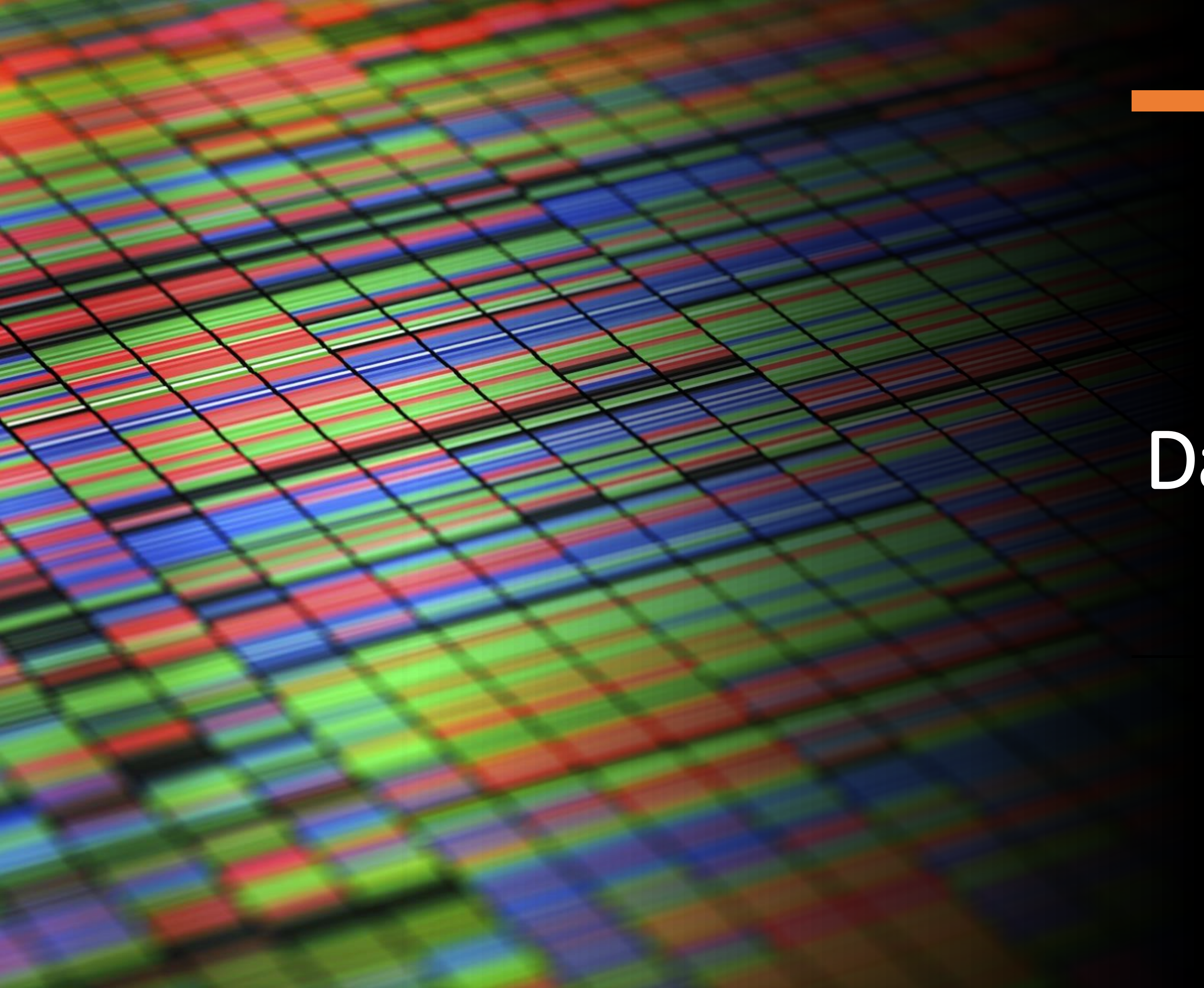
Lending Club Case Study

Submitted
by
Harish Kulkarni

Understanding, Processing & Analyzing Data

1. Understanding the data:
    a. Inspect all available columns
    b. Understand the domain specific meaning for the columns, refer to the dictionary provided for better understanding
    c. Check the key business requirement and try to understand the possible relation between the columns
2. Deleting data with no analytical value
    a. Bulk delete null columns and rows
    b. Delete columns which have same value across all rows and is not of analytical value
    c. Delete rows which are null on all columns
    d. Delete columns which are having most of the values in its row same or have most of the values empty and are of no analytical value
    e. Remove any data which are considered as outliers and are of no analytical value
3. Processing data analysis
    a. Remove unwanted string and bring data to a form acceptable to analytics utilities like – plots, correlation functions etc.
    b. Covert data to numeric and summary forms as needed
    c. Create any grouping needed for analysis

Data provided for analysis is to find out the indicator of a possible defaulters for the bank loan taken. There are various fields related to a bank loan application process, of which some of them could be a the key indicators of someone failing to repay the loan taken.



Data Processing

Processing columns with just NULL data.

1. There are 111 columns of data in the provided Dataset
2. Out of which 54 columns are having null values in every row of the columns
3. All these fields dropped from the dataframe using the command

```
loan.dropna(how="all", axis=1)
```

Here are the list of all fields in the data set:

[39717 rows x 111 columns]
****
['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'pymnt_plan', 'url', 'desc', 'purpose', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq', 'mths_since_last_record', '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', 'next_pymnt_d', 'last_credit_pull_d', 'collections_12_mths_ex_med', 'mths_since_last_major_derog', 'policy_code', 'application_type', 'annual_inc_joint', 'dti_joint', 'verification_status_joint', 'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'open_acc_6m', 'open_il_6m', 'open_il_12m', 'open_il_24m', 'mths_since_rcnt_il', 'total_bal_il', 'il_util', 'open_rv_12m', 'open_rv_24m', 'max_bal_bc', 'all_util', 'total_rev_hi_lim', 'inq_fi', 'total_cu_tl', 'inq_last_12m', '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', 'mths_since_recent_bc_dlq', 'mths_since_recent_inq', 'mths_since_recent_revol_delinq', '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', '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']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Columns: 111 entries, id to total_il_high_credit_limit
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB

And after removing the null fields the null fields here is the list:

[39717 rows x 57 columns]
****
['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'pymnt_plan', 'url', 'desc', 'purpose', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq', 'mths_since_last_record', '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', 'next_pymnt_d', 'last_credit_pull_d', 'collections_12_mths_ex_med', 'policy_code', 'application_type', 'acc_now_delinq', 'chargeoff_within_12_mths', 'delinq_amnt', 'pub_rec_bankruptcies', 'tax_liens']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 57 columns):
dtypes: float64(20), int64(13), object(24)
memory usage: 17.3+ MB

There are fields which have same value in all row, which may not contribute much to the analysis, removing these fields

```
loan_processed=loan_processed[[i for i in loan_processed.columns if(len(loan_processed.loc[:,i].unique())!=1)]]
```

Following fields are left after this processing

```
[39717 rows x 51 columns]
*************************
['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'url', 'desc', 'purpose', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq', 'mths_since_last_record', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', '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', 'next_pymnt_d', 'last_credit_pull_d', 'collections_12_mths_ex_med', 'chargeoff_within_12_mths', 'pub_rec_bankruptcies', 'tax_liens']
*************************
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 51 columns):
```

Removing fields which may have all values same across all rows but most of the values are same: columns which have same quantils at 25,50 and 75

```
def checkSameQuantile(t,i):
    if t.loc[:,i].dtypes==np.object:
        return(0)
    if (t.loc[:,i].quantile(0.25)==t.loc[:,i].quantile(0.50)==t.loc[:,i].quantile(0.75)):
        return(1)
    return(0)
loan_processed=loan_processed[[i for i in loan_processed.columns if(checkSameQuantile(loan_processed,i)!=1)]]
```

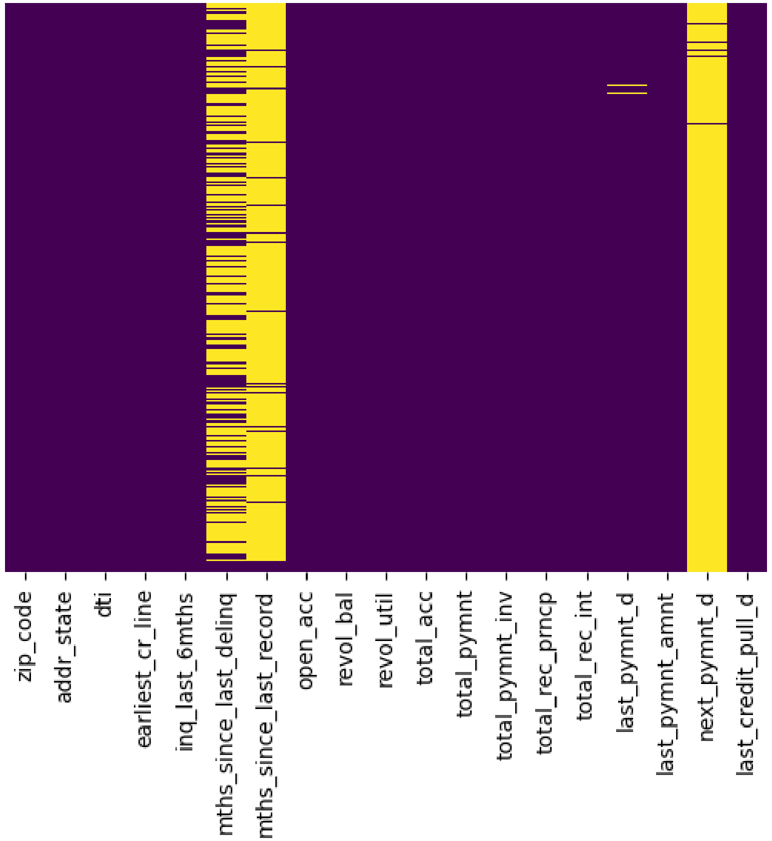
[39717 rows x 40 columns]

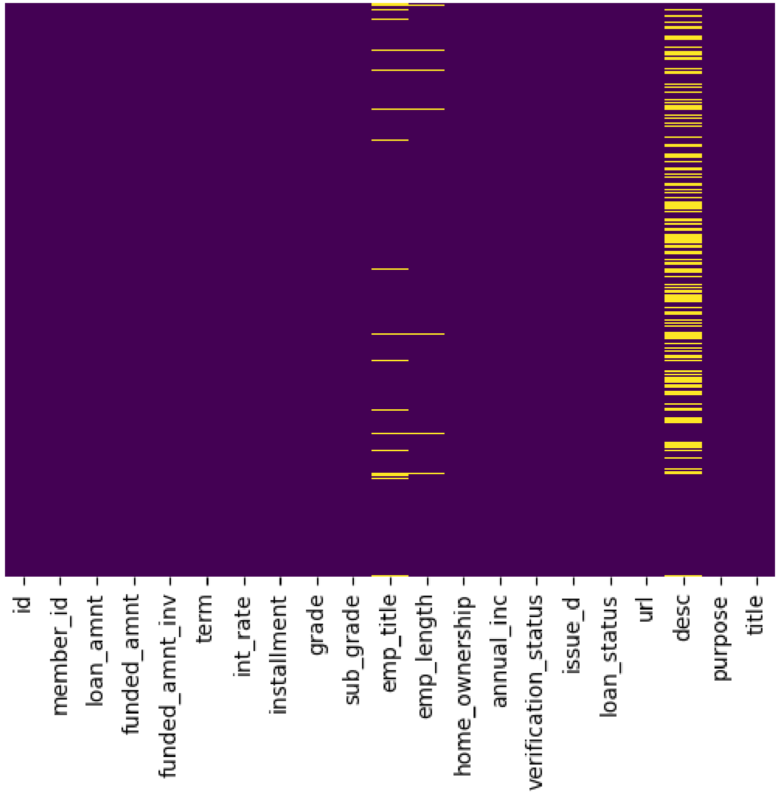
****

['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'emp_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'url', 'desc', 'purpose', 'title', 'zip_code', 'addr_state', 'dti', 'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq', 'mths_since_last_record', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'last_pymnt_d', 'last_pymnt_amnt', 'next_pymnt_d', 'last_credit_pull_d']

Checking for the fields which are still left with null values out after bulk processing. This is done using visualizing the data for null values

```
sns.heatmap(loan_processed.iloc[:, 0:20].isnull(),yticklabels=False,cbar=False, cmap='viridis' )
sns.heatmap(loan_processed.iloc[:, 21:].isnull(),yticklabels=False,cbar=False, cmap='viridis' )
```





As we can see there are few columns which are having mostly null values and don't seems to be of much value for analysis, hence removing them from the dataset

```
loan_processed=loan_processed.drop(['next_pymnt_d', 'mths_since_last_record'], axis=1)
```

Columns left are as follows:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (**total 38 columns**):

Next dropping fields which do not have any impact on the analysis and removing those. These fields don’t seems to have much correlational value for loan default

```
loan_processed=loan_processed.drop(['id','member_id','desc','emp_title','sub_grade' ,'url'
,'issue_d','title','zip_code','addr_state','earliest_cr_line','last_pymnt_d','last_pymnt_amnt','last_credit_pull_d'], axis=1)
```

Columns left after this step are as follows:

39717 rows × 24 columns

['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'loan_status', 'purpose', 'dti', 'inq_last_6mths', 'mths_since_last_delinq', 'open_acc', 'revol_bal', 'revol_util', 'total_acc', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int']

Processing columns to bring them to numeric forms to allow subjecting them to utilities available for analysis, like plotting, correlation functions etc.

Loan status is variable of type object which has string which is describing the status of loan. The various values available in this field are

```
Fully Paid     32950
Charged Off     5627
Current         1140
```

As is these values will be difficult to use for analysis, and hence will be converting them to numeric values. As there are not many in value category ‘Current’, and as the status of loan does not say default, it will be merged into value ‘Fully Paid’.

```
# loan_status - converting loan_status from object to int64
loan_processed=loan_processed
loan_processed['loan_status']=loan_processed['loan_status'].apply(lambda x:1 if x=='Charged Off' else 0)
loan_processed['loan_status']=loan_processed['loan_status'].apply(lambda x:pd.to_numeric(x))
```

Loan term provides information about the term for which the loan has been taken, and its in a form of string as below

```
36 months    29096
60 months    10621
```

```
# term - converting term from object to int64
loan_processed=loan_processed
loan_processed['term']=loan_processed['term'].apply(lambda x:36 if x==' 36 months' else 60)
loan_processed['term']=loan_processed['term'].apply(lambda x:pd.to_numeric(x))
```

Interest rate against the loan taken by the customer. This values varies a lot and has % in the value which needs to removed to make data fit for further analytical processing. Below are the values and code to process this data and bring in to numeric form:

```
10.99%    956
13.49%    826
11.49%    825
7.51%     787
7.88%     725
         ...
24.40%      1
21.48%      1
22.64%      1
17.44%      1
18.72%      1
Name: int_rate, Length: 371, dtype: int64
```

```
# int_rate - converting term from object to int64
loan_processed['int_rate']=loan_processed['int_rate'].apply(lambda
x:pd.to_numeric(x.split('%')[0]))
```

Grade processing. Every applicant who submits a request for loan is reviewed and provided a grade this grade is in the form as follows, and not very process friendly, hence converting it to a numeric form:

```
B    12020
A    10085
C     8098
D     5307
E     2842
F     1049
G      316
```

```
# grade - converting from object to int64
loan_processed['grade']=loan_processed['grade'].apply(lambda x:1 if x=='A' else x)
loan_processed['grade']=loan_processed['grade'].apply(lambda x:2 if x=='B' else x)
loan_processed['grade']=loan_processed['grade'].apply(lambda x:3 if x=='C' else x)
loan_processed['grade']=loan_processed['grade'].apply(lambda x:4 if x=='D' else x)
loan_processed['grade']=loan_processed['grade'].apply(lambda x:5 if x=='E' else x)
loan_processed['grade']=loan_processed['grade'].apply(lambda x:6 if x=='F' else x)
loan_processed['grade']=loan_processed['grade'].apply(lambda x:7 if x=='G' else x)
```

Number of years an employee has been working is an important criterion for analysis to predict if loan would be cleared in time or not. This data is logged in the database in various forms which needs to be cleaned and brought to a form good for analysis:

```
10+ years    8879
< 1 year     4583
2 years      4388
3 years      4095
4 years      3436
5 years      3282
1 year       3240
6 years      2229
7 years      1773
8 years      1479
9 years      1258
Name: emp_length, dtype: int64
```

```
# fixing emp_length, which formats like '10+ years', '< 1 year', '2 years'
loan_processed["emp_length"]=loan_processed["emp_length"].str.replace("+", "")
loan_processed["emp_length"]=loan_processed["emp_length"].str.replace("< ", "")
loan_processed["emp_length"]=loan_processed["emp_length"].str.replace(" years", "")
loan_processed["emp_length"]=loan_processed["emp_length"].str.replace(" year", "")
loan_processed['emp_length']=loan_processed['emp_length'].apply(lambda x:pd.to_numeric(x))
```

Verification Status is set by bank once the salary of the employee is verified. This is a key indicator for eligibility of the employee for a bank loan. This is marked in a non-number format which needs to be converted to a numeric form:

```
Not Verified       16921
Verified           12809
Source Verified     9987
```

```
# 'verification_status - converting 'verification_status from object to int64
loan_processed['verification_status']=loan_processed['verification_status'].apply(lambda x:0 if x=='Not Verified' else 1)
```

Purpose column provide information about why a user is applying for the loan. Once again this column is in descriptive form which needs to be provided with a number for analytical use

```
debt_consolidation    18641
credit_card            5130
other                  3993
home_improvement       2976
major_purchase         2187
small_business         1828
car                    1549
wedding                 947
medical                 693
moving                  583
vacation                381
house                   381
educational             325
renewable_energy        103
Name: purpose, dtype: int64
```

```
# purpose - converting term from object to int64
p=loan_processed['purpose'].value_counts()
pi=p.index.tolist()
pi.index('car')
loan_processed['purpose']=loan_processed['purpose'].apply(lambda x:pi.index(x))
```

Revol_util give information about how much of the credit available with a user being used. This once again givens an important insight about clients repayment credibility.

```
0%         977
0.20%       63
63%         62
40.70%      58
0.10%       58
          ...
0.75%        1
1.88%        1
81.31%       1
5.34%        1
88.48%       1
Name: revol_util, Length: 1089, dtype: int64
```

```
#loan_processed['revol_util']=loan_processed['revol_util'].apply(lambda x:pd.to_numeric(x.split('%')[0]))
loan_processed['revol_util']=loan_processed['revol_util'].str.replace("%","")
loan_processed['revol_util']=loan_processed['revol_util'].apply(lambda x:pd.to_numeric(x))
```

Home ownership status tells a bank about the client financial status who is applying for loan, and if he is owner or had a mortgaged property, he could be treated as someone who can be offered a loan or not. The values are as string and will be converted to numbers.

```
RENT        18899
MORTGAGE    17659
OWN          3058
OTHER          98
NONE            3
Name: home_ownership, dtype: int64
```

```
# home_ownership - converting term from object to int64
loan_processed['home_ownership']=loan_processed['home_ownership'].apply(lambda x:1 if x=='RENT' else x)
loan_processed['home_ownership']=loan_processed['home_ownership'].apply(lambda x:2 if x=='MORTGAGE' else x)
loan_processed['home_ownership']=loan_processed['home_ownership'].apply(lambda x:3 if x=='OWN' else x)
loan_processed['home_ownership']=loan_processed['home_ownership'].apply(lambda x:4 if x=='OTHER' else x)
loan_processed['home_ownership']=loan_processed['home_ownership'].apply(lambda x:4 if x=='NONE' else x)
```

After all the processing is done we are left with 24 columns as follows:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Data columns (total 24 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   loan_amnt               39717 non-null  int64  
 1   funded_amnt             39717 non-null  int64  
 2   funded_amnt_inv         39717 non-null  float64
 3   term                    39717 non-null  int64  
 4   int_rate                39717 non-null  float64
 5   installment             39717 non-null  float64
 6   grade                   39717 non-null  int64  
 7   emp_length              38642 non-null  float64
 8   home_ownership          39717 non-null  int64  
 9   annual_inc              39717 non-null  float64
 10  verification_status     39717 non-null  int64  
 11  loan_status             39717 non-null  int64  
 12  purpose                 39717 non-null  int64  
 13  dti                     39717 non-null  float64
 14  inq_last_6mths          39717 non-null  int64  
 15  mths_since_last_delinq  14035 non-null  float64
 16  open_acc                39717 non-null  int64  
 17  revol_bal               39717 non-null  int64  
 18  revol_util              39667 non-null  float64
 19  total_acc               39717 non-null  int64  
 20  total_pymnt             39717 non-null  float64
 21  total_pymnt_inv         39717 non-null  float64
 22  total_rec_prncp         39717 non-null  float64
 23  total_rec_int           39717 non-null  float64
dtypes: float64(12), int64(12)
memory usage: 7.3 MB
```

Analyzing Data for Correlations

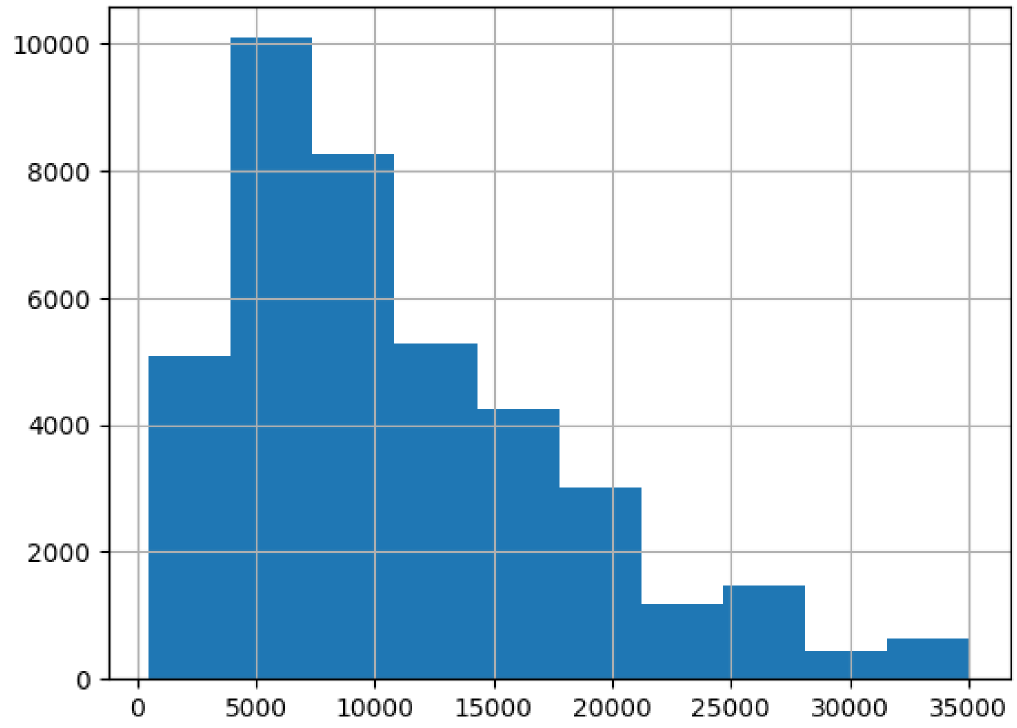
Univariate Analysis

In Univariate analysis we try to take one variable at a time to see how its behaving and how that data could impact the overall correlation with the variable of business interest.

This graph is plotted for amount funded in the loan. What is the distribution for loans being provided.

```
loan_processed['funded_amnt'].hist()
```

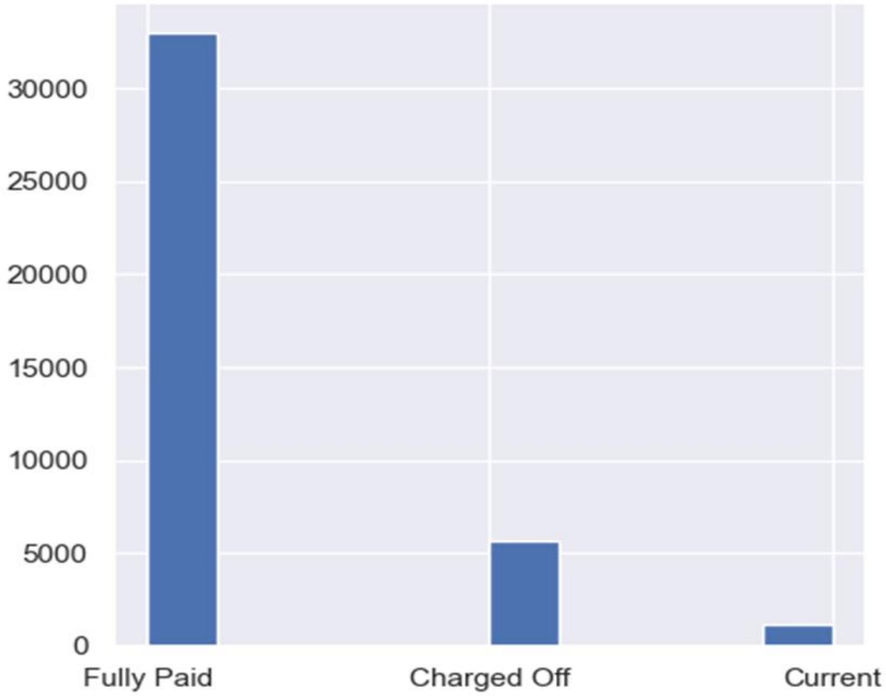
```
<AxesSubplot:>
```



The graph below provides an insight to the status of loans provided. How many people have repaid their loans, how many are currently paying without failure, and how many have defaulted. This data can be a variable to finding correlation of defaulters with respective other variables in the dataset.

```
loan['loan_status'].hist()
```

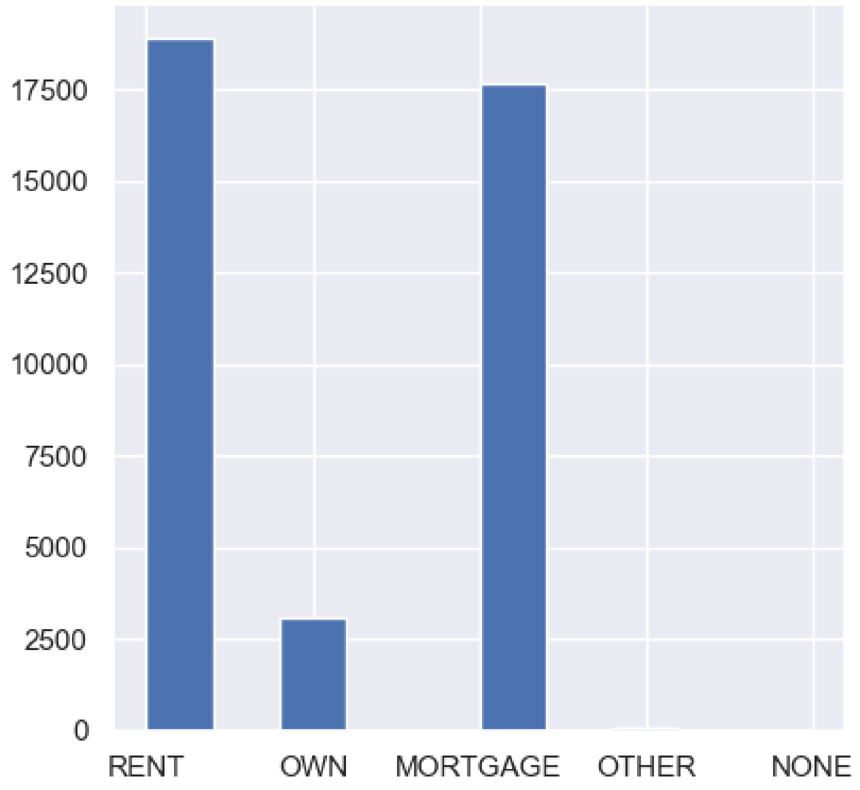
```
<AxesSubplot:>
```



Below is the graph for the kind of property the applicant is staying in, whether its owned by applicant, or mortgaged or rented etc. Once again, could an important indicator for predicting defaulters.

```
loan['home_ownership'].hist()
```

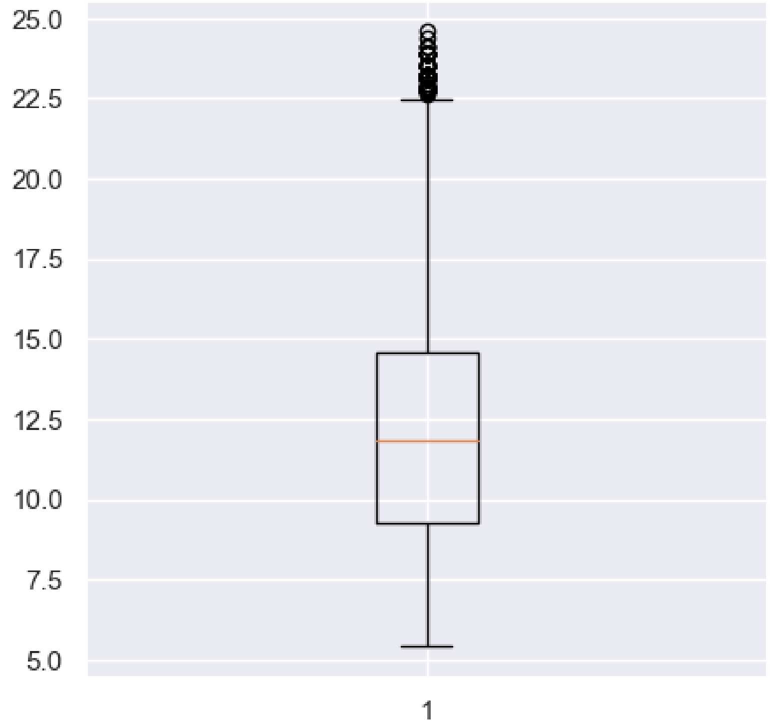
<AxesSubplot:>



Below is a box plot of **interest rates** charged by the banks for loans provided. This gives us information about how interest rates are varying (min, max, median, 25th and 75th percentile), and what are outliers.

```python
plt.boxplot(loan_processed['int_rate'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1f02fe0ba58>,
  <matplotlib.lines.Line2D at 0x1f02fe0bcc0>],
 'caps': [<matplotlib.lines.Line2D at 0x1f02fe0bf98>,
  <matplotlib.lines.Line2D at 0x1f02fe2e2b0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f02fe0b7b8>],
 'medians': [<matplotlib.lines.Line2D at 0x1f02fe2e588>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f02fe2e860>],
 'means': []}
```



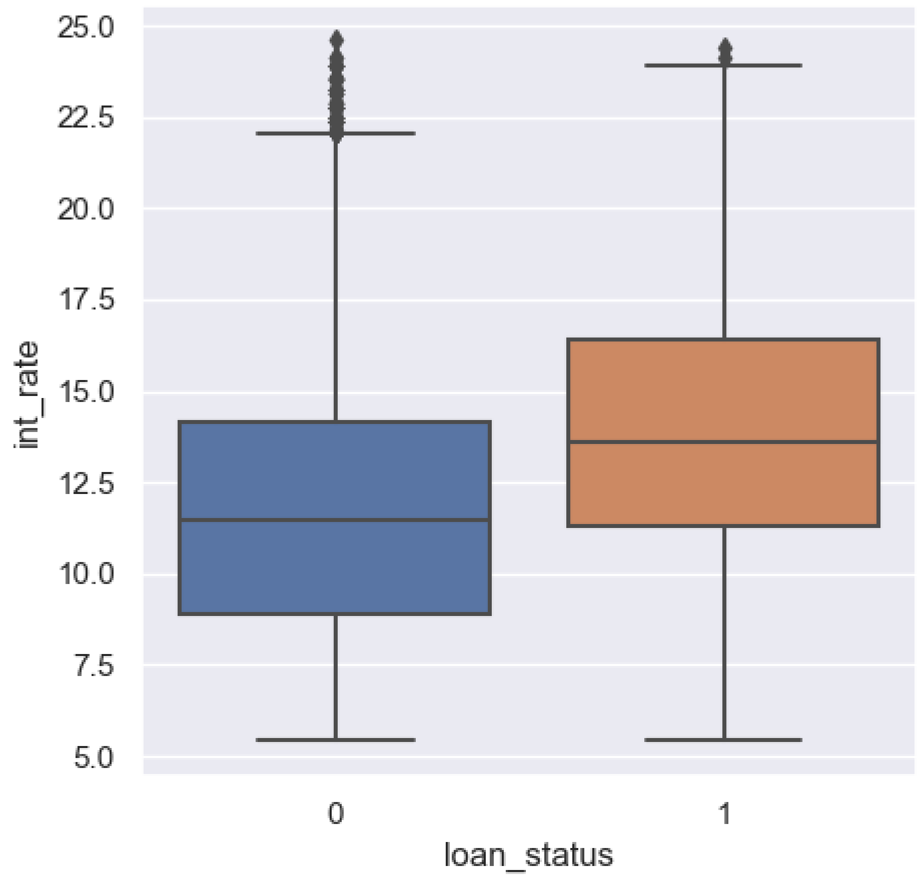
Bivariate Analysis

In bivariate analysis the comparison is between two variables to plot the relation between them. Below are few comparisons to understand how one is affecting other

This box plot compares the relationship between loan status and interest rate. We can clearly see that interest rate is charged more for the clients who are defaults. The higher interest rates might have been selected for possible defaulters to mitigate the risk.

```
sns.boxplot(x='loan_status', y='int_rate', data=loan_processed)
```

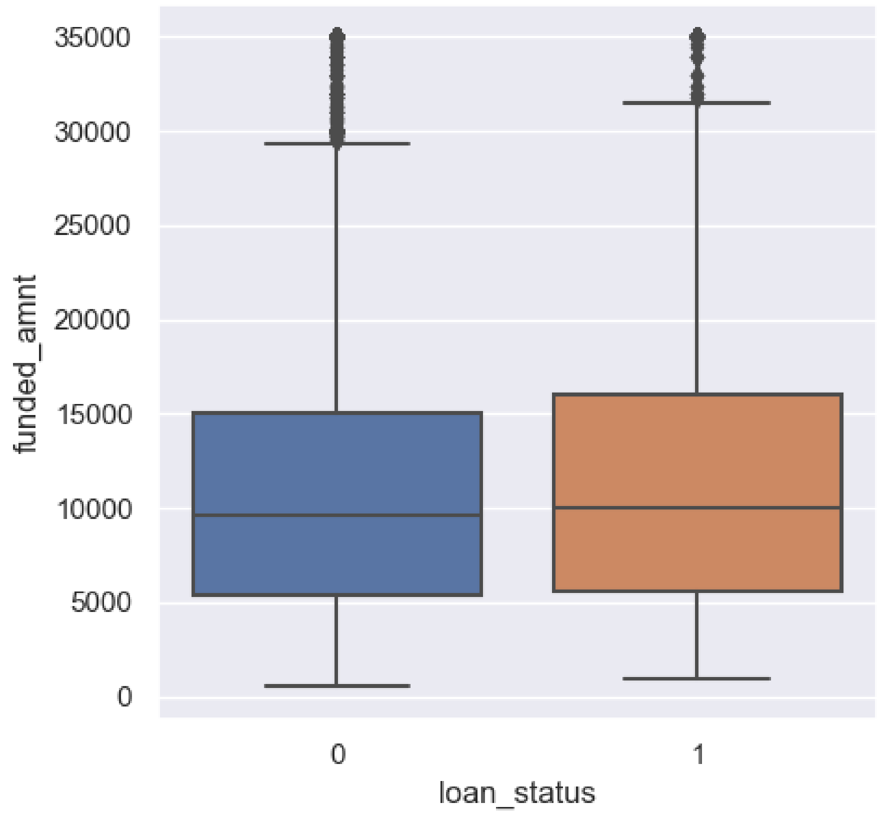
```
<AxesSubplot:xlabel='loan_status', ylabel='int_rate'>
```



Below plot is for the **loan status** and **amount funded**. There is a little higher correlation with defaulters and higher amounts.

```
sns.boxplot(x='loan_status', y='funded_amnt', data=loan_processed)
```

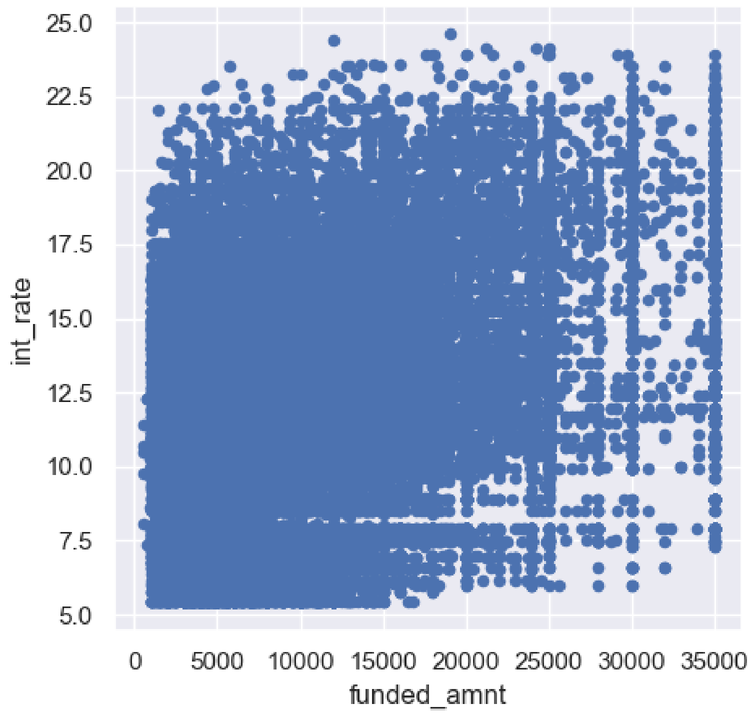
```
<AxesSubplot:xlabel='loan_status', ylabel='funded_amnt'>
```



The below **scatter plot** between **amount funded** and **interest rate** is little difficult to ready but gives an idea about how the relation and distribution is. The subsequent plot takes a view with part of data to get a better picture in the area of interest

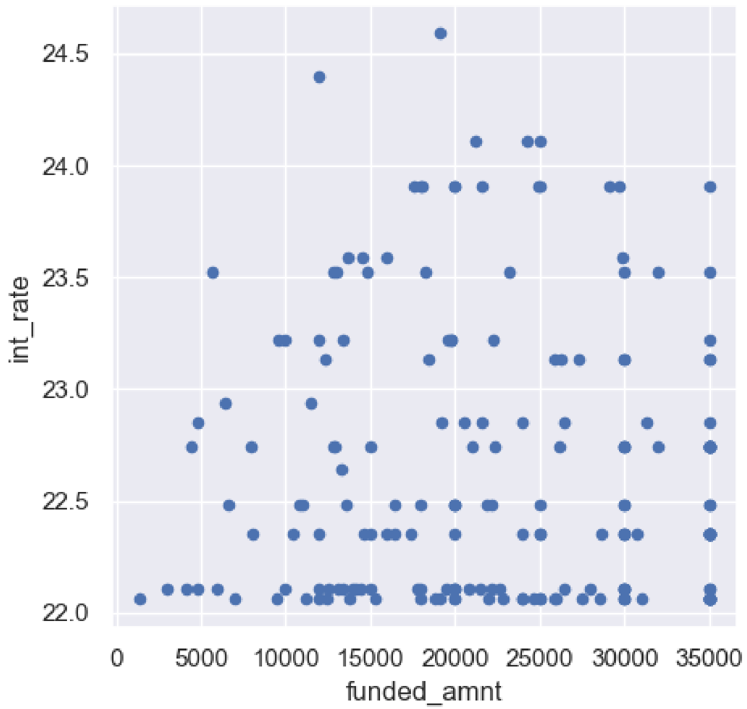
```
loan_processed.plot.scatter(x='funded_amnt', y='int_rate')
```

```
<AxesSubplot:xlabel='funded_amnt', ylabel='int_rate'>
```




```
loan_processed[loan_processed['int_rate'] > 22].plot.scatter(x='funded_amnt', y='int_rate')
```

```
<AxesSubplot:xlabel='funded_amnt', ylabel='int_rate'>
```



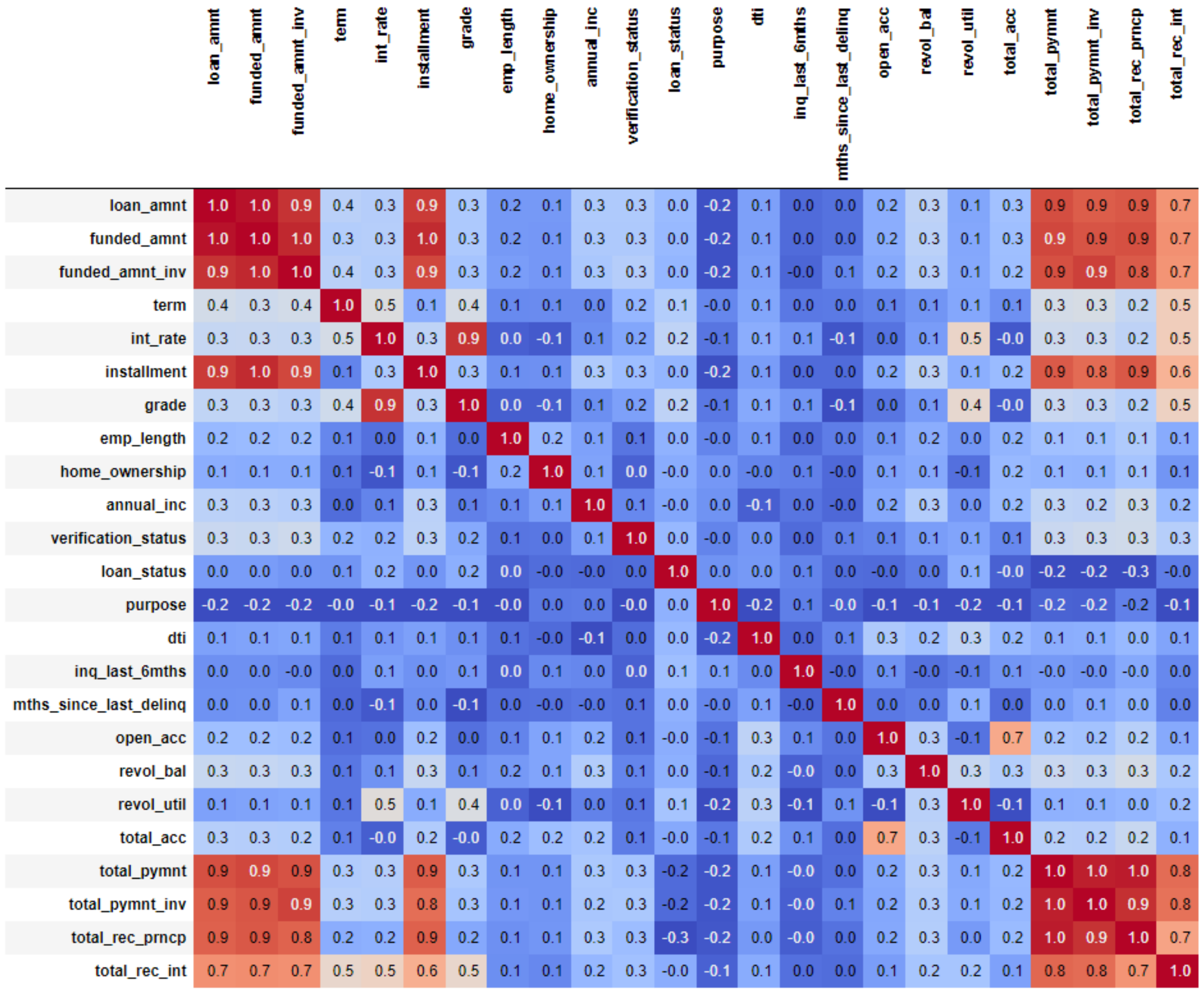
Here we are trying to find out correlation between all fields with loan_status to find the which fields are a good indicator of possible defaulters

```
corr1 = loan_processed.corr()['loan_status'].sort_values(ascending=False)
print('Most Positive Correlations:\n', corr1.head(10))
print('\nMost Negative Correlations:\n', corr1.tail(10))
```

```
Most Positive Correlations:
 loan_status            1.000000
int_rate               0.196253
grade                  0.190409
term                   0.146038
revol_util             0.096560
inq_last_6mths         0.071717
loan_amnt              0.048217
funded_amnt            0.045544
dti                    0.041701
verification_status    0.037280
Name: loan_status, dtype: float64

Most Negative Correlations:
 mths_since_last_delinq    0.004941
revol_bal                 0.003369
open_acc                 -0.010742
total_rec_int            -0.010780
home_ownership           -0.016313
total_acc                -0.023563
annual_inc               -0.041662
total_pymnt_inv          -0.236232
total_pymnt              -0.238844
total_rec_prncp          -0.335019
Name: loan_status, dtype: float64
```

Correlation Matrix Plots to see how each of these variables are interrelated.



To summarize - for finding required correlation following steps were done

1. Tried to understand data in dataset provide
2. Process data for analysis
3. Analyze the processed and draw possible conclusions.

Conclusion:

1. We can safely observe correlations between the variables
2. Business would be able use the processed data to safely draw conclusions.
3. These conclusions can be part of the decision, whether to provide or deny loan

Thanks for providing this opportunity to do a study and provide insights.

Harish Kulkarni