금융 빅데이터 분석을 통한 신용카드 연체 리스크 관리

최선종

목차

01. 프로젝트 개요

02. 데이터 정규화 개념

03. 데이터 전처리 및 EDA

04. 머신러닝 구현

05. 가설 검정

06. 인사이트 도출

01. 프로젝트 개요

1) 프로젝트

프로젝트명	파이썬, R프로그래밍 활용한 신용등급 예측 알고리즘 구현
기간	2021.04.23. ~ 05.21.(4주간)
개발환경	Windows 10
사용도구	Pycharm, Jupyter, R studio
사용기술	Python, R programming, Tableau
주요 라이브러리	Numpy, pandas, seaborn, matplotlib
주요 모델링	Classification, xgboost, KNN, SVM, D-tree, R-forest
주요 전처리	결측치 제거, 변수타입 및 형태 변환, 스케일링, 구간화

2) 목 적

목 적	신용카드 사용자의 데이터를 활용하여 대금 연체 정도를 예측하는 최적의 모델 및 파라미터를 찾아 이를 적용하여 신용카드사의 재무건전성 회복
데이터출처	DACON, 신용카드 사용자 연체 예측 AI 경진대회
데이터명	sample_submission.csv, train.csv, test.csv

01. 프로젝트 개요


git
GitHub
Pandas
dmlc
XGBoost
tableau
SOFTWARE
ANACONDA.
jupyter
scikit
learn
seaborn
NumPy
R
Studio
Notion

01. 프로젝트 개요

프로젝트 기간 (WBS)

사전 준비

04.23~04.26

팀 구성 및 프로젝트 아이디어 공유

기획

04.26~04.30

프로젝트 주제 선정
문제 탐색, 정의 및 데이터 선정

데이터 분석

05.01~05.10

데이터 전처리 및 EDA / 시각화 (탐색적 데이터 분석)

머신러닝(ML)

05.10~05.19

머신러닝 구현

가설 검정

05.19~05.20

통계적 추론을 통한 가설 검정

인사이트 도출

05.20~05.21

프로젝트 최종 인사이트 도출

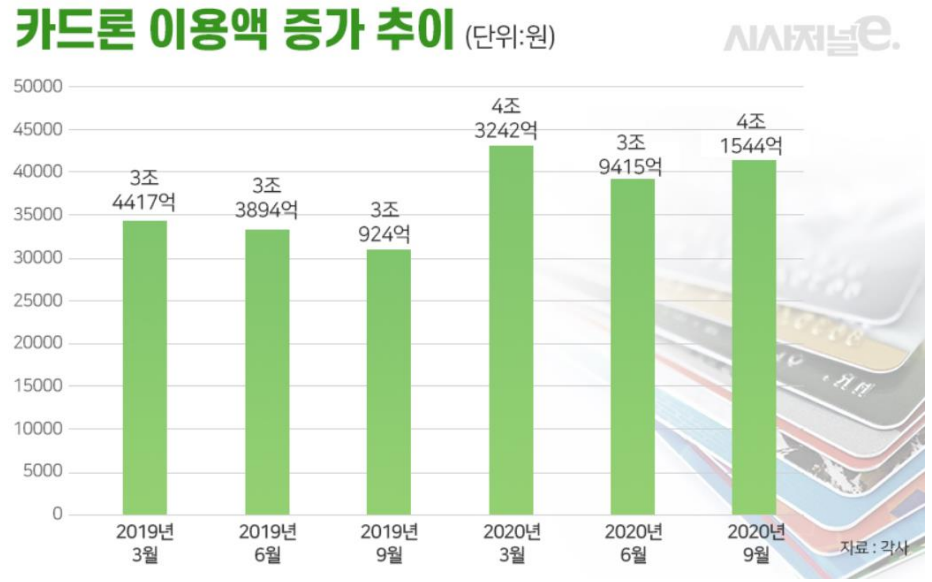
마무리

05.20~05.21

PPT 작성 및 최종 마무리

01. 프로젝트 개요

출처: 서울경제(2019.12.02) – 카드빚 못갚는 서민…연체 54개월來 최대



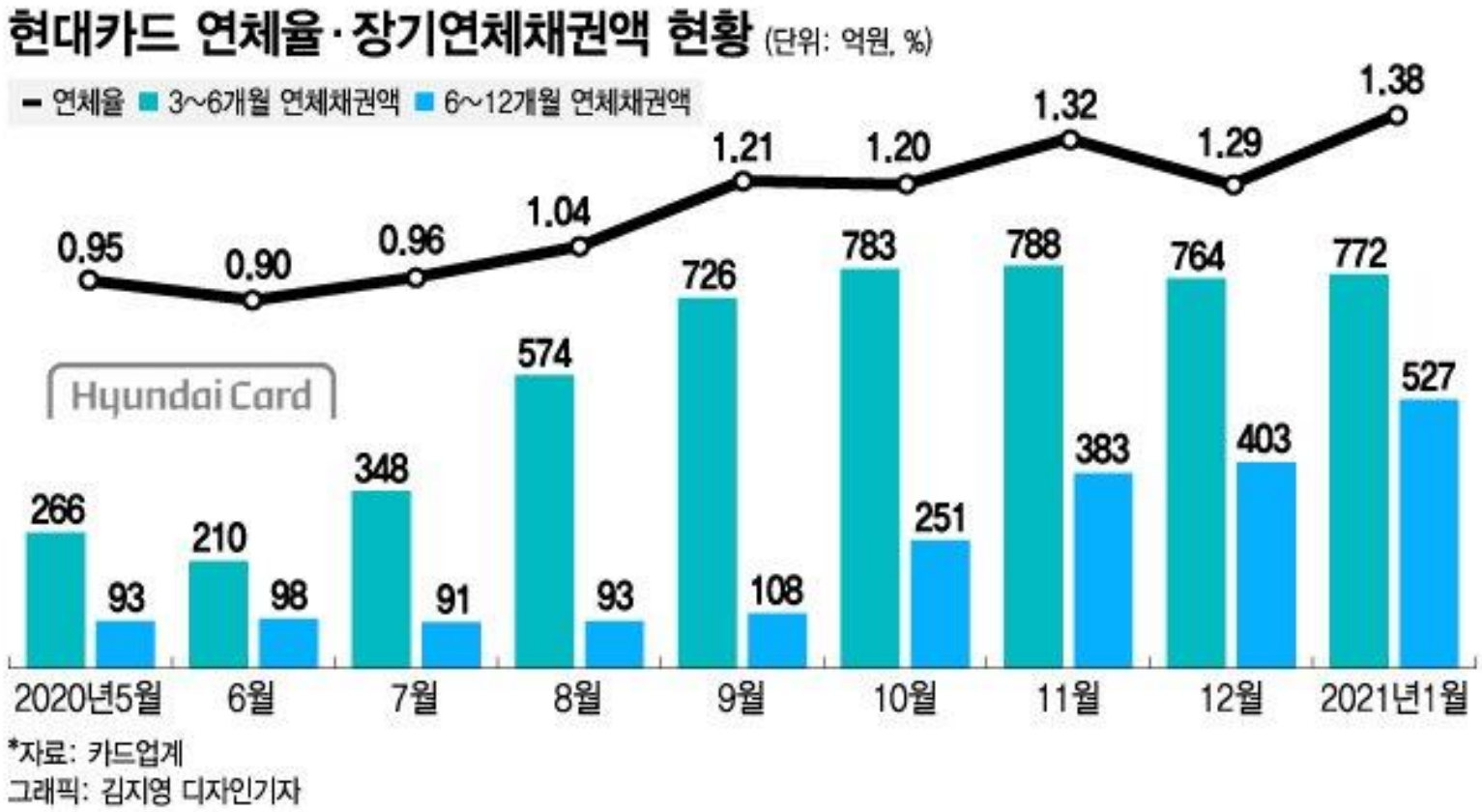
출처: 시사저널e(2020.10.28) – 늘어나는 카드론 불어나는 연체 리스크

○ 정책
- 20년 9월말 기준 가계대출잔액 1,585조원
- 금융기관의 부실채권 회수 과정에서 **채무자의 권리 보호**의 관한 문제 대두
- 21년「소비자신용에 관한 법률」개정, 채무자로부터 채권 회수가 어려워 짐

○ 개인
- 경기침체 장기화 및 부동산, 주식투자 등 **대출 열풍**
- 신용등급에 따라 부여된 한도 내 카드 회원을 대상으로 대출을 해주는 카드론 장기대출 상품 많음
- 토스, 뱅크샐러드 등 금융플랫폼사들의 현금서비스 이용액 감소 (이윤이 낮은 **카드론 서비스 이용 증가**)

○ 기업
- 20년 기준, 카드론 연체금액 5,400억원(매년 증가)
- 신용대출 금리(연 2%) 대비 카드론 금리(3.9%) 1% 차이
  → **카드론 사용 부담 ↓**
- 금융소비자보호법으로 인한, 시중 은행의 비대면 대출 중단
  → **카드론 사용 급증**
  (카드론 중도 상환 수수료, 취급 수수료, 담보대출 없어 접근성이 높음)
- 수신기능이 없는 카드사는 연체관리가 되지 않을 경우 여전채를 통해 조달 (카드회사 부채 비율 증가)
- **연체 관리가 필요(건전성 리스크 관리)**

01. 프로젝트 개요



출처: 머니투데이(2021.03.05) – 현대카드 연체율, 장기연체채권액 현황

이러한 배경에 따라 카드론 회사들은 채권자들의 연체 관리를 통하여 재무건전성을 확보해 리스크를 줄여야 함

→ "개인정보를 활용, 신용등급을 예측하여 연체자를 사전에 관리할 수 있는가?" 관심을 갖게 됨

02. 제1정규화 – 비정규형

학생수강성적

학생번호	학생이름	주소	학과	학과사무실	강좌이름	강의실	성적
501	박지성	영국 맨체스타	컴퓨터과	공학관101	데이터베이스	공학관 110	3.5
401	김연아	대한민국 서울	체육학과	체육관101	데이터베이스	공학관 110	4.0
402	장미란	대한민국 강원도	체육학과	체육관101	스포츠경영학	체육관 103	3.5
502	추신수	미국 클리블랜드	컴퓨터과	공학관101	자료구조	공학관 111	4.0
501	박지성	영국 맨체스타	컴퓨터과	공학관101	자료구조	공학관 111	3.5

학생번호	학생이름	주소	학과	학과사무실
501	박지성	영국 맨체스타	컴퓨터과	공학관101
401	김연아	대한민국 서울	체육학과	체육관101
402	장미란	대한민국 강원도	체육학과	체육관101
502	추신수	미국 클리블랜드	컴퓨터과	공학관101

학생번호	강좌이름	강의실	성적
501	데이터베이스	공학관 110	3.5
401	데이터베이스	공학관 110	4.0
402	스포츠경영학	체육관 103	3.5
502	자료구조	공학관 111	4.0
501	자료구조	공학관 111	3.5

제1정규화는 각각의 도메인에 하나의 원자 값만 가져야 한다.

이름, 주소, 학과, 사무실을 종속 값으로 가지기 때문에 학생번호는 개체 무결성을 가져 pk로 설정된다.

그러나 강좌이름, 강의실, 성적은 강좌 이름에 종속 되지 않기에 학생 테이블과 성적 테이블 2개로 분리된다.

또한 성적은 학생번호, 강좌이름에 종속 되므로 해당 칼럼이 개체 무결성을 가져 복합 식별자로 설정된다.

02. 제2정규화 – 부분함수 종속 제거

학생번호	강좌이름	강의실	성적
501	데이터베이스	공학관 110	3.5
401	데이터베이스	공학관 110	4.0
402	스포츠경영학	체육관 103	3.5
502	자료구조	공학관 111	4.0
501	자료구조	공학관 111	3.5

강좌이름	강의실
데이터베이스	공학관 110
스포츠경영학	체육관 103
자료구조	공학관 111

학생번호	강좌이름	성적
501	데이터베이스	3.5
401	데이터베이스	4.0
402	스포츠경영학	3.5
502	자료구조	4.0
501	자료구조	3.5

제1정규화에서 복합 식별자가 발생하였으므로 제2정규화 작업이 필요.

복합키인 학생번호, 강좌이름에 성적은 종속되지만 강의실은 학생번호와 관계가 없고 강좌이름과 관계가 있다. 또한 학생번호, 강좌이름은 분리된 테이블로부터 참조 무결성을 가져 fk로 설정된다.

따라서 서로 종속 관계인 테이블을 분류하여 제2정규화 작업을 마친다.

02. 제3정규화 – 이행적 함수 종속 제거

학생번호	학생이름	주소	학과	학과사무실
501	박지성	영국 맨체스타	컴퓨터과	공학관101
401	김연아	대한민국 서울	체육학과	체육관101
402	장미란	대한민국 강원도	체육학과	체육관101
502	추신수	미국 클리블랜드	컴퓨터과	공학관101

학생번호	학생이름	주소	학과
501	박지성	영국 맨체스타	컴퓨터과
401	김연아	대한민국 서울	체육학과
402	장미란	대한민국 강원도	체육학과
502	추신수	미국 클리블랜드	컴퓨터과

학과	학과사무실
컴퓨터과	공학관101
체육학과	체육관101

강좌이름	강의실
데이터베이스	공학관 110
스포츠경영학	체육관 103
자료구조	공학관 111

학생번호	강좌이름	성적
501	데이터베이스	3.5
401	데이터베이스	4.0
402	스포츠경영학	3.5
502	자료구조	4.0
501	자료구조	3.5

모든 non key 컬럼값이 key 컬럼에 종속되어야 한다.

학과 – 학과사무실, 강좌이름 – 강의실, 학생번호+강좌이름(복합 식별자) – 성적으로

종속 관계가 성립하므로 위와 같이 테이블을 분류해준다.

02. SQL문 활용 – view & join

```
create view sinfo as
select s.s_no, s.s_name, s.major, p.m_office
from shop.student s
join shop.part p
on s.major = p.major
```

sinfo

Enter a SQL expression to filter results (use Ctrl+Space)

	s_no	s_name	major	m_office
1	401	김연아	체육학과	체육관101
2	402	장미란	체육학과	체육관101
3	501	박지성	컴퓨터과	공학관101
4	502	추신수	컴퓨터과	공학관101

Student 테이블과 part 테이블에서 동일한 컬럼인 전공을 기준으로 s_no, s_name, s.major, p.m_office 컬럼을 하나의 테이블에 view 사용하여 보여준다.

*<localhost> Script-19 | part | class | score | shop | student | s_info

```
select *
from shop.student
join shop.score
on shop.student.s_no = shop.score.s_no
```

student(+) 1

select * from shop.student left join shop.score on sh

Enter a SQL expression to filter results (use Ctrl+Spa

	s_no	s_name	address	major	s_no	c_name	score
1	401	김연아	대한민국 서울	체육학과	401	데이터베이스	4
2	402	장미란	대한민국 강원도	체육학과	402	스포츠경영학	3.5
3	501	박지성	영국 맨체스타	컴퓨터과	501	데이터베이스	3.5
4	501	박지성	영국 맨체스타	컴퓨터과	501	자료구조	3.5
5	502	추신수	미국 클리블랜드	컴퓨터과	502	자료구조	4

Student 테이블과 score 테이블을 s_no(학생번호)가 같은 것을 기준으로 join한다.

03. 데이터 전처리 및 EDA

index	gender	car	real...	child_num	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH
DAYS_EMPLOYED	FLAG_MOBIL	work_phone	phone	email	occyp_type	family_size	begin_month	credit		

- **Index**
- **gender:** 성별
- **car:** 차량 소유 여부
- **reality:** 부동산 소유 여부
- **child_num:** 자녀 수
- **income_total:** 연간 소득
- **income_type:** 소득 분류
  ['Commercial associate', 'Working', 'State servant',
  'Pensioner', 'Student']
- **edu_type:** 교육 수준
  ['Higher education' ,'Secondary / secondary special',
  'Incomplete higher', 'Lower secondary', 'Academic degree']
- **family_type:** 결혼 여부
  ['Married', 'Civil marriage', 'Separated',
  'Single / not married', 'Widow']
- **house_type:** 생활 방식
  ['Municipal apartment', 'House / apartment', 'With parents',
  'Co-op apartment', 'Rented apartment', 'Office apartment']
- **DAYS_BIRTH:** 출생일
  데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전에 태어났음을 의미
- **DAYS_EMPLOYED:** 업무 시작일
  데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 하루 전부터 일을 시작함을 의미, 양수 값은 고용되지 않은 상태를 의미함
- **FLAG_MOBIL:** 핸드폰 소유 여부
- **work_phone:** 업무용 전화 소유 여부
- **phone:** 전화 소유 여부
- **email:** 이메일 소유 여부
- **occyp_type:** 직업 유형
- **family_size:** 가족 규모
- **begin_month:** 신용카드 발급 월
  데이터 수집 당시 (0)부터 역으로 셈, 즉, -1은 데이터 수집일 한 달 전에 신용카드를 발급함을 의미
- **credit: 사용자의 신용카드 대금 연체를 기준으로 한 신용도**
  => 낮을 수록 높은 신용의 신용카드 사용자를 의미함

03. 데이터 전처리 및 EDA

	index	child_num	income_total	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL
count	26457.000000	26457.000000	2.645700e+04	26457.000000	26457.000000	26457.0
mean	13228.000000	0.428658	1.873065e+05	-15958.053899	59068.750728	1.0
std	7637.622372	0.747326	1.018784e+05	4201.589022	137475.427503	0.0
min	0.000000	0.000000	2.700000e+04	-25152.000000	-15713.000000	1.0
25%	6614.000000	0.000000	1.215000e+05	-19431.000000	-3153.000000	1.0
50%	13228.000000	0.000000	1.575000e+05	-15547.000000	-1539.000000	1.0
75%	19842.000000	1.000000	2.250000e+05	-12446.000000	-407.000000	1.0
max	26456.000000	19.000000	1.575000e+06	-7705.000000	365243.000000	1.0

work_phone	phone	email	family_size	begin_month	credit
26457.000000	26457.000000	26457.000000	26457.000000	26457.000000	26457.000000
0.224742	0.294251	0.091280	2.196848	-26.123294	1.519560
0.417420	0.455714	0.288013	0.916717	16.559550	0.702283
0.000000	0.000000	0.000000	1.000000	-60.000000	0.000000
0.000000	0.000000	0.000000	2.000000	-39.000000	1.000000
0.000000	0.000000	0.000000	2.000000	-24.000000	2.000000
0.000000	1.000000	0.000000	3.000000	-12.000000	2.000000
1.000000	1.000000	1.000000	20.000000	0.000000	2.000000

```
# 데이터 타입 및 결측치 확인
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26457 entries, 0 to 26456
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   index           26457 non-null  int64
 1   gender          26457 non-null  object
 2   car             26457 non-null  object
 3   reality         26457 non-null  object
 4   child_num       26457 non-null  int64
 5   income_total    26457 non-null  float64
 6   income_type     26457 non-null  object
 7   edu_type        26457 non-null  object
 8   family_type     26457 non-null  object
 9   house_type      26457 non-null  object
 10  DAYS_BIRTH      26457 non-null  int64
 11  DAYS_EMPLOYED   26457 non-null  int64
 12  FLAG_MOBIL      26457 non-null  int64
 13  work_phone      26457 non-null  int64
 14  phone           26457 non-null  int64
 15  email           26457 non-null  int64
 16  occyp_type      18286 non-null  object
 17  family_size     26457 non-null  int64
 18  begin_month     26457 non-null  int64
 19  credit          26457 non-null  int64
dtypes: float64(1), int64(11), object(8)
memory usage: 4.0+ MB
```

data.info 코드 사용하여 데이터 타입 및 결측치를 확인

data.describe 사용하여 데이터들의 요약 통계량을 확인하여 각 변수당 데이터 개수, 평균, 최소, 최대값 등 데이터의 기본적인 정보확인

03. 데이터 전처리 및 EDA

	index	gender	car	reality	child_num	income_total	income_type	edu_type	family_type	house_type
0	0	F	N	N	0	202500.0	Commercial associate	Higher education	Married	Municipal apartment
1	1	F	N	Y	1	247500.0	Commercial associate	Secondary / secondary special	Civil marriage	House / apartment
2	2	M	Y	Y	0	450000.0	Working	Higher education	Married	House / apartment
3	3	F	N	Y	0	202500.0	Commercial associate	Secondary / secondary special	Married	House / apartment
4	4	F	Y	Y	0	157500.0	State servant	Higher education	Married	House / apartment

DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	work_phone	phone	email	occyp_type	family_size	begin_month	credit
-13899	-4709	1	0	0	0	NaN	2	-6	1
-11380	-1540	1	0	0	1	Laborers	3	-5	1
-19087	-4434	1	0	1	0	Managers	2	-22	2
-15088	-2092	1	0	1	0	Sales staff	2	-37	0
-15037	-2105	1	0	0	0	Managers	2	-26	2

① 각 변수의 명목형 값을 이진법 0, 1로 분류

② income_total, int 타입의 값 크기 조정 필요
기타 명목형 변수 형태 값을 이산형 변수로 변환

③ int 타입의 음수 값들을 양수로 변환

④ 결측치 처리 및 명목형 변수 형태의 값을
이산형 변수로 변환

⑤ int 타입의 음수 값들을 양수로 변환

데이터의 전처리 진행 전, 데이터의 기본적인 형태를 파악하고 각 변수들이 갖는 value 값들에 대한 이해를 우선적으로 선행

* 데이터분석시 컴퓨터가 이해할 수 있는 데이터로의 변환해주는 작업이 필요

03. 데이터 전처리 및 EDA

```
#결측치 제거
train.isnull().sum()
```

```
index              0
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      8171
family_size        0
begin_month        0
credit             0
dtype: int64
```

①

```
train=train.drop('occyp_type', axis=1)
test=test.drop('occyp_type', axis=1)
```

```
#이진분류
train['gender'] = train['gender'].replace(['F','M'],[0,1])
test['gender'] = test['gender'].replace(['F','M'],[0,1])
print('성별 :')
print(train['gender'].value_counts())
print('--------------')

print('차량 소유여부 : ')
train['car'] = train['car'].replace(['N','Y'],[0,1])
test['car'] = test['car'].replace(['N','Y'],[0,1])
print(train['car'].value_counts())
print('--------------')

print('자가 소유여부: ')
train['reality'] = train['reality'].replace(['N','Y'],[0,1])
test['reality'] = test['reality'].replace(['N','Y'],[0,1])
print(train['reality'].value_counts())
print('--------------')

print('전화기 소유여부: ')
print(train['phone'].value_counts())
print('--------------')

print('이메일 소유여부: ')
print(train['email'].value_counts())
print('--------------')

print('업무용 핸드폰 소유여부: ')
print(train['work_phone'].value_counts())
print('--------------')
```

②

```
성별 :
0    17697
1     8760
Name: gender, dtype: int64
--------------
차량 소유여부 :
0    16410
1    10047
Name: car, dtype: int64
--------------
자가 소유여부:
1    17830
0     8627
Name: reality, dtype: int64
--------------
전화기 소유여부:
0    18672
1     7785
Name: phone, dtype: int64
--------------
이메일 소유여부:
0    24042
1     2415
Name: email, dtype: int64
--------------
업무용 핸드폰 소유여부:
0    20511
1     5946
Name: work_phone, dtype: int64
```

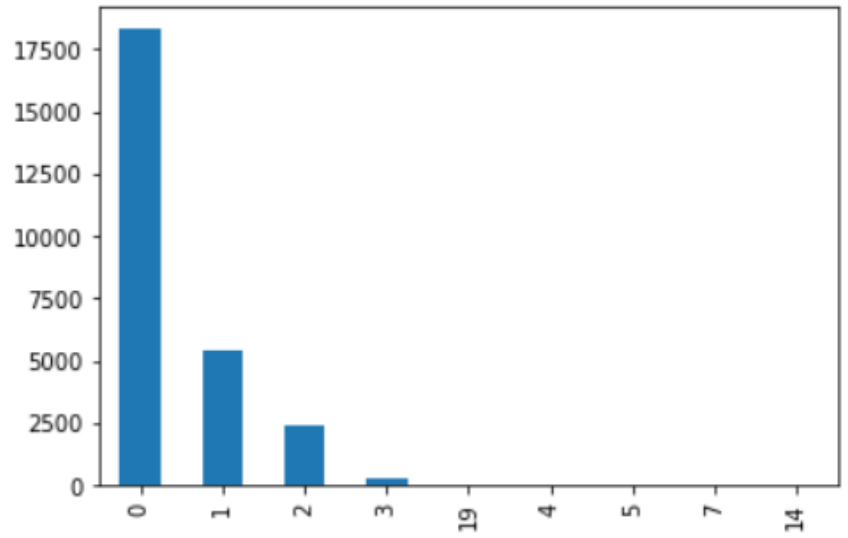
① 결측치 및 Secondary / secondary special, Higher education 카테고리 대비 기타 변수 간 표본차가 크기 때문에 변수 삭제

② 명목형으로 되어있는 변수의 경우 컴퓨터가 이해하기 어려움
컴퓨터가 이해할 수 있는 Binary 값인 0, 1로 변경

03. 데이터 전처리 및 EDA

```
train['child_num'].value_counts(sort=False).plot.bar()
```

```
<AxesSubplot:>
```




```
train.loc[train['child_num'] >= 2,'child_num']=2
test.loc[test['child_num']>=2, 'child_num']=2
```

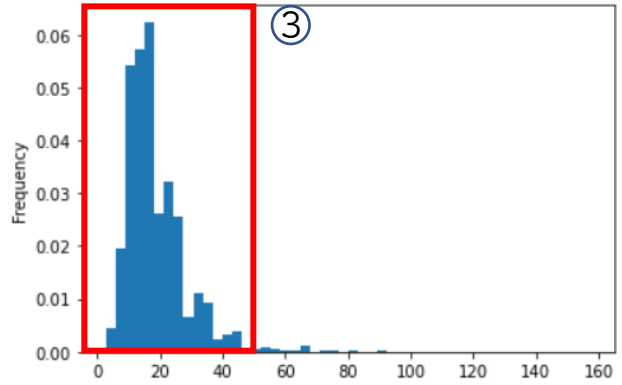
① child_num 값이 3 이상 존재하지 않기 때문에 0, 1, 2 값만 가지도록 데이터 변환

```
train['income_total'] = train['income_total'].astype(object)
train['income_total'] = train['income_total']/10000
test['income_total'] = test['income_total']/10000

print(train['income_total'].value_counts(bins=10,sort=False))
train['income_total'].plot(kind='hist',bins=50,density=True)
```

②

```
(2.544, 18.18]      16212
(18.18, 33.66]       8330
(33.66, 49.14]       1530
(49.14, 64.62]        206
(64.62, 80.1]         121
(80.1, 95.58]          44
(95.58, 111.06]         3
(111.06, 126.54]        2
(126.54, 142.02]        4
(142.02, 157.5]         5
Name: income_total, dtype: int64
```

```
<AxesSubplot:ylabel='Frequency'>
```



② 최대 6자리 이상 숫자로 구성된 Income_total 변수 값의 크기 및 타입 조정

③ 소득이 특정 구간에 분포하고 있고 표본차가 크기 때문에 구간을 좁힐 필요가 있음

```
count, bin_dividers =np.histogram(train['income_total'], bins=7)
bin_names=['소득'+str(i) for i in range(7) ]
train['income_total']=pd.cut(x=train['income_total'], bins=bin_dividers, labels=bin_names, include_lowest=True)
test['income_total']=pd.cut(x=test['income_total'], bins=bin_dividers, labels=bin_names, include_lowest=True)
```

④

④ train['income_total'] 기준으로 7개 구간으로 나눔 income_total 변수 값을 for문 사용하여 소득 + index로 변경 후 내림차순으로 정렬

03. 데이터 전처리 및 EDA

```
from sklearn import preprocessing
label_encoder=preprocessing.LabelEncoder()
train['income_type']=label_encoder.fit_transform(train['income_type'])
test['income_type']=label_encoder.transform(test['income_type'])

train['edu_type']=label_encoder.fit_transform(train['edu_type'])
test['edu_type']=label_encoder.transform(test['edu_type'])

train['family_type']=label_encoder.fit_transform(train['family_type'])
test['family_type']=label_encoder.transform(test['family_type'])

train['house_type']=label_encoder.fit_transform(train['house_type'])
test['house_type']=label_encoder.transform(test['house_type'])

train['income_total']=label_encoder.fit_transform(train['income_total'])
test['income_total']=label_encoder.fit_transform(test['income_total'])
```

① 스케일링 및 변수 변환을 위해 sklearn 라이브러리의 preprocessing 패키지 사용
(데이터를 정규분포로 만들어 평균을 0, 분산을 1로 스케일링 하여 변수간 차이를 확인하기 위함)

```
def make_bin(variable, n):
    train[variable]=-train[variable]
    test[variable]=-test[variable]
    count, bin_dividers =np.histogram(train[variable], bins=n) #train의 구간화를 적용
    bin_names=[str(i) for i in range(n)]
    train[variable]=pd.cut(x=train[variable], bins=bin_dividers, labels=bin_names, include_lowest=True)
    test[variable]=pd.cut(x=test[variable], bins=bin_dividers, labels=bin_names, include_lowest=True)
    test[variable].fillna(str(0), inplace=True) #test에는 없는 것을 임의의 값으로 채움

    train[variable]=label_encoder.fit_transform(train[variable])
    test[variable]=label_encoder.transform(test[variable])
```

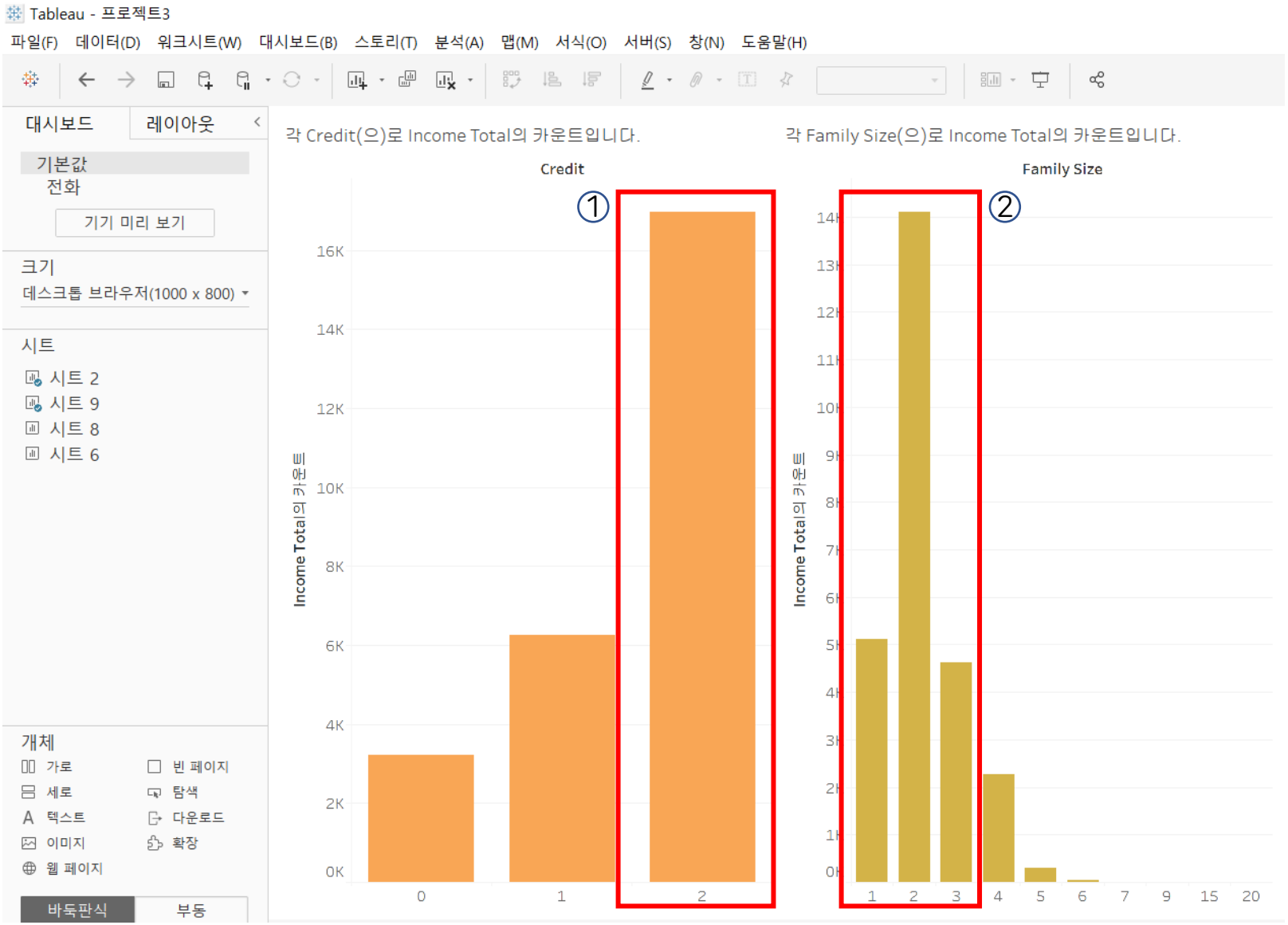
② train, test 데이터의 변수에 음수 값들을 양수로 변환

③ Numpy 라이브러리의 histogram 패키지를 사용, train 데이터를 n개 만큼 구간화를 적용

④ test 데이터 변수의 모든 결측치 값을 0으로 변환

⑤ 변경된 데이터 값을 train, test 데이터 변수에 저장

03. 데이터 전처리 및 EDA



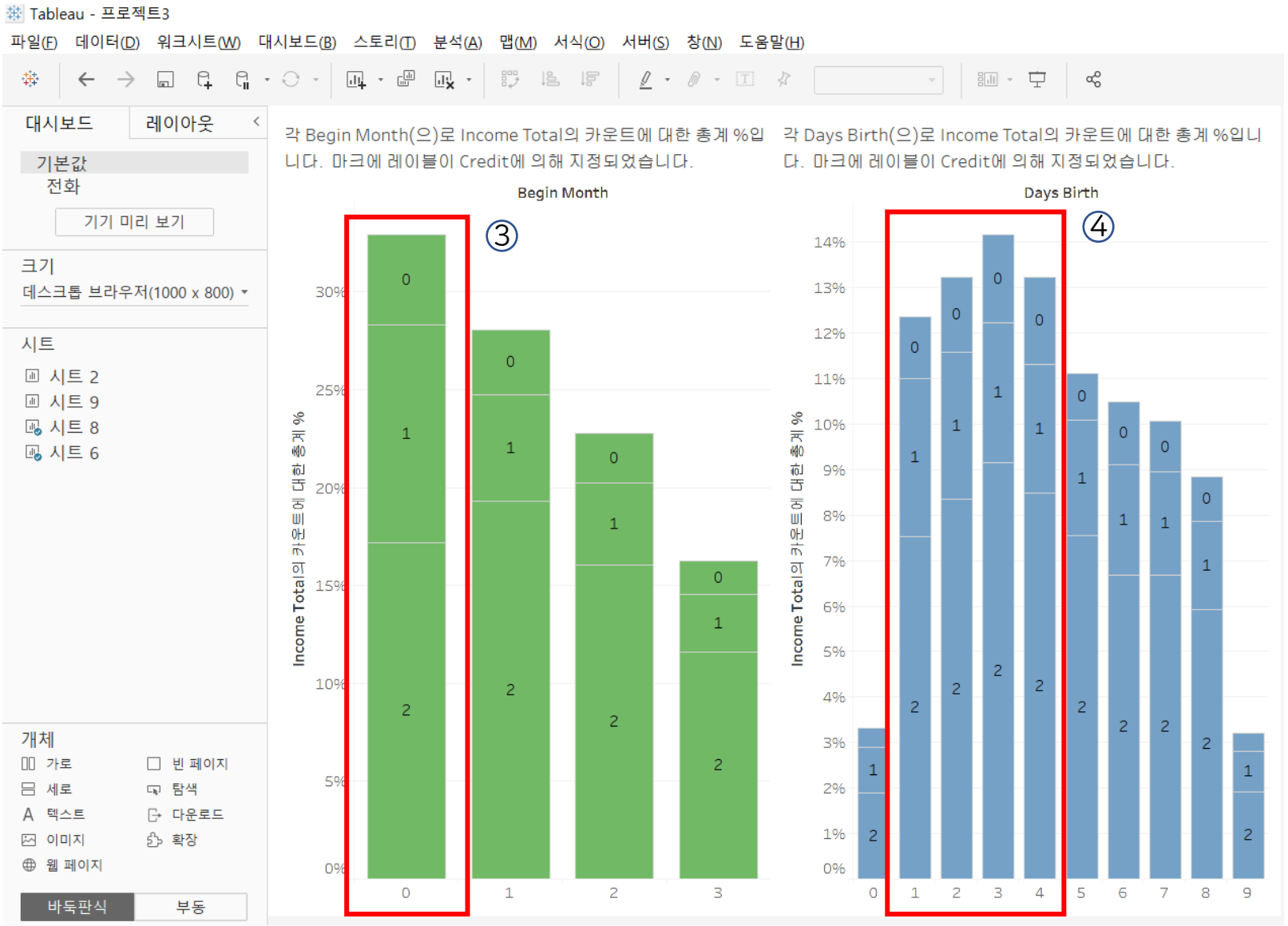
① 신용등급(Credit)기준 총소득(Income Total)자 절반 이상이 가장 낮은 등급인 2등급에 속하고 있음

▷ 낮은 신용등급의 카드 사용자에 대한 관리 필요함을 파악

② 가족 수(Family Size)기준 총 소득 비교결과 가족 수가 2명인 카테고리에 대부분의 소득자가 분포되어 있음

▷ 가족 구성원이 3인 이하인 카드 사용자에 대한 관리 필요

03. 데이터 전처리 및 EDA



③ 카드사용구간(Begin Month)중, 사용 기간이 길수록 사용자가 줄어 듦
카드사용 0 구간에서 신용등급 1급이 가장 많이 분포

▷ 카드사용기간 0 구간(1년 이하)의 사용자에 대한 연체관리와 구간별 신용등급 2등급 관리 필요함

④ 연령구간(Days Birth)중, 3 구간에 가장 많은 소득자가 분포
이후 연령이 높아질수록 연령대비 총 소득자의 숫자는 하락

▷ 3 구간(30 – 40대) 사용자 및 1 – 4 구간의 전반적인 연체관리가 필요

03. 데이터 전처리 및 EDA

```
# 신용도(Credit) 상관관계가 적은 컬럼들 삭제하여 상관관계 분석
train2 = train.drop(['index','gender','car','reality','FLAG_MOBIL','phone','email','work_phone'], axis=1)
train2
```

```
train2.corr()  # 각 항목들의 상관도를 구해줌(-1 ~ 1사이)
```

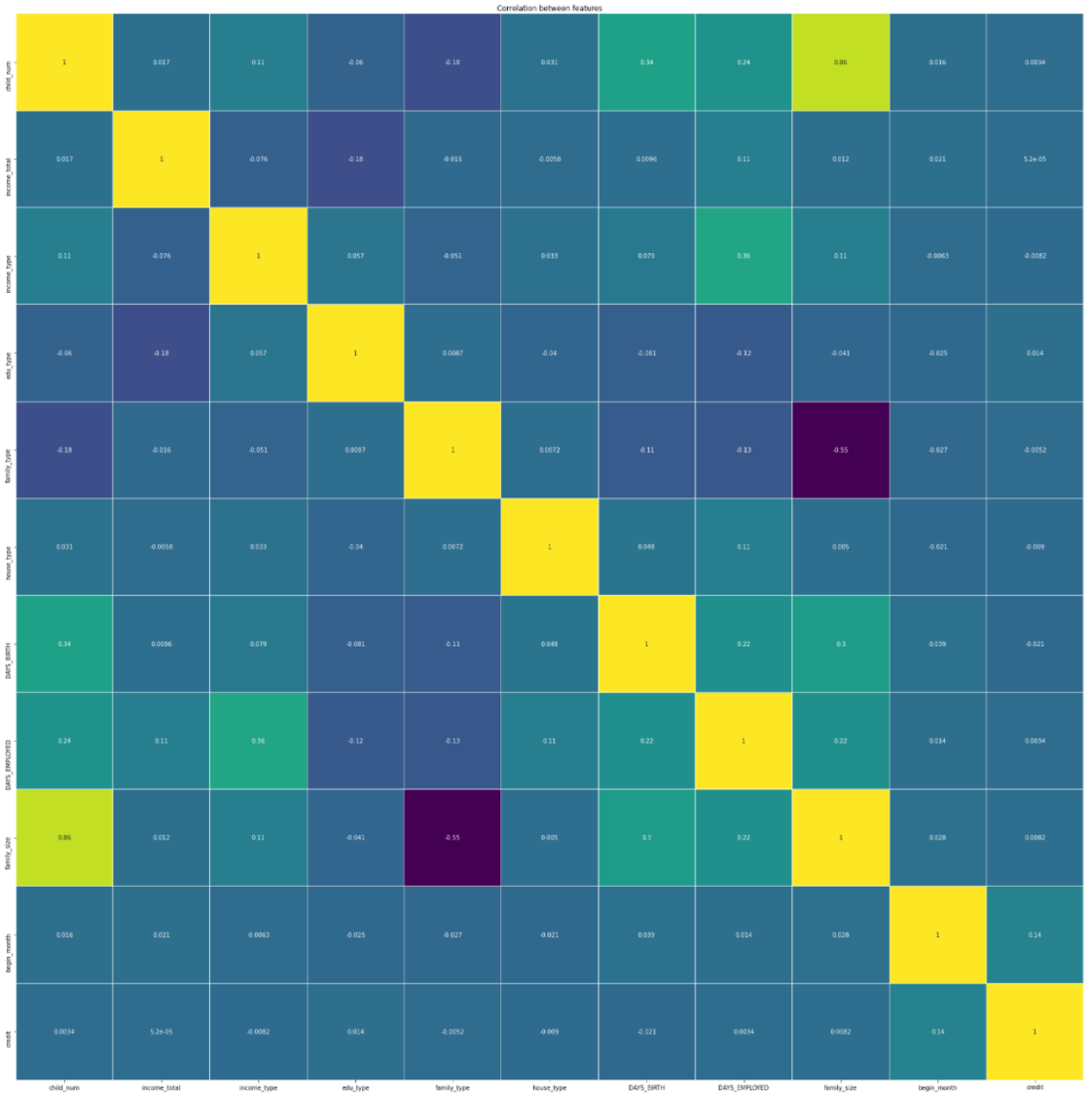
	child_num	income_total	income_type	edu_type	family_type	house_type	DAYS_BIRTH	DAYS_EMPLOYED	family_size	begin_month	credit
child_num	1.000000	0.016577	0.108831	-0.060029	-0.175368	0.030583	0.339653	0.244999	0.860830	0.015749	0.003441
income_total	0.016577	1.000000	-0.075698	-0.179422	-0.015504	-0.005790	0.009600	0.107209	0.012105	0.021353	0.000052
income_type	0.108831	-0.075698	1.000000	0.056688	-0.050790	0.033013	0.079155	0.360725	0.107698	-0.006318	-0.008163
edu_type	-0.060029	-0.179422	0.056688	1.000000	0.008717	-0.039668	-0.080656	-0.121542	-0.041345	-0.025240	0.013780
family_type	-0.175368	-0.015504	-0.050790	0.008717	1.000000	0.007189	-0.110266	-0.125216	-0.545149	-0.027340	-0.005230
house_type	0.030583	-0.005790	0.033013	-0.039668	0.007189	1.000000	0.047538	0.107004	0.005010	-0.021428	-0.009023
DAYS_BIRTH	0.339653	0.009600	0.079155	-0.080656	-0.110266	0.047538	1.000000	0.221323	0.303256	0.038673	-0.021176
S_EMPLOYED	0.244999	0.107209	0.360725	-0.121542	-0.125216	0.107004	0.221323	1.000000	0.223751	0.014264	0.003429
family_size	0.860830	0.012105	0.107698	-0.041345	-0.545149	0.005010	0.303256	0.223751	1.000000	0.027541	0.008227
begin_month	0.015749	0.021353	-0.006318	-0.025240	-0.027340	-0.021428	0.038673	0.014264	0.027541	1.000000	0.142109
credit	0.003441	0.000052	-0.008163	0.013780	-0.005230	-0.009023	-0.021176	0.003429	0.008227	0.142109	1.000000

연체관리를 위해서는 어떠한 개인정보가 신용등급에 영향을 미치는지 확인하기 위해 corr 함수 사용

신용카드 사용자의 신용등급(Credit)에 영향을 주는 요소를 파악하기 위해 상관관계가 적은 변수 삭제 후 train2 변수에 저장

상관관계 분석 결과 눈에 띄는 결과는 보이지 않았지만 신용등급 - 카드사용기간 간의 약한 상관관계가 있는 것을 확인

```
# 변수간 상관관계 확인
plt.figure(figsize=(40, 40))  # figure size 생성
sns.heatmap(train2.corr(), linewidths=0.01, square=True,  # squre = 정사각형 , linewidths = 사각형간 폭
            annot=True, cmap=plt.cm.viridis, linecolor="white")  #annot = 사각형내부 숫자표기, cmap = 색반전
plt.title('Correlation between features')
plt.show()
```



03. 데이터 전처리 및 EDA

```
# 네이버 API 사용

import requests # HTTP 요청을 보내는 모듈
import pandas as pd

client_id = "vfWfdjH8P62AiPuDjNRs" # 네이버 api Client ID
client_secret = "i7m_dOLHIh"  # 네이버 api Client Secret
```

```
search_word = '카드연체' # 검색어
encode_type = 'json' # 출력 방식 json 또는 xml
max_display = 100 # 출력 뉴스 수
sort = 'sim' # 결과값의 정렬기준 시간순 date, 관련도 순 sim
start = 1 # 출력 위치

# get 메서드를 사용하여 네이버 기사를 검색해오기
url = f"https://openapi.naver.com/v1/search/news.{encode_type}
?query={search_word}&display={str(int(max_display))}
&start={str(int(start))}&sort={sort}"

# 헤더에 아이디와 키 정보 빙력
headers = {'X-Naver-Client-Id' : client_id,
           'X-Naver-Client-Secret':client_secret
           }

#HTTP요청 보내기 (r = response)
r = requests.get(url, headers=headers)
#요청 결과 보기 200 이면 정상적으로 요청 완료
print(r)
```

'카드연체' 키워드로 검색

네이버 api 사용하여
기사에서 검색어 검색

```
import re # 정규표현식 쓰기 위한 라이브러리
```

```
# 정규표현식을 이용하여 html코드의 불필요한 항목 제거
# 함수 정의
def clean_html(x):
  x = re.sub("㎡&㎡w*㎡;","",x) # &quot; 제거
  x = re.sub("<.*?>","",x) # <b>, </b> 제거
  return x

# title과 description에만 clean_html() 적용
df['title'] = df['title'].apply(lambda x: clean_html(x))
df['description'] = df['description'].apply(lambda x: clean_html(x))
```

정규식 표현 사용하여
단어에 들어있는 불필요항
목 제거

```
# 필요없는 컬럼 삭제
del df["link"]
del df["originallink"]
del df["pubDate"]
df
```

```
from konlpy.tag import Okt # Open-korean-tect;과거 트위터 형태소 분석기
from collections import Counter # 단어 개수 카운트하는 라이브러리
from wordcloud import WordCloud # 워드클라우드 라이브러리
```

Konlpy, collections,
wordcloud 라이브러리
임포트

네이버 뉴스 API 사용, '카드연체' 키워드와 관련있는
연관 단어들은 어떤 것들이 검색되는지 확인

03. 데이터 전처리 및 EDA

```python
# 명사 추출
def get_noun(df):
    t = Okt() # Okt객체 선언
    noun = t.nouns(df) # 명사만 추출해서 리스트에 담기
    for i,v in enumerate(noun): # i:index, v:value
        if len(v) < 2: # 한글자인 명사 제거
            noun.pop(i)
    count = Counter(noun)
    noun_list = count.most_common(100) # 빈도가 높은 100개의 명사 카운트
    return noun_list
```

추출한 단화를 시각화 하여 png 파일로 저장

```python
# 시각화
def visualize(noun_list):
    wc = WordCloud(font_path=
    'C:/Users/CJ/Anaconda3/Lib/site-packages/pytagcloud/fonts/NanumGothic.ttf',
                width = 1000,
                height = 1000,
                background_color="white",
                max_words = 100,
                max_font_size = 300)
    wc.generate_from_frequencies(dict(noun_list))
    wc.to_file(f'{search_word}.png')
```

```python
if __name__=="__main__":
    f = open(f'news_search_result_{search_word}.csv',mode='r',encoding='utf-8')
    df = f.read() # 파일 읽기
    noun_list = get_noun(df) # 명사 추출
    visualize(noun_list) # 시각화
print(noun_list)
```

'카드연체' 키워드로 뉴스에서 검색한 결과 대출, 신용, 금융 등 기업에게 있어
카드 연체관리가 중요한 사회적 이슈임을 확인할 수 있었다.

04. 가설 검정

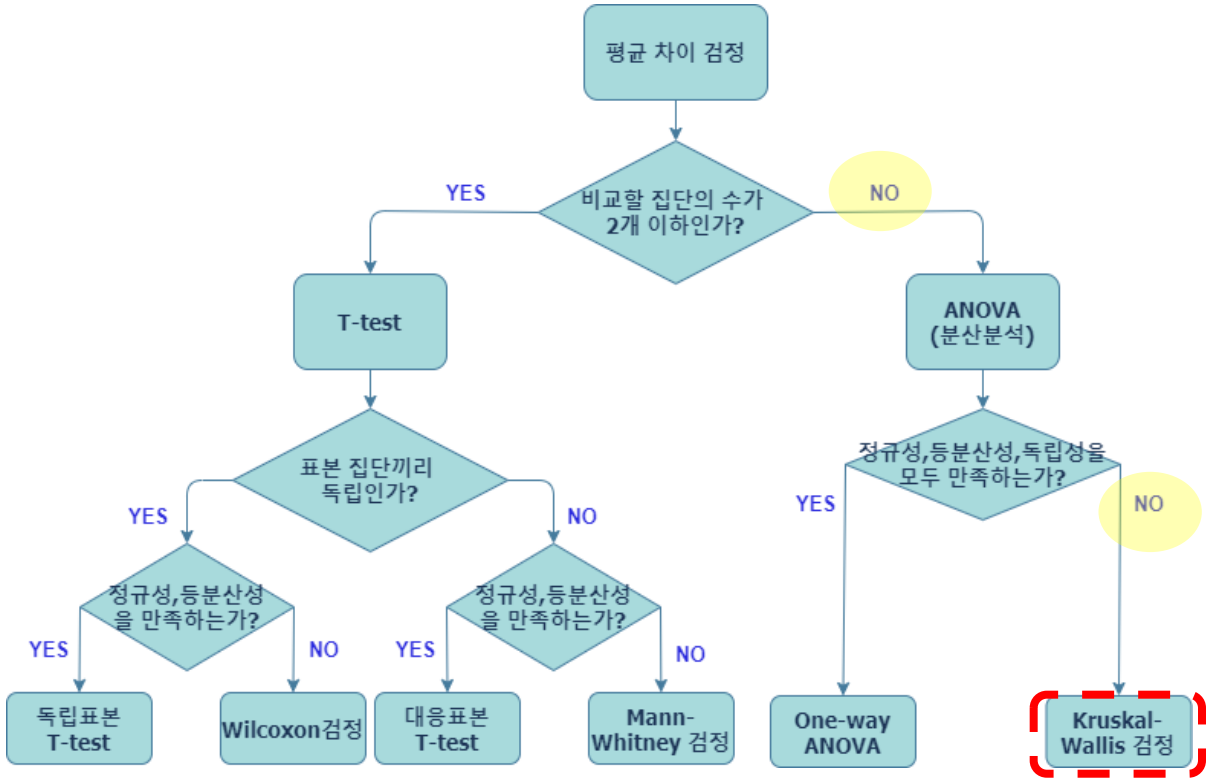
추론통계

연속형 변수 begin_month(카드사용기간)와
credit(신용등급)과의 관계를 통계적으로 검증하고자 함

➔ 카드발급일로부터 데이터 수집 당시 까지의 기간
(즉, 카드 사용 기간) 이 신용등급에 영향을 준다.

* **대립가설** : 카드사용기간(0~3범주)별 신용등급에 대한 평균의 차이가 유의하다.
* **귀무가설** : 카드사용기간별 신용등급에 대한 평균의 차이는 유의하지 않다.
카드사용기간이 신용등급에 영향을 주지 않는다.

Kruskal-Wallis 통계검정
세 집단 이상의 집단 분포를 비교하는
비모수 검정 방법



04. 가설 검정

데이터 읽어오기

```
df <- read.csv('C:/Users/CJ/Documents/open2/train2.csv')
```

→ 전 처리 과정을 통해 범주형 변수로 변경된 train2.csv 데이터 셋 파일 읽어 오기

등분산성 여부 검정

```
> bartlett.test(credit~begin_month, data = df)

        Bartlett test of homogeneity of variances

data:  credit by begin_month
Bartlett's K-squared = 43.015, df = 11, p-value = 1.08e-05
```

→ 등분산성 여부 검정
: P-value < 0.05 → 등분산성(동질성) 만족하지 않음.

Kruskal-Wallis 검정

```
> kruskal.test(credit~begin_month, data = df)

        Kruskal-Wallis rank sum test

data:  credit by begin_month
Kruskal-Wallis chi-squared = 2108.3, df = 11, p-value < 2.2e-16
```

→ 독립변수 begin_month에 따른 종속변수 credit의 P-value 값이 유의수준 0.05보다 작기 때문에 통계적으로 유의함을 확인 → **귀무가설 기각, 대립가설 채택**

Kruskal-Wallis 검정 결과 따르면
카드사용기간이 신용등급에 영향을 미치는 것으로 확인

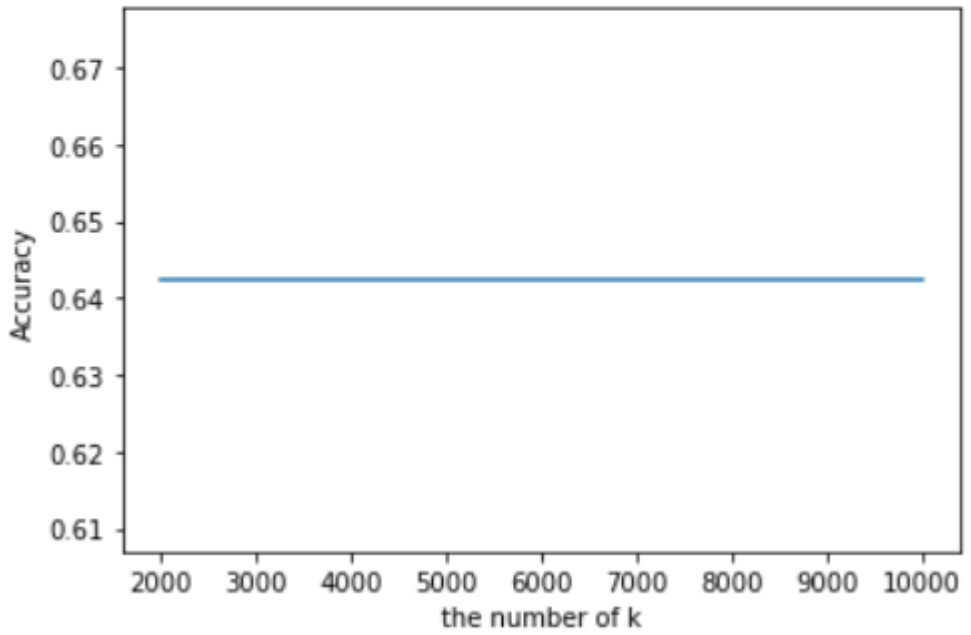
05. 머신러닝 구현 - KNN

```
# 최적의 k를 찾기 위해 교차 검증을 수행할 k의 범위를 학습 데이터 절반까지 지정
max_k_range = train_set.shape[0] // 2

#정확도 값을 저장하기 위한 리스트 초기화
k_list = []

#학습 범위는 max_k_range에서 2000개 잘라서 리스트 작성
for i in range(2000, max_k_range, 2000):
    k_list.append(i)

cross_validation_scores = []


# 교차 검증(10-fold)을 각 k를 대상으로 수행해 검증 결과를 저장
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k) #KNN 분류기 사용, k번 학습 반복

    #단순 교차검증의 파라미터: (모델명, 훈련데이터, 타깃(정답), cv=폴드 수)
    scores = cross_val_score(knn, x_train_scale, y_train.values.ravel(), cv=10,
                             scoring='accuracy') #scoring은 출력항목
    cross_validation_scores.append(scores.mean()) #교차검증 값의 평균을 출력

cross_validation_scores
```

KNN 모델의 교차검증 결과약
64% 값이 도출됨

```
[0.6423501756665118,
 0.6423501756665118,
 0.6423501756665118,
 0.6423501756665118,
 0.6423501756665118]
```

```
# k에 따른 정확도를 시각화
plt.plot(k_list, cross_validation_scores)
plt.xlabel('the number of k')
plt.ylabel('Accuracy')
plt.show()
```



KNN(K-Nearest Neighbor) 알고리즘 : 거리, 빈도를 기준으로 샘플링의 라벨링을 결정

sklearn 라이브러리의 KNNClassifier를 사용, K-fold 교차 검증을 수행한 알고리즘
최적의 K값을 찾기 위해 전체 데이터에서 절반까지 범위로 지정
10-fold 방식으로 10번 교차 검증을 수행, 평균값을 도출

05. 머신러닝 구현 - KNN

```
# 가장 예측율이 높은 k를 선정
k = k_list[cross_validation_scores.index
           (max(cross_validation_scores))]
print("The best number of k : " + str(k))
```

```
The best number of k : 2000
```

```
# KNN에 스케일링된 학습 데이터셋과
# 정답 레이블을 만든 데이터셋을 대응하여 저장
knn.fit(x_train_scale, y_train.values.ravel())

# 스케일링된 테스트 데이트를 예측하고 저장
pred = knn.predict(x_test_scale)

# 모델 예측 정확도 출력
print("accuracy : " +
      str(accuracy_score(y_test.values.ravel(), pred)))
```

```
accuracy : 0.6322751322751323
```

교차 검증을 통해 가장 예측률이 높은 최적의 K값 도출
KNN 모델의 예측도는 63% 정확도를 보임
test 데이터 셋의 예측 값과 정답 값 비교

```
# KNN 학습방법을 이용하여
# test-set의 값을 예측하여 정답과 함께 표시
comparison = pd.DataFrame(
    {'prediction':pred,
     'ground_truth':y_test.values.ravel()})
comparison
```

	prediction	ground_truth
0	2.0	2.0
1	2.0	2.0
2	2.0	1.0
3	2.0	1.0
4	2.0	0.0
...	...	...
2641	2.0	2.0
2642	2.0	1.0
2643	2.0	2.0
2644	2.0	1.0
2645	2.0	1.0

2646 rows × 2 columns

predict 함수를 사용, 스케일링된 문제(x_test)를 예측

예측한 결과와 비교해서 정답(y_test)을 비교하여 정확도를 구하는 함수 사용(accuracy_score)

05. 머신러닝 구현 - SVM

```
def svc_param_selection(x, y, nfolds):

# SVM(Support Vector Machine) 변수
#   Radial Basis Function(방사형 기저 함수)
#   커널에서는 C, gamma변수가 필요

# (RBF)방사형 기저 함수
#   예측 변수의 값을 기준으로
#   하나 이상의 종속변수에 대한 예측 모형

# C(Cost): 얼마나 많은 데이터 샘플이
#         다른 클래스에 놓이는 것을 허용하는지

# gamma: 하나의 데이터 샘플이 영향력을 행사하는
#       거리를 결정, 값이 커질수록 거리는 짧아짐
```

```
    svm_parameters = [
                        {'kernel' : ['rbf'],
                         'gamma': [0.01, 0.1, 1],
                         'C': [10, 100, 1000]}
                      ]

    # sklearn 라이브러리의
    # GridSearchCV 패키지 사용하여 최적의 파라미터를 구함
    clf = GridSearchCV(SVC(), svm_parameters, cv=5)
    clf.fit(x_train, y_train.values.ravel())
    print(clf.best_params_)

    return clf

# 최적의 파라미터로 학습된 모델을 classifier로 저장
clf = svc_param_selection(x_train, y_train.values.ravel(), 5)
```

최적의 파라미터를 찾아주는
GridSearchCV 사용
파라미터를 입력하고 5회 진행

```
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

SVM(Support Vector Machine) 알고리즘 : 마진을 최대화 하여 최적의 결정 경계를 결정하여 데이터를 분류

마진이 클수록 새로운 데이터에 대해 안정적으로 분류할 가능성이 높아짐
커널 트릭 사용하여 KNN보다 강력함, 파라미터 조정을 통한 과적합 문제 해결, 적은 데이터로 괜찮은 성능 기대
하지만, 데이터 전처리 과정이 중요, 결정 경계 및 시각화가 어렵다는 단점이 있음.

05. 머신러닝 구현 - SVM

```
# 최적의 파라미터로 학습된 모델로 테스트를 진행
y_true, y_pred = y_test, clf.predict(x_test)

# 평가 지표를 확인하기 위해 classification_report 함수 사용
print(classification_report(y_true, y_pred))
print()
print("accuracy : " + str(accuracy_score(y_true, y_pred)) )

# macro : 라벨 별(Credit) 각 합의 평균
#         (0확률 평균 + 1확률 평균 + 2확률평균 / 3) - 평균의 평균

# weighted : 가중치 평균, 라벨(Credit) 0 1 2 별 가중치 값이 나왔을 때
#            0의 가중치 + 1의 가중치 + 2의 가중치 / 3
```

	precision	recall	f1-score	support
0.0	0.33	0.04	0.07	311
1.0	0.52	0.15	0.23	629
2.0	0.67	0.95	0.78	1706
accuracy			0.65	2646
macro avg	0.50	0.38	0.36	2646
weighted avg	0.59	0.65	0.57	2646

정밀도, 재현율, F1-Score
전부 신용등급(Credit)
2등급을 맞추는 확률이
가장 높음을 확인할 수 있음

```
accuracy : 0.6519274376417233
```

최적의 파라미터로 학습된 모델로 테스트 진행
전체적인 평가 지표를 보여주는 sklearn 라이브러리의
classification_report 함수 사용

```
# SVM 학습방법을 이용하여
# test-set의 값을 예측하여 정답과 함께 표시
comparison2 = pd.DataFrame(
    {'prediction':y_pred,
     'ground_truth':y_true.values.ravel()})
comparison2
```

	prediction	ground_truth
0	2.0	1.0
1	2.0	2.0
2	2.0	2.0
3	2.0	2.0
4	2.0	1.0
...	...	...
2641	2.0	2.0
2642	2.0	2.0
2643	2.0	2.0
2644	2.0	2.0
2645	2.0	2.0

2646 rows × 2 columns

predict 함수를 사용,
스케일링된 문제
(x_test)를 예측

예측한 결과와 비교해서
정답(y_test)을 비교하
여 정확도를 구하는 함수
사용(accuracy_score)

05. 머신러닝 구현 - 의사결정나무

```
# 의사결정나무(DecisionTree):
# 목적과 자료기준에 따라 분리기준과 정지규칙을
# 지정하여 의사결정나무를 얻는 알고리즘.

# 한 번의 분기 때마다 변수 영역을 두개로 구분
# 기본 개념은 Leaf Node가 섞이지 않는 상태로 완전히 분류되는 것,

# 즉, 복잡성(entropy)이 낮도록 만드는 것.

# 분리기준: 부모마디 대비 자식마디에서 순수도가 증가하는 정도

# 정지규칙: 더 이상 분리가 일어나지 않음

# 가지치기: 적절하지 않은 마디 제거
```

```
# preprocessing 라이브러리에서
# 라벨값 입력해주는 함수를 불러옴.(0 ~ 항목의 갯수-1)
# 라벨값을 부여해야 의사결정나무에서 기준 설정 가능
le = preprocessing.LabelEncoder()

# fit_transform 함수를 통해 데이터셋을
# 명목형 데이터셋으로 스케일링하고 학습시킬 수 있음
y_encoded = le.fit_transform(y_train)

# 의사결정나무(랜덤 시드값 5) 생성,
# 학습데이터의 x, y를 대응시킨 후 학습
clf = tree.DecisionTreeClassifier
    (random_state=5).fit(x_train, y_encoded)
```

```
pred = clf.predict(x_DTtest)
```

```
print("accuracy : " + str(
    accuracy_score(y_test.values.ravel(),
                   le.classes_[pred])))
```

```
accuracy : 0.5884353741496599
```

의사결정나무(DecisionTree) 알고리즘 : 일련의 분류 규칙을 통해 데이터를 분류하는 학습 모델

데이터 전처리 과정이 중요, 결정 경계 및 시각화가 어려움. 데이터의 특징을 바탕으로 연속적으로 분리하다 보면 결국 하나의 정답으로 데이터를 분류
과대적합의 위험이 높으므로 학습 시 Leaf와 Node에 적절한 제한이 필요함

05. 머신러닝 구현 - 의사결정나무

```
import graphviz
#DOT(그래프 기술 언어) 형태로 추출
dot_data = tree.export_graphviz(clf, out_file=None)

#추출된 DOT 데이터를 시각화
graph = graphviz.Source(dot_data)

#타겟(신용등급)을 목표로 의사결정나무 파일을 만들어준다.
graph.render("credit")
dot_data = tree.export_graphviz(clf, out_file=None,
            feature_names = ['gender', 'car', 'reality', 'child_num',
                             'income_total', 'income_type', 'edu_type',
                             'family_type', 'house_type', 'DAYS_BIRTH',
                             'DAYS_EMPLOYED', 'FLAG_MOBIL', 'work_phone',
                             'phone', 'email', 'family_size', 'begin_month',
                             'credit'],
            class_names=['0', '1', '2'],
            filled=True, rounded=True,
            special_characters=True)

graph = graphviz.Source(dot_data)
graph
```

```
#의사결정나무를 학습방법을 이용하여
# test-set의 값을 예측하여 정답과 함께 표시
comparison3 = pd.DataFrame(
    {'prediction':le.classes_[pred],
     'ground_truth':y_DTtest.values.ravel()})
comparison3
```

	prediction	ground_truth
0	2.0	1.0
1	2.0	2.0
2	2.0	2.0
3	2.0	2.0
4	1.0	1.0
...	...	...
2641	2.0	1.0
2642	0.0	0.0
2643	0.0	2.0
2644	2.0	2.0
2645	2.0	2.0

2646 rows × 2 columns

DecisionTree 모델의 예측도는 58% 정확도를 보임

test 데이터 셋의 예측 값과 정답 값 비교

Graphviz 라이브러리 사용하여 DOT 형태로 추출, 추출한 DOT 시각화한 후 타겟(신용등급)을 목표로 의사결정나무 파일을 만들어 줌

05. 머신러닝 구현 - 랜덤포레스트

```
# 의사결정나무를 이용한 랜덤포레스트 추가
# 의사결정나무가 가진 오버피팅 단점을 해결
clf = RandomForestClassifier()
clf.fit(x_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
pred = clf.predict(x_test)
```

```
print("accuracy : " + str(accuracy_score(y_test, pred)))
```

```
accuracy : 0.6379440665154951
```

→ RandomForest 정확도 약 64%

```
# 랜덤포레스트 학습방법을 이용하여
# test-set의 값을 예측하여 정답과 함께 표시
comparison4 = pd.DataFrame(
    {'prediction':pred,
     'ground_truth':y_test.values.ravel()})
comparison4
```

	prediction	ground_truth
0	2.0	2.0
1	2.0	2.0
2	2.0	2.0
3	2.0	1.0
4	2.0	1.0
2644	2.0	2.0
2645	2.0	1.0

2646 rows × 2 columns

랜덤포레스트(RandomForest) 알고리즘 : 여러 개 의사결정나무를 활용하여 데이터를 분류

여러 개의 의사결정나무를 활용한 bagging 방식의 대표적인 알고리즘
앙상블 알고리즘 중 비교적 빠른 수행 속도를 지니고 다양한 분야에서 좋은 성능을 나타내지만
하이퍼 파라미터가 많아 튜닝 시간 소요가 된다는 단점을 가지고 있다.

05. 머신러닝 구현 - XGBoost

학습 단계별 가중치 설정
(Default=0.1)

```
#XGBoost의 분류기 사용, 횟수는 500회, 학습률은 0.2, 최대 깊이는 4
xgb_classifier = XGBClassifier(n_estimator=500, learning_rate=0.2, max_depth=4)
xgb_classifier.fit(X_train, Y_train)
xgb_pred = xgb_classifier.predict(X_test)
```

트리의 최대 깊이
(Default=6)

결정트리의 개수 지정
(Default=10)

```
[15:35:17] WARNING: ..₩src₩learner.cc:541:
Parameters: { n_estimator } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[15:35:17] WARNING: ..₩src₩learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation metric used
with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_metric if
you'd like to restore the old behavior.
```

```
#예측한 값을 정답과 비교
accuracy_score(Y_test, xgb_pred)
```

```
0.6462585034013606
```

XGBoost 정확도 64%

classificaion_report 사용하여 평가지표를 한눈에 확인

```
#평가지표를 한번에 볼 수 있는 함수
classification_report(Y_test, xgb_pred, target_names=['0', '1', '2'])
```

```
'              precision    recall  f1-score   support₩n₩n           0       1.00      0.00      0.01
306₩n           1       0.44      0.09      0.15       636₩n           2       0.66      0.97      0.78
1704₩n₩navg / total       0.64      0.65      0.54      2646₩n'
```

XGBoost 알고리즘
(eXtreme Gradient Boosting)

Ensemble – Boosting 기법의 한 종류

이전 모델에서의 실제 값의 오차(loss)를 훈련데이터에 투입하고 gradient를 이용하여 오류를 보완하는 방식

Gradient Boosting Machine에 기반하고 있지만, GBM의 단점인 느린 수행시간, 과적합 규제 등을 해결한 알고리즘

05. 머신러닝 구현 - XGBoost

```
xgb_Classifier = XGBClassifier()

xgb_param_grid = {
    'n_estimators' : [20, 40, 60, 80, 100],
    'learning_rate' : [0.01, 0.05, 0.1, 0.15, 0.2],
    'max_depth' : [4, 6, 8, 10, 12],
}

#GridSearchCV방법을 이용, 최적의 파라미터를 찾아낸다.
#n_jobs=-1: CPU코어 중 하나를 가져다 쓴다.
#verbose: 학습중인 과정을 보이는 파라미터
#0: silent, 1: progress bar, 2: one line per epoch
xgb_grid = GridSearchCV(xgb_Classifier, param_grid = xgb_param_grid,
                        scoring="accuracy", n_jobs=-1, verbose=1)
xgb_grid.fit(X_train, Y_train)
```

```
Fitting 5 folds for each of 125 candidates, totalling 625 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   30.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  9.4min
[Parallel(n_jobs=-1)]: Done 625 out of 625 | elapsed: 13.6min finished
```

```
print("최고 평균 정확도 : {0:.4f}".format(xgb_grid.best_score_))
print("최고의 파라미터 : ", xgb_grid.best_params_)
```

```
최고 평균 정확도 : 0.6423
최고의 파라미터 :  {'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 100}
```

```
cv_result_df = pd.DataFrame(xgb_grid.cv_results_)
cv_result_df.sort_values(by=['rank_test_score'], inplace=True)
```

```
cv_result_df[['params', 'mean_test_score', 'rank_test_score']].head(10)
```

	params	mean_test_score	rank_test_score
104	{'learning_rate': 0.2, 'max_depth': 4, 'n_esti...	0.642308	1
103	{'learning_rate': 0.2, 'max_depth': 4, 'n_esti...	0.642266	2
77	{'learning_rate': 0.15, 'max_depth': 4, 'n_est...	0.642098	3
81	{'learning_rate': 0.15, 'max_depth': 6, 'n_est...	0.641888	4
52	{'learning_rate': 0.1, 'max_depth': 4, 'n_esti...	0.641846	5
58	{'learning_rate': 0.1, 'max_depth': 6, 'n_esti...	0.641804	6
53	{'learning_rate': 0.1, 'max_depth': 4, 'n_esti...	0.641762	7
102	{'learning_rate': 0.2, 'max_depth': 4, 'n_esti...	0.641762	8
76	{'learning_rate': 0.15, 'max_depth': 4, 'n_est...	0.641762	9
78	{'learning_rate': 0.15, 'max_depth': 4, 'n_est...	0.641636	10

GridSearchCV를 활용하여 최적의 파라미터 값을 찾음

평균 정확도는 64% 이고, 최적의 파라미터는

Learning rate: 0.2, max_depth: 4, n_estimators: 100 얻을 수 있었음

05. 머신러닝 구현 - XGBoost

```
#파이썬 라이브러리를 사용해서 XGBoost를 사용하려면 xgb에 내장된 함수를 이용하여
#데이터프레임 형식으로 바꿔 준 후 XGBoost를 사용해야 한다.
dtrain = xgb.DMatrix(data=X_train, label = Y_train)
dtest = xgb.DMatrix(data=X_test, label=Y_test)
```

```
#XGBoost의 파라미터 설정
params = {'max_depth' : 4,
          'eta' : 0.2,
          'eval_metric' : 'mlogloss',
          'objective' : 'multi:softmax',
          'num_class' : 3,
          'early_stoppings' : 100 }

num_rounds = 400
```

```
#제출용 답안에 컬럼[0, 1, 2]의 모든 row을 0으로 초기화
sub=np.zeros((X_test.shape[0], 3))

#XGBoost의 학습, 정확도 산출을 위해 리스트 형태로 테스트 데이터와 훈련 데이터를 묶어줌
wlist = [(dtrain, 'train'), (dtest,'eval')]

# 하이퍼 파라미터와 early stopping 파라미터를 train() 함수의 파라미터로 전달
xgb_model = xgb.train(params = params, dtrain=dtrain,
                      num_boost_round=num_rounds, evals=wlist)

#값 예측
predictions = xgb_model.predict(dtest)
```

전 단계에서 얻은
최적 파라미터 값을 설정해줌
* 파이썬, 사이킷런 래퍼에 따라 명칭은 다를 수 있음

eval_metric : mlogloss
(손실 적용하여 모델을 최적화)

early_stoppings : 100
(100회 학습 후, 오류 개선되지 않으면 수행을 중지)

```
# XGBoost 학습방법을 이용하여
# test-set의 값을 예측하여 정답과 함께 표시
comparison4 = pd.DataFrame(
    {'prediction':sub,
     'ground_truth':Y_test.values.ravel()})
comparison4
```

	prediction	ground_truth
0	2.0	2.0
1	2.0	2.0
2	2.0	1.0
3	2.0	2.0
4	2.0	2.0
...	...	...
2641	2.0	0.0
2642	1.0	2.0
2643	2.0	2.0
2644	0.0	0.0
2645	2.0	2.0

2646 rows × 2 columns

05. 머신러닝 구현 - 모델 성능평가

```
#각 정답별 ROC curve 구하기

#정답은 0, 1, 2 셋 중 하나이므로 class갯수 선언
n_classes = 3

#데이터 가져오기
a = train.drop(['credit'], axis=1)
b = train['credit']

#원-핫 인코딩처럼 각 변수별로 해당 변수는 1, 나머지는 0 식으로 클래스별로 각각 부여
b = label_binarize(b, classes=[0, 1, 2])

#훈련과 검증용 데이터로 나누기
a_train, a_test, b_train, b_test = train_test_split(a, b, test_size=0.1)

#OnevsRest 분류기에 xgboost분류기를 넣어 정답별로 xgboost를 사용했을 때 roc곡선을 구함
ovrclf = OneVsRestClassifier(XGBClassifier())
ovrclf.fit(a_train, b_train)
#ovrclf의 예측 확률을 담은 변수 선언
b_score = ovrclf.predict_proba(a_test)

#x_train : y_train, x_test : y_test 쌍을 만들기 위해 딕셔너리 형태 사용
fpr = dict() #FPR: FP / all negatives(FP + TN), 특이도
tpr = dict() #TPR: TP / all positives(TP + FN), 재현율(민감도)
roc_auc = dict()
```

ROC_AUC Curve

* ROC - 모델의 분류 성능에 대한 측정 그래프

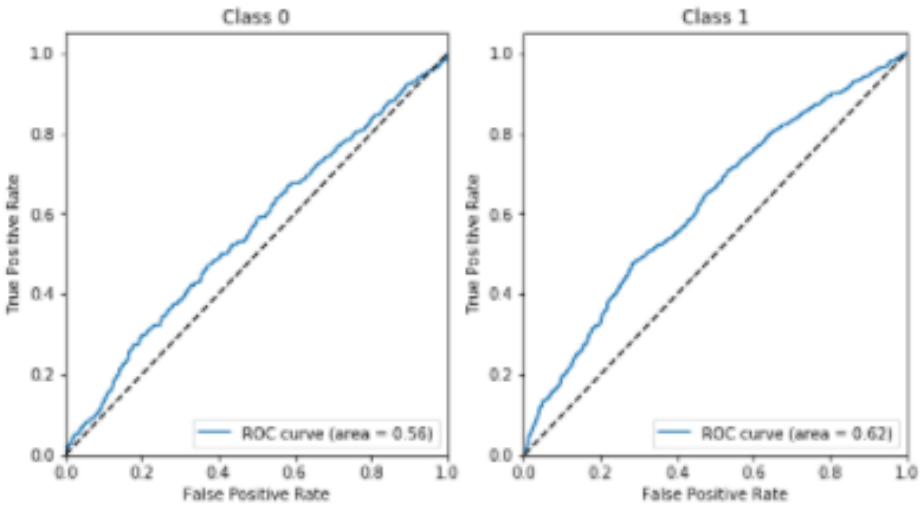
* AUC - ROC 곡선 아래 영역 (AUC ↑, 모델 성능 훌륭함)

Multiclass 라이브러리 OnevsRestClassifier를 사용변수vs나머지 총 3가지 경우로 나누어 3개의 roc곡선을 표시

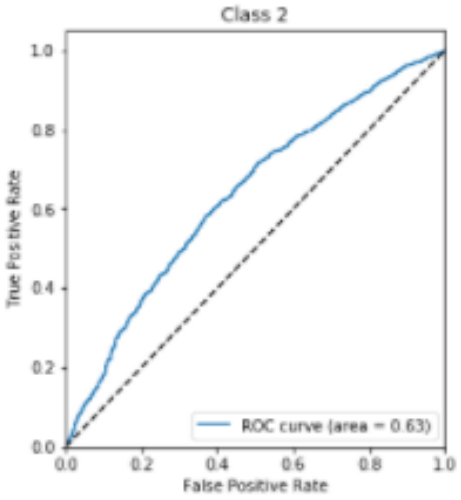
특이도 - 실제값이 False인 관측치 중 예측치가 적중한 정도

재현율 - 실제값이 True인 관측치 중 예측치가 적중한 정도

05. 머신러닝 구현 - 모델 성능평가



roc_auc_score: 0.6031106478810117



AUC값이 1에 가까울 수록

모델의 성능 좋음

(TPR ↑, FPR ↓)

모델의 성능평과 결과 0.6, 매우 좋진 않으나 나쁘지 않다고 정도로 볼 수 있음

```python
#n_classes(0, 1, 2)에 대한 각각의 roc곡선을 그리기 위해 사용
for i in range(n_classes):
    #입력 파라미터: index[0] = 실제 class값의 array(데이터 건수)
    #               index[1] = predict_proba의 반환값
    #0, 1, 2 컬럼(항목)별로 데이터 값을 가져옴
    fpr[i], tpr[i], _ = roc_curve(b_test[:, i], b_score[:, i])
    #roc 곡선을 그리기 위해 fpr, tpr을 파라미터로 auc함수 사용
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(15, 5))

#3개의 roc곡선을 그리기 위해 열거형으로 반복문 사용
#순서가 있는 자료형(리스트, 튜플, 문자열)을 받아 인덱스값을 포함하는 열거형 객체 리턴
for idx, i in enumerate(range(n_classes)):
    plt.subplot(131 + idx)
    #설명력, roc_auc[i]의 수치 만큼 설명하는데 근거가 된다.
    #%0.2f에 값을 대응하기 위한 포맷으로 %roc_auc[i]의 값을 가져온다.
    plt.plot(fpr[i], tpr[i], label = 'ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    #pyplot을 그렸을 때 x의 끝이 1.0으로 설정
    plt.xlim([0.0, 1.0])
    #pyplot을 그렸을 때 x의 끝을 1.05로 설정, ROC곡선을 보기 위함
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Class %0.0f' % idx)
    plt.legend(loc = "lower right")

plt.show()

print("roc_auc_score: ", roc_auc_score(b_test, b_score, multi_class='raise'))
```

X축 특이도
Y축 재현율(민감도)

06. 인사이트 도출

카드사용 초기부터 소비정보 제공 및 한도 조정으로 연체 관리

- 신용등급 1, 2등급의 약 50% 이상, 카드사용 초기에 분포
- 사용기간 1년 이하, 신용등급 1등급 가장 많이 분포
- 사용기간 3년 이상부터 신용등급 2등급 사용자가 증가

✓ 카드 소비 실시간 현황 파악을 위한 방안 마련

△ 사용자 스스로 카드사용을 관리를 통한 연체 리스크 관리

ex) 소비내역 문자 및 어플 알림 등 다방면에서의 실시간 소비 정보 제공

✓ 카드 사용 및 대출 한도 점진적 상향 조정

△ 대출한도를 하향 조정하여 카드 사용초기 연체에 따른 리스크 최소화

ex) 사용자의 사용 평가에 따른 한도액 단계별 상향 조정