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 1.94s, finished 02:20:02 2021-04-21

In [2]:
```
def column_info(col_name):
    return column_defs.loc[col_name]['Description']
```

executed in 14ms, finished 02:20:02 2021-04-21

In [3]:
```
def na_check(data):
    check = np.round(data.isna().mean().sort_values(ascending=False),2)
    return check
```

executed in 14ms, finished 02:20:02 2021-04-21

In [4]:
```
column_defs = pd.read_excel('data\LCDataDictionary.xlsx',index_col='LoanStatNew')
column_defs.columns
```

executed in 45ms, finished 02:20:02 2021-04-21

Out[4]:
```
Index(['Description'], dtype='object')
```

In [5]:

```
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 18ms, finished 02:20:03 2021-04-21

In [6]:
```
df = pd.read_csv('data/cleaned_data')
```

executed in 12.5s, finished 02:20:18 2021-04-21

In [7]:
```
reduce_mem_usage(df,int_cast=False)
```

executed in 8.93s, finished 02:20:27 2021-04-21

```
  0%|          | 0/78 [00:00<?, ?it/s]

Memory usage of dataframe is 1036.09 MB

100%|██████████| 78/78 [00:08<00:00,  9.57it/s]

Memory usage after optimization is: 537.972 MB
Decreased by 48.1%
```

Out[7]:

	Unnamed: 0	id	loan_amnt	term	int_rate	installment	grade	sub_g
0	42536	10129454	12000.0	36 months	10.99%	392.750	B	
1	42537	10149488	4800.0	36 months	10.99%	157.125	B	
2	42538	10149342	27056.0	36 months	10.99%	885.500	B	
3	42539	10148122	12000.0	36 months	7.62%	374.000	A	
4	42540	10129477	14000.0	36 months	12.85%	470.750	B	
...	...	...	...	...	...	...	...	...
1741058	2925488	102556443	24000.0	60 months	23.99%	690.500	E	
1741059	2925489	102653304	10000.0	36 months	7.99%	313.250	A	
1741060	2925490	102628603	10048.0	36 months	16.99%	358.250	D	
1741061	2925491	102196576	6000.0	36 months	11.44%	197.750	B	
1741062	2925492	99799684	30000.0	60 months	25.49%	889.000	E	

1741063 rows × 78 columns

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

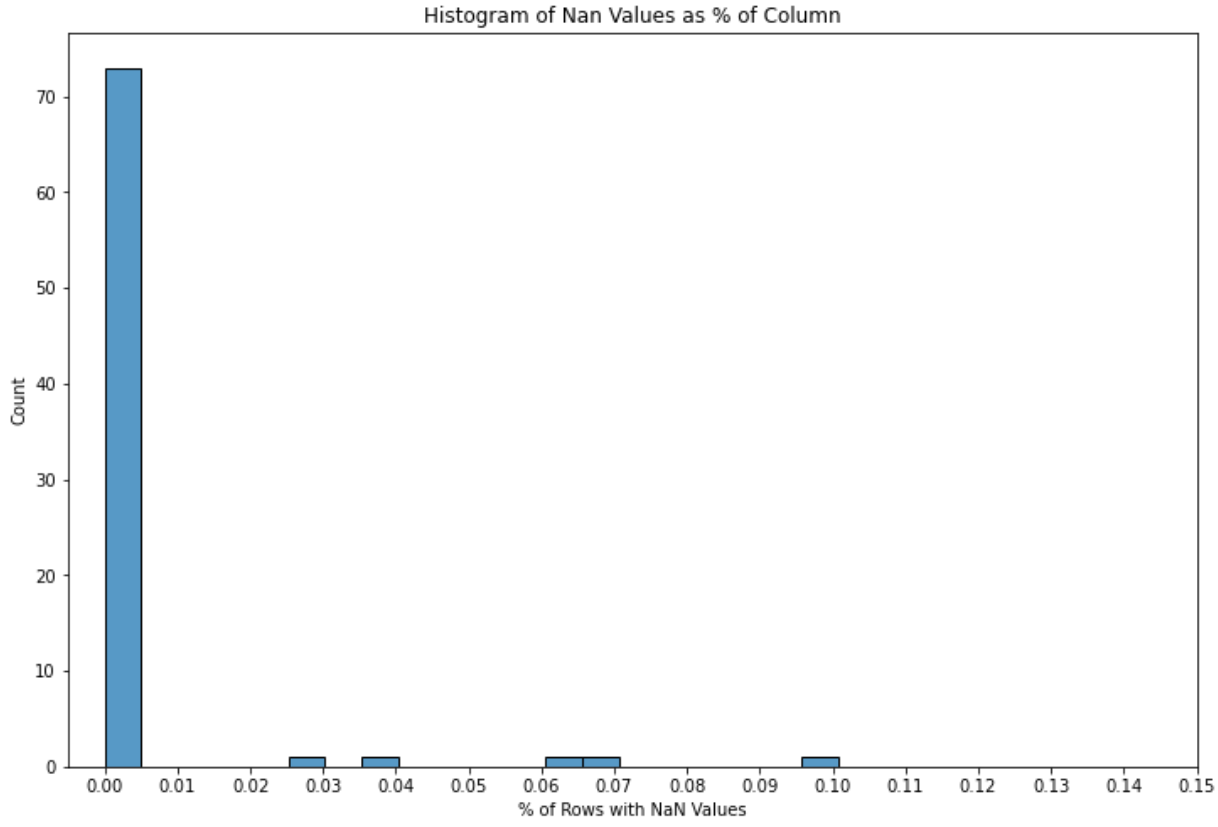
executed in 15ms, finished 02:20:28 2021-04-21

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1741063 entries, 0 to 1741062
Data columns (total 78 columns):
 #   Column                      Dtype
---  ------                      -----
 0   Unnamed: 0                  int32
 1   id                          int32
 2   loan_amnt                   float16
 3   term                        object
 4   int_rate                    object
 5   installment                 float16
 6   grade                       object
 7   sub_grade                   object
 8   emp_title                   object
 9   emp_length                  object
 10  home_ownership              object
 11  annual_inc                  float32
 12  verification_status         object
 13  issue_d                     object
```

In [9]:
```
fig,ax = plt.subplots(figsize=(12,8))
sns.histplot(data=df.isna().mean(),bins=20)
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 1.57s, finished 02:24:32 2021-04-21

Out[9]:
```
([<matplotlib.axis.YTick at 0x1fa3790e580>,
  <matplotlib.axis.YTick at 0x1fa3790e160>,
  <matplotlib.axis.YTick at 0x1fa569ec3a0>,
  <matplotlib.axis.YTick at 0x1fa569ec880>,
  <matplotlib.axis.YTick at 0x1fa569ecd90>,
  <matplotlib.axis.YTick at 0x1fa597a72e0>,
  <matplotlib.axis.YTick at 0x1fa597a77f0>,
  <matplotlib.axis.YTick at 0x1fa597a7d00>],
 [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 [16]:
```
#remaining features with significant NA values to handle
na_check(df).head(5)
```

executed in 1.35s, finished 02:25:49 2021-04-21

Out[16]:
```
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
dtype: float64
```

In [11]:
```
mths_weights = list(df['mths_since_recent_inq'].value_counts(normalize=True,dropr
mths_vals = list(df['mths_since_recent_inq'].value_counts(normalize=True,dropna=T
```

executed in 57ms, finished 02:24:39 2021-04-21

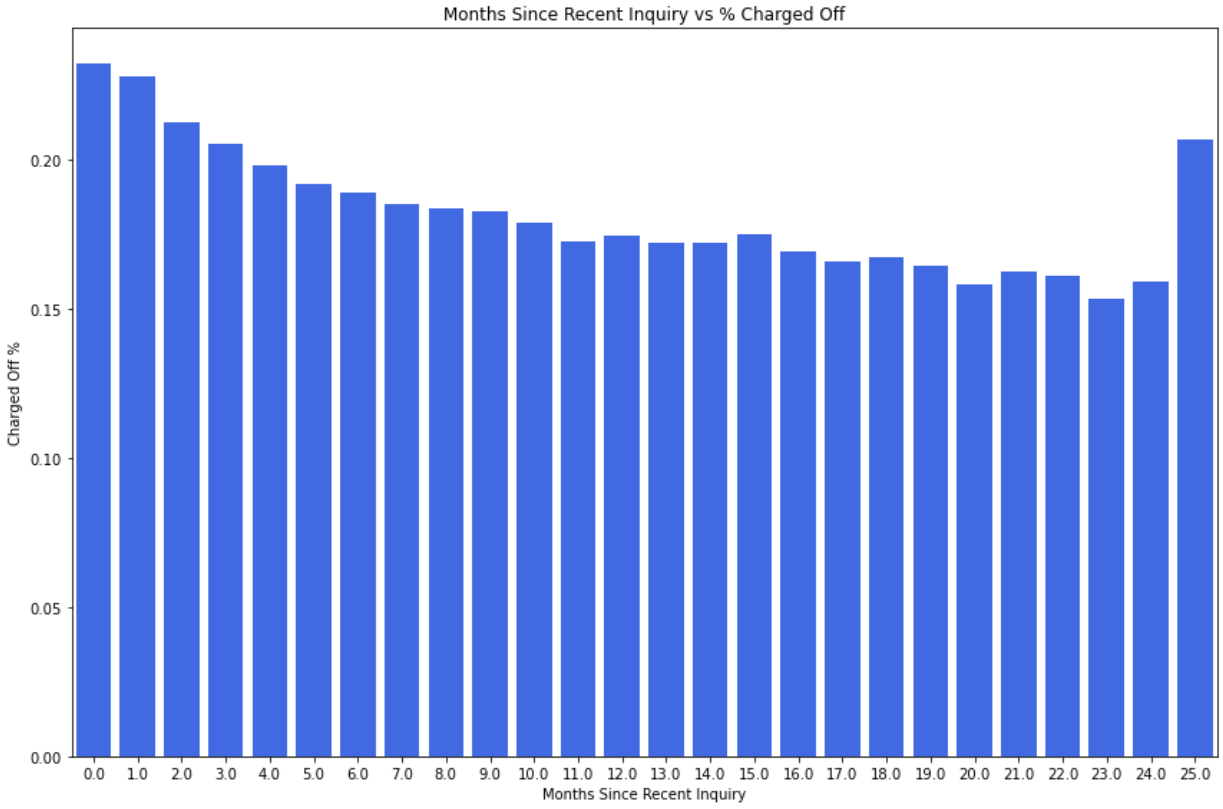
In [12]:
```
df['mths_since_recent_inq'] = df['mths_since_recent_inq'].apply(lambda x:
                                                    np.random.choice(
```

executed in 5.08s, finished 02:24:45 2021-04-21

In [13]:
```
fig,ax = plt.subplots(figsize=(12,8))
charge_off_rates = df.groupby('mths_since_recent_inq')['loan_status'].value_count
sns.barplot(x=charge_off_rates.index, y=charge_off_rates.values, color='royalblue
plt.xlabel('Months Since Recent Inquiry')
plt.ylabel('Charged Off %')
plt.title('Months Since Recent Inquiry vs % Charged Off')

plt.tight_layout()
```

executed in 599ms, finished 02:24:46 2021-04-21



Graph shows us that overtime if there hasnt been an inquiry then they are less likely to charge off their loan

In [15]:
```
column_info('emp_title')
```

executed in 19ms, finished 02:25:21 2021-04-21

Out[15]:
```
'The job title supplied by the Borrower when applying for the loan.*'
```

In [17]:
```
df.emp_title.value_counts()
```
executed in 669ms, finished 02:25:57 2021-04-21

Out[17]:
```
Teacher                          29823
Manager                          27324
Owner                            15298
Registered Nurse                 12225
RN                               11793
                                 ...
GL Bookkeeper                        1
EXECUTIVE PASTOR                     1
Ga Dept of Transportation            1
Pouschine Cook Capital               1
Sr International Accountant          1
Name: emp_title, Length: 410817, dtype: int64
```

In [18]:
```
#Too many unique titles to run as dummy variables as shown below 410k will drop
df.drop(columns='emp_title',axis=1,inplace=True)
```
executed in 553ms, finished 02:26:07 2021-04-21

In [19]:
```
df['emp_length'].value_counts(dropna=False)
```
executed in 127ms, finished 02:26:09 2021-04-21

Out[19]:
```
10+ years    577115
2 years      157469
< 1 year     143287
3 years      139470
1 year       113914
NaN          109077
5 years      107285
4 years      103825
6 years       79175
8 years       74087
7 years       73084
9 years       63275
Name: emp_length, dtype: int64
```

In [20]:
```
emp_length_weight = list(df['emp_length'].value_counts(normalize=True,dropna=True
emp_len_index = list(df['emp_length'].value_counts(normalize=True,dropna=True).in
```
executed in 258ms, finished 02:26:12 2021-04-21

In [21]:
```
df['emp_length'].fillna(value='x',inplace=True)
```
executed in 63ms, finished 02:26:13 2021-04-21

In [22]:
```
df['emp_length'] = df['emp_length'].apply(lambda x: np.random.choice(emp_len_inde
```
executed in 2.38s, finished 02:26:17 2021-04-21

In [23]:
```
#dropping sympols before converting to int
df['emp_length'] = df['emp_length'].apply(lambda x: str(x).replace('+','').replac
```
executed in 2.68s, finished 02:26:21 2021-04-21

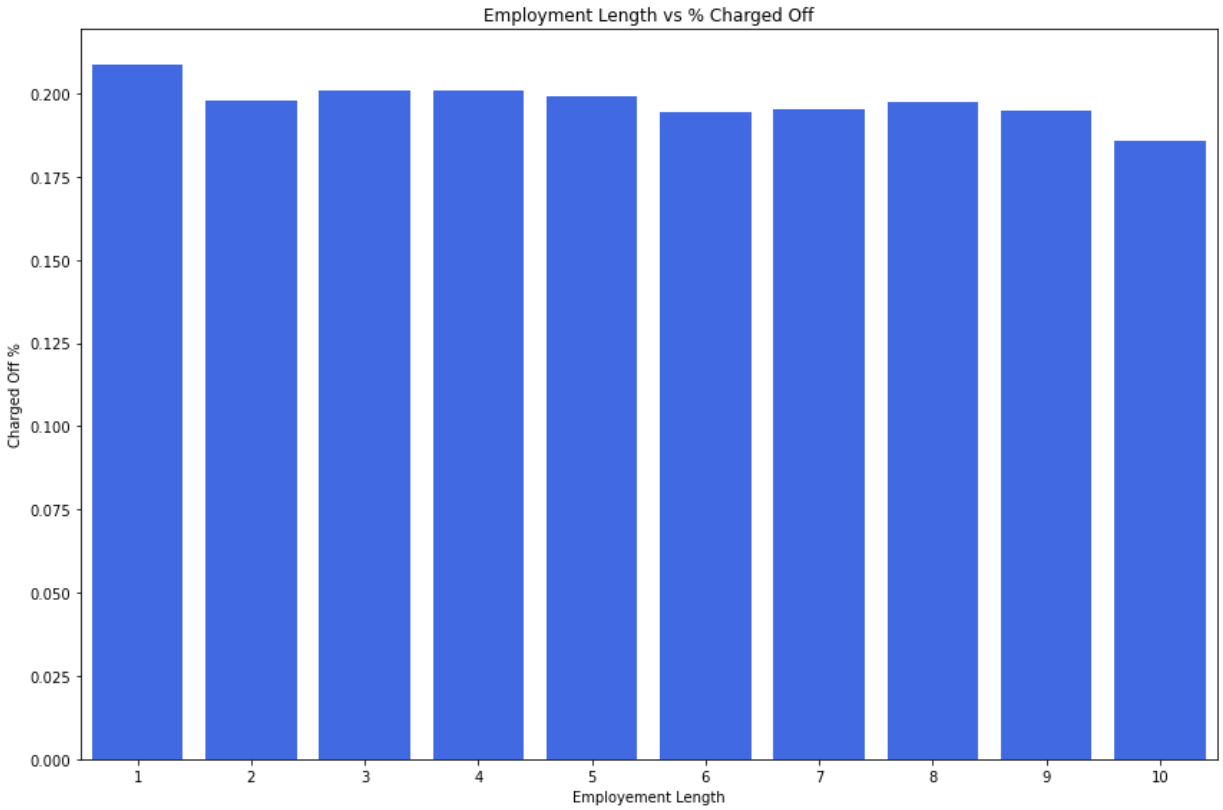
In [24]:
```
df['emp_length'] = df['emp_length'].apply(lambda x: int(x[0]) )
```
executed in 984ms, finished 02:26:22 2021-04-21

In [25]:
```
df['emp_length']
```
executed in 15ms, finished 02:26:23 2021-04-21

Out[25]:
```
0           4
1           2
2          10
3           3
4           4
           ..
1741058     1
1741059    10
1741060     8
1741061     5
1741062     4
Name: emp_length, Length: 1741063, dtype: int64
```

In [26]:
```
fig,ax = plt.subplots(figsize=(12,8))
charge_off_rates = df.groupby('emp_length')['loan_status'].value_counts(normalize
sns.barplot(x=charge_off_rates.index, y=charge_off_rates.values, color='royalblue
plt.xlabel('Employement Length')
plt.ylabel('Charged Off %')
plt.title('Employment Length vs % Charged Off')

plt.tight_layout()
```
executed in 458ms, finished 02:26:26 2021-04-21



In [27]:
```
na_check(df)
```
executed in 1.22s, finished 02:26:30 2021-04-21

Out[27]:
```
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
                            ...
bc_util                     0.00
chargeoff_within_12_mths    0.00
delinq_amnt                 0.00
mo_sin_old_rev_tl_op        0.00
Unnamed: 0                  0.00
Length: 77, dtype: float64
```

In [28]:
```
column_info('num_tl_120dpd_2m')
```
executed in 10ms, finished 02:26:32 2021-04-21

Out[28]:
```
'Number of accounts currently 120 days past due (updated in past 2 months)'
```

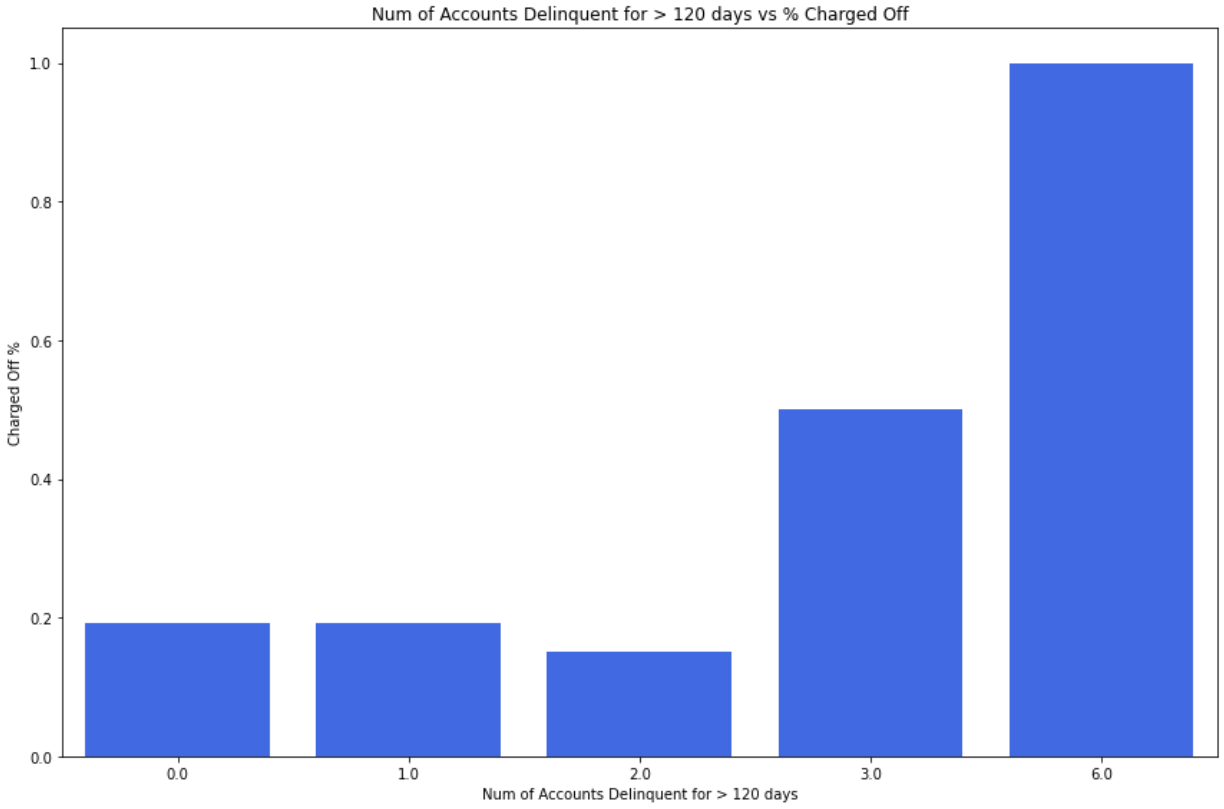
In [29]:
```
df['num_tl_120dpd_2m'].value_counts(normalize=True)
```
executed in 29ms, finished 02:28:01 2021-04-21

Out[29]:
```
0.0    9.993247e-01
1.0    6.478184e-04
2.0    2.388271e-05
3.0    2.388271e-06
6.0    5.970677e-07
4.0    5.970677e-07
Name: num_tl_120dpd_2m, dtype: float64
```

In [30]:
```
fig,ax = plt.subplots(figsize=(12,8))
charge_off_rates = df.groupby('num_tl_120dpd_2m')['loan_status'].value_counts(nor
sns.barplot(x=charge_off_rates.index, y=charge_off_rates.values, color='royalblue
plt.xlabel('Num of Accounts Delinquent for > 120 days')
plt.ylabel('Charged Off %')
plt.title('Num of Accounts Delinquent for > 120 days vs % Charged Off')

plt.tight_layout()
```

executed in 346ms, finished 02:28:17 2021-04-21



In [31]:
```
p = list(df['num_tl_120dpd_2m'].value_counts(normalize=True,dropna=True))
a = list(df['num_tl_120dpd_2m'].value_counts(normalize=True,dropna=True).index)
```

executed in 51ms, finished 02:28:21 2021-04-21

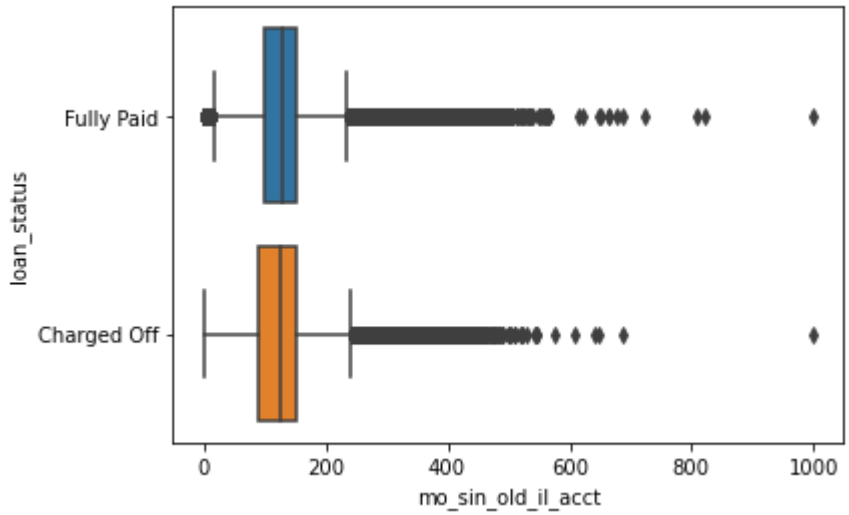
In [32]:
```
df['num_tl_120dpd_2m'] = df['num_tl_120dpd_2m'].apply(lambda x: np.random.choice(
```

executed in 2.96s, finished 02:28:24 2021-04-21

In [33]:
```
sns.boxplot(x='mo_sin_old_il_acct',y='loan_status',data=df)
```

executed in 741ms, finished 02:28:26 2021-04-21

Out[33]:
```
<AxesSubplot:xlabel='mo_sin_old_il_acct', ylabel='loan_status'>
```



In [34]:
```
column_info('mo_sin_old_il_acct')
```

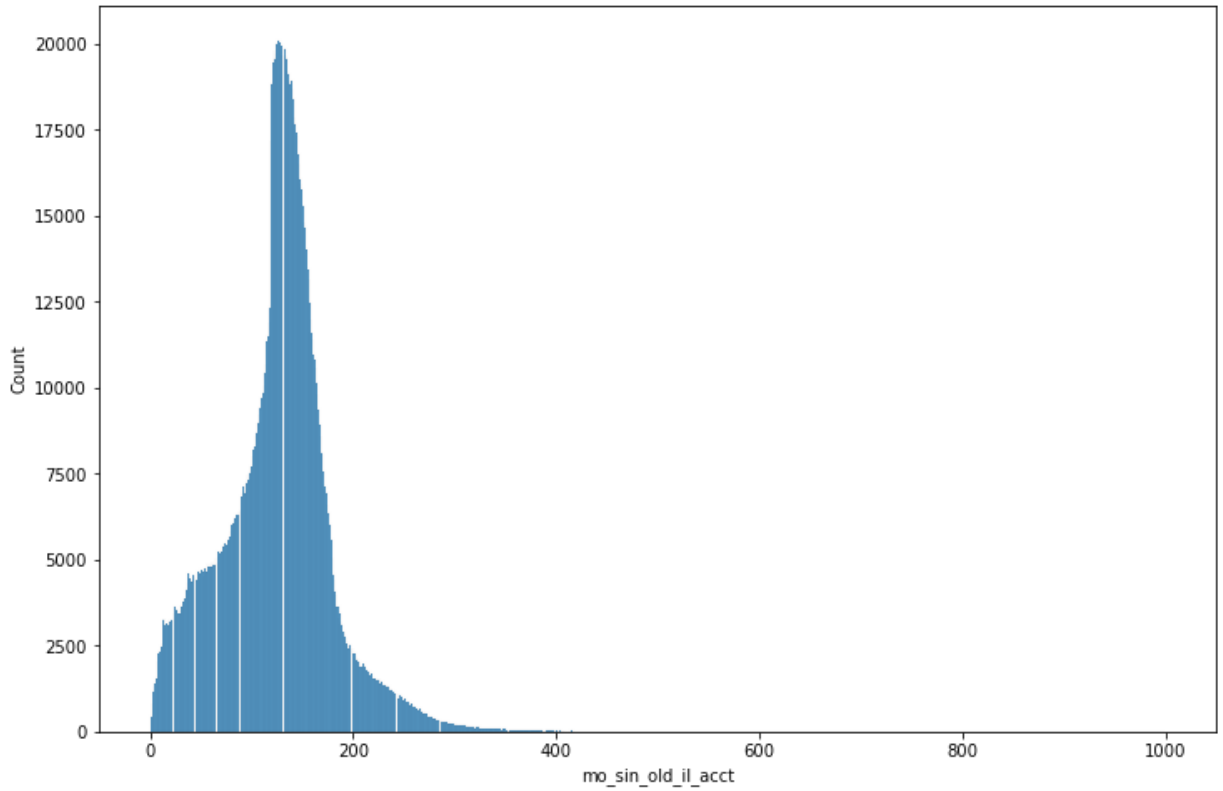
executed in 16ms, finished 02:28:32 2021-04-21

Out[34]:
```
'Months since oldest bank installment account opened'
```

In [35]:
```
fig,ax = plt.subplots(figsize=(12,8))
sns.histplot(df.loc[df['mo_sin_old_il_acct'].notnull(), 'mo_sin_old_il_acct'], kd
```

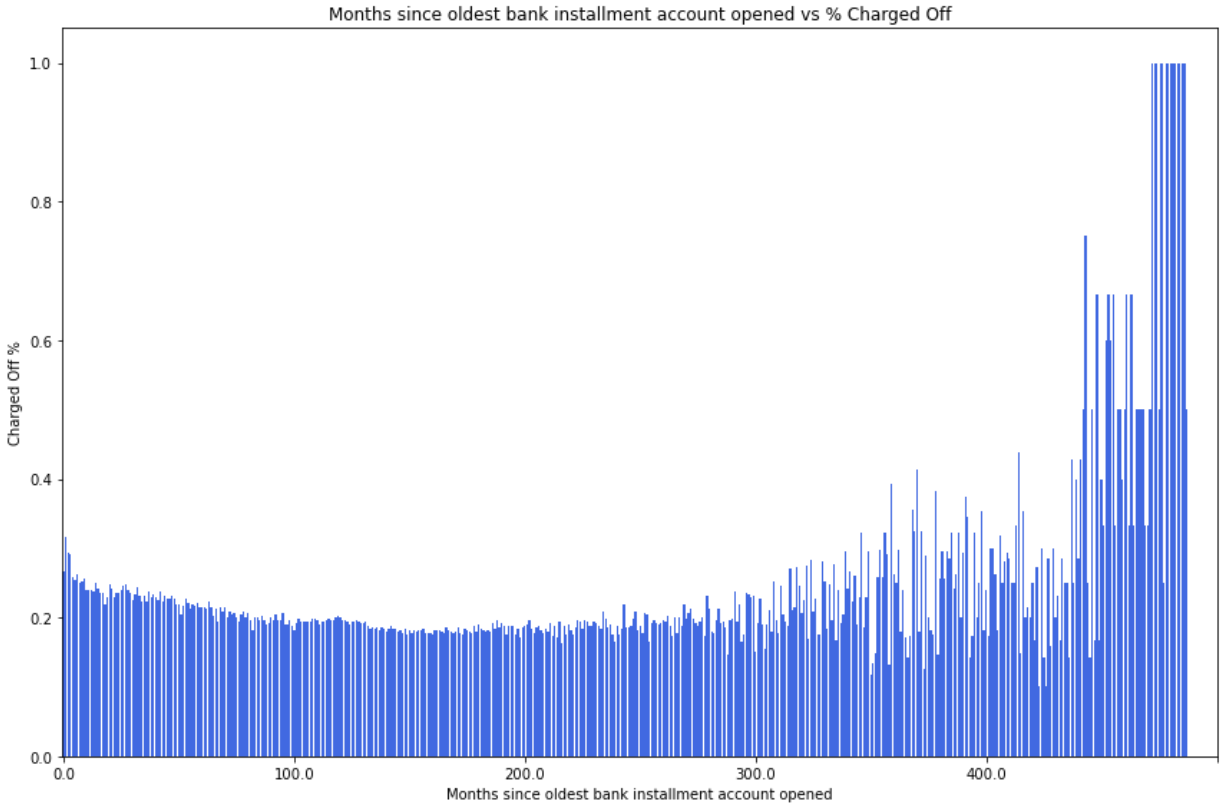
executed in 2.03s, finished 02:28:34 2021-04-21

Out[35]: <AxesSubplot:xlabel='mo_sin_old_il_acct', ylabel='Count'>



In [37]:
```
fig,ax = plt.subplots(figsize=(12,8))
charge_off_rates = df.groupby('mo_sin_old_il_acct')['loan_status'].value_counts(r
sns.barplot(x=charge_off_rates.index, y=charge_off_rates.values, color='royalblue
plt.xlabel(column_info('mo_sin_old_il_acct'))
plt.ylabel('Charged Off %')
plt.title(column_info('mo_sin_old_il_acct')+ ' vs % Charged Off')
plt.xticks(np.linspace(0,500,6))
plt.tight_layout()
```

executed in 2.38s, finished 02:28:42 2021-04-21



All values to right of 200 are extreme outliers so will not relevant. Also making determination on oldest bank installments of 33 years prior doesn't seem to make too much sense

In [38]:
```
df.drop(columns=['mo_sin_old_il_acct'],axis=1,inplace=True)
```

executed in 595ms, finished 02:28:57 2021-04-21

In [39]:
```
#for cols with insignificant amount of nas missing
df.dropna(inplace=True)
```

executed in 2.17s, finished 02:29:00 2021-04-21

In [40]:
```
#now that there are no na values will start feature inspection here with this fil
df.to_csv('data/preprocessed.csv')
```

executed in 1m 15.6s, finished 02:30:16 2021-04-21