Keras API Project Exercise - Solutions

The Data

We will be using a subset of the LendingClub DataSet obtained from Kaggle: https://www.kaggle.com/wordsforthewise/lending-club

NOTE: Do not download the full zip from the link! We provide a special version of this file that has some extra feature engineering for you to do. You won't be able to follow along with the original file!

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.

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.

Data Overview

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

Starter Code

Note: We also provide feature information on the data as a .csv file for easy lookup throughout the notebook:

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

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

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

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

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

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

```
Number of mortgage accounts.
```

Loading the data and other imports

In [6]:
```
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
```

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

In [8]:
```
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
```

Project Tasks

Complete the tasks below! Keep in mind is usually more than one way to complete the task! Enjoy

---

Section 1: Exploratory Data Analysis

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

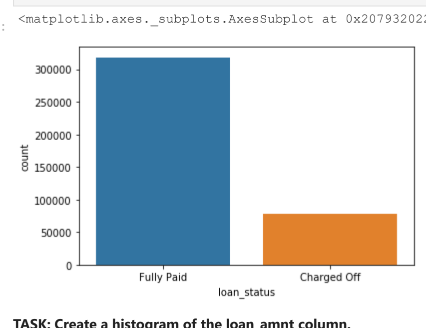
---

TASK: Since we will be attempting to predict loan_status, create a countplot as shown below.

In [9]:
```
# CODE HERE
```

In [10]:
```
sns.countplot(x='loan_status',data=df)
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x207932022c8>
```

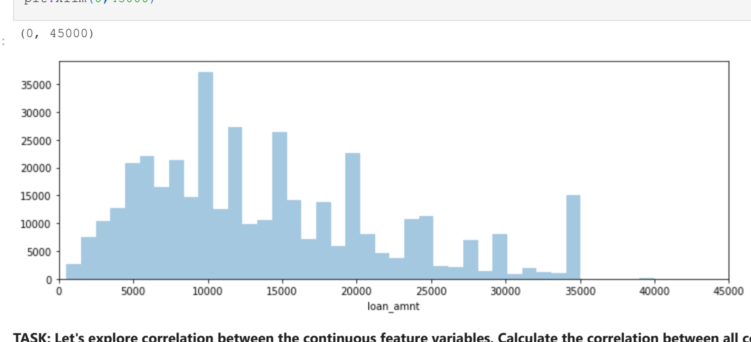


TASK: Create a histogram of the loan_amnt column.

In [11]:
```
# CODE HERE
```

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

Out[12]:
```
(0, 45000)
```



TASK: Let's explore correlation between the continuous feature variables. Calculate the correlation between all continuous numeric variables using .corr() method.

In [13]:
```
# CODE HERE
```

In [14]:
```
df.corr()
```

Out[14]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_acc
loan_amnt	1.000000	0.168921	0.953929	0.336887	0.016636	0.198556	-0.077779	0.328320	0.099911	0.223886	0.222315
int_rate	0.168921	1.000000	0.162758	-0.056771	0.079038	0.011649	0.060986	-0.011280	0.293659	-0.036404	-0.082583
installment	0.953929	0.162758	1.000000	0.330381	0.015786	0.188973	-0.067892	0.316455	0.123915	0.202430	0.193694
annual_inc	0.336887	-0.056771	0.330381	1.000000	-0.081685	0.136150	-0.013720	0.299773	0.027871	0.193023	0.236320
dti	0.016636	0.079038	0.015786	-0.081685	1.000000	0.136181	-0.017639	0.063571	0.088375	0.102128	-0.025439
open_acc	0.198556	0.011649	0.188973	0.136150	0.136181	1.000000	-0.018392	0.221192	-0.131420	0.680728	0.109205
pub_rec	-0.077779	0.060986	-0.067892	-0.013720	-0.017639	-0.018392	1.000000	-0.101664	-0.075910	0.019723	0.011552
revol_bal	0.328320	-0.011280	0.316455	0.299773	0.063571	0.221192	-0.101664	1.000000	0.226346	0.191616	0.194925
revol_util	0.099911	0.293659	0.123915	0.027871	0.088375	-0.131420	-0.075910	0.226346	1.000000	-0.104273	0.007514
total_acc	0.223886	-0.036404	0.202430	0.193023	0.102128	0.680728	0.019723	0.191616	-0.104273	1.000000	0.381072
mort_acc	0.222315	-0.082583	0.193694	0.236320	-0.025439	0.109205	0.011552	0.194925	0.007514	0.381072	1.000000
pub_rec_bankruptcies	-0.106539	0.057450	-0.098628	-0.050162	-0.014558	-0.027732	0.699408	-0.124532	-0.086751	0.042035	0.027239

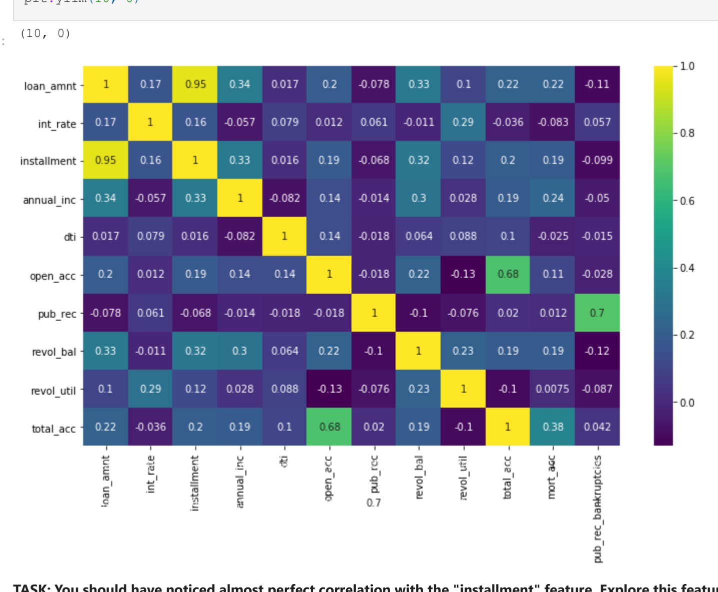
TASK: Visualize this using a heatmap. Depending on your version of matplotlib, you may need to manually adjust the heatmap.

- Heatmap info
- Help with resizing

In [15]:
```
# CODE HERE
```

In [16]:
```
plt.figure(figsize=(12,7))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
plt.ylim(10, 0)
```

Out[16]:
```
(10, 0)
```



TASK: You should have noticed almost perfect correlation with the "installment" feature. Explore this feature further. Print out their descriptions and perform a scatterplot between them. Does this relationship make sense to you? Do you think there is duplicate information here?

In [17]:
```
# CODE HERE
```

In [18]:
```
feat_info('installment')
```

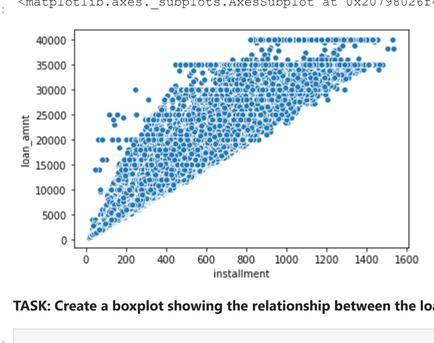
```
The monthly payment owed by the borrower if the loan originates.
```

In [19]:
```
feat_info('loan_amnt')
```

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

In [20]:
```
sns.scatterplot(x='installment',y='loan_amnt',data=df,)
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20798026f48>
```

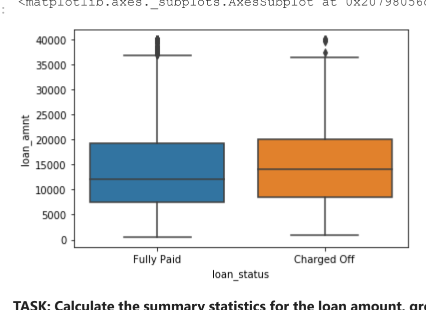


TASK: Create a boxplot showing the relationship between the loan_status and the Loan Amount.

In [21]:
```
# CODE HERE
```

In [22]:
```
sns.boxplot(x='loan_status',y='loan_amnt',data=df)
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20798056c48>
```



TASK: Calculate the summary statistics for the loan amount, grouped by the loan_status.

In [23]:
```
# CODE HERE
```

In [24]:
```
df.groupby('loan_status')['loan_amnt'].describe()
```

Out[24]:

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
Fully Paid	318357.0	13866.878771	8302.319699	500.0	7500.0	12000.0	19225.0	40000.0

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

In [25]:
```
# CODE HERE
```

In [26]:
```
sorted(df['grade'].unique())
```

Out[26]:
```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

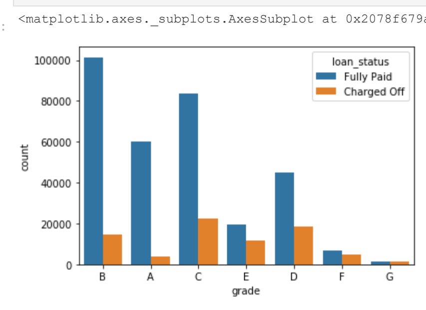
In [27]:
```
sorted(df['sub_grade'].unique())
```

Out[27]:
```
['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']
```

TASK: Create a countplot per grade. Set the hue to the loan_status label.

In [28]:
```
sns.countplot(x='grade',data=df,hue='loan_status')
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2078f679ac8>
```

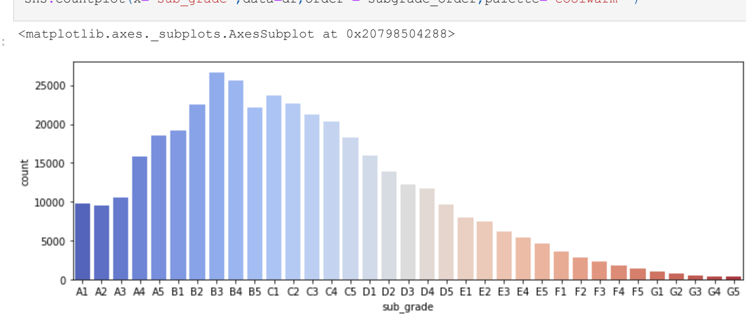


TASK: Display a count plot per subgrade. You may need to resize for this plot and reorder the x axis. Feel free to edit the color palette. Explore both all loans made per subgrade as well being separated based on the loan_status

In [29]:
```
#CODE HERE
```

In [30]:
```
plt.figure(figsize=(12,4))
subgrade_order = sorted(df['sub_grade'].unique())
sns.countplot(x='sub_grade',data=df,order = subgrade_order,palette='coolwarm' )
```

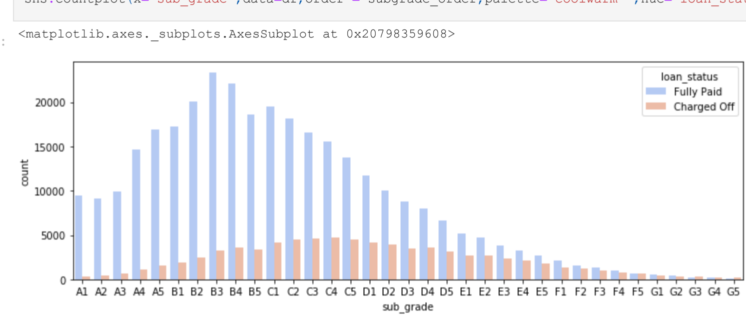
Out[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20798504288>
```



In [31]:
```
# CODE HERE
```

In [32]:
```
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[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20798359608>
```



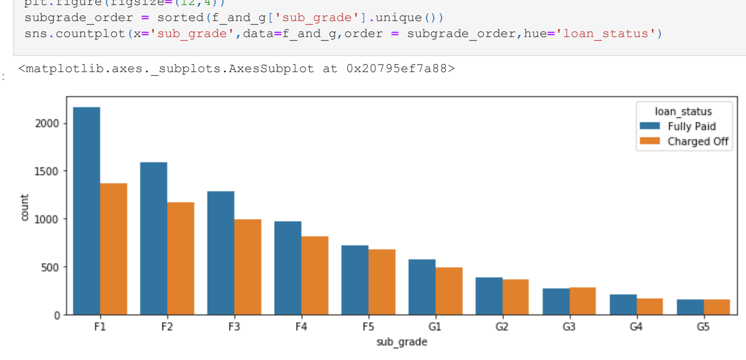
TASK: It looks like F and G subgrades don't get paid back that often. Isloate those and recreate the countplot just for those subgrades.

In [33]:
```
# CODE HERE
```

In [34]:
```
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,hue='loan_status')
```

Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20795ef7a88>
```



TASK: Create a new column called 'load_repaid' which will contain a 1 if the loan status was "Fully Paid" and a 0 if it was "Charged Off".

In [35]:
```
# CODE HERE
```

In [36]:
```
df['loan_status'].unique()
```

Out[36]:
```
array(['Fully Paid', 'Charged Off'], dtype=object)
```

In [37]:
```
df['loan_repaid'] = df['loan_status'].map({'Fully Paid':1,'Charged Off':0})
```

In [38]:
```
df[['loan_repaid','loan_status']]
```

Out[38]:

	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

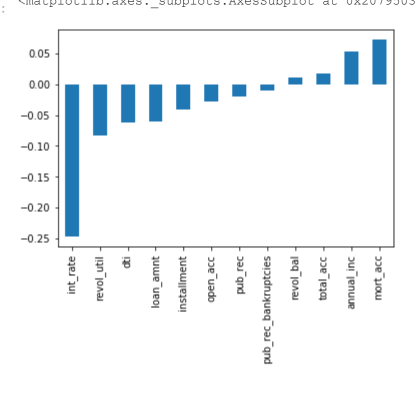
396030 rows × 2 columns

CHALLENGE TASK: (Note this is hard, but can be done in one line!) Create a bar plot showing the correlation of the numeric features to the new loan_repaid column. Helpful Link

In [39]:
```
#CODE HERE
```

In [40]:
```
df.corr()['loan_repaid'].sort_values().drop('loan_repaid').plot(kind='bar')
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20795034cc8>
```



Section 2: Data PreProcessing

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

In [41]:
```
df.head()
```

Out[41]:

	loan_amnt	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	home_ownership	annual_inc	...	pub_rec	revol_bal
0	10000.0	36 months	11.44	329.48	B	B4	Marketing	10+ years	RENT	117000.0	...	0.0	36369.0
1	8000.0	36 months	11.99	265.68	B	B5	Credit analyst	4 years	MORTGAGE	65000.0	...	0.0	20131.0
2	15600.0	36 months	10.49	506.97	B	B3	Statistician	< 1 year	RENT	43057.0	...	0.0	11987.0
3	7200.0	36 months	6.49	220.65	A	A2	Client Advocate	6 years	RENT	54000.0	...	0.0	5472.0
4	24375.0	60 months	17.27	609.33	C	C5	Destiny Management Inc.	9 years	MORTGAGE	55000.0	...	0.0	24584.0

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.

TASK: What is the length of the dataframe?

In [42]:
```
# CODE HERE
```

In [43]:
```
len(df)
```

Out[43]:
```
396030
```

TASK: Create a Series that displays the total count of missing values per column.

In [44]:
```
# CODE HERE
```

In [45]:
```
df.isnull().sum()
```

Out[45]:
```
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
```

TASK: Convert this Series to be in term of percentage of the total DataFrame

In [46]:
```
# CODE HERE
```

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

TASK: Let's examine emp_title and emp_length to see whether it will be okay to drop them. Print out their feature information using the feat_info() function from the top of this notebook.

In [48]:
```
# CODE HERE
```

In [49]:
```
feat_info('emp_title')
print('\n')
feat_info('emp_length')
```

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


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.
```

TASK: How many unique employment job titles are there?

In [50]:
```
# CODE HERE
```

In [51]:
```
df['emp_title'].nunique()
```

Out[51]:
```
173105
```

In [52]:
```
df['emp_title'].value_counts()
```

Out[52]:
```
Teacher                            4389
Manager                            4250
Registered Nurse                   1856
RN                                 1846
Supervisor                         1830
                                   ...
mechanic/lead                         1
SUPV. MECHINE SHOP                    1
Mcccd                                 1
Dr. Dennis Norkiewicz DDS             1
bernie little distributing llc        1
Name: emp_title, Length: 173105, dtype: int64
```

TASK: Realistically there are too many unique job titles to try to convert this to a dummy variable feature. Let's remove that emp_title column.

In [53]:
```
# CODE HERE
```

In [54]:
```
df = df.drop('emp_title',axis=1)
```

TASK: Create a count plot of the emp_length feature column. Challenge: Sort the order of the values.

In [55]:
```
# CODE HERE
```

In [56]:
```
sorted(df['emp_length'].dropna().unique())
```

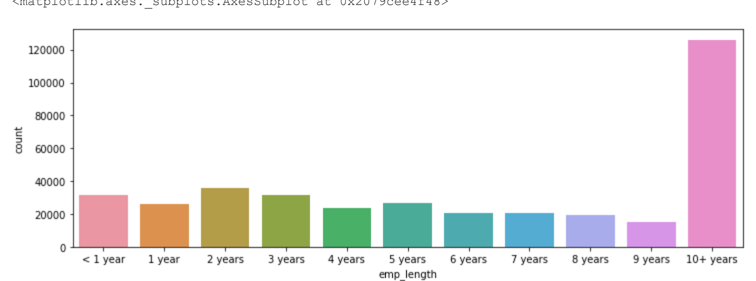
Out[56]:
```
['1 year',
 '10+ years',
 '2 years',
 '3 years',
 '4 years',
 '5 years',
 '6 years',
 '7 years',
 '8 years',
 '9 years',
 '< 1 year']
```

In [57]:
```
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 [58]:
```
plt.figure(figsize=(12,4))

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

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x2079cee4f48>

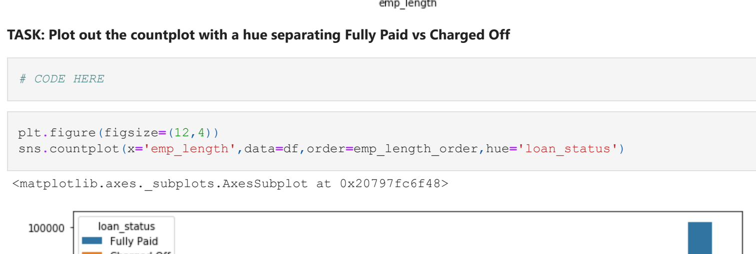


TASK: Plot out the countplot with a hue separating Fully Paid vs Charged Off

In [59]:
```
# CODE HERE
```

In [60]:
```
plt.figure(figsize=(12,4))
sns.countplot(x='emp_length',data=df,order=emp_length_order,hue='loan_status')
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x20797fc6f48>



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. This may be tricky, refer to solutions if you get stuck on creating this Series.

In [61]:
```
# CODE HERE
```

In [62]:
```
emp_co = df[df['loan_status']=="Charged Off"].groupby("emp_length").count()['loan_status']
```

In [63]:
```
emp_fp = df[df['loan_status']=="Fully Paid"].groupby("emp_length").count()['loan_status']
```

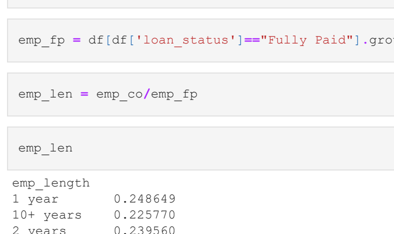
In [64]:
```
emp_len = emp_co/emp_fp
```

In [65]:
```
emp_len
```

Out[65]:
```
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
```

In [66]:
```
emp_len.plot(kind='bar')
```

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x20798297d88>



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

In [67]:
```
# CODE HERE
```

In [68]:
```
df = df.drop('emp_length',axis=1)
```

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

In [ ]:

In [69]:
```
df.isnull().sum()
```

Out[69]:
```
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?

In [70]:
```
# CODE HERE
```

In [71]:
```
df['purpose'].head(10)
```

Out[71]:
```
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
```

In [72]:
```
df['title'].head(10)
```

Out[72]:
```
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
```

TASK: The title column is simply a string subcategory/description of the purpose column. Go ahead and drop the title column.

In [73]:
```
# CODE HERE
```

In [74]:
```
df = df.drop('title',axis=1)
```

NOTE: This is one of the hardest parts of the project! Refer to the solutions video if you need guidance, feel free to fill or drop the missing values of the mort_acc however you see fit! Here we're going with a very specific approach.

TASK: Find out what the mort_acc feature represents

In [75]:
```
# CODE HERE
```

In [76]:
```
feat_info('mort_acc')
```

```
Number of mortgage accounts.
```

TASK: Create a value_counts of the mort_acc column.

In [77]:
```
# CODE HERE
```

In [78]:
```
df['mort_acc'].value_counts()
```

Out[78]:
```
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
```

TASK: 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 [ ]:

In [79]:
```
print("Correlation with the mort_acc column")
df.corr()['mort_acc'].sort_values()
```

Out[79]:
```
Correlation with the mort_acc column
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
```

TASK: 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. To get the result below:

In [ ]:

In [80]:
```
print("Mean of mort_acc column per total_acc")
df.groupby('total_acc').mean()['mort_acc']
```

Out[80]:
```
Mean of mort_acc column per total_acc
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
```

CHALLENGE TASK: 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

In [81]:
```
# CODE HERE
```

In [82]:
```
total_acc_avg = df.groupby('total_acc').mean()['mort_acc']
```

In [83]:
```
total_acc_avg[2.0]
```

Out[83]:
```
0.0
```

In [84]:
```
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 value
    for the corresponding total_acc value for that row.

    total_acc_avg here should be a Series or dictionary containing the mapping 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 [85]:
```
df['mort_acc'] = df.apply(lambda x: fill_mort_acc(x['total_acc'], x['mort_acc']), axis=1)
```

In [86]:
```
df.isnull().sum()
```

Out[86]:
```
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
```

TASK: 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 [87]:
```
# CODE HERE
```

In [88]:
```
df = df.dropna()
```

In [89]:
```
df.isnull().sum()
```

Out[89]:
```
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
```

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.

TASK: List all the columns that are currently non-numeric. Helpful Link

Another very useful method call

In [90]:
```
# CODE HERE
```

In [91]:
```
df.select_dtypes(['object']).columns
```

Out[91]:
```
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')
```

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

---

term feature

TASK: Convert the term feature into either a 36 or 60 integer numeric data type using .apply() or .map().

In [92]:
```
# CODE HERE
```

In [93]:
```
df['term'].value_counts()
```

Out[93]:
```
 36 months    301247
 60 months     93972
Name: term, dtype: int64
```

In [94]:
```
# Or just use .map()
df['term'] = df['term'].apply(lambda term: int(term[:3]))
```

grade feature

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

In [95]:
```
# CODE HERE
```

In [96]:
```
df = df.drop('grade',axis=1)
```

TASK: Convert the subgrade into dummy variables. Then concatenate these new columns to the original dataframe. Remember to drop the original subgrade column and to add drop_first=True to your get_dummies call.

In [97]:
```
# CODE HERE
```

In [98]:
```
subgrade_dummies = pd.get_dummies(df['sub_grade'],drop_first=True)
```

In [99]:
```
df = pd.concat([df.drop('sub_grade',axis=1),subgrade_dummies],axis=1)
```

In [100…
```
df.columns
```

Out[100…
```
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')
```

In [101…
```
df.select_dtypes(['object']).columns
```

Out[101…
```
Index(['home_ownership', 'verification_status', 'issue_d', 'loan_status',
       'purpose', 'earliest_cr_line', 'initial_list_status',
       'application_type', 'address'],
      dtype='object')
```

verification_status, application_type,initial_list_status,purpose

TASK: Convert these columns: ['verification_status', 'application_type','initial_list_status','purpose'] into dummy variables and concatenate them with the original dataframe. Remember to set drop_first=True and to drop the original columns.

In [102…
```
# CODE HERE
```

In [103…
```
dummies = pd.get_dummies(df[['verification_status', 'application_type','initial_list_status','purpose' ]],drop_
df = df.drop(['verification_status', 'application_type','initial_list_status','purpose'],axis=1)
df = pd.concat([df,dummies],axis=1)
```

In [ ]:

home_ownership

TASK:Review the value_counts for the home_ownership column.

In [104…
```
#CODE HERE
```

In [105…
```
df['home_ownership'].value_counts()
```

Out[105…
```
MORTGAGE    198022
RENT        159395
OWN          37660
OTHER          110
NONE            29
ANY              3
Name: home_ownership, dtype: int64
```

TASK: Convert these to dummy variables, but replace 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 [106…
```
#CODE HERE
```

In [107…
```
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)
```

address

TASK: 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 [108…
```
#CODE HERE
```

In [109…
```
df['zip_code'] = df['address'].apply(lambda address:address[-5:])
```

TASK: Now 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 [ ]:

In [110…
```
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)
```

issue_d

TASK: 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, drop this feature.

In [111…
```
#CODE HERE
```

In [112…
```
df = df.drop('issue_d',axis=1)
```

earliest_cr_line

TASK: This appears to be a historical time stamp feature. Extract the year from this feature using a .apply function, then convert it to a numeric feature. Set this new data to a feature column called 'earliest_cr_year'.Then drop the earliest_cr_line feature.

In [113…
```
#CODE HERE
```

In [114…
```
df['earliest_cr_year'] = df['earliest_cr_line'].apply(lambda date:int(date[-4:]))
df = df.drop('earliest_cr_line',axis=1)
```

In [115…
```
df.select_dtypes(['object']).columns
```

Out[115…
```
Index(['loan_status'], dtype='object')
```

Train Test Split

TASK: Import train_test_split from sklearn.

In [116…
```
from sklearn.model_selection import train_test_split
```

TASK: 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 [117…
```
# CODE HERE
```

In [118…
```
df = df.drop('loan_status',axis=1)
```

TASK: Set X and y variables to the .values of the features and label.

In [119…
```
#CODE HERE
```

In [120…
```
X = df.drop('loan_repaid',axis=1).values
y = df['loan_repaid'].values
```

OPTIONAL

Grabbing a Sample for Training Time

OPTIONAL: Use .sample() to grab a sample of the 490k+ entries to save time on training. Highly recommended for lower RAM computers or if you are not using GPU.

In [121…
```
# df = df.sample(frac=0.1,random_state=101)
print(len(df))
```

```
395219
```

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

In [122…
```
#CODE HERE
```

In [123…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=101)
```

Normalizing the Data

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

In [124…
```
# CODE HERE
```

In [125…
```
from sklearn.preprocessing import MinMaxScaler
```

In [126…
```
scaler = MinMaxScaler()
```

In [127…
```
X_train = scaler.fit_transform(X_train)
```

In [128…
```
X_test = scaler.transform(X_test)
```

Creating the Model

TASK: Run the cell below to import the necessary Keras functions.

In [129…
```
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
```

TASK: 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 1) 2

In [130…
```
# CODE HERE
model = Sequential()

# Choose whatever number of layers/neurons you want.

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

# Remember to compile()
```

In [131…
```
model = Sequential()

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


# 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')
```

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.

In [132…
```
# CODE HERE
```

In [133…
```
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 [==============================] - 4s 13us/sample - loss: 0.2959 - val_loss: 0.2652
Epoch 2/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2652 - val_loss: 0.2643
Epoch 3/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2628 - val_loss: 0.2626
Epoch 4/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2613 - val_loss: 0.2621
Epoch 5/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2609 - val_loss: 0.2621
Epoch 6/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2603 - val_loss: 0.2618
Epoch 7/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2600 - val_loss: 0.2616
Epoch 8/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2595 - val_loss: 0.2616
Epoch 9/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2593 - val_loss: 0.2620
Epoch 10/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2589 - val_loss: 0.2609
Epoch 11/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2588 - val_loss: 0.2613
Epoch 12/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2584 - val_loss: 0.2607
Epoch 13/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2581 - val_loss: 0.2613
Epoch 14/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2580 - val_loss: 0.2605
Epoch 15/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2580 - val_loss: 0.2607
Epoch 16/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2574 - val_loss: 0.2609
Epoch 17/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2575 - val_loss: 0.2606
Epoch 18/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2573 - val_loss: 0.2614
Epoch 19/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2572 - val_loss: 0.2611
Epoch 20/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2567 - val_loss: 0.2606
Epoch 21/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2569 - val_loss: 0.2606
Epoch 22/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2565 - val_loss: 0.2608
Epoch 23/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2564 - val_loss: 0.2612
Epoch 24/25
316175/316175 [==============================] - 3s 10us/sample - loss: 0.2561 - val_loss: 0.2609
Epoch 25/25
316175/316175 [==============================] - 3s 11us/sample - loss: 0.2560 - val_loss: 0.2612
```

Out[133…
```
<tensorflow.python.keras.callbacks.History at 0x20a2a8474c8>
```

TASK: OPTIONAL: Save your model.

In [134…
```
# CODE HERE
```

In [135…
```
from tensorflow.keras.models import load_model
```

In [136…
```
model.save('full_data_project_model.h5')
```

Section 3: Evaluating Model Performance.

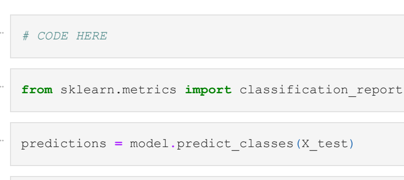
TASK: Plot out the validation loss versus the training loss.

In [137…
```
# CODE HERE
```

In [138…
```
losses = pd.DataFrame(model.history.history)
```

In [139…
```
losses[['loss','val_loss']].plot()
```

Out[139… <matplotlib.axes._subplots.AxesSubplot at 0x20a2cf62f48>



TASK: Create predictions from the X_test set and display a classification report and confusion matrix for the X_test set.

In [140…
```
# CODE HERE
```

In [141…
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [142…
```
predictions = model.predict_classes(X_test)
```

In [143…
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.44      0.61     15658
           1       0.88      1.00      0.93     63386

    accuracy                           0.89     79044
   macro avg       0.93      0.72      0.77     79044
weighted avg       0.90      0.89      0.87     79044
```

In [144…
```
confusion_matrix(y_test,predictions)
```

Out[144…
```
array([[ 6850,  8808],
       [  100, 63286]], dtype=int64)
```

TASK: Given the customer below, would you offer this person a loan?

In [145…
```
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[145…
```
loan_amnt           25000.00
term                    6.00
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
```

In [146…
```
# CODE HERE
```

In [147…
```
model.predict_classes(new_customer.values.reshape(1,78))
```

Out[147…
```
array([[1]])
```

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

In [149…
```
df.iloc[random_ind]['loan_repaid']
```

Out[149…
```
1.0
```