Keras API Project Exercise

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 [12]:
```
import pandas as pd
```

In [13]:
```
data_info = pd.read_csv('../DATA/lending_club_info.csv',index_col='LoanStatNew')
```

In [14]:
```
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 [15]:
```
def feat_info(col_name):
    print(data_info.loc[col_name]['Description'])
```

In [16]:
```
feat_info('mort_acc')
```

```
Number of mortgage accounts.
```

Loading the data and other imports

In [17]:
```
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 [18]:
```
df = pd.read_csv('../DATA/lending_club_loan_two.csv')
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 27 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  object 
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   grade                 396030 non-null  object 
 5   sub_grade             396030 non-null  object 
 6   emp_title             373103 non-null  object 
 7   emp_length            377729 non-null  object 
 8   home_ownership        396030 non-null  object 
 9   annual_inc            396030 non-null  float64
 10  verification_status   396030 non-null  object 
 11  issue_d               396030 non-null  object 
 12  loan_status           396030 non-null  object 
 13  purpose               396030 non-null  object 
 14  title                 394275 non-null  object 
 15  dti                   396030 non-null  float64
 16  earliest_cr_line      396030 non-null  object 
 17  open_acc              396030 non-null  float64
 18  pub_rec               396030 non-null  float64
 19  revol_bal             396030 non-null  float64
 20  revol_util            395754 non-null  float64
 21  total_acc             396030 non-null  float64
 22  initial_list_status   396030 non-null  object 
 23  application_type      396030 non-null  object 
 24  mort_acc              358235 non-null  float64
 25  pub_rec_bankruptcies  395495 non-null  float64
 26  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

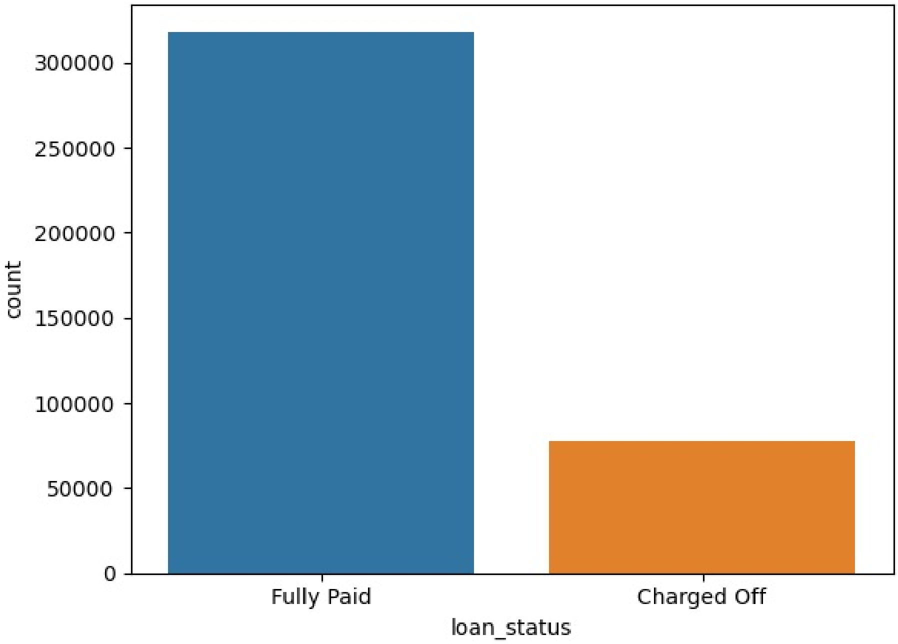
OVERALL GOAL: Get an understanding for which variables are important, view summary statistics, and visualize the data

---

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

In [20]:
```
sns.countplot(x='loan_status',data=df)
```

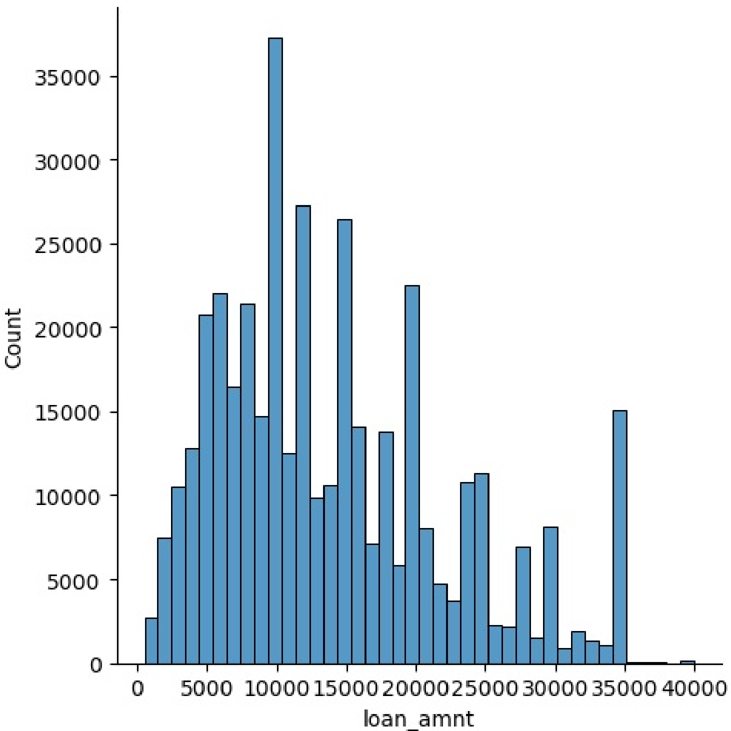
Out[20]:
```
<AxesSubplot:xlabel='loan_status', ylabel='count'>
```



TASK: Create a histogram of the loan_amnt column.

In [37]:
```
plt.figure(figsize=(12,4))
sns.displot(df['loan_amnt'],kde=False,bins=40)
```

Out[37]:
```
<seaborn.axisgrid.FacetGrid at 0x27c206a0730>
<Figure size 1200x400 with 0 Axes>
```



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

In [38]:
```
df.corr()
```

Out[38]:

	loan_amnt	int_rate	installment	annual_inc	dti	open_acc	pub_rec	revol_bal	revol_util	total_acc	mort_ac
loan_amnt	1.000000	0.168921	0.953929	0.336887	0.016636	0.198556	-0.077779	0.328320	0.099911	0.223886	0.2223
int_rate	0.168921	1.000000	0.162758	-0.056771	0.079038	0.011649	0.060986	-0.011280	0.293659	-0.036404	-0.0825
installment	0.953929	0.162758	1.000000	0.330381	0.015786	0.188973	-0.067892	0.316455	0.123915	0.202430	0.1936
annual_inc	0.336887	-0.056771	0.330381	1.000000	-0.081685	0.136150	-0.013720	0.299773	0.027871	0.193023	0.2363
dti	0.016636	0.079038	0.015786	-0.081685	1.000000	0.136181	-0.017639	0.063571	0.088375	0.102128	-0.0254
open_acc	0.198556	0.011649	0.188973	0.136150	0.136181	1.000000	-0.018392	0.221192	-0.131420	0.680728	0.1092
pub_rec	-0.077779	0.060986	-0.067892	-0.013720	-0.017639	-0.018392	1.000000	-0.101664	-0.075910	0.019723	0.0115
revol_bal	0.328320	-0.011280	0.316455	0.299773	0.063571	0.221192	-0.101664	1.000000	0.226346	0.191616	0.1949
revol_util	0.099911	0.293659	0.123915	0.027871	0.088375	-0.131420	-0.075910	0.226346	1.000000	-0.104273	0.0075
total_acc	0.223886	-0.036404	0.202430	0.193023	0.102128	0.680728	0.019723	0.191616	-0.104273	1.000000	0.3810
mort_acc	0.222315	-0.082583	0.193694	0.236320	-0.025439	0.109205	0.011552	0.194925	0.007514	0.381072	1.0000
pub_rec_bankruptcies	-0.106539	0.057450	-0.098628	-0.050162	-0.014558	-0.027732	0.699408	-0.124532	-0.086751	0.042035	0.0272

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 [41]:
```
plt.figure(figsize=(12,7))
sns.heatmap(df.corr(),annot=True,cmap='viridis')
```

Out[41]: <AxesSubplot:>

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 [42]:
```
feat_info('installment') #We want to check if they are not way too correlated to be duplicate information
```

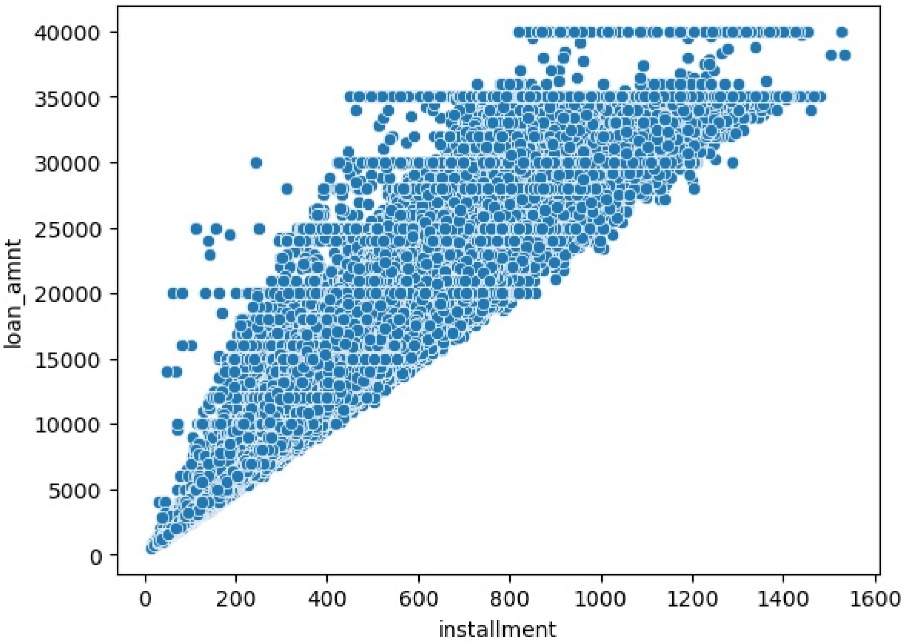
The monthly payment owed by the borrower if the loan originates.

In [19]:
```
feat_info('loan_amt') #
```

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.

In [43]:
```
sns.scatterplot(x='installment',y='loan_amnt',data=df)
```

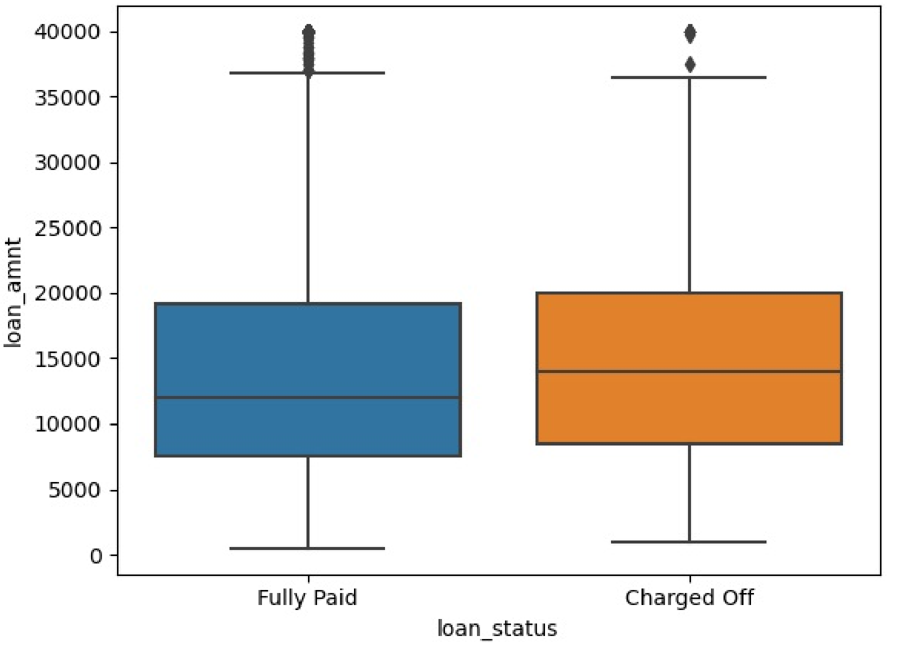
Out[43]: <AxesSubplot:xlabel='installment', ylabel='loan_amnt'>



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

In [45]:
```
sns.boxplot(x='loan_status',y='loan_amnt',data=df)
```

Out[45]: <AxesSubplot:xlabel='loan_status', ylabel='loan_amnt'>



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

In [47]:
```
df.groupby('loan_status')['loan_amnt'].describe()
```

Out[47]:

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 [48]:
```
df['grade'].unique()
```

Out[48]:
```
array(['B', 'A', 'C', 'E', 'D', 'F', 'G'], dtype=object)
```

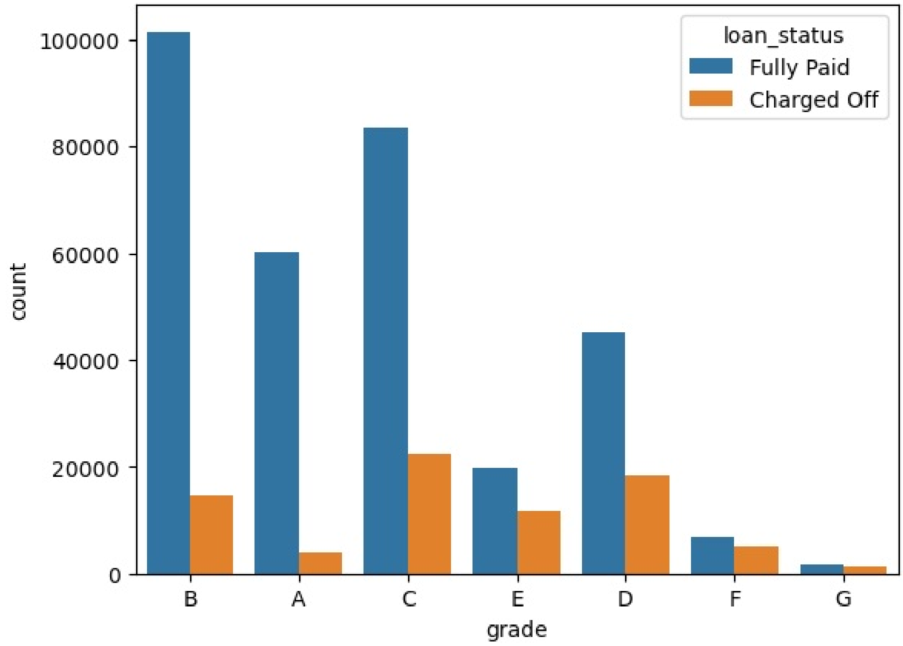
In [49]:
```
df['sub_grade'].unique()
```

Out[49]:
```
array(['B4', 'B5', 'B3', 'A2', 'C5', 'C3', 'A1', 'B2', 'C1', 'A5', 'E4',
       'A4', 'A3', 'D1', 'C2', 'B1', 'D3', 'D5', 'D2', 'E1', 'E2', 'E5',
       'F4', 'E3', 'D4', 'G1', 'F5', 'G2', 'C4', 'F1', 'F3', 'G5', 'G4',
       'F2', 'G3'], dtype=object)
```

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

In [50]:
```
sns.countplot(x='grade',data=df,hue='loan_status')
```

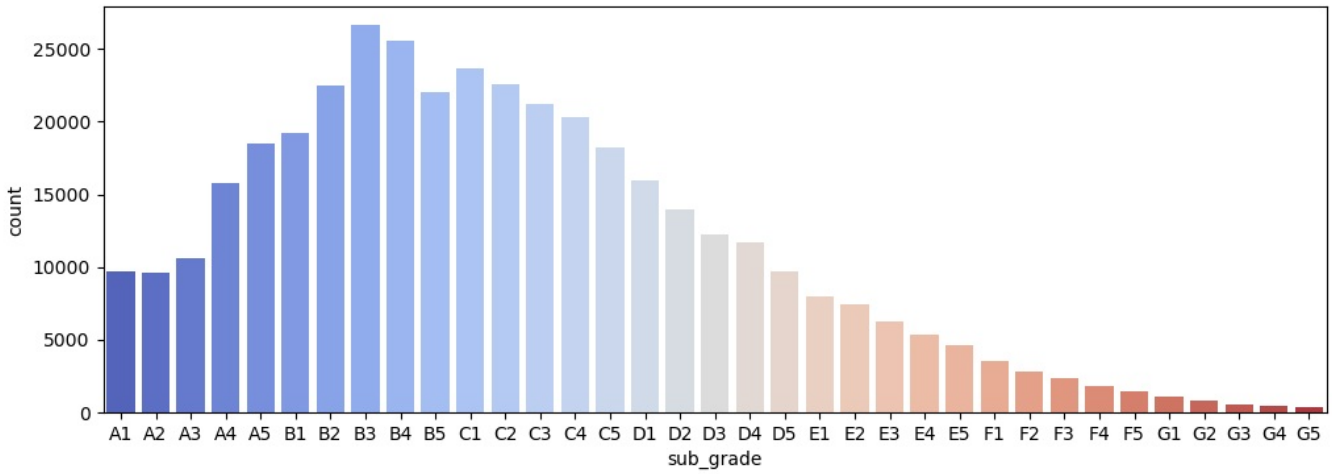
Out[50]:
```
<AxesSubplot:xlabel='grade', ylabel='count'>
```



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. After creating this plot, go ahead and create a similar plot, but set hue="loan_status"

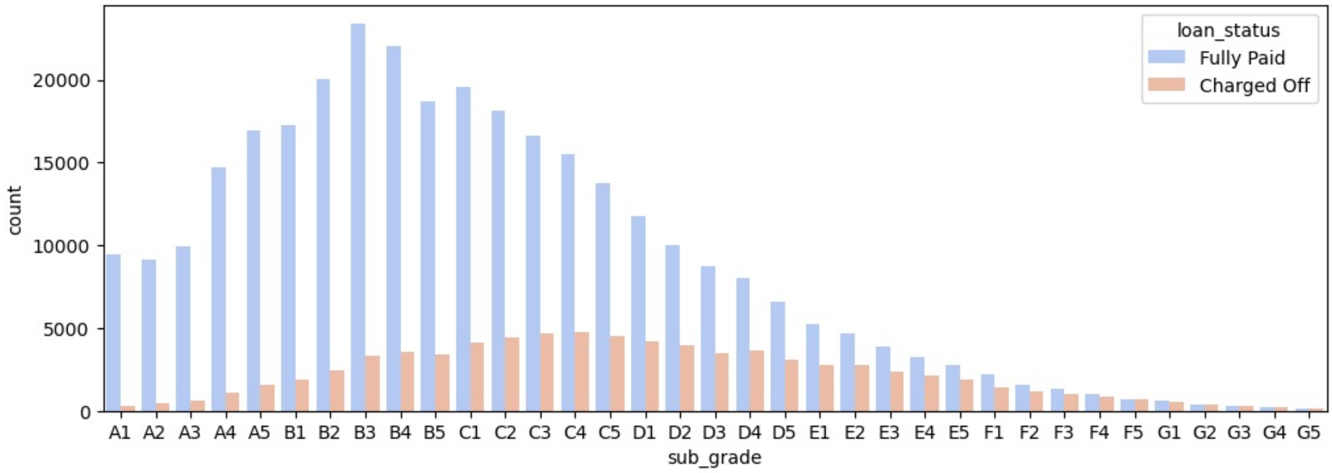
In [54]:
```
plt.figure(figsize=(12,4))
subgrade_order = sorted(df['sub_grade'].unique())
sns.countplot(x='sub_grade',data=df,order=subgrade_order,palette='coolwarm')
```

Out[54]:
```
<AxesSubplot:xlabel='sub_grade', ylabel='count'>
```



In [55]:
```
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[55]:
```
<AxesSubplot:xlabel='sub_grade', ylabel='count'>
```

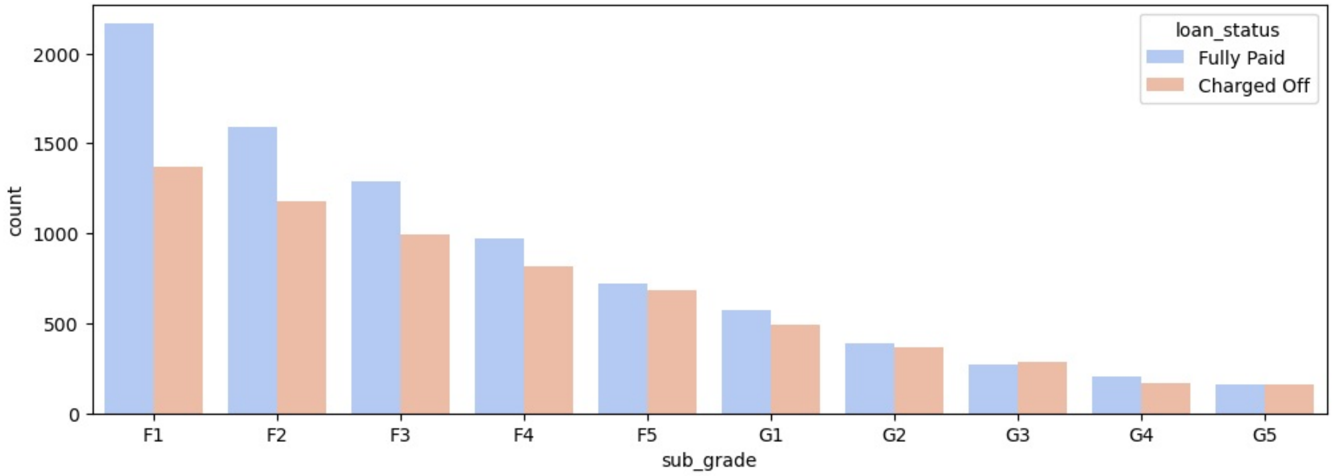


TASK: It looks like F and G subgrades don't get paid back that often. Isloate those and recreate the countplot just for those subgrades.

In [57]:
```
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=df,order=subgrade_order,palette='coolwarm',hue='loan_status')
```

Out[57]:
```
<AxesSubplot:xlabel='sub_grade', ylabel='count'>
```



TASK: Create 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".

In [58]:
```
df['loan_repaid'] = df['loan_status'].map({'Fully Paid':1,'Charged Off':0})
```

In [60]:
```
df[['loan_repaid','loan_status']]
```

Out[60]:

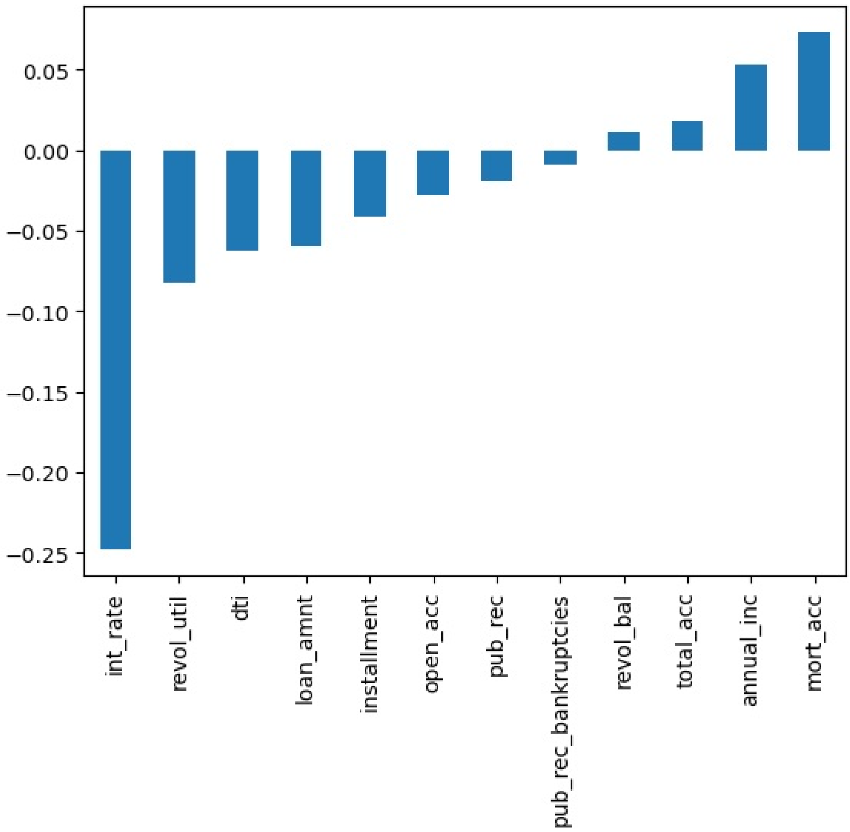
	loan_repaid	loan_status
0	1	Fully Paid
1	1	Fully Paid
2	1	Fully Paid
3	1	Fully Paid
4	0	Charged Off
...	...	...
396025	1	Fully Paid
396026	1	Fully Paid
396027	1	Fully Paid
396028	1	Fully Paid
396029	1	Fully Paid

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 [61]:
```
df.corr()['loan_repaid'].sort_values().drop('loan_repaid').plot(kind='bar')
```

Out[61]: <AxesSubplot:>



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 [62]:
```
df.head()
```

Out[62]:

	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 [63]:
```
len(df)
```

Out[63]:
```
396030
```

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

In [64]:
```
df.isnull().sum()
```

Out[64]:
```
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 [65]:
```
100* df.isnull().sum() / len(df)
```

Out[65]:
```
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 [66]:
```
feat_info('emp_title')
```

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

TASK: How many unique employment job titles are there?

In [67]:
```
df['emp_title'].nunique()
```

Out[67]:
```
173105
```

In [68]:
```
df['emp_title'].value_counts() #Just to check
```

Out[68]:
```
Teacher                    4389
Manager                    4250
Registered Nurse           1856
RN                         1846
Supervisor                 1830
                           ...
Postman                       1
McCarthy & Holthus, LLC       1
jp flooring                   1
Histology Technologist        1
Gracon Services, Inc          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 [69]:
```
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 [70]:
```
df['emp_length'].dropna().unique()
```

Out[70]:
```
array(['10+ years', '4 years', '< 1 year', '6 years', '9 years',
       '2 years', '3 years', '8 years', '7 years', '5 years', '1 year'],
      dtype=object)
```

In [71]:
```
sorted(df['emp_length'].dropna().unique())
```

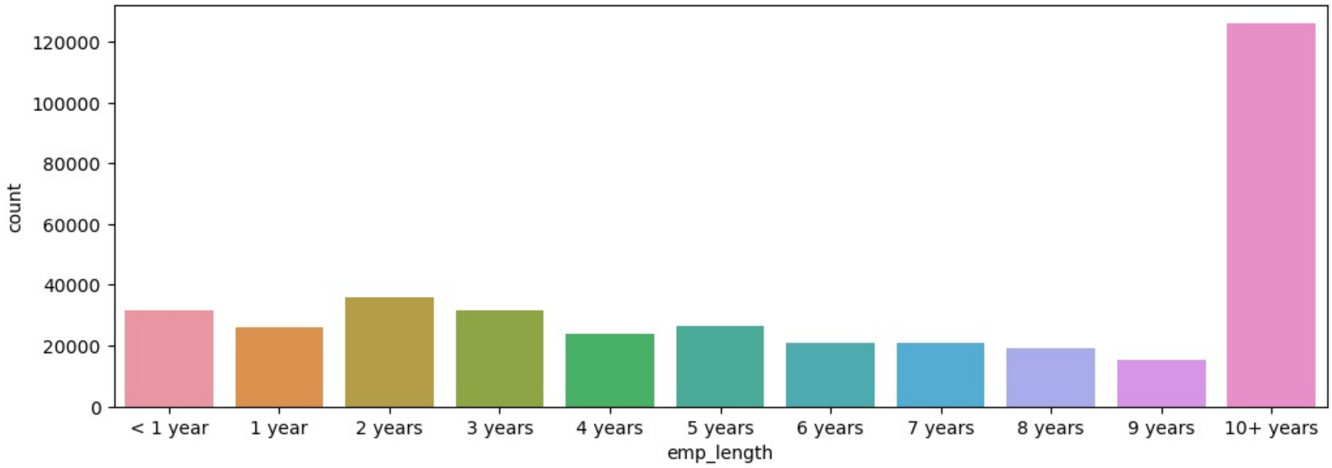
Out[71]:
```
['1 year',
 '10+ years',
 '2 years',
 '3 years',
 '4 years',
 '5 years',
 '6 years',
 '7 years',
 '8 years',
 '9 years',
 '< 1 year']
```

In [72]:
```
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 [73]:
```
plt.figure(figsize=(12,4))
sns.countplot(x='emp_length',data=df,order=emp_length_order)
```

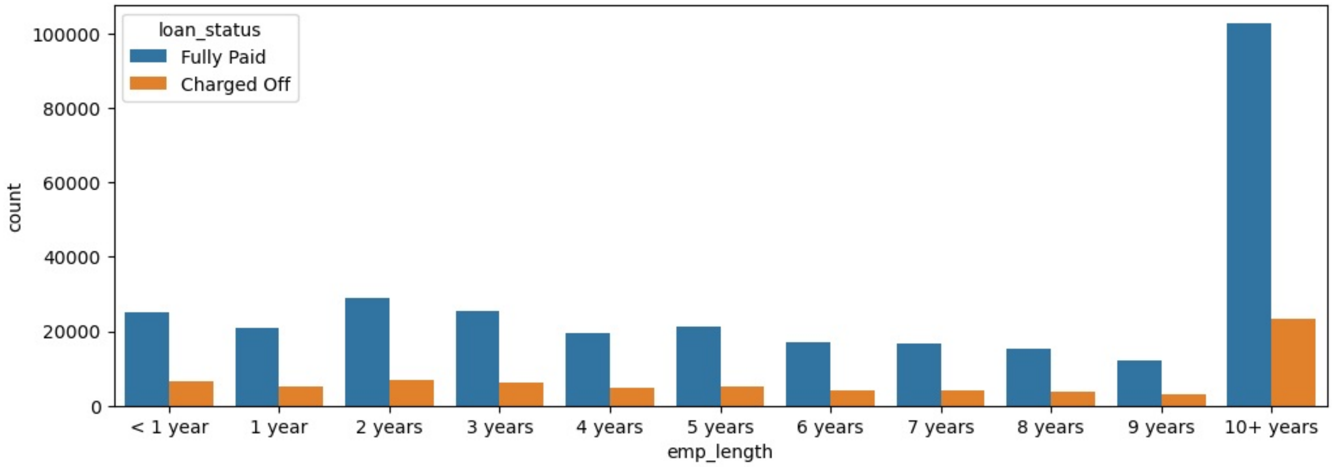
Out[73]: <AxesSubplot:xlabel='emp_length', ylabel='count'>



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

In [74]:
```
plt.figure(figsize=(12,4))
sns.countplot(x='emp_length',data=df,order=emp_length_order,hue='loan_status')
```

Out[74]: <AxesSubplot:xlabel='emp_length', ylabel='count'>



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 [82]:
```
emp_co = df[df['loan_status']=='Charged Off'].groupby('emp_length').count()['loan_status']
```

In [83]:
```
emp_fp = df[df['loan_status']=='Fully Paid'].groupby('emp_length').count()['loan_status']
```

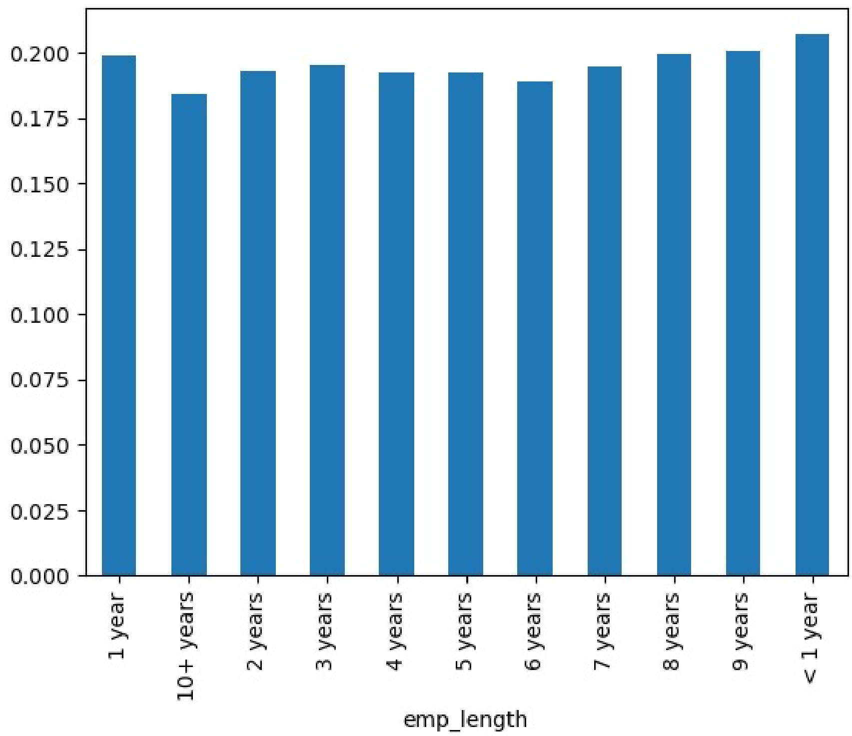
In [84]:
```
emp_co/emp_fp
```

Out[84]:
```
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 [85]:
```
emp_len = emp_co/(emp_co+emp_fp)
```

In [86]:
```
emp_len.plot(kind='bar')
```

Out[86]:
```
<AxesSubplot:xlabel='emp_length'>
```



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

In [87]:
```
df = df.drop('emp_length',axis=1)
```

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

In [88]:
```
df.isnull().sum()
```

Out[88]:
```
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 [90]:
```
feat_info('title')
```

```
The loan title provided by the borrower
```

In [93]:
```
df['title'].head(10)
```

Out[93]:
```
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
```

In [94]:
```
feat_info('purpose')
```

```
A category provided by the borrower for the loan request.
```

In [95]:
```
df['purpose'].head(10)
```

Out[95]:
```
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
```

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

In [96]:
```
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 [97]:
```
feat_info('mort_acc')
```

```
Number of mortgage accounts.
```

TASK: Create a value_counts of the mort_acc column.

In [98]:
```
df['mort_acc'].value_counts()
```

Out[98]:
```
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
32.0         2
31.0         2
23.0         2
26.0         2
28.0         1
30.0         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 [99]:
```
df.corr()['mort_acc'].sort_values()
```

Out[99]:
```
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 [103…
```
df.groupby('total_acc').mean()['mort_acc']
```

Out[103]:
```
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
```

In [104…
```
total_acc_avg = df.groupby('total_acc').mean()['mort_acc']
```

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 [105…

```python
def fill_mort_acc(total_acc,mort_acc):

    if np.isnan(mort_acc):
        return total_acc_avg[total_acc]
    else:
        return mort_acc
```

In [106…

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

Out[106]:

```
0         0.000000
1         3.000000
2         0.000000
3         0.000000
4         1.000000
            ...
396025    0.000000
396026    1.000000
396027    0.000000
396028    5.000000
396029    1.358013
Length: 396030, dtype: float64
```

In [107…

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

In [108…

```python
df.isnull().sum() #Just to check if the formula went through
```

Out[108]:

```
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 [109…

```python
df = df.dropna()
```

In [110…

```python
df.isnull().sum() #No more missing data
```

Out[110]:

```
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 [111…
```
df.select_dtypes(['object']).columns
```

Out[111]:
```
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 [112…
```
feat_info('term')
```

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

In [113…
```
df['term'].value_counts()
```

Out[113]:
```
 36 months    301247
 60 months     93972
Name: term, dtype: int64
```

In [114…
```
df['term'] = df['term'].apply(lambda term: int(term[:3]))
```

In [115…
```
df['term'].value_counts()
```

Out[115]:
```
36    301247
60     93972
Name: term, dtype: int64
```

grade feature

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

In [116…
```
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 [117…
```
dummies = pd.get_dummies(df['sub_grade'],drop_first=True)

df = pd.concat([df.drop('sub_grade',axis=1),dummies],axis=1)
```

In [118…
```
df.columns
```

Out[118]:
```
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')
```

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 [119…
```
dummies = pd.get_dummies(df[['verification_status', 'application_type','initial_list_status','purpose']],drop_f

df = pd.concat([df.drop(['verification_status', 'application_type','initial_list_status','purpose'],axis=1),dum
```

home_ownership

TASK:Review the value_counts for the home_ownership column.

In [120…
```
df['home_ownership'].value_counts()
```

Out[120]:
```
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 [121…
```
df['home_ownership'] = df['home_ownership'].replace(['NONE','ANY'],'OTHER')
```

In [122…
```
dummies = pd.get_dummies(df['home_ownership'],drop_first=True)

df = pd.concat([df.drop('home_ownership',axis=1),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 [123…
```
df['address']
```

Out[123]:
```
0              0174 Michelle Gateway\nMendozaberg, OK 22690
1           1076 Carney Fort Apt. 347\nLoganmouth, SD 05113
2           87025 Mark Dale Apt. 269\nNew Sabrina, WV 05113
3                     823 Reid Ford\nDelacruzside, MA 00813
4                      679 Luna Roads\nGreggshire, VA 11650
                                ...
396025       12951 Williams Crossing\nJohnnyville, DC 30723
396026    0114 Fowler Field Suite 028\nRachelborough, LA...
396027     953 Matthew Points Suite 414\nReedfort, NY 70466
396028    7843 Blake Freeway Apt. 229\nNew Michael, FL 2...
396029          787 Michelle Causeway\nBriannaton, AR 48052
Name: address, Length: 395219, dtype: object
```

In [124…
```
df['address'].apply(lambda address:address[-5:])
```

Out[124]:
```
0         22690
1         05113
2         05113
3         00813
4         11650
          ...
396025    30723
396026    05113
396027    70466
396028    29597
396029    48052
Name: address, Length: 395219, dtype: object
```

In [125…
```
df['zip_code'] = df['address'].apply(lambda address:address[-5:])
```

In [126…
```
df['zip_code'].value_counts()
```

Out[126]:
```
70466    56880
22690    56413
30723    56402
48052    55811
00813    45725
29597    45393
05113    45300
11650    11210
93700    11126
86630    10959
Name: zip_code, dtype: int64
```

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 [128…
```
dummies = pd.get_dummies(df['zip_code'],drop_first=True)

df = pd.concat([df.drop('zip_code',axis=1),dummies],axis=1)
```

In [129…
```
df = df.drop('address',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 [130…
```
feat_info('issue_d')
```

```
The month which the loan was funded
```

In [131…
```
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 [132…
```
feat_info('earliest_cr_line')
```

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

In [133…
```
df['earliest_cr_line']
```

Out[133]:
```
0         Jun-1990
1         Jul-2004
2         Aug-2007
3         Sep-2006
4         Mar-1999
            ...
396025    Nov-2004
396026    Feb-2006
396027    Mar-1997
396028    Nov-1990
396029    Sep-1998
Name: earliest_cr_line, Length: 395219, dtype: object
```

In [134…
```
df['earliest_cr_line'] = df['earliest_cr_line'].apply(lambda date: int(date[-4:]))
```

In [135…
```
df['earliest_cr_line']
```

Out[135]:
```
0         1990
1         2004
2         2007
3         2006
4         1999
          ...
396025    2004
396026    2006
396027    1997
396028    1990
396029    1998
Name: earliest_cr_line, Length: 395219, dtype: int64
```

Train Test Split

TASK: Import train_test_split from sklearn.

In [139…
```
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 [140…
```
df = df.drop('loan_status',axis=1)
```

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

In [141…
```
X = df.drop('loan_repaid',axis=1).values
```

In [142…
```
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 [143…
```
# df = df.sample(frac=0.1,random_state=101) if ever the laptop is slow
print(len(df))
```

```
395219
```

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

In [144…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, 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 [145…
```
from sklearn.preprocessing import MinMaxScaler
```

In [146…
```
scaler = MinMaxScaler()
```

In [148…
```
X_train = scaler.fit_transform(X_train)
```

In [149…
```
X_test = scaler.transform(X_test)
```

Creating the Model

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

In [150…
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout
```

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 [153…
```
# CODE HERE
model = Sequential()

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

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

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

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

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

In [154…
```
X_train.shape #It's good if your first layer matches the same number of features(neurons)
```

Out[154]:
```
(316175, 78)
```

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 [156…
```
model.fit(x=X_train,y=y_train,epochs=25,batch_size=256,
          validation_data=(X_test,y_test))
```

```
Epoch 1/25
1236/1236 [==============================] - 8s 5ms/step - loss: 0.3002 - val_loss: 0.2648
Epoch 2/25
1236/1236 [==============================] - 7s 5ms/step - loss: 0.2659 - val_loss: 0.2629
Epoch 3/25
1236/1236 [==============================] - 6s 4ms/step - loss: 0.2632 - val_loss: 0.2626
Epoch 4/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2618 - val_loss: 0.2628
Epoch 5/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2610 - val_loss: 0.2618
Epoch 6/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2603 - val_loss: 0.2621
Epoch 7/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2598 - val_loss: 0.2621
Epoch 8/25
1236/1236 [==============================] - 7s 6ms/step - loss: 0.2595 - val_loss: 0.2611
Epoch 9/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2593 - val_loss: 0.2611
Epoch 10/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2589 - val_loss: 0.2610
Epoch 11/25
1236/1236 [==============================] - 8s 6ms/step - loss: 0.2587 - val_loss: 0.2611
Epoch 12/25
1236/1236 [==============================] - 9s 7ms/step - loss: 0.2586 - val_loss: 0.2612
Epoch 13/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2581 - val_loss: 0.2611
Epoch 14/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2580 - val_loss: 0.2614
Epoch 15/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2578 - val_loss: 0.2612
Epoch 16/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2576 - val_loss: 0.2613
Epoch 17/25
1236/1236 [==============================] - 7s 6ms/step - loss: 0.2572 - val_loss: 0.2618
Epoch 18/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2572 - val_loss: 0.2614
Epoch 19/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2571 - val_loss: 0.2611
Epoch 20/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2568 - val_loss: 0.2612
Epoch 21/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2567 - val_loss: 0.2609
Epoch 22/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2565 - val_loss: 0.2614
Epoch 23/25
1236/1236 [==============================] - 6s 5ms/step - loss: 0.2562 - val_loss: 0.2609
Epoch 24/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2559 - val_loss: 0.2610
Epoch 25/25
1236/1236 [==============================] - 5s 4ms/step - loss: 0.2560 - val_loss: 0.2611
```

Out[156]:

```
<keras.callbacks.History at 0x27c3bbc9b50>
```

TASK: OPTIONAL: Save your model.

In [157…

```
from tensorflow.keras.models import load_model
```

In [158…

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

Section 3: Evaluating Model Performance.

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

In [159…

```
model.history.history
```

Out[159]:

```
{'loss': [0.3002484142780304,
  0.2659139633178711,
  0.2632180452346802,
  0.2618030607700348,
  0.26101648807525635,
  0.2602875232696533,
  0.2597786486148834,
  0.2594921886920929,
  0.25930967926979065,
  0.2588726282119751,
  0.25868892669677734,
  0.25856730341911316,
  0.25814399123191833,
  0.25802865624427795,
  0.2577919065952301,
  0.2575809955596924,
  0.25722038745880127,
  0.25720295310020447,
  0.2571004629135132,
  0.25680455565452576,
  0.2566736042499542,
  0.25647303462028503,
  0.2561701536178589,
  0.25592780113220215,
  0.256005197763443],
 'val_loss': [0.26480749249458313,
  0.26292890310287476,
  0.2626335620880127,
  0.26279768347740173,
  0.2617569863796234,
  0.26207107305526733,
  0.2620513141155243,
  0.261094868183136,
  0.2611057758331299,
  0.2609754800796509,
  0.2610848546028137,
  0.26122891902923584,
  0.2610914409160614,
  0.26135972142219543,
  0.2611832618713379,
  0.261321097612381,
  0.26175546646118164,
  0.2614286243915558,
  0.2610941231250763,
  0.2612297534942627,
  0.26094672083854675,
  0.2614441215991974,
  0.26086318492889404,
  0.26100075244903564,
  0.261123389005661]}
```

In [160…

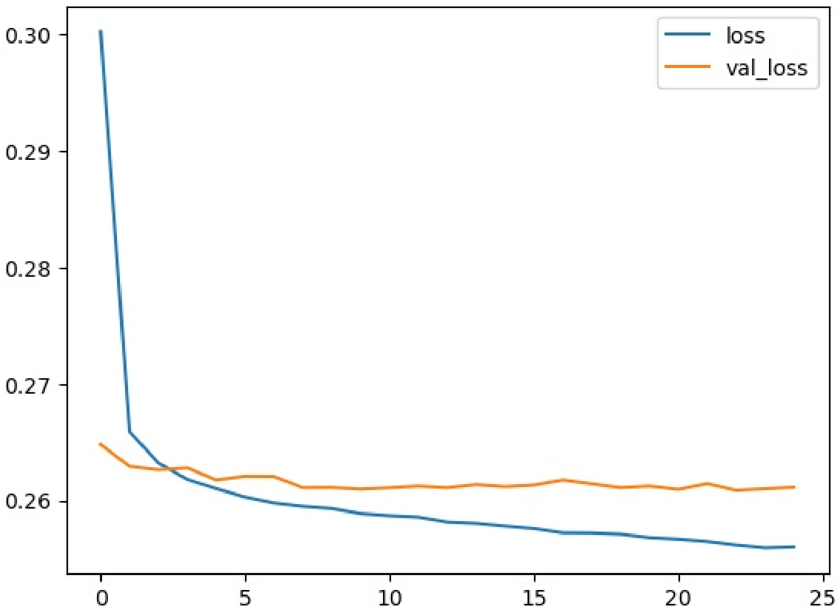
```
losses = pd.DataFrame(model.history.history)
```

In [161…

```
losses.plot()
```

Out[161]:

```
<AxesSubplot:>
```



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

In [162…

```
from sklearn.metrics import classification_report,confusion_matrix
```

In [164…
```
predictions = (model.predict(X_test) > 0.5)*1
```
```
2471/2471 [==============================] - 5s 2ms/step
```

In [165…
```
print(classification_report(y_test,predictions))
```
```
              precision    recall  f1-score   support

           0       0.98      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 [166…
```
print(confusion_matrix(y_test,predictions))
```
```
[[ 6848  8810]
 [  119 63267]]
```

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

In [167…
```
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[167]:
```
loan_amnt      25000.00
term              60.00
int_rate          18.24
installment      638.11
annual_inc     61665.00
                 ...
30723              1.00
48052              0.00
70466              0.00
86630              0.00
93700              0.00
Name: 305323, Length: 78, dtype: float64
```

In [170…
```
new_customer = scaler.transform(new_customer.values.reshape(1,78))
```

In [175…
```
model.predict(new_customer)
```
```
1/1 [==============================] - 0s 105ms/step
```
Out[175]:
```
array([[0.6004283]], dtype=float32)
```

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

In [176…
```
df.iloc[random_ind]['loan_repaid']
```
Out[176]:
```
1.0
```

GREAT JOB!

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js