In [1]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sp
sns.set(style='whitegrid')
```

In [2]:
```
loans = pd.read_csv(r'C:\Users\Admin\Downloads\lending_club_loans.csv',skiprows=1)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3444: DtypeWarning: Columns (0,49) have mix
ed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [3]:
```
1 loans
```

Out[3]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	...	num_tl_90g_dpd_24m
0	1077501	1296599.0	5000.0	5000.0	4975.0	36 months	10.65%	162.87	B	B2	...	NaN
1	1077430	1314167.0	2500.0	2500.0	2500.0	60 months	15.27%	59.83	C	C4	...	NaN
2	1077175	1313524.0	2400.0	2400.0	2400.0	36 months	15.96%	84.33	C	C5	...	NaN
3	1076863	1277178.0	10000.0	10000.0	10000.0	36 months	13.49%	339.31	C	C1	...	NaN
4	1075358	1311748.0	3000.0	3000.0	3000.0	60 months	12.69%	67.79	B	B5	...	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
42533	72176	70868.0	2525.0	2525.0	225.0	36 months	9.33%	80.69	B	B3	...	NaN
42534	71623	70735.0	6500.0	6500.0	0.0	36 months	8.38%	204.84	A	A5	...	NaN
42535	70686	70681.0	5000.0	5000.0	0.0	36 months	7.75%	156.11	A	A3	...	NaN
42536	Total amount funded in policy code 1: 460296150	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
42537	Total amount funded in policy code 2: 0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN

42538 rows × 115 columns

In [4]:
```
1 loans.shape
```

Out[4]:
```
(42538, 115)
```

In [5]:
```
1 loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42538 entries, 0 to 42537
Columns: 115 entries, id to total_il_high_credit_limit
dtypes: float64(90), object(25)
memory usage: 37.3+ MB
```

In [6]:
```
1 print(loans.dtypes)
```

```
id                           object
member_id                   float64
loan_amnt                   float64
funded_amnt                 float64
funded_amnt_inv             float64
                              ...
tax_liens                   float64
tot_hi_cred_lim             float64
total_bal_ex_mort           float64
total_bc_limit              float64
total_il_high_credit_limit  float64
Length: 115, dtype: object
```

In [7]:
```
for var in loans.dtypes:
    print(var)
```

```
object
float64
float64
float64
float64
object
object
float64
object
object
object
object
object
float64
object
object
object
object
object
```

In [8]:
```
1 loans.sample(5)
```

Out[8]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	...	num_tl_90g_dpd_24m
21111	648600	829772.0	5000.0	5000.0	5000.0	60 months	9.25%	104.40	B	B2	...	NaN
3314	1020863	1249650.0	12000.0	12000.0	12000.0	36 months	17.58%	431.31	D	D4	...	NaN
18135	705659	897741.0	10000.0	10000.0	10000.0	36 months	11.11%	327.91	B	B5	...	NaN
3882	1007709	1234351.0	12600.0	12600.0	12350.0	36 months	11.71%	416.76	B	B3	...	NaN
13941	774142	976246.0	10000.0	10000.0	10000.0	36 months	11.99%	332.10	B	B5	...	NaN

5 rows × 115 columns

In [9]:
```
1 loans['loan_status']
```

Out[9]:
```
0                                              Fully Paid
1                                             Charged Off
2                                              Fully Paid
3                                              Fully Paid
4                                                 Current
                                ...
42533    Does not meet the credit policy. Status:Fully ...
42534    Does not meet the credit policy. Status:Fully ...
42535    Does not meet the credit policy. Status:Fully ...
42536                                                 NaN
42537                                                 NaN
Name: loan_status, Length: 42538, dtype: object
```

In [10]:
```
1 loans['loan_status'].value_counts(dropna=False)
```

Out[10]:
```
Fully Paid                                             33586
Charged Off                                             5653
Does not meet the credit policy. Status:Fully Paid      1988
Does not meet the credit policy. Status:Charged Off      761
Current                                                  513
In Grace Period                                           16
Late (31-120 days)                                        12
Late (16-30 days)                                          5
NaN                                                        3
Default                                                    1
Name: loan_status, dtype: int64
```

In [11]:
```
1 loans = loans.loc[loans['loan_status'].isin(['Fully Paid', 'Charged Off'])]
```

In [12]:
```
1 loans.shape
```

Out[12]:
```
(39239, 115)
```

In [13]:
```
1 loans['loan_status'].value_counts(dropna=False)
```

Out[13]:
```
Fully Paid     33586
Charged Off     5653
Name: loan_status, dtype: int64
```

In [14]:
```
1 loans['loan_status'].value_counts(normalize=True, dropna=False)
```

Out[14]:
```
Fully Paid     0.855934
Charged Off    0.144066
Name: loan_status, dtype: float64
```

About 85% of the remaining loans have been fully paid and 14% have charged off, so we have a somewhat unbalanced classification problem.

In [ ]:
```
1
```

In [15]:
```
1 missing_fractions = loans.isnull().mean().sort_values(ascending=False)
```

In [16]:
```
1 missing_fractions.head(10)
```

Out[16]:
```
annual_inc_joint          1.0
mo_sin_rcnt_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
inq_fi                    1.0
dtype: float64
```

In [17]:
```
1 plt.figure(figsize=(6,3), dpi=90)
2 missing_fractions.plot.hist(bins=20)
3 plt.title('Histogram of Feature Incompleteness')
4 plt.xlabel('Fraction of data missing')
5 plt.ylabel('Feature count')
```

Out[17]:
```
Text(0, 0.5, 'Feature count')
```

In [18]:
```
drop_list = sorted(list(missing_fractions[missing_fractions > 0.3].index))
print(drop_list)
```

```
['acc_open_past_24mths', 'all_util', 'annual_inc_joint', 'avg_cur_bal', 'bc_open_to_buy', 'bc_util', 'desc', 'dti_join
t', '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_t
l_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_dlq', 'mths_since_recent_inq', 'mths_since_recent_r
evol_delinq', 'next_pymnt_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_accts', 'num_rev_tl_bal_gt_0', 'num_sats', 'num_tl_120dpd_2m', 'num_tl_30dpd',
'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_mo
rt', 'total_bal_il', 'total_bc_limit', 'total_cu_tl', 'total_il_high_credit_limit', 'total_rev_hi_lim', 'verification_s
tatus_joint']
```

In [19]:
```
len(drop_list)
```

Out[19]:
```
58
```

In [ ]:
```

```

In [20]:
```
loans.drop(labels=drop_list, axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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-co
py)
  return super().drop(
```

In [21]:
```
loans.shape
```

Out[21]:
```
(39239, 57)
```

In [22]:
```
print(sorted(loans.columns))
```

```
['acc_now_delinq', 'addr_state', 'annual_inc', 'application_type', 'chargeoff_within_12_mths', 'collection_recovery_fe
e', 'collections_12_mths_ex_med', 'delinq_2yrs', 'delinq_amnt', 'dti', 'earliest_cr_line', 'emp_length', 'emp_title',
'fico_range_high', 'fico_range_low', 'funded_amnt', 'funded_amnt_inv', 'grade', 'home_ownership', 'id', 'initial_list_s
tatus', 'inq_last_6mths', 'installment', 'int_rate', 'issue_d', 'last_credit_pull_d', 'last_fico_range_high', 'last_fic
o_range_low', 'last_pymnt_amnt', 'last_pymnt_d', 'loan_amnt', 'loan_status', 'member_id', 'open_acc', 'out_prncp', 'out
_prncp_inv', 'policy_code', 'pub_rec', 'pub_rec_bankruptcies', 'purpose', 'pymnt_plan', 'recoveries', 'revol_bal', 'rev
ol_util', 'sub_grade', 'tax_liens', 'term', 'title', 'total_acc', 'total_pymnt', 'total_pymnt_inv', 'total_rec_int', 't
otal_rec_late_fee', 'total_rec_prncp', 'url', 'verification_status', 'zip_code']
```

In [23]:
```
keep_list = ['addr_state', 'annual_inc', 'application_type', 'dti', 'earliest_cr_line', 'emp_length', 'emp_title', '
```

In [24]:
```
len(keep_list)
```

Out[24]: 31

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

In [25]:
```
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', 'delinq_2yrs', 'inq_last_6mths', 'out_prncp', 'out
_prncp_inv', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee', 'recoveries',
'collection_recovery_fee', 'last_pymnt_d', 'last_pymnt_amnt', 'last_credit_pull_d', 'last_fico_range_high', 'last_fico_
range_low', 'collections_12_mths_ex_med', 'policy_code', 'acc_now_delinq', 'chargeoff_within_12_mths', 'delinq_amnt',
'tax_liens']
```

In [26]:
```
len(drop_list)
```

Out[26]: 27

In [ ]:
```

```

In [27]:
```
loans.drop(labels=drop_list, axis=1, inplace=True)
```

In [28]:
```
1 loans.shape
```

Out[28]: (39239, 30)

Pre-Proccessing

We'll inspect each feature individually, and do the following:

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:

Print the remaining features for future reference:

In [29]:
```
def plot_var(col_name, full_name, continuous):
    """
    Visualize a variable with and without faceting on the loan status.
    - 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 otherwise
    """
    f, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12,3), dpi=90)

    # Plot without loan status
    if continuous:
        sns.distplot(loans.loc[loans[col_name].notnull(), col_name], kde=False, ax=ax1)
    else:
        sns.countplot(loans[col_name], order=sorted(loans[col_name].unique()), color='#5975A4', saturation=1, ax=ax1
    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'].value_counts(normalize=True).loc[:,'Charged Off']
        sns.barplot(x=charge_off_rates.index, y=charge_off_rates.values, color='#5975A4', saturation=1, ax=ax2)
        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()
```

In [30]:
```
print(list(loans.columns))
```

```
['id', 'loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownershi
p', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'purpose', 'title', 'zip_code', 'addr_state', 'dti',
'earliest_cr_line', 'fico_range_low', 'fico_range_high', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
'initial_list_status', 'application_type', 'pub_rec_bankruptcies']
```

id

In [31]:
```
1 loans['id'].sample(5)
```

Out[31]:
```
21728     641001
6717      893714
9821      831941
10464     832618
2990     1027779
Name: id, dtype: object
```

In [32]:
```
1 loans['id'].describe()
```

Out[32]:
```
count       39239
unique      39239
top       1077501
freq            1
Name: id, dtype: int64
```

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

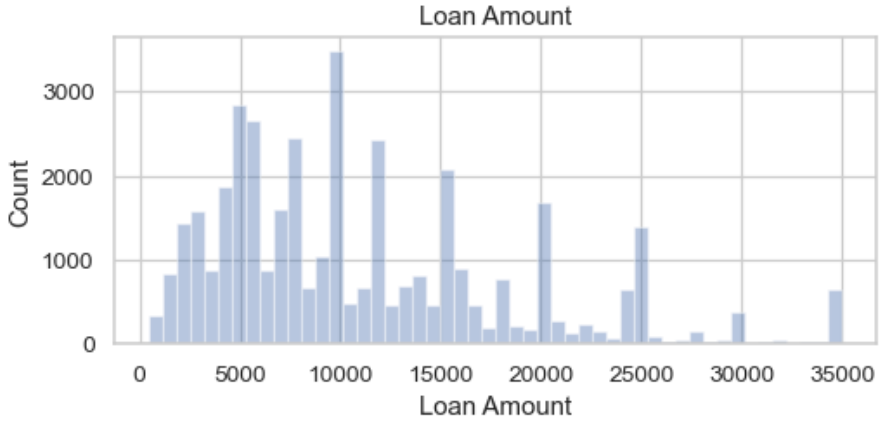
loan_amnt

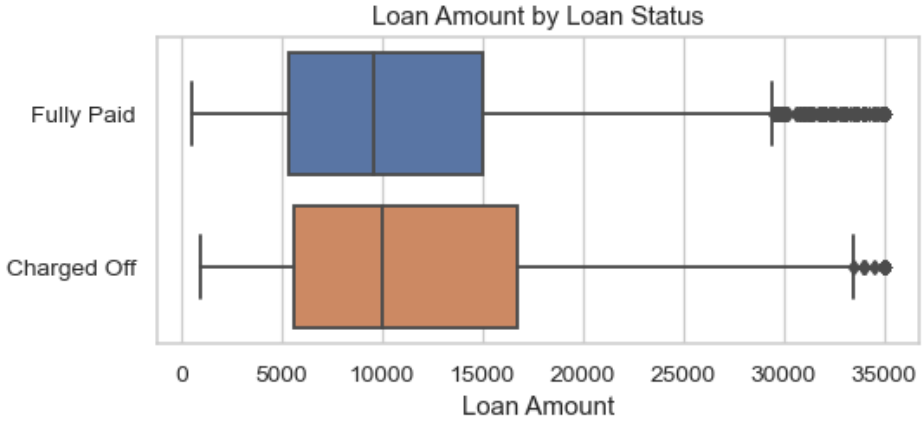
In [34]:
```
1 loans['loan_amnt'].describe()
```

Out[34]:
```
count    39239.000000
mean     11134.730115
std       7398.238030
min        500.000000
25%       5400.000000
50%      10000.000000
75%      15000.000000
max      35000.000000
Name: loan_amnt, dtype: float64
```

In [35]:
```
1 plot_var('loan_amnt', 'Loan Amount', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-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 [36]:
```
1 loans.groupby('loan_status')['loan_amnt'].describe()
```

Out[36]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	12133.849284	8099.601906	900.0	5600.0	10000.0	16750.0	35000.0
Fully Paid	33586.0	10966.564193	7260.165934	500.0	5375.0	9600.0	15000.0	35000.0

term

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

In [37]:
```
1 loans['term'].value_counts(dropna=False)
```

Out[37]:
```
 36 months    29096
 60 months    10143
Name: term, dtype: int64
```

convert term into integer

In [38]:
```
1 loans['term'] = loans['term'].apply(lambda s: np.int8(s.split()[0]))
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/15788848.py: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/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)
  loans['term'] = loans['term'].apply(lambda s: np.int8(s.split()[0]))
```

In [39]:
```
1 loans['term'].value_counts(normalize=True)
```

Out[39]:
```
36    0.741507
60    0.258493
Name: term, dtype: float64
```

Compare the charge-off rate by loan period:

In [40]:
```
1 loans.groupby('term')['loan_status'].value_counts(normalize=True).loc[:,'Charged Off']
```

Out[40]:
```
term
36    0.110909
60    0.239180
Name: loan_status, dtype: float64
```

int_rate

Data Dictionary: "Interest Rate on the loan."

In [41]:
```
1 loans['int_rate'].describe()
```

Out[41]:
```
count       39239
unique        371
top         10.99%
freq          957
Name: int_rate, dtype: object
```

In [42]:
```
1 loans["int_rate"] = loans["int_rate"].str[0:5]
2 loans['int_rate']
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/2284296573.py: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/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)
  loans["int_rate"] = loans["int_rate"].str[0:5]
```

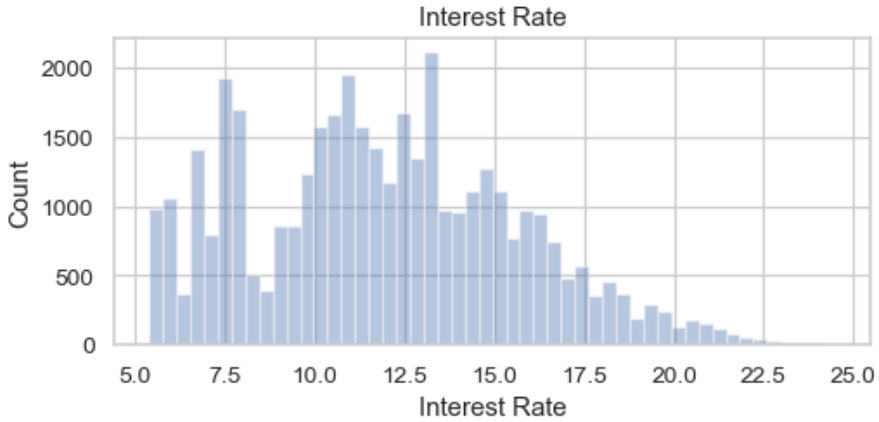
Out[42]:
```
0         10.6
1         15.2
2         15.9
3         13.4
5          7.9
         ...
39781      8.0
39782     10.2
39783      8.0
39784      7.4
39785     13.7
Name: int_rate, Length: 39239, dtype: object
```

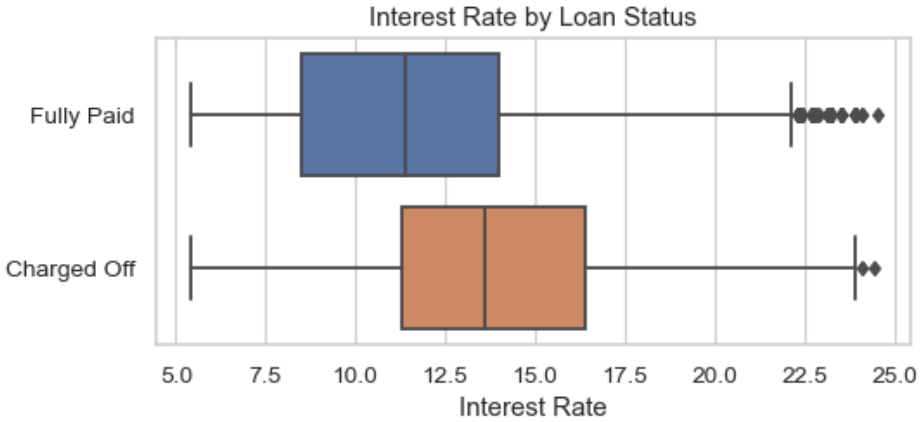
In [43]:
```
1 loans['int_rate']=loans['int_rate'].astype(str).astype(float)
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/2786868629.py: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/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)
  loans['int_rate']=loans['int_rate'].astype(str).astype(float)
```

In [44]:
```
1 plot_var('int_rate', 'Interest Rate', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [45]:
```
1 loans.groupby('loan_status')['int_rate'].describe()
```

Out[45]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	13.774544	3.653450	5.4	11.3	13.6	16.4	24.4
Fully Paid	33586.0	11.617046	3.623644	5.4	8.5	11.4	14.0	24.5

Installments

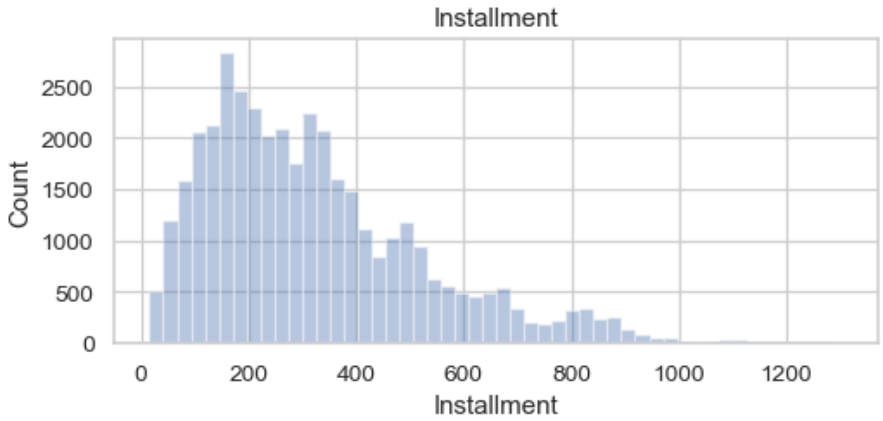
In [46]:
```
1 loans['installment'].describe()
```

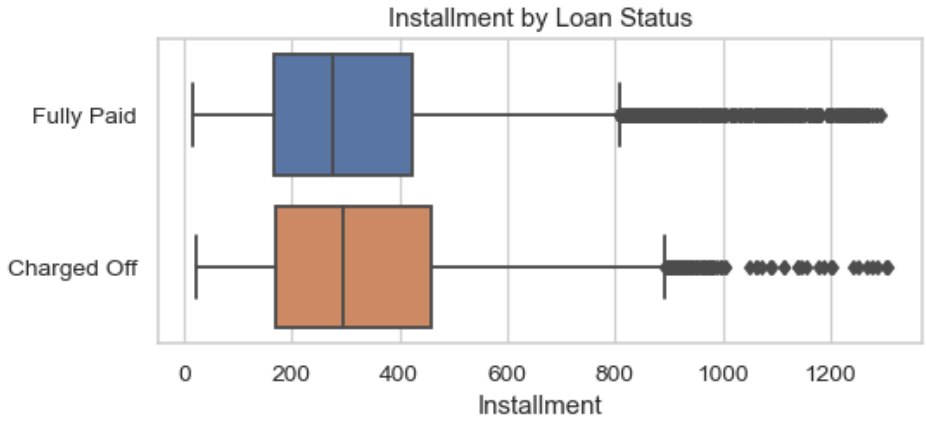
Out[46]:
```
count    39239.000000
mean       323.273499
std        208.463559
min         15.690000
25%        166.305000
50%        279.010000
75%        427.280000
max       1305.190000
Name: installment, dtype: float64
```

Installments range from 15.69 to 1305, with a median of 279.01.

In [47]:
```
1 plot_var('installment', 'Installment', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [48]:
```
1 loans.groupby('loan_status')['installment'].describe()
```

Out[48]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	336.621615	217.135181	22.79	168.74	294.7	458.46	1305.19
Fully Paid	33586.0	321.026822	206.886968	15.69	165.74	276.8	423.11	1295.21

grade, sub_grade

Data Dictionary for grade: "LendingClub assigned loan grade."

Data Dictionary for sub_grade: "LendingClub assigned loan subgrade."

What are the possible values of grade and sub_grade?

In [49]:
```
1 loans['grade']
```

Out[49]:
```
0        B
1        C
2        C
3        C
5        A
        ..
39781    A
39782    C
39783    A
39784    A
39785    E
Name: grade, Length: 39239, dtype: object
```

In [50]:
```
1 print(sorted(loans['grade'].unique()))
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

In [51]:
```
1 loans['sub_grade']
```

Out[51]:
```
0        B2
1        C4
2        C5
3        C1
5        A4
         ..
39781    A4
39782    C1
39783    A4
39784    A2
39785    E2
Name: sub_grade, Length: 39239, dtype: object
```

In [52]:
```
1 print(sorted(loans['sub_grade'].unique()))
```

```
['A1', 'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5', 'C1', 'C2', 'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D
5', 'E1', 'E2', 'E3', 'E4', 'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5']
```

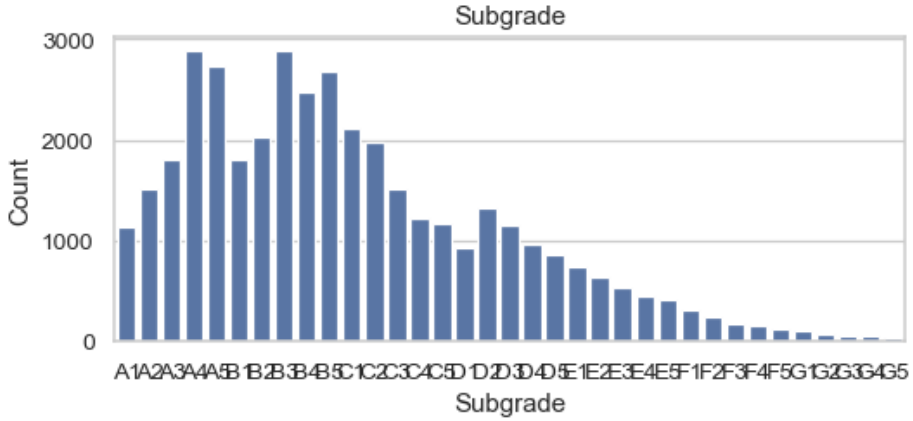
The grade is implied by the subgrade, so let's drop the grade column.

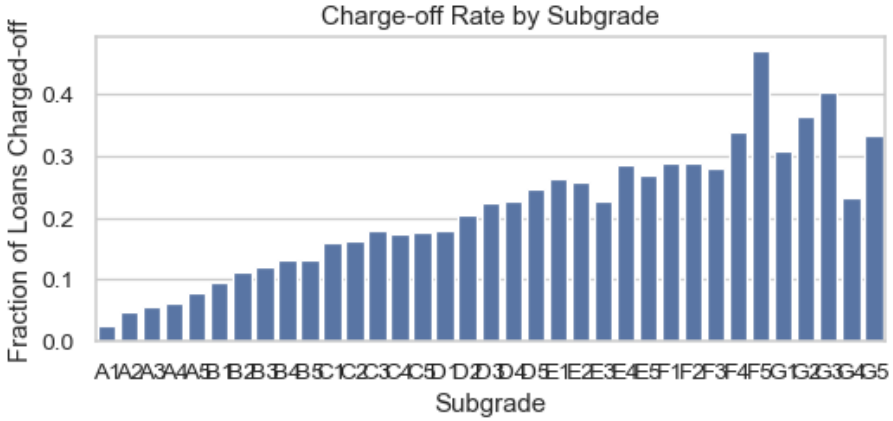
In [53]:
```
1 loans.drop('grade', axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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-co
py)
  return super().drop(
```

In [54]:
```
1 plot_var('sub_grade', 'Subgrade', continuous=False)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```





emp_title

In [55]:
```
1 loans['emp_title'].fillna(0)
```

Out[55]:
```
0                                0
1                              Ryder
2                                0
3                AIR RESOURCES BOARD
5               Veolia Transportaton
                     ...
39781                FiSite Research
39782    Squarewave Solutions, Ltd.
39783                            0
39784                            0
39785               Evergreen Center
Name: emp_title, Length: 39239, dtype: object
```

In [56]:
```
1 loans['emp_title'].describe()
```

Out[56]:
```
count       36812
unique      28467
top       US Army
freq          134
Name: emp_title, dtype: object
```

In [57]:
```
1 loans.drop(labels='emp_title', axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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)
  return super().drop(
```

emp_length

In [58]:
```
1 loans['emp_length']
```

Out[58]:
```
0        10+ years
1         < 1 year
2        10+ years
3        10+ years
5          3 years
           ...
39781      4 years
39782      3 years
39783     < 1 year
39784     < 1 year
39785     < 1 year
Name: emp_length, Length: 39239, dtype: object
```

In [59]:
```
1 loans['emp_length'].value_counts(dropna=False).sort_index()
```

Out[59]:
```
1 year       3214
10+ years    8717
2 years      4349
3 years      4054
4 years      3394
5 years      3250
6 years      2202
7 years      1742
8 years      1459
9 years      1245
< 1 year     4556
NaN          1057
Name: emp_length, dtype: int64
```

In [60]:
```
loans['emp_length'].replace(to_replace='10+ years', value='10 years', inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\generic.py:6619: 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)
  return self._update_inplace(result)
```

In [61]:
```
loans['emp_length'].replace('< 1 year', '0 years', inplace=True)
```

In [62]:
```
def emp_length_to_int(s):
    if pd.isnull(s):
        return s
    else:
        return np.int8(s.split()[0])
```

In [63]:
```
loans['emp_length'] = loans['emp_length'].apply(emp_length_to_int)
```

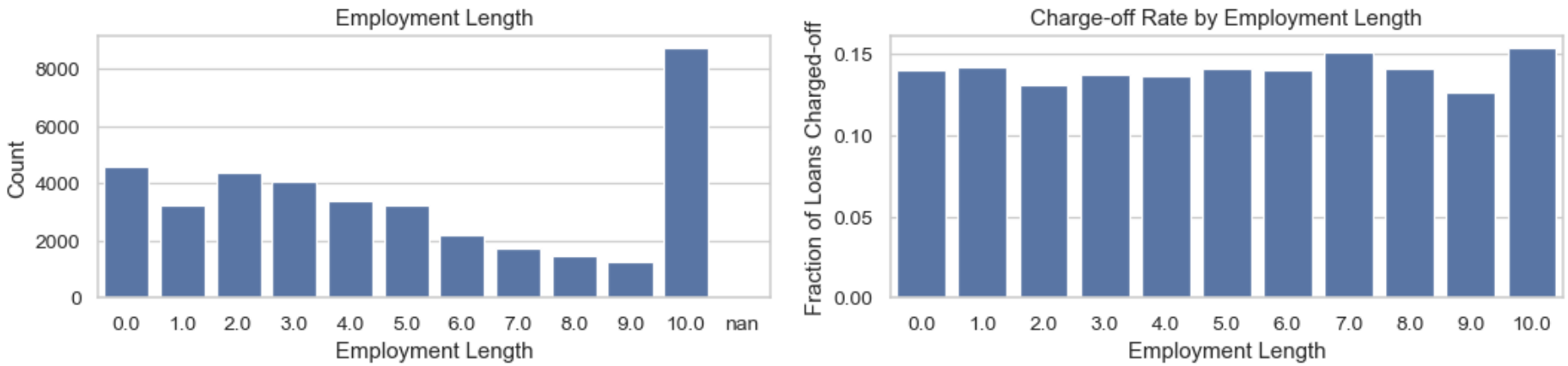
```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/3487204237.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/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)
  loans['emp_length'] = loans['emp_length'].apply(emp_length_to_int)
```

In [64]:
```
1 loans['emp_length'].value_counts(dropna=False).sort_index()
```

Out[64]:
```
0.0     4556
1.0     3214
2.0     4349
3.0     4054
4.0     3394
5.0     3250
6.0     2202
7.0     1742
8.0     1459
9.0     1245
10.0    8717
NaN     1057
Name: emp_length, dtype: int64
```

In [65]:
```
1 plot_var('emp_length', 'Employment Length', continuous=False)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Loan status does not appear to very much with employment length on average, except for a small drop in charge-offs for borrowers with over 10 years of employment.

home_ownership

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 [66]:
```
1 loans['home_ownership'].value_counts(dropna=False)
```

Out[66]:
```
RENT        18714
MORTGAGE    17396
OWN          3028
OTHER          98
NONE            3
Name: home_ownership, dtype: int64
```

In [67]:
```
1 loans['home_ownership'].replace(['NONE'], 'OTHER', inplace=True)
2
```

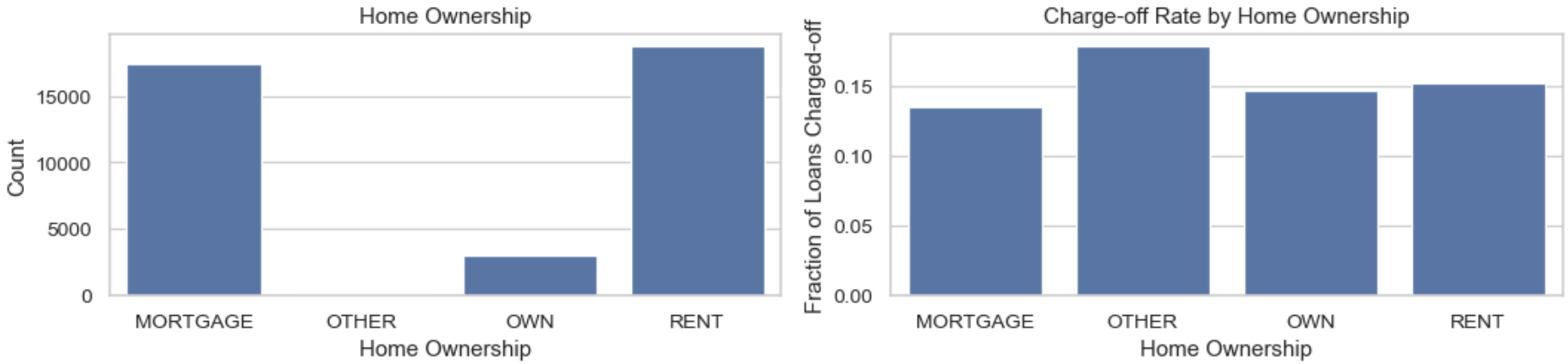
```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\generic.py:6619: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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)
  return self._update_inplace(result)
```

In [68]:
```
1 loans['home_ownership'].value_counts(dropna=False)
```

Out[68]:
```
RENT        18714
MORTGAGE    17396
OWN          3028
OTHER         101
Name: home_ownership, dtype: int64
```

In [69]:
```
1 plot_var('home_ownership', 'Home Ownership', continuous=False)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



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

In [70]:
```
1 loans.groupby('home_ownership')['loan_status'].value_counts(normalize=True).loc[:,'Charged Off']
```

Out[70]:
```
home_ownership
MORTGAGE    0.134744
OTHER       0.178218
OWN         0.146962
RENT        0.152079
Name: loan_status, dtype: float64
```

annual_inc

In [71]:
```
1 loans['annual_inc']
```

Out[71]:
```
0         24000.0
1         30000.0
2         12252.0
3         49200.0
5         36000.0
           ...
39781    110000.0
39782     18000.0
39783    100000.0
39784    200000.0
39785     22000.0
Name: annual_inc, Length: 39239, dtype: float64
```

In [72]:
```
1 loans['annual_inc'].describe()
```

Out[72]:
```
count    3.923900e+04
mean     6.888432e+04
std      6.400031e+04
min      4.000000e+03
25%      4.001400e+04
50%      5.900000e+04
75%      8.200000e+04
max      6.000000e+06
Name: annual_inc, dtype: float64
```

Annual income ranges from 4.0000+05 to 6.0000000+06, with a median of 5.900000+4. Because of the large range of incomes, we should take a log transform of the annual income variable.

In [73]:
```
1 loans['log_annual_inc'] = loans['annual_inc'].apply(lambda x: np.log10(x+1))
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/3393809776.py: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/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)
  loans['log_annual_inc'] = loans['annual_inc'].apply(lambda x: np.log10(x+1))
```

In [74]:
```
1 loans.drop('annual_inc', axis=1, inplace=True)
```

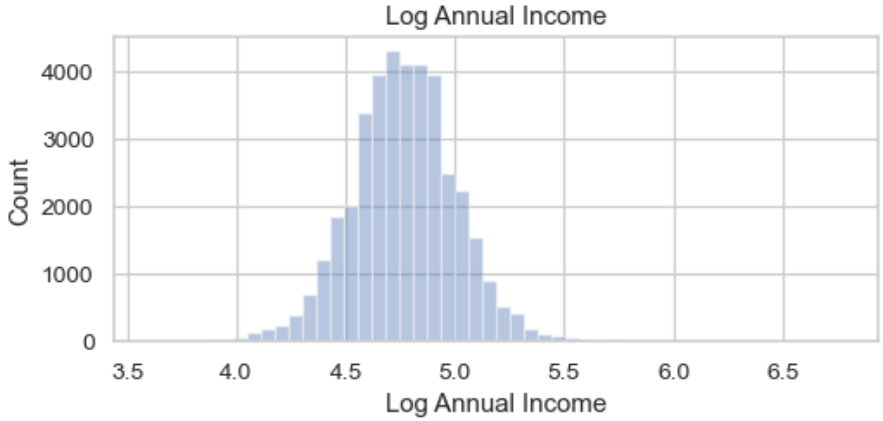
```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

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)
  return super().drop(
```

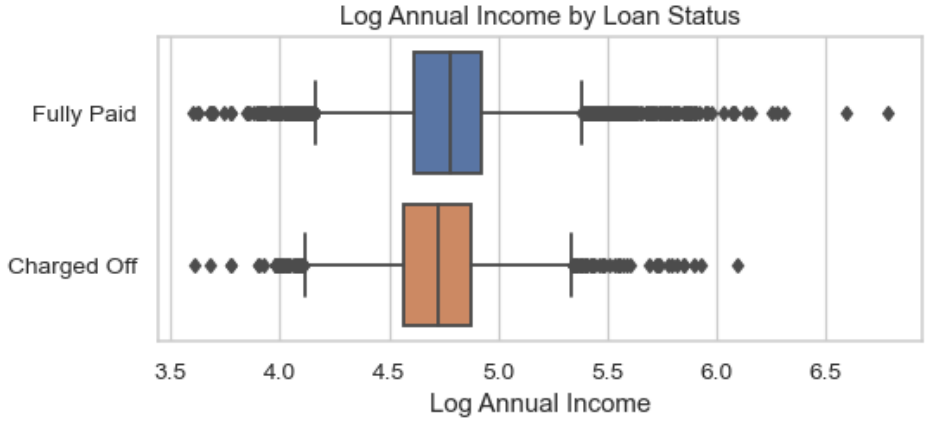
In [75]:
```
1 loans['log_annual_inc'].describe()
```

Out[75]:
```
count    39239.000000
mean         4.764773
std          0.242951
min          3.602169
25%          4.602223
50%          4.770859
75%          4.913819
max          6.778151
Name: log_annual_inc, dtype: float64
```

In [76]:
```
plot_var('log_annual_inc', 'Log Annual Income', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-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 [77]:
```
loans.groupby('loan_status')['log_annual_inc'].describe()
```

Out[77]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	4.723800	0.243079	3.610767	4.568307	4.724284	4.875067	6.096910
Fully Paid	33586.0	4.771669	0.242253	3.602169	4.618070	4.778158	4.924284	6.778151

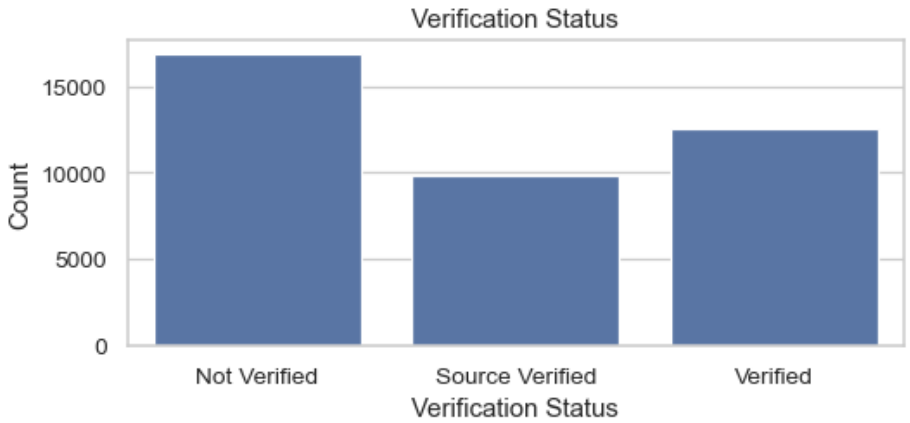
verification_status

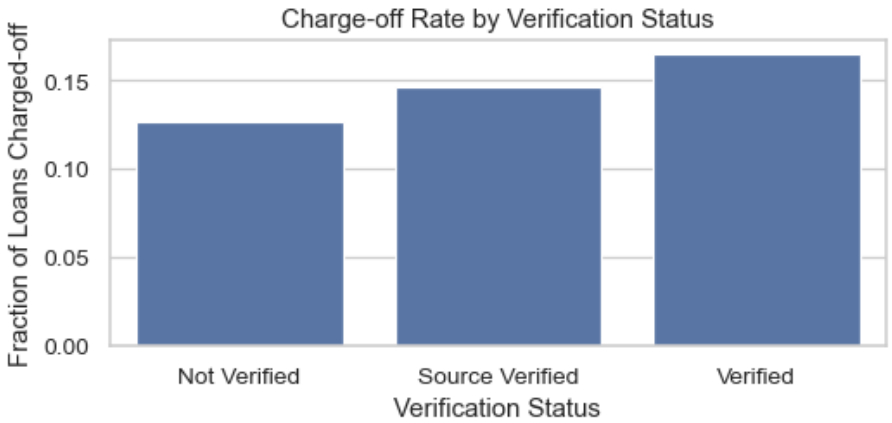
In [78]:
```
1 loans['verification_status']
```

Out[78]:
```
0              Verified
1        Source Verified
2           Not Verified
3        Source Verified
5        Source Verified
              ...
39781       Not Verified
39782       Not Verified
39783       Not Verified
39784       Not Verified
39785       Not Verified
Name: verification_status, Length: 39239, dtype: object
```

In [79]:
```
1 plot_var('verification_status', 'Verification Status', continuous=False)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```





issue_d

Data Dictionary: "The month which the loan was funded."

Because we're only using variables available to investors before the loan was funded, issue_d will not be included in the final model. We're keeping it for now just to perform the train/test split later, then we'll drop it.

purpose

In [80]:
```
1 loans['purpose']
```

Out[80]:
```
0              credit_card
1                      car
2           small_business
3                    other
5                  wedding
                  ...
39781      home_improvement
39782           credit_card
39783    debt_consolidation
39784                 other
39785    debt_consolidation
Name: purpose, Length: 39239, dtype: object
```

In [81]:
```
1 loans['purpose'].value_counts()
```

Out[81]:
```
debt_consolidation    18370
credit_card            5076
other                  3937
home_improvement       2949
major_purchase         2184
small_business         1796
car                    1536
wedding                 941
medical                 690
moving                  582
vacation                378
house                   372
educational             325
renewable_energy        103
Name: purpose, dtype: int64
```

In [82]:
```
1 loans.groupby('purpose')['loan_status'].value_counts(normalize=True).loc[:,'Charged Off'].sort_values()
```

Out[82]:
```
purpose
major_purchase        0.101648
wedding               0.102019
car                   0.104167
credit_card           0.107368
home_improvement      0.118345
vacation              0.140212
debt_consolidation    0.151606
medical               0.153623
moving                0.158076
house                 0.158602
other                 0.161290
educational           0.172308
renewable_energy      0.184466
small_business        0.265033
Name: loan_status, dtype: float64
```

Notice that only 10% complete loans for major_purchase have charged-off, but 26% ofcomplete small bussiness loans have charged-off.

title

In [83]:
```
1 loans['title']
```

Out[83]:
```
0                                    Computer
1                                        bike
2                          real estate business
3                                    personel
5         My wedding loan I promise to pay back
                          ...
39781                          Home Improvement
39782                 Retiring credit card debt
39783                    MBA Loan Consolidation
39784                                 JAL Loan
39785                       Consolidation Loan
Name: title, Length: 39239, dtype: object
```

In [84]:
```
1 loans['title'].describe()
```

Out[84]:
```
count                 39228
unique                19512
top       Debt Consolidation
freq                   2144
Name: title, dtype: object
```

In [85]:
```
1 loans['title'].value_counts().head(10)
```

Out[85]:
```
Debt Consolidation          2144
Debt Consolidation Loan     1671
Personal Loan                650
Consolidation                502
debt consolidation           495
Credit Card Consolidation    354
Home Improvement             350
Debt consolidation           331
Small Business Loan          317
Credit Card Loan             310
Name: title, dtype: int64
```

Ther are 19512 different title in the dataset, an the based on the top 10 purpose variable appears to already contain this information, so we drop the

title variable.

In [86]:
```
1 loans.drop('title', axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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)
  return super().drop(
```

zip_code, addr_state

In [87]:
```
1 loans['zip_code']
```

Out[87]:
```
0        860xx
1        309xx
2        606xx
3        917xx
5        852xx
         ...
39781    802xx
39782    274xx
39783    017xx
39784    208xx
39785    027xx
Name: zip_code, Length: 39239, dtype: object
```

In [88]:
```
1 loans['zip_code'].sample(5)
```

Out[88]:
```
19257    532xx
39155    300xx
27516    030xx
37282    891xx
23839    327xx
Name: zip_code, dtype: object
```

In [89]:
```
1 loans['zip_code'].nunique()
```

Out[89]:
```
822
```

In [90]:
```
1 loans['addr_state']
```

Out[90]:
```
0        AZ
1        GA
2        IL
3        CA
5        AZ
         ..
39781    CO
39782    NC
39783    MA
39784    MD
39785    MA
Name: addr_state, Length: 39239, dtype: object
```

In [91]:
```
1 loans['addr_state'].sample(5)
```

Out[91]:
```
24034    NY
28972    CA
9467     TX
4438     CA
22138    NY
Name: addr_state, dtype: object
```

In [92]:
```
1 loans['addr_state'].nunique()
```

Out[92]:
```
50
```

There are a lot of different zip codes, so let's just keep the state column.

In [93]:
```
1 loans.drop(labels='zip_code', axis=1, inplace=True)
```

In [94]:
```
1 loans.groupby('addr_state')['loan_status'].value_counts(normalize=True).loc[:,'Charged Off'].sort_values()
```

Out[94]:
```
addr_state
WY    0.048193
DC    0.070423
DE    0.105263
MS    0.105263
VT    0.113208
AR    0.115702
KS    0.116981
TN    0.117647
TX    0.118236
WV    0.120690
PA    0.120908
MA    0.121693
AL    0.122172
LA    0.122685
CO    0.125964
RI    0.126904
CT    0.127371
VA    0.128150
OH    0.128952
IL    0.130779
MT    0.130952
NY    0.131950
MN    0.132570
OK    0.138983
WI    0.140625
SC    0.140725
KY    0.141066
AZ    0.144175
MI    0.144847
NC    0.147477
NH    0.147929
WA    0.152828
NJ    0.153005
GA    0.155684
MD    0.156580
NM    0.160428
CA    0.160478
OR    0.160633
```

```
UT    0.160784
HI    0.163743
ID    0.166667
MO    0.168142
FL    0.179252
AK    0.189873
SD    0.193548
NV    0.222904
NE    0.600000
Name: loan_status, dtype: float64
```

The charge-off rate ranges from .048% in Wyoming, WY to 60% in Nebraska.

dti

In [95]:
```
1 loans['dti']
```

Out[95]:
```
0        27.65
1         1.00
2         8.72
3        20.00
5        11.20
         ...
39781    11.33
39782     6.40
39783     2.30
39784     3.72
39785    14.29
Name: dti, Length: 39239, dtype: float64
```

In [96]:
```
1 loans['dti'].describe()
```

Out[96]:
```
count    39239.000000
mean        13.293984
std          6.676607
min          0.000000
25%          8.160000
50%         13.390000
75%         18.570000
max         29.990000
Name: dti, dtype: float64
```

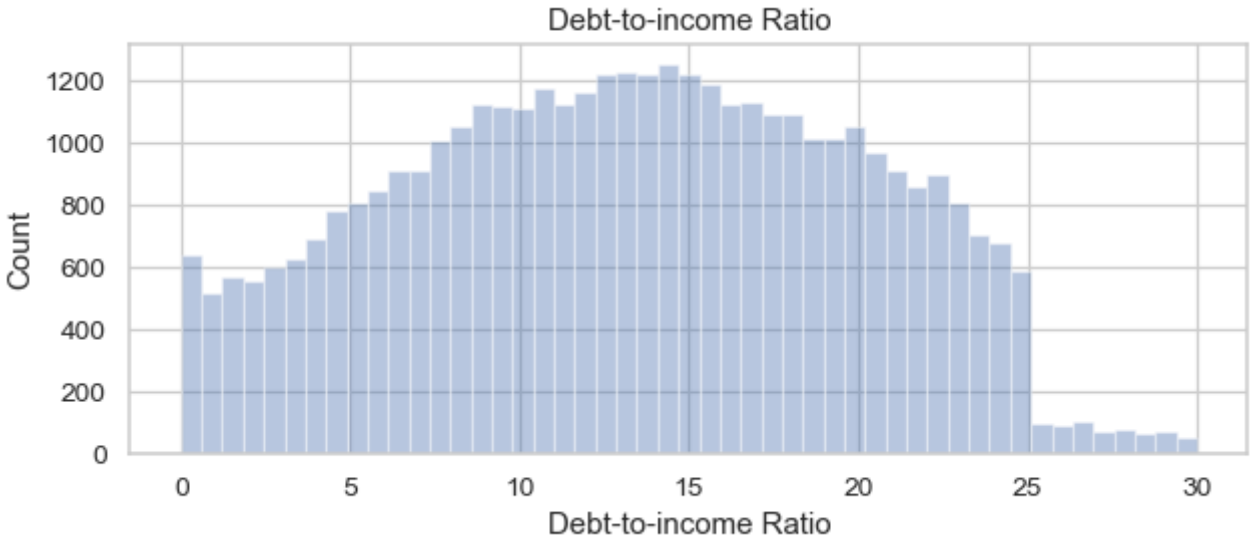
Note sure if the values of 0 and 29 make sense...

There are several outliers that mess up our default plots. Plot a histogram for dti less than 60:

In [97]:
```
plt.figure(figsize=(8,3), dpi=90)
sns.distplot(loans.loc[loans['dti'].notnull() & (loans['dti']<60), 'dti'], kde=False)
plt.xlabel('Debt-to-income Ratio')
plt.ylabel('Count')
plt.title('Debt-to-income Ratio')

```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[97]: Text(0.5, 1.0, 'Debt-to-income Ratio')



How many of the dti values are "outliers" (above 60)?

In [98]:
```
(loans['dti']>=60).sum()
```

Out[98]:
```
0
```

Very few. Compare the summary statistics by loan status:

In [99]:
```
loans.groupby('loan_status')['dti'].describe()
```

Out[99]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	14.007444	6.58172	0.0	9.06	14.31	19.29	29.85
Fully Paid	33586.0	13.173899	6.68506	0.0	8.00	13.22	18.43	29.99

earliest_cr_line

In [100]:
```
loans['earliest_cr_line']
```

Out[100]:
```
0        Jan-1985
1        Apr-1999
2        Nov-2001
3        Feb-1996
5        Nov-2004
           ...
39781    Nov-1990
39782    Dec-1986
39783    Oct-1998
39784    Nov-1988
39785    Oct-2003
Name: earliest_cr_line, Length: 39239, dtype: object
```

In [101]:
```
1 loans['earliest_cr_line'].sample(5)
```

Out[101]:
```
22711    Feb-1991
15562    Feb-2002
3194     Sep-1994
11517    Nov-1999
8250     Sep-1977
Name: earliest_cr_line, dtype: object
```

In [102]:
```
1 loans['earliest_cr_line'].isnull().any()
```

Out[102]:
```
False
```

Let's just retain the year for simplicity:

In [103]:
```
1 loans['earliest_cr_line'] = loans['earliest_cr_line'].apply(lambda s: int(s[-4:]))
```

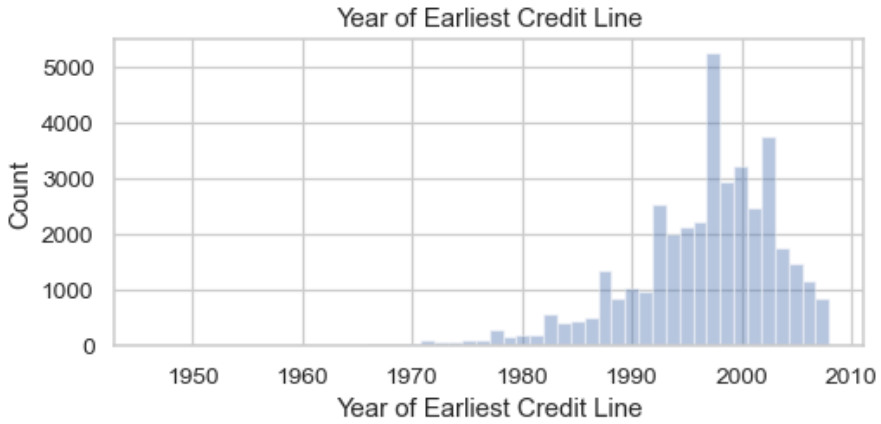
```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/1237313599.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/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)
  loans['earliest_cr_line'] = loans['earliest_cr_line'].apply(lambda s: int(s[-4:]))
```

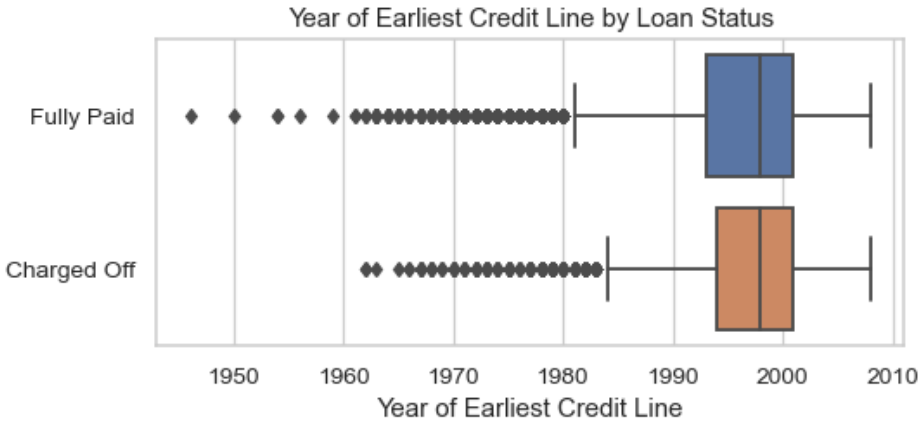
In [104]:
```
1 loans['earliest_cr_line'].describe()
```

Out[104]:
```
count    39239.000000
mean      1996.574913
std          6.826492
min       1946.000000
25%       1993.000000
50%       1998.000000
75%       2001.000000
max       2008.000000
Name: earliest_cr_line, dtype: float64
```

In [105]:
```
1 plot_var('earliest_cr_line', 'Year of Earliest Credit Line', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





fico_range_low, fico_range_high

Data Dictionary for fico_range_low: "The lower boundary range the borrower’s FICO at loan origination belongs to."

Data Dictionary for fico_range_high: "The upper boundary range the borrower’s FICO at loan origination belongs to."

In [106]:
```
1 loans[['fico_range_low', 'fico_range_high']].describe()
```

Out[106]:

	fico_range_low	fico_range_high
count	39239.000000	39239.000000
mean	715.000765	719.000765
std	35.868102	35.868102
min	625.000000	629.000000
25%	685.000000	689.000000
50%	710.000000	714.000000
75%	740.000000	744.000000
max	825.000000	829.000000

In [107]:
```
1 loans[['fico_range_low','fico_range_high']].corr()
```

Out[107]:

	fico_range_low	fico_range_high
fico_range_low	1.0	1.0
fico_range_high	1.0	1.0

In [108]:
```
1 loans['fico_score'] = 0.5*loans['fico_range_low'] + 0.5*loans['fico_range_high']
```

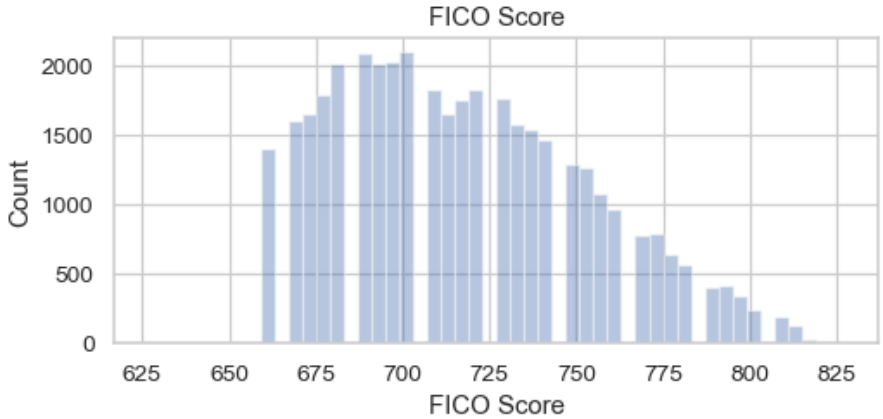
```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/1640605821.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/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)
  loans['fico_score'] = 0.5*loans['fico_range_low'] + 0.5*loans['fico_range_high']
```

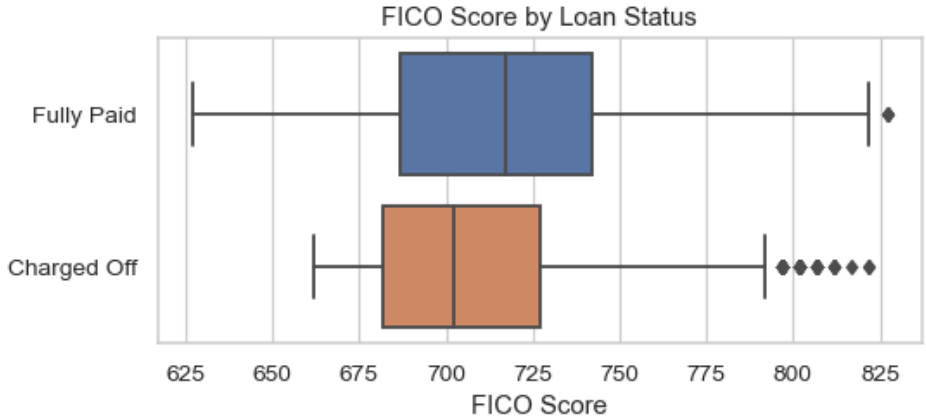
In [109]:
```
1 loans.drop(['fico_range_high', 'fico_range_low'], axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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)
  return super().drop(
```

In [110]:
```
1 plot_var('fico_score', 'FICO Score', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [111]:
```
1 loans.groupby('loan_status')['fico_score'].describe()
```

Out[111]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	705.644083	31.885577	662.0	682.0	702.0	727.0	822.0
Fully Paid	33586.0	718.912255	36.147005	627.0	687.0	717.0	742.0	827.0

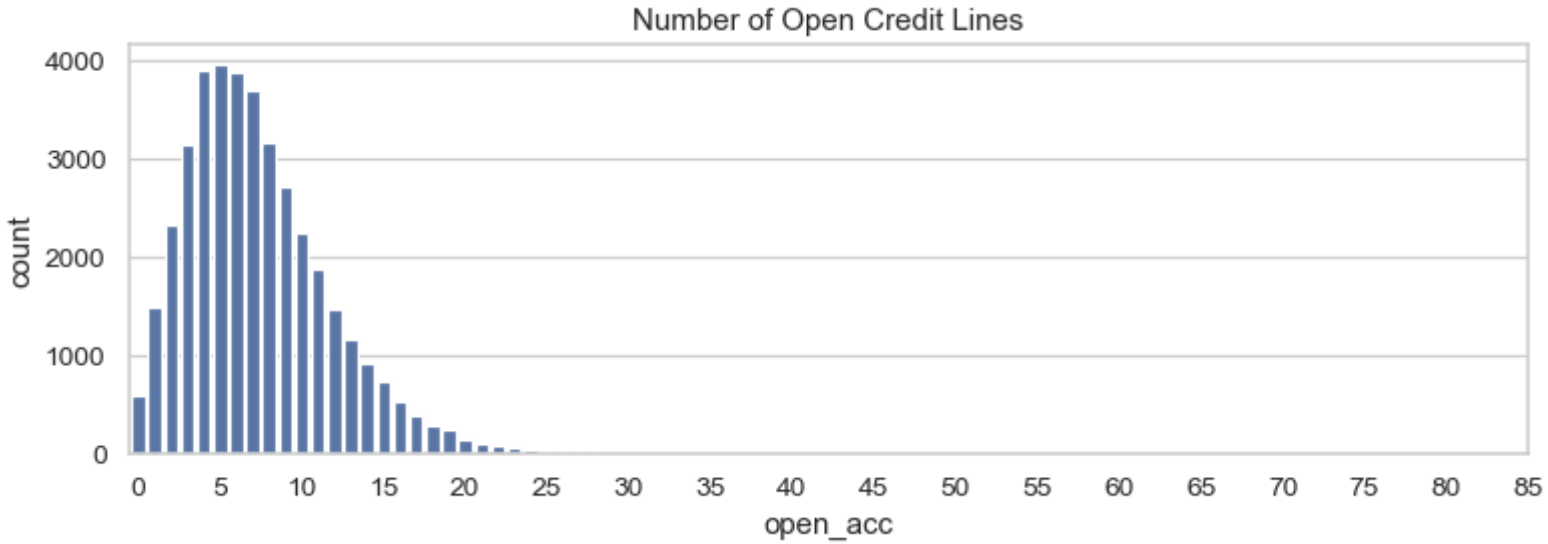
Loans that charge off have a FICO score 10 points lower on average.

open_acc

In [112]:
```
plt.figure(figsize=(10,3), dpi=90)
sns.countplot(loans['open_acc'], order=sorted(loans['open_acc'].unique()), color='#5975A4', saturation=1)
_, _ = plt.xticks(np.arange(0, 90, 5), np.arange(0, 90, 5))
plt.title('Number of Open Credit Lines')

```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[112]:
```
Text(0.5, 1.0, 'Number of Open Credit Lines')
```



Is there a difference in number of credit lines between fully paid loans and charged-off loans?

In [113]:
```
1 loans.groupby('loan_status')['open_acc'].describe()
```

Out[113]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	9.180789	4.526857	2.0	6.0	8.0	12.0	38.0
Fully Paid	33586.0	9.304293	4.380524	2.0	6.0	9.0	12.0	44.0

There's does not appear to be a significant difference.

pub_rec

In [114]:
```
1 loans['pub_rec']
```

Out[114]:
```
0        0.0
1        0.0
2        0.0
3        0.0
5        0.0
        ...
39781    0.0
39782    0.0
39783    0.0
39784    0.0
39785    0.0
Name: pub_rec, Length: 39239, dtype: float64
```

In [115]:
```
1 loans['pub_rec'].value_counts().sort_index()
```

Out[115]:
```
0.0    37136
1.0     2044
2.0       49
3.0        8
4.0        2
Name: pub_rec, dtype: int64
```

In [116]:
```
1 loans.groupby('loan_status')['pub_rec'].describe()
```

Out[116]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	0.084911	0.285049	0.0	0.0	0.0	0.0	2.0
Fully Paid	33586.0	0.050438	0.228826	0.0	0.0	0.0	0.0	4.0

revol_bal

In [117]:
```
1 loans['revol_bal']
```

Out[117]:
```
0        13648.0
1         1687.0
2         2956.0
3         5598.0
5         7963.0
           ...
39781     7274.0
39782     8847.0
39783     9698.0
39784    85607.0
39785     4175.0
Name: revol_bal, Length: 39239, dtype: float64
```

In [118]:
```
1 loans['revol_bal'].describe()
```

Out[118]:
```
count     39239.000000
mean      13329.338898
std       15876.810124
min           0.000000
25%        3670.000000
50%        8803.000000
75%       16981.500000
max      149588.000000
Name: revol_bal, dtype: float64
```

Do a log transform:

In [119]:
```
1 loans['log_revol_bal'] = loans['revol_bal'].apply(lambda x: np.log10(x+1))
```

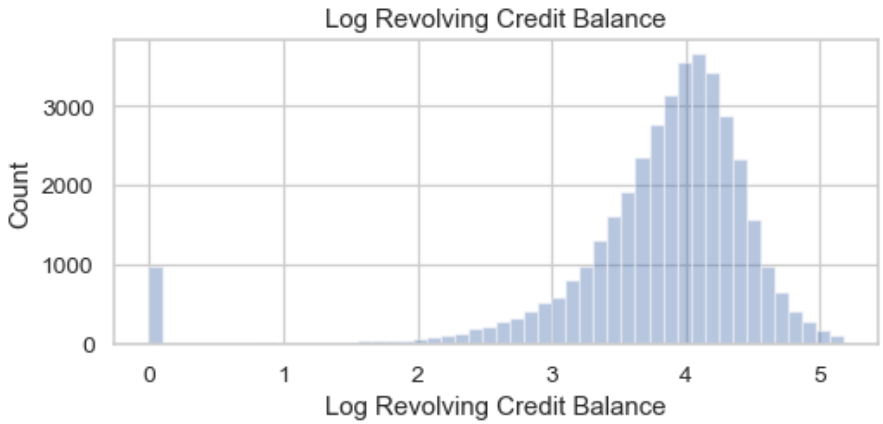
```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/1354505710.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/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)
  loans['log_revol_bal'] = loans['revol_bal'].apply(lambda x: np.log10(x+1))
```

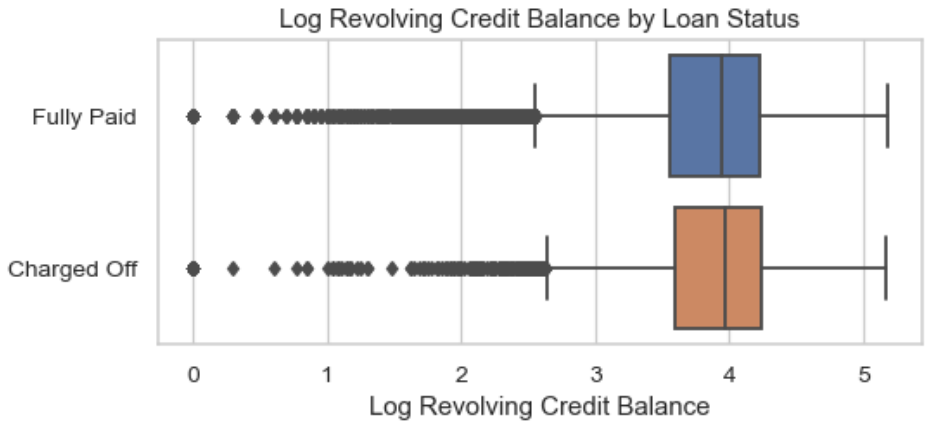
In [120]:
```
1 loans.drop('revol_bal', axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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)
  return super().drop(
```

In [121]:
```
1 plot_var('log_revol_bal', 'Log Revolving Credit Balance', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [122]:
```
1 loans.groupby('loan_status')['log_revol_bal'].describe()
```

Out[122]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	3.788520	0.840733	0.0	3.600428	3.966329	4.244698	5.17269
Fully Paid	33586.0	3.772857	0.821528	0.0	3.559068	3.941089	4.227643	5.17490

revol_util

In [123]:
```
1 loans['revol_util'].describe()
```

Out[123]:
```
count     39189
unique     1089
top          0%
freq        972
Name: revol_util, dtype: object
```

In [124]:
```
1 loans['revol_util'].str[:4]
```

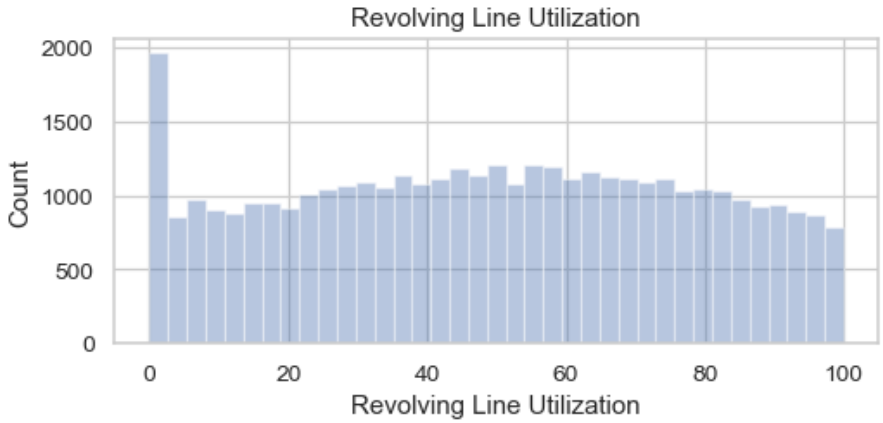
Out[124]:
```
0        83.7
1        9.4%
2        98.5
3         21%
5        28.3
         ...
39781    13.1
39782    26.9
39783    19.4
39784    0.7%
39785    51.5
Name: revol_util, Length: 39239, dtype: object
```

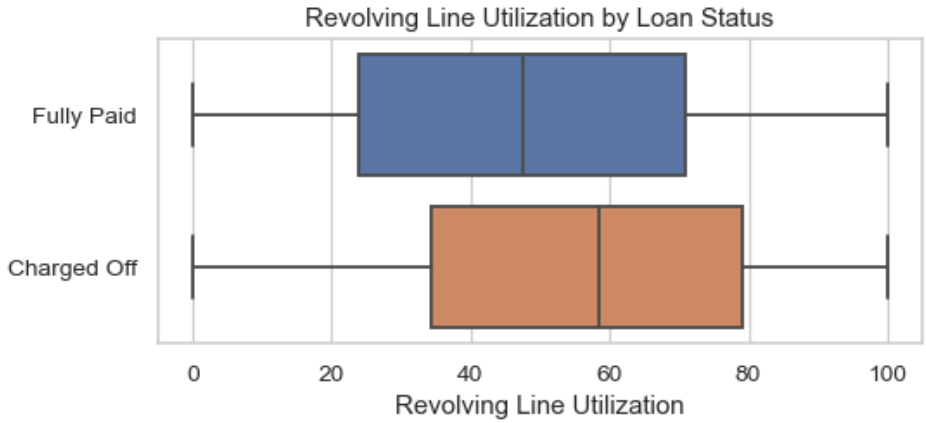
In [125]:
```
1 loans['revol_util'] = loans['revol_util'].str.rstrip('%').astype('float')
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/1838940800.py: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/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)
  loans['revol_util'] = loans['revol_util'].str.rstrip('%').astype('float')
```

In [126]:
```
1 plot_var('revol_util', 'Revolving Line Utilization', continuous=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated fun
ction and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function
with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```





In [127]:
```
1 loans.groupby('loan_status')['revol_util'].describe()
```

Out[127]:

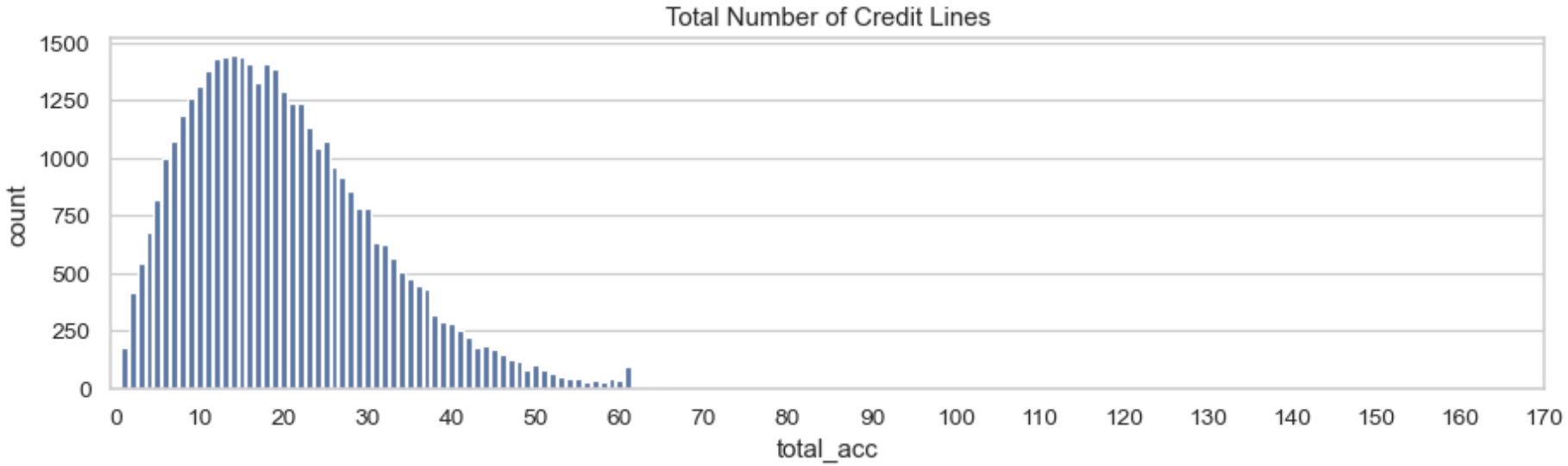
loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5637.0	55.618416	27.899443	0.0	34.4	58.4	79.1	99.9
Fully Paid	33552.0	47.581640	28.262587	0.0	24.0	47.6	70.8	99.9

total_acc

In [128]:
```
plt.figure(figsize=(12,3), dpi=90)
sns.countplot(loans['total_acc'], order=sorted(loans['total_acc'].unique()), color='#5975A4', saturation=1)
_, _ = plt.xticks(np.arange(0, 176, 10), np.arange(0, 176, 10))
plt.title('Total Number of Credit Lines')
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[128]:
```
Text(0.5, 1.0, 'Total Number of Credit Lines')
```



In [129]:
```
1 loans.groupby('loan_status')['total_acc'].describe()
```

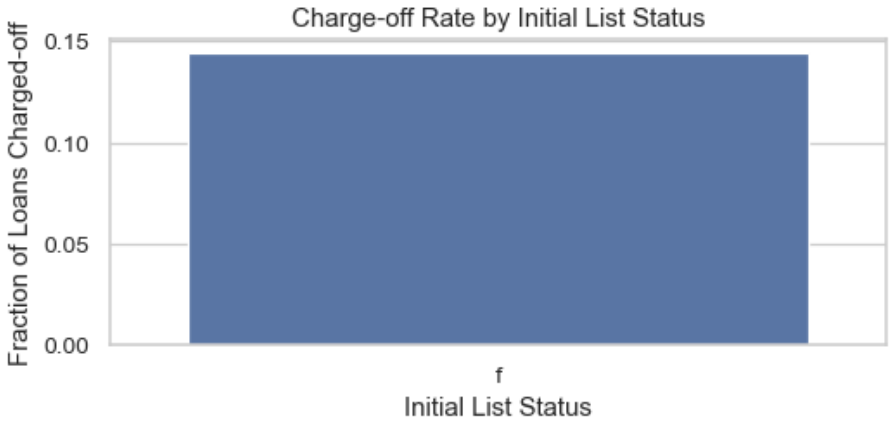
Out[129]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	5653.0	21.439236	11.443764	2.0	13.0	20.0	28.0	74.0
Fully Paid	33586.0	22.179003	11.403770	2.0	14.0	21.0	29.0	90.0

initial_list_status

In [130]:
```
1 plot_var('initial_list_status', 'Initial List Status', continuous=False)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



application_type

In [131]:
```
1 loans['application_type']
```

Out[131]:
```
0         INDIVIDUAL
1         INDIVIDUAL
2         INDIVIDUAL
3         INDIVIDUAL
5         INDIVIDUAL
             ...
39781     INDIVIDUAL
39782     INDIVIDUAL
39783     INDIVIDUAL
39784     INDIVIDUAL
39785     INDIVIDUAL
Name: application_type, Length: 39239, dtype: object
```

In [132]:
```
1 loans['application_type'].value_counts()
```

Out[132]:
```
INDIVIDUAL    39239
Name: application_type, dtype: int64
```

In [133]:
```
1 loans.groupby('application_type')['loan_status'].value_counts(normalize=True).loc[:,'Charged Off']
```

Out[133]:
```
application_type
INDIVIDUAL    0.144066
Name: loan_status, dtype: float64
```

pub_rec_bankruptcies

In [134]:
```
1 loans['pub_rec_bankruptcies']
```

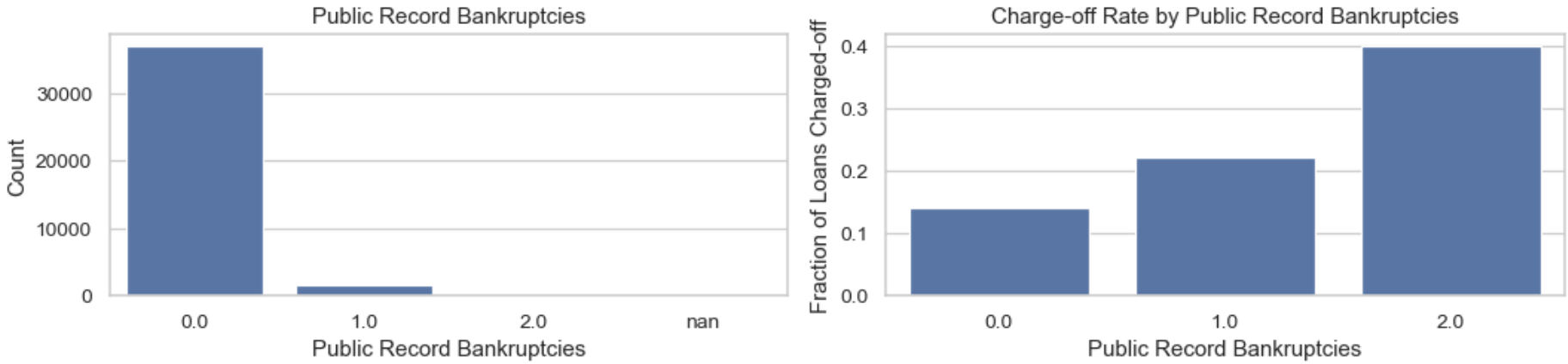
Out[134]:
```
0        0.0
1        0.0
2        0.0
3        0.0
5        0.0
        ...
39781    NaN
39782    NaN
39783    NaN
39784    NaN
39785    NaN
Name: pub_rec_bankruptcies, Length: 39239, dtype: float64
```

In [135]:
```
1 loans['pub_rec_bankruptcies'].value_counts().sort_index()
```

Out[135]:
```
0.0    36872
1.0     1665
2.0        5
Name: pub_rec_bankruptcies, dtype: int64
```

In [136]:
```
1 plot_var('pub_rec_bankruptcies', 'Public Record Bankruptcies', continuous=False)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a k
eyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments withou
t an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



More Pre-processing

In [137]:
```
loans['charged_off'] = (loans['loan_status'] == 'Charged Off').apply(np.int8)
loans.drop('loan_status', axis=1, inplace=True)
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/150504833.py: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/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)
  loans['charged_off'] = (loans['loan_status'] == 'Charged Off').apply(np.int8)
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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)
  return super().drop(
```

In [138]:
```
loans.shape
```

Out[138]:
```
(39239, 24)
```

If any categorical variables have missing values, we'll need to create NaN dummy variables for those. So first check which variables have missing data:

In [139]:
```
missing_fractions = loans.isnull().mean().sort_values(ascending=False) # Fraction of data missing for each variable
```

In [140]:
```
print(missing_fractions[missing_fractions > 0]) # Print variables that are missing data
```

```
emp_length              0.026937
pub_rec_bankruptcies    0.017763
revol_util              0.001274
dtype: float64
```

There are no categorical variables with missing values, and therefore we don't need any NaN dummy variables.

Create dummy variables for the categorical variables:

In [141]:
```
print(loans.columns)
```

```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'sub_grade',
       'emp_length', 'home_ownership', 'verification_status', 'issue_d',
       'purpose', 'addr_state', 'dti', 'earliest_cr_line', 'open_acc',
       'pub_rec', 'revol_util', 'total_acc', 'initial_list_status',
       'application_type', 'pub_rec_bankruptcies', 'log_annual_inc',
       'fico_score', 'log_revol_bal', 'charged_off'],
      dtype='object')
```

In [142]:
```
loans = pd.get_dummies(loans, columns=['sub_grade', 'home_ownership', 'verification_status', 'purpose', 'addr_state'
```

In [143]:
```
loans.shape
```

Out[143]: (39239, 118)

In [144]:
```
loans.sample(5)
```

Out[144]:

	loan_amnt	term	int_rate	installment	emp_length	issue_d	dti	earliest_cr_line	open_acc	pub_rec	...	addr_state_SD	addr_state_TN	a
5951	12000.0	60	13.4	276.06	2.0	Oct-2011	16.22	2002	8.0	0.0	...	0	0	
3919	24000.0	60	17.5	603.98	10.0	Nov-2011	20.96	1979	21.0	0.0	...	0	0	
35267	18000.0	36	11.8	596.41	6.0	Sep-2009	8.22	1989	10.0	0.0	...	0	0	
30654	10000.0	36	12.7	335.67	5.0	Apr-2010	14.55	2000	8.0	0.0	...	0	0	
35350	12000.0	36	8.9	381.26	7.0	Sep-2009	11.01	1986	7.0	0.0	...	0	0	

5 rows × 118 columns

Train/test split

In [145]:
```
1 loans['issue_d'].sample(5)
```

Out[145]:
```
28061    Jul-2010
30806    Apr-2010
17717    Mar-2011
17497    Apr-2011
27313    Aug-2010
Name: issue_d, dtype: object
```

In [146]:
```
1 loans['issue_d'].isnull().any()
```

Out[146]:
```
False
```

No. Let's convert the issue dates to datetime objects:

In [147]:
```
1 loans['issue_d'] = pd.to_datetime(loans['issue_d'])
```

In [148]:
```
1 loans['issue_d'].sample(5)
```

Out[148]:
```
34457   2009-10-01
3962    2011-11-01
16922   2011-04-01
23998   2010-11-01
1374    2011-12-01
Name: issue_d, dtype: datetime64[ns]
```

The new datetime values are all on the first day of the month. Check the summary statistics of the issue dates:

In [149]:
```
1 loans['issue_d'].describe()
```

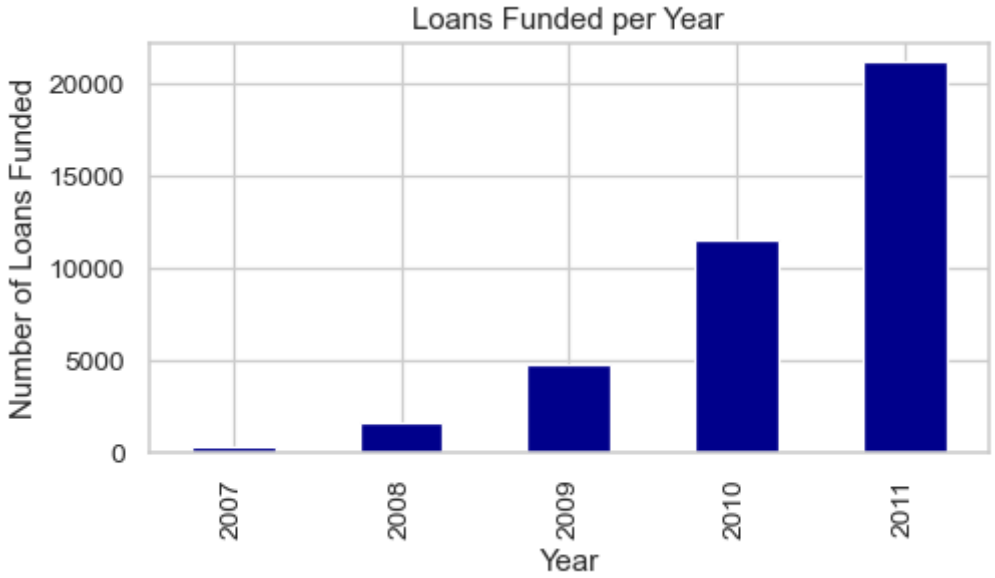
```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/2940031951.py:1: FutureWarning: Treating datetime data as categorical
rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specify `datetime_i
s_numeric=True` to silence this warning and adopt the future behavior now.
  loans['issue_d'].describe()
```

Out[149]:
```
count                   39239
unique                     55
top       2011-11-01 00:00:00
freq                     2085
first     2007-06-01 00:00:00
last      2011-12-01 00:00:00
Name: issue_d, dtype: object
```

There are only 55 unique issue dates over the 11-year period because we only have month/year information. In this particular dataset, the first loans were issued in June 2007, and the most recent loans were issued in December 2011. The busiest month was November 2011 with 2085 loans funded in that month. What is the distribution of loans funded in each year?

In [150]:
```
plt.figure(figsize=(6,3), dpi=90)
loans['issue_d'].dt.year.value_counts().sort_index().plot.bar(color='darkblue')
plt.xlabel('Year')
plt.ylabel('Number of Loans Funded')
plt.title('Loans Funded per Year')
```

Out[150]:
```
Text(0.5, 1.0, 'Loans Funded per Year')
```



In [151]:
```
loans_train = loans.loc[loans['issue_d'] <  loans['issue_d'].quantile(0.9)]
loans_test =  loans.loc[loans['issue_d'] >= loans['issue_d'].quantile(0.9)]
```

In [152]:
```
print('Number of loans in the partition:   ', loans_train.shape[0] + loans_test.shape[0])
print('Number of loans in the full dataset:', loans.shape[0])
```

```
Number of loans in the partition:    39239
Number of loans in the full dataset: 39239
```

What is test size?

In [153]:
```
1 loans_test.shape[0] / loans.shape[0]
```

Out[153]:
```
0.1061189123066337
```

About 10.6%. The partition looks good, so let we consider, we can delete the original loans dataframe:

In [154]:
```
1 loans_train['issue_d'].describe()
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/1587581987.py:1: FutureWarning: Treating datetime data as categorical
rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specify `datetime_i
s_numeric=True` to silence this warning and adopt the future behavior now.
  loans_train['issue_d'].describe()
```

Out[154]:
```
count                   35075
unique                     53
top       2011-09-01 00:00:00
freq                     2017
first     2007-06-01 00:00:00
last      2011-10-01 00:00:00
Name: issue_d, dtype: object
```

In [155]:
```
1 loans_test['issue_d'].describe()
```

```
C:\Users\Admin\AppData\Local\Temp/ipykernel_11992/4293394310.py:1: FutureWarning: Treating datetime data as categorical
rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specify `datetime_i
s_numeric=True` to silence this warning and adopt the future behavior now.
  loans_test['issue_d'].describe()
```

Out[155]:
```
count                    4164
unique                      2
top       2011-11-01 00:00:00
freq                     2085
first     2011-11-01 00:00:00
last      2011-12-01 00:00:00
Name: issue_d, dtype: object
```

The training set includes loans from June 2007 to October 2011. The test set includes loans from November 2011 to December 2011.

Now we need to delete the issue_d variable, because it was not available before the loan was funded.

In [156]:
```
loans_train.drop('issue_d', axis=1, inplace=True)
loans_test.drop('issue_d', axis=1, inplace=True)
```

```
C:\Users\Admin\anaconda3\lib\site-packages\pandas\core\frame.py:4906: 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)
  return super().drop(
```

Now separate the predictor variables from the response variable:

In [157]:
```
y_train = loans_train['charged_off']
y_test = loans_test['charged_off']
```

In [158]:
```
X_train = loans_train.drop('charged_off', axis=1)
X_test = loans_test.drop('charged_off', axis=1)
```

Linear Dependence of Charge-off on the Predictors

In [159]:
```
linear_dep = pd.DataFrame()
```

Pearson correlations:

In [160]:
```
1 print(X_train.dtypes)
```

```
loan_amnt        float64
term               int8
int_rate         float64
installment      float64
emp_length       float64
                  ...
addr_state_VT      uint8
addr_state_WA      uint8
addr_state_WI      uint8
addr_state_WV      uint8
addr_state_WY      uint8
Length: 116, dtype: object
```

In [161]:
```
1 for col in X_train.dtypes:
2     print(col)
```

```
float64
int8
float64
float64
float64
float64
int64
float64
float64
float64
float64
float64
float64
float64
float64
uint8
uint8
uint8
uint8
```

In [ ]:
```
1
```

In [162]:
```
1 #X_train[col].dtypes
```

In [163]:
```
1 # X_train.drop('emp_title',axis=1,inplace=True)
```

In [164]:
```
1 # X_train.drop('zip_code',axis=1,inplace=True)
```

In [ ]:
```
1
```

In [166]:
```
1 for col in X_train.columns:
2     linear_dep.loc[col, 'pearson_corr'] = X_train[col].corr(y_train)
3 linear_dep['abs_pearson_corr'] = abs(linear_dep['pearson_corr'])
```

In [169]:
```
1 for col in X_train.columns:
2     print(linear_dep.loc[col, 'pearson_corr'])
```

```
0.044438129416275916
0.1452957180496136
0.1918006227543228
0.01833818208139645
0.012956178415630158
0.03927461576767455
0.02013805443074666
-0.012764811165652961
0.05501879548815001
0.0940817848659754
-0.02489615652310805
0.05189107157940904
-0.07073908172155081
-0.12755297445701236
0.00457443338323912
-0.052704005071748175
-0.05803666815910609
-0.06468467713529052
-0.04874648865907472
```

F-statistics:

In [167]:
```
from sklearn.feature_selection import f_classif
for col in X_train.columns:
    mask = X_train[col].notnull()
    (linear_dep.loc[col, 'F'], linear_dep.loc[col, 'p_value']) = f_classif(pd.DataFrame(X_train.loc[mask, col]), y_t
```

Sort the results by the absolute value of the Pearson correlation:

In [170]:
```
linear_dep.sort_values('abs_pearson_corr', ascending=False, inplace=True)
linear_dep.drop('abs_pearson_corr', axis=1, inplace=True)
```

Reset the index:

In [171]:
```
linear_dep.reset_index(inplace=True)
linear_dep.rename(columns={'index':'variable'}, inplace=True)
```

View the results for the top 20 predictors most correlated with charged_off:

In [172]:
```
1 linear_dep.head(20)
```

Out[172]:

	variable	pearson_corr	F	p_value
0	int_rate	0.191801	1339.524995	7.748636e-288
1	term	0.145296	756.378601	0.000000e+00
2	fico_score	-0.127553	580.066888	3.920684e-127
3	revol_util	0.094082	312.788272	1.084535e-69
4	purpose_small_business	0.075732	202.318024	1.401298e-45
5	log_annual_inc	-0.070739	176.388564	3.724483e-40
6	sub_grade_A4	-0.064685	147.365875	7.633027e-34
7	sub_grade_A3	-0.058037	118.534142	1.466202e-27
8	pub_rec	0.055019	106.490706	6.249204e-25
9	sub_grade_A2	-0.052704	97.694061	5.233169e-23
10	pub_rec_bankruptcies	0.051891	92.813599	6.124905e-22
11	sub_grade_A5	-0.048746	83.539558	6.571150e-20
12	sub_grade_F5	0.047954	80.839569	2.567553e-19
13	sub_grade_E1	0.045968	74.267906	7.100893e-18
14	loan_amnt	0.044438	69.397356	8.338587e-17
15	sub_grade_E2	0.041482	60.455681	7.730389e-15
16	purpose_credit_card	-0.040825	58.552040	2.030445e-14
17	sub_grade_D5	0.040721	58.254669	2.361163e-14
18	dti	0.039275	54.183521	1.866125e-13
19	sub_grade_E4	0.039127	53.775829	2.295702e-13

The variables most linearly correlated with charged_off are the interest rate, loan period (term), FICO score, debt-to-income ratio, income, the loan grade, and the loan amount.

In [173]:
```
1 linear_dep.tail(20)
```

Out[173]:

	variable	pearson_corr	F	p_value
96	addr_state_CT	-0.003795	0.505254	0.477205
97	addr_state_ME	-0.003722	0.485908	0.485763
98	addr_state_WV	-0.003395	0.404253	0.524905
99	addr_state_WI	-0.003318	0.386190	0.534313
100	addr_state_MN	-0.003252	0.370935	0.542499
101	addr_state_NM	0.002875	0.289912	0.590280
102	home_ownership_OWN	0.002764	0.267975	0.604697
103	addr_state_HI	0.002482	0.216139	0.642001
104	addr_state_MS	-0.002294	0.184523	0.667517
105	addr_state_MI	-0.001648	0.095321	0.757520
106	addr_state_NC	0.001385	0.067347	0.795242
107	addr_state_TN	-0.001382	0.067008	0.795746
108	addr_state_AZ	0.001115	0.043573	0.834651
109	addr_state_SC	-0.001112	0.043388	0.834998
110	addr_state_ID	0.001030	0.037226	0.847007
111	purpose_vacation	0.000878	0.027004	0.869473
112	addr_state_KY	0.000653	0.014938	0.902726
113	addr_state_OK	0.000326	0.003716	0.951392
114	addr_state_MT	-0.000317	0.003515	0.952721
115	addr_state_NH	-0.000060	0.000125	0.991075

It looks like the borrower's state of residence, the revolving balance, and several of the loan purposes are irrelevant for predicting charge-off.

Model Training and Testing

We implement machine learning pipelines consisting of one or more of the following steps, depending on the particular model:

1.Mean imputation of missing values

2.Dimension reduction using linear discriminant analysis (LDA)

3.Data standardization: rescaling to zero mean and unit variance

4.The chosen model

We will evaluate and compare the following models using a cross-validated AUROC score on the training set:

1.Logistic regression with SGD training

2.Random forest

3.k-nearest neighbors

We'll perform some hyperparameter tuning for each model to choose the most promising model, then more carefully tune the hyperparameters of the best-performing model.

In [174]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
```

In [175]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
```

In [176]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import GridSearchCV
```

Logistic regression with SGD training

The SGDClassifier estimator in scikit-learn implements linear classifiers (SVM, logistic regression, and others) with stochastic gradient descent (SGD) training. A particular linear classifier is chosen through the loss hyperparameter. Because we want to predict the probability of charge-off, we choose logistic regression (a probabilistic classifier) by setting loss = 'log'.

In [177]:
```
from sklearn.linear_model import SGDClassifier
```

The machine learning pipeline:

In [178]:
```
pipeline_sgdlogreg = Pipeline([
    ('SimpleImputer', SimpleImputer(copy=False)),                    # Mean imputation by default
    ('scaler', StandardScaler(copy=False)),
    ('model', SGDClassifier(loss='log', max_iter=1000, tol=1e-3, random_state=1, warm_start=True))
])
```

A small grid of hyperparameters to search over:

In [179]:
```
param_grid_sgdlogreg = {
    'model__alpha': [10**-5, 10**-2, 10**1],
    'model__penalty': ['l1', 'l2']
}
```

Create the search grid object:

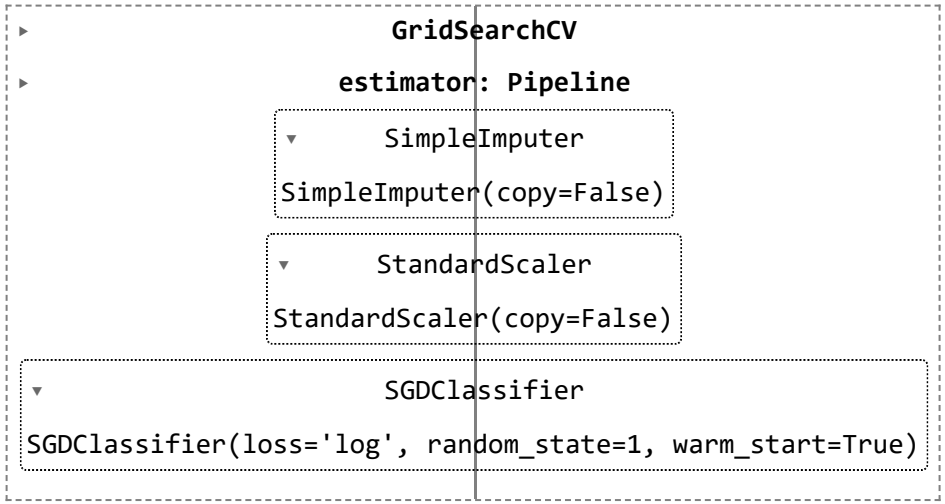
In [180]:
```
grid_sgdlogreg = GridSearchCV(estimator=pipeline_sgdlogreg, param_grid=param_grid_sgdlogreg, scoring='roc_auc', n_jo
```

Conduct the grid search and train the final model on the whole dataset:

In [181]:
```
1 grid_sgdlogreg.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits

C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:173: FutureWarning: The loss 'l
og' was deprecated in v1.1 and will be removed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
```

Out[181]:



In [182]:
```
1 grid_sgdlogreg.best_score_
```

Out[182]:
```
0.6879561568539666
```

Best hyperparameters:

In [183]:
```
1 grid_sgdlogreg.best_params_
```

Out[183]:
```
{'model__alpha': 0.01, 'model__penalty': 'l2'}
```

Random forest classifier

Next we train a random forest model. Note that data standardization is not necessary for a random forest.

In [184]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [185]:
```
pipeline_rfc = Pipeline([
    ('SimpleImputer', SimpleImputer(copy=False)),
    ('model', RandomForestClassifier(n_jobs=-1, random_state=1))
])
```

The random forest takes very long to train, so we don't test different hyperparameter choices. We'll still use GridSearchCV for the sake of consistency.

In [186]:
```
param_grid_rfc = {
    'model__n_estimators': [50] # The number of randomized trees to build
}
```

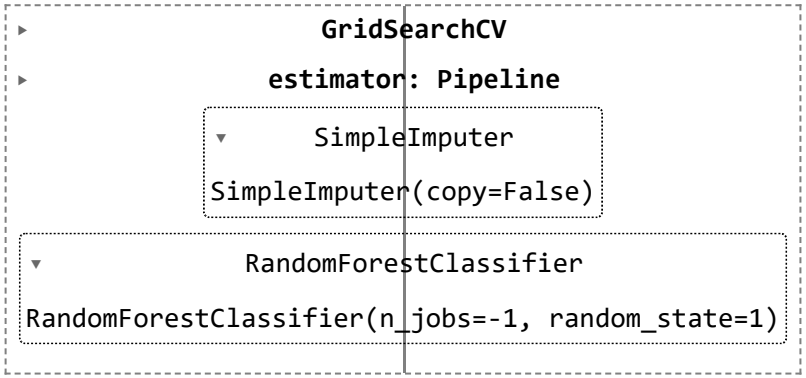
The AUROC will always improve (with decreasing gains) as the number of estimators increases, but it's not necessarily worth the extra training time and model complexity.

In [187]:
```
grid_rfc = GridSearchCV(estimator=pipeline_rfc, param_grid=param_grid_rfc, scoring='roc_auc', n_jobs=-1, pre_dispatc
```

In [188]:
```
grid_rfc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

Out[188]:



Mean cross-validated AUROC score of the random forest:

In [189]:
```
1 grid_rfc.best_score_
```

Out[189]:
```
0.6609818938876553
```

Not quite as good as logistic regression, at least according to this metric.

Tune hyperparameters on the chosen model more finely

In [195]:
```
1 print('Cross-validated AUROC scores')
2 print(grid_sgdlogreg.best_score_, '- Logistic regression')
3 print(grid_rfc.best_score_, '- Random forest')
```

```
Cross-validated AUROC scores
0.6879561568539666 - Logistic regression
0.6609818938876553 - Random forest
```

In [196]:
```
1 param_grid_sgdlogreg = {
2     'model__alpha': np.logspace(-4.5, 0.5, 11), # Fills in the gaps between 10^-5 and 10^1
3     'model__penalty': ['l1', 'l2']
4 }
5
6 print(param_grid_sgdlogreg)
7
```

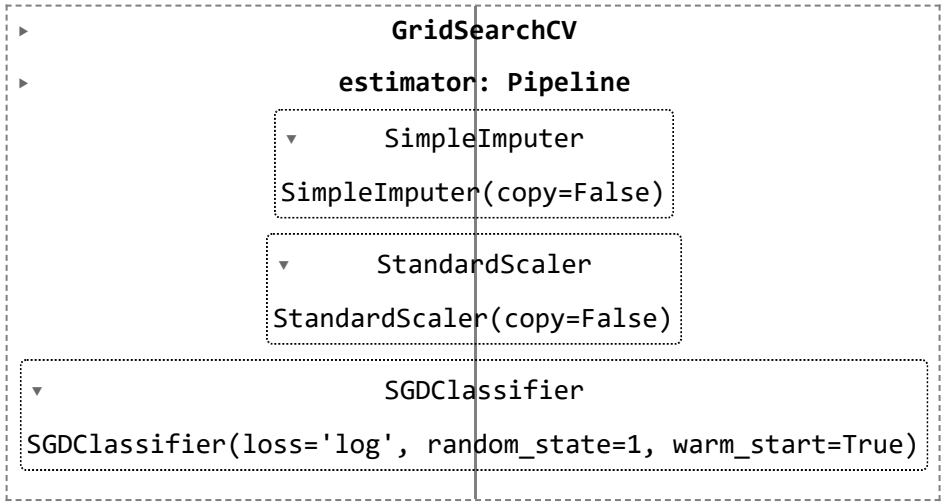
```
{'model__alpha': array([3.16227766e-05, 1.00000000e-04, 3.16227766e-04, 1.00000000e-03,
       3.16227766e-03, 1.00000000e-02, 3.16227766e-02, 1.00000000e-01,
       3.16227766e-01, 1.00000000e+00, 3.16227766e+00]), 'model__penalty': ['l1', 'l2']}
```

In [197]:
```
1 grid_sgdlogreg = GridSearchCV(estimator=pipeline_sgdlogreg, param_grid=param_grid_sgdlogreg, scoring='roc_auc', n_jo
```

In [198]:
```
1 grid_sgdlogreg.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 22 candidates, totalling 110 fits

C:\Users\Admin\anaconda3\lib\site-packages\sklearn\linear_model\_stochastic_gradient.py:173: FutureWarning: The loss 'l
og' was deprecated in v1.1 and will be removed in version 1.3. Use `loss='log_loss'` which is equivalent.
  warnings.warn(
```

Out[198]:



Mean cross-validated AUROC score of the best model:

In [200]:
```
1 grid_sgdlogreg.best_score_
```

Out[200]:
```
0.6897506850630147
```

Best hyperparameters:

In [202]:
```
1 grid_sgdlogreg.best_params_
```

Out[202]:
```
{'model__alpha': 0.03162277660168379, 'model__penalty': 'l2'}
```

Test set evaluation

In [203]:
```
from sklearn.metrics import roc_auc_score
```

In [204]:
```
y_score = grid_sgdlogreg.predict_proba(X_test)[:,1]
roc_auc_score(y_test, y_score)
```

Out[204]:
```
0.7289313212195873
```

The test set AUROC score is somewhat lower than the cross-validated score (0.687).

In [ ]:
```

```