In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt   #use the matplotlib visualisation library
from sklearn.preprocessing import OneHotEncoder #use converting categorical into numberical value
from sklearn.model_selection import train_test_split  #Sprit Train - test data
from sklearn.neighbors import KNeighborsClassifier
import seaborn as sns  # visualisation library
from sklearn.preprocessing import MinMaxScaler  #scale numberical data to range between 0 and 1
import re
```

In [40]:
```
df = pd.read_csv('Credit_Score_Train.csv')    #Load data train / test dataset
test = pd.read_csv('Credit_Score_Test.csv')
```

In [42]:
```
df['Credit_Score'].value_counts()     #check data in credit score column
```

Out[42]:
```
Credit_Score
Standard    26411
Poor        14709
Good         8879
Name: count, dtype: int64
```

In [3]:
```
df.shape ## returns the shape of data - 49999 rows, columns
```

Out[3]:
```
(49999, 28)
```

In [4]:
```
df.columns  #Listing the dataframe columns
```

Out[4]:
```
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')
```

In [5]:
```
#Drop the column which is out of model scope
d_col = ['ID','Customer_ID','Month','Name','SSN','Monthly_Inhand_Salary','Num_Bank_Accounts','Num_Credit_Card',
         'Interest_Rate','Num_of_Loan','Type_of_Loan','Changed_Credit_Limit','Num_Credit_Inquiries','Credit_Mix',
         'Credit_Utilization_Ratio','Amount_invested_monthly']
drop_df = df.drop(d_col , axis=1).copy()
drop_df
```

Out[5]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount
0	23	Scientist	19114.12	3	7	809.98	22 Years and 1 Months	No
1	23	Scientist	19114.12	-1	NaN	809.98	NaN	No
2	-500	Scientist	19114.12	3	7	809.98	22 Years and 3 Months	No
3	23	Scientist	19114.12	5	4	809.98	22 Years and 4 Months	No
4	23	Scientist	19114.12	6	NaN	809.98	22 Years and 5 Months	No
...	...	...	...	...	...	...	...	...
49994	17	Developer	35662.88	19	13	2391.98	18 Years and 9 Months	Yes
49995	17	Developer	35662.88_	19	14	2391.98	18 Years and 10 Months	Yes
49996	17	Developer	35662.88	19	16	2391.98	18 Years and 11 Months	Yes
49997	18	_______	35662.88	15	14	2391.98	19 Years and 0 Months	Yes
49998	18	Developer	35662.88	19	15	2391.98	19 Years and 1 Months	Yes

49999 rows × 12 columns

In [6]:
```
#Explore the NAN value in the dataset
drop_df.isnull().sum()
```

Out[6]:
```
Age                          0
Occupation                   0
Annual_Income                0
Delay_from_due_date          0
Num_of_Delayed_Payment    3470
Outstanding_Debt             0
Credit_History_Age        4549
Payment_of_Min_Amount        0
Total_EMI_per_month          0
Payment_Behaviour            0
Monthly_Balance            631
Credit_Score                 0
dtype: int64
```

In [7]:
```
drop_na = drop_df.dropna().copy()  #create a new DataFrame called drop_na by removing rows with missing (NaN) values from the
```

In [8]:
```
drop_na.head(10)    #used to display the first 10 rows of the DataFrame drop_na
```

Out[8]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	T
0	23	Scientist	19114.12	3	7	809.98	22 Years and 1 Months	No	
2	-500	Scientist	19114.12	3	7	809.98	22 Years and 3 Months	No	
3	23	Scientist	19114.12	5	4	809.98	22 Years and 4 Months	No	
5	23	Scientist	19114.12	8	4	809.98	22 Years and 6 Months	No	
6	23	Scientist	19114.12	3	8_	809.98	22 Years and 7 Months	No	
8	28_	_______	34847.84	3	4	605.03	26 Years and 7 Months	No	
9	28	Teacher	34847.84	7	1	605.03	26 Years and 8 Months	No	
10	28	Teacher	34847.84_	3	-1	605.03	26 Years and 9 Months	No	
11	28	Teacher	34847.84	3	3_	605.03	26 Years and 10 Months	No	
12	28	Teacher	34847.84	3	1	605.03	26 Years and 11 Months	No	

In [9]:
```
for i in drop_na:  # This loop iterates over the columns of the DataFrame drop_na
  print('\n',i,drop_na[i].unique()) #prints the column name with the unique values in  column
```

```
Age ['23' '-500' '28_' '28' '34' '54' '55' '21' '31' '33' '34_' '30' '30_'
 '24' '24_' '44' '45' '40' '32' '33_' '35' '35_' '36' '39' '37' '181' '20'
 '46' '26' '41' '42' '19' '31_' '48' '995' '40_' '37_' '38' '54_' '5079'
 '43' '21_' '22' '16' '7080' '18' '3885' '15' '27' '25' '3052' '14' '5342'
 '17' '18_' '4431' '2657' '2111_' '46_' '47' '1032' '16_' '19_' '456'
 '5717' '53_' '53' '56' '25_' '38_' '27_' '55_' '3169' '1191' '29' '43_'
 '48_' '49' '49_' '6955' '2534' '3115' '7657' '51' '50' '5112' '50_' '32_'
 '6452' '2744' '22_' '1439' '5795' '20_' '4872' '1772' '15_' '1383' '5657'
 '52' '1934' '51_' '8352' '3734' '26_' '2056' '2339' '14_' '8406' '39_'
 '36_' '6953' '5626' '4471' '548' '44_' '769' '5490' '525' '4202' '3665'
 '7670_' '4670' '3616' '6922' '42_' '6619' '1808' '7992' '45_' '223'
 '1232' '4659' '6895' '395' '7099' '6048' '3936' '3512' '123' '5639' '471'
 '7359' '29_' '23_' '4049' '5053' '2109' '7183' '5604' '1206' '6835'
 '4067' '41_' '1170' '3625' '6354' '3724' '5610' '4710' '47_' '52_' '3937'
 '3542' '2239' '17_' '5645' '7425' '7851' '2027' '6306' '835' '3513'
 '6846' '6868' '7805' '7274' '831' '8394' '2751' '733_' '783' '2455'
 '4119' '4645' '8105' '1400' '7431' '3666' '2350' '1116' '692' '5429'
 '4745' '4017' '651' '5769_' '6586' '7699' '919' '6765' '2546' '3909'
 '8655' '4383_' '2824' '56_' '7865' '2823' '5688' '6178' '886' '3353'
 '8562' '7750' '5186' '8080' '3553' '3967' '6520' '2650_' '1335' '7068'
 '6100' '2636' '1753' '6389' '2419' '2961' '3307_' '5504' '4060' '5237'
 '5583' '4958' '6395' '3319' '7705' '5194' '5154' '4155' '7336' '7195'
 '8005' '5165' '3861' '2584' '5592' '3771' '1188' '5017' '8034' '8173'
 '6962_' '3578' '4572' '5589_' '1678' '1512' '6556_' '5430' '8442' '4517'
 '7525' '1203' '7980' '2400' '6331' '3715' '3285' '8043' '8250' '8628'
 '8608' '7016' '989' '359' '2090' '4126' '3791' '5782' '6292' '8662'
 '1681' '1338' '8000' '5608' '1006' '1451' '622' '4857' '8216' '7330'
 '7289' '3601' '4127' '236' '6707' '493' '4682' '287' '375' '1969' '5788'
 '6413' '2388' '5152' '1715' '8200' '4336' '8299' '1308' '1051' '134'
 '5897' '6350' '2379' '3197' '3341' '7123' '556' '1447_' '5579' '1692'
 '4204' '7968' '1148' '2980' '4626' '3492' '1323' '2799' '6426' '186'
 '5498' '7060' '1752' '1816' '609' '2514' '6276' '8421' '8153_' '4609'
 '1625' '4603' '7026' '4010' '1143' '5870' '3607' '8505' '528' '1578'
 '4414' '6704' '2509' '7353' '3075' '2081' '925' '3602' '5930' '306'
 '7197' '1066' '6510' '1062' '5221' '7166' '4391' '6651' '1227' '3214'
 '8472' '836' '7553' '7504' '1357' '3284' '2721' '6385' '2997' '2451'
 '3480' '3357' '711' '8632' '1388' '5696' '5524' '356' '6608' '5705'
 '8567' '3132' '1695' '8639' '8582' '2275' '4959' '3834_' '623' '3345'
 '3493' '182' '126' '6744_' '4390' '7715' '6130' '6471_' '7723_' '3768'
 '4178' '4630' '1420' '6417' '2560' '3985' '3750' '4444' '234' '3795'
 '637' '4380' '5714' '7693' '2174' '2366' '5677' '3038' '4897' '7937'
 '365' '6408_' '1112' '3546' '8424' '124' '5747' '8082' '7564' '1436'
 '3519' '6175' '2091' '8249' '7316_' '2578' '4687' '4163' '6943' '6345'
 '1364' '6777' '5981' '963' '5907' '6978' '673' '6322' '6384' '1270'
 '8561' '2352' '6800' '6781' '7042' '1868' '2764' '6796' '6646' '142'
 '6141' '1852' '4416' '8466' '4576' '3410' '3581' '5479' '1102_' '8669_'
 '5074' '2463_' '3940' '5508' '1788' '5173' '7965' '7987' '6666_' '921'
 '4719' '7092' '1657' '5738' '349' '6995' '7441' '5976' '6485' '7702'
 '5708' '946' '4564' '3093' '6767' '3933' '3458' '1300' '5851' '774'
 '2577' '7480' '7113' '6273' '1452' '7520' '7414' '4370' '4649' '7845'
 '2864' '7925' '5550' '4165' '4063' '3215' '169' '3055_' '1248_' '586'
 '2386' '5649' '8520' '100' '3978' '3244' '3778' '879' '1024' '423' '3938'
 '1083' '6036' '1484' '4643' '4510' '8514' '2220_' '6360' '5996' '6892'
 '2159_' '5001' '576' '5018' '387' '4890' '5507' '4661' '1386' '330'
 '6074' '8592' '102' '5906' '203' '3640' '8124' '4484' '2778' '2434' '742'
 '7794' '4246' '208' '7508' '5765' '7710' '1734' '5202' '853' '3450'
 '1630' '3843' '7038' '8384' '8179' '5551' '6593' '7490' '2574' '1644'
 '4975' '5478' '1176' '1418' '4679' '5751' '3779' '6611' '325' '7879'
 '6227' '6111' '3726' '6722' '4732' '1754' '2437' '2212' '8081' '3119'
 '7549' '4583_' '3452' '1908' '3988_' '5621' '7178' '5741' '4774' '5116'
 '3378' '8302' '7651' '404' '3023' '5032' '4787' '4107' '381' '8623'
 '6407' '3675' '1265' '399' '2461' '2977' '3490' '438' '1328' '2037'
 '1609' '7325' '8682' '4470' '899' '5532' '4083' '3096' '1971' '1094'
 '1294' '7736' '6366' '5287' '4244' '3853' '1402' '2672' '5008' '2155_'
 '4846' '7773' '2182' '5994' '6301' '3363' '8663' '678' '4056' '7279'
 '828' '4076' '1491' '6697' '5340' '813' '6001' '6197' '5177' '7014'
 '1686' '2382' '1378' '2650' '3264' '6006' '4736' '4213' '532' '506'
 '3276' '4525' '6653' '8306' '4177' '2048' '95' '6799' '267' '1786' '7787'
 '7797' '4590' '6378' '3369' '4542' '1366' '540' '6770_' '1497' '6550'
 '5539' '7133' '5156' '1363' '1648' '993' '1330' '5066' '6463' '6349'
 '2988' '3145' '793' '1119' '8159' '8396' '3193' '3085' '5959' '1843_'
 '449' '5377' '4387' '5902' '620' '514' '6915' '2974' '6381' '652' '1367_'
 '553' '1355' '8291' '8154' '7933' '2347' '6098' '2731' '3742_' '2593'
 '4232' '5909' '2171_' '4120' '2449' '7556' '1404' '7856' '640' '2847'
 '2093' '1571' '846' '1435' '8333' '6863' '655' '6262' '1053' '3747'
 '4931' '7315' '292' '2499' '4847' '736' '2858' '6633' '2989' '4135'
 '3639' '4746' '1787' '4298' '8049' '6909' '7894' '8525' '1947' '7765'
 '8234' '6708' '2404' '6506' '1661' '8246' '7923' '2047' '3365' '4863'
 '701' '8450' '581' '502' '1618' '2077' '737' '8251' '2305' '7298' '221'
 '6663' '6657' '6588' '3463' '6337' '6516' '3984_' '2076' '2513' '6280'
 '3237' '3454' '4622' '5712' '275' '5868' '2096' '8235' '5471' '1785'
 '1593' '2933' '7131' '7444' '5046_' '4133' '3258' '4407' '7715_' '1565'
 '6159' '2276' '7459']

Occupation ['Scientist' '_______' 'Teacher' 'Engineer' 'Entrepreneur' 'Developer'
 'Lawyer' 'Media_Manager' 'Doctor' 'Journalist' 'Manager' 'Accountant'
 'Musician' 'Mechanic' 'Writer' 'Architect']

Annual_Income ['19114.12' '34847.84' '34847.84_' ... '45675' '35662.88' '35662.88_']

Delay_from_due_date [ 3  5  8  7 13 10  0  4  1  9 11 -1 30 31 34 27 14  2 -2 16 17 15 23 22
```

```
12 18 19 51 53 26 48 43 52 28 25 20 49 61 29 50 58 45  6 55 56 59 57 54
62 67 36 41 21 24 65 33 32 39 47 46 60 64 35 44 38 -3 63 42 40 37 -5 -4
66]

Num_of_Delayed_Payment ['7' '4' '8_' '1' '-1' '3_' '0' '8' '6' '5' '3' '9' '12' '15' '17' '2'
'2_' '14' '11' '20' '22' '10' '13' '13_' '14_' '16' '12_' '18' '19' '23'
'24' '21' '3318' '3083' '22_' '1338' '4_' '26' '11_' '3104' '25' '10_'
'183_' '9_' '1106' '834' '19_' '24_' '23_' '2672' '20_' '2008' '-3' '538'
'6_' '1_' '16_' '27' '-2' '3478' '2420' '15_' '707' '26_' '18_' '28'
'17_' '5_' '1867' '2250' '1463' '7_' '4126' '2882' '1941' '2655' '2628'
'132' '3069' '306' '0_' '3539' '3684' '1823' '4128' '1946' '827' '2297'
'2566' '904' '929' '3568' '2503' '1552' '2812' '1697' '851' '3905' '923'
'88' '1668' '3253' '808' '21_' '2689' '3858' '642' '3457' '1402' '1732'
'847' '3037' '3103' '1063' '2056' '1282' '1841' '2569_' '25_' '211' '793'
'3484' '411' '3491' '2072' '3050' '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' '3545' '779'
'192' '4311' '-2_' '2323' '1471' '3529' '439' '3456' '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' '2438' '809' '47' '1996' '1370' '1204' '2167'
'4011' '2594' '2533' '1663' '1018' '3316' '2589' '2801' '3355' '2529'
'2488' '4266' '1243' '739' '845' '4107' '1884' '337' '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' '2694' '519' '2677' '2413' '4139' '2609' '4326' '4211'
'823' '3011' '1608' '2860' '4219' '4047' '1531' '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' '787'
'4135' '3878' '1218' '4051' '1766' '1359' '3107' '585' '1263' '709'
'3632' '4077' '2943' '2793' '3245' '2317' '1640' '2237_' '3819' '3978'
'1498' '1833' '2737' '1192' '1481' '700' '271' '2286' '273' '3944' '1478'
'3749' '871' '2959' '3097_' '3511' '415' '2196' '2138' '2149' '1874'
'1553' '3847' '1222' '3051' '98' '1598' '416' '2314' '2955']

Outstanding_Debt ['809.98' '605.03' '1303.01' ... '3650.33' '1771.8' '2391.98']

Credit_History_Age ['22 Years and 1 Months' '22 Years and 3 Months' '22 Years and 4 Months'
'22 Years and 6 Months' '22 Years and 7 Months' '26 Years and 7 Months'
'26 Years and 8 Months' '26 Years and 9 Months' '26 Years and 10 Months'
'26 Years and 11 Months' '27 Years and 0 Months' '27 Years and 1 Months'
'27 Years and 2 Months' '17 Years and 9 Months' '17 Years and 10 Months'
'17 Years and 11 Months' '18 Years and 1 Months' '18 Years and 2 Months'
'18 Years and 3 Months' '18 Years and 4 Months' '17 Years and 3 Months'
'17 Years and 4 Months' '17 Years and 5 Months' '17 Years and 6 Months'
'17 Years and 7 Months' '17 Years and 8 Months' '30 Years and 10 Months'
'30 Years and 11 Months' '31 Years and 0 Months' '31 Years and 1 Months'
'31 Years and 2 Months' '31 Years and 3 Months' '32 Years and 0 Months'
'32 Years and 2 Months' '32 Years and 3 Months' '32 Years and 5 Months'
'32 Years and 6 Months' '30 Years and 7 Months' '30 Years and 8 Months'
'30 Years and 9 Months' '14 Years and 8 Months' '14 Years and 9 Months'
'14 Years and 10 Months' '14 Years and 11 Months' '15 Years and 0 Months'
'15 Years and 1 Months' '15 Years and 2 Months' '21 Years and 4 Months'
'21 Years and 5 Months' '21 Years and 6 Months' '21 Years and 7 Months'
'21 Years and 8 Months' '21 Years and 9 Months' '21 Years and 11 Months'
'26 Years and 6 Months' '19 Years and 2 Months' '19 Years and 3 Months'
'19 Years and 4 Months' '19 Years and 5 Months' '19 Years and 6 Months'
'19 Years and 7 Months' '19 Years and 8 Months' '25 Years and 5 Months'
'25 Years and 6 Months' '25 Years and 7 Months' '25 Years and 8 Months'
'25 Years and 9 Months' '25 Years and 10 Months' '25 Years and 11 Months'
'27 Years and 3 Months' '27 Years and 4 Months' '27 Years and 5 Months'
'8 Years and 11 Months' '9 Years and 0 Months' '9 Years and 2 Months'
'9 Years and 3 Months' '9 Years and 4 Months' '9 Years and 6 Months'
'18 Years and 5 Months' '18 Years and 6 Months' '18 Years and 8 Months'
'18 Years and 9 Months' '16 Years and 10 Months' '16 Years and 11 Months'
'17 Years and 0 Months' '17 Years and 1 Months' '17 Years and 2 Months'
'29 Years and 2 Months' '29 Years and 3 Months' '29 Years and 4 Months'
'29 Years and 6 Months' '29 Years and 8 Months' '29 Years and 9 Months'
'6 Years and 5 Months' '6 Years and 6 Months' '6 Years and 7 Months'
'6 Years and 8 Months' '6 Years and 10 Months' '7 Years and 0 Months'
'27 Years and 7 Months' '27 Years and 8 Months' '27 Years and 9 Months'
'18 Years and 7 Months' '19 Years and 9 Months' '19 Years and 10 Months'
'10 Years and 2 Months' '10 Years and 3 Months' '10 Years and 4 Months'
'10 Years and 6 Months' '10 Years and 7 Months' '10 Years and 8 Months'
'32 Years and 9 Months' '32 Years and 10 Months' '32 Years and 11 Months'
'33 Years and 0 Months' '33 Years and 1 Months' '33 Years and 4 Months'
'12 Years and 3 Months' '12 Years and 4 Months' '12 Years and 5 Months'
'12 Years and 6 Months' '12 Years and 7 Months' '12 Years and 8 Months'
'12 Years and 10 Months' '12 Years and 9 Months' '13 Years and 8 Months'
'13 Years and 11 Months' '14 Years and 0 Months' '14 Years and 1 Months'
'14 Years and 2 Months' '14 Years and 3 Months' '30 Years and 3 Months'
'30 Years and 4 Months' '30 Years and 5 Months' '30 Years and 6 Months'
'8 Years and 9 Months' '8 Years and 10 Months' '9 Years and 1 Months'
'18 Years and 10 Months' '18 Years and 11 Months' '19 Years and 0 Months'
'19 Years and 1 Months' '8 Years and 8 Months' '13 Years and 1 Months'
'13 Years and 2 Months' '13 Years and 3 Months' '13 Years and 5 Months'
'13 Years and 7 Months' '21 Years and 10 Months' '22 Years and 0 Months'
'26 Years and 0 Months' '26 Years and 1 Months' '26 Years and 2 Months'
'13 Years and 4 Months' '13 Years and 6 Months' '13 Years and 9 Months'
```

```
 '27 Years and 11 Months' '28 Years and 0 Months' '28 Years and 1 Months'
 '28 Years and 2 Months' '28 Years and 3 Months' '28 Years and 4 Months'
 '28 Years and 5 Months' '28 Years and 6 Months' '7 Years and 10 Months'
 '7 Years and 11 Months' '8 Years and 0 Months' '8 Years and 1 Months'
 '8 Years and 2 Months' '8 Years and 3 Months' '8 Years and 4 Months'
 '8 Years and 5 Months' '24 Years and 3 Months' '24 Years and 7 Months'
 '24 Years and 8 Months' '24 Years and 9 Months' '1 Years and 3 Months'
 '1 Years and 4 Months' '1 Years and 5 Months' '1 Years and 6 Months'
 '1 Years and 7 Months' '1 Years and 8 Months' '11 Years and 0 Months'
 '11 Years and 1 Months' '11 Years and 2 Months' '11 Years and 3 Months'
 '11 Years and 4 Months' '11 Years and 5 Months' '11 Years and 6 Months'
 '20 Years and 0 Months' '20 Years and 1 Months' '10 Years and 9 Months'
 '10 Years and 10 Months' '14 Years and 4 Months' '14 Years and 5 Months'
 '14 Years and 6 Months' '20 Years and 9 Months' '21 Years and 0 Months'
 '21 Years and 1 Months' '21 Years and 2 Months' '21 Years and 3 Months'
 '0 Years and 4 Months' '0 Years and 5 Months' '0 Years and 6 Months'
 '0 Years and 9 Months' '0 Years and 10 Months' '31 Years and 7 Months'
 '31 Years and 8 Months' '31 Years and 9 Months' '31 Years and 11 Months'
 '32 Years and 1 Months' '12 Years and 11 Months' '13 Years and 0 Months'
 '27 Years and 6 Months' '27 Years and 10 Months' '11 Years and 7 Months'
 '11 Years and 8 Months' '11 Years and 9 Months' '11 Years and 10 Months'
 '24 Years and 10 Months' '24 Years and 11 Months' '25 Years and 0 Months'
 '25 Years and 1 Months' '25 Years and 2 Months' '25 Years and 3 Months'
 '10 Years and 1 Months' '10 Years and 5 Months' '31 Years and 4 Months'
 '31 Years and 5 Months' '31 Years and 6 Months' '5 Years and 2 Months'
 '5 Years and 3 Months' '5 Years and 4 Months' '5 Years and 5 Months'
 '5 Years and 6 Months' '5 Years and 7 Months' '5 Years and 8 Months'
 '5 Years and 9 Months' '2 Years and 11 Months' '3 Years and 0 Months'
 '3 Years and 1 Months' '3 Years and 2 Months' '3 Years and 3 Months'
 '3 Years and 4 Months' '3 Years and 5 Months' '3 Years and 6 Months'
 '24 Years and 4 Months' '24 Years and 5 Months' '24 Years and 6 Months'
 '16 Years and 4 Months' '16 Years and 5 Months' '16 Years and 6 Months'
 '16 Years and 7 Months' '16 Years and 8 Months' '16 Years and 9 Months'
 '22 Years and 11 Months' '23 Years and 3 Months' '23 Years and 4 Months'
 '23 Years and 5 Months' '23 Years and 6 Months' '8 Years and 6 Months'
 '8 Years and 7 Months' '4 Years and 5 Months' '4 Years and 6 Months'
 '4 Years and 7 Months' '4 Years and 8 Months' '4 Years and 9 Months'
 '4 Years and 10 Months' '4 Years and 11 Months' '5 Years and 0 Months'
 '32 Years and 8 Months' '33 Years and 2 Months' '33 Years and 3 Months'
 '12 Years and 2 Months' '32 Years and 4 Months' '29 Years and 11 Months'
 '30 Years and 0 Months' '30 Years and 2 Months' '26 Years and 3 Months'
 '26 Years and 4 Months' '26 Years and 5 Months' '18 Years and 0 Months'
 '7 Years and 6 Months' '7 Years and 7 Months' '7 Years and 9 Months'
 '28 Years and 7 Months' '28 Years and 8 Months' '28 Years and 9 Months'
 '28 Years and 10 Months' '29 Years and 5 Months' '29 Years and 7 Months'
 '19 Years and 11 Months' '20 Years and 2 Months' '20 Years and 3 Months'
 '20 Years and 4 Months' '20 Years and 5 Months' '20 Years and 6 Months'
 '20 Years and 7 Months' '20 Years and 8 Months' '28 Years and 11 Months'
 '29 Years and 0 Months' '13 Years and 10 Months' '1 Years and 10 Months'
 '1 Years and 11 Months' '33 Years and 5 Months' '33 Years and 6 Months'
 '33 Years and 7 Months' '33 Years and 8 Months' '29 Years and 1 Months'
 '31 Years and 10 Months' '5 Years and 10 Months' '5 Years and 11 Months'
 '6 Years and 0 Months' '6 Years and 1 Months' '6 Years and 2 Months'
 '6 Years and 3 Months' '22 Years and 5 Months' '22 Years and 9 Months'
 '22 Years and 10 Months' '23 Years and 1 Months' '23 Years and 2 Months'
 '22 Years and 2 Months' '15 Years and 4 Months' '15 Years and 5 Months'
 '15 Years and 6 Months' '15 Years and 7 Months' '15 Years and 8 Months'
 '15 Years and 9 Months' '15 Years and 10 Months' '15 Years and 11 Months'
 '2 Years and 3 Months' '2 Years and 4 Months' '2 Years and 5 Months'
 '2 Years and 6 Months' '2 Years and 7 Months' '2 Years and 8 Months'
 '2 Years and 9 Months' '2 Years and 10 Months' '5 Years and 1 Months'
 '1 Years and 9 Months' '2 Years and 0 Months' '16 Years and 2 Months'
 '16 Years and 3 Months' '22 Years and 8 Months' '9 Years and 5 Months'
 '9 Years and 7 Months' '9 Years and 8 Months' '9 Years and 9 Months'
 '11 Years and 11 Months' '12 Years and 0 Months' '12 Years and 1 Months'
 '23 Years and 0 Months' '16 Years and 0 Months' '16 Years and 1 Months'
 '25 Years and 4 Months' '15 Years and 3 Months' '6 Years and 11 Months'
 '7 Years and 1 Months' '7 Years and 2 Months' '7 Years and 4 Months'
 '7 Years and 5 Months' '23 Years and 7 Months' '23 Years and 8 Months'
 '23 Years and 9 Months' '20 Years and 11 Months' '30 Years and 1 Months'
 '7 Years and 3 Months' '7 Years and 8 Months' '9 Years and 10 Months'
 '9 Years and 11 Months' '10 Years and 0 Months' '2 Years and 2 Months'
 '23 Years and 10 Months' '23 Years and 11 Months' '24 Years and 0 Months'
 '24 Years and 2 Months' '14 Years and 7 Months' '10 Years and 11 Months'
 '24 Years and 1 Months' '6 Years and 4 Months' '0 Years and 1 Months'
 '0 Years and 2 Months' '0 Years and 3 Months' '0 Years and 7 Months'
 '0 Years and 8 Months' '29 Years and 10 Months' '3 Years and 8 Months'
 '32 Years and 7 Months' '20 Years and 10 Months' '3 Years and 7 Months'
 '3 Years and 9 Months' '3 Years and 10 Months' '6 Years and 9 Months'
 '0 Years and 11 Months' '1 Years and 0 Months' '1 Years and 1 Months'
 '1 Years and 2 Months' '4 Years and 4 Months' '3 Years and 11 Months'
 '4 Years and 1 Months' '4 Years and 2 Months' '4 Years and 3 Months'
 '2 Years and 1 Months' '4 Years and 0 Months']

Payment_of_Min_Amount ['No' 'NM' 'Yes']

Total_EMI_per_month [ 49.57494921  18.81621457 246.9923195  ...  51.17896891 134.2702558
  60.78774439]

Payment_Behaviour ['High_spent_Small_value_payments' 'Low_spent_Medium_value_payments'
 'Low_spent_Small_value_payments' '!@9#%8'
 'High_spent_Large_value_payments' 'High_spent_Medium_value_payments'
```

```
'Low_spent_Large_value_payments']

Monthly_Balance ['312.4940887' '331.2098629' '223.4513097' ... '357.56701' '401.2391219'
'315.3263847']

Credit_Score ['Good' 'Standard' 'Poor']
```

In [ ]:
```
cupation'].str.contains('_______') == False] #drop rows the 'Occupation' column contains the string '_______'.
yment_Behaviour'].str.contains('!@9#%8') == False] #drop rows where the 'Payment_Behaviour' column contains the string '!@9#%
```

In [11]:
```
sym = "\\`*_{}[]()>#@+!$:;"
col_int = ['Age','Delay_from_due_date','Num_of_Delayed_Payment','Outstanding_Debt',
           'Total_EMI_per_month','Monthly_Balance','Annual_Income']
col_str = ['Occupation','Credit_History_Age','Payment_of_Min_Amount','Credit_Score']
for i in col_int:    # Loop over integer columns
  for c in sym:      # Loop over symbols in the 'sym' string
    drop_na[i] = drop_na[i].astype(str).str.replace(c,'')
for i in col_str:  #Loop over string columns
  for c in sym:  # Loop over symbols in the 'sym' string
    drop_na[i] = drop_na[i].replace(c,'') # Replace each symbol with an empty string in the specified column
drop_na.head()   ## Display the result of the cleaned DataFrame
```

Out[11]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	Tot
0	23	Scientist	19114.12	3	7	809.98	22 Years and 1 Months	No	
2	-500	Scientist	19114.12	3	7	809.98	22 Years and 3 Months	No	
3	23	Scientist	19114.12	5	4	809.98	22 Years and 4 Months	No	
6	23	Scientist	19114.12	3	8	809.98	22 Years and 7 Months	No	
9	28	Teacher	34847.84	7	1	605.03	26 Years and 8 Months	No	

In [12]:
```
#converts the 'Credit_History_Age' column to a string, then replaces the substring ' Years and ' with a dot '.'
drop_na['Credit_History_Age'] = drop_na['Credit_History_Age'].astype(str).str.replace(' Years and ','.')
#converts the 'Credit_History_Age' column to a string and then removes the substring 'Months'
drop_na['Credit_History_Age'] = drop_na['Credit_History_Age'].astype(str).str.replace('Months','')
```

In [13]:
```
#Payment_Behaviour' column repalce with numerical values.
drop_na['Payment_Behaviour'] = drop_na['Payment_Behaviour'].astype(str).str.replace('Low_spent_Small_value_payments','1')
drop_na['Payment_Behaviour'] = drop_na['Payment_Behaviour'].astype(str).str.replace('Low_spent_Medium_value_payments','2')
drop_na['Payment_Behaviour'] = drop_na['Payment_Behaviour'].astype(str).str.replace('Low_spent_Large_value_payments','3')
drop_na['Payment_Behaviour'] = drop_na['Payment_Behaviour'].astype(str).str.replace('High_spent_Small_value_payments','4')
drop_na['Payment_Behaviour'] = drop_na['Payment_Behaviour'].astype(str).str.replace('High_spent_Medium_value_payments','5')
drop_na['Payment_Behaviour'] = drop_na['Payment_Behaviour'].astype(str).str.replace('High_spent_Large_value_payments','6')
drop_na.head()
```

Out[13]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	Tot
0	23	Scientist	19114.12	3	7	809.98	22.1	No	
2	-500	Scientist	19114.12	3	7	809.98	22.3	No	
3	23	Scientist	19114.12	5	4	809.98	22.4	No	
6	23	Scientist	19114.12	3	8	809.98	22.7	No	
9	28	Teacher	34847.84	7	1	605.03	26.8	No	

In [14]:
```
#converting of selected columns the float data type
col_int2 = ['Age','Delay_from_due_date','Num_of_Delayed_Payment','Outstanding_Debt',
            'Total_EMI_per_month','Monthly_Balance','Payment_Behaviour','Credit_History_Age','Annual_Income']
for i in col_int2:
  drop_na[i] = drop_na[i].astype(float)
drop_na.dtypes
```

Out[14]:
```
Age                       float64
Occupation                 object
Annual_Income             float64
Delay_from_due_date       float64
Num_of_Delayed_Payment    float64
Outstanding_Debt          float64
Credit_History_Age        float64
Payment_of_Min_Amount      object
Total_EMI_per_month       float64
Payment_Behaviour         float64
Monthly_Balance           float64
Credit_Score               object
dtype: object
```

In [15]:
```
#replaces specific string values in the 'Credit_Score' column with Good:3 ,Standard:2, Poor:1 by using pd.to_numeric
drop_na['Credit_Score'] = drop_na['Credit_Score'].str.replace('Good', '3', n=-1)
drop_na['Credit_Score'] = drop_na['Credit_Score'].str.replace('Standard', '2', n=-1)
drop_na['Credit_Score'] = drop_na['Credit_Score'].str.replace('Poor', '1', n=-1)
drop_na['Credit_Score'] = drop_na[['Credit_Score']].apply(pd.to_numeric)
#converts the values in the 'Payment_of_Min_Amount' column to numeric values using pd.to_numeric
drop_na['Payment_of_Min_Amount'] = drop_na['Payment_of_Min_Amount'].str.replace('NM', '0')
drop_na['Payment_of_Min_Amount'] = drop_na['Payment_of_Min_Amount'].str.replace('Yes', '1')
drop_na['Payment_of_Min_Amount'] = drop_na['Payment_of_Min_Amount'].str.replace('No', '2')
drop_na['Payment_of_Min_Amount'] = drop_na[['Payment_of_Min_Amount']].apply(pd.to_numeric)
drop_na
```

Out[15]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amou
0	23.0	Scientist	19114.12	3.0	7.0	809.98	22.10	
2	-500.0	Scientist	19114.12	3.0	7.0	809.98	22.30	
3	23.0	Scientist	19114.12	5.0	4.0	809.98	22.40	
6	23.0	Scientist	19114.12	3.0	8.0	809.98	22.70	
9	28.0	Teacher	34847.84	7.0	1.0	605.03	26.80	
...	...	...	...	...	...	...	...	
49992	17.0	Developer	35662.88	19.0	16.0	2391.98	18.70	
49994	17.0	Developer	35662.88	19.0	13.0	2391.98	18.90	
49995	17.0	Developer	35662.88	19.0	14.0	2391.98	18.10	
49996	17.0	Developer	35662.88	19.0	16.0	2391.98	18.11	
49998	18.0	Developer	35662.88	19.0	15.0	2391.98	19.10	

35962 rows × 12 columns

In [16]:
```
#Collect only integer in df['Age']
def extract_numeric(valChecks if there is a match (numeric digits were found).ue):  #defines a function named extract_numeric
    match = re.search(r'\d+', str(value)) #Uses a regular expression to search for one or more numeric digits (\d+)
    if match: #Checks if there is a match (numeric digits were found).
        return int(match.group()) #If there is a match, the function returns the integer representation of the matched digits
    else:
        return None

drop_na['Age'] = drop_na['Age'].apply(extract_numeric) #Applies the extract_numeric function to each element in the 'Age' col
```

In [17]:
```
drop_na['Age'] = drop_na['Age'].astype(int)  #Converts the 'Age' column in the DataFrame drop_na to the integer data type usi
drop_na = drop_na[(drop_na['Age'] >= 0) & (drop_na['Age'] <= 150)]  #Including only rows where the 'Age' column values are gr
```

In [18]:
```
drop_na.count()  #count the number of non-null  values in each column of the DataFrame
```

Out[18]:
```
Age                      34943
Occupation               34943
Annual_Income            34943
Delay_from_due_date      34943
Num_of_Delayed_Payment   34943
Outstanding_Debt         34943
Credit_History_Age       34943
Payment_of_Min_Amount    34943
Total_EMI_per_month      34943
Payment_Behaviour        34943
Monthly_Balance          34943
Credit_Score             34943
dtype: int64
```

In [19]:
```
drop_na = drop_na.drop_duplicates() #used to remove duplicate rows from a DataFrame
drop_na.count()
```

Out[19]:
```
Age                      34943
Occupation               34943
Annual_Income            34943
Delay_from_due_date      34943
Num_of_Delayed_Payment   34943
Outstanding_Debt         34943
Credit_History_Age       34943
Payment_of_Min_Amount    34943
Total_EMI_per_month      34943
Payment_Behaviour        34943
Monthly_Balance          34943
Credit_Score             34943
dtype: int64
```

In [20]:
```
drop_na.head()  #display the first few rows of the DataFrame drop_na
```

Out[20]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	To
0	23	Scientist	19114.12	3.0	7.0	809.98	22.1	2	
3	23	Scientist	19114.12	5.0	4.0	809.98	22.4	2	
6	23	Scientist	19114.12	3.0	8.0	809.98	22.7	2	
9	28	Teacher	34847.84	7.0	1.0	605.03	26.8	2	
10	28	Teacher	34847.84	3.0	-1.0	605.03	26.9	2	

In [21]:
```
df_cleaned = drop_na #Assigning the DataFrame drop_na to a new variable called df_cleaned
```

In [22]:
```
df_cleaned.describe()  #using to generate descriptive statistics of the DataFrame df_cleaned
```

Out[22]:

	Age	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	To
count	34943.000000	3.494300e+04	34943.000000	34943.000000	34943.000000	34943.000000	34943.000000	
mean	33.269210	1.772806e+05	21.132301	30.478122	1417.479133	18.477834	1.237644	
std	10.794765	1.445054e+06	14.825659	221.328847	1158.982603	8.298258	0.648449	
min	14.000000	7.005930e+03	-5.000000	-3.000000	0.540000	0.100000	0.000000	
25%	24.000000	1.935675e+04	10.000000	9.000000	557.020000	12.100000	1.000000	
50%	33.000000	3.696389e+04	18.000000	14.000000	1149.630000	18.300000	1.000000	
75%	42.000000	7.256770e+04	28.000000	18.000000	1923.900000	25.200000	2.000000	
max	142.000000	2.419806e+07	67.000000	4384.000000	4998.070000	33.800000	2.000000	

In [23]:
```
#Performing outlier removal for the 'Annual_Income' column in the DataFrame df_cleaned using the Interquartile Range (IQR) me
Q1 = df_cleaned.Annual_Income.quantile(0.25) #Calculate the 25th percentile (Q1) of the 'Annual_Income' column.
Q3 = df_cleaned.Annual_Income.quantile(0.75) #Calculate the 75th percentile (Q3) of the 'Annual_Income' column.
IQR = Q3 - Q1  #Calculate the Interquartile Range (IQR) as the difference between Q3 and Q1.
df_cleaned = df_cleaned.drop(df_cleaned.loc[df_cleaned['Annual_Income'] > (Q3 + 1.5 * IQR)].index) #Drop rows where 'Annual_I
df_cleaned = df_cleaned.drop(df_cleaned.loc[df_cleaned['Annual_Income'] < (Q1 - 1.5 * IQR)].index) #Drop rows where 'Annual_I
df_cleaned
```

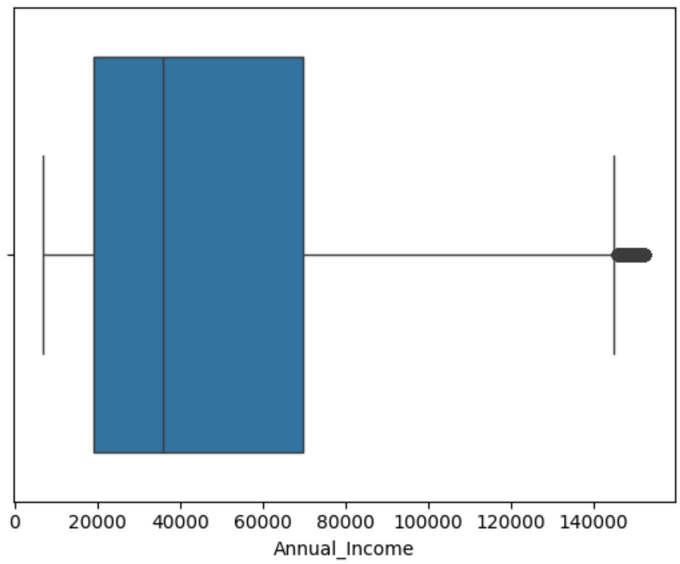
Out[23]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount
0	23	Scientist	19114.12	3.0	7.0	809.98	22.10	2
3	23	Scientist	19114.12	5.0	4.0	809.98	22.40	2
6	23	Scientist	19114.12	3.0	8.0	809.98	22.70	2
9	28	Teacher	34847.84	7.0	1.0	605.03	26.80	2
10	28	Teacher	34847.84	3.0	-1.0	605.03	26.90	2
...	...	...	...	...	...	...	...	...
49992	17	Developer	35662.88	19.0	16.0	2391.98	18.70	1
49994	17	Developer	35662.88	19.0	13.0	2391.98	18.90	1
49995	17	Developer	35662.88	19.0	14.0	2391.98	18.10	1
49996	17	Developer	35662.88	19.0	16.0	2391.98	18.11	1
49998	18	Developer	35662.88	19.0	15.0	2391.98	19.10	1

33937 rows × 12 columns

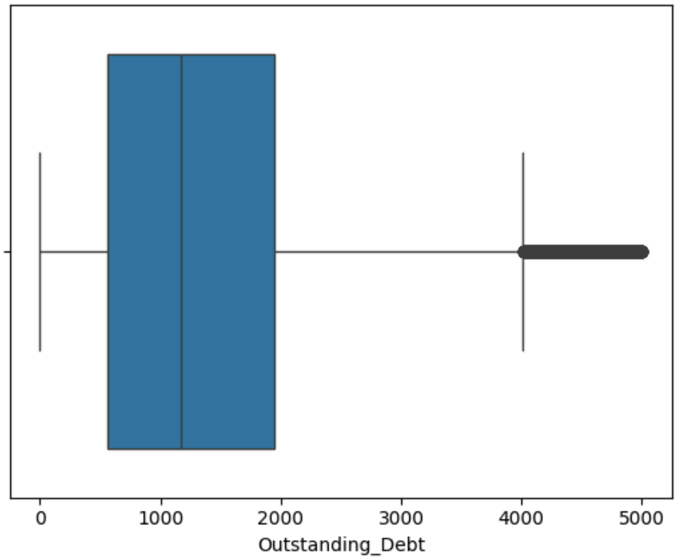
In [24]:
```
sns.boxplot(x=df_cleaned['Annual_Income'])  #Uses the Seaborn library to create a boxplot for the 'Annual_Income' column in t
```

Out[24]: `<Axes: xlabel='Annual_Income'>`



In [25]:
```
oxplot(x=df_cleaned['Outstanding_Debt']) #uses the Seaborn library to create a boxplotbfor the'Outstanding_Debt' column in the
```

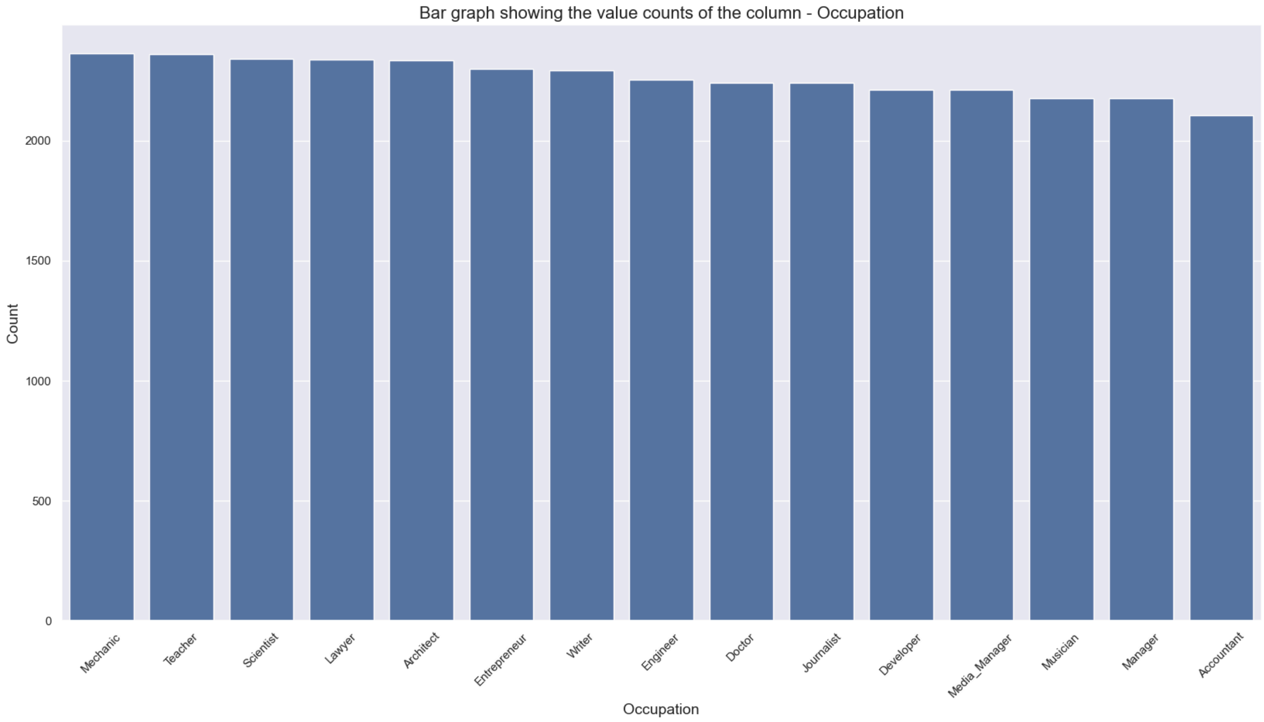
Out[25]: <Axes: xlabel='Outstanding_Debt'>



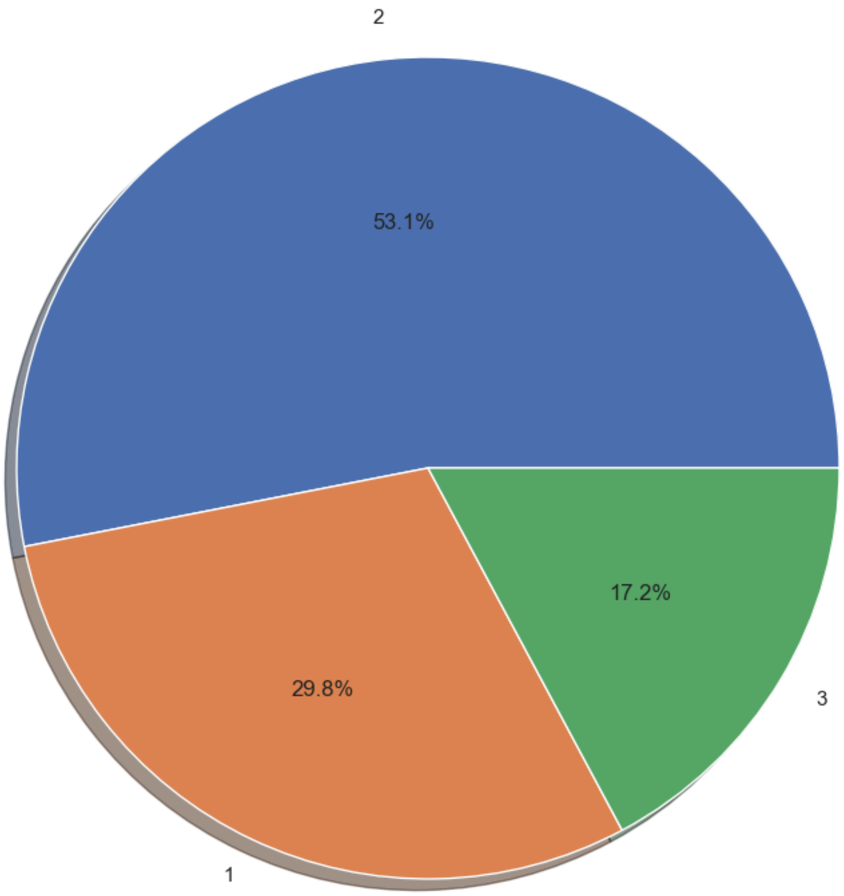
In [26]:
```
occupation_count = df_cleaned['Occupation'].value_counts(dropna = False) #using the value_counts() function to count unique v
occupation_count
```

Out[26]:
```
Occupation
Mechanic         2363
Teacher          2359
Scientist        2341
Lawyer           2338
Architect        2334
Entrepreneur     2298
Writer           2292
Engineer         2252
Doctor           2239
Journalist       2239
Developer        2213
Media_Manager    2210
Musician         2177
Manager          2176
Accountant       2106
Name: count, dtype: int64
```

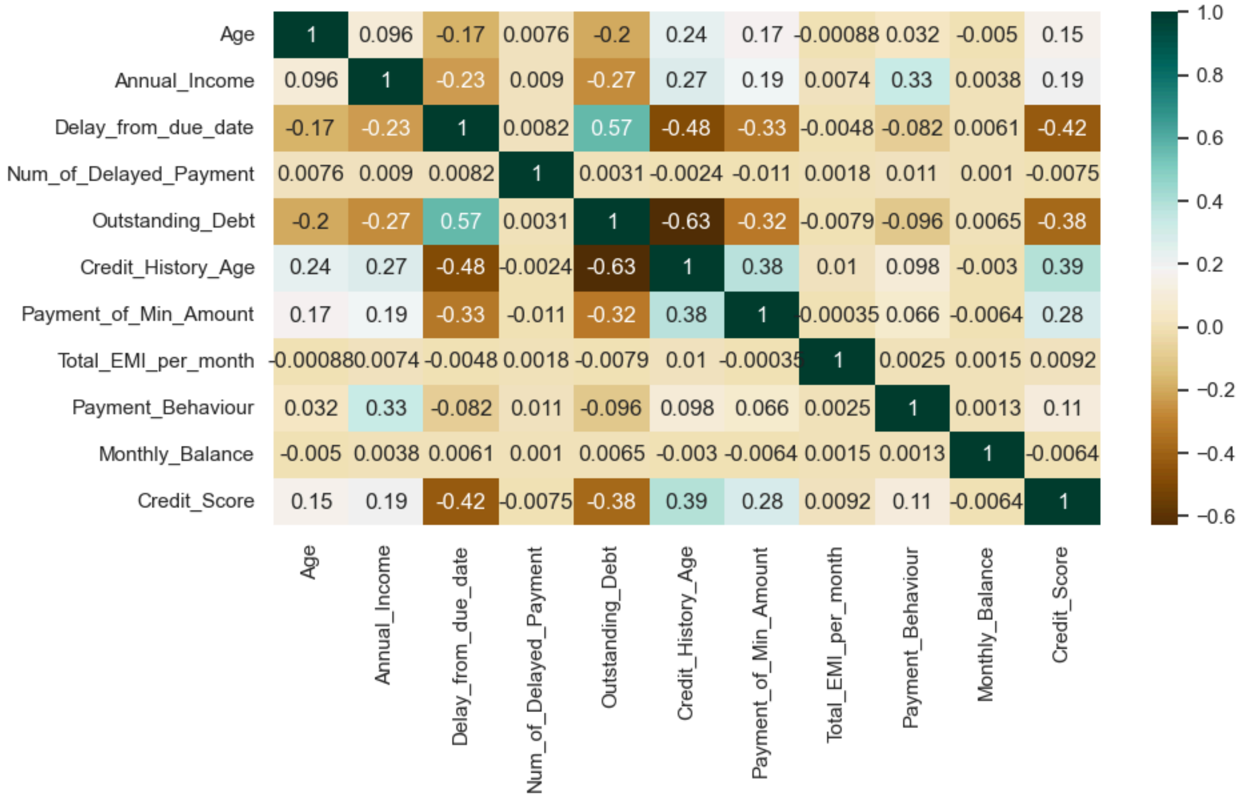
In [27]:
```
# Set the figure size
sns.set(rc={'figure.figsize': (20, 10)}) #Sets the figure size for the plot.
#Create a bar plot
sns.barplot(x=occupation_count.index, y=occupation_count.values)
# Set plot title and axis labels
plt.title('Bar graph showing the value counts of the column - Occupation', fontsize=16)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Occupation', fontsize=14)
# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
#Display the plot
plt.show()
```



In [28]:
```
label = df_cleaned.Credit_Score.value_counts().index  #Retrieves unique values (labels) in the 'Credit_Score' column.
label_count = df_cleaned.Credit_Score.value_counts().values  #Retrieves the corresponding counts for each unique value.
#Creating a Pie Chart:
plt.pie(data=df_cleaned, x=label_count, labels=label, autopct='%1.1f%%', shadow=True, radius=1)
plt.show() # Displays the pie chart.
```

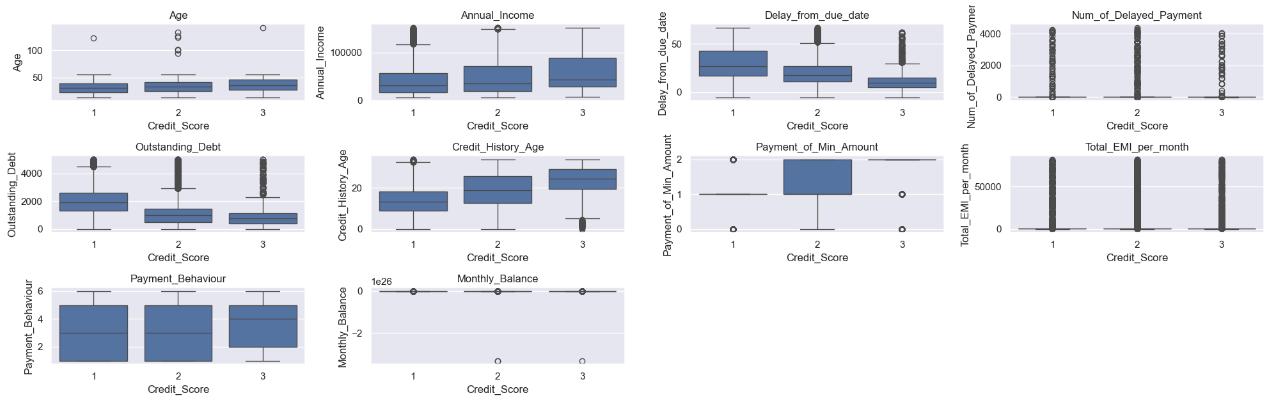


In [29]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Select only numeric columns for correlation
df_numeric = df_cleaned.select_dtypes(include=[np.number])

# Calculate correlation matrix
c = df_numeric.corr()

# Plot heatmap
plt.figure(figsize=(10,5))
sns.heatmap(c, cmap="BrBG", annot=True)
plt.show()

# Display correlation matrix
print(c)
```



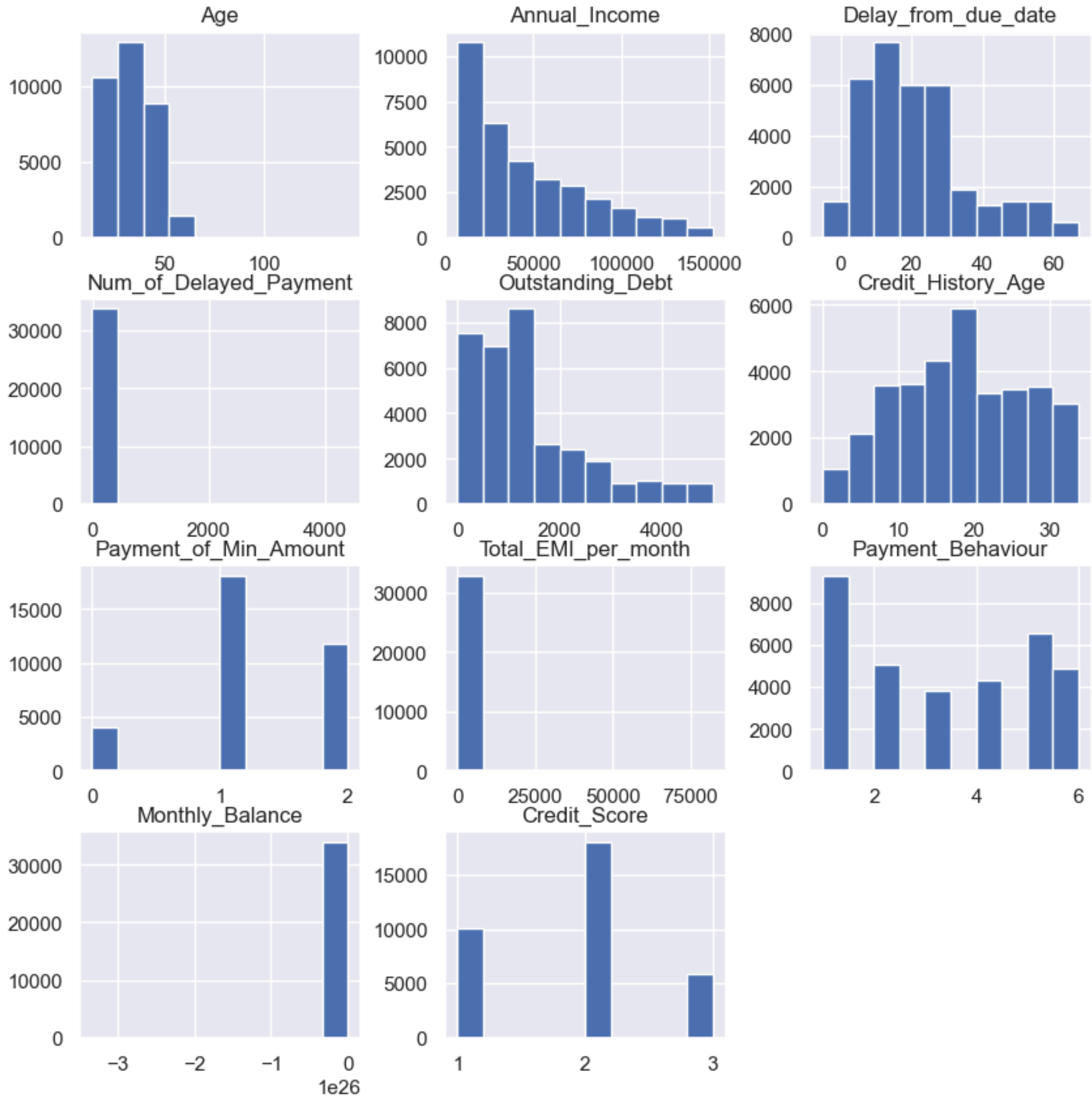
```
                            Age  Annual_Income  Delay_from_due_date  \
Age                    1.000000       0.095633            -0.174124   
Annual_Income          0.095633       1.000000            -0.230239   
Delay_from_due_date   -0.174124      -0.230239             1.000000   
Num_of_Delayed_Payment  0.007570       0.008966             0.008211   
Outstanding_Debt      -0.198137      -0.269094             0.570430   
Credit_History_Age     0.239645       0.272156            -0.484380   
Payment_of_Min_Amount  0.172988       0.186814            -0.329665   
Total_EMI_per_month   -0.000880       0.007359            -0.004823   
Payment_Behaviour      0.032079       0.327195            -0.081651   
Monthly_Balance       -0.005032       0.003796             0.006146   
Credit_Score           0.154511       0.194485            -0.424786   

                        Num_of_Delayed_Payment  Outstanding_Debt  \
Age                                   0.007570         -0.198137   
Annual_Income                         0.008966         -0.269094   
Delay_from_due_date                   0.008211          0.570430   
Num_of_Delayed_Payment                1.000000          0.003142   
Outstanding_Debt                      0.003142          1.000000   
Credit_History_Age                   -0.002359         -0.629654   
Payment_of_Min_Amount                -0.011139         -0.321478   
Total_EMI_per_month                   0.001847         -0.007893   
Payment_Behaviour                     0.011480         -0.096254   
Monthly_Balance                       0.001017          0.006464   
Credit_Score                         -0.007455         -0.380951   

                        Credit_History_Age  Payment_of_Min_Amount  \
Age                               0.239645               0.172988   
Annual_Income                     0.272156               0.186814   
Delay_from_due_date              -0.484380              -0.329665   
Num_of_Delayed_Payment           -0.002359              -0.011139   
Outstanding_Debt                 -0.629654              -0.321478   
Credit_History_Age                1.000000               0.382144   
Payment_of_Min_Amount             0.382144               1.000000   
Total_EMI_per_month               0.010400              -0.000351   
Payment_Behaviour                 0.097718               0.065564   
Monthly_Balance                  -0.002970              -0.006422   
Credit_Score                      0.389106               0.277669   

                        Total_EMI_per_month  Payment_Behaviour  \
Age                               -0.000880           0.032079   
Annual_Income                      0.007359           0.327195   
Delay_from_due_date               -0.004823          -0.081651   
Num_of_Delayed_Payment             0.001847           0.011480   
Outstanding_Debt                  -0.007893          -0.096254   
Credit_History_Age                 0.010400           0.097718   
Payment_of_Min_Amount             -0.000351           0.065564   
Total_EMI_per_month                1.000000           0.002499   
Payment_Behaviour                  0.002499           1.000000   
Monthly_Balance                    0.001513           0.001266   
Credit_Score                       0.009186           0.112404   

                        Monthly_Balance  Credit_Score  
Age                           -0.005032      0.154511  
Annual_Income                  0.003796      0.194485  
Delay_from_due_date            0.006146     -0.424786  
Num_of_Delayed_Payment         0.001017     -0.007455  
Outstanding_Debt               0.006464     -0.380951  
Credit_History_Age            -0.002970      0.389106  
Payment_of_Min_Amount         -0.006422      0.277669  
Total_EMI_per_month            0.001513      0.009186  
Payment_Behaviour              0.001266      0.112404  
Monthly_Balance                1.000000     -0.006411  
Credit_Score                  -0.006411      1.000000  
```

In [48]:
```
#create a set of box plots, with Credit_Score' in the DataFrame
for ax,col in enumerate(df_numeric.columns[:-1]):
    plt.subplot(5,4,ax+1)
    plt.title(col)
    sns.boxplot(x='Credit_Score', y=col, data=df_numeric)
    # plt.legend()
plt.tight_layout()
```



In [30]:
```
df_cleaned.hist(figsize=(10, 10))# create histograms  with figure to 10x10 inches.
```

Out[30]:
```
array([[<Axes: title={'center': 'Age'}>,
        <Axes: title={'center': 'Annual_Income'}>,
        <Axes: title={'center': 'Delay_from_due_date'}>],
       [<Axes: title={'center': 'Num_of_Delayed_Payment'}>,
        <Axes: title={'center': 'Outstanding_Debt'}>,
        <Axes: title={'center': 'Credit_History_Age'}>],
       [<Axes: title={'center': 'Payment_of_Min_Amount'}>,
        <Axes: title={'center': 'Total_EMI_per_month'}>,
        <Axes: title={'center': 'Payment_Behaviour'}>],
       [<Axes: title={'center': 'Monthly_Balance'}>,
        <Axes: title={'center': 'Credit_Score'}>, <Axes: >]], dtype=object)
```



In [31]:
```
df_cleaned.head() #display column
```

Out[31]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	Tc
0	23	Scientist	19114.12	3.0	7.0	809.98	22.1	2	
3	23	Scientist	19114.12	5.0	4.0	809.98	22.4	2	
6	23	Scientist	19114.12	3.0	8.0	809.98	22.7	2	
9	28	Teacher	34847.84	7.0	1.0	605.03	26.8	2	
10	28	Teacher	34847.84	3.0	-1.0	605.03	26.9	2	

In [32]:
```
scaler = MinMaxScaler() #Uses the MinMaxScaler scaling on selected numerical columns in the DataFrame df_cleaned
col_float = ['Age','Annual_Income','Delay_from_due_date','Num_of_Delayed_Payment',
             'Outstanding_Debt','Credit_History_Age','Total_EMI_per_month','Monthly_Balance']
for i in df_cleaned[col_float]:
  df_cleaned[i] = scaler.fit_transform(df_cleaned[[i]])
df_cleaned.head()
```

Out[32]:

	Age	Occupation	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amoun
0	0.070312	Scientist	0.083312	0.111111	0.002279	0.161968	0.652819	:
3	0.070312	Scientist	0.083312	0.138889	0.001596	0.161968	0.661721	:
6	0.070312	Scientist	0.083312	0.111111	0.002507	0.161968	0.670623	:
9	0.109375	Teacher	0.191571	0.166667	0.000912	0.120958	0.792285	:
10	0.109375	Teacher	0.191571	0.111111	0.000456	0.120958	0.795252	:

In [33]:
```
df_cleaned.columns  #Display columns
```

Out[33]:
```
Index(['Age', 'Occupation', 'Annual_Income', 'Delay_from_due_date',
       'Num_of_Delayed_Payment', 'Outstanding_Debt', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month', 'Payment_Behaviour',
       'Monthly_Balance', 'Credit_Score'],
      dtype='object')
```

In [34]:
```
#get_dummies function to perform one-hot encoding on the 'Occupation' column
df_cleaned = pd.get_dummies(df_cleaned, prefix='Occupation', columns=['Occupation'], drop_first=False)
df_cleaned.head()
```

Out[34]:

	Age	Annual_Income	Delay_from_due_date	Num_of_Delayed_Payment	Outstanding_Debt	Credit_History_Age	Payment_of_Min_Amount	Total_EMI_
0	0.070312	0.083312	0.111111	0.002279	0.161968	0.652819	2	
3	0.070312	0.083312	0.138889	0.001596	0.161968	0.661721	2	
6	0.070312	0.083312	0.111111	0.002507	0.161968	0.670623	2	
9	0.109375	0.191571	0.166667	0.000912	0.120958	0.792285	2	
10	0.109375	0.191571	0.111111	0.000456	0.120958	0.795252	2	

5 rows × 26 columns

In [ ]:
```
#extracts a subset of columns from the DataFrame df_cleaned and assigns it to a new DataFrame called feed
feed = df_cleaned[['Age', 'Annual_Income', 'Delay_from_due_date', 'Num_of_Delayed_Payment',
       'Outstanding_Debt', 'Credit_History_Age', 'Payment_of_Min_Amount',
       'Total_EMI_per_month', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score', 'Occupation_Accountant', 'Occupation_Architect',
       'Occupation_Developer', 'Occupation_Doctor', 'Occupation_Engineer',
       'Occupation_Entrepreneur', 'Occupation_Journalist', 'Occupation_Lawyer',
       'Occupation_Manager', 'Occupation_Mechanic', 'Occupation_Media_Manager',
       'Occupation_Musician', 'Occupation_Scientist', 'Occupation_Teacher',
       'Occupation_Writer']]
```

In [ ]:
```
#Data preparation process for a machine learning
df_train_x = feed.drop('Credit_Score',axis = 1) #Creates a  df_train_x and dropping the 'Credit_Score' column.
df_train_y = feed['Credit_Score'] #Creates a  df_train_y that stores the target variable 'Credit_Score'.
#the size of the test dataset as 20% of the total data.Each time the program is run, based on the seed value 42.
x_train, x_test, y_train, y_test = train_test_split(df_train_x, df_train_y, test_size=0.20, random_state=42)
```

In [37]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.neighbors import KNeighborsClassifier

# Assuming 'feed' is your DataFrame and it's already been defined

# Splitting dataset into train and test
df_train_x = feed.drop('Credit_Score', axis=1)
df_train_y = feed['Credit_Score']
x_train, x_test, y_train, y_test = train_test_split(df_train_x, df_train_y, test_size=0.20, random_state=42)

# Training KNeighbors Classifier
kn = KNeighborsClassifier()
kn.fit(x_train, y_train)

# Predicting the test set results
kn_y_pred = kn.predict(x_test)

# Confusion Matrix
kn_cm = confusion_matrix(y_test, kn_y_pred)
print("Confusion Matrix:")
print(kn_cm)
print("  ")

# Accuracy
kn_accuracy = accuracy_score(y_test, kn_y_pred)
print("Accuracy:", kn_accuracy)

# Precision, Recall, and F1 Score
kn_precision = precision_score(y_test, kn_y_pred, average='macro')  # Use 'micro', 'macro', or 'weighted' based on your class
kn_recall = recall_score(y_test, kn_y_pred, average='macro')
kn_f1 = f1_score(y_test, kn_y_pred, average='macro')

print("Precision:", kn_precision)
print("Recall:", kn_recall)
print("F1 Score:", kn_f1)
```

```
Confusion Matrix:
[[1213  754   79]
 [ 704 2498  361]
 [ 143  617  419]]

Accuracy: 0.6084266352386565
Precision: 0.5740854350718929
Recall: 0.5497815428699989
F1 Score: 0.5580861034326624
```

In [49]:
```
from sklearn.model_selection import cross_validate # imports the cross_validate function from scikit-learn.
from sklearn.neighbors import KNeighborsClassifier # imports the KNeighborsClassifier from scikit-learn.

# Assuming 'feed' is your DataFrame and it's already been defined

# Define features and target
X = feed.drop('Credit_Score', axis=1)
y = feed['Credit_Score']

# Initialize KNeighbors Classifier
kn = KNeighborsClassifier()

# Define scoring methods you're interested in
scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

# Perform cross-validation
cv_results = cross_validate(kn, X, y, cv=5, scoring=scoring)

# Display results
print("Cross-validation results:")
for score in scoring:
    print(f"{score}: {cv_results['test_'+score].mean()} ± {cv_results['test_'+score].std()}")
```

```
Cross-validation results:
accuracy: 0.5626307755502282 ± 0.005277274329495184
precision_macro: 0.5178390467064145 ± 0.006970798419478863
recall_macro: 0.5021349830575466 ± 0.005357711512197793
f1_macro: 0.5072771888589445 ± 0.005845186562454469
```

In [39]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression #imports the KNeighborsClassifier from scikit-learn.
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
# Initialize the model
log_reg = LogisticRegression(max_iter=1000)  # Increase max_iter if the model doesn't converge

# Fit the model
log_reg.fit(x_train, y_train)
# Predicting the Test set results
y_pred = log_reg.predict(x_test)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
print("  ")

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Precision, Recall, and F1 Score
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Confusion Matrix:
[[ 805 1182   59]
 [ 487 2875  201]
 [  21  884  274]]

Accuracy: 0.5824985268120212
Precision: 0.569358134924104
Recall: 0.477585089596948
F1 Score: 0.49178943791157526
```

In [50]:
```
from sklearn.model_selection import cross_validate  #inport cross validate from scikit learn
from sklearn.linear_model import LogisticRegression ##imports the LogicticRegression from scikit-learn.

# Assuming x_train, x_test, y_train, and y_test are already defined as part of your dataset

# Initialize Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)  # Adjust max_iter as necessary

# Define scoring metrics
scoring = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro']

# Perform 5-fold cross-validation
cv_results = cross_validate(log_reg, X, y, cv=5, scoring=scoring)

# Display results
print("Cross-validation results:")
for metric in scoring:
    print(f"{metric}: Mean = {cv_results['test_'+metric].mean()}, Standard Deviation = {cv_results['test_'+metric].std()}")
```

```
Cross-validation results:
accuracy: Mean = 0.5777472687524653, Standard Deviation = 0.007790170436948096
precision_macro: Mean = 0.5599026468999495, Standard Deviation = 0.011847646669655322
recall_macro: Mean = 0.4734434413176382, Standard Deviation = 0.011635950495032406
f1_macro: Mean = 0.4877431701688543, Standard Deviation = 0.013765055188555765
```

In [45]:
```
import warnings  ##User Warnings during the execution of code
warnings.simplefilter('ignore', category=UserWarning)
```