
```

License '#' : Missing values and 0 values handled using ascii column

```
## License #

## some license numbers are missing or labelled '0.0' (0.0 makes no
sense)
## for each license # that is 0.0 or missing, we impute it with a
string
## 'DBA Name + Address' and also convert the full column to string
datatype
x['License #']=np.where(x['License #']==0.0, x['ascii'], x['License
```

```
#'])
x['License #']=np.where(x['License #'].isnull()==True,
                        x['ascii'], x['License #'])

print("Checking no. of missing License #s after imputation",
      x['License #'].isna().sum())
```

```
Checking no. of missing License #s after imputation 0
```

Facility Type (Facility Type_new)

```
x['Facility Type_Old'] = x['Facility Type']
x['Facility Type'] = x['Facility Type_new']
x=x.drop('Facility Type_new',axis=1)
## Facility Type : 22 categories mentioned in readme doc
print("# Missing Values in Facility Type",x['Facility
Type'].isna().sum())
print()
print("# Unique Facility Type Categories",len(x['Facility
Type'].unique()))
print()
x['Facility Type'].unique()
```

```
# Missing Values in Facility Type 5043

# Unique Facility Type Categories 47


array(['healthcare_provider', 'School', nan, 'Restaurant', 'Grocery',
       'bakery_deli', 'outdoor_food_service', 'food_retail',
       'comm_kitchen_catering', 'juice', 'Bar', 'Events',
       'Childrens_Services', 'large_venue', 'nonfood_retail',
       'liquor_store', 'Convenience', 'Senior_Care', 'Wholesale',
       'Cafeteria', 'Hotel', 'Gym', 'icecream', 'Cafe', 'Banquet',
       'cooking_school', 'Other', 'Theater', 'Industrial', 'Pantry',
       'Government', 'Senior Care', 'Candy', 'alt_pharmacy',
'Religious',
       'Donut', 'Commissary', 'Market', 'drug', 'Agriculture',
       'Cooking_School', 'Butcher', 'gallery_museum', 'Hooka',
'Vending',
       'Café', 'Recreation'], dtype=object)
```

Facility Type : Handle missing values (set to nan) : 47 categories

```
## Dealing with missing values of Facility Type : add "nan"
x['Facility Type']=np.where(x['Facility Type'].isnull()==True,
                            'Other', x['Facility Type'])
```

Facility Type: Spelling errors are handled and we have 46 types of facilities

Risk : Replace missing values and 'All' with 'nan'

```
## Risk : set missing and 'All' values to 'nan'
x['Risk'] = np.where(x['Risk'].isnull()==True,'nan',x['Risk'])
x['Risk'] = np.where(x['Risk']=='All','nan',x['Risk'])
x['Risk'].unique()
```

```
array(['Risk 1 (High)', 'nan', 'Risk 2 (Medium)', 'Risk 3 (Low)'],
      dtype=object)
```

City/State/Zip : handle missing values and spelling errors

```
x['City'].unique()
```

```
array(['CHICAGO', 'BERWYN', 'ALGONQUIN', 'ELMHURST', 'WILMETTE',
       'MAYWOOD', 'SCHAUMBURG', 'ELK GROVE VILLAGE', 'BLOOMINGDALE',
       'CICERO', 'EVANSTON', 'JUSTICE', 'TINLEY PARK', 'LAKE ZURICH',
       'OLYMPIA FIELDS', 'BLUE ISLAND', 'WORTH', 'OAK PARK',
'MERRIVILLE',
       'TORRANCE', 'WHITING', 'GLEN ELLYN', 'SUMMIT', 'OAK LAWN',
       'BURNHAM', 'EVERGREEN PARK', 'LOS ANGELES', 'PLAINFIELD',
       'CALUMET CITY', 'MORTON GROVE', 'BRIDGEVIEW', 'GRIFFITH',
'NILES',
       'NEW HOLSTEIN', 'NEW YORK', 'LANSING', 'WADSWORTH', 'ROSEMONT',
       'WHEATON', 'HIGHLAND PARK', 'PALOS PARK', 'LAKE BLUFF',
       'SCHILLER PARK', 'SKOKIE', 'DEERFIELD', 'NORRIDGE',
       'CHARLES A HAYES', 'CHICAGO HEIGHTS', 'WESTMONT', 'LOMBARD',
       'EAST HAZEL CREST', 'COUNTRY CLUB HILLS', 'DES PLAINES',
       'BOLINGBROOK', 'STREAMWOOD', 'ALSIP', 'GLENCOE', 'NAPERVILLE',
       'FRANKFORT', 'BROADVIEW', 'Maywood'], dtype=object)
```

```
x['City'] = np.where(x['City']=='Maywood','MAYWOOD',x['City'])
x['City'].unique()
```

```
array(['CHICAGO', 'BERWYN', 'ALGONQUIN', 'ELMHURST', 'WILMETTE',
       'MAYWOOD', 'SCHAUMBURG', 'ELK GROVE VILLAGE', 'BLOOMINGDALE',
       'CICERO', 'EVANSTON', 'JUSTICE', 'TINLEY PARK', 'LAKE ZURICH',
       'OLYMPIA FIELDS', 'BLUE ISLAND', 'WORTH', 'OAK PARK',
'MERRIVILLE',
       'TORRANCE', 'WHITING', 'GLEN ELLYN', 'SUMMIT', 'OAK LAWN',
       'BURNHAM', 'EVERGREEN PARK', 'LOS ANGELES', 'PLAINFIELD',
       'CALUMET CITY', 'MORTON GROVE', 'BRIDGEVIEW', 'GRIFFITH',
'NILES',
       'NEW HOLSTEIN', 'NEW YORK', 'LANSING', 'WADSWORTH', 'ROSEMONT',
       'WHEATON', 'HIGHLAND PARK', 'PALOS PARK', 'LAKE BLUFF',
       'SCHILLER PARK', 'SKOKIE', 'DEERFIELD', 'NORRIDGE',
```

```
       'CHARLES A HAYES', 'CHICAGO HEIGHTS', 'WESTMONT', 'LOMBARD',
       'EAST HAZEL CREST', 'COUNTRY CLUB HILLS', 'DES PLAINES',
       'BOLINGBROOK', 'STREAMWOOD', 'ALSIP', 'GLENCOE', 'NAPERVILLE',
       'FRANKFORT', 'BROADVIEW'], dtype=object)
```

```
## handle missing zip codes: set them to 0
x['Zip'] = np.where(x['Zip'].isnull()==True,0,x['Zip'])
```

Inspection Type : Handle 1 missing value. Creating meaningful buckets assigned to Christian.

```
x['Inspection Type'] = np.where(x['Inspection Type'].isnull()==True,
                                'OTHER', x['Inspection Type'])

## CANVASS, CANVASS REINSPECTION - related
x['Inspection Type'] = np.where(x['Inspection Type']=='Canvass',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CANVASS SPECIAL
EVENTS',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Canvass Re-
Inspection',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CANVASS
SCHOOL/SPECIAL EVENT',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='CANVAS','CANVASS',
                                x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CANVASS RE
INSPECTION OF CLOSE UP',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CANVASS/SPECIAL
EVENT',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Sample
Collection',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='HACCP
QUESTIONAIRE',
                                'CANVASS',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CANVASS FOR RIB
FEST',
                                'CANVASS',x['Inspection Type'])

## LICENSE, LICENSE REINSPECTION - related
x['Inspection Type'] = np.where(x['Inspection
Type']=='License','LICENSE',
                                x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='License Re-
```

```
Inspection',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='1315 license
reinspection',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='OWNER SUSPENDED
OPERATION/LICENSE',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE
CANCELED BY OWNER',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='license task
1474',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE
REQUEST',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='expansion','LICENSE',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE RENEWAL
FOR DAYCARE',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE/NOT
READY',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE RENEWAL
INSPECTION FOR DAYCARE',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='DAY CARE
LICENSE RENEWAL',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE DAYCARE
1586',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE WRONG
ADDRESS',
                                  'LICENSE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='license','LICENSE',
                                  x['Inspection Type'])

## COMPLAINT, COMPLAINT REINSPECTION - related
x['Inspection Type'] = np.where(x['Inspection
Type']=='Complaint','COMPLAINT',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Short Form
Complaint',
                                  'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Short Form
```

```
Fire-Complaint',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Complaint-
Fire',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='COVID
COMPLAINT',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='fire
complaint',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='FIRE','COMPLAINT',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Complaint Re-
Inspection',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Complaint-Fire
Re-inspection',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Illegal
Operation',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='FIRE/COMPLAIN','COMPLAINT',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Suspected Food
Poisoning',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Suspected Food
Poisoning Re-inspection',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TWO PEOPLE ATE
AND GOT SICK.',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='SFP','COMPLAINT',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='SFP RECENTLY
INSPECTED',
                                    'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='SFP/Complaint','COMPLAINT',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='SFP/COMPLAINT','COMPLAINT',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='SMOKING
COMPLAINT','COMPLAINT',
                                    x['Inspection Type'])
```

```
x['Inspection Type'] = np.where(x['Inspection Type']=='NO ENTRY-SHORT
COMPLAINT)',
                                'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='CLOSE-UP/COMPLAINT REINSPECTION',
                                'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='sfp/complaint','COMPLAINT',
                                x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='finish
complaint inspection from 5-18-10',
                                'COMPLAINT',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='RE-INSPECTION
OF CLOSE-UP',
                                'COMPLAINT',x['Inspection Type'])

## CONSULTATION - related
x['Inspection Type'] = np.where(x['Inspection
Type']=='Consultation','CONSULTATION',
                                x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='License
consultation','CONSULTATION',
                                x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE
CONSULTATION','CONSULTATION',
                                x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Pre-License
Consultation','CONSULTATION',
                                x['Inspection Type'])

## TASK FORCE INSPECTION - related
x['Inspection Type'] = np.where(x['Inspection Type']=='License-Task
Force',
                                'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Package Liquor
1474',
                                'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Task Force
Liquor 1475',
                                'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE
LIQUOR 1470',
                                'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Task Force for
liquor 1474',
                                'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Task Force
Liquor Catering',
                                'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Task force
```

```
liquor inspection 1474',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE
NIGHT',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE TASK
FORCE / NOT -FOR-PROFIT CLU',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LICENSE TASK
FORCE / NOT -FOR-PROFIT CLUB',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TAVERN
1470','TASKFORCE',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='CITF','TASKFORCE',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='task
force(1470) liquor tavern',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LIQUOR
CATERING',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE
PACKAGE LIQUOR',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Special Task
Force',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='POSSIBLE FBI',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='task force',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE NOT
READY',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE
LIQUOR (1481)',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE
LIQUOR 1474',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='SPECIAL TASK
FORCE',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='LIQOUR TASK
FORCE NOT READY',
                                  'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE
PACKAGE GOODS 1474',
```

```
                                    'TASKFORCE',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASK FORCE',
                                    'TASKFORCE',x['Inspection Type'])

## OTHER - related
x['Inspection Type'] = np.where(x['Inspection Type']=='Non-
Inspection','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Not
Ready','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='No
Entry','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='NO
ENTRY','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Recent
Inspection',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='OUT OF
BUSINESS',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Out of
Business',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Tag Removal',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='O.B.','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Recent
inspection',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Special Events
(Festivals)',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CORRECTIVE
ACTION',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='KITCHEN CLOSED
FOR RENOVATION',
                                    'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='No
entry','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='ADDENDUM','OTHER',
                                    x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='no
entry','OTHER',
```

```
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='error
save','OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='KIDS
CAFE','OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Summer
Feeding','OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='CHANGED COURT
DATE',
                                  'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='REINSPECTION','OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='citation re-
issued',
                                  'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='REINSPECTION OF
48 HOUR NOTICE',
                                  'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='TASTE OF
CHICAGO',
                                  'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='out
ofbusiness',
                                  'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection
Type']=='Duplicated','OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=='Business Not
Located',
                                  'OTHER',x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=="Kids
Cafe'",'OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=="nan",'OTHER',
                                  x['Inspection Type'])
x['Inspection Type'] = np.where(x['Inspection Type']=="RECALL
INSPECTION",
                                  'OTHER',x['Inspection Type'])


x['Inspection Type'].unique()
```

```
array(['CANVASS', 'LICENSE', 'OTHER', 'COMPLAINT', 'CONSULTATION',
       'TASKFORCE'], dtype=object)
```

```
data['Results'].unique()
```

```
array(['Pass', 'Not Ready', 'No Entry', 'Out of Business',
       'Pass w/ Conditions', 'Fail', 'Business Not Located'],
dtype=object)
```

Violations

```
data['Violations'][100009]
```

```
{"type":"string"}
```

Location / Latitude / Longitude - Missing values

```
x['Location'] = np.where(x['Location'].isnull()==True,'nan',
x['Location'])
x['Longitude'] = np.where(x['Longitude'].isnull()==True, 0,
x['Longitude'])
x['Latitude'] = np.where(x['Latitude'].isnull()==True,0,
x['Latitude'])
```

TIMESERIES COLUMNS

First, we need to determine how to identify unique facilities in the dataset. The 2 candidates are 'DBA NAME + ADDRESS' or 'License #' or combination (identifier bias?) ::

For simplicity, we go with License # for unqiue facility IDs.

For entries where License # is missing or 0.0, we impute by creating unique ids for each facility by combining facility DBA Name and address. We have already done this in the final form of ascii values.

Side Notes:

1) unique no. of DBA Names can be an undercount when we have multiple facilities with the same name eg. "SUBWAY".

2) unique no. of Addresses is an undercount as we can have multiple facilities with the same address if they are in the same building or the same street for example.

3) Facilities can also change DBA name - Licence number will also change. Businesses may abuse this Loophole - Facilities that go out of business can change DBA and AKA names, get a new licence number and expunge their inspection/violation history. How to deal with that? This is relevant for timeseries as we may have the same restaurants posing as multiple restuarants over time. Not found a solution yet to this. Perhaps, there is no solution.

SORT dataframe in asceding order of (modified) License No. (float dtype) & Inspection Date (datetime dtype) - makes it easier to add timeseries columns

```
## we create a new dataframe df with rows sorted in ascending order of
License #
## and Inspection Date : date order within years was not consistent in
the dataset
df = x.sort_values(['License #','Inspection Date'])
```

Add new column called 'prev_inspection_type'

```
## add a new column called 'prev_inspection_type' to the dataframe

## identifier pointer :: difference != 0 when facility id changes in
## the ordered sequence
df['prev_id']=df['License #'].shift()
df['prev_type_temp']= df['Inspection Type'].shift()
df['difference']= abs(df['prev_id']-df['License #'])

## assigning values to prev_inspection_type column
df['prev_inspection_type'] = np.where(df['difference']==0,
                                      df['prev_type_temp'],'nan')
df = df.drop(['prev_type_temp','difference','prev_id'], axis=1)

## demo visual
df[['License #','Inspection Date','Inspection
Type','prev_inspection_type']][0:10]
```

	License #	Inspection Date	Inspection Type	prev_inspection_type
241405	1.0	2010-06-04	OTHER	nan
241638	2.0	2010-06-15	CANVASS	nan
231139	2.0	2011-02-15	CANVASS	CANVASS
206644	2.0	2012-03-19	CANVASS	CANVASS
210615	2.0	2012-03-21	CANVASS	CANVASS
184540	2.0	2013-07-19	CANVASS	CANVASS
169965	2.0	2014-04-18	CANVASS	CANVASS
156595	2.0	2014-11-20	CANVASS	CANVASS
149419	2.0	2015-04-21	CANVASS	CANVASS
136178	2.0	2015-12-14	COMPLAINT	CANVASS

Add new column called 'prev_result'

```
## add a new column called 'prev_result' to the dataframe

## identifier pointer :: difference != 0 when facility id changes in
## the ordered sequence
df['prev_id']=df['License #'].shift()
df['prev_result_temp']= df['Results'].shift()
```

```
df['difference']= abs(df['prev_id']-df['License #'])

## assigning values to prev_result column
df['prev_result'] = np.where(df['difference']==0,
df['prev_result_temp'],'nan')
df = df.drop(['prev_result_temp','difference','prev_id'], axis=1)

## demo visual
df[['License #','Inspection Date','Results','prev_result']][1500:1510]
```

```
        License # Inspection Date             Results
prev_result
155833     1380.0      2014-11-13  Pass w/ Conditions  Pass w/
Conditions
144107     1380.0      2015-07-14                Fail  Pass w/
Conditions
143499     1380.0      2015-07-21                Pass
Fail
128111     1380.0      2016-03-24  Pass w/ Conditions
Pass
108914     1380.0      2017-02-17                Pass  Pass w/
Conditions
100568     1380.0      2017-07-13                Fail
Pass
99604      1380.0      2017-07-20                Pass
Fail
91352      1380.0      2018-02-02                Fail
Pass
87104      1380.0      2018-02-14                Pass
Fail
85244      1380.0      2018-04-02                Pass
Pass
```

Add new column called 'prev_#_inspections'

```
## add a new column called 'prev_#_inspections' to the dataframe
df['prev_#_inspections'] = df.groupby(['License #']).cumcount()

## visualize this column
df[['License #','prev_#_inspections']][0:10]
```

```
        License #  prev_#_inspections
241405        1.0                   0
241638        2.0                   0
231139        2.0                   1
206644        2.0                   2
210615        2.0                   3
184540        2.0                   4
169965        2.0                   5
156595        2.0                   6
```

```
149419        2.0                    7
136178        2.0                    8
```

Add new column called '#violations'

```
## reading information from violations column
df['Violations']=df['Violations'].astype(str)

df['#violations']=np.where(df['Violations'].isna()==True, 0,
                    list(map(lambda a: len(a.split('|')),
df['Violations'])))

df['#violations']=np.where(df['Violations']=="nan", 0,
df['#violations'])

## visualize this new column
df[['Violations','#violations']][100008:100018]
```

```
        License #
Violations  \
99331   1885160.0                                                    nan

94792   1885160.0                                                    nan

231984  1885162.0  32. FOOD AND NON-FOOD CONTACT SURFACES PROPERL...

209668  1885162.0  32. FOOD AND NON-FOOD CONTACT SURFACES PROPERL...

183595  1885162.0  33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSI...

163069  1885162.0  35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTR...

133824  1885162.0  33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSI...

138067  1885162.0  2. FACILITIES TO MAINTAIN PROPER TEMPERATURE -...

116539  1885162.0  35. WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTR...

95627   1885162.0                                                    nan


        #violations
99331             0
94792             0
231984            3
209668            3
183595            2
163069            2
133824            1
```

```
138067            3
116539            1
95627             0
```

```
print(list(df['Violations'])[100029])
print(list(df['#violations'])[100029])
```

```
41. PREMISES MAINTAINED FREE OF LITTER, UNNECESSARY ARTICLES, CLEANING
EQUIPMENT PROPERLY STORED - Comments: MUST REMOVE CLUTTER IN THE
UNUSED REAR WALK IN COOLER OUTSIDE OF THE EMPLOYEE WASHROOMS AND REAR
STORAGE AREA. MUST STORE ALL ITEMS 6 INCHES OFF THE FLOOR.
1
```

Add new column called 'prev_#_violations'

```
df['d']=df['#violations'].shift()
df['prev_#_violations'] = df.groupby(['License #'])['d'].cumsum()
df['prev_#_violations'] =
np.where(df['prev_#_violations'].isnull()==True,
                                  0.0,df['prev_#_violations'])
df = df.drop(['d'],axis=1)

## visualize this new column
df[['License #', 'Violations','#violations','prev_#_violations']]
[0:10]
```

```
        License #
Violations  \
241405        1.0                                                 nan

241638        2.0  32. FOOD AND NON-FOOD CONTACT SURFACES PROPERL...

231139        2.0  32. FOOD AND NON-FOOD CONTACT SURFACES PROPERL...

206644        2.0  4. SOURCE OF CROSS CONTAMINATION CONTROLLED I....

210615        2.0                                                 nan

184540        2.0  32. FOOD AND NON-FOOD CONTACT SURFACES PROPERL...

169965        2.0  32. FOOD AND NON-FOOD CONTACT SURFACES PROPERL...

156595        2.0  31. CLEAN MULTI-USE UTENSILS AND SINGLE SERVIC...

149419        2.0  31. CLEAN MULTI-USE UTENSILS AND SINGLE SERVIC...

136178        2.0  33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSI...
```

```
        #violations  prev_#_violations
241405            0                0.0
241638            5                0.0
231139            5                5.0
206644            4               10.0
210615            0               14.0
184540            3               14.0
169965            4               17.0
156595            3               21.0
149419            5               24.0
136178            5               29.0
```

Create dummy variables for Inspection Type (8) and Results (4) -> then create historical variables for each category during last 3,6,9,12 years.

```
df_dummy = pd.get_dummies(df, columns=['Inspection Type', 'Results'])
## now we have 8 dummy variables for Inspection Type and 7 dummy
variables for
## Results
list(df_dummy.columns)
```

```
['Unnamed: 0',
 'Inspection ID',
 'DBA Name',
 'AKA Name',
 'License #',
 'Facility Type',
 'Risk',
 'Address',
 'City',
 'State',
 'Zipcode',
 'Inspection Date',
 'Violations',
 'Latitude',
 'Longitude',
 'Location',
 'month.year',
 'previous_month_year',
 'Zillow.Price.index',
 'Zillow.Date',
 'BATTERY',
 'CRIMINAL.DAMAGE',
 'MOTOR.VEHICLE.THEFT',
 'ROBBERY',
 'SEX.OFFENSE',
 'ASSAULT',
 'BURGLARY',
```

```
'CRIM.SEXUAL.ASSAULT',
'CRIMINAL.TRESPASS',
'DECEPTIVE.PRACTICE',
'NARCOTICS',
'OTHER.OFFENSE',
'PUBLIC.PEACE.VIOLATION',
'THEFT',
'ARSON',
'HOMICIDE',
'INTERFERENCE.WITH.PUBLIC.OFFICER',
'INTIMIDATION',
'LIQUOR.LAW.VIOLATION',
'OFFENSE.INVOLVING.CHILDREN',
'PROSTITUTION',
'STALKING',
'WEAPONS.VIOLATION',
'KIDNAPPING',
'GAMBLING',
'OBSCENITY',
'OTHER.NARCOTIC.VIOLATION',
'CRIMINAL.SEXUAL.ASSAULT',
'PUBLIC.INDECENCY',
'NON.CRIMINAL',
'HUMAN.TRAFFICKING',
'NON.CRIMINAL..SUBJECT.SPECIFIED.',
'NON...CRIMINAL',
'CONCEALED.CARRY.LICENSE.VIOLATION',
'RITUALISM',
'Zip',
'X2020.Population',
'Current.Business.Licences',
'Current.Business.Licences.per.Capita',
'X2013.Grocery.Stores',
'X2013.Grocery.Stores.per.Capita',
'Library.Holds',
'Library.Holds.per.Capita',
'Shotspotter.Alerts.per.Capita',
'Shotspotter.Alerts',
'ShotSpotter.Shots',
'ShotSpotter.Shots.per.Capita',
'dbaname&address',
'ascii',
'Facility Type_Old',
'prev_inspection_type',
'prev_result',
'prev_#_inspections',
'#violations',
'prev_#_violations',
'Inspection Type_CANVASS',
'Inspection Type_COMPLAINT',
```

```
 'Inspection Type_CONSULTATION',
 'Inspection Type_LICENSE',
 'Inspection Type_OTHER',
 'Inspection Type_TASKFORCE',
 'Results_Business Not Located',
 'Results_Fail',
 'Results_No Entry',
 'Results_Not Ready',
 'Results_Out of Business',
 'Results_Pass',
 'Results_Pass w/ Conditions']
```

Add 18 new columns for prev_successful/ failed/ Neither cumulative results of 'CANVASS', 'COMPLAINT', 'LICENSE', 'TASKFORCE', and 'OTHER' types of INSPECTIONS for each facility :

'prev_cum_success_res_canvass_itype', 'prev_cum_fail_res_canvass_itype', 'prev_cum_other_res_canvass_itype',

'prev_cum_success_res_complaint_itype', 'prev_cum_fail_res_complaint_itype', 'prev_cum_other_res_complaint_itype',

'prev_cum_success_res_consultation_itype', 'prev_cum_fail_res_consultation_itype', 'prev_cum_other_res_consultation_itype',

'prev_cum_success_res_license_itype', 'prev_cum_fail_res_license_itype', 'prev_cum_other_res_license_itype',

'prev_cum_success_res_taskforce_itype', 'prev_cum_fail_res_taskforce_itype', 'prev_cum_other_res_taskforce_itype',

'prev_cum_success_res_other_itype', 'prev_cum_other_res_canvass_other_itype', 'prev_cum_fail_res_canvass_other_itype'

```
## creating historical variable for previous cumulative number of
## success/fail/other results for different types of inspection types
## for a given facility at every row level

df_dummy['pass']=df_dummy['Results_Pass']+df_dummy['Results_Pass w/
Conditions']
df_dummy['fail']=df_dummy['Results_Fail']
df_dummy['other']=df_dummy['Results_No Entry'] + df_dummy['Results_Out
of Business'] + df_dummy['Results_Not Ready'] +
df_dummy['Results_Business Not Located']

df_dummy['canvass_pass']= np.where(df_dummy['Inspection
Type_CANVASS']==0,0,df_dummy['pass'])
df_dummy['canvass_fail']= np.where(df_dummy['Inspection
```

```
Type_CANVASS']==0,0,df_dummy['fail'])
df_dummy['canvass_other']= np.where(df_dummy['Inspection
Type_CANVASS']==0,0,df_dummy['other'])

df_dummy['complaint_pass']= np.where(df_dummy['Inspection
Type_COMPLAINT']==0,0,df_dummy['pass'])
df_dummy['complaint_fail']= np.where(df_dummy['Inspection
Type_COMPLAINT']==0,0,df_dummy['fail'])
df_dummy['complaint_other']= np.where(df_dummy['Inspection
Type_COMPLAINT']==0,0,df_dummy['other'])

df_dummy['consultation_pass']= np.where(df_dummy['Inspection
Type_CONSULTATION']==0,0,df_dummy['pass'])
df_dummy['consultation_fail']= np.where(df_dummy['Inspection
Type_CONSULTATION']==0,0,df_dummy['fail'])
df_dummy['consultation_other']= np.where(df_dummy['Inspection
Type_CONSULTATION']==0,0,df_dummy['other'])

df_dummy['license_pass']= np.where(df_dummy['Inspection
Type_LICENSE']==0,0,df_dummy['pass'])
df_dummy['license_fail']= np.where(df_dummy['Inspection
Type_LICENSE']==0,0,df_dummy['fail'])
df_dummy['license_other']= np.where(df_dummy['Inspection
Type_LICENSE']==0,0,df_dummy['other'])

df_dummy['taskforce_pass']= np.where(df_dummy['Inspection
Type_TASKFORCE']==0,0,df_dummy['pass'])
df_dummy['taskforce_fail']= np.where(df_dummy['Inspection
Type_TASKFORCE']==0,0,df_dummy['fail'])
df_dummy['taskforce_other']= np.where(df_dummy['Inspection
Type_TASKFORCE']==0,0,df_dummy['other'])

df_dummy['other_itype_pass']= np.where(df_dummy['Inspection
Type_OTHER']==0,0,df_dummy['pass'])
df_dummy['other_itype_fail']= np.where(df_dummy['Inspection
Type_OTHER']==0,0,df_dummy['fail'])
df_dummy['other_itype_other']= np.where(df_dummy['Inspection
Type_OTHER']==0,0,df_dummy['other'])

##########

df_dummy['Success_Canvass_prev']=(df_dummy.groupby(['License #'])
['canvass_pass'].cumsum()).shift()
df_dummy['Fail_Canvass_prev']=(df_dummy.groupby(['License #'])
['canvass_fail'].cumsum()).shift()
df_dummy['Other_Canvass_prev']=(df_dummy.groupby(['License #'])
['canvass_other'].cumsum()).shift()

df_dummy['Success_Complaint_prev']=(df_dummy.groupby(['License #'])
```

```
['complaint_pass'].cumsum()).shift()
df_dummy['Fail_Complaint_prev']=(df_dummy.groupby(['License #'])
['complaint_fail'].cumsum()).shift()
df_dummy['Other_Complaint_prev']=(df_dummy.groupby(['License #'])
['complaint_other'].cumsum()).shift()

df_dummy['Success_Consultation_prev']=(df_dummy.groupby(['License #'])
['consultation_pass'].cumsum()).shift()
df_dummy['Fail_Consultation_prev']=(df_dummy.groupby(['License #'])
['consultation_fail'].cumsum()).shift()
df_dummy['Other_Consultation_prev']=(df_dummy.groupby(['License #'])
['consultation_other'].cumsum()).shift()

df_dummy['Success_License_prev']=(df_dummy.groupby(['License #'])
['license_pass'].cumsum()).shift()
df_dummy['Fail_License_prev']=(df_dummy.groupby(['License #'])
['license_fail'].cumsum()).shift()
df_dummy['Other_License_prev']=(df_dummy.groupby(['License #'])
['license_other'].cumsum()).shift()

df_dummy['Success_Taskforce_prev']=(df_dummy.groupby(['License #'])
['taskforce_pass'].cumsum()).shift()
df_dummy['Fail_Taskforce_prev']=(df_dummy.groupby(['License #'])
['taskforce_fail'].cumsum()).shift()
df_dummy['Other_Taskforce_prev']=(df_dummy.groupby(['License #'])
['taskforce_other'].cumsum()).shift()

df_dummy['Success_Other_prev']=(df_dummy.groupby(['License #'])
['other_itype_pass'].cumsum()).shift()
df_dummy['Fail_Other_prev']=(df_dummy.groupby(['License #'])
['other_itype_fail'].cumsum()).shift()
df_dummy['Other_Other_prev']=(df_dummy.groupby(['License #'])
['other_itype_other'].cumsum()).shift()

##########

df_dummy['Success_Canvass_prev']=np.where(df_dummy['Success_Canvass_pr
ev'].isnull()==True,0.0,df_dummy['Success_Canvass_prev'])
df_dummy['Fail_Canvass_prev']=np.where(df_dummy['Fail_Canvass_prev'].i
snull()==True,0.0,df_dummy['Fail_Canvass_prev'])
df_dummy['Other_Canvass_prev']=np.where(df_dummy['Other_Canvass_prev']
.isnull()==True,0.0,df_dummy['Other_Canvass_prev'])

df_dummy['Success_Complaint_prev']=np.where(df_dummy['Success_Complain
t_prev'].isnull()==True,0.0,df_dummy['Success_Complaint_prev'])
df_dummy['Fail_Complaint_prev']=np.where(df_dummy['Fail_Complaint_prev
'].isnull()==True,0.0,df_dummy['Fail_Complaint_prev'])
df_dummy['Other_Complaint_prev']=np.where(df_dummy['Other_Complaint_pr
ev'].isnull()==True,0.0,df_dummy['Other_Complaint_prev'])
```

```
df_dummy['Success_Consultation_prev']=np.where(df_dummy['Success_Consu
ltation_prev'].isnull()==True,0.0,df_dummy['Success_Consultation_prev'
])
df_dummy['Fail_Consultation_prev']=np.where(df_dummy['Fail_Consultatio
n_prev'].isnull()==True,0.0,df_dummy['Fail_Consultation_prev'])
df_dummy['Other_Consultation_prev']=np.where(df_dummy['Other_Consultat
ion_prev'].isnull()==True,0.0,df_dummy['Other_Consultation_prev'])

df_dummy['Success_License_prev']=np.where(df_dummy['Success_License_pr
ev'].isnull()==True,0.0,df_dummy['Success_License_prev'])
df_dummy['Fail_License_prev']=np.where(df_dummy['Fail_License_prev'].i
snull()==True,0.0,df_dummy['Fail_License_prev'])
df_dummy['Other_License_prev']=np.where(df_dummy['Other_License_prev']
.isnull()==True,0.0,df_dummy['Other_License_prev'])

df_dummy['Success_Taskforce_prev']=np.where(df_dummy['Success_Taskforc
e_prev'].isnull()==True,0.0,df_dummy['Success_Taskforce_prev'])
df_dummy['Fail_Taskforce_prev']=np.where(df_dummy['Fail_Taskforce_prev
'].isnull()==True,0.0,df_dummy['Fail_Taskforce_prev'])
df_dummy['Other_Taskforce_prev']=np.where(df_dummy['Other_Taskforce_pr
ev'].isnull()==True,0.0,df_dummy['Other_Taskforce_prev'])

df_dummy['Success_Other_prev']=np.where(df_dummy['Success_Other_prev']
.isnull()==True,0.0,df_dummy['Success_Other_prev'])
df_dummy['Fail_Other_prev']=np.where(df_dummy['Fail_Other_prev'].isnul
l()==True,0.0,df_dummy['Fail_Other_prev'])
df_dummy['Other_Other_prev']=np.where(df_dummy['Other_Other_prev'].isn
ull()==True,0.0,df_dummy['Other_Other_prev'])

# demo of one of the columns
df_dummy['Success_Canvass_prev'][15000:15010]
```

```
192029     3.0
166990     4.0
145637     5.0
146172     5.0
132042     6.0
129761     6.0
111887     7.0
84840      8.0
69989      9.0
52932     10.0
Name: Success_Canvass_prev, dtype: float64
```

Add 6 new columns for prev_successful/ failed/ Neither cumulative results of 'CANVASS', 'COMPLAINT', 'LICENSE', 'TASKFORCE', and 'OTHER' types of INSPECTIONS for each facility :

'prev_cum_#violations_canvass_itype',

'prev_cum_#violations_complaint_itype',

'prev_cum_#violations_consultation_itype',

'prev_cum_#violations_license_itype',

'prev_cum_#violations_taskforce_itype',

'prev_cum_#violations_other_itype'

```
df_dummy['canvass_viol']= np.where(df_dummy['Inspection
Type_CANVASS']==0,0,df_dummy['#violations'])
df_dummy['complaint_viol']= np.where(df_dummy['Inspection
Type_COMPLAINT']==0,0,df_dummy['#violations'])
df_dummy['consultation_viol']= np.where(df_dummy['Inspection
Type_CONSULTATION']==0,0,df_dummy['#violations'])
df_dummy['license_viol']= np.where(df_dummy['Inspection
Type_LICENSE']==0,0,df_dummy['#violations'])
df_dummy['taskforce_viol']= np.where(df_dummy['Inspection
Type_TASKFORCE']==0,0,df_dummy['#violations'])
df_dummy['other_viol']= np.where(df_dummy['Inspection
Type_OTHER']==0,0,df_dummy['#violations'])


df_dummy['Canvass_prev_viol']=(df_dummy.groupby(['License #'])
['canvass_viol'].cumsum()).shift()
df_dummy['Complaint_prev_viol']=(df_dummy.groupby(['License #'])
['complaint_viol'].cumsum()).shift()
df_dummy['Consultation_prev_viol']=(df_dummy.groupby(['License #'])
['consultation_viol'].cumsum()).shift()
df_dummy['License_prev_viol']=(df_dummy.groupby(['License #'])
['license_viol'].cumsum()).shift()
df_dummy['Taskforce_prev_viol']=(df_dummy.groupby(['License #'])
['taskforce_viol'].cumsum()).shift()
df_dummy['Other_prev_viol']=(df_dummy.groupby(['License #'])
['other_viol'].cumsum()).shift()

df_dummy['Canvass_prev_viol']=np.where(df_dummy['Canvass_prev_viol'].i
snull()==True,0.0,df_dummy['Canvass_prev_viol'])
df_dummy['Complaint_prev_viol']=np.where(df_dummy['Complaint_prev_viol
'].isnull()==True,0.0,df_dummy['Complaint_prev_viol'])
df_dummy['Consultation_prev_viol']=np.where(df_dummy['Consultation_pre
v_viol'].isnull()==True,0.0,df_dummy['Consultation_prev_viol'])
df_dummy['License_prev_viol']=np.where(df_dummy['License_prev_viol'].i
```

```
snull()==True,0.0,df_dummy['License_prev_viol'])
df_dummy['Taskforce_prev_viol']=np.where(df_dummy['Taskforce_prev_viol
'].isnull()==True,0.0,df_dummy['Taskforce_prev_viol'])
df_dummy['Other_prev_viol']=np.where(df_dummy['Other_prev_viol'].isnul
l()==True,0.0,df_dummy['Other_prev_viol'])

##demo visual
df_dummy['Canvass_prev_viol'][0:10]
```

```
241405     0.0
241638     0.0
231139     5.0
206644    10.0
210615    14.0
184540    14.0
169965    17.0
156595    21.0
149419    24.0
136178    29.0
Name: Canvass_prev_viol, dtype: float64
```

```
df_dummy.columns[67:]
```

```
Index(['dbaname&address', 'ascii', 'Facility Type_Old',
'prev_inspection_type',
       'prev_result', 'prev_#_inspections', '#violations',
'prev_#_violations',
       'Inspection Type_CANVASS', 'Inspection Type_COMPLAINT',
       'Inspection Type_CONSULTATION', 'Inspection Type_LICENSE',
       'Inspection Type_OTHER', 'Inspection Type_TASKFORCE',
       'Results_Business Not Located', 'Results_Fail', 'Results_No
Entry',
       'Results_Not Ready', 'Results_Out of Business', 'Results_Pass',
       'Results_Pass w/ Conditions', 'pass', 'fail', 'other',
'canvass_pass',
       'canvass_fail', 'canvass_other', 'complaint_pass',
'complaint_fail',
       'complaint_other', 'consultation_pass', 'consultation_fail',
       'consultation_other', 'license_pass', 'license_fail',
'license_other',
       'taskforce_pass', 'taskforce_fail', 'taskforce_other',
       'other_itype_pass', 'other_itype_fail', 'other_itype_other',
       'Success_Canvass_prev', 'Fail_Canvass_prev',
'Other_Canvass_prev',
       'Success_Complaint_prev', 'Fail_Complaint_prev',
'Other_Complaint_prev',
       'Success_Consultation_prev', 'Fail_Consultation_prev',
       'Other_Consultation_prev', 'Success_License_prev',
'Fail_License_prev',
       'Other_License_prev', 'Success_Taskforce_prev',
'Fail_Taskforce_prev',
```

```
       'Other_Taskforce_prev', 'Success_Other_prev',
'Fail_Other_prev',
       'Other_Other_prev', 'canvass_viol', 'complaint_viol',
       'consultation_viol', 'license_viol', 'taskforce_viol',
'other_viol',
       'Canvass_prev_viol', 'Complaint_prev_viol',
'Consultation_prev_viol',
       'License_prev_viol', 'Taskforce_prev_viol', 'Other_prev_viol'],
      dtype='object')

df_dummy.to_csv('finalmlproj.csv')
```

ML Final Project

Adding Zillow and Crime Data to Inspections

Bue, Alex

2023-03-10

Contents

1 Libraries

```
library(dplyr)
library(tidyverse)
library(data.table)
library(lubridate)
library(stringr)
library(tidyr)
library(zoo)
library(writexl)
library(ggplot2)
library(scales)
```

2 Inspections

2.1 Loading

```
df <- read.csv("Food_inspections.csv")
```

2.2 Cleaning

2.2.1 Unique values per column

```
unique.per.column <- sapply(df, function(x) n_distinct(x))

print(unique.per.column)
```

2.2.2 Creating Dates

We will match prices by month & year plus zipcode. So I create here a "month.year" column.

```
df$Inspection.Date <- mdy(df$Inspection.Date)

df$month.year <- as.yearmon(df$Inspection.Date, "%m%Y")

df$previous_month_year <- df$Inspection.Date %m+% months(-1)

head(df$Inspection.Date)
head(df$previous_month)

df$previous_month_year <- as.yearmon(df$previous_month, "%m%Y")

str(df)
```

2.2.3 Changing Zip to Zipcodes

```
df = rename(df, Zipcode = Zip)
colnames(df)
```

3 Zillow

3.1 Loading

```
zillow <- read.csv("zipcodes.csv")

zillow <- zillow %>%
  filter(zillow$City %like% "Chicago")

str(zillow)
```

3.2 Cleaning

3.2.1 Removing "X" form Column Names

```
dates <- 10:length(zillow)

x.dates <- colnames(zillow[dates])

replace.dates <- substr(
  names(zillow[dates]),
  start = 2, stop = 20
)

dates = zillow %>%
  select(x.dates) %>%
  setNames(replace.dates)

zillow <- cbind(zillow[1:9], dates)

colnames(zillow)
```

3.2.2 Creating Rows from Date Columns

```
not.dates <- colnames(zillow[1:9])

zillow <- zillow %>%
  pivot_longer(cols = !not.dates, names_to = "Date", values_to = "Price")

str(zillow)
```

3.2.3 Removing Variables

```
zillow <- data.frame(
  Zipcode = zillow$RegionName, Price = zillow$Price, Date = zillow$Date
)

str(zillow)
```

3.2.4 Creating month.year column.

Titled "previous month" for consistency with Inspections.

```
zillow$Date <- mdy(zillow$Date)

zillow$previous_month_year <- as.yearmon(zillow$Date, "%m%Y")

str(zillow)
```

3.3 Joining Zillow and Inspections

```
df <- left_join(df, zillow, by = c("previous_month_year", "Zipcode"))

colnames(df)
```

3.4 Cleaning

3.4.1 Renaming columns for clarity

```
df <- rename(df, Zillow.Price.index = "Price", Zillow.Date = "Date")

colnames(df)
```

4 Crime

4.1 Loading

```
crime <- read.csv("Crimes.csv")

zip_codes <- read.csv("Zip_Codes.csv")
```

4.2 Cleaning

4.2.1 Adding Zip Codes According to Boundaries

```
crime <- rename(crime, Zip_code_boundaries = "Boundaries...ZIP.Codes")

zip_codes <- rename(
  zip_codes, Zip_code_boundaries = "OBJECTID", Zip_Codes = "ZIP"
)

crime <- crime %>%
  left_join(zip_codes, by = "Zip_code_boundaries")
```

4.2.2 Unique Values

```
unique.per.column <- sapply(crime, function(x) n_distinct(x))

print(unique.per.column)
```

4.2.3 Creating Dates

Again, "month year" called previous_month_year for matching to DF.

```
crime$Date <- mdy_hms(
  crime$Date, tz = "America/Chicago", locale = Sys.getlocale("LC_TIME")
)

crime$previous_month_year <- as.yearmon(crime$Date, "%m%Y")
```

4.2.4 Removing Longitude, latitude, and geographic predictors

```
column.names.to.remove <- c(
  "Y.Coordinate", "X.Coordinate", "Longitude", "Latitude",
  "Location", "Ward", "Beat", "Census.Tracts", "Zip_code_boundaries",
  "Community.Area", "Community.Areas", "Historical.Wards.2003.2015",
  "Police.Districts", "Police.Beats", "Wards", "Block", "District",
  "Zip.Codes"
)


crime <- crime[, !names(crime) %in%
  column.names.to.remove]

colnames(crime)
```

4.2.5 Removing Other Superfluous, idiosyncratic variables

```
column.names.to.remove <- c(
  "ID", "Case.Number", "Updated.On", "FBI.Code", "IUCR", "Description"
)


crime <- crime[, !names(crime) %in%
  column.names.to.remove]

colnames(crime)
```

4.2.6 Grouping

```
crime <- crime %>%
  group_by(previous_month_year, Zip_Codes, Primary.Type) %>%
  tally()

head(crime)
```

4.2.7 Filtering

```
crime <- crime %>%
  pivot_wider(
    names_from = Primary.Type, values_from = n, values_fill = 0
  )


colnames(crime)
```

4.2.8 Renaming

```
crime = rename(crime, Zipcode = Zip_Codes)
colnames(crime)
```

4.3 Joining DF and Crime data

```
df <- left_join(df, crime, by = c("previous_month_year", "Zipcode"))
colnames(df)
head(df)
```

5 Saving to R Object

```
save(df, file = "Inspections + Zillow + Crime.RData")

write.csv(
  df, "C:/Users/alexo/OneDrive - The University of Chicago/Desktop/School/Classes/Machine Learning/Proj
)
```