Lending Club

The Data

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

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), we build a Neural Network model that can predict wether or not a borrower will pay back their loan.

The "loan_status" column contains our label.

Let's import the packages we will need

In [1]:
```
import numpy as np
import pandas as pd                              # Allows working with dataframes

import matplotlib.pyplot as plt                  # Graphics package
import seaborn as sns                            # Enhanced graphics package
sns.set(style='darkgrid')

import re                                        #regex(regular expression) module
```

In [2]:
```
# ML Data preparation

from sklearn.preprocessing import MinMaxScaler                          # Data normalizati
from sklearn.model_selection import train_test_split                    # Model training/t
from sklearn.metrics import classification_report,confusion_matrix      # Model performanc
from sklearn.impute import KNNImputer                                   # Filling missing
```

In [3]:
```
#Neural Network

import tensorflow as tf

# Neural network settings
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

# Preventing overfitting
from tensorflow.keras.callbacks import EarlyStopping         # Training stopping when
from tensorflow.keras.layers import Dropout                  # Rnadomly drops nodes co
```

1. Data Overview

Let's load the data and try to visualize some general information in the dataset

'lending_club_info.csv' contains a description of the variables whose values are given in the historic file 'lending_club_loan_two.csv'

In [4]:
```
lc_info = pd.read_csv('../DATA/lending_club_info.csv')
pd.options.display.max_colwidth = 150 # This sets the amount length of string to be sh
lc_info
```

Out[4]:

	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 b...
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, d...
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

In [5]:
```
lc_loan = pd.read_csv('../DATA/lending_club_loan_two.csv')
lc_loan.head()
```

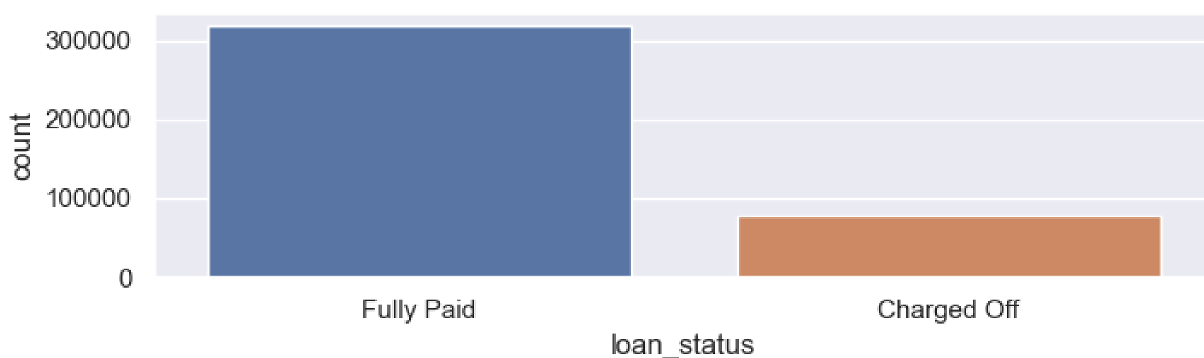
Out[5]:

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

5 rows × 27 columns

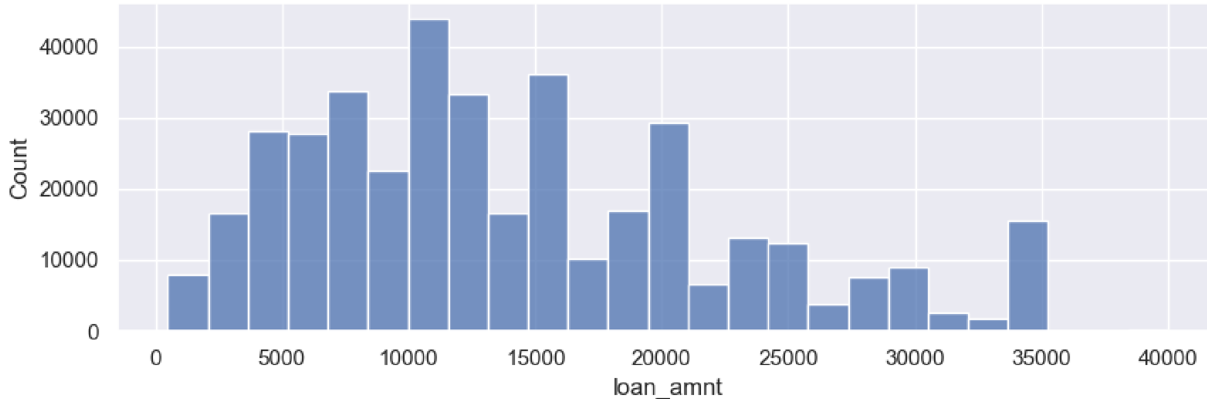
In [6]:
```
# Proportion of loans that are defaulted
plt.figure(figsize=(8,2))
sns.countplot(data=lc_loan, x='loan_status')
```

Out[6]: <Axes: xlabel='loan_status', ylabel='count'>



In [7]:
```
# Distribution of the amount borrowed for historic loans
plt.figure(figsize=(10,3))
sns.histplot(lc_loan['loan_amnt'], kde=False, bins=25)
```

Out[7]: <Axes: xlabel='loan_amnt', ylabel='Count'>



As we can see, the historical amount of money borrowed concentrates around 10K USD.

Let's now turn loan_status to a dummy variable so we can check the correlations with each other variable

In [8]:
```
# We associate each element of loan_status with a 0 or a 1 in the a column
lc_loan['status_dummy'] = (lc_loan['loan_status'].map({'Fully Paid':1,'Charged Off':0}
lc_loan[['loan_status','status_dummy']].tail(10)
```

Out[8]:

	loan_status	status_dummy
396020	Fully Paid	1
396021	Fully Paid	1
396022	Fully Paid	1
396023	Fully Paid	1
396024	Fully Paid	1
396025	Fully Paid	1
396026	Fully Paid	1
396027	Fully Paid	1
396028	Fully Paid	1
396029	Fully Paid	1

In [9]:
```
# Plot the correlations
plt.figure(figsize=(10,5))
sns.heatmap(lc_loan.corr(), annot=True)
```

```
C:\Users\lol_s\AppData\Local\Temp\ipykernel_15752\2775831331.py:3: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future version, i
t will default to False. Select only valid columns or specify the value of numeric_on
ly to silence this warning.
  sns.heatmap(lc_loan.corr(), annot=True)
```

Out[9]:
```
<Axes: >
```

Let's see description of the variables that show the greatest correlations

In [10]:
```
lc_info.set_index('LoanStatNew').loc[['installment', 'pub_rec', 'pub_rec_bankruptcies'
```

Out[10]:

LoanStatNew	Description
installment	The monthly payment owed by the borrower if the loan originates.
pub_rec	Number of derogatory public records
pub_rec_bankruptcies	Number of public record bankruptcies
open_acc	The number of open credit lines in the borrower's credit file.
total_acc	The total number of credit lines currently in the borrower's credit file

As expected, installment is pretty correlated to the total amount of money borrowed

Again, pub_rec and pub_rc_bankruptcies show similar information about public records about payment default

The relation between open_acc and total_acc is probably not that evident, as having a bigger or smaller record does not imply having more or less credit accounts currently open. This said, I would suggest the following reasons that make this happen (We would need more information to accept or reject them):

- The more experience a person have with credit lines, the more comfortable it feels to have them and perhaps the more that person can feel in need to have extra money for his

projects
- The older the person gets, the more accounts he would have oppened and because of his age, the more money income he could have managed to get and also (related or not to the latter) the less he would worry about having a money debt

Correlation with status_dummy

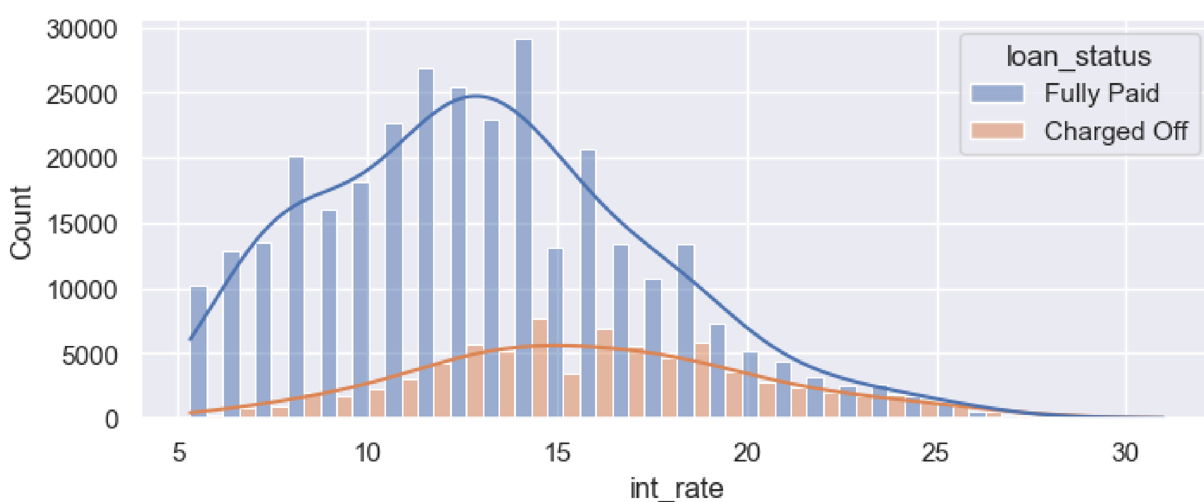
In [11]:
```
# Correlations with the target feature
lc_loan.corr()['status_dummy'].abs().sort_values()
```

```
C:\Users\lol_s\AppData\Local\Temp\ipykernel_15752\3795390216.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  lc_loan.corr()['status_dummy'].abs().sort_values()
```

Out[11]:
```
pub_rec_bankruptcies    0.009383
revol_bal               0.010892
total_acc               0.017893
pub_rec                 0.019933
open_acc                0.028012
installment             0.041082
annual_inc              0.053432
loan_amnt               0.059836
dti                     0.062413
mort_acc                0.073111
revol_util              0.082373
int_rate                0.247758
status_dummy            1.000000
Name: status_dummy, dtype: float64
```

In [12]:
```
# Distribution with respect to the interest rate and clusterized by payment status
plt.figure(figsize=(8,3))
sns.histplot(lc_loan, x='int_rate',hue='loan_status', multiple='dodge', bins=30, kde=T
```

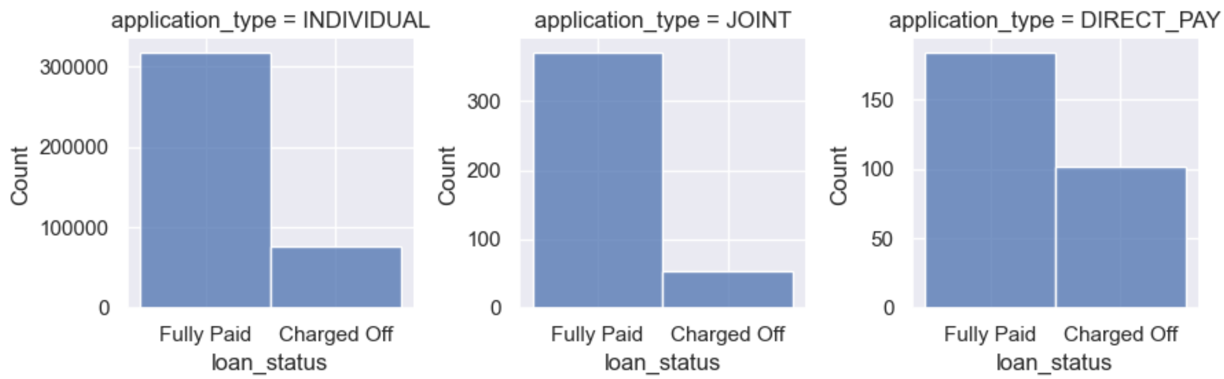
Out[12]: <Axes: xlabel='int_rate', ylabel='Count'>



There's better payment rate for bigger interest rate, although the difference is not significant

In [13]:
```
# Proportion of Fully paid/Charged off loans for each kind of application type
g = sns.FacetGrid(lc_loan,col='application_type', sharey=False, height=3)
#If we let share y, the 2nd and 3rd graphs would be tiny if compared to the 1st
g.map(sns.histplot,'loan_status')
```

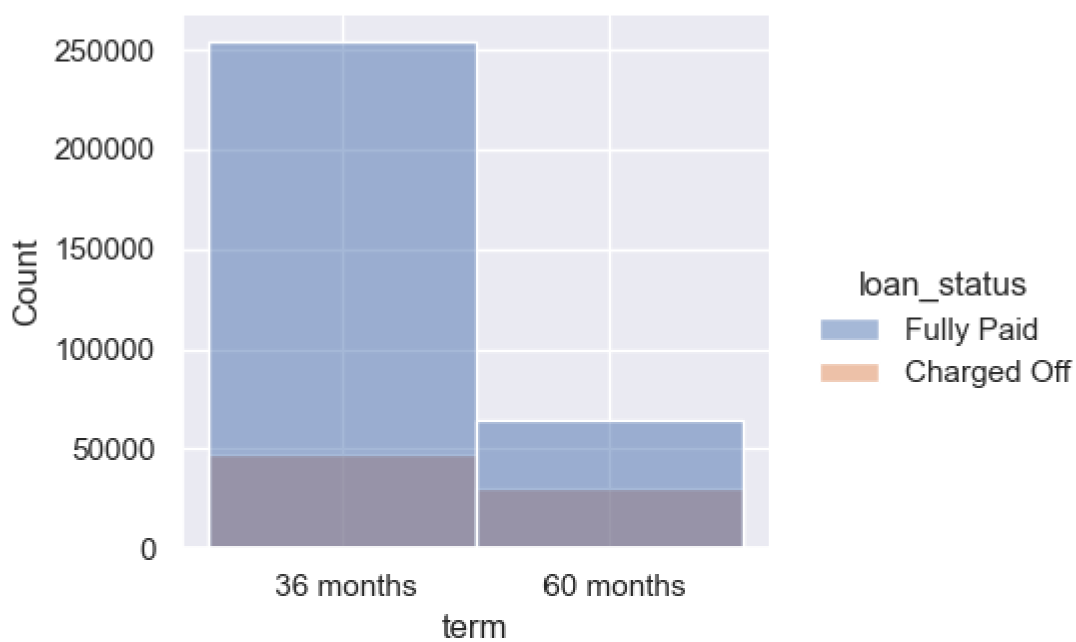
Out[13]: <seaborn.axisgrid.FacetGrid at 0x2083e69ecb0>



The direct payment type is the most risky and the joint type is the least

In [14]:
```
# Dependency of loan_status with the duration of the loan (term column)
sns.displot(lc_loan, x='term', hue='loan_status', height=3.5, aspect=1.2)
```

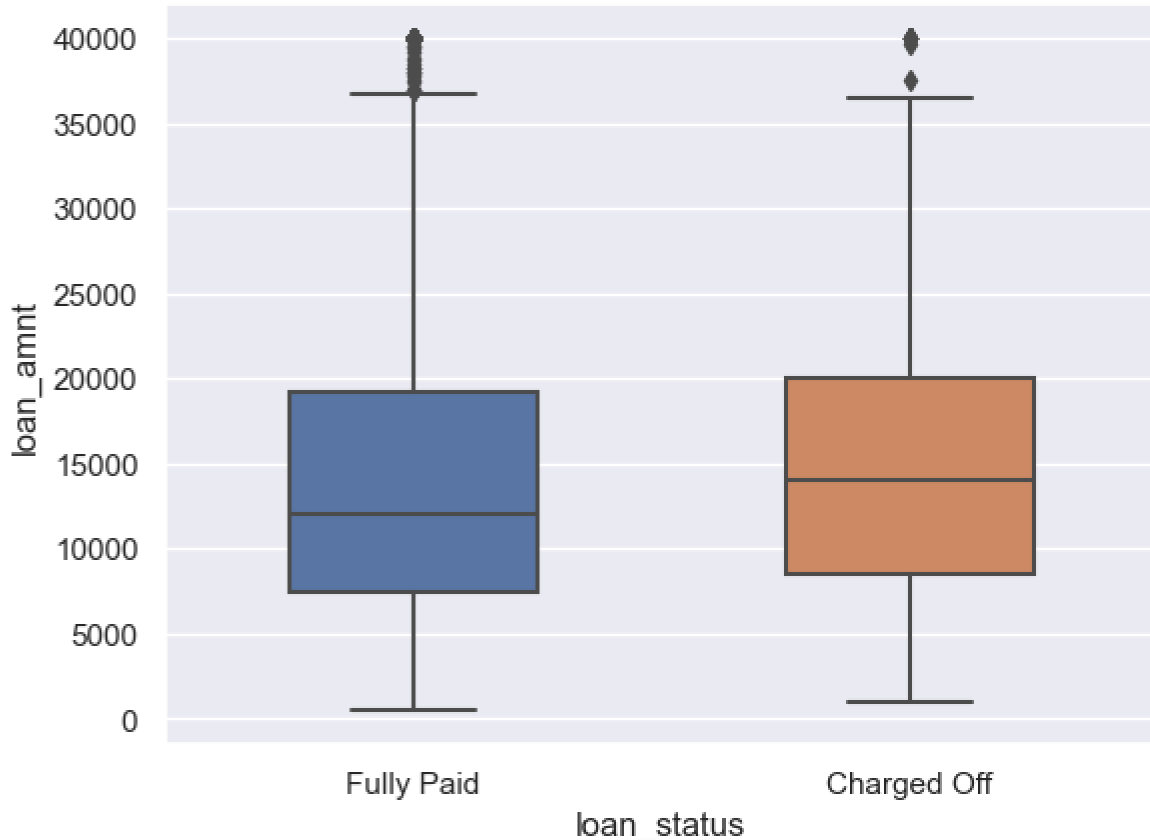
Out[14]: <seaborn.axisgrid.FacetGrid at 0x2083e712440>



The proportion Fully paid vs. Charged off is much greater in loans with a 36 months duration

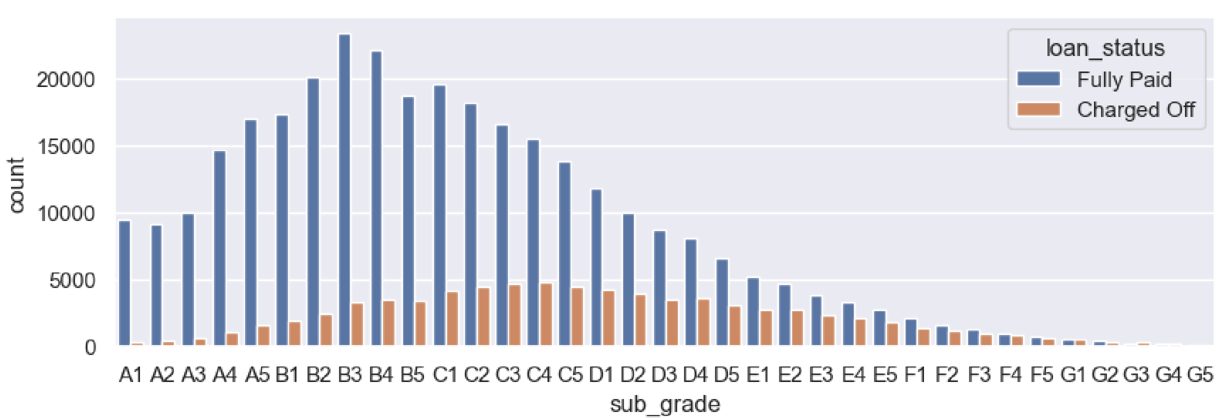
In [15]:
```
# Distribution of Fully loan amounts for fully paid and charged off loans
sns.boxplot(data=lc_loan, x='loan_status',y='loan_amnt', width=0.5)
```

Out[15]: <Axes: xlabel='loan_status', ylabel='loan_amnt'>



In [16]:
```
# Distribution of loans respect to the quality score assigned (sub_grade)
plt.figure(figsize=(10,3))
sns.countplot(data=lc_loan, x='sub_grade', hue='loan_status', order=sorted(lc_loan['su
```

Out[16]:
```
<Axes: xlabel='sub_grade', ylabel='count'>
```



As shown, the proportion of 'fully paid' vs. 'charged off' loans grows between the A1 and the B3 subgrades, then it keeps reducing at a decreasing rate up to the F3 subgrade and from there it's fairly constant

In [17]:
```
lc_info[lc_info['LoanStatNew'] == 'initial_list_status'].Description
```

Out[17]:
```
24    The initial listing status of the loan. Possible values are – W, F
Name: Description, dtype: object
```

W Loans are a random set of loans which were initially available for whole purchase for investors

In [18]:
```
# Dependency of loan_status with the initial list status
plt.figure(figsize=(4,3.5))
sns.histplot(lc_loan, x='initial_list_status', hue='loan_status', multiple='fill')
```

Out[18]:
```
<Axes: xlabel='initial_list_status', ylabel='Count'>
```

2. Data transformation

2.1. Direct transformations

In [19]:
```
lc_loan.head()
```

Out[19]:

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

5 rows × 28 columns

We need to turn some of the features to numerical type in order to use them in our neural network

emp_length

In [20]:
```
lc_loan['emp_length'].unique() #There are NaN values
```

Out[20]:
```
array(['10+ years', '4 years', '< 1 year', '6 years', '9 years',
       '2 years', '3 years', '8 years', '7 years', '5 years', '1 year',
       nan], dtype=object)
```

In [21]:
```
def str_to_num(str):
    '''
    This function returns the input variable if it's a float, 0 if the that variable s
    in the string variable in any other case
    '''
    if type(str)==float:
        return str
    elif str.split()[0]=='<':
        return 0
    else:
        return int(re.findall(r'-?\d+\.?\d*', str)[0])
```

In [22]:
```
lc_loan['emp_length'] = lc_loan['emp_length'].apply(str_to_num)
lc_loan['emp_length'].unique()
```

Out[22]:
```
array([10.,  4.,  0.,  6.,  9.,  2.,  3.,  8.,  7.,  5.,  1., nan])
```

Same process with 'term'

In [23]:
```
lc_loan['term'] = lc_loan['term'].apply(str_to_num)
lc_loan['term'].unique()
```

Out[23]:
```
array([36, 60], dtype=int64)
```

Subgrade column to numerical

following the next pattern:

A1 -> 1.00 B1 -> 2.00 ... G1 -> 7.00

A2 -> 1.20 B2 -> 2.20 ... G2 -> 7.20

A3 -> 1.40 B3 -> 2.40 ... G3 -> 7.40

A4 -> 1.60 B4 -> 2.60 ... G4 -> 7.60

A5 -> 1.80 B5 -> 2.80 ... G5 -> 7.80

In [24]:
```
lc_loan['sub_grade'].unique()
```

Out[24]:
```
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)
```

In [25]:
```
def alphnum_to_num(grad):
    '''
    This function makes the letter in subgrade correspond to the integer part of an ou
    to the decimal part of the output, i.e., it turns the alphanumerical input to nume
    '''
    grad_n=[]

    #First, we check the letter
    sep = [char for char in grad]
    if sep[0]=='A':
        grad_n.append(1)
    elif sep[0]=='B':
        grad_n.append(2)
    elif sep[0]=='C':
        grad_n.append(3)
    elif sep[0]=='D':
        grad_n.append(4)
    elif sep[0]=='E':
        grad_n.append(5)
    elif sep[0]=='F':
        grad_n.append(6)
    elif sep[0]=='G':
        grad_n.append(7)

    #Now let's check subgrades:
    if sep[1]=='1':
        grad_n.append(0.00)
    elif sep[1]=='2':
        grad_n.append(0.20)
    elif sep[1]=='3':
        grad_n.append(0.40)
    elif sep[1]=='4':
        grad_n.append(0.60)
    elif sep[1]=='5':
```

```
        grad_n.append(0.80)

    return sum(grad_n)
```

In [26]:
```
lc_loan['sub_grade'] = lc_loan['sub_grade'].apply(alphnum_to_num)
lc_loan.sub_grade.isnull().sum() # No null values
```

Out[26]:
```
0
```

In [27]:
```
# Drop grade column
lc_loan.drop(['grade'], axis=1, inplace=True)
```

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

Out[28]:

	loan_amnt	term	int_rate	installment	sub_grade	emp_title	emp_length	home_ownership	an
0	10000.0	36	11.44	329.48	2.6	Marketing	10.0	RENT	1
1	8000.0	36	11.99	265.68	2.8	Credit analyst	4.0	MORTGAGE	
2	15600.0	36	10.49	506.97	2.4	Statistician	0.0	RENT	
3	7200.0	36	6.49	220.65	1.2	Client Advocate	6.0	RENT	
4	24375.0	60	17.27	609.33	3.8	Destiny Management Inc.	9.0	MORTGAGE	

5 rows × 27 columns

2.2. Create dummy variables for categorical features

application_type

In [29]:
```
lc_loan['application_type'].unique()
```

Out[29]:
```
array(['INDIVIDUAL', 'JOINT', 'DIRECT_PAY'], dtype=object)
```

In [30]:
```
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum', columns=['application_type'], drc
lc_loan.head()
```

Out[30]:

	loan_amnt	term	int_rate	installment	sub_grade	emp_title	emp_length	home_ownership	an
0	10000.0	36	11.44	329.48	2.6	Marketing	10.0	RENT	1
1	8000.0	36	11.99	265.68	2.8	Credit analyst	4.0	MORTGAGE	
2	15600.0	36	10.49	506.97	2.4	Statistician	0.0	RENT	
3	7200.0	36	6.49	220.65	1.2	Client Advocate	6.0	RENT	
4	24375.0	60	17.27	609.33	3.8	Destiny Management Inc.	9.0	MORTGAGE	

5 rows × 28 columns

In [31]:
```
lc_loan.columns
```

Out[31]:
```
Index(['loan_amnt', 'term', 'int_rate', 'installment', 'sub_grade',
       'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
       'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',
       'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'mort_acc',
       'pub_rec_bankruptcies', 'address', 'status_dummy', 'dum_INDIVIDUAL',
       'dum_JOINT'],
      dtype='object')
```

In [32]:
```
# We can rename the new columns if we want
lc_loan.rename(
    columns={"dum_INDIVIDUAL": "Individual", "dum_JOINT": "Joint"},
    inplace=True
)
```

emp_title

In [33]:
```
# Number of unique employments
lc_loan['emp_title'].nunique()
```

Out[33]:
```
173105
```

In [34]:
```
lc_loan['emp_title'].value_counts()[:30]
```

Out[34]:
```
Teacher                    4389
Manager                    4250
Registered Nurse           1856
RN                         1846
Supervisor                 1830
Sales                      1638
Project Manager            1505
Owner                      1410
Driver                     1339
Office Manager             1218
manager                    1145
Director                   1089
General Manager            1074
Engineer                    995
teacher                     962
driver                      882
Vice President              857
Operations Manager          763
Administrative Assistant    756
Accountant                  748
President                   742
owner                       697
Account Manager             692
Police Officer              686
supervisor                  673
Attorney                    667
Sales Manager               665
sales                       645
Executive Assistant         642
Analyst                     623
Name: emp_title, dtype: int64
```

As there are too many different jobs, we drop emp_title

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

title and purpose columns

In [36]:
```
lc_loan['title'].head(10)
```

Out[36]:
```
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 [37]:
```
lc_loan['purpose'].head(10)
```

Out[37]:
```
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
```

title and purpose show about the same information. We can drop title

In [38]:
```
lc_loan.drop('title', axis=1, inplace=True)
```

In [39]:
```
# Number of unique values
lc_loan['purpose'].nunique()
```

Out[39]:
```
14
```

Let's make it a dummy variable

In [40]:
```
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum', columns=['purpose'], drop_first=T
```

In [41]:
```
lc_loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 38 columns):
 #   Column                   Non-Null Count   Dtype  
---  ------                   --------------   -----  
 0   loan_amnt                396030 non-null  float64
 1   term                     396030 non-null  int64  
 2   int_rate                 396030 non-null  float64
 3   installment              396030 non-null  float64
 4   sub_grade                396030 non-null  float64
 5   emp_length               377729 non-null  float64
 6   home_ownership           396030 non-null  object 
 7   annual_inc               396030 non-null  float64
 8   verification_status      396030 non-null  object 
 9   issue_d                  396030 non-null  object 
 10  loan_status              396030 non-null  object 
 11  dti                      396030 non-null  float64
 12  earliest_cr_line         396030 non-null  object 
 13  open_acc                 396030 non-null  float64
 14  pub_rec                  396030 non-null  float64
 15  revol_bal                396030 non-null  float64
 16  revol_util               395754 non-null  float64
 17  total_acc                396030 non-null  float64
 18  initial_list_status      396030 non-null  object 
 19  mort_acc                 358235 non-null  float64
 20  pub_rec_bankruptcies     395495 non-null  float64
 21  address                  396030 non-null  object 
 22  status_dummy             396030 non-null  uint8  
 23  Individual               396030 non-null  uint8  
 24  Joint                    396030 non-null  uint8  
 25  dum_credit_card          396030 non-null  uint8  
 26  dum_debt_consolidation   396030 non-null  uint8  
 27  dum_educational          396030 non-null  uint8  
 28  dum_home_improvement     396030 non-null  uint8  
 29  dum_house                396030 non-null  uint8  
 30  dum_major_purchase       396030 non-null  uint8  
 31  dum_medical              396030 non-null  uint8  
 32  dum_moving               396030 non-null  uint8  
 33  dum_other                396030 non-null  uint8  
 34  dum_renewable_energy     396030 non-null  uint8  
 35  dum_small_business       396030 non-null  uint8  
 36  dum_vacation             396030 non-null  uint8  
 37  dum_wedding              396030 non-null  uint8  
dtypes: float64(14), int64(1), object(7), uint8(16)
memory usage: 72.5+ MB
```

home_ownership

In [42]:
```
lc_loan['home_ownership'].value_counts()
```

Out[42]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          112
NONE            31
ANY              3
Name: home_ownership, dtype: int64
```

Let's join Other, None and Any into 1 category called Other

In [43]:
```
lc_loan['home_own'] = lc_loan['home_ownership']
```

In [44]:
```
lc_loan.head(30)
```

Out[44]:

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	home_ownership	annual_inc	veri
0	10000.0	36	11.44	329.48	2.6	10.0	RENT	117000.0	
1	8000.0	36	11.99	265.68	2.8	4.0	MORTGAGE	65000.0	
2	15600.0	36	10.49	506.97	2.4	0.0	RENT	43057.0	
3	7200.0	36	6.49	220.65	1.2	6.0	RENT	54000.0	
4	24375.0	60	17.27	609.33	3.8	9.0	MORTGAGE	55000.0	
5	20000.0	36	13.33	677.07	3.4	10.0	MORTGAGE	86788.0	
6	18000.0	36	5.32	542.07	1.0	2.0	MORTGAGE	125000.0	
7	13000.0	36	11.14	426.47	2.2	10.0	RENT	46000.0	
8	18900.0	60	10.99	410.84	2.4	10.0	RENT	103000.0	
9	26300.0	36	16.29	928.40	3.8	3.0	MORTGAGE	115000.0	
10	10000.0	36	13.11	337.47	2.6	2.0	RENT	95000.0	
11	35000.0	36	14.64	1207.13	3.4	8.0	MORTGAGE	130000.0	
12	7500.0	36	9.17	239.10	2.2	7.0	OWN	55000.0	
13	35000.0	60	12.29	783.70	3.0	10.0	MORTGAGE	157000.0	
14	25975.0	36	6.62	797.53	1.2	9.0	MORTGAGE	65000.0	
15	18000.0	36	8.39	567.30	1.8	8.0	MORTGAGE	45000.0	
16	32350.0	60	21.98	893.11	5.6	10.0	MORTGAGE	72000.0	
17	11200.0	60	12.29	250.79	3.0	10.0	MORTGAGE	81000.0	
18	34000.0	36	7.90	1063.87	1.6	10.0	RENT	130580.0	
19	20000.0	36	6.97	617.27	1.4	7.0	MORTGAGE	85000.0	
20	9200.0	36	6.62	282.48	1.2	0.0	RENT	65000.0	

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	home_ownership	annual_inc	veri
21	7350.0	36	13.11	248.05	2.6	10.0	MORTGAGE	54800.0	
22	4200.0	36	6.99	129.67	1.4	5.0	OWN	24000.0	
23	20000.0	36	8.39	630.34	1.8	10.0	OWN	55000.0	
24	5000.0	36	15.61	174.83	4.0	5.0	RENT	75000.0	
25	6000.0	36	11.36	197.47	2.8	2.0	RENT	46680.0	
26	8400.0	36	13.35	284.45	3.2	6.0	RENT	35000.0	
27	23050.0	36	12.12	766.92	2.4	3.0	RENT	80000.0	
28	15000.0	36	9.99	483.94	2.0	10.0	MORTGAGE	79000.0	
29	20000.0	36	8.19	628.49	1.8	10.0	MORTGAGE	87000.0	

30 rows × 39 columns

In [45]:
```
lc_loan['home_own'] = lc_loan['home_ownership'].apply(lambda x: 'OTHER' if x in ['NONE
# The lambda function used returns OTHER if the input is NONE or ANY
```

In [46]:
```
lc_loan['home_own'].value_counts()
```

Out[46]:
```
MORTGAGE    198348
RENT        159790
OWN          37746
OTHER          146
Name: home_own, dtype: int64
```

In [47]:
```
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum', columns=['home_own'], drop_first=
lc_loan.drop(['home_ownership'], axis=1, inplace=True)          #Drop original column
```

In [48]:
```
lc_loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 40 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             396030 non-null  float64
 1   term                  396030 non-null  int64  
 2   int_rate              396030 non-null  float64
 3   installment           396030 non-null  float64
 4   sub_grade             396030 non-null  float64
 5   emp_length            377729 non-null  float64
 6   annual_inc            396030 non-null  float64
 7   verification_status   396030 non-null  object 
 8   issue_d               396030 non-null  object 
 9   loan_status           396030 non-null  object 
 10  dti                   396030 non-null  float64
 11  earliest_cr_line      396030 non-null  object 
 12  open_acc              396030 non-null  float64
 13  pub_rec               396030 non-null  float64
 14  revol_bal             396030 non-null  float64
 15  revol_util            395754 non-null  float64
 16  total_acc             396030 non-null  float64
 17  initial_list_status   396030 non-null  object 
 18  mort_acc              358235 non-null  float64
 19  pub_rec_bankruptcies  395495 non-null  float64
 20  address               396030 non-null  object 
 21  status_dummy          396030 non-null  uint8  
 22  Individual            396030 non-null  uint8  
 23  Joint                 396030 non-null  uint8  
 24  dum_credit_card       396030 non-null  uint8  
 25  dum_debt_consolidation  396030 non-null  uint8  
 26  dum_educational       396030 non-null  uint8  
 27  dum_home_improvement  396030 non-null  uint8  
 28  dum_house             396030 non-null  uint8  
 29  dum_major_purchase    396030 non-null  uint8  
 30  dum_medical           396030 non-null  uint8  
 31  dum_moving            396030 non-null  uint8  
 32  dum_other             396030 non-null  uint8  
 33  dum_renewable_energy  396030 non-null  uint8  
 34  dum_small_business    396030 non-null  uint8  
 35  dum_vacation          396030 non-null  uint8  
 36  dum_wedding           396030 non-null  uint8  
 37  dum_OTHER             396030 non-null  uint8  
 38  dum_OWN               396030 non-null  uint8  
 39  dum_RENT              396030 non-null  uint8  
dtypes: float64(14), int64(1), object(6), uint8(19)
memory usage: 70.6+ MB
```

verification_status

In [49]:
```
lc_loan['verification_status'].value_counts()
```

Out[49]:
```
Verified           139563
Source Verified    131385
Not Verified       125082
Name: verification_status, dtype: int64
```

In [50]:
```
# Make it a dummy
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum', columns=['verification_status'],
```

Issue_d

Let's recall what issue_d means

In [51]:
```
lc_info.columns
```

Out[51]:
```
Index(['LoanStatNew', 'Description'], dtype='object')
```

In [52]:
```
lc_info[lc_info['LoanStatNew'] == 'issue_d'].Description
```

Out[52]:
```
11    The month which the loan was funded
Name: Description, dtype: object
```

In [53]:
```
lc_loan['issue_d']
```

Out[53]:
```
0         Jan-2015
1         Jan-2015
2         Jan-2015
3         Nov-2014
4         Apr-2013
            ...
396025    Oct-2015
396026    Feb-2015
396027    Oct-2013
396028    Aug-2012
396029    Jun-2010
Name: issue_d, Length: 396030, dtype: object
```

If we can broke down issue_d into Year and Month, would both variables be treated lenearly?

In [54]:
```
# Save the Loan month
lc_loan['loan_month'] = lc_loan['issue_d'].apply(lambda x: x[-8:-5]) # 8th position to
```

In [55]:
```
lc_loan['loan_month'].value_counts()
```

Out[55]:
```
Oct    42130
Jul    39714
Jan    34682
Nov    34068
Apr    33223
Aug    32816
Mar    31919
May    31895
Jun    30140
Dec    29082
Feb    28742
Sep    27619
Name: loan_month, dtype: int64
```

In [56]:
```
# Save the Loan year
lc_loan['loan_year'] = lc_loan['issue_d'].apply(lambda x: x[-4:]).astype(np.uint16)
```

In [57]:
```
lc_loan['loan_year'].value_counts()
```

Out[57]:
```
2014    102860
2013     97662
2015     94264
2012     41202
2016     28088
2011     17435
2010      9258
2009      3826
2008      1240
2007       195
Name: loan_year, dtype: int64
```

We have the 12 months of a Year and 10 diferent years. We can treat the "month variable categorically" as the month in which future loans will be requested will fall into one of the existing categories (the 12 months), but regarding the years, new loans will occur in future years and the values of the past won't be repeated (maybe the last year will if it's not over yet). Because of that I will try "years linearly"

Turn month into dummy

In [58]:
```
# Turn month into dummy
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum_loan', columns=['loan_month'], drop
lc_loan.drop(['issue_d'], axis=1, inplace=True)        # Drop original column
```

In [59]:
```
# Let's have a look
lc_loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 52 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   loan_amnt              396030 non-null  float64
 1   term                   396030 non-null  int64  
 2   int_rate               396030 non-null  float64
 3   installment            396030 non-null  float64
 4   sub_grade              396030 non-null  float64
 5   emp_length             377729 non-null  float64
 6   annual_inc             396030 non-null  float64
 7   loan_status            396030 non-null  object 
 8   dti                    396030 non-null  float64
 9   earliest_cr_line       396030 non-null  object 
 10  open_acc               396030 non-null  float64
 11  pub_rec                396030 non-null  float64
 12  revol_bal              396030 non-null  float64
 13  revol_util             395754 non-null  float64
 14  total_acc              396030 non-null  float64
 15  initial_list_status    396030 non-null  object 
 16  mort_acc               358235 non-null  float64
 17  pub_rec_bankruptcies   395495 non-null  float64
 18  address                396030 non-null  object 
 19  status_dummy           396030 non-null  uint8  
 20  Individual             396030 non-null  uint8  
 21  Joint                  396030 non-null  uint8  
 22  dum_credit_card        396030 non-null  uint8  
 23  dum_debt_consolidation  396030 non-null  uint8  
 24  dum_educational        396030 non-null  uint8  
 25  dum_home_improvement   396030 non-null  uint8  
 26  dum_house              396030 non-null  uint8  
 27  dum_major_purchase     396030 non-null  uint8  
 28  dum_medical            396030 non-null  uint8  
 29  dum_moving             396030 non-null  uint8  
 30  dum_other              396030 non-null  uint8  
 31  dum_renewable_energy   396030 non-null  uint8  
 32  dum_small_business     396030 non-null  uint8  
 33  dum_vacation           396030 non-null  uint8  
 34  dum_wedding            396030 non-null  uint8  
 35  dum_OTHER              396030 non-null  uint8  
 36  dum_OWN                396030 non-null  uint8  
 37  dum_RENT               396030 non-null  uint8  
 38  dum_Source Verified    396030 non-null  uint8  
 39  dum_Verified           396030 non-null  uint8  
 40  loan_year              396030 non-null  uint16 
 41  dum_loan_Aug           396030 non-null  uint8  
 42  dum_loan_Dec           396030 non-null  uint8  
 43  dum_loan_Feb           396030 non-null  uint8  
 44  dum_loan_Jan           396030 non-null  uint8  
 45  dum_loan_Jul           396030 non-null  uint8  
 46  dum_loan_Jun           396030 non-null  uint8  
 47  dum_loan_Mar           396030 non-null  uint8  
 48  dum_loan_May           396030 non-null  uint8  
 49  dum_loan_Nov           396030 non-null  uint8  
 50  dum_loan_Oct           396030 non-null  uint8  
 51  dum_loan_Sep           396030 non-null  uint8  
dtypes: float64(14), int64(1), object(4), uint16(1), uint8(32)
memory usage: 70.2+ MB
```

earliest_cr_line

In [60]:
```
lc_loan['earliest_cr_line']
```

Out[60]:
```
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: 396030, dtype: object
```

In [61]:
```
# Description
lc_info[lc_info['LoanStatNew'] == 'earliest_cr_line']
```

Out[61]:

	LoanStatNew	Description
18	earliest_cr_line	The month the borrower's earliest reported credit line was opened

Let's do the same we did with issue_d

In [62]:
```
# Save the first credit line year
lc_loan['earliest_cr_year'] = lc_loan['earliest_cr_line'].apply(lambda x: x[-4:]).asty

# Save the first credit line month
lc_loan['earliest_cr_month'] = lc_loan['earliest_cr_line'].apply(lambda x: x[-8:-5])

# Turn month into dummy
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum_ear_cr', columns=['earliest_cr_mont
lc_loan.drop(['earliest_cr_line'], axis=1, inplace=True)        # Drop original colum
```

Initial list status

In [63]:
```
# Turn month into dummy
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum', columns=['initial_list_status'],
```

Address

In [64]:
```
lc_loan['address'].head()
```

Out[64]:
```
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
Name: address, dtype: object
```

In [65]:
```
lc_loan['zip'] = lc_loan['address'].apply(lambda x: x[-5:])
```

In [66]:
```
lc_loan['address'].apply(lambda x: x[-8:])
```

Out[66]:
```
0         OK 22690
1         SD 05113
2         WV 05113
3         MA 00813
4         VA 11650
            ...
396025    DC 30723
396026    LA 05113
396027    NY 70466
396028    FL 29597
396029    AR 48052
Name: address, Length: 396030, dtype: object
```

In [67]:
```
# How many State-zip code combinations are there?
lc_loan['address'].apply(lambda x: x[-8:]).nunique()
```

Out[67]:
```
540
```

In [68]:
```
# How many different zip codes?
lc_loan['zip'].nunique()
```

Out[68]:
```
10
```

Let's check how many States there are in the data

In [69]:
```
lc_loan['address'].apply(lambda x: x[-8:-6])
```

Out[69]:
```
0         OK
1         SD
2         WV
3         MA
4         VA
          ..
396025    DC
396026    LA
396027    NY
396028    FL
396029    AR
Name: address, Length: 396030, dtype: object
```

In [70]:
```
# How many different States?
lc_loan['address'].apply(lambda x: x[-8:-6]).nunique()
```

Out[70]:
```
54
```

I will just use the zipcode for now because there are too many states

In [71]:
```
lc_loan = pd.get_dummies(data=lc_loan, prefix='dum_zip', columns=['zip'], drop_first=T
lc_loan.drop('address', axis=1, inplace=True)          # Drop the original column
lc_loan.head()
```

Out[71]:

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	annual_inc	loan_status	dti	ope
0	10000.0	36	11.44	329.48	2.6	10.0	117000.0	Fully Paid	26.24	
1	8000.0	36	11.99	265.68	2.8	4.0	65000.0	Fully Paid	22.05	
2	15600.0	36	10.49	506.97	2.4	0.0	43057.0	Fully Paid	12.79	
3	7200.0	36	6.49	220.65	1.2	6.0	54000.0	Fully Paid	2.60	
4	24375.0	60	17.27	609.33	3.8	9.0	55000.0	Charged Off	33.95	

5 rows × 71 columns

In [72]:

```
lc_loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 396030 entries, 0 to 396029
Data columns (total 71 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   loan_amnt              396030 non-null  float64
 1   term                   396030 non-null  int64  
 2   int_rate               396030 non-null  float64
 3   installment            396030 non-null  float64
 4   sub_grade              396030 non-null  float64
 5   emp_length             377729 non-null  float64
 6   annual_inc             396030 non-null  float64
 7   loan_status            396030 non-null  object 
 8   dti                    396030 non-null  float64
 9   open_acc               396030 non-null  float64
 10  pub_rec                396030 non-null  float64
 11  revol_bal              396030 non-null  float64
 12  revol_util             395754 non-null  float64
 13  total_acc              396030 non-null  float64
 14  mort_acc               358235 non-null  float64
 15  pub_rec_bankruptcies   395495 non-null  float64
 16  status_dummy           396030 non-null  uint8  
 17  Individual             396030 non-null  uint8  
 18  Joint                  396030 non-null  uint8  
 19  dum_credit_card        396030 non-null  uint8  
 20  dum_debt_consolidation 396030 non-null  uint8  
 21  dum_educational        396030 non-null  uint8  
 22  dum_home_improvement   396030 non-null  uint8  
 23  dum_house              396030 non-null  uint8  
 24  dum_major_purchase     396030 non-null  uint8  
 25  dum_medical            396030 non-null  uint8  
 26  dum_moving             396030 non-null  uint8  
 27  dum_other              396030 non-null  uint8  
 28  dum_renewable_energy   396030 non-null  uint8  
 29  dum_small_business     396030 non-null  uint8  
 30  dum_vacation           396030 non-null  uint8  
 31  dum_wedding            396030 non-null  uint8  
 32  dum_OTHER              396030 non-null  uint8  
 33  dum_OWN                396030 non-null  uint8  
 34  dum_RENT               396030 non-null  uint8  
 35  dum_Source Verified    396030 non-null  uint8  
 36  dum_Verified           396030 non-null  uint8  
 37  loan_year              396030 non-null  uint16 
 38  dum_loan_Aug           396030 non-null  uint8  
 39  dum_loan_Dec           396030 non-null  uint8  
 40  dum_loan_Feb           396030 non-null  uint8  
 41  dum_loan_Jan           396030 non-null  uint8  
 42  dum_loan_Jul           396030 non-null  uint8  
 43  dum_loan_Jun           396030 non-null  uint8  
 44  dum_loan_Mar           396030 non-null  uint8  
 45  dum_loan_May           396030 non-null  uint8  
 46  dum_loan_Nov           396030 non-null  uint8  
 47  dum_loan_Oct           396030 non-null  uint8  
 48  dum_loan_Sep           396030 non-null  uint8  
 49  earliest_cr_year       396030 non-null  uint16 
 50  dum_ear_cr_Aug         396030 non-null  uint8  
 51  dum_ear_cr_Dec         396030 non-null  uint8  
 52  dum_ear_cr_Feb         396030 non-null  uint8  
 53  dum_ear_cr_Jan         396030 non-null  uint8  
 54  dum_ear_cr_Jul         396030 non-null  uint8  
```

```
 55  dum_ear_cr_Jun         396030 non-null  uint8
 56  dum_ear_cr_Mar         396030 non-null  uint8
 57  dum_ear_cr_May         396030 non-null  uint8
 58  dum_ear_cr_Nov         396030 non-null  uint8
 59  dum_ear_cr_Oct         396030 non-null  uint8
 60  dum_ear_cr_Sep         396030 non-null  uint8
 61  dum_w                  396030 non-null  uint8
 62  dum_zip_05113          396030 non-null  uint8
 63  dum_zip_11650          396030 non-null  uint8
 64  dum_zip_22690          396030 non-null  uint8
 65  dum_zip_29597          396030 non-null  uint8
 66  dum_zip_30723          396030 non-null  uint8
 67  dum_zip_48052          396030 non-null  uint8
 68  dum_zip_70466          396030 non-null  uint8
 69  dum_zip_86630          396030 non-null  uint8
 70  dum_zip_93700          396030 non-null  uint8
dtypes: float64(14), int64(1), object(1), uint16(2), uint8(53)
memory usage: 69.9+ MB
```

3. Missing Data

In [73]:

```
# Number of missing values (NaNs)
lc_loan.isna().sum().sort_values()
```

Out[73]:

```
loan_amnt                   0
dum_ear_cr_Aug              0
earliest_cr_year            0
dum_loan_Sep                0
dum_loan_Oct                0
                        ...
dum_zip_93700               0
revol_util                276
pub_rec_bankruptcies      535
emp_length              18301
mort_acc                37795
Length: 71, dtype: int64
```

In [74]:

```
# Missing values ratio
lc_loan.isna().sum().sort_values()/len(lc_loan)*100
```

Out[74]:

```
loan_amnt               0.000000
dum_ear_cr_Aug          0.000000
earliest_cr_year        0.000000
dum_loan_Sep            0.000000
dum_loan_Oct            0.000000
                          ...
dum_zip_93700           0.000000
revol_util              0.069692
pub_rec_bankruptcies    0.135091
emp_length              4.621115
mort_acc                9.543469
Length: 71, dtype: float64
```

As the number of NaNs in "revol_util" and "pub_rec_bankruptcies" is relatively low, we can just drop the rows that contain those NaNs and fill the missing data in the other two columns

In [75]:
```
lc_loan.drop(lc_loan[lc_loan['revol_util'].isna()].index, inplace=True)
lc_loan.drop(lc_loan[lc_loan['pub_rec_bankruptcies'].isna()].index, inplace=True)
lc_loan.isna().sum().sort_values()/len(lc_loan)*100
```

Out[75]:
```
loan_amnt                 0.000000
dum_ear_cr_Aug            0.000000
earliest_cr_year          0.000000
dum_loan_Sep              0.000000
dum_loan_Oct              0.000000
                            ...
dum_debt_consolidation    0.000000
dum_medical               0.000000
dum_zip_93700             0.000000
emp_length                4.627814
mort_acc                  9.413768
Length: 71, dtype: float64
```

4. Scaling the data

In [76]:
```
lc_loan.drop(('loan_status'), axis=1, inplace=True)
lc_loan.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 395219 entries, 0 to 396029
Data columns (total 70 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   loan_amnt               395219 non-null  float64
 1   term                    395219 non-null  int64
 2   int_rate                395219 non-null  float64
 3   installment             395219 non-null  float64
 4   sub_grade               395219 non-null  float64
 5   emp_length              376929 non-null  float64
 6   annual_inc              395219 non-null  float64
 7   dti                     395219 non-null  float64
 8   open_acc                395219 non-null  float64
 9   pub_rec                 395219 non-null  float64
 10  revol_bal               395219 non-null  float64
 11  revol_util              395219 non-null  float64
 12  total_acc               395219 non-null  float64
 13  mort_acc                358014 non-null  float64
 14  pub_rec_bankruptcies    395219 non-null  float64
 15  status_dummy            395219 non-null  uint8
 16  Individual              395219 non-null  uint8
 17  Joint                   395219 non-null  uint8
 18  dum_credit_card         395219 non-null  uint8
 19  dum_debt_consolidation  395219 non-null  uint8
 20  dum_educational         395219 non-null  uint8
 21  dum_home_improvement    395219 non-null  uint8
 22  dum_house               395219 non-null  uint8
 23  dum_major_purchase      395219 non-null  uint8
 24  dum_medical             395219 non-null  uint8
 25  dum_moving              395219 non-null  uint8
 26  dum_other               395219 non-null  uint8
 27  dum_renewable_energy    395219 non-null  uint8
 28  dum_small_business      395219 non-null  uint8
 29  dum_vacation            395219 non-null  uint8
 30  dum_wedding             395219 non-null  uint8
 31  dum_OTHER               395219 non-null  uint8
 32  dum_OWN                 395219 non-null  uint8
 33  dum_RENT                395219 non-null  uint8
 34  dum_Source Verified     395219 non-null  uint8
 35  dum_Verified            395219 non-null  uint8
 36  loan_year               395219 non-null  uint16
 37  dum_loan_Aug            395219 non-null  uint8
 38  dum_loan_Dec            395219 non-null  uint8
 39  dum_loan_Feb            395219 non-null  uint8
 40  dum_loan_Jan            395219 non-null  uint8
 41  dum_loan_Jul            395219 non-null  uint8
 42  dum_loan_Jun            395219 non-null  uint8
 43  dum_loan_Mar            395219 non-null  uint8
 44  dum_loan_May            395219 non-null  uint8
 45  dum_loan_Nov            395219 non-null  uint8
 46  dum_loan_Oct            395219 non-null  uint8
 47  dum_loan_Sep            395219 non-null  uint8
 48  earliest_cr_year        395219 non-null  uint16
 49  dum_ear_cr_Aug          395219 non-null  uint8
 50  dum_ear_cr_Dec          395219 non-null  uint8
 51  dum_ear_cr_Feb          395219 non-null  uint8
 52  dum_ear_cr_Jan          395219 non-null  uint8
 53  dum_ear_cr_Jul          395219 non-null  uint8
 54  dum_ear_cr_Jun          395219 non-null  uint8
```

```
 55  dum_ear_cr_Mar          395219 non-null  uint8
 56  dum_ear_cr_May          395219 non-null  uint8
 57  dum_ear_cr_Nov          395219 non-null  uint8
 58  dum_ear_cr_Oct          395219 non-null  uint8
 59  dum_ear_cr_Sep          395219 non-null  uint8
 60  dum_w                   395219 non-null  uint8
 61  dum_zip_05113           395219 non-null  uint8
 62  dum_zip_11650           395219 non-null  uint8
 63  dum_zip_22690           395219 non-null  uint8
 64  dum_zip_29597           395219 non-null  uint8
 65  dum_zip_30723           395219 non-null  uint8
 66  dum_zip_48052           395219 non-null  uint8
 67  dum_zip_70466           395219 non-null  uint8
 68  dum_zip_86630           395219 non-null  uint8
 69  dum_zip_93700           395219 non-null  uint8
dtypes: float64(14), int64(1), uint16(2), uint8(53)
memory usage: 69.7 MB
```

In [77]:

```python
# Create a dataframe without the dummy columns to scale just those variables

# use numpy r_ to concatenate slices
dumms = lc_loan.iloc[:,np.r_[16:36, 37:48, 49:70, 15]]
noDum = lc_loan.iloc[:,np.r_[:15, 36, 48]]
```

In [78]:

```python
# We should scale the data as KNN is a distance based algorithm and it reduces biases

# Instanciate the scaler model and scale the data
scaler = MinMaxScaler()
scaled_noDum = pd.DataFrame(scaler.fit_transform(noDum), columns = noDum.columns)
```

In [79]:

```python
scaled_noDum.describe()
```

Out[79]:

	loan_amnt	term	int_rate	installment	sub_grade	emp_length	
count	395219.000000	395219.000000	395219.000000	395219.000000	395219.000000	376929.000000	39
mean	0.344862	0.237772	0.324195	0.274086	0.325956	0.594187	
std	0.211571	0.425719	0.174248	0.165181	0.194124	0.364464	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.189873	0.000000	0.201402	0.154342	0.176471	0.300000	
50%	0.291139	0.000000	0.312037	0.236808	0.294118	0.600000	
75%	0.493671	0.000000	0.437476	0.363510	0.441176	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

It looks good.

In [80]:

```python
# Add the dummy columns
scaled_df = scaled_noDum.join(dumms, on=dumms.index)
scaled_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 395219 entries, 0 to 395218
Data columns (total 70 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   loan_amnt             395219 non-null  float64
 1   term                  395219 non-null  float64
 2   int_rate              395219 non-null  float64
 3   installment           395219 non-null  float64
 4   sub_grade             395219 non-null  float64
 5   emp_length            376929 non-null  float64
 6   annual_inc            395219 non-null  float64
 7   dti                   395219 non-null  float64
 8   open_acc              395219 non-null  float64
 9   pub_rec               395219 non-null  float64
 10  revol_bal             395219 non-null  float64
 11  revol_util            395219 non-null  float64
 12  total_acc             395219 non-null  float64
 13  mort_acc              358014 non-null  float64
 14  pub_rec_bankruptcies  395219 non-null  float64
 15  loan_year             395219 non-null  float64
 16  earliest_cr_year      395219 non-null  float64
 17  Individual            395219 non-null  uint8  
 18  Joint                 395219 non-null  uint8  
 19  dum_credit_card       395219 non-null  uint8  
 20  dum_debt_consolidation  395219 non-null  uint8  
 21  dum_educational       395219 non-null  uint8  
 22  dum_home_improvement  395219 non-null  uint8  
 23  dum_house             395219 non-null  uint8  
 24  dum_major_purchase    395219 non-null  uint8  
 25  dum_medical           395219 non-null  uint8  
 26  dum_moving            395219 non-null  uint8  
 27  dum_other             395219 non-null  uint8  
 28  dum_renewable_energy  395219 non-null  uint8  
 29  dum_small_business    395219 non-null  uint8  
 30  dum_vacation          395219 non-null  uint8  
 31  dum_wedding           395219 non-null  uint8  
 32  dum_OTHER             395219 non-null  uint8  
 33  dum_OWN               395219 non-null  uint8  
 34  dum_RENT              395219 non-null  uint8  
 35  dum_Source Verified   395219 non-null  uint8  
 36  dum_Verified          395219 non-null  uint8  
 37  dum_loan_Aug          395219 non-null  uint8  
 38  dum_loan_Dec          395219 non-null  uint8  
 39  dum_loan_Feb          395219 non-null  uint8  
 40  dum_loan_Jan          395219 non-null  uint8  
 41  dum_loan_Jul          395219 non-null  uint8  
 42  dum_loan_Jun          395219 non-null  uint8  
 43  dum_loan_Mar          395219 non-null  uint8  
 44  dum_loan_May          395219 non-null  uint8  
 45  dum_loan_Nov          395219 non-null  uint8  
 46  dum_loan_Oct          395219 non-null  uint8  
 47  dum_loan_Sep          395219 non-null  uint8  
 48  dum_ear_cr_Aug        395219 non-null  uint8  
 49  dum_ear_cr_Dec        395219 non-null  uint8  
 50  dum_ear_cr_Feb        395219 non-null  uint8  
 51  dum_ear_cr_Jan        395219 non-null  uint8  
 52  dum_ear_cr_Jul        395219 non-null  uint8  
 53  dum_ear_cr_Jun        395219 non-null  uint8  
 54  dum_ear_cr_Mar        395219 non-null  uint8  
```

```
 55  dum_ear_cr_May          395219 non-null  uint8
 56  dum_ear_cr_Nov          395219 non-null  uint8
 57  dum_ear_cr_Oct          395219 non-null  uint8
 58  dum_ear_cr_Sep          395219 non-null  uint8
 59  dum_w                   395219 non-null  uint8
 60  dum_zip_05113           395219 non-null  uint8
 61  dum_zip_11650           395219 non-null  uint8
 62  dum_zip_22690           395219 non-null  uint8
 63  dum_zip_29597           395219 non-null  uint8
 64  dum_zip_30723           395219 non-null  uint8
 65  dum_zip_48052           395219 non-null  uint8
 66  dum_zip_70466           395219 non-null  uint8
 67  dum_zip_86630           395219 non-null  uint8
 68  dum_zip_93700           395219 non-null  uint8
 69  status_dummy            395219 non-null  uint8
dtypes: float64(17), uint8(53)
memory usage: 71.2 MB
```

5. Neural Network

In [81]:
```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)
```

In [82]:
```
scaled_df.shape # Width of 48 neurons, as the last one is the target
```

Out[82]:
```
(395219, 70)
```

In [83]:
```
# Setting up the Neural Network

model = Sequential()

model.add(Dense(units=70, activation='relu'))   # Input layer
model.add(Dropout(0.2))
model.add(Dense(units=35, activation='relu'))   # Hidden Layer 1
model.add(Dropout(0.2))
model.add(Dense(units=12, activation='relu'))   # Hidden Layer 2
model.add(Dropout(0.2))
model.add(Dense(units=1, activation='sigmoid')) # Output layer

# For a binary classification problem : binary_crossentropy
model.compile(loss='binary_crossentropy', optimizer='adam')
```

5.1. Model training/testing

After trying many runs with different NN model hyperparameters and number of neighbors for the imputation (in which the results didn't change too much), the best result was achieved with the following setup

In [84]:
```
imputer = KNNImputer(n_neighbors=2)
imputed_df = imputer.fit_transform(scaled_df)  # This Outputs an array

# Neural networks work with arrays
y = imputed_df[:,69]                           # status_dummy
X = imputed_df[:,:69]                          # rest of columns

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state
```

```
model.fit(x=X_train,
          y=y_train,
          epochs=150,                  # number times that the learning algorithm will
          batch_size=256,              # number of samples processed before the model
          validation_data=(X_test, y_test), verbose=1,
          callbacks=[early_stop]
          )
```

```
Epoch 1/150
1158/1158 [==============================] - 3s 2ms/step - loss: 0.3145 - val_loss: 0.2685
Epoch 2/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2674 - val_loss: 0.2619
Epoch 3/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2634 - val_loss: 0.2611
Epoch 4/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2618 - val_loss: 0.2612
Epoch 5/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2609 - val_loss: 0.2608
Epoch 6/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2598 - val_loss: 0.2602
Epoch 7/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2593 - val_loss: 0.2609
Epoch 8/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2589 - val_loss: 0.2608
Epoch 9/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2589 - val_loss: 0.2606
Epoch 10/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2583 - val_loss: 0.2623
Epoch 11/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2580 - val_loss: 0.2598
Epoch 12/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2579 - val_loss: 0.2598
Epoch 13/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2573 - val_loss: 0.2598
Epoch 14/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2569 - val_loss: 0.2598
Epoch 15/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2566 - val_loss: 0.2601
Epoch 16/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2566 - val_loss: 0.2597
Epoch 17/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2564 - val_loss: 0.2596
Epoch 18/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2560 - val_loss: 0.2600
Epoch 19/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2558 - val_loss: 0.2594
Epoch 20/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2554 - val_loss: 0.2592
```

```
Epoch 21/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2553 - val_loss:
0.2597
Epoch 22/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2551 - val_loss:
0.2594
Epoch 23/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2546 - val_loss:
0.2596
Epoch 24/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2548 - val_loss:
0.2593
Epoch 25/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2545 - val_loss:
0.2597
Epoch 26/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2545 - val_loss:
0.2596
Epoch 27/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2541 - val_loss:
0.2602
Epoch 28/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2541 - val_loss:
0.2593
Epoch 29/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2540 - val_loss:
0.2597
Epoch 30/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2537 - val_loss:
0.2603
Epoch 31/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2537 - val_loss:
0.2599
Epoch 32/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2533 - val_loss:
0.2592
Epoch 33/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2535 - val_loss:
0.2593
Epoch 34/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2531 - val_loss:
0.2591
Epoch 35/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2530 - val_loss:
0.2605
Epoch 36/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2528 - val_loss:
0.2591
Epoch 37/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2531 - val_loss:
0.2594
Epoch 38/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2527 - val_loss:
0.2595
Epoch 39/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2528 - val_loss:
0.2591
Epoch 40/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2530 - val_loss:
0.2590
```

```
Epoch 41/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2526 - val_loss:
0.2592
Epoch 42/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2523 - val_loss:
0.2602
Epoch 43/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2521 - val_loss:
0.2591
Epoch 44/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2520 - val_loss:
0.2613
Epoch 45/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2523 - val_loss:
0.2593
Epoch 46/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2521 - val_loss:
0.2590
Epoch 47/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2516 - val_loss:
0.2601
Epoch 48/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2518 - val_loss:
0.2596
Epoch 49/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2517 - val_loss:
0.2592
Epoch 50/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2516 - val_loss:
0.2594
Epoch 51/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2516 - val_loss:
0.2598
Epoch 52/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2515 - val_loss:
0.2597
Epoch 53/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2516 - val_loss:
0.2607
Epoch 54/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2514 - val_loss:
0.2597
Epoch 55/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2512 - val_loss:
0.2598
Epoch 56/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2512 - val_loss:
0.2592
Epoch 57/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2511 - val_loss:
0.2600
Epoch 58/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2513 - val_loss:
0.2596
Epoch 59/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2513 - val_loss:
0.2594
Epoch 60/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2511 - val_loss:
0.2620
```

```
Epoch 61/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2511 - val_loss:
0.2594
Epoch 62/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2507 - val_loss:
0.2595
Epoch 63/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2509 - val_loss:
0.2595
Epoch 64/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2509 - val_loss:
0.2596
Epoch 65/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2505 - val_loss:
0.2594
Epoch 66/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2509 - val_loss:
0.2593
Epoch 67/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2506 - val_loss:
0.2594
Epoch 68/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2508 - val_loss:
0.2594
Epoch 69/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2506 - val_loss:
0.2595
Epoch 70/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2503 - val_loss:
0.2590
Epoch 71/150
1158/1158 [==============================] - 2s 2ms/step - loss: 0.2503 - val_loss:
0.2593
Epoch 71: early stopping
```

Out[84]:

```
<keras.callbacks.History at 0x23133b0eda0>
```

Model performance

In [90]:

```
# Create global variables containing the predictions
pred2 = (model.predict(X_test) > 0.5).astype("int32")

print(classification_report(y_test.astype(int),pred2))
```

```
              precision    recall  f1-score   support

           0       0.92      0.48      0.63     19458
           1       0.88      0.99      0.93     79347

    accuracy                           0.89     98805
   macro avg       0.90      0.73      0.78     98805
weighted avg       0.89      0.89      0.87     98805
```

It would be better that the recall for "0" (charged-off loans) was higher, as this is the model capability to detect all the loans of this kind. Despite that, this is a normal result, given the big quantity difference between fully-paid and charged-off loans present in the data.