Case Study

Loan Dataset:

This data set **includes customers who have paid off their loans, who have been past due and put into collection** without paying back their loan and interests,

Code:

Import Required Modules

```
import pandas as pd
import seaborn as sns
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
```

pandas - Pandas for Data Frame

datetime - datetime for work with the time zone, months, year

numpy - NumPy for work with numeric

matplotlib.pyplot - Data Visualization

seaborn - Data Visualization

Step1: Read the given loan.csv file

```
loandata = pd.read_csv('loan.csv')
loandata.head()
```

- "loandata" is a variable defined for loan data
- Print the top 5 rows from the dataframe

Step 2: Data Filter and Cleaning

2(a) Remove columns containing either 75 % or more than 75 % Null Values

Most of the columns contain null values, which contain huge amounts of data. Remove the columns that have greater than 50% or 75% null values from the entire data frame.

```
loandata = loandata.loc[:, loandata.isnull().mean() < .75]
loandata.head(5)
```

2(b). List out the columns after dropping them

List of the columns after dropping them above .

```
loandata.columns      ###List out the columns after removing columns
```

```
Index(['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',
       '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',
       'collections_12_mths_ex_med', 'policy_code', 'application_type',
       'acc_now_delinq', 'chargeoff_within_12_mths', 'delinq_amnt',
       'pub_rec_bankruptcies', 'tax_liens'],
      dtype='object')
```

2(c). Count the missing values from all the columns.

Count the null values from all the columns in a DataFrame.

```
loanna = loandata.isnull().sum()
loanna
```

Output:

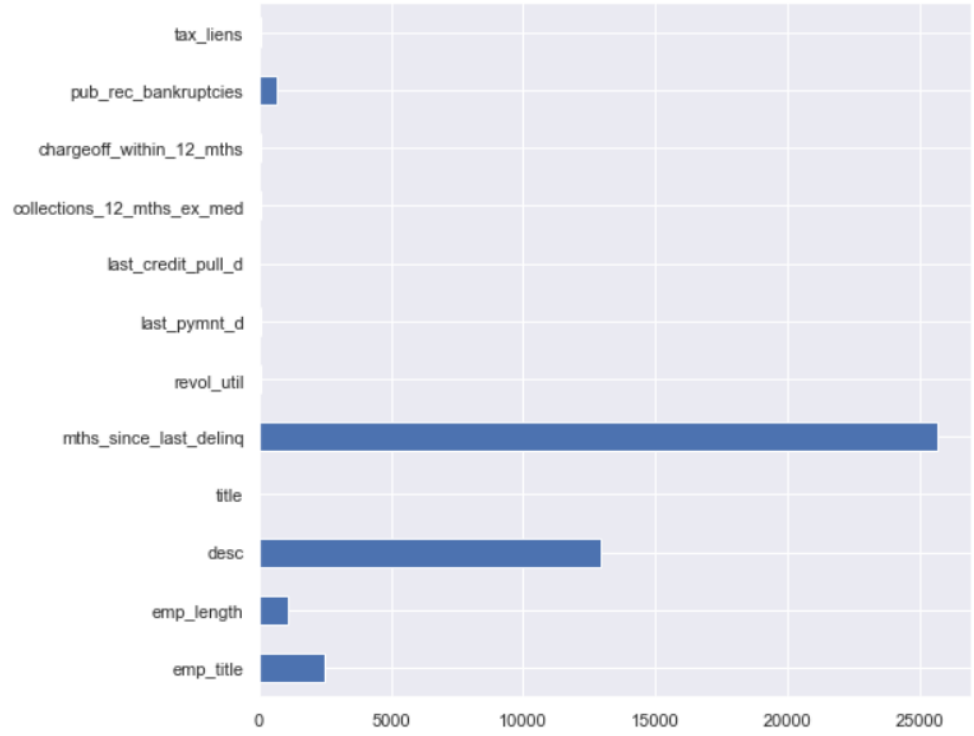
```
id                            0
member_id                     0
loan_amnt                     0
funded_amnt                   0
funded_amnt_inv               0
term                          0
int_rate                      0
installment                   0
grade                         0
sub_grade                     0
emp_title                  2459
emp_length                 1075
home_ownership                0
annual_inc                    0
verification_status           0
issue_d                       0
loan_status                   0
pymnt_plan                    0
url                           0
desc                      12940
purpose                       0
title                        11
zip_code                      0
addr_state                    0
dti                           0
delinq_2yrs                   0
earliest_cr_line              0
inq_last_6mths                0
mths_since_last_delinq    25682
open_acc                      0
pub_rec                       0
revol_bal                     0
revol_util                   50
total_acc                     0
initial_list_status           0
out_prncp                     0
out_prncp_inv                 0
total_pymnt                   0
total_pymnt_inv               0
total_rec_prncp               0
total_rec_int                 0
total_rec_late_fee            0
recoveries                    0
collection_recovery_fee       0
last_pymnt_d                 71
last_pymnt_amnt               0
last_credit_pull_d            2
collections_12_mths_ex_med   56
policy_code                   0
application_type              0
acc_now_delinq                0
chargeoff_within_12_mths     56
delinq_amnt                   0
pub_rec_bankruptcies        697
tax_liens                    39
dtype: int64
```

2(d). Display the bar chart of null values from the dataframe

Display the bar histogram chart for null values count from the columns

```
loandata.isnull().sum()[loandata.isnull().sum() >0].plot(kind='barh')
```

```
<AxesSubplot:>
```



Step3: Fill the Null values and manipulate the values in all columns

3(a). Fill the null values in the columns

Fill the null values with NA

```
loandata['emp_length'] = loandata['emp_length'].fillna('NA')
loandata['desc'] = loandata['desc'].fillna('NA')
```

3(b). Fill the missing values with the observable/Relevant data

Fill the null values with relevant data, based on the column data.

```
loandata['emp_title'] = loandata['emp_title'].fillna('other')
loandata['title'] = loandata['title'].fillna('other')
loandata['mths_since_last_delinq'] = loandata['mths_since_last_delinq'].fillna('0.0')
loandata['collections_12_mths_ex_med'] = loandata['collections_12_mths_ex_med'].fillna('0.0')
loandata['chargeoff_within_12_mths'] = loandata['chargeoff_within_12_mths'].fillna('0.0')
loandata['revol_util'] = loandata['revol_util'].fillna('0%')
loandata['pub_rec_bankruptcies'] = loandata['pub_rec_bankruptcies'].fillna('0.0')
loandata['tax_liens'] = loandata['tax_liens'].fillna('0.0')
```

3(c). Remove left space and extra characters from the column values

Some of the column data was not formatted properly, applying the lambda function to remove extra spaces and special characters etc.,

```
loandata['term'] = loandata['term'].apply(lambda x: x.lstrip())
```

```
loandata['desc'] = loandata['desc'].apply(lambda x: x.lstrip())
```

```
loandata['zip_code'] = loandata['zip_code'].apply(lambda x: x.rstrip('xx'))
```

```
loandata['int_rate'] = loandata['int_rate'].apply(lambda x: x.rstrip('%'))
```

3(d). Fill the data in Date columns

Date formats are not properly formed, to avoid the issues/errors converting the date format with a normal format's like %y-%d-%m or %d-%m-%y or %m-%d-%y

```
loandata['last_pymnt_d'] = loandata['last_pymnt_d'].add('-2023') ###Adding the year
```

```
loandata['issue_d'] = loandata['issue_d'].add('-2023')   ###Adding the year
```

Apply the function Date for convertion:

Parse the dates from the DataFrame (strptime) and converting them with normal format using strftime. After that applying the function in the date columns using with lambda

```
def convert_date(date_str):
    parsed_date = datetime.strptime(str(date_str), '%b-%d-%Y')
    formatted_date = parsed_date.strftime("%Y-%m-%d")
    return formatted_date
```

```
loandata['last_pymnt_d'] = loandata['last_pymnt_d'].apply(lambda x: convert_date(x))
loandata['last_pymnt_d']
```

```
loandata['issue_d'] = loandata['issue_d'].apply(lambda x: convert_date(x))
loandata['issue_d']
```

Updating the 'earliest_cr_line' date format, which it have month-year and convert to year-month-date:

One of the column days are missing in the entire column. Adding the random days(1-31) value using with np.random.radint. And adding with join in the specific column.

```
loandata['earliest_cr_line'].shape[0]
```

```
39717
```

Adding Random days in a new column 'days'

```
import numpy as np
loandata['days'] = np.random.randint(1, 31, loandata['earliest_cr_line'].shape[0])
loandata['days'].shape[0]
```

```
39717
```

```
loandata['earliestcr_format'] = loandata[['earliest_cr_line','days']].apply(lambda x: '-'.join(x.values.astype(str
loandata['earliestcr_format'].unique()
```

```
array(['Jan-85-15', 'Apr-99-6', 'Nov-01-22', ..., 'Feb-85-21',
       'Feb-69-19', 'Nov-88-2'], dtype=object)
```

Step4: Filter the outlier and Filter it

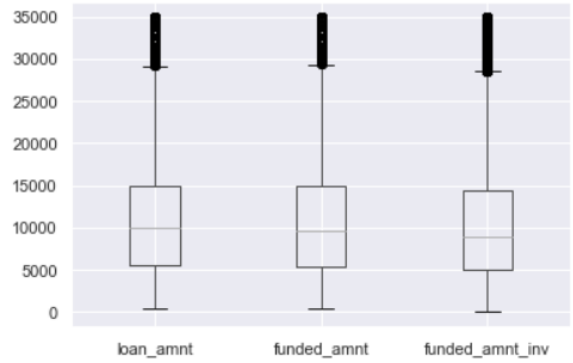
Create box plot and check Outlier for set of the columns and apply the inter-quartile range to detect the outliers and filter it

```
sns.set(rc={"figure.figsize":(6, 4)})
```

04(a). loan_amnt, funded_amnt, funded_amnt_inv

```
loandata.boxplot(column=['loan_amnt', 'funded_amnt', 'funded_amnt_inv'])
```

```
<AxesSubplot:>
```



Filter the data with IQR (Interquartile range)

Example:

Q1 = data['column'].quantile(0.25)

Q3 = data['column'].quantile(0.75)

IQR = Q3 - Q1 #IQR is interquartile range.

filter = (data['column'] >= Q1 - 1.5 * IQR) & (data['column'] <= Q3 + 1.5 *IQR)

data = data.loc[filter]

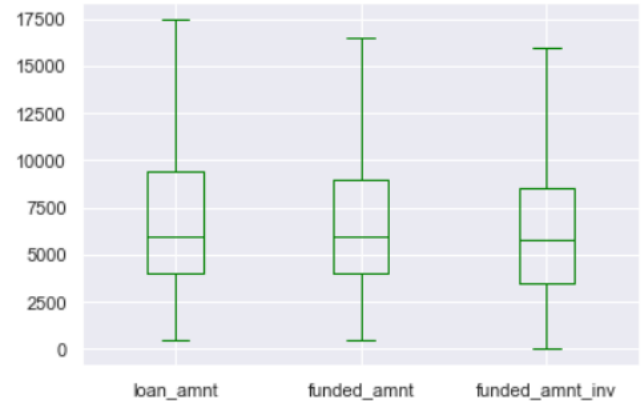
```
Q1 = loandata['loan_amnt'].quantile(0.25)
Q3 = loandata['loan_amnt'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['loan_amnt'] >= Q1 - 1.5 * IQR) & (loan['loan_amnt'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
Q1 = loandata['funded_amnt'].quantile(0.25)
Q3 = loandata['funded_amnt'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['funded_amnt'] >= Q1 - 1.5 * IQR) & (loandata['funded_amnt'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
Q1 = loandata['funded_amnt_inv'].quantile(0.25)
Q3 = loandata['funded_amnt_inv'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['funded_amnt_inv'] >= Q1 - 1.5 * IQR) & (loandata['funded_amnt_inv'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
loandata.boxplot(column=['loan_amnt', 'funded_amnt', 'funded_amnt_inv'], color="green")
```

```
<AxesSubplot:>
```

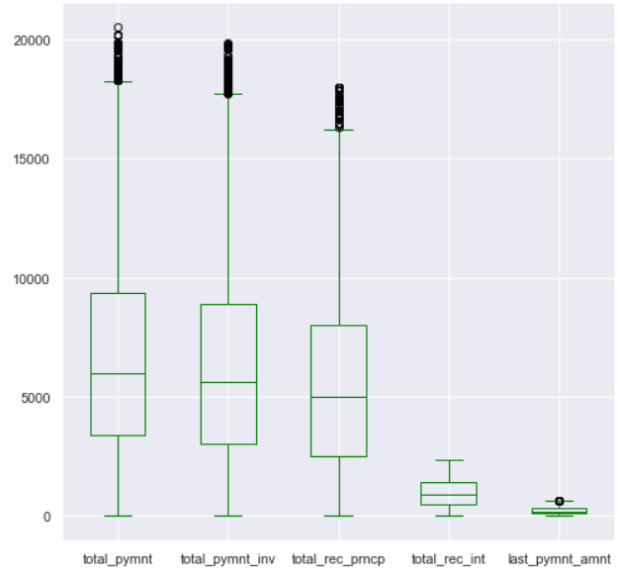


04(b). total_pymnt, total_pymnt_inv, total_rec_prncp, total_rec_int, last_pymnt_amnt

```
sns.set(rc={"figure.figsize":(9, 9)})
```

```
loan.boxplot(column=['total_pymnt','total_pymnt_inv','total_rec_prncp','total_rec_int', 'last_pymnt_amnt'], color="green")
```

```
<AxesSubplot:>
```



Filter the data with IQR (Interquartile range)

```
Q1 = loandata['total_pymnt'].quantile(0.25)
Q3 = loandata['total_pymnt'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['total_pymnt'] >= Q1 - 1.5 * IQR) & (loandata['total_pymnt'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
Q1 = loandata['total_pymnt_inv'].quantile(0.25)
Q3 = loandata['total_pymnt_inv'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['total_pymnt_inv'] >= Q1 - 1.5 * IQR) & (loandata['total_pymnt_inv'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

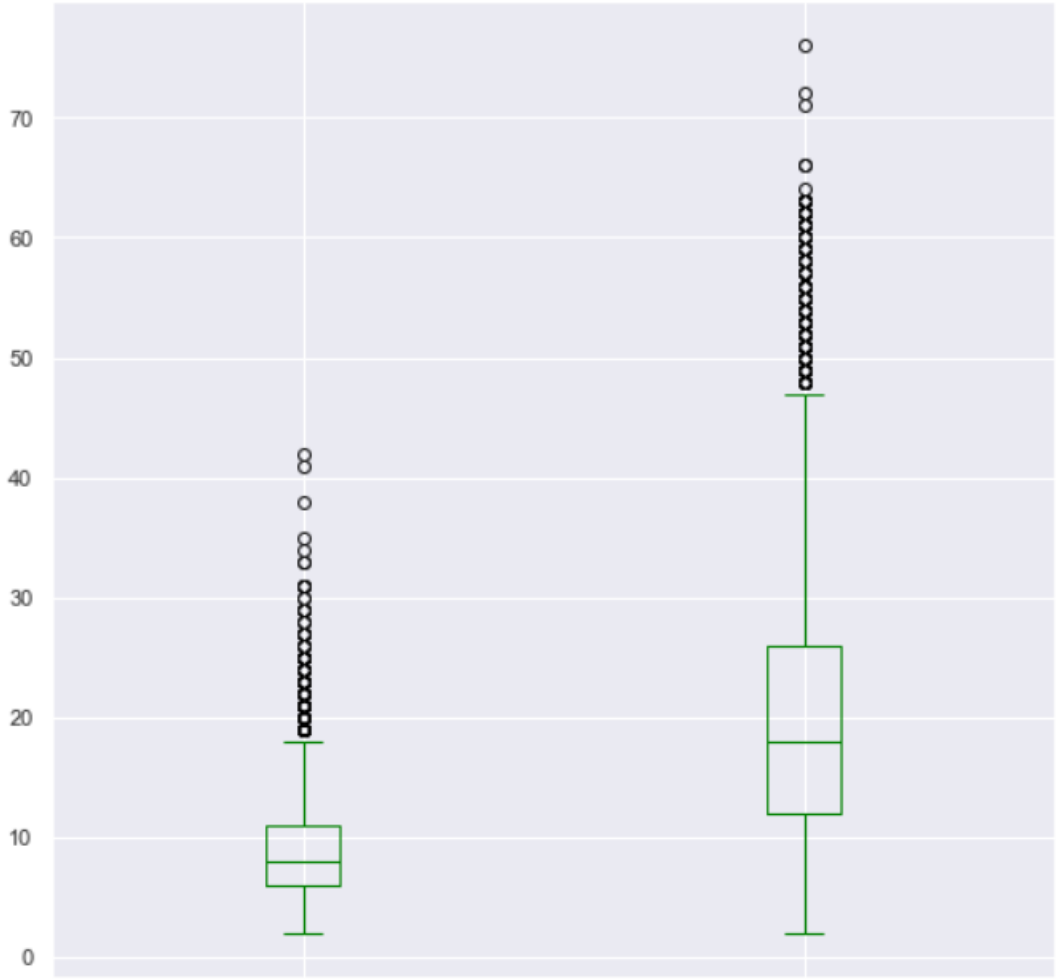
```
Q1 = loandata['last_pymnt_amnt'].quantile(0.25)
Q3 = loandata['last_pymnt_amnt'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['last_pymnt_amnt'] >= Q1 - 1.5 * IQR) & (loandata['last_pymnt_amnt'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
loandata.boxplot(column=['total_pymnt','total_pymnt_inv','total_rec_prncp','total_rec_int', 'last_pymnt_amnt'], color="blue")
```

4(c). total_acc and open_acc

```
loandata.boxplot(column=['open_acc','total_acc'], color="green")
```

```
<AxesSubplot:>
```

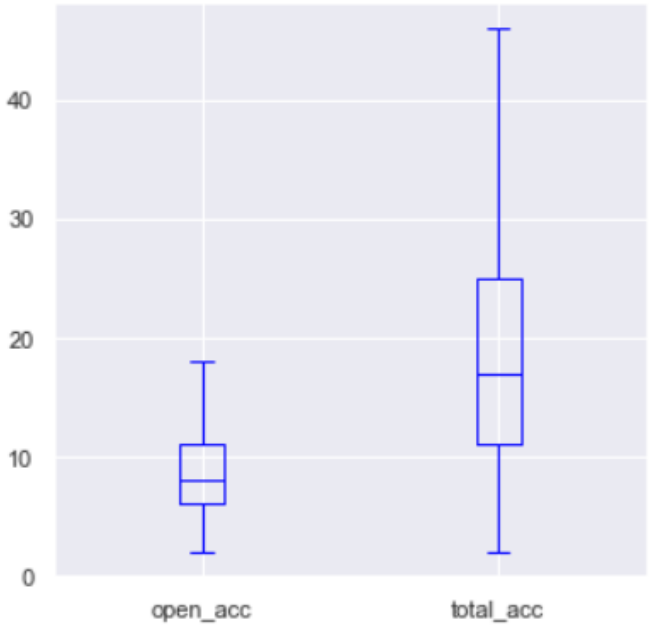



```
Q1 = loandata['open_acc'].quantile(0.25)
Q3 = loandata['open_acc'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['open_acc'] >= Q1 - 1.5 * IQR) & (loandata['open_acc'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
Q1 = loandata['total_acc'].quantile(0.25)
Q3 = loandata['total_acc'].quantile(0.75)
IQR = Q3 - Q1    #IQR is interquartile range.

filter = (loandata['total_acc'] >= Q1 - 1.5 * IQR) & (loandata['total_acc'] <= Q3 + 1.5 *IQR)
loandata = loandata.loc[filter]
```

```
sns.set(rc={"figure.figsize":(5, 5)})
loandata.boxplot(column=['open_acc','total_acc'], color="blue")
```

```
<AxesSubplot:>
```



Step5: Data Analysis

5(a). Derived Metrics

```
loandata['loan_inc_ratio'] = loandata['loan_amnt']/loandata['annual_inc']
loandata['loan_inc_ratio'].head(8)
```

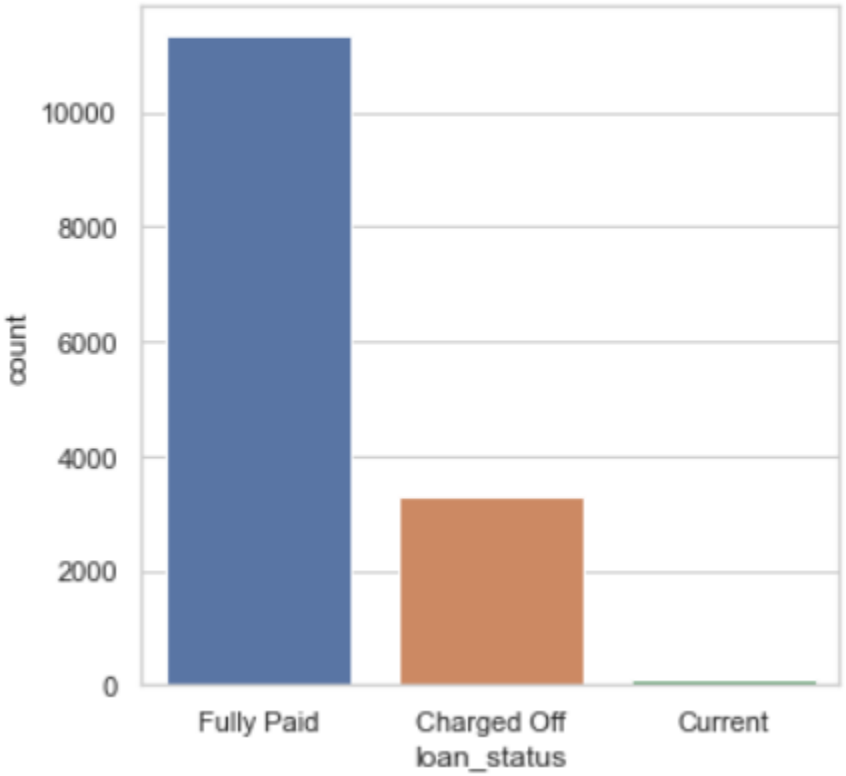
```
0    0.208333
1    0.083333
3    0.203252
4    0.037500
5    0.138889
7    0.062500
8    0.140000
9    0.358333
Name: loan_inc_ratio, dtype: float64
```

5(b). Categorical Variables

Loan Status

```
sns.set_theme(style="whitegrid")
sns.countplot(loandata['loan_status'])
```

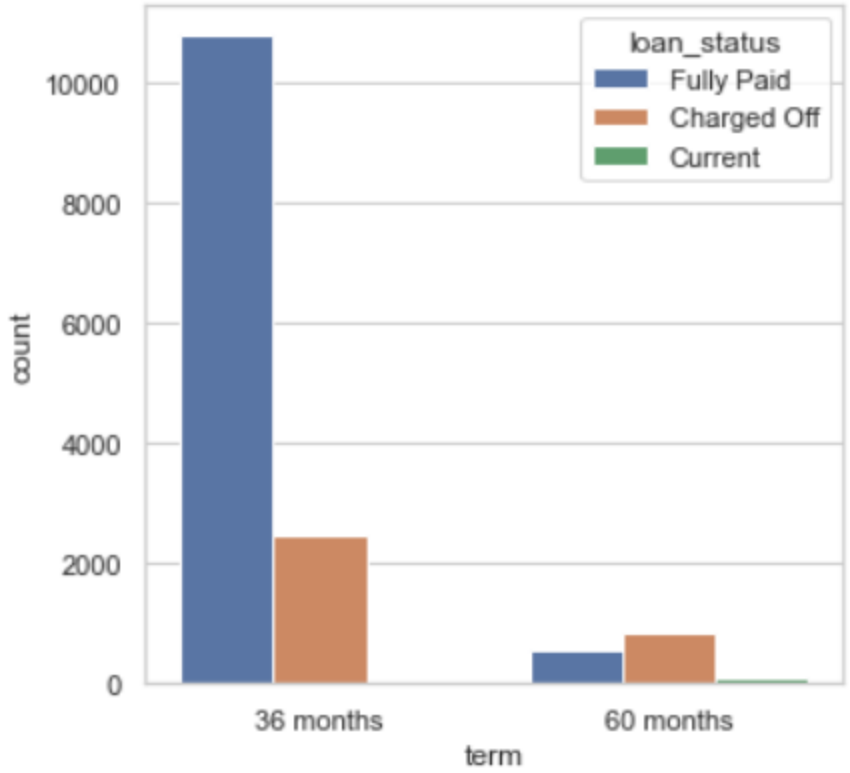
```
<AxesSubplot:xlabel='loan_status', ylabel='count'>
```



Loan Term with 'Loan Status'

```
sns.countplot(loandata['term'],hue=loandata['loan_status'])
```

```
<AxesSubplot:xlabel='term', ylabel='count'>
```



5(c). Correlation from the dataframe

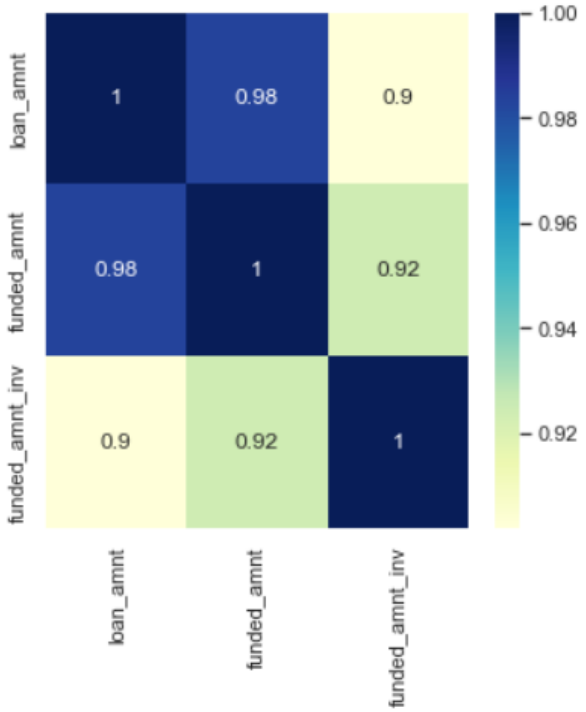
correlation 'loan_amnt', 'funded_amnt', 'funded_amnt_inv'

correlation 'loan_amnt', 'funded_amnt', 'funded_amnt_inv'

```
datcorla = loandata[['loan_amnt', 'funded_amnt', 'funded_amnt_inv']]
```

```
sns.heatmap(datcorla.corr(), cmap="YlGnBu", annot=True)
```

```
<AxesSubplot:>
```

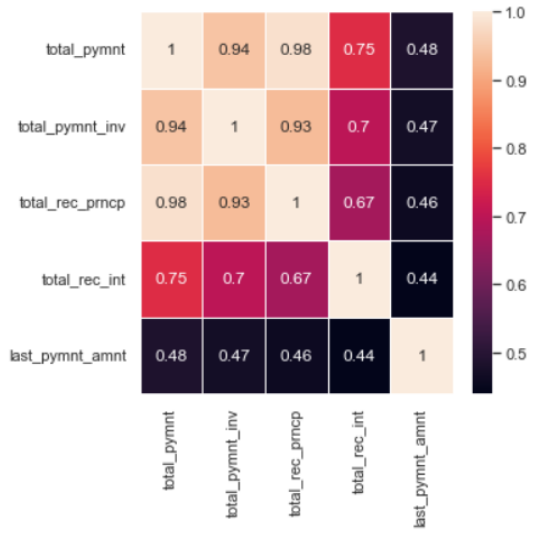


correlation from 'total_pymnt','total_pymnt_inv','total_rec_prncp','total_rec_int', 'last_pymnt_amnt'

```
datcorlb = loandata[['total_pymnt','total_pymnt_inv','total_rec_prncp','total_rec_int', 'last_pymnt_amnt']]
```

```
sns.heatmap(datcorlb.corr(), annot=True, linewidth=.5)
```

```
<AxesSubplot:>
```



Loan amount applied by member living in own or rent and analysing with funded amount & annual income (Risk analisys)

```
df = loandata.groupby(by=['home_ownership','loan_amnt']).mean()
df1 = df[['funded_amnt','annual_inc']]
df1
```

		funded_amnt	annual_inc
home_ownership	**loan_amnt**		
	500	500.000000	30968.013333
	800	800.000000	35000.000000
MORTGAGE	**1000**	1000.000000	58087.648649
	1050	1050.000000	31200.000000
	1100	1100.000000	36000.000000
...	**...**	...	...
	16750	11600.000000	39000.000000
	16775	9275.000000	83000.000000
RENT	**16800**	10775.000000	60000.000000
	17000	10987.500000	62500.000000
	17500	11808.333333	53600.000000

Heat map for risk analysis corelation between laon_amnt, funded_amnt and anual_inc

```
risk = loandata[['loan_amnt','funded_amnt','annual_inc']]
```

```
sns.heatmap(risk.corr(), annot=True, linewidth=.5)
```

```
<AxesSubplot:>
```