1. Introduction
    a. Business Objectives
    b. Problem Statement
2. Data Understanding
    a. Data Cleanup
    b. Data Preprocessing
3. Exploratory Data Analysis & Data Visualizations
    a. Univariate Analysis
    b. Derived Metrics
    c. Bivariate Analysis
4. Insights and Recommendations
    a. Summary of Findings
    b. Identification of Driving Factors / Recommendations for Risk Assessment

# 1. Introduction

## a. Business Objectives

I. This company is the largest online loan marketplace, facilitating personal loans, business loans, and financing of medical procedures. Borrowers can easily access lower interest rate loans through a fast online interface.

II. Like most other lending companies, lending loans to 'risky' applicants is the largest source of financial loss (called credit loss). The credit loss is the amount of money lost by the lender when the borrower refuses to pay or runs away with the money owed. In other words, borrowers who default cause the largest amount of loss to the lenders. In this case, the customers labelled as 'charged-off' are the 'defaulters'.

III. If one is able to identify these risky loan applicants, then such loans can be reduced thereby cutting down the amount of credit loss. Identification of such applicants using EDA is the aim of this case study.

IV. In other words, the company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default. The company can utilize this knowledge for its portfolio and risk assessment.

## b. Problem Statement

I. When the company receives a loan application, the company has to make a decision for loan approval based on the applicant's profile.

II. Two types of risks are associated with the bank's decision:

- If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company
- If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company

III. The data provided contains the information about past loan applicants and whether they 'defaulted' or not. The aim is to identify patterns which indicate if a person is likely to default, which may be used for taking actions such as denying the loan, reducing the amount of loan, lending (to risky applicants) at a higher interest rate, etc.

## 2. Data Understanding

Data analysis sets the groundwork and foundation for the entire process of analysis. It gives you insights into the structure, quality and limitations of the data which helps you make informed decisions during data cleaning and pre-processing activity.

### a. Data Cleanup

- Data cleanup is a critical step for data analysis as we address errors, missing values, inconsistencies etc., and enhances the data quality making it more reliable.
- Below are some steps that were performed as part of data cleanup for this analysis.

I. Import relevant libraries to load the data

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

II. Load the data into a dataframe

```
loan_df = pd.read_csv('loan/loan.csv')
print(loan_df)
print(loan_df.shape)
```

```
            id  member_id  loan_amnt  funded_amnt  funded_amnt_inv  \
0      1077501    1296599       5000         5000          4975.00
1      1077430    1314167       2500         2500          2500.00
2      1077175    1313524       2400         2400          2400.00
3      1076863    1277178      10000        10000         10000.00
4      1075358    1311748       3000         3000          3000.00
...        ...        ...        ...          ...              ...
39712    92187      92174       2500         2500          1075.00
39713    90665      90607       8500         8500           875.00
39714    90395      90390       5000         5000          1325.00
39715    90376      89243       5000         5000           650.00
39716    87023      86999       7500         7500           800.00

             term int_rate  installment grade sub_grade  ...  \
0       36 months   10.65%       162.87     B        B2  ...
1       60 months   15.27%        59.83     C        C4  ...
2       36 months   15.96%        84.33     C        C5  ...
3       36 months   13.49%       339.31     C        C1  ...
4       60 months   12.69%        67.79     B        B5  ...
...           ...      ...          ...   ...       ...  ...
39712   36 months    8.07%        78.42     A        A4  ...
39713   36 months   10.28%       275.38     C        C1  ...
39714   36 months    8.07%       156.84     A        A4  ...
39715   36 months    7.43%       155.38     A        A2  ...
39716   36 months   13.75%       255.43     E        E2  ...
```

III. Drop all columns where all values are Null / NA

```
# Drop columns with all NA values
loan_df_1 = loan_df.dropna(axis=1, how='all')
```

IV. Drop all columns where a single / same value is present for all entries.

```
# Drop columns with the same value for all entries
loan_df_1 = loan_df_1.loc[:, loan_df_1.nunique() > 1]
```

V. Drop all columns where more than 90% of the values are missing data

```
#List columns where all data is missing
missing_cols = loan_df_1.columns[100*(loan_df_1.isnull().sum()/len(loan_df_1)) > 90]
missing_cols
```

VI. Steps 3,4,5 helps to reduce the data to be analyzed by eliminating all columns with redundant and missing data and only retaining the ones with valid data.

```
# Print the remaining columns
print(loan_df_1.columns.tolist())
print(loan_df_1.shape)
```

```
['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'em
p_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'url', 'desc', 'purpos
e', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq', 'm
ths_since_last_record', 'open_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'out_prncp', 'out_prncp_inv', 'total_pym
nt', 'total_pymnt_inv', 'total_rec_prncp', 'total_rec_int', 'total_rec_late_fee', 'recoveries', 'collection_recovery_fee', 'las
t_pymnt_d', 'last_pymnt_amnt', 'next_pymnt_d', 'last_credit_pull_d', 'pub_rec_bankruptcies']
(39717, 48)
```

VII. Once we cleanup the columns data, we look into the rows and identify all rows where more than 90% of the data is missing and drop those records / rows

```
# Calculate the percentage of missing values in each row
missing_percentages_rows = loan_df_1.isnull().mean(axis=0) * 100

# Filter rows where more than 90% of data is missing
rows_to_drop = missing_percentages_rows[missing_percentages_rows > 90].index

# Drop the rows from the DataFrame
loan_df_1 = loan_df_1.drop(rows_to_drop)
loan_df_1.shape
```

```
(39717, 46)
```

VIII. Post data cleanup, we get a more refined dataset to work with.

```
In [7]: loan_df_1.head()
```

Out[7]:

| | id | member_id | loan_amnt | funded_amnt | funded_amnt_inv | term | int_rate | installment | grade | sub_grade | ... | total_pymnt_inv | total_rec_prncp | total_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1077501 | 1296599 | 5000 | 5000 | 4975.00 | 36 months | 10.65% | 162.87 | B | B2 | ... | 5833.84 | 5000.00 | |
| 1 | 1077430 | 1314167 | 2500 | 2500 | 2500.00 | 60 months | 15.27% | 59.83 | C | C4 | ... | 1008.71 | 456.46 | |
| 2 | 1077175 | 1313524 | 2400 | 2400 | 2400.00 | 36 months | 15.96% | 84.33 | C | C5 | ... | 3005.67 | 2400.00 | |
| 3 | 1076863 | 1277178 | 10000 | 10000 | 10000.00 | 36 months | 13.49% | 339.31 | C | C1 | ... | 12231.89 | 10000.00 | 2 |
| 4 | 1075358 | 1311748 | 3000 | 3000 | 3000.00 | 60 months | 12.69% | 67.79 | B | B5 | ... | 3513.33 | 2475.94 | 1 |

5 rows × 46 columns

▪

```
In [8]: # List of columns to work with
        print(loan_df_1.columns.tolist())

['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term', 'int_rate', 'installment', 'grade', 'sub_grade', 'em
p_title', 'emp_length', 'home_ownership', 'annual_inc', 'verification_status', 'issue_d', 'loan_status', 'url', 'desc', 'purpos
e', 'title', 'zip_code', 'addr_state', 'dti', 'delinq_2yrs', 'earliest_cr_line', 'inq_last_6mths', 'mths_since_last_delinq', 'o
pen_acc', 'pub_rec', 'revol_bal', 'revol_util', 'total_acc', 'out_prncp', 'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv', 't
otal_rec_prncp', 'total_rec_int', 'total_rec_late_fee', 'recoveries', 'collection_recovery_fee', 'last_pymnt_d', 'last_pymnt_am
nt', 'last_credit_pull_d', 'pub_rec_bankruptcies']
```

▪

## b. Data Preprocessing

- Data preprocessing involves converting categorical variables into numeric forms or marking them with labels for ease of analysis and ease of plotting graphs to get a better understanding of data.
- Below are some steps which were performed as part of data pre-processing for this analysis.
  - I. Categorical Values to Numeric data mapping
    - Mapping loan status to numeric values

```
#Numeric mapping for loan status for ease of plotting on graphs
status_mapping = {
    'Fully Paid': 0,
    'Charged Off': 1,
    'Current': 2
}
loan_df_1['loan_status_numeric'] = loan_df_1['loan_status'].map(status_mapping)
```

  - II. Converting “term” data from “36 months”, “60 months” to numeric form 36 and 60

```
In [10]: loan_df_1['term'].value_counts()

Out[10]:  36 months    29096
          60 months    10621
         Name: term, dtype: int64

In [11]: # Remove "months" and convert to integer value for ease of plotting graphs
         loan_df_1['term'] = loan_df_1['term'].str.replace(' months', '').astype(int)
         loan_df_1['term'].value_counts()

Out[11]: 36    29096
         60    10621
         Name: term, dtype: int64
```

III. Converting interest rate (int_rate), Revolving line utilization (revolve_util) and employment length (emp_length) to float values

```
In [12]: # Remove "%" and convert to float
         loan_df_1['int_rate'] = loan_df_1['int_rate'].str.replace('%', '').astype(float)
         loan_df_1['int_rate'].value_counts()

Out[12]: 10.99    956
         13.49    826
         11.49    825
         7.51     787
         7.88     725
                 ...
         17.54      1
         17.44      1
         20.52      1
         24.59      1
         17.34      1
         Name: int_rate, Length: 371, dtype: int64
```

▪

```
In [71]: # Remove "%" and convert to float
         loan_df_1['revol_util'] = loan_df_1['revol_util'].str.replace('%', '').astype(float)
         loan_df_1['revol_util'].value_counts()

Out[71]: 0.00     977
         0.20      63
         63.00     62
         40.70     58
         66.70     58
                 ...
         70.94      1
         8.49       1
         77.63      1
         10.17      1
         24.66      1
         Name: revol_util, Length: 1089, dtype: int64
```

▪

```
In [13]: # Remove "%" and convert to float
         loan_df_1['emp_length_num'] = loan_df_1['emp_length']
         loan_df_1['emp_length_num'] = loan_df_1['emp_length_num'].str.replace(' years', '')
         loan_df_1['emp_length_num'] = loan_df_1['emp_length_num'].str.replace('< 1 year', '0.9')
         loan_df_1['emp_length_num'] = loan_df_1['emp_length_num'].str.replace('1 year', '1')
         loan_df_1['emp_length_num'] = loan_df_1['emp_length_num'].str.replace('10\+', '11')
         loan_df_1['emp_length_num'] = loan_df_1['emp_length_num'].astype(float)
         loan_df_1['emp_length_num'].value_counts()

Out[13]: 11.00    8879
         0.90     4583
         2.00     4388
         3.00     4095
         4.00     3436
         5.00     3282
         1.00     3240
         6.00     2229
         7.00     1773
         8.00     1479
         9.00     1258
         Name: emp_length_num, dtype: int64
```

▪

# 3. Exploratory Data Analysis (EDA) & Data Visualizations

EDA helps with understanding the data, identifying patterns and relationships between variables in the data, checking quality of data, identifying outliers in the data etc.

## a. Univariate Analysis

Performing univariate analysis involves analyzing each individual variable to understand the summary statistics using functions like head(), describe(), info() etc., and also identifying outliers. Some examples of Univariate analysis are shared below:

I. Removing outliers in annual income

```
In [16]: #Starting with Annual income
         loan_df_1['annual_inc'].describe()

Out[16]: count      39717.00
         mean       68968.93
         std        63793.77
         min         4000.00
         25%        40404.00
         50%        59000.00
         75%        82300.00
         max      6000000.00
         Name: annual_inc, dtype: float64
```

- 

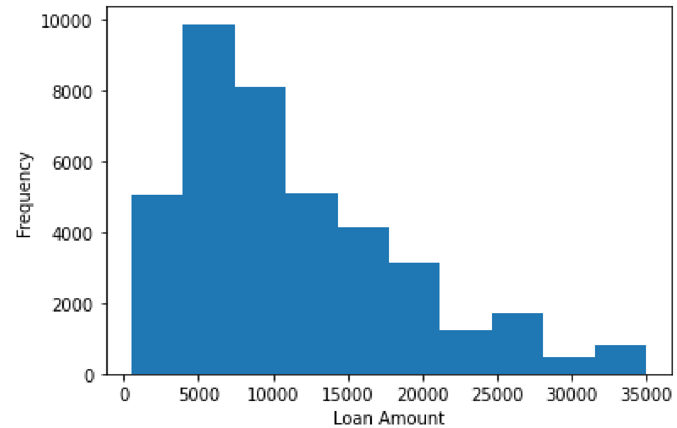
```
In [17]: #Remove Outliers
         print((loan_df_1['annual_inc'] == 6000000).sum())
         loan_df_1 = loan_df_1[loan_df_1['annual_inc'] != 6000000]
         loan_df_1['annual_inc'].describe()

         1

Out[17]: count      39716.00
         mean       68819.59
         std        56426.82
         min         4000.00
         25%        40403.00
         50%        59000.00
         75%        82300.00
         max      3900000.00
         Name: annual_inc, dtype: float64
```

- 

## II. Loan Amount

```
print(loan_df_1['loan_amnt'].describe())
plt.hist(loan_df_1['loan_amnt'])
plt.xlabel('Loan Amount')
plt.ylabel('Frequency')
plt.show()
```

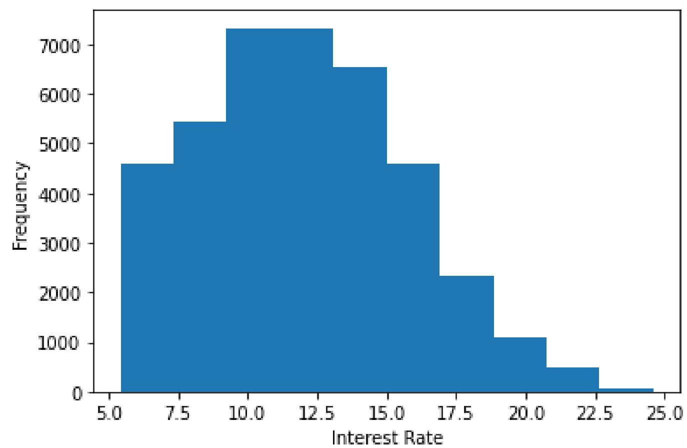
```
count    39716.00
mean     11219.60
std       7456.70
min        500.00
25%       5500.00
50%      10000.00
75%      15000.00
max      35000.00
Name: loan_amnt, dtype: float64
```




## III. Interest Rate

```
print(loan_df_1['int_rate'].describe())
plt.hist(loan_df_1['int_rate'])
plt.xlabel('Interest Rate')
plt.ylabel('Frequency')
plt.show()
```

```
count    39716.00
mean        12.02
std          3.72
min          5.42
25%          9.25
50%         11.86
75%         14.59
max         24.59
Name: int_rate, dtype: float64
```



## b. Derived Metrics

Derived metrics is a data which is introduced newly to the dataset using existing variables. This can be done by performing calculations or data transformation etc. Derived metrics helps us gain to view the data from a different perspective giving new insights. Some examples of derived metrics which were calculated for this analysis are shared below:

I. Identify all records which fall under 100000 for Annual income category and mark them as True for low income as this seems a possible factor to assess for loan credibility
II. Total credit revolving balance / Annual income. Good candidate for loan credibility factor. If high then providing loan to such candidates can be risky
III. Annual interest amount
IV. Interest to income percentage

```
In [14]: # Identify all records which fall under 100000 for Annual income category and mark them as True for low income as this seems a po
         loan_df_1['low_income'] = loan_df_1['annual_inc'] < 100000

In [15]: # Total credit revolving balance / Annual income. Good candidate for loan credibility factor. If high then providing loan to such
         loan_df_1['TRB_AI'] = (loan_df_1['revol_bal'] // loan_df_1['annual_inc']) * 100
         loan_df_1['TRB_AI'] = loan_df_1['TRB_AI'].astype(int)

In [82]: #Annual interest amount
         loan_df_1['annual_int_amnt'] = loan_df_1['loan_amnt'] * (loan_df_1['int_rate'] / 100)

In [86]: #Interest to income percentage
         loan_df_1['interest_to_income'] = (loan_df_1['annual_int_amnt'] / loan_df_1['annual_inc']) * 100
```

V.

## c. Bivariate Analysis

Bivariate Analysis is used to explore relationships between variables in the dataset.

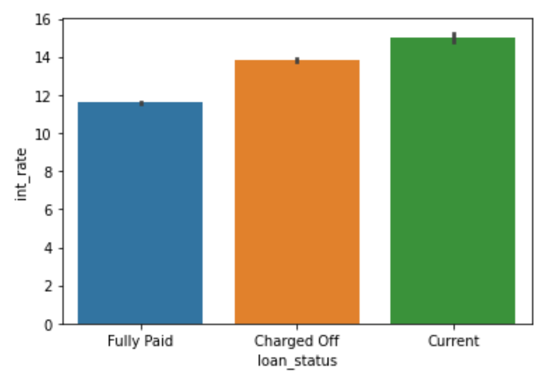
NOTE: For ease of analysis and understanding, I have divided the dataset into 2 sets

- charged_off_df - which contains all the data for customers marked as “Charged Off”
- fully_paid_df – which contains all the data for customers marked as “Fully Paid”

```
In [87]: # Before starting, Let's create separate datasets for Fully paid and Charg
         charged_off_df = loan_df_1[loan_df_1['loan_status'] == "Charged Off"]
         charged_off_df.shape

Out[87]: (5627, 52)

In [88]: fully_paid_df = loan_df_1[loan_df_1['loan_status'] == "Fully Paid"]
         fully_paid_df.shape

Out[88]: (32949, 52)
```

- Below is a list of factors / variables which indicate risky applicants who are prone to default on loans.
  I. Interest rate

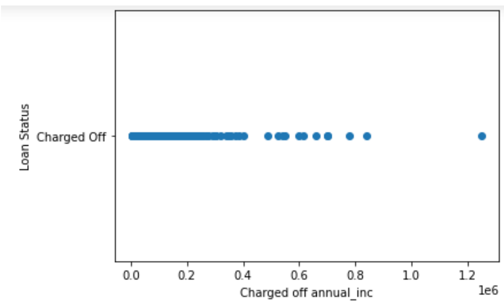
```
# Interest rate vs Loan Status
sns.barplot(x='loan_status',y='int_rate',data=loan_df_1)
plt.show()
#As we can see in below graph, Interest rate seems to be playing a signoficant role in case of defaulters.
```

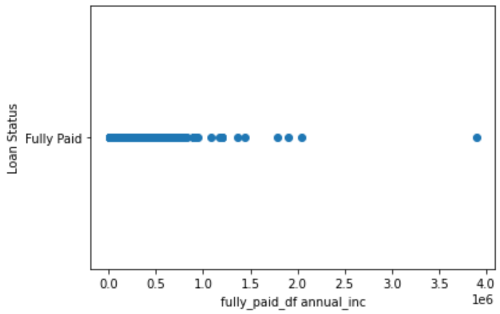


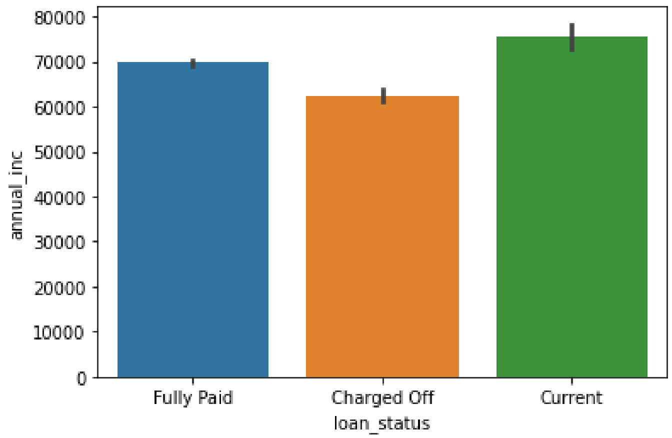

▪

## II. Annual income

```
# Annual income
plt.scatter(charged_off_df['annual_inc'], charged_off_df['loan_status'])
plt.xlabel('Charged off annual_inc')
plt.ylabel('Loan Status')
plt.show()
plt.scatter(fully_paid_df['annual_inc'], fully_paid_df['loan_status'])
plt.xlabel('fully_paid_df annual_inc')
plt.ylabel('Loan Status')
plt.show()
sns.barplot(x='loan_status',y='annual_inc',data=loan_df_1)
plt.show()
#There's a significant dip in annual income for Charged off customers.
```



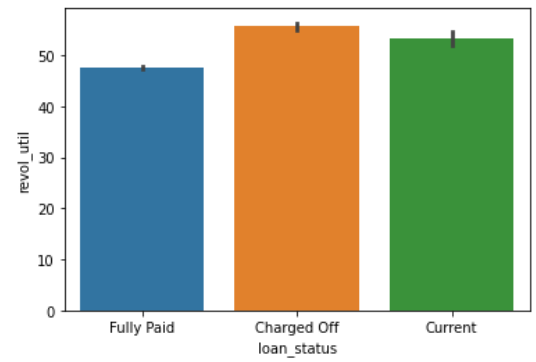



III. Revolving line utilization rate

```
print(f"Charged Off Data: \n{charged_off_df['revol_util'].describe()}")
print(f"Fully Paid Data: \n{fully_paid_df['revol_util'].describe()}")
sns.barplot(x='loan_status',y='revol_util',data=loan_df_1)
plt.show()
#Significant increase in case of defaulters
```

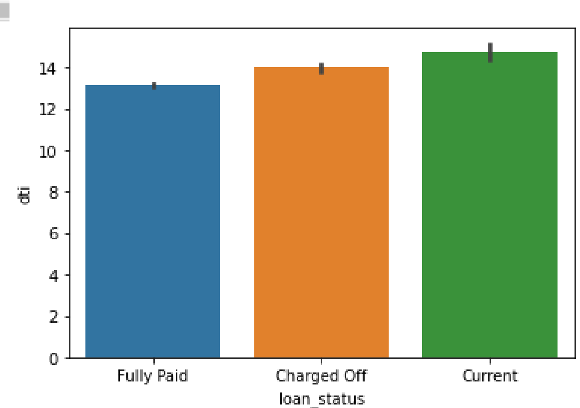
```
Charged Off Data:
count    5611.00
mean       55.57
std        27.91
min         0.00
25%        34.40
50%        58.40
75%        79.00
max        99.90
Name: revol_util, dtype: float64
Fully Paid Data:
count    32915.00
mean        47.53
std         28.28
min          0.00
25%         23.90
50%         47.60
75%         70.80
max         99.90
Name: revol_util, dtype: float64
```




IV. DTI - Debt to income - There seems to be a slight increase in DTI for charged off customers. Since the number is not significantly large, we can explore for other factors which do make a significant impact.

```
print(f"Charged Off Data: \n{charged_off_df['dti'].describe()}")
print(f"Fully Paid Data: \n{fully_paid_df['dti'].describe()}")
sns.barplot(x='loan_status',y='dti',data=loan_df_1)
plt.show()
# There seems to be a slight increase in DTI for charged off cust
```

```
Charged Off Data:
count    5627.00
mean       14.00
std         6.59
min         0.00
25%         9.05
50%        14.29
75%        19.29
max        29.85
Name: dti, dtype: float64
Fully Paid Data:
count    32949.00
mean        13.15
std          6.68
min          0.00
25%          7.98
50%         13.20
75%         18.39
max         29.99
Name: dti, dtype: float64
```

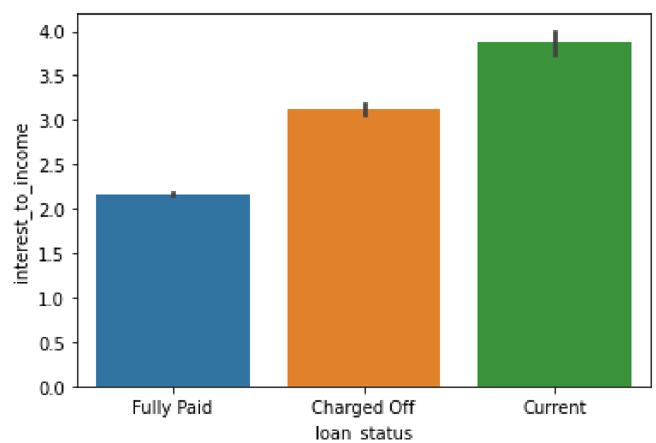


V. Interest to income - Derived column – Ratio of interest expenses to income – Higher the number, more likely for customer to default.

```
#Interest to income percentage
loan_df_1['interest_to_income'] = (loan_df_1['annual_int_amnt'] / loan_df_1['annual_inc']) * 100
```

```
print(f"Charged Off Data: \n{charged_off_df['interest_to_income'].describe()}")
print(f"Fully Paid Data: \n{fully_paid_df['interest_to_income'].describe()}")
sns.barplot(x='loan_status',y='interest_to_income',data=loan_df_1)
plt.show()
```

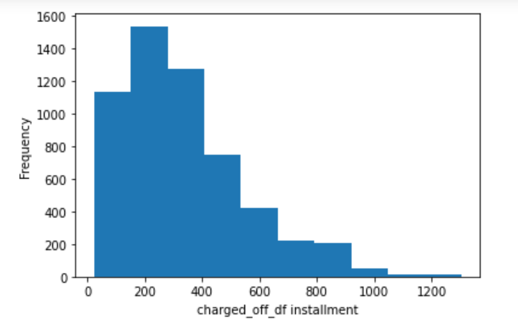
```
Charged Off Data:
count   5627.00
mean       3.12
std        2.18
min        0.07
25%        1.41
50%        2.60
75%        4.32
max       12.81
Name: interest_to_income, dtype: float64
Fully Paid Data:
count   32949.00
mean        2.17
std         1.66
min         0.00
25%         0.94
50%         1.72
75%         2.95
max        14.12
Name: interest_to_income, dtype: float64
```

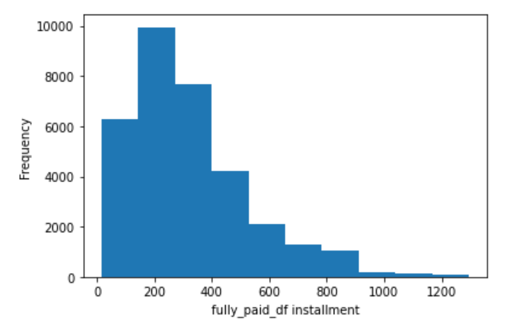


VI. Univariate analysis on individual datasets for installment – If we compare the summary stats we can see that the numbers are considerably high for Charged off customers.

```
# installment
print(f"Charged Off Data: \n{charged_off_df['installment'].describe()}")
print(f"Fully Paid Data: \n{fully_paid_df['installment'].describe()}")

plt.hist(charged_off_df['installment'])
plt.xlabel('charged_off_df installment')
plt.ylabel('Frequency')
plt.show()

plt.hist(fully_paid_df['installment'])
plt.xlabel('fully_paid_df installment')
plt.ylabel('Frequency')
plt.show()
```

```
Charged Off Data:
count    5627.00
mean      336.18
std       217.05
min        22.79
25%       168.56
50%       293.87
75%       457.84
max      1305.19
Name: installment, dtype: float64
Fully Paid Data:
count    32949.00
mean       320.13
std        207.08
min         15.69
25%        165.27
50%        275.66
75%        420.74
max       1295.21
Name: installment, dtype: float64
```



# 4. Insights and Recommendations

## a. Summary of findings

I. After eliminating outliers, redundant, inconsistent and null data and performing EDA we have seen some factors which impact the Loan status.
II. We have identified that a higher **interest rate** plays a role in customers turning into defaulters.
III. After splitting the dataset into 2 and analyzing the data for **Annual income**, It can clearly be seen that low annual income plays a significant role in customers turning into loan defaulters.
IV. Revolving line utilization rate plays a significant impact, Higher value increases risk of customers turning into defaulters.
V. A small increase can be noticed in the DTI value for charged off customers but the difference / increase is not very significant compared to other variables.
VI. Interest to income which is a derived column plays a very good role in identifying customers who are prone to be defaulters.
VII. Installment can also be considered as a factor as it can be seen to be playing a small role in identifying loan defaulters.

## b. Identification of Driving Factors / Recommendations for Risk Assessment

I. Annual income

- Consider customers with above average annual income. (In this case more than 70000)

II. Revolving line utilization rate

- Consider customers with below average revol_util rate. (In this case below 48%)

III. Interest to income

- Consider customers with below average or STD rate. (In this case below 2.1 % / 1.6 % respectively)

IV. Loan status numeric mapping

- Fully Paid: 0, Charged Off: 1, Current: 2