A CAL Project Report

on

"Advance Credit Lending Fault Detection System Using Data Mining Techniques"

to be submitted in partial fulfilling of the requirements for the course on

Data Mining and Business Intelligence – ITA5007

(B1+TB1)

by

Joyshree Dutta 21MCA0203

Ritik Gupta 21MCA0249

Trishita Das 21MCA0250

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Winter Semester 2021-2022

TABLE OF CONTENTS

ABSTRACT

Banks earn a major revenue from lending loans. But it is often associated with risk. The borrowers may default on the loan. Interest on loans and associated fees are some of the biggest revenue sources for most banks and credit unions. More than 44 million borrowers collectively owe about $3.5 trillion in total outstanding consumer credit. As of October 2015, However, more than 1 million people default on loans each year. Considering the magnitude of risk and financial loss involved, it is essential for banks to give loans to credible applicants who are highly likely to pay back the loan amount. The objective of my project is to assess the likelihood of loan default based on customer demographics and financial data. To mitigate this issue, we have collected past data on the loan borrowers & would like to develop a strong ML Model to classify if any new borrower is likely to default or not. The dataset is enormous & consists of multiple deterministic factors like borrower's income, gender, loan purpose etc.

1. INTRODUCTION

Banks are essential bodies of the society which not only serve as a secure vault for storing money but also help people in time of need by providing loans and credit. Banks provide loans based on credit history and credibility of the applicant. A loan default occurs when a borrower takes money from a bank and does not repay the loan. People often default on loans due to various reasons. Borrowers who default on loans not only damage their credit but also risk being sued and having their wages garnished. However, the rate of loan default is increasing exponentially.

According to Forbes, say that nearly 40 percent of borrowers are expected to default on their student loans by 2023. Considering the statistics, instead of making money from loan interest, banks will suffer a huge capital loss. Though lending loans is quite beneficial for both the parties, the activity does carry great risks. These risks represent the inability of a borrower to pay back the loan by the designated time which was decided mutually by both the lender and the borrower and it is referred to as 'Credit Risk'. For that, it is highly necessary to assess the clients' credit suitability before authorizing a loan. In the traditional lending process, banking authorities mainly adopt the '5C principle', i.e., Character, Capital, Capacity, Collateral, and Conditions to evaluate a borrower. This evaluation mainly relied on personal experience and knowledge of customer dealing. This method has great limitations. Banks and large financial firms approve loan requests after a verification and validation process of regress, but still, there's no guarantee that the applicant selected after the process will be able to repay the loan in time. In order to prevent the loss, it is very important to have a system in place which will accurately predict the loan defaulters even before approving the loan.

Therefore, addressing the aforementioned scenario, the goal of this project is to discuss the application of different machine learning models in the loan lending process and work out the best approach for a financial institution which accurately identifies whom to lend loan to and help banks identify the loan defaulters for much-reduced credit risk.

2. REVIEW-1 (Survey & Dataset Collection)

DATA DESCRIPTION: -

- Data is in form of excel worksheet.
- Worksheet contains 396k rows and 27 columns.
- Data has housing and customer employment information such as housing ownership, years in job and annual income.

	LoanStatNew	Description
0	loan_amnt	The listed amount of the loan applied for by t...
1	term	The number of payments on the loan. Values are...
2	int_rate	Interest Rate on the loan
3	installment	The monthly payment owed by the borrower if th...
4	grade	LC assigned loan grade
5	sub_grade	LC assigned loan subgrade
6	emp_title	The job title supplied by the Borrower when ap...
7	emp_length	Employment length in years. Possible values ar...
8	home_ownership	The home ownership status provided by the borr...
9	annual_inc	The self-reported annual income provided by th...
10	verification_status	Indicates if income was verified by LC, not ve...
11	issue_d	The month which the loan was funded

12	loan_status	Current status of the loan
13	purpose	A category provided by the borrower for the lo...
14	title	The loan title provided by the borrower
15	zip_code	The first 3 numbers of the zip code provided b...
16	addr_state	The state provided by the borrower in the loan...
17	dti	A ratio calculated using the borrower's total ...
18	earliest_cr_line	The month the borrower's earliest reported cre...
19	open_acc	The number of open credit lines in the borrowe...
20	pub_rec	Number of derogatory public records
21	revol_bal	Total credit revolving balance
22	revol_util	Revolving line utilization rate, or the amount...
23	total_acc	The total number of credit lines currently in ...
24	initial_list_status	The initial listing status of the loan. Possib...
25	application_type	Indicates whether the loan is an individual ap...
26	mort_acc	Number of mortgage accounts.
27	pub_rec_bankruptcies	Number of public record bankruptcies

3. REVIEW-2 (Design)

METHODLOGY: -

The scope of this project is to implement and investigate how different supervised classification methods impact default prediction. The model evaluation techniques used in this project are limited to precision, sensitivity. The classifiers that will be implemented and studied are:

- Logistic Regression
- Random Forest
- Naïve Bayes
- Linear Support Vector Machine

DATA EXPLORATION: -

```
In [6]: df.head().T # df.head.T to get all the columns of the data
```

Out[6]:

	0	1	2	3	4
loan_amnt	10000.0	8000.0	15600.0	7200.0	24375.0
term	36 months	36 months	36 months	36 months	60 months
int_rate	11.44	11.99	10.49	6.49	17.27
installment	329.48	265.68	506.97	220.65	609.33
grade	B	B	B	A	C
sub_grade	B4	B5	B3	A2	C5
emp_title	Marketing	Credit analyst	Statistician	Client Advocate	Destiny Management Inc.
emp_length	10+ years	4 years	< 1 year	6 years	9 years
home_ownership	RENT	MORTGAGE	RENT	RENT	MORTGAGE
annual_inc	117000.0	65000.0	43057.0	54000.0	55000.0
verification_status	Not Verified	Not Verified	Source Verified	Not Verified	Verified
issue_d	Jan-2015	Jan-2015	Jan-2015	Nov-2014	Apr-2013
loan_status	Fully Paid	Fully Paid	Fully Paid	Fully Paid	Charged Off
purpose	vacation	debt_consolidation	credit_card	credit_card	credit_card
title	Vacation	Debt consolidation	Credit card refinancing	Credit card refinancing	Credit Card Refinance
dti	26.24	22.05	12.79	2.6	33.95
earliest_cr_line	Jun-1990	Jul-2004	Aug-2007	Sep-2006	Mar-1999
open_acc	16.0	17.0	13.0	6.0	13.0
pub_rec	0.0	0.0	0.0	0.0	0.0
revol_bal	36369.0	20131.0	11987.0	5472.0	24584.0
revol_util	41.8	53.3	92.2	21.5	69.8
total_acc	25.0	27.0	26.0	13.0	43.0
initial_list_status	w	f	f	f	f
application_type	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL
mort_acc	0.0	3.0	0.0	0.0	1.0
pub_rec_bankruptcies	0.0	0.0	0.0	0.0	0.0
address	0174 Michelle Gateway\nMendozaberg, OK 22690	1076 Carney Fort Apt. 347\nLoganmouth, SD 05113	87025 Mark Dale Apt. 269\nNew Sabrina, WV 05113	823 Reid Ford\nDelacruzside, MA 00813	679 Luna Roads\nGreggshire, VA 11650

most of the columns are categorical in nature and the First thing we should always do is to look into the distribution of target variable The "target variable" is the variable whose values are to be modeled and predicted by other variables.

Count-Plot:

```
Out[7]: <AxesSubplot:xlabel='loan_status', ylabel='count'>
```

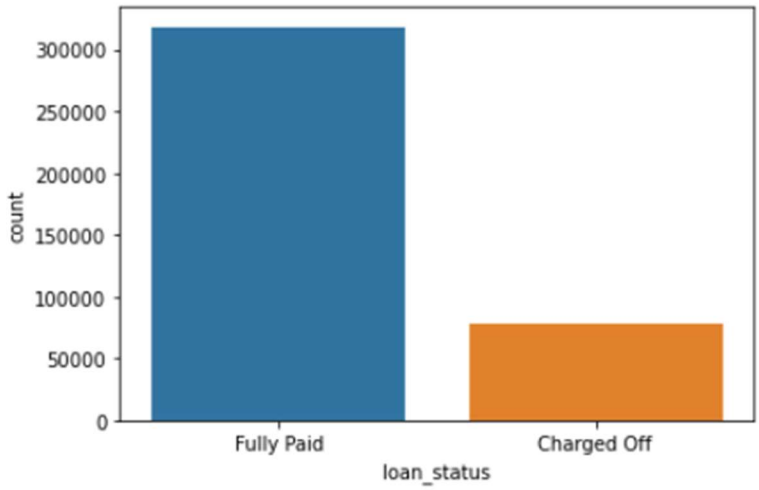
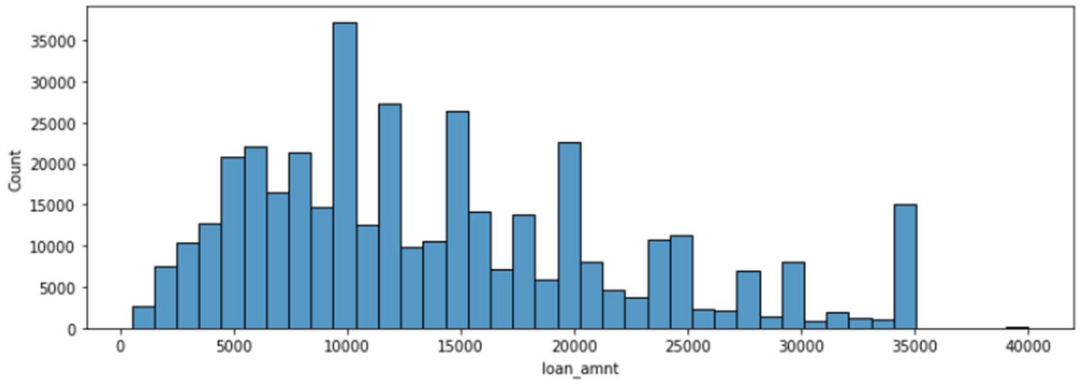


Figure 1 sns.countplot(x="loan_status",data=df)

Histogram:

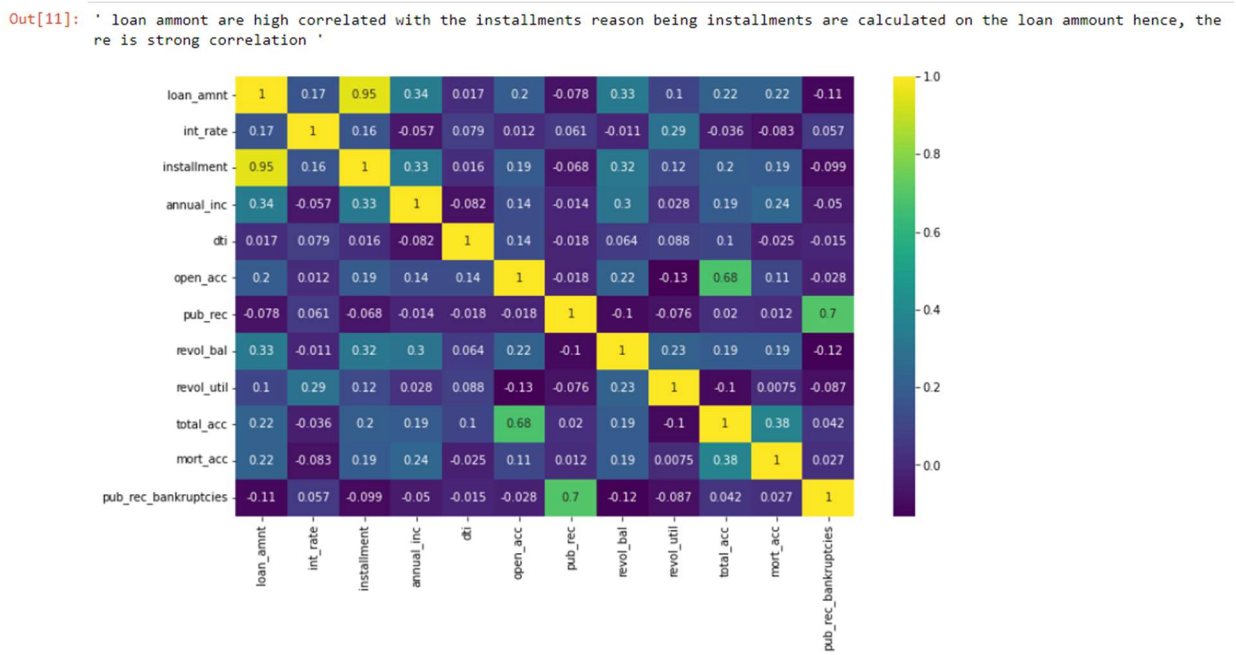
visualization of the loan amount since it is a continuous variable

```
Out[10]: <AxesSubplot:xlabel='loan_amnt', ylabel='Count'>
```



Heat-map

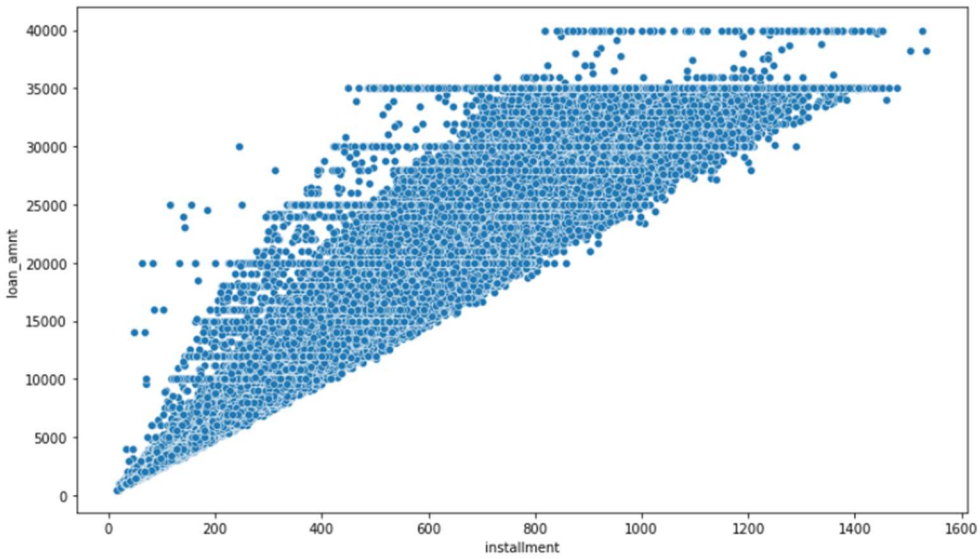
Figure 2heatmap- better way to visualize the correlation matrix



loan amount are high correlated with the installments reason being installments are calculated on the loan amount hence, there is strong correlation.

Scatter Plot:

scatter plot gives more clarity how the relationship is working

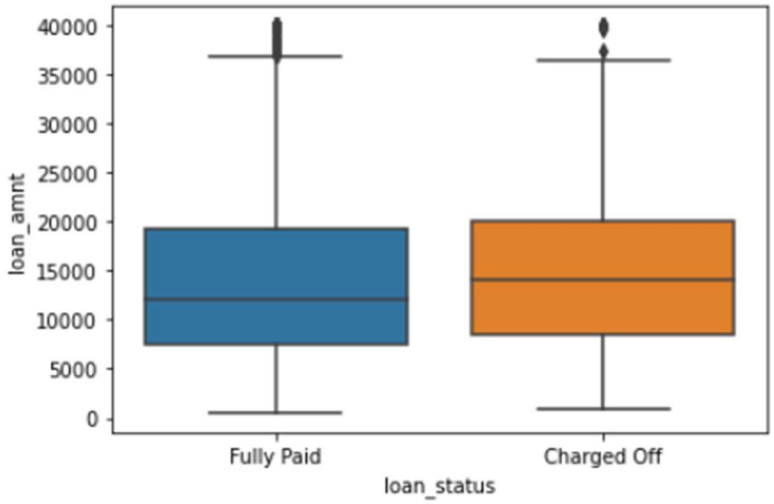


we can conclude that the loan installments is different for same loan amount and their can be many factor like s omeone has taken a loan for period of 10 year and someone took loan for 15 year we make use of no of years a nd the int rate.

Box-Plot

Box plot uses for outliers detection

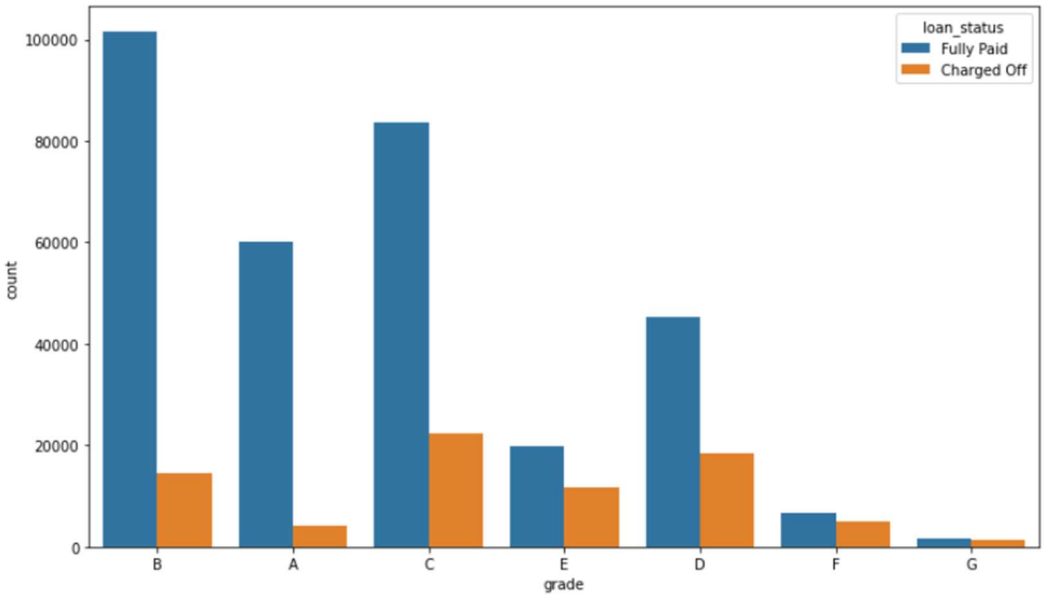
```
Out[13]: <AxesSubplot:xlabel='loan_status', ylabel='loan_amnt'>
```



Count-plot:

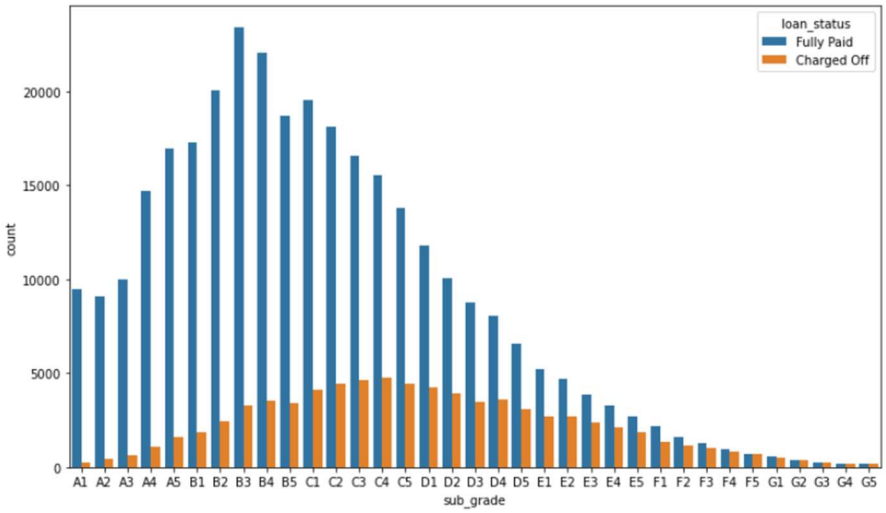
To know how grading affect the loan

```
Out[18]: <AxesSubplot:xlabel='grade', ylabel='count'>
```

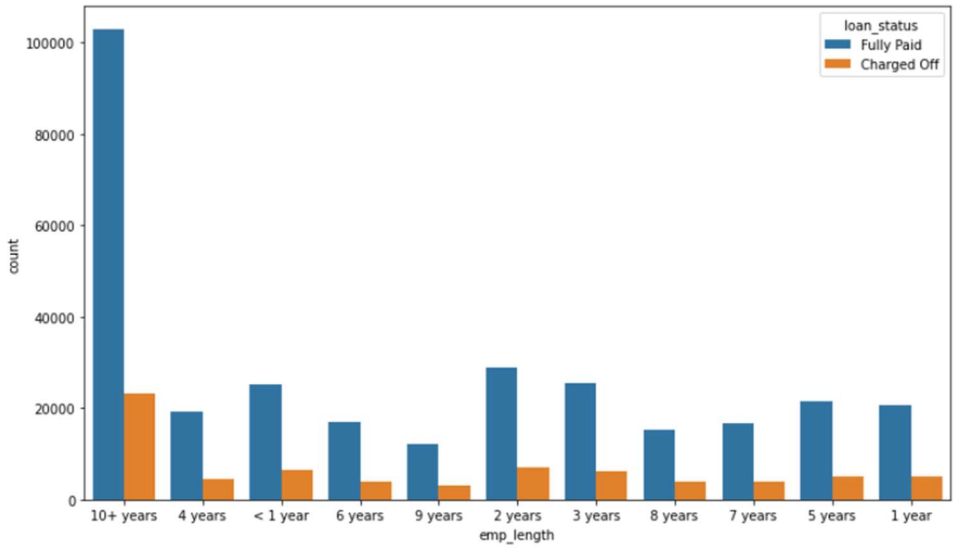


In [24]:
```
plt.figure(figsize=(12,7))
sns.countplot(x= "sub_grade", hue = "loan_status",order=sub_order, data=df)
```

Out[24]: <AxesSubplot:xlabel='sub_grade', ylabel='count'>



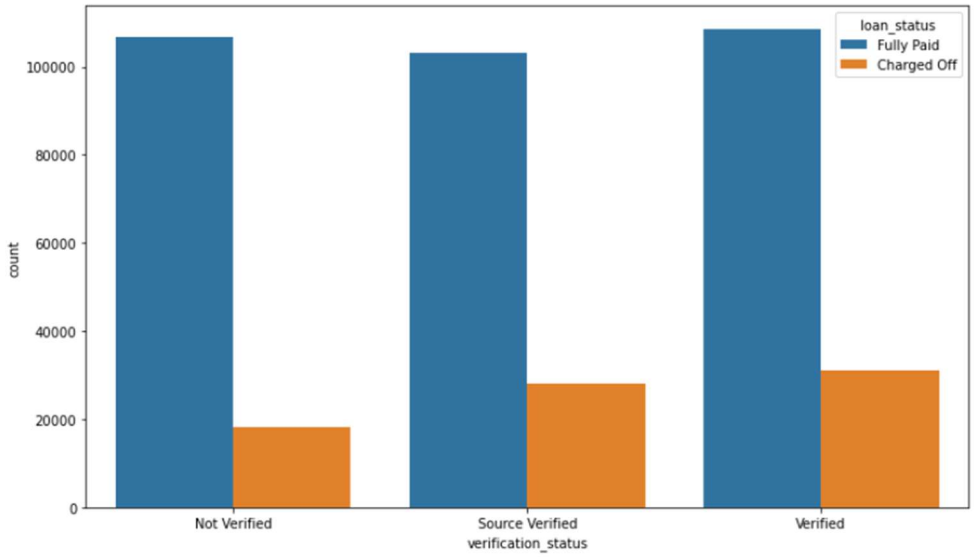
Out[28]: <AxesSubplot:xlabel='emp_length', ylabel='count'>



How employee length affect the loan status

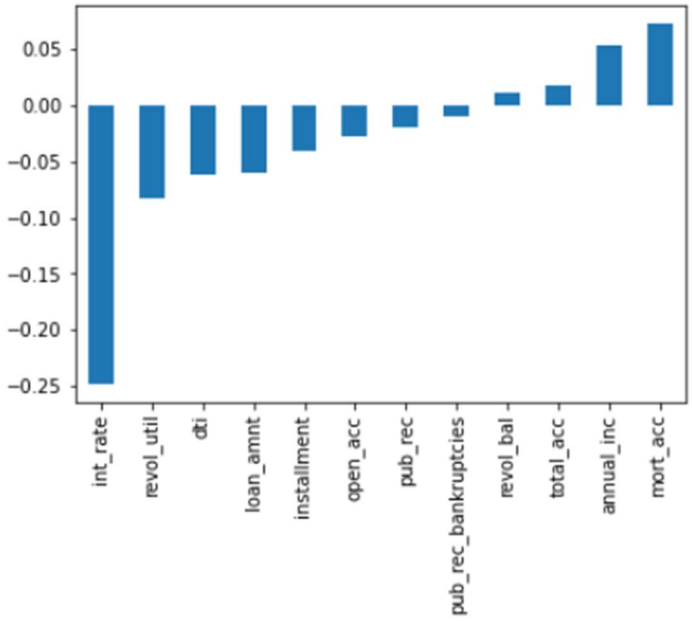
For the verification status

```
Out[29]: <AxesSubplot:xlabel='verification_status', ylabel='count'>
```



the above diagram shows the distribution for note verified, source verified, verified is almost similar the graph shows the company verification process is not as efficient as it should be there should be decreased the count of charged off for source verified.

```
Out[33]: <AxesSubplot:>
```



REVIEW-3 (Code)

Data Mining Project

Advance Credit Lending Fault Detection System Using Data Mining Technique

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

In [2]:
```
df = pd.read_csv("lending_club_loan_two.csv")
```

In [3]:
```
meta_data = pd.read_csv("lending_club_info.csv")
meta_data
```

Out[3]:

	LoanStatNew	Description
0	loan_amnt	The listed amount of the loan applied for by t...
1	term	The number of payments on the loan. Values are...
2	int_rate	Interest Rate on the loan
3	installment	The monthly payment owed by the borrower if th...
4	grade	LC assigned loan grade
5	sub_grade	LC assigned loan subgrade
6	emp_title	The job title supplied by the Borrower when ap...
7	emp_length	Employment length in years. Possible values ar...
8	home_ownership	The home ownership status provided by the borr...
9	annual_inc	The self-reported annual income provided by th...
10	verification_status	Indicates if income was verified by LC, not ve...
11	issue_d	The month which the loan was funded
12	loan_status	Current status of the loan
13	purpose	A category provided by the borrower for the lo...
14	title	The loan title provided by the borrower
15	zip_code	The first 3 numbers of the zip code provided b...
16	addr_state	The state provided by the borrower in the loan...
17	dti	A ratio calculated using the borrower's total ...
18	earliest_cr_line	The month the borrower's earliest reported cre...

	LoanStatNew	Description
19	open_acc	The number of open credit lines in the borrowe...
20	pub_rec	Number of derogatory public records
21	revol_bal	Total credit revolving balance
22	revol_util	Revolving line utilization rate, or the amount...
23	total_acc	The total number of credit lines currently in ...
24	initial_list_status	The initial listing status of the loan. Possib...
25	application_type	Indicates whether the loan is an individual ap...
26	mort_acc	Number of mortgage accounts.
27	pub_rec_bankruptcies	Number of public record bankruptcies

In [4]:
```
meta_data["Description"][4]
# for description of data
```

Out[4]:
```
'LC assigned loan grade'
```

In [5]:
```
df.shape
# to get the shape of the data
# we have 396k Rows & 27 columns which means we have 26 features
```

Out[5]:
```
(396030, 27)
```

In [6]:
```
df.head().T # df.head.T to get all the columns of the data
```

Out[6]:

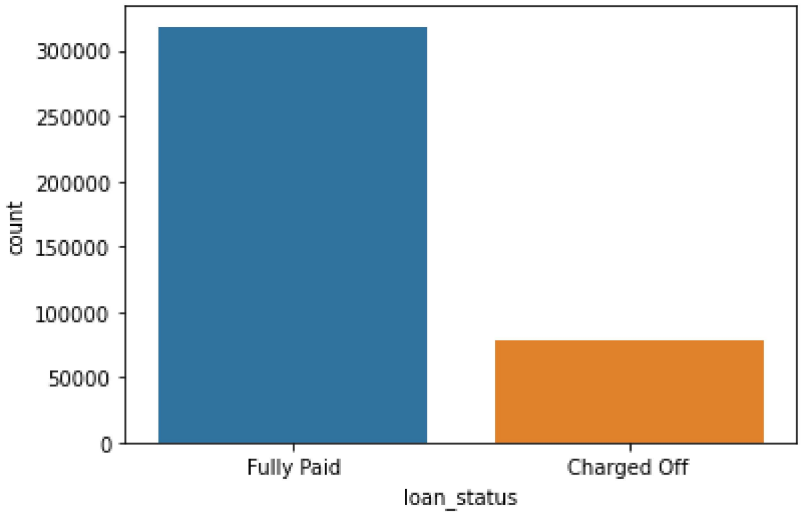
	0	1	2	3	
loan_amnt	10000.0	8000.0	15600.0	7200.0	
term	36 months	36 months	36 months	36 months	
int_rate	11.44	11.99	10.49	6.49	
installment	329.48	265.68	506.97	220.65	
grade	B	B	B	A	
sub_grade	B4	B5	B3	A2	
emp_title	Marketing	Credit analyst	Statistician	Client Advocate	Mana
emp_length	10+ years	4 years	< 1 year	6 years	
home_ownership	RENT	MORTGAGE	RENT	RENT	
annual_inc	117000.0	65000.0	43057.0	54000.0	
verification_status	Not Verified	Not Verified	Source Verified	Not Verified	
issue_d	Jan-2015	Jan-2015	Jan-2015	Nov-2014	

	0	1	2	3	
loan_status	Fully Paid	Fully Paid	Fully Paid	Fully Paid	
purpose	vacation	debt_consolidation	credit_card	credit_card	
title	Vacation	Debt consolidation	Credit card refinancing	Credit card refinancing	
dti	26.24	22.05	12.79	2.6	
earliest_cr_line	Jun-1990	Jul-2004	Aug-2007	Sep-2006	
open_acc	16.0	17.0	13.0	6.0	
pub_rec	0.0	0.0	0.0	0.0	
revol_bal	36369.0	20131.0	11987.0	5472.0	
revol_util	41.8	53.3	92.2	21.5	
total_acc	25.0	27.0	26.0	13.0	
initial_list_status	w	f	f	f	
application_type	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	INDIVIDUAL	
mort_acc	0.0	3.0	0.0	0.0	
pub_rec_bankruptcies	0.0	0.0	0.0	0.0	
address	0174 Michelle Gateway\nMendozaberg, OK 22690	1076 Carney Fort Apt. 347\nLoganmouth, SD 05113	87025 Mark Dale Apt. 269\nNew Sabrina, WV 05113	823 Reid Ford\nDelacruzside, MA 00813	Roads\

most of the columns are categorical in nature and the First thing we should always do is to look into the distribution of target variable The “target variable” is the variable whose values are to be modeled and predicted by other variables.

In [7]:
```
# our target variable is categorical in nature we are going to count plot using the sea
sns.countplot(x="loan_status",data=df)
# charged off means didn't paid the loan
```

Out[7]:
```
<AxesSubplot:xlabel='loan_status', ylabel='count'>
```



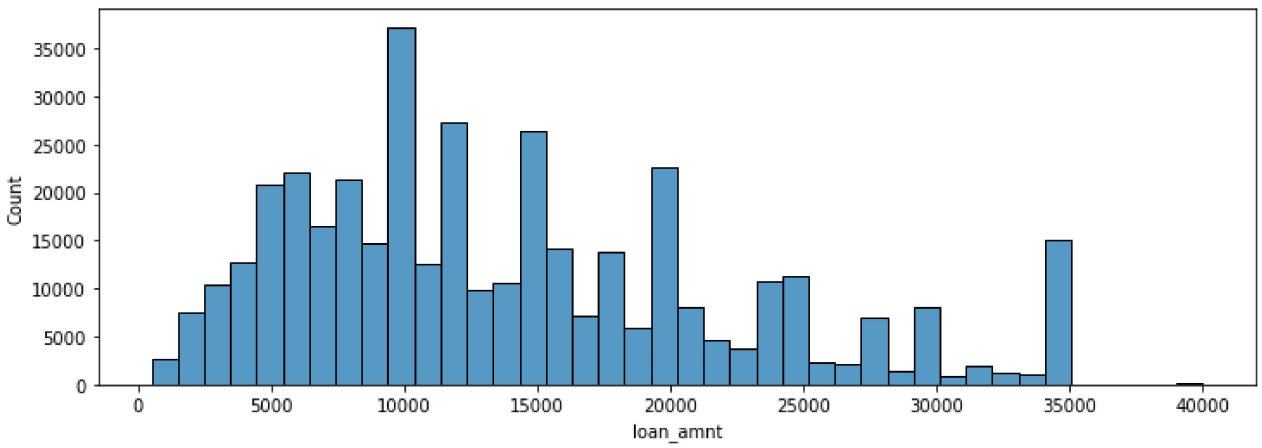
In [8]:
```
# to get the exact of of values we will use the value count methond
values = df['loan_status'].value_counts()
values*100/len(df) # for better understanding we are using percentages
```

Out[8]:
```
Fully Paid     80.387092
Charged Off    19.612908
Name: loan_status, dtype: float64
```

In [9]:
```
# we are having the ratio of 80:20
# this is called highly imbalanced dataset which will lead to more data of full paid so
#fully paid
```

In [10]:
```
# next we will visualize the loan ammount since it is a continous variable we have to u
# we are using the plot size of 12 by 4
plt.figure(figsize=(12,4))
sns.histplot(df["loan_amnt"],bins=40)
# it has right skew distribution positively skewed distri
# 40000 is the outliers
```

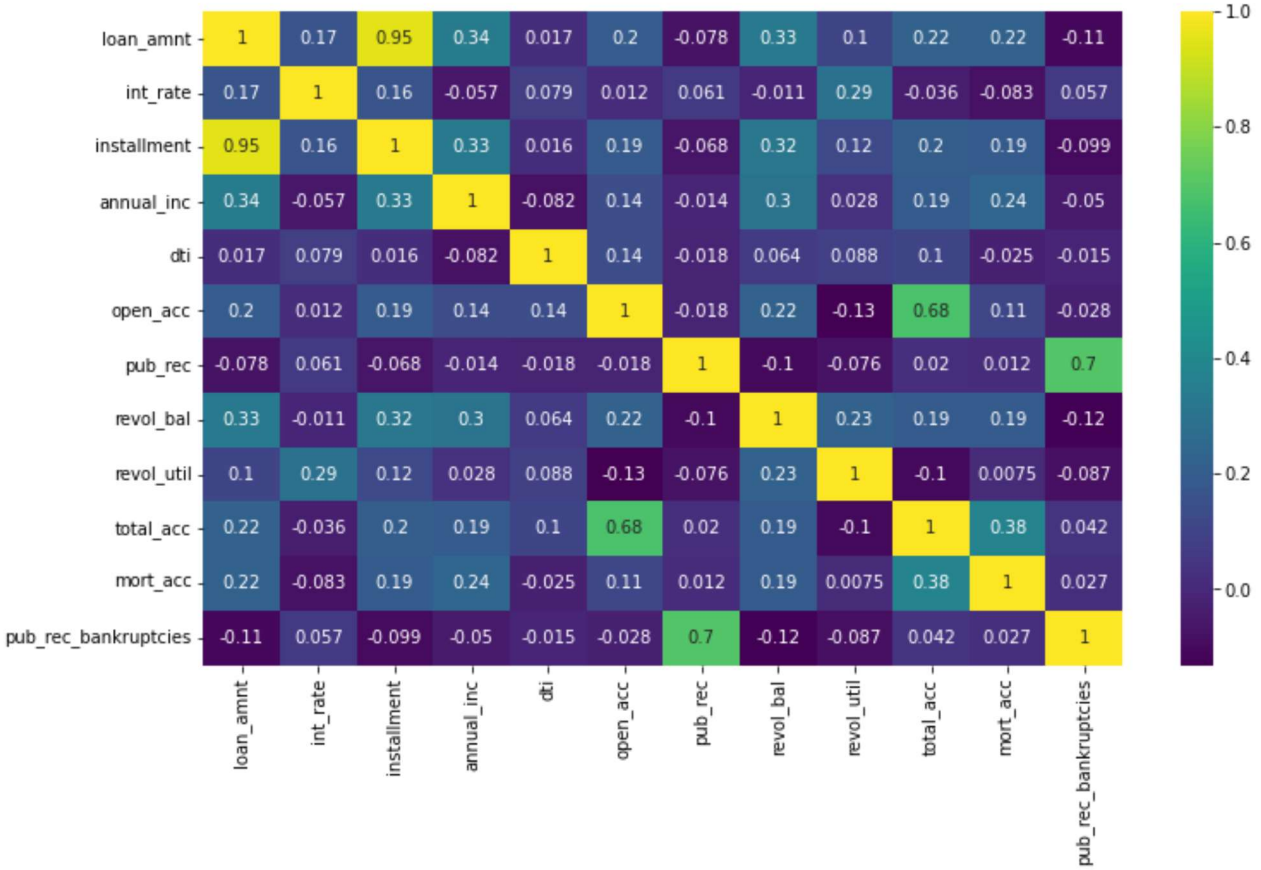
Out[10]:
```
<AxesSubplot:xlabel='loan_amnt', ylabel='Count'>
```



In [11]:
```
# heatmap is the better way to visualize the corelation matrix
plt.figure(figsize=(12,7))
sns.heatmap(df.corr(),annot = True, cmap='viridis') # it takes few parameters first cor
```

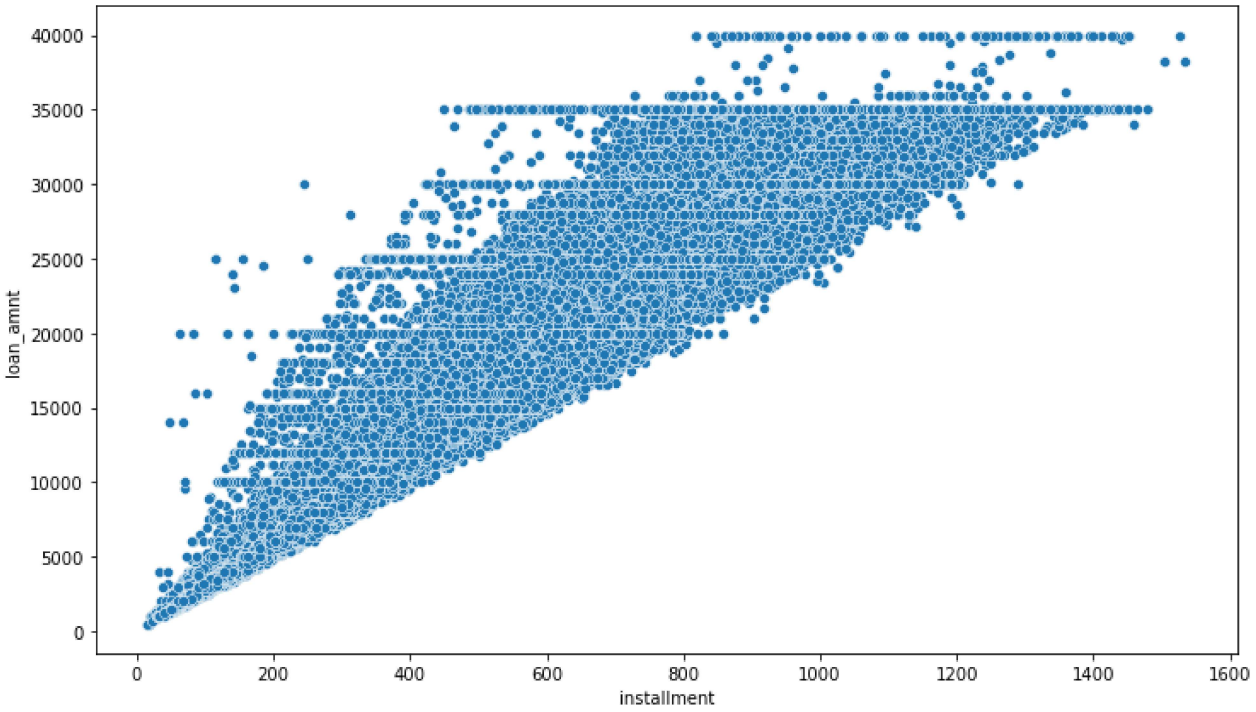
```
# n then the color map for the heat map

# yellow represents the positive correlation
# dark voilet represents the negetive correlation
""" loan ammont are high correlated with the installments reason being installments are
# there is a strong realtion between the installments and the loan amounts
```

Out[11]: ' loan ammont are high correlated with the installments reason being installments are ca
lculated on the loan ammount hence, there is strong correlation '



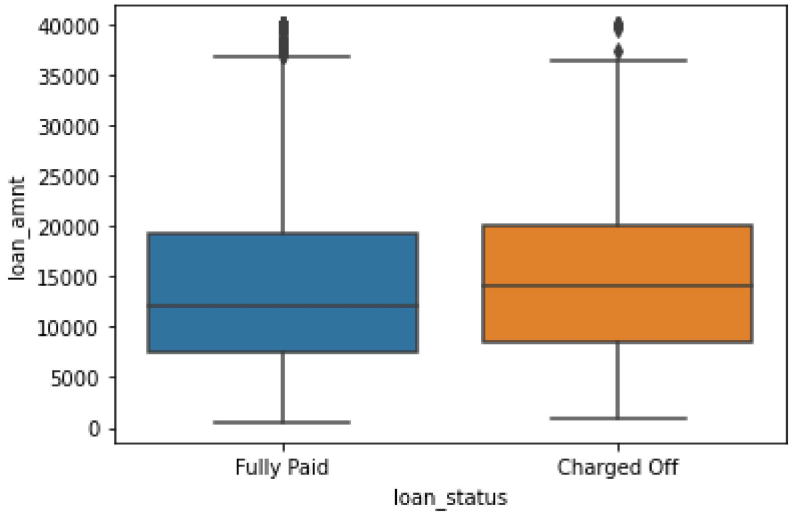
In [12]:
```
# scatter plot gives more clarity how the relationship is working
plt.figure(figsize=(12,7))
sns.scatterplot(x="installment", y="loan_amnt", data=df)

'''we can conclude that the loan installments is different for same loan ammount
and their can be many factor like someone has taken a loan for period of 10 year and so

we make use of no of years and the int rate
'''
```

Out[12]: 'we can conclude that the loan installments is different for same loan ammount \nand the
ir can be many factor like someone has taken a loan for period of 10 year and someone to
ok loan for 15 year\n\nwe make use of no of years and the int rate\n'



In [13]:
```
# next are gonna see how loan ammount affect the loan status
# box plot uses for outliers detection
sns.boxplot(x='loan_status', y='loan_amnt', data=df)
# the line inside the box is median
# median loan ammount is almost equal so based on this we can make an assumption loan a
# whether the person will default or not
```

Out[13]:
```
<AxesSubplot:xlabel='loan_status', ylabel='loan_amnt'>
```



In [14]:
```
df.groupby('loan_status')['loan_amnt'].describe()
# it will give the same summary as the box plot gave
```

Out[14]:

loan_status	count	mean	std	min	25%	50%	75%	max
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0	40000.0

loan_status	count	mean	std	min	25%	50%	75%	max
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

In [15]:
```
# next grade variable
df['grade'] # to check the meta of the grade
```

Out[15]:
```
0         B
1         B
2         B
3         A
4         C
         ..
396025    B
396026    C
396027    B
396028    C
396029    C
Name: grade, Length: 396030, dtype: object
```

In [16]:
```
meta_data["Description"][4]
# it is a grade given to a particular loan
# if  a person is paying everything on time and doing everything right the customer is
```

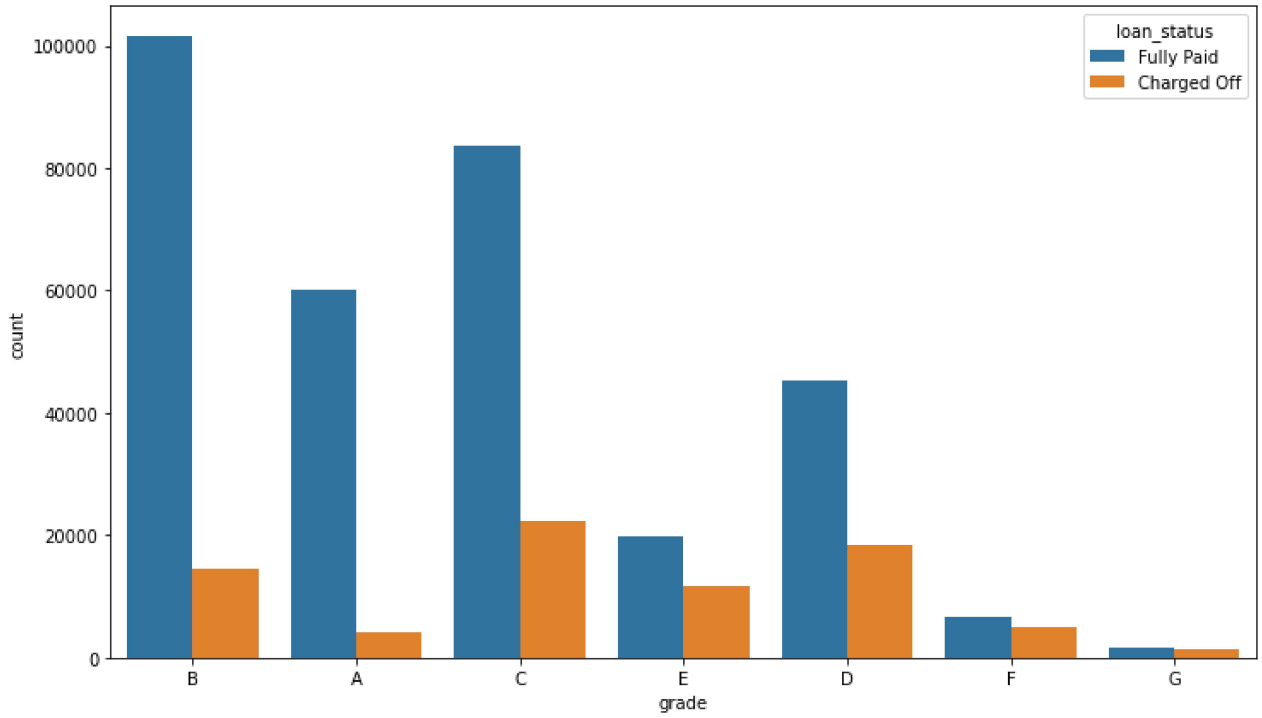
Out[16]:
```
'LC assigned loan grade'
```

In [17]:
```
df['grade'].unique()
```

Out[17]:
```
array(['B', 'A', 'C', 'E', 'D', 'F', 'G'], dtype=object)
```

In [18]:
```
# check how grading affect the loan
plt.figure(figsize=(12,7))
sns.countplot(x= "grade", hue = "loan_status", data=df)
```

Out[18]:
```
<AxesSubplot:xlabel='grade', ylabel='count'>
```



In [19]:
```
# this shows as the grade goes from A -> B, B->C the default rate increases
df[df["grade"]=="A"].groupby("loan_status")["grade"].count()*100/len(df[df["grade"]=="A
# for type A loan the percentage of people defaulted is 6% only
```

Out[19]:
```
loan_status
Charged Off     6.287878
Fully Paid     93.712122
Name: grade, dtype: float64
```

In [20]:
```
df[df["grade"]=="B"].groupby("loan_status")["grade"].count()*100/len(df[df["grade"]=="B
# for type A loan the percentage of people defaulted is 12%
```

Out[20]:
```
loan_status
Charged Off    12.573049
Fully Paid     87.426951
Name: grade, dtype: float64
```

In [21]:
```
df[df["grade"]=="G"].groupby("loan_status")["grade"].count()*100/len(df[df["grade"]=="G
# for type A loan the percentage of people defaulted is 47%
```

Out[21]:
```
loan_status
Charged Off    47.8389
Fully Paid     52.1611
Name: grade, dtype: float64
```

In [22]:
```
df["sub_grade"].unique()
```

Out[22]:
```
array(['B4', 'B5', 'B3', 'A2', 'C5', 'C3', 'A1', 'B2', 'C1', 'A5', 'E4',
       'A4', 'A3', 'D1', 'C2', 'B1', 'D3', 'D5', 'D2', 'E1', 'E2', 'E5',
       'F4', 'E3', 'D4', 'G1', 'F5', 'G2', 'C4', 'F1', 'F3', 'G5', 'G4',
       'F2', 'G3'], dtype=object)
```

In [23]:
```
sub_order=sorted(df["sub_grade"].unique())
```

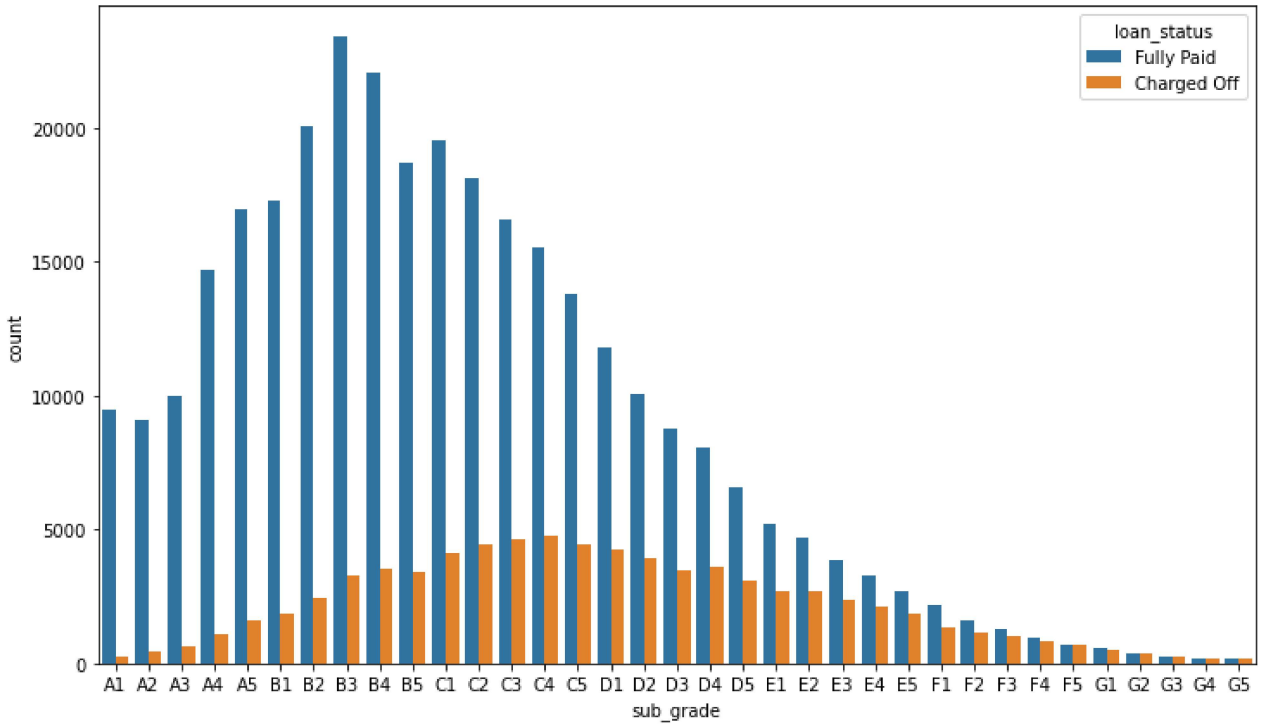
```
sub_order
```

Out[23]:
```
['A1',
 'A2',
 'A3',
 'A4',
 'A5',
 'B1',
 'B2',
 'B3',
 'B4',
 'B5',
 'C1',
 'C2',
 'C3',
 'C4',
 'C5',
 'D1',
 'D2',
 'D3',
 'D4',
 'D5',
 'E1',
 'E2',
 'E3',
 'E4',
 'E5',
 'F1',
 'F2',
 'F3',
 'F4',
 'F5',
 'G1',
 'G2',
 'G3',
 'G4',
 'G5']
```

we are sorting the subgrade

In [24]:
```
plt.figure(figsize=(12,7))
sns.countplot(x= "sub_grade", hue = "loan_status",order=sub_order, data=df)
```

Out[24]:
```
<AxesSubplot:xlabel='sub_grade', ylabel='count'>
```



In [25]:
```
df[df["sub_grade"]=="A1"].groupby("loan_status")["sub_grade"].count()*100/len(df[df["su
```

Out[25]:
```
loan_status
Charged Off     2.867715
Fully Paid     97.132285
Name: sub_grade, dtype: float64
```

In [26]:
```
df[df["sub_grade"]=="A2"].groupby("loan_status")["sub_grade"].count()*100/len(df[df["su
```

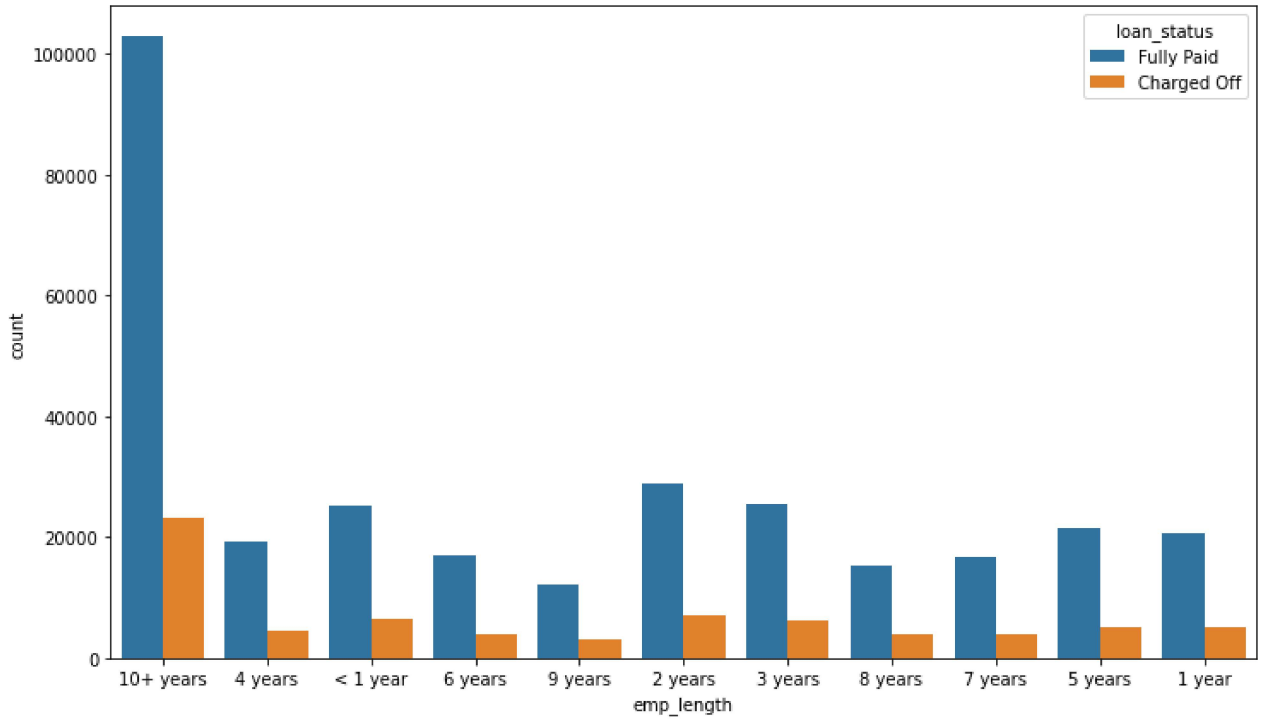
Out[26]:
```
loan_status
Charged Off     4.818647
Fully Paid     95.181353
Name: sub_grade, dtype: float64
```

In [27]:
```
df[df["sub_grade"]=="B2"].groupby("loan_status")["sub_grade"].count()*100/len(df[df["su
```

Out[27]:
```
loan_status
Charged Off    10.8513
Fully Paid     89.1487
Name: sub_grade, dtype: float64
```

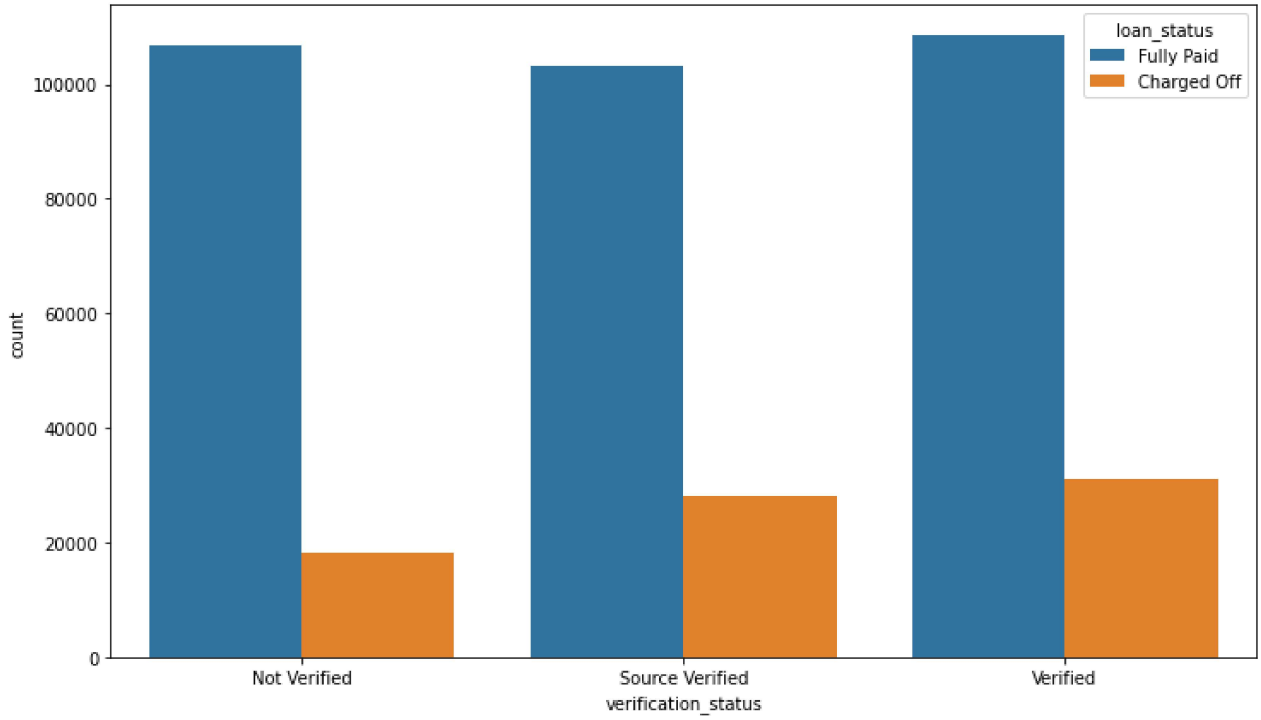
In [28]:
```
# Now how employee length affect the loan status
plt.figure(figsize=(12,7))
sns.countplot(x= "emp_length", hue = "loan_status", data=df)
```

Out[28]:
```
<AxesSubplot:xlabel='emp_length', ylabel='count'>
```



In [29]:
```
#check for the verfication status
plt.figure(figsize=(12,7))
sns.countplot(x= "verification_status", hue = "loan_status", data=df)
```

Out[29]:
```
<AxesSubplot:xlabel='verification_status', ylabel='count'>
```



the above diagram shows the distribution for note verfied, source verified, verified is almost simmilar the graph shows the company verification process is not as effiecient as it should be there should be decreased the the count of charged off for source verified

In [30]:
```
df['loan_status'].unique()
```

Out[30]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

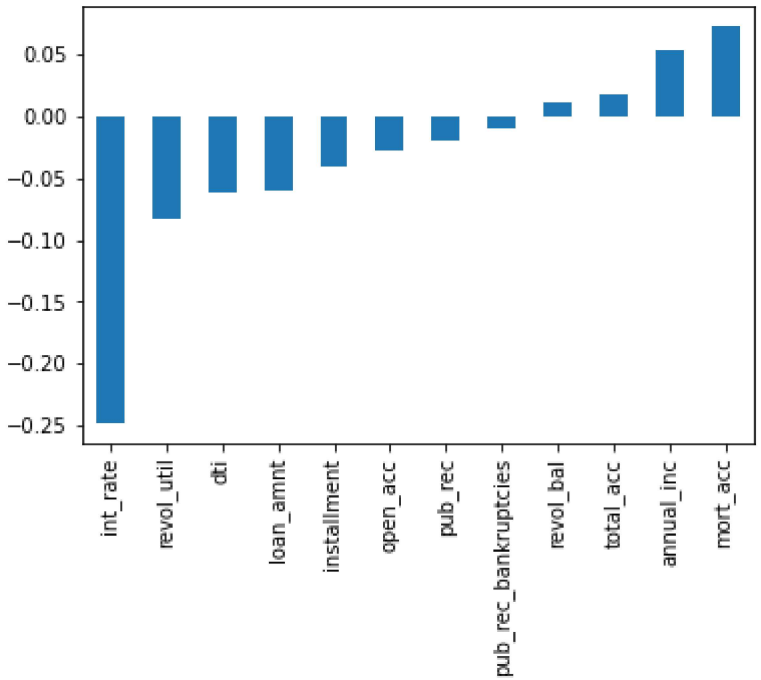
In [31]:
```
df["loan_status"]=df["loan_status"].apply(lambda x:1 if x == "Fully Paid" else 0)
```

In [32]:
```
df['loan_status'].unique()
```

Out[32]:
```
array([1, 0], dtype=int64)
```

In [33]:
```
df.corr()["loan_status"].sort_values().drop("loan_status").plot(kind="bar")
```

Out[33]:
```
<AxesSubplot:>
```



In [34]:
```
# checking the Missing Values
df.isnull().sum()*100/len(df)
```

Out[34]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_title               5.789208
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
```

```
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

Filling Missing Values

Emp title

In [35]:
```
df['emp_title'].nunique()
```

Out[35]:
```
173105
```

In [36]:
```
df.shape
```

Out[36]:
```
(396030, 27)
```

In [37]:
```
df.drop("emp_title", axis=1,inplace= True)
```

In [38]:
```
df.isnull().sum()*100/len(df)
```

Out[38]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
emp_length              4.621115
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
```

```
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

emp_length

In [39]:
```
df['emp_length'].unique()
```

Out[39]:
```
array(['10+ years', '4 years', '< 1 year', '6 years', '9 years',
       '2 years', '3 years', '8 years', '7 years', '5 years', '1 year',
       nan], dtype=object)
```

In [40]:
```
df.drop("emp_length", axis=1,inplace= True)
```

In [41]:
```
df.isnull().sum()*100/len(df)
```

Out[41]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
title                   0.443148
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

title column

In [42]:
```
df['title'].nunique()
```

Out[42]:
```
48817
```

In [43]:
```
# purpose is the duplicate variable same as title with less unique values
df['purpose']
```

Out[43]:
```
0                   vacation
```

```
1         debt_consolidation
2                credit_card
3                credit_card
4                credit_card
                 ...
396025    debt_consolidation
396026    debt_consolidation
396027    debt_consolidation
396028    debt_consolidation
396029    debt_consolidation
Name: purpose, Length: 396030, dtype: object
```

In [44]:
```
df.drop("title", axis=1, inplace= True)
```

In [45]:
```
df.isnull().sum()*100/len(df)
```

Out[45]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
grade                   0.000000
sub_grade               0.000000
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                9.543469
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

mort_acc

we are using mean to fill the missing values

In [46]:
```
df["mort_acc"]=df["mort_acc"].fillna(np.mean(df["mort_acc"]))
```

In [47]:
```
df.isnull().sum()*100/len(df)
```

Out[47]:
```
loan_amnt               0.000000
term                    0.000000
int_rate                0.000000
installment             0.000000
```

```
grade                   0.000000
sub_grade               0.000000
home_ownership          0.000000
annual_inc              0.000000
verification_status     0.000000
issue_d                 0.000000
loan_status             0.000000
purpose                 0.000000
dti                     0.000000
earliest_cr_line        0.000000
open_acc                0.000000
pub_rec                 0.000000
revol_bal               0.000000
revol_util              0.069692
total_acc               0.000000
initial_list_status     0.000000
application_type        0.000000
mort_acc                0.000000
pub_rec_bankruptcies    0.135091
address                 0.000000
dtype: float64
```

dropping the rest values which have the missing data less than the -1

In [48]:
```
df.dropna(inplace=True)
```

In [49]:
```
df.isnull().sum()*100/len(df)
```

Out[49]:
```
loan_amnt               0.0
term                    0.0
int_rate                0.0
installment             0.0
grade                   0.0
sub_grade               0.0
home_ownership          0.0
annual_inc              0.0
verification_status     0.0
issue_d                 0.0
loan_status             0.0
purpose                 0.0
dti                     0.0
earliest_cr_line        0.0
open_acc                0.0
pub_rec                 0.0
revol_bal               0.0
revol_util              0.0
total_acc               0.0
initial_list_status     0.0
application_type        0.0
mort_acc                0.0
pub_rec_bankruptcies    0.0
address                 0.0
dtype: float64
```

Feature Engineering

means when we try to extract the meaningful data from those data which doesnt seem meaningful at all

In [50]:
```
# printing all the categorical data from the data set
df.select_dtypes(['object']).columns
```

Out[50]:
```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'issue_d', 'purpose', 'earliest_cr_line', 'initial_list_status',
       'application_type', 'address'],
      dtype='object')
```

Now, we are trying to grab the features out of all the data present

In [51]:
```
df['term']
```

Out[51]:
```
0          36 months
1          36 months
2          36 months
3          36 months
4          60 months
             ...
396025     60 months
396026     36 months
396027     36 months
396028     60 months
396029     36 months
Name: term, Length: 395219, dtype: object
```

In [52]:
```
df['term'] = df['term'].apply(lambda x : int(x[:3]))
```

In [53]:
```
df['grade']
```

Out[53]:
```
0         B
1         B
2         B
3         A
4         C
         ..
396025    B
396026    C
396027    B
396028    C
396029    C
Name: grade, Length: 395219, dtype: object
```

In [54]:
```
df['sub_grade']
```

Out[54]:
```
0         B4
1         B5
2         B3
3         A2
4         C5
          ..
396025    B4
396026    C1
```

```
396027    B1
396028    C2
396029    C2
Name: sub_grade, Length: 395219, dtype: object
```

sub_grade column derived from grade so we can drop either of them

In [55]:
```
df.drop('sub_grade',axis =1, inplace = True)
```

In [56]:
```
df['home_ownership'].value_counts()
```

Out[56]:
```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

In [57]:
```
df["home_ownership"] = df['home_ownership'].replace(['NONE','ANY'], "OTHER")
```

In [58]:
```
df['home_ownership'].value_counts()
```

Out[58]:
```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          142
Name: home_ownership, dtype: int64
```

In [59]:
```
df['address']
```

Out[59]:
```
0              0174 Michelle Gateway\nMendozaberg, OK 22690
1           1076 Carney Fort Apt. 347\nLoganmouth, SD 05113
2           87025 Mark Dale Apt. 269\nNew Sabrina, WV 05113
3                    823 Reid Ford\nDelacruzside, MA 00813
4                     679 Luna Roads\nGreggshire, VA 11650
                              ...
396025       12951 Williams Crossing\nJohnnyville, DC 30723
396026    0114 Fowler Field Suite 028\nRachelborough, LA...
396027     953 Matthew Points Suite 414\nReedfort, NY 70466
396028    7843 Blake Freeway Apt. 229\nNew Michael, FL 2...
396029          787 Michelle Causeway\nBriannaton, AR 48052
Name: address, Length: 395219, dtype: object
```

In [60]:
```
df['zip'] = df['address'].apply(lambda x: x[-5:])
```

In [61]:
```
df['zip']
```

Out[61]:
```
0         22690
1         05113
2         05113
3         00813
4         11650
```

```
          ...
396025    30723
396026    05113
396027    70466
396028    29597
396029    48052
Name: zip, Length: 395219, dtype: object
```

In [62]:
```
df.drop('address',axis=1, inplace=True)
```

In [63]:
```
df['earliest_cr_line']
```

Out[63]:
```
0         Jun-1990
1         Jul-2004
2         Aug-2007
3         Sep-2006
4         Mar-1999
            ...
396025    Nov-2004
396026    Feb-2006
396027    Mar-1997
396028    Nov-1990
396029    Sep-1998
Name: earliest_cr_line, Length: 395219, dtype: object
```

In [64]:
```
df['earliest_cr_line'] = df['earliest_cr_line'].apply(lambda x : x[-4:])
```

In [65]:
```
df['earliest_cr_line']
```

Out[65]:
```
0         1990
1         2004
2         2007
3         2006
4         1999
          ...
396025    2004
396026    2006
396027    1997
396028    1990
396029    1998
Name: earliest_cr_line, Length: 395219, dtype: object
```

In [66]:
```
df['pub_rec'].value_counts() # 0 is good customer and everything else is bad customer
```

Out[66]:
```
0.0     337489
1.0      49713
2.0       5474
3.0       1521
4.0        527
5.0        237
6.0        122
7.0         56
8.0         34
9.0         12
10.0        11
```

```
11.0         8
13.0         4
12.0         4
19.0         2
40.0         1
17.0         1
86.0         1
24.0         1
15.0         1
Name: pub_rec, dtype: int64
```

In [67]:
```
df['pub_rec'] = df['pub_rec'].apply(lambda x : 0 if x==0 else 1)
```

In [68]:
```
df['pub_rec'].value_counts()
```

Out[68]:
```
0    337489
1     57730
Name: pub_rec, dtype: int64
```

In [69]:
```
df
```

Out[69]:

	loan_amnt	term	int_rate	installment	grade	home_ownership	annual_inc	verification_status
0	10000.0	36	11.44	329.48	B	RENT	117000.0	Not Verified
1	8000.0	36	11.99	265.68	B	MORTGAGE	65000.0	Not Verified
2	15600.0	36	10.49	506.97	B	RENT	43057.0	Source Verified
3	7200.0	36	6.49	220.65	A	RENT	54000.0	Not Verified
4	24375.0	60	17.27	609.33	C	MORTGAGE	55000.0	Verified
...	...	...	...	...	...	...	...	...
396025	10000.0	60	10.99	217.38	B	RENT	40000.0	Source Verified
396026	21000.0	36	12.29	700.42	C	MORTGAGE	110000.0	Source Verified
396027	5000.0	36	9.99	161.32	B	RENT	56500.0	Verified
396028	21000.0	60	15.31	503.02	C	MORTGAGE	64000.0	Verified
396029	2000.0	36	13.61	67.98	C	RENT	42996.0	Verified

395219 rows × 23 columns

Converting Cateogorical Data into Numerical

we will use the Numerical Encoding

In [70]:
```
cols = df.select_dtypes(['object']).columns
cols
```

Out[70]:
```
Index(['grade', 'home_ownership', 'verification_status', 'issue_d', 'purpose',
       'earliest_cr_line', 'initial_list_status', 'application_type', 'zip'],
      dtype='object')
```

In [71]:
```
def ordinal_mapper(data,var):
    ordinal_map = {k:i for i,k in enumerate(data[var].unique(),0)}
    data[var] = data[var].map(ordinal_map)
```

In [72]:
```
for var in cols:
    ordinal_mapper(df,var)
```

In [73]:
```
df[cols]
```

Out[73]:

	grade	home_ownership	verification_status	issue_d	purpose	earliest_cr_line	initial_list_status
0	0	0	0	0	0	0	0
1	0	1	0	0	1	1	1
2	0	0	1	0	2	2	1
3	1	0	0	1	2	3	1
4	2	1	2	2	2	4	1
...	...	...	...	...	...	...	...
396025	0	0	1	14	1	1	0
396026	2	1	1	23	1	3	1
396027	0	0	2	21	1	7	1
396028	2	1	2	55	1	0	1
396029	2	0	2	80	1	16	1

395219 rows × 9 columns

Train Test Split

In [74]:
```
data=df.sample(frac=.5,random_state=101)
```

In [75]:
```
from sklearn.model_selection import train_test_split
```

In [76]:
```
X = data.drop('loan_status', axis=1).values # feature is everything except the loan sta
y = data['loan_status'].values # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1
```

Normalizing the Data

In [77]:
```
from sklearn.preprocessing import MinMaxScaler # to scale the dataset
```

In [78]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
 # Next We have to fit the MinMax scaler on the traning data as well as transform it
```

In [79]:
```
X_train
```

Out[79]:
```
array([[0.62025316, 0.        , 0.13946241, ..., 0.02941176, 0.        ,
        0.22222222],
       [0.09367089, 0.        , 0.25983638, ..., 0.05335267, 0.        ,
        0.44444444],
       [0.16455696, 1.        , 0.33735878, ..., 0.05335267, 0.        ,
        0.22222222],
       ...,
       [0.59493671, 0.        , 0.22088041, ..., 0.02941176, 0.125     ,
        0.11111111],
       [0.18797468, 0.        , 0.30346708, ..., 0.08823529, 0.        ,
        0.11111111],
       [0.36708861, 0.        , 0.25983638, ..., 0.05882353, 0.        ,
        0.22222222]])
```

In [ ]:
```

```

Modelling

In [80]:
```
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix
```

In [81]:
```
algo = [LinearSVC(),
        LogisticRegression(random_state = 0 ,solver = "lbfgs", max_iter=1000),
        GaussianNB(),
        RandomForestClassifier(random_state=0, n_estimators=10)]
```

In [84]:
```
def model_clf(x):
    cl = x # instanciating the model
    cl.fit(X_train, y_train) # fit the model & pass the data
    predict = cl.predict(X_test) # prediction on test data
    print("-------classification Report---------")
    print(classification_report(y_test,predict))
    confusion = confusion_matrix(y_test, predict)
    print('--------Confusion Matrix-----------\n',x)
    print(confusion)
    print('-----------xxxxxxxxxxxx-----------------')
```

In [85]:
```
for alg in algo:
    model_clf(alg)
```

```
-------classification Report---------
              precision    recall  f1-score   support

           0       0.73      0.11      0.19      7947
           1       0.82      0.99      0.89     31575

    accuracy                           0.81     39522
   macro avg       0.77      0.55      0.54     39522
weighted avg       0.80      0.81      0.75     39522

--------Confusion Matrix-----------
 LinearSVC()
[[  871  7076]
 [  329 31246]]
-----------xxxxxxxxxxxx-----------------
-------classification Report---------
              precision    recall  f1-score   support

           0       0.66      0.19      0.29      7947
           1       0.83      0.98      0.90     31575

    accuracy                           0.82     39522
   macro avg       0.74      0.58      0.59     39522
weighted avg       0.79      0.82      0.77     39522

--------Confusion Matrix-----------
 LogisticRegression(max_iter=1000, random_state=0)
[[ 1475  6472]
 [  754 30821]]
-----------xxxxxxxxxxxx-----------------
-------classification Report---------
              precision    recall  f1-score   support

           0       0.45      0.38      0.41      7947
           1       0.85      0.88      0.87     31575

    accuracy                           0.78     39522
   macro avg       0.65      0.63      0.64     39522
weighted avg       0.77      0.78      0.77     39522

--------Confusion Matrix-----------
 GaussianNB()
[[ 3026  4921]
 [ 3699 27876]]
```

```
-----------xxxxxxxxxxx-----------------
-------classification Report---------
              precision    recall  f1-score   support

           0       0.78      0.51      0.62      7947
           1       0.89      0.96      0.92     31575

    accuracy                           0.87     39522
   macro avg       0.83      0.74      0.77     39522
weighted avg       0.86      0.87      0.86     39522

--------Confusion Matrix-----------
 RandomForestClassifier(n_estimators=10, random_state=0)
[[ 4050  3897]
 [ 1150 30425]]
-----------xxxxxxxxxxx-----------------
```

Random Forest can work as good as Neural Network in this case

CONCLUSION

Linear SVC

```
-------classification Report--------- LinearSVC()
              precision    recall  f1-score   support

           0       0.73      0.11      0.19      7947
           1       0.82      0.99      0.89     31575

    accuracy                           0.81     39522
   macro avg       0.77      0.55      0.54     39522
weighted avg       0.80      0.81      0.75     39522

--------Confusion Matrix-----------
 LinearSVC()
[[  871  7076]
 [  329 31246]]
-----------xxxxxxxxxxxx-----------------
```

Logistics Regression

```
-------classification Report--------- LogisticRegression(max_iter=1000, random_state=0)
              precision    recall  f1-score   support

           0       0.66      0.19      0.29      7947
           1       0.83      0.98      0.90     31575

    accuracy                           0.82     39522
   macro avg       0.74      0.58      0.59     39522
weighted avg       0.79      0.82      0.77     39522

--------Confusion Matrix-----------
 LogisticRegression(max_iter=1000, random_state=0)
[[ 1475  6472]
 [  754 30821]]
-----------xxxxxxxxxxxx-----------------
```

Naïve- bayes

```
-------classification Report--------- GaussianNB()
              precision    recall  f1-score   support

           0       0.45      0.38      0.41      7947
           1       0.85      0.88      0.87     31575

    accuracy                           0.78     39522
   macro avg       0.65      0.63      0.64     39522
weighted avg       0.77      0.78      0.77     39522

--------Confusion Matrix-----------
 GaussianNB()
[[ 3026  4921]
 [ 3699 27876]]
-----------xxxxxxxxxxxx-----------------
```

Random Forest

```
-------classification Report--------- RandomForestClassifier(n_estimators=10, random_state=0)
              precision    recall  f1-score   support

           0       0.78      0.51      0.62      7947
           1       0.89      0.96      0.92     31575

    accuracy                           0.87     39522
   macro avg       0.83      0.74      0.77     39522
weighted avg       0.86      0.87      0.86     39522

--------Confusion Matrix-----------
 RandomForestClassifier(n_estimators=10, random_state=0)
[[ 4050  3897]
 [ 1150 30425]]
-----------xxxxxxxxxxxx-----------------
```

Based on the results, we can be concluded that this model can predict loan defaulters with an accuracy of 87%. Companies can employ similar models and potentially avoid giving loans to applicants who are highly likely to default. Doing so will reduce risk and financial loss for lending companies. As with all predictive models, data should be monitored and re-evaluated on a regular basis.

For this Dataset the random forest classification algorithm is working as good as Neural networks with 87% accuracy.

REFERENCES

1. 40% of the borrowers may default on their student loans by Zack Friedman, Posted on Nov 5, 2018, https://www.forbes.com/sites/zackfriedman/2018/11/05/student-loans-default strategy/#e2d5c6f17c69
2. A look on the shocking student loan debt, Posted on Feb 4,2019, https://studentloanhero.com/student-loan-debt-statistics/ https://www.cometfi.com/student-loan-debt-statistics
3. loan Statistics, https://www.finder.com/car-loan-statistics
4. More than 1 million people default on their student loan every year on Aug 13, 2018, https://www.cnbc.com/2018/08/13/twenty-two-percent-of-student-loan-borrowers-fall-into default.htm