Exploratory Data Analysis (EDA) on Loan application dataset

December 14, 2021

[4]:
```
%%javascript
var nb = IPython.notebook;
var kernel = IPython.notebook.kernel;
var command = "NOTEBOOK_FULL_PATH = '" + nb.notebook_path + "'";
kernel.execute(command);
```

```
<IPython.core.display.Javascript object>
```

[5]:
```
import io
from nbformat import read, NO_CONVERT

with io.open(NOTEBOOK_FULL_PATH.split("/")[-1], 'r', encoding='utf-8') as f:
    nb = read(f, NO_CONVERT)

word_count = 0
for cell in nb.cells:
    if cell.cell_type == "markdown":
        word_count += len(cell['source'].replace('#', '').lstrip().split(' '))
print(f"Word count: {word_count}")
```

```
Word count: 1176
```

[6]:
```
#Importing the necessary module packages for performing the EDA
import pandas as pd
import numpy as np
```

[7]:
```
#Loading our dataset which is in csv format
df = pd.read_csv('loanapp.csv')
```

[8]:
```
#Inspecting the dataset
df.head(100)
```

[8]:
```
   married   race loan_decision  occupancy  loan_amount  applicant_income  \
0     True  white        reject          1          128                74
1    False  white       approve          1          128                84
2     True  white       approve          1           66                36
3     True  white       approve          1          120                59
4    False  white       approve          1          111                63
```

```
..     …      …         …          …          …              …
95   True   black    approve          1        114             70
96  False   white    approve          1        131             68
97  False   white     reject          1        320              0
98   True  hispan    approve          1        192            105
99  False   white    approve          1        101             52

    num_units  num_dependants  self_employed  monthly_income  purchase_price  \
0         1.0             1.0          False            4583           160.0
1         1.0             0.0          False            2666           143.0
2         1.0             0.0           True            3000           110.0
3         1.0             0.0          False            2583           134.0
4         1.0             0.0          False            2208           138.0
..        …              …              …              …              …
95        1.0             0.0          False            3500           131.0
96        1.0             1.0          False            5667           145.0
97        1.0             3.0          False           12500           630.0
98        1.0             2.0          False            8750           225.0
99        1.0             0.0          False            2493           107.0

    liquid_assets  mortage_payment_history  consumer_credit_history  \
0           52.00                        2                        2
1           37.00                        2                        2
2           19.00                        2                        6
3           31.00                        2                        1
4          169.00                        2                        6
..            …                        …                        …
95          35.00                        2                        1
96          38.00                        2                        2
97         421.20                        1                        3
98          17.00                        1                        1
99          17.68                        2                        1

    filed_bankruptcy  property_type  gender
0              False              2    male
1              False              2    male
2               True              2    male
3              False              1    male
4              False              2    male
..               …            …      …
95             False              2    male
96             False              1    male
97              True              2    male
98             False              2  female
99             False              1    male

[100 rows x 17 columns]
```

0.1 1. Display descriptive statistics on the dataset.

[9]:
```
df.describe(include = 'all')
```

[9]:
```
        married   race loan_decision    occupancy  loan_amount  \
count      1985   1988          1988  1988.000000  1988.000000
unique        2      3             2          NaN          NaN
top        True  white       approve          NaN          NaN
freq       1308   1680          1744          NaN          NaN
mean        NaN    NaN           NaN     1.031690   143.272636
std         NaN    NaN           NaN     0.191678    80.531470
min         NaN    NaN           NaN     1.000000     2.000000
25%         NaN    NaN           NaN     1.000000   100.000000
50%         NaN    NaN           NaN     1.000000   126.000000
75%         NaN    NaN           NaN     1.000000   165.000000
max         NaN    NaN           NaN     3.000000   980.000000

        applicant_income    num_units  num_dependants self_employed  \
count        1988.000000  1984.000000     1985.000000          1988
unique               NaN          NaN             NaN             2
top                  NaN          NaN             NaN         False
freq                 NaN          NaN             NaN          1731
mean           84.684105     1.122480        0.771285           NaN
std            87.079777     0.437315        1.104464           NaN
min             0.000000     1.000000        0.000000           NaN
25%            48.000000     1.000000        0.000000           NaN
50%            64.000000     1.000000        0.000000           NaN
75%            88.000000     1.000000        1.000000           NaN
max           972.000000     4.000000        8.000000           NaN

        monthly_income  purchase_price   liquid_assets  \
count      1988.000000     1988.000000     1988.000000
unique             NaN             NaN             NaN
top                NaN             NaN             NaN
freq               NaN             NaN             NaN
mean       5195.220825      196.304088     4620.333873
std        5270.360946      128.136030    67142.936043
min           0.000000       25.000000        0.000000
25%        2875.750000      129.000000       20.000000
50%        3812.500000      163.000000       38.000000
75%        5594.500000      225.000000       83.000000
max       81000.000000     1535.000000  1000000.000000

        mortage_payment_history  consumer_credit_history filed_bankruptcy  \
count               1988.000000              1988.000000             1988
unique                      NaN                      NaN                2
top                         NaN                      NaN            False
```

```
freq                        NaN                     NaN          1851
mean                   1.708249                2.110161           NaN
std                    0.555335                1.663256           NaN
min                    1.000000                1.000000           NaN
25%                    1.000000                1.000000           NaN
50%                    2.000000                1.000000           NaN
75%                    2.000000                2.000000           NaN
max                    4.000000                6.000000           NaN

        property_type gender
count     1988.000000   1974
unique            NaN      2
top               NaN   male
freq              NaN   1605
mean         1.861167    NaN
std          0.535448    NaN
min          1.000000    NaN
25%          2.000000    NaN
50%          2.000000    NaN
75%          2.000000    NaN
max          3.000000    NaN
```

0.2 2. Check if any records in the data have any missing values; handle the missing data as appropriate.

0.2.1 Checking if there are any null values

[10]:
```
#Checking for total number of null values in each column
df.isnull().sum()
```

[10]:
```
married                     3
race                        0
loan_decision               0
occupancy                   0
loan_amount                 0
applicant_income            0
num_units                   4
num_dependants              3
self_employed               0
monthly_income              0
purchase_price              0
liquid_assets               0
mortage_payment_history     0
consumer_credit_history     0
filed_bankruptcy            0
property_type               0
gender                     14
```

```
dtype: int64
```

0.2.2 Handling null values

Replacing the missing values using mode of the categorical variable we ensure the central tendency of the dataset is not disturbed, since the measure for central tendency for cactegorical variable is mode we use it to replace null values.

[11]:
```python
df['married'].fillna(df.married.mode()[0],inplace=True)
df['num_dependants'].fillna(df.num_dependants.mode()[0],inplace=True)
df['num_units'].fillna(df.num_units.mode()[0],inplace=True)
df['gender'].fillna(df.gender.mode()[0],inplace=True)
```

[12]:
```python
#Test to check the total null values after replacing the missing values
df.isnull().sum()
```

[12]:
```
married                    0
race                       0
loan_decision              0
occupancy                  0
loan_amount                0
applicant_income           0
num_units                  0
num_dependants             0
self_employed              0
monthly_income             0
purchase_price             0
liquid_assets              0
mortage_payment_history    0
consumer_credit_history    0
filed_bankruptcy           0
property_type              0
gender                     0
dtype: int64
```

[13]:
```python
#Check for duplicate rows in our dataset
df.duplicated().sum()
```

[13]:
```
1
```

[14]:
```python
print('Shape of dataframe before dropping duplicates' + str(df.shape))
df = df.drop_duplicates()
print('Share of dataframe after dropping duplicate' + str(df.shape))
```

```
Shape of dataframe before dropping duplicates(1988, 17)
Share of dataframe after dropping duplicate(1987, 17)
```

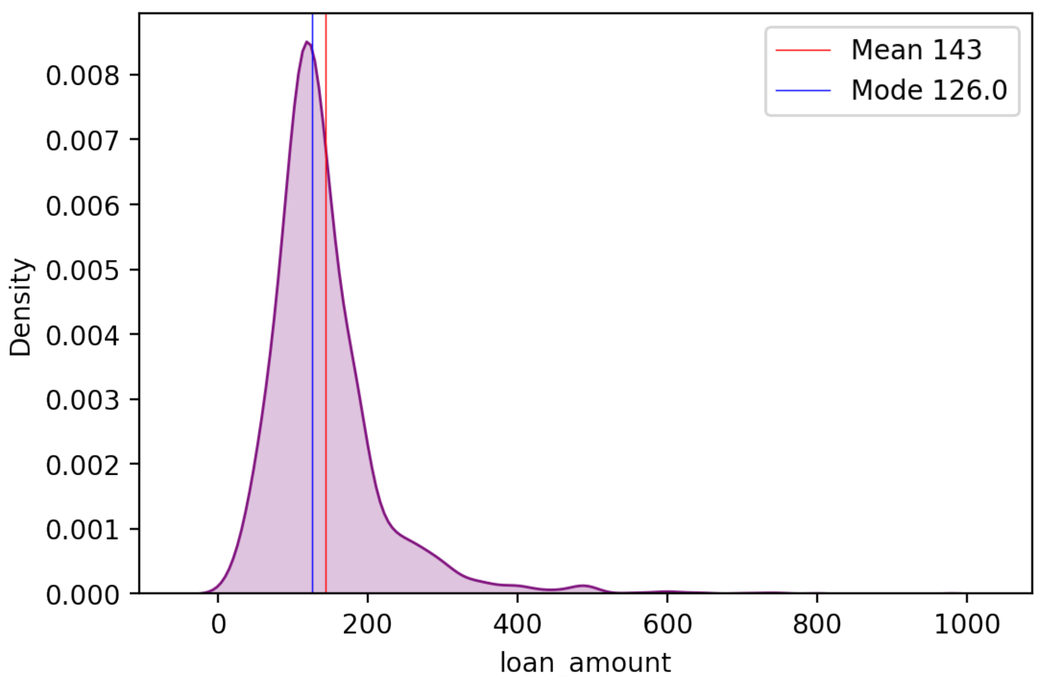
0.3 3. Build a graph visualizing the distribution of one or more individual continuous variables of the dataset

[15]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

0.3.1 3.1 Kernel Density Plot - distribution of data over a continuous interval

[16]:
```python
#Importing figure to control the dimensions of the output graph
from matplotlib.pyplot import figure
#Setting figure dimensions
figure(figsize=(6, 4), dpi=200)
#Setting color scheme
sns.set_palette("BuPu_r")
#Calling kdeplot function from seaborn for visualizing loan amount concentration
sns.kdeplot(data = df.loan_amount, shade = True)
#Creating axis line for mean
plt.axvline(x=df.loan_amount.mean(), color='red', linewidth = 0.5, label='Mean␣
↪' + str(round(np.mean(df.loan_amount))))
#Creating axis line for mode
plt.axvline(x=df.loan_amount.median(), color='blue', linewidth = 0.5,␣
↪label='Mode ' + str(np.median(df.loan_amount)))
#Setting location of the legend
plt.legend(loc='upper right')
```

[16]:
```
<matplotlib.legend.Legend at 0x2461b1e7a60>
```



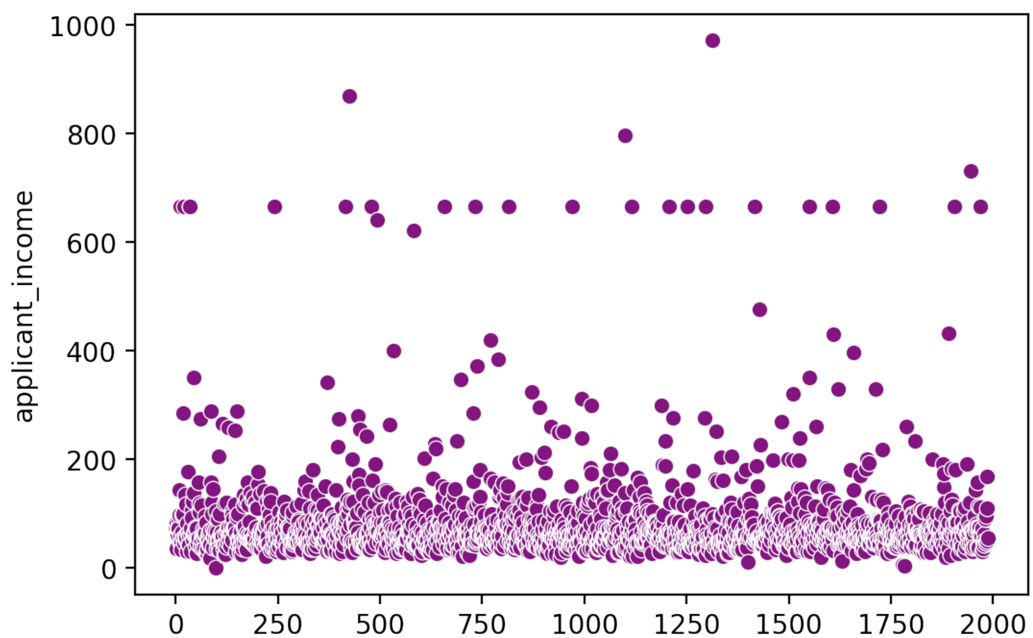
0.3.2 Interpretation

The loan amount is a right skewed distribution and uni modal, we can also see that there is a concentration of the data near the mode of the dataset that is 126.0 thousands of dollars

0.3.3 3.2 Scatterplot visualization - for understanding the spread of the data

[17]:
```
figure(figsize=(6, 4), dpi=200)
#Calling the scatterplot function from seaborn for visualizing applicant income␣
↪spread
sns.scatterplot(x = df.index, y = 'applicant_income', data = df)
```

[17]: <AxesSubplot:ylabel='applicant_income'>



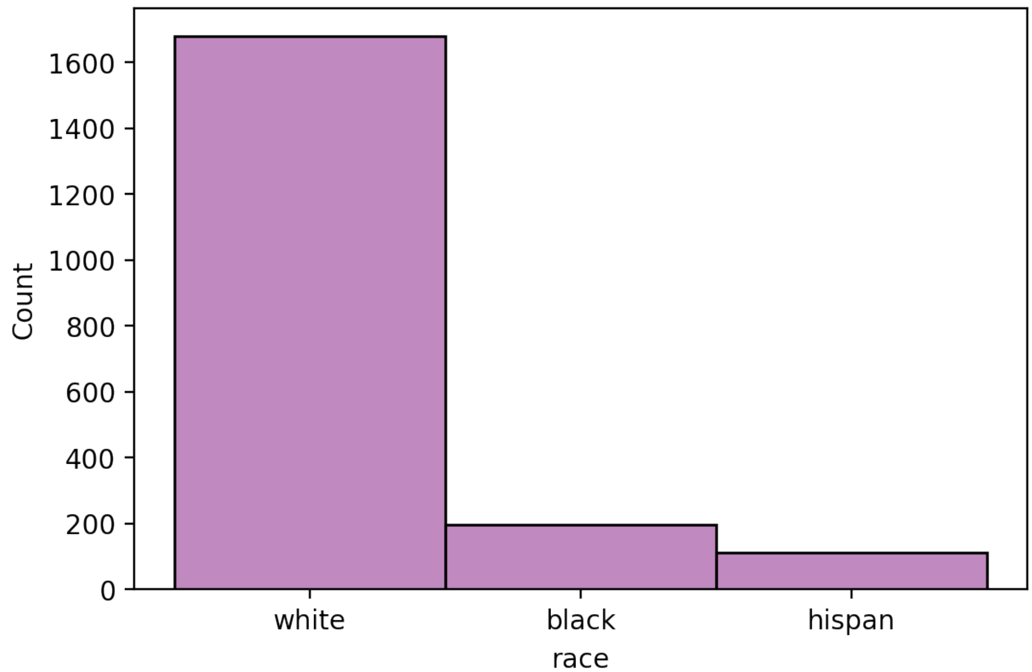
0.3.4 Interpretation

This visualization gives a key insight into the applicants looking for loan, it can be easily interpretted that majority of the applications have their income in the range of 0 - 200 thousands of dollar, and we can also see some more concentration between the range 600 - 800 thousands of dollar. We can also safely assume the applicant incomes between 400 - 600 thousands of dollar are less

0.3.5 3.2 Histogram - visually summarize concentration of categorical variables

[18]:
```
figure(figsize=(6, 4), dpi=200)
#Setting the color sche,e for the histplot
sns.set_palette("BuPu_r")
#Calling histplot function from seaborn to visualize frequency of race
sns.histplot(df, x = 'race', alpha = 0.5)
```

[18]:
```
<AxesSubplot:xlabel='race', ylabel='Count'>
```



0.3.6 Interpretation

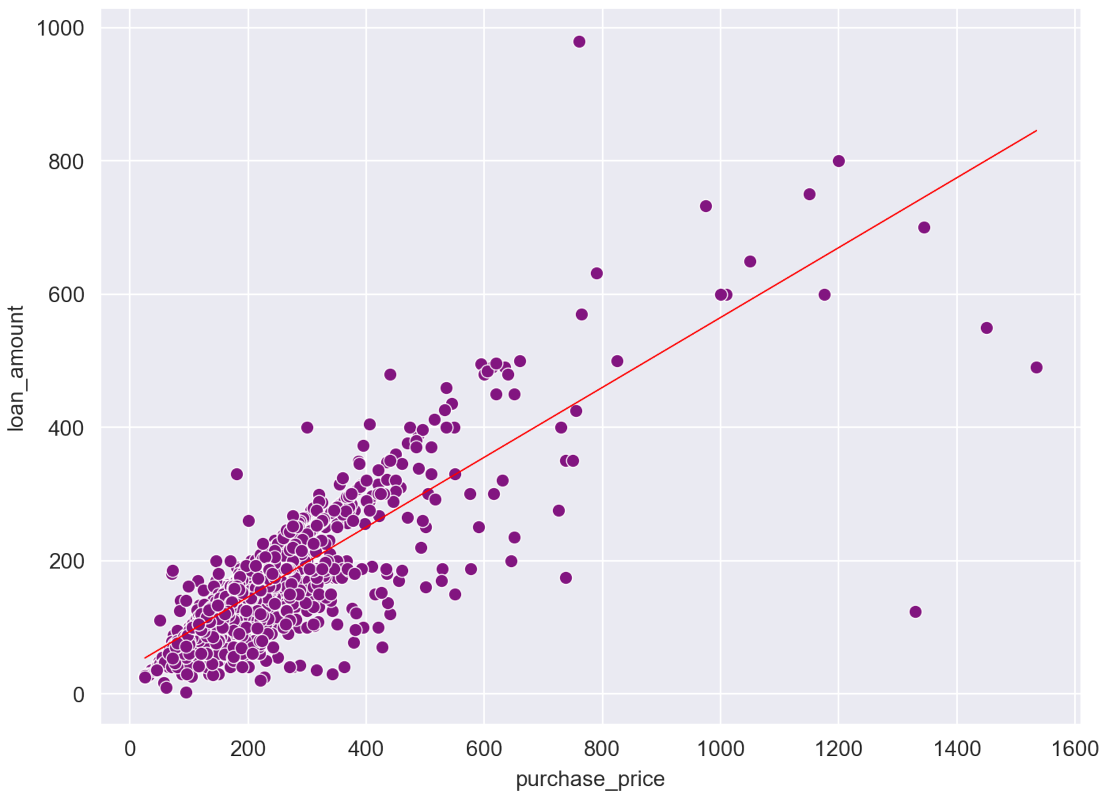
From the above histogram the understanding is that the white race is dominating in the sample dataset followed by black and then hispan

0.4 4. Build a graph visualizing the relationship in a pair of continuous variables. Determine the correlation between them.

0.4.1 4.1 Scatter plot for visualizing pair of continuous variables

[19]:
```
figure(figsize=(8, 6), dpi=200)
sns.set_style("darkgrid")
sns.set_palette("BuPu_r")
#Calling scatterplot function to visualize dependent variable loan amount with␣
↪independent variable purchase price
```

```
sns.scatterplot(data=df, x='purchase_price', y='loan_amount')
#finding the slope and intercept of the best fit line
m, b = np.polyfit(df['purchase_price'], df['loan_amount'], 1)
#Using plot function to plot the straght line from calculated slope and␣
↪intercept values
plt.plot(df['purchase_price'], m*df['purchase_price'] + b, color = 'red',␣
↪linewidth = 0.5)
plt.show()
```



0.4.2 Interpretation

By looking at the scatterplot it is evident that there seems to be some sort of linear relationship between the 2 selected continuous variables, further a best fit line is fitted to confirm this. For purchase price and loan amount requested the concentration of the scatter is found more in the range of 0 to 600 thousands of dollars.

0.4.3 4.2 Correlation coefficient calculation

[20]:
```
#Function to check correlation coefficient using pearson method
correlation_val = df['purchase_price'].corr(df['loan_amount'], method =␣
↪'pearson')
```

```
print(f'The correlation between purchase price and the loan amount is␣
↪{correlation_val}')

#Creating function to determine type of relation based on correlation value
def significance_checker(corr_val):
    #conditional logic to verify siginficant correlation
    if (0 < corr_val < 0.7 ) or ( -0.7 < corr_val < 0):
        print("This is not a significant correlation")
    else:
        print('This is a significant correlation')

#Calliing the created function
significance_checker(correlation_val)
```

```
The correlation between purchase price and the loan amount is 0.8344298313626284
This is a significant correlation
```

0.5 5. Display unique values of a categorical variable.

[21]:
```
#Creation of a function to extract the unique values by passing 2 parameters␣
↪that is dataframe and the column
def unique_extractor(dataframe, column):
    #function to extract unique values of categorical variable
    unique_vals = dataframe[column].unique()
    print(f'The unique values in the column - {column} are {unique_vals}')

#Calling the function to display the unique values of the column
unique_extractor(df, 'property_type')
unique_extractor(df, 'gender')
unique_extractor(df, 'occupancy')
unique_extractor(df, 'race')
```

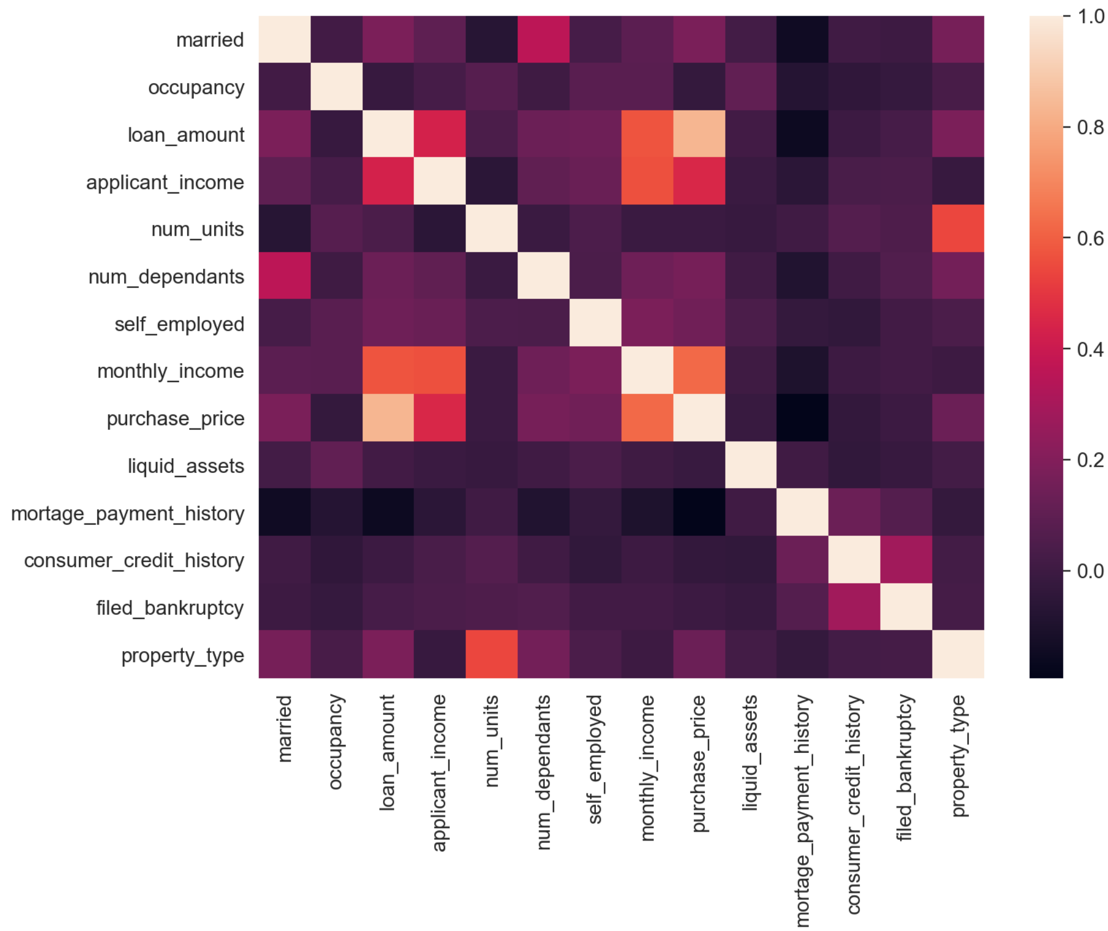
```
The unique values in the column - property_type are [2 1 3]
The unique values in the column - gender are ['male' 'female']
The unique values in the column - occupancy are [1 2 3]
The unique values in the column - race are ['white' 'black' 'hispan']
```

0.6 6. Build a contingency table of two potentially related categorical variables. Conduct a statistical test of the independence between them.

0.6.1 6.1 Identifying potentially related categorical table

[22]:
```
#using corr() function to create contingency table
correlation_table = df.corr()
figure(figsize=(8, 6), dpi=200)
#Easily identify potentially correlated variables using heatmap
sns.heatmap(correlation_table)
```

[22]: <AxesSubplot:>



0.6.2 Interpretation

We can see spots of light color in betweent the correlation heatmap, we use this insight to identify potentially correlated variables to perform our hypothesis testing

0.6.3 6.1 Contingency table creation

[23]:
```
#Function to create contingency table
def contingency_creator(dataframe_column1, dataframe_column2):
    #Using crosstab function from pandas to output contingency table
    contingency_table = pd.crosstab(dataframe_column1, dataframe_column2)
    return contingency_table

#Calling the contingency table creator to obtain the table output
contingency_table = contingency_creator(df['married'], df['num_dependants'])
contingency_table
```

[23]:
```
num_dependants  0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0
married
False           580   61   20   11    4    0    0    0    0
True            596  256  307  115   27    5    3    1    1
```

[24]:
```
from scipy.stats import chi2_contingency
#Creating a function to test the hypothesis based on the contingency input
def chisquare_tester(dataframe, column1, column2):
    #Creating a contingency table
    intro_cont = 'The contingency table is '
    cont_table = pd.crosstab(dataframe[column1], dataframe[column2])

    #Definition of null hypothesis and alternative hypothesis
    null_hypo = f'The null hypothesis is - no relationship exists between␣
↪{column1} and {column2} column in the population; they are independent'
    alter_hypo = f'The alternative hypothesis is - relationship exists between␣
↪{column1} and {column2} column in the population; they are dependent'

    #Running a chi square test to test the hypothesis of independence between 2␣
↪columns using chi2_contingency function
    chi2, pval, dof, expected = chi2_contingency(cont_table)
    pval_out = f'the pval putput from hypothesis testing is {pval}'
    #Conditional logic to accept or reject null hypothesis
    if pval < 0.05:
        result = f'Since the pval is less than 0.05 we reject the null␣
↪hypothesis, hence we confirm - {alter_hypo}'
    else:
        result = f'Since the pval is more than 0.05 we accept the null␣
↪hypothesis, hence we confirm - {null_hypo}'
    endings = '************************************************************'
    intro = 'Hypothesis testing - Chi-Square test report '
    return endings, intro, intro_cont, cont_table, pval_out, null_hypo,␣
↪alter_hypo, result, endings
```

[25]:
```
#Calling the function by passing dataframe and the 2 columns that needs to be␣
↪tested
chisquare_tester(df, 'married', 'num_dependants')
```

[25]:
```
('************************************************************',
 'Hypothesis testing - Chi-Square test report ',
 'The contingency table is ',
 num_dependants  0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0
 married
 False           580   61   20   11    4    0    0    0    0
 True            596  256  307  115   27    5    3    1    1,
 'the pval putput from hypothesis testing is 4.0590686914667564e-63',
 'The null hypothesis is - no relationship exists between married and
```

```
num_dependants column in the population; they are independent',
 'The alternative hypothesis is - relationship exists between married and
num_dependants column in the population; they are dependent',
 'Since the pval is less than 0.05 we reject the null hypothesis, hence we
confirm - The alternative hypothesis is - relationship exists between married
and num_dependants column in the population; they are dependent',
 '*****************************************************************')
```

0.7 7. Retrieve one or more subset of rows based on two or more criteria and present descriptive statistics on the subset(s).

[26]:
```python
#Creating subset of rows based on the condition that the applicant is black and␣
 ↪is married
df_sub1 = df.loan_amount[(df['loan_decision'] == 'reject') &␣
 ↪(df['property_type'] == 1)]
#Creating subset of rows based on the condition that the applicant is hispanic␣
 ↪and is married
df_sub2 = df.loan_amount[(df['loan_decision'] == 'reject') &␣
 ↪(df['property_type'] == 2)]
#Creating subset of rows based on the condition that the applicant is white and␣
 ↪is married
df_sub3 = df.loan_amount[(df['loan_decision'] == 'reject') &␣
 ↪(df['property_type'] == 3)]
df_sub1.describe()
```

[26]:
```
count     57.000000
mean     111.105263
std       48.308046
min        9.000000
25%       84.000000
50%      105.000000
75%      124.000000
max      300.000000
Name: loan_amount, dtype: float64
```

[27]:
```python
df_sub2.describe()
```

[27]:
```
count    139.000000
mean     146.705036
std       72.098107
min       35.000000
25%      104.000000
50%      128.000000
75%      176.000000
max      495.000000
Name: loan_amount, dtype: float64
```

[28]:
```
df_sub3.describe()
```

[28]:
```
count     48.000000
mean     160.479167
std       71.040890
min       72.000000
25%      123.750000
50%      157.500000
75%      180.250000
max      570.000000
Name: loan_amount, dtype: float64
```

We find that mean is varying heavily between the sub categories

0.8 8. Conduct a statistical test of the significance of the difference between the means of two subsets of the data.

[29]:
```
from scipy.stats import ttest_ind

def twosample_ttester(data1, data2):
    #Defining null and alternative hypothesis
    null_hypo = f'Null hypothesis - The means of both the datasets are equal'
    alter_hypo = f'Alternative hypothesis - The means of both datasets are not␣
 ↪equal'
    #Running two sample t test using ttest_ind function from scipy module to␣
 ↪obtain pval to confirm or reject null hypothesis
    tstat, pval = ttest_ind(data1, data2)
    pval_out = f'the pval putput from hypothesis testing is {pval}'
    if pval < 0.05:
        result = f'Since the pval is less than 0.05 we reject the null␣
 ↪hypothesis, hence we confirm - {alter_hypo}'
    else:
        result = f'Since the pval is more than 0.05 we accept the null␣
 ↪hypothesis, hence we confirm - {null_hypo}'
    endings =␣
 ↪'*******************************************************************'
    intro = 'Hypothesis testing - two sample t-test report '
    return endings, intro, pval_out, null_hypo, alter_hypo, result, endings
```

0.8.1 We test to check if the means of the data set for the condition - [rejection of loan and property type 1] and [rejection of loan and proerty type 2] are same

[30]:
```
twosample_ttester(df_sub1, df_sub2)
```

[30]:
```
('*******************************************************************',
 'Hypothesis testing - two sample t-test report ',
 'the pval putput from hypothesis testing is 0.0007543700321850669',
```

```
 'Null hypothesis - The means of both the datasets are equal',
 'Alternative hypothesis - The means of both datasets are not equal',
 'Since the pval is less than 0.05 we reject the null hypothesis, hence we
confirm - Alternative hypothesis - The means of both datasets are not equal',
 '*****************************************************************')
```

0.8.2 We test to check if the means of the data set for the condition - [rejection of loan and property type 1] and [rejection of loan and proerty type 3] are same

[31]:
```
twosample_ttester(df_sub1, df_sub3)
```

[31]:
```
('*****************************************************************',
 'Hypothesis testing - two sample t-test report ',
 'the pval putput from hypothesis testing is 5.328468590330096e-05',
 'Null hypothesis - The means of both the datasets are equal',
 'Alternative hypothesis - The means of both datasets are not equal',
 'Since the pval is less than 0.05 we reject the null hypothesis, hence we
confirm - Alternative hypothesis - The means of both datasets are not equal',
 '*****************************************************************')
```

0.8.3 We test to check if the means of the data set for the condition - [rejection of loan and property type 2] and [rejection of loan and proerty type 3] are same

[32]:
```
twosample_ttester(df_sub2, df_sub3)
```

[32]:
```
('*****************************************************************',
 'Hypothesis testing - two sample t-test report ',
 'the pval putput from hypothesis testing is 0.2535204579270613',
 'Null hypothesis - The means of both the datasets are equal',
 'Alternative hypothesis - The means of both datasets are not equal',
 'Since the pval is more than 0.05 we accept the null hypothesis, hence we
confirm - Null hypothesis - The means of both the datasets are equal',
 '*****************************************************************')
```

0.9 9. Create one or more tables that group the data by a certain categorical variable and displays summarized information for each group (e.g. the mean or sum within the group).

[33]:
```
df_group1 = df.groupby(['num_dependants']).mean()
df_group1
```

[33]:

num_dependants	married	occupancy	loan_amount	applicant_income	num_units	\
0.0	0.506803	1.028912	134.484694	79.267857	1.125000	
1.0	0.807571	1.037855	149.845426	85.271293	1.110410	
2.0	0.938838	1.039755	156.553517	89.110092	1.131498	
3.0	0.912698	1.031746	170.476190	110.626984	1.119048	

```
4.0                 0.870968   1.000000   139.967742        120.000000   1.064516
5.0                 1.000000   1.000000   140.400000         98.000000   1.000000
6.0                 1.000000   1.000000   300.333333        172.000000   1.333333
7.0                 1.000000   1.000000   300.000000        180.000000   1.000000
8.0                 1.000000   1.000000   120.000000         54.000000   1.000000

                self_employed  monthly_income  purchase_price  liquid_assets  \
num_dependants
0.0                  0.113946     4698.875850      180.018219    4343.663517
1.0                  0.157729     5328.151420      204.605883    6416.680621
2.0                  0.146789     5673.743119      225.898165      83.841792
3.0                  0.150794     7764.500000      239.904262   15963.771675
4.0                  0.129032     5832.354839      202.851613      85.467742
5.0                  0.400000     5493.200000      191.400000     118.718000
6.0                  0.000000    15486.000000      503.000000     181.366667
7.0                  0.000000    15000.000000      430.000000     273.000000
8.0                  0.000000     4513.000000      240.000000      30.000000

                mortage_payment_history  consumer_credit_history  \
num_dependants
0.0                            1.766156                 2.120748
1.0                            1.634069                 2.009464
2.0                            1.571865                 2.085627
3.0                            1.682540                 2.309524
4.0                            1.806452                 2.225806
5.0                            1.400000                 2.600000
6.0                            2.333333                 2.000000
7.0                            2.000000                 1.000000
8.0                            1.000000                 1.000000

                filed_bankruptcy  property_type
num_dependants
0.0                     0.056973       1.786565
1.0                     0.085174       1.911672
2.0                     0.073394       2.009174
3.0                     0.103175       2.015873
4.0                     0.161290       1.935484
5.0                     0.200000       2.000000
6.0                     0.000000       2.333333
7.0                     0.000000       1.000000
8.0                     0.000000       2.000000
```

[34]:

```python
df_group1 = df.groupby(['occupancy']).mean()
df_group1
```

[34]:

```
            married  loan_amount  applicant_income  num_units  num_dependants  \
occupancy
```

```
1           0.657513   143.469430        84.125389   1.115544        0.768394
2           0.784314   138.588235       108.078431   1.392157        0.901961
3           0.333333   121.000000        69.000000   1.000000        0.333333

           self_employed  monthly_income  purchase_price  liquid_assets  \
occupancy
1               0.123834     5103.520725      196.795092    3200.152373
2               0.333333     8764.803922      184.666667   58989.347843
3               0.166667     4330.500000      141.666667      71.153333

           mortage_payment_history  consumer_credit_history  filed_bankruptcy  \
occupancy
1                         1.716580                 2.123316          0.069948
2                         1.372549                 1.686275          0.039216
3                         1.833333                 1.666667          0.000000

           property_type
occupancy
1               1.857513
2               2.000000
3               1.833333
```

0.10 10. Implement a linear regression model and interpret its output.

0.10.1 10.1 Initial model

[35]:
```python
#Creating the initial model with all continuous variables included
import statsmodels.api as sm
#Using original least square method to build the model, find coefficients
model = sm.OLS.from_formula('loan_amount ~ married + applicant_income +␣
↪num_dependants + self_employed + monthly_income + liquid_assets +␣
↪mortage_payment_history + consumer_credit_history + purchase_price +␣
↪filed_bankruptcy + property_type', data=df).fit()
model.summary()
```

[35]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:            loan_amount   R-squared:                       0.711
Model:                            OLS   Adj. R-squared:                  0.710
Method:                 Least Squares   F-statistic:                     442.1
Date:                Tue, 14 Dec 2021   Prob (F-statistic):               0.00
Time:                        17:32:20   Log-Likelihood:                -10306.
No. Observations:                1987   AIC:                         2.064e+04
Df Residuals:                    1975   BIC:                         2.070e+04
Df Model:                          11
Covariance Type:            nonrobust
```

```
==============================================================================
============
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
Intercept                 14.3527      5.163      2.780      0.005       4.228
24.478
married[T.True]            5.8257      2.248      2.592      0.010       1.417
10.234
self_employed[T.True]      2.1000      2.963      0.709      0.479      -3.711
7.911
filed_bankruptcy[T.True]   8.0695      4.027      2.004      0.045       0.173
15.966
applicant_income           0.0474      0.014      3.435      0.001       0.020
0.074
num_dependants            -2.1629      0.960     -2.253      0.024      -4.046
-0.280
monthly_income             0.0011      0.000      4.306      0.000       0.001
0.002
liquid_assets              2.8e-05   1.45e-05      1.926      0.054   -5.05e-07
5.65e-05
mortage_payment_history    0.3679      1.819      0.202      0.840      -3.199
3.935
consumer_credit_history    0.4862      0.618      0.787      0.431      -0.725
1.698
purchase_price             0.4736      0.010     46.286      0.000       0.454
0.494
property_type             11.4340      1.883      6.072      0.000       7.741
15.127
==============================================================================
Omnibus:                      741.103   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            83844.660
Skew:                          -0.738   Prob(JB):                         0.00
Kurtosis:                      34.789   Cond. No.                     3.76e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.76e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

Initial model has variables which are having higher p-value which are - Self_employed, mortage_payment_history, consumer_credit_history, p value higher than the significance threshold of 5% is indicative of insignificance of these variables in the model

0.10.2 10.2 Final model

[36]:
```
import statsmodels.api as sm
model = sm.OLS.from_formula('loan_amount ~ married + applicant_income +␣
 ↪num_dependants + monthly_income + purchase_price + filed_bankruptcy +␣
 ↪property_type', data=df).fit()
model.summary()
```

[36]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:            loan_amount   R-squared:                       0.710
Model:                            OLS   Adj. R-squared:                  0.709
Method:                 Least Squares   F-statistic:                     693.8
Date:                Tue, 14 Dec 2021   Prob (F-statistic):               0.00
Time:                        17:32:20   Log-Likelihood:                -10308.
No. Observations:                1987   AIC:                         2.063e+04
Df Residuals:                    1979   BIC:                         2.068e+04
Df Model:                           7
Covariance Type:            nonrobust
================================================================================
============
                             coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
Intercept                 16.0141      3.792      4.223      0.000       8.577
23.452
married[T.True]            5.8661      2.237      2.622      0.009       1.479
10.253
filed_bankruptcy[T.True]   8.9208      3.858      2.312      0.021       1.354
16.487
applicant_income           0.0479      0.014      3.481      0.001       0.021
0.075
num_dependants            -2.1712      0.960     -2.262      0.024      -4.053
-0.289
monthly_income             0.0012      0.000      4.441      0.000       0.001
0.002
purchase_price             0.4727      0.010     46.805      0.000       0.453
0.493
property_type             11.5894      1.881      6.160      0.000       7.900
15.279
==============================================================================
Omnibus:                      739.418   Durbin-Watson:                   1.980
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            82281.338
Skew:                          -0.739   Prob(JB):                         0.00
```

```
Kurtosis:                      34.491   Cond. No.                     3.16e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 3.16e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

This is the final output model, the adjusted R squared value being 0.709 is of acceptable value and the p-values for the variables are all less than 0.05

0.10.3 10.3 Residual analysis for model adequacy

[37]:
```
from bokeh.io import output_notebook
output_notebook()
from bokeh.plotting import figure
from bokeh.io import show

fig = figure(height = 400, width = 400)
st_resids = model.get_influence().resid_studentized_internal
fig.circle(model.fittedvalues, st_resids)
show(fig)
```

[38]:
```
hist, edges = np.histogram(st_resids, bins=20)
fig = figure(height=400, width=400)
fig.quad(top=hist, bottom=0, left=edges[:-1], right=edges[1:],␣
 ↪line_color="white")
show(fig)
```

0.10.4 Analysis of adequacy of the model

Visually analyzing the scatterplot and histogram obtained we can say that the residuals are scattered around 0 and the residuals are also normally distributed. However Jarque Berra test seems to be failing as we are made to accept the null hypothesis that the data is normally distributed.

0.10.5 10.4 Loan amount predictor

[39]:
```
#Creating a function which when passed with coefficients from the regression␣
 ↪model shall predict the output value
def loan_amount_predictor(married, filed_bankruptcy, applicant_income,␣
 ↪num_dependants, monthly_income, purchase_price, property_type):
    #Assigning the coefficient value for each variable and formulating the␣
 ↪equation
```

```
    loan_amount_output = 16.0187 + 5.8556*married + 8.9170*filed_bankruptcy + 0.
↪0479*applicant_income - 2.1721*num_dependants + 0.0012*monthly_income + 0.
↪4727*purchase_price + 11.5932*property_type
    return loan_amount_output

loan_amount_predictor(1, 1, 72, 1, 3234, 99.7, 2)
```

[39]: 106.26338999999999

The actual value is 100, obtained value is 106.2633. Difference is less

[40]:
```
loan_amount_predictor(1, 0, 285, 1, 12841, 387, 2)
```

[40]: 254.8842

The actual value is 349, obtained output as 254, there is significance difference found at higher ranges of data

[41]:
```
loan_amount_predictor(1, 0, 74, 1, 3717, 158, 3)
```

[41]: 137.17340000000002

The actual value is 125, obtained value is 137.17. Difference is less

1 11 Conclusion

Univariate analysis we found that major applications were being recieved for the price range of around 150 thousand dollars, which almost matched the applicant income range as we saw in the second visualization. We also saw in the univariate analysis majority of the white race applying for loans, this can be used to interpret that the location from where the loan application analysis was done, whites are the majority in population.

Bivariate analysis The bivariate analysis was conducted to understand the spread of 2 continuous variables - purchase price and loan amount. Here we found important insights on their relationship, the visually found insight was then confirmed by quantifying using correlation analysis by finding strong positive correlation coefficient value.

Hypothesis testing Some key hypothesis which rose while inspecting the summary statistics of the dataset were tested, which was to find relationship between 2 categorical variables - married and num of dependents. Post running the statistical test (chi square) we found our assumption was in fact true that both of these variables indeed are dependent on each other. Another hypothesis that was to understand if there was a difference in the loan amount means while the loan was rejected based on different property types. Here we found out that mean of the loan amount with loan rejection having property type 1 was significantly different from the mean of loan amount with loan rejection having property type 2, the same was found to be true between property type 1 and property type 3. Hence giving a caution to the buyer to be careful in deciding property type which may impact loan decision

Grouping categorical variables and understanding summary statistics By performing this activity we understood some useful insights on how different groups are asking for loan amount, we understood that with increase in number of dependents generally the ask also was increasing as this is naturally expected, occupancy type 3 was found to have lowest applicant income and occupancy type 2 the highest

Regression model A regression model was constructed keeping the loan amount as dependent variable and combination of significant variables, the coefficients were successfully obtained and prediction was run based on the existing data. However there was a conflict of adequacy for the model, in one of the assumption for normality as the model failed Jarque-Bera test. Hence there is some scope for improvement for the model where we can consider adding more relevant variables to successfully obtain parsimonious model and run predictions