In [1]:
```
import pandas as pd
import numpy as np
import pandas as pd
import os

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sco
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE

from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn import tree
import xgboost as xgb
from numpy import loadtxt
from xgboost import XGBClassifier
from xgboost import plot_tree

import gc
from tqdm import tqdm
```

executed in 3.31s, finished 01:44:09 2021-04-21

In [2]:
```
column_defs = pd.read_excel('data\LCDataDictionary.xlsx',index_col='LoanStatNew')
column_defs.columns
```

executed in 77ms, finished 01:44:09 2021-04-21

Out[2]:
```
Index(['Description'], dtype='object')
```

In [3]:
```
column_defs
```
executed in 20ms, finished 01:44:15 2021-04-21

Out[3]:

LoanStatNew	Description
acc_now_delinq	The number of accounts on which the borrower i...
acc_open_past_24mths	Number of trades opened in past 24 months.
addr_state	The state provided by the borrower in the loan...
all_util	Balance to credit limit on all trades
annual_inc	The self-reported annual income provided by th...
...	...
settlement_amount	The loan amount that the borrower has agreed t...
settlement_percentage	The settlement amount as a percentage of the p...
settlement_term	The number of months that the borrower will be...
NaN	NaN
NaN	* Employer Title replaces Employer Name for al...

153 rows × 1 columns

In [4]:
```
def column_info(col_name):
    return column_defs.loc[col_name]['Description']
```
executed in 15ms, finished 01:44:29 2021-04-21

In [24]:
```
cols_to_drop
```

executed in 13ms, finished 01:56:24 2021-04-21

Out[24]:

```
136                          hardship_loan_status
127                               hardship_reason
128                               hardship_status
135                                  hardship_dpd
133                        payment_plan_start_date
126                                  hardship_type
129                                  deferral_term
131                             hardship_start_date
132                               hardship_end_date
134                               hardship_length
137    orig_projected_additional_accrued_interest
130                               hardship_amount
138                hardship_payoff_balance_amount
139                  hardship_last_payment_amount
120                            sec_app_revol_util
57                       verification_status_joint
113                              revol_bal_joint
119                            sec_app_open_acc
116                       sec_app_earliest_cr_line
118                              sec_app_mort_acc
117                          sec_app_inq_last_6mths
123              sec_app_chargeoff_within_12_mths
124            sec_app_collections_12_mths_ex_med
122                           sec_app_num_rev_accts
115                         sec_app_fico_range_high
114                         sec_app_fico_range_low
121                            sec_app_open_act_il
56                                      dti_joint
55                                annual_inc_joint
29                          mths_since_last_record
88                        mths_since_recent_bc_dlq
52                     mths_since_last_major_derog
90                   mths_since_recent_revol_delinq
47                                   next_pymnt_d
28                          mths_since_last_delinq
67                                       il_util
65                             mths_since_rcnt_il
71                                       all_util
61                                    open_acc_6m
74                                    total_cu_tl
75                                  inq_last_12m
62                                    open_act_il
63                                    open_il_12m
64                                    open_il_24m
66                                   total_bal_il
68                                    open_rv_12m
69                                    open_rv_24m
73                                         inq_fi
70                                     max_bal_bc
Name: index, dtype: object
```

In [25]:
```
len(columns_to_drop)
```
executed in 20ms, finished 01:57:09 2021-04-21

Out[25]: 67

In [5]:
```
columns_to_drop = ['mths_since_last_delinq',
 'mths_since_last_record',
 'next_pymnt_d',
 'mths_since_last_major_derog',
 'annual_inc_joint',
 'dti_joint',
 'verification_status_joint',
 'mths_since_rcnt_il',
 'il_util',
 'mths_since_recent_bc_dlq',
 'mths_since_recent_revol_delinq',
 'revol_bal_joint',
 'sec_app_fico_range_low',
 'sec_app_fico_range_high',
 'sec_app_earliest_cr_line',
 'sec_app_inq_last_6mths',
 'sec_app_mort_acc',
 'sec_app_open_acc',
 'sec_app_revol_util',
 'sec_app_open_act_il',
 'sec_app_num_rev_accts',
 'sec_app_chargeoff_within_12_mths',
 'sec_app_collections_12_mths_ex_med',
 'hardship_type',
 'hardship_reason',
 'hardship_status',
 'deferral_term',
 'hardship_amount',
 'hardship_start_date',
 'hardship_end_date',
 'payment_plan_start_date',
 'hardship_length',
 'hardship_dpd',
 'hardship_loan_status',
 'orig_projected_additional_accrued_interest',
 'hardship_payoff_balance_amount',
 'hardship_last_payment_amount','policy_code','url','out_prncp_inv','total_pymnt_
              'num_sats','tot_hi_cred_lim','num_rev_tl_bal_gt_0','collection_recc
             ,'bc_open_to_buy','percent_bc_gt_75','open_acc','num_actv_bc_tl','nu
             ,'total_rev_hi_lim','all_util', 'inq_last_12m', 'total_cu_tl', 'oper
       'open_rv_12m', 'open_act_il', 'open_il_24m', 'total_bal_il',
       'open_il_12m', 'open_rv_24m', 'max_bal_bc', 'inq_fi']
```
executed in 10ms, finished 01:44:30 2021-04-21

In [26]:
```
len(corr_drops)
```
executed in 9ms, finished 01:57:14 2021-04-21

Out[26]: 15

In [6]:
```
corr_drops = ['out_prncp_inv','total_pymnt_inv','funded_amnt_inv','funded_amnt',
              'num_sats','tot_hi_cred_lim','num_rev_tl_bal_gt_0','collection_recc
             ,'bc_open_to_buy','percent_bc_gt_75','open_acc','num_actv_bc_tl','nu
             ,'total_rev_hi_lim']
```

executed in 16ms, finished 01:44:33 2021-04-21

In [7]:
```
df = pd.read_csv('data\Loan_status_2007-2020Q3.gzip',low_memory=False)
```

executed in 2m 22s, finished 01:47:01 2021-04-21

In [8]:
```
df.head()
```

executed in 35ms, finished 01:47:41 2021-04-21

Out[8]:

	Unnamed: 0	id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	ins
0	0	1077501	5000.0	5000.0	4975.0	36 months	10.65%	
1	1	1077430	2500.0	2500.0	2500.0	60 months	15.27%	
2	2	1077175	2400.0	2400.0	2400.0	36 months	15.96%	
3	3	1076863	10000.0	10000.0	10000.0	36 months	13.49%	
4	4	1075358	3000.0	3000.0	3000.0	60 months	12.69%	

5 rows × 142 columns

In [9]:

```
def reduce_mem_usage(df, int_cast=True, obj_to_category=False, subset=None):
    """
    Iterate through all the columns of a dataframe and modify the data type to re
    :param df: dataframe to reduce (pd.DataFrame)
    :param int_cast: indicate if columns should be tried to be casted to int (boo
    :param obj_to_category: convert non-datetime related objects to category dtyp
    :param subset: subset of columns to analyse (list)
    :return: dataset with the column dtypes adjusted (pd.DataFrame)
    """
    start_mem = df.memory_usage().sum() / 1024 ** 2;
    gc.collect()
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    cols = subset if subset is not None else df.columns.tolist()

    for col in tqdm(cols):
        col_type = df[col].dtype

        if col_type != object and col_type.name != 'category' and 'datetime' not
            c_min = df[col].min()
            c_max = df[col].max()

            # test if column can be converted to an integer
            treat_as_int = str(col_type)[:3] == 'int'
            if int_cast and not treat_as_int:
                treat_as_int = check_if_integer(df[col])

            if treat_as_int:
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).ma
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.uint8).min and c_max < np.iinfo(np.uint8
                    df[col] = df[col].astype(np.uint8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.uint16).min and c_max < np.iinfo(np.uint
                    df[col] = df[col].astype(np.uint16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.uint32).min and c_max < np.iinfo(np.uint
                    df[col] = df[col].astype(np.uint32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64
                    df[col] = df[col].astype(np.int64)
                elif c_min > np.iinfo(np.uint64).min and c_max < np.iinfo(np.uint
                    df[col] = df[col].astype(np.uint64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.flo
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        elif 'datetime' not in col_type.name and obj_to_category:
            df[col] = df[col].astype('category')
    gc.collect()
    end_mem = df.memory_usage().sum() / 1024 ** 2
    print('Memory usage after optimization is: {:.3f} MB'.format(end_mem))
```

```
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df
```

executed in 42ms, finished 01:47:50 2021-04-21

In [10]:
```
#df.drop(columns=columns_to_drop,axis=1,inplace=True)
```

executed in 1.24s, finished 21:57:29 2021-04-01

In [10]:
```
reduce_mem_usage(df,int_cast=False)
```

executed in 1m 14.9s, finished 01:49:18 2021-04-21

```
  0%|          | 0/142 [00:00<?, ?it/s]

Memory usage of dataframe is 3169.40 MB

100%|██████████| 142/142 [01:12<00:00,  1.96it/s]

Memory usage after optimization is: 1484.263 MB
Decreased by 53.2%
```

Out[10]:

	Unnamed: 0	id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rat
0	0	1077501	5000.0	5000.0	4976.0	36 months	10.6
1	1	1077430	2500.0	2500.0	2500.0	60 months	15.2
2	2	1077175	2400.0	2400.0	2400.0	36 months	15.9
3	3	1076863	10000.0	10000.0	10000.0	36 months	13.4
4	4	1075358	3000.0	3000.0	3000.0	60 months	12.6
...	...	...	...	...	...	...	
2925488	105446	102556443	24000.0	24000.0	24000.0	60 months	23.9
2925489	105447	102653304	10000.0	10000.0	10000.0	36 months	7.9
2925490	105448	102628603	10048.0	10048.0	10048.0	36 months	16.9
2925491	105449	102196576	6000.0	6000.0	6000.0	36 months	11.4
2925492	105450	99799684	30000.0	30000.0	30000.0	60 months	25.4

2925493 rows × 142 columns

In [11]:
```
df.info(verbose=True)
```

executed in 32ms, finished 01:49:25 2021-04-21

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2925493 entries, 0 to 2925492
Data columns (total 142 columns):
 #   Column                                      Dtype
---  ------                                      -----
 0   Unnamed: 0                                  int32
 1   id                                          object
 2   loan_amnt                                   float16
 3   funded_amnt                                 float16
 4   funded_amnt_inv                             float16
 5   term                                        object
 6   int_rate                                    object
 7   installment                                 float16
 8   grade                                       object
 9   sub_grade                                   object
 10  emp_title                                   object
 11  emp_length                                  object
 12  home_ownership                              object
 13  annual_inc                                  float32
```

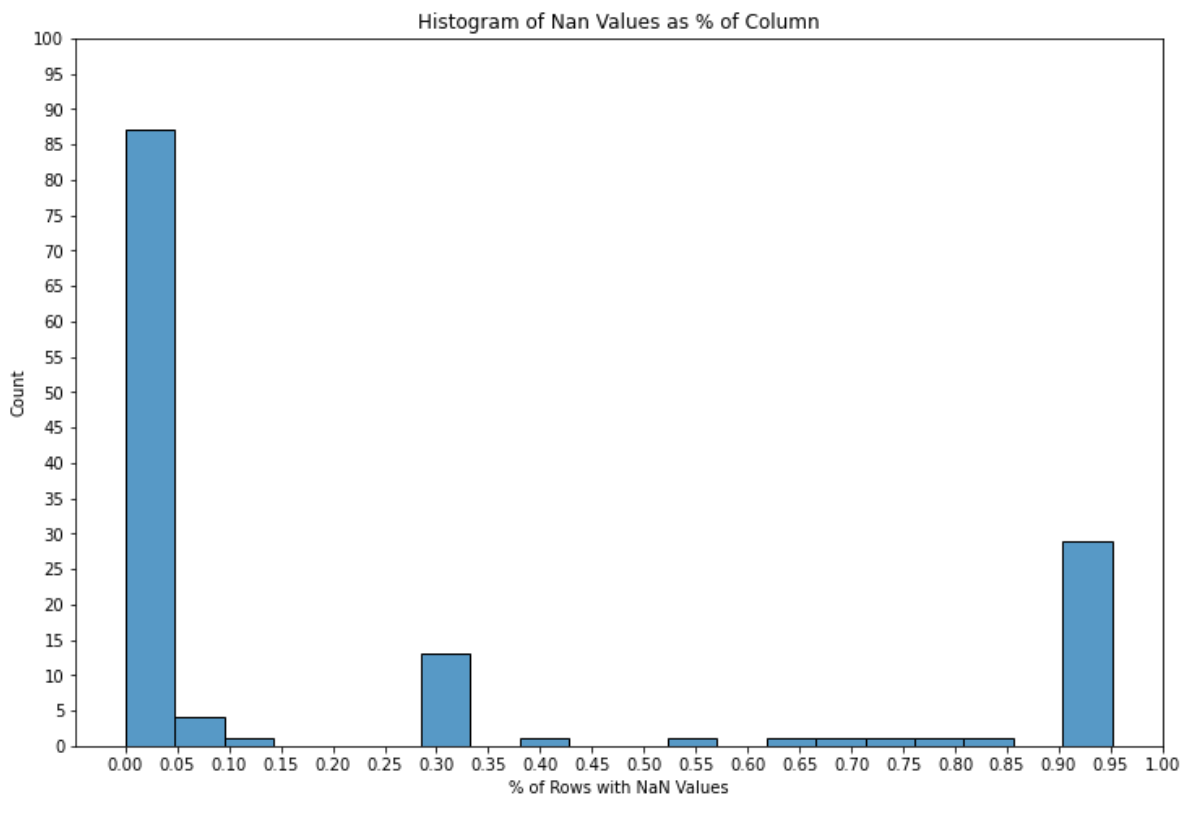
In [12]:
```
df.drop('Unnamed: 0',axis=1,inplace=True)
```

executed in 2.67s, finished 01:49:54 2021-04-21

In [13]:
```
fig,ax = plt.subplots(figsize=(12,8))
sns.histplot(data=df.isna().mean(),bins=20)
plt.xticks(np.linspace(0,1,21))
plt.title("Histogram of Nan Values as % of Column")
plt.xlabel('% of Rows with NaN Values')
plt.yticks(np.linspace(0,100,21))
```

executed in 5.55s, finished 01:50:19 2021-04-21

Out[13]:
```
([<matplotlib.axis.YTick at 0x26c8003fa00>,
  <matplotlib.axis.YTick at 0x26c8003f5e0>,
  <matplotlib.axis.YTick at 0x26c800d2790>,
  <matplotlib.axis.YTick at 0x26c800d2c70>,
  <matplotlib.axis.YTick at 0x26c800d71c0>,
  <matplotlib.axis.YTick at 0x26c800d76d0>,
  <matplotlib.axis.YTick at 0x26c80034430>,
  <matplotlib.axis.YTick at 0x26c800e9370>,
  <matplotlib.axis.YTick at 0x26c800e9880>,
  <matplotlib.axis.YTick at 0x26c800e9d90>,
  <matplotlib.axis.YTick at 0x26c800ee2e0>,
  <matplotlib.axis.YTick at 0x26c800e9910>,
  <matplotlib.axis.YTick at 0x26c800e3910>,
  <matplotlib.axis.YTick at 0x26c800de490>,
  <matplotlib.axis.YTick at 0x26c800ee9a0>,
  <matplotlib.axis.YTick at 0x26c800eeeb0>,
  <matplotlib.axis.YTick at 0x26c800f6400>,
  <matplotlib.axis.YTick at 0x26c800f6910>,
  <matplotlib.axis.YTick at 0x26c800f6e20>,
  <matplotlib.axis.YTick at 0x26c800fb370>,
  <matplotlib.axis.YTick at 0x26c800f6a90>],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



Roughly 50 columns missing 25% of values in the columns will drop these

In [18]:
```
df.isna().mean().reset_index().sort_values(by=0,ascending=False)
```

executed in 4.84s, finished 01:53:12 2021-04-21

Out[18]:

	index	0
136	hardship_loan_status	9.509789e-01
127	hardship_reason	9.509033e-01
128	hardship_status	9.509023e-01
135	hardship_dpd	9.509016e-01
133	payment_plan_start_date	9.509013e-01
...	...	...
38	total_pymnt	3.418227e-07
37	out_prncp_inv	3.418227e-07
36	out_prncp	3.418227e-07
35	initial_list_status	3.418227e-07
0	id	0.000000e+00

141 rows × 2 columns

In [19]:
```
nan_pct = df.isna().mean().reset_index().sort_values(by=0,ascending=False)
```

executed in 5.06s, finished 01:54:40 2021-04-21

In [21]:
```
nan_pct[nan_pct[0] >=.25]['index'], print(len(nan_pct[nan_pct[0] >=.25]['index']))
```

executed in 16ms, finished 01:55:12 2021-04-21

```
49
```

Out[21]:

```
(136                          hardship_loan_status
 127                               hardship_reason
 128                               hardship_status
 135                                  hardship_dpd
 133                       payment_plan_start_date
 126                                 hardship_type
 129                                 deferral_term
 131                            hardship_start_date
 132                              hardship_end_date
 134                              hardship_length
 137    orig_projected_additional_accrued_interest
 130                              hardship_amount
 138                hardship_payoff_balance_amount
 139                  hardship_last_payment_amount
 120                            sec_app_revol_util
 57                       verification_status_joint
 113                              revol_bal_joint
 119                            sec_app_open_acc
 116                     sec_app_earliest_cr_line
 118                            sec_app_mort_acc
 117                       sec_app_inq_last_6mths
 123              sec_app_chargeoff_within_12_mths
 124            sec_app_collections_12_mths_ex_med
 122                         sec_app_num_rev_accts
 115                       sec_app_fico_range_high
 114                        sec_app_fico_range_low
 121                           sec_app_open_act_il
 56                                      dti_joint
 55                              annual_inc_joint
 29                        mths_since_last_record
 88                      mths_since_recent_bc_dlq
 52                    mths_since_last_major_derog
 90                  mths_since_recent_revol_delinq
 47                                   next_pymnt_d
 28                        mths_since_last_delinq
 67                                       il_util
 65                           mths_since_rcnt_il
 71                                     all_util
 61                                  open_acc_6m
 74                                  total_cu_tl
 75                                inq_last_12m
 62                                  open_act_il
 63                                  open_il_12m
 64                                  open_il_24m
 66                                total_bal_il
 68                                  open_rv_12m
 69                                  open_rv_24m
 73                                       inq_fi
 70                                   max_bal_bc
Name: index, dtype: object,
None)
```

In [27]:
```
cols_to_drop = nan_pct[nan_pct[0] >=.25]['index']
```
executed in 19ms, finished 01:58:15 2021-04-21

In [28]:
```
#Dropped features with 25% or more nan values
df.drop(columns=cols_to_drop,axis=1,inplace=True)
```
executed in 1.64s, finished 01:58:25 2021-04-21

In [29]:
```
#checking predictors for multicollinearity
test = df.corr().abs().stack().reset_index().sort_values(0,ascending=False)
test['pairs'] = list(zip(test.level_0,test.level_1))
test.set_index(['pairs'], inplace=True)
test.drop(columns=['level_1','level_0'], inplace=True)
test.columns = ['cc']
test.drop_duplicates(inplace=True)
test.sort_values('cc',ascending=False, inplace=True)
multicollinear_predictors = test[test.cc >.75]
multicollinear_predictors.reset_index(inplace=True)
```
executed in 45.3s, finished 01:59:30 2021-04-21

In [30]:
```
multicollinear_predictors['column_1'] = multicollinear_predictors.pairs.map(lambd
multicollinear_predictors['column_2'] = multicollinear_predictors.pairs.map(lambd
```
executed in 15ms, finished 02:02:15 2021-04-21

```
<ipython-input-30-a30768636eb9>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  multicollinear_predictors['column_1'] = multicollinear_predictors.pairs.map(l
ambda x : x[0])
<ipython-input-30-a30768636eb9>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  multicollinear_predictors['column_2'] = multicollinear_predictors.pairs.map(l
ambda x : x[1])
```

In [31]:
```
pd.options.display.max_rows = 40
```
executed in 14ms, finished 02:02:22 2021-04-21

In [32]:
```
multicollinear_predictors
```

executed in 45ms, finished 02:02:24 2021-04-21

Out[32]:

	pairs	cc	column_1	column_2
0	(loan_amnt, loan_amnt)	1.000000	loan_amnt	loan_amnt
1	(fico_range_low, fico_range_high)	1.000000	fico_range_low	fico_range_high
2	(out_prncp, out_prncp_inv)	0.999998	out_prncp	out_prncp_inv
3	(funded_amnt, loan_amnt)	0.999822	funded_amnt	loan_amnt
4	(total_pymnt_inv, total_pymnt)	0.999569	total_pymnt_inv	total_pymnt
5	(funded_amnt, funded_amnt_inv)	0.999520	funded_amnt	funded_amnt_inv
6	(loan_amnt, funded_amnt_inv)	0.999299	loan_amnt	funded_amnt_inv
7	(num_sats, open_acc)	0.999149	num_sats	open_acc
8	(num_actv_rev_tl, num_rev_tl_bal_gt_0)	0.986211	num_actv_rev_tl	num_rev_tl_bal_gt_0
9	(collection_recovery_fee, recoveries)	0.979642	collection_recovery_fee	recoveries
10	(tot_cur_bal, tot_hi_cred_lim)	0.976404	tot_cur_bal	tot_hi_cred_lim
11	(total_rec_prncp, total_pymnt)	0.966415	total_rec_prncp	total_pymnt
12	(total_pymnt_inv, total_rec_prncp)	0.965965	total_pymnt_inv	total_rec_prncp
13	(installment, funded_amnt)	0.943827	installment	funded_amnt
14	(installment, loan_amnt)	0.943581	installment	loan_amnt
15	(funded_amnt_inv, installment)	0.943187	funded_amnt_inv	installment
16	(total_bal_ex_mort, total_il_high_credit_limit)	0.882000	total_bal_ex_mort	total_il_high_credit_limit
17	(bc_open_to_buy, total_bc_limit)	0.852275	bc_open_to_buy	total_bc_limit
18	(bc_util, percent_bc_gt_75)	0.847521	bc_util	percent_bc_gt_75
19	(last_fico_range_high, last_fico_range_low)	0.847111	last_fico_range_high	last_fico_range_low
20	(num_op_rev_tl, open_acc)	0.842363	num_op_rev_tl	open_acc
21	(num_sats, num_op_rev_tl)	0.841691	num_sats	num_op_rev_tl
22	(num_bc_tl, num_rev_accts)	0.841267	num_bc_tl	num_rev_accts
23	(num_bc_sats, num_actv_bc_tl)	0.836619	num_bc_sats	num_actv_bc_tl
24	(num_actv_rev_tl, num_actv_bc_tl)	0.830364	num_actv_rev_tl	num_actv_bc_tl
25	(num_rev_tl_bal_gt_0, num_actv_bc_tl)	0.825230	num_rev_tl_bal_gt_0	num_actv_bc_tl
26	(avg_cur_bal, tot_cur_bal)	0.823918	avg_cur_bal	tot_cur_bal
27	(num_rev_tl_bal_gt_0, num_op_rev_tl)	0.803469	num_rev_tl_bal_gt_0	num_op_rev_tl
28	(acc_now_delinq, num_tl_30dpd)	0.801519	acc_now_delinq	num_tl_30dpd
29	(num_rev_accts, num_op_rev_tl)	0.800397	num_rev_accts	num_op_rev_tl
30	(num_op_rev_tl, num_actv_rev_tl)	0.800002	num_op_rev_tl	num_actv_rev_tl

	pairs	cc	column_1	column_2
31	(revol_bal, total_rev_hi_lim)	0.791658	revol_bal	total_rev_hi_lim
32	(tot_hi_cred_lim, avg_cur_bal)	0.782711	tot_hi_cred_lim	avg_cur_bal
33	(num_bc_sats, num_op_rev_tl)	0.780491	num_bc_sats	num_op_rev_tl
34	(total_rev_hi_lim, total_bc_limit)	0.779951	total_rev_hi_lim	total_bc_limit
35	(acc_open_past_24mths, num_tl_op_past_12m)	0.771788	acc_open_past_24mths	num_tl_op_past_12m
36	(total_acc, num_rev_accts)	0.764571	total_acc	num_rev_accts
37	(num_bc_sats, num_bc_tl)	0.763108	num_bc_sats	num_bc_tl

In [2]:
```
len(corr_drops)
```
executed in 26ms, finished 01:18:45 2021-04-22

Out[2]: 15

In [1]:
```
corr_drops = ['out_prncp_inv','total_pymnt_inv','funded_amnt_inv','funded_amnt',
              'num_sats','tot_hi_cred_lim','num_rev_tl_bal_gt_0','collection_recc
             ,'bc_open_to_buy','percent_bc_gt_75','open_acc','num_actv_bc_tl','nu
             ,'total_rev_hi_lim']
```
executed in 16ms, finished 01:18:40 2021-04-22

In [34]:
```
#Dropping columns that have high correlation (over .75) of which I deemed unneces
df.drop(columns=corr_drops,axis=1,inplace=True)
```
executed in 1.45s, finished 02:02:57 2021-04-21

Modify:

1. fico_low high take average
2. last_fico_range_high low average

Drop:

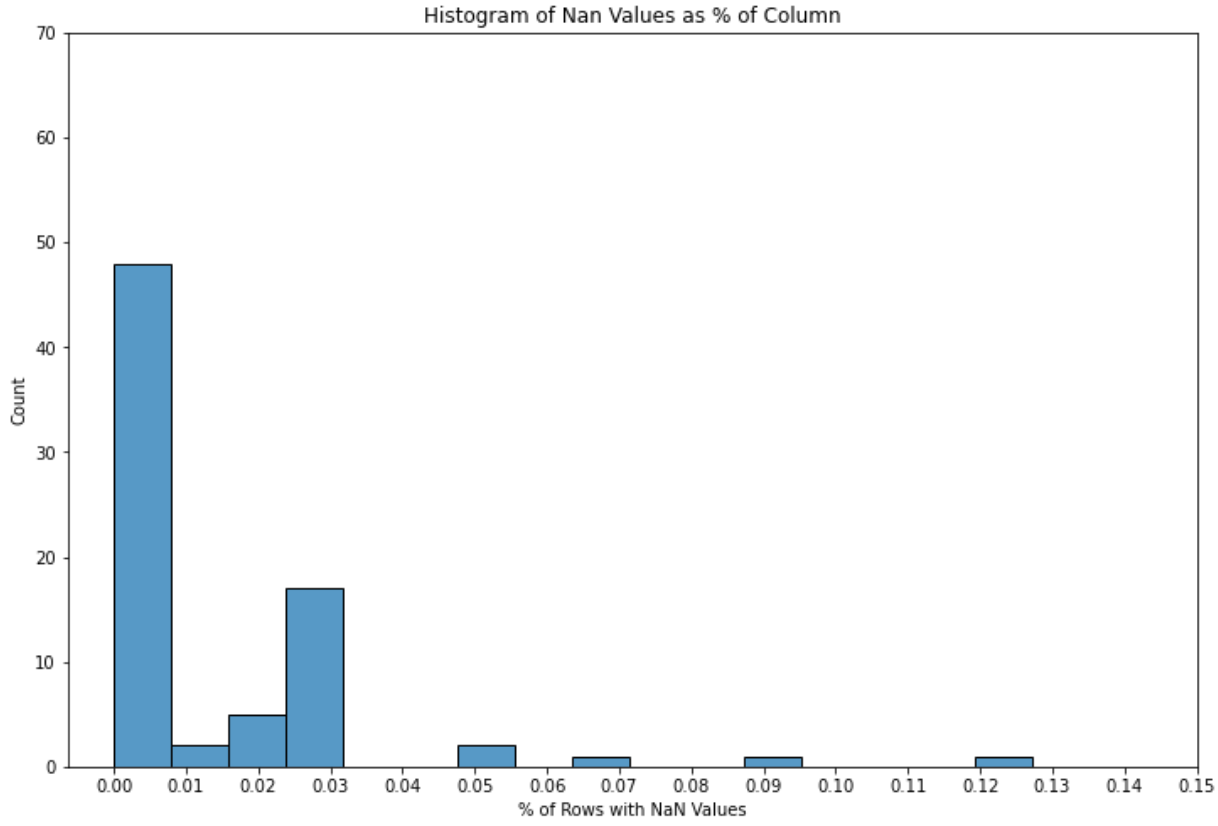
- out_prncp_inv
- total_pymnt_inv
- funded_amnt_inv
- funded_amnt these
- num_sats
- member_id
- num_rev_tl_bal_gt_0
- collection_recovery_fee
- bc_open_to_buy
- percent_bc_gt_75
- open_acc
- num_actv_bc_tl
- num_actv_rev_tl

- num_rev_accts
- tot_hi_cred_lim
- total_rev_hi_lim

In [35]:
```
fig,ax = plt.subplots(figsize=(12,8))
sns.histplot(data=df.isna().mean(),bins=16)
plt.xticks(np.linspace(0,.15,16))
plt.title("Histogram of Nan Values as % of Column")
plt.xlabel('% of Rows with NaN Values')
plt.yticks(np.linspace(0,70,8))
```

executed in 3.63s, finished 02:03:43 2021-04-21

Out[35]:
```
([<matplotlib.axis.YTick at 0x26c8146de20>,
  <matplotlib.axis.YTick at 0x26c8146f790>,
  <matplotlib.axis.YTick at 0x26c8143b1f0>,
  <matplotlib.axis.YTick at 0x26c8143b220>,
  <matplotlib.axis.YTick at 0x26c8143b250>,
  <matplotlib.axis.YTick at 0x26c8143b370>,
  <matplotlib.axis.YTick at 0x26c8143d280>,
  <matplotlib.axis.YTick at 0x26c81512580>],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, '')])
```



In [36]:
```
def na_check(data):
    check = np.round(data.isna().mean().sort_values(ascending=False),2)
    return check
```

executed in 15ms, finished 02:03:52 2021-04-21

1860331 of loans fully paid or charged off i.e. target variable

In [37]:
```
df.info()
```

executed in 20ms, finished 02:04:09 2021-04-21

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2925493 entries, 0 to 2925492
Data columns (total 77 columns):
 #   Column                         Dtype
---  ------                         -----
 0   id                             object
 1   loan_amnt                      float16
 2   term                           object
 3   int_rate                       object
 4   installment                    float16
 5   grade                          object
 6   sub_grade                      object
 7   emp_title                      object
 8   emp_length                     object
 9   home_ownership                 object
 10  annual_inc                     float32
 11  verification_status            object
 12  issue_d                        object
 13  loan_status                    object
 14  pymnt_plan                     object
 15  url                            object
 16  purpose                        object
 17  title                          object
 18  zip_code                       object
 19  addr_state                     object
 20  dti                            float16
 21  delinq_2yrs                    float16
 22  earliest_cr_line               object
 23  fico_range_low                 float16
 24  fico_range_high                float16
 25  inq_last_6mths                 float16
 26  pub_rec                        float16
 27  revol_bal                      float32
 28  revol_util                     object
 29  total_acc                      float16
 30  initial_list_status            object
 31  out_prncp                      float16
 32  total_pymnt                    float32
 33  total_rec_prncp                float16
 34  total_rec_int                  float16
 35  total_rec_late_fee             float16
 36  recoveries                     float16
 37  last_pymnt_d                   object
 38  last_pymnt_amnt                float16
 39  last_credit_pull_d             object
 40  last_fico_range_high           float16
 41  last_fico_range_low            float16
 42  collections_12_mths_ex_med     float16
 43  policy_code                    float16
 44  application_type               object
 45  acc_now_delinq                 float16
 46  tot_coll_amt                   float32
 47  tot_cur_bal                    float32
 48  acc_open_past_24mths           float16
```

```
 49  avg_cur_bal                 float32
 50  bc_util                     float16
 51  chargeoff_within_12_mths    float16
 52  delinq_amnt                 float32
 53  mo_sin_old_il_acct          float16
 54  mo_sin_old_rev_tl_op        float16
 55  mo_sin_rcnt_rev_tl_op       float16
 56  mo_sin_rcnt_tl              float16
 57  mort_acc                    float16
 58  mths_since_recent_bc        float16
 59  mths_since_recent_inq       float16
 60  num_accts_ever_120_pd       float16
 61  num_bc_sats                 float16
 62  num_bc_tl                   float16
 63  num_il_tl                   float16
 64  num_op_rev_tl               float16
 65  num_tl_120dpd_2m            float16
 66  num_tl_30dpd                float16
 67  num_tl_90g_dpd_24m          float16
 68  num_tl_op_past_12m          float16
 69  pct_tl_nvr_dlq              float16
 70  pub_rec_bankruptcies        float16
 71  tax_liens                   float16
 72  total_bal_ex_mort           float32
 73  total_bc_limit              float32
 74  total_il_high_credit_limit  float32
 75  hardship_flag               object
 76  debt_settlement_flag        object
dtypes: float16(42), float32(10), object(25)
memory usage: 903.9+ MB
```

In [41]:
```
len(df)
```
executed in 9ms, finished 02:10:05 2021-04-21

Out[41]:
```
2925493
```

In [40]:
```
len(df.loc[df['loan_status'].isin(['Fully Paid','Charged Off'])])
```
executed in 1.11s, finished 02:09:56 2021-04-21

Out[40]:
```
1860331
```

In [42]:
```
#our target variable is loan status and whether or not a client who received a lo
#will make a df with only these values so we can model
clean = df.loc[df['loan_status'].isin(['Fully Paid','Charged Off'])]
```
executed in 937ms, finished 02:11:07 2021-04-21

In [43]:
```
na_check(clean).head(20)
```
executed in 2.05s, finished 02:11:13 2021-04-21

Out[43]:
```
mths_since_recent_inq         0.12
num_tl_120dpd_2m              0.07
emp_title                     0.07
mo_sin_old_il_acct            0.07
emp_length                    0.06
bc_util                       0.04
pct_tl_nvr_dlq                0.04
avg_cur_bal                   0.04
mo_sin_rcnt_rev_tl_op         0.04
mo_sin_old_rev_tl_op          0.04
num_tl_30dpd                  0.04
num_accts_ever_120_pd         0.04
tot_cur_bal                   0.04
num_tl_90g_dpd_24m            0.04
total_il_high_credit_limit    0.04
num_tl_op_past_12m            0.04
mo_sin_rcnt_tl                0.04
num_bc_tl                     0.04
num_il_tl                     0.04
num_op_rev_tl                 0.04
dtype: float64
```

In [44]:
```
rem_nans = np.round(clean.isna().mean().sort_values(ascending=False),2)
five_pc_nan = list(rem_nans[(rem_nans>0) & (rem_nans <.05)].index)
```
executed in 2.29s, finished 02:12:54 2021-04-21

In [47]:
```
len(clean)
```
executed in 8ms, finished 02:13:29 2021-04-21

Out[47]:
```
1860331
```

In [48]:
```
clean.dropna(subset=five_pc_nan,inplace=True)
```
executed in 1.67s, finished 02:13:43 2021-04-21

```
<ipython-input-48-a53ae10695cc>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  clean.dropna(subset=five_pc_nan,inplace=True)
```

0.07% of rows dropped for nan values or 119,268 rows

In [50]:
```
1860331 - len(clean)
```
executed in 6ms, finished 02:14:07 2021-04-21

Out[50]:
```
119268
```

In [20]:
```
(na_check(clean)).head(10)
```

executed in 1.30s, finished 21:59:25 2021-04-01

Out[20]:

```
mths_since_recent_inq    0.10
emp_title                0.07
emp_length               0.06
num_tl_120dpd_2m         0.04
mo_sin_old_il_acct       0.03
last_pymnt_d             0.00
dti                      0.00
last_credit_pull_d       0.00
zip_code                 0.00
inq_last_6mths           0.00
dtype: float64
```

In [51]:
```
#exporting as csv and finishing in another notebook as the sheer size of the file
clean.to_csv('data\cleaned_data')
```

executed in 1m 55.9s, finished 02:17:01 2021-04-21