ANN_LoanLending_Prediction

January 23, 2021

1 ANN Implementation Tensorflow

LendingClub is a US peer-to-peer lending company, headquartered in San Francisco, California.[3] It was the first peer-to-peer lender to register its offerings as securities with the Securities and Exchange Commission (SEC), and to offer loan trading on a secondary market. LendingClub is the world's largest peer-to-peer lending platform.

1.0.1 Our Goal

Given historical data on loans given out with information on whether or not the borrower defaulted (charge-off), can we build a model thatcan predict wether or nor a borrower will pay back their loan? This way in the future when we get a new potential customer we can assess whether or not they are likely to pay back the loan. Keep in mind classification metrics when evaluating the performance of your model!

The "loan_status" column contains our label.

1.0.2 Data Overview

1.1 —-

There are many LendingClub data sets on Kaggle. Here is the information on this particular data set:

LoanStatNew

Description

0

loan_amnt

The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.

1

term

The number of payments on the loan. Values are in months and can be either 36 or 60.

2

int_rate

Interest Rate on the loan

3

installment

The monthly payment owed by the borrower if the loan originates.

4

grade

LC assigned loan grade

5

sub_grade

LC assigned loan subgrade

6

emp_title

The job title supplied by the Borrower when applying for the loan.*

7

emp_length

Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.

8

home_ownership

The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER

9

annual_inc

The self-reported annual income provided by the borrower during registration.

10

verification_status

Indicates if income was verified by LC, not verified, or if the income source was verified

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 loan request.

14

title

The loan title provided by the borrower

15

zip_code

The first 3 numbers of the zip code provided by the borrower in the loan application.

16

addr_state

The state provided by the borrower in the loan application

17

dti

A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.

18

earliest_cr_line

The month the borrower's earliest reported credit line was opened

19

open_acc

The number of open credit lines in the borrower's credit file.

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 of credit the borrower is using relative to all available revolving credit.

23

total_acc

The total number of credit lines currently in the borrower's credit file

24

initial_list_status

The initial listing status of the loan. Possible values are – W, F

25

application_type

Indicates whether the loan is an individual application or a joint application with two co-borrowers

26

mort_acc

Number of mortgage accounts.

27

pub_rec_bankruptcies

Number of public record bankruptcies

1.2 —

1.3 Exploring info csv to get an idea of the information each column contains

[1]:
```python
import pandas as pd
```

[2]:
```python
data_info = pd.read_csv('../DATA/lending_club_info.csv',index_col='LoanStatNew')
```

[3]:
```python
print(data_info.loc['revol_util']['Description'])
```

```
Revolving line utilization rate, or the amount of credit the borrower is using
relative to all available revolving credit.
```

[4]:
```python
def feat_info(col_name):
    print(data_info.loc[col_name]['Description'])
```

[5]:
```python
feat_info('mort_acc')
```

```
Number of mortgage accounts.
```

1.4 Loading the data and other imports

[6]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# might be needed depending on your version of Jupyter
%matplotlib inline
```

[7]:
```python
df = pd.read_csv('../DATA/lending_club_loan_two.csv')
```

[8]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
loan_amnt               396030 non-null float64
term                    396030 non-null object
int_rate                396030 non-null float64
installment             396030 non-null float64
grade                   396030 non-null object
sub_grade               396030 non-null object
emp_title               373103 non-null object
emp_length              377729 non-null object
home_ownership          396030 non-null object
annual_inc              396030 non-null float64
verification_status     396030 non-null object
issue_d                 396030 non-null object
loan_status             396030 non-null object
purpose                 396030 non-null object
title                   394275 non-null object
dti                     396030 non-null float64
earliest_cr_line        396030 non-null object
open_acc                396030 non-null float64
pub_rec                 396030 non-null float64
revol_bal               396030 non-null float64
revol_util              395754 non-null float64
total_acc               396030 non-null float64
initial_list_status     396030 non-null object
application_type        396030 non-null object
mort_acc                358235 non-null float64
pub_rec_bankruptcies    395495 non-null float64
address                 396030 non-null object
dtypes: float64(12), object(15)
memory usage: 81.6+ MB
```

2 Exploratory Data Analysis

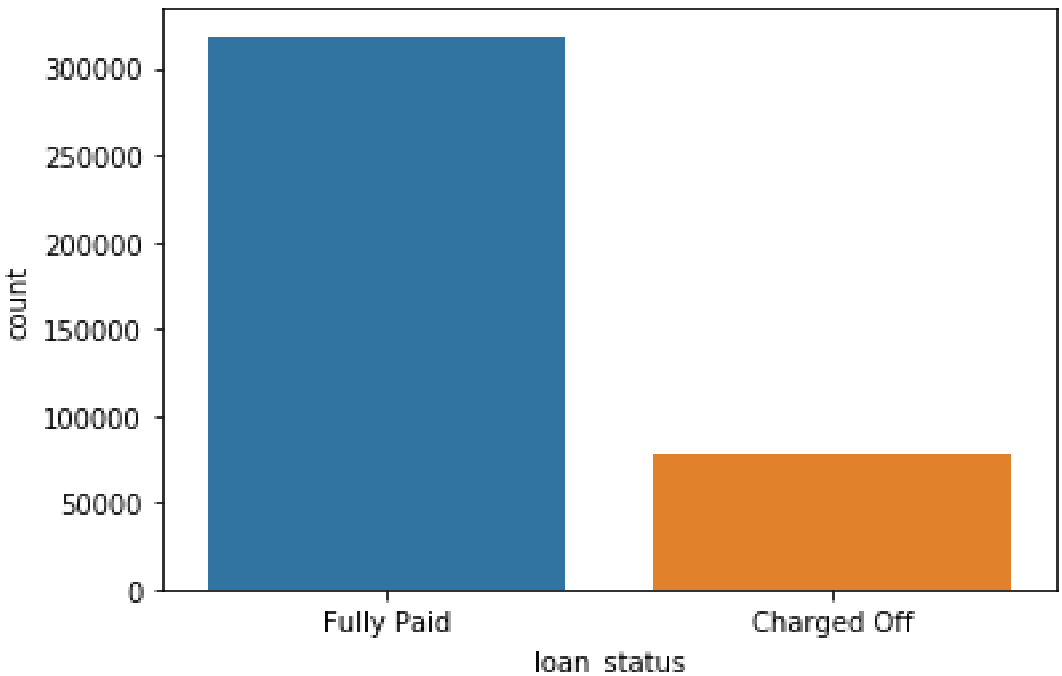
OVERALL GOAL: Get an understanding for which variables are important, view summary statistics, and visualize the data

---

TASK: Since we will be attempting to predict loan_status, creating a countplot.

[9]:
```
sns.countplot('loan_status',data=df)
```

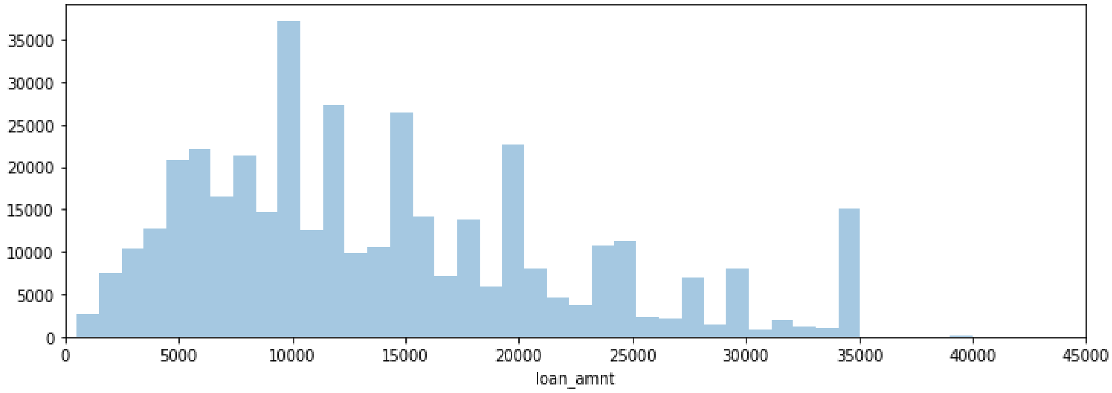
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1b94d32bb48>



Creating a histogram of the loan_amnt column.

[10]:
```
plt.figure(figsize=(12,4))
sns.distplot(df['loan_amnt'],kde=False,bins=40)
plt.xlim(0,45000)
```

[10]: (0, 45000)



Exploring correlation between the continuous feature variables.

[11]:
```
df.corr()
```

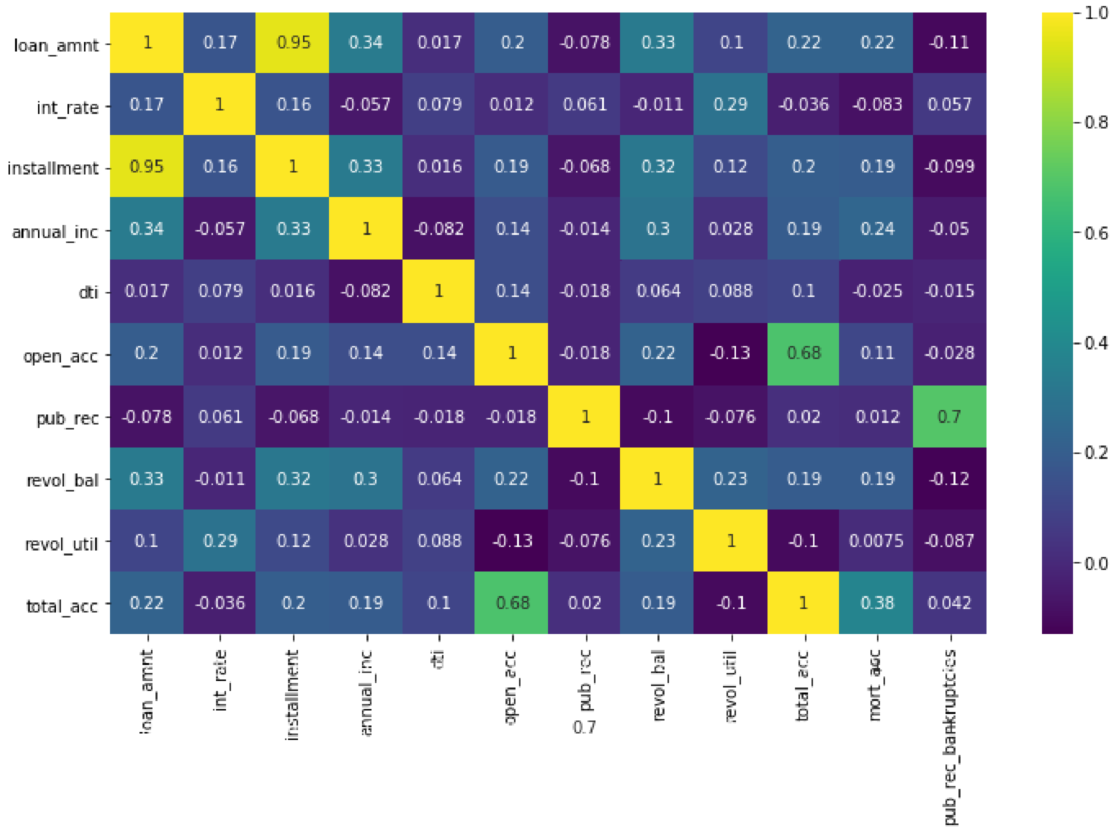
[11]:
```
                      loan_amnt  int_rate  installment  annual_inc       dti  \
loan_amnt              1.000000  0.168921     0.953929    0.336887  0.016636
int_rate               0.168921  1.000000     0.162758   -0.056771  0.079038
installment            0.953929  0.162758     1.000000    0.330381  0.015786
annual_inc             0.336887 -0.056771     0.330381    1.000000 -0.081685
dti                    0.016636  0.079038     0.015786   -0.081685  1.000000
open_acc               0.198556  0.011649     0.188973    0.136150  0.136181
pub_rec               -0.077779  0.060986    -0.067892   -0.013720 -0.017639
revol_bal              0.328320 -0.011280     0.316455    0.299773  0.063571
revol_util             0.099911  0.293659     0.123915    0.027871  0.088375
total_acc              0.223886 -0.036404     0.202430    0.193023  0.102128
mort_acc               0.222315 -0.082583     0.193694    0.236320 -0.025439
pub_rec_bankruptcies  -0.106539  0.057450    -0.098628   -0.050162 -0.014558

                      open_acc   pub_rec  revol_bal  revol_util  total_acc  \
loan_amnt             0.198556 -0.077779   0.328320    0.099911   0.223886
int_rate              0.011649  0.060986  -0.011280    0.293659  -0.036404
installment           0.188973 -0.067892   0.316455    0.123915   0.202430
annual_inc            0.136150 -0.013720   0.299773    0.027871   0.193023
dti                   0.136181 -0.017639   0.063571    0.088375   0.102128
open_acc              1.000000 -0.018392   0.221192   -0.131420   0.680728
pub_rec              -0.018392  1.000000  -0.101664   -0.075910   0.019723
revol_bal             0.221192 -0.101664   1.000000    0.226346   0.191616
revol_util           -0.131420 -0.075910   0.226346    1.000000  -0.104273
total_acc             0.680728  0.019723   0.191616   -0.104273   1.000000
mort_acc              0.109205  0.011552   0.194925    0.007514   0.381072
pub_rec_bankruptcies -0.027732  0.699408  -0.124532   -0.086751   0.042035

                      mort_acc  pub_rec_bankruptcies
```

```
loan_amnt              0.222315          -0.106539
int_rate              -0.082583           0.057450
installment            0.193694          -0.098628
annual_inc             0.236320          -0.050162
dti                   -0.025439          -0.014558
open_acc               0.109205          -0.027732
pub_rec                0.011552           0.699408
revol_bal              0.194925          -0.124532
revol_util             0.007514          -0.086751
total_acc              0.381072           0.042035
mort_acc               1.000000           0.027239
pub_rec_bankruptcies   0.027239           1.000000
```

Visualizing correlation using a heatmap.

[12]:
```python
plt.figure(figsize=(12,7))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
plt.ylim(10,0)
```

[12]: (10, 0)



From above heatmap we noticed almost perfect correlation with the "installment" feature. Lets explore this feature further and find out if the information is revelant enough.

[13]:
```
print(feat_info('installment'))
```

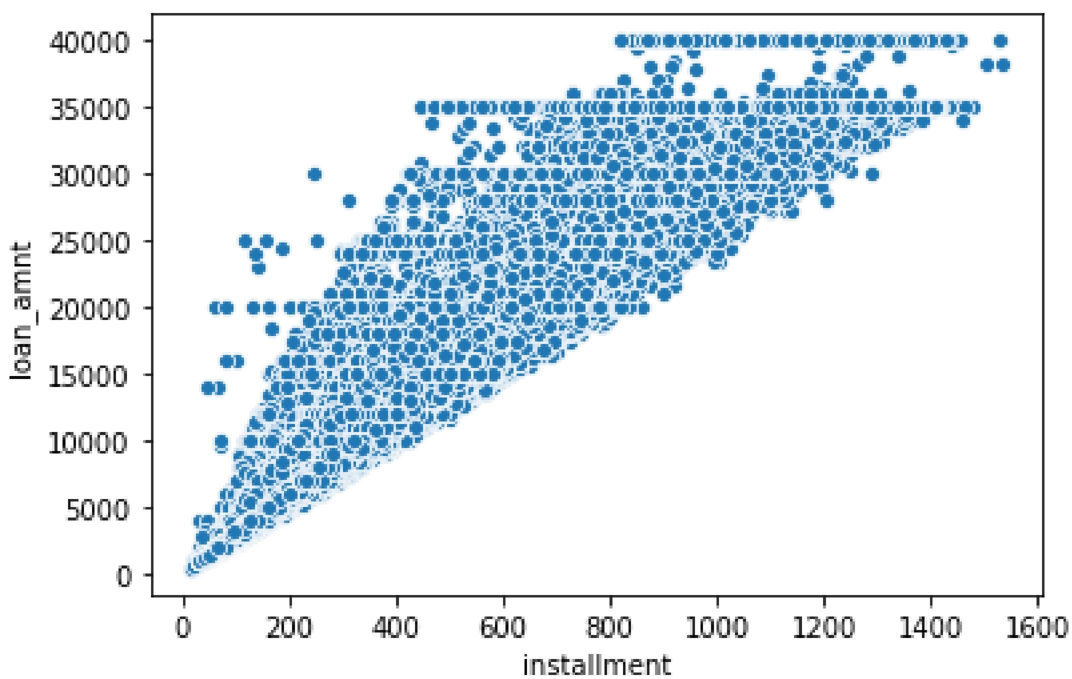
```
The monthly payment owed by the borrower if the loan originates.
None
```

[14]:
```
print(feat_info('loan_amnt'))
```

```
The listed amount of the loan applied for by the borrower. If at some point in
time, the credit department reduces the loan amount, then it will be reflected
in this value.
None
```

[15]:
```
sns.scatterplot(df['installment'],df['loan_amnt'])
```

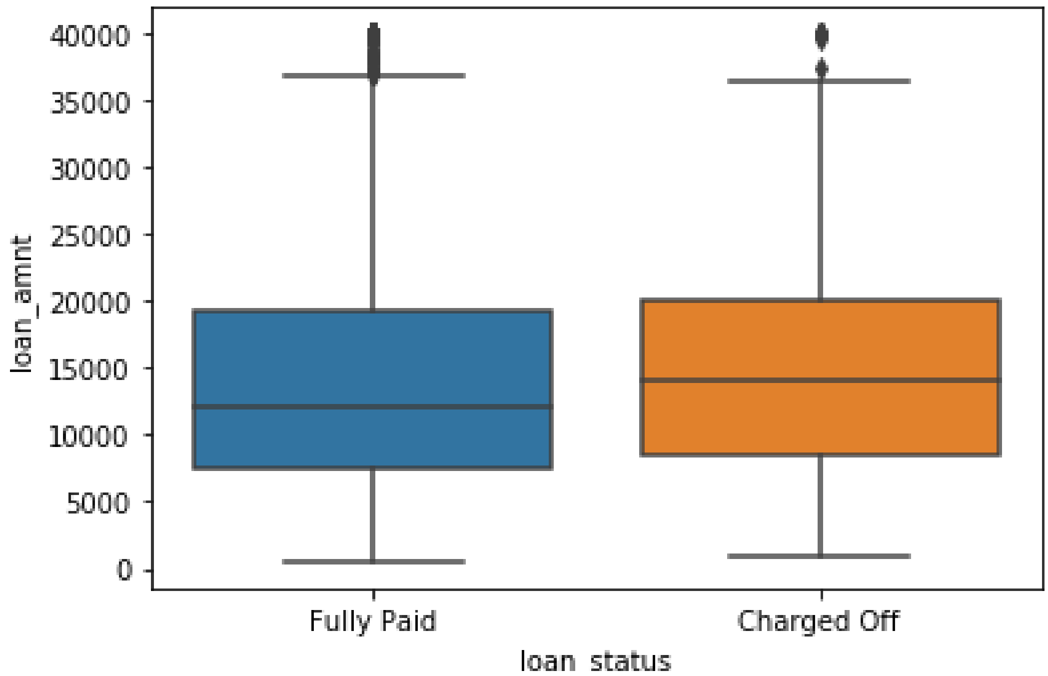
[15]: <matplotlib.axes._subplots.AxesSubplot at 0x1b9607d25c8>



Creating a boxplot showing the relationship between the loan_status and the Loan Amount.

[17]:
```
sns.boxplot(df['loan_status'],df['loan_amnt'])
```

[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1b961089148>



[18]:
```
df.groupby('loan_status')['loan_amnt'].describe()
```

[18]:
```
                count          mean          std     min     25%      50%  \
loan_status
Charged Off   77673.0  15126.300967  8505.090557  1000.0  8525.0  14000.0
Fully Paid   318357.0  13866.878771  8302.319699   500.0  7500.0  12000.0

                 75%      max
loan_status
Charged Off  20000.0  40000.0
Fully Paid   19225.0  40000.0
```

Exploring Grade and SubGrade columns that LendingClub attributes to the loans.

[19]:
```
sorted(list(df['grade'].unique()))
```

[19]:
```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

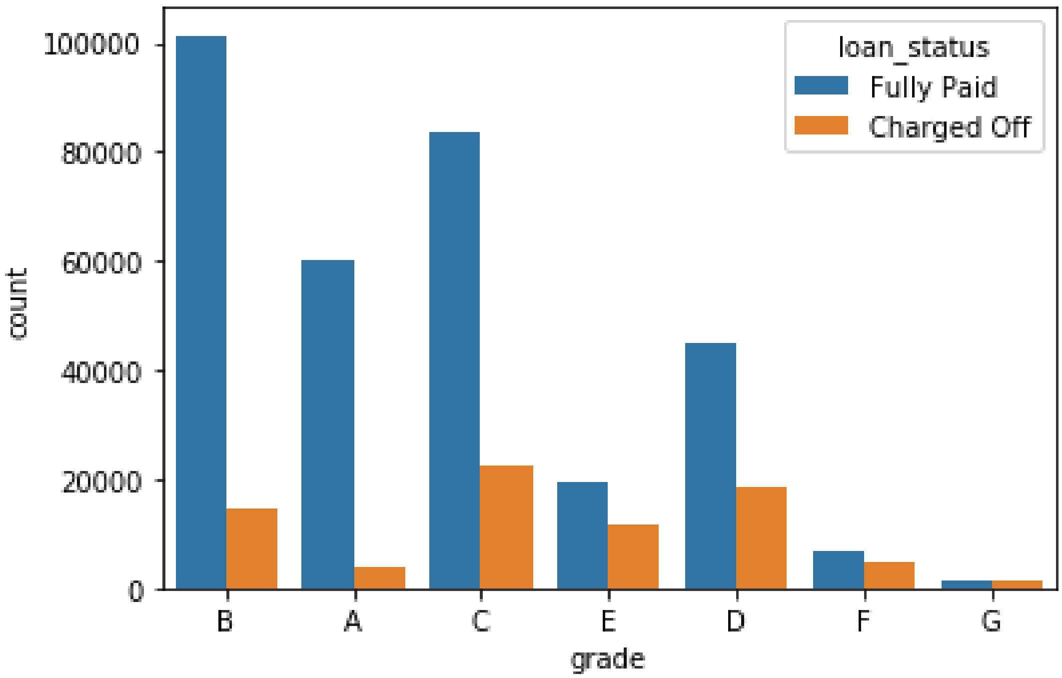
[20]:
```
sorted(list(df['sub_grade'].unique()))
```

[20]:
```
['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']
```

[21]:
```
#Creating a countplot on grade with different loan_status
sns.countplot(df['grade'],hue=df['loan_status'])
```

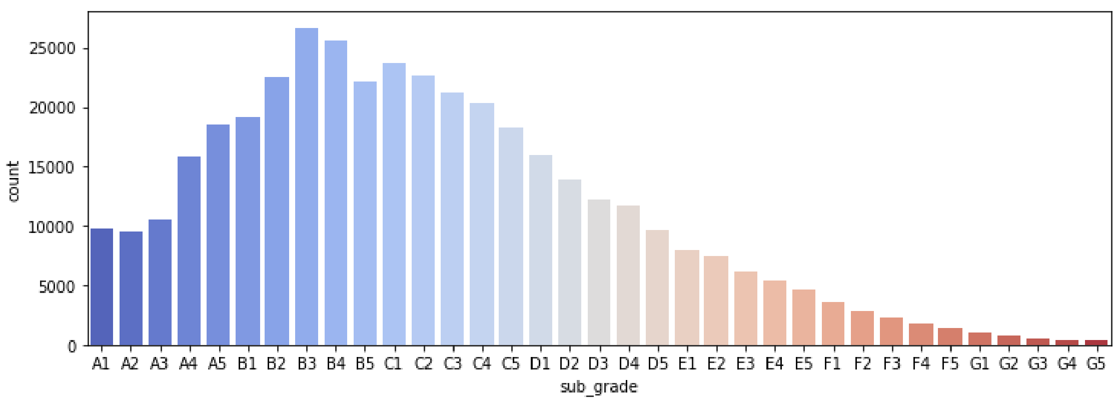
[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b95e4db848>
```



Creating a countplot for subgrades column.

[22]:
```
plt.figure(figsize=(12,4))
sub_grade_order=sorted(df['sub_grade'].unique())
sns.countplot(df['sub_grade'],order=sub_grade_order,palette='coolwarm')
```

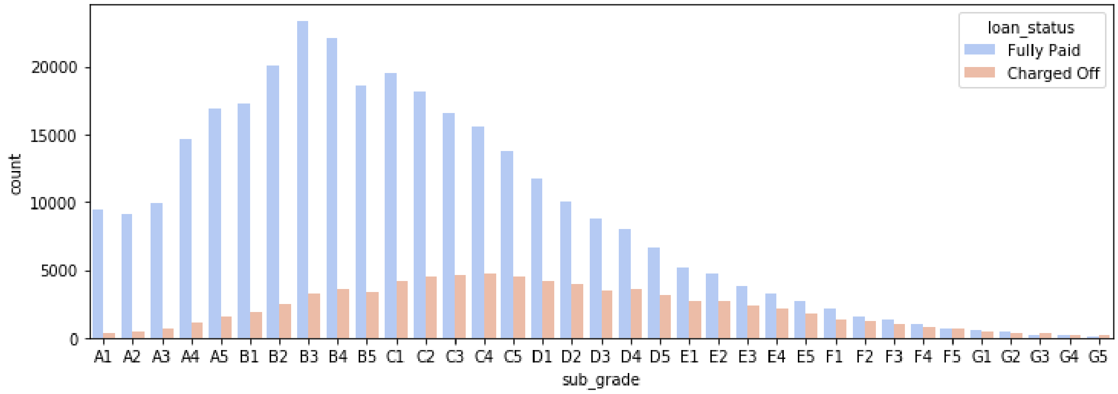
[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1b95e45b1c8>



[23]:
```
plt.figure(figsize=(12,4))
sub_grade_order=sorted(df['sub_grade'].unique())
```

```
sns.
↪countplot(df['sub_grade'],order=sub_grade_order,palette='coolwarm',hue=df['loan_status'])
```

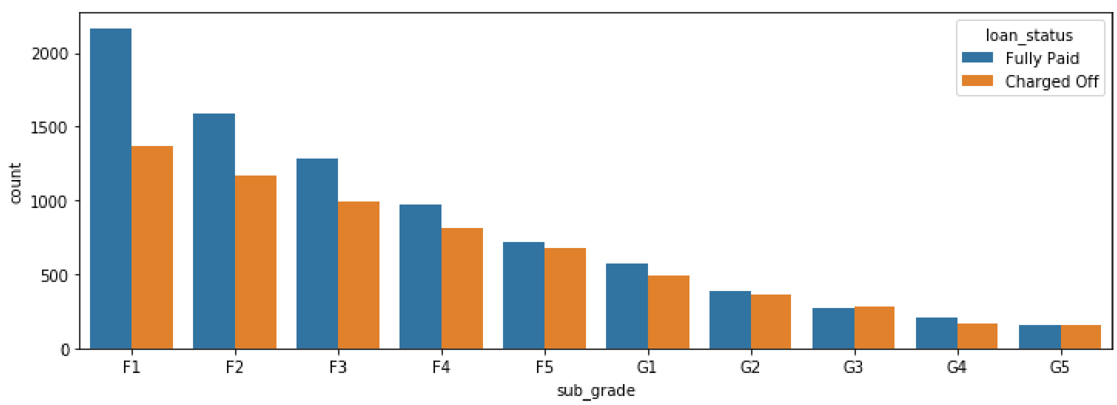
[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1b96107c788>



It looks like F and G subgrades don't get paid back that often. Isloating these and recreating the countplot just for these subgrades.

[24]:
```
# Extracting subgrades containing only F and G
sub_df=df[df.sub_grade.str.contains('F|G')]
plt.figure(figsize=(12,4))
sub_grade_order=sorted(sub_df['sub_grade'].unique())
sns.
↪countplot(sub_df['sub_grade'],order=sub_grade_order,hue=sub_df['loan_status'])
```

[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1b96062fa08>



Creating a new column called 'loan_repaid' which will contain a 1 if the loan status was "Fully Paid" and a 0 if it was "Charged Off".

[25]:
```
df['loan_repaid']=df['loan_status'].map({'Fully Paid':1,'Charged Off':0})
print(df['loan_status'].value_counts())
df['loan_repaid'].value_counts()
```

```
Fully Paid     318357
Charged Off     77673
Name: loan_status, dtype: int64
```

[25]:
```
1    318357
0     77673
Name: loan_repaid, dtype: int64
```

[26]:
```
df['loan_status'].unique()
```

[26]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

[27]:
```
df[['loan_repaid','loan_status']]
```

[27]:

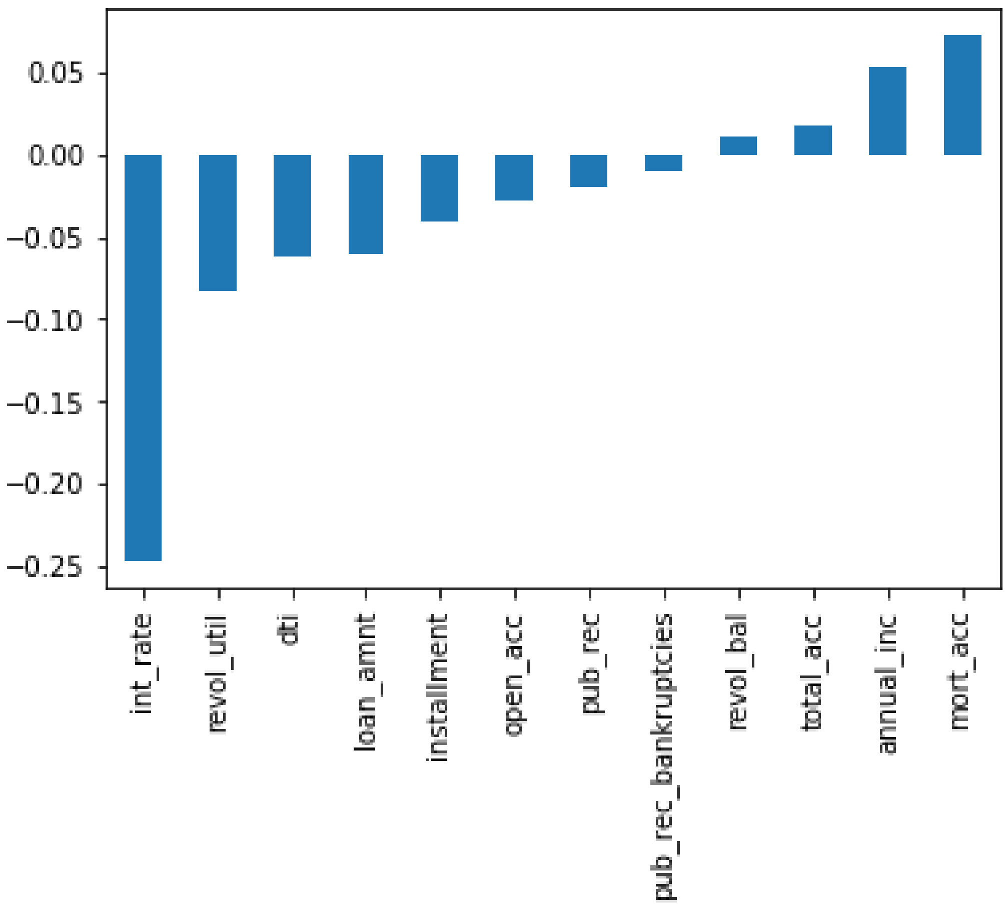
	loan_repaid	loan_status
0	1	Fully Paid
1	1	Fully Paid
2	1	Fully Paid
3	1	Fully Paid
4	0	Charged Off
…	…	…
396025	1	Fully Paid
396026	1	Fully Paid
396027	1	Fully Paid
396028	1	Fully Paid
396029	1	Fully Paid

[396030 rows x 2 columns]

Checking the correlation of loan_repaid feature with other features and plotting using bar plot.

[28]:
```
df.corr()['loan_repaid'].sort_values().drop('loan_repaid').plot(kind='bar')
```

[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b96070cb88>
```



3 Data PreProcessing

Remove or fill any missing data. Remove unnecessary or repetitive features. Convert categorical string features to dummy variables.

4 Missing Data

[29]:
```
# Length of dataframe
df.shape[0]
```

[29]: 396030

[ ]:
```

```

[30]:
```
# No of NaN values per feature
df.isna().sum()
```

[30]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
emp_title               22927
emp_length              18301
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
loan_repaid                 0
dtype: int64
```

Percentage of NaN per feature in total DataFrame

[31]:
```
df.isna().sum()/df.shape[0] * 100
```

[31]:
```
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
loan_repaid             0.000000
dtype: float64
```

[32]:
```python
feat_info('emp_title')
```

```
The job title supplied by the Borrower when applying for the loan.*
```

[33]:
```python
feat_info('emp_length')
```

```
Employment length in years. Possible values are between 0 and 10 where 0 means
less than one year and 10 means ten or more years.
```

Getting count of unique employment job titles.

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

[35]:
```
173105
```

[36]:
```python
df['emp_title'].value_counts()
```

[36]:
```
Teacher                        4389
Manager                        4250
Registered Nurse               1856
RN                             1846
Supervisor                     1830
                               …
la clinica delaraza               1
Associate District Manager        1
New England Wireless              1
ssg e6                            1
PaleoSun, Inc.                    1
Name: emp_title, Length: 173105, dtype: int64
```

As there are too many unique job titles to try to convert this to a dummy variable

feature. Removing emp_title column.

[37]:
```
df=df.drop(columns=['emp_title'])
```

Creating a count plot for emp_length feature.

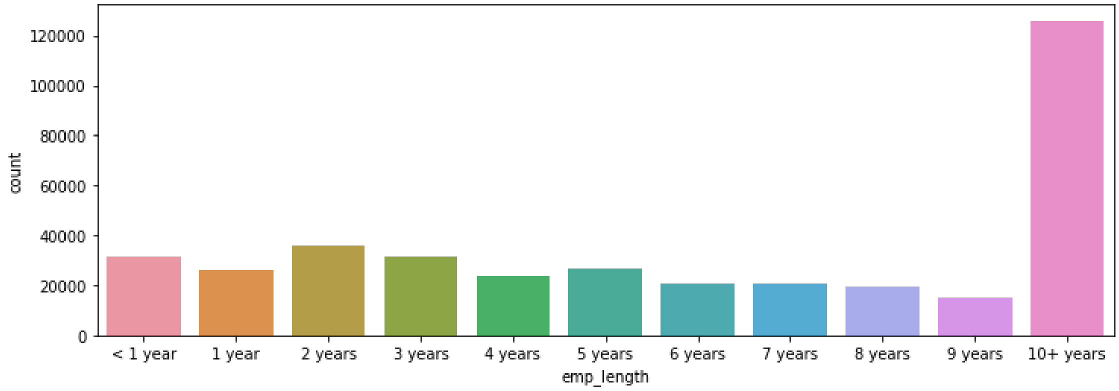
[38]:
```
sorted(df['emp_length'].dropna().unique())
```

[38]:
```
['1 year',
 '10+ years',
 '2 years',
 '3 years',
 '4 years',
 '5 years',
 '6 years',
 '7 years',
 '8 years',
 '9 years',
 '< 1 year']
```

[39]:
```
emp_length_order = [ '< 1 year',
                      '1 year',
                     '2 years',
                     '3 years',
                     '4 years',
                     '5 years',
                     '6 years',
                     '7 years',
                     '8 years',
                     '9 years',
                     '10+ years']
```

[40]:
```
plt.figure(figsize=(12,4))

sns.countplot(x='emp_length',data=df,order=emp_length_order)
```

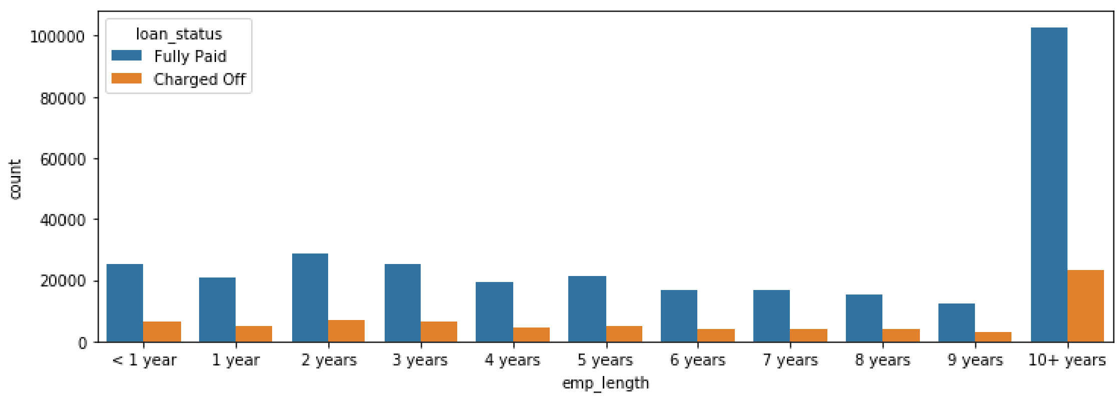
[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b961c0ef88>
```



Plotting countplot with a hue separating Fully Paid vs Charged Off

[41]:
```
plt.figure(figsize=(12,4))
sns.countplot(x='emp_length',data=df,order=emp_length_order,hue='loan_status')
```

[41]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b961c10548>
```



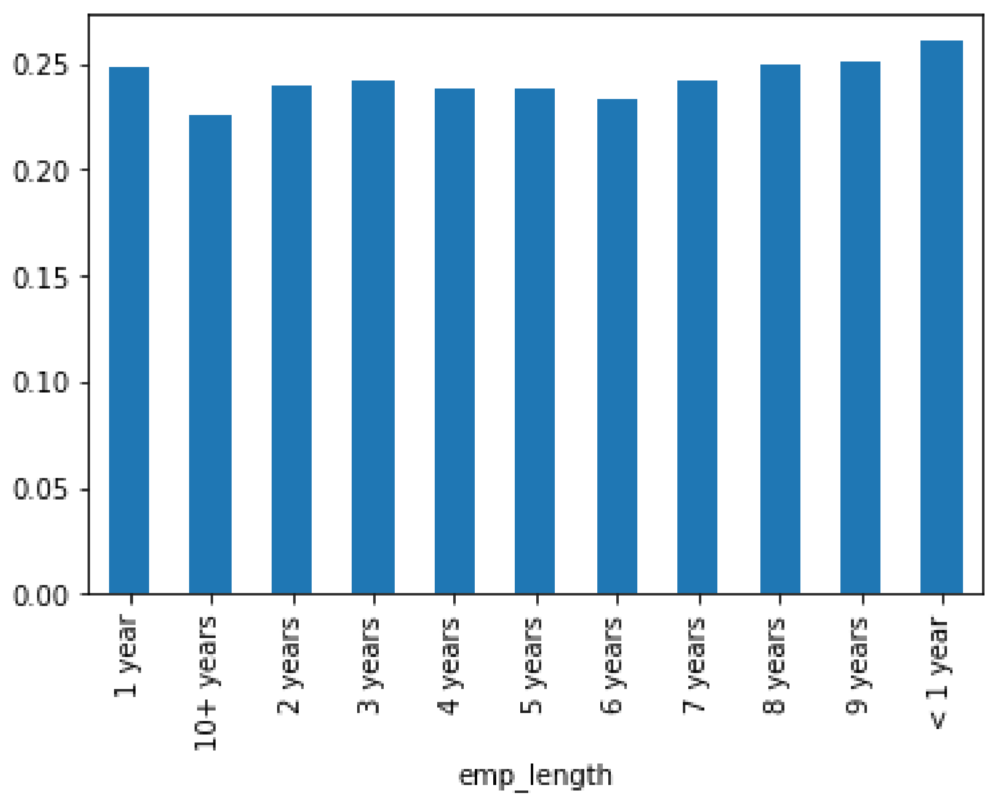
This still doesn't really inform us if there is a strong relationship between employment length and being charged off, what we want is the percentage of charge offs per category. Essentially informing us what percent of people per employment category didn't pay back their loan.

[42]:
```
emp_co = df[df['loan_status']=="Charged Off"].groupby("emp_length").
 ↪count()['loan_status']
emp_fp = df[df['loan_status']=="Fully Paid"].groupby("emp_length").
 ↪count()['loan_status']
emp_len = emp_co/emp_fp
emp_len
```

[42]:
```
emp_length
1 year       0.248649
10+ years    0.225770
2 years      0.239560
3 years      0.242593
4 years      0.238213
5 years      0.237911
6 years      0.233341
7 years      0.241887
8 years      0.249625
9 years      0.250735
< 1 year     0.260830
Name: loan_status, dtype: float64
```

[43]:
```python
emp_len.plot(kind='bar')
```

[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b961393508>
```



Charge off rates are extremely similar across all employment lengths. Dropping the emp_length column.

[44]:
```python
df=df.drop(columns=['emp_length'])
```

TASK: Revisit the DataFrame to see what feature columns still have missing data.

[45]:
```python
df.isna().sum()
```

[45]:
```
loan_amnt                   0
term                        0
int_rate                    0
installment                 0
grade                       0
sub_grade                   0
home_ownership              0
annual_inc                  0
verification_status         0
issue_d                     0
loan_status                 0
purpose                     0
title                    1755
dti                         0
earliest_cr_line            0
open_acc                    0
pub_rec                     0
revol_bal                   0
revol_util                276
total_acc                   0
initial_list_status         0
application_type            0
mort_acc                37795
pub_rec_bankruptcies      535
address                     0
loan_repaid                 0
dtype: int64
```

TASK: Review the title column vs the purpose column. Is this repeated information?

[46]:
```python
df['purpose'].head(10)
```

[46]:
```
0              vacation
1    debt_consolidation
2           credit_card
3           credit_card
4           credit_card
5    debt_consolidation
6      home_improvement
7           credit_card
8    debt_consolidation
9    debt_consolidation
Name: purpose, dtype: object
```

[47]:
```python
df['title'].head(10)
```

[47]:
```
0                   Vacation
1         Debt consolidation
2    Credit card refinancing
3    Credit card refinancing
4      Credit Card Refinance
5         Debt consolidation
6           Home improvement
7       No More Credit Cards
8         Debt consolidation
9         Debt Consolidation
Name: title, dtype: object
```

The title column is simply a string subcategory/description of the purpose column.Dropping the title column.

[48]:
```python
df=df.drop(columns=['title'])
```

Exploring mort_acc feature

[49]:
```python
feat_info('mort_acc')
```

```
Number of mortgage accounts.
```

[50]:
```python
df['mort_acc'].value_counts()
```

[50]:
```
0.0     139777
1.0      60416
2.0      49948
3.0      38049
4.0      27887
5.0      18194
6.0      11069
7.0       6052
8.0       3121
9.0       1656
10.0       865
11.0       479
12.0       264
13.0       146
14.0       107
15.0        61
16.0        37
17.0        22
18.0        18
```

```
19.0       15
20.0       13
24.0       10
22.0        7
21.0        4
25.0        4
27.0        3
23.0        2
32.0        2
26.0        2
31.0        2
30.0        1
28.0        1
34.0        1
Name: mort_acc, dtype: int64
```

[51]:
```python
print("Correlation with the mort_acc column")
df.corr()['mort_acc'].sort_values()
```

```
Correlation with the mort_acc column
```

[51]:
```
int_rate               -0.082583
dti                    -0.025439
revol_util              0.007514
pub_rec                 0.011552
pub_rec_bankruptcies    0.027239
loan_repaid             0.073111
open_acc                0.109205
installment             0.193694
revol_bal               0.194925
loan_amnt               0.222315
annual_inc              0.236320
total_acc               0.381072
mort_acc                1.000000
Name: mort_acc, dtype: float64
```

[52]:
```python
print("Mean of mort_acc column per total_acc")
df.groupby('total_acc').mean()['mort_acc']
```

```
Mean of mort_acc column per total_acc
```

[52]:
```
total_acc
2.0      0.000000
3.0      0.052023
4.0      0.066743
5.0      0.103289
6.0      0.151293
```

```
         …
124.0    1.000000
129.0    1.000000
135.0    3.000000
150.0    2.000000
151.0    0.000000
Name: mort_acc, Length: 118, dtype: float64
```

[53]:
```
total_acc_avg = df.groupby('total_acc').mean()['mort_acc']
```

[54]:
```
def fill_mort_acc(total_acc,mort_acc):
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc]
    else:
        return mort_acc
```

[55]:
```
df['mort_acc'] = df.apply(lambda x: fill_mort_acc(x['total_acc'],␣
 ↪x['mort_acc']), axis=1)
```

[56]:
```
df.isnull().sum()
```

[56]:
```
loan_amnt                 0
term                      0
int_rate                  0
installment               0
grade                     0
sub_grade                 0
home_ownership            0
annual_inc                0
verification_status       0
issue_d                   0
loan_status               0
purpose                   0
dti                       0
earliest_cr_line          0
open_acc                  0
pub_rec                   0
revol_bal                 0
revol_util              276
total_acc                 0
initial_list_status       0
application_type          0
mort_acc                  0
pub_rec_bankruptcies    535
address                   0
loan_repaid               0
dtype: int64
```

revol__util and the pub__rec__bankruptcies have missing data points, but they account for less than 0.5% of the total data.

[57]:
```
# Dropping remain NaN values
df=df.dropna()
```

[58]:
```
df.isna().sum()
```

[58]:
```
loan_amnt               0
term                    0
int_rate                0
installment             0
grade                   0
sub_grade               0
home_ownership          0
annual_inc              0
verification_status     0
issue_d                 0
loan_status             0
purpose                 0
dti                     0
earliest_cr_line        0
open_acc                0
pub_rec                 0
revol_bal               0
revol_util              0
total_acc               0
initial_list_status     0
application_type        0
mort_acc                0
pub_rec_bankruptcies    0
address                 0
loan_repaid             0
dtype: int64
```

4.1 Categorical Variables and Dummy Variables

We're done working with the missing data! Now we just need to deal with the string values due to the categorical columns.

[59]:
```
# Getting a list of string/object columns in dataframe
df.select_dtypes(include=['object']).columns
```

[59]:
```
Index(['term', 'grade', 'sub_grade', 'home_ownership', 'verification_status',
       'issue_d', 'loan_status', 'purpose', 'earliest_cr_line',
       'initial_list_status', 'application_type', 'address'],
      dtype='object')
```

[60]:
```python
# term feature
df['term'].value_counts()
```

[60]:
```
 36 months    301247
 60 months     93972
Name: term, dtype: int64
```

[61]:
```python
df['term'] = df['term'].apply(lambda term: int(term[:3]))
```

4.1.1 grade feature

[62]:
```python
df = df.drop('grade',axis=1)
```

sub_grade feature

[63]:
```python
subgrade_dummies = pd.get_dummies(df['sub_grade'],drop_first=True)
df = pd.concat([df.drop('sub_grade',axis=1),subgrade_dummies],axis=1)
df.columns
```

[63]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'home_ownership',
       'annual_inc', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'initial_list_status',
       'application_type', 'mort_acc', 'pub_rec_bankruptcies', 'address',
       'loan_repaid', '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'],
      dtype='object')
```

[64]:
```python
df.select_dtypes(['object']).columns
```

[64]:
```
Index(['home_ownership', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'earliest_cr_line', 'initial_list_status',
       'application_type', 'address'],
      dtype='object')
```

4.1.2 verification_status, application_type,initial_list_status,purpose

[65]:
```python
dummies = pd.get_dummies(df[['verification_status',␣
↪'application_type','initial_list_status','purpose' ]],drop_first=True)
df = df.drop(['verification_status',␣
↪'application_type','initial_list_status','purpose'],axis=1)
df = pd.concat([df,dummies],axis=1)
```

4.1.3 home_ownership feature

[66]:
```
df['home_ownership'].value_counts()
```

[66]:
```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

[67]:
```
df['home_ownership']=df['home_ownership'].replace(['NONE', 'ANY'], 'OTHER')

dummies = pd.get_dummies(df['home_ownership'],drop_first=True)
df = df.drop('home_ownership',axis=1)
df = pd.concat([df,dummies],axis=1)
```

4.1.4 address feature

[68]:
```
df['zip_code'] = df['address'].apply(lambda address:address[-5:])
df=df.drop(columns=['address'])
```

[69]:
```
dummies = pd.get_dummies(df['zip_code'],drop_first=True)
df = df.drop('zip_code',axis=1)
df = pd.concat([df,dummies],axis=1)
```

[70]:
```
df.columns
```

[70]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'annual_inc', 'issue_d',
       'loan_status', 'dti', 'earliest_cr_line', 'open_acc', 'pub_rec',
       'revol_bal', 'revol_util', 'total_acc', 'mort_acc',
       'pub_rec_bankruptcies', 'loan_repaid', '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', 'verification_status_Source Verified',
       'verification_status_Verified', 'application_type_INDIVIDUAL',
       'application_type_JOINT', 'initial_list_status_w',
       'purpose_credit_card', 'purpose_debt_consolidation',
       'purpose_educational', 'purpose_home_improvement', 'purpose_house',
       'purpose_major_purchase', 'purpose_medical', 'purpose_moving',
       'purpose_other', 'purpose_renewable_energy', 'purpose_small_business',
       'purpose_vacation', 'purpose_wedding', 'OTHER', 'OWN', 'RENT', '05113',
       '11650', '22690', '29597', '30723', '48052', '70466', '86630', '93700'],
      dtype='object')
```

4.1.5 issue_d feature dropping

[71]:
```
df=df.drop(columns='issue_d')
```

4.1.6 earliest_cr_line

This appears to be a historical time stamp feature. Extracting the year from this feature

[72]:
```
df['earliest_cr_year'] = df['earliest_cr_line'].apply(lambda date:int(date[-4:
↪]))
df = df.drop('earliest_cr_line',axis=1)
```

[73]:
```
df.select_dtypes(['object']).columns
```

[73]:
```
Index(['loan_status'], dtype='object')
```

4.2 Train Test Split

[74]:
```
from sklearn.model_selection import train_test_split
```

[75]:
```
df = df.drop('loan_status',axis=1)
```

[76]:
```
len(df.columns)
```

[76]:
```
79
```

Setting X and y variables to the .values of the features and label.

[77]:
```
X = df.drop('loan_repaid',axis=1).values
y = df['loan_repaid'].values
```

TASK: Perform a train/test split with test_size=0.2 and a random_state of 101.

[78]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,␣
↪random_state=101)
```

4.3 Normalizing the Data

[80]:
```
from sklearn.preprocessing import MinMaxScaler
```

[81]:
```
#Using MinMaxScaler for the same
scaler = MinMaxScaler()
```

[82]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

5 Creating the Model using Tensorflow and Keras

[83]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation,Dropout
from tensorflow.keras.constraints import max_norm
```

[84]:
```
#Initializing a seqntial model
model = Sequential()

#Input layers with no of neurons as the no of columns and activtion fn as relu
model.add(Dense(78,activation='relu'))
model.add(Dropout(0.3))

#Hidden Layer1
model.add(Dense(39,activation='relu'))
#Adding dropouts inorder to avoid overfitting.
model.add(Dropout(0.3))

#Hidden Layer2
model.add(Dense(19,activation='relu'))
model.add(Dropout(0.3))

#Output Layer, as this is a classification binary problem hence using␣
↪activation as sigmoid.
model.add(Dense(units=1,activation='sigmoid'))

#Compiling model with loss as binary_crossentropy and optimizer as adam
model.compile(loss='binary_crossentropy', optimizer='adam')
```

TASK: Fit the model to the training data for at least 25 epochs. Also add in the validation data for later plotting. Optional: add in a batch_size of 256.

[85]:
```
#Fitting the model in train data as well as checking the loss in test data to␣
↪get an instanct idea of model performance.
model.
↪fit(x=X_train,y=y_train,epochs=25,batch_size=256,validation_data=(X_test,y_test))
```

```
Train on 316175 samples, validate on 79044 samples
Epoch 1/25
316175/316175 [==============================] - 7s 21us/sample - loss: 0.3048 -
val_loss: 0.2651
Epoch 2/25
316175/316175 [==============================] - 6s 18us/sample - loss: 0.2674 -
val_loss: 0.2633
Epoch 3/25
316175/316175 [==============================] - 6s 18us/sample - loss: 0.2642 -
val_loss: 0.2629
```

```
Epoch 4/25
316175/316175 [==============================] - 6s 18us/sample - loss: 0.2630 -
val_loss: 0.2629
Epoch 5/25
316175/316175 [==============================] - 6s 17us/sample - loss: 0.2621 -
val_loss: 0.2625
Epoch 6/25
316175/316175 [==============================] - 7s 22us/sample - loss: 0.2619 -
val_loss: 0.2620
Epoch 7/25
316175/316175 [==============================] - 11s 35us/sample - loss: 0.2612
- val_loss: 0.2629
Epoch 8/25
316175/316175 [==============================] - 9s 28us/sample - loss: 0.2611 -
val_loss: 0.2621
Epoch 9/25
316175/316175 [==============================] - 8s 25us/sample - loss: 0.2608 -
val_loss: 0.2620
Epoch 10/25
316175/316175 [==============================] - 8s 25us/sample - loss: 0.2603 -
val_loss: 0.2619
Epoch 11/25
316175/316175 [==============================] - 8s 25us/sample - loss: 0.2602 -
val_loss: 0.2621
Epoch 12/25
316175/316175 [==============================] - 8s 25us/sample - loss: 0.2600 -
val_loss: 0.2621
Epoch 13/25
316175/316175 [==============================] - 8s 25us/sample - loss: 0.2596 -
val_loss: 0.2618
Epoch 14/25
316175/316175 [==============================] - 11s 34us/sample - loss: 0.2598
- val_loss: 0.2620
Epoch 15/25
316175/316175 [==============================] - 12s 37us/sample - loss: 0.2593
- val_loss: 0.2615
Epoch 16/25
316175/316175 [==============================] - 11s 33us/sample - loss: 0.2592
- val_loss: 0.2618
Epoch 17/25
316175/316175 [==============================] - 13s 42us/sample - loss: 0.2591
- val_loss: 0.2617
Epoch 18/25
316175/316175 [==============================] - 12s 38us/sample - loss: 0.2591
- val_loss: 0.2614
Epoch 19/25
316175/316175 [==============================] - 10s 32us/sample - loss: 0.2589
- val_loss: 0.2611
```

```
Epoch 20/25
316175/316175 [==============================] - 11s 33us/sample - loss: 0.2588
- val_loss: 0.2619
Epoch 21/25
316175/316175 [==============================] - 10s 31us/sample - loss: 0.2587
- val_loss: 0.2614
Epoch 22/25
316175/316175 [==============================] - 9s 27us/sample - loss: 0.2583 -
val_loss: 0.2616
Epoch 23/25
316175/316175 [==============================] - 11s 35us/sample - loss: 0.2581
- val_loss: 0.2612
Epoch 24/25
316175/316175 [==============================] - 11s 34us/sample - loss: 0.2583
- val_loss: 0.2615
Epoch 25/25
316175/316175 [==============================] - 11s 34us/sample - loss: 0.2582
- val_loss: 0.2621
```

[85]:
```
<tensorflow.python.keras.callbacks.History at 0x1b90528ce48>
```

[86]:
```python
#Saving the model
from tensorflow.keras.models import load_model

model.save('loan_predictor_model.h5')
```

[ ]:
```python

```

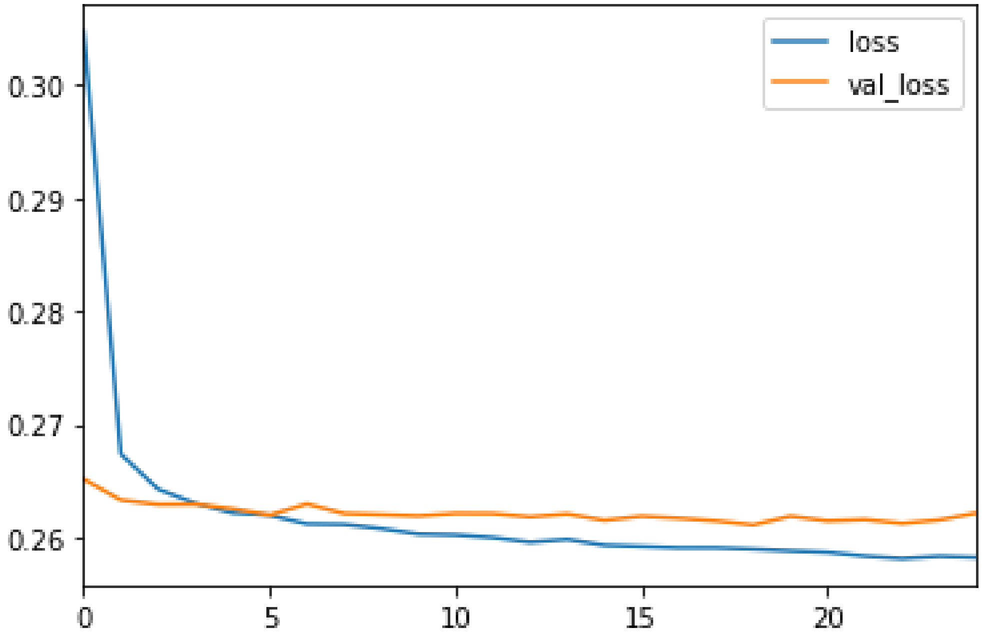
6 Evaluating Model Performance.

Plotting out the validation loss versus the training loss.

[87]:
```python
losses=pd.DataFrame(model.history.history)
```

[88]:
```python
losses.plot()
```

[88]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1b96155c988>
```



[89]:
```
# Creating classification report and confusion matrix
from sklearn.metrics import classification_report,confusion_matrix
```

[90]:
```
predictions = model.predict_classes(X_test)
```

[91]:
```
#Getting an idea of model performance and accuracy using f1-score and precision␣
 ↪and recall values
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      0.43      0.60     15658
           1       0.88      1.00      0.93     63386

    accuracy                           0.89     79044
   macro avg       0.94      0.71      0.77     79044
weighted avg       0.90      0.89      0.87     79044
```

[92]:
```
confusion_matrix(y_test,predictions)
```

[92]:
```
array([[ 6725,  8933],
       [    1, 63385]], dtype=int64)
```

The ration of TPR and TNR is high hence the model is working quite well.

[93]:
```
#Checking prediction for a random value
import random
random.seed(101)
random_ind = random.randint(0,len(df))

new_customer = df.drop('loan_repaid',axis=1).iloc[random_ind]
new_customer
```

[93]:
```
loan_amnt           25000.00
term                   60.00
int_rate               18.24
installment           638.11
annual_inc          61665.00
                      …
48052                   0.00
70466                   0.00
86630                   0.00
93700                   0.00
earliest_cr_year     1996.00
Name: 305323, Length: 78, dtype: float64
```

[94]:
```
model.predict_classes(new_customer.values.reshape(1,78))
```

[94]:
```
array([[1]])
```

[95]:
```
df.iloc[random_ind]['loan_repaid']
```

[95]:
```
1.0
```