Assignment 4

February 1, 2021

---

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

0.1 Assignment 4 - Understanding and Predicting Property Maintenance Fines

This assignment is based on a data challenge from the Michigan Data Science Team (MDST).

The Michigan Data Science Team (MDST) and the Michigan Student Symposium for Interdisciplinary Statistical Sciences (MSSISS) have partnered with the City of Detroit to help solve one of the most pressing problems facing Detroit - blight. Blight violations are issued by the city to individuals who allow their properties to remain in a deteriorated condition. Every year, the city of Detroit issues millions of dollars in fines to residents and every year, many of these fines remain unpaid. Enforcing unpaid blight fines is a costly and tedious process, so the city wants to know: how can we increase blight ticket compliance?

The first step in answering this question is understanding when and why a resident might fail to comply with a blight ticket. This is where predictive modeling comes in. For this assignment, your task is to predict whether a given blight ticket will be paid on time.

All data for this assignment has been provided to us through the Detroit Open Data Portal. **Only the data already included in your Coursera directory can be used for training the model for this assignment.** Nonetheless, we encourage you to look into data from other Detroit datasets to help inform feature creation and model selection. We recommend taking a look at the following related datasets:

- Building Permits
- Trades Permits
- Improve Detroit: Submitted Issues
- DPD: Citizen Complaints
- Parcel Map

We provide you with two data files for use in training and validating your models: train.csv and test.csv. Each row in these two files corresponds to a single blight ticket, and includes information about when, why, and to whom each ticket was issued. The target variable is compliance, which is True if the ticket was paid early, on time, or within one month of the hearing data, False if the ticket was paid after the hearing date or not at all, and Null if the violator was found not responsible. Compliance, as well as a handful of other variables that will not be available at test-time, are only included in train.csv.

Note: All tickets where the violators were found not responsible are not considered during evaluation. They are included in the training set as an additional source of data for visualization, and to enable unsupervised and semi-supervised approaches. However, they are not included in the test set.

File descriptions (Use only this data for training your model!)

```
readonly/train.csv - the training set (all tickets issued 2004-2011)
readonly/test.csv - the test set (all tickets issued 2012-2016)
readonly/addresses.csv & readonly/latlons.csv - mapping from ticket id to addresses
 Note: misspelled addresses may be incorrectly geolocated.
```

Data fields

train.csv & test.csv

```
ticket_id - unique identifier for tickets
agency_name - Agency that issued the ticket
inspector_name - Name of inspector that issued the ticket
violator_name - Name of the person/organization that the ticket was issued to
violation_street_number, violation_street_name, violation_zip_code - Address where
mailing_address_str_number, mailing_address_str_name, city, state, zip_code, non_us
ticket_issued_date - Date and time the ticket was issued
hearing_date - Date and time the violator's hearing was scheduled
violation_code, violation_description - Type of violation
disposition - Judgment and judgement type
fine_amount - Violation fine amount, excluding fees
admin_fee - $20 fee assigned to responsible judgments
```

state_fee - $10 fee assigned to responsible judgments late_fee - 10% fee assigned to responsible judgments discount_amount - discount applied, if any clean_up_cost - DPW clean-up or graffiti removal cost judgment_amount - Sum of all fines and fees grafitti_status - Flag for graffiti violations

train.csv only

```
payment_amount - Amount paid, if any
payment_date - Date payment was made, if it was received
payment_status - Current payment status as of Feb 1 2017
balance_due - Fines and fees still owed
collection_status - Flag for payments in collections
compliance [target variable for prediction]
 Null = Not responsible
 0 = Responsible, non-compliant
 1 = Responsible, compliant
compliance_detail - More information on why each ticket was marked compliant or non
```

0.2 Evaluation

Your predictions will be given as the probability that the corresponding blight ticket will be paid on time.

The evaluation metric for this assignment is the Area Under the ROC Curve (AUC).

Your grade will be based on the AUC score computed for your classifier. A model which with an AUROC of 0.7 passes this assignment, over 0.75 will recieve full points. ___

For this assignment, create a function that trains a model to predict blight ticket compliance in Detroit using `readonly/train.csv`. Using this model, return a series of length 61001 with the data being the probability that each corresponding ticket from `readonly/test.csv` will be paid, and the index being the ticket_id.

Example:

```
ticket_id
   284932    0.531842
   285362    0.401958
   285361    0.105928
   285338    0.018572
             ...
   376499    0.208567
   376500    0.818759
   369851    0.018528
   Name: compliance, dtype: float32
```

0.2.1 Hints

- Make sure your code is working before submitting it to the autograder.
- Print out your result to see whether there is anything weird (e.g., all probabilities are the same).
- Generally the total runtime should be less than 10 mins. You should NOT use Neural Network related classifiers (e.g., MLPClassifier) in this question.
- Try to avoid global variables. If you have other functions besides blight_model, you should move those functions inside the scope of blight_model.
- Refer to the pinned threads in Week 4's discussion forum when there is something you could not figure it out.

```
In [1]: ###########################################################################

        # Final Double commented whole cell - Jan 2021

        # import pandas as pd
        # # blight_df = pd.read_csv('train.csv')
        # # UnicodeDecodeError: 'utf-8' codec can't decode byte 0x92 in position 9

        # # blight_df = pd.read_fwf('train.csv')
```

```
# # Works but not proper formatting

# # blight_df = pd.read_table('train.csv')
# # UnicodeDecodeError: 'utf-8' codec can't decode byte 0x92 in position 61

# # blight_df = pd.read_excel('train.csv')
# # XLRDError: Unsupported format, or corrupt file: Expected BOF record; fo


# ######
# # google -
# # The data is indeed not encoded as UTF-8;
# # df1 = pd.read_csv("mycsv.csv", sep=";", encoding='cp1252')

# # 0x92 is a smart quote(’) of Windows-1252. It simply doesn't exist in un
# # Use encoding='cp1252' will solve the issue.
# ######




# blight_df = pd.read_csv('train.csv', encoding='cp1252')


# ##WORKS

# # Check warning - /opt/conda/lib/python3.6/site-packages/IPython/core/int
# #   interactivity=interactivity, compiler=compiler, result=result)




# # blight_df



# # blight_df.shape
# # (250306, 34)





# blight_df.head()
```

In [2]:
```
################################################################################
```

```
# blight_df.columns


# Index(['ticket_id', 'agency_name', 'inspector_name', 'violator_name',
#        'violation_street_number', 'violation_street_name',
#        'violation_zip_code', 'mailing_address_str_number',
#        'mailing_address_str_name', 'city', 'state', 'zip_code',
#        'non_us_str_code', 'country', 'ticket_issued_date', 'hearing_date
#        'violation_code', 'violation_description', 'disposition', 'fine_am
#        'admin_fee', 'state_fee', 'late_fee', 'discount_amount',
#        'clean_up_cost', 'judgment_amount', 'payment_amount', 'balance_due
#        'payment_date', 'payment_status', 'collection_status',
#        'grafitti_status', 'compliance_detail', 'compliance'],
#       dtype='object')
```

In [3]:
```
###########################################################################


# blight_df.dtypes


# ticket_id                      int64
# agency_name                   object
# inspector_name                object
# violator_name                 object
# violation_street_number      float64
# violation_street_name         object
# violation_zip_code           float64
# mailing_address_str_number   float64
# mailing_address_str_name      object
# city                          object
# state                         object
# zip_code                      object
# non_us_str_code               object
# country                       object
# ticket_issued_date            object
# hearing_date                  object
# violation_code                object
# violation_description         object
# disposition                   object
# fine_amount                  float64
# admin_fee                    float64
# state_fee                    float64
# late_fee                     float64
# discount_amount              float64
# clean_up_cost                float64
# judgment_amount              float64
# payment_amount               float64
```

```
# balance_due                     float64
# payment_date                     object
# payment_status                   object
# collection_status                object
# grafitti_status                  object
# compliance_detail                object
# compliance                      float64
# dtype: object





# blight_df.ftypes?

# Type:        property
# String form: <property object at 0x7fbb27c0c1d8>
# Docstring:  Return the ftypes (indication of sparse/dense and dtype) in t






# blight_df.ftypes

# ticket_id                          int64:dense
# agency_name                       object:dense
# inspector_name                    object:dense
# violator_name                     object:dense
# violation_street_number          float64:dense
# violation_street_name             object:dense
# ....

# ALL DENSE
```

In [4]:
```
################################################################################
```

```
# blight_df.describe()



#         ticket_id       violation_street_number       violation_zip_coo
# count        250306.000000       2.503060e+05        0.0        2.46704(
# mean        152665.543099       1.064986e+04       NaN        9.149788e
# std        77189.882881        3.188733e+04       NaN        3.602034e+0
```

```
# min        18645.000000       0.000000e+00       NaN       1.000000e+0
# 25%        86549.250000       4.739000e+03       NaN       5.440000e+0
# 50%        152597.500000       1.024400e+04       NaN       2.456000e-
# 75%        219888.750000       1.576000e+04       NaN       1.292725e-
# max        366178.000000       1.415411e+07       NaN       5.111345e-
```

```
In [5]: ###########################################################################

# Data Preprocessing/analyzing STEPS


# 1. Removing Null Target value rows
# #2#. Separating independent and dependent variables in individual datafra
# #    - after further analysis, felt that this should be done in last, jus
# #      , as we need target column along with whole data to do filtering/c
# 2. Removing Future columns - which will not be available at test time / r
# 3. Checking Class distributin - skewed/imbalanced or not
# 4. Removing columns which have 99% null values - after checking the imbal
# 5. Removing columns which have 99% same values - after checking the imbal
# 6. Removing columns which have 99% unique values - after checking the imb
# 7. Removing non relevant columns - that seem non relevant
# 8. Removing dependant/correlated columns - that seem non relevant

# 9. Handling ADDRESS categorical values? Microscopic or Macroscopic level?

# 10. Handling DATE categorical values? - Dummify, or use bins?

# 11. dummyfying other categorical values - but how many?
#        - What if a column has 5000 distinct values? should we dummify?

# 12. Check CORRELATION???
# 13. Plot data to visualize??
# 14. Separating independent and dependent variables in individual datafran
# 15. Train Test Split
# 16. Run models
# 17. Tune models (tune hyperparameters) individually - to select the best
# 18. Evaluate and finalize



###
# OTHER STEPS to be included??
# Step 1 - Make a copy of original dataframe
# Step 2 - Before doing complete feature engineering - SHOULD we do a dry r
```

```
# 1. Removing columns which have 99% null values - violation_zip_code, non_
# 2. Removing columns which have 99% unique values -
# 3. Removing columns which have 99% same values -
# 4. dummyfying categorical values -
```

In [6]:
```
#################################################################################
### From AUG 2020


###WHAT ALL PREPROCESSING we need to do with the RAW data???
#1. remove NULL columns altogether? - where ALL / maximum null
#2. fill NULL values - with some constant / static / average value - does t
#3. drop NULL values / rows - dropna
#4. scaling/NORMALIZATION/standardization ??? - of int/float type columns
#5. segregate / process different data type columns
#6. check for CORRELATION between input paramters??? - heatmap, matrix, etc
#7. Removing non number columns - non int, float - INVALID STEP - HAVE TO C
#8. check unique - remove ones with 100% / 99.99% / 90% same values - or wh
#9. check unique - remove ones with 100% / 99.99% / 90% UNIQUE values - or
#10. categorization/splitting - get_dummies - after checking unique counts



#QUESTIONS - (Pre processing)
#1. what should be the ideal percent of null values in a column so that it
#2. When should we do scaling/NORMALIZATION/standardization
#3. what should we do after we have checked correlation? what if we find th
#4. which columns should we dummify? - say multiple columns have few unique
#5. can we drop instances/rows containing NaN values? what is the maximum p
#6. But if we apply above policy of dropping rows containing NaN values, an
#7. can we drop instances/rows containing 99% same values? what is the maxi
```

In [7]:
```
#################################################################################



        # 0         ticket_id         250306
# 1         agency_name         5
# 2         inspector_name         173
        # 3         violator_name         119993
        # 4         violation_street_number         19175
# 5         violation_street_name         1791
                # 6         violation_zip_code         1
# 7         mailing_address_str_number         15827
```

```
# 8        mailing_address_str_name        37897
# 9        city        5184
# 10        state        60
# 11        zip_code        5643
                # 12        non_us_str_code        3
# 13        country        5
        # 14        ticket_issued_date        86979
# 15        hearing_date        6223
# 16        violation_code        235
# 17        violation_description        258
# 18        disposition        9
# 19        fine_amount        44
# 20        admin_fee        2
# 21        state_fee        2
# 22        late_fee        37
# 23        discount_amount        13
                # 24        clean_up_cost        1
# 25        judgment_amount        57


# 26        payment_amount        533
# 27        balance_due        606
# 28        payment_date        2308
# 29        payment_status        3
# 30        collection_status        2
# 31        grafitti_status        2
# 32        compliance_detail        10
        # 33        compliance        3




##############################################################################


# train.csv only - so have to exclude these columns.

# payment_amount - Amount paid, if any
# payment_date - Date payment was made, if it was received
# payment_status - Current payment status as of Feb 1 2017
# balance_due - Fines and fees still owed
# collection_status - Flag for payments in collections
# compliance [target variable for prediction]
#  Null = Not responsible
#  0 = Responsible, non-compliant
#  1 = Responsible, compliant
# compliance_detail - More information on why each ticket was marked compli
```

In [8]:
```
###########################################################################

# Final Double commented whole cell - Jan 2021
# blight_train_df = blight_df.copy()
```

In [9]:
```
###########################################################################

# Final Double commented whole cell - Jan 2021

# # FIRST STEP
# # REMOVING NULL COMPLIANCE






# # blight_train_df['compliance'].unique()
# # array([  0.,   1.,  nan])


# # compliance_srs = blight_train_df['compliance']
# # compliance_srs.isnull()
# # ~compliance_srs.isnull()


# # blight_train_df.where(~blight_train_df['compliance'].isnull())






# blight_train_df = blight_train_df[~blight_train_df['compliance'].isnull()

# # 159880 rows × 34 columns
```

In [10]:
```
###########################################################################

# Final Double commented whole cell - Jan 2021

# ######### SECOND STEP
# ####### SEPARATING target variable and independent variables
# # #    - after further analysis, felt that this should be done in last,
# # #      , as we need target column along with whole data to do filterin

# ####X_blight = blight_train_df[:, :-1]
# ###blight_train_df[[:, 1:10]]
```

```
# # X_blight = blight_train_df.iloc[:, :-1]
# y_blight = blight_train_df.iloc[:, -1]



# # As the change of step - moving dependant and independent variables int
# # is a major change and was done later, it affected whole notebook, requ
# # So for time being taking X_blight as blight_train_df only.

# X_blight = blight_train_df.iloc[:, :]
```

In [11]:
```
##########################################################################


# Final Double commented whole cell - Jan 2021

# # SECOND STEP
# # Also removing future columns - which are not available at test time?

# not_in_test_cols = ['payment_amount', 'payment_date', 'payment_status',


# # blight_train_df.columns in ['payment_amount']

# # blight_train_df.columns.isin(['payment_amount'])

# # blight_train_df.columns

# # blight_train_df.columns.isin(['payment_amount'])
# # array([False, False, False, False, False, False, False, False, False,
# #        False, False, False, False, False, False, False, False, False,
# #        False, False, False, False, False, False, False, False,  True,
# #        False, False, False, False, False, False, False], dtype=bool)


# # blight_train_df.columns.isin(not_in_test_cols)

# # array([False, False, False, False, False, False, False, False, False,
# #        False, False, False, False, False, False, False, False, False,
# #        False, False, False, False, False, False, False, False,  True,
# #         True,  True,  True,  True, False,  True, False], dtype=bool)



# # ~blight_train_df.columns.isin(not_in_test_cols)

# # array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
```

```
# #         True,  True,  True,  True,  True,  True,  True,  True,  True,
# #         True,  True,  True,  True,  True,  True,  True,  True, False,
# #        False, False, False, False,  True, False,  True], dtype=bool)



# # X_blight.columns[~X_blight.columns.isin(not_in_test_cols)]

# # Index(['ticket_id', 'agency_name', 'inspector_name', 'violator_name',
# #        'violation_street_number', 'violation_street_name',
# #        'violation_zip_code', 'mailing_address_str_number',
# #        'mailing_address_str_name', 'city', 'state', 'zip_code',
# #        'non_us_str_code', 'country', 'ticket_issued_date', 'hearing_da
# #        'violation_code', 'violation_description', 'disposition', 'fine
# #        'admin_fee', 'state_fee', 'late_fee', 'discount_amount',
# #        'clean_up_cost', 'judgment_amount', 'grafitti_status'],
# #       dtype='object')


# sel_cols = X_blight.columns[~X_blight.columns.isin(not_in_test_cols)]

# X_blight = X_blight[sel_cols]


# # # X_blight.columns
# # # Index(['ticket_id', 'agency_name', 'inspector_name', 'violator_name
# # #        'violation_street_number', 'violation_street_name',
# # #        'violation_zip_code', 'mailing_address_str_number',
# # #        'mailing_address_str_name', 'city', 'state', 'zip_code',
# # #        'non_us_str_code', 'country', 'ticket_issued_date', 'hearing_
# # #        'violation_code', 'violation_description', 'disposition', 'fi
# # #        'admin_fee', 'state_fee', 'late_fee', 'discount_amount',
# # #        'clean_up_cost', 'judgment_amount', 'grafitti_status'],
# # #       dtype='object')


# # # X_blight.shape
# # # (159880, 27)

# # # len(X_blight.columns)
# # # 27
# # # length of columns reduced to 27 from 33 (excluding 34th initial comp
```

```
In [12]: # Final Double commented whole cell - Jan 2021

         # # THIRD STEP
         # # Checking the target variable - what is the class distribution - is it
         # # Differences in the class distribution for an imbalanced classificatio
```

```
# # y_blight
# # 0         0.0
# # 1         1.0
# # 5         0.0
# # 6         0.0

# # np.bincount(y_blight)
# # TypeError: Cannot cast array data from dtype('float64') to dtype('int6


# y_blight = y_blight.astype('int64')
# # y_blight


# # np.bincount(y_blight)
# # array([148283,  11597])

# import numpy as np

# np.bincount(y_blight)/len(y_blight)*100
# # array([ 92.74643483,   7.25356517])

# # 0 is not paid, 1 is paid
# # so 93% have not paid, only 7% have paid

# # This seems like a skewed / imbalanced class distribution
# # So we will have to prepare data accordingly.
```

In [13]:

```
# Final Double commented whole cell - Jan 2021


# # FOURTH STEP
# # Removing / Ignoring columns which contain 99% null values
# # - or 95% - or 90% - what should be the percentage???
# # Is 10% available values enough?
# # Or we should have atleast 50% available valeus??
# # What should be the percentage of available values to make a meaningful

# # Depends on class distribution also




# ########################################################################

# # blight_train_df.isnull().values.any()
# # True
```

```
# # blight_train_df.isnull().sum()

# # ticket_id                           0
# # agency_name                         0
# # inspector_name                      0
# # violator_name                      34
# # violation_street_number             0
# # violation_street_name               0
# # violation_zip_code             250306
# # mailing_address_str_number       3602
# # mailing_address_str_name            4
# # city                                0
# # state                              93
# # zip_code                            1
# # non_us_str_code                250303
# # country                             0
# # ticket_issued_date                  0
# # hearing_date                    12491
# # violation_code                      0
# # violation_description               0
# # disposition                         0
# # fine_amount                         1
# # admin_fee                           0
# # state_fee                           0
# # late_fee                            0
# # discount_amount                     0
# # clean_up_cost                       0
# # judgment_amount                     0
# # payment_amount                      0
# # balance_due                         0
# # payment_date                   209193
# # payment_status                      0
# # collection_status              213409
# # grafitti_status                250305
# # compliance_detail                   0
# # compliance                      90426
# # dtype: int64
```

```
# # blight_train_df.count?
# # Signature: blight_train_df.count(axis=0, level=None, numeric_only=Fals
# # Docstring: Return Series with number of non-NA/null observations over
# #     data as well (detects NaN and None)
```

```
# # blight_train_df.count()

# # ticket_id                      250306
# # agency_name                    250306
# # inspector_name                 250306
# # violator_name                  250272
# # violation_street_number        250306
# # violation_street_name          250306
# # violation_zip_code                  0
# # mailing_address_str_number     246704
# # mailing_address_str_name       250302
# # city                           250306
# # state                          250213
# # zip_code                       250305
# # non_us_str_code                     3
# # country                        250306
# # ticket_issued_date             250306
# # hearing_date                   237815
# # violation_code                 250306
# # violation_description          250306
# # disposition                    250306
# # fine_amount                    250305
# # admin_fee                      250306
# # state_fee                      250306
# # late_fee                       250306
# # discount_amount                250306
# # clean_up_cost                  250306
# # judgment_amount                250306
# # payment_amount                 250306
# # balance_due                    250306
# # payment_date                    41113
# # payment_status                 250306
# # collection_status               36897
# # grafitti_status                     1
# # compliance_detail              250306
# # compliance                     159880
# # dtype: int64
```

```
# # blight_train_df.violation_zip_code

# # 0        NaN
# # 1        NaN
# # 2        NaN
# # 3        NaN
# # 4        NaN




# # You could subtract the total length from the count of non-nan values:
# # count_nan = len(df) - df.count()
# # You should time it on your data. For small Series got a 3x speed up ir


# # count_nan = len(blight_train_df) - blight_train_df.count()
# # count_nan


# ##########################################################################


# # X_blight.count gives the number of non null/NaN values
# # X_blight.count()

# # ticket_id                     159880
# # agency_name                   159880
# # inspector_name                159880
# # violator_name                 159854
# # violation_street_number       159880
# # violation_street_name         159880
# # violation_zip_code                 0
# # mailing_address_str_number    157322
# # mailing_address_str_name      159877
# # city                          159880
# # state                         159796
# # zip_code                      159879
# # non_us_str_code                    3
```

```
# # X_blight.count()/X_blight.shape[0]
# # This will give the percentage of non null values

# # ticket_id                     1.000000
# # agency_name                   1.000000
# # inspector_name                1.000000
# # violator_name                 0.999837
# # violation_street_number       1.000000
# # violation_street_name         1.000000
# # violation_zip_code            0.000000
# # mailing_address_str_number    0.984001



# # If we want the percentage of null values to be say 95% - then we want
# # to be less than than 5%





# # we get more than 5% null value columns by selecting columns with less
# # X_blight.count()/X_blight.shape[0]<0.05


# # ticket_id                     False
# # agency_name                   False
# # inspector_name                False
# # violator_name                 False
# # violation_street_number       False
# # violation_street_name         False
# # violation_zip_code             True
# # mailing_address_str_number    False
# # mailing_address_str_name      False
# # city                          False
# # state                         False
# # zip_code                      False
# # non_us_str_code                True



# # we get more than 5% null value columns by selecting columns with less
# # X_blight.columns[X_blight.count()/X_blight.shape[0]<0.95]
# # Index(['violation_zip_code', 'non_us_str_code', 'grafitti_status'], dt



# # we get more than 10% null value columns by selecting columns with less
```

```
# # X_blight.columns[X_blight.count()/X_blight.shape[0]<0.90]
# # Index(['violation_zip_code', 'non_us_str_code', 'grafitti_status'], dt


# # we get more than 99% null value columns by selecting columns with less
# # Since it is a imbalanced class with 93:7 ratio - so 99% seems appropri
# # X_blight.columns[X_blight.count()/X_blight.shape[0]<0.01]
# # Index(['violation_zip_code', 'non_us_str_code', 'grafitti_status'], dt


# # same columns for null values more than 5% and 10% and 1%


# #################


# # Just checking columns with less than 50% available/non null values
# # - though can we ignore columns with 50% available values
# # columns with more than 50% null value columns by selecting columns wit

# # X_blight.columns[X_blight.count()/X_blight.shape[0]<0.50]
# # Index(['violation_zip_code', 'non_us_str_code', 'grafitti_status'], dt
# # Same for this data set - but not in general - what should we do in ger


# # Just checking columns with less than 70% available/non null values
# # - though can we ignore columns with 30% available values
# # columns with more than 70% null value columns by selecting columns wit
# # X_blight.columns[X_blight.count()/X_blight.shape[0]<0.30]
# # Index(['violation_zip_code', 'non_us_str_code', 'grafitti_status'], dt
# # Same for this data set - but not in general - what should we do in ger


# #################


# # FOR THIS particular problem, with imbalanced classes of ratio 93 to 7,
# # which means columns with more than 1% available data

# # Since it is a imbalanced class with 93:7 ratio - so 99% seems appropri
```

```
# # X_blight.columns[X_blight.count()/X_blight.shape[0]<0.01]
# # Index(['violation_zip_code', 'non_us_str_code', 'grafitti_status'], dt


# columns_with_null = X_blight.columns[X_blight.count()/X_blight.shape[0]<

# sel_nonnull_cols = X_blight.columns[~X_blight.columns.isin(columns_with_

# X_blight = X_blight[sel_nonnull_cols]



# # X_blight.columns

# # Index(['ticket_id', 'agency_name', 'inspector_name', 'violator_name',
# #        'violation_street_number', 'violation_street_name',
# #        'mailing_address_str_number', 'mailing_address_str_name', 'city
# #        'state', 'zip_code', 'country', 'ticket_issued_date', 'hearing_
# #        'violation_code', 'violation_description', 'disposition', 'fine
# #        'admin_fee', 'state_fee', 'late_fee', 'discount_amount',
# #        'clean_up_cost', 'judgment_amount'],
# #       dtype='object')


# # ['violation_zip_code', 'non_us_str_code', 'grafitti_status'] columns r



# # # len(X_blight.columns)
# # # 24
# # # length of columns reduced to 24 from previous 27 and initial 33
```

In [14]:

```
##########################################################################


# Final Double commented whole cell - Jan 2021

# # Had missed this from the description this time in Jan 2021 - saw in Au
# # we have 2 more csv's
# # addresses.csv and latlons.csv

# # readonly/addresses.csv & readonly/latlons.csv - mapping from ticket id
# #  Note: misspelled addresses may be incorrectly geolocated.


# addresses = pd.read_csv("readonly/addresses.csv",encoding="cp1252")
```

```
# latlons = pd.read_csv("readonly/latlons.csv",encoding="cp1252")
# X_blight = X_blight.merge(addresses, left_on='ticket_id', right_on='tick
# X_blight = X_blight.merge(latlons, left_on='address', right_on='address


# # Aug 2020
# # train_address = traindata.merge(addresses, left_on='ticket_id', right_
# # train_allDetails = train_address.merge(latlons, left_on='address', rig
```

In [15]:
```
##########################################################################

# Final Double commented whole cell - Jan 2021

# # # FIFTH STEP
# # # Removing / Ignoring columns which contain 99% same values
# # # - or 95% - or 90% - what should be the percentage???

# # # Also

# # # SIXTH STEP
# # # Removing columns which have 99% unique values - after checking the i

# # # As the two are linked - which have 99% same values and which have 99


# ########################################################################


# # ### Trying to get unique values in all columns from a single command
# # ### Could not find a single command.
# # ### Have to use a loop through df.columns
# # ### This is not used




# # # type(blight_train_df.columns)
# # # pandas.indexes.base.Index

# # # type(blight_train_df.columns.values)
# # # numpy.ndarray

# # # pd.unique(blight_train_df.columns.values)
# # # Only gives column names


# # # pd.unique(blight_train_df['agency_name'])
# # # array(['Buildings, Safety Engineering & Env Department',
```

```
# # #        'Health Department', 'Department of Public Works',
# # #        'Detroit Police Department', 'Neighborhood City Halls'], dtyp


# # # pd.unique(blight_train_df['agency_name'])



# # # type(blight_train_df.columns.tolist())
# # # list



# # # blight_train_df[blight_train_df.columns.tolist()]


# # # pd.unique(blight_train_df[['agency_name', 'inspector_name', 'city']]
# # # ValueError: cannot copy sequence with size 3 to array axis with dime


# # # pd.unique(blight_train_df['agency_name'], blight_train_df['inspector
# # # TypeError: unique() takes 1 positional argument but 2 were given



# ##################################################################



# # # len(X_blight['agency_name'].unique())
# # # 5

# # # [len(X_blight[col].unique()) for col in X_blight.columns]


# # # [159880,
# # #  5,
# # #  159,
# # #  84657,
# # #  18096,
# # #  1716,
# # #  14091,
# # #  28441,
# # #  4093,
# # #  60,
# # #  4623,
# # #  5,
# # #  68097,
```

```
# # #  5971,
# # #  189,
# # #  207,
# # #  4,
# # #  40,
# # #  1,
# # #  1,
# # #  37,
# # #  13,
# # #  1,
# # #  57]



# # # unique_counts = [len(X_blight[col].unique()) for col in X_blight.col
# # # Can also use X_blight[col].nunique() instead of len(X_blight[col].un


# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]

# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts


# ################

# # # DataFrame.nunique

# # #We don't have to write list comprehension - but we can use DataFrame.
# # # of pandas - it is available from probably version 1 (1.2.1) - our pa
# # # pd.__version__
# # # '0.19.2'


# # # pandas.Series.nunique
# # # Series has nunique available in this version

# ################

# # # unique_counts_df

# # #          cols         unique_counts
# # # 0         ticket_id         159880
# # # 1         agency_name         5
# # # 2         inspector_name         159
# # # 3         violator_name         84657
# # # 4         violation_street_number         18096
# # # 5         violation_street_name         1716
# # # 6         mailing_address_str_number         14091
```

```
# # # 7        mailing_address_str_name        28441
# # # 8        city        4093
# # # 9        state        60
# # # 10        zip_code        4623
# # # 11        country        5
# # # 12        ticket_issued_date        68097
# # # 13        hearing_date        5971
# # # 14        violation_code        189
# # # 15        violation_description        207
# # # 16        disposition        4
# # # 17        fine_amount        40
# # # 18        admin_fee        1
# # # 19        state_fee        1
# # # 20        late_fee        37
# # # 21        discount_amount        13
# # # 22        clean_up_cost        1
# # # 23        judgment_amount        57

# ###############
# # # just checking unique counts on original dataframe - if missed anythi

# # # unique_counts = [len(blight_train_df[col].unique()) for col in bligh
# # # unique_counts_df = pd.DataFrame({'cols':blight_train_df.columns, 'un
# # # unique_counts_df


# ###############


# # # unique counts ratio of total
# # # Unique counts ratio will be used to remove columns which have 99% Un
# # # Or - 95%  - or 90 % - how to decide?





# # unique_counts_df['ratio'] = unique_counts/X_blight.shape[0]
# # TypeError: unsupported operand type(s) for /: 'list' and 'int'


# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s
# # # unique_counts_df


# # #         cols        unique_counts        ratio
# # # 0        ticket_id        159880        1.000000
# # # 1        agency_name        5        0.000031
```

```
# # # 2        inspector_name        159        0.000994
# # # 3        violator_name        84657        0.529503
# # # 4        violation_street_number        18096        0.113185



# ###############

# # # VALUE_COUNTS()
# # # value_counts() will show the distinct element and their number of oc
# # # CHECK its usage later - how can this be used.




# # # X_blight['agency_name'].value_counts()

# # # Buildings, Safety Engineering & Env Department    95863
# # # Department of Public Works                        52445
# # # Health Department                                  7107
# # # Detroit Police Department                          4464
# # # Neighborhood City Halls                               1
# # # Name: agency_name, dtype: int64
```

In [16]:

```
##########################################################################

# Final Double commented whole cell - Jan 2021

# # # Back to # FIFTH STEP
# # # FIRST Removing / Ignoring columns which contain 100% same values - a


# # # unique_counts_df['cols']
# # # unique_counts_df['unique_counts']==1

# # # type(unique_counts_df[unique_counts_df['unique_counts']==1]['cols'])
# # # pandas.core.series.Series




# cols_all_same = unique_counts_df[unique_counts_df['unique_counts']==1][

# # # cols_all_same

# # # 18        admin_fee
# # # 19        state_fee
# # # 22    clean_up_cost
```

```
# # # Name: cols, dtype: object

# sel_nonsame_cols = X_blight.columns[~X_blight.columns.isin(cols_all_same

# X_blight = X_blight[sel_nonsame_cols]


# # # X_blight.columns

# # # Index(['ticket_id', 'agency_name', 'inspector_name', 'violator_name
# # #        'violation_street_number', 'violation_street_name',
# # #        'mailing_address_str_number', 'mailing_address_str_name', 'ci
# # #        'state', 'zip_code', 'country', 'ticket_issued_date', 'hearin
# # #        'violation_code', 'violation_description', 'disposition', 'fi
# # #        'late_fee', 'discount_amount', 'judgment_amount'],
# # #       dtype='object')

# # # admin_fee, state_fee and clean_up_cost removed


# # # len(X_blight.columns)
# # # 21
# # # length of columns reduced to 21 from previous 24 and initial 33
```

In [17]:

```
##########################################################################

# Final Double commented whole cell - Jan 2021

# # # Back to # SIXTH STEP
# # # FIRST Removing / Ignoring columns which contain 99% unique values


# # # Again updating unique_counts_df based on updated X_blight
# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s


# # # unique_counts_df




# cols_all_unique = unique_counts_df[unique_counts_df['ratio']>0.99]['cols

# # # # cols_all_unique
# # # # 0    ticket_id
# # # # Name: cols, dtype: object
```

```
# sel_multi_value_cols = X_blight.columns[~X_blight.columns.isin(cols_all_

# X_blight = X_blight[sel_multi_value_cols]


# # # X_blight.columns

# # # Index(['agency_name', 'inspector_name', 'violator_name',
# # #        'violation_street_number', 'violation_street_name',
# # #        'mailing_address_str_number', 'mailing_address_str_name', 'ci
# # #        'state', 'zip_code', 'country', 'ticket_issued_date', 'hearin
# # #        'violation_code', 'violation_description', 'disposition', 'fi
# # #        'late_fee', 'discount_amount', 'judgment_amount'],
# # #       dtype='object')

# # # # ticket_id removed


# # # len(X_blight.columns)
# # # 20
# # # length of columns reduced to 20 from previous 21 and initial 33
```

In [18]:
```
#############################################################################

# Final Double commented whole cell - Jan 2021

# # STEP SEVEN
# # Removing non relevant columns - that seem non relevant


# # STEP EIGHT
# # Removing dependant/correlated columns - that seem non relevant


# # # Again updating unique_counts_df based on updated X_blight
# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s

# # unique_counts_df


# # #         cols        unique_counts        ratio
# # # 0        agency_name        5        0.000031
# # # 1        inspector_name        159        0.000994
#         # # 2        violator_name        84656        0.529497
#     # # 3        violation_street_number        18096        0.113185
```

```
#     # # 4        violation_street_name        1716        0.010733
#     # # 5        mailing_address_str_number        14090        0.08812
#     # # 6        mailing_address_str_name        28440        0.177883
# # # 7        city        4093        0.025600
# # # 8        state        59        0.000369
# # # 9        zip_code        4622        0.028909
# # # 10        country        5        0.000031
#     # # 11        ticket_issued_date        68097        0.425926
# # # 12        hearing_date        5970        0.037341
# # # 13        violation_code        189        0.001182
# # # 14        violation_description        207        0.001295
# # # 15        disposition        4        0.000025
# # # 16        fine_amount        40        0.000250
# # # 17        late_fee        37        0.000231
# # # 18        discount_amount        13        0.000081
# # # 19        judgment_amount        57        0.000357


# ################

# # Refer to the description of the fields also as mentioned in the proble

# # violation_street_number, violation_street_name, violation_zip_code - 
# # mailing_address_str_number, mailing_address_str_name, city, state, zi
# # violation_code, violation_description - Type of violation
# # disposition - Judgment and judgement type
# # fine_amount - Violation fine amount, excluding fees
# # late_fee - 10% fee assigned to responsible judgments
# # discount_amount - discount applied, if any
# # judgment_amount - Sum of all fines and fees


# ################


# # violator_name - name can be anything - and there are 84656 uniques na

# non_relevant_columns = ['violator_name']




# ####################




# # dependant_columns =
```

```
# ############
# # GETTING and REMOVING all DEPENDANT COLUMNS now
# ############


# #####################


# # violation_street_name, violation_street_number - we have violation_zip
# ### Checked below - violation_zip_code was removed as it had all null va


# # mailing_address_str_number, mailing_address_str_name, city, state, zip
# # violation_description



# #####################

# #####Added Later - had missed latitude longitude fields earlier - so we
# # considered for address

# #####################


# ## Checking all 4 violation_street_name, violation_street_number,mailing



# # X_blight[['violation_street_name', 'violation_street_number','mailing_

# ###########
# ### CHECKING CORRELATION

# # X_blight[['violation_street_name', 'violation_street_number']].corr()
# #         violation_street_number
# # violation_street_number        1.0


# # X_blight[['violation_street_name', 'mailing_address_str_number']].corr

# # X_blight[['violation_street_name', 'mailing_address_str_name']].corr(

# # X_blight[['violation_street_name', 'violation_street_number','mailing_
# #         violation_street_number        mailing_address_str_number
# # violation_street_number        1.000000        0.008428
```

```
# # mailing_address_str_number        0.008428        1.000000



# ######
# # Correlation is shown only between numeric columns - not between non nu
# # QUESTION - How to check correlation between non numeric columns?
# # Say 'violation_street_name', 'violation_street_number' - they are exac

# ###########

# # Another way of checking link between 'violation_street_name' and 'viol

# # unique_counts_df

# # No link between unique values of the two columns

# ###########



# # X_blight[['violation_street_name', 'violation_street_number','mailing_

# # X_blight[['violation_street_name', 'violation_street_number','mailing_



# ###########


# # X_blight['violation_street_name'].value_counts()

# # SEVEN MILE          2373
# # MCNICHOLS           2144
# # LIVERNOIS           1607
# # GRAND RIVER         1185
# # EVERGREEN           1067
# # WARREN               998
# # FENKELL              997
# # ASBURY PARK          900
# # WYOMING              872
# # GRATIOT              863
# # ARCHDALE             859
# # JOY RD               845


# # X_blight['violation_street_number'].value_counts()
```

```
# # 19300.0     103
# # 15700.0      85
# # 600.0        83
# # 2900.0       76
# # 6300.0       74
# # 20400.0      73
# # 7400.0       72
# # 18400.0      71
# # 8200.0       71
# # 18500.0      71


# # X_blight['mailing_address_str_number'].value_counts()
# # 213.0        1508
# # 1.0          1078
# # 4.0           726
# # 3.0           637
# # 3476.0        460
# # 11.0          456
# # 715.0         381
# # 28.0          371
# # 5.0           363
# # 21.0          341
# # 243.0         334
# # 127.0         317
# # 2.0           296
# # 4828.0        292


# # X_blight['mailing_address_str_name'].value_counts()

# # PO BOX                            5754
# # P.O. BOX                          4733
# # GRAND RIVER                        890
# # LIVERNOIS                          829
# # W. MCNICHOLS                       687
# # HARPER                             541
# # GREENFIELD                         509
# # W. SEVEN MILE                      498
# # GRATIOT                            498
# # P.O. Box                           492
# # P. O. BOX                          466
# # MACK                               461



# ####################
```

```
# #####Added Later - had missed latitude longitude fields earlier - so we
# # considered for address


# ###########


# # Checking violation_zip_code


# # X_blight['violation_street_number', 'violation_zip_code'].corr()
# # Checked - violation_zip_code was removed as it had all null values

# # blight_train_df['violation_zip_code'].value_counts()
# # Series([], Name: violation_zip_code, dtype: int64)

# # blight_train_df['violation_zip_code']
# # 0        NaN
# # 1        NaN
# # 5        NaN
# # 6        NaN
# # 7        NaN


# #####################

# #####Added Later - had missed latitude longitude fields earlier - so we
# # considered for address


# ###########

# # Checking zip_code


# # X_blight[['mailing_address_str_number', 'zip_code']].corr()


# #                              mailing_address_str_number
# # mailing_address_str_number        1.0

# # corr matrix does not contain zip_code - as zip_code is not pure numeri
# # - so we need to make those values NaN


# # zipsrs = X_blight['zip_code']
```

```
# # zipsrs.dtype
# # dtype('O')

# # zipsrs[0]
# # 60606

# # type(zipsrs[0])
# # int


# # zipsrs==60606


# # 'zip_code' is of type Object - we have to convert it to int

# # zipsrs.astype(int)
# # ValueError: invalid literal for int() with base 10: '92637-2854'

# # zipsrs.str.isnumeric()
# # np.isreal?

# # zipsrs[~np.isreal(zipsrs)]



# # np.bincount(np.isreal(zipsrs).astype(int))
# # array([     0, 159880])






# # zipsrs=='92637-2854'

# # zipsrs[zipsrs=='92637-2854']
# # 42394    92637-2854
# # Name: zip_code, dtype: object

# # zipsrs[zipsrs[zipsrs=='92637-2854'].index.values[0]]
# # type(zipsrs[zipsrs[zipsrs=='92637-2854'].index.values[0]])
# # str





# # zipsrs[pd.to_numeric(zipsrs, errors='coerce').isnull()]
```

```
# ### ***To convert a column to numeric - and also update non numeric valu


# # pd.to_numeric?
# # Signature: pd.to_numeric(arg, errors='raise', downcast=None)
# # Docstring:Convert argument to a numeric type.
# # Parameters - arg : list, tuple, 1-d array, or Series
# # errors : {'ignore', 'raise', 'coerce'}, default 'raise'
# #     - If 'raise', then invalid parsing will raise an exception
# #     - If 'coerce', then invalid parsing will be set as NaN
# #     - If 'ignore', then invalid parsing will return the input


# # pd.to_numeric(zipsrs)
# # ValueError: Unable to parse string "92637-2854" at position 28220

# # pd.to_numeric(zipsrs, errors='coerce')
# # 0              60606.0
# # 1              48208.0
# # 5          908041512.0
# # 6              48038.0

# # pd.to_numeric(zipsrs, errors='coerce').isnull()
# # 0          False
# # 1          False
# # 5          False
# # 6          False
# # 7          False
# # 8          False
# # 9          False



# # zipsrs[pd.to_numeric(zipsrs, errors='coerce').isnull()]

# # 42394     92637-2854
# # ...
# # 121000    48243-0671
# # 122078           NaN
# # ...
# # 186126       L5N 3H5
# # 191245       SE 770X
# # 193547      Deli-7DN
# # ...
# # 212865        V4W2R7
# # ...
# # 244227        N9A2H9
```

```
# # Name: zip_code, dtype: object



# # zipsrs[pd.to_numeric(zipsrs, errors='coerce').isnull()].count()
# # 74


# ######
# ###Now Updating these values to null / NaN


# ###X_blight['zip_code'] = pd.to_numeric(zipsrs, errors='coerce')
# X_blight['zip_code'] = pd.to_numeric(X_blight['zip_code'], errors='coerc



# # X_blight['zip_code']


# # X_blight[X_blight['zip_code'].isnull()]
# # 75 rows × 20 columns
# # So it means the non numeric values in zip_code have been updated to Na


# # X_blight[['mailing_address_str_number', 'zip_code']].corr()

# #             mailing_address_str_number        zip_code
# # mailing_address_str_number        1.000000        -0.002438
# # zip_code        -0.002438        1.000000


# # No correlation....







# ### Incorrect / incomplete other approaches

# ### X_blight['zip_code']=pd.to_numeric(zipsrs, errors='coerce').isnull()
# ### X_blight[X_blight['zip_code'].isnull()]
# ### X_blight.where(X_blight['zip_code'].isin(zipsrs[pd.to_numeric(zipsrs
# ### X_blight.where not working???
# # X_blight.where?

# ### X_blight['zip_code'].isin(zipsrs[pd.to_numeric(zipsrs, errors='coerc
```

```
# ### X_blight[X_blight['zip_code'].isin(zipsrs[pd.to_numeric(zipsrs, erro
# ### X_blight[X_blight['zip_code'].isin(zipsrs[pd.to_numeric(zipsrs, erro





# #####################


# # Checking - # violation_code, violation_description - Type of violatio
# # X_blight[['violation_code', 'violation_description']]
# ###X_blight[['violation_code', 'violation_description']].nunique()

# # X_blight['violation_code'].nunique()
# # 189

# # X_blight['violation_description'].nunique()
# # 207






# # X_blight['violation_code'].value_counts()
# # 9-1-36(a)                64414
# # 9-1-81(a)                23145
# # 22-2-88                  19073
# # 9-1-104                  16927
# # 22-2-88(b)                4879
# # 22-2-45                   4200
# # 9-1-105                   3619
# # 9-1-110(a)                3147


# # type(X_blight['violation_code'].value_counts())
# # pandas.core.series.Series


# # X_blight['violation_code'].value_counts().shape
# # (189,)

# # X_blight['violation_code'].value_counts().index
# # Index(['9-1-36(a)', '9-1-81(a)', '22-2-88', '9-1-104', '22-2-88(b)',
```

```
# # X_blight['violation_description'].value_counts()

# # Failure of owner to obtain certif...      64414
# # Failure to obtain certificate of ...      23145
# # Failure of owner to keep property...      19072
# # Excessive weeds or plant growth o...      16927
# # Allowing bulk solid waste to lie ...      4879
# # Violation of time limit for appro...      4200
# # Rodent harborage one-or two-famil...      3619
# # Inoperable motor vehicle(s) one- ...      3147


# # type(X_blight['violation_description'].value_counts())
# # pandas.core.series.Series



# # Removing X_blight['violation_description']



# #####################


# # Checking link between fine_amount and late_fee

# # X_blight[['fine_amount', 'late_fee']].corr()
# #          fine_amount        late_fee
# # fine_amount        1.000000        0.986787
# # late_fee        0.986787        1.000000


# # type(X_blight[['fine_amount', 'late_fee']].corr())
# # pandas.core.frame.DataFrame



# # X_blight[['fine_amount', 'late_fee']].corr().iloc[0, 1]
# # 0.98678695299925989

# # 99 % correlation

# # Removing late_fee


# #####################
```

```
# # Checking link between fine_amount and judgment_amount

# # X_blight[['fine_amount', 'judgment_amount']].corr()

# # X_blight[['fine_amount', 'judgment_amount']].corr().iloc[0, 1]
# # 0.99989002729923671


# # 99.99 % correlation

# # Removing judgment_amount


# # #####################

# # dependant_columns = ['violation_description', 'late_fee', 'judgment_an



# # #####################
# # LATER UPDATE

# # Idea from Aug 2020 implementation

# # Incorrectly removed late fee - thinking it is correlated with fine_amo
# # Though there was 99.99% correlation
# # BUT late fee is not there for all tickets.
# # So we have to convert it to a boolean

# # np.bincount(blight_df['late_fee']>0)
# # array([105884, 144422])


# ###Aug 2020
# ###BOOLEANIZED
# # finalDfForModelling['late_feeBool'] = finalDfForModelling['late_fee']>
# # booleanizedColumnsToDrop = ['late_fee']
# # finalDfForModelling.drop(booleanizedColumnsToDrop, axis=1, inplace=Tru
```

```
In [19]: ###########################################################################

         # Final Double commented whole cell - Jan 2021

         # non_relevant_columns = ['violator_name']
         # relevant_columns= X_blight.columns[~X_blight.columns.isin(non_relevant_c
         # X_blight = X_blight[relevant_columns]
```

```
# # Incorrectly removed late fee - thinking it is correlated with fine_amo
# # BUT late fee is not there for all tickets. # So we have to convert it

# # dependant_columns = ['violation_description', 'late_fee', 'judgment_am
# dependant_columns = ['violation_description', 'judgment_amount']
# non_dependant_columns= X_blight.columns[~X_blight.columns.isin(dependant
# X_blight = X_blight[non_dependant_columns]


# # X_blight.columns
# # Index(['agency_name', 'inspector_name', 'violation_street_number',
# #        'violation_street_name', 'mailing_address_str_number',
# #        'mailing_address_str_name', 'city', 'state', 'zip_code', 'count
# #        'ticket_issued_date', 'hearing_date', 'violation_code', 'dispos
# #        'fine_amount', 'discount_amount'],
# #       dtype='object')


# # # len(X_blight.columns)
# # # 16
# # # length of columns reduced to 16 from previous 20 and initial 33
```

In [20]:
```
##########################################################################

# Out of the 16 left, 8 are address columns - need to check which one or m
# appropriately represents the address

# 'violation_street_number',
# 'violation_street_name',
# 'mailing_address_str_number',
# 'mailing_address_str_name',
# 'city',
# 'state',
# 'zip_code',
# 'country',
```

In [21]:
```
##########################################################################

# Final Double commented whole cell - Jan 2021

# # STEP 9, 10, 11

# # 9. Handling ADDRESS categorical values? Microscopic or Macroscopic lev

# # 10. Handling DATE categorical values? - Dummify, or use bins?

# # 11. dummyfying other categorical values - but how many?
# #        - What if a column has 5000 distinct values? should we dummify?
```

```
# # # Again updating unique_counts_df based on updated X_blight
# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s

# unique_counts_df


# #         cols        unique_counts        ratio
# # 0        agency_name        5        0.000031
# # 1        inspector_name        159        0.000994
# # 2        violation_street_number        18096        0.113185
# # 3        violation_street_name        1716        0.010733
# # 4        mailing_address_str_number        14090        0.088129
# # 5        mailing_address_str_name        28440        0.177883
# # 6        city        4093        0.025600
# # 7        state        59        0.000369
# # 8        zip_code        3445        0.021547
# # 9        country        5        0.000031
# # 10        ticket_issued_date        68097        0.425926
# # 11        hearing_date        5970        0.037341
# # 12        violation_code        189        0.001182
# # 13        disposition        4        0.000025
# # 14        fine_amount        40        0.000250
# # 15        discount_amount        13        0.000081




# #############

# # QUESTION - ??? - # How to handle CATEGORICAL columns with MANY UNIQUE
# # The following columns have so many unique values - and are categorical
# # Most probably we cannot dummify them - there would be so many columns
# # So what can we do with them?
# # We cannot ignore them, what if ther is a link between say city and com


# # violation_street_number        18096
# # violation_street_name        1716
# # mailing_address_str_number        14090
# # mailing_address_str_name        28440
# # ticket_issued_date        68097


# # city        4093
```

```
# # hearing_date        5970
# # zip_code        3445


# #############

# # QUESTION - ??? - # How to handle ADDRESS?
# # Which columns to consider?
# # zip_code is unique so does it capture all? But there are so many zip_c
# # zip_code is microscopic level
# # Do we need to see on a Macroscopic level - like City or state or count
# # WHY are there more unique cities than Zipcodes???

# # violation_street_number        18096
# # violation_street_name        1716
# # mailing_address_str_number        14090
# # mailing_address_str_name        28440
# # city        4093
# # state        59
# # zip_code        3445
# # country        5


# #############

# # QUESTION - ??? - # How to handle DATE columns?
# # WHAT about DATES columns - it is also categorical - so many different
# # How to use a DATE column?
# # We cannot dummify a date field
# # So what can we do with them?
# # We cannot ignore them, what if ther is a link between say city and com

# # ticket_issued_date        68097
# # hearing_date        5970


# #############


# # SHOULD we CUT BINS for categorical and date columns with many unique v


# #############
```

```
In [22]: ###########################################################################


         # Final Double commented whole cell - Jan 2021
```

```
# # STEP 9, 10, 11

# # 9. Handling ADDRESS categorical values? Microscopic or Macroscopic lev


# #####################

# #####Added Later - had missed latitude longitude fields earlier - so we
# # considered for address



# #############

# # QUESTION - ??? - # How to handle ADDRESS?
# # Which columns to consider?
# # zip_code is unique so does it capture all? But there are so many zip_c
# # zip_code is microscopic level
# # Do we need to see on a Macroscopic level - like City or state or count
# # WHY are there more unique cities than Zipcodes???

# # Check Unique distribution by value_counts
# # Check Label distribution by Cutting bins or check Groupby or dummify?


# # Should we groupby column first say city and then check distribution of
# # OR Should we separate target variable - compliance / non-compliance -
# # column vlues in each? i.e. - X_compliant = , X_noncompliat=, X_complia



# # violation_street_number        18096
# # violation_street_name        1716
# # mailing_address_str_number        14090
# # mailing_address_str_name        28440
# # city        4093
# # state        59
# # zip_code        3445
# # country        5

# #############

# remove_address_columns = []

# #############
```

```
# #########
# ### country ###


# # Check Unique distribution by value_counts



# # X_blight['country'].value_counts()

# # USA      159869
# # Cana          6
# # Egyp          2
# # Aust          2
# # Germ          1
# # Name: country, dtype: int64


# # (X_blight['country'].value_counts()/len(X_blight['country'])) *100
# # USA      99.993120
# # Cana      0.003753
# # Egyp      0.001251
# # Aust      0.001251
# # Germ      0.000625
# # Name: country, dtype: float64

# #REMOVING COUNTRY - 99.99 in one country - so no information gain

# # remove_address_columns = ['country']

# #########

# ### state ###

# # (X_blight['state'].value_counts()/len(X_blight['state'])) *100
# # MI    89.851764
# # CA     2.394296
# # TX     1.220916
# # FL     1.050788
# # SC     0.666750
# # IL     0.560420
# # OH     0.399675
# # NY     0.334626
# # MN     0.279585
# # GA     0.253315
# # NV     0.249562
# # PA     0.232049
```

```
# # UT     0.205779
# # .......



# #########

# ### city ###
# (X_blight['city'].value_counts()/len(X_blight['city'])) *100

# # DETROIT              54.682262
# # SOUTHFIELD            5.497873
# # Detroit               3.986740
# # DEARBORN              1.506130
# # detroit               1.374156
# # FARMINGTON HILLS      0.918189
# # OAK PARK              0.898174
# # WARREN                0.772454
# # W. BLOOMFIELD         0.646110
# # DET                   0.631724
# # REDFORD               0.594821
# # TROY                  0.555417
# # LIVONIA               0.490368
# # WEST BLOOMFIELD       0.452840
# # Southfield            0.429697
# # ....


# # city_gpby = X_blight.groupby('city')
# # Grouping in blight_train_df and not X_blight as we do not have target
# ###city_gpby = blight_train_df.groupby('city')
# ###city_level_compliance = city_gpby.agg({'compliance':np.average})
# ###city_level_compliance.sort('compliance')


# # Grouping 'city' and 'compliance' together to check distribution of cor

# # gpby_citycompliance = blight_train_df.groupby(['city', 'compliance'])
# # gpby_citycompliance.size()



# #########

# ### violation_street_number ###


# # X_blight['violation_street_number']
```

```
# # 0          2900.0
# # 1          4311.0
# # 5          6478.0
# # 6          8027.0
# # 7          8228.0
# # 8          8228.0


# # (X_blight['violation_street_number'].value_counts()/len(X_blight['vio
# # 19300.0     0.064423
# # 15700.0     0.053165
# # 600.0       0.051914
# # 2900.0      0.047536
# # 6300.0      0.046285
# # 20400.0     0.045659

# # X_blight['violation_street_number'].value_counts().sort_values(ascendi

# # 19300.0    103
# # 15700.0     85
# # 600.0       83
# # 2900.0      76
# # 6300.0      74
# # ......
# # 9385.0       1
# # 15843.0      1
# # 5159.0       1
# # 11787.0      1
# # 11159.0      1
# # 16383.0      1


# # Seems ['violation_street_number'] is not contributing

# #REMOVING violation_street_number

# # remove_address_columns = ['country', 'violation_street_number']

# #########

# ### violation_street_name ###

# # X_blight['violation_street_name'].value_counts().sort_values(ascending

# # SEVEN MILE          2373
# # MCNICHOLS           2144
# # LIVERNOIS           1607
# # GRAND RIVER         1185
```

```
# # EVERGREEN           1067
# # WARREN               998
# # FENKELL              997
# # ASBURY PARK          900
# # WYOMING              872
# # GRATIOT              863
# # ARCHDALE             859
# # ...
# # STEARNS                1
# # SHEEHAN                1
# # FORDYCE                1
# # MIAMI                  1
# # DOREMUS                1
# # KINGSTON RD            1
# # LAFAYETTE PLAISA       1



# # Seems ['violation_street_name'] is not contributing

# #REMOVING violation_street_name

# # remove_address_columns = ['country', 'violation_street_number', 'viola


# #########

# ### mailing_address_str_number
# ### mailing_address_str_name


# # Similarly removing  - mailing_address_str_number, mailing_address_str_

# remove_address_columns = ['country', 'violation_street_number', 'violati



# #########


# # Left with 3 address columns

# # city        4093
# # state         59
# # zip_code        3445
```

```
# ##############
# ##############

# # Postponing further in depth analysis of address for later as short of
# # Evaluating considering major parameters - single parameter for address


# ##############
# ##############


# relevant_addressother_columns= X_blight.columns[~X_blight.columns.isin(r
# X_blight = X_blight[relevant_addressother_columns]


# # X_blight.columns
# # Index(['agency_name', 'inspector_name', 'city', 'state', 'zip_code',
# #        'ticket_issued_date', 'hearing_date', 'violation_code', 'dispos
# #        'fine_amount', 'discount_amount', 'compliance'],
# #       dtype='object')


# # # len(X_blight.columns)
# # # 12
# # # length of columns reduced to 12 from previous 16 and initial 33
```

In [23]:

```
##########################################################################

# Final Double commented whole cell - Jan 2021

# # Separating complaint and non compliant data - to analyze idividually

# X_blight_compliant = X_blight[X_blight['compliance']==1]
# X_blight_noncompliant = X_blight[X_blight['compliance']==0]


# # 2        city        4093        0.025600
# # 3        state        59        0.000369
# # 4        zip_code        3445        0.021547

# # X_blight_noncompliant['agency_name'].value_counts()


# # X_blight.groupby(['agency_name', ''])
# # gpby_agencycompliance = blight_train_df.groupby(['agency_name', 'compl
# # gpby_agencycompliance.size()
```

```
# # agency_name                                          compliance
# # Buildings, Safety Engineering & Env Department  0.0           90040
# #                                                 1.0            5823
# # Department of Public Works                      0.0           47727
# #                                                 1.0            4718
# # Detroit Police Department                       0.0            3876
# #                                                 1.0             588
# # Health Department                               0.0            6639
# #                                                 1.0             468
# # Neighborhood City Halls                         0.0               1
# # dtype: int64
```

In [24]:
```
#########################################################################


# # Again updating unique_counts_df based on updated X_blight
# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s

# unique_counts_df

#         cols        unique_counts        ratio
# 0        agency_name        5        0.000031
# 1        inspector_name        159        0.000994
# 2        city        4093        0.025600
# 3        state        59        0.000369
# 4        zip_code        3445        0.021547
# 5        ticket_issued_date        68097        0.425926
# 6        hearing_date        5970        0.037341
# 7        violation_code        189        0.001182
# 8        disposition        4        0.000025
# 9        fine_amount        40        0.000250
# 10        discount_amount        13        0.000081
# 11        compliance        2        0.000013




# X_blight.head()



#########################################################################

# checking value_counts() for relevant columns
```

```
# X_blight['agency_name'].value_counts().sort_values(ascending=False)
# Buildings, Safety Engineering & Env Department    95863
# Department of Public Works                        52445
# Health Department                                  7107
# Detroit Police Department                          4464
# Neighborhood City Halls                               1
# Name: agency_name, dtype: int64


# X_blight['inspector_name'].value_counts().sort_values(ascending=False)
# Morris, John           11604
# Samaan, Neil J          8720
# O'Neal, Claude          8075
# Steele, Jonathan        6962
# Devaney, John           6837
# Hayes, Billy J          6385
# Sloane, Bennie J        5624
# Sims, Martinzie         5526
# Zizi, Josue             5060
# Doetsch, James          4337
#......


# X_blight['city'].value_counts().sort_values(ascending=False)
# DETROIT                            87426
# SOUTHFIELD                          8790
# Detroit                             6374
# DEARBORN                            2408
# detroit                             2197
# FARMINGTON HILLS                    1468
# OAK PARK                            1436
#......


# X_blight['state'].value_counts().sort_values(ascending=False)
# MI    143655
# CA      3828
# TX      1952
# FL      1680
# SC      1066
# IL       896
# OH       639
# NY       535
#......


# X_blight['zip_code'].value_counts().sort_values(ascending=False)
# 48227.0    7316
```

```
# 48221.0    7213
# 48235.0    6842
# 48228.0    6026
# 48219.0    5875
# 48238.0    5435
# 48224.0    5421
# 48205.0    4764
# 48204.0    4357
#......


# X_blight['ticket_issued_date'].value_counts().sort_values(ascending=Fals
# 2007-12-21 09:00:00    60
# 2010-02-17 09:00:00    57
# 2008-01-22 09:00:00    56
# 2008-04-29 09:00:00    52
# 2006-03-08 09:00:00    52
# 2007-10-17 09:00:00    51
# 2007-10-25 11:00:00    50
# 2006-06-19 09:00:00    49
# 2006-01-12 09:00:00    48
# 2010-03-05 09:00:00    47
# 2006-04-03 09:00:00    47
# 2010-02-02 09:00:00    47
#......


# X_blight['hearing_date'].value_counts().sort_values(ascending=False)
# 2005-12-20 09:00:00    590
# 2005-12-22 10:30:00    546
# 2005-12-20 10:30:00    538
# 2005-12-21 10:30:00    529
# 2005-12-22 09:00:00    511
# 2005-12-21 09:00:00    403
# 2005-12-27 10:30:00    399
# ....


# X_blight['violation_code'].value_counts().sort_values(ascending=False)
# 9-1-36(a)               64414
# 9-1-81(a)               23145
# 22-2-88                 19073
# 9-1-104                 16927
# 22-2-88(b)               4879
# 22-2-45                  4200
# 9-1-105                  3619
# 9-1-110(a)               3147
# 9-1-43(a) - (Dwellin     3043
```

```
# 9-1-103(C)                2641
# 22-2-22                   2612
# ....


# X_blight['disposition'].value_counts().sort_values(ascending=False)
# Responsible by Default                  138340
# Responsible by Admission                 13701
# Responsible by Determination              7644
# Responsible (Fine Waived) by Deter         195
# Name: disposition, dtype: int64


# X_blight['fine_amount'].value_counts().sort_values(ascending=False)
# 250.0      86798
# 50.0       20415
# 100.0      15488
# 200.0      12710
# 500.0       6918
# 1000.0      4965
# 3500.0      3859
# 300.0       3768
# 2500.0      1545
# 25.0        1378
# ...



# X_blight['discount_amount'].value_counts().sort_values(ascending=False)
# 0.0      158700
# 25.0        605
# 5.0         167
# 10.0        155
# 20.0        135
# 50.0         43
# 3.0          19
# 30.0         17
# 100.0        16
# 350.0        15
# 250.0         6
# 13.0          1
# 40.0          1



# X_blight['lat'].value_counts().sort_values(ascending=False)
# 42.377249    387
# 42.341729    355
```

```
# 42.341730    329
# 42.410855    264
# 42.352331    248
# 42.407284    233
# 42.349328    116
# 42.410644    107
# ....

# X_blight['lon'].value_counts().sort_values(ascending=False)
# -83.238943    387
# -83.262245    355
# -83.262271    329
# -83.046409    264
# -83.252023    248
# -83.277150    233
# -83.256895    116
# -83.136195    107
# -83.257840     86
# ....

# X_blight['address'].value_counts().sort_values(ascending=False)
# 600 woodward ave, Detroit MI        52
# 16189 schaefer, Detroit MI          50
# 4471 parkinson, Detroit MI          42
# 935 louisiana, Detroit MI           35
# 9125 jefferson, Detroit MI          33
# ....



########################################################################

# Columns which we can dumify (try)

# 0        agency_name        5        0.000031
# 1        inspector_name        159        0.000994
# 3        state        59        0.000369
# 7        violation_code        189        0.001182
# 8        disposition        4        0.000025

# 9        fine_amount        40        0.000250
# 10        discount_amount        13        0.000081

#Should we consider the above two amount columns as categorical or continu



# Columns which have too many distinct values - probably cannot be dummifi
```

```
# 2        city        4093        0.025600
# 4        zip_code        3445        0.021547
# 5        ticket_issued_date        68097        0.425926
# 6        hearing_date        5970        0.037341
```

In [25]:
```
########################################################################


# Checking dtypes again - if we have to dummify also cut bins - then type

# X_blight.dtypes


# agency_name            object
# inspector_name         object
# city                   object
# state                  object
# zip_code              float64
# ticket_issued_date     object
# hearing_date           object
# violation_code         object
# disposition            object
# fine_amount           float64
# discount_amount       float64
# compliance            float64
# dtype: object
```

In [26]:
```
# Jan 2021 - extra columns
# 'ticket_issued_date', 'hearing_date'
# ['inspector_name', 'city', 'state', 'discount_amount']


# Aug 2020 - extra columns
# ['violation_street_name', 'lat', 'lon']
```

In [27]:
```
#########################################################################

# Final Double commented whole cell - Jan 2021

# # Re considering late_fee - there in Aug 2020 implementation but not in


# # Idea from Aug 2020 implementation

# # Incorrectly removed late fee - thinking it is correlated with fine_amo
# # Though there was 99% correlation
```

```
# # BUT late fee is not there for all tickets.

# # np.bincount(blight_df['late_fee']>0)
# # array([105884, 144422])



# # KEEP late_fee - booleanize



# # Aug 2020
# ###BOOLEANIZED

# # finalDfForModelling['late_feeBool'] = finalDfForModelling['late_fee']>
# # booleanizedColumnsToDrop = ['late_fee']
# # finalDfForModelling.drop(booleanizedColumnsToDrop, axis=1, inplace=Tru


# X_blight['late_feeBool'] = X_blight['late_fee']>0
# booleanizedColumnsToDrop = ['late_fee']
# X_blight.drop(booleanizedColumnsToDrop, axis=1, inplace=True)
```

In [28]:
```
#########################################################################

# Final Double commented whole cell - Jan 2021

# # Re considering discount_amount - not there in Aug 2020 implementation

# # (X_blight['discount_amount'].value_counts()/len(X_blight['discount_amo

# # 0.0      99.261946
# # 25.0      0.378409
# # 5.0       0.104453
# # 10.0      0.096948
# # 20.0      0.084438
# # 50.0      0.026895
# # 3.0       0.011884
# # 30.0      0.010633
# # 100.0     0.010008
# # 350.0     0.009382
# # 250.0     0.003753
# # 40.0      0.000625
# # 13.0      0.000625
# # Name: discount_amount, dtype: float64


# # As 'discount_amount' has distribution of 99.26 : 0.74 of majority : ot
```

```
# # target variable (compliance) is 92.75 : 7.25, so removing 'discount_an
# # np.bincount(y_blight) # array([148283,  11597])
# # np.bincount(y_blight)/len(y_blight)*100
# # array([ 92.74643483,   7.25356517])

# # REMOVE 'discount_amount'

# other_discount_amount_columns= X_blight.columns[~X_blight.columns.isin(
# X_blight = X_blight[other_discount_amount_columns]
```

In [29]:
```
##########################################################################


# Re considering inspector_name - not there in Aug 2020 implementation


# X_blight['inspector_name'].value_counts().sort_values(ascending=False)

# X_blight['inspector_name'].nunique()
# 159


# np.bincount(X_blight['inspector_name'].value_counts()>1000)
# array([111,  48])

# More than 100 have more than 1000 each - uniform distribution

# But #BUT WHAT IF THE inspector_name does affet the result??
#say caste/religion based ???
#How do we check

#QUESTION - How do we dummify this large number of unique values?
#173 unique inspector_name

# Undecided what to do + how to handle if keeping???
```

In [30]:
```
##########################################################################


# Final Double commented whole cell - Jan 2021

# # X_blight.columns


# # unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# # unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_coun
# # unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight
```

```
# # unique_counts_df


# # X_blight['zip_code'].nunique()
# # 3445

# # Removing zip_code as there are many and there is a uniform distributio
# #REMOVE zip_code

# # Removing lat as there are many and there is a uniform distribution
# #REMOVE lat

# # Removing lon as there are many and there is a uniform distribution
# #REMOVE lon


# # Removing address as there are many and there is a uniform distributio
# #REMOVE address

# other_columns_remove = ['zip_code', 'lat', 'lon', 'address']


# relevant_columns_left= X_blight.columns[~X_blight.columns.isin(other_co
# X_blight = X_blight[relevant_columns_left]
```

In [31]:
```
##########################################################################

# Handling remaining columns as follows

# agency_name - all - dummify
# inspector_name - top 10 + others - dummify
# city - top 10 + others - dummify
# state - top 10 + others - dummify
# violation_code - top 10 + others - dummify
# ticket_issued_date - date bins??
# hearing_date - date bins??
# fine_amount - top 10 + others - dummify
# disposition - all - dummify
```

In [32]:
```
##########################################################################


# Final Double commented whole cell - Jan 2021

# # How to select top 10 based on value counts - doing for 'inspector_nam

# # X_blight['inspector_name'].value_counts().sort_values(ascending=False
```

```
# # type(X_blight['inspector_name'].value_counts().sort_values(ascending=F
# # pandas.core.series.Series

# X_blight['inspector_name'].value_counts().nlargest(10)

# # Morris, John        11604
# # Samaan, Neil J       8720
# # O'Neal, Claude       8075
# # Steele, Jonathan     6962
# # Devaney, John        6837
# # Hayes, Billy J       6385
# # Sloane, Bennie J     5624
# # Sims, Martinzie      5526
# # Zizi, Josue          5060
# # Doetsch, James       4337
# # Name: inspector_name, dtype: int64


# # type(X_blight['inspector_name'].value_counts().nlargest(10))
# # pandas.core.series.Series


# # X_blight['inspector_name'].value_counts().nlargest(10).sum()
# # 69130

# # Top 10 do not even constitute half.


# top10srs = X_blight['inspector_name'].value_counts().nlargest(10)
# # top10srs.index.values

# X_blight['inspector_name'] = [inspector if inspector in top10srs.index.v
# # X_blight['inspector_name']
# # 0         Sims, Martinzie
# # 1         Sims, Martinzie
# # 2         Sims, Martinzie
# # 3                   other
# # 4            Morris, John
# # 5                   other
# # 6         Sims, Martinzie
# # 7          Samaan, Neil J
# # 8                   other
# # 9                   other
# # 10                  other
# # 11                  other
```

```
# ####################
# # Doing for all other columns - selecting top 10 and marking others as o
# # city - top 10 + others - dummify
# # state - top 10 + others - dummify
# # violation_code - top 10 + others - dummify
# # fine_amount - top 10 + others - dummify

# top10srs = X_blight['city'].value_counts().nlargest(10)
# X_blight['city'] = [city if city in top10srs.index.values else 'other_ci







# top10srs = X_blight['state'].value_counts().nlargest(10)
# X_blight['state'] = [state if state in top10srs.index.values else 'other







# top10srs = X_blight['violation_code'].value_counts().nlargest(10)
# X_blight['violation_code'] = [violation_code if violation_code in top10s







# top10srs = X_blight['fine_amount'].value_counts().nlargest(10)
# X_blight['fine_amount'] = [fine_amount if fine_amount in top10srs.index.


# # X_blight.head(10)

# #         agency_name        inspector_name        city        state
# # 0        Buildings, Safety Engineering & Env Department        Sims, M
# # 1        Buildings, Safety Engineering & Env Department        Sims, M
# # 2        Buildings, Safety Engineering & Env Department        Sims, M
# # 3        Department of Public Works        other        other        I
# # 4        Buildings, Safety Engineering & Env Department        Morris,
# # 5        Buildings, Safety Engineering & Env Department        other
```

In [33]:
```
# # Again updating unique_counts_df based on updated X_blight
# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s

# unique_counts_df

#         cols        unique_counts        ratio
# 0        agency_name        5        0.000031
# 1        inspector_name        11        0.000069
# 2        city        11        0.000069
# 3        state        11        0.000069
# 4        ticket_issued_date        68097        0.425926
# 5        hearing_date        5970        0.037341
# 6        violation_code        11        0.000069
# 7        disposition        4        0.000025
# 8        fine_amount        11        0.000069
# 9        compliance        2        0.000013
# 10        late_feeBool        2        0.000013
```

In [34]:
```
##########################################################################

# Analyzing DATE Columns - trying to cut bins


# X_blight['ticket_issued_date'].value_counts().nlargest(10)

# tktdt = X_blight['ticket_issued_date']
# tktdt.sort_values()

# type(tktdt[2])
# str
# pandas.tslib.Timestamp
# It as string initially - but after converting X_blight['ticket_issued_da


# tktdt.sort_index()

# if we sort by index, and try to cut bins,
# it will not be logical cutting - it will be random.
## random items will be grouped together and not in any sequence
```

In [35]:
```
##########################################################################

# Analyzing DATE Columns - trying to cut bins
```

```
# pd.cut?

# Signature: pd.cut(x, bins, right=True, labels=None, retbins=False, preci
# Docstring: Return indices of half-open bins to which each value of `x` b

# Parameters
# ----------
# x : array-like
#     Input array to be binned. It has to be 1-dimensional.
# bins : int or sequence of scalars
#     If `bins` is an int, it defines the number of equal-width bins in th
#     range of `x`. However, in this case, the range of `x` is extended
#     by .1% on each side to include the min or max values of `x`. If
#     `bins` is a sequence it defines the bin edges allowing for
#     non-uniform bin width. No extension of the range of `x` is done in
#     this case.
# right : bool, optional
#     Indicates whether the bins include the rightmost edge or not. If
#     right == True (the default), then the bins [1,2,3,4] indicate
#     (1,2], (2,3], (3,4].
# labels : array or boolean, default None
#     Used as labels for the resulting bins. Must be of the same length as
#     the resulting bins. If False, return only integer indicators of the
#     bins.
```

```
In [36]: ##########################################################################

         # Analyzing DATE Columns - trying to cut bins



         # n=10


         # ['ticket_date' + str(n) for n in range(10)]
         # pd.cut(X_blight['ticket_issued_date'], n, labels=['ticket_date' + str(n)
         # pd.cut(X_blight['ticket_issued_date'], n, labels=['ticket_date' + str(n)

         # type(X_blight['ticket_issued_date'][1])
         # str


         # X_blight['ticket_issued_date'] = X_blight['ticket_issued_date'].astype(n

         # X_blight['ticket_issued_date'].astype(np.datetime64)[1]
         # Timestamp('2006-05-24 09:00:00')
```

```
# pd.cut(X_blight['ticket_issued_date'], n, labels=['ticket_date' + str(n)
# TypeError: cannot astype a datetimelike from [datetime64[ns]] to [dateti


# pd.to_datetime(X_blight['ticket_issued_date'])
# type(pd.to_datetime(X_blight['ticket_issued_date'])[1])
# pandas.tslib.Timestamp


# X_blight['ticket_issued_date'] = pd.to_datetime(X_blight['ticket_issued_

# pd.cut(X_blight['ticket_issued_date'], n, labels=['ticket_date' + str(n)
# pd.cut(X_blight['ticket_issued_date'], n)
# TypeError: unsupported operand type(s) for +: 'Timestamp' and 'float'



## Can only CUT CONTINUOUS variable / values into BINS - cannot cut string

## Other approach is to cut into 10 bins by index - but it will not be log
## random items will be grouped together and not in any sequence
```

In [37]:

```
################################################################################


# Final Double commented whole cell - Jan 2021

# # Analyzing DATE Columns - trying to cut bins

# # As shortage of time fro assignment submission, leaving out date column

# date_columns_remove = ['ticket_issued_date', 'hearing_date']

# nondate_columns_left= X_blight.columns[~X_blight.columns.isin(date_colum
# X_blight = X_blight[nondate_columns_left]

# # X_blight.columns

# # Index(['agency_name', 'inspector_name', 'city', 'state', 'violation_co
# #        'disposition', 'fine_amount', 'compliance', 'late_feeBool'],
# #       dtype='object')



# # # Again updating unique_counts_df based on updated X_blight
# # unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# # unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_coun
```

```
        # # unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight

        # # unique_counts_df


        # #         cols        unique_counts        ratio
        # # 0        agency_name        5        0.000031
        # # 1        inspector_name        11        0.000069
        # # 2        city        11        0.000069
        # # 3        state        11        0.000069
        # # 4        violation_code        11        0.000069
        # # 5        disposition        4        0.000025
        # # 6        fine_amount        11        0.000069
        # # 7        compliance        2        0.000013
        # # 8        late_feeBool        2        0.000013

In [38]: ######################################################################
        # Whole cell commented

        # # Aug 2020


        # ###DUMMIFIED
        # ###Correct term for this process / step - "Encoding Categorical Values i







        # agency_dummies = pd.get_dummies(finalDfForModelling['agency_name'])

        # #agency_dummies
        # #         Buildings, Safety Engineering & Env Department         Departm
        # #0         1         0         0         0         0
        # #1         1         0         0         0         0
        # #2         1         0         0         0         0

        # disposition_dummies = pd.get_dummies(finalDfForModelling['disposition'])
        # #         Not responsible by City Dismissal         Not responsible by D
        # #0         0         0         0         0         0         0         1
        # #1         0         0         0         0         0         0         1
        # #2         0         0         0         0         0         0         1

        # #pd.concat([finalDfForModelling, agency_dummies], axis=1)
        # #pd.concat([finalDfForModelling, disposition_dummies], axis=1)
        # dummifiedColumnsToDrop = ['agency_name', 'disposition']
```

```
# ##CHECK - do we drop main columns first then add/concat new hot encoded
# #or we can drop later also after adding/concatanating hot encoded colum

# #tried dropping later and all columns were dropped...




# ####################################
# ###UPDATE - after proper datapreprocessing completed and 80%+ result ach
# ###- Keeping 'zip_code' and 'violation_street_name' and 'violation_code

# #######################
# #Final update
# #One Hot encoding NOT WORKING on this machine for the above three - mach
# #One Hot encoding NOT WORKING for features with lot of unique values -
# #1791, 5643 and 235 unique values for each
# #"The kernel appears to have died. It will restart automatically."
# #So have to remove these hot encodings for now



# #violation_street_name_dummies = pd.get_dummies(finalDfForModelling['vio
# #zip_code_dummies = pd.get_dummies(finalDfForModelling['zip_code'])
# #violation_code_dummies = pd.get_dummies(finalDfForModelling['violation_




# ###ALSO DUMMIFY -'violation_street_name', 'zip_code', 'violation_code'
# #dummifiedColumnsToDrop = ['agency_name', 'disposition', 'violation_stre




# finalDfForModelling.drop(dummifiedColumnsToDrop, axis=1, inplace=True)
```

In [39]:
```
##########################################################################


# Final Double commented whole cell - Jan 2021


# # ###DUMMIFYING - JAN 2021
# # ###Correct term for this process / step - "Encoding Categorical Values
```

```
# agency_dummies = pd.get_dummies(X_blight['agency_name'])

# # agency_dummies
# #         Buildings, Safety Engineering & Env Department        Departme
# # 0        1        0        0        0        0
# # 1        1        0        0        0        0


# inspector_dummies = pd.get_dummies(X_blight['inspector_name'])
# city_dummies = pd.get_dummies(X_blight['city'])
# state_dummies = pd.get_dummies(X_blight['state'])
# violation_code_dummies = pd.get_dummies(X_blight['violation_code'])
# fine_amount_dummies = pd.get_dummies(X_blight['fine_amount'])

# # fine_amount


# disposition_dummies = pd.get_dummies(X_blight['disposition'])
# # disposition_dummies
# #         Responsible (Fine Waived) by Deter        Responsible by Admis
# # 0        0        0        1        0
# # 1        0        0        1        0



# dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'stat
# X_blight.drop(dummifiedColumnsToDrop, axis=1, inplace=True)
```

In [40]:
```
#########################################################################

# Final Double commented whole cell - Jan 2021


# # X_blight.columns
# # Index(['fine_amount', 'compliance', 'late_feeBool'], dtype='object')


# # # agency_dummies = pd.get_dummies(X_blight['agency_name'])
# # # inspector_dummies = pd.get_dummies(X_blight['inspector_name'])
# # # city_dummies = pd.get_dummies(X_blight['city'])
# # # state_dummies = pd.get_dummies(X_blight['state'])
# # # violation_code_dummies = pd.get_dummies(X_blight['violation_code'])
# # # disposition_dummies = pd.get_dummies(X_blight['disposition'])
```

```
# # X_blight.join?
# # Signature: X_blight.join(other, on=None, how='left', lsuffix='', rsuf
# # Docstring: Join columns with other DataFrame either on index or on a k
# #     objects by index at once by passing a list.




# X_blight = X_blight.join(agency_dummies)
# X_blight = X_blight.join(inspector_dummies)
# X_blight = X_blight.join(city_dummies)
# X_blight = X_blight.join(state_dummies)
# X_blight = X_blight.join(violation_code_dummies)
# X_blight = X_blight.join(fine_amount_dummies)
# X_blight = X_blight.join(disposition_dummies)


# # X_blight.columns


# # 'compliance',
# #                                                   'late_feeBool',
# #         'Buildings, Safety Engineering & Env Department',
# #                                 'Department of Public Works',
# #                                  'Detroit Police Department',
# #                                          'Health Department',
# #                                   'Neighborhood City Halls',
# #                                             'Devaney, John',
# #                                            'Doetsch, James',
# #                                            'Hayes, Billy J',
# #                                             'Morris, John',
# #                                            'O'Neal, Claude',
# #                                            'Samaan, Neil J',
# #                                           'Sims, Martinzie',
# #                                          'Sloane, Bennie J',
# #                                          'Steele, Jonathan',
# #                                              'Zizi, Josue',
# #                                          'other_inspector',
# #                                                'DEARBORN',
# #                                                     'DET',
# #                                                 'DETROIT',
# #                                                 'Detroit',
# #                                        'FARMINGTON HILLS',
# #                                                'OAK PARK',
```

```
        # #                                                   'SOUTHFIELD',
        # #                                                'W. BLOOMFIELD',
        # #                                                       'WARREN',
        # #                                                      'detroit',
        # #                                                   'other_city',
        # #                                                           'CA',
        # #                                                           'FL',
        # #                                                           'GA',
        # #                                                           'IL',
        # #                                                           'MI',
        # #                                                           'MN',
        # #                                                           'NY',
        # #                                                           'OH',
        # #                                                           'SC',
        # #                                                           'TX',
        # #                                                  'other_state',
        # #                                                      '22-2-45',
        # #                                                      '22-2-88',
        # #                                                   '22-2-88(b)',
        # #                                                   '9-1-103(C)',
        # #                                                     '9-1-104',
        # #                                                     '9-1-105',
        # #                                                   '9-1-110(a)',
        # #                                                    '9-1-36(a)',
        # #                                         '9-1-43(a) - (Dwellin',
        # #                                                    '9-1-81(a)',
        # #                                              'other_violation',
        # #                                                         25.0,
        # #                                                         50.0,
        # #                                                        100.0,
        # #                                                        200.0,
        # #                                                        250.0,
        # #                                                        300.0,
        # #                                                        500.0,
        # #                                                       1000.0,
        # #                                                       2500.0,
        # #                                                       3500.0,
        # #                                                 'other_amount',
        # #                         'Responsible (Fine Waived) by Deter',
        # #                                   'Responsible by Admission',
        # #                                     'Responsible by Default',
        # #                               'Responsible by Determination'],
        # #       dtype='object')
```

```
In [41]: # X_blight.isnull().sum().sum()
         # 0
```

```
In [42]: ###########################################################################
         # Whole cell commented
```

```
# # Aug 2020



# #pd.concat([finalDfForModelling, agency_dummies], axis=1)
# #pd.concat([finalDfForModelling, disposition_dummies], axis=1)

# #pd.concat did not work, hot encoded columns were not added to main DF -

# finalDfForModelling = finalDfForModelling.join(agency_dummies)
# finalDfForModelling = finalDfForModelling.join(disposition_dummies)



# ###UPDATE - after proper datapreprocessing completed and 80%+ result ach
# ###- Keeping 'zip_code' and 'violation_street_name' and 'violation_code

# finalDfForModelling = finalDfForModelling.join(violation_street_name_dum
# finalDfForModelling = finalDfForModelling.join(zip_code_dummies)
# finalDfForModelling = finalDfForModelling.join(violation_code_dummies)


# ######################
# #Final update
# #One Hot encoding NOT WORKING on this machine for the above three - mach
# #One Hot encoding NOT WORKING for features with lot of unique values -
# #1791, 5643 and 235 unique values for each
# #"The kernel appears to have died. It will restart automatically."
# #So have to remove these hot encodings for now



# #INCORRECTLY DUMMIFIED ALL types of disposition_dummies - need to remove
# #remove - 'Not responsible by City Dismissal', 'Not responsible by Deter
# #       'Not responsible by Dismissal', 'PENDING JUDGMENT',

# #ALSO the name - 'Buildings, Safety Engineering & Env Department' does n
```

In [43]:
```
############################################################################
# Whole cell commented



# Aug 2020
```

```
# finalDfForModelling.columns
# Index([

# 'fine_amount',

# 'compliance',


# 'lat', 'lon',


# 'late_feeBool',


#        'Buildings, Safety Engineering & Env Department',
#        'Department of Public Works', 'Detroit Police Department',
#        'Health Department', 'Neighborhood City Halls',
#        'Responsible (Fine Waived) by Deter', 'Responsible by Admission',
#        'Responsible by Default', 'Responsible by Determination'],



#       dtype='object')
```

```
In [44]: ###########################################################################
         # Whole cell commented


         # # Aug 2020




         # ###After column/feature selection now Filling missing data
         # ###This step also below in later cell "Other useful techniques"-from fi

         # finalDfForModelling.isnull().sum()


         # #fine_amount                                          0
         # #lat                                                  2
         # #lon                                                  2
         # #late_feeBool                                         0
         # #Buildings, Safety Engineering & Env Department       0
         # #Department of Public Works                           0
         # #Detroit Police Department                            0
         # #Health Department                                    0
```

```
# #Neighborhood City Halls                          0
# #Responsible (Fine Waived) by Deter               0
# #Responsible by Admission                         0
# #Responsible by Default                           0
# #Responsible by Determination                     0
# #dtype: int64


# #have to do all the steps for test data as well.
# #test_allDetails defined 2 cells later

# finalDfForModelling['lat'].fillna(test_allDetails.lat.mean(),inplace = T
# finalDfForModelling['lon'].fillna(test_allDetails.lon.mean(),inplace = T

# finalDfForModelling.isnull().sum()
# #ALL zero now
```

In [45]:
```
#########################################################################

# Final Double commented whole cell - Jan 2021


# # ####Moved removing target column compliance to end - as there is no wa
# # ###with initial parent DF traindata. We have removed all unique identi


# y_blight=X_blight['compliance']

# #dropping compliance from X_blight now

# targetColumnToRemove = ['compliance']
# X_blight.drop(targetColumnToRemove, axis=1,  inplace=True)

# # # y_blight.head(10)
# # # 0    0.0
# # # 1    0.0
# # # 2    0.0
# # # 3    0.0
# # # 4    0.0
# # # 5    1.0
# # # 6    0.0
# # # 7    0.0
# # # 8    0.0
# # # 9    0.0
# # # Name: compliance, dtype: float64


# # X_blight.shape
```

```
        # # (159880, 65)

        # # y_blight.shape
        # # (159880,)

In [46]: # Final Double commented whole cell - Jan 2021

         # from sklearn.model_selection import train_test_split

         # X_train, X_test, y_train, y_test = train_test_split(X_blight, y_blight)

In [47]: # X_train.shape
         # (119910, 65)

In [48]: ##########################################################################
         ################ FROM AUG 2020

         ################ Also used in JAN 2021


         # Final Double commented whole cell - Jan 2021



         # ###USING DecisionTreeClassifier first

         # #from sklearn.tree import DecisionTreeClassifier
         # #cf = DecisionTreeClassifier(max_depth=3).fit(X_train, y_train)

         # #In first (error free) run, used default settings.
         # #In second (error free) run, used max_depth=3 and AUC score increased

         # #(ERROR) First run - error for non numeric values

         # #(ERROR) Second run - error for NaN values
         # #ValueError: Input contains NaN, infinity or a value too large for dtype


         # #Subsequent runs - after fixing for Non numeric and Nan - no error

         # ################
         # ###USING RandomForestClassifier - 2nd classifier/model
         # # from sklearn.ensemble import RandomForestClassifier

         # # cf = RandomForestClassifier().fit(X_train, y_train)


         # ################
```

```
# ###USING Support Vector Classifier
# ###It does not converge -
# ###from sklearn.svm import SVC
# ###cf = SVC().fit(X_train, y_train)



# ###############
# ###Using KNN - DONT USE - SYSTEM HANGS
# #from sklearn.neighbors import KNeighborsClassifier
# #cf=KNeighborsClassifier().fit(X_train, y_train)


# ###############
# ###USING Naive Bias


# #from sklearn.naive_bayes import GaussianNB
# #cf = GaussianNB().fit(X_train, y_train)

# ###############
# ###Using LogisticRegression - regularization - C value
# from sklearn.linear_model import LogisticRegression
# cf = LogisticRegression(C=0.01).fit(X_train, y_train)




# ##############################################

# ###USING GridSearchCV
# #from sklearn.model_selection import GridSearchCV
# #from sklearn.metrics import roc_auc_score




# ###############

# ###USING GridSearchCV - with Logistics regression
# ###Using a grid of all 3 solvers, penalty and c_values made the system h





# #cf = LogisticRegression()
```

```
# ###solvers = ['newton-cg', 'lbfgs', 'liblinear']
# ###penalty = ['l2']
# #c_values = [100, 10, 1.0, 0.1, 0.01]

# ###grid = dict(solver=solvers,penalty=penalty,C=c_values)
# #grid = dict(C=c_values)
# ###Using a grid of all 3 solvers, penalty and c_values made the system h



# ###############

# ###USING GridSearchCV - with RandomForestClassifier
# ###Using a grid of both n_estimators and max_features made the system ha
# ###Even Using a grid of only using n_estimators parameter made the syste

# #cf = RandomForestClassifier()
# #n_estimators = [10, 100, 1000]
# #max_features = ['sqrt', 'log2']
# #grid = dict(n_estimators=n_estimators,max_features=max_features)
# #grid = dict(n_estimators=n_estimators)


# ###############

# #grid_clf_auc = GridSearchCV(estimator=cf, param_grid=grid, n_jobs=-1, s

# ###Added following two lines for error
# ##c, r = y_train.shape
# ##y_train = y_train.values.reshape(c,)

# #grid_clf_auc.fit(X_train, y_train)

# #y_decision_fn_scores_auc = grid_clf_auc.decision_function(X_test)
# #print('Test set AUC: ', roc_auc_score(y_test, y_decision_fn_scores_auc)
# #print('Grid best parameter (max. AUC): ', grid_clf_auc.best_params_)
# #print('Grid best score (AUC): ', grid_clf_auc.best_score_)



# #Test set AUC:  0.75457318348
# #Grid best parameter (max. AUC):  {'C': 0.01}
# #Grid best score (AUC):  0.757423175482


# ###############
```

```
# ################################


# #y_predict=cf.predict(X_test)


# #Why following warning with RandomForestClassifier
# #/opt/conda/lib/python3.6/site-packages/ipykernel/__main__.py:19: DataCo
```

In [49]:
```
################################################################################
############### FROM AUG 2020

# import sklearn
# print(sklearn.__version__)

# 0.18.1
```

In [50]:
```
################################################################################
############### FROM AUG 2020


#help(KNeighborsClassifier)



#help(LogisticRegression)
#penalty : str, 'l1' or 'l2', default: 'l2'
#C : float, default: 1.0
#solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag'}, default: 'liblinear
```

In [51]:
```
# ###############################################################################
# ############### FROM AUG 2020

# Commented whole cell again in Jan 2021



# #help(cf.score)
# ###X_train.shape #(8697, 12)


# print(cf.score(X_train, y_train))

# print(cf.score(X_test, y_test))

# from sklearn.metrics import accuracy_score, roc_auc_score
```

```
# #print(accuracy_score(y_predict, y_test)) #Same as cf.score

# #y_decision_fn_scores_auc = cf.decision_function(X_test)
# #roc_auc_score(y_test, y_decision_fn_scores_auc)
# #AttributeError: 'DecisionTreeClassifier' object has no attribute 'decis

# #We CAN and have to use predict_proba SCORES in place of decision_functi
# #that dont have decision_function

# predictions = cf.predict_proba(X_test)
# ###predictions.shape #(39970, 2)
# print(roc_auc_score(y_test, predictions[:,1]))


# ###################################################################
# #BEFORE proper data preprocessing

# #In first (error free) run, used default settings
# #####0.99307814193978816
# #####0.91063297473104832
# #####0.66236418792945417


# #In second (error free) run, used max_depth=3 - Train score decreased, t
# #####0.934809440414
# #####0.9369777333
# #####0.75928850230859413


# #Random forest, default settings
# #####0.984388291218
# #####0.93347510633
# #####0.758054299143


# #Random forest, max_depth=3
# #####0.934801100826
# #####0.937027770828
# #####0.762768369932


# #Random forest, max_depth=5
# #####0.93478442165
# #####0.937127845884
# #####0.772813244017

# #Random forest, max_depth=7
# #####0.934809440414
```

```
# #####0.937152864648
# #####0.775815212662


# #Random forest, max_depth=10

# #####0.936077057793
# #####0.937177883413
# #####0.790481267468


# #Random forest, max_depth=15

# #####0.936077057793
# #####0.937177883413
# #####0.790481267468


# #Random forest, max_depth=50 - performance decreased
# #####0.984688516387
# #####0.934425819365
# #####0.767225921937


# ###USING Support Vector Classifier
# ###It takes a lot of time


# ###USING KNeighborsClassifier - default settings (n_neighbors=5)
# #####0.932349261946
# #####0.923392544408
# #####0.62580266691


# ###USING KNeighborsClassifier - n_neighbors=10 - no change



# ###USING GaussianNB - default settings

# #########0.934175631724
# #####0.933775331499
# #####0.583239015819

# ###USING LogisticRegression - default settings - #BEST Results SO FAR
# #####0.934083896256
# #####0.933299974981
# ####0.759953020432
```

```
# #BEST SO FAR



# ###USING LogisticRegression - C=100

# #####0.927328829956
# #####0.926770077558
# #####0.758023351427


# ###USING LogisticRegression - C=0.1

# #####0.934083896256
# #####0.933299974981
# #####0.759947871996


# ###USING LogisticRegression - C=0.01


# #####0.934409140188
# #####0.932349261946
# #####0.760788448739



# ####################################################################
# #After proper data preprocessing

# #Decision Tree - max_depth=3
# #In second (error free) run, used max_depth=3 - Train score decreased, t

# #0.938378784088
# #0.937953465099
# #0.782374230923


# #Random forest, default settings

# #0.982828788258
# #0.931998999249
# #0.759044331425


# #Random forest, max_depth=3
```

```
# #Random forest, max_depth=5


# #Random forest, max_depth=7



# #Random forest, max_depth=10



# #Random forest, max_depth=15

# #0.945584188141
# #0.937953465099
# #0.809202349911



# #Random forest, max_depth=50 - performance decreased



# ###USING Support Vector Classifier
# ###It takes a lot of time


# ###USING KNeighborsClassifier - default settings (n_neighbors=5)



# ###USING KNeighborsClassifier - n_neighbors=10 - no change



# ###USING GaussianNB - default settings

# #0.856125427404
# #0.855941956467
# #0.783358194115



# ###USING LogisticRegression - default settings - #BEST Results SO FAR

# #0.93849553832
# #0.93752814611
```

```
        # #0.779832652676



        # ###USING LogisticRegression - C=100

        # #0.93849553832
        # #0.93752814611
        # #0.779833014193


        # ###USING LogisticRegression - C=0.1
        # #0.938437161204
        # #0.937478108581
        # #0.779917451583


        # ###USING LogisticRegression - C=0.01


        # #0.936327245434
        # #0.935376532399
        # #0.781913572246

In [52]: # Final Double commented whole cell - Jan 2021


        # print(cf.score(X_train, y_train))

        # print(cf.score(X_test, y_test))

        # from sklearn.metrics import accuracy_score, roc_auc_score
        # predictions = cf.predict_proba(X_test)
        # print(roc_auc_score(y_test, predictions[:,1]))

In [53]: #Decision Tree - Default

        # 0.940980735552
        # 0.937953465099
        # 0.769302666672

        #Decision Tree - max_depth=3
        # 0.93777833375
        # 0.939754816112
        # 0.765655954364

        #Random forest, default settings
```

```
# 0.940805604203
# 0.937803352514
# 0.78295373124


#Random forest, max_depth=3

# 0.928054374114
# 0.929847385539
# 0.778038246927

#Random forest, max_depth=5

#Random forest, max_depth=7

#Random forest, max_depth=10

#Random forest, max_depth=15

#Random forest, max_depth=50 - performance decreased



###USING Support Vector Classifier
###It takes a lot of time


###USING KNeighborsClassifier - default settings (n_neighbors=5)



###USING KNeighborsClassifier - n_neighbors=10 - no change



###USING GaussianNB - default settings

# 0.791360186807
# 0.794120590443
# 0.745034741671

###USING LogisticRegression - default settings - #BEST Results SO FAR

###USING LogisticRegression - C=100
# 0.937553164874
# 0.939454590943
# 0.792000252842
```

```
###USING LogisticRegression - C=0.1

# 0.937561504462
# 0.939454590943
# 0.792087233616


###USING LogisticRegression - C=0.01

# 0.935560003336
# 0.937853390043
# 0.795380693703
```

In [54]:
```
# return a series of length 61001 with the data being the probability that

# Example:

# ticket_id
#    284932    0.531842
#    285362    0.401958
#    285361    0.105928
#    285338    0.018572
#              ...
#    376499    0.208567
#    376500    0.818759
#    369851    0.018528
#    Name: compliance, dtype: float32
```

In [64]:
```
# ######## TRAIN.CSV


# # FINAL
# # Double commented

# import pandas as pd
# import numpy as np
# blight_df = pd.read_csv('train.csv', encoding='cp1252')
# blight_train_df = blight_df.copy()
# blight_train_df = blight_train_df[~blight_train_df['compliance'].isnull
# X_blight = blight_train_df.iloc[:, :]
# y_blight = blight_train_df.iloc[:, -1]
# not_in_test_cols = ['payment_amount', 'payment_date', 'payment_status',

# sel_cols = X_blight.columns[~X_blight.columns.isin(not_in_test_cols)]

# X_blight = X_blight[sel_cols]
# columns_with_null = X_blight.columns[X_blight.count()/X_blight.shape[0]<
```

```
# sel_nonnull_cols = X_blight.columns[~X_blight.columns.isin(columns_with_

# X_blight = X_blight[sel_nonnull_cols]
# addresses = pd.read_csv("readonly/addresses.csv",encoding="cp1252")
# latlons = pd.read_csv("readonly/latlons.csv",encoding="cp1252")
# # For assignment submission - comment above two lines and uncomment belo
# # addresses = pd.read_csv("addresses.csv",encoding="cp1252")
# # latlons = pd.read_csv("latlons.csv",encoding="cp1252")
# # FileNotFoundError: File b'addresses.csv' does not exist
# X_blight = X_blight.merge(addresses, left_on='ticket_id', right_on='tick
# X_blight = X_blight.merge(latlons, left_on='address', right_on='address


# unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
# unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_counts
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight.s

# cols_all_same = unique_counts_df[unique_counts_df['unique_counts']==1][
# sel_nonsame_cols = X_blight.columns[~X_blight.columns.isin(cols_all_same
# X_blight = X_blight[sel_nonsame_cols]

# cols_all_unique = unique_counts_df[unique_counts_df['ratio']>0.99]['cols
# sel_multi_value_cols = X_blight.columns[~X_blight.columns.isin(cols_all_
# X_blight = X_blight[sel_multi_value_cols]

# X_blight['zip_code'] = pd.to_numeric(X_blight['zip_code'], errors='coerc

# non_relevant_columns = ['violator_name']
# relevant_columns= X_blight.columns[~X_blight.columns.isin(non_relevant_c
# X_blight = X_blight[relevant_columns]

# dependant_columns = ['violation_description', 'judgment_amount']
# non_dependant_columns= X_blight.columns[~X_blight.columns.isin(dependant
# X_blight = X_blight[non_dependant_columns]


# remove_address_columns = ['country', 'violation_street_number', 'violati
# relevant_addressother_columns= X_blight.columns[~X_blight.columns.isin(r
# X_blight = X_blight[relevant_addressother_columns]

# X_blight['late_feeBool'] = X_blight['late_fee']>0
# booleanizedColumnsToDrop = ['late_fee']
# X_blight.drop(booleanizedColumnsToDrop, axis=1, inplace=True)

# other_discount_amount_columns= X_blight.columns[~X_blight.columns.isin(
# X_blight = X_blight[other_discount_amount_columns]

# other_columns_remove = ['zip_code', 'address']
```

```
# # other_columns_remove = ['zip_code', 'lat', 'lon', 'address']

# X_blight['lat'].fillna(X_blight.lat.mean(),inplace = True)
# X_blight['lon'].fillna(X_blight.lon.mean(),inplace = True)


# relevant_columns_left= X_blight.columns[~X_blight.columns.isin(other_col
# X_blight = X_blight[relevant_columns_left]




# # Problem was - test agencies were only 3 - so test columns were less.
# # Solution - we can select only those number of columns from train dummi
# # BUT - what is there are different dummy columns?? - Will there be a pr
# # HENCE - Commenting dummy columns for the time being.

# n = 3
# # top10srs = X_blight['agency_name'].value_counts().nlargest(n)
# # X_blight['agency_name'] = [agency if agency in top10srs.index.values e
# # top10srs = X_blight['inspector_name'].value_counts().nlargest(n)
# # X_blight['inspector_name'] = [inspector if inspector in top10srs.index
# # top10srs = X_blight['city'].value_counts().nlargest(n)
# # X_blight['city'] = [city if city in top10srs.index.values else 'other_
# # top10srs = X_blight['state'].value_counts().nlargest(n)
# # X_blight['state'] = [state if state in top10srs.index.values else 'oth
# top10srs = X_blight['violation_code'].value_counts().nlargest(n)
# X_blight['violation_code'] = [violation_code if violation_code in top10s
# # top10srs = X_blight['fine_amount'].value_counts().nlargest(n)
# # X_blight['fine_amount'] = [fine_amount if fine_amount in top10srs.inde
# # top10srs = X_blight['disposition'].value_counts().nlargest(n)
# # X_blight['disposition'] = [disposition if disposition in top10srs.inde

# # agency_dummies = pd.get_dummies(X_blight['agency_name'])
# # inspector_dummies = pd.get_dummies(X_blight['inspector_name'])
# # city_dummies = pd.get_dummies(X_blight['city'])
# # state_dummies = pd.get_dummies(X_blight['state'])
# violation_code_dummies = pd.get_dummies(X_blight['violation_code'])
# # fine_amount_dummies = pd.get_dummies(X_blight['fine_amount'])
# # disposition_dummies = pd.get_dummies(X_blight['disposition'])

# # dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'st

# dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'stat
# X_blight.drop(dummifiedColumnsToDrop, axis=1, inplace=True)

# # X_blight = X_blight.join(agency_dummies)
# # X_blight = X_blight.join(inspector_dummies)
# # X_blight = X_blight.join(city_dummies)
```

```
# # X_blight = X_blight.join(state_dummies)
# X_blight = X_blight.join(violation_code_dummies)
# # X_blight = X_blight.join(fine_amount_dummies)
# # X_blight = X_blight.join(disposition_dummies)

# date_columns_remove = ['ticket_issued_date', 'hearing_date']
# nondate_columns_left= X_blight.columns[~X_blight.columns.isin(date_colu
# X_blight = X_blight[nondate_columns_left]



# y_blight=X_blight['compliance']
# targetColumnToRemove = ['compliance']
# X_blight.drop(targetColumnToRemove, axis=1,  inplace=True)



# # #AS a sanity check, removing all rows with any null value in final DF
# # #X_blight.dropna(axis=0, how='any', inplace=True)

# X_blight.fillna(X_blight.mean())
# # X_blight = X_blight.reset_index()

# from sklearn.model_selection import train_test_split
# X_train, X_test, y_train, y_test = train_test_split(X_blight, y_blight)

# from sklearn.linear_model import LogisticRegression
# cf = LogisticRegression(C=0.01).fit(X_train, y_train)

# #print(cf.score(X_train, y_train))
# #print(cf.score(X_test, y_test))

# from sklearn.metrics import accuracy_score, roc_auc_score
# predictions = cf.predict_proba(X_test)
# #print(roc_auc_score(y_test, predictions[:,1]))
# final_auc_score = roc_auc_score(y_test, predictions[:,1])
# #final_auc_score
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2717: Dtype
  interactivity=interactivity, compiler=compiler, result=result)
```

In [65]:
```
# final_auc_score
# # X_blight.columns
# # X_blight
```

Out[65]: 0.75636386405569178

In [71]:
```
# # TEST.CSV
```

```
# # FINAL
# # Double commented



# blight_df = pd.read_csv('readonly/test.csv', encoding='cp1252')
# #blight_df = pd.read_csv('test.csv', encoding='cp1252')
# blight_test_df = blight_df.copy()
# X_blight_test = blight_test_df.iloc[:, :]

# columns_with_null = X_blight_test.columns[X_blight_test.count()/X_blight

# sel_nonnull_cols = X_blight_test.columns[~X_blight_test.columns.isin(col

# X_blight_test = X_blight_test[sel_nonnull_cols]

# X_blight_test = X_blight_test.merge(addresses, left_on='ticket_id', righ
# X_blight_test = X_blight_test.merge(latlons, left_on='address', right_on


# unique_counts = [X_blight_test[col].nunique() for col in X_blight_test.c
# unique_counts_df = pd.DataFrame({'cols':X_blight_test.columns, 'unique_c
# unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight_t

# cols_all_same = unique_counts_df[unique_counts_df['unique_counts']==1][
# sel_nonsame_cols = X_blight_test.columns[~X_blight_test.columns.isin(col
# X_blight_test = X_blight_test[sel_nonsame_cols]

# # cols_all_unique = unique_counts_df[unique_counts_df['ratio']>0.99]['co
# # sel_multi_value_cols = X_blight_test.columns[~X_blight_test.columns.is
# # X_blight_test = X_blight_test[sel_multi_value_cols]
# X_blight_test.set_index('ticket_id', inplace=True)

# X_blight_test['zip_code'] = pd.to_numeric(X_blight_test['zip_code'], err

# non_relevant_columns = ['violator_name']
# relevant_columns= X_blight_test.columns[~X_blight_test.columns.isin(non_
# X_blight_test = X_blight_test[relevant_columns]

# dependant_columns = ['violation_description', 'judgment_amount']
# non_dependant_columns= X_blight_test.columns[~X_blight_test.columns.isin
# X_blight_test = X_blight_test[non_dependant_columns]


# remove_address_columns = ['country', 'violation_street_number', 'violati
# relevant_addressother_columns= X_blight_test.columns[~X_blight_test.colu
# X_blight_test = X_blight_test[relevant_addressother_columns]
```

```
# X_blight_test['late_feeBool'] = X_blight_test['late_fee']>0
# booleanizedColumnsToDrop = ['late_fee']
# X_blight_test.drop(booleanizedColumnsToDrop, axis=1, inplace=True)

# other_discount_amount_columns= X_blight_test.columns[~X_blight_test.colu
# X_blight_test = X_blight_test[other_discount_amount_columns]


# X_blight_test['lat'].fillna(X_blight_test.lat.mean(),inplace = True)
# X_blight_test['lon'].fillna(X_blight_test.lon.mean(),inplace = True)


# other_columns_remove = ['zip_code', 'address']

# # other_columns_remove = ['zip_code', 'lat', 'lon', 'address']
# relevant_columns_left= X_blight_test.columns[~X_blight_test.columns.isin
# X_blight_test = X_blight_test[relevant_columns_left]


# # Problem was - test agencies were only 3 - so test columns were less.
# # Solution - we can select only those number of columns from train dummi
# # BUT - what is there are different dummy columns?? - Will there be a pr
# # HENCE - Commenting dummy columns for the time being.

# n = 3
# # top10srs = X_blight_test['agency_name'].value_counts().nlargest(n)
# # X_blight_test['agency_name'] = [agency if agency in top10srs.index.val
# # top10srs = X_blight_test['inspector_name'].value_counts().nlargest(n)
# # X_blight_test['inspector_name'] = [inspector if inspector in top10srs.
# # top10srs = X_blight_test['city'].value_counts().nlargest(n)
# # X_blight_test['city'] = [city if city in top10srs.index.values else 'c
# # top10srs = X_blight_test['state'].value_counts().nlargest(n)
# # X_blight_test['state'] = [state if state in top10srs.index.values else
# top10srs = X_blight_test['violation_code'].value_counts().nlargest(n)
# X_blight_test['violation_code'] = [violation_code if violation_code in t
# # top10srs = X_blight_test['fine_amount'].value_counts().nlargest(n)
# # X_blight_test['fine_amount'] = [fine_amount if fine_amount in top10srs
# # top10srs = X_blight_test['disposition'].value_counts().nlargest(n)
# # X_blight_test['disposition'] = [disposition if disposition in top10srs

# # agency_dummies = pd.get_dummies(X_blight_test['agency_name'])
# # inspector_dummies = pd.get_dummies(X_blight_test['inspector_name'])
# # city_dummies = pd.get_dummies(X_blight_test['city'])
# # state_dummies = pd.get_dummies(X_blight_test['state'])
# # violation_code_dummies = pd.get_dummies(X_blight_test['violation_code
# # fine_amount_dummies = pd.get_dummies(X_blight_test['fine_amount'])
# # disposition_dummies = pd.get_dummies(X_blight_test['disposition'])
```

```
# # dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'st

# dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'stat
# X_blight_test.drop(dummifiedColumnsToDrop, axis=1, inplace=True)

# # X_blight_test = X_blight_test.join(agency_dummies)
# # X_blight_test = X_blight_test.join(inspector_dummies)
# # X_blight_test = X_blight_test.join(city_dummies)
# # X_blight_test = X_blight_test.join(state_dummies)
# X_blight_test = X_blight_test.join(violation_code_dummies)
# # X_blight_test = X_blight_test.join(fine_amount_dummies)
# # X_blight_test = X_blight_test.join(disposition_dummies)

# date_columns_remove = ['ticket_issued_date', 'hearing_date']
# nondate_columns_left= X_blight_test.columns[~X_blight_test.columns.isin
# X_blight_test = X_blight_test[nondate_columns_left]

# # MANUALLY dropping ['violation_zip_code', 'clean_up_cost']
# # as shortage of time before assignment submission
# manually_removing_columns = ['violation_zip_code', 'clean_up_cost']
# X_blight_test.drop(manually_removing_columns, axis=1, inplace=True)
# X_blight_test.fillna(X_blight_test.mean(), inplace=True)
# X_blight_test.fillna(0.0, inplace=True)
# # cf.predict_proba(X_blight_test)[:, 1]
# pd.Series(cf.predict_proba(X_blight_test)[:, 1], index=X_blight_test.inc
```

Out[71]:
```
ticket_id
284932    0.044847
285362    0.030615
285361    0.047058
285338    0.044979
285346    0.047162
285345    0.044979
285347    0.048290
285342    0.325708
285530    0.030724
284989    0.038969
285344    0.048281
285343    0.030678
285340    0.030684
285341    0.048291
289828    0.038996
289830    0.043925
289829    0.043925
292133    0.030684
292134    0.048291
285349    0.047163
```

```
285348    0.044980
284991    0.038969
285532    0.039044
286073    0.039044
285406    0.038918
285001    0.038975
285006    0.030667
365862    0.320194
285405    0.030616
287857    0.014772
            ...
376276    0.038915
376218    0.043845
376368    0.043897
376369    0.044952
376225    0.043837
376222    0.038883
376362    0.043797
376363    0.044848
376228    0.043870
376265    0.043854
376286    0.319722
376320    0.043860
376314    0.043822
376327    0.320100
376435    0.268609
376434    0.048131
376459    0.047082
376478    0.004315
376473    0.043838
376484    0.042849
376482    0.038873
376480    0.038873
376479    0.038873
376481    0.038873
376483    0.043787
376496    0.030637
376497    0.030637
376499    0.047022
376500    0.047022
369851    0.342093
dtype: float64
```

In [72]:
```
# X_blight_test
# cols_all_unique
```

Out[72]:
```
           fine_amount        lat        lon late_feeBool  22-2-88  9-1-36
ticket_id
```

284932	200.0	42.407581	-82.986642	True	0.0
285362	1000.0	42.426239	-83.238259	True	0.0
285361	100.0	42.426239	-83.238259	True	0.0
285338	200.0	42.309661	-83.122426	True	0.0
285346	100.0	42.308830	-83.121116	True	0.0
285345	200.0	42.308830	-83.121116	True	0.0
285347	50.0	42.308830	-83.121116	True	0.0
285342	200.0	42.313314	-83.108636	False	0.0
285530	1000.0	42.261245	-83.160878	True	0.0
284989	500.0	42.342537	-83.148025	True	0.0
285344	50.0	42.314038	-83.109165	True	0.0
285343	1000.0	42.314038	-83.109165	True	0.0
285340	1000.0	42.308638	-83.124173	True	0.0
285341	50.0	42.308638	-83.124173	True	0.0
289828	500.0	42.308638	-83.124173	True	0.0
289830	250.0	42.308638	-83.124173	True	0.0
289829	250.0	42.308638	-83.124173	True	0.0
292133	1000.0	42.308638	-83.124173	True	0.0
292134	50.0	42.308638	-83.124173	True	0.0
285349	100.0	42.308349	-83.123961	True	0.0
285348	200.0	42.308349	-83.123961	True	0.0
284991	500.0	42.342551	-83.147582	True	0.0
285532	500.0	42.262999	-83.154649	True	0.0
286073	500.0	42.262999	-83.154649	True	0.0
285406	500.0	42.403742	-83.173347	True	0.0
285001	500.0	42.324182	-83.084811	True	0.0
285006	1000.0	42.324182	-83.084811	True	0.0
365862	250.0	42.324182	-83.084811	False	0.0
285405	1000.0	42.427711	-83.250202	True	0.0
287857	2500.0	42.427711	-83.250202	True	0.0
...	...	...	...	...	...
376276	500.0	42.414291	-83.208107	True	0.0
376218	250.0	42.409008	-83.236295	True	0.0
376368	250.0	42.335629	-83.124043	True	0.0
376369	200.0	42.335629	-83.124043	True	0.0
376225	250.0	42.419294	-83.248783	True	0.0
376222	500.0	42.403075	-82.980001	True	0.0
376362	250.0	42.431516	-83.114224	True	0.0
376363	200.0	42.431516	-83.114224	True	0.0
376228	250.0	42.345385	-83.040250	True	0.0
376265	250.0	42.399954	-83.234485	True	0.0
376286	250.0	42.425716	-83.150609	False	0.0
376320	250.0	42.361969	-83.073230	True	0.0
376314	250.0	42.372530	-82.945200	True	0.0
376327	250.0	42.335418	-83.052811	False	0.0
376435	750.0	42.335131	-83.042013	False	0.0
376434	50.0	42.431923	-83.030224	True	0.0
376459	100.0	42.361012	-83.021640	True	0.0

376478	5000.0	42.361012	-83.021640	True	0.0
376473	250.0	42.368823	-83.001506	True	0.0
376484	300.0	42.344692	-83.079369	True	0.0
376482	500.0	42.431325	-83.069009	True	0.0
376480	500.0	42.431325	-83.069009	True	0.0
376479	500.0	42.431325	-83.069009	True	0.0
376481	500.0	42.431325	-83.069009	True	0.0
376483	250.0	42.431325	-83.069009	True	0.0
376496	1000.0	42.376675	-83.140869	True	0.0
376497	1000.0	42.376675	-83.140869	True	0.0
376499	100.0	42.409430	-82.992015	True	0.0
376500	100.0	42.409525	-82.991747	True	0.0
369851	50.0	42.349152	-83.120740	False	0.0

	9-1-81(a)	other_violation
ticket_id		
284932	0.0	0.0
285362	0.0	0.0
285361	0.0	0.0
285338	0.0	0.0
285346	0.0	0.0
285345	0.0	0.0
285347	0.0	0.0
285342	0.0	0.0
285530	0.0	0.0
284989	0.0	0.0
285344	0.0	0.0
285343	0.0	0.0
285340	0.0	0.0
285341	0.0	0.0
289828	0.0	0.0
289830	0.0	0.0
289829	0.0	0.0
292133	0.0	0.0
292134	0.0	0.0
285349	0.0	0.0
285348	0.0	0.0
284991	0.0	0.0
285532	0.0	0.0
286073	0.0	0.0
285406	0.0	0.0
285001	0.0	0.0
285006	0.0	0.0
365862	0.0	0.0
285405	0.0	0.0
287857	0.0	0.0
...	...	...
376276	0.0	0.0

```
376218          0.0          0.0
376368          0.0          0.0
376369          0.0          0.0
376225          0.0          0.0
376222          0.0          0.0
376362          0.0          0.0
376363          0.0          0.0
376228          0.0          0.0
376265          0.0          0.0
376286          0.0          0.0
376320          0.0          0.0
376314          0.0          0.0
376327          0.0          0.0
376435          0.0          0.0
376434          0.0          0.0
376459          0.0          0.0
376478          0.0          0.0
376473          0.0          0.0
376484          0.0          0.0
376482          0.0          0.0
376480          0.0          0.0
376479          0.0          0.0
376481          0.0          0.0
376483          0.0          0.0
376496          0.0          0.0
376497          0.0          0.0
376499          0.0          0.0
376500          0.0          0.0
369851          0.0          0.0

[61001 rows x 8 columns]
```

In [58]:
```
# X_blight_test
# X_blight_test.fillna(inplace=True)
```

In [59]:
```
###########################################################################

#####


# Column mismatch between Train and Test run
# Checking and Resolving


#####
```

```
# cf.predict_proba?

# X_blight_test.shape
# (61001, 69)


# ValueError: could not convert string to float: '48208'


# X_blight_test.columns[X_blight_test.dtypes!=np.uint8]
# Index(['violation_zip_code', 'clean_up_cost', 'late_feeBool'], dtype='ob
# X_blight_test[['violation_zip_code', 'clean_up_cost', 'late_feeBool']]


# X_blight_test['late_feeBool'].value_counts()


# X_blight_test.isnull().sum()

# violation_zip_code                                   36977
# clean_up_cost                                            0
# late_feeBool                                             0




# MANUALLY dropping ['violation_zip_code', 'clean_up_cost']
# as shortage of time before assignment submission


# X_blight_test.shape
# (61001, 67)


# X_blight.shape
# (159880, 65)

# X_blight.columns
# X_blight_test.columns

# X_blight.columns.values in X_blight_test.columns.values


# type(X_blight.columns.values)
# numpy.ndarray
```

```
# np.setdiff1d(X_blight_test.columns.values, X_blight.columns.values)


# import numpy as np
# array1 = np.array(['0', '10'])
# print("Array1: ",array1)
# array2 = ['10', '30', '40', '50', '70']
# print("Array2: ",array2)
# print("Unique values in array1 that are not in array2:")
# print(np.setdiff1d(array1, array2))

# np.setdiff1d?


# # X_blight.columns

# Index([                                'late_feeBool',
#         'Buildings, Safety Engineering & Env Department',
#                            'Department of Public Works',
#                                   'Health Department',
#                                        'other_agency',
#                                        'Morris, John',
#                                     'O'Neal, Claude',
#                                     'Samaan, Neil J',
#                                    'other_inspector',
#                                           'DETROIT',
#                                           'Detroit',
#                                        'SOUTHFIELD',
#                                        'other_city',
#                                               'CA',
#                                               'MI',
#                                               'TX',
#                                       'other_state',
#                                          '22-2-88',
#                                        '9-1-36(a)',
#                                        '9-1-81(a)',
#                                   'other_violation',
#                                               50.0,
#                                              100.0,
#                                              250.0,
#                                      'other_amount',
#                           'Responsible by Admission',
#                             'Responsible by Default',
#                        'Responsible by Determination',
#                                 'other_disposition'],

# X_blight_test.columns
```

```
# Index([                                      'late_feeBool',
#        'Buildings, Safety Engineering & Env Department',
#                                'Department of Public Works',
#                                 'Detroit Police Department',
#                                            'Lusk, Gertrina',
#                                           'Snyder, Derrell',
#                                               'Zizi, Josue',
#                                           'other_inspector',
#                                                   'DETROIT',
#                                                   'Detroit',
#                                                'SOUTHFIELD',
#                                                'other_city',
#                                                        'CA',
#                                                        'MI',
#                                                        'TX',
#                                               'other_state',
#                                                '22-2-88(b)',
#                                                   '9-1-104',
#                                                 '9-1-36(a)',
#                                           'other_violation',
#                                                        50.0,
#                                                       100.0,
#                                                       250.0,
#                                              'other_amount',
#                                  'Responsible by Admission',
#                                    'Responsible by Default',
#                              'Responsible by Determination',
#                                         'other_disposition'],


# Problem was - test agencies were only 3 - so test columns were less.


# type(agency_dummies)
# pandas.core.frame.DataFrame




# agency_dummies.shape
# (61001, 3)

# agency_dummies.columns
# Index(['Buildings, Safety Engineering & Env Department',
#        'Department of Public Works', 'Detroit Police Department'],
#       dtype='object')
```

```
# Problem was - test agencies were only 3 - so test columns were less.

# Solution - we can select only those number of columns from train dummies
# BUT - what is there are different dummy columns??
# Will there be a problem?


# HENCE
# Commenting dummy columns for the time being.
```

In [75]:
```
import pandas as pd
import numpy as np

def blight_model():

    #TRAIN
    blight_df = pd.read_csv('train.csv', encoding='cp1252')
    blight_train_df = blight_df.copy()
    blight_train_df = blight_train_df[~blight_train_df['compliance'].isnul
    X_blight = blight_train_df.iloc[:, :]
    y_blight = blight_train_df.iloc[:, -1]
    not_in_test_cols = ['payment_amount', 'payment_date', 'payment_status'

    sel_cols = X_blight.columns[~X_blight.columns.isin(not_in_test_cols)]

    X_blight = X_blight[sel_cols]
    columns_with_null = X_blight.columns[X_blight.count()/X_blight.shape[(

    sel_nonnull_cols = X_blight.columns[~X_blight.columns.isin(columns_wit

    X_blight = X_blight[sel_nonnull_cols]
    #addresses = pd.read_csv("readonly/addresses.csv",encoding="cp1252")
    #latlons = pd.read_csv("readonly/latlons.csv",encoding="cp1252")
    # For assignment submission - comment above two lines and uncomment be
    addresses = pd.read_csv("addresses.csv",encoding="cp1252")
    latlons = pd.read_csv("latlons.csv",encoding="cp1252")
    # Submission error - FileNotFoundError: File b'addresses.csv' does not
    X_blight = X_blight.merge(addresses, left_on='ticket_id', right_on='ti
    X_blight = X_blight.merge(latlons, left_on='address', right_on='addres


    unique_counts = [X_blight[col].nunique() for col in X_blight.columns]
    unique_counts_df = pd.DataFrame({'cols':X_blight.columns, 'unique_cour
    unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight

    cols_all_same = unique_counts_df[unique_counts_df['unique_counts']==1]
```

```
sel_nonsame_cols = X_blight.columns[~X_blight.columns.isin(cols_all_sa
X_blight = X_blight[sel_nonsame_cols]

cols_all_unique = unique_counts_df[unique_counts_df['ratio']>0.99]['co
sel_multi_value_cols = X_blight.columns[~X_blight.columns.isin(cols_al
X_blight = X_blight[sel_multi_value_cols]

X_blight['zip_code'] = pd.to_numeric(X_blight['zip_code'], errors='coe

non_relevant_columns = ['violator_name']
relevant_columns= X_blight.columns[~X_blight.columns.isin(non_relevant
X_blight = X_blight[relevant_columns]

dependant_columns = ['violation_description', 'judgment_amount']
non_dependant_columns= X_blight.columns[~X_blight.columns.isin(dependa
X_blight = X_blight[non_dependant_columns]


remove_address_columns = ['country', 'violation_street_number', 'viola
relevant_addressother_columns= X_blight.columns[~X_blight.columns.isin
X_blight = X_blight[relevant_addressother_columns]

X_blight['late_feeBool'] = X_blight['late_fee']>0
booleanizedColumnsToDrop = ['late_fee']
X_blight.drop(booleanizedColumnsToDrop, axis=1, inplace=True)

other_discount_amount_columns= X_blight.columns[~X_blight.columns.isin
X_blight = X_blight[other_discount_amount_columns]

# other_columns_remove = ['zip_code', 'lat', 'lon', 'address']

X_blight['lat'].fillna(X_blight.lat.mean(),inplace = True)
X_blight['lon'].fillna(X_blight.lon.mean(),inplace = True)


other_columns_remove = ['zip_code', 'address']
relevant_columns_left= X_blight.columns[~X_blight.columns.isin(other_c
X_blight = X_blight[relevant_columns_left]



# Problem was - test agencies were only 3 - so test columns were less.
# Solution - we can select only those number of columns from train dur
# BUT - what is there are different dummy columns?? - Will there be a
# HENCE - Commenting dummy columns for the time being.

n = 3
# top10srs = X_blight['agency_name'].value_counts().nlargest(n)
```

```
# X_blight['agency_name'] = [agency if agency in top10srs.index.values
# top10srs = X_blight['inspector_name'].value_counts().nlargest(n)
# X_blight['inspector_name'] = [inspector if inspector in top10srs.inc
# top10srs = X_blight['city'].value_counts().nlargest(n)
# X_blight['city'] = [city if city in top10srs.index.values else 'othe
# top10srs = X_blight['state'].value_counts().nlargest(n)
# X_blight['state'] = [state if state in top10srs.index.values else 'c
top10srs = X_blight['violation_code'].value_counts().nlargest(n)
X_blight['violation_code'] = [violation_code if violation_code in top1
# top10srs = X_blight['fine_amount'].value_counts().nlargest(n)
# X_blight['fine_amount'] = [fine_amount if fine_amount in top10srs.in
# top10srs = X_blight['disposition'].value_counts().nlargest(n)
# X_blight['disposition'] = [disposition if disposition in top10srs.in

# agency_dummies = pd.get_dummies(X_blight['agency_name'])
# inspector_dummies = pd.get_dummies(X_blight['inspector_name'])
# city_dummies = pd.get_dummies(X_blight['city'])
# state_dummies = pd.get_dummies(X_blight['state'])
violation_code_dummies = pd.get_dummies(X_blight['violation_code'])
# fine_amount_dummies = pd.get_dummies(X_blight['fine_amount'])
# disposition_dummies = pd.get_dummies(X_blight['disposition'])

# dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city',

dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'st
X_blight.drop(dummifiedColumnsToDrop, axis=1, inplace=True)

# X_blight = X_blight.join(agency_dummies)
# X_blight = X_blight.join(inspector_dummies)
# X_blight = X_blight.join(city_dummies)
# X_blight = X_blight.join(state_dummies)
X_blight = X_blight.join(violation_code_dummies)
# X_blight = X_blight.join(fine_amount_dummies)
# X_blight = X_blight.join(disposition_dummies)

date_columns_remove = ['ticket_issued_date', 'hearing_date']
nondate_columns_left= X_blight.columns[~X_blight.columns.isin(date_col
X_blight = X_blight[nondate_columns_left]



y_blight=X_blight['compliance']
targetColumnToRemove = ['compliance']
X_blight.drop(targetColumnToRemove, axis=1,  inplace=True)



# AS a sanity check, removing all rows with any null value in final DE
```

```
# X_blight.dropna(axis=0, how='any', inplace=True)


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_blight, y_blight

from sklearn.linear_model import LogisticRegression
cf = LogisticRegression(C=0.01).fit(X_train, y_train)

#print(cf.score(X_train, y_train))
#print(cf.score(X_test, y_test))

from sklearn.metrics import accuracy_score, roc_auc_score
predictions = cf.predict_proba(X_test)
#print(roc_auc_score(y_test, predictions[:,1]))
final_auc_score = roc_auc_score(y_test, predictions[:,1])
#final_auc_score



#TEST









#blight_df = pd.read_csv('readonly/test.csv', encoding='cp1252')
blight_df = pd.read_csv('test.csv', encoding='cp1252')
blight_test_df = blight_df.copy()
X_blight_test = blight_test_df.iloc[:, :]

columns_with_null = X_blight_test.columns[X_blight_test.count()/X_blig

sel_nonnull_cols = X_blight_test.columns[~X_blight_test.columns.isin(c

X_blight_test = X_blight_test[sel_nonnull_cols]

X_blight_test = X_blight_test.merge(addresses, left_on='ticket_id', ri
X_blight_test = X_blight_test.merge(latlons, left_on='address', right_


unique_counts = [X_blight_test[col].nunique() for col in X_blight_test
unique_counts_df = pd.DataFrame({'cols':X_blight_test.columns, 'unique
```

```
unique_counts_df['ratio'] = unique_counts_df['unique_counts']/X_blight

cols_all_same = unique_counts_df[unique_counts_df['unique_counts']==1]
sel_nonsame_cols = X_blight_test.columns[~X_blight_test.columns.isin(c
X_blight_test = X_blight_test[sel_nonsame_cols]

# cols_all_unique = unique_counts_df[unique_counts_df['ratio']>0.99][
# sel_multi_value_cols = X_blight_test.columns[~X_blight_test.columns.
# X_blight_test = X_blight_test[sel_multi_value_cols]
X_blight_test.set_index('ticket_id', inplace=True)

X_blight_test['zip_code'] = pd.to_numeric(X_blight_test['zip_code'], e

non_relevant_columns = ['violator_name']
relevant_columns= X_blight_test.columns[~X_blight_test.columns.isin(no
X_blight_test = X_blight_test[relevant_columns]

dependant_columns = ['violation_description', 'judgment_amount']
non_dependant_columns= X_blight_test.columns[~X_blight_test.columns.is
X_blight_test = X_blight_test[non_dependant_columns]


remove_address_columns = ['country', 'violation_street_number', 'viola
relevant_addressother_columns= X_blight_test.columns[~X_blight_test.co
X_blight_test = X_blight_test[relevant_addressother_columns]

X_blight_test['late_feeBool'] = X_blight_test['late_fee']>0
booleanizedColumnsToDrop = ['late_fee']
X_blight_test.drop(booleanizedColumnsToDrop, axis=1, inplace=True)

other_discount_amount_columns= X_blight_test.columns[~X_blight_test.co
X_blight_test = X_blight_test[other_discount_amount_columns]


X_blight_test['lat'].fillna(X_blight_test.lat.mean(),inplace = True)
X_blight_test['lon'].fillna(X_blight_test.lon.mean(),inplace = True)


other_columns_remove = ['zip_code', 'address']

# other_columns_remove = ['zip_code', 'lat', 'lon', 'address']
relevant_columns_left= X_blight_test.columns[~X_blight_test.columns.is
X_blight_test = X_blight_test[relevant_columns_left]


# Problem was - test agencies were only 3 - so test columns were less.
# Solution - we can select only those number of columns from train du
# BUT - what is there are different dummy columns?? - Will there be a
```

```
# HENCE - Commenting dummy columns for the time being.

n = 3
# top10srs = X_blight_test['agency_name'].value_counts().nlargest(n)
# X_blight_test['agency_name'] = [agency if agency in top10srs.index.v
# top10srs = X_blight_test['inspector_name'].value_counts().nlargest(n
# X_blight_test['inspector_name'] = [inspector if inspector in top10sr
# top10srs = X_blight_test['city'].value_counts().nlargest(n)
# X_blight_test['city'] = [city if city in top10srs.index.values else
# top10srs = X_blight_test['state'].value_counts().nlargest(n)
# X_blight_test['state'] = [state if state in top10srs.index.values el
top10srs = X_blight_test['violation_code'].value_counts().nlargest(n)
X_blight_test['violation_code'] = [violation_code if violation_code in
# top10srs = X_blight_test['fine_amount'].value_counts().nlargest(n)
# X_blight_test['fine_amount'] = [fine_amount if fine_amount in top10s
# top10srs = X_blight_test['disposition'].value_counts().nlargest(n)
# X_blight_test['disposition'] = [disposition if disposition in top10s

# agency_dummies = pd.get_dummies(X_blight_test['agency_name'])
# inspector_dummies = pd.get_dummies(X_blight_test['inspector_name'])
# city_dummies = pd.get_dummies(X_blight_test['city'])
# state_dummies = pd.get_dummies(X_blight_test['state'])
# violation_code_dummies = pd.get_dummies(X_blight_test['violation_coc
# fine_amount_dummies = pd.get_dummies(X_blight_test['fine_amount'])
# disposition_dummies = pd.get_dummies(X_blight_test['disposition'])

# dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city',

dummifiedColumnsToDrop = ['agency_name', 'inspector_name', 'city', 'st
X_blight_test.drop(dummifiedColumnsToDrop, axis=1, inplace=True)

# X_blight_test = X_blight_test.join(agency_dummies)
# X_blight_test = X_blight_test.join(inspector_dummies)
# X_blight_test = X_blight_test.join(city_dummies)
# X_blight_test = X_blight_test.join(state_dummies)
X_blight_test = X_blight_test.join(violation_code_dummies)
# X_blight_test = X_blight_test.join(fine_amount_dummies)
# X_blight_test = X_blight_test.join(disposition_dummies)

date_columns_remove = ['ticket_issued_date', 'hearing_date']
nondate_columns_left= X_blight_test.columns[~X_blight_test.columns.isi
X_blight_test = X_blight_test[nondate_columns_left]

# MANUALLY dropping ['violation_zip_code', 'clean_up_cost']
# as shortage of time before assignment submission
manually_removing_columns = ['violation_zip_code', 'clean_up_cost']
X_blight_test.drop(manually_removing_columns, axis=1, inplace=True)
```

```
            X_blight_test.fillna(X_blight_test.mean(), inplace=True)
            X_blight_test.fillna(0.0, inplace=True)
            X_blight_test.reset_index()
            #finalSrs = pd.Series(cf.predict_proba(X_blight_test)[:, 1], index=ran
            finalSrs = pd.Series(cf.predict_proba(X_blight_test)[:, 1], index=X_bl
            return finalSrs
```

```
In [76]: blight_model()
```

```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2827: Dtype
  if self.run_code(code, result):
```

```
Out[76]: ticket_id
         284932    0.044661
         285362    0.030853
         285361    0.046821
         285338    0.044823
         285346    0.046933
         285345    0.044824
         285347    0.048022
         285342    0.332277
         285530    0.030977
         284989    0.038997
         285344    0.048012
         285343    0.030921
         285340    0.030929
         285341    0.048024
         289828    0.039027
         289830    0.043804
         289829    0.043804
         292133    0.030929
         292134    0.048024
         285349    0.046934
         285348    0.044825
         284991    0.038997
         285532    0.039084
         286073    0.039084
         285406    0.038938
         285001    0.038999
         285006    0.030907
         365862    0.326846
         285405    0.030854
         287857    0.015217
                     ...
         376276    0.038937
         376218    0.043719
```

```
376368    0.043772
376369    0.044792
376225    0.043710
376222    0.038886
376362    0.043655
376363    0.044672
376228    0.043735
376265    0.043729
376286    0.326315
376320    0.043725
376314    0.043674
376327    0.326725
376435    0.276067
376434    0.047832
376459    0.046834
376478    0.004612
376473    0.043695
376484    0.042751
376482    0.038880
376480    0.038880
376479    0.038880
376481    0.038880
376483    0.043641
376496    0.030874
376497    0.030874
376499    0.046762
376500    0.046762
369851    0.348292
dtype: float64
```

In [91]:
```
# Use LabelEncoder instead of dummies (pd.get_dummies) as when there are l
# But with LabelEncoder we will have single column with different values f



# from sklearn.preprocessing import LabelEncoder
# from sklearn.ensemble import RandomForestRegressor
# from sklearn.ensemble import GradientBoostingClassifier
# from sklearn.model_selection import train_test_split
# from sklearn.model_selection import GridSearchCV


# train = pd.read_csv('train.csv',encoding='ISO-8859-1')
# test = pd.read_csv('readonly/test.csv',encoding='ISO-8859-1')
# test.set_index(test['ticket_id'],inplace=True)

# # Cleaning:
# train.dropna(subset=['compliance'],inplace=True)
```

```
# train = train[train['country']=='USA']
# #test = test[test['country']=='USA']

# label_encoder = LabelEncoder()
# label_encoder.fit(train['disposition'].append(test['disposition'], ignor
# train['disposition'] = label_encoder.transform(train['disposition'])
# test['disposition'] = label_encoder.transform(test['disposition'])

# label_encoder = LabelEncoder()
# label_encoder.fit(train['violation_code'].append(test['violation_code'],
# train['violation_code'] = label_encoder.transform(train['violation_code
# test['violation_code'] = label_encoder.transform(test['violation_code'])

# feature_names=['disposition','violation_code']
# X = train[feature_names]
# y = train['compliance']
# test = test[feature_names]
# X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=0)

# # grid search
# model = RandomForestRegressor()
# param_grid = {'n_estimators':[5,7], 'max_depth':[5,10]}
# grid_search = GridSearchCV(model, param_grid, scoring="roc_auc")
# grid_result = grid_search.fit(X_train, y_train)
```

In [87]:
```
# grid_result.cv_results_
# grid_result.best_score_
# 0.77247031568651303

# pd.DataFrame(grid_result.predict(test),index=test.index,columns=['compli
# 0.77247031568651303
```

Out[87]: 0.77247031568651303