Exploratory Data Analysis

- Exploratory data analysis is a data exploration technique to understand the various aspects of the data. It is a kind of summary of data. It is one of the most important steps before performing any machine learning or deep learning tasks.
- Data Scientists carry out exploratory data analysis procedures to explore, dissect, and sum up the fundamental qualities of datasets, regularly using information representation approaches. EDA procedures take into consideration compelling control of information sources, empowering Data Scientists to discover the appropriate responses they need by finding information designs, spotting inconsistencies, checking suppositions, or testing speculation.
- Data Scientists utilize exploratory data analysis to observe what datasets can uncover further past conventional demonstrating of information or speculation testing assignments. This empowers them to acquire top to bottom information on the factors in datasets and their connections. Exploratory data analysis can help recognize clear mistakes, distinguish exceptions in datasets, get connections, uncover significant elements, discover designs inside information, and give new bits of knowledge.

Steps to perform Exploratory Data Analysis

Importing libraries

```
# Import the numpy, pandas, datetime, matplotlib, & seaborn packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Supress Warnings
import warnings
warnings.filterwarnings('ignore')
```

Loading the Data

```
loan = pd.read_csv('loan.csv')
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0   36
```

```
months
1  1077430    1314167       2500          2500           2500.0   60
months
2  1077175    1313524       2400          2400           2400.0   36
months
3  1076863    1277178      10000         10000          10000.0   36
months
4  1075358    1311748       3000          3000           3000.0   60
months

  int_rate  installment grade sub_grade  ... num_tl_90g_dpd_24m  \
0   10.65%       162.87     B         B2  ...                 NaN
1   15.27%        59.83     C         C4  ...                 NaN
2   15.96%        84.33     C         C5  ...                 NaN
3   13.49%       339.31     C         C1  ...                 NaN
4   12.69%        67.79     B         B5  ...                 NaN

  num_tl_op_past_12m pct_tl_nvr_dlq  percent_bc_gt_75
pub_rec_bankruptcies  \
0                NaN            NaN               NaN
0.0
1                NaN            NaN               NaN
0.0
2                NaN            NaN               NaN
0.0
3                NaN            NaN               NaN
0.0
4                NaN            NaN               NaN
0.0

  tax_liens tot_hi_cred_lim total_bal_ex_mort total_bc_limit  \
0       0.0             NaN               NaN            NaN
1       0.0             NaN               NaN            NaN
2       0.0             NaN               NaN            NaN
3       0.0             NaN               NaN            NaN
4       0.0             NaN               NaN            NaN

  total_il_high_credit_limit
0                        NaN
1                        NaN
2                        NaN
3                        NaN
4                        NaN

[5 rows x 111 columns]
```

Inspecting The Dataframe

```
# INFO
print("-"*20,"Info","-"*20)
loan.info()
```

```
-------------------- Info --------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39717 entries, 0 to 39716
Columns: 111 entries, id to total_il_high_credit_limit
dtypes: float64(74), int64(13), object(24)
memory usage: 33.6+ MB
```

```
# SHAPE
print("-"*20,"Shape of the Dataframe","-"*20)
print(loan.shape)
```

```
-------------------- Shape of the Dataframe --------------------
(39717, 111)
```

```
# DESCRIBE
print("-"*20,"Describe","-"*20)
print(loan.describe(include='all'))
```

```
-------------------- Describe --------------------
                  id     member_id     loan_amnt   funded_amnt  \
count   3.971700e+04  3.971700e+04  39717.000000  39717.000000
unique           NaN           NaN           NaN           NaN
top              NaN           NaN           NaN           NaN
freq             NaN           NaN           NaN           NaN
mean    6.831319e+05  8.504636e+05  11219.443815  10947.713196
std     2.106941e+05  2.656783e+05   7456.670694   7187.238670
min     5.473400e+04  7.069900e+04    500.000000    500.000000
25%     5.162210e+05  6.667800e+05   5500.000000   5400.000000
50%     6.656650e+05  8.508120e+05  10000.000000   9600.000000
75%     8.377550e+05  1.047339e+06  15000.000000  15000.000000
max     1.077501e+06  1.314167e+06  35000.000000  35000.000000

        funded_amnt_inv        term int_rate   installment  grade
sub_grade  \
count      39717.000000       39717    39717  39717.000000  39717
39717
unique              NaN           2      371           NaN      7
35
top                 NaN   36 months   10.99%           NaN      B
B3
freq                NaN       29096      956           NaN  12020
2917
mean       10397.448868         NaN      NaN    324.561922    NaN
NaN
std         7128.450439         NaN      NaN    208.874874    NaN
NaN
```

```
min            0.000000          NaN      NaN     15.690000    NaN
NaN
25%         5000.000000          NaN      NaN    167.020000    NaN
NaN
50%         8975.000000          NaN      NaN    280.220000    NaN
NaN
75%        14400.000000          NaN      NaN    430.780000    NaN
NaN
max        35000.000000          NaN      NaN   1305.190000    NaN
NaN

        ... num_tl_90g_dpd_24m num_tl_op_past_12m pct_tl_nvr_dlq  \
count   ...                0.0                0.0            0.0
unique  ...                NaN                NaN            NaN
top     ...                NaN                NaN            NaN
freq    ...                NaN                NaN            NaN
mean    ...                NaN                NaN            NaN
std     ...                NaN                NaN            NaN
min     ...                NaN                NaN            NaN
25%     ...                NaN                NaN            NaN
50%     ...                NaN                NaN            NaN
75%     ...                NaN                NaN            NaN
max     ...                NaN                NaN            NaN

        percent_bc_gt_75 pub_rec_bankruptcies tax_liens
tot_hi_cred_lim  \
count               0.0          39020.000000   39678.0
0.0
unique              NaN                   NaN       NaN
NaN
top                 NaN                   NaN       NaN
NaN
freq                NaN                   NaN       NaN
NaN
mean                NaN              0.043260       0.0
NaN
std                 NaN              0.204324       0.0
NaN
min                 NaN              0.000000       0.0
NaN
25%                 NaN              0.000000       0.0
NaN
50%                 NaN              0.000000       0.0
NaN
75%                 NaN              0.000000       0.0
NaN
max                 NaN              2.000000       0.0
NaN

       total_bal_ex_mort total_bc_limit total_il_high_credit_limit
```

```
count                 0.0            0.0                          0.0
unique                NaN            NaN                          NaN
top                   NaN            NaN                          NaN
freq                  NaN            NaN                          NaN
mean                  NaN            NaN                          NaN
std                   NaN            NaN                          NaN
min                   NaN            NaN                          NaN
25%                   NaN            NaN                          NaN
50%                   NaN            NaN                          NaN
75%                   NaN            NaN                          NaN
max                   NaN            NaN                          NaN

[11 rows x 111 columns]
```

```
# NULL VALUES
print("-"*20,"Columns having null Values","-"*20)
loan.isnull().sum()
```

```
-------------------- Columns having null Values --------------------

id                                  0
member_id                           0
loan_amnt                           0
funded_amnt                         0
funded_amnt_inv                     0
                                  ...
tax_liens                          39
tot_hi_cred_lim                 39717
total_bal_ex_mort               39717
total_bc_limit                  39717
total_il_high_credit_limit      39717
Length: 111, dtype: int64
```

```
#DATA TYPES
print("-"*20,"Data Types of Variables","-"*20)
loan.dtypes
```

```
-------------------- Data Types of Variables --------------------

id                                int64
member_id                         int64
loan_amnt                         int64
funded_amnt                       int64
funded_amnt_inv                 float64
                                  ...
tax_liens                       float64
tot_hi_cred_lim                 float64
total_bal_ex_mort               float64
total_bc_limit                  float64
total_il_high_credit_limit      float64
Length: 111, dtype: object
```

```
# UNIQUE VALUES
print("-"*20,"Unique Values in the dataset/target column","-"*20)
unique_columns = []

# Iterate through the columns in the DataFrame
for column in loan.columns:
    # Check if the number of unique values in the column is equal to
the total number of rows
    if loan[column].nunique() == loan.shape[0]:
        unique_columns.append(column)


unique_columns
```

```
-------------------- Unique Values in the dataset/target column
--------------------
```

```
['id', 'member_id', 'url']
```

```
non_numeric_columns = loan.select_dtypes(exclude=['number']).columns
print(non_numeric_columns)
```

```
Index(['term', 'int_rate', 'grade', 'sub_grade', 'emp_title',
'emp_length',
       'home_ownership', 'verification_status', 'issue_d',
'loan_status',
       'pymnt_plan', 'url', 'desc', 'purpose', 'title', 'zip_code',
       'addr_state', 'earliest_cr_line', 'revol_util',
'initial_list_status',
       'last_pymnt_d', 'next_pymnt_d', 'last_credit_pull_d',
       'application_type'],
      dtype='object')
```

Removing columns with 100% NULL VALUES

```
# check for columns that have 100% null values and remove them
cols = loan.columns
cols
```

```
Index(['id', 'member_id', 'loan_amnt', 'funded_amnt',
'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       ...
       'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'pct_tl_nvr_dlq',
       'percent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens',
       'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit'],
      dtype='object', length=111)
```

```
miss_percent = round(100*(loan.isnull().sum()/len(loan.index)), 2)
miss_percent
```

```
id                                0.0
member_id                         0.0
loan_amnt                         0.0
funded_amnt                       0.0
funded_amnt_inv                   0.0
                                 ...
tax_liens                         0.1
tot_hi_cred_lim                 100.0
total_bal_ex_mort               100.0
total_bc_limit                  100.0
total_il_high_credit_limit      100.0
Length: 111, dtype: float64
```

```
miss_df = pd.DataFrame({'column_name': cols,'percent_missing':
miss_percent})
miss_df
```

```
                                           column_name
percent_missing
id                                                  id
0.0
member_id                                    member_id
0.0
loan_amnt                                    loan_amnt
0.0
funded_amnt                                funded_amnt
0.0
funded_amnt_inv                        funded_amnt_inv
0.0
...                                                ...                ..
.
tax_liens                                    tax_liens
0.1
tot_hi_cred_lim                        tot_hi_cred_lim
100.0
total_bal_ex_mort                    total_bal_ex_mort
100.0
total_bc_limit                        total_bc_limit
100.0
total_il_high_credit_limit  total_il_high_credit_limit
100.0

[111 rows x 2 columns]
```

```
missing_drop = list(miss_df[miss_df.percent_missing ==
100.00].column_name)
missing_drop
```

```
['mths_since_last_major_derog',
 'annual_inc_joint',
 'dti_joint',
 'verification_status_joint',
 'tot_coll_amt',
 'tot_cur_bal',
 'open_acc_6m',
 'open_il_6m',
 'open_il_12m',
 'open_il_24m',
 'mths_since_rcnt_il',
 'total_bal_il',
 'il_util',
 'open_rv_12m',
 'open_rv_24m',
 'max_bal_bc',
 'all_util',
 'total_rev_hi_lim',
 'inq_fi',
 'total_cu_tl',
 'inq_last_12m',
 'acc_open_past_24mths',
 'avg_cur_bal',
 'bc_open_to_buy',
 'bc_util',
 'mo_sin_old_il_acct',
 'mo_sin_old_rev_tl_op',
 'mo_sin_rcnt_rev_tl_op',
 'mo_sin_rcnt_tl',
 'mort_acc',
 'mths_since_recent_bc',
 'mths_since_recent_bc_dlq',
 'mths_since_recent_inq',
 'mths_since_recent_revol_delinq',
 'num_accts_ever_120_pd',
 'num_actv_bc_tl',
 'num_actv_rev_tl',
 'num_bc_sats',
 'num_bc_tl',
 'num_il_tl',
 'num_op_rev_tl',
 'num_rev_accts',
 'num_rev_tl_bal_gt_0',
 'num_sats',
 'num_tl_120dpd_2m',
 'num_tl_30dpd',
 'num_tl_90g_dpd_24m',
 'num_tl_op_past_12m',
 'pct_tl_nvr_dlq',
 'percent_bc_gt_75',
```

```
 'tot_hi_cred_lim',
 'total_bal_ex_mort',
 'total_bc_limit',
 'total_il_high_credit_limit']

loan = loan.drop(missing_drop, axis=1)
loan.shape
```

```
(39717, 57)
```

```
# Dropping the columns on customer's information
loan = loan.drop(['title', 'desc', 'url', 'emp_title', 'zip_code'],
axis = 1)
# loan = loan.drop(['id','member_id'], axis=1)

# The following columns are not

loan = loan.drop(['last_credit_pull_d', 'last_pymnt_amnt',
'last_pymnt_d', 'collection_recovery_fee', 'recoveries',
                  'total_rec_late_fee', 'total_rec_int',
'total_rec_prncp', 'total_pymnt_inv', 'out_prncp',
                    'out_prncp_inv'], axis = 1)

# These variables are not necessary for variables so removing the is
logical.

loan = loan.drop(['delinq_2yrs', 'total_acc', 'pub_rec',
'inq_last_6mths', 'earliest_cr_line', 'collections_12_mths_ex_med',
                  'acc_now_delinq', 'chargeoff_within_12_mths',
'delinq_amnt',  'tax_liens', 'installment',
                   'mths_since_last_delinq', 'mths_since_last_record',
'revol_bal', 'revol_util', 'total_pymnt', 'next_pymnt_d',
                  'initial_list_status', 'pub_rec_bankruptcies',
'policy_code', 'application_type', 'pymnt_plan', 'open_acc'],
                 axis = 1)
loan.shape
```

```
(39717, 18)
```

```
import pandas as pd

def count_duplicates(df, column_name):
    """
    Count the number of duplicate values in a DataFrame column.

    Parameters:
    - df: The DataFrame to check for duplicates.
    - column_name: The name of the column to check for duplicates.

    Returns:
    - The count of duplicate values in the specified column.
```

```
    """
    duplicate_count = df[column_name].duplicated().sum()
    return duplicate_count

# Example usage:
# Assuming you have a DataFrame 'loan' and want to check duplicates in
the 'loan_id' column
# Replace 'loan' and 'loan_id' with your actual DataFrame and column
name
duplicates_count = count_duplicates(loan, 'id')
print("Number of duplicates:", duplicates_count)
```

```
Number of duplicates: 0
```

```
def count_duplicates_in_all_columns(df):
    """
    Count the number of duplicate values in all columns of a
DataFrame.

    Parameters:
    - df: The DataFrame to check for duplicates.

    Returns:
    - A dictionary where keys are column names and values are the
count of duplicates in each column.
    """
    duplicate_counts = {}
    for column in df.columns:
        duplicate_count = df[column].duplicated().sum()
        duplicate_counts[column] = duplicate_count
    return duplicate_counts

# Example usage:
# Assuming you have a DataFrame 'loan' and want to check duplicates in
all columns
# Replace 'loan' with your actual DataFrame name
duplicates_counts = count_duplicates_in_all_columns(loan)
print("Duplicate counts in each column:")
print(duplicates_counts)
```

```
Duplicate counts in each column:
{'id': 0, 'member_id': 0, 'loan_amnt': 38832, 'funded_amnt': 38676,
'funded_amnt_inv': 31512, 'term': 39715, 'int_rate': 39346, 'grade':
39710, 'sub_grade': 39682, 'emp_length': 39705, 'home_ownership':
39712, 'annual_inc': 34399, 'verification_status': 39714, 'issue_d':
39662, 'loan_status': 39714, 'purpose': 39703, 'addr_state': 39667,
'dti': 36849}
```

```
# Checking how many rows have null values
round(100*(loan.isnull().sum()/len(loan.index)), 2)
```

```
id                      0.00
member_id               0.00
loan_amnt               0.00
funded_amnt             0.00
funded_amnt_inv         0.00
term                    0.00
int_rate                0.00
grade                   0.00
sub_grade               0.00
emp_length              2.71
home_ownership          0.00
annual_inc              0.00
verification_status     0.00
issue_d                 0.00
loan_status             0.00
purpose                 0.00
addr_state              0.00
dti                     0.00
dtype: float64
```

```
# Remove the null rows from emp_length field as it is less than 5%
loan = loan.dropna(subset=['emp_length'])
```

```
print("-"*20,"Rows having NULL VALUES","-"*20)
round(100*(loan.isnull().sum()/len(loan.index)), 2)
```

```
-------------------- Rows having NULL VALUES --------------------
```

```
id                      0.0
member_id               0.0
loan_amnt               0.0
funded_amnt             0.0
funded_amnt_inv         0.0
term                    0.0
int_rate                0.0
grade                   0.0
sub_grade               0.0
emp_length              0.0
home_ownership          0.0
annual_inc              0.0
verification_status     0.0
issue_d                 0.0
loan_status             0.0
purpose                 0.0
addr_state              0.0
dti                     0.0
dtype: float64
```

```
# Checking the Purpose for which the loan has beek taken as it is a
part of Analysis in percentage form
loan.purpose.value_counts()*100/len(loan)
```

```
purpose
debt_consolidation    47.207701
credit_card           12.939289
other                  9.919259
home_improvement       7.450443
major_purchase         5.473319
small_business         4.614150
car                    3.874023
wedding                2.417059
medical                1.728689
moving                 1.446612
house                  0.952332
vacation               0.910926
educational            0.820351
renewable_energy       0.245846
Name: count, dtype: float64
```

```
# As we don't have information about the term 'Other' we will be
removing these values as they are not beneficial or reliable for
further Analysis
loan.drop(loan[loan.purpose == 'other'].index, inplace=True)

# Checking the Purpose column again
print((loan.purpose.value_counts()*100)/len(loan))
```

```
purpose
debt_consolidation    52.405987
credit_card           14.364101
home_improvement       8.270849
major_purchase         6.076015
small_business         5.122239
car                    4.300612
wedding                2.683214
medical                1.919044
moving                 1.605907
house                  1.057198
vacation               1.011233
educational            0.910684
renewable_energy       0.272918
Name: count, dtype: float64
```

```
loan['term']
# We will be converting the object data type of 'term' column into
'integer'
```

```
0         36 months
1         60 months
```

```
2          36 months
5          36 months
6          60 months
             ...
39711      36 months
39712      36 months
39713      36 months
39714      36 months
39716      36 months
Name: term, Length: 34809, dtype: object
```

```
# Converting the "term" column to int by removing the word 'months'
loan['term'] = loan['term'].str.replace('months', '')
loan['term'] = loan['term'].astype(int)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv  term
int_rate  \
0  1077501    1296599       5000         5000           4975.0    36
10.65%
1  1077430    1314167       2500         2500           2500.0    60
15.27%
2  1077175    1313524       2400         2400           2400.0    36
15.96%
5  1075269    1311441       5000         5000           5000.0    36
7.90%
6  1069639    1304742       7000         7000           7000.0    60
15.96%

  grade sub_grade emp_length home_ownership  annual_inc
verification_status  \
0     B        B2  10+ years           RENT     24000.0
Verified
1     C        C4   < 1 year           RENT     30000.0     Source
Verified
2     C        C5  10+ years           RENT     12252.0        Not
Verified
5     A        A4    3 years           RENT     36000.0     Source
Verified
6     C        C5    8 years           RENT     47004.0        Not
Verified

  issue_d  loan_status             purpose addr_state    dti
0  Dec-11   Fully Paid         credit_card         AZ  27.65
1  Dec-11  Charged Off                 car         GA   1.00
2  Dec-11   Fully Paid      small_business         IL   8.72
5  Dec-11   Fully Paid             wedding         AZ  11.20
6  Dec-11   Fully Paid  debt_consolidation         NC  23.51
```

```
loan['int_rate']
```

```
0        10.65%
1        15.27%
2        15.96%
5         7.90%
6        15.96%
          ...
39711     8.70%
39712     8.07%
39713    10.28%
39714     8.07%
39716    13.75%
Name: int_rate, Length: 34809, dtype: object
```

```
# We will Remove the '%' character in 'int_rate' and convert it to
data type 'float'
loan['int_rate'] = loan['int_rate'].str.replace('%', '')
loan['int_rate'] = loan['int_rate'].astype(float)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36

6  1069639    1304742       7000         7000           7000.0    60


   int_rate grade sub_grade emp_length home_ownership  annual_inc  \
0     10.65     B        B2  10+ years           RENT     24000.0
1     15.27     C        C4   < 1 year           RENT     30000.0
2     15.96     C        C5  10+ years           RENT     12252.0
5      7.90     A        A4    3 years           RENT     36000.0
6     15.96     C        C5    8 years           RENT     47004.0

  verification_status issue_d  loan_status             purpose
addr_state  \
0            Verified  Dec-11   Fully Paid         credit_card
AZ
1     Source Verified  Dec-11  Charged Off                 car
GA
2        Not Verified  Dec-11   Fully Paid      small_business
IL
5     Source Verified  Dec-11   Fully Paid             wedding
AZ
6        Not Verified  Dec-11   Fully Paid  debt_consolidation
```

```
NC
```

```
     dti
0  27.65
1   1.00
2   8.72
5  11.20
6  23.51
```

```
# We have to process the 'emp_length' column as it is important for
the Analysis
# We will be removing the 'string' part and only leaving the 'integer'
value
loan['emp_length']
```

```
0        10+ years
1         < 1 year
2        10+ years
5          3 years
6          8 years
           ...
39711      5 years
39712      4 years
39713      3 years
39714     < 1 year
39716     < 1 year
Name: emp_length, Length: 34809, dtype: object
```

```
# cleaning up the emp_length column and convert it to int
# will be treating '0' and '<1 year' as 0 as they all fall under less
than 1 year category
# will be treating '10+ years' as 10 which would mean 10 years and
above

loan['emp_length'] = loan['emp_length'].str.replace('years', '')
loan['emp_length'] = loan['emp_length'].str.replace('< 1 year', '0')
loan['emp_length'] = loan['emp_length'].str.strip('+ ')
loan['emp_length'] = loan['emp_length'].str.replace('year', '')
loan['emp_length'] = loan['emp_length'].astype(int)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36
```

```
6  1069639    1304742       7000         7000           7000.0    60


   int_rate grade sub_grade  emp_length home_ownership  annual_inc  \
0     10.65     B        B2          10           RENT     24000.0
1     15.27     C        C4           0           RENT     30000.0
2     15.96     C        C5          10           RENT     12252.0
5      7.90     A        A4           3           RENT     36000.0
6     15.96     C        C5           8           RENT     47004.0

  verification_status issue_d  loan_status             purpose
addr_state  \
0            Verified  Dec-11   Fully Paid         credit_card
AZ
1     Source Verified  Dec-11  Charged Off                 car
GA
2        Not Verified  Dec-11   Fully Paid      small_business
IL
5     Source Verified  Dec-11   Fully Paid             wedding
AZ
6        Not Verified  Dec-11   Fully Paid  debt_consolidation
NC

     dti
0  27.65
1   1.00
2   8.72
5  11.20
6  23.51
```

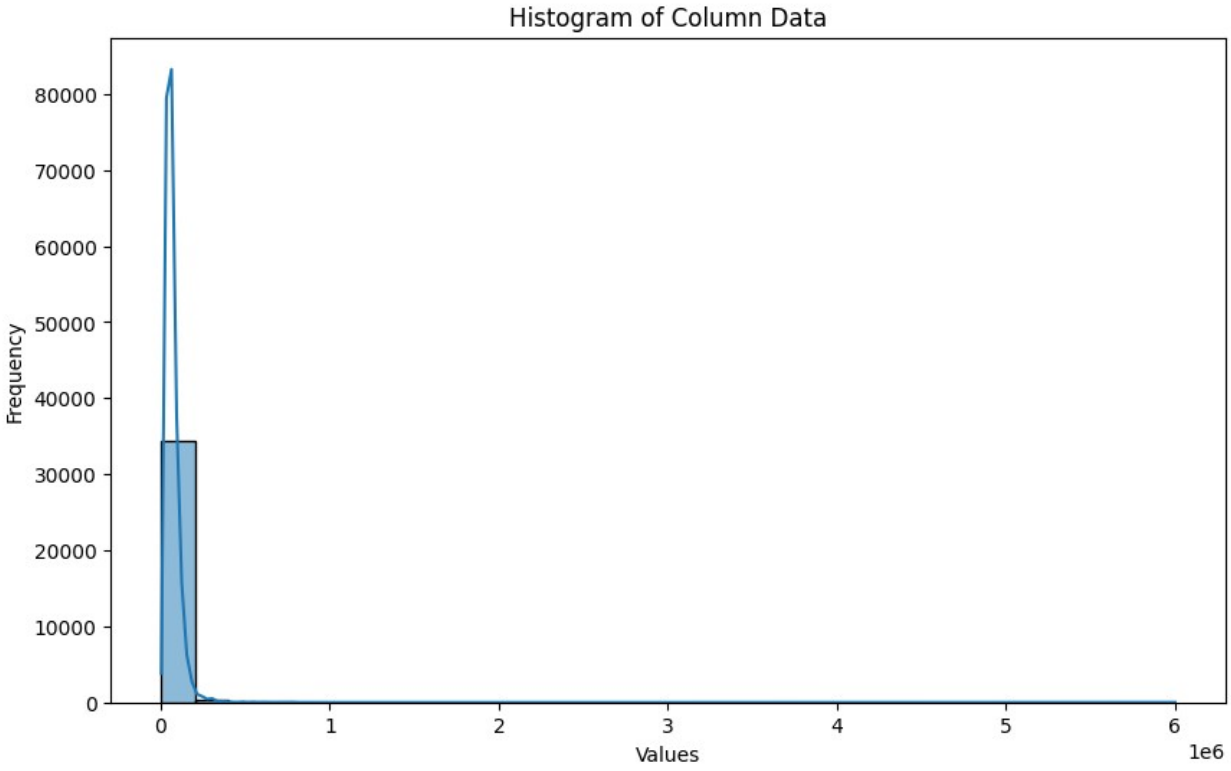
CHecking the **anual_inc** column to make sure it does not cause any problems during the Data Visualization

```
loan['annual_inc'].describe()
```

```
count    3.480900e+04
mean     7.022400e+04
std      6.550528e+04
min      4.000000e+03
25%      4.200000e+04
50%      6.000000e+04
75%      8.400000e+04
max      6.000000e+06
Name: annual_inc, dtype: float64
```

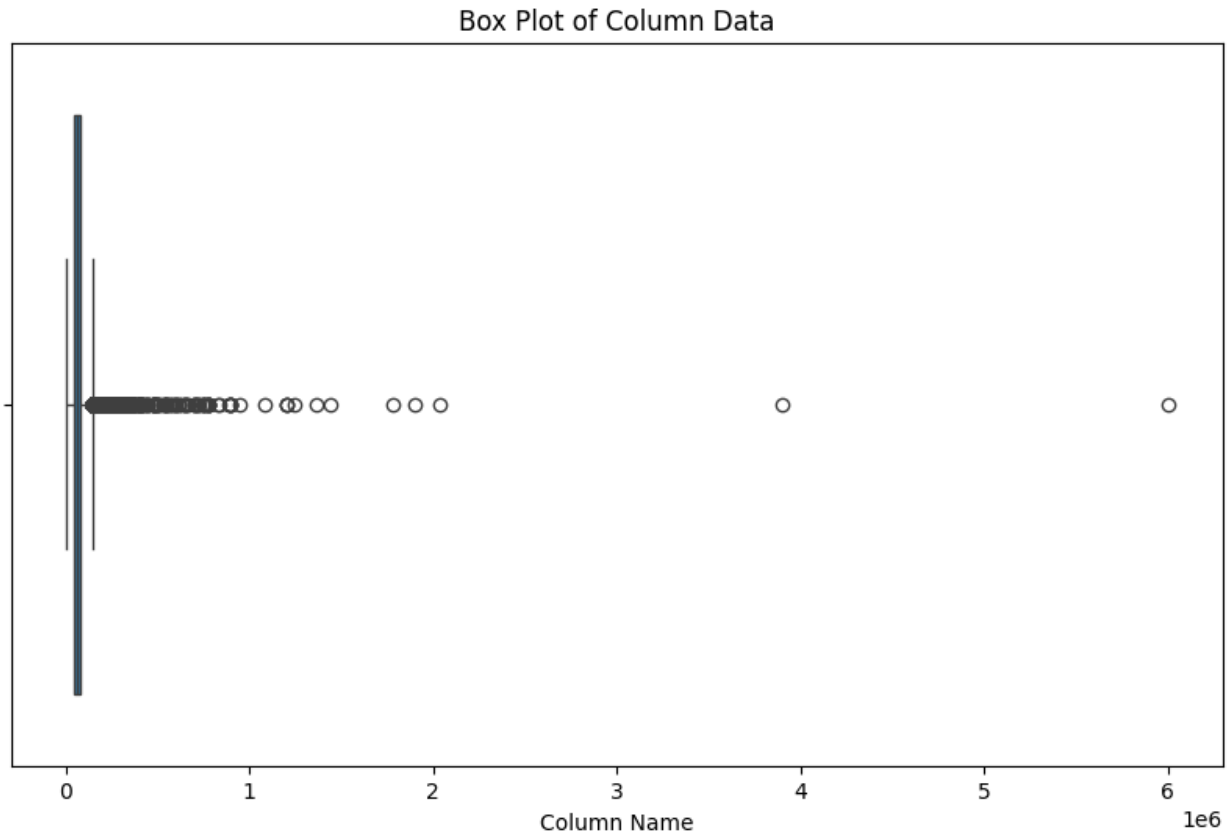
```
# Step 1: Visualize the Data
plt.figure(figsize=(10, 6))
sns.histplot(loan['annual_inc'], bins=30, kde=True)
plt.title('Histogram of Column Data')
plt.xlabel('Values')
```

```
plt.ylabel('Frequency')
plt.show()

# Step 2: Identify Outliers using the IQR method
Q1 = loan['annual_inc'].quantile(0.25)
Q3 = loan['annual_inc'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = loan[(loan['annual_inc'] < lower_bound) |
(loan['annual_inc'] > upper_bound)]
print("Identified Outliers:")
print(outliers)

# Visualize Outliers using a Box Plot
plt.figure(figsize=(10, 6))
sns.boxplot(x=loan['annual_inc'])
plt.title('Box Plot of Column Data')
plt.xlabel('Column Name')
plt.show()
```



```
Identified Outliers:
            id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
74     1068893    1303514      14400        14400          14400.0
36
80     1068994    1303432      35000        22075          22050.0
60
170    1067434    1301822      25000        25000          25000.0
36
185    1067084    1301459      35000        35000          35000.0
36
298    1065717    1299834       8000         8000           8000.0
36
...        ...        ...        ...          ...              ...   .
..
39676    88854      70699       5000         5000            225.0
36
39694    90966      90962       5000         5000           4150.0
36
39703    94838      73673       3000         3000           2550.0
36
39706    92676      92671       5000         5000            150.0
36
39707    92666      92661       5000         5000            525.0
36

       int_rate grade sub_grade  emp_length home_ownership  annual_inc
\
74         8.90     A        A5           0            OWN    150000.0

80        17.27     D        D3           3       MORTGAGE    150000.0

170       12.42     B        B4           2           RENT    225000.0

185       10.65     B        B2           2       MORTGAGE    168000.0

298       11.71     B        B3           1           RENT    150000.0

...         ...   ...       ...         ...            ...         ...

39676      7.43     A        A2           4           RENT    200000.0

39694      7.43     A        A2           8       MORTGAGE    150000.0

39703     10.28     C        C1           2       MORTGAGE    200000.0

39706      8.07     A        A4           0       MORTGAGE    180000.0

39707      9.33     B        B3           2       MORTGAGE    180000.0
```

```
      verification_status issue_d loan_status             purpose
addr_state  \
74         Source Verified  Dec-11  Fully Paid  debt_consolidation
NY
80                Verified  Dec-11  Fully Paid    home_improvement
NY
170               Verified  Dec-11  Fully Paid  debt_consolidation
NJ
185               Verified  Dec-11  Fully Paid  debt_consolidation
TX
298           Not Verified  Dec-11  Fully Paid         credit_card
NY
...                    ...     ...         ...                 ...
...
39676         Not Verified  Aug-07  Fully Paid               house
NY
39694         Not Verified  Jul-07  Fully Paid    home_improvement
GA
39703         Not Verified  Jul-07  Fully Paid    home_improvement
NY
39706         Not Verified  Jul-07  Fully Paid    home_improvement
WI
39707         Not Verified  Jul-07  Fully Paid    home_improvement
WI

          dti
74      14.85
80       7.51
170      8.32
185      3.17
298      2.48
...       ...
39676    0.28
39694    0.00
39703    0.00
39706    5.55
39707   11.93

[1640 rows x 18 columns]
```



There is a huge difference between the mean and the max values in the 'annual_inc' column. Therefore we need to remove the outliers

```
# We will remove any value which is outside the 99.9% quartile

nn_quartile = loan['annual_inc'].quantile(0.99)
loan = loan[loan["annual_inc"] < nn_quartile]
loan["annual_inc"].describe()
```

```
count     34460.000000
mean      66703.202944
std       35218.600503
min        4000.000000
25%       42000.000000
50%       60000.000000
75%       82642.500000
max      236004.000000
Name: annual_inc, dtype: float64
```

```
# Creating separate columns for 'issue_month' and 'issue_year'
loan[['issue_month', 'issue_year']] = loan['issue_d'].str.split('-',
expand=True)

# Adding '20' to the 'issue_year' column to make it complete year
```

```
loan['issue_year'] = '20' + loan['issue_year']

# Converting 'issue_year' to numeric data type as it is necessary for
the Data Visualization
loan['issue_year'] = pd.to_numeric(loan['issue_year'])

loan[['issue_month', 'issue_year']]
```

```
      issue_month  issue_year
0             Dec        2011
1             Dec        2011
2             Dec        2011
5             Dec        2011
6             Dec        2011
...           ...         ...
39711         Jul        2007
39712         Jul        2007
39713         Jul        2007
39714         Jul        2007
39716         Jun        2007

[34460 rows x 2 columns]
```

```
# create a new column for loan to 'income ratio'
loan['inc_ratio']= 100*(loan['loan_amnt']/loan['annual_inc'])
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36

6  1069639    1304742       7000         7000           7000.0    60


   int_rate grade sub_grade  emp_length  ... annual_inc
verification_status  \
0     10.65     B        B2          10  ...    24000.0
Verified
1     15.27     C        C4           0  ...    30000.0      Source
Verified
2     15.96     C        C5          10  ...    12252.0         Not
Verified
5      7.90     A        A4           3  ...    36000.0      Source
Verified
6     15.96     C        C5           8  ...    47004.0         Not
```

```
Verified

  issue_d  loan_status             purpose addr_state    dti
issue_month  \
0  Dec-11   Fully Paid         credit_card         AZ  27.65
Dec
1  Dec-11  Charged Off                 car         GA   1.00
Dec
2  Dec-11   Fully Paid      small_business         IL   8.72
Dec
5  Dec-11   Fully Paid             wedding         AZ  11.20
Dec
6  Dec-11   Fully Paid  debt_consolidation         NC  23.51
Dec

  issue_year  inc_ratio
0       2011  20.833333
1       2011   8.333333
2       2011  19.588639
5       2011  13.888889
6       2011  14.892350

[5 rows x 21 columns]
```

Categorizing the 'INCOME_RATIO' column

```
# For categorisation we need to get 25%, 50%, 75% quartiles of 'income
ratio' column
print('-'*20,'Quartiles','-'*20)
loan['inc_ratio'].quantile([.25, .5, .75])

-------------------- Quartiles --------------------

0.25    10.416667
0.50    16.949440
0.75    25.769231
Name: inc_ratio, dtype: float64
```

- Categorise the 'inc_ratio' column into 'categorised_inc_ratio' column:-
- < 10 is Low
- Between 10 and 16 (inclusive) is Medium
- 17 and 24 are High
- 25 is Very High

```
# Definig a function 'ratio_category' and using apply method

def ratio_category(n):
    if n < 10:
        return 'Low'
```

```
    elif n >=10 and n < 17:
        return 'Medium'
    elif n >= 17 and n < 25:
        return 'High'
    else:
        return 'Very High'

loan['categorized_inc_ratio'] =
loan['inc_ratio'].apply(ratio_category)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36

6  1069639    1304742       7000         7000           7000.0    60


   int_rate grade sub_grade  emp_length  ... verification_status
issue_d  \
0     10.65     B        B2          10  ...             Verified
Dec-11
1     15.27     C        C4           0  ...     Source Verified
Dec-11
2     15.96     C        C5          10  ...        Not Verified
Dec-11
5      7.90     A        A4           3  ...     Source Verified
Dec-11
6     15.96     C        C5           8  ...        Not Verified
Dec-11

   loan_status             purpose addr_state    dti issue_month
issue_year  \
0   Fully Paid         credit_card         AZ  27.65         Dec
2011
1  Charged Off                 car         GA   1.00         Dec
2011
2   Fully Paid      small_business         IL   8.72         Dec
2011
5   Fully Paid             wedding         AZ  11.20         Dec
2011
6   Fully Paid  debt_consolidation         NC  23.51         Dec
2011
```

```
   inc_ratio  categorized_inc_ratio
0  20.833333                  High
1   8.333333                   Low
2  19.588639                  High
5  13.888889                Medium
6  14.892350                Medium

[5 rows x 22 columns]
```

Categorizing the 'INTEREST_RATE' column

```
# Similarly for 'int_rate' column for categorisation of 25%, 50%, 75%
quartiles
print('-'*20,'Quartiles','-'*20)
loan['int_rate'].quantile([.25, .5, .75])
```

```
-------------------- Quartiles --------------------

0.25     9.32
0.50    11.86
0.75    14.61
Name: int_rate, dtype: float64
```

- Categorise the int_rate column into categorised_int_rate_perc column:-
- < 9% is Low
- Between 9% and 11% (both inclusive) is Medium
- 12% and 13% are High
- 14% is Very High

```
def interest_rates(n):
    if n < 9:
        return 'Low'
    elif n >=9 and n < 12:
        return 'Medium'
    elif n >= 12 and n < 14:
        return 'High'
    else:
        return 'Very High'

loan['categorised_int_rate_perc'] =
loan['int_rate'].apply(interest_rates)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0
36

1  1077430    1314167       2500         2500           2500.0
60

2  1077175    1313524       2400         2400           2400.0
36
```

```
5  1075269    1311441       5000          5000           5000.0    36

6  1069639    1304742       7000          7000           7000.0    60


   int_rate grade sub_grade  emp_length  ... issue_d  loan_status  \
0     10.65     B        B2          10  ...  Dec-11   Fully Paid
1     15.27     C        C4           0  ...  Dec-11  Charged Off
2     15.96     C        C5          10  ...  Dec-11   Fully Paid
5      7.90     A        A4           3  ...  Dec-11   Fully Paid
6     15.96     C        C5           8  ...  Dec-11   Fully Paid

               purpose addr_state    dti issue_month issue_year
inc_ratio  \
0          credit_card         AZ  27.65         Dec       2011
20.833333
1                  car         GA   1.00         Dec       2011
8.333333
2      small_business         IL   8.72         Dec       2011
19.588639
5              wedding         AZ  11.20         Dec       2011
13.888889
6  debt_consolidation         NC  23.51         Dec       2011
14.892350

  categorized_inc_ratio  categorised_int_rate_perc
0                  High                     Medium
1                   Low                  Very High
2                  High                  Very High
5                Medium                        Low
6                Medium                  Very High

[5 rows x 23 columns]
```

Categorizing the 'EMP_LENGTH' column

```
# For Categorisation getting the 25%, 50% and 75% quartiles of the
'emp_length' column
print('-'*20,'Quartile','-'*20)
loan['emp_length'].quantile([.25, .5, .75])

-------------------- Quartile --------------------

0.25    2.0
0.50    4.0
0.75    9.0
Name: emp_length, dtype: float64
```

- Categorise the emp_length column into categorised_emp_length column:-

- < 9 is Entry Level
- Between 2 and 4 (both inclusive) is Junior Level
- Between 4 and 8 (noth inclusive) is Middle Level
- 9 is Senior Level

```
# Defining the funcion 'emp_len_category' and using apply method

def emp_len_category(n):
    if n < 2:
        return 'Entry Level'
    elif n >=2 and n < 4:
        return 'Junior Level'
    elif n >= 4 and n < 9:
        return 'Middle Level'
    else:
        return 'Senior Level'

loan['categorised_emp_length'] =
loan['emp_length'].apply(emp_len_category)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0
36

1  1077430    1314167       2500         2500           2500.0
60

2  1077175    1313524       2400         2400           2400.0
36

5  1075269    1311441       5000         5000           5000.0
36

6  1069639    1304742       7000         7000           7000.0
60


   int_rate grade sub_grade  emp_length  ...  loan_status
purpose  \
0     10.65     B        B2          10  ...   Fully Paid
credit_card
1     15.27     C        C4           0  ...  Charged Off
car
2     15.96     C        C5          10  ...   Fully Paid
small_business
5      7.90     A        A4           3  ...   Fully Paid
wedding
6     15.96     C        C5           8  ...   Fully Paid
debt_consolidation

  addr_state    dti issue_month issue_year  inc_ratio
categorized_inc_ratio  \
0         AZ  27.65         Dec       2011  20.833333
High
```

```
1          GA   1.00          Dec       2011   8.333333
Low
2          IL   8.72          Dec       2011  19.588639
High
5          AZ  11.20          Dec       2011  13.888889
Medium
6          NC  23.51          Dec       2011  14.892350
Medium

  categorised_int_rate_perc  categorised_emp_length
0                    Medium            Senior Level
1                 Very High             Entry Level
2                 Very High            Senior Level
5                       Low            Junior Level
6                 Very High            Middle Level

[5 rows x 24 columns]
```

Categorizing the 'ANNUAL_INCOME' column

```
# For Categorisation getting the 25%, 50% and 75% quartiles of the
'annual_inc' column
print('-'*20,'Quartile','-'*20)
loan['annual_inc'].quantile([.25, .5, .75])
```

```
-------------------- Quartile --------------------

0.25    42000.0
0.50    60000.0
0.75    82642.5
Name: annual_inc, dtype: float64
```

- Categorise the annual_in column into categorised_annual_inc column:-
- < 41000 is low
- Between 41000 and 59000 (both inclusive) is Medium
- Between 60000 and 82000 (both inclusive) is High
- 83000 is Very High

```
# Defining the funcion 'ann_inc_cat' and using apply method

def ann_inc_cat(n):
    if n < 41000:
        return 'Low'
    elif n >=41000 and n < 60000:
        return 'Medium'
    elif n >= 60000 and n < 83000:
        return 'High'
    else:
        return 'Very High'
```

```
loan['categorised_annual_inc'] = loan['annual_inc'].apply(ann_inc_cat)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36

6  1069639    1304742       7000         7000           7000.0    60


   int_rate grade sub_grade  emp_length  ...             purpose
addr_state  \
0     10.65     B        B2          10  ...         credit_card
AZ
1     15.27     C        C4           0  ...                 car
GA
2     15.96     C        C5          10  ...      small_business
IL
5      7.90     A        A4           3  ...             wedding
AZ
6     15.96     C        C5           8  ...  debt_consolidation
NC

     dti issue_month issue_year  inc_ratio categorized_inc_ratio  \
0  27.65         Dec       2011  20.833333                 High
1   1.00         Dec       2011   8.333333                  Low
2   8.72         Dec       2011  19.588639                 High
5  11.20         Dec       2011  13.888889               Medium
6  23.51         Dec       2011  14.892350               Medium

   categorised_int_rate_perc categorised_emp_length
categorised_annual_inc
0                      Medium           Senior Level
Low
1                   Very High            Entry Level
Low
2                   Very High           Senior Level
Low
5                         Low           Junior Level
Low
6                   Very High           Middle Level
Medium
```

```
[5 rows x 25 columns]
```

Categorizing the 'DEBT-TO-INCOME-RATIO' column

```
# For categorization the 25%, 50% and 75% quartiles of the dti ( debt
to income ratio) column
print('-'*20,'Quartile','-'*20)
loan['dti'].quantile([.25, .5, .75])
```

```
-------------------- Quartile --------------------

0.25     8.46
0.50    13.58
0.75    18.70
Name: dti, dtype: float64
```

- Categorise the dti column into categorised_dti column:-
- < 8 is Low
- Between 8 and 12 (both inclusive) is Medium
- Between 13 and 18 (noth inclusive) is High
- 19 is Very High

```
# Defining the funcion 'dti_cat' and using apply method

def dti_cat(n):
    if n < 8:
        return 'Low'
    elif n >=8 and n < 13:
        return 'Medium'
    elif n >= 13 and n < 19:
        return 'High'
    else:
        return 'Very High'

loan['categorised_dti'] = loan['dti'].apply(dti_cat)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36

6  1069639    1304742       7000         7000           7000.0    60
```

```
   int_rate grade sub_grade  emp_length  ... addr_state    dti
issue_month  \
0     10.65     B        B2          10  ...         AZ  27.65
Dec
1     15.27     C        C4           0  ...         GA   1.00
Dec
2     15.96     C        C5          10  ...         IL   8.72
Dec
5      7.90     A        A4           3  ...         AZ  11.20
Dec
6     15.96     C        C5           8  ...         NC  23.51
Dec

  issue_year  inc_ratio categorized_inc_ratio
categorised_int_rate_perc  \
0       2011  20.833333                  High
Medium
1       2011   8.333333                   Low                  Very
High
2       2011  19.588639                  High                  Very
High
5       2011  13.888889                Medium
Low
6       2011  14.892350                Medium                  Very
High

   categorised_emp_length categorised_annual_inc  categorised_dti
0            Senior Level                    Low        Very High
1             Entry Level                    Low              Low
2            Senior Level                    Low           Medium
5            Junior Level                    Low           Medium
6            Middle Level                 Medium        Very High

[5 rows x 26 columns]
```

Categorizing the 'LOAN AMOUNT' column

```
# For categorization the 25%, 50% and 75% quartiles of the 'loan_amnt'
column
print('-'*20,'Quartile','-'*20)
loan['loan_amnt'].quantile([.25, .5, .75])
```

```
-------------------- Quartile --------------------

0.25     6000.0
0.50    10000.0
0.75    15000.0
Name: loan_amnt, dtype: float64
```

- Categorise the loan_amnt column into categorised_loan_amnt column:-
- < 5400 is Low
- Between 5400 and 9599 (both inclusive) is Medium
- Between 9600 and 14999 (noth inclusive) is High
- 15000 is Very High

```
# Defining the funcion 'loan_amnt_cat' and using apply method

def loan_amnt_cat(n):
    if n < 5400:
        return 'Low'
    elif n >=5400 and n < 9600:
        return 'Medium'
    elif n >= 9600 and n < 15000:
        return 'High'
    else:
        return 'Very High'

loan['categorised_loan_amnt'] = loan['loan_amnt'].apply(loan_amnt_cat)
loan.head()
```

```
        id  member_id  loan_amnt  funded_amnt  funded_amnt_inv
term  \
0  1077501    1296599       5000         5000           4975.0    36

1  1077430    1314167       2500         2500           2500.0    60

2  1077175    1313524       2400         2400           2400.0    36

5  1075269    1311441       5000         5000           5000.0    36

6  1069639    1304742       7000         7000           7000.0    60


   int_rate grade sub_grade  emp_length  ...    dti  issue_month
issue_year  \
0     10.65     B        B2          10  ...  27.65          Dec
2011
1     15.27     C        C4           0  ...   1.00          Dec
2011
2     15.96     C        C5          10  ...   8.72          Dec
2011
5      7.90     A        A4           3  ...  11.20          Dec
2011
6     15.96     C        C5           8  ...  23.51          Dec
2011

   inc_ratio categorized_inc_ratio categorised_int_rate_perc  \
0  20.833333                 High                    Medium
1   8.333333                  Low                 Very High
```

```
2  19.588639                   High                  Very High
5  13.888889                 Medium                        Low
6  14.892350                 Medium                  Very High

  categorised_emp_length  categorised_annual_inc categorised_dti  \
0           Senior Level                     Low       Very High
1            Entry Level                     Low             Low
2           Senior Level                     Low          Medium
5           Junior Level                     Low          Medium
6           Middle Level                  Medium       Very High

   categorised_loan_amnt
0                    Low
1                    Low
2                    Low
5                    Low
6                 Medium

[5 rows x 27 columns]
```

DATA VISUALIZATION

UNIVARIATE ANALYSIS

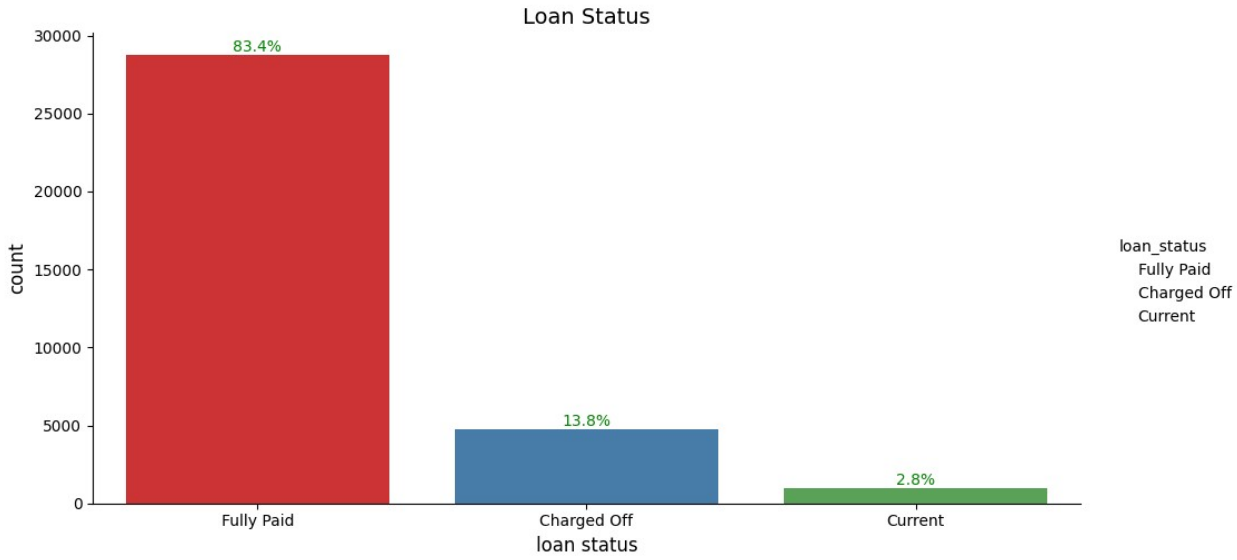
- Univariate analysis is the simplest form of data analysis. It involves analyzing a single variable in a dataset.
- The prefix **"uni"** means **"one"**. The purpose of univariate analysis is to understand the distribution of values for a single variable.
- It doesn't deal with causes or relationships.
- Univariate analysis can be either descriptive or inferential.
- It takes data, summarizes it, and finds patterns. **Here are some examples of univariate data:-**
- The salaries of workers in a specific industry
- The heights of ten students in a class
- The weight of 20 puppies

Univariate data requires analyzing each variable separately. Data is gathered to answers some question, or more specifically, for research question.

STATUS OF LOAN

```
# Plotting the distribution of loan status as this is one of the key
factors for this analysis

plot = sns.catplot(x="loan_status", kind="count", data=loan,
palette="Set1", aspect=2,legend=True)

plt.title('Loan Status', fontsize = 14)
```

```
plt.xlabel("loan status", fontsize = 12)
plt.ylabel("count", fontsize = 12)
plt.legend
# Print the counts

ax = plot.facet_axis(0,0)
for p in ax.patches:
    ax.annotate('{:1.1f}
%'.format((p.get_height()*100)/float(len(loan))), (p.get_x() +
p.get_width()/2., p.get_height()),
                color='green', ha='center', va='bottom')

plt.show()
```

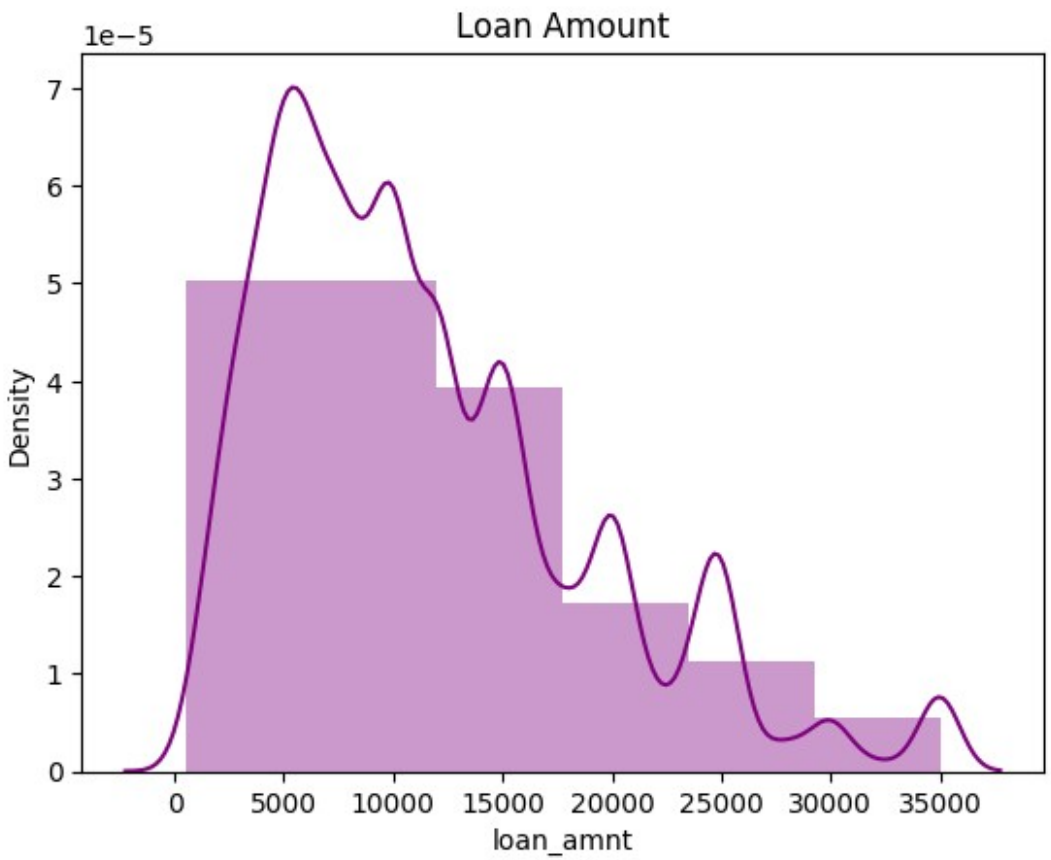


Here in this catplot we can see that there are

- **83.4%** people who have Fully Paid
- **13.8%** people who have defaulted or charged off
- **2.8%** people are currently paying the loan

AMOUNT OF LOAN

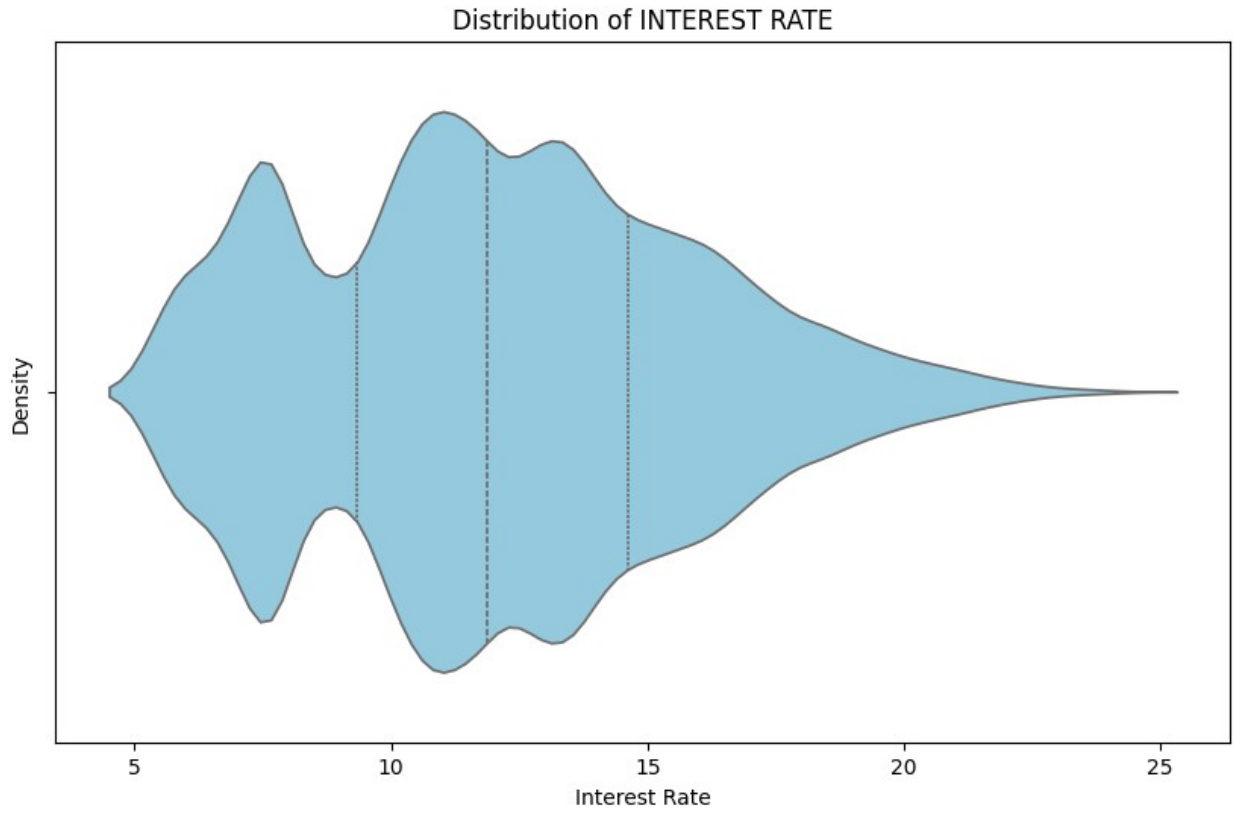
```
# Plotting th distribution of loan amount

plt.title('Loan Amount')
sns.distplot(loan['loan_amnt'],color='purple', bins=6,kde=True)
plt.show()
```



The Distplot Graph above shows that distribution is between **5000** - **20000** approximately

INTEREST RATE

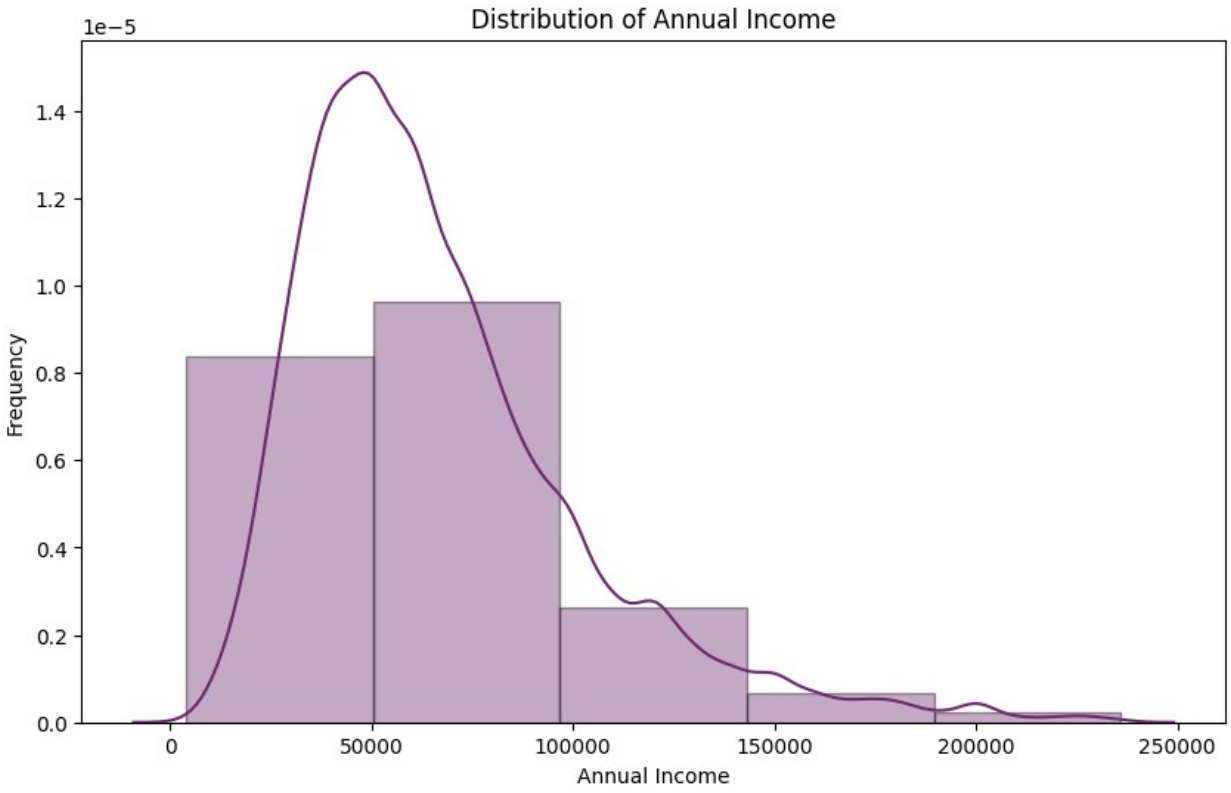
```
plt.figure(figsize=(10, 6))

# Plot the violin plot
sns.violinplot(x=loan['int_rate'], inner='quartile', color='skyblue')

# Add a title and labels
plt.title("Distribution of INTEREST RATE")
plt.xlabel("Interest Rate")
plt.ylabel("Density")

plt.show()
```



- The above violinplot graph shows that INTEREST RATE is massively spread between **8%** - **14%** approximately

ANNUAL INCOME

```
colors = sns.color_palette("flare")

# Create a figure and axis
plt.figure(figsize=(10, 6))

# Plot the distribution using a specified color and binwidth
sns.distplot(loan['annual_inc'], bins=5, color=colors[5], kde=True,
hist_kws={'edgecolor': 'black'}, hist=True)

# Add a title and labels
plt.title('Distribution of Annual Income')
plt.xlabel('Annual Income')
plt.ylabel('Frequency')

plt.show()
```



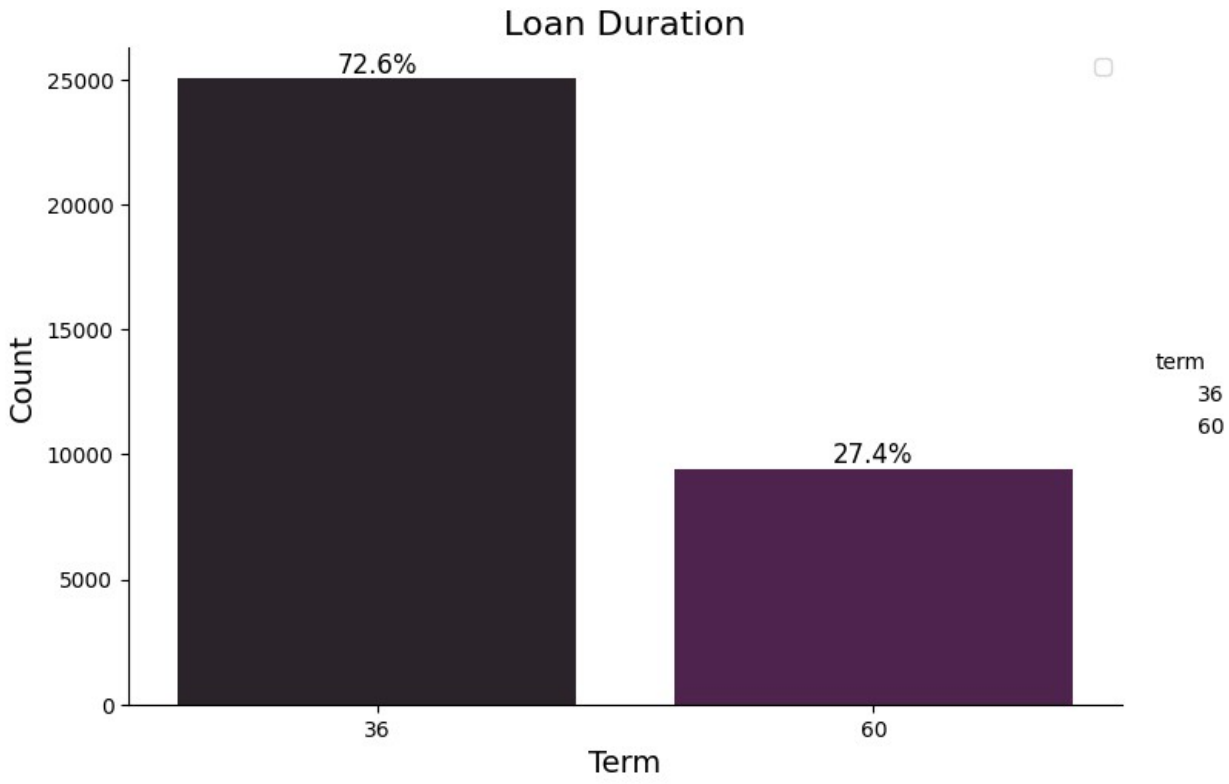
- According to the above Distplot graph, majority of the applicants have annual income between **40000** to **90000** approximately

DURATION OF LOAN

```
colors = sns.color_palette("dark:magenta")

# Create a figure and axis
plt.figure(figsize=(8, 5))

# Plot the loan duration counts with a darker color palette
plot = sns.catplot(x="term", kind="count", data=loan, palette=colors,
aspect=1.5)

# Set title and labels
plt.title('Loan Duration', fontsize=16)
plt.xlabel("Term", fontsize=14)
plt.ylabel("Count", fontsize=14)
plt.legend()

# Print the counts on the plot
ax = plot.facet_axis(0, 0)
for p in ax.patches:
    ax.annotate('{:1.1f}%'.format((p.get_height() * 100) /
float(len(loan))),
                (p.get_x() + p.get_width() / 2., p.get_height()),
```

```
                color='black', fontsize=12, ha='center', va='bottom')

plt.show()
```

```
No artists with labels found to put in legend.  Note that artists
whose label start with an underscore are ignored when legend() is
called with no argument.

<Figure size 800x500 with 0 Axes>
```

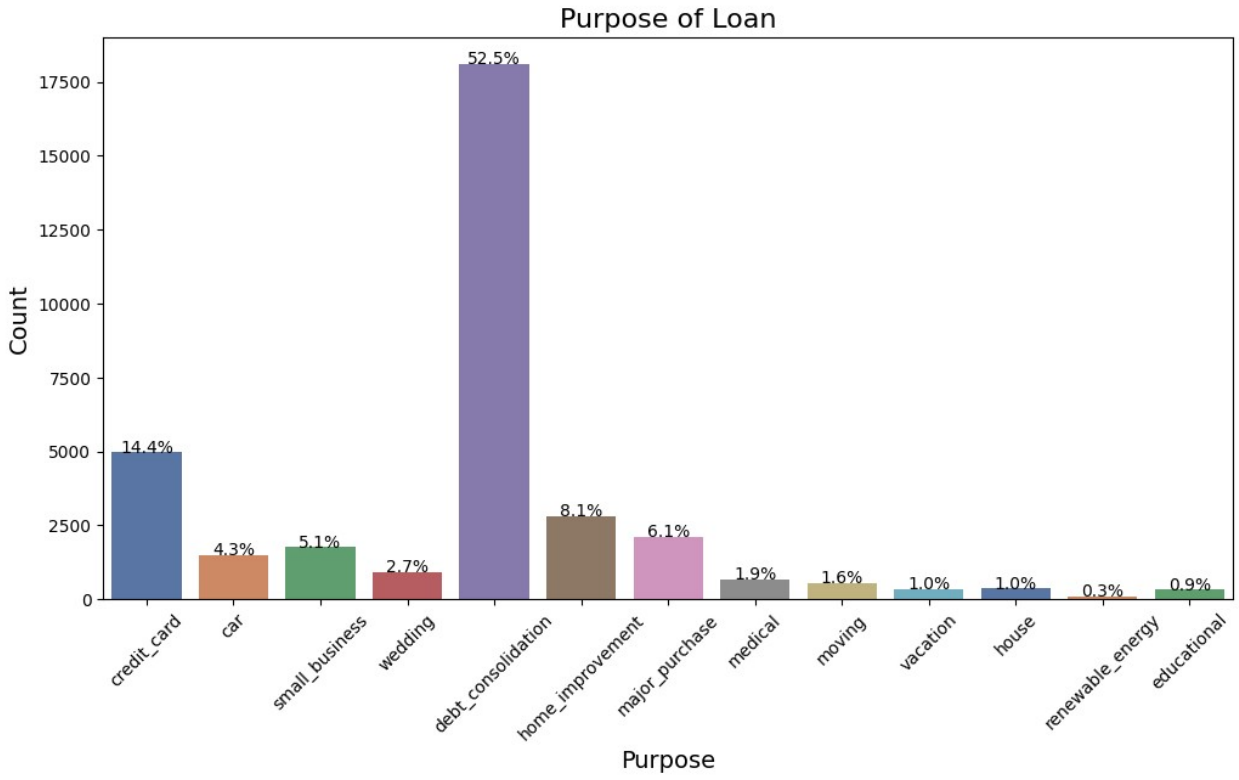


- According to the above Distplot graph we know that large no. of applicants have taken loan duration as **36 months (72.6)**

PURPOSE OF LOAN

```
plt.figure(figsize=(12, 6))

# Plot the purposes for which applicants have applied for loans with a
custom color palette
plot = sns.countplot(x="purpose", data=loan, palette="deep")

# Set title and labels
plt.title('Purpose of Loan', fontsize=16)
plt.xlabel("Purpose", fontsize=14)
plt.ylabel("Count", fontsize=14)
```

```
plt.xticks(rotation=45)

# Print the counts on the plot
for p in plot.patches:
    height = p.get_height()
    plot.text(p.get_x() + p.get_width() / 2., height + 2, '{:1.1f}
%'.format((height * 100) / len(loan)), ha="center")

plt.show()
```

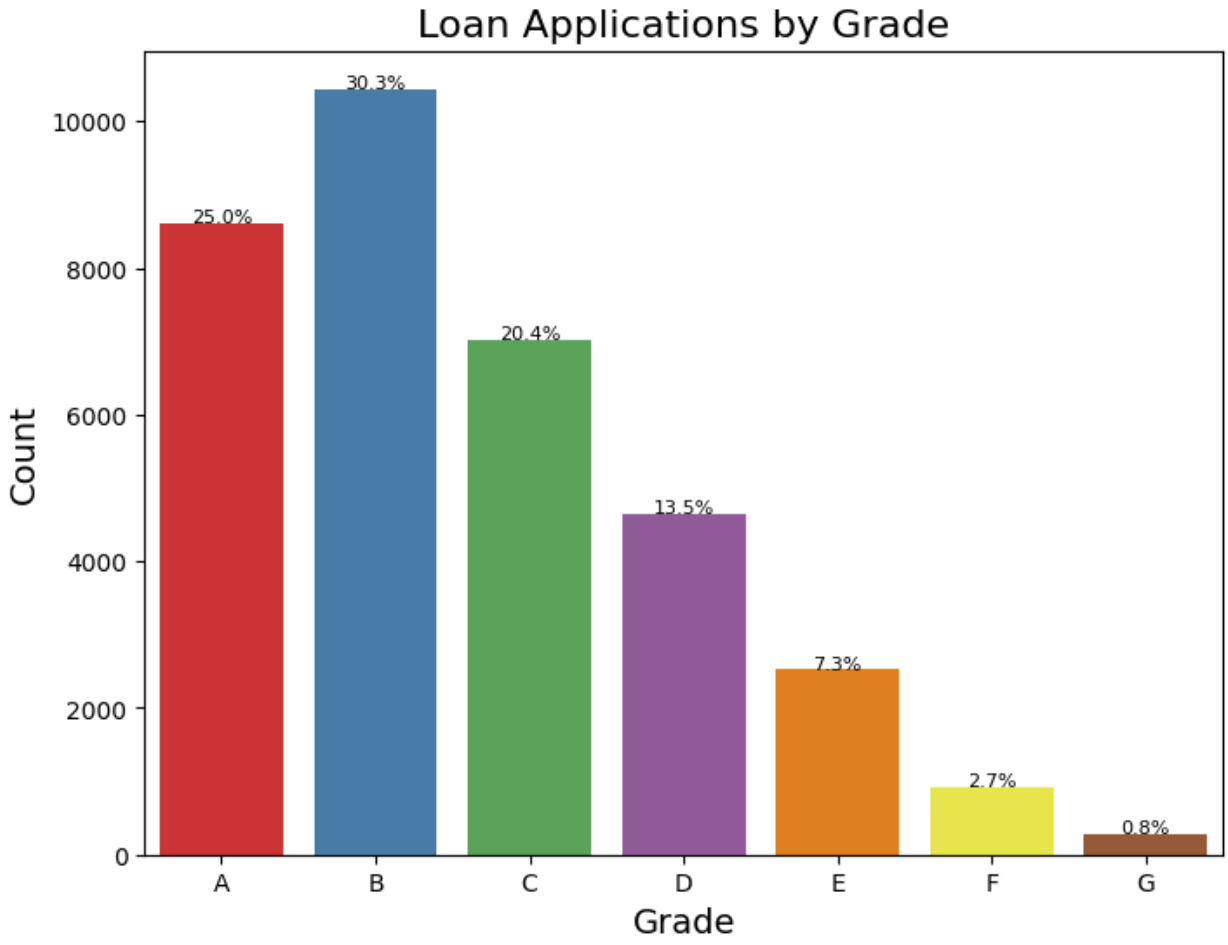


- According to the above graph following are the purposes for which more than 5% applicants have taken loan:-
  - debt_consolidation - 52.5%
  - credit_card - 14.4%
  - home_improvement - 8.1%
  - major_purchase - 6.1%
  - small business - 5.1%

LOAN APPLICATIONS BY GRADE

```
plt.figure(figsize=(8, 6))

# Plot the distribution of loan applications by grade
sns.countplot(x="grade", data=loan, palette="Set1",
```

```
order=sorted(loan['grade'].unique()))

# Set title and labels
plt.title('Loan Applications by Grade', fontsize=16)
plt.xlabel("Grade", fontsize=14)
plt.ylabel("Count", fontsize=14)

plt.xticks(rotation=0)

total = len(loan)  # Total number of loan applications
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=8)
plt.show()
```

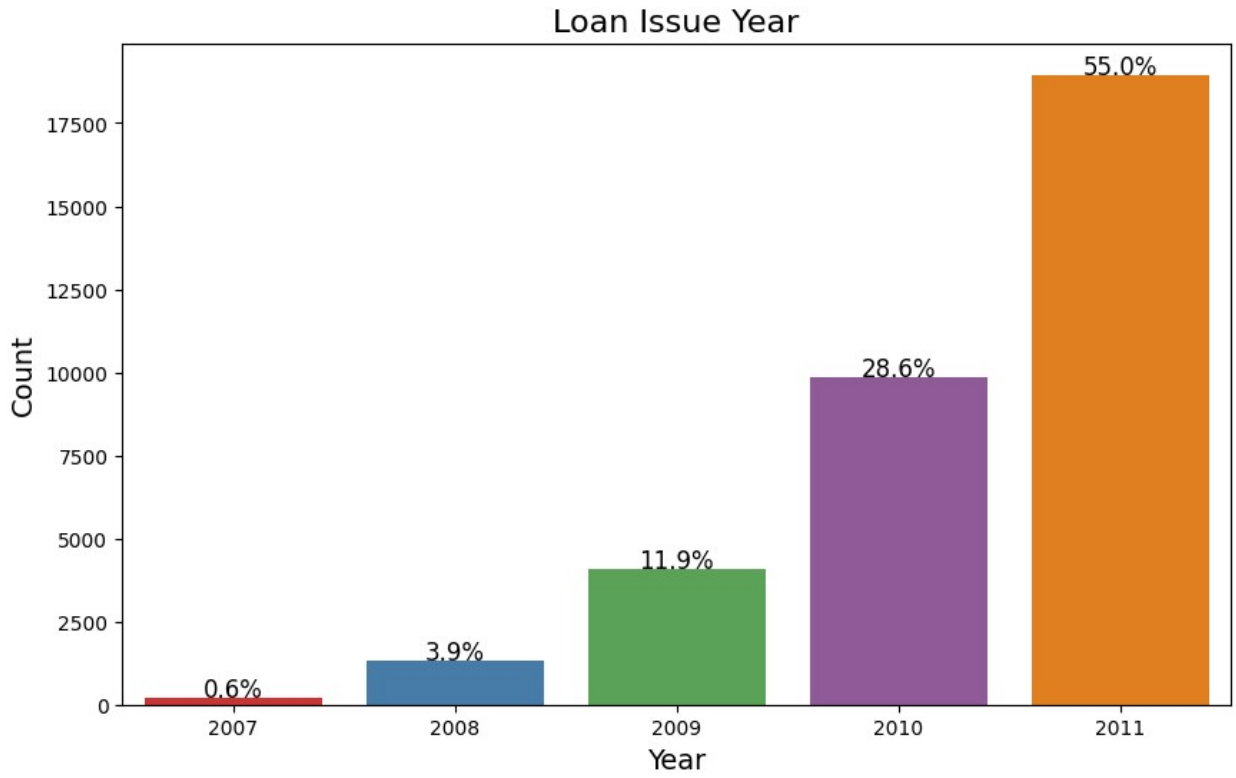


- According to the above Countplot graph shows that large no.of applicants fall under the grade:-

- B (30.3%)
- A (25%)
- C (20.4%)

LOAN ISSUE YEAR

```
plt.figure(figsize=(10, 6))

# Plot the distribution of loan issue years
plot = sns.countplot(x="issue_year", data=loan, palette="Set1")

# Set title and labels
plt.title('Loan Issue Year', fontsize=16)
plt.xlabel("Year", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of loan applications
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=12)

plt.show()
```



- According to the above graph, applicants for loan increased as the years increased. In 2011 the no. of applicants for loan was 53.7%. Since the variable **issue year** does not provide us any direction in the analysis we will not use this variable for any further analysis.

Segmented Univariate Analysis

- It is one of the simplest form of visualization to analyze data. In its name 'Uni' means one which itself describes that it considers only a single data variable for analysis. analysis. Segmented analysis here means that the data variable is analyzed in subsets and is very useful as it can show the change metric in pattern across the different segments of the same variable.

- It looks at one variable at a time for example the data shown below in the frequency can be used for univariate analysis. This example is of a quantity sales of different products at a medical store.

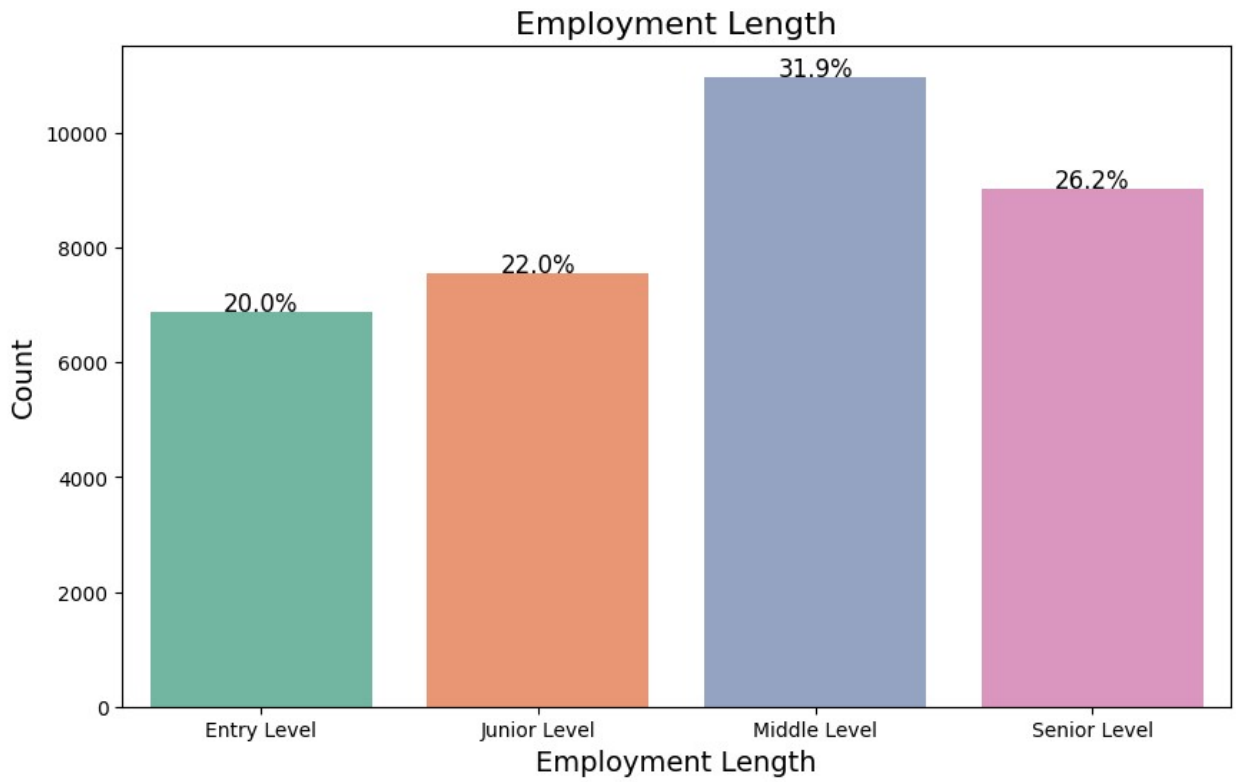
Application:-

- Segmented Univariate analysis can be used to find summary of a single data variable in form of segments. The dataset variable is divided into subsets and patterns can be observed across the segments. The central tendencies such as mean, mode, and median; maximum and minimum; range; variance and standard deviation are also detected. Visualized charts of various kind are used to show the description.

```
# Various order categories will be frequently used for ordering

order_category = ['Low','Medium','High','Very High']
order_emp_category = ['Entry Level', 'Junior Level', 'Middle Level',
'Senior Level']
order_grade = ['A','B','C','D','E','F','G']
filtered_purpose_df = loan[(loan['purpose']=='debt_consolidation')
                           |(loan['purpose']=='credit_card')
                           |(loan['purpose']=='home_improvement')
                           |(loan['purpose']=='major_purchase')
                           |(loan['purpose']=='small_business')]
```

EMPLOYMENT LENGTH

```
plt.figure(figsize=(10, 6))

# Plot the distribution of employment length
plot = sns.countplot(x="categorised_emp_length", data=loan,
palette="Set2", order=order_emp_category)

# Set title and labels
plt.title('Employment Length', fontsize=16)
plt.xlabel("Employment Length", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=12)

plt.show()
```

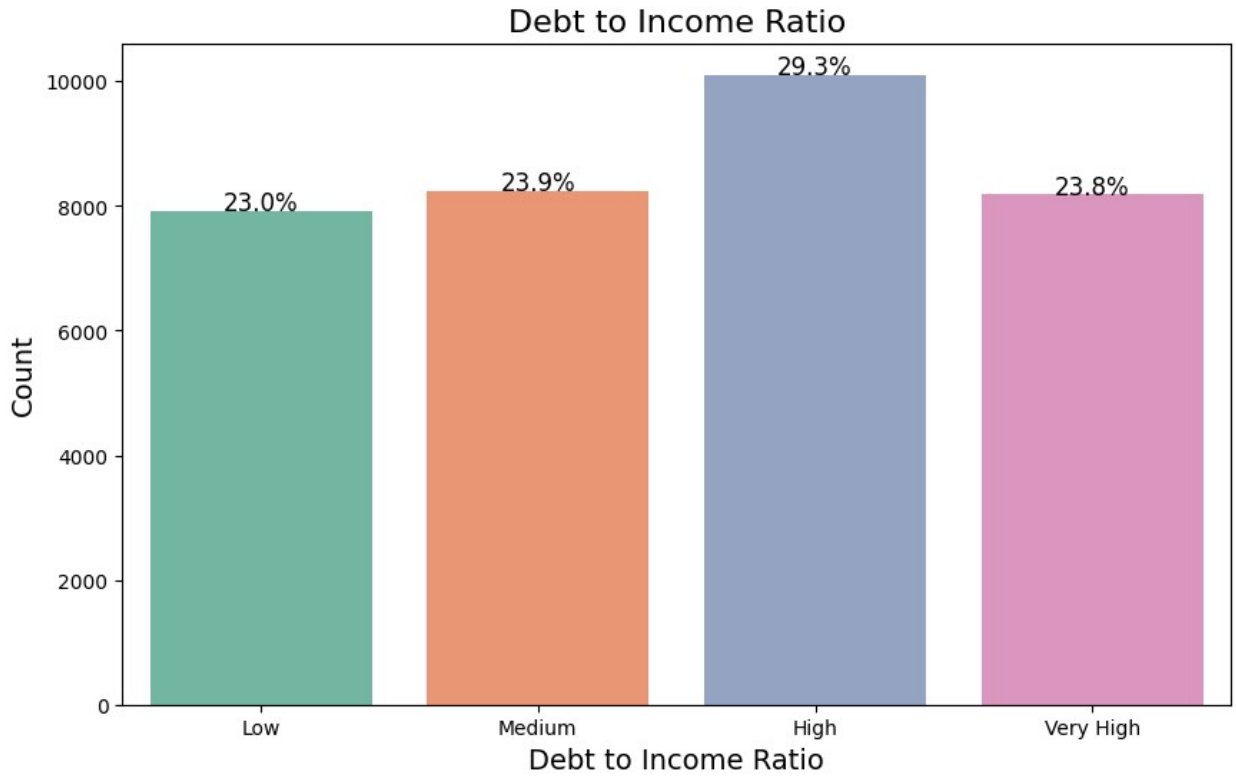


- According to the above graph there are more number of loan applicants belonging to the **Middle level** category **(31.9%)**

DEBT TO INCOME RATIO

```
plt.figure(figsize=(10, 6))

# Plot the distribution of debt to income ratio
plot = sns.countplot(x="categorised_dti", data=loan, palette="Set2",
order=order_category)

# Set title and labels
plt.title('Debt to Income Ratio', fontsize=16)
plt.xlabel("Debt to Income Ratio", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=12)
```

```
plt.show()
```

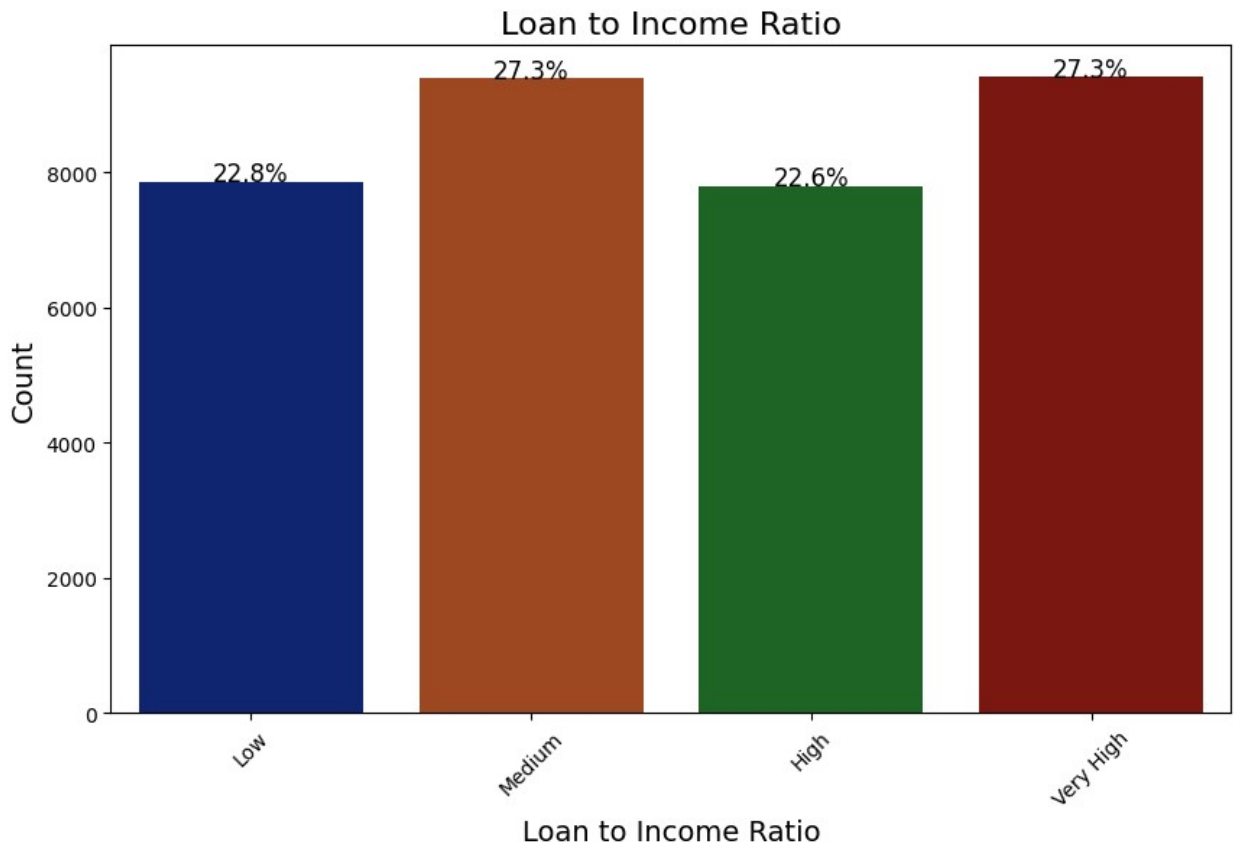


- According to the above graph majority of applicants have **High** Debt to Income Ratio **29.3%** approximately.

LOAN TO INCOME RATIO

```
plt.figure(figsize=(10, 6))

# Plot the distribution of loan to income ratio
plot = sns.countplot(x="categorized_inc_ratio", data=loan,
palette="dark", order=order_category)

# Set title and labels
plt.title('Loan to Income Ratio', fontsize=16)
plt.xlabel("Loan to Income Ratio", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
```

```
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=12)

plt.show()
```



- According to the above graph majority lies in 2 columns **MEDIUM (27.3%)** & **VERY HIGH (27.3%)** approximately

Bivariate analysis

- It is a statistical method that examines the relationship between two variables.
- The goal of bivariate analysis is to determine if there is a statistical link between the two variables, and if so, how strong and in which direction that link is.
- Bivariate analysis is one of the simplest forms of statistical analysis.
- It usually involves the variables X and Y. The variables are called a dependent variable and an independent variable.

- The dependent variable depends on the independent variable, whose value changes in specific relation to the independent variable.
- Bivariate analysis can be helpful in testing simple hypotheses of association.

We will now perform Bivariate Analysis by comparing the variable **loan_status** with other parameters.

** FIrst we need to obtain the correlation among all the variables of numeric values in the loan dataset**

loan_correlation = loan.corr() loan_correlation

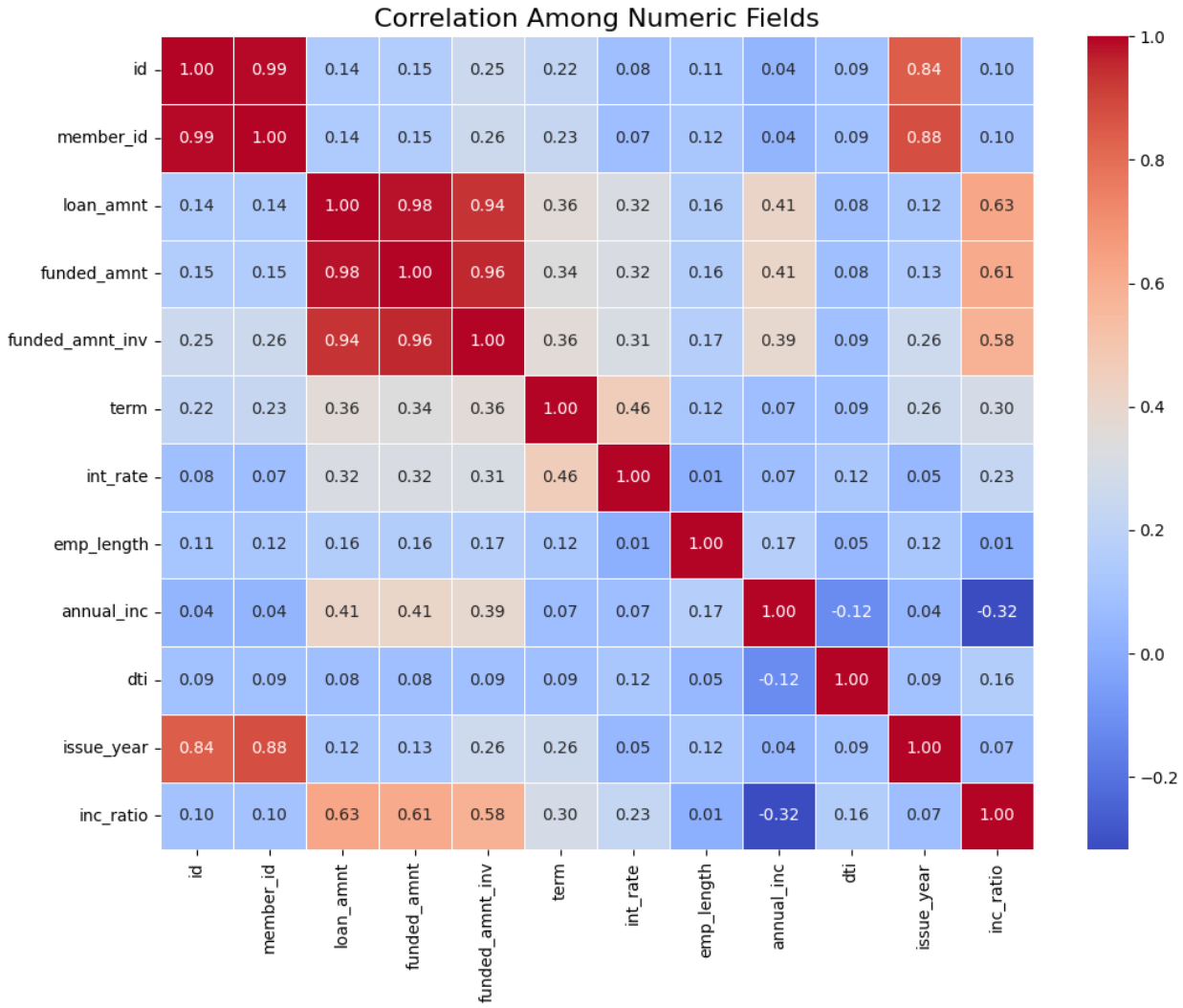
- Runnig the above code will give us error because the **corr()** function in **Pandas** calculates the correlation between numeric columns, but our "**grade**" column is categorical with string values (e.g., 'A', 'B', etc.). Hence it cannot be included in the correlation calculation without preprocessing.
- To calculate the correlation among numeric fields and exclude non-numeric fields like "**grade**", we have to first select only the numeric columns for the correlation calculation.

```
# Select only numeric columns from the DataFrame
numeric_columns = loan.select_dtypes(include=['number'])

# Calculate the correlation among numeric columns
loan_correlation = numeric_columns.corr()

loan_correlation
```

```
                        id  member_id  loan_amnt  funded_amnt
funded_amnt_inv  \
id                1.000000   0.993542   0.143595     0.153696
0.253620
member_id         0.993542   1.000000   0.142919     0.152139
0.262586
loan_amnt         0.143595   0.142919   1.000000     0.980731
0.937708
funded_amnt       0.153696   0.152139   0.980731     1.000000
0.956684
funded_amnt_inv   0.253620   0.262586   0.937708     0.956684
1.000000
term              0.217068   0.233356   0.364130     0.342525
0.363156
int_rate          0.079010   0.074303   0.316505     0.320330
0.313472
emp_length        0.114408   0.119154   0.160349     0.159423
0.170542
annual_inc        0.035899   0.036906   0.413471     0.408175
```

```
0.390833
dti               0.090836   0.091683   0.079573     0.079224
0.085866
issue_year        0.844337   0.880656   0.123429     0.133472
0.264951
inc_ratio         0.097500   0.095600   0.626012     0.613845
0.583041

                      term  int_rate  emp_length  annual_inc       dti
\
id                0.217068  0.079010    0.114408    0.035899  0.090836

member_id         0.233356  0.074303    0.119154    0.036906  0.091683

loan_amnt         0.364130  0.316505    0.160349    0.413471  0.079573

funded_amnt       0.342525  0.320330    0.159423    0.408175  0.079224

funded_amnt_inv   0.363156  0.313472    0.170542    0.390833  0.085866

term              1.000000  0.455487    0.121488    0.074122  0.087028

int_rate          0.455487  1.000000    0.012320    0.073791  0.117236

emp_length        0.121488  0.012320    1.000000    0.170615  0.051418

annual_inc        0.074122  0.073791    0.170615    1.000000 -0.120289

dti               0.087028  0.117236    0.051418   -0.120289  1.000000

issue_year        0.263152  0.052615    0.119488    0.038099  0.091822

inc_ratio         0.295181  0.228478    0.012210   -0.318029  0.156569


                  issue_year  inc_ratio
id                  0.844337   0.097500
member_id           0.880656   0.095600
loan_amnt           0.123429   0.626012
funded_amnt         0.133472   0.613845
funded_amnt_inv     0.264951   0.583041
term                0.263152   0.295181
int_rate            0.052615   0.228478
emp_length          0.119488   0.012210
annual_inc          0.038099  -0.318029
dti                 0.091822   0.156569
issue_year          1.000000   0.073515
inc_ratio           0.073515   1.000000

plt.figure(figsize=(12, 9))
```

```
# Generate the correlation heatmap
sns.heatmap(loan_correlation, annot=True, cmap='coolwarm',
linewidths=0.5, fmt=".2f")

# Set the title and adjust the fontsize
plt.title('Correlation Among Numeric Fields', fontsize=16)

plt.show()
```



- According to the above **HEATMAP** we can say that:-
  - **loan_amount**
  - **funded_amount**
  - **funded_amount_inv**

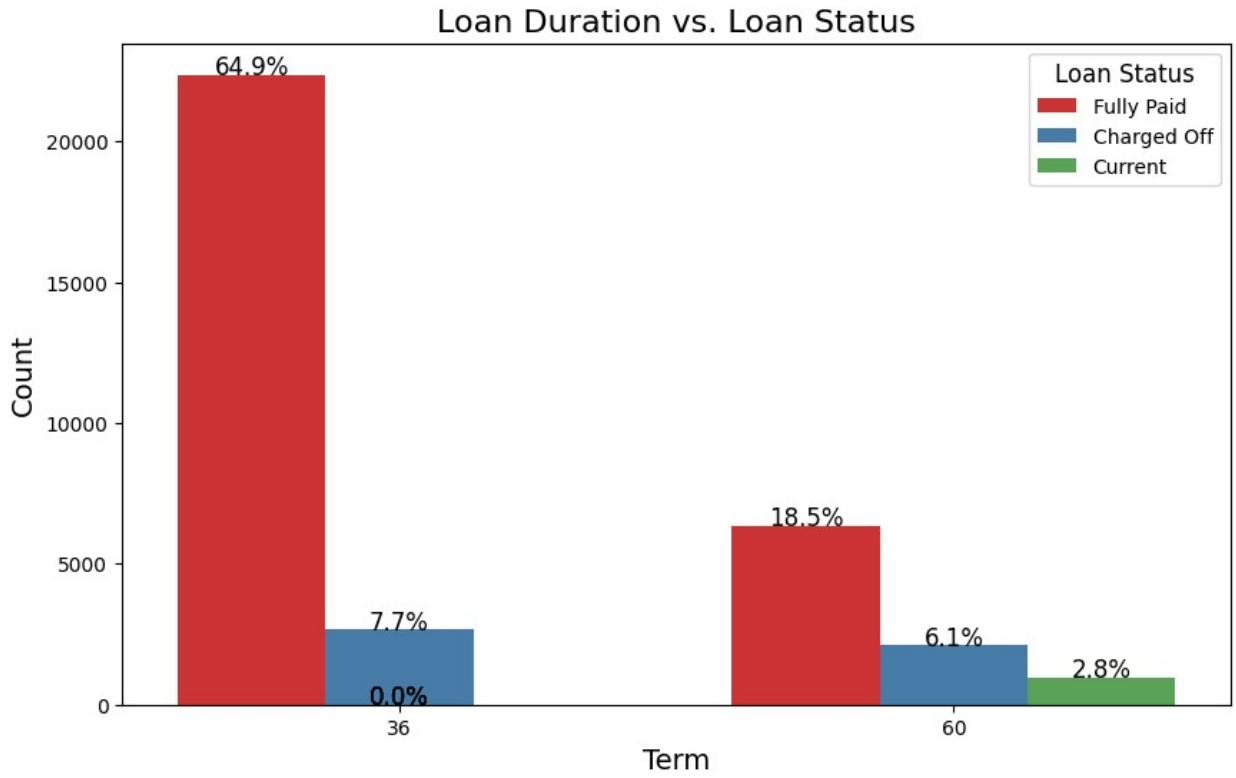
are very closely correlated. Hence we can safely take any one of the fields from the above 3 fields for our analysis.

Hence we will be using the **loan_amount** field for the further analysis

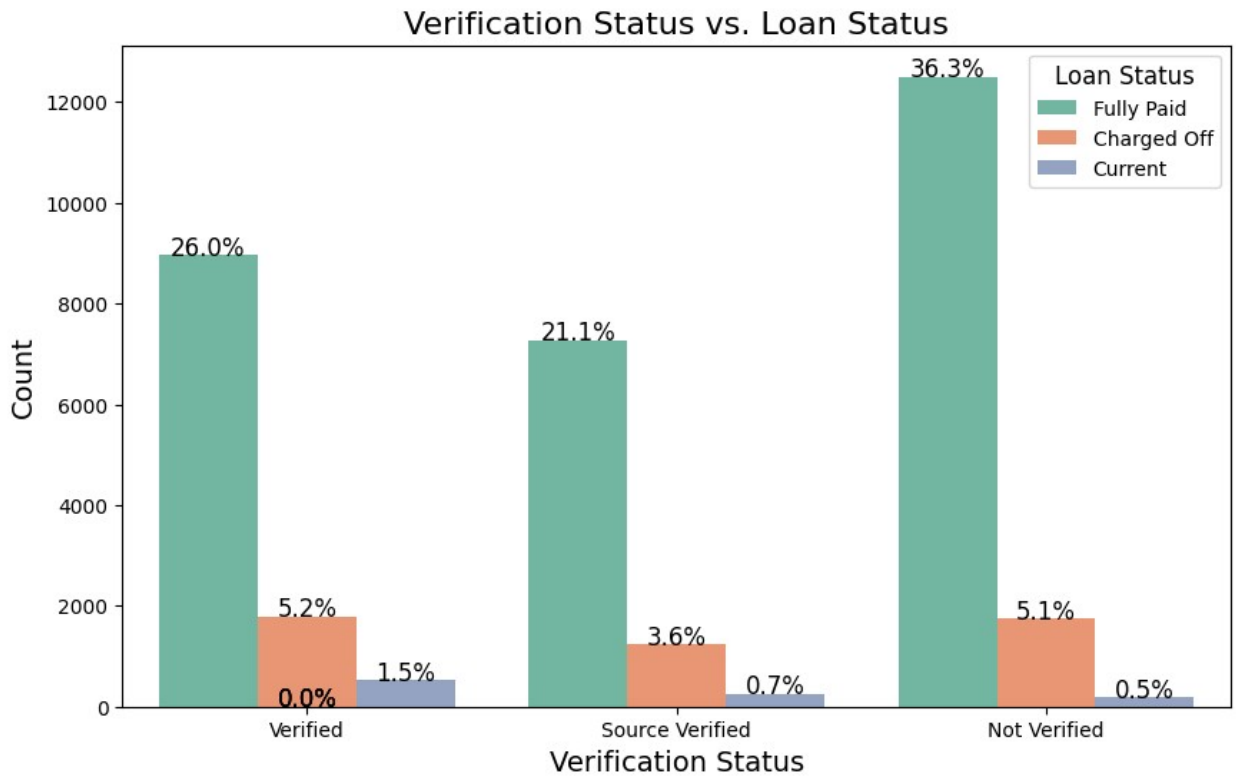
ANALYSIS OF 2 COLUMNS

LOAN DURATION V/S LOAN STATUS

```
plt.figure(figsize=(10, 6))

# Plot the Loan Duration vs. Loan Status with a hue
plot = sns.countplot(x="term", hue="loan_status", data=loan,
palette="Set1")

# Set title and labels
plt.title('Loan Duration vs. Loan Status', fontsize=16)
plt.xlabel("Term", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=12)

plt.show()
```




```
plt.figure(figsize=(10, 6))

# Plot the Verification Status vs. Loan Status with a hue
plot = sns.countplot(x="verification_status", hue="loan_status",
data=loan, palette="Set2")

# Set title and labels
plt.title('Verification Status vs. Loan Status', fontsize=16)
plt.xlabel("Verification Status", fontsize=14)
plt.ylabel("Count", fontsize=14)

plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height + 5),
                ha='center', fontsize=12)

plt.show()
```

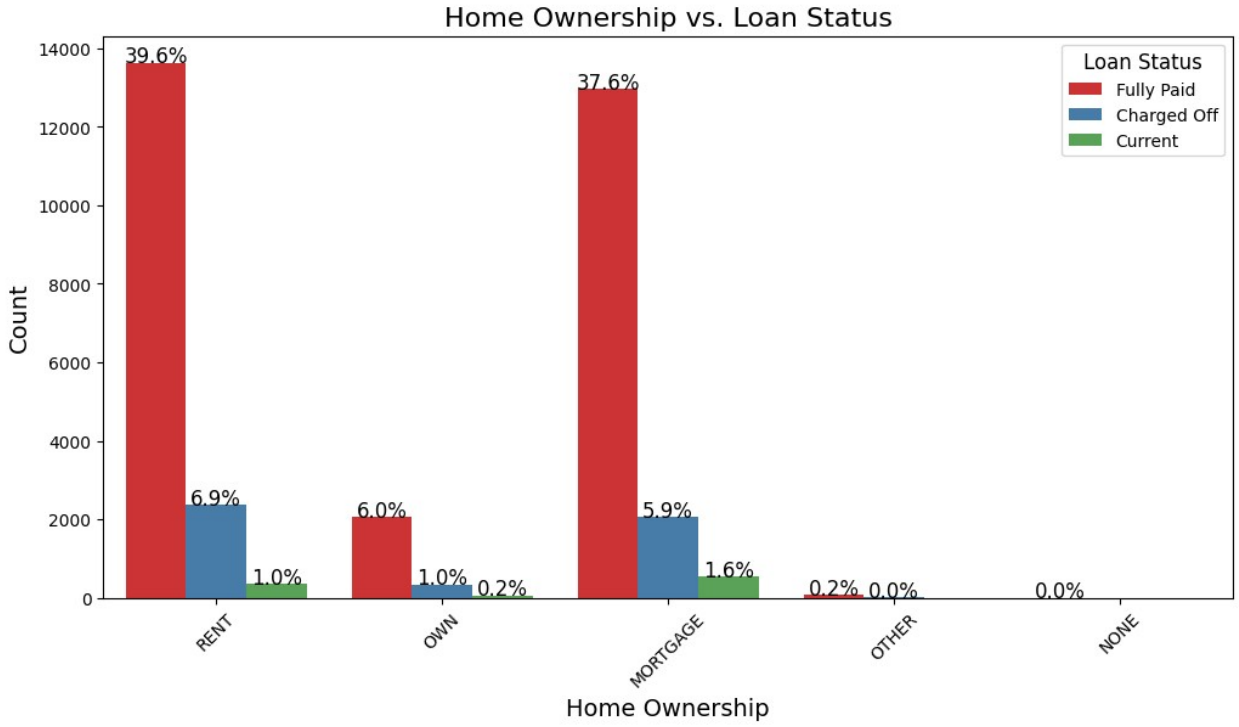


- According to this graph we can see that people who have **Verified** source of income are more **Defaulted** in numbers. Then it's not important for further analysis.

HOME OWNERSHIP V/S LOAN STATUS

```
plt.figure(figsize=(12, 6))

# Plot the Home Ownership vs. Loan Status with a hue
plot = sns.countplot(x="home_ownership", hue="loan_status", data=loan,
palette="Set1")

# Set title and labels
plt.title('Home Ownership vs. Loan Status', fontsize=16)
plt.xlabel("Home Ownership", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=45)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
```

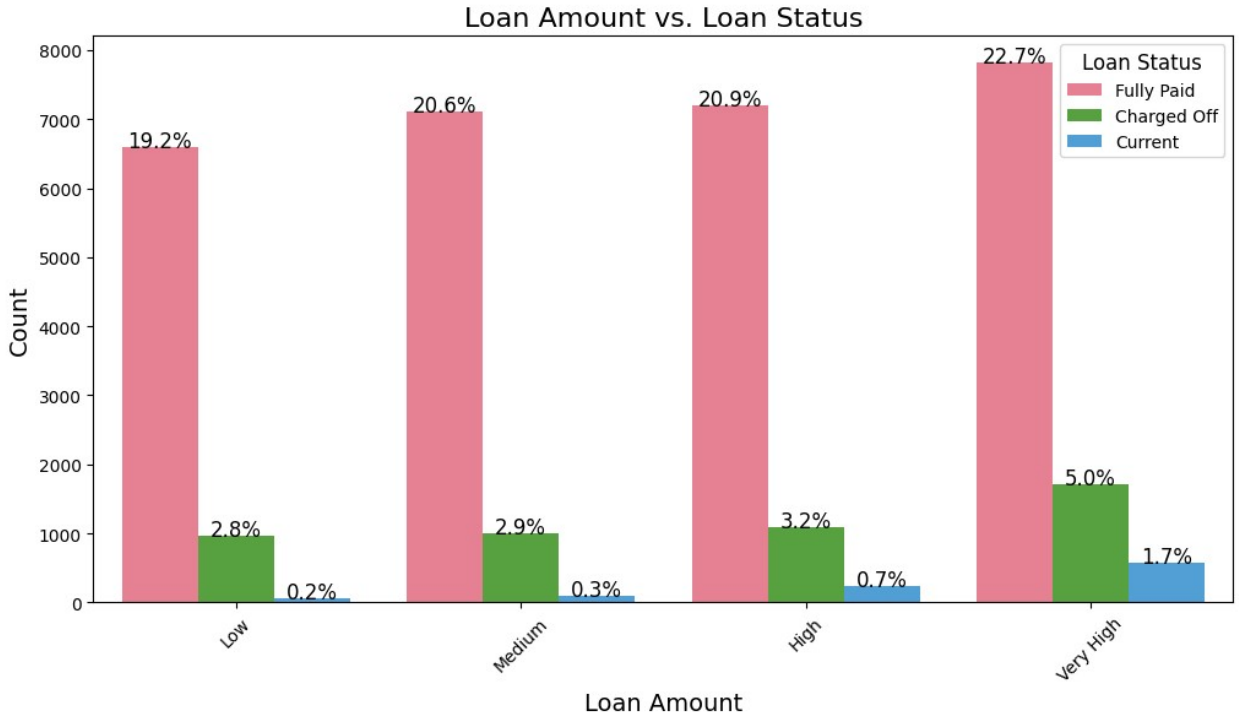
```
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```



LOAN AMOUNT V/S LOAN STATUS

```
plt.figure(figsize=(12, 6))

# Plot the Loan Amount vs. Loan Status with a hue
plot = sns.countplot(x="categorised_loan_amnt", hue="loan_status",
data=loan, palette="husl", order=order_category)

# Set title and labels
plt.title('Loan Amount vs. Loan Status', fontsize=16)
plt.xlabel("Loan Amount", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

# Rotate x-axis labels for better readability
```

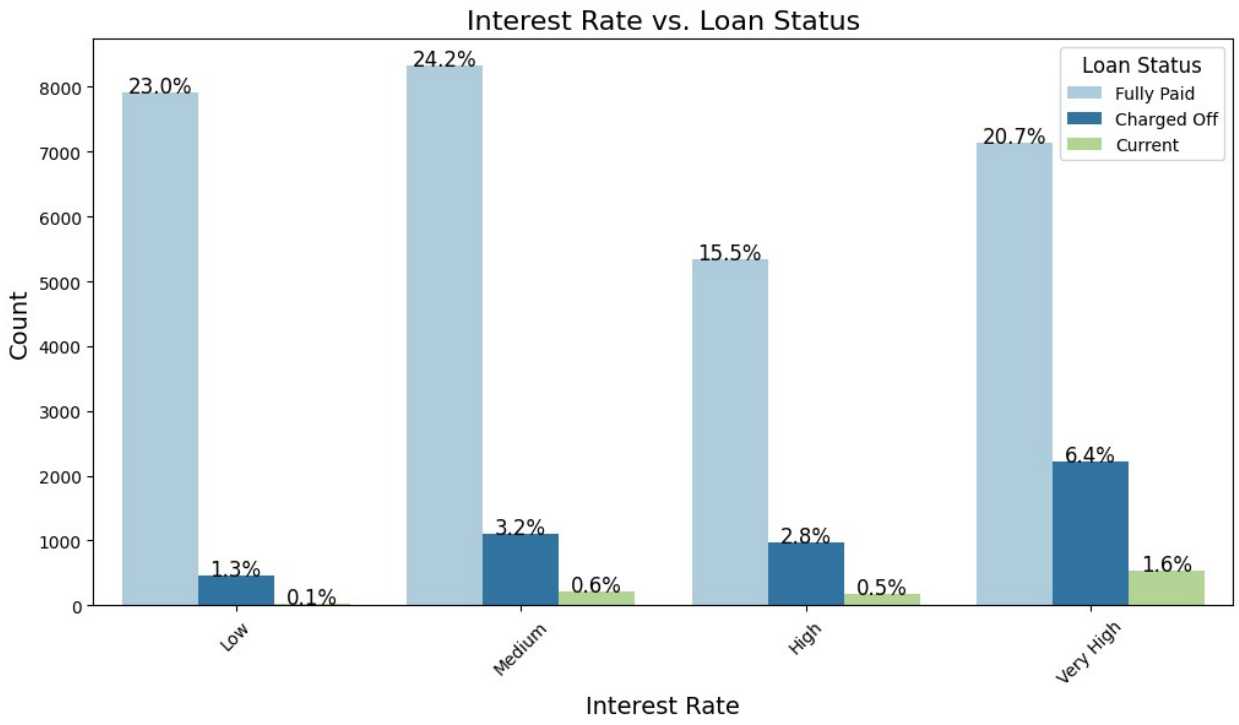
```
plt.xticks(rotation=45)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)
plt.show()
```



INTEREST RATE V/S LOAN STATUS

```
plt.figure(figsize=(12, 6))

# Plot the Interest Rate vs. Loan Status with a hue
plot = sns.countplot(x="categorised_int_rate_perc", hue="loan_status",
data=loan, palette="Paired", order=order_category)

# Set title and labels
plt.title('Interest Rate vs. Loan Status', fontsize=16)
plt.xlabel("Interest Rate", fontsize=14)
plt.ylabel("Count", fontsize=14)
```

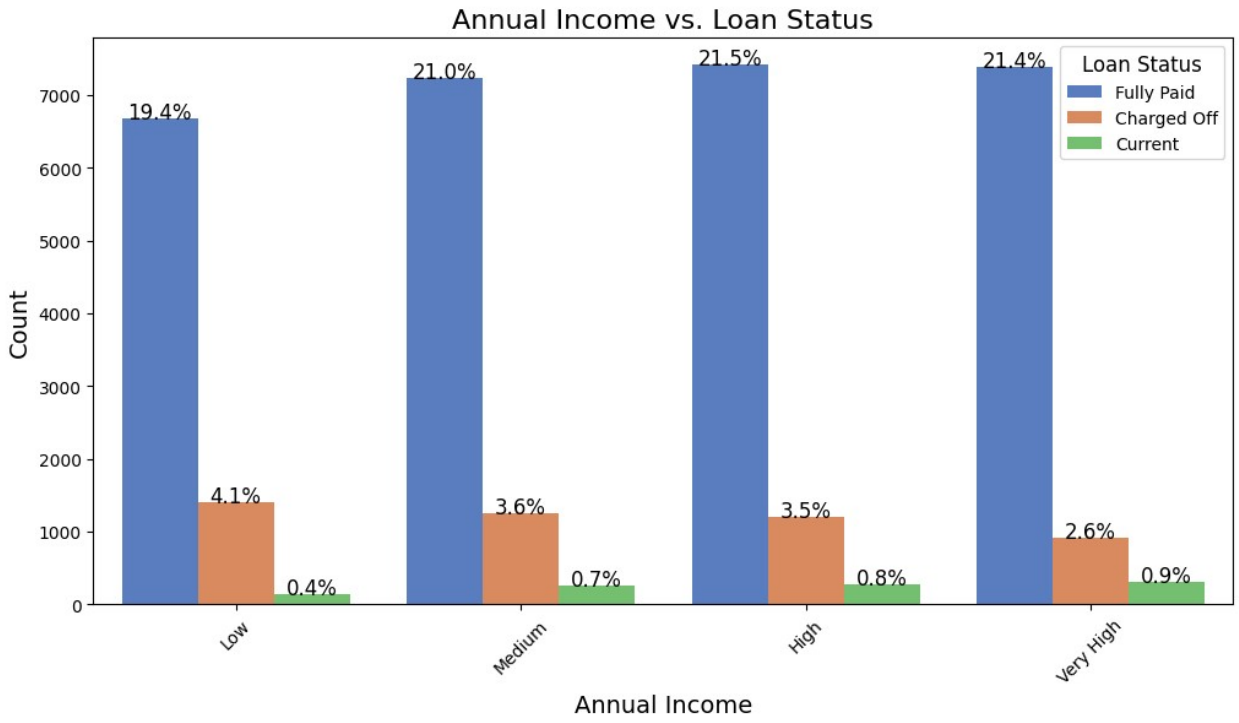
```
# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=45)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```



ANNUAL INCOME V/S LOAN STATUS

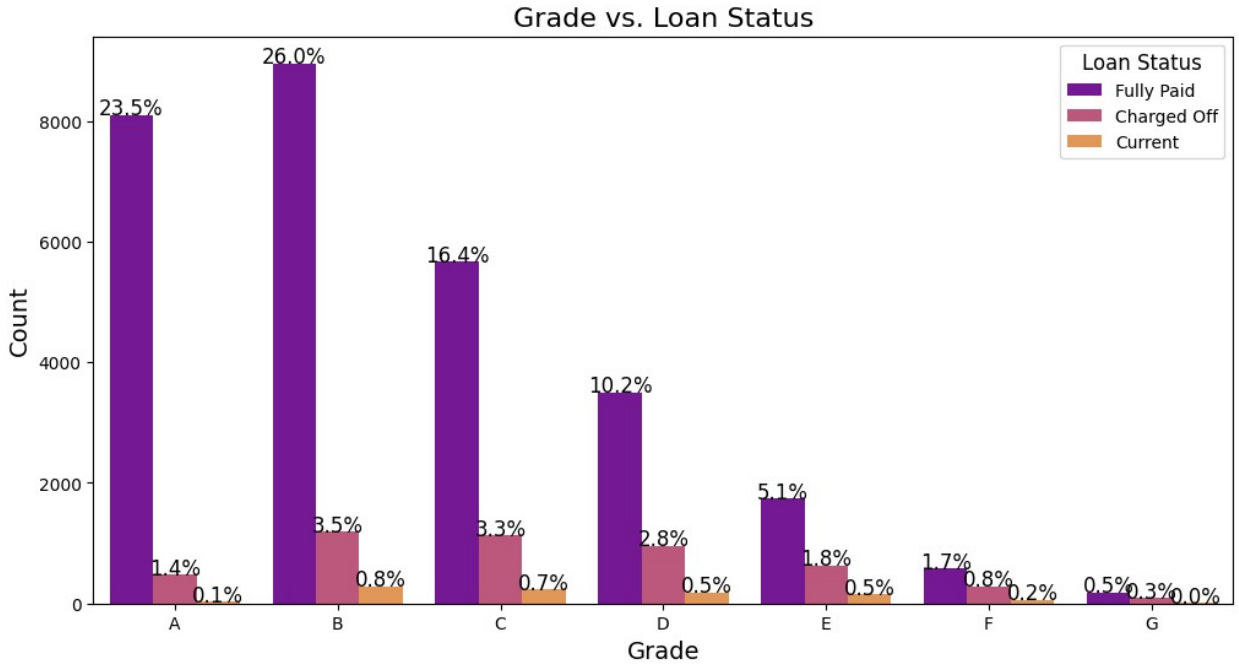
```
plt.figure(figsize=(12, 6))

# Plot the Annual Income vs. Loan Status with a hue
plot = sns.countplot(x="categorised_annual_inc", hue="loan_status",
data=loan, palette="muted", order=order_category)
```

```
# Set title and labels
plt.title('Annual Income vs. Loan Status', fontsize=16)
plt.xlabel("Annual Income", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=45)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```



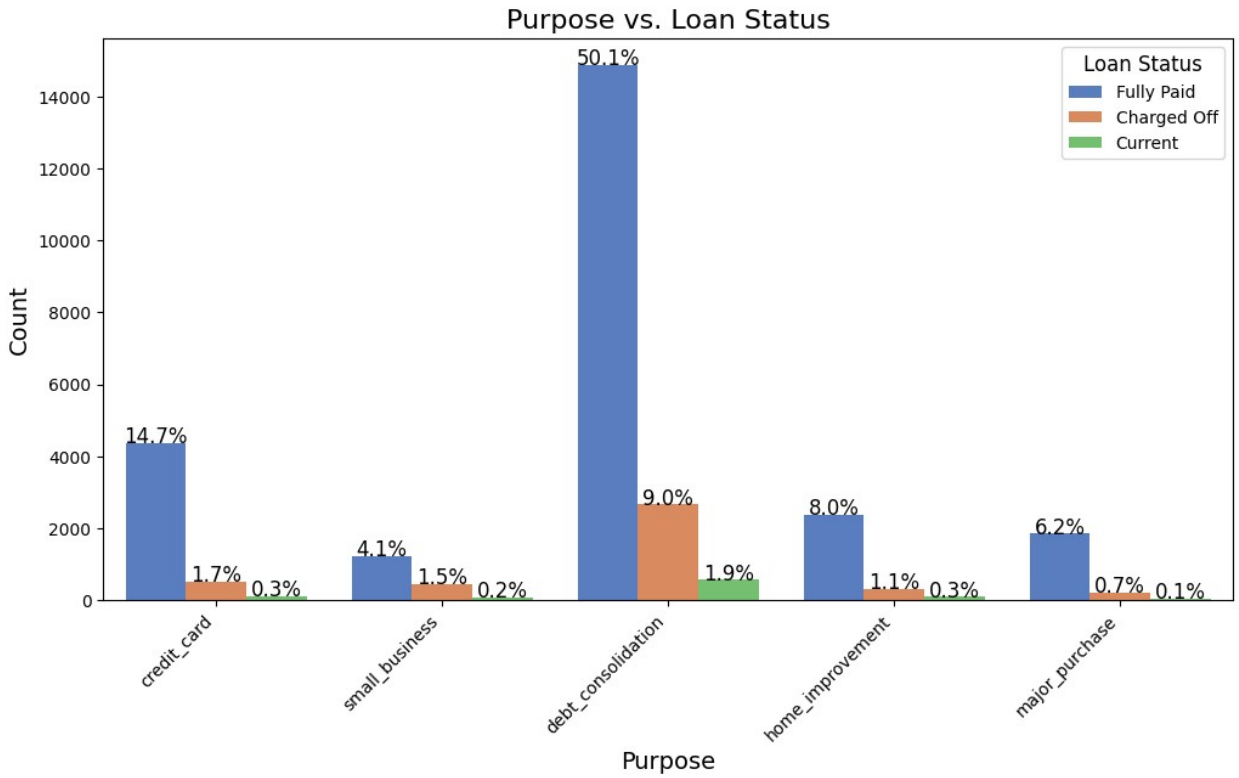
GRADE V/S LOAN STATUS

```
plt.figure(figsize=(12, 6))

# Plot the Grade vs. Loan Status with a hue
plot = sns.countplot(x="grade", hue="loan_status", data=loan,
palette="plasma", order=order_grade)

# Set title and labels
plt.title('Grade vs. Loan Status', fontsize=16)
plt.xlabel("Grade", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=0)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)
plt.show()
```



PURPOSE V/S LOAN STATUS

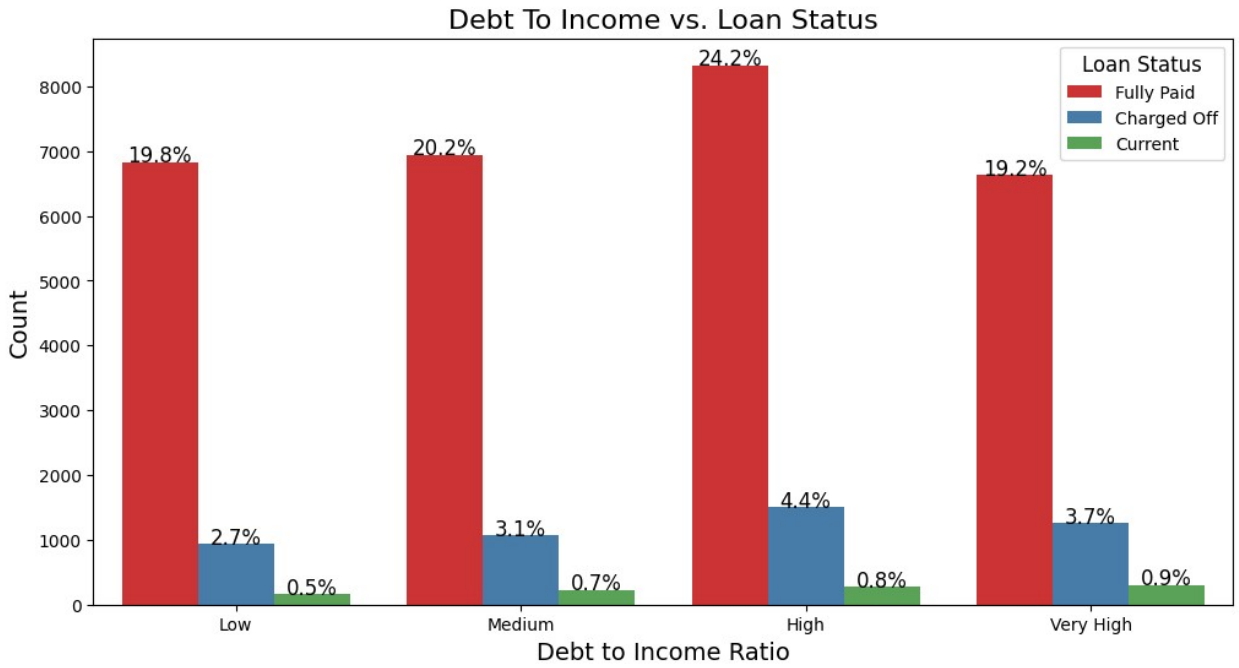
```
plt.figure(figsize=(12, 6))

# Plot the Purpose vs. Loan Status with a hue
plot = sns.countplot(x="purpose", hue="loan_status",
data=filtered_purpose_df, palette="muted")

# Set title and labels
plt.title('Purpose vs. Loan Status', fontsize=16)
plt.xlabel("Purpose", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=45, ha="right")

# Calculate and add percentage labels to the bars
total = len(filtered_purpose_df)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```



DEBT-TO-INCOME V/S LOAN STATUS

```
plt.figure(figsize=(12, 6))

# Plot the Debt To Income vs. Loan Status with a hue
plot = sns.countplot(x="categorised_dti", hue="loan_status",
data=loan, palette="Set1", order=order_category)

# Set title and labels
plt.title('Debt To Income vs. Loan Status', fontsize=16)
plt.xlabel("Debt to Income Ratio", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=0)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
```

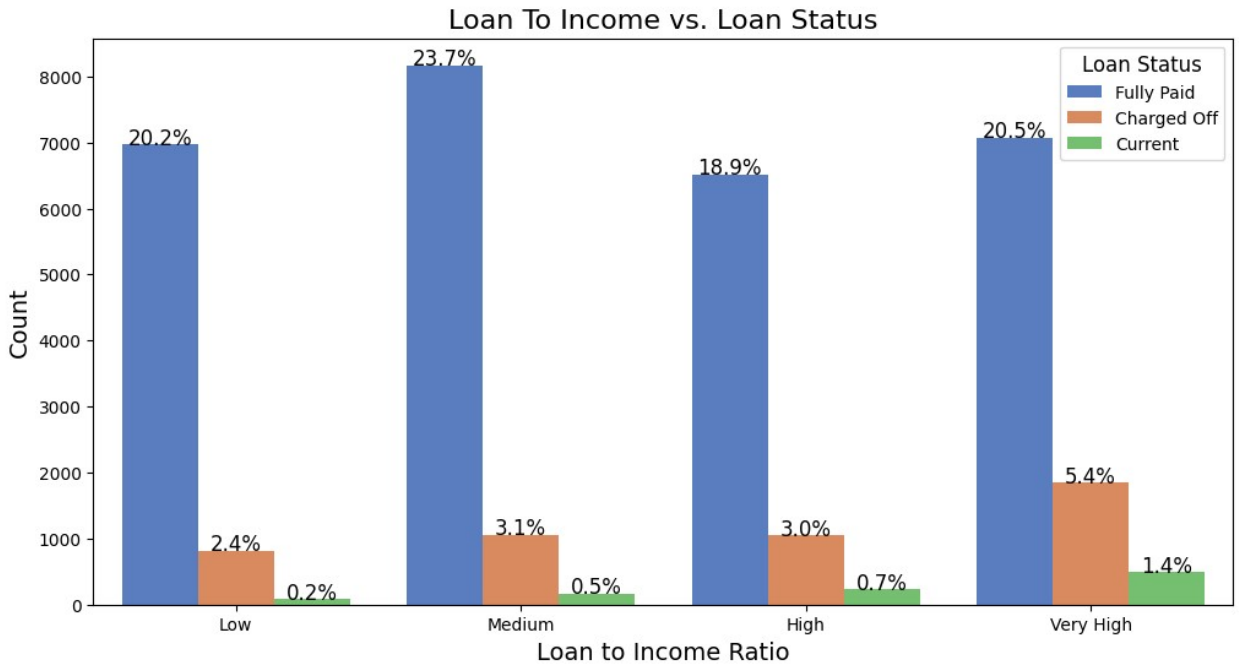
```
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)
plt.show()
```



LOAN-TO-INCOME V/S LOAN STATUS

```
plt.figure(figsize=(12, 6))

# Plot the Loan To Income vs. Loan Status with a hue
plot = sns.countplot(x="categorized_inc_ratio", hue="loan_status",
data=loan, palette="muted", order=order_category)

# Set title and labels
plt.title('Loan To Income vs. Loan Status', fontsize=16)
plt.xlabel("Loan to Income Ratio", fontsize=14)
plt.ylabel("Count", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")

plt.xticks(rotation=0)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
```

```
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```



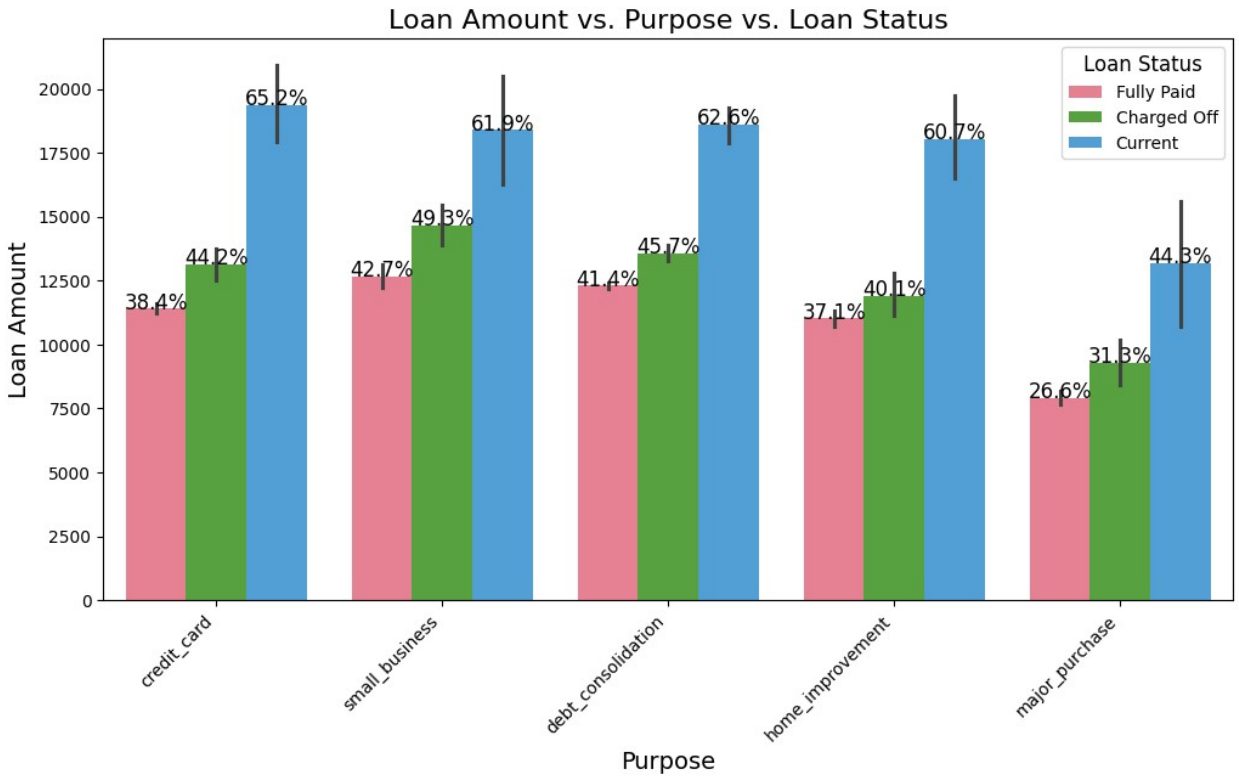
Information Obtained based on the analysis:-

- Applicants with a **Loan Term** of 36 months are defaulted **(7.7%)**
- In case of **Home Ownership**, applicants living in **Rented house** defaulted highest **(6.9%)**
- Applicants who have taken **very high** loan amount (>= 15000) defaulted highest **(5.0%)**
- Applicants paying **very high** (>= 14%) interest rates are defaulted highest **(6.4%)**
- Loan applicants with **Low** annual income (< 41000) defaulted highest **(4.1%)**
- Applicants of **Grade B** are the highest in default **(3.5%)** followed by **C (3.3%)** and **D (2.8%)** respectively.
- Applicants with **High debt to income** ratio defaulted highest **(4.5%)**
- Applicants who have **Very high loan to income** ratio (>= 25) defaulted highest **(5.5%)**
- The Default rate in terms of **Purpose** of loan are shown below:-
  - Debt Consolidation - 9.0%
  - Credit Card - 1.7%
  - Small Business - 1.5%
  - Home Improvement - 1.1%
  - Major Purchase - 0.7%

ANALYSIS OF MORE THAN 2 COLUMNS

LOAN STATUS V/S LOAN AMOUNT V/S PURPOSE

```
plt.figure(figsize=(12, 6))

# Plot the Loan Status vs. Purpose vs. Loan Amount with a hue
plot = sns.barplot(x="purpose", y="loan_amnt", hue="loan_status",
data=filtered_purpose_df, palette="husl")

# Set title and labels
plt.title('Loan Amount vs. Purpose vs. Loan Status', fontsize=16)
plt.xlabel("Purpose", fontsize=14)
plt.ylabel("Loan Amount", fontsize=14)

# Improve legend placement
plt.legend(title="Loan Status", title_fontsize=12, loc="upper right")
# Calculate and add percentage labels to the bars
total = len(filtered_purpose_df)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)
plt.xticks(rotation=45, ha="right")

plt.show()
```

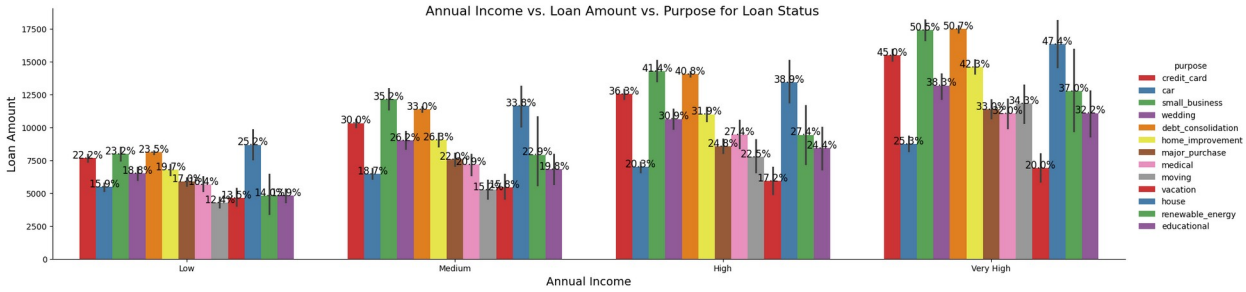


ANNUAL INCOME V/S LOAN AMOUNT V/S PURPOSE

```
plt.figure(figsize=(12, 6))

# Plot the Annual Income vs. Loan Amount vs. Purpose for Charged Off
Loan Status
plot = sns.catplot(x="categorised_annual_inc", y='loan_amnt',
hue='purpose', kind="bar",
                   data=loan, palette="Set1", aspect=3.9,
order=order_category)

# Set title and labels
plt.suptitle('Annual Income vs. Loan Amount vs. Purpose for Loan
Status', fontsize=16)
plt.xlabel("Annual Income", fontsize=14)
plt.ylabel("Loan Amount", fontsize=14)

# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
```

```
height),
                ha='center', fontsize=12)
plt.show()
```

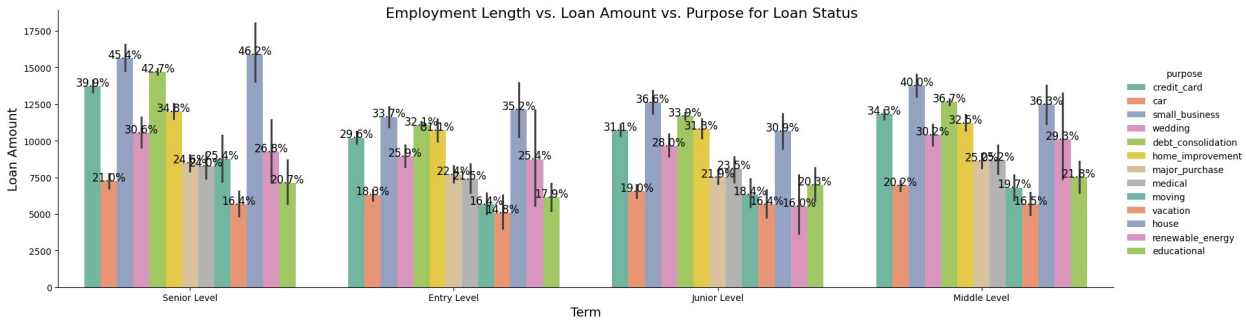
```
<Figure size 1200x600 with 0 Axes>
```



EMPLOYMENT LENGTH V/S LOAN AMOUNT V/S PURPOSE

```
plt.figure(figsize=(12, 6))

# Plot the Term vs. Loan Amount vs. Purpose for Charged Off Loan
Status
plot = sns.catplot(x="categorised_emp_length", y="loan_amnt",
hue="purpose", kind="bar",
                   data=loan, palette="Set2", aspect=3.5)

# Set title and labels
plt.suptitle('Employment Length vs. Loan Amount vs. Purpose for Loan
Status', fontsize=16)
plt.xlabel("Term", fontsize=14)
plt.ylabel("Loan Amount", fontsize=14)
# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```

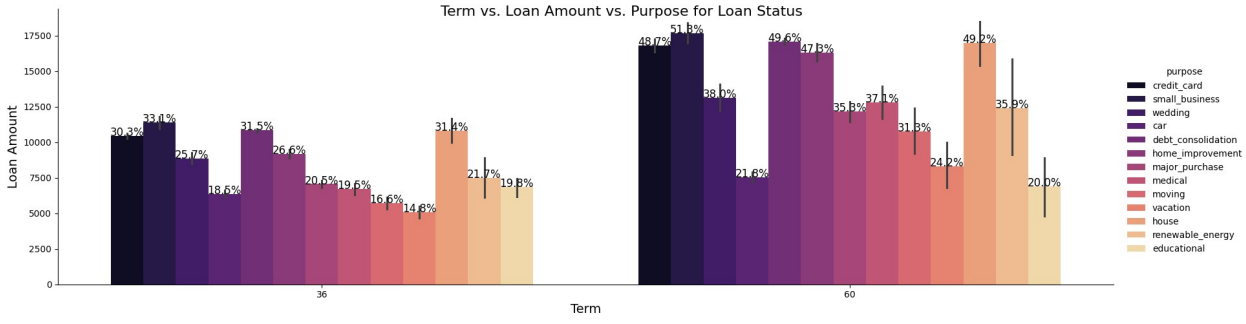
```
<Figure size 1200x600 with 0 Axes>
```



TERM V/S LOAN AMOUNT V/S PURPOSE

```
# Create a figure and axis with a larger size
plt.figure(figsize=(12, 6))

# Plot the Term vs. Loan Amount vs. Purpose for Charged Off Loan
Status
plot = sns.catplot(x="term", y="loan_amnt", hue="purpose", kind="bar",
                   data=loan, palette="magma", aspect=3.5)

# Set title and labels
plt.suptitle('Term vs. Loan Amount vs. Purpose for Loan Status',
fontsize=16)
plt.xlabel("Term", fontsize=14)
plt.ylabel("Loan Amount", fontsize=14)
# Calculate and add percentage labels to the bars
total = len(loan)  # Total number of applicants
ax = plt.gca()  # Get the current axis
for p in ax.patches:
    if not p.get_height():  # Skip bars with zero height
        continue
    height = int(p.get_height())
    percentage = (height / total) * 100
    ax.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2.,
height),
                ha='center', fontsize=12)

plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



Information Obtained based on the analysis:-

- No. of Applicants taking loan for **small business** is highest in default **(49.3%)**
- In the Annual Income column Applicants defaulted in:-
  - **LOW**
    - House **(25.2%)**
    - Debt Consolidation **(23.5%)**
    - Small Business **(23.2%)**
  - **MEDIUM**
    - Small Business **(35.2%)**
    - House **(33.8)**
    - Debt Consolidation **(33.0%)**
  - **HIGH**
    - Small Business **(41.4%)**
    - Debt Consolidation **(40.8%)**
    - House **(38.9%)**
  - **VERY HIGH**
    - Debt Consolidation **(50.7%)**
    - Small Business **(50.5)**
    - House **(47.4%)**
- In the Employment Length column Applicants defaulted in:-
  - **SENIOR LVL**
    - House **(46.7%)**
    - Small Business **(45.4%)**
    - Debt Consolidation **(42.7%)**
  - **ENTRY LVL**
    - House **(35.2%)**
    - Small Business **(33.7%)**
    - Debt Consolidation **(33.1%)**
  - **JUNIOR LVL**
    - Small Business **(36.6%)**
    - Debt Consolidation **(33.9%)**
    - Home Improvement **(31.3%)**
  - **MIDDLE LVL**
    - Small Business **(40.0%)**
    - Debt Consolidation **(36.7%)**
    - House **(36.3%)**
- In the Term column Applicants defaulted in:-
  - **36 Months**
    - Small Business **(33.1%)**
  - **60 Months**
    - Small Business **(51.3%)**

From the above observation we can say that the Deafult Rate is highest in Loans taken for

- Small Business
- Debt Consolidation

ANALYSIS OF DEFAULT RATE ACROSS DIFFERENT PARAMETERS

FOR DIFFERENT STATES

```
state_default_info = pd.DataFrame()

# Group the data by state and calculate counts for each loan status
state_default_info['Fully Paid'] = loan[loan['loan_status'] == 'Fully
Paid'].groupby('addr_state')['loan_status'].count()
state_default_info['Charged Off'] = loan[loan['loan_status'] ==
'Charged Off'].groupby('addr_state')['loan_status'].count()
state_default_info['Current'] = loan[loan['loan_status'] ==
'Current'].groupby('addr_state')['loan_status'].count()

# Fill NaN values with 0 for states without certain loan statuses
state_default_info.fillna(0, inplace=True)

# Calculate the default rate for each state
state_default_info['Default Rate'] = (state_default_info['Charged
Off'] / (state_default_info[['Fully Paid', 'Charged Off',
'Current']].max(axis=1))) * 100
state_default_info.columns = ['Fully_Paid', 'Charged_Off', 'Current',
'Default_Rate']


print('-'*20,'Default Rate Across the States','-'*20)
print(state_default_info)
```

```
-------------------- Default Rate Across the States
--------------------
            Fully_Paid  Charged_Off  Current  Default_Rate
addr_state
AK                  54         14.0      2.0     25.925926
AL                 338         46.0     15.0     13.609467
AR                 187         21.0      8.0     11.229947
AZ                 616         99.0     23.0     16.071429
CA                5103        952.0    128.0     18.655693
CO                 600         80.0     26.0     13.333333
CT                 550         82.0     20.0     14.909091
DC                 184         13.0      3.0      7.065217
DE                  88          8.0      1.0      9.090909
FL                1958        433.0     71.0     22.114402
```

GA	1003	179.0	33.0	17.846461
HI	120	24.0	7.0	20.000000
IA	4	0.0	0.0	0.000000
ID	3	1.0	0.0	33.333333
IL	1123	178.0	39.0	15.850401
IN	8	0.0	0.0	0.000000
KS	199	22.0	15.0	11.055276
KY	246	39.0	12.0	15.853659
LA	327	45.0	9.0	13.761468
MA	994	133.0	38.0	13.380282
MD	754	138.0	22.0	18.302387
ME	1	0.0	0.0	0.000000
MI	515	80.0	13.0	15.533981
MN	451	72.0	9.0	15.964523
MO	499	98.0	13.0	19.639279
MS	15	2.0	0.0	13.333333
MT	59	10.0	2.0	16.949153
NC	547	103.0	30.0	18.829982
NE	2	3.0	0.0	100.000000
NH	116	23.0	4.0	19.827586
NJ	1324	252.0	50.0	19.033233
NM	135	29.0	3.0	21.481481
NV	323	87.0	15.0	26.934985
NY	2724	417.0	89.0	15.308370
OH	908	131.0	39.0	14.427313
OK	213	35.0	10.0	16.431925
OR	322	66.0	14.0	20.496894
PA	1109	149.0	46.0	13.435528
RI	140	24.0	4.0	17.142857
SC	330	48.0	10.0	14.545455
SD	44	10.0	1.0	22.727273
TN	12	2.0	0.0	16.666667
TX	2042	256.0	56.0	12.536729
UT	196	33.0	5.0	16.836735
VA	1063	147.0	36.0	13.828786
VT	41	6.0	1.0	14.634146
WA	598	104.0	19.0	17.391304
WI	341	54.0	18.0	15.835777
WV	134	18.0	4.0	13.432836
WY	62	3.0	3.0	4.838710

FILTERING THE STATES TO TAKE ONLY THOSE WITH MORE THAN 100 APPLICANTS

```
# Check for '0.0' values in every column of state_default_ratio
zero_values_check = state_default_info.isin([0.0])

# Print the results
print(zero_values_check)
```

addr_state	Fully_Paid	Charged_Off	Current	Default_Rate
AK	False	False	False	False
AL	False	False	False	False
AR	False	False	False	False
AZ	False	False	False	False
CA	False	False	False	False
CO	False	False	False	False
CT	False	False	False	False
DC	False	False	False	False
DE	False	False	False	False
FL	False	False	False	False
GA	False	False	False	False
HI	False	False	False	False
IA	False	True	True	True
ID	False	False	True	False
IL	False	False	False	False
IN	False	True	True	True
KS	False	False	False	False
KY	False	False	False	False
LA	False	False	False	False
MA	False	False	False	False
MD	False	False	False	False
ME	False	True	True	True
MI	False	False	False	False
MN	False	False	False	False
MO	False	False	False	False
MS	False	False	True	False
MT	False	False	False	False
NC	False	False	False	False
NE	False	False	True	False
NH	False	False	False	False
NJ	False	False	False	False
NM	False	False	False	False
NV	False	False	False	False
NY	False	False	False	False
OH	False	False	False	False
OK	False	False	False	False
OR	False	False	False	False
PA	False	False	False	False
RI	False	False	False	False
SC	False	False	False	False
SD	False	False	False	False
TN	False	False	True	False
TX	False	False	False	False
UT	False	False	False	False
VA	False	False	False	False
VT	False	False	False	False
WA	False	False	False	False
WI	False	False	False	False

```
WV                False         False     False          False
WY                False         False     False          False
```

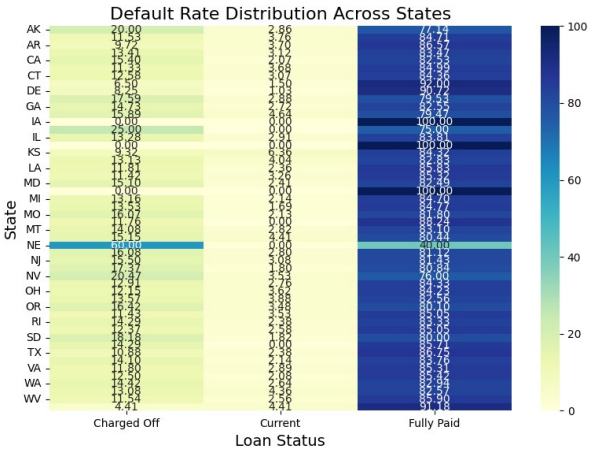
We need to rid of the zero values

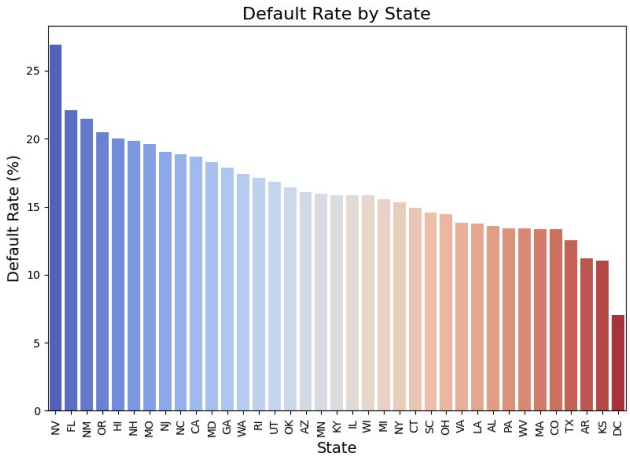
```
# Fill NaN values with 0 for states without certain loan statuses
state_default_info.fillna(0, inplace=True)

# Calculate the default rate for each state
state_default_info['Default_Rate'] =
(state_default_info['Charged_Off'] / state_default_info[['Fully_Paid',
'Charged_Off', 'Current']].max(axis=1)) * 100

# # Filter states with at least 100 applicants
min_applicants = 100
state_default_info =
state_default_info[state_default_info[['Fully_Paid', 'Charged_Off',
'Current']].sum(axis=1) >= min_applicants]


# Display the filtered DataFrame
print('-'*20,'Filtered Dateframe','-'*20)
print(state_default_info)
```

```
-------------------- Filtered Dateframe --------------------
            Fully_Paid  Charged_Off  Current  Default_Rate
addr_state
AL                 338         46.0     15.0     13.609467
AR                 187         21.0      8.0     11.229947
AZ                 616         99.0     23.0     16.071429
CA                5103        952.0    128.0     18.655693
CO                 600         80.0     26.0     13.333333
CT                 550         82.0     20.0     14.909091
DC                 184         13.0      3.0      7.065217
FL                1958        433.0     71.0     22.114402
GA                1003        179.0     33.0     17.846461
HI                 120         24.0      7.0     20.000000
IL                1123        178.0     39.0     15.850401
KS                 199         22.0     15.0     11.055276
KY                 246         39.0     12.0     15.853659
LA                 327         45.0      9.0     13.761468
MA                 994        133.0     38.0     13.380282
MD                 754        138.0     22.0     18.302387
MI                 515         80.0     13.0     15.533981
MN                 451         72.0      9.0     15.964523
MO                 499         98.0     13.0     19.639279
NC                 547        103.0     30.0     18.829982
NH                 116         23.0      4.0     19.827586
NJ                1324        252.0     50.0     19.033233
```

```
NM                   135          29.0       3.0     21.481481
NV                   323          87.0      15.0     26.934985
NY                  2724         417.0      89.0     15.308370
OH                   908         131.0      39.0     14.427313
OK                   213          35.0      10.0     16.431925
OR                   322          66.0      14.0     20.496894
PA                  1109         149.0      46.0     13.435528
RI                   140          24.0       4.0     17.142857
SC                   330          48.0      10.0     14.545455
TX                  2042         256.0      56.0     12.536729
UT                   196          33.0       5.0     16.836735
VA                  1063         147.0      36.0     13.828786
WA                   598         104.0      19.0     17.391304
WI                   341          54.0      18.0     15.835777
WV                   134          18.0       4.0     13.432836
```

```
# Creating a  table using pd.crosstab
contingency_table = pd.crosstab(index=loan['addr_state'],
columns=loan['loan_status'], normalize='index') * 100

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Plot the default rate distribution as a heatmap
sns.heatmap(contingency_table, annot=True, fmt=".2f", cmap="YlGnBu",
ax=axes[0])
axes[0].set_title('Default Rate Distribution Across States',
fontsize=16)
axes[0].set_xlabel('Loan Status', fontsize=14)
axes[0].set_ylabel('State', fontsize=14)

# Plot a bar chart for the state default rates
state_default_info = state_default_info.sort_values(by='Default_Rate',
ascending=False)
sns.barplot(x=state_default_info.index,
y=state_default_info['Default_Rate'], palette='coolwarm', ax=axes[1])
axes[1].set_title('Default Rate by State', fontsize=16)
axes[1].set_xlabel('State', fontsize=14)
axes[1].set_ylabel('Default Rate (%)', fontsize=14)
axes[1].tick_params(axis='x', rotation=90)

plt.tight_layout()
plt.show()
```





- According to the analysis subplots we can see that the state **NV (Nevada)** has the highest default rate at **26.9%**

```
# Filter states with a default rate higher than 20
high_default_rate_states =
state_default_info[state_default_info['Default_Rate'] > 20]

# Print the filtered DataFrame
print(high_default_rate_states)
```

addr_state	Fully_Paid	Charged_Off	Current	Default_Rate
NV	323	87.0	15.0	26.934985
FL	1958	433.0	71.0	22.114402
NM	135	29.0	3.0	21.481481
OR	322	66.0	14.0	20.496894

FOR INTEREST RATES

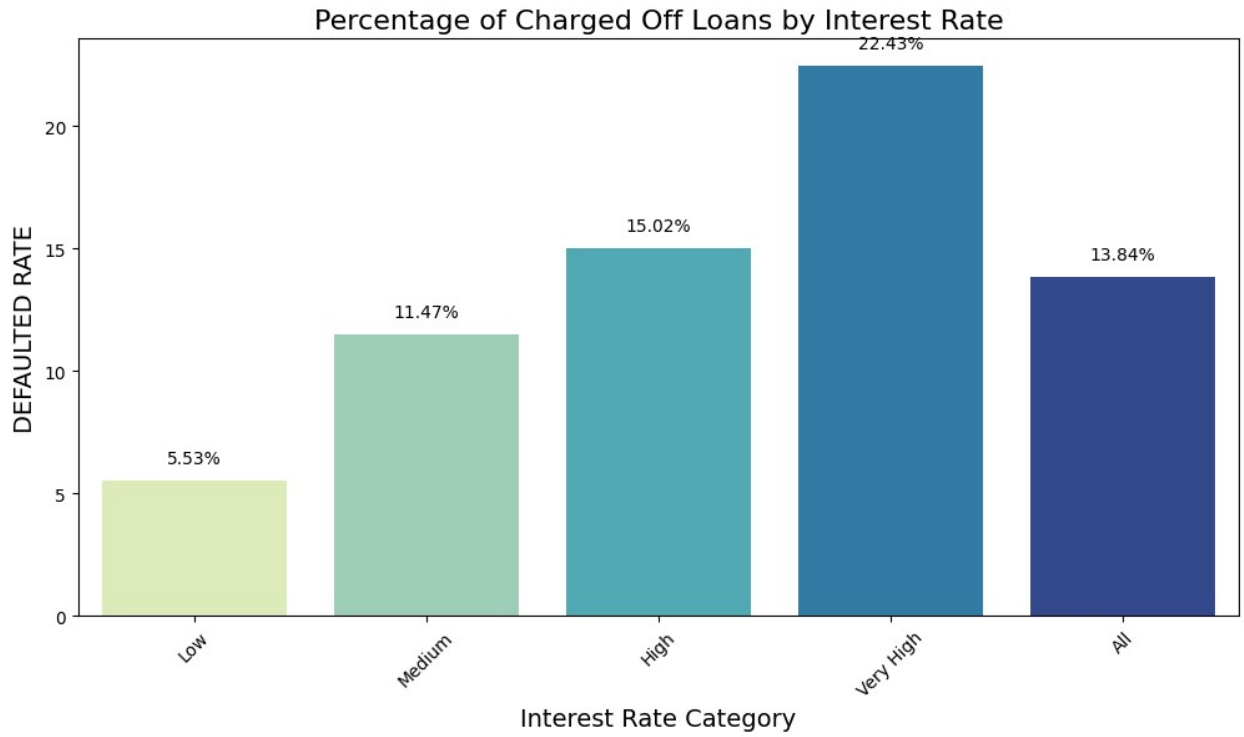
```
# Percentage default for int_rate
int_rate_loan_df = loan.copy()

# Sort categorised_int_rates based on custom sorting order
int_rate_loan_df['categorised_int_rate_perc'] =
pd.Categorical(int_rate_loan_df['categorised_int_rate_perc'],
categories=order_category, ordered=True)
int_rate_loan_df =
int_rate_loan_df.sort_values(by='categorised_int_rate_perc')

# Create a contingency table
int_rate_crosstab =
pd.crosstab(index=int_rate_loan_df['categorised_int_rate_perc'],
columns=int_rate_loan_df['loan_status'], margins=True,
margins_name="All")

# Calculate the percentage of charged off loans
```

```
int_rate_crosstab['percentage_defaulted'] =
round((int_rate_crosstab['Charged Off'] / int_rate_crosstab['All']) *
100, 2)

int_rate_crosstab
```

loan_status	Charged Off	Current	Fully Paid	All
categorised_int_rate_perc				
Low	465	28	7917	8410
Medium	1107	220	8324	9651
High	975	179	5339	6493
Very High	2222	539	7145	9906
All	4769	966	28725	34460

loan_status	percentage_defaulted
categorised_int_rate_perc	
Low	5.53
Medium	11.47
High	15.02
Very High	22.43
All	13.84

```
# Plot the percentage of charged off loans
plt.figure(figsize=(12, 6))
sns.barplot(x=int_rate_crosstab.index,
y=int_rate_crosstab['percentage_defaulted'], palette='YlGnBu')
plt.title('Percentage of Charged Off Loans by Interest Rate',
fontsize=16)
plt.xlabel('Interest Rate Category', fontsize=14)
plt.ylabel('DEFAULTED RATE', fontsize=14)
plt.xticks(rotation=45)

# Add data labels
ax = plt.gca()
for p in ax.patches:
    percentage = p.get_height()
    ax.annotate(f'{percentage:.2f}%', (p.get_x() + p.get_width() / 2.,
percentage),
                ha='center', fontsize=10, color='black', xytext=(0,
10), textcoords='offset points')

plt.show()
```



- According to this Barplot we can see that the Applicants with the **Very High (22.43%)** Interest Rate have the highest default rate.

FOR DIFFERENT PURPOSES

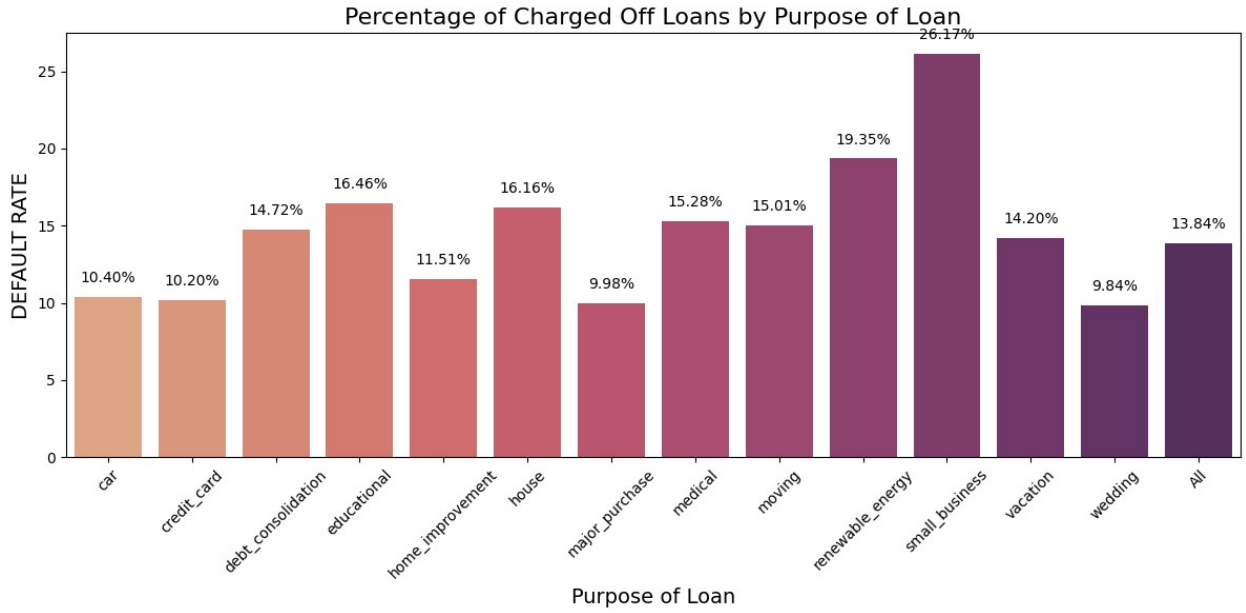
```
purpose_loan_df = loan.copy()

# Create a contingency table
purpose_crosstab = pd.crosstab(index=purpose_loan_df['purpose'],
columns=purpose_loan_df['loan_status'], margins=True,
margins_name="All")

# Calculate the percentage of charged off loans
purpose_crosstab['percentage_defaulted'] =
round((purpose_crosstab['Charged Off'] / purpose_crosstab['All']) *
100, 2)

purpose_crosstab
```

loan_status	Charged Off	Current	Fully Paid	All
purpose				
car	155	49	1286	1490
credit_card	506	100	4356	4962
debt_consolidation	2665	561	14876	18102
educational	52	0	264	316
home_improvement	322	90	2386	2798

```
house                          58        14          287    359
major_purchase                209        35         1851   2095
medical                       101        12          548    661
moving                         83         7          463    553
renewable_energy               18         1           74     93
small_business                459        73         1222   1754
vacation                       50         4          298    352
wedding                        91        20          814    925
All                          4769       966        28725  34460
```

```
loan_status          percentage_defaulted
purpose
car                                10.40
credit_card                        10.20
debt_consolidation                 14.72
educational                        16.46
home_improvement                   11.51
house                              16.16
major_purchase                      9.98
medical                            15.28
moving                             15.01
renewable_energy                   19.35
small_business                     26.17
vacation                           14.20
wedding                             9.84
All                                13.84
```

```
# Plot the percentage of charged off loans
plt.figure(figsize=(12, 6))
sns.barplot(x=purpose_crosstab.index,
y=purpose_crosstab['percentage_defaulted'], palette='flare')
plt.title('Percentage of Charged Off Loans by Purpose of Loan',
fontsize=16)
plt.xlabel('Purpose of Loan', fontsize=14)
plt.ylabel('DEFAULT RATE', fontsize=14)
plt.xticks(rotation=45)

# Add data labels
ax = plt.gca()
for p in ax.patches:
    percentage = p.get_height()
    ax.annotate(f'{percentage:.2f}%', (p.get_x() + p.get_width() / 2.,
percentage),
                ha='center', fontsize=10, color='black', xytext=(0,
10), textcoords='offset points')

plt.tight_layout()
plt.show()
```



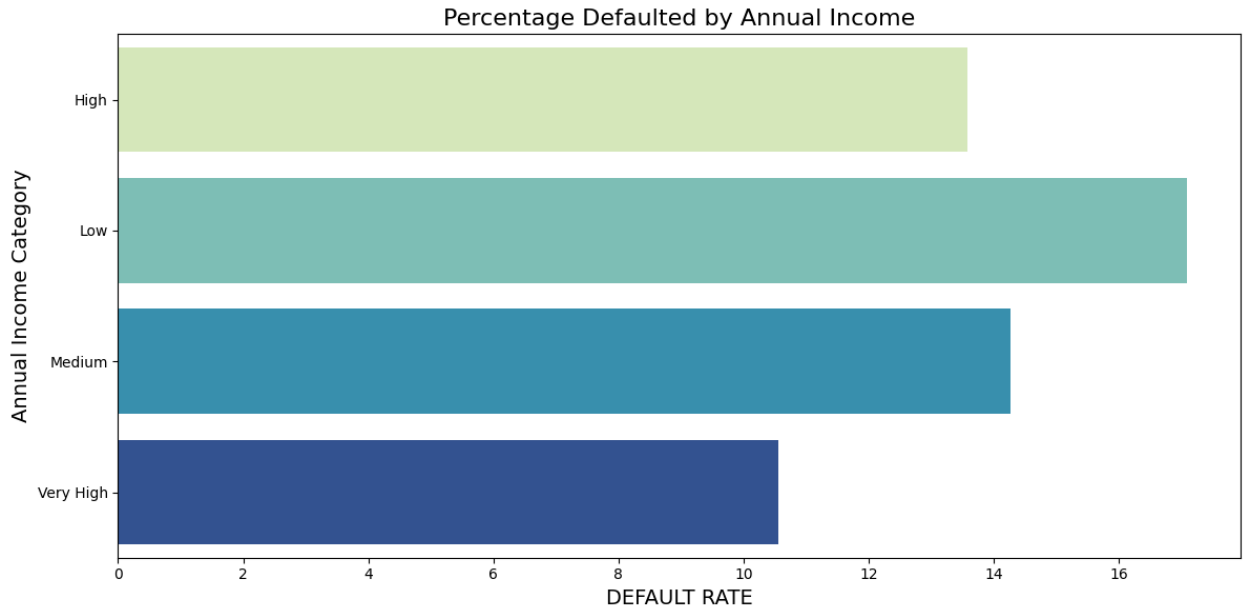
- According to the the above barplot the PURPOSE **Small Business (26.17%)** has the highest default rate

ANNUAL INCOME

```
# Sort categorised_annual_inc based on custom sorting
annual_inc_df = loan.copy()  # Create a copy of the DataFrame for
processing
annual_inc_df['categorised_annual_inc'] =
annual_inc_df['categorised_annual_inc'].astype("category")
annual_inc_df['categorised_annual_inc'].cat.set_categories(order_categ
ory)
annual_inc_df.sort_values(["categorised_annual_inc"], inplace=True)

# Create a contingency table
annual_inc_crosstab =
pd.crosstab(index=annual_inc_df['categorised_annual_inc'],
columns=annual_inc_df['loan_status'], margins=True,
margins_name="All")

# Calculate the percentage of charged-off loans
annual_inc_crosstab['percentage_defaulted'] =
round((annual_inc_crosstab['Charged Off'] /
annual_inc_crosstab['All']) * 100, 2)

# Drop the last row (the 'All' row)
annual_inc_crosstab.drop(annual_inc_crosstab.tail(1).index,
inplace=True)

# Display the contingency table
annual_inc_crosstab
```

```
loan_status              Charged Off  Current  Fully Paid   All  \
categorised_annual_inc
High                            1208      273        7414  8895
Low                             1406      137        6686  8229
Medium                          1247      254        7239  8740
Very High                        908      302        7386  8596

loan_status              percentage_defaulted
categorised_annual_inc
High                                   13.58
Low                                    17.09
Medium                                 14.27
Very High                              10.56
```

```
# Plot the data
plt.figure(figsize=(12, 6))
sns.barplot(data=annual_inc_crosstab, x='percentage_defaulted',
y=annual_inc_crosstab.index, palette='YlGnBu')
plt.title('Percentage Defaulted by Annual Income', fontsize=16)
plt.xlabel('DEFAULT RATE', fontsize=14)
plt.ylabel('Annual Income Category', fontsize=14)

plt.tight_layout()
plt.show()
```



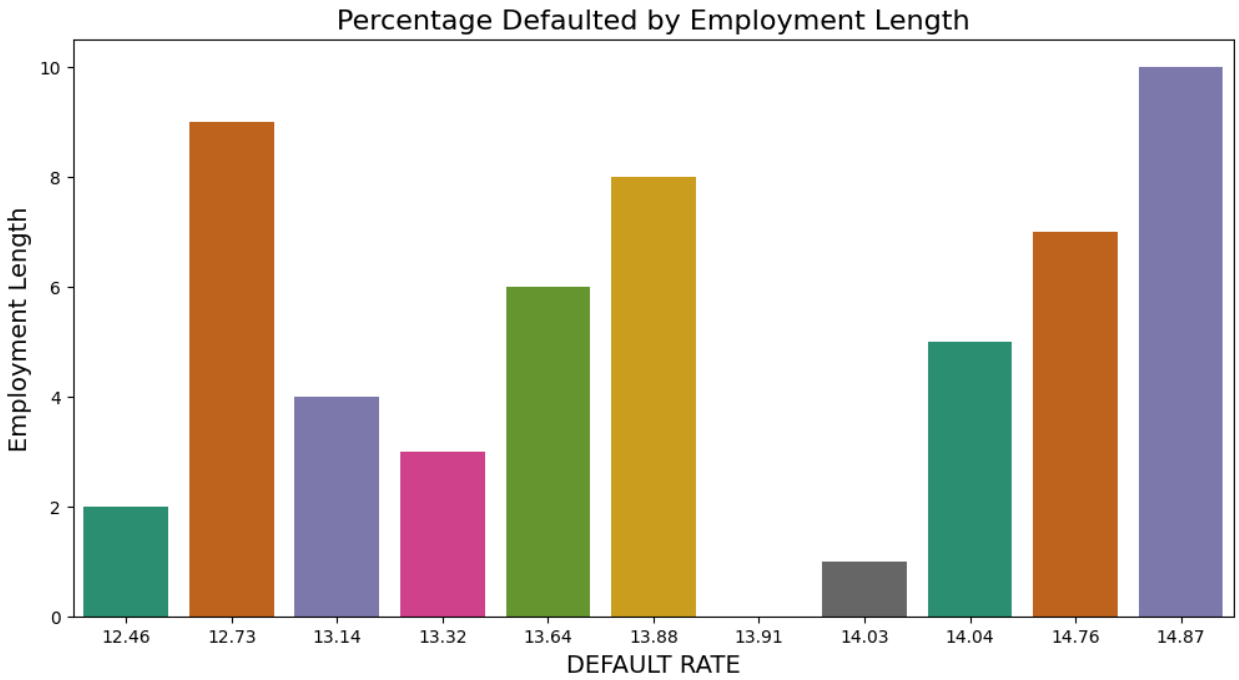
- According to the the above barplot the Applicants having **LOW (17.09%)** Annual Income has the highest default rate

FOR EMPLOYEMNT LENGTH

```
# Calculate the percentage of defaults for employment length
emp_length_crosstab = pd.crosstab(loan['emp_length'],
loan['loan_status'], margins=True)
emp_length_crosstab.drop(emp_length_crosstab.tail(1).index,
inplace=True)
emp_length_crosstab['percentage_defaulted'] = round(100 *
(emp_length_crosstab['Charged Off'] / emp_length_crosstab['All']), 2)

emp_length_crosstab
```

```
loan_status  Charged Off  Current  Fully Paid   All
percentage_defaulted
emp_length

0                    561       63        3408  4032
13.91
1                    400       63        2389  2852
14.03
2                    485       88        3321  3894
12.46
3                    489       66        3115  3670
13.32
4                    406       83        2600  3089
13.14
5                    414       77        2458  2949
14.04
6                    276       51        1696  2023
13.64
7                    235       54        1303  1592
14.76
8                    184       34        1108  1326
13.88
9                    144       30         957  1131
12.73
10                  1175      357        6370  7902
14.87
```

```
# Plot the data as a bar plot
plt.figure(figsize=(12, 6))
sns.barplot(data=emp_length_crosstab, x='percentage_defaulted',
y=emp_length_crosstab.index, palette='Dark2')
plt.title('Percentage Defaulted by Employment Length', fontsize=16)
plt.xlabel('DEFAULT RATE', fontsize=14)
plt.ylabel('Employment Length', fontsize=14)


plt.show()
```



- In the Employment Length column Applicants with:-
  - **2 to 4 years** have the least default rate
  - **8 to 10 years** have the most default rate

CONCLUSION

- **We have taken 5 variables in account to provide meaningful insights based on our observations obtained:-**
  - Different States
  - Interest Rate
  - Different Purposes
  - Annual Income
  - Employment Length
- The Bank should focus more on the Applicants with following characteristics:-
  - Having ***2 - 4 years*** of Employment length
  - Having ***Very High*** Income Group
  - Taking loan for ***Wedding Purpose***
  - Taking the ***Low Interest Rate (<9%)***
  - Applicants taking loan term of ***36 months***.

- The Bank should avoid Applicants with the following characteristics:-
  - The Purpose is ***Small Business***
  - Having ***Low*** Income Group
  - Taking the ***Very High Interest Rate (>15%)***

  - Applicants taking loan term of ***60 months***.
- We are going one step further to create a **Report** of the processed dataset ***loan*** to provide a better undrstanding

```
# from ydata_profiling import ProfileReport
# Final_Report = ProfileReport(loan, title="Final Profiling Report")
# Final_Report.to_file(output_file='Final ProfilingReport.html')
```