Lending Club Case Study

By :
Govind Biradar &
Rohit Jaiswal

Problem Statement :

How to identify the risky loan applicants based on their profile ?

Two types of risk associated with the bank's decision :

- If the applicant likely to repay the loan, then not approving loan results in loss of business.
- If the applicant not likely to repay the loan, then approving loan may lead to financial loss.

Aim :

To identify patterns which indicate if a person is likely to repay or default the loan.

Analysis :

Using Exploratory Data Analysis (EDA) to understand how consumer and loan attributes influence the tendency of repay or default

Data Analysis :

- Quantum of data
- Shape (rows and columns)
- Data distribution

In [3]:
```python
# check the shape of data to see the amount of data we are going to dealt with.
df.shape
```

Out[3]:
```
(39717, 111)
```

In [5]:
```python
df.info()
```

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

In [6]:
```python
# Describe the data to check the data distribution in a glimpse
df.describe()
```

Out[6]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	installment	annual_inc	dti	delinq_2yrs	inq_last_6mths	...
count	3.971700e+04	3.971700e+04	39717.000000	39717.000000	39717.000000	39717.000000	3.971700e+04	39717.000000	39717.000000	39717.000000	...
mean	6.831319e+05	8.504636e+05	11219.443815	10947.713196	10397.448868	324.561922	6.896893e+04	13.315130	0.146512	0.869200	...
std	2.106941e+05	2.656783e+05	7456.670694	7187.238670	7128.450439	208.874874	6.379377e+04	6.678594	0.491812	1.070219	...
min	5.473400e+04	7.069900e+04	500.000000	500.000000	0.000000	15.690000	4.000000e+03	0.000000	0.000000	0.000000	...
25%	5.162210e+05	6.667800e+05	5500.000000	5400.000000	5000.000000	167.020000	4.040400e+04	8.170000	[illegible]	[illegible]	...
50%	6.656650e+05	8.508120e+05	10000.000000	9600.000000	8975.000000	280.220000	5.900000e+04	13.400000	[illegible]		

Data Cleaning :

- Identify columns having null values entirely.

- Identify columns that may not be suitable for our analysis given our problem statement.

- Remove these unnecessary data.

```
df.isnull().sum()
```

```
id                            0
member_id                     0
loan_amnt                     0
funded_amnt                   0
funded_amnt_inv               0
                            ...
tax_liens                    39
tot_hi_cred_lim           39717
total_bal_ex_mort         39717
total_bc_limit            39717
total_il_high_credit_limit    39717
Length: 111, dtype: int64
```

```
# Sort the null values columns in decreasing order just to feel the quant
df.isna().sum().sort_values(ascending=False).head(60)
```

```
verification_status_joint         39717
annual_inc_joint                  39717
mo_sin_old_rev_tl_op              39717
mo_sin_old_il_acct                39717
bc_util                           39717
bc_open_to_buy                    39717
avg_cur_bal                       39717
acc_open_past_24mths              39717
inq_last_12m                      39717
total_cu_tl                       39717
inq_fi                            39717
total_rev_hi_lim                  39717
```

Further Dropping Unnecessary Columns :

- There are few more columns having "NaN" Values which should be dropped.

- Also there are columns which have more than 30% "NaN" values and these columns are not required for our analysis.

- The columns are "next_pymnt_d", "mths_since_last_record", "mths_since_last_delinq", "desc".

Segmentation Of Columns :

➢ Differentiating the attributes having null values depending on the data types to get either “median” or “mean” values for filling the “NaN” values.

➢ Therefore, the columns are segmented into,

- Categorical columns
- Continuous columns

In [16]:
```
# differentiating the attributes having null values depending on the data types to get either median or mean values for filling
# NaN/Null values in the data
categorical = ['emp_title', 'emp_length', 'title', 'revol_util', 'last_pymnt_d', 'last_credit_pull_d']
continous = ['collections_12_mths_ex_med', 'chargeoff_within_12_mths', 'pub_rec_bankruptcies', 'tax_liens']
```

In [17]:
```
# Get the percentage of data having Nan values
for cat in categorical:
    print(df[df[cat].isna()].shape[0]/df.shape[0] * 100)
```

```
6.191303472064859
2.7066495455346575
0.027695948838029054
0.12589067653649572
0.1787647606818239
0.0050356270614598285
```

In [20]:
```
# Now the problem is how should we replace filling NaN values or Should we ignore it
df.dtypes.sort_values()
```

Out[20]:
```
id                            int64
delinq_amnt                   int64
acc_now_delinq                int64
policy_code                   int64
total_acc                     int64
revol_bal                     int64
pub_rec                       int64
open_acc                      int64
delinq_2yrs                   int64
```

Missing Value Check :

- After sorting out the values we will do value_counts ( ) for "loan_status" to know the shape of Fully paid, Charged off and Current loans.

```
34]: df['loan_status'].value_counts()

34]: Fully Paid     32950
     Charged Off     5627
     Current         1140
     Name: loan_status, dtype: int64

37]: # Now we do not want to analyse people who has already fully paid the loan which we are interested in Charged off and C
     # Current is not defaulted but still let's study those as well


     #We will filter the records and take other than Fully Paid

     defaulter_loan = df[df['loan_status'] == "Charged Off"]

58]: defaulter_loan['term'].value_counts()

58]:  36 months    3227
      60 months    2400
     Name: term, dtype: int64
```

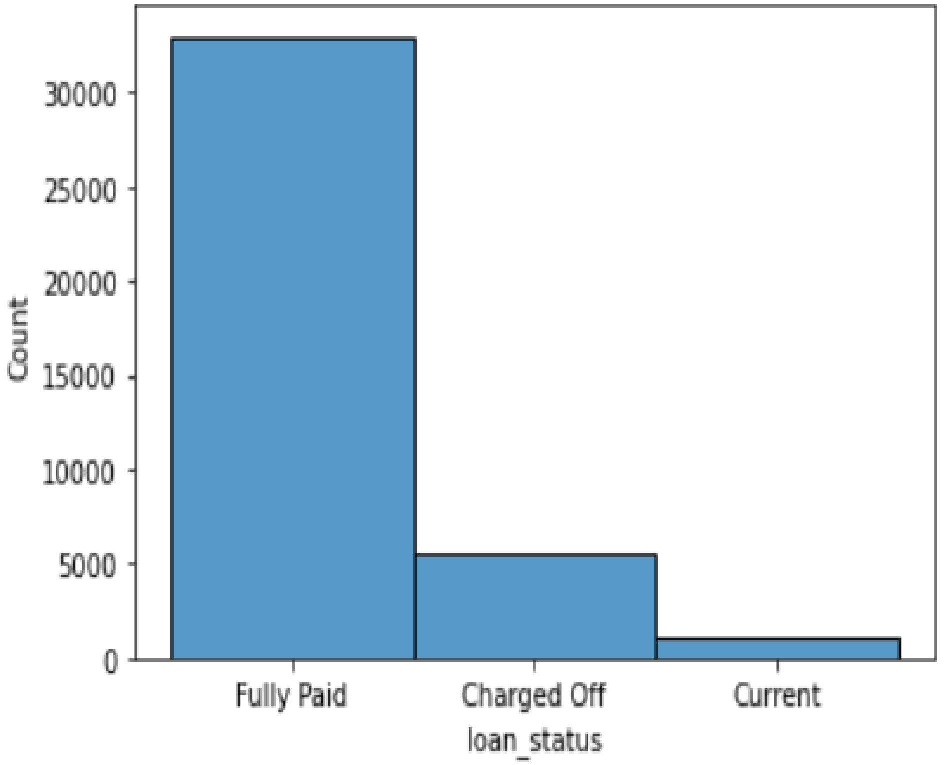
Univariate Analysis :

- ‘Uni’ means one and the data having only one variable is called as Univariate Analysis.
- The major reason for univariate analysis is to use the given data to describe it with different parameters.
- We’ll take data and do univariate analysis to summarise it and find some pattern in the data.

- The following image shows the univariate analysis for “loan_status “

```
sns.histplot(df['loan_status'])
```

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



Analysis :

- More than 30,000 applicants “ Fully paid” the amount.
- More than 5000 applicants “ Charged off ”
- Around 1000 applicants are currently paying.

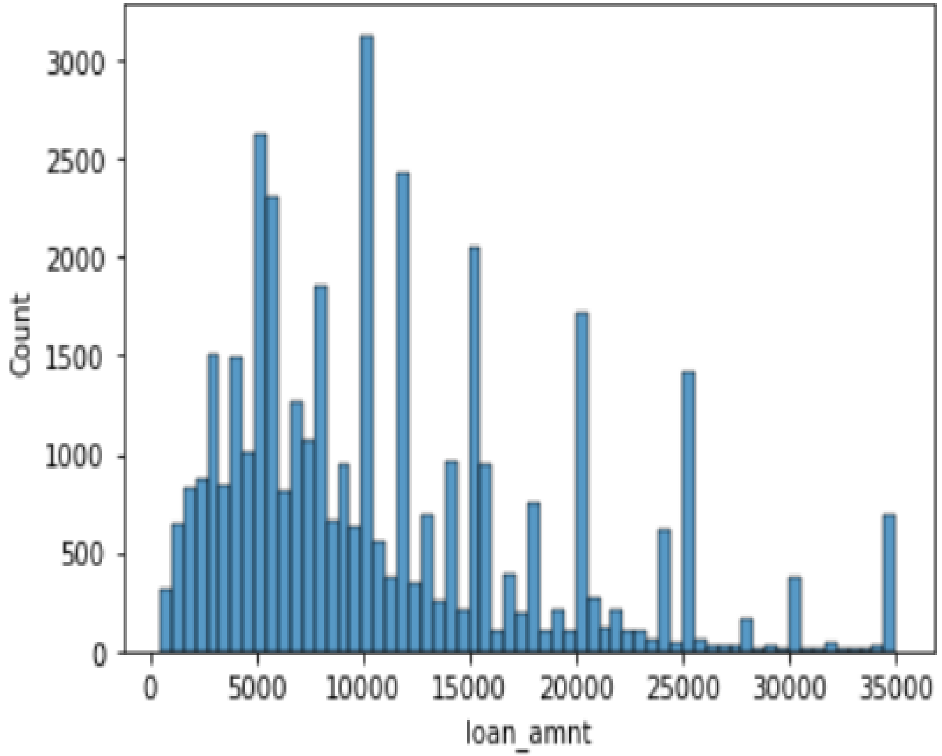
Distribution of loan_amnt

Analysis :

1. Loan amount of 10,000 have been availed by maximum number of people.
2. Less than 1000 consumers have availed loan amount of 35,000.

```
sns.histplot(df['loan_amnt'])
```

<AxesSubplot:xlabel='loan_amnt', ylabel='Count'>



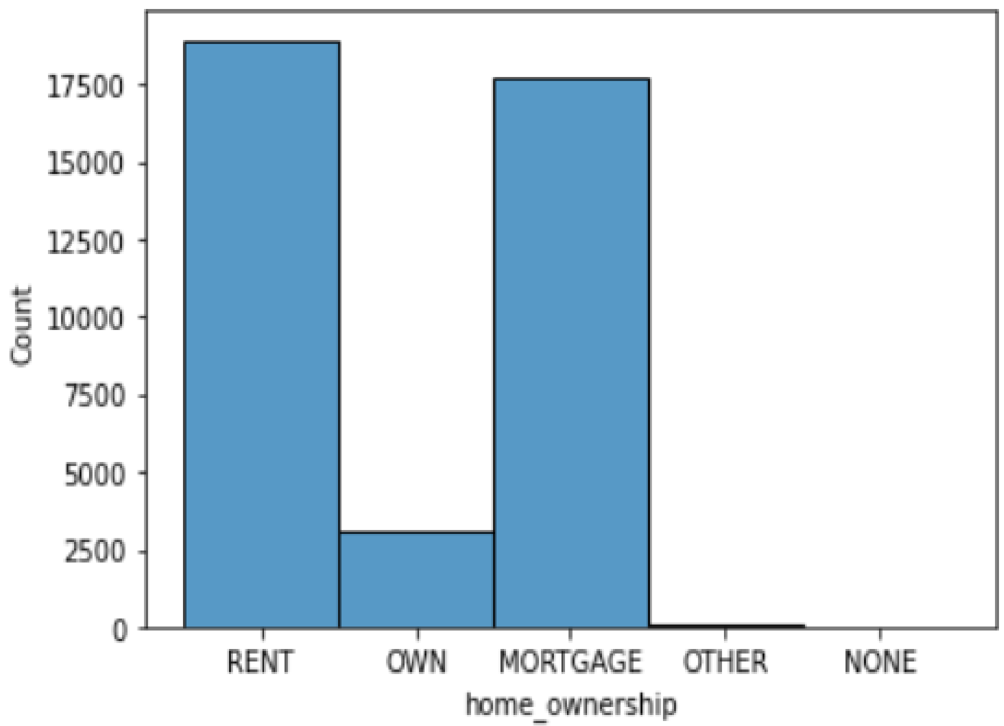
Distribution of home_ownership

Analysis :

1. Majorly there are three types of ownership, they are “RENT”, “OWN”, “MORTGAGE”

```python
sns.histplot(df['home_ownership'])
```

```
<AxesSubplot:xlabel='home_ownership', ylabel='Count'>
```



Bivariate Analysis :

- Bivariate analysis is the analysis of exactly two variables.
- Bivariate data is the data on each of two variables, where each value of one of the variables is paired with a value of the other variable.

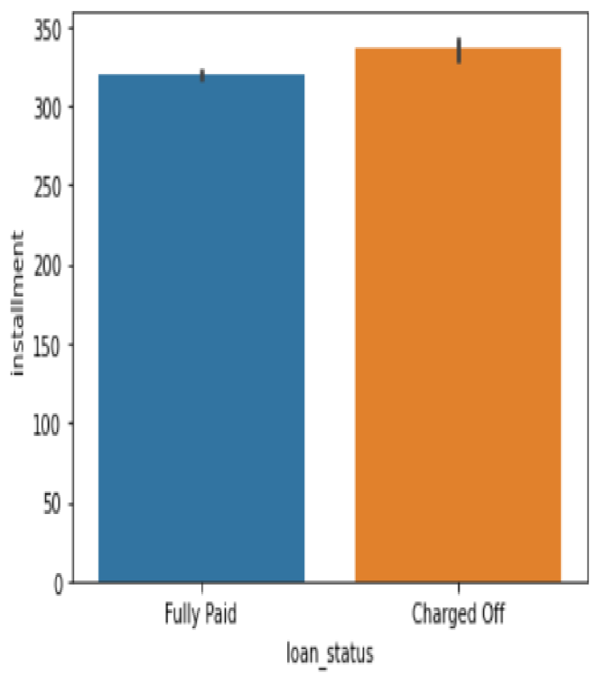
- The following image shows the “Bivariate Analysis” for “ Installment vs loan_status “
- Analysis :

1. Applicants who’s installment is more than 300, they are more likely to default.

```
sns.barplot(x=df['loan_status'], y=df['installment'])

# The Barplot indicates that person having installment amount more than 300 tends to default
```

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



loan_status vs term

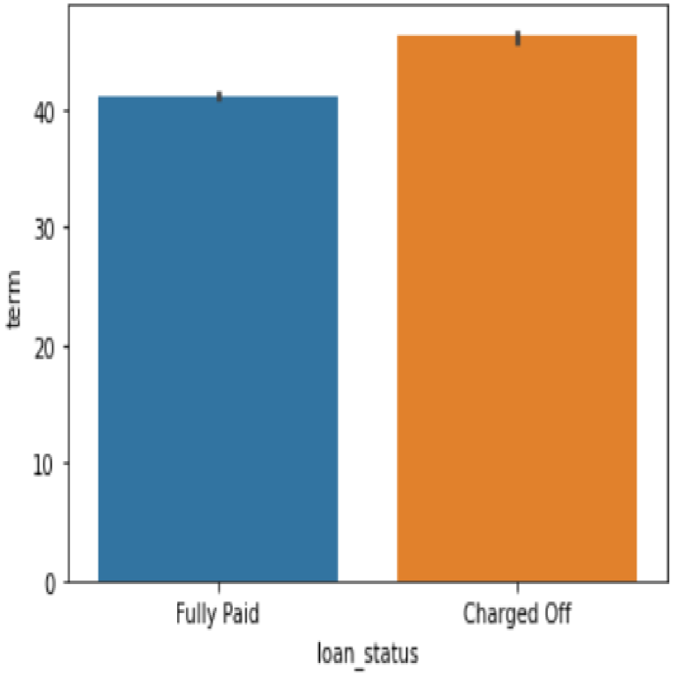
Analysis :

1. Consumer having term more than 40 months tends to default.

```
sns.barplot(x=df['loan_status'], y=df['term'].apply(lambda x: int(x.replace("months", ""))))

# The Barplot indicates that person having term more than 40 months tends to default
```

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



loan_status vs funded_amnt

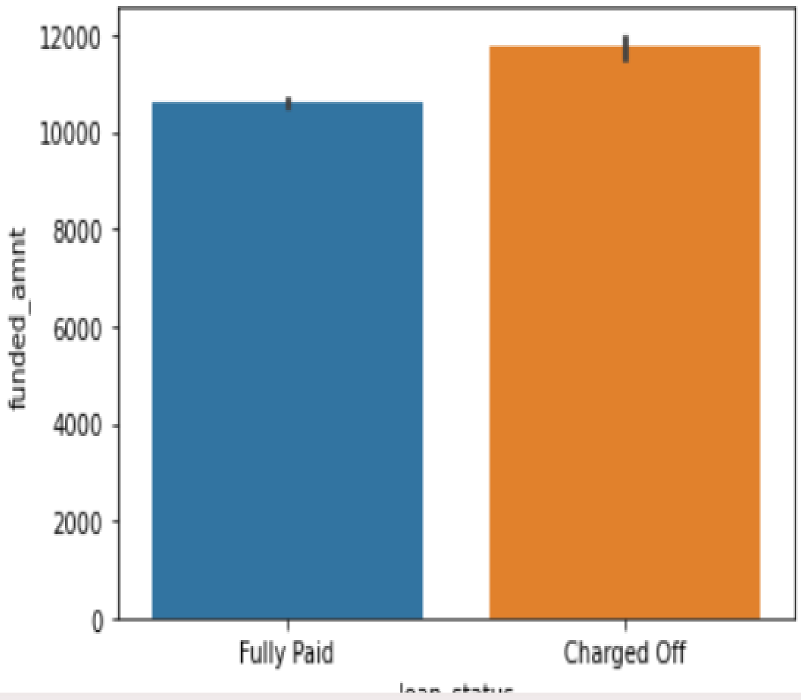
Analysis :

1. Consumer whose funded amount exceeds 10,000 are likely to default.

```
sns.barplot(x=df['loan_status'], y=df['funded_amnt'])


# Whenver the funded_amnt is more than 10k the customer tends to default
```

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



loan_status vs annual_inc

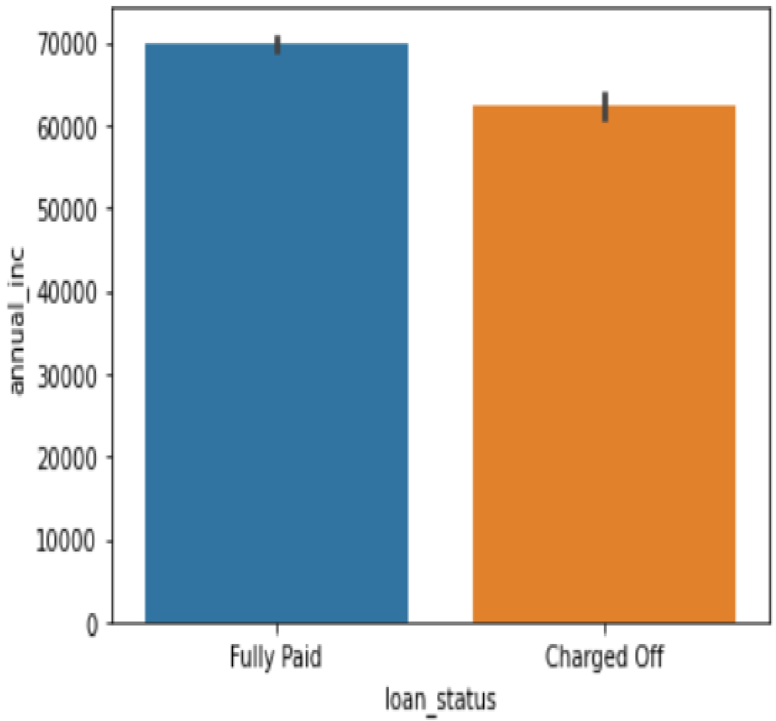
Analysis :

1. Consumers having annual income more than 60000 tends to default less.

```
sns.barplot(x=df['loan_status'], y=df['annual_inc'])

# Customer having an annual income more than 60000 tends to default less
```

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



Summary :

1. In our analysis we found that applicants whose annual income is more than 60,000 they are less likely to charge off.
2. Consumers who have term availed for 40 or more months they are more likely to charge off.
3. Customers who loan amount if funded more than 10,000 are more likely to charge off.
4. Majority of people who have availed from “C. A.” state are more likely to pay full loan.
5. Consumers who is having more than 10 years of experience they are more likely to repay full loan.

Thank You