Use of Exploratory Data Analysis techniques to perform Credit Risk Analytics for LendingClub

BY

AGNEL RAJ

AKASHDEEP HOWLADAR

LendingCLub
Company Information

Lending Club, a peer-to-peer lending company based in the United States, was reviewed. Here, investors fund potential borrowers and earn profits based on the risks associated with the borrower's credit score. The company serves as a bridge between investors and borrowers.

Problem statement

Introduction

To develop a basic understanding of risk analytics in banking and financial services and understand how data is used to minimise the risk of losing money while lending to customers.

Business Risks

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.

Problem statement

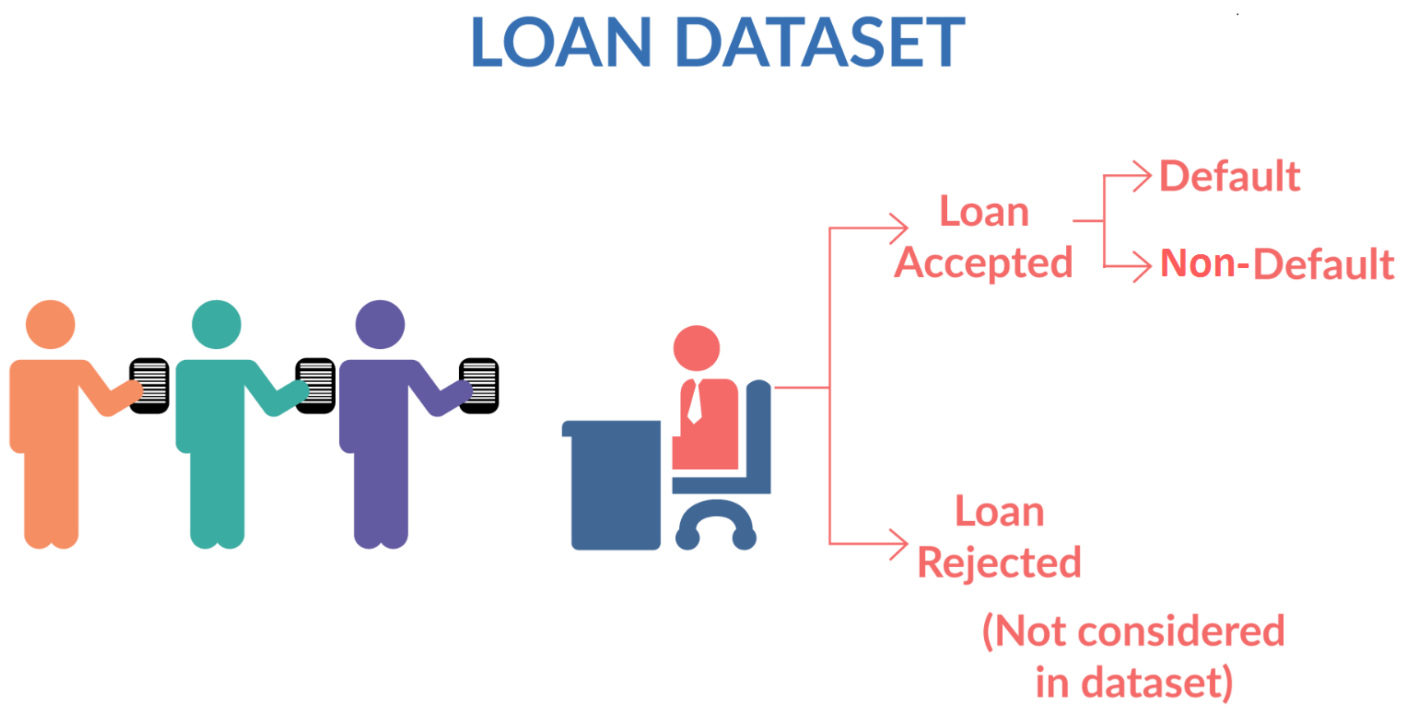
Business Objectives

The company wants to understand the driving factors (or driver variables) behind loan default, i.e. the variables which are strong indicators of default.

Aim of this Case Study

To identify "risky applicants" so that loans can be reduced thereby cutting down the amount of credit loss. [Credit loss it the amount of money lost by the lender when the borrower refuses to pay or runs away with the money that needs to be paid to the company]

Diagram as per problem statement



To understand the meaning of variables the data dictionary was used

acc_now_delinq	The number of accounts on which the borrower is now delinquent.
acc_open_past_24mths	Number of trades opened in past 24 months.
addr_state	The state provided by the borrower in the loan application
all_util	Balance to credit limit on all trades
annual_inc	The self-reported annual income provided by the borrower during registration.
annual_inc_joint	The combined self-reported annual income provided by the co-borrowers during registration

Understanding Data

- The code snippets in subsequent slides provide various methods to explore and understand the structure and content of a DataFrame named `loan_df`.
- First, is configured to display all columns without truncation. Then the `shape` method is used to retrieve the dimensions of the DataFrame, giving the total count of rows and columns. The `info()` method provides details about the DataFrame, such as data types and non-null counts. `loan_df.columns` lists all column names.
- The `describe()` method generates descriptive statistics for numeric columns. `loan_df.dtypes` returns the data type of each column.
- Lastly, `loan_df.isnull().sum().sum()` calculates the total number of missing values across all columns, which totals 2,263,366 missing entries, indicating areas that may require data cleaning or imputation.
- Many other steps are carried out till cell no. 146 which are commented in the python notebook

Understanding Data

```
# Show all the columns in a data frame
pd.set_option('display.max_columns', None)
[135]
```

```
# Get the number of rows and columns in a data frame using shape method

loan_df.shape
[136]
```

```
(39717, 111)
```

Total number of Rows: 39717
Total number of columns: 111

```
# Dataframe details using info object

loan_df.info()
[137]
```

```
<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
```

```
# List of columns in dataframe
loan_df.columns
[138]
```

```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
```

```
# loading the data from csv file into data frame
loan_df = pd.read_csv(r'C:\loan.csv')
```

Printing the first 5 rows of the data frame

```
loan_df.head()
```

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_gra
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	162.87	B	
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	59.83	C	
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	84.33	C	
3	1076863	1277178	10000	10000	10000.0	36 months	13.49%	339.31	C	

Data Cleaning & Manipulation

Step 1: Dropping the (54) Columns with all null values

[147]
```
loan_df.dropna(axis = 1, how = 'all', inplace = True)
```

[148]
```
# Loading the dataframe after dropping the columns
loan_df.head()
```

...

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp
0	1077501	1296599	5000	5000	4975.0	36 months	10.65%	162.87	B	B2	
1	1077430	1314167	2500	2500	2500.0	60 months	15.27%	59.83	C	C4	
2	1077175	1313524	2400	2400	2400.0	36 months	15.96%	84.33	C	C5	
3	1076863	1277178	10000	10000	10000.0	36	13.49%	339.31	C	C1	RESOU

Step 1: Dropping the (54) Columns with all null values

```
# Checking for rows and columns count

loan_df.shape # Returns 39717 rows and 57 columns
```

[149]

```
(39717, 57)
```

```
# Find all columns with single value
singlevaluedcol = loan_df.nunique()
print(singlevaluedcol[singlevaluedcol == 1])
```

[150]

```
pymnt_plan                    1
initial_list_status           1
collections_12_mths_ex_med    1
policy_code                   1
application_type              1
acc_now_delinq                1
chargeoff_within_12_mths      1
delinq_amnt                   1
tax_liens                     1
dtype: int64
```

Step 2:
Dropping columns with same values that is unique value count is 1

```
loan_df.drop(["pymnt_plan","initial_list_status","collections_12_mths_ex_med","policy_code","appl
```

[151]

```
# Checking for rows and columns count

loan_df.shape # Returns 39717 rows and 48 columns
```

[152]

```
(39717, 48)
```

```
##### Step 3: Checking columns with large percentage of null values and drop these columns
```

```
# Finding the percentage of missing values
100 * (loan_df.isnull().sum()/len(loan_df.index))
```

[153]

```
id                         0.000000
member_id                  0.000000
loan_amnt                  0.000000
funded_amnt                0.000000
funded_amnt_inv            0.000000
term                       0.000000
int_rate                   0.000000
installment                0.000000
grade                      0.000000
sub_grade                  0.000000
emp_title                  6.191303
```

Step 3:
Checking columns with large percentage of null values and drop these columns

```
# Finding the percentage of missing values
100 * (loan_df.isnull().sum()/len(loan_df.index))
```

```
id                  0.000000
member_id           0.000000
loan_amnt           0.000000
funded_amnt         0.000000
funded_amnt_inv     0.000000
term                0.000000
int_rate            0.000000
installment         0.000000
grade               0.000000
```

```
# Filter only columns having values greater than zero

missingvaluespercentage = 100 * (loan_df.isnull().sum()/len(loan_df.index))
print(missingvaluespercentage[missingvaluespercentage > 0.0])
```

```
emp_title                  6.191303
emp_length                 2.706650
desc                      32.585543
title                      0.027696
mths_since_last_delinq    64.662487
mths_since_last_record    92.985372
revol_util                 0.125891
last_pymnt_d               0.178765
next_pymnt_d              97.129693
last_credit_pull_d         0.005036
pub_rec_bankruptcies       1.754916
dtype: float64
```

```
# Dropping columns with high missing value percentage
loan_df.drop(["mths_since_last_delinq", "mths_since_last_record", "next_pymnt_d"], axis = 1, inplace = True)
```

```
# Removing desc column as it a free text data as per data dictionary
loan_df.drop(["desc"], axis = 1, inplace = True)

# Removing emp_title column which does not contribute any meaning on analysis and having 6% of missing values
loan_df.drop(["emp_title"], axis = 1, inplace = True)
```

Step 4:
Checking columns for the percentage of null values and fill missing values

```
[159]: # Check for the missing percentage of null values
       missingvaluespercentage = 100 * (loan_df.isnull().sum()/len(loan_df.index))
       print(missingvaluespercentage[missingvaluespercentage > 0.0])
```

```
emp_length              2.706650
revol_util              0.125891
pub_rec_bankruptcies    1.754916
dtype: float64
```

```
[160]: # Check the above column for values before filling missing/imputing values

       loan_df.emp_length.value_counts()
```

```
[160]: emp_length
       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: count, dtype: int64
```

```
[161]: loan_df.revol_util.value_counts()
```

```
[161]: revol_util
       0%       977
       0.20%     63
       63%       62
```

Step 4:
Checking columns for the percentage of null values and fill missing values

```
[162]: loan_df.pub_rec_bankruptcies.value_counts()

[162]: pub_rec_bankruptcies
       0.0    37339
       1.0     1674
       2.0        7
       Name: count, dtype: int64
```

```
[161]: loan_df.revol_util.value_counts()

[161]: revol_util
       0%       977
       0.20%     63
       63%       62
```

```
[163]: # Imputing missing values with most frequently found values as the percent
       records.
       # Using mode function

       loan_df.revol_util.fillna(loan_df.revol_util.mode()[0], inplace = True)
       loan_df.pub_rec_bankruptcies.fillna(loan_df.pub_rec_bankruptcies.mode()[0], inplace = True)
```

```
[164]: # Verifying for null values

       print(loan_df.revol_util.isna().sum())
       print(loan_df.pub_rec_bankruptcies.isna().sum())

       0
       0
```

```
[165]: # Checking for all the columns for missing values
       100 * (loan_df.isnull().sum()/len(loan_df.index)) # Missing percentage is zero as per the result
```

Step 5:
Checking columns that are not necessary and dropping them.

- member_id
- url
- zip_code

As these columns are having details corresponding to individual applicant and does not help in our analysis

[167]:
```
# Removing the above columns

loan_df.drop(["member_id", "url", "zip_code"], axis = 1, inplace = True)
```

Step 6:
Removing records that are not needed (Cleaning Rows)

[168]:
```
# Keeping only records with loan status as completed and charged off for our analysis as per the problem statement.
loan_df = loan_df[loan_df.loan_status != "Current"]
```

[169]:
```
# Checking the loan status values
loan_df.loan_status.unique()
```

[169]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

[170]:
```
# Check of the shape of the dataframe
loan_df.shape
```

[170]:
```
(38577, 37)
```

Step 7:
Correcting the datatype and value of columns that are invalid

[171]:
```python
# identify the column types
loan_df.dtypes
```

[171]:
```
id                       int64
loan_amnt                int64
funded_amnt              int64
funded_amnt_inv        float64
term                    object
int_rate                object
installment            float64
grade                   object
sub_grade               object
emp_length              object
home_ownership          object
annual_inc             float64
verification_status     object
issue_d                 object
loan_status             object
purpose                 object
addr_state              object
dti                    float64
delinq_2yrs              int64
earliest_cr_line        object
inq_last_6mths           int64
open_acc                 int64
pub_rec                  int64
revol_bal                int64
revol_util              object
```

[172]:
```python
# Analysing each column with the data type as object
# Check for term data
print(loan_df.term.unique())
```

```
[' 36 months' ' 60 months']
```

[173]:
```python
# Apply data correction on term data
loan_df['term'] = loan_df['term'].str.rstrip(' months').astype('int')
```

[174]:
```python
# Check for int_rate
print(loan_df.int_rate.unique())
```

```
['10.65%' '15.27%' '15.96%' '13.49%' '7.90%' '18.64%' '21.28%' '12.69%'
 '14.65%' '9.91%' '16.29%' '6.03%' '11.71%' '12.42%' '14.27%' '16.77%'
 '7.51%' '8.90%' '18.25%' '6.62%' '19.91%' '17.27%' '17.58%' '21.67%'
 '19.42%' '20.89%' '20.30%' '23.91%' '19.03%' '23.13%' '22.74%' '22.35%'
 '22.06%' '24.11%' '6.00%' '23.52%' '22.11%' '7.49%' '11.99%' '5.99%'
 '10.99%' '9.99%' '18.79%' '11.49%' '8.49%' '15.99%' '16.49%' '6.99%'
 '12.99%' '15.23%' '14.79%' '5.42%' '10.59%' '17.49%' '15.62%' '19.29%'
 '13.99%' '18.39%' '16.89%' '17.99%' '20.99%' '22.85%' '19.69%' '20.62%'
 '20.25%' '21.36%' '23.22%' '21.74%' '22.48%' '23.59%' '12.62%' '18.07%'
 '11.63%' '7.91%' '7.42%' '11.14%' '20.20%' '12.12%' '19.39%' '16.11%'
 '17.54%' '22.64%' '13.84%' '16.59%' '17.19%' '12.87%' '20.69%' '9.67%'
 '21.82%' '19.79%' '18.49%' '22.94%' '24.40%' '21.48%' '14.82%' '14.17%'
 '7.29%' '17.88%' '20.11%' '16.02%' '13.43%' '14.91%' '13.06%' '15.28%'
```

And several other steps up to cell 195

Step 8:
Identifying Outliers and removing those records

[196]:
```
# Check if there any outliers in annual_inc column

loan_df.annual_inc.describe().apply(lambda x: format(x, 'f'))
```

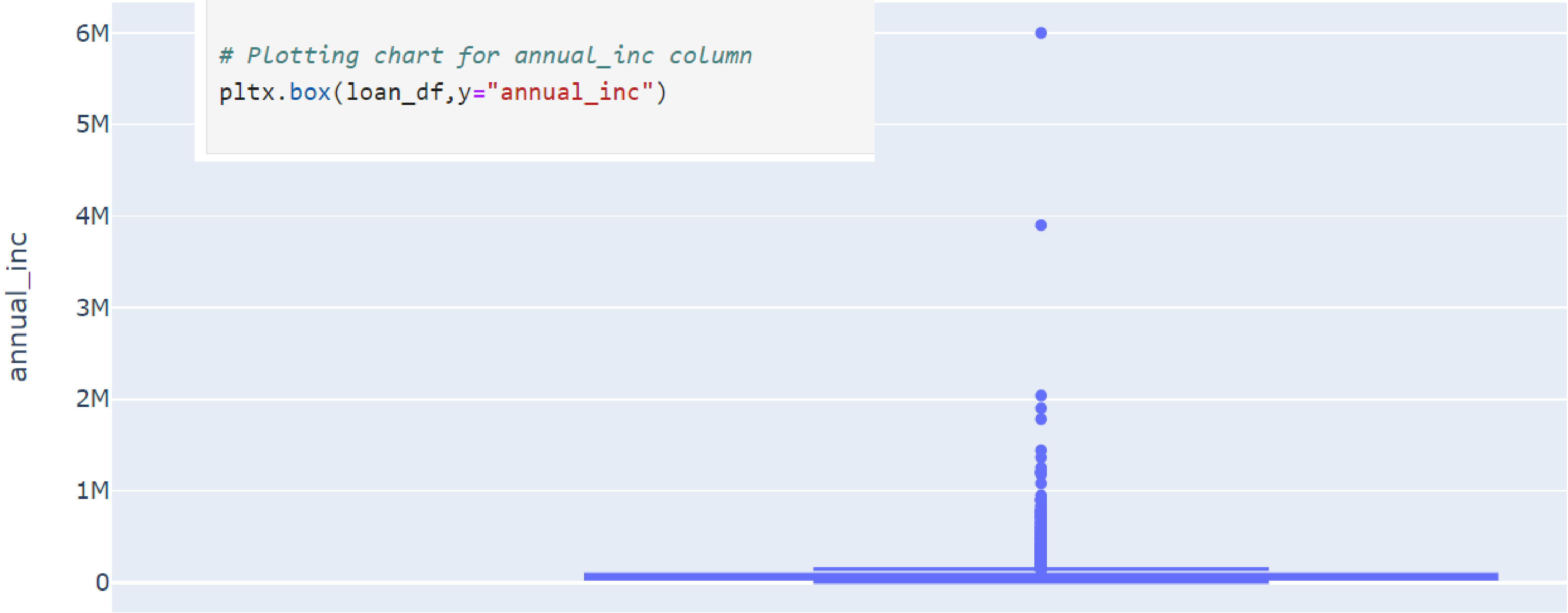
[196]:
```
count      38577.000000
mean       68777.973681
std        64218.681802
min         4000.000000
25%        40000.000000
50%        58868.000000
75%        82000.000000
max      6000000.000000
Name: annual_inc, dtype: object
```

[197]:
```
#using Plotly express for interactive charts

import plotly.express as pltx

# Plotting chart for annual_inc column
pltx.box(loan_df,y="annual_inc")
```

And several other steps up to cell 207

Step 8:
Identifying Outliers and removing those records

```python
#using Plotly express for interactive charts

import plotly.express as pltx

# Plotting chart for annual_inc column
pltx.box(loan_df,y="annual_inc")
```



And several other steps up to cell 207

Step 8:
Identifying Outliers and removing those records

```
[198]: # ...                                    s 145k an outlier since there is no continous distribution.
       # ...
       an...
       pr...
       17...
```

```
#using Plotly express for interactive charts

import plotly.express as pltx

# Plotting chart for annual_inc column
pltx.box(loan_df,y="annual_inc")
```

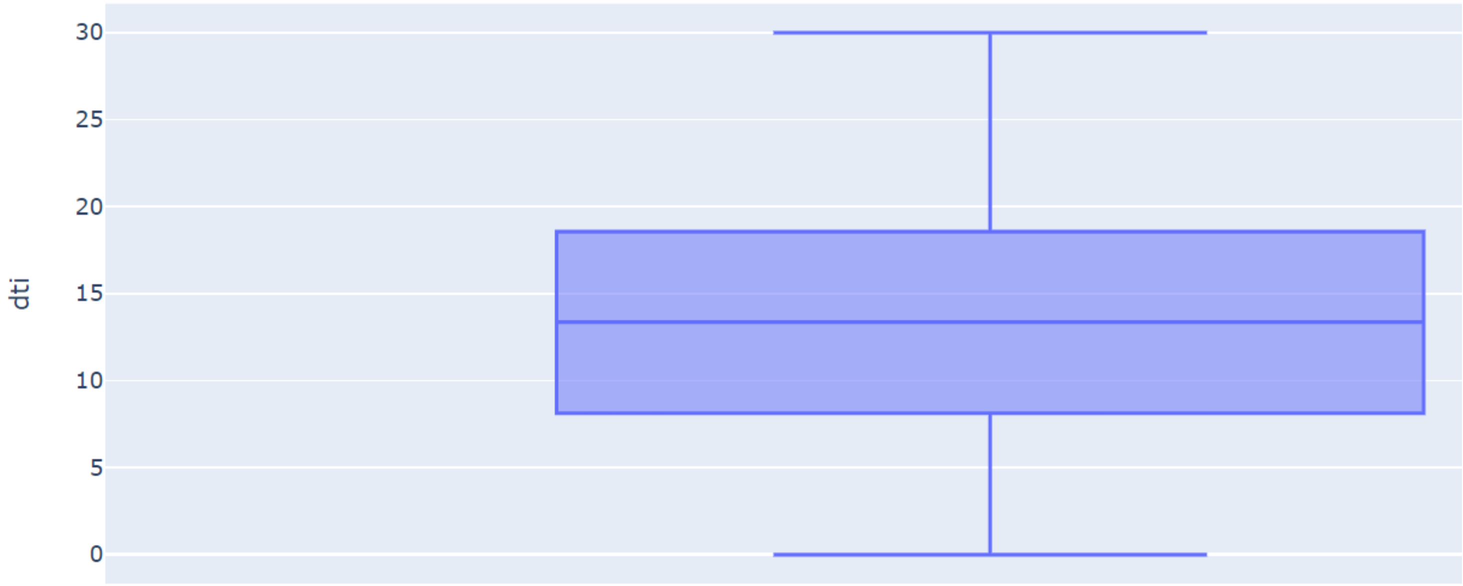
```
[199]: # ...

loan_df.dti.describe().apply(lambda x: format(x, 'f'))
```

```
[199]: count    38577.000000
       mean        13.272727
       std          6.673044
       min          0.000000
       25%          8.130000
       50%         13.370000
       75%         18.560000
       max         29.990000
       Name: dti, dtype: object
```

And several other steps up to cell 207

Step 8:For dti column we see the values are evenly spread hence no outliers cleanup required for this column

[200]:
```
#plotting chart for dti column
pltx.box(loan_df,y="dti")
```

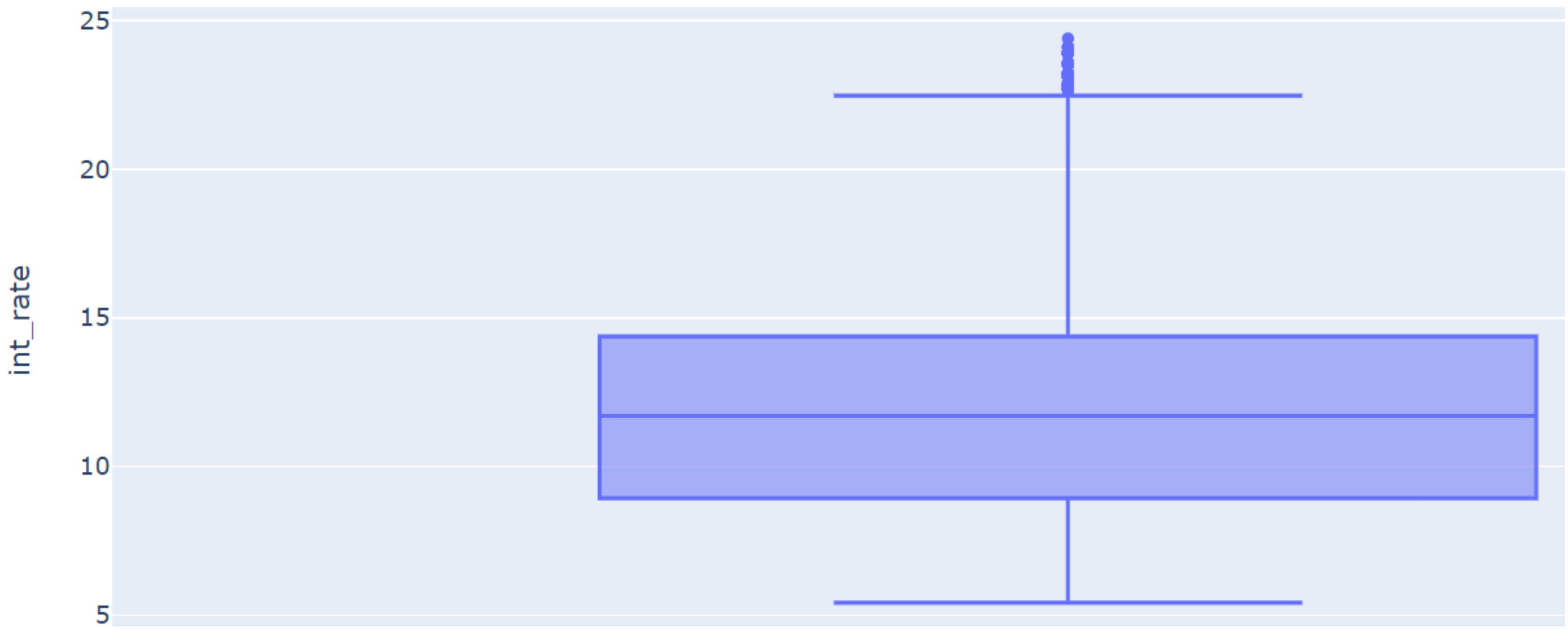


And several other steps up to cell 207

Step 8: Identifying Outliers and removing those records

```
[201]: #plotting chart for interest rate column
pltx.box(loan_df,y="int_rate")
```

- For int_rate column we see the values are evenly spread with some outliers seen above the upper fence.
- since the difference is not that huge we can skip the cleanup for this column.



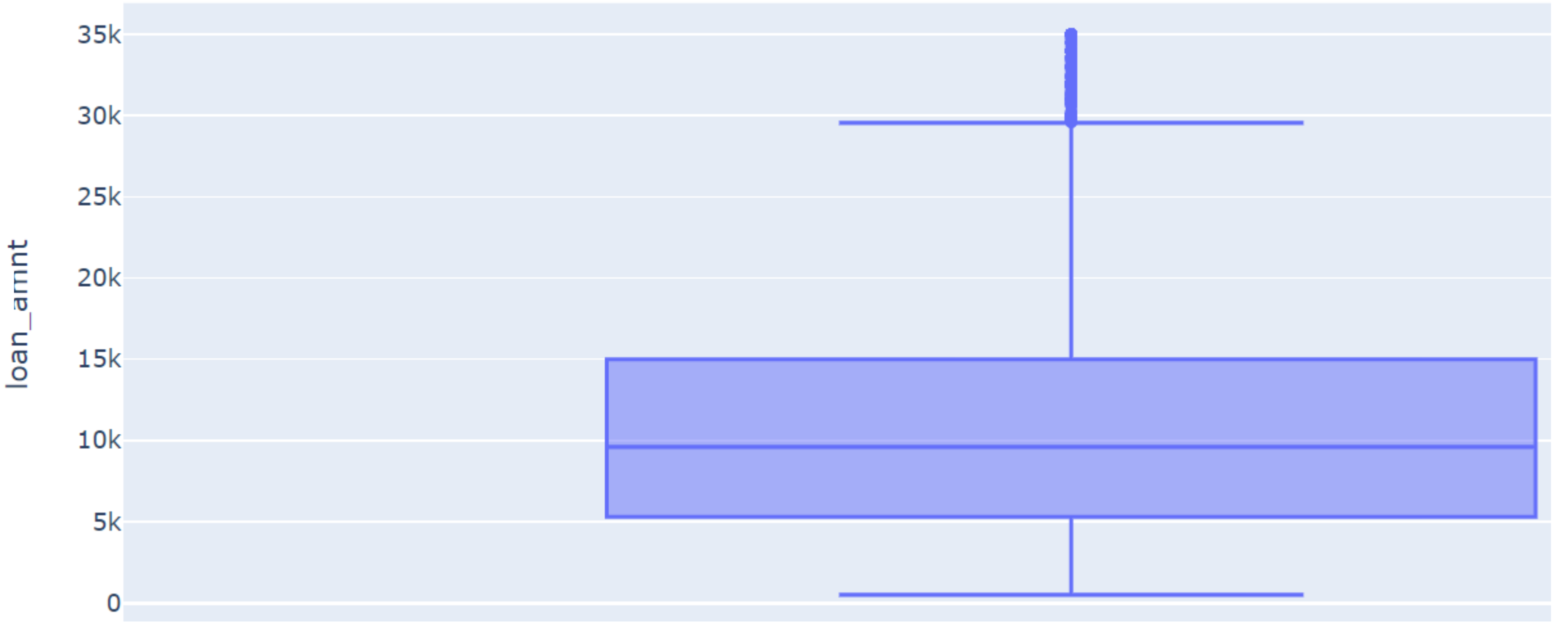
And several other steps up to cell 207

Step 8: Identifying Outliers and removing those records

- For loan amount column we see the values are evenly spread with some outliers seen above the upper fence.

- Since the difference is not that huge we can skip the cleanup for this column.

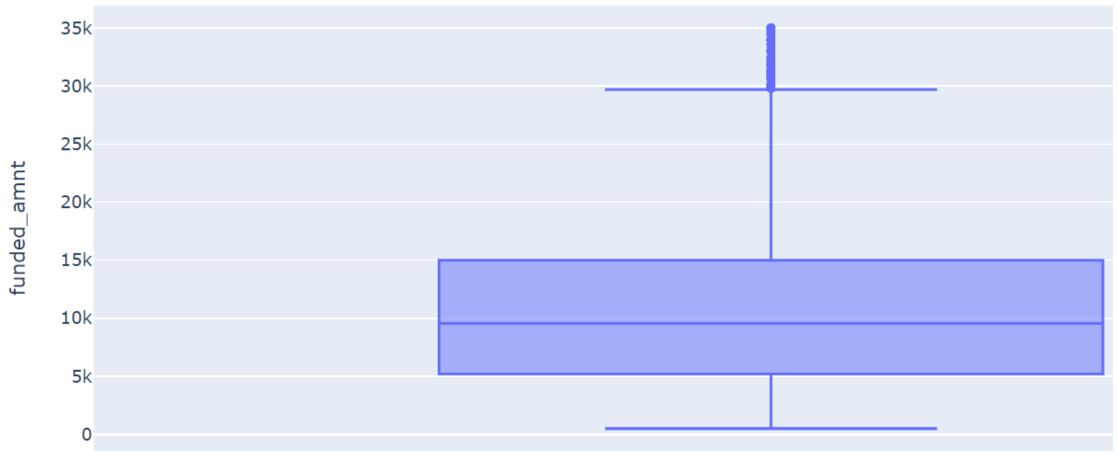
```
#plotting chart for loan amount column
pltx.box(loan_df,y="loan_amnt")
```



And several other steps up to cell 207

Step 8: Identifying Outliers and removing those records

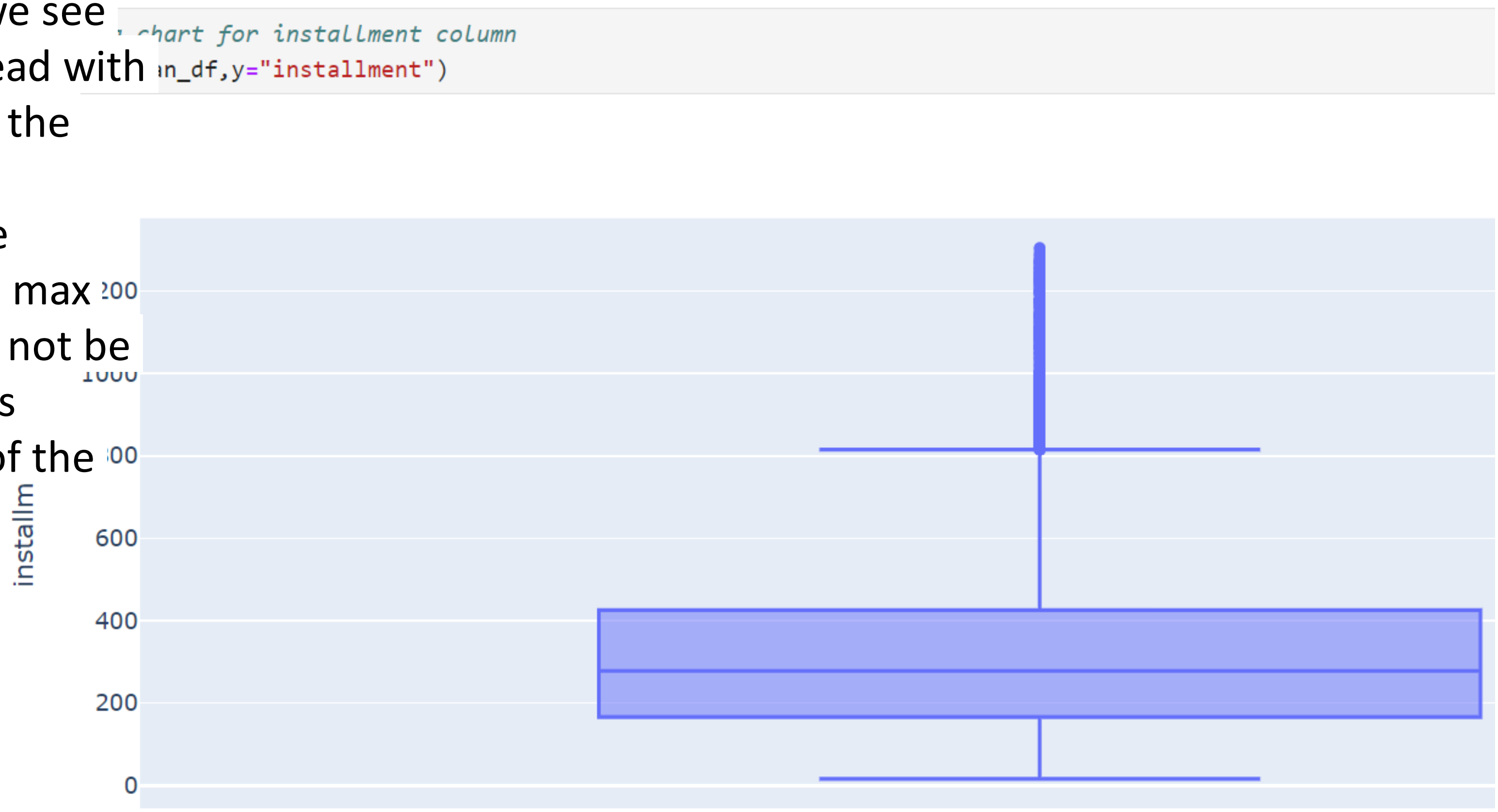
[203]:
```
#plotting chart for funded amount column
pltx.box(loan_df,y="funded_amnt")
```



- For funded amount column we see the values are evenly spread with some outliers seen above the upper fence.
- since the difference is not that huge we can skip the cleanup for this column.

Step 8: Identifying Outliers and removing those records

- For installment column we see the values are evenly spread with some outliers seen above the upper fence.
- We see a huge difference between upper fence and max value but this column will not be that useful for our analysis
- hence skipping cleanup of the outliers



Step 9: Check for the duplicated records after cleanup activity

- The code snippet checks for duplicated records in the `loan_df_for_analysis` DataFrame by applying the `.duplicated()` method, which identifies duplicate rows.

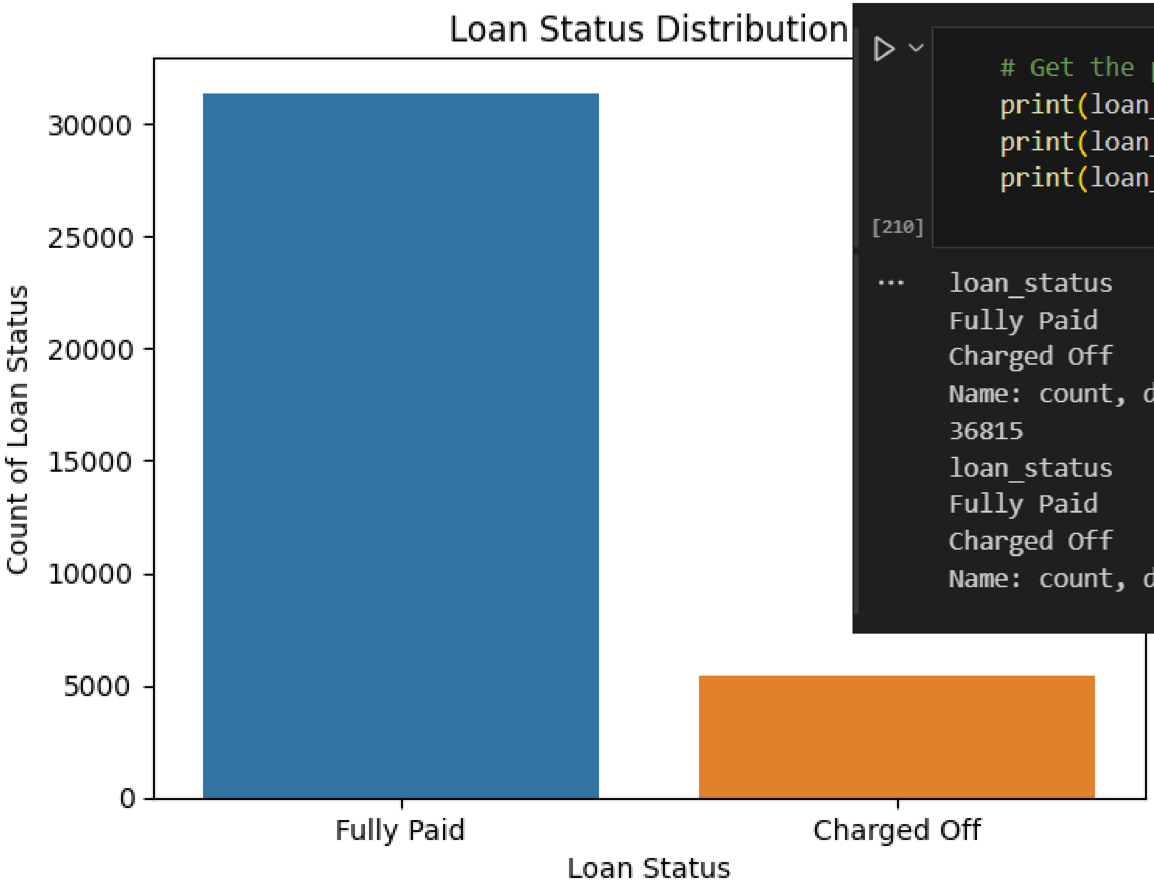
- The `.value_counts()` method is then used to count how many rows are unique versus duplicates. The output indicates that there are 36,815 unique rows and no duplicates (`False` indicates no duplicates).

- This confirms the data cleanup was effective, and the dataset is now ready for further analysis without redundancy issues.

```
[208]: # Check for duplicated records

       loan_df_for_analysis.duplicated().value_counts()

       # No duplicates found
```

```
[208]: False    36815
       Name: count, dtype: int64
```

Univariate Analysis :
Determining charged off loan percentage vs. fully paid.




```python
# Get the percentage of data for the above loan_status
print(loan_df_for_analysis.loan_status.value_counts())
print(loan_df_for_analysis.loan_status.count())
print(loan_df_for_analysis.loan_status.value_counts()*100/loan_df_for_analysis.loan_status.count())
```

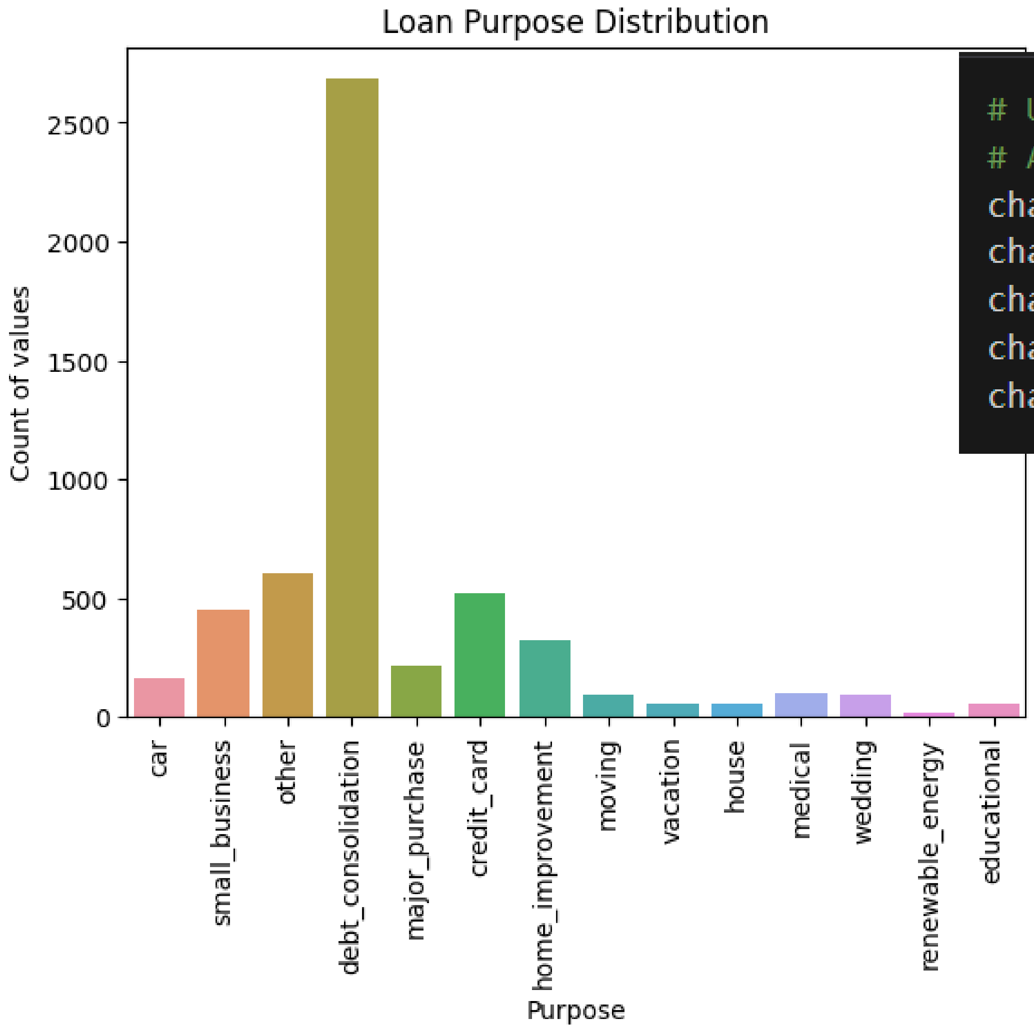
[210]

```
loan_status
Fully Paid     31384
Charged Off     5431
Name: count, dtype: int64
36815
loan_status
Fully Paid     85.247861
Charged Off    14.752139
Name: count, dtype: float64
```

As per the graph, charged off loan percentage is less compared to fully paid.

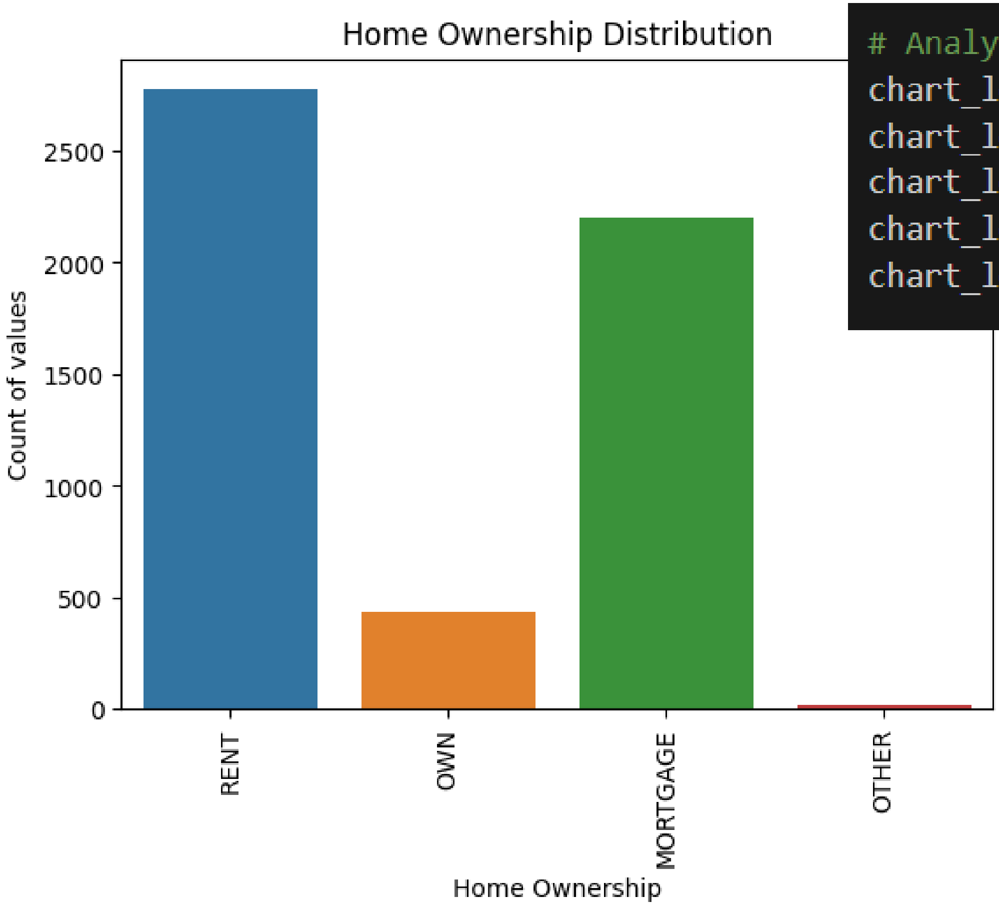
Categorical Variable Analaysis on Charged Off (Default Customers)

Observation 1:
Majority of Charged off loans with "purpose" variable indicates that applicant took loans to pay off other debts



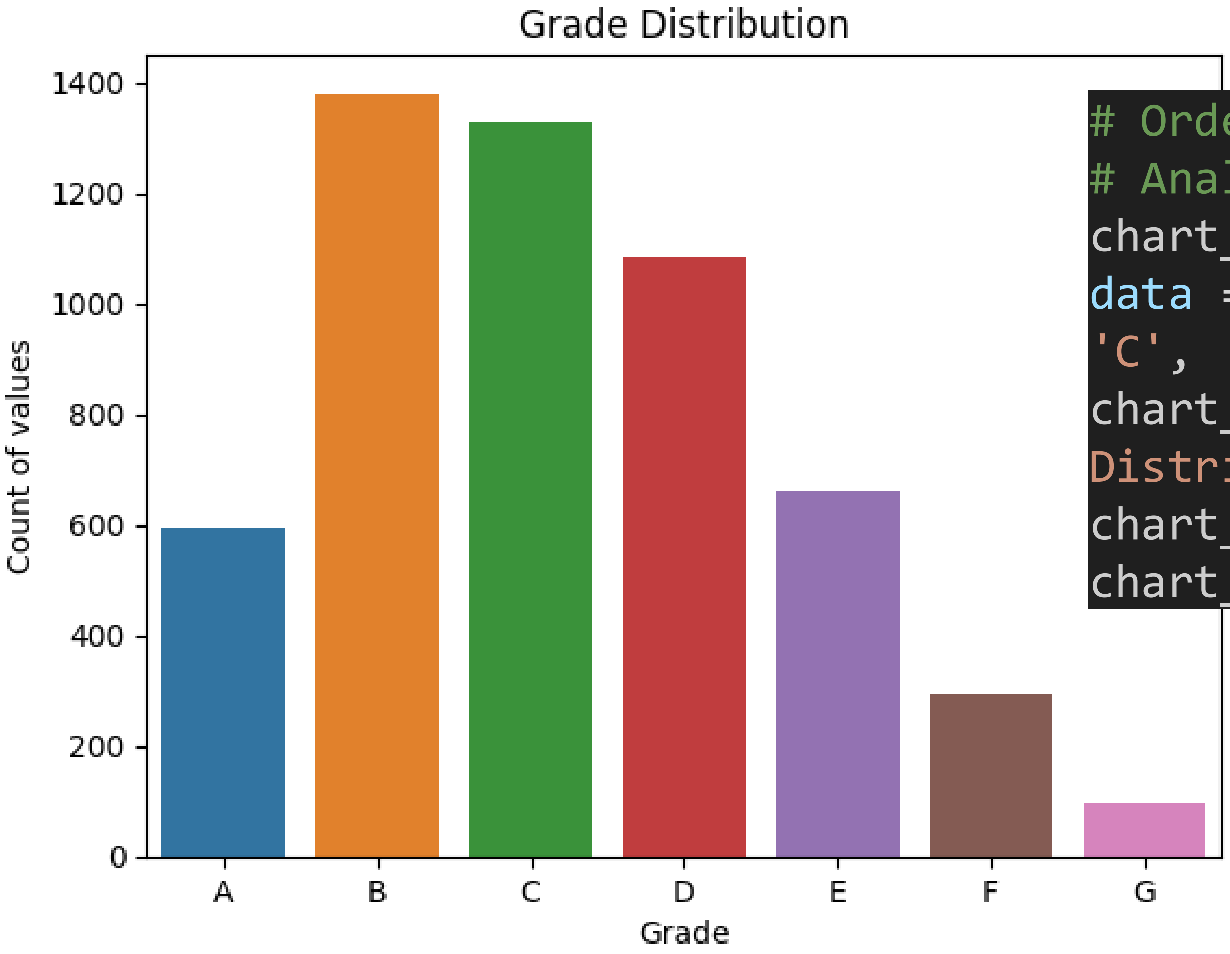

```python
# Unordered Categorical variable
# Analysis on purpose variable
chart_loan_purpose = sns.countplot(x = 'purpose', data = loan_df_chargedoff)
chart_loan_purpose.set_title("Loan Purpose Distribution")
chart_loan_purpose.set_xlabel("Purpose")
chart_loan_purpose.set_ylabel("Count of values")
chart_loan_purpose.tick_params(axis='x', rotation=90)
```

Observation 2:
Majority of Charged off loans with "home_ownership" variable indicates that applicant either stayed in rent or mortgaged house



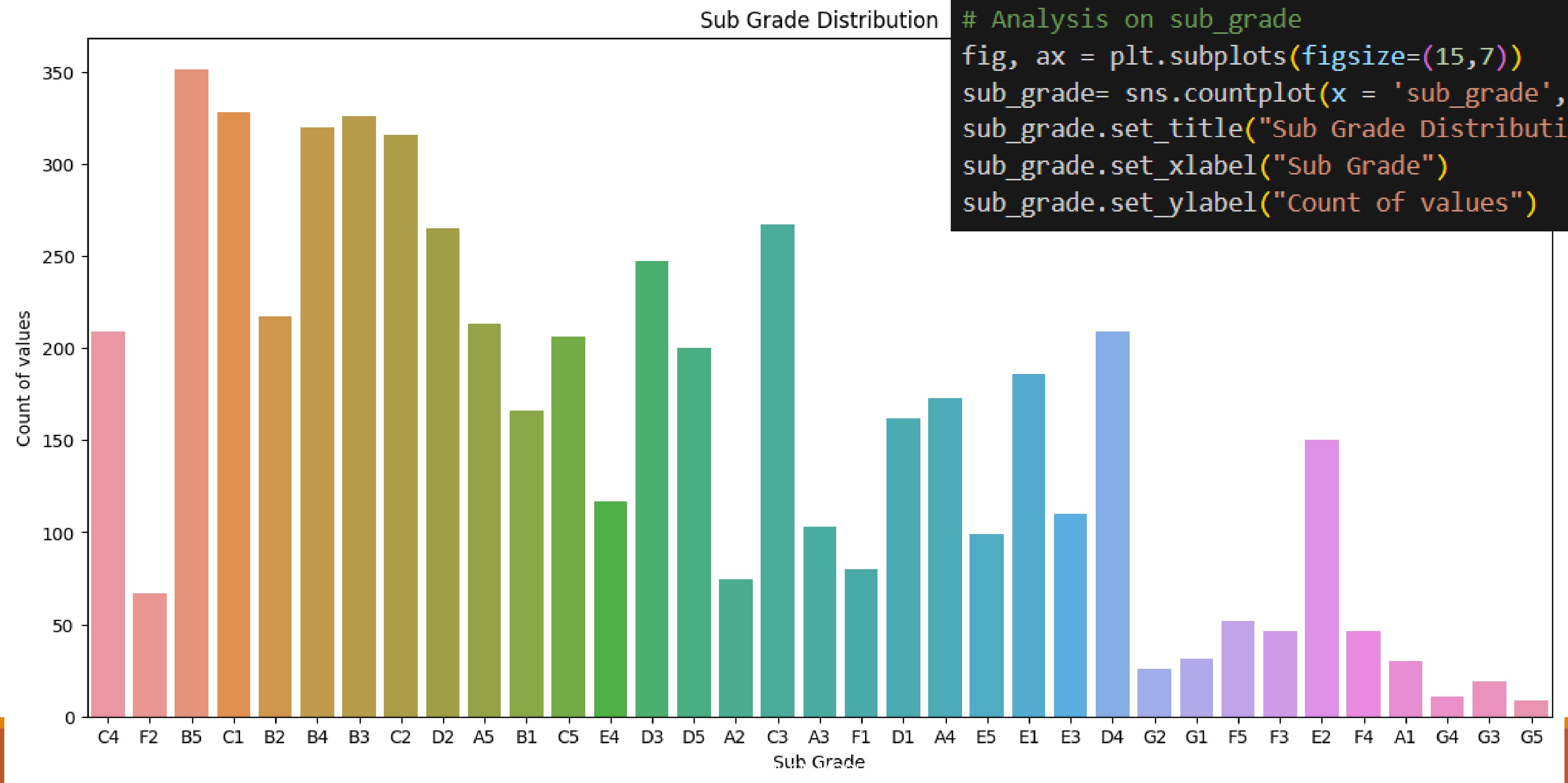

```python
# Analysis on home_ownership
chart_loan_ownership = sns.countplot(x = 'home_ownership', data = loan_df_chargedoff)
chart_loan_ownership.set_title("Home Ownership Distribution")
chart_loan_ownership.set_xlabel("Home Ownership")
chart_loan_ownership.set_ylabel("Count of values")
chart_loan_ownership.tick_params(axis='x', rotation=90)
```

Observation 3:
Majority of Charged off loans with "grade" variable indicates that applicant with grade "B" defaulted the most, followed by "C" & "D"




```
# Ordered Categorical variable
# Analysis on grade
chart_loan_grade= sns.countplot(x = 'grade',
data = loan_df_chargedoff, order = ['A', 'B',
'C', 'D', 'E', 'F', 'G'])
chart_loan_grade.set_title("Grade
Distribution")
chart_loan_grade.set_xlabel("Grade")
chart_loan_grade.set_ylabel("Count of values")
```

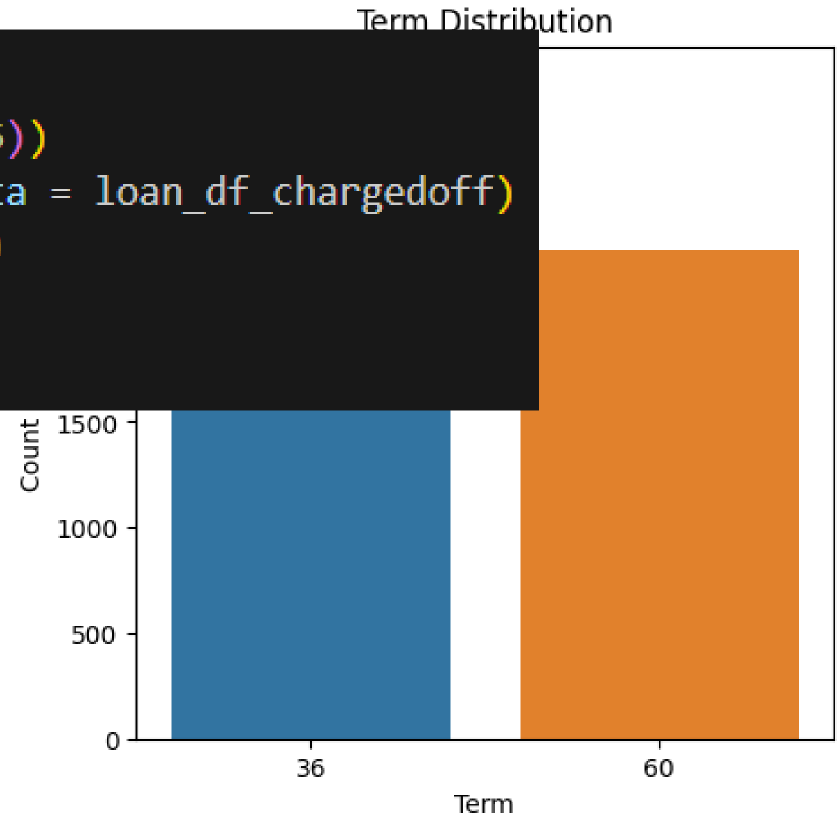
Observation 4:
"sub_grade" variable indicates that applicant with sub grade "B5" defaulted the most, followed by "C2" & "D2"




```python
# Analysis on sub_grade
fig, ax = plt.subplots(figsize=(15,7))
sub_grade= sns.countplot(x = 'sub_grade', data = loan_df_chargedoff)
sub_grade.set_title("Sub Grade Distribution")
sub_grade.set_xlabel("Sub Grade")
sub_grade.set_ylabel("Count of values")
```

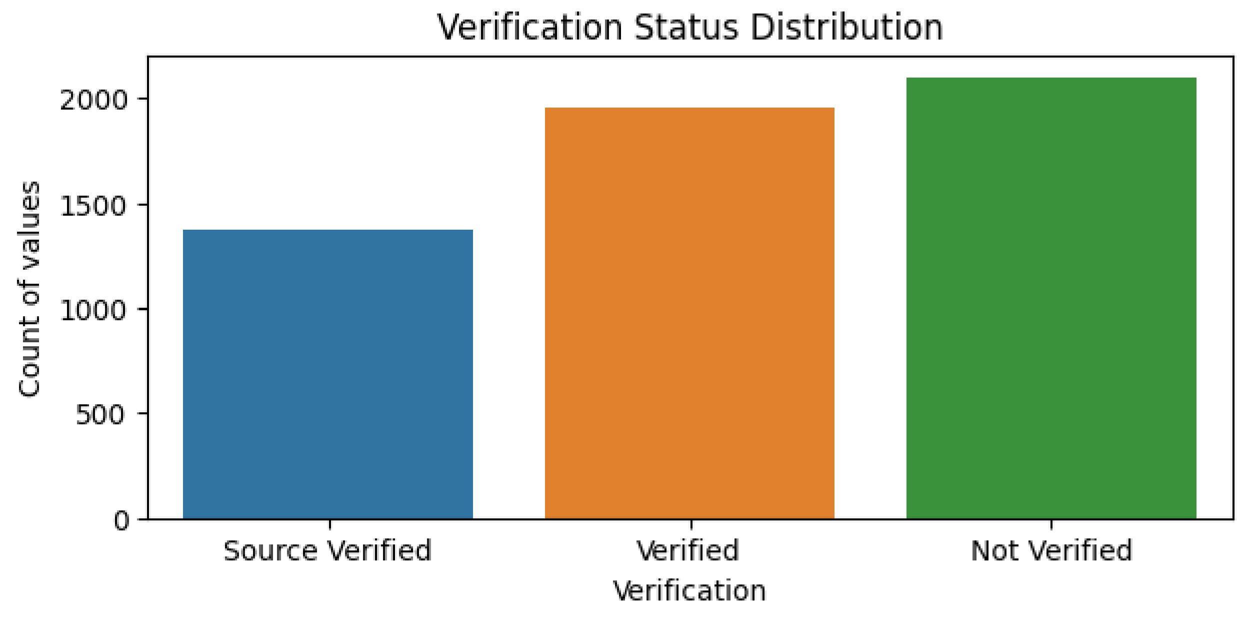
Observation 5:
"term" variable indicates that the applicant with term of 36 months defaulted the most.

```
# Analysis on term
fig, ax = plt.subplots(figsize=(5,5))
term= sns.countplot(x = 'term', data = loan_df_chargedoff)
term.set_title("Term Distribution")
term.set_xlabel("Term")
term.set_ylabel("Count of values")
```

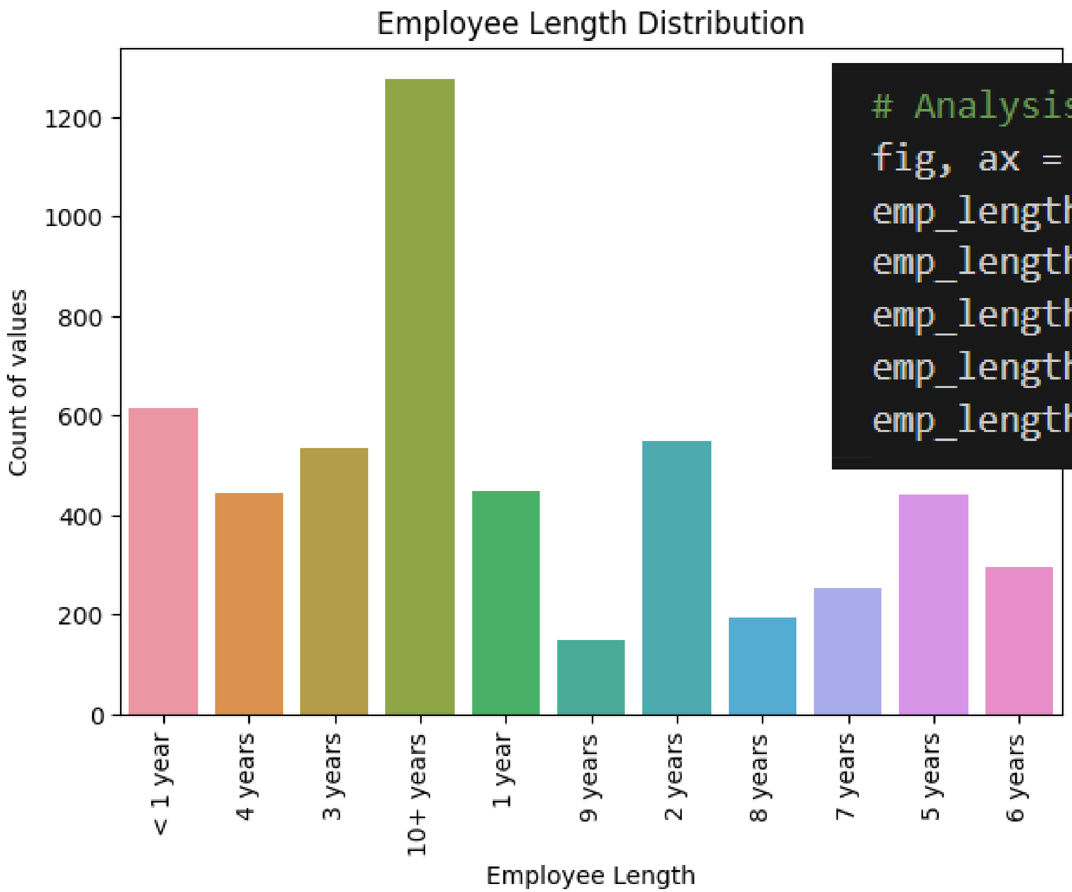


Observation 6:
Borrowers whose income was not verified defaulted more then verified

```
# Analysis on verification status
fig, ax = plt.subplots(figsize=(7,3))
ver_status= sns.countplot(x = 'verification_status', data = loan_df_chargedoff)
ver_status.set_title("Verification Status Distribution")
ver_status.set_xlabel("Verification")
ver_status.set_ylabel("Count of values")
```



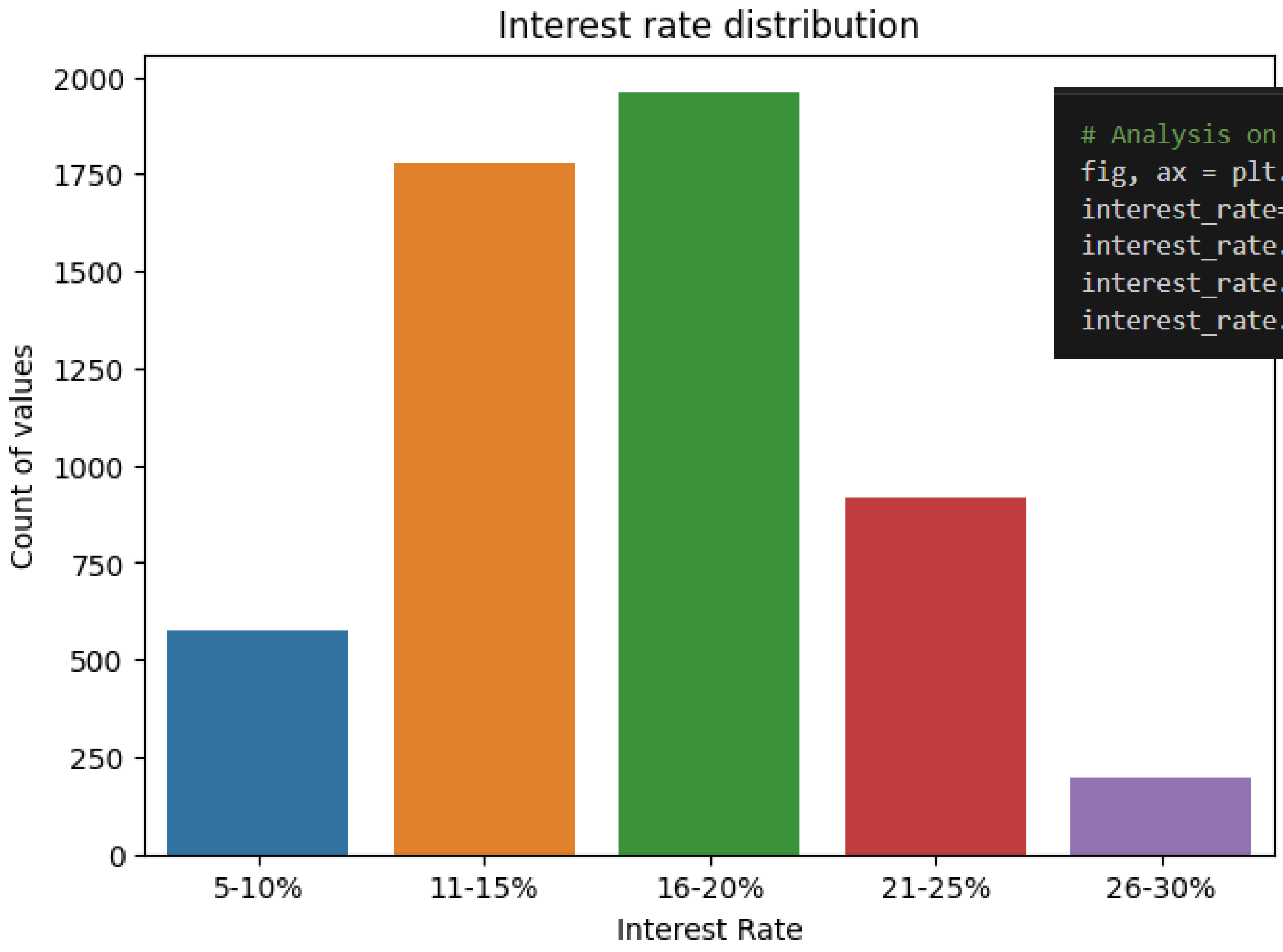
Observation 7: Numerical Variable Analaysis on Charged Off (Default Customers)




```python
# Analysis on emp_length
fig, ax = plt.subplots(figsize=(7,5))
emp_length= sns.countplot(x = 'emp_length', data = loan_df_chargedoff)
emp_length.set_title("Employee Length Distribution")
emp_length.set_xlabel("Employee Length")
emp_length.set_ylabel("Count of values")
emp_length.tick_params(axis='x', rotation=90)
```

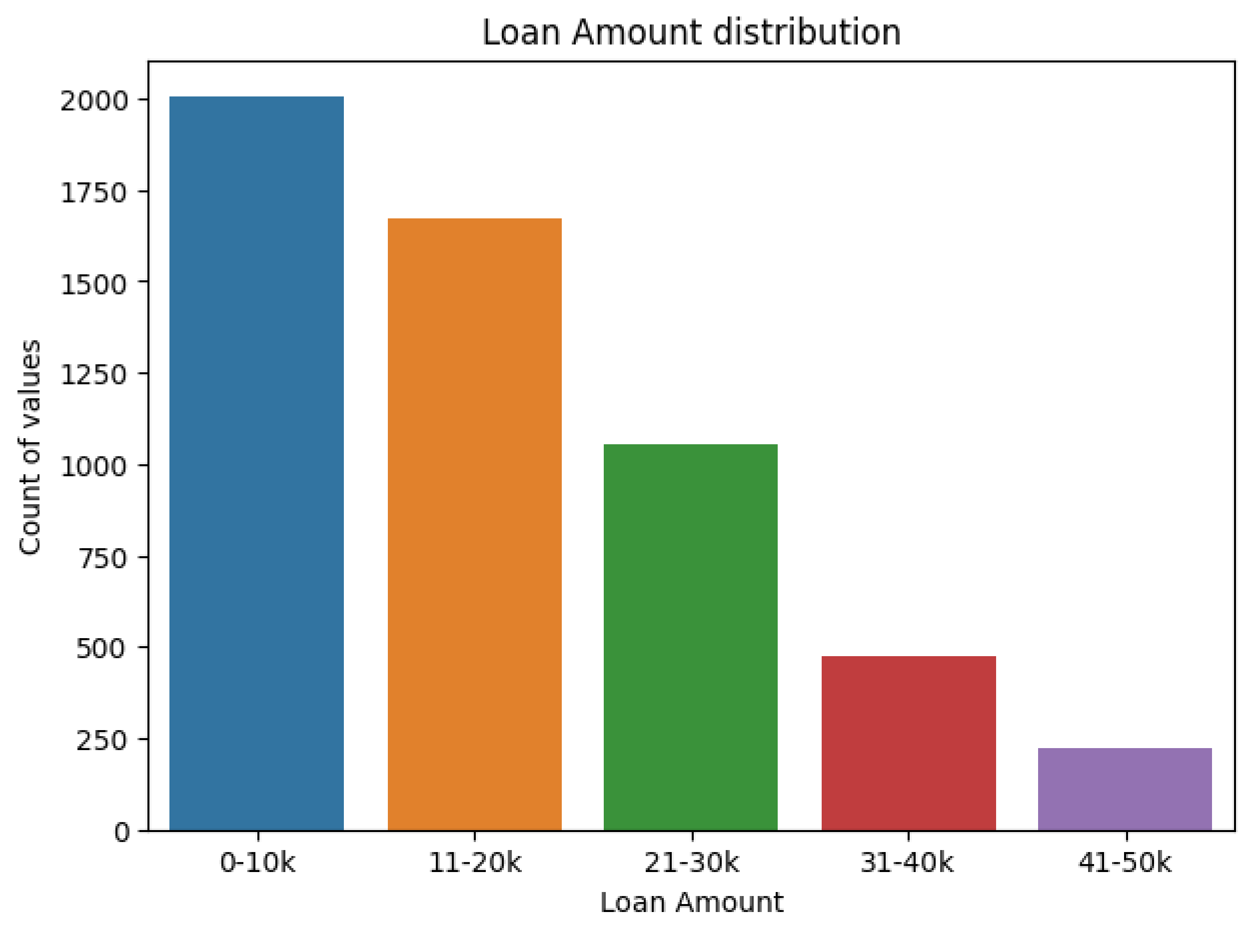
Borrowers whose experience is more 10 years or more defaulted the most.

Observation 8:
Borrowers who took loans at an interest rate between 16-20% defaulted the most.




```
# Analysis on interest rate
fig, ax = plt.subplots(figsize=(7,5))
interest_rate= sns.countplot(x = 'int_rate_buckets', data = loan_df_chargedoff)
interest_rate.set_title("Interest rate distribution")
interest_rate.set_xlabel("Interest Rate")
interest_rate.set_ylabel("Count of values")
```

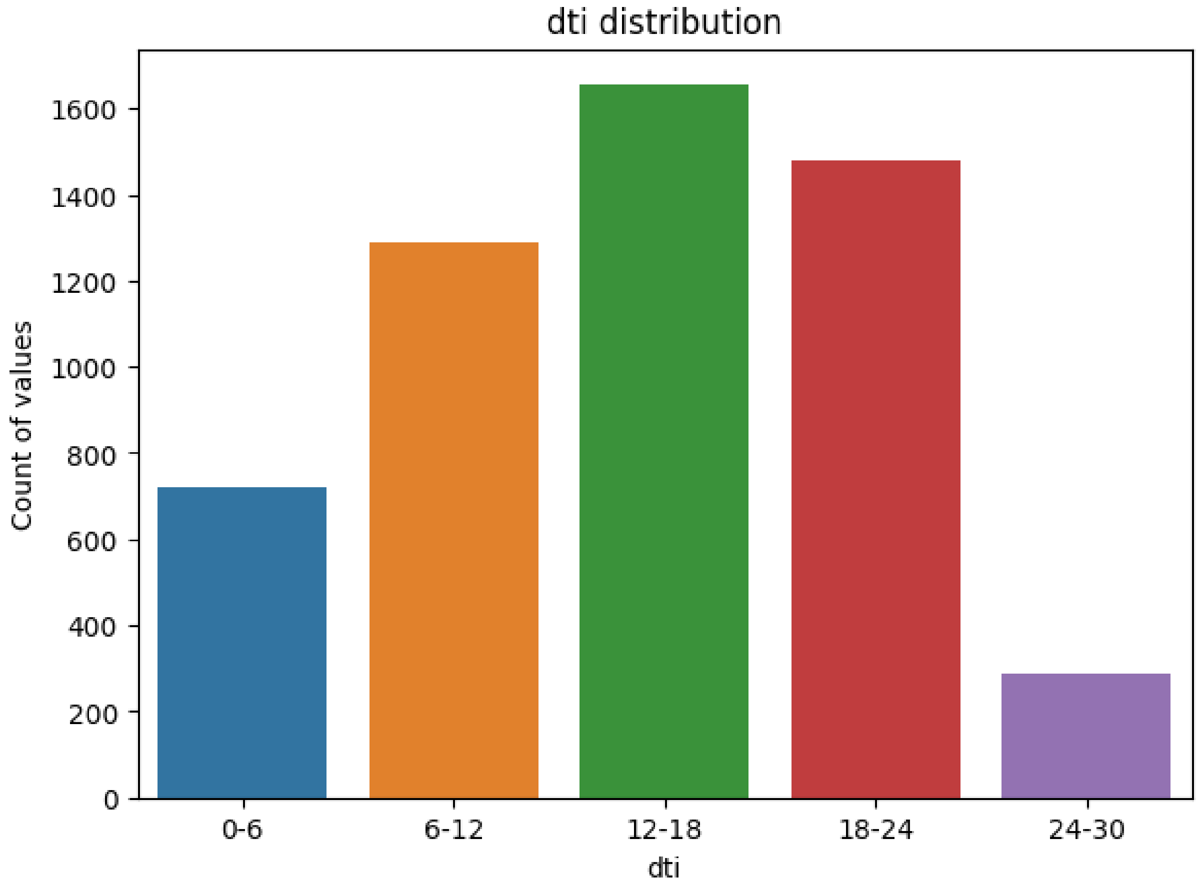
Observation 9:
Borrowers who took loans amount between 0-10k defaulted the most.



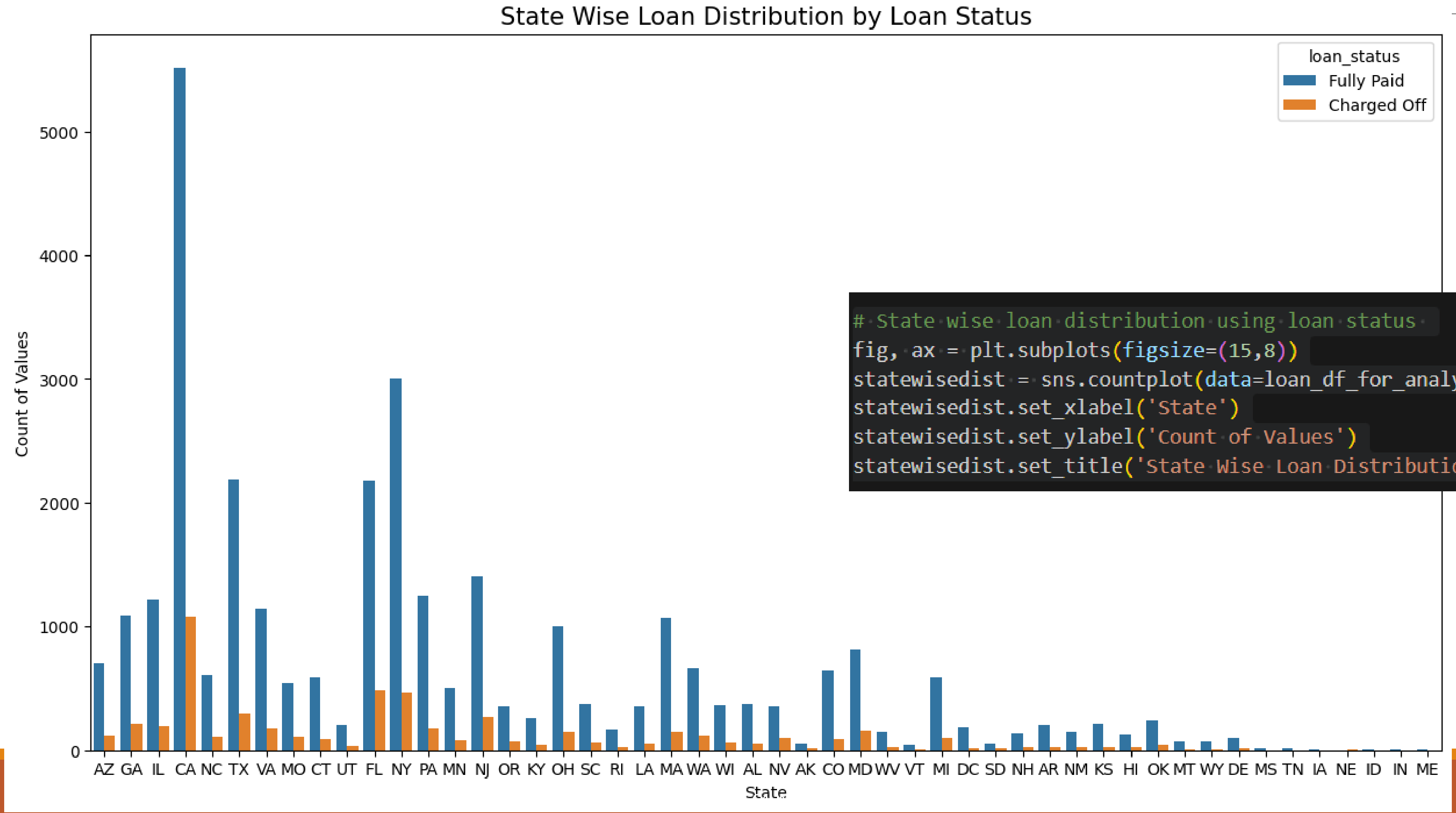

```
mount
ts(figsize=(7,5))
plot(x = 'loan_amnt_buckets', data = loan_df_chargedoff)
"Loan Amount distribution")
("Loan Amount")
("Count of values")
```

Observation 10:
Borrowers with dti range between 12-18 defaulted the most

```
# Analysis on Dti (debt to income ratio)

fig, ax = plt.subplots(figsize=(7,5))
Dti= sns.countplot(x = 'dti_buckets', data = loan_df_chargedoff)
Dti.set_title("dti distribution")
Dti.set_xlabel("dti")
Dti.set_ylabel("Count of values")
```

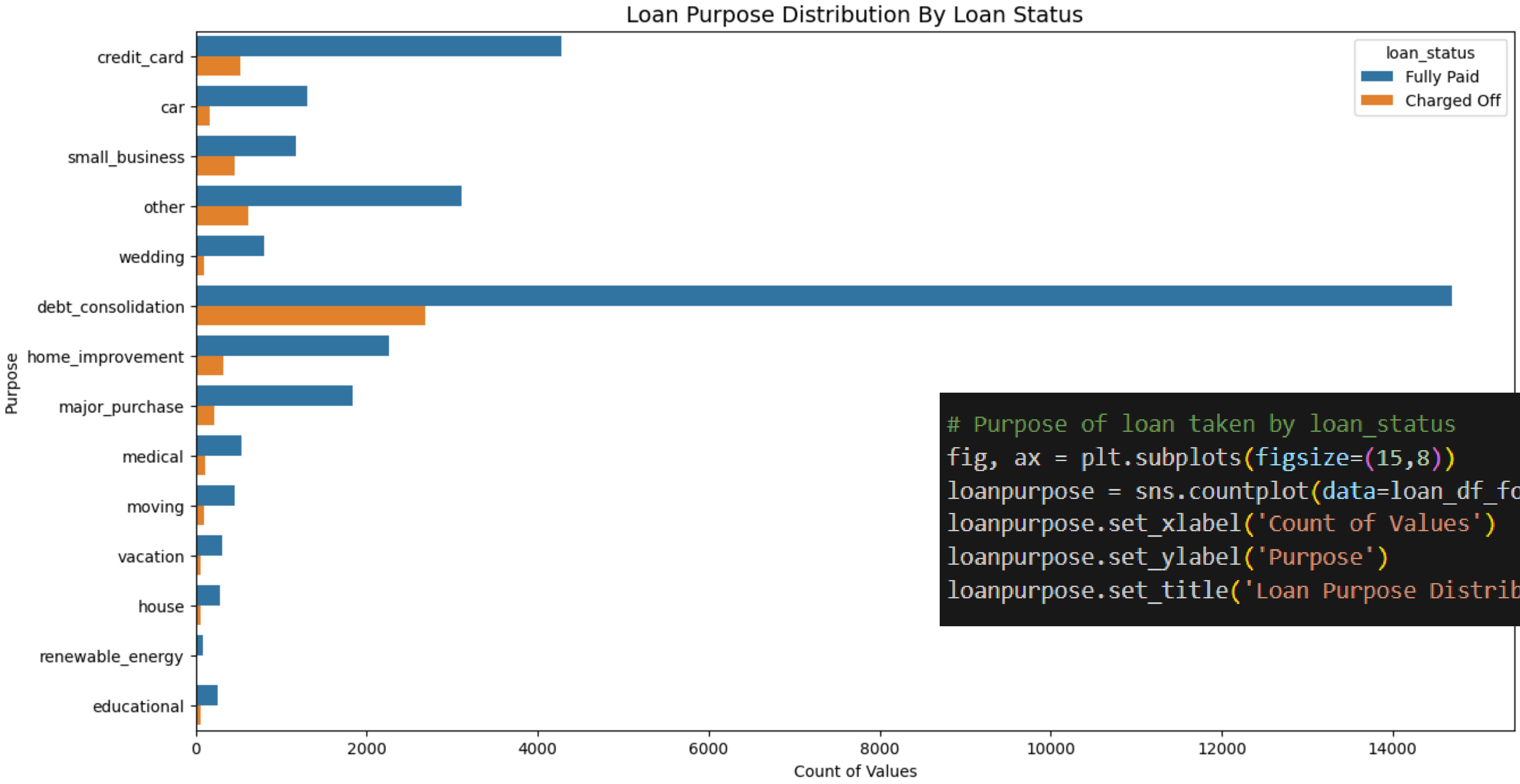


Segmented Univariate Analysis Observation 10:
Most of the borrowers are from the state "CA","NY","TX" and "FL" where major defaulters are from "CA", "FL" and "NY".




```
# State wise loan distribution using loan status
fig, ax = plt.subplots(figsize=(15,8))
statewisedist = sns.countplot(data=loan_df_for_analysis,x='addr_state',hue='loan_status')
statewisedist.set_xlabel('State')
statewisedist.set_ylabel('Count of Values')
statewisedist.set_title('State Wise Loan Distribution by Loan Status',fontsize=15)
```

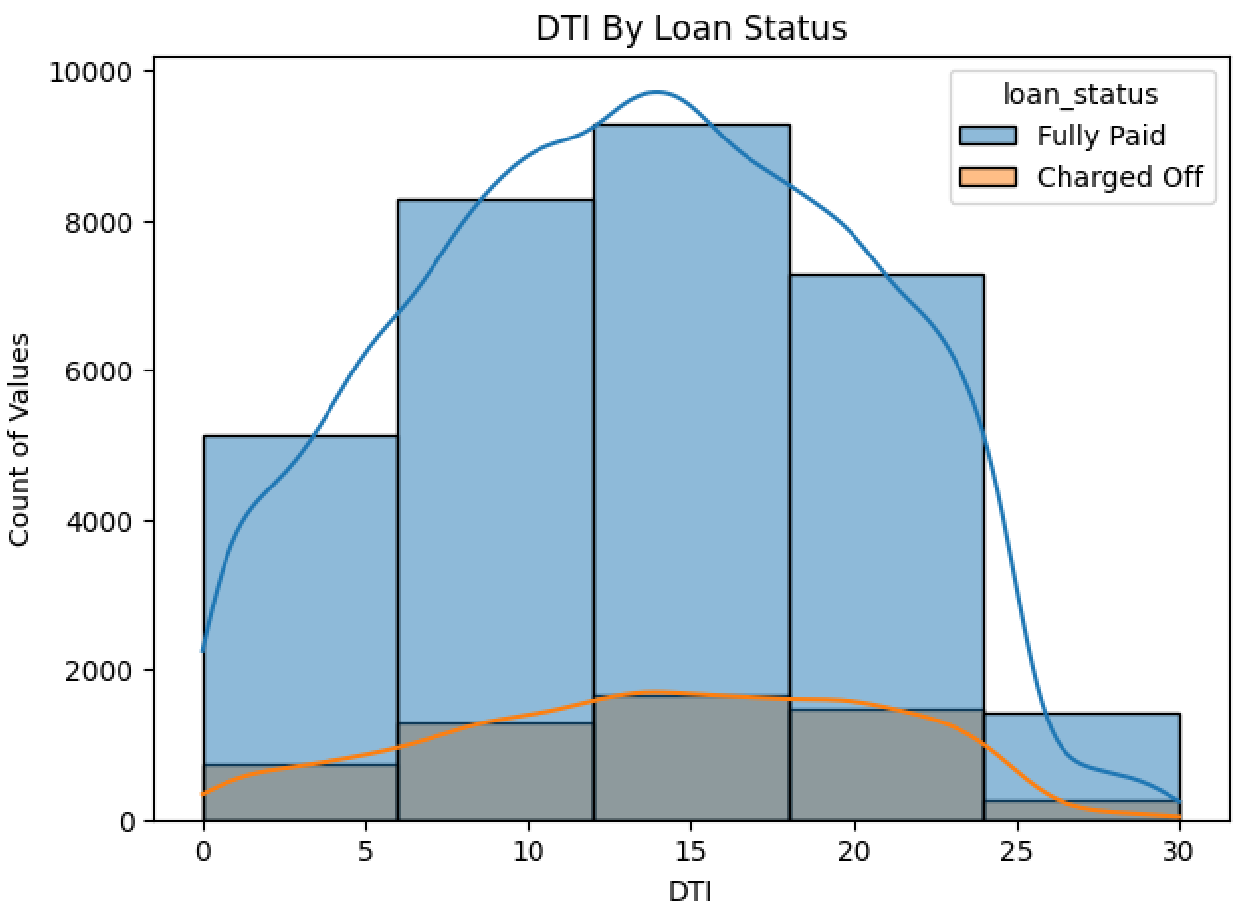
Observation 11:
Debt consolidation is the major reason for both fully paid and charged off loan applicant.




```
# Purpose of loan taken by loan_status
fig, ax = plt.subplots(figsize=(15,8))
loanpurpose = sns.countplot(data=loan_df_for_analysis,y='purpose',hue='loan_status')
loanpurpose.set_xlabel('Count of Values')
loanpurpose.set_ylabel('Purpose')
loanpurpose.set_title('Loan Purpose Distribution By Loan Status',fontsize=14)
```

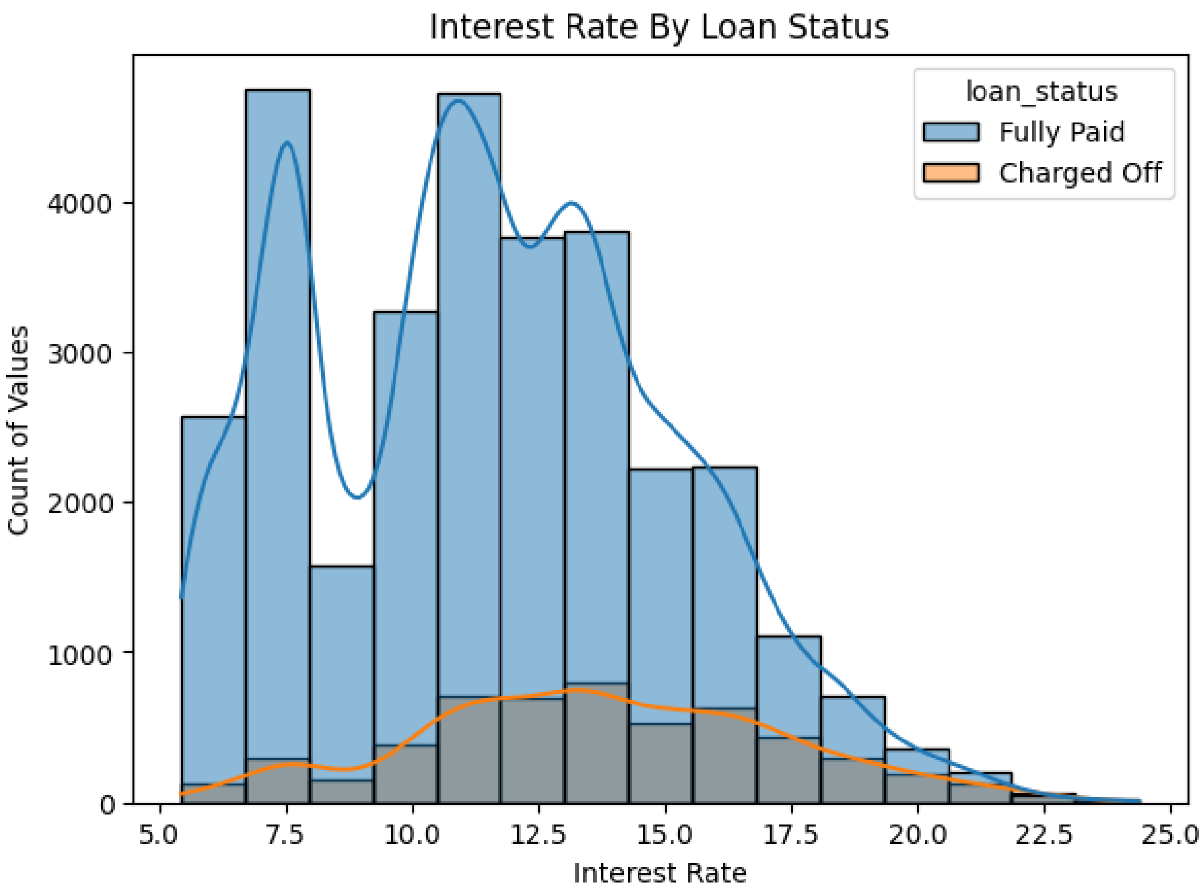
Observation 12:
DTI between 10-20 indicates a higher risks in terms of defaulters

```
# Dti (Debt to Income ratio) Vs Loan Status

fig, ax = plt.subplots(figsize=(7,5))

Dti = sns.histplot(data=loan_df_for_analysis,x='dti',hue='loan_status', bins=5,kde=True)

Dti.set_xlabel('DTI')

Dti.set_ylabel('Count of Values')

Dti.set_title('DTI By Loan Status',fontsize=12)
```



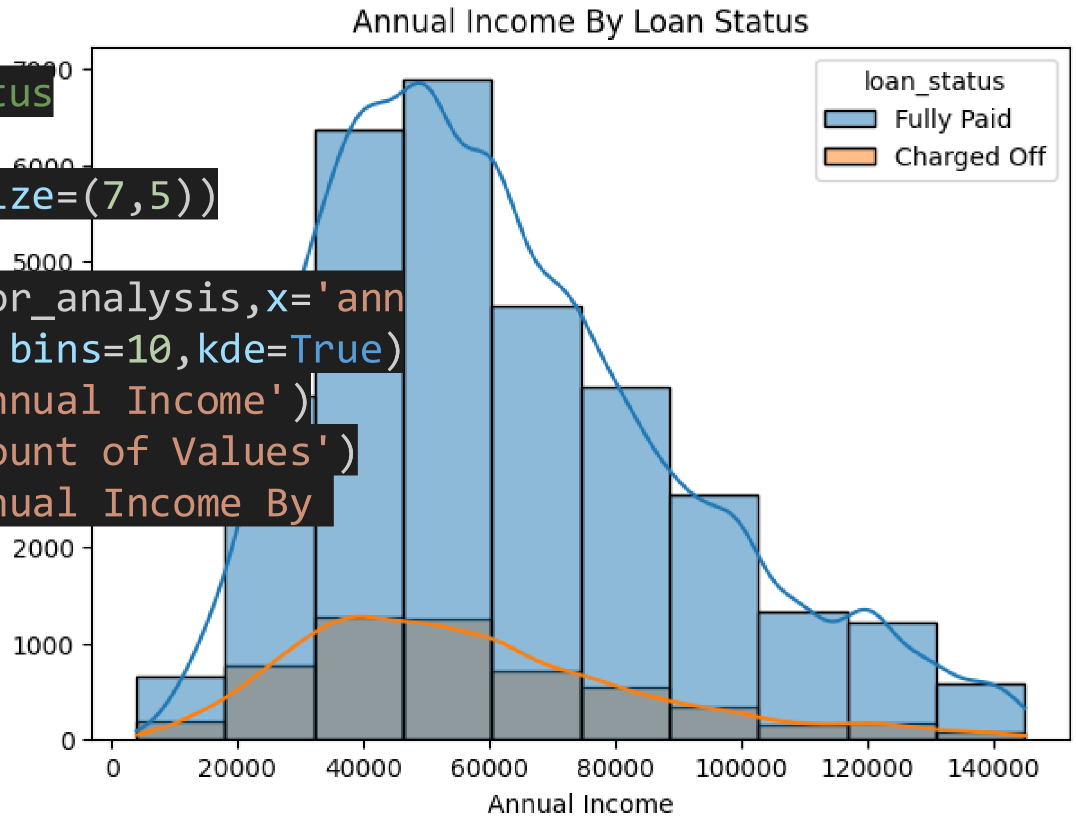
Observation 13:
Interest Rate between 10-17.5 has more number of defaulters

```
# Interest Rate Vs Loan Status

fig, ax = plt.subplots(figsize=(7,5))
Interest_Rate = 
sns.histplot(data=loan_df_for_analysis,x='int_rate
',hue='loan_status', bins=15,kde=True)
Interest_Rate.set_xlabel('Interest Rate')
Interest_Rate.set_ylabel('Count of Values')
Interest_Rate.set_title('Interest Rate By Loan 
Status',fontsize=12)
```



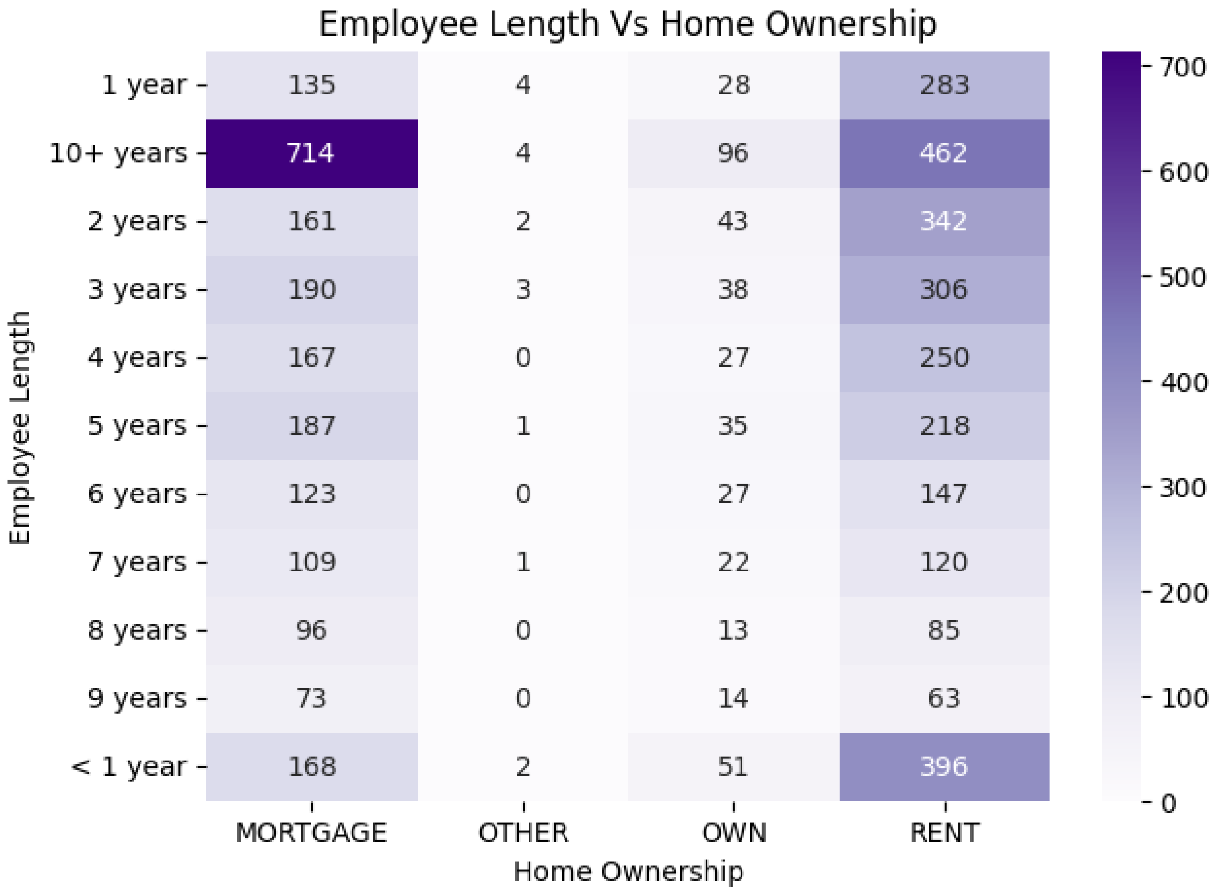
Observation 14:
Borowwers with Annual Income greater than 20K and less than 50k are more prone to default.

```python
# Annual Income Vs Loan Status

fig, ax = plt.subplots(figsize=(7,5))
Annual_Income = sns.histplot(data=loan_df_for_analysis,x='annual_inc',hue='loan_status', bins=10,kde=True)
Annual_Income.set_xlabel('Annual Income')
Annual_Income.set_ylabel('Count of Values')
Annual_Income.set_title('Annual Income By Loan Status',fontsize=12)
```



Categorical Bivariate Analysis Observation 15:

```python
# On Employee Length and Home Ownership
 # Creating a cross tab
cross_emplen_homeowner
=  pd.crosstab(loan_df_chargedoff['emp_length'],loan
_df_chargedoff['home_ownership'])

fig, ax = plt.subplots(figsize=(7,5))
emplen_homeowner =
sns.heatmap(cross_emplen_homeowner, fmt='d',
cmap='Purples', annot=True)
emplen_homeowner.set_xlabel('Home Ownership')
emplen_homeowner.set_ylabel('Employee Length')
emplen_homeowner.set_title('Employee Length Vs Home
Ownership',fontsize=12)
```

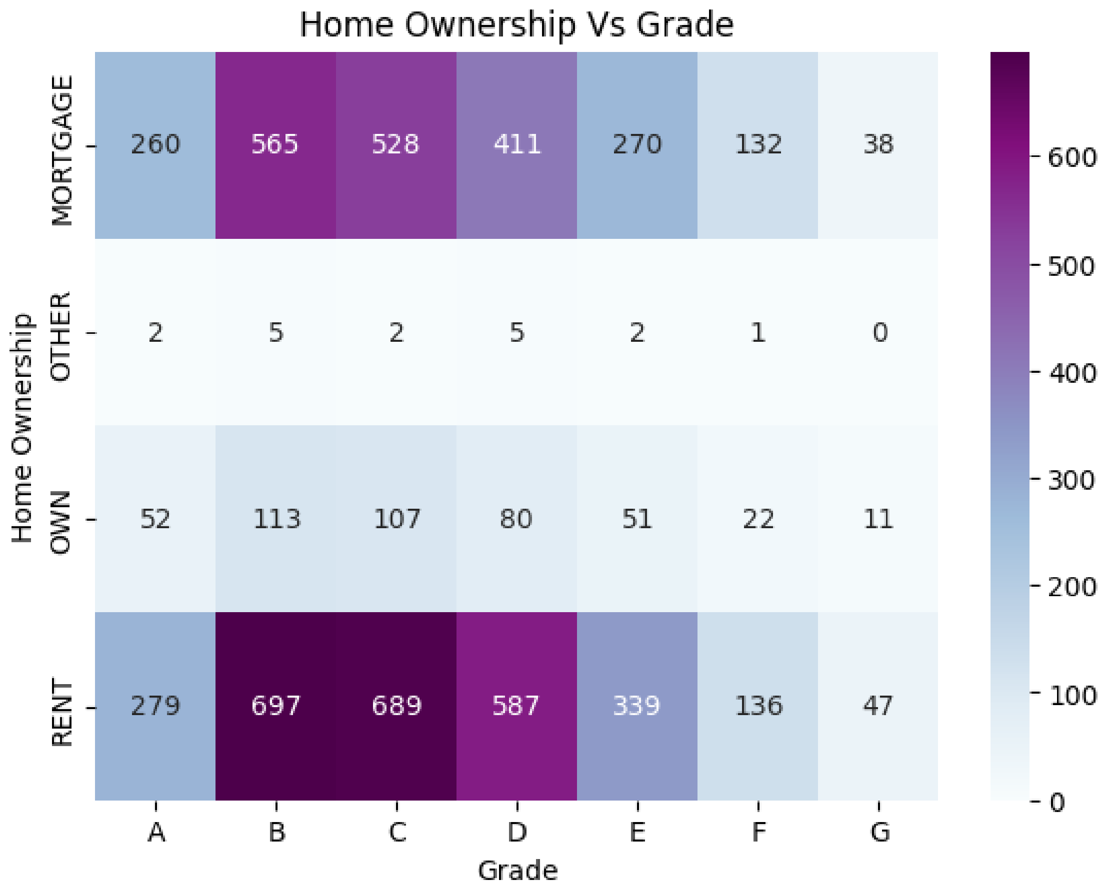


Observation 15:

Borrowers who have more than 10+ years of experience and live on rent or mortgages have high default rate

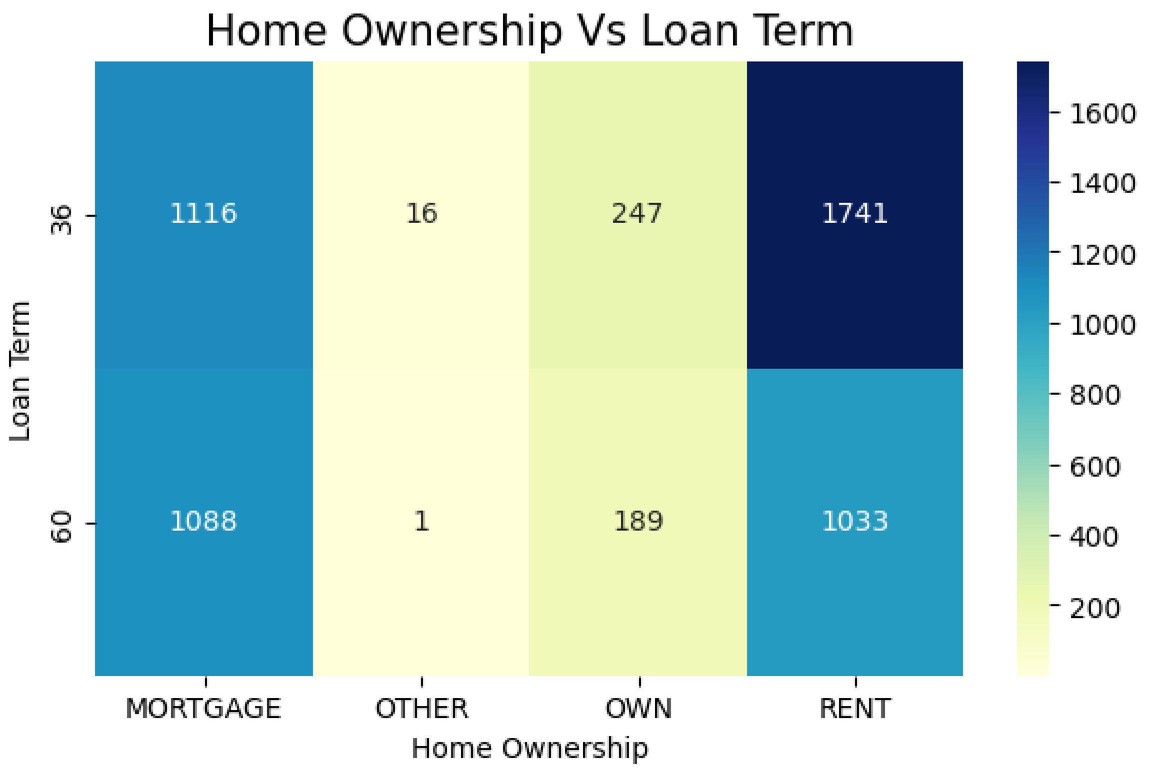
Observation 16:
Borrowers with assigned grade B,C & D fall in higher catergory of defaulters

```
# On Home Ownership Vs Grade
# Creating a cross tab
cross_grade_homeowner
=  pd.crosstab(loan_df_chargedoff['home_owne
rship'], loan_df_chargedoff['grade'])

fig, ax = plt.subplots(figsize=(7,5))
cross_grade_homeowner =
sns.heatmap(cross_grade_homeowner, fmt='d',
cmap='BuPu', annot=True)
cross_grade_homeowner.set_xlabel('Grade')
cross_grade_homeowner.set_ylabel('Home
Ownership')
cross_grade_homeowner.set_title('Home
Ownership Vs Grade',fontsize=12)
```

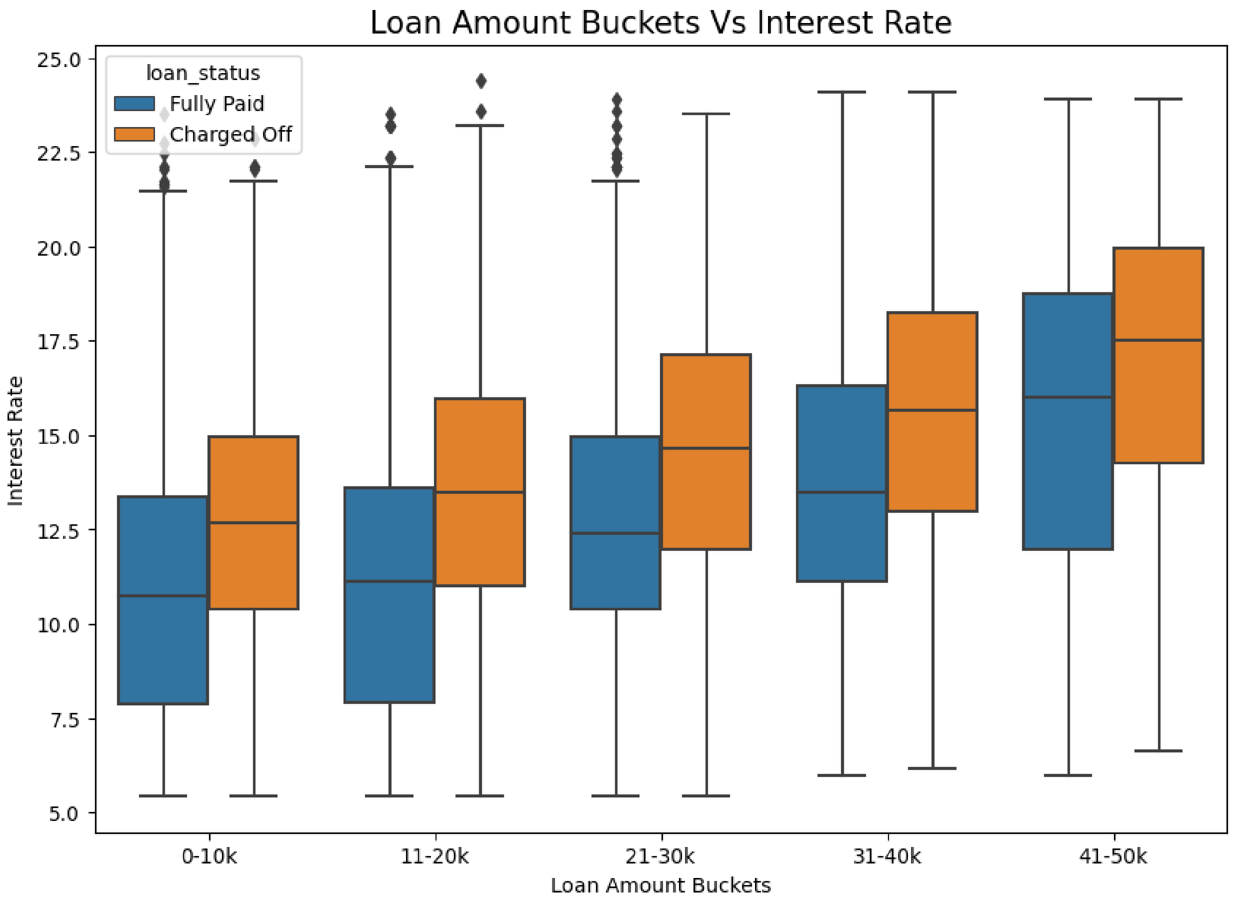


Observation 17:
Borrowers staying on Rent and with shorter loan term tend to default more.

```
# Home Ownership Vs Loan Term
# Creating a cross tab
ctab_term_homeownership =
pd.crosstab(loan_df_chargedoff['term'],
loan_df_chargedoff['home_ownership'])

fig, ax = plt.subplots(figsize=(7,4))
ctab_term_homeownership =
sns.heatmap(ctab_term_homeownership,
annot=True, fmt='d', cmap='YlGnBu')
ctab_term_homeownership.set_xlabel('Home
Ownership')
ctab_term_homeownership.set_ylabel('Loan Term')
ctab_term_homeownership.set_title('Home
Ownership Vs Loan Term',fontsize=15)
```



Numerical Bivariate Analysis Observation 18:



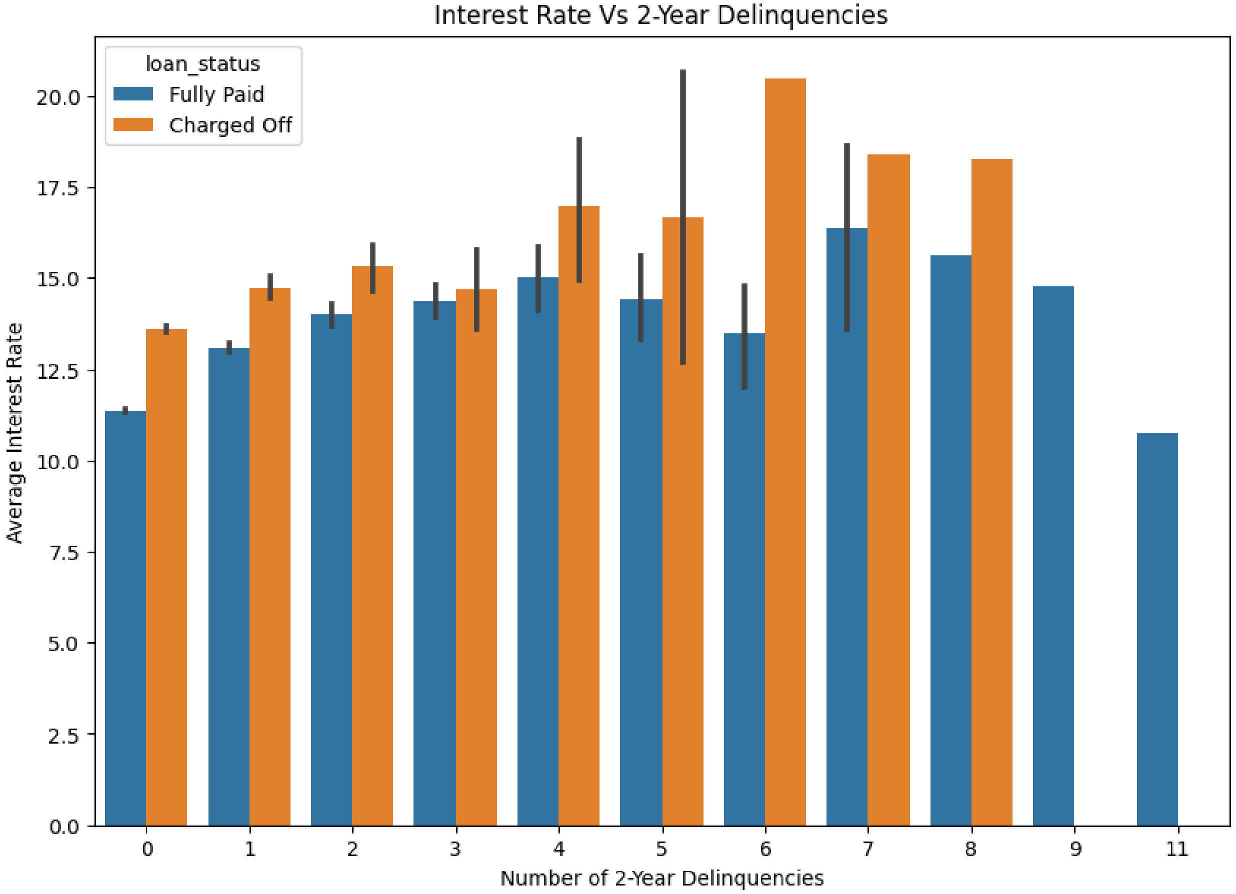
Observation 18:
As per the above data points we see can interest rate is high for the charged off loan_status compared to fully paid.

```
# On Interest rate and Loan Amount Bucket
fig, ax = plt.subplots(figsize=(10,7))
interest_rate_Loan = sns.boxplot(y='int_rate',x='loan_amnt_buckets',data=loan_df_for_analysis, hue='loan_status')
interest_rate_Loan.set_xlabel('Loan Amount Buckets')
interest_rate_Loan.set_ylabel('Interest Rate')
interest_rate_Loan.set_title('Loan Amount Buckets Vs Interest Rate',fontsize=15)
```

Numerical Bivariate Analysis Observation 19:

Interest rate vs delinq

It is observed that *2-year delinq with value 6 has the highest interest rate.*

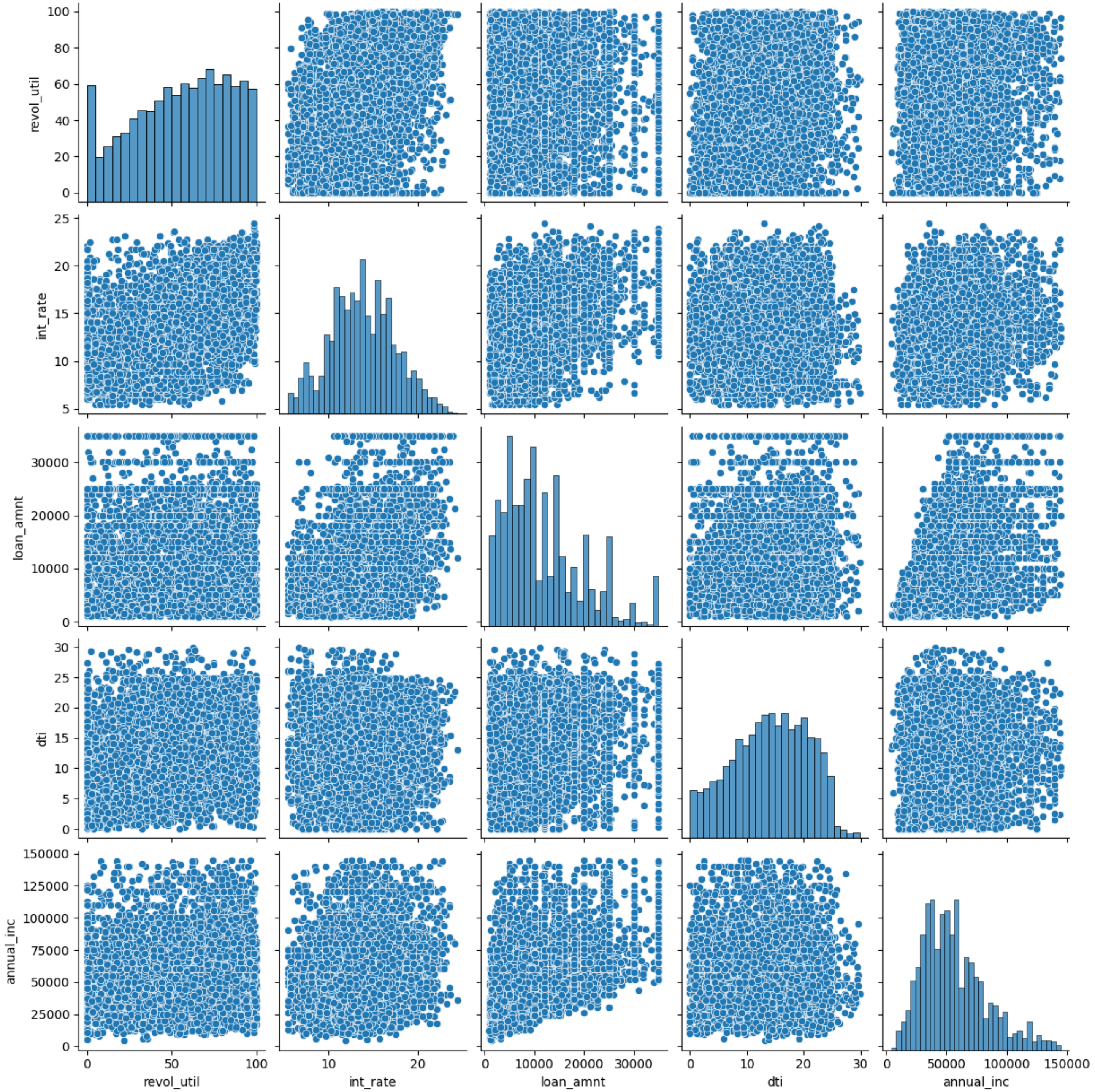



```
fig, ax = plt.subplots(figsize=(10,7))
interestratevsdelinq = sns.barplot(x='delinq_2yrs', y='int_rate', data=loan_df_for_analysis , hue='loan_status')
interestratevsdelinq.set_xlabel('Number of 2-Year Delinquencies')
interestratevsdelinq.set_ylabel('Average Interest Rate')
interestratevsdelinq.set_title('Interest Rate Vs 2-Year Delinquencies')
```

```
Text(0.5, 1.0, 'Interest Rate Vs 2-Year Delinquencies')
```

Using pair plot for understanding the distribution accross different numerical variables

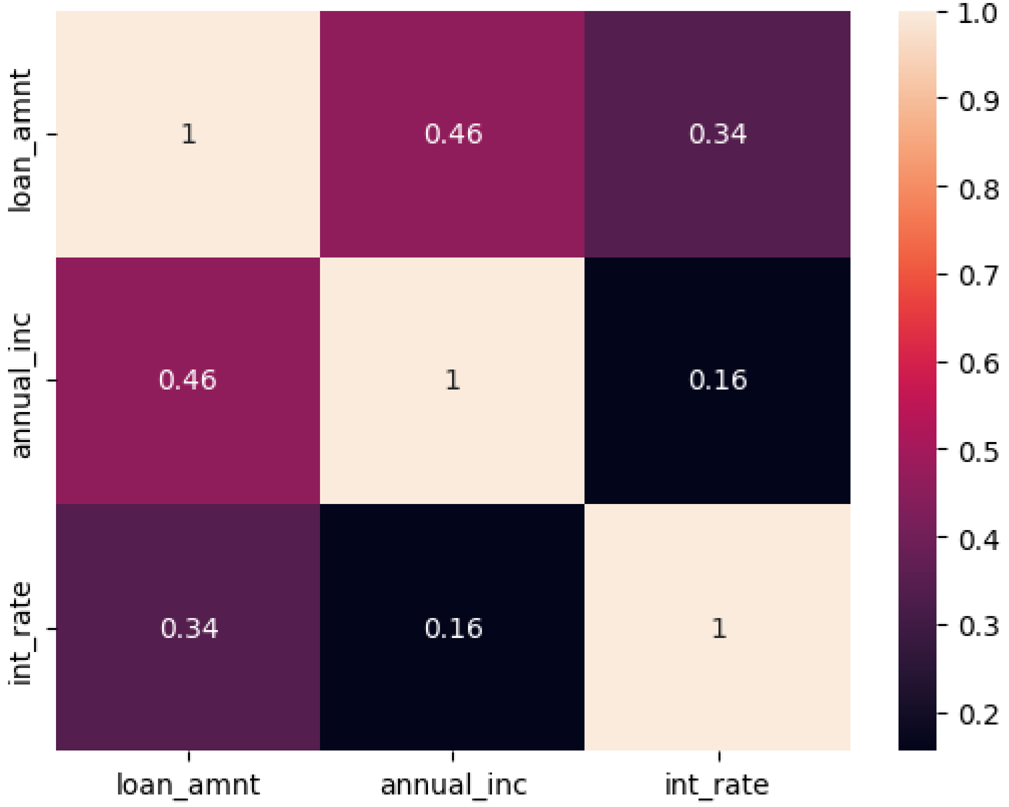
```
sns.pairplot(loan_df_chargedoff,
vars=['revol_util', 'int_rate',
'loan_amnt', 'dti', 'annual_inc'])
```



Multivariate Analysis

The code generates a heatmap visualizing the correlation among 'loan_amnt', 'annual_inc', and 'int_rate' within `loan_df_chargedoff`.

It selects these numerical columns, computes their correlation matrix, and uses Seaborn's `heatmap` function to display the results with annotations, helping identify relationships between these financial variables.



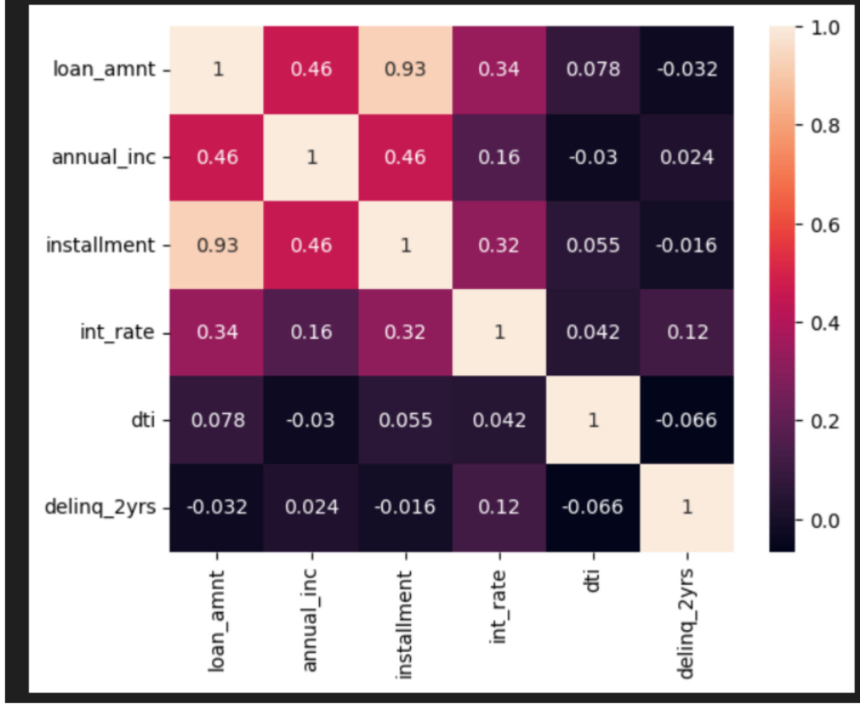

```
# Plotting correlation between loan amount, annual income and interest rate on charged off loan status
num_cols = ['loan_amnt', 'annual_inc','int_rate']
sns.heatmap(loan_df_chargedoff[num_cols].corr(),annot=True)
```

Multivariate Analysis

- This code snippet visualizes the correlation among selected financial attributes—loan amount, annual income, installment, interest rate, debt-to-income ratio (DTI), and delinquencies over the past two years—for loans that have been charged off.

- It defines these attributes in a list, calculates their correlation matrix from the `loan_df_chargedoff` DataFrame, and then uses Seaborn's `heatmap` function to create a heatmap. Annotations are enabled to display the correlation coefficients on the heatmap. The `plt.show()` function is used to display the plot.

```
num_cols = ['loan_amnt', 'annual_inc', 'installment', 'int_rate', 'dti','delinq_2yrs']
sns.heatmap(loan_df_chargedoff[num_cols].corr(),annot=True)
plt.show()
```

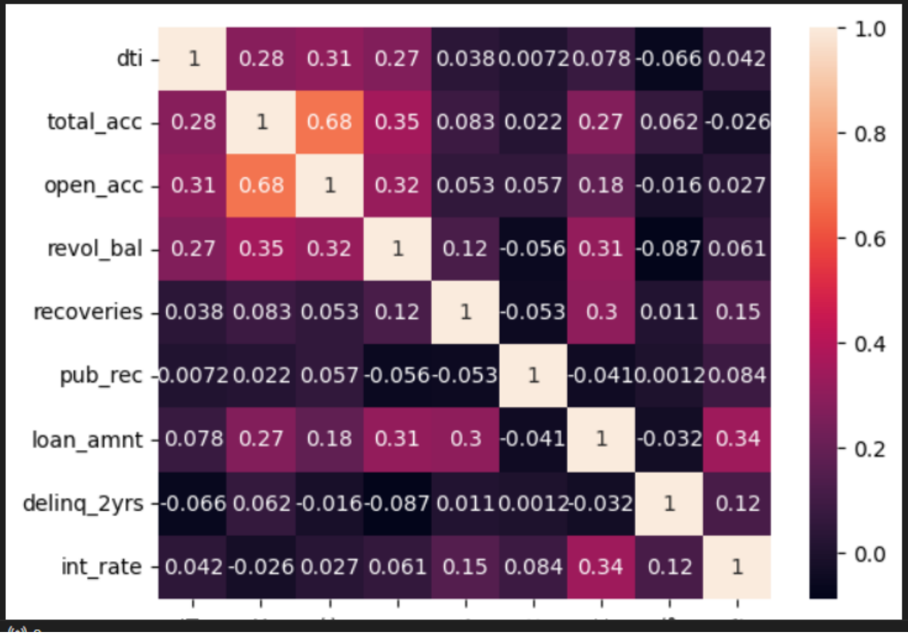


Multivariate Analysis

- By calculating and visualizing the correlation matrix for these variables from the loan_df_chargedoff DataFrame, the code helps identify stronger or newer relationships that could impact the likelihood of a loan charge-off, enhancing the understanding of factors influencing loan defaults.
- The annot=True parameter in the heatmap function ensures that correlation values are displayed on the heatmap, making it easier to interpret the relationships visually.

```
num_cols = ['dti','total_acc','open_acc','revol_bal','recoveries','pub_rec','loan_amnt','delinq_2yrs','int_rate']
sns.heatmap(loan_df_chargedoff[num_cols].corr(),annot=True)
```

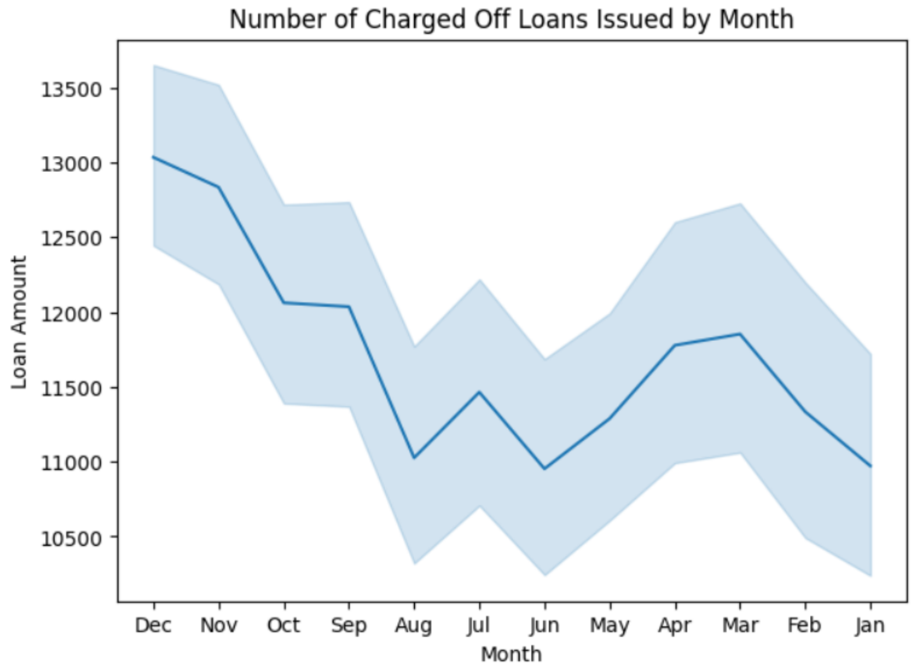
```
<Axes: >
```



Observation 20:
Maximum charged off loans were issued in the month of Dec

The code creates a line plot to visualize the distribution of charged-off loans by month using Matplotlib and Seaborn.

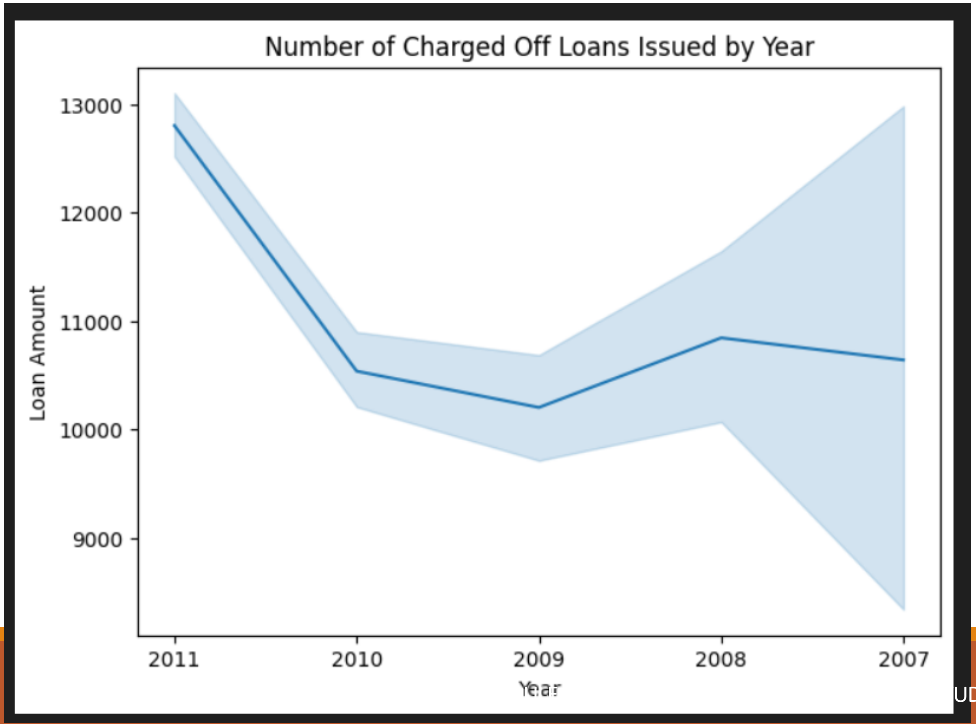
It sets the figure size, plots 'loan_amnt' against 'loan_issue_month' fron `loan_df_chargedoff`, and labels the axes and title, helping identify the peak month for loan defaults.




```
# Using derived column month to plot charged off loans to identify the month where loans taken was maximum.

fig, ax = plt.subplots(figsize=(7,5))
loan_issue_month = sns.lineplot(data =loan_df_chargedoff,y='loan_amnt', x='loan_issue_month')
loan_issue_month.set_xlabel("Month")
loan_issue_month.set_ylabel("Loan Amount")
loan_issue_month.set_title('Number of Charged Off Loans Issued by Month')
```

Obsevation 21:
Maximum charged off loans were issued in the year 2011

```
# Using derived column year to plot charged off loans to identify the month where loans taken was maximum.

fig, ax = plt.subplots(figsize=(7,5))
loan_issue_month = sns.lineplot(data =loan_df_chargedoff,y='loan_amnt', x='loan_issue_year')
loan_issue_month.set_xlabel("Year")
loan_issue_month.set_ylabel("Loan Amount")
loan_issue_month.set_title('Number of Charged Off Loans Issued by Year')
```



Conclusion

The completion of this assignment has provided an understanding of how real business problems are addressed using EDA. In this case study, techniques learned in EDA were applied, and a basic comprehension of risk analytics in banking and financial services was developed. It was also understood how data is utilized to minimize the risk of financial losses while lending to customers.

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

In this case study, EDA was utilized to understand how consumer attributes and loan attributes influence the tendency to default.