Predicting Property Maintenance Fines

1. Data description

- 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 the violation occurred
  mailing_address_str_number, mailing_address_str_name, city, state, zip_code, non_us_str_code, country - Mailing address of the violator
  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-compliant

- readonly/addresses.csv & readonly/latlons.csv

mapping from ticket id to addresses, and from addresses to lat/lon coordinates.

2. Load the datasets

In [79]:

```
import pandas as pd
import numpy as np
```

```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

from IPython.display import display
pd.options.display.max_columns = None
```

In [80]:

```python
#figure out the encoding
with open('test.csv') as f:
    print(f)
```

```
<_io.TextIOWrapper name='test.csv' mode='r' encoding='cp1252'>
```

In [81]:

```python
train = pd.read_csv('train.csv', encoding = "cp1252", low_memory=False)
```

In [82]:

```python
train.head(3)
```

Out[82]:

	ticket_id	agency_name	inspector_name	violator_name	violation_street_number	violation_street_name	violation_:
0	22056	Buildings, Safety Engineering & Env Department	Sims, Martinzie	INVESTMENT INC., MIDWEST MORTGAGE	2900.0	TYLER	
1	27586	Buildings, Safety Engineering & Env Department	Williams, Darrin	Michigan, Covenant House	4311.0	CENTRAL	
2	22062	Buildings, Safety Engineering & Env Department	Sims, Martinzie	SANDERS, DERRON	1449.0	LONGFELLOW	

In [83]:

```python
test = pd.read_csv('test.csv', encoding = "cp1252", low_memory=False)
test.head(3)
```

Out[83]:

	ticket_id	agency_name	inspector_name	violator_name	violation_street_number	violation_street_name	violation_:
0	284932	Department of Public Works	Granberry, Aisha B	FLUELLEN, JOHN A	10041.0	ROSEBERRY	
1	285362	Department of Public Works	Lusk, Gertrina	WHIGHAM, THELMA	18520.0	EVERGREEN	
2	285361	Department of Public Works	Lusk, Gertrina	WHIGHAM, THELMA	18520.0	EVERGREEN	

In [84]:

```python
latlons = pd.read_csv('latlons.csv')
address =  pd.read_csv('addresses.csv')
address.head()
```

Out[84]:

	ticket_id	address
0	22056	2900 tyler, Detroit MI
1	27586	4311 central, Detroit MI
2	22062	1449 longfellow, Detroit MI
3	22084	1441 longfellow, Detroit MI
4	22093	2449 churchill, Detroit MI

In [85]:

```
latlons.head()
```

Out[85]:

	address	lat	lon
0	4300 rosa parks blvd, Detroit MI 48208	42.346169	-83.079962
1	14512 sussex, Detroit MI	42.394657	-83.194265
2	3456 garland, Detroit MI	42.373779	-82.986228
3	5787 wayburn, Detroit MI	42.403342	-82.957805
4	5766 haverhill, Detroit MI	42.407255	-82.946295

3. Explore the datasets

For training set:

In [86]:

```
len(train)
```

Out[86]:

```
250306
```

In [87]:

```
train.columns
```

Out[87]:

```
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_amount',
       '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 [88]:

```
train.dtypes
```

Out[88]:

```
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
```

In [89]:

```
categorical_variables_train = list(train.select_dtypes(include=['object']).columns)
categorical_variables_train
```

Out[89]:

```
['agency_name',
 'inspector_name',
 'violator_name',
 'violation_street_name',
 'mailing_address_str_name',
 'city',
 'state',
 'zip_code',
 'non_us_str_code',
 'country',
 'ticket_issued_date',
 'hearing_date',
 'violation_code',
 'violation_description',
 'disposition',
 'payment_date',
 'payment_status',
 'collection_status',
 'grafitti_status',
 'compliance_detail']
```

In [90]:

```
numerical_variables_train = list(train.select_dtypes(include=['float64','int64']).columns)
numerical_variables_train
```

Out[90]:

```
['ticket_id',
 'violation_street_number',
 'violation_zip_code',
 'mailing_address_str_number',
 'fine_amount',
 'admin_fee',
 'state_fee',
 'late_fee',
 'discount_amount',
 'clean_up_cost',
```

```
 'judgment_amount',
 'payment_amount',
 'balance_due',
 'compliance']
```

In [91]:

```
display(train.describe(include='all'))
```

	ticket_id	agency_name	inspector_name	violator_name	violation_street_number	violation_street_name
count	250306.000000	250306	250306	250272	2.503060e+05	250306
unique	NaN	5	173	119992	NaN	1791
top	NaN	Buildings, Safety Engineering & Env Department	Morris, John	INVESTMENT, ACORN	NaN	SEVEN MILE
freq	NaN	157784	17926	809	NaN	3482
mean	152665.543099	NaN	NaN	NaN	1.064986e+04	NaN
std	77189.882881	NaN	NaN	NaN	3.188733e+04	NaN
min	18645.000000	NaN	NaN	NaN	0.000000e+00	NaN
25%	86549.250000	NaN	NaN	NaN	4.739000e+03	NaN
50%	152597.500000	NaN	NaN	NaN	1.024400e+04	NaN
75%	219888.750000	NaN	NaN	NaN	1.576000e+04	NaN
max	366178.000000	NaN	NaN	NaN	1.415411e+07	NaN

In [92]:

```
#missing value rate
pd.DataFrame(train.isna().sum()/len(train)).sort_values(by=[0],ascending=False).head(10)
```

Out[92]:

	0
violation_zip_code	1.000000
grafitti_status	0.999996
non_us_str_code	0.999988
collection_status	0.852592
payment_date	0.835749
compliance	0.361262
hearing_date	0.049903
mailing_address_str_number	0.014390
state	0.000372
violator_name	0.000136

In [93]:

```
#columns in the training data but not in the testing data
train_test_difference=[i for i in train.columns.tolist() if i not in test.columns.tolist()
train_test_difference.remove("compliance")
train_test_difference
```

```
['payment_amount',
```

Out[93]:
```
 'balance_due',
 'payment_date',
 'payment_status',
 'collection_status',
 'compliance_detail']
```

For testing set:

In [94]:
```
len(test)
```

Out[94]:
```
61001
```

In [95]:
```
test.columns
```

Out[95]:
```
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_amount',
       'admin_fee', 'state_fee', 'late_fee', 'discount_amount',
       'clean_up_cost', 'judgment_amount', 'grafitti_status'],
      dtype='object')
```

In [96]:
```
test.dtypes
```

Out[96]:
```
ticket_id                       int64
agency_name                    object
inspector_name                 object
violator_name                  object
violation_street_number       float64
violation_street_name          object
violation_zip_code             object
mailing_address_str_number     object
mailing_address_str_name       object
city                           object
state                          object
zip_code                       object
non_us_str_code               float64
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
grafitti_status                object
dtype: object
```

In [97]:
```
categorical_variables_test = list(test.select_dtypes(include=['object']).columns)
categorical_variables_test
```

Out[97]:
```
['agency_name',
 'inspector_name',
 'violator_name',
```

```
 'violation_street_name',
 'violation_zip_code',
 'mailing_address_str_number',
 'mailing_address_str_name',
 'city',
 'state',
 'zip_code',
 'country',
 'ticket_issued_date',
 'hearing_date',
 'violation_code',
 'violation_description',
 'disposition',
 'grafitti_status']
```

In [98]:

```
numerical_variables_test = list(test.select_dtypes(include=['float64','int64']).columns)
numerical_variables_test
```

Out[98]:

```
['ticket_id',
 'violation_street_number',
 'non_us_str_code',
 'fine_amount',
 'admin_fee',
 'state_fee',
 'late_fee',
 'discount_amount',
 'clean_up_cost',
 'judgment_amount']
```

In [99]:

```
display(test.describe(include='all'))
```

	ticket_id	agency_name	inspector_name	violator_name	violation_street_number	violation_street_name
count	61001.000000	61001	61001	60973	6.100100e+04	61001
unique	NaN	3	116	38515	NaN	1477
top	NaN	Department of Public Works	Zizi, Josue	HOMES LDHA LP, MLK	NaN	MCNICHOLS
freq	NaN	40731	6293	91	NaN	1125
mean	331724.532811	NaN	NaN	NaN	1.256638e+04	NaN
std	25434.932141	NaN	NaN	NaN	1.414373e+05	NaN
min	284932.000000	NaN	NaN	NaN	-1.512600e+04	NaN
25%	310111.000000	NaN	NaN	NaN	6.008000e+03	NaN
50%	332251.000000	NaN	NaN	NaN	1.213400e+04	NaN
75%	353031.000000	NaN	NaN	NaN	1.716500e+04	NaN
max	376698.000000	NaN	NaN	NaN	2.010611e+07	NaN

In [100…

```
#missing value rate
pd.DataFrame(test.isna().sum()/len(test)).sort_values(by=[0],ascending=False).head(10)
```

Out[100…

	0
non_us_str_code	1.000000

	0
grafitti_status	0.963591
violation_zip_code	0.606170
hearing_date	0.036016
mailing_address_str_number	0.016623
state	0.005426
violator_name	0.000459
zip_code	0.000049
mailing_address_str_name	0.000049
city	0.000016

For address dataset:

In [101…

```
len(address)
```

Out[101…

```
311307
```

In [102…

```
address.columns
```

Out[102…

```
Index(['ticket_id', 'address'], dtype='object')
```

In [103…

```
address.dtypes
```

Out[103…

```
ticket_id      int64
address       object
dtype: object
```

In [104…

```
#missing value rate
pd.DataFrame(address.isna().sum()/len(address)).sort_values(by=[0],ascending=False).head(1
```

Out[104…

	0
ticket_id	0.0
address	0.0

For latlons dataset:

In [105…

```
len(latlons)
```

Out[105…

```
121769
```

In [106…

```
latlons.columns
```

Out[106…

```
Index(['address', 'lat', 'lon'], dtype='object')
```

In [107…

```
latlons.dtypes
```

Out[107…
```
address     object
lat        float64
lon        float64
dtype: object
```

In [108…
```
#missing value rate
pd.DataFrame(latlons.isna().sum()/len(latlons)).sort_values(by=[0],ascending=False).head(1
```

Out[108…

	0
lat	0.000057
lon	0.000057
address	0.000000

4. Initial Thoughts

General:

- There are over 250000 entries in the training set and over 60000 entries in the test set.
- There are some NaN values exist in the dataset that need to be dealt with.

For modeling:

- violation_zip_code, grafitti_status, non_us_str_code has high NaN rate in both the training and testing set so they should also be removed (both from the training and testing set).
- There are many categorical variables need to be converted into dummy variables.
- Some variables may not proper to used as predictors.(need further consideration to pick the predictors)
- The latlons and address dataframes should be joint to the main dataset(train/test) to provide another two important features for prediction: lat and lon.
- Drop all the rows with NaN compliance.
- Drop all the features in the train dataframe that are not in the test dataframe, because they would not be able to used to predict.
- There are some addresses that don't have provided lat and lon.

5. Data wrangling

In [109…
```
#mapping id to coordinates
id_latlon=address.merge(latlons,how="left",on="address")
id_latlon.drop('address', axis=1, inplace=True)
id_latlon.head()
```

Out[109…

	ticket_id	lat	lon
0	22056	42.390729	-83.124268
1	27586	42.326937	-83.135118
2	22062	42.380516	-83.096069
3	22084	42.380570	-83.095919
4	22093	42.145257	-83.208233

In [110…
```
#datasets modified for modeling
test_m=test.copy()
train_m=train.copy()
```

In [111…
```
#drop all the columns in training set that doesn't exist in test dataset
train_m.drop(train_test_difference, axis=1, inplace=True)
```

In [112…
```
#drop the columns in both the training and testing set that have a big NaN rate
columns_to_drop_nan=['violation_zip_code', 'grafitti_status', 'non_us_str_code']
train_m.drop(columns_to_drop_nan, axis=1, inplace=True)
test_m.drop(columns_to_drop_nan, axis=1, inplace=True)
```

In [113…
```
#map the training/testing set to the lats and lons
train_m=train_m.merge(id_latlon,how="left",on="ticket_id")
test_m=test_m.merge(id_latlon,how="left",on="ticket_id")
```

In [114…
```
#drop the rows with null compliance in the training set
train_m = train_m[train_m.compliance.notnull()]
```

In [115…
```
#make sure the columns are aligned
set(train_m.columns.tolist())-set(test_m.columns.tolist())
```

Out[115…
```
{'compliance'}
```

Choose the predictors:

In [116…
```
numerical=list(train_m.select_dtypes(include=['float64','int64']).columns)
numerical
```

Out[116…
```
['ticket_id',
 'violation_street_number',
 'mailing_address_str_number',
 'fine_amount',
 'admin_fee',
 'state_fee',
 'late_fee',
 'discount_amount',
 'clean_up_cost',
 'judgment_amount',
 'compliance',
 'lat',
 'lon']
```

In [117…
```
categorical=list(train_m.select_dtypes(include=['object']).columns)
categorical
```

Out[117…
```
['agency_name',
 'inspector_name',
 'violator_name',
 'violation_street_name',
 'mailing_address_str_name',
 'city',
 'state',
 'zip_code',
 'country',
 'ticket_issued_date',
```

```
 'hearing_date',
 'violation_code',
 'violation_description',
 'disposition']
```

Pick the categorical variables used to predict:

In [118…

```
#Some categorical variables are not proper to predict:
columns_to_drop=['inspector_name','violator_name','zip_code','violation_street_name','mai
train_m.drop(columns_to_drop, axis=1, inplace=True)
test_m.drop(columns_to_drop, axis=1, inplace=True)
```

In [119…

```
categorical=list(train_m.select_dtypes(include=['object']).columns)
categorical
```

Out[119…

```
['agency_name', 'city', 'state', 'violation_code', 'disposition']
```

In [120…

```
for i in categorical:
    print(len(pd.unique(train_m[i])))
```

```
5
4093
60
189
4
```

In [121…

```
#drop "city"
train_m.drop("city", axis=1, inplace=True)
test_m.drop("city", axis=1, inplace=True)
```

In [122…

```
objects=list(train_m.select_dtypes(include=['object']).columns)
```

In [123…

```
#list(test_m.select_dtypes(include=['object']).columns)
```

In [124…

```
#align the unique values to create dummy variables
def align_dummy(df1,df2,var):
    diff1_2=list(set(pd.unique(df1[var]))-set(pd.unique(df2[var])))
    diff2_1=list(set(pd.unique(df2[var]))-set(pd.unique(df1[var])))
    for i in diff1_2:
        df1=df1[df1[var]!=i]
    #for i in diff2_1:
        #df2=df2[df2[var]!=i]
    for i in diff2_1:
        df2.loc[df2[var]==i,var] =df1[var].iloc[3]
    #any value from the training set... just to make the test set "not removed any row", a
    return df1,df2
```

In [125…

```
for i in list(objects):
    train_m,test_m=align_dummy(train_m,test_m,i)
```

In [126…

```
#convert the remaining categorical columns into dummy variables
dummy_columns=list(train_m.select_dtypes(include=['object']).columns)
train_m = pd.get_dummies(train_m, columns=dummy_columns)
test_m = pd.get_dummies(test_m, columns=dummy_columns)
```

In [127…

```
set(train_m.columns.tolist())-set(test_m.columns.tolist())
```

Out[127…

```
{'compliance'}
```

Pick the numerical variables used to predict:

In [128…

```
numerical=list(train_m.select_dtypes(include=['float64','int64']).columns)
numerical
```

Out[128…

```
['ticket_id',
 'violation_street_number',
 'fine_amount',
 'admin_fee',
 'state_fee',
 'late_fee',
 'discount_amount',
 'clean_up_cost',
 'judgment_amount',
 'compliance',
 'lat',
 'lon']
```

In [129…

```
numerical2=list(test_m.select_dtypes(include=['float64','int64']).columns)
numerical2
#set(numerical2)-set(numerical)
```

Out[129…

```
['ticket_id',
 'violation_street_number',
 'fine_amount',
 'admin_fee',
 'state_fee',
 'late_fee',
 'discount_amount',
 'clean_up_cost',
 'judgment_amount',
 'lat',
 'lon']
```

In [130…

```
#Some numerical variables are not proper to predict:
#len(pd.unique(train_m.'mailing_address_str_number'))
columns_to_drop=['ticket_id','state_fee','violation_street_number','admin_fee','late_fee']
                 #late_fee: 10%, the same with find_amount
train_m.drop(columns_to_drop, axis=1, inplace=True)
test_m.drop(columns_to_drop, axis=1, inplace=True)
```

In [131…

```
#now, see the datasets
train_m.head()
```

Out[131…

	fine_amount	discount_amount	clean_up_cost	judgment_amount	compliance	lat	lon	agency_nar Safety E Env
0	250.0	0.0	0.0	305.0	0.0	42.390729	-83.124268	
5	250.0	0.0	0.0	305.0	0.0	42.145257	-83.208233	
28	250.0	0.0	0.0	305.0	0.0	42.383385	-83.072582	
30	250.0	0.0	0.0	305.0	0.0	42.389290	-83.134006	

	fine_amount	discount_amount	clean_up_cost	judgment_amount	compliance	lat	lon	agency_nar Safety E Env
31	250.0	0.0	0.0	305.0	0.0	42.393440	-83.127929	

In [132…

```
test_m.head()
```

Out[132…

	fine_amount	discount_amount	clean_up_cost	judgment_amount	lat	lon	agency_name_Buildings, Safety Engineering & Env Department
0	200.0	0.0	0.0	250.0	42.407581	-82.986642	0
1	1000.0	0.0	0.0	1130.0	42.426239	-83.238259	0
2	100.0	0.0	0.0	140.0	42.426239	-83.238259	0
3	200.0	0.0	0.0	250.0	42.309661	-83.122426	0
4	100.0	0.0	0.0	140.0	42.308830	-83.121116	0

Now, deal with the NaN values:

In [133…

```
pd.DataFrame(train_m.isna().sum()).sort_values(by=[0],ascending=False).head(5)
```

Out[133…

	0
lat	2
lon	2
fine_amount	0
violation_code_9-1-103 (a) or (b)	0
violation_code_61-8-127	0

In [134…

```
pd.DataFrame(test_m.isna().sum()).sort_values(by=[0],ascending=False).head(5)
```

Out[134…

	0
lat	5
lon	5
fine_amount	0
violation_code_9-1-103(C)	0
violation_code_61-8-27	0

In [135…

```
len(train_m),len(test_m)
```

Out[135…

```
(152354, 61001)
```

In [136…

```
#pretty good - drop rows with nan values
train_m=train_m.dropna(how='any')
#test_m=test_m.dropna(how='any')
```

```
test_m=test_m.fillna(0) #just not remove any row from test set(a lazy way, because only 5
len(train_m),len(test_m)
```

Out[136…

```
(152352, 61001)
```

In [ ]:

6. Modeling

Create X_train, y_train and scaling

In [137…

```
train_features = train_m.columns.drop('compliance').tolist()
X_train = train_m[train_features].values
y_train = train_m['compliance'].values

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(test_m.values)
```

help functions

In [138…

```
#cv evaluation the model (using auc)
def auc_cv(model,X_train,y_train,fold=5):
    cv_scores_auc = cross_val_score(model,X_train,y_train, cv=fold, scoring = 'roc_auc')
    return ('{:.3f}\n'.format(np.mean(cv_scores_auc)))
#to align with the output

test_original=pd.read_csv('test.csv', encoding = "cp1252", low_memory=False)
test_original.set_index('ticket_id', inplace=True)

def return_function(test_predict_proba):
    test_original['compliance'] = test_predict_proba
    return test_original.compliance
```

(1) Logistic regression

In [144…

```
logistic_clf = LogisticRegression(C=100,max_iter=10000)
logistic_clf.fit(X_train_scaled_subset, y_train_subset)
```

Out[144…

```
LogisticRegression(C=100, max_iter=10000)
```

In [145…

```
auc_cv(logistic_clf,X_train_scaled_subset,y_train_subset)
```

Out[145…

```
'0.763\n'
```

In [146…

```
test_predict_logistic_proba=logistic_clf.predict_proba(X_test_scaled)[:,1]
```

In [147…

```
test_predict_logistic_proba
```

Out[147…

```
array([0.18550603, 0.02065259, 0.08471621, ..., 0.10407106, 0.10408304,
       0.07516562])
```

In [148…
```
return_function(test_predict_logistic_proba)
```
Out[148…
```
ticket_id
284932    0.185506
285362    0.020653
285361    0.084716
285338    0.052964
285346    0.105009
            ...
376496    0.085585
376497    0.085585
376499    0.104071
376500    0.104083
369851    0.075166
Name: compliance, Length: 61001, dtype: float64
```

(2) SVM

In [149…
```
svm_clf = SVC(kernel = 'rbf', gamma = 1, C = 15,probability=True) #predict probs
svm_clf.fit(X_train_scaled_subset, y_train_subset)
```
Out[149…
```
SVC(C=15, gamma=1, probability=True)
```
In [150…
```
auc_cv(svm_clf,X_train_scaled_subset,y_train_subset)
```
Out[150…
```
'0.634\n'
```
In [151…
```
test_predict_svm_proba=svm_clf.predict_proba(X_test_scaled)[:,1]
```
In [152…
```
test_predict_svm_proba
```
Out[152…
```
array([0.06601662, 0.06569584, 0.06806311, ..., 0.06328469, 0.06328575,
       0.06636747])
```
In [153…
```
return_function(test_predict_svm_proba)
```
Out[153…
```
ticket_id
284932    0.066017
285362    0.065696
285361    0.068063
285338    0.067180
285346    0.066599
            ...
376496    0.067524
376497    0.067524
376499    0.063285
376500    0.063286
369851    0.066367
Name: compliance, Length: 61001, dtype: float64
```

(3) Random Forest

In [154…
```
randomforest_clf= RandomForestClassifier()
randomforest_clf.fit(X_train_scaled_subset, y_train_subset)
```
Out[154…
```
RandomForestClassifier()
```

In [155…
```
auc_cv(randomforest_clf,X_train_scaled_subset,y_train_subset)
```

Out[155…
```
'0.754\n'
```

In [156…
```
test_predict_rf_proba=randomforest_clf.predict_proba(X_test_scaled)[:,1]
```

In [157…
```
test_predict_rf_proba
```

Out[157…
```
array([0.11, 0.04, 0.24, ..., 0.08, 0.08, 0.56])
```

In [158…
```
return_function(test_predict_rf_proba)
```

Out[158…
```
ticket_id
284932    0.11
285362    0.04
285361    0.24
285338    0.03
285346    0.09
          ...
376496    0.03
376497    0.03
376499    0.08
376500    0.08
369851    0.56
Name: compliance, Length: 61001, dtype: float64
```