s2

December 5, 2023

0.0.1 Packages

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

#
previous_application = pd.read_csv("./data/previous_application.csv")
application_data = pd.read_csv("./data/application_data.csv")
description = pd.read_excel("./data/columns_description.xlsx")
```

0.0.2 Data Cleaning

Prior to commencing our analysis, we conducted an initial assessment of the dataframe and identified the presence of NaN values. This initial data cleaning phase serves the dual purpose of streamlining our dataset. It involves the removal of superfluous columns, addressing specific subtopics, and mitigating substantial discrepancies arising from missing data.

[3]:
```
# Count the number of null values in each column of the 'application_data'␣
↪DataFrame
application_data.isnull().sum()
```

[3]:
```
SK_ID_CURR                      0
TARGET                          0
NAME_CONTRACT_TYPE              0
CODE_GENDER                     0
FLAG_OWN_CAR                    0
                              …
AMT_REQ_CREDIT_BUREAU_DAY   41519
AMT_REQ_CREDIT_BUREAU_WEEK  41519
AMT_REQ_CREDIT_BUREAU_MON   41519
AMT_REQ_CREDIT_BUREAU_QRT   41519
AMT_REQ_CREDIT_BUREAU_YEAR  41519
```

```
Length: 122, dtype: int64
```

[4]:
```
# Count the number of null values in each column of the 'application_data'␣
↪DataFrame
null_data = application_data.isnull().sum() * 100/len(application_data)
missing_data_columns = null_data[null_data >= 40] # for greater then 40% of nan␣
↪values allocate them in this variable
missing_data_columns # print value
```

[4]:
```
OWN_CAR_AGE                     65.990810
EXT_SOURCE_1                    56.381073
APARTMENTS_AVG                  50.749729
BASEMENTAREA_AVG                58.515956
YEARS_BEGINEXPLUATATION_AVG     48.781019
YEARS_BUILD_AVG                 66.497784
COMMONAREA_AVG                  69.872297
ELEVATORS_AVG                   53.295980
ENTRANCES_AVG                   50.348768
FLOORSMAX_AVG                   49.760822
FLOORSMIN_AVG                   67.848630
LANDAREA_AVG                    59.376738
LIVINGAPARTMENTS_AVG            68.354953
LIVINGAREA_AVG                  50.193326
NONLIVINGAPARTMENTS_AVG         69.432963
NONLIVINGAREA_AVG               55.179164
APARTMENTS_MODE                 50.749729
BASEMENTAREA_MODE               58.515956
YEARS_BEGINEXPLUATATION_MODE    48.781019
YEARS_BUILD_MODE                66.497784
COMMONAREA_MODE                 69.872297
ELEVATORS_MODE                  53.295980
ENTRANCES_MODE                  50.348768
FLOORSMAX_MODE                  49.760822
FLOORSMIN_MODE                  67.848630
LANDAREA_MODE                   59.376738
LIVINGAPARTMENTS_MODE           68.354953
LIVINGAREA_MODE                 50.193326
NONLIVINGAPARTMENTS_MODE        69.432963
NONLIVINGAREA_MODE              55.179164
APARTMENTS_MEDI                 50.749729
BASEMENTAREA_MEDI               58.515956
YEARS_BEGINEXPLUATATION_MEDI    48.781019
YEARS_BUILD_MEDI                66.497784
COMMONAREA_MEDI                 69.872297
ELEVATORS_MEDI                  53.295980
ENTRANCES_MEDI                  50.348768
FLOORSMAX_MEDI                  49.760822
```

```
FLOORSMIN_MEDI                67.848630
LANDAREA_MEDI                 59.376738
LIVINGAPARTMENTS_MEDI         68.354953
LIVINGAREA_MEDI               50.193326
NONLIVINGAPARTMENTS_MEDI      69.432963
NONLIVINGAREA_MEDI            55.179164
FONDKAPREMONT_MODE            68.386172
HOUSETYPE_MODE                50.176091
TOTALAREA_MODE                48.268517
WALLSMATERIAL_MODE            50.840783
EMERGENCYSTATE_MODE           47.398304
dtype: float64
```

[5]:
```
# dropping the above columns from the analysis as they contain a large␣
↪percentage of Nan values
application_data_df = application_data.drop(columns=missing_data_columns.index)
```

[6]:
```
application_data_df.shape

# Calculate the proportion of missing values in each row
missing_rows = application_data_df.isnull().sum(axis=1)/application_data_df.
↪shape[1]

# Assert that there are no rows with more than 50% NaN values
assert len(missing_rows[missing_rows > 50]) == 0

# Print a message indicating that none of the rows have more than 50% NaN values
print("we see that none of the rows have more than 50% nan values. so we will␣
↪proceed with further checks.")

# Display rows with more than 50% NaN values (if any)
missing_rows[missing_rows > 50]
```

```
we see that none of the rows have more than 50% nan values. so we will proceed
with further checks.
```

[6]:
```
Series([], dtype: float64)
```

Now that we removed the values greater then 50%, now we can take care of the null values, but will try to find the values with which it can be imputed at later point in time

[7]:
```
# Filter columns with missing data counts between 1 and 15
minor_missing_data_col = null_data[(null_data <= 15) & (null_data > 0)].
↪sort_values(ascending=False)

# Display columns with minor missing data
minor_missing_data_col
```

[7]:
```
AMT_REQ_CREDIT_BUREAU_HOUR    13.501631
AMT_REQ_CREDIT_BUREAU_DAY     13.501631
AMT_REQ_CREDIT_BUREAU_WEEK    13.501631
AMT_REQ_CREDIT_BUREAU_MON     13.501631
AMT_REQ_CREDIT_BUREAU_QRT     13.501631
AMT_REQ_CREDIT_BUREAU_YEAR    13.501631
NAME_TYPE_SUITE                0.420148
OBS_30_CNT_SOCIAL_CIRCLE       0.332021
DEF_30_CNT_SOCIAL_CIRCLE       0.332021
OBS_60_CNT_SOCIAL_CIRCLE       0.332021
DEF_60_CNT_SOCIAL_CIRCLE       0.332021
EXT_SOURCE_2                   0.214626
AMT_GOODS_PRICE                0.090403
AMT_ANNUITY                    0.003902
CNT_FAM_MEMBERS                0.000650
DAYS_LAST_PHONE_CHANGE         0.000325
dtype: float64
```

[8]:
```python
# Filter columns with missing data counts greater than 13 from
↪'minor_missing_data_col'
tmep_columns = minor_missing_data_col[minor_missing_data_col > 13].index.values

# Display information about the selected columns in 'application_data_df'
application_data_df[tmep_columns].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 6 columns):
 #   Column                      Non-Null Count   Dtype
---  ------                      --------------   -----
 0   AMT_REQ_CREDIT_BUREAU_HOUR  265992 non-null  float64
 1   AMT_REQ_CREDIT_BUREAU_DAY   265992 non-null  float64
 2   AMT_REQ_CREDIT_BUREAU_WEEK  265992 non-null  float64
 3   AMT_REQ_CREDIT_BUREAU_MON   265992 non-null  float64
 4   AMT_REQ_CREDIT_BUREAU_QRT   265992 non-null  float64
 5   AMT_REQ_CREDIT_BUREAU_YEAR  265992 non-null  float64
dtypes: float64(6)
memory usage: 14.1 MB
```

[9]:
```python
# Iterate over columns in 'tmep_columns' and print unique values count for each
↪column
for col in tmep_columns:
    print(f"{col} unique values: {application_data_df[col].nunique()}")
```

```
AMT_REQ_CREDIT_BUREAU_HOUR unique values: 5
AMT_REQ_CREDIT_BUREAU_DAY unique values: 9
AMT_REQ_CREDIT_BUREAU_WEEK unique values: 9
AMT_REQ_CREDIT_BUREAU_MON unique values: 24
```

```
AMT_REQ_CREDIT_BUREAU_QRT unique values: 11
AMT_REQ_CREDIT_BUREAU_YEAR unique values: 25
```

[10]:
```
# Display the column names of the 'application_data_df' DataFrame
application_data_df.columns
```

[10]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
       'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'ORGANIZATION_TYPE', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
       'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6',
       'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9',
       'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
       'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15',
       'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',
       'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object')
```

[11]:
```
# This columns have no information we can gain from keeping these flag documents
application_data_df.drop(columns=['FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3',
                              'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',␣
 ↪'FLAG_DOCUMENT_6',
                              'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8',␣
 ↪'FLAG_DOCUMENT_9',
                              'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11',␣
 ↪'FLAG_DOCUMENT_12',
                              'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14',␣
 ↪'FLAG_DOCUMENT_15',
                              'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',␣
 ↪'FLAG_DOCUMENT_18',
```

```
                           'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',␣
 ↪'FLAG_DOCUMENT_21',], inplace=True)
```

[31]:
```
application_data_df.columns
```

[31]:
```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'YEARS_BIRTH',
       'YEARS_EMPLOYED', 'YEARS_REGISTRATION', 'YEARS_ID_PUBLISH',
       'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE',
       'FLAG_PHONE', 'FLAG_EMAIL', 'OCCUPATION_TYPE', 'CNT_FAM_MEMBERS',
       'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY',
       'ORGANIZATION_TYPE', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE',
       'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR', 'income_group'],
      dtype='object')
```

[14]:
```
# convert those negative values into positive values
days_cols = ['DAYS_BIRTH' ,'DAYS_EMPLOYED' ,'DAYS_REGISTRATION'␣
 ↪,'DAYS_ID_PUBLISH']
application_data_df[days_cols] = application_data_df[days_cols].abs()
application_data_df[days_cols] = application_data_df[days_cols]/365
application_data_df[days_cols].describe()
```

[14]:

	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_REGISTRATION	DAYS_ID_PUBLISH
count	307511.000000	307511.000000	307511.000000	307511.000000
mean	43.936973	185.547239	13.660604	8.203294
std	11.956133	382.037676	9.651743	4.135481
min	20.517808	0.000000	0.000000	0.000000
25%	34.008219	2.556164	5.506849	4.712329
50%	43.150685	6.079452	12.339726	8.915068
75%	53.923288	15.635616	20.491781	11.778082
max	69.120548	1000.665753	67.594521	19.717808

[15]:
```
# rename from day to year
application_data_df.rename(columns={'DAYS_BIRTH':'YEARS_BIRTH' ,'DAYS_EMPLOYED':
 ↪'YEARS_EMPLOYED' ,
```

```
    'DAYS_REGISTRATION':'YEARS_REGISTRATION' ,'DAYS_ID_PUBLISH':
↪'YEARS_ID_PUBLISH'}, inplace=True)
```

0.1 Subtopic 2

Hypothesis: - During economic downturns, individuals without current employment are more likely to experience challenges in making timely payments on pre-existing loans compared to those with stable employment.

In this section, we will examine the complex interactions between income conditions and employment and how these affect loan repayment. It is vital for financial organizations to comprehend the ways in which income levels and employment stability impact customer in order to minimize hazards and enhance lending tactics.

Columns: - Target – Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases) - AMT_INCOME_TOTAL – Income of the client - OCCUPATION_TYPE – What kind of occupation does the client have

application_data_df.columns

[16]:
```
# convert those negative values into positive values
days_cols = ['DAYS_BIRTH' ,'DAYS_EMPLOYED' ,'DAYS_REGISTRATION'␣
↪,'DAYS_ID_PUBLISH']
application_data_df[days_cols] = application_data_df[days_cols].abs()
application_data_df[days_cols] = application_data_df[days_cols]/365
application_data_df[days_cols].describe()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/Users/lucavivona/Documents/Programs/York/MATH/1130/Project/s2.ipynb Cell 22␣
↪line 3
      <a href='vscode-notebook-cell:/Users/lucavivona/Documents/Programs/York/
↪MATH/1130/Project/s2.ipynb#X30sZmlsZQ%3D%3D?line=0'>1</a> # convert those␣
↪negative values into positive values
      <a href='vscode-notebook-cell:/Users/lucavivona/Documents/Programs/York/
↪MATH/1130/Project/s2.ipynb#X30sZmlsZQ%3D%3D?line=1'>2</a> days_cols =␣
↪['DAYS_BIRTH' ,'DAYS_EMPLOYED' ,'DAYS_REGISTRATION' ,'DAYS_ID_PUBLISH']
----> <a href='vscode-notebook-cell:/Users/lucavivona/Documents/Programs/York/
↪MATH/1130/Project/s2.ipynb#X30sZmlsZQ%3D%3D?line=2'>3</a>␣
↪application_data_df[days_cols] = application_data_df[days_cols].abs()
      <a href='vscode-notebook-cell:/Users/lucavivona/Documents/Programs/York/
↪MATH/1130/Project/s2.ipynb#X30sZmlsZQ%3D%3D?line=3'>4</a>␣
↪application_data_df[days_cols] = application_data_df[days_cols]/365
      <a href='vscode-notebook-cell:/Users/lucavivona/Documents/Programs/York/
↪MATH/1130/Project/s2.ipynb#X30sZmlsZQ%3D%3D?line=4'>5</a>␣
↪application_data_df[days_cols].describe()

File ~/opt/anaconda3/envs/base310/lib/python3.10/site-packages/pandas/core/frame.
↪py:3811, in DataFrame.__getitem__(self, key)
```

```
   3809     if is_iterator(key):
   3810         key = list(key)
-> 3811     indexer = self.columns._get_indexer_strict(key, "columns")[1]
   3813 # take() does not accept boolean indexers
   3814 if getattr(indexer, "dtype", None) == bool:

File ~/opt/anaconda3/envs/base310/lib/python3.10/site-packages/pandas/core/
↪indexes/base.py:6113, in Index._get_indexer_strict(self, key, axis_name)
   6110 else:
   6111     keyarr, indexer, new_indexer = self._reindex_non_unique(keyarr)
-> 6113 self._raise_if_missing(keyarr, indexer, axis_name)
   6115 keyarr = self.take(indexer)
   6116 if isinstance(key, Index):
   6117     # GH 42790 - Preserve name from an Index

File ~/opt/anaconda3/envs/base310/lib/python3.10/site-packages/pandas/core/
↪indexes/base.py:6173, in Index._raise_if_missing(self, key, indexer, axis_name)
   6171     if use_interval_msg:
   6172         key = list(key)
-> 6173     raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   6175 not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
   6176 raise KeyError(f"{not_found} not in index")

KeyError: "None of [Index(['DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION',␣
↪'DAYS_ID_PUBLISH'], dtype='object')] are in the [columns]"
```

[ ]:
```
# rename from day to year
application_data_df.rename(columns={'DAYS_BIRTH':'YEARS_BIRTH' ,'DAYS_EMPLOYED':
↪'YEARS_EMPLOYED' ,
    'DAYS_REGISTRATION':'YEARS_REGISTRATION' ,'DAYS_ID_PUBLISH':
↪'YEARS_ID_PUBLISH'}, inplace=True)
```

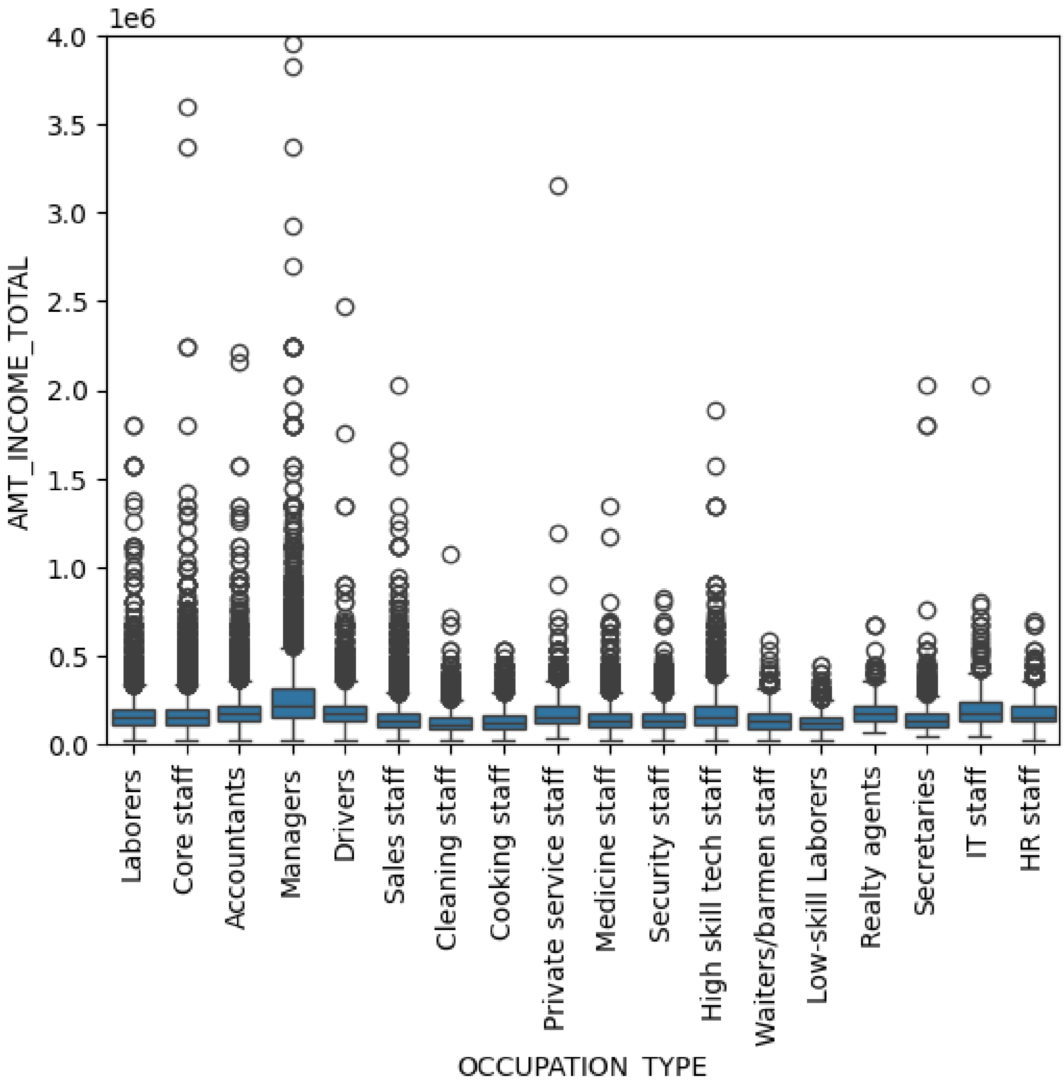
0.1.1 Outlier Detection

In our dataset, any data points lying outside the range delineated by the box in the box plot are considered outliers. These outliers are values that fall significantly beyond the typical distribution and may warrant further investigation.

0.1.2 Employment

[17]:
```
# Create a boxplot using Seaborn for 'AMT_INCOME_TOTAL' grouped by␣
↪'OCCUPATION_TYPE'
ax = sns.
↪boxplot(data=application_data_df,y='AMT_INCOME_TOTAL',x='OCCUPATION_TYPE')

# Rotate x-axis labels for better visibility
plt.xticks(rotation=90)
```

```
# Set y-axis limits for better visualization
plt.ylim(0, 04e6)

# Display the plot
plt.show()
```



To analysis wha how employment we need to be able to distinguish between low paying income jobs, the reason for such that our hypothesis presume that low paying jobs will likely effect risk of payment either that being on time or for approval for the bank

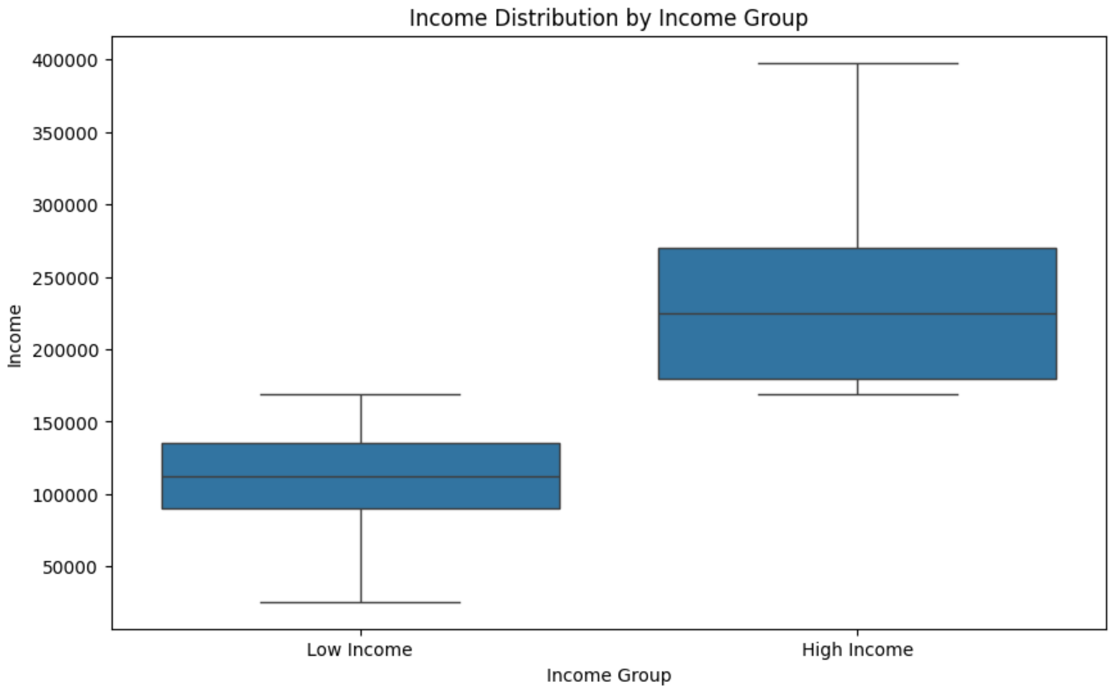
[18]:
```
# Create a new column 'income_group' based on the threshold
income_threshold = application_data_df["AMT_INCOME_TOTAL"].mean()
```

```
application_data_df['income_group'] = pd.
 ↪cut(application_data_df['AMT_INCOME_TOTAL'], bins=[float('-inf'),␣
 ↪income_threshold, float('inf')],
                              labels=['Low Income', 'High Income'], right=False)
```

[19]:
```
# Print the percentage of null values in the 'OCCUPATION_TYPE' column
print("Total of", application_data_df["OCCUPATION_TYPE"].isnull().sum() /␣
 ↪len(application_data_df["OCCUPATION_TYPE"].notnull()), "% is null for␣
 ↪occupation type")

# lets remove since its relative small piece of the dataset we could just␣
 ↪remove it for simplicity of course this comes at a cost
application_data_df.dropna(subset=['OCCUPATION_TYPE'], inplace=True)

# Print the percentage of null values in the 'OCCUPATION_TYPE' column
print("Total of", application_data_df["OCCUPATION_TYPE"].isnull().sum() /␣
 ↪len(application_data_df["OCCUPATION_TYPE"].notnull()), "% is null for␣
 ↪occupation type")
```

```
Total of 0.31345545362604915 % is null for occupation type
Total of 0.0 % is null for occupation type
```

[20]:
```
# delegate our income groups two groups
# have an outlier threshold of 0.4e6
plt.figure(figsize=(10, 6))
sns.boxplot(x='income_group', y='AMT_INCOME_TOTAL',␣
 ↪data=application_data_df[application_data_df["AMT_INCOME_TOTAL"]< 0.40e6])
plt.title('Income Distribution by Income Group')
plt.xlabel('Income Group')
plt.ylabel('Income')
plt.show()
```



Within our box graph were shown how each occupation type is distributed, some some notable

Does Low income have a high chance to miss a Target payment?

[21]:
```
transformed_application_data_df =␣
↪application_data_df[application_data_df["AMT_INCOME_TOTAL"]< 0.4e6]
lower_income =␣
↪transformed_application_data_df[transformed_application_data_df["income_group"]␣
↪== "Low Income" ]
high_income =␣
↪transformed_application_data_df[transformed_application_data_df["income_group"]␣
↪== "High Income" ]
```

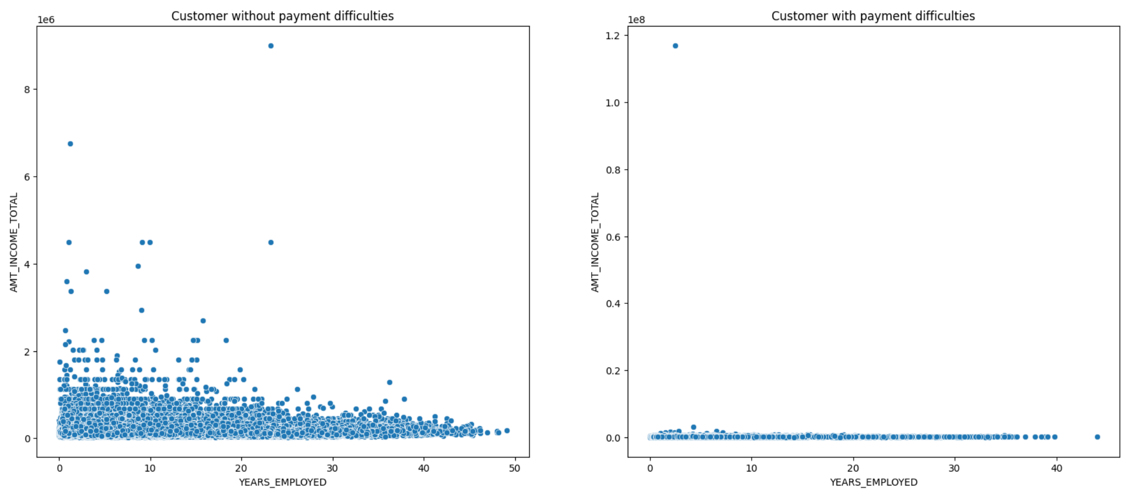
[22]:
```
len(lower_income), len(high_income)
```

[22]: (121871, 83055)

[23]:
```
# customer with no payment difficulty
application_data_df_0 = application_data_df[application_data_df["TARGET"] == 0]
# customer with payment difficulty
application_data_df_1 = application_data_df[application_data_df["TARGET"] == 1]
```

0.1.3 Analysis

[24]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
ax = sns.
↪scatterplot(data=application_data_df_0[application_data_df_0['YEARS_EMPLOYED']<1000],␣
↪x='YEARS_EMPLOYED',y='AMT_INCOME_TOTAL')
plt.title('Customer without payment difficulties')

plt.subplot(1,2,2)
ax = sns.
↪scatterplot(data=application_data_df_1[application_data_df_1['YEARS_EMPLOYED']<1000],␣
↪x='YEARS_EMPLOYED',y='AMT_INCOME_TOTAL')
plt.title('Customer with payment difficulties')
plt.show()
```



Customers with payment difficulties to exist on all years of employments, but when it comes to income we can fit small region of people on a line, excluding the outlier existing within the our set where customers with payment difficulties. the reason why this may occur could be issuing based income total when applying for the loan people within lower incomes have a high tendency to struggle with payments making them risker ventures for the bank but this is quite news, as other thing like high interests rate, and high credit not retrospect to income.

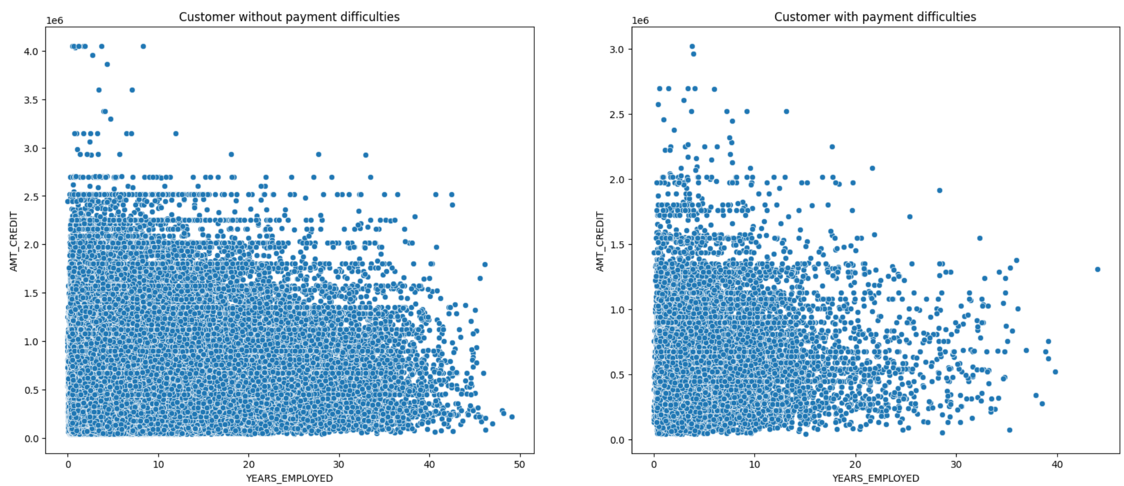
To look further we need to analyze two instances to find a better solution of mitigation of Customer payment difficulties

1. Low income house holds, with pre-existing application without payment difficulties
2. Low income house holds, with pre-existing application with payment difficulties

looking at those we can then analysis we can credit received by these institution and see if these income total are positively correlated within lower income

[25]:
```
# Excluding the point
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
ax = sns.
 ↪scatterplot(data=application_data_df_0[application_data_df_0['YEARS_EMPLOYED']<1000],␣
 ↪x='YEARS_EMPLOYED',y='AMT_CREDIT')
plt.title('Customer without payment difficulties')

plt.subplot(1,2,2)
ax = sns.
 ↪scatterplot(data=application_data_df_1[application_data_df_1['YEARS_EMPLOYED']<1000],␣
 ↪x='YEARS_EMPLOYED',y='AMT_CREDIT')
plt.title('Customer with payment difficulties')
```

[25]:
```
Text(0.5, 1.0, 'Customer with payment difficulties')
```



Scatter plot above mainly shows use the level of density of years employment people are in with respect of there credit application.

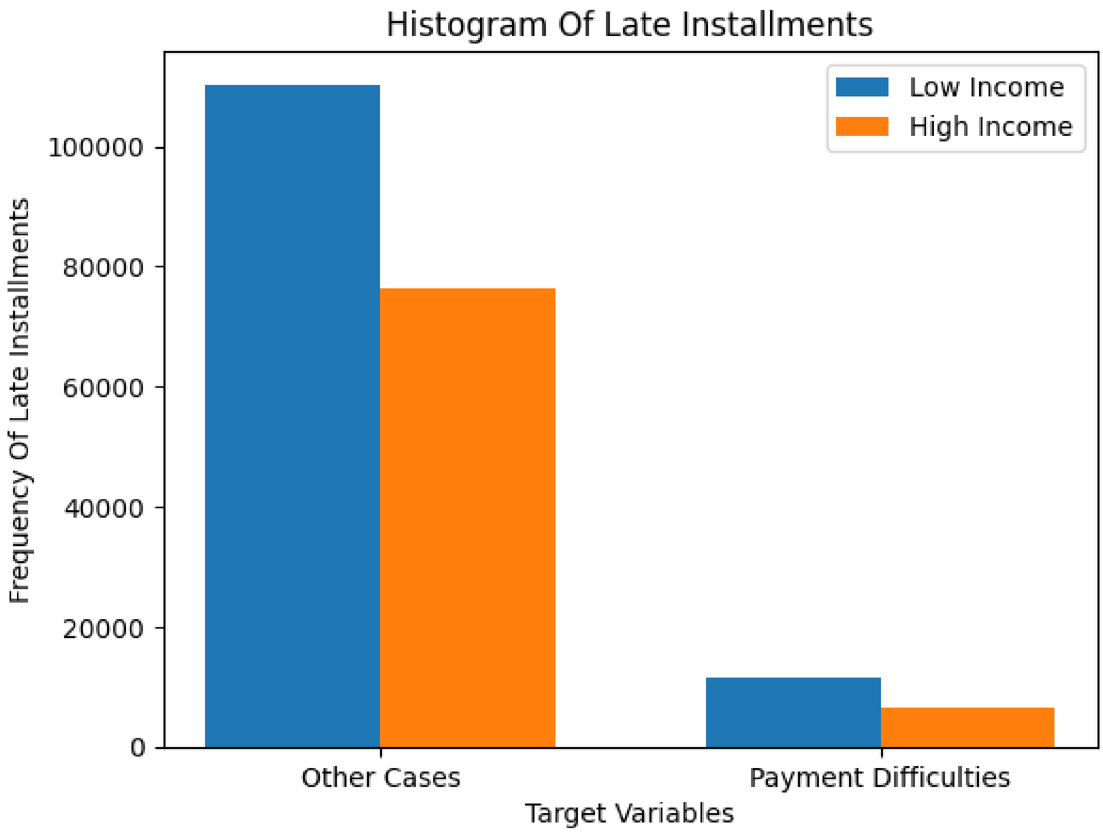
Customers without payment difficulties - High density people who have been employed for a long period of time - Similar to the payment difficulties there exist people employed near 0 years within the work force that tend for a higher credit.

Customer with payment difficulties - High density plots are within 0-20 range years employed. - AMT_CREDIT seems tapers off emailer then people without payment to difficulties an notable thing as well is there seems to be larger AMT_CREDITs within earlier years of employment this may be due to incoming workers within the work force beginning there job and wanting to take out a loan, for a house, or car

It's quite difficult to extrapolate from just this assessment so we have to dig little deeper. on some branch on employment, and see if jobs, and income of those jobs effect people ability to obtain loan, as well

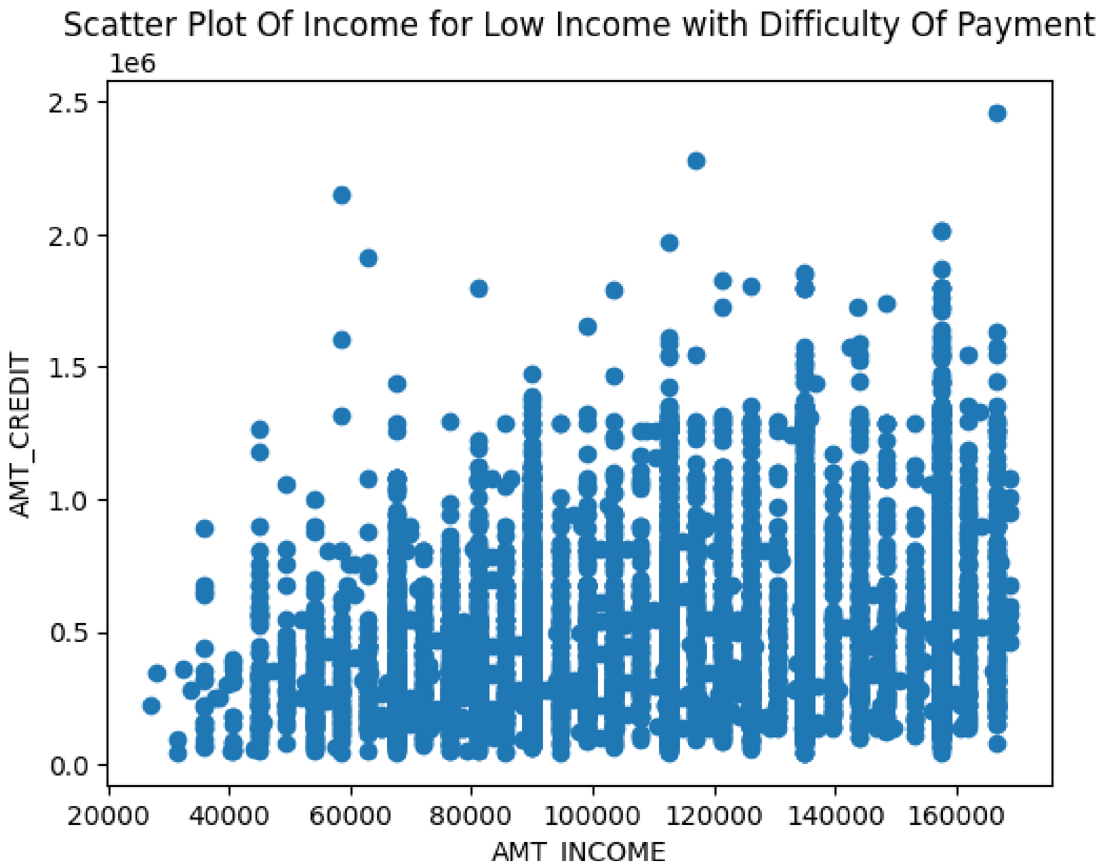
[26]:
```python
low_income_payment_difficulty = lower_income[lower_income["TARGET"] == 1]
low_income_no_payment_difficulty = lower_income[lower_income["TARGET"] == 0]


high_income_payment_difficulty = high_income[high_income["TARGET"] == 1]
high_income_no_payment_difficulty = high_income[high_income["TARGET"] == 0]
```

[30]:
```python
categories = ['Other Cases', 'Payment Difficulties']
target_low = lower_income.groupby("TARGET")["TARGET"].value_counts().to_list()
target_high = high_income.groupby("TARGET")["TARGET"].value_counts().to_list()
bar_width = 0.35
index = np.arange(len(categories))

# Create bar plot
plt.bar(index, target_low, bar_width, label='Low Income')
plt.bar(index + bar_width, target_high, bar_width, label='High Income')

# Customize plot
plt.xlabel('Target Variables')
plt.ylabel('Frequency Of Late Installments')
plt.title('Histogram Of Late Installments')
plt.xticks(index + bar_width / 2, categories)
plt.legend()

# Show plot
plt.show()
```



Similar within the first subtopic, we can look at the how low and high income distributes between payment difficulties, and non payment difficulties. we can notice within the distribution we have a larger sample of Lower income cases in both categories.

```
[ ]: payment_difficulty_li = lower_income[lower_income['TARGET'] == 1]
     plt.scatter(y=payment_difficulty_li["AMT_CREDIT"],␣
      ↪x=payment_difficulty_li["AMT_INCOME_TOTAL"])
     plt.title('Scatter Plot Of Income for Low Income with Difficulty Of Payment')
     plt.xlabel('AMT_INCOME')
     plt.ylabel('AMT_CREDIT')
     plt.show()
```

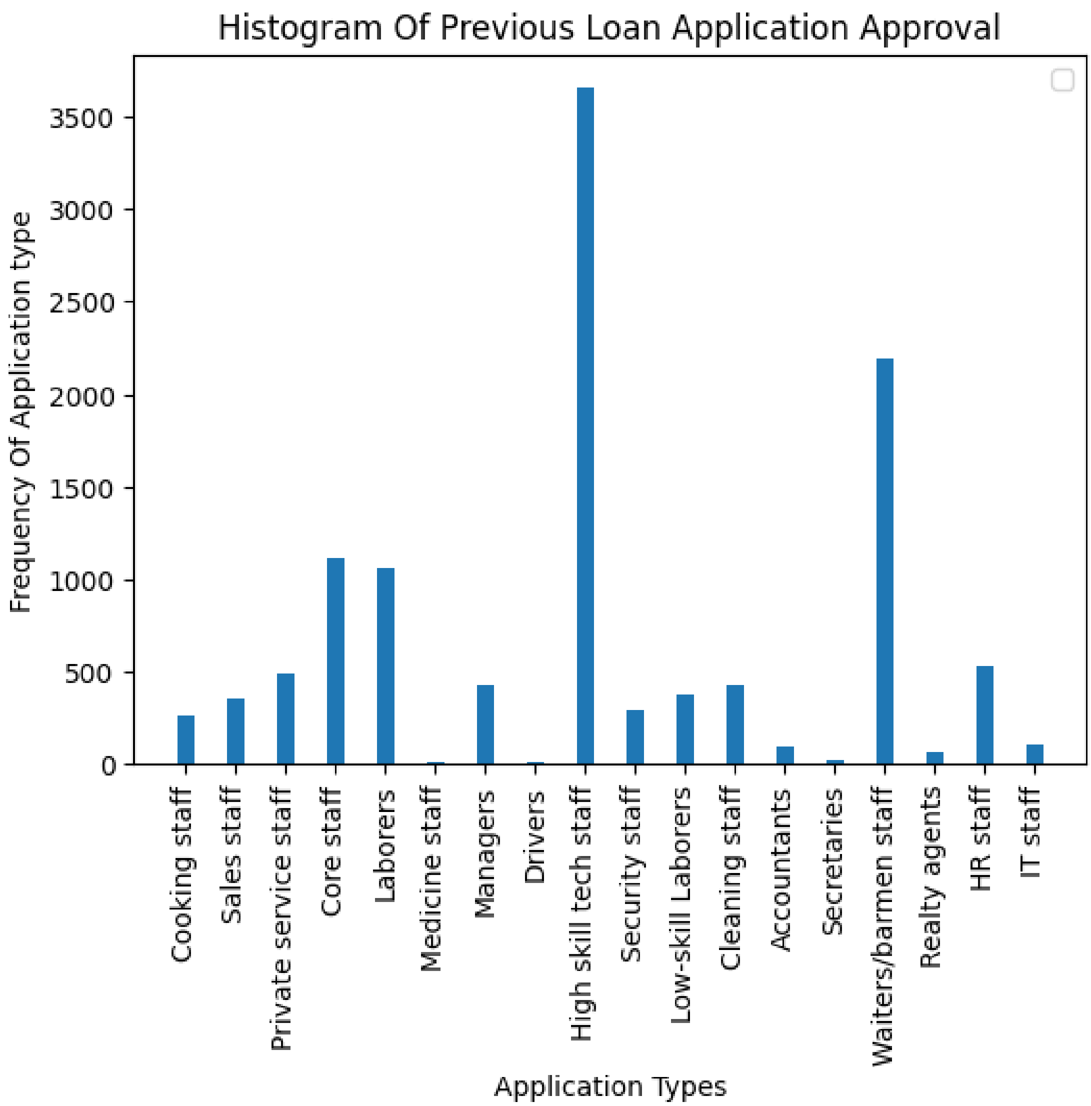


Quite difficult to read we see a trend upwards as the person has more income the larger the credit they ask. but who are these groups that are proving the most fault within our dataset

[ ]:
```
categories = payment_difficulty_li["OCCUPATION_TYPE"].unique().tolist()
bar_width = 0.35
index = np.arange(len(categories))

target = payment_difficulty_li.groupby("OCCUPATION_TYPE")["OCCUPATION_TYPE"].
  ↪value_counts().tolist()

# Create bar plot
plt.bar(index, target, bar_width)

# Customize plot
plt.xlabel('Application Types')
plt.ylabel('Frequency Of Application type')
plt.title('Histogram Of Previous Loan Application Approval')
plt.xticks(index, categories, rotation=90)
plt.legend()
```

```
# Show plot
plt.show()
```

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



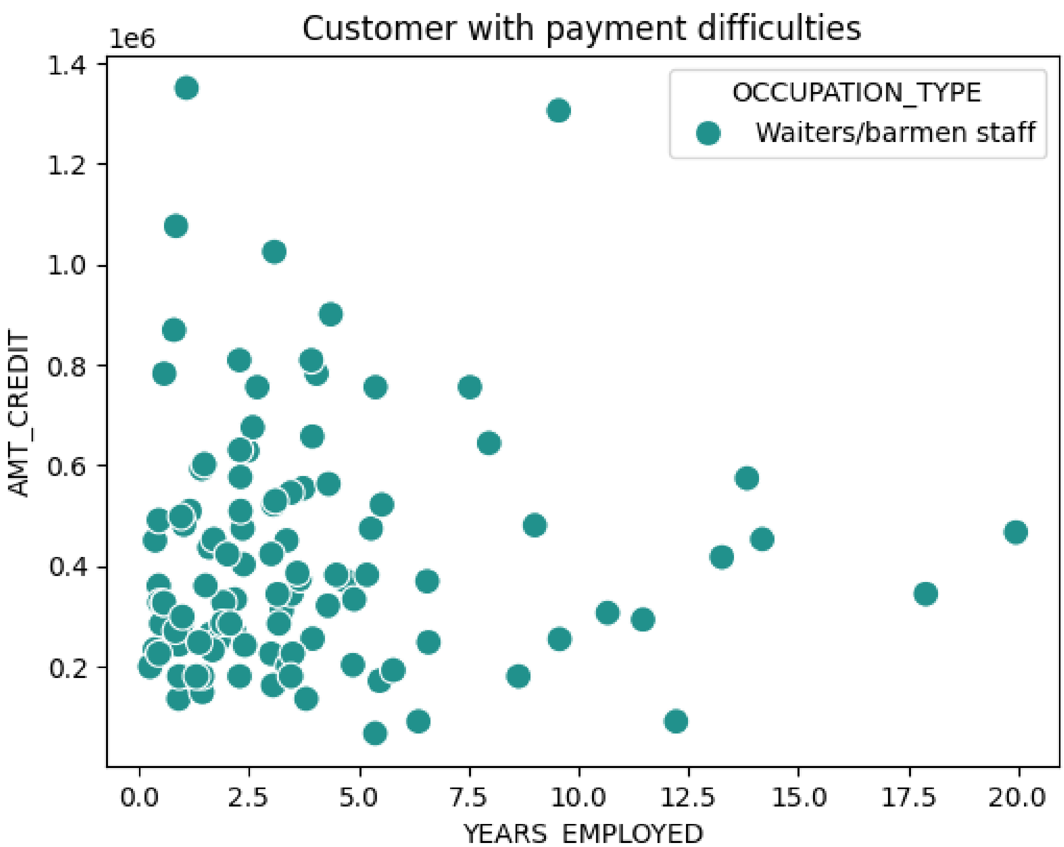
With surprising results, it seems that High skilled tech staff have a largest frequency of payment defaults within the lower income subset, with waiters/barmen staff in close second. this may be speculation but within high skill tech staff it might be in the company best interest

[ ]:
```
payment_difficulty_li[payment_difficulty_li["OCCUPATION_TYPE"] == "High skill␣
↪tech staff"]["YEARS_EMPLOYED"]
```

```
count    427.000000
mean       5.628020
std        6.265681
min        0.208219
25%        1.480822
50%        3.608219
75%        6.915068
max       33.391781
Name: YEARS_EMPLOYED, dtype: float64
```

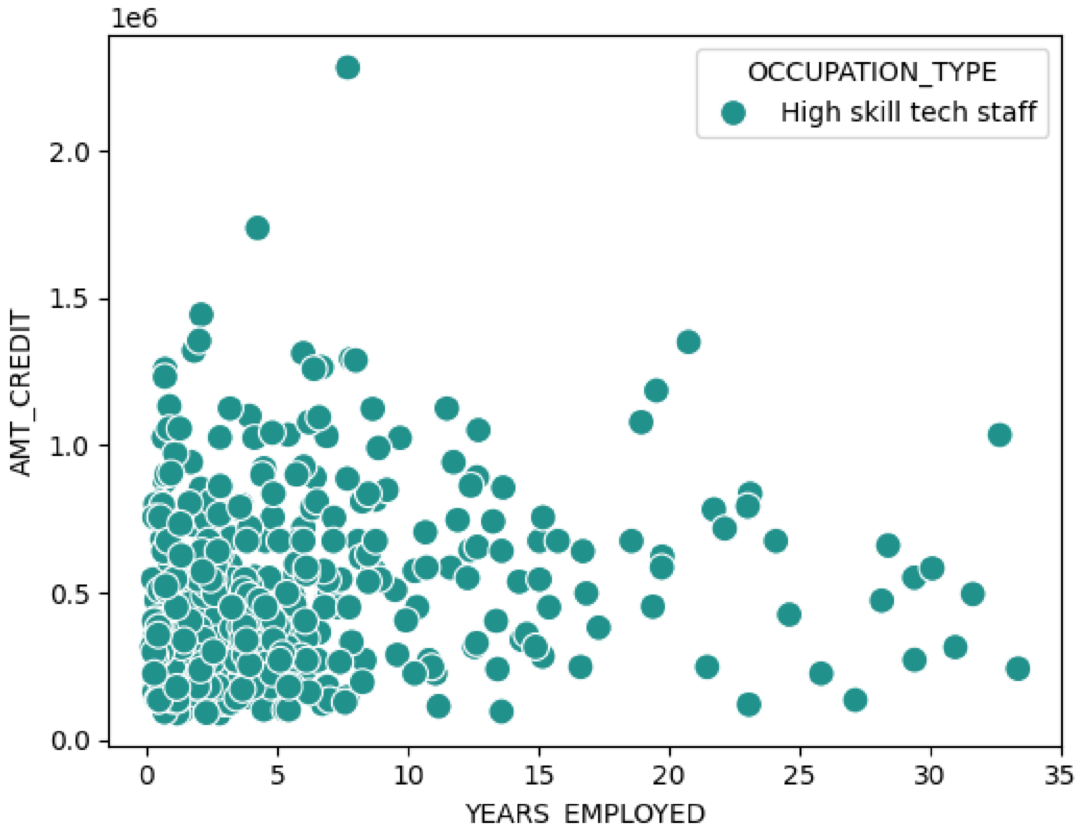
[ ]:
```
ax = sns.
 ↪scatterplot(data=payment_difficulty_li[payment_difficulty_li["OCCUPATION_TYPE"]␣
 ↪== "Waiters/barmen staff"], x='YEARS_EMPLOYED',y='AMT_CREDIT',␣
 ↪hue='OCCUPATION_TYPE', palette='viridis', s=100)
plt.title('Customer with payment difficulties')
```

```
Text(0.5, 1.0, 'Customer with payment difficulties')
```



[ ]:

```
ax = sns.
↪scatterplot(data=payment_difficulty_li[payment_difficulty_li["OCCUPATION_TYPE"]␣
↪== "High skill tech staff"], x='YEARS_EMPLOYED',y='AMT_CREDIT',␣
↪hue='OCCUPATION_TYPE', palette='viridis', s=100)
```



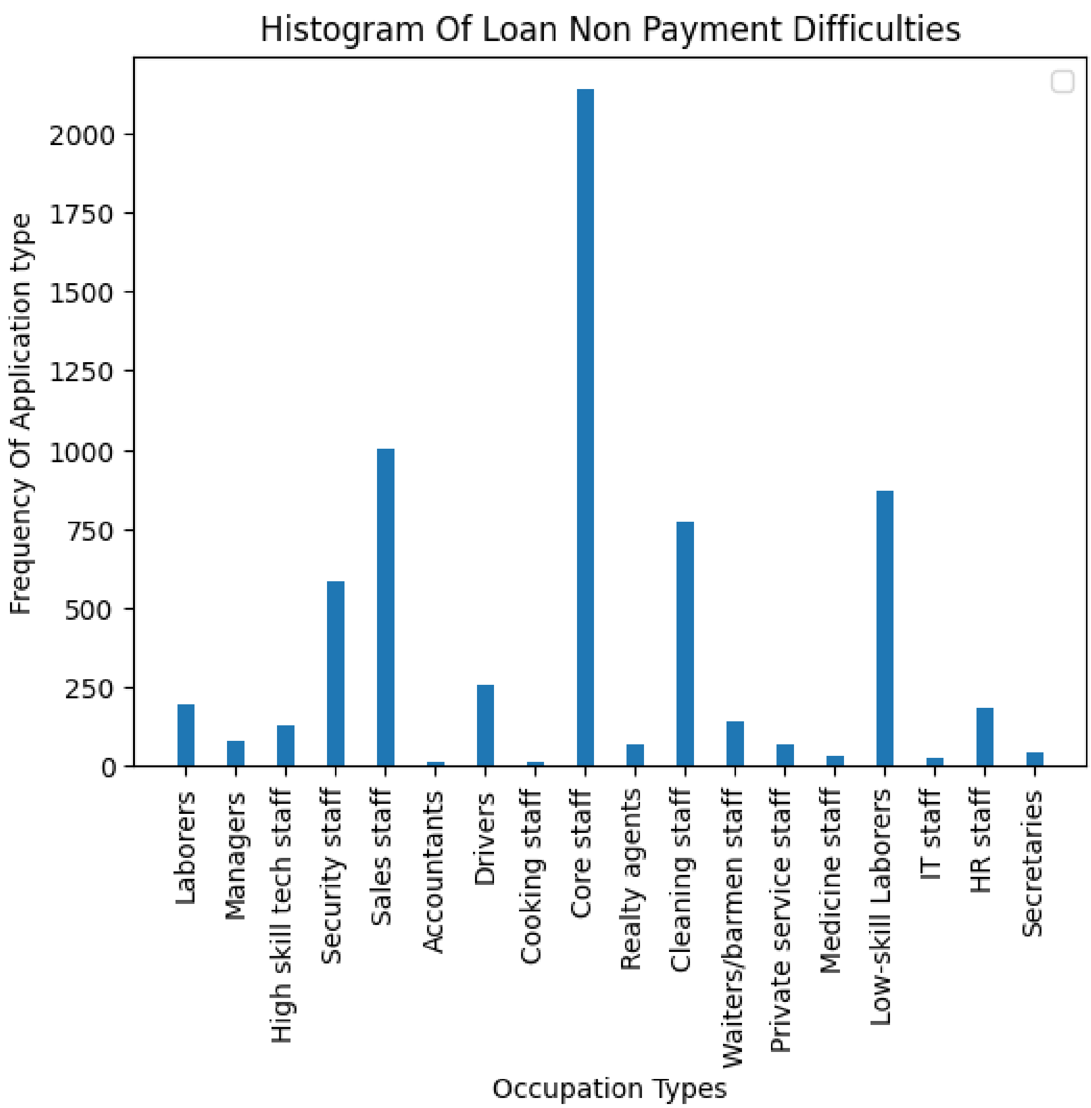
Visualizing the density of Years employed with out top two lowest earners, we see a majority of those workers are within the beginning of there occupation.

For simplicity lets take the top two, and see does this apply to high income customers within these fields.

[ ]:
```
payment_difficulty_hi= high_income[high_income['TARGET'] == 1]

categories = payment_difficulty_hi["OCCUPATION_TYPE"].unique().tolist()
bar_width = 0.35
index = np.arange(len(categories))

target = payment_difficulty_hi.groupby("OCCUPATION_TYPE")["OCCUPATION_TYPE"].
↪value_counts().tolist()

# Create bar plot
```

```
plt.bar(index, target, bar_width)

# Customize plot
plt.xlabel('Occupation Types')
plt.ylabel('Frequency Of Application type')
plt.title('Histogram Of Loan Non Payment Difficulties')
plt.xticks(index, categories, rotation=90)
plt.legend()

# Show plot
plt.show()
```

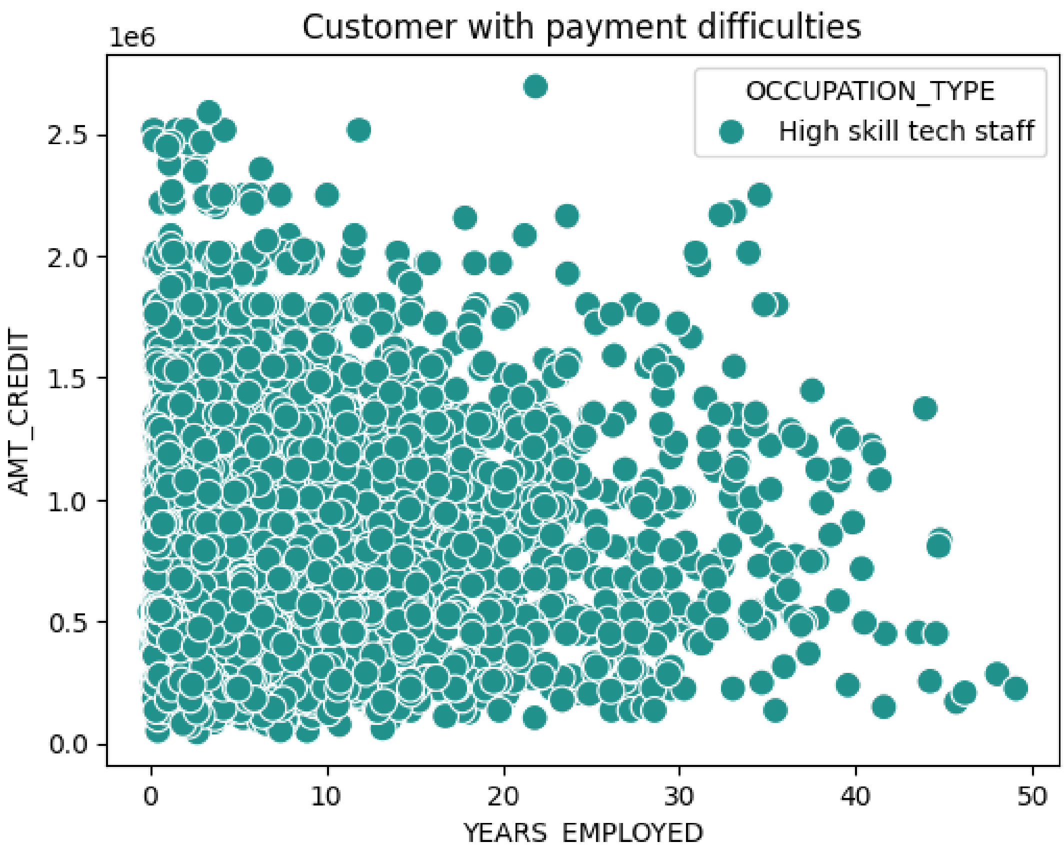
```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```



[ ]:

```
ax = sns.
 ↪scatterplot(data=payment_difficulty_hi[payment_difficulty_hi["OCCUPATION_TYPE"]␣
 ↪== "High skill tech staff"], x='YEARS_EMPLOYED',y='AMT_CREDIT',␣
 ↪hue='OCCUPATION_TYPE', palette='viridis', s=100)
plt.title('Customer with payment difficulties')
```

```
Text(0.5, 1.0, 'Customer with payment difficulties')
```



[ ]:

```
payment_difficulty_hi[payment_difficulty_hi["OCCUPATION_TYPE"] == "High skill␣
 ↪tech staff"]["YEARS_EMPLOYED"].describe()
```

```
count    258.000000
mean       6.281555
std        6.545439
min        0.249315
25%        1.989726
50%        4.335616
75%        8.110959
max       34.131507
```

```
Name: YEARS_EMPLOYED, dtype: float64
```

Conclusion

In various occupational categories, the income levels exhibit variability, reflecting the diverse salary structures associated with these professions. While predicting these variations proves challenging, discernible trends emerge from our analysis. Notably, a distinctive pattern surfaces, revealing that individuals with a propensity for late payments often belong to the category of high-skilled technology staff.

Problem With Our Analysis: Interpreting Missed Payments as Early Warning Signs: - Our analysis presupposes that missing a payment is a reliable early warning sign, but this assumption may oversimplify the issue. Various external factors, beyond an individual's control, could contribute to missed payments, challenging the accuracy of this indicator.

Fixed Parameters in Analysis: - Our analysis relies on fixed parameters, potentially overlooking dynamic factors such as changes in living expenses, access to credit, and individual financial management skills. The complexity of these variables could confound the relationships we are attempting to assess.

Limited Sample Representativeness: - It's important to acknowledge that our sample, derived from a single bank's dataset, may not be fully representative of the broader population. Generalizing findings based on this limited scope might not accurately reflect the diverse financial behaviors present in the wider community.

Granularity Sacrificed in Threshold Definition: - Establishing a threshold for defining high income, such as using the mean, introduces a trade-off by sacrificing granularity in the information. Aggregating beyond this threshold may obscure nuanced variations in income levels, limiting the depth of our insights.

```
[ ]:
```