Purpose of this analysis

Purpose of this mini project is to experiment some of data wrangling techniques. We have rcieved Lending Club Loan Data set from 2007 to 2015. There are total 2.2 million rows and 145 columns. It is important to do some pre-processing work in order to analyze data and fill out missing values.

Packages

We will start by importing some of packages. Following packages will be imported.

In [17]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns


%matplotlib inline

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

sns.set_style('whitegrid')
```

Step 1 : Reading Data in Dataframe

Now we will be reading data from "Loan.csv" to Pandas Dataframe.

In [18]:
```
data_list = []

for chunk in  pd.read_csv('loan.csv', chunksize=200000, low_memory=False):
    data_list.append(chunk)

data = pd.concat(data_list, axis= 0)
del data_list

# data = pd.read_csv("loan.csv", low_memory=False)
```

Step 2 : Checking data size

Let's quickly check the shape of data.This will give us idea as how large is current dataset. As we can see, current data set contains more than 2 million rows and 145 columns. Not all columns are useful for our analysis.

In [19]:
```
# print(list(data.columns))
print(data.shape)
```

```
(2260668, 145)
```

Step 3 : Calculating Percentage of missing data per column

Here we will be creating Dataframe called "df_null". This Data frame consist of column name with percent of missing data in each column in descending order. It is noted that any column that has more than 60% of data missing are useless for our analysis and we will simply drop it. As we see some column like "url", "id", "member_id" has 100% missing values. Apart from that lot of columns have more than 80% missing data. We will simply drop column which has more than 60% data missing.

In [20]:
```
# print((data.isna().sum()[data.isna().sum() > 0]))
df_null = pd.DataFrame({'Count': data.isnull().sum(), 'Percent': 100*data.isnu
ll().sum()/len(data)})
print(df_null[df_null['Percent'] > 0].sort_values(by='Percent', ascending=Fals
e))



# print(data.head(3))
```

	Count	Percent
id	2260668	100.00
url	2260668	100.00
member_id	2260668	100.00
orig_projected_additional_accrued_interest	2252242	99.63
hardship_length	2250055	99.53
hardship_reason	2250055	99.53
hardship_status	2250055	99.53
deferral_term	2250055	99.53
hardship_amount	2250055	99.53
hardship_start_date	2250055	99.53
hardship_end_date	2250055	99.53
payment_plan_start_date	2250055	99.53
hardship_loan_status	2250055	99.53
hardship_dpd	2250055	99.53
hardship_payoff_balance_amount	2250055	99.53
hardship_last_payment_amount	2250055	99.53
hardship_type	2250055	99.53
debt_settlement_flag_date	2227612	98.54
settlement_status	2227612	98.54
settlement_date	2227612	98.54
settlement_amount	2227612	98.54
settlement_percentage	2227612	98.54
settlement_term	2227612	98.54
sec_app_mths_since_last_major_derog	2224726	98.41
sec_app_revol_util	2154484	95.30
revol_bal_joint	2152648	95.22
sec_app_open_acc	2152647	95.22
sec_app_chargeoff_within_12_mths	2152647	95.22
sec_app_open_act_il	2152647	95.22
sec_app_collections_12_mths_ex_med	2152647	95.22
sec_app_mort_acc	2152647	95.22
sec_app_inq_last_6mths	2152647	95.22
sec_app_earliest_cr_line	2152647	95.22
sec_app_num_rev_accts	2152647	95.22
verification_status_joint	2144938	94.88
dti_joint	2139962	94.66
annual_inc_joint	2139958	94.66
desc	2134601	94.42
mths_since_last_record	1901512	84.11
mths_since_recent_bc_dlq	1740967	77.01
mths_since_last_major_derog	1679893	74.31
mths_since_recent_revol_delinq	1520309	67.25
next_pymnt_d	1303607	57.66
mths_since_last_delinq	1158502	51.25
il_util	1068850	47.28
mths_since_rcnt_il	909924	40.25
all_util	866348	38.32
inq_last_12m	866130	38.31
total_cu_tl	866130	38.31
open_acc_6m	866130	38.31
open_il_24m	866129	38.31
open_act_il	866129	38.31
inq_fi	866129	38.31
max_bal_bc	866129	38.31
open_rv_24m	866129	38.31
open_rv_12m	866129	38.31

total_bal_il	866129	38.31
open_il_12m	866129	38.31
mths_since_recent_inq	295435	13.07
emp_title	166969	7.39
num_tl_120dpd_2m	153657	6.80
emp_length	146907	6.50
mo_sin_old_il_acct	139071	6.15
bc_util	76071	3.36
percent_bc_gt_75	75379	3.33
bc_open_to_buy	74935	3.31
mths_since_recent_bc	73412	3.25
pct_tl_nvr_dlq	70431	3.12
avg_cur_bal	70346	3.11
num_rev_accts	70277	3.11
mo_sin_rcnt_rev_tl_op	70277	3.11
mo_sin_old_rev_tl_op	70277	3.11
num_tl_90g_dpd_24m	70276	3.11
num_actv_rev_tl	70276	3.11
tot_coll_amt	70276	3.11
tot_cur_bal	70276	3.11
total_rev_hi_lim	70276	3.11
num_tl_op_past_12m	70276	3.11
num_op_rev_tl	70276	3.11
num_il_tl	70276	3.11
mo_sin_rcnt_tl	70276	3.11
num_accts_ever_120_pd	70276	3.11
num_actv_bc_tl	70276	3.11
num_rev_tl_bal_gt_0	70276	3.11
num_tl_30dpd	70276	3.11
tot_hi_cred_lim	70276	3.11
num_bc_tl	70276	3.11
total_il_high_credit_limit	70276	3.11
num_bc_sats	58590	2.59
num_sats	58590	2.59
acc_open_past_24mths	50030	2.21
mort_acc	50030	2.21
total_bc_limit	50030	2.21
total_bal_ex_mort	50030	2.21
title	23325	1.03
last_pymnt_d	2426	0.11
revol_util	1802	0.08
dti	1711	0.08
pub_rec_bankruptcies	1365	0.06
chargeoff_within_12_mths	145	0.01
collections_12_mths_ex_med	145	0.01
tax_liens	105	0.00
last_credit_pull_d	73	0.00
inq_last_6mths	30	0.00
open_acc	29	0.00
total_acc	29	0.00
earliest_cr_line	29	0.00
delinq_2yrs	29	0.00
delinq_amnt	29	0.00
acc_now_delinq	29	0.00
pub_rec	29	0.00
annual_inc	4	0.00
zip_code	1	0.00

Step 4 : Dropping column with more than 60% missing data

We will create a list of columns that a data missing more than 60%. We will simply drop those columns and create new dataframe called df_clean. We wil print shape of new dataframe to see how much data has been reduced. We can see total count of columns from 145 to reduced to 103.

In [21]:
```
# Creating list of column with more than 60% data missing.
list_60 = list(df_null[df_null['Percent']>60].index)

df_clean = data.drop(list_60, axis=1)
print(df_clean.shape)
```

```
(2260668, 103)
```

Step 5 : Analyzing columns with missing data more than 10% and 60%

Now that we have eliminated major columns that had missing values. Lets analyze remaining column with missing values. For that we will print all column that has missing data and see how we can predict missing values. We will predict all columns that has missing data grether than 10%. Since we already dropped columns that has missing data more than 60%, in our dataframe.

In [22]:
```
df_null = pd.DataFrame({'Count': df_clean.isnull().sum(), 'Percent': 100*df_cl
ean.isnull().sum()/len(df_clean)})
print(df_null[df_null['Percent'] > 10])
```

```
                          Count  Percent
mths_since_last_delinq  1158502    51.25
next_pymnt_d            1303607    57.66
open_acc_6m              866130    38.31
open_act_il              866129    38.31
open_il_12m              866129    38.31
open_il_24m              866129    38.31
mths_since_rcnt_il       909924    40.25
total_bal_il             866129    38.31
il_util                 1068850    47.28
open_rv_12m              866129    38.31
open_rv_24m              866129    38.31
max_bal_bc               866129    38.31
all_util                 866348    38.32
inq_fi                   866129    38.31
total_cu_tl              866130    38.31
inq_last_12m             866130    38.31
mths_since_recent_inq    295435    13.07
```

Step 5A : Checking data in columns with missing data more than 10%

Lets examine few values of this columns so we can get idea what kind of data estimation we need to do.

In [23]:
```
col_10 = list(df_null[df_null['Percent'] > 10].index)
print(df_clean[col_10].head(3))
```

```
   mths_since_last_delinq next_pymnt_d  open_acc_6m  open_act_il  open_il_12m
\
0                     nan     Mar-2019         2.00         2.00         1.00
1                   71.00     Mar-2019         4.00         4.00         2.00
2                     nan     Mar-2019         0.00         1.00         0.00

   open_il_24m  mths_since_rcnt_il  total_bal_il  il_util  open_rv_12m  \
0         2.00                2.00      12560.00    69.00         2.00
1         3.00                3.00      87153.00    88.00         4.00
2         2.00               14.00       7150.00    72.00         0.00

   open_rv_24m  max_bal_bc  all_util  inq_fi  total_cu_tl  inq_last_12m  \
0         7.00     2137.00     28.00    1.00        11.00          2.00
1         5.00      998.00     57.00    2.00        15.00          2.00
2         2.00        0.00     35.00    1.00         5.00          0.00

   mths_since_recent_inq
0                   2.00
1                   4.00
2                  14.00
```

Step 5B : Dropping payment dates columns

As we see above data set most missing data is numeric and we can use mode or median in order to fill missing values. By looking at dataset and name of columns, mode will be better option compared to median for most of columns. For "total_bil_il" we will use median. Lets analyze mode and median of those columns. But before we need to drop 3 columns which is not useful for our analysis. Those are 1) "next_pymnt_dt 2) 'last_credit_pull_d' 3) 'last_pymnt_d'

Lets create a list of these columns.

In [24]:
```
# print(df_clean.shape)
cols_to_drop = ['next_pymnt_d', 'last_credit_pull_d', 'last_pymnt_d']
df_clean.drop(cols_to_drop, axis=1, inplace=True)
df_null = pd.DataFrame({'Count': df_clean.isnull().sum(), 'Percent': 100*df_cl
ean.isnull().sum()/len(df_clean)})
```

Step 5C : Checking mode and median of missing data columns

Now we will eamine mode and median of all columns those we trying to do estiamtion.

In [25]:
```
col_10 = list(df_null[df_null['Percent'] > 10].index)
# print(col_10)
# print(df_null.loc[col_10, :])
for index in col_10:
    df_null.loc[index, 'mode'] = df_clean[index].mode()[0]
    df_null.loc[index, 'median'] = df_clean[index].median()
print(df_null.loc[col_10, :])
```

```
                          Count  Percent  mode   median
mths_since_last_delinq  1158502    51.25 12.00    31.00
open_acc_6m              866130    38.31  0.00     1.00
open_act_il              866129    38.31  1.00     2.00
open_il_12m              866129    38.31  0.00     0.00
open_il_24m              866129    38.31  1.00     1.00
mths_since_rcnt_il       909924    40.25  7.00    13.00
total_bal_il             866129    38.31  0.00 23127.00
il_util                 1068850    47.28 78.00    72.00
open_rv_12m              866129    38.31  0.00     1.00
open_rv_24m              866129    38.31  1.00     2.00
max_bal_bc               866129    38.31  0.00  4413.00
all_util                 866348    38.32 59.00    58.00
inq_fi                   866129    38.31  0.00     1.00
total_cu_tl              866130    38.31  0.00     0.00
inq_last_12m             866130    38.31  0.00     1.00
mths_since_recent_inq    295435    13.07  1.00     5.00
```

Step 5D : predicting missing values using mode

By looking at above table, it makes more sense to fill all above columns by mode rather than median. Lets fill those missing values in above columns using mode.

In [26]:
```
df_clean[col_10] = df_clean[col_10].fillna(df_clean.mode().iloc[0])
```

Step 6 : Dropping all missing data

Now lets examine remaining missing data. Most columns will have missing data less than 10%. We will simply drop them. We may see lot of columns having missing data less than 10%. We will simply drop them. That way we will only lose 10% of data

In [27]:
```
# df_null = pd.DataFrame({'Count': df_clean.isnull().sum(), 'Percent': 100*df_
clean.isnull().sum()/len(df_clean)})
# print(df_null[df_null['Percent'] > 0])
df_clean.dropna(inplace=True)
print(df_clean.shape)
```

```
(1854073, 100)
```

Step 7 : Analyzing datatypes of clean data

Now we have get rid of all missing data. Dataset is almost clean. Lets see datatypes of datasets.

In [28]:
```
print(df_clean.dtypes)
```

```
loan_amnt                    int64
funded_amnt                  int64
funded_amnt_inv            float64
term                        object
int_rate                   float64
installment                float64
grade                       object
sub_grade                   object
emp_title                   object
emp_length                  object
home_ownership              object
annual_inc                 float64
verification_status         object
issue_d                     object
loan_status                 object
pymnt_plan                  object
purpose                     object
title                       object
zip_code                    object
addr_state                  object
dti                        float64
delinq_2yrs                float64
earliest_cr_line            object
inq_last_6mths             float64
mths_since_last_delinq     float64
open_acc                   float64
pub_rec                    float64
revol_bal                    int64
revol_util                 float64
total_acc                  float64
initial_list_status         object
out_prncp                  float64
out_prncp_inv              float64
total_pymnt                float64
total_pymnt_inv            float64
total_rec_prncp            float64
total_rec_int              float64
total_rec_late_fee         float64
recoveries                 float64
collection_recovery_fee    float64
last_pymnt_amnt            float64
collections_12_mths_ex_med float64
policy_code                  int64
application_type            object
acc_now_delinq             float64
tot_coll_amt               float64
tot_cur_bal                float64
open_acc_6m                float64
open_act_il                float64
open_il_12m                float64
open_il_24m                float64
mths_since_rcnt_il         float64
total_bal_il               float64
il_util                    float64
open_rv_12m                float64
open_rv_24m                float64
max_bal_bc                 float64
```

```
all_util                      float64
total_rev_hi_lim              float64
inq_fi                        float64
total_cu_tl                   float64
inq_last_12m                  float64
acc_open_past_24mths          float64
avg_cur_bal                   float64
bc_open_to_buy                float64
bc_util                       float64
chargeoff_within_12_mths      float64
delinq_amnt                   float64
mo_sin_old_il_acct            float64
mo_sin_old_rev_tl_op          float64
mo_sin_rcnt_rev_tl_op         float64
mo_sin_rcnt_tl                float64
mort_acc                      float64
mths_since_recent_bc          float64
mths_since_recent_inq         float64
num_accts_ever_120_pd         float64
num_actv_bc_tl                float64
num_actv_rev_tl               float64
num_bc_sats                   float64
num_bc_tl                     float64
num_il_tl                     float64
num_op_rev_tl                 float64
num_rev_accts                 float64
num_rev_tl_bal_gt_0           float64
num_sats                      float64
num_tl_120dpd_2m              float64
num_tl_30dpd                  float64
num_tl_90g_dpd_24m            float64
num_tl_op_past_12m            float64
pct_tl_nvr_dlq                float64
percent_bc_gt_75              float64
pub_rec_bankruptcies          float64
tax_liens                     float64
tot_hi_cred_lim               float64
total_bal_ex_mort             float64
total_bc_limit                float64
total_il_high_credit_limit    float64
hardship_flag                  object
disbursement_method            object
debt_settlement_flag           object
dtype: object
```

Step 8 : Fixing "Employment Length" column

In above datatypes one of the important title is employment length. We should be expecting it as integer or float but it is object type. Lets analyze this column.

In [29]:
```
print(df_clean['emp_length'].value_counts())
```

```
10+ years    665309
2 years      179041
3 years      158836
< 1 year     157037
1 year       129053
5 years      121646
4 years      119208
6 years       89893
7 years       81825
8 years       81520
9 years       70705
Name: emp_length, dtype: int64
```

Now lets remove unnecessary keywords suh as "year", "<" and "+". We will consider all records of empoyment length <1 years as 0 years and 10+ years as 10.

In [30]:
```
# df_clean['emp_length'].astype('str').dtypes
df_clean['emp_length'] = df_clean['emp_length'].str.replace(r'[+]\s|[a-z]', ''
)
df_clean['emp_length'] = df_clean['emp_length'].str.replace('< 1', '0')
# df_clean['emp_length'] = df_clean['emp_length'].str.replace('10+ ', '10')

print(df_clean['emp_length'].value_counts())
```

```
10    665309
2     179041
3     158836
0     157037
1     129053
5     121646
4     119208
6      89893
7      81825
8      81520
9      70705
Name: emp_length, dtype: int64
```

Step 9 : Analyzing clean data

Now that we have clean data, we will analyze its shape. Majority of columns are numeric and hence we will use dataframe.describe function to just see summary of numeric columns.

In [31]:
```
print(df_clean.shape)
pd.options.display.float_format = '{:.2f}'.format
print(df_clean.describe())
```

```
(1854073, 100)
       loan_amnt  funded_amnt  funded_amnt_inv   int_rate  installment  \
count 1854073.00   1854073.00       1854073.00 1854073.00   1854073.00
mean    15425.32     15425.30         15420.12      13.04       455.20
std      9248.70      9248.70          9246.58       4.85       268.33
min      1000.00      1000.00           725.00       5.31         4.93
25%      8000.00      8000.00          8000.00       9.44       260.07
50%     13875.00     13875.00         13825.00      12.62       388.14
75%     20000.00     20000.00         20000.00      15.88       605.58
max     40000.00     40000.00         40000.00      30.99      1719.83

        annual_inc        dti  delinq_2yrs  inq_last_6mths  \
count   1854073.00 1854073.00   1854073.00      1854073.00
mean      80628.03      18.95         0.31            0.54
std      111555.69      11.56         0.88            0.84
min           0.00      -1.00         0.00            0.00
25%       49900.00      12.29         0.00            0.00
50%       68000.00      18.08         0.00            0.00
75%       95481.00      24.64         0.00            1.00
max   110000000.00     999.00        58.00            8.00

       mths_since_last_delinq   open_acc    pub_rec  revol_bal  revol_util  \
count              1854073.00 1854073.00 1854073.00 1854073.00  1854073.00
mean                    23.18      11.94       0.19   16784.81       50.25
std                     19.06       5.66       0.57   22077.55       24.46
min                      0.00       1.00       0.00       0.00        0.00
25%                     12.00       8.00       0.00    6219.00       31.60
50%                     12.00      11.00       0.00   11670.00       50.10
75%                     31.00      15.00       0.00   20652.00       69.00
max                    202.00     101.00      86.00 2904836.00      366.60

       total_acc  out_prncp  out_prncp_inv  total_pymnt  total_pymnt_inv  \
count 1854073.00 1854073.00     1854073.00   1854073.00       1854073.00
mean       24.76    4597.74        4596.74     12119.76         12115.17
std        11.95    7680.93        7679.98     10001.53          9997.77
min         3.00       0.00           0.00         0.00             0.00
25%        16.00       0.00           0.00      4440.39          4438.73
50%        23.00       0.00           0.00      9384.05          9380.92
75%        31.00    7071.73        7068.60     17113.35         17106.40
max       176.00   39800.00       39800.00     63296.88         63296.88

       total_rec_prncp  total_rec_int  total_rec_late_fee  recoveries  \
count       1854073.00     1854073.00          1854073.00  1854073.00
mean           9539.78        2440.88                1.47      137.63
std            8420.34        2699.28               11.61      734.68
min               0.00           0.00               -0.00        0.00
25%            2970.19         712.77                0.00        0.00
50%            7000.00        1527.80                0.00        0.00
75%           14000.00        3135.12                0.00        0.00
max           40000.00       28192.50             1427.25    39859.55

       collection_recovery_fee  last_pymnt_amnt  collections_12_mths_ex_med
\
count               1854073.00       1854073.00                  1854073.00
mean                     23.04          3516.82                        0.02
std                     128.23          6127.53                        0.14
min                       0.00             0.00                        0.00
```

25%	0.00	317.11	0.00
50%	0.00	610.75	0.00
75%	0.00	3906.47	0.00
max	7174.72	42192.05	20.00

	policy_code	acc_now_delinq	tot_coll_amt	tot_cur_bal	open_acc_6m
count	1854073.00	1854073.00	1854073.00	1854073.00	1854073.00
mean	1.00	0.00	236.19	147955.33	0.59
std	0.00	0.07	9207.01	160776.92	1.01
min	1.00	0.00	0.00	0.00	0.00
25%	1.00	0.00	0.00	32279.00	0.00
50%	1.00	0.00	0.00	87165.00	0.00
75%	1.00	0.00	0.00	220562.00	1.00
max	1.00	14.00	9152545.00	9971659.00	16.00

	open_act_il	open_il_12m	open_il_24m	mths_since_rcnt_il
count	1854073.00	1854073.00	1854073.00	1854073.00
mean	2.24	0.44	1.41	15.52
std	2.63	0.82	1.30	20.75
min	0.00	0.00	0.00	0.00
25%	1.00	0.00	1.00	7.00
50%	1.00	0.00	1.00	7.00
75%	3.00	1.00	2.00	16.00
max	57.00	15.00	28.00	503.00

	total_bal_il	il_util	open_rv_12m	open_rv_24m	max_bal_bc
count	1854073.00	1854073.00	1854073.00	1854073.00	1854073.00
mean	23869.30	73.33	0.80	2.08	3688.84
std	40349.77	17.79	1.33	2.20	5256.76
min	0.00	0.00	0.00	0.00	0.00
25%	0.00	69.00	0.00	1.00	0.00
50%	8165.00	78.00	0.00	1.00	1982.00
75%	32819.00	78.00	1.00	3.00	5514.00
max	1837038.00	1000.00	28.00	60.00	776843.00

	all_util	total_rev_hi_lim	inq_fi	total_cu_tl	inq_last_12m
count	1854073.00	1854073.00	1854073.00	1854073.00	1854073.00
mean	58.13	34929.89	0.67	0.96	1.30
std	16.04	36335.78	1.31	2.25	2.18
min	0.00	100.00	0.00	0.00	0.00
25%	53.00	15200.00	0.00	0.00	0.00
50%	59.00	26100.00	0.00	0.00	0.00
75%	64.00	43800.00	1.00	1.00	2.00
max	239.00	9999999.00	48.00	77.00	67.00

	acc_open_past_24mths	avg_cur_bal	bc_open_to_buy	bc_util
count	1854073.00	1854073.00	1854073.00	1854073.00
mean	4.59	13815.37	11694.34	57.73
std	3.17	16110.25	16804.79	28.51
min	0.00	0.00	0.00	0.00
25%	2.00	3320.00	1699.00	35.30
50%	4.00	7814.00	5664.00	59.90
75%	6.00	19199.00	14807.00	82.90
max	64.00	623229.00	711140.00	339.60

	chargeoff_within_12_mths	delinq_amnt	mo_sin_old_il_acct
count	1854073.00	1854073.00	1854073.00

mean	0.01	11.37	125.66	
std	0.10	658.08	53.09	
min	0.00	0.00	0.00	
25%	0.00	0.00	95.00	
50%	0.00	0.00	130.00	
75%	0.00	0.00	154.00	
max	9.00	138474.00	999.00	

	mo_sin_old_rev_tl_op	mo_sin_rcnt_rev_tl_op	mo_sin_rcnt_tl	mort_acc
count	1854073.00	1854073.00	1854073.00	1854073.00
mean	177.06	13.98	8.10	1.57
std	91.99	17.12	8.47	1.90
min	1.00	0.00	0.00	0.00
25%	115.00	4.00	3.00	0.00
50%	161.00	8.00	6.00	1.00
75%	225.00	17.00	10.00	3.00
max	999.00	406.00	197.00	94.00

	mths_since_recent_bc	mths_since_recent_inq	num_accts_ever_120_pd
count	1854073.00	1854073.00	1854073.00
mean	24.90	6.42	0.52
std	31.64	5.94	1.39
min	0.00	0.00	0.00
25%	6.00	1.00	0.00
50%	14.00	5.00	0.00
75%	30.00	10.00	0.00
max	615.00	24.00	58.00

	num_actv_bc_tl	num_actv_rev_tl	num_bc_sats	num_bc_tl	num_il_tl
count	1854073.00	1854073.00	1854073.00	1854073.00	1854073.00
mean	3.68	5.57	4.73	7.77	8.91
std	2.25	3.24	2.85	4.65	7.44
min	0.00	0.00	0.00	1.00	1.00
25%	2.00	3.00	3.00	4.00	4.00
50%	3.00	5.00	4.00	7.00	7.00
75%	5.00	7.00	6.00	10.00	12.00
max	48.00	59.00	71.00	86.00	150.00

	num_op_rev_tl	num_rev_accts	num_rev_tl_bal_gt_0	num_sats
count	1854073.00	1854073.00	1854073.00	1854073.00
mean	8.31	14.03	5.59	11.91
std	4.66	7.95	3.25	5.65
min	1.00	1.00	0.00	1.00
25%	5.00	8.00	3.00	8.00
50%	7.00	12.00	5.00	11.00
75%	11.00	18.00	7.00	15.00
max	91.00	151.00	59.00	101.00

	num_tl_120dpd_2m	num_tl_30dpd	num_tl_90g_dpd_24m	num_tl_op_past_12m
count	1854073.00	1854073.00	1854073.00	1854073.00
mean	0.00	0.00	0.08	2.09
std	0.03	0.06	0.48	1.83
min	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	1.00
50%	0.00	0.00	0.00	2.00

```
75%               0.00          0.00               0.00               3.00
max               7.00          4.00              58.00              31.00

       pct_tl_nvr_dlq  percent_bc_gt_75  pub_rec_bankruptcies  tax_liens  \
count      1854073.00        1854073.00            1854073.00 1854073.00
mean            94.23             42.16                  0.12       0.05
std              8.89             36.09                  0.36       0.39
min              0.00              0.00                  0.00       0.00
25%             91.70              0.00                  0.00       0.00
50%            100.00             37.50                  0.00       0.00
75%            100.00             70.00                  0.00       0.00
max            100.00            100.00                 12.00      85.00

       tot_hi_cred_lim  total_bal_ex_mort  total_bc_limit  \
count       1854073.00         1854073.00      1854073.00
mean         184593.25           53827.65        23685.32
std          181271.77           50927.40        22892.73
min             200.00               0.00          100.00
25%           55116.00           23198.00         8700.00
50%          122782.00           40261.00        16800.00
75%          265807.00           67348.00        31000.00
max         9999999.00         3408095.00      1105500.00

       total_il_high_credit_limit
count                  1854073.00
mean                     46653.13
std                      45724.16
min                          0.00
25%                      17910.00
50%                      35203.00
75%                      61793.00
max                    2118996.00
```

Step 10: Exporting clean data to 'csv"

In [32]:
```
df_clean.to_csv("Loan_cleandata.csv")
```