Keras Exercise and Deep Learning project on Lending Club Dataset

Before starting: If you find this kernal helpful please upvote the project.

Company Information:

Lending Club is a peer to peer lending company based in , headquartered in San Francisco, California, in which investors provide funds for potential borrowers and investors earn a profit depending on the risk they take (the borrowers credit score). Lending Club provides the "bridge" between investors and borrowers. LendingClub is the world's largest peer-to-peer lending platform. For more basic information about the company please check out: Lending Club Information

Our Goal

Given historical data on loans can we build a model to predict wether or nor a borrower will pay back their loan? For example, in the future when we get a new potential customer we can assess whether or not they are likely to pay back the loan.

The "loan_status" column contains our label.

Outline:

I. Introduction
a) Data Overview (https://www.kaggle.com/hadiyad/keras-exercises-lending-club#data_overview)
b) Starter Code (https://www.kaggle.com/hadiyad/keras-exercises-lending-club#starter_code)
c) Loading data and other imports (https://www.kaggle.com/hadiyad/keras-exercises-lending-club#loading)

II. Exploratory Data Analysis (https://www.kaggle.com/hadiyad/keras-exercises-lending-club#EDA)
a) Exploring correlation between the continuous feature variables b) Changing the catagorical label of loan_status to numerical one

III. Data Preprocessing (https://www.kaggle.com/hadiyad/keras-exercises-lending-club#data_prepro)
a) Missing data
b) Categorical Variables and Dummy Variables
c) Normalizing the Data

IV. Creating the Model

V. **Evaluating Model Performance.**

References:

1) Python for Data Science and Machine Learning Bootcamp by Jose Portilla
2) Lending Club || Risk Analysis and Metrics by Janio Martinez

Introduction

Data Overview

There are many LendingClub data sets on Kaggle. Here is the information on this particular data set. It is always a good practice to investigate data and trys to guess what is going to happen through it. So, read the data columns and descriptions and imagine how the system works, in this case how lending clubs company works , how it colects data and try to guess which data you think is important. At the end, you will see how you are far of your gut instinct! let's look at and think:

	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.

	LoanStatNew	Description
27	pub_rec_bankruptcies	Number of public record bankruptcies

In [1]:
```
import pandas as pd
data_info = pd.read_csv('../input/lendingclub-data-sets/lending_club_inf
o.csv',index_col='LoanStatNew')
print(data_info.loc['revol_util']['Description'])
```

In [2]:
```
def feat_info(col_name):
    print(data_info.loc[col_name]['Description'])
feat_info('mort_acc')
```

Loading the data and other imports

In [3]:

```
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

#reading data
df = pd.read_csv('../input/lendingclub-data-sets/lending_club_loan_two.csv')
df.info()
```

Exploratory Data Analysis

Why we want to explore data? Answer: To better understand which variables are important, view summary statistics, and visualize the data

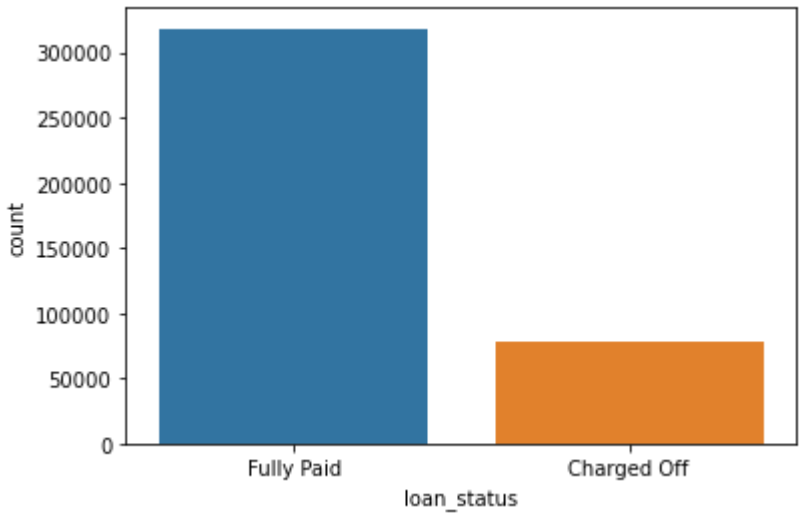
let see the target value? Loan_status.

It is the ultimate goal of the lending club! If the customers fully paid their loan or not! Which information are really important and detrimine if a person would returns the loan or not.

First let's see the Loan_status: It is especially good for classification task to see how our target vlaues are balanced.

In [4]:
```
sns.countplot(x='loan_status', data=df)
```

Out[4]:



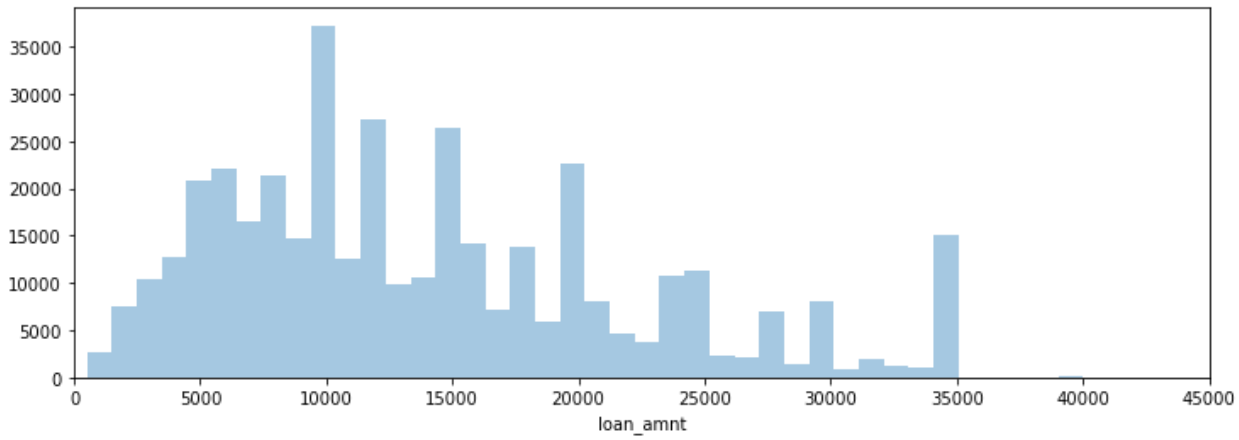
We would say this is unbalanced problem. From above plot we see that most of loans were paid off by 80%. Further comments comes to my mind: It is a little unbalnced dataset.

This is really common thing in calssification problems like fruad or spam. There is a lot of less instances of fraud or spam then are of ligimtate actions , such as ligimitate email, ligimatate credit card purchase and ligimitate paid off.

It means that we can expect very well accuracy while evaluating our ML model. But,in fact the precision and recall are going to be ture metrics.

In [5]:
```
plt.figure(figsize=(12,4))
sns.distplot(df['loan_amnt'], kde=False, bins=40)
plt.xlim(0,45000)
```

Out[5]:



In the plot you can see some spikes is happining between bins. It means we have some loans as a standard. In all, you can see the amount of the loan applied for by the borrower. means seems to around 10000 $ and you see the distribution.

Exploring correlation between the continuous feature variables

We can explore the correlation between numeric variables using .corr() method:

In [6]:
```
df.corr()
```

Out[6]:

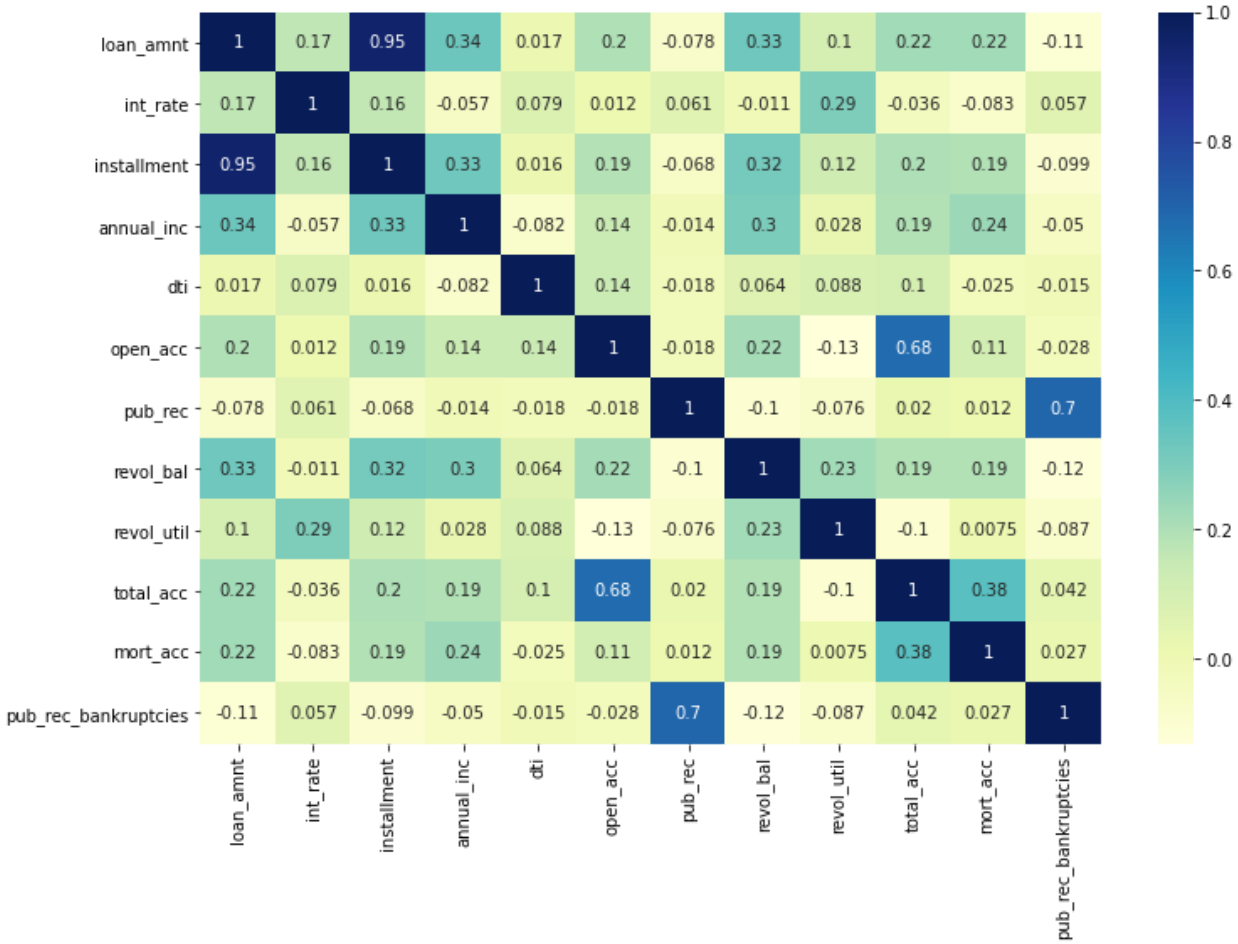
	loan_amnt	int_rate	installment	annual_inc	dti	open_acc
loan_amnt	1.000000	0.168921	0.953929	0.336887	0.016636	0.198556
int_rate	0.168921	1.000000	0.162758	-0.056771	0.079038	0.011649
installment	0.953929	0.162758	1.000000	0.330381	0.015786	0.188973
annual_inc	0.336887	-0.056771	0.330381	1.000000	-0.081685	0.136150
dti	0.016636	0.079038	0.015786	-0.081685	1.000000	0.136181
open_acc	0.198556	0.011649	0.188973	0.136150	0.136181	1.000000
pub_rec	-0.077779	0.060986	-0.067892	-0.013720	-0.017639	-0.018392
revol_bal	0.328320	-0.011280	0.316455	0.299773	0.063571	0.221192
revol_util	0.099911	0.293659	0.123915	0.027871	0.088375	-0.131420
total_acc	0.223886	-0.036404	0.202430	0.193023	0.102128	0.680728
mort_acc	0.222315	-0.082583	0.193694	0.236320	-0.025439	0.109205
pub_rec_bankruptcies	-0.106539	0.057450	-0.098628	-0.050162	-0.014558	-0.027732

It is a hard to read and visualise above correlation values. So let us try heatmap to visualise it:

In [7]:

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True,cmap="YlGnBu")
```

Out[7]:



You should have noticed almost perfect correlation with the "installment" feature. loan_amnt has 0.95 correlation with 'installment' and it is quiet intersting.So, let us explore this feature further. But way we want to explor this furthur?

The reason : We do not want to accedintly leaking of data from the features into our label. We do not want to see that a single label is a perfect predictor of a label. Because, it indicates that it is not really a feature but it is probably just some duplicated information very similar to the label. Let's go ahead and print that out:

In [8]:
```
feat_info('installment')
```

Here the keyword is "if"!

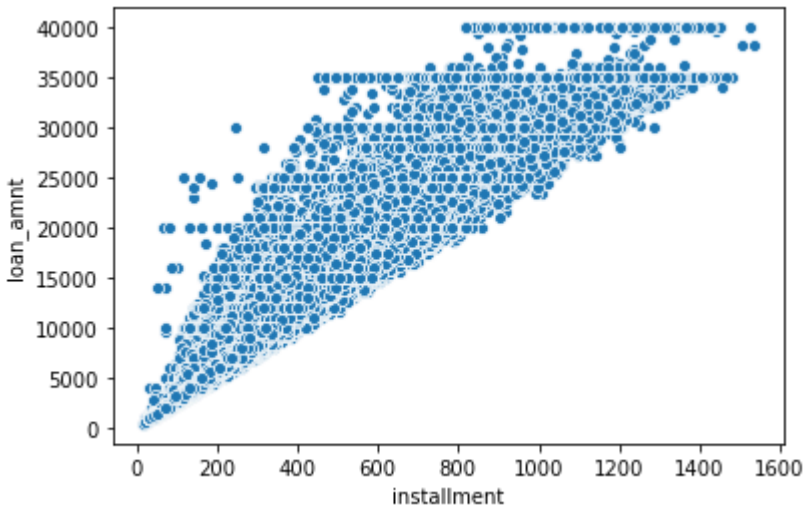
In [9]:
```
feat_info('loan_amnt')
```

It pretty much makes sense that the "instalment" and "loan_amnt" would have be extremely correlated. Because, they are essentialy correlated by some sort of internal formula that this company uses. We also can do a scatter plot to confirm this and view this high correlation.

note: It is abvious that if a loan amount is high, the monthly payment would be high.

In [10]:
```
sns.scatterplot(x='installment',y='loan_amnt',data=df)
```

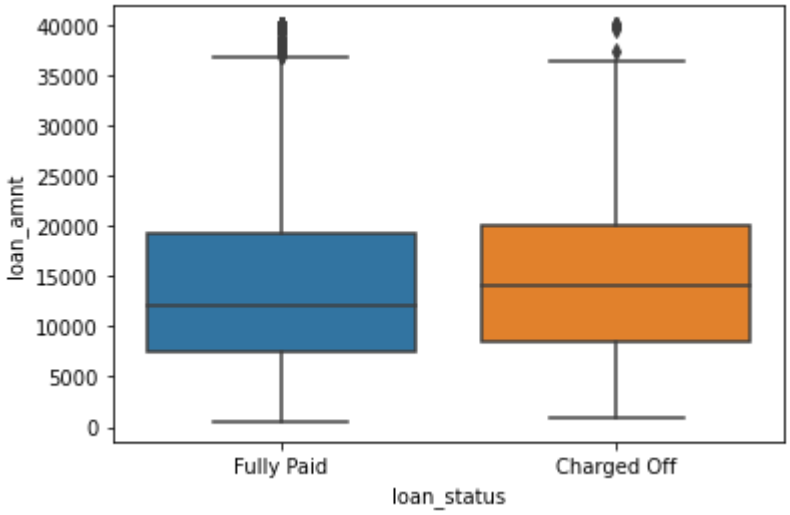
Out[10]:



let's creat a boxplot showing the relationship between the loan_status and the loan amount.

In [11]:
```
sns.boxplot(x='loan_status',y='loan_amnt',data=df)
```

Out[11]:



In gerneral it looks like both similar. Charged off is slightly higher, meaning that if our loan amount is highr we have a slight increase in liklihood of being charded off. Which again it intuitivly makes sense that it is hard to pay back larger loans than smaller loans. We can calculate the summary of statistics for the loan amount by the loan status:

This is a quantitative description of the above box plot:

In [12]:
```
df.groupby('loan_status')['loan_amnt'].describe()
```

Out[12]:

	count	mean	std	min	25%	50%	75%
loan_status							
Charged Off	77673.0	15126.300967	8505.090557	1000.0	8525.0	14000.0	20000.0
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0

Let's explore the Grade and SubGrade columns that LendingClub attributes to the loans. What are the unique possible grades and subgrades?

Note: These columns are catagoricals and not numerical. So it is abvius that you can not investigate corr() method on it. Although, you can investigate its relation with output by simply use countplot by specifying output label. So, you can observe for each specific catagorical feature what how many belongs to different output class!

In [13]:
```
df['grade'].unique()
```

Out[13]:

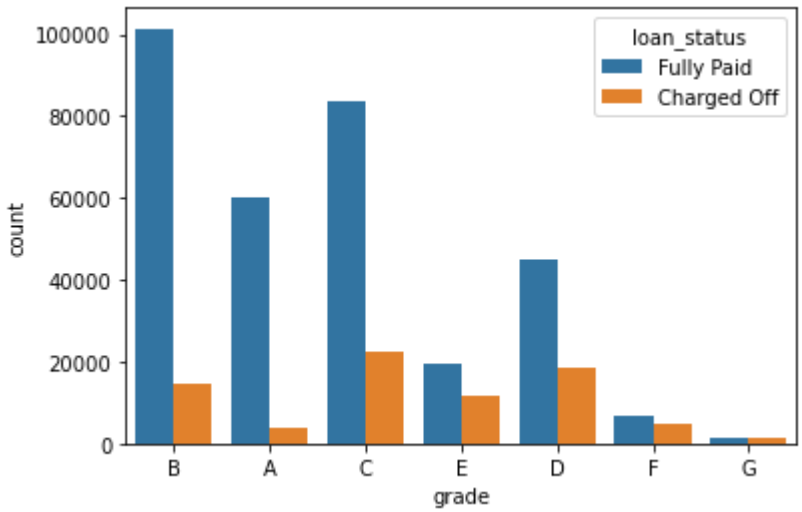
In [14]:
```
df['sub_grade'].unique()
```

Out[14]:

Let's check if we have any relation with grade and the label:

In [15]:
```
sns.countplot(x='grade',data=df, hue='loan_status')
```

Out[15]:

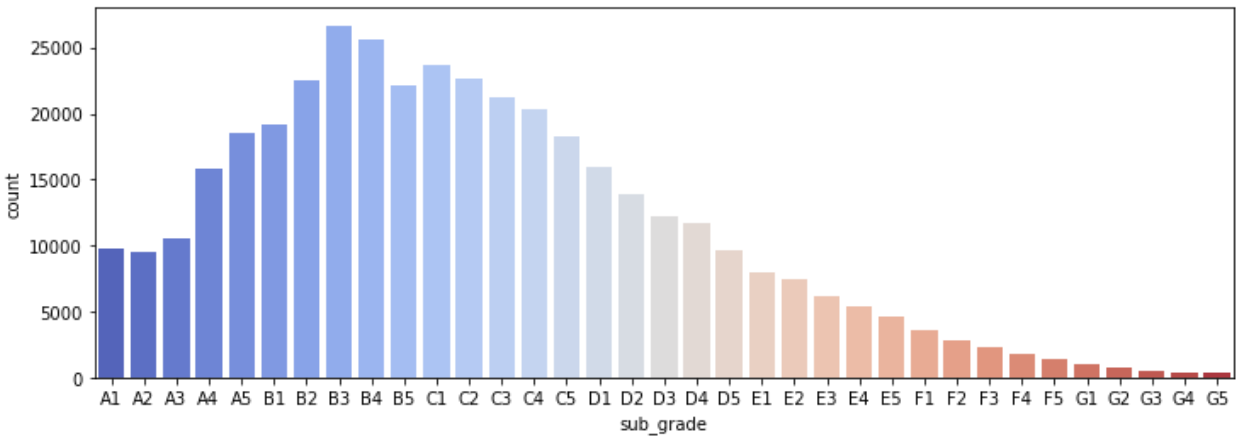


Here, we can see clear relationship , but it is hard to tell the order of grade! So we may need to reorder them. The percentage of charged off loans seem increasing as the letter grade gets higher. Actually we can compare it by ratios. However, let's do it with subgrades

Display a count plot per subgrade. You may need to resize for this plot and reorder the x axis. Feel free to edit the color palette. Explore both all loans made per subgrade as well being separated based on the loan_status. After creating this plot, go ahead and create a similar plot, but set hue="loan_status"

In [16]:
```
plt.figure(figsize=(12,4))
subgrade_order = sorted(df['sub_grade'].unique())
sns.countplot(x='sub_grade',data=df,order=subgrade_order,palette='coolwarm')
```

Out[16]:

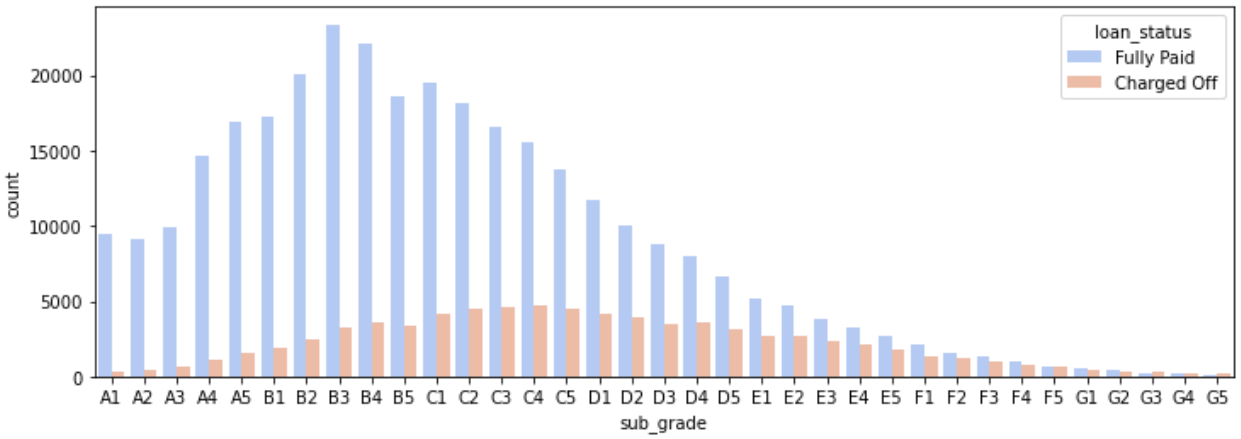


Nice, we can see all sub_grades counts. Now, let's see the relation of sub_grades with the output.

In [17]:

```
plt.figure(figsize=(12,4))
subgrade_order = sorted(df['sub_grade'].unique())
sns.countplot(x='sub_grade',data=df,order=subgrade_order,palette='coolwarm',hue='loan_status')
```

Out[17]:



You see that first groups starting from A group are commenly pay back their loans. But this trend decreasing as we continue to the rest. Think about these groups. In description of sub_grade you can see that it is a grade assigned by lending club. Why and how lending clubs assign this feature or information on each person!? What is the forumla and the reason behind that? It looks like that as grades goes down the loan status get worse and people can not pay their loan back.Is it kind of prediction performed by lending club? These are all questions that we should answer? We also need to talk to expert from lending club to get more insight.

Let's continue our analysis. It looks like F and G subgrades don't get paid back that often. Isloate those and recreate the countplot just for those subgrades.

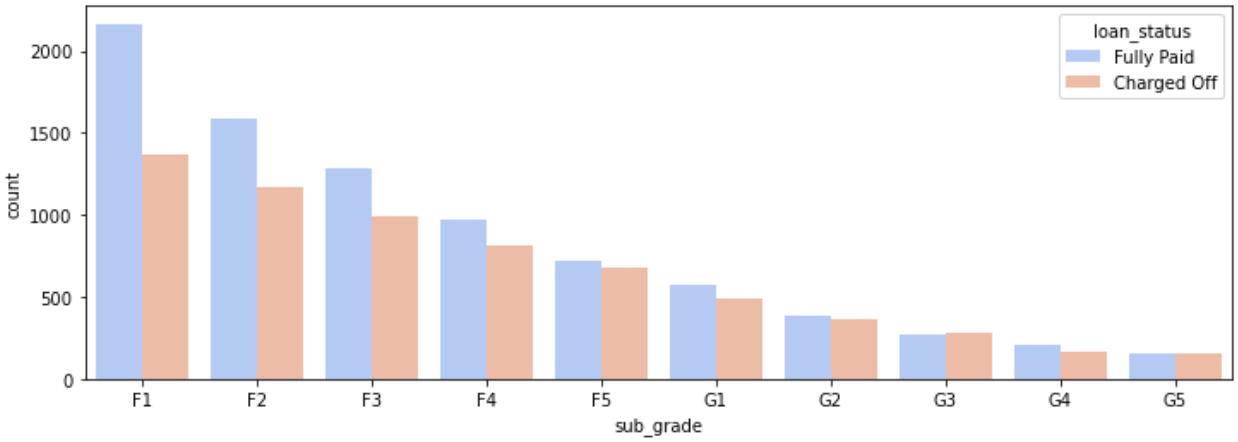
let's zoom at that section of the plot:

In [19]:

```
f_and_g = df[(df['grade']=='G') | (df['grade']=='F')]

plt.figure(figsize=(12,4))
subgrade_order = sorted(f_and_g['sub_grade'].unique())
sns.countplot(x='sub_grade',data=f_and_g,order=subgrade_order,palette='co
olwarm',hue='loan_status')
```

Out[19]:



Here, you can see the statuses for almost the worse subgrades. For example, for G5 the liklihood almost same for both paid of and charged off.

Changing the catagorical label of loan_status to numerical one:

By doing this we are able use it in our model.

In [23]:

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

In [24]:
```
df[['loan_repaid','loan_status']]
```

Out[24]:

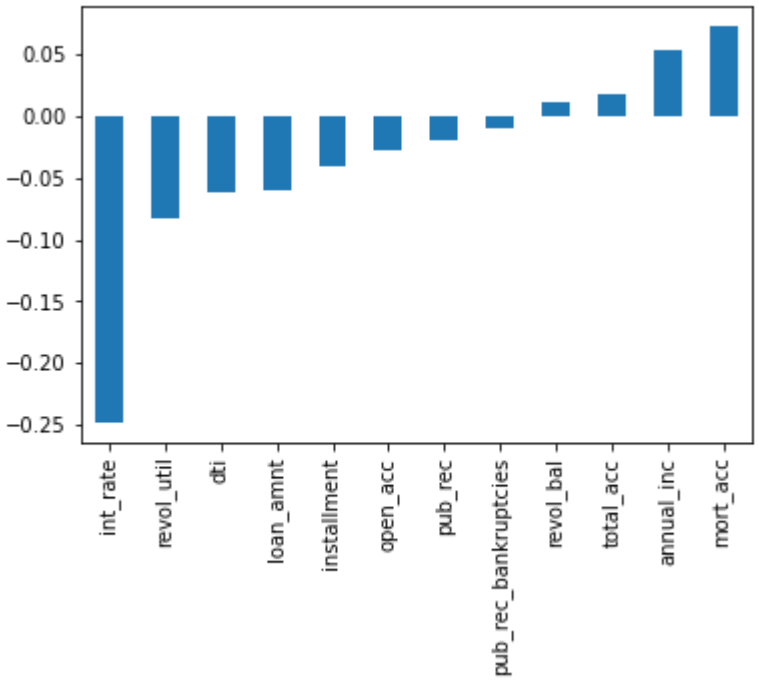
	loan_repaid	loan_status
0	1	Fully Paid
1	1	Fully Paid
2	1	Fully Paid
3	1	Fully Paid
4	0	Charged Off
...	...	...
396025	1	Fully Paid
396026	1	Fully Paid
396027	1	Fully Paid
396028	1	Fully Paid
396029	1	Fully Paid

396030 rows × 2 columns

Now, we can see which one of our features has highest correlation with the label(it is numerical now and the name is loan_repaid):

In [25]:
```
df.corr()['loan_repaid'].sort_values().drop('loan_repaid').plot(kind='ba
r')
```

Out[25]:



You can see that int_rate has high negative correlation with loan repaid. It actually makes sene that with high interest rate it is acutally hard to repay the loan!

You can further explore the data base on your interset, expert knowlege , or google some info about the problem and gain some knowledge about this specific task and use it as expert domain knowledge! It is obt to you. But it is enough for now let go to the next section.

Data Preprocessing

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

In [28]:
```
df.head()
```

Out[28]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	ho
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RI
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	M
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RI
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RI
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	M

5 rows × 28 columns

Missing Data

Let's explore this missing data columns. We use a variety of factors to decide whether or not they would be useful, to see if we should keep, discard, or fill in the missing data.

So what is the length of the dataframe?

In [29]:
```
len(df)
```

Out[29]:

let's create a Series that displays the total count of missing values per column.

In [30]:
```
df.isnull().sum()
```

Out[30]:

You see some values are missing. In order to have better insight that what percengtage of the total Datafram is missing you can calculate the percentage:

In [31]:
```
100 *df.isnull().sum()/len(df)
```

Out[31]:

emp_title and emp_length have almost 5 percent missing data. But, let's see what are these features. Let's see even if we could drop those columns or not. We can print out their information using the feat_info() function which we have defined at the first of the notbook:

In [32]:
```
feat_info('emp_title')
print('\n')
feat_info('emp_length')
```

In catagorical features , the important thing that we should always consider is uniqueness of its values. We actully want to convert catagorical featurs to numerical through dummy varaible. But, if there are too many unique sample from a feature, then it makes harder to convert it to dummy variable feature. Let's check it the problem here;

In [33]:
```
df['emp_title'].nunique()
```

Out[33]:

In [34]:
```
df['emp_title'].value_counts()
```

Out[34]:

Realistically there are too many unique job titles to try to convert this to a dummy variable feature. In fact this columns is nominal data which is a varibale that has no numerical importance, such as occupation, person name etc. let's remove this column.

In [35]:
```
df = df.drop('emp_title',axis=1)
```

Let's now explore emp_length feature column. It is kind of timeseries data. Time series data has a temporal value attached to it, so this would be something like a date or a time stamp that you can look for trends in time.

In [36]:
```
sorted(df['emp_length'].dropna().unique())
```

Out[36]:

In [37]:

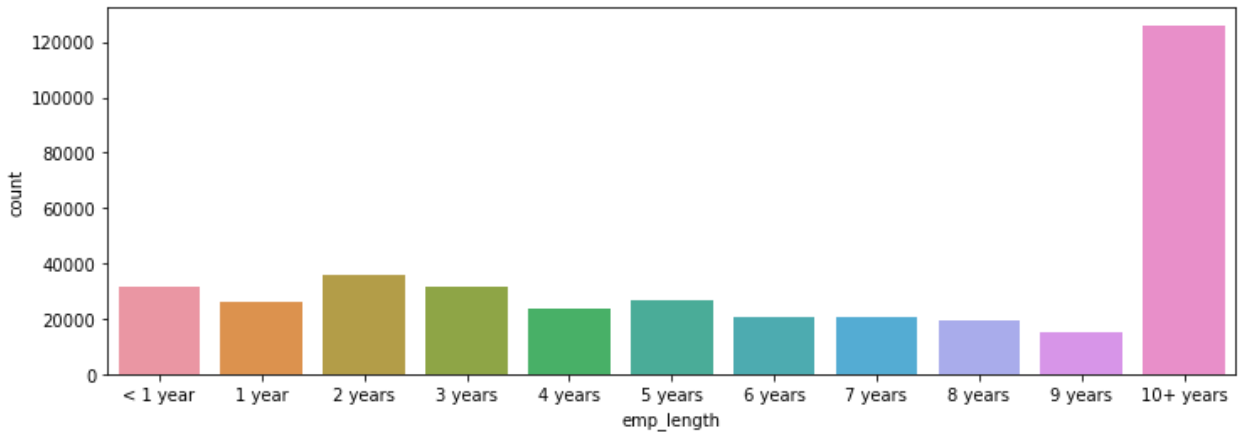
```
emp_length_order = [ '< 1 year',
                     '1 year',
                    '2 years',
                    '3 years',
                    '4 years',
                    '5 years',
                    '6 years',
                    '7 years',
                    '8 years',
               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

In [38]:

```
plt.figure(figsize=(12,4))

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

Out[38]:

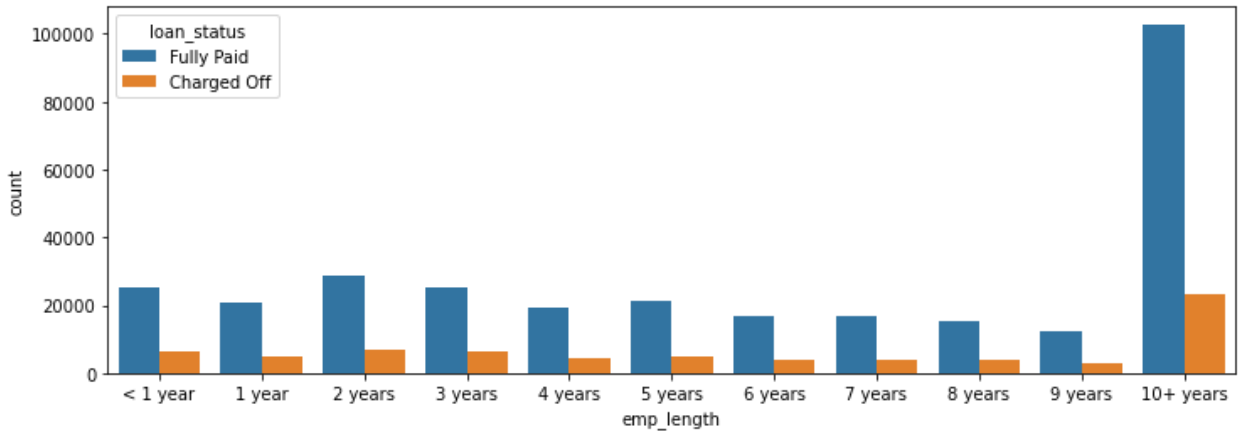


Now, we can also plot out the countplot with a hue separating Fully Paid vs Charged Off. Because, the most important thing for us here is if a person paid back his loan or not?!

In [39]:

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

Out[39]:

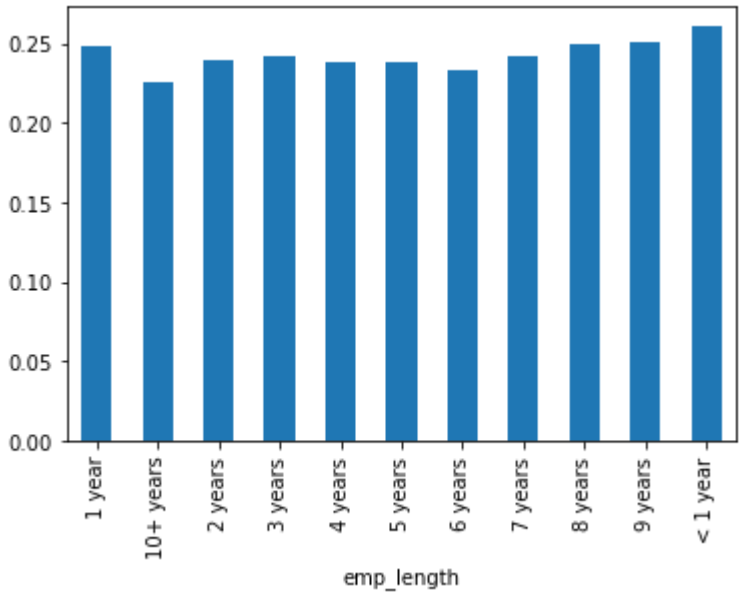


CHALLENGE TASK: This still doesn't really inform us if there is a strong relationship between employment length and being charged off, what we want is the percentage of charge offs per category. Essentially informing us what percent of people per employment category didn't pay back their loan. There are a multitude of ways to create this Series. Once you've created it, see if visualize it with a bar plot (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.plot.html). This may be tricky, refer to solutions if you get stuck on creating this Series.

In [40]:

```
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.plot(kind='bar')
```

Out[40]:



Charge off rates are extremely similar across all employment lengths. Go ahead and drop the emp_length column.

In [41]:

```
df = df.drop('emp_length',axis=1)
```

Ok, let's investigate other columns with missing data

In [42]:
```
df.isnull().sum()
```

Out[42]:

We have title value with some missing data. But if you explore more, you can find out that the purpose column is actully contains repeated information of title group.

In [43]:
```
df['purpose'].head(10)
```

Out[43]:

In [44]:
```
df['title'].head(10)
```

Out[44]:

You see we do not need title column actually. Let's drop it.

In [45]:
```
df = df.drop('title',axis=1)
```

We continue with the rest columns containing missing values.

In [46]:

```
feat_info('mort_acc')
print('\n')
df['mort_acc'].head(10)
```

Out[46]:

AS you see it is a numerical feature, and has 37795 missing data. Is this feature important to us. Do you think it has relation with the target output. Let's create a value_counts of the mort_acc column.

In [47]:

```
df['mort_acc'].value_counts()
```

Out[47]:

There are many ways we could deal with this missing data. We could attempt to build a simple model to fill it in, such as a linear model, we could just fill it in based on the mean of the other columns, or you could even bin the columns into categories and then set NaN as its own category. There is no 100% correct approach! Let's review the other columsn to see which most highly correlates to mort_acc

In [48]:
```
print("Correlation with the mort_acc column")
df.corr()['mort_acc'].sort_values()
```

Out[48]:

Looks like the total_acc feature correlates with the mort_acc , this makes sense! Let's try this fillna() approach. We will group the dataframe by the total_acc and calculate the mean value for the mort_acc per total_acc entry.

In [49]:
```
print("Mean of mort_acc column per total_acc")
df.groupby('total_acc').mean()['mort_acc']
```

Out[49]:

Let's fill in the missing mort_acc values based on their total_acc value. If the mort_acc is missing, then we will fill in that missing value with the mean value corresponding to its total_acc value from the Series we created above. This involves using an .apply() method with two columns. Check out the link below for more info, or review the solutions video/notebook.

Helpful Link (https://stackoverflow.com/questions/13331698/how-to-apply-a-function-to-two-columns-of-pandas-dataframe)

In [50]:
```
total_acc_avg = df.groupby('total_acc').mean()['mort_acc']
total_acc_avg[2.0]
```

Out[50]:

In [51]:

```
def fill_mort_acc(total_acc,mort_acc):
    '''
    Accepts the total_acc and mort_acc values for the row.
    Checks if the mort_acc is NaN , if so, it returns the avg mort_acc valu
e
    for the corresponding total_acc value for that row.

    total_acc_avg here should be a Series or dictionary containing the mapp
ing of the
    groupby averages of mort_acc per total_acc values.
    '''
    if np.isnan(mort_acc):
        return total_acc_avg[total_acc]
    else:
        return mort_acc
```

In [52]:

```
df['mort_acc'] = df.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort
_acc']), axis=1)
```

In [53]:
```
df.isnull().sum()
```

Out[53]:

revol_util and the pub_rec_bankruptcies have missing data points, but they account for less than 0.5% of the total data. Go ahead and remove the rows that are missing those values in those columns with dropna().

In [54]:
```
df = df.dropna()
```

In [55]:
```
df.isnull().sum()
```

Out[55]:

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.
Note: For more informaiton on Categorical feature encoding, you could check my other kernal on Categorical feature encoding. (https://www.kaggle.com/hadiyad/categorical-feature-encoding-skill)

Note: A categorical variable is a variable that can take some limited number of values.for example,day of the week.It can be one of 1,2,3,4,5,6,7 only.

We can list all the columns that are currently non-numeric. For more info visit: Helpful Link (https://stackoverflow.com/questions/22470690/get-list-of-pandas-dataframe-columns-based-on-data-type)

Another very useful method call (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.select_dtypes.html)

In [56]:
```
df.select_dtypes(['object']).columns
```

Out[56]:

Let's now go through all the string features to see what we should do with them.

1. **'term'** feature:

Let's convert the "term" feature into either a 36 or 60 integer numeric data type using .apply() or .map().

In [57]:
```
df['term'].value_counts()
```

Out[57]:

In [58]:
```
df['term'] = df['term'].apply(lambda term: int(term[:3]))
```

2. **'grade'** feature:

We already know grade is part of sub_grade, so just drop the grade feature.

In [59]:
```
df = df.drop('grade',axis=1)
```

You could convert the subgrade into dummy variables. Then concatenate these new columns to the original dataframe. Remember you always drop the original subgrade column and add drop_first=True to your get_dummies call.

In [60]:
```
subgrade_dummies = pd.get_dummies(df['sub_grade'],drop_first=True)
df = pd.concat([df.drop('sub_grade',axis=1),subgrade_dummies],axis=1)
```

If we check now df we will have more columns, because, more subgrade columns are added through one hot encoding:

In [61]:
```
df.columns
```

Out[61]:

3. '**verification_status', 'application_type','initial_list_status','purpose**' ' features:

Let's convert also these columns: ['verification_status', 'application_type','initial_list_status','purpose'] into dummy variables and concatenate them with the original dataframe. The reason is these columns are really good candidate to be converted to dummy variables, because their values are few catagories. Remember to set drop_first=True and to drop the original columns.

In [62]:
```
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. **'home_ownership'** feature:

First, review the value_counts for the home_ownership column.

In [63]:
```
df['home_ownership'].value_counts()
```

Out[63]:

You can convert these to dummy variables, but replace (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html) NONE and ANY with OTHER, so that we end up with just 4 categories, MORTGAGE, RENT, OWN, OTHER. Then concatenate them with the original dataframe. Remember to set drop_first=True and to drop the original columns.

In [64]:
```
df['home_ownership']=df['home_ownership'].replace(['NONE', 'ANY'], 'OTHE
R')

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

5. **'address'** feature:

Let's feature engineer a zip code column from the address in the data set. Create a column called 'zip_code' that extracts the zip code from the address column.

In [65]:
```
df['zip_code'] = df['address'].apply(lambda address:address[-5:])
```

Now let's make this zip_code column into dummy variables using pandas. Concatenate the result and drop the original zip_code column along with dropping the address column.

In [66]:
```
dummies = pd.get_dummies(df['zip_code'],drop_first=True)
df = df.drop(['zip_code','address'],axis=1)
df = pd.concat([df,dummies],axis=1)
```

6. **'issue_d '** feature:

This would be data leakage, we wouldn't know beforehand whether or not a loan would be issued when using our model, so in theory we wouldn't have an issue_date, so we drop this feature.

In [67]:
```
df = df.drop('issue_d',axis=1)
```

7. **'earliest_cr_line'** feature:

This appears to be a historical time stamp feature. We can extract the year from this feature using a .apply function, then convert it to a numeric feature. We set this new data to a feature column called 'earliest_cr_year'.Then we can drop the earliest_cr_line feature.

In [68]:
```
df['earliest_cr_year'] = df['earliest_cr_line'].apply(lambda date:int(dat
e[-4:]))
df = df.drop('earliest_cr_line',axis=1)
```

Great! we are done with all categorical and no onve left. let's check it again!:

In [69]:
```
df.select_dtypes(['object']).columns
```

Out[69]:

We only have 'loan_status' left, and if you remember we converted this catagorical label to numerical label as 'loan_repaid'!

In [70]:
```
df[['loan_status','loan_repaid']]
```

Out[70]:

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

395219 rows × 2 columns

So far so good, we have finished EDA and some feature engineering. We are ready to go to the next step. The training and modeling. We now use these clean, and well prepared data to build a machine learning model! Let's begin modlleing But before that let's check all columns and features that we have now:

In [71]:
```
df.columns
```

Out[71]:

In [ ]:
```

```

Train Test Split

Let's first split our train and test dataset as always we do before creating model. The reason is that we want to test the model after training.

In [72]:
```
from sklearn.model_selection import train_test_split
```

we drop the load_status column we created earlier, since its a duplicate of the loan_repaid column. We'll use the loan_repaid column since its already in 0s and 1s.

In [73]:
```
df = df.drop('loan_status',axis=1)
```

We set X and y variables to the .values of the features and label.

In [74]:
```
X = df.drop('loan_repaid',axis=1).values
y = df['loan_repaid'].values
```

And let's perform a train/test split with test_size=0.2,

In [75]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=101)
```

Normalizing the Data

We use a MinMaxScaler to normalize the feature data X_train and X_test. We don't want data leakge from the test set so we only fit on the X_train data.

In [76]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Creating the Model

Run the cell below to import the necessary Keras functions.

In [77]:
```
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
```

Now, we can build a sequential model to will be trained on the data. You have unlimited options here, but here is what the solution uses: a model that goes 78 → 39 → 19→ 1 output neuron. OPTIONAL: Explore adding Dropout layers (https://keras.io/layers/core/) 1 (https://en.wikipedia.org/wiki/Dropout_(neural_networks) and 2 (https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab)

In [78]:

```
model = Sequential()

# https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of
-hidden-layers-and-nodes-in-a-feedforward-neural-netw



# input layer
model.add(Dense(78,  activation='relu'))
model.add(Dropout(0.2))

# hidden layer
model.add(Dense(39, activation='relu'))
model.add(Dropout(0.2))

# hidden layer
model.add(Dense(19, activation='relu'))
model.add(Dropout(0.2))

# output layer
model.add(Dense(units=1,activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam')
```

Now, it is time to fit our model to the training data. We can 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.

In [79]:

```
model.fit(x=X_train,
          y=y_train,
          epochs=25,
          batch_size=256,
          validation_data=(X_test, y_test),
          )
```

Out[79]:

let's also save our model to outuput.

In [80]:
```
from tensorflow.keras.models import load_model
```

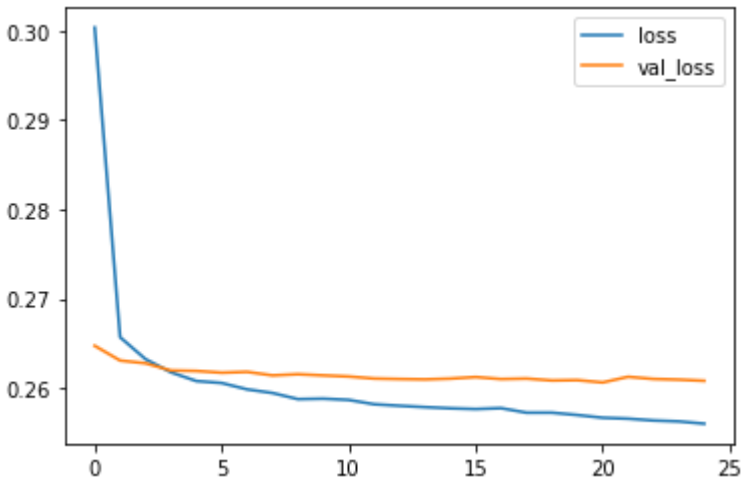
In [81]:
```
model.save('full_data_project_model.h5')
```

Evaluating Model Performance

Yes , it is time to evaluate our model.

In [82]:
```
losses = pd.DataFrame(model.history.history)
losses[['loss','val_loss']].plot()
```

Out[82]:



Now, let's create predictions from the X_test set and display a classification report and confusion matrix for the X_test set.

In [83]:
```
from sklearn.metrics import classification_report,confusion_matrix
predictions = model.predict_classes(X_test)
print(classification_report(y_test,predictions))
```

In [84]:
```
confusion_matrix(y_test,predictions)
```

Out[84]:

The results are kind of satisfactory. However, we can do further works to even get better results.

It is time that we use our model to see if we want to offer a customer a loan or not?

Given the customer below, would you offer this person a loan? We randomly select one of costumers and we want to see if the model works well or not!

In [85]:
```
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
```

Out[85]:

In [86]:
```
model.predict_classes(new_customer.values.reshape(1,78))
```

Out[86]:

Now check, did this person actually end up paying back their loan?

In [87]:
```
df.iloc[random_ind]['loan_repaid']
```

Out[87]:

Nice and great job!

In [ ]: