Credit EDA & Credit Score Calculation with Python

Introduction

In today's financial landscape, understanding customer creditworthiness is paramount for both lenders and consumers. This project aims to conduct a thorough exploratory data analysis (EDA) of a comprehensive dataset that encompasses essential customer details and extensive credit-related information. By delving into variables such as annual income, outstanding debt, and payment behavior, the analysis seeks to uncover meaningful patterns and insights that inform credit risk assessments.

The primary objective is to create new, informative features and develop a hypothetical credit score model inspired by established methodologies, such as FICO scores. Through this deep analysis, we aim to identify factors influencing creditworthiness, provide individualized credit scores for each customer, and explore potential risk mitigation strategies. This project not only enhances our understanding of customer credit profiles but also serves as a valuable tool for financial institutions in making informed lending decisions.

Exploratory Data Analysis

Importing the required libraries

In [122…

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [122…

```
df = pd.read_csv('Credit_score.csv')
```

In [122…

```
df.head()
```

Out[122…

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	NaN	3	
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	

In [122…
```
df.shape
```

Out[122…
```
(100000, 27)
```

In [122…
```
df.columns
```

Out[122…

```
Index(['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance'],
      dtype='object')
```

In [122…

```
df.dtypes
```

Out[122…

```
ID                          object
Customer_ID                 object
Month                       object
Name                        object
Age                         object
SSN                         object
Occupation                  object
Annual_Income               object
Monthly_Inhand_Salary      float64
Num_Bank_Accounts            int64
Num_Credit_Card              int64
Interest_Rate                int64
Num_of_Loan                 object
Type_of_Loan                object
Delay_from_due_date          int64
Num_of_Delayed_Payment      object
Changed_Credit_Limit        object
Num_Credit_Inquiries       float64
Credit_Mix                  object
Outstanding_Debt            object
Credit_Utilization_Ratio   float64
Credit_History_Age          object
Payment_of_Min_Amount       object
Total_EMI_per_month        float64
Amount_invested_monthly     object
Payment_Behaviour           object
Monthly_Balance             object
dtype: object
```

In [122…
```
df.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
```

In [123…
```
df.describe()
```

Out[123…

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Delay_from_due_date	Num_Credit_Inquiries	Credit_U
count	84998.000000	100000.000000	100000.00000	100000.000000	100000.000000	98035.000000	
mean	4194.170850	17.091280	22.47443	72.466040	21.068780	27.754251	
std	3183.686167	117.404834	129.05741	466.422621	14.860104	193.177339	
min	303.645417	-1.000000	0.00000	1.000000	-5.000000	0.000000	
25%	1625.568229	3.000000	4.00000	8.000000	10.000000	3.000000	
50%	3093.745000	6.000000	5.00000	13.000000	18.000000	6.000000	
75%	5957.448333	7.000000	7.00000	20.000000	28.000000	9.000000	
max	15204.633330	1798.000000	1499.00000	5797.000000	67.000000	2597.000000	

In [123…

```
df.nunique()
```

Out[123…

```
ID                        100000
Customer_ID                12500
Month                          8
Name                       10139
Age                         1788
SSN                        12501
Occupation                    16
Annual_Income              18940
Monthly_Inhand_Salary      13235
Num_Bank_Accounts            943
Num_Credit_Card             1179
Interest_Rate               1750
Num_of_Loan                  434
Type_of_Loan                6260
Delay_from_due_date           73
Num_of_Delayed_Payment       749
Changed_Credit_Limit        3635
Num_Credit_Inquiries        1223
Credit_Mix                     4
Outstanding_Debt           13178
Credit_Utilization_Ratio   99998
Credit_History_Age           404
Payment_of_Min_Amount          3
Total_EMI_per_month        14950
Amount_invested_monthly    91049
Payment_Behaviour              7
Monthly_Balance            98790
dtype: int64
```

In [123…

```
df.sample(1).T
```

Out[123...

	3938
ID	0x2d14
Customer_ID	CUS_0x122c
Month	March
Name	Papadimasf
Age	47
SSN	883-73-9594
Occupation	Entrepreneur
Annual_Income	36346.13_
Monthly_Inhand_Salary	3188.844167
Num_Bank_Accounts	8
Num_Credit_Card	6
Interest_Rate	11
Num_of_Loan	3
Type_of_Loan	Credit-Builder Loan, Home Equity Loan, and Aut...
Delay_from_due_date	26
Num_of_Delayed_Payment	NaN
Changed_Credit_Limit	10.54
Num_Credit_Inquiries	4.0
Credit_Mix	_
Outstanding_Debt	1252.46
Credit_Utilization_Ratio	25.149442
Credit_History_Age	21 Years and 9 Months

	3938
Payment_of_Min_Amount	No
Total_EMI_per_month	89.061554
Amount_invested_monthly	205.3964341
Payment_Behaviour	Low_spent_Medium_value_payments
Monthly_Balance	304.4264285

Dropping unnecessary columns as ID and SSN are not involved in the credit score calculation

In [123…
```
df.drop(columns=['ID','SSN'],inplace=True)
```

Handling Mismatched Data Types

Let's convert the data types for better analysis and performance

In [123…
```
categorical_columns = ['Month','Credit_Mix','Payment_of_Min_Amount','Payment_Behaviour','Occupation','Type_of_Loan',]

for i in categorical_columns:
    df[i] = df[i].astype('category')
```

In [123…
```
floating_columns = ['Annual_Income','Monthly_Inhand_Salary','Num_of_Loan','Num_of_Delayed_Payment','Interest_Rate',
                    'Changed_Credit_Limit','Outstanding_Debt','Amount_invested_monthly',
                    'Monthly_Balance']

# Since there are few characters in these floating columns which is restricting us to change its data type.
# So let's replace it and change its data type

for i in floating_columns:
        df[i] = df[i].astype('str').str.replace('_','').replace('',pd.NA).dropna().astype('float')
```

In [123…
```
df['Age'] = df['Age'].str.replace(r'[_\s-]', '', regex=True)
df['Age']=df['Age'].astype('int')
```

In [123…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 25 columns):
 #   Column                    Non-Null Count   Dtype   
---  ------                    --------------   -----   
 0   Customer_ID               100000 non-null  object  
 1   Month                     100000 non-null  category
 2   Name                      90015 non-null   object  
 3   Age                       100000 non-null  int32   
 4   Occupation                100000 non-null  category
 5   Annual_Income             100000 non-null  float64 
 6   Monthly_Inhand_Salary     84998 non-null   float64 
 7   Num_Bank_Accounts         100000 non-null  int64   
 8   Num_Credit_Card           100000 non-null  int64   
 9   Interest_Rate             100000 non-null  float64 
 10  Num_of_Loan               100000 non-null  float64 
 11  Type_of_Loan              88592 non-null   category
 12  Delay_from_due_date       100000 non-null  int64   
 13  Num_of_Delayed_Payment    92998 non-null   float64 
 14  Changed_Credit_Limit      97909 non-null   float64 
 15  Num_Credit_Inquiries      98035 non-null   float64 
 16  Credit_Mix                100000 non-null  category
 17  Outstanding_Debt          100000 non-null  float64 
 18  Credit_Utilization_Ratio  100000 non-null  float64 
 19  Credit_History_Age        90970 non-null   object  
 20  Payment_of_Min_Amount     100000 non-null  category
 21  Total_EMI_per_month       100000 non-null  float64 
 22  Amount_invested_monthly   95521 non-null   float64 
 23  Payment_Behaviour         100000 non-null  category
 24  Monthly_Balance           98800 non-null   float64 
dtypes: category(6), float64(12), int32(1), int64(3), object(3)
memory usage: 15.0+ MB
```

Handling missing values and inconsistent data

In [123…
```
df.isnull().sum()
```

Out[123…
```
Customer_ID                   0
Month                         0
Name                       9985
Age                           0
Occupation                    0
Annual_Income                 0
Monthly_Inhand_Salary     15002
Num_Bank_Accounts             0
Num_Credit_Card               0
Interest_Rate                 0
Num_of_Loan                   0
Type_of_Loan              11408
Delay_from_due_date           0
Num_of_Delayed_Payment     7002
Changed_Credit_Limit       2091
Num_Credit_Inquiries       1965
Credit_Mix                    0
Outstanding_Debt              0
Credit_Utilization_Ratio      0
Credit_History_Age         9030
Payment_of_Min_Amount         0
Total_EMI_per_month           0
Amount_invested_monthly    4479
Payment_Behaviour             0
Monthly_Balance            1200
dtype: int64
```

In [123…
```
df[df['Name'].isna()].head()
```

Out[123...

	Customer_ID	Month	Name	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Inte
7	CUS_0xd40	August	NaN	23	Scientist	19114.12	1824.843333	3	4	
17	CUS_0x2dbc	February	NaN	34	Engineer	143162.64	12187.220000	1	5	
22	CUS_0x2dbc	July	NaN	34	Engineer	143162.64	12187.220000	1	5	
64	CUS_0x4157	January	NaN	23	Doctor	114838.41	9843.867500	2	5	
80	CUS_0xa66b	January	NaN	40	Teacher	33751.27	2948.605833	5	5	

In [124...

```
missing_names = df[df['Name'].isna()]['Customer_ID'].unique()
missing_names
```

Out[124...

```
array(['CUS_0xd40', 'CUS_0x2dbc', 'CUS_0x4157', ..., 'CUS_0x372c',
       'CUS_0xf16', 'CUS_0x8600'], dtype=object)
```

In [124...

```
match_names = df[df['Customer_ID'].isin(missing_names)]['Name'].unique()
match_names
```

Out[124…
```
array(['Aaron Maashoh', nan, 'Langep', ..., 'Xolai', 'Sabina Zawadzkig',
       'Sarah McBridec'], dtype=object)
```

We can clearly identify records for customer names by matching them with their customer IDs, allowing us to replace the missing ones. This pattern might apply to other missing records as well. Therefore, we can align all of them based on the most frequent value for each customer.

In [124…
```
missing_values_col = ['Name','Monthly_Inhand_Salary']

for i in missing_values_col:
    df[i]= df.groupby('Customer_ID')[i].transform(lambda x: x.mode()[0])
```

Handling Inconsistent data

In [124…
```
inconsistent_data_col = ['Age','Annual_Income','Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate','Num_of_Loan','Num_Credit_Inquiries','Num_of_Delayed_Payment','Outstanding_Debt','To
```

In [124…
```
df[inconsistent_data_col].value_counts()
```

Out[124…

```
Age   Annual_Income  Num_Bank_Accounts  Num_Credit_Card  Interest_Rate  Num_of_Loan  Num_Credit_Inquiries  Num_of_Delayed_Pay
ment  Outstanding_Debt  Total_EMI_per_month
14    3.921588e+04   5                  6                13.0           6.0          5.0                   18.0
1903.55           159.122144             8
22    1.928073e+04   5                  9                32.0           2.0          10.0                  11.0
2412.06           17.677036              8
39    1.073736e+04   8                  4                14.0           4.0          8.0                   16.0
1126.18           19.502581              8
22    1.173429e+05   6                  6                17.0           1.0          6.0                   19.0
764.82            74.134685              8
23    2.222215e+04   7                  6                20.0           4.0          8.0                   9.0
81.68             44.470649              8

..
29    2.229960e+07   9                  794              22.0           9.0          8.0                   15.0
4064.57           57.931689              1
      2.264433e+07   10                 5                34.0           9.0          12.0                  18.0
4687.35           59.248652              1
30    7.084365e+03   6                  6                27.0           7.0          4.0                   13.0
1293.67           26.758124              1
                                                                                                          14.0
1293.67           26.758124              1
8698  1.742369e+04   6                  5                15.0           0.0          7.0                   11.0
1158.96           0.000000               1
Name: count, Length: 62225, dtype: int64
```

We can see the columns Age, Num of bank accounts, Num of credit cards, Interest rate has many inconsistent data. We can treat them by detecting the outliers and rectifying it

Handling Outliers

In [124…

```
df.describe()
```

Out[124...

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Dela
count	100000.000000	1.000000e+05	100000.000000	100000.000000	100000.00000	100000.000000	100000.000000	
mean	119.509700	1.764157e+05	4197.328114	17.091280	22.47443	72.466040	3.009960	
std	684.757313	1.429618e+06	3186.918569	117.404834	129.05741	466.422621	62.647879	
min	14.000000	7.005930e+03	303.645417	-1.000000	0.00000	1.000000	-100.000000	
25%	25.000000	1.945750e+04	1626.719792	3.000000	4.00000	8.000000	1.000000	
50%	34.000000	3.757861e+04	3092.270000	6.000000	5.00000	13.000000	3.000000	
75%	42.000000	7.279092e+04	5958.695625	7.000000	7.00000	20.000000	5.000000	
max	8698.000000	2.419806e+07	15204.633330	1798.000000	1499.00000	5797.000000	1496.000000	

In [124...

```
df.columns
```

Out[124...

```
Index(['Customer_ID', 'Month', 'Name', 'Age', 'Occupation', 'Annual_Income',
       'Monthly_Inhand_Salary', 'Num_Bank_Accounts', 'Num_Credit_Card',
       'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan', 'Delay_from_due_date',
       'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance'],
      dtype='object')
```

In [124...

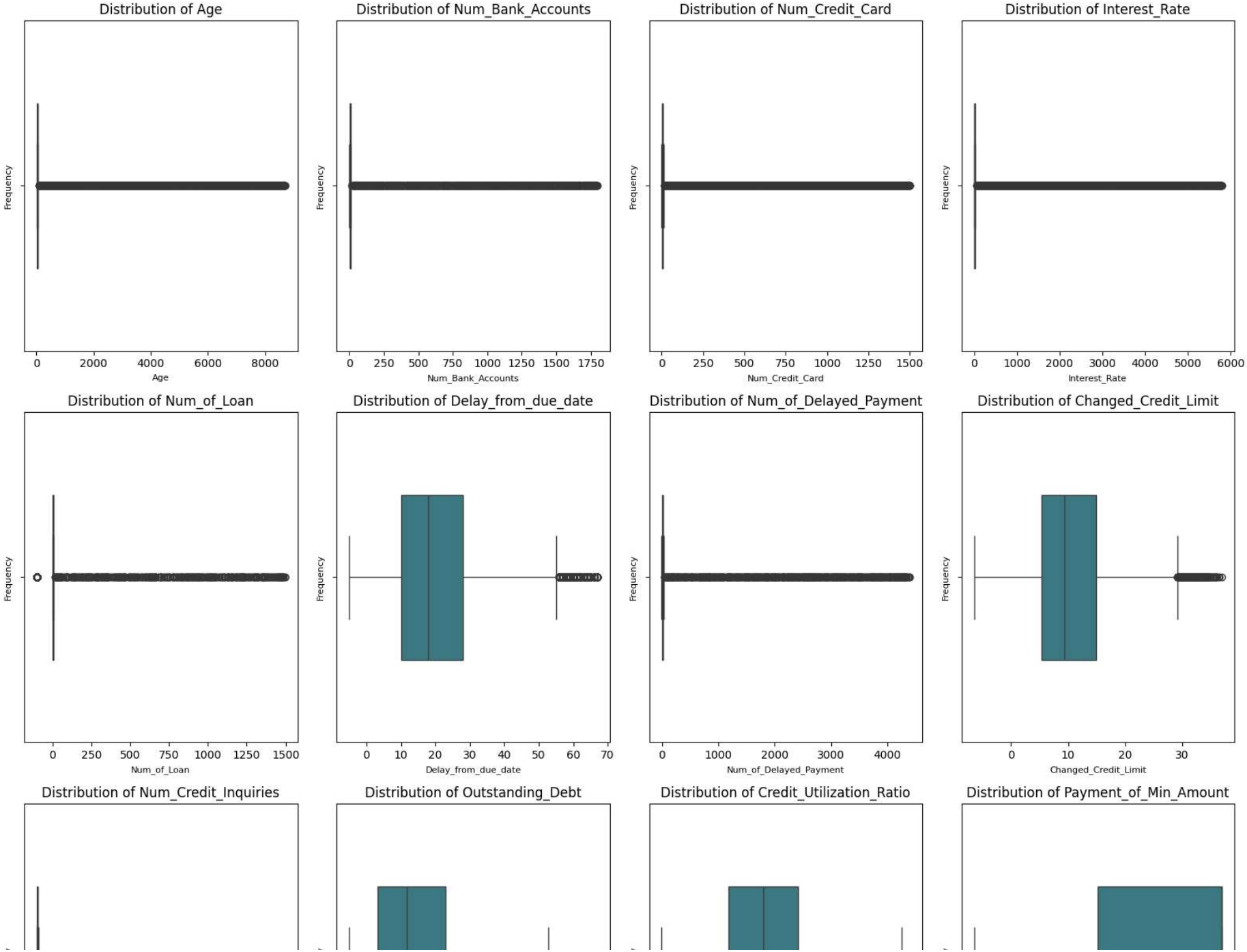
```
float_columns = ['Age','Num_Bank_Accounts','Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Outstanding_Debt','Credit_Utilization_Ratio', 'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Monthly_Balance']
```

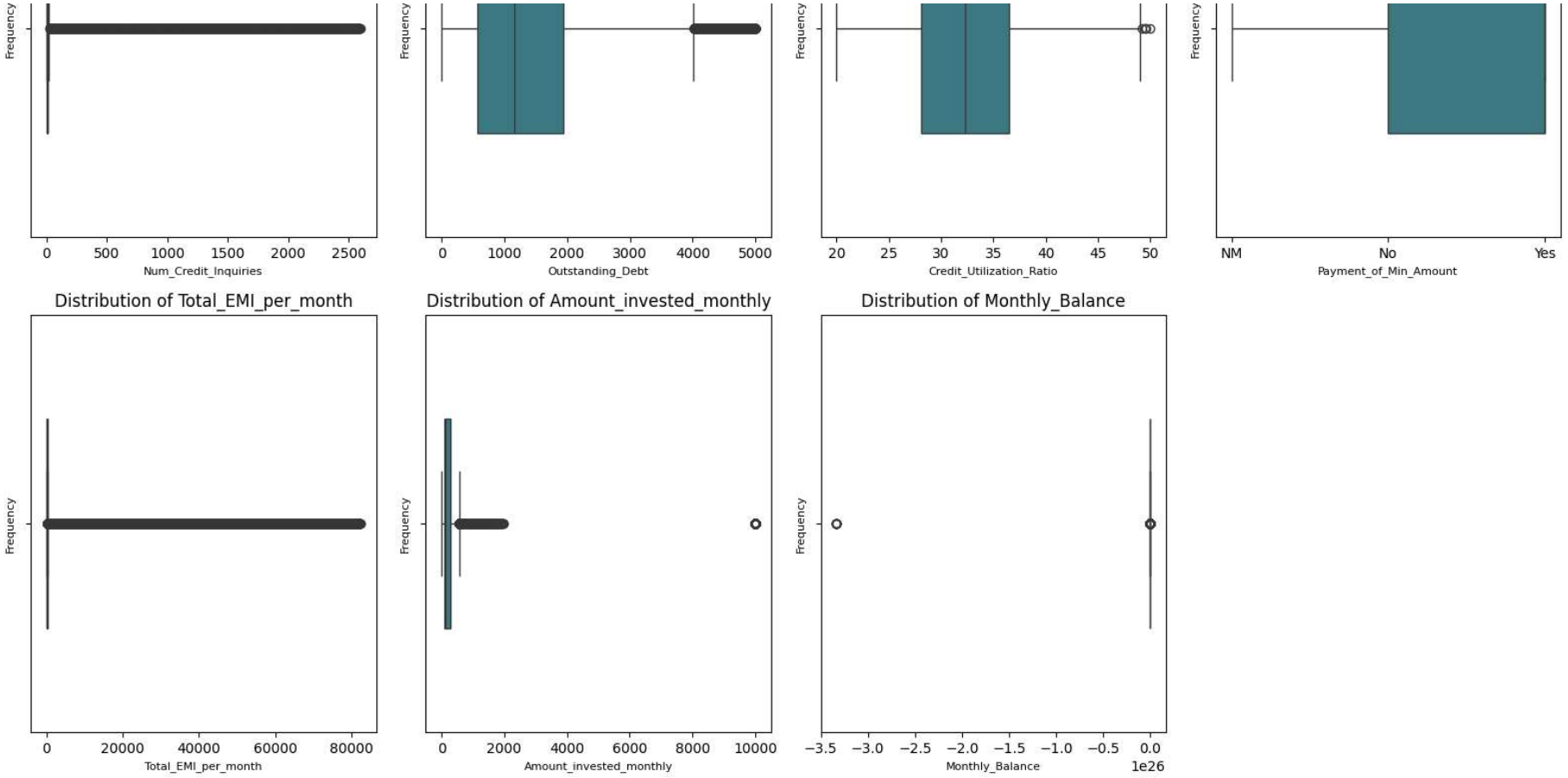
In [124...

```
plt.figure(figsize = (16,20))
for i in range(1,16):
    plt.subplot(4,4,i)
    sns.boxplot(data=df,x=float_columns[i-1],palette='crest',width=0.5)
```

```
    plt.title(f'Distribution of {float_columns[i-1]}')
    plt.ylabel('Frequency',fontsize=8)
    plt.xlabel(f'{float_columns[i-1]}',fontsize = 8)

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


Distribution of Age
Frequency
0 2000 4000 6000 8000
Age
Distribution of Num_Bank_Accounts
Frequency
0 250 500 750 1000 1250 1500 1750
Num_Bank_Accounts
Distribution of Num_Credit_Card
Frequency
0 250 500 750 1000 1250 1500
Num_Credit_Card
Distribution of Interest_Rate
Frequency
0 1000 2000 3000 4000 5000 6000
Interest_Rate
Distribution of Num_of_Loan
Frequency
0 250 500 750 1000 1250 1500
Num_of_Loan
Distribution of Delay_from_due_date
Frequency
0 10 20 30 40 50 60 70
Delay_from_due_date
Distribution of Num_of_Delayed_Payment
Frequency
0 1000 2000 3000 4000
Num_of_Delayed_Payment
Distribution of Changed_Credit_Limit
Frequency
0 10 20 30
Changed_Credit_Limit
Distribution of Num_Credit_Inquiries
Distribution of Outstanding_Debt
Distribution of Credit_Utilization_Ratio
Distribution of Payment_of_Min_Amount



We can clearly visualize that there are numerous outliers across different columns

In [124…
```
for i in inconsistent_data_col:
    df[i]= df.groupby('Customer_ID')[i].transform(lambda x: x.mode()[0])
```

In [125…
```
df['Num_Bank_Accounts'] = df['Num_Bank_Accounts'].apply(lambda x: 1 if x==-1 else x)
```

In [125…
```
df['Delay_from_due_date'] = df['Delay_from_due_date'].apply(lambda x: np.nan if x<0 else x)
df['Delay_from_due_date'] = df['Delay_from_due_date'].ffill()
```

In [125…
```
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].apply(lambda x: np.nan if x<0 else x)
```

```
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].ffill()
```

In [125…
```
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].apply(lambda x: np.nan if x<0 else x)
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].ffill()
```

In [125…
```
inconsistent_data_col_mean = ['Amount_invested_monthly','Monthly_Balance']

for i in inconsistent_data_col_mean:
    df[i]= df.groupby('Customer_ID')[i].transform(lambda x: x.mean())
```

In [125…
```
df[inconsistent_data_col_mean].describe()
```

Out[125…

	Amount_invested_monthly	Monthly_Balance
count	100000.000000	1.000000e+05
mean	637.569686	-3.000000e+22
std	751.516474	1.117637e+24
min	15.292436	-4.166667e+25
25%	117.185215	2.844555e+02
50%	233.589940	3.383843e+02
75%	1336.541021	4.652797e+02
max	5748.505316	1.349265e+03

In [125…
```
df[inconsistent_data_col].describe()
```

Out[125…

	Age	Annual_Income	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Num_Credit_Inquiries	Num_of
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.00000	100000.000000	100000.000000	
mean	33.274560	50505.123449	5.368160	5.532720	14.53208	3.532880	5.677760	
std	10.764438	38299.422093	2.591996	2.067504	8.74133	2.446356	3.827248	
min	14.000000	7005.930000	0.000000	0.000000	1.00000	0.000000	0.000000	
25%	24.000000	19342.972500	3.000000	4.000000	7.00000	2.000000	3.000000	
50%	33.000000	36999.705000	5.000000	5.000000	13.00000	3.000000	5.000000	
75%	42.000000	71683.470000	7.000000	7.000000	20.00000	5.000000	8.000000	
max	56.000000	179987.280000	10.000000	11.000000	34.00000	9.000000	17.000000	

In [125…

```
df['Credit_Mix'].value_counts()
```

Out[125…

```
Credit_Mix
Standard    36479
Good        24337
_           20195
Bad         18989
Name: count, dtype: int64
```

In [125…

```
df['Credit_Mix'] = df['Credit_Mix'].apply(lambda x: np.nan if x=='_' else x)
df['Credit_Mix']= df.groupby('Customer_ID')['Credit_Mix'].transform(lambda x: x.mode()[0])
```

In [125…

```
df['Occupation'].value_counts()
```

Out[125…

```
Occupation
_______          7062
Lawyer           6575
Architect        6355
Engineer         6350
Scientist        6299
Mechanic         6291
Accountant       6271
Developer        6235
Media_Manager    6232
Teacher          6215
Entrepreneur     6174
Doctor           6087
Journalist       6085
Manager          5973
Musician         5911
Writer           5885
Name: count, dtype: int64
```

In [126…

```
df['Occupation'] = df['Occupation'].apply(lambda x: np.nan if x=='_______' else x)
df['Occupation']= df.groupby('Customer_ID')['Occupation'].transform(lambda x: x.mode()[0])
```

In [126…

```
df['Payment_of_Min_Amount'].value_counts()
```

Out[126…

```
Payment_of_Min_Amount
Yes    52326
No     35667
NM     12007
Name: count, dtype: int64
```

In [126…

```
df['Payment_of_Min_Amount'] = df['Payment_of_Min_Amount'].apply(lambda x: 'No' if x=='NM' else x)
```

In [126…

```
df[['Payment_Behaviour']].value_counts()
```

Out[126…
```
Payment_Behaviour
Low_spent_Small_value_payments      25513
High_spent_Medium_value_payments    17540
Low_spent_Medium_value_payments     13861
High_spent_Large_value_payments     13721
High_spent_Small_value_payments     11340
Low_spent_Large_value_payments      10425
!@9#%8                               7600
Name: count, dtype: int64
```

In [126…
```
df['Payment_Behaviour'] = df['Payment_Behaviour'].apply(lambda x: np.nan if x=='!@9#%8' else x)
df['Payment_Behaviour'] = df.groupby('Customer_ID')['Payment_Behaviour'].transform(lambda x: x.mode()[0])
```

In [126…
```
df[categorical_columns].describe()
```

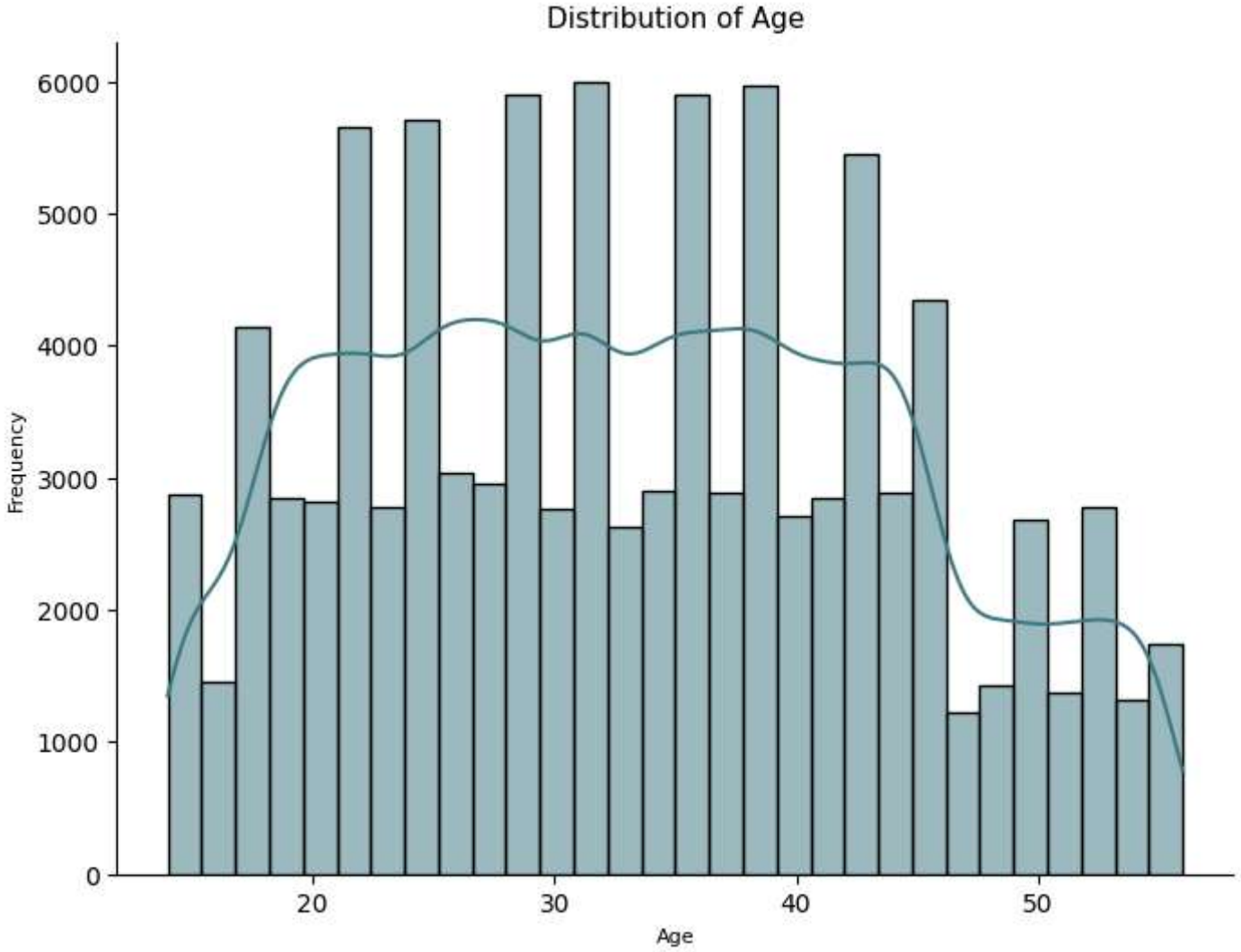
Out[126…

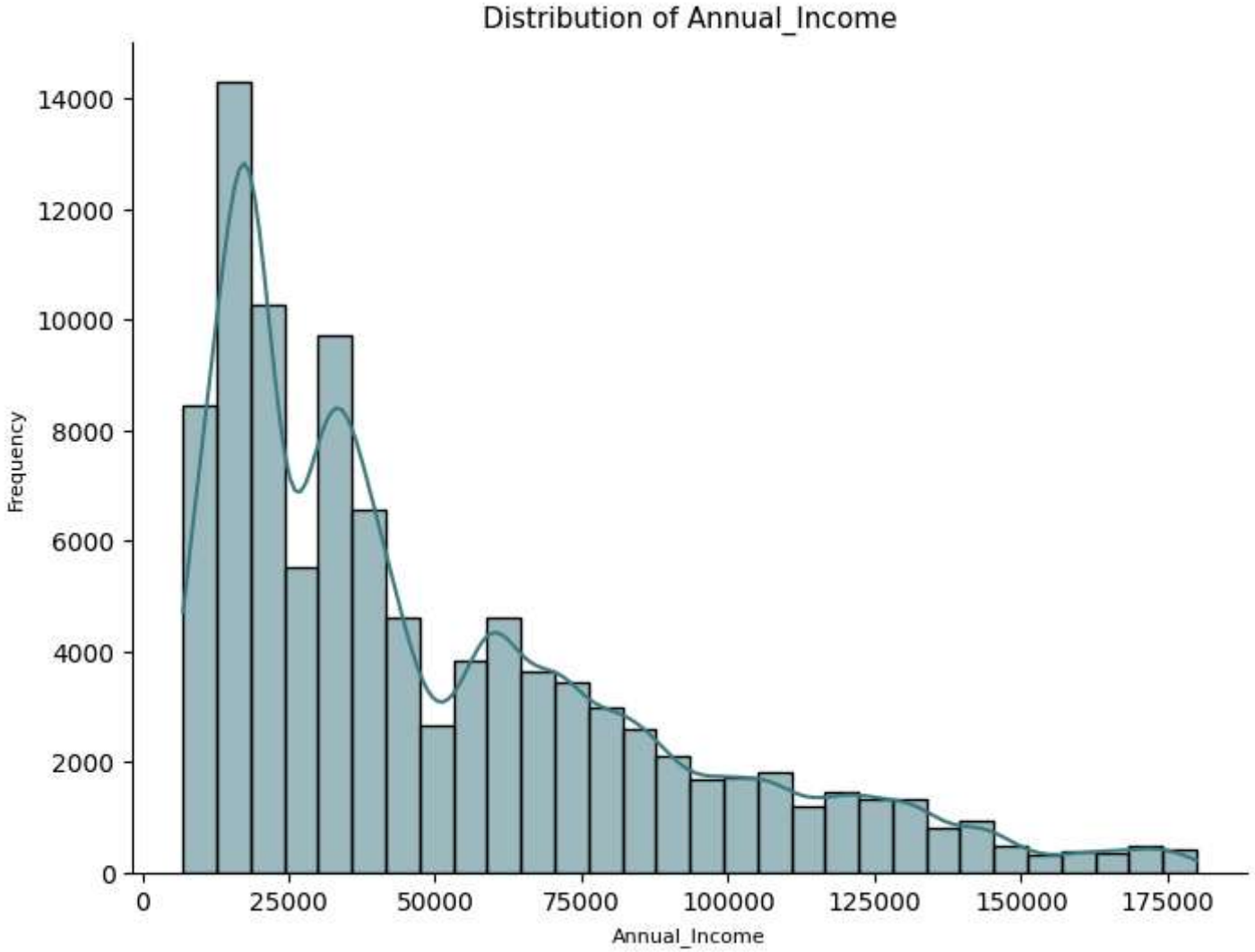
	Month	Credit_Mix	Payment_of_Min_Amount	Payment_Behaviour	Occupation	Type_of_Loan
count	100000	100000	100000	100000	100000	88592
unique	8	3	2	6	15	6260
top	April	Standard	Yes	Low_spent_Small_value_payments	Lawyer	Not Specified
freq	12500	45848	52326	30880	7096	1408

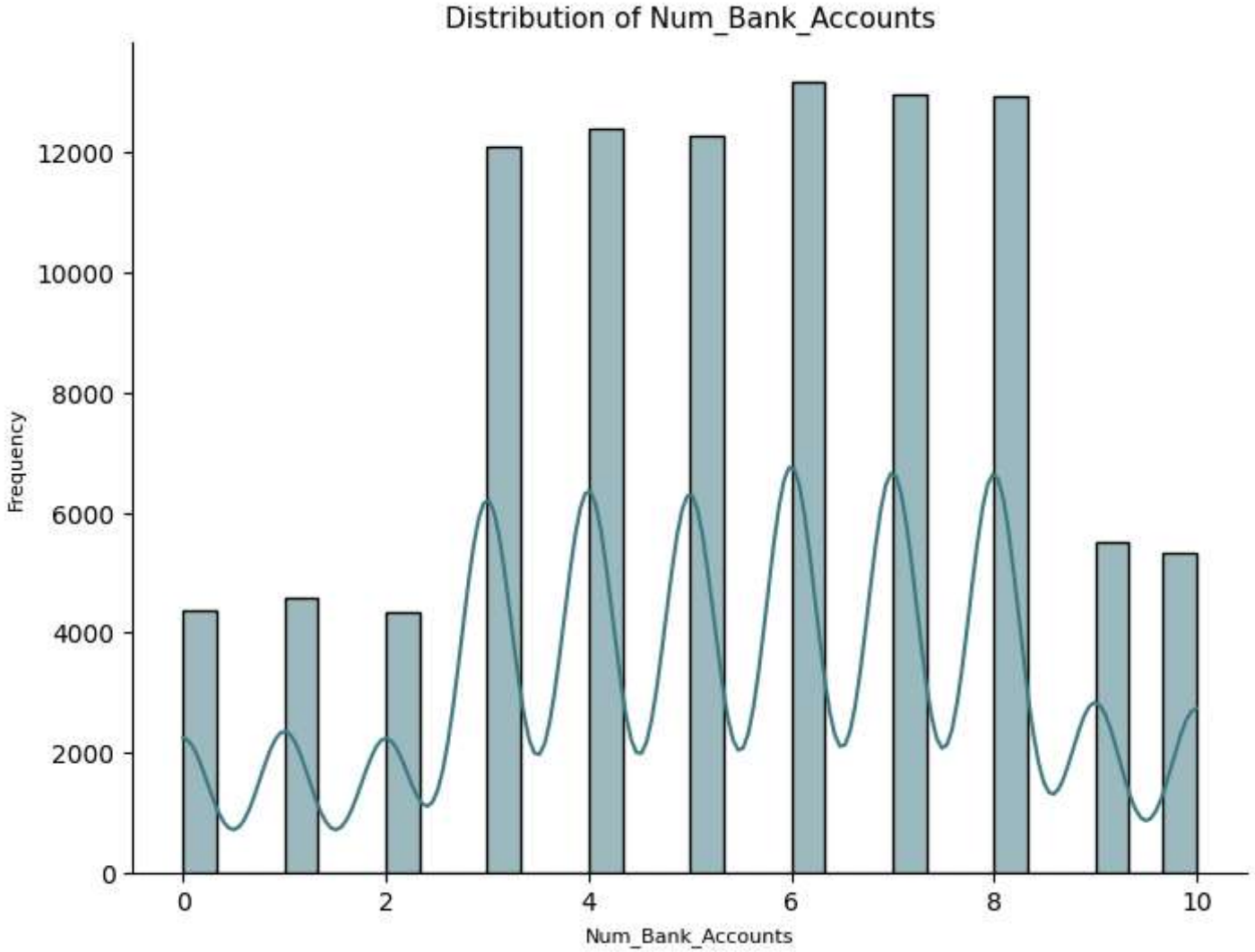
Let's try to visualize various distribution across the dataframe

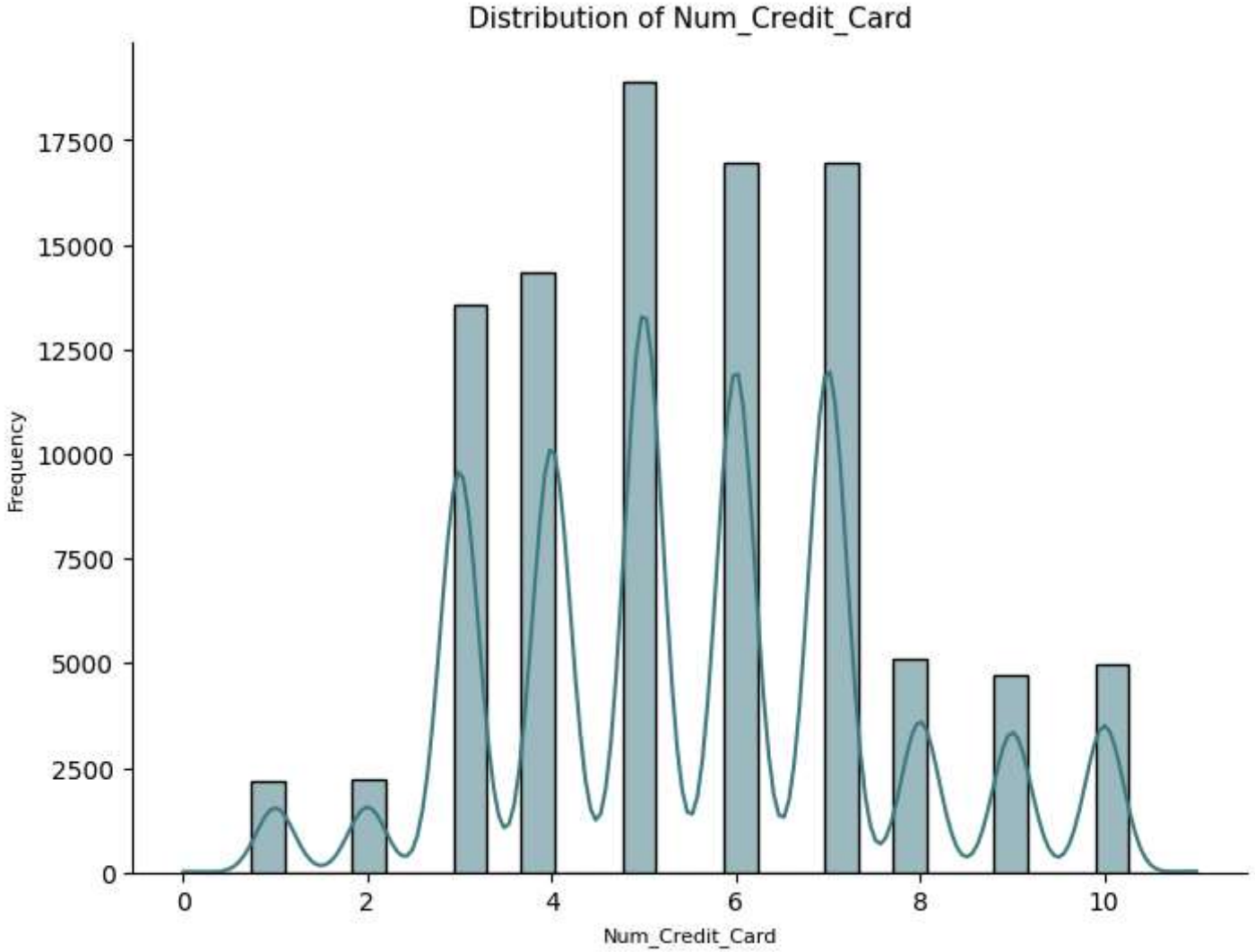
In [126…
```
def plot_distribution(df, column_name):
    plt.figure(figsize=(8, 6))
    sns.histplot(data=df, x=column_name, bins=30, kde=True, color='#3E7C82', edgecolor='black')
    plt.title(f'Distribution of {column_name}', fontproperties = {'size':11})
    plt.xlabel(column_name, fontsize=8)
    plt.ylabel('Frequency', fontsize=8)
    sns.despine()
    plt.show()
```

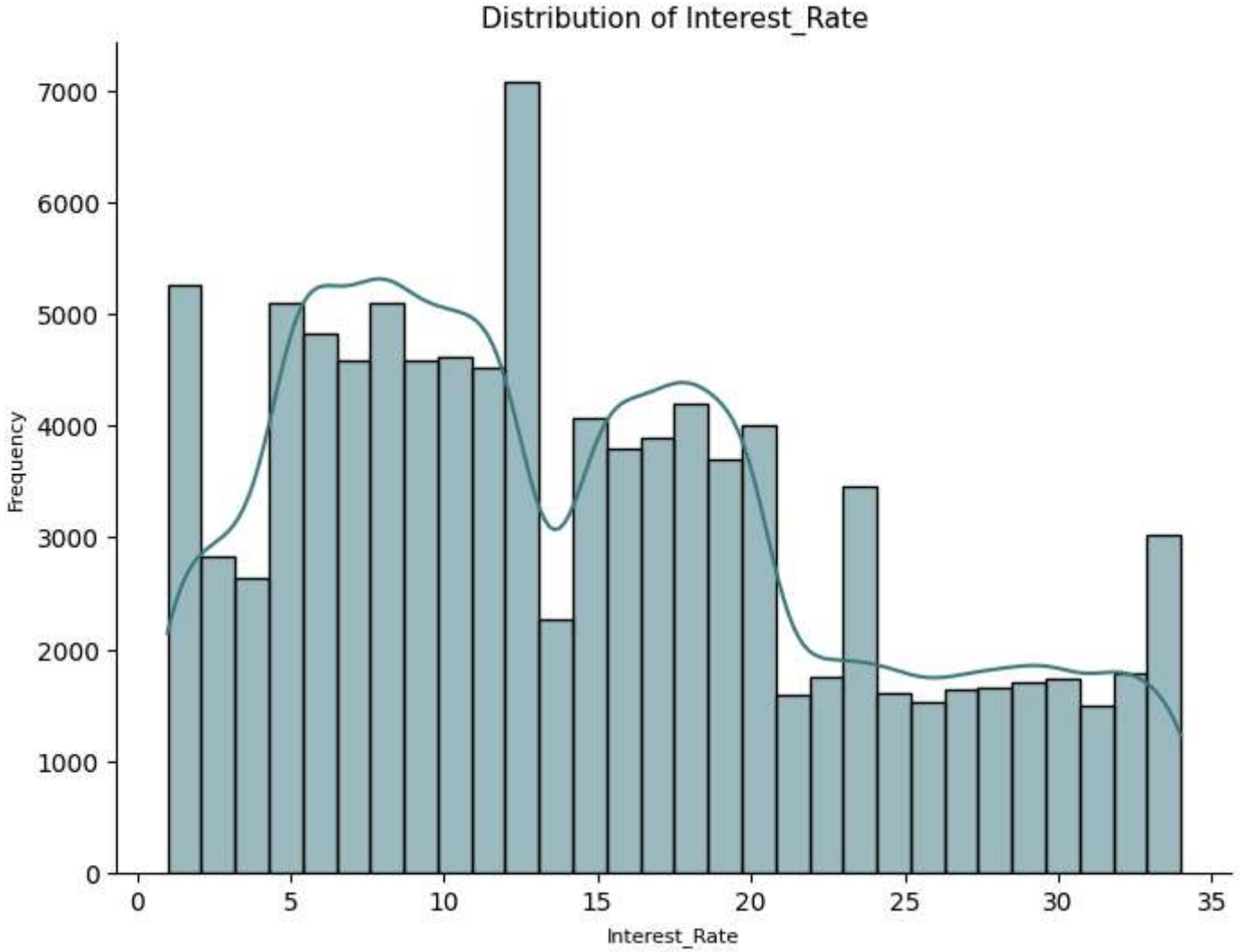
In [126…
```
for x in range(0,len(inconsistent_data_col)):
    plot_distribution(df,inconsistent_data_col[x])
```

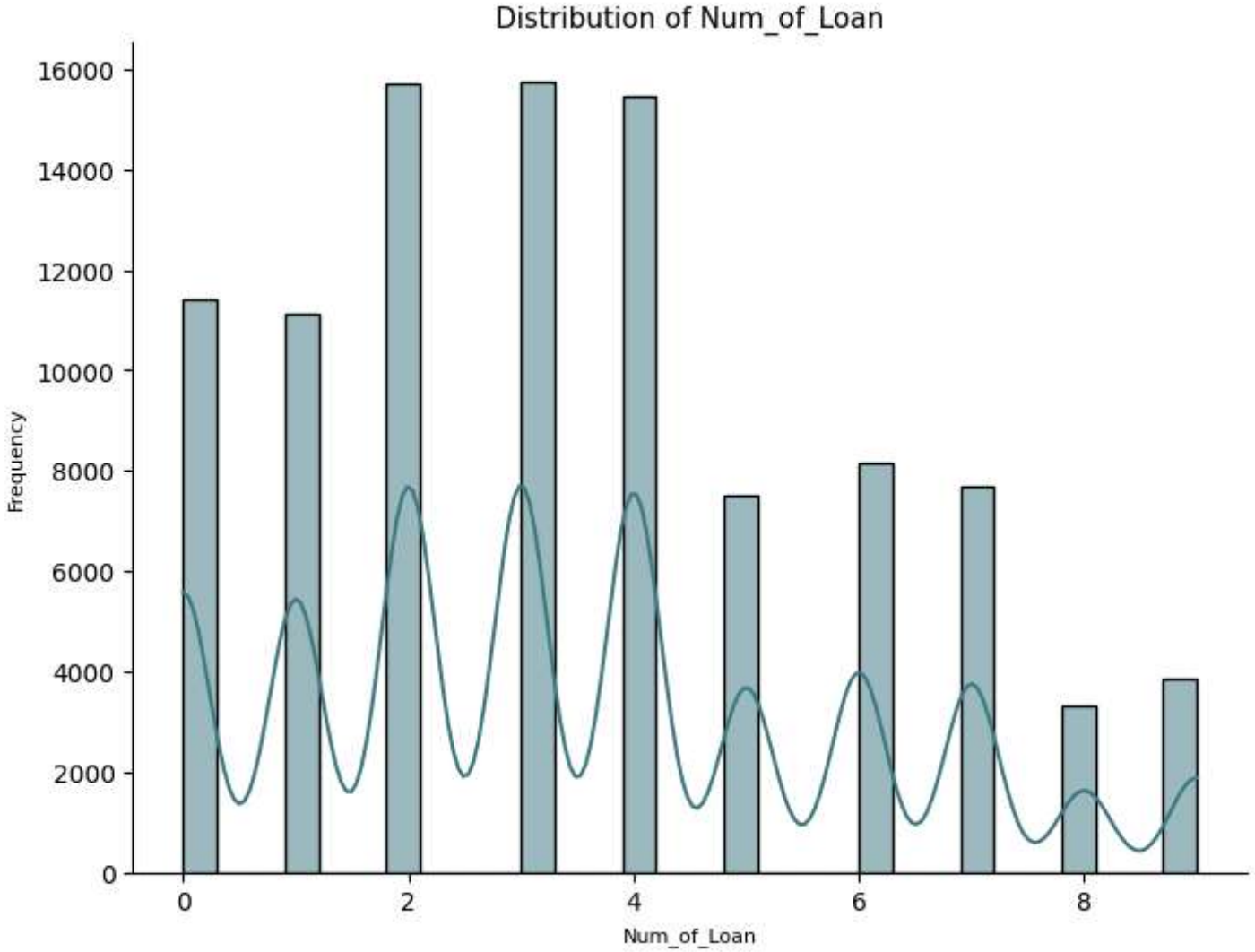

Distribution of Age
Frequency
Age
0
1000
2000
3000
4000
5000
6000
20
30
40
50

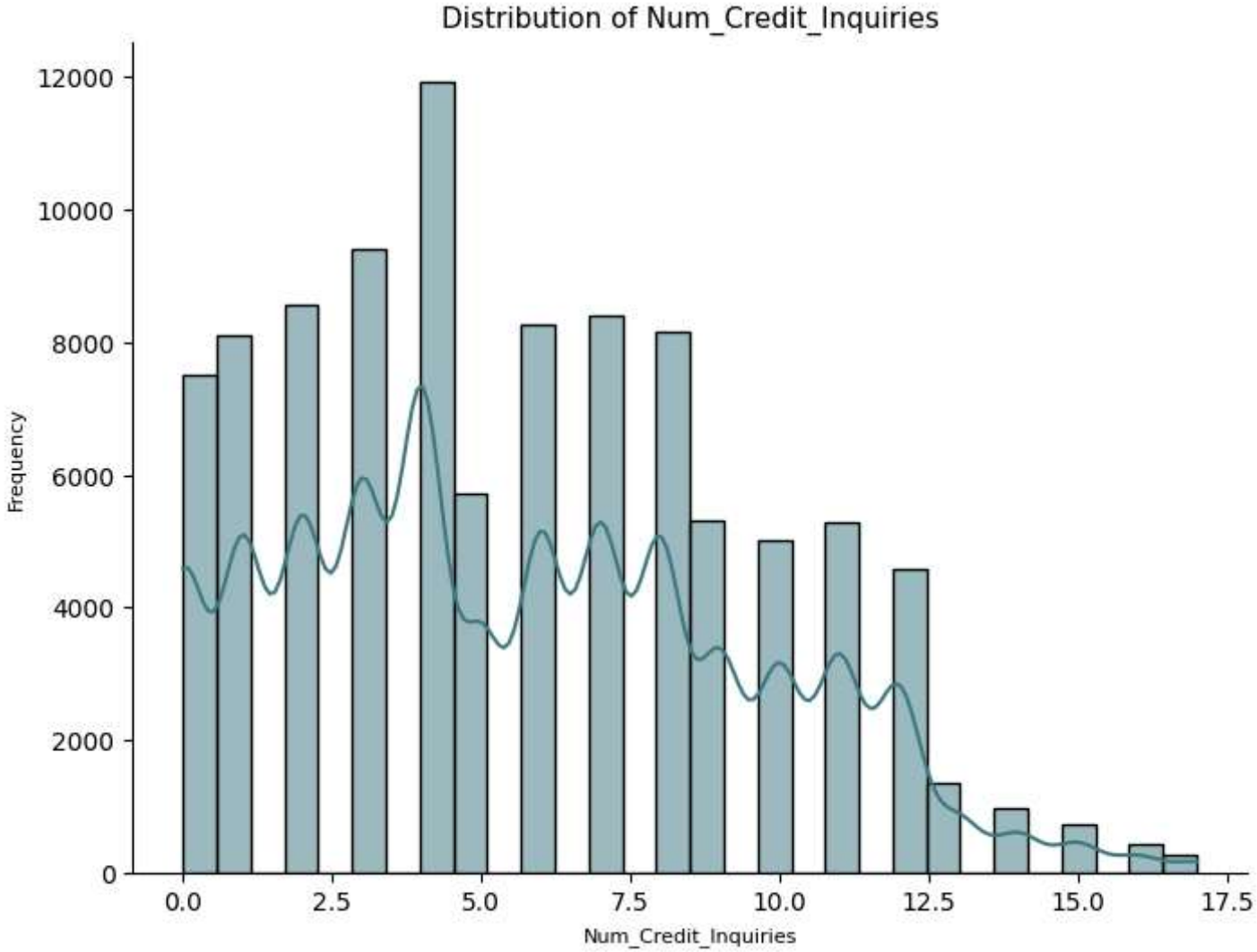

Distribution of Annual_Income
Frequency
14000
12000
10000
8000
6000
4000
2000
0
0
25000
50000
75000
100000
125000
150000
175000
Annual_Income

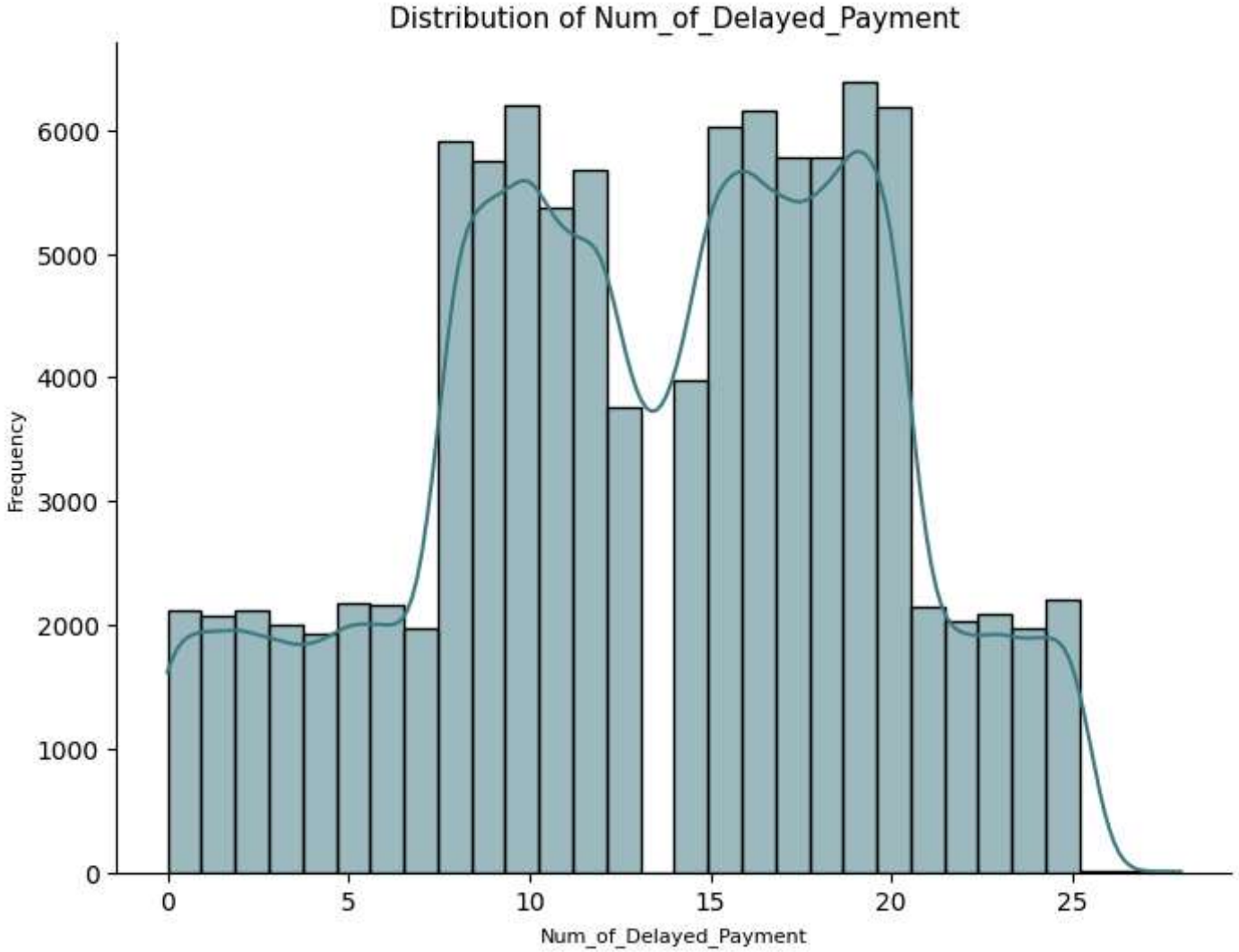

Distribution of Num_Bank_Accounts
Frequency
0
2000
4000
6000
8000
10000
12000
0
2
4
6
8
10
Num_Bank_Accounts

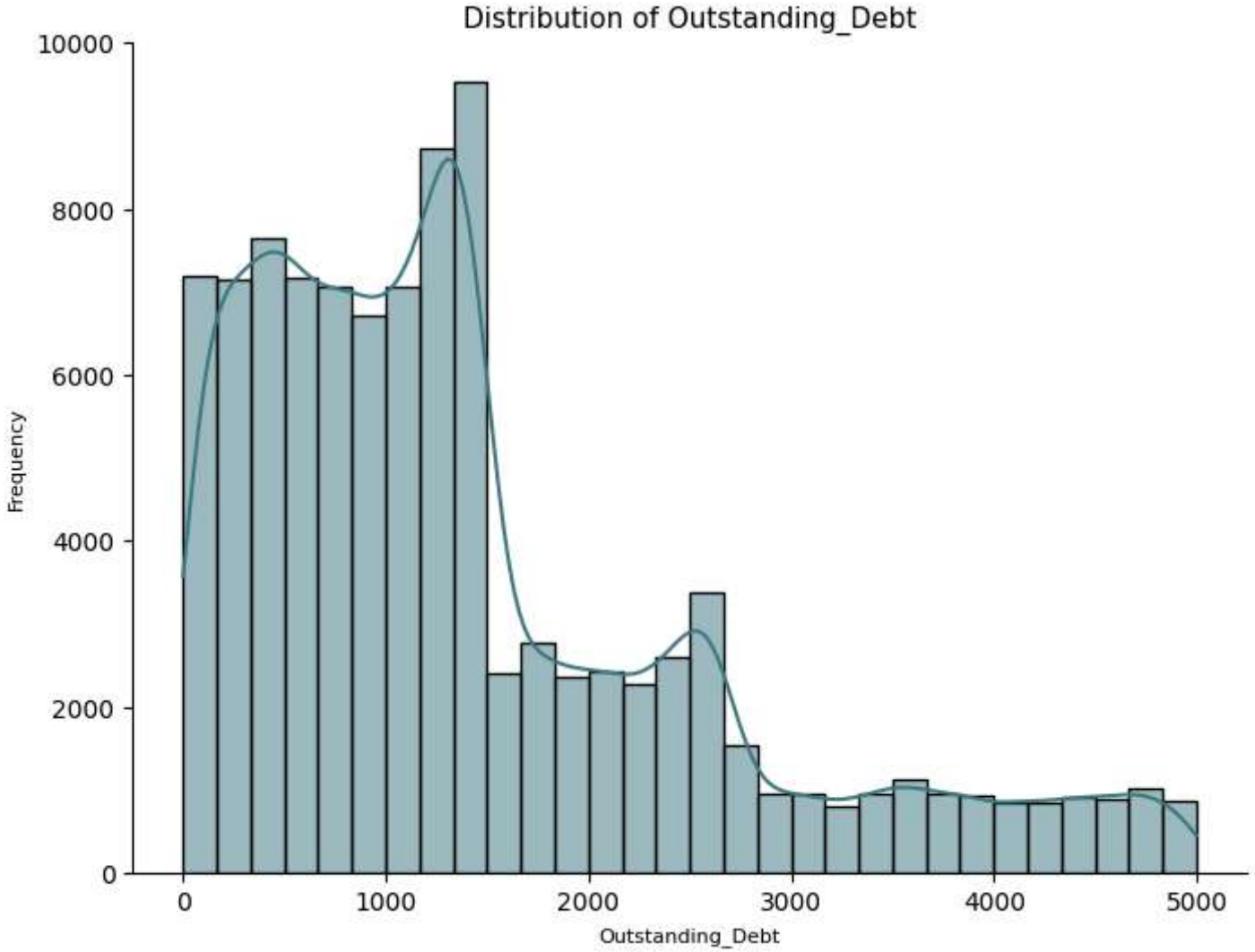

Distribution of Num_Credit_Card
Frequency
17500
15000
12500
10000
7500
5000
2500
0
0
2
4
6
8
10
Num_Credit_Card

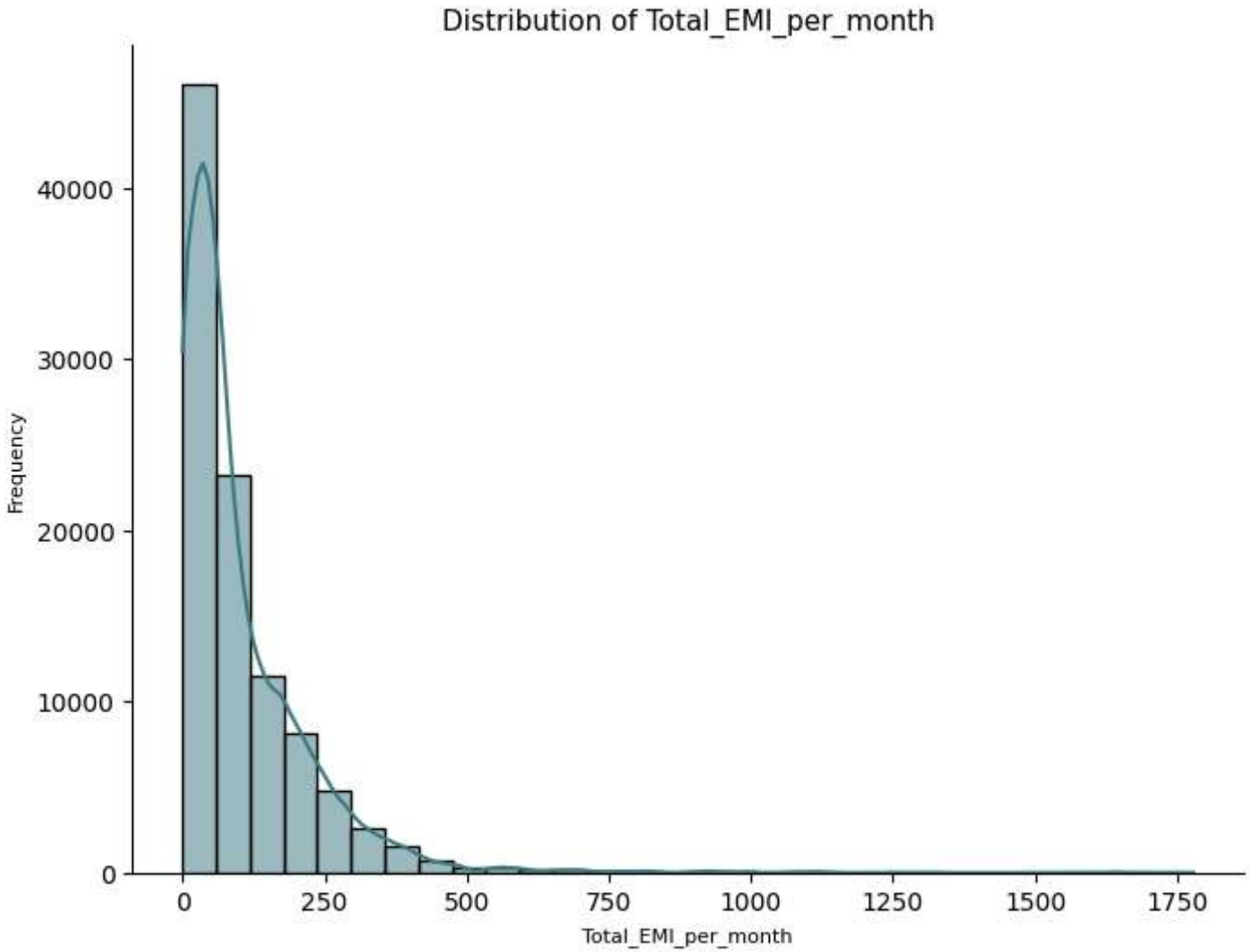

Distribution of Interest_Rate
Frequency
Interest_Rate
0
1000
2000
3000
4000
5000
6000
7000
0
5
10
15
20
25
30
35


Distribution of Num_of_Loan
Frequency
16000
14000
12000
10000
8000
6000
4000
2000
0
0
2
4
6
8
Num_of_Loan


Distribution of Num_Credit_Inquiries
Frequency
0
2000
4000
6000
8000
10000
12000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Num_Credit_Inquiries


Distribution of Num_of_Delayed_Payment
Frequency
0
1000
2000
3000
4000
5000
6000
0
5
10
15
20
25
Num_of_Delayed_Payment


Distribution of Outstanding_Debt
Frequency
Outstanding_Debt
0
2000
4000
6000
8000
10000
0
1000
2000
3000
4000
5000



Feature Engineering

In [126…

```
df.head()
```

Out[126...

	Customer_ID	Month	Name	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Int
0	CUS_0xd40	January	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
1	CUS_0xd40	February	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
2	CUS_0xd40	March	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
3	CUS_0xd40	April	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
4	CUS_0xd40	May	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	

In [126...

```
credit_mix_mapping = {'Bad': 0, 'Standard': 1, 'Good': 2}
payment_behaviour_mapping = {
    'Low_spent_Small_value_payments' : 0,
    'Low_spent_Medium_value_payments' : 1,
    'Low_spent_Large_value_payments' : 2,
    'High_spent_Small_value_payments' : 3,
    'High_spent_Medium_value_payments' : 4,
```

```
    'High_spent_Large_value_payments' : 5
    }
df['Credit_Mix_Score'] = df['Credit_Mix'].map(credit_mix_mapping)
df['Payment_Behaviour_Score'] = df['Payment_Behaviour'].map(payment_behaviour_mapping)
```

In [127…

```
def adjust_credit_history(x):
    if pd.notnull(x):
        year = int(x.split(' ')[0])
        month = int(x.split(' ')[3])
        return (year*12) + month
    return x


# Group by Customer_ID and apply the forward fill and adjustment function
df['Credit_History_Age'] = df.groupby('Customer_ID', group_keys=False)['Credit_History_Age'].apply(lambda x: x.interpolate().b
df['Credit_History_Age_Months'] = df.Credit_History_Age.apply(lambda x: adjust_credit_history(x))
```

In [127…

```
df['Credit_History_Age_Months'].describe()
```

Out[127…

```
count    100000.00000
mean        221.27786
std          99.67973
min           1.00000
25%         145.00000
50%         220.00000
75%         302.00000
max         404.00000
Name: Credit_History_Age_Months, dtype: float64
```

In [127…

```
def credit_history_score(x):
    if pd.notnull(x):
       if x<12:
        return 0
       elif 12<=x<36:
        return 1
       elif 36<=x<84:
        return 2
       elif 84<=x<180:
        return 3
       elif x>180:
```

```
        return 4
    return x

df['Credit_History_Score'] = df.Credit_History_Age_Months.apply(lambda x: credit_history_score(x))
```

In [127…
```
df['Credit_Utilization_Ratio'].describe()
```

Out[127…
```
count    100000.000000
mean         32.285173
std           5.116875
min          20.000000
25%          28.052567
50%          32.305784
75%          36.496663
max          50.000000
Name: Credit_Utilization_Ratio, dtype: float64
```

In [127…
```
def credit_utlization_score(x):
    if pd.notnull(x):
        if x < 30:
            return 2
        elif 30<=x <50:
            return 1
        else:
            return 0

df['Credit_Utilization_Score'] = df.Credit_Utilization_Ratio.apply(lambda x: credit_utlization_score(x))
```

In [127…
```
df['Outstanding_Debt'].describe()
```

Out[127…
```
count    100000.000000
mean       1426.220376
std        1155.129026
min           0.230000
25%         566.072500
50%        1166.155000
75%        1945.962500
max        4998.070000
Name: Outstanding_Debt, dtype: float64
```

In [127…

```
def outstanding_debt_score(x):
    if pd.notnull(x):
        if x == 0:
            return 5
        elif 1 <= x < 1000:
            return 4
        elif 1000 <= x < 2000:
            return 3
        elif 2000 <= x < 3000:
            return 2
        elif 3000 <= x < 4000:
            return 1
        else:
            return 0

df['Outstanding_Debt_Score'] = df.Outstanding_Debt.apply(lambda x: outstanding_debt_score(x))
```

In [127…

```
df['Num_Credit_Inquiries'].describe()
```

Out[127…

```
count    100000.000000
mean          5.677760
std           3.827248
min           0.000000
25%           3.000000
50%           5.000000
75%           8.000000
max          17.000000
Name: Num_Credit_Inquiries, dtype: float64
```

In [127…

```
def segment_credit_inquiries(x):
    if pd.notnull(x):
        if x == 0:
            return 5
        elif 1 <= x <= 2:
            return 4
        elif 3 <= x <= 4:
            return 3
        elif 5 <= x <= 6:
            return 2
        elif 7 <= x <= 9:
```

```
            return 1
        else:
            return 0

df['Credit_Inquiries_Score'] = df.Num_Credit_Inquiries.apply(lambda x: segment_credit_inquiries(x))
```

In [127…

```
def min_payment_score(x):
    if pd.notnull(x):
        if x == 'Yes':
            return 1
        else:
            return 0

df['Payment_of_Min_Amount_Score'] = df.Payment_of_Min_Amount.apply(lambda x: min_payment_score(x))
```

In [128…

```
df['Num_of_Delayed_Payment'].describe()
```

Out[128…

```
count    100000.000000
mean         13.279520
std           6.186425
min           0.000000
25%           9.000000
50%          14.000000
75%          18.000000
max          28.000000
Name: Num_of_Delayed_Payment, dtype: float64
```

In [128…

```
def segment_delayed_payments(x):
    if pd.notnull(x):
        if x == 0:
            return 5
        elif 1 <= x <= 5:
            return 4
        elif 6 <= x <= 10:
            return 3
        elif 11 <= x <= 15:
            return 2
        elif 16 <= x <= 20:
            return 1
        else:
```

```
            return 0

df['Delayed_Payment_Score'] = df.Num_of_Delayed_Payment.apply(lambda x: segment_delayed_payments(x))
```

In [128…

```
df.head()
```

Out[128…

	Customer_ID	Month	Name	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Int
0	CUS_0xd40	January	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
1	CUS_0xd40	February	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
2	CUS_0xd40	March	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
3	CUS_0xd40	April	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
4	CUS_0xd40	May	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	

Hypothetical Credit Score Calculation

In [128...
```
credit_score_df = df.iloc[:, -9:]
```

In [128...
```
credit_score_df.head()
```

Out[128...

	Credit_Mix_Score	Payment_Behaviour_Score	Credit_History_Age_Months	Credit_History_Score	Credit_Utilization_Score	Outstanding_Deb
0	2	4	265	4	2	
1	2	4	267	4	1	
2	2	4	267	4	2	
3	2	4	268	4	1	
4	2	4	269	4	2	

In [128...
```
df_normalized = (credit_score_df - credit_score_df.min()) / (credit_score_df.max() - credit_score_df.min())
```

In [128...
```
df_normalized.head()
```

Out[128...

	Credit_Mix_Score	Payment_Behaviour_Score	Credit_History_Age_Months	Credit_History_Score	Credit_Utilization_Score	Outstanding_Deb
0	1.0	0.8	0.655087	0.022222	1.0	
1	1.0	0.8	0.660050	0.022222	0.5	
2	1.0	0.8	0.660050	0.022222	1.0	
3	1.0	0.8	0.662531	0.022222	0.5	
4	1.0	0.8	0.665012	0.022222	1.0	

In [128…

```
def calculate_credit_score(row):
    score = (
        (0.20 * (row['Credit_Utilization_Score'])) +
        (0.15 * (row['Credit_History_Score'])) +
        (0.1 * (row['Payment_of_Min_Amount_Score'])) +
        (0.1 * (row['Delayed_Payment_Score'])) +
        (0.1 * (row['Credit_Inquiries_Score'])) +
        (0.1 * (row['Outstanding_Debt_Score'])) +
        (0.15 * (row['Credit_Mix_Score'])) +
        (0.1 * (row['Payment_Behaviour_Score']))
    )
    return score*100
```

In [128…

```
df_normalized['Credit_Score'] = df_normalized.apply(calculate_credit_score, axis=1)
```

In [128…

```
df_normalized.head()
```

Out[128…

	Credit_Mix_Score	Payment_Behaviour_Score	Credit_History_Age_Months	Credit_History_Score	Credit_Utilization_Score	Outstanding_Deb
0	1.0	0.8	0.655087	0.022222	1.0	
1	1.0	0.8	0.660050	0.022222	0.5	
2	1.0	0.8	0.660050	0.022222	1.0	
3	1.0	0.8	0.662531	0.022222	0.5	
4	1.0	0.8	0.665012	0.022222	1.0	

In [129…

```
df_normalized['Credit_Score'].describe()
```

Out[129…
```
count    100000.000000
mean         48.090813
std          11.675146
min          10.000000
25%          39.833333
50%          49.333333
75%          57.166667
max          80.000000
Name: Credit_Score, dtype: float64
```

In [129…
```python
def scale_value(old_value, old_min=0, old_max=100, new_min=300, new_max=900):
    scaled_value = new_min + ((old_value - old_min) * (new_max - new_min)) / (old_max - old_min)
    return scaled_value

old_values = df_normalized['Credit_Score']
scaled_values = [scale_value(val) for val in old_values]

df_normalized['Scaled_Credit_Score'] = scaled_values
```

In [129…
```python
df_normalized['Scaled_Credit_Score'].describe()
```

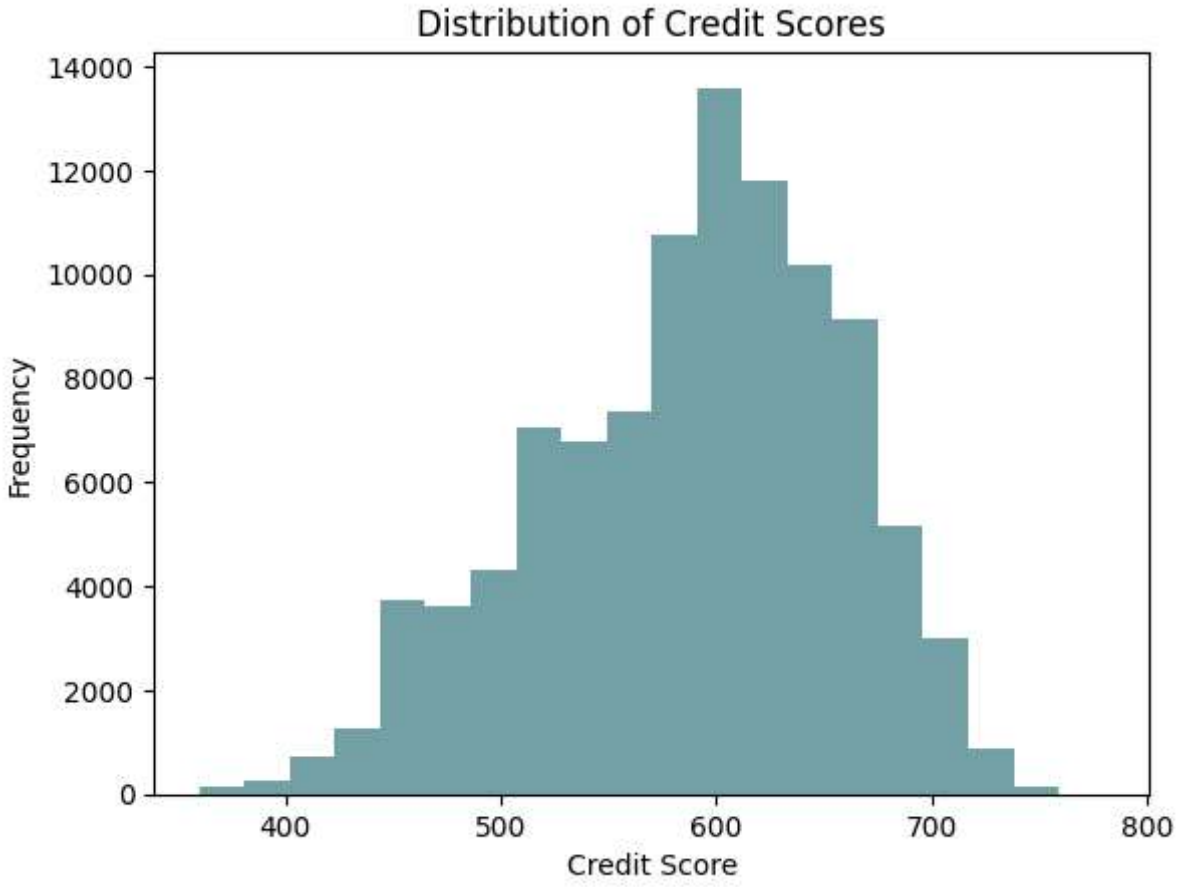
Out[129…
```
count    100000.000000
mean        588.544880
std          70.050879
min         360.000000
25%         539.000000
50%         596.000000
75%         643.000000
max         780.000000
Name: Scaled_Credit_Score, dtype: float64
```

Customers in our dataframe have a credit score range that varies from 360 - 780.

In [129…
```python
df['Credit_Score'] = df_normalized['Scaled_Credit_Score']
```

In [129…
```python
plt.hist(df['Credit_Score'], bins=20, color='#3E7C82', alpha=0.7)
plt.title('Distribution of Credit Scores')
plt.xlabel('Credit Score')
```

```
plt.ylabel('Frequency')
plt.show()
```



In [129…

```
def segment_credit_score(x):
    if pd.notnull(x):
        if x < 300:
            return 'No History'
        elif 300 <= x < 550:
            return 'Poor'
        elif 550 <= x <= 650:
            return 'Fair'
        elif 650 <= x <= 750:
            return 'Good'
```

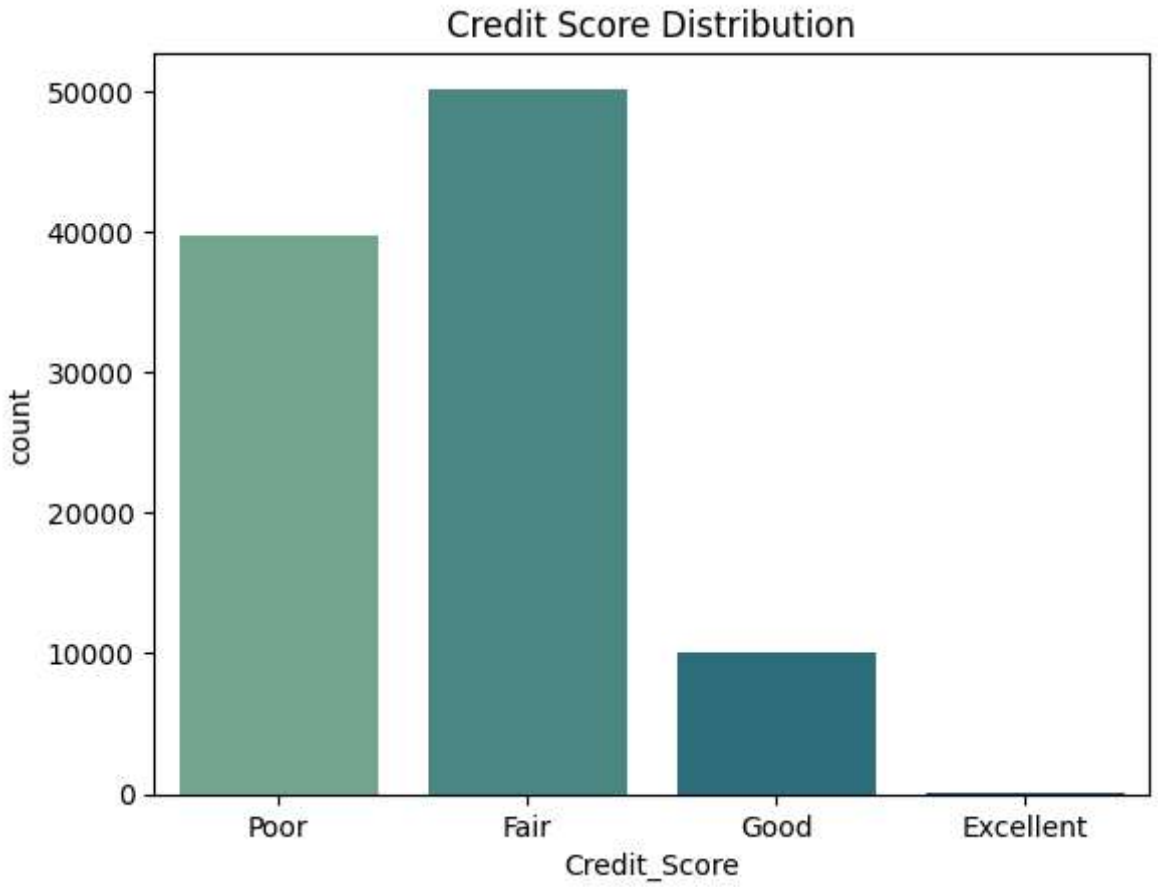
```
        else:
            return 'Excellant'

df['Credit_Score_Segment'] = df.Credit_Score.apply(lambda x: segment_credit_score(x))
```

In [129…

```
score_ranges = pd.cut(df['Credit_Score'], bins=[300, 580, 670, 740, 850], labels=['Poor','Fair', 'Good', 'Excellent'])
score_distribution = score_ranges.value_counts()

sns.barplot(data=score_distribution,palette='crest')
plt.title('Credit Score Distribution')
plt.show()
```



Methodology for Credit Score Calculation

The credit score is a crucial metric used to assess an individual's creditworthiness and potential risk to lenders. In this project, we developed a scoring model that incorporates multiple factors to calculate a hypothetical credit score, with the final scores scaled to a range between 300 and 900.

Scoring Components

The scoring model is based on a weighted sum of various individual scores derived from different aspects of the customer's credit profile. Each component reflects a distinct facet of credit behavior, ensuring a comprehensive evaluation. The components and their respective weights are as follows:

- Credit Utilization Score (20%): This score assesses the ratio of credit used to the total credit limit. Lower utilization indicates responsible credit management, contributing positively to the credit score.
- Credit History Score (15%): This score reflects the length and reliability of an individual's credit history. A longer, positive credit history enhances the score, signaling stability and reliability to lenders.
- Payment of Minimum Amount Score (10%): This score evaluates whether customers consistently pay only the minimum required amount on their credit accounts. Paying more than the minimum is indicative of good credit behavior and is rewarded in the scoring system.
- Delayed Payment Score (10%): This score accounts for the frequency and severity of delayed payments. A lower score is assigned for frequent delays, negatively impacting the overall creditworthiness.
- Credit Inquiries Score (10%): This score considers the number of credit inquiries made by the individual. Excessive inquiries may indicate financial distress, thereby reducing the score.
- Outstanding Debt Score (10%): This score assesses the total outstanding debt of the customer. Higher levels of debt can be a red flag, leading to a lower score.
- Credit Mix Score (15%): This score evaluates the diversity of credit types (e.g., credit cards, loans) held by the individual. A varied credit mix generally indicates a more robust credit profile.

- Payment Behavior Score (10%): This score looks at the overall payment behavior, including consistency and timeliness in fulfilling payment obligations.

- Calculation Process The individual scores are multiplied by their respective weights, reflecting their significance in determining creditworthiness. The weighted scores are then summed to produce a composite score:

$$CreditScore = (0.20 \times CreditUtilizationScore) + (0.15 \times CreditHistoryScore) + (0.10 \times PaymentofMinimumAmountScore) + \\ + (0.10 \times OutstandingDebtScore) + (0.15 \times CreditMixScore) + (0.10 \times Paymen$$

Scaling the Score

Once the raw credit score is calculated, it is scaled to fit within a standardized range of 300 to 900. This is done to align the scores with industry standards and provide a more intuitive understanding of credit risk. The scaling ensures that the final score reflects a credible measure of creditworthiness, with higher scores indicating better credit profiles.

Conclusion

This scoring methodology combines various dimensions of credit behavior into a single metric that aids in risk assessment. By utilizing a systematic and weighted approach, we can derive a comprehensive credit score that provides valuable insights into an individual's creditworthiness, enabling financial institutions to make informed lending decisions.

In [129…

```
df.corr(numeric_only=True)
```

Out[129...

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	N
Age	1.000000	0.091785	0.090753	-0.190795	-0.148665	-0.217841	
Annual_Income	0.091785	1.000000	0.998158	-0.284090	-0.217863	-0.302352	
Monthly_Inhand_Salary	0.090753	0.998158	1.000000	-0.283052	-0.217006	-0.301753	
Num_Bank_Accounts	-0.190795	-0.284090	-0.283052	1.000000	0.442455	0.584453	
Num_Credit_Card	-0.148665	-0.217863	-0.217006	0.442455	1.000000	0.497921	
Interest_Rate	-0.217841	-0.302352	-0.301753	0.584453	0.497921	1.000000	
Num_of_Loan	-0.213684	-0.255560	-0.254316	0.472511	0.417835	0.559156	
Delay_from_due_date	-0.174081	-0.249631	-0.248931	0.558709	0.478365	0.588170	
Num_of_Delayed_Payment	-0.187294	-0.289451	-0.288343	0.611130	0.429535	0.579867	
Changed_Credit_Limit	-0.155976	-0.174765	-0.174614	0.331633	0.254267	0.368115	
Num_Credit_Inquiries	-0.256668	-0.281242	-0.280505	0.522969	0.463219	0.638544	
Outstanding_Debt	-0.202274	-0.269319	-0.269037	0.507073	0.490266	0.629414	
Credit_Utilization_Ratio	0.025508	0.175930	0.176105	-0.071849	-0.055353	-0.075700	
Total_EMI_per_month	-0.054230	0.451305	0.436125	0.058891	0.065674	0.081963	
Amount_invested_monthly	0.016119	0.160620	0.160523	-0.052490	-0.034087	-0.050328	
Monthly_Balance	-0.004857	-0.001906	-0.002445	0.014169	0.004031	0.010505	
Credit_Mix_Score	0.243187	0.342239	0.340962	-0.723768	-0.544726	-0.753124	
Payment_Behaviour_Score	0.046142	0.519274	0.520796	-0.156398	-0.117691	-0.169884	
Credit_History_Age_Months	0.234331	0.272360	0.271441	-0.485328	-0.416936	-0.576226	
Credit_History_Score	0.013780	0.010202	0.009924	-0.030997	-0.028476	-0.031393	
Credit_Utilization_Score	-0.017354	-0.110791	-0.110920	0.044882	0.033388	0.045880	
Outstanding_Debt_Score	0.197020	0.257799	0.257482	-0.493236	-0.473300	-0.608014	

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	N
Credit_Inquiries_Score	0.260559	0.273542	0.272701	-0.514419	-0.442103	-0.621022	
Payment_of_Min_Amount_Score	-0.230356	-0.268284	-0.267605	0.505532	0.359790	0.546340	
Delayed_Payment_Score	0.183817	0.282107	0.281025	-0.595677	-0.421586	-0.571101	
Credit_Score	0.182609	0.377284	0.376897	-0.546703	-0.454546	-0.598634	

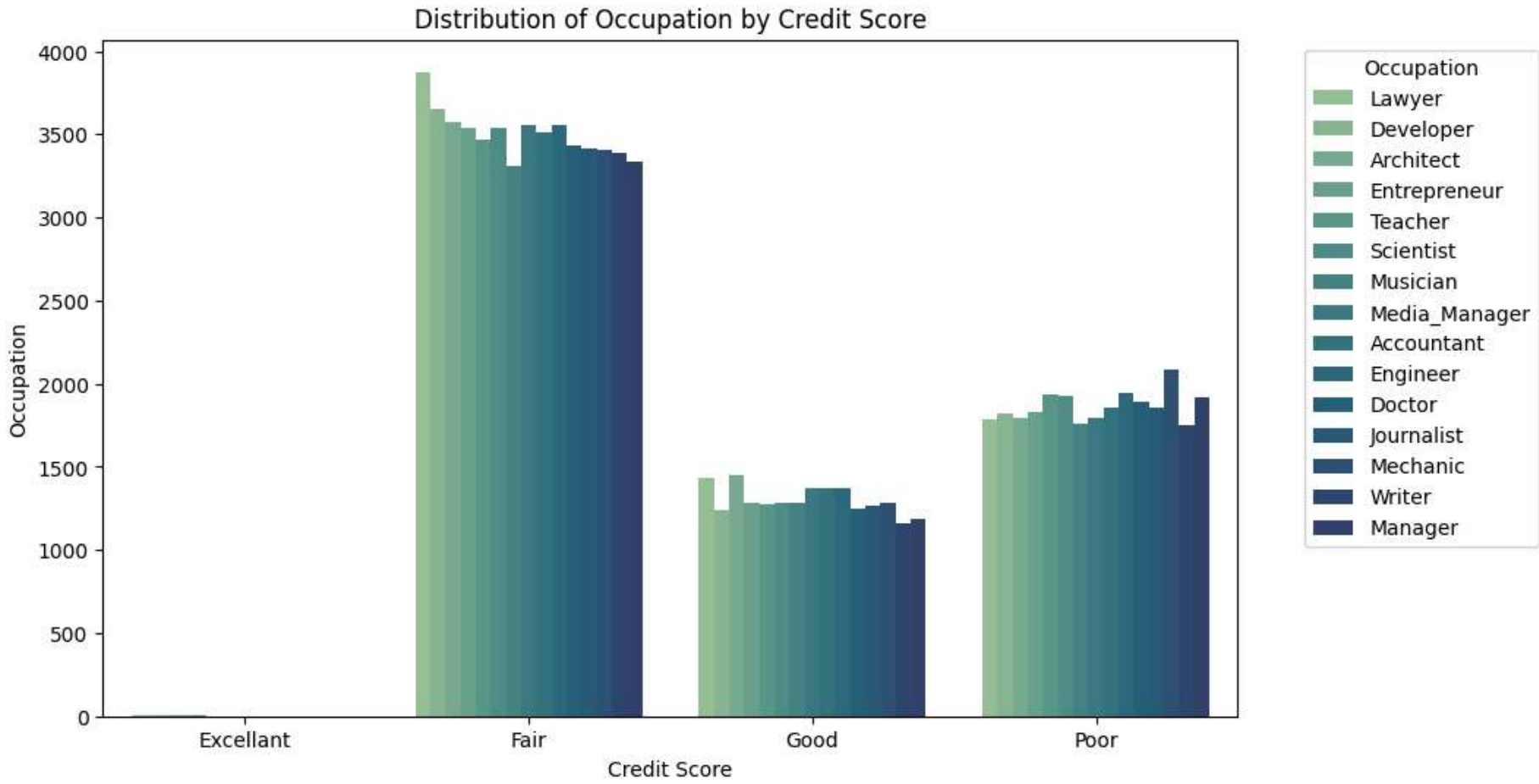
Distribution of various Features by Credit Score

In [129…
```
df['Age'].describe()
```

Out[129…
```
count    100000.000000
mean         33.274560
std          10.764438
min          14.000000
25%          24.000000
50%          33.000000
75%          42.000000
max          56.000000
Name: Age, dtype: float64
```

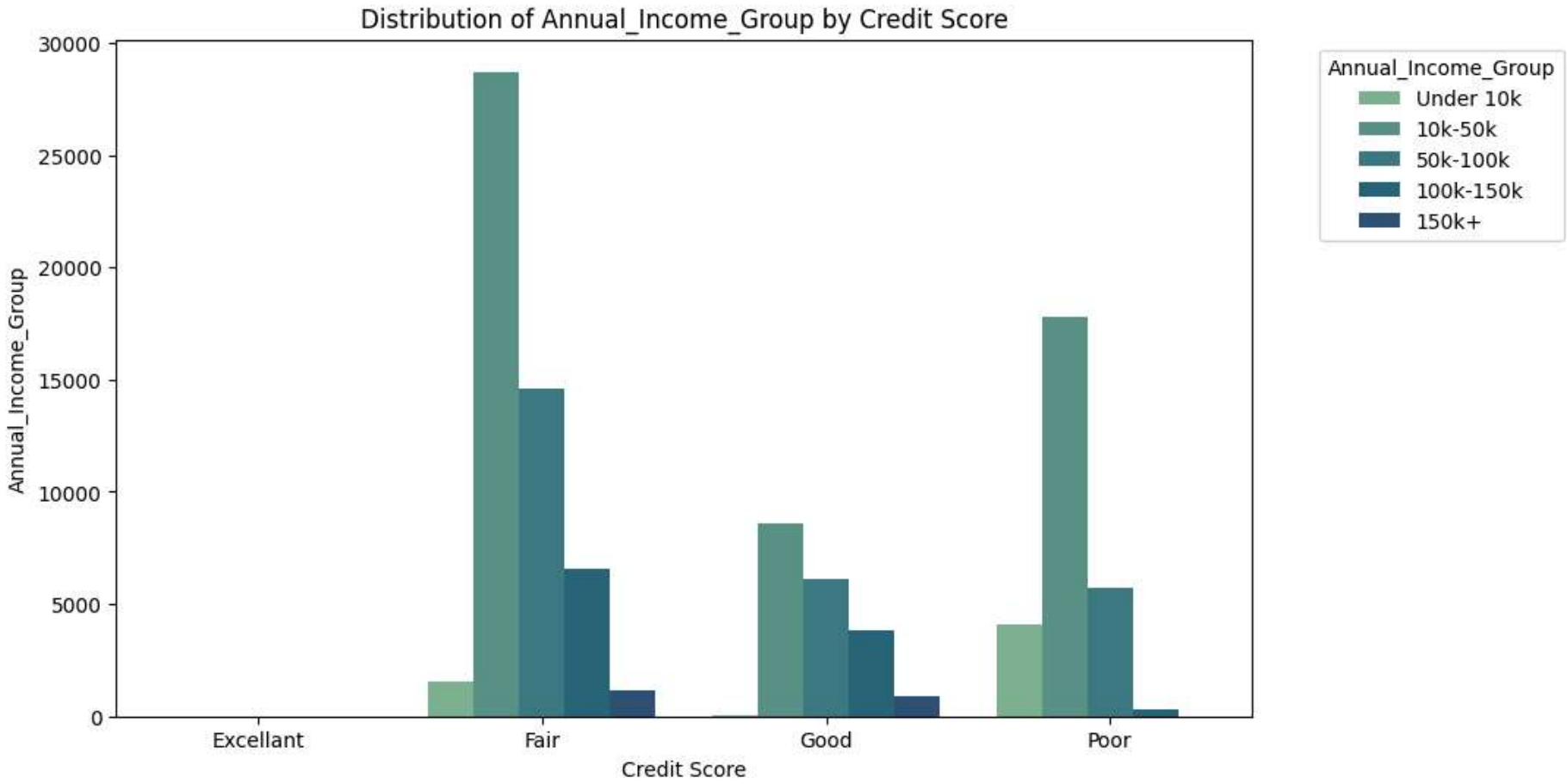
In [129…
```
def column_by_credit_score(col):
    total_by_credit_score = df.groupby('Credit_Score_Segment')[col].value_counts().reset_index(name='Count')
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Credit_Score_Segment', y='Count', hue=col, data=total_by_credit_score, palette='crest')
    plt.title(f'Distribution of {col} by Credit Score')
    plt.xlabel('Credit Score')
    plt.ylabel(f'{col}')
    plt.legend(title=col, bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()

column_by_credit_score('Occupation')
```



In [130…

```
bins = [0, 10000, 50000, 100000, 150000,200000]
labels = ['Under 10k', '10k-50k', '50k-100k', '100k-150k', '150k+']
df['Annual_Income_Group'] = pd.cut(df['Annual_Income'], bins=bins, labels=labels)
column_by_credit_score('Annual_Income_Group')
```



From this bar plot showing the total count of credit scores by annual income groups, we can infer the following:

1. **Fair Credit Score and Income**:
    - The majority of individuals with a **Fair** credit score fall into the $10k$—**50k** income group, followed by a significant number in the $50k$—**100k** group. This suggests that people with mid-range incomes are most likely to have a fair credit score.
    - Those with incomes **under $10k** also have a relatively high count in the "Fair" category, indicating that lower-income individuals tend to fall in this credit range as well.
2. **Good Credit Score and Higher Income**:

- As income increases, the proportion of people with a **Good** credit score also increases, particularly in the $50k$—**100k** and $100k$—**150k** income ranges. This shows a positive correlation between higher income and good credit scores, suggesting that individuals with more financial resources are generally able to maintain better credit.
- However, the total count of individuals with a "Good" credit score remains smaller compared to those with a fair credit score, even for higher income groups.

3. **Poor Credit Score Across Income Groups**:

- Interestingly, individuals with a **Poor** credit score are spread across all income groups, with a noticeable concentration in the $10k$—**50k** and $50k$—**100k** ranges. This implies that even middle-income earners can struggle with managing their credit effectively.
- Lower income groups (under $10k) also have a sizable count in the "Poor" credit category, which is expected as financial challenges at lower income levels often lead to higher credit risk.

4. **Scarcity of Excellent Credit Scores**:

- There are almost no individuals in the **Excellent** credit score category across all income groups, implying that very few people, regardless of income, achieve this score. It highlights the difficulty of reaching and maintaining excellent credit, which may depend on factors beyond income alone, such as financial habits and history.

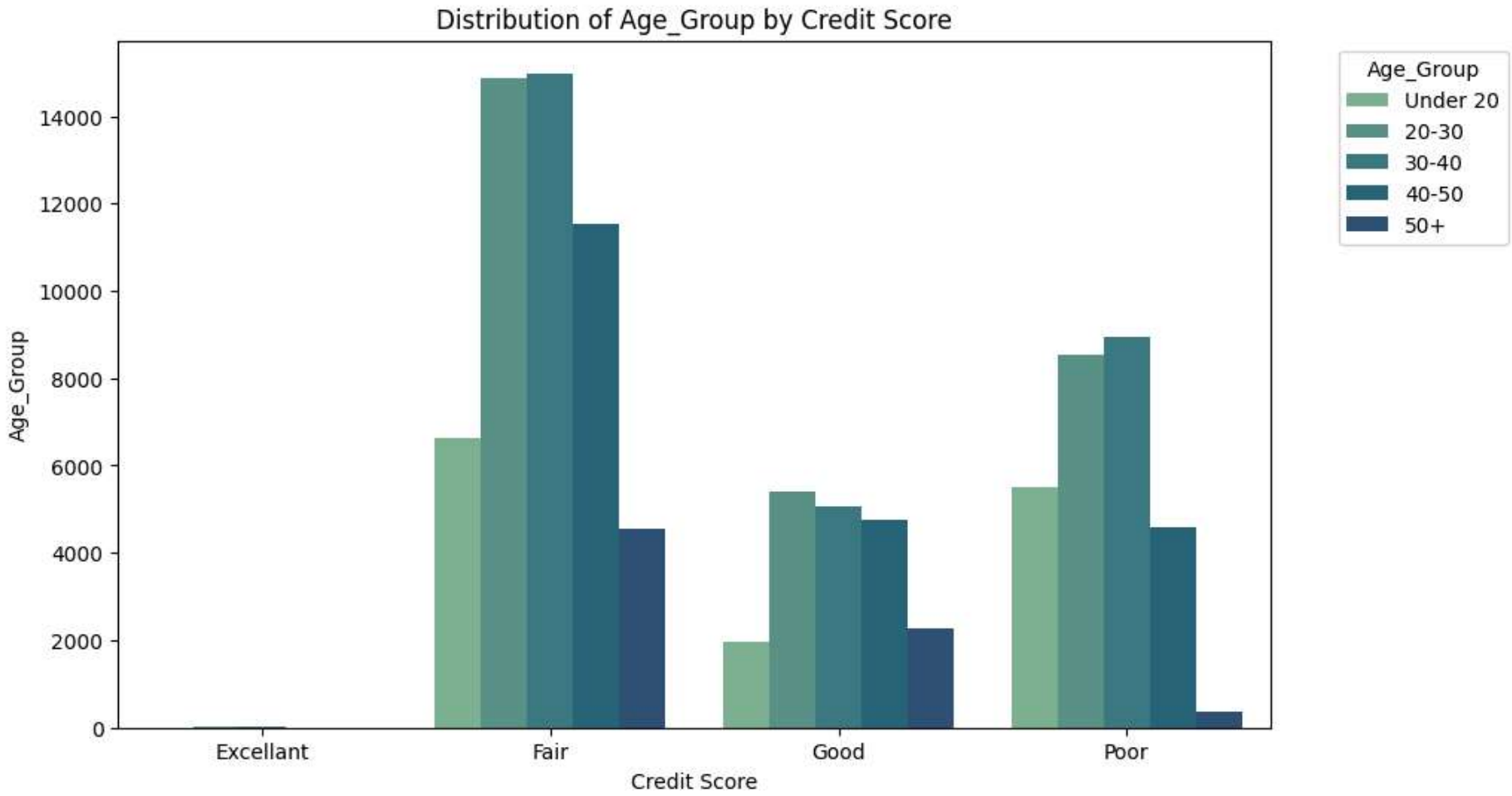
5. **Overall Trend**:

- Higher income groups tend to have better credit scores (Good), while lower and middle-income groups are predominantly represented in the "Fair" and "Poor" categories. This suggests that income is a factor influencing credit score, but not the only determinant, as people from all income levels can still face credit challenges.

In summary, income does play a role in credit scores, with higher incomes generally associated with better scores. However, financial management and other credit-related behaviors likely play an equally important role, as credit challenges are visible across all income levels.

In [130…

```
bins = [0, 20, 30, 40, 50, 56]
labels = ['Under 20', '20-30', '30-40', '40-50', '50+']
df['Age_Group'] = pd.cut(df['Age'], bins=bins, labels=labels)
column_by_credit_score('Age_Group')
```



From the bar plot showing the total count by credit score across different age groups, we can infer several insights:

1. **Credit Score Distribution**: Most individuals fall under the "Fair" credit score category across all age groups. Very few people have "Excellent" credit scores, and a notable portion falls into the "Poor" category.

2. **Age Group Patterns**:

   - The **20-30 and 30-40 age groups** have the highest counts in the "Fair" category. This suggests that younger adults tend to have average credit scores, possibly due to shorter credit histories or higher debt utilization.

- The **40-50 age group** shows a balanced distribution across "Fair," "Good," and "Poor" scores, indicating some financial maturity but also potential risk due to delayed payments or credit issues.
- Individuals **under 20** and those **over 50** are more likely to have "Poor" credit scores, which could be due to a lack of credit history in younger individuals or financial difficulties among older people.

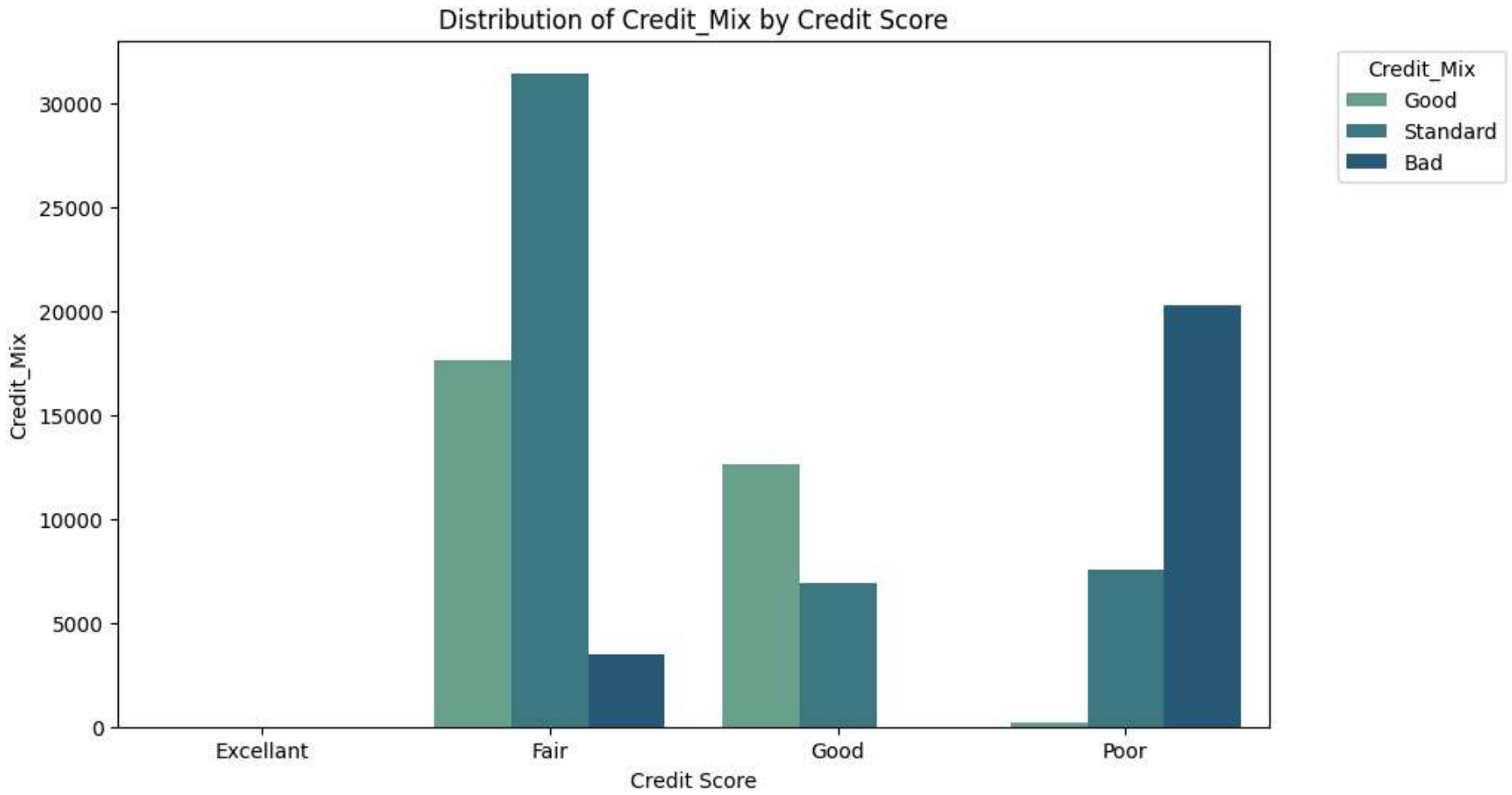
3. **Scarcity of "Excellent" Credit Scores**: There are hardly any individuals with an "Excellent" credit score across all age groups, which could indicate that achieving a high credit score may be difficult or uncommon within the dataset.

4. **Poor Credit Scores**: A significant number of individuals in all age groups have "Poor" credit scores, which is concerning from a financial stability perspective, as these individuals may struggle with higher interest rates or difficulty securing loans.

In summary, most customers have "Fair" credit scores, particularly in the 20-40 age range, while poor credit scores are more frequent for younger and older age groups. This could be a result of factors like financial inexperience, high debt levels, or insufficient credit history.

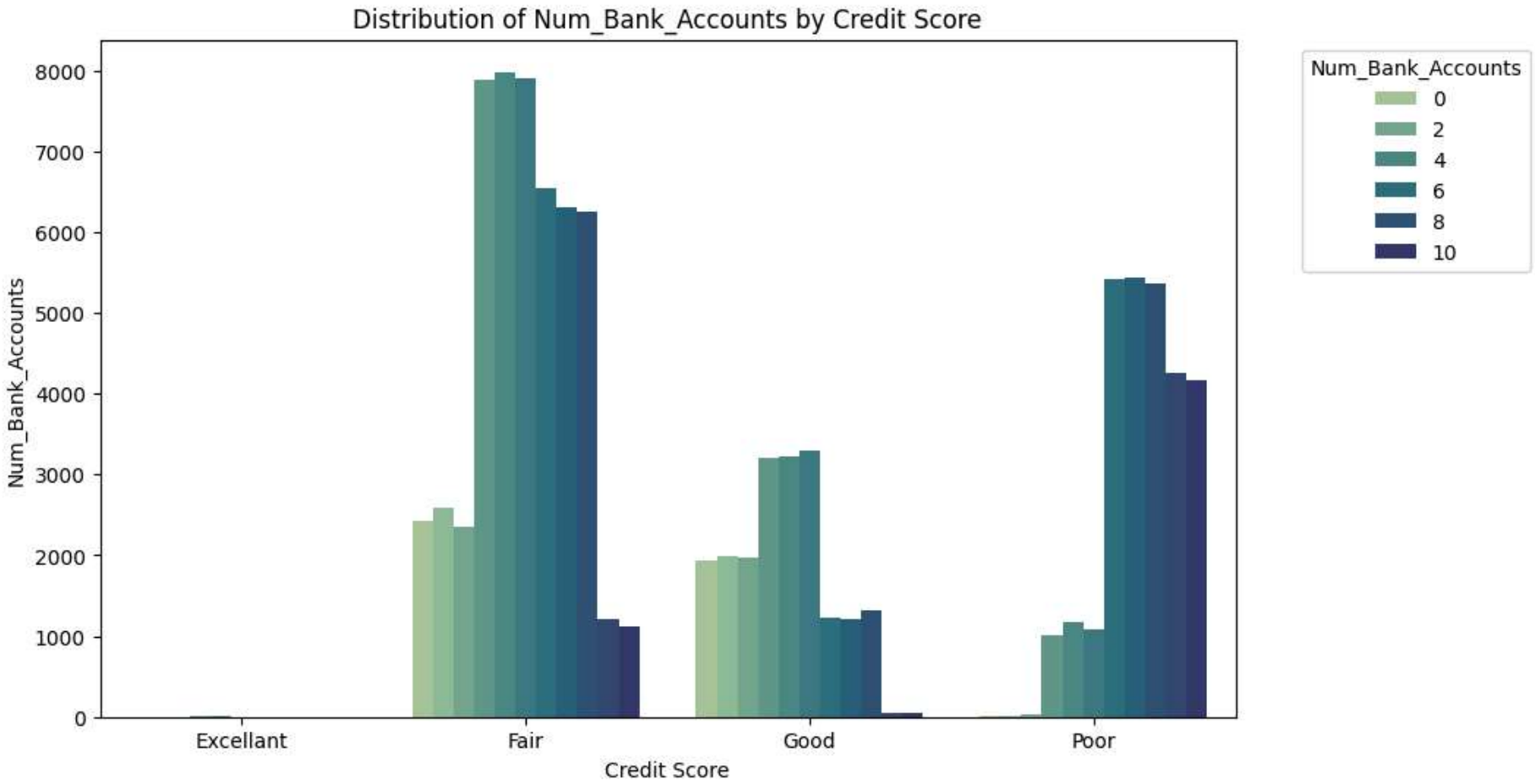
In [130…

```
column_by_credit_score('Credit_Mix')
```



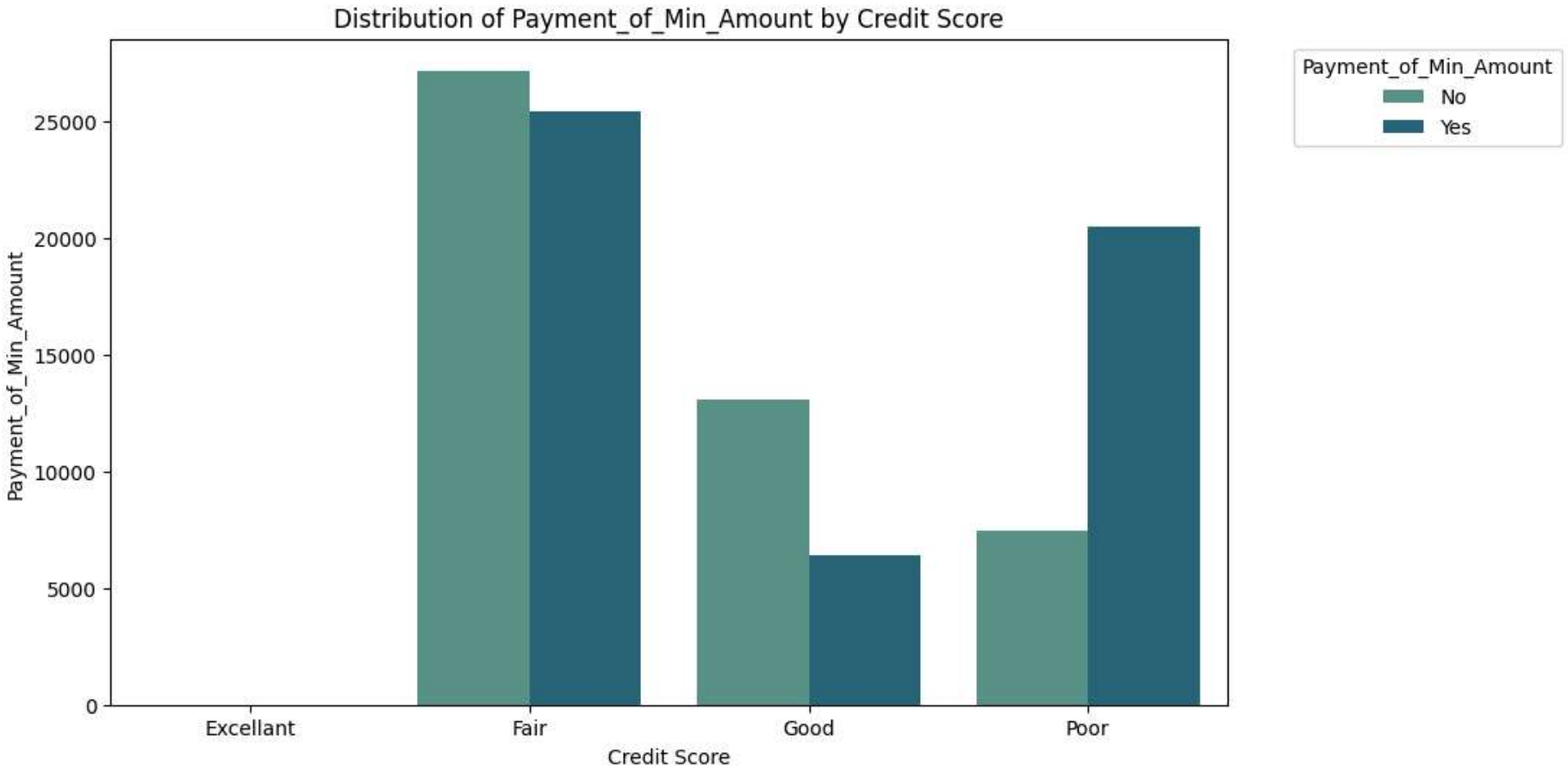
In [130…

```
column_by_credit_score('Num_Bank_Accounts')
```



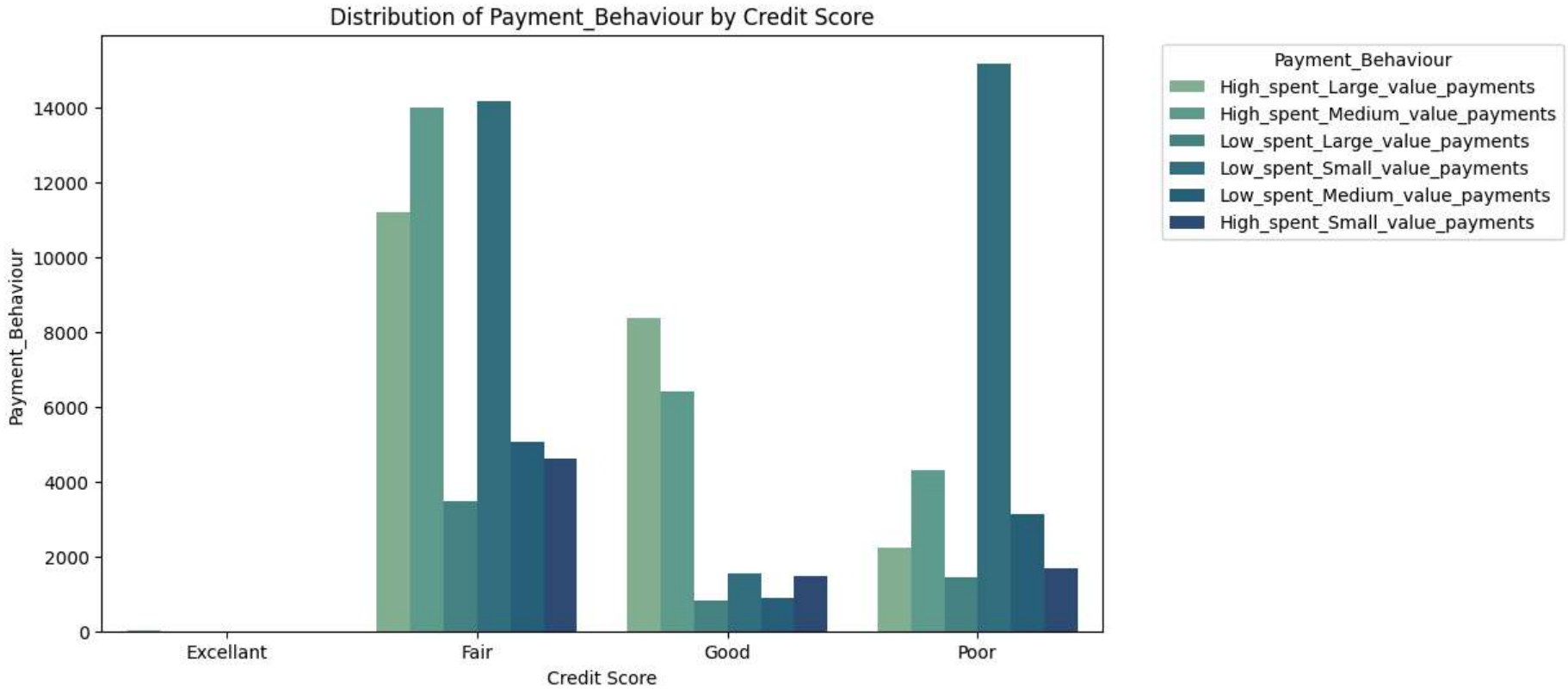
In [130…

```
column_by_credit_score('Payment_of_Min_Amount')
```



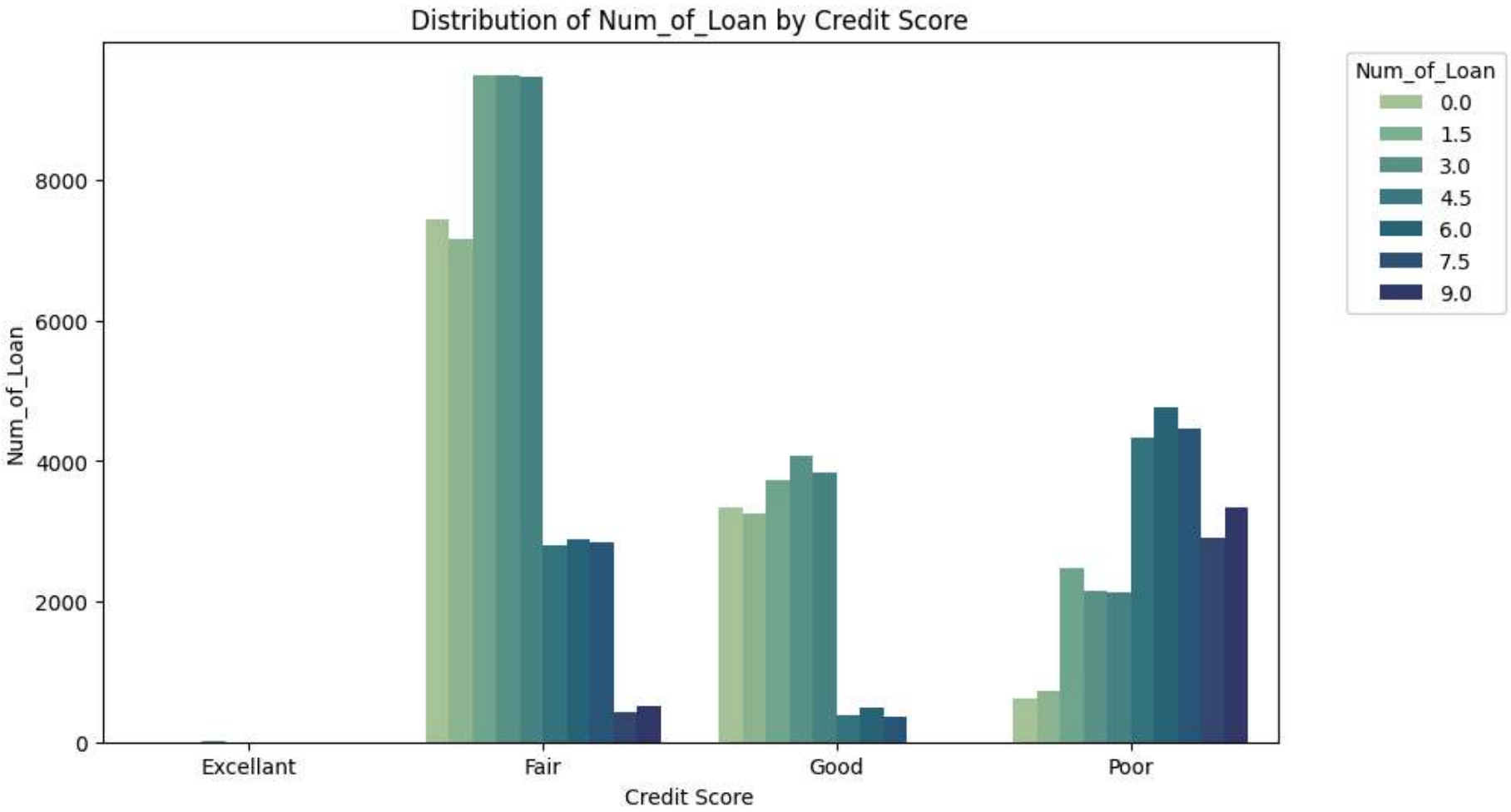
In [130…

```
column_by_credit_score('Payment_Behaviour')
```



In [130…

```
column_by_credit_score('Num_of_Loan')
```



From the bar plot showing the number of loans versus the total count of credit scores, we can infer the following:

1. **Credit Score Distribution and Loan Counts**:

   - **Fair Credit Score**: The majority of individuals with a "Fair" credit score have either 0, 1.5, or 3 loans. This suggests that maintaining a moderate number of loans tends to result in an average (fair) credit score for many customers.
   - **Good Credit Score**: Individuals with a "Good" credit score are distributed across 0 to 6 loans, but those with fewer loans (0 or 1.5) have a higher total count in this category. This indicates that fewer loans are generally associated with better credit scores.

- **Poor Credit Score**: Customers with a "Poor" credit score show a higher count for 4.5 to 9 loans. This suggests that having more loans is correlated with poor credit scores, likely due to the increased financial burden or repayment challenges.

2. **Loan Impact on Credit Score**:

   - As the **number of loans increases**, especially beyond 3 loans, the likelihood of having a "Poor" credit score increases. This highlights the strain that multiple loans can place on an individual's creditworthiness.
   - Conversely, **fewer loans** (especially between 0 and 3) seem to be associated with "Fair" and "Good" credit scores, implying that controlled borrowing has a positive impact on credit health.

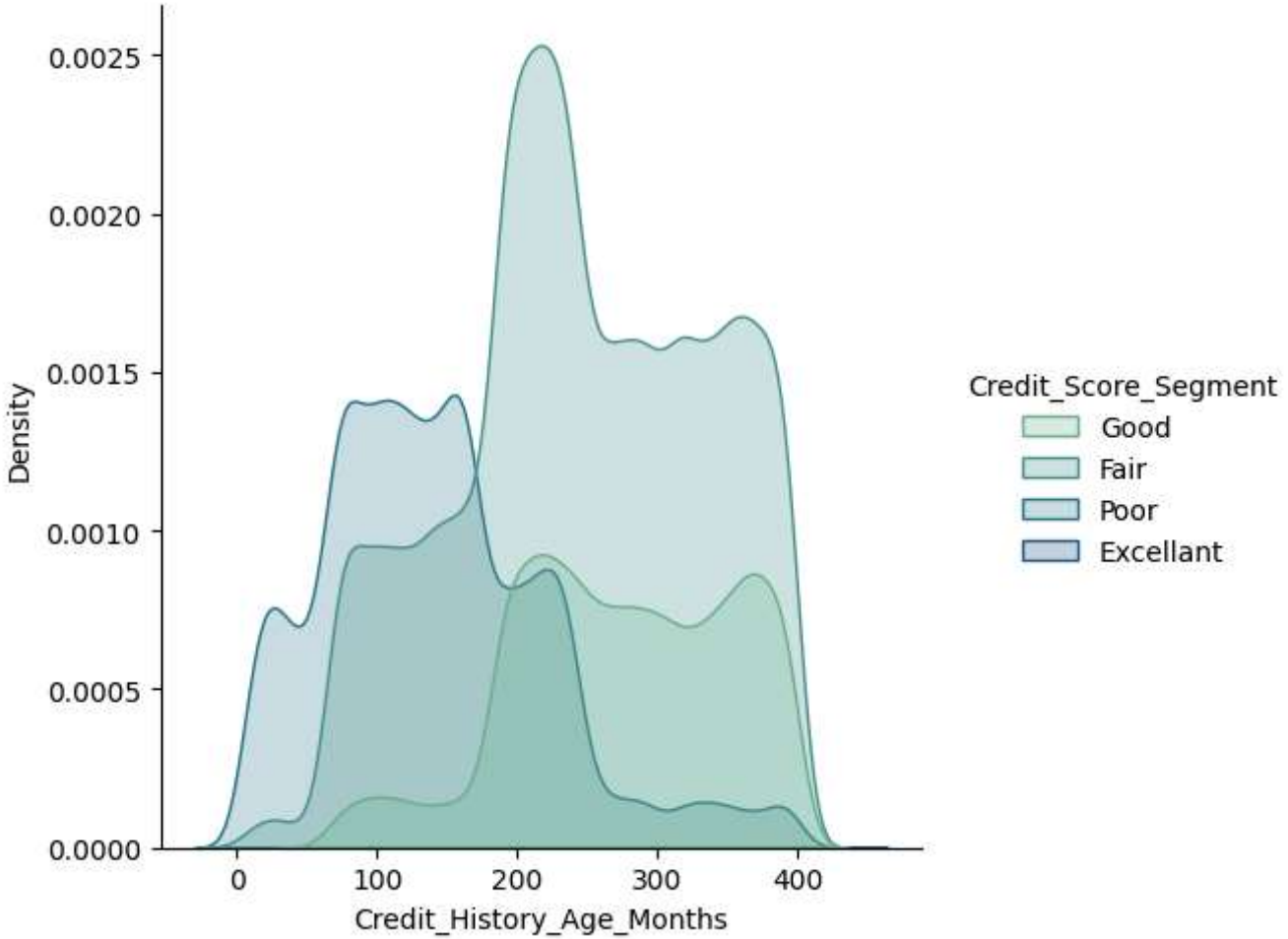
3. **Scarcity of Excellent Scores**: Very few individuals across all loan categories have an "Excellent" credit score, reinforcing that maintaining an excellent score requires not just a minimal number of loans but also other favorable financial behaviors.

In summary, the number of loans has a significant relationship with credit scores, where fewer loans are generally associated with better credit scores, while a higher number of loans tends to coincide with poorer scores.

In [130…

```
sns.displot(
    df,
    x='Credit_History_Age_Months',
    hue='Credit_Score_Segment',
    kind='kde',
    fill=True,palette='crest')

plt.show()
```



Recency Based Metrics

Let's pick last 3 months data and calculate the metrics

In [130…

```
customer_group = df[(df['Month']=='June') | (df['Month']=='July') | (df['Month']=='August')]
customer_group.head()
```

Out[130…

	Customer_ID	Month	Name	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	In
5	CUS_0xd40	June	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
6	CUS_0xd40	July	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
7	CUS_0xd40	August	Aaron Maashoh	23	Scientist	19114.12	1824.843333	3	4	
13	CUS_0x21b1	June	Rick Rothackerj	28	Teacher	34847.84	3037.986667	2	4	
14	CUS_0x21b1	July	Rick Rothackerj	28	Teacher	34847.84	3037.986667	2	4	

In [130…

```
average_credit_score = customer_group.groupby('Customer_ID',group_keys=False).agg({'Credit_Score':'mean'})
average_credit_score.sort_values(by='Credit_Score',ascending=False)
```

Out[130…

Customer_ID	Credit_Score
CUS_0x3ec1	743.0
CUS_0x9da4	740.0
CUS_0x6a79	732.0
CUS_0x7a59	731.0
CUS_0x5205	728.0
...	...
CUS_0x2c15	401.5
CUS_0x992b	401.5
CUS_0x1c89	401.5
CUS_0x7d0b	401.5
CUS_0x84d1	381.0

12500 rows × 1 columns

These are the top 5 and last 5 customers with average credit score

In [131…

```
average_delayed_payments = customer_group.groupby('Customer_ID',group_keys=False).agg({'Delayed_Payment_Score':'sum'})
average_delayed_payments.sort_values(by='Delayed_Payment_Score',ascending=False)
```

Out[131...

Customer_ID	Delayed_Payment_Score
CUS_0x7312	15
CUS_0x319c	15
CUS_0x9011	15
CUS_0x8fee	15
CUS_0xb264	15
...	...
CUS_0x46b5	0
CUS_0x81fe	0
CUS_0x81f4	0
CUS_0xb173	0
CUS_0x1000	0

12500 rows × 1 columns

Insights from Delayed Payment Score (Last 3 Months)

1. **High Delayed Payment Scores (15)**:
    - Customers with no delayed payments in the last 3 months show **strong payment discipline** and a **low-risk profile**.
    - These customers are less likely to default and contribute positively to their credit score.
2. **Low Delayed Payment Scores (0)**:
    - Customers with high delayed payments exhibit a **high-risk profile**, indicating consistent missed payments.
    - These customers need to be flagged for **risk mitigation** strategies as their behavior could harm their credit standing.
3. **Segmentation**:

- Customers can be segmented into risk tiers:
  - **Low Risk**: No delayed payments (scores of 5).
  - **Moderate Risk**: 1-10 delayed payments (scores of 3-4).
  - **High Risk**: 11+ delayed payments (scores of 2 or below).

4. **Actionable Strategies**:

- **For High Risk**: Implement reminders, credit restructuring, or limit adjustments to mitigate risks.
- **For Low Risk**: Offer incentives like lower interest rates or pre-approvals for credit line extensions.

This data helps in **recency-based risk management** by focusing on deteriorating customer behaviors, allowing for proactive risk mitigation.

Change in Credit Score over a period

In [131…

```
new_df = df[(df['Month']=='January') | (df['Month']=='August')][['Customer_ID','Month','Credit_Score']]
new_df.head()
```

Out[131…

	Customer_ID	Month	Credit_Score
0	CUS_0xd40	January	704.0
7	CUS_0xd40	August	704.0
8	CUS_0x21b1	January	728.0
15	CUS_0x21b1	August	668.0
16	CUS_0x2dbc	January	665.0

In [131…

```
def change_in_credit_score(group):
    group = group.sort_values(by='Month')
    return ((group['Credit_Score'].iloc[-1] - group['Credit_Score'].iloc[0])/group['Credit_Score'].iloc[0])*100
```

In [131…

```
percentage_change = new_df.groupby('Customer_ID').apply(change_in_credit_score).reset_index(name='Percentage_Change')
percentage_change.sort_values(by='Percentage_Change',ascending=False).reset_index(drop=True)
```

Out[131...

	Customer_ID	Percentage_Change
0	CUS_0x84d1	33.240997
1	CUS_0x4519	33.195021
2	CUS_0x89a9	33.195021
3	CUS_0xa8ab	32.258065
4	CUS_0xc242	32.171582
...	...	...
12495	CUS_0x85cb	-24.922118
12496	CUS_0x55fd	-24.948025
12497	CUS_0xe7a	-25.000000
12498	CUS_0x9a84	-27.470356
12499	CUS_0xc505	-28.024194

12500 rows × 2 columns

Insights from Percentage Change in Credit Scores

1. **Positive Changes**:
   - Customers with positive percentage changes (e.g., **CUS_0x84d1 at 33.24%**) have improved credit scores.
   - **Inference**: This suggests responsible credit behavior, such as timely payments, indicating lower risk and eligibility for additional credit offerings.
2. **Moderate Improvements**:
   - A number of customers show moderate positive changes (around **32% to 33%**).
   - **Inference**: This may reflect effective credit management practices or successful interventions from financial institutions.
3. **Negative Changes**:

- Customers with negative changes (e.g., **CUS_0x85cb at -24.92%**) have experienced declines in their credit scores.
- **Inference**: These customers may face challenges like missed payments and should be flagged for risk mitigation strategies.

4. **Risk Segmentation**:
   - Customers can be categorized into:
     - **High Growth**: Positive changes.
     - **Moderate Change**: Minimal changes around 0%.
     - **High Risk**: Significant negative changes (e.g., below -20%).
   - **Action**: Tailor communication and support based on these segments.
5. **Overall Trends**:
   - Analyzing the distribution of changes can highlight broader trends in customer behavior, indicating overall economic conditions or institutional effectiveness.
   - A notable number of negative changes may signal emerging issues.
6. **Long-Term Monitoring**:
   - Continuous monitoring is essential to identify consistent patterns and potential risks over time.

The analysis reveals distinct customers, with positive changes indicating responsible management and negative changes highlighting the need for intervention. These insights can guide risk management strategies and customer outreach efforts.

Insights

1. **Trends in Credit Scores**

- **Fluctuations in Credit Scores**: The credit scores show variability over time. Customers with stable payment behavior tend to maintain higher credit scores, whereas late or missed payments lead to score fluctuations.
- **Consistency Across Months**: For certain customers, the credit score alternates between two segments (e.g., "Good" to "Fair") depending on factors such as delayed payments or utilization scores.

2. Feature Importance

The dataset contains multiple features contributing to the credit score:

- **Delayed Payment Score**: Significant impact on credit score. Customers with higher delayed payment scores tend to have lower credit scores.
- **Credit Utilization Score**: Another key feature influencing credit score fluctuations. Customers who consistently utilize less credit are more likely to have a good credit score.
- **Credit History**: A higher number of months in credit history positively correlates with better credit scores, indicating long-term creditworthiness.
- **Outstanding Debt and Payment Behavior**: Both features reflect the ability to manage existing debt, directly influencing the overall credit score.

3. Risk Mitigation Strategies

- **Encouraging Timely Payments**: Implementing automated reminders and incentives for early payments can help reduce the number of delayed payments, leading to better scores.
- **Credit Utilization Management**: Advising customers to keep their credit utilization below a certain threshold can improve their credit score over time.
- **Offering Debt Management Tools**: Providing support to customers with high outstanding debt scores could mitigate financial risks and help improve their credit scores.

4. Future Work

- **Prediction Models**: Utilize machine learning models to predict future credit score trends based on historical data and individual features like credit history and utilization.
- **Customer Segmentation**: Further segment customers based on their risk profiles and tailor mitigation strategies accordingly.
- **Enhanced Visualizations**: Create dashboards to track credit score trends over time, highlight key influencers, and identify customers at risk of falling into lower credit segments.
