

Hi Guys! Today, I have chosen an important problem statement to work upon. Let me break it down into simple words, and you will soon understand its importance.

- Financial institutions invest a ton of money for constructing credit risk analysis models to determine the probability of default of a potential borrower. The models provide information on the level of a borrower's credit risk at any particular time.

- Some of you might be wondering what "credit" is. Well here's come the definition:



The word "credit" is written in a stylized, hand-drawn font. The letters are filled with various colors: blue, green, red, and yellow. There are also small decorative elements like stars and dots scattered around the letters.

FLAMINGTEXT.COM

- "Credit is the ability to borrow money or access goods or services with the understanding that you'll pay later."
- "Creditworthiness is how a lender determines that you will default on your debt obligations, or how worthy you are to receive new credit. Your creditworthiness is what creditors look at before they approve any new credit to you."

Credit risks are a commonly observed phenomenon in areas of finance that relate to mortgages, credit cards, and other kinds of loans. There is always a probability that the borrower may not get back with the amount. Traditionally, it refers to the risk that a lender may not receive the owed principal and interest, which results in an interruption of cash flows and increased costs for collection.

Hence properly assessing and managing credit risk can lessen the severity of a loss.

Contents

1. About Dataset - Source , What it Contains, How it will be Useful.

2. Importing Libraries / APIs / Datasets

3. Segregating Dataset

4. Preliminary Investigation of Data

5. About Target Variable

6. Domain Knowledge

7. About Categorical Variables

8. Descriptive Statistics

9. Missing Values

10. Duplicate Values

11. Imbalanced - ness of Target Variable

12. Distribution, Skewness, Kurtosis

13. Variance / Standard Deviation

14. Outliers & Anomalies

15. "Credit risk Prediction Using LGBM"

16. Detailed Insights (Variable - wise)

17. Important Insights

18. Data Pre - Processing Checklist

1. About Dataset - Source, Features

Quick Feature Description

Feature - 1 ---> Age (in years)

Feature - 2 ---> Annual Income (In INR)

Feature - 3 ---> Home ownership (Nominal Categorical)

Feature - 4 ---> Employment length (in years)

Feature - 5 ---> Loan intention (Purpose of Loan)

Feature - 6 ---> Loan grade (Ordinal Categorical)

Feature - 7 ---> Loan amount (in INR)

Feature - 8 ---> Interest rate (in percentage)

Feature - 9 ---> Loan status (0 is non default 1 is default) ---> Target ---> Binary Classification

Feature - 10 ---> Loan - Percent income (Debt - Equity Ratio) (in percentage)

Feature - 11 ---> History of default (Nominal Categorical)

Feature - 12 ---> Credit history length (in years)

2. Importing Libraries / APIs / Datasets

In [545]:

```
# Data Manipulation Libraries
import numpy as np
import pandas as pd

# Data Visualization
import matplotlib.pyplot as plt
%matplotlib inline
# ---> It is an example of a predefined magic function in Ipython.%matplotlib inline makes your plot outputs appear and be stored within the notebook.
import seaborn as sns
%config InlineBackend.figure_format = 'retina' # ---> Plotting pretty figures and avoiding
sns.set_style("darkgrid")

# Specialized Library for Missing Values
! pip install missingno --quiet
import missingno as msno

# For Uni - Variate Analysis
from sklearn.feature_selection import VarianceThreshold

# Other Libraries
import os
import io
import requests
import warnings
warnings.filterwarnings(action='once')
```

DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number.
pip 23.3 will enforce this behaviour change. A possible replacement is to
upgrade to a newer version of pyodbc or contact the author to suggest that
they release a version with a conforming version number. Discussion can
be found at <https://github.com/pypa/pip/issues/12063> (<https://github.com/pypa/pip/issues/12063>)

In [546]:

Import Dataset

```
df = pd.read_csv("C:\\\\Users\\\\bolla\\\\OneDrive\\\\Desktop\\\\Notes\\\\credit_risk_dataset.csv",  
df.head(30)
```

Out[546]:

	person_age	person_income	person_home_ownership	person_emp_length	lo	lo
0	22	59000	RENT	123.0	PE	Y
1	21	9600	OWN	5.0	ED	Y
2	25	9600	MORTGAGE	1.0		Y
3	23	65500	RENT	4.0		Y
4	24	54400	RENT	8.0		Y
5	21	9900	OWN	2.0		Y
6	26	77100	RENT	8.0	ED	Y
7	24	78956	RENT	5.0		Y
8	24	83000	RENT	8.0	PE	Y
9	21	10000	OWN	6.0		Y
10	22	85000	RENT	6.0		Y
11	21	10000	OWN	2.0	HOMEIMPRC	
12	23	95000	RENT	2.0		Y
13	26	108160	RENT	4.0	ED	
14	23	115000	RENT	2.0	ED	
15	23	500000	MORTGAGE	7.0	DEBTCONSOI	
16	23	120000	RENT	0.0	ED	
17	23	92111	RENT	7.0		
18	23	113000	RENT	8.0	DEBTCONSOI	
19	24	10800	MORTGAGE	8.0	ED	
20	25	162500	RENT	2.0		Y
21	25	137000	RENT	9.0	PE	
22	22	65000	RENT	4.0	ED	
23	24	10980	OWN	0.0	PE	
24	22	80000	RENT	3.0	PE	
25	24	67746	RENT	8.0	HOMEIMPRC	
26	21	11000	MORTGAGE	3.0		Y
27	23	11000	OWN	0.0	PE	
28	24	65000	RENT	6.0	HOMEIMPRC	
29	21	11389	OTHER	5.0	ED	



In [547]:

```
# Back - up Copy

df_copy = df.copy(deep = True)
```

In [548]:

```
# About dimensions of DataFrame

df.shape
```

Out[548]:

(32581, 12)

In [549]:

```
# Information about no. of Variables and its Type.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   person_age       32581 non-null   int64  
 1   person_income    32581 non-null   int64  
 2   person_home_ownership 32581 non-null   object  
 3   person_emp_length 31686 non-null   float64 
 4   loan_intent      32581 non-null   object  
 5   loan_grade       32581 non-null   object  
 6   loan_amnt        32581 non-null   int64  
 7   loan_int_rate    29465 non-null   float64 
 8   loan_status      32581 non-null   int64  
 9   loan_percent_income 32581 non-null   float64 
 10  cb_person_default_on_file 32581 non-null   object  
 11  cb_person_cred_hist_length 32581 non-null   int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

In [550]:

df.size

Out[550]:

390972

3. Segregating Dataset

1. Segregating Target Variable from Predictors
2. Segregating Numerical Variables from Categorical Variables

In [551]:

```
# Segregating Target Variable from Predictors

df_x = df.drop(labels = ["loan_status"], axis = 1) # ---> (32581, 11)
df_y = df["loan_status"] # ---> (32581,)
```

In [552]:

```
# Segregating Numerical Variables from Categorical Variables
```

```
df_num = df.select_dtypes(include='number') # ---> (32581, 8)
df_cat = df.select_dtypes(include=['object', 'category']) # ---> (32581, 4)
```

In [553]:

df_cat

Out[553]:

	person_home_ownership	loan_intent	loan_grade	cb_person_default_on_file
0	RENT	PERSONAL	D	Y
1	OWN	EDUCATION	B	N
2	MORTGAGE	MEDICAL	C	N
3	RENT	MEDICAL	C	N
4	RENT	MEDICAL	C	Y
...
32576	MORTGAGE	PERSONAL	C	N
32577	MORTGAGE	PERSONAL	A	N
32578	RENT	HOMEIMPROVEMENT	B	N
32579	MORTGAGE	PERSONAL	B	N
32580	RENT	MEDICAL	B	N

32581 rows × 4 columns



In [554]:

df_num

Out[554]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
0	22	59000		123.0	35000	16.02
1	21	9600		5.0	1000	11.14
2	25	9600		1.0	5500	12.87
3	23	65500		4.0	35000	15.23
4	24	54400		8.0	35000	14.27
...
32576	57	53000		1.0	5800	13.16
32577	54	120000		4.0	17625	7.49
32578	65	76000		3.0	35000	10.99
32579	56	150000		5.0	15000	11.48
32580	66	42000		2.0	6475	9.99

32581 rows × 8 columns



4. Preliminary Investigation of Data

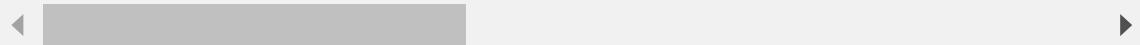
1. df.sample()
2. df.shape
3. df.info()
4. df.size
5. df.describe()
6. df["loan_status"].describe()

In [555]:

```
df.sample(10)
```

Out[555]:

	person_age	person_income	person_home_ownership	person_emp_length	
28932	28	65000	RENT	2.0	HOMEIMF
1179	24	125000	RENT	8.0	
14736	24	113000	MORTGAGE	8.0	
352	23	16000	MORTGAGE	3.0	DEBTCON
12083	22	79200	MORTGAGE	6.0	
9823	22	36000	RENT	3.0	
10209	26	66000	OWN	0.0	HOMEIMF
30724	40	62400	MORTGAGE	7.0	
19978	27	51600	RENT	1.0	
9042	24	60000	MORTGAGE	3.0	HOMEIMF



In [556]:

```
# About dimensions of DataFrame
```

```
df.shape
```

Out[556]:

```
(32581, 12)
```

In [557]:

Information about no. of Variables and its Type.

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   person_age       32581 non-null   int64  
 1   person_income    32581 non-null   int64  
 2   person_home_ownership 32581 non-null   object  
 3   person_emp_length 31686 non-null   float64 
 4   loan_intent      32581 non-null   object  
 5   loan_grade       32581 non-null   object  
 6   loan_amnt        32581 non-null   int64  
 7   loan_int_rate    29465 non-null   float64 
 8   loan_status      32581 non-null   int64  
 9   loan_percent_income 32581 non-null   float64 
 10  cb_person_default_on_file 32581 non-null   object  
 11  cb_person_cred_hist_length 32581 non-null   int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

In [558]:

df.size

Out[558]:

390972

In [559]:

df.describe()

Out[559]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_percent_income
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	32581.000000
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	1.000000
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	1.000000
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	1.000000
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	1.000000
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	1.000000
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	1.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	1.000000

5. About Target Variable

1. `df["loan_status"].describe()`
2. `df.loan_status.value_counts() / len(df) * 100`
3. Pie Chart
4. Bar Chart

In [560]:

```
# About Target Variable.
```

```
df["loan_status"].describe()
```

Out[560]:

```
count    32581.000000
mean      0.218164
std       0.413006
min      0.000000
25%     0.000000
50%     0.000000
75%     0.000000
max      1.000000
Name: loan_status, dtype: float64
```

In [561]:

```
df.loan_status.value_counts() / len(df) * 100
```

Out[561]:

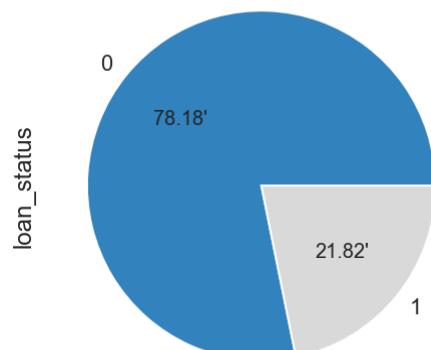
```
0    78.183604
1    21.816396
Name: loan_status, dtype: float64
```

In [363]:

```
df["loan_status"].value_counts().plot(kind="pie", autopct ="%.2f'", cmap= 'tab20c')
```

Out[363]:

```
<AxesSubplot:ylabel='loan_status'>
```



In [*]:

```
# Countplot (Bar Chart) for distribution of Target Variable

total = float(len(df)) # one person per row

ax = sns.countplot(x = "loan_status", data = df)
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x() + p.get_width() / 2.,
            height + 3,
            '{:1.2f}'.format(height/total),
            ha="center")
```

In [*]:

```
cor_target = df.corrwith(df["loan_status"])

cor_target
```

1. Though this method of variable ranking has more relevance in Regression model,
it gives some insights. ???

In [*]:

```
# Sort the variables in descending order (Ranking of Variables w.r.t. to corr with Target)

cor_target.sort_values(axis = 0, ascending = False)
```

6. Domain Knowledge

Domain Knowledge - I

1. Loan Percent Income Variable ---> Numerical Variable which essentially means the loan amount as percentage of income of borrower.
2. In finance jargon, it is called as "**Debt to Income ratio**". (**DTI**)
3. It is one of the most important deciding factor while lending a loan.
4. It is calculated as,

**DTI ratio = Total monthly debt payments/Gross monthly income
 (before Tax) X 100.**

5. Good / Ideal DTI ratio ---> The percentage of DTI ratio may vary from lender to lender. However, in general, a DTI ratio of up to 40% may be considered suitable for getting a loan approved. **DTI of 21% - 35% is considered as very good..**
6. A low debt-to-income (DTI) ratio demonstrates a good balance between debt and income. In other words, if your DTI ratio is 15%, that means that 15% of your monthly gross income goes to debt payments each month.

Domain Knowledge - II

1. Person's Credit History length ---> Length of Credit History.
 2. Longer the history, more knowledge about the Borrower which means more transparency and ultimately more reliable. So, based on Domain knowledge, Person's Credit History length variable is positively correlated with Target Variable. This shall be cross- verified using correlation matrix, Scatter Plot and Heat Map in later part of the Project.
-

6. About Categorical Variables

1. df.describe(include = "O").T
2. df.select_dtypes(include=['object','category']).columns.tolist()
3. df_cat = df.select_dtypes(include=['object','category'])
4. df.dtypes.value_counts()
5. df["person_home_ownership"].value_counts()
6. df.person_home_ownership.value_counts() / len(df) * 100
7. df[cat_col].mode()
8. Types of Categorical values
9. Bar Plot of distribution of unique values of Categorical Variables

In [565]:

```
# A quick glance over Categorical Variables

df.describe(include = "O").T
```

Out[565]:

	count	unique	top	freq
person_home_ownership	32581	4	RENT	16446
loan_intent	32581	6	EDUCATION	6453
loan_grade	32581	7	A	10777
cb_person_default_on_file	32581	2	N	26836

In [566]:

```
df.select_dtypes(include=['object','category']).columns.tolist()
```

Out[566]:

```
['person_home_ownership',
 'loan_intent',
 'loan_grade',
 'cb_person_default_on_file']
```

In [562]:

```
# List of Categorical Variables in a DataFrame  
  
df_cat = df.select_dtypes(include=['object', 'category'])  
df_cat.count()
```

Out[562]:

```
person_home_ownership      32581  
loan_intent                 32581  
loan_grade                  32581  
cb_person_default_on_file   32581  
dtype: int64
```

In [563]:

```
# No of Categorical Variables in a DataFrame  
  
df.dtypes.value_counts()
```

Out[563]:

```
int64      5  
object     4  
float64    3  
dtype: int64
```

In [564]:

```
# Frequency of Unique Values in each Categorical Variable  
  
df["person_home_ownership"].value_counts()  
df["loan_intent"].value_counts()  
df["loan_grade"].value_counts()  
df["cb_person_default_on_file"].value_counts()
```

Out[564]:

```
N      26836  
Y      5745  
Name: cb_person_default_on_file, dtype: int64
```

In [567]:

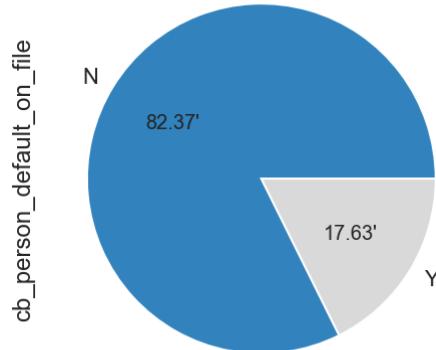
```
df["cb_person_default_on_file"].value_counts().plot(kind="pie", autopct ="%.2f'", cmap=
```

History of Default - Yes - True - 17.63%

Not History of Default - No - False - 82.37%

Out[567]:

<AxesSubplot:ylabel='cb_person_default_on_file'>



In [568]:

```
# Frequency of Unique Values in each Categorical Variable in percentage form
```

```
df.person_home_ownership.value_counts() / len(df) * 100
df.loan_intent.value_counts() / len(df) * 100
(df.loan_grade.value_counts() / len(df) * 100).round(2)
df.cb_person_default_on_file.value_counts() / len(df) * 100
```

OR

```
(df.person_home_ownership.value_counts(normalize = True) * 100).round(2)
(df.loan_intent.value_counts(normalize = True) * 100).round(2)
(df.loan_grade.value_counts(normalize = True) * 100).round(2)
# (df.cb_person_default_on_file.value_counts(normalize = True) * 100).round(2)
```

Out[568]:

A	33.08
B	32.08
C	19.82
D	11.13
E	2.96
F	0.74
G	0.20

Name: loan_grade, dtype: float64

In [569]:

Mode of Categorical Values

```
cat_col = ["person_home_ownership", "loan_intent", "loan_grade", "cb_person_default_on_file"]
df[cat_col].mode()
```

Out[569]:

	person_home_ownership	loan_intent	loan_grade	cb_person_default_on_file
0	RENT	EDUCATION	A	N

In [570]:

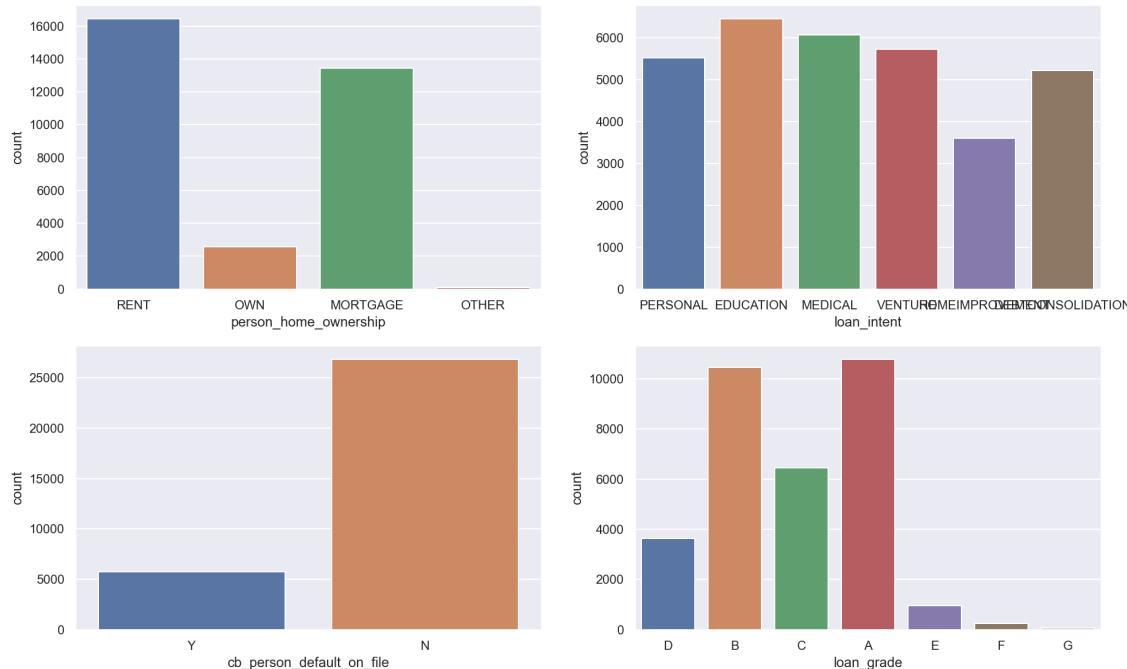
Countplot of all Categorical variables

```
fig,axes = plt.subplots(2,2, figsize=(16,10))

sns.countplot('person_home_ownership', data=df, ax=axes[0,0])
sns.countplot('loan_intent', data=df, ax=axes[0,1])
sns.countplot('loan_grade', data=df, ax=axes[1,1])
sns.countplot('cb_person_default_on_file', data=df, ax=axes[1,0])
```

Out[570]:

<AxesSubplot:xlabel='cb_person_default_on_file', ylabel='count'>



In [571]:

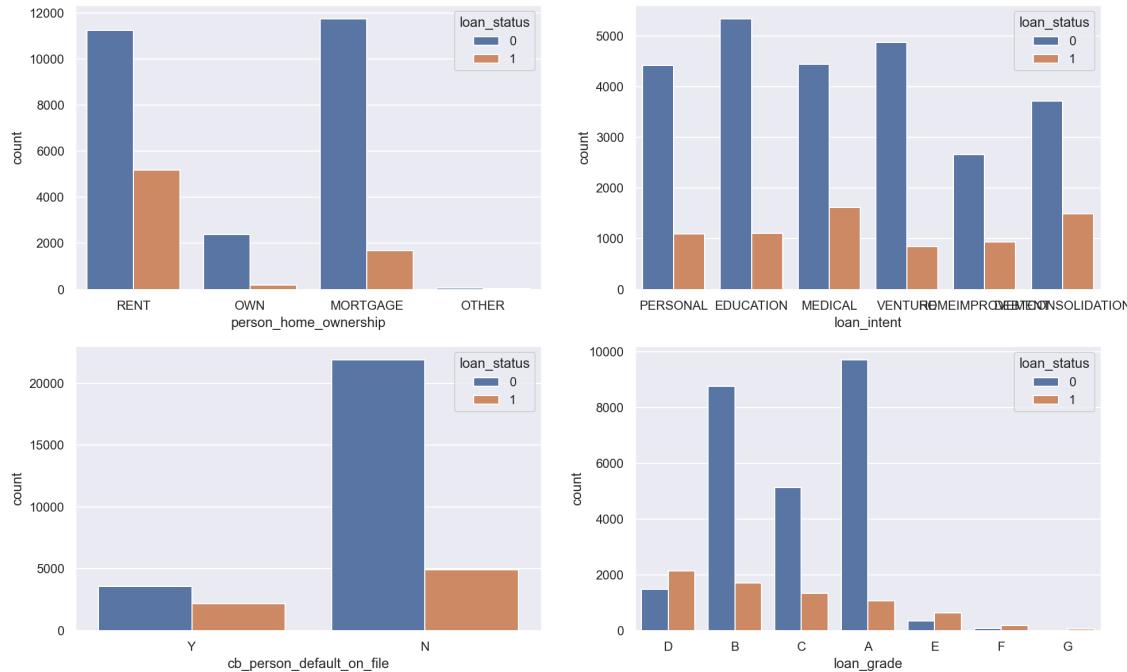
Bi - Variate Analysis of Categorical Variables vis-a-vis Target Variables.

```
fig,axes = plt.subplots(2,2, figsize=(16,10))

sns.countplot('person_home_ownership', hue = "loan_status", data = df, ax = axes[0,0])
sns.countplot('loan_intent', hue = "loan_status", data = df, ax = axes[0,1])
sns.countplot('loan_grade', hue = "loan_status", data = df, ax = axes[1,1])
sns.countplot('cb_person_default_on_file', hue = "loan_status", data = df, ax = axes[1,0])
```

Out[571]:

<AxesSubplot:xlabel='cb_person_default_on_file', ylabel='count'>



7. Descriptive Statistics

In [572]:

```
# Basic Statistical summary of Numerical variables.
```

```
df_num.describe().T.round(2)
```

Out[572]:

	count	mean	std	min	25%	50%	7
person_age	32581.0	27.73	6.35	20.00	23.00	26.00	30
person_income	32581.0	66074.85	61983.12	4000.00	38500.00	55000.00	79200
person_emp_length	31686.0	4.79	4.14	0.00	2.00	4.00	7
loan_amnt	32581.0	9589.37	6322.09	500.00	5000.00	8000.00	12200
loan_int_rate	29465.0	11.01	3.24	5.42	7.90	10.99	13
loan_status	32581.0	0.22	0.41	0.00	0.00	0.00	0
loan_percent_income	32581.0	0.17	0.11	0.00	0.09	0.15	0
cb_person_cred_hist_length	32581.0	5.80	4.06	2.00	3.00	4.00	8

8. Missing Values

1. df.isnull().sum()
2. df.isna().sum()
3. df.isnull().sum().sum()
4. df.isnull().mean() * 100
5. percent_missing = df.isnull().sum() * 100 / len(df)
6. percent_missing = df.isnull().sum().sort_values(ascending = False) * 100 / len(df)
7. df.isnull().sum().sort_values(ascending = False)
8. df.isna().sum().sort_values(ascending = False)
9. Missing Value Visualization - Bar Chart
10. Missing Value Visualization - Dendrogram

In [573]:

```
# Missing Values  
df.isna().sum().sort_values(ascending = False)
```

Out[573]:

```
loan_int_rate            3116  
person_emp_length        895  
person_age                 0  
person_income                 0  
person_home_ownership                 0  
loan_intent                 0  
loan_grade                  0  
loan_amnt                   0  
loan_status                  0  
loan_percent_income                 0  
cb_person_default_on_file                 0  
cb_person_cred_hist_length                 0  
dtype: int64
```

In [574]:

```
# Total Missing Values in a DataFrame  
df.isnull().sum().sum()
```

Out[574]:

```
4011
```

In [575]:

```
# Column-wise Missing Values in relative form.  
  
percent_missing = (df.isnull().sum().sort_values(ascending = False) * 100 / len(df)).round(2)
```

Out[575]:

```
loan_int_rate            9.56  
person_emp_length        2.75  
person_age                 0.00  
person_income                 0.00  
person_home_ownership                 0.00  
loan_intent                 0.00  
loan_grade                  0.00  
loan_amnt                   0.00  
loan_status                  0.00  
loan_percent_income                 0.00  
cb_person_default_on_file                 0.00  
cb_person_cred_hist_length                 0.00  
dtype: float64
```

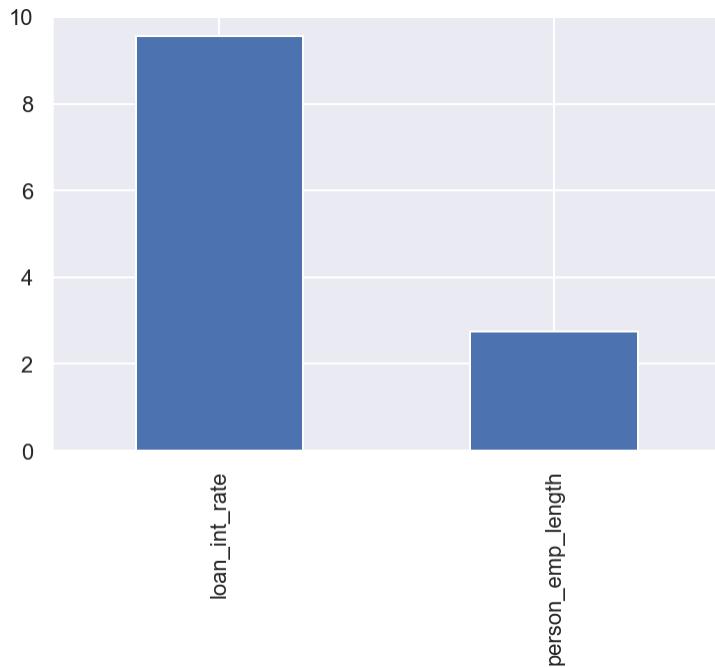
In [576]:

```
# Bar Chart of Missing Values Column- wise in percentage form.
```

```
percent_missing[percent_missing != 0].plot(kind='bar')
```

Out[576]:

<AxesSubplot:>



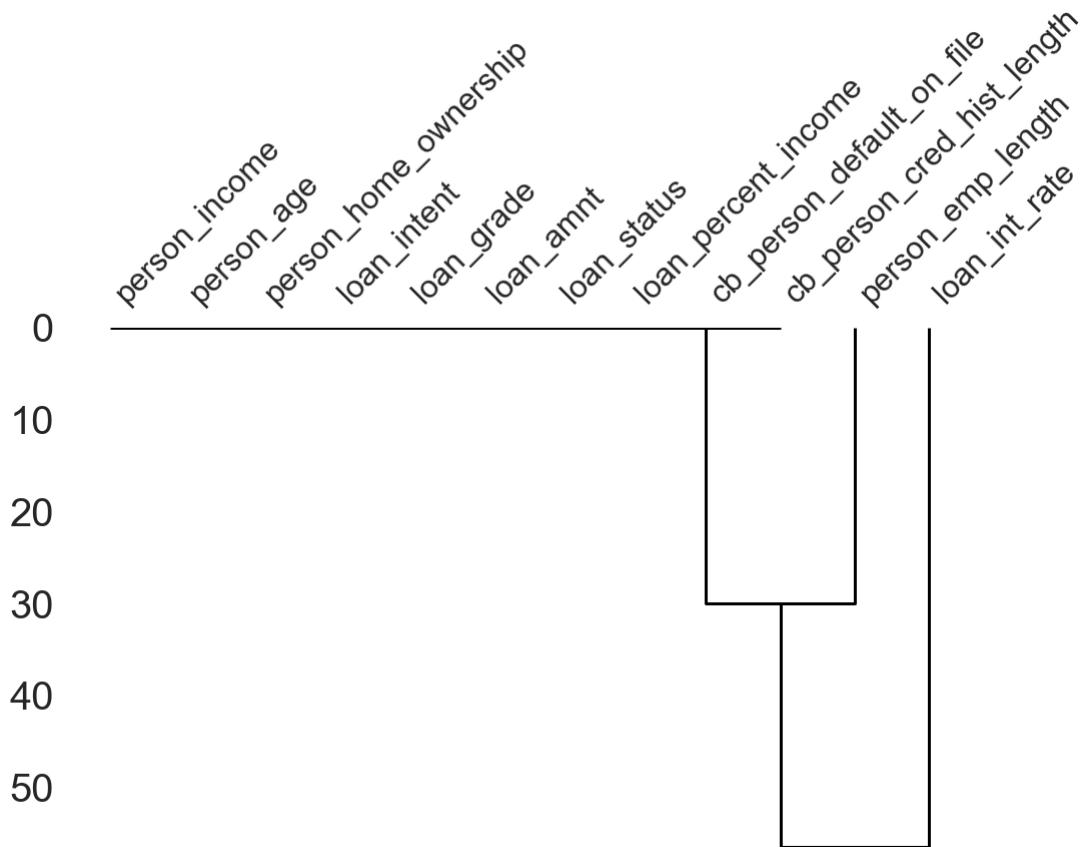
In [577]:

```
# Dendrogram of Missing Values
```

```
msno.dendrogram(df, figsize = (8, 5))
```

Out[577]:

<AxesSubplot:>



9. Duplicate Values

1. df.duplicated().sum()
2. df.loc[df.duplicated(), :]

In [578]:

```
# Duplicate Values
```

```
df.duplicated().sum()
```

Out[578]:

165

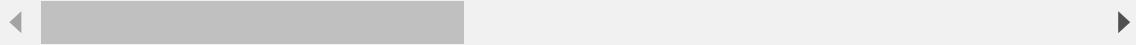
In [579]:

```
df.loc[df.duplicated(), :]
```

Out[579]:

	person_age	person_income	person_home_ownership	person_emp_length	
15975	23	42000	RENT	5.0	
15989	23	90000	MORTGAGE	7.0	
15995	24	48000	MORTGAGE	4.0	
16025	24	10000	RENT	8.0	
16028	23	100000	MORTGAGE	7.0	
...
32010	42	39996	MORTGAGE	2.0	HOMEIMF
32047	36	250000	RENT	2.0	DEBTCON
32172	49	120000	MORTGAGE	12.0	
32259	39	40000	OWN	4.0	
32279	43	11340	RENT	4.0	

165 rows × 12 columns



10. Imbalanced - ness of Target Variable

1. df.loan_status.value_counts()
2. df.loan_status.value_counts() / len(df) * 100

In [580]:

```
df.loan_status.value_counts()
```

Out[580]:

```
0    25473
1    7108
Name: loan_status, dtype: int64
```

In [581]:

```
(df.loan_status.value_counts() / len(df) * 100).round(2)
```

Out[581]:

```
0    78.18
1    21.82
Name: loan_status, dtype: float64
```

11. Distribution, Skewness, Kurtosis

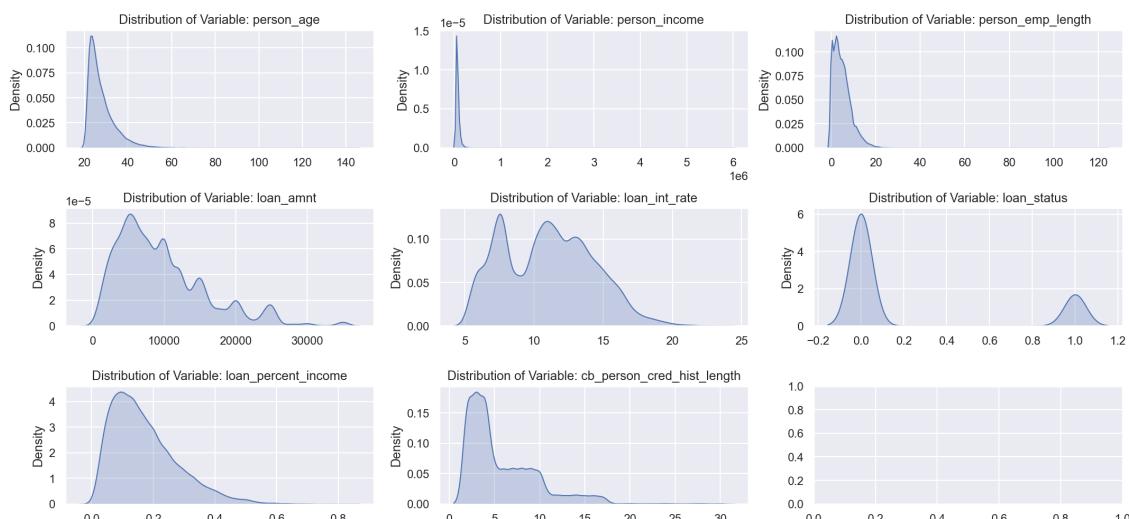
1. Density Plot for Distribution
2. `df[""].skew()`
3. `df.kurtosis()`

In [582]:

```
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 7), sharex = False, sharey = False)
axes = axes.ravel() # array to 1D
cols = df_num.columns[:] # create a list of dataframe columns to use

for col, ax in zip(cols, axes):
    data = df_num # select the data
    sns.kdeplot(data=data, x=col, shade=True, ax=ax)
    ax.set(title=f'Distribution of Variable: {col}', xlabel=None)

# fig.delaxes(axes[5]) # delete the empty subplot
fig.tight_layout()
plt.show()
```



In [583]:

```
# Skewness of Numerical Variables

# df["person_age"].skew()                      # ---> 2.5813925261849846
# df["person_income"].skew()                    # ---> 32.865348833904434
# df["person_emp_length"].skew()                # ---> 2.6144551214595424
# df["loan_amnt"].skew()                        # ---> 1.1924774277661998
# df["loan_int_rate"].skew()                    # ---> 0.2085503016908977
# df["loan_percent_income"].skew()              # ---> 1.0646686367683245
# df["cb_person_cred_hist_length"].skew()        # ---> 1.66179011991262

df_num.skew()
```

Out[583]:

person_age	2.581393
person_income	32.865349
person_emp_length	2.614455
loan_amnt	1.192477
loan_int_rate	0.208550
loan_status	1.364888
loan_percent_income	1.064669
cb_person_cred_hist_length	1.661790
dtype: float64	

In [584]:

```
# Kurtosis of Numerical Variables

df_num.kurtosis()
```

Out[584]:

person_age	18.560825
person_income	2693.272776
person_emp_length	43.722338
loan_amnt	1.423565
loan_int_rate	-0.671609
loan_status	-0.137088
loan_percent_income	1.223687
cb_person_cred_hist_length	3.716194
dtype: float64	

12. Variance / Standard Deviation

1. df.var()
2. from sklearn.feature_selection import VarianceThreshold

In [585]:

```
# It's a part of Feature Selection to drop those features who have Zero - Variance or None
# features are un-informative.
# in this section, I am identifying the variables with Zero and near - Zero Variance using
# 1. Feature Selection Method : Variance Threshold to remove the Constant Features
# 2. Feature Selection Method : Variance Threshold to remove the Quasi-Constant Features
```

In [586]:

```
# 1. Feature Selection Method : Variance Threshold to remove the Constant Features

# Those features which contain constant values (only one value for all the outputs or target).
# These features provide no information that allows ML models to predict the target.
# threshold = 0

# Lets apply variancethreshold

from sklearn.feature_selection import VarianceThreshold
```

In [587]:

```
constant_selector= VarianceThreshold(threshold = 0)
constant_selector.fit(df_num)
```

Out[587]:

```
VarianceThreshold(threshold=0)
```

In [588]:

```
# True means non-constant features.
# There are total 8 numerical features. Out of which none is Constant feature.

constant_selector.get_support()
```

Out[588]:

```
array([ True,  True,  True,  True,  True,  True,  True,  True])
```

In [589]:

```
# Sum of non - constant feature.

sum(constant_selector.get_support())
```

Out[589]:

```
8
```

In [590]:

```
# Now, let's find constant feaures which is None (from above)

constant_columns = [column for column in df_num.columns
                     if column not in df_num.columns[constant_selector.get_support()]]

print(len(constant_columns))

# Note

# 1. As was predicted, there are no Constant features.
```

0

In [591]:

```
# List of constant features.

for column in constant_columns:
    print(column)
```

In [592]:

```
# Dropping of Constant features, which is beyond the scope of this EDA.

# df_num.drop(constant_columns, axis = 1, inplace = True)
```

In [593]:

```
# 2. Feature Selection Method : Variance Threshold to remove the Quasi-Constant Features

quasi_constant_selector = VarianceThreshold(threshold = 0.01)      # ---> Threshold = 1%
quasi_constant_selector.fit(df_num)
```

Out[593]:

VarianceThreshold(threshold=0.01)

In [594]:

```
# Get features which have the variance GREATER than the set threshold value = 0.01

quasi_constant_selector.get_support()
```

Out[594]:

array([True, True, True, True, True, True, True, True])

In [595]:

```
# Sum of non - quasi constant feature.

sum(quasi_constant_selector.get_support())
```

Out[595]:

8

In [596]:

```
# Now, let's find quasi constant feaures which is None (from above)

quasi_constant_columns = [column for column in df_num.columns
                           if column not in df_num.columns[quasi_constant_selector.get_support()]]
print(len(quasi_constant_columns))

# Note

# 1. As was predicted, there are no Constant features.
```

0

In [597]:

```
# List of quasi constant features.

for column in quasi_constant_columns:
    print(column)
```

In [598]:

```
# Dropping of quasi Constant features, which is beyond the scope of this EDA.

# df_num.drop(quasi_constant_columns, axis = 1, inplace = True)
```

13. Outliers & Anomalies

1. Uni-variate Outliers

- IQR Proximity Rule (Non - Parametric)
- Box - Plot (Non - Parametric)
- Z - Score (Parammetric) - X
- Percentile Method
- Isolation Forest algorithm - X
- Median Absolute Deviation (MAD) - X
- KNN - X

2. Bi - Variate Outliers

- Scatter Plot

Note

1. Z - Score is not preferred method to detect Outliers in this particular dataset since it is Parametric test and none of the variable (column) in the dataset is either normally distributed or closely normally distributed. (Refer - Pt. 12).
2. Use of IQR proximity rule is preferable as the data is not normally distributed.

In [599]:

```
# Uni - Variate Outliers

# Method - I ( IQR Proximity Rule)

percentile_25 = df_num["person_age"].quantile(0.25)      # ---> 23
percentile_75 = df_num["person_age"].quantile(0.75)      # ---> 30

percentile_25
percentile_75

# Value of percentile_25 & percentile_75 cross- checked with df.describe() function.

# Now,

iqr = percentile_75 - percentile_25
iqr

# Defining Upper & Lower Limits

upper_limit = percentile_75 + 1.5 * iqr      # ---> 40.5
lower_limit = percentile_25 - 1.5 * iqr      # ---> 12.5 (min = 20). Hence, no Outliers c

upper_limit
lower_limit

# Outliers

df_num[df_num["person_age"] > upper_limit]    # ---> 1494 Outliers
df_num[df_num["person_age"] < lower_limit]    # ---> 0 Outliers

# OR

IQR_outliers = df_num[(df_num.person_age <= lower_limit) | (df_num.person_age >= upper_]
IQR_outliers  # ---> 1494
```

Out[599]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
81	144	250000		4.0	4800	13.57
183	144	200000		4.0	6000	11.86
575	123	80004		2.0	20400	10.25
747	123	78000		7.0	20000	NaN
29121	50	900000		11.0	30000	12.69
...
32576	57	53000		1.0	5800	13.16
32577	54	120000		4.0	17625	7.49
32578	65	76000		3.0	35000	10.99
32579	56	150000		5.0	15000	11.48
32580	66	42000		2.0	6475	9.99

1494 rows × 8 columns



In [600]:

```
# Uni - Variate Outliers

# Method - II (Percentile Method)

# It is similar to IQR method. Here, we arbitrarily decides the cut - off percentiles on

upper_bound = df_num["person_age"].quantile(0.99) # ---> 50.0
lower_bound = df_num["person_age"].quantile(0.01) # ---> 22.0

# upper_bound = df_num["person_age"].quantile(0.95) # ---> 40.0
# Lower_bound = df_num["person_age"].quantile(0.05) # ---> 22.0

# upper_bound = df_num["person_age"].quantile(0.90) # ---> 36.0
# Lower_bound = df_num["person_age"].quantile(0.10) # ---> 22.0

print("upper_bound : ",upper_bound)
print("lower_bound : ",lower_bound)

upper_bound : 50.0
lower_bound : 21.0
```

In [601]:

```
# Uni - Variate Outliers

# Method - II (Percentile Method)

# df_num[(df_num["person_age"] > 50.0) | (df_num["person_age"] < 22.0)]      # Outliers @
# df_num[(df_num["person_age"] > 40.0) | (df_num["person_age"] < 22.0)]      # Outliers @
df_num[(df_num["person_age"] > 36.0) | (df_num["person_age"] < 22.0)]      # Outliers @

# Outliers as per IQR proximity rule ---> 1494
```

Out[601]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
1	21	9600		5.0	1000	11.14
5	21	9900		2.0	2500	7.14
9	21	10000		6.0	1600	14.74
11	21	10000		2.0	4500	8.63
26	21	11000		3.0	4575	17.74
...
32576	57	53000		1.0	5800	13.16
32577	54	120000		4.0	17625	7.49
32578	65	76000		3.0	35000	10.99
32579	56	150000		5.0	15000	11.48
32580	66	42000		2.0	6475	9.99

4162 rows × 8 columns



In [602]:

```
# Uni - Variate Outliers

# Method - I ( IQR Proximity Rule)

percentile_25 = df_num["cb_person_cred_hist_length"].quantile(0.25)      # ---> 3
percentile_75 = df_num["cb_person_cred_hist_length"].quantile(0.75)      # ---> 8

percentile_25
percentile_75

# Value of percentile_25 & percentile_75 cross- checked with df.describe() function.

# Now,

iqr = percentile_75 - percentile_25      # ---> 5
iqr

# Defining Upper & Lower Limits

upper_limit = percentile_75 + 1.5 * iqr    # ---> 15.5
lower_limit = percentile_25 - 1.5 * iqr    # ---> 4.5

upper_limit
lower_limit

# Outliers

df_num[df_num["cb_person_cred_hist_length"] > upper_limit]    # ---> 1142
df_num[df_num["cb_person_cred_hist_length"] < lower_limit]    # ---> 0

# OR

IQR_outliers = df_num[(df_num.cb_person_cred_hist_length <= lower_limit) | (df_num.cb_pe
```

Out[602]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
29120	40	1200000		1.0	10000	NaN
29125	37	80000		0.0	35000	11.99
29127	44	584800		4.0	20000	NaN
29130	37	85000		5.0	35000	8.90
29133	41	480000		3.0	16000	7.90
...
32576	57	53000		1.0	5800	13.16
32577	54	120000		4.0	17625	7.49
32578	65	76000		3.0	35000	10.99
32579	56	150000		5.0	15000	11.48
32580	66	42000		2.0	6475	9.99

1142 rows × 8 columns

Outliers Variable - wise (As per IQR Proximity Rule)

1. person_age = 1494
2. person_income = 1484
3. person_emp_length = 853
4. loan_amnt = 1689
5. loan_int_rate = 6
6. loan_status = X
7. loan_percent_income = 651
8. cb_person_cred_hist_length = 1142

In [603]:

```
# Outlier treatment

# Mainly there 2 options - Drop or Cap

# Option - 1 (Drop)

# new_df_num = df[df['person_age'] < upper_limit & df[df['person_age'] > lower_limit]

# Option - 2 (Cap)

# new_df_cap['person_age'] = np.where(
#     new_df_cap['person_age'] > upper_limit,
#     upper_limit,
#     np.where(
#         new_df_cap['person_age'] < lower_limit,
#         lower_limit,
#         new_df_cap['person_age']
#     )
# )

# Note -

# 1. Since Outliers are too many, it's not wise to opt the option - 1 i.e drop rows as it
#    information. Capping is preferred over Dropping. On flip side, the Outliers are because
#    Max age of 144 indicates the Human error. In that Case dropping Outliers is a better

# 2. Qty. of Outliers = 1494 / 32581 * 100 = 4.58 %

# 3. Another option is to modify the Upper Limit and Lower Limit by
#    multiply the IQR with higher no. say 2 instead of 1.5.

# 4. Which Outlier to choose IQR method or percentile method of various percentiles ?
#    Check the descriptive stastics. Choose the one which least affect the mean and other
#    Also, DS can put the knowledge of Domain to identify the ideal cut - off (threshold)
```

In [604]:

```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.set(rc = {"figure.figsize" : (18,5)})

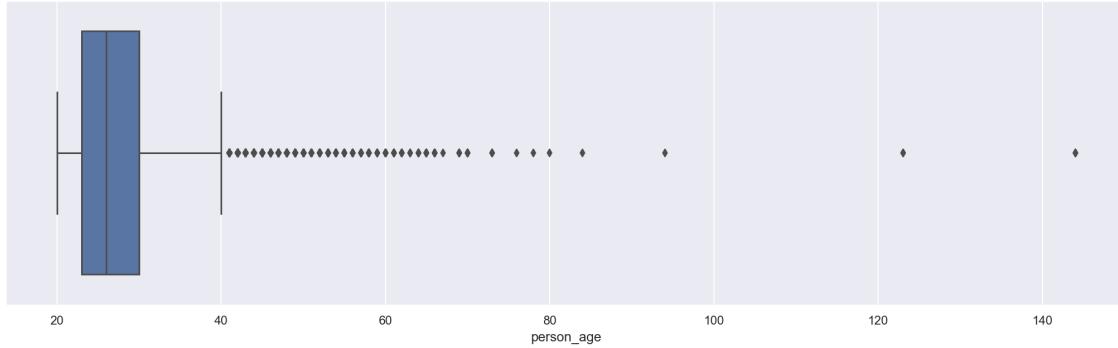
sns.boxplot(df_num[ "person_age"])

# Code for multiple box - plots in single frame

# num_col_list = df_num[:]
# sns.boxplot(x = "variable", y = "value", data = df_num.melt(value_vars=num_col_list))
```

Out[604]:

<AxesSubplot:xlabel='person_age'>



In [605]:

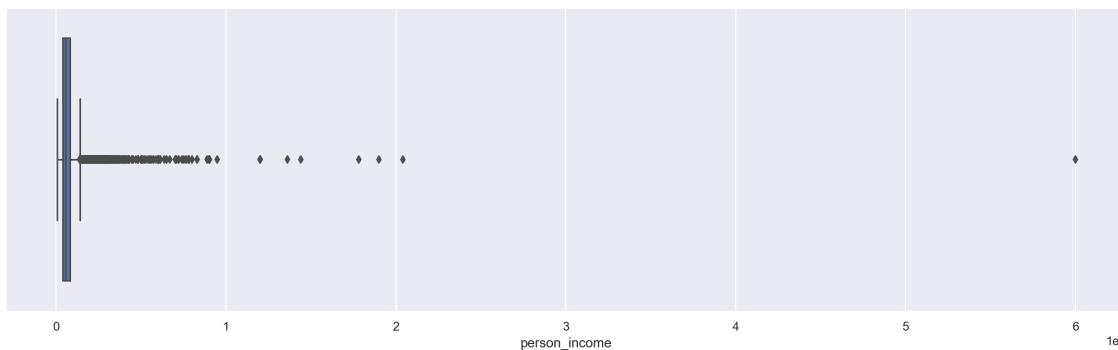
```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.boxplot(df_num[ "person_income"])
```

Out[605]:

<AxesSubplot:xlabel='person_income'>



In [606]:

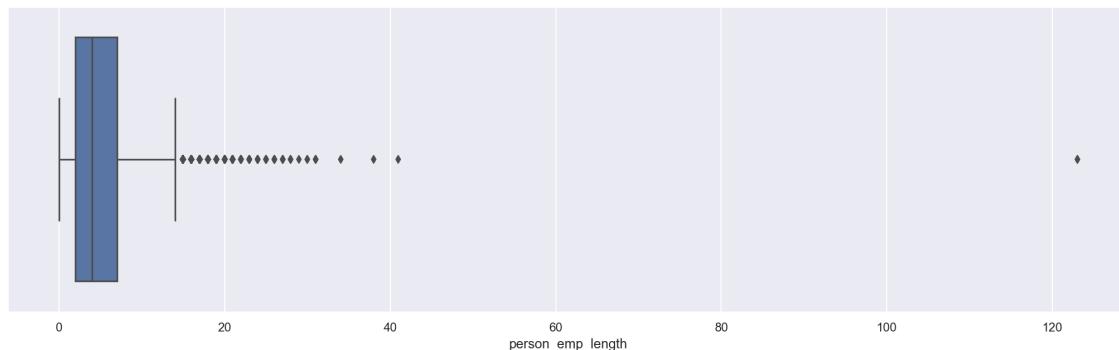
```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.boxplot(df_num["person_emp_length"])
```

Out[606]:

```
<AxesSubplot:xlabel='person_emp_length'>
```



In [607]:

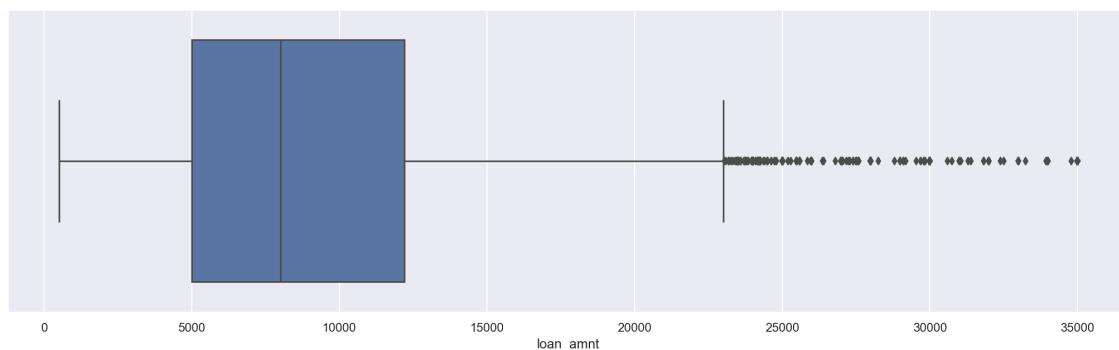
```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.boxplot(df_num["loan_amnt"])
```

Out[607]:

```
<AxesSubplot:xlabel='loan_amnt'>
```



In [608]:

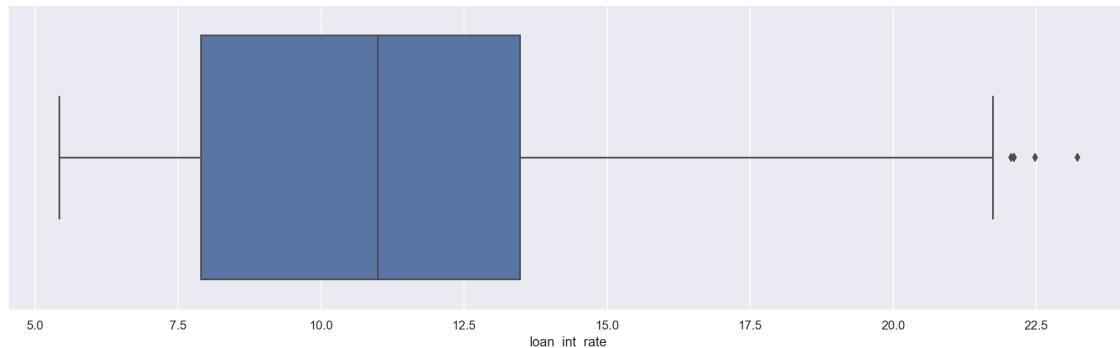
```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.boxplot(df_num["loan_int_rate"])
```

Out[608]:

```
<AxesSubplot:xlabel='loan_int_rate'>
```



In [609]:

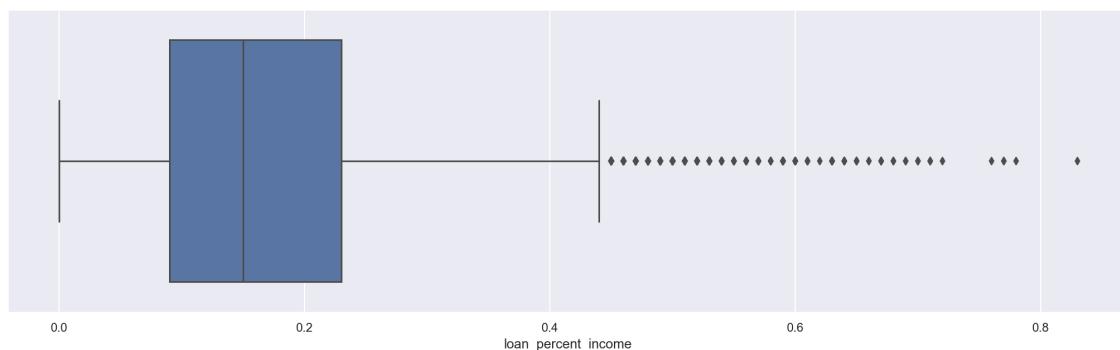
```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.boxplot(df_num["loan_percent_income"])
```

Out[609]:

```
<AxesSubplot:xlabel='loan_percent_income'>
```



In [610]:

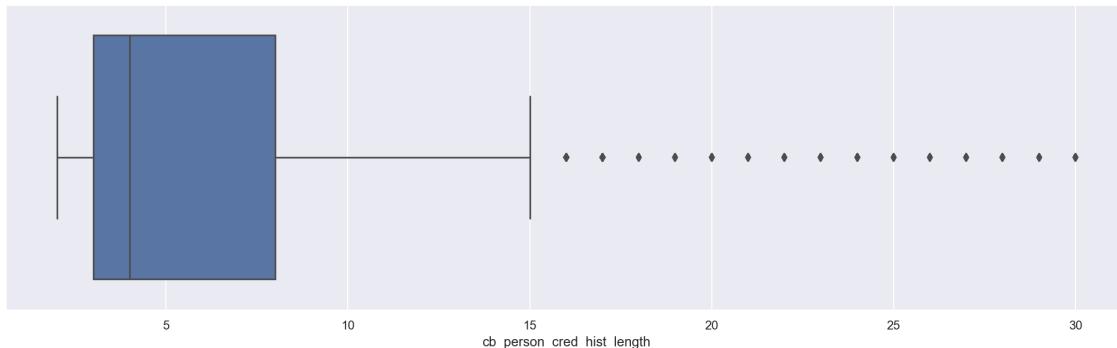
```
# Uni - Variate Outliers

# Method - III ( Uni - Variate Box - Plot)

sns.boxplot(df_num["cb_person_cred_hist_length"])
```

Out[610]:

```
<AxesSubplot:xlabel='cb_person_cred_hist_length'>
```



Bi - Variate Outliers

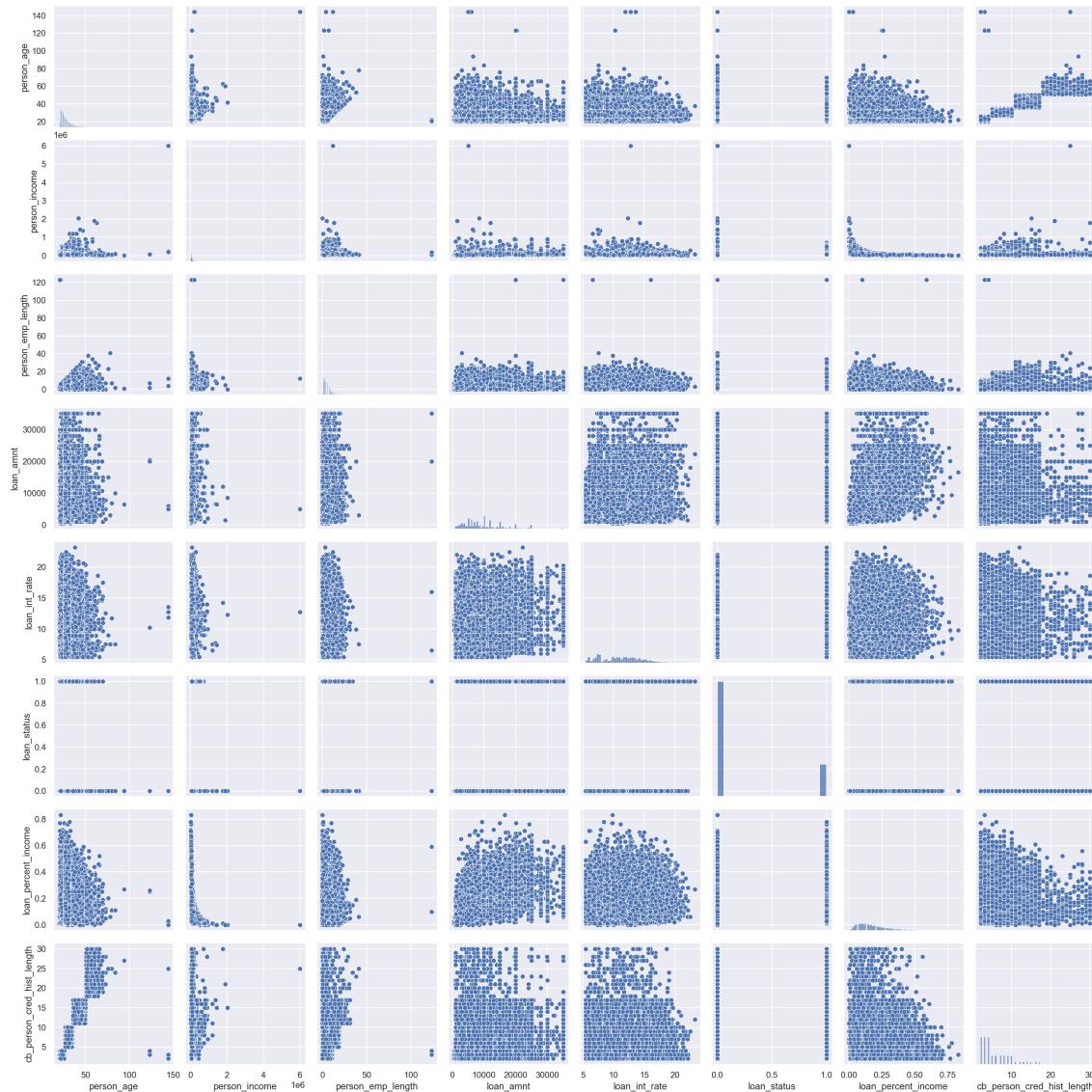
1. Scatter Plots (Numerical - Numerical)

In [416]:

```
# Bi - Variate Outliers

# Method - I (Scatter Plot)

sns.pairplot(df_num)
```



14. Bi - Variate & Multi - Variate Analysis

Bi - Variate

1. Categorical - Categorical
2. Numerical - Numerical
3. Numerical - Categorical

Multi - Variate

1. Numerical - Categorical - Categorical
2. Numerical - Categorical - Numerical

Bi - Variate - (Categorical - Categorical)

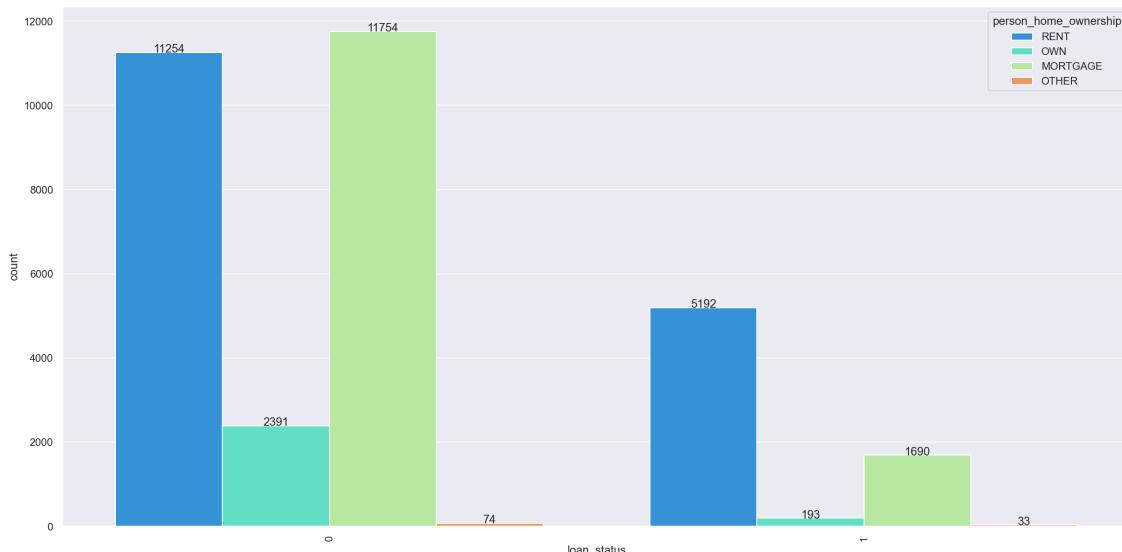
1. Target - person_home_ownership
2. Target - loan_intent
3. Target - loan_grade
4. Target - cb_person_default_on_file
5. pd.crosstab() ---> working with categorical data
6. HeatMap of cross - tab

In [417]:

```
fig, ax1 = plt.subplots(figsize=(20,10))
graph = sns.countplot(ax=ax1,x='loan_status',hue="person_home_ownership",data=df, palette="Set1")
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1, height ,ha="center")
```

Note

```
# 1. Categorical - Categorical ---> Target - Loan_status
# ---> Predictor - person_home_ownership
```

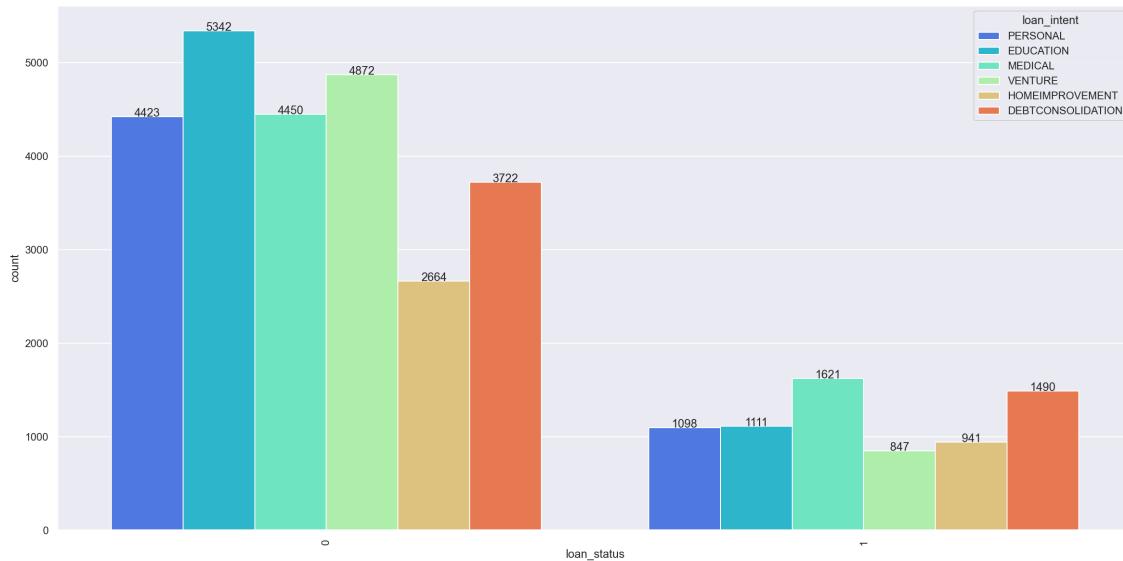


In [418]:

```
fig, ax1 = plt.subplots(figsize=(20,10))
graph = sns.countplot(ax=ax1,x='loan_status',hue="loan_intent",data=df, palette="rainbow")
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1, height ,ha="center")

# Note

# 1. Categorical - Categorical ---> Target - Loan_status
#                                     ---> Predictor - Loan_intent
```

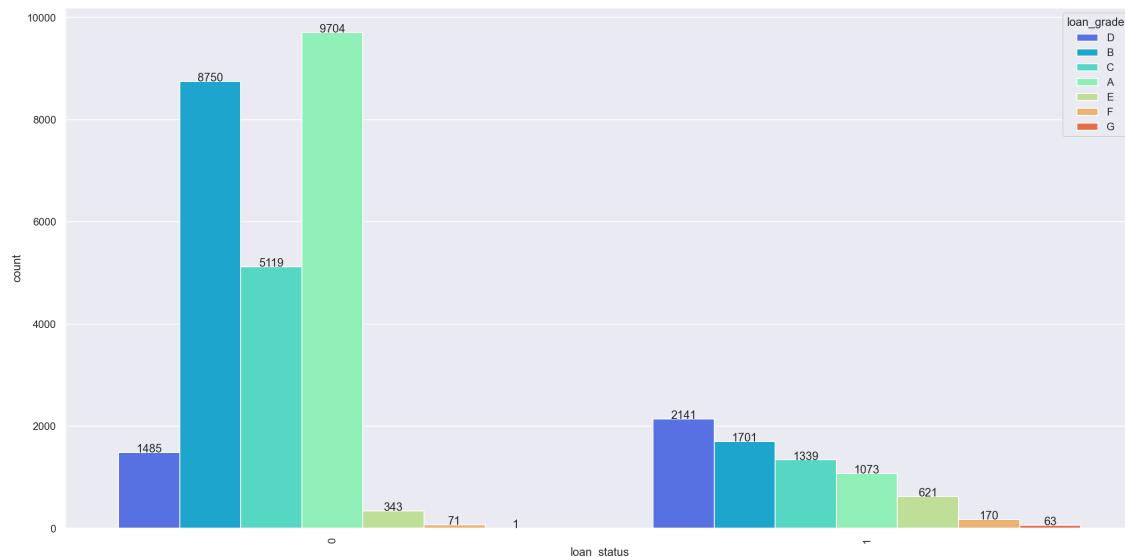


In [419]:

```
fig, ax1 = plt.subplots(figsize=(20,10))
graph = sns.countplot(ax=ax1,x='loan_status',hue="loan_grade",data=df, palette="rainbow")
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1, height ,ha="center")

# Note

# 1. Categorical - Categorical ---> Target - Loan_status
#                               ---> Predictor - Loan_grade
```



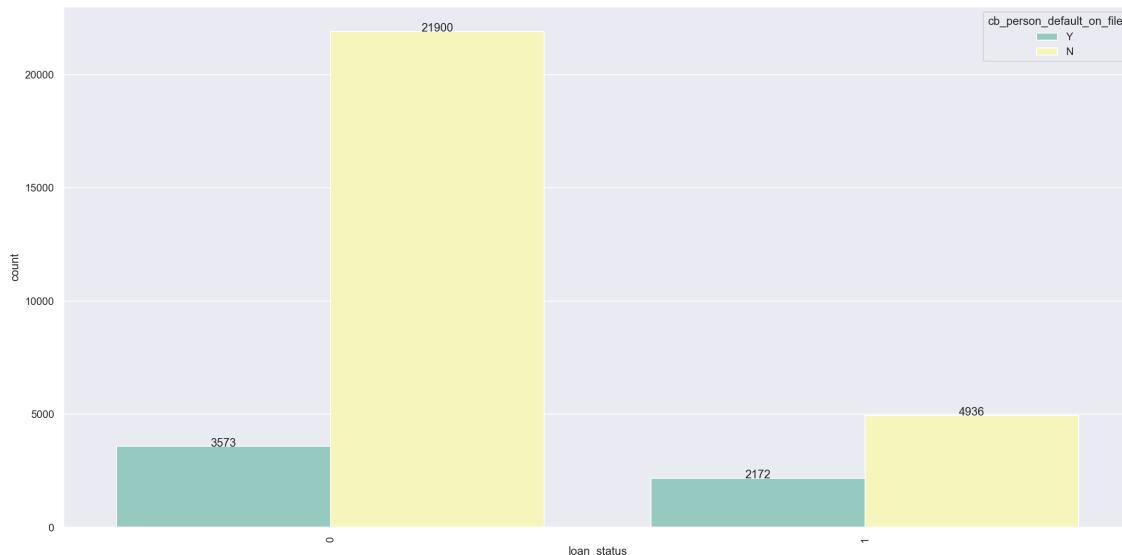
In [420]:

```
# Bi - Variate - Categorical - Categorical

fig, ax1 = plt.subplots(figsize=(20,10))
graph = sns.countplot(ax=ax1,x='loan_status',hue="cb_person_default_on_file",data=df, palette="Set1")
graph.set_xticklabels(graph.get_xticklabels(),rotation=90)
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1, height ,ha="center")

# Note

# 1. Categorical - Categorical ---> Target - loan_status
#                               ---> Predictor - cb_person_default_on_file
```



In [421]:

```
# Extracting the top 10 ratios

df_num["loan_percent_income"].nlargest(10) # ---> Top 10 in terms of values.

df_num["loan_percent_income"][:10] # ---> Top 10 in terms of frequencies.
```

Out[421]:

```
0    0.59
1    0.10
2    0.57
3    0.53
4    0.55
5    0.25
6    0.45
7    0.44
8    0.42
9    0.16
Name: loan_percent_income, dtype: float64
```

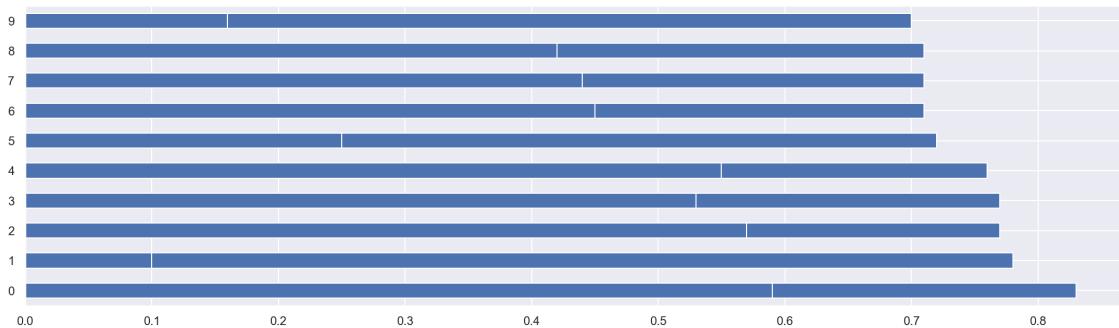
In [422]:

```
# Plotting the Loan - to - Income ratio

df_num["loan_percent_income"].nlargest(10).plot(kind ='barh') # ---> Top 10 in terms
df_num["loan_percent_income"][:10].plot(kind ='barh') # ---> Top 10 in terms of freq
```

Out[422]:

<AxesSubplot:>



In [423]:

```
cross_dei = pd.crosstab(index = df['loan_status'], columns = df['loan_percent_income'],
cross_dei
```

Out[423]:

loan_percent_income	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	...	
loan_status												
0	100.0	94.24	91.08	90.38	87.7	90.07	88.98	88.31	85.66	87.74	...	
1	0.0	5.76	8.92	9.62	12.3	9.93	11.02	11.69	14.34	12.26	...	

2 rows × 77 columns



In [424]:

```
# Bi - Variate - Categorical - Categorical

# Cross - tabulation in Absolute form

pd.crosstab(index = df_cat['loan_intent'], columns = df_cat['loan_grade'])

# Cross - tabulation in Relative form -
#                                     - row - wise
#                                     - columns - wise
#                                     - df - wise

# row - wise cross - tabulation

cross_index = pd.crosstab(index = df_cat['loan_intent'], columns = df_cat['loan_grade'],
                           normalize='index').round(4) * 100

cross_index
```

Out[424]:

	loan_grade	A	B	C	D	E	F	G
loan_intent								
DEBTCONSOLIDATION	33.63	31.08	19.99	11.51	2.76	0.83	0.19	
EDUCATION	33.69	31.40	21.01	10.18	2.87	0.71	0.14	
HOMEIMPROVEMENT	31.23	30.93	19.22	13.51	3.97	0.89	0.25	
MEDICAL	32.27	33.04	20.05	10.84	2.75	0.86	0.20	
PERSONAL	33.18	32.84	18.95	11.65	2.66	0.54	0.18	
VENTURE	33.80	32.72	19.30	10.16	3.11	0.66	0.24	

In [425]:

```
# Cross - tabulation in Relative form - percentage of all datapoints of df.
```

```
cross_df = pd.crosstab(index = df_cat['loan_intent'], columns = df_cat['loan_grade'],
margins = True, margins_name= "Total", normalize = True).round(4) * 100
cross_df
```

Out[425]:

loan_grade	A	B	C	D	E	F	G	Total
loan_intent								
DEBTCONSOLIDATION	5.38	4.97	3.20	1.84	0.44	0.13	0.03	16.00
EDUCATION	6.67	6.22	4.16	2.02	0.57	0.14	0.03	19.81
HOMEIMPROVEMENT	3.46	3.42	2.13	1.49	0.44	0.10	0.03	11.06
MEDICAL	6.01	6.16	3.74	2.02	0.51	0.16	0.04	18.63
PERSONAL	5.62	5.56	3.21	1.97	0.45	0.09	0.03	16.95
VENTURE	5.93	5.74	3.39	1.78	0.55	0.12	0.04	17.55
Total	33.08	32.08	19.82	11.13	2.96	0.74	0.20	100.00

In [426]:

```
cross_col = pd.crosstab(index = df_cat['loan_intent'], columns = df_num['loan_status'],
normalize='index').round(4) * 100
```

cross_col

Out[426]:

loan_status	0	1
loan_intent		
DEBTCONSOLIDATION	71.41	28.59
EDUCATION	82.78	17.22
HOMEIMPROVEMENT	73.90	26.10
MEDICAL	73.30	26.70
PERSONAL	80.11	19.89
VENTURE	85.19	14.81

In [427]:

```
cross_grade = pd.crosstab(index = df_cat['loan_grade'], columns = df_num['loan_status'],
                           normalize='index').round(4) * 100
```

```
cross_grade
```

Out[427]:

loan_status	0	1
loan_grade		
A	90.04	9.96
B	83.72	16.28
C	79.27	20.73
D	40.95	59.05
E	35.58	64.42
F	29.46	70.54
G	1.56	98.44

In [428]:

```
# Bi - Variate Analysis Default History vis-a-vis
```

```
pd.crosstab(index = df_cat['cb_person_default_on_file'], columns = df_num['loan_status'],
            normalize='index').round(4) * 100
```

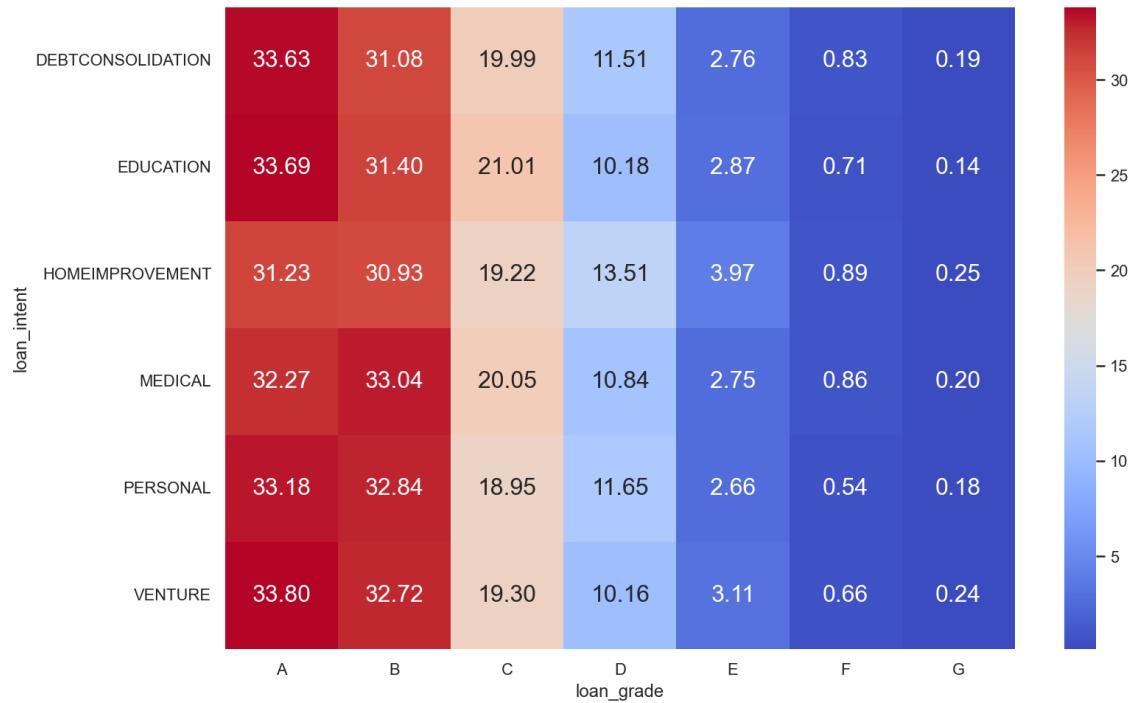
Out[428]:

loan_status	0	1
cb_person_default_on_file		
N	81.61	18.39
Y	62.19	37.81

In [429]:

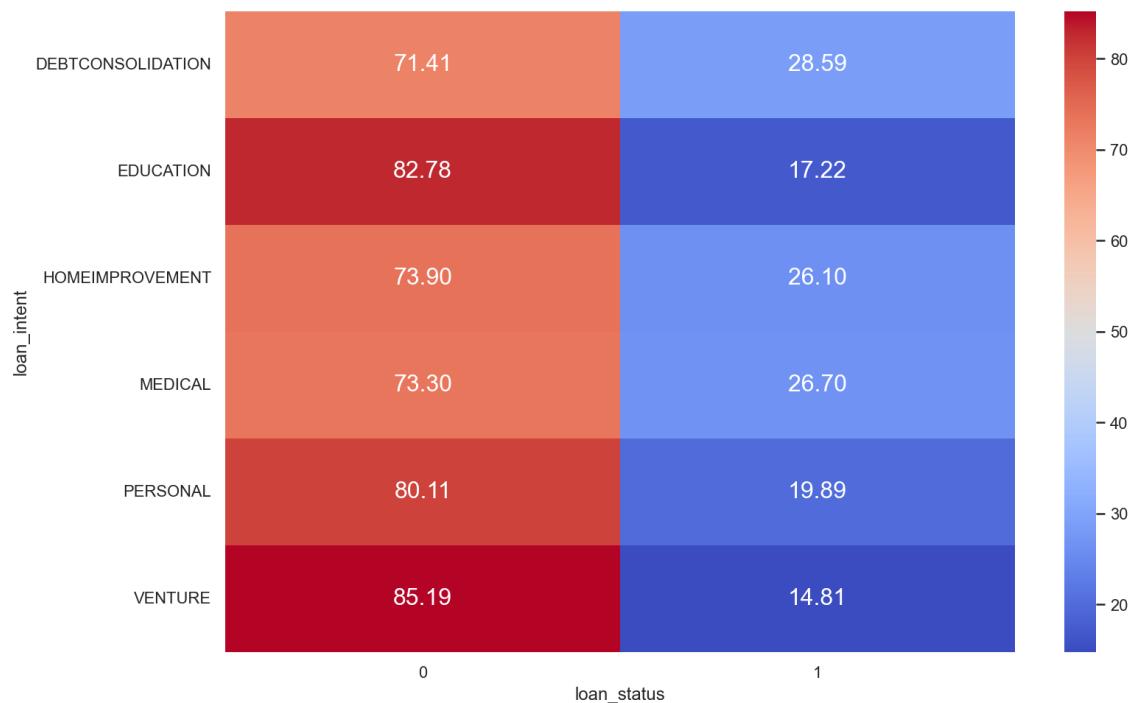
HeatMap of Cross - tabulation

```
plt.figure(figsize=(12,8))
sns.heatmap(cross_index, cmap='coolwarm', annot=True, annot_kws={"size": 16}, fmt=".2f")
```



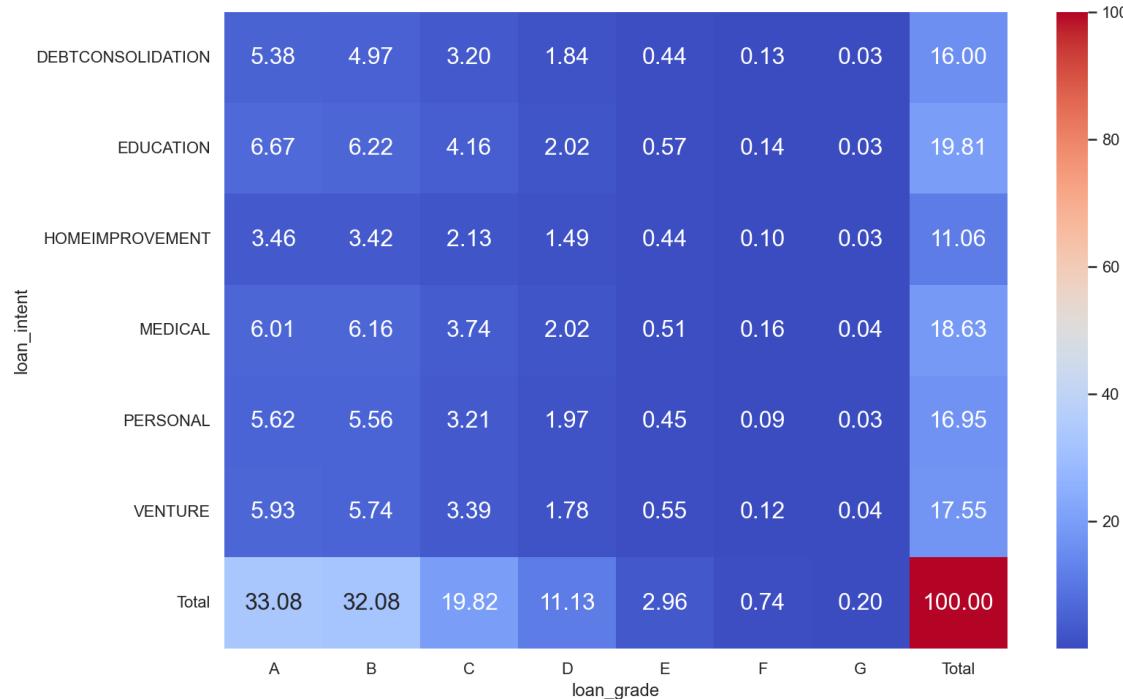
In [430]:

```
plt.figure(figsize=(12,8))
sns.heatmap(cross_col, cmap='coolwarm', annot=True, annot_kws={"size": 16}, fmt=".2f");
```



In [431]:

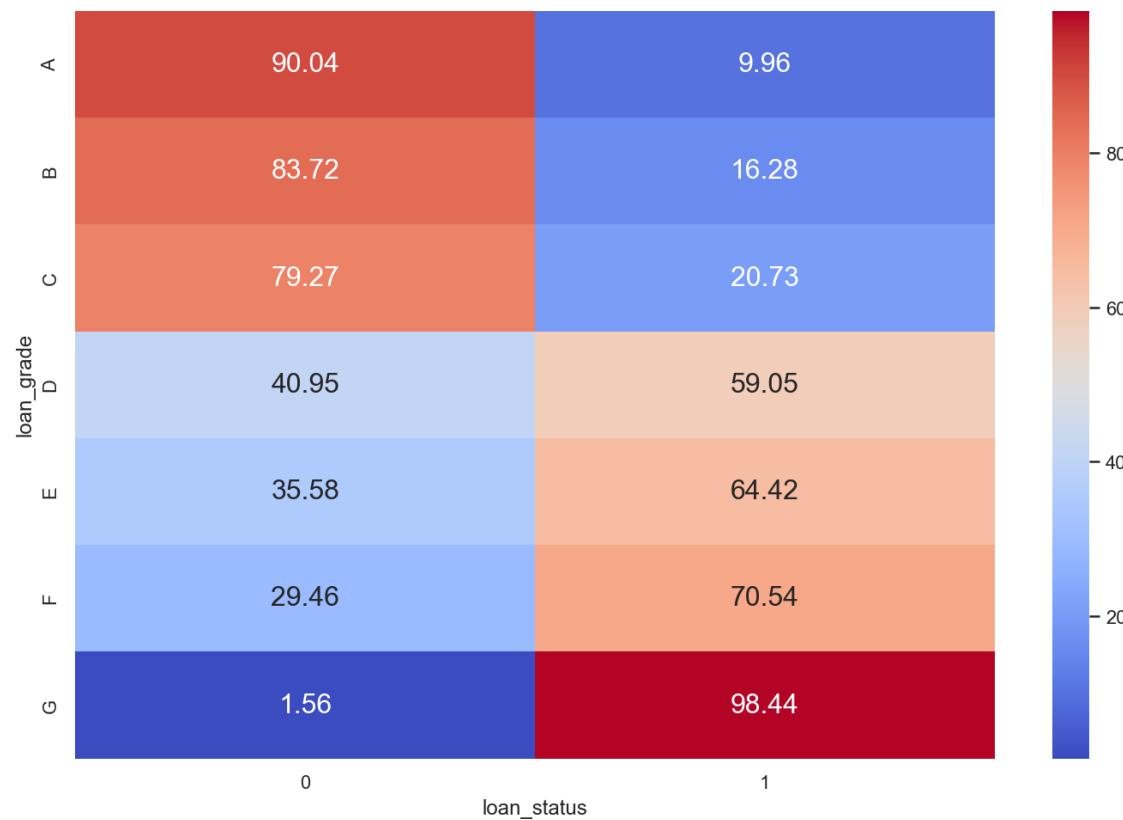
```
plt.figure(figsize=(12,8))
sns.heatmap(cross_df, cmap='coolwarm', annot=True, annot_kws={"size": 16}, fmt=".2f");
```



In [432]:

HeatMap of Cross - tabulation

```
plt.figure(figsize=(12,8))
sns.heatmap(cross_grade, cmap='coolwarm', annot=True, annot_kws={"size": 16}, fmt='.2f')
```



In [433]:

```
# Bi - Variate analysis Loan Status vis-a-vis Length of Credit History

cross_his = pd.crosstab(index = df['cb_person_cred_hist_length'], columns = df_num['loan_status'],
                        normalize='index').round(4) * 100

cross_his
```

Out[433]:

cb_person_cred_hist_length	loan_status	0	1
2	76.43	23.57	
3	77.72	22.28	
4	77.69	22.31	
5	79.37	20.63	
6	79.21	20.79	
7	80.06	19.94	
8	78.65	21.35	
9	79.16	20.84	
10	79.73	20.27	
11	79.91	20.09	
12	79.59	20.41	
13	80.14	19.86	
14	78.86	21.14	
15	78.26	21.74	
16	78.49	21.51	
17	80.84	19.16	
18	78.95	21.05	
19	76.19	23.81	
20	70.97	29.03	
21	70.00	30.00	
22	72.73	27.27	
23	86.36	13.64	
24	80.00	20.00	
25	72.22	27.78	
26	93.75	6.25	
27	63.64	36.36	
28	66.67	33.33	
29	57.14	42.86	
30	77.27	22.73	

In [434]:

```
cross_his.plot(kind = "line")
```

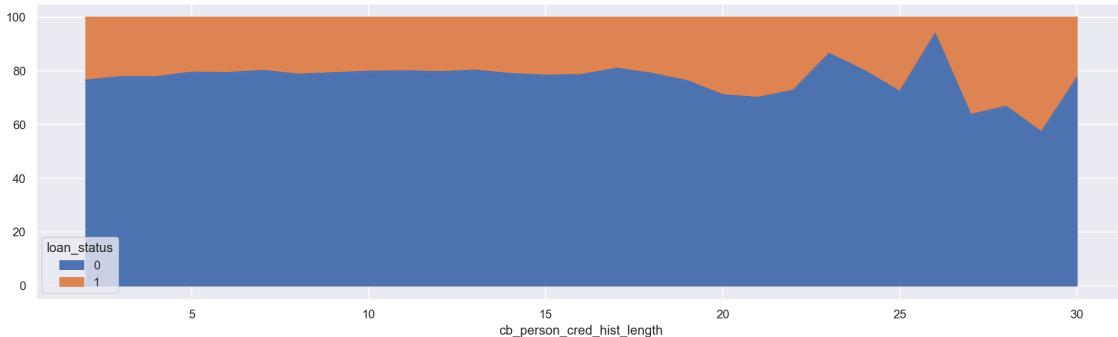
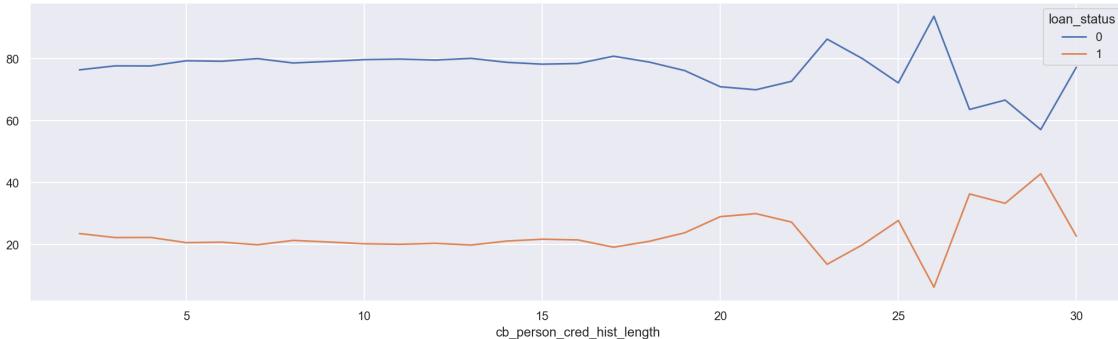
```
cross_his.plot(kind = "area")
```

Observations

1. For Lesser no. of years of credit history, the default rate is Low.
2. From 19 years onwards, it keeps increasing with exception 23rd and 26th year.
3. 26th year has Lowest default rate of mere 6.25%.

Out[434]:

```
<AxesSubplot:xlabel='cb_person_cred_hist_length'>
```



Bi - Variate - (Numerical - Numerical)

1. Correlation Matrix
2. Correlation Heatmap
3. Scatter Plot

In [435]:

Correlation Matrix

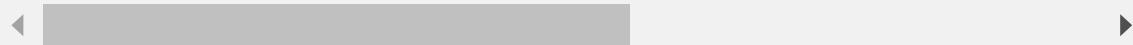
```
corr_matrix = df.corr(method = "pearson") # other Method Options = method : {'pearson', 'kendall', 'spearman'}
```

```
corr_matrix.style.background_gradient(cmap='Greens')
```

DataFrame.corrwith : Compute pairwise correlation with another DataFrame or Series.

Out[435]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
person_age	1.000000	0.173202	0.163106	0.050787				
person_income	0.173202	1.000000	0.134268	0.266820				
person_emp_length	0.163106	0.134268	1.000000	0.113082				
loan_amnt	0.050787	0.266820	0.113082	1.000000				
loan_int_rate	0.012580	0.000792	-0.056405	0.146813				
loan_status	-0.021629	-0.144449	-0.082489	0.105376				
loan_percent_income	-0.042411	-0.254471	-0.054111	0.572612				
cb_person_cred_hist_length	0.859133	0.117987	0.144699	0.041967				



In [436]:

DataFrame.corrwith : Compute pairwise correlation with Target and Other Predictor Variables

df.corrwith(df["loan_status"])

Out[436]:

```
person_age           -0.021629
person_income        -0.144449
person_emp_length    -0.082489
loan_amnt            0.105376
loan_int_rate         0.335133
loan_status           1.000000
loan_percent_income   0.379366
cb_person_cred_hist_length -0.015529
dtype: float64
```

In [437]:

```
# with the following function we can select highly correlated features.  
# It will remove the first feature that is correlated with anything other feature  
  
def correlation(dataset, threshold):  
    col_corr = set()                      # ---> Set of all the names of correlated columns  
    corr_matrix = dataset.corr()  
    for i in range(len(corr_matrix.columns)):  
        for j in range(i):  
            if abs(corr_matrix.iloc[i, j]) > threshold:      # ---> we are interested in absolute values  
                colname = corr_matrix.columns[i]              # ---> getting the name of column  
                col_corr.add(colname)  
    return col_corr
```

In [438]:

```
# Correlation - filtering out highly correlated features based on Threshold limit  
  
corr_features = correlation(df, 0.7)  
len(set(corr_features))
```

Out[438]:

1

In [439]:

```
corr_features
```

Out[439]:

```
{'cb_person_cred_hist_length'}
```

In [440]:

```
# Dropping Highly correlated Variables  
  
# df.drop(corr_features, axis=1)  # ----> Not dropping here since its is only EDA. Part of EDA
```

In [441]:

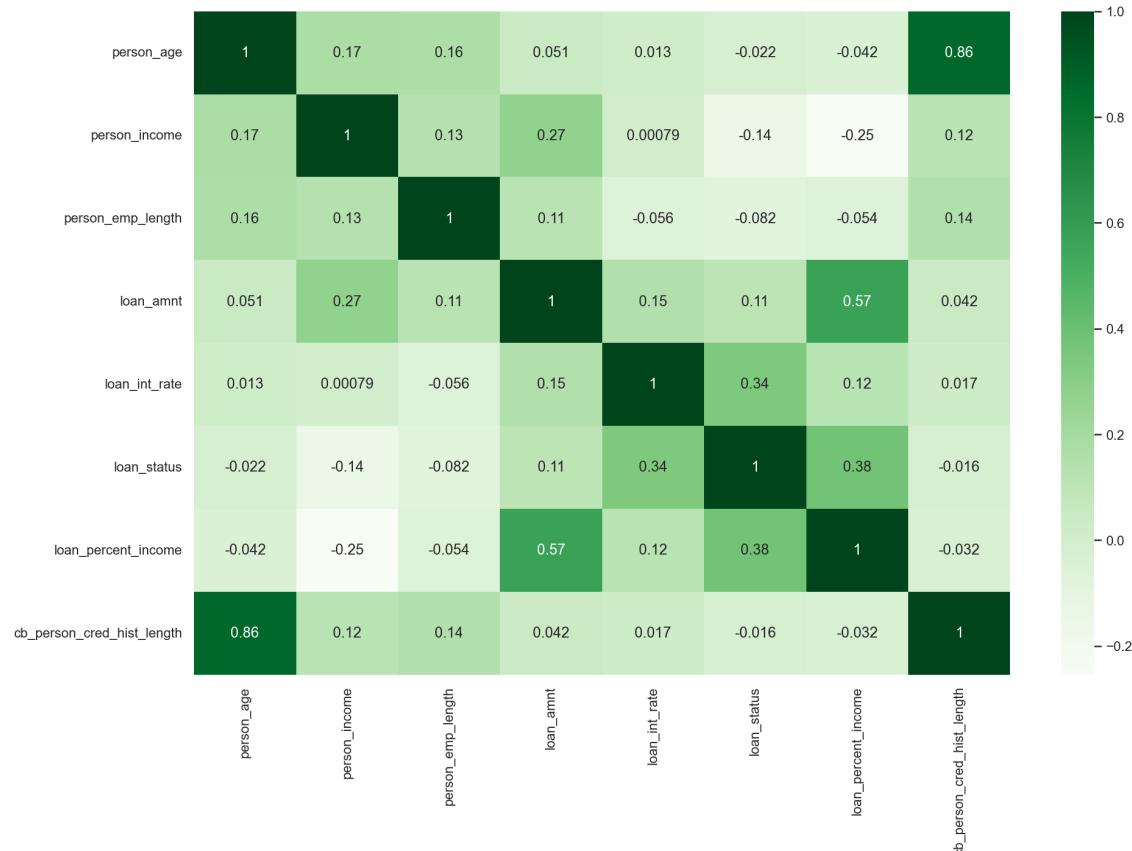
Correlation Heatmap

Heat Map

```
plt.figure(figsize=(15,10))
cor = df.corr()
sns.heatmap(cor, annot=True, cmap= 'Greens')      # ---> cmap options - viridis, Greens, coolwarm, etc.
plt.show()
```

Note:

1. Since no. of features are not very Large, Heatmap is readable. However, in case where there are many features, then sort the features according to importance and then create Heatmap. Utilize the sort function.



Bi - Variate - (Numerical - Categorical)

1. Bar Charts

In [442]:

```
# List of Categorical Variables -  
  
# 1. person_home_ownership      ---> 4  
# 2. Loan_intent                ---> 6  
# 3. Loan_grade                 ---> 7  
# 4. cb_person_default_on_file ---> 2  
  
# List of Numerical Variables -  
  
# 1. person_age  
# 2. person_income  
# 3. person_emp_length  
# 4. Loan_amnt  
# 5. Loan_int_rate  
# 6. Loan_percent_income  
# 7. cb_person_cred_hist_length  
  
# Target Variable  
  
# Loan_status ---> Binary Classification  
  
# Now,  
  
# Q.1 --> What's the average age of applicants having their own home ?  
# Q.2 --> What's the relationship between Income and Loan grade ?  
# Q.3 --> What's the relationship between Income and cb_person_default_on_file ?  
# Q.4 --> What's the relationship between Income and Loan amount ?
```

In [443]:

```
# Q.1 --> What's the average age of applicants having their own home ?
```

```
plt.figure(figsize=(10,5))
plt.xticks(rotation=75)
plt.title("Age by Ownership")

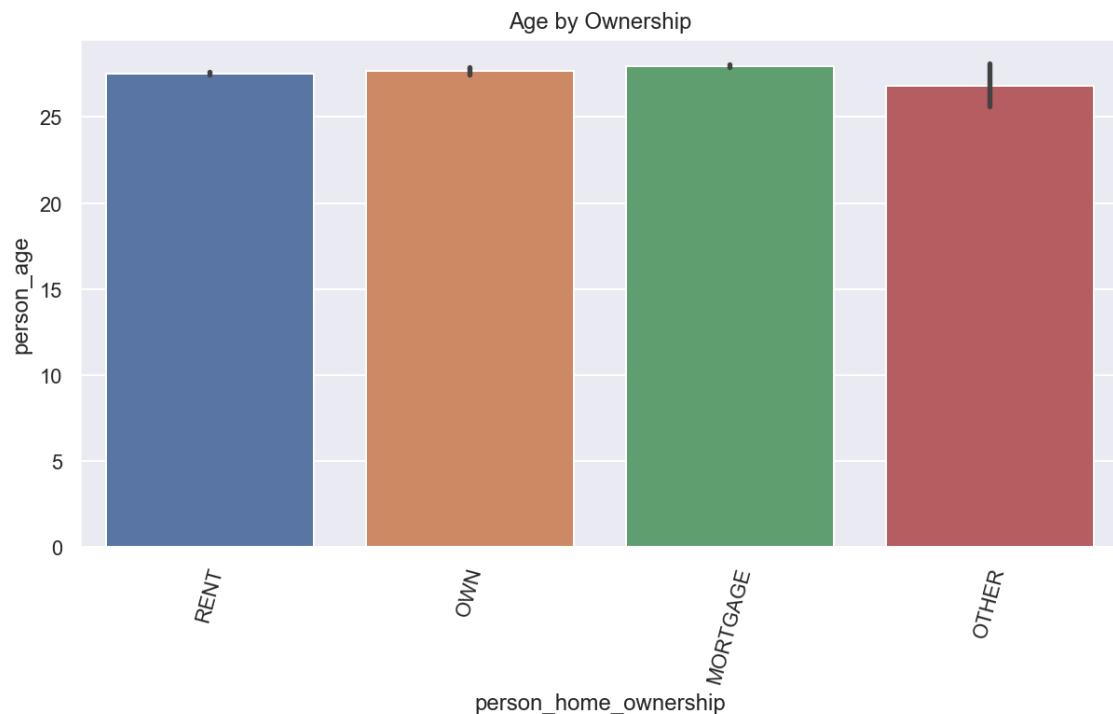
sns.barplot(x = df_cat["person_home_ownership"], y = df_num["person_age"])

# Observations

# 1. Average age is 27.73 years.
```

Out[443]:

```
<AxesSubplot:title={'center':'Age by Ownership'}, xlabel='person_home_ownership', ylabel='person_age'>
```



In [444]:

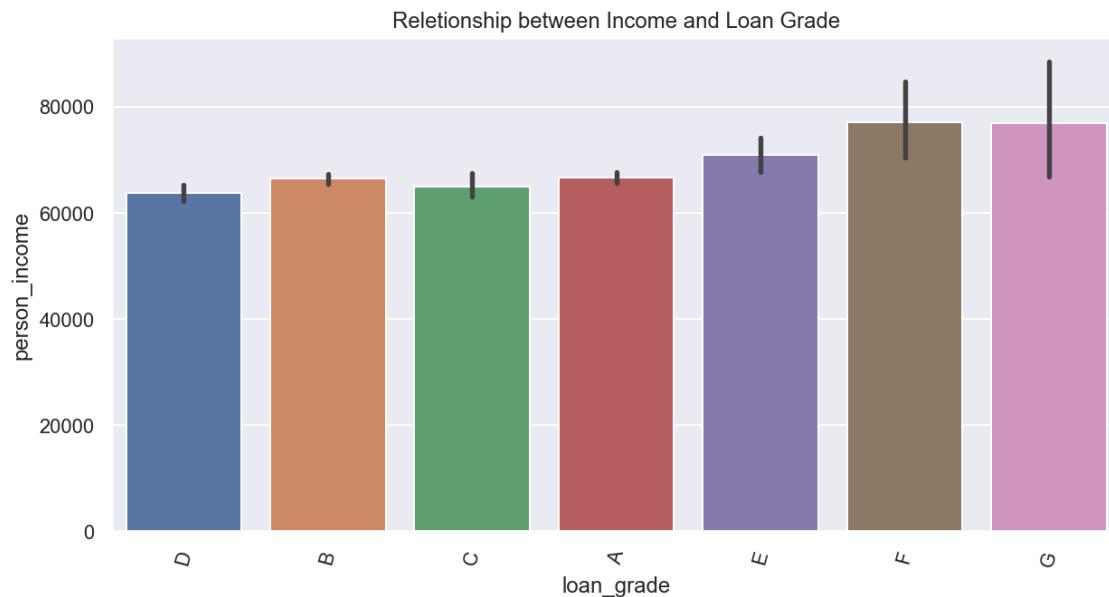
```
# Q.2 --> What's the relationship between Income and Loan grade ?
```

```
plt.figure(figsize=(10,5))
plt.xticks(rotation=75)
plt.title("Reletionship between Income and Loan Grade")

sns.barplot(x = df_cat["loan_grade"], y = df_num["person_income"])
```

Out[444]:

```
<AxesSubplot:title={'center':'Reletionship between Income and Loan Grade'}, xlabel='loan_grade', ylabel='person_income'>
```



In [445]:

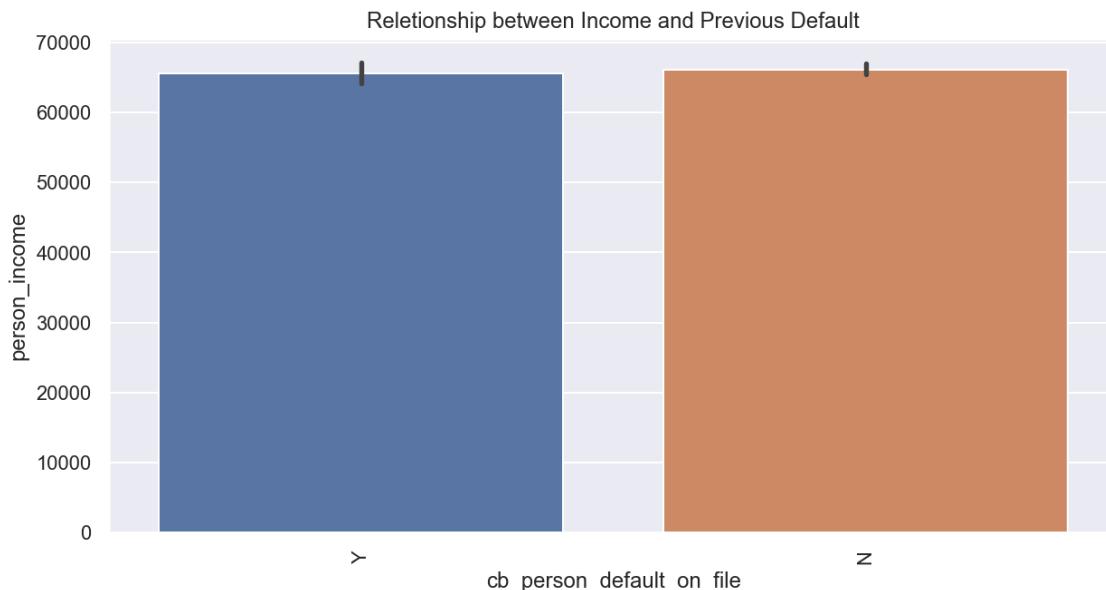
```
# Q.3 --> What's the relationship between Income and cb_person_default_on_file ?

plt.figure(figsize=(10,5))
plt.xticks(rotation=90)
plt.title("Relationship between Income and Previous Default")

sns.barplot(x = df_cat["cb_person_default_on_file"], y = df_num["person_income"])
```

Out[445]:

```
<AxesSubplot:title={'center':'Relationship between Income and Previous De
fault'}, xlabel='cb_person_default_on_file', ylabel='person_income'>
```



15. "Credit risk Prediction Using LGBM"

In [446]:

```
# Dealing with Null Values

df.isnull().sum()
```

Out[446]:

person_age	0
person_income	0
person_home_ownership	0
person_emp_length	895
loan_intent	0
loan_grade	0
loan_amnt	0
loan_int_rate	3116
loan_status	0
loan_percent_income	0
cb_person_default_on_file	0
cb_person_cred_hist_length	0
dtype: int64	

In [447]:

```
# drop values that are null
df.dropna(axis=0,inplace=True)
df.isnull().sum()
```

Out[447]:

```
person_age          0
person_income        0
person_home_ownership 0
person_emp_length    0
loan_intent          0
loan_grade           0
loan_amnt            0
loan_int_rate         0
loan_status           0
loan_percent_income   0
cb_person_default_on_file 0
cb_person_cred_hist_length 0
dtype: int64
```

In [448]:

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

Out[448]:

```
0    22435
1    6203
Name: loan_status, dtype: int64
```

In [449]:

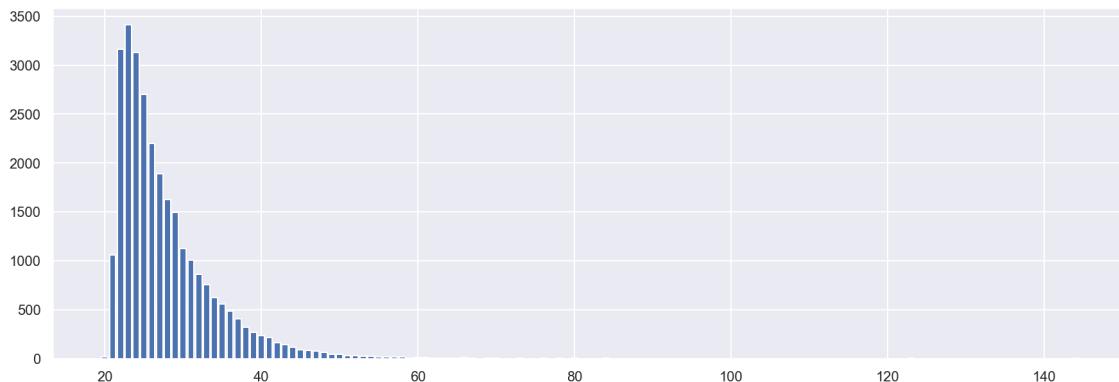
```
#person_age max 144 (issue)
#person_emp_Length max 123 (issue)
data.reset_index(inplace = True)
```

In [450]:

```
import matplotlib.pyplot as plt
verti = df['person_age'].value_counts().values
hori = df['person_age'].value_counts().index
fig = plt.figure(figsize = (15, 5))
plt.bar(hori, verti)
# after 80 it is rare
```

Out[450]:

<BarContainer object of 57 artists>



In [451]:

```
(df['person_age'].value_counts().values > 90).sum()
# dropping rows that have age greater than 90
```

Out[451]:

25

Creating New Features by Grouping

In [452]:

```
max_ = df['person_age'].max()
min_ = df['person_age'].min()
print(f"maximum {max_}")
print(f"minimum {min_}")
```

maximum 144
minimum 20

In [453]:

```
data = data.drop(data[data['person_age'] > 80].index, axis=0)
#dropping age above 80
```

In [454]:

```
# people with an age between x and y
def age_group(arr):
    lenarr = len(arr)
    for i in range(0,lenarr-1):
        next = arr[i]+1
        num_people = df['person_age'].between(next,arr[i+1]).sum()
        print(f'There are {num_people} people with an age between {next} and {arr[i+1]}.')

age_group([0 ,18, 26, 36, 46, 56, 66])

print(df.shape[0])
```

There are 0 people with an age between 1 and 18.
 There are 15682 people with an age between 19 and 26.
 There are 10422 people with an age between 27 and 36.
 There are 2054 people with an age between 37 and 46.
 There are 372 people with an age between 47 and 56.
 There are 87 people with an age between 57 and 66.
 28638

In [455]:

```
df['age_group'] = pd.cut(df['person_age'],
                         bins=[20, 26, 36, 46, 56, 66],
                         labels=['20-25', '26-35', '36-45', '46-55', '56-65'])
```

In [456]:

```
df['age_group'].dtype
```

Out[456]:

CategoricalDtype(categories=['20-25', '26-35', '36-45', '46-55', '56-65'], ordered=True)

In [457]:

```
df.head()
```

Out[457]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	I
0	22	59000		RENT	123.0	PERSONAL
1	21	9600		OWN	5.0	EDUCATION
2	25	9600		MORTGAGE	1.0	MEDICAL
3	23	65500		RENT	4.0	MEDICAL
4	24	54400		RENT	8.0	MEDICAL

In [458]:

df.shape

Out[458]:

(28638, 13)

Income Group

In [459]:

```
max_ = df['person_income'].max()
min_ = df['person_income'].min()

print(f"maximum {max_}")
print(f"minimum {min_}")
```

maximum 6000000
minimum 4000

In [460]:

```
# people with an income between x and y
def income_group(arr):
    lenarr = len(arr)
    for i in range(0,lenarr-1):
        next = arr[i]+1
        num_people = df['person_income'].between(next,arr[i+1]).sum()
        print(f'There are {num_people} people with an income between {next} and {arr[i+1]')

income_group([0, 25000, 50000, 75000, 100000,float('inf')))

print(df.shape[0])
```

There are 1972 people with an income between 1 and 25000.
 There are 10198 people with an income between 25001 and 50000.
 There are 8532 people with an income between 50001 and 75000.
 There are 4199 people with an income between 75001 and 100000.
 There are 3737 people with an income between 100001 and inf.

28638

In [461]:

```
df['income_group'] = pd.cut(df['person_income'],
                            bins=[0, 25000, 50000, 75000, 100000, float('inf')],
                            labels=['low', 'low-middle', 'middle', 'high-middle', 'high'])
```

In [462]:

```
df['income_group']
```

Out[462]:

```
0           middle
1            low
2            low
3           middle
4           middle
...
32576      middle
32577      high
32578  high-middle
32579      high
32580  low-middle
Name: income_group, Length: 28638, dtype: category
Categories (5, object): ['low' < 'low-middle' < 'middle' < 'high-middle'
< 'high']
```

LOAN AMOUNT

In [463]:

```
max_loan_amount = df['loan_amnt'].max()
min_loan_amount = df['loan_amnt'].min()

print(f"maximum {max_loan_amount}")
print(f"minimum {min_loan_amount}")
```

```
maximum 35000
minimum 500
```

In [464]:

```
# people with an income between x and y
def loan_amount_group(arr):
    lenarr = len(arr)
    for i in range(0, lenarr-1):
        next = arr[i]+1
        num_people = df['loan_amnt'].between(next, arr[i+1]).sum()
        print(f'There are {num_people} people with an loan_amount between {next} and {arr[i+1]}')


loan_amount_group([0, 5000, 10000, 15000, float('inf')])

print(df.shape[0])
```

There are 8184 people with an loan_amount between 1 and 5000.

There are 10610 people with an loan_amount between 5001 and 10000.

There are 5430 people with an loan_amount between 10001 and 15000.

There are 4414 people with an loan_amount between 15001 and inf.

28638

In [465]:

```
df['loan_amount_group'] = pd.cut(df['loan_amnt'],
                                  bins=[0, 5000, 10000, 15000, float('inf')],
                                  labels=['small', 'medium', 'large', 'very large'])
```

In [466]:

```
df['loan_amount_group']
```

Out[466]:

```
0      very large
1          small
2      medium
3      very large
4      very large
...
32576      medium
32577  very large
32578  very large
32579      large
32580      medium
Name: loan_amount_group, Length: 28638, dtype: category
Categories (4, object): ['small' < 'medium' < 'large' < 'very large']
```

In [467]:

```
df = df.drop(df[df['person_emp_length'] > 60].index, axis=0)
```

In [468]:

```
df.head()
```

Out[468]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	I
1	21	9600		OWN	5.0	EDUCATION
2	25	9600		MORTGAGE	1.0	MEDICAL
3	23	65500		RENT	4.0	MEDICAL
4	24	54400		RENT	8.0	MEDICAL
5	21	9900		OWN	2.0	VENTURE

In [469]:

```
df['person_home_ownership'].value_counts()
```

Out[469]:

RENT	14550
MORTGAGE	11800
OWN	2192
OTHER	94
Name: person_home_ownership, dtype: int64	

In [470]:

```
df.head()
```

Out[470]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	I
1	21	9600	OWN	5.0	EDUCATION	
2	25	9600	MORTGAGE	1.0	MEDICAL	
3	23	65500	RENT	4.0	MEDICAL	
4	24	54400	RENT	8.0	MEDICAL	
5	21	9900	OWN	2.0	VENTURE	

Creating New Features

In [471]:

```
# Create loan-to-income ratio
df['loan_to_income_ratio'] = df['loan_amnt'] / df['person_income']

# Create loan-to-employment length ratio
df['loan_to_emp_length_ratio'] = df['person_emp_length'] / df['loan_amnt']

# Create interest rate-to-loan amount ratio
df['int_rate_to_loan_amt_ratio'] = df['loan_int_rate'] / df['loan_amnt']
```

In [472]:

```
df.columns
```

Out[472]:

```
Index(['person_age', 'person_income', 'person_home_ownership',
       'person_emp_length', 'loan_intent', 'loan_grade', 'loan_amnt',
       'loan_int_rate', 'loan_status', 'loan_percent_income',
       'cb_person_default_on_file', 'cb_person_cred_hist_length', 'age_group',
       'income_group', 'loan_amount_group', 'loan_to_income_ratio',
       'loan_to_emp_length_ratio', 'int_rate_to_loan_amt_ratio'],
      dtype='object')
```

In [473]:

```
raw_data = df
```

Feature Engineering

In [474]:

```
col_list = ['person_age',#  
'person_income',#  
'person_home_ownership',#  
'person_emp_length',#  
'loan_intent', #  
'loan_grade',#  
'loan_amnt',#  
'loan_int_rate',#  
'loan_status',#  
'loan_percent_income',#  
'cb_person_default_on_file',#  
'cb_person_cred_hist_length',  
'age_group','income_group','loan_amount_group']  
  
drop_columns = []  
scale_cols = ['person_income','person_age','person_emp_length', 'loan_amnt','loan_int_ra  
'int_rate_to_loan_amt_ratio']  
ohe_columns = ['cb_person_default_on_file','loan_grade', 'person_home_ownership','loan_ir  
le_columns = []
```

In [475]:

```
df.columns
```

Out[475]:

```
Index(['person_age', 'person_income', 'person_home_ownership',  
       'person_emp_length', 'loan_intent', 'loan_grade', 'loan_amnt',  
       'loan_int_rate', 'loan_status', 'loan_percent_income',  
       'cb_person_default_on_file', 'cb_person_cred_hist_length', 'age_gr  
oup',  
       'income_group', 'loan_amount_group', 'loan_to_income_ratio',  
       'loan_to_emp_length_ratio', 'int_rate_to_loan_amt_ratio'],  
      dtype='object')
```

In [476]:

```
X = df.drop(['loan_status'], axis=1)  
Y = df['loan_status']
```

In [477]:

```
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.2,random_state=12)
```

In [478]:

```
print(x_train.shape,x_test.shape)
```

(22908, 17) (5728, 17)

In [479]:

```
x_train.reset_index(inplace = True)  
x_test.reset_index(inplace = True)
```

In [480]:

```
x_train.columns
```

Out[480]:

```
Index(['index', 'person_age', 'person_income', 'person_home_ownership',  
       'person_emp_length', 'loan_intent', 'loan_grade', 'loan_amnt',  
       'loan_int_rate', 'loan_percent_income', 'cb_person_default_on_file',  
       'cb_person_cred_hist_length', 'age_group', 'income_group',  
       'loan_amount_group', 'loan_to_income_ratio', 'loan_to_emp_length_ratio',  
       'int_rate_to_loan_amt_ratio'],  
      dtype='object')
```

In [481]:

```
from sklearn.preprocessing import OneHotEncoder  
ohe = OneHotEncoder()  
ohe.fit(x_train[ohe_columns])
```

Out[481]:

```
OneHotEncoder()
```

In [482]:

```
ohe.categories_
```

Out[482]:

```
[array(['N', 'Y'], dtype=object),  
 array(['A', 'B', 'C', 'D', 'E', 'F', 'G'], dtype=object),  
 array(['MORTGAGE', 'OTHER', 'OWN', 'RENT'], dtype=object),  
 array(['DEBTCONSOLIDATION', 'EDUCATION', 'HOMEIMPROVEMENT', 'MEDICAL',  
       'PERSONAL', 'VENTURE'], dtype=object),  
 array(['high', 'high-middle', 'low', 'low-middle', 'middle'], dtype=object),  
 array(['20-25', '26-35', '36-45', '46-55', '56-65', nan], dtype=object),  
 array(['large', 'medium', 'small', 'very large'], dtype=object)]
```

In [483]:

```
merge_ohe_col = np.concatenate((ohe.categories_[0],  
                                ohe.categories_[1],  
                                ohe.categories_[2],  
                                ohe.categories_[3],  
                                ohe.categories_[4],  
                                ohe.categories_[5],  
                                ohe.categories_[6],))
```

In [484]:

```
merge_ohe_col
```

Out[484]:

```
array(['N', 'Y', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'MORTGAGE', 'OTHER',  
      'OWN', 'RENT', 'DEBTCONSOLIDATION', 'EDUCATION', 'HOMEIMPROVEMEN  
T',  
      'MEDICAL', 'PERSONAL', 'VENTURE', 'high', 'high-middle', 'low',  
      'low-middle', 'middle', '20-25', '26-35', '36-45', '46-55',  
      '56-65', nan, 'large', 'medium', 'small', 'very large'],  
      dtype=object)
```

In [485]:

```
ohe_data = pd.DataFrame(ohe.transform(x_train[ohe_columns]).toarray(), columns=merge_ohe_
```



```
ohe_data2 = pd.DataFrame(ohe.transform(x_test[ohe_columns]).toarray(), columns=merge_ohe_c
```

In [486]:

```
X_new = pd.concat([ohe_data, x_train], axis=1)  
X_new = X_new.drop(ohe_columns, axis=1)  
  
X_new_test = pd.concat([ohe_data2, x_test], axis=1)  
X_new_test = X_new_test.drop(ohe_columns, axis=1)
```

In [487]:

```
X_new.head()  
X_new.shape
```

Out[487]:

```
(22908, 45)
```

In [488]:

```
X_new_test.head()
```

Out[488]:

	N	Y	A	B	C	D	E	F	G	MORTGAGE	...	person_age	person_income	p
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	47	38400	
1	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	26	152229	
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	34	42000	
3	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	26	53000	
4	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	21	69000	

5 rows × 45 columns

In [489]:

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

scale_cols = ['person_income', 'person_age', 'person_emp_length', 'loan_amnt', 'loan_int_rate', 'int_rate_to_loan_amt_ratio']

uniform_col = []

normal_col = ['person_income', 'person_age', 'person_emp_length', 'loan_amnt', 'loan_int_rate', 'int_rate_to_loan_amt_ratio']

bimodal_col = []

scaler_uniform = MinMaxScaler()
#X_new.loc[:, uniform_col] = scaler_uniform.fit_transform(X_new.loc[:, uniform_col])
#X_new_test.loc[:, uniform_col] = scaler_uniform.transform(X_new_test.loc[:, uniform_col])

scaler_normal = StandardScaler()
X_new.loc[:, normal_col] = scaler_normal.fit_transform(X_new.loc[:, normal_col])
X_new_test.loc[:, normal_col] = scaler_normal.transform(X_new_test.loc[:, normal_col])

scaler_bimodal = RobustScaler()
#X_new.loc[:, bimodal_col] = scaler_bimodal.fit_transform(X_new.loc[:, bimodal_col])
#X_new_test.loc[:, bimodal_col] = scaler_bimodal.transform(X_new_test.loc[:, bimodal_col])
```

In [490]:

```
X_new_test.head()
```

Out[490]:

	N	Y	A	B	C	D	E	F	G	MORTGAGE	...	person_age	person_income	p
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	...	3.031277	-0.435272	
1	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.271736	1.313861	
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.986555	-0.379954	
3	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	-0.271736	-0.210924	
4	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-1.058167	0.034937	

5 rows × 45 columns

```
X_new_test.head()
```

Modeling

In [491]:

```
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import GradientBoostingClassifier
! pip install xgboost
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
! pip install catboost
from catboost import CatBoostClassifier
! pip install lightgbm
import lightgbm as lgb
```

Requirement already satisfied: xgboost in c:\users\bolla\anaconda3\lib\site-packages (1.7.6)

DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number. pip 23.3 will enforce this behaviour change. A possible replacement is to upgrade to a newer version of pyodbc or contact the author to suggest that they release a version with a conforming version number. Discussion can be found at <https://github.com/pypa/pip/issues/12063> (<https://github.com/pypa/pip/issues/12063>)

Requirement already satisfied: numpy in c:\users\bolla\anaconda3\lib\site-packages (from xgboost) (1.22.4)

Requirement already satisfied: scipy in c:\users\bolla\anaconda3\lib\site-packages (from xgboost) (1.7.1)

Requirement already satisfied: catboost in c:\users\bolla\anaconda3\lib\site-packages (1.2)

DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number. pip 23.3 will enforce this behaviour change. A possible replacement is to upgrade to a newer version of pyodbc or contact the author to suggest that they release a version with a conforming version number. Discussion can be found at <https://github.com/pypa/pip/issues/12063> (<https://github.com/pypa/pip/issues/12063>)

```
Requirement already satisfied: graphviz in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (3.4.3)
Requirement already satisfied: numpy>=1.16.0 in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (1.22.4)
Requirement already satisfied: pandas>=0.24 in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (1.3.4)
Requirement already satisfied: scipy in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (1.7.1)
Requirement already satisfied: plotly in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (5.15.0)
Requirement already satisfied: six in c:\users\bolla\anaconda3\lib\site-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\bolla\anaconda3\lib\site-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\bolla\anaconda3\lib\site-packages (from pandas>=0.24->catboost) (2021.3)
Requirement already satisfied: cycler>=0.10 in c:\users\bolla\anaconda3\lib\site-packages (from matplotlib->catboost) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\bolla\anaconda3\lib\site-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\bolla\anaconda3\lib\site-packages (from matplotlib->catboost) (8.4.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\bolla\anaconda3\lib\site-packages (from matplotlib->catboost) (3.0.4)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\bolla\anaconda3\lib\site-packages (from plotly->catboost) (8.2.2)
Requirement already satisfied: packaging in c:\users\bolla\anaconda3\lib\site-packages (from plotly->catboost) (23.1)
In [49]: Requirement already satisfied: lightgbm in c:\users\bolla\anaconda3\lib\site-packages (4.0.0)
knc = KNeighborsClassifier() #algorithm='ball_tree', leaf_size=10, n_neighbors=18, p=1,
mnb = MultinomialNB()
DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number.
dtf = DecisionTreeClassifier()
DEPRECATION: pyodbc 23.3 will enforce this behaviour change. A possible replacement is to
upgrade to a newer version of pyodbc or contact the author to suggest tha
t they release a version with a conforming version number. Discussion can
be found at https://github.com/pypa/pip/issues/12063 (https://github.com/pypa/pip/issues/12063)
etc = ExtraTreesClassifier()
gbdt = GradientBoostingClassifier()
Requirement already satisfied: numpy in c:\users\bolla\anaconda3\lib\site-packages (from lightgbm)
lightgbm = LightGBMClassifier(foldsamplebytree=0.737803019867917, learning_rate=0.007929963347654646, max_depth=11, min_child_weight=1e-05, num_leaves=31, num_threads=4, objective='regression', random_state=42, tree_learner='serial')
#best #colsample_bytree= 0.737803019867917, Learning_rate= 0.007929963347654646, max_depth=11, min_child_weight= 1e-05, num_leaves=31, num_threads=4, objective='regression', random_state=42, tree_learner='serial'
```

In [493]:

```
clfs = {
    'KN' : knc,
    'xgb':xgb,
    'cat':cat,
    'lgb':lgb
}
```

In [494]:

```
# clfs = {
#     'SVC' : svc,
#     'KN' : knc,
#     'NB': mnb,
#     'DT': dtc,
#     'LR': lrc,
#     'RF': rfc,
#     'AdaBoost': abc,
#     'BgC': bc,
#     'ETC': etc,
#     'GBDT':gbdt,
#     'xgb':xgb
# }
```

In [495]:

```
from sklearn.metrics import precision_score,accuracy_score
def train_classifier(clf,X_train,y_train,X_test,y_test):
    clf.fit(X_train,y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    precision = precision_score(y_test,y_pred)

    return accuracy,precision
```

In [496]:

```
X_new = X_new.drop(columns=[col for col in X_new.columns if pd.isna(col)], axis=1)
X_new_test = X_new_test.drop(columns=[col for col in X_new_test.columns if pd.isna(col)])
```

In [497]:

```
accuracy_scores = []
precision_scores = []

for name,clf in clfs.items():

    current_accuracy,current_precision = train_classifier(clf, X_new,y_train,X_new_test,
    
    print("For ",name)
    print("Accuracy - ",current_accuracy)
    print("Precision - ",current_precision)

    accuracy_scores.append(current_accuracy)
    precision_scores.append(current_precision)
```

```
For KN
Accuracy - 0.7840432960893855
Precision - 0.5171232876712328
For xgb
Accuracy - 0.9350558659217877
Precision - 0.9643231899265478
For cat
Accuracy - 0.9362779329608939
Precision - 0.9764957264957265
[LightGBM] [Warning] Found whitespace in feature_names, replace with underscores
[LightGBM] [Info] Number of positive: 4945, number of negative: 17963
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006835 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2016
[LightGBM] [Info] Number of data points in the train set: 22908, number of used features: 44
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.215863 -> initscore=-1.289937
[LightGBM] [Info] Start training from score -1.289937
For lgb
Accuracy - 0.8950768156424581
Precision - 1.0
```

In [498]:

```
from sklearn.ensemble import VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier

knn = KNeighborsClassifier()
#xgb = XGBClassifier()
cat = CatBoostClassifier(verbose=0)
lgb = LGBMClassifier()

ensemble = VotingClassifier(estimators=[('knn', knn), ('cat', cat), ('lgb', lgb)], voting='soft')
ensemble_accuracy, ensemble_precision = train_classifier(ensemble, X_new, y_train, X_new)

print("For ensemble")
print("Accuracy - ", ensemble_accuracy)
print("Precision - ", ensemble_precision)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underscores
[LightGBM] [Info] Number of positive: 4945, number of negative: 17963
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.006703 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2016
[LightGBM] [Info] Number of data points in the train set: 22908, number of used features: 44
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.215863 -> initscore=-1.289937
[LightGBM] [Info] Start training from score -1.289937
For ensemble
Accuracy - 0.9305167597765364
Precision - 0.986409966024915
```

Hyperparameter Tuning

In [499]:

```
from lightgbm import LGBMRegressor
# Evaluation and bayesian optimization
from sklearn.metrics import make_scorer, mean_absolute_error
from sklearn.metrics import mean_squared_error as MSE
! pip install hyperopt
from hyperopt import hp, fmin, tpe
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.model_selection import KFold, cross_val_score
from bayes_opt import BayesianOptimization
from lightgbm import LGBMClassifier
```

Requirement already satisfied: hyperopt in c:\users\bolla\anaconda3\lib\site-packages (0.2.7)

DEPRECATION: pyodbc 4.0.0-unsupported has a non-standard version number. pip 23.3 will enforce this behaviour change. A possible replacement is to upgrade to a newer version of pyodbc or contact the author to suggest that they release a version with a conforming version number. Discussion can be found at <https://github.com/pypa/pip/issues/12063> (<https://github.com/pypa/pip/issues/12063>)

Requirement already satisfied: numpy in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (1.22.4)

Requirement already satisfied: scipy in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (1.7.1)

Requirement already satisfied: six in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (1.16.0)

Requirement already satisfied: networkx>=2.2 in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (2.6.3)

Requirement already satisfied: future in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (0.18.2)

Requirement already satisfied: tqdm in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (4.62.3)

Requirement already satisfied: cloudpickle in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (2.0.0)

Requirement already satisfied: py4j in c:\users\bolla\anaconda3\lib\site-packages (from hyperopt) (0.10.9.7)

Requirement already satisfied: colorama in c:\users\bolla\anaconda3\lib\site-packages (from tqdm->hyperopt) (0.4.6)

In [500]:

```
# Hyperparameter-tuning: Bayesian Optimization, bayes_opt
def lgbm_clf_bo(num_leaves, max_depth, learning_rate, min_child_weight, subsample, colsample_bytree):
    params_lgbm = {'objective': 'binary'}
    params_lgbm['num_leaves'] = round((2**round(max_depth))*num_leaves)
    params_lgbm['max_depth'] = round(max_depth)
    params_lgbm['learning_rate'] = learning_rate
    params_lgbm['min_child_weight'] = min_child_weight
    params_lgbm['subsample'] = subsample
    params_lgbm['colsample_bytree'] = colsample_bytree

    scores = cross_val_score(LGBMClassifier(random_state=12, **params_lgbm),
                             X_new, y_train, scoring='accuracy', cv=10).mean()
    return scores

# Set parameters distribution
params_lgbm = {
    'num_leaves':(0.5,0.9),
    'max_depth': (3, 15),
    'learning_rate': (0.005, 0.3),
    'min_child_weight':(1e-6, 1e-1),
    'subsample':(0.5, 1),
    'colsample_bytree':(0.5, 1)
}

# Run Bayesian Optimization
lgbm_bo = BayesianOptimization(lgbm_clf_bo, params_lgbm)
lgbm_bo.maximize(init_points=2, n_iter=20)

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Found whitespace in feature_names, replace with underscores
[LightGBM] [Info] Number of positive: 4450, number of negative: 16167
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the over
-ride is taking ~ 0.00000 seconds
```

In [501]:

```
# Best hyperparameters
params_lgbm = lgbm_bo.max['params']
params_lgbm['max_depth'] = round(params_lgbm['max_depth'])
params_lgbm['num_leaves'] = round((2**round(params_lgbm['max_depth'])))*params_lgbm['num_leaves']
```

Out[501]:

```
{'colsample_bytree': 1.0,
 'learning_rate': 0.3,
 'max_depth': 8,
 'min_child_weight': 0.1,
 'num_leaves': 230,
 'subsample': 0.5}
```

Feature Importance

In [502]:

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor()
from sklearn.feature_selection import RFE

clf = RandomForestRegressor()

clf.fit(X_new,y_train)

feature_scores = pd.Series(clf.feature_importances_, index=X_new.columns).sort_values(as
feature_scores
```

Out[502]:

loan_to_income_ratio	0.234596
loan_int_rate	0.156334
RENT	0.128660
person_income	0.099040
index	0.045728
person_emp_length	0.037581
MEDICAL	0.033397
int_rate_to_loan_amt_ratio	0.029444
loan_to_emp_length_ratio	0.029391
DEBTCONSOLIDATION	0.028264
person_age	0.026387
C	0.025579
D	0.018315
loan_percent_income	0.017144
cb_person_cred_hist_length	0.014648
loan_amnt	0.014144
HOMEIMPROVEMENT	0.012180
OWN	0.008802
MORTGAGE	0.004468
EDUCATION	0.003941
PERSONAL	0.003687
G	0.003365
VENTURE	0.003331
medium	0.002038
26-35	0.002023
Y	0.001930
N	0.001916
E	0.001506
OTHER	0.001487
middle	0.001390
B	0.001320
large	0.001209
low-middle	0.001199
high-middle	0.000981
20-25	0.000782
36-45	0.000645
F	0.000593
small	0.000558
A	0.000486
46-55	0.000424
very large	0.000413
high	0.000329
low	0.000266
56-65	0.000082

dtype: float64

In [532]:

```
!pip install pycaret
```

In [533]:

```
from pycaret.classification import *
```

In [504]:

```
raw_data.head()
```

Out[504]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	I
1	21	9600	OWN	5.0	EDUCATION	
2	25	9600	MORTGAGE	1.0	MEDICAL	
3	23	65500	RENT	4.0	MEDICAL	
4	24	54400	RENT	8.0	MEDICAL	
5	21	9900	OWN	2.0	VENTURE	



In [534]:

```
model = setup(data = raw_data, target = 'loan_status', session_id=123)
```

In [535]:

```
compare_models()
```

In [536]:

```
model = setup(data = data_original, target = 'loan_status', session_id=123)
```

In [537]:

```
best = compare_models()
```

In [538]:

```
evaluate_model(best)
```

In [539]:

```
pred_holdout = predict_model(best)
```

In [540]:

```
!pip install autogluon
```

In [511]:

```
from sklearn.model_selection import train_test_split

#split dataframe into train and test sets
train, test = train_test_split(data, test_size=0.2, random_state=12)

#print size of each set
print(train.shape, test.shape)
```

(26059, 9) (6515, 9)

In [512]:

train.head()

Out[512]:

	index	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	lo
22020	22020	28	65000	3.0	7000	8.49	
6424	6424	22	49920	1.0	10000	12.53	
11634	11634	26	75000	10.0	7200	NaN	
10057	10057	24	65000	8.0	2500	6.39	
18125	18125	31	86000	1.0	25000	16.00	

◀ ▶

In [542]:

from autogluon.tabular import TabularDataset, TabularPredictor

```
predictor = TabularPredictor(label='loan_status').fit(train_data=train)
predictions = predictor.predict(test)
```

In [543]:

predictor.evaluate(test, silent=True)

In [544]:

predictor.leaderboard(test, silent=True)

16. Detailed Insights (variable - wise)

Age

1. Age is numerical variable.
2. There are no Missing values.
3. Age variable ranges from 20 to 144 which definitely creates suspicion about the accuracy of data.

4. Mean age of 27.74 indicates data is highly right skewed. skew = 2.58.
5. As per box – plot proximity rule, there are 1494 Outliers. (4.58%)
6. All outliers are on higher side.
7. As per Box-plot, Outliers ranges from 41 to 144.
8. Most Outliers lie between age 41 to 70.
9. As per Scatter plot, the variable Age has non – linear correlation with all other numerical variables except Credit history length. There is no fixed pattern.

Annual Income

1. Annual Income is again a numerical variable.
2. The problem statement is silent about the currency. We assume the currency as INR.
3. There are no Missing values.
4. Income Variable ranges from 4000 to 6000000. Huge range and must be Standardized. Annual Income of Rs. 4000 must not be eligible for any type of loan.
5. The distribution of Annual Income is highly right skewed.
6. Mean Annual Income is Rs. 66075 and Standard Deviation is 61983. This seems to be a great variation.
7. As per box – plot proximity rule, there are 1484 Outliers (4.55%).
8. All outliers are on higher side.
9. As per Box-plot, Outliers ranges from to
10. As per Scatter plot, the variable Annual Income has non – linear correlation with all other numerical variables. There is no fixed pattern.

Home ownership

1. Home Ownership is a Nominal Categorical Variable.
2. There are 4 unique values in it namely, RENT - 50.48% MORTGAGE - 41.26% OWN - 7.93% OTHER- 0.33%
3. The category Other is not defined and is very rare and is safe to ignore.
4. There are no Missing Values in the Variable Home Ownership.
5. From Bank's perspective, the borrower with Own House are considered to be less risky. However, it should not be the sole consideration.
6. Analyse the above categories with Target Variable i.e. Loan Status.

Employment length (in years)

1. No. of years of employment is a numerical variable.
2. This variable has Missing values. The magnitude of missing is around 2.75%.
3. It seems the missing values are not because of any error.
4. It's a classic case of Structurally Missing Data. It is missing because it should not exist. The reason for missingness is perhaps some borrowers have their Own business and is not employed.
5. Considering the reason of missingness and magnitude of missing data, the suggested course of action is List – wise deletion of missing values.
6. The maximum employment length is 123 years, an indication of error.
7. The distribution of years of employment is again right skewed.
8. As per IQR proximity rule, there are 853 Outliers. (2.70%)

Loan Intention

1. Loan intention is a Nominal Categorical Variable.

2. There are 6 unique values in it namely, EDUCATION - 19.81% MEDICAL - 18.63% VENTURE - 17.55% PERSONAL - 16.94% DEBT CONSOLIDATION - 16.00% HOME IMPROVEMENT - 11.06%
3. As per my experience, the default rate in Personal and Debt Consolidation is highest.
4. There are no Missing values in Loan intention variable.

Loan Grade

1. *prima facie*, this variable is Ordinal Categorical Variable.
2. There are no details given about this Variable like on what criteria grades are allotted. Here, we are assuming, higher the grade, lower is the chances of Default. Going by that logic, borrowers categorized as Grade – A must have low probability of getting Default and vice – versa. We will check the percentage of Default for every grade.
3. There are no Missing Values.
4. There are 7 unique values in it namely, A - 33.08% B - 32.08% C - 19.82% D - 11.13% E - 2.96% F - 0.74% G - 0.20%

Loan Amount

1. Loan amount is a Numerical Variable.
2. There are no Missing Values.
3. Loan amount ranges from Rs. 500 to Rs. 35000.
4. Loan amount of Rs. 500 is too low to believe on genuineness of data.
5. Loan amount distribution is asymmetrical and skewed.
6. As per IQR Proximity Rule, there are 1689 Outliers in the variable Loan Amount. (5.18%).
7. As per Box – Plot, most of the Outliers are between 23,000 to 27,000.
8. As per Scatter Plot, Loan amount is in non – linear relation with other variables.

Interest rate

1. Interest rate is a Numerical Variable.
2. Though explicitly not mentioned anywhere about the Unit but it is in terms of percentage (%).
3. Interest rates ranges from 5.42% to 23.2%.
4. Technically, this variable should be Categorical with fewer no. of interest rates, if it belongs to One Bank or FI.
5. There are 29465 values and 348 unique values.
6. There are too many Missing values 3116 (9.56%).
7. There are mere 6 Outliers in Intr. rate variable.
8. Nature of Variable and Magnitude of Missing values makes this variable a tricky one.
9. As per my Domain Knowledge and Dendrogram, this variable is important and have high predictive power.
10. So, dropping the variable is not an option.
11. As per Dendrogram, this variable is highly co - related to variable Employment length and variable Past Default History.
12. Suggested course of action is – i. Drop Outliers ii. Impute Missing values iii. Filter top 10 or 20 high frequency int. rate iv. Analyse the above with Target Variable.

Loan - Income Percent

1. Loan to Income is a Numerical Variable.
2. It is in relative term. (In terms of ratio).
3. In finance jargon, it is called as "Debt to Income ratio". (DTI).

4. Good / Ideal DTI ratio ---> The percentage of DTI ratio may vary from lender to lender. However, in general, a DTI ratio of up to 40% may be considered suitable for getting a loan approved. DTI of 21% - 35% is considered as very good.
5. Lesser the ratio, lesser are the chances of Default and vice – versa.
6. There are no Missing values.
7. The ratio ranges from 0 to 0.83.
8. The distribution is asymmetrical and skewed.
9. As per IQR proximity rules, there are 651 Outliers (2%).
10. There are just 77 unique values. Technically, it is Categorical and must analyse that way only.
11. By analysing, the loan status with DTI ratio, it supports the above rationale. As the DTI increases, the rate of default also increases and vice – versa.

History of Default

1. History of Default is Nominal Categorical Variable.
2. There are no Missing values.
3. There are only 2 unique values namely,
4. Yes – 17.63%
5. No - 82.37%
6. Analysing this variable w.r.t. Loan Status variable will give some vital insights.

Credit history length

1. Credit history length is Numerical variable.
2. There are no Missing values.
3. The variable ranges from 2 years to 30 years.
4. Longer the history, more knowledge about the Borrower which means more transparency and ultimately more reliable.
5. So based on Domain knowledge. Person's Credit History length variable is positively correlated with

16. Important Insights

1. It's a Binary Classification problem consist of Target Variable "Loan Status" -
 - 0 --> Not Default
 - 1 --> Default
2. There are records of **32,581** borrowers evaluated on 12 parameters (columns).
3. Target Variable is highly **Imbalanced**. The ratio of Target Variables is 21.82: 78.18 (0:1) (Majority class is twice of Minority class).
4. Mainly, there are 4 Categorical Variables (dtype = Object). The details of which as hereunder:
 - Binary – History of Default
 - Nominal – Home Ownership, Loan Intention
 - Ordinal – Loan Grade
5. There are only 2 Variables which have Missing Values:
 - Interest rate - **9.56%**
 - Employment length - **2.75%** There are total 4011 missing values in a given dataset.
6. There are **165** duplicate values.
7. Except Int. rate variable, all variables are Asymmetric and are Skewed.
8. There are neither Zero – Variance (constant) variables nor near Zero – Variance (Quasi-Constant) variables. (Threshold = 0.10).
9. Below are the Variable – wise Outliers computed on the basis of IQR Proximity Rules:

- Age = 1494
- Income = 1484
- Employment length = 853
- Loan amount = 1689
- Interest rate = 6
- Debt to Income ratio = 651
- Credit History length = 1142

10. As per Heatmap, the following pair of variables are highly correlated:

- Age & Credit History length - 0.86

17. Data Pre - Processing Checklist

1. Import Scikit learn library.
2. Handling Missing Values - Drop / Impute
3. Handling Outliers - Drop
4. Handling Imbalance of Target Variable - Resampling
5. Remove Duplicate values
6. Feature Scaling of High range Variables like Income variable
7. Categorical Encoding of Categorical features
8. Feature Selection based on multi-collinearity, zero or low variance.
9. Splitting the data into train - test.

In []:

In []: