Objective: Calculating credit worthiness for rural India (Bank Sector)

- Achieve the objective by **statistical analysis** and **model building using Python** (The code should be documented for reference).
- **Python coding** techniques used to achieve above objective should be shared, along with a small **document or presentation** which explains the approach**.**

Problem statement:

- **Credit score** of an individual or the organization plays a crucial information or role to know the customer financial status for financial institutions especially during loan applications approval.
- **Relevant to Indian market** the credit score of an individual or organization located in the rural area is not available. So, it is a challenging task for the bank or financial institutions to take decision whether to approve the loan applications submitted by the potential client/customer/organization located in the rural area without the credit history.
- The **objective is to give the insightful information to the business** from the datasheet provided (**trainingData**).
- Using **statistical**, **programming/coding** and **machine learning** techniques, the business expects to develop a model which will assist the financial institutions to decide or streamline the loan application approval by knowing the fair idea of the maximum repayment capability of customers which can be used to grant them the desired amount which in turn may reduce the loan recovery risk on the financial institutions.

Input supplied to achieve goal/objective:

- **Through** an email shared **Exercise/Task** & **training Data Excel.**
- **Exercise/Task** sheet in pdf format highlights the targeted objectives of the business.
- **trainingData** in CSV format which will provide the potential rural customer information.

Brief description about the input training data:

- The **supplied data** contains **40,000 rows/records** and **21 columns/features** which includes both predictors as well as target feature too.
- The **quantity/volume of supplied data** is highly appreciated which will be beneficial to achieve the targeted objective of the business.
- But the **quality of data supplied** needs treatment before it is used in the ML model by using the art of **advanced data feature engineering imputation techniques** which needs **skill set in statistics, programming, and machine learning algorithms**.
- The detailed description of the supplied features will be discussed in the subsequent section of the report whenever it is necessary, but now the bird overview of the trainingData supplied will be presented as below:

Description of variables (This information extracted from the input task/exercise pdf):

- **Primary Key**: Id
- **Personal_Details**: city, age, sex, social_class
- **Financial_Details**: primary_business, secondary_business, annual_income, monthly_expenses, old_dependents, young_dependents.
- **House Details**: home_ownership, type_of_house, occupants_count, house_area, sanitary_availability, water_availability

- **Loan Details**: loan_purpose, loan_tenure, loan_installments, loan_amount (*these contain loan details of loans that have been previously given, and which have been repaid*)

The above data makes up 40000 rows/records and 21 columns/features including both predictors and target as already stated above.

- So, from above information, now **it can be inferred which are predictors and target feature** as follow:

predictor_features = ['Id', 'city', 'age', 'sex', 'social_class', 'primary_business', 'secondary_business', 'annual_income', 'monthly_expenses', 'old_dependents', 'young_dependents', 'home_ownership', 'type_of_house', 'occupants_count', 'house_area', 'sanitary_availability', 'water_availabity', 'loan_purpose', 'loan_tenure', 'loan_installments']

target_feature = ['loan_amount']

As **target feature is continuous in nature so it is a REGRESSION category problem**.

- **Note:** Considering the problem statement and objective of the problem given the **Loan amount** feature is considered as the TARGET FEATURE. As target feature is continuous in nature it ultimately turns out to be a **regression domain specific problem or task**.

- In **real time** we need to sit with all the **stakeholders/client/business_expert_leaders** to take **input feature importance** and **Target variable information** well ahead before data pre-processing which will help to start the project with good commanding knowledge.

- **Data types**: There are total 21 features including target (Categorical type = 07, Float type= 07 & Int type = 07)

```
#Count of data type features
df.dtypes.value_counts()
```

```
float64    7
int64      7
object     7
dtype: int64
```

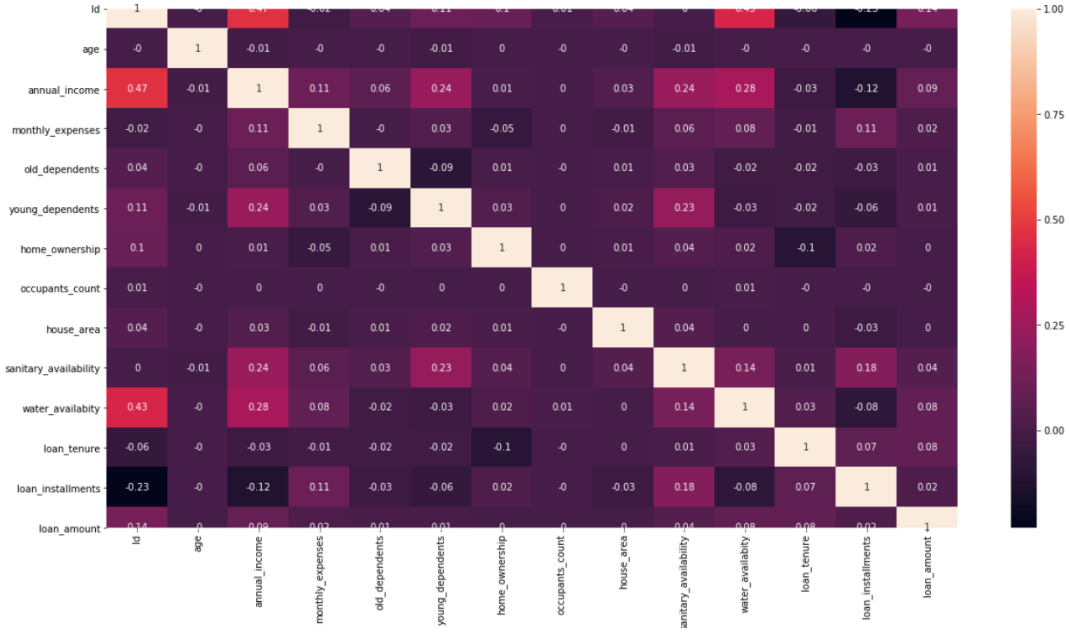
```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 21 columns):
Id                        40000 non-null int64
city                      38136 non-null object
age                       40000 non-null int64
sex                       40000 non-null object
social_class              34745 non-null object
primary_business          39974 non-null object
secondary_business        34759 non-null object
annual_income             40000 non-null float64
monthly_expenses          39880 non-null float64
old_dependents            40000 non-null int64
young_dependents          40000 non-null int64
home_ownership            39621 non-null float64
type_of_house             39306 non-null object
occupants_count           40000 non-null int64
house_area                40000 non-null float64
sanitary_availability     39792 non-null float64
water_availabity          34747 non-null float64
loan_purpose              39974 non-null object
loan_tenure               40000 non-null int64
loan_installments         40000 non-null int64
loan_amount               40000 non-null float64
dtypes: float64(7), int64(7), object(7)
memory usage: 6.4+ MB
```

- **Initial statistical** summary (i.e., Five-point summary: mean. Median, mode, min & max) indicates that the '**age'** feature has **an issue of outlier**.

```
# df.describe().T

df.describe()
```

	Id	age
count	40000.00000	40000.00000
mean	20000.50000	55.15990
std	11547.14972	3830.35566
min	1.00000	2.00000
25%	10000.75000	29.00000
50%	20000.50000	35.00000
75%	30000.25000	42.00000
max	40000.00000	766105.00000

- **Pearson correlation matrix** as shown below indicates there are no correlation between any predictor/independent features.



- **Observation of Correlation prospective study:** According to **pair plot** and **correlation matrix** values, it is observed that there is **no strong correlation between the predictor features.** However, there is a slight correlation between *loan tenure* and *loan instalments* which needs further investigation. **Initial statistic study** suggests that there is **no correlation between the predictor features** which signals that all the predictor features are important from the ML modelling point of view.

- **Missing values count**:

```
#Total no. of missing values relevant to each feature
# df.apply(lambda x: x.isnull().sum())
# df.isnull().sum().head()
df.isnull().sum()
```

```
Id                          0
city                     1864
age                         0
sex                         0
social_class             5255
primary_business           26
secondary_business       5241
annual_income               0
monthly_expenses          120
old_dependents              0
young_dependents            0
home_ownership            379
type_of_house             694
occupants_count             0
house_area                  0
sanitary_availability     208
water_availabity         5253
loan_purpose               26
loan_tenure                 0
loan_installments           0
loan_amount                 0
dtype: int64
```

Missing values count

```
### missing values:
round(df.isnull().sum()/df.shape[0]*100, 2)
```

```
Id                        0.00
city                      4.66
age                       0.00
sex                       0.00
social_class             13.14
primary_business          0.06
secondary_business       13.10
annual_income             0.00
monthly_expenses          0.30
old_dependents            0.00
young_dependents          0.00
home_ownership            0.95
type_of_house             1.74
occupants_count           0.00
house_area                0.00
sanitary_availability     0.52
water_availabity         13.13
loan_purpose              0.06
loan_tenure               0.00
loan_installments         0.00
loan_amount               0.00
dtype: float64
```

:- Missing values %

- **Observations from missing summary**:

- We **cannot drop any feature straight away** from the trainingData because here all feature indicates the **missing values < 70%** (This is an initial observation).

- It needs **feature engineering techniques to be applied to handle all the missing values in features of trainingData** by filling with appropriate mean/median/mode/OHE etc or any other appropriate methods.

- Also, non-missing feature needs feature engineering process due to outliers existence as per preliminary statistical analysis.

```
# Missing values handling
#Total no. of missing values
print("Total number of missing values in the data_set:",df.isnull().sum().values.sum())
print("Total %ge of missing values in the data_set:",round((df.isnull().sum().values.sum()/df.shape[0])*100,2))
```

```
Total number of missing values in the data_set: 19066
Total %ge of missing values in the data_set: 47.66
```

≃ 50%

It is observed that if we **neglect all the missing value records** in the supplied trainingData to quickly wind up data pre-processing activity then we **may end up almost 50% of records or data out of our analysis scope** which is not a good sign. That is why data pre-processing capability or skill is needed for data scientist to accomplish the most important stage of project activity. Usually data pre-processing takes quality time of data scientist. Of course, at the end quality data matters to achieve the accuracy.

Data imputation is needed to get the realistic solution of the problem statement stated above.

Observation: Categorical and floating type features experience missing values problem. There are no missing values in integer type features.

- **Skewness in the supplied trainingData**:

```
dummy_df  = skew(df[numerical_features])
_ = pd.concat([pd.DataFrame(numerical_features, columns = ['Features']),
          pd.DataFrame(dummy_df, columns = ['Skewness'])], axis = 1)
_ .sort_values('Skewness', ascending = False)

# for positive skewness: log, sqrt root, 1/3rd root...
# for negative skewness: square, cube, ....
```

	Features	Skewness
7	occupants_count	190.211847
1	age	190.211609
11	loan_tenure	5.745310
4	old_dependents	5.612282
12	loan_installments	0.955830
5	young_dependents	0.638660
13	loan_amount	0.611564
8	house_area	0.297576
2	annual_income	0.258040

From the above statistical analysis indicates that some features in the supplied data **experiences the skewness** which signals even the target feature **loan amount** also experiences the skewness.

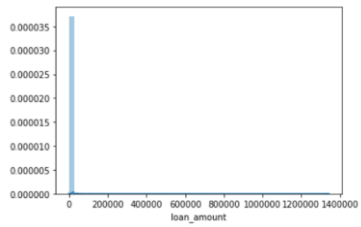
From the histogram of numerical features, it is evident that most of the features have outliers exists, which needs an imputation process.

Handling above issue is important because most of the ML algorithms are designed to work by assuming input data to the ML models are in NORMAL DISTRIBUTION.

- **Target feature (loan_amount) initial overview**:

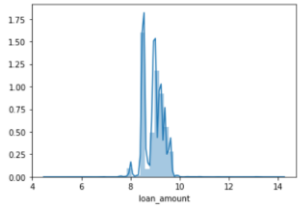
```
sns.distplot(df.loan_amount)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x261a16049c8>
```




```
# taking log transformation on loan_amount
sns.distplot(np.log1p(df.loan_amount)) # log1p -> adds 1 to entire data and then takes log
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x261a1726d88>
```



From the above plot it is evident that the target feature ‘loan_amount’ as stated above experiences skewness (as shown in left plot). To rectify skewness in the data the log plot shown right side of the plot.

It is necessary to handle all these outliers, missing values, scaling, skewness, conversion of categorical features into appropriate format etc to feed into ML model for the purpose of achieving targeted results.

This is the end of brief description of input trainingData. In the next section we will analyse each predictor in detail in alien to the problem statements to be accomplished as per task sheet which are as follows:

Section-1: Do a descriptive analysis of all the variables.

Section-2: There is a new customer who needs a loan. Which models will be best suited to predict the loan_amount that can be granted to the customer?

Section-3: Build a model to predict the maximum loan_amount that can be granted to the customer. Which all variables are good predictors?

Section-4: Build at least one model from scratch that fits this data, without using any third-party packages like sklearn, glm, lm, rpart, etc. You are free to use linear algebra packages like scipy, numpy or any blas derivative. We would be more interested in the convergence of the algorithm rather than the prediction accuracy.

Section-5: Is loan_purpose a significant predictor? The business has insisted on using loan_purpose as a predictor. If it is not already a significant contributor, can we still modify the model to include it?

Section-6: How will you measure the fitness of the model? Which metrics (accuracy, recall, etc.) are most relevant?

The above sections will be discussed in detail individually in the subsequent pages of the report which marks the end of this assignment report. Nevertheless, conclusion summary, innovative approach, best practices, methods followed documentation and any recommendations etc will be added in the report if necessary, to add value to the business for upcoming projects.

Section-1: Do a descriptive analysis of all the variables.

- **Do a descriptive analysis of all the variables:**

There are 40000 rows and 21 columns/variables including target feature.

➢ **Correlation check:**

According to pair plot and correlation matrix values, it is observed that there are **no strong correlation between the predictor features.** However, there is a slight correlation between loan tenure and loan instalments. **There is no correlation between the predictor features.**

➢ **Missing values count:**

Total number of missing values in the dataset: **19066 (Out of 40000)**
Total % of missing values in the dataset: **47.66% (Out of 100%)**

- IF WE NEGLECT ALL MISSING VALUE RECORDS, THEN WE MAY **END UP IN ALMOST 50% OF DATA OUT OF OUR ANALYSIS SCOPE.** SO, **DATA IMPUTATION** IS NEEDED TO GET REALISTIC SOLUTION FOR THE PROBLEM STATEMENT STATED.

- Categorical and floating type features experience missing values problem. There were no missing values in integer type features.

➢ Independent / predictor features EDA:

➢ **Target features (20 numbers):**

1. Id:

Id feature will be dropped due to **Nominal data type** in nature**. It does not add any feature importance to the model.**

2. city:

City feature is dropped which may not add much benefits from analytics perspective but depending on client/stakeholders concern it may be considered.

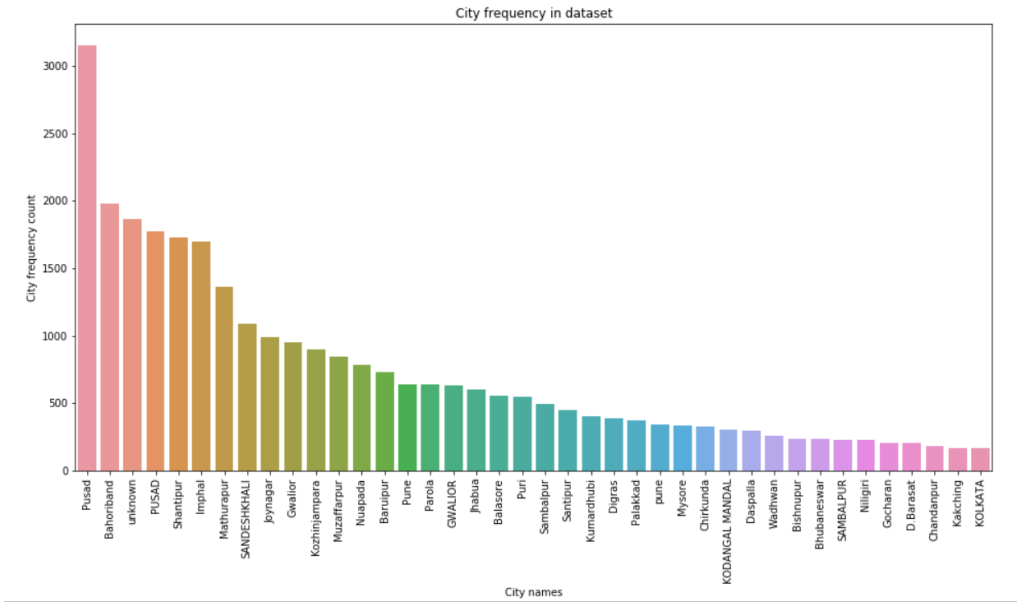
```
df.city.isnull().sum()
```

```
1864
```

```
print(df.city.nunique())
# print(df.social_class.unique())
```

```
856
```

City type feature has 1864 missing values (4.66% of total data) and 856 unique values (objective data type).



The top 40 city frequency plot of the trainingData set shown as above shows the overview of the prospective client or loan application base location regions.

There are 856 cities in the data set. The most frequent 40 cities are shown in the above visualization plot. Now we will drop this feature. Depending on stakeholders and business understanding if required we will consider this feature in ML model.

As stated above “city” feature is dropped from the analysis which may not add much benefits from analytics perspective but depending on client/stakeholders concern it may be considered. However, if business expects this “city” feature needs to be implemented in ML model, then following approaches are used.

```
#####################reducing city subcategories***************#####

# print(df.city.value_counts().head(40))
# print(df.city.nunique())
# print(df.city.unique())

df['city'].replace('Plakkad','Palakkad',inplace=True)

df['city'].replace('Niligiri','Nilgiri',inplace=True)
df['city'].replace('Nilgir','Nilgiri',inplace=True)
df['city'].replace('Niligir','Nilgiri',inplace=True)
df['city'].replace('nilgiri','Nilgiri',inplace=True)

df['city'].replace('KOLKAT','Kolkata',inplace=True)
df['city'].replace('KOLKATA','Kolkata',inplace=True)

df['city'].replace('COIMBATORE','Coimbatore',inplace=True)

#############################################
```

- **Lot of city names entered in minor variation in text** which needs to be **clubbed to make supplied raw data in meaningful data transition**.
- The **approach used as shown in the programming section**. Now few cities done but in future this city feature needs to be added in the model as per business requirement then we will process all data effectively.

```
print(df.city.nunique())
# print(df.city.unique())
```

```
849
```

By just matching/correcting few city spellings differences the city subcategories reduced to (856-849) = 7 number. There is good scope to do like this and we can make the city feature eligible to include in ML model.

Note: Still there is lot of scope to reduce the city subcategories. Either we can use manual or REGEX technique to find and match the similar cities in the dataset.

Most frequent or repeated present city in the records is Pusad

NOTE: Although there is an option to add as a replacement for missing city by using most frequent occurring city (Pusad) in the data set. This can be done by **asking client/stakeholders** later stages. **At this moment worth replacing missing city records as unknown to retain records**.

After above process, the final approach used to transfer city feature data in ML understanding language!!!

```
#####**********Frequency OneHOT ENCODE**********#####
```

```
####### Following work (i.e, city feature imputation) used if and only if business suggest we must use this feature #######
```

- If business wants to add **city feature into the model** then following approch is used, which is called **Frequency OneHOT ENCODE.**
- This method is preferred when the **column has too many unique values in it.**

```
# df.city.unique()
df.city.nunique()
```

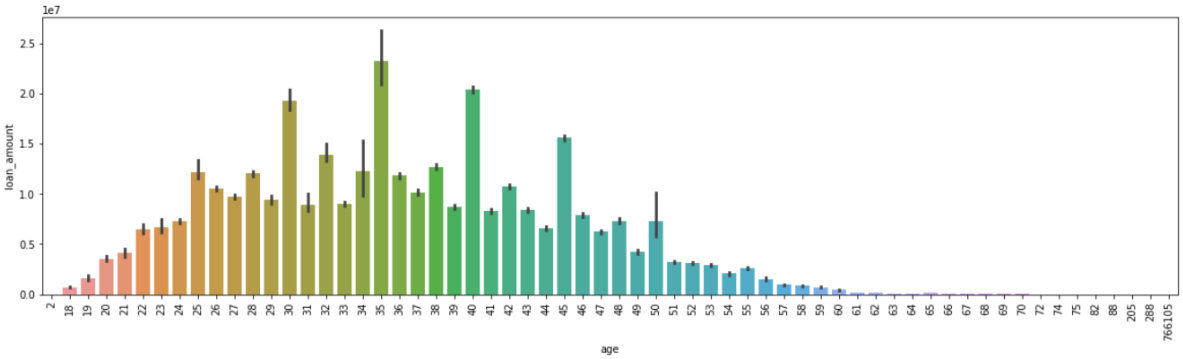
```
849
```

- There are total **857 city names** in **city feature** of the data set. If client/business wants to add city feature in the model, then it is absolutly necessary to do feature engineering. **Here some top frequent occuring cities are considered as a POC approach.**

Note: city feature is dropped which may not add much benefits from analytics perspective but depending on client/stakeholders concern it may be considered.

3. age:

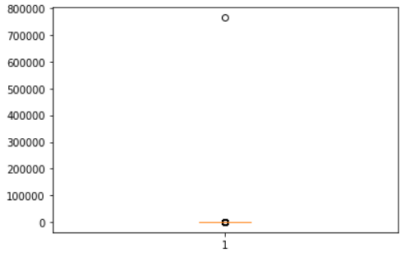
The distribution of **age** feature with respect to target **loan_amount** feature as shown below which clearly indicates the presence of outlies.



Preliminary observations from statistical descriptive summary:

- **age** predictor ranges from min. 1 to max. 766105. **The max age is unrealistic which needs attention.**

```
#personal_details
plt.boxplot(df['age'])
plt.show()
```



Observations: Outlier exists in age feature

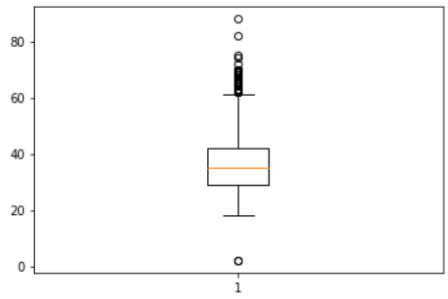
- Age feature needs resonable values modification

```
# df.groupby('age')[['loan_amount']].mean().sort_values(by='loan_amount', ascending=False).head(10)

df[['age', 'loan_amount']].agg(['min','mean','max'])
```

	age	loan_amount
min	2.0000	1.000000e+02
mean	55.1599	8.412593e+03
max	766105.0000	1.343000e+06

Initial raw data of age feature has high value entered which seems unrealistic in nature. Just by taking out very few visible outliers following box plot observed.

```
#personal_details
plt.boxplot(df['age'])
plt.show()
```

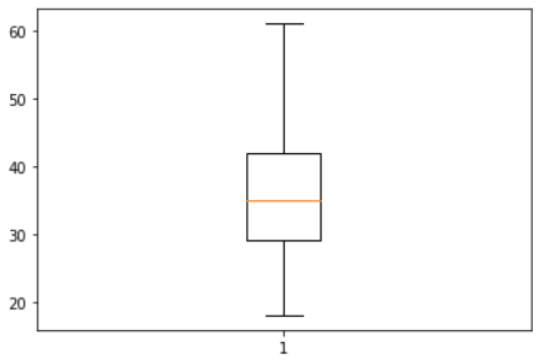



```
# Detecting outlier
print(df.age.min())
print(df.age.max())
print(df.age.median())
print(df.age.quantile(0.25))
print(df.age.quantile(0.75))
IQR = df.age.quantile(0.75) - df.age.quantile(0.25)
print(IQR)

UQR = df.age.quantile(0.75) + 1.5 * IQR
LQR = df.age.quantile(0.25) - 1.5 * IQR
print(LQR)
print(UQR)
```

With proper statistical approach, the **age** feature comes out with flying color without an oulier which is ready to enter into ML model.

```
df.drop(index=df[(df.age < 9.5) | (df.age > 61.5)].index, inplace=True)
```

```
#personal_details
plt.boxplot(df['age'])
plt.show()
```



It is evident from boxplot that now age feature is free from OUTLIERS

It is evident from boxplot that now age feature is free from OUTLIERS

```
print("Min. age:",df.age.min(), "&", "Max. age:",df.age.max())
```

```
Min. age: 18 & Max. age: 61
```

```
df.shape
```

```
(39906, 40)
```

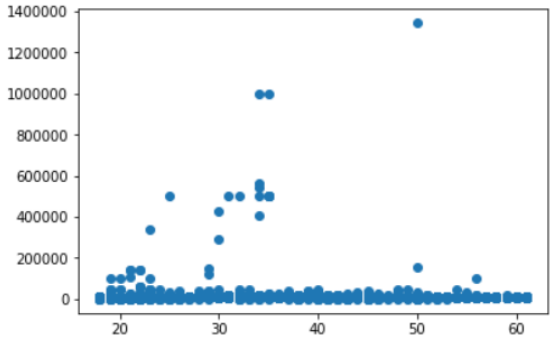
Note: Due to outlier issue in age feature 94 records are dropped

Observation of "age" feature after outlier:

- **After outlier treatment, age feature turns out in realistic nature** with min age 18 and max age 61.
- This is **achieved at the cost of 94 records drop** in which we may lose some information or insight.

```
plt.figure
plt.scatter(df.age, df.loan_amount)
```

```
<matplotlib.collections.PathCollection at 0x1f285059748>
```

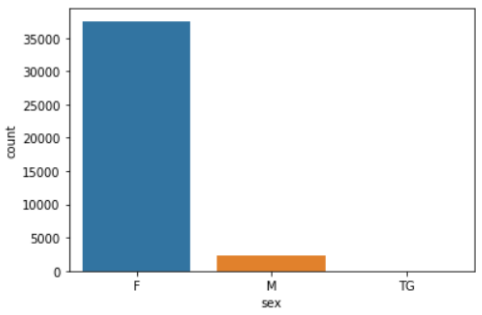


Observation: There is no correlation between age and loan_amount

There is no strong correlation between the predictor feature "age" and target feature "loan_amount".

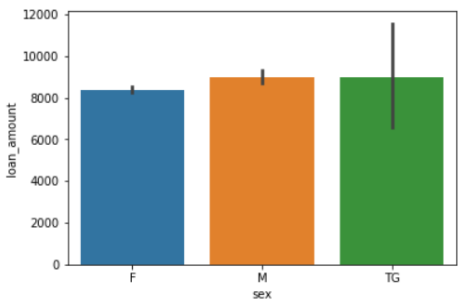
4. sex:

```
sns.countplot(df['sex'])
plt.show()
```




```
sns.barplot(df.sex, df.loan_amount)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f285a95348>
```



From the above plot it is evident that there is **imbalance distribution of sex feature**. However, in relation to target feature loan_amount all sex subcategory balances well (Of course transgender has high variance loan amount which is seen in the plot).

```
for each in df.sex.unique():
    print(each,"=", df[df.sex==each].shape[0]/df.shape[0])
```

```
F = 0.9409362000701649
M = 0.058888387711121135
TG = 0.00017541221871397784
```

```
df.sex.value_counts()
```

```
F     37622
M      2371
TG        7
Name: sex, dtype: int64
```

It is evident from the above statistical calculation ~94% data dominated by female (F), 5.8% by male (M) and rest portion by transgender (TG).

In later stages, depending on situations we will handle this imbalance issue using available different techniques like SMOTE etc.

```
df.drop(df[df.sex == 'TG'].index, inplace=True)
```

```
df.sex.value_counts()
```

```
F    37549
M     2350
Name: sex, dtype: int64
```

The seven records of transgender sub-category of sex dropped and remaining data further pre-processed for ML model intended prospective point.

OHE for sex feature:

```
# Converted to binary to help later on with models and plots
# df['sex'] = df['sex'].map({'M':1, 'F':0})
df_sex_new = pd.get_dummies(df.sex, prefix='sex', drop_first=True)
```

```
df_sex_new.head(3)
```

	sex_M
0	0
1	0
2	1

```
df = pd.concat([df, df_sex_new], axis=1)
```

```
df = df.drop(columns=['sex'], axis=1)
```

Now the **"age"** feature ready to enter ML algorithm.

5. social_class:

There are 517 subcategories of social_class and out of 40k records 5243 records are missing values in relevant to all social class subcategories as shown below.

```
print(df.social_class.nunique())
# print(df.social_class.unique())
```

```
517
```

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

```
5243
```

Pretty similar approach used to pre-process social_class feature as used in city feature.

The following approach is used to reduced social subcategories count:

```
#####******************reducing social_class subcategories***************#####

# #
# #### Convert all (no, No, NO) to 'no' and all (yes, yEs, Yes) to 'yes'
# df['target'] = df['target'].map(lambda x: x.lower())

df['social_class'].replace('o.b.c','obc',inplace=True)
df['social_class'].replace('o b c','obc',inplace=True)
df['social_class'].replace('o.b.c.','obc',inplace=True)
df['social_class'].replace('hindu_sc','obc',inplace=True)


df['social_class'].replace('s.c','sc',inplace=True)
df['social_class'].replace('s.c.','sc',inplace=True)
df['social_class'].replace('schudle cast','sc',inplace=True)
df['social_class'].replace('schudle cast','sc',inplace=True)
df['social_class'].replace('scheduled caste','sc',inplace=True)
df['social_class'].replace('sechudle caste','sc',inplace=True)
df['social_class'].replace('s c','sc',inplace=True)
df['social_class'].replace('schduled cast','sc',inplace=True)
```

```
print(df.social_class.nunique())
# print(df.social_class.unique())
```

```
429
```

Note: Still there is scope to reduce the the social_class subcategories

Most frequent or repeted present social_class in the records is OBC

```
print("Total no. of social_class feature: ",df.social_class.nunique())
```

```
Total no. of social_class feature:  429
```

Note: social_class is dropped which may not add much benefits from analytics prospective but depending on client/stakeholders concern it may be considered.

NOTE: Althought there is an option to add as a replacement for missing social_class by using most frequent occuring social_class (OBC) in the data set. This can be done by asking client/stakeholders later stages. At this moment worth replacing missing ocial_class recorts as unknown to retain records.

After reducing the possible social_subcategories, another approach called **FREQUENCY OHE** used to generate dummy features with the intention of converting categorical feature in numerical to enter ML model.

```
#####***********Frequency OneHOT ENCODE social_class feature**********************#####

### Following work (i.e, social_class feature imputation) used if and only if business suggest we must use this feature #####
```

- If business wants to add social_class feature into the model then following approch is used, which is called **Frequency OneHOT ENCODE.**
- This method is preferred when the **column has too many unique values in it.**

```
# df.social_class.unique()
df.social_class.nunique()
```

```
429
```

- There are total **360 social_class names** (Initially there are 517 social_class which reduced to 360 after feature engineering....still there is a scope to reduce.....if required absolutly this feature then we will do it in perfect way to go into ML model) in **social_class feature** of the data set. If client/business wants to add **social_class feature in the model**, then it is absolutly necessary to do feature engineering. **Here some top frequent occuring social_class are considered as a POC approach.**

```
# Here some top frequent occuring social_class are considered but depending on necessity we can add or reduce
# df.social_class.value_counts().head(20)
# df.social_class.value_counts().head(20)[0:20]
# df.social_class.value_counts().head(20).index
# df.social_class.value_counts().head(20).index.to_list()
print(df.social_class.value_counts().head(20).index.to_list())
```

```
['obc', 'sc', 'general', 'unknown', 'st', 'muslim', 'minority', 'hindu', 'nt', 'bc', 'open', 'vjnt', 'lingayat', 'mahar', 'christian', 'ezhava', 'm
aratha', 'okkaliga', 'minority comunity', 'kuruba']
```

```
temp = ['obc', 'sc', 'general', 'st', 'muslim', 'minority', 'hindu', 'nt', 'bc', 'open', 'vjnt', 'lingayat', 'mahar',
        'christian', 'ezhava', 'maratha', 'okkaliga', 'minority comunity', 'kuruba', 'nayaka']
```

```
df.social_class = df.social_class.apply(lambda x : x if x in temp else 'unknown_social_class')
```

```
df.social_class.value_counts().head(6)
```

```
obc                    11408
unknown_social_class    7038
sc                      6543
general                 5276
st                      2996
muslim                  2263
Name: social_class, dtype: int64
```

```
df_social_class_new = pd.get_dummies(df.social_class, prefix='social_class', drop_first=True)
```

```
df = pd.concat([df, df_social_class_new], axis=1)
```

```
df = df.drop(columns = ['social_class'], axis=1)
```

```
#####**********************************************#####
```

At this stage we have left with rows/records =39899 (out of 40k) **and (predictors+target) = 59. Still journey needs to be continued for rest of the feature imputation.**

Social_class feature imputation completed!!!!.

6. primary_business:

Primary business has 440 subcategories and 26 missing records.

```
print(df.primary_business.nunique())
# print(df.primary_business.unique())
```

```
440
```

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

```
26
```

Following figure indicates top 20 primary_business categories.

```
# Cross checking highest frequent occured city
# df.groupby(['primary_business']).size().sort_values(ascending=False).tail(20)
df.groupby(['primary_business']).size().sort_values(ascending=False).head(20)
```

```
primary_business
Tailoring               3969
Goat rearing            2265
Cow Rearing             2070
Handloom Work           2068
Vegetable cultivation   1699
Grocery store           1367
School                  1332
Milk business           1305
Vegetable vendor        1168
Weaver                  1157
Saree business          1000
Embroidery work          941
Fish rearing             734
Poultry farm             723
Rice business            676
General store            660
Cloth business           608
Paddy cultivation        590
Education Loan           556
Buffalo rearing          500
dtype: int64
```

The missing value records are dropped, and further data pre-processing carried out as shown below:

```
df.shape
```

```
(39899, 59)
```

```
df.dropna(subset=['primary_business'],inplace=True)
```

```
df.shape
```

```
(39873, 59)
```

```
df.primary_business.nunique()
```

```
440
```

```
#####################OHE for primary_business feature*******************************#####
```

```
df_primry_busines = pd.get_dummies(df.primary_business, prefix='primary_business', drop_first=True)
```

```
df = pd.concat([df, df_primry_busines], axis=1)
```

```
df = df.drop(columns=['primary_business'], axis=1)
```

```
#####################################################*#####
```

```
df.shape
```

```
(39873, 497)
```

Really increasing **predictor feature number!!!.....already rushing towards increasing trend** 497!!!. Now primary_business predictor ready to move in ML model process activity.

7. secondary_business:

Pretty similar approach used to pre-process secondary_business feature as used in primary_business feature.

There are 5 subcategories in secondary_business feature. **It is worth to note it includes "none", "nan" & "others" on top of 5229 missing values in this feature**. Altogether may account large portion of data set relevant to secondary_business feature.

```
print(df.secondary_business.nunique())
print(df.secondary_business.unique())
```

```
5
['Others' 'none' 'Agriculture' 'Daily wage labourer' 'Livestock rearing'
 nan]
```

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

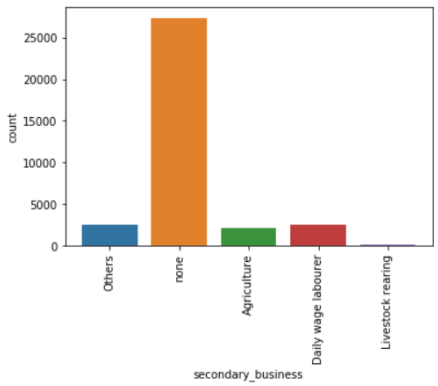
```
5229
```

Depending on business understanding or consulting it will be decided to consider a secondary_business feature in the model.

Following plot informs the distribution pattern of secondary_business feature subcategories.

```
# plt.figure(figsize=(10,8))
sns.countplot('secondary_business',data = df)
print(len(df.secondary_business.unique()))
plt.xticks(rotation=90)
plt.plot()
```

```
6
[]
```



Note: Others and none parametrs of secondary_business feature leads to 86.1%. which may not give much insight.

```
print(df.secondary_business.nunique())
print(df.secondary_business.unique())
```

```
5
['Others' 'none' 'Agriculture' 'Daily wage labourer' 'Livestock rearing'
 nan]
```

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

```
5229
```

```
df.secondary_business.value_counts().values.sum()
```

```
34644
```

```
df.secondary_business.value_counts()
```

```
none                   27268
Others                  2559
Daily wage labourer     2540
Agriculture             2100
Livestock rearing        177
Name: secondary_business, dtype: int64
```

```
df['secondary_business'].fillna('none', inplace=True)
```

Missing values are replaced with "**none**" which is already present subcategory of secondary_business feature.

```
######################################################################

# df.secondary_business ==   ['Others', 'none', 'nan']
total_miss_sec_busins = len(df[df.secondary_business == 'Others']) + len(df[df.secondary_business == 'none']) + len(df[df.secondary_business == 'nan

perntage_total_miss_sec_busins = round((total_miss_sec_busins/df.shape[0])*100, 2)
perntage_total_miss_sec_busins
```

```
87.92
```

Note: Seconay business features have lot of missing data along with 'none' and 'other' entries which altogether makes up approx. 87.93%. Ask the client to send the updated records for this secondary_business feature as most of the data are not making value addition. **It is worth to drop this column but before dropping this feature it is worth to discuss with stakeholder/customer approval how important this feature if dropped in the modeling.**

```
######################################################################
```

Note: As per above secondary_business statics data analysis it hints "other", "non", "nan" leads to ~88 % which may not give much insight from business perspective.

The missing value records are dropped, and further data pre-processing carried out as shown below:

```
# df['secondary_business'].value_counts().sort_values(ascending=False).head(20)
df['secondary_business'].value_counts()
```

```
none                   32497
Others                  2559
Daily wage labourer     2540
Agriculture             2100
Livestock rearing        177
Name: secondary_business, dtype: int64
```

```
df.shape
```

```
(39873, 497)
```

```
df.secondary_business.nunique()
```

```
5
```

```
#####*****************OHE for secondary_business feature*********************************#####
```

```
df_secondry_busines = pd.get_dummies(df.secondary_business, prefix='secondary_business', drop_first=True)
```

```
df = pd.concat([df, df_secondry_busines], axis=1)
```

```
df = df.drop(columns=['secondary_business'], axis=1)
```

```
#####*****************************************************#####
```

```
df.shape
```

```
(39873, 500)
```

Finally, after secondary_business data pre-processing, we have data frame size = 39873 (rows) X 500 (columns+TARGET).

Now, secondary_business feature imputation completed!!!

8. annual_income:

There are **NO MISSING VALUES** in annual_income predictor!!!!!

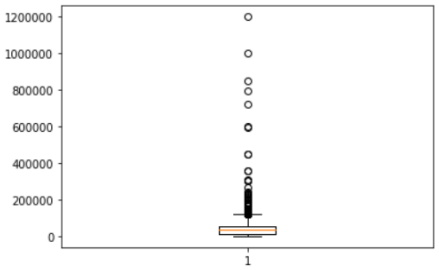
```
# df.annual_income
# df.annual_income.min() # 0.0
# df.annual_income.max() # 1200000.0
# df.annual_income.mean() # 43328.572001963556
df.annual_income.isnull().sum() # 0
```

```
0
```

Regression models need FEATURE SCALING due to high spread in data range

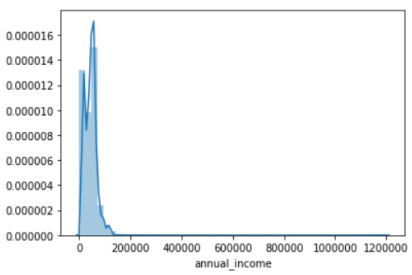
From the above stat analysis data, it indicates a large spread in the annual_income feature which is evident from below box and distribution plot.

```
plt.boxplot(df.annual_income)
plt.show()
```




```
sns.distplot(df.annual_income)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x261a1be3e88>
```



Outliers in the annual_income handled as shown below:

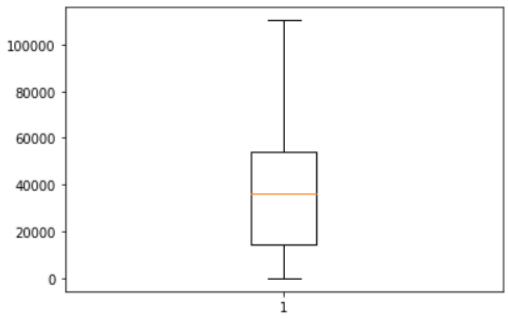
```
# Detecting outlier in annual_income
print(df.annual_income.min())
print(df.annual_income.max())
print(df.annual_income.median())
print(df.annual_income.quantile(0.25))
print(df.annual_income.quantile(0.75))
IQR = df.annual_income.quantile(0.75) - df.annual_income.quantile(0.25)
print(IQR)

UQR = df.annual_income.quantile(0.75) + 1.5 * IQR
LQR = df.annual_income.quantile(0.25) - 1.5 * IQR
print(LQR)
print(UQR)
```

```
0.0
1200000.0
36000.0
14400.0
56000.0
41600.0
-48000.0
118400.0
```

```
# df[(df.annual_income <= -48000.0) | (df.annual_income >= 118400.0)].index
# df.drop(index=df[(df.annual_income <= -48000.0) | (df.annual_income >= 118400.0)].index, inplace=True)
df.drop(index=df[(df.annual_income <= -48000.0) | (df.annual_income >= 114000.0)].index, inplace=True)
```

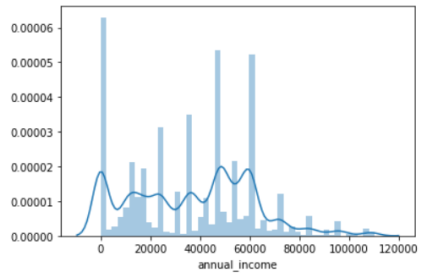
Finally, the outcome of annual_income feature data imputation results are as follows.

```
plt.boxplot(df.annual_income)
plt.show()
```




```
sns.distplot(df.annual_income)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x261a2922208>
```



Finally, after annual_income data pre-processing, we have data frame size = 39481 (rows) X 500 (columns+TARGET).

Now, annual_income feature imputation completed!!!

9. monthly_expenses:

There are **120 records/rows of monthly_expenses have MISSING VALUES** which needs feature engineering.

```
df.monthly_expenses.isnull().sum() #120 records missing
```

```
120
```

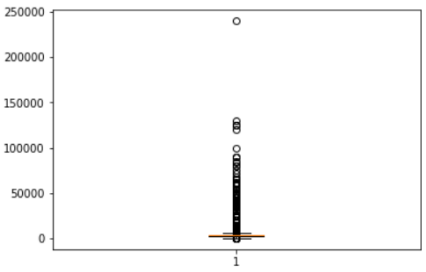
Missing values are treated with the mean values of the monthly_expenses feature as shown below:

```
# df.monthly_expenses.fillna(np.mean(df['monthly_expenses']), inplace=True)
df['monthly_expenses'].fillna(df['monthly_expenses'].mean(),inplace=True)
```

From the below stat analysis data, it indicates a large spread in the monthly_expenses feature which is evident from below box and distribution plot.

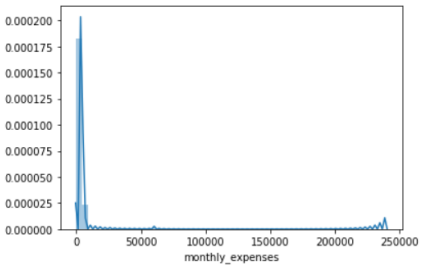
```
# df.monthly_expenses
# df.monthly_expenses.min() # 2.0
# df.monthly_expenses.max() # 240000.0
# df.monthly_expenses.mean() # 3866.72
# df.monthly_expenses.isnull().sum() # 0
```

```
plt.boxplot(df.monthly_expenses)
plt.show()
```




```
sns.distplot(df.monthly_expenses)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f286308708>
```



Outliers in the monthly_expenses handled as shown below:

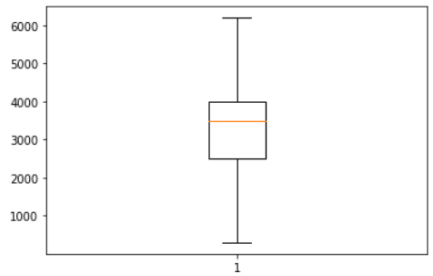
```
# Detecting outlier in monthly_expenses
print(df.monthly_expenses.min())
print(df.monthly_expenses.max())
print(df.monthly_expenses.median())
print(df.monthly_expenses.quantile(0.25))
print(df.monthly_expenses.quantile(0.75))
IQR = df.monthly_expenses.quantile(0.75) - df.monthly_expenses.quantile(0.25)
print(IQR)

UQR = df.monthly_expenses.quantile(0.75) + 1.5 * IQR
LQR = df.monthly_expenses.quantile(0.25) - 1.5 * IQR
print(LQR)
print(UQR)
```

```
2.0
240000.0
3500.0
2500.0
4000.0
1500.0
250.0
6250.0
```

```
# df[(df.monthly_expenses <= 250.0) | (df.monthly_expenses >= 6250.0)].index
df.drop(index=df[(df.monthly_expenses <= 250.0) | (df.monthly_expenses >= 6250.0)].index, inplace=True)
```

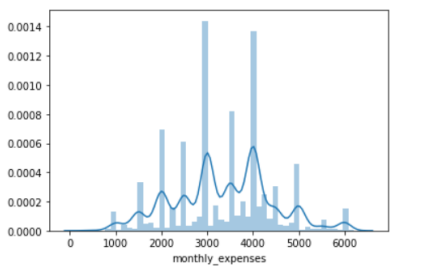
Finally, the outcome of monthly_expenses data imputation results are as follows.

```
plt.boxplot(df.monthly_expenses)
plt.show()
```




```
sns.distplot(df.monthly_expenses)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f285baea88>
```



Finally, after monthly_expenses data pre-processing, we have data frame size = 38284 (rows) X 500 (columns+TARGET).

Now, monthly_expenses feature imputation completed!!!

10. old_dependents:

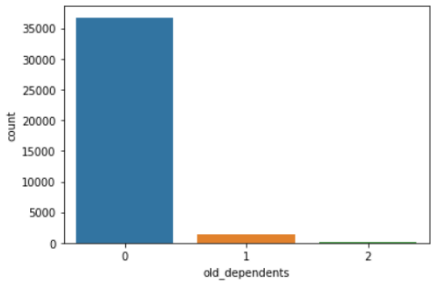
There are **NO MISSING VALUES** in old_dependent predictor!!!!!

```
# df.old_dependents
# df.old_dependents.min() # 0
# df.old_dependents.max() # 3
# df.old_dependents.mean() # 0.05
# df.old_dependents.isnull().sum() # 0
# df.old_dependents.nunique() # 4
```

From the below plot and statistics, it is evident that lot of imbalance exists in the sub-category of old_dependents which needs discussion with the business understanding.

```
sns.countplot(df.old_dependents)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e9eaf3d188>
```




```
print(df.old_dependents.nunique())
print(df.old_dependents.sort_values().unique())
```

```
3
[0 1 2]
```

```
df.old_dependents.value_counts()
```

```
0    36749
1     1423
2      112
Name: old_dependents, dtype: int64
```

```
# df[df.old_dependents >= 3]
df.drop(index=df[df.old_dependents >= 3].index, inplace=True)
```

```
df.old_dependents.value_counts()
```

```
0    36749
1     1423
2      112
Name: old_dependents, dtype: int64
```

In case, business insists to include this feature in the model it is possible either by OHE coding or by giving higher weightage factor to the minor old_dependent feature of subcategories.

As of now, the data count and its sub_category perfectly accomplished weightage factor on its own! So, this feature ready to go for ML model implementation.

11. young_dependents:

There are **NO MISSING VALUES** in young_dependent predictor!!!!

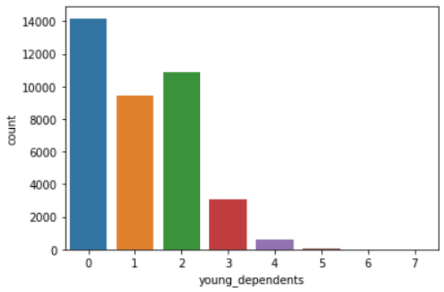
```
# df.young_dependents
# df.young_dependents.min() # 0
# df.young_dependents.max() # 7
# df.young_dependents.mean() # 1.31
# df.young_dependents.isnull().sum() # 0
# df.young_dependents.nunique() # 8
```

From the below plot and statistics, it is evident that lot of imbalance exists in the sub-category of young_dependents which needs discussion with the business understanding.

However, extreme outliers shown gate pass to make sure those will not participate in ML model building activity.

```
sns.countplot(df.young_dependents)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f2fe091388>
```




```
print(df.young_dependents.nunique())
print(df.young_dependents.sort_values().unique())
```

```
8
[0 1 2 3 4 5 6 7]
```

```
df.young_dependents.value_counts()
```

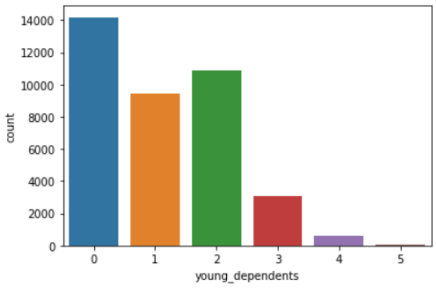
```
0    14184
2    10865
1     9474
3     3063
4      595
5       81
6       18
7        4
Name: young_dependents, dtype: int64
```

```
df[df.young_dependents > 5]
df.drop(index=df[df.young_dependents > 5].index, inplace=True)
```

In case, business insists to include this feature in the model it is possible either by OHE coding or by giving higher weightage factor to the minor old_dependent feature of subcategories.

```
sns.countplot(df.young_dependents)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f28220b988>
```




```
df.young_dependents.value_counts()
```

```
0    14184
2    10865
1     9474
3     3063
4      595
5       81
Name: young_dependents, dtype: int64
```

As of now, the data count and its sub_category are **NOT IN** perfectly accomplished weightage factor on its own!!!, which we seen in case of old_dependends feature. So, this feature ready to go for ML model implementation with **slight compromise of weightage factor**!

12. home_ownership:

Along with 355 missing records in the home_ownership feature, there exists a "nan" subcategory.

```
print(df.home_ownership.nunique())
print(df.home_ownership.unique())
```

```
2
[ 1.  0. nan]
```

```
df.home_ownership.value_counts()
```

```
1.0    37008
0.0      899
Name: home_ownership, dtype: int64
```

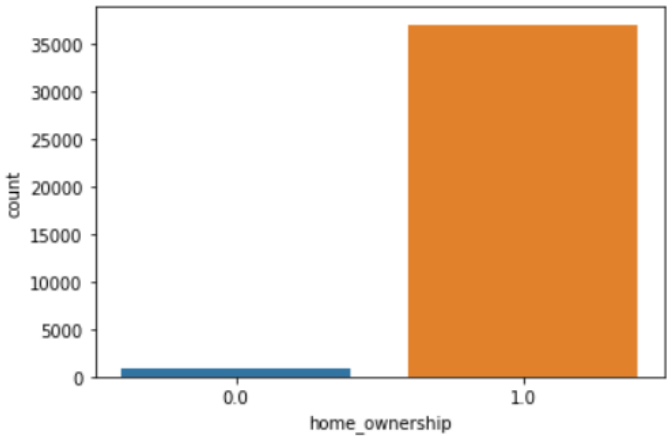
```
df.home_ownership.isnull().sum()
```

```
355
```

From the following plot it is evident that imbalance exists in the subcategory of home_ownership feature.

```
sns.countplot(df.home_ownership)
plt.plot()
```

```
[]
```



The "nan" value handled by replacing it with major subcategory value 1 as shown below.

```
# df['home_ownership'].unique()
# # df['home_ownership'] = df['home_ownership'].replace(np.nan, 'unknown')
# df['home_ownership'] = df['home_ownership'].replace(np.nan, 1.0)
```

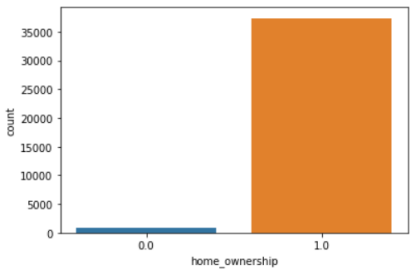
```
df['home_ownership'].fillna(1,inplace=True)
```

```
df.home_ownership.value_counts()
```

```
1.0    37363
0.0      899
Name: home_ownership, dtype: int64
```

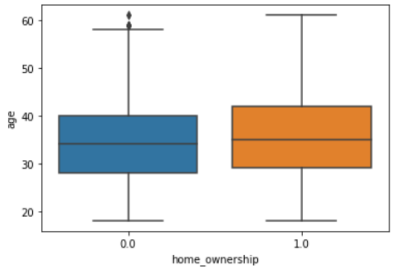
```
sns.countplot(df.home_ownership)
plt.plot()
```

```
[]
```




```
sns.boxplot(x = 'home_ownership', y='age', data = df)
# sns.boxplot(x = 'home_ownership', y='loan_amount', data = df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f28258b688>
```



Still there exists an imbalance in the subcategory of home_ownership which needs to be addressed if proper data input expected for ML model generation to get determined objective of the problem.

The insight here home_ownership observed in the age group 30-40 range which is quite true from the current business trend!!!

As of now, the data count and its sub_category are **NOT IN** perfectly accomplished weightage factor on its own!!!, which we have seen in case of old_dependends feature. So, this feature **IS NOT** ready to go for ML model implementation until either **OHE** or correct **weightage factor assignment**!

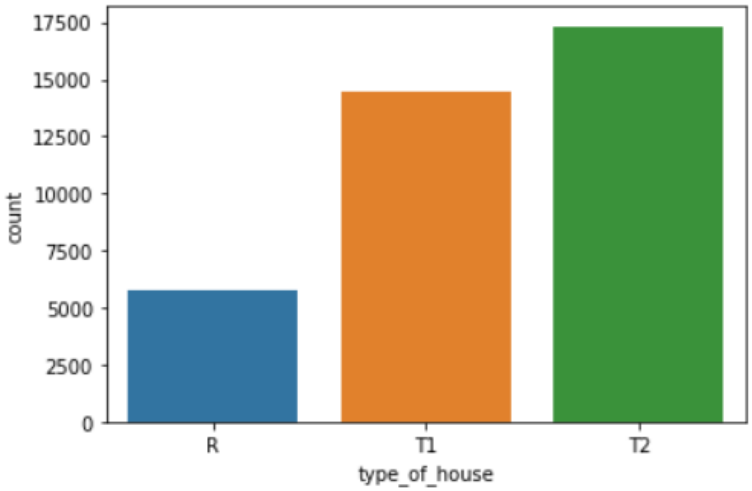
Finally, after home_ownership data pre-processing, we have **data frame size = 38262 (rows) X 500 (columns+TARGET)**.

Now, home_ownership feature imputation completed!!!

13. type_of_house:

The counts of type_of_house sub-categories shown in the following plot which indicates R type_of_house fall in minor subcategory of type_of_house feature.

```
sns.countplot(df['type_of_house'])
plt.show()
```



From the statistical analysis there exists 680 missing values. These missing values and 'nan' records dropped from the analysis.

```
df.type_of_house.mode()
```

```
0    T2
dtype: object
```

```
print(df.type_of_house.nunique())
print(df.type_of_house.unique())
```

```
3
['R' 'T1' 'T2' nan]
```

```
df.type_of_house.value_counts()
```

```
T2    17320
T1    14472
R      5790
Name: type_of_house, dtype: int64
```

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

```
680
```

```
df.shape
```

```
(38262, 500)
```

```
df.dropna(subset=['type_of_house'], inplace=True)
```

```
df.shape
```

```
(37582, 500)
```

```
df.type_of_house.value_counts()
```

```
T2    17320
T1    14472
R      5790
```

```
df.type_of_house.value_counts()
```

```
T2    17320
T1    14472
R      5790
Name: type_of_house, dtype: int64
```

```
#####**********OHE for type_of_house feature**********#####
```

```
# pd.get_dummies(df.type_of_house, prefix='type_of_house', drop_first=True).head()
df_type_of_house_new = pd.get_dummies(df.type_of_house, prefix='type_of_house', drop_first=True)
```

```
# df_type_of_house_new
df_type_of_house_new.head()
```

	type_of_house_T1	type_of_house_T2
0	0	0
1	1	0
2	1	0
3	1	0
4	1	0

```
df = pd.concat([df, df_type_of_house_new], axis=1)
```

```
df = df.drop(columns=['type_of_house'], axis=1)
```

```
#####**********#####
```

Then the remaining records of data are pre-processed using OHE coding technique to generate dummy variable for the type_of_house subcategories. It is evident from calculation the minor parameter R-type feature omitted (NOT FROM ANALYSIS!!!) and kept the two major subcategories in logical manner. Once dummy variable created, then later time the main type_of_house feature will be made obsolete.

Finally, after type_of_house data pre-processing, we have **data frame size = 37582 (rows) X 501 (columns+TARGET)**.

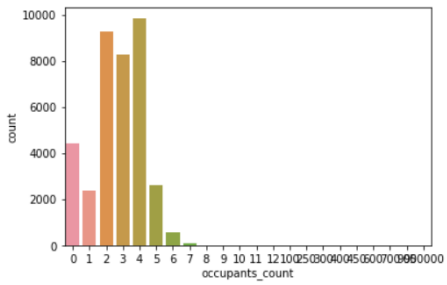
Now, type_of_house feature imputation completed!!!

14. occupants_count:

From the following plot it is evident that imbalance exists in the subcategory of occupants_count feature.

```
sns.countplot(df.occupants_count)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f285640548>
```



There is unrealistic occupant count which can be seen from the following statistical analysis.

```
# df.occupants_count
# df.occupants_count.min() # 0
# df.occupants_count.max() # 950000
# df.occupants_count.mean() # 30.761312119199562
# df.occupants_count.isnull().sum() # 0
# df.occupants_count.nunique() # 23
df.occupants_count.sort_values().unique() # 23
```

```
array([     0,      1,      2,      3,      4,      5,      6,      7,
            8,      9,     10,     11,     12,    100,    250,    300,
          400,    450,    600,    700,    900, 950000], dtype=int64)
```

There are unrealist occupant_count which needs imputation

Usually now a days in Indian family members size by common sense in most general cases will not go beyond 8 numbers. This may require discussion with the business but not so crucial now.

Keeping the said constraint in mind all the records which have **family member size greater than 8 dropped from the scope**.

```
df.shape
```

```
(37582, 501)
```

```
# df[df.occupants_count >= 15].index
# df[df.occupants_count > 8].index
df.drop(index=df[df.occupants_count >= 8].index, inplace=True)
```

```
df.shape
```

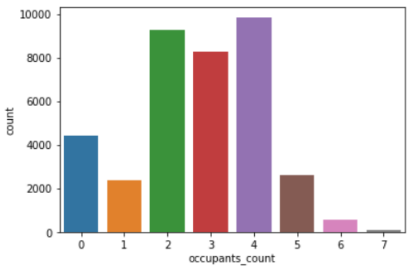
```
(37489, 501)
```

Ultimately **this results in loss of 93 records**. **This loss or sacrifice of records leads to avoid overfitting model issues as well as increased predictive power of the model. In this way we can make our model more general predictive power to face at least unseen handling capability**.

After imputation, following plot shows the pattern count of occupants in the family.

```
sns.countplot(df.occupants_count)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f28283bec8>
```




```
df.occupants_count.value_counts()
```

```
4    9824
2    9252
3    8251
0    4413
5    2649
1    2404
6     578
7     118
Name: occupants_count, dtype: int64
```

Still there exists an imbalance in the subcategory of occupant_count which needs to be addressed if proper data input expected for ML model generation to get determined objective of the problem.

The insight here occupant_count numbers of Indian family observed mostly in the count 2-4 range which is quite true from the current Indian family size considered!!!

Note: Even **there exists a data with 0 (ZERO) count family member** which is unrealistic in nature. It should be **discussed with the business if our understanding or interpretation may be not in the right direction** and then corrective action should be implemented once consulting with either domain expert or business. But not an issue to check with this data to feel its impact. If business expects this occupant_count as part of the model then the final model should need properly imputed occupant_count category.

As of now, the data count and its sub_category are **NOT IN** perfectly accomplished weightage factor on its own!!!, which we seen in case of old_dependends feature. So, this feature **IS NOT** ready to go for ML model implementation until either **OHE** or correct **weightage factor assignment**!

Finally, after occupant_count data pre-processing, we have **data frame size = 37489 (rows) X 501 (columns+TARGET)**.

Now, occupant_count feature imputation completed!!!

15.house_area:

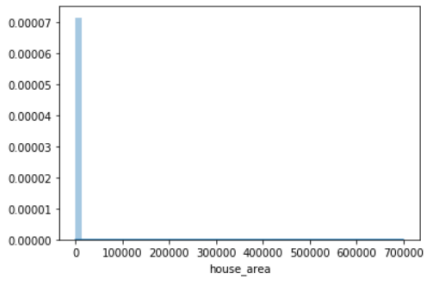
From the statistical point of view, it is evident there exists a widespread or higher dispersion in data of house_area feature as indicated by the following calculations.

```
# df.house_area
# df.house_area.min() # 0.0
# df.house_area.max() # 700000.0
# df.house_area.mean() # 621.3039596498347
# df.house_area.isnull().sum() # 0
```

Distribution plot below hints there are outliers in the house_area feature, which needs an attention to rectify. Also, the plot with target feature loan_amount, there is no clear-cut sign of relation. It needs relook the matter once outlier handled.

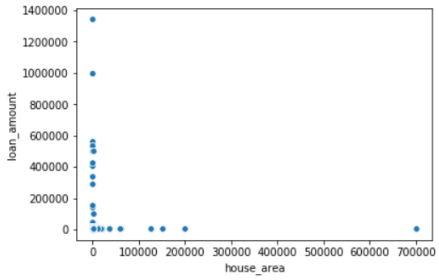
```
sns.distplot(df.house_area)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f288349dc8>
```




```
sns.scatterplot(df.house_area, df.loan_amount)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f2881980c8>
```



Outliers handled as shown in the below way!!!

```
# Detecting outlier in house_area
print(df.house_area.min())
print(df.house_area.max())
print(df.house_area.median())
print(df.house_area.quantile(0.25))
print(df.house_area.quantile(0.75))
IQR = df.house_area.quantile(0.75) - df.house_area.quantile(0.25)
print(IQR)

UQR = df.house_area.quantile(0.75) + 1.5 * IQR
LQR = df.house_area.quantile(0.25) - 1.5 * IQR
print(LQR)
print(UQR)
```

```
0.0
700000.0
400.0
200.0
620.0
420.0
-430.0
1250.0
```

```
df[(df.house_area <= -300) | (df.house_area >= 1300)].index
```

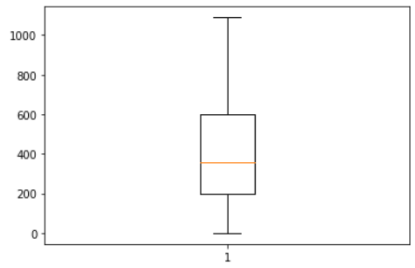
```
Int64Index([  137,   145,   147,   150,   169,   170,   196,   348,   647,   653,
            ...
            39759, 39760, 39761, 39762, 39788, 39844, 39850, 39856, 39892, 39918], dtype='int64', length=2804)
```

```
# df.drop(index=df[(df.house_area <= -300) | (df.house_area >= 1300)].index, inplace=True)
df.drop(index=df[(df.house_area <= -300) | (df.house_area >= 1100)].index, inplace=True)
```

Finally, the outcome of house_area data after imputation results are as follows.

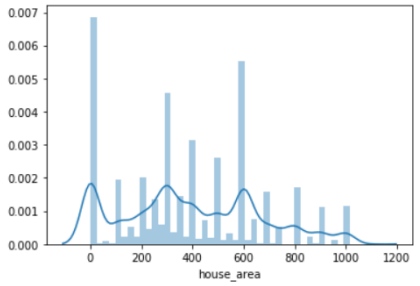
```
plt.boxplot(df.house_area)
plt.plot()
```

```
[]
```




```
sns.distplot(df.house_area)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f282874ac8>
```



Finally, after house_area data pre-processing, we have data frame size = 34268 (rows) X 501 (columns+TARGET).

Now, house_area feature imputation completed!!! which is ready to go!!!

16.sanitary_availability:

Observation: Along with 52 missing records in the sanitory_availability feature, there exists a "nan" subcategory.

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

```
52
```

```
print(df.sanitary_availability.nunique())
print(df.sanitary_availability.unique())
```

```
2
[ 1.  0. nan]
```

```
df.sanitary_availability.value_counts()
```

```
1.0    23103
0.0    11113
Name: sanitary_availability, dtype: int64
```

```
df.shape
```

```
(34268, 501)
```

```
# df[df.sanitary_availability == -1.]
```

```
# df.drop(index=[28817], inplace=True)
```

```
df.shape
```

```
(34268, 501)
```

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

```
52
```

```
df.dropna(subset=['sanitary_availability'], inplace=True)
```

```
df.shape
```

```
(34216, 501)
```

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

```
0
```

```
df.sanitary_availability.value_counts()
```

```
1.0    23103
0.0    11113
Name: sanitary_availability, dtype: int64
```

```
# df.sanitary_availability.mean()
```

```
# from sklearn.impute import SimpleImputer
# imputer = SimpleImputer(missing_values='NaN', strategy='most_frequent')
# imputer = SimpleImputer(missing_values='NaN', strategy='constant', fill_value='missing')
# imputer = SimpleImputer(missing_values=np.nan,strategy="mean")
# Can only use these strategies: ['mean', 'median', 'most_frequent', 'constant']

# imputer.fit_transform(df[:,['sanitary_availability']])
```

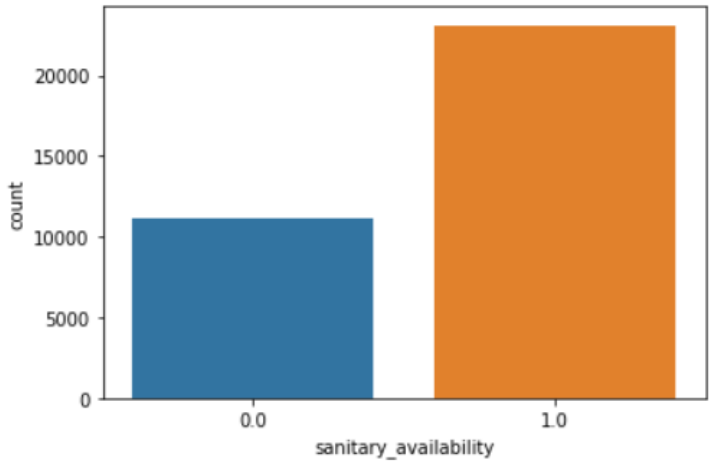
```
# df['sanitary_availability'].fillna(df['sanitary_availability'].mean(),inplace=True)
```

```
# df.sanitary_availability.value_counts()
```

Missing records relevant to sanitory_availability is dropped from the analysis scope.

Finally, **after sanitory_availabity feature imputation** the final count range relevant to each subcategory plot shown below.

```
sns.countplot(df.sanitary_availability)
plt.show()
```



Not a bad distribution situation!!!! compared to previous some imbalance classes seen in old_dependents, young dependents, home ownership, occupant count predictor features etc.

17.water_availabity:

Similar feature imputation approach (refer: **sanitory_availabity)** is used here too.

Observation: There is 4340 missing records in the sanitory_availability feature. These missing value records are dropped from the analysis scope.

```
print(df.water_availabity.nunique())
print(df.water_availabity.unique())
```

```
3
[0.5 nan 1.  0. ]
```

```
df.water_availabity.value_counts()
```

```
1.0    15434
0.5    14303
0.0      139
Name: water_availabity, dtype: int64
```

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

```
4340
```

```
df.shape
```

```
(34216, 501)
```

```
df.dropna(subset=['water_availabity'], inplace=True)
```

```
df.shape
```

```
(29876, 501)
```

```
print(df.water_availabity.unique())
```

```
[0.5 1.  0. ]
```

```
df.water_availabity.replace(0.5, 0.0, inplace=True)
```

```
df.water_availabity.value_counts()
```

```
1.0    15434
0.0    14442
Name: water_availabity, dtype: int64
```

```
df.shape
```

```
(29876, 501)
```

```
# df.water_availabity.mean()
```

```
# df['water_availabity'].fillna(df['water_availabity'].mean(),inplace=True)
```

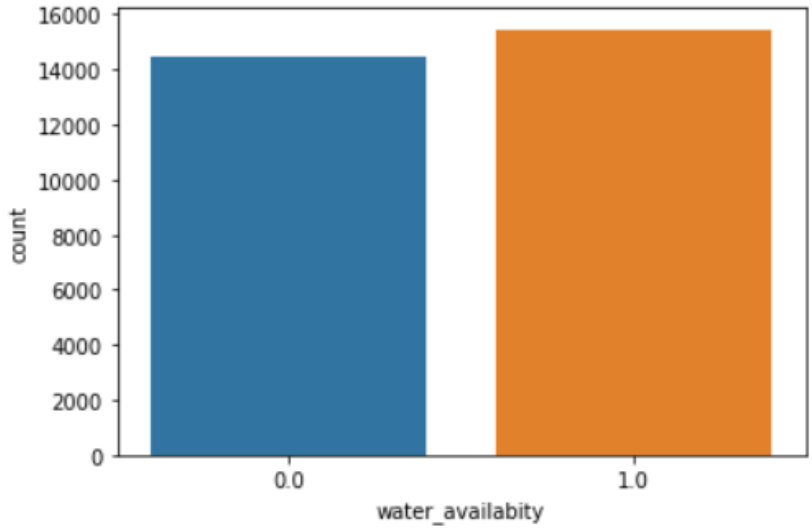
```
# df.water_availabity.value_counts()
```

After dropping missing feature records in the water_availability feature there exists three sub-categories with following count pattern:

{1.0: 15434,
0.5: 14303**,**
0.0: 139**}**

Here 0.5 subcategory clubbed with 0.0 class so that we have a balanced subcategory in water_availability predictor. **Following plot shows the impact of the outcome**.

```
sns.countplot(df.water_availabity)
plt.show()
```



Finally, after water_availability predictor data pre-processing, we have **data frame size = 29876 (rows) X 501 (columns+TARGET).**

Now, water_availability feature imputation completed!!! which is ready to go in perfect way!!!

18.loan_purpose:

Here go…. this loan_purpose predictor feature needs detail analysis because….it is one of the objective of the problem statement to be achieved which I quote below….

"Is loan_purpose a significant predictor? The business has insisted on using loan_purpose as a predictor. If it is not already a significant contributor, can we still modify the model to include it?"

Keeping in mind the above quote/task, we will go ahead……so that we can answer or interpret model effectively in case if business insists to include.

Note: There are 26 records of loan_purpose initially missing in the provided trainingData set. Fortunately, **those records vanished away during the data pre-processing or imputation process**.

Now loan_pupose predictor feature is free from any missing values in the record!!!!!!!

There are **36 subcategories** of **different industrial sectors** and its count (frequency of occurrence) in the trainingData provided are listed in **loan_purpose** predictor feature as listed below.

```
# len(df.loan_purpose.value_counts())
print(df.loan_purpose.nunique())
print(df.loan_purpose.unique())
```

```
36
['Apparels' 'Beauty Salon' 'Retail Store' 'Eateries' 'Meat Businesses'
 'Animal husbandry' 'Agro Based Businesses' 'Farming/ Agriculture'
 'Retail Sale' 'Carpentery work' 'Construction Related Activities'
 'Business Services - II' 'Tobacco Related Activities' 'Repair Services'
 'Laundry Services' 'Food Items' 'Handicrafts' 'Business Services - I'
 'Transportation Services' 'Flower Business' 'Artifical Jewellry Selling'
 'Jewellry Shop' 'Agarbatti Business' 'Miscellaneous' 'Education Loan'
 'Recycling/ Waste Management' 'Tuition Centre' 'Utensil Selling'
 'Cyber Caf_' 'Others' 'Training' 'Professional' 'Cable TV Services'
 'Tent Services' 'Sanitation' 'Manufacturing']
```

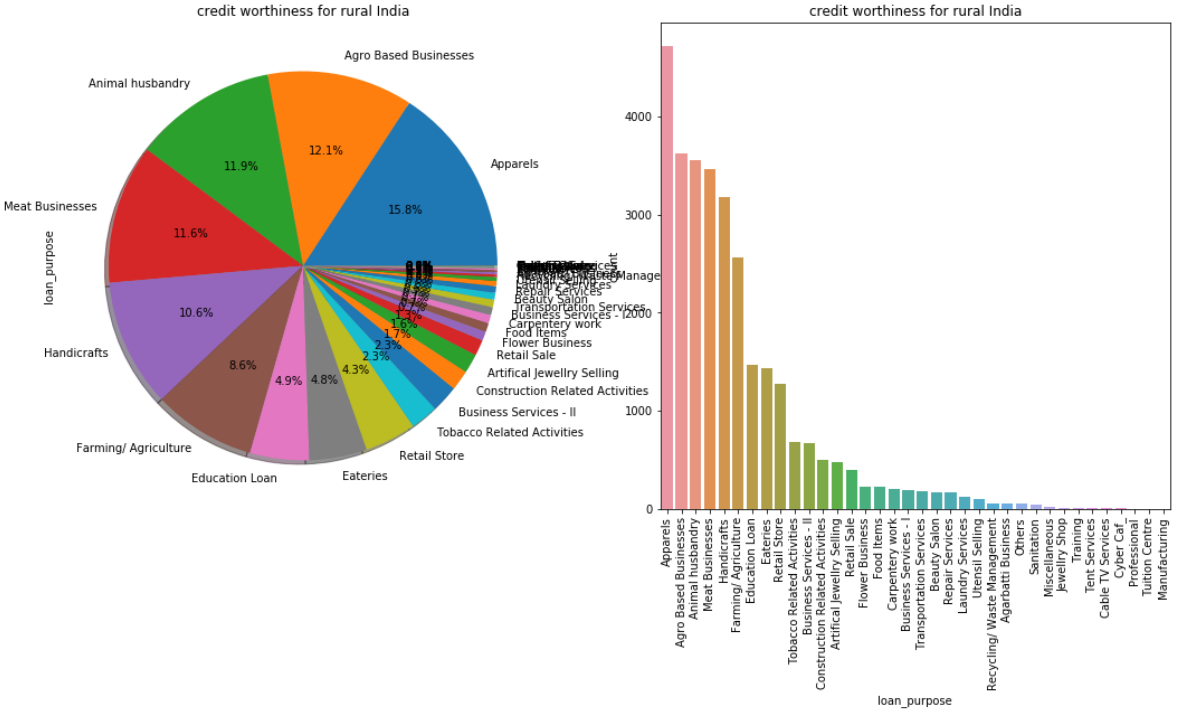
Apparels	4719
Agro Based Businesses	3623
Animal husbandry	3553
Meat Businesses	3460
Handicrafts	3178
Farming/ Agriculture	2560
Education Loan	1468
Eateries	1433
Retail Store	1277
Tobacco Related Activities	685
Business Services - II	674
Construction Related Activities	501
Artifical Jewellry Selling	478
Retail Sale	395
Flower Business	224
Food Items	223
Carpentery work	205
Business Services - I	198
Transportation Services	186
Beauty Salon	169
Repair Services	166
Laundry Services	127
Utensil Selling	99
Recycling/ Waste Management	61
Agarbatti Business	57
Others	52
Sanitation	45
Miscellaneous	16
Jewellry Shop	15
Training	7
Tent Services	6
Cable TV Services	5
Cyber Caf_	5
Professional	3
Tuition Centre	2
Manufacturing	1

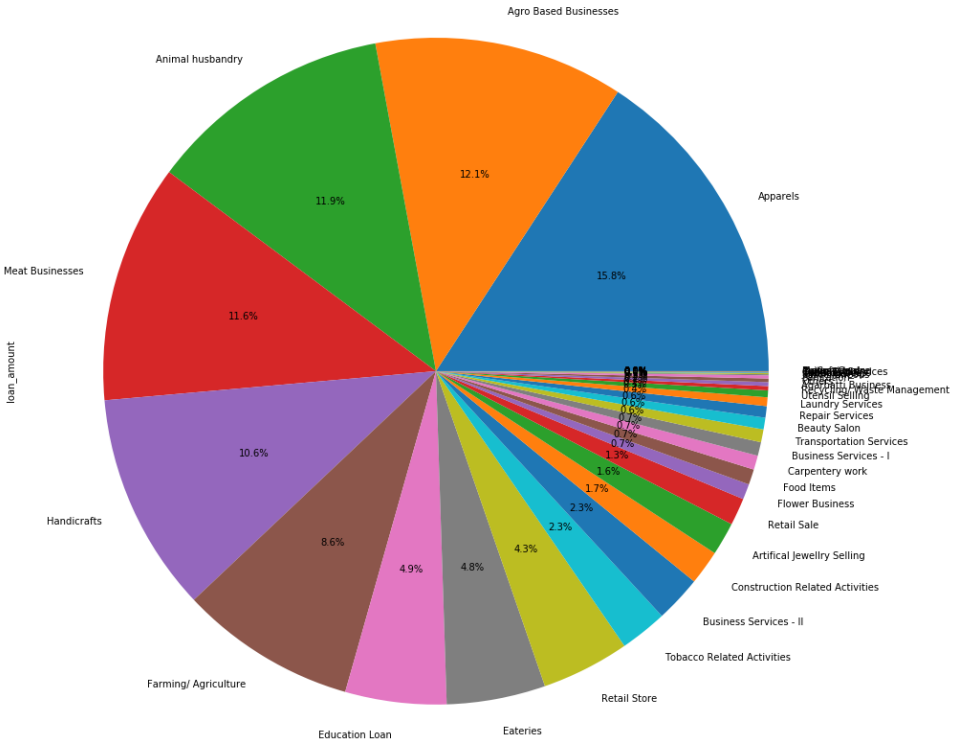
Following data indicates the % of loan_purpose in highest to lowest sorting order. **Apparels industry** stands in the top subcategory of loan_purpose.

```
Apparels = 0.15795287187039764
Beauty Salon = 0.005656714419601017
Retail Store = 0.042743339135091714
Eateries = 0.04796492167626188
Meat Businesses = 0.11581202302851787
Animal husbandry = 0.11892488954344624
Agro Based Businesses = 0.12126790735038158
Farming/ Agriculture = 0.08568750836792074
Retail Sale = 0.01322131476770652
Carpentery work = 0.006861695006024903
Construction Related Activities = 0.016769313161065738
Business Services - II = 0.022559914312491634
Tobacco Related Activities = 0.022928102825010042
Repair Services = 0.00555629937073236
Laundry Services = 0.004250903735439818
Food Items = 0.007464185299236845
Handicrafts = 0.10637300843486411
Business Services - I = 0.0066273932253313695
Transportation Services = 0.006225733029856741
Flower Business = 0.007497656982193065
Artifical Jewellry Selling = 0.0159994644530727
Jewellry Shop = 0.0005020752443432856
Agarbatti Business = 0.0019078859285044851
Miscellaneous = 0.0005355469272995046
Education Loan = 0.04913643057972955
Recycling/ Waste Management = 0.0020417726603293614
Tuition Centre = 6.694336591243807e-05
Utensil Selling = 0.0033136966126656848
Cyber Caf_ = 0.0001673584147810952
Others = 0.00174052751372339
Training = 0.00023430178069353328
Professional = 0.00010041504886865712
Cable TV Services = 0.0001673584147810952
Tent Services = 0.00020083009773731424
Sanitation = 0.0015062257330298568
Manufacturing = 3.3471682956219036e-05
```

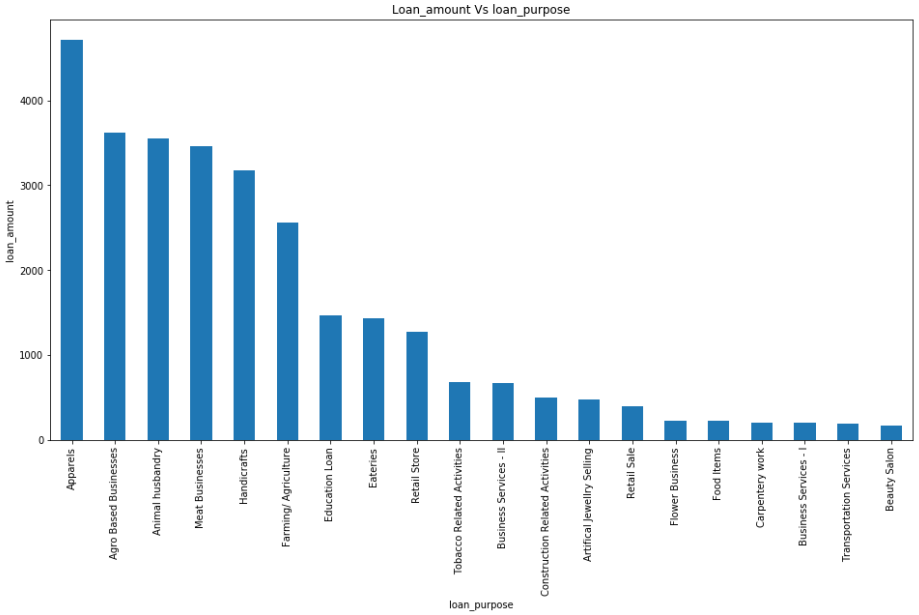
Pictorial view of the loan_purpose subcategories are shown using pie and bar chart as follows:

```
f,ax=plt.subplots(1,2,figsize=(18,8))
df['loan_purpose'].value_counts().plot.pie(autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('credit worthiness for rural India')
#ax[0].set_ylabel('Count')
sns.countplot('loan_purpose',data=df,ax=ax[1],order=df['loan_purpose'].value_counts().index)
ax[1].set_title('credit worthiness for rural India')
plt.xticks(rotation=90)
#ax[1].set_xticklabels(rotation=30)
plt.show()
```





```
plt.figure(figsize=(16,8))
all_loan_purpose = df.groupby([df.loan_purpose])['loan_amount'].count()
all_loan_purpose.sort_values(ascending = False)[0:20].plot(kind='bar')
plt.title ("Loan_amount Vs loan_purpose")
plt.xlabel("loan_purpose")
plt.ylabel("loan_amount")
plt.show()
```



Now we have data frame size = 29876 (rows) X 501 (columns+TARGET) as per below data.

```
df.shape
```

```
(29876, 501)
```

```
df.dropna(subset=['loan_purpose'], inplace=True)
```

```
df.shape
```

```
(29876, 501)
```

```
# df.loan_purpose.value_counts()
```

```
len(df.loan_purpose.value_counts())
```

```
36
```

As per above data there are 36 unique sub-categories present in loan_prupose predictor. So, if business insists to implement loan_prupose predictor feature in the ML model then our model data shape or dimension will turn into as follows: data frame size = 29876 (rows) X (501+(36-1-1)), (columns+TARGET) which turns out to be **data frame size = 29876 (rows) X 535 (columns+TARGET).** This can be achieved using OHE as shown below:

```
#####################OHE for loan_purpose feature#################################
df_loan_purpose = pd.get_dummies(df.loan_purpose, prefix='loan_purpose', drop_first=True)
# df_loan_purpose
df = pd.concat([df, df_loan_purpose], axis=1)
df = df.drop(columns=['loan_purpose'], axis=1)
#####################################################################
df.shape
(29876, 535)
```

There are lot of techniques available to cook or transfer this raw data into expected form for ML model requirement point of view.

As it is asked in the objective task, so I am just adding one more simple technique to handle this as shown below:

```
# print(df.isnull().sum())
# df.head()
# df.columns.tolist()
# df['loan_purpose']=df['loan_purpose'].map({"Apparels":1,"Agro Based Businesses":2,"Animal husbandry":3, "Meat Businesses":4,
# "Handicrafts":5,"Farming/ Agriculture":6,"Education Loan":7,"Retail Store":8,
# "Eateries":9,"Business Services - II":10,"Tobacco Related Activities":11,
# "Construction Related Activities":12,"Retail Sale":13,"Artifical Jewellry Selling":14,
# "Carpentery work":15,"Food Items":16,"Business Services - I":17,"Transportation Services":18,
# "Flower Business":19,"Beauty Salon":20,"Repair Services":21,"Laundry Services":22,
# "Agarbatti Business":23,"Utensil Selling":24,"Sanitation":25,"Recycling/ Waste Management":26,
# "Others":27,"Vocational Loans":28,"Jewellry Shop":29,"Training":30,"Miscellaneous":31,
# "Cyber Caf_":32,"Tent Services":33,"Cable TV Services":34,"Professional":35,
# "Tuition Centre":36,"Manufacturing":37})
# df['loan_purpose'].fillna(1, inplace=True) # 26 nulls are filled with max cout values
# df.loan_purpose.dtypes
# df['loan_purpose']=df['loan_purpose'].astype(int)
# df.loan_purpose.dtypes
# print(df.isnull().sum())
# df.info()
```

The above approach implemented as per weightage impact factor applied to each subcategories of loan_purpose predictor feature as per its frequency in the dataset.

Finally, after loan_purpose data pre-processing, we have **data frame size = 29876 (rows) X 535 (PREDICTORS+TARGET)**.

Now, loan_purpose feature imputation completed!!!

Hope, I also answered **PARTIALLY** the objective quoted above/below too.....

"Is loan_purpose a significant predictor? The business has insisted on using loan_purpose as a predictor. If it is not already a significant contributor, can we still modify the model to include it?"

Yes, it is possible, loan_purpose predictor can be modified and allowed to participate in the ML model training activity for the better prediction purpose.

"Is loan_purpose a significant predictor?....I will answer this portion of question once I build the model….train the model……interpreting/visualizing the results. This I owe it!!!

Note:

- **Not always** more predictors or adding predictors or considering all existing independent/predictor features **will yield the expected objective or accuracy in the model**. It may also suffer from overfitting or sometimes model will not give expected results.
- **So, considering only relevant predictor feature is the wise decision in ML model building applications.**
- In turns it adds **unnecessary burden on the business** which will have **many impacts like delay in processing the data** & **due to high number of predictors/records sometimes the model complexity increases which reduces its interpretability**.
- **Sometimes simple ML algorithms model solve the purpose**, so take away from above discussion is that **keep the model simple and concise as for as possible**.
- Simple is Best: Occam's Razor principle in Data Science. Simple meaning it should be competitively simple compared to other models. If considered model is too simple it may experience **under fitting issue**.
- Here I am going through all the predictor features because of my assessment. Otherwise in real scenario sitting with business leader / client we have fair idea which all predictors influence at some extent.

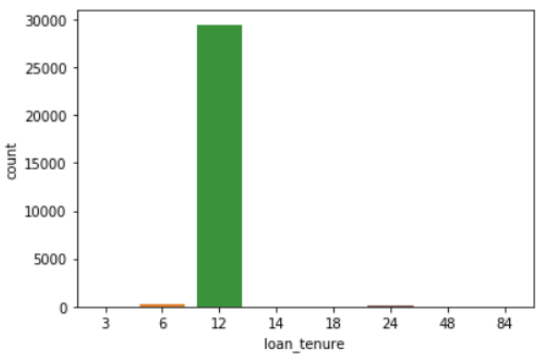
19. loan_tenure:

Note: There is no missing records in the loan_tenure predictor.

From the following plot it is evident that imbalance exists in the loan_tenure predictor feature.

```
sns.countplot(df.loan_tenure)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f28f48b308>
```



Minor subcategories of loan_tenure predictor is dropped from the analysis scope as shown below how the process implemented.

```
# df.loan_tenure
# df.loan_tenure.nunique() # 8
df.loan_tenure.unique()
```

```
array([12, 24, 84,  3,  6, 18, 14, 48], dtype=int64)
```

```
df.loan_tenure.dtypes
```

```
dtype('int64')
```

```
df.loan_tenure.value_counts()
```

```
12    29447
6       313
24      106
48        3
84        2
3         2
18        2
14        1
Name: loan_tenure, dtype: int64
```

```
df.shape
```

```
(29876, 535)
```

```
# df[(df.loan_tenure == 48) | (df.loan_tenure == 84) | (df.loan_tenure == 3) | (df.loan_tenure == 18) | (df.loan_tenure == 14)].index
df.drop(index=df[(df.loan_tenure == 48) | (df.loan_tenure == 84) | (df.loan_tenure == 3) | (df.loan_tenure == 18) | (df.loan_tenure == 14)].index,
```

```
df.shape
```

```
(29866, 535)
```

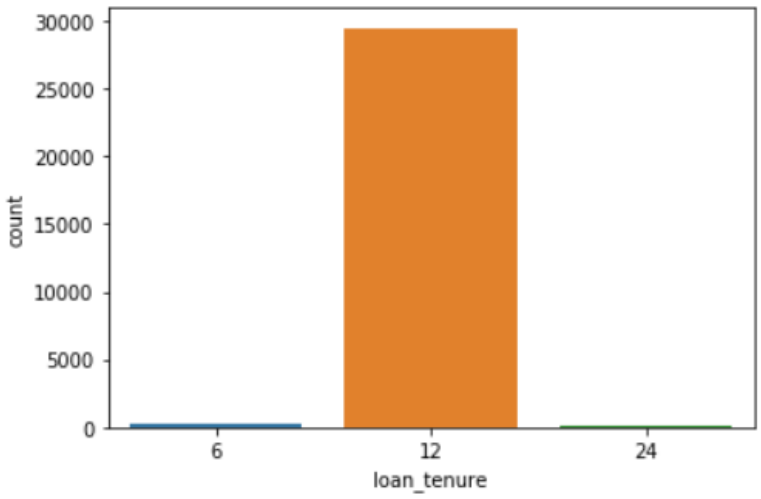
After imputation, still imbalance exists as shown below:

```
df.shape
```

```
(29866, 535)
```

```
sns.countplot(df.loan_tenure)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f28f33fd88>
```




```
df.loan_tenure.value_counts()
```

```
12    29447
6       313
24      106
Name: loan_tenure, dtype: int64
```

As of now, the data count and its sub_category are **NOT IN** perfectly accomplished weightage factor on its own!!!, which we seen in case of old_dependends feature. So, this feature **IS NOT** ready to go for ML model implementation until either **OHE** or correct **weightage factor assignment**!

Finally, after loan_tenure data pre-processing, we have **data frame size = 29866 (rows) X 535 (columns+TARGET)**.

Now, loan_tenure feature imputation completed!!! (Of course it needs either OHE or weight factor impact assignment for the subcategories of loan_tenure to implement in ML model)

20.loan_installments:

From the below statistical analysis it is evident that, there is no missing values in the loan_installments.

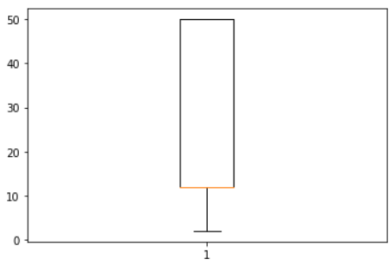
```
# df.loan_installments.dtypes # dtype('int64')
# df.loan_installments.isnull().sum() # 0
# df.loan_installments.nunique() # 16
df.loan_installments.sort_values().unique()
```

```
array([ 2,  4,  6,  8, 12, 24, 50], dtype=int64)
```

Also, there were no outliers which is concluded from below box plot.

```
plt.boxplot(df.loan_installments)
plt.plot()
```

```
[]
```



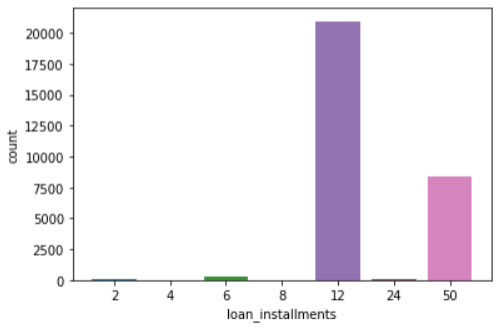

```
df.loan_installments.value_counts()
```

```
12    20961
50     8439
6       314
24       99
2        44
8         7
4         2
Name: loan_installments, dtype: int64
```

However, there huge imbalance in the subcategories of loan_installments as shown below

```
sns.countplot(df.loan_installments)
plt.plot()
```

```
[]
```



As of now, the data count and its sub_category are **NOT IN** perfectly accomplished weightage factor on its own!!!, which we seen in case of old_dependends feature. So, this feature **IS NOT** ready to go for ML model implementation until either **OHE** or correct **weightage factor assignment**!

Finally, after loan_installment data pre-processing, we have **data frame size = 29866 (rows) X 535 (columns+TARGET)**.

Now, loan_installment feature imputation completed!!! (Of course, it needs either OHE or weight factor impact assignment for the subcategories of loan_installment to implement in ML model)

Note: Feature scaling (either Standardised or Normalise etc) of regression model is necessary including both predictors and target feature due to the high variance of the received data.

Now it marks the end of predictors assessment (total 20 features assessed in detailed manner). Start Target feature assessment next…….

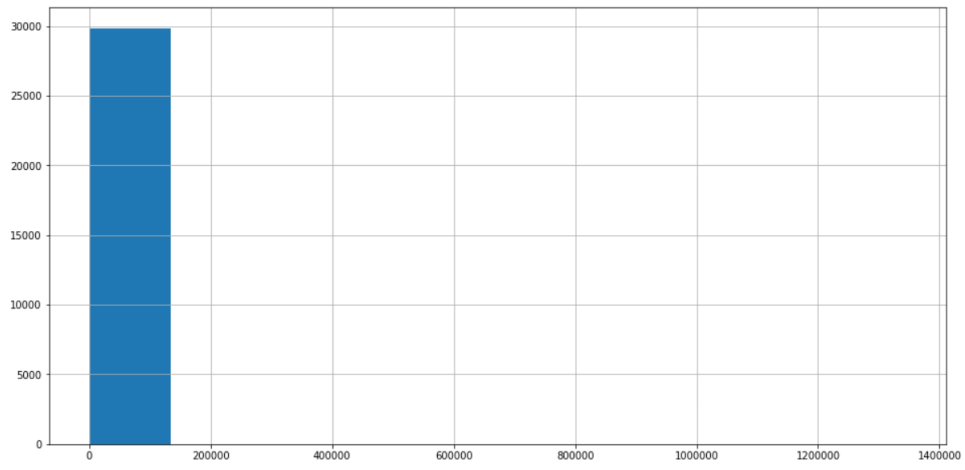
- **Target feature (01 number):**

21. loan_amount:

Target feature "l**oan_amount**" is continuous in nature, so it is a REGRESSION PROBLEM!

The histogram of Target feature "l**oan_amount**" shows distribution of data as shown below.

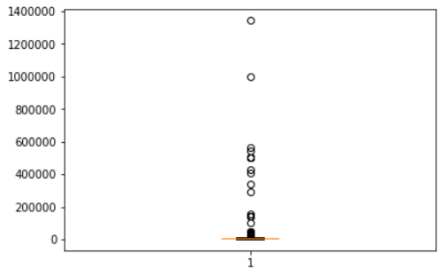
```
df.loan_amount.hist(figsize=(16,8))
plt.show()
```



The boxplot and distplot of target feature "l**oan_amount**" shows there exists an outlier in it as shown below.

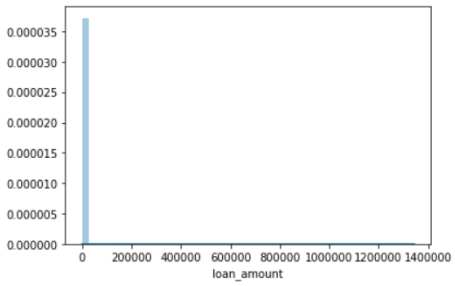
```
plt.boxplot(df.loan_amount)
plt.plot()
```

```
[]
```




```
sns.distplot(df.loan_amount)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f290c7c288>
```



The outlier treatment is done using five point summary / box and whisker plot as shown below:

```
# Detecting outlier
print(df.loan_amount.min())
print(df.loan_amount.max())
print(df.loan_amount.median())
print(df.loan_amount.quantile(0.25))
print(df.loan_amount.quantile(0.75))
IQR = df.loan_amount.quantile(0.75) - df.loan_amount.quantile(0.25)
print(IQR)

UQR = df.loan_amount.quantile(0.75) + 1.5 * IQR
LQR = df.loan_amount.quantile(0.25) - 1.5 * IQR
print(LQR)
print(UQR)
```

```
500.0
1343000.0
8000.0
5000.0
10000.0
5000.0
-2500.0
17500.0
```

```
df[(df.loan_amount < -2500.0) | (df.loan_amount > 17500.0)].index
```

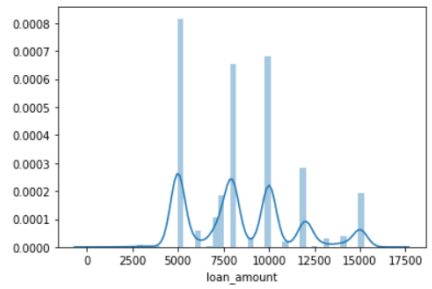
```
Int64Index([ 2341,  2343,  2799,  2836,  2860, 15295, 15536, 15537, 15538, 15539,
            ...
            39570, 39571, 39573, 39579, 39582, 39585, 39588, 39591, 39592, 39615], dtype='int64', length=101)
```

```
df.drop(index=df[(df.loan_amount < -2500.0) | (df.loan_amount > 17500.0)].index, inplace=True)
```

After outlier treatment, result is the following clean data for target feature loan_amount as shown below.

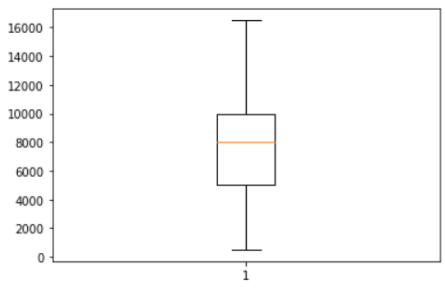
```
sns.distplot(df.loan_amount)
plt.plot()
```

```
[]
```




```
plt.boxplot(df.loan_amount)
plt.plot()
```

```
[]
```



```
df.shape
```

```
(29765, 535)
```

Finally, after all predictors and target_feature data pre-processing, we have **final data frame size = 29765 (rows) X 535 (columns+TARGET)**.

Now it marks the end of target feature assessment (total 01 features assessed in detailed manner).

Completed!!! **Section-1**: Do a descriptive analysis of all the variables.

Now Start **Section-2** model building and assessment activities.....

IMPORTANT:
Saved the data set in the form of CSV with the following name:
Cleaned_trainingData_for_assignment_DEATAILED_FINAL_07062020.csv

But in report file name is:
Cleaned_trainingData_for_assignment_08062020

Now the above-named csv is the master copy for our model building activities.

Section-2: There is a new customer who needs a loan. Which models will be best suited to predict the loan_amount that can be granted to the customer?

- **The given problem is REGRESSION category (because of continuous target feature), so the REGRESSION ALGORITHS can be used to predict the loan_amount that can be granted to the new customer**.

- Most common algorithms are **Linear Regression, Ridge and Lasso Regression: L1 and L2 Regularization, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor**.

- If data is in nonlinear passion, then it is advisable to use **SVR** or **KNeighborsRegressor** which are computationally very expensive.

- It is very difficult to judge the given data without preliminary analysis whether it is in **linearly separable or not**.

- **Thumb rule to use very computationally expensive nonlinear algorithm like SVR or KNeighborsRegressor** is that if linear algorithms not in a position to improve **R2 score** or **reduce the cost function error (RMSE)** after all the **hyperparameter tuning**. Even after using nonlinear algorithms if there is no improvement in results then it can be concluded that the data are not in nonlinear space. If there is an improvement in results it indicates the data supplied are in nonlinear space.

Section-3: Build a model to predict the maximum loan_amount that can be granted to the customer. Which all variables are good predictors?

Iteration-1: Considered complete data set as input the ML model training.

Independent and dependent feature creation:

Independent features (X):

```
# Model used all the INDEPENDENT/PREDICTOR features to predict the TARGET/DEPENDENT feature
X = df.drop(['Id','loan_amount'], axis=1)
```

```
df.shape
```

```
(29765, 535)
```

The complete model used has the dimension = 29765(rows) X 535 (columns only predictors)

Dependent features (y):

```
# feature_class = df.iloc[:, df.columns == 'loan_amount']
# target
# y = df.pop("loan_amount")
y = df['loan_amount']
```

```
X=X.iloc[:, :].values
```

```
X
```

```
y = y.iloc[:].values
```

```
y
```

Split data into a train-test dataset:

```
# training_set, test_set, class_set, test_class_set = train_test_split(feature_space,feature_class,test_size = 0.20,random_state = 42)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(23812, 533) (5953, 533) (23812,) (5953,)
```

```
# # Cleaning test sets to avoid future warning messages
# class_set = class_set.values.ravel()
# test_class_set = test_class_set.values.ravel()
```

```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

Feature scaling:

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Data prediction starts...

Linear Regression:

```
lr_reg = LinearRegression().fit(X_train, y_train)
lr_reg
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred = lr_reg.predict(X_test)
y_pred
```

```
array([7394.87713922, 8512.87713922, 8486.87713922, ..., 8102.87713922,
       8746.87713922, 6146.87713922])
```

```
#Checking accuracy on training data
accuracy = lr_reg.score(X_train,y_train)
print(accuracy*100,'%')
```

```
37.18135505943445 %
```

```
#Checking accuracy on test data
accuracy = lr_reg.score(X_test,y_test)
print(accuracy*100,'%')
```

```
-1.5066618516091085e+28 %
```

Note: By obserwing train and test accuray, the model needs **Review!!!!!**. As it is just a trail run with all predictors. Hoping later iterations it may get improved results!!!!.

```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_pred, y_test)
```

```
1.2675585785264401e+33
```

```
rmse = np.sqrt(mean_squared_error(y_pred, y_test))
rmse
```

```
3.5602788915005524e+16
```

```
r2_score(y_pred, y_test)
```

```
-1.7996992442537163e-05
```

```
# X_test #the test data - predictors
# y_test #the actual values in test data - target column
# y_pred
# lr_reg.predict(X_test) #predcited values on test data
# lr_reg.intercept_ #checking the intercept of model equation
# lr_reg.coef_ #Checking the coefficients of model equation
```

```
# mean squared error
mse = np.sum((y_pred - y_test)**2)
print("mse: ", mse)

# root mean squared error
# m is the number of training examples
rmse = np.sqrt(mse)
print("rmse: ",rmse)
```

```
mse:  7.545776217967897e+36
rmse:  2.7469576294453284e+18
```

Random Forest Regressor:

```
from sklearn.ensemble import RandomForestRegressor
```

```
## modelling
rf_regr = RandomForestRegressor(max_depth=4, random_state=42,n_estimators=25)
# training
rf_regr.fit(X_train, y_train)

##### model evaluation
print('On train data: r^2 score', r2_score(y_train, rf_regr.predict(X_train)))
print('On test data: r^2 score', r2_score(y_test, rf_regr.predict(X_test)))
```

```
On train data: r^2 score 0.352503478845135
On test data: r^2 score 0.3420038942293002
```

```
## modelling

rf_regr = RandomForestRegressor(oob_score = True)
rf_regr.fit(X_train,y_train)  # training
##### model evaluation
print('On train data: r^2 score', r2_score(y_train, rf_regr.predict(X_train)))
print('On test data: r^2 score', r2_score(y_test, rf_regr.predict(X_test)))
```

```
On train data: r^2 score 0.9233836287499105
On test data: r^2 score 0.5812131130018995
```

```
scores = cross_val_score(RandomForestRegressor(), X_train, y_train, scoring = 'r2',cv = 3 )
print('standard deviation:',np.std(scores))
print('Average:',np.mean(scores))
print(scores)
```

```
standard deviation: 0.0022893037240534282
Average: 0.5613345833543116
[0.558105   0.56275266 0.5631461 ]
```

```
# y_pred=rf_regr.predict(X_test)
# rmse(y_pred,y_test)
```

```
#Checking accuracy on training data
accuracy = rf_regr.score(X_train,y_train)
print(accuracy*100,'%')
```

```
92.33836287499105 %
```

```
#Checking accuracy on test data
accuracy = rf_regr.score(X_test,y_test)
print(accuracy*100,'%')
```

```
58.12131130018995 %
```

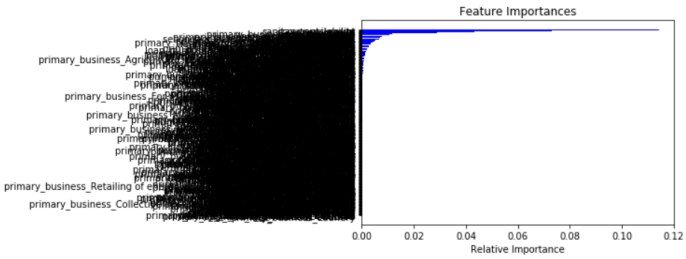
```
r2_score(y_pred, y_test)
```

```
0.38772973711965364
```

```
# rf_regr.feature_importances_
```

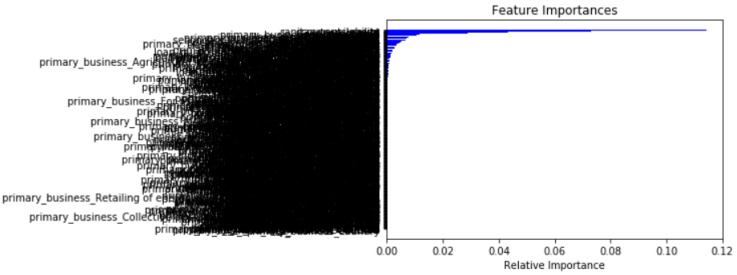
```
features=df.columns
importances = rf_regr.feature_importances_
indices = np.argsort(importances)

plt.figure(1)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), features[indices])
plt.xlabel('Relative Importance')
```

```
Text(0.5, 0, 'Relative Importance')
```




```
# features=df.columns[[3,4,6,8,9,10]]
features=df.columns
importances = rf_regr.feature_importances_
indices = np.argsort(importances)

plt.figure(1)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), features[indices])
plt.xlabel('Relative Importance')
```

```
Text(0.5, 0, 'Relative Importance')
```



From the RF regressor model it indicates primary_business predictor is the most important feature. As of now it is a trial iteration, so we cannot be decisive at this moment.

GradientBoostingRegressor:

```
param = {'n_estimators': 500,
         'max_depth': 4,
         'min_samples_split': 5,
         'learning_rate': 0.01,
         'loss': 'ls'}
```

```
lgb_reg = GradientBoostingRegressor(**param)
lgb_reg.fit(X_train, y_train)
print(rmse(lgb_reg.predict(X_test), y_test))
```

```
2137.831660420529
```

```
#Checking accuracy on training data
accuracy = lgb_reg.score(X_train,y_train)
print(accuracy*100,'%')
```

```
47.812360742978804 %
```

```
#Checking accuracy on test data
accuracy = lgb_reg.score(X_test,y_test)
print(accuracy*100,'%')
```

```
45.675622012429564 %
```

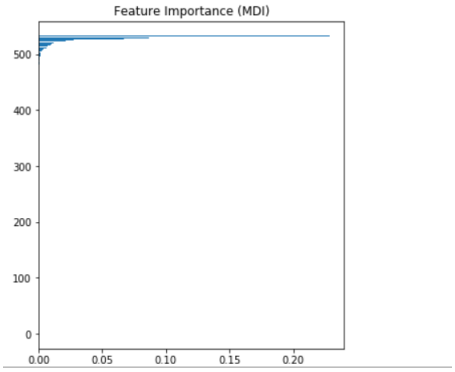
```
r2_score(lgb_reg.predict(X_test), y_test)
```

```
-0.6826790479161062
```

```
#Feature important 'purpose_loan'
```

```
plt.figure(figsize=(20,12))
feature_importance = lgb_reg.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align='center')
# plt.yticks(pos, np.array(temp[cols_to_use])[sorted_idx])
plt.title('Feature Importance (MDI)')
#fig.tight_layout()
plt.show()
```

```
<Figure size 1440x864 with 0 Axes>
```



Gradent Boosting Regressor gives the rmse = 2137 which is the lowest in all algorithms.

Observation from Iteration-1:

- **All the predictor features available used in the model,** it is **observed that the model complexity increased multi fold as well as computationally expensive.**
- Due to a huge increase in predictor feature number, now model needs **feature reduction techniques** like **Principal Component Analysis (PCA)** etc.
- As observed in the above EDA study, there were not much considerable correlation between predictor feature, so if business orders to consider all the predictor features then it needs along with PCA, hyperparameter tuning using grid search CV or random search CV to improve the model performance (RMSE, R2 score and accuracy).
- Now we will do further iterations to building/training the model by considering most important predictor feature against target feature (loan_amount) for prediction.
- **Currently MACHINE LEARNING MODEL set up done in all aspect. For model performance improvement further study or iterations will be carried out**.

Section-4: Build at least one model from scratch that fits this data, without using any third-party packages like sklearn, glm, lm, rpart, etc. You are free to use linear algebra packages like scipy, numpy or any blas derivative. We would be more interested in the convergence of the algorithm rather than the prediction accuracy.

Without using scikitlearn Library:

```
# Import required libraries packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Import trainingData_for_assignment.csv
df_1 = pd.read_csv("trainingData_for_assignment.csv")
# X = data[['annual_income', 'house_area', 'loan_tenure', 'loan_installments']]
X = data['annual_income']
Y = data['loan_amount']

class LinearRegression:
    def fit(self,X,Y):
        X=np.array(X).reshape(-1,1)
        Y=np.array(Y).reshape(-1,1)

        x_shape = X.shape

        num_var = x_shape[1]
        weight_matrix = np.random.normal(0,1,(num_var,1))
        intercept = np.random.rand(1)

        for i in range(50):
            dcostdm = np.sum(np.multiply(((np.matmul(X,weight_matrix)+intercept)-Y),X))*2/x_shape[0]
            dcostdc = np.sum(((np.matmul(X,weight_matrix)+intercept)-Y))*2/x_shape[0]
            weight_matrix -= 0.1*dcostdm
            intercept -= 0.1*dcostdc
        return weight_matrix,intercept
# print(df.drop(['sex'],axis=1))
reg = LinearRegression()
x = (df_1['annual_income']-df_1['annual_income'].mean())/df['annual_income'].std()
y = (df_1["loan_amount"]-df_1['loan_amount'].mean())/df["loan_amount"].std()
params = reg.fit(x,y)

plt.scatter(x[:180],y[:180])
pred = np.matmul(np.array(x[:2000]).reshape(-1,1),params[0])+params[1]
plt.plot(x[:2000],pred)
print(params)
```

- Using scikitlearn Library

Univariate Linear Regression Analysis (With Scikit-Learn Library)

```
# Import required libraries packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Import trainingData_for_assignment.csv
data = pd.read_csv("trainingData_for_assignment.csv")
data_X = data['annual_income'].values.reshape(-1, 1)
data_y = data.loan_amount.values.reshape(-1, 1)

# scikit-learn implementation
data_x_train, data_x_test, data_y_train, data_y_test = train_test_split(data_X, data_y, test_size=0.2, random_state=42)

sc = StandardScaler()
data_x_train = sc.fit_transform(data_x_train)
data_x_test = sc.transform(data_x_test)

# Model initialization
regression_model = LinearRegression()
# Fit the data(train the model)
regression_model.fit(data_x_train, data_y_train)
# Predict
y_predicted = regression_model.predict(data_x_test)

# model evaluation
rmse = mean_squared_error(data_y_test, y_predicted)
r2 = r2_score(data_y_test, y_predicted)

# printing values
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)
```

```
Slope: [[1190.47942491]]
Intercept: [8401.290625]
Root mean squared error:  228281176.8791922
R2 score:  0.004798460509626179
```

Mulativariate Linear Regression Analysis (With Scikit-Learn Library)

```
# Import required libraries packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Import trainingData_for_assignment.csv
data = pd.read_csv("trainingData_for_assignment.csv")
data_X = data[['annual_income', 'house_area', 'loan_tenure', 'loan_installments']]
data_y = data.loan_amount

# sckit-learn implementation
data_x_train, data_x_test, data_y_train, data_y_test = train_test_split(data_X, data_y, test_size=0.2, random_state=42)

sc = StandardScaler()
data_x_train = sc.fit_transform(data_x_train)
data_x_test = sc.transform(data_x_test)

# Model initialization
regression_model = LinearRegression()
# Fit the data(train the model)
regression_model.fit(data_x_train, data_y_train)
# Predict
y_predicted = regression_model.predict(data_x_test)

# model evaluation
rmse = mean_squared_error(data_y_test, y_predicted)
r2 = r2_score(data_y_test, y_predicted)

# printing values
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)
```

```
Slope: [1244.38789957  113.96589924  896.43713063  314.33767793]
Intercept: 8401.290625
Root mean squared error:  226343043.29524696
R2 score:  0.013247836638000776
```

Section-5: Is loan_purpose a significant predictor? The business has insisted on using loan_purpose as a predictor. If it is not already a significant contributor, can we still modify the model to include it?

- **Explained loan_purpose predictor implementation procedure in ML model at Section-1 which is dedicated to EDA activity.**

- **Is loan_purposesignificant predictor analysed below:**

Predictors list without loan_purpose to check its impact on the model predictability:

```
# # Is loan_purpose a significant predictor

# Without loan_purpose_predictor
X = df[['age','annual_income', 'monthly_expenses', 'house_area']]

# With loan_purpose_predictor
# X = df[['age','annual_income', 'monthly_expenses', 'house_area', 'loan_purpose_Agro Based Businesses',
#   'loan_purpose_Animal husbandry', 'loan_purpose_Apparels', 'loan_purpose_Artifical Jewellry Selling', 'loan_purpose_Beauty Salon',
#   'loan_purpose_Business Services - I', 'loan_purpose_Business Services - II', 'loan_purpose_Cable TV Services', 'loan_purpose_Carpentery work',
#   'loan_purpose_Construction Related Activities', 'loan_purpose_Cyber Caf_', 'loan_purpose_Eateries', 'loan_purpose_Education Loan', 'loan_purpose_Farming/ Agriculture',
#   'loan_purpose_Flower Business', 'loan_purpose_Food Items', 'loan_purpose_Handicrafts', 'loan_purpose_Jewellry Shop', 'loan_purpose_Laundry Services',
#   'loan_purpose_Manufacturing', 'loan_purpose_Meat Businesses', 'loan_purpose_Miscellaneous', 'loan_purpose_Others', 'loan_purpose_Professional',
#   'loan_purpose_Recycling/ Waste Management', 'loan_purpose_Repair Services', 'loan_purpose_Retail Sale', 'loan_purpose_Retail Store', 'loan_purpose_Sanitation',
#   'loan_purpose_Tent Services', 'loan_purpose_Tobacco Related Activities', 'loan_purpose_Training', 'loan_purpose_Transportation Services', 'loan_purpose_Tuition Centre',
#   'loan_purpose_Utensil Selling']]
```

Predictors list with loan_purpose to check its impact on the model predictability:

```
# # Is loan_purpose a significant predictor

# Without loan_purpose_predictor
# X = df[['age','annual_income', 'monthly_expenses', 'house_area']]

# With loan_purpose_predictor
X = df[['age','annual_income', 'monthly_expenses', 'house_area', 'loan_purpose_Agro Based Businesses',
 'loan_purpose_Animal husbandry', 'loan_purpose_Apparels', 'loan_purpose_Artifical Jewellry Selling', 'loan_purpose_Beauty Salon',
 'loan_purpose_Business Services - I', 'loan_purpose_Business Services - II', 'loan_purpose_Cable TV Services', 'loan_purpose_Carpentery work',
 'loan_purpose_Construction Related Activities', 'loan_purpose_Cyber Caf_', 'loan_purpose_Eateries', 'loan_purpose_Education Loan', 'loan_purpose_Farming/ Agriculture',
 'loan_purpose_Flower Business', 'loan_purpose_Food Items', 'loan_purpose_Handicrafts', 'loan_purpose_Jewellry Shop', 'loan_purpose_Laundry Services',
 'loan_purpose_Manufacturing', 'loan_purpose_Meat Businesses', 'loan_purpose_Miscellaneous', 'loan_purpose_Others', 'loan_purpose_Professional',
 'loan_purpose_Recycling/ Waste Management', 'loan_purpose_Repair Services', 'loan_purpose_Retail Sale', 'loan_purpose_Retail Store', 'loan_purpose_Sanitation',
 'loan_purpose_Tent Services', 'loan_purpose_Tobacco Related Activities', 'loan_purpose_Training', 'loan_purpose_Transportation Services', 'loan_purpose_Tuition Centre',
 'loan_purpose_Utensil Selling']]
```

Predictions comparison:

WITHOUT LOAN PURPOSE

GradientBoostingRegressor

```
# np.random.seed(42)
# start = time.time()

# param = {'n_estimators': 500,
#          'max_depth': 4,
#          'min_samples_split': 5,
#          'learning_rate': 0.01,
#          'loss': 'ls'}

# lgb_reg = GradientBoostingRegressor(**param)
# lgb_reg.fit(X_train, y_train)
# print(rmse(lgb_reg.predict(X_test), y_test))

# end = time.time()
# print('Time taken in grid search: {0: .2f}'.format(end - start))
```

```
param = {'n_estimators': 500,
         'max_depth': 4,
         'min_samples_split': 5,
         'learning_rate': 0.01,
         'loss': 'ls'}
```

```
lgb_reg = GradientBoostingRegressor(**param)
lgb_reg.fit(X_train, y_train)
print(rmse(lgb_reg.predict(X_test), y_test))
```

```
2450.3752104037444
```

```
#Checking accuracy on training data
accuracy = lgb_reg.score(X_train,y_train)
print(accuracy*100,'%')
```

```
34.352448424888869 %
```

```
#Checking accuracy on test data
accuracy = lgb_reg.score(X_test,y_test)
print(accuracy*100,'%')
```

```
33.378795292767684 %
```

```
r2_score(lgb_reg.predict(X_test), y_test)
```

```
-1.516880306389043
```

WITH LOAN PURPOSE

GradientBoostingRegressor

```
# np.random.seed(42)
# start = time.time()

# param = {'n_estimators': 500,
#          'max_depth': 4,
#          'min_samples_split': 5,
#          'learning_rate': 0.01,
#          'loss': 'ls'}

# lgb_reg = GradientBoostingRegressor(**param)
# lgb_reg.fit(X_train, y_train)
# print(rmse(lgb_reg.predict(X_test), y_test))

# end = time.time()
# print('Time taken in grid search: {0: .2f}'.format(end - start))
```

```
param = {'n_estimators': 500,
         'max_depth': 4,
         'min_samples_split': 5,
         'learning_rate': 0.01,
         'loss': 'ls'}
```

```
lgb_reg = GradientBoostingRegressor(**param)
lgb_reg.fit(X_train, y_train)
print(rmse(lgb_reg.predict(X_test), y_test))
```

```
2451.327474964986
```

```
#Checking accuracy on training data
accuracy = lgb_reg.score(X_train,y_train)
print(accuracy*100,'%')
```

```
36.5946903178478 %
```

```
#Checking accuracy on test data
accuracy = lgb_reg.score(X_test,y_test)
print(accuracy*100,'%')
```

```
36.43253201357266 %
```

```
r2_score(lgb_reg.predict(X_test), y_test)
```

```
-1.366597588416604
```

- From the above ML model prediction results it is possible to decide LOAN_PURPOSE predictor is not a significant predictor because there is not much difference in RMSE values with and without LOAN_PURPOSE predictor.
- However, it needs further analysis to take final decision about LOAN_PURPOSE predictor significance by considering other predictor combinations. After some iterations with different predictor if there is no influence on RMSE it should be out of model building activity.

Section-6: How will you measure the fitness of the model? Which metrics (accuracy, recall, etc.) are most relevant?

- A **well-fitting regression model** results in **predicted values close to the observed data values**. The mean model, which uses the mean for every predicted value, generally would be used if there were no informative predictor variables. The fit of a proposed regression model should therefore be better than the fit of the mean model.
- For regression problems **R-squared and Adjusted R-squared & RMSE** used as a measure of model performance.
- **Higher** the R2 value better the performance of the model.
- **Lower** the 'Mean Absolute Error (MAE)', 'Mean Squared Error (MSE)' and 'Root Mean Squared Error (RMSE)' better the performance of the model.
- For Classification problems model performance is measured using confusion metrics, accuracy, error rate, precision, recall, F1score, AUC-ROC. At the moment the current problem is regression, so these parameters are out of scope at the moment to discuss.