Credit score classification

 Kaggle의 데이터셋을 활용하여 신용점수 분류 문제를 풀어보는 프로젝트

Ⅰ. 프로젝트 개요

1. 프로젝트 목표

목적

- Kaggle의 'Credit score classification' 데이터셋을 활용하여 기초적인 데이터 전처리 과정과 분류 모델을 생성하고 평가하는 방법을 학습하려 한다.

2. 가설 설정

데이터 칼럼

```
['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
 'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
 'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
 'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
 'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
```

```
 'Credit_Utilization_Ratio', 'Credit_History_Age',
 'Payment_of_Min_Amount', 'Total_EMI_per_month',
 'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance',
 'Credit_Score'],
```

- 해당 데이터는 ID, Age, Annual_Income, Num_of_Loan, Payment_Behaviour 등 고객의 개인정보와 신용 관련 정보들이 feature로 구성되어 있다. label은 Credit_Score로 고객의 신용 점수이다.
  즉, 해당 프로젝트는 고객의 신용 관련 정보를 바탕으로 고객의 신용 점수 등급을 분류하는 것이 목표이다.
  독립변수 중 Annual_Income(연소득), Monthly_Inhand_Salary(월급여), Num_of_Loan(대출 개수)가 신용등급에 큰 영향을 줄 것으로 예상된다.

3. 사용한 데이터 및 출처

원본 데이터 및 출처

- Kaggle의 'Credit score classification' 데이터셋을 사용한다.
- 라이선스: CC0: Public Domain

II. 데이터 수집 및 전처리

1. 데이터 출처 및 수집 방법

1. 데이터 수집, 불러오기

i) 데이터 수집

- 'Credit score classification' 페이지에서 csv파일을 다운받아 사용하였다.

ii) 데이터 불러오기

- train.csv 파일의 데이터만을 사용하며, 'Credit_Score' 컬럼을 종속변수로 한다.

```
import pandas as pd
X = pd.read_csv("/content/drive/MyDrive/Adam/신용등급_분류/train.csv")
```

2. 데이터 파악, 전처리

i) 데이터 컬럼과 자료형 확인

- pandas의 info를 통해서 출력된 내용 첨부한다.

```
X.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
```

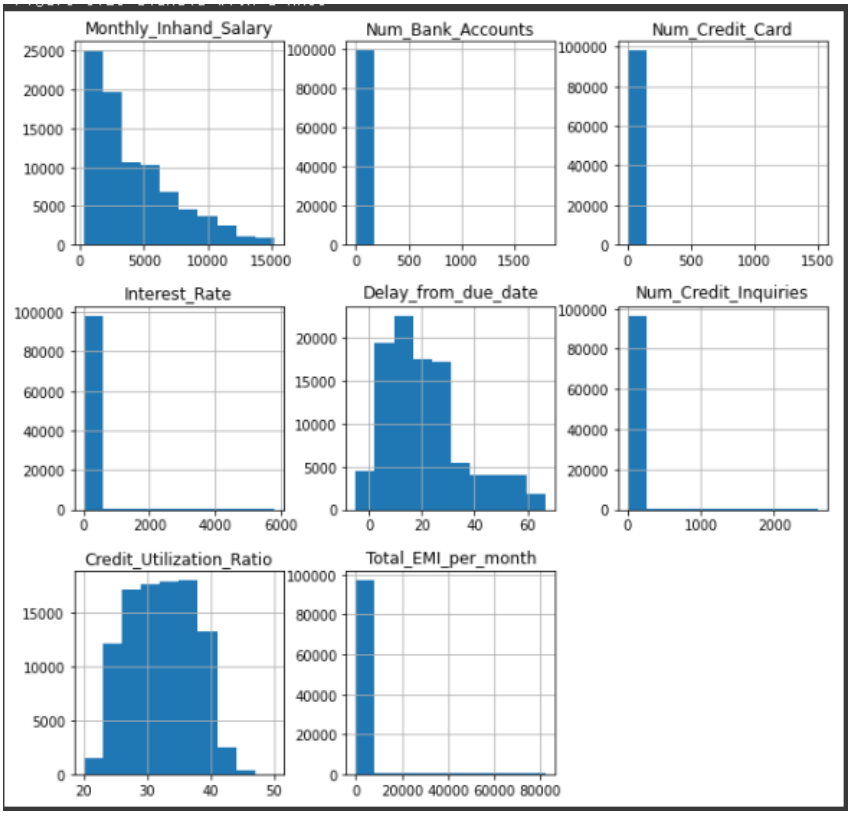
- 전체 데이터의 수(row), 컬럼의 수를 파악한다.

```
print('전체 데이터 수 : ', len(X) )
print('컬럼 수 : ', len(X.columns))
```

```
전체 데이터 수 :  100000
컬럼 수 :  28
```

ii) 전처리 전 시각화를 통한 데이터 파악

- 숫자형(int, float) 컬럼의 히스토그램을 출력한다.



- 범주형 컬럼의 value_counts()를 출력한다.

```
1 X['Month'].value_counts()
```

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

```
1 X['SSN'].value_counts()
```

```
#F%$D@*&8      5572
078-73-5990       8
486-78-3816       8
750-67-7525       8
903-50-0305       8
                ...
856-06-6147       4
753-72-2651       4
331-28-1921       4
604-62-6133       4
286-44-9634       4
Name: SSN, Length: 12501, dtype: int64
```

```
1 X['Occupation'].value_counts()
```

```
_______       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
```

```
1 X['Annual_Income'].value_counts()
```

```
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
```

```
1 X['Num_of_Loan'].value_counts()

3         14386
2         14250
4         14016
0         10383
1         10083
          ...
1320_         1
103           1
1444          1
392           1
966           1
Name: Num_of_Loan, Length: 434, dtype: int64
```

```
1 X['Credit_Mix'].value_counts()

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

```
1 X['Credit_History_Age'].value_counts()

15 Years and 11 Months    446
19 Years and 4 Months     445
19 Years and 5 Months     444
17 Years and 11 Months    443
19 Years and 3 Months     441
                         ...
0 Years and 3 Months       20
0 Years and 2 Months       15
33 Years and 7 Months      14
33 Years and 8 Months      12
0 Years and 1 Months        2
Name: Credit_History_Age, Length: 404, dtype: int64
```

```
1 X['Monthly_Balance'].value_counts()

__-333333333333333333333333333__    9
312.49408867943663                  1
415.32532309844316                  1
252.08489793906085                  1
254.9709216273975                   1
                                   ..
366.2890379762706                   1
151.1882696261166                   1
306.75027851710234                  1
278.8720257394474                   1
393.6736955618808                   1
Name: Monthly_Balance, Length: 98792, dtype: int64
```

```
1 X['Type_of_Loan'].value_counts()

Not Specified
1408
Credit-Builder Loan
1280
Personal Loan
1272
Debt Consolidation Loan
1264
Student Loan
1240

...
Not Specified, Mortgage Loan, Auto Loan, and Payday Loan
8
Payday Loan, Mortgage Loan, Debt Consolidation Loan, and Student Loan
8
Debt Consolidation Loan, Auto Loan, Personal Loan, Debt Consolidation Loan, Student Loan, and Credit-Builder Loan
8
Student Loan, Auto Loan, Student Loan, Credit-Builder Loan, Home Equity Loan, Debt Consolidation Loan, and Debt
Consolidation Loan      8
Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan
8
Name: Type_of_Loan, Length: 6260, dtype: int64
```

```
1 X['Changed_Credit_Limit'].value_counts()

_                     2091
8.22                   133
11.5                   127
11.32                  126
7.35                   121
                      ...
-1.84                    1
0.8899999999999999       1
28.06                    1
1.5599999999999996       1
21.17                    1
Name: Changed_Credit_Limit, Length: 4384, dtype: int64
```

```
1 X['Outstanding_Debt'].value_counts()

1360.45     24
460.46      23
1151.7      23
1109.03     23
467.7       16
            ..
245.46_      1
645.77_      1
174.79_      1
1181.13_     1
1013.53_     1
Name: Outstanding_Debt, Length: 13178, dtype: int64
```

```
1 X['Payment_Behaviour'].value_counts()

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

- ‘Customer_ID‘가 있으므로 식별자인 ‘Name’칼럼과 ‘SSN’칼럼은 의미가 중복되므로 칼럼을 제거한다.

```
del X['Name']
del X['SSN']
```

iii) 결측치 확인 및 처리

- 각 컬럼의 결측치 비율을 `.isna()`를 통해 확인한다.

```
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
```

- 숫자형 데이터의 결측값 처리

숫자형 데이터인 Monthly_Inhand_Salary 칼럼에 결측치가 15,002개, Num_Credit_Inquires에는 1,965개가 있는 것을 확인하였다. describe를 통해서 데이터의 분포를 확인해보니 최대값이 매우 큰걸로 보아 이상값이 존재하는 것으로 확인하였다. 따라서 평균 대신 중앙값으로 결측값을 대체해주었다.

	Monthly_Inhand_Salary
count	84998.000000
mean	4194.170850
std	3183.686167
min	303.645417
25%	1625.568229
50%	3093.745000
75%	5957.448333
max	15204.633333

Num_Credit_Inquiries
98035.000000
27.754251
193.177339
0.000000
3.000000
6.000000
9.000000
2597.000000

```
X['Monthly_Inhand_Salary'] = X['Monthly_Inhand_Salary'].fillna(X['Monthly_Inhand_Salary'].median())
X['Num_Credit_Inquiries'] = X['Num_Credit_Inquiries'].fillna(X['Num_Credit_Inquiries'].median())
```

	Monthly_Inhand_Salary
count	100000.000000
mean	4029.084964
std	2961.363540
min	303.645417
25%	1792.084167
50%	3093.745000
75%	5371.525000
max	15204.633333

Num_Credit_Inquiries
100000.000000
27.326780
191.293766
0.000000
3.000000
6.000000
9.000000
2597.000000

- 범주형 데이터의 결측값 처리

`unknown` 이라는 새로운 범주로 결측치를 대체하였다.

```
# 결측치를 'unknown'으로 대체
X['Name'] = X['Name'].fillna('unknown')
X['Type_of_Loan'] = X['Type_of_Loan'].fillna('unknown')
X['Credit_History_Age'] = X['Credit_History_Age'].fillna('unknown')
```

- 숫자값인데 범주형인 데이터의 결측값을 처리해주었다. 특수문자를 제거한 후 중위값으로 대체 후 숫자형으로 변환해주었다.

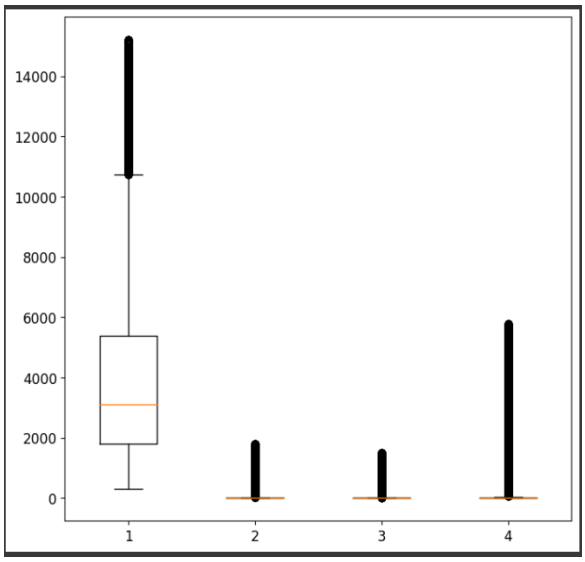
```
#'_'특수문자 제거(모든 특수 문자 제거 : pat = r'[^\w]')
X['Num_of_Delayed_Payment'] = X['Num_of_Delayed_Payment'].str.replace(pat=r'_',repl=r'',regex=True)
#결측값을 중위값으로 대체
X['Num_of_Delayed_Payment'] = X['Num_of_Delayed_Payment'].fillna(X['Num_of_Delayed_Payment'].median())
#문자형 값들을 정수형으로 변환
X['Num_of_Delayed_Payment'] = X['Num_of_Delayed_Payment'].astype(int)
X['Num_of_Delayed_Payment']

X['Amount_invested_monthly'] = X['Amount_invested_monthly'].str.replace(pat=r'_',repl=r'',regex=True)
X['Amount_invested_monthly'] = X['Amount_invested_monthly'].astype(float)
X['Amount_invested_monthly'] = X['Amount_invested_monthly'].fillna(X['Amount_invested_monthly'].median())

X['Monthly_Balance'] = X['Monthly_Balance'].str.replace(pat=r'_',repl=r'',regex=True)
X['Monthly_Balance'] = X['Monthly_Balance'].astype(float)
X['Monthly_Balance'] = X['Monthly_Balance'].fillna(X['Monthly_Balance'].median())
```

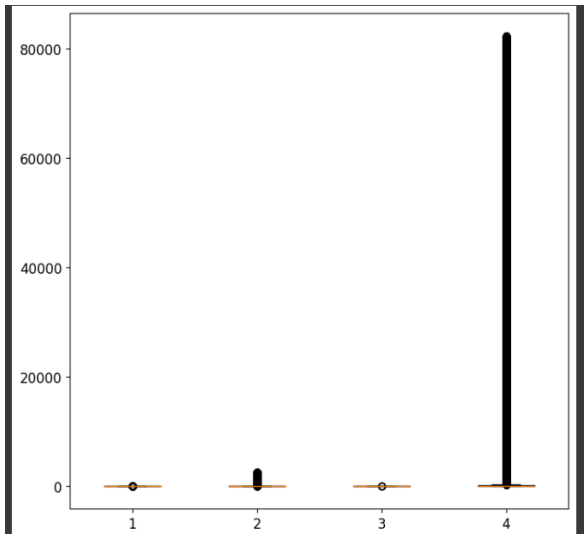
```
X.isnull().sum()
```

```
ID                          0
Customer_ID                 0
Month                       0
Name                        0
Age                         0
SSN                         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
Type_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
```

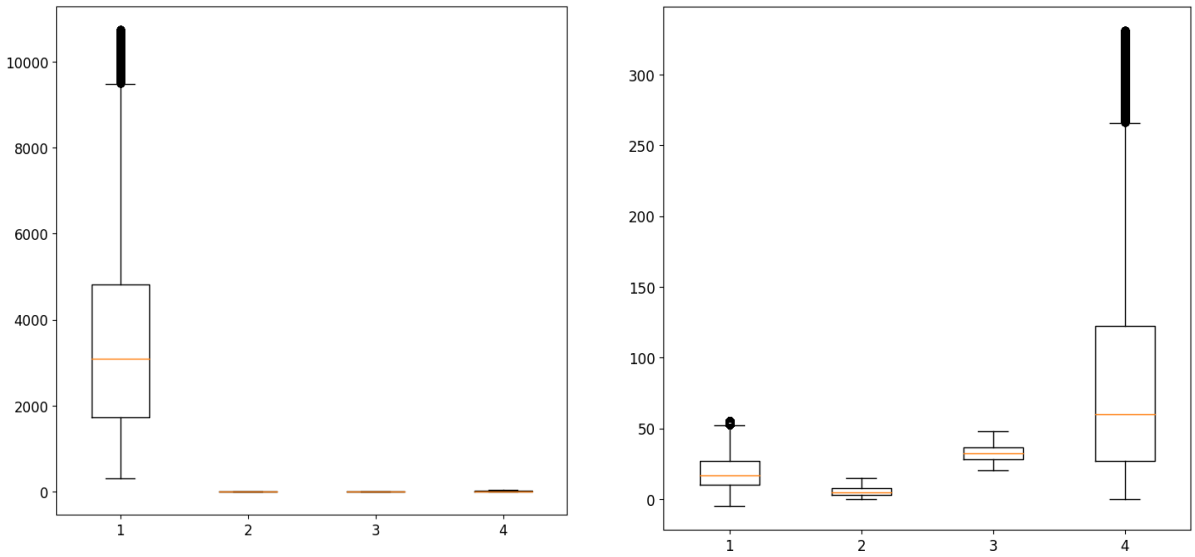
iv) 이상치 확인 및 처리

- 숫자형 데이터 : 결측값 처리 전





- 숫자형 데이터 : 결측값 처리 전



v) 범주형 변수의 처리

- Annual_Income, Num_of_Loan, Changed_Credit_Limit, Outstanding_Debt, 'Monthly_Balance → 숫자인데 특수기호 때문에 범주형으로 인식됨, 특수값 제외한 후 float형으로 변환한다.

```
X['ID'] = X.ID.apply(lambda x: int(x, 16)) #16진수 -> 10진수 정수형으로 변환
X['Customer_ID'] = X.Customer_ID.apply(lambda x : int(x[4:],16)) #4번째 문자열부터 16진수 문자열을 정수형으로

X['Month'] = pd.to_datetime(X.Month, format = '%B').dt.month

#Age칼럼의 '_'특수문자를 제거
X['Age'] = X['Age'].str.replace(pat=r'_',repl=r'',regex=True)
#특수문자 제거후 정수형으로 변환
X['Age'] = X['Age'].astype(int)

#'_'특수문자 제거
X['Annual_Income'] = X['Annual_Income'].str.replace(pat=r'_',repl=r'',regex=True)
X['Annual_Income'] = X['Annual_Income'].astype(float)

X['Num_of_Loan'] = X['Num_of_Loan'].str.replace(pat=r'_',repl=r'',regex=True)
X['Num_of_Loan'] = X['Num_of_Loan'].astype(int)

X['Outstanding_Debt'] = X['Outstanding_Debt'].str.replace(pat=r'_',repl=r'0',regex=True)
X['Outstanding_Debt'] =X['Outstanding_Debt'].astype(float)

X['Changed_Credit_Limit'] = X['Changed_Credit_Limit'].str.replace(pat=r'_',repl=r'0',regex=True)
X['Changed_Credit_Limit'] = X['Changed_Credit_Limit'].astype(float)
```

- Label Encoding : n개의 범주형 데이터를 0부터 n-1까지의 연속적 수치 데이터로 표현한다. 몇몇 ML알고리즘에는 이를 적용할 경우 숫자로 되어 있어 가중치로 인식하게 되면서 값의 왜곡이 생기기도 한다. 따라서 선형회귀 알고리즘에서는 적용하지 않는다. 하지만 분류 계열 ML알고리즘에서는 이러한 특성을 반영하지 않으므로 레이블 인코딩도 별 문제가 없다.

```
from sklearn.preprocessing import LabelEncoder as le
Occupation_le = le()
Type_of_Loan_le = le()
Credit_Mix_le = le()
Credit_History_Age_le = le()
Payment_of_Min_Amount_le = le()
Payment_Behaviour_le = le()
Credit_Score_le = le()
```

```
X['Occupation'] = Occupation_le.fit_transform(X['Occupation'])
X['Type_of_Loan'] = Type_of_Loan_le.fit_transform(X['Type_of_Loan'])
X['Credit_Mix'] = Credit_Mix_le.fit_transform(X['Credit_Mix'])
X['Credit_History_Age'] = Credit_History_Age_le.fit_transform(X['Credit_History_Age'])
X['Payment_of_Min_Amount'] = Payment_of_Min_Amount_le.fit_transform(X['Payment_of_Min_Amount'])
```

```
X['Payment_Behaviour'] = Payment_Behaviour_le.fit_transform(X['Payment_Behaviour'])
X['Credit_Score'] = Credit_Score_le.fit_transform(X['Credit_Score'])
```

3. 통계적 요약과 변수 간 관계

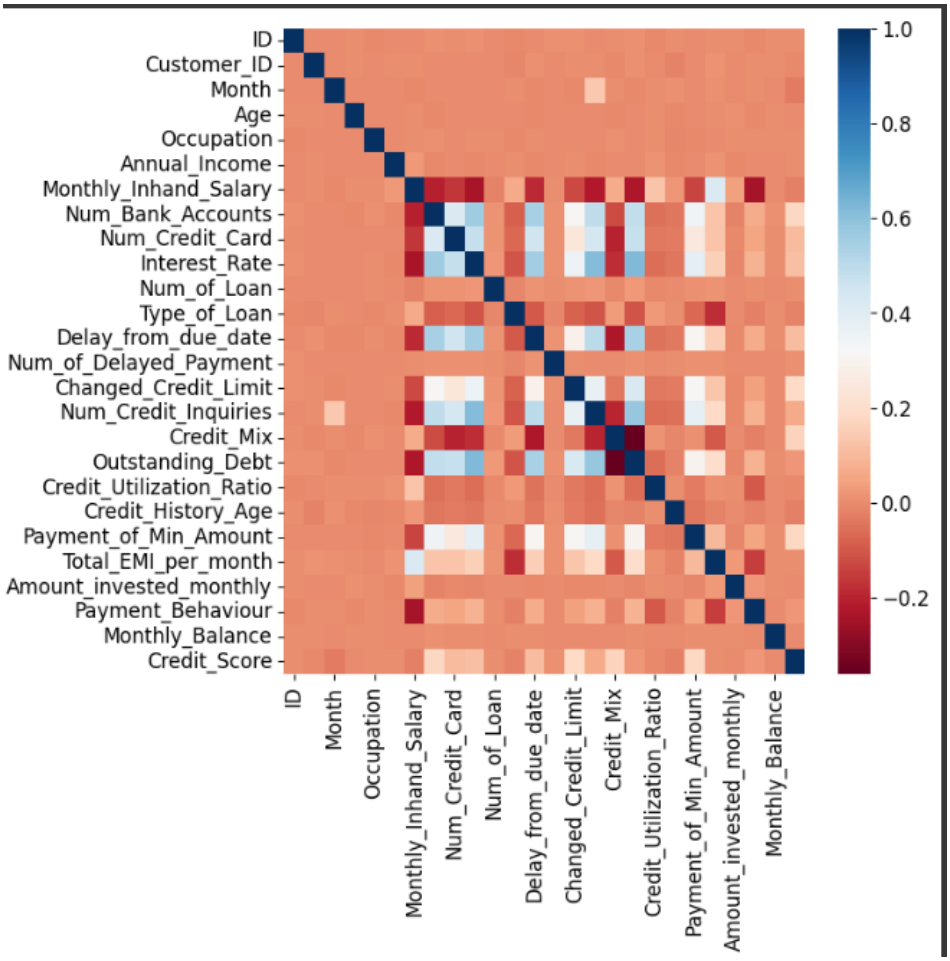
i) 데이터의 통계적 성질

	ID	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_Pay
count	78577.000000	78577.000000	78577.000000	78577.000000	7.857700e+04	78577.000000	78577.000000	78577.000000	78577.000000	78577.000000	78577.000000	78577.00
mean	80751.101747	25908.247668	4.465442	113.405742	1.697450e+05	3595.219722	5.323072	5.478690	14.243799	2.932830	19.677768	29.4
std	43212.538236	14336.710576	2.292677	698.983459	1.426289e+06	2377.335928	2.553855	2.031832	8.561554	63.356592	12.868045	216.3
min	5634.000000	1006.000000	1.000000	-500.000000	7.005930e+03	303.645417	-1.000000	0.000000	1.000000	-100.000000	-5.000000	-3.00
25%	43339.000000	13615.000000	2.000000	24.000000	1.904044e+04	1721.569167	3.000000	4.000000	7.000000	1.000000	10.000000	9.00
50%	80794.000000	25640.000000	4.000000	33.000000	3.502349e+04	3093.745000	5.000000	5.000000	13.000000	3.000000	17.000000	14.00
75%	118093.000000	38307.000000	6.000000	42.000000	6.607492e+04	4827.682500	7.000000	7.000000	20.000000	5.000000	27.000000	18.00
max	155629.000000	50999.000000	8.000000	8698.000000	2.419806e+07	10739.366667	11.000000	11.000000	34.000000	1496.000000	55.000000	4397.00

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
78577.000000	78577.000000	78577.000000	78577.000000	78577.000000	78577.000000	78577.000000	78577.000000	7.857700e+04
19.677768	29.410540	10.135135	5.610599	1373.889859	32.184861	83.966892	595.758882	-2.969486e+22
12.868045	216.319185	6.754597	3.688576	1117.349135	5.064060	76.894593	1996.092166	3.146034e+24
-5.000000	-3.000000	-6.490000	0.000000	0.230000	20.172942	0.000000	0.000000	-3.333333e+26
10.000000	9.000000	5.070000	3.000000	552.500000	27.971484	26.854020	73.883676	2.723158e+02
17.000000	14.000000	9.290000	5.000000	1132.540000	32.189707	59.957811	135.030453	3.364493e+02
27.000000	18.000000	14.530000	8.000000	1817.000000	36.410399	122.634520	231.749480	4.487940e+02
55.000000	4397.000000	36.970000	15.000000	4998.070000	48.247003	331.073500	10000.000000	1.525368e+03

ii) 변수 간 관계

- heatmap : 변수 간 상관관계 시각화



4. 파생변수 생성과 데이터 정제

i) 파생변수 생성

- (해당 과정은 선택적으로 진행함)

ii) 최종 데이터 선정 및 시각화

- 최종 데이터프레임

	ID	Customer_ID	Month	Age	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card
0	5634	3392	1	23	12	19114.12	1824.843333	3	4
1	5635	3392	2	23	12	19114.12	3093.745000	3	4
2	5636	3392	3	-500	12	19114.12	3093.745000	3	4
3	5637	3392	4	23	12	19114.12	3093.745000	3	4
4	5638	3392	5	23	12	19114.12	1824.843333	3	4
...	...	...	...	...	...	...	...	...	...
99994	155624	37932	3	25	9	39628.99	3359.415833	4	6
99995	155625	37932	4	25	9	39628.99	3359.415833	4	6
99996	155626	37932	5	25	9	39628.99	3359.415833	4	6
99998	155628	37932	7	25	9	39628.99	3359.415833	4	6
99999	155629	37932	8	25	9	39628.99	3359.415833	4	6

78577 rows × 26 columns

- 최종 칼럼

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

iii) 데이터 저장

```
X.to_csv('preprocessing_data.csv',index=False)
```

III. 머신러닝 모델링

1. 모델 선정

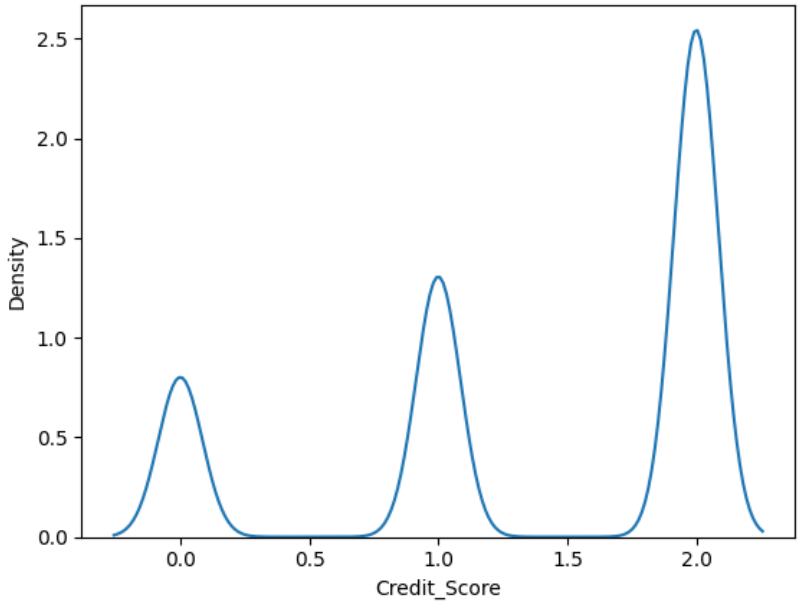
- target값이 3개로 구성된 다중분류 문제이므로 둘 이상의 클래스를 구별하는 다중 분류기를 사용해야 한다.
- OVO전략 : 0과 1구별, 0과 2 구별, 1과 2구별 등 각 숫자의 조합마다 이진 분류기를 훈련시킨다. 각 분류기의 훈련에 전체 훈련 세트 중 구별할 두 클래스에 해당하는 샘플만 필요하다.
- RandomForestClassifier : 랜덤포레스트는 여러 개의 결정 트리 분류기가 전체 데이터에서 배깅 방식으로 각자의 데이터를 샘플링해 개별적으로 학습을 수행한 뒤 최종적으로 만든 분류기가 보팅을 통해 예측 결정한다.
- XGBOOST : 약한 분류기에 가중치를 부여하며 훈련시키는 GBM과 유사한 하이퍼 파라미터를 가지고 있으며 여기에 조기 중단, 과적합을 규제하기 위한 하이퍼 파라미터가 추가된 알고리즘이다.

2. 훈련 및 검증 데이터 분할

```
from sklearn.model_selection import train_test_split
#데이터의 피처값
X = data.iloc[:,:-1]
#데이터의 타겟값
y = data.iloc[:,-1]
#train은 70%, test는 30%비율로 분할함
#random_state=42로 진행함
X_train , X_test , y_train , y_test = train_test_split(X,y , test_size= 0.3, random_state=42)
print(f'X_train shape : {X_train.shape}')
print(f'y_train shape : {y_train.shape}')
print(f'X_test shape : {X_test.shape}')
print(f'y_test shape : {y_test.shape}')
```

```
X_train shape : (55003, 25)
y_train shape : (55003,)
X_test shape : (23574, 25)
y_test shape : (23574,)
```

- 분할 후 y_train 분포



⇒다중분류 문제이다.

3. 모델 학습

Boosting계열인 XGBOOST모델을 사용하였다.

학습을 수행할 결정 트리 모델의 개수인 n_estimators는 300, random_state는156, 다중 분류일 때 적용하는 손실함수인 objective는 'multi:softmax'로 파라미터를 지정해주었다.

```
from xgboost import XGBClassifier
xgb_clf = XGBClassifier(n_estimators=300, random_state=156,objective = 'multi:softmax')
xgb_clf.fit(X_train, y_train, early_stopping_rounds=100, eval_metric='auc', eval_set=[(X_train, y_train), (X_test, y_test)])
xgb_pred = xgb_clf.predict(X_test)
```

4. 모델 학습 결과 및 평가 지표

XGBOOST 평가 지표

```
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score, f1_score,precision_score, recall_score
print(classification_report(y_test, xgb_pred))
print("평균 f1 score : {:.3f}".format(f1_score(y_test,xgb_pred,average = "micro")))
print("평균 precision score : {:.3f}".format(precision_score(y_test,xgb_pred,average = "micro")))
print("평균 recall score : {:.3f}".format(recall_score(y_test,xgb_pred,average = "micro")))
print('accuracy score : {:.3f}'.format(accuracy_score(y_test,xgb_pred)))
```

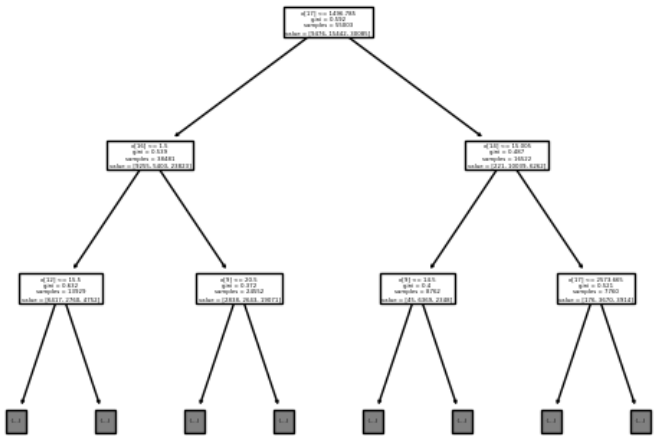
```
              precision    recall  f1-score   support

           0       0.77      0.75      0.76      4036
           1       0.79      0.78      0.78      6558
           2       0.82      0.84      0.83     12980

    accuracy                           0.80     23574
   macro avg       0.79      0.79      0.79     23574
weighted avg       0.80      0.80      0.80     23574

평균 f1 score : 0.804
평균 precision score : 0.804
평균 recall score : 0.804
accuracy score : 0.804
```

- 각각의 클래스에 대해 자신을 양성 클래스로, 다른 클래스를 음성 클래스로 가정하여 OvR문제를 풀고 각각에 대해 정밀도, 재현율, 위양성률 등의 평가 점수를 구했다.
- 클래스별로 각각 다른 평가점수가 나오므로 micro기준으로 평균을 구해 성능을 구해보았다.
- accuracy : 정확도로 전체 학습데이터의 개수에서 각 클래스에서 자신의 클래스를 정확하게 맞춘 개수의 비율을 의미한다.
- F1 score : 정밀도와 재현율의 가중조화평균을 의미한다.
- Recall Score : 양성 클래스에 속한 표본 중에 양성 클래스에 속한다고 출력한 표본 수의 비율을 뜻한다.
- Precision Score : 양성이라고 예측한 샘플 중 실제로 양성 클래스에 속하는 샘플 수의 비율을 의미한다.

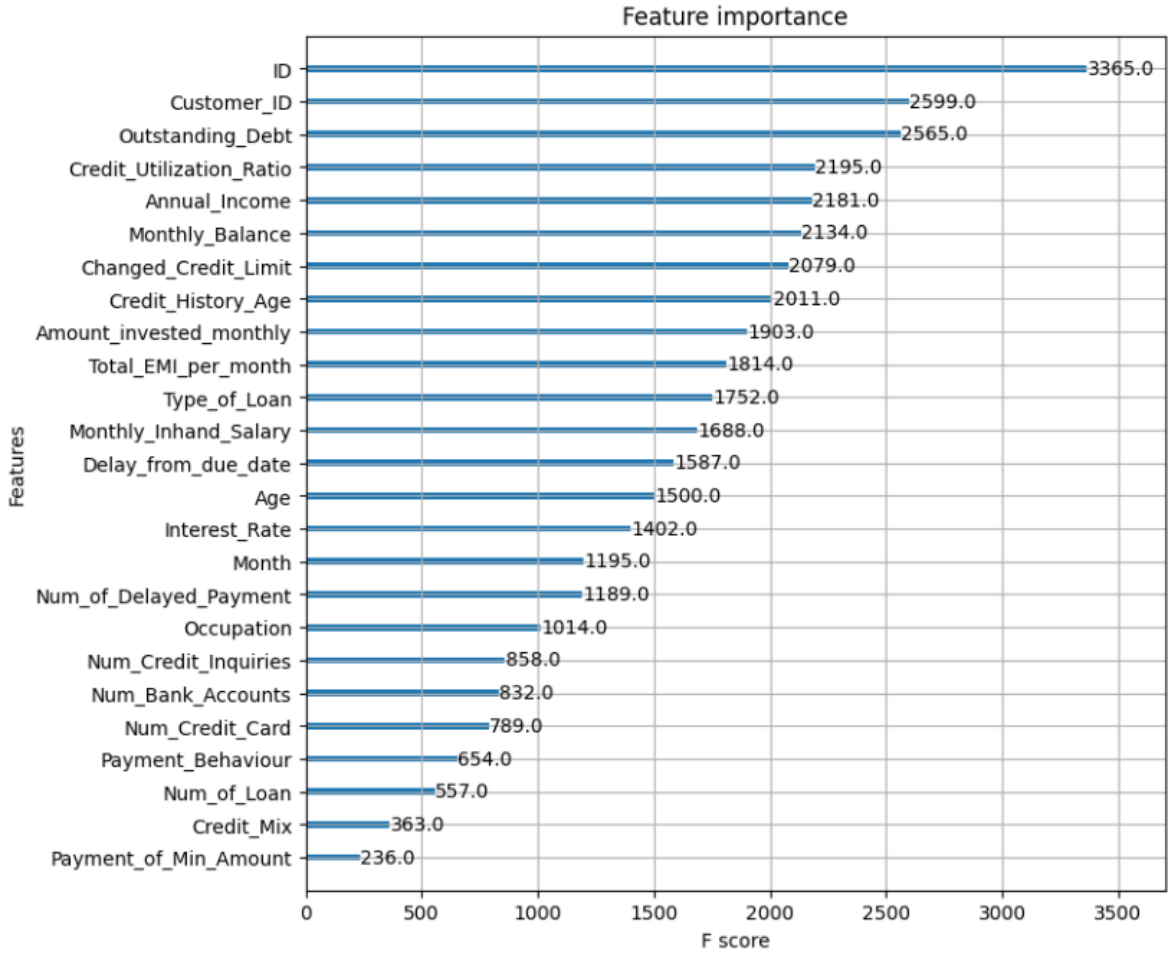
모델별 성능

- RandomForestClassifier : accuracy_score = 0.8014
- OneVsOneClassifier : accuracy_score = 0.282
- XGBOOSTClassifier : accuracy_score = 0.804

plot_tree함수로 결정기준 확인



feature importance



IV. 결과 해석 및 중요한 특성 분석

1. 분석 결과의 의미 및 가설 검증

ID, Customer_ID, Oustanding_Debt, Credit_Utilization_Ratio, Annual_income의 feature importance가 높게 나왔으므로 유의미한 변수라고 볼 수 있다.

신용등급을 평가함에 있어서 연소득(Anuual_income)과 이자율(Interest Rate)이 중요한 영향을 줄 것이라고 가정하였다.

분석 결과 연소득은 실제로 신용등급에 큰 영향을 준다는 것을 확인하였다.

하지만 이자율의 경우 상대적으로 적은 영향을 주었다.

이자율의 경우 중위적으로 사용되므로 이러한 결과가 나왔다고 생각한다. 신용등급이 낮은 사람이 대출을 받을 때 높은 이자율을 적용받지만 은행에 예금을 하게 될 경우 낮은 이자율을 적용받게 된다. 이렇게 같은 고객이더라도 이자율이 다를 수 있기 때문에 이자율의 피처중요도가 낮은 것으로 판단된다.

2. 결과에 대한 해석 및 활용 방안

고객의 금융 정보를 가지고 있는 카드사에서 새로운 고객을 유치할 때 가지고 있는 고객의 금융 정보를 기반으로 신용등급을 판별해서 맞춤형 카드를 추천하는데 활용할 수 있다.

Ⅴ. 코드 및 서비스화

1. 전체 과정의 코드

📎 신용등급분류_전처리_코드

📎 신용등급분류_모델링_코드

Ⅵ. 프로젝트 요약

1. 프로젝트 과정 요약

- kaggle에서 Credit Score Classification 데이터를 다운로드 하였다.
- 이상값 처리, 결측치 처리, 범주형 변수 처리 등 데이터 전처리를 진행하였다.
- 데이터의 특성을 알아본 후 XGBoost Classifier을 이용해서 모델링을 진행하였다.
- 모델의 성능 평가 지표를 도출 후 해석하였다.

2. 프로젝트를 통해 얻은 인사이트, 피드백

i) 새롭게 학습한 내용

- 다중 클래스 분류
- OVO, OVA 다중 분류 모델 알고리즘

ii) 아쉬운 점

모델의 성능이 낮게 나온점이 아쉬웠다.

iii) 다음 분석에서 보완할 것

타켓값을 잘 설명하는 유의미한 파생변수를 생성해볼 것이다.

하이퍼파라미터 튜닝을 진행해서 모델의 성능을 좀 더 높여볼 것이다.

3. 사용한 도구 및 라이브러리, 참고자료 모음
머신러닝 완벽가이드

https://datascienceschool.net/03 machine learning/09.04 분류 성능평가.html

https://wotres.tistory.com/entry/분류-성능-측정하는법-Accuracy-Precision-Recall-F1-score-ROC-AUC-in-python

https://engineer-mole.tistory.com/329
