Credit Score Classification

Introduction

Dans ce projet, nous explorons un ensemble de données sur les scores de crédit avec l'objectif de comprendre les relations entre différentes variables et de préparer les données pour la modélisation de la classification du score de crédit

Importer

Entrée [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings, re, joblib
warnings.filterwarnings("ignore")
from scipy.stats import probplot
```

PARTIE 1

Aperçu statistique de l'ensemble de données

Lire des données

Entrée [2]:
```
df = pd.read_csv("train.csv")
df.head()
```

Out[2]:

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_In
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 rows × 28 columns

La dimension de trames de données

Entrée [3]:
```
df.shape
```

Out[3]: (100000, 28)

Entrée [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   ID                        100000 non-null  object 
 1   Customer_ID               100000 non-null  object 
 2   Month                     100000 non-null  object 
 3   Name                      90015 non-null   object 
 4   Age                       100000 non-null  object 
 5   SSN                       100000 non-null  object 
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  object 
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64  
 10  Num_Credit_Card           100000 non-null  int64  
 11  Interest_Rate             100000 non-null  int64  
 12  Num_of_Loan               100000 non-null  object 
 13  Type_of_Loan              88592 non-null   object 
 14  Delay_from_due_date       100000 non-null  int64  
 15  Num_of_Delayed_Payment    92998 non-null   object 
 16  Changed_Credit_Limit      100000 non-null  object 
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  object 
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object 
 22  Payment_of_Min_Amount     100000 non-null  object 
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object 
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           98800 non-null   object 
 27  Credit_Score              100000 non-null  object 
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

Affichage certains détails statistiques de base comme le centile, la moyenne, l'écart type,...etc

Entrée [5]:
```
df.describe()
```

Out[5]:

	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Delay_fror
count	84998.000000	100000.000000	100000.00000	100000.000000	10(
mean	4194.170850	17.091280	22.47443	72.466040	
std	3183.686167	117.404834	129.05741	466.422621	
min	303.645417	-1.000000	0.00000	1.000000	
25%	1625.568229	3.000000	4.00000	8.000000	
50%	3093.745000	6.000000	5.00000	13.000000	
75%	5957.448333	7.000000	7.00000	20.000000	
max	15204.633333	1798.000000	1499.00000	5797.000000	

Compter le nombre de valeurs manquantes (NaN ou NULL)

Entrée [6]:
```
df.isna().sum()
```

Out[6]:
```
ID                              0
Customer_ID                     0
Month                           0
Name                         9985
Age                             0
SSN                             0
Occupation                      0
Annual_Income                   0
Monthly_Inhand_Salary       15002
Num_Bank_Accounts               0
Num_Credit_Card                 0
Interest_Rate                   0
Num_of_Loan                     0
Type_of_Loan                11408
Delay_from_due_date             0
Num_of_Delayed_Payment       7002
Changed_Credit_Limit            0
Num_Credit_Inquiries         1965
Credit_Mix                      0
Outstanding_Debt                0
Credit_Utilization_Ratio        0
Credit_History_Age           9030
Payment_of_Min_Amount           0
Total_EMI_per_month             0
Amount_invested_monthly      4479
Payment_Behaviour               0
Monthly_Balance              1200
Credit_Score                    0
dtype: int64
```

Vérifier s'il y a des lignes dupliquées

Entrée [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

Supprimer ces colonnes n'affecte pas le score de crédit comme 'ID', Customer_ID', 'Name', 'SSN'

Entrée [8]:
```
df.drop(['ID', 'Customer_ID', 'Name', 'SSN','Type_of_Loan'], axis=1, inplace =
df.columns
```

Out[8]:
```
Index(['Month', 'Age', 'Occupation', 'Annual_Income', 'Monthly_Inhand_Salar
y',
       'Num_Bank_Accounts', 'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loa
n',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limi
t',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score'],
      dtype='object')
```

Alternative à l'information

Entrée [9]:
```
def columns_info (df):
    columns=[]
    dtypes=[]
    unique=[]
    nunique=[]
    nulls=[]

    for cols in df.columns:
        columns.append(cols)
        dtypes.append(df[cols].dtypes)
        unique.append(df[cols].unique())
        nunique.append(df[cols].nunique())
        nulls.append(df[cols].isna().sum())

    return pd.DataFrame({'Columns': columns,
                         'Data Types': dtypes,
                         'Unique Values': unique,
                         'Number of unique': nunique,
                         'Missing Values': nulls
                        })
columns_info(df)
```

Out[9]:

	Columns	Data Types	Unique Values	Number of unique	Missing Values
0	Month	object	[January, February, March, April, May, June, J...	8	0
1	Age	object	[23, -500, 28_, 28, 34, 54, 55, 21, 31, 33, 34...	1788	0
2	Occupation	object	[Scientist, _______, Teacher, Engineer, Entrep...	16	0
3	Annual_Income	object	[19114.12, 34847.84, 34847.84_, 143162.64, 306...	18940	0
4	Monthly_Inhand_Salary	float64	[1824.8433333333328, nan, 3037.986666666666, 1...	13235	15002
5	Num_Bank_Accounts	int64	[3, 2, 1, 7, 4, 0, 8, 5, 6, 9, 10, 1414, 1231,...	943	0
6	Num_Credit_Card	int64	[4, 1385, 5, 1288, 1, 7, 6, 1029, 488, 8, 1381...	1179	0
7	Interest_Rate	int64	[3, 6, 8, 4, 5, 5318, 15, 7, 12, 20, 1, 433, 1...	1750	0
8	Num_of_Loan	object	[4, 1, 3, 967, -100, 0, 0_, 2, 3_, 2_, 7, 5, 5...	434	0
9	Delay_from_due_date	int64	[3, -1, 5, 6, 8, 7, 13, 10, 0, 4, 9, 1, 12, 11...	73	0
10	Num_of_Delayed_Payment	object	[7, nan, 4, 8_, 6, 1, -1, 3_, 0, 8, 5, 3, 9, 1...	749	7002
11	Changed_Credit_Limit	object	[11.27, _, 6.27, 9.27, 5.42, 7.42, 6.42, 7.1, ...	4384	0
12	Num_Credit_Inquiries	float64	[4.0, 2.0, 3.0, nan, 5.0, 9.0, 8.0, 7.0, 6.0, ...	1223	1965
13	Credit_Mix	object	[_, Good, Standard, Bad]	4	0
14	Outstanding_Debt	object	[809.98, 605.03, 1303.01, 632.46, 943.86, 548....	13178	0
15	Credit_Utilization_Ratio	float64	[26.822619623699016, 31.94496005538421, 28.609...	100000	0
16	Credit_History_Age	object	[22 Years and 1 Months, nan, 22 Years and 3 Mo...	404	9030
17	Payment_of_Min_Amount	object	[No, NM, Yes]	3	0
18	Total_EMI_per_month	float64	[49.57494921489417, 18.816214573128885, 246.99...	14950	0
19	Amount_invested_monthly	object	[80.41529543900253, 118.28022162236736, 81.699...	91049	4479
20	Payment_Behaviour	object	[High_spent_Small_value_payments, Low_spent_La...	7	0
21	Monthly_Balance	object	[312.49408867943663, 284.62916249607184, 331.2...	98792	1200
22	Credit_Score	object	[Good, Standard, Poor]	3	0

PARTIE 2

Ingénierie des fonctionnalités

Fonction pour le traitement des valeurs aberrantes (outliers) en utilisant Interquartile Range (IQR)

Entrée [10]:
```
def check_outliers(col, df):
    col_data= df[col]
    q1 = col_data.quantile(0.25)
    q3 = col_data.quantile(0.75)
    iqr= q3-q1
    lower_bound = q1-1.5*iqr
    upper_bound = q3+1.5*iqr
    outliers = []

    #outliers = col_data[(col_data<lower_bound)|(col_data>upper_bound)]

    for i in range(len(df)):
        value = df.loc[i, col]
        if value > upper_bound or value < lower_bound:
            outliers.append(value)

    return outliers
```

Entrée [11]:
```
def handle_outliers(col, df):
    col_data= df[col]
    q1 = col_data.quantile(0.25)
    q3 = col_data.quantile(0.75)
    iqr= q3-q1
    lower_bound = q1-1.5*iqr
    upper_bound = q3+1.5*iqr
    outliers = []

    # Remplacer les valeurs aberrantes par les bornes
    #df[col] = df[col].clip(lower=lower_bound, upper=upper_bound)

    for i in range(len(df)):
        if df.loc[i, col] > upper_bound:
            df.loc[i, col] = upper_bound
        elif df.loc[i, col] < lower_bound:
            df.loc[i, col] = lower_bound
```

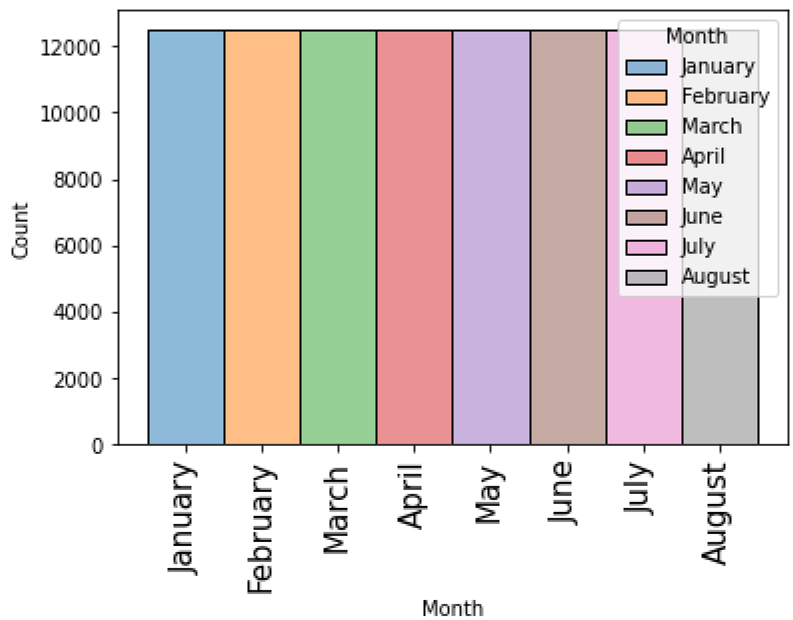
Nettoyage des données et Traitement des outliers

1. Month

Entrée [12]:
```
df["Month"].value_counts()
```

Out[12]:
```
January     12500
February    12500
March       12500
April       12500
May         12500
June        12500
July        12500
August      12500
Name: Month, dtype: int64
```

Entrée [13]:
```
#plt.figure(figsize=(7,5))
#sns.countplot(y="Month", data=df, palette="Dark2")
plt.xticks(fontsize=15, rotation = 'vertical')
sns.histplot(df, x='Month', hue='Month')
plt.show()
```



Entrée [14]:
```
df['Month'] = df['Month'].map({'January':1,'February':2,'March':3,'April':4,'M
df['Month'].unique()
```

Out[14]:
```
array([1, 2, 3, 4, 5, 6, 7, 8], dtype=int64)
```

2. Age

Entrée [15]:
```
df['Age'].unique()
```

Out[15]:
```
array(['23', '-500', '28_', ..., '4808_', '2263', '1342'], dtype=object)
```

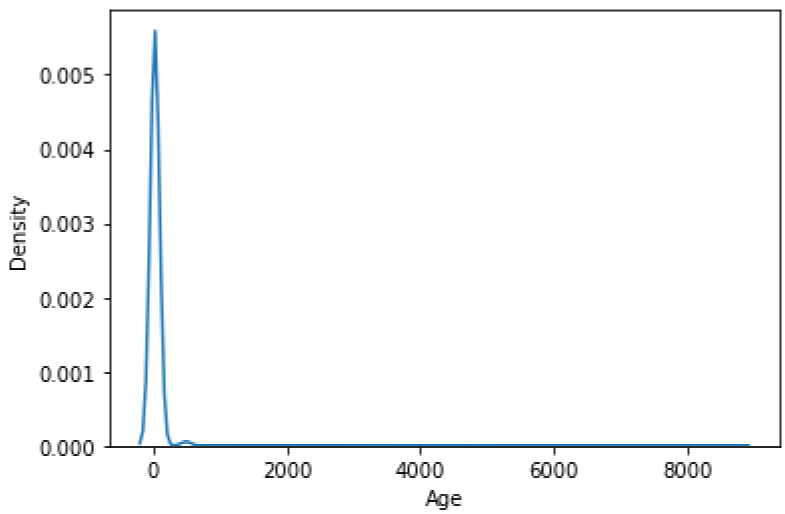
Entrée [16]:
```
df['Age'] = df['Age'].str.replace('-','')
df['Age'] = df['Age'].str.replace('_','')
df['Age'] = df['Age'].astype(int)
df["Age"].unique()
```

Out[16]:
```
array([  23,  500,   28, ..., 4808, 2263, 1342])
```

Entrée [17]:
```
df['Age'].isna().sum()
```

Out[17]:
```
0
```

Entrée [18]:
```
# Tracer la distribution des valeurs de la colonne 'Age' sous forme de Kernel 
# qui est une estimation de la distribution de probabilité continue des donnée
sns.kdeplot(df['Age'])
plt.show()
```

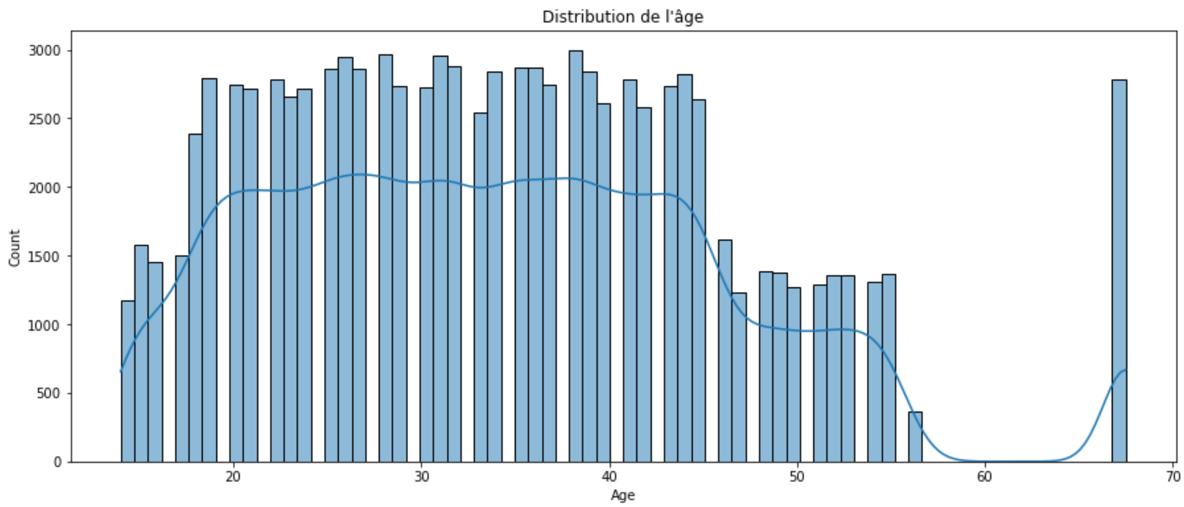


Entrée [19]:
```
# vérifier les outliers de la colonne 'Age'
check_outliers('Age',df)
handle_outliers('Age',df)
```

Entrée [20]:
```
check_outliers('Age',df)
```

Out[20]:
```
[]
```

Entrée [21]:
```
plt.figure(figsize=(15,6))
sns.histplot(x='Age', data= df, kde=True)
plt.title("Distribution de l'âge")
plt.show()
```



3. Occupation

Entrée [22]:
```
df['Occupation'].value_counts()
```

Out[22]:
```
_______          7062
Lawyer           6575
Architect        6355
Engineer         6350
Scientist        6299
Mechanic         6291
Accountant       6271
Developer        6235
Media_Manager    6232
Teacher          6215
Entrepreneur     6174
Doctor           6087
Journalist       6085
Manager          5973
Musician         5911
Writer           5885
Name: Occupation, dtype: int64
```

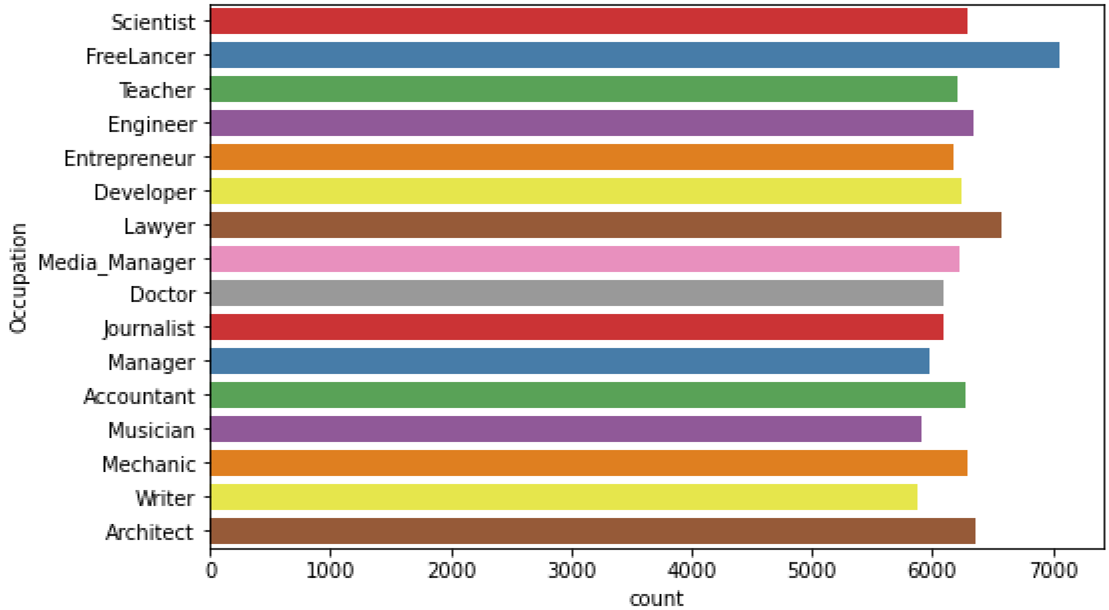
Entrée [23]:
```
df['Occupation'] = df['Occupation'].replace('_______','FreeLancer')
```

Entrée [24]:
```
df['Occupation'].unique()
```

Out[24]:
```
array(['Scientist', 'FreeLancer', 'Teacher', 'Engineer', 'Entrepreneur',
       'Developer', 'Lawyer', 'Media_Manager', 'Doctor', 'Journalist',
       'Manager', 'Accountant', 'Musician', 'Mechanic', 'Writer',
       'Architect'], dtype=object)
```

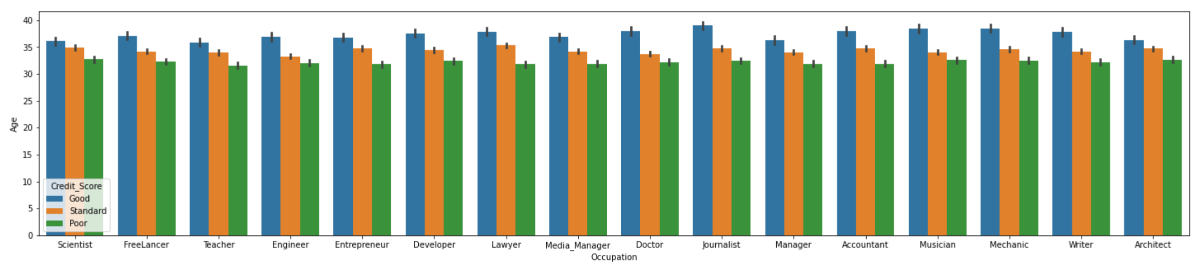
Entrée [25]:

```
plt.figure(figsize=(8,5))
sns.countplot(y="Occupation", data=df, palette="Set1")
plt.show()
```



Entrée [26]:

```
plt.figure(figsize=(25,5))
sns.barplot(x='Occupation', y='Age', data=df, hue='Credit_Score')
plt.show()
```



Entrée [27]:
```
df['Occupation'].map({
    'Scientist':0,
    'Engineer':2,
    'Teacher':3,
    'Entrepreneur':4,
    'Developer':5,
    'Lawyer':6,
    'Media_Manager':7,
    'Doctor':8,
    'Journalist':9,
    'Manager':10,
    'Accountant':11,
    'Musician':12,
    'Mechanic':13,
    'Writer':14,
    'Architect':15
})
```

Out[27]:
```
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
         ...
99995    13.0
99996    13.0
99997    13.0
99998    13.0
99999    13.0
Name: Occupation, Length: 100000, dtype: float64
```

Entrée [28]:
```
df['Occupation'].value_counts()
```

Out[28]:
```
FreeLancer       7062
Lawyer           6575
Architect        6355
Engineer         6350
Scientist        6299
Mechanic         6291
Accountant       6271
Developer        6235
Media_Manager    6232
Teacher          6215
Entrepreneur     6174
Doctor           6087
Journalist       6085
Manager          5973
Musician         5911
Writer           5885
Name: Occupation, dtype: int64
```

4. Annual_Income

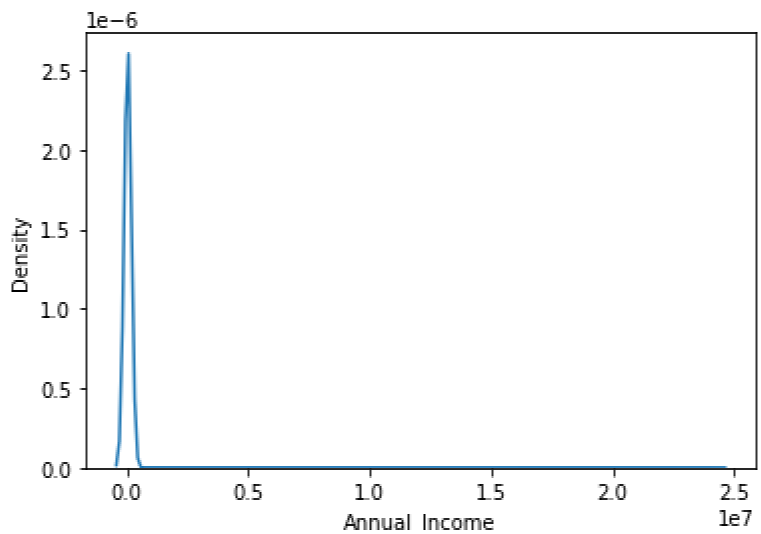
Entrée [29]:
```
df['Annual_Income'].value_counts()
```

Out[29]:
```
36585.12      16
20867.67      16
17273.83      16
9141.63       15
33029.66      15
              ..
20269.93_      1
15157.25_      1
44955.64_      1
76650.12_      1
4262933.0      1
Name: Annual_Income, Length: 18940, dtype: int64
```

Entrée [30]:
```
df['Annual_Income'] = df['Annual_Income'].str.replace('_','')
df['Annual_Income'] = df['Annual_Income'].str.replace('-','')
df['Annual_Income'] = df['Annual_Income'].astype(float)
df['Annual_Income'].unique()
```

Out[30]:
```
array([ 19114.12,  34847.84, 143162.64, ...,  37188.1 ,  20002.88,
        39628.99])
```

Entrée [31]:
```
sns.kdeplot(df['Annual_Income'])
plt.show()
```



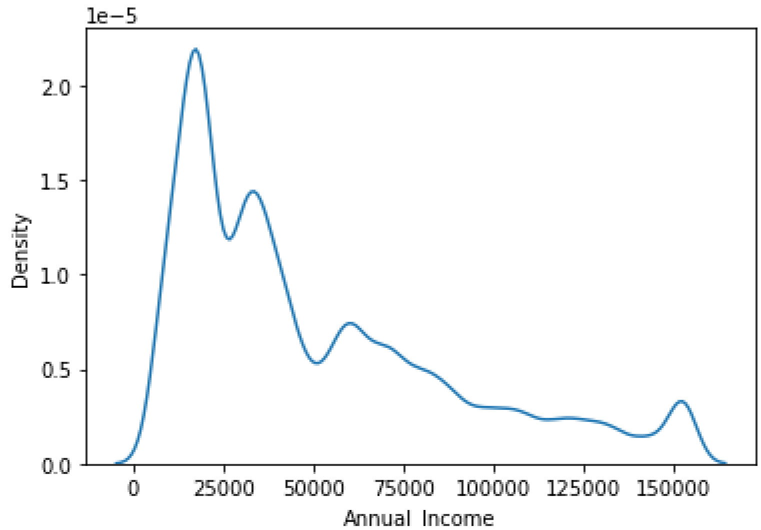
Entrée [32]:
```
df['Annual_Income'].describe()
```

Out[32]:
```
count    1.000000e+05
mean     1.764157e+05
std      1.429618e+06
min      7.005930e+03
25%      1.945750e+04
50%      3.757861e+04
75%      7.279092e+04
max      2.419806e+07
Name: Annual_Income, dtype: float64
```

Entrée [33]:
```
check_outliers('Annual_Income',df)
handle_outliers('Annual_Income',df)
check_outliers('Annual_Income',df)
```

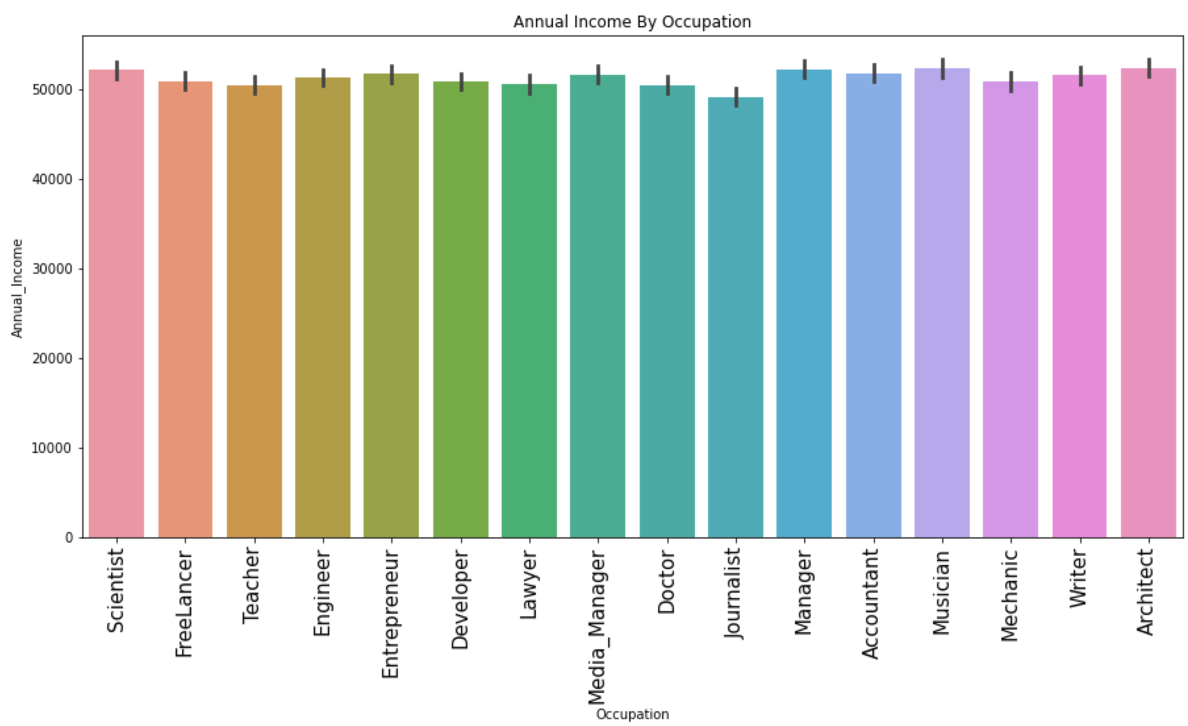
Out[33]:
```
[]
```

Entrée [34]:
```
sns.kdeplot(df['Annual_Income'])
plt.show()
```



Entrée [35]:
```
# figure du revenu annuel par profession
plt.figure(figsize=(15,7))
plt.xticks(fontsize=15, rotation = "vertical")
sns.barplot(y=df['Annual_Income'], x=df["Occupation"])
plt.title("Annual Income By Occupation")
```

Out[35]:
```
Text(0.5, 1.0, 'Annual Income By Occupation')
```



5. Monthly_Inhand_Salary

Entrée [36]:
```
df['Monthly_Inhand_Salary'].unique()
```

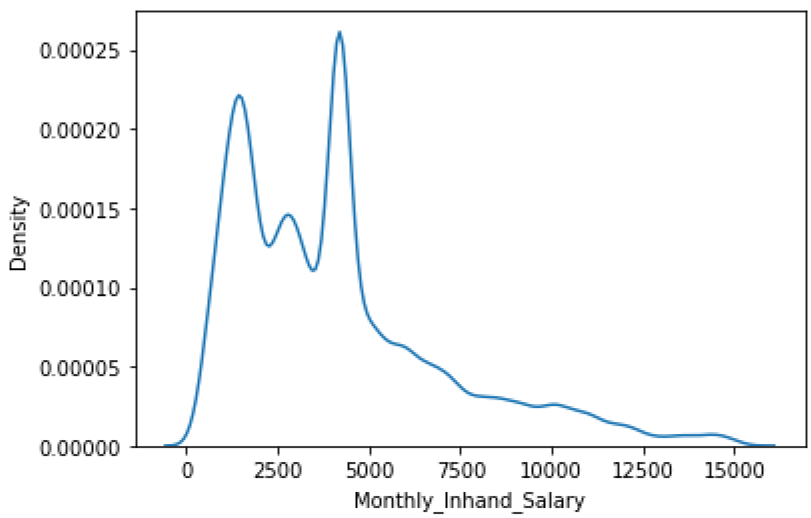
Out[36]:
```
array([1824.84333333,           nan, 3037.98666667, ..., 3097.00833333,
       1929.90666667, 3359.41583333])
```

Entrée [37]:
```
# Remplacer les valeurs qui ne sont pas un entier par la valeur moyenne de la
df['Monthly_Inhand_Salary'].fillna(df['Monthly_Inhand_Salary'].mean(), inplace
```

Entrée [38]:
```
# Vérifier s'il y a encore des valeurs 'Not A Number'
df['Monthly_Inhand_Salary'].isna().sum()
```

Out[38]:
```
0
```

Entrée [39]:
```
sns.kdeplot(df['Monthly_Inhand_Salary'])
plt.show()
```



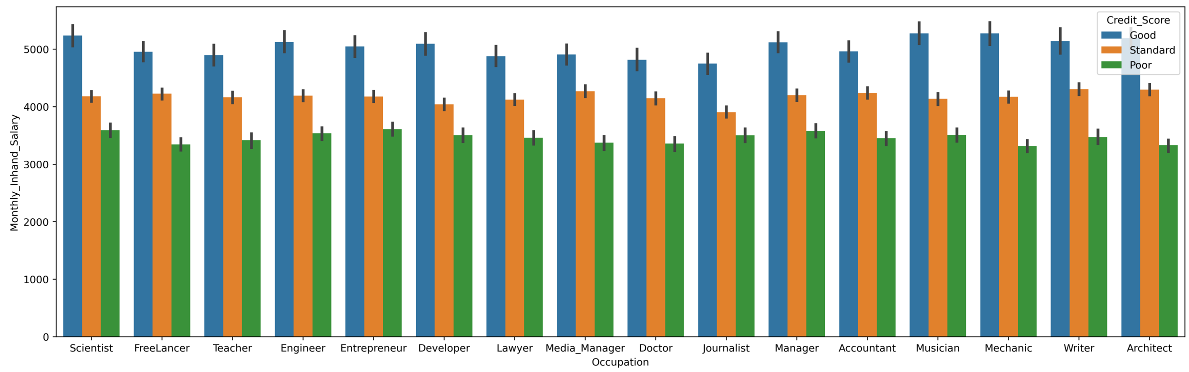
Entrée [40]:
```
check_outliers('Monthly_Inhand_Salary', df)
handle_outliers('Monthly_Inhand_Salary', df)
check_outliers('Monthly_Inhand_Salary', df)
```

Out[40]:
```
[]
```

Entrée [41]:
```
df['Monthly_Inhand_Salary'].describe()
```

Out[41]:
```
count    100000.000000
mean       4121.979810
std        2733.865830
min         303.645417
25%        1792.084167
50%        3852.736667
75%        5371.525000
max       10740.686250
Name: Monthly_Inhand_Salary, dtype: float64
```

Entrée [42]:
```
plt.figure(figsize = (20,6), dpi=400)
sns.barplot(x='Occupation', y='Monthly_Inhand_Salary', data=df, hue='Credit_Sc
plt.show()
```



6. Num_Bank_Accounts

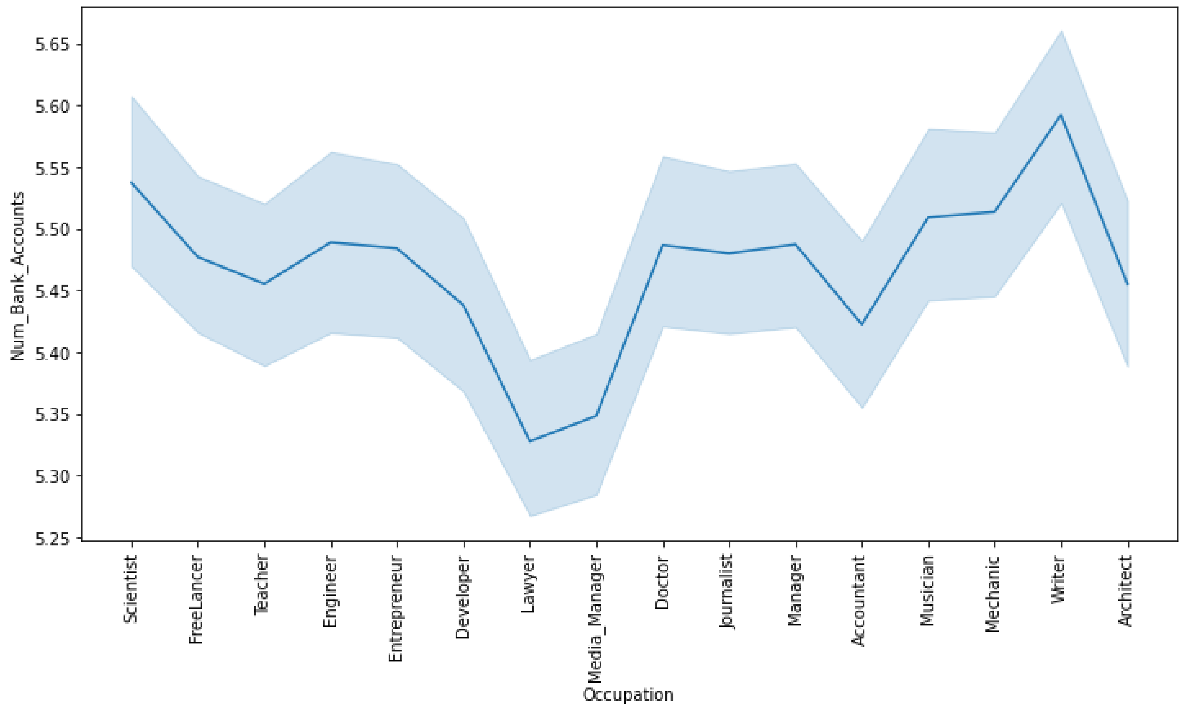
Entrée [43]:
```
df['Num_Bank_Accounts'].value_counts()
```

Out[43]:
```
6       13001
7       12823
8       12765
4       12186
5       12118
        ...
1626        1
1470        1
887         1
211         1
697         1
Name: Num_Bank_Accounts, Length: 943, dtype: int64
```

Entrée [44]:
```
handle_outliers('Num_Bank_Accounts', df)
check_outliers('Num_Bank_Accounts', df)
```

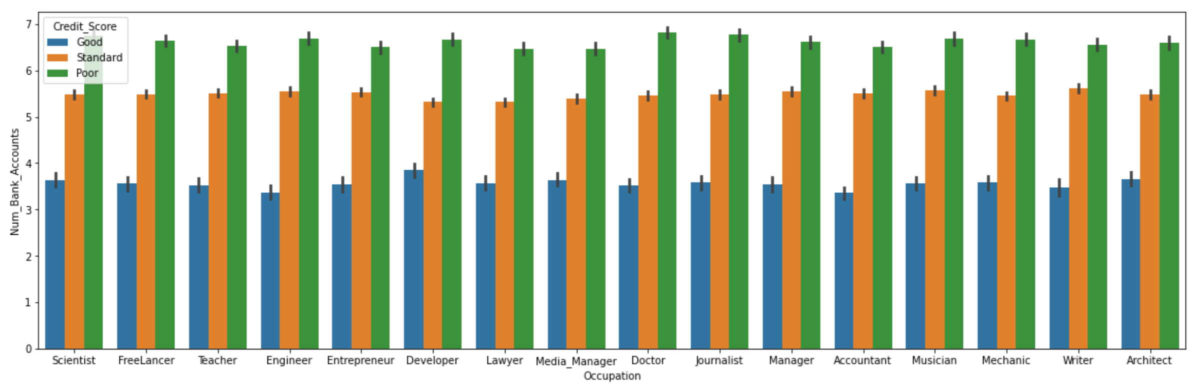
Out[44]:
```
[]
```

Entrée [45]:
```
plt.figure(figsize=(12,6))
plt.xticks(fontsize=10, rotation='vertical')
sns.lineplot(data=df, x='Occupation', y='Num_Bank_Accounts')
plt.show()
```



Entrée [46]:
```
df[df['Num_Credit_Card'] < 0]= 0
```

Entrée [47]:
```
plt.figure(figsize=(20,6))
sns.barplot(x=df['Occupation'], y= df['Num_Bank_Accounts'], data=df, hue='Cred
plt.show()
```



7. Num_Credit_Card

Entrée [48]:
```
df['Num_Credit_Card'].value_counts()
```

Out[48]:
```
5       18459
7       16615
6       16559
4       14030
3       13277
        ...
791         1
1118        1
657         1
640         1
679         1
Name: Num_Credit_Card, Length: 1179, dtype: int64
```

Entrée [49]:
```
df['Num_Credit_Card'].describe()
```

Out[49]:
```
count    100000.00000
mean         22.47443
std         129.05741
min           0.00000
25%           4.00000
50%           5.00000
75%           7.00000
max        1499.00000
Name: Num_Credit_Card, dtype: float64
```

Entrée [50]:
```
handle_outliers('Num_Credit_Card',df)
check_outliers('Num_Credit_Card',df)
```

Out[50]:
```
[]
```

8. Interest_Rate

Entrée [51]:
```
df['Interest_Rate'].value_counts()
```

Out[51]:
```
8       5012
5       4979
6       4721
12      4540
10      4540
        ...
4995       1
1899       1
2120       1
5762       1
5729       1
Name: Interest_Rate, Length: 1750, dtype: int64
```

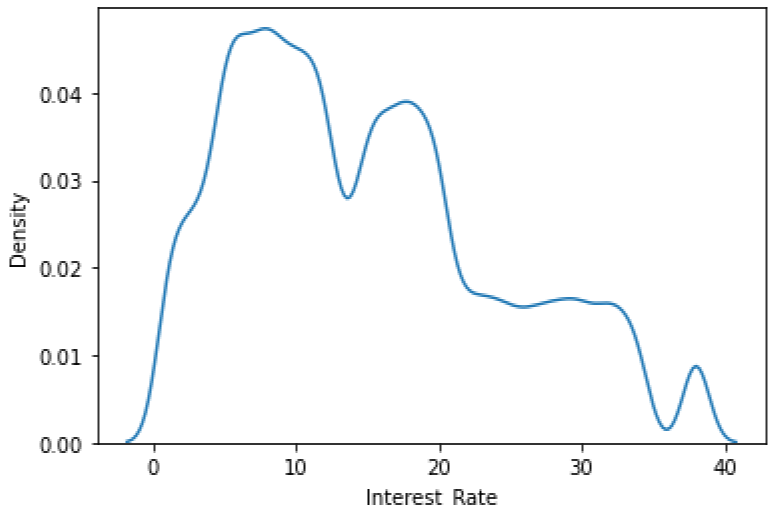
Entrée [52]:
```
df['Interest_Rate'].isna().sum()
```

Out[52]:
```
0
```

Entrée [53]:
```
handle_outliers('Interest_Rate',df)
check_outliers('Interest_Rate',df)
```

Out[53]: []

Entrée [54]:
```
sns.kdeplot(x='Interest_Rate',data=df)
plt.show()
```



9. Num_of_Loan

Entrée [55]:
```
df['Num_of_Loan'].unique()
```

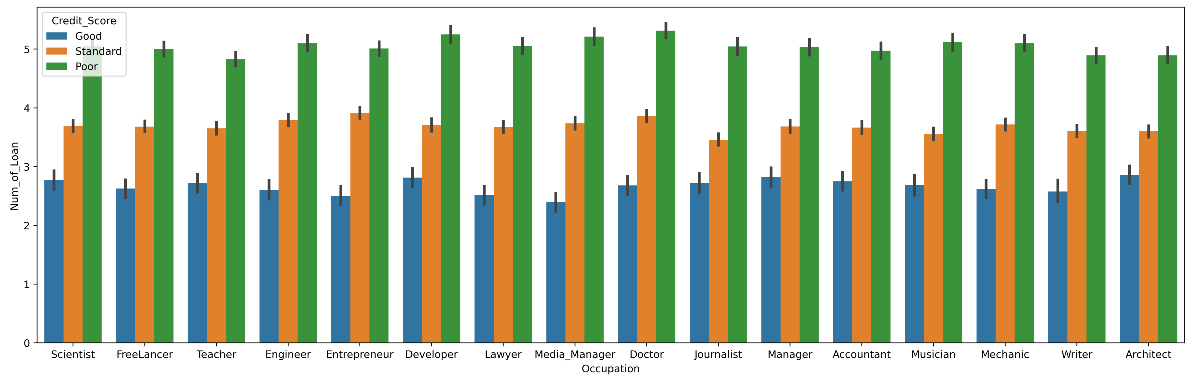
Out[55]:
```
array(['4', '1', '3', '967', '-100', '0', '0_', '2', '3_', '2_', '7', '5',
       '5_', '6', '8', '8_', '9', '9_', '4_', '7_', '1_', '1464', '6_',
       '622', '352', '472', '1017', '945', '146', '563', '341', '444',
       '720', '1485', '49', '737', '1106', '466', '728', '313', '843',
       '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'], dtype=object)
```

Entrée [56]:
```
df['Num_of_Loan']= df['Num_of_Loan'].str.replace('_','')
df['Num_of_Loan']= df['Num_of_Loan'].str.replace('-','')
df['Num_of_Loan']= df['Num_of_Loan'].astype(int)
```

Entrée [57]:
```
handle_outliers('Num_of_Loan',df)
check_outliers('Num_of_Loan', df)
```

Out[57]: []

Entrée [58]:
```
plt.figure(figsize=(20,6), dpi=400)
sns.barplot(x='Occupation', y='Num_of_Loan',data=df,hue='Credit_Score')
plt.show()
```



10. Delay_from_due_date

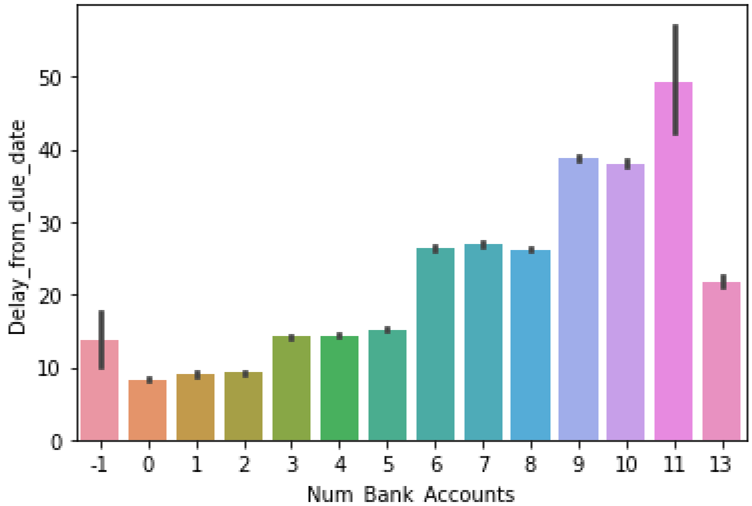
Entrée [59]:
```
df['Delay_from_due_date'].unique()
```

Out[59]:
```
array([ 3, -1,  5,  6,  8,  7, 13, 10,  0,  4,  9,  1, 12, 11, 30, 31, 34,
       27, 14,  2, -2, 16, 17, 15, 23, 22, 21, 18, 19, 52, 51, 48, 53, 26,
       43, 28, 25, 20, 47, 46, 49, 24, 61, 29, 50, 58, 45, 59, 55, 56, 57,
       54, 62, 65, 64, 67, 36, 41, 33, 32, 39, 44, 42, 60, 35, 38, -3, 63,
       40, 37, -5, -4, 66], dtype=int64)
```

Entrée [60]:
```
# Convertir la valeur négatif en 0
df[df['Delay_from_due_date']<0]=0
df['Delay_from_due_date'].unique()
```

Out[60]:
```
array([ 3,  0,  5,  6,  8,  7, 13, 10,  4,  9,  1, 12, 11, 30, 31, 34, 27,
       14,  2, 16, 17, 15, 23, 22, 21, 18, 19, 52, 51, 48, 53, 26, 43, 28,
       25, 20, 47, 46, 49, 24, 61, 29, 50, 58, 45, 59, 55, 56, 57, 54, 62,
       65, 64, 67, 36, 41, 33, 32, 39, 44, 42, 60, 35, 38, 63, 40, 37, 66],
      dtype=int64)
```

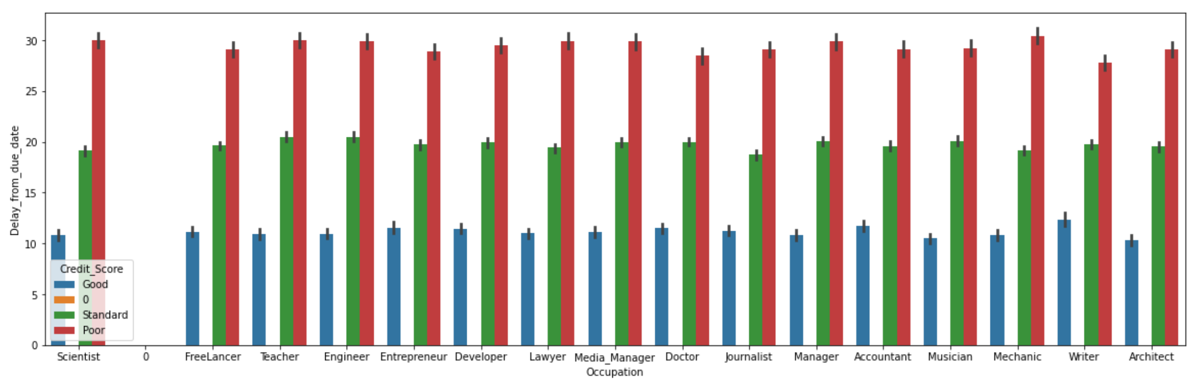
Entrée [61]:
```
sns.barplot(x=df['Num_Bank_Accounts'], y=df['Delay_from_due_date'], data=df)
plt.show()
```



Entrée [62]:
```
handle_outliers('Delay_from_due_date',df)
check_outliers('Delay_from_due_date',df)
```

Out[62]: []

Entrée [63]:
```
plt.figure(figsize=(20,6))
sns.barplot(x='Occupation', y='Delay_from_due_date',data=df, hue='Credit_Score
plt.show()
```



11. Num_of_Delayed_Payment

Entrée [64]:
```
df['Num_of_Delayed_Payment'].unique()
```

Out[64]:
```
array(['7', 0, '4', nan, '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', '3316', '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', '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', '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', '1086', '2216', '1087_',
       '2457', '3522', '3274', '3488', '2854', '238', '351', '3706',
       '4280', '4095', '2926', '1329', '3370', '283', '1392', '1743',
       '2429', '974', '3156', '1133', '4388', '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', '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'], dtype=object)
```

Entrée [65]:
```
df['Num_of_Delayed_Payment']= df['Num_of_Delayed_Payment'].str.replace('_','')
df['Num_of_Delayed_Payment']= df['Num_of_Delayed_Payment'].str.replace('-','')
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].astype(float)
```

Entrée [66]:
```
handle_outliers('Num_of_Delayed_Payment',df)
check_outliers('Num_of_Delayed_Payment',df)
```

Out[66]:
```
[]
```

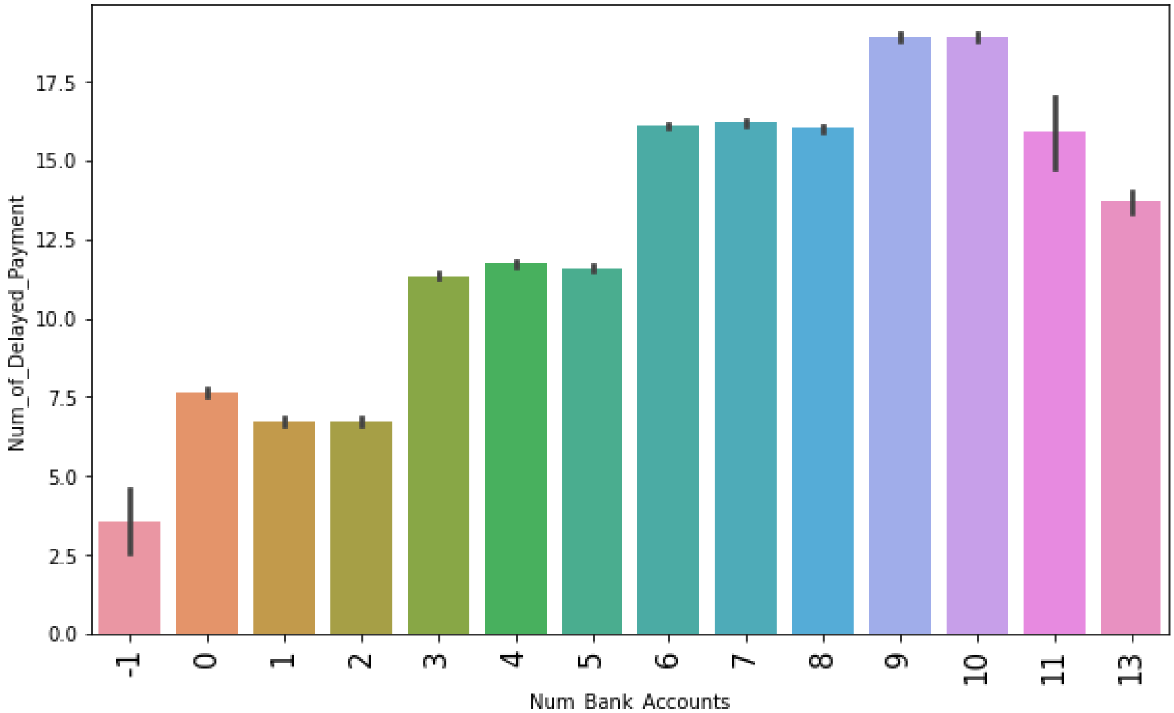
Entrée [67]:
```
df['Num_of_Delayed_Payment'].isna().sum()
```

Out[67]:
```
7556
```

Entrée [68]:
```
df['Num_of_Delayed_Payment'].fillna(df['Num_of_Delayed_Payment'].mean(), inpla
df['Num_of_Delayed_Payment'].isna().sum()
```

Out[68]:
```
0
```

Entrée [69]:
```
plt.figure(figsize=(10,6))
sns.barplot(x=df['Num_Bank_Accounts'], y=df['Num_of_Delayed_Payment'],data=df)
plt.xticks(fontsize=15, rotation='vertical')
plt.show()
```



12. Changed_Credit_Limit

Entrée [70]:
```
df['Changed_Credit_Limit'].unique()
```

Out[70]:
```
array(['11.27', 0, '_', ..., '17.509999999999998', '25.16', '21.17'],
      dtype=object)
```

Entrée [71]:
```
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].str.replace('_','0')
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].str.replace('-', '')
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].astype(float)
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].replace('0', np.nan)
df['Changed_Credit_Limit'] = df['Changed_Credit_Limit'].replace(np.nan, df['Ch
```

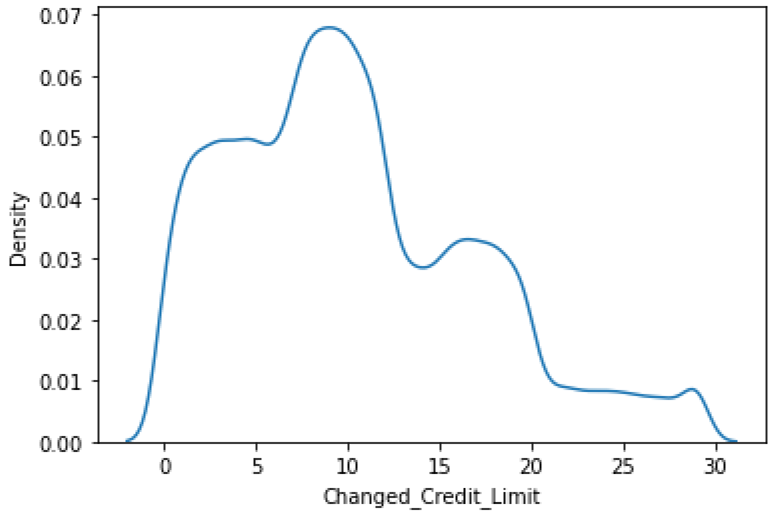
Entrée [72]:
```
df['Changed_Credit_Limit'].unique()
```

Out[72]:
```
array([11.27      , 10.27100615,  0.        , ..., 17.51      ,
       25.16      , 21.17      ])
```

Entrée [73]:
```
check_outliers('Changed_Credit_Limit',df)
handle_outliers('Changed_Credit_Limit',df)
check_outliers('Changed_Credit_Limit',df)
```

Out[73]:
```
[]
```

Entrée [74]:
```
sns.kdeplot(df['Changed_Credit_Limit'])
plt.show()
```



13. Num_Credit_Inquiries

Entrée [75]:
```
df['Num_Credit_Inquiries'].unique()
```

Out[75]:
```
array([   4.,    0.,    2., ..., 1361.,  310.,   74.])
```

Entrée [76]:
```
df['Num_Credit_Inquiries'].isna().sum()
```

Out[76]:
```
1951
```

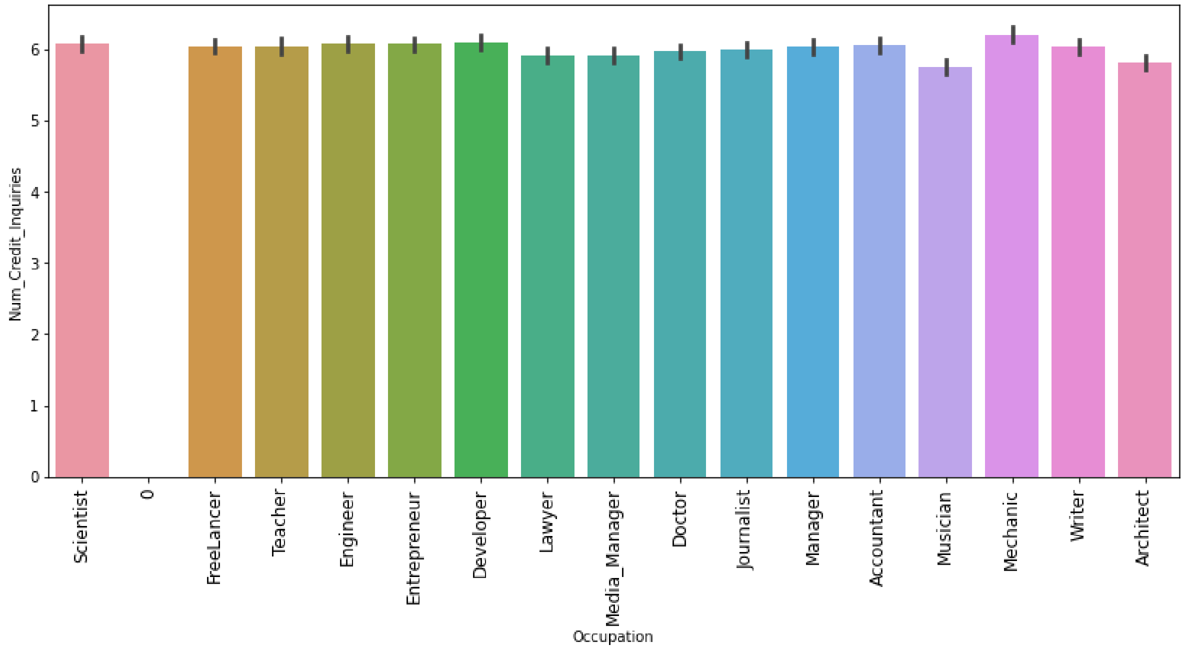
Entrée [77]:
```
df['Num_Credit_Inquiries'].fillna(df['Num_Credit_Inquiries'].median(),inplace=
df['Num_Credit_Inquiries'].isna().sum()
```

Out[77]:
```
0
```

Entrée [78]:
```
check_outliers('Num_Credit_Inquiries',df)
handle_outliers('Num_Credit_Inquiries',df)
check_outliers('Num_Credit_Inquiries',df)
```

Out[78]:
```
[]
```

Entrée [79]:
```
plt.figure(figsize=(14,6))
sns.barplot(x='Occupation', y='Num_Credit_Inquiries',data=df)
plt.xticks(fontsize=12,rotation='vertical')
plt.show()
```



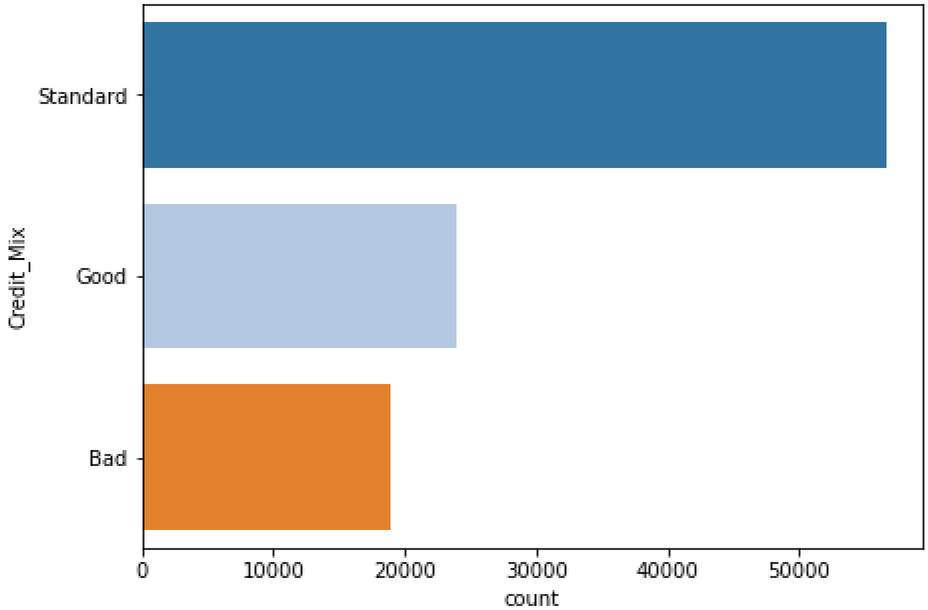
14. Credit_Mix

Entrée [80]:
```
df['Credit_Mix'].value_counts()
```

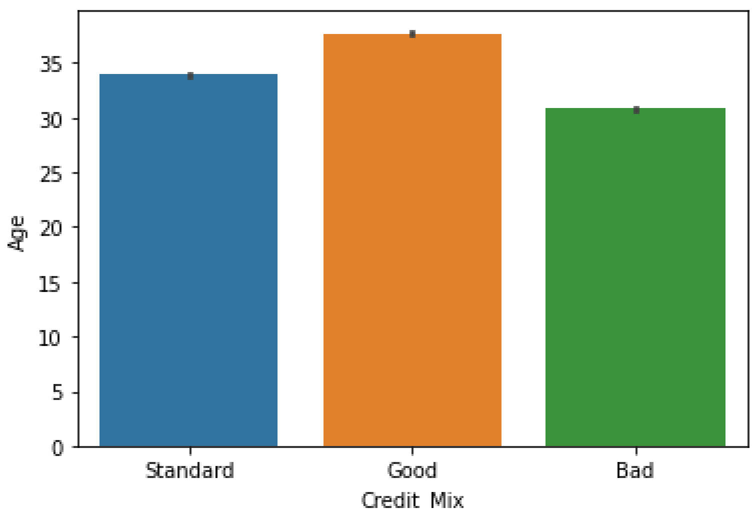
Out[80]:
```
Standard    36479
Good        23859
_           20082
Bad         18989
0             591
Name: Credit_Mix, dtype: int64
```

Entrée [81]:
```
df['Credit_Mix']= df['Credit_Mix'].str.replace('_','Standard')
```

Entrée [82]:
```
plt.figure(figsize=(7,5))
sns.countplot(y='Credit_Mix', data=df,palette="tab20")
plt.show()
```



Entrée [83]:
```
sns.barplot(x=df['Credit_Mix'],y=df['Age'], data=df)
plt.show()
```



Entrée [84]:
```
df['Credit_Mix']=df['Credit_Mix'].map({'Bad':1,'Standard':2,'Good':3})
```

Entrée [85]:
```
df['Credit_Mix'].isna().sum()
```

Out[85]: 591

Entrée [86]:
```
df['Credit_Mix'].fillna(df['Credit_Mix'].median(),inplace=True)
df['Credit_Mix'].isna().sum()
```

Out[86]: 0

15. Outstanding_Debt

Entrée [87]:
```
df['Outstanding_Debt'].unique()
```

Out[87]:
```
array(['809.98', 0, '605.03', ..., '3571.7_', '3571.7', '502.38'],
      dtype=object)
```

Entrée [88]:
```
df['Outstanding_Debt'] = df['Outstanding_Debt'].str.replace('_','')
df['Outstanding_Debt'] = df['Outstanding_Debt'].str.replace('-','')
df['Outstanding_Debt'] = df['Outstanding_Debt'].astype(float)
```

Entrée [89]:
```
check_outliers('Outstanding_Debt',df)
handle_outliers('Outstanding_Debt',df)
check_outliers('Outstanding_Debt',df)
```

Out[89]:
```
[]
```

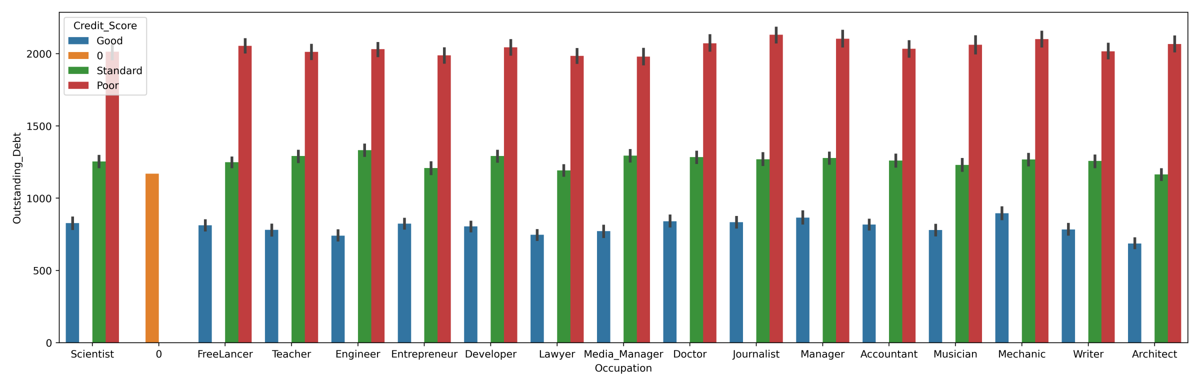
Entrée [90]:
```
df['Outstanding_Debt'].isna().sum()
```

Out[90]:
```
591
```

Entrée [91]:
```
df['Outstanding_Debt'].fillna(df['Outstanding_Debt'].median(), inplace=True)
df['Outstanding_Debt'].isna().sum()
```

Out[91]:
```
0
```

Entrée [92]:
```
plt.figure(figsize=(20,6),dpi=400)
sns.barplot(x='Occupation',y='Outstanding_Debt',data=df, hue='Credit_Score')
plt.show()
```

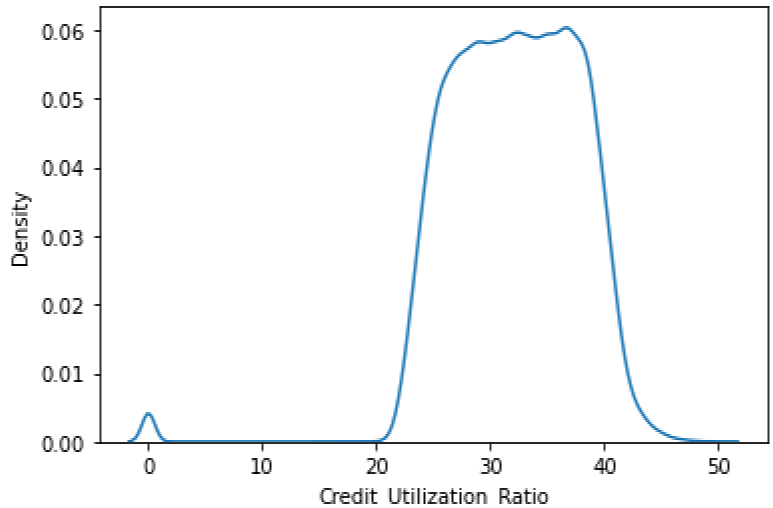


16. Credit_Utilization_Ratio

Entrée [93]:
```
df['Credit_Utilization_Ratio'].unique()
```

Out[93]:
```
array([26.82261962,  0.        , 28.60935202, ..., 41.25552226,
       33.63820798, 34.19246265])
```

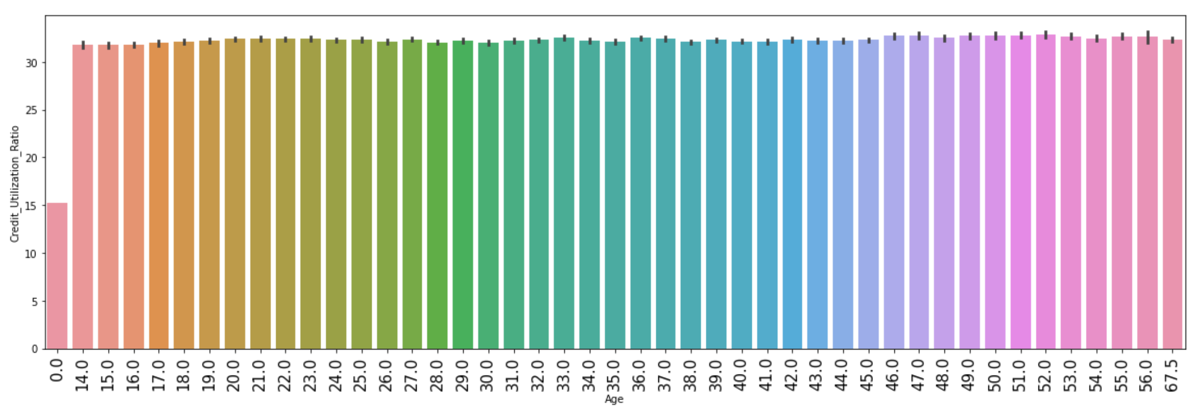
Entrée [94]:
```
sns.kdeplot(x=df['Credit_Utilization_Ratio'],data=df)
plt.show()
```



Entrée [95]:
```
check_outliers('Credit_Utilization_Ratio',df)
handle_outliers('Credit_Utilization_Ratio',df)
check_outliers('Credit_Utilization_Ratio',df)
```

Out[95]: []

Entrée [96]:
```
plt.figure(figsize=(20,6))
plt.xticks(fontsize=15, rotation='vertical')
sns.barplot(x=df['Age'],y=df['Credit_Utilization_Ratio'], data=df)
plt.show()
```



17. Payment_Behaviour

Entrée [97]:
```
df['Payment_Behaviour'].value_counts()
```

Out[97]:
```
Low_spent_Small_value_payments      25391
High_spent_Medium_value_payments    17445
Low_spent_Medium_value_payments     13761
High_spent_Large_value_payments     13616
High_spent_Small_value_payments     11280
Low_spent_Large_value_payments      10363
!@9#%8                               7553
0                                     591
Name: Payment_Behaviour, dtype: int64
```

Entrée [98]:
```
df['Payment_Behaviour'] = df['Payment_Behaviour'].str.replace('!@9#%8', 'Low_s
df['Payment_Behaviour'].unique()
```

Out[98]:
```
array(['High_spent_Small_value_payments', nan,
       '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'], dtype=object)
```

Entrée [99]:
```
df['Payment_Behaviour'].isna().sum()
```

Out[99]:
```
591
```

Entrée [100]:
```
df['Payment_Behaviour']=df['Payment_Behaviour'].map({'High_spent_Small_value_
                                                     'Low_spent_Large_value_p
                                                     'Low_spent_Medium_value_
                                                     'Low_spent_Small_value_p
                                                     'High_spent_Medium_value
                                                     'High_spent_Large_value_
                                                     })
```

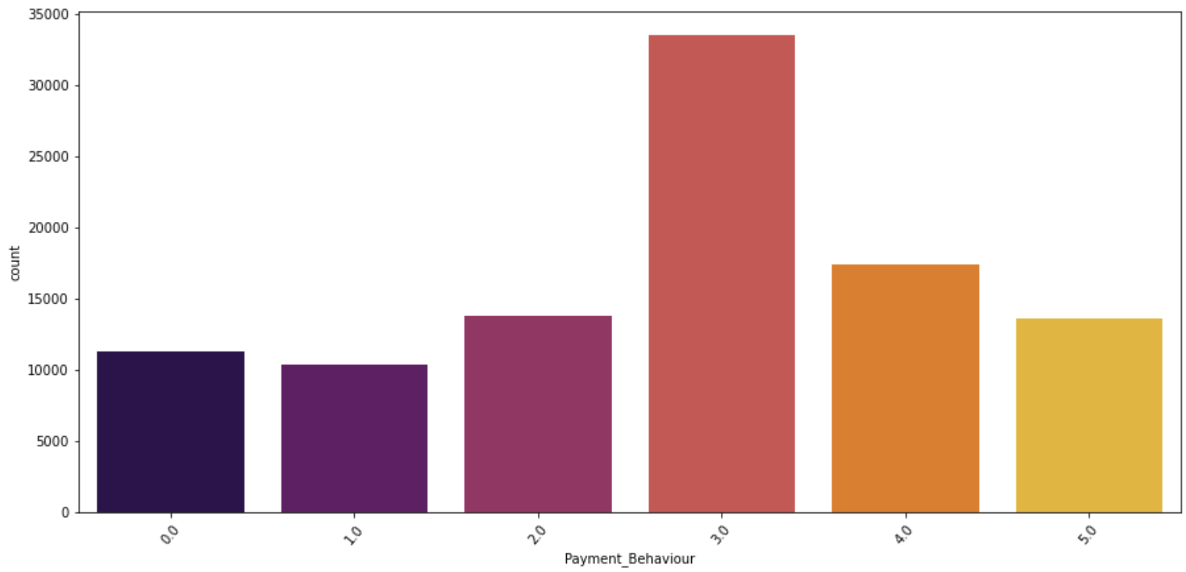
Entrée [101]:
```
df['Payment_Behaviour'].fillna(df['Payment_Behaviour'].median(),inplace=True)
df['Payment_Behaviour'].isna().sum()
```

Out[101]:
```
0
```

Entrée [102]:
```
df['Payment_Behaviour'].value_counts()
```

Out[102]:
```
3.0    33535
4.0    17445
2.0    13761
5.0    13616
0.0    11280
1.0    10363
Name: Payment_Behaviour, dtype: int64
```

Entrée [103]:
```
plt.figure(figsize=(15,7))
sns.countplot(x='Payment_Behaviour', data = df, palette='inferno')
plt.xticks(rotation=50)
plt.show()
```



18. Total_EMI_per_month

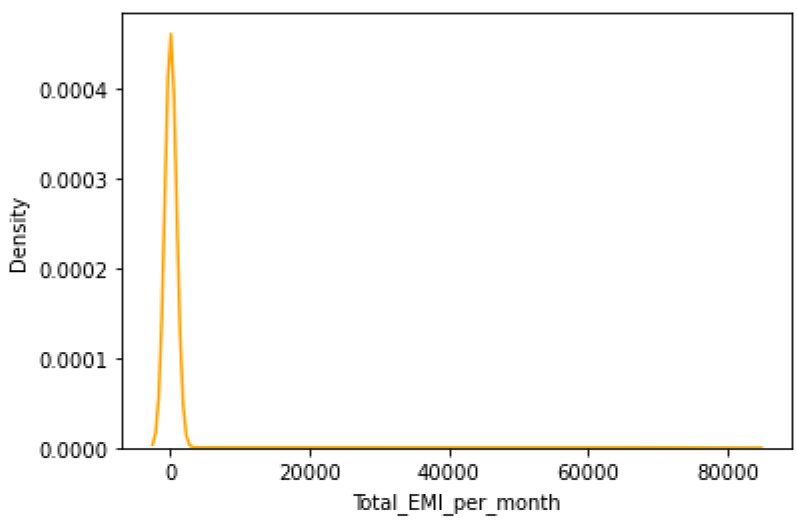
Entrée [104]:
```
df['Total_EMI_per_month'].unique()
```

Out[104]:
```
array([4.95749492e+01, 0.00000000e+00, 1.88162146e+01, ...,
       1.21120000e+04, 3.51040226e+01, 5.86380000e+04])
```

Entrée [105]:
```
df['Total_EMI_per_month'].isna().sum()
```

Out[105]:
```
0
```

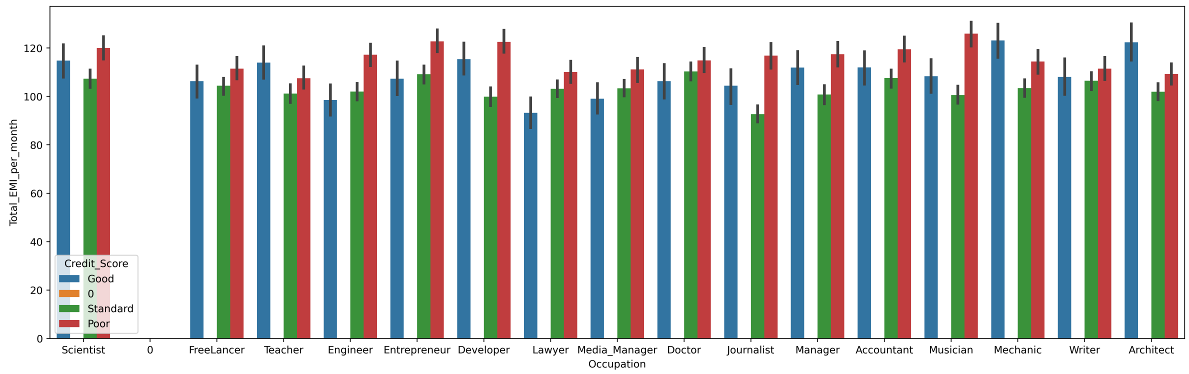
Entrée [106]:
```
sns.kdeplot(x=df['Total_EMI_per_month'], data=df, color="orange")
plt.show()
```



Entrée [107]:
```
check_outliers('Total_EMI_per_month',df)
handle_outliers('Total_EMI_per_month',df)
check_outliers('Total_EMI_per_month',df)
```

Out[107]: []

Entrée [108]:
```
plt.figure(figsize=(20,6),dpi=400)
sns.barplot(x='Occupation', y='Total_EMI_per_month', data=df, hue='Credit_Sco
plt.show()
```



19. Colonne 'Credit_History_Age'

Entrée [109]:
```
df['Credit_History_Age'].isna().sum()
```

Out[109]: 8965

Entrée [110]:
```
# Remplir les valeurs manquantes par la valeur qui apprait le plus fréquemmen
df['Credit_History_Age'].fillna(df['Credit_History_Age'].mode(), inplace=True
df['Credit_History_Age'].isna().sum()
```

Out[110]: 8965

Entrée [111]:
```
# Diviser la colonne 'Credit_History_Age' en deux parties (year et month)
df['Credit_History_Year'], df['Credit_History_Month'] = df['Credit_History_Ag
# Supprimer la colonne 'Credit_History_Age'
df.drop('Credit_History_Age', axis=1, inplace=True)
df['Credit_History_Year'] = df['Credit_History_Year'].str.replace('Years','')
df['Credit_History_Month'] = df['Credit_History_Month'].str.replace('Months',
```

Entrée [112]:
```
df['Credit_History_Year']
```

Out[112]:
```
0         22
1          NaN
2         22
3         22
4         22
          ...
99995     31
99996     31
99997     31
99998     31
99999     31
Name: Credit_History_Year, Length: 100000, dtype: object
```

Entrée [113]:
```
df['Credit_History_Month']
```

Out[113]:
```
0           1
1          NaN
2           3
3           4
4           5
          ...
99995       6
99996       7
99997       8
99998       9
99999      10
Name: Credit_History_Month, Length: 100000, dtype: object
```

20. Payment_of_Min_Amount

Entrée [114]:
```
df['Payment_of_Min_Amount'].unique()
```

Out[114]:
```
array(['No', 0, 'NM', 'Yes'], dtype=object)
```

Entrée [115]:
```
df['Payment_of_Min_Amount'].value_counts()
```

Out[115]:
```
Yes    52326
No     35138
NM     11945
0        591
Name: Payment_of_Min_Amount, dtype: int64
```

Entrée [116]:
```
df['Payment_of_Min_Amount'].isna().sum()
```

Out[116]:
```
0
```

21. Amount_invested_monthly

Entrée [117]:
```
df['Amount_invested_monthly'].value_counts()
```

Out[117]:
```
__10000__           4280
0                    591
0.0                  169
157.6434518748769      1
224.43978111915573     1
                     ...
140.80972223052834     1
38.73937670100975      1
109.296681189146       1
33.6098814431885       1
167.1638651610451      1
Name: Amount_invested_monthly, Length: 90507, dtype: int64
```

Entrée [118]:
```
df['Amount_invested_monthly']= df['Amount_invested_monthly'].str.replace('__'
df['Amount_invested_monthly']= df['Amount_invested_monthly'].astype(float)
```

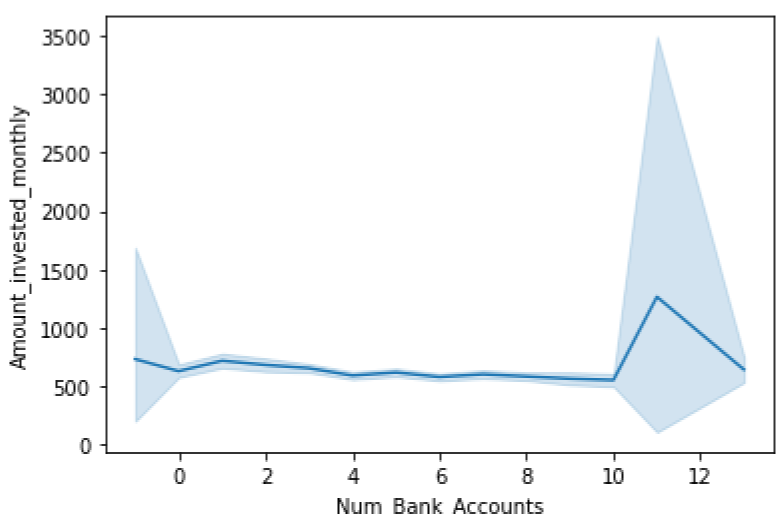
Entrée [119]:
```
df['Amount_invested_monthly'].isna().sum()
```

Out[119]: 5047

Entrée [120]:
```
df['Amount_invested_monthly'].fillna(df['Amount_invested_monthly'].median(),i
df['Amount_invested_monthly'].isna().sum()
```

Out[120]: 0

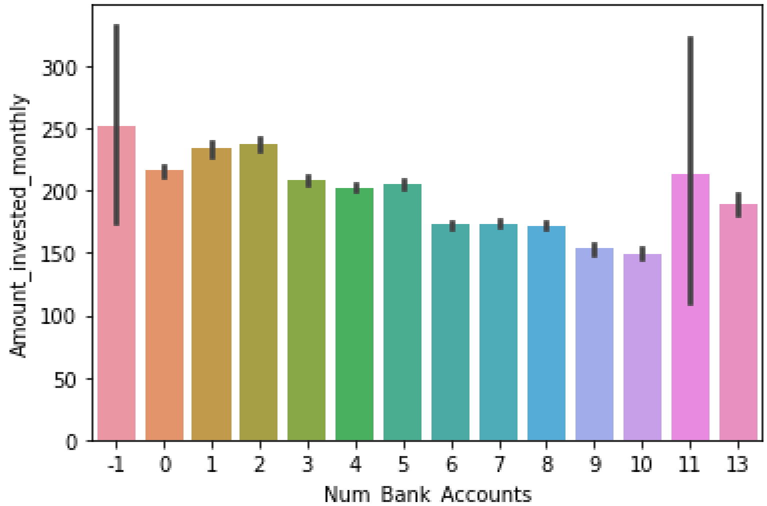
Entrée [121]:
```
sns.lineplot(x=df['Num_Bank_Accounts'],y=df['Amount_invested_monthly'],data=d
plt.show()
```



Entrée [122]:
```
handle_outliers('Amount_invested_monthly',df)
check_outliers('Amount_invested_monthly',df)
```

Out[122]: []

Entrée [123]:
```
sns.barplot(x=df['Num_Bank_Accounts'], y=df['Amount_invested_monthly'],data=d
plt.show()
```



22. Colonne 'Monthly_Balance'

Entrée [124]:
```
df['Monthly_Balance'].value_counts()
```

Out[124]:
```
0                                    591
__-333333333333333333333333333__       9
342.8948382302856                      1
305.3244921836277                      1
343.5103089241464                      1
                                    ...
278.8720257394474                      1
376.7024623690405                      1
321.2336043357731                      1
373.29270287694055                     1
393.6736955618808                      1
Name: Monthly_Balance, Length: 98210, dtype: int64
```

Entrée [125]:
```
# Convertir la colonne en numériques, si une valeur ne peut pas être converti
#df["Monthly_Balance"] = pd.to_numeric(df["Monthly_Balance"], errors="coerce"
#month_mean=df["Monthly_Balance"].mean()
#df["Monthly_Balance"].fillna(month_mean, inplace=True)
#df['Monthly_Balance'].isna().sum()
df['Monthly_Balance']=df['Monthly_Balance'].str.replace('__','')
df['Monthly_Balance']=df['Monthly_Balance'].astype(float)
```

Entrée [126]:
```
df['Monthly_Balance'].isna().sum()
```

Out[126]:
```
3441
```

Entrée [127]:
```
df['Monthly_Balance'].fillna(df['Monthly_Balance'].median(), inplace=True)
df['Monthly_Balance'].isna().sum()
```

Out[127]: 0

Entrée [128]:
```
check_outliers('Monthly_Balance',df)
```

Out[128]:
```
[1043.3159778669492,
 998.8692967863226,
 810.7821526659284,
 963.9215811205684,
 968.5555173846187,
 895.494583180492,
 796.2349097481042,
 858.462474411158,
 1038.5694068321734,
 899.1987716145285,
 963.2548189998564,
 1140.0673399198365,
 802.3004421328528,
 785.2583558699787,
 772.411908624267,
 792.0256603398883,
 854.5248768604907,
 823.7133773417005,
 878.2514462337779,
```

Entrée [129]:
```
handle_outliers('Monthly_Balance',df)
check_outliers('Monthly_Balance',df)
```

Out[129]: []

23. Credit History Month

Entrée [130]:
```
df['Credit_History_Month'].unique()
```

Out[130]:
```
array([' 1 ', nan, ' 3 ', ' 4 ', ' 5 ', ' 6 ', ' 7 ', ' 8 ', ' 9 ',
       ' 10 ', ' 11 ', ' 0 ', ' 2 '], dtype=object)
```

Entrée [131]:
```
df['Credit_History_Month'].isna().sum()
```

Out[131]: 9556

Entrée [132]:
```
df['Credit_History_Month'].fillna(df['Credit_History_Month'].median(), inplac
df['Credit_History_Month'].isna().sum()
```

Out[132]: 0

24. Credit History Year

Entrée [133]:
```
df['Credit_History_Year'].unique()
```

Out[133]:
```
array(['22  ', nan, '26  ', '27  ', '17  ', '18  ', '30  ', '31  ',
       '32  ', '14  ', '15  ', '21  ', '19  ', '25  ', '8  ', '9  ',
       '16  ', '29  ', '6  ', '7  ', '10  ', '33  ', '12  ', '13  ',
       '28  ', '24  ', '1  ', '11  ', '20  ', '0  ', '5  ', '2  ', '3  ',
       '23  ', '4  '], dtype=object)
```

Entrée [134]:
```
df['Credit_History_Year'].isna().sum()
```

Out[134]:
```
9556
```

Entrée [135]:
```
df['Credit_History_Year'].fillna(df['Credit_History_Year'].median(), inplace=
df['Credit_History_Year'].isna().sum()
```

Out[135]:
```
0
```

25. Credit_Score

Entrée [136]:
```
df['Credit_Score'].value_counts()
```

Out[136]:
```
Standard    52961
Poor        28949
Good        17499
0             591
Name: Credit_Score, dtype: int64
```

Entrée [137]:
```
df['Credit_Score']= df['Credit_Score'].map({'Poor':0,'Standard':1,'Good':2})
```

Entrée [138]:
```
df['Credit_Score'].isna().sum()
```

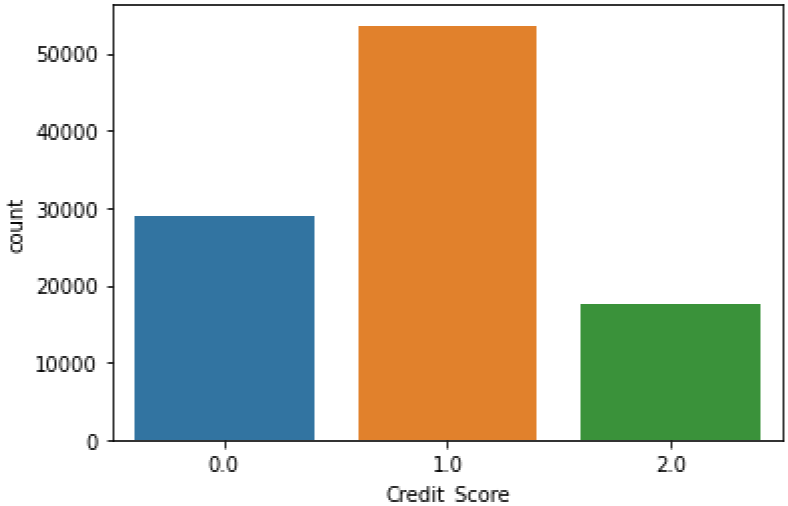
Out[138]:
```
591
```

Entrée [139]:
```
df['Credit_Score'].fillna(df['Credit_Score'].median(), inplace=True)
```

Entrée [140]:
```
df['Credit_Score'].isna().sum()
```

Out[140]:
```
0
```

Entrée [141]:
```
sns.countplot(df['Credit_Score'])
plt.show()
```



Entrée [142]:
```
# Créer des variables indicatrices pour la colonne 'Occupation'
df = pd.get_dummies(df, columns=['Occupation'],drop_first=True)
df = pd.get_dummies(df)
```

Entrée [143]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 88 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   Month                       100000 non-null  int64  
 1   Age                         100000 non-null  float64
 2   Annual_Income               100000 non-null  float64
 3   Monthly_Inhand_Salary       100000 non-null  float64
 4   Num_Bank_Accounts           100000 non-null  int64  
 5   Num_Credit_Card             100000 non-null  float64
 6   Interest_Rate               100000 non-null  int64  
 7   Num_of_Loan                 100000 non-null  int32  
 8   Delay_from_due_date         100000 non-null  int64  
 9   Num_of_Delayed_Payment      100000 non-null  float64
 10  Changed_Credit_Limit        100000 non-null  float64
 11  Num_Credit_Inquiries        100000 non-null  float64
 12  Credit_Mix                  100000 non-null  float64
 13  Outstanding_Debt            100000 non-null  float64
 14  Credit_Utilization_Ratio    100000 non-null  float64
 15  Total_EMI_per_month         100000 non-null  float64
 16  Amount_invested_monthly     100000 non-null  float64
 17  Payment_Behaviour           100000 non-null  float64
 18  Monthly_Balance             100000 non-null  float64
 19  Credit_Score                100000 non-null  float64
 20  Occupation_Accountant       100000 non-null  uint8  
 21  Occupation_Architect        100000 non-null  uint8  
 22  Occupation_Developer        100000 non-null  uint8  
 23  Occupation_Doctor           100000 non-null  uint8  
 24  Occupation_Engineer         100000 non-null  uint8  
 25  Occupation_Entrepreneur     100000 non-null  uint8  
 26  Occupation_FreeLancer       100000 non-null  uint8  
 27  Occupation_Journalist       100000 non-null  uint8  
 28  Occupation_Lawyer           100000 non-null  uint8  
 29  Occupation_Manager          100000 non-null  uint8  
 30  Occupation_Mechanic         100000 non-null  uint8  
 31  Occupation_Media_Manager    100000 non-null  uint8  
 32  Occupation_Musician         100000 non-null  uint8  
 33  Occupation_Scientist        100000 non-null  uint8  
 34  Occupation_Teacher          100000 non-null  uint8  
 35  Occupation_Writer           100000 non-null  uint8  
 36  Payment_of_Min_Amount_0     100000 non-null  uint8  
 37  Payment_of_Min_Amount_NM    100000 non-null  uint8  
 38  Payment_of_Min_Amount_No    100000 non-null  uint8  
 39  Payment_of_Min_Amount_Yes   100000 non-null  uint8  
 40  Credit_History_Year_18.0    100000 non-null  uint8  
 41  Credit_History_Year_0       100000 non-null  uint8  
 42  Credit_History_Year_1       100000 non-null  uint8  
 43  Credit_History_Year_10      100000 non-null  uint8  
 44  Credit_History_Year_11      100000 non-null  uint8  
 45  Credit_History_Year_12      100000 non-null  uint8  
 46  Credit_History_Year_13      100000 non-null  uint8  
 47  Credit_History_Year_14      100000 non-null  uint8  
 48  Credit_History_Year_15      100000 non-null  uint8  
 49  Credit_History_Year_16      100000 non-null  uint8  
 50  Credit_History_Year_17      100000 non-null  uint8  
 51  Credit_History_Year_18      100000 non-null  uint8  
```

```
 52  Credit_History_Year_19     100000 non-null  uint8
 53  Credit_History_Year_2      100000 non-null  uint8
 54  Credit_History_Year_20     100000 non-null  uint8
 55  Credit_History_Year_21     100000 non-null  uint8
 56  Credit_History_Year_22     100000 non-null  uint8
 57  Credit_History_Year_23     100000 non-null  uint8
 58  Credit_History_Year_24     100000 non-null  uint8
 59  Credit_History_Year_25     100000 non-null  uint8
 60  Credit_History_Year_26     100000 non-null  uint8
 61  Credit_History_Year_27     100000 non-null  uint8
 62  Credit_History_Year_28     100000 non-null  uint8
 63  Credit_History_Year_29     100000 non-null  uint8
 64  Credit_History_Year_3      100000 non-null  uint8
 65  Credit_History_Year_30     100000 non-null  uint8
 66  Credit_History_Year_31     100000 non-null  uint8
 67  Credit_History_Year_32     100000 non-null  uint8
 68  Credit_History_Year_33     100000 non-null  uint8
 69  Credit_History_Year_4      100000 non-null  uint8
 70  Credit_History_Year_5      100000 non-null  uint8
 71  Credit_History_Year_6      100000 non-null  uint8
 72  Credit_History_Year_7      100000 non-null  uint8
 73  Credit_History_Year_8      100000 non-null  uint8
 74  Credit_History_Year_9      100000 non-null  uint8
 75  Credit_History_Month_5.0   100000 non-null  uint8
 76  Credit_History_Month_ 0    100000 non-null  uint8
 77  Credit_History_Month_ 1    100000 non-null  uint8
 78  Credit_History_Month_ 10   100000 non-null  uint8
 79  Credit_History_Month_ 11   100000 non-null  uint8
 80  Credit_History_Month_ 2    100000 non-null  uint8
 81  Credit_History_Month_ 3    100000 non-null  uint8
 82  Credit_History_Month_ 4    100000 non-null  uint8
 83  Credit_History_Month_ 5    100000 non-null  uint8
 84  Credit_History_Month_ 6    100000 non-null  uint8
 85  Credit_History_Month_ 7    100000 non-null  uint8
 86  Credit_History_Month_ 8    100000 non-null  uint8
 87  Credit_History_Month_ 9    100000 non-null  uint8
dtypes: float64(15), int32(1), int64(4), uint8(68)
memory usage: 21.4 MB
```

PARTIE 3

Machine Learning

Classificateur de forêt aléatoire (Random Forest Classifier)

Modèle de classification de Random Forest est entraîné sur les données, et les performances sont évaluées en termes d'exactitude, de précision et de F1 score.

Entrée [144]:
```
# les valeurs indépendantes(contient toutes les cols sauf 'Credit_Score')
X = df.drop('Credit_Score', axis=1).values
# la variable dépendante contient la colonne 'Credit_Score'
y = df['Credit_Score'].values
```

Entrée [145]:
```
from sklearn.model_selection import train_test_split
# diviser les données en ensembles d'entraînement de test, 80% des données so
# la graine aléatoire (random_state=42) assure la reproductibilité des résult
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
```

Entrée [146]:
```
from sklearn.ensemble import RandomForestClassifier
# créer et ajuster le modèle aux données d'entraînement
model = RandomForestClassifier()
model.fit(X_train, y_train)
# utiliser modèle entraîné pour faire des prédictions sur l'ens de test
y_pred = model.predict(X_test)
```

Entrée [147]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score,f1_
# fct qui prend les données de test, les étiquettes de test et le modèle entr
# renvoie des métriques d'évaluation tellles que l'exactitude, le rappel, la 
def evaluate_model(X_test,y_test, model):
    y_pred = model.predict(X_test)
    #accuracy
    acc = accuracy_score(y_test,y_pred)
    recall = recall_score(y_test,y_pred, average='macro')
    precision = precision_score(y_test,y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    cm = confusion_matrix(y_test, y_pred)

    # Collecter les métriques dans le data frame pour les ensembles de test d
    return pd.Series({'Accuracy':acc,'Recall':recall,'Precision': precision,'
```

Entrée [148]:
```
pd.DataFrame({'Random Forest Classifier (Test)': evaluate_model(X_test,y_test
              'Random Forest Classifier (Train)': evaluate_model(X_train, y_t
             })
```

Out[148]:

	Random Forest Classifier (Test)	Random Forest Classifier (Train)
Accuracy	0.780100	1.0
Recall	0.757201	1.0
Precision	0.765316	1.0
F1 Score	0.761107	1.0

Performance générale: L'exactitude(Accuracy) du modèle sur l'ensemble de test est d'environ 78%

Performance sur l'ensemble de test et d'entrainement: L'exactitude, le rappel, la précision et le score de F1 sur l'ensemble de test sont légèrement inférieurs à ceux sur l'ensemble d'entraînement. Cela indique que le modèle pourrait être légèrement surajusté(overfitting) aux données d'entrainement.

Recall: Le rappel est d'environ 75.7%, indiquant la capacité du modèle à identifier les vrais positifs parmi tous les cas réels positifs. Une valeur inférieure de rappel peut signifier que le modèle peut manquer certains cas positifs.

Precision: La précision est d'environ 76.5%, indiquant la capacité du modèle à ne pas classer à tort les négatifs comme possitifs. Une valeur inférieure de précision pourrait signifier qu'il y a un nombre notable de faux positifs.

Score F1: Le score F1, qui prend en compte à la fois la précision et le rappel, est d'environ 76%. Il fournit une mesure équilibrée entre la précision et le rappel

K-Nearest Neighbors (KNN)

Entrée [149]:
```
X = df.drop('Credit_Score',axis=1).values
y = df['Credit_Score'].values
```

Entrée [150]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random
```

Entrée [151]:
```
from sklearn.preprocessing import StandardScaler
# Standardiser les données pour mettre à l'échelle les caractéristiques.
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Entrée [152]:
```
from sklearn.neighbors import KNeighborsClassifier
# Utiliser 5 voisins dans le modèle
model2 = KNeighborsClassifier(n_neighbors=5)
model2.fit(X_train, y_train)
y_pred = model2.predict(X_test)
```

```
  File "C:\Users\33766\AppData\Roaming\Python\Python39\site-packages\joblib\e
xternals\loky\backend\context.py", line 282, in _count_physical_cores
    raise ValueError(f"found {cpu_count_physical} physical cores < 1")
```

Entrée [153]:
```
pd.DataFrame({'KNN (Test)': evaluate_model(X_test, y_test, model2),
              'KNN (Train)': evaluate_model(X_train, y_train, model2)
             })
```

Out[153]:

	KNN (Test)	KNN (Train)
Accuracy	0.568000	0.711838
Recall	0.489791	0.646304
Precision	0.526497	0.708883
F1 Score	0.497576	0.664984

Performance générale L'exactitude sur l'ensemble de test est d'environ 56.8%, ce qui indique que le modèle prédit correctement la classe de crédit dans environ 56.8% des cas.

Performance sur l'ensemble de test vs d'entraînement :L'exactitude, le rappel, la précision et le score F1 sur l'ensemble de test sont tous inférieurs à ceux sur l'ensemble d'entraînement. Cela suggère un possible surajustement (overfitting) du modèle aux données d'entraînement.

Rappel : Le rappel est d'environ 48.98%, indiquant la capacité du modèle à identifier les vrais positifs parmi tous les cas réels positifs. Une valeur plus basse de rappel suggère que le modèle peut manquer certains cas positifs.

Précision : La précision est d'environ 52.6%, indiquant la capacité du modèle à ne pas classer à tort les négatifs comme positifs. Une valeur plus basse de précision peut signifier qu'il y a un nombre notable de faux positifs.

Score F1 : Le score F1, qui prend en compte à la fois la précision et le rappel, est d'environ 49.76%. Comme le score F1 est la moyenne harmonique entre la précision et le rappel, une valeur plus basse suggère un équilibre suboptimal entre ces deux métriques.