In [1]:

```
import os
os.chdir("E:\\Debtor Delinquency Prediction\\")
```

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
def read_data():
    path = input(str("Enter the path: "))
    data = pd.read_csv(path)
    return data
```

In [4]:

```
train = read_data()
```

```
Enter the path: E:\\Debtor Delinquency Prediction\\train.csv
```

In [66]:

```
test = read_data()
```

```
Enter the path: E:\\Debtor Delinquency Prediction\\test.csv
```

In [67]:

```
test_loan_id = test['loan_id']
```

In [6]:

```
data = train.append(test,ignore_index=True)
```

In [7]:

```
test.head()
```

Out[7]:

	loan_id	source	financial_institution	interest_rate	unpaid_principal_bal	loan_term	originatic
0	1	Y	Browning-Hart	3.875	417000	360	(
1	2	X	OTHER	4.500	113000	360	(
2	3	Y	OTHER	4.500	72000	360	(
3	4	X	Miller, Mcclure and Allen	4.125	123000	180	(
4	5	X	Browning-Hart	3.250	166000	180	(

5 rows × 28 columns

In [8]:

```
train.head()
```

Out[8]:

	loan_id	source	financial_institution	interest_rate	unpaid_principal_bal	loan_term	or
0	268055008619	Z	Turner, Baldwin and Rhodes	4.250	214000	360	
1	672831657627	Y	Swanson, Newton and Miller	4.875	144000	360	
2	742515242108	Z	Thornton-Davis	3.250	366000	180	
3	601385667462	X	OTHER	4.750	135000	360	
4	273870029961	X	OTHER	4.750	124000	360	

5 rows × 29 columns

In [9]:

```
list(zip(train.columns,train.nunique(),train.isnull().sum()))
```

Out[9]:

```
[('loan_id', 116058, 0),
 ('source', 3, 0),
 ('financial_institution', 19, 0),
 ('interest_rate', 923, 0),
 ('unpaid_principal_bal', 646, 0),
 ('loan_term', 140, 0),
 ('origination_date', 3, 0),
 ('first_payment_date', 4, 0),
 ('loan_to_value', 92, 0),
 ('number_of_borrowers', 2, 0),
 ('debt_to_income_ratio', 58, 0),
 ('borrower_credit_score', 221, 0),
 ('loan_purpose', 3, 0),
 ('insurance_percent', 14, 0),
 ('co-borrower_credit_score', 216, 0),
 ('insurance_type', 2, 0),
 ('m1', 4, 0),
```

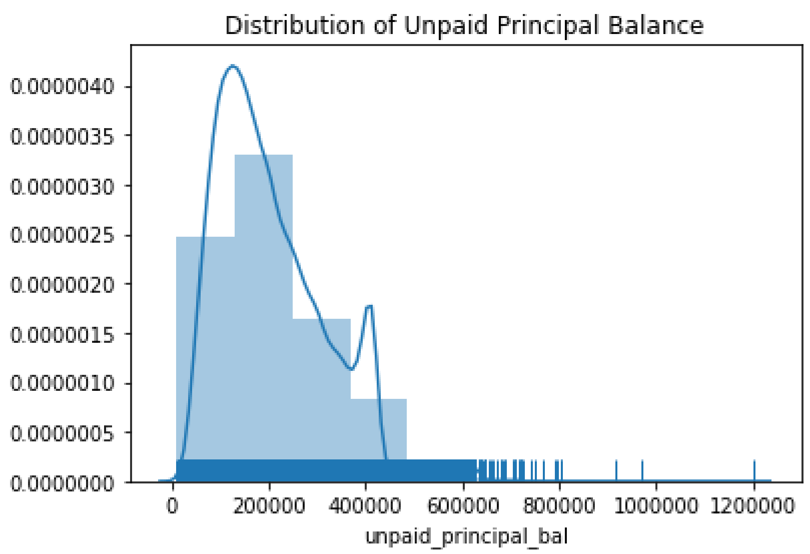
There is no null values in the dataset.

In [11]:

```
sns.distplot(train.unpaid_principal_bal,bins=10,rug=True)
plt.title("Distribution of Unpaid Principal Balance")

# Most of unpaid pricipal balance is between 0 to 400K

# The distribution is right skewed so mean will be greater than median.
```

Out[11]:

```
Text(0.5, 1.0, 'Distribution of Unpaid Principal Balance')
```

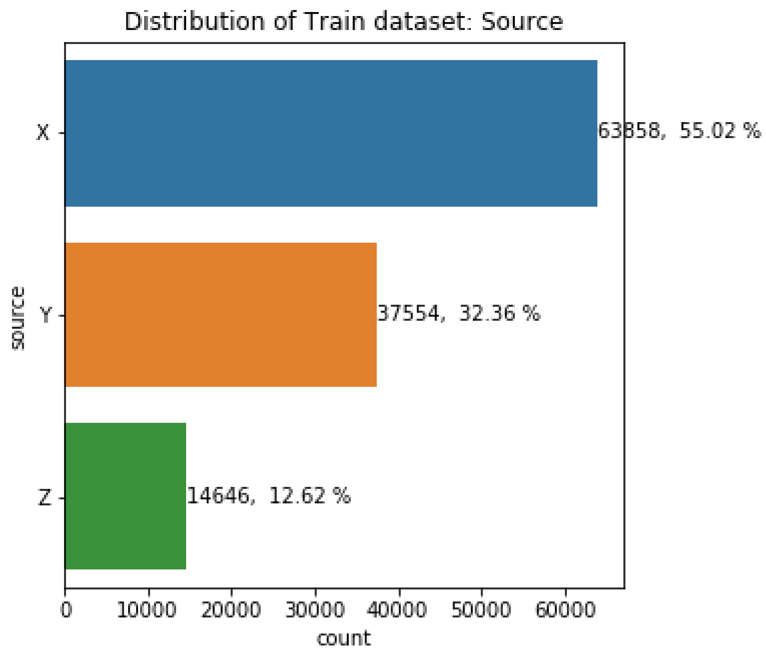


In [12]:

```
# This function returns the count plot of a column with percentage of each class
def plot_bar_counts_categorical(data_se, title, figsize, sort_by_counts=False):
    info = data_se.value_counts()
    info_norm = data_se.value_counts(normalize=True)
    categories = info.index.values
    counts = info.values
    counts_norm = info_norm.values
    fig, ax = plt.subplots(figsize=figsize)
    if data_se.dtype in ['object']:
        if sort_by_counts == False:
            inds = categories.argsort()
            counts = counts[inds]
            counts_norm = counts_norm[inds]
            categories = categories[inds]
        ax = sns.barplot(counts, categories, orient = 'h', ax=ax)
        ax.set(xlabel="count", ylabel=data_se.name)
        ax.set_title("Distribution of " + title)
        for n, da in enumerate(counts):
            ax.text(da, n, str(da)+ ",  " + str(round(counts_norm[n]*100,2)) + " %", fontsi
    else:
        inds = categories.argsort()
        counts_sorted = counts[inds]
        counts_norm_sorted = counts_norm[inds]
        ax = sns.barplot(categories, counts, orient = 'h', ax=ax)
        ax.set(xlabel=data_se.name, ylabel='count')
        ax.set_title("Distribution of " + title)
        for n, da in enumerate(counts_sorted):
            ax.text(n, da, str(da)+ ",  " + str(round(counts_norm_sorted[n]*100,2)) + " %",
```

In [13]:

```
plot_bar_counts_categorical(train['source'], 'Train dataset: Source', (5,5))
```

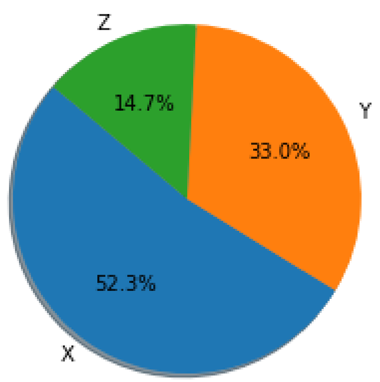


In [14]:

```
temp = train.groupby(['source'])['unpaid_principal_bal'] \
                    .sum() \
                    .reset_index(name = 'Sum') \
                    .sort_values(['Sum'],ascending=False)
labels = temp.source
sizes = temp.Sum
plt.pie(sizes,
        labels=labels,
        autopct='%1.1f%%',
        shadow=True,
        startangle=140)

# More than 80% for total unpaid balance is contributed by source x and y.
```

Out[14]:

```
([<matplotlib.patches.Wedge at 0x210638889b0>,
  <matplotlib.patches.Wedge at 0x21063895320>,
  <matplotlib.patches.Wedge at 0x21063895c50>],
 [Text(-0.6440095291888237, -0.891768874941254, 'X'),
  Text(0.974371618007160 9, 0.5104899117731, 'Y'),
  Text(-0.4383865740890013, 1.0088692738202056, 'Z')],
 [Text(-0.35127792501208566, -0.486419386331593, '52.3%'),
  Text(0.5314754280039059, 0.2784490427853272, '33.0%'),
  Text(-0.23911994950309157, 0.5502923311746575, '14.7%')])
```

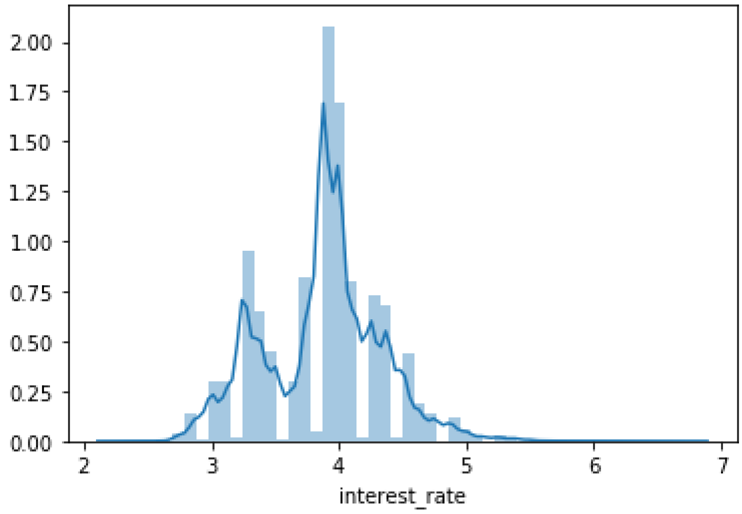


In [15]:

```
sns.distplot(train.interest_rate)

# Distribution of interest rate is from 2 to 7.
# Most datapoints lies between 3 to 5.
# The distribution seems to be normally distributed.
```

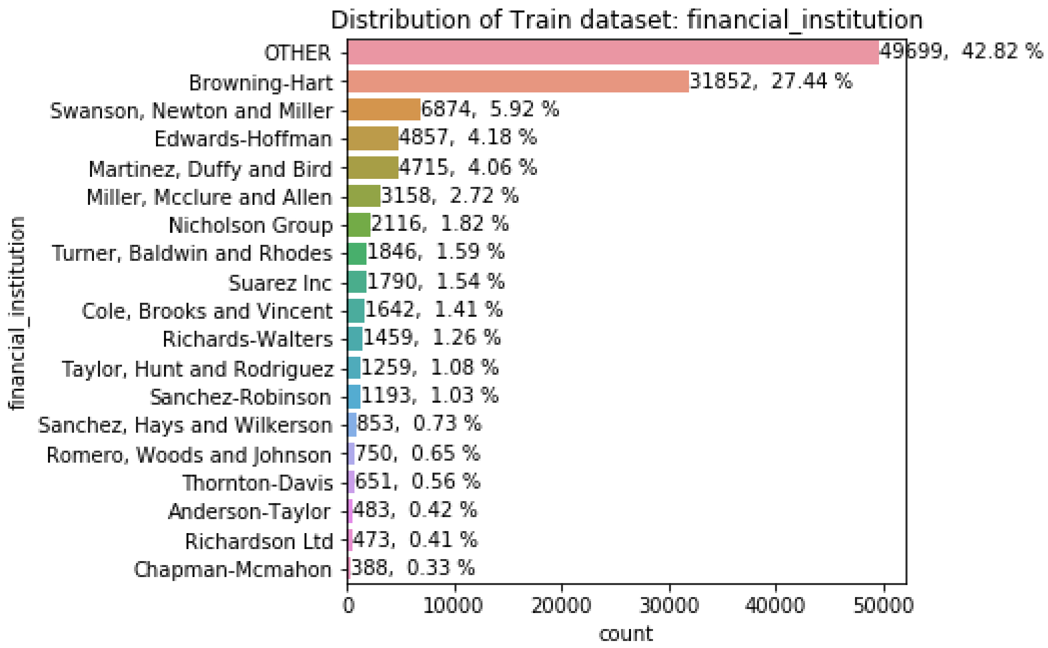
Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x210638a15f8>
```



In [16]:

```
plot_bar_counts_categorical(train['financial_institution'], 'Train dataset: financial_insti
```

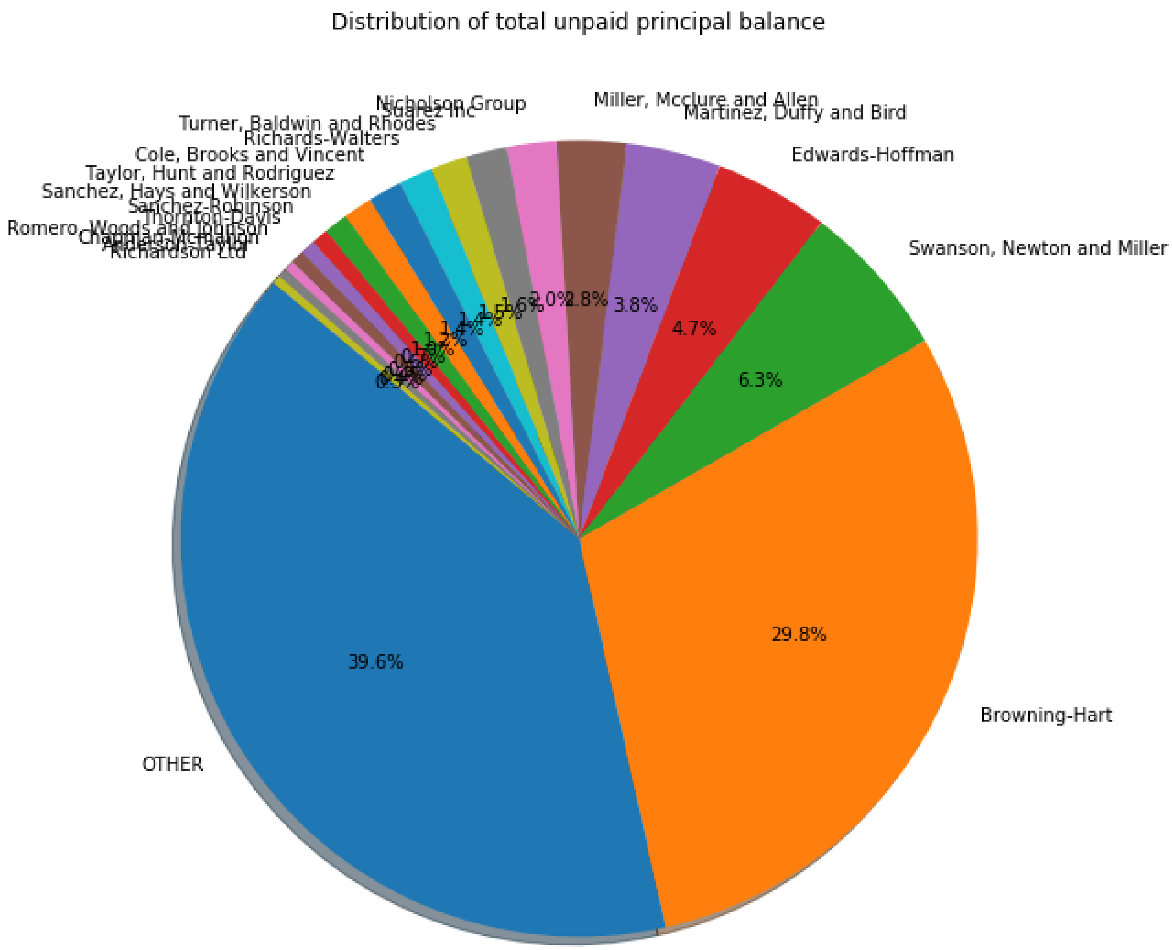


In [17]:

```
temp = train.groupby(['financial_institution'])['unpaid_principal_bal'] \
                    .sum() \
                    .reset_index(name = 'Sum') \
                    .sort_values(['Sum'],ascending=False)

labels = temp.financial_institution
sizes = temp.Sum

plt.figure(figsize=(10,10))
plt.xticks(rotation=180)
plt.pie(sizes,
        labels=labels,
        autopct='%1.1f%%',
        shadow=True,
        startangle=140)
plt.title("Distribution of total unpaid principal balance")
```

Out[17]:

```
Text(0.5, 1.0, 'Distribution of total unpaid principal balance')
```



In [18]:

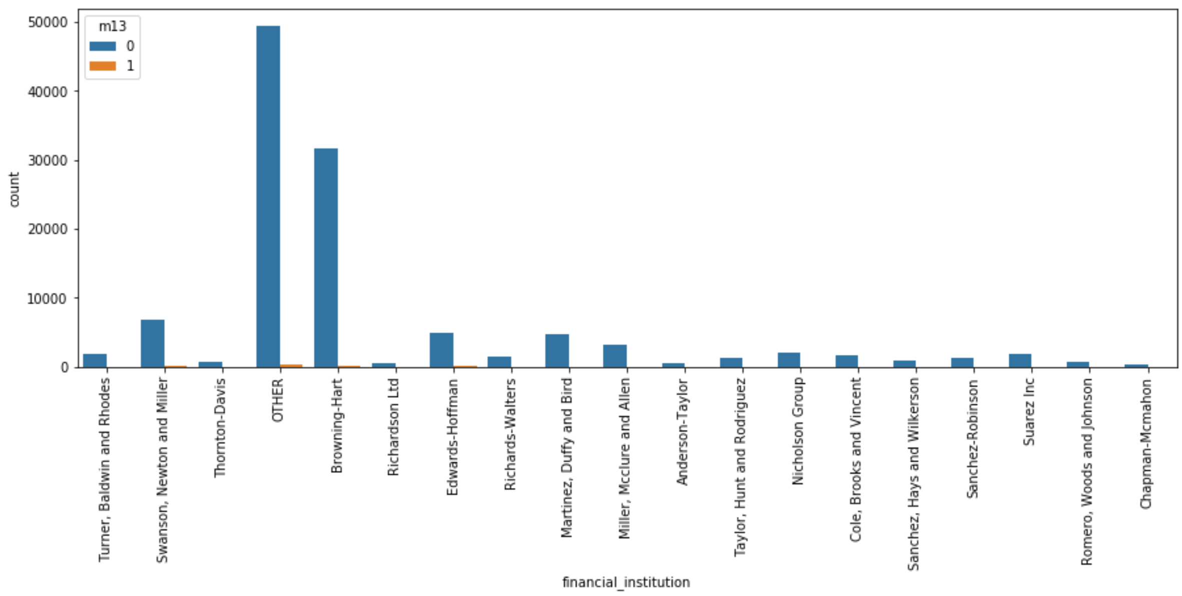
```
# Browning-Hart have sanctioned most number of loans contributing approx 27% and approx 30%
```

In [19]:

```
plt.figure(figsize=(15,5))
plt.xticks(rotation=90)
sns.countplot(x=train.financial_institution,hue=train.m13)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21063b7b7f0>
```

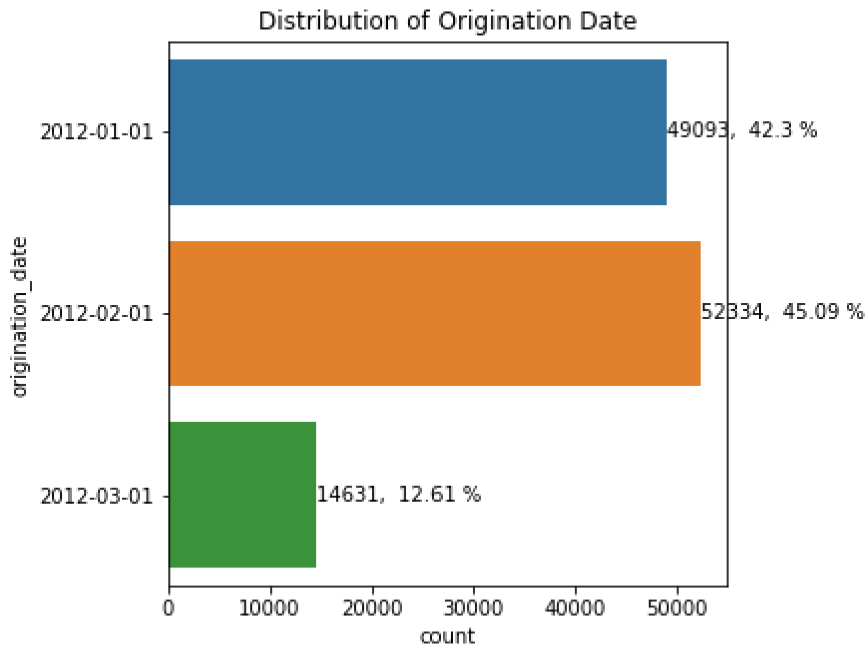


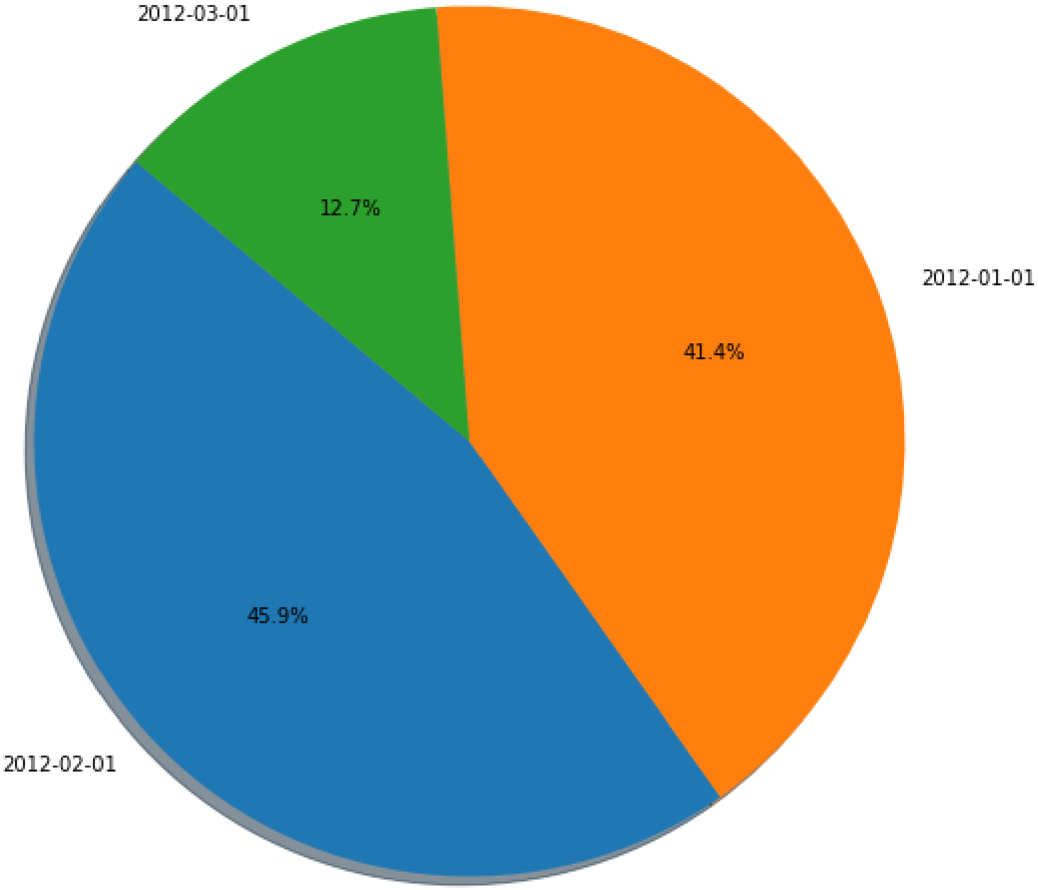
In [20]:

```
plot_bar_counts_categorical(train['origination_date'],"Origination Date",(5,5))

temp = train.groupby(['origination_date'])['unpaid_principal_bal'] \
                    .sum() \
                    .reset_index(name = 'Sum') \
                    .sort_values(['Sum'],ascending=False)

labels = temp.origination_date
sizes = temp.Sum

plt.figure(figsize=(10,10))
plt.xticks(rotation=180)
plt.pie(sizes,
        labels=labels,
        autopct='%1.1f%%',
        shadow=True,
        startangle=140)
```

Out[20]:

```
([<matplotlib.patches.Wedge at 0x21063a382b0>,
  <matplotlib.patches.Wedge at 0x21063a38be0>,
  <matplotlib.patches.Wedge at 0x21063dd4e48>],
 [Text(-0.8098851644625782, -0.7443695455776136, '2012-02-01'),
  Text(1.035196738147911, 0.3719781086676012, '2012-01-01'),
  Text(-0.5024432572358658, 0.9785452331181291, '2012-03-01')],
 [Text(-0.44175554425231534, -0.4060197521332437, '45.9%'),
  Text(0.5646527662624968, 0.20289715018232793, '41.4%'),
  Text(-0.27405995849229037, 0.5337519453371613, '12.7%')])
```





Studying Relationship of Predictor and Target Variables

In [21]:

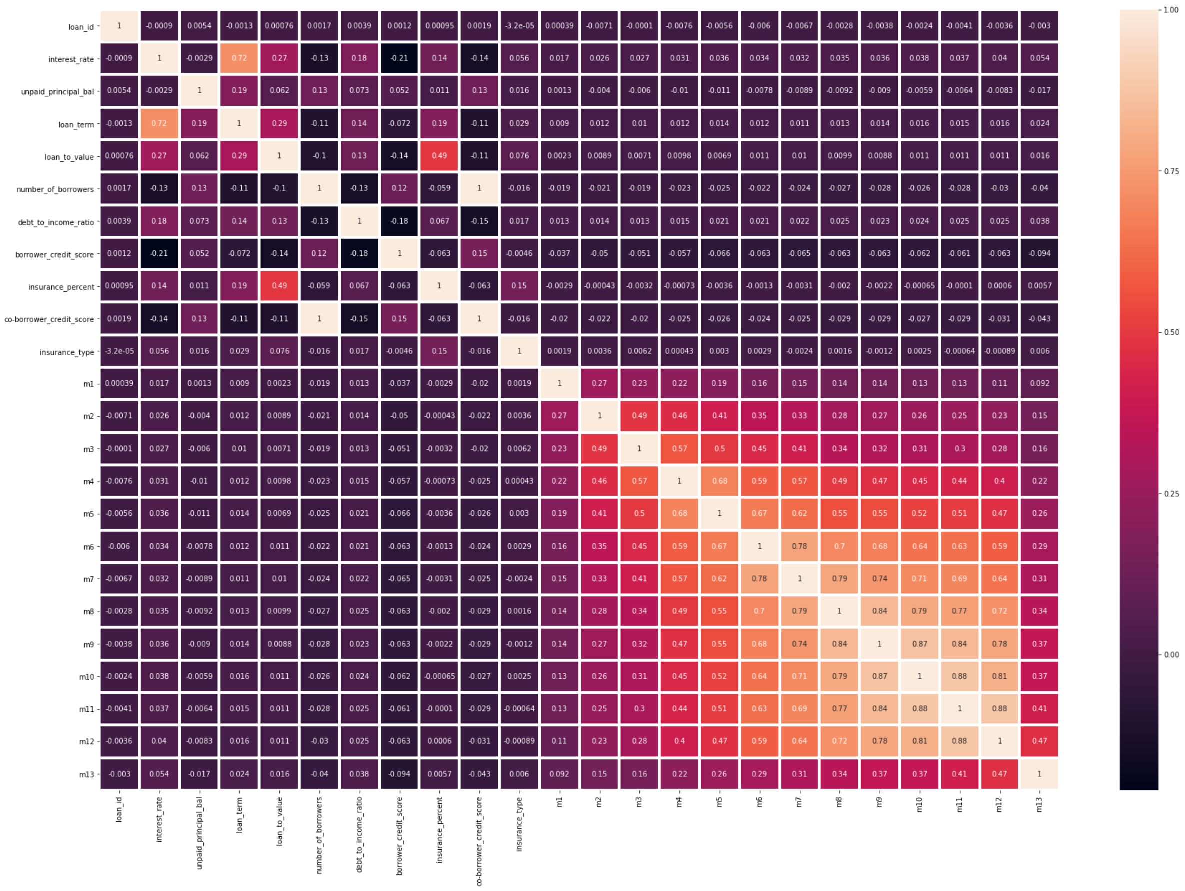
```
categorical_features = train.select_dtypes(include='object')
numerical_features = train.select_dtypes(include='number')
```

In [22]:

```
plt.figure(figsize=(30,20))
sns.heatmap(numerical_features.corr(),annot=True,linewidths=3)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21063a651d0>
```

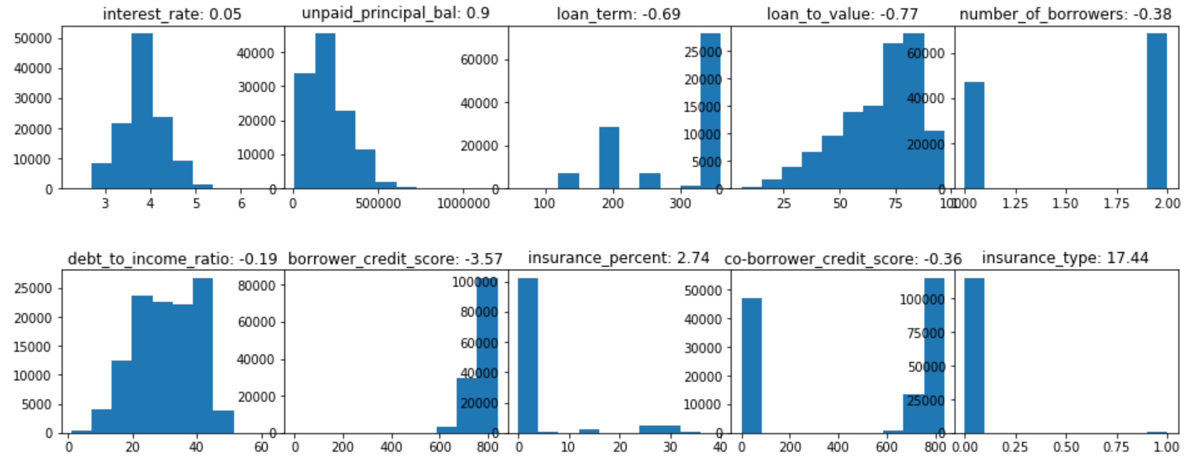


In [23]:

```
# Number of co borrower is proportional to co borrower credit score.
# Loan Term is directly proportional to interest rate.
# Insurance rate is related to loan to value.

# One of these columns should be removed.
```

In [24]:

```
fig, axs = plt.subplots(2,5, figsize=(16, 6), facecolor='w', edgecolor='k')
fig.subplots_adjust(hspace = .5, wspace=.001)
axs = axs.ravel()

for i,j in zip([i for i in numerical_features.columns.to_list()[1:] if len(i) >3],range(10)
    axs[j].hist(numerical_features[i])
    axs[j].set_title(i+': '+str(np.round(numerical_features[i].skew(),2)))
```



Interest Rate is normally distributed.

Unpaid Principal Balance is moderately right skewed.

Loan Term is moderately left skewed.

Loan to Value is moderately left skewed.

Number of borrowers is either 1 or 2.

Debt to income ratio is normally distributed.

Borrower Credit Score is highly left skewed.

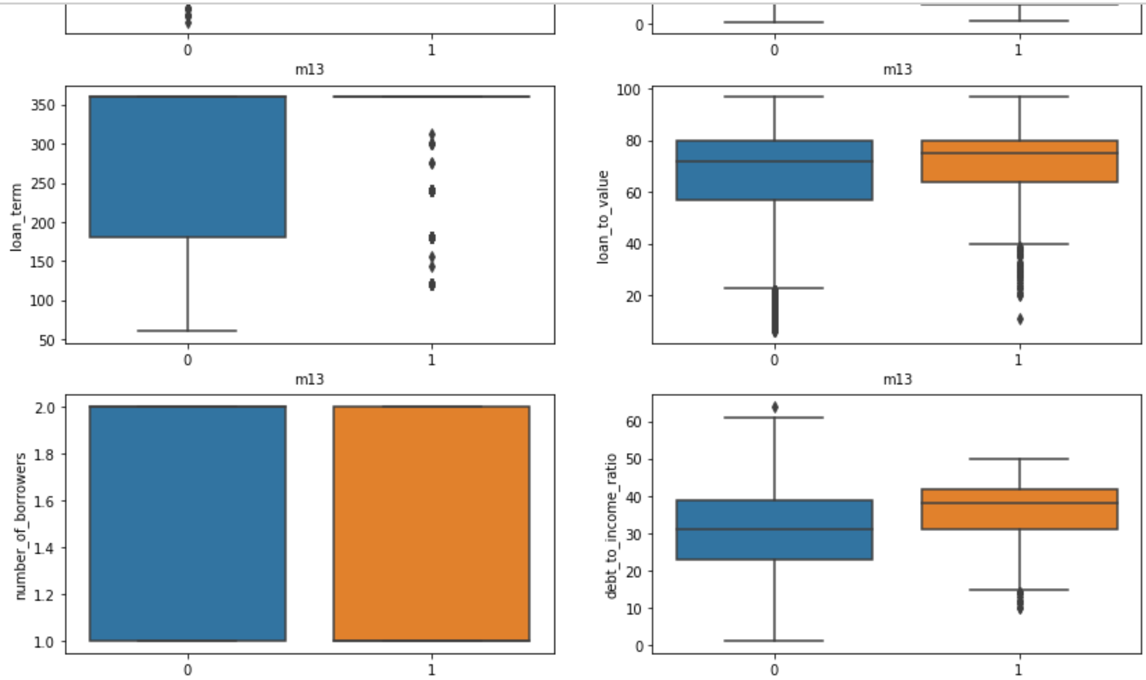
Insurance percent is highly right skewed.

Co-borrower credit score is moderately left skewed.

Insurance has 2 values only

In [25]:

```
fig, ((ax1, ax2), (ax3, ax4), (ax5, ax6), (ax7, ax8), (ax9,ax10)) = plt.subplots(nrows=5, r
AX = [ax1, ax2, ax3, ax4, ax5, ax6, ax7, ax8, ax9, ax10]

for i,j in zip([i for i in numerical_features.columns.to_list()[1:] if len(i) >3],AX):
    sns.boxplot(x = 'm13', y = i, data =numerical_features,ax=j)
```



Interest Rate is higher when a person is delinquent.

Debt to income ratio is higher if a person is delinquent.

Borrower credit score is lower if borrower is delinquent.

For other variables, there is no any relationship wrt to target variable (m13).

Feature Engineering

In [26]:

```
data.borrower_credit_score = pd.cut(data.borrower_credit_score,3,labels = ['low','medium','
```

In [27]:

```
data.debt_to_income_ratio = pd.cut(data.debt_to_income_ratio,5,labels = ['low','Mid Low','M
```

In [28]:

```
data.interest_rate = pd.cut(data.interest_rate,3,labels = ['low','medium','high'])
```

In [29]:

```
def insurance(insurance):
    if insurance == 0.0:
        return('No Insurance')
    else:
        return('Insured')

data.insurance_percent = data.insurance_percent.apply(insurance)
```

In [30]:

```
'] = data.m1 + data.m2 + data.m3 + data.m4 + data.m5 + data.m6 + data.m7 + data.m8 + data.m1
```

In [31]:

```
data['Deliquency Score'] = pd.cut(data['Deliquency Score'],5,labels = ['low','Mid Low','Mec
```

In [32]:

```
data.columns
```

Out[32]:

```
Index(['borrower_credit_score', 'co-borrower_credit_score',
       'debt_to_income_ratio', 'financial_institution', 'first_payment_dat
e',
       'insurance_percent', 'insurance_type', 'interest_rate', 'loan_id',
       'loan_purpose', 'loan_term', 'loan_to_value', 'm1', 'm10', 'm11', 'm1
2',
       'm13', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9',
       'number_of_borrowers', 'origination_date', 'source',
       'unpaid_principal_bal', 'Deliquency Score'],
      dtype='object')
```

In [33]:

```
cols = ['co-borrower_credit_score','first_payment_date','insurance_type','loan_id','loan_pu

data.drop(cols,inplace=True,axis=1)
```

In [34]:

```
data['unpaid_principal_bal'] = pd.cut(data['unpaid_principal_bal'],5,labels = ['low','Mid L
```

In [35]:

```
data.drop(['origination_date'],axis=1,inplace=True)
```

Data Wrangling

In [36]:

```
data = data[['borrower_credit_score', 'debt_to_income_ratio',
       'financial_institution', 'insurance_percent', 'interest_rate',
       'number_of_borrowers', 'source', 'unpaid_principal_bal',
       'Deliquency Score', 'm13']]
```

In [37]:

```
dummies = pd.get_dummies(data['borrower_credit_score'],drop_first=True)
```

In [38]:

```
data = pd.merge(data,dummies,left_index=True,right_index=True)
```

In [39]:

```
data1 = data.copy()
```

In [40]:

```
for i in data.columns[:9]:
    dummies = pd.get_dummies(data[i],drop_first=True)
    data = pd.merge(data,dummies,left_index=True,right_index=True)
    data.drop(i,inplace=True,axis=1)
```

In [41]:

```
train = data[data['m13'].notnull()]
```

In [42]:

```
test = data[data['m13'].isnull()]
```

In [61]:

```
x = train.iloc[:,1:]
y = train.iloc[:,:1]
x1 = test.iloc[:,1:]
y1 = test.iloc[:,:1]
```

Simple Model

In [44]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
from sklearn.feature_selection import RFE, RFECV
```

In [45]:

```
def simple_model(alg):

    # splitting data into training and validation set
    xtrain, xtest, ytrain, ytest = train_test_split(x, y.values.ravel(), random_state=42, t
    model = alg
    model.fit(xtrain, ytrain) # training the model
#     prediction = model.predict_proba(xtest) # predicting on the validation set
#     prediction_int = prediction[:,1] >= 0.3 # if prediction is greater than or equal to 0
#     prediction_int = prediction_int.astype(np.int)

    print("f1_score:",f1_score(ytest, model.predict(xtest))) # calculating f1 score
    print("Accuracy on train data:",model.score(xtrain,ytrain))
    print("Accuracy on test data:",model.score(xtest,ytest))
```

In [46]:

```
algs = [LogisticRegression(),DecisionTreeClassifier(),RandomForestClassifier(),AdaBoostClas
algs_lst = ['LR','DTC','RFC','ABC']
for alg,l in zip(algs,algs_lst):
    print(l)
    simple_model(alg=alg)
```

```
LR
f1_score: 0.07518796992481203
Accuracy on train data: 0.994797837278935
Accuracy on test data: 0.9947010167154919
DTC
f1_score: 0.1037037037037037
Accuracy on train data: 0.9949593951274153
Accuracy on test data: 0.9947871790453214
RFC
f1_score: 0.08888888888888888
Accuracy on train data: 0.9949378540809513
Accuracy on test data: 0.9947010167154919
ABC
f1_score: 0.1037037037037037
Accuracy on train data: 0.9948624604183272
Accuracy on test data: 0.9947871790453214
```

DTC, RFC and ABC have the same f1 score. Need to reduce the dimension.

Dementinality Reduction Using Recursive Feature Elimination.

RFE Logistic Regression

In [47]:

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y.values.ravel(), random_state=42, test_
X = xtrain
y = ytrain

n_feat = [8,10,12,15,20,25]
for n in n_feat:
    model = LogisticRegression(solver='warn')
    rfe = RFE(model, n_features_to_select = n)
    rfe = rfe.fit(X, y)
    cols = xtrain.columns.tolist()
    sel_feat = pd.DataFrame({"cols": cols, "support": rfe.support_, "rank": rfe.ranking_})
    print("Top features: ", n)
    print(rfe.score(X, y))
    print(f1_score(ytest, rfe.predict(xtest)))
    print(sel_feat[sel_feat['rank'] == 1]['cols'].unique(), '\n')
```

```
Top features:  8
0.9948301612223797
0.05797101449275362
['high_x' 'medium_y' 'high_y' 'high' 2.0 'Mid Low' 'Medium' 'Mid High']

Top features:  10
0.9948301612223797
0.05797101449275362
['high_x' 'medium_y' 'high_y' 'Medium_x' 'Mid High_x' 'high' 2.0 'Mid Low'
 'Medium' 'Mid High']

Top features:  12
0.9948301612223797
0.05797101449275362
['high_x' 'medium_y' 'high_y' 'Medium_x' 'Mid High_x'
 'Sanchez, Hays and Wilkerson' 'high' 2.0 'Medium_y' 'Mid Low' 'Medium'
 'Mid High']

Top features:  15
0.9948110136713514
0.05797101449275362
['medium_x' 'high_x' 'medium_y' 'high_y' 'Medium_x' 'Mid High_x'
 'Chapman-Mcmahon' 'Sanchez, Hays and Wilkerson' 'medium' 'high' 2.0
 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  20
0.9948205874468655
0.05797101449275362
['medium_x' 'high_x' 'medium_y' 'high_y' 'Medium_x' 'Mid High_x'
 'Chapman-Mcmahon' 'Edwards-Hoffman' 'Sanchez, Hays and Wilkerson'
 'Sanchez-Robinson' 'Suarez Inc' 'Taylor, Hunt and Rodriguez' 'medium'
 'high' 2.0 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  25
0.9948205874468655
0.05797101449275362
['medium_x' 'high_x' 'medium_y' 'high_y' 'Medium_x' 'Mid High_x'
 'Browning-Hart' 'Chapman-Mcmahon' 'Edwards-Hoffman' 'Richards-Walters'
 'Richardson Ltd' 'Sanchez, Hays and Wilkerson' 'Sanchez-Robinson'
 'Suarez Inc' 'Taylor, Hunt and Rodriguez' 'medium' 'high' 2.0 'Y' 'Z'
 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']
```

In [48]:

```
n_feat = [8,10,12,15,20,25]
for n in n_feat:
    model = RandomForestClassifier()
    rfe = RFE(model, n_features_to_select = n)
    rfe = rfe.fit(X, y)
    cols = X.columns.tolist()
    sel_feat = pd.DataFrame({"cols": cols, "support": rfe.support_, "rank": rfe.ranking_})
    print("Top features: ", n)
    print(rfe.score(X, y))
    print(f1_score(ytest, rfe.predict(xtest)))
    print(sel_feat[sel_feat['rank'] == 1]['cols'].unique(), '\n')
```

```
Top features:  8
0.994945046528549
0.11267605633802817
['medium_y' 'Mid High_x' 'Browning-Hart' 2.0 'Mid Low_y' 'Mid Low'
 'Medium' 'Mid High']

Top features:  10
0.994945046528549
0.11267605633802817
['medium_x' 'Mid High_x' 'No Insurance' 'medium' 2.0 'Y' 'Mid Low_y'
 'Mid Low' 'Medium' 'Mid High']

Top features:  12
0.994945046528549
0.11267605633802817
['medium_y' 'Medium_x' 'Mid High_x' 'Browning-Hart' 'medium' 2.0 'Y' 'Z'
 'Mid Low_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  15
0.9949258989775208
0.11267605633802817
['Medium_x' 'Mid High_x' 'Browning-Hart' 'Edwards-Hoffman' 'OTHER'
 'No Insurance' 'medium' 'high' 2.0 'Y' 'Z' 'Mid Low_y' 'Mid Low' 'Medium'
 'Mid High']

Top features:  20
0.994945046528549
0.0821917808219178
['medium_y' 'Mid Low_x' 'Medium_x' 'Mid High_x' 'Browning-Hart'
 'Edwards-Hoffman' 'Martinez, Duffy and Bird' 'Nicholson Group' 'OTHER'
 'No Insurance' 'medium' 'high' 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y'
 'Mid Low' 'Medium' 'Mid High']

Top features:  25
0.9949641940795773
0.08571428571428572
['medium_y' 'Mid Low_x' 'Medium_x' 'Mid High_x' 'Browning-Hart'
 'Cole, Brooks and Vincent' 'Edwards-Hoffman' 'Martinez, Duffy and Bird'
 'Miller, Mcclure and Allen' 'Nicholson Group' 'OTHER' 'Suarez Inc'
 'Swanson, Newton and Miller' 'Turner, Baldwin and Rhodes' 'No Insurance'
 'medium' 'high' 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium'
 'Mid High']
```

In [49]:

```
n_feat = [6,7,8,10,12,15,18,20,23,25]
for n in n_feat:
    model = DecisionTreeClassifier()
    rfe = RFE(model, n_features_to_select = n)
    rfe = rfe.fit(X, y)
    cols = X.columns.tolist()
    sel_feat = pd.DataFrame({"cols": cols, "support": rfe.support_, "rank": rfe.ranking_})
    print("Top features: ", n)
    print(rfe.score(X, y))
    print(f1_score(ytest, rfe.predict(xtest)))
    print(sel_feat[sel_feat['rank'] == 1]['cols'].unique(), '\n')
```

```
Top features:  6
0.9949258989775208
0.11267605633802817
['medium_x' 'Mid High_x' 'Mid Low_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  7
0.9949354727530348
0.11267605633802817
['medium_y' 'Mid High_x' 2.0 'Mid Low_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  8
0.9949354727530348
0.11267605633802817
['medium_y' 'Mid High_x' 2.0 'Z' 'Mid Low_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  10
0.994945046528549
0.11267605633802817
['medium_y' 'Mid High_x' 'Edwards-Hoffman' 'OTHER' 2.0 'Z' 'Mid Low_y'
 'Mid Low' 'Medium' 'Mid High']

Top features:  12
0.994945046528549
0.11267605633802817
['medium_x' 'Mid High_x' 'Edwards-Hoffman' 'OTHER' 'No Insurance' 'medium'
 2.0 'Z' 'Mid Low_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  15
0.9949546203040631
0.11267605633802817
['medium_y' 'Mid High_x' 'Edwards-Hoffman' 'Nicholson Group' 'OTHER'
 'No Insurance' 'medium' 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low'
 'Medium' 'Mid High']

Top features:  18
0.9949546203040631
0.11267605633802817
['medium_x' 'Medium_x' 'Mid High_x' 'Browning-Hart' 'Edwards-Hoffman'
 'Nicholson Group' 'OTHER' 'Turner, Baldwin and Rhodes' 'No Insurance'
 'medium' 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  20
0.9949546203040631
0.11267605633802817
['medium_y' 'Mid Low_x' 'Mid High_x' 'Browning-Hart' 'Edwards-Hoffman'
 'Nicholson Group' 'OTHER' 'Suarez Inc' 'Swanson, Newton and Miller'
```

```
 'Turner, Baldwin and Rhodes' 'No Insurance' 'medium' 2.0 'Y' 'Z'
 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  23
0.9949737678550913
0.11267605633802817
['medium_y' 'Medium_x' 'Mid High_x' 'Browning-Hart'
 'Cole, Brooks and Vincent' 'Edwards-Hoffman' 'Nicholson Group' 'OTHER'
 'Suarez Inc' 'Swanson, Newton and Miller' 'Taylor, Hunt and Rodriguez'
 'Turner, Baldwin and Rhodes' 'No Insurance' 'medium' 'high' 2.0 'Y' 'Z'
 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  25
0.9949737678550913
0.11267605633802817
['medium_y' 'Mid Low_x' 'Medium_x' 'Mid High_x' 'Browning-Hart'
 'Cole, Brooks and Vincent' 'Edwards-Hoffman' 'Martinez, Duffy and Bird'
 'Nicholson Group' 'OTHER' 'Suarez Inc' 'Swanson, Newton and Miller'
 'Taylor, Hunt and Rodriguez' 'Turner, Baldwin and Rhodes' 'No Insurance'
 'medium' 'high' 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium'
 'Mid High']
```

In [50]:

```
n_feat = [8,10,12,15,20,23,25,27,30]
for n in n_feat:
    model = AdaBoostClassifier()
    rfe = RFE(model, n_features_to_select = n)
    rfe = rfe.fit(X, y)
    cols = X.columns.tolist()
    sel_feat = pd.DataFrame({"cols": cols, "support": rfe.support_, "rank": rfe.ranking_})
    print("Top features: ", n)
    print(rfe.score(X, y))
    print(f1_score(ytest, rfe.predict(xtest)))
    print(sel_feat[sel_feat['rank'] == 1]['cols'].unique(), '\n')
```

```
Top features:  8
0.9948301612223797
0.05797101449275362
['Mid Low_x' 'Medium_x' 'Mid High_x' 2.0 'Y' 'Z' 'Mid Low_y' 'Mid Low']

Top features:  10
0.9948301612223797
0.05797101449275362
['Mid Low_x' 'Medium_x' 'Mid High_x' 'OTHER' 2.0 'Y' 'Z' 'Mid Low_y'
 'Medium_y' 'Mid Low']

Top features:  12
0.9948301612223797
0.05797101449275362
['Mid Low_x' 'Medium_x' 'Mid High_x' 'OTHER' 'medium' 'high' 2.0 'Y' 'Z'
 'Mid Low_y' 'Medium_y' 'Mid Low']

Top features:  15
0.9948301612223797
0.05797101449275362
['Mid Low_x' 'Medium_x' 'Mid High_x' 'Edwards-Hoffman'
 'Martinez, Duffy and Bird' 'OTHER' 'Richards-Walters' 'medium' 'high' 2.0
 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low']

Top features:  20
0.9948971776509784
0.11267605633802817
['Mid Low_x' 'Medium_x' 'Mid High_x' 'Chapman-Mcmahon' 'Edwards-Hoffman'
 'Martinez, Duffy and Bird' 'OTHER' 'Richards-Walters'
 'Sanchez, Hays and Wilkerson' 'Sanchez-Robinson' 'medium' 'high' 2.0 'Y'
 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  23
0.9948971776509784
0.11267605633802817
['Mid Low_x' 'Medium_x' 'Mid High_x' 'Chapman-Mcmahon' 'Edwards-Hoffman'
 'Martinez, Duffy and Bird' 'OTHER' 'Richards-Walters' 'Richardson Ltd'
 'Sanchez, Hays and Wilkerson' 'Sanchez-Robinson'
 'Taylor, Hunt and Rodriguez' 'Turner, Baldwin and Rhodes' 'medium' 'high'
 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  25
0.9948971776509784
0.11267605633802817
['Mid Low_x' 'Medium_x' 'Mid High_x' 'Browning-Hart' 'Chapman-Mcmahon'
 'Edwards-Hoffman' 'Martinez, Duffy and Bird' 'OTHER' 'Richards-Walters'
```

```
 'Richardson Ltd' 'Sanchez, Hays and Wilkerson' 'Sanchez-Robinson'
 'Suarez Inc' 'Taylor, Hunt and Rodriguez' 'Turner, Baldwin and Rhodes'
 'medium' 'high' 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium'
 'Mid High']

Top features:  27
0.9948876038754644
0.11267605633802817
['medium_x' 'Mid Low_x' 'Medium_x' 'Mid High_x' 'Browning-Hart'
 'Chapman-Mcmahon' 'Edwards-Hoffman' 'Martinez, Duffy and Bird' 'OTHER'
 'Richards-Walters' 'Richardson Ltd' 'Romero, Woods and Johnson'
 'Sanchez, Hays and Wilkerson' 'Sanchez-Robinson' 'Suarez Inc'
 'Taylor, Hunt and Rodriguez' 'Turner, Baldwin and Rhodes' 'medium' 'high'
 2.0 'Y' 'Z' 'Mid Low_y' 'Medium_y' 'Mid Low' 'Medium' 'Mid High']

Top features:  30
0.9948876038754644
0.11267605633802817
['medium_y' 'Mid Low_x' 'Medium_x' 'Mid High_x' 'Browning-Hart'
 'Chapman-Mcmahon' 'Edwards-Hoffman' 'Martinez, Duffy and Bird'
 'Miller, Mcclure and Allen' 'Nicholson Group' 'OTHER' 'Richards-Walters'
 'Richardson Ltd' 'Romero, Woods and Johnson'
 'Sanchez, Hays and Wilkerson' 'Sanchez-Robinson' 'Suarez Inc'
 'Swanson, Newton and Miller' 'Taylor, Hunt and Rodriguez'
 'Turner, Baldwin and Rhodes' 'medium' 'high' 2.0 'Y' 'Z' 'Mid Low_y'
 'Medium_y' 'Mid Low' 'Medium' 'Mid High']
```

In [51]:

```
cols = ['medium_x','Medium_x' ,'Mid High_x' ,'Browning-Hart' ,'Edwards-Hoffman',
 'Martinez, Duffy and Bird' ,'Nicholson Group' ,'OTHER',
 'Swanson, Newton and Miller' ,'No Insurance' ,'medium_y' ,'high_y' ,2.0 ,'Y',
 'Z' ,'Mid Low_y' ,'Medium_y' ,'Mid Low' ,'Medium' ,'Mid High']
```

In [55]:

```
x = x[cols]
y = y
x1 = x1[cols]
y1 = y1



def simple_model(alg):

    # splitting data into training and validation set
    xtrain, xtest, ytrain, ytest = train_test_split(x, y.values.ravel(), random_state=42, t
    model = alg
    model.fit(xtrain, ytrain) # training the model
#     prediction = model.predict_proba(xtest) # predicting on the validation set
#     prediction_int = prediction[:,1] >= 0.3 # if prediction is greater than or equal to 0
#     prediction_int = prediction_int.astype(np.int)

    print("f1_score:",f1_score(ytest, model.predict(xtest))) # calculating f1 score
    print("Accuracy on train data:",model.score(xtrain,ytrain))
    print("Accuracy on test data:",model.score(xtest,ytest))
```

In [56]:

```
for alg,l in zip(algs,algs_lst):
    print(l)
    simple_model(alg=alg)
```

```
LR
f1_score: 0.07518796992481203
Accuracy on train data: 0.994819378325399
Accuracy on test data: 0.9947010167154919
DTC
f1_score: 0.1037037037037037
Accuracy on train data: 0.9949486246041833
Accuracy on test data: 0.9947871790453214
RFC
f1_score: 0.1037037037037037
Accuracy on train data: 0.9949378540809513
Accuracy on test data: 0.9947871790453214
ABC
f1_score: 0.11764705882352941
Accuracy on train data: 0.9948732309415591
Accuracy on test data: 0.9948302602102361
```

In [57]:

```
from sklearn.metrics import confusion_matrix
xtrain, xtest, ytrain, ytest = train_test_split(x, y.values.ravel(), random_state=42, test_
model = AdaBoostClassifier()
model.fit(xtrain, ytrain)
ypred = model.predict(xtest)
confusion_matrix = confusion_matrix(ytest,ypred)
print(confusion_matrix)
```

```
[[23084     0]
 [  120     8]]
```

In [58]:

```
he result of the confusion matrix shows that out of 128 cases, 8 was predicted correctly and
```

In [59]:

```
from sklearn.metrics import classification_report
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

         0.0       0.99      1.00      1.00     23084
         1.0       1.00      0.06      0.12       128

    accuracy                           0.99     23212
   macro avg       1.00      0.53      0.56     23212
weighted avg       0.99      0.99      0.99     23212
```

In [60]:

```
# Precision Recall and F1 score of 0 is high, but for 0 is too low. This shows that data is
```

In [62]:

```
xtrain = x[cols]
ytrain = y
xtest = x1[cols]
ytest = y1
```

In [71]:

```
abbs = AdaBoostClassifier()
abbs.fit(xtrain,ytrain)
pred = abbs.predict(xtest)
```

In [72]:

```
pred
```

Out[72]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [64]:

```
ytest.m13.value_counts()
```

Out[64]:

```
0.0    35853
1.0       13
Name: m13, dtype: int64
```

In [73]:

```
submission = pd.DataFrame({'loan_id':test_loan_id,'m13':pred})
```

In [75]:

```
submission.to_csv('Submission.csv',index=False)
```

In [76]:

```
submission
```

...

In [ ]: