Rahul_Ranjan_Credit_EDA_and_Scoring_Case_Study_Final_Submission

October 9, 2024

[77]:
```
!pip install pingouin
!pip install --upgrade category_encoders
import pingouin as pg
from decimal import Decimal, getcontext
import numpy as np
import random
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
from scipy.stats import poisson, expon, geom,norm
from statsmodels.stats import weightstats as stests
import statsmodels.api as sm
from scipy.stats import ttest_1samp
from scipy.stats import ttest_ind
from scipy.stats import ttest_rel
from scipy.stats import powerlaw
from scipy.stats import chisquare # Statistical test (chistat, pvalue)
from scipy.stats import chi2
from scipy.stats import chi2_contingency
from statsmodels.stats.proportion import proportions_ztest
from scipy.stats import f_oneway
from scipy.stats import kruskal
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import shapiro
from scipy.stats import levene
from scipy.stats import pearsonr,spearmanr
import warnings
warnings.filterwarnings('ignore')
from scipy.stats import skew
from scipy.stats import skew, kurtosis
from scipy.stats import kstest
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from category_encoders import TargetEncoder
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.preprocessing import MinMaxScaler
import scipy.stats as stats
```

```
Requirement already satisfied: pingouin in /usr/local/lib/python3.10/dist-
packages (0.5.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from pingouin) (3.7.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from pingouin) (1.26.4)
Requirement already satisfied: pandas>=1.5 in /usr/local/lib/python3.10/dist-
packages (from pingouin) (2.2.2)
Requirement already satisfied: pandas-flavor in /usr/local/lib/python3.10/dist-
packages (from pingouin) (0.6.0)
Requirement already satisfied: scikit-learn>=1.2 in
/usr/local/lib/python3.10/dist-packages (from pingouin) (1.5.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from pingouin) (1.13.1)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-
packages (from pingouin) (0.13.1)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (from pingouin) (0.14.4)
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-
packages (from pingouin) (0.9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.5->pingouin) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.5->pingouin) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.5->pingouin) (2024.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=1.2->pingouin) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2->pingouin)
(3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->pingouin) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->pingouin) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib->pingouin) (3.1.4)
Requirement already satisfied: xarray in /usr/local/lib/python3.10/dist-packages
(from pandas-flavor->pingouin) (2024.9.0)
Requirement already satisfied: patsy>=0.5.6 in /usr/local/lib/python3.10/dist-
packages (from statsmodels->pingouin) (0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.6->statsmodels->pingouin) (1.16.0)
Requirement already satisfied: category_encoders in
/usr/local/lib/python3.10/dist-packages (2.6.4)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.26.4)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders) (1.5.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (1.13.1)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.4)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-
packages (from category_encoders) (0.5.6)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.5->category_encoders) (2024.2)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.20.0->category_encoders) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn>=0.20.0->category_encoders) (3.5.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from
statsmodels>=0.9.0->category_encoders) (24.1)
```

[78]:
```
dataset=pd.read_csv("/content/Credit_score.csv")
dataset
```

[78]:

	ID	Customer_ID	Month	Name	Age	SSN	\
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	

```
4       0x1606   CUS_0xd40       May  Aaron Maashoh    23  821-00-0265
…         …        …       …            …   …           …
99995  0x25fe9  CUS_0x942c     April          Nicks    25  078-73-5990
99996  0x25fea  CUS_0x942c       May          Nicks    25  078-73-5990
99997  0x25feb  CUS_0x942c      June          Nicks    25  078-73-5990
99998  0x25fec  CUS_0x942c      July          Nicks    25  078-73-5990
99999  0x25fed  CUS_0x942c    August          Nicks    25  078-73-5990

      Occupation Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  …  \
0      Scientist      19114.12            1824.843333                  3  …
1      Scientist      19114.12                    NaN                  3  …
2      Scientist      19114.12                    NaN                  3  …
3      Scientist      19114.12                    NaN                  3  …
4      Scientist      19114.12            1824.843333                  3  …
…          …            …                     …                  …  …
99995   Mechanic      39628.99            3359.415833                  4  …
99996   Mechanic      39628.99            3359.415833                  4  …
99997   Mechanic      39628.99            3359.415833                  4  …
99998   Mechanic      39628.99            3359.415833                  4  …
99999   Mechanic     39628.99_            3359.415833                  4  …

       Num_Credit_Inquiries  Credit_Mix Outstanding_Debt  \
0                       4.0           _            809.98
1                       4.0        Good            809.98
2                       4.0        Good            809.98
3                       4.0        Good            809.98
4                       4.0        Good            809.98
…                       …          …                …
99995                   3.0           _            502.38
99996                   3.0           _            502.38
99997                   3.0        Good            502.38
99998                   3.0        Good            502.38
99999                   3.0        Good            502.38

      Credit_Utilization_Ratio      Credit_History_Age Payment_of_Min_Amount  \
0                    26.822620   22 Years and 1 Months                    No
1                    31.944960                     NaN                    No
2                    28.609352   22 Years and 3 Months                    No
3                    31.377862   22 Years and 4 Months                    No
4                    24.797347   22 Years and 5 Months                    No
…                          …                       …                    …
99995                34.663572   31 Years and 6 Months                    No
99996                40.565631   31 Years and 7 Months                    No
99997                41.255522   31 Years and 8 Months                    No
99998                33.638208   31 Years and 9 Months                    No
99999                34.192463  31 Years and 10 Months                    No
```

```
      Total_EMI_per_month  Amount_invested_monthly  \
0               49.574949              80.41529544
1               49.574949              118.2802216
2               49.574949              81.69952126
3               49.574949              199.4580744
4               49.574949              41.42015309
…                     …                        …
99995           35.104023              60.97133256
99996           35.104023              54.18595029
99997           35.104023              24.02847745
99998           35.104023              251.6725822
99999           35.104023              167.1638652

                     Payment_Behaviour Monthly_Balance
0       High_spent_Small_value_payments     312.4940887
1        Low_spent_Large_value_payments     284.6291625
2       Low_spent_Medium_value_payments     331.2098629
3        Low_spent_Small_value_payments     223.4513097
4      High_spent_Medium_value_payments      341.489231
…                                    …             …
99995   High_spent_Large_value_payments      479.866228
99996  High_spent_Medium_value_payments       496.65161
99997   High_spent_Large_value_payments      516.809083
99998    Low_spent_Large_value_payments      319.164979
99999                            !@9#%8      393.673696

[100000 rows x 27 columns]
```

[79]:
```
### Shape of the dataset
dataset.shape
```

[79]:
```
(100000, 27)
```

Performing EDA on the categorical columns of the dataset - and the numerical columns of the dataset

Evaluating -Datatypes, Missing Data, and Summary Statistics

[80]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 27 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  object
```

```
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  object
 7   Annual_Income             100000 non-null  object
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  object
 13  Type_of_Loan              88592 non-null   object
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   object
 16  Changed_Credit_Limit      100000 non-null  object
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object
 19  Outstanding_Debt          100000 non-null  object
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  object
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object
 25  Payment_Behaviour         100000 non-null  object
 26  Monthly_Balance           98800 non-null   object
dtypes: float64(4), int64(4), object(19)
memory usage: 20.6+ MB
```

Observation:- Columns - Month, Occupation, Type_of_Loan, Credit_Mix, Payment_of_Min_Amount, Payment_Behaviour, Credit_Score are categorical. Therefore,modifying the datatypes of these columns to category.

[81]:
```
### Changing the datatype of the above mentioned columns to category
dataset.Month = dataset.Month.astype('category')
dataset.Occupation = dataset.Occupation.astype('category')
dataset.Type_of_Loan = dataset.Type_of_Loan.astype('category')
dataset.Credit_Mix = dataset.Credit_Mix.astype('category')
dataset.Payment_of_Min_Amount = dataset.Payment_of_Min_Amount.astype('category')
dataset.Payment_Behaviour = dataset.Payment_Behaviour.astype('category')
```

[82]:
```
### Looking at the modified datatypes of the data
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 27 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
```

```
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  category
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  category
 7   Annual_Income             100000 non-null  object
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  object
 13  Type_of_Loan              88592 non-null   category
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   object
 16  Changed_Credit_Limit      100000 non-null  object
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  category
 19  Outstanding_Debt          100000 non-null  object
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  category
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object
 25  Payment_Behaviour         100000 non-null  category
 26  Monthly_Balance           98800 non-null   object
dtypes: category(6), float64(4), int64(4), object(13)
memory usage: 16.9+ MB
```

Observation:- From the above data, we can see that the columns - Age, Annual_Income, Num_of_Loan, Num_of_Delayed_Payment, Changed_Credit_Limit, Outstanding_Debt, Amount_invested_monthly, Monthly_Balance are object datatyped but they should be a number datatype like int or float.

Therefore,modifying the datatypes the datatype of these columns from object to a numerical datatype like int or float.

[83]:
```python
# This function removes leading and trailing underscores from a value
def removeUnderscore(value):
    # Remove underscores from the start and end of the value
    value = value.strip('_')

    # Return the cleaned value, or 0 if the result is empty
    return value if value else 0

# This function modifies the data in the specified columns
def modifyData(columns):
    for each_column in columns:
```

```
        # Convert the column values to strings
        data = [str(value) for value in dataset[each_column]]
        new_data = []

        for value in data:
            # If the value is 'nan', replace it with NaN
            if value == 'nan':
                new_data.append(float('nan'))
            else:
                # Otherwise, clean the value by removing underscores
                new_data.append(float(removeUnderscore(value)))

        # Update the dataset with the cleaned data
        dataset[each_column] = new_data

# Modify the specified columns in the dataset
modifyData(['Age', 'Annual_Income', 'Num_of_Loan', 'Num_of_Delayed_Payment',␣
 ↪'Outstanding_Debt',
           'Changed_Credit_Limit', 'Amount_invested_monthly',␣
 ↪'Monthly_Balance'])
```

[84]:
```
### Looking at the datatypes of the data
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 27 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  category
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  float64
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  category
 7   Annual_Income             100000 non-null  float64
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  float64
 13  Type_of_Loan              88592 non-null   category
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   float64
 16  Changed_Credit_Limit      100000 non-null  float64
 17  Num_Credit_Inquiries      98035 non-null   float64
```

```
 18  Credit_Mix                100000 non-null  category
 19  Outstanding_Debt          100000 non-null  float64
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  category
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   float64
 25  Payment_Behaviour         100000 non-null  category
 26  Monthly_Balance           98800 non-null   float64
dtypes: category(6), float64(12), int64(4), object(5)
memory usage: 16.9+ MB
```

[85]:
```
### Missing data by columns in the dataset
dataset.isnull().sum().sort_values(ascending = False)
```

[85]:
```
Monthly_Inhand_Salary       15002
Type_of_Loan                11408
Name                         9985
Credit_History_Age           9030
Num_of_Delayed_Payment       7002
Amount_invested_monthly      4479
Num_Credit_Inquiries         1965
Monthly_Balance              1200
Annual_Income                   0
Credit_Mix                      0
Payment_Behaviour               0
Month                           0
Total_EMI_per_month             0
Payment_of_Min_Amount           0
Age                             0
Credit_Utilization_Ratio        0
Outstanding_Debt                0
SSN                             0
Num_Bank_Accounts               0
Changed_Credit_Limit            0
Occupation                      0
Delay_from_due_date             0
Customer_ID                     0
Num_of_Loan                     0
Interest_Rate                   0
Num_Credit_Card                 0
ID                              0
dtype: int64
```

[86]:
```
### Summary statistics of the numerical columns in the dataset
dataset.describe()
```

[86]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts
count	100000.000000	1.000000e+05	84998.000000	100000.000000
mean	110.649700	1.764157e+05	4194.170850	17.091280
std	686.244717	1.429618e+06	3183.686167	117.404834
min	-500.000000	7.005930e+03	303.645417	-1.000000
25%	24.000000	1.945750e+04	1625.568229	3.000000
50%	33.000000	3.757861e+04	3093.745000	6.000000
75%	42.000000	7.279092e+04	5957.448333	7.000000
max	8698.000000	2.419806e+07	15204.633330	1798.000000

	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date
count	100000.00000	100000.000000	100000.000000	100000.000000
mean	22.47443	72.466040	3.009960	21.068780
std	129.05741	466.422621	62.647879	14.860104
min	0.00000	1.000000	-100.000000	-5.000000
25%	4.00000	8.000000	1.000000	10.000000
50%	5.00000	13.000000	3.000000	18.000000
75%	7.00000	20.000000	5.000000	28.000000
max	1499.00000	5797.000000	1496.000000	67.000000

	Num_of_Delayed_Payment	Changed_Credit_Limit	Num_Credit_Inquiries
count	92998.000000	100000.000000	98035.000000
mean	30.923342	10.171791	27.754251
std	226.031892	6.880628	193.177339
min	-3.000000	-6.490000	0.000000
25%	9.000000	4.970000	3.000000
50%	14.000000	9.250000	6.000000
75%	18.000000	14.660000	9.000000
max	4397.000000	36.970000	2597.000000

	Outstanding_Debt	Credit_Utilization_Ratio	Total_EMI_per_month
count	100000.000000	100000.000000	100000.000000
mean	1426.220376	32.285173	1403.118217
std	1155.129026	5.116875	8306.041270
min	0.230000	20.000000	0.000000
25%	566.072500	28.052567	30.306660
50%	1166.155000	32.305784	69.249473
75%	1945.962500	36.496663	161.224249
max	4998.070000	50.000000	82331.000000

	Amount_invested_monthly	Monthly_Balance
count	95521.000000	9.880000e+04
mean	637.412998	-3.036437e+22
std	2043.319327	3.181295e+24
min	0.000000	-3.333333e+26
25%	74.534002	2.700922e+02
50%	135.925681	3.367192e+02

```
75%                265.731733      4.702202e+02
max              10000.000000      1.602041e+03
```

** Univariate Analysis**

[87]:
```
# Univariate analysis for numerical columns
def univariate_numeric(data, column):
    plt.figure(figsize=(10, 6))

    # Drop NaN values
    data_clean = data[column].dropna()

    # Check if there are valid data points to plot
    if data_clean.empty:
        print(f"No valid data to plot for {column}. Skipping...")
        plt.close()  # Close the figure
        return

    # Histogram
    plt.subplot(1, 2, 1)
    plt.hist(data_clean, bins=20, color='skyblue', edgecolor='black')
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

    # Kernel Density Estimation (KDE)
    plt.subplot(1, 2, 2)
    sns.kdeplot(data_clean, shade=True, color='green')
    plt.title(f'KDE of {column}')
    plt.xlabel(column)

    plt.tight_layout()
    plt.show()

# Univariate analysis for categorical columns
def univariate_categorical(data, column):
    plt.figure(figsize=(10, 6))

    # Drop NaN values and count value occurrences
    value_counts = data[column].dropna().value_counts()

    # Check if there are valid categories to plot
    if value_counts.empty:
        print(f"No valid data to plot for {column}. Skipping...")
        plt.close()  # Close the figure
        return
```

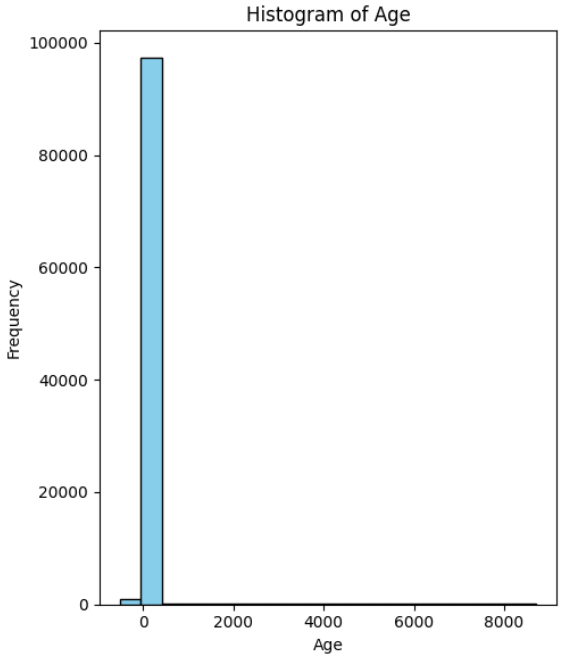
```
    plt.bar(value_counts.index, value_counts.values, color='purple',␣
 ↪edgecolor='black')
    plt.title(f'Bar Plot of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=90)

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

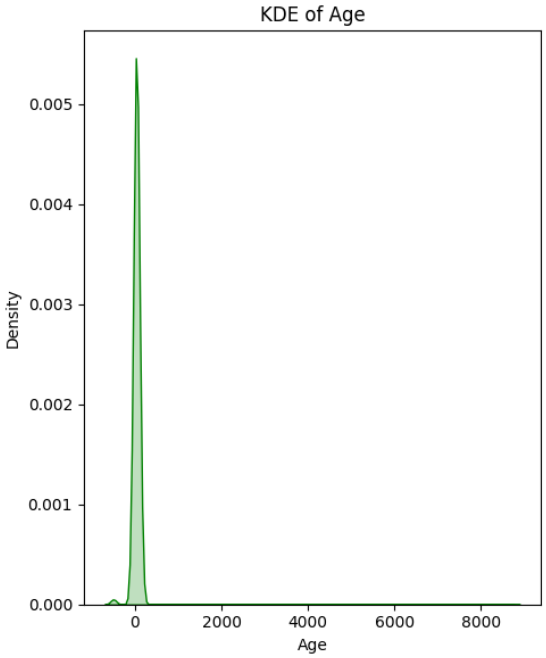
Univariate Analysis for numeric Column

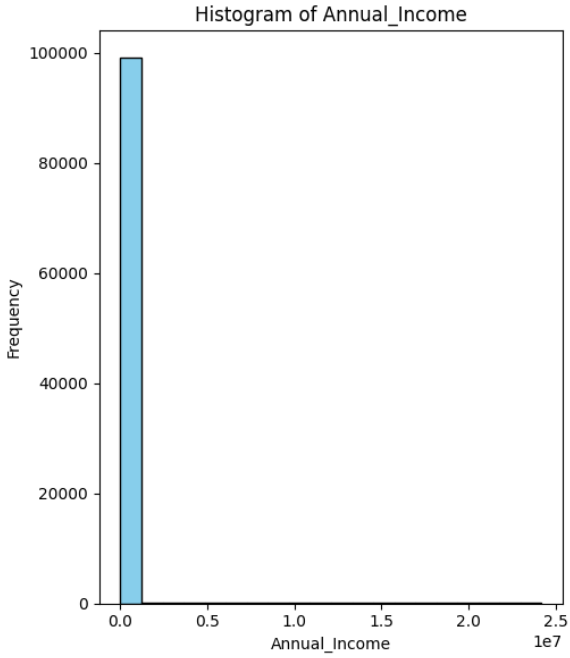
1. Annual_Income
2. Monthly_Inhand_Salary
3. Interest_Rate
4. Num_of_Loan
5. Num_of_Delayed_Payment
6. Outstanding_Debt
7. Credit_Utilization_Ratio
8. Credit_History_Age
9. Total_EMI_per_month
10. Amount_invested_monthly

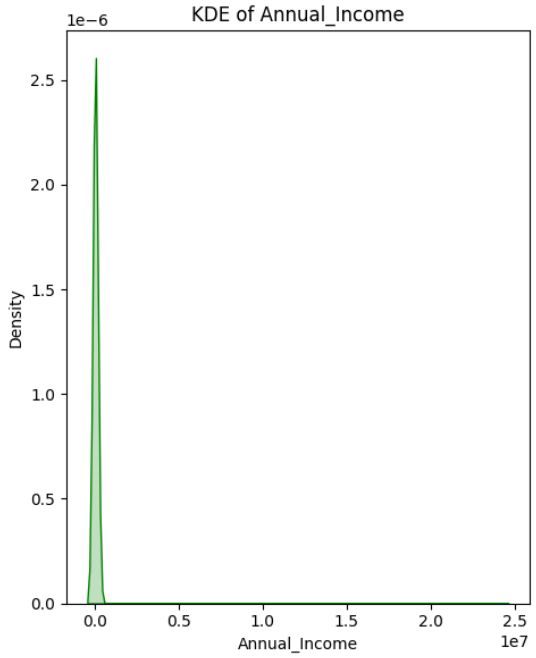
[88]:
```
# List of columns to analyze
numeric_columns = ['Age',
    'Annual_Income',
    'Monthly_Inhand_Salary',
    'Interest_Rate',
    'Num_of_Loan',
    'Num_of_Delayed_Payment',
    'Outstanding_Debt',
    'Credit_Utilization_Ratio',
    'Credit_History_Age',
    'Total_EMI_per_month',
    'Amount_invested_monthly'
]
```

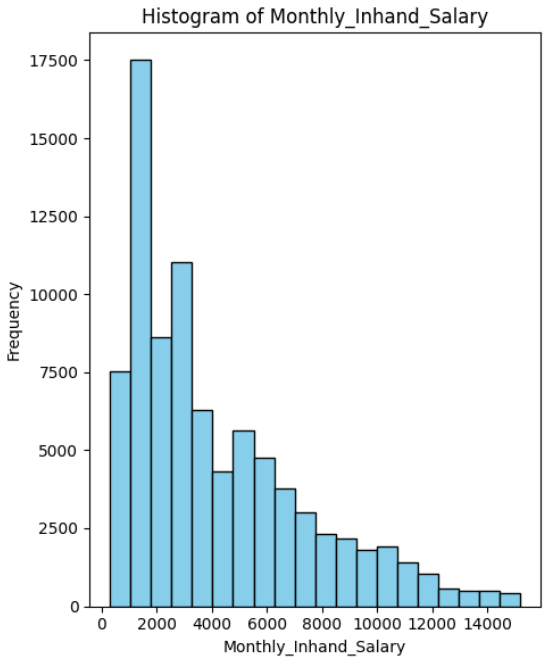
[89]:
```
# Convert to numeric, forcing errors to NaN
dataset[numeric_columns] = dataset[numeric_columns].apply(pd.to_numeric,␣
 ↪errors='coerce')

for col in numeric_columns:
    univariate_numeric(dataset, col)
```

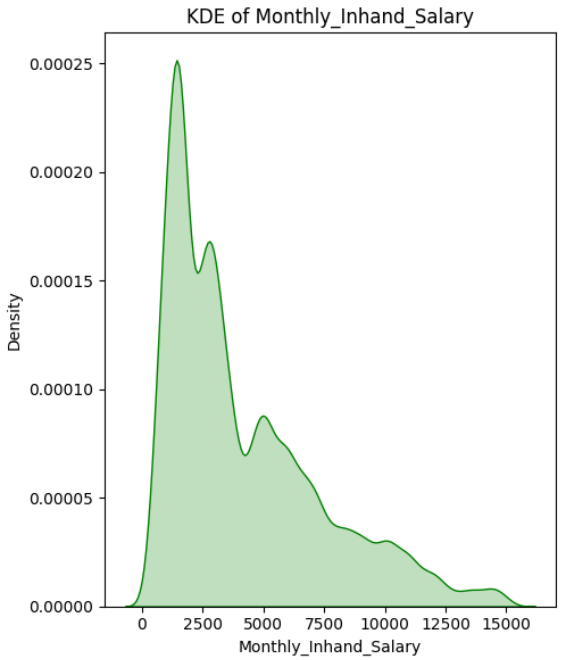

Histogram of Age
Frequency
Age
0
20000
40000
60000
80000
100000
0
2000
4000
6000
8000

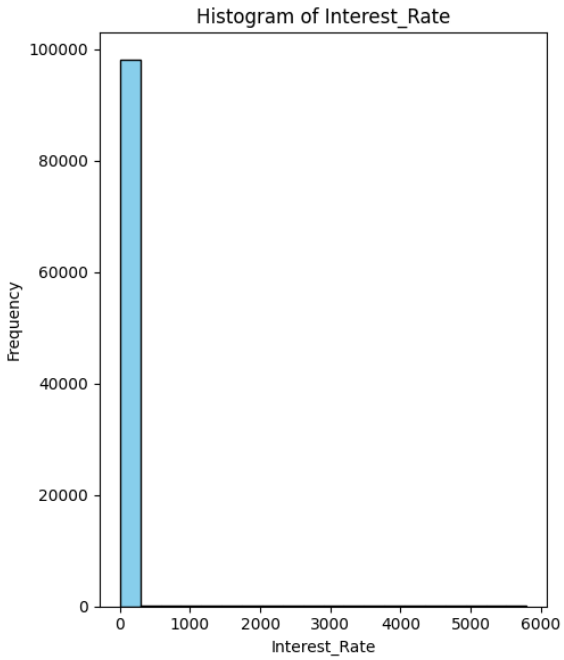

KDE of Age
Density
Age
0.000
0.001
0.002
0.003
0.004
0.005
0
2000
4000
6000
8000

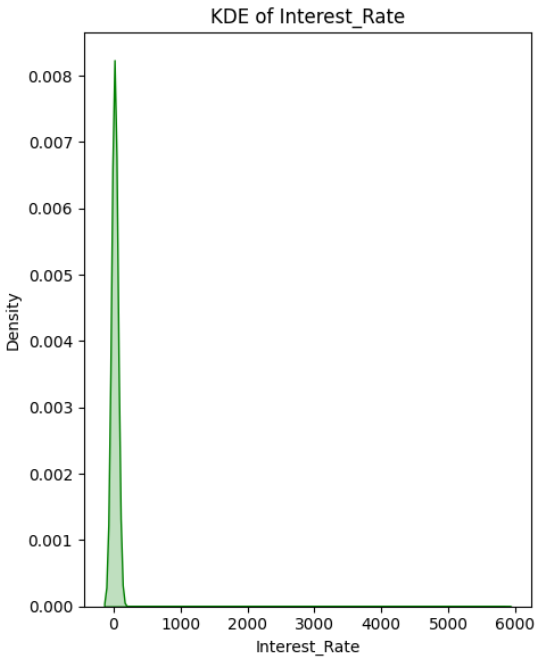

Histogram of Annual_Income
Frequency
Annual_Income
1e7
0
20000
40000
60000
80000
100000
0.0
0.5
1.0
1.5
2.0
2.5

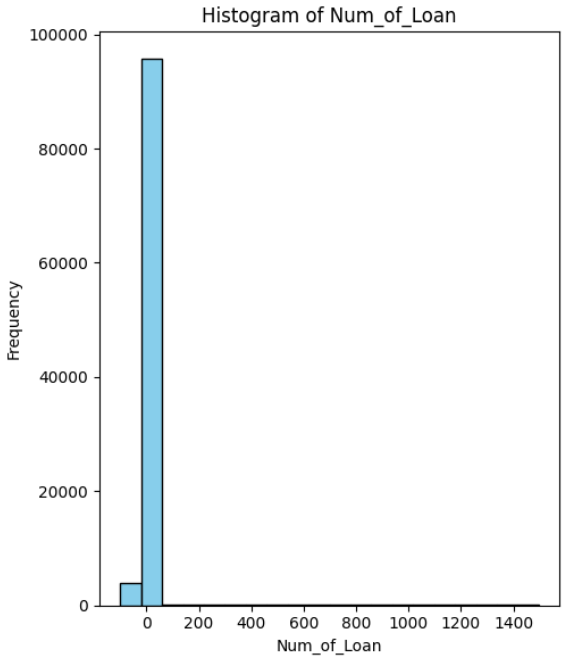

KDE of Annual_Income
1e−6
Density
Annual_Income
1e7
0.0
0.5
1.0
1.5
2.0
2.5

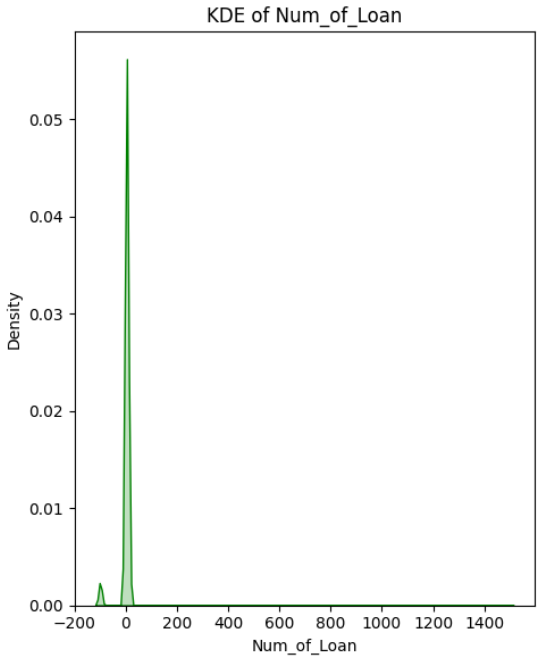

Histogram of Monthly_Inhand_Salary
Frequency
Monthly_Inhand_Salary
0
2500
5000
7500
10000
12500
15000
17500
0
2000
4000
6000
8000
10000
12000
14000

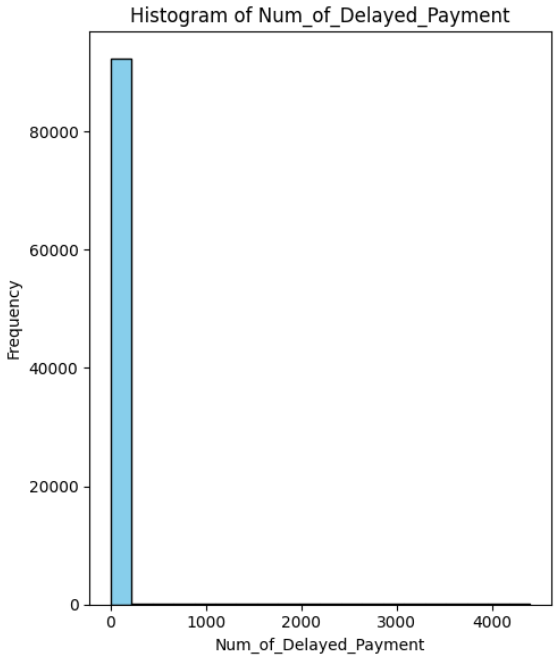

KDE of Monthly_Inhand_Salary
Density
Monthly_Inhand_Salary
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025
0
2500
5000
7500
10000
12500
15000

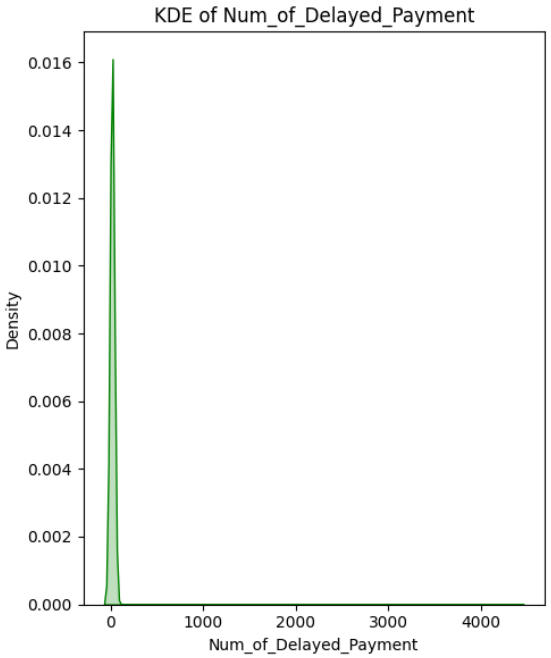

Histogram of Interest_Rate
Frequency
Interest_Rate
0
20000
40000
60000
80000
100000
0
1000
2000
3000
4000
5000
6000

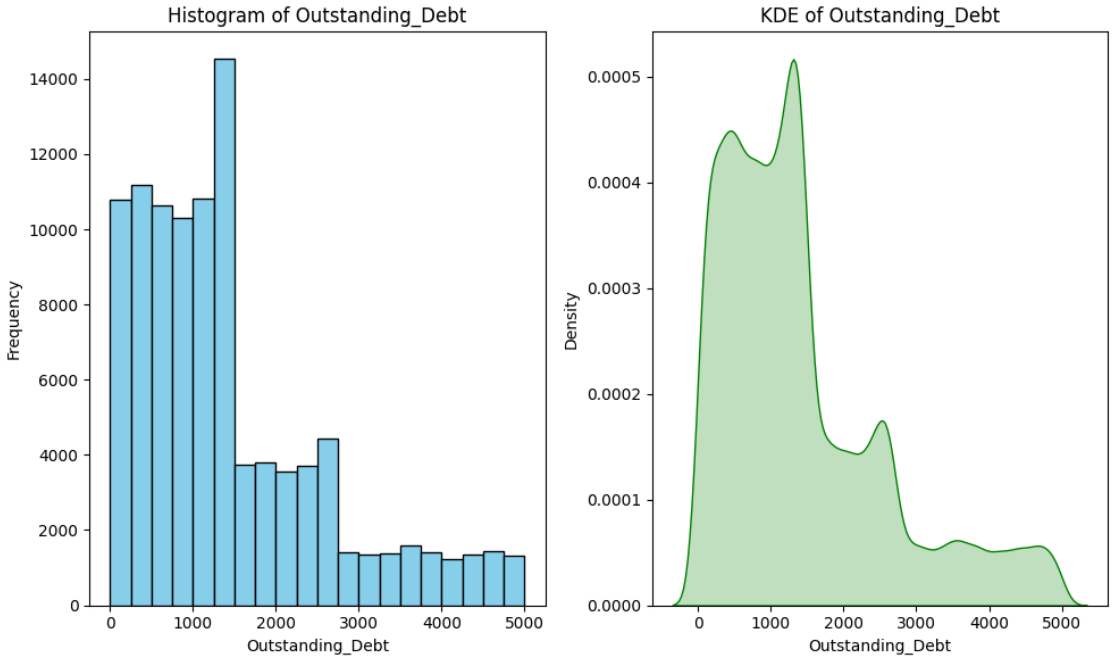

KDE of Interest_Rate
Density
Interest_Rate
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
0
1000
2000
3000
4000
5000
6000

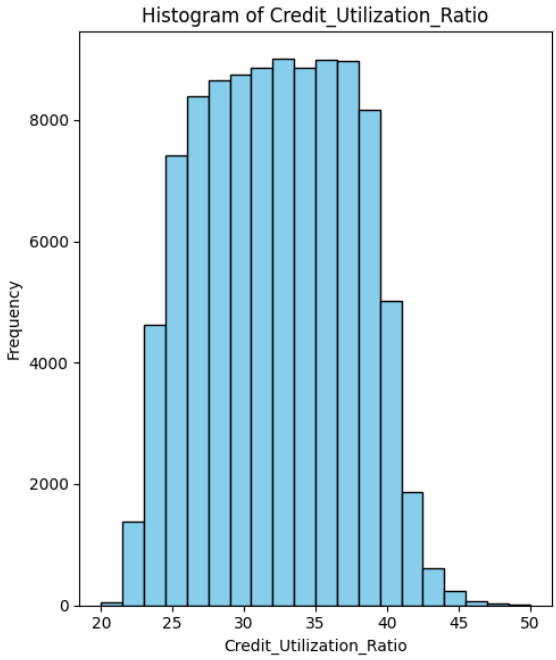

Histogram of Num_of_Loan
Frequency
Num_of_Loan
0
20000
40000
60000
80000
100000
0
200
400
600
800
1000
1200
1400

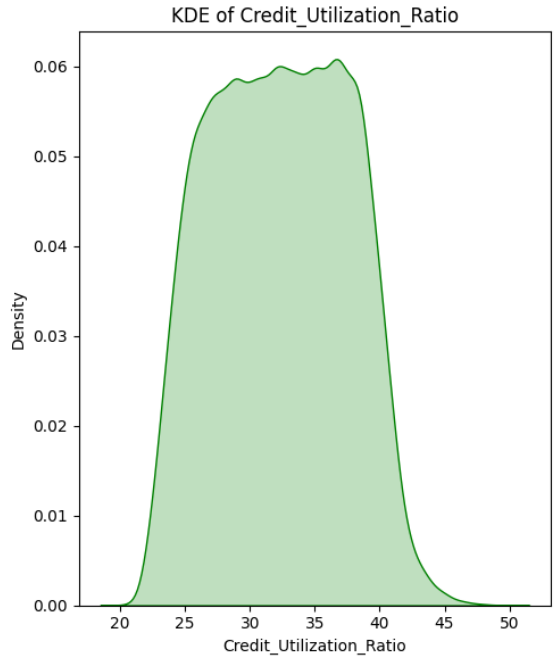

KDE of Num_of_Loan
Density
Num_of_Loan
0.00
0.01
0.02
0.03
0.04
0.05
−200
0
200
400
600
800
1000
1200
1400


Histogram of Num_of_Delayed_Payment
Frequency
Num_of_Delayed_Payment
0
20000
40000
60000
80000
0
1000
2000
3000
4000

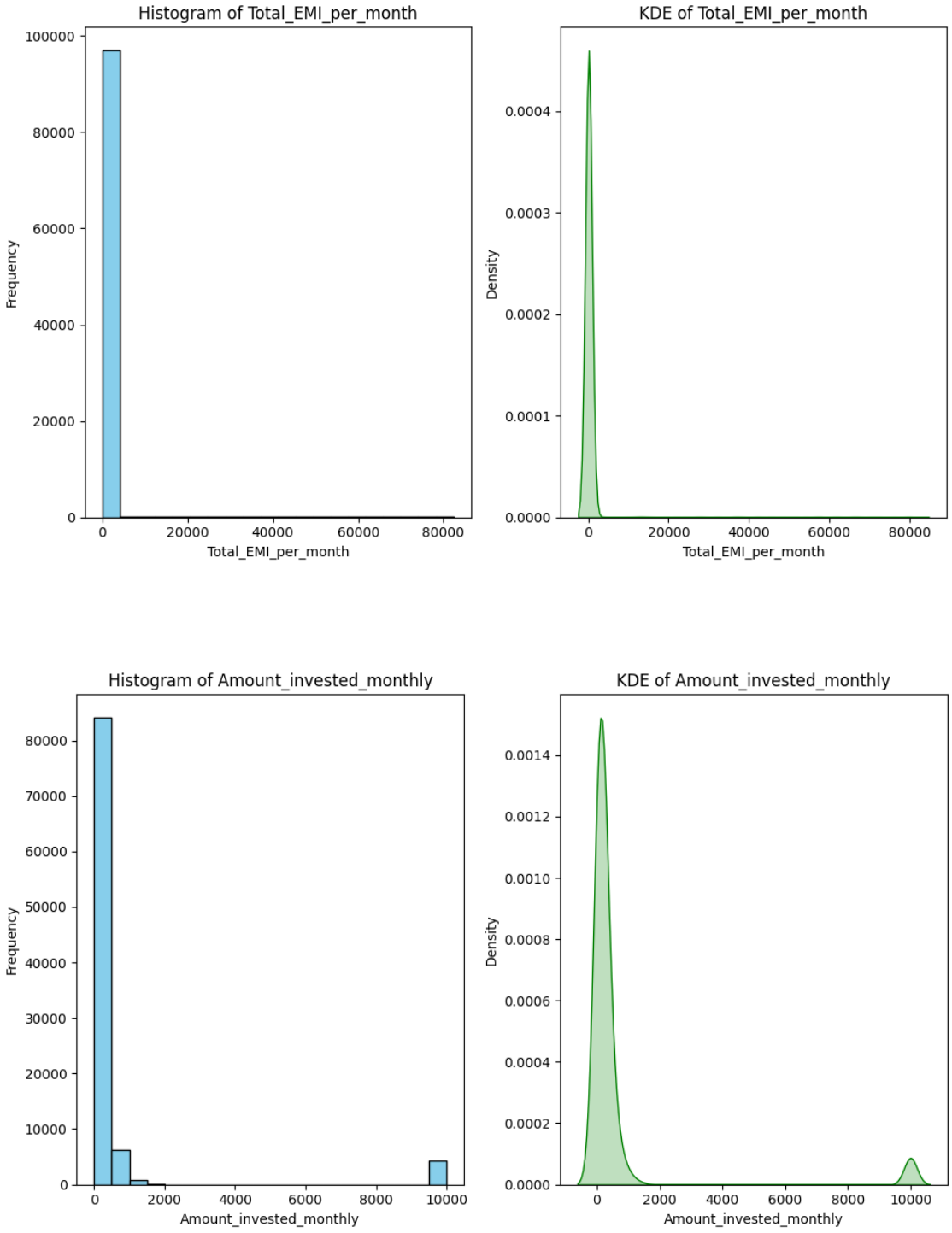

KDE of Num_of_Delayed_Payment
Density
Num_of_Delayed_Payment
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
0.016
0
1000
2000
3000
4000







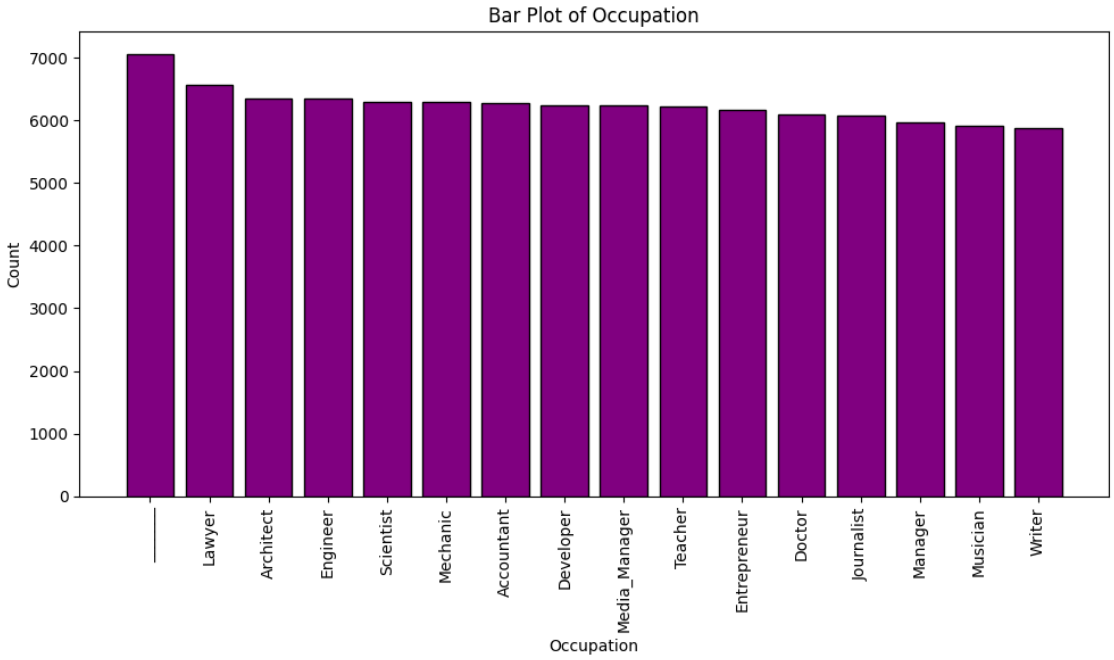
No valid data to plot for Credit_History_Age. Skipping…

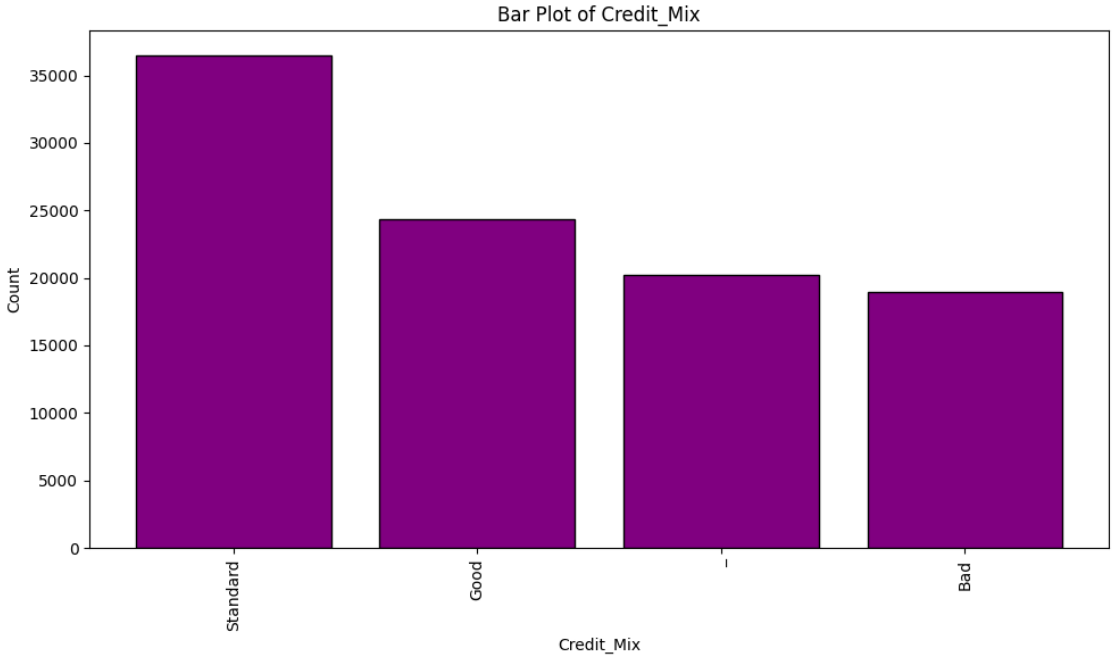


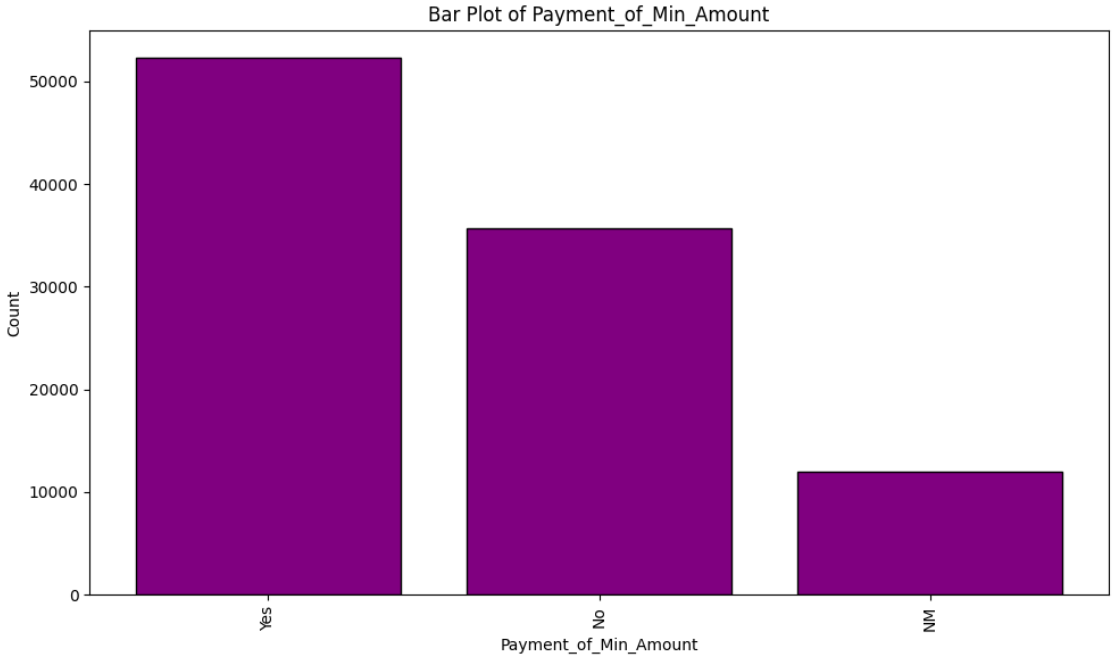
Univariate analysis for categorical column 1. Occupation 2. Type_of_Loan 3. Credit_Mix 4. Payment_of_Min_Amount 5. Payment_Behaviour

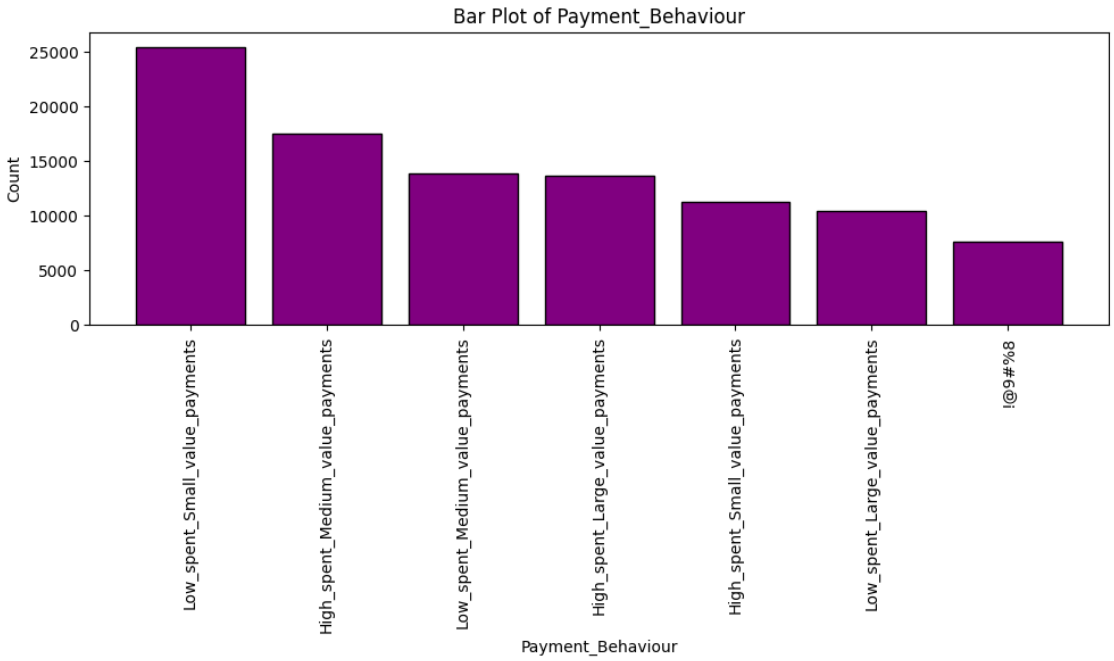
[90]:
```
categorical_columns = [
    'Occupation',
'Credit_Mix',
'Payment_of_Min_Amount',
'Payment_Behaviour'

]
```

[91]:
```
for col in categorical_columns:
    univariate_categorical(dataset, col)
```




Bar Plot of Credit_Mix
Count
0
5000
10000
15000
20000
25000
30000
35000
Standard
Good
_
Bad
Credit_Mix


Bar Plot of Payment_of_Min_Amount
Count
0
10000
20000
30000
40000
50000
Yes
No
NM
Payment_of_Min_Amount

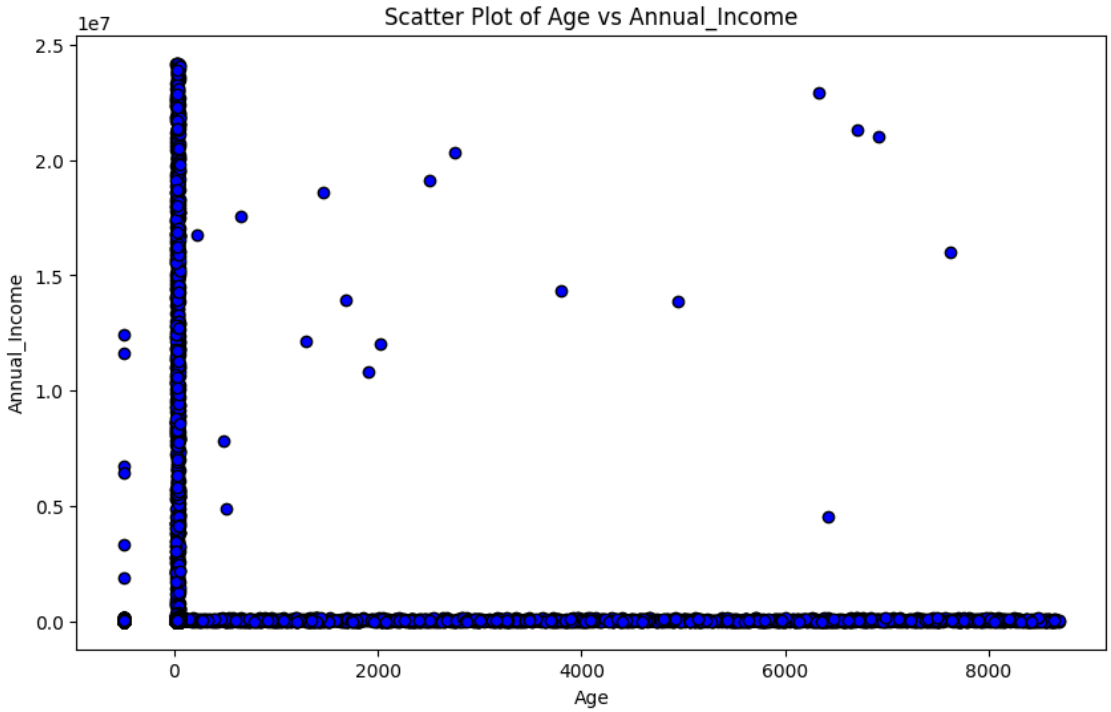


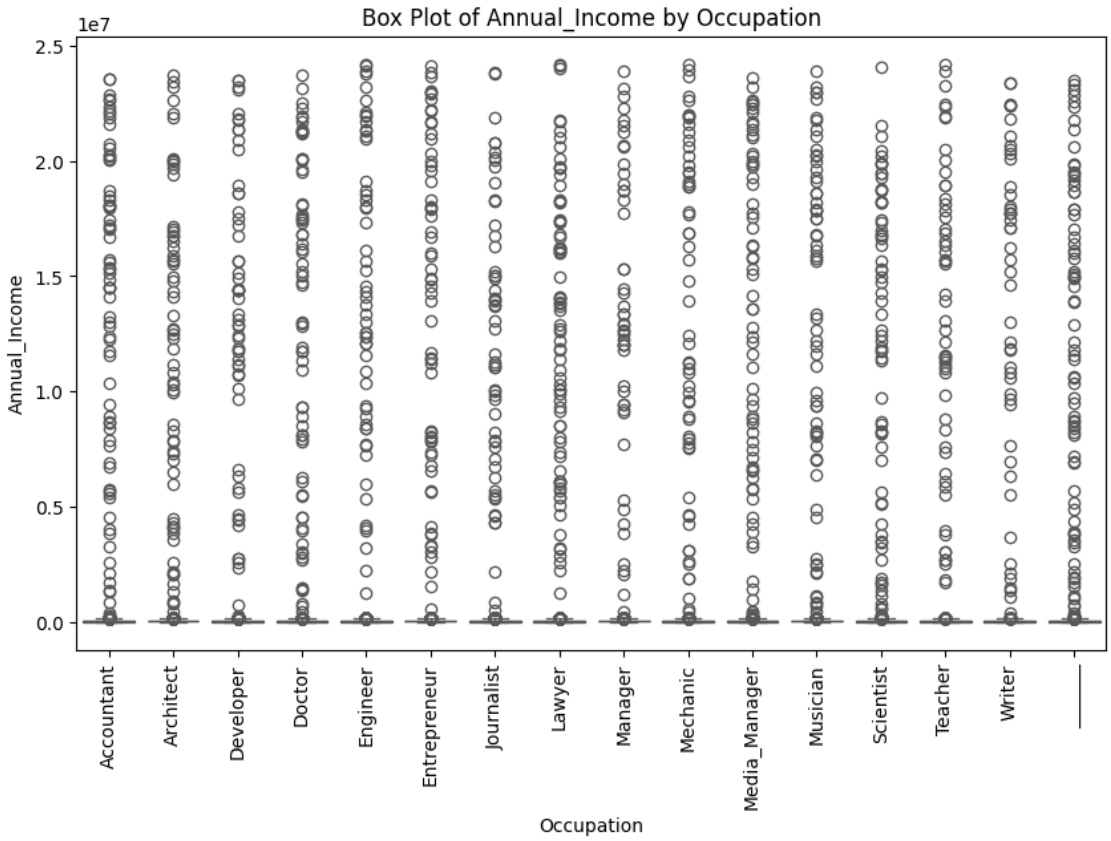
Bivariate Analysis

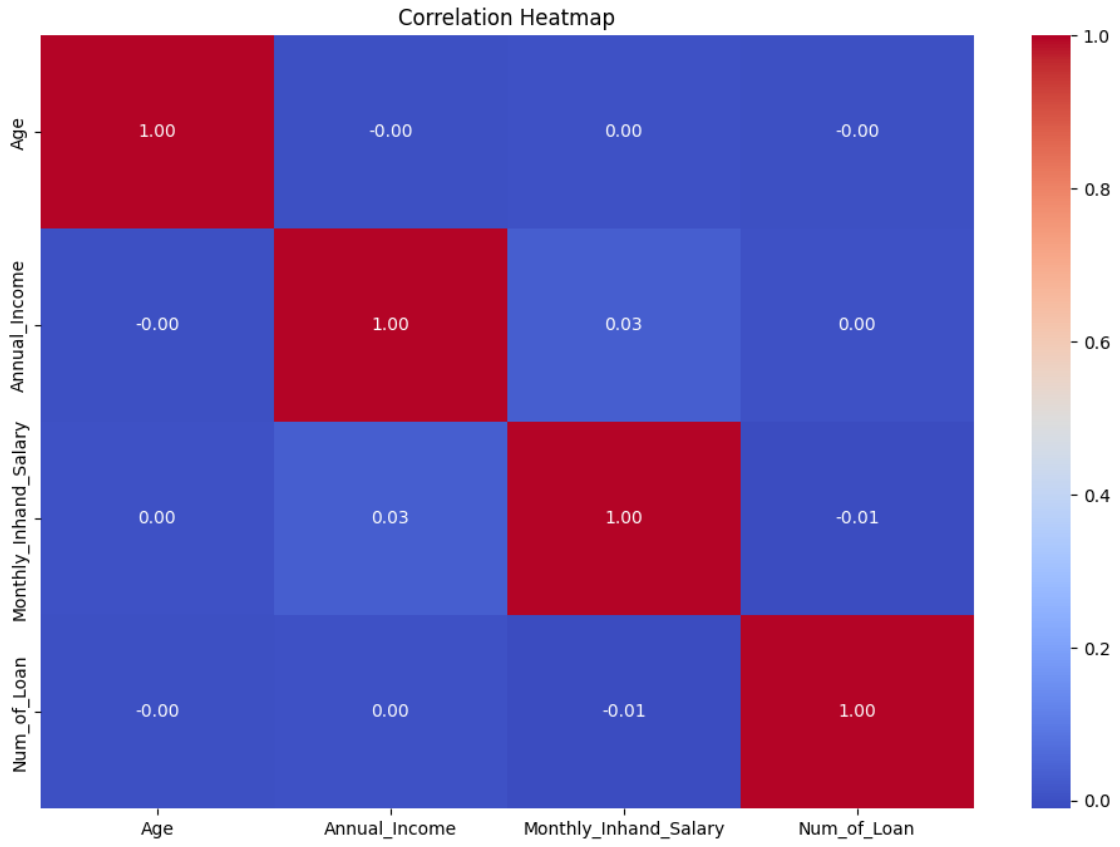
[92]:
```
# Bivariate analysis: Numerical-Numerical (Scatter Plot)
def bivariate_numeric_numeric(data, col1, col2):
    plt.figure(figsize=(10, 6))
    plt.scatter(data[col1], data[col2], color='blue', edgecolor='black')
    plt.title(f'Scatter Plot of {col1} vs {col2}')
    plt.xlabel(col1)
    plt.ylabel(col2)
    plt.show()

# Bivariate analysis: Categorical-Numerical (Box Plot)
def bivariate_categorical_numeric(data, cat_col, num_col):
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=cat_col, y=num_col, data=data, palette='Set2')
    plt.title(f'Box Plot of {num_col} by {cat_col}')
    plt.xlabel(cat_col)
    plt.ylabel(num_col)
    plt.xticks(rotation=90)
    plt.show()

# Bivariate analysis: Correlation Heatmap
def correlation_heatmap(data, num_cols):
    plt.figure(figsize=(12, 8))
    corr = data[num_cols].corr()
    sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
```

```
    plt.title('Correlation Heatmap')
    plt.show()
```

[93]:
```
# Scatter Plot for numerical columns
bivariate_numeric_numeric(dataset, 'Age', 'Annual_Income')

# Box Plot for categorical and numerical columns
bivariate_categorical_numeric(dataset, 'Occupation', 'Annual_Income')

# Correlation heatmap for numerical columns
numerical_cols = ['Age', 'Annual_Income', 'Monthly_Inhand_Salary',␣
 ↪'Num_of_Loan']
correlation_heatmap(dataset, numerical_cols)
```




Box Plot of Annual_Income by Occupation
1e7
2.5
2.0
1.5
1.0
0.5
0.0
Annual_Income
Accountant
Architect
Developer
Doctor
Engineer
Entrepreneur
Journalist
Lawyer
Manager
Mechanic
Media_Manager
Musician
Scientist
Teacher
Writer
_____
Occupation



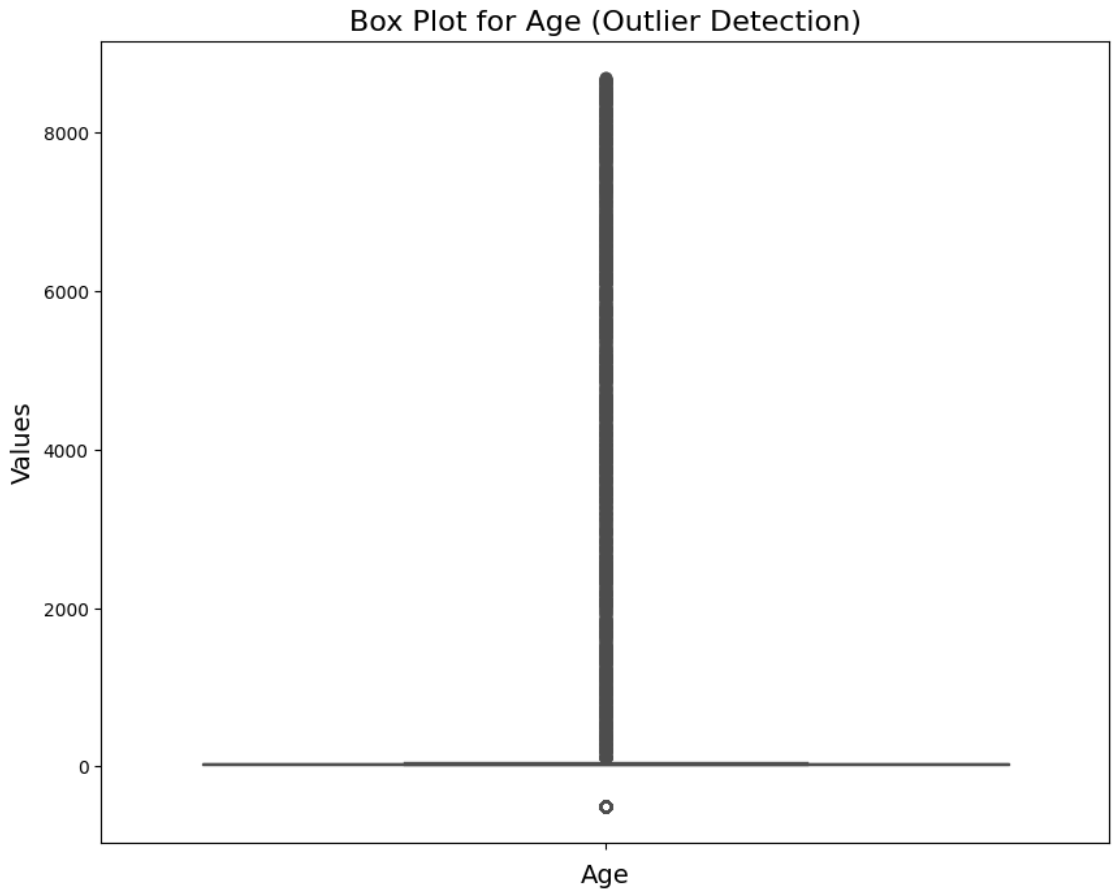
Outlier Detection using Boxplot

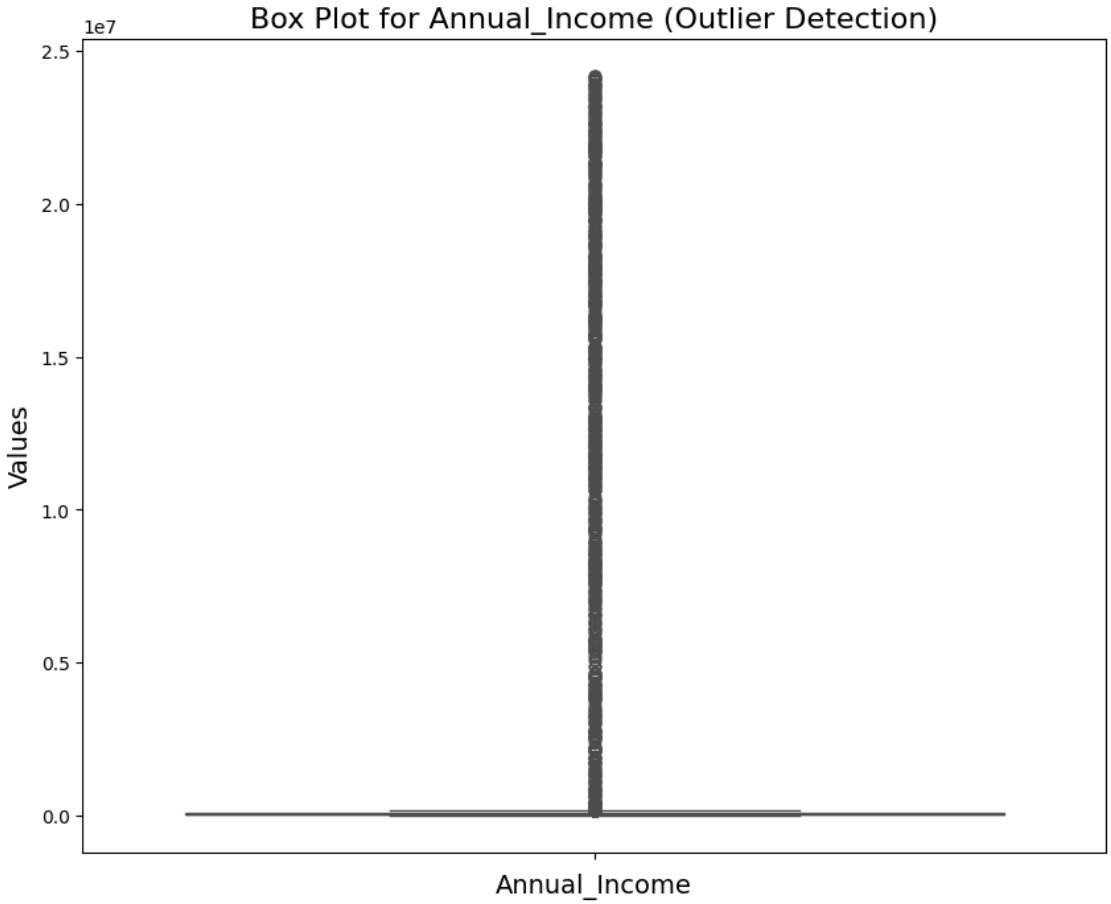
[94]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def detect_outliers(data, column):
    plt.figure(figsize=(10, 8))  # Adjusted figure size for better visibility
    sns.boxplot(data[column], color='orange')
    plt.title(f'Box Plot for {column} (Outlier Detection)', fontsize=16)
    plt.xlabel(column, fontsize=14)
    plt.ylabel('Values', fontsize=14)
    plt.show()
```

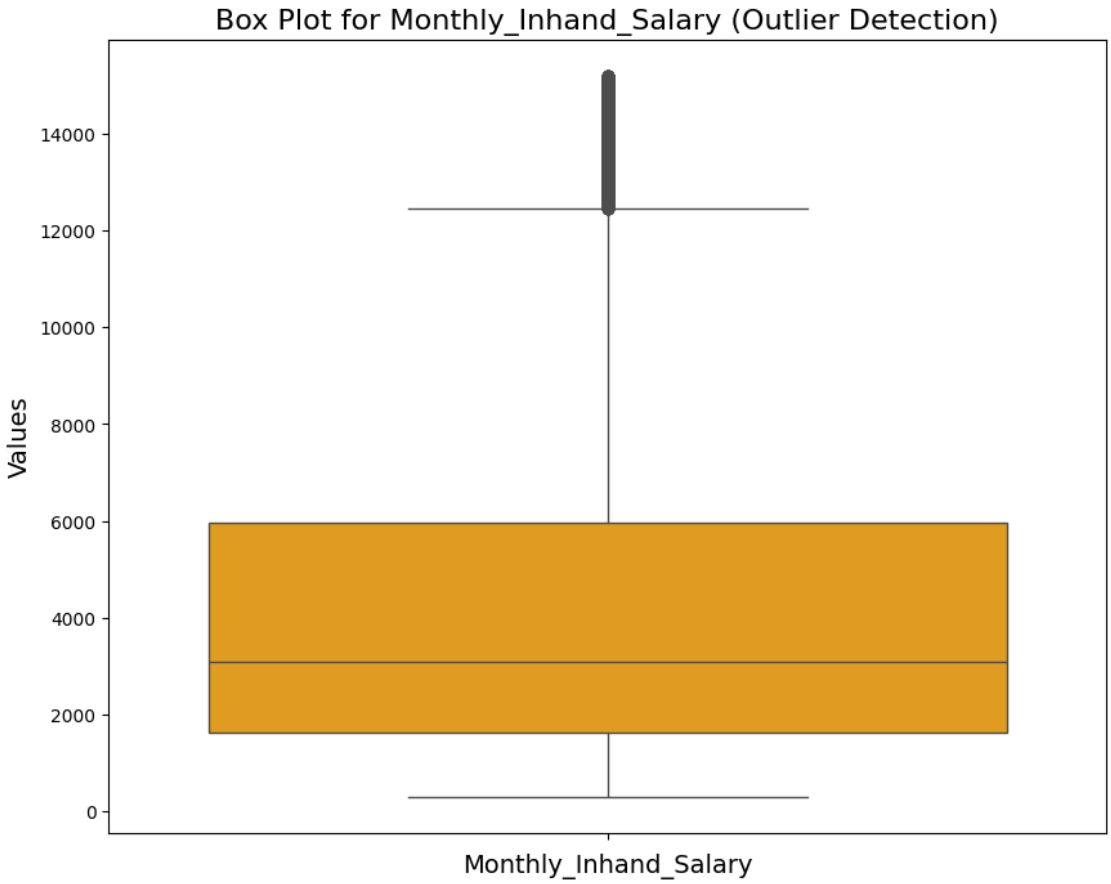
[95]:
```
# List of columns to analyze
numeric_columns1 = ['Age',
    'Annual_Income',
    'Monthly_Inhand_Salary',
    'Interest_Rate',
    'Num_of_Loan',
    'Num_of_Delayed_Payment',
    'Outstanding_Debt',
```

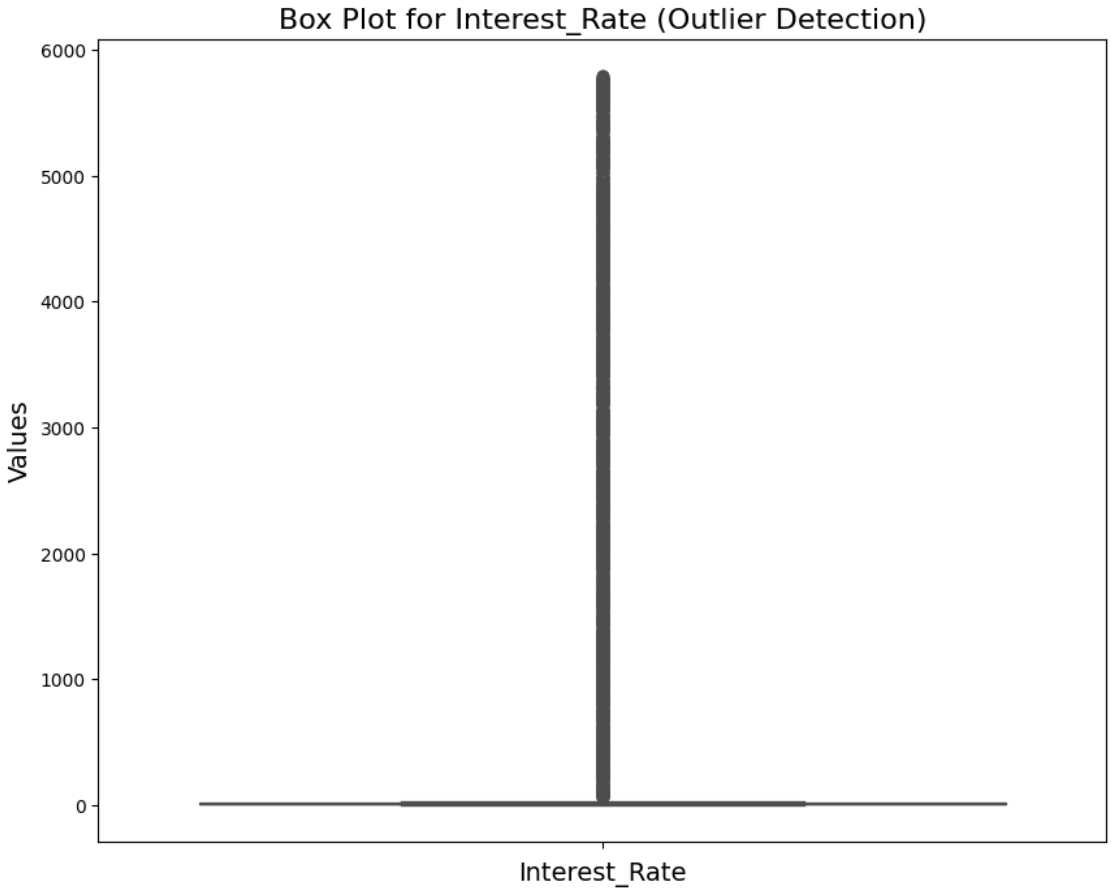
```
    'Credit_Utilization_Ratio'

]
```

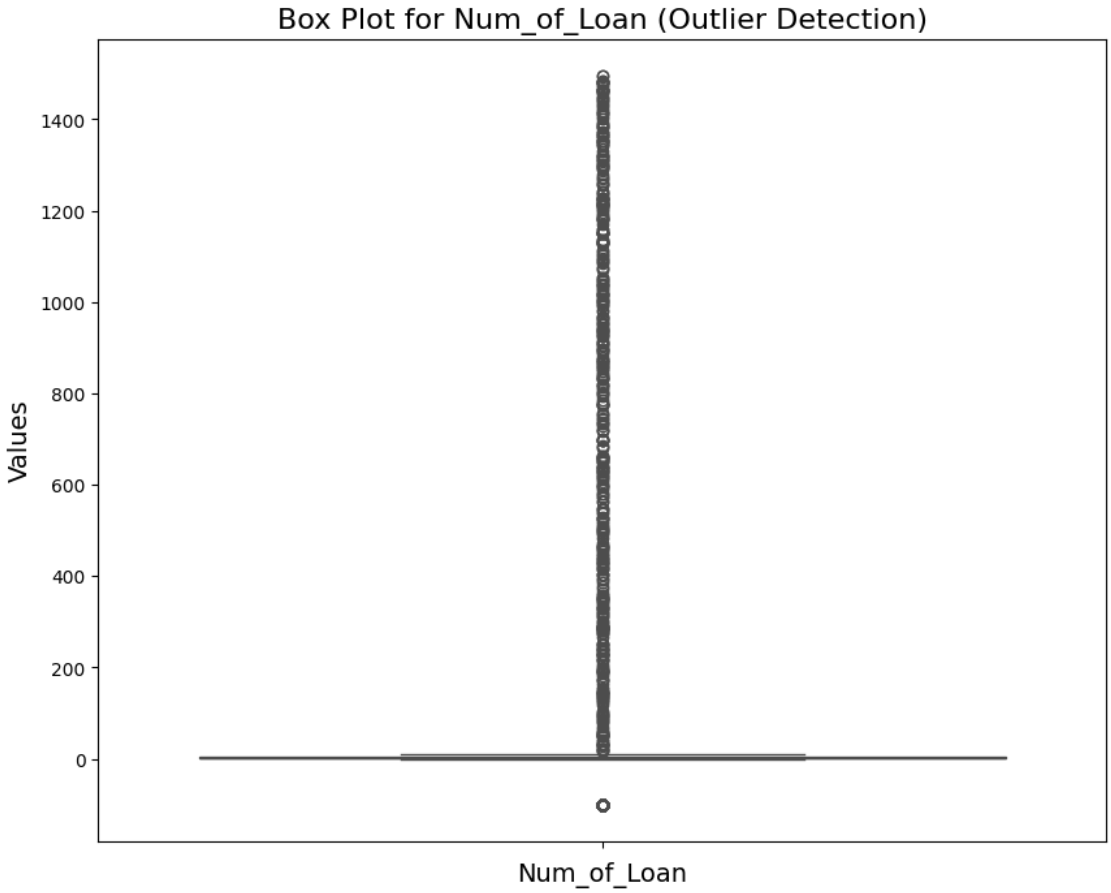
[96]:
```
for col in numeric_columns1:
    detect_outliers(dataset, col)
```

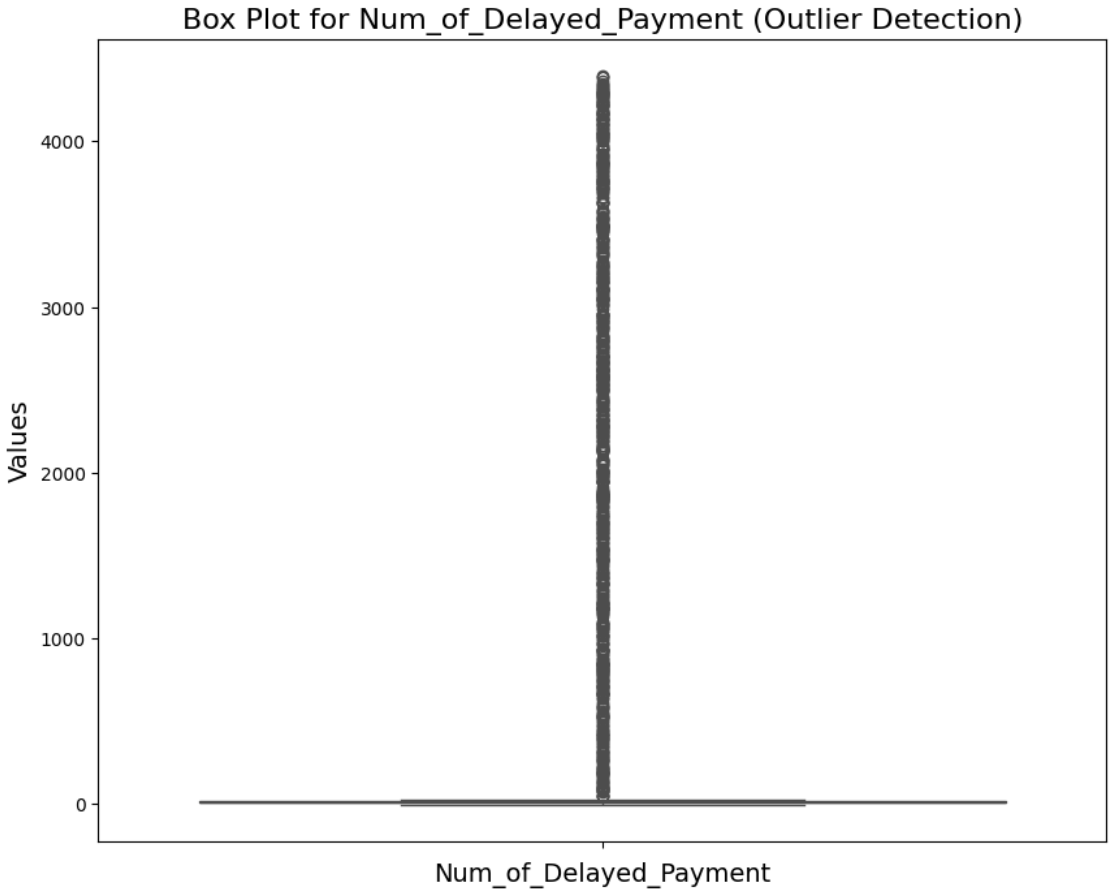


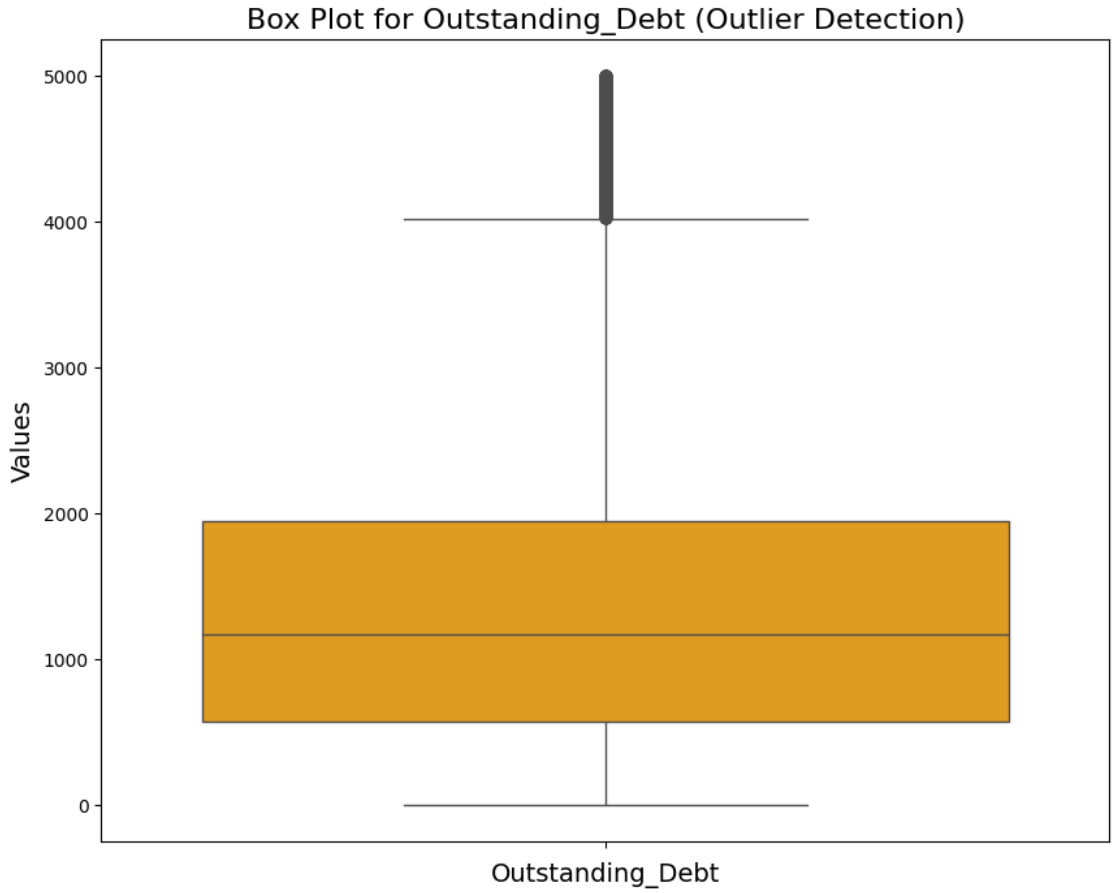

Box Plot for Annual_Income (Outlier Detection)
1e7
2.5
2.0
1.5
1.0
0.5
0.0
Values
Annual_Income

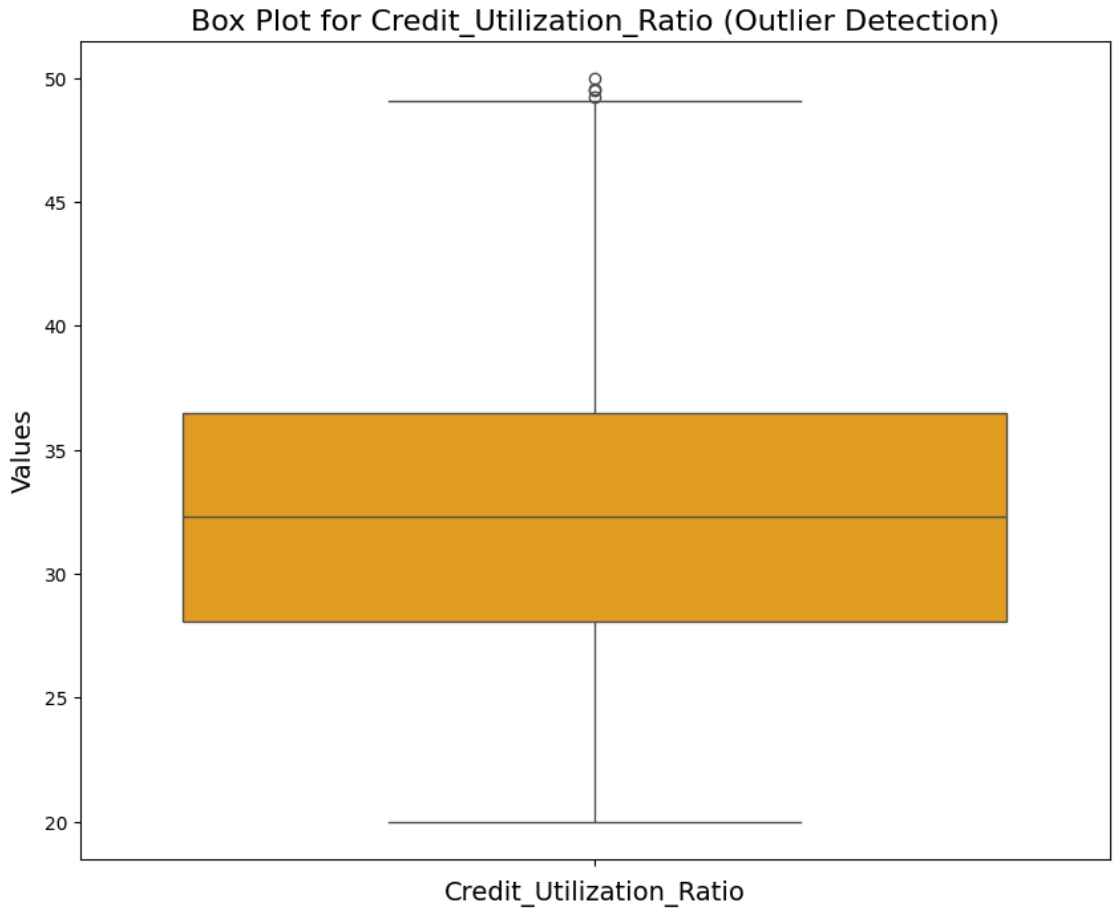

Box Plot for Monthly_Inhand_Salary (Outlier Detection)
Values
14000
12000
10000
8000
6000
4000
2000
0
Monthly_Inhand_Salary


Box Plot for Interest_Rate (Outlier Detection)
Values
6000
5000
4000
3000
2000
1000
0
Interest_Rate


Box Plot for Num_of_Loan (Outlier Detection)
Values
1400
1200
1000
800
600
400
200
0
Num_of_Loan


Box Plot for Num_of_Delayed_Payment (Outlier Detection)
Values
4000
3000
2000
1000
0
Num_of_Delayed_Payment


Box Plot for Outstanding_Debt (Outlier Detection)
Values
5000
4000
3000
2000
1000
0
Outstanding_Debt



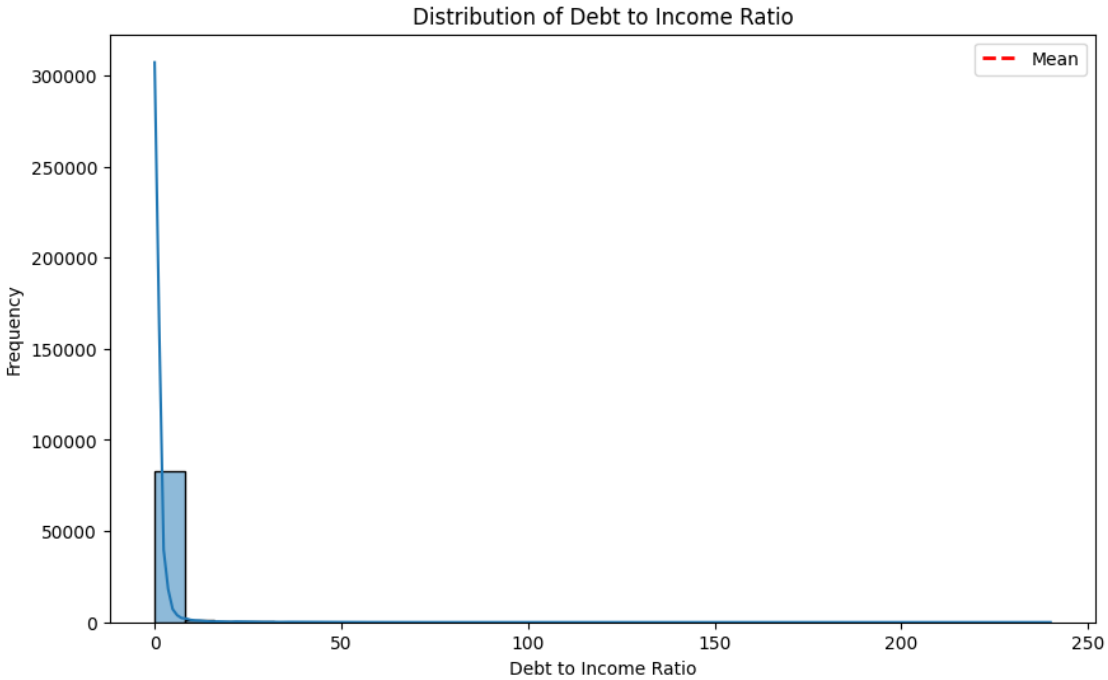
Feature Engineering for Credit Scores through Domain Knowledge and EDA Insights

[97]:
```
# Convert relevant columns to numeric
columns_to_convert = ['Total_EMI_per_month', 'Outstanding_Debt',␣
 ↪'Monthly_Inhand_Salary', 'Changed_Credit_Limit', 'Num_Credit_Card']
for col in columns_to_convert:
    dataset[col] = pd.to_numeric(dataset[col], errors='coerce')

# Fill NaN values only in numeric columns
numeric_cols = dataset.select_dtypes(include=['number']).columns
dataset[numeric_cols] = dataset[numeric_cols].fillna(0)

# Now create the features
dataset['Debt_to_Income_Ratio'] = (dataset['Total_EMI_per_month'] +␣
 ↪dataset['Outstanding_Debt']) / dataset['Monthly_Inhand_Salary']
dataset['Credit_Utilization_Ratio'] = dataset['Outstanding_Debt'] /␣
 ↪(dataset['Num_Credit_Card'] * dataset['Changed_Credit_Limit'])
```

```
dataset['Payment_Behavior_Score'] = dataset['Total_EMI_per_month'] /␣
 ↪(dataset['Num_of_Delayed_Payment'] + dataset['Total_EMI_per_month'])
dataset['Delinquency_Ratio'] = dataset['Num_of_Delayed_Payment'] /␣
 ↪(dataset['Num_of_Loan'] + 1)
dataset['Monthly_Savings_Rate'] = dataset['Monthly_Balance'] /␣
 ↪dataset['Monthly_Inhand_Salary']

# Categorize 'Age' after handling NaNs
dataset['Age'] = dataset['Age'].fillna(0)  # Fill NaN values with 0 before␣
 ↪categorizing
dataset['Age_Group'] = pd.cut(dataset['Age'], bins=[0, 24, 34, 44, 54, 64, 74,␣
 ↪100], labels=['<25', '25-34', '35-44', '45-54', '55-64', '65-74', '75+'])
```

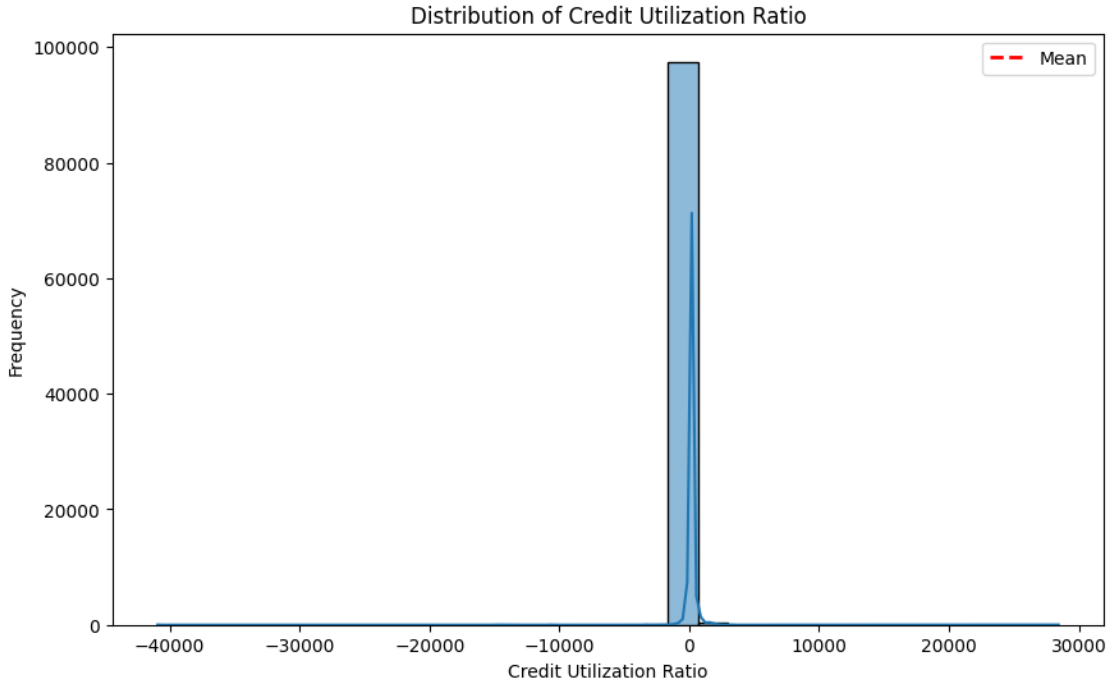
Visualization of Debt to Income Ratio

[98]:
```
# Distribution of Debt to Income Ratio
plt.figure(figsize=(10, 6))
sns.histplot(dataset['Debt_to_Income_Ratio'], bins=30, kde=True)
plt.title('Distribution of Debt to Income Ratio')
plt.xlabel('Debt to Income Ratio')
plt.ylabel('Frequency')
plt.axvline(dataset['Debt_to_Income_Ratio'].mean(), color='r',␣
 ↪linestyle='dashed', linewidth=2, label='Mean')
plt.legend()
plt.show()
```



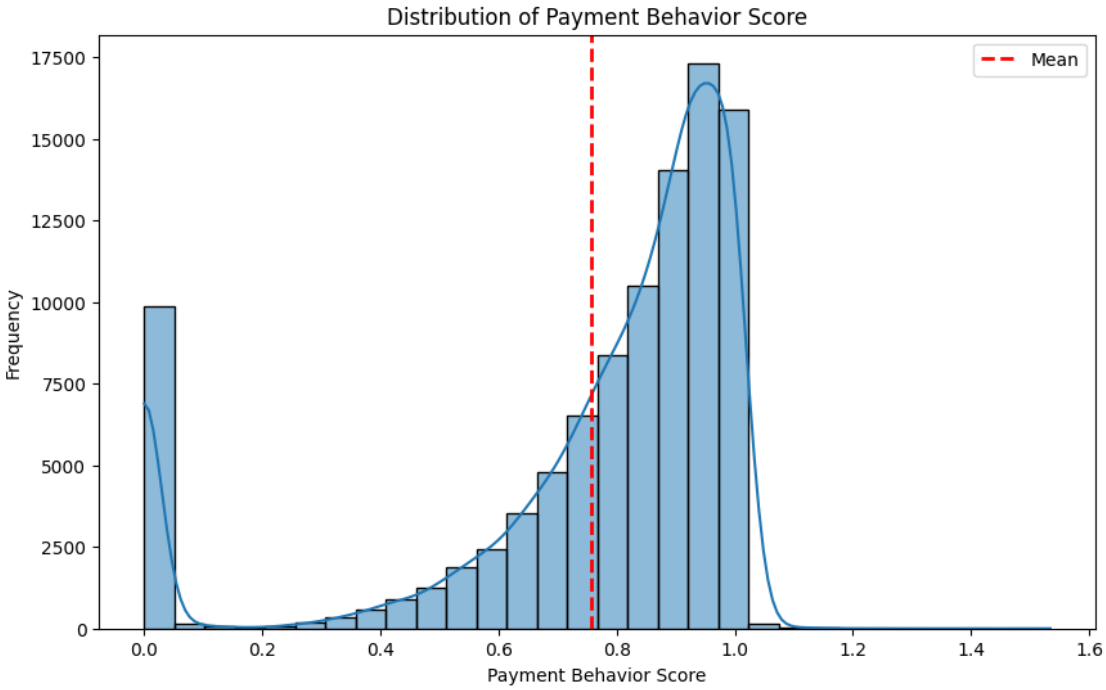
Visualization of Credit Utilization Ratio

[99]:
```
# Distribution of Credit Utilization Ratio
plt.figure(figsize=(10, 6))
sns.histplot(dataset['Credit_Utilization_Ratio'], bins=30, kde=True)
plt.title('Distribution of Credit Utilization Ratio')
plt.xlabel('Credit Utilization Ratio')
plt.ylabel('Frequency')
plt.axvline(dataset['Credit_Utilization_Ratio'].mean(), color='r',␣
 ↪linestyle='dashed', linewidth=2, label='Mean')
plt.legend()
plt.show()
```



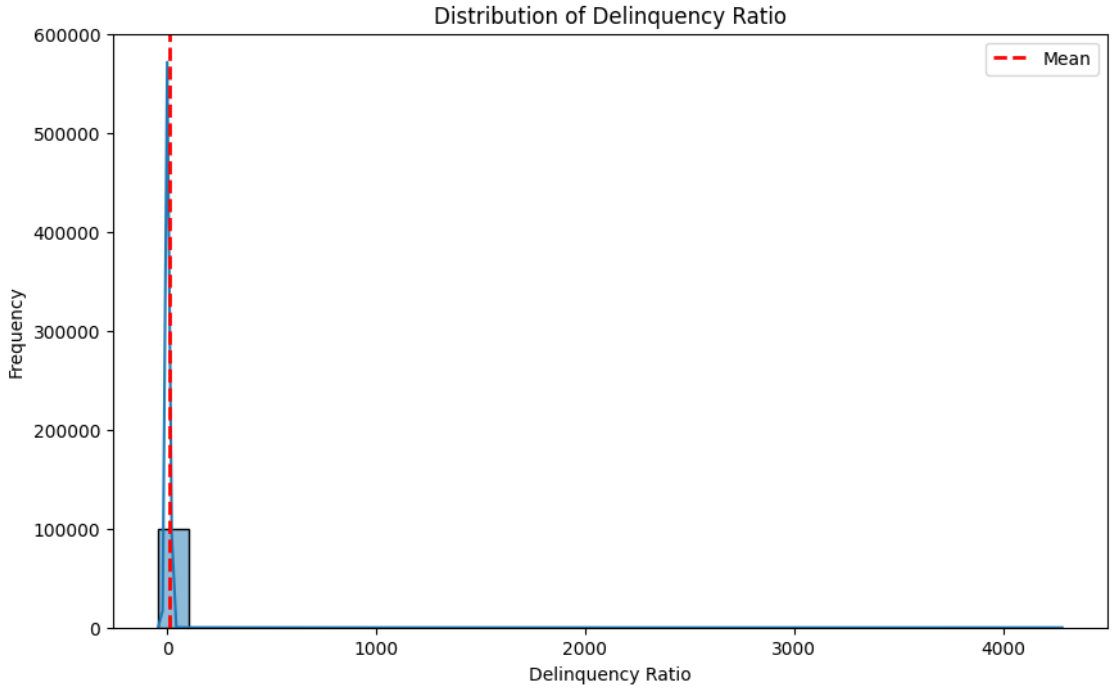
Visualization of Payment Behavior Score

[100]:
```
# Distribution of Payment Behavior Score
plt.figure(figsize=(10, 6))
sns.histplot(dataset['Payment_Behavior_Score'], bins=30, kde=True)
plt.title('Distribution of Payment Behavior Score')
plt.xlabel('Payment Behavior Score')
plt.ylabel('Frequency')
plt.axvline(dataset['Payment_Behavior_Score'].mean(), color='r',␣
 ↪linestyle='dashed', linewidth=2, label='Mean')
plt.legend()
plt.show()
```



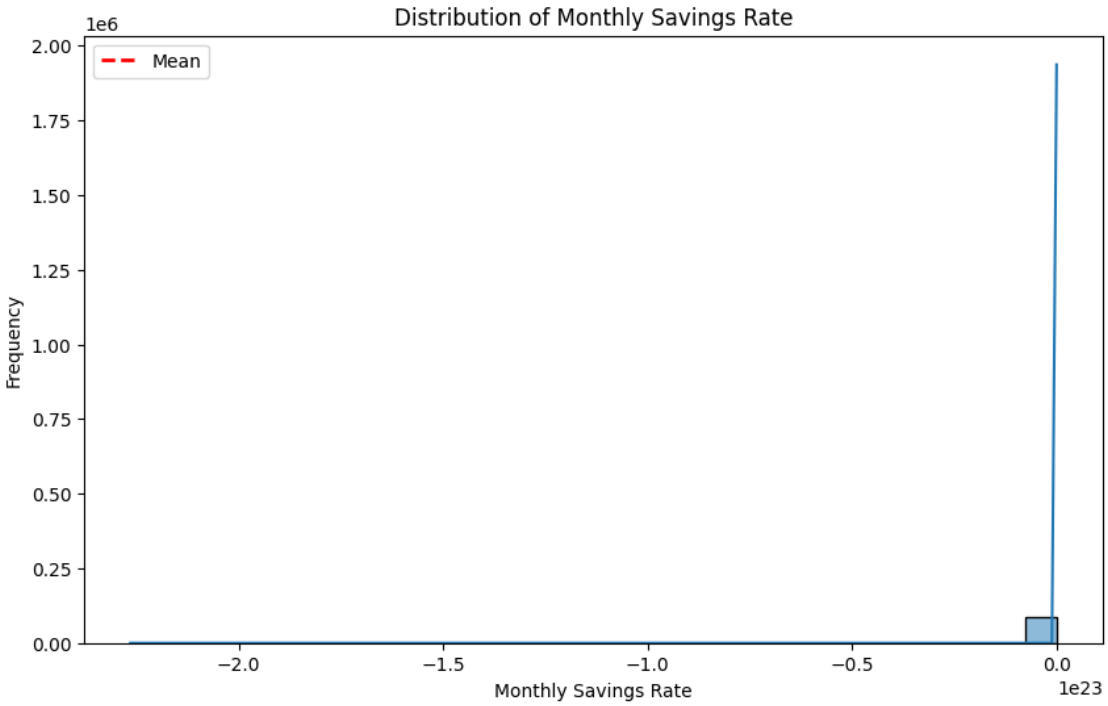
Visualization of Delinquency Ratio

[101]:
```
# Distribution of Delinquency Ratio
plt.figure(figsize=(10, 6))
sns.histplot(dataset['Delinquency_Ratio'], bins=30, kde=True)
plt.title('Distribution of Delinquency Ratio')
plt.xlabel('Delinquency Ratio')
plt.ylabel('Frequency')
plt.axvline(dataset['Delinquency_Ratio'].mean(), color='r', linestyle='dashed',␣
↪linewidth=2, label='Mean')
plt.legend()
plt.show()
```



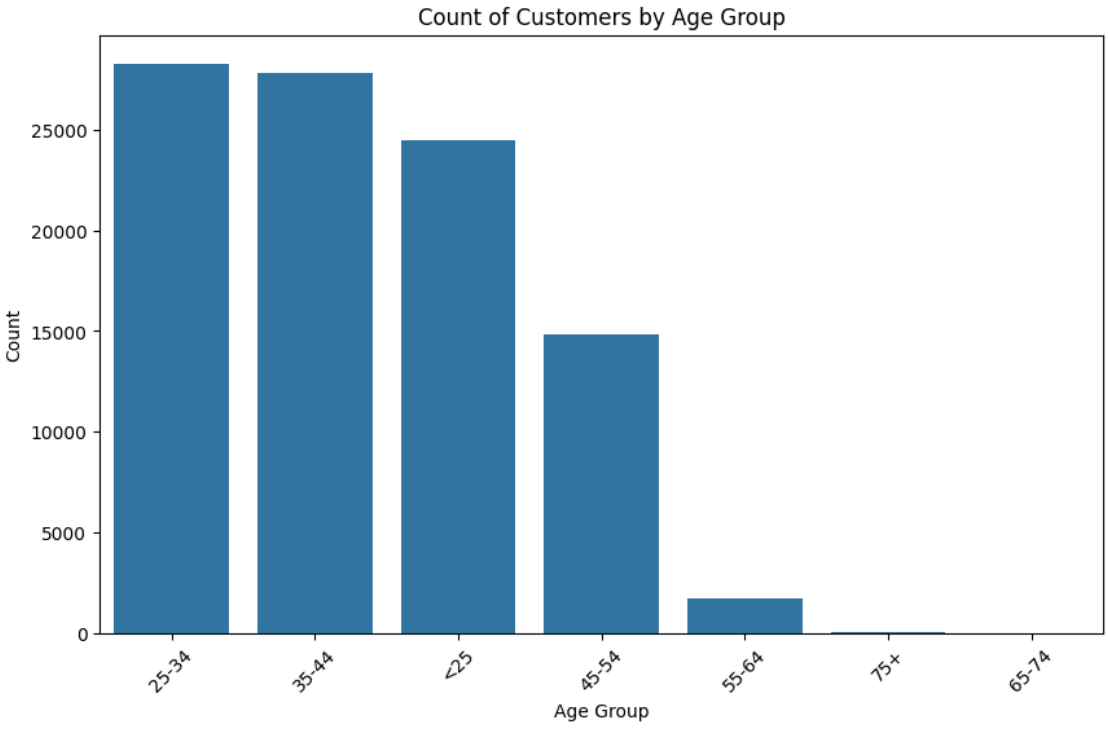
Visualization of Monthly Savings Rate

[102]:
```
# Distribution of Monthly Savings Rate
plt.figure(figsize=(10, 6))
sns.histplot(dataset['Monthly_Savings_Rate'], bins=30, kde=True)
plt.title('Distribution of Monthly Savings Rate')
plt.xlabel('Monthly Savings Rate')
plt.ylabel('Frequency')
plt.axvline(dataset['Monthly_Savings_Rate'].mean(), color='r',␣
 ↪linestyle='dashed', linewidth=2, label='Mean')
plt.legend()
plt.show()
```



Visualization of Age Group Distribution

[103]:
```
# Count plot of Age Groups
plt.figure(figsize=(10, 6))
sns.countplot(data=dataset, x='Age_Group', order=dataset['Age_Group'].
 ↪value_counts().index)
plt.title('Count of Customers by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```


Count of Customers by Age Group
Count
25000
20000
15000
10000
5000
0
25-34
35-44
<25
45-54
55-64
75+
65-74
Age Group