Loan_Data_Exploration_Part1

September 7, 2022

1 Part I - Exploration of the `Prosper Loan Data`

1.1 by Kelvin Njuguna

1.2 Introduction

The Loan data from prosper contains a total of 113,937 loan records, each with a total of 81 variables. Due to the bulk nature of the dataset, for this Exploratory exercise, only 15 loan variables will be considered. they include: > 1. **Term:** The length of the loan expressed in months. > 2. **ProsperScore:** A custom risk score built using historical Prosper data. The score ranges from 1-10, with 10 being the best, or lowest risk score. Applicable for loans originated after July 2009. > 3. **BorrowerState:** The two letter abbreviation of the state of the address of the borrower at the time the Listing was created. > 4. **Occupation:** The Occupation selected by the Borrower at the time they created the listing. > 5. **EmploymentStatus:** The employment status of the borrower at the time they posted the listing. > 6. **EmploymentStatusDuration:** The length in months of the employment status at the time the listing was created. > 7. **IsBorrowerHomeowner:** A Borrower will be classified as a homowner if they have a mortgage on their credit profile or provide documentation confirming they are a homeowner. > 8. **CreditScoreMid_range:** The arithmetic mean of the `CreditScoreRangeLower` and `CreditScoreRangeUpper` > 9. **DelinquenciesLast7Years:** Number of delinquencies in the past 7 years at the time the credit profile was pulled. > 10. **StatedMonthlyIncome:** The monthly income the borrower stated at the time the listing was created. > 11. **LoanOriginalAmount:** The origination amount of the loan. > 12. **MonthlyLoanPayment:** The scheduled monthly loan payment. > 13. **LP_CustomerPayments:** Pre charge-off cumulative gross payments made by the borrower on the loan. If the loan has charged off, this value will exclude any recoveries. > 14. **LP_InterestandFees:** Pre charge-off cumulative interest and fees paid by the borrower. If the loan has charged off, this value will exclude any recoveries.

1.3 Load the Datasets

[1]:
```
# Import the required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import plotly.express as px

%matplotlib inline
```

[2]:
```
# Loading the data
loan_df = pd.read_csv('Datasets/prosperLoanData.csv')
```

[3]:
```
loan_df.head()
```

[3]:
```
                ListingKey  ListingNumber            ListingCreationDate  \
0  1021339766868145413AB3B         193129  2007-08-26 19:09:29.263000000
1  10273602499503308B223C1        1209647  2014-02-27 08:28:07.900000000
2  0EE9337825851032864889A          81716  2007-01-05 15:00:47.090000000
3  0EF5356002482715299901A         658116  2012-10-22 11:02:35.010000000
4  0F023589499656230C5E3E2         909464  2013-09-14 18:38:39.097000000

  CreditGrade  Term LoanStatus           ClosedDate  BorrowerAPR  \
0           C    36  Completed  2009-08-14 00:00:00      0.16516
1         NaN    36    Current                  NaN      0.12016
2          HR    36  Completed  2009-12-17 00:00:00      0.28269
3         NaN    36    Current                  NaN      0.12528
4         NaN    36    Current                  NaN      0.24614

   BorrowerRate  LenderYield  ...  LP_ServiceFees  LP_CollectionFees  \
0        0.1580       0.1380  ...         -133.18                0.0
1        0.0920       0.0820  ...            0.00                0.0
2        0.2750       0.2400  ...          -24.20                0.0
3        0.0974       0.0874  ...         -108.01                0.0
4        0.2085       0.1985  ...          -60.27                0.0

   LP_GrossPrincipalLoss  LP_NetPrincipalLoss LP_NonPrincipalRecoverypayments  \
0                    0.0                  0.0                             0.0
1                    0.0                  0.0                             0.0
2                    0.0                  0.0                             0.0
3                    0.0                  0.0                             0.0
4                    0.0                  0.0                             0.0

   PercentFunded  Recommendations InvestmentFromFriendsCount  \
0            1.0                0                          0
1            1.0                0                          0
2            1.0                0                          0
3            1.0                0                          0
4            1.0                0                          0

  InvestmentFromFriendsAmount Investors
0                         0.0       258
1                         0.0         1
2                         0.0        41
```

```
3                          0.0        158
4                          0.0         20

[5 rows x 81 columns]
```

1.4 Preliminary Wrangling

1.4.1 What is the structure of your dataset?

[4]:
```
loan_df.shape
```

[4]:
```
(113937, 81)
```

The Loans dataframe has 113937 records, with each record having 81 variables. As mentioned above, for this project, there are only 14 variables of interest. Therefore, it is important to drop off the unwanted variables and only retain the 14 variables of interest. Since we want to retain the original dataset intact, it is necessary to make a copy of the dataset and work with copy. **Note** _Since `CreditScoreMid_range` is a calculated value which is obtained by finding the arithmetic average of `CreditScoreRangeLower` and `CreditScoreRangeUpper`, for now, we will retain the `CreditScoreRangeLower` and `CreditScoreRangeUpper` columns.

Make a copy of the original dataset

[5]:
```
loan_copy_df = loan_df.copy()
```

Verify that the copy of the data was created

[6]:
```
# If the copy of the dataframe was created successfully, the assert statement␣
↪below will pass without an assertion
# error.
assert loan_copy_df.equals(loan_df)
```

[7]:
```
needed_columns = ['Term', 'ProsperScore', 'BorrowerState', 'Occupation',␣
↪'EmploymentStatus', 'EmploymentStatusDuration',
                  'IsBorrowerHomeowner', 'CreditScoreRangeLower',␣
↪'CreditScoreRangeUpper', 'DelinquenciesLast7Years',
                  'StatedMonthlyIncome', 'LoanOriginalAmount',␣
↪'MonthlyLoanPayment', 'LP_CustomerPayments',
                  'LP_InterestandFees', 'LoanNumber']
```

[8]:
```
# Drop the unwanted columns
for col in loan_copy_df:
    if col not in needed_columns:
        loan_copy_df.drop(columns=col, axis = 1, inplace=True)
```

[9]:
```
# Verify that the columns were dropped successfully
# If the columns were dropped successfully, the assert tests below will pass␣
↪without spitting an assertion error
```

```
for col in loan_copy_df:
    assert col in needed_columns
for column in needed_columns:
    assert column in loan_copy_df.columns
```

Now, we need to check that all the columns hold the correct data types.

[10]:
```
loan_copy_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 113937 entries, 0 to 113936
Data columns (total 16 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Term                       113937 non-null  int64
 1   ProsperScore               84853 non-null   float64
 2   BorrowerState              108422 non-null  object
 3   Occupation                 110349 non-null  object
 4   EmploymentStatus           111682 non-null  object
 5   EmploymentStatusDuration   106312 non-null  float64
 6   IsBorrowerHomeowner        113937 non-null  bool
 7   CreditScoreRangeLower      113346 non-null  float64
 8   CreditScoreRangeUpper      113346 non-null  float64
 9   DelinquenciesLast7Years    112947 non-null  float64
 10  StatedMonthlyIncome        113937 non-null  float64
 11  LoanNumber                 113937 non-null  int64
 12  LoanOriginalAmount         113937 non-null  int64
 13  MonthlyLoanPayment         113937 non-null  float64
 14  LP_CustomerPayments        113937 non-null  float64
 15  LP_InterestandFees         113937 non-null  float64
dtypes: bool(1), float64(9), int64(3), object(3)
memory usage: 13.1+ MB
```

All the columns hold data in the correct datatypes.

Check for null entries

[11]:
```
loan_copy_df.isnull().sum()
```

[11]:
```
Term                            0
ProsperScore                29084
BorrowerState                5515
Occupation                   3588
EmploymentStatus             2255
EmploymentStatusDuration     7625
IsBorrowerHomeowner             0
CreditScoreRangeLower         591
CreditScoreRangeUpper         591
DelinquenciesLast7Years       990
```

```
StatedMonthlyIncome              0
LoanNumber                       0
LoanOriginalAmount               0
MonthlyLoanPayment               0
LP_CustomerPayments              0
LP_InterestandFees               0
dtype: int64
```

The information above reveals that the data is missing some information that is critical to our analysis: > 1. The `employment status` and `occupation` are critical because without them, we cannot tell where the `stated monthly income` is coming from. > 2. The `Borrower state` is also an important variable since part of the analysis will focus on how the borrower's state of residence affects loan repayment. > 3. `Delinquencies withing the last 7 years` is a critical variable since it will play a major role in the data analysis process.

However, some of the missing records can be explained and are therefore not critical. For instance, `a loan applicant with no borrowing history will not have credit scores and prosper score`.

Therefore, we will need to delete entries with null values in the `Occupation`, `EmploymentStatus`, `BorrowerState` and `DelinquenciesLast7Years` columns.

Drop the entries with null values in the `Occupation`, `EmploymentStatus`, `BorrowerState` and `DelinquenciesLast7Years` columns.

[12]:
```
loan_copy_df.dropna(subset=['BorrowerState', 'Occupation', 'EmploymentStatus', 'DelinquenciesLast7Years'], inplace=True)
```

Verify that the records were dropped successfully.

[13]:
```
loan_copy_df.isnull().sum()
```

[13]:
```
Term                             0
ProsperScore                 22639
BorrowerState                    0
Occupation                       0
EmploymentStatus                 0
EmploymentStatusDuration      2931
IsBorrowerHomeowner              0
CreditScoreRangeLower            0
CreditScoreRangeUpper            0
DelinquenciesLast7Years          0
StatedMonthlyIncome              0
LoanNumber                       0
LoanOriginalAmount               0
MonthlyLoanPayment               0
LP_CustomerPayments              0
LP_InterestandFees               0
dtype: int64
```

After the operation of deleting the records with the null values, we can gain more insight about the null entries in the `ProsperScore` column. First, it comes to light that since all the records have

CreditScoreRange entries, we can argue that all the loan applicants have some borrowing history. Therefore, we would expect that all the records should have a ProspectScore since it is calculated using historical prosper data. **Therefore, we need to drop all the records with null entries in the ProsperScore column since it is evident that they are erroneous omissions that cannot be explained by the data**

[14]:
```python
# Drop the records with null entries in the ProsperScore column
loan_copy_df.dropna(subset=['ProsperScore'], inplace=True)
```

[15]:
```python
# Verify that the null entries were dropped successfully
loan_copy_df.isnull().sum()
```

[15]:
```
Term                          0
ProsperScore                  0
BorrowerState                 0
Occupation                    0
EmploymentStatus              0
EmploymentStatusDuration     13
IsBorrowerHomeowner           0
CreditScoreRangeLower         0
CreditScoreRangeUpper         0
DelinquenciesLast7Years       0
StatedMonthlyIncome           0
LoanNumber                    0
LoanOriginalAmount            0
MonthlyLoanPayment            0
LP_CustomerPayments           0
LP_InterestandFees            0
dtype: int64
```

[16]:
```python
# Check whether there are records with '0' as the entry in the␣
↪EmploymentStatusDuration column
loan_copy_df[loan_copy_df.EmploymentStatusDuration == 0]
```

[16]:
```
        Term  ProsperScore BorrowerState    Occupation EmploymentStatus  \
51        36           8.0            AR    Nurse (RN)         Employed
680       36           2.0            TX         Other          Retired
754       36           9.0            WA  Professional         Employed
804       36           6.0            KS         Other         Employed
833       36           2.0            FL         Other            Other
...      ...           ...           ...           ...              ...
113298    36           4.0            MD         Other            Other
113466    36           6.0            UT         Other            Other
113503    36           7.0            OK  Professional        Full-time
113548    60           8.0            CO         Other         Employed
113703    36           2.0            NY     Professor         Employed

        EmploymentStatusDuration  IsBorrowerHomeowner  CreditScoreRangeLower  \
51                           0.0                 True                  640.0
680                          0.0                False                  640.0
```

```
754                     0.0           True              800.0
804                     0.0          False              680.0
833                     0.0           True              760.0
...                     ...            ...                ...
113298                  0.0          False              700.0
113466                  0.0           True              740.0
113503                  0.0          False              760.0
113548                  0.0           True              700.0
113703                  0.0          False              660.0

        CreditScoreRangeUpper  DelinquenciesLast7Years  StatedMonthlyIncome  \
51                      659.0                      0.0          4853.333333
680                     659.0                     14.0          1750.000000
754                     819.0                      0.0          2583.333333
804                     699.0                      0.0          3833.333333
833                     779.0                      0.0          4500.000000
...                       ...                      ...                  ...
113298                  719.0                     12.0          4166.666667
113466                  759.0                      0.0          4166.666667
113503                  779.0                      0.0          2000.000000
113548                  719.0                      0.0          7666.666667
113703                  679.0                      0.0          6333.333333

        LoanNumber  LoanOriginalAmount  MonthlyLoanPayment  \
51           55623                2500               90.37
680         103738                3500              148.56
754          76485               10000              321.45
804          50568                2000               81.64
833          99150                4000              160.44
...            ...                 ...                 ...
113298      100909                5000              186.30
113466       66570                5000              189.61
113503       41005                3200              142.95
113548      134250               14000              367.81
113703      102597                4000              163.28

        LP_CustomerPayments  LP_InterestandFees
51                2439.9900            696.6000
680                742.8000            424.1100
754              10386.2100            386.2100
804               2113.5700            113.5700
833                962.6400            492.6000
...                     ...                 ...
113298            1928.7342            410.3542
113466            5304.7400            304.7400
113503            1010.6500            677.8400
113548               0.0000              0.0000
```

```
113703             813.4422           432.4822

[522 rows x 16 columns]
```

Notice that for the records of individuals who had not Employment status duration, the entry is 0. This means that the entries with blank values in the `EmploymentStatusDuration` column cannot be justified. Therefore, we will need to drop all the records with null entries in the `EmploymentStatusDuration` column

[17]:
```python
# Drop the records with null entries in the EmploymentStatusDuration column
loan_copy_df.dropna(subset='EmploymentStatusDuration', inplace=True)
```

[18]:
```python
# Verify that the records were dropped successfully
loan_copy_df.isnull().sum()
```

[18]:
```
Term                        0
ProsperScore                0
BorrowerState               0
Occupation                  0
EmploymentStatus            0
EmploymentStatusDuration    0
IsBorrowerHomeowner         0
CreditScoreRangeLower       0
CreditScoreRangeUpper       0
DelinquenciesLast7Years     0
StatedMonthlyIncome         0
LoanNumber                  0
LoanOriginalAmount          0
MonthlyLoanPayment          0
LP_CustomerPayments         0
LP_InterestandFees          0
dtype: int64
```

[19]:
```python
loan_copy_df.shape
```

[19]:
```
(83507, 16)
```

[20]:
```python
loan_copy_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 83507 entries, 1 to 113936
Data columns (total 16 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Term                      83507 non-null  int64
 1   ProsperScore              83507 non-null  float64
 2   BorrowerState             83507 non-null  object
 3   Occupation                83507 non-null  object
 4   EmploymentStatus          83507 non-null  object
 5   EmploymentStatusDuration  83507 non-null  float64
 6   IsBorrowerHomeowner       83507 non-null  bool
```

```
 7   CreditScoreRangeLower     83507 non-null  float64
 8   CreditScoreRangeUpper     83507 non-null  float64
 9   DelinquenciesLast7Years   83507 non-null  float64
 10  StatedMonthlyIncome       83507 non-null  float64
 11  LoanNumber                83507 non-null  int64
 12  LoanOriginalAmount        83507 non-null  int64
 13  MonthlyLoanPayment        83507 non-null  float64
 14  LP_CustomerPayments       83507 non-null  float64
 15  LP_InterestandFees        83507 non-null  float64
dtypes: bool(1), float64(9), int64(3), object(3)
memory usage: 10.3+ MB
```

[21]:
```
loan_copy_df.describe()
```

[21]:
```
               Term  ProsperScore  EmploymentStatusDuration  \
count  83507.000000  83507.000000              83507.000000
mean      42.515957      5.964877                103.344306
std       11.669560      2.375938                 96.219084
min       12.000000      1.000000                  0.000000
25%       36.000000      4.000000                 31.000000
50%       36.000000      6.000000                 75.000000
75%       60.000000      8.000000                148.000000
max       60.000000     11.000000                755.000000

       CreditScoreRangeLower  CreditScoreRangeUpper  DelinquenciesLast7Years  \
count           83507.000000           83507.000000             83507.000000
mean              699.649610             718.649610                 3.638893
std                47.201439              47.201439                 9.319570
min               600.000000             619.000000                 0.000000
25%               660.000000             679.000000                 0.000000
50%               700.000000             719.000000                 0.000000
75%               720.000000             739.000000                 2.000000
max               880.000000             899.000000                99.000000

       StatedMonthlyIncome     LoanNumber  LoanOriginalAmount  \
count         8.350700e+04   83507.000000        83507.000000
mean          5.966762e+03   86055.585472         9104.256541
std           8.297231e+03   28765.247582         6300.693843
min           0.000000e+00   38045.000000         1000.000000
25%           3.500000e+03   60616.500000         4000.000000
50%           5.000000e+03   87125.000000         7500.000000
75%           7.166667e+03  108399.500000        14000.000000
max           1.750003e+06  136486.000000        35000.000000

       MonthlyLoanPayment  LP_CustomerPayments  LP_InterestandFees
count        83507.000000         83507.000000        83507.000000
mean           292.488673          3718.453936         1055.368058
std            186.969305          4247.818219         1139.777457
```

```
min          0.000000      -2.349900     -2.349900
25%        158.030000     837.720000    264.090000
50%        252.480000    2267.880000    689.450000
75%        389.485000    4939.490000   1453.300000
max       2251.510000   37369.160000  10572.780000
```

As indicated in the introduction section, on of the datasets that will be using for the analysis phase is `CreditScoreMid_range`. We need to calculate this column from the `CreditScoreRangeLower` and `CreditScoreRangeUpper`

Calculate the `CreditScoreMid_range` column from the `CreditScoreRangeLower` and `CreditScoreRangeUpper` column data and drop the unwanted columns.

[22]:
```
# calculate the mid_range column
loan_copy_df['CreditScoreMid_range'] = loan_copy_df.loc[:,␣
 ↪['CreditScoreRangeLower', 'CreditScoreRangeUpper']].mean(axis=1)

# Now drop the unused columns.
loan_copy_df.drop(columns=['CreditScoreRangeLower', 'CreditScoreRangeUpper'],␣
 ↪inplace=True)
```

Create tests that verifies that the above operation was successful. `**Note:** If the operations were successful, the assert tests will pass without spitting assertion errors.`

[23]:
```
# Verify that the new column was created successfully
new_col = 'CreditScoreMid_range'
assert new_col in loan_copy_df.columns, "New column not created successfully"

# Verify that the data was dropped as required.
unused_cols=['CreditScoreRangeLower', 'CreditScoreRangeUpper']
for col in unused_cols:
    assert col not in loan_copy_df.columns, "There was an error dropping the␣
 ↪columns"
```

1.4.2 The data is now clean and can be saved in a new csv file which will be used for further analysis.

Save the cleaned data in a csv file named `clean_loan_df`

[24]:
```
loan_copy_df.to_csv('Datasets/clean_loan_data.csv', index=False)
```

1.5 Data Exploration

1.5.1 Load the Cleaned data into the pandas dataframe

[25]:
```python
clean_loan_df = loan_copy_df
# clean_loan_df = pd.read_csv('Datasets/clean_loan_data.csv')
clean_loan_df.sample(10)
```

[25]:
```
        Term  ProsperScore BorrowerState              Occupation  \
18395     36           7.0            IL                   Other
46290     36           6.0            NC                   Other
18031     36           4.0            FL          Postal Service
48127     60           7.0            TX                   Other
34634     12           1.0            CO                   Other
101044    36           4.0            MI            Construction
47146     36           7.0            TX                   Other
90057     60           4.0            NV                   Other
73309     60           5.0            SC           Skilled Labor
106384    36           7.0            TX  Food Service Management

       EmploymentStatus  EmploymentStatusDuration  IsBorrowerHomeowner  \
18395          Employed                     327.0                 True
46290         Full-time                     197.0                 True
18031          Employed                     192.0                 True
48127          Employed                      31.0                 True
34634     Self-employed                      55.0                 True
101044         Employed                     162.0                False
47146          Employed                       8.0                 True
90057          Employed                      65.0                False
73309          Employed                     219.0                False
106384         Employed                     152.0                 True

        DelinquenciesLast7Years  StatedMonthlyIncome  LoanNumber  \
18395                       0.0          6833.333333      120888
46290                       0.0          4166.666667       50418
18031                       0.0          5666.666667       79424
48127                       0.0         15000.000000       91742
34634                      10.0         25000.000000       58428
101044                      3.0          6250.000000       93654
47146                       0.0          5833.333333      129031
90057                       0.0          2083.333333       82543
73309                       2.0          4000.000000       98905
106384                      0.0          9750.000000      132407

        LoanOriginalAmount  MonthlyLoanPayment  LP_CustomerPayments  \
18395                 8500              294.66             585.8268
46290                 3000              109.95            3518.4000
18031                10000              358.22            2134.3200
```

```
48127           13000       290.43      2613.8700
34634            4000       381.65      4580.1800
101044           4000       142.19      4078.5600
47146           20000       658.47       658.4700
90057            5000       143.78      1869.1400
73309            4500       118.82       712.9200
106384          25000       829.04         0.0000

        LP_InterestandFees  CreditScoreMid_range
18395             210.7068                 709.5
46290             944.5200                 669.5
18031             819.0400                 689.5
48127            1130.9900                 729.5
34634             580.1800                 669.5
101044             78.5600                 709.5
47146             193.4700                 689.5
90057            1225.9300                 729.5
73309             438.8000                 669.5
106384              0.0000                 689.5
```

Question 1: How is the population of borrowers distributed throughout the States?

To check the number of borrowers in every state, we can simply use the pandas `value_counts()` function as shown below.

[26]:
```python
# Using the value_counts() function to view the number of borrowers per state.
borrowers_by_state = clean_loan_df.BorrowerState.value_counts()
borrowers_by_state
```

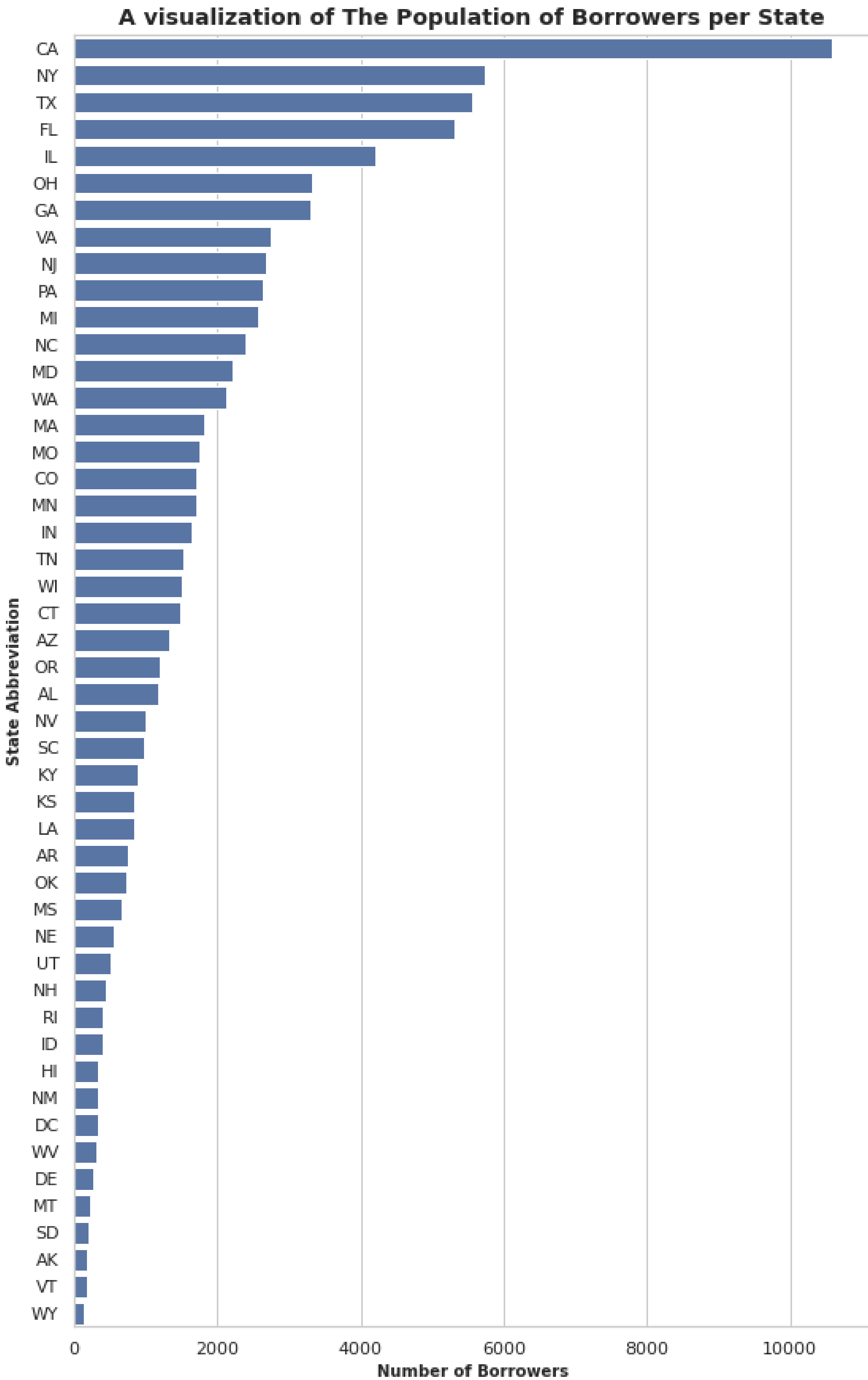
[26]:
```
CA    10589
NY     5743
TX     5552
FL     5314
IL     4204
OH     3316
GA     3296
VA     2740
NJ     2682
PA     2635
MI     2556
NC     2400
MD     2218
WA     2125
MA     1818
MO     1758
CO     1710
MN     1702
IN     1634
```

```
TN      1517
WI      1494
CT      1480
AZ      1327
OR      1187
AL      1163
NV       992
SC       976
KY       873
KS       839
LA       829
AR       754
OK       720
MS       657
NE       545
UT       509
NH       438
RI       403
ID       393
HI       340
NM       328
DC       325
WV       301
DE       265
MT       218
SD       188
AK       167
VT       165
WY       122
Name: BorrowerState, dtype: int64
```

Although this does give a dictionary showing the number of borrowers in every state, the data presentation is not that insightful. To make it more presentable, we can use seaborn to visualize it.

[27]:
```
# Visualize the data above using seaborn

# Set the theme of the visualization
sns.set_theme(style="whitegrid")

# Set the size of the visualization
f, ax = plt.subplots(figsize=(9,15))

# Set the color of the visualization
base_color = sns.color_palette()[0]

# Define the order in which the bars will appear in the visualization
population_order = clean_loan_df.BorrowerState.value_counts().index
```

```python
# Visualize the bar graph.
sns.countplot(data=clean_loan_df, y = 'BorrowerState', color = base_color, order=population_order);

# Set the labels and plot title
plt.title('A visualization of The Population of Borrowers per State', size = 14, weight='bold')
plt.xlabel('Number of Borrowers', size = 10, weight='bold')
plt.ylabel('State Abbreviation', size = 10, weight='bold');
```


A visualization of The Population of Borrowers per State
State Abbreviation
CA
NY
TX
FL
IL
OH
GA
VA
NJ
PA
MI
NC
MD
WA
MA
MO
CO
MN
IN
TN
WI
CT
AZ
OR
AL
NV
SC
KY
KS
LA
AR
OK
MS
NE
UT
NH
RI
ID
HI
NM
DC
WV
DE
MT
SD
AK
VT
WY
0
2000
4000
6000
8000
10000
Number of Borrowers

The visualization above clearly indicates that the California (CA) had the highest number of borrowers (Over 10,000), while Wyoming State had the lowest population of borrowers.

The above visualization looks much better than the initially displayed series that contained the state code as an index and the borrower population in that state as the values. However, much can still be done to make the data more presentable. It is possible to display the data in form of a heatmap on the United States geographical map.

The code in the cell below does just that…

[28]:
```
# First convert the Series to a pandas dataframe.
borrowers_by_state = borrowers_by_state.to_frame(name='borrower_population').
 ↪reset_index()
borrowers_by_state.head()
```

[28]:
```
  index  borrower_population
0    CA                10589
1    NY                 5743
2    TX                 5552
3    FL                 5314
4    IL                 4204
```

[29]:
```
# Rename the column with the name index to "State"
borrowers_by_state.rename(columns = {'index':'State'}, inplace=True)

# verify that the column header has been changed as required.
################## This should pass without an error ######################
assert 'index' not in borrowers_by_state.columns and 'State' in␣
 ↪borrowers_by_state.columns
```

[30]:
```
# Using the newly created dataframe, generate a heatmap indicating the␣
 ↪population of borrowers in every state.
fig = px.choropleth(borrowers_by_state,
                   locations='State',
                    locationmode='USA-states',
                   scope = 'usa',
                   color='borrower_population',
                   color_continuous_scale=px.colors.sequential.Inferno_r)
fig.update_layout(title_text = 'Borrower population by State',
                  title_font_size = 22,
                  title_font_color = 'black',
                  title_x = 0.5)
fig.show();
```

From a user's perspective, it is evident that California has the highest population of borrowers. In this visualization, the user can clearly see the location of every State, which makes the data even more engaging.

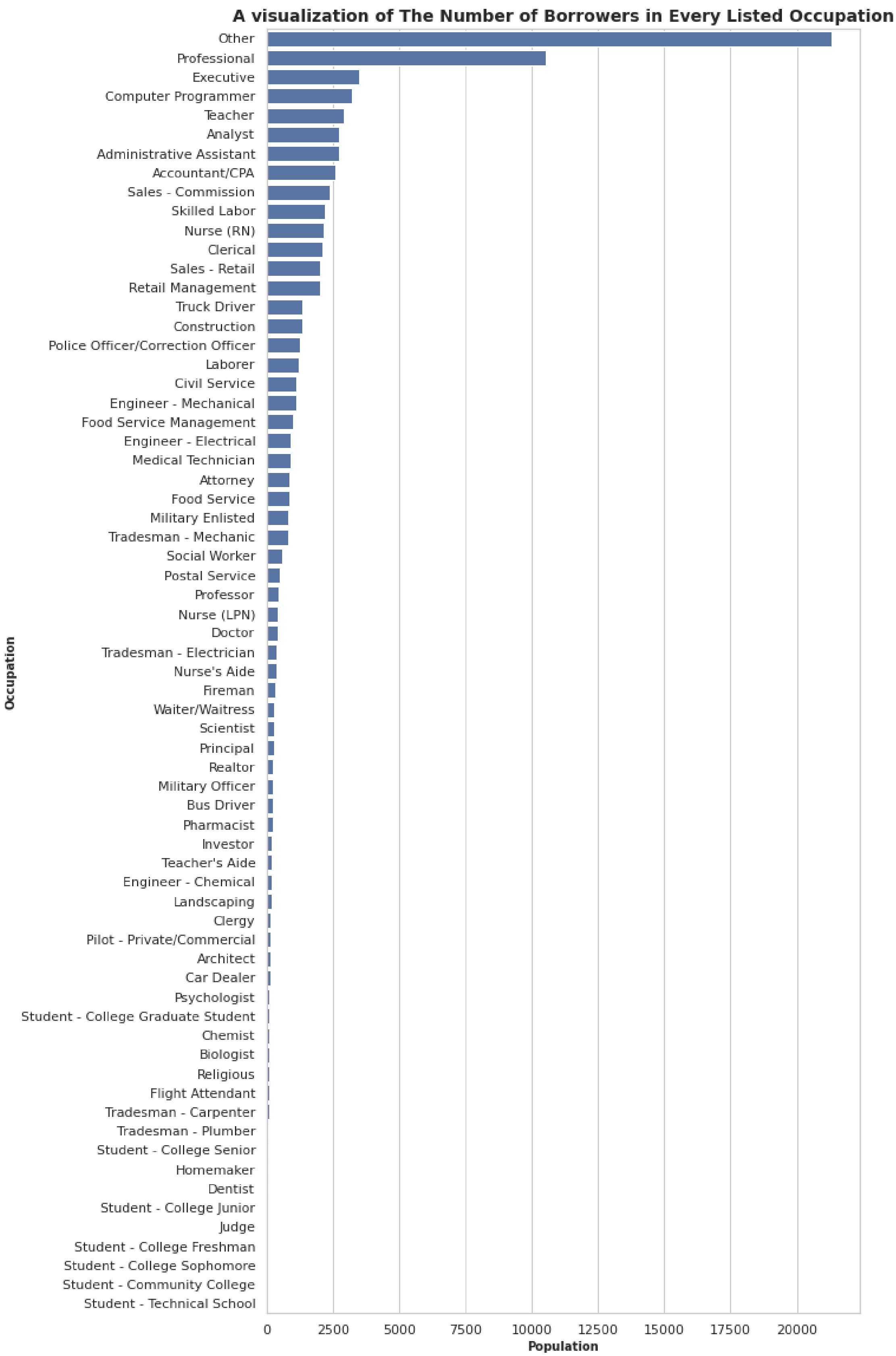
Question 2: How many borrowers fall under every listed occupation?

[31]:
```
clean_loan_df.Occupation.value_counts()
```

[31]:
```
Other                          21317
Professional                   10539
Executive                       3468
Computer Programmer             3236
Teacher                         2888
                               ...
Judge                             22
Student - College Freshman        17
Student - College Sophomore       16
Student - Community College       10
Student - Technical School         2
Name: Occupation, Length: 67, dtype: int64
```

Rubric Tip: Visualizations should depict the data appropriately so that the plots are easily interpretable. You should choose an appropriate plot type, data encodings, and formatting as needed. The formatting may include setting/adding the title, labels, legend, and comments. Also, do not overplot or incorrectly plot ordinal data.

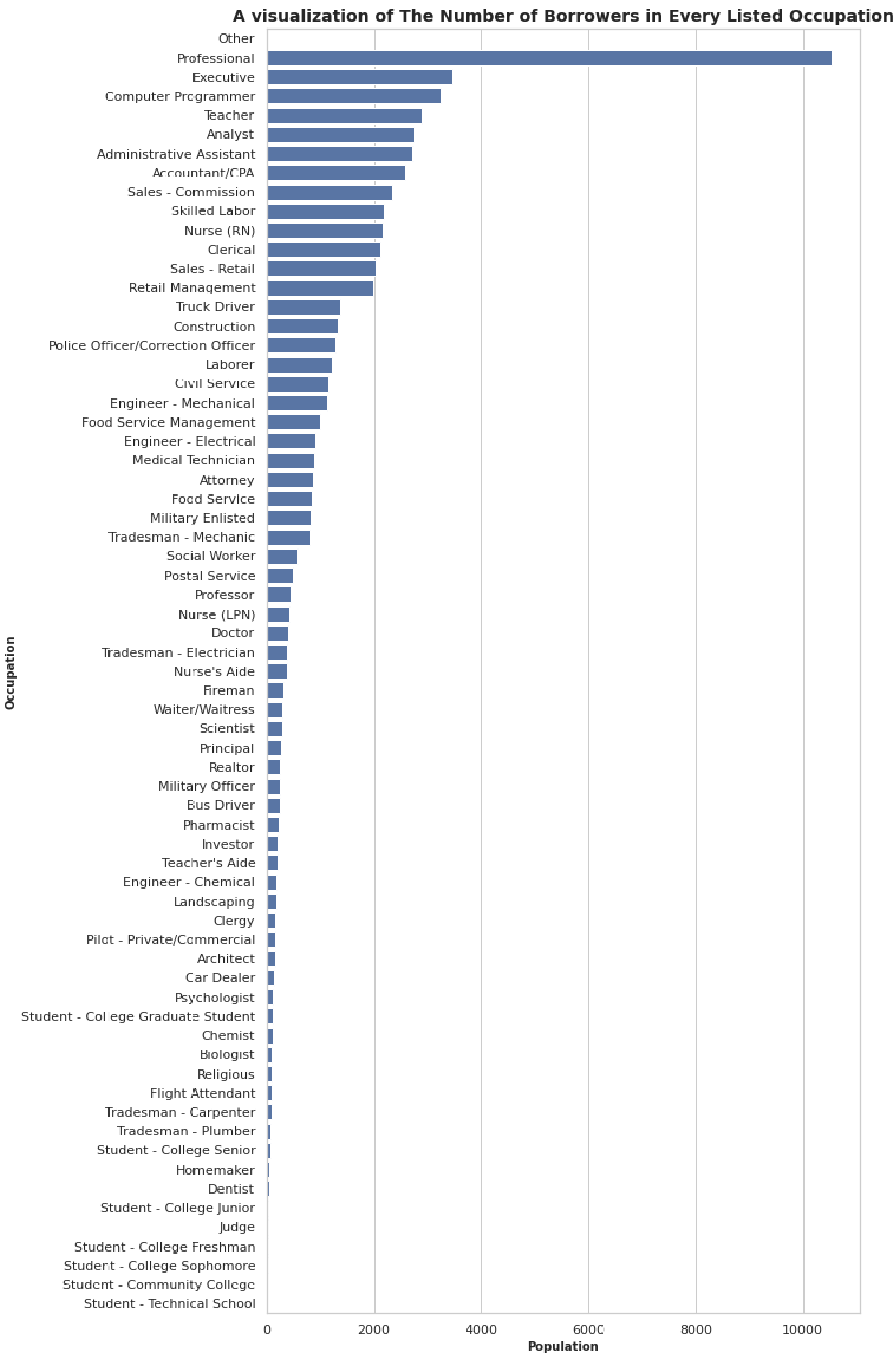
[32]:
```
# Set the theme of the visualization
sns.set_theme(style="whitegrid")

# Set the size of the visualization
f, ax = plt.subplots(figsize=(9,20))

# Set the color of the visualization
base_color = sns.color_palette()[0]

# Define the order in which the bars will appear in the visualization
occupations_order = clean_loan_df.Occupation.value_counts().index

# Visualize the bar graph.
sns.countplot(data=clean_loan_df, y = 'Occupation', color = base_color,␣
 ↪order=occupations_order);

# Set the labels and plot title
plt.title('A visualization of The Number of Borrowers in Every Listed␣
 ↪Occupation', size = 14, weight='bold')
plt.xlabel('Population', size = 10, weight='bold')
plt.ylabel('Occupation', size = 10, weight='bold');
```


A visualization of The Number of Borrowers in Every Listed Occupation
Occupation
Other
Professional
Executive
Computer Programmer
Teacher
Analyst
Administrative Assistant
Accountant/CPA
Sales - Commission
Skilled Labor
Nurse (RN)
Clerical
Sales - Retail
Retail Management
Truck Driver
Construction
Police Officer/Correction Officer
Laborer
Civil Service
Engineer - Mechanical
Food Service Management
Engineer - Electrical
Medical Technician
Attorney
Food Service
Military Enlisted
Tradesman - Mechanic
Social Worker
Postal Service
Professor
Nurse (LPN)
Doctor
Tradesman - Electrician
Nurse's Aide
Fireman
Waiter/Waitress
Scientist
Principal
Realtor
Military Officer
Bus Driver
Pharmacist
Investor
Teacher's Aide
Engineer - Chemical
Landscaping
Clergy
Pilot - Private/Commercial
Architect
Car Dealer
Psychologist
Student - College Graduate Student
Chemist
Biologist
Religious
Flight Attendant
Tradesman - Carpenter
Tradesman - Plumber
Student - College Senior
Homemaker
Dentist
Student - College Junior
Judge
Student - College Freshman
Student - College Sophomore
Student - Community College
Student - Technical School
0
2500
5000
7500
10000
12500
15000
17500
20000
Population

The above visualization indicates that the individuals who listed their occupation as `Other` top the list, followed by those the a `Professional` occupation. The borrowers with `Student - Technical School` as their occupation are the fewest. In fact, the data indicates that they were only 2 of them.

To get an even better insight about the number of borrowers who listed a specific occupation, let us get rid of those who listed their occupation as `Other` from the visualization.

[33]:
```
# Set the theme of the visualization
sns.set_theme(style="whitegrid")

# Set the size of the visualization
f, ax = plt.subplots(figsize=(9,20))

# Set the color of the visualization
base_color = sns.color_palette()[0]

# Define the order in which the bars will appear in the visualization
occupations_order = clean_loan_df.Occupation.value_counts().index

# Visualize the bar graph.
sns.countplot(data=clean_loan_df[-(clean_loan_df.Occupation == 'Other')], y =␣
↪'Occupation', color = base_color, order=occupations_order);

# Set the labels and plot title
plt.title('A visualization of The Number of Borrowers in Every Listed␣
↪Occupation', size = 14, weight='bold')
plt.xlabel('Population', size = 10, weight='bold')
plt.ylabel('Occupation', size = 10, weight='bold');
```



A visualization of The Number of Borrowers in Every Listed Occupation
Occupation
Population
Other
Professional
Executive
Computer Programmer
Teacher
Analyst
Administrative Assistant
Accountant/CPA
Sales - Commission
Skilled Labor
Nurse (RN)
Clerical
Sales - Retail
Retail Management
Truck Driver
Construction
Police Officer/Correction Officer
Laborer
Civil Service
Engineer - Mechanical
Food Service Management
Engineer - Electrical
Medical Technician
Attorney
Food Service
Military Enlisted
Tradesman - Mechanic
Social Worker
Postal Service
Professor
Nurse (LPN)
Doctor
Tradesman - Electrician
Nurse's Aide
Fireman
Waiter/Waitress
Scientist
Principal
Realtor
Military Officer
Bus Driver
Pharmacist
Investor
Teacher's Aide
Engineer - Chemical
Landscaping
Clergy
Pilot - Private/Commercial
Architect
Car Dealer
Psychologist
Student - College Graduate Student
Chemist
Biologist
Religious
Flight Attendant
Tradesman - Carpenter
Tradesman - Plumber
Student - College Senior
Homemaker
Dentist
Student - College Junior
Judge
Student - College Freshman
Student - College Sophomore
Student - Community College
Student - Technical School
0
2000
4000
6000
8000
10000

The visualization above clearly indicates that after eliminating the Borrowers who listed their occupation as `Other`, the number of Borrowers who have `Professional` occupations is disproportionately high.

Question 3: How many borrowers fall in every employment status category?

[34]:
```
# Display the number of borrowers in every category
clean_loan_df.EmploymentStatus.value_counts()
```

[34]:
```
Employed         67309
Full-time         7916
Self-employed     4536
Other             2474
Not employed       649
Retired            367
Part-time          256
Name: EmploymentStatus, dtype: int64
```

Notice that in the employment status column, there are borrowers who listed their status as `Employed - (67309)`, `Full-time - (7916)` and `Part-time - (256)`. Here, one concern is that the borrowers who listed their employment status as `employed` are either part-time or full-time employees. However, there is not means of telling in which category each one of them is. Therefore, it would make more sense if all the three employment status categories were collapsed into one category that only indicates whether the employee is employed, dabbed `Employed`.

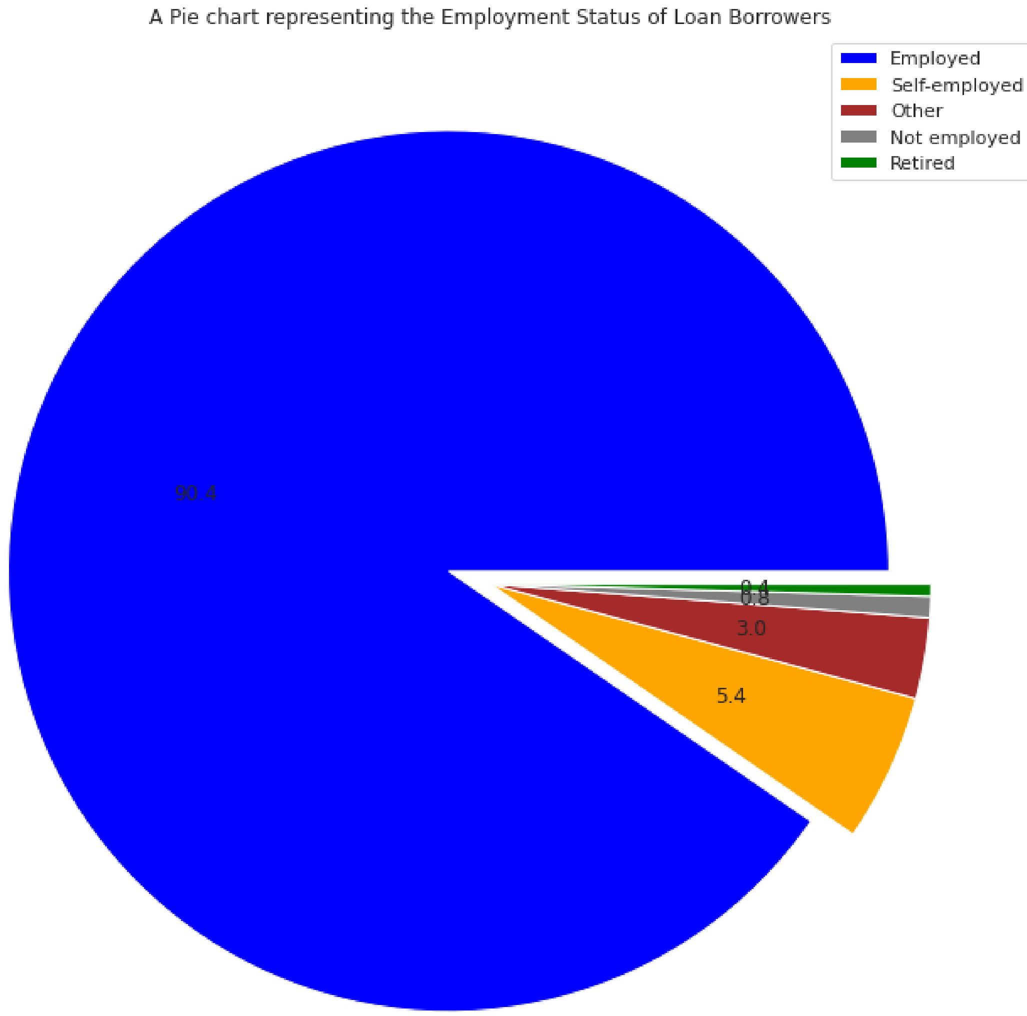
[35]:
```
# Edit the 'full-time' and 'part-time' entries in the EmploymentStatus column␣
↪to simply 'employed'
clean_loan_df.EmploymentStatus.replace(['Full-time', 'Part-time'], 'Employed',␣
↪inplace=True)
```

[36]:
```
# Verify whether the entries have been updated successfully.

# If the entry replacement was successful, then, the test below will pass␣
↪without an assertion error.
assert 'Part-time' and 'Full-time' not in clean_loan_df.EmploymentStatus
```

Now, since there are only five categories to draw insights from, we can use a pie chart to visualize the data

[37]:
```
# use a pie chart to visualize the various employment status categories.
e_status_sorted = clean_loan_df.EmploymentStatus.value_counts()

pie_colors = ['blue', 'orange', 'brown', 'gray', 'green']
f, ax = plt.subplots(figsize=(12,12))
plt.pie(e_status_sorted, explode=(0.1, 0, 0, 0, 0),colors=pie_colors, autopct =␣
↪'%.1f');
```

```
plt.legend(e_status_sorted.index)
plt.title('A Pie chart representing the Employment Status of Loan Borrowers');
```



Insights from the pie chart.

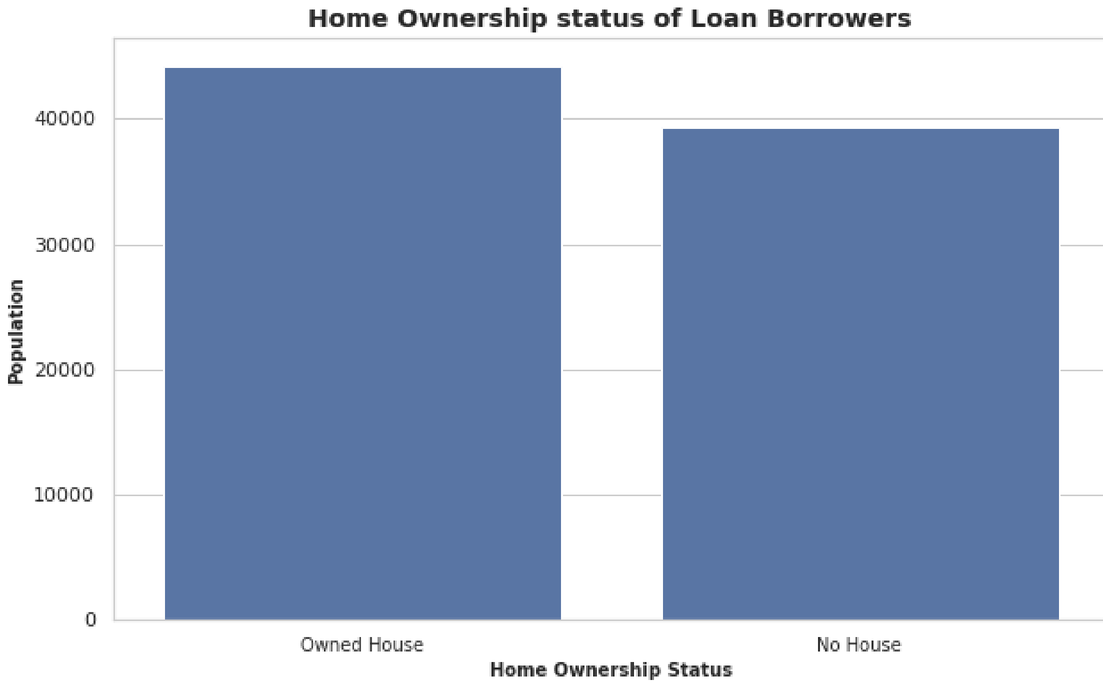
- The pie chart above indicates that the Employed individuals make up more than 90% of the loan borrower population.
- About 5.4% of the Borrowers were self employed.
- About 3.0% of the Borrowers listed they employment status as `other`.
- About 0.8% of the Borrowers were unemployed.
- Only 0.4% of the Borrowers were retired.

Question 4: What proportion of the borrowers owned homes?

[38]:
```
clean_loan_df.IsBorrowerHomeowner.value_counts()
```

[38]:
```
True     44232
False    39275
Name: IsBorrowerHomeowner, dtype: int64
```

[39]:
```
# Set the theme of the visualization
sns.set_theme(style="whitegrid")

# Set the size of the visualization
f, ax = plt.subplots(figsize=(10,6))

# Set the color of the visualization
base_color = sns.color_palette()[0]

# Define the order in which the bars will appear in the visualization
arr_order = clean_loan_df.IsBorrowerHomeowner.value_counts().index

# Visualize the bar graph.
sns.countplot(data=clean_loan_df, x = 'IsBorrowerHomeowner', color =␣
 ↪base_color, order=arr_order);

ax.set_xticklabels(['Owned House', 'No House'], size=10)

# Set the labels and plot title
plt.title('Home Ownership status of Loan Borrowers', size = 14, weight='bold')
plt.xlabel('Home Ownership Status', size = 10, weight='bold')
plt.ylabel('Population', size = 10, weight='bold');
```



Question 5: How are the stated monthly incomes distributed?

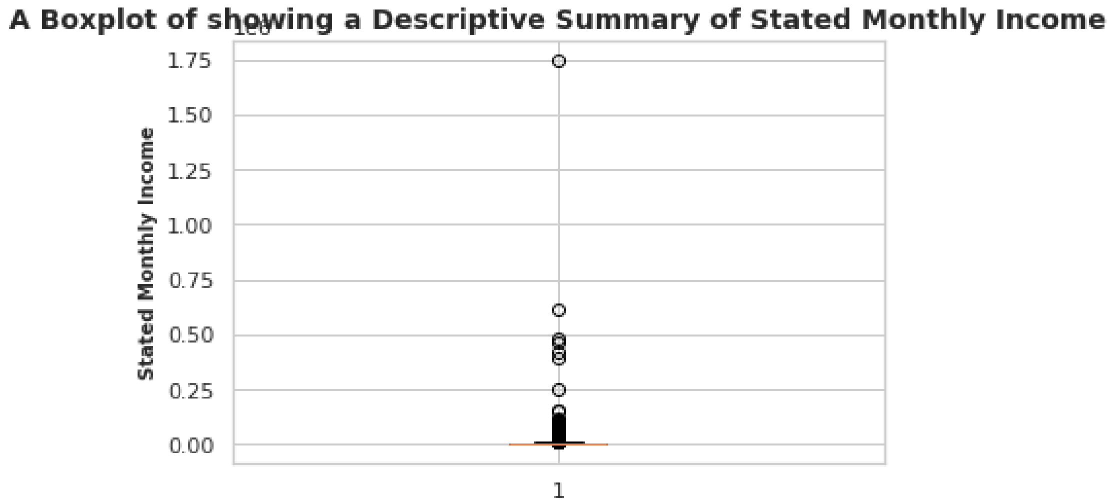
[40]:
```
clean_loan_df.StatedMonthlyIncome.describe()
```

[40]:
```
count    8.350700e+04
mean     5.966762e+03
std      8.297231e+03
min      0.000000e+00
25%      3.500000e+03
50%      5.000000e+03
75%      7.166667e+03
max      1.750003e+06
Name: StatedMonthlyIncome, dtype: float64
```

From the above descriptive summary, the minimum stated monthly income was 0 while the maximum stated monthly income was 1750002.916667. To gain some more insight into how the monthly salaries have been distributed, we generate a box plot to visualize the summary statistics

[41]:
```
# Generate a boxplot to visualize the distribution of the data in the stated␣
↪monthly income column.
monthly_income = clean_loan_df['StatedMonthlyIncome']
fig, ax = plt.subplots()
ax.boxplot(monthly_income)
```

```
plt.title('A Boxplot of showing a Descriptive Summary of Stated Monthly␣
↪Income', size= 14, weight = 'bold')
plt.ylabel('Stated Monthly Income', size = 10, weight = 'bold')
plt.show()
```



The above generated box plot indicates that the data has very many outliers. To make the visualization more insightful, it is necessary to drop the outliers.

Fix the outlier problem In statistics, the max = q75+(1.5*intr_qr) and min = q25-(1.5*intr_qr) where: - Maxim = Maximum value - minim = minimum value - q75 = 75th percentile - q25 = 25th percentile - intr_qr = interquartile range

[42]:
```
# Calculate the lower quartile and the upper quartile values.
q75, q25 = np.percentile(clean_loan_df.loc[:,'StatedMonthlyIncome'],[75,25])

# Calculate the Interquartile Range
intr_qr = q75-q25

# Calculate the Minimum and maximum possible values for the stated monthly␣
↪income entries.
maxim = q75+(1.5*intr_qr)
minim = q75-(1.5*intr_qr)

# Replace the outliers with np.nan
clean_loan_df.loc[clean_loan_df['StatedMonthlyIncome'] <␣
↪minim,'StatedMonthlyIncome'] = np.nan
clean_loan_df.loc[clean_loan_df['StatedMonthlyIncome'] >␣
↪maxim,'StatedMonthlyIncome'] = np.nan
```

[43]:
```
# Check the number of outliers replaced with np.nan in the StatedMonthlyIncome␣
↪column.
clean_loan_df.isnull().sum()
```

[43]:
```
Term                         0
ProsperScore                 0
BorrowerState                0
Occupation                   0
EmploymentStatus             0
EmploymentStatusDuration     0
IsBorrowerHomeowner          0
DelinquenciesLast7Years      0
StatedMonthlyIncome       6738
LoanNumber                   0
LoanOriginalAmount           0
MonthlyLoanPayment           0
LP_CustomerPayments          0
LP_InterestandFees           0
CreditScoreMid_range         0
dtype: int64
```

After the above operation, a total of 6738 outliers have been replaced with np.nan.

To complete the operation of dealing with outliers, we need to drop the null entries from the dataset.

[44]:
```
# Drop the null entries
clean_loan_df.dropna(inplace=True)
```

[45]:
```
# Verify that the Null entries were dropped.
clean_loan_df.isnull().sum()
```

[45]:
```
Term                        0
ProsperScore                0
BorrowerState               0
Occupation                  0
EmploymentStatus            0
EmploymentStatusDuration    0
IsBorrowerHomeowner         0
DelinquenciesLast7Years     0
StatedMonthlyIncome         0
LoanNumber                  0
LoanOriginalAmount          0
MonthlyLoanPayment          0
LP_CustomerPayments         0
LP_InterestandFees          0
CreditScoreMid_range        0
dtype: int64
```

[46]:
```python
# Check the descriptive summary of the cleaned data.
clean_loan_df.StatedMonthlyIncome.describe()
```

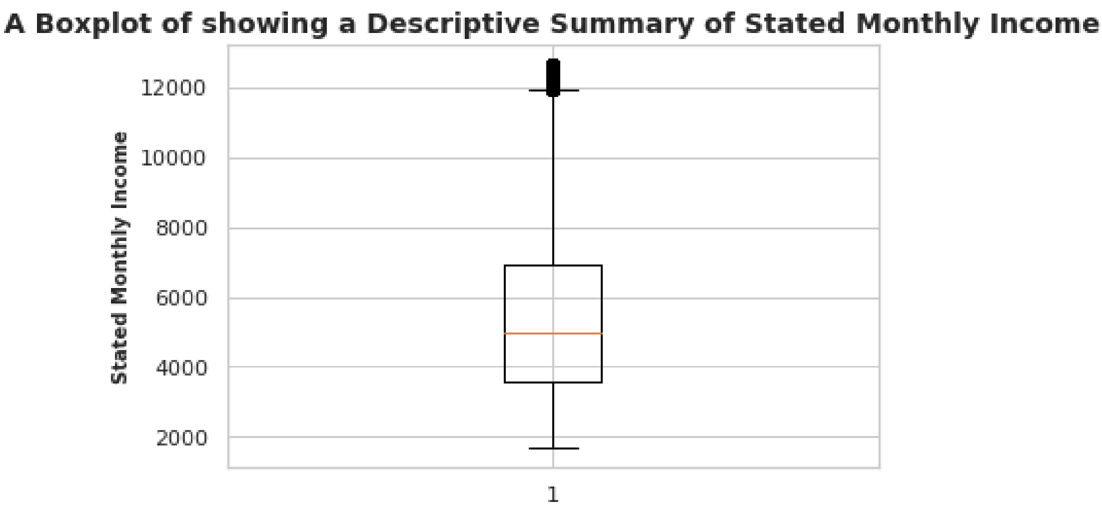
[46]:
```
count    76769.000000
mean      5464.303542
std       2449.339903
min       1666.666667
25%       3583.333333
50%       5000.000000
75%       6916.666667
max      12666.666667
Name: StatedMonthlyIncome, dtype: float64
```

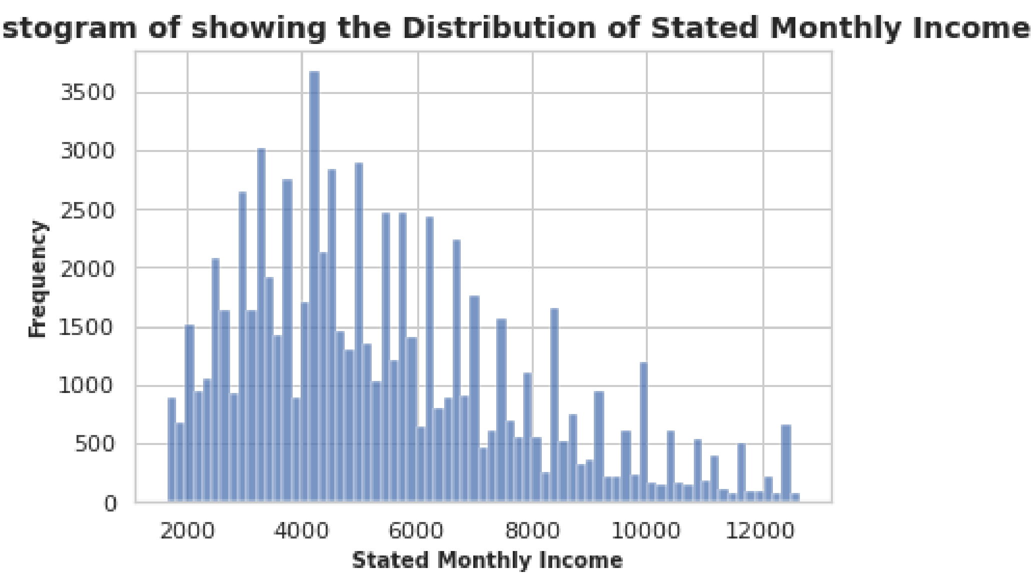
1.5.2 Note that the column headers of the dataframe have been written in camelcase. This means that there will be the need to clean the labels to make them more appealing, that is, adding a space before the capital leters would be elegant. We write a function to do the work.

[47]:
```python
def splitString(string):
    '''
    A function that adds a space before an upper case letter

    Input:
        The raw string.
    Output:
        The string with the spaces after every capital letter.
    '''
    return ''.join([x if x.islower() else f" {x}" for x in string])
```

[48]:
```python
# Now, use the cleaned data to generate a more reasonable histogram.
monthly_income = clean_loan_df['StatedMonthlyIncome']
fig, ax = plt.subplots()
ax.boxplot(monthly_income)
plt.title('A Boxplot of showing a Descriptive Summary of Stated Monthly␣
 ↪Income', size= 14, weight = 'bold')
plt.ylabel('Stated Monthly Income', size = 10, weight = 'bold')
plt.show()

sns.histplot(clean_loan_df, x = 'StatedMonthlyIncome')
plt.title('A Histogram of showing the Distribution of Stated Monthly Income',␣
 ↪size= 14, weight = 'bold')

# Edit the x label
plt.xlabel(splitString('StatedMonthlyIncome'), size=10, weight = 'bold')

# Edit the Y label
plt.ylabel('Frequency', size=10, weight = 'bold')
```

```
plt.show()
```

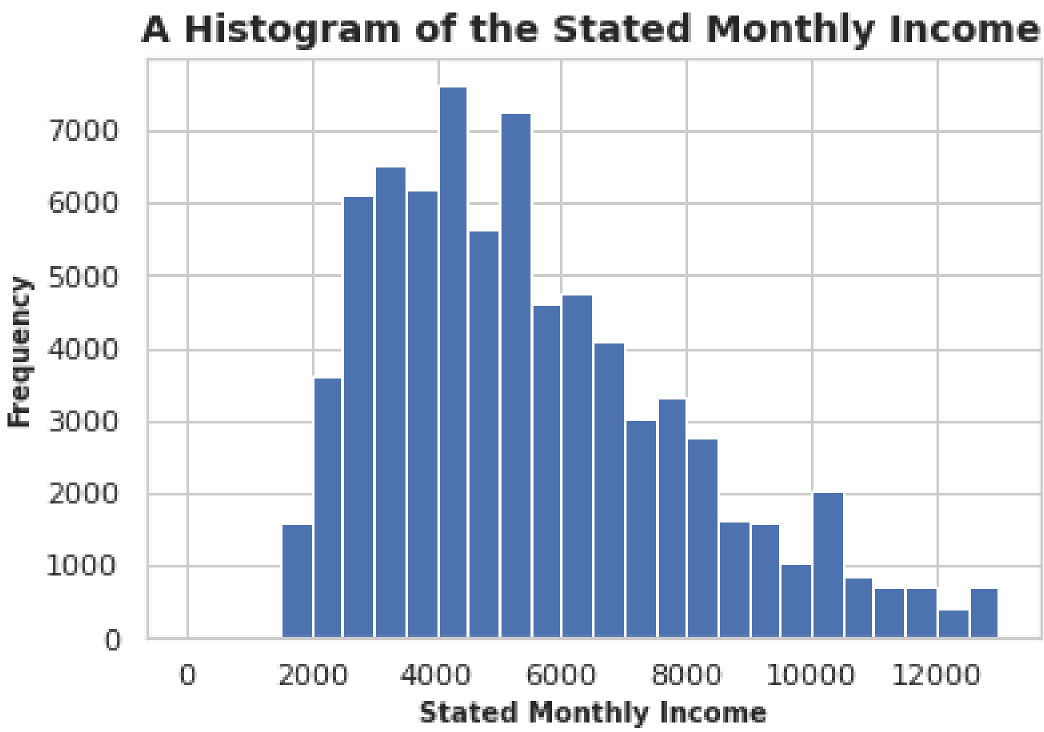




The above visualizations indicates that the distribution of stated montly income is right tailed. The histogram can be refined further by setting the bins size.

[49]:
```
# Set the bins size to refine the histogram.
bins = np.arange(0, clean_loan_df.StatedMonthlyIncome.max()+500, 500)
plt.hist(data = clean_loan_df, x = 'StatedMonthlyIncome', bins=bins);
```

```
plt.title('A Histogram of the Stated Monthly Income', size = 14, weight =␣
 ↪'bold')
plt.xlabel('Stated Monthly Income', size = 10, weight = 'bold')
plt.ylabel('Frequency', size = 10, weight = 'bold');
```



This histogram is more refined as compared to the initial one. Here, the trend of the data is clear, that most of the records are concentrated around between 2000 and 6000, which makes the distribution of the stated monthly incomes right tailed.

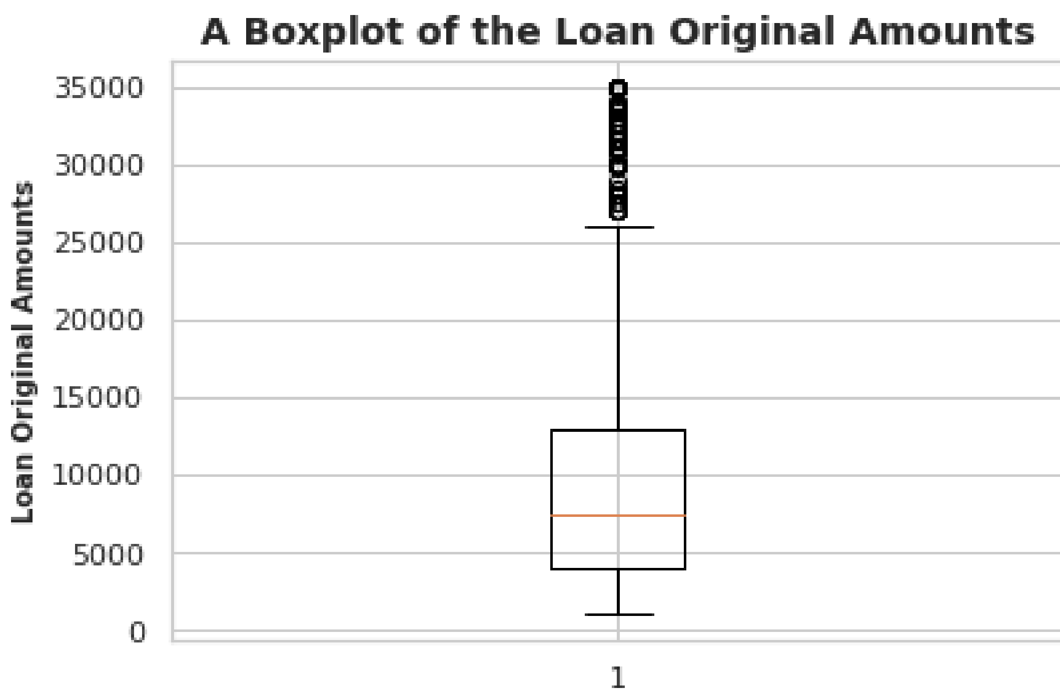
Question 6: How are the Loan Original Amounts distributed?

[50]:
```
clean_loan_df.LoanOriginalAmount.describe()
```

[50]:
```
count    76769.000000
mean      8987.549024
std       6060.323590
min       1000.000000
25%       4000.000000
50%       7500.000000
75%      13000.000000
max      35000.000000
Name: LoanOriginalAmount, dtype: float64
```

From the descriptive statistics above, the minimum original loan amount was 1000 while the maximum loan amount was 35000. The mean of all the loan entries is approximately 8987. To get better insight into these statistics, `we can generate a boxplot to show the 5-number summary. A Violine plot can also suffice.`

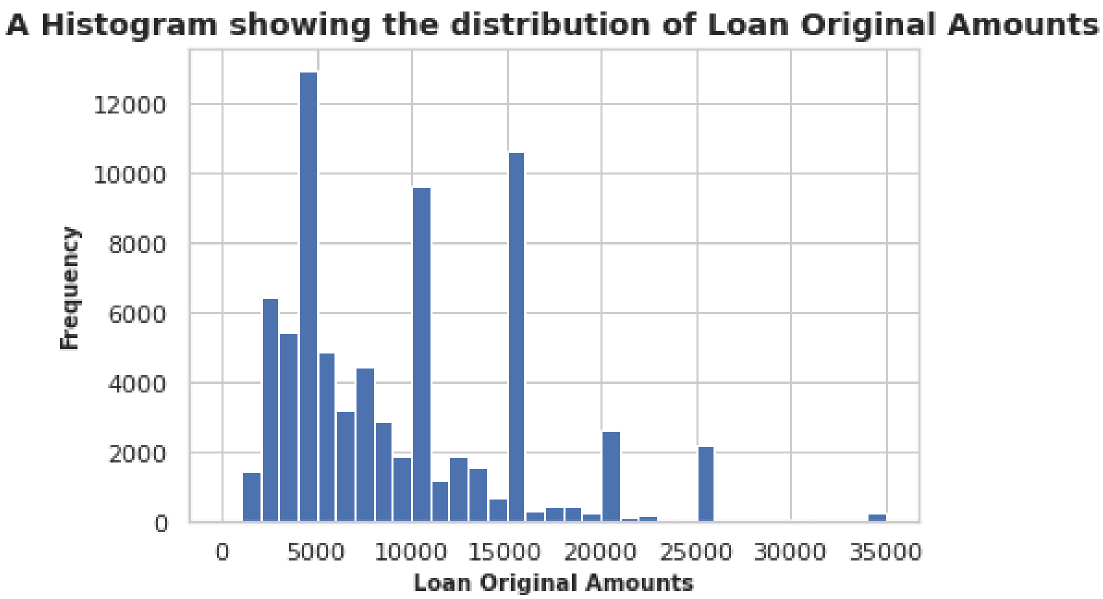
[51]:
```
# Generate a boxplot to visualize the descriptive statistics.
fig, ax = plt.subplots()
ax.boxplot(clean_loan_df.LoanOriginalAmount)
plt.title('A Boxplot of the Loan Original Amounts', size = 14, weight = 'bold')
plt.ylabel('Loan Original Amounts', size = 10, weight = 'bold')
plt.show()
```

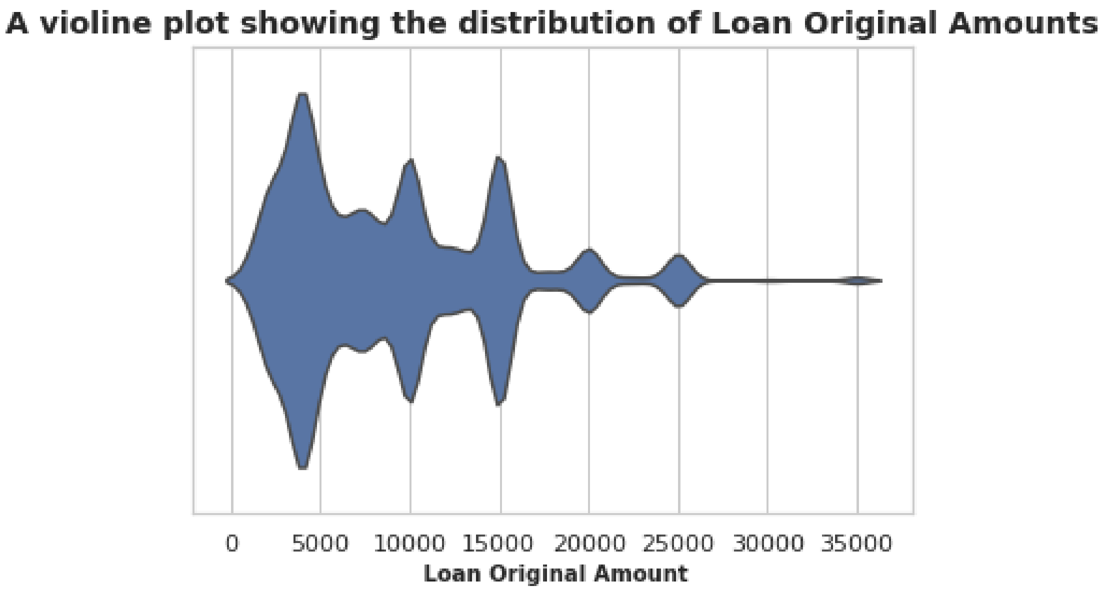


The boxplot above shows that most of the data is concentrated between 4000 and 13000. The visualization also suggests that there are some entries that can be termed as outliers. However, the boxplot fails to clearly show the shape of the distribution. A histogram with a curve that shows the distribution would work well to show how the original loan amount are distributed.

[52]:
```
# Histogram showing the distribution of Loan Original Amounts.
bins = np.arange(0, clean_loan_df.LoanOriginalAmount.max()+1000, 1000)

# plt.subplots(1,2)
# plt.subplot(1,2,1)
plt.hist(data = clean_loan_df, x = 'LoanOriginalAmount', bins=bins);
```

```
plt.title('A Histogram showing the distribution of Loan Original Amounts', size␣
 ↪= 14, weight = 'bold')
plt.xlabel('Loan Original Amounts', size = 10, weight = 'bold')
plt.ylabel('Frequency', size = 10, weight = 'bold');
plt.show()

# Generate a Viloine plot to visualize the descriptive statistics.
# plt.subplot(1,2,2)
sns.violinplot(data=clean_loan_df, x = 'LoanOriginalAmount', inner = None)
plt.title('A violine plot showing the distribution of Loan Original Amounts',␣
 ↪size = 14, weight = 'bold')
plt.xlabel(splitString('LoanOriginalAmount'), size = 10, weight = 'bold')
plt.show()
```





The histogram suggests that the distribution of loan original amounts is right tailed. 35000 appears to be an outlier in the data. To get more insight about this entry, we need to check the data from the dataframe.

[53]:
```
# Isolate the records that contain 35000 in the loan original amount column
df = clean_loan_df[clean_loan_df.LoanOriginalAmount == 35000]
df.head()
```

[53]:

	Term	ProsperScore	BorrowerState	Occupation
29	36	6.0	LA	Other
687	60	10.0	GA	Pilot - Private/Commercial
1223	36	9.0	TX	Computer Programmer
1368	36	10.0	VT	Executive
1422	60	9.0	NC	Executive

	EmploymentStatus	EmploymentStatusDuration	IsBorrowerHomeowner
29	Employed	16.0	True
687	Employed	32.0	True
1223	Employed	0.0	False
1368	Employed	30.0	True
1422	Employed	117.0	True

	DelinquenciesLast7Years	StatedMonthlyIncome	LoanNumber
29	0.0	10416.666667	127405
687	0.0	9166.666667	118574
1223	0.0	8333.333333	104437
1368	0.0	11666.666667	110981
1422	0.0	9166.666667	108299

```
      LoanOriginalAmount  MonthlyLoanPayment  LP_CustomerPayments  \
29                 35000             1169.03            1157.1492
687                35000              814.21            2442.6300
1223               35000             1196.05            5966.8349
1368               35000             1162.33            3475.4927
1422               35000              836.33            3330.7447

      LP_InterestandFees  CreditScoreMid_range
29              356.4292                 749.5
687            1219.6000                 769.5
1223           1921.9449                 749.5
1368           1021.3127                 749.5
1422           1719.7847                 749.5
```

[54]:

```
# Assess the isolated records to see the occupation of the employees who got a␣
↪loan of 35000
df.Occupation.value_counts()
```

[54]:

```
Professional                          47
Other                                 40
Executive                             34
Computer Programmer                   17
Nurse (RN)                            13
Accountant/CPA                        12
Analyst                               11
Police Officer/Correction Officer     10
Engineer - Electrical                  7
Engineer - Mechanical                  7
Construction                           6
Pharmacist                             5
Attorney                               5
Sales - Commission                     5
Pilot - Private/Commercial             4
Teacher                                4
Principal                              3
Professor                              3
Car Dealer                             2
Doctor                                 2
Scientist                              2
Chemist                                2
Skilled Labor                          2
Engineer - Chemical                    1
Food Service Management                1
Retail Management                      1
Tradesman - Electrician                1
Social Worker                          1
Medical Technician                     1
```

```
Dentist                                 1
Fireman                                 1
Military Officer                        1
Sales - Retail                          1
Name: Occupation, dtype: int64
```

The exploration above indicates that majority of the individuals who got a loan of 35000 were `professionals`, `executives`, `computer programmers`, `Registered Nurses`, and `Accountants`.

Question 7: Is there a correlation between the stated monthly salary and montly loan payment?

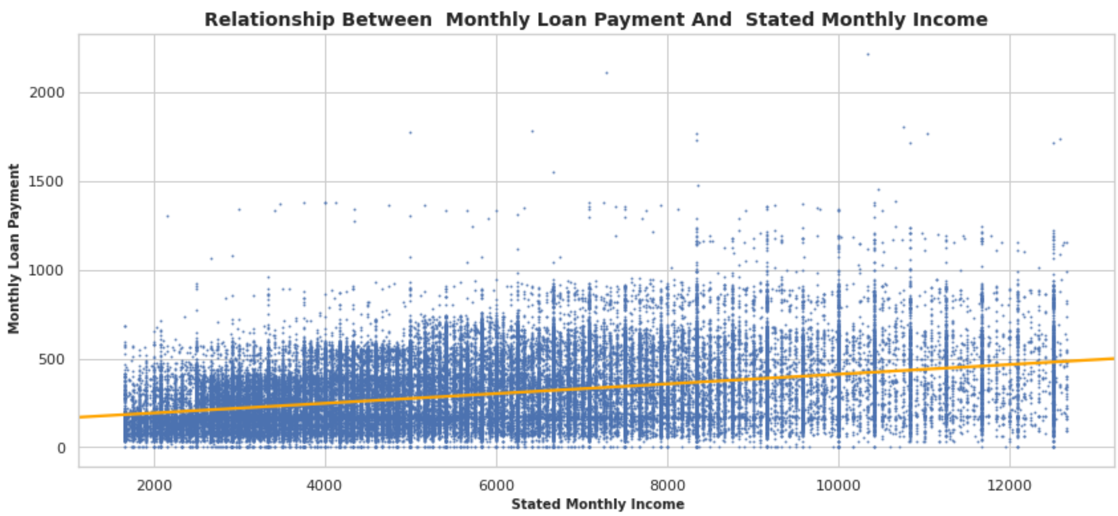
To effectively respond to this question, we generate a scatter plot that explores the relationship between these two variables

Note that there will be several scatter plots that will need to be generated. To avoid repetitive code, it will be more elegant to write a function that will be used to generate the scatter plots and regression lines

[55]:
```
def regPlots (df, xVar, yVar, x_jitter, y_jitter, truncate=False, s=1,␣
↪color='orange'):
    '''
    inputs:
        df -> The dataframe
        xVar -> The x variable
        yVar -> The y variable
    output:
        The regression plot of xVar against yVar.
    '''
    # Specify the dimensions of the plot.
    plt.figure(figsize=[14, 6])
    # The regression plot
    sns.regplot(data=df,
                x=xVar,
                y=yVar,
                truncate=truncate,
                x_jitter=x_jitter,
                y_jitter=y_jitter,
                scatter_kws={'s':s},
                line_kws={'color':color});
    # Tidying the labels
    xVar = splitString(xVar).replace("_"," ")
    yVar = splitString(yVar).replace("_"," ")

    # Add a title
    plt.title(f'Relationship between {yVar} and {xVar}'.title(), fontsize=14,␣
↪weight='bold')
    # Add the x label
```

```
    plt.xlabel(xVar.title(), fontsize = 10, weight = "bold")
    # Add the y label
    plt.ylabel(yVar.title(), fontsize = 10, weight = "bold")
```

[56]:
```
regPlots(clean_loan_df, 'StatedMonthlyIncome', 'MonthlyLoanPayment', 0, 0)
```

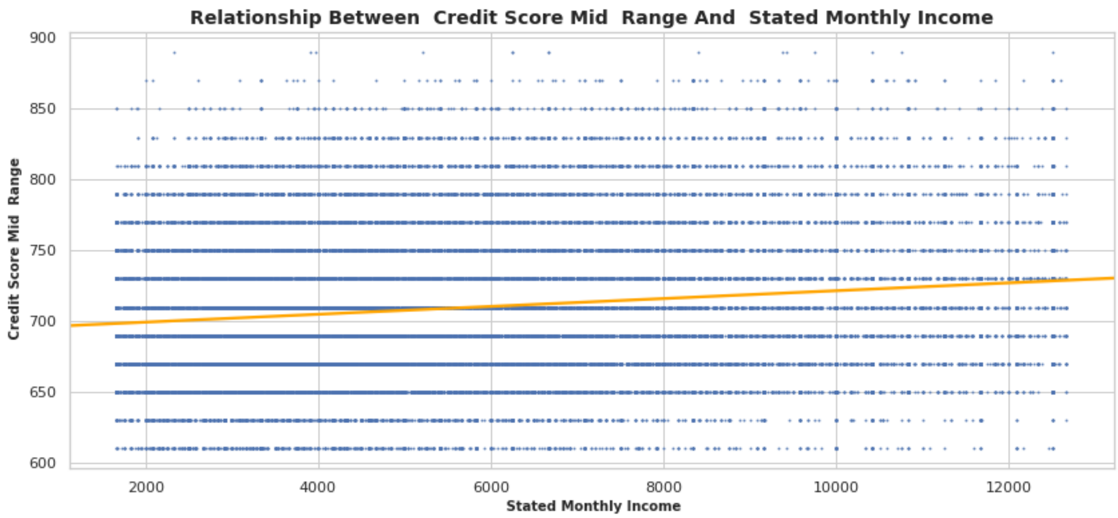


Generally, the regression line indicates that an upward trend, that is, a positive relationship between stated monthly income and montly loan payment.

Question 8: Is there a correlation between the stated monthly salary and the Credit Score Mid_range?

To respond to this question, we need to generate a scatter plot of `stated monthly income` against `Credit Score Midrange`

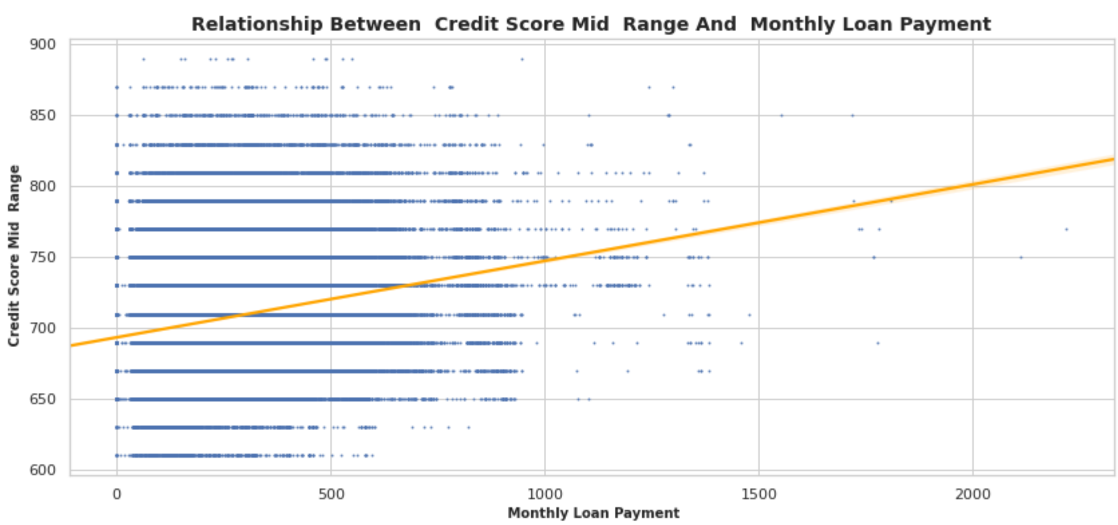
[57]:
```
# A scatter plot of stated monthly income against Credit Score Midrange
regPlots(clean_loan_df, 'StatedMonthlyIncome', 'CreditScoreMid_range', 0, 0)
```



The scatter plot above indicates a positive relationship between the `stated monthly income` and `credit score midrange`. This implies that `borrowers with a higher stated monthly income tend to have a higher credit score mid_range`

Question 9: Is there a correlation between the `Monthly Loan Payment` and the `Credit Score Mid_range`?

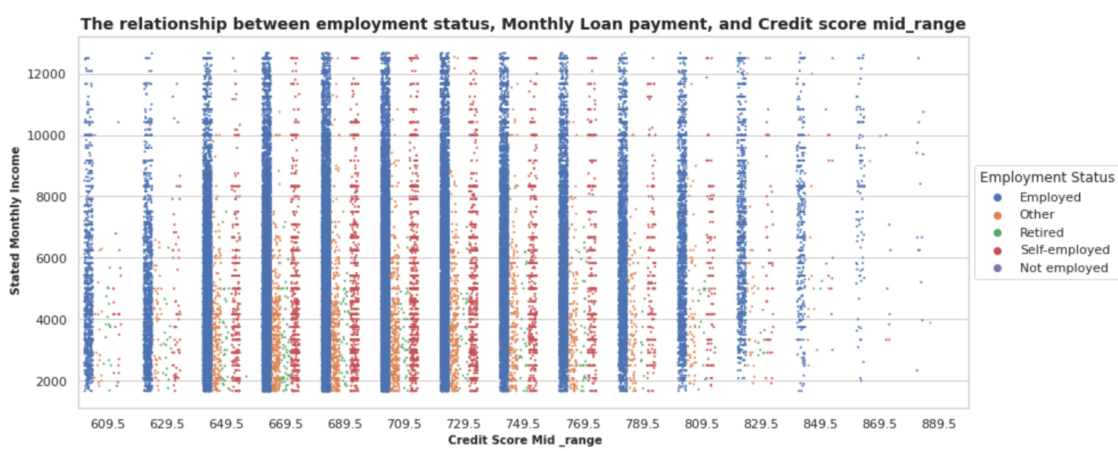
[58]:
```
# A scatter plot of Monthly Loan Payment against Credit Score Midrange
regPlots(clean_loan_df, 'MonthlyLoanPayment', 'CreditScoreMid_range', 0, 0)
```



The scatter plot above indicates a positive relationship between the `Monthly Loan Payment` and `credit score midrange`. This implies that `borrowers with a higher Monthly Loan Payment tend to have a higher credit score mid_range`

Question 10: How does `employment status` play in the relationship between `Monthly Loan Payment` and the `Credit Score Mid_range`?

[59]:
```
plt.figure(figsize = (14,6))
sns.stripplot(data = clean_loan_df,
                  x = 'CreditScoreMid_range',
                  y = 'StatedMonthlyIncome',
                  hue = 'EmploymentStatus',
                  size= 2,
                  jitter = 0.35,
                  dodge=True,
                 )
# g.add_legend();
plt.title('The relationship between employment status, Monthly Loan payment,␣
 ↪and Credit score mid_range',
          fontsize=14,
          weight='bold'
         );
# Edit the legend title
plt.legend(loc='center left', bbox_to_anchor=(1,0.5), title="Employment␣
 ↪Status", title_fontsize = 12);

# Format the xlabels using the splitString function
plt.xlabel(splitString('CreditScoreMid_range'), fontsize=10, weight='bold')

# Format the ylabels using the splitString function
plt.ylabel(splitString('StatedMonthlyIncome'), fontsize=10, weight='bold');
```



1.6 Conclusions

The above Data analysis process has revealed several traits about the loan data.

1.6.1 Findings:

1. About 90.4% of the Borrowers were Employed, 5.4% were self-employed, 3% listed their emloyment status as `Other`, 0.8% were unemployed and 0.4% were retired.
2. About 52% of the borrowers were home owners while about 48% did not own a home.
3. The distribution of montly incomes is right tailed
4. The distribution of Loan original amount is right tailed.
5. There exist a positive correlation between stated monthly income and montly loan payment.
6. There exist a positive correlation between stated monthly income and borrower's credit score.
7. There exist a positive correlation between monthly loan payment and credit score.