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

from matplotlib.ticker import FuncFormatter
```

executed in 3.04s, finished 21:39:40 2021-04-21

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

executed in 61ms, finished 21:39:40 2021-04-21

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

In [3]:
```
def plot_cats(column):
    loan_statuses = ['Fully Paid','Charged Off']
    g = df.groupby(column)['loan_status'].value_counts(normalize=True).unstack()
    list_dfs = []
    for status in loan_statuses:
        vals = g[status].values
        idx = g[status].index
        frame = pd.DataFrame(data=vals,index=idx,columns=['value_counts']).reset_
        frame['loan_status'] = status
        list_dfs.append(frame)
    comb = pd.concat([list_dfs[0],list_dfs[1]])
    num = df[column].nunique()
    fig, (ax,ax1) = plt.subplots(1,2,figsize=(15,8))
    sns.histplot(x=column, data=df.sort_values(column),bins=(num/4),ax=ax)
    ax.set_xlabel(column)
    ax.set_ylabel('Count')
    ax.set_title(column + ' Histplot')
    sns.despine()
    sns.barplot(x=column,y='value_counts',hue='loan_status',data=comb,ax=ax1)
    ax1.set_xlabel(column)
    ax1.set_ylabel('% of Total Loans')
    ax1.set_title(column + ' by Loan Status')
    plt.tight_layout()
```
executed in 14ms, finished 21:39:40 2021-04-21

In [4]:
```
def na_check(data):
    check = np.round(data.isna().mean().sort_values(ascending=False),2)
    return check
```
executed in 13ms, finished 21:39:40 2021-04-21

In [5]:
```
def column_info(col_name):
    return column_defs.loc[col_name]['Description']
```
executed in 13ms, finished 21:39:40 2021-04-21

In [6]:
```
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.float32).min and c_max < np.finfo(np.float
                    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 27ms, finished 21:39:41 2021-04-21

In [7]:
```
df = pd.read_csv('data/preprocessed.csv')
```
executed in 19.6s, finished 21:40:04 2021-04-21

In [8]:
```
df.drop(columns=['Unnamed: 0','Unnamed: 0.1'],axis=1,inplace=True)
```
executed in 1.16s, finished 21:40:07 2021-04-21

In [9]:
```
reduce_mem_usage(df,int_cast=False)
```
executed in 12.0s, finished 21:40:20 2021-04-21

```
  0%|          | 0/75 [00:00<?, ?it/s]

Memory usage of dataframe is 993.88 MB

100%|██████████| 75/75 [00:10<00:00,  6.88it/s]

Memory usage after optimization is: 637.742 MB
Decreased by 35.8%
```

Out[9]:

	id	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_
0	10129454	12000.0	36 months	10.99%	392.799988	B	B2	
1	10149488	4800.0	36 months	10.99%	157.100006	B	B2	
2	10149342	27060.0	36 months	10.99%	885.500000	B	B2	

1 Feature Inspection and EDA

I've imported my preprocessed data which has dealt with null values in our data. Going through our feature inspection, I will append the list below for features that our inspection & EDA find's to be unimportant

In [10]:
```
features_to_drop = []
```
executed in 12ms, finished 21:40:33 2021-04-21

In [11]:
```
#Shows to 5 most recurring values for each feature.
for col in df.columns:
    print(col)
    print(column_info(col))
    print(df[col].value_counts(normalize = True, ascending=False).head(5))
    print("-------------------------------------------------------------")
```
executed in 9.03s, finished 21:40:43 2021-04-21

```
id
A unique LC assigned ID for the loan listing.
4196351      5.757261e-07
75101579     5.757261e-07
137966995    5.757261e-07
62459284     5.757261e-07
68525187     5.757261e-07
Name: id, dtype: float64
-------------------------------------------------------------
loan_amnt
The listed amount of the loan applied for by the borrower. If at some point i
n time, the credit department reduces the loan amount, then it will be reflec
ted in this value.
10000.0    0.078220
20000.0    0.054624
12000.0    0.053229
15000.0    0.053078
35000.0    0.037777
Name: loan_amnt, dtype: float64
```

Going through this check we found payment plan and outstanding principal had only 1 value so not providing any insight for us. Also ID is unique to each loan so will not unlock any information for us either

In [12]:
```
for feature in ['id','pymnt_plan','out_prncp']:
    features_to_drop.append(feature)
```
executed in 14ms, finished 21:40:44 2021-04-21

In [13]:
```
#converting our issue_date to datetime so we can evaluate loans at origination ov
df.issue_d = pd.to_datetime(df.issue_d)
```
executed in 507ms, finished 21:40:45 2021-04-21

In [ ]:
```
#df_cont_z = df[(np.abs(stats.zscore(df[cont_columns]))<4).all(axis=1)]
```

In [16]:
```python
df.info()
```
executed in 28ms, finished 21:41:38 2021-04-21

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1736937 entries, 0 to 1736936
Data columns (total 75 columns):
 #   Column                       Dtype
---  ------                       -----
 0   id                           int32
 1   loan_amnt                    float32
 2   term                         object
 3   int_rate                     object
 4   installment                  float32
 5   grade                        object
 6   sub_grade                    object
 7   emp_length                   int8
 8   home_ownership               object
 9   annual_inc                   float32
 10  verification_status          object
 11  issue_d                      datetime64[ns]
 12  loan_status                  object
 13  pymnt_plan                   object
```

1.1 Loan Amount

In [17]:
```python
print('Loan Amount:\n{}'.format(column_info('loan_amnt')))
```
executed in 14ms, finished 21:41:42 2021-04-21

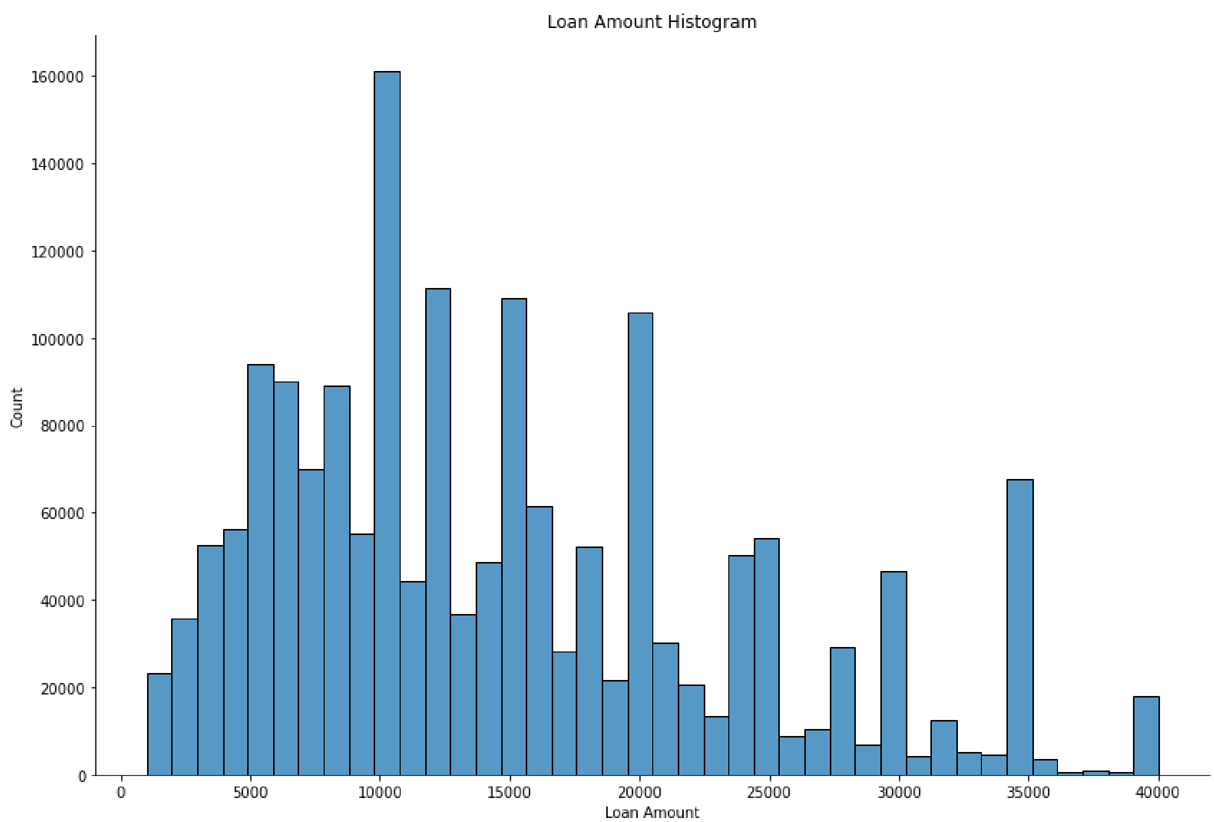
```
Loan Amount:
The listed amount of the loan applied for by the borrower. If at some point in
time, the credit department reduces the loan amount, then it will be reflected
in this value.
```

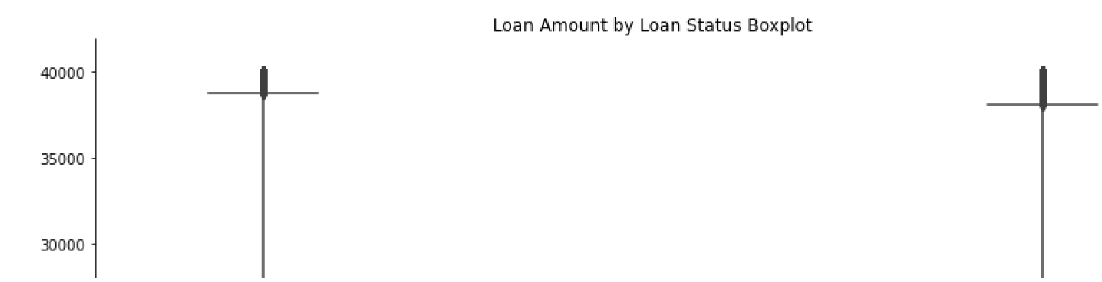
In [18]:
```python
df.loan_amnt.value_counts()
```
executed in 62ms, finished 21:41:43 2021-04-21

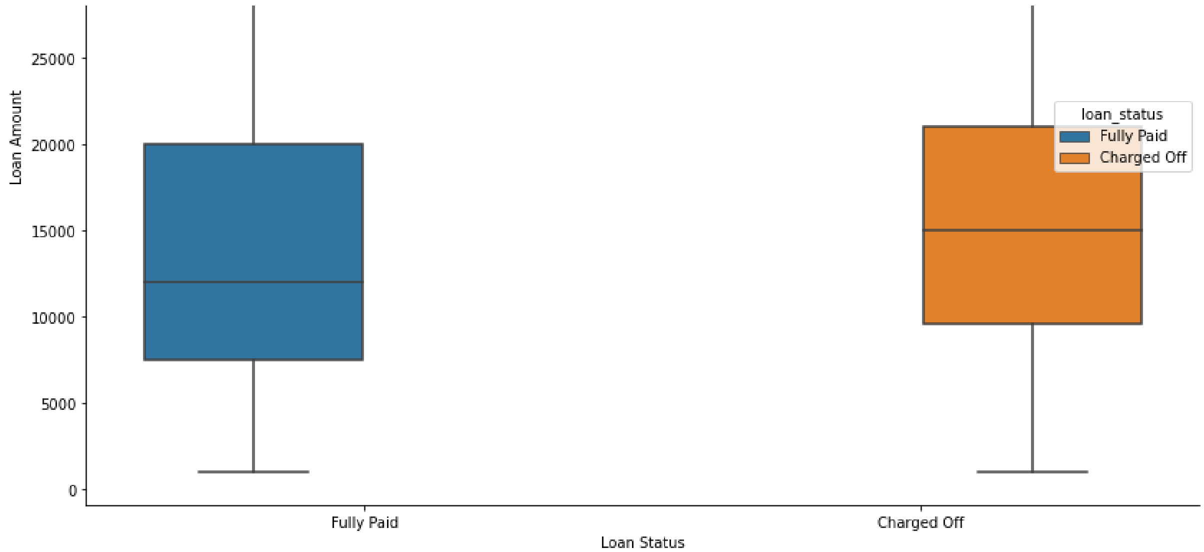
Out[18]:
```
10000.0    135864
20000.0     94879
12000.0     92455
15000.0     92193
35000.0     65617
            ...
39330.0         3
39230.0         3
36770.0         3
35680.0         2
37980.0         1
Name: loan_amnt, Length: 1498, dtype: int64
```

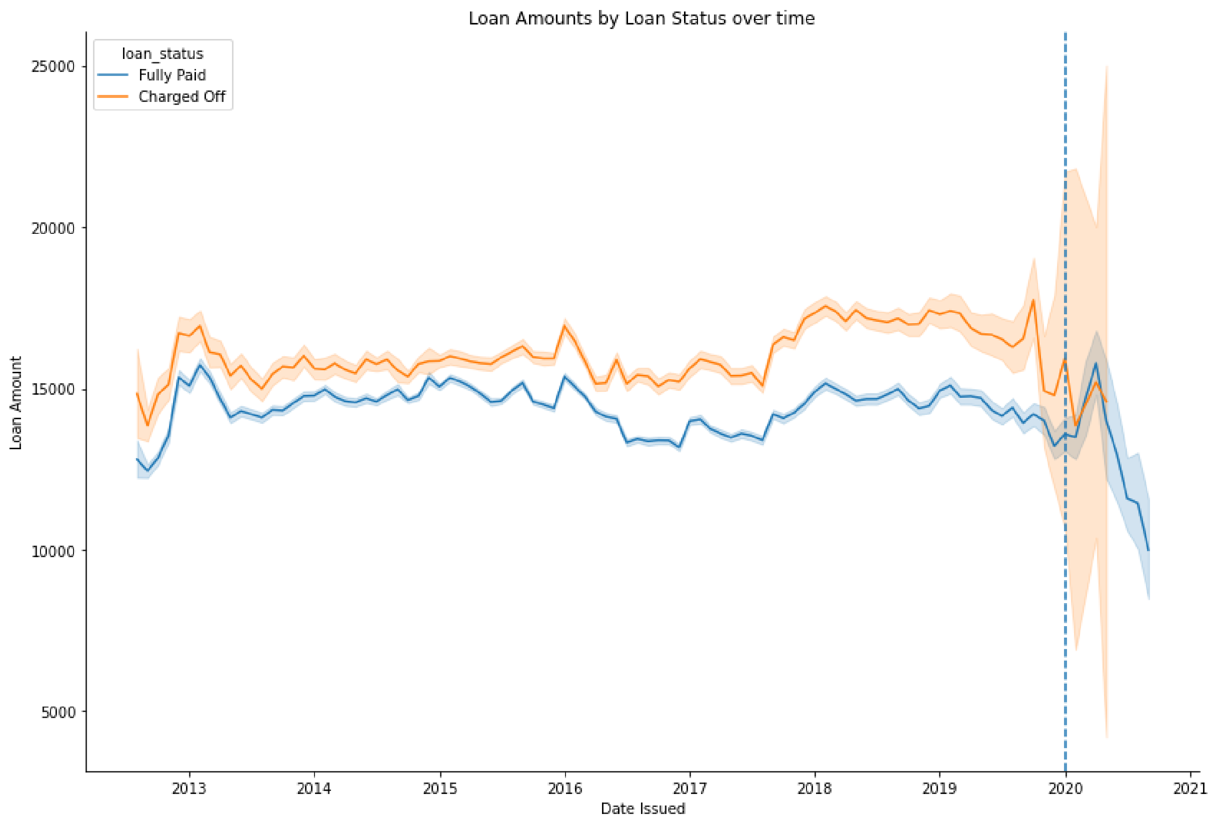
In [103]:
```
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
sns.histplot(x='loan_amnt',data=df,bins=40,ax=ax1)
ax1.set_xlabel('Loan Amount')
ax1.set_ylabel('Count')
ax1.set_title('Loan Amount Histogram')

sns.despine()
sns.boxplot(x='loan_status' ,y='loan_amnt',hue='loan_status',data=df,ax=ax2)
ax2.set_xlabel('Loan Status')
ax2.set_ylabel('Loan Amount')
ax2.set_title('Loan Amount by Loan Status Boxplot')




sns.lineplot(x='issue_d' ,y='loan_amnt',hue='loan_status',data=df,ax=ax3)
ax3.set_xlabel('Date Issued')
ax3.set_ylabel('Loan Amount')
ax3.set_title('Loan Amounts by Loan Status over time')
ax3.axvline(x='2020',linestyle='--')
plt.tight_layout()
plt.savefig('images/loan_amount.png')
```

executed in 15.7s, finished 01:45:24 2021-04-22









From our histogram we can see that loans are generally issued at increment of $5000

Our Boxplot shows u that loans that were charged off on average had higher loan amounts also represented in the final chart over time.

Also, our last graph shows us unsurprisingly that during covid loan amounts across both groups were reduced.

1.2 Grade, SubGrade

In [22]:
```
df.sub_grade.value_counts()
```
executed in 192ms, finished 21:59:41 2021-04-21

Out[22]:
```
C1    111249
B5    108420
B4    108080
B3    101337
C2    101066
C3     98078
C4     97457
B1     96292
B2     95881
C5     89676
A5     82978
A4     70457
D1     65042
A1     63754
D2     59184
A3     53336
A2     51599
D3     50195
D4     44203
D5     37361
E1     27161
E2     24135
E3     21158
E4     17949
E5     17352
F1     10737
F2      7502
F3      6385
F4      5030
F5      4176
G1      3236
G2      2202
G3      1696
G4      1341
G5      1232
Name: sub_grade, dtype: int64
```

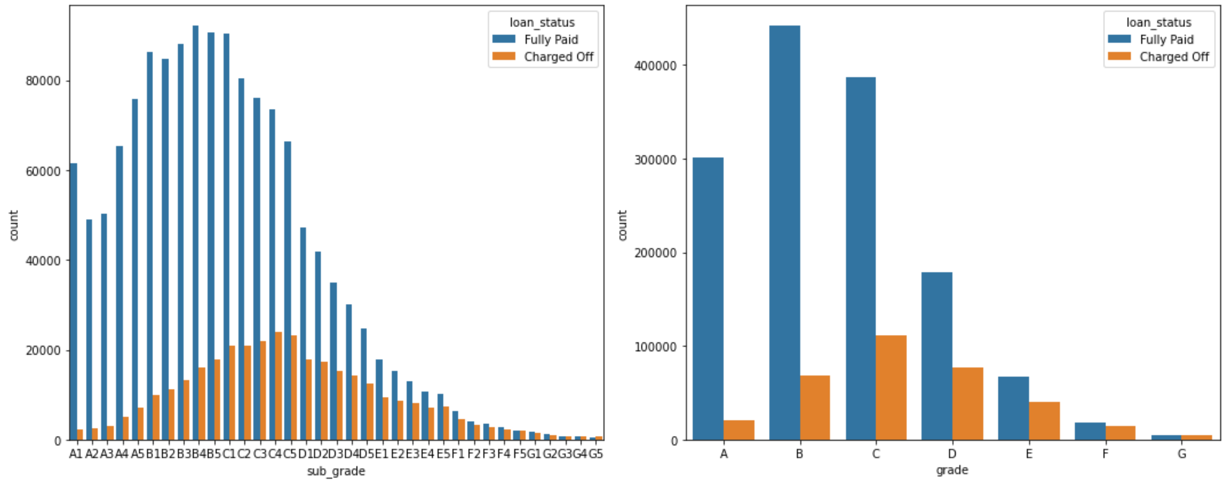
In [23]:
```
df.grade.value_counts()
```
executed in 125ms, finished 21:59:42 2021-04-21

Out[23]:
```
B    510010
C    497526
A    322124
D    255985
E    107755
F     33830
G      9707
Name: grade, dtype: int64
```

In [24]:

```
fig,(ax,ax1) = plt.subplots(1,2 ,figsize=(15,6))
sns.countplot(x='sub_grade',hue='loan_status',data=df.sort_values('sub_grade'),ax

sns.countplot(x='grade',hue='loan_status',data=df.sort_values('grade'),ax=ax1)
plt.tight_layout()
```

executed in 15.9s, finished 21:59:59 2021-04-21



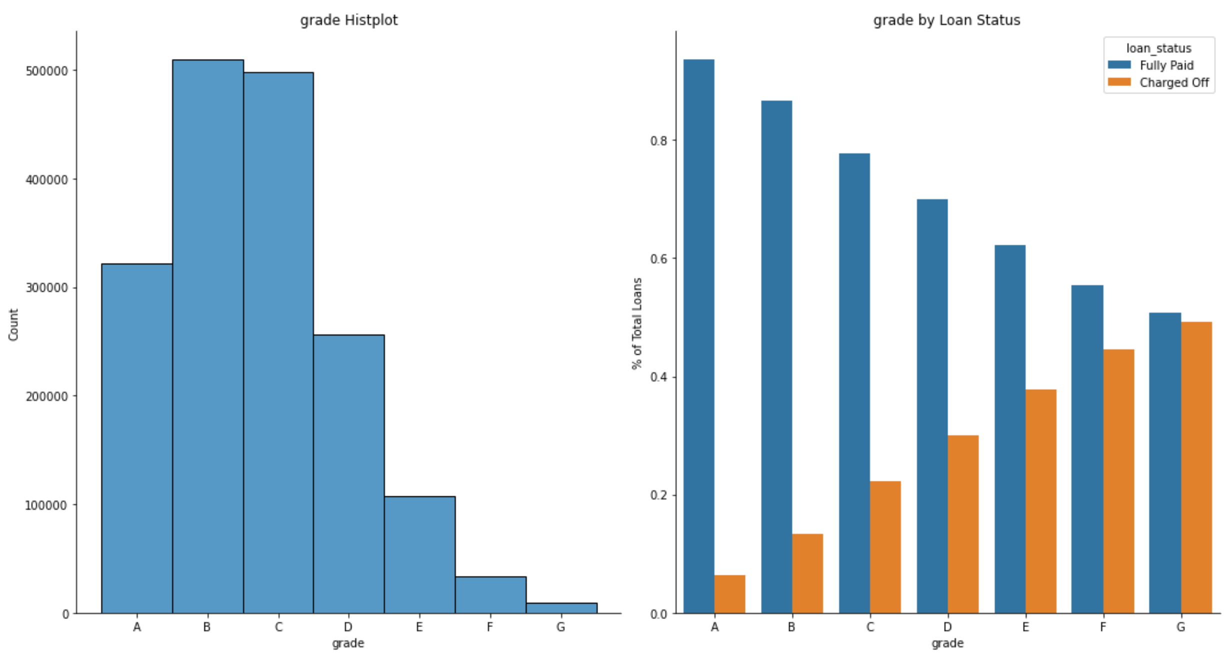
In [ ]:

executed in 11ms, finished 22:29:52 2021-04-12

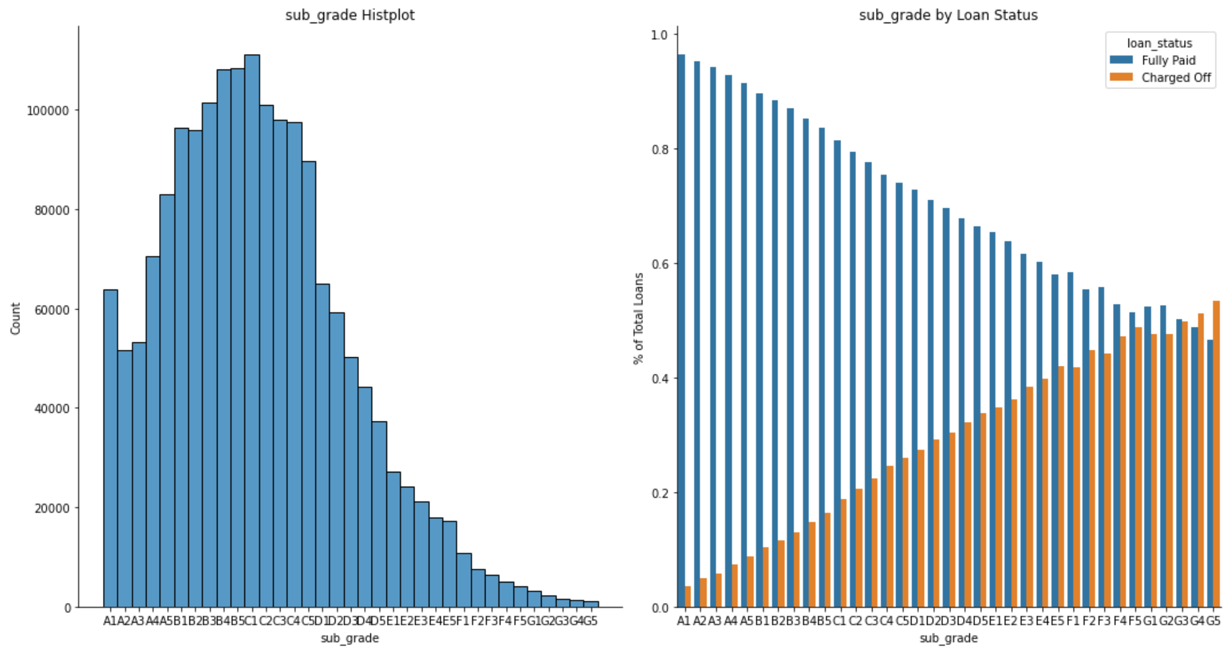
In [104]:

```
plot_cats('grade')
plt.savefig('grade.png')
```

executed in 3.80s, finished 01:46:38 2021-04-22



In [105]:
```
plot_cats('sub_grade')
plt.savefig('sub_grade.png')
```
executed in 5.17s, finished 01:46:49 2021-04-22



Can see Sub_Grade and Grade both increase the probability of charge of the worst the credit grade, as we would suspect

Sub grade corresponds to grade levels but is more granular, we will keep both for now as our modeling will may overfit with more categorical variables

1.3 Fico Score

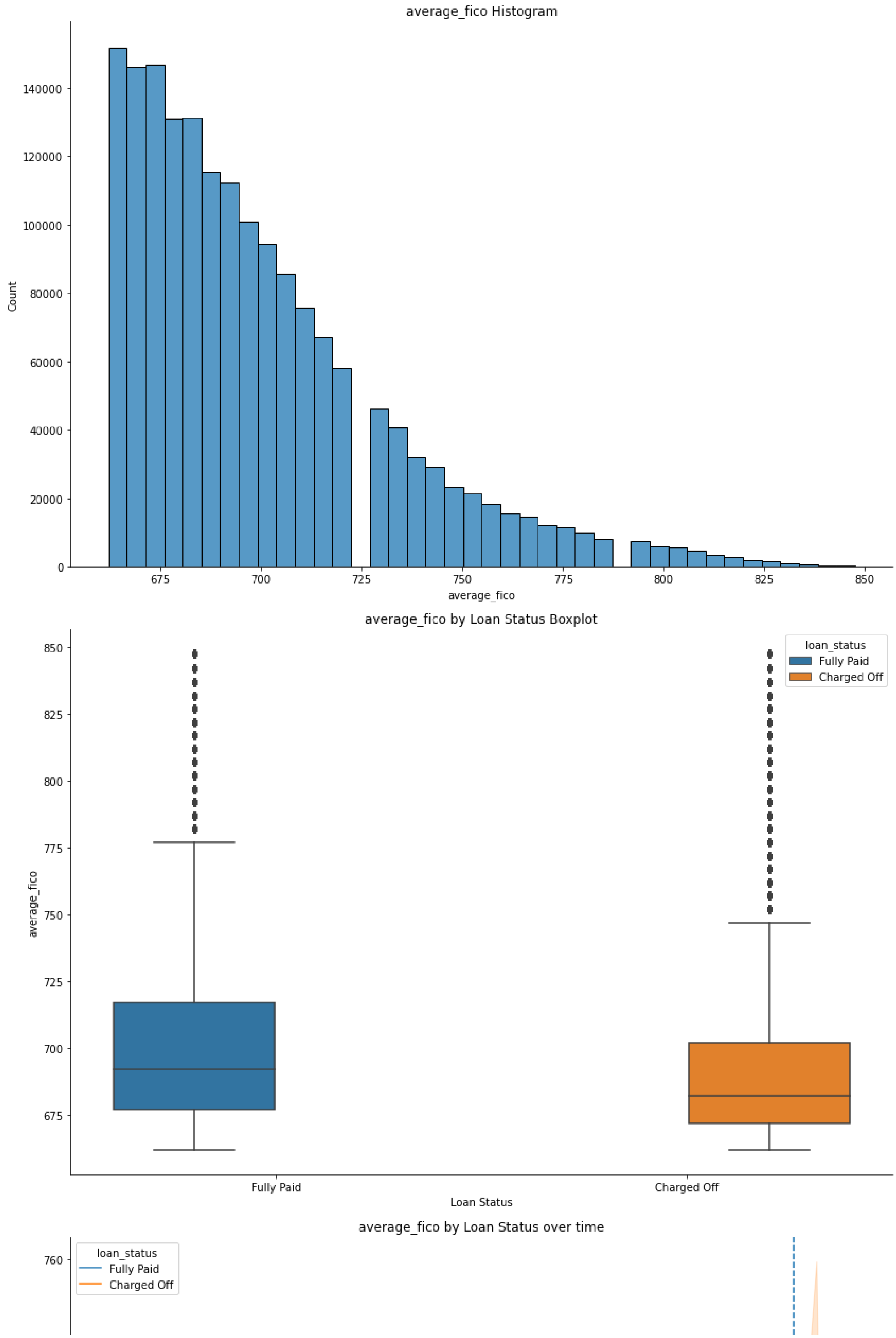
As we have both fico range high and low will first average them and drop the range boundaries

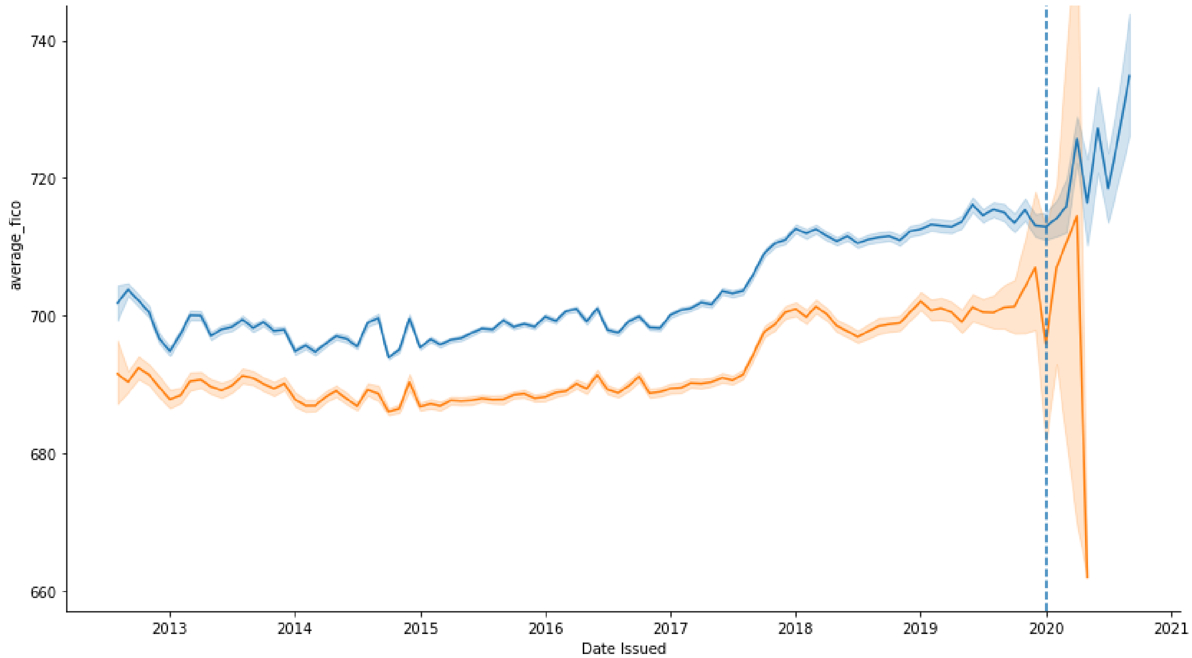
In [31]:
```
# making average fico score and dropping the fico range high and low
df['average_fico'] = (df['fico_range_high'] + df['fico_range_low'])/2
df.drop(columns=['fico_range_high','fico_range_low'],axis=1,inplace=True)
```
executed in 1.27s, finished 22:02:38 2021-04-21

In [32]:
```
def continuous_plot(feature):
    fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
    sns.histplot(x=feature,data=df,bins=40,ax=ax1)
    ax1.set_xlabel(feature)
    ax1.set_ylabel('Count')
    ax1.set_title('{} Histogram'.format(feature))

    sns.despine()
    sns.boxplot(x='loan_status' ,y=feature,hue='loan_status',data=df,ax=ax2)
    ax2.set_xlabel('Loan Status')
    ax2.set_ylabel(feature)
    ax2.set_title('{} by Loan Status Boxplot'.format(feature))

    sns.lineplot(x='issue_d' ,y=feature,hue='loan_status',data=df,ax=ax3)
    ax3.set_xlabel('Date Issued')
    ax3.set_ylabel(feature)
    ax3.set_title('{} by Loan Status over time'.format(feature))
    ax3.axvline(x='2020',linestyle='--')
    plt.tight_layout()
```

executed in 13ms, finished 22:02:39 2021-04-21

In [106]:
```python
continuous_plot('average_fico')
plt.savefig('fico_score.png')
```
executed in 15.7s, finished 01:47:37 2021-04-22



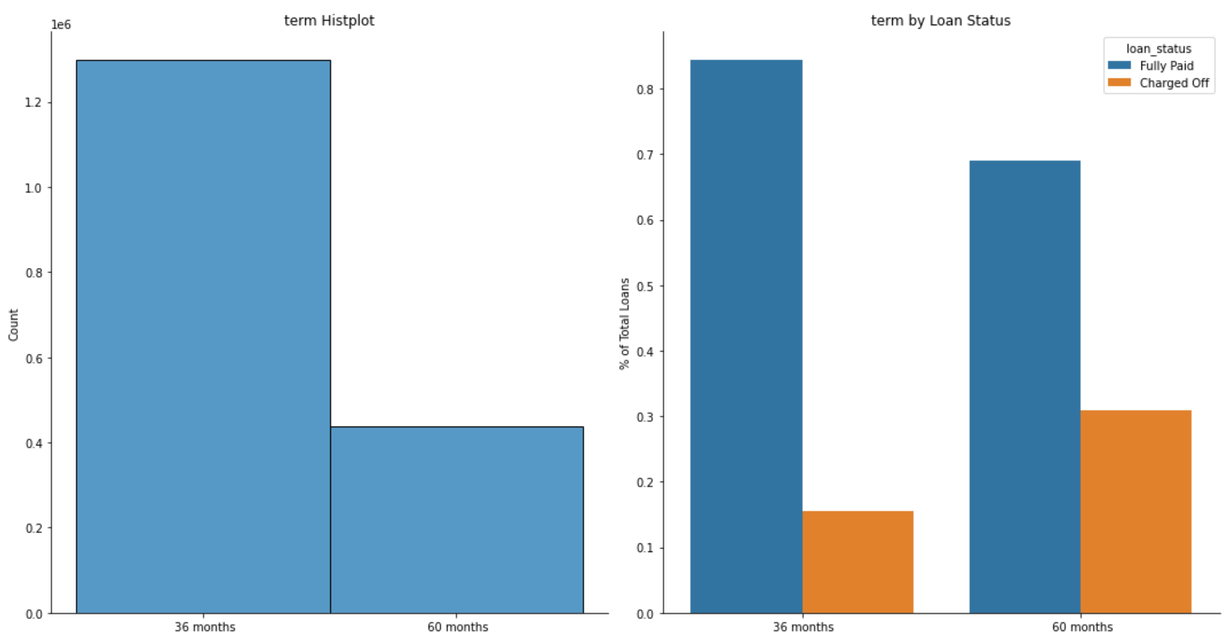


So interestingly, more loans were issued to applicants with lower fico scores, specifically below 725 But, as suspected the higher mean fico score was higher for loans that were fully paid off

This can be seen overtime as well, but additionally, the pandemic seemed to accentuate these relationships showing that average fico scores diverged sharply for fully paid and charged off loans

1.4 Term

In [107]:
```
plot_cats('term')
plt.savefig('term.png')
```
executed in 3.25s, finished 01:47:41 2021-04-22



In [35]:
```
df[df.term == ' 60 months']['loan_status'].value_counts(normalize=True)
```
executed in 441ms, finished 22:03:09 2021-04-21

Out[35]:
```
Fully Paid     0.690241
Charged Off    0.309759
Name: loan_status, dtype: float64
```

In [36]:
```
df[df.term == ' 36 months']['loan_status'].value_counts(normalize=True)
```
executed in 1.05s, finished 22:03:12 2021-04-21

Out[36]:
```
Fully Paid     0.844838
Charged Off    0.155162
Name: loan_status, dtype: float64
```

Loans are either issued at 36 or 60 month terms, but there are more than double the amount of 36 month loans compared to 60 month ones

Loans over longer time horizons (60 months) have double the percentage of charge offs increasing from 15% to 31%

1.5 Home Ownership

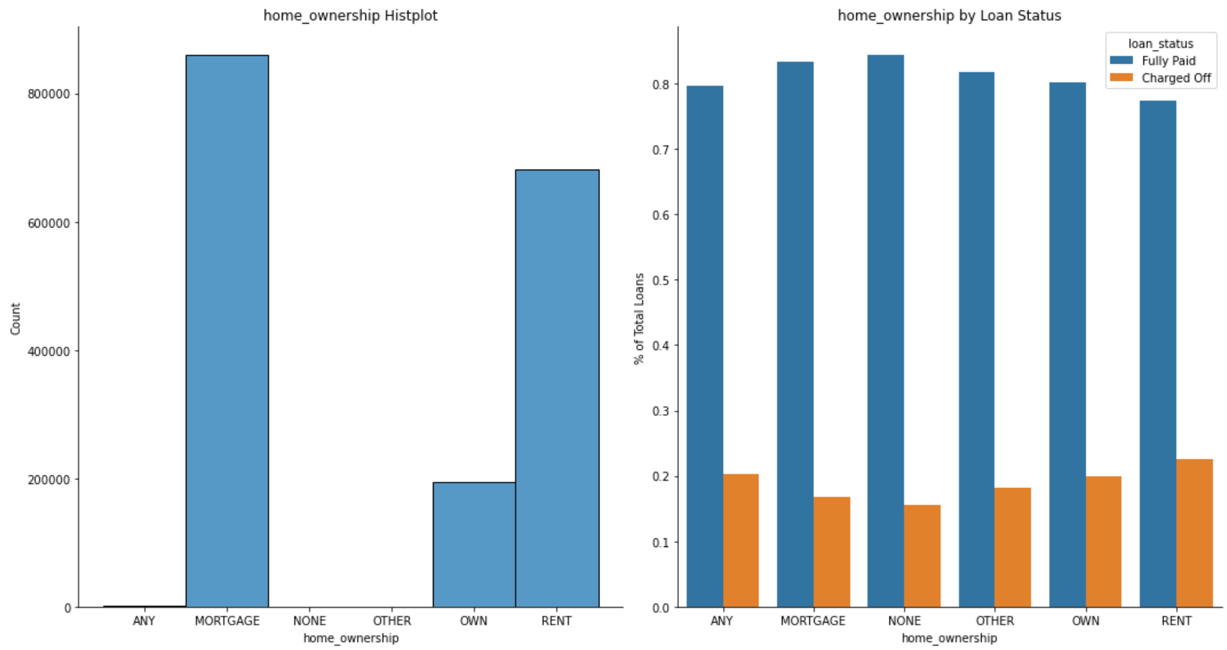
In [125]:
```
column_info('home_ownership')
```
executed in 18ms, finished 02:44:38 2021-04-22

Out[125]:
```
'The home ownership status provided by the borrower during registration\xa0or o
btained from the credit report.\xa0Our values are: RENT, OWN, MORTGAGE, OTHER'
```

In [37]:
```
df.home_ownership.value_counts()
```
executed in 174ms, finished 22:03:13 2021-04-21

Out[37]:
```
MORTGAGE    860991
RENT        681150
OWN         193574
ANY           1133
NONE            45
OTHER           44
Name: home_ownership, dtype: int64
```

In [124]:
```
plot_cats('home_ownership')
plt.savefig('home.png')
```
executed in 5.29s, finished 02:43:24 2021-04-22



For our home ownership feature most of the loans go to either applicants who have an existing mortgage or rent, followed by applicants who own their home.

Renters have the highest rate of charge offs across types of home ownership but the change is only a 5 % increase

1.6 Title and Purpose

In [39]:
```
column_info('title')
```
executed in 14ms, finished 22:03:21 2021-04-21

Out[39]: 'The loan title provided by the borrower'

In [40]:
```
df['title'].value_counts()
```
executed in 252ms, finished 22:03:22 2021-04-21

Out[40]:
```
Debt consolidation             921572
Credit card refinancing        361407
Home improvement               107869
Other                           98205
Major purchase                  35205
                                ...
Debt Consolidation/Remodel          1
House Updates                       1
AMEX payoff                         1
Debt and gutters                    1
CONSOLIDATION FREEDOM               1
Name: title, Length: 38722, dtype: int64
```

Title columns has too many unique values and are embedded in purpose so will drop

In [41]:
```
column_info('purpose')
```
executed in 14ms, finished 22:03:23 2021-04-21

Out[41]:
```
'A category provided by the borrower for the loan request. '
```

In [42]:
```
df['purpose'].value_counts()
```
executed in 203ms, finished 22:03:24 2021-04-21

Out[42]:
```
debt_consolidation    997908
credit_card           392750
home_improvement      114781
other                 103123
major_purchase         37334
medical                20527
car                    17473
small_business         17082
vacation               12146
moving                 11847
house                  10032
renewable_energy        1070
wedding                  862
educational                2
Name: purpose, dtype: int64
```

Looking at title and purpose, many fields over overlap, but as for title applicants could write in their responses there are many values with only 1 count. For our purposes, we will drop title and keep purpose, no pun intended, as there are fewer unique values

In [43]:
```
features_to_drop.append('title')
```
executed in 13ms, finished 22:03:25 2021-04-21

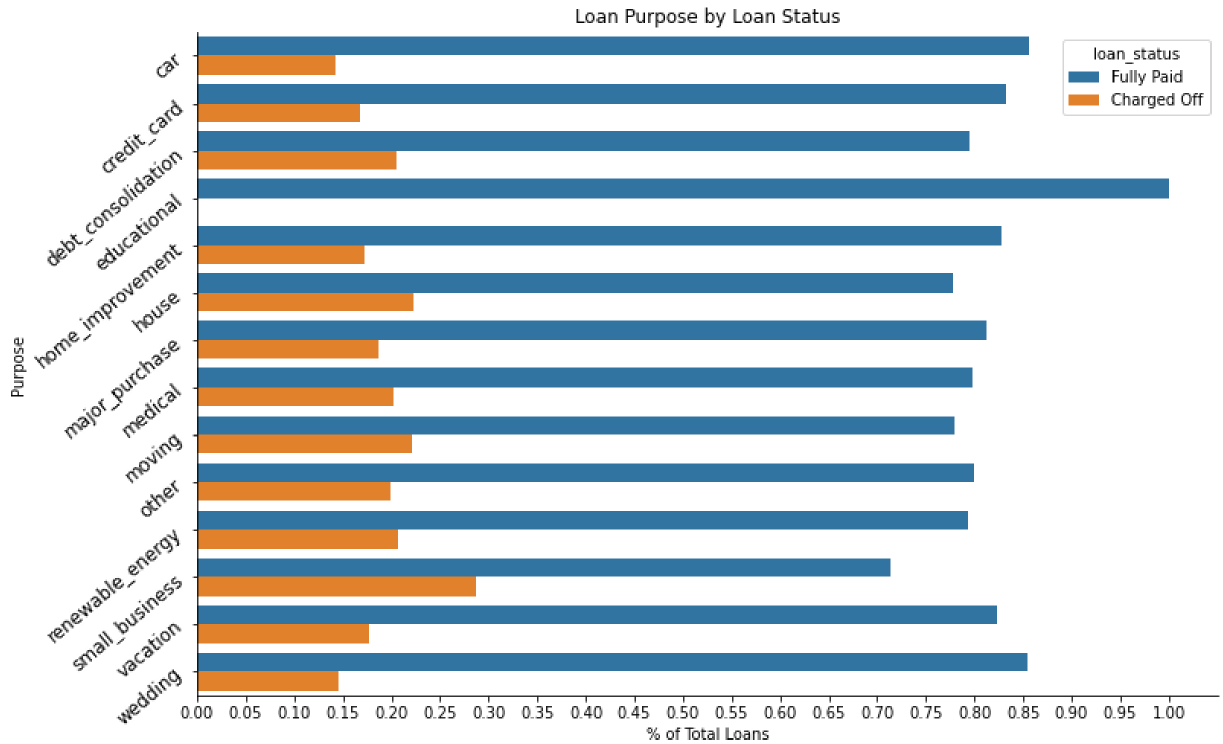
In [121]:
```
loan_statuses = ['Fully Paid','Charged Off']
g = df.groupby('purpose')['loan_status'].value_counts(normalize=True).unstack()
list_dfs = []
for status in loan_statuses:
    vals = g[status].values
    idx = g[status].index
    frame = pd.DataFrame(data=vals,index=idx,columns=['value_counts']).reset_inde
    frame['loan_status'] = status
    list_dfs.append(frame)
comb = pd.concat([list_dfs[0],list_dfs[1]])
num = df['purpose'].nunique()
```
executed in 444ms, finished 02:37:49 2021-04-22

In [122]:
```
comb.purpose.values
```
executed in 15ms, finished 02:37:50 2021-04-22

Out[122]:
```
array(['car', 'credit_card', 'debt_consolidation', 'educational',
       'home_improvement', 'house', 'major_purchase', 'medical', 'moving',
       'other', 'renewable_energy', 'small_business', 'vacation',
       'wedding', 'car', 'credit_card', 'debt_consolidation',
       'educational', 'home_improvement', 'house', 'major_purchase',
       'medical', 'moving', 'other', 'renewable_energy', 'small_business',
       'vacation', 'wedding'], dtype=object)
```

In [123]:
```
plt.figure(figsize=(12,8))
sns.barplot(x='value_counts',y='purpose',hue='loan_status',data=comb)
plt.ylabel('Purpose')
plt.xlabel('% of Total Loans')
plt.xticks(np.linspace(0,1,21))
plt.title('Loan Purpose by Loan Status')
plt.yticks(rotation=40,fontsize=12)
sns.despine()
plt.show()
#ax1.set_yticklabels()yticklabels(comb.purpose.values,rotation=35)
plt.tight_layout()
plt.savefig('purpose.png')
```

executed in 379ms, finished 02:37:52 2021-04-22

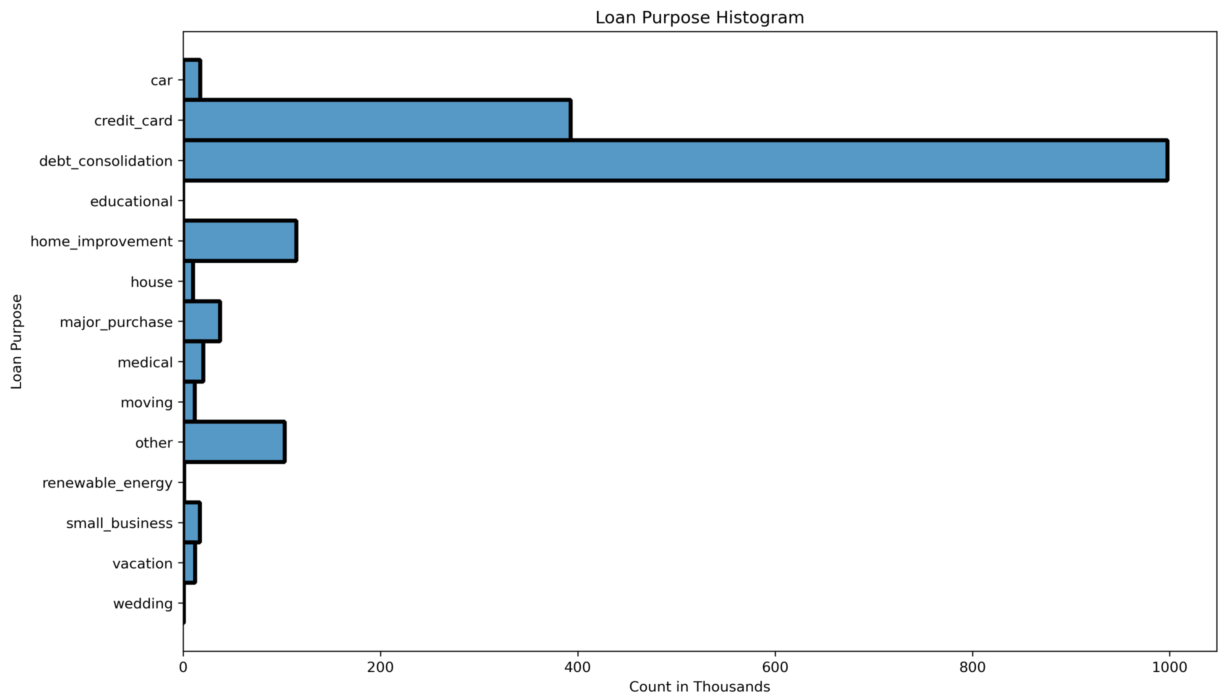



```
<Figure size 432x288 with 0 Axes>
```

In [118]:

```
plt.figure(figsize=(13,8),dpi=300)
sns.histplot(y='purpose', data=df.sort_values('purpose'),bins=(num),line_kws={'me
plt.ylabel('Loan Purpose')
plt.xlabel('Count in Thousands')
plt.title('Loan Purpose Histogram')
plt.gca().xaxis.set_major_formatter(FuncFormatter(lambda x, _: int(round(x,0)/100
plt.show()
plt.savefig('loan_purpose.png')
```

executed in 3.31s, finished 02:35:51 2021-04-22




```
<Figure size 432x288 with 0 Axes>
```

Can see that a vast majority of the loans are for debt consolidation followed by credit card

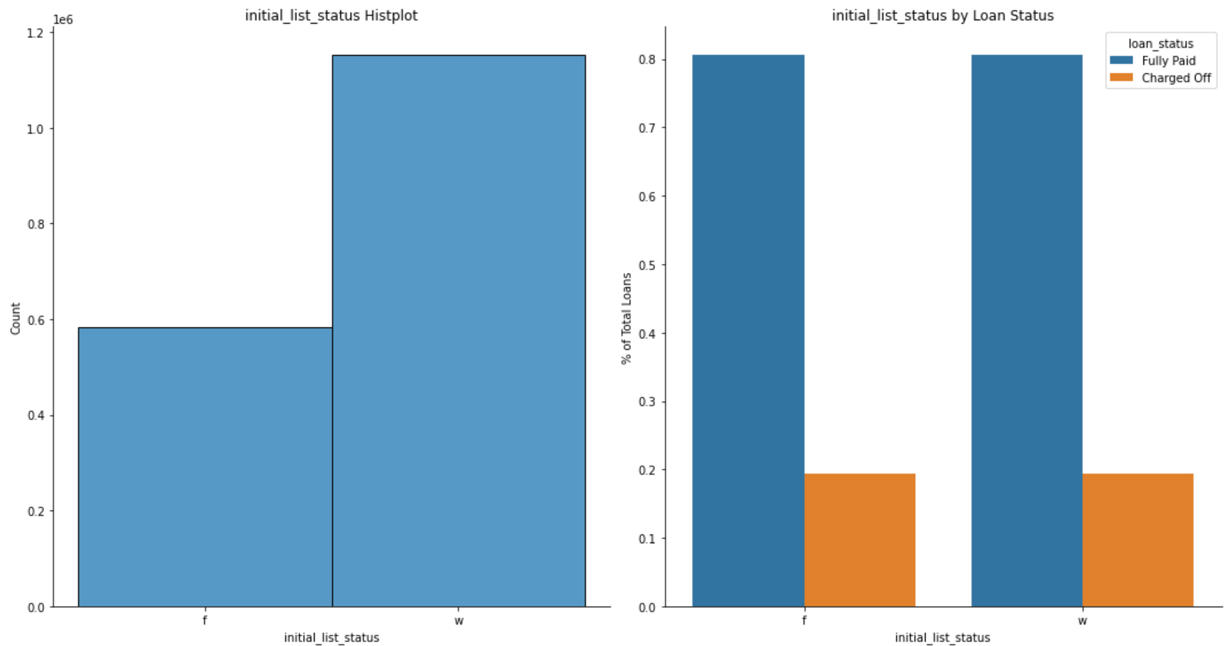
1.7 Initial Status

In [48]:
```
column_info('initial_list_status')
```
executed in 14ms, finished 22:03:37 2021-04-21

Out[48]:
```
'The initial listing status of the loan. Possible values are - W, F'
```

After doing research, W and F here stand for whole or fractional loans but from the perspective of the investor. As the loans are chosen at random they don't influence our data as we can see in our barplots that the percentages for charged off rate and fully paid are the same

In [49]:
```
plot_cats('initial_list_status')
```
executed in 4.18s, finished 22:03:42 2021-04-21



In [50]:
```
features_to_drop.append('initial_list_status')
```
executed in 14ms, finished 22:03:43 2021-04-21

1.8 States and Zipcodes

In [51]:
```
column_info('zip_code')
```
executed in 14ms, finished 22:03:44 2021-04-21

Out[51]:
```
'The first 3 numbers of the zip code provided by the borrower in the loan appli
cation.'
```

In [52]:
```
df['zip_code'].nunique()
```
executed in 125ms, finished 22:03:45 2021-04-21

Out[52]:
```
947
```

In [53]:
```
column_info('addr_state')
```
executed in 13ms, finished 22:03:46 2021-04-21

Out[53]:
```
'The state provided by the borrower in the loan application'
```

In [54]:
```
#includes washington DC here to explain for 51
df['addr_state'].value_counts()
```
executed in 171ms, finished 22:03:47 2021-04-21

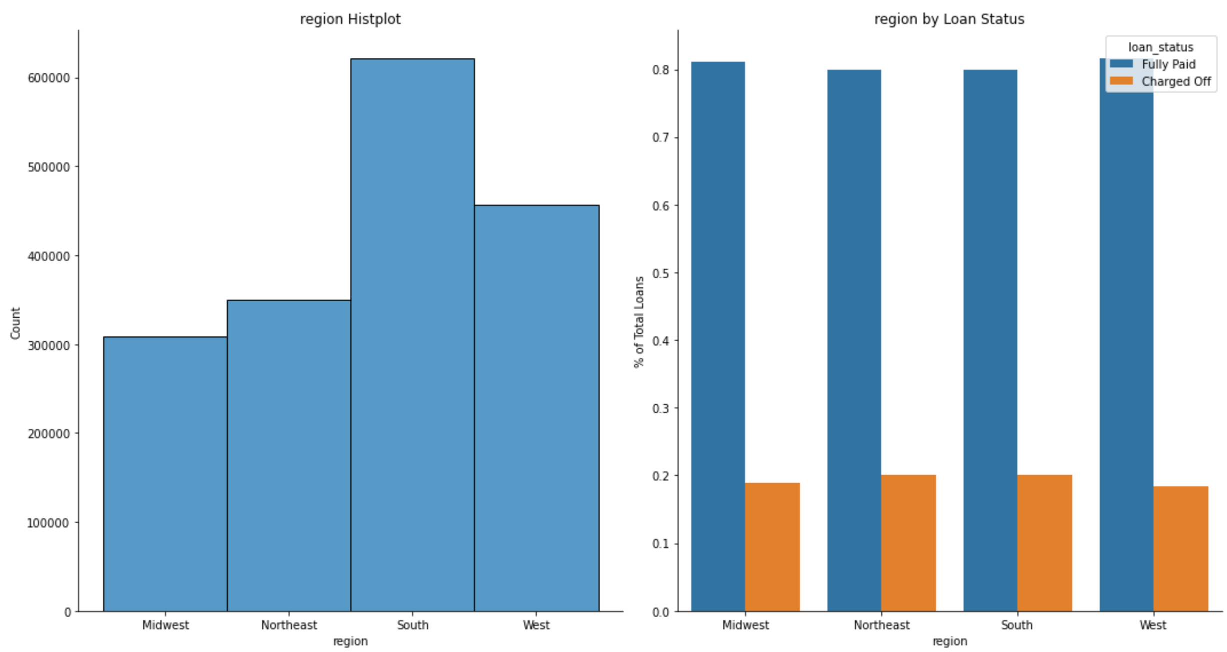
Out[54]:
```
CA    245034
TX    143643
NY    139854
FL    124679
IL     68114
NJ     62247
PA     58423
OH     57216
GA     56445
NC     48688
VA     47940
MI     45736
AZ     42741
MD     40493
MA     39678
CO     38195
WA     37058
MN     30985
IN     29736
TN     28057
MO     27747
NV     26183
CT     26138
WI     23247
AL     21176
OR     21093
SC     20867
LA     19542
KY     16761
OK     15871
KS     14430
AR     13143
UT     12419
MS      9487
NM      9362
NH      8520
HI      8403
RI      7677
WV      6122
NE      5739
MT      4935
DE      4904
DC      4130
AK      4036
WY      3707
VT      3638
SD      3532
ME      3460
ID      3142
ND      2563
IA         1
Name: addr_state, dtype: int64
```

As there are too many zipcodes to encode for our models we will use states and group by region

In [55]:
```
regions = pd.read_excel('data/state_regions.xlsx')
```
executed in 29ms, finished 22:03:49 2021-04-21

In [56]:
```
df['region'] = df.addr_state.apply(lambda x: regions.loc[regions['State Code'] ==
```
executed in 11m 56s, finished 22:15:46 2021-04-21

In [57]:
```
plot_cats('region')
```
executed in 5.16s, finished 22:15:52 2021-04-21



In [58]:
```
features_to_drop.append('zip_code')
```
executed in 13ms, finished 22:15:53 2021-04-21

In [59]:
```
features_to_drop.append('addr_state')
```
executed in 14ms, finished 22:15:54 2021-04-21

Lending Club loans have a high distribution in the South and the West but these don't seem to affect the performance of the loan grouped by region

1.9 Revol Util

In [60]:
```
column_info('revol_util')
```
executed in 12ms, finished 22:15:55 2021-04-21

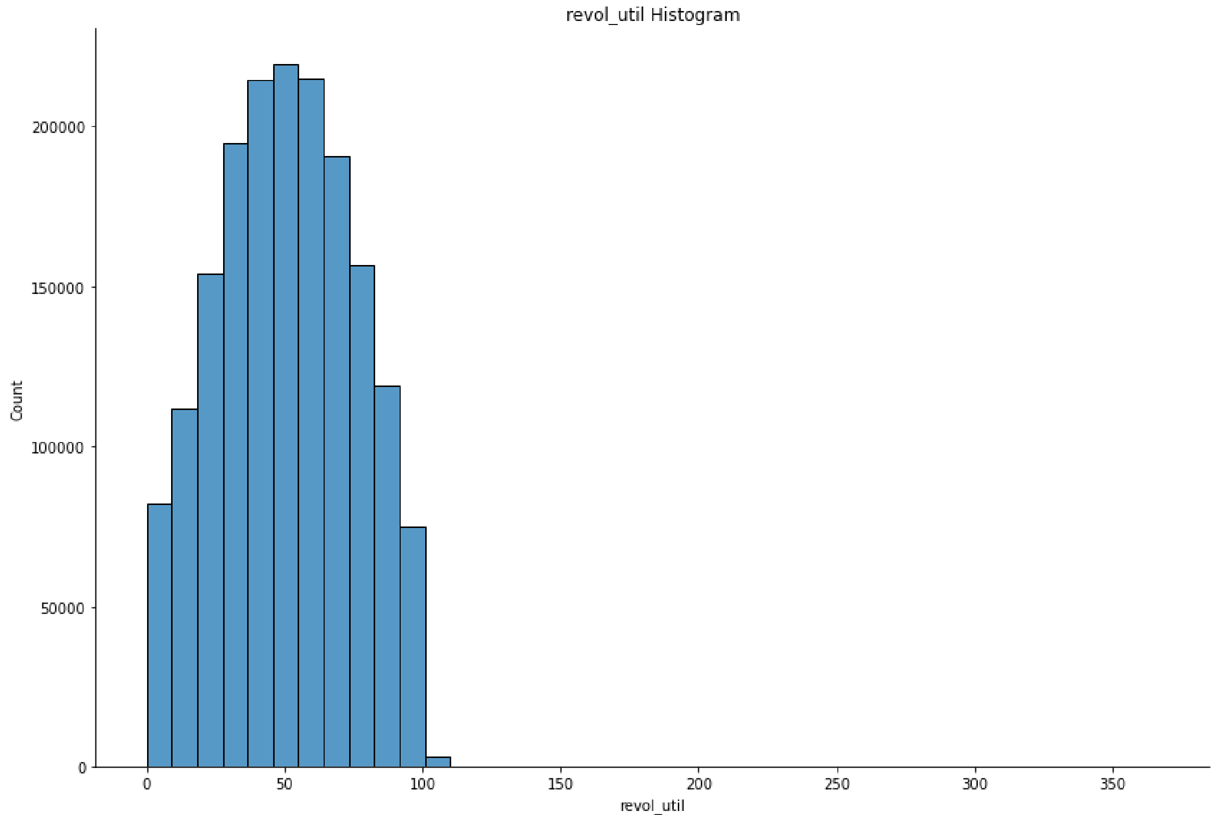
Out[60]:
```
'Revolving line utilization rate, or the amount of credit the borrower is using
relative to all available revolving credit.'
```

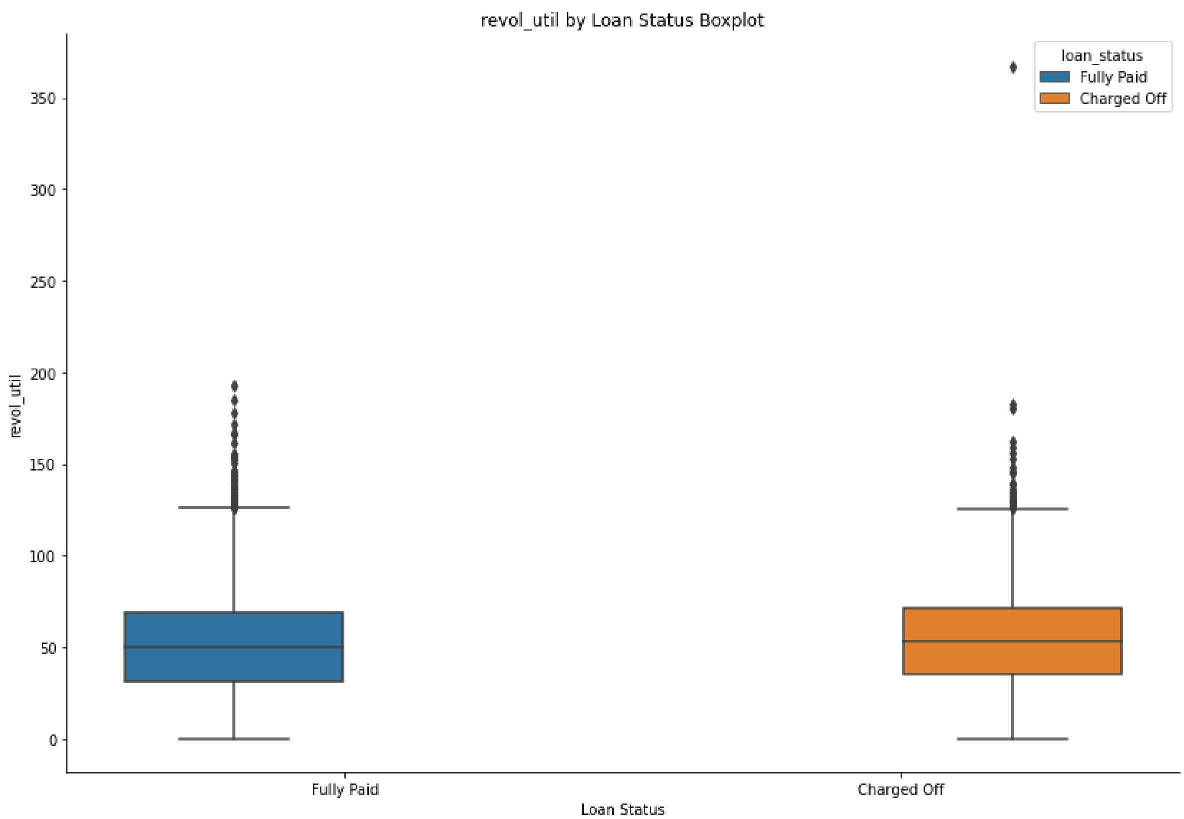
In [61]:
```
df.revol_util.value_counts()
```
executed in 235ms, finished 22:15:55 2021-04-21

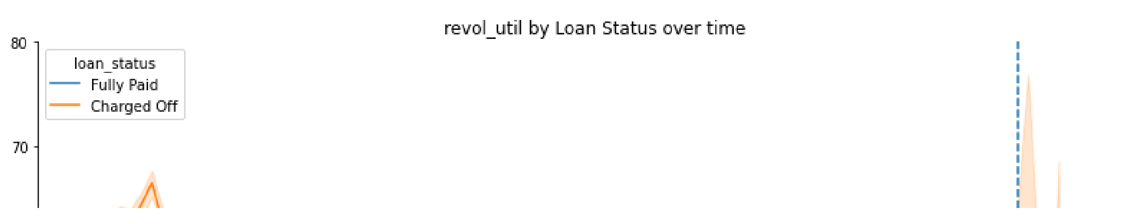
Out[61]:
```
0%        7980
48%       3358
57%       3356
59%       3331
58%       3329
          ...
132.2%       1
180.3%       1
129.4%       1
150.7%       1
123.5%       1
Name: revol_util, Length: 1304, dtype: int64
```

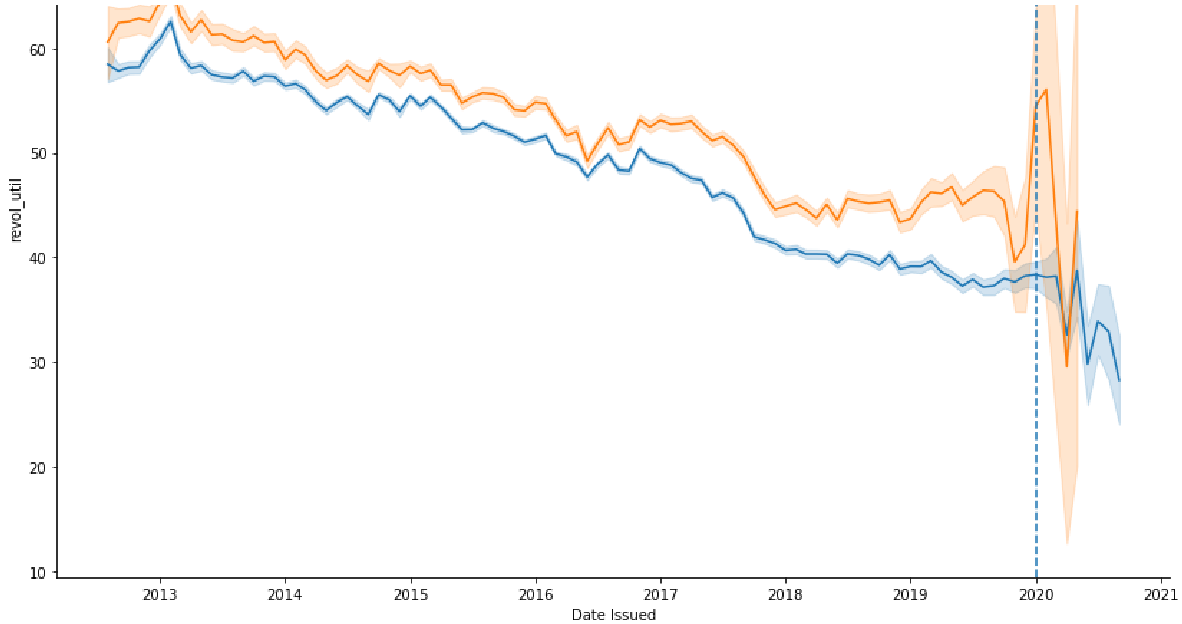
In [62]:
```
df.revol_util = df.revol_util.map(lambda x: np.float(x.replace('%','')))
```
executed in 1.49s, finished 22:15:59 2021-04-21

In [63]:
```
continuous_plot('revol_util')
```
executed in 21.1s, finished 22:16:21 2021-04-21









Main observations here are that charged off loans had higher revovolving balance utilizations but not by much more, this trend holds true over time

In [64]:
```
df[df.loan_status =='Fully Paid']['revol_util'].describe()
```
executed in 986ms, finished 22:16:23 2021-04-21

Out[64]:
```
count    1.399842e+06
mean     4.983598e+01
std      2.470900e+01
min      0.000000e+00
25%      3.090000e+01
50%      4.970000e+01
75%      6.890000e+01
max      1.930000e+02
Name: revol_util, dtype: float64
```

In [65]:
```
df[df.loan_status =='Charged Off']['revol_util'].describe()
```
executed in 393ms, finished 22:16:24 2021-04-21

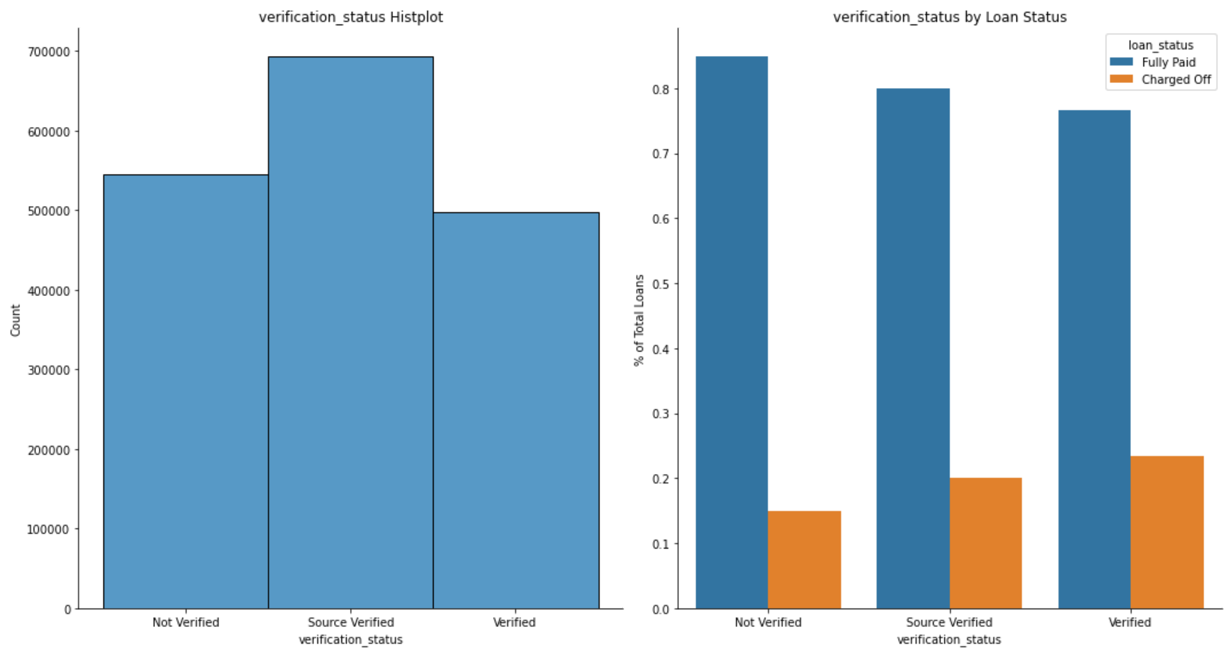
Out[65]:
```
count    337095.000000
mean         52.936573
std          24.029543
min           0.000000
25%          35.200000
50%          53.400000
75%          71.300000
max         366.600000
Name: revol_util, dtype: float64
```

1.10 Verification Status

In [66]:
```
column_info('verification_status')
```
executed in 15ms, finished 22:16:25 2021-04-21

Out[66]:
```
'Indicates if income was verified by LC, not verified, or if the income source was verified'
```

In [67]:
```
plot_cats('verification_status')
```
executed in 4.83s, finished 22:16:31 2021-04-21



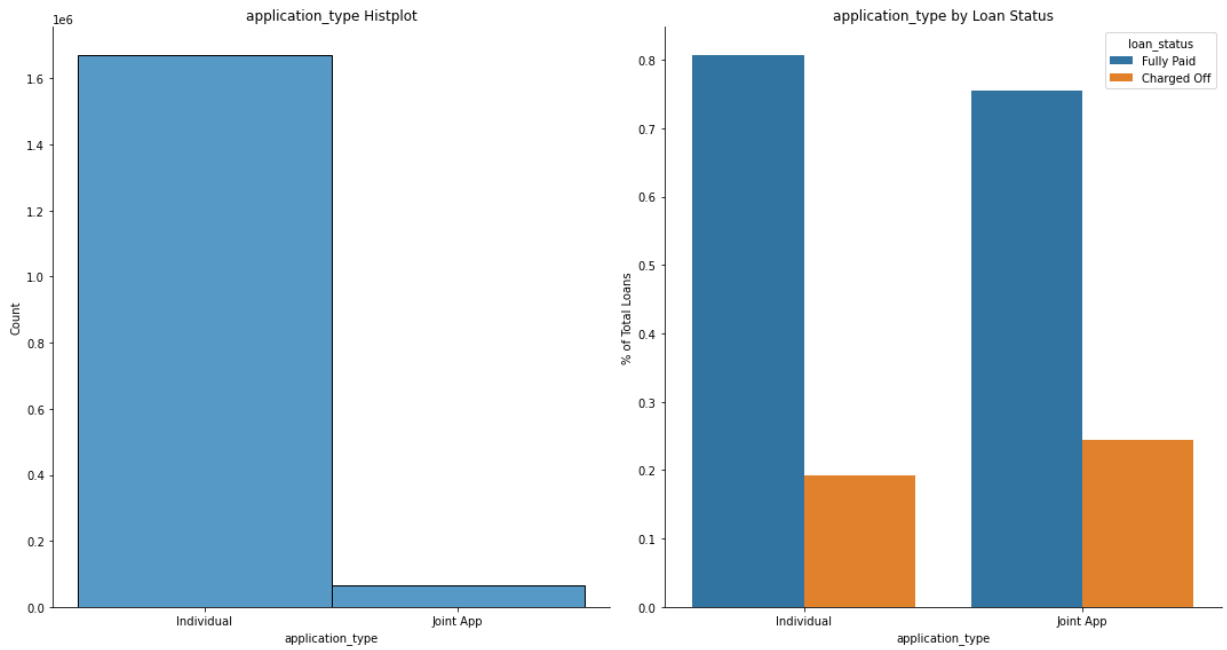
Contrary to logic, verified loans seem to have a higher rate of charge off than non-verified loans

1.11 Application Type

In [68]:
```
column_info('application_type')
```
executed in 14ms, finished 22:16:32 2021-04-21

Out[68]:
```
'Indicates whether the loan is an individual application or a joint application with two co-borrowers'
```

In [69]:
```
plot_cats('application_type')
```
executed in 4.26s, finished 22:16:36 2021-04-21



Majority of the loans in our data are Individual loans, but joint applications are have higher charge off rates on average

1.12 Interest Rate

In [70]:
```
column_info('int_rate')
```
executed in 14ms, finished 22:16:38 2021-04-21

Out[70]: 'Interest Rate on the loan'

In [71]:
```
df.int_rate.value_counts
```
executed in 14ms, finished 22:16:38 2021-04-21

Out[71]:
```
<bound method IndexOpsMixin.value_counts of 0           10.99%
1           10.99%
2           10.99%
3            7.62%
4           12.85%
             ...
1736932     23.99%
1736933      7.99%
1736934     16.99%
1736935     11.44%
1736936     25.49%
Name: int_rate, Length: 1736937, dtype: object>
```

In [72]:
```
df.int_rate = df.int_rate.map(lambda x: np.float(x.replace('%','')))
```
executed in 1.33s, finished 22:16:41 2021-04-21

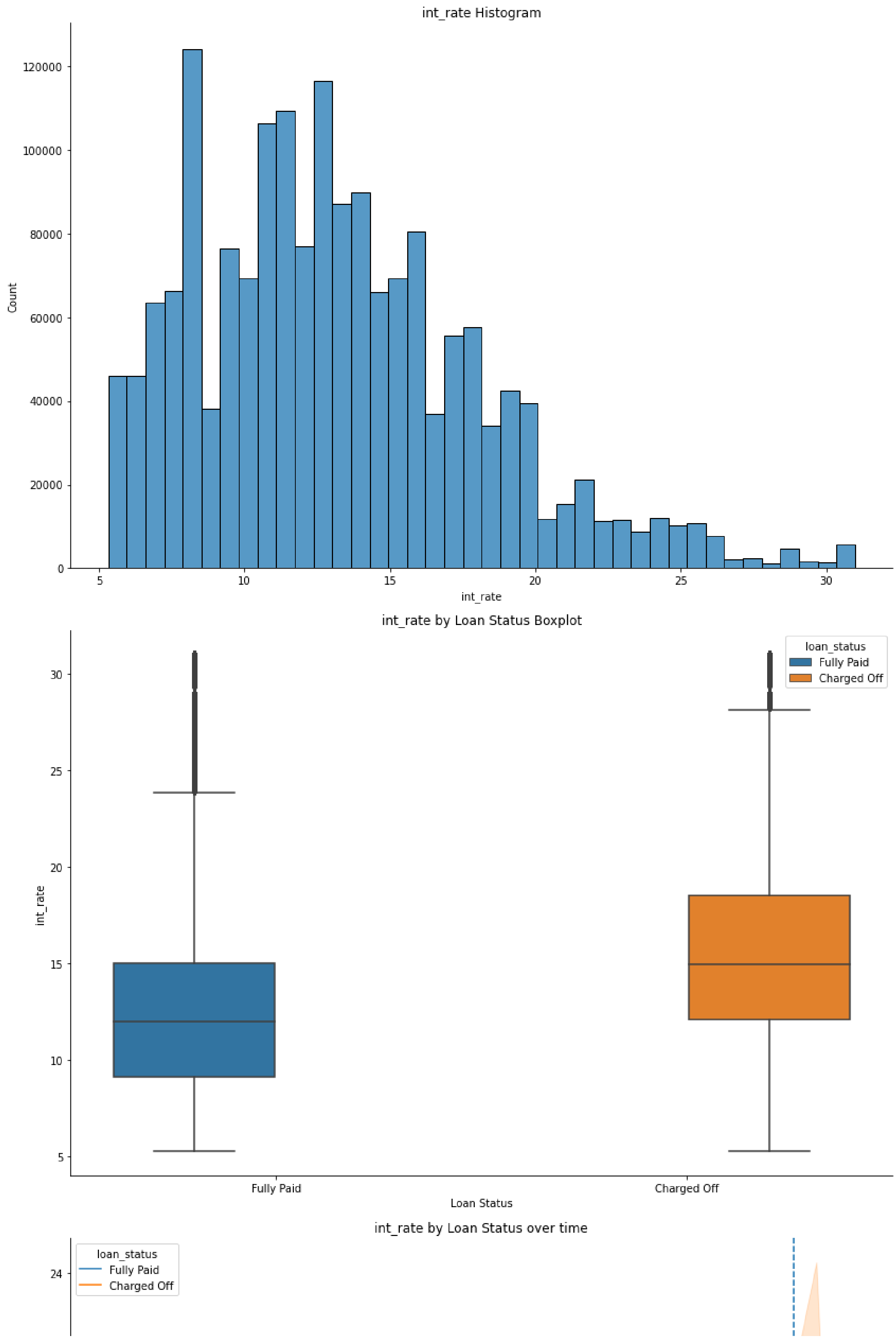
In [114]:
```
df[df.loan_status == "Charged Off"].int_rate.describe()
```
executed in 470ms, finished 02:20:06 2021-04-22

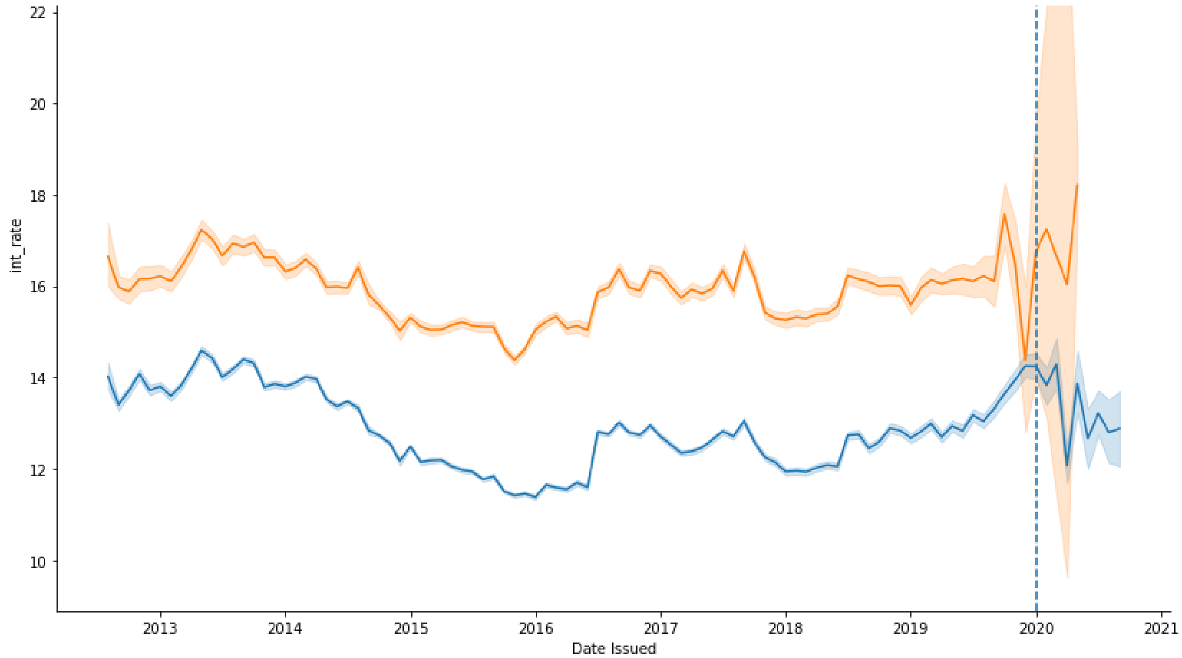
Out[114]:
```
count    337095.000000
mean         15.664992
std           5.022668
min           5.310000
25%          12.120000
50%          14.990000
75%          18.550000
max          30.990000
Name: int_rate, dtype: float64
```

In [113]:
```
df[df.loan_status == "Fully Paid"].int_rate.describe()
```
executed in 1.17s, finished 02:19:50 2021-04-22

Out[113]:
```
count    1.399842e+06
mean     1.256354e+01
std      4.596495e+00
min      5.310000e+00
25%      9.160000e+00
50%      1.199000e+01
75%      1.505000e+01
max      3.099000e+01
Name: int_rate, dtype: float64
```

In [110]:
```python
continuous_plot('int_rate')
plt.savefig('interest.png')
```
executed in 14.7s, finished 01:50:06 2021-04-22





A majority of the loans bear an interest rate of 5% - 15%, and as expected higher interest bearing loans have higher charge of likelihoods

1.13 Debt to Income Ratio (DTI)

In [79]:
```
column_info('dti')
```
executed in 13ms, finished 22:19:50 2021-04-21

Out[79]:
```
'A ratio calculated using the borrower’s total monthly debt payments on the tot
al debt obligations, excluding mortgage and the requested LC loan, divided by t
he borrower’s self-reported monthly income.'
```

In [80]:
```
df.dti.describe()
```
executed in 125ms, finished 22:19:51 2021-04-21

Out[80]:
```
count    1.736937e+06
mean     1.872237e+01
std      1.317191e+01
min     -1.000000e+00
25%      1.196000e+01
50%      1.786000e+01
75%      2.448000e+01
max      9.990000e+02
Name: dti, dtype: float64
```

In [81]:
```
df.dti.value_counts()
```
executed in 78ms, finished 22:19:52 2021-04-21

Out[81]:
```
16.799999     1724
19.200001     1643
17.700001     1593
16.879999     1587
16.270000     1580
              ...
131.899994       1
758.500000       1
756.500000       1
449.200012       1
201.800003       1
Name: dti, Length: 5199, dtype: int64
```

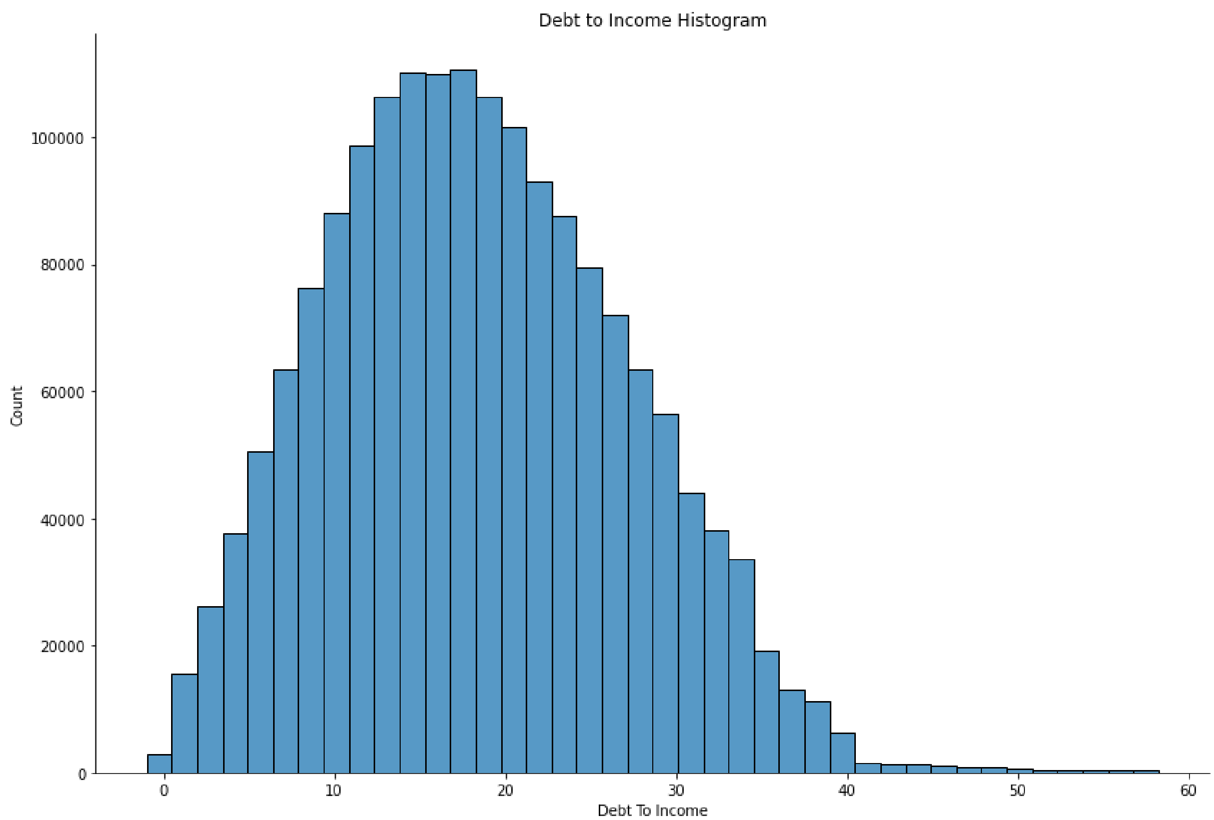
In [82]:
```
#df_cont_z = df[(np.abs(stats.zscore(df[cont_columns]))<4).all(axis=1)]
```
executed in 13ms, finished 22:19:53 2021-04-21

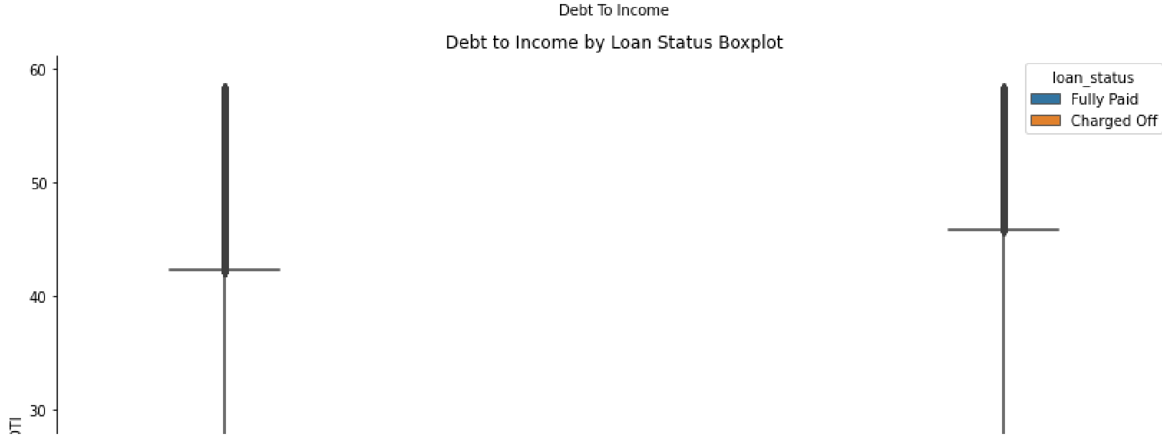
Upon inpsection, there we some extreme outliers that were skewing our plots so the below plots are having dropped values for DTI outside 3 standard deviations from the mean. This will need to be considered for modeling

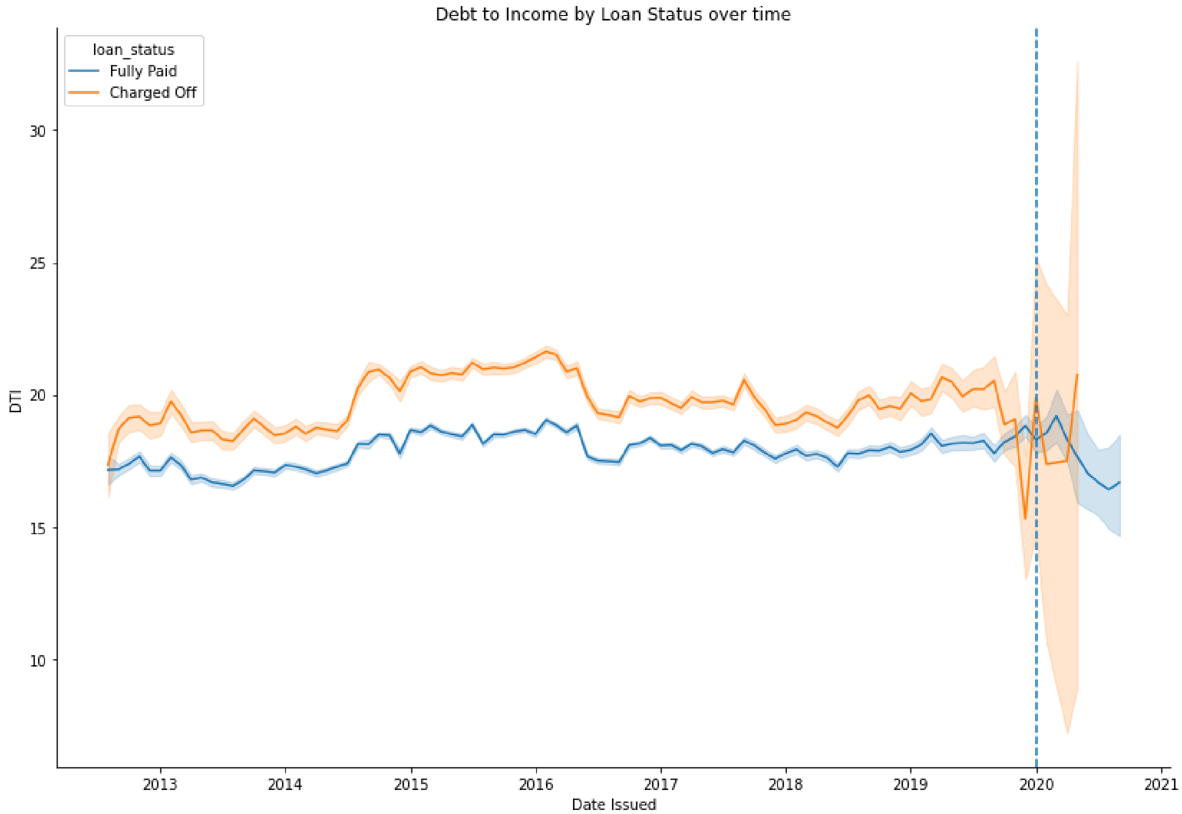
In [85]:
```
from scipy import stats
```
executed in 14ms, finished 22:20:12 2021-04-21

In [115]:
```
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
sns.histplot(x='dti',data=df[(np.abs(stats.zscore(df['dti']))<3)],bins=40,ax=ax1)
ax1.set_xlabel("Debt To Income")
ax1.set_ylabel('Count')
ax1.set_title('Debt to Income Histogram')

sns.despine()
sns.boxplot(x='loan_status' ,y='dti',hue='loan_status',data=df[(np.abs(stats.zsco
ax2.set_xlabel('Loan Status')
ax2.set_ylabel("DTI")
ax2.set_title('Debt to Income by Loan Status Boxplot')

sns.lineplot(x='issue_d' ,y='dti',hue='loan_status',data=df[(np.abs(stats.zscore(
ax3.set_xlabel('Date Issued')
ax3.set_ylabel("DTI")
ax3.set_title('Debt to Income by Loan Status over time')
ax3.axvline(x='2020',linestyle='--')
plt.tight_layout()
plt.savefig('dti.png')
```

executed in 28.9s, finished 02:26:44 2021-04-22







Most loans are between 10-20 DTI and there is a slightly higher Charged Off rate for higher DTI's with fully paid loans at 18% vs 20% for charged off loans

In [87]:
```
temp = df[(np.abs(stats.zscore(df['dti']))<3)]
```
executed in 722ms, finished 22:20:42 2021-04-21

In [88]:
```
temp[temp.loan_status =='Fully Paid']['dti'].describe()
```
executed in 805ms, finished 22:20:45 2021-04-21

Out[88]:
```
count     1.396292e+06
mean      1.803052e+01
std       8.577942e+00
min      -1.000000e+00
25%       1.164000e+01
50%       1.740000e+01
75%       2.390000e+01
max       5.822000e+01
Name: dti, dtype: float64
```

In [89]:
```
temp[temp.loan_status =='Charged Off']['dti'].describe()
```
executed in 425ms, finished 22:20:47 2021-04-21

Out[89]:
```
count    335658.000000
mean         20.024944
std           8.981527
min           0.000000
25%          13.410000
50%          19.719999
75%          26.400000
max          58.220001
Name: dti, dtype: float64
```

1.14 Public Record Bankruptcies

In [90]:
```
column_info('pub_rec_bankruptcies')
```
executed in 14ms, finished 22:20:49 2021-04-21

Out[90]:
```
'Number of public record bankruptcies'
```

In [91]:
```
df[(np.abs(stats.zscore(df['pub_rec_bankruptcies']))<3)]
```
executed in 1.62s, finished 22:20:52 2021-04-21

Out[91]:

	id	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_ler
0	10129454	12000.0	36 months	10.99	392.799988	B	B2	
1	10149488	4800.0	36 months	10.99	157.100006	B	B2	
2	10149342	27060.0	36 months	10.99	885.500000	B	B2	
3	10148122	12000.0	36 months	7.62	374.000000	A	A3	
4	10129477	14000.0	36 months	12.85	470.799988	B	B4	
...	...	...	...	...	...	...	...	
1736932	102556443	24000.0	60 months	23.99	690.500000	E	E2	
1736933	102653304	10000.0	36 months	7.99	313.200012	A	A5	
1736934	102628603	10050.0	36 months	16.99	358.200012	D	D1	
1736935	102196576	6000.0	36 months	11.44	197.800003	B	B4	
1736936	99799684	30000.0	60 months	25.49	889.000000	E	E4	

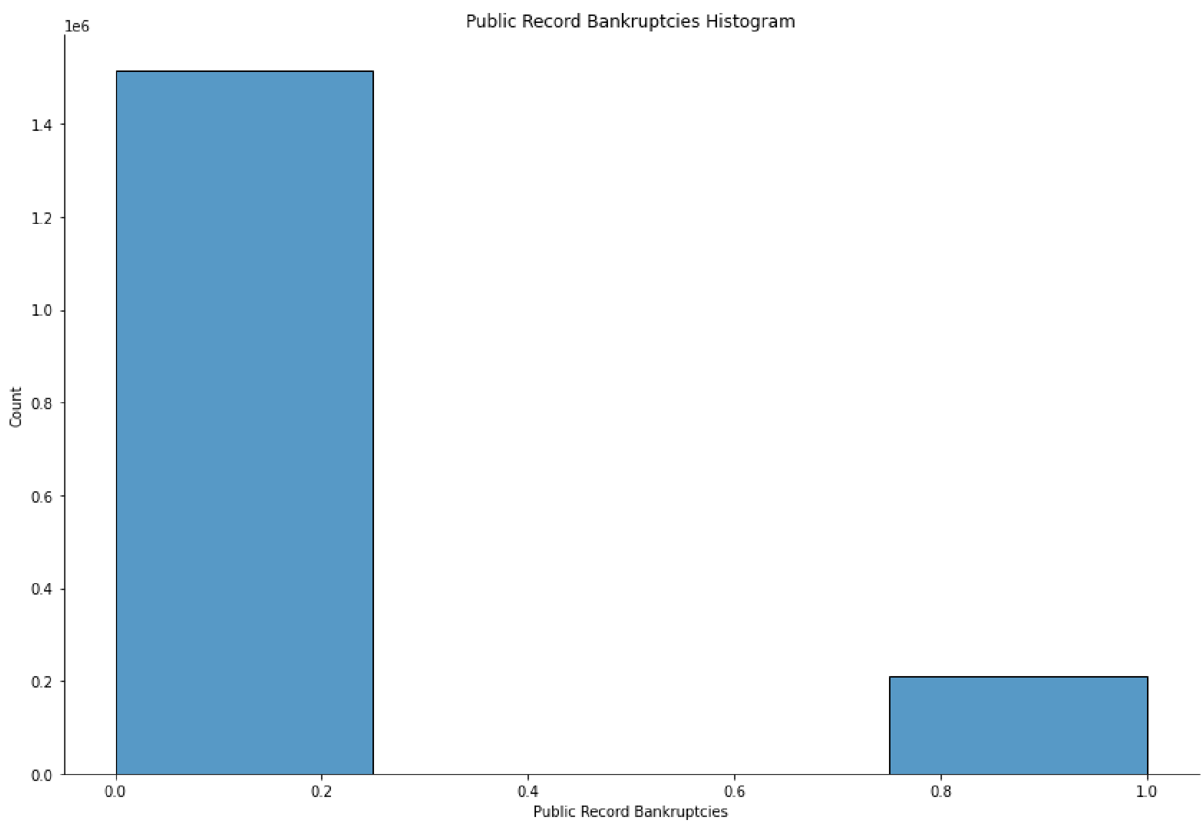
1724900 rows × 75 columns

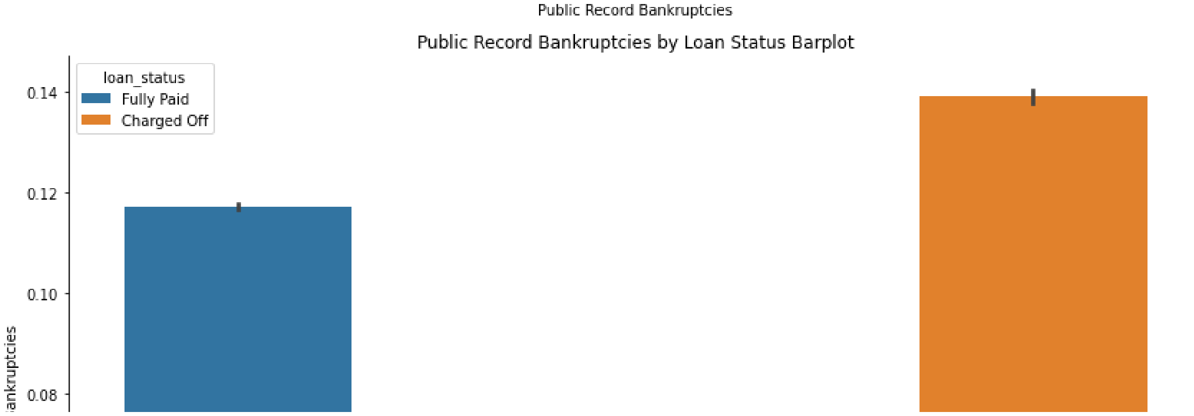
In [92]:
```
df.pub_rec_bankruptcies
```
executed in 14ms, finished 22:20:54 2021-04-21

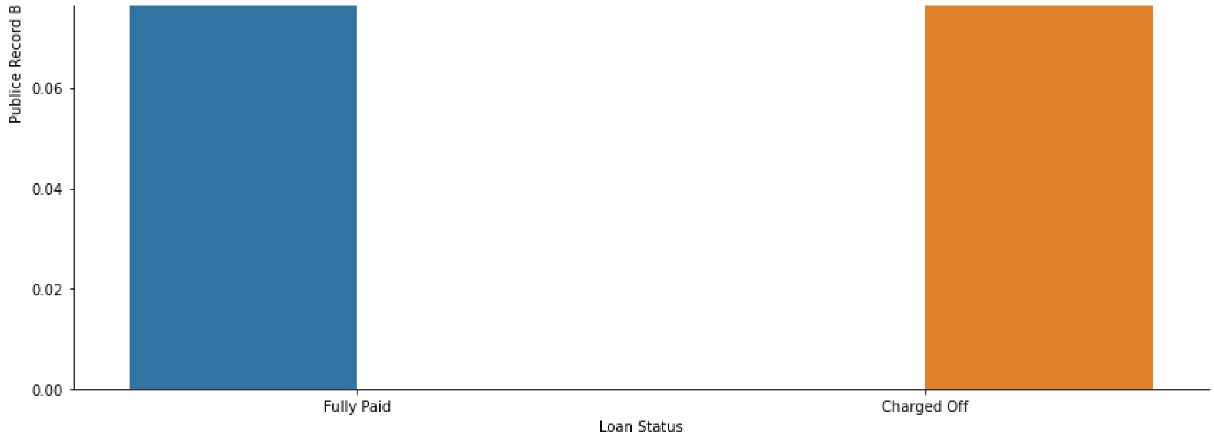
Out[92]:
```
0          0.0
1          0.0
2          0.0
3          0.0
4          1.0
          ...
1736932    1.0
1736933    0.0
1736934    0.0
1736935    0.0
1736936    0.0
Name: pub_rec_bankruptcies, Length: 1736937, dtype: float32
```

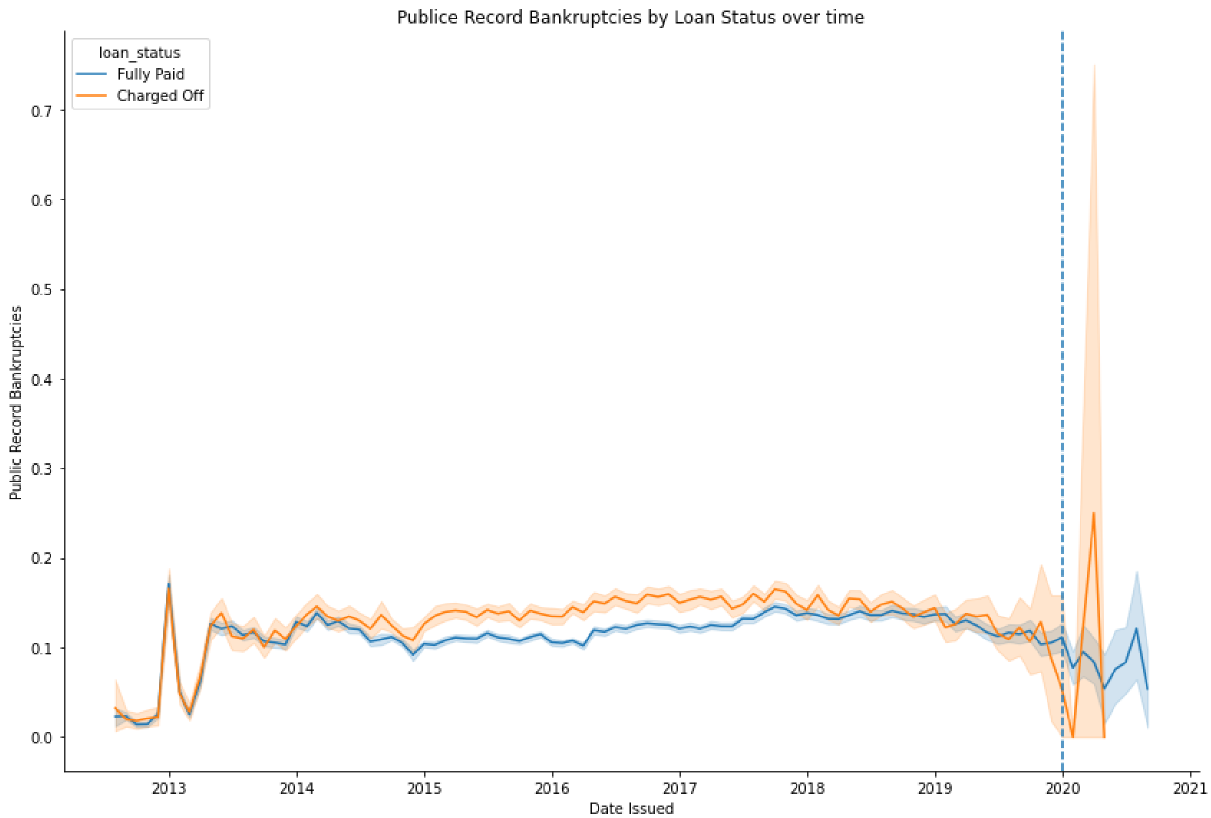
In [93]:
```python
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
sns.histplot(x='pub_rec_bankruptcies',data=df[(np.abs(stats.zscore(df['pub_rec_ba
ax1.set_xlabel("Public Record Bankruptcies")
ax1.set_ylabel('Count')
ax1.set_title('Public Record Bankruptcies Histogram')

sns.despine()
sns.barplot(x='loan_status' ,y='pub_rec_bankruptcies',hue='loan_status',data=df[(
ax2.set_xlabel('Loan Status')
ax2.set_ylabel('Publice Record Bankruptcies')
ax2.set_title('Public Record Bankruptcies by Loan Status Barplot')

sns.lineplot(x='issue_d' ,y='pub_rec_bankruptcies',hue='loan_status',data=df[(np.
ax3.set_xlabel('Date Issued')
ax3.set_ylabel("Public Record Bankruptcies")
ax3.set_title('Publice Record Bankruptcies by Loan Status over time')
ax3.axvline(x='2020',linestyle='--')
plt.tight_layout()
```

executed in 55.1s, finished 22:21:51 2021-04-21









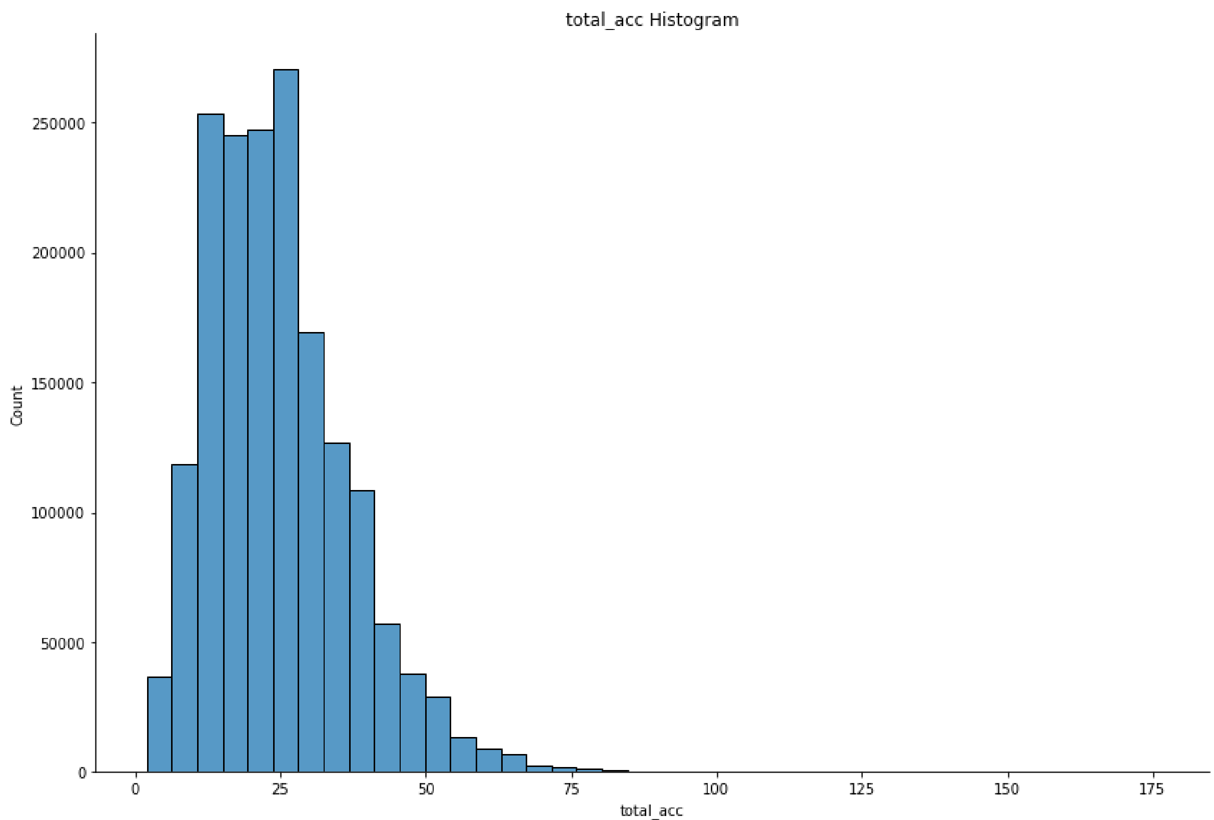
There are around 200k loans that have a public bankruptcy record. The higher average pub rec bankruptcies have a small relationship with higher charge offs

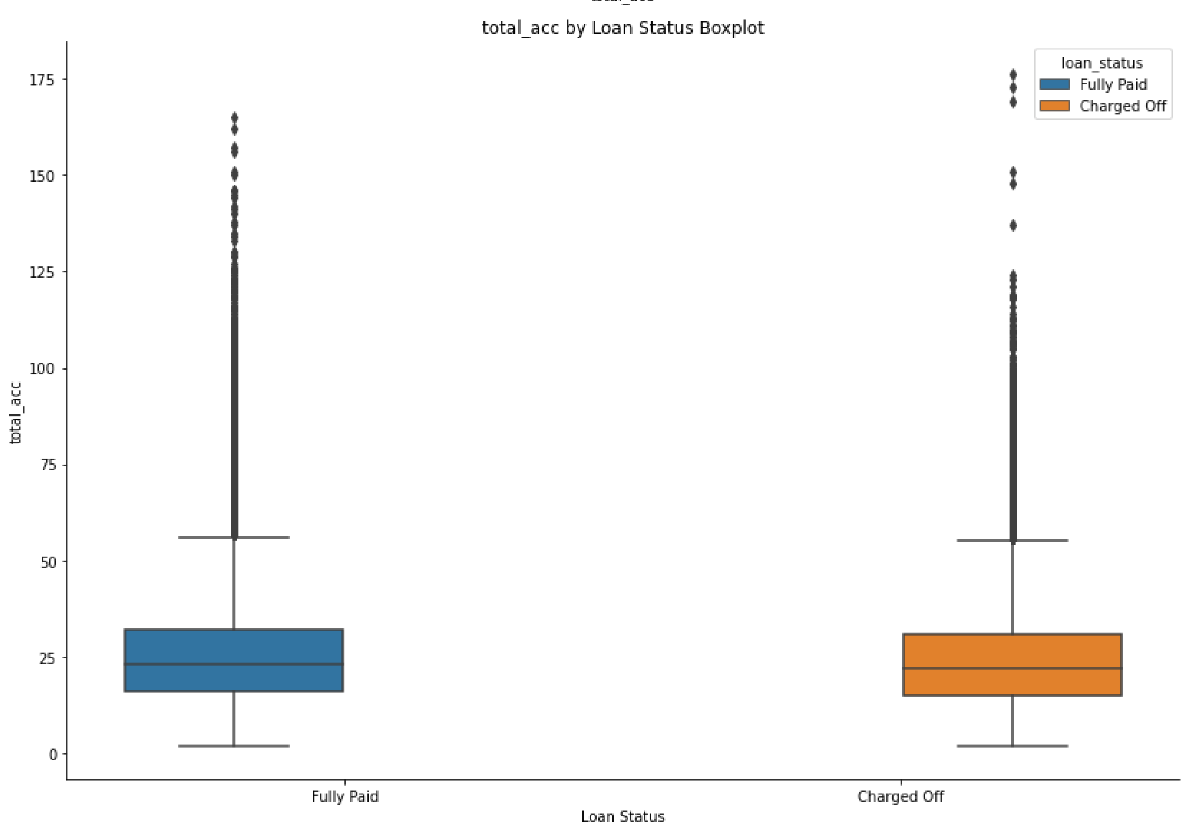
1.15 Total Accounts

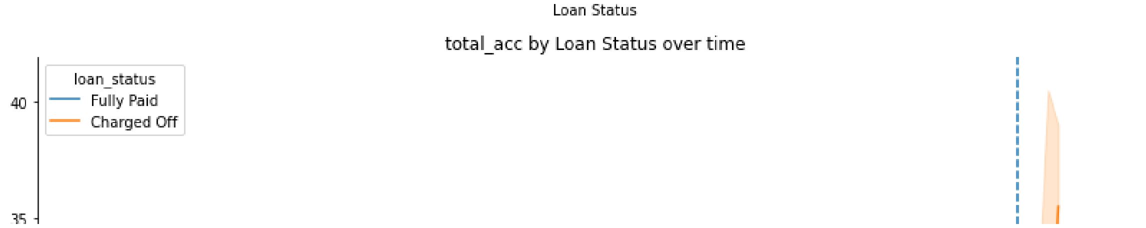
In [94]:
```
column_info('total_acc')
```
executed in 13ms, finished 22:21:53 2021-04-21

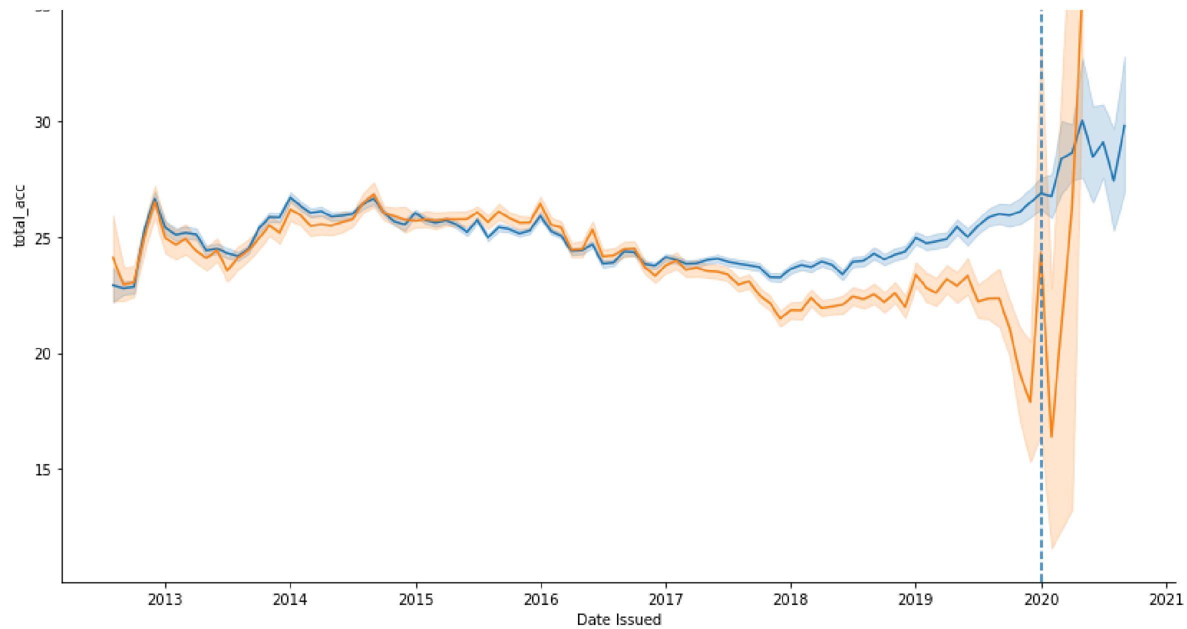
Out[94]: "The total number of credit lines currently in the borrower's credit file"

In [95]:
```
continuous_plot('total_acc')
```
executed in 22.8s, finished 22:22:18 2021-04-21









No clear difference in charge off rates by total accounts

1.16 Annual Income

In [96]:
```
column_info('annual_inc')
```
executed in 14ms, finished 22:22:19 2021-04-21

Out[96]:
```
'The self-reported annual income provided by the borrower during registration.'
```

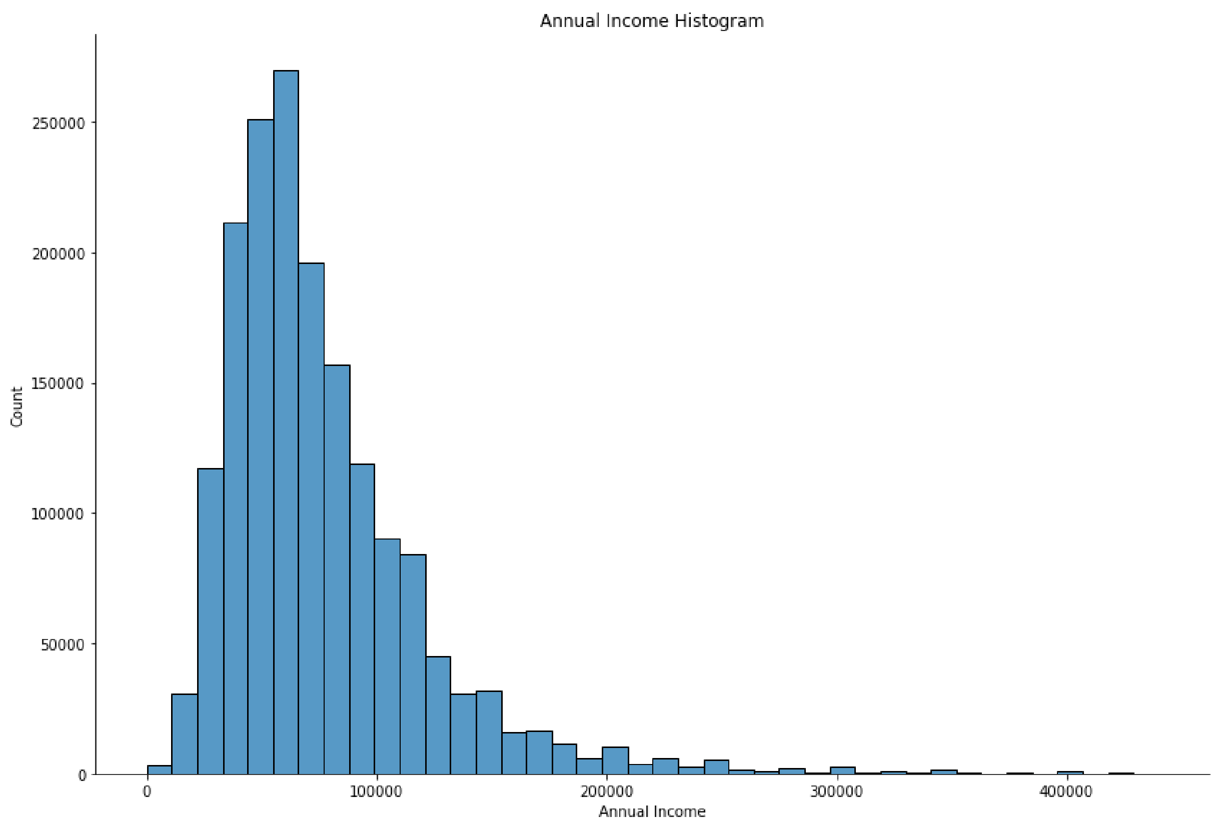
In [97]:
```
df[(np.abs(stats.zscore(df['annual_inc']))<3)]['annual_inc'].describe()
```
executed in 1.05s, finished 22:22:22 2021-04-21

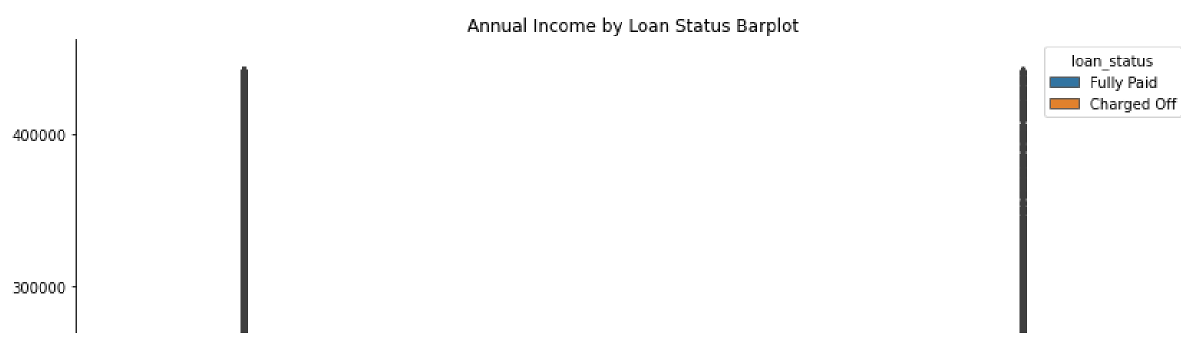
Out[97]:
```
count    1.732761e+06
mean     7.611088e+04
std      4.542821e+04
min      0.000000e+00
25%      4.670000e+04
50%      6.500000e+04
75%      9.200000e+04
max      4.400000e+05
Name: annual_inc, dtype: float64
```

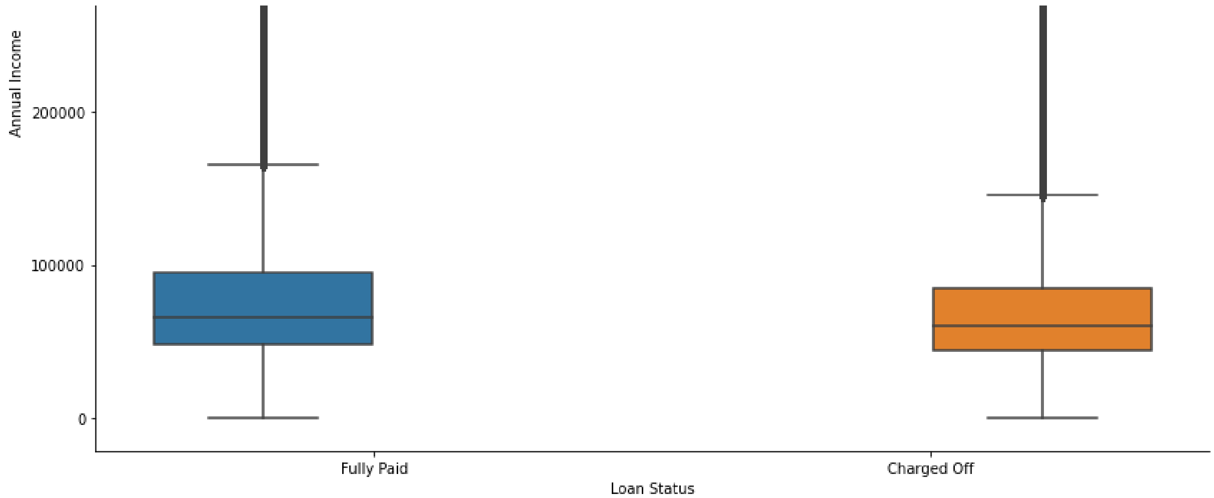
In [116]:

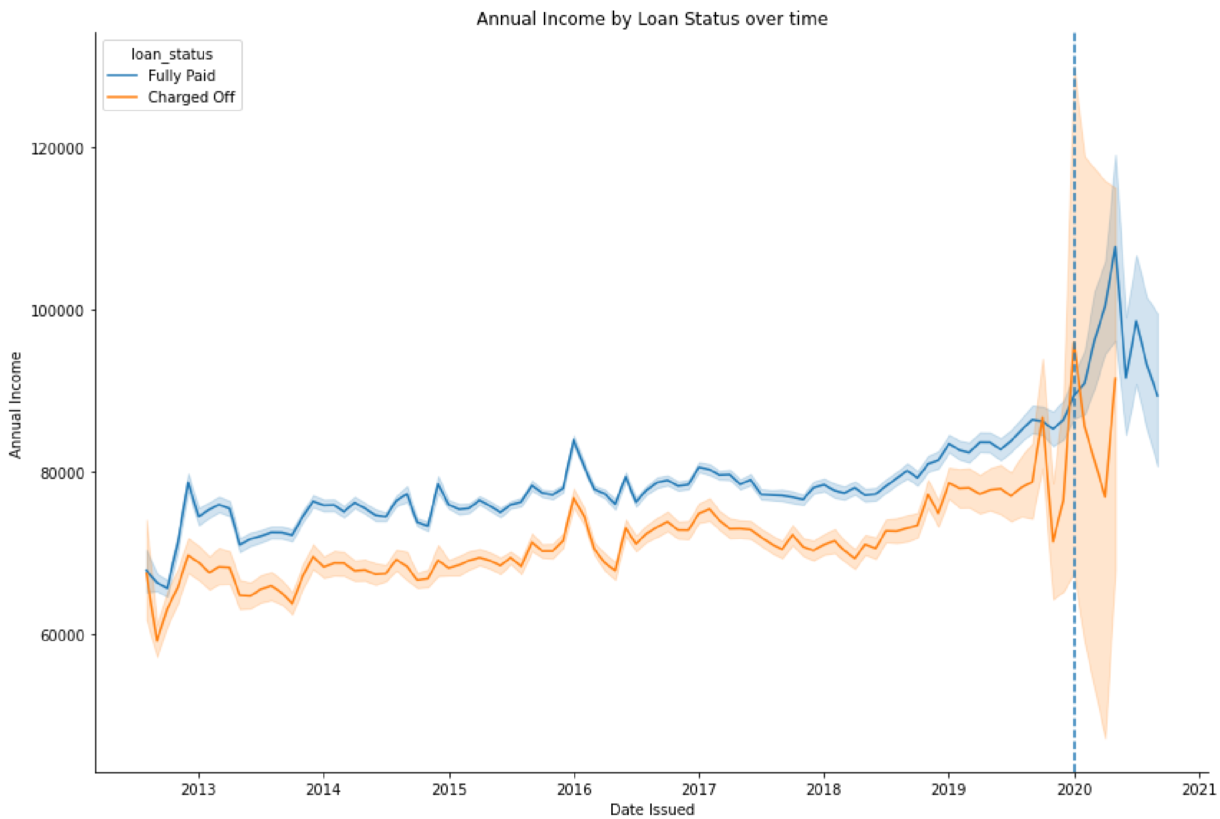
```
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
sns.histplot(x='annual_inc',data=df[(np.abs(stats.zscore(df['annual_inc']))<3)],b
ax1.set_xlabel("Annual Income")
ax1.set_ylabel('Count')
ax1.set_title('Annual Income Histogram')

sns.despine()
sns.boxplot(x='loan_status' ,y='annual_inc',hue='loan_status',
            data=df[(np.abs(stats.zscore(df['annual_inc']))<3)],ax=ax2,)
ax2.set_xlabel('Loan Status')
ax2.set_ylabel('Annual Income')
ax2.set_title('Annual Income by Loan Status Barplot')

sns.lineplot(x='issue_d' ,y='annual_inc',hue='loan_status',
             data=df[(np.abs(stats.zscore(df['annual_inc']))<3)])
ax3.set_xlabel('Date Issued')
ax3.set_ylabel("Annual Income")
ax3.set_title('Annual Income by Loan Status over time')
ax3.axvline(x='2020',linestyle='--')
plt.tight_layout()
plt.savefig('annual_inc.png')
```

executed in 27.8s, finished 02:32:10 2021-04-22









In [99]:
```
temp = df[(np.abs(stats.zscore(df['annual_inc']))<3)]
```
executed in 867ms, finished 22:22:52 2021-04-21

In [100]:
```
temp[temp.loan_status =='Fully Paid']['annual_inc'].describe()
```
executed in 857ms, finished 22:22:54 2021-04-21

Out[100]:
```
count    1.396161e+06
mean     7.737917e+04
std      4.623032e+04
min      0.000000e+00
25%      4.800000e+04
50%      6.600000e+04
75%      9.500000e+04
max      4.400000e+05
Name: annual_inc, dtype: float64
```

In [101]:
```
temp[temp.loan_status =='Charged Off']['annual_inc'].describe()
```
executed in 505ms, finished 22:22:57 2021-04-21

Out[101]:
```
count    336600.000000
mean      70850.304688
std       41526.242188
min          20.000000
25%       44400.000000
50%       60092.500000
75%       85000.000000
max      440000.000000
Name: annual_inc, dtype: float64
```

Can see a normal distribution of incomes (after dropping outliers for 3 std devs) with median income around 70k

Charged off loans had 7k less income

1.17 Total Received Late Fee

In [126]:
```
df.total_rec_late_fee.describe()
```
executed in 112ms, finished 02:53:41 2021-04-22

Out[126]:
```
count    1.736937e+06
mean     2.033143e+00
std      1.415953e+01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      1.599000e+03
Name: total_rec_late_fee, dtype: float64
```

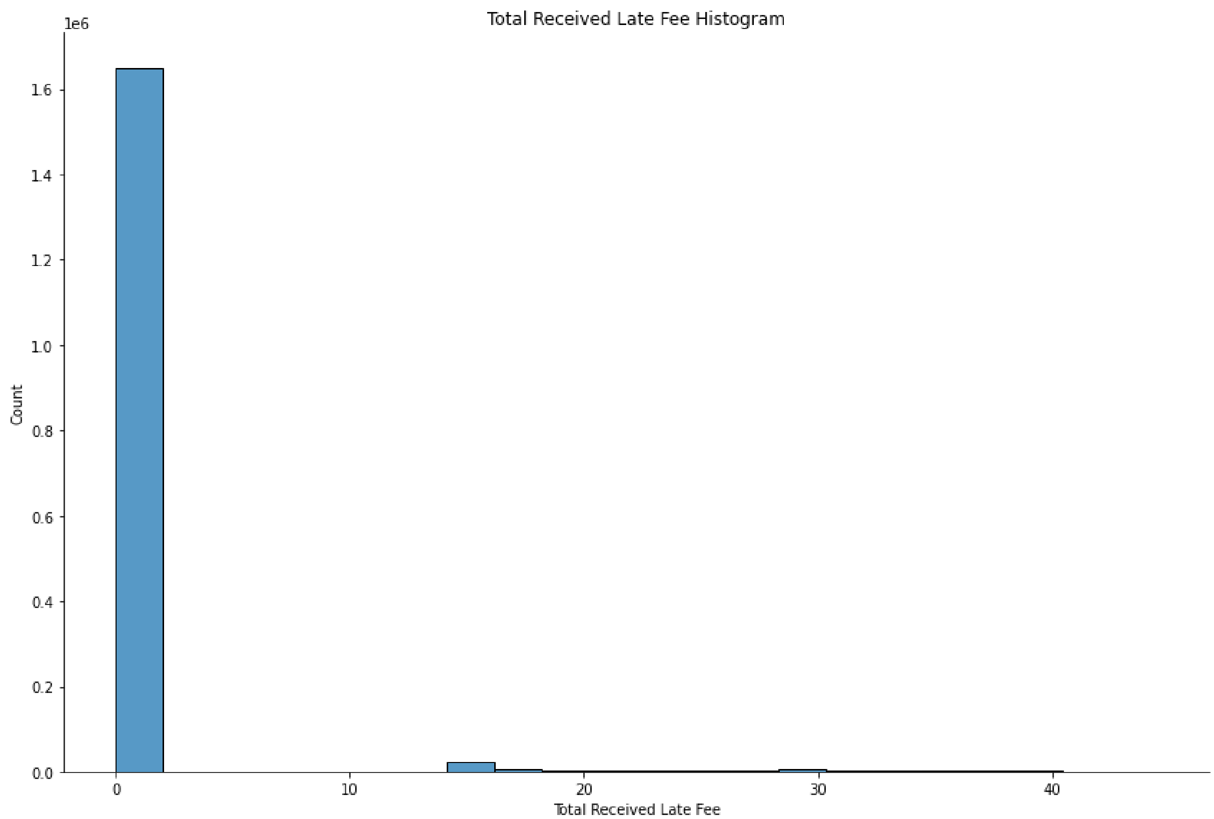
In [127]:
```
column_info('total_rec_late_fee')
```
executed in 19ms, finished 02:54:02 2021-04-22

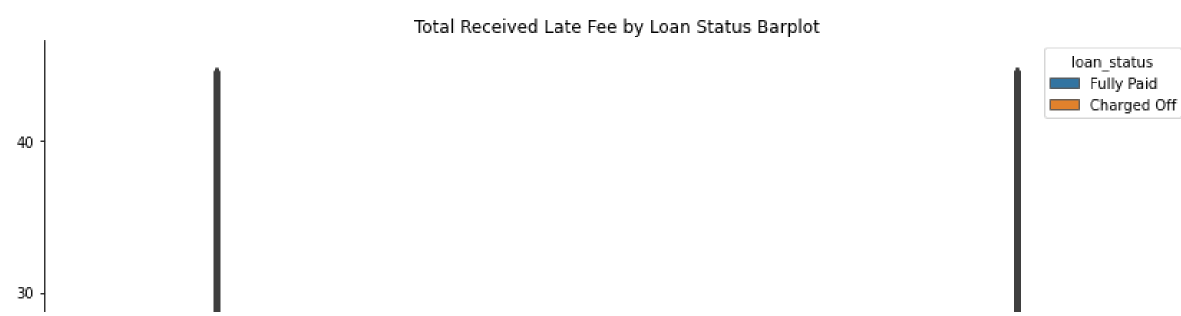
Out[127]:
```
'Late fees received to date'
```

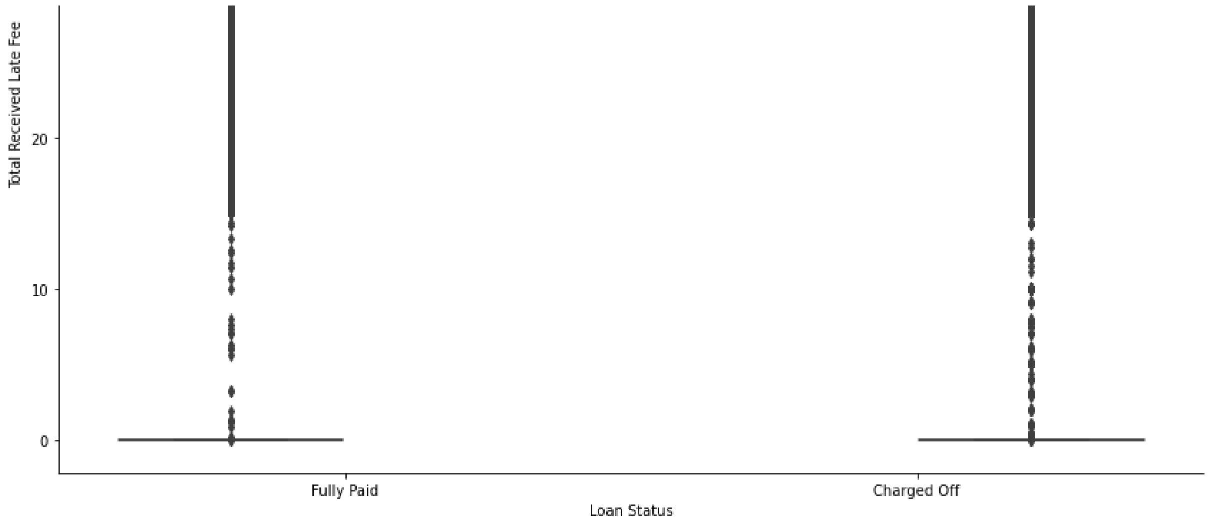
In [134]:

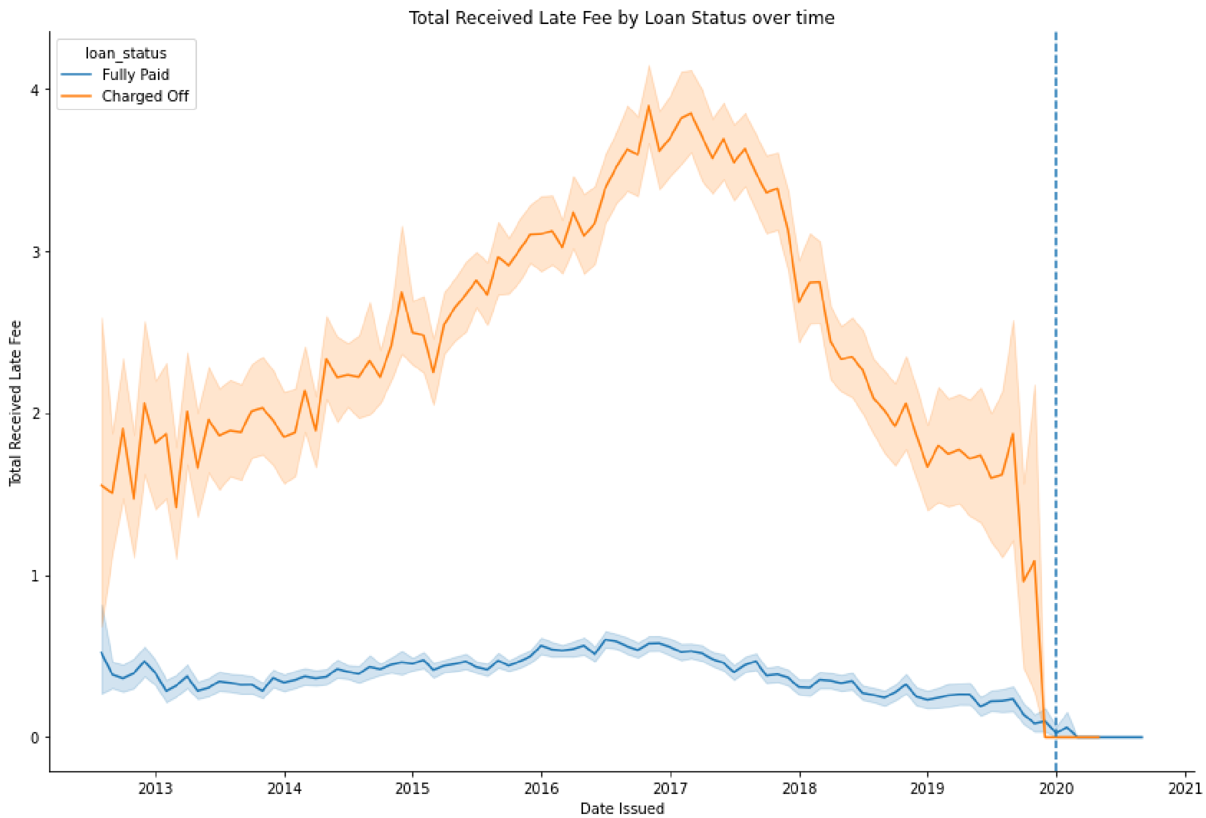
```
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
sns.histplot(x='total_rec_late_fee',data=df[(np.abs(stats.zscore(df['total_rec_la
ax1.set_xlabel("Total Received Late Fee")
ax1.set_ylabel('Count')
ax1.set_title('Total Received Late Fee Histogram')

sns.despine()
sns.boxplot(x='loan_status' ,y='total_rec_late_fee',hue='loan_status',
            data=df[(np.abs(stats.zscore(df['total_rec_late_fee']))<3)],ax=ax2,)
ax2.set_xlabel('Loan Status')
ax2.set_ylabel('Total Received Late Fee')
ax2.set_title('Total Received Late Fee by Loan Status Barplot')

sns.lineplot(x='issue_d' ,y='total_rec_late_fee',hue='loan_status',
             data=df[(np.abs(stats.zscore(df['total_rec_late_fee']))<3)])
ax3.set_xlabel('Date Issued')
ax3.set_ylabel("Total Received Late Fee")
ax3.set_title('Total Received Late Fee by Loan Status over time')
ax3.axvline(x='2020',linestyle='--')
plt.tight_layout()
plt.savefig('late_fee.png')
```

executed in 17.2s, finished 03:03:17 2021-04-22









In [131]:
```
late = df[(np.abs(stats.zscore(df['total_rec_late_fee']))<3)]
```
executed in 847ms, finished 02:59:27 2021-04-22

In [132]:
```
late[late.loan_status == 'Fully Paid']['total_rec_late_fee'].describe()
```
executed in 941ms, finished 03:00:25 2021-04-22

Out[132]:
```
count    1.389853e+06
mean     4.325036e-01
std      3.335348e+00
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      4.450000e+01
Name: total_rec_late_fee, dtype: float64
```

In [138]:
```
1.389853e+06* 4.325036e-01
```
executed in 9ms, finished 03:09:01 2021-04-22

Out[138]:
```
601116.4259708
```

In [135]:
```
late[late.loan_status == 'Charged Off']['total_rec_late_fee'].describe()
```
executed in 283ms, finished 03:05:25 2021-04-22

Out[135]:
```
count    324778.000000
mean          2.804207
std           8.159093
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max          44.500000
Name: total_rec_late_fee, dtype: float64
```

In [137]:
```
324778/(1.389853e+06+324778)
```
executed in 10ms, finished 03:06:03 2021-04-22

Out[137]:
```
0.18941568185807908
```

In [139]:
```
 324778.000000* 2.804207
```
executed in 9ms, finished 03:09:28 2021-04-22

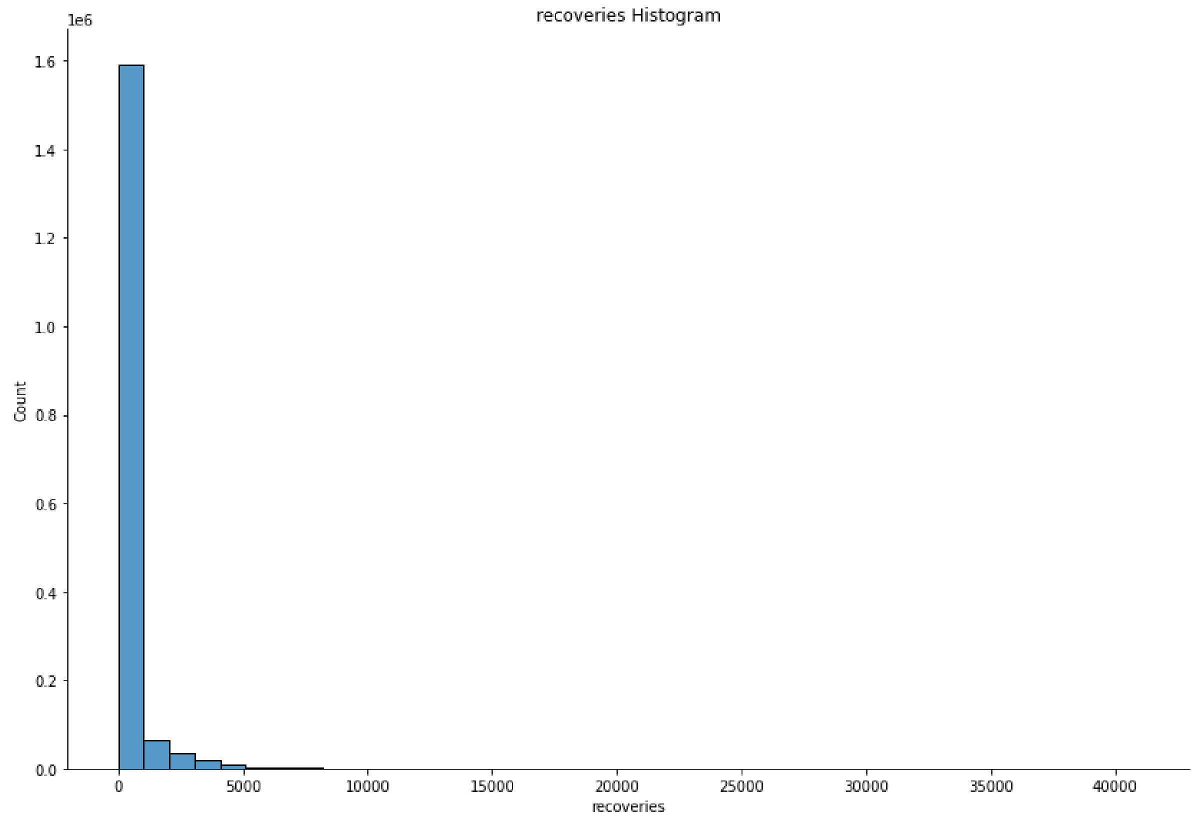
Out[139]:
```
910744.7410459999
```

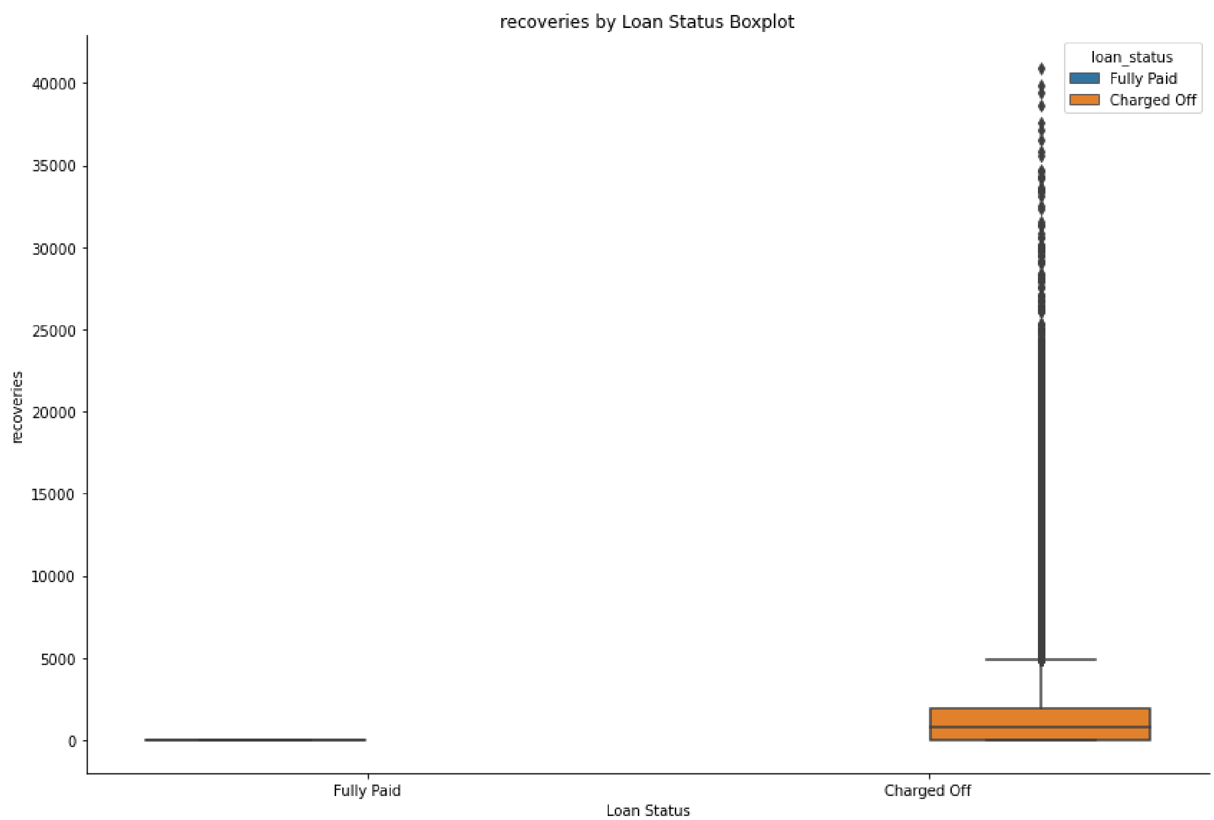
1.18 Recoveries

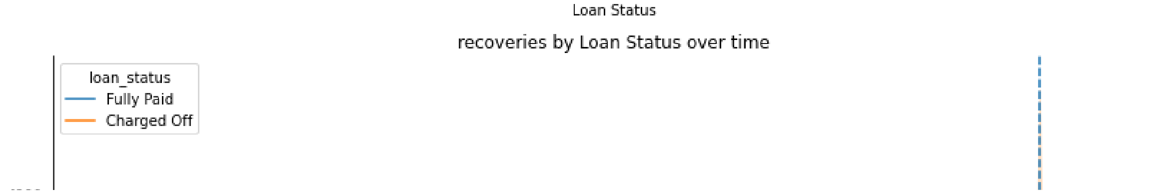
In [143]:
```
column_info('recoveries')
```
executed in 30ms, finished 06:05:49 2021-04-22

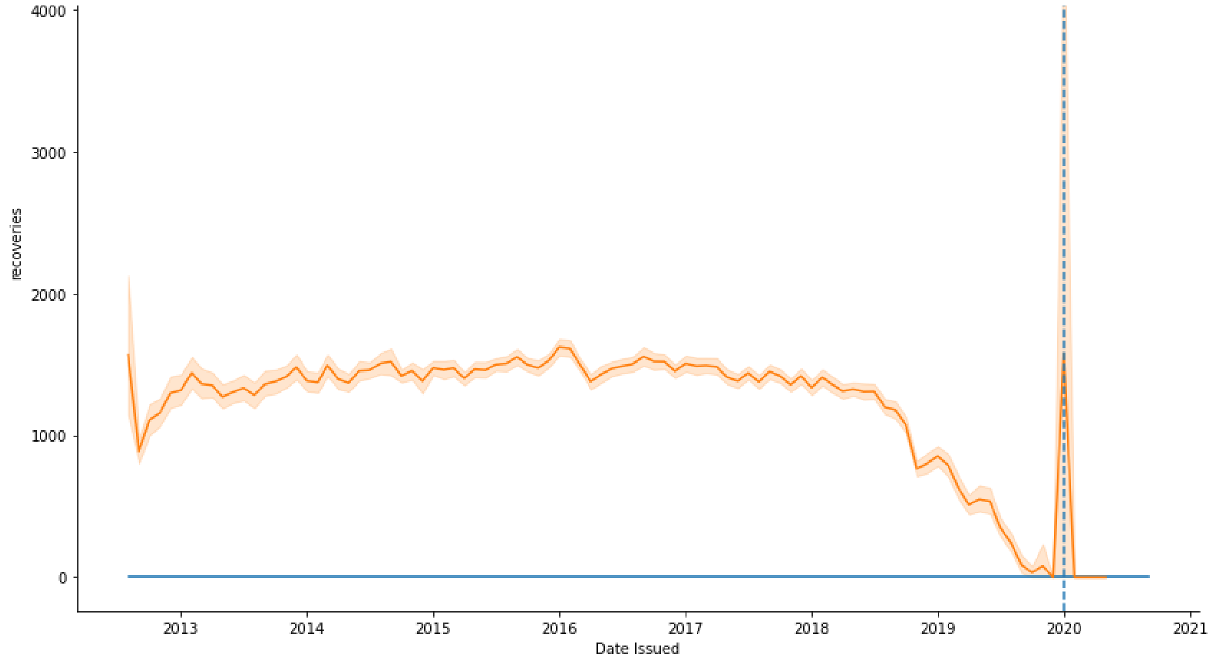
Out[143]:
```
'post charge off gross recovery'
```

In [142]:
```
continuous_plot('recoveries')
plt.savefig('recoveries.png')
```
executed in 15.7s, finished 03:14:15 2021-04-22









1.19 Inquiry in the last 6 months

In [145]:
```
df.inq_last_6mths.value_counts()
```
executed in 63ms, finished 06:06:32 2021-04-22

Out[145]:
```
0.0    1032003
1.0     464220
2.0     162402
3.0      56828
4.0      15306
5.0       5315
6.0        859
7.0          3
8.0          1
Name: inq_last_6mths, dtype: int64
```

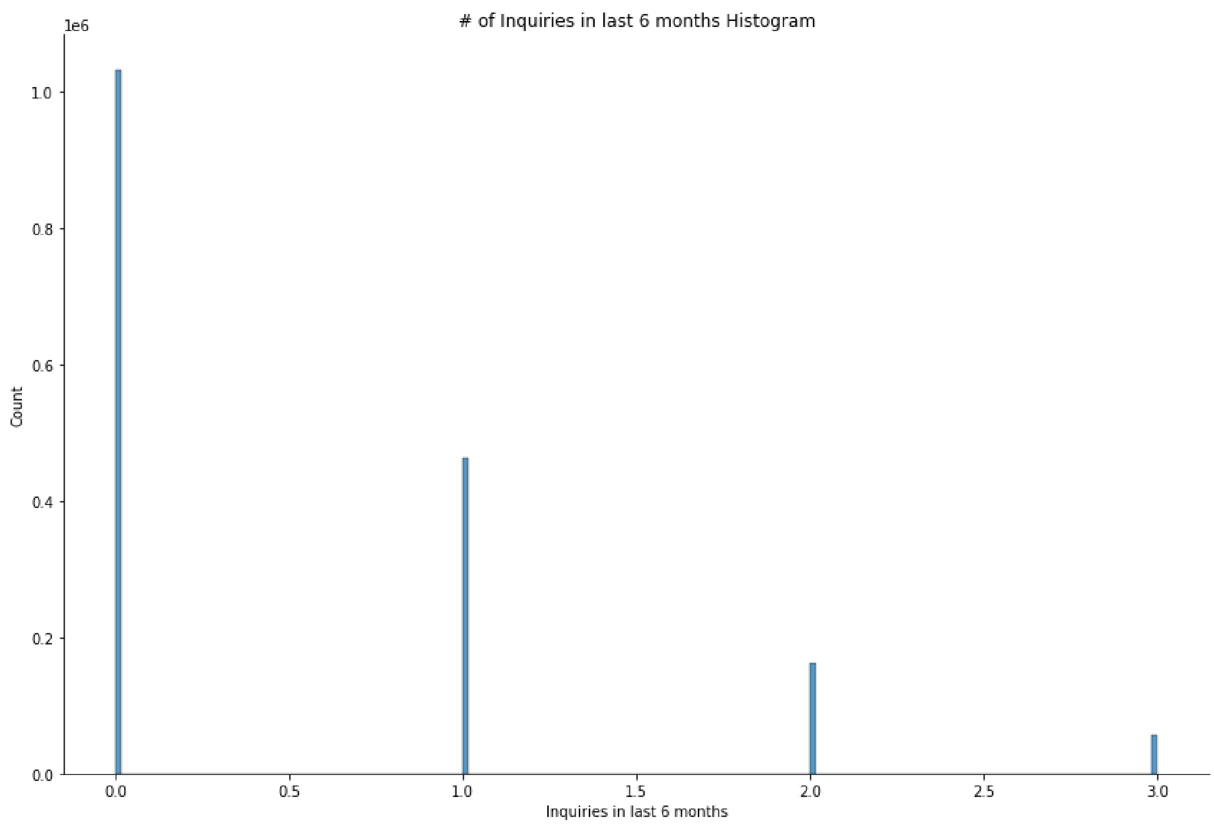
In [144]:
```
column_info('inq_last_6mths')
```
executed in 9ms, finished 06:06:21 2021-04-22

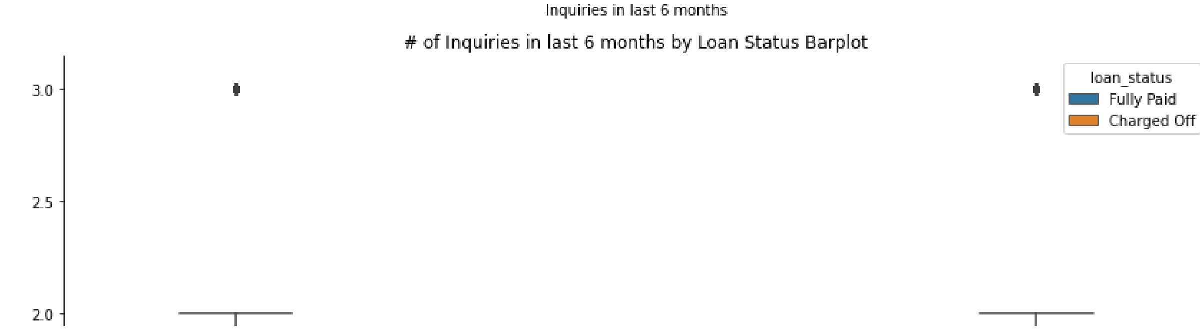
Out[144]:
```
'The number of inquiries in past 6 months (excluding auto and mortgage inquirie
s)'
```

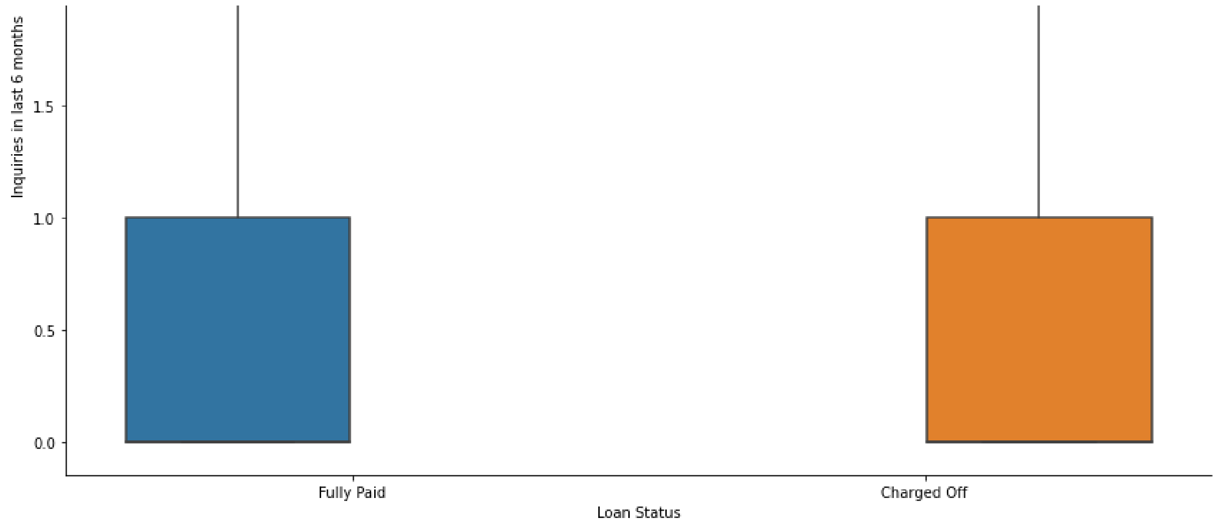
In [150]:

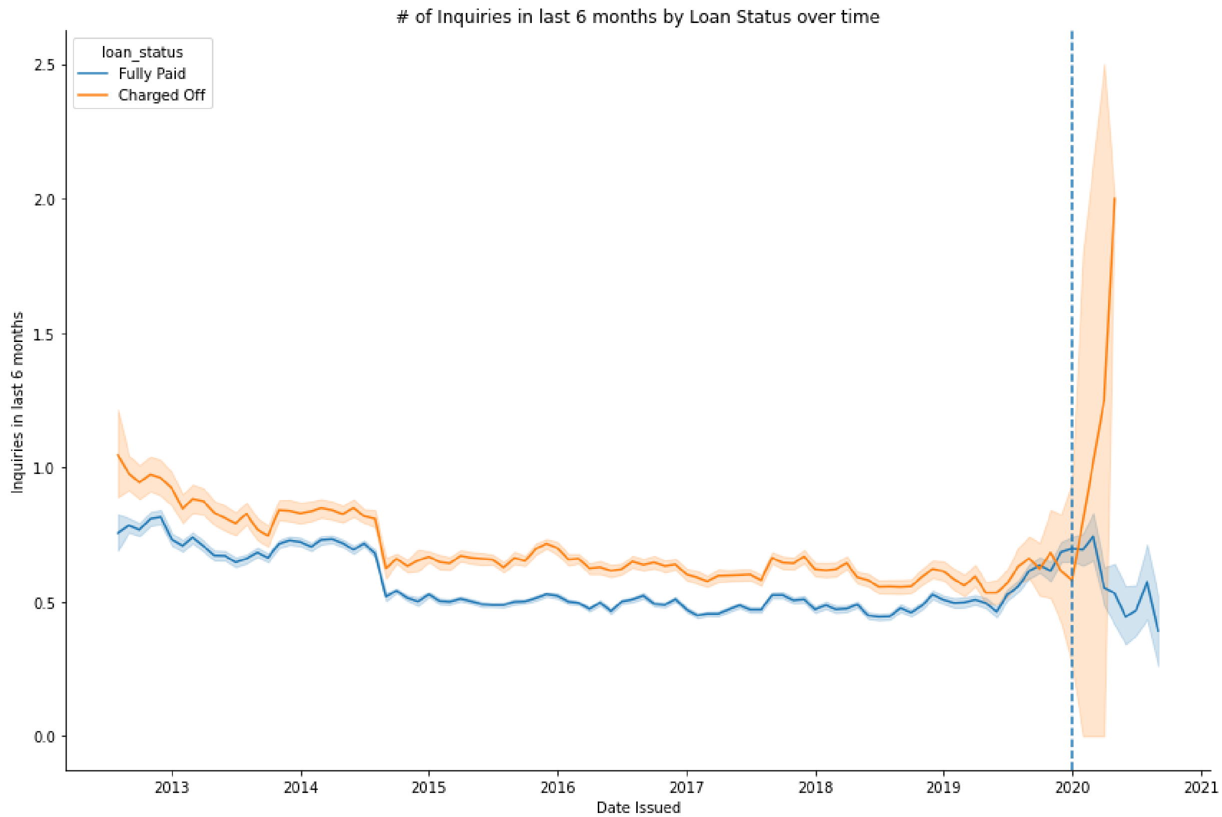
```
fig, (ax1,ax2,ax3) = plt.subplots(3,1,figsize=(12,24))
sns.histplot(x='inq_last_6mths',data=df[(np.abs(stats.zscore(df['inq_last_6mths']
ax1.set_xlabel("Inquiries in last 6 months")
ax1.set_ylabel('Count')
ax1.set_title('# of Inquiries in last 6 months Histogram')

sns.despine()
sns.boxplot(x='loan_status' ,y='inq_last_6mths',hue='loan_status',
            data=df[(np.abs(stats.zscore(df['inq_last_6mths']))<3)],ax=ax2,)
ax2.set_xlabel('Loan Status')
ax2.set_ylabel('Inquiries in last 6 months')
ax2.set_title('# of Inquiries in last 6 months by Loan Status Barplot')

sns.lineplot(x='issue_d' ,y='inq_last_6mths',hue='loan_status',
             data=df[(np.abs(stats.zscore(df['inq_last_6mths']))<3)])
ax3.set_xlabel('Date Issued')
ax3.set_ylabel("Inquiries in last 6 months")
ax3.set_title('# of Inquiries in last 6 months by Loan Status over time')
ax3.axvline(x='2020',linestyle='--')
plt.tight_layout()
```

executed in 35.5s, finished 06:13:05 2021-04-22









In [147]:
```
inq = df[(np.abs(stats.zscore(df['inq_last_6mths']))<3)]
```
executed in 6.65s, finished 06:08:17 2021-04-22

In [148]:
```
inq[inq.loan_status == 'Fully Paid']['inq_last_6mths'].describe()
```
executed in 957ms, finished 06:08:55 2021-04-22

Out[148]:
```
count    1.384187e+06
mean     5.345470e-01
std      7.806073e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      1.000000e+00
max      3.000000e+00
Name: inq_last_6mths, dtype: float64
```

In [149]:
```
inq[inq.loan_status == 'Charged Off']['inq_last_6mths'].describe()
```
executed in 487ms, finished 06:08:59 2021-04-22

Out[149]:
```
count    331266.000000
mean          0.662896
std           0.852488
min           0.000000
25%           0.000000
50%           0.000000
75%           1.000000
max           3.000000
Name: inq_last_6mths, dtype: float64
```

1.20 Number of accounts opened in last 12 months

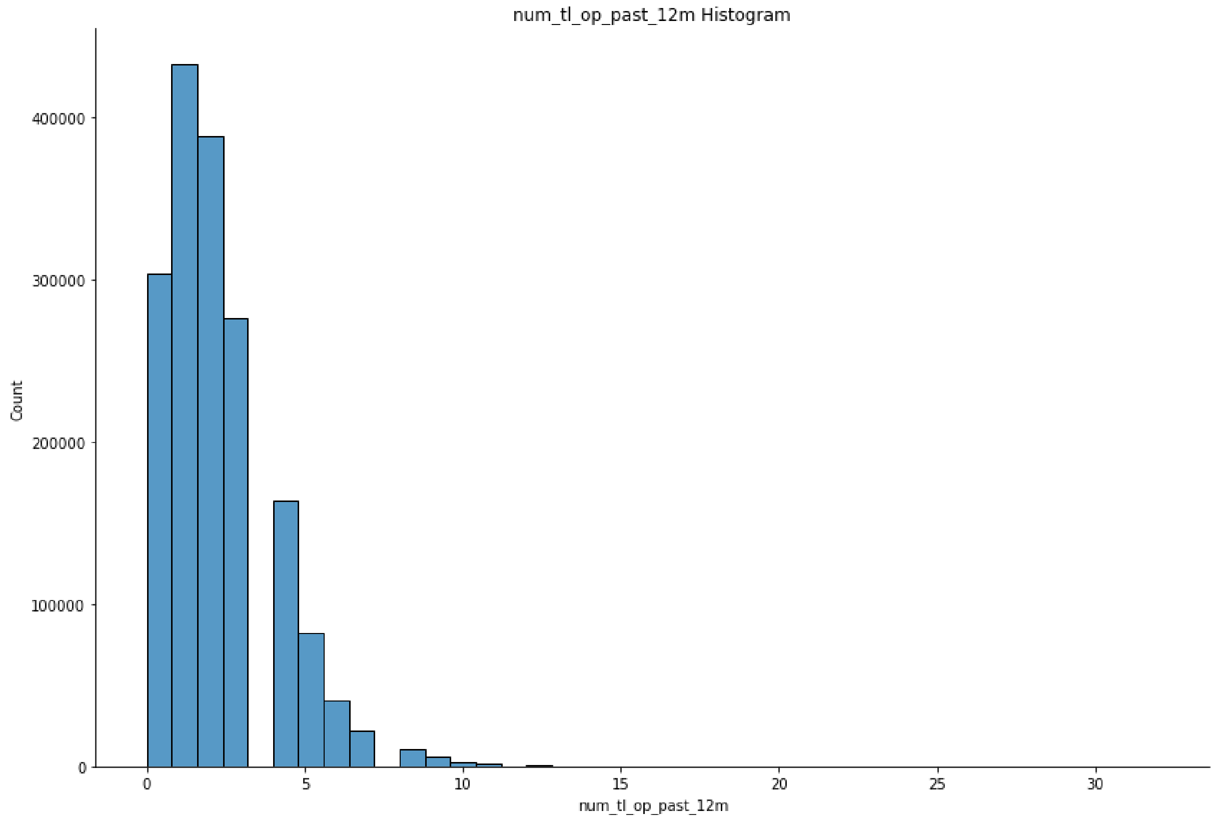
In [151]:
```
column_info('num_tl_op_past_12m')
```
executed in 17ms, finished 06:59:29 2021-04-22

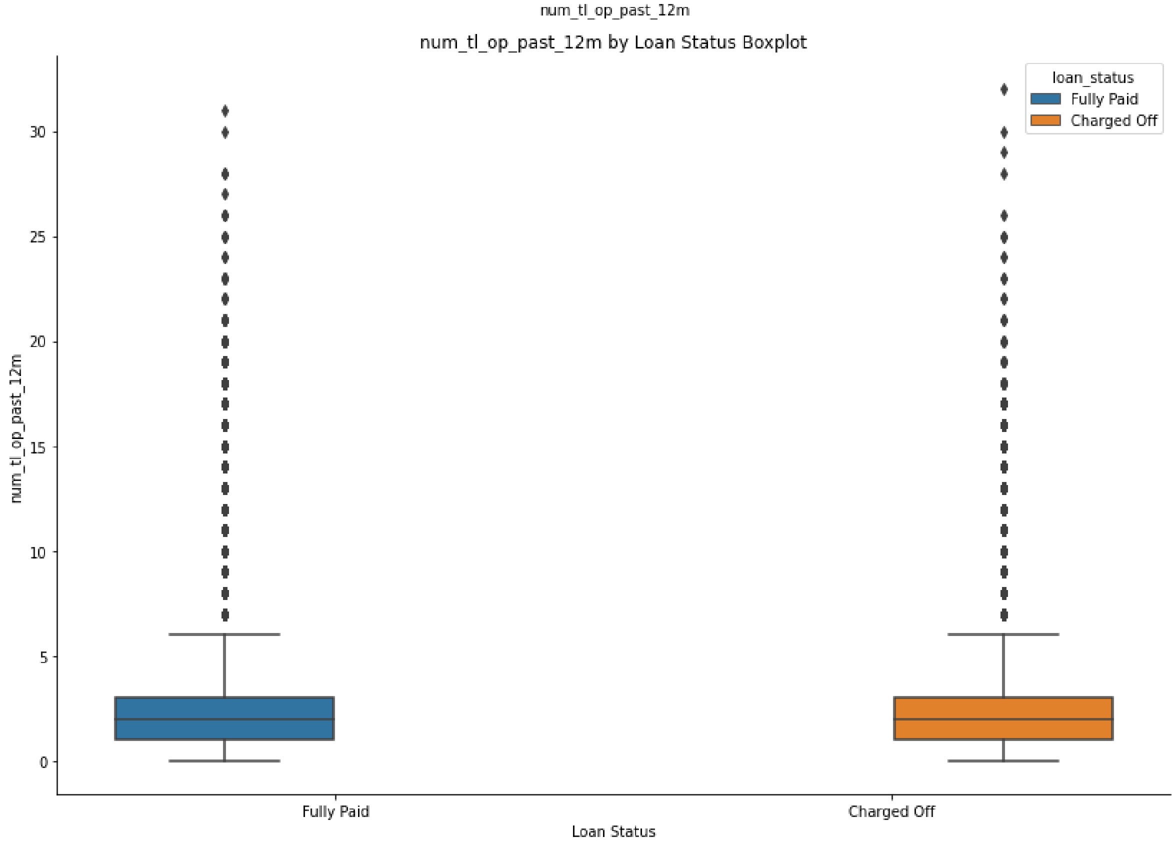
Out[151]:
```
'Number of accounts opened in past 12 months'
```

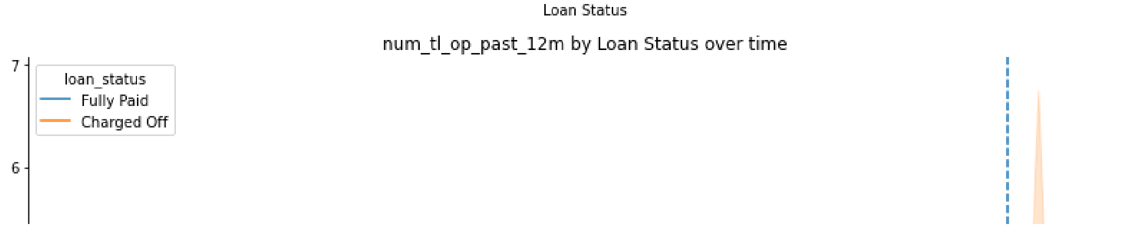
In [152]:
```
df.num_tl_op_past_12m.value_counts()
```
executed in 111ms, finished 07:50:27 2021-04-22

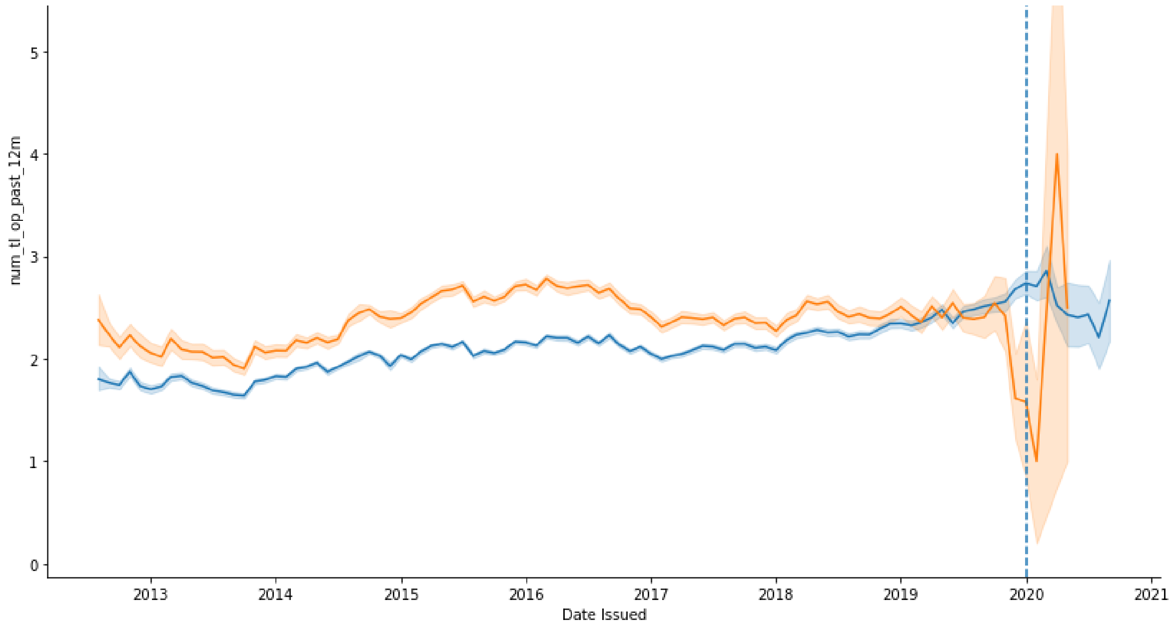
```
14.0      403
15.0      236
16.0      168
17.0      108
18.0       51
19.0       38
20.0       29
21.0       20
23.0       13
22.0        9
25.0        7
24.0        5
28.0        4
26.0        4
30.0        2
27.0        1
32.0        1
29.0        1
31.0        1
```

In [153]:
```
continuous_plot('num_tl_op_past_12m')
```
executed in 22.5s, finished 07:51:04 2021-04-22








5
4
3
2
1
0
num_tl_op_past_12m
2013
2014
2015
2016
2017
2018
2019
2020
2021
Date Issued

In [ ]: