In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
import pandas as pd
import numpy as np
```

1. Import Cleaned Data

- Rows: 17,545,457
- Columns: 86

In [3]:
```
hmda21_df = pd.read_csv('C:\Temp\hmda21_cleaned.csv', dtype = str)

hmda21_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26204358 entries, 0 to 26204357
Data columns (total 86 columns):
 #   Column                                    Dtype 
---  ------                                    ----- 
 0   activity_year                             object
 1   lei                                       object
 2   derived_msa_md                            object
 3   state_code                                object
 4   county_code                               object
 5   census_tract                              object
 6   conforming_loan_limit                     object
 7   action_taken                              object
 8   purchaser_type                            object
 9   preapproval                               object
 10  loan_type                                 object
 11  loan_purpose                              object
 12  lien_status                               object
 13  reverse_mortgage                          object
 14  open_end_line_of_credit                   object
 15  business_or_commercial_purpose            object
 16  loan_amount                               object
 17  combined_loan_to_value_ratio              object
 18  interest_rate                             object
 19  rate_spread                               object
 20  hoepa_status                              object
 21  total_loan_costs                          object
 22  total_points_and_fees                     object
 23  origination_charges                       object
 24  discount_points                           object
 25  lender_credits                            object
 26  loan_term                                 object
 27  prepayment_penalty_term                   object
 28  intro_rate_period                         object
 29  negative_amortization                     object
 30  interest_only_payment                     object
 31  balloon_payment                           object
 32  other_nonamortizing_features              object
 33  property_value                            object
 34  construction_method                       object
 35  occupancy_type                            object
 36  manufactured_home_secured_property_type   object
 37  manufactured_home_land_property_interest  object
 38  total_units                               object
 39  multifamily_affordable_units              object
 40  income                                    object
 41  debt_to_income_ratio                      object
 42  applicant_credit_score_type               object
 43  co_applicant_credit_score_type            object
 44  applicant_ethnicity_1                     object
 45  co_applicant_ethnicity_1                  object
 46  applicant_ethnicity_observed              object
 47  co_applicant_ethnicity_observed           object
 48  applicant_race_1                          object
 49  co_applicant_race_1                       object
 50  applicant_race_observed                   object
 51  co_applicant_race_observed                object
 52  applicant_sex                             object
 53  co_applicant_sex                          object
 54  applicant_sex_observed                    object
```

```
 55  co_applicant_sex_observed                 object
 56  applicant_age                             object
 57  co_applicant_age                          object
 58  applicant_age_above_62                    object
 59  co_applicant_age_above_62                 object
 60  submission_of_application                 object
 61  initially_payable_to_institution          object
 62  aus_1                                     object
 63  aus_2                                     object
 64  aus_3                                     object
 65  aus_4                                     object
 66  aus_5                                     object
 67  denial_reason_1                           object
 68  denial_reason_2                           object
 69  denial_reason_3                           object
 70  denial_reason_4                           object
 71  tract_population                          object
 72  tract_minority_population_percent         object
 73  ffiec_msa_md_median_family_income         object
 74  tract_to_msa_income_percentage            object
 75  tract_owner_occupied_units                object
 76  tract_one_to_four_family_homes            object
 77  tract_median_age_of_housing_units         object
 78  state_fips                                object
 79  county_fips                               object
 80  app_race_ethnicity                        object
 81  coapp_race_ethnicity                      object
 82  coapp_same_race                           object
 83  app_credit_model                          object
 84  co_applicant                              object
 85  loan_outcome                              object
dtypes: object(86)
memory usage: 16.8+ GB
```

2. Join with Lender Info

In [4]:
```
lender_def = pd.read_csv('C:\Temp\lender_definitions_20210804.csv', dtype = str)

lender_def.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5508 entries, 0 to 5507
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   lei              5508 non-null   object
 1   agency_code      5508 non-null   object
 2   respondent_name  5508 non-null   object
 3   lar_count        5508 non-null   object
 4   assets           5508 non-null   object
 5   lender_def       5508 non-null   object
 6   lender_size      5508 non-null   object
 7   con_apps         5148 non-null   object
dtypes: object(8)
memory usage: 344.4+ KB
```

In [5]:
```
lender_def2 = lender_def[['lei', 'lar_count', 'assets', 'lender_def', 'con_apps']].cop
lender_def2.head(1)
```

Out[5]:

	lei	lar_count	assets	lender_def	con_apps
0	254900ZBZ4M7TCGJWL09	2	48362	2	1.0

In [6]:
```
hmda21_df = pd.merge(hmda21_df, lender_def2, how = 'left', on = ['lei'])
```

In [7]:
```
hmda21_df['lar_count'].isnull().values.sum()
```

Out[7]:
```
283974
```

In [11]:
```
hmda21_df = hmda21_df[hmda21_df['lar_count'].notna()]
```

In [12]:
```
hmda21_df['lar_count'].isnull().values.sum()
```

Out[12]:
```
0
```

Lender Definition

Only 30,000 records, less than one percent, in overall HMDA data come from no definitions for lenders.

- 1: Banks
- 2: Credit Union
- 3: Independent Mortgage Companies
- 4: No definition

In [14]:
```
print(hmda21_df['lender_def'].value_counts(dropna = False, normalize = True) * 100)
```

```
3    61.334732
1    29.591572
2     8.913788
4     0.159909
Name: lender_def, dtype: float64
```

3. Adding Metro Definitions

In [19]:
```
counties_df = pd.read_csv('C:\Temp\counties_210804.csv', dtype = str)

counties_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3225 entries, 0 to 3224
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   fips_state_code   3224 non-null   object
 1   fips_county_code  3224 non-null   object
 2   county_name       3220 non-null   object
 3   state_name        3220 non-null   object
 4   cbsa_code         1916 non-null   object
 5   cbsa_title        1916 non-null   object
 6   csa_code          1256 non-null   object
 7   csa_title         1256 non-null   object
 8   metro_type        3225 non-null   object
 9   metro_code        1916 non-null   object
 10  metro_name        1916 non-null   object
 11  metro_pop         1915 non-null   object
 12  metro_percentile  3225 non-null   object
 13  metro_type_def    3225 non-null   object
dtypes: object(14)
memory usage: 352.9+ KB
```

In [20]:
```
counties_df2 = counties_df[['fips_state_code', 'fips_county_code', 'metro_code', 'metr
                            'metro_percentile']].copy()

counties_df2 = counties_df2.rename(columns = {'fips_state_code': 'state_fips',
                                              'fips_county_code': 'county_fips'})

counties_df2.head(1)
```

Out[20]:

	state_fips	county_fips	metro_code	metro_type_def	metro_percentile
0	02	013	NaN	4	000

Metro Percentile Definitions

Majority of applications come from metros in the 80th percentile or larger ones.

- 111: Micro
- 000: No Metro
- 99: 99th percentile
- 9: 90th percentile

In [21]:
```
hmda21_df = pd.merge(hmda21_df, counties_df2, how = 'left', on = ['state_fips', 'count

hmda21_df['metro_percentile'].value_counts(dropna = False, normalize = True) * 100
```

Out[21]:
```
9      35.750493
8      15.832412
99     10.245766
7       9.278223
111     6.095461
6       5.744830
000     4.641883
5       3.751329
4       2.639853
3       2.151326
2       1.666978
1       1.265440
0       0.935958
NaN     0.000046
Name: metro_percentile, dtype: float64
```

4. Add Property Value by County

In [22]:
```
prop_values_df = pd.read_csv('C:\Temp\countryPropertValues23T115616.csv', dtype = str)

prop_values_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3221 entries, 0 to 3220
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   GEO_ID       3221 non-null   object
 1   NAME         3221 non-null   object
 2   B25077_001E  3221 non-null   object
 3   B25077_001M  3221 non-null   object
dtypes: object(4)
memory usage: 100.8+ KB
```

First pass at cleaning median property value data

In [23]:
```
prop_values_df2 = prop_values_df[(prop_values_df['GEO_ID'] != 'id')]

prop_values_df3 = prop_values_df2.rename(columns = {'B25077_001E': 'median_value',
                                                    'B25077_001M': 'median_value_moe'})

prop_values_df3['state_fips'] = prop_values_df3['GEO_ID'].str[9:11]
prop_values_df3['county_fips'] = prop_values_df3['GEO_ID'].str[11:]

prop_values_df4 = prop_values_df3[['state_fips', 'county_fips', 'median_value']].copy()


prop_values_df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3220 entries, 1 to 3220
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   state_fips    3220 non-null   object
 1   county_fips   3220 non-null   object
 2   median_value  3220 non-null   object
dtypes: object(3)
memory usage: 100.6+ KB
```

Convert property value to numeric

- No property value for these two counties

In [24]:
```
prop_values_df4[(prop_values_df4['median_value'] == '-')]
```

Out[24]:

	state_fips	county_fips	median_value
549	15	005	-
2674	48	301	-

In [25]:
```
prop_values_df4.loc[(prop_values_df4['median_value'] != '-'), 'median_prop_value'] = p
prop_values_df4.loc[(prop_values_df4['median_value'] == '-'), 'median_prop_value'] = r
prop_values_df4['median_prop_value'] = pd.to_numeric(prop_values_df4['median_prop_valu

prop_values_df4[(prop_values_df4['median_prop_value'].isnull())]
```

Out[25]:

	state_fips	county_fips	median_value	median_prop_value
549	15	005	-	NaN
2674	48	301	-	NaN

In [26]:
```
hmda21_df = pd.merge(hmda21_df, prop_values_df4, how = 'left', on = ['state_fips', 'co
```

In [27]:
```
hmda21_df.loc[(hmda21_df['property_value'] != 'Exempt'), 'prop_value'] = hmda21_df['pr

hmda21_df.loc[(hmda21_df['property_value'] == 'Exempt'), 'prop_value'] = np.nan

hmda21_df['prop_value'] = pd.to_numeric(hmda21_df['prop_value'])
```

5. Add Race and Ethnicity Demographic per Census Tract

In [28]:
```
race_df = pd.read_csv('C:\Temp\demographics_210204.csv',
                      dtype = str)

race_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74001 entries, 0 to 74000
Data columns (total 28 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   year                   74001 non-null  object
 1   state                  74001 non-null  object
 2   county                 74001 non-null  object
 3   tract                  74001 non-null  object
 4   total_estimate         74001 non-null  object
 5   total_moe              74001 non-null  object
 6   white_alone            74001 non-null  object
 7   white_alone_moe        74001 non-null  object
 8   black_alone            74001 non-null  object
 9   black_alone_moe        74001 non-null  object
 10  native_alone           74001 non-null  object
 11  native_alone_moe       74001 non-null  object
 12  asian_alone            74001 non-null  object
 13  asian_moe              74001 non-null  object
 14  pacislander_alone      74001 non-null  object
 15  pacislander_alone_moe  74001 non-null  object
 16  latino_alone           74001 non-null  object
 17  latino_moe             74001 non-null  object
 18  asian_cb               74001 non-null  object
 19  other_cb               74001 non-null  object
 20  white_pct              73298 non-null  object
 21  black_pct              73298 non-null  object
 22  native_pct             73298 non-null  object
 23  asian_pct              73298 non-null  object
 24  pacislander_pct        73298 non-null  object
 25  latino_pct             73298 non-null  object
 26  asiancb_pct            73298 non-null  object
 27  othercb_pct            73298 non-null  object
dtypes: object(28)
memory usage: 15.8+ MB
```

In [29]:
```
race_df['white_pct'] = pd.to_numeric(race_df['white_pct'])

race_df['census_tract'] = race_df['state'] + race_df['county'] + race_df['tract']

race_df2 = race_df[['census_tract', 'total_estimate', 'white_pct', 'black_pct', 'nativ
                    'asian_pct', 'pacislander_pct', 'othercb_pct', 'asiancb_pct']].cop

race_df2.sample(2, random_state = 303)
```

Out[29]:

	census_tract	total_estimate	white_pct	black_pct	native_pct	latino_pct
6006	48479000300	1859	0.376547	0.21516944593867668	0.0	99.40828402366864
23211	39057240100	3026	90.647720	4.758757435558493	0.0	2.4124256444150696

Create White Gradiant

In [30]:
```
race_df2.loc[(race_df2['white_pct'] > 75), 'diverse_def'] = '1'

race_df2.loc[(race_df2['white_pct'] <= 75) & (race_df2['white_pct'] > 50), 'diverse_de
```

```
race_df2.loc[(race_df2['white_pct'] <= 50) & (race_df2['white_pct'] > 25), 'diverse_de

race_df2.loc[(race_df2['white_pct'] <= 25), 'diverse_def'] = '4'

race_df2.loc[(race_df2['white_pct'].isnull()), 'diverse_def'] = '5'

race_df2['diverse_def'].value_counts(dropna = False)
```

Out[30]:
```
1    31608
2    17416
4    13573
3    10701
5      703
Name: diverse_def, dtype: int64
```

- 0: No census data there
- NaN: Records that don't find a match in the census data

In [31]:
```
hmda21_df = pd.merge(hmda21_df, race_df2, how = 'left', on = ['census_tract'])
```

Convert the NaN to 0's

In [32]:
```
hmda21_df.loc[(hmda21_df['diverse_def'].isnull()), 'diverse_def'] = '0'

hmda21_df['diverse_def'].value_counts(dropna = False)
```

Out[32]:
```
1    12167849
2     7139054
3     3678592
4     2601437
0      333022
5         430
Name: diverse_def, dtype: int64
```

7. Clean Debt-to-Income Ratio

In [34]:
```
dti_df = pd.DataFrame(hmda21_df['debt_to_income_ratio'].value_counts(dropna = False)).
         rename(columns = {'index': 'debt_to_income_ratio', 'debt_to_income_ratio': 'c

### Convert the nulls for cleaning purposes
dti_df = dti_df.fillna('null')

dti_df.head(2)
```

Out[34]:

	debt_to_income_ratio	count
0	null	8941730
1	20%-<30%	3568956

In [36]:
```
def setup_dti_cat(row):

    '''
    Setup dti categories based on lenders and CFPB approach to them
    '''
```

```
    dti = row['debt_to_income_ratio']

    healthy = ['<20%', '20%-<30%', '30%-<36%', ]
    manageable = ['36', '37', '38', '39', '40', '41', '42',]
    unmanageable = ['43', '44', '45', '46', '47', '48', '49']
    struggling = ['50%-60%', '>60%']

    if dti in healthy:
        return '1'
    elif dti in manageable:
        return '2'
    elif dti in unmanageable:
        return '3'
    elif dti in struggling:
        return '4'

    elif dti == 'Exempt':
        return '5'
    elif dti == 'null':
        return '6'
```

In [37]:
```
### Running function to organize debt-to-income ratio
dti_df['dti_cat'] = dti_df.apply(setup_dti_cat, axis = 1)

dti_df.head(2)
```

Out[37]:

	debt_to_income_ratio	count	dti_cat
0	null	8941730	6
1	20%-<30%	3568956	1

In [38]:
```
### Drop count column and replace the null values back to NaN
dti_df2 = dti_df.drop(columns = ['count'], axis = 1)
dti_df2 = dti_df2.replace('null', np.nan)

dti_df2.head(2)
```

Out[38]:

	debt_to_income_ratio	dti_cat
0	NaN	6
1	20%-<30%	1

A third of entire dataset is null, when it comes to DTI ratio.

In [39]:
```
hmda21_df = pd.merge(hmda21_df, dti_df2, how = 'left', on = ['debt_to_income_ratio'])

hmda21_df['dti_cat'].value_counts(dropna = False, normalize = True) * 100
```

Out[39]:
```
6    34.496904
1    31.268568
2    14.803315
3    12.223596
4     5.856599
5     1.351018
Name: dti_cat, dtype: float64
```

8. Combine Loan-to-Value Ratio

In [41]:
```
cltv_df = pd.DataFrame(hmda21_df['combined_loan_to_value_ratio'].value_counts(dropna =
          rename(columns = {'index': 'combined_loan_to_value_ratio', 'combined_loan_to

### Convert cltv to numeric
cltv_df.loc[(cltv_df['combined_loan_to_value_ratio'] != 'Exempt'), 'cltv_ratio'] =\
            cltv_df['combined_loan_to_value_ratio']

cltv_df['cltv_ratio'] = pd.to_numeric(cltv_df['cltv_ratio'])
```

Downpayment Flag

- 1: 20 percent or more downpayment
- 2: Less than 20 percent
- 3: Nulls

In [43]:
```
def categorize_cltv(row):

    '''
    Use CLTV to create a downpayment flag

    '''

    cltv = row['cltv_ratio']

    if cltv <= 80:
        return '1'
    elif cltv > 80:
        return '2'

    else:
        return '3'
```

In [45]:
```
cltv_df['downpayment_flag'] = cltv_df.apply(categorize_cltv, axis = 1)
cltv_df2 = cltv_df.drop(columns = ['count', 'cltv_ratio'], axis = 1)


hmda21_df = pd.merge(hmda21_df, cltv_df2, how = 'left', on = ['combined_loan_to_value_
hmda21_df['downpayment_flag'].value_counts(dropna = False)
```

Out[45]:
```
1    11692020
3     9172974
2     5055390
Name: downpayment_flag, dtype: int64
```

9. Property Value Ratio Z-Score

Property value ratios are more normally distributed than raw property values. Because there's they are normally distributed below the 10th ratio, I will use the z-scores and place them into buckets based on those z-scores.

In [46]:
```
property_value_df = pd.DataFrame(hmda21_df.groupby(by = ['state_fips', 'county_fips',
                    'prop_value', 'median_prop_value'], dropna = False).size()).reset_
                     rename(columns = {0: 'count'})
```

In [48]:
```
def calculate_prop_zscore(row):

    '''
    Calculate property z scores
    Mean and standard deviation are based on ratio below 10
    '''



    ### This numbers come from 1_property_value_analysis Jupyter Notebook
    standard_deviation = 0.907
    mean = 1.475

    prop_value_ratio = row['property_value_ratio']

    z_score = ((prop_value_ratio - mean)/standard_deviation)

    return z_score
```

In [49]:
```
def categorize_property_value_ratio(row):

    '''
    Creating categories based on z-scores
    '''

    prop_value_ratio = row['property_value_ratio']


    ### More than one negative standard deviation
    if prop_value_ratio >= 0.009 and prop_value_ratio <= 0.567:
        return '1'

    ### Between negative one standatd deviation and zero
    elif prop_value_ratio >= 0.568 and prop_value_ratio <= 1.474:
        return '2'

    ### Between zero and one standard deviation
    elif prop_value_ratio >= 1.475 and prop_value_ratio <= 2.381:
        return '3'

    ### Between one standard deviation and two standard deviations
    elif prop_value_ratio >= 2.382 and prop_value_ratio <= 3.288:
        return '4'

    ### Greater than two standard deviations but less than 10 for the property value r
    elif prop_value_ratio >= 3.289 and prop_value_ratio < 10:
        return '5'

    ### Greater than 10 property value ratio
    elif prop_value_ratio >= 10:
        return '6'

    else:
        return '7'
```

In [50]:
```
property_value_df['property_value_ratio'] = property_value_df['prop_value'].\
                                            div(property_value_df['median_prop_value']
```

```
property_value_df['prop_zscore'] = property_value_df.apply(calculate_prop_zscore, axis

property_value_df['prop_value_cat'] = property_value_df.apply(categorize_property_valu

property_value_df.sample(3, random_state = 303)
```

Out[50]:

	state_fips	county_fips	property_value	prop_value	median_prop_value	count	property_valu
121353	23	031	2145000	2145000.0	252300.0	1	
144850	27	133	265000	265000.0	153000.0	11	
254872	48	023	125000	125000.0	62000.0	6	

In [51]:
```
property_value_df2 = property_value_df[['state_fips', 'county_fips', 'property_value',
                                        'median_prop_value', 'property_value_ratio', '
                                        'prop_value_cat']].copy()
```

In [52]:
```
hmda21_df = pd.merge(hmda21_df, property_value_df2, how = 'left', on = ['state_fips',
                     'property_value', 'median_prop_value'])
```

10. Applicant Age

- 9999: No Co-applicant
- 8888: Not Applicable

In [53]:
```
age_df = pd.DataFrame(hmda21_df['applicant_age'].value_counts(dropna = False)).reset_i
         rename(columns = {'index': 'applicant_age', 'applicant_age': 'count'})
```

In [54]:
```
def categorize_age(row):

    '''
    Creating categories for applicant's age
    '''

    age = row['applicant_age']

    if age == '<25':
        return '1'

    elif age == '25-34':
        return '2'

    elif age == '35-44':
        return '3'

    elif age == '45-54':
        return '4'

    elif age == '55-64':
        return '5'

    elif age == '65-74':
        return '6'
```

```
    elif age == '>74':
        return '7'

    elif age == '8888' or age == '9999':
        return '8'
```

In [55]:
```
age_df['applicant_age_cat'] = age_df.apply(categorize_age, axis = 1)

age_df = age_df.drop(columns = ['count'], axis = 1)
```

Age Categories

- 1: Less than 25
- 2: 25 through 34
- 3: 35 through 44
- 4: 45 through 54
- 5: 55 through 64
- 6: 65 through 74
- 7: Greater than 74
- 8: Not Applicable

In [56]:
```
hmda21_df = pd.merge(hmda21_df, age_df, how = 'left', on = ['applicant_age'])

hmda21_df['applicant_age_cat'].value_counts(dropna = False)
```

Out[56]:
```
3    5906015
4    5266912
2    4193434
5    4177364
8    2481651
6    2474719
7     888340
1     531949
Name: applicant_age_cat, dtype: int64
```

11. Income and Loan Amount Log

In [57]:
```
hmda21_df['income'] = pd.to_numeric(hmda21_df['income'])
hmda21_df['loan_amount'] = pd.to_numeric(hmda21_df['loan_amount'])

hmda21_df['income_log'] = np.log(hmda21_df['income'])
hmda21_df['loan_log'] = np.log(hmda21_df['loan_amount'])
```

```
C:\Users\sumit.kamarajugadda\anaconda3\lib\site-packages\pandas\core\arraylike.py:39
7: RuntimeWarning: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\sumit.kamarajugadda\anaconda3\lib\site-packages\pandas\core\arraylike.py:39
7: RuntimeWarning: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

12. Applicant Sex

- 1: Male

- 2: Female
- 3: Information not provided
- 4: Not Applicable
- 5: No Co-Applicable
- 6: Marked Both

In [58]:
```
sex_df = pd.DataFrame(hmda21_df['applicant_sex'].value_counts(dropna = False)).reset_i
         rename(columns = {'index': 'applicant_sex', 'applicant_sex': 'count'})
```

In [59]:
```
def categorize_sex(row):

    '''
    Categorizing applicant's sex
    '''

    sex = row['applicant_sex']


    if sex == '1':
        return '1'

    elif sex == '2':
        return '2'

    elif sex == '3' or sex == '4':
        return '3'

    elif sex == '6':
        return '6'
```

In [60]:
```
sex_df = sex_df.drop(columns = ['count'], axis = 1)

sex_df['applicant_sex_cat'] = sex_df.apply(categorize_sex, axis = 1)
```

New applicant sex categories

- 1: Male
- 2: Female
- 3: Not applicable
- 4: Makred both sexes

In [61]:
```
hmda21_df = pd.merge(hmda21_df, sex_df, how = 'left', on = ['applicant_sex'])

hmda21_df['applicant_sex_cat'].value_counts(dropna = False)
```

Out[61]:
```
1    14080736
2     7349079
3     4477447
6       13122
Name: applicant_sex_cat, dtype: int64
```

14. Loan Term

```
loanterm_df = pd.DataFrame(hmda21_df['loan_term'].value_counts(dropna = False)).reset_
```

In [63]:
```
                rename(columns = {'index': 'loan_term', 'loan_term': 'count'})

loanterm_df.loc[(loanterm_df['loan_term'] != 'Exempt'), 'em_loan_term'] = loanterm_df[

loanterm_df['em_loan_term'] = pd.to_numeric(loanterm_df['em_loan_term'])
```

In [65]:
```
def categorize_loan_term(row):

    '''
    Categorize loan term into more or less than 30 years or exactly 30 years
    '''

    loan_term = row['em_loan_term']

    ### 30 year mortgage
    if loan_term == 360:
        return '1'
    ### less than 30
    elif loan_term < 360:
        return '2'
    ### More than 30
    elif loan_term > 360:
        return '3'
    ### Exempts and NAs
    else:
        return '4'
```

In [66]:
```
loanterm_df['mortgage_term'] = loanterm_df.apply(categorize_loan_term, axis = 1)

loanterm_df = loanterm_df.drop(columns = ['count', 'em_loan_term'])
```

Mortgage Term

- 1: 30 year mortgage
- 2: Less than 30 years
- 3: More than 30 years
- 4: Not applicable

In [67]:
```
hmda21_df = pd.merge(hmda21_df, loanterm_df, how = 'left', on = ['loan_term'])

hmda21_df['mortgage_term'].value_counts(dropna = False)
```

Out[67]:
```
1    18345419
2     6778222
4      606186
3      190557
Name: mortgage_term, dtype: int64
```

15. Tract MSA Income Percentage

In [68]:
```
tractmsa_income_df = pd.DataFrame(hmda21_df['tract_to_msa_income_percentage'].value_co
                     reset_index().rename(columns = {'index': 'tract_to_msa_income_per
                                                     'tract_to_msa_income_percentage':

tractmsa_income_df['tract_msa_ratio'] = pd.to_numeric(tractmsa_income_df['tract_to_msa
```

In [70]:
```
def categorize_lmi(row):

    '''
    Categorize low-to-moderate income neighborhoods
    '''

    tract_msa_ratio = row['tract_msa_ratio']

    ### Low
    if tract_msa_ratio > 0 and tract_msa_ratio < 50:
        return '1'
    ### Moderate
    elif tract_msa_ratio >= 50 and tract_msa_ratio < 80:
        return '2'
    ### Middle
    elif tract_msa_ratio >= 80 and tract_msa_ratio < 120:
        return '3'
    ### Upper
    elif tract_msa_ratio >= 120:
        return '4'
    ### None
    elif tract_msa_ratio == 0:
        return '5'
```

In [71]:
```
tractmsa_income_df['lmi_def'] = tractmsa_income_df.apply(categorize_lmi, axis = 1)

tractmsa_income_df = tractmsa_income_df.drop(columns = ['count', 'tract_msa_ratio'], a
```

LMI Definition

- 1: Low
- 2: Moderate
- 3: Middle
- 4: Upper
- 5: None

In [72]:
```
hmda21_df = pd.merge(hmda21_df, tractmsa_income_df, how = 'left', on = ['tract_to_msa_

hmda21_df['lmi_def'].value_counts(dropna = False)
```

Out[72]:
```
3    11139226
4    10123389
2     3686496
1      611299
5      359974
Name: lmi_def, dtype: int64
```

16. Filter:

For Conventional and FHA loans that first-lien, one-to-four unit, site built unites for home purchase where the applicant is going to live in that property

In [73]:
```
one_to_four = ['1', '2', '3', '4']

hmda21_df2 = hmda21_df[((hmda21_df['loan_type'] == '1') | (hmda21_df['loan_type'] == '
```

```
                        & (hmda21_df['occupancy_type'] == '1') &\
                          (hmda21_df['total_units'].isin(one_to_four)) &\
                          (hmda21_df['loan_purpose'] == '1') &\
                          (hmda21_df['action_taken'] != '6') &\
                          (hmda21_df['construction_method'] == '1') &\
                          (hmda21_df['lien_status'] == '1') &\
                          (hmda21_df['business_or_commercial_purpose'] != '1')].copy()

print('hmda21_df: ' + str(len(hmda21_df)))
print('hmda21_df2: ' + str(len(hmda21_df2)))
```

```
hmda21_df: 25920384
hmda21_df2: 5329539
```

17. Write new dataframe to CSV

In [77]:
```
hmda21_df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5329539 entries, 2 to 25920383
Columns: 117 entries, activity_year to lmi_def
dtypes: float64(8), int64(1), object(108)
memory usage: 4.7+ GB
```

In [79]:
```
#free up space
del(hmda21_df)
```

In [80]:
```
hmda21_df2.to_csv('C:\Temp\hmda21_categorized.csv', index = False)
```

In [82]:
```
del(hmda21_df2)
```

In [ ]:
```

```