CSPE65: MACHINE LEARNING TECHNIQUES AND PRACTICES

Assignment - 1

106119100 - Rajneesh Pandey

Code is runned on Kaggle Notebook

Data Preprocessing steps to make it fit for applying machine learning algorithms.

1. Importing Libraries for Data Preprocessing

[45]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


# 'Numpy' is used for mathematical operations on large, multi-dimensional arrays and matrices
# 'Pandas' is used for data manipulation and analysis
# 'Matplotlib' is a data visualization library for 2D and 3D plots, built on numpy
# 'Seaborn' is based on matplotlib; used for plotting statistical graphics

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline

import warnings
warnings.filterwarnings("ignore")
```

Reading the data and viewing them

[46]:
```
df = pd.read_csv("../input/ml-dataset/lending-club-loans.csv", low_memory=False) #Dataset
```

[47]:
```
# Checking the Dimensions of our dataset

df.shape
```

[47…
```
(42553, 115)
```

[48]:
```
df.columns
```

[48…
```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       ...
       'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'pct_tl_nvr_dlq',
       'percent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens',
       'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit'],
      dtype='object', length=115)
```

To get familiarize ourselves with this dataset, and understand what these columns represent.

[90]:
```
description = pd.read_csv('../input/lcdata/LCDataDictionary.csv').dropna()
description.style.set_properties(subset=['Description'], **{'width' :'850px'})
```

[90…

	LoanStatNew	Description
0	acc_now_delinq	The number of accounts on which the borrower is now delinquent.
1	acc_open_past_24mths	Number of trades opened in past 24 months.
2	addr_state	The state provided by the borrower in the loan application
3	all_util	Balance to credit limit on all trades
4	annual_inc	The self-reported annual income provided by the borrower during registration.
5	annual_inc_joint	The combined self-reported annual income provided by the co-borrowers during registration
6	application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
7	avg_cur_bal	Average current balance of all accounts
8	bc_open_to_buy	Total open to buy on revolving bankcards.
9	bc_util	Ratio of total current balance to high credit/credit limit for all bankcard accounts.
10	chargeoff_within_12_mths	Number of charge-offs within 12 months
11	collection_recovery_fee	post charge off collection fee
12	collections_12_mths_ex_med	Number of collections in 12 months excluding medical collections
13	delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
14	delinq_amnt	The past-due amount owed for the accounts on which the borrower is now delinquent.

[91]:
```
df.info()
```

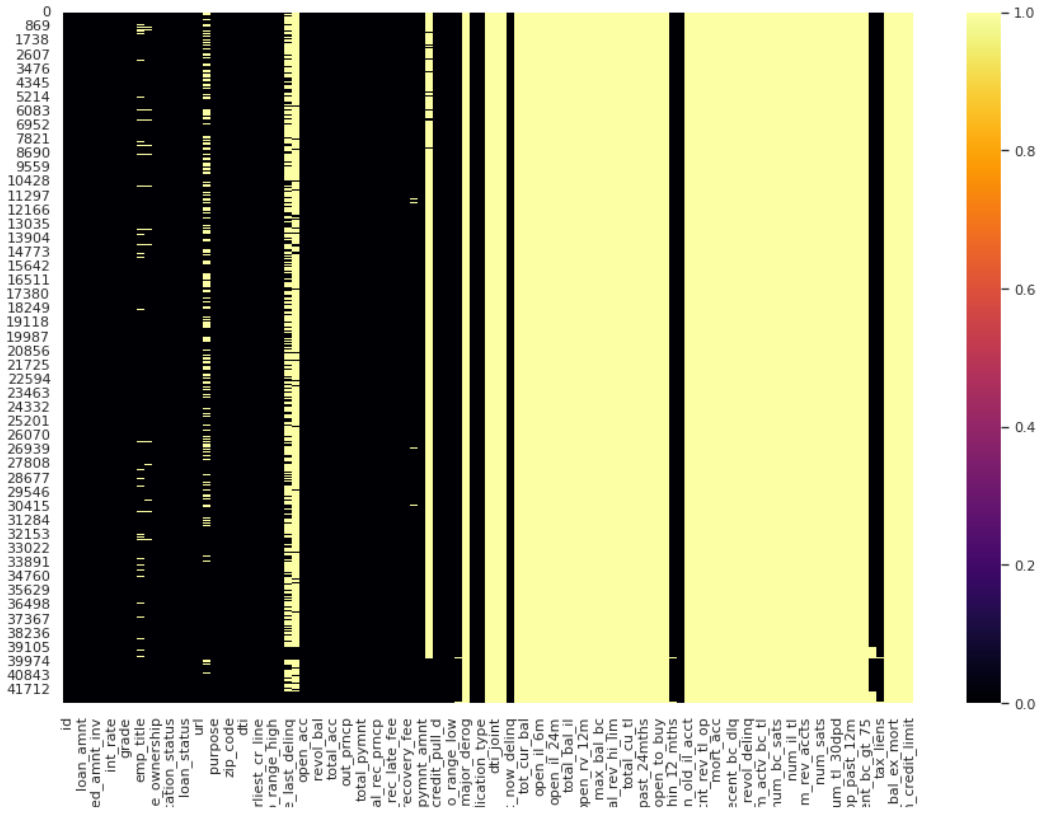
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42553 entries, 0 to 42552
Columns: 115 entries, id to total_il_high_credit_limit
dtypes: float64(86), object(29)
memory usage: 37.3+ MB
```

It is evident from the above heatmap that our dataset contains a lot of missing values and we can not use feature that has so many missing values.

More Darker Color represent the data is filled.

```
[92]: fig = plt.figure(figsize=(15,10))
      sns.heatmap(df.isna(),cmap='inferno')
```

```
[92… <AxesSubplot:>
```



Another thing I want to examine is that how many loans have a default loan status in comparison to other loans.

So,
A common thing to predict in datasets like these are if a new loan will get default or not.
I will be keeping loans with default status as my target variable.

[93]:
```
df['loan_status'].value_counts()
```

[93…
```
Fully Paid                                             33594
Charged Off                                             5657
Does not meet the credit policy. Status:Fully Paid      1990
Does not meet the credit policy. Status:Charged Off      762
Current                                                  513
In Grace Period                                           16
Late (31-120 days)                                        12
Late (16-30 days)                                          5
Default                                                    1
Name: loan_status, dtype: int64
```

Target feature

[94]:
```
target = [1 if i=='Default' else 0 for i in df['loan_status']]
df['target'] = target
df['target'].value_counts()
```

[94…
```
0    42552
1        1
Name: target, dtype: int64
```

+ Code + Markdown

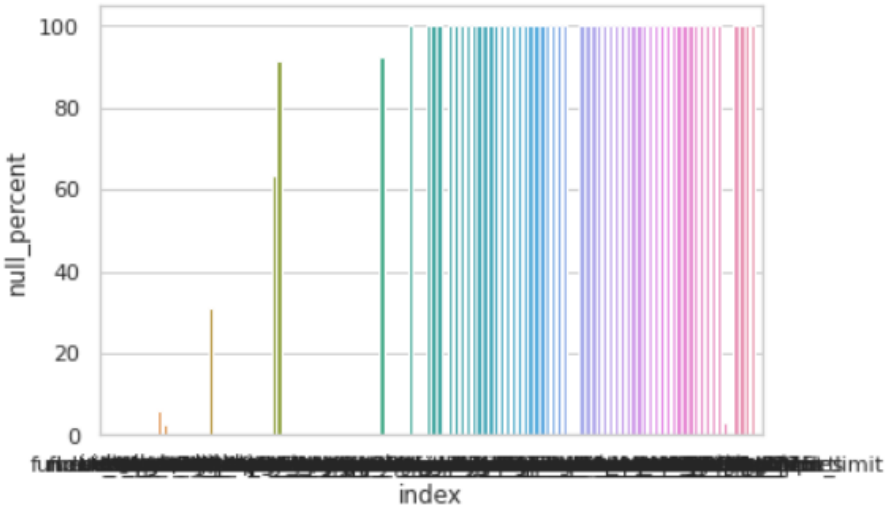
Finding the null percent to see the null values in the data frame

[95]:
```
nulls = pd.DataFrame(round(df.isnull().sum()/len(df.index)*100,2),columns=['null_percent'])
sns.barplot(x='index',y='null_percent',data=nulls.reset_index())
nulls[nulls['null_percent']!=0.00].sort_values('null_percent',ascending=False)
```

[95…

	null_percent
dti_joint	100.00
mo_sin_rcnt_rev_tl_op	100.00
mo_sin_old_il_acct	100.00
bc_util	100.00
bc_open_to_buy	100.00
...	...
application_type	0.01
fico_range_high	0.01
fico_range_low	0.01
dti	0.01
member_id	0.01

114 rows × 1 columns



There are several columns that fit into one of the following categories:

Unnecessary data: URL to view the loan.
Redundant data: Loan description may be useful to some, but loan purpose fits our needs
User provided information: Employer titles may offer some insight into employment industry.
Operational data: The next payment date for the loan at the time the data was generated is not relevant to us

96]:

```
# Drop unneccesary columns
df = df.drop(['url', 'desc', 'policy_code', 'last_pymnt_d', 'next_pymnt_d', 'earliest_cr_line', 'emp_title'], axis=1)
df = df.drop(['id', 'title', 'total_rec_int', 'total_rec_late_fee', 'total_rec_prncp', 'zip_code'], axis=1)
```

Since customer has taken loan again, we cannot drop member id.

[97]:
```
df['member_id'].value_counts().head(5)
```

[97…
```
505512.0     2
1064904.0    2
856744.0     2
730106.0     2
598172.0     2
Name: member_id, dtype: int64
```

Some rows that we decided were not relevant to our needs.
The 'loan_status' column is the source of answer to the main question that if people are paying the loans they take out.

Some records with a loan_status of “Does not meet the credit policy". These may be older loans that would simply not be accepted under LendingClubs current criteria.
As these data points will provide no value moving forward, I have excluded them from our data.

Similiarily, recently issued loans could mislead our analysis, as no payment has been expected yet.

[98]:
```
i = len(df)
df = pd.DataFrame(df[df['loan_status'] != "Does not meet the credit policy. Status:Fully Paid"])
df = pd.DataFrame(df[df['loan_status'] != "Does not meet the credit policy. Status:Charged Off"])
df = pd.DataFrame(df[df['loan_status'] != "Issued"])
df = pd.DataFrame(df[df['loan_status'] != "In Grace Period"])
a = len(df)
print(f"I Dropped {i-a} rows, a {((i-a)/((a+i)/2))*100}% reduction in rows")
```

```
I Dropped 2768 rows, a 6.723505550292696% reduction in rows
```

+ Code + Markdown

Converting the data to proper data type

Data Type Handling

[99]:
```
# formating int_rate and term to proper datatypes
df["int_rate"] =df["int_rate"].str.rstrip('%').astype('float')
df["term"]=df["term"].str.rstrip(' months').astype('float')
```

Now again viewing the data

Exploring the Data

Looking the Distribution of Data types

[100]:
```
df.dtypes
```

[10…
```
member_id                    float64
loan_amnt                    float64
funded_amnt                  float64
funded_amnt_inv              float64
term                         float64
                              ...
tot_hi_cred_lim              float64
total_bal_ex_mort            float64
total_bc_limit               float64
total_il_high_credit_limit   float64
target                         int64
Length: 103, dtype: object
```

Let us see how many Object type features are actually Categorical

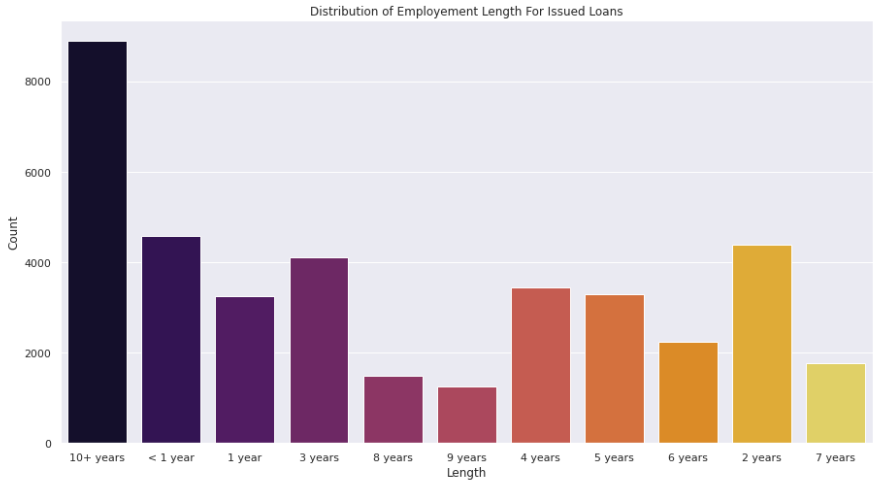
```
# Let us see how many Object type features are actually Categorical
df.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

[10…
```
grade                        7
sub_grade                   35
emp_length                  11
home_ownership               5
verification_status          3
issue_d                     55
loan_status                  6
pymnt_plan                   2
purpose                     14
addr_state                  50
initial_list_status          1
last_credit_pull_d         110
collections_12_mths_ex_med   1
application_type             1
acc_now_delinq               1
chargeoff_within_12_mths     1
tax_liens                    1
dtype: int64
```

It can be seen that people who have worked for 10 or more years are more likely to take loans

The distribution of Employement Lengths.

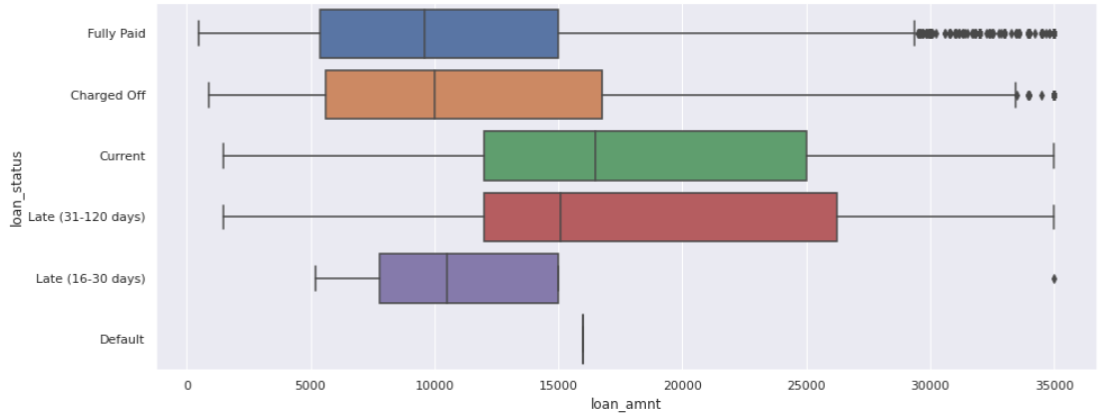
[103]:
```
sns.set(rc={'figure.figsize':(15,8)})
sns.countplot(df['emp_length'],palette='inferno')
plt.xlabel("Length")
plt.ylabel("Count")
plt.title("Distribution of Employement Length For Issued Loans")
plt.show()
```



Plotting the box plots to see the outliers

[104]:
```
sns.set(rc={'figure.figsize':(15,6)})
sns.boxplot(x='loan_amnt', y='loan_status', data=df)
```

[10… <AxesSubplot:xlabel='loan_amnt', ylabel='loan_status'>



+ Code + Markdown

Cleaning The Data

[105]:
```
df.columns
```

[10…
```
Index(['member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term',
       'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       ...
       'num_tl_op_past_12m', 'pct_tl_nvr_dlq', 'percent_bc_gt_75',
       'pub_rec_bankruptcies', 'tax_liens', 'tot_hi_cred_lim',
       'total_bal_ex_mort', 'total_bc_limit', 'total_il_high_credit_limit',
       'target'],
      dtype='object', length=103)
```

There are some columns/features that are not required. I have already dropped those features.

+ Code + Markdown

[106]:
```
df.shape
```

[10…
```
(39785, 103)
```

[107]:
```
df.head(5)
```

[10…

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_length	...	num_tl_op_past_12m	pct_tl_nvr_dlq	percent_bc_gt_75
0	1296599.0	5000.0	5000.0	4975.0	36.0	10.65	162.87	B	B2	10+ years	...	NaN	NaN	NaN
1	1314167.0	2500.0	2500.0	2500.0	60.0	15.27	59.83	C	C4	< 1 year	...	NaN	NaN	NaN
2	1313524.0	2400.0	2400.0	2400.0	36.0	15.96	84.33	C	C5	10+ years	...	NaN	NaN	NaN
3	1277178.0	10000.0	10000.0	10000.0	36.0	13.49	339.31	C	C1	10+ years	...	NaN	NaN	NaN
4	1311748.0	3000.0	3000.0	3000.0	60.0	12.69	67.79	B	B5	1 year	...	NaN	NaN	NaN

5 rows × 103 columns

Dropping the colunms with more than 75% null values

Drop columns that have more than 75% null values

[108]:
```
nulls = pd.DataFrame(round(df.isnull().sum()/len(df.index)*100,2),columns=['null_percent'])
drop_cols = nulls[nulls['null_percent']>75.0].index
df.drop(drop_cols, axis=1, inplace=True)
```

[109]:
```
df.shape
```

[10… (39785, 48)

[111]:
```
df.columns
```

[11…
```
Index(['member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'term',
       'int_rate', 'installment', 'grade', 'sub_grade', 'emp_length',
       'home_ownership', 'annual_inc', 'verification_status', 'issue_d',
       'loan_status', 'pymnt_plan', 'purpose', 'addr_state', 'dti',
       'delinq_2yrs', 'fico_range_low', 'fico_range_high', 'inq_last_6mths',
       'mths_since_last_delinq', 'open_acc', 'pub_rec', 'revol_bal',
       'revol_util', 'total_acc', 'initial_list_status', 'out_prncp',
       'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv', 'recoveries',
       'collection_recovery_fee', 'last_pymnt_amnt', 'last_credit_pull_d',
       'last_fico_range_high', 'last_fico_range_low',
       'collections_12_mths_ex_med', 'application_type', 'acc_now_delinq',
       'chargeoff_within_12_mths', 'delinq_amnt', 'pub_rec_bankruptcies',
       'tax_liens', 'target'],
      dtype='object')
```

Replacing the null values in the below column with median

:
```
df['mths_since_last_delinq'].fillna(df['mths_since_last_delinq'].median(), inplace=True)
```

Making the categorical Data item list

[113]:
```
categorical = []
for column in df:
    if df[column].dtype == 'object':
        categorical.append(column)
categorical
```

[11…
```
['grade',
 'sub_grade',
 'emp_length',
 'home_ownership',
 'verification_status',
 'issue_d',
 'pymnt_plan',
 'purpose',
 'addr_state',
 'initial_list_status',
 'last_credit_pull_d',
 'collections_12_mths_ex_med',
 'application_type',
 'acc_now_delinq',
 'chargeoff_within_12_mths',
 'tax_liens']
```

+ Code + Markdown

Defining the fuction for various task to perform on data

```
def remove_invalid_entries(data):
    # these indexes contains id as invalid type and having rest of the features null
    idxs= data[data['member_id'].isnull()].index.tolist()
    data= data.drop(idxs, axis=0)
    return data
```

This function is to remove invalid entries in the data

```
def remove_constant_featurues(data):
    cols= [col for col in df.columns if len(df[col].unique())==1]
    data = data.drop(cols, axis=1)
    return data
```

This function is to remove constant feature in the data

Remove duplicate column and duplicate row

```
df = df.loc[:,~df.T.duplicated(keep='first')]  # removing duplicate columns

df= remove_constant_featurues(df) # removing constant features
df.drop_duplicates(keep=False, inplace=True) # removing duplicate rows
```

Creating the mapping dictionary to convert the categorical data

[118]:
```
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},

 "loan_status": {
        "Fully Paid": 1, "Charged Off": 0
     }
}
```

+ Code + Markdown

[119]:
```
df["int_rate"]
```

[11…
```
0        10.65
1        15.27
2        15.96
3        13.49
4        12.69
         ...
39781     8.07
39782    10.28
39783     8.07
39784     7.43
39785    13.75
Name: int_rate, Length: 39758, dtype: float64
```

[120]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 39758 entries, 0 to 39785
Data columns (total 44 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   member_id                   39758 non-null  float64
 1   loan_amnt                   39758 non-null  float64
 2   funded_amnt                 39758 non-null  float64
 3   funded_amnt_inv             39758 non-null  float64
 4   term                        39758 non-null  float64
 5   int_rate                    39758 non-null  float64
 6   installment                 39758 non-null  float64
 7   grade                       39758 non-null  object 
 8   sub_grade                   39758 non-null  object 
 9   emp_length                  38681 non-null  object 
 10  home_ownership              39758 non-null  object 
 11  annual_inc                  39758 non-null  float64
 12  verification_status         39758 non-null  object 
 13  issue_d                     39758 non-null  object 
 14  pymnt_plan                  39758 non-null  object 
 15  purpose                     39758 non-null  object 
 16  addr_state                  39758 non-null  object 
 17  dti                         39758 non-null  float64
 18  delinq_2yrs                 39758 non-null  float64
 19  fico_range_low              39758 non-null  float64
 20  fico_range_high             39758 non-null  float64
 21  inq_last_6mths              39758 non-null  float64
 22  mths_since_last_delinq      39758 non-null  float64
 23  open_acc                    39758 non-null  float64
```

Function to handle the outlier that we have seen in the above

```
def handle_outliers(data, column):
    '''
    criteria for outliers
    count for outliers
    if count<1%
        remove entries

    else if 1%< count <15%
        subtitute with medians

    else
        log features
    '''
    Q1=df[column].quantile(0.25)
    Q3=df[column].quantile(0.75)
    IQR=Q3-Q1
    idxs= data[((data[column]<(Q1-1.5*IQR)) | (data[column]>(Q3+1.5*IQR)))].index.tolist()

    if(len(data[col].unique()) < 4 or len(idxs)==0):
        return data

    if(len(idxs)<(len(data)*0.01)):
        print(f"Removing {len(idxs)} entries in {col}")
        data= data.drop(idxs, axis=0)

    elif ((len(data)*0.01)< len(idxs) and len(idxs) <= (len(data)*0.10)):
        print(f"Substitued with Mean in {col}")
        data[column][idxs]= data[column].mean()

    elif len(idxs) > (len(data)*0.10):
        sns.displot(data, x= col)
        print(f"Log transformation is done in {col}")
        data[column] = np.log(data[column])
        sns.displot(data, x= col)

    return data
```

Creating num_feature and cat_feature for conversion and converting the categorical data

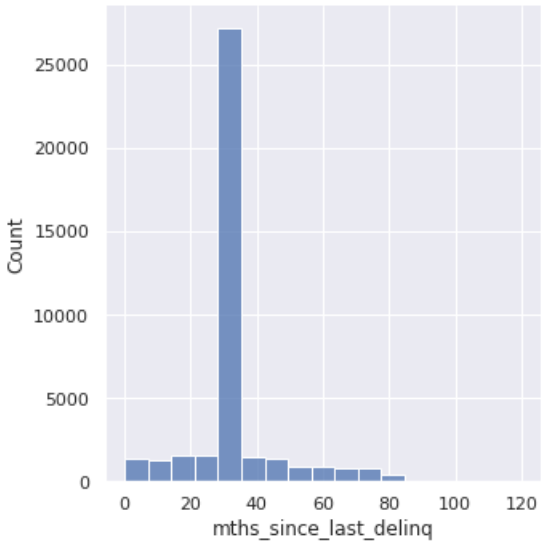
Also handling the outliers

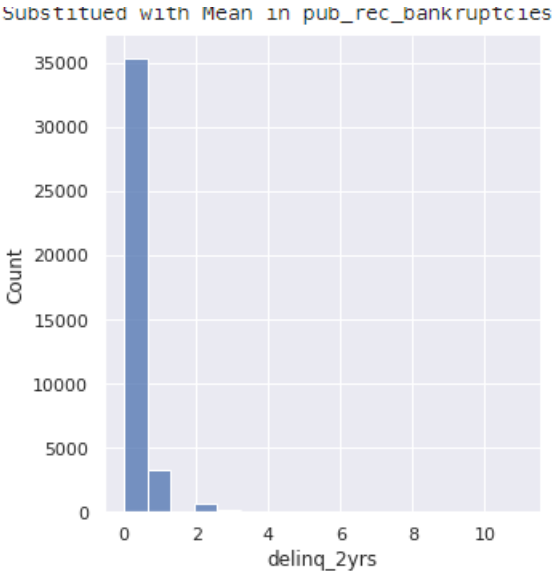
+ Code + Markdown

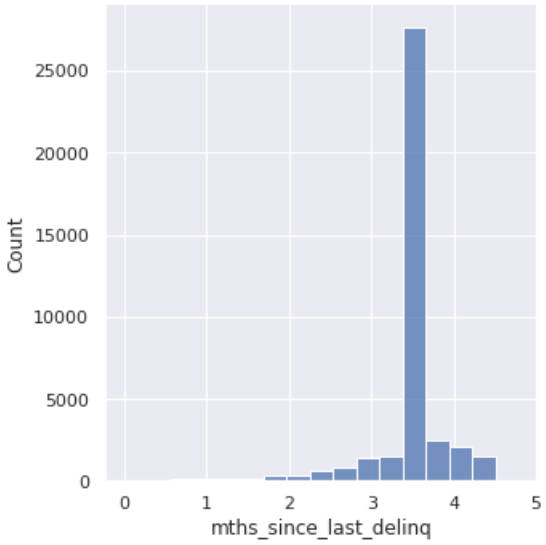
[121]:
```
num_features = [col for col in df.columns if df[col].dtypes != "object"]
cat_features = [col for col in df.columns if df[col].dtypes == "object"]
```

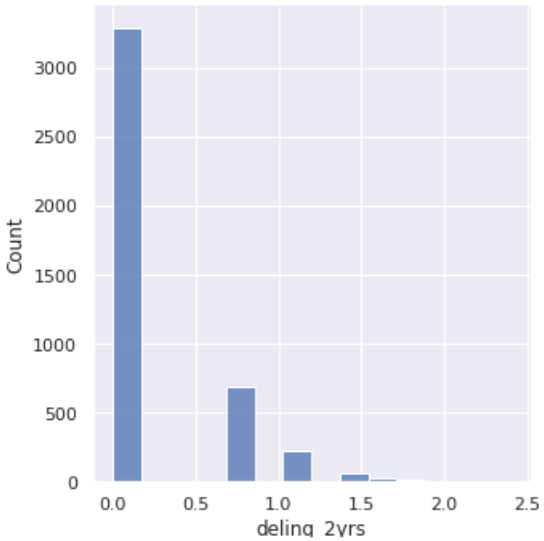
[122]:
```
# mapping considerate categorical features to their corresponding numerical values
df = df.replace(mapping_dict)

for col in num_features:

        df = handle_outliers(df, col)


for col in num_features:
    df[col].fillna(df[col].mean(), inplace= True) # filling null with mean in numerical_features

for col in cat_features:
    df[col].fillna(df[col].mode(), inplace= True)
```

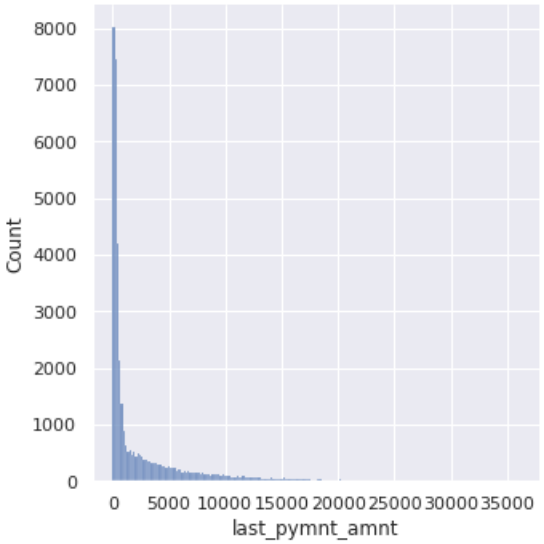
```
Removing 27 entries in member_id
Substitued with Mean in loan_amnt
Substitued with Mean in funded_amnt
Substitued with Mean in funded_amnt_inv
Removing 77 entries in int_rate
Substitued with Mean in installment
Substitued with Mean in annual_inc
Log transformation is done in delinq_2yrs
Removing 3 entries in fico_range_low
Substitued with Mean in inq_last_6mths
Log transformation is done in mths_since_last_delinq
Substitued with Mean in open_acc
Substitued with Mean in pub_rec
Substitued with Mean in revol_bal
Substitued with Mean in total_acc
Substitued with Mean in out_prncp
Substitued with Mean in out_prncp_inv
Substitued with Mean in total_pymnt
Substitued with Mean in total_pymnt_inv
Log transformation is done in recoveries
Substitued with Mean in collection_recovery_fee
Log transformation is done in last_pymnt_amnt
Removing 33 entries in last_fico_range_high
Substitued with Mean in last_fico_range_low
Substitued with Mean in pub_rec_bankruptcies
```

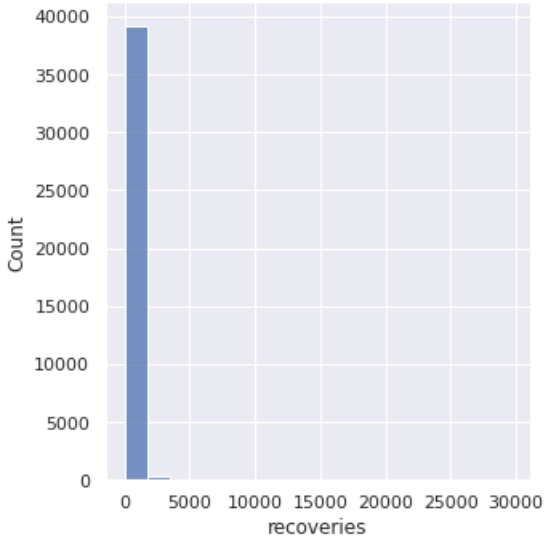

25000
20000
15000
10000
5000
0
Count
0
20
40
60
80
100
120
mths_since_last_delinq

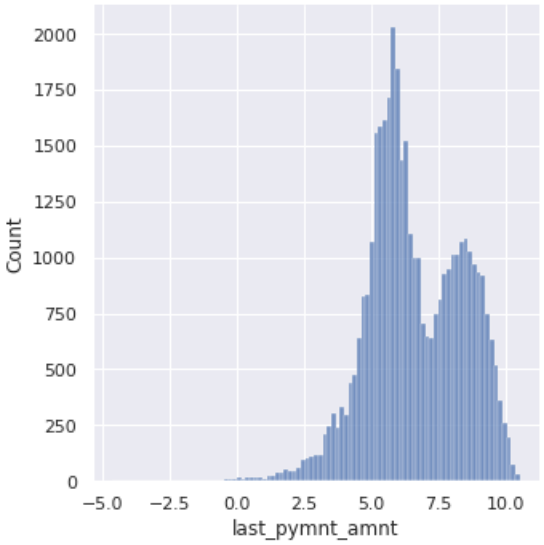

Substitued with Mean in pub_rec_bankruptcies
35000
30000
25000
20000
15000
10000
5000
0
Count
0
2
4
6
8
10
delinq_2yrs

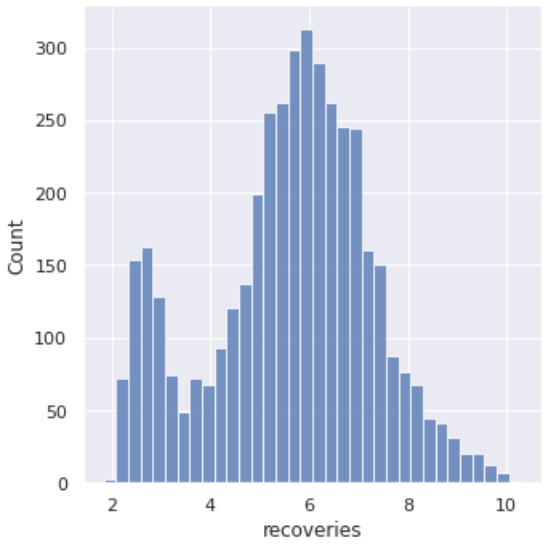

25000
20000
15000
10000
5000
0
Count
0
1
2
3
4
5
mths_since_last_delinq


3000
2500
2000
1500
1000
500
0
Count
0.0
0.5
1.0
1.5
2.0
2.5
delinq_2yrs


Count
last_pymnt_amnt


Count
recoveries


Count
last_pymnt_amnt


Count
recoveries