```
Importing Data
```

In [1]:

```
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Pandas options
pd.set_option('display.max_colwidth', 1000, 'display.max_rows', None

# Plotting options
%matplotlib inline
mpl.style.use('ggplot')
sns.set(style='whitegrid')
```

In [12]:

```
loans = pd.read_csv(r"C:\Users\DELL\Downloads\loan data.csv")
```

```
C:\Users\DELL\anaconda3\lib\site-packages\IPython\core\interactiveshel
l.py:3444: DtypeWarning: Columns (47) have mixed types.Specify dtype op
tion on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Checkig basic information about data:

In [13]:

```
loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Columns: 111 entries, id to total_il_high_credit_limit
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB
```

In [14]:

```
loans.sample(5)
```

f	0.0	0.0	12070.20354	11946.41	9750.00	2320.20	0.

```
We're are trying to make EDA on the `loan_status` variable.
```

In [16]:
```
loans['loan_status'].value_counts(dropna=False)
```

Out[16]:
```
Fully Paid     32950
Charged Off     5627
Current         1140
Name: loan_status, dtype: int64
```

In [17]:
```
loans = loans.loc[loans['loan_status'].isin(['Fully Paid', 'Charged
```

In [18]:
```
loans.shape
```

Out[18]:
```
(38577, 111)
```

In [19]:
```
loans['loan_status'].value_counts(dropna=False)
```

Out[19]:
```
Fully Paid     32950
Charged Off     5627
Name: loan_status, dtype: int64
```

In [20]:
```
loans['loan_status'].value_counts(normalize=True, dropna=False)
```

Out[20]:
```
Fully Paid     0.854136
Charged Off    0.145864
Name: loan_status, dtype: float64
```

About 79% of the remaining loans have been fully paid and 21% have charged off, so we have a somewhat unbalanced classification problem.

calculate the percentage of missing data for each feature if they are in less numbers so we can delete the rows

In [21]:
```
missing_fractions = loans.isnull().mean().sort_values(ascending=Fals
```

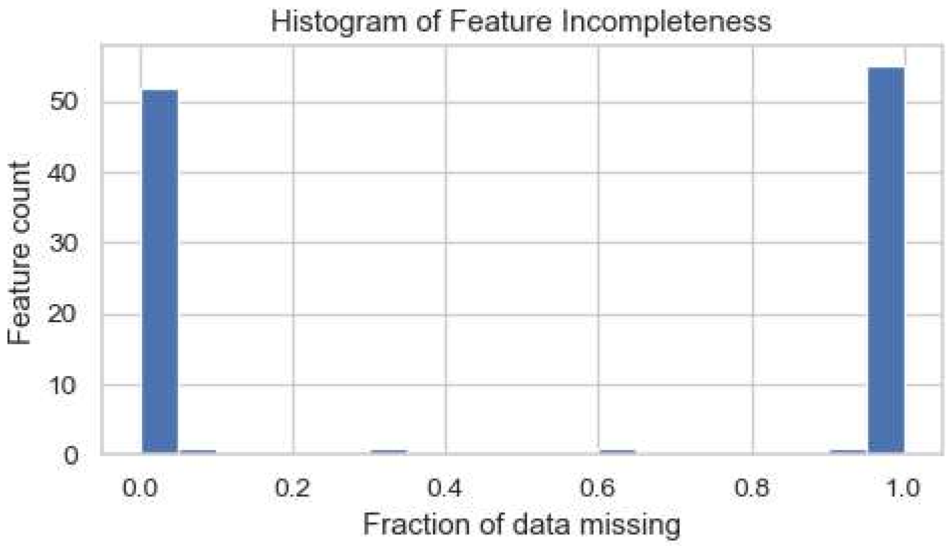
In [22]:
```
missing_fractions.head(10)
```

Out[22]:
```
verification_status_joint    1.0
annual_inc_joint             1.0
mo_sin_old_rev_tl_op         1.0
mo_sin_old_il_acct           1.0
bc_util                      1.0
bc_open_to_buy               1.0
avg_cur_bal                  1.0
acc_open_past_24mths         1.0
inq_last_12m                 1.0
total_cu_tl                  1.0
dtype: float64
```

Let's visualize the distribution of missing data percentages:

In [23]:
```
plt.figure(figsize=(6,3), dpi=90)
missing_fractions.plot.hist(bins=20)
plt.title('Histogram of Feature Incompleteness')
plt.xlabel('Fraction of data missing')
plt.ylabel('Feature count')
```

Out[23]:
```
Text(0, 0.5, 'Feature count')
```



In [24]:
```
drop_list = sorted(list(missing_fractions[missing_fractions > 0.3].i
print(drop_list)
```

```
['acc_open_past_24mths', 'all_util', 'annual_inc_joint', 'avg_cur_bal',
'bc_open_to_buy', 'bc_util', 'desc', 'dti_joint', 'il_util', 'inq_fi',
'inq_last_12m', 'max_bal_bc', 'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_
op', 'mo_sin_rcnt_rev_tl_op', 'mo_sin_rcnt_tl', 'mort_acc', 'mths_since
_last_delinq', 'mths_since_last_major_derog', 'mths_since_last_record',
'mths_since_rcnt_il', 'mths_since_recent_bc', 'mths_since_recent_bc_dl
q', 'mths_since_recent_inq', 'mths_since_recent_revol_delinq', 'next_py
mnt_d', 'num_accts_ever_120_pd', 'num_actv_bc_tl', 'num_actv_rev_tl',
'num_bc_sats', 'num_bc_tl', 'num_il_tl', 'num_op_rev_tl', 'num_rev_acct
s', 'num_rev_tl_bal_gt_0', 'num_sats', 'num_tl_120dpd_2m', 'num_tl_30dp
d', 'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'open_acc_6m', 'open_il
_12m', 'open_il_24m', 'open_il_6m', 'open_rv_12m', 'open_rv_24m', 'pct_
tl_nvr_dlq', 'percent_bc_gt_75', 'tot_coll_amt', 'tot_cur_bal', 'tot_hi
_cred_lim', 'total_bal_ex_mort', 'total_bal_il', 'total_bc_limit', 'tot
al_cu_tl', 'total_il_high_credit_limit', 'total_rev_hi_lim', 'verificat
ion_status_joint']
```

How many features will be dropped?

In [25]:
```
len(drop_list)
```

Out[25]:
```
58
```

Drop these features:

In [26]:
```
loans.drop(labels=drop_list, axis=1, inplace=True)
```

In [27]:
```
loans.shape
```

Out[27]: (38577, 53)

Only keep loan features known to potential investors

In [28]:
```
print(sorted(loans.columns))
```

```
['acc_now_delinq', 'addr_state', 'annual_inc', 'application_type', 'cha
rgeoff_within_12_mths', 'collection_recovery_fee', 'collections_12_mths
_ex_med', 'delinq_2yrs', 'delinq_amnt', 'dti', 'earliest_cr_line', 'emp
_length', 'emp_title', 'funded_amnt', 'funded_amnt_inv', 'grade', 'home
_ownership', 'id', 'initial_list_status', 'inq_last_6mths', 'installmen
t', 'int_rate', 'issue_d', 'last_credit_pull_d', 'last_pymnt_amnt', 'la
st_pymnt_d', 'loan_amnt', 'loan_status', 'member_id', 'open_acc', 'out_
prncp', 'out_prncp_inv', 'policy_code', 'pub_rec', 'pub_rec_bankruptcie
s', 'purpose', 'pymnt_plan', 'recoveries', 'revol_bal', 'revol_util',
'sub_grade', 'tax_liens', 'term', 'title', 'total_acc', 'total_pymnt',
'total_pymnt_inv', 'total_rec_int', 'total_rec_late_fee', 'total_rec_pr
ncp', 'url', 'verification_status', 'zip_code']
```

For each of these features, we check the description in the Data Dictionary and only keep the features that would have been available to investors considering an investment in the loan. These include features in the loan application, and any features added by LendingClub when the loan listing was accepted, such as the loan grade and interest rate.

In [29]:
```
keep_list = ['addr_state', 'annual_inc', 'application_type', 'dti',
```

In [30]:
```
len(keep_list)
```

Out[30]: 31

The list of features to drop is any feature not in `keep_list` :

In [31]:
```
drop_list = [col for col in loans.columns if col not in keep_list]
print(drop_list)
```

```
['member_id', 'funded_amnt', 'funded_amnt_inv', 'pymnt_plan', 'url', 'd
elinq_2yrs', 'inq_last_6mths', 'out_prncp', 'out_prncp_inv', 'total_pym
nt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_
late_fee', 'recoveries', 'collection_recovery_fee', 'last_pymnt_d', 'la
st_pymnt_amnt', 'last_credit_pull_d', 'collections_12_mths_ex_med', 'po
licy_code', 'acc_now_delinq', 'chargeoff_within_12_mths', 'delinq_amn
t', 'tax_liens']
```

In [32]:
```
len(drop_list)
```

Out[32]: 25

Drop these features:

In [33]:
```
loans.drop(labels=drop_list, axis=1, inplace=True)
```

In [34]:
```
loans.shape
```

Out[34]: (38577, 28)

Exploratory Analysis

Steps:

1. Drop the feature if it is not useful for predicting the loan status.
2. View summary statistics and visualize the data, plotting against the loan status.
3. Modify the feature to make it useful for modeling, if necessary.

We define a function for plotting a variable and comparing with the loan status:

In [69]:
```
def plot_var(col_name, full_name, continuous):
    """
    Visualize a variable with and without faceting on the loan statu
    - col_name is the variable name in the dataframe
    - full_name is the full variable name
    - continuous is True if the variable is continuous, False otherw
    """
    f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,3), d

    # Plot without loan status
    if continuous:
        sns.distplot(loans.loc[loans[col_name].notnull(), col_name],
    else:
        sns.countplot(loans[col_name], order=sorted(loans[col_name].
    ax1.set_xlabel(full_name)
    ax1.set_ylabel('Count')
    ax1.set_title(full_name)

    # Plot with loan status
    if continuous:
        sns.boxplot(x=col_name, y='loan_status', data=loans, ax=ax2)
        ax2.set_ylabel('')
        ax2.set_title(full_name + ' by Loan Status')
    else:
        charge_off_rates = loans.groupby(col_name)['loan_status'].va
        sns.barplot(x=charge_off_rates.index, y=charge_off_rates.val
        ax2.set_ylabel('Fraction of Loans Charged-off')
        ax2.set_title('Charge-off Rate by ' + full_name)
    ax2.set_xlabel(full_name)

    plt.tight_layout()
```

Print the remaining features for future reference:

In [36]:
```
print(list(loans.columns))
```

```
['id', 'loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_gr
ade', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verif
ication_status', 'issue_d', 'loan_status', 'purpose', 'title', 'zip_cod
e', 'addr_state', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 're
vol_bal', 'revol_util', 'total_acc', 'initial_list_status', 'applicatio
n_type', 'pub_rec_bankruptcies']
```

Data Dictionary: "A unique [LendingClub] assigned ID for the loan listing."

In [37]:
```
loans['id'].sample(5)
```

Out[37]:
```
29805     517982
3229     1022153
21086     647976
39339     224614
19208     679644
Name: id, dtype: int64
```

Are all the IDs unique?

In [38]:
```
loans['id'].describe()
```

Out[38]:
```
count    3.857700e+04
mean     6.763787e+05
std      2.092639e+05
min      5.473400e+04
25%      5.120330e+05
50%      6.564230e+05
75%      8.291460e+05
max      1.077501e+06
Name: id, dtype: float64
```

Yes, they are all unique. The ID is not useful for modeling, either as a categorical variable (there are too many distinct values) or as a numerical variable (the IDs vary wildly in magnitude, likely without any significance), so we drop this variable.

In [39]:
```
loans.drop('id', axis=1, inplace=True)
```

Data Dictionary: "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."

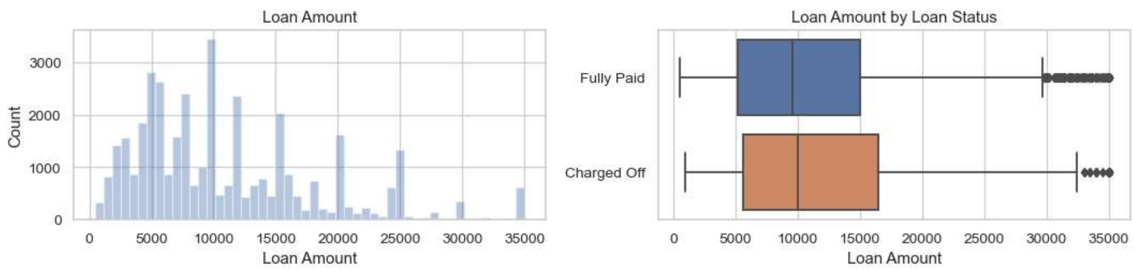
In [40]:
```
1 loans['loan_amnt'].describe()
```

Out[40]:
```
count    38577.000000
mean     11047.025430
std       7348.441646
min        500.000000
25%       5300.000000
50%       9600.000000
75%      15000.000000
max      35000.000000
Name: loan_amnt, dtype: float64
```

Loan amounts range from $500to$40,000, with a median of $12,000.

In [41]:
```
1 plot_var('loan_amnt', 'Loan Amount', continuous=True)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be remov
ed in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an ax
es-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Charged-off loans tend to have higher loan amounts. Let's compare the summary statistics by loan status:

In [42]:
```
1 loans.groupby('loan_status')['loan_amnt'].describe()
```

Out[42]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5627.0	12104.385108	8085.732038	900.0	5600.0	10000.0	16500.0	35000.0
Fully Paid	32950.0	10866.455994	7199.629493	500.0	5200.0	9600.0	15000.0	35000.0

Data Dictionary: "The number of payments on the loan. Values are in months and can be either 36 or 60."

In [43]:
```
1 loans['term'].value_counts(dropna=False)
```

Out[43]:
```
 36 months    29096
 60 months     9481
Name: term, dtype: int64
```

Convert `term` to integers.

In [44]:
```
loans['term'] = loans['term'].apply(lambda s: np.int8(s.split()[0]))
```

In [45]:
```
loans['term'].value_counts(normalize=True)
```

Out[45]:
```
36    0.754232
60    0.245768
Name: term, dtype: float64
```

Compare the charge-off rate by loan period:

In [46]:
```
loans.groupby('term')['loan_status'].value_counts(normalize=True).lc
```

Out[46]:
```
term
36    0.110909
60    0.253138
Name: loan_status, dtype: float64
```

About 76% of the completed loans have three-year periods, and the rest have five-year periods. Loans with five-year periods are more than twice as likely to charge-off as loans with three-year periods.

Data Dictionary: "Interest Rate on the loan."

In [47]:
```
loans['int_rate'].describe()
```

Out[47]:
```
count      38577
unique       370
top       10.99%
freq         913
Name: int_rate, dtype: object
```

Interest rates range from 5.32% to 30.99% (!) with a median of 13.1%.

Charged-off loans tend to have much higher interest rates. Let's compare the summary statistics by loan status:

In [49]:
```
loans.groupby('loan_status')['int_rate'].describe()
```

Out[49]:

	count	unique	top	freq
loan_status				
Charged Off	5627	332	13.49%	127
Fully Paid	32950	360	10.99%	818

Data Dictionary: "The monthly payment owed by the borrower if the loan originates."

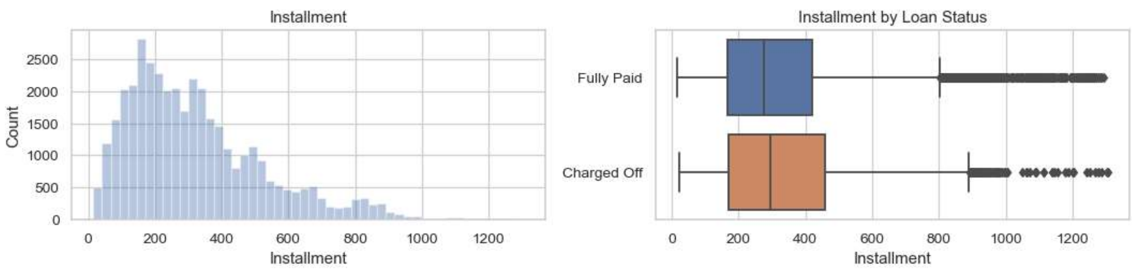
In [50]:
```
1 loans['installment'].describe()
```

Out[50]:
```
count    38577.000000
mean       322.466318
std        208.639215
min         15.690000
25%        165.740000
50%        277.860000
75%        425.550000
max       1305.190000
Name: installment, dtype: float64
```

Installments range from $4.93to$1,714, with a median of $377.

In [51]:
```
1 plot_var('installment', 'Installment', continuous=True)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be remov
ed in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an ax
es-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Charged-off loans tend to have higher installments. Let's compare the summary statistics by loan status:

In [52]:
```
1 loans.groupby('loan_status')['installment'].describe()
```

Out[52]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5627.0	336.175006	217.051841	22.79	168.5550	293.87	457.840	1305.19
Fully Paid	32950.0	320.125232	207.081110	15.69	165.2825	275.65	420.735	1295.21

Loans that charge off have $30 higher installments on average.

Data Dictionary: "The job title supplied by the Borrower when applying for the loan."

In [53]:
```
loans['emp_title'].describe()
```

Out[53]:
```
count       36191
unique      28027
top       US Army
freq          131
Name: emp_title, dtype: object
```

There are too many different job titles for this feature to be useful, so we drop it.

In [54]:
```
loans.drop(labels='emp_title', axis=1, inplace=True)
```

Data Dictionary: "The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER."

In [55]:
```
loans['home_ownership'].value_counts(dropna=False)
```

Out[55]:
```
RENT        18480
MORTGAGE    17021
OWN          2975
OTHER          98
NONE            3
Name: home_ownership, dtype: int64
```

Replace the values `ANY` and `NONE` with `OTHER` :

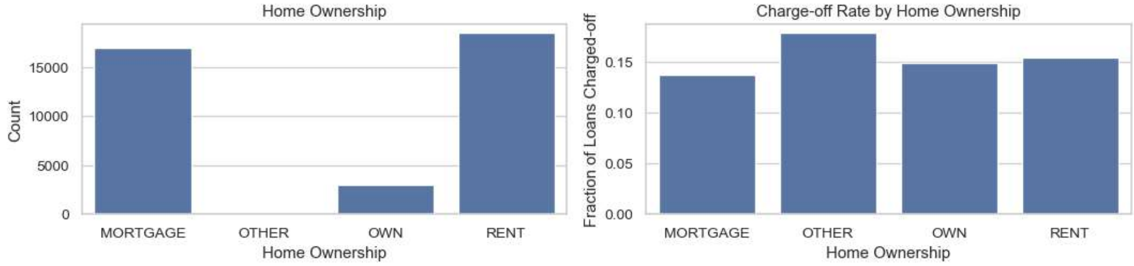
In [56]:
```
loans['home_ownership'].replace(['NONE', 'ANY'], 'OTHER', inplace=Tr
```

In [57]:
```
loans['home_ownership'].value_counts(dropna=False)
```

Out[57]:
```
RENT        18480
MORTGAGE    17021
OWN          2975
OTHER         101
Name: home_ownership, dtype: int64
```

In [58]:
```
plot_var('home_ownership', 'Home Ownership', continuous=False)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fu
tureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error o
r misinterpretation.
  warnings.warn(
```



There appear to be large differences in charge-off rates by home ownership status. Renters and homeowners have a higher probability of charge-off. Let's compare the charge-off rates:

In [59]:
```
loans.groupby('home_ownership')['loan_status'].value_counts(normaliz
```

Out[59]:
```
home_ownership
MORTGAGE    0.136713
OTHER       0.178218
OWN         0.148908
RENT        0.153626
Name: loan_status, dtype: float64
```

Data Dictionary: "The self-reported annual income provided by the borrower during registration."

In [60]:
```
loans['annual_inc'].describe()
```

Out[60]:
```
count    3.857700e+04
mean     6.877797e+04
std      6.421868e+04
min      4.000000e+03
25%      4.000000e+04
50%      5.886800e+04
75%      8.200000e+04
max      6.000000e+06
Name: annual_inc, dtype: float64
```

Annual income ranges from $0to$9,550,000, with a median of $65,000. Because of the large range of incomes, we should take a log transform of the annual income variable.

In [61]:
```
loans['log_annual_inc'] = loans['annual_inc'].apply(lambda x: np.log
```

In [62]:
```
loans.drop('annual_inc', axis=1, inplace=True)
```

In [63]:
```
loans['log_annual_inc'].describe()
```

Out[63]:
```
count    38577.000000
mean         4.763961
std          0.243124
min          3.602169
25%          4.602071
50%          4.769887
75%          4.913819
max          6.778151
Name: log_annual_inc, dtype: float64
```

In [64]:
```
plot_var('log_annual_inc', 'Log Annual Income', continuous=True)
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:261
9: FutureWarning: `distplot` is a deprecated function and will be remov
ed in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an ax
es-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

It appears that individuals with higher income are more likely to pay off their loans. Let's compare the summary statistics by loan status:

In [65]:
```
loans.groupby('loan_status')['log_annual_inc'].describe()
```

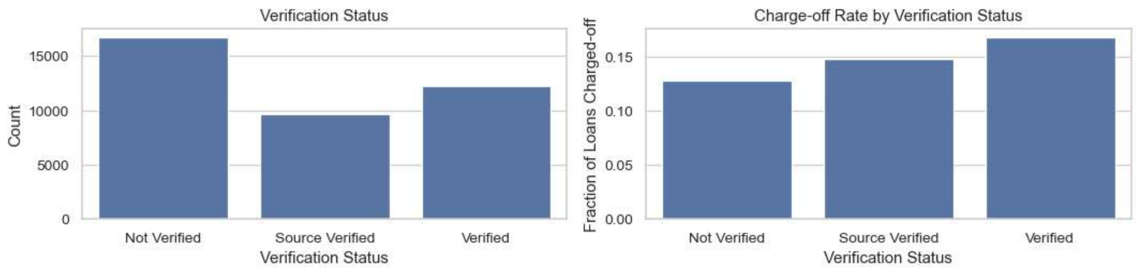
Out[65]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5627.0	4.723318	0.243179	3.610767	4.568213	4.724284	4.875067	6.09691C
Fully Paid	32950.0	4.770901	0.242438	3.602169	4.614198	4.778158	4.924284	6.778151

Data Dictionary: "Indicates if income was verified by [Lending Club], not verified, or if the income source was verified."

In [66]:
```
plot_var('verification_status', 'Verification Status', continuous=Fa
```

```
C:\Users\DELL\anaconda3\lib\site-packages\seaborn\_decorators.py:36: Fu
tureWarning: Pass the following variable as a keyword arg: x. From vers
ion 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error o
r misinterpretation.
  warnings.warn(
```



Data Dictionary: "A category provided by the borrower for the loan request."

In [67]:
```
loans['purpose'].value_counts()
```

Out[67]:
```
debt_consolidation    18055
credit_card            5027
other                  3865
home_improvement       2875
major_purchase         2150
small_business         1754
car                    1499
wedding                 926
medical                 681
moving                  576
vacation                375
house                   367
educational             325
renewable_energy        102
Name: purpose, dtype: int64
```

Calculate the charge-off rates by purpose:

In [68]:
```
loans.groupby('purpose')['loan_status'].value_counts(normalize=True)
```

Out[68]:
```
purpose
major_purchase        0.103256
wedding               0.103672
car                   0.106738
credit_card           0.107818
home_improvement      0.120696
vacation              0.141333
debt_consolidation    0.153254
medical               0.155653
moving                0.159722
house                 0.160763
other                 0.163777
educational           0.172308
renewable_energy      0.186275
small_business        0.270810
Name: loan_status, dtype: float64
```

Notice that only 12% of completed loans for weddings have charged-off, but 30% of completed small business loans have charged-off.