# Background Info

ID: Represents a unique identification of an entry

Customer_ID: Represents a unique identification of a person

Name: Represents the name of a person

Month: Represents the month of the year

Age: Represents the age of the person

SSN: Represents the social security number of a person

Occupation: Represents the occupation of the person

Annual_Income: Represents the annual income of the person

Monthly_Base_Salary: Represents the monthly base salary of a person

Num_Bank_Accounts: Represents the number of bank accounts a person holds

Num_Credit_Card: Represents the number of other credit cards held by a person

Interest_Rate: Represents the interest rate on credit card

Num_of_Loan: Represents the number of loans taken from the bank

Type_of_Loan: Represents the types of loan taken by a person

Delay_from_due_date: Represents the average number of days delayed from the payment date

Num_of_delayed_Payment: Represents the average number of payments delayed by a person

Changed_Credit_Limit: Represents the percentage change in credit card limit

Num_Credit_Inquiries: Represents the number of credit card inquiries

Credit_Mix: Represents the classification of the mix of credits

Outstanding_Debt: Represents the remaining debt to be paid (in USD)

Credit_Utilization_Ratio: Represents the utilization ratio of credit card

Credit_History_Age: Represents the age of credit history of the person

Payment_of_Min_Amount: Represents whether only the minimum amount was paid by the person

Total_EMI_per_month: Represents the monthly EMI payments (in USD)

Amount_invested_monthly: Represents the monthly amount invested by the customer (in USD)

Payment_Behaviour: Represents the payment behavior of the customer (in USD)

Monthly_Balance: Represents the monthly balance amount of the customer (in USD)

Credit_Score: Represents the bracket of credit score (Poor, Standard, Good)

# Packages and read in data

In [ ]:

```
# pip install missingno
```

In [2]:

```
##import library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pylab as plt
from scipy import stats
import missingno as msno
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import OneHotEncoder
from mlxtend.plotting import plot_decision_regions
import io
```

In [3]:

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving test.csv to test (1).csv
Saving train.csv to train (1).csv
```

In [4]:

```
#Due to some columns contain more than one data type, there is a warning message, just
df = pd.read_csv(io.StringIO(uploaded['train.csv'].decode('utf-8')))
```

```
<ipython-input-4-293aebc0c709>:2: DtypeWarning: Columns (26) have mixed types. Specif
y dtype option on import or set low_memory=False.
  df = pd.read_csv(io.StringIO(uploaded['train.csv'].decode('utf-8')))
```

In [5]:

```
pd.set_option('display.max_columns', None)
df.iloc[:10,:]
df.head(10)
```

Out[5]:

| | ID | Customer_ID | Month | Name | Age | SSN | Occupation | Annual_Income | Monthl |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0x1602 | CUS_0xd40 | January | Aaron Maashoh | 23 | 821-00-0265 | Scientist | 19114.12 | |
| 1 | 0x1603 | CUS_0xd40 | February | Aaron Maashoh | 23 | 821-00-0265 | Scientist | 19114.12 | |
| 2 | 0x1604 | CUS_0xd40 | March | Aaron Maashoh | -500 | 821-00-0265 | Scientist | 19114.12 | |
| 3 | 0x1605 | CUS_0xd40 | April | Aaron Maashoh | 23 | 821-00-0265 | Scientist | 19114.12 | |
| 4 | 0x1606 | CUS_0xd40 | May | Aaron Maashoh | 23 | 821-00-0265 | Scientist | 19114.12 | |
| 5 | 0x1607 | CUS_0xd40 | June | Aaron Maashoh | 23 | 821-00-0265 | Scientist | 19114.12 | |
| 6 | 0x1608 | CUS_0xd40 | July | Aaron Maashoh | 23 | 821-00-0265 | Scientist | 19114.12 | |
| 7 | 0x1609 | CUS_0xd40 | August | NaN | 23 | #F%$D@*&8 | Scientist | 19114.12 | |
| 8 | 0x160e | CUS_0x21b1 | January | Rick Rothackerj | 28_ | 004-07-5839 | _______ | 34847.84 | |
| 9 | 0x160f | CUS_0x21b1 | February | Rick Rothackerj | 28 | 004-07-5839 | Teacher | 34847.84 | |

In [6]:

```
display(df.describe(exclude=np.number).T)
#to see non-numeric columns remain in the dataframe
df.describe()
```

| | count | unique | top | freq |
|---|---|---|---|---|
| ID | 100000 | 100000 | 0x1602 | 1 |
| Customer_ID | 100000 | 12500 | CUS_0xd40 | 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 | 36585.12 | 16 |
| Num_of_Loan | 100000 | 434 | 3 | 14386 |
| 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 |

Out[6]:

| | Monthly_Inhand_Salary | Num_Bank_Accounts | Num_Credit_Card | Interest_Rate | Delay_from_due_ |
|---|---|---|---|---|---|
| count | 84998.000000 | 100000.000000 | 100000.00000 | 100000.000000 | 100000.00 |
| mean | 4194.170850 | 17.091280 | 22.47443 | 72.466040 | 21.06 |
| std | 3183.686167 | 117.404834 | 129.05741 | 466.422621 | 14.86 |
| min | 303.645417 | -1.000000 | 0.00000 | 1.000000 | -5.00 |
| 25% | 1625.568229 | 3.000000 | 4.00000 | 8.000000 | 10.00 |
| 50% | 3093.745000 | 6.000000 | 5.00000 | 13.000000 | 18.00 |
| 75% | 5957.448333 | 7.000000 | 7.00000 | 20.000000 | 28.00 |
| max | 15204.633333 | 1798.000000 | 1499.00000 | 5797.000000 | 67.00 |

In [7]:

```
df['ID'].count(), df['Name'].count(), df['Name'].isna().sum()
# The difference between count and unique is num of NA in the column
# We also find that there are outliers. Next step, we will first drop NA and remove ou
```

Out[7]: (100000, 90015, 9985)

In [6]:

# EDA

In [8]:
```
df1=df.copy() #df1, we are going to change all values into null, and replace it
```

In [9]:
```
#first we deal with strange value in categorical columns
categorical_cols = [c for c in df.columns if df[c].dtype == 'object']
for i in categorical_cols:
    print(f'Unique Values of {i} is {df[i].unique()}')
```

```
Unique Values of ID is ['0x1602' '0x1603' '0x1604' ... '0x25feb' '0x25fec' '0x25fed']
Unique Values of Customer_ID is ['CUS_0xd40' 'CUS_0x21b1' 'CUS_0x2dbc' ... 'CUS_0xaf6
1' 'CUS_0x8600'
 'CUS_0x942c']
Unique Values of Month is ['January' 'February' 'March' 'April' 'May' 'June' 'July'
'August']
Unique Values of Name is ['Aaron Maashoh' nan 'Rick Rothackerj' ... 'Chris Wickhamm'
 'Sarah McBridec' 'Nicks']
Unique Values of Age is ['23' '-500' '28_' ... '4808_' '2263' '1342']
Unique Values of SSN is ['821-00-0265' '#F%$D@*&8' '004-07-5839' ... '133-16-7738' '0
31-35-0942'
 '078-73-5990']
Unique Values of Occupation is ['Scientist' '_______' 'Teacher' 'Engineer' 'Entrepren
eur' 'Developer'
 'Lawyer' 'Media_Manager' 'Doctor' 'Journalist' 'Manager' 'Accountant'
 'Musician' 'Mechanic' 'Writer' 'Architect']
Unique Values of Annual_Income is ['19114.12' '34847.84' '34847.84_' ... '20002.88'
'39628.99' '39628.99_']
Unique Values of Num_of_Loan is ['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']
Unique Values of Type_of_Loan is ['Auto Loan, Credit-Builder Loan, Personal Loan, and
Home Equity Loan'
```

```
 'Credit-Builder Loan' 'Auto Loan, Auto Loan, and Not Specified' ...
 'Home Equity Loan, Auto Loan, Auto Loan, and Auto Loan'
 'Payday Loan, Student Loan, Mortgage Loan, and Not Specified'
 'Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan']
Unique Values of Num_of_Delayed_Payment is ['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']
Unique Values of Changed_Credit_Limit is ['11.27' '_' '6.27' ... '17.509999999999998'
'25.16' '21.17']
Unique Values of Credit_Mix is ['_' 'Good' 'Standard' 'Bad']
Unique Values of Outstanding_Debt is ['809.98' '605.03' '1303.01' ... '3571.7_' '357
1.7' '502.38']
Unique Values of Credit_History_Age is ['22 Years and 1 Months' nan '22 Years and 3 M
onths'
 '22 Years and 4 Months' '22 Years and 5 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 8 Months' '30 Years and 9 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'
 '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 10 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'
 '26 Years and 0 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 1 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 9 Months' '6 Years and 10 Months' '6 Years and 11 Months'
 '7 Years and 0 Months' '27 Years and 6 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 1 Months'
```

'10 Years and 2 Months' '10 Years and 3 Months' '10 Years and 4 Months'
'10 Years and 5 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'
'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 6 Months' '13 Years and 7 Months' '22 Years and 0 Months'
'26 Years and 1 Months' '26 Years and 2 Months' '13 Years and 4 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 4 Months' '24 Years and 5 Months' '24 Years and 6 Months'
'24 Years and 7 Months' '24 Years and 8 Months' '24 Years and 9 Months'
'1 Years and 2 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' '10 Years and 11 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'
'19 Years and 11 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 8 Months'
'20 Years and 9 Months' '20 Years and 10 Months' '20 Years and 11 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 8 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 10 Months' '31 Years and 11 Months'
'32 Years and 1 Months' '12 Years and 11 Months' '13 Years and 0 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' '18 Years and 0 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' '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 0 Months'
'23 Years and 2 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' '7 Years and 6 Months'
 '7 Years and 7 Months' '7 Years and 8 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'
 '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'
 '28 Years and 11 Months' '29 Years and 0 Months' '13 Years and 10 Months'
 '1 Years and 9 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' '5 Years and 1 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 9 Months' '22 Years and 10 Months' '23 Years and 1 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' '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' '24 Years and 2 Months' '16 Years and 0 Months'
 '16 Years and 1 Months' '14 Years and 7 Months' '25 Years and 4 Months'
 '15 Years and 3 Months' '7 Years and 1 Months' '7 Years and 2 Months'
 '7 Years and 3 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'
 '30 Years and 1 Months' '29 Years and 10 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 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'
 '3 Years and 8 Months' '32 Years and 7 Months' '3 Years and 7 Months'
 '3 Years and 9 Months' '3 Years and 10 Months' '0 Years and 11 Months'
 '1 Years and 0 Months' '1 Years and 1 Months' '4 Years and 4 Months'
 '3 Years and 11 Months' '4 Years and 0 Months' '4 Years and 1 Months'
 '4 Years and 2 Months' '4 Years and 3 Months' '2 Years and 1 Months']
Unique Values of Payment_of_Min_Amount is ['No' 'NM' 'Yes']
Unique Values of Amount_invested_monthly is ['80.41529543900253' '118.28022162236736'
'81.699521264648' ...
 '24.02847744864441' '251.67258219721603' '167.1638651610451']
Unique Values of Payment_Behaviour is ['High_spent_Small_value_payments' 'Low_spent_L
arge_value_payments'
 'Low_spent_Medium_value_payments' 'Low_spent_Small_value_payments'
 'High_spent_Medium_value_payments' '!@9#%8'
 'High_spent_Large_value_payments']
Unique Values of Monthly_Balance is ['312.49408867943663' '284.62916249607184' '331.2
098628537912' ...
 516.8090832742814 319.1649785257098 393.6736955618808]
Unique Values of Credit_Score is ['Good' 'Standard' 'Poor']
```

In [10]:

```
#there are many numeric value followed by '_', let's remove it
for i in categorical_cols:
    df1[i] = df1[i].str.strip('_')
    df1[i] = df1[i].replace({'':np.nan})
    try:
        df1[i] = df1[i].astype('float64')
    except:
        df1[i] = df1[i]
```

```python
for col in categorical_cols:
    df1[col] = df1[col].replace({'!@9#%8':np.nan, '#F%$D@*&8':np.nan})
```

In [11]:
```python
display(df1.describe(exclude=np.number).T)
#to recheck non-numeric columns remain in the dataframe
#strange value is cleaned and seems successful
#as reminder, Credit_History_Age can change into float by re
```

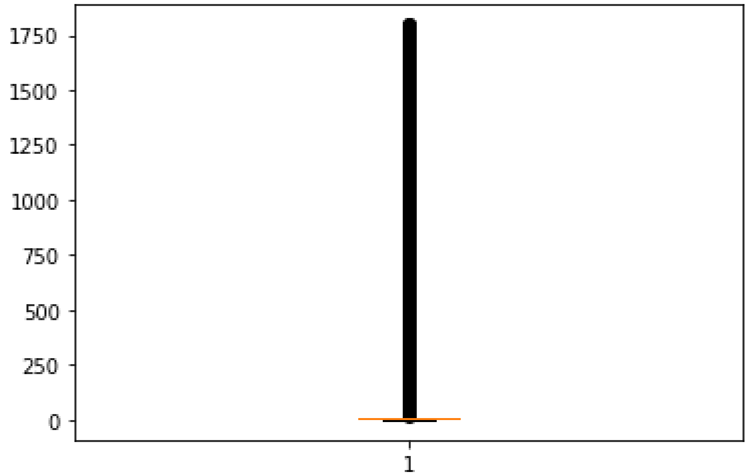
| | count | unique | top | freq |
|---|---|---|---|---|
| **ID** | 100000 | 100000 | 0x1602 | 1 |
| **Customer_ID** | 100000 | 12500 | CUS_0xd40 | 8 |
| **Month** | 100000 | 8 | January | 12500 |
| **Name** | 90015 | 10139 | Langep | 44 |
| **SSN** | 94428 | 12500 | 078-73-5990 | 8 |
| **Occupation** | 92938 | 15 | Lawyer | 6575 |
| **Type_of_Loan** | 88592 | 6260 | Not Specified | 1408 |
| **Credit_Mix** | 79805 | 3 | Standard | 36479 |
| **Credit_History_Age** | 90970 | 404 | 15 Years and 11 Months | 446 |
| **Payment_of_Min_Amount** | 100000 | 3 | Yes | 52326 |
| **Payment_Behaviour** | 92400 | 6 | Low_spent_Small_value_payments | 25513 |
| **Credit_Score** | 100000 | 3 | Standard | 53174 |

In [12]:
```python
#Recall from describe, after we deal with strange values in categorical columns, outli
#to prevent data loss, we want outliers to be replaced by same value from same users
dict1 = pd.Series(df1['Num_Bank_Accounts'].values,index=df1['ID']).to_dict()
#However, dictionary is not the possible solution. There are 1315 users who own bank a
df['Num_Bank_Accounts'].quantile(.95) # =10

a  = 0
for i in dict1.values():
    if i>11:
        a = a+1
a
```

Out[12]: 1315

In [13]:
```python
plt.boxplot(df1['Num_Bank_Accounts'])
df1['Num_Bank_Accounts'].quantile(.99), df1['Num_Bank_Accounts'].quantile(.95)
```

Out[13]: (445.00999999999476, 10.0)



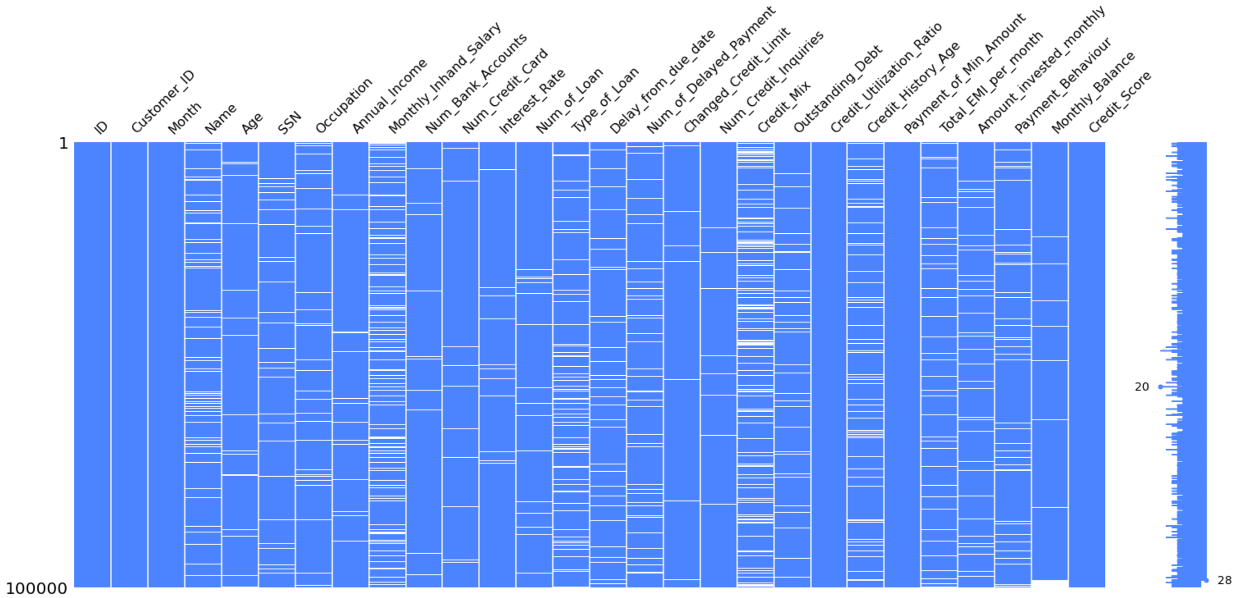

```
In [14]: #We replace outlier to nas
         numerical_cols = [col for col in df1.columns if (df1[col].dtype == 'int64') | (df1[col

         # for x in list(numerical_cols):
         #     q_low = df1[x].quantile(0.05)
         #     q_hi  = df1[x].quantile(0.95)

         #     df1.loc[df1[x] < q_low,x] = np.nan
         #     df1.loc[df1[x] > q_hi,x] = np.nan


         for x in list(numerical_cols):
             q75,q25 = np.percentile(df1.loc[:,x],[75,25])
             intr_qr = q75-q25

             max = q75+(1.5*intr_qr)
             min = q25-(1.5*intr_qr)

             df1.loc[df1[x] < min,x] = np.nan
             df1.loc[df1[x] > max,x] = np.nan
```

```
In [15]: numerical_cols
```

```
Out[15]: ['Age',
          'Annual_Income',
          'Monthly_Inhand_Salary',
          'Num_Bank_Accounts',
          'Num_Credit_Card',
          'Interest_Rate',
          'Num_of_Loan',
          'Delay_from_due_date',
          'Num_of_Delayed_Payment',
          'Changed_Credit_Limit',
          'Num_Credit_Inquiries',
          'Outstanding_Debt',
          'Credit_Utilization_Ratio',
          'Total_EMI_per_month',
          'Amount_invested_monthly',
          'Monthly_Balance']
```

```
In [16]: #NA can see in graph as white line
         msno.matrix(df1, color=(0.30, 0.52, 1.0))
```

Out[16]: <Axes: >




In [17]:
```
#as a short intermediate summary for data cleaning, let's see number of nas in each co
miss =df1.isnull().sum()
miss =miss[miss>0]
miss
#you can see, if we just drop all outlier rows, we will loose a lot of informations
#thus, we choose to replace value instead of drop it
#instead of drop the row, replace the outlier from upper row or median are better choi
```

Out[17]:
```
Name                        9985
Age                         2781
SSN                         5572
Occupation                  7062
Annual_Income               2783
Monthly_Inhand_Salary      15002
Num_Bank_Accounts           1315
Num_Credit_Card             2271
Interest_Rate               2034
Num_of_Loan                 4348
Type_of_Loan               11408
Delay_from_due_date         4002
Num_of_Delayed_Payment      7002
Changed_Credit_Limit        2091
Num_Credit_Inquiries        1965
Credit_Mix                 20195
Outstanding_Debt            5272
Credit_Utilization_Ratio       4
Credit_History_Age          9030
Total_EMI_per_month         6795
Amount_invested_monthly     4479
Payment_Behaviour           7600
Monthly_Balance             2868
dtype: int64
```

In [18]:
```
#we can replace nan based on same consumter ID
#display the first 10 rows, we can find
#ID is representation of unique row, but cannot be representation of unique costumer,
# we decide to drop Type_of_Loan, due to complexity
df1.drop(columns=['ID','Name','SSN','Type_of_Loan'], axis=1, inplace=True)
df1.head(10)
```

```
#Among first eight records, they come from same person.
#Nans in Monthly_Inhand_Salary of first person can be replaced
```

Out[18]:

| | Customer_ID | Month | Age | Occupation | Annual_Income | Monthly_Inhand_Salary | Num_Bank_Accou |
|---|---|---|---|---|---|---|---|
| **0** | CUS_0xd40 | January | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **1** | CUS_0xd40 | February | 23.0 | Scientist | 19114.12 | NaN | |
| **2** | CUS_0xd40 | March | NaN | Scientist | 19114.12 | NaN | |
| **3** | CUS_0xd40 | April | 23.0 | Scientist | 19114.12 | NaN | |
| **4** | CUS_0xd40 | May | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **5** | CUS_0xd40 | June | 23.0 | Scientist | 19114.12 | NaN | |
| **6** | CUS_0xd40 | July | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **7** | CUS_0xd40 | August | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **8** | CUS_0x21b1 | January | 28.0 | NaN | 34847.84 | 3037.986667 | |
| **9** | CUS_0x21b1 | February | 28.0 | Teacher | 34847.84 | 3037.986667 | |

In [19]:

```
#we cand fill in based on same Customer_ID, it's also unique, means this person always
numerical_df1 = {col for col in df1.columns if (df1[col].dtype=='int64') | (df1[col].c
categorical_df1 = {col for col in df1.columns if (df1[col].dtype=='object')}

unique_based = [col for col in numerical_df1 if df1[col].head(8).nunique() == 1] #fill
df1[unique_based] = df1[unique_based].fillna(method='bfill')
for col in unique_based:
    df1[col]=df1[col].fillna(method='bfill')

non_unique_based = [col for col in numerical_df1 if df1[col].head(8).nunique() != 1] #
df1[non_unique_based] = df1.groupby(by=['Customer_ID'])[non_unique_based].transform('m
for col in non_unique_based:
    df1[col]=df1.groupby(by=['Customer_ID'])[col].transform('median')
```

In [20]:

```
#do same fill by back for categorical columns
#notice this time "Credit_History_Age" is skipped
missing_cols = {col for col in miss.index}
categorical_miss=[col for col in categorical_df1.intersection(missing_cols)]
categorical_miss=['Payment_Behaviour','Credit_Mix','Occupation']
df1[categorical_miss] = df1[categorical_miss].fillna(method='bfill')
df1[categorical_miss] = df1[categorical_miss].fillna(method='ffill')
```

In [21]:

```
#as summary to df1, compare to original df, we drop four columns 'ID','Name','SSN','Ty
df_2 = df1.copy()
df_2.shape
```

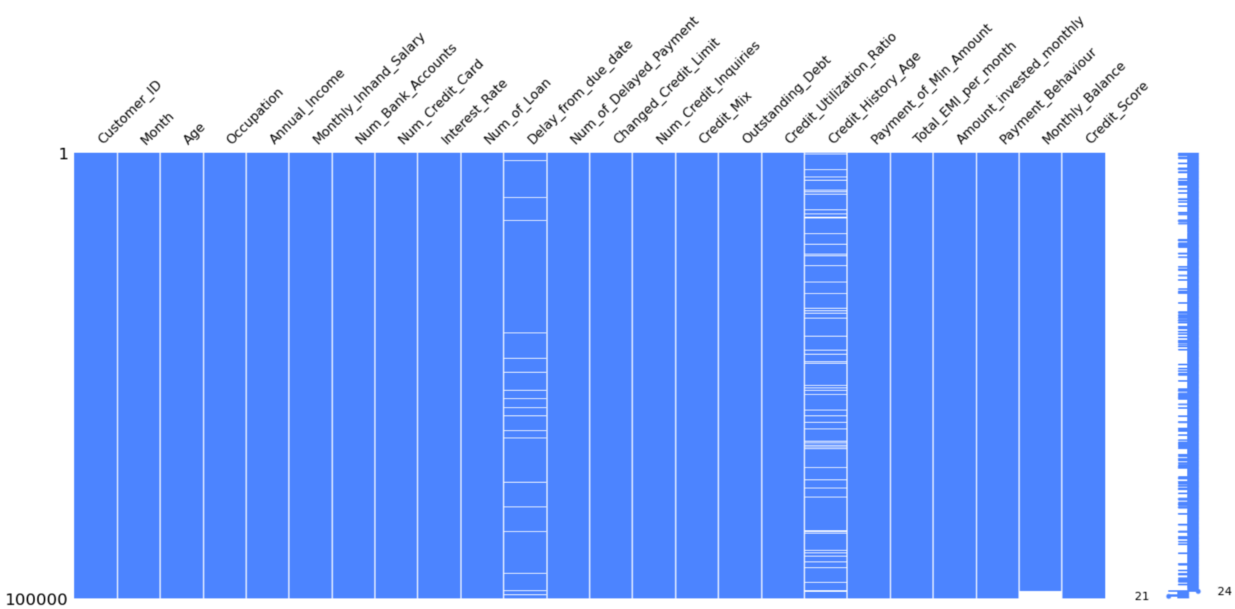
Out[21]: (100000, 24)

In [22]:
```python
df_2.isna().sum()
```

Out[22]:
```
Customer_ID                    0
Month                          0
Age                            0
Occupation                     0
Annual_Income                  0
Monthly_Inhand_Salary          0
Num_Bank_Accounts              0
Num_Credit_Card                0
Interest_Rate                  0
Num_of_Loan                    0
Delay_from_due_date         2656
Num_of_Delayed_Payment         0
Changed_Credit_Limit           0
Num_Credit_Inquiries           0
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        0
Payment_Behaviour              0
Monthly_Balance             1696
Credit_Score                   0
dtype: int64
```

In [23]:
```python
msno.matrix(df_2, color=(0.30, 0.52, 1.0))
```

Out[23]: <Axes: >




In [24]:
```python
#as summary to df2, compare to df1, we drop all strange value(now na) in those numeric
df_c=df_2.dropna()
df_c.shape
```

Out[24]: (87049, 24)

In [25]:
```
df_c.isna().sum()
df_c.dropna()
df_c['Credit_Mix'].value_counts() #???
```

Out[25]:
```
Standard    40583
Good        27187
Bad         19279
Name: Credit_Mix, dtype: int64
```

In [26]:
```
#Next step, we transform text columns
#replace with order numbers
df_clean=df_c.copy()
df_clean['Credit_Score'] = df_clean['Credit_Score'].map({'Poor':1, 'Standard':2, 'Gooc
```

In [27]:
```
#Transfrom Month to float
df_clean['Month'] = df_clean['Month'].map({'January':1, 'February':2, 'March':3, 'Apri
                                           'June':6, 'July':7, 'August':8, 'September'
                                            'November':11, 'December':12})
```

In [28]:
```
df_clean['Credit_Mix'].value_counts() #??? should use credit score
```

Out[28]:
```
Standard    40583
Good        27187
Bad         19279
Name: Credit_Mix, dtype: int64
```

In [29]:
```
df_clean['Payment_of_Min_Amount'] = df_clean['Payment_of_Min_Amount'].map({'No':0,'Yes
df_clean['Credit_Mix'] = df_clean['Credit_Mix'].map({'Bad':0,'Standard':1,'Good':2})
```

In [30]:
```
df_clean.isna().sum()
df_clean
```

Out[30]:

| | Customer_ID | Month | Age | Occupation | Annual_Income | Monthly_Inhand_Salary | Num_Bank_Ac |
|---|---|---|---|---|---|---|---|
| **0** | CUS_0xd40 | 1 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **2** | CUS_0xd40 | 3 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **3** | CUS_0xd40 | 4 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **4** | CUS_0xd40 | 5 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **5** | CUS_0xd40 | 6 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **98298** | CUS_0x9d41 | 3 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98299** | CUS_0x9d41 | 4 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98300** | CUS_0x9d41 | 5 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98302** | CUS_0x9d41 | 7 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98303** | CUS_0x9d41 | 8 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |

87049 rows × 24 columns

In [31]:

```
#Transform the Credit_History_Age to float in unit of years
df_clean['Credit_History_Age'] = df_clean['Credit_History_Age'].astype(str).str.replac
df_clean['Credit_History_Age'] = df_clean['Credit_History_Age'].astype(str).str.replac
df_clean.head()

def ym(x):
    Y=float(x.split('.')[0])
    M=float(x.split('.')[1])
    return round(Y+M/12, 2)
df_clean.Credit_History_Age=df_clean.Credit_History_Age.apply(lambda x :ym(x))
```

In [32]:

```
df_clean.isna().sum()
```

Out[32]:
```
Customer_ID                 0
Month                       0
Age                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       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
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          0
Payment_of_Min_Amount       0
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_Score                0
dtype: int64
```

In [33]:
```
df_clean.head(10)
df_clean.reset_index(drop=True)
```

Out[33]:

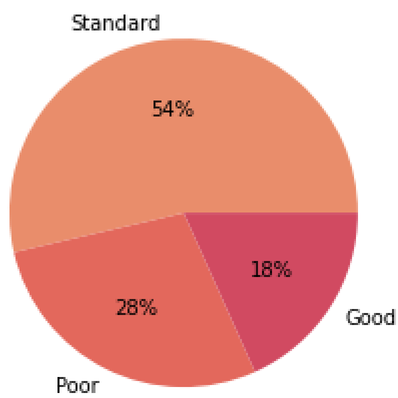
| | Customer_ID | Month | Age | Occupation | Annual_Income | Monthly_Inhand_Salary | Num_Bank_Ac |
|---|---|---|---|---|---|---|---|
| **0** | CUS_0xd40 | 1 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **1** | CUS_0xd40 | 3 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **2** | CUS_0xd40 | 4 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **3** | CUS_0xd40 | 5 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **4** | CUS_0xd40 | 6 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **87044** | CUS_0x9d41 | 3 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **87045** | CUS_0x9d41 | 4 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **87046** | CUS_0x9d41 | 5 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **87047** | CUS_0x9d41 | 7 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **87048** | CUS_0x9d41 | 8 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |

87049 rows × 24 columns

# Visualization

In [34]:

```
#pie chart of our response: Credit_Score
data = df_c.Credit_Score.value_counts()
keys = ['Standard', 'Poor', 'Good']
palette_color = sns.color_palette("flare")
plt.pie(data, labels=keys, colors=palette_color, autopct='%.0f%%')
plt.show()
```




In [35]:

```
#Some setups
columns=[col for col in df_clean.columns]
for col in columns:
    print('========================')
    print(df_clean[col].value_counts())
```

```
========================
CUS_0x61c6    8
CUS_0x56d6    8
CUS_0x90d8    8
CUS_0x4adc    8
CUS_0x9319    8
             ..
CUS_0xb163    4
CUS_0x8988    4
CUS_0x235a    4
CUS_0x38c0    3
CUS_0x489     3
Name: Customer_ID, Length: 11960, dtype: int64
========================
2    10905
1    10897
8    10887
5    10881
6    10880
3    10879
7    10872
4    10848
Name: Month, dtype: int64
========================
26.0    2703
31.0    2655
38.0    2625
28.0    2621
36.0    2561
41.0    2558
25.0    2548
32.0    2542
19.0    2516
35.0    2515
22.0    2514
29.0    2514
27.0    2514
34.0    2496
44.0    2478
20.0    2477
39.0    2476
30.0    2473
37.0    2438
43.0    2437
24.0    2417
21.0    2405
23.0    2363
45.0    2333
33.0    2310
42.0    2290
40.0    2263
18.0    2098
46.0    1500
15.0    1413
17.0    1297
16.0    1275
55.0    1274
48.0    1274
52.0    1259
53.0    1258
```

```
49.0    1241
54.0    1219
51.0    1198
50.0    1175
47.0    1128
14.0    1062
56.0     336
Name: Age, dtype: int64
========================
Lawyer           6106
Architect        5956
Scientist        5933
Mechanic         5893
Engineer         5888
Teacher          5850
Media_Manager    5839
Entrepreneur     5826
Developer        5818
Accountant       5815
Doctor           5789
Journalist       5778
Manager          5558
Musician         5514
Writer           5486
Name: Occupation, dtype: int64
========================
31323.88    23
57079.50    23
72458.44    16
38087.00    16
15359.33    16
            ..
18719.74     3
9388.27      3
15660.15     3
39641.54     1
74346.44     1
Name: Annual_Income, Length: 11734, dtype: int64
========================
6769.130000    16
2295.058333    15
6358.956667    15
6082.187500    13
4387.272500    13
               ..
3990.212538     1
9664.622500     1
4962.540000     1
817.769538      1
7237.397300     1
Name: Monthly_Inhand_Salary, Length: 12689, dtype: int64
========================
 6.0     11230
 8.0     11159
 7.0     11133
 5.0     10988
 4.0     10982
 3.0     10865
 9.0      4474
 10.0     4323
```

```
 1.0      4064
 0.0      3929
 2.0      3877
-1.0        20
 11.0        5
Name: Num_Bank_Accounts, dtype: int64
========================
5.0     16592
6.0     14776
7.0     14679
4.0     12805
3.0     12208
8.0      4132
10.0     4078
9.0      3834
2.0      1968
1.0      1942
11.0       22
0.0        13
Name: Num_Credit_Card, dtype: int64
========================
8.0     4569
5.0     4534
6.0     4263
12.0    4180
10.0    4102
7.0     4101
9.0     4090
11.0    4040
18.0    3596
15.0    3587
20.0    3467
17.0    3388
16.0    3286
19.0    3238
3.0     2536
1.0     2472
4.0     2389
2.0     2268
13.0    2154
14.0    2015
32.0    1458
23.0    1447
24.0    1440
22.0    1430
27.0    1395
30.0    1365
28.0    1361
25.0    1354
21.0    1332
29.0    1323
34.0    1266
33.0    1214
26.0    1195
31.0    1194
Name: Interest_Rate, dtype: int64
========================
3.0    13937
2.0    13864
4.0    13699
```

```
0.0    10223
1.0     9879
6.0     6827
7.0     6460
5.0     6333
9.0     3123
8.0     2704
Name: Num_of_Loan, dtype: int64
========================
 15.0    3376
 13.0    3156
 8.0     3146
 14.0    3035
 7.0     2958
         ...
 36.5       7
 34.5       7
 31.5       7
-1.5        7
-2.0        6
Name: Delay_from_due_date, Length: 109, dtype: int64
========================
 10.0    5254
 19.0    5154
 16.0    5144
 8.0     5088
 20.0    5087
 15.0    5009
 12.0    4873
 9.0     4867
 17.0    4772
 18.0    4728
 11.0    4591
 14.0    3409
 13.0    3270
 6.0     1863
 2.0     1858
 0.0     1826
 5.0     1822
 21.0    1773
 3.0     1757
 25.0    1748
 1.0     1735
 23.0    1691
 24.0    1648
 4.0     1634
 7.0     1605
 22.0    1586
 18.5     320
 10.5     235
 9.5      216
 11.5     212
 8.5      203
 16.5     203
 19.5     183
 17.5     163
 15.5     162
 20.5     146
 7.5      142
 14.5     129
```

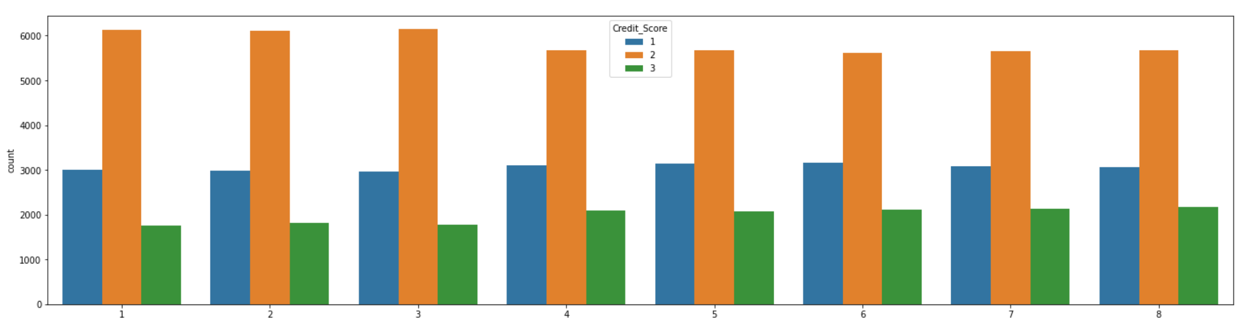
```
 12.5     127
 2.5       86
 13.5      82
 1.5       75
 3.5       74
 6.5       72
 5.5       57
 21.5      50
 23.5      49
 22.5      49
 4.5       45
 0.5       45
 25.5      42
 24.5      24
-0.5       23
-1.0       20
 26.5       8
 26.0       8
-1.5        7
Name: Num_of_Delayed_Payment, dtype: int64
========================
8.22     136
11.32    131
11.50    127
10.06    123
7.35     122
        ...
27.99      5
29.60      5
16.39      5
29.31      4
20.31      4
Name: Changed_Credit_Limit, Length: 2494, dtype: int64
========================
4.0       10281
3.0        8135
2.0        7283
6.0        7161
7.0        7112
          ...
1362.0        1
2368.0        1
675.0         1
2069.0        1
2274.0        1
Name: Num_Credit_Inquiries, Length: 1100, dtype: int64
========================
1    40583
2    27187
0    19279
Name: Credit_Mix, dtype: int64
========================
1794.71    30
1457.54    30
759.86     29
1360.45    29
3838.00    24
           ..
789.47      4
117.63      4
```

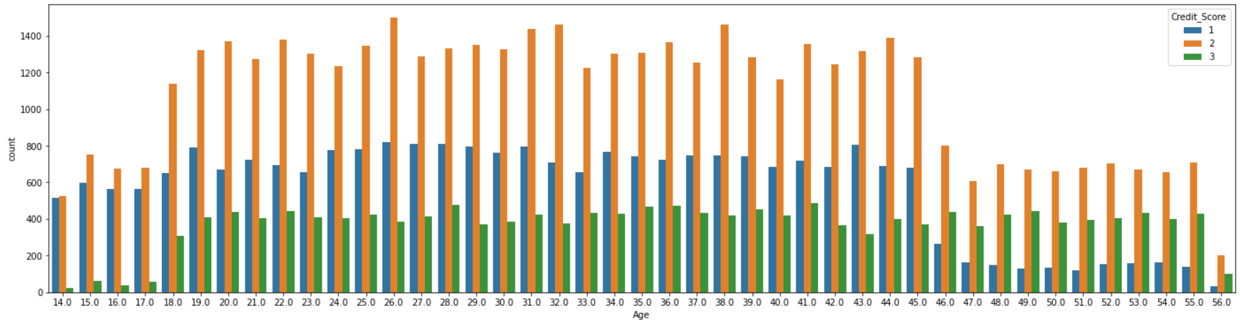
```
586.89      4
2673.96     3
1354.33     3
Name: Outstanding_Debt, Length: 11108, dtype: int64
========================
36.567317    8
32.669809    8
31.064051    8
32.766879    8
35.035083    8
            ..
36.742579    4
30.421941    4
30.931976    4
35.642169    3
32.394336    3
Name: Credit_Utilization_Ratio, Length: 11960, dtype: int64
========================
17.92    431
15.92    430
19.42    428
17.75    427
19.33    425
        ...
0.25      17
33.58     14
0.17      13
33.67     12
0.08       2
Name: Credit_History_Age, Length: 404, dtype: int64
========================
1    44718
0    42331
Name: Payment_of_Min_Amount, dtype: int64
========================
0.000000      10157
42.688001        24
265.400552       22
158.044048       21
103.456213       20
              ...
165.490401        1
73.765767         1
203.355777        1
267.644093        1
53.999103         1
Name: Total_EMI_per_month, Length: 10691, dtype: int64
========================
529.409251    8
100.969530    8
223.263554    8
148.791526    8
232.939679    8
             ..
254.028815    4
88.172905     4
138.319269    4
98.464011     3
44.834839     3
Name: Amount_invested_monthly, Length: 11960, dtype: int64
```

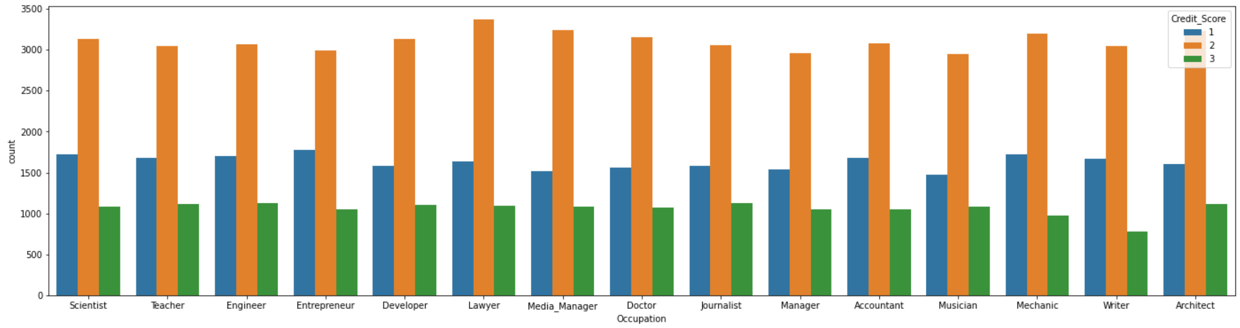
```
========================
Low_spent_Small_value_payments      23874
High_spent_Medium_value_payments    16543
Low_spent_Medium_value_payments     13089
High_spent_Large_value_payments     13041
High_spent_Small_value_payments     10714
Low_spent_Large_value_payments       9788
Name: Payment_Behaviour, dtype: int64
========================
591.789398    8
303.559275    8
219.610157    8
161.618939    8
389.053909    8
             ..
577.812382    4
318.570048    4
397.971226    4
270.546560    3
298.060814    3
Name: Monthly_Balance, Length: 11960, dtype: int64
========================
2    46671
1    24457
3    15921
Name: Credit_Score, dtype: int64
```

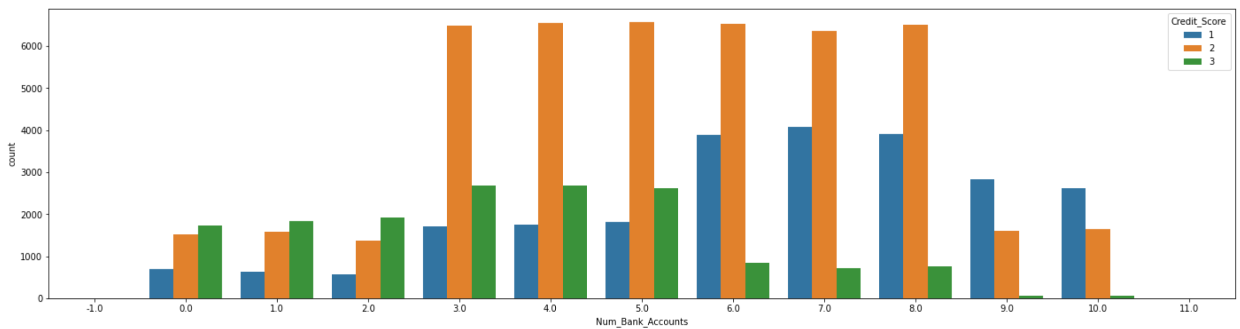
In [36]:
```
##classify the columns according to number of unique values
col_classified=[]
col_left=[]
for col in columns[1:]:
    if len(df_clean[col].value_counts())<100:
        col_classified.append(col)
    else:
        col_left.append(col)
```

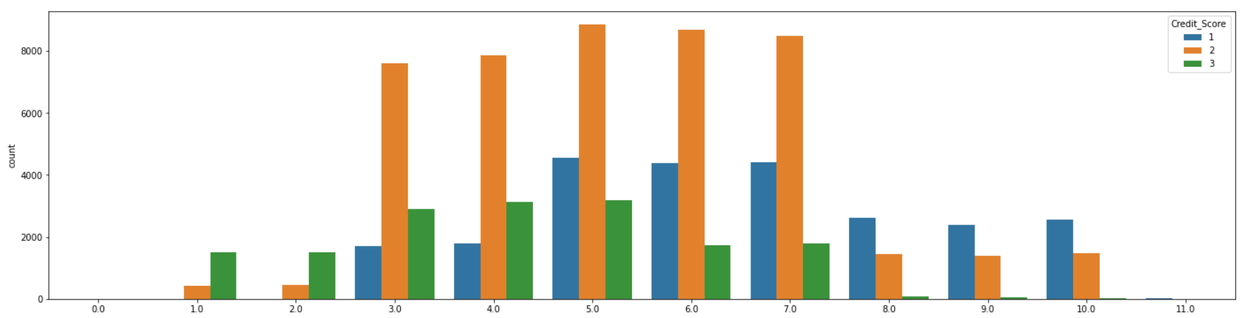
In [37]:
```
## Simple realtionship between different columns and Credit Scores
#Object Columns

for col in col_classified:
    plt.figure(figsize=(24,6))
    sns.countplot(x=col,data=df_clean, hue="Credit_Score")
    plt.show()
```

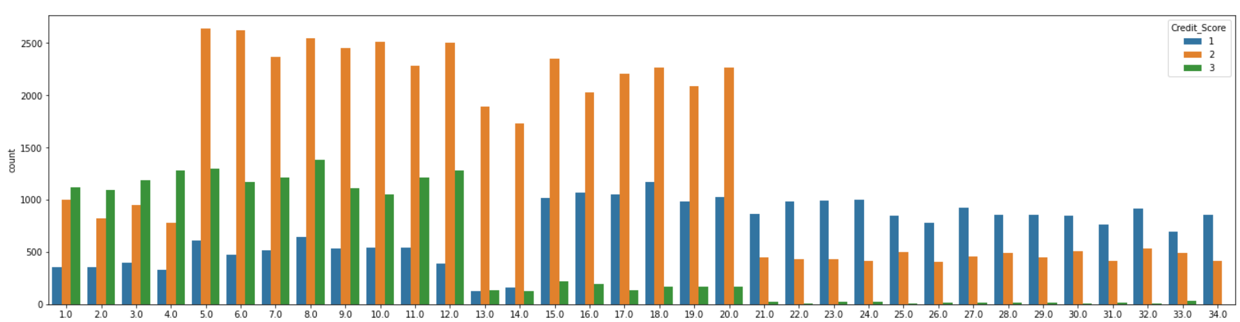


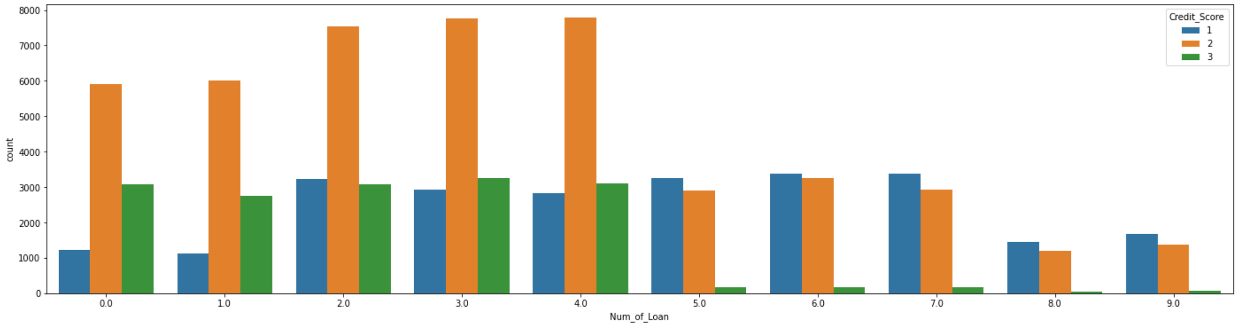

Credit_Score
1
2
3
count
0 200 400 600 800 1000 1200 1400
14.0 15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0 25.0 26.0 27.0 28.0 29.0 30.0 31.0 32.0 33.0 34.0 35.0 36.0 37.0 38.0 39.0 40.0 41.0 42.0 43.0 44.0 45.0 46.0 47.0 48.0 49.0 50.0 51.0 52.0 53.0 54.0 55.0 56.0
Age

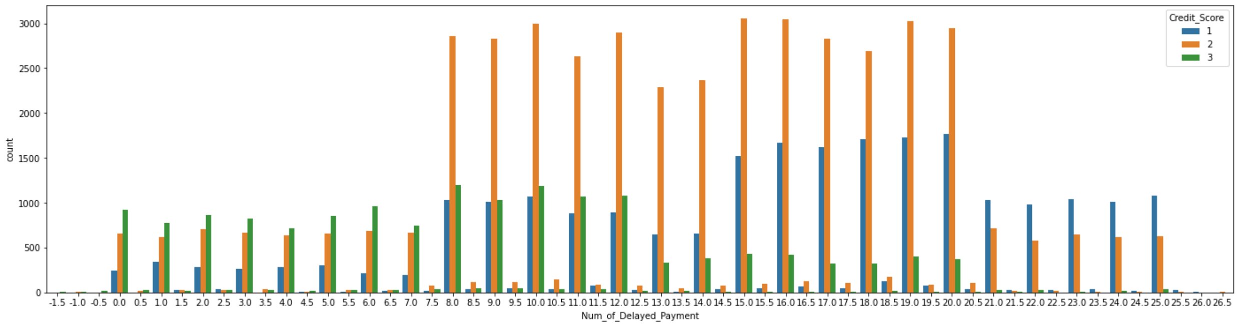

Credit_Score
1
2
3
count
0 500 1000 1500 2000 2500 3000 3500
Scientist Teacher Engineer Entrepreneur Developer Lawyer Media_Manager Doctor Journalist Manager Accountant Musician Mechanic Writer Architect
Occupation

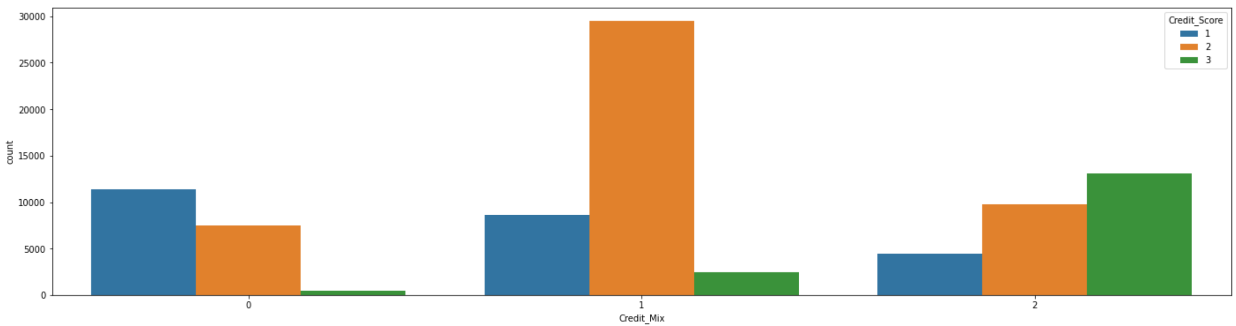

Credit_Score
1
2
3
count
0 1000 2000 3000 4000 5000 6000
-1.0 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0
Num_Bank_Accounts

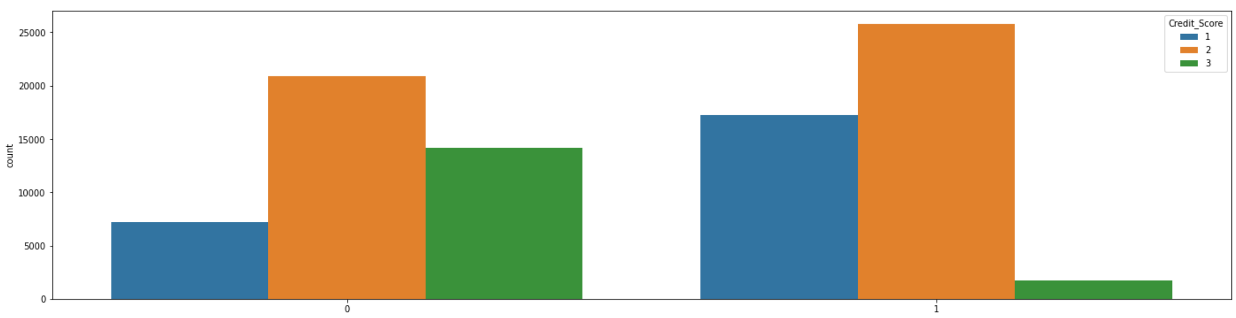

Credit_Score
1
2
3
count
0 2000 4000 6000 8000
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0
Num_Credit_Card

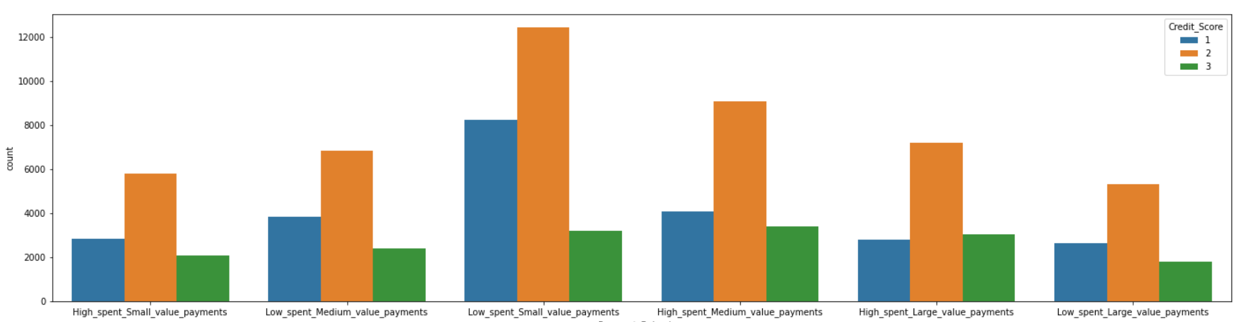

Credit_Score
1
2
3
count
0 500 1000 1500 2000 2500
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 13.0 14.0 15.0 16.0 17.0 18.0 19.0 20.0 21.0 22.0 23.0 24.0 25.0 26.0 27.0 28.0 29.0 30.0 31.0 32.0 33.0 34.0
Interest_Rate

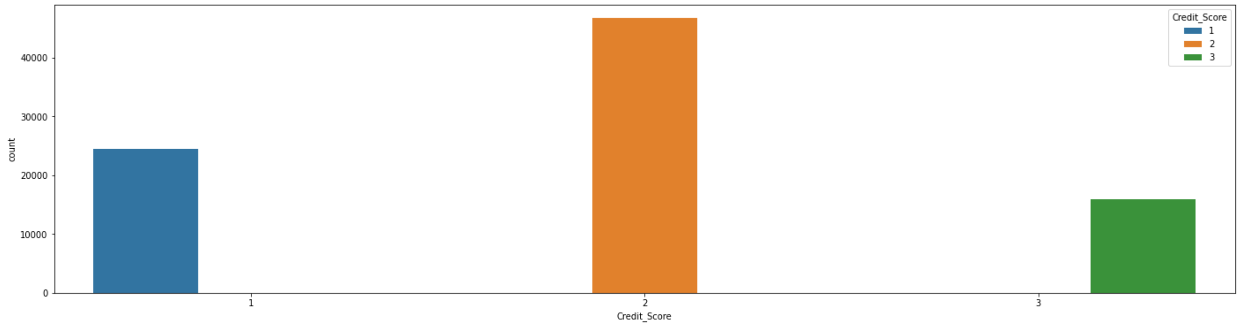

Credit_Score
1
2
3
count
0 1000 2000 3000 4000 5000 6000 7000 8000
0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
Num_of_Loan


Credit_Score
1
2
3
count
0 500 1000 1500 2000 2500 3000
-1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0 24.5 25.0 25.5 26.0 26.5
Num_of_Delayed_Payment


Credit_Score
1
2
3
count
0 5000 10000 15000 20000 25000 30000
0 1 2
Credit_Mix


Credit_Score
1
2
3
count
0 5000 10000 15000 20000 25000
0 1
Payment_of_Min_Amount


Credit_Score
1
2
3
count
0 2000 4000 6000 8000 10000 12000
High_spent_Small_value_payments
Low_spent_Medium_value_payments
Low_spent_Small_value_payments
High_spent_Medium_value_payments
High_spent_Large_value_payments
Low_spent_Large_value_payments
Payment_Behaviour



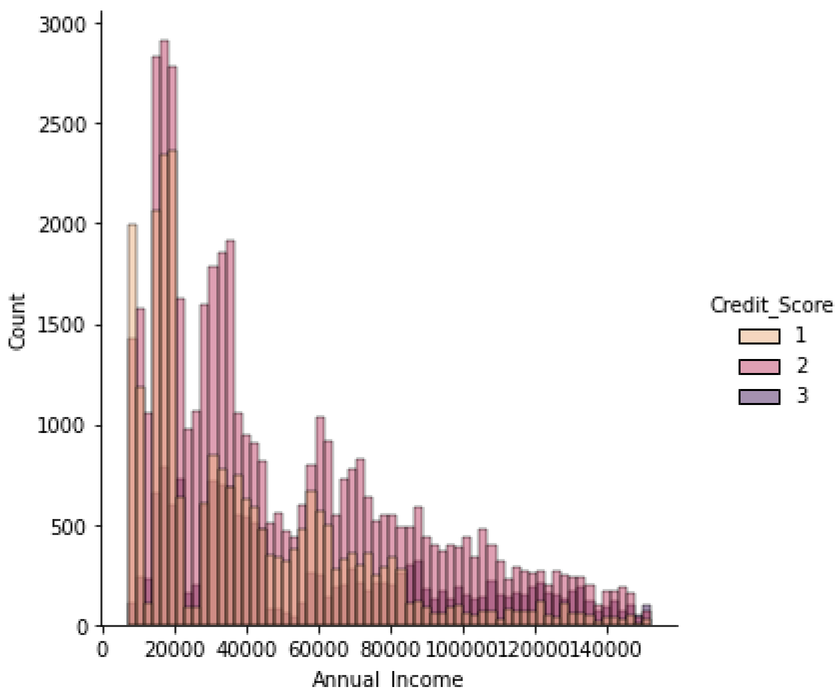

In [ ]:

```
###From barchat above, different Month and Occupation show little relevance to the Cre
```

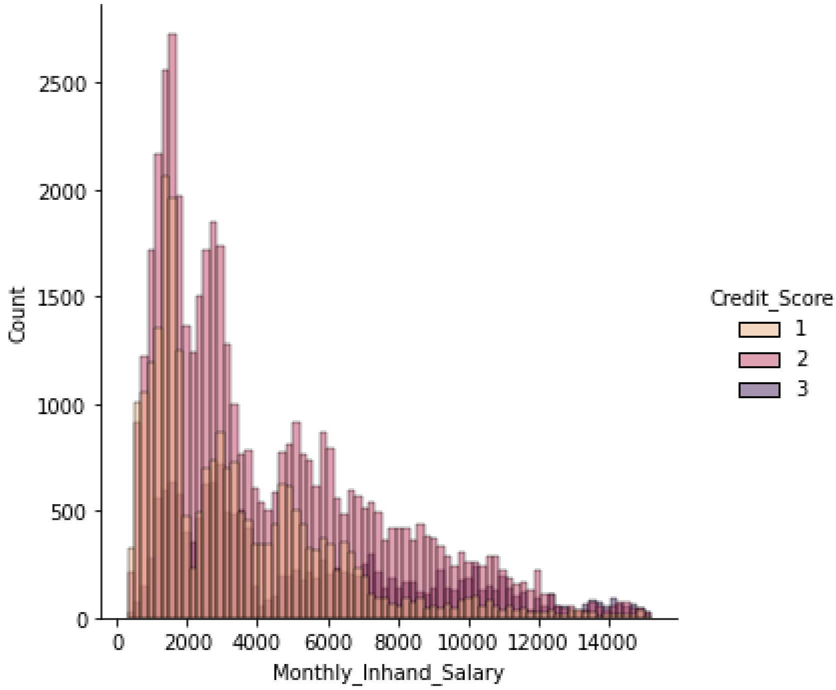
In [38]:

```
#Notice, due to max limits of jupyter,
#['Monthly_Balance' , 'Amount_invested_monthly' and' Num_of_Delayed_Payment']
#These three columns are not presenting here.

for col in['Annual_Income', 'Monthly_Inhand_Salary', 'Changed_Credit_Limit', 'Outstand
    plt.figure(figsize=(30,6))
    sns.displot(x=col,data=df_clean, hue='Credit_Score', palette=("flare"))
    plt.show()
```

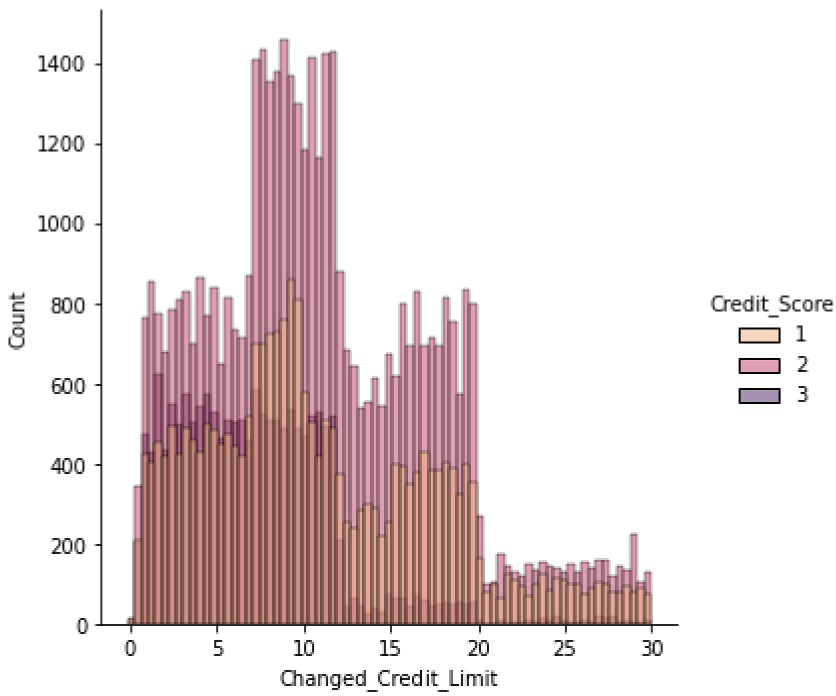
<Figure size 2160x432 with 0 Axes>



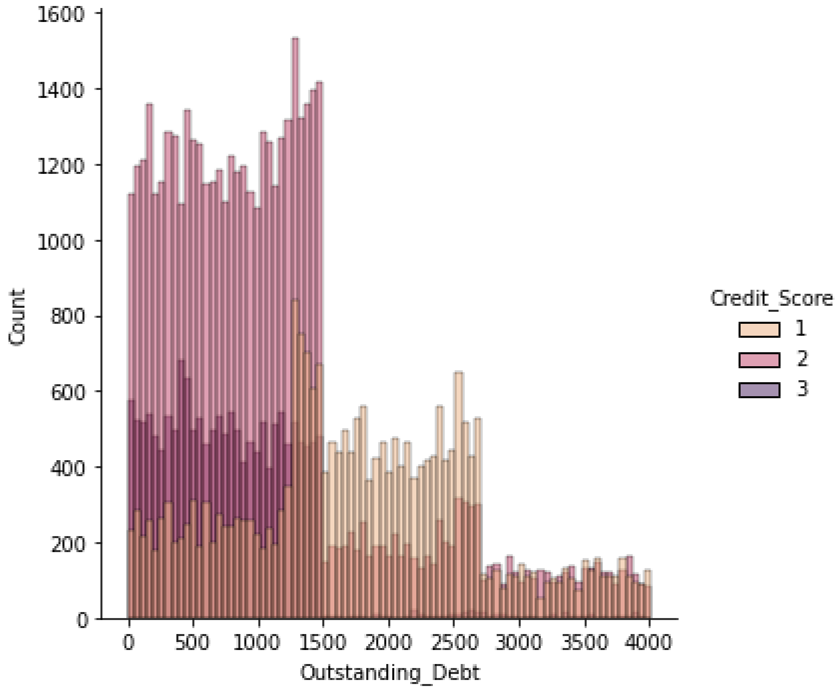

<Figure size 2160x432 with 0 Axes>



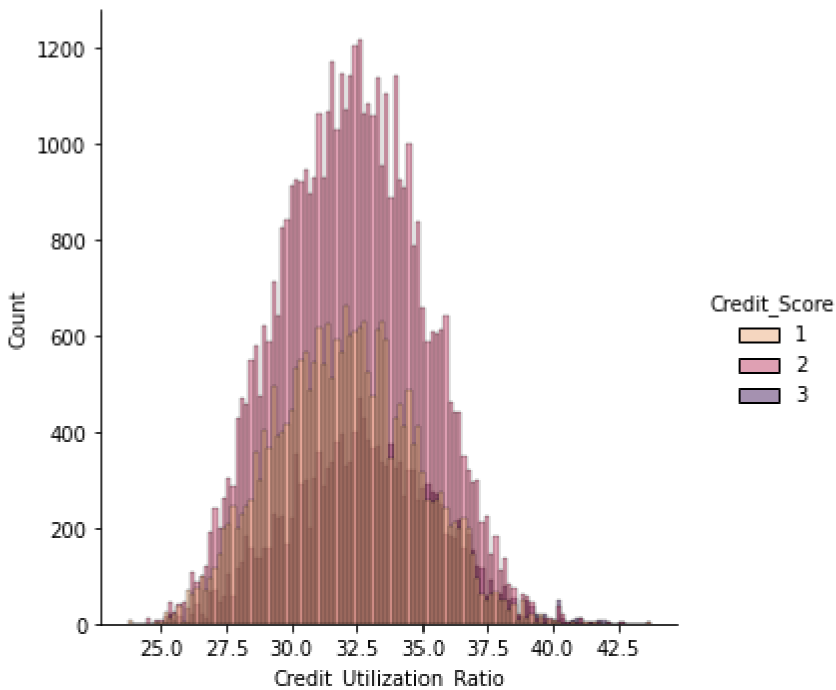

<Figure size 2160x432 with 0 Axes>



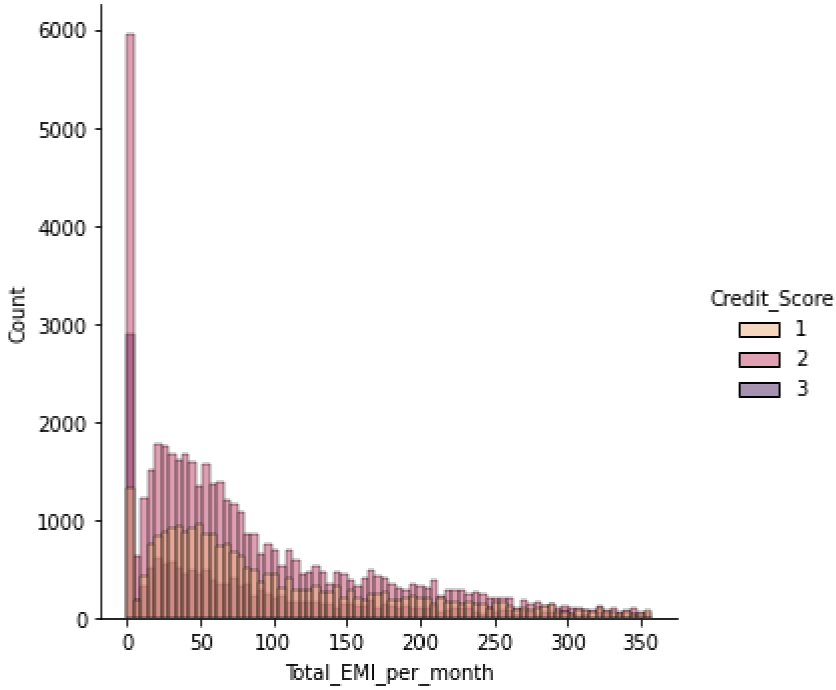

<Figure size 2160x432 with 0 Axes>




<Figure size 2160x432 with 0 Axes>




<Figure size 2160x432 with 0 Axes>




In [39]:
```
#Most counts of Credit_Score in Total_EMI_per_month concentrate naer 0, suggest drop
#Credit_Utilization_Ration has a good normal distribution
```

# Transform dummy for two text columns, then reach final cleaned set

In [40]:
```
#We start to transform dummy for Occupation and Payment_Behaviour
#Dummy Encoding Occupation column
df_cleaned = pd.get_dummies(df_clean, prefix='Occupation', columns=['Occupation'], drc
```

In [41]:
```
#Transform Payment_Behaviour
df_cleaned = pd.get_dummies(df_cleaned, prefix='Payment_Behaviour', columns=['Payment_
df_cleaned.columns
```

Out[41]:
```
Index(['Customer_ID', 'Month', 'Age', 'Annual_Income', 'Monthly_Inhand_Salary',
       'Num_Bank_Accounts', 'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', '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',
       'Payment_Behaviour_High_spent_Large_value_payments',
       'Payment_Behaviour_High_spent_Medium_value_payments',
       'Payment_Behaviour_High_spent_Small_value_payments',
       'Payment_Behaviour_Low_spent_Large_value_payments',
       'Payment_Behaviour_Low_spent_Medium_value_payments',
       'Payment_Behaviour_Low_spent_Small_value_payments'],
      dtype='object')
```

## Possible Column Select via corr_matrix

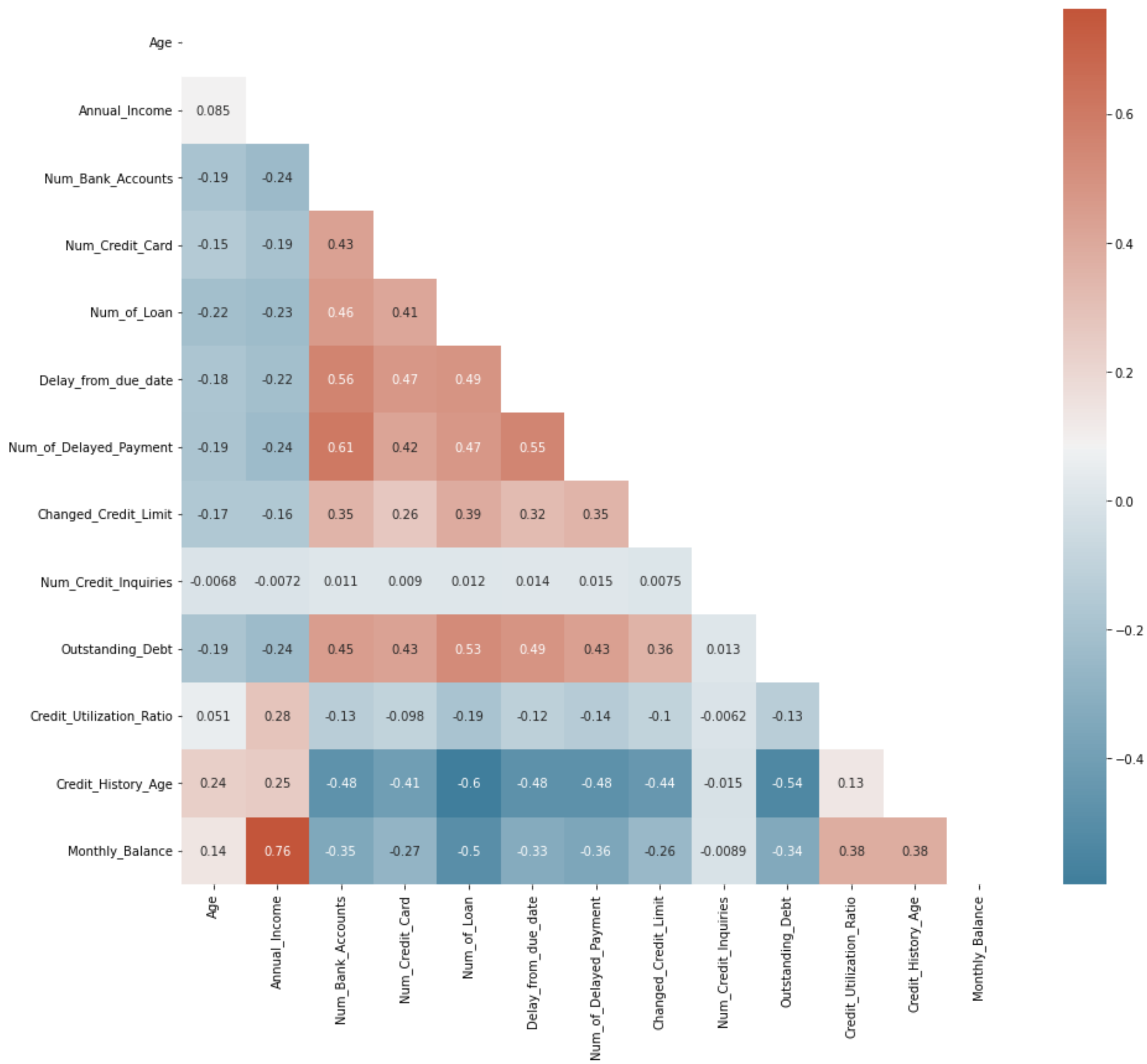
In [42]:
```
#Try to select columns by correlation_matrix, see cor betw explanatory variables and r
corr_matrix = df_cleaned.corr()
corr_matrix["Credit_Score"].sort_values(ascending=False)
```

Out[42]:
```
Credit_Score                                       1.000000
Credit_Mix                                         0.477495
Credit_History_Age                                 0.382949
Monthly_Balance                                    0.261639
Monthly_Inhand_Salary                              0.200337
Annual_Income                                      0.190312
Age                                                0.159261
Amount_invested_monthly                            0.132906
Credit_Utilization_Ratio                           0.090805
Payment_Behaviour_High_spent_Large_value_payments  0.073923
Payment_Behaviour_High_spent_Medium_value_payments 0.040760
Month                                              0.015343
Payment_Behaviour_High_spent_Small_value_payments  0.013625
Occupation_Musician                                0.010056
Occupation_Media_Manager                           0.009851
Occupation_Journalist                              0.007709
Occupation_Architect                               0.006754
Payment_Behaviour_Low_spent_Large_value_payments   0.006626
Occupation_Developer                               0.006178
Occupation_Doctor                                  0.005040
Occupation_Manager                                 0.004045
Occupation_Lawyer                                  0.003854
Occupation_Teacher                                 0.000725
Occupation_Engineer                                0.000433
Occupation_Scientist                              -0.003599
Occupation_Accountant                             -0.004559
Payment_Behaviour_Low_spent_Medium_value_payments -0.007129
Occupation_Entrepreneur                           -0.010552
Occupation_Mechanic                               -0.011679
Num_Credit_Inquiries                              -0.012496
Occupation_Writer                                 -0.024623
Total_EMI_per_month                               -0.067942
Payment_Behaviour_Low_spent_Small_value_payments  -0.103992
Changed_Credit_Limit                              -0.181296
Num_of_Loan                                       -0.349573
Num_of_Delayed_Payment                            -0.370945
Num_Bank_Accounts                                 -0.377880
Payment_of_Min_Amount                             -0.379970
Outstanding_Debt                                  -0.388489
Num_Credit_Card                                   -0.396775
Delay_from_due_date                               -0.428338
Interest_Rate                                     -0.477400
Name: Credit_Score, dtype: float64
```

In [43]:
```
#Drop unnecessary columns
d_col = ['Customer_ID','Month','Monthly_Inhand_Salary',
         'Interest_Rate','Credit_Mix',
         'Amount_invested_monthly', 'Total_EMI_per_month', 'Payment_of_Min_Amount']
df_cleaned_selected = df_cleaned.drop(d_col , axis=1).copy()
#drop 'Customer_ID' due to no predicting power
#drop 'Total_EMI_per_month' due to crowding at low value from distribution graph
#drop 'Monthly_Inhand_Salary','Amount_invested_monthly' due to high inner cor
```

```
#drop 'Credit_Mix', 'Interest_Rate' highly correlate with all other columns in dataset
#drop 'Total_EMI_per_month','Month' from distribution graph
#drop 'Payment_of_Min_Amount' is a dummy column and hard to interpret


corr = df_cleaned_selected.iloc[:,:13].corr()

f, ax = plt.subplots(figsize=(15, 13))
mask = np.triu(np.ones_like(corr, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, annot=True, mask = mask, cmap=cmap)
#By result,we decide to drop 'Annual_Income'
```

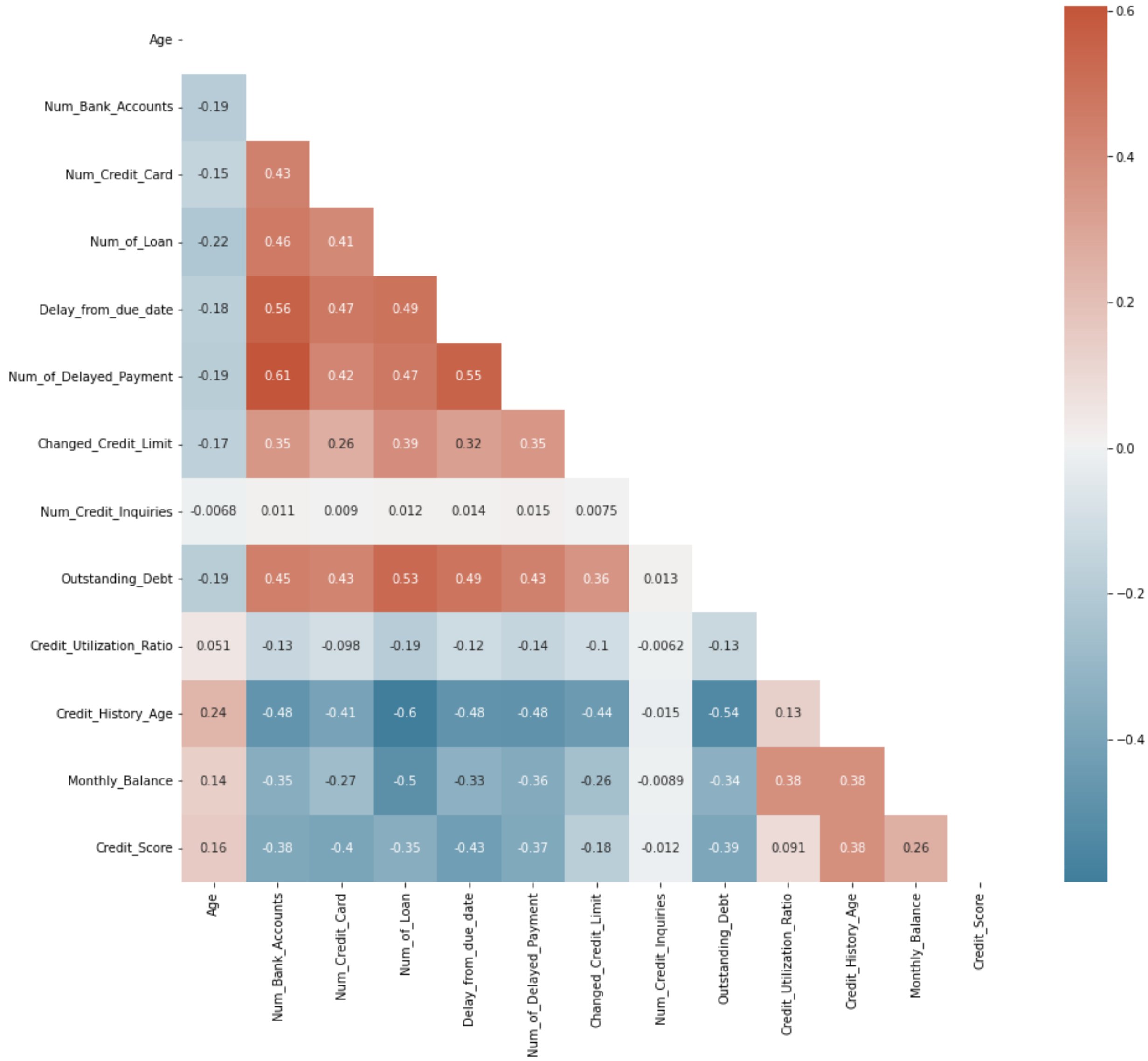
Out[43]: <Axes: >




In [44]:

```
d_col = ['Customer_ID','Month','Monthly_Inhand_Salary',
         'Interest_Rate','Credit_Mix','Annual_Income',
         'Amount_invested_monthly', 'Total_EMI_per_month', 'Payment_of_Min_Amount']
df_cleaned_selected = df_cleaned.drop(d_col , axis=1).copy()
corr = df_cleaned_selected.iloc[:,:13].corr()
f, ax = plt.subplots(figsize=(15, 13))
mask = np.triu(np.ones_like(corr, dtype=bool))
cmap = sns.diverging_palette(230, 20, as_cmap=True)
sns.heatmap(corr, annot=True, mask = mask, cmap=cmap)
```

Out[44]: <Axes: >




In [45]:
```
df_final_cleaned = df_cleaned_selected
df_final_cleaned.head(10)
```

Out[45]:

| | Age | Num_Bank_Accounts | Num_Credit_Card | Num_of_Loan | Delay_from_due_date | Num_of_Delayed |
|---|---|---|---|---|---|---|
| **0** | 23.0 | 3.0 | 4.0 | 4.0 | 3.0 | |
| **2** | 23.0 | 3.0 | 4.0 | 4.0 | 3.0 | |
| **3** | 23.0 | 3.0 | 4.0 | 4.0 | 3.0 | |
| **4** | 23.0 | 3.0 | 4.0 | 4.0 | 3.0 | |
| **5** | 23.0 | 3.0 | 4.0 | 4.0 | 3.0 | |
| **6** | 23.0 | 3.0 | 4.0 | 4.0 | 3.0 | |
| **8** | 28.0 | 2.0 | 4.0 | 1.0 | 3.0 | |
| **9** | 28.0 | 2.0 | 4.0 | 1.0 | 3.0 | |
| **10** | 28.0 | 2.0 | 4.0 | 1.0 | 3.0 | |
| **11** | 28.0 | 2.0 | 4.0 | 1.0 | 3.0 | |

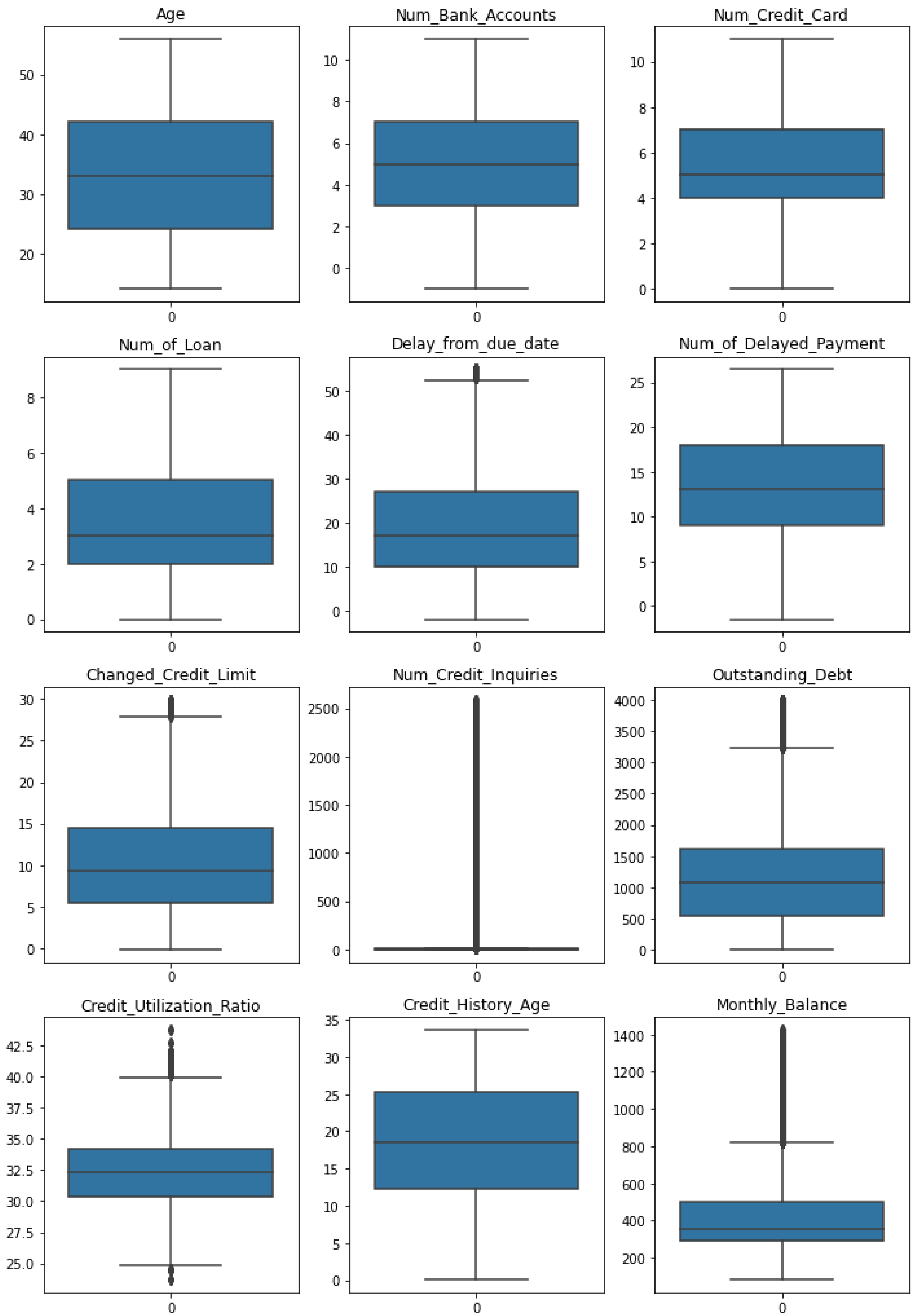
In [46]:

```
df_final_cleaned = df_final_cleaned.iloc[:,:13]
```

In [47]:

```
# detecting outliers
def plot_boxplot(df):
    fig,ax = plt.subplots(3, 3, figsize=(12,18))
    for i in range(df.shape[1]):
        plt.subplot(4,3,i+1)
        ax = sns.boxplot(data = df.iloc[:,i])
        ax.set_title(df.columns[i])
plot_boxplot(df_final_cleaned.iloc[:,:12])
```

```
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
<ipython-input-47-266ad0551cbd>:5: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; ex
plicitly call ax.remove() as needed.
  plt.subplot(4,3,i+1)
```

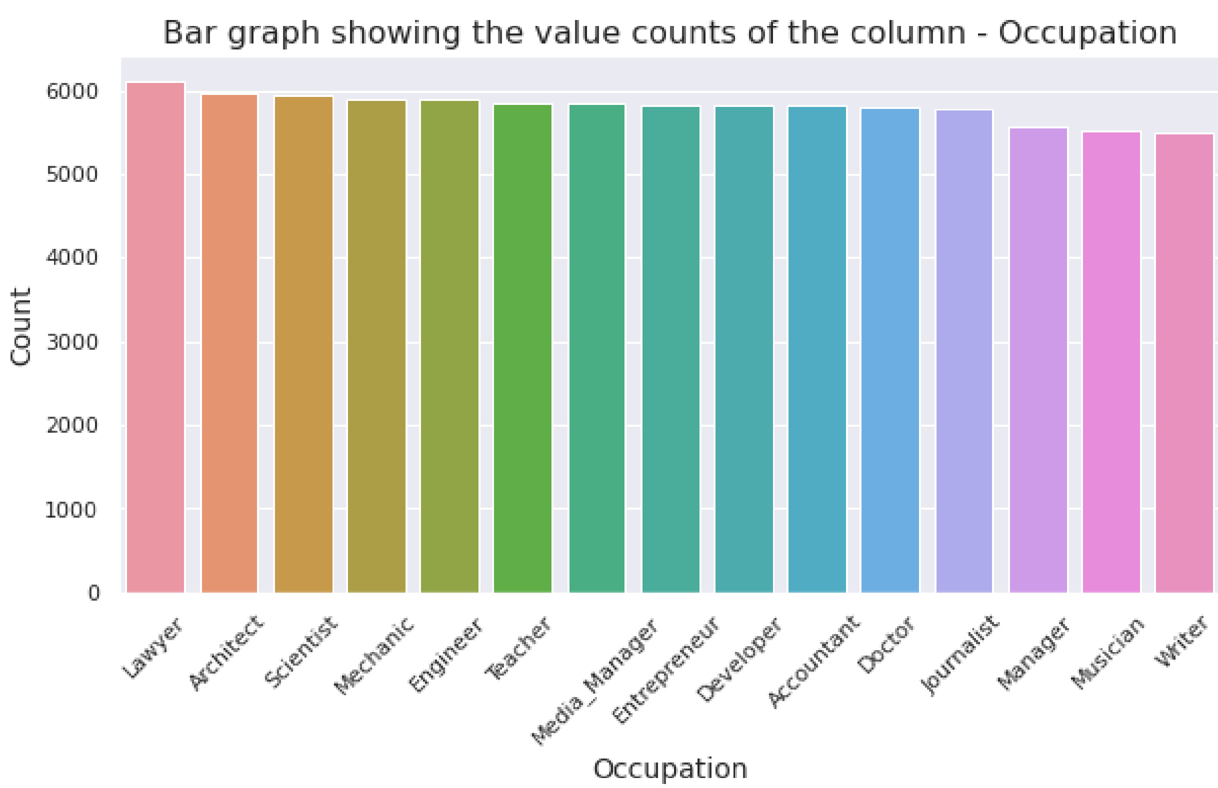

Age
Num_Bank_Accounts
Num_Credit_Card
Num_of_Loan
Delay_from_due_date
Num_of_Delayed_Payment
Changed_Credit_Limit
Num_Credit_Inquiries
Outstanding_Debt
Credit_Utilization_Ratio
Credit_History_Age
Monthly_Balance

# KNN and RF

In [49]:
```
#df_clean
occupation_count = df_clean['Occupation'].value_counts(dropna = False)
occupation_count
```

Out[49]:
```
Lawyer           6106
Architect        5956
Scientist        5933
Mechanic         5893
Engineer         5888
Teacher          5850
Media_Manager    5839
Entrepreneur     5826
Developer        5818
Accountant       5815
Doctor           5789
Journalist       5778
Manager          5558
Musician         5514
Writer           5486
Name: Occupation, dtype: int64
```

In [50]:
```
sns.set(rc={'figure.figsize': (10, 5)})
sns.barplot(x=occupation_count.index, y=occupation_count.values)
plt.title('Bar graph showing the value counts of the column - Occupation', fontsize=16
plt.ylabel('Count', fontsize=14)
plt.xlabel('Occupation', fontsize=14)
plt.xticks(rotation=45)
plt.show()
```




In [51]:
```
df_clean
```

Out[51]:

| | Customer_ID | Month | Age | Occupation | Annual_Income | Monthly_Inhand_Salary | Num_Bank_Ac |
|---|---|---|---|---|---|---|---|
| **0** | CUS_0xd40 | 1 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **2** | CUS_0xd40 | 3 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **3** | CUS_0xd40 | 4 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **4** | CUS_0xd40 | 5 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **5** | CUS_0xd40 | 6 | 23.0 | Scientist | 19114.12 | 1824.843333 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **98298** | CUS_0x9d41 | 3 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98299** | CUS_0x9d41 | 4 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98300** | CUS_0x9d41 | 5 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98302** | CUS_0x9d41 | 7 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |
| **98303** | CUS_0x9d41 | 8 | 38.0 | Lawyer | 41015.55 | 3152.962500 | |

87049 rows × 24 columns

In [52]:

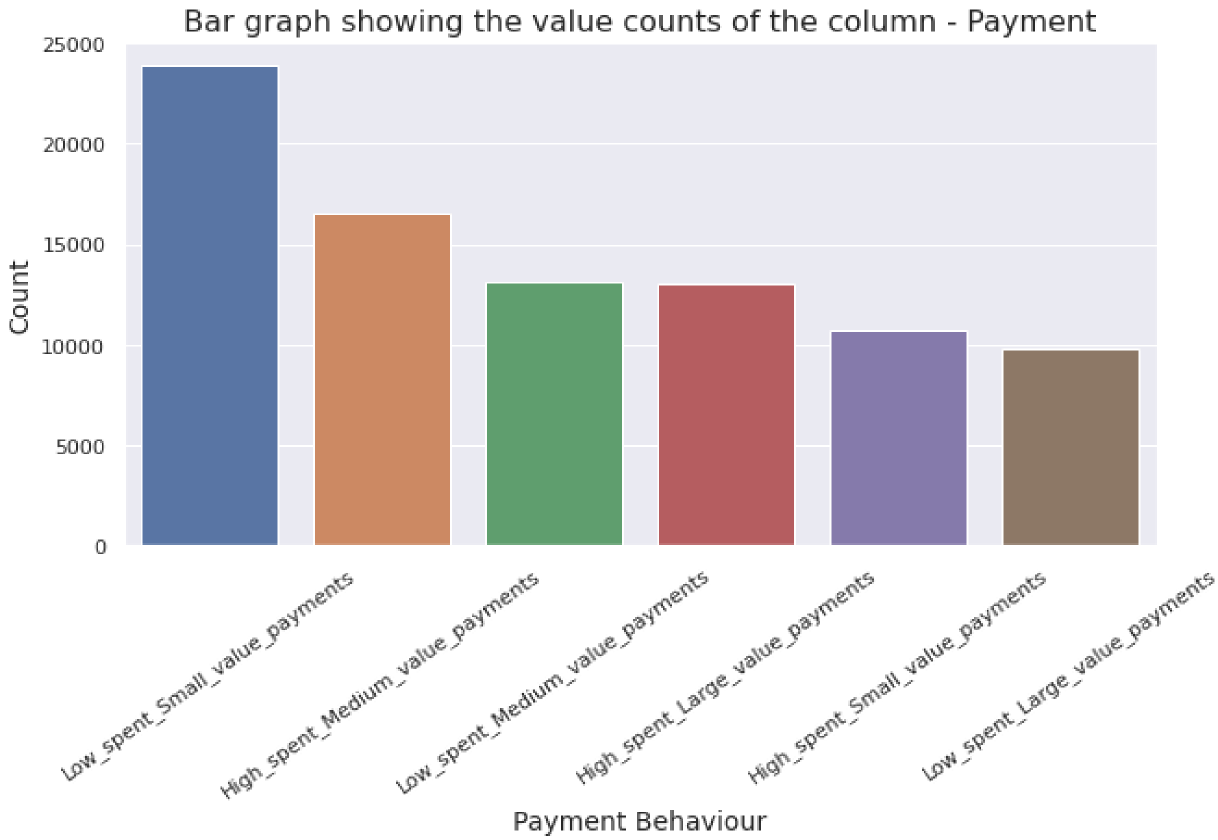
```
#df_clean
payment_count = df_clean['Payment_Behaviour'].value_counts(dropna = False)
payment_count
```

Out[52]:

```
Low_spent_Small_value_payments      23874
High_spent_Medium_value_payments    16543
Low_spent_Medium_value_payments     13089
High_spent_Large_value_payments     13041
High_spent_Small_value_payments     10714
Low_spent_Large_value_payments       9788
Name: Payment_Behaviour, dtype: int64
```

In [53]:

```
sns.set(rc={'figure.figsize': (10, 5)})
sns.barplot(x=payment_count.index, y=payment_count.values)
plt.title('Bar graph showing the value counts of the column - Payment', fontsize=16)
plt.ylabel('Count', fontsize=14)
plt.xlabel('Payment Behaviour', fontsize=14)
plt.xticks(rotation=35)
plt.show()
```




In [54]:
```
y = df_final_cleaned['Credit_Score']
x = df_final_cleaned.loc[:, df_final_cleaned.columns != 'Credit_Score']
```

In [55]:
```
#train test split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,
                                                 train_size=.80,
                                                 test_size=.20,
                                                 stratify=y, # maintain label proporti
                                                 random_state=0)


print(x_test.shape)
print(y_test.value_counts())
```

```
(17410, 12)
2    9334
1    4892
3    3184
Name: Credit_Score, dtype: int64
```

In [56]:
```
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
x_train_ss = pd.DataFrame(ss.fit_transform(x_train), columns = x_train.columns)
x_test_ss = pd.DataFrame(ss.transform(x_test), columns = x_test.columns)
```

In [57]:
```
x_train_ss
```

Out[57]:

| | Age | Num_Bank_Accounts | Num_Credit_Card | Num_of_Loan | Delay_from_due_date | Num_o |
|---|---|---|---|---|---|---|
| **0** | -0.405589 | 1.431971 | 2.207033 | 1.876174 | 0.383355 | |
| **1** | 1.816363 | -1.274446 | -2.179798 | 0.222022 | -0.368742 | |
| **2** | 0.335062 | 0.272078 | 0.257330 | 1.462636 | 1.962759 | |
| **3** | -1.423983 | 0.272078 | 2.207033 | 1.876174 | 1.135452 | |
| **4** | 1.816363 | 0.272078 | 0.257330 | -0.605054 | 0.007307 | |
| **...** | ... | ... | ... | ... | ... | |
| **69634** | 1.908944 | -2.047709 | -0.230096 | -0.191516 | -1.421678 | |
| **69635** | 0.427643 | 1.045340 | 0.744756 | -0.191516 | 2.338808 | |
| **69636** | -1.423983 | -0.114553 | -0.717521 | -0.191516 | -1.421678 | |
| **69637** | -0.498170 | 0.272078 | -0.717521 | 0.222022 | -0.970420 | |
| **69638** | 1.446037 | -0.887815 | 0.744756 | -1.018592 | -0.594371 | |

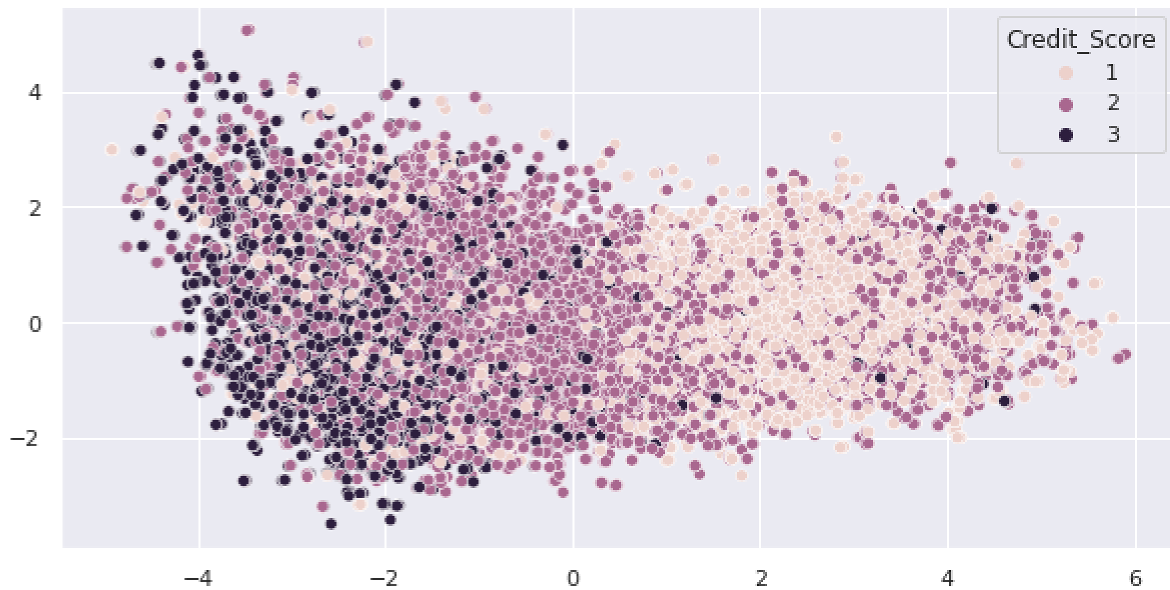
69639 rows × 12 columns

# PCA (not ideal)

In [55]:

```
# feature selection
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

pca = PCA(n_components=0.9, random_state=123)

x_train_pca = pca.fit_transform(x_train_ss)
#x_test_pca = pca.transform(x_test_ss)
print(x_train_pca.shape)
sns.scatterplot(x= x_train_pca[:,0], y=x_train_pca[:,1], hue=y_train);
```

```
(69639, 10)
```




## RF with feature selection

In [56]:
```
#GridSearch for best model hyperparameters
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

params = {'n_estimators':[i for i in range(10,200,10)],
          'max_depth':[i for i in range(3,13,3)]}
gscv = GridSearchCV(RandomForestClassifier(), param_grid=params, cv=5, refit=True)
gscv.fit(x_train_ss, y_train)
gscv.best_params_
```

Out[56]:
```
{'max_depth': 12, 'n_estimators': 190}
```

In [58]:
```
sns.set_style('darkgrid')
feat_imp = pd.Series(data = gscv.best_estimator_.feature_importances_, index = x_trair
feat_imp.sort_values().plot.barh()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-58-884d1d98c96b> in <module>
      1 sns.set_style('darkgrid')
----> 2 feat_imp = pd.Series(data = gscv.best_estimator_.feature_importances_, index
= x_train_ss.columns)
      3 feat_imp.sort_values().plot.barh()

NameError: name 'gscv' is not defined
```

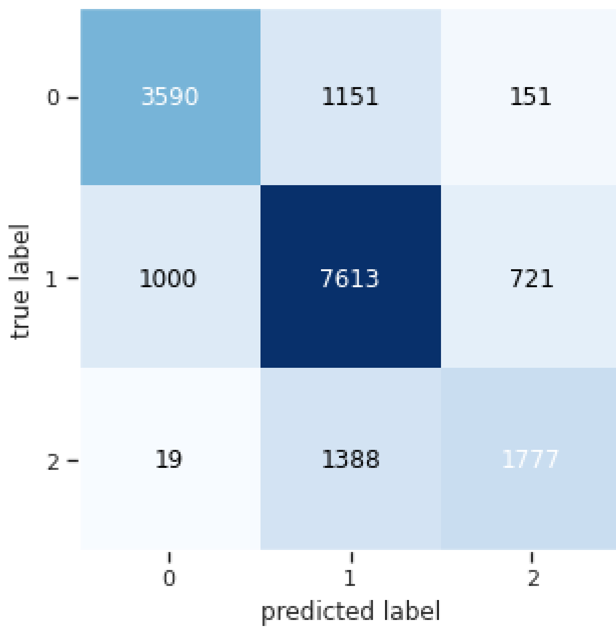
In [59]:
```
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix
```

In [61]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=160, # number of trees in ensemble
                             max_depth = 12,
                             n_jobs= -1,       # parallelize using all available cores
```

```
                              random_state=0   # for demonstration only
                             )
rfc.fit(x_train_ss, y_train)

y_pred_rfc = rfc.predict(x_test_ss)
print(accuracy_score(y_test, y_pred_rfc))
print(classification_report(y_test, y_pred_rfc))
```

```
0.7455485353245261
              precision    recall  f1-score   support

           1       0.78      0.73      0.76      4892
           2       0.75      0.82      0.78      9334
           3       0.67      0.56      0.61      3184

    accuracy                           0.75     17410
   macro avg       0.73      0.70      0.72     17410
weighted avg       0.74      0.75      0.74     17410
```

In [62]:
```
#confusion matrix
rfc_cm = plot_confusion_matrix(confusion_matrix(y_test,y_pred_rfc));
```




# KNN

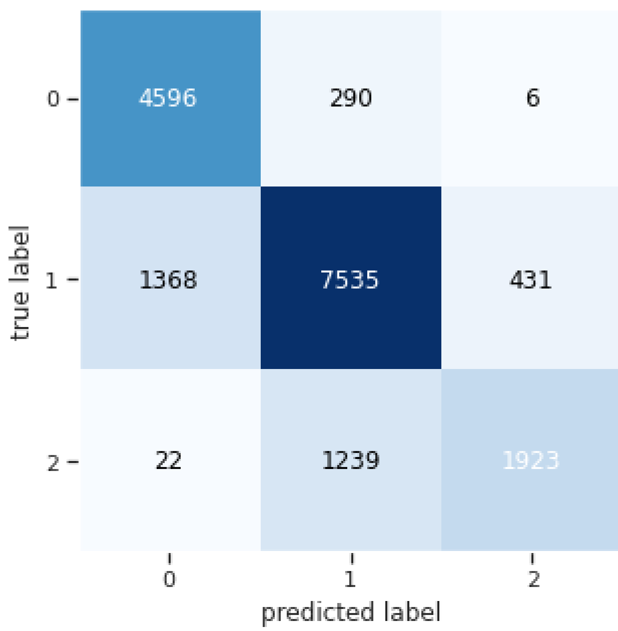
In [ ]:
```
#find the best number of neighbor
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
mean_scores = []
for n in [2,4,6,8,10,12]:
    knn = KNeighborsClassifier(n_neighbors = n)
    scores = cross_val_score(knn, x_train_ss, y_train, cv = 3)
    mean_scores.append((n, scores.mean().round(4)))
sorted(mean_scores, key=lambda x:x[1],reverse=True)[0]
```

Out[ ]:
```
(2, 0.775)
```

In [64]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(x_train_ss, y_train)

y_pred_knn = knn.predict(x_test_ss)
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
Accuracy: 0.8072372199885124
              precision    recall  f1-score   support

           1       0.77      0.94      0.85      4892
           2       0.83      0.81      0.82      9334
           3       0.81      0.60      0.69      3184

    accuracy                           0.81     17410
   macro avg       0.80      0.78      0.79     17410
weighted avg       0.81      0.81      0.80     17410
```

In [65]:
```
#confusion matrix
knn_cm = plot_confusion_matrix(confusion_matrix(y_test,y_pred_knn));
```




# Balancing Data

In [66]:
```
# class count
class_count_1, class_count_2, class_count_3 = df_final_cleaned['Credit_Score'].value_c

# Separate class
class_1 = df_final_cleaned[df_final_cleaned['Credit_Score'] == 1]
class_2 = df_final_cleaned[df_final_cleaned['Credit_Score'] == 2]
class_3 = df_final_cleaned[df_final_cleaned['Credit_Score'] == 3]# print the shape of

print('class 1:', class_1.shape)
print('class 2:', class_2.shape)
print('class 3:', class_3.shape)
```

```
class 1: (24457, 13)
class 2: (46671, 13)
class 3: (15921, 13)
```

In [ ]:

In [67]:

```
class_1_under = class_1.sample(class_count_3)
class_2_under = class_2.sample(class_count_3)

test_under = pd.concat([class_1_under, class_2_under, class_3], axis=0)

print("total class of 1 and0:",test_under['Credit_Score'].value_counts())# plot the cc
test_under['Credit_Score'].value_counts().plot(kind='bar', title='Credit Score count')
```

```
total class of 1 and0: 1    15921
2    15921
3    15921
Name: Credit_Score, dtype: int64
```

Out[67]:

```
<Axes: title={'center': 'Credit Score count'}>
```


In [68]:

```
y_bal = test_under['Credit_Score']
x_bal= test_under.loc[:, test_under.columns != 'Credit_Score']

#train test split
from sklearn.model_selection import train_test_split
x_bal_train,x_bal_test,y_bal_train,y_bal_test = train_test_split(x_bal,
                                                    y_bal,
                                                    train_size=.80,
                                                    test_size=.20,
                                                    stratify=y_bal, # maintain label prop
                                                    random_state=0
                                                   )

print(x_bal_test.shape)
print(y_bal_test.value_counts())

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
```

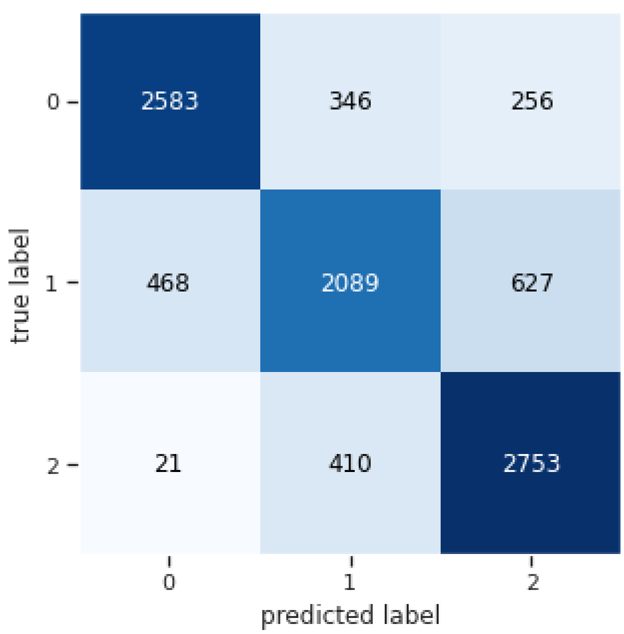
```
x_bal_train_ss = pd.DataFrame(ss.fit_transform(x_bal_train), columns = x_bal_train.col
x_bal_test_ss = pd.DataFrame(ss.transform(x_bal_test), columns = x_bal_test.columns)
```

```
(9553, 12)
1    3185
3    3184
2    3184
Name: Credit_Score, dtype: int64
```

In [69]:
```
from sklearn.ensemble import RandomForestClassifier
rfc_bal = RandomForestClassifier(n_estimators=180, # number of trees in ensemble
                             max_depth = 15,
                             n_jobs= -1,       # parallelize using all available cores
                             random_state=0   # for demonstration only
                            )
rfc_bal.fit(x_bal_train_ss, y_bal_train)

y_bal_pred_rfc = rfc_bal.predict(x_bal_test_ss)
print(accuracy_score(y_bal_test, y_bal_pred_rfc))
print(classification_report(y_bal_test, y_bal_pred_rfc))
```

```
0.7772427509682822
              precision    recall  f1-score   support

           1       0.84      0.81      0.83      3185
           2       0.73      0.66      0.69      3184
           3       0.76      0.86      0.81      3184

    accuracy                           0.78      9553
   macro avg       0.78      0.78      0.78      9553
weighted avg       0.78      0.78      0.78      9553
```

In [70]:
```
#confusion matrix
rfc_bal_cm = plot_confusion_matrix(confusion_matrix(y_bal_test, y_bal_pred_rfc));
```



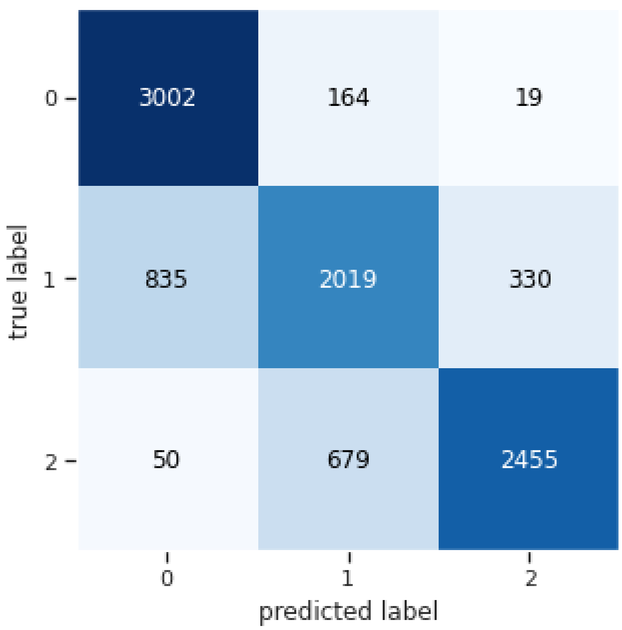

In [71]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_bal = KNeighborsClassifier(n_neighbors=2)
knn_bal.fit(x_bal_train_ss, y_bal_train)
```

```
y_bal_pred_knn = knn_bal.predict(x_bal_test_ss)
print("Accuracy:", accuracy_score(y_bal_test, y_bal_pred_knn))
print(classification_report(y_bal_test, y_bal_pred_knn))
```

```
Accuracy: 0.7825813880456401
              precision    recall  f1-score   support

           1       0.77      0.94      0.85      3185
           2       0.71      0.63      0.67      3184
           3       0.88      0.77      0.82      3184

    accuracy                           0.78      9553
   macro avg       0.78      0.78      0.78      9553
weighted avg       0.78      0.78      0.78      9553
```

In [72]:

```
#confusion matrix
knn_bal_cm = plot_confusion_matrix(confusion_matrix(y_bal_test,y_bal_pred_knn));
```




In [ ]:

## Softmax regression

In [73]:

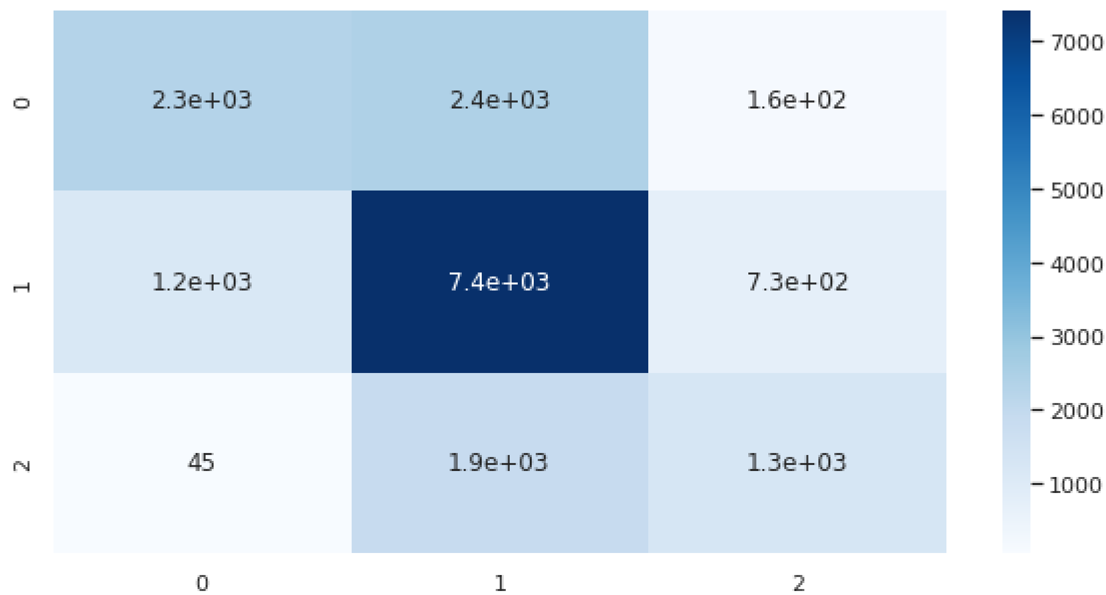
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPClassifier
```

In [74]:

```
from sklearn.metrics import confusion_matrix

softmax = LogisticRegression(multi_class='multinomial')
softmax.fit(x_train_ss, y_train)
y_pred = softmax.predict(x_test_ss)
```

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap='Blues')
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6325674899483056
              precision    recall  f1-score   support

           1       0.66      0.47      0.55      4892
           2       0.63      0.80      0.71      9334
           3       0.59      0.40      0.47      3184

    accuracy                           0.63     17410
   macro avg       0.63      0.56      0.58     17410
weighted avg       0.63      0.63      0.62     17410
```




## Grid Search

In [78]:

```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 1000],
    'penalty': ['l1','l2', 'none'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

}

# Define the model to use for grid search
model = LogisticRegression(multi_class='multinomial', max_iter=1000, random_state=0)

# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, n_jobs=-1, error_score=
grid_search.fit(x_train_ss, y_train)

# Print the best hyperparameters and corresponding mean cross-validation score
print("Best hyperparameters: ", grid_search.best_params_)
```

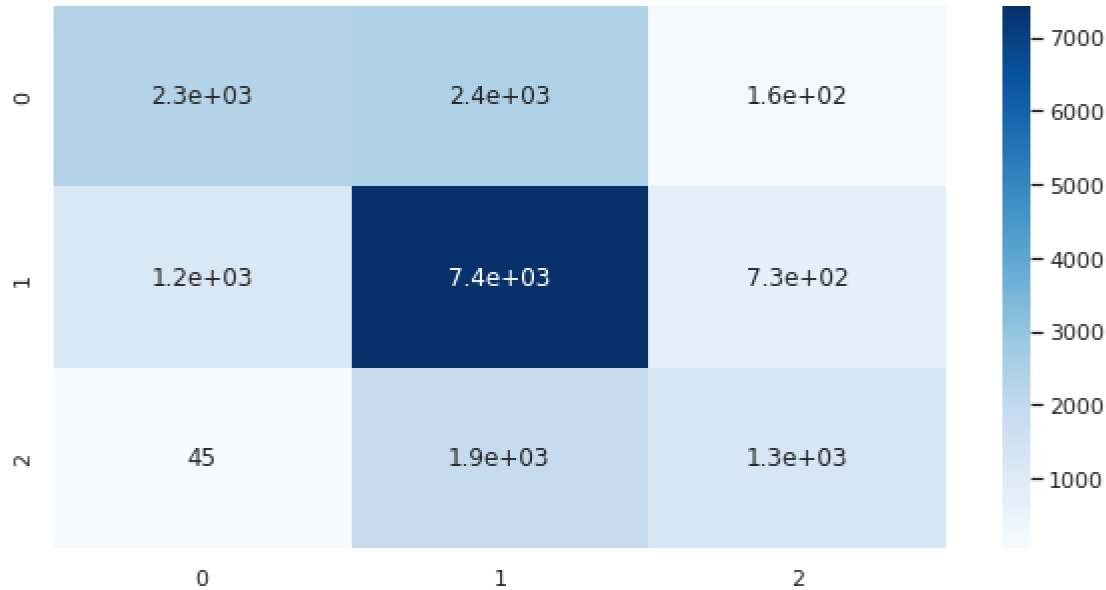
```
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:378: Fi
tFailedWarning:
180 fits failed out of a total of 450.
The score on these train-test partitions for these parameters will be set to -1.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1207, in fit
    multi_class = _check_multi_class(self.multi_class, solver, len(self.classes_))
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 90, in _check_multi_class
    raise ValueError("Solver %s does not support a multinomial backend." % solver)
ValueError: Solver liblinear does not support a multinomial backend.

--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
```

```
ne 54, in _check_solver
    raise ValueError(
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 71, in _check_solver
    raise ValueError("penalty='none' is not supported for the liblinear solver")
ValueError: penalty='none' is not supported for the liblinear solver

  warnings.warn(some_fits_failed_message, FitFailedWarning)
Best hyperparameters:  {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
```

In [79]:

```
y_pred_sf = grid_search.predict(x_test_ss)

# Evaluate the accuracy of the model on your test set

cm = confusion_matrix(y_test, y_pred_sf)
sns.heatmap(cm, annot=True, cmap='Blues')
print("Best hyperparameters: ", grid_search.best_params_)
print("Accuracy: ", accuracy_score(y_test, y_pred_sf))
print(classification_report(y_test, y_pred_sf))
```

```
Best hyperparameters:  {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
Accuracy:  0.6327972429638139
              precision    recall  f1-score   support

           1       0.66      0.47      0.55      4892
           2       0.63      0.80      0.71      9334
           3       0.59      0.40      0.47      3184

    accuracy                           0.63     17410
   macro avg       0.63      0.56      0.58     17410
weighted avg       0.63      0.63      0.62     17410
```

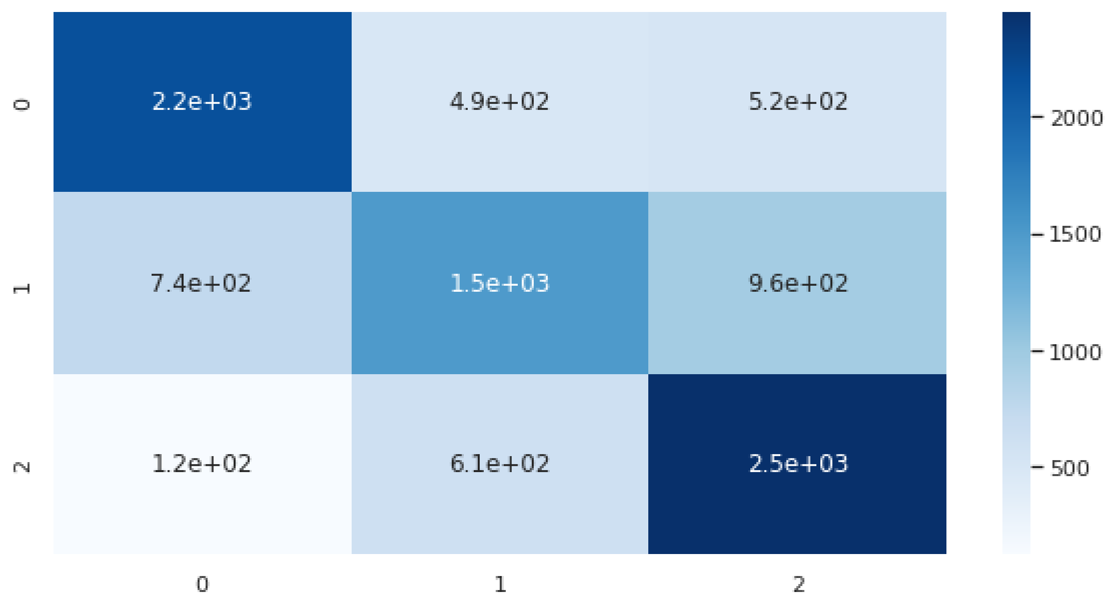



# Balancing data

In [80]:
```
grid_search.fit(x_bal_train_ss, y_bal_train)

# Predict the class labels on your test set using the best model
y_pred_bal_sf = grid_search.predict(x_bal_test_ss)

# Evaluate the accuracy of the model on your test set

cm = confusion_matrix(y_bal_test, y_pred_bal_sf)
sns.heatmap(cm, annot=True, cmap='Blues')
print("Best hyperparameters: ", grid_search.best_params_)
print("Accuracy: ", accuracy_score(y_bal_test, y_pred_bal_sf))
print(classification_report(y_bal_test, y_pred_bal_sf))
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:378: Fi
tFailedWarning:
180 fits failed out of a total of 450.
The score on these train-test partitions for these parameters will be set to -1.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 54, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
60 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1207, in fit
    multi_class = _check_multi_class(self.multi_class, solver, len(self.classes_))
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 90, in _check_multi_class
    raise ValueError("Solver %s does not support a multinomial backend." % solver)
ValueError: Solver liblinear does not support a multinomial backend.

--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
```

```
ne 54, in _check_solver
    raise ValueError(
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.p
y", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py", li
ne 71, in _check_solver
    raise ValueError("penalty='none' is not supported for the liblinear solver")
ValueError: penalty='none' is not supported for the liblinear solver

  warnings.warn(some_fits_failed_message, FitFailedWarning)
Best hyperparameters:  {'C': 0.001, 'penalty': 'l1', 'solver': 'saga'}
Accuracy:  0.6396943368575316
              precision    recall  f1-score   support

           1       0.72      0.68      0.70      3185
           2       0.57      0.47      0.52      3184
           3       0.62      0.77      0.69      3184

    accuracy                           0.64      9553
   macro avg       0.64      0.64      0.63      9553
weighted avg       0.64      0.64      0.63      9553
```




# Ensemble-adaboost

In [ ]:

```
from sklearn.ensemble import VotingClassifier, AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 1000],
```

```
    'penalty': ['l1','l2', 'none'],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

}

# Define the model to use for grid search
model = LogisticRegression(multi_class='multinomial', max_iter=1000, random_state=0)

# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5, n_jobs=-1, error_score=
rfc_bal = RandomForestClassifier(n_estimators=180, # number of trees in ensemble
                             max_depth = 15,
                             n_jobs= -1,       # parallelize using all available cores
                             random_state=0   # for demonstration only
                            )

boosting_ensemble = AdaBoostClassifier(
    VotingClassifier(estimators=[('rf', rfc_bal), ('softmax', model)], voting='soft'),
    n_estimators=180,
    random_state=0
)

boosting_ensemble.fit(x_bal_train_ss, y_bal_train)
y_boost_pred = boosting_ensemble.predict(x_bal_test_ss)
accuracy = accuracy_score(y_bal_test, y_boost_pred)
print("Accuracy:", accuracy)
```

In [93]:

```
combined_preds.reshape(-1, 1)
```

Out[93]:

```
array([[1],
       [3],
       [3],
       ...,
       [2],
       [1],
       [1]])
```

In [87]:

```
## stacking
from sklearn.ensemble import VotingClassifier, AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc_bal = RandomForestClassifier(n_estimators=180, # number of trees in ensemble
                             max_depth = 15,
                             n_jobs= -1,       # parallelize using all available cores
                             random_state=0   # for demonstration only
                            )
rfc_bal.fit(x_bal_train_ss, y_bal_train)
y_bal_pred_rfc = rfc_bal.predict(x_bal_test_ss)
combined_preds = np.hstack([y_bal_pred_rfc, y_pred_bal_sf])
model.fit(combined_preds.reshape(-1, 1), y_bal_train.reshape(-1, 1))
y_pred_stack = model.predict(combined_preds)
print(accuracy_score(y_bal_test, y_pred_stack))
print(classification_report(y_bal_test, y_pred_stack))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-87-7e458d230769> in <module>
     11 y_bal_pred_rfc = rfc_bal.predict(x_bal_test_ss)
     12 combined_preds = np.hstack([y_bal_pred_rfc, y_pred_bal_sf])
---> 13 model.fit(combined_preds.reshape(-1, 1), y_bal_train.reshape(-1, 1))
     14 y_pred_stack = model.predict(combined_preds)
     15 print(accuracy_score(y_bal_test, y_pred_stack))

/usr/local/lib/python3.9/dist-packages/pandas/core/generic.py in __getattr__(self, na
me)
   5573         ):
   5574             return self[name]
-> 5575         return object.__getattribute__(self, name)
   5576 
   5577     def __setattr__(self, name: str, value) -> None:

AttributeError: 'Series' object has no attribute 'reshape'
```

## Naive Bayes

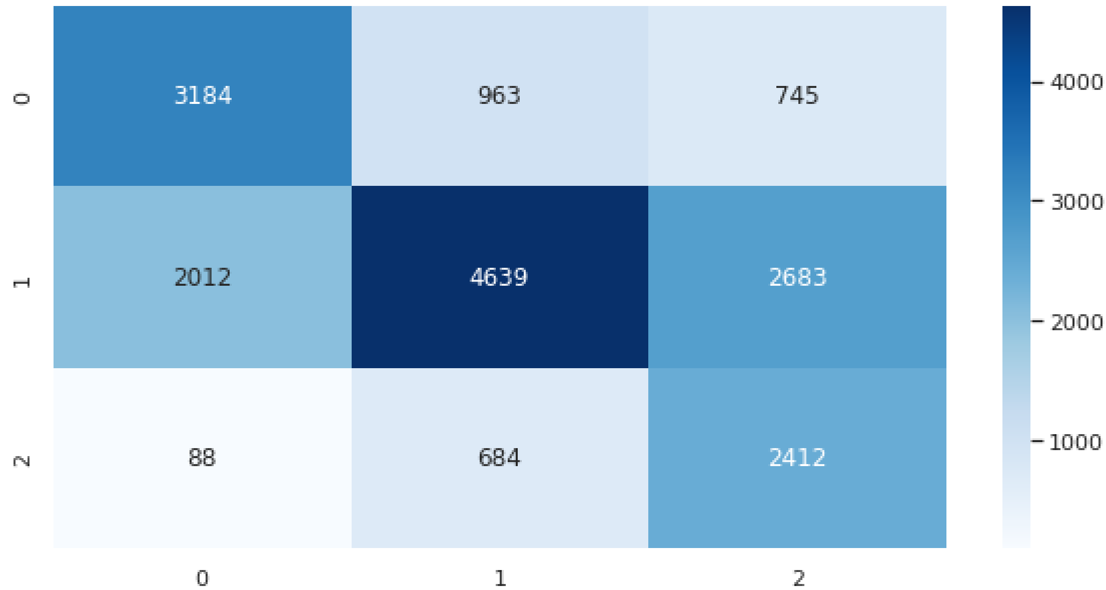
In [ ]:

```
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(x_train, y_train)
y_pred = nb_classifier.predict(x_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')
```

```
Accuracy: 0.5878805284319357
              precision    recall  f1-score   support

           1       0.60      0.65      0.63      4892
           2       0.74      0.50      0.59      9334
           3       0.41      0.76      0.53      3184

    accuracy                           0.59     17410
   macro avg       0.58      0.64      0.58     17410
weighted avg       0.64      0.59      0.59     17410
```

Out[ ]:

```
<AxesSubplot:>
```


3184
963
745
2012
4639
2683
88
684
2412
0
1
2
0
1
2
4000
3000
2000
1000