채무불이행 분류 예측 모델

B조
학번_이름 :
2019147036_김륜구
2019147014_이혁주
2017147016_이준엽
2020147051_장건

Contents

데이터 탐색

Data types 확인

```
[ ]  df.dtypes
```

```
gender            object
car               object
reality           object
child_num          int64
income_total     float64
income_type       object
edu_type          object
family_type       object
house_type        object
DAYS_BIRTH         int64
DAYS_EMPLOYED      int64
FLAG_MOBIL         int64
work_phone         int64
phone              int64
email              int64
occyp_type        object
family_size        int64
begin_month        int64
credit             int64
dtype: object
```

각 column들의 data type을 확인하였다.

숫자로 이루어진 column(int64, float64)과 문자로 구성된 column(object)들이 섞여 있다.

데이터 탐색

```
pandas_profiling.ProfileReport(df)
```

Overview

Overview | Reproduction | Warnings 3

Dataset statistics

Number of variables	19
Number of observations	13228
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	488
Duplicate rows (%)	3.7%
Total size in memory	1.9 MiB
Average record size in memory	152.0 B

Variable types

BOOL	7
CAT	6
NUM	6

Pandas_profiling 라이브러리를 통해 각 column들의 기본적 분석을 진행하였다.

Column개수, 전체 데이터 수, 결측치가 있는 데이터 등등을 확인할 수 있었다.

Gender과 같이 데이터가 문자인 경우 데이터의 종류, 결측치 개수, 데이터의 분포 등을 알 수 있다.

Child_num과 같이 데이터가 숫자인 경우 추가로 데이터들의 평균, 최소/최댓값, 0값의 개수 및 비율을 추가로 알 수 있다.

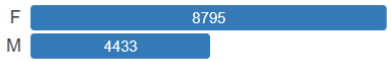
데이터 탐색

Gender 분석

Variables

gender
Categorical

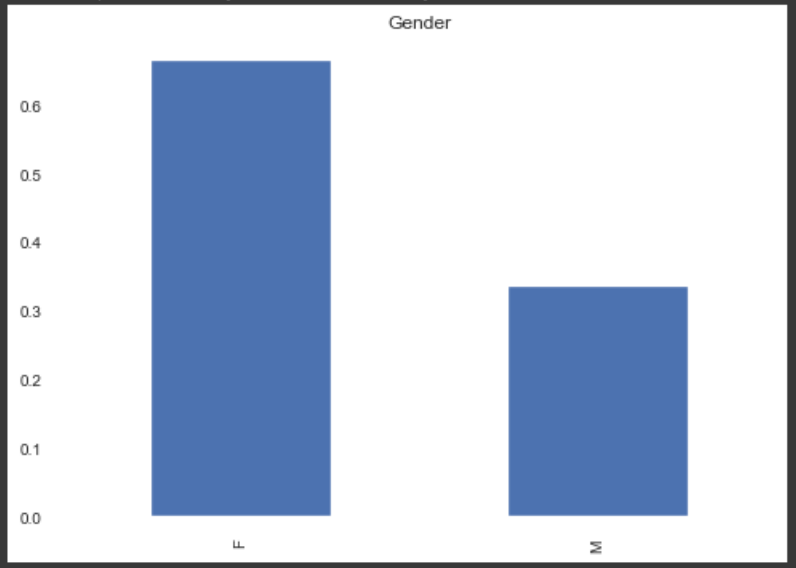
Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB



여성의 비율이 66.49%로 남성보다 높은 비율임을 확인하였다.

```
[ ]  df['gender'].value_counts(normalize=True) * 100

F    66.487753
M    33.512247
Name: gender, dtype: float64
```

```
[ ]  df['gender'].value_counts(normalize=True).plot.bar(title= 'Gender')

<AxesSubplot:title={'center':'Gender'}>
```



데이터 탐색

Child_num 분석

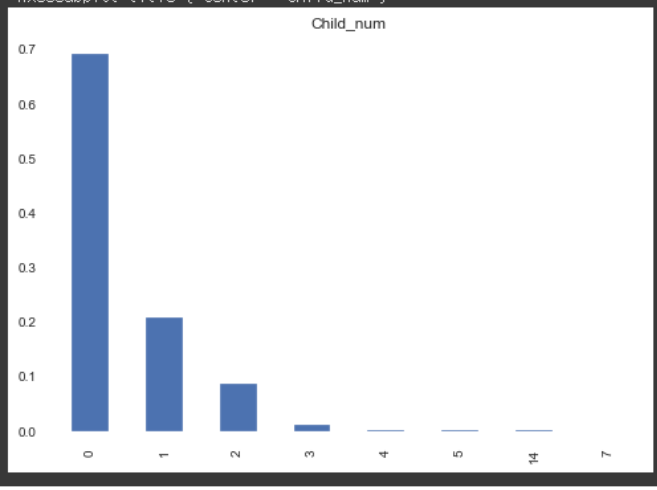
child_num
Real number (ℝ≥0)

ZEROS

Distinct count	8
Unique (%)	0.1%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	[제목 없음] 0.4251874566071967
Minimum	0
Maximum	14
Zeros	9144
Zeros (%)	69.1%
Memory size	103.3 KiB

```
[ ]  df['child_num'].value_counts(normalize=True) * 100

0     69.126096
1     20.796795
2      8.693680
3      1.118839
4      0.204112
5      0.037799
14     0.015119
7      0.007560
Name: child_num, dtype: float64
```

```
[ ]  df['child_num'].value_counts(normalize=True).plot.bar(title= 'Child_num')

<AxesSubplot:title={'center':'Child_num'}>
```



자녀의 수가 0명인 경우가 70%로 대부분이고 1명인 경우 20%, 2명인 경우 8%로 자녀가 있다면 1명 혹은 2명인 것을 확인할 수 있다. Maximum 값으로 14명이 존재했는데 child_num의 숫자가 너무 큰 경우 이상치를 의심해야 한다.

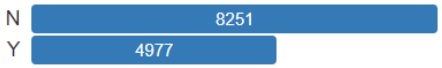
데이터 탐색

Car 분석

car
Boolean

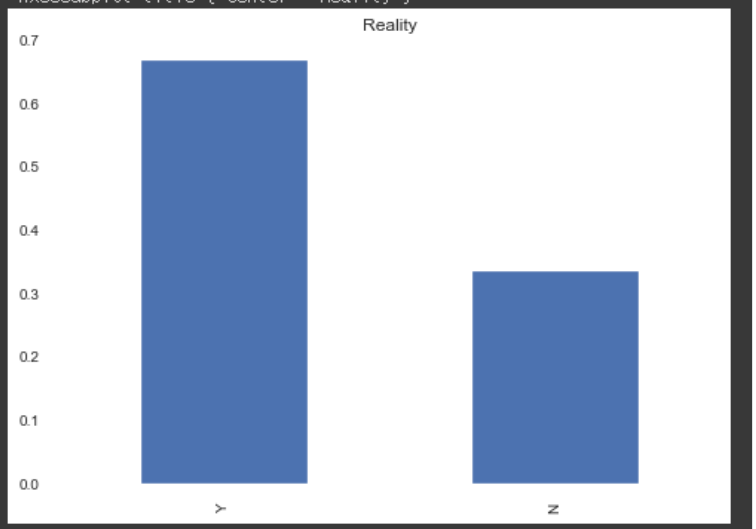
Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB

[제목 없음]




```
[ ]  df['reality'].value_counts(normalize=True) * 100

     Y    66.699425
     N    33.300575
     Name: reality, dtype: float64
```

```
[ ]  df['reality'].value_counts(normalize=True).plot.bar(title= 'Reality')
```



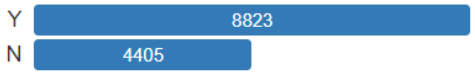
자동차를 보유하고 있지 않은 사람이 62% 보유한 사람이 37%로 자동차를 보유하지 않은 사람의 비율이 높은 것을 확인할 수 있다.

데이터 탐색

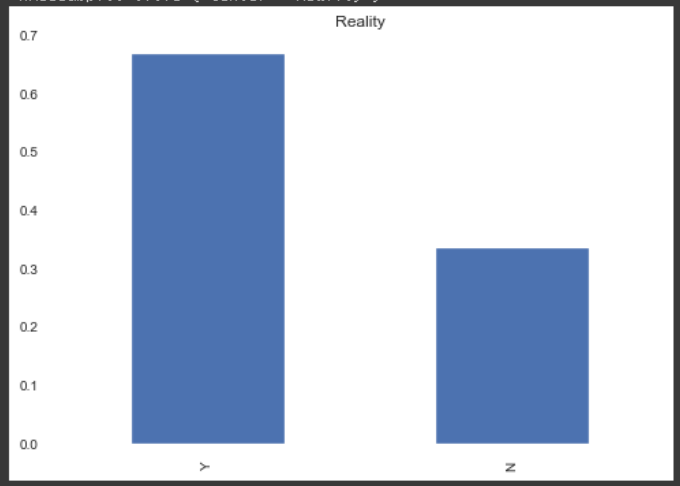
Reality 분석

reality
Boolean

Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB




```
[ ]  df['reality'].value_counts(normalize=True) * 100

Y    66.699425
N    33.300575
Name: reality, dtype: float64

[ ]  df['reality'].value_counts(normalize=True).plot.bar(title= 'Reality')

<AxesSubplot:title={'center':'Reality'}>
```



부동산 소유 여부에서는 약 67%가 소유하고 있고, 33%는 소유하고 있지 않다.

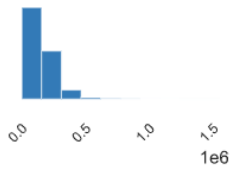
데이터 탐색

Income_total 분석

income_total
Real number (ℝ≥0)

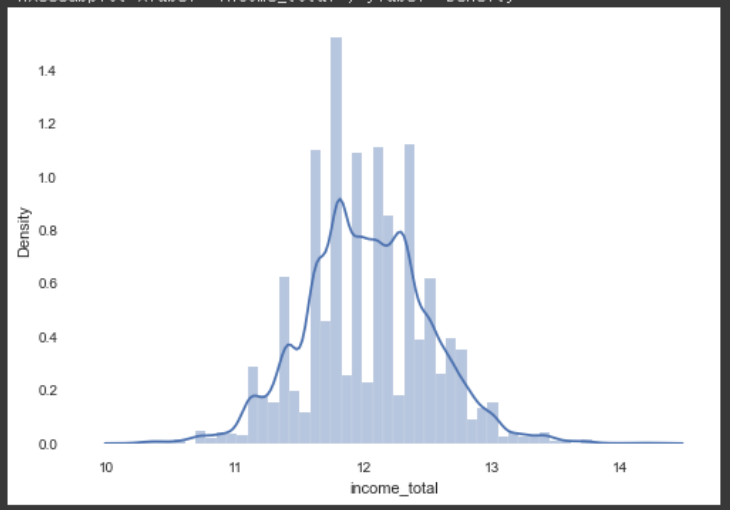
Distinct count	216
Unique (%)	1.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	188821.2474674932
Minimum	27000.0
Maximum	1575000.0
Zeros	0
Zeros (%)	0.0%
Memory size	103.3 KiB




```
[ ]  sns.distplot(df["income_total"])
```

```
<AxesSubplot:xlabel='income_total', ylabel='Density'>
```



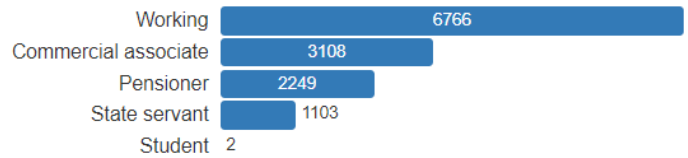
기존 income_total은 왼쪽으로 치우쳐져 있어, 모델 학습 시 log-scaling 정규화가 필요함을 알 수 있다. 오른쪽은 정규화 이후의 income_total 데이터이다. Maximum값과 minimum 값이 꽤 차이가 있다.

데이터 탐색

Income_type 분석

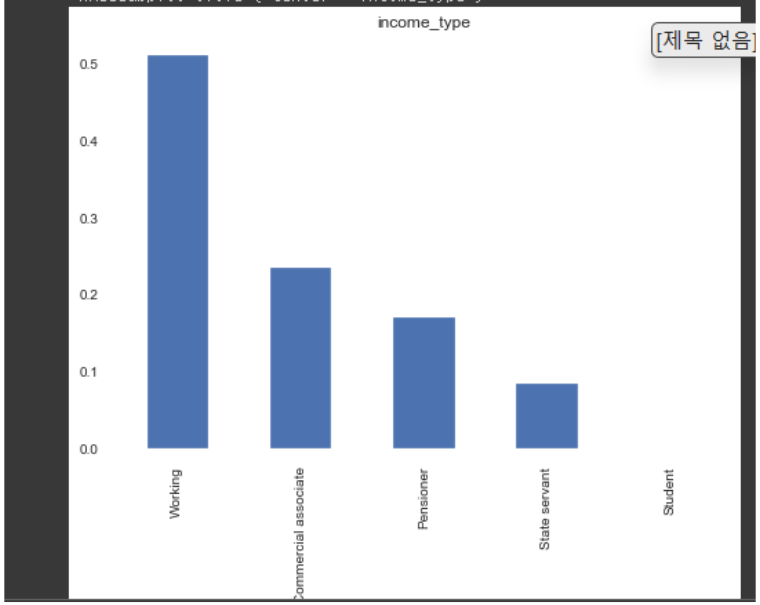
income_type
Categorical

Distinct count	5
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB




```
[ ]  df['income_type'].value_counts(normalize=True) * 100

Working               51.149078
Commercial associate  23.495615
Pensioner             17.001814
State servant          8.338373
Student                0.015119
Name: income_type, dtype: float64
```

```
[ ]  df['income_type'].value_counts(normalize=True).plot.bar(title= 'income_
```



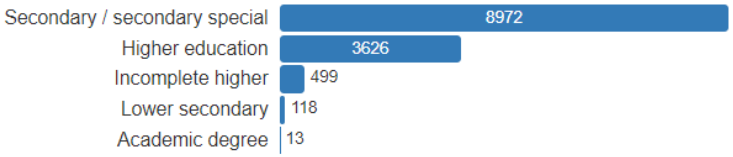
소득분류 데이터에서는 working이 51%로 과반을 차지하고 있고, commercial associate이 23%, pensioner이 17%, state servant가 8%, student가 2명으로 극히 일부분을 차지하고 있다.

데이터 탐색

Edu_type 분석

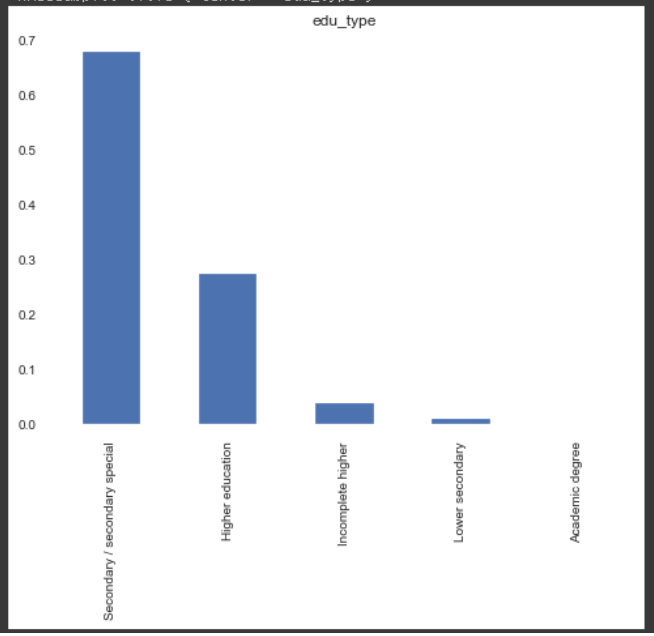
edu_type
Categorical

Distinct count	5
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB




```
[ ]  df['edu_type'].value_counts(normalize=True) * 100

Secondary / secondary special    67.825824
Higher education                 27.411551
Incomplete higher                 3.772301
Lower secondary                   0.892047
Academic degree                   0.098276
Name: edu_type, dtype: float64
```

```
[ ]  df['edu_type'].value_counts(normalize=True).plot.bar(title= 'edu_type')

<AxesSubplot:title={'center':'edu_type'}>
```



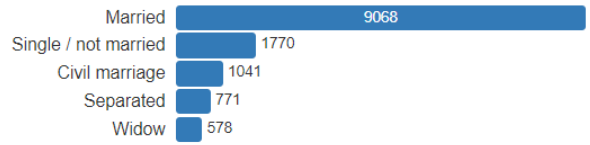
Edu-type은 Secondary/secondary special이 67%, Higher education이 27%로 두개가 대부분을 차지하고 있다.

데이터 탐색

family_type 분석

family_type
Categorical

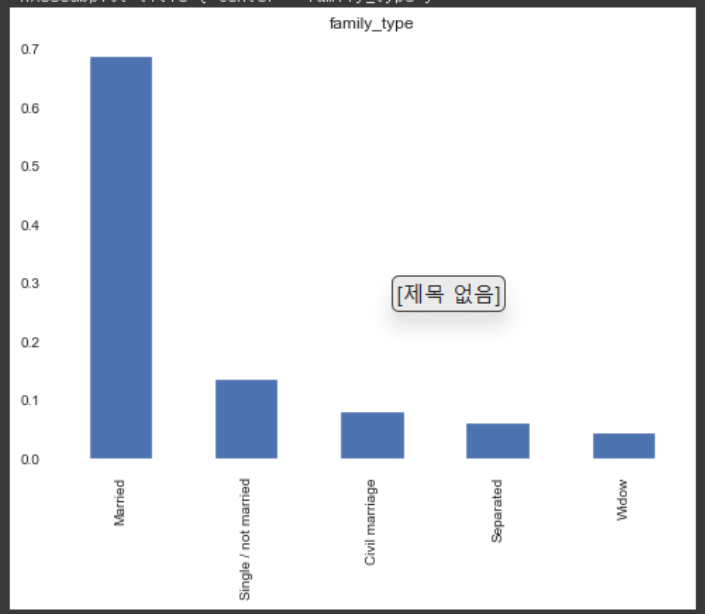
Distinct count	5
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB




```
[ ]  df['family_type'].value_counts(normalize=True) * 100

Married                 68.551557
Single / not married    13.380708
Civil marriage           7.869670
Separated                5.828546
Widow                    4.369519
Name: family_type, dtype: float64
```

```
df['family_type'].value_counts(normalize=True).plot.bar(title= 'family_type')
```

```
<AxesSubplot:title={'center':'family_type'}>
```



Family_type에서는 약 70%가 married한 것을 볼 수 있다.

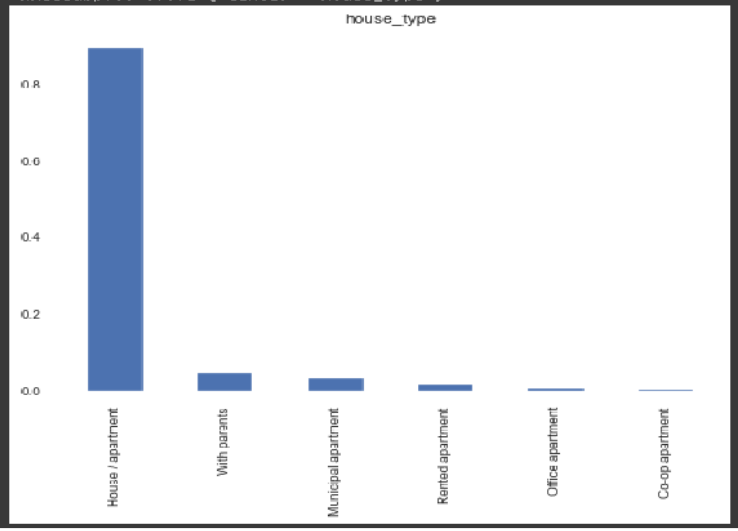
데이터 탐색

house_type 분석

house_type
Categorical

Distinct count	6
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB


```
[ ]  df['house type'].value counts(normalize=True) * 100

House / apartment      09.355912
With parents            4.747505
Municipal apartment     3.122165
Rented apartment        1.625340
Office apartment        0.703054
Co-op apartment         0.446024
Name: house_type, dtype: float64

[ ]  df['house_type'].value_counts(normalize=True).plot.bar(title= 'house_type')

<AxesSubplot:title={'center':'house_type'}>
```



주거 형태는 약 90%가 house/apartment에 거주함을 알 수 있다.

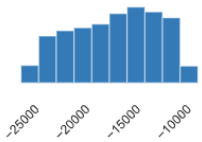
데이터 탐색

DAYS_BIRTH 분석

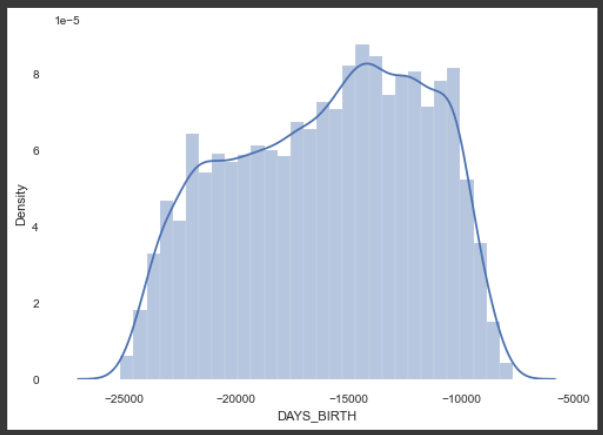
DAYS_BIRTH
Real number (ℝ)

Distinct count	5262
Unique (%)	39.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

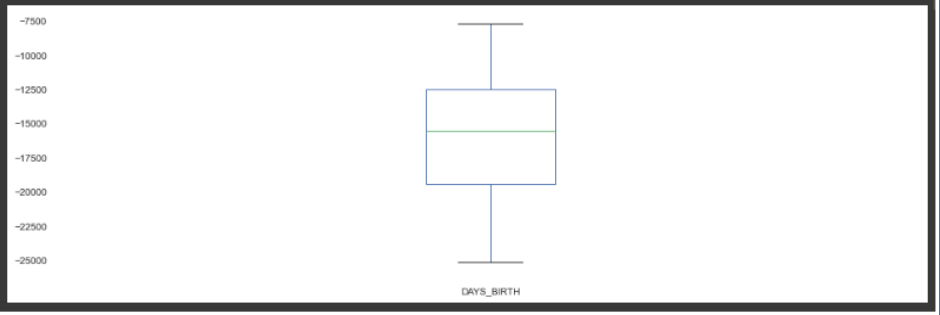
Mean	-15958.14340792259
Minimum	-25152
Maximum	-7705
Zeros	0
Zeros (%)	0.0%
Memory size	103.3 KiB




```
[ ]  sns.distplot(df["DAYS_BIRTH"]);
```




```
[ ]  df["DAYS_BIRTH"].plot.box(figsize=(16,5))
     plt.show()
```



Days_birth의 경우 다양한 값이 나오는 만큼 분포가 비교적 균형있게 나왔다.

데이터 탐색

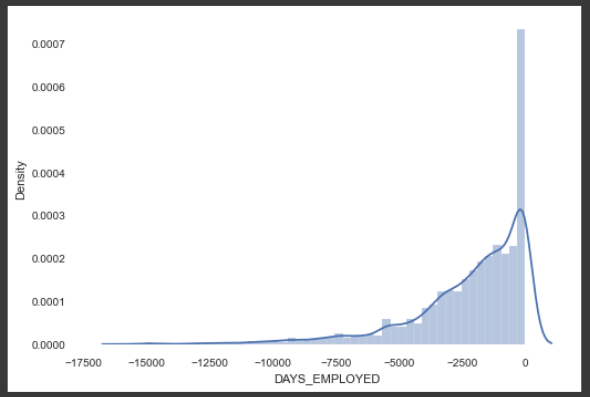
DAYS_EMPLOYED 분석

DAYS_EMPLOYED
Real number (ℝ)

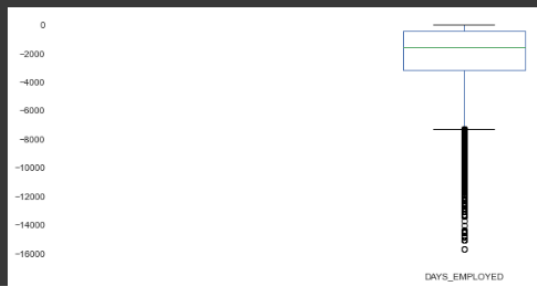
Distinct count	2947
Unique (%)	22.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	59854.03749622014
Minimum	-15713
Maximum	365243
Zeros	0
Zeros (%)	0.0%
Memory size	103.3 KiB


```
[ ] sns.distplot(df["DAYS_EMPLOYED"]);
```




```
[ ] df["DAYS_EMPLOYED"].plot.box(figsize=(16,5))
    plt.show()
```



Days_employed 데이터의 경우 '365243'데이터를 가지고 있는 사람은 고용이 되지 않은 상태인데, 나중에 전처리로 0 등으로 바꿔줘야 할 필요가 있을 것 같다. 또한 right-skewed 모형으로 정규화가 필요함을 확인했다.

데이터 탐색

~~FLAG_MOBIL~~
~~Boolean~~

CONSTANT

REJECTED

Distinct count	1
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB

1 13228

← FLAG_MOBIL(핸드폰 소유 여부)의 경우 모두 1값으로 모든 사람이 핸드폰을 가지고 있다. 무의미한 column임을 알 수 있다.

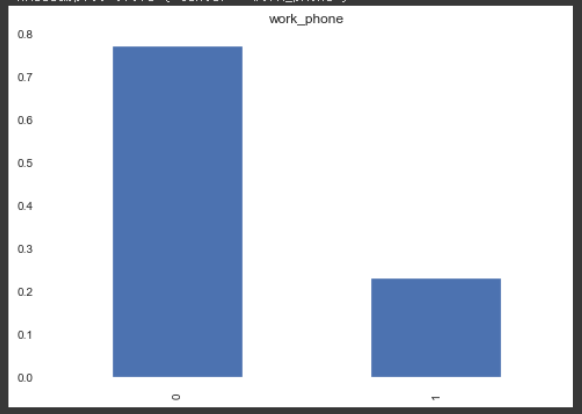
work_phone
Boolean

Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB

0 10182
1 3046

```
[ ]  df['work_phone'].value_counts(normalize=True) * 100

0    76.973087
1    23.026913
Name: work_phone, dtype: float64
```

```
[ ]  df['work_phone'].value_counts(normalize=True).plot.bar(title= 'work_phone')

<AxesSubplot:title={'center':'work_phone'}>
```

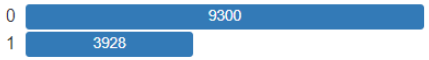


데이터 탐색

Phone 분석

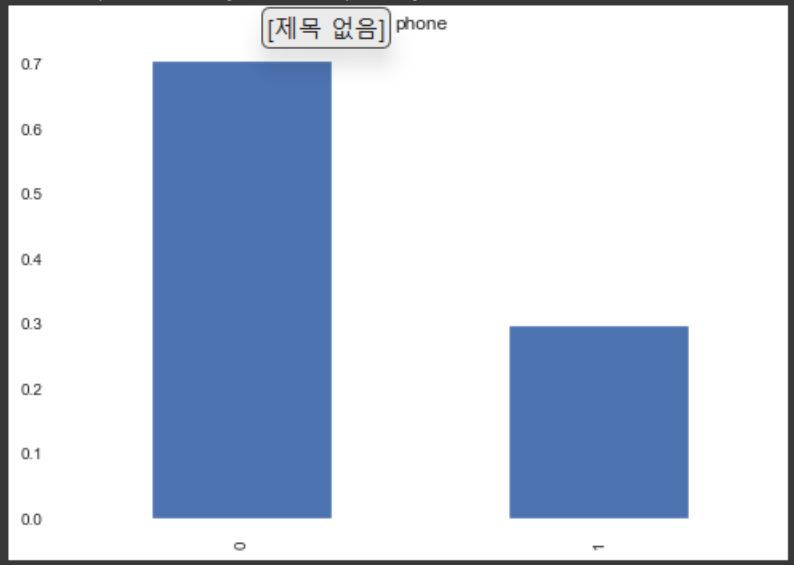
phone
Boolean

Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB




```
[ ]  df['phone'].value_counts(normalize=True) * 100

     0    70.305413
     1    29.694587
     Name: phone, dtype: float64
```

```
[ ]  df['phone'].value_counts(normalize=True).plot.bar(title= 'phone')

     <AxesSubplot:title={'center':'phone'}>
```



전화 소유 여부의 경우, 70%가 가지고 있지 않았다. 휴대전화가 아닌 집 전화의 개념인 것 같다.

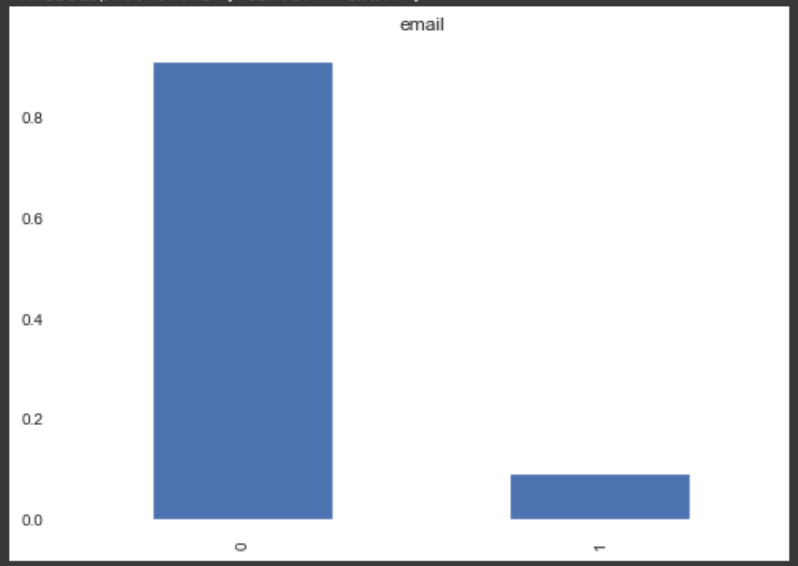
데이터 탐색

email 분석

email
Boolean

Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB


```
[ ]  df['email'].value_counts(normalize=True) * 100

     0    91.049289
     1     8.950711
     Name: email, dtype: float64

[ ]  df['email'].value_counts(normalize=True).plot.bar(title= 'email')

     <AxesSubplot:title={'center':'email'}>
```



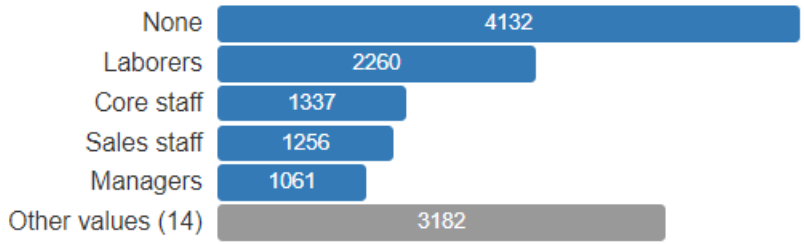
E-mail은 91%가 가지고 있지 않는다는 데이터를 확인할 수 있었다.

데이터 탐색

occyp_type

Categorical

Distinct count	19
Unique (%)	0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB




```
[ ] df.isnull().any()

gender           False
car              False
reality          False
child_num        False
income_total     False
income_type      False
edu_type         False
family_type      False
house_type       False
DAYS_BIRTH       False
DAYS_EMPLOYED    False
FLAG_MOBIL       False
work_phone       False
phone            False
email            False
occyp_type        True
family_size      False
begin_month      False
credit           False
dtype: bool
```

```
df.isnull().sum()

gender              0
car                 0
reality             0
child_num           0
income_total        0
income_type         0
edu_type            0
family_type         0
house_type          0
DAYS_BIRTH          0
DAYS_EMPLOYED       0
FLAG_MOBIL          0
work_phone          0
phone               0
email               0
occyp_type       4132
family_size         0
begin_month         0
credit              0
dtype: int64
```

Occyp_type(직업)의 경우 null값이 상당수 존재하여 전처리 과정에서 삭제를 할지, 직업이 없다고 생각할지 정해야 함을 알 수 있었다. 위의 표는 None으로 처리한 후의 비율을 확인할 수 있다.

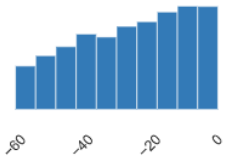
데이터 탐색

Begin_month 분석

begin_month
Real number (ℝ)

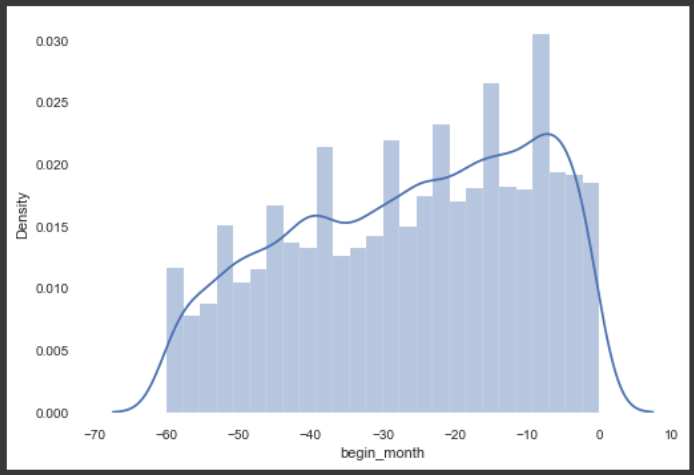
Distinct count	61
Unique (%)	0.5%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	-26.198291502872696
Minimum	-60
Maximum	0
Zeros	102
Zeros (%)	0.8%
Memory size	103.3 KiB



신용카드 발급월은 종류가 많은 만큼 다양한 데이터가 존재하였다.

```
[ ]  sns.distplot(df["begin_month"]);
```




```
[ ]  df["begin_month"].plot.box(figsize=(16,5))
     plt.show()
```

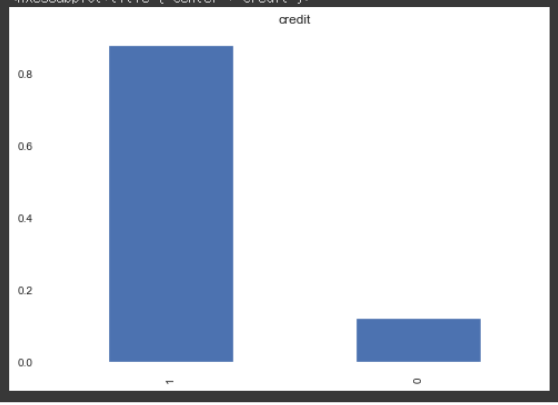
데이터 탐색

Credit 분석

credit
Boolean

Distinct count	2
Unique (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Memory size	103.3 KiB


```
[ ]  # 그래프를 통해 확인하기 위한 normalizing 작업

     df['credit'].value_counts(normalize=True)*100

     1    87.821288
     0    12.178712
     Name: credit, dtype: float64

[ ]  # 아래의 그래프를 통해 낮은 신용도를 가진 00 1에 비해 가진 데이터가 아주 작은것을 확인할 수 있다.

     df['credit'].value_counts(normalize=True).plot.bar(title = 'credit')

     <AxesSubplot:title={'center':'credit'}>
```

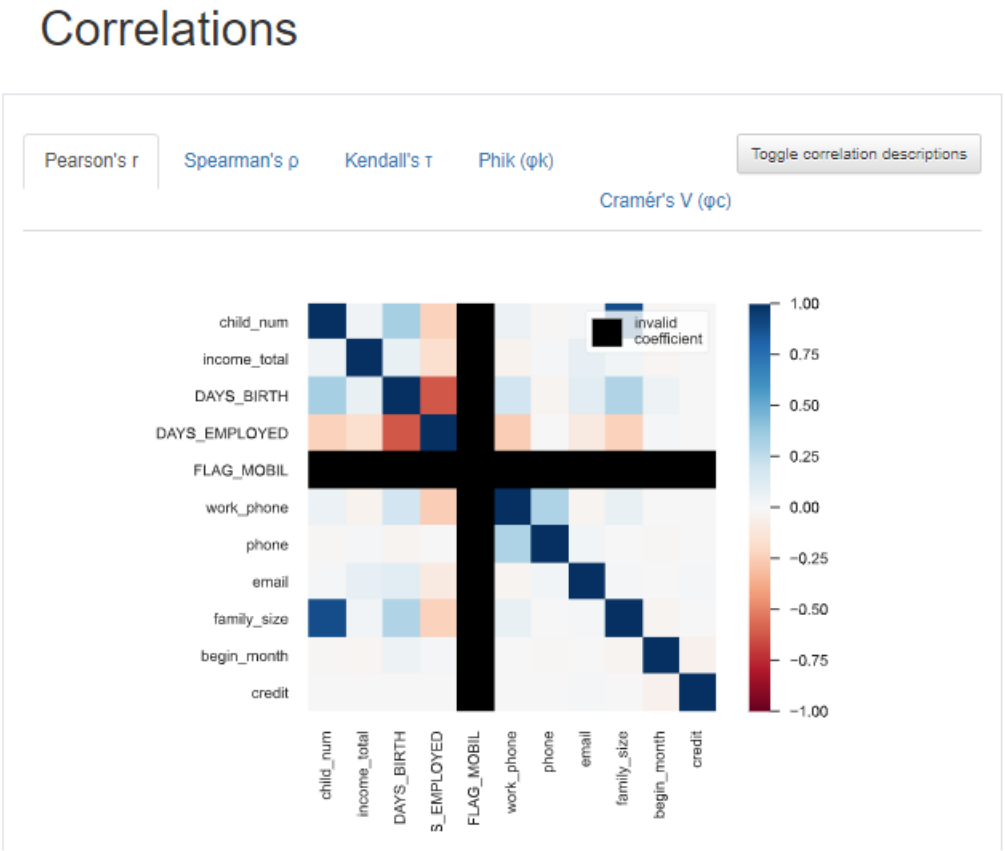


목표 데이터인 credit은 약 87% 신용도가 높다고 나와있는데, class imbalance 문제가 생길 것을 확인할 수 있다. 이에 모델 학습 시 class imbalance인 경우 사용할 수 있는 방법론을 사용해야 함을 느꼈다.

데이터 탐색

Interactions에서는 숫자로 이루어진 데이터, 각 변수들의 이중 상관관계를 확인할 수 있다. 이 과정에서 family_size와 child_num이 상관관계가 높아 다중공선성을 발생시킬 수 있음을 확인했다.

데이터 탐색



상관분석에서는 모든 변수의 상관관계를 heat map 형태로 파악할 수 있다. DAYS_EMPLOYED와 DAYS_BIRTH의 상관성이 높은 것을 확인했고, credit과 높은 상관관계를 가지는 변수가 없음을 확인했다.

데이터 탐색

Column 각각 하나씩 분석하였을 때보다 각각의 column과 credit의 관계를 분석하기 위해 다음 슬라이드와 같이 분석해 보았다.

어떤 column이 credit과 유의미한 관계를 단독적으로 가지고 있다면 그 column의 요소의 credit의 0과1의 비율이 확연하게 차이가 날 것이다.

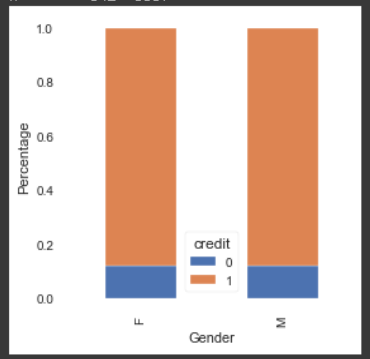
맨 처음 gender로 예를 들면, 성별이 credit에 영향을 주는 column이라면, female중에 credit의 0과1의 비율과 male의 credit 0과 1의 비율이 차이가 날 것이라고 예상하였다.

데이터 탐색

Gender 와 Credit 간의 관계

```
print(pd.crosstab(df["gender"],df["credit"]))
Gender = pd.crosstab(df["gender"],df["credit"])
Gender.div(Gender.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
plt.xlabel("Gender")
plt.ylabel("Percentage")
plt.show()
```

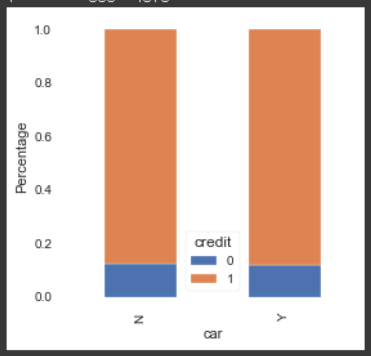
```
credit     0     1
gender
F       1069  7726
M        542  3891
```



Car 와 Credit 간의 관계

```
print(pd.crosstab(df["car"],df["credit"]))
Car = pd.crosstab(df["car"],df["credit"])
Car.div(Car.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
plt.xlabel("car")
plt.ylabel("Percentage")
plt.show()
```

```
credit     0     1
car
N       1012  7239
Y        599  4378
```



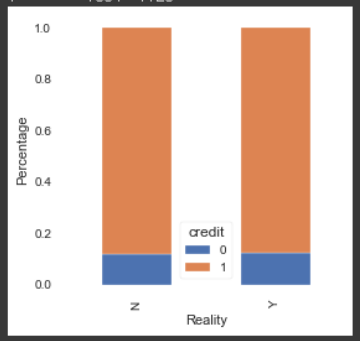
남성과 여성, 차 소지 여부는 credit의 비율차이가 거의 나지 않음을 확인했다.

데이터 탐색

Reality 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["reality"],df["credit"]))
    Reality = pd.crosstab(df["reality"],df["credit"])
    Reality.div(Reality.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Reality")
    plt.ylabel("Percentage")
    plt.show()
```

```
credit      0     1
reality
N         517  3888
Y        1094  7729
```

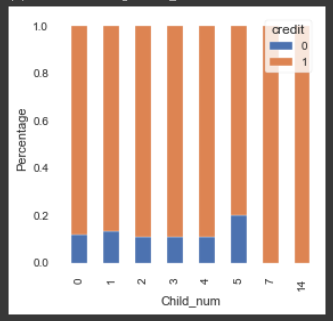


Reality의 경우에도 소지 여부에 따른 credit의 유의미한 비율 차이는 없었다.

Child_num 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["child_num"],df["credit"]))
    Child_num = pd.crosstab(df["child_num"],df["credit"])
    Child_num.div(Child_num.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Child_num")
    plt.ylabel("Percentage")
    plt.show()
```

```
credit        0     1
child_num
0          1100  8044
1           363  2388
2           128  1022
3            16   132
4             3    24
5             1     4
7             0     1
14            0     2
```



자녀수가 7명과 14명인 데이터는 credit이 모두 1이었다. 하지만 자녀수가 7명과 14명인 데이터는 13227개 중 3개이기 때문에 유의미한 영향은 없을 것으로 예상했다.

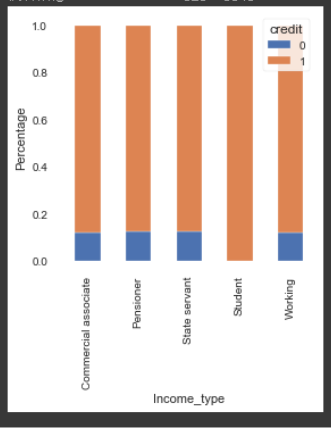
데이터 탐색

Income_type 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["income_type"],df["credit"]))
    Income_type = pd.crosstab(df["income_type"],df["credit"])
    Income_type.div(Income_type.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Income_type")
    plt.ylabel("Percentage")
    plt.show()
```

credit	0	1
income_type		
Commercial associate	376	2732
Pensioner	279	1970
State servant	136	967
Student	0	2
Working	820	5946

[제목 없음]



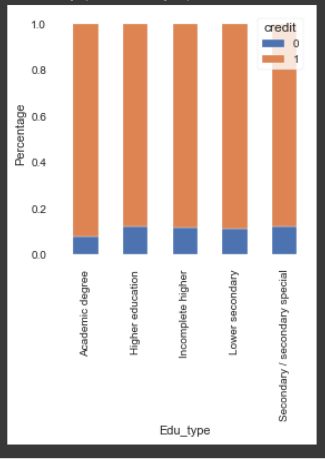
Income_type이 student인 데이터는 credit이 모두 1이었다. 하지만 student인 데이터는 13227개 중 2개이기 때문에 유의미한 데이터는 아니라고 판단했다..

Edu_type 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["edu_type"],df["credit"]))
    Edu_type = pd.crosstab(df["edu_type"],df["credit"])
    Edu_type.div(Edu_type.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Edu_type")
    plt.ylabel("Percentage")
    plt.show()
```

credit	0	1
edu_type		
Academic degree	1	12
Higher education	438	3188
Incomplete higher	59	440
Lower secondary	13	105
Secondary / secondary special	1100	7872

[제목 없음]



Academic degree의 경우 눈에 띄게 0의 값이 적은 걸 알 수 있지만 총 데이터가 13개에 지나지 않아 확신을 가질 수는 없었다.

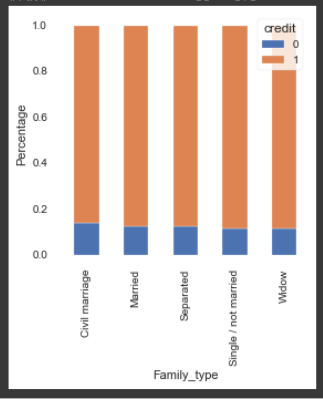
데이터 탐색

Family_type 와 Credit 간의 관계

```
[ ]  print(pd.crosstab(df["family_type"],df["credit"]))
     Family_type = pd.crosstab(df["family_type"],df["credit"])
     Family_type.div(Family_type.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
     plt.xlabel("Family_type")
     plt.ylabel("Percentage")
     plt.show()
```

credit	0	1
family_type		
Civil marriage	143	898
Married	1104	7964
Separated	96	675
Single / not married	203	1567
Widow	65	513

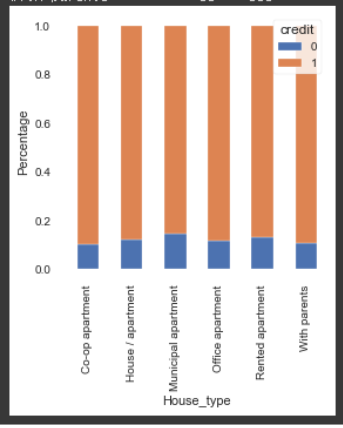
[제목 없음]



House_type 와 Credit 간의 관계

```
[ ]  print(pd.crosstab(df["house_type"],df["credit"]))
     House_type = pd.crosstab(df["house_type"],df["credit"])
     House_type.div(House_type.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
     plt.xlabel("House_type")
     plt.ylabel("Percentage")
     plt.show()
```

credit	0	1
house_type		
Co-op apartment	6	53
House / apartment	1438	10382
Municipal apartment	60	353
Office apartment	11	82
Rented apartment	28	187
With parents	68	560



Family_type 와 Credit 간의 관계, House_type 와 Credit 간의 관계 분석을 통해 유의미한 credit 비율 차이를 찾아내지는 못했다.

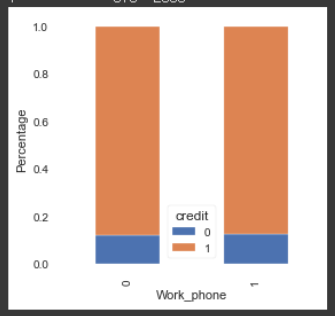
데이터 탐색

Work_phone 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["work_phone"],df["credit"]))
    Work_phone = pd.crosstab(df["work_phone"],df["credit"])
    Work_phone.div(Work_phone.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Work_phone")
    plt.ylabel("Percentage")
    plt.show()
```

```
credit          0     1
work_phone
0            1233  8949
1             378  2668
```

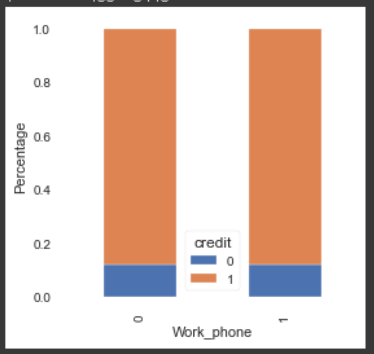
[제목 없음]



Phone 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["phone"],df["credit"]))
    Phone = pd.crosstab(df["phone"],df["credit"])
    Phone.div(Phone.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Work_phone")
    plt.ylabel("Percentage")
    plt.show()
```

```
credit     0     1
phone
0       1131  8169
1        480  3448
```



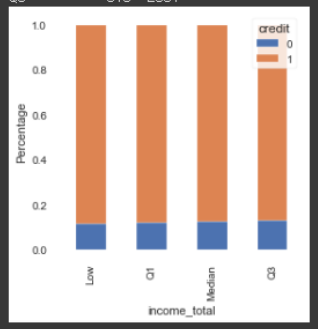
Work_phone 와 Credit 간의 관계, Phone 와 Credit 간의 관계 분석을 통해 유의미한 credit 비율 차이를 찾아내지는 못했다.

데이터 탐색

Income_total와 Credit 간의 관계

```
[ ] bins=[0,121500,157500,225000,1575000]
    group=['Low','Q1','Median', 'Q3']
    df['Income_bin']=pd.cut(df['income_total'],bins,labels=group)

[ ] print(pd.crosstab(df["Income_bin"],df["credit"]))
    Income_bin = pd.crosstab(df["Income_bin"], df["credit"])
    Income_bin.div(Income_bin.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("income_total")
    plt.ylabel("Percentage")
    plt.show()
```

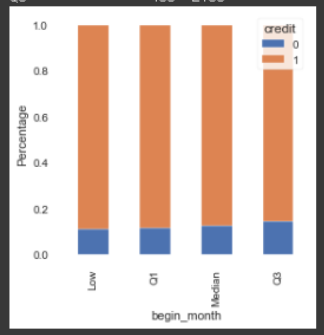
```
credit         0     1
Income_bin
Low          389  2986
Q1           394  2902
Median       450  3222
Q3           378  2507
```



Begin_month 와 Credit 간의 관계

```
[ ] bins=[-60,-40,-24,-12,0]
    group=['Low','Q1','Median', 'Q3']
    df['begin_month_bin']=pd.cut(df['begin_month'],bins,labels=group)

[ ] print(pd.crosstab(df["begin_month_bin"],df["credit"]))
    Income_bin = pd.crosstab(df["begin_month_bin"], df["credit"])
    Income_bin.div(Income_bin.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("begin_month")
    plt.ylabel("Percentage")
    plt.show()
```

```
credit             0     1
begin_month_bin
Low              351  2859
Q1               401  3136
Median           387  2755
Q3               463  2759
```



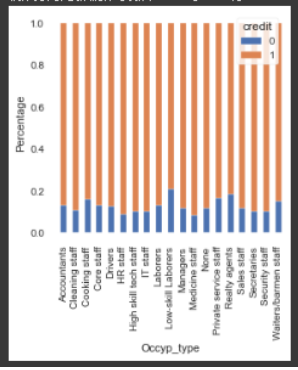
Income_total와 Credit 간의 관계, Begin_month 와 Credit 간의 관계 분석을 통해 유의미한 credit 비율 차이를 찾아내지는 못했다.

데이터 탐색

Occyp_type 와 Credit 간의 관계

```
[ ] print(pd.crosstab(df["occyp_type"],df["credit"]))
    Occyp_type = pd.crosstab(df["occyp_type"],df["credit"])
    Occyp_type.div(Occyp_type.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("Occyp_type")
    plt.ylabel("Percentage")
    plt.show()
```

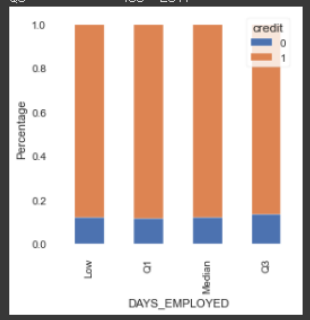
credit	0	1
occyp_type		
Accountants	59	386
Cleaning staff	21	177
Cooking staff	34	180
Core staff	175	1162
Drivers	100	691
HR staff	3	32
High skill tech staff	52	469
IT staff	2	18
Laborers	296	1964
Low-skill Laborers	13	50
Managers	122	939
Medicine staff	34	386
None	491	3641
Private service staff	19	96
Realty agents	6	27
Sales staff	148	1108
Secretaries	5	43
Security staff	23	203
Waiters/barmen staff	8	45



Days_employed 와 Credit 간의 관계

```
[ ] bins=[-15713,-3153,-1539, -401.75,365243]
    group=['Low','Q1','Median', 'Q3']
    df['Employed_bin']=pd.cut(df['DAYS_EMPLOYED'],bins,labels=group)

[ ] print(pd.crosstab(df["Employed_bin"],df["credit"]))
    Income_bin = pd.crosstab(df["Employed_bin"], df["credit"])
    Income_bin.div(Income_bin.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))
    plt.xlabel("DAYS_EMPLOYED")
    plt.ylabel("Percentage")
    plt.show()
```

credit	0	1
Employed_bin		
Low	395	2915
Q1	380	2943
Median	400	2887
Q3	436	2871



Occyp_type 와 Credit 간의 관계 분석을 통해 Low-skill Laborers가 다른 직군에 비해 신용도가 낮을 확률이 있을 수 있음을 확인했다.

전처리 방법

앞선 EDA를 바탕으로 총 14가지의 전처리 실행

→ occupy_type null 값 None으로 변경

→ FLAG_MOBIL column 삭제

→ 새로운 column ID 생성

→ income_total log scaling

→ 음수 데이터 양수 변환

→ DAYS_EMPLOYED 365243 값 0으로 변경

→ DAYS_EMPLOYED log scaling

→ Phone column 삭제

→ child_num column 삭제

→ Categorical data에 대한 one hot incoding

→ 범주가 많은 데이터 ordinary encoding

→ MinMax Scaling

→ K-Means clustering

→ SOM

전처리 방법

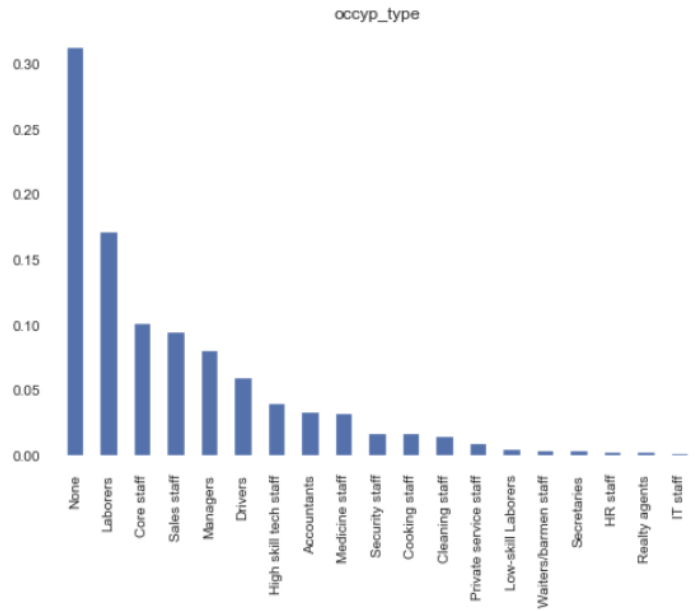
occupy_type null 값 None으로 변경

```
df.isnull().sum()
```

gender	0
car	0
reality	0
child_num	0
income_total	0
income_type	0
edu_type	0
family_type	0
house_type	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
FLAG_MOBIL	0
work_phone	0
phone	0
email	0
occyp_type	4132
family_size	0
begin_month	0
credit	0

dtype: int64

```
df = df.fillna('None')
```



EDA를 통해 occyp_type의 경우에만 총 13227개의 데이터 중에 4132개의 데이터가 null 값으로 이루어진 것을 확인할 수 있다. Occyp_type은 직업 유형으로 null의 의미가 직업이 없는 사람을 뜻한다면 신용도를 평가하는데 의미가 있을 수 있기 때문에 None으로 바꾸어 주었다.

전처리 방법

FLAG_MOBIL column 삭제

```
X['FLAG_MOBIL'].value_counts()
```

```
1    13228
Name: FLAG_MOBIL, dtype: int64
```

```
df = df.drop(columns=['FLAG_MOBIL'])
```

EDA를 통해 FLAG_MOBIL의 경우 모든 데이터가 1로만 이루어 진 것을 확인할 수 있었다. 이는 학습에 도움이 되지 않으므로 drop을 활용하여 FLAG_MOBIL column을 삭제해 주었다.

전처리 방법

새로운 column ID 생성

```
df['ID'] = ₩
df['child_num'].astype(str) + '_' + df['income_total'].astype(str) + '_' +₩
df['DAYS_BIRTH'].astype(str) + '_' + df['DAYS_EMPLOYED'].astype(str) + '_' +₩
df['work_phone'].astype(str) + '_' + df['phone'].astype(str) + '_' +₩
df['email'].astype(str) + '_' + df['family_size'].astype(str) + '_' +₩
df['gender'].astype(str) + '_' + df['car'].astype(str) + '_' +₩
df['reality'].astype(str) + '_' + df['income_type'].astype(str) + '_' +₩
df['edu_type'].astype(str) + '_' + df['family_type'].astype(str) + '_' +₩
df['house_type'].astype(str) + '_' + df['occyp_type'].astype(str)
```

```
df['ID']

0        0_202500.0_-19031_365243_0_0_0_2_F_Y_Y_Pension...
1        1_157500.0_-15773_-309_0_1_0_3_F_N_N_Working_H...
2        0_135000.0_-13483_-1816_1_1_0_2_M_Y_N_Working_...
3        2_112500.0_-12270_-150_0_1_0_4_F_Y_N_Working_S...
4        1_225000.0_-16175_-2371_0_0_0_3_M_Y_Y_Working_...
                               ...
13223    0_225000.0_-20657_-5637_0_0_0_1_F_N_N_Working_...
13224    0_292500.0_-18409_-3482_0_0_0_2_F_Y_Y_Commerci...
13225    0_135000.0_-14625_-7827_0_1_1_2_F_N_Y_Working_...
13226    1_157500.0_-10676_-2326_0_1_1_3_M_N_Y_Commerci...
13227    2_67500.0_-11925_-1621_0_0_0_4_F_Y_Y_Working_H...
Name: ID, Length: 13228, dtype: object
```

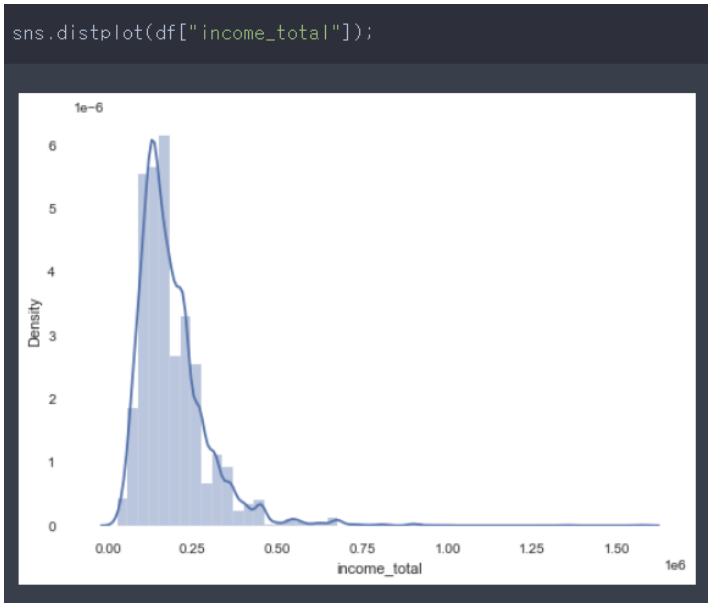
```
df['ID'].value_counts()

0_297000.0_-15519_-3234_0_0_0_1_F_N_Y_Commercial associate_Secondary / secondary special_Single / not married_Rented apartment_Laborers
19
2_225000.0_-16768_-3088_1_0_0_4_M_N_N_Working_Higher education_Civil marriage_House / apartment_Laborers
15
0_225000.0_-22976_365243_0_0_0_1_F_N_Y_Pensioner_Secondary / secondary special_Single / not married_House / apartment_nan
14
1_562500.0_-13790_-5639_1_1_0_3_M_N_Y_Working_Incomplete higher_Married_House / apartment_nan
13
0_130500.0_-13520_-5488_0_0_0_2_F_N_Y_Working_Secondary / secondary special_Married_House / apartment_Core staff
11

..
0_360000.0_-20929_-2046_0_0_1_1_F_Y_Y_Commercial associate_Higher education_Single / not married_House / apartment_Managers
1
0_67500.0_-23300_365243_0_1_0_1_F_N_Y_Pensioner_Secondary / secondary special_Widow_House / apartment_nan
1
0_189000.0_-20701_-1746_0_0_0_1_F_N_N_Working_Secondary / secondary special_Single / not married_House / apartment_Sales staff
1
0_202500.0_-10754_-3870_1_1_0_1_F_Y_Y_Working_Higher education_Single / not married_House / apartment_Accountants
1
0_112500.0_-22408_-2685_0_1_0_2_F_N_Y_Working_Secondary / secondary special_Married_House / apartment_Cooking staff
1
Name: ID, Length: 6487, dtype: int64
```

EDA를 통해 begin_month만 다를 뿐 같은 사람에 대한 여러 credit 결과가 존재하는 것을 확인할 수 있었다. 같은 사람에 대한 credit 결과는 의미가 있을 수 있기 때문에 새로운 ID column을 생성하여 new feature로 사용하였다. 결과를 확인해보면 많게는 19개의 데이터가 동일한 사람인 것을 확인할 수 있다. 이는 추후 ordinary incoding 통해 처리해 주었다.

전처리 방법

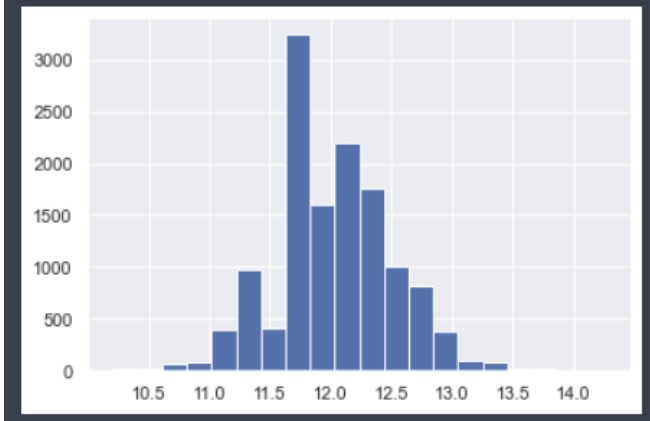
income_total log scaling

```
sns.distplot(df["income_total"]);
```




```
df['income_total'] = np.log(df['income_total'])
df['income_total'].hist(bins=20)
```

```
<AxesSubplot:>
```



EDA를 통해 income_total data가 left skewed 된 것을 확인할 수 있었다. Skewed 되어있는 값을 그대로 학습시키면 꼬리 부분이 상대적으로 모델에 영향이 거의 없이 학습된다. 꼬리 부분이 노이즈가 아닌 유의미한 데이터일 때 문제가 발생할 수 있으므로 log scaling을 통해 정규화를 진행했다. 정규분포를 따르는 데이터로 예측을 진행할 때 더욱 신뢰할 수 있는 모델이 만들어 진다.

전처리 방법

음수 데이터 양수 변환

```
df['DAYS_EMPLOYED'] = df['DAYS_EMPLOYED'] * -1
df['DAYS_BIRTH'] = df['DAYS_BIRTH'] * -1
df['begin_month'] = df['begin_month'] * -1
```

```
df[['DAYS_EMPLOYED', 'DAYS_BIRTH', 'begin_month']]
```

	DAYS_EMPLOYED	DAYS_BIRTH	begin_month
0	-365243	19031	53
1	309	15773	26
2	1816	13483	9
3	150	12270	12
4	2371	16175	3
...	...	...	...
13223	5637	20657	43
13224	3482	18409	53
13225	7827	14625	34
13226	2326	10676	16
13227	1621	11925	4

13228 rows × 3 columns

EDA를 통해 DAYS_EMPLOYED, DAYS_BIRTH, begin_month의 경우 데이터들이 음수로 이루어 진 것을 확인할 수 있었다. 음수 값은 시각화도 쉽지 않기 때문에 양수로 바꾸어 주었다. DAYS_EMPLOYED에서 양수였던 값이 음수로 변환되었는데 이는 다음 슬라이드에서 처리 방법을 제시하였다.

전처리 방법

DAYS_EMPLOYED 365243 값 0으로 변경

```
df.loc[df['DAYS_EMPLOYED'] == -365243]['DAYS_EMPLOYED']
```

```
0       -365243
6       -365243
7       -365243
12      -365243
14      -365243
          ...
13191   -365243
13193   -365243
13195   -365243
13207   -365243
13221   -365243
Name: DAYS_EMPLOYED, Length: 2247, dtype: int64
```

```
df.loc[df['DAYS_EMPLOYED'] == -365243, 'DAYS_EMPLOYED'] = 0
```

```
df.loc[df['DAYS_EMPLOYED'] == 0]['DAYS_EMPLOYED']
```

```
0       0
6       0
7       0
12      0
14      0
       ..
13191   0
13193   0
13195   0
13207   0
13221   0
Name: DAYS_EMPLOYED, Length: 2247, dtype: int64
```

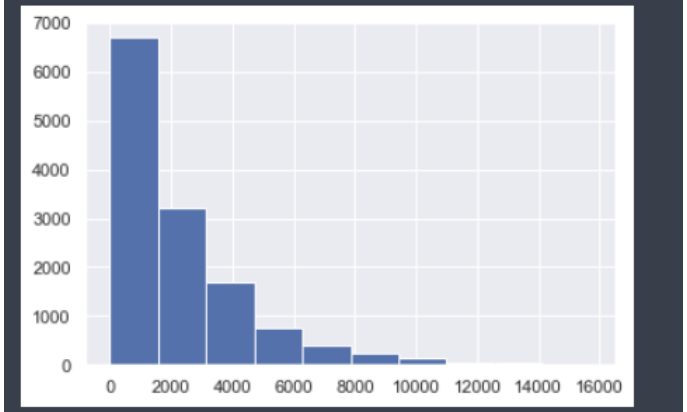
DAYS_EMPLOYED의 경우 원래는 양수 값으로 365243은 고용되지 않음을 뜻한다. 앞 슬라이드의 음수 값을 양수로 바꾸어 주는 전처리로 인해 -365243으로 값이 변경되었는데, 일을 하루도 하지 않음은 0으로 표현해야 함으로 처리해 주었다.

전처리 방법

DAYS_EMPLOYED log scaling

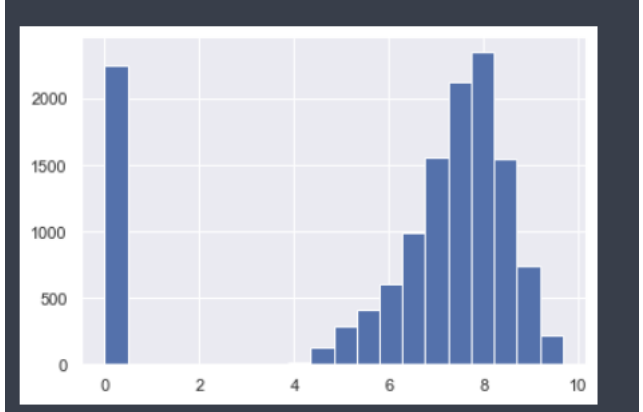
```
df['DAYS_EMPLOYED'].hist()
```

```
<AxesSubplot:>
```




```
df['DAYS_EMPLOYED'] = np.log1p(df['DAYS_EMPLOYED'])
df['DAYS_EMPLOYED'].hist(bins=20)
```

```
<AxesSubplot:>
```



DAYS_EMPLOYED의 경우 left skewed된 그래프로 income_total 때와 같이 꼬리 데이터도 학습에 잘 반영하기 위해 log scaling을 진행해 주었다. 0값은 고용되지 않음을 뜻하기 때문에 scaling에서 제외하여 그대로 유지해주었다.

전처리 방법

Phone column 삭제

```
temp=pd.concat([train, train_label],axis=1)
temp.corr()
```

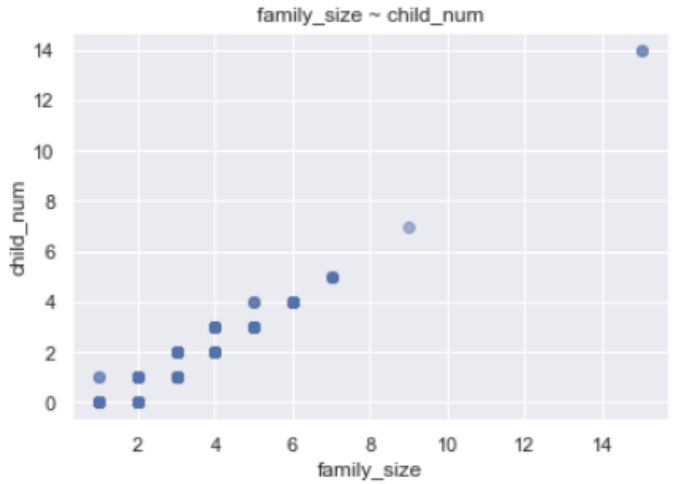
	child_num	income_total	DAYS_BIRTH	DAYS_EMPLOYED	work_phone	phone	email	family_size	begin_month	credit
child_num	1.000000	0.036664	0.331566	-0.230904	0.055515	-0.012421	0.010769	0.887714	-0.009875	0.001773
income_total	0.036664	1.000000	0.072416	-0.163243	-0.032231	0.014885	0.084248	0.030857	-0.015968	-0.007616
DAYS_BIRTH	0.331566	0.072416	1.000000	-0.622112	0.188929	-0.029370	0.101710	0.297342	0.051113	-0.007499
DAYS_EMPLOYED	-0.230904	-0.163243	-0.622112	1.000000	-0.247447	-0.006760	-0.089023	-0.227730	0.014770	-0.003447
work_phone	0.055515	-0.032231	0.188929	-0.247447	1.000000	0.311056	-0.029336	0.072938	-0.003033	-0.003863
phone	-0.012421	0.014885	-0.029370	-0.006760	0.311056	1.000000	0.028640	0.003233	-0.012166	-0.000820
email	0.010769	0.084248	0.101710	-0.089023	-0.029336	0.028640	1.000000	0.011273	-0.000435	0.011495
family_size	0.887714	0.030857	0.297342	-0.227730	0.072938	0.003233	0.011273	1.000000	-0.027298	-0.002398
begin_month	-0.009875	-0.015968	0.051113	0.014770	-0.003033	-0.012166	-0.000435	-0.027298	1.000000	-0.042281
credit	0.001773	-0.007616	-0.007499	-0.003447	-0.003863	-0.000820	0.011495	-0.002398	-0.042281	1.000000

```
df = df.drop(columns=['phone'])
```

EDA를 통해 Phone column의 경우 credit과의 상관관계가 가장 낮은 것을 확인했으므로 overfitting 방지를 위해 제거해주었다.

전처리 방법

child_num column 삭제




```
df = df.drop(columns=['child_num'])
```

EDA를 통해 family_size, child_num 상관관계가 매우 높은 것을 확인했다. 다중공선성으로 인한 파라미터 불안정성을 제거하기 위해 child_num의 경우 family_size에 포함되는 개념으로 생각되어 child_num column을 삭제해주었다.

전처리 방법

Categorical data에 대한 one hot incoding

```
df[['gender', 'car', 'reality']]
```

	gender	car	reality
0	F	Y	Y
1	F	N	N
2	M	Y	N
3	F	Y	N
4	M	Y	Y
...	...	...	...
13223	F	N	N
13224	F	Y	Y
13225	F	N	Y
13226	M	N	Y
13227	F	Y	Y

13228 rows × 3 columns

```
df.loc[df['gender'] == 'F', 'gender'] = 0
df.loc[df['gender'] == 'M', 'gender'] = 1
```

```
df.loc[df['reality'] == 'Y', 'reality'] = 1
df.loc[df['reality'] == 'N', 'reality'] = 0
```

```
df.loc[df['car'] == 'Y', 'car'] = 1
df.loc[df['car'] == 'N', 'car'] = 0
```

```
df[['gender', 'car', 'reality']]
```

	gender	car	reality
0	0	1	1
1	0	0	0
2	1	1	0
3	0	1	0
4	1	1	1
...	...	...	...
13223	0	0	0
13224	0	1	1
13225	0	0	1
13226	1	0	1
13227	0	1	1

13228 rows × 3 columns

Gender, car, reality의 경우 F, M이거나 Y, N처럼 이항 데이터인 것을 확인할 수 있다. 이를 학습 데이터로 사용하기 위해 one hot incoding으로 1, 0 변환을 진행해주었다.

전처리 방법

범주가 많은 데이터 ordinary encoding

```
categorical_feats = ['income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type', 'ID']

encoder = OrdinalEncoder(categorical_feats)
df[categorical_feats] = encoder.fit_transform(df[categorical_feats], df['credit'])

df['ID'] = df['ID'].astype('int64')
```

```
df[['income_type', 'edu_type', 'family_type', 'house_type', 'occyp_type', 'ID']]
```

	income_type	edu_type	family_type	house_type	occyp_type	ID
0	1	1	1	1	1	1
1	2	2	1	1	2	2
2	2	1	1	1	3	3
3	2	1	1	1	4	4
4	2	1	1	1	5	5
...	...	...	...	...	...	...
13223	2	1	4	1	9	4791
13224	3	2	1	1	2	1863
13225	2	1	5	1	1	6486
13226	3	1	1	1	3	1330
13227	2	2	1	1	1	6487

13228 rows × 6 columns

범주가 많은 데이터의 경우 ordinary encoding을 해주었다. category_encoders 라이브러리에서 OrdinalEncoder를 import하여 사용하였다.

Ordinary encoding의 경우 데이터 간의 선후 관계가 생겨 잘못 학습될 경우가 있지만 범주가 많아 모두 one hot encoding을 진행 시 column이 너무 많아 지기 때문에 ordinary encoding을 진행했다.

전처리 방법

MinMax Scaling

```
# Min-Max Normalization
scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(new_train)

# df로 정리해서 확인
new_train_scaled = pd.DataFrame(normalized_data, columns= new_train.columns)
new_train_scaled.head()
```

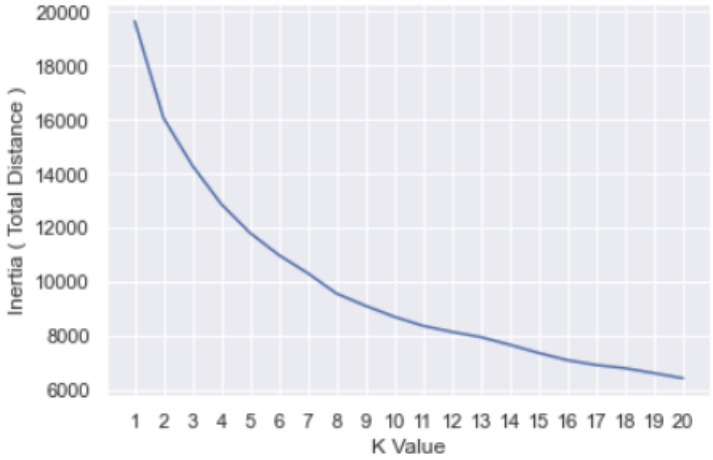
	gender	car	reality	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH	DAYS_EMPLOYED	work_p
0	0.0	1.0	1.0	0.495528	0.00	0.00	0.0	0.0	0.649166	0.000000	0.0
1	0.0	0.0	0.0	0.433722	0.25	0.25	0.0	0.0	0.462429	0.593706	0.0
2	1.0	1.0	0.0	0.395811	0.25	0.00	0.0	0.0	0.331174	0.776724	1.0
3	0.0	1.0	0.0	0.350973	0.25	0.00	0.0	0.0	0.261650	0.519263	0.0
4	1.0	1.0	1.0	0.521439	0.25	0.00	0.0	0.0	0.485470	0.804310	0.0

데이터들이 각 column에 따라 각자 다른 단위를 가지고 있을 때 상대적으로 범위가 넓은 변수는 거리를 계산하는 과정에서 문제를 일으킬 수 있다(k-means, som). 딥러닝에서는 scaling 되지 않은 데이터를 사용할 경우 학습하는데 오랜 시간이 걸릴 수 있으므로 데이터에 대해서 MinMax scaling을 진행하였다.

전처리 방법

(1) K-Means clustering (Elbow method)

```
def cluster_variance(n):
    variances=[]
    kmeans=[]
    outputs=[]
    K=[i for i in range(1,n+1)]
    for i in range(1,n+1):
        variance=0
        model=KMeans(n_clusters=i,random_state=1,verbose=2).fit(new_train_scaled)
        kmeans.append(model)
        variances.append(model.inertia_)

    return variances,K,n
variances,K,n=cluster_variance(20)
plt.plot(K,variances)
plt.ylabel("Inertia ( Total Distance )")
plt.xlabel("K Value")
plt.xticks([i for i in range(1,n+1)])
plt.show()
```



K-Means clusterin을 진행하기 전, Elbow method란 군집분석에서 군집수를 결정하는 방법이다. 군집수에 따라 군집내 총 제곱합(WSS)을 플롯팅하여 팔꿈치의 위치를 일반저그로 적절한 군집수로 선택한다. Elbow method를 사용한 결과 정확한 팔꿈치 위치는 발생하지 않았지만 그래프 상10개의 클러스터 정도면 괜찮은 정도라고 가정하여 진행하였다.

전처리 방법

(2) K-Means clustering

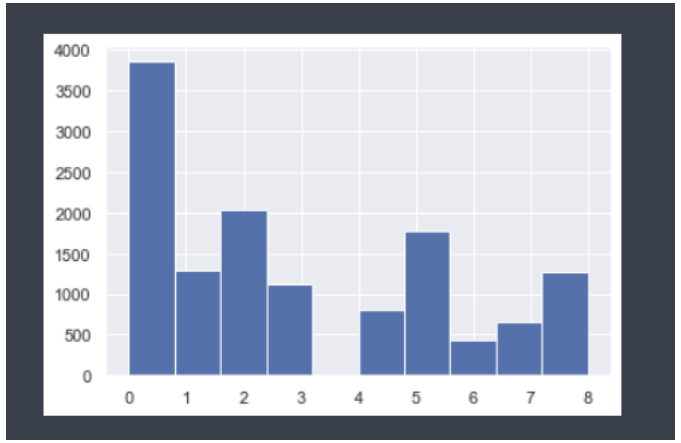
```
k = 10

# 그룹 수, random_state 설정
model = KMeans(n_clusters = k, random_state = 1)

# 정규화된 데이터에 학습
model.fit(new_train_scaled)

# 클러스터링 결과 각 데이터가 몇 번째 그룹에 속하는지 저장
new_train_scaled['cluster'] = model.fit_predict(new_train_scaled)
```

Elbow method를 통해 나온 10개의 군집으로 데이터를 나누고 cluster라는 새로운 column을 만들어 이후 사용될 모델에 도움이 될 feature로 사용하였다. 데이터를 군집화하여 새로운 column으로 사용하면 더 좋은 학습이 될 것이라는 가정에서 비롯되었다. sklearn.cluster 라이브러리 안에서 Kmeans를 import하여 사용하였다.

전처리 방법

SOM

```
som = MiniSom(x= 3, y= 3, input_len= 16, sigma = 1.0, learning_rate = 0.5, random_seed=1)
som.random_weights_init(som_data)
som.train_random(data = som_data, num_iteration = 100)
som_shape = (3, 3)
```



SOM이란 사람이 눈으로 볼 수 있는 저차원(2차원 내지 3차원) 격자에 고차원 데이터의 각 개체들이 대응하도록 인공신경망과 유사한 방식의 학습을 통해 군집을 도출해내는 기법이다. 앞선 K-means clustering에서 10개의 군집으로 분류한 것을 기반으로 3 * 3 = 9개의 군집으로 분류하였다.

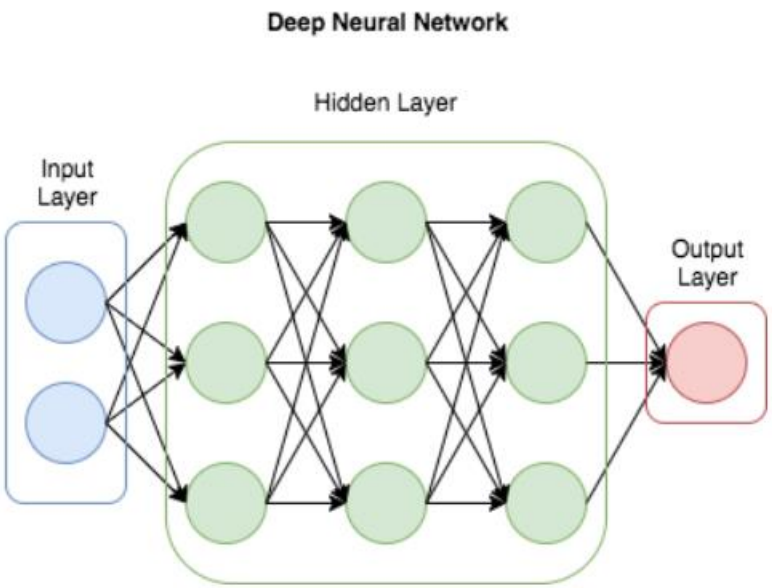
K-means 때와 같이 데이터를 군집화하여 새로운 column으로 사용하면 더 좋은 학습이 될 것이라는 가정에서 비롯되었다. Minisom 라이브러리에서 MiniSom을 import하여 진행하였으며 이는 cluster라는 새로운 column으로 추가되어 모델 학습 시 사용되었다.

전처리 방법

최종 데이터

	credit data
new_train.csv : 전처리 train data	new_train_label.csv
new_train_scaled.csv : 전처리 + scaling data	new_train_scaled _label.csv
new_train_scaled+kmeans.csv : 전처리 + scaling + kmeans	new_train_scaled _label +kmeans.csv
new_train_scaled+som.csv : 전처리 + scaling + som	new_train_scaled _label +som.csv
new_test.csv : 전처리 test data	new_test _label.csv
new_test_scaled.csv : 전처리 + scaling data	new_test _label _scaled.csv
new_test_scaled+kmeans.csv : 전처리 + scaling + kmeans	new_test _label _scaled+kmeans.csv
new_test_scaled+som.csv : 전처리 + scaling + som	new_test _label _scaled+som.csv

모델 소개 - DNN



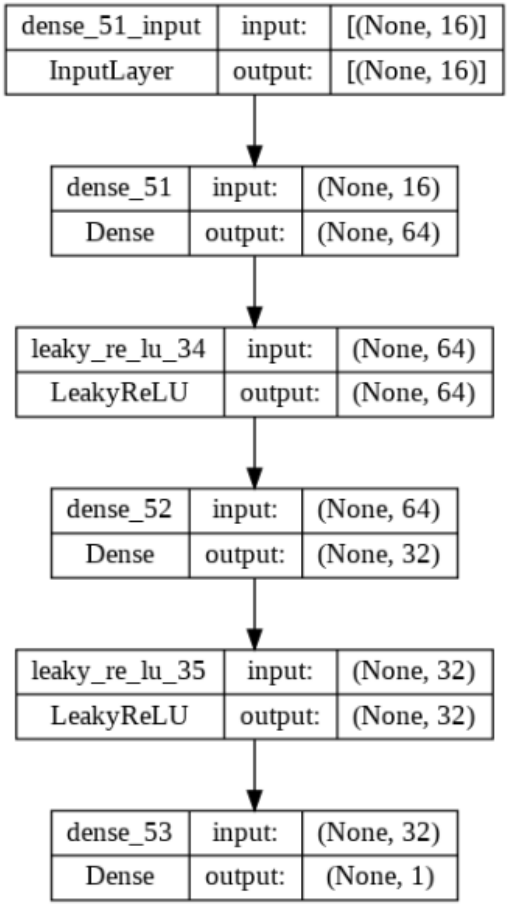
- DNN은 입력층(input layer)과 출력층(output layer) 사이에 2개 이상의 은닉층(hidden layer)들로 이뤄진 인공신경망이다.
  DNN은 주로 분류 및 수치예측을 위해 사용한다.

- DNN의 장점으로는 연속형, 범주형 변수에 상관없이 모두 분석 가능하고, 입력 변수들 간의 비선형 조합이 가능하다.

- DNN의 단점으로는 데이터 양이 적으면 성능이 좋지 않다.

분류 예측 – 모델 학습(DNN)

제작한 DNN 모델 그림



DNN 모델 정의 코드

```
def create_model(input_add=0):
    model = Sequential()
    input_shape_ = 16+input_add
    model.add(Dense(64, input_shape=(input_shape_,)))
    model.add(LeakyReLU())
    model.add(Dense(32, input_shape=(64,)))
    model.add(LeakyReLU())
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

    model_graph = tf.keras.utils.plot_model(model,show_shapes=True, show_layer_names=True)
    return model, model_graph
```

은닉층의 활성화 함수로 LeakyReLU,
opimizer는 adam을 설정

분류 예측 – 모델 학습(DNN)

제작한 DNN 모델 그림

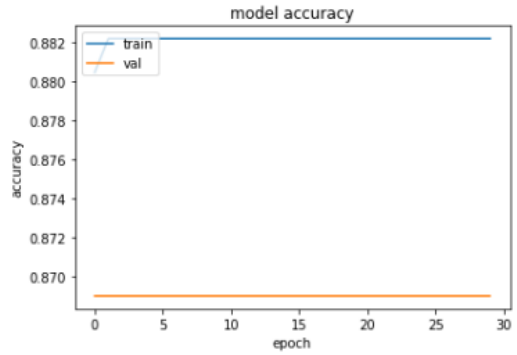
```
def train_each(data_type=None):
    data_add = 0
    if data_type:
        data_add = 1

    X, Y = read_data(data_type)

    classifier, model_graph = create_model(data_add)

    train_history = classifier.fit(X, Y,
                                   epochs=30,
                                   batch_size=16,
                                   validation_split=0.3,
                                   shuffle=True)
```

- 하이퍼 파라미터로 epochs=30, batch size=16로 설정
- train set: val set = 7:3 (데이터 shuffle적용)
- For 문을 통하여 데이터 전처리에 따라서 변형된 3개의 입력 데이터를 각각 학습

분류 예측 – 모델 학습(DNN)

Regular Data 학습 결과

Log Loss Score: 0.360401




```
Y = write_values(model_regular)
print("-"*40)
print("The proportion of prediction values of 1 is:")
print(f"{Y.sum()/len(Y)*100:.2f}%")
```

```
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
----------------------------------------
The proportion of prediction values of 1 is:
100.00%
```

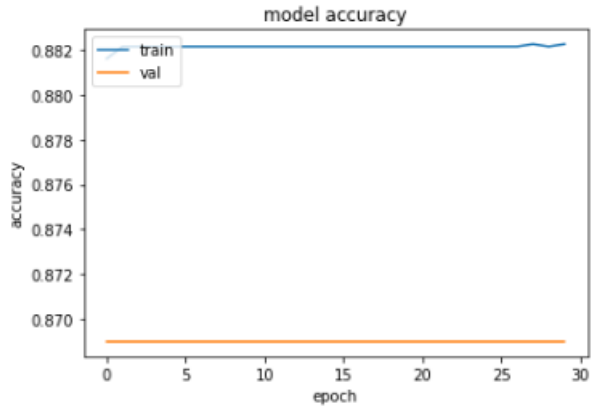
Log Loss Score: 0.360401

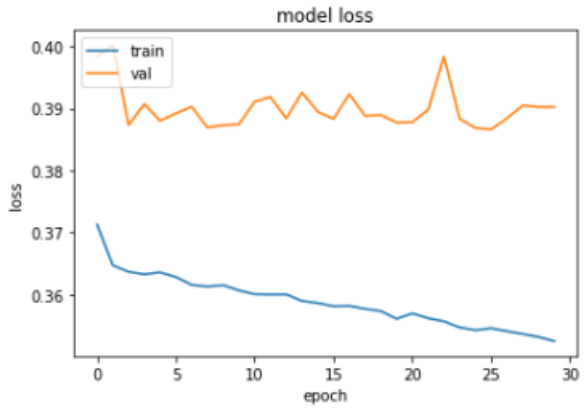
Log loss는 높지 않지만 모델이 데이터의 최적 label을 모두 1값으로 예측하였고 이는 local mimina에 빠진 것이다. 따라서 분류 예측에 적절하지 않은 학습이다.

분류 예측 – 모델 학습(DNN)

Kmeans data 학습 결과

Log Loss Score: 0.361309






```
Y = write_values(model_kmeans, "kmeans")
print("-"*40)
print("The proportion of prediction values of 1 is:")
print(f"{Y.sum()/len(Y)*100:.2f}%")
```

```
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
----------------------------------------
The proportion of prediction values of 1 is:
100.00%
```

Log Loss Score: 0.361309

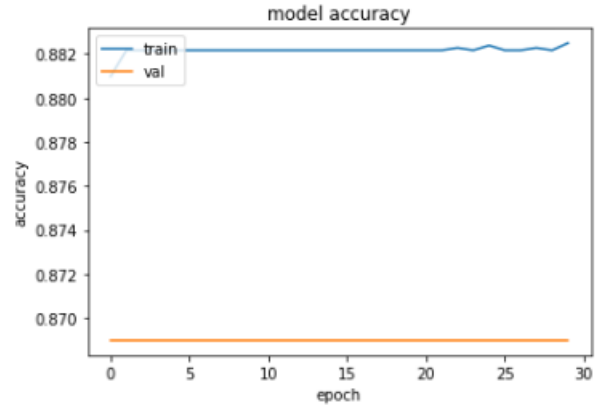
마찬가지로 y값이 모두 1이라고 예측하므로

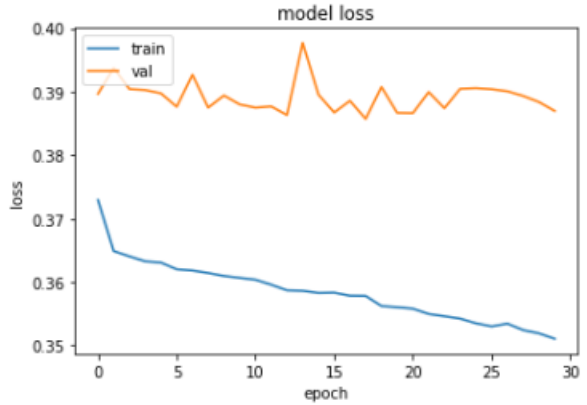
분류 예측에 적절하지 않은 학습니다.

분류 예측 – 모델 학습(DNN)

SOM data 학습 결과

Log Loss Score: 0.360032






```
Y = write_values(model_som, "som")
print("-"*40)
print("The proportion of prediction values of 1 is:")
print(f"{Y.sum()/len(Y)*100:.2f}%")
```

```
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
----------------------------------------
The proportion of prediction values of 1 is:
100.00%
```

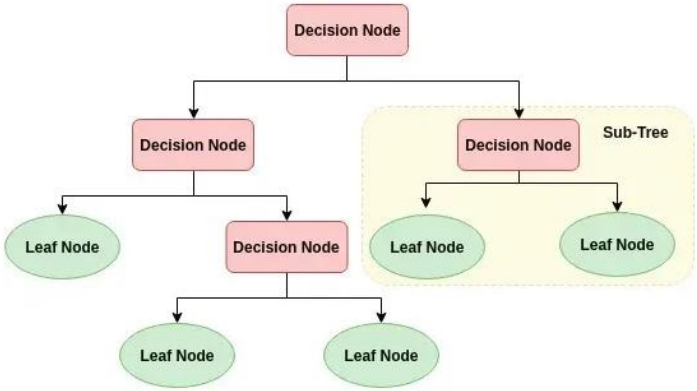
Log Loss Score: 0.360032

마찬가지로 y값이 모두 1이라고 예측하므로

분류 예측에 적절하지 않은 학습니다.

모델 소개

1. Decision Tree – 결정 트리
   1) 분류와 회귀가 모두 가능한 머신러닝 알고리즘
   2) 질문을 던져서 맞고 틀리는 것에 따라 대상을 좁혀나감



규칙 노드(Decision Node) – 규칙 조건
리프 노드(Leaf Node) – 결정된 클래스 값

새로운 규칙 조건마다 서브 트리(Sub Tree)가 생성

모델 소개

장점

데이터의 전처리 (정규화, 결측치, 이상치 등) 를 하지 않아도 됨

수치형과 범주형 변수를 한꺼번에 다룰 수 있음

한계

만약 샘플의 사이즈가 크면 효율성 및 가독성이 떨어짐

과적합으로 알고리즘 성능이 떨어질 수 있음

한 번에 하나의 변수만을 고려하므로 변수간 상호작용을 파악하기가 어려움

최적의 해를 보장하지는 못함

약간의 차이에 따라 (레코드의 개수의 약간의 차이) 트리의 모양이 많이 달라질 수 있음

두 변수가 비슷한 수준의 정보력을 갖는다고 했을 때, 약간의 차이에 의해 다른 변수가 선택되면 이 후의 트리 구성이 크게 달라질 수 있음

분류 예측 – 전처리 적용

1. 전처리를 데이터에 적용
   1) 전처리가 진행된 기본, scaled, Clustering Analysis, SOM의 데이터를 아래와 같이 불러오는 작업을 진행했다.

```
train_data = pd.read_csv('/content/drive/MyDrive/new_data/new_train.csv')
test_data = pd.read_csv('/content/drive/MyDrive/new_data/new_test.csv')
train_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_train_label.csv')
test_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_test_label.csv')
```

```
train_data = pd.read_csv('/content/drive/MyDrive/new_data/new_train_scaled.csv')
test_data = pd.read_csv('/content/drive/MyDrive/new_data/new_test_scaled.csv')
train_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_train_label_scaled.csv')
test_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_test_label_scaled.csv')
```

```
train_data = pd.read_csv('/content/drive/MyDrive/new_data/new_train_scaled+kmeans.csv')
test_data = pd.read_csv('/content/drive/MyDrive/new_data/new_test_scaled+kmeans.csv')
train_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_train_label_scaled+kmeans.csv')
test_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_test_label_scaled+kmeans.csv')
```

```
train_data = pd.read_csv('/content/drive/MyDrive/new_data/new_train_scaled+som.csv')
test_data = pd.read_csv('/content/drive/MyDrive/new_data/new_test_scaled+som.csv')
train_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_train_label_scaled+som.csv')
test_data_label=pd.read_csv('/content/drive/MyDrive/new_data/new_test_label_scaled+som.csv')
```

분류 예측 – 전처리 적용

```
X_train = np.array(pd.DataFrame(train_data, columns=[ 'gender', 'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
  'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID'
]))
y_train = np.array(pd.DataFrame(train_data_label, columns=['credit']))


X_test =np.array( pd.DataFrame(test_data, columns=['gender',  'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
  'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID']))
y_test = np.array(pd.DataFrame(test_data_label, columns=['credit']))
```

```
X_train = np.array(pd.DataFrame(train_data, columns=[ 'gender', 'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
  'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID', 'cluster'
]))
y_train = np.array(pd.DataFrame(train_data_label, columns=['credit']))


X_test =np.array( pd.DataFrame(test_data, columns=['gender',  'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
  'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID', 'cluster']))
y_test = np.array(pd.DataFrame(test_data_label, columns=['credit']))
```

변수 train_data, train_data_label, test_data, test_data_label의 컬럼들의 값들을 데이터프레임 형태로 추출하고 np.array 함수를 이용해 추출한 데이터를 배열형태로 변환한 후 변수 X_train, y_train, X_test, y_test에 저장했다.

Clustering Analysis 와 SOM에서는 cluster 컬럼이 추가되었다.

분류 예측 – 모델 학습

DecisionTreeClassifier 모델을 이용해 학습을 진행하였다.
이 때 이를 평가하는 지표로 f1 score와 specificity score를 이용하였다.

f1 score - 정밀도와 재현율의 조화 평균
specificity score – 0이란 값을 넣었을 때 0 이라는 얻은 결과 비율 (credit값이 0 이 압도적으로 적기 때문에 사용)

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
from imblearn.metrics import specificity_score

import warnings
warnings.filterwarnings(action = 'ignore')



data = pd.DataFrame(X_train)
data['label'] = y_train



model = DecisionTreeClassifier(max_depth=35)


SPLITS = 10
skf = StratifiedKFold(n_splits = SPLITS)
n_iter = 0

features = data.iloc[:,:-1]
label = pd.DataFrame(data['label'])

score_list = []
rscore_list=[]

for train_idx, test_idx in skf.split(features, label):
  n_iter += 1
  print(f'--------------------{n_iter}번째 KFold------------------')
  print(f'학습 값 : {len(train_idx)} / 검증값 : {len(test_idx)}')
  print('학습 레이블 데이터 분포: ₩n', train_idx)
  print('검증 레이블 데이터 분포: ₩n', test_idx)
```

```
  label_train = label.iloc[train_idx]
  label_test = label.iloc[test_idx]

  X_train_1, X_test_1 = features.iloc[train_idx, :], features.iloc[test_idx, :]
  y_train_1, y_test_1 = label.iloc[train_idx,:], label.iloc[test_idx,:]

  model.fit(X_train_1, y_train_1)
  preds = model.predict(X_test_1)
  score = f1_score(y_test_1, preds)
  rscore=specificity_score(y_test_1,preds)


  print(f'{n_iter}번째 단일 f1_score:{score}')

  print(f'{n_iter}번째 단일 specificity_score:{rscore}')


  score_list.append(score)
  rscore_list.append(rscore)



print('=====================================================')
print(f'최종 평균 f1_socre : {sum(score_list)/len(score_list)}')
print(f'최종 평균 specificity_score : {sum(rscore_list)/len(rscore_list)}')
```

분류 예측 – 모델 학습

이 때 tree의 max_depth를 35로 설정한 이유는
1부터 100까지의 다양한 max_depth 값을 설정해본 결과
f1 score는 max_depth 값이 커질수록 점점 작아지고
specificity score는 max_depth 값이 커질수록 점점 커지다가 35에서 수렴하기에
max_depth가 35일때 가장 효율적인 학습이 이루어진다고 판단해 35로 설정하였다.

교차검증

StratifiedKFold를 사용해 교차 검증을 진행
credit 값에서 1의 비율이 0의 비율보다 압도적으로 많기에 불균형 분포도의 레이블 데이터 집합에 사용되는 StratifiedKFold를 사용

분류 예측 – 모델 학습

```
--------------------1번째 KFold--------------------
학습 값 : 11905 / 검증값 : 1323
학습 레이블 데이터 분포:
 [ 1310  1311  1312 ... 13225 13226 13227]
검증 레이블 데이터 분포:
 [   0    1    2 ... 1420 1429 1440]
1번째 단일 f1_score:0.8910284463894966
1번째 단일 specificity_score:0.34782608695652173
--------------------2번째 KFold--------------------
학습 값 : 11905 / 검증값 : 1323
학습 레이블 데이터 분포:
 [    0     1     2 ... 13225 13226 13227]
검증 레이블 데이터 분포:
 [1310 1311 1312 ... 2697 2698 2702]
2번째 단일 f1_score:0.8909169926119079
2번째 단일 specificity_score:0.2919254658385093
--------------------3번째 KFold--------------------
학습 값 : 11905 / 검증값 : 1323
학습 레이블 데이터 분포:
 [    0     1     2 ... 13225 13226 13227]
검증 레이블 데이터 분포:
 [2636 2637 2638 ... 4038 4053 4064]
3번째 단일 f1_score:0.8846815834767642
3번째 단일 specificity_score:0.16770186335403728
--------------------4번째 KFold--------------------
학습 값 : 11905 / 검증값 : 1323
학습 레이블 데이터 분포:
 [    0     1     2 ... 13225 13226 13227]
검증 레이블 데이터 분포:
 [3955 3957 3958 ... 5487 5497 5506]
4번째 단일 f1_score:0.895742832319722
4번째 단일 specificity_score:0.32298136645962733
--------------------5번째 KFold--------------------
학습 값 : 11905 / 검증값 : 1323
```

기본

최종 평균 f1_score : 0.8824541973210968
최종 평균 specificity_score : 0.29546047082278964

scaled

최종 평균 f1_score : 0.8841867711283629
최종 평균 specificity_score : 0.2873859366612989

Clustering analysis

최종 평균 f1_score : 0.8876037540320614
최종 평균 specificity_score : 0.2967180430948547

SOM

최종 평균 f1_score : 0.8834480051752269
최종 평균 specificity_score : 0.30166781688520816

평균 f1 score와 specificity_score가 가장 높은

SOM이 DT에서는 가장 우수한 성능을 보여주었다.

모델 소개

Logistic Regression (로지스틱 회귀 분석)

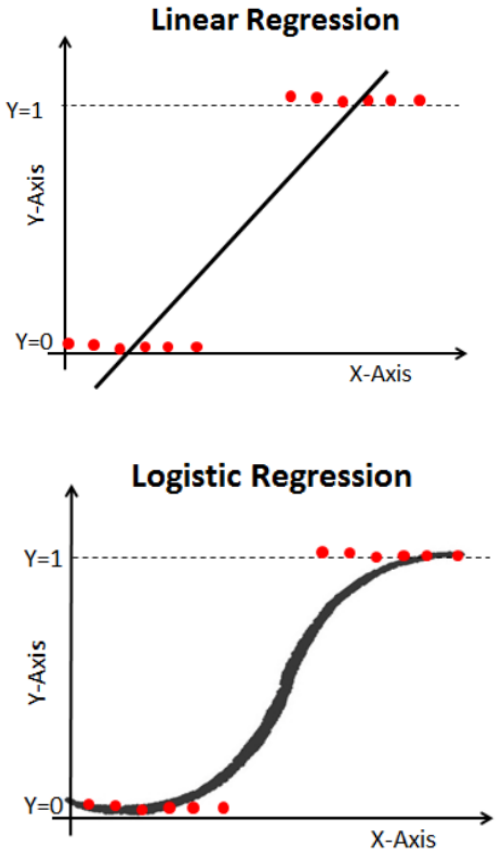
로지스틱 회귀는 이벤트가 발생할 확률을 결정하는 데 사용되는 통계모델이다.

회귀를 사용하여 데이터가 어떤 범주에 속할 확률을 0에서 1 사이의 값으로 예측하고 그 확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해주는 지도 학습 알고리즘이다.

이번 과제의 경우 **credit이 0이냐 1이냐를 결정**해준다.

모델 소개

Linear Regression(선형 회귀) vs Logistic Regression(로지스틱 회귀)



선형회귀란 독립 변수 x를 사용해 종속 변수 y의 움직임을 예측하고 설명한다.

그림에서처럼 1차 방정식의 함수식을 구하는 것이라고 생각하면 될 것이다.

하지만 **실생활에서 모든 원인과 결과는 직선 형태로 표현할 수 없기 때문**에 선형회귀를 보완한 것이 로지스틱 회귀이다.

로지스틱 회귀는 선형회귀의 **직선 대신 S자 곡선**을 이용하여 분류의 정확도를 향상한 방법이다. 여기서 S자 형태의 곡선은 시그모이드 함수(Sigmoid)를 주로 사용한다.

모델 소개

Logistic Regression 의 장단점

장점

-> 대부분의 분류 모형(random forest, decision tree 등) 은 특정 클래스를 예측하는 것과 달리, 로지스틱 회귀는 클래스에 속할 확률을 계산해준다. 따라서 확률 자체에 관심이 있는 분야에서는 LR이 자주 활용된다고 한다.

-> 저차원 자료에서 데이터 샘플 수가 충분하다면 과적합이 덜 일어난다.

단점

-> 선형이기 때문에 이상치에 민감할 수 밖에 없다.

-> 비선형 경계를 가지는 복잡한 패턴을 학습하기가 어렵다.

분류 예측 – 모델 학습

```
import pandas as pd
import numpy as np
import pydotplus
import os
from pprint import pprint
import csv
```

필요한 함수들을 호출하였다

```
[ ]  # 기본
     test_data = pd.read_csv('new_test_scaled.csv')
     test_data_label = pd.read_csv('new_test_label_scaled.csv')
     train_data = pd.read_csv('new_train_scaled.csv')
     train_data_label = pd.read_csv('new_train_label_scaled.csv')

[ ]  # scaled + kmeans
     test_data = pd.read_csv('new_test_scaled+kmeans.csv')
     test_data_label = pd.read_csv('new_test_label_scaled+kmeans.csv')
     train_data = pd.read_csv('new_train_scaled+kmeans.csv')
     train_data_label = pd.read_csv('new_train_label_scaled+kmeans.csv')

[ ]  # scaled + som
     test_data = pd.read_csv('new_test_scaled+som.csv')
     test_data_label = pd.read_csv('new_test_label_scaled+som.csv')
     train_data = pd.read_csv('new_train_scaled+som.csv')
     train_data_label = pd.read_csv('new_train_label_scaled+som.csv')
```

전처리된 데이터 파일들을 불러오는 코드를 작성하였다.

이후 모델을 돌릴 때 각각 실행 하여 비교할 예정이다.

분류 예측 – 모델 학습

```
[ ]  # 기본
     X_train = np.array(pd.DataFrame(train_data, columns=[ 'gender', 'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
       'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID']))
     y_train = np.array(pd.DataFrame(train_data_label, columns=['credit']))



     X_test = np.array(pd.DataFrame(test_data, columns=['gender',  'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
       'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID']))
     y_test = np.array(pd.DataFrame(test_data_label, columns=['credit']))
```

```
[ ]  #  som or kmeans
     X_train = np.array(pd.DataFrame(train_data, columns=[ 'gender', 'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
       'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID', 'cluster']))
     y_train = np.array(pd.DataFrame(train_data_label, columns=['credit']))



     X_test = np.array(pd.DataFrame(test_data, columns=['gender',  'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
       'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID', 'cluster']))
     y_test = np.array(pd.DataFrame(test_data_label, columns=['credit']))
```

Train_data, train_data_label, test_data, test_data_label의 값들을 데이터프레임으로 추출하고 np.array를 이용해 추출한 데이터를 배열로 바꾼 후 X_train, y_train, X_test, y_test에 저장하였다.

som 과 clustering analysi를 적용한 데이터에는 'cluster' column이 추가되었다.

분류 예측 – 모델 학습

```
[5]  n_fold = 5
     seed = 42
     cv = KFold(n_splits=n_fold, shuffle=True, random_state=seed)
```

```
model = LogisticRegression(class_weight='balanced')
data = pd.DataFrame(X_train)
data['label'] = y_train
label = pd.DataFrame(data['label'])
features = data.iloc[:,:-1]
n_iter = 0
score_list = []
logscore_list=[]
for train_idx, test_idx in cv.split(X_train):
  n_iter += 1
  print(f'--------------------{n_iter}번째 KFold------------------')

  print(f'학습 값 : {len(train_idx)} / 검증값 : {len(test_idx)}')
  print('학습 레이블 데이터 분포: ₩n', train_idx)
  print('검증 레이블 데이터 분포: ₩n', test_idx)

  label_train = label.iloc[train_idx]
  label_test = label.iloc[test_idx]

  X_train_1, X_test_1 = features.iloc[train_idx, :], features.iloc[test_idx, :]
  y_train_1, y_test_1 = label.iloc[train_idx,:], label.iloc[test_idx,:]

  model.fit(X_train_1, y_train_1)
  preds = model.predict(X_test_1)
  score = f1_score(y_test_1, preds)
  logscore= log_loss(y_test_1,preds)

  print(f'{n_iter}번째 단일 f1_score:{score}')

  score_list.append(score)
```

Logistic regression 모델을 사용해 진행하였다.
이를 평가하는 지표로 f1 score을 사용하였다.

교차검증은 Kfold를 사용하였다.
집단의 개수는 5개로 설정하였다.

분류 예측 – 모델 학습

```
--------------------1번째 KFold--------------------
학습 값 : 10582 / 검증값 : 2646
학습 레이블 데이터 분포:
 [    1     2     3 ... 13224 13225 13227]
검증 레이블 데이터 분포:
 [    0     8    14 ... 13221 13223 13226]
1번째 단일 f1_score:0.6475252939567953
--------------------2번째 KFold--------------------
학습 값 : 10582 / 검증값 : 2646
학습 레이블 데이터 분포:
 [    0     1     2 ... 13223 13225 13226]
검증 레이블 데이터 분포:
 [    3    10    12 ... 13219 13224 13227]
2번째 단일 f1_score:0.6357836338418862
--------------------3번째 KFold--------------------
학습 값 : 10582 / 검증값 : 2646
학습 레이블 데이터 분포:
 [    0     1     2 ... 13225 13226 13227]
검증 레이블 데이터 분포:
 [   15    26    27 ... 13207 13216 13222]
3번째 단일 f1_score:0.6611126041386723
--------------------4번째 KFold--------------------
학습 값 : 10583 / 검증값 : 2645
학습 레이블 데이터 분포:
 [    0     1     3 ... 13225 13226 13227]
검증 레이블 데이터 분포:
 [    2     6     7 ... 13217 13218 13220]
--------------------5번째 KFold--------------------
학습 값 : 10583 / 검증값 : 2645
학습 레이블 데이터 분포:
 [    0     2     3 ... 13224 13226 13227]
검증 레이블 데이터 분포:
 [    1     4     5 ... 13211 13214 13225]
5번째 단일 f1_score:0.6645230439442658
==================================================
최종평균 f1_score: 0.6508638050441654
```

평균 f1 score은 som으로 전처리한 데이터를 이용한 모델이 LR에서는 가장 우수한 성능을 보였다.

```
최종평균 f1_score: 0.6519163314406737
```

분류 예측 – 최종 예측

F1 score	DNN	DT	LR
Original	0.9352	0.884	0.640
K – Means	0.9352	0.888	0.651
SOM	0.9352	0.884	0.652

최종적으로 f1 score를 비교해 보았을 때 DNN의 f1 score가 가장 높았다. 하지만 예측한 결과를 확인해보니 모든 값을 1로 예측하여 오류 탐지에 적절하지 않은 모델임을 알 수 있었다.

다음으로 DT의 k-means clustering을 포함한 데이터를 활용한 모델이 가장 높은 성능을 보였다. 하지만 예측의 목표는 신용도가 낮은 인물을 가려내는 것이다. 때문에 추가적으로 Specificity score를 활용했고 세 모델 중 som을 포함한 데이터로 학습한 모델이 0.31의 score로 가장 높아 최종 모델을 SOM 모델로 택하였다.

분류 예측 – 최종 예측

```
dt_clf =  DecisionTreeClassifier(max_depth=35)
```

```
dt_clf = dt_clf.fit(X_train, y_train)
```

```
y_test = dt_clf.predict(X_test)
```

```
pd.DataFrame(y_test.round()).to_csv('/content/drive/MyDrive/new_data/test_label.csv', header=["credit"], index=False)
```

채택된 DT-SOM 방법의 코드에 이어서

최종 예측을 위해서 전체 학습 파일을 모델에 넣어주고,
예측해야하는 사람들의 데이터를 입력해
변수 y_test에 넣어주었다.
그리고 이 값을 test_label.csv 파일로 저장했다.

분류 예측 – 최종 예측

```
feature_names = [ 'gender', 'car' ,'reality'  ,'income_total' ,'income_type', 'edu_type'  ,'family_type'  ,'house_type',  'DAYS_BIRTH',
  'DAYS_EMPLOYED',  'work_phone'  ,'email'  ,'occyp_type',  'family_size' ,'begin_month', 'ID','cluster']
```

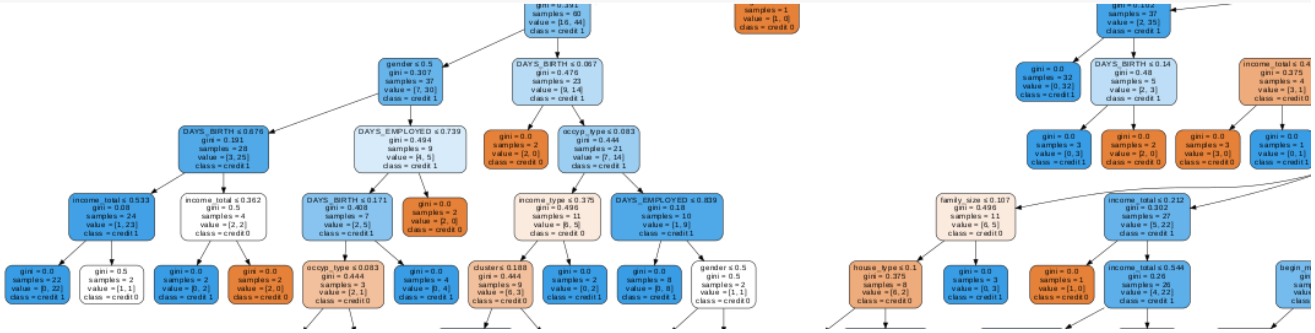
```
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'
```

```
target_name = np.array(['credit 0', 'credit 1'])
```

```
dt_dot_data = tree.export_graphviz(dt_clf, out_file = None,
                                   feature_names = feature_names,
                                   class_names = target_name,
                                   filled = True, rounded = True,
                                   special_characters = True)
```

```
dt_graph = pydotplus.graph_from_dot_data(dt_dot_data)
```

```
Image(dt_graph.create_png())
```



위와 같이 결정 트리를 시각화하는 코드도 작성해 주었다.