In [38]:

```
#import necessry libraries
import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
import seaborn as sns

#read data.csv into a dataframe df and check basic information
df=pd.read_csv('/Users/danhongcheng/desktop/data.csv')
df.describe()
```

```
/Users/danhongcheng/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3165: DtypeWarning: Colum
ns (18,56) have mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Out[38]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	installment	annual_inc	dt
count	1.999990e+05	0.0	199999.000000	199999.000000	199999.000000	199999.000000	199999.000000	1.999990e+05	199997.00000(
mean	6.229635e+07	NaN	15278.117141	15278.117141	15269.472943	12.361755	441.383672	7.815068e+04	19.16449(
std	3.941126e+06	NaN	8651.138790	8651.138790	8646.324513	4.242108	247.052659	8.051380e+04	9.15759!
min	5.670500e+04	NaN	1000.000000	1000.000000	900.000000	5.320000	14.770000	0.000000e+00	0.00000(
25%	5.941173e+07	NaN	8500.000000	8500.000000	8475.000000	9.170000	261.880000	4.757100e+04	12.54000(
50%	6.221754e+07	NaN	14000.000000	14000.000000	14000.000000	12.290000	383.810000	6.500000e+04	18.61000(
75%	6.564457e+07	NaN	20000.000000	20000.000000	20000.000000	14.650000	580.730000	9.340000e+04	25.41000(
max	6.861706e+07	NaN	35000.000000	35000.000000	35000.000000	28.990000	1445.460000	9.000000e+06	999.00000(

8 rows × 115 columns

In [39]:

```
df.shape
```

Out[39]:

```
(199999, 148)
```

In [40]:

```
df.sample(10)
```

Out[40]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	...	hardship_payoff_l
611	68606692	NaN	15000	15000	15000.0	36 months	12.88	504.55	C	C2	...	
118631	61480137	NaN	28000	28000	28000.0	36 months	7.26	867.90	A	A4	...	
145656	60185273	NaN	8000	8000	8000.0	36 months	10.99	261.88	B	B4	...	
75555	64097871	NaN	35000	35000	35000.0	60 months	16.99	869.66	D	D3	...	
16530	67485341	NaN	27600	27600	27600.0	60 months	10.64	595.15	B	B4	...	
184121	57135502	NaN	10000	10000	10000.0	36 months	12.69	335.45	C	C2	...	
76528	63179137	NaN	18000	18000	18000.0	36 months	10.99	589.22	B	B4	...	
121651	60785001	NaN	4000	4000	4000.0	36 months	7.89	125.15	A	A5	...	
182855	57296281	NaN	28000	28000	28000.0	60 months	9.99	594.78	B	B3	...	
148176	59709156	NaN	35000	35000	35000.0	36 months	12.69	1174.07	C	C2	...	

10 rows × 148 columns

In [41]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 199999 entries, 0 to 199998
Columns: 148 entries, id to settlement_term
dtypes: float64(70), int64(45), object(33)
memory usage: 225.8+ MB
```

In [42]:

```
df['loan_status'].value_counts(dropna=False)
```

Out[42]:

```
Fully Paid            140991
```

```
Charged Off             35090
Current                 22637
Late (31-120 days)        785
In Grace Period           347
Late (16-30 days)         148
Default                     1
Name: loan_status, dtype: int64
```

In [43]:

```python
# slice the dataframe so as to only keep records that have 'loan_status' in ['Fully Paid', 'Charged off']
df=df[df['loan_status'].isin(['Fully Paid', 'Charged Off'])]
print(df['loan_status'].value_counts())
df.shape
```

```
Fully Paid     140991
Charged Off     35090
Name: loan_status, dtype: int64
```

Out[43]:

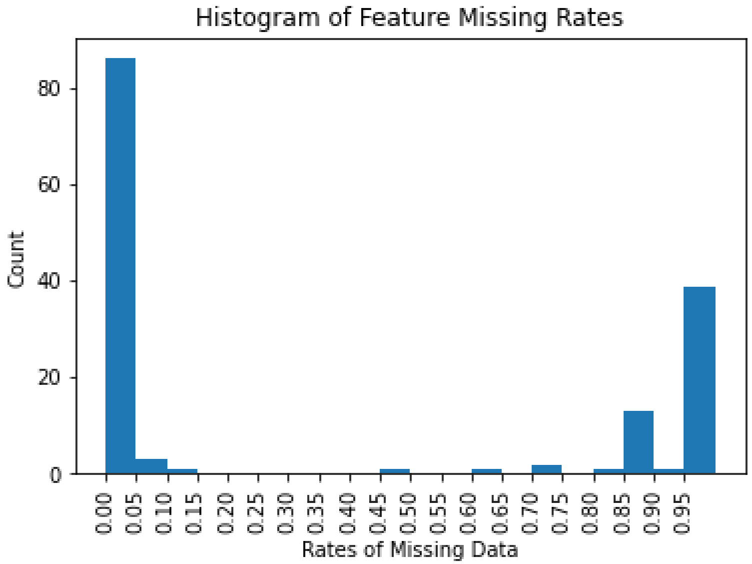
```
(176081, 148)
```

In [44]:

```python
#cacluate missing rates of each feature and plot histogram of feature missing rates
missing_rate=df.isnull().mean().sort_values(ascending=False)
plt.hist(x=missing_rate, bins=20)
plt.xticks(np.arange(0,1,0.05), rotation='vertical')
plt.title('Histogram of Feature Missing Rates')
plt.xlabel('Rates of Missing Data')
plt.ylabel('Count')
```

Out[44]:

```
Text(0, 0.5, 'Count')
```



In [45]:

```
#by visualizing the missing rate of features, we can set the threshold of missing rate to 0.2 to drop features
drop_feature=sorted(list(missing_rate[missing_rate.values>0.05].index))
print(drop_feature)
```

```
['all_util', 'annual_inc_joint', 'debt_settlement_flag_date', 'deferral_term', 'desc', 'dti_joint', 'emp_length', 'emp_t
itle', 'hardship_amount', 'hardship_dpd', 'hardship_end_date', 'hardship_last_payment_amount', 'hardship_length', 'hards
hip_loan_status', 'hardship_payoff_balance_amount', 'hardship_reason', 'hardship_start_date', 'hardship_status', 'hardsh
ip_type', 'il_util', 'inq_fi', 'inq_last_12m', 'max_bal_bc', 'member_id', 'mths_since_last_delinq', 'mths_since_last_maj
or_derog', 'mths_since_last_record', 'mths_since_rcnt_il', 'mths_since_recent_bc_dlq', 'mths_since_recent_inq', 'mths_si
nce_recent_revol_delinq', 'next_pymnt_d', 'num_tl_120dpd_2m', 'open_acc_6m', 'open_act_il', 'open_il_12m', 'open_il_24
m', 'open_rv_12m', 'open_rv_24m', 'orig_projected_additional_accrued_interest', 'payment_plan_start_date', 'revol_bal_jo
int', 'sec_app_chargeoff_within_12_mths', 'sec_app_collections_12_mths_ex_med', 'sec_app_earliest_cr_line', 'sec_app_fic
o_range_high', 'sec_app_fico_range_low', 'sec_app_inq_last_6mths', 'sec_app_mort_acc', 'sec_app_mths_since_last_major_de
rog', 'sec_app_num_rev_accts', 'sec_app_open_acc', 'sec_app_open_act_il', 'sec_app_revol_util', 'settlement_amount', 'se
ttlement_date', 'settlement_percentage', 'settlement_status', 'settlement_term', 'total_bal_il', 'total_cu_tl', 'verific
ation_status_joint']
```

In [46]:

```
# Drop drop_feature from dataframe and check remaining features
df.drop(labels=drop_feature, axis=1, inplace=True)
df.shape # check dataframe number of rows and columns after drop some features
```

Out[46]: (176081, 86)

In [47]:
```
print(df.columns)
```

```
Index(['id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate',
       'installment', 'grade', 'sub_grade', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'pymnt_plan',
       'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line',
       'fico_range_low', 'fico_range_high', 'inq_last_6mths', 'open_acc',
       'pub_rec', 'revol_bal', 'revol_util', 'total_acc',
       'initial_list_status', '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', 'application_type',
       'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim',
       'acc_open_past_24mths', 'avg_cur_bal', 'bc_open_to_buy', 'bc_util',
       'chargeoff_within_12_mths', 'delinq_amnt', '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_recent_bc', '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_30dpd', 'num_tl_90g_dpd_24m', 'num_tl_op_past_12m',
       'pct_tl_nvr_dlq', 'percent_bc_gt_75', 'pub_rec_bankruptcies',
       'tax_liens', 'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit', 'hardship_flag', 'disbursement_method',
       'debt_settlement_flag'],
      dtype='object')
```

In [48]:
```
# further drop some features such as'id' at Author's judgement that they're not as helpful to modeling
drop_list=['id', 'addr_state', 'funded_amnt', 'funded_amnt_inv', 'pymnt_plan', 'delinq_2yrs', 'inq_last_6mths', 'out_pr
df.drop(labels=drop_list, axis=1, inplace=True)
```

In [49]:
```
# check number of rows and columns in dataframe after drop
df.shape
```

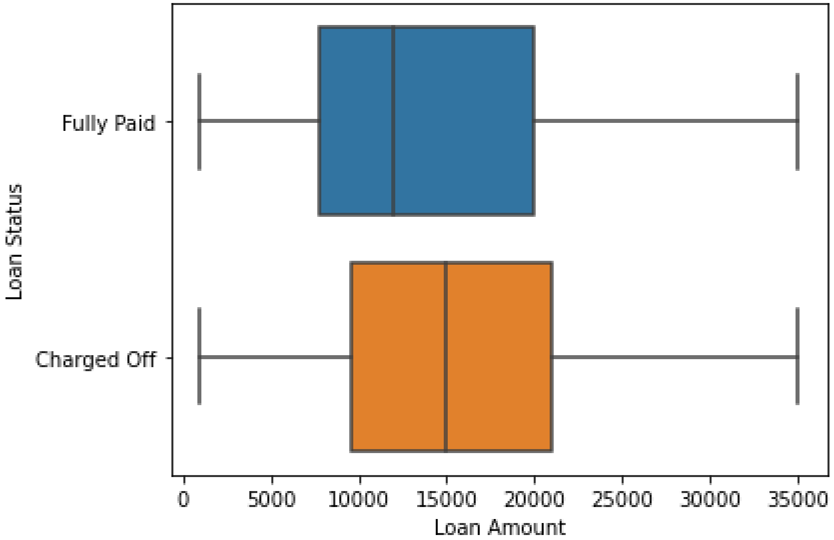
Out[49]:
```
(176081, 26)
```

In [50]:
```
# define a plot function to plot features
def plot_feature (col_name, full_name, continuous):
    f, ax1=plt.subplots(1,1) # create figures
    if continuous:
        ax1=sns.boxplot(data=df, x=col_name, y='loan_status')
```

```
        ax1.set_xlabel(full_name)
        ax1.set_ylabel('Loan Status') # if the feature is continuous, create boxplot and set x, y labels
    else:
        charge_off_rates=df.groupby(col_name)['loan_status'].value_counts(normalize=True).loc[:, 'Charged Off']
        ax1=sns.barplot(x=charge_off_rates.index, y=charge_off_rates.values)
        ax1.set_xlabel(full_name)
        ax1.set_ylabel('Charge_off_rates') # if the feature is categorical, calcuate charge_off_rates grouped by featur
    plt.tight_layout()
```

In [51]:

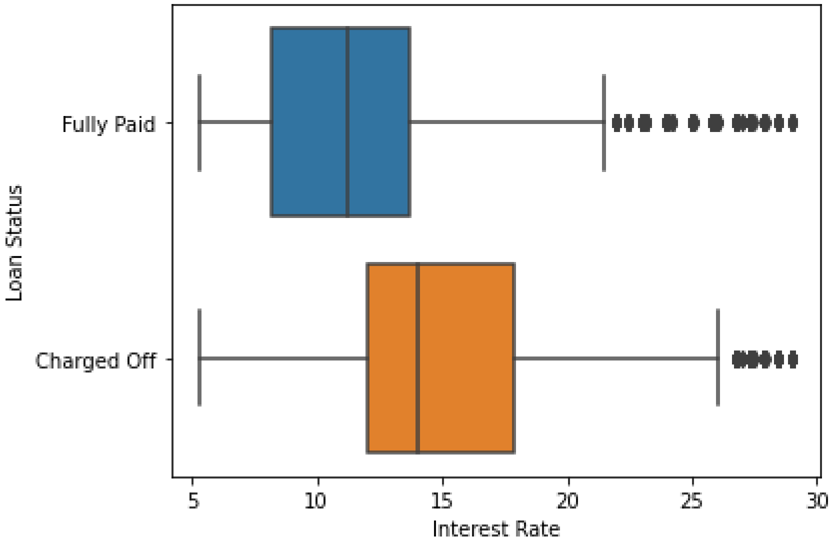
```
# choose some features to plot and explore based on business judgement
plot_feature('loan_amnt', 'Loan Amount', continuous=True)
```



In [52]:

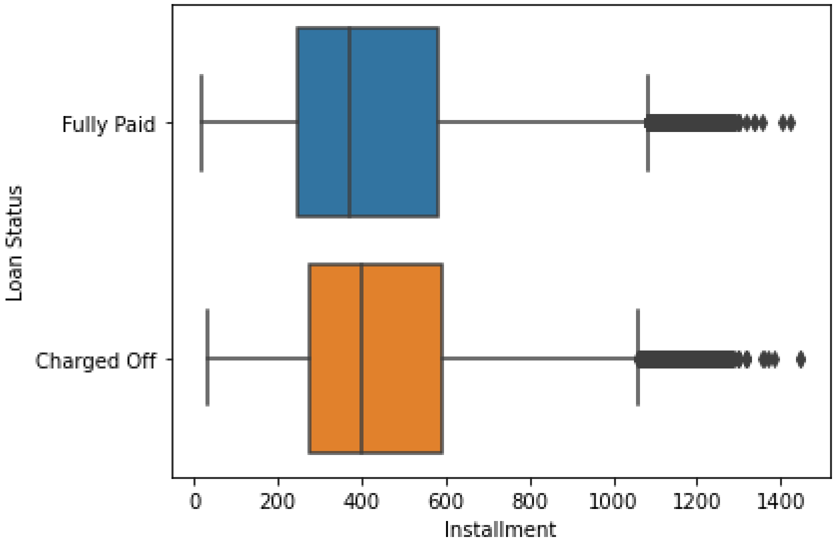
```
# As boxplot shown above, the fully paid loan amount is lower than the charged off loan amount

plot_feature('int_rate', 'Interest Rate', continuous=True)
```



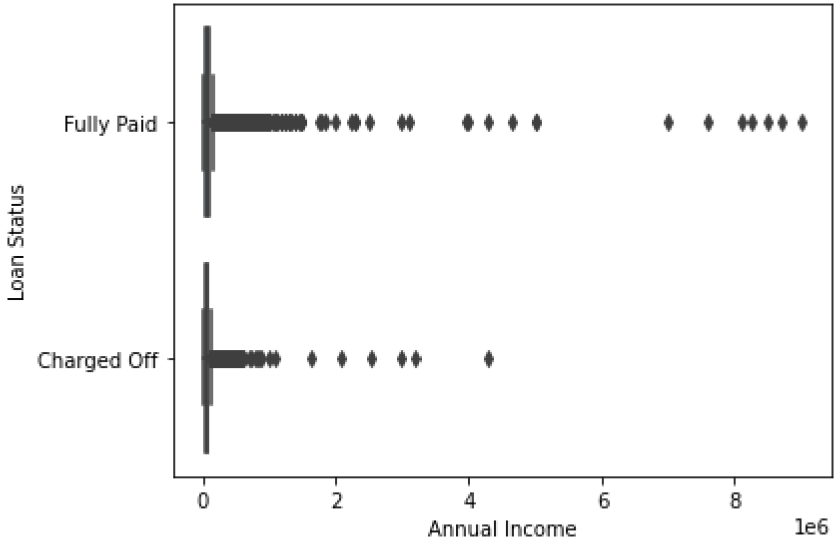
In [53]:
```
# As shown above, the interquartile interest rate of Charged Off loans ranges fraom 12% to 18% while interquartile of i
#It is safe to say that Charged Off loans tend to have higher interest rate than Fully Paid loans
```

In [54]:
```
plot_feature('installment', 'Installment', continuous=True)
```



In [55]:
```
# We can see Charged Off loans have slightly higher installments
```

In [56]:
```
plot_feature('annual_inc', 'Annual Income', continuous=True)
```

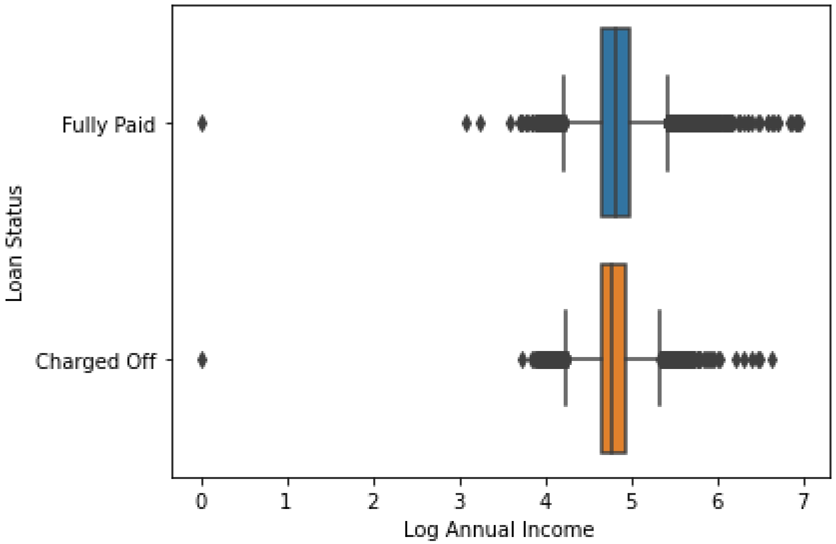


In [57]:
```
# Modify annual income to log values for better visualization
df['log_annual_inc']=df['annual_inc'].apply(lambda x:np.log10(x+1))
df['log_annual_inc'].describe()
```

Out[57]:
```
count    176081.000000
mean          4.818483
std           0.236587
min           0.000000
25%           4.662767
50%           4.812920
75%           4.963793
max           6.954243
Name: log_annual_inc, dtype: float64
```

In [58]:
```
# drop annual income column since we have log annual income now
df.drop(labels='annual_inc', axis=1, inplace=True)
```

In [59]:
```
plot_feature('log_annual_inc', 'Log Annual Income', continuous=True)
```



In [60]:
```
# As we can see, Fully Paid Loan borrowers have higher annual income than Charged Off loan borrowers
```

In [61]:
```
#check grade and sub_grade
print(sorted(df['grade'].unique()))
```

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

In [62]:
```
print(sorted(df['sub_grade'].unique()))
```

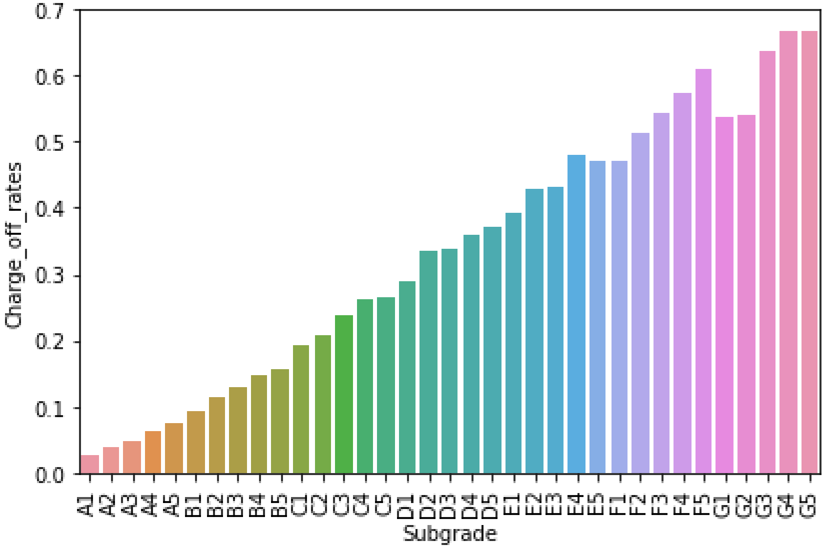
```
['A1', 'A2', 'A3', 'A4', 'A5', 'B1', 'B2', 'B3', 'B4', 'B5', 'C1', 'C2', 'C3', 'C4', 'C5', 'D1', 'D2', 'D3', 'D4', 'D5',
'E1', 'E2', 'E3', 'E4', 'E5', 'F1', 'F2', 'F3', 'F4', 'F5', 'G1', 'G2', 'G3', 'G4', 'G5']
```

In [63]:
```
# As we can see, grade is included in the sub_grade, so we can drop grade
df.drop(labels='grade', axis=1, inplace=True)
```

In [64]:
```
plot_feature('sub_grade', 'Subgrade', continuous=False)
plt.xticks(rotation='vertical')
```

Out[64]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
```

```
        34]),
 [Text(0, 0, 'A1'),
  Text(1, 0, 'A2'),
  Text(2, 0, 'A3'),
  Text(3, 0, 'A4'),
  Text(4, 0, 'A5'),
  Text(5, 0, 'B1'),
  Text(6, 0, 'B2'),
  Text(7, 0, 'B3'),
  Text(8, 0, 'B4'),
  Text(9, 0, 'B5'),
  Text(10, 0, 'C1'),
  Text(11, 0, 'C2'),
  Text(12, 0, 'C3'),
  Text(13, 0, 'C4'),
  Text(14, 0, 'C5'),
  Text(15, 0, 'D1'),
  Text(16, 0, 'D2'),
  Text(17, 0, 'D3'),
  Text(18, 0, 'D4'),
  Text(19, 0, 'D5'),
  Text(20, 0, 'E1'),
  Text(21, 0, 'E2'),
  Text(22, 0, 'E3'),
  Text(23, 0, 'E4'),
  Text(24, 0, 'E5'),
  Text(25, 0, 'F1'),
  Text(26, 0, 'F2'),
  Text(27, 0, 'F3'),
  Text(28, 0, 'F4'),
  Text(29, 0, 'F5'),
  Text(30, 0, 'G1'),
  Text(31, 0, 'G2'),
  Text(32, 0, 'G3'),
  Text(33, 0, 'G4'),
  Text(34, 0, 'G5')])
```

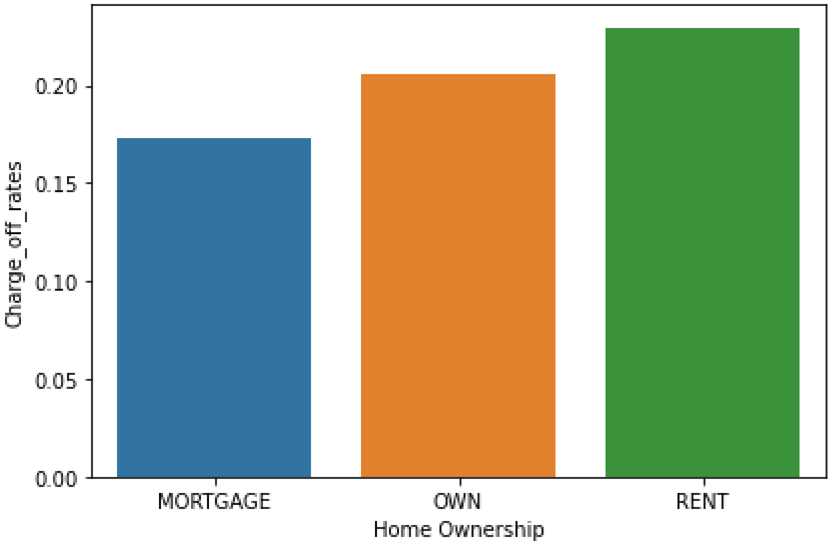


In [65]:
```
# It is very clear that as sub_grade goes worse, Charge Off rates goes higher
```

In [66]:
```
# check feature home_ownership
df['home_ownership'].value_counts(dropna=False)
```

Out[66]:
```
MORTGAGE    86296
RENT        69952
OWN         19832
ANY             1
Name: home_ownership, dtype: int64
```

In [67]:
```
plot_feature('home_ownership', 'Home Ownership', continuous=False)
```

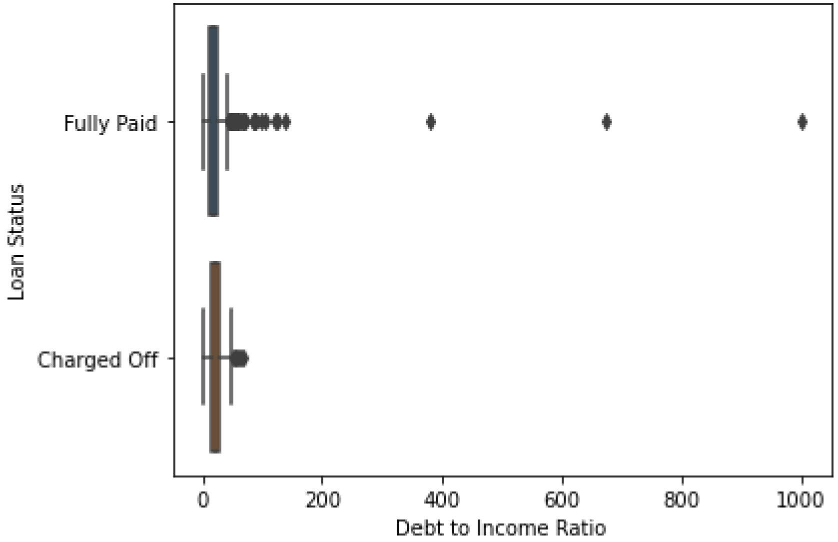


In [68]:
```
# As we can see, mortgaged home owners have lowest loan charge_off_rates, renters have the highest charge off rates. In
```

In [69]:
```
# check dti, debt to income ratio
df['dti'].describe()
```

Out[69]:
```
count    176079.000000
mean         18.911104
std           9.180844
min           0.000000
25%          12.310000
50%          18.340000
75%          25.090000
max         999.000000
Name: dti, dtype: float64
```

In [70]:
```
plot_feature('dti', 'Debt to Income Ratio', continuous=True)
```



In [71]:
```
# Looks like there are some outliers. Remove records that have dti larger than 200
```

In [72]:
```
# count how mnay outliers (dti>200)
(df['dti']>=40).sum()
```

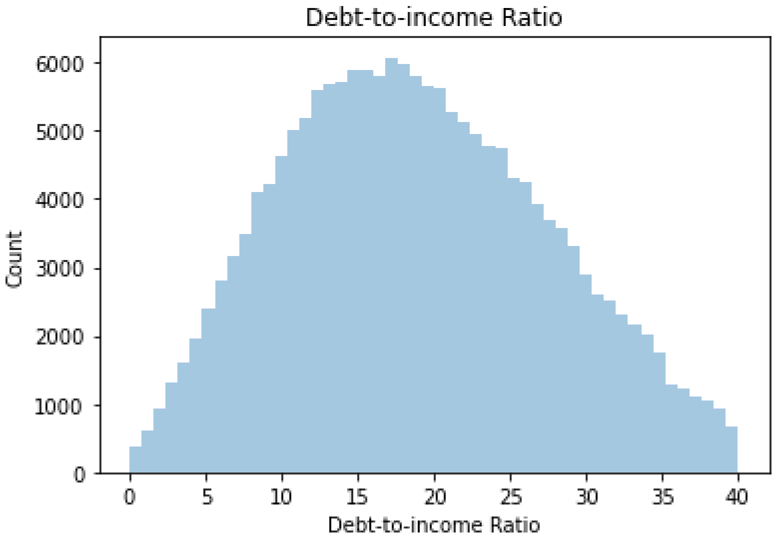
Out[72]: 69

In [73]:
```
# only 69 compare to total data point of 176079, 69 is very few, we can plot without these outliers
```

In [74]:
```
plt.figure()
sns.distplot(df.loc[df['dti'].notnull()&(df['dti']<40), 'dti'], kde=False)
plt.xlabel('Debt-to-income Ratio')
plt.ylabel('Count')
plt.title('Debt-to-income Ratio')
```

```
/Users/danhongcheng/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` i
s a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figur
e-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

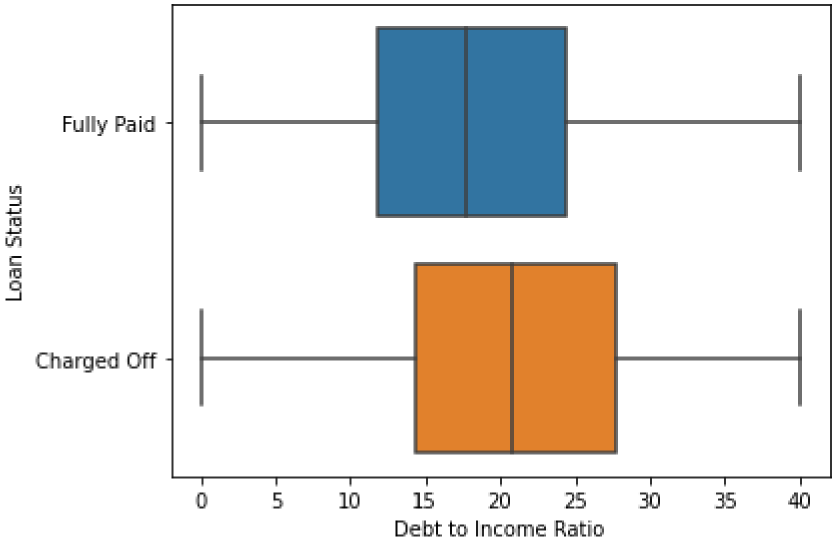
Out[74]: Text(0.5, 1.0, 'Debt-to-income Ratio')



In [75]:
```
# drop rows that have dti higher than 40
df=df[df.dti<=40]
df['dti'].describe()
```

Out[75]:
```
count    176010.00000
mean         18.88532
std           8.65833
min           0.00000
25%          12.31000
50%          18.34000
75%          25.08000
max          39.99000
Name: dti, dtype: float64
```

In [76]:
```
plot_feature('dti', 'Debt to Income Ratio', continuous=True)
```



In [77]:
```
# As we can see, Charged Off loan borrowers have higher debt to incomre rato than Fully Paid loan borrowers
```

In [78]:
```
#Check Fico score
df[['fico_range_low', 'fico_range_high']].describe()
```

Out[78]:

	fico_range_low	fico_range_high
count	176010.000000	176010.000000
mean	694.533066	698.533197
std	31.063531	31.064167
min	660.000000	664.000000
25%	670.000000	674.000000
50%	685.000000	689.000000
75%	710.000000	714.000000
max	845.000000	850.000000

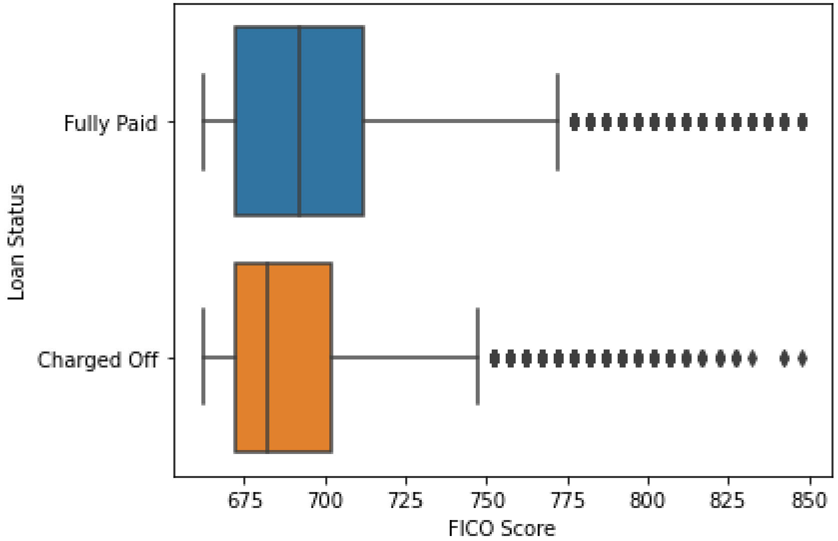
In [79]:
```
# take average of fico_range_low and fico_range_high to get a fico_score
```

```
df['fico_score']=0.5*(df['fico_range_low']+df['fico_range_high'])
df['fico_score'].describe()
```

Out[79]:
```
count    176010.000000
mean        696.533132
std          31.063848
min         662.000000
25%         672.000000
50%         687.000000
75%         712.000000
max         847.500000
Name: fico_score, dtype: float64
```

In [80]:
```
#drop fico_range_high and fico_range_low since we have fico_score
df.drop(labels=['fico_range_high', 'fico_range_low'], axis=1, inplace=True)
```

In [81]:
```
# plot fico_score
plot_feature('fico_score', 'FICO Score', continuous=True)
```



In [82]:
```
# We can see Fully Paid loan borrowers have higher fico scores than Charged Off loan borrowers
```

In [83]:
```
# Now as we explored some features and they all make sense when we plot them. That indicates this dataset is so far so
```

```
# First we need to change loan status to 0/1
df['charged_off']=(df['loan_status']=='Charged Off').apply(np.uint8)
print(df['charged_off'].value_counts())
```

```
0    140936
1     35074
Name: charged_off, dtype: int64
```

In [84]:

```
# drop original loan_status
df.drop(labels='loan_status', axis=1, inplace=True)
```

In [85]:

```
# check how many rows and features do we have
df.shape
```

Out[85]:

```
(176010, 24)
```

In [86]:

```
# Find columns that have categorical variables
num_cols=df._get_numeric_data().columns # find numerical columns
cols=df.columns # get all columns
cat_cols=list(set(cols)-set(num_cols)) # create a list that contain categorical columns
df=pd.get_dummies(df, columns=cat_cols, drop_first=True)
df.shape
```

Out[86]:

```
(176010, 1590)
```

In [87]:

```
print(df.sample(5))
```

```
        loan_amnt  int_rate  installment    dti  open_acc  pub_rec  revol_bal  \
173258       1000     12.69        33.55  11.10         3        0       3469
87731        3500      9.17       111.58   0.88        11        1       1856
136648      16000     11.53       527.85  20.21        11        0      21072
114965      35000     18.25      1269.73  27.20        12        0      24774
65415        5000      6.89       154.14  29.04        14        0      16197

        revol_util  total_acc  mort_acc  ...  sub_grade_F2  sub_grade_F3  \
173258        82.6          9         0  ...             0             0
87731         12.3         31         4  ...             0             0
136648        39.5         39         0  ...             0             0
114965        87.9         45         4  ...             0             0
65415         77.5         19         0  ...             0             0
```

```
        sub_grade_F4  sub_grade_F5  sub_grade_G1  sub_grade_G2  sub_grade_G3  \
173258             0             0             0             0             0
87731              0             0             0             0             0
136648             0             0             0             0             0
114965             0             0             0             0             0
65415              0             0             0             0             0

        sub_grade_G4  sub_grade_G5  term_ 60 months
173258             0             0                0
87731              0             0                0
136648             0             0                0
114965             0             0                0
65415              0             0                0

[5 rows x 1590 columns]
```

In [110…

```
## Check total number of null values in dataframe
null_row_total=np.sum(df.isnull().sum())
print(null_row_total)
```

```
1683
```

In [111…

```
## Check the total number of inifinity values in dataframe
infinity_row_total=np.sum(np.isinf(df).values.sum())
print(infinity_row_total)
```

```
0
```

In [112…

```
## Check total entries in the dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 176010 entries, 0 to 199998
Columns: 1590 entries, loan_amnt to term_ 60 months
dtypes: float64(7), int64(8), uint8(1575)
memory usage: 289.9 MB
```

In [ ]:

```
## As we can see, 1683 null values is small compared to total entries of 176010. We will drop these null values.
```

In [116…

```
df.dropna(axis=0, inplace=True)
```

In [117…

```
## Double check if there is still null value
print(sum(df.isnull().sum()))
```

0

In [118…

```
## Now we are ready to build machine learning models. We will use the following three machine learning methods:
## 1. Random forest
## 2. K-nearest neighbors
## 3. Logistic regression
## 4. Neural Network
## And then we will compare their performances.

# Create responsive variable y and predictors X
y=df['charged_off']
X=df.loc[:,df.columns!='charged_off']

# import train_test_split from sklearn
from sklearn.model_selection import train_test_split
```

In [119…

```
# split the data into 80% for training and 20% for test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, shuffle=True)
```

In [333…

```
# 1. Random Forest Classifier
# Import necessary modules for RandomForest Classifier
```

In [120…

```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators=100) # build randomforestclassifier clf
clf.fit(X_train, y_train) # fit training data to the classifier
y_pred=clf.predict(X_test) # predict charged_off using test data
```

In [141…

```
# Evalute the performance of RandomForest Classifier

from sklearn import metrics # import metrics module
from sklearn.metrics import roc_auc_score # import roc_auc_score

# Caculate accuracy, precision, recall, f1 score and roc_auc_score of the metrics
clf_accuracy=metrics.accuracy_score(y_test, y_pred)
```

```
clf_precision=metrics.precision_score(y_test, y_pred, average='weighted')
clf_recall=metrics.recall_score(y_test, y_pred, average='weighted')
clf_f1=metrics.f1_score(y_test, y_pred, average='weighted')
clf_roc_auc=roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])
```

In [181…

```
# 2. k-nearest neighbors
# import modules for k-nearest neighbors
from sklearn import neighbors
# build knn model
knn=neighbors.KNeighborsClassifier(n_neighbors=5)
# train knn model on training data
knn.fit(X_train, y_train)
```

Out[181… KNeighborsClassifier()

In [154…

```
# predict charged_off using knn model and test data
y_pred_knn=knn.predict(X_test)
```

In [150…

```
# evaluate performance of knn model by calculating accuracy, precision, recall, f1_score, roc_auc_score
knn_accuracy=metrics.accuracy_score(y_test, y_pred_knn)
knn_precision=metrics.precision_score(y_test, y_pred_knn, average='weighted')
knn_recall=metrics.recall_score(y_test, y_pred_knn, average='weighted')
knn_f1=metrics.recall_score(y_test, y_pred_knn, average='weighted')
knn_roc_auc=roc_auc_score(y_test, knn.predict_proba(X_test)[:,1])
```

In [161…

```
# 3. Logistic regression
# Import modules for logistic regression from sklearn
from sklearn import linear_model
# build a logistic regression classifier lrg and train on training data
lrg=linear_model.LogisticRegression(random_state=42, n_jobs=-1).fit(X_train, y_train)
y_pred_lrg=lrg.predict(X_test)
```

In [163…

```
# evaluate performance of logistic regression model by calculating accuracy, prediction, recall, f1_score, roc_auc_scor
lrg_accuracy=metrics.accuracy_score(y_test, y_pred_lrg)
lrg_precision=metrics.precision_score(y_test, y_pred_lrg, average='weighted')
lrg_recall=metrics.recall_score(y_test, y_pred_lrg, average='weighted')
```

```
lrg_f1=metrics.recall_score(y_test, y_pred_lrg, average='weighted')
lrg_roc_auc=roc_auc_score(y_test, lrg.predict_proba(X_test)[:,1])
```

In [174…

```
# 4. Neural network
# import modules for neural network
from sklearn.neural_network import MLPClassifier
# import standardscaler to scale data
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
# scale train data
scaler.fit(X_train)
X_train=scaler.transform(X_train)
# apply same transformation to test data
X_test=scaler.transform(X_test)
# build neural network multi-layer classifier mlp
mlp=MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(100, ), max_iter=1000, random_state=1)
# train the model on training data
mlp.fit(X_train, y_train)
# predict charge_off
y_pred_mlp=mlp.predict(X_test)
```

In [175…

```
# evaluate performance of neural network model by calculating accuracy, prediction, recall, f1_score, roc_auc_score
mlp_accuracy=metrics.accuracy_score(y_test, y_pred_mlp)
mlp_precision=metrics.precision_score(y_test, y_pred_mlp, average='weighted')
mlp_recall=metrics.recall_score(y_test, y_pred_mlp, average='weighted')
mlp_f1=metrics.recall_score(y_test, y_pred_mlp, average='weighted')
mlp_roc_auc=roc_auc_score(y_test, mlp.predict_proba(X_test)[:,1])
```

In [ ]:

```
# build a dictionary to store models and their performance scores
```

In [201…

```
Score_list={'Accuracy':[clf_accuracy, knn_accuracy, lrg_accuracy, mlp_accuracy], 'Precision':[clf_precision, knn_precis…
```

In [204…

```
# convert dictionary into dataframe
Score=pd.DataFrame(Score_list, index=['Random Forest', 'K-Nearest Neighbors', 'Logistic Regression', 'Multi-layer Neura…
```

In [205…

```
print(Score)
```

	Accuracy	Precision	Recall	F1	ROC_AUC
Random Forest	0.807810	0.769618	0.807810	0.748266	0.726702
K-Nearest Neighbors	0.771769	0.302817	0.771769	0.771769	0.726702
Logistic Regression	0.802191	0.742376	0.802191	0.802191	0.694921
Multi-layer Neural Network	0.720762	0.724500	0.720762	0.720762	0.609645

In [ ]:

```
## Conclusions: 1. Random Forest has the highest accuracy at 0.807, it also has good precision, recall, F1 and ROC_AUC
## I would recommend random forest to classify. It can make correct predictions at the probability of 0.807 (accuracy),
```