ARTIFICIAL INTELLIGENCE CAPSTONE PROJECT

Development and Deployment of Credit Scoring Framework Using Machine Learning

Yen Phi Do

INTRODUCTION

In financial landscape, the absence of robust credit scoring mechanisms poses significant challenges for both lenders and borrowers. Traditional financial institutions often overlook a large portion of the population, leading to the rise of unregulated credit-only institutions that exploit unconventional criteria for lending. To address this issue, our project aims to develop a comprehensive credit scoring framework tailored to context. By leveraging advanced data analytics and machine learning techniques, we seek to enhance financial inclusion and competition while enabling regulated institutions to make informed lending decisions based on diverse datasets, including Know Your Customer (KYC) information. For this endeavor, we will utilize the credit score classification dataset available at Kaggle.

INITIAL SETUP - DATA ACQUISITION

In this section, we will start by importing necessary libraries such as pandas, numpy, matplotlib.pyplot, and seaborn that we will use in the Explorartory Data Analysis Stage.

In [ ]:

```
#importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
```

Then we will load the dataset that we shall be working on using the pd.read_csv() function. The dataset will be loaded to a dataframe named df.

In [ ]:

```
# import the csv files using pandas
df = pd.read_csv('/content/train.csv')
```

```
<ipython-input-174-0ee617eada71>:2: DtypeWarning:

Columns (26) have mixed types. Specify dtype option on import or set low_memory=False.
```

DATA DESCRIPTION

To get an overview of how our data looks like, we will use the df.head() function. This will display the first 5 rows of our dataset and all the columns contained in our dataset.

In [ ]:

```
df.head()
```

Out[ ]:

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

5 rows × 28 columns

The df.duplicate() function will let us know if there are any duplicate rows in our DataFrame 'df'

In [ ]:

```
#checking for duplicates
df[df.duplicated()]
```

Out[ ]:

ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	C

0 rows × 28 columns

The df.info() function will provide more information about our data. It will let us know the data type available in our dataset and whether we have any missing values.

In [ ]:

```
#Getting the data frame information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 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
 27  Credit_Score              100000 non-null  object
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

Generate descriptive statistics of the DataFrame 'df' and transpose the result for better readability

In [ ]:

```
# Get the statistics of the data frame
df.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Monthly_Inhand_Salary	84998.0	4194.170850	3183.686167	303.645417	1625.568229	3093.745000	5957.448333	15204.633333
Num_Bank_Accounts	100000.0	17.091280	117.404834	-1.000000	3.000000	6.000000	7.000000	1798.000000
Num_Credit_Card	100000.0	22.474430	129.057410	0.000000	4.000000	5.000000	7.000000	1499.000000
Interest_Rate	100000.0	72.466040	466.422621	1.000000	8.000000	13.000000	20.000000	5797.000000
Delay_from_due_date	100000.0	21.068780	14.860104	-5.000000	10.000000	18.000000	28.000000	67.000000
Num_Credit_Inquiries	98035.0	27.754251	193.177339	0.000000	3.000000	6.000000	9.000000	2597.000000
Credit_Utilization_Ratio	100000.0	32.285173	5.116875	20.000000	28.052567	32.305784	36.496663	50.000000
Total_EMI_per_month	100000.0	1403.118217	8306.041270	0.000000	30.306660	69.249473	161.224249	82331.000000

It can be observed that we have outlier values in our dataset. For instance, the number of bank accounts has a high max value, this is also true for the number of credit cards and number of credit card inquiries which also do have a high max value.

Generate descriptive statistics for columns in the DataFrame 'df' and transpose the result

In [ ]:

```
#To check the frequency of unique values and the counts
df.describe(exclude=np.number).T
```

Out[ ]:

	count	unique	top	freq
ID	100000	100000	0x25fb6	1
Customer_ID	100000	12500	CUS_0x942c	8
Month	100000	8	January	12500
Name	90015	10139	Langep	44
Age	100000	1788	38	2833
SSN	100000	12501	#F%$D@*&8	5572
Occupation	100000	16	_____	7062
Annual_Income	100000	18940	20867.67	16
Num_of_Loan	100000	434	3	14386

	count	unique	top	freq
Type_of_Loan	88592	6260	Not Specified	1408
Num_of_Delayed_Payment	92998	749	19	5327
Changed_Credit_Limit	100000	4384	_	2091
Credit_Mix	100000	4	Standard	36479
Outstanding_Debt	100000	13178	1360.45	24
Credit_History_Age	90970	404	15 Years and 11 Months	446
Payment_of_Min_Amount	100000	3	Yes	52326
Amount_invested_monthly	95521	91049	__10000__	4305
Payment_Behaviour	100000	7	Low_spent_Small_value_payments	25513
Monthly_Balance	98800	98792	__-333333333333333333333333333__	9
Credit_Score	100000	3	Standard	53174

DATA PREPROCESSING

Essential for preparing raw data for analysis and modeling, improve data quality and enhance model performance.

Iterate through a list of columns named 'wrong_numeric_list' in the DataFrame 'df'

Print unique values for each column in the list

In [ ]:

```
# Getting the unique numeric values in the specific columns
wrong_numeric_list=["Age","Num_of_Loan","Num_of_Delayed_Payment","Outstanding_Debt","Amou
nt_invested_monthly","Monthly_Balance"]

for i in wrong_numeric_list:
    print(df[i].unique())
```

```
['23' '-500' '28_' ... '4808_' '2263' '1342']
['4' '1' '3' '967' '-100' '0' '0_' '2' '3_' '2_' '7' '5' '5_' '6' '8' '8_'
 '9' '9_' '4_' '7_' '1_' '1464' '6_' '622' '352' '472' '1017' '945' '146'
 '563' '341' '444' '720' '1485' '49' '737' '1106' '466' '728' '313' '843'
 '597_' '617' '119' '663' '640' '92_' '1019' '501' '1302' '39' '716' '848'
 '931' '1214' '186' '424' '1001' '1110' '1152' '457' '1433' '1187' '52'
 '1480' '1047' '1035' '1347_' '33' '193' '699' '329' '1451' '484' '132'
 '649' '995' '545' '684' '1135' '1094' '1204' '654' '58' '348' '614'
 '1363' '323' '1406' '1348' '430' '153' '1461' '905' '1312' '1424' '1154'
 '95' '1353' '1228' '819' '1006' '795' '359' '1209' '590' '696' '1185_'
 '1465' '911' '1181' '70' '816' '1369' '143' '1416' '455' '55' '1096'
 '1474' '420' '1131' '904' '89' '1259' '527' '1241' '449' '983' '418'
 '319' '23' '238' '638' '138' '235_' '280' '1070' '1484' '274' '494'
 '1459_' '404' '1354' '1495' '1391' '601' '1313' '1319' '898' '231' '752'
 '174' '961' '1046' '834' '284' '438' '288' '1463' '1151' '719' '198'
 '1015' '855' '841' '392' '1444' '103' '1320_' '745' '172' '252' '630_'
 '241' '31' '405' '1217' '1030' '1257' '137' '157' '164' '1088' '1236'
 '777' '1048' '613' '330' '1439' '321' '661' '952' '939' '562' '1202'
 '302' '943' '394' '955' '1318' '936' '781' '100' '1329' '1365' '860'
 '217' '191' '32' '282' '351' '1387' '757' '416' '833' '359_' '292'
 '1225_' '1227' '639' '859' '243' '267' '510' '332' '996' '597' '311'
 '492' '820' '336' '123' '540' '131_' '1311_' '1441' '895' '891' '50'
 '940' '935' '596' '29' '1182' '1129_' '1014' '251' '365' '291' '1447'
 '742' '1085' '148' '462' '832' '881' '1225' '1412' '785_' '1127' '910'
 '538' '999' '733' '101' '237' '87' '659' '633' '387' '447' '629' '831'
 '1384' '773' '621' '1419' '289' '143_' '285' '1393' '1131_' '27_' '1359'
 '1482' '1189' '1294' '201' '579' '814' '141' '1320' '581' '1171_' '295'
 '290' '433' '679' '1040' '1054' '1430' '1023' '1077' '1457' '1150' '701'
 '1382' '889' '437' '372' '1222' '126' '1159' '868' '19' '1297' '227_'
 '190' '809' '1216' '1074' '571' '520' '1274' '1340' '991' '316' '697'
 '926' '873' '1002' '378_' '65' '875' '867' '548' '652' '1372' '606'
 '1036' '1300' '17' '1178' '802' '1219_' '1271' '1137' '1496' '439' '196'
 '636' '192' '228' '1053' '229' '753' '1296' '1371' '254' '863' '464'
```

```
 '515' '838' '1160' '1289' '1298' '799' '182' '574' '527_' '242' '415'
 '869' '958' '54' '1265' '656' '275' '778' '208' '147' '350' '507' '463'
 '497' '1129' '927' '653' '662' '529' '635' '1027_' '897' '1039' '227'
 '1345' '924' '696_' '1279' '546' '1112' '1210' '526' '300' '1103' '504'
 '136' '1400' '78' '686' '1091' '344' '215' '84' '628' '1470' '968' '1478'
 '83' '1196' '1307' '1132_' '1008' '917' '657' '56' '18' '41' '801' '978'
 '216' '349' '966']
['7' nan '4' '8_' '6' '1' '-1' '3_' '0' '8' '5' '3' '9' '12' '15' '17'
 '10' '2' '2_' '11' '14' '20' '22' '13' '13_' '14_' '16' '12_' '18' '19'
 '23' '24' '21' '3318' '3083' '22_' '1338' '4_' '26' '11_' '3104' '21_'
 '25' '10_' '183_' '9_' '1106' '834' '19_' '24_' '17_' '23_' '2672' '20_'
 '2008' '-3' '538' '6_' '1_' '16_' '27' '-2' '3478' '2420' '15_' '707'
 '708' '26_' '18_' '3815' '28' '5_' '1867' '2250' '1463' '25_' '7_' '4126'
 '2882' '1941' '2655' '2628' '132' '3069' '306' '0_' '3539' '3684' '1823'
 '4128' '1946' '827' '2297' '2566' '904' '182' '929' '3568' '2503' '1552'
 '2812' '1697' '3764' '851' '3905' '923' '88' '1668' '3253' '808' '2689'
 '3858' '642' '3457' '1402' '1732' '3154' '847' '3037' '2204' '3103'
 '1063' '2056' '1282' '1841' '2569_' '211' '793' '3484' '411' '3491'
 '2072' '3050' '1049' '2162' '3402' '2753' '27_' '1718' '1014' '3260'
 '3855' '84' '2311' '3251' '1832' '4069' '3010' '733' '4241' '166' '2461'
 '1749' '3200' '663_' '2185' '4161' '3009' '359' '2015' '1523' '594'
 '1079' '1199' '186' '1015' '1989' '281' '559' '2165' '1509' '3545' '779'
 '192' '4311' '-2_' '2323' '1471' '1538' '3529' '439' '3456' '3040' '2697'
 '3179' '1332' '3175' '3112' '829' '4022' '3870' '4023' '531' '1511'
 '3092' '3191' '2400' '3621' '3536' '544' '1864' '28_' '142' '2300' '264'
 '72' '497' '398' '2222' '3960' '1473' '3043' '4216' '2903' '2658' '-1_'
 '4042' '1323_' '2184' '921' '1328' '3404' '2438' '809' '47' '1996' '4164'
 '1370' '1204' '2167' '4011' '2590' '2594' '2533' '1663' '1018' '2919'
 '3458' '3316' '2589' '2801' '3355' '2529' '2488' '4266' '1243' '739'
 '845' '4107' '1884' '337' '2660' '290' '674' '2450' '3738' '1792' '2823'
 '2570' '775' '960' '482' '1706' '2493' '3623' '3031' '2794_' '2219_'
 '758_' '1849' '3559' '4096' '3726' '1953' '2657' '4043' '2938' '4384'
 '1647' '2694' '3533' '519' '2677' '2413' '-3_' '4139' '2609' '4326'
 '4211' '823' '3011' '1608' '2860' '4219' '4047' '1531' '742' '52' '4024'
 '1673' '49' '2243' '1685' '1869' '2587' '3489' '749' '1164' '2616' '848_'
 '4134' '1530' '1502' '4075' '3845' '1060' '2573' '2128' '328' '640'
 '2585' '2230' '1795' '1180' '1534' '3739' '3313' '4191' '996' '372'
 '3340' '3177' '602' '787' '4135' '3878' '4059' '1218' '4051' '1766'
 '1359' '3107' '585' '1263' '2511' '709' '3632' '4077' '2943' '2793'
 '3245' '2317' '1640' '2237_' '3819' '252' '3978' '1498' '1833' '2737'
 '1192' '1481' '700' '271' '2286' '273' '1215' '3944' '2070' '1478' '3749'
 '871' '2508' '2959' '130' '294' '3097_' '3511' '415' '2196' '2138' '2149'
 '1874' '1553' '3847' '3222' '1222' '2907' '3051' '98' '1598' '416' '2314'
 '2955' '1691' '1450' '2021' '1636' '80' '3708' '195' '320' '2945' '1911'
 '3416' '3796' '4159' '2255' '938' '4397' '3776' '2148' '1994' '853'
 '1178' '1633' '196' '3864' '714' '1687' '1034' '468' '1337' '2044' '1541'
 '3661' '1211' '2645' '2007' '102' '1891' '3162' '3142' '2566_' '2766'
 '3881' '2728' '2671' '1952' '3580' '2705' '4251' '3840_' '972' '3119'
 '3502' '4185' '2954' '683' '1614' '1572' '4302' '3447' '1852' '2131'
 '1900' '1699' '133' '2018' '2127' '508' '210' '577' '1664' '2604' '1411'
 '2351' '867' '1371' '2352' '1191' '905' '4053' '3869' '933' '3660' '3300'
 '3629' '3208' '2142' '2521' '450' '583' '876' '121' '3919' '2560' '2578'
 '2060' '813' '1236' '1489' '4360' '1154' '2544' '4172' '2924' '426'
 '4270' '2768' '3909' '3951' '2712' '2498' '3171' '1750' '197' '2569'
 '265' '4293' '887' '2707' '2397' '4337' '4249' '2751' '2950' '1859' '107'
 '2348' '2506' '2810' '2873' '1301' '2262' '1890' '3078' '3865' '3268'
 '2777' '3105' '1278' '3793' '2276' '2879' '4298' '2141' '223' '2239'
 '846' '1862' '2756' '1181' '1184' '2617' '3972' '2334' '3900' '2759'
 '4169' '2280' '2492' '2729' '3750' '1825' '309' '2431' '3099' '2080'
 '2279' '2666' '3722' '1976' '529' '1985' '3060' '4278' '3212' '46' '3148'
 '3467' '4231' '3790' '473' '1536' '3955' '2324' '2381' '1177' '371'
 '2896' '3880' '2991' '4319' '1061' '662' '4144' '693' '2006' '3115'
 '2278_' '3751' '1861' '4262' '2913' '2615' '3492' '800' '3766' '384'
 '3407' '1087' '3329' '1086' '2216' '1087_' '2457' '3522' '3274' '3488'
 '2854' '238' '351' '3706' '4280' '4095' '2926' '1329' '3370' '283' '1392'
 '1743' '2429' '974' '3156' '1133' '4388' '3243' '4282' '2523' '4281'
 '3415' '2001' '441' '94' '3499' '969' '3368' '106' '1004' '2638' '3946'
 '2956' '4324' '85' '4113' '819' '615' '1172' '2553' '1765' '3495' '2820'
 '4239' '4340' '1295_' '2636' '4295' '1653' '1325' '1879' '1096' '1735'
 '3584' '1073' '1975' '3827' '2552' '3754' '2378' '532' '926' '2376'
 '3636' '3763' '778' '2621' '804' '754' '2418' '4019' '3926' '3861_'
 '3574' '175' '162' '2834' '3765' '2354' '523' '2274' '1606' '1443' '1354'
```

```
 '2142_' '1422' '2278' '1045' '4106' '3155' '666' '659' '3229' '1216'
 '2076' '1473_' '2384' '1954' '719' '2534' '4002' '541' '2875' '4344'
 '2081' '3894' '1256' '676' '4178' '399' '86' '1571' '4037' '1967' '4005'
 '3216' '1150' '2591' '1801' '3721' '1775' '2260' '3707' '4292' '1820'
 '145' '1480' '1850' '430' '217' '3920_' '1389' '1579' '3391' '2385'
 '3336' '3392' '3688' '221' '2047']
['809.98' '605.03' '1303.01' ... '3571.7_' '3571.7' '502.38']
['80.41529543900253' '118.28022162236736' '81.699521264648' ...
 '24.02847744864441' '251.67258219721603' '167.1638651610451']
['312.49408867943663' '284.62916249607184' '331.2098628537912' ...
 516.8090832742814 319.1649785257098 393.6736955618808]
```

Iterate through a list of columns named 'wrong_numeric_list' in the DataFrame 'df'

Replace underscores in each column and attempt to convert the values to float

In [ ]:

```
#Replace underscore with nothing
for i in wrong_numeric_list:
    df[i]=df[i].str.replace("_","")
    try:
        df[i]=df[i].astype("float")
        df[i]=df[i].astype("float")
    except:
        continue
```

Iterate through the columns in the 'wrong_numeric_list' and print the column name along with its data type

In [ ]:

```
#checking the data types in the choosen columns
for i in wrong_numeric_list:
    print(i,": ",df[i].dtype)
```

```
Age :  float64
Num_of_Loan :  float64
Num_of_Delayed_Payment :  float64
Outstanding_Debt :  float64
Amount_invested_monthly :  float64
Monthly_Balance :  float64
```

Display the dimensions of the DataFrame 'df' (number of rows and columns)

In [ ]:

```
#Getting the number of rows and columns in the data frame
df.shape
```

Out[ ]:

```
(100000, 28)
```

Display the column names of the DataFrame 'df'

In [ ]:

```
#checking the column names of our data
df.columns
```

Out[ ]:

```
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',
       'Credit_Score'],
      dtype='object')
```

Check and display the sum of null values in each column of the DataFrame 'df'

In [ ]:

```
#finding the missing values in the variables
df.isnull().sum()
```

Out[ ]:

```
ID                            0
Customer_ID                   0
Month                         0
Name                       9985
Age                           0
SSN                           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          0
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            2868
Credit_Score                  0
dtype: int64
```

From is.null().sum() shows that name, Monthly_Inhand_Salary , Type of loan, Num_of_Delayed_Payments, Num_Credit_Inquiries,credit_History_Age,Amount_Invested_Monthly and Monthly_Balance have null values.

DATA MANIPULATION

Ensures data quality, consistency, and relevance for downstream analyses and applications.

Drop specified columns from the DataFrame 'df'

In [ ]:

```
#droping the columns that does not add any significance to our data during machine learni
ng
df.drop(['Name','ID','Customer_ID','SSN','Credit_Mix','Credit_History_Age','Payment_Behav
iour','Changed_Credit_Limit','Type_of_Loan','Annual_Income'],axis=1,inplace=True)
```

Convert specified columns in the DataFrame 'df' to numeric, replacing any non-numeric values with NaN

In [ ]:

```
#Replacing the non-numeric data to numeric values for further analysis
num_cols=['Monthly_Inhand_Salary', 'Num_Bank_Accounts','Num_Credit_Card', 'Interest_Rate'
, 'Num_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment',
```

```
        'Num_Credit_Inquiries', 'Outstanding_Debt',
        'Credit_Utilization_Ratio', 'Total_EMI_per_month',
        'Amount_invested_monthly','Age']

df[num_cols]=df[num_cols].apply(pd.to_numeric,errors="coerce")
```

In [ ]:

```
#Getting the data frame info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Month                     100000 non-null  object 
 1   Age                       100000 non-null  float64
 2   Occupation                100000 non-null  object 
 3   Monthly_Inhand_Salary     84998 non-null   float64
 4   Num_Bank_Accounts         100000 non-null  int64  
 5   Num_Credit_Card           100000 non-null  int64  
 6   Interest_Rate             100000 non-null  int64  
 7   Num_of_Loan               100000 non-null  float64
 8   Delay_from_due_date       100000 non-null  int64  
 9   Num_of_Delayed_Payment    92998 non-null   float64
 10  Num_Credit_Inquiries      98035 non-null   float64
 11  Outstanding_Debt          100000 non-null  float64
 12  Credit_Utilization_Ratio  100000 non-null  float64
 13  Payment_of_Min_Amount     100000 non-null  object 
 14  Total_EMI_per_month       100000 non-null  float64
 15  Amount_invested_monthly   95521 non-null   float64
 16  Monthly_Balance           97132 non-null   float64
 17  Credit_Score              100000 non-null  object 
dtypes: float64(10), int64(4), object(4)
memory usage: 13.7+ MB
```

Replace values in the 'Age' column of the DataFrame 'df' that are less than or equal to 0 with 50

Replace values in the 'Age' column of the DataFrame 'df' that are greater than or equal to 100 with 50

In [ ]:

```
#Normalizing the age column
df.loc[df['Age'] <= 0, 'Age'] = 50
df.loc[df['Age'] >= 100, 'Age'] = 50
```

Calculate and display the mean of the 'Age' column

In [ ]:

```
#getting the mean value of the variable age
df["Age"].mean()
```

Out[ ]:

```
33.78572
```

Check and display the sum of null values in each column

In [ ]:

```
#checking number of null values in each column
df.isnull().sum()
```

Out[ ]:

```
Month                          0
Age                            0
Occupation                     0
Monthly_Inhand_Salary      15002
```

```
Monthly_Inhand_Salary      15002
Num_Bank_Accounts              0
Num_Credit_Card                0
Interest_Rate                  0
Num_of_Loan                    0
Delay_from_due_date            0
Num_of_Delayed_Payment      7002
Num_Credit_Inquiries        1965
Outstanding_Debt               0
Credit_Utilization_Ratio       0
Payment_of_Min_Amount          0
Total_EMI_per_month            0
Amount_invested_monthly     4479
Monthly_Balance             2868
Credit_Score                   0
dtype: int64
```

In [ ]:

```
#checking the mean value of each column
df.mean()
```

```
<ipython-input-192-d888385a4346>:2: FutureWarning:

The default value of numeric_only in DataFrame.mean is deprecated. In a future version, i
t will default to False. In addition, specifying 'numeric_only=None' is deprecated. Selec
t only valid columns or specify the value of numeric_only to silence this warning.
```

Out[ ]:

```
Age                         3.378572e+01
Monthly_Inhand_Salary       4.194171e+03
Num_Bank_Accounts           1.709128e+01
Num_Credit_Card             2.247443e+01
Interest_Rate               7.246604e+01
Num_of_Loan                 3.009960e+00
Delay_from_due_date         2.106878e+01
Num_of_Delayed_Payment      3.092334e+01
Num_Credit_Inquiries        2.775425e+01
Outstanding_Debt            1.426220e+03
Credit_Utilization_Ratio    3.228517e+01
Total_EMI_per_month         1.403118e+03
Amount_invested_monthly     6.374130e+02
Monthly_Balance            -3.088580e+22
dtype: float64
```

Fill missing values in specific columns with predetermined values

In [ ]:

```
#Replacing the null values with respective mean
df["Monthly_Inhand_Salary"].fillna(np.mean(df["Monthly_Inhand_Salary"]),inplace=True)
df["Num_of_Delayed_Payment"].fillna(30, inplace=True)
df["Num_Credit_Inquiries"].fillna(28, inplace=True)
df["Outstanding_Debt"].fillna(1427, inplace=True)
df["Amount_invested_monthly"].fillna(195, inplace=True)
df["Num_of_Loan"].fillna(3, inplace=True)
df["Monthly_Balance"].fillna(403, inplace=True)
df["Age"].fillna(34, inplace=True)
```

In [ ]:

```
#checking that all columns are clean with no null values
df.isnull().sum()
```

Out[ ]:

```
Month                       0
Age                         0
Occupation                  0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
```

```
Num_Bank_Accounts          0
Num_Credit_Card            0
Interest_Rate              0
Num_of_Loan                0
Delay_from_due_date        0
Num_of_Delayed_Payment     0
Num_Credit_Inquiries       0
Outstanding_Debt           0
Credit_Utilization_Ratio   0
Payment_of_Min_Amount      0
Total_EMI_per_month        0
Amount_invested_monthly    0
Monthly_Balance            0
Credit_Score               0
dtype: int64
```

In [ ]:

```
#checking the mnumber of rows and columns in our dataframe
df.shape
```

Out[ ]:

```
(100000, 18)
```

Replace values in the 'Age' column based on conditions in the 'Num_of_Loan' column

In [ ]:

```
# Normalizing the age variable based on the Num_of_Loan
# Update the 'Age' column to 3 for rows where 'Num_of_Loan' is less than 0
df.loc[df['Num_of_Loan'] < 0, 'Age'] = 3

# Update the 'Age' column to 3 for rows where 'Num_of_Loan' is greater than 10
df.loc[df['Num_of_Loan'] > 10, 'Age'] = 3
```

In [ ]:

```
# Display the unique values in the "Occupation" column of the DataFrame df
print(df["Occupation"].unique())
```

Out[ ]:

```
array(['Scientist', '_______', 'Teacher', 'Engineer', 'Entrepreneur',
       'Developer', 'Lawyer', 'Media_Manager', 'Doctor', 'Journalist',
       'Manager', 'Accountant', 'Musician', 'Mechanic', 'Writer',
       'Architect'], dtype=object)
```

In [ ]:

```
# Replace specific values in the "Occupation" column of the DataFrame df
# Replace the value "_______" with "Others"
df["Occupation"].replace("_______", "Others", inplace=True)

# Display the unique values in the "Occupation" column after replacement
# This helps to verify that the replacement was successful
print(df["Occupation"].unique())
```

Out[ ]:

```
array(['Scientist', 'Others', 'Teacher', 'Engineer', 'Entrepreneur',
       'Developer', 'Lawyer', 'Media_Manager', 'Doctor', 'Journalist',
       'Manager', 'Accountant', 'Musician', 'Mechanic', 'Writer',
       'Architect'], dtype=object)
```

DATA EXPLORATION - DATA VISUALIZATION

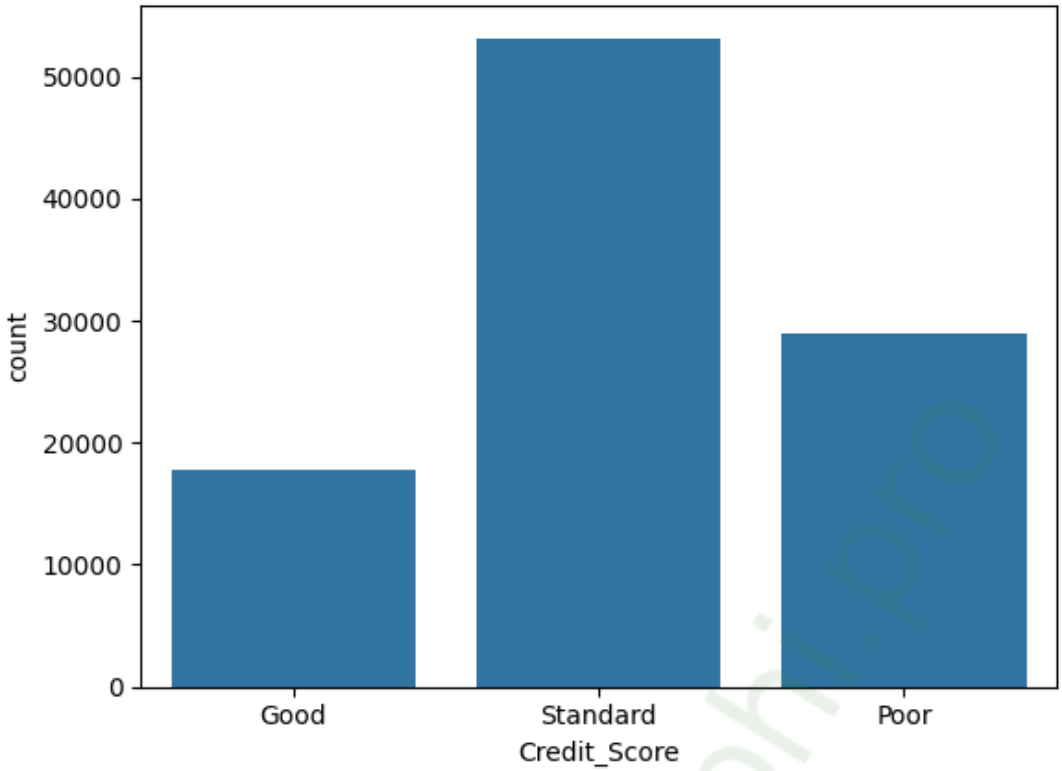
Data exploration through visualization involves using graphical representations to gain insights into the structure, patterns, and relationships within the dataset. By creating plots, charts, and graphs, we can identify trends, outliers, distributions, and correlations among variables to understand the underlying characteristics of

the data and informs subsequent analysis and modeling decisions.

Create a countplot using seaborn to visualize the distribution of 'Credit_Score'

In [ ]:

```
#Visualization of credit score
sns.countplot(data=df, x="Credit_Score")
plt.show()
```

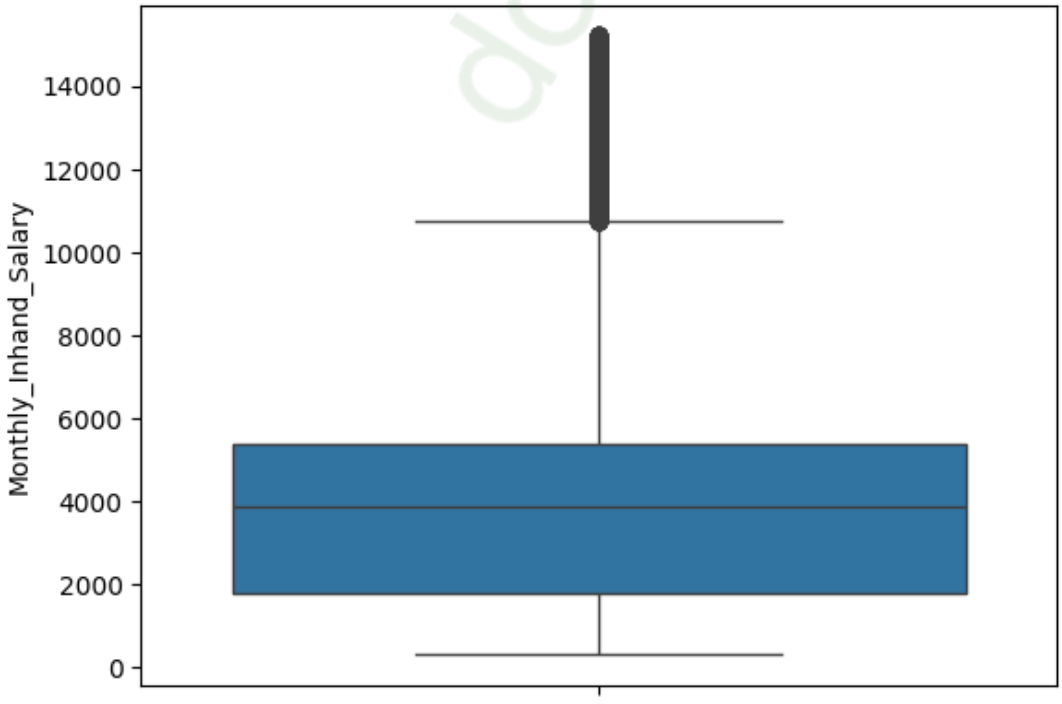


In [ ]:

```
#Visualization of Monthly_Inhand_Salary variable
sns.boxplot(df['Monthly_Inhand_Salary'])
```

Out[ ]:

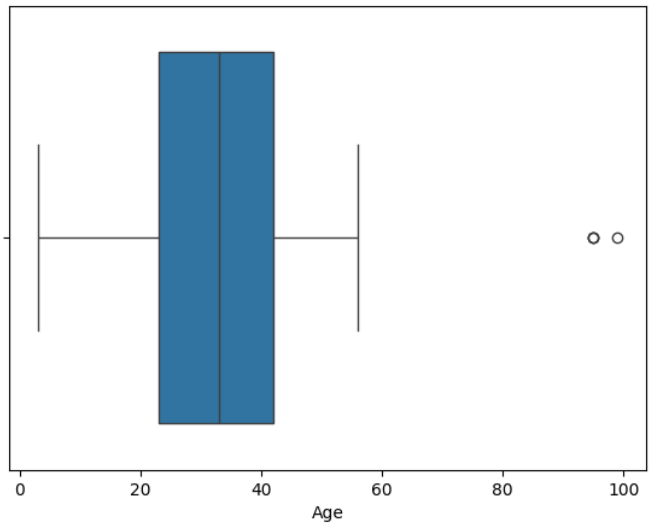
```
<Axes: ylabel='Monthly_Inhand_Salary'>
```

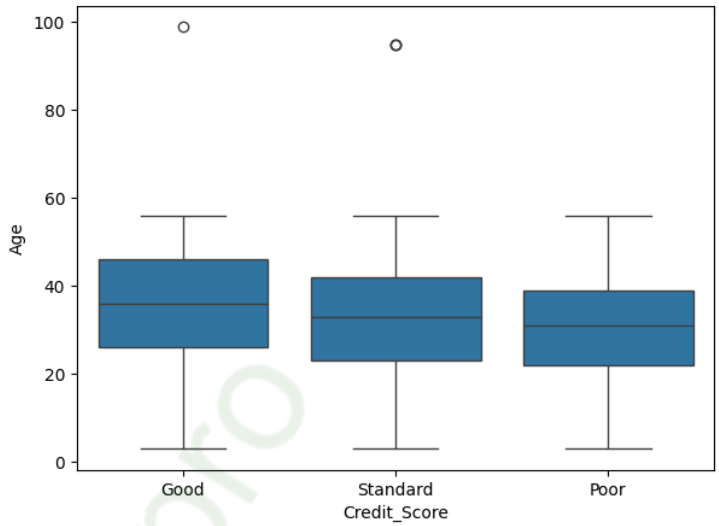


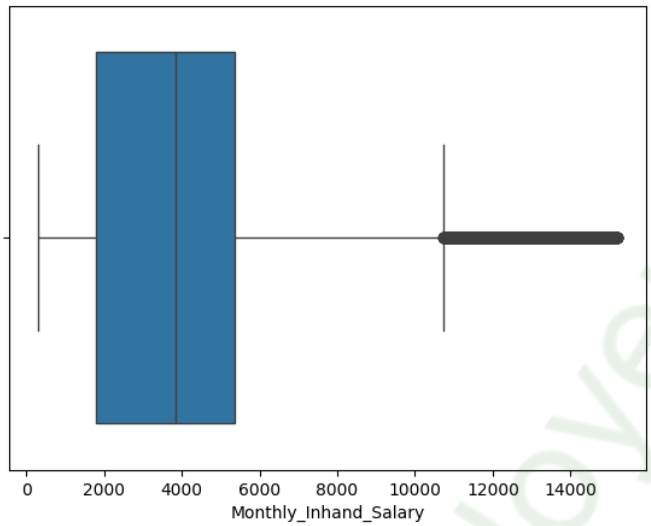
To see how each variable compares to credit score in a box plot

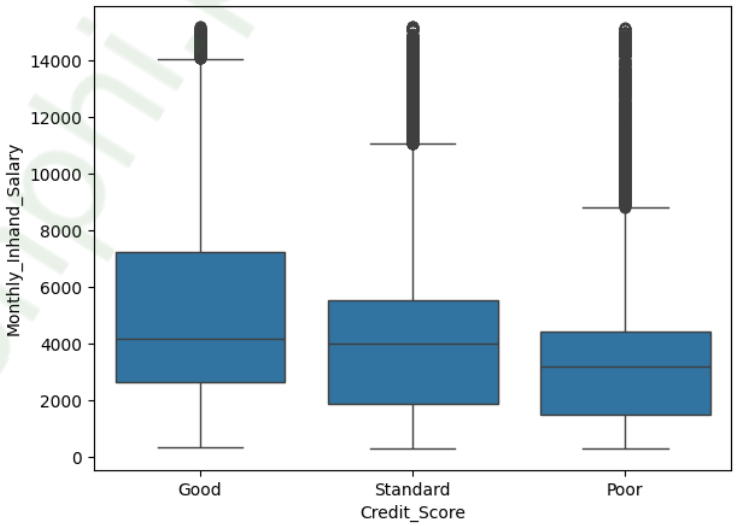
In [ ]:

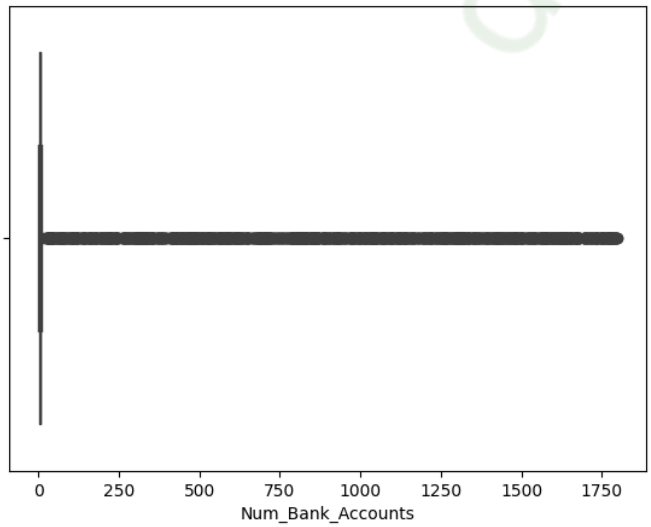
```
#comparison of credit score with different variables
for col in df.select_dtypes(include = np.number):
    plt.figure(figsize=(15,5))
    plt.subplot(1,2,1)
    sns.boxplot(x=col, data =df)
    plt.subplot(1,2,2)
    sns.boxplot(x='Credit_Score',y=col, data =df)
    plt.show()
```

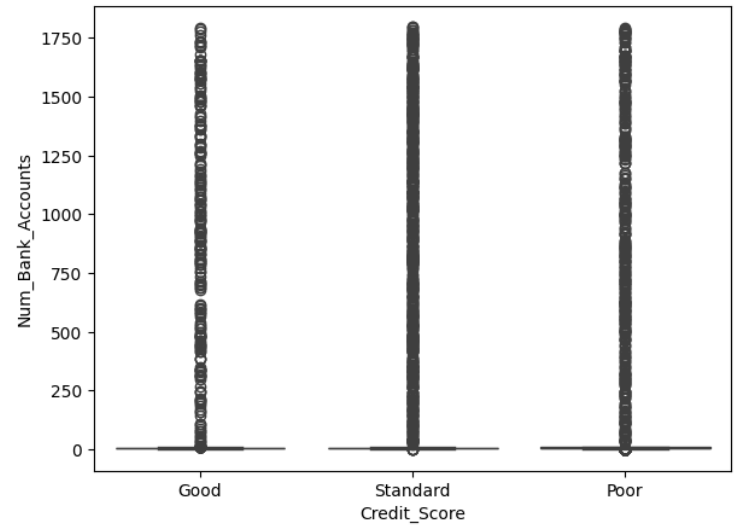


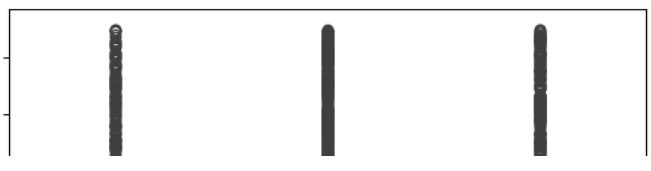


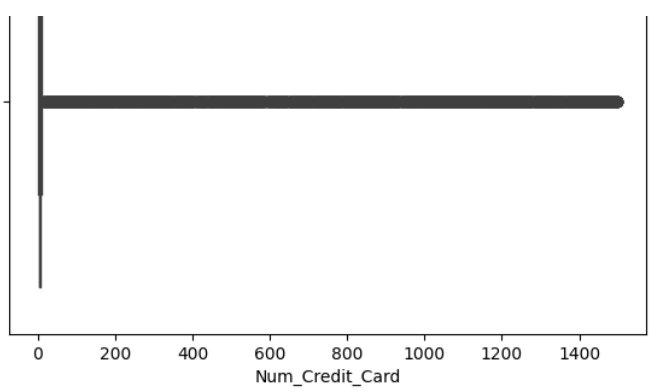


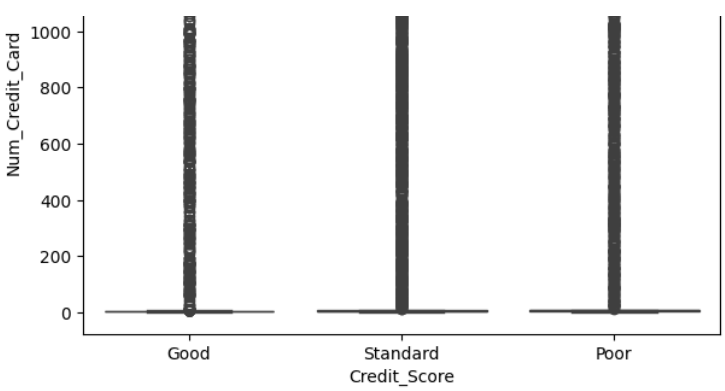


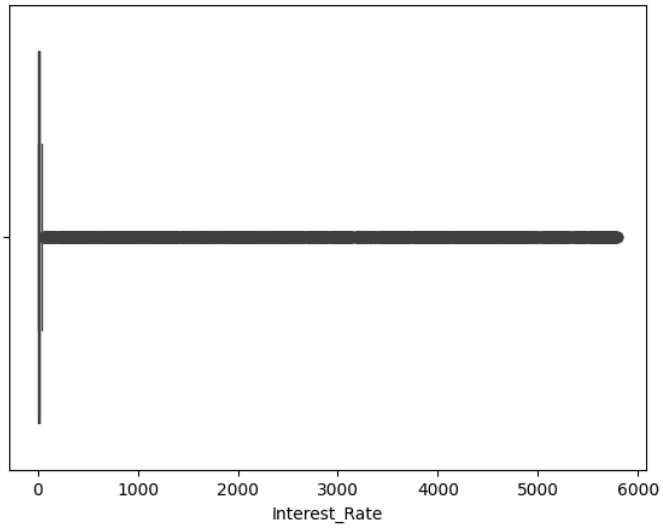


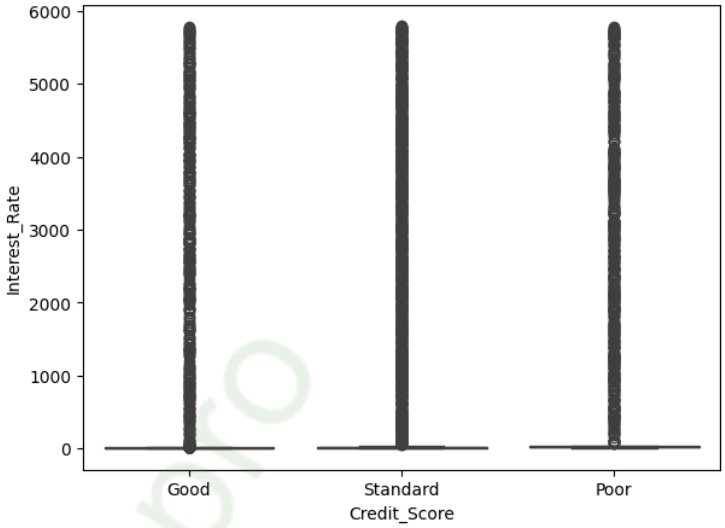


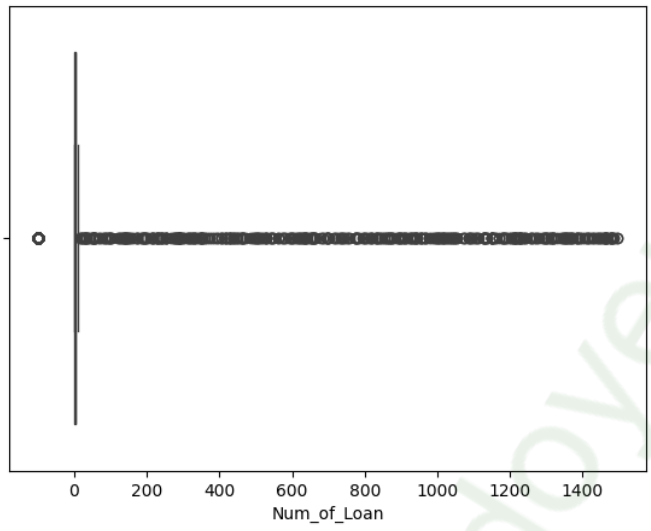


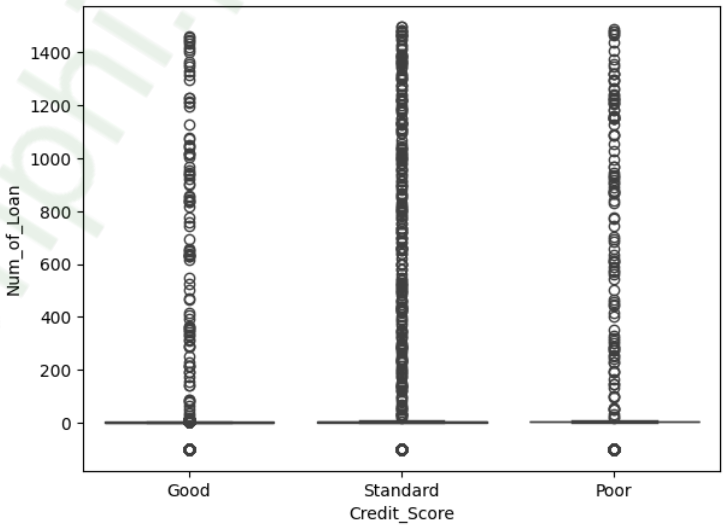


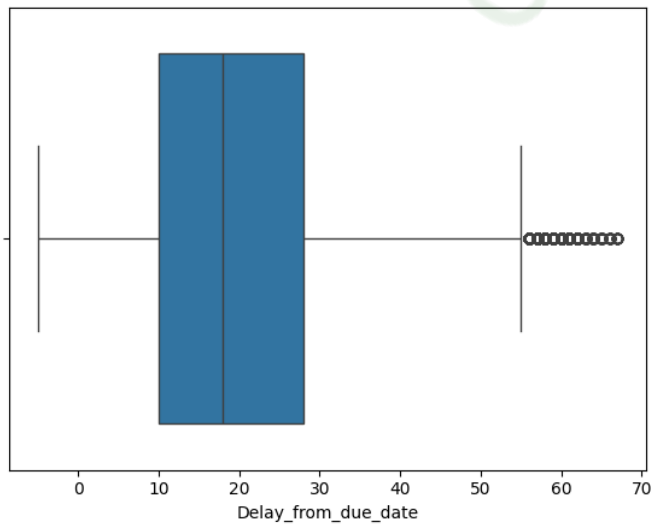


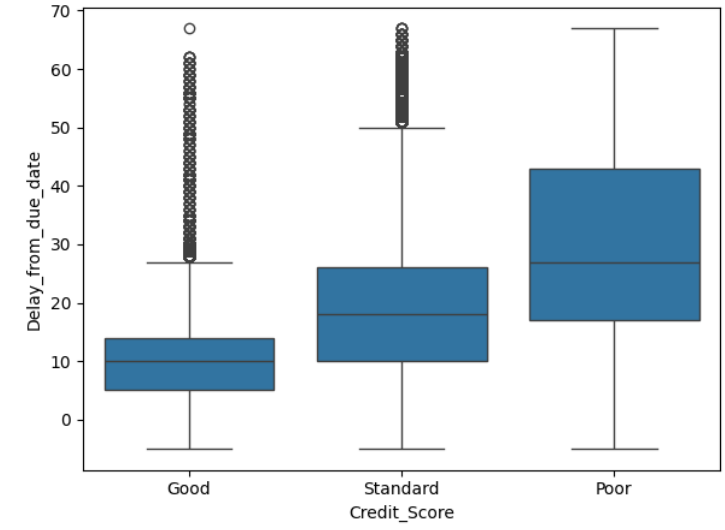


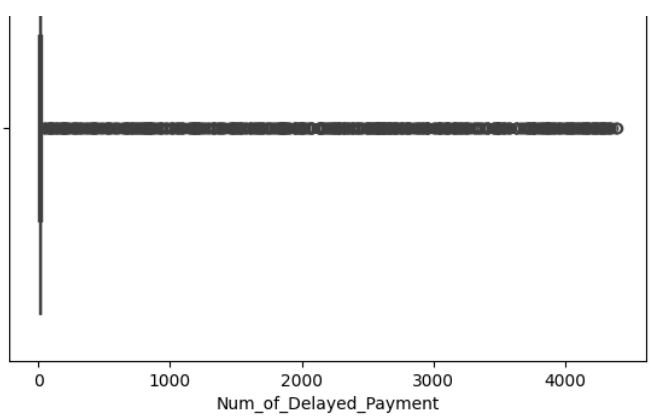


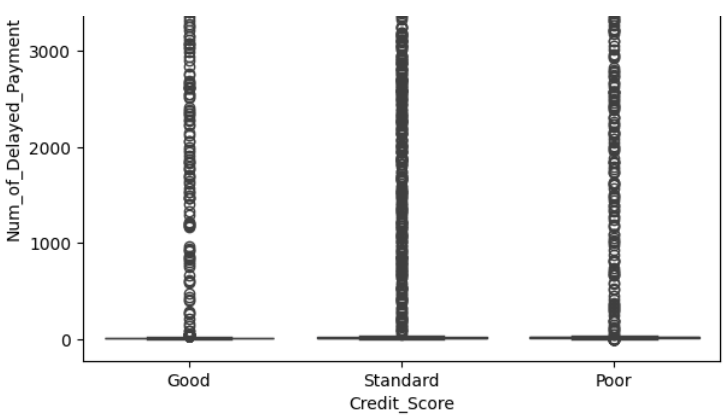


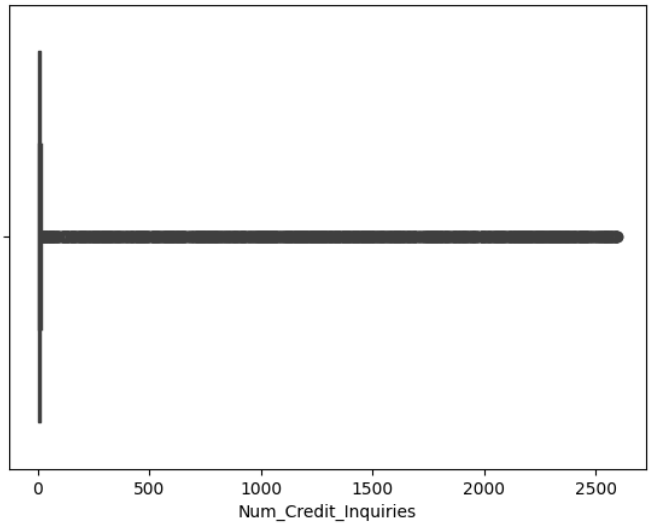


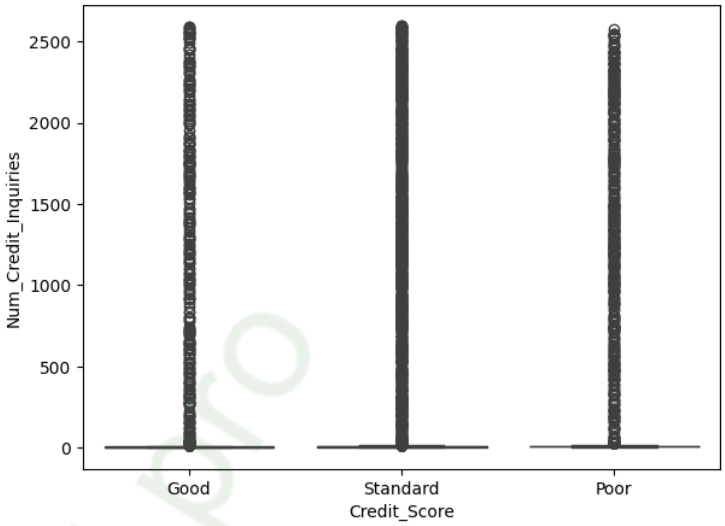


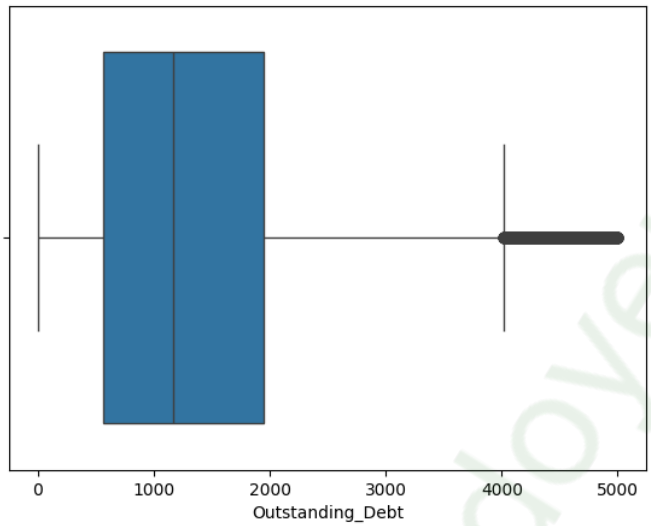


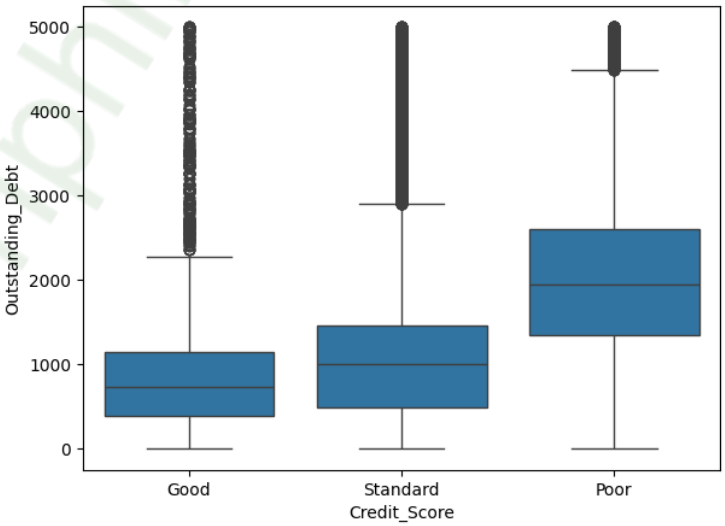


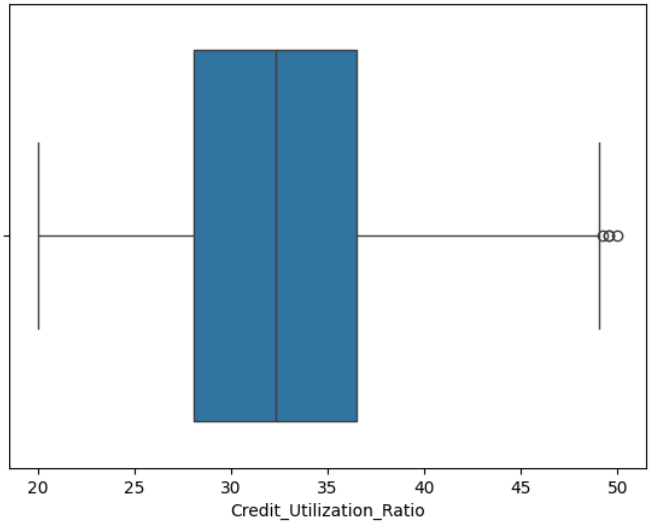


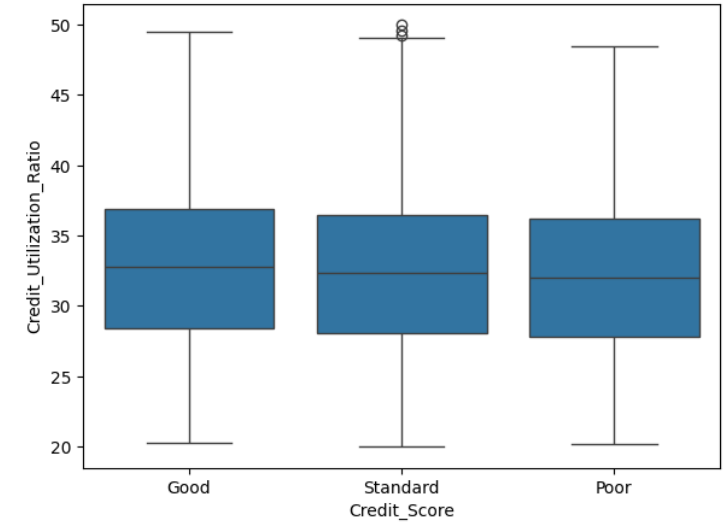


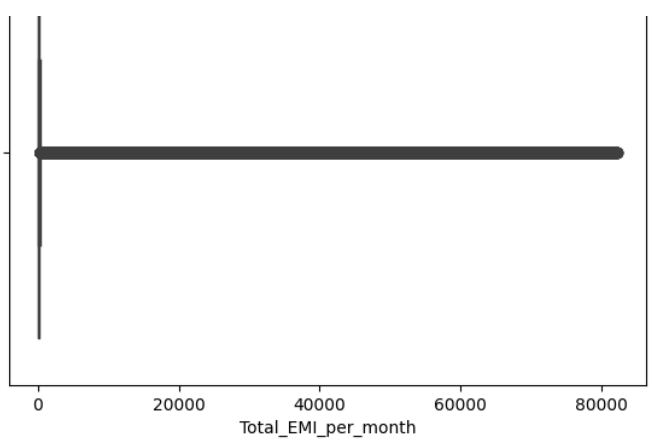


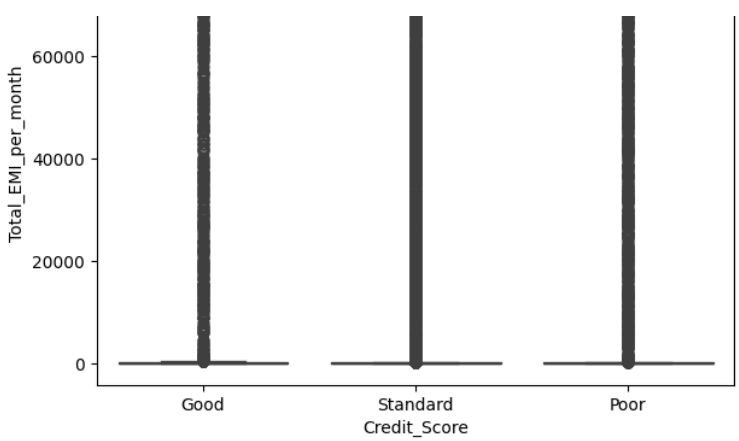


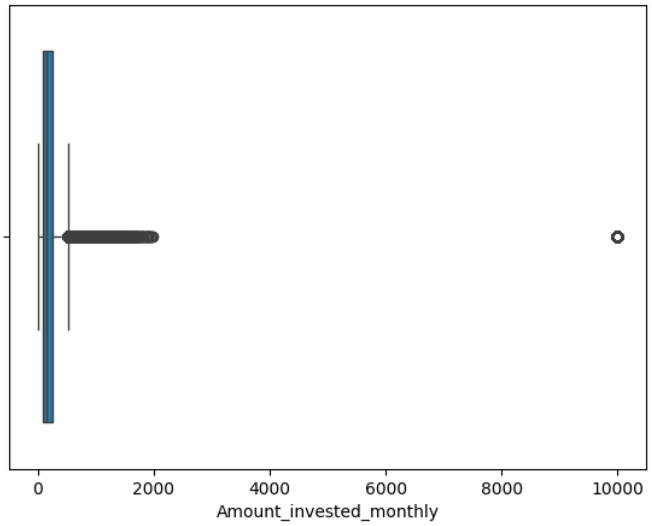


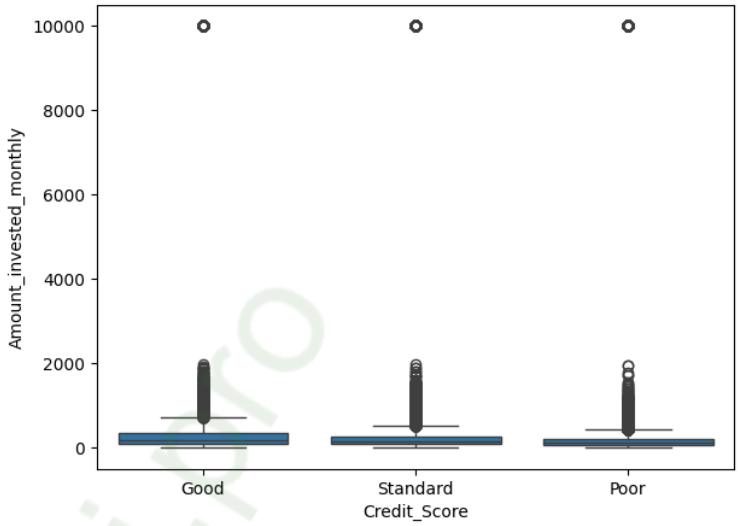


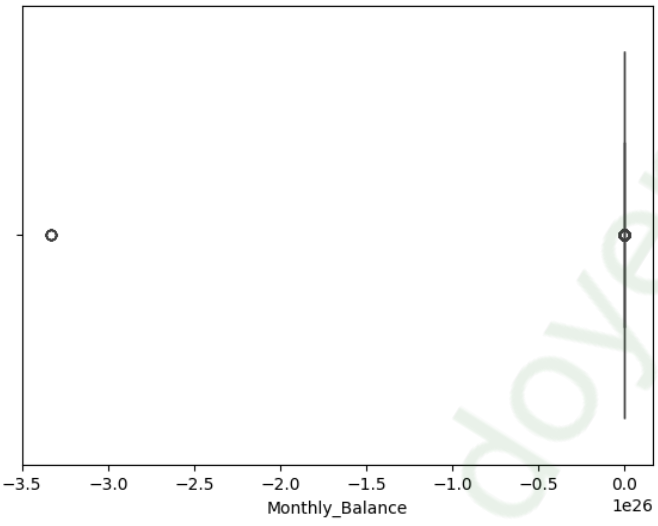


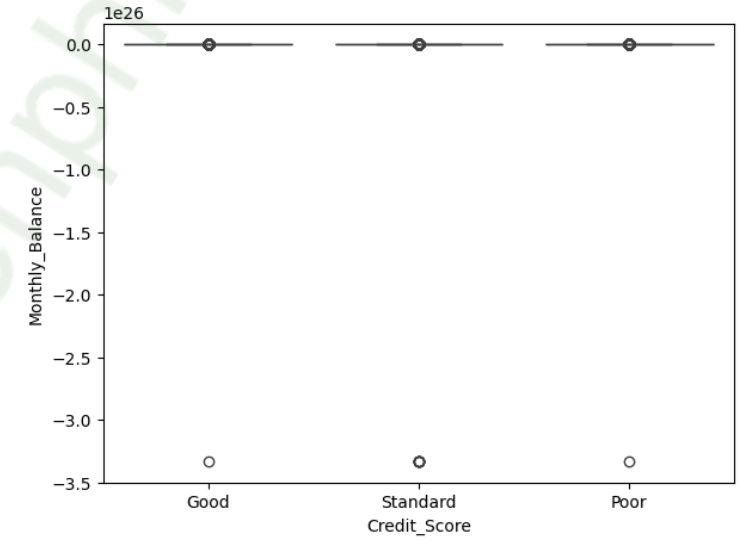








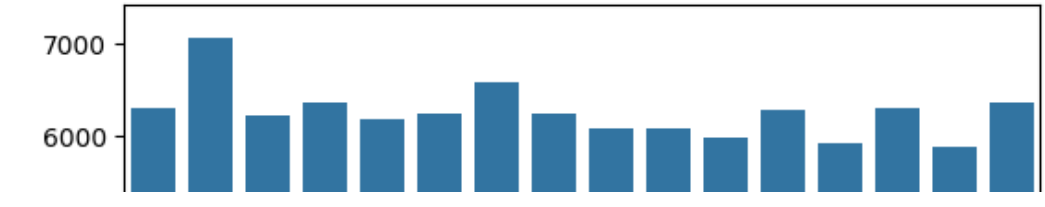


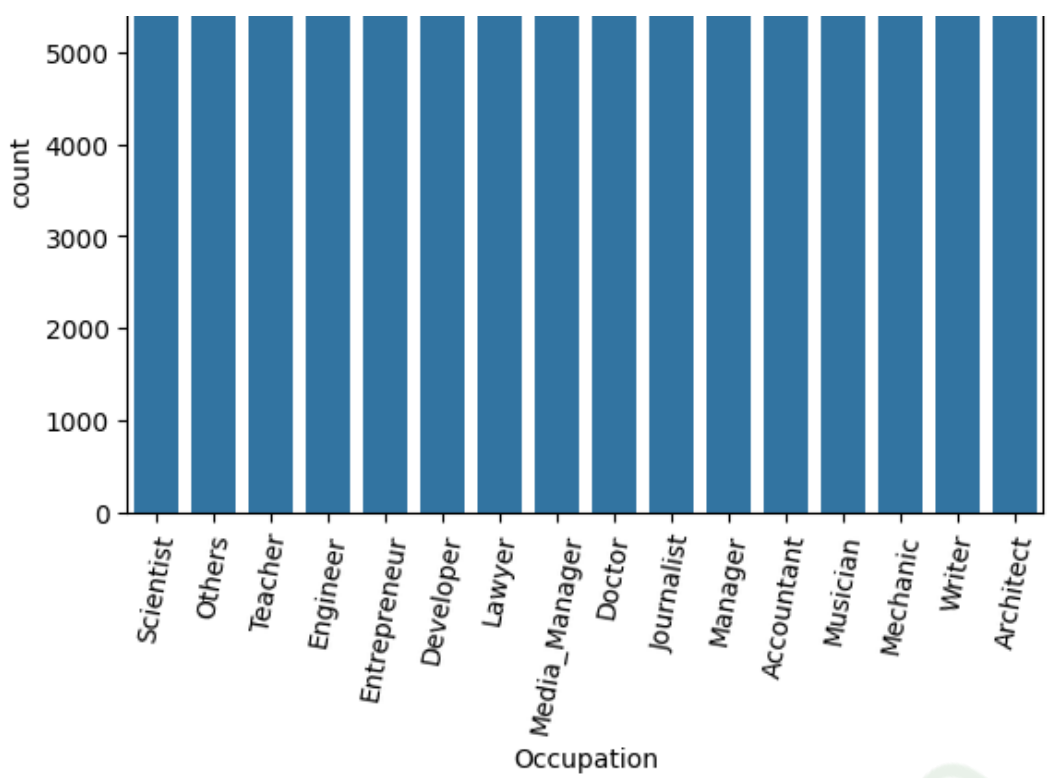


Create a countplot using seaborn to visualize the distribution of 'Occupation'

In [ ]:

```
# Create a count plot for the 'Occupation' column
sns.countplot(x=df["Occupation"])

# Rotate the x-axis labels for better readability
plt.xticks(rotation=80)

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



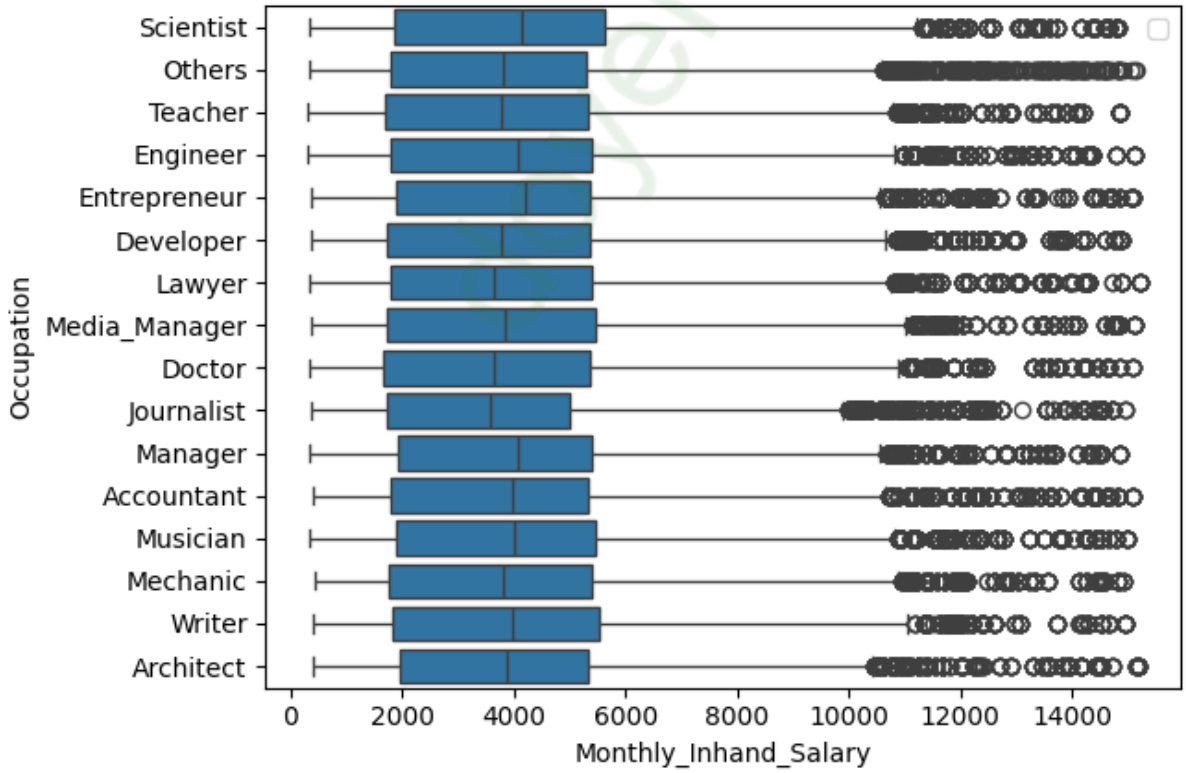


Create a boxplot using seaborn to visualize the distribution of 'Monthly_Inhand_Salary' based on 'Occupation'

In [ ]:

```
# Create a box plot for 'Monthly_Inhand_Salary' grouped by 'Occupation'
sns.boxplot(data=df, y="Occupation", x="Monthly_Inhand_Salary")
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artis
ts whose label start with an underscore are ignored when legend() is called with no argum
ent.
```



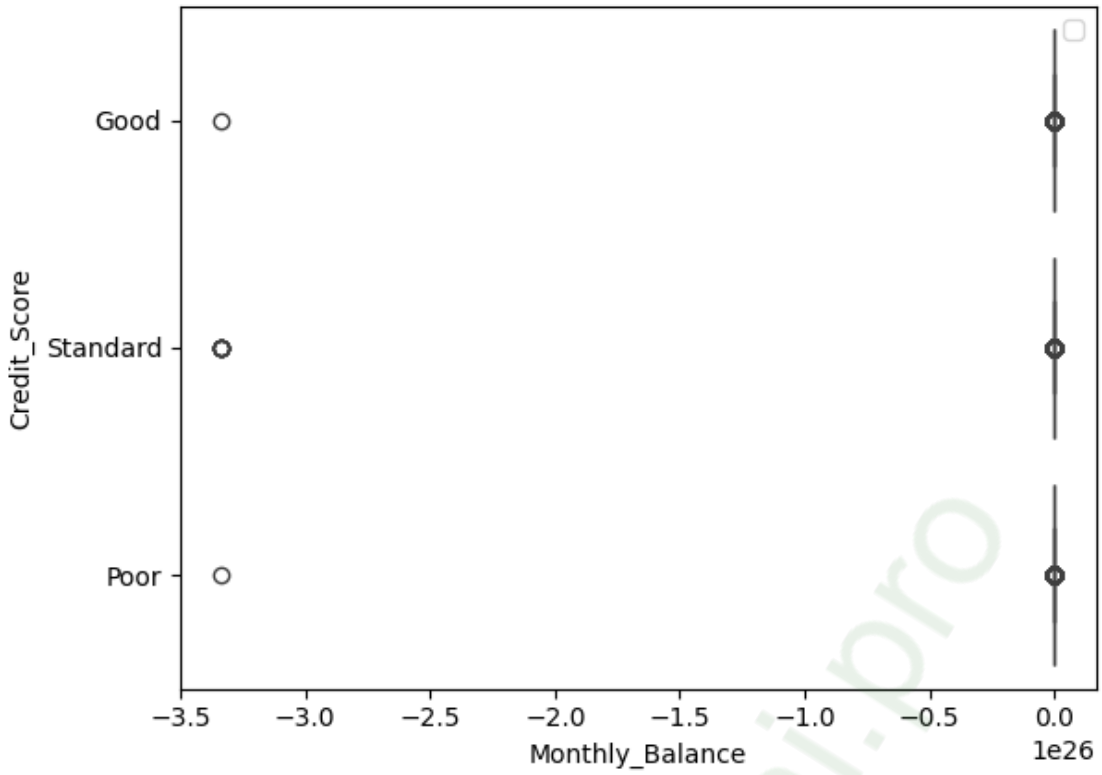
Create a boxplot using seaborn to visualize the distribution of 'Monthly_Balance' based on 'Credit_Score'

In [ ]:

```
# Create a box plot for 'Credit_Score' grouped by 'Monthly_Balance'
```

```
sns.boxplot(data=df, y="Credit_Score", x="Monthly_Balance")
plt.legend()
plt.show()
```

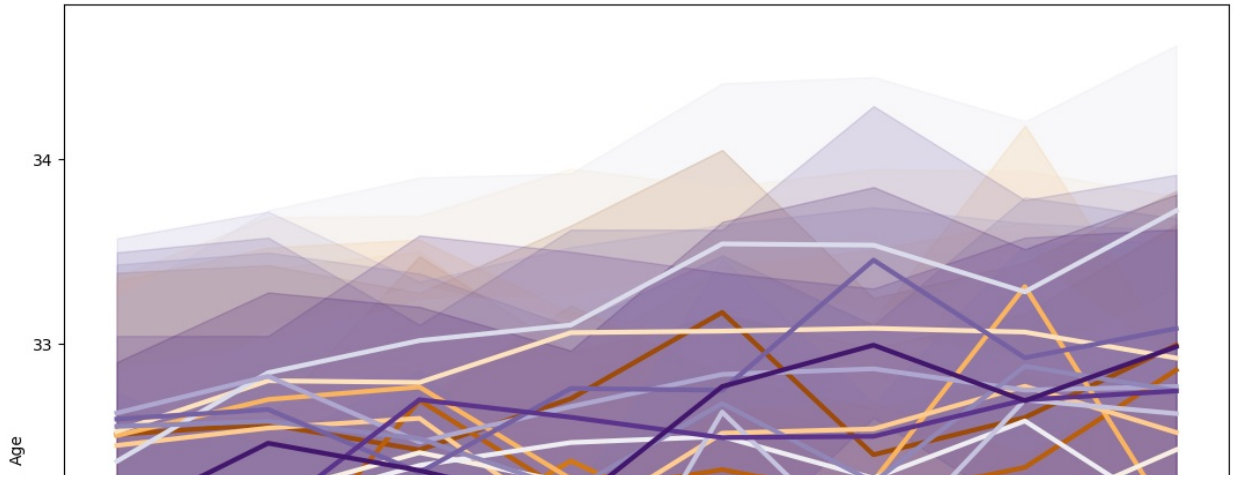
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artis
ts whose label start with an underscore are ignored when legend() is called with no argum
ent.
```

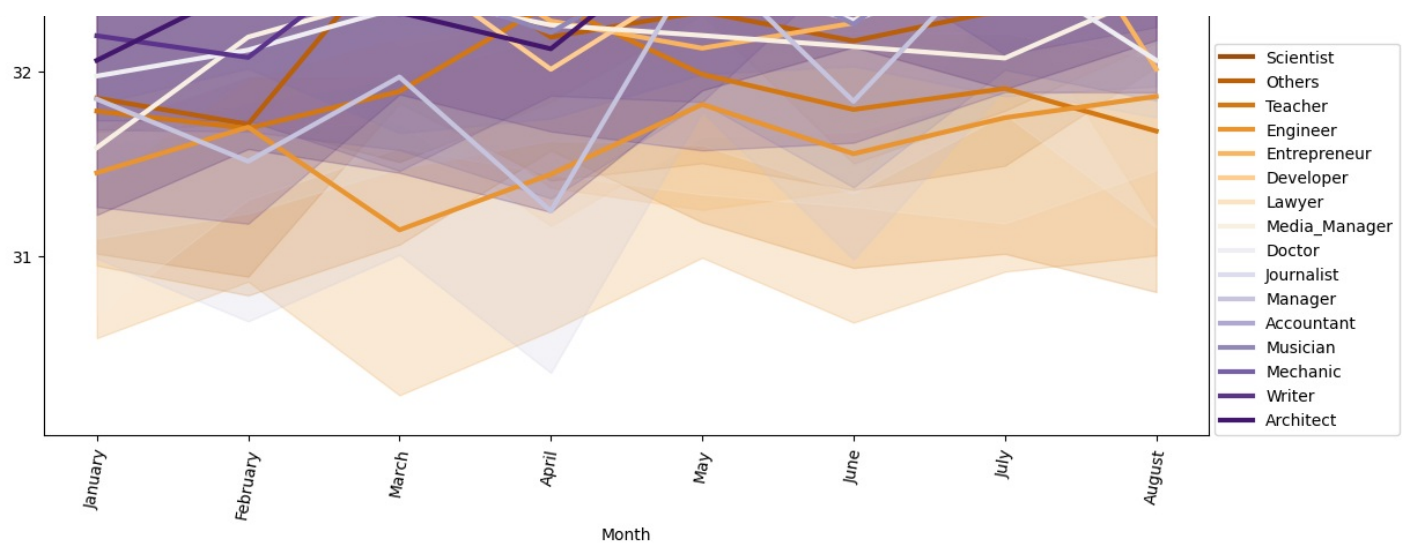


Generates a line plot to visualize the relationship between the 'Month' variable and 'Age' variable grouped by 'Occupation'

In [ ]:

```
# Set the size of the figure
plt.figure(figsize=(13,10))

# Create a line plot to visualize the relationship between 'Month' and 'Age' grouped by '
Occupation'
sns.lineplot(data=df, x='Month', y='Age', hue="Occupation", palette="PuOr", linewidth=3)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Adjust legend position
plt.legend(loc="right", bbox_to_anchor=(0.67, 0., 0.5, 0.442))

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

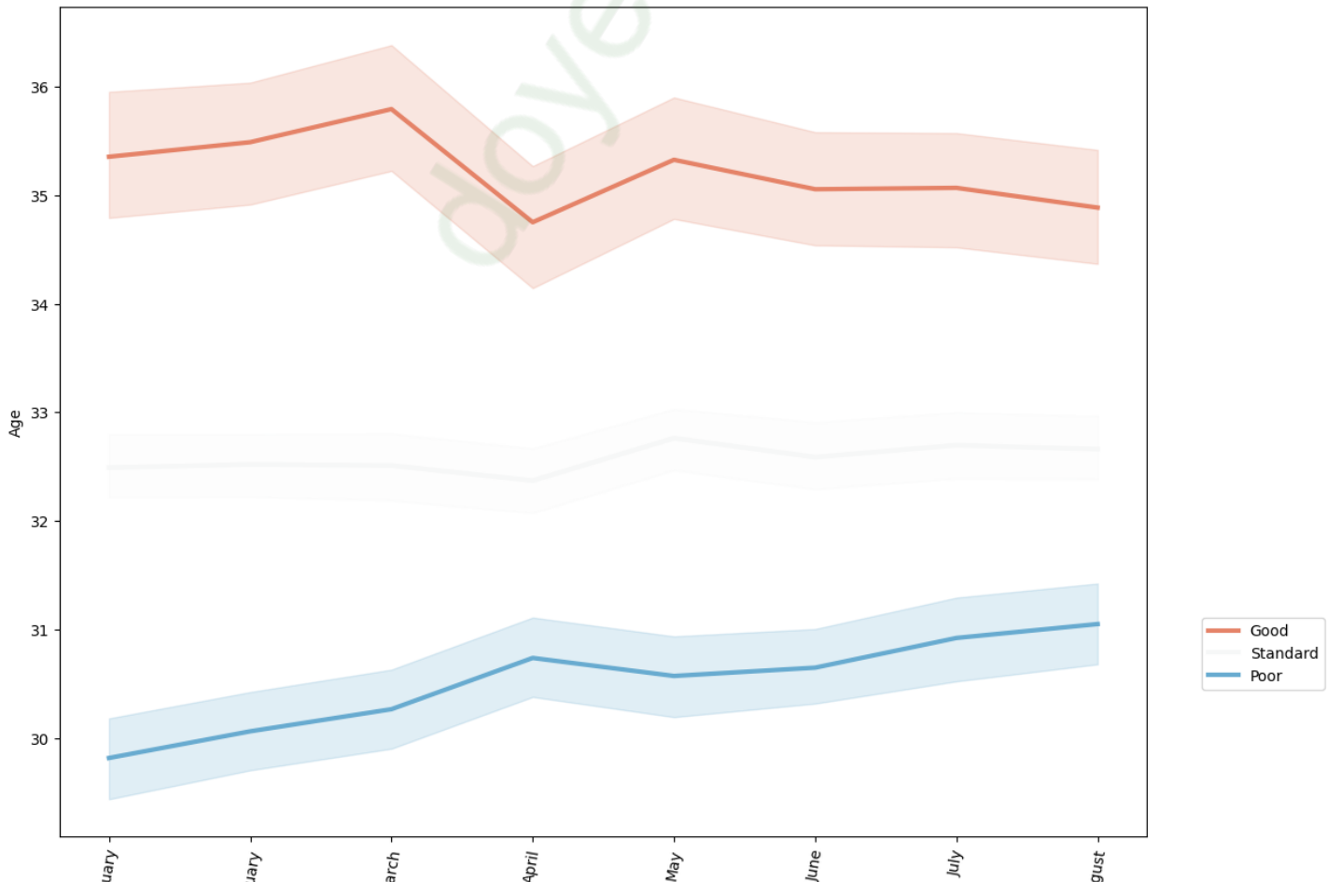




Create a line plot using seaborn to visualize the relationship between 'Age' and 'Month' with respect to 'Credit_Score'

In [ ]:

```
# Set the size of the figure
plt.figure(figsize=(13,10))

# Create a line plot to visualize the relationship between 'Month' and 'Age' grouped by 'Credit_Score'
sns.lineplot(data=df, x='Month', y='Age', hue="Credit_Score", palette="RdBu", linewidth=3)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Adjust legend position
plt.legend(loc="right", bbox_to_anchor=(0.67, 0., 0.5, 0.442))

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

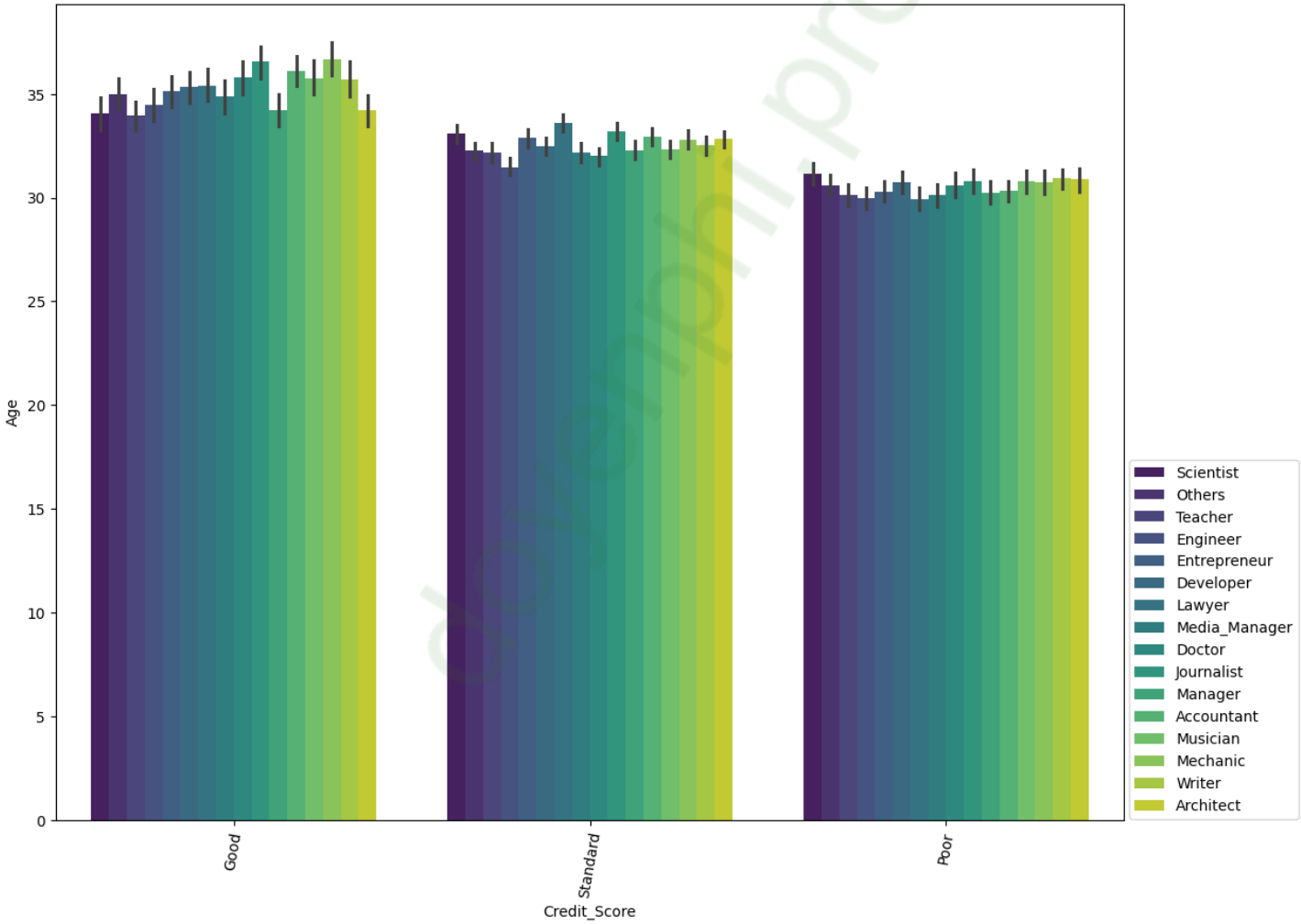


Month

Create a bar plot using seaborn to visualize the relationship between 'Age' and 'Credit_Score' with respect to 'Occupation'

In [ ]:

```
# Set the size of the figure
plt.figure(figsize=(13,10))

# Create a bar plot to visualize the relationship between 'Credit_Score' and 'Age' groupe
d by 'Occupation'
sns.barplot(data=df, x='Credit_Score', y='Age', hue="Occupation", palette="viridis", lin
ewidth=3)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Adjust legend position
plt.legend(loc="right", bbox_to_anchor=(0.67, 0., 0.5, 0.442))

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



Create a heatmap using seaborn to visualize the correlation matrix of numeric columns

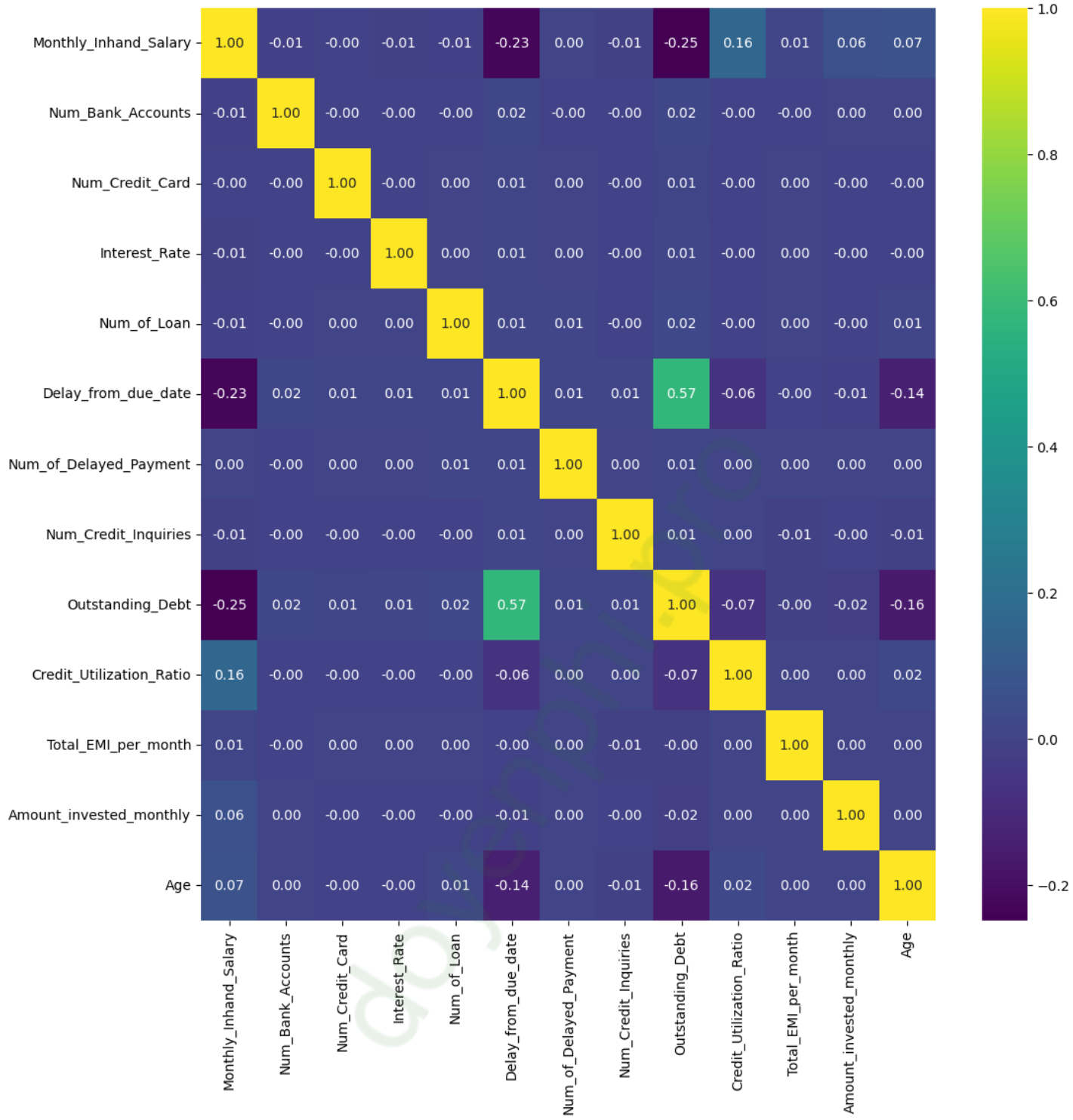
In [ ]:

```
# Set the size of the figure
plt.figure(figsize=(12, 12))

# Visualisation using heatmap to see correlations among different numerical variables
sns.heatmap(df[num_cols].corr(), annot=True, fmt='.2f', cmap="viridis")

# Show the plot
```

```
plt.show()
```

Out[ ]:

```
<Axes: >
```



MODEL TRAINING

In the Model Training phase of this project, machine learning algorithms are applied to the preprocessed data to build predictive models. These models learn from the patterns and relationships present in the data to make predictions on new, unseen data. The goal is to train models that accurately predict the target variable, such as credit score or loan approval status, based on the available features.

One-hot encode the 'Occupation' column and drop the first column to avoid multicollinearity

In [ ]:

```
# Converting categorical values to numerical
df['Credit_Score'] = df['Credit_Score'].map({'Good':0,'Standard':1,'Poor':2})
```

```
df['Month'] = df['Month'].map({'January':1, 'February':2, 'March':3, 'April':4, 'May':5,
'June':6, 'July':7,'August':8})
df['Payment_of_Min_Amount'] = df['Payment_of_Min_Amount'].map({'No':0, 'NM':0, 'Yes':1})
df['Occupation'] = df['Occupation'].map({'Scientist':1, 'Others':2, 'Teacher':3, 'Engine
er':4, 'Entrepreneur':5,'Developer':12, 'Lawyer':13, 'Media_Manager':14, 'Doctor':15, 'J
ournalist':16,
       'Manager':6, 'Accountant':7, 'Musician':8, 'Mechanic':9, 'Writer':10,'Architect':
11})
```

In [ ]:

```
#checking the unique values of our target credit score
df['Credit_Score'].unique()
```

Out[ ]:

```
array([0, 1, 2])
```

In [ ]:

```
#checking whether all the columns are converted to numerical values
df.dtypes
```

Out[ ]:

```
Month                       int64
Age                       float64
Occupation                  int64
Monthly_Inhand_Salary     float64
Num_Bank_Accounts           int64
Num_Credit_Card             int64
Interest_Rate               int64
Num_of_Loan               float64
Delay_from_due_date         int64
Num_of_Delayed_Payment    float64
Num_Credit_Inquiries      float64
Outstanding_Debt          float64
Credit_Utilization_Ratio  float64
Payment_of_Min_Amount       int64
Total_EMI_per_month       float64
Amount_invested_monthly   float64
Monthly_Balance           float64
Credit_Score                int64
dtype: object
```

Separate the target variable 'Credit_Score' and the features in the DataFrame 'df'

In [ ]:

```
#defining X and Y for our Model training
y = df['Credit_Score']
X = df.drop(['Credit_Score','Num_of_Delayed_Payment'],axis=1)
```

Count the occurrences of each unique value in the 'Credit_Score' column

In [ ]:

```
#checking how many unique values are in the Y variable
[sum(y==item) for item in np.unique(y)]
```

Out[ ]:

```
[17828, 53174, 28998]
```

Import the train_test_split function from the sklearn.model_selection module

Split the dataset into training and testing sets with 80% for training and 20% for testing. The 'x' variable contains the features, and the 'y' variable contains the target variable. The random_state parameter ensures reproducibility of the split

In [ ]:

```
#defining the train and the test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.20,random_state=42)
```

Further split the testing set into validation and testing sets with 50% for each. The 'X_test' and 'y_test' variables are used for this split. The random_state parameter ensures reproducibility of the split

In [ ]:

```
#Defining the training and testing data sets
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, train_size=0.50, random_
state=42)
```

Install the CatBoost library

Import necessary libraries for different classifiers, preprocessing, and evaluation metrics

In [ ]:

```
#importing libraries for model training
!pip install catboost
!pip install scikit-learn
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix,accuracy_score,mean_absolute_error,mean_squa
red_error,r2_score,log_loss,f1_score,jaccard_score
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.10/dist-packages (1.2.3
)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from
catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (fro
m catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (
from catboost) (1.25.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (f
rom catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from cat
boost) (1.11.4)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from ca
tboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catbo
ost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-p
ackages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (f
rom pandas>=0.24->catboost) (2023.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-package
s (from matplotlib->catboost) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (f
rom matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->catboost) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packag
es (from matplotlib->catboost) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages
(from matplotlib->catboost) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (
from matplotlib->catboost) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-package
```

```
s (from matplotlib->catboost) (3.1.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages
(from plotly->catboost) (8.2.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1
.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (
from scikit-learn) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (f
rom scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (
from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn) (3.3.0)
```

Initialize a RandomForestClassifier and fit it to the training data

Calculate the accuracy score on the testing data

In [ ]:

```
#checking the data type in the Y_train data
print(type(y_train))
```

```
<class 'pandas.core.series.Series'>
```

In [ ]:

```
#checking the first 10 values in the Y_train data
print(y_train[:10])
```

```
75220    1
48955    0
44966    0
13568    1
92727    0
51349    1
86979    1
3806     1
91822    0
6006     1
Name: Credit_Score, dtype: int64
```

In [ ]:

```
#checking the shape of the y_train
print(y_train.shape)
```

```
(80000,)
```

In [ ]:

```
#checking the columns of the X_train
print(X_train.shape[1])
```

```
16
```

In [ ]:

```
#checking thcolumns and the rows of the X_train
print(X_train.shape)
```

```
(80000, 16)
```

In [ ]:

```
#Checking the number of samples in X and y train
print("Number of samples in X_train:", len(X_train))
print("Number of samples in y_train:", len(y_train))
```

```
Number of samples in X_train: 80000
Number of samples in y_train: 80000
```

In [ ]:

```
#Model training and testing using Random forest and determining the accuracy level
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf.score(X_test,y_test)
```

Out[ ]:

0.7897

Initialize an XGBClassifier and fit it to the training data

Calculate the accuracy score on the testing data

In [ ]:

```
#Model training and testing using XGBClassifier and determining the accuracy level
xgb = XGBClassifier()
xgbmodel = xgb.fit(X_train,y_train)
xgbmodel.score(X_test,y_test)
```

Out[ ]:

0.7355

Initialize a GradientBoostingClassifier and fit it to the training data

Calculate the accuracy score on the testing data

In [ ]:

```
#Model training and testing using Gradient Boost and determining the accuracy level
gbc = GradientBoostingClassifier()
gbcmodel = gbc.fit(X_train,y_train)
xgbmodel.score(X_test,y_test)
```

Out[ ]:

0.7355

In [ ]:

```
#Model training and testing using catBoostClassifier and determining the accuracy level
cbc = CatBoostClassifier()
cbcmodel = cbc.fit(X_train,y_train, verbose=False)
cbcmodel.score(X_test,y_test)
```

Out[ ]:

0.7258

Create a pipeline using MinMaxScaler for preprocessing and RandomForestClassifier for classification

Fit the pipeline to the training data

Calculate the accuracy score on the testing data

In [ ]:

```
#pipeline for RandomForestClassifier
pipe = make_pipeline(MinMaxScaler(), RandomForestClassifier())
pipe.fit(X_train, y_train)
Pipeline(steps=[('MinMaxScaler', MinMaxScaler()),('RandomForestClassifier', RandomForestC
lassifier())])
pipe.score(X_test, y_test)
```

Out[ ]:

0.7956

Make predictions using the trained RandomForestClassifier on the testing data

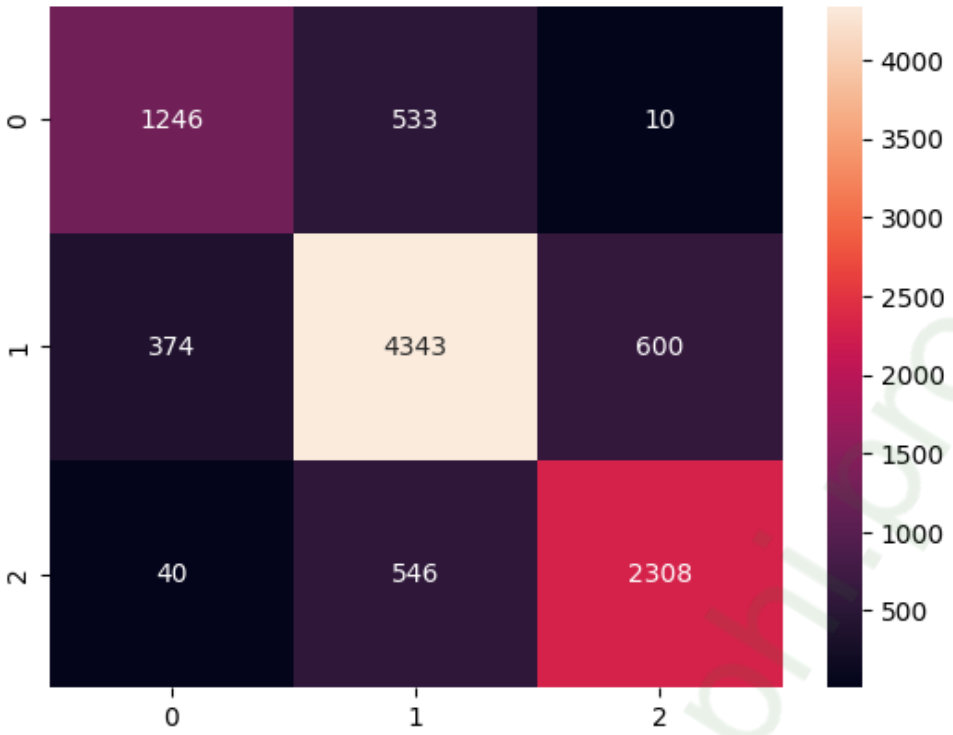
Visualize the confusion matrix using seaborn's heatmap

In [ ]:

```
#Getting the heatmap of the confusion matrix
y_predrfc = rf.predict(X_test)
sns.heatmap(confusion_matrix(y_test,y_predrfc), annot=True, fmt='.0f')
```

Out[ ]:

```
<Axes: >
```



Make predictions using the trained RandomForestClassifier on the testing data

In [ ]:

```
# predicting the X_test data sample
rf.predict(X_test)
```

Out[ ]:

```
array([1, 1, 2, ..., 1, 0, 2])
```

Convert the training and validation data to NumPy arrays and cast them to float32 data type

In [ ]:

```
#conversion of training and validation data to Numpy array
X=np.asarray(X_train).astype(np.float32)
Y=np.asarray(y_train).astype(np.float32)
X_val=np.asarray(X_val).astype(np.float32)
Y_val=np.asarray(y_val).astype(np.float32)
```

Import TensorFlow and classification_report from sklearn.metrics

Display the shape of the input data X

In [ ]:

```
#checking the columns and rows of the X
X.shape
```

Out[ ]:

```
(80000, 16)
```

Evaluate the model on the training data X and Y

MODEL DEPLOYMENT

In the Model Deployment phase of this project, the trained machine learning models will be deployed into a production environment to facilitate real-time predictions. This involves integrating the models into existing systems or applications, enabling them to make automated decisions based on new data inputs. The deployment process aims to leverage the predictive capabilities of the models to enhance decision-making processes and deliver value in real-world scenarios.

Set up a FastAPI instance to create an API for scoring clients based on the trained model, including loading the model from a pickle file, defining API routes for scoring, and customizing the OpenAPI documentation. Finally, make a request to the API for scoring a client.

In [ ]:

```
# Save the trained model to a pickle file
model=rf

import pickle
# save model for creating our dashboard app
with open("model.pkl", "wb") as f:
    pickle.dump(model, f)
```

In [ ]:

```
# Install required packages for the API
!pip install fastapi
!pip install typing_extensions --upgrade
!pip install requests
!pip show requests

# Import necessary libraries
from typing_extensions import Doc
import pandas as pd
import pickle
import requests
import json
from fastapi import FastAPI
from fastapi.openapi.utils import get_openapi

# Load the trained model
with open("model.pkl", "rb") as f:
      model = pickle.load(f)

# Create a FastAPI instance
app = FastAPI()

# Define a function to customize the OpenAPI documentation
def custom_openapi():
      if app.openapi_schema:
          return app.openapi_schema
      openapi_schema = get_openapi(
          title="Credit Default Prediction ",
          version="0.0.1",
          description="Classification with Logistic Regression. To try it out, click 'Try
it out' under '/score' ",
          routes=app.routes,
      )
      app.openapi_schema = openapi_schema
      return app.openapi_schema

app.openapi = custom_openapi
```

```
# Define a route for the root endpoint
@app.get("/")
async def root():
      return {"message": "Hello World"}

# Define a route for scoring clients
@app.post('/score')
async def give_score():
      data_dict = data.dict()
      df = pd.DataFrame([data_dict])
      score = model.predict_proba(df)[0][1]
      score = round(score, 2)
      if score < 0.4933:
          status = 'Accepted'
      else:
          status = 'Rejected'

      return {"risk_score": score,
              'application_status': status}

  # Make a request to the API
response = requests.post("http://localhost:8080/score", json={"client_id": 1, "age": 30,
"income": 50000, "loan_amount": 10000})


  # Print the response
print(response)
```

```
Requirement already satisfied: fastapi in /usr/local/lib/python3.10/dist-packages (0.110.
0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7
.4 in /usr/local/lib/python3.10/dist-packages (from fastapi) (2.6.1)
Requirement already satisfied: starlette<0.37.0,>=0.36.3 in /usr/local/lib/python3.10/dis
t-packages (from fastapi) (0.36.3)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist
-packages (from fastapi) (4.9.0)
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-p
ackages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7.4->fastapi) (0.6
.0)
Requirement already satisfied: pydantic-core==2.16.2 in /usr/local/lib/python3.10/dist-pa
ckages (from pydantic!=1.8,!=1.8.1,!=2.0.0,!=2.0.1,!=2.1.0,<3.0.0,>=1.7.4->fastapi) (2.16
.2)
Requirement already satisfied: anyio<5,>=3.4.0 in /usr/local/lib/python3.10/dist-packages
(from starlette<0.37.0,>=0.36.3->fastapi) (3.7.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from
anyio<5,>=3.4.0->starlette<0.37.0,>=0.36.3->fastapi) (3.6)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (f
rom anyio<5,>=3.4.0->starlette<0.37.0,>=0.36.3->fastapi) (1.3.0)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages
(from anyio<5,>=3.4.0->starlette<0.37.0,>=0.36.3->fastapi) (1.2.0)
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.10/dist-packag
es (4.9.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (2.31.
0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist
-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packa
ges (from requests) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests) (2024.2.2)
Name: requests
Version: 2.31.0
Summary: Python HTTP for Humans.
Home-page: https://requests.readthedocs.io
Author: Kenneth Reitz
Author-email: me@kennethreitz.org
License: Apache 2.0
Location: /usr/local/lib/python3.10/dist-packages
Requires: certifi, charset-normalizer, idna, urllib3
```

```
Required-by: bigframes, CacheControl, community, dash, earthengine-api, fastai, folium, g
csfs, gdown, geocoder, google-api-core, google-cloud-bigquery, google-cloud-storage, goog
le-colab, gspread, huggingface-hub, kaggle, kagglehub, moviepy, music21, pandas-datareade
r, panel, pins, pooch, pymystem3, requests-oauthlib, spacy, Sphinx, streamlit, tensorboar
d, tensorflow-datasets, torchdata, torchtext, torchvision, transformers, tweepy, weasel,
yfinance
<Response [404]>
```

In [ ]:

```
# Restart the kernel
!jupyter kernelspec list
!jupyter notebook stop
!jupyter notebook start
!pip install dash
!pip install dash_bootstrap_components

# Import necessary libraries and modules
from dash import Dash, dcc, html, Input, Output
import dash_bootstrap_components as dbc
import pandas as pd
import numpy as np
from pathlib import Path
import plotly.graph_objs as go
import requests
import inspect

# Get the current file path
current_file_path = inspect.getfile(inspect.currentframe())

# Define the base directory of the current file
BASE_DIR = Path('/content/train.csv').parent

# Instantiate the dash app
app = Dash(external_stylesheets=[dbc.themes.LUX])
server = app.server
app.title = "Credit Scoring"

# Load dataset and saved shap values from pickle
df = pd.read_csv("/content/train.csv")

# Define options for a dropdown box that select a client id.
client_options = [{'label': str(x), 'value': x} for x in df['Customer_ID'].unique()]

# Drop down box for client selection
select_client = html.Div([
    dcc.Dropdown(
        id="client_id",
        options=client_options,
        multi=False,
        value=100001
    )
])

# Define client profile in detail
client_profile = html.Div([
    dbc.Row([
        html.Span("Num Credit Inquiries: ", style={"font-weight": "bold"}),
        html.Div(id='Num_Credit_Inquiries')  # Define a div with ID 'Num_Credit_Inquirie
s'
    ]),
    dbc.Row([
        html.Span("Credit Utilization Ratio: ", style={"font-weight": "bold"}),
        html.Div(id='Credit_Utilization_Ratio')  # Define a div with ID 'Credit_Utilizat
ion_Ratio'
    ]),
    dbc.Row([
        html.Span("Credit History Age: ", style={"font-weight": "bold"}),
        html.Div(id='Credit_History_Age')  # Define a div with ID 'Credit_History_Age'
    ]),
    dbc.Row([
        html.Span("Payment of Min Amount: ", style={"font-weight": "bold"}),
```

```
        html.Div(id='Payment_of_Min_Amount')  # Define a div with ID 'Payment_of_Min_Amo
unt'
    ]),
    dbc.Row([
        html.Span("Credit Score: ", style={"font-weight": "bold"}),
        html.Div(id='Credit_Score')  # Define a div with ID 'Credit_Score'
    ]),
])

# Define dashboard layout
app.layout = dbc.Container([
    html.H1("Credit Scoring", className='text-center mb-2'), # Page title
    dbc.Row([
        dbc.Col(dbc.Card([
                dbc.CardHeader("Select Client ID",
                               style={"background-color": "#1A85FF", "color": "white", "
font-weight": "bold"}),
                select_client,
                dbc.CardBody(dcc.Graph(id="gauge-chart"))
            ], className='mb-4', style={"height": "100%"}), width=5), # Card component w
ith a dropdown box and a gauge chart
        dbc.Col(dbc.Card([
                dbc.CardHeader("Client Profile",
                               style={"background-color": "#1A85FF", "color": "white", "
font-weight": "bold"}),
                client_profile
            ], className='mb-4', style={"height": "100%"}), width=5)
    ], className='text-center', align='stretch', justify='center'), # A card components
displaying selected client features
], fluid=True,)

# Define callback to update client features and gauge chart
@app.callback(
    [
        Output("gauge-chart", "figure"),
        Output("Num_Credit_Inquiries", "children"),
        Output("Credit_Utilization_Ratio", "children"),
        Output("Credit_History_Age", "children"),
        Output("Payment_of_Min_Amount", "children"),
        Output("Credit_Score", "children")
    ],
    [Input("client_id", "value")]
)
def update_dashboard(client_id):
    try:
        # Get data for the selected client
        data = df[df['Customer_ID'] == client_id]
        if data.empty:
            raise ValueError(f"No data found for client ID: {client_id}")

        # Make API request to get model prediction
        url = "https://oc-project-7-fastapi.herokuapp.com/score"
        payload = {
            "Num_Credit_Inquiries": data['Num_Credit_Inquiries'].item(),
            "Credit_Utilization_Ratio": data['Credit_Utilization_Ratio'].item(),
            "Credit_History_Age": data['Credit_History_Age'].item(),
            "Payment_of_Min_Amount": data['Payment_of_Min_Amount'].item(),
            "Credit_Score": data['Credit_Score'].item(),
            "EXT_SOURCE_2": data['EXT_SOURCE_2'].item(),
            "EXT_SOURCE_3": data['EXT_SOURCE_3'].item()
        }

        response = requests.post(url, json=payload)
        data_from_api = response.json()
        risk_score = data_from_api.get("risk_score")
        status = data_from_api.get("application_status")

        if risk_score is None or status is None:
            raise ValueError("Invalid response from API")

        # Create gauge chart
        fig_gauge = go.Figure(go.Indicator(
```

```
            domain={'x': [0, 1], 'y': [0, 1]},
            value=np.around(risk_score, 2),
            mode="gauge+number",
            title={'text': f"{status}", 'font': {'size': 24}},
            gauge={'axis': {'range': [0, 1], 'tickwidth': 2, 'tickcolor': "grey"},
                   'bar': {'color': "black"}, 'bordercolor': "black",
                   'steps': [{'range': [0, 0.4933], 'color': "#1A85FF"},
                             {'range': [0.4933, 1], 'color': "#D41159"}],
                   'threshold': {'line': {'color': "black", 'width': 1}, 'thickness': 1,
'value': 0.4933}}))

        # Prepare client profile details
        client_profile_details = [
            data['Num_Credit_Inquiries'].values[0],
            data['Credit_Utilization_Ratio'].values[0],
            data['Credit_History_Age'].values[0],
            int(data['Payment_of_Min_Amount'].values[0] / 365),
            round(data['Credit_Score'].values[0], 2)
        ]

        return fig_gauge, *client_profile_details

    except Exception as e:
        print(f"Error in update_dashboard callback: {e}")
        return {}, "", "", "", "", ""

if __name__ == '_main_':
    app.run_server(debug=True)
```

```
Available kernels:
  ir         /usr/local/share/jupyter/kernels/ir
  python3    /usr/local/share/jupyter/kernels/python3
There is currently no server running on 8888
Ports/sockets currently in use:
  - 9000
[C 17:19:04.982 NotebookApp] No such file or directory: /content/start
Requirement already satisfied: dash in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packag
es (from dash) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (f
rom dash) (3.0.1)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (
from dash) (5.15.0)
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from dash) (2.0.0)
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from dash) (2.0.0)
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-packag
es (from dash) (5.0.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist
-packages (from dash) (4.9.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
dash) (2.31.0)
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (from
dash) (1.3.4)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (f
rom dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fro
m dash) (67.7.2)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packa
ges (from dash) (7.0.1)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (fr
om Flask<3.1,>=1.0.4->dash) (3.1.3)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packag
es (from Flask<3.1,>=1.0.4->dash) (2.1.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (fro
m Flask<3.1,>=1.0.4->dash) (8.1.7)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages
(from plotly>=5.0.0->dash) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from
plotly>=5.0.0->dash) (23.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packag
```

```
es (from Werkzeug<3.1->dash) (2.1.5)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from
importlib-metadata->dash) (3.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist
-packages (from requests->dash) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests->dash) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packa
ges (from requests->dash) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests->dash) (2024.2.2)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (fro
m retrying->dash) (1.16.0)
Requirement already satisfied: dash_bootstrap_components in /usr/local/lib/python3.10/dis
t-packages (1.5.0)
Requirement already satisfied: dash>=2.0.0 in /usr/local/lib/python3.10/dist-packages (fr
om dash_bootstrap_components) (2.15.0)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packag
es (from dash>=2.0.0->dash_bootstrap_components) (2.2.5)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (f
rom dash>=2.0.0->dash_bootstrap_components) (3.0.1)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (
from dash>=2.0.0->dash_bootstrap_components) (5.15.0)
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from dash>=2.0.0->dash_bootstrap_components) (2.0.0)
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from dash>=2.0.0->dash_bootstrap_components) (2.0.0)
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-packag
es (from dash>=2.0.0->dash_bootstrap_components) (5.0.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist
-packages (from dash>=2.0.0->dash_bootstrap_components) (4.9.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from
dash>=2.0.0->dash_bootstrap_components) (2.31.0)
Requirement already satisfied: retrying in /usr/local/lib/python3.10/dist-packages (from
dash>=2.0.0->dash_bootstrap_components) (1.3.4)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (f
rom dash>=2.0.0->dash_bootstrap_components) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fro
m dash>=2.0.0->dash_bootstrap_components) (67.7.2)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packa
ges (from dash>=2.0.0->dash_bootstrap_components) (7.0.1)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-packages (fr
om Flask<3.1,>=1.0.4->dash>=2.0.0->dash_bootstrap_components) (3.1.3)
Requirement already satisfied: itsdangerous>=2.0 in /usr/local/lib/python3.10/dist-packag
es (from Flask<3.1,>=1.0.4->dash>=2.0.0->dash_bootstrap_components) (2.1.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-packages (fro
m Flask<3.1,>=1.0.4->dash>=2.0.0->dash_bootstrap_components) (8.1.7)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages
(from plotly>=5.0.0->dash>=2.0.0->dash_bootstrap_components) (8.2.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from
plotly>=5.0.0->dash>=2.0.0->dash_bootstrap_components) (23.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packag
es (from Werkzeug<3.1->dash>=2.0.0->dash_bootstrap_components) (2.1.5)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-packages (from
importlib-metadata->dash>=2.0.0->dash_bootstrap_components) (3.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist
-packages (from requests->dash>=2.0.0->dash_bootstrap_components) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (f
rom requests->dash>=2.0.0->dash_bootstrap_components) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packa
ges (from requests->dash>=2.0.0->dash_bootstrap_components) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packa
ges (from requests->dash>=2.0.0->dash_bootstrap_components) (2024.2.2)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (fro
m retrying->dash>=2.0.0->dash_bootstrap_components) (1.16.0)
```

```
<ipython-input-234-94c258a02799>:28: DtypeWarning:

Columns (26) have mixed types. Specify dtype option on import or set low_memory=False.
```

Define a FastAPI application for predicting credit default using a trained model. It loads the trained model from a

pickle file, sets up routes for handling HTTP requests, and provides an endpoint ('/score') for scoring client data. The predicted risk score and application status are returned as JSON responses. Finally, it makes a request to the API and prints the response.

Create a Dash web app for credit scoring, enabling users to select a client ID and view their profile details alongside a risk gauge chart based on machine learning model predictions. Update information based on user input using callbacks and integrates API requests for model predictions.

Set up the URL for the deployed model API and constructs a payload containing filtered client features. Send a POST request to the API using the payload and parses the response data to obtain the risk score and application status.

CONCLUSION

In conclusion, this project has effectively tackled the credit scoring task by employing machine learning methodologies and thorough data analysis. Through comprehensive data preprocessing, exploration, and visualization, valuable insights into the determinants of creditworthiness were extracted. The model training phase resulted in the development of robust algorithms capable of accurately assessing credit risk, while their deployment into a production environment enables real-time predictions and automated decision-making. By integrating these models into existing systems, operational efficiency can be enhanced, and risks associated with manual evaluations can be mitigated. This project underscores the transformative potential of machine learning in credit scoring, promising improved outcomes for lenders and borrowers alike through continued refinement and adaptation of these models.