2110446 Data Science and Data Engineering

Preparing and Cleaning Data for Machine Learning

First, lets import some of the libraries that we'll be using, and set some parameters to make the output easier to read.

1. Examining the Data Set

Lending Club periodically releases data for all the approved and declined loan applications on their website. So you're working with the same data we are, we've mirrored the data on data.world. You can select different year ranges to download the dataset (in CSV format) for both approved and declined loans.

You'll also find a data dictionary (in XLS format), towards the bottom of the page, which contains information on the different column names. The data dictionary is useful to help understand what a column represents in the dataset.

The data dictionary contains two sheets:

LoanStats sheet: describes the approved loans dataset RejectStats sheet: describes the rejected loans dataset We'll be using the LoanStats sheet since we're interested in the approved loans dataset.

The approved loans dataset contains information on current loans, completed loans, and defaulted loans. For this challenge, we'll be working with approved loans data for the years 2007 to 2011.

First, lets import some of the libraries that we'll be using, and set some parameters to make the output easier to read.

In [1]:
```
import pandas as pd
import numpy as np
#pd.set_option('max_columns', 120)
#pd.set_option('max_colwidth', 5000)

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.rcParams['figure.figsize'] = (12,8)
```

Loading The Data Into Pandas

We've downloaded our dataset and named it lending_club_loans.csv, but now we need to load it into a pandas DataFrame to explore it.

To ensure that code run fast for us, we need to reduce the size of lending_club_loans.csv by doing the following:

Remove the first line: It contains extraneous text instead of the column titles. This text prevents the dataset from being parsed properly by the pandas library. Remove the 'desc' column: it contains a long text explanation for the loan. Remove the 'url' column: it contains a link to each on Lending Club which can only be accessed with an investor account. Removing all columns with more than 50% missing values: This allows us to move faster since don't need to spend time trying to fill these values. We'll also name the filtered dataset loans_2007 and later at the end of this section save it as loans_2007.csv to keep it separate from the raw data. This is good practice and makes sure we have our original data in case we need to go back and retrieve any of the original data we're removing.

Now, let's go ahead and perform these steps:

In [2]:
```
# skip row 1 so pandas can parse the data properly.
loans_2007 = pd.read_csv('dataset/lending_club_loans.csv', low_memory=F
print (loans_2007.shape)
half_count = len(loans_2007) / 2
loans_2007 = loans_2007.dropna(thresh=half_count,axis=1) # Drop any co
print (loans_2007.shape)
loans_2007 = loans_2007.drop(['url','desc'],axis=1)      # These colum
print (loans_2007.shape)
```

```
(42538, 115)
(42538, 58)
(42538, 56)
```

Let's use the pandas head() method

to display first three rows of the loans_2007 DataFrame, just to make sure we were able to load the dataset properly:

In [3]:
```
loans_2007.head(3)
```

Out[3]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installm
0	1077501	1296599.0	5000.0	5000.0	4975.0	36 months	0.1065	162
1	1077430	1314167.0	2500.0	2500.0	2500.0	60 months	0.1527	59
2	1077175	1313524.0	2400.0	2400.0	2400.0	36 months	0.1596	84

3 rows × 56 columns

In [4]:
```
loans_2007.describe()
```

Out[4]:

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	int_rate	installme
count	4.253500e+04	42535.000000	42535.000000	42535.000000	42535.000000	42535.0000
mean	8.257026e+05	11089.722581	10821.585753	10139.830603	0.121650	322.6230
std	2.795409e+05	7410.938391	7146.914675	7131.686447	0.037079	208.9272
min	7.047300e+04	500.000000	500.000000	0.000000	0.054200	15.6700
25%	6.384795e+05	5200.000000	5000.000000	4950.000000	0.096300	165.5200
50%	8.241780e+05	9700.000000	9600.000000	8500.000000	0.119900	277.6900
75%	1.033946e+06	15000.000000	15000.000000	14000.000000	0.147200	428.1800
max	1.314167e+06	35000.000000	35000.000000	35000.000000	0.245900	1305.1900

8 rows × 31 columns

Let's also use pandas .shape attribute

to view the number of samples and features we're dealing with at this stage:

In [5]:
```
loans_2007.shape
```

Out[5]: (42538, 56)

2. Narrowing down our columns

It's a great idea to spend some time to familiarize ourselves with the columns in the dataset, to understand what each feature represents. This is important, because a poor understanding of the features could cause us to make mistakes in the data analysis and the modeling process.

We'll be using the data dictionary Lending Club provided to help us become familiar with the columns and what each represents in the dataset. To make the process easier, we'll create a DataFrame to contain the names of the columns, data type, first row's values, and description from the data dictionary.

To make this easier, we've pre-converted the data dictionary from Excel format to a CSV.

In [6]:
```
data_dictionary = pd.read_csv('dataset/LCDataDictionary.csv') # Loadin
print(data_dictionary.shape[0])
print(data_dictionary.columns.tolist())
```

```
117
['LoanStatNew', 'Description']
```

In [7]:
```
data_dictionary.head()
data_dictionary = data_dictionary.rename(columns={'LoanStatNew': 'name
```

Now that we've got the data dictionary loaded.

Let's join the first row of loans_2007 to the data_dictionary DataFrame to give us a preview DataFrame with the following columns:

name — contains the column names of loans_2007. dtypes — contains the data types of the loans_2007 columns. first value — contains the values of loans_2007 first row. description — explains what each column in loans_2007

In [8]:
```
loans_2007_dtypes = pd.DataFrame(loans_2007.dtypes,columns=['dtypes'])
loans_2007_dtypes = loans_2007_dtypes.reset_index()
loans_2007_dtypes['name'] = loans_2007_dtypes['index']
#print (loans_2007_dtypes)
loans_2007_dtypes = loans_2007_dtypes[['name','dtypes']]

loans_2007_dtypes['first value'] = loans_2007.loc[0].values
preview = loans_2007_dtypes.merge(data_dictionary, on='name',how='left
```

In [9]:
```
preview.head()
```

Out[9]:

	name	dtypes	first value	description
0	id	object	1077501	A unique LC assigned ID for the loan listing.
1	member_id	float64	1.2966e+06	A unique LC assigned Id for the borrower member.
2	loan_amnt	float64	5000	The listed amount of the loan applied for by t...
3	funded_amnt	float64	5000	The total amount committed to that loan at tha...
4	funded_amnt_inv	float64	4975	The total amount committed by investors for th...

When we printed the shape of loans_2007 earlier, we noticed that it had 56 columns which also means this preview DataFrame has 56 rows. It can be cumbersome to try to explore all the rows of preview at once, so instead we'll break it up into three parts and look at smaller selection of features each time.

As you explore the features to better understand each of them, you'll want to pay attention to any column that:

leaks information from the future (after the loan has already been funded), don't affect the borrower's ability to pay back the loan (e.g. a randomly generated ID value by Lending Club), is formatted poorly, requires more data or a lot of preprocessing to turn into useful a feature, or contains redundant information. I'll say it again to emphasize it because it's important: We need to especially pay close attention to data leakage, which can cause the model to overfit. This is because the model would be also learning from features that wouldn't be available when we're using it make predictions on future loans.

First Group Of Columns Let's display the first 19 rows of preview and analyze them:

In [10]:
```
preview[:19]
```

Out[10]:

	name	dtypes	first value	description
0	id	object	1077501	A unique LC assigned ID for the loan listing.
1	member_id	float64	1.2966e+06	A unique LC assigned Id for the borrower member.
2	loan_amnt	float64	5000	The listed amount of the loan applied for by t...
3	funded_amnt	float64	5000	The total amount committed to that loan at tha...
4	funded_amnt_inv	float64	4975	The total amount committed by investors for th...
5	term	object	36 months	The number of payments on the loan. Values are...
6	int_rate	float64	0.1065	Interest Rate on the loan
7	installment	float64	162.87	The monthly payment owed by the borrower if th...
8	grade	object	B	LC assigned loan grade
9	sub_grade	object	B2	LC assigned loan subgrade
10	emp_title	object	NaN	The job title supplied by the Borrower when ap...
11	emp_length	object	10+ years	Employment length in years. Possible values ar...
12	home_ownership	object	RENT	The home ownership status provided by the borr...
13	annual_inc	float64	24000	The self-reported annual income provided by th...
14	verification_status	object	Verified	Indicates if income was verified by LC, not ve...
15	issue_d	object	Dec-2011	The month which the loan was funded
16	loan_status	object	Fully Paid	Current status of the loan
17	pymnt_plan	object	False	Indicates if a payment plan has been put in pl...
18	purpose	object	credit_card	A category provided by the borrower for the lo...

After analyzing the columns, we can conclude that the following features can be removed:

id — randomly generated field by Lending Club for unique identification purposes only. member_id — also randomly generated field by Lending Club for identification purposes only. funded_amnt — leaks information from the future(after the loan is already started to be funded). funded_amnt_inv — also leaks data from the future. sub_grade — contains redundant information that is already in the grade column (more below). int_rate — also included within the grade column. emp_title — requires other data and a lot of processing to become potentially useful issued_d — leaks data from the future. Lending Club uses a borrower's grade and payment term (30 or months) to assign an interest rate (you can read more about Rates & Fees). This causes variations in interest rate within a given grade. But, what may be useful for our model is to focus on clusters of borrowers instead of individuals. And, that's exactly what grading does - it segments borrowers based on their credit score and other behaviors, which is we should keep the grade column and drop interest int_rate and sub_grade.

Let's drop these columns from the DataFrame before moving onto to the next group of columns.

In [11]:
```
drop_list = ['id','member_id','funded_amnt','funded_amnt_inv',
             'int_rate','sub_grade','emp_title','issue_d']
loans_2007 = loans_2007.drop(drop_list,axis=1)
```

Second Group Of Columns Let's move on to the next 19 columns:

In [12]:
```
preview[19:38]
```

Out[12]:

	name	dtypes	first value	description
19	title	object	Computer	The loan title provided by the borrower
20	zip_code	object	860xx	The first 3 numbers of the zip code provided b...
21	addr_state	object	AZ	The state provided by the borrower in the loan...
22	dti	float64	27.65	A ratio calculated using the borrower’s total ...
23	delinq_2yrs	float64	0	The number of 30+ days past-due incidences of ...
24	earliest_cr_line	object	Jan-1985	The month the borrower's earliest reported cre...
25	fico_range_low	float64	735	The lower boundary range the borrower’s FICO a...
26	fico_range_high	float64	739	The upper boundary range the borrower’s FICO a...
27	inq_last_6mths	float64	1	The number of inquiries in past 6 months (excl...
28	open_acc	float64	3	The number of open credit lines in the borrowe...
29	pub_rec	float64	0	Number of derogatory public records
30	revol_bal	float64	13648	Total credit revolving balance
31	revol_util	float64	0.837	Revolving line utilization rate, or the amount...
32	total_acc	float64	9	The total number of credit lines currently in ...
33	initial_list_status	object	False	The initial listing status of the loan. Possib...
34	out_prncp	float64	0	Remaining outstanding principal for total amou...
35	out_prncp_inv	float64	0	Remaining outstanding principal for portion of...
36	total_pymnt	float64	5863.16	Payments received to date for total amount funded
37	total_pymnt_inv	float64	5833.84	Payments received to date for portion of total...

In this group,take note of the fico_range_low and fico_range_high columns. Both are in this second group of columns but because they related to some other columns, we'll talk more about them after looking at the last group of columns.

We can drop the following columns:

zip_code - mostly redundant with the addr_state column since only the first 3 digits of the 5 digit zip code are visible. out_prncp - leaks data from the future. out_prncp_inv - also leaks data from the future. total_pymnt - also leaks data from the future. total_pymnt_inv - also leaks data from the future. Let's go ahead and remove these 5 columns from the DataFrame:

In [13]:
```
drop_cols = [ 'zip_code','out_prncp','out_prncp_inv',
              'total_pymnt','total_pymnt_inv']
loans_2007 = loans_2007.drop(drop_cols, axis=1)
```

Third Group Of Columns Let's analyze the last group of features:

In [14]:
```
preview[38:]
```

Out[14]:

	name	dtypes	first value	description
38	total_rec_prncp	float64	5000	Principal received to date
39	total_rec_int	float64	863.16	Interest received to date
40	total_rec_late_fee	float64	0	Late fees received to date
41	recoveries	float64	0	post charge off gross recovery
42	collection_recovery_fee	float64	0	post charge off collection fee
43	last_pymnt_d	object	Jan-2015	Last month payment was received
44	last_pymnt_amnt	float64	171.62	Last total payment amount received
45	last_credit_pull_d	object	Sep-2016	The most recent month LC pulled credit for thi...
46	last_fico_range_high	float64	744	The upper boundary range the borrower’s last F...
47	last_fico_range_low	float64	740	The lower boundary range the borrower’s last F...
48	collections_12_mths_ex_med	object	False	Number of collections in 12 months excluding m...
49	policy_code	object	True	publicly available policy_code=1\nnew products...
50	application_type	object	INDIVIDUAL	Indicates whether the loan is an individual ap...
51	acc_now_delinq	object	False	The number of accounts on which the borrower i...
52	chargeoff_within_12_mths	object	False	Number of charge-offs within 12 months
53	delinq_amnt	float64	0	The past-due amount owed for the accounts on w...
54	pub_rec_bankruptcies	float64	0	Number of public record bankruptcies
55	tax_liens	object	False	Number of tax liens

In this last group of columns, we need to drop the following, all of which leak data from the future:

total_rec_prncp total_rec_int total_rec_late_fee recoveries collection_recovery_fee last_pymnt_d last_pymnt_amnt Let's drop our last group of columns:

In [15]:
```
drop_cols = ['total_rec_prncp','total_rec_int', 'total_rec_late_fee',
             'recoveries', 'collection_recovery_fee', 'last_pymnt_d',
             'last_pymnt_amnt']

loans_2007 = loans_2007.drop(drop_cols, axis=1)
print (loans_2007.shape)
```

```
(42538, 36)
```

Investigating FICO Score Columns

Now, besides the explanations provided here in the Description column,let's learn more about fico_range_low, fico_range_high, last_fico_range_low, and last_fico_range_high.

FICO scores are a credit score, or a number used by banks and credit cards to represent how credit-worthy a person is. While there are a few types of credit scores used in the United States, the FICO score is the best known and most widely used.

When a borrower applies for a loan, Lending Club gets the borrowers credit score from FICO - they are given a lower and upper limit of the range that the borrowers score belongs to, and they store those values as fico_range_low, fico_range_high. After that, any updates to the borrowers score are recorded as last_fico_range_low, and last_fico_range_high.

A key part of any data science project is to do everything you can to understand the data. While researching this data set, I found a project done in 2014 by a group of students from Stanford University on this same dataset.

In the report for the project, the group listed the current credit score (last_fico_range) among late fees and recovery fees as fields they mistakenly added to the features but state that they later learned these columns all leak information into the future.

However, following this group's project, another group from Stanford worked on this same Lending Club dataset. They used the FICO score columns, dropping only last_fico_range_low, in their modeling. This second group's report described last_fico_range_high as the one of the more important features in predicting accurate results.

The question we must answer is, do the FICO credit scores information into the future? Recall a column is considered leaking information when especially it won't be available at the time we use our model - in this case when we use our model on future loans.

This blog examines in-depth the FICO scores for lending club loans, and notes that while looking at the trend of the FICO scores is a great predictor of whether a loan will default, that because FICO scores continue to be updated by the Lending Club after a loan is funded, a defaulting loan can lower the borrowers score, or in other words, will leak data.

Therefore we can safely use fico_range_low and fico_range_high, but not last_fico_range_low, and last_fico_range_high. Lets take a look at the values in these columns:

In [16]:
```
print(loans_2007['fico_range_low'].unique())
print(loans_2007['fico_range_high'].unique())
```

```
[735. 740. 690. 695. 730. 660. 675. 725. 710. 705. 720. 665. 670. 760.
 685. 755. 680. 700. 790. 750. 715. 765. 745. 770. 780. 775. 795. 810.
 800. 815. 785. 805. 825. 820. 630. 625.  nan 650. 655. 645. 640. 635.
 610. 620. 615.]
[739. 744. 694. 699. 734. 664. 679. 729. 714. 709. 724. 669. 674. 764.
 689. 759. 684. 704. 794. 754. 719. 769. 749. 774. 784. 779. 799. 814.
 804. 819. 789. 809. 829. 824. 634. 629.  nan 654. 659. 649. 644. 639.
 614. 624. 619.]
```

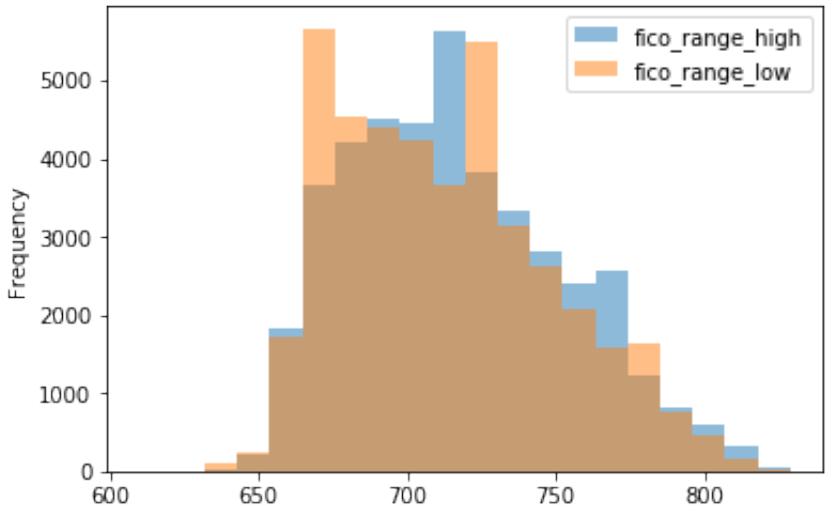
Let's get rid of the missing values, then plot histograms to look at the ranges of the two columns:

In [17]:
```
fico_columns = ['fico_range_high','fico_range_low']

print(loans_2007.shape[0])
loans_2007.dropna(subset=fico_columns,inplace=True)
print(loans_2007.shape[0])

loans_2007[fico_columns].plot.hist(alpha=0.5,bins=20);
```

```
42538
42535
```



Let's now go ahead and create a column for the average of fico_range_low and fico_range_high columns and name it fico_average.

Note that this is not the average FICO score for each borrower, but rather an average of the high and low range that we know the borrower is in.

In [18]:
```
loans_2007['fico_average'] = (loans_2007['fico_range_high'] + loans_20
```

In [19]:
```
# Let's check what we just did.
```

In [20]:
```
cols = ['fico_range_low','fico_range_high']
loans_2007[cols].head()
```

Out[20]:

	fico_range_low	fico_range_high
0	735.0	739.0
1	740.0	744.0
2	735.0	739.0
3	690.0	694.0
4	695.0	699.0

Good! We got the mean calculations and everything right.

Now, we can go ahead and drop fico_range_low, fico_range_high, last_fico_range_low, and last_fico_range_high columns.

In [21]:
```
drop_cols = ['fico_range_low','fico_range_high','last_fico_range_low',
             'last_fico_range_high']
loans_2007 = loans_2007.drop(drop_cols, axis=1)
loans_2007.shape
```

Out[21]:
```
(42535, 33)
```

Notice just by becoming familiar with the columns in the dataset, we're able to reduce the number of columns from 56 to 33.

Decide On A Target Column

Now, let's decide on the appropriate column to use as a target column for modeling - keep in mind the main goal is predict who will pay off a loan and who will default.

We learned from the description of columns in the preview DataFrame that loan_status is the only field in the main dataset that describe a loan status, so let's use this column as the target column.

In [22]:
```
preview[preview.name == 'loan_status']
```

Out[22]:

	name	dtypes	first value	description
16	loan_status	object	Fully Paid	Current status of the loan

Currently, this column contains text values that need to be converted to numerical values to be able use for training a model.

Let's explore the different values in this column and come up with a strategy for converting the values in this column. We'll use the DataFrame method value_counts() to return the frequency of the unique values in the loan_status column.

In [23]:
```
loans_2007["loan_status"].value_counts()
```

Out[23]:
```
Fully Paid                                             33586
Charged Off                                             5653
Does not meet the credit policy. Status:Fully Paid      1988
Does not meet the credit policy. Status:Charged Off      761
Current                                                  513
In Grace Period                                           16
Late (31-120 days)                                        12
Late (16-30 days)                                          5
Default                                                    1
Name: loan_status, dtype: int64
```

The loan status has nine different possible values!

Let's learn about these unique values to determine the ones that best describe the final outcome of a loan, and also the kind of classification problem we'll be dealing with.

You can read about most of the different loan statuses on the Lending Club website as well as these posts on the Lend Academy and Orchard forums. I have pulled that data together in a table below so we can see the unique values, their frequency in the dataset and what each means:

In [24]:
```
eaning = [
    "Loan has been fully paid off.",
    "Loan for which there is no longer a reasonable expectation of furth
    "While the loan was paid off, the loan application today would no lo
    "While the loan was charged off, the loan application today would no
    "Loan is up to date on current payments.",
    "The loan is past due but still in the grace period of 15 days.",
    "Loan hasn't been paid in 31 to 120 days (late on the current paymen
    "Loan hasn't been paid in 16 to 30 days (late on the current payment
    "Loan is defaulted on and no payment has been made for more than 121

tatus, count = loans_2007["loan_status"].value_counts().index, loans_20

oan_statuses_explanation = pd.DataFrame({'Loan Status': status,'Count':
oan_statuses_explanation
```

Out[24]:

	Loan Status	Count	Meaning
0	Fully Paid	33586	Loan has been fully paid off.
1	Charged Off	5653	Loan for which there is no longer a reasonable...
2	Does not meet the credit policy. Status:Fully ...	1988	While the loan was paid off, the loan applicat...
3	Does not meet the credit policy. Status:Charge...	761	While the loan was charged off, the loan appli...
4	Current	513	Loan is up to date on current payments.
5	In Grace Period	16	The loan is past due but still in the grace pe...
6	Late (31-120 days)	12	Loan hasn't been paid in 31 to 120 days (late ...
7	Late (16-30 days)	5	Loan hasn't been paid in 16 to 30 days (late o...
8	Default	1	Loan is defaulted on and no payment has been m...

Remember, our goal is to build a machine learning model that can learn from past loans in trying to predict which loans will be paid off and which won't. From the above table, only the Fully Paid and Charged Off values describe the final outcome of a loan. The other values describe loans that are still on going, and even though some loans are late on payments, we can't jump the gun and classify them as Charged Off.

Also, while the Default status resembles the Charged Off status, in Lending Club's eyes, loans that are charged off have essentially no chance of being repaid while default ones have a small chance. Therefore, we should use only samples where the loan_status column is 'Fully Paid' or 'Charged Off'.

We're not interested in any statuses that indicate that the loan is ongoing or in progress, because predicting that something is in progress doesn't tell us anything.

Since we're interested in being able to predict which of these 2 values a loan will fall under, we can treat the problem as binary classification.

Let's remove all the loans that don't contain either 'Fully Paid' or 'Charged Off' as the loan's status and then transform the 'Fully Paid' values to 1 for the positive case and the 'Charged Off' values to 0 for the negative case.

This will mean that out of the ~42,000 rows we have, we'll be removing just over 3,000.

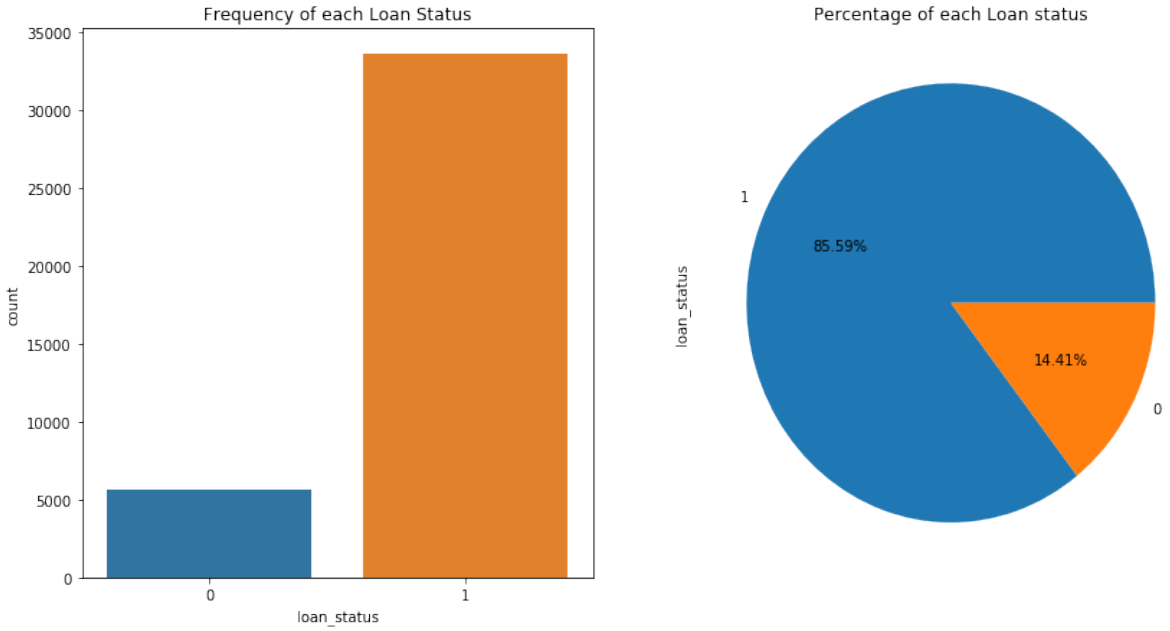
There are few different ways to transform all of the values in a column, we'll use the DataFrame method replace().

In [25]:
```
loans_2007 = loans_2007[(loans_2007["loan_status"] == "Fully Paid") |
                        (loans_2007["loan_status"] == "Charged Off

mapping_dictionary = {"loan_status":{ "Fully Paid": 1, "Charged Off": (
loans_2007 = loans_2007.replace(mapping_dictionary)
```

Visualizing the Target Column Outcomes

In [26]:
```
filtered_loans = pd.read_csv('dataset/filtered_loans.csv')
filtered_loans.head(3)
fig, axs = plt.subplots(1,2,figsize=(14,7))
sns.countplot(x='loan_status',data=filtered_loans,ax=axs[0])
axs[0].set_title("Frequency of each Loan Status")
filtered_loans.loan_status.value_counts().plot(x=None,y=None, kind='pi
axs[1].set_title("Percentage of each Loan status")
plt.show()
```



These plots indicate that a significant number of borrowers in our dataset paid off their loan - 85.62% of loan borrowers paid off amount borrowed, while 14.38% unfortunately defaulted. From our loan data it is these 'defaulters' that we're more interested in filtering out as much as possible to reduce loses on investment returns.

Remove Columns with only One Value

To wrap up this section, let's look for any columns that contain only one unique value and remove them. These columns won't be useful for the model since they don't add any information to each loan application. In addition, removing these columns will reduce the number of columns we'll need to explore further in the next stage.

The pandas Series method nunique() returns the number of unique values, excluding any null values. We can use apply this method across the dataset to remove these columns in one easy step.

In [27]:
```
loans_2007 = loans_2007.loc[:,loans_2007.apply(pd.Series.nunique) != 1
```

Again, there may be some columns with more than one unique values but one of the values has insignificant frequency in the dataset. Let's find out and drop such column(s):

In [28]:
```
for col in loans_2007.columns:
    if (len(loans_2007[col].unique()) < 4):
        print(loans_2007[col].value_counts())
        print()
```

```
 36 months    29096
 60 months    10143
Name: term, dtype: int64

Not Verified       16845
Verified           12526
Source Verified     9868
Name: verification_status, dtype: int64

1    33586
0     5653
Name: loan_status, dtype: int64

False    39238
True         1
Name: pymnt_plan, dtype: int64
```

The payment plan column (pymnt_plan) has two unique values, 'y' and 'n', with 'y' occurring only once. Let's drop this column:

In [29]:
```
print(loans_2007.shape[1])
loans_2007 = loans_2007.drop('pymnt_plan', axis=1)
print("We've been able to reduced the features to => {}".format(loans_
```

```
25
We've been able to reduced the features to => 24
```

Lastly, lets save our work in this section to a CSV file.

In [30]:
```
loans_2007.to_csv("filtered_loans_2007_test1.csv",index=False)
```

3. Preparing the Features for Machine Learning

In this section, we'll prepare the filtered_loans_2007.csv data for machine learning. We'll focus on handling missing values, converting categorical columns to numeric columns and removing any other extraneous columns.

We need to handle missing values and categorical features before feeding the data into a machine learning algorithm, because the mathematics underlying most machine learning models assumes that the data is numerical and contains no missing values. To reinforce this requirement, scikit-learn will return an error if you try to train a model using data that contain missing values or non-numeric values when working with models like linear regression and logistic regression.

Here's an outline of what we'll be doing in this stage:

Handle Missing Values Investigate Categorical Columns Convert Categorical Columns To Numeric Features Map Ordinal Values To Integers Encode Nominal Values As Dummy Variables First though, let's load in the data from last section's final output:

In [31]:
```
filtered_loans = pd.read_csv('filtered_loans_2007_test1.csv')
print(filtered_loans.shape)
filtered_loans.head()
```

```
(39239, 24)
```

Out[31]:

	loan_amnt	term	installment	grade	emp_length	home_ownership	annual_inc	verificati
0	5000.0	36 months	162.87	B	10+ years	RENT	24000.0	
1	2500.0	60 months	59.83	C	< 1 year	RENT	30000.0	Sour
2	2400.0	36 months	84.33	C	10+ years	RENT	12252.0	N
3	10000.0	36 months	339.31	C	10+ years	RENT	49200.0	Sour
4	5000.0	36 months	156.46	A	3 years	RENT	36000.0	Sour

5 rows × 24 columns

Handle Missing Values

Let's compute the number of missing values and determine how to handle them. We can return the number of missing values across the DataFrame by:

First, use the Pandas DataFrame method isnull() to return a DataFrame containing Boolean values: True if the original value is null False if the original value isn't null Then, use the Pandas DataFrame method sum() to calculate the number of null values in each column.

In [32]:
```
null_counts = filtered_loans.isnull().sum()
print("Number of null values in each column:\n{}".format(null_counts))
```

```
Number of null values in each column:
loan_amnt                  0
term                       0
installment                0
grade                      0
emp_length              1057
home_ownership             0
annual_inc                 0
verification_status        0
loan_status                0
purpose                    0
title                     11
addr_state                 0
dti                        0
delinq_2yrs                0
earliest_cr_line           0
inq_last_6mths             0
open_acc                   0
pub_rec                    0
revol_bal                  0
revol_util                50
total_acc                  0
last_credit_pull_d         2
pub_rec_bankruptcies     697
fico_average               0
dtype: int64
```

Notice while most of the columns have 0 missing values, title has 9 missing values, revol_util has 48, and pub_rec_bankruptcies contains 675 rows with missing values. Let's remove columns entirely where more than 1% (392) of the rows for that column contain a null value. In addition, we'll remove the remaining rows containing null values, which means we'll lose a bit of data, but in return keep some extra features to use for prediction.

This means that we'll keep the title and revol_util columns, just removing rows containing missing values, but drop the pub_rec_bankruptcies column entirely since more than 1% of the rows have a missing value for this column.

Here's a list of steps we can use to achieve that:

Use the drop method to remove the pub_rec_bankruptcies column from filtered_loans. Use the dropna method to remove all rows from filtered_loans containing any missing values.

In [33]:
```
filtered_loans = filtered_loans.drop("pub_rec_bankruptcies",axis=1)
filtered_loans = filtered_loans.dropna()
```

Next, we'll focus on the categorical columns.

Investigate Categorical Columns

Keep in mind, the goal in this section is to have all the columns as numeric columns (int or float data type), and containing no missing values. We just dealt with the missing values, so let's now find out the number of columns that are of the object data type and then move on to process them into numeric form.

In [34]:
```
print("Data types and their frequency\n{}".format(filtered_loans.dtypes
```

```
Data types and their frequency
float64    12
object     10
int64       1
dtype: int64
```

In [35]:
```
object_columns_df = filtered_loans.select_dtypes(include=['object'])
print(object_columns_df.iloc[0])
```

```
term                    36 months
grade                           B
emp_length              10+ years
home_ownership               RENT
verification_status       Verified
purpose                credit_card
title                     Computer
addr_state                      AZ
earliest_cr_line          Jan-1985
last_credit_pull_d        Sep-2016
Name: 0, dtype: object
```

In [36]:
```
#filtered_loans['revol_util'] = filtered_loans['revol_util'].str.rstri
cols = ['home_ownership', 'grade','verification_status', 'emp_length',
for name in cols:
    print(name,':')
    print(object_columns_df[name].value_counts(),'\n')
```

```
home_ownership :
RENT        18271
MORTGAGE    16945
OWN          2808
OTHER          96
NONE            3
Name: home_ownership, dtype: int64

grade :
B    11545
A     9675
C     7801
D     5086
E     2715
F      993
G      308
Name: grade, dtype: int64

verification_status :
Not Verified       16391
Verified           12070
Source Verified     9662
Name: verification_status, dtype: int64

emp_length :
10+ years    8715
< 1 year     4542
2 years      4344
3 years      4050
4 years      3385
5 years      3243
1 year       3207
```

```
6 years      2198
7 years      1738
8 years      1457
9 years      1244
Name: emp_length, dtype: int64

term :
 36 months    28234
 60 months     9889
Name: term, dtype: int64

addr_state :
CA    6833
NY    3657
FL    2741
TX    2639
NJ    1802
IL    1476
PA    1460
VA    1359
GA    1340
MA    1292
OH    1167
MD    1020
AZ     819
WA     796
CO     755
NC     747
CT     719
MI     684
MO     653
MN     586
NV     473
SC     461
WI     433
OR     427
AL     424
LA     422
KY     315
OK     289
KS     253
UT     250
AR     232
DC     211
RI     195
NM     182
HI     166
WV     165
NH     159
DE     110
WY      79
MT      78
AK      77
```

```
SD      60
VT      53
MS      19
TN      17
IN       9
ID       6
IA       5
NE       5
ME       3
Name: addr_state, dtype: int64
```

In [37]:
```
for name in ['purpose','title']:
    print("Unique Values in column: {}\n".format(name))
    print(filtered_loans[name].value_counts(),'\n')
```

```
Unique Values in column: purpose

debt_consolidation    17965
credit_card            4944
other                  3764
home_improvement       2852
major_purchase         2105
small_business         1749
car                    1483
wedding                 927
medical                 663
moving                  556
house                   359
vacation                349
educational             312
renewable_energy         95
Name: purpose, dtype: int64

Unique Values in column: title

Debt Consolidation                                2102
Debt Consolidation Loan                           1635
Personal Loan                                      632
Consolidation                                      495
debt consolidation                                 476
Credit Card Consolidation                          346
Home Improvement                                   340
Debt consolidation                                 321
Small Business Loan                                305
Credit Card Loan                                   299
Personal                                           294
Consolidation Loan                                 254
Home Improvement Loan                              235
personal loan                                      223
personal                                           204
Loan                                               204
Wedding Loan                                       204
```

```
Car Loan                                                191
consolidation                                           190
Other Loan                                              174
Wedding                                                 151
Credit Card Payoff                                      147
Credit Card Refinance                                   140
Major Purchase Loan                                     136
Consolidate                                             125
Medical                                                 112
Credit Card                                             110
home improvement                                        103
My Loan                                                  92
Credit Cards                                             92
                                                        ...
I WANT TO PAY THIS OFF                                    1
Loan for College and Living Expenses                      1
3 and out                                                 1
Buy a Car                                                 1
No More High Interest Credit Cards                        1
Credit Card Interest Rate Is a Money Pit                  1
08/23/11 $6,000                                           1
On my way to financial freedom!                           1
loan to buy rental property                               1
HELP ME!!!                                                1
Med Bills                                                 1
NO MORE MEDICAL BILLS!!!                                  1
Refinance debt                                            1
neverlate                                                 1
lower! rate!                                              1
New location to Tempe                                     1
Vegetarian Triathlete Seeks Payoff for Citi Loan          1
 investment loan                                          1
Investment Property Rehab                                 1
Finishing Off a Student Loan                              1
Home Flooring Investment                                  1
ADS Cr Card Ln                                            1
Dental loan                                               1
Personal Loan 11/2009                                     1
Bonus                                                     1
Adding 4th Unit on 3 Unit Home                            1
F250                                                      1
Big Banks Banking On My Stable History                    1
Debt Refi & Consolidation                                 1
Consolidate debt.. Never late..                           1
Name: title, Length: 19021, dtype: int64
```

In [38]:
```
drop_cols = ['last_credit_pull_d','addr_state','title','earliest_cr_li
filtered_loans = filtered_loans.drop(drop_cols,axis=1)
```

Convert Categorical Columns to Numeric Features

First, let's understand the two types of categorical features we have in our dataset and how we can convert each to numerical features:

Ordinal values: these categorical values are in natural order. That's you can sort or order them either in increasing or decreasing order. For instance, we learnt earlier that Lending Club grade loan applicants from A to G, and assign each applicant a corresponding interest rate - grade A is less riskier while grade B is riskier than A in that order: A < B < C < D < E < F < G ; where < means less riskier than Nominal Values: these are regular categorical values. You can't order nominal values. For instance, while we can order loan applicants in the employment length column (emp_length) based on years spent in the workforce: year 1 < year 2 < year 3 ... < year N, we can't do that with the column purpose. It wouldn't make sense to say:

car < wedding < education < moving < house These are the columns we now have in our dataset:

Ordinal Values grade emp*length Nominal Values* home_ownership verification_status purpose term There are different approaches to handle each of these two types. In the steps following, we'll convert each of them accordingly.

To map the ordinal values to integers, we can use the pandas DataFrame method replace() to map both grade and emp_length to appropriate numeric values

In [39]:
```
mapping_dict = {
    "emp_length": {
        "10+ years": 10,
        "9 years": 9,
        "8 years": 8,
        "7 years": 7,
        "6 years": 6,
        "5 years": 5,
        "4 years": 4,
        "3 years": 3,
        "2 years": 2,
        "1 year": 1,
        "< 1 year": 0,
        "n/a": 0

    },
    "grade":{
        "A": 1,
        "B": 2,
        "C": 3,
        "D": 4,
        "E": 5,
        "F": 6,
        "G": 7
    }
}

filtered_loans = filtered_loans.replace(mapping_dict)
filtered_loans[['emp_length','grade']].head()
```

Out[39]:

	emp_length	grade
0	10	2
1	0	3
2	10	3
3	10	3
4	3	1

Perfect! Let's move on to the Nominal Values. The approach to converting nominal features into numerical features is to encode them as dummy variables. The process will be:

Use pandas' get_dummies() method to return a new DataFrame containing a new column for each dummy variable Use the concat() method to add these dummy columns back to the original DataFrame Then drop the original columns entirely using the drop method Lets' go ahead and encode the nominal columns that we now have in our dataset.

In [40]:
```
nominal_columns = ["home_ownership", "verification_status", "purpose",
dummy_df = pd.get_dummies(filtered_loans[nominal_columns])
filtered_loans = pd.concat([filtered_loans, dummy_df], axis=1)
filtered_loans = filtered_loans.drop(nominal_columns, axis=1)
```

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

Out[41]:

	loan_amnt	installment	grade	emp_length	annual_inc	loan_status	dti	delinq_2yrs	inc
0	5000.0	162.87	2	10	24000.0	1	27.65	0.0	
1	2500.0	59.83	3	0	30000.0	0	1.00	0.0	
2	2400.0	84.33	3	10	12252.0	1	8.72	0.0	
3	10000.0	339.31	3	10	49200.0	1	20.00	0.0	
4	5000.0	156.46	1	3	36000.0	1	11.20	0.0	

5 rows × 39 columns

To wrap things up, let's inspect our final output from this section to make sure all the features are of the same length, contain no null value, and are numericals.

Let's use pandas info method to inspect the filtered_loans DataFrame:

In [42]:
```
filtered_loans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38123 entries, 0 to 39238
Data columns (total 39 columns):
loan_amnt                                38123 non-null float64
installment                              38123 non-null float64
grade                                    38123 non-null int64
emp_length                               38123 non-null int64
annual_inc                               38123 non-null float64
loan_status                              38123 non-null int64
dti                                      38123 non-null float64
delinq_2yrs                              38123 non-null float64
inq_last_6mths                           38123 non-null float64
open_acc                                 38123 non-null float64
pub_rec                                  38123 non-null float64
revol_bal                                38123 non-null float64
revol_util                               38123 non-null float64
total_acc                                38123 non-null float64
fico_average                             38123 non-null float64
home_ownership_MORTGAGE                  38123 non-null uint8
home_ownership_NONE                      38123 non-null uint8
home_ownership_OTHER                     38123 non-null uint8
home_ownership_OWN                       38123 non-null uint8
home_ownership_RENT                      38123 non-null uint8
verification_status_Not Verified         38123 non-null uint8
verification_status_Source Verified      38123 non-null uint8
verification_status_Verified             38123 non-null uint8
purpose_car                              38123 non-null uint8
purpose_credit_card                      38123 non-null uint8
purpose_debt_consolidation               38123 non-null uint8
purpose_educational                      38123 non-null uint8
purpose_home_improvement                 38123 non-null uint8
purpose_house                            38123 non-null uint8
purpose_major_purchase                   38123 non-null uint8
purpose_medical                          38123 non-null uint8
purpose_moving                           38123 non-null uint8
purpose_other                            38123 non-null uint8
purpose_renewable_energy                 38123 non-null uint8
purpose_small_business                   38123 non-null uint8
purpose_vacation                         38123 non-null uint8
purpose_wedding                          38123 non-null uint8
term_ 36 months                          38123 non-null uint8
term_ 60 months                          38123 non-null uint8
dtypes: float64(12), int64(3), uint8(24)
memory usage: 5.5 MB
```

Save to CSV

It is a good practice to store the final output of each section or stage of your workflow in a separate csv file. One of the benefits of this practice is that it helps us to make changes in our data processing flow without having to recalculate everything.

In [43]:
```
filtered_loans.to_csv("cleaned_loans_2007_test.csv",index=False)
```