[은행파산 위기 데이터 분석]

- 개인의 파산을 예측하는 대회
- 대회명 : Bankruptcy Risk Prediction
- 데이터 출처 : https://www.kaggle.com/competitions/bankruptcy-risk-prediction

- 데이터 분석 코드
    - HTML코드

학습 내용

- 대회 개요 및 데이터 설명
- 라이브러리 및 데이터 불러오기
- 데이터 살펴보기(시각화, EDA)

목차

01 대회 개요 및 데이터 설명

목차로 이동하기

대회 개요

- 대회 측정 지표 : ROC~AUC
    - ROC (Receiver Operating Characteristic) 곡선은 이진 분류 모델의 성능을 평가하는 데 사용되는 그래프입니다. 이 곡선은 모델의 True Positive Rate (TPR)와 False Positive Rate (FPR)를 다양한 임계값에서 비교하여 작성됩니다.
    - AUC (Area Under the Curve)는 ROC 곡선 아래의 면적을 나타내며, 모델의 분류 성능을 하나의 수치로 표현합니다. AUC 값은 0에서 1 사이의 값으로 표현됩니다.
- 예측 : proba - true label 예측 확률

- 데이터 셋
    - test.csv
    - train.csv
    - submission_example.csv

데이터에 대한 설명

- train.csv : 800행 22열 (bankruptcy : 파산여부, 타겟변수 0 또는 1)
- test.csv : 200행 21열 (ID 포함, 타겟 변수 없음)
- submission.csv : 200행 2열 (ID와 proba)

train.csv

변수명	설명
id	예제의 식별 번호
sum	대출 금액
term	대출 기간(월 수)
payment	월 상환액
guarantees	담보의 종류
reason	대출 사유
credits	기존의 대출 수
other_credits	다른 대출의 상태
credit_report	신용 보고서 상태
marital_status	결혼 상태
age	나이
employment	고용 기간
qualification	자격
immigrant	이민자 여부
residence_since	현재 거주지에 거주한 기간(년 수)
accommodation	주거 형태
estate	소유 부동산
savings	저축 계좌 상태
dependents	부양 가족 수
phone	전화 소유 여부
status	체크 계좌 상태
bankruptcy	파산 여부(타겟 변수, 0 또는 1)

범주형, 연속형 변수 구분

연속형

변수명	설명	예제 값	구분
term	대출 기간(월 수)	6, 48, 12, 42	숫자형
sum	대출 금액	1169, 5951, 2096, 7882	숫자형
payment	월 상환액	4, 2, 2, 2	숫자형

변수명	설명	예제 값	구분
residence_since	현재 거주지에 거주한 기간(년 수)	4, 2, 3, 4	숫자형
age	나이	67, 22, 49, 45	숫자형
credits	기존의 대출 수	2, 1, 1, 1	숫자형
dependents	부양 가족 수	1, 1, 2, 2	숫자형

범주형 변수

변수명	설명	예제 값	구분
status	체크 계좌 상태	less-than 0 cu, 0 to 200 cu, no checking account	범주형
credit_report	신용 보고서 상태	critical account or other credits existing (not at this bank), existing credits paid back duly till now	범주형
reason	대출 사유	television or radio, education, furniture or equipment	범주형
savings	저축 계좌 상태	unknown or no savings account, less-than 100 cu	범주형
employment	고용 기간	7+ y., 1 to 4 y., 4 to 7 y.	범주형
marital_status	결혼 상태	male single, female divorced or separated or married	범주형
guarantees	담보의 종류	none, guarantor	범주형
estate	소유 부동산	real estate, building society savings agreement or life insurance	범주형
other_credits	다른 대출의 상태	none	범주형
accommodation	주거 형태	own, for free	범주형
qualification	자격	skilled employee, unskilled resident	범주형
phone	전화 소유 여부	yes, none	범주형
immigrant	이민자 여부	yes	범주형

02 라이브러리 및 데이터 기본탐색

목차로 이동하기

라이브러리 불러오기

In [ ]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

데이터 불러오기

In [ ]:
```
sub = pd.read_csv("/kaggle/input/bankruptcy-risk-prediction/submission_example.csv")
train = pd.read_csv("/kaggle/input/bankruptcy-risk-prediction/train.csv")
test = pd.read_csv("/kaggle/input/bankruptcy-risk-prediction/test.csv")

train.shape, test.shape, sub.shape
```

Out[ ]:
```
((800, 22), (200, 21), (200, 2))
```

In [ ]:
```
# 학습용 데이터 셋 컬럼명 확인
train.columns
```

Out[ ]:
```
Index(['id', 'sum', 'term', 'payment', 'guarantees', 'reason', 'credits',
       'other_credits', 'credit_report', 'marital_status', 'age', 'employment',
       'qualification', 'immigrant', 'residence_since', 'accommodation',
       'estate', 'savings', 'dependents', 'phone', 'status', 'bankruptcy'],
      dtype='object')
```

In [ ]:
```
# 테스트용 데이터 셋 컬럼명 확인
test.columns
```

Out[ ]:
```
Index(['id', 'sum', 'term', 'payment', 'guarantees', 'reason', 'credits',
       'other_credits', 'credit_report', 'marital_status', 'age', 'employment',
       'qualification', 'immigrant', 'residence_since', 'accommodation',
       'estate', 'savings', 'dependents', 'phone', 'status'],
      dtype='object')
```

- bankruptcy의 컬럼의 값을 예측하는 과제

In [ ]:
```
# bankruptcy(파산 여부)의 값 확인
train['bankruptcy'].unique()
```

Out[ ]:
```
array([0, 1])
```

In [ ]:
```
# bankruptcy의 값과 개수 확인
train['bankruptcy'].value_counts()
```

Out[ ]:
```
bankruptcy
0    561
1    239
Name: count, dtype: int64
```

In [ ]:
```
# 데이터 셋의 전반적인 정보 확인
# 컬럼명, 결측치 확인, 컬럼의 자료형 형태 등
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 22 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               800 non-null    int64
 1   sum              800 non-null    int64
 2   term             800 non-null    int64
 3   payment          800 non-null    int64
 4   guarantees       800 non-null    object
 5   reason           800 non-null    object
 6   credits          800 non-null    int64
 7   other_credits    800 non-null    object
 8   credit_report    800 non-null    object
 9   marital_status   800 non-null    object
 10  age              800 non-null    int64
 11  employment       800 non-null    object
 12  qualification    800 non-null    object
 13  immigrant        800 non-null    object
 14  residence_since  800 non-null    int64
 15  accommodation    800 non-null    object
 16  estate           800 non-null    object
 17  savings          800 non-null    object
 18  dependents       800 non-null    int64
 19  phone            800 non-null    object
 20  status           800 non-null    object
 21  bankruptcy       800 non-null    int64
dtypes: int64(9), object(13)
memory usage: 137.6+ KB
```

In [ ]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               200 non-null    int64
 1   sum              200 non-null    int64
 2   term             200 non-null    int64
 3   payment          200 non-null    int64
 4   guarantees       200 non-null    object
 5   reason           200 non-null    object
 6   credits          200 non-null    int64
 7   other_credits    200 non-null    object
 8   credit_report    200 non-null    object
 9   marital_status   200 non-null    object
 10  age              200 non-null    int64
 11  employment       200 non-null    object
 12  qualification    200 non-null    object
 13  immigrant        200 non-null    object
 14  residence_since  200 non-null    int64
 15  accommodation    200 non-null    object
 16  estate           200 non-null    object
 17  savings          200 non-null    object
 18  dependents       200 non-null    int64
 19  phone            200 non-null    object
 20  status           200 non-null    object
dtypes: int64(8), object(13)
memory usage: 32.9+ KB
```

- 결측치 없음.

데이터 나누기

In [ ]:
```
# 데이터의 앞 부분 확인, 뒷부분은 tail()로 확인 가능
train.head()
```

Out[ ]:

	id	sum	term	payment	guarantees	reason	credits	other_credits	credit_report	marital_s
0	0	1169	6	4	none	television or radio	2	none	critical account or other credits existing(not...	male
1	1	5951	48	2	none	television or radio	1	none	existing credits paid back duly till now	f divor separa m
2	2	2096	12	2	none	education	1	none	critical account or other credits existing(not...	male
3	3	7882	42	2	guarantor	furniture or equipment	1	none	existing credits paid back duly till now	male
4	4	4870	24	3	none	new car	2	none	delay in paying off in the past	male

5 rows × 22 columns

In [ ]:
```
# 데이터가 많을 때, 생략되어, 이를 전체를 보이도록 하기 위해
# 중간에 생략되는 행과 열을 보이도록 하는 설정
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

In [ ]:
```
train.head()
```

Out[ ]:

	id	sum	term	payment	guarantees	reason	credits	other_credits	credit_report	marital_s
0	0	1169	6	4	none	television or radio	2	none	critical account or other credits existing(not...	male
1	1	5951	48	2	none	television or radio	1	none	existing credits paid back duly till now	f divor separa m
2	2	2096	12	2	none	education	1	none	critical account or other credits existing(not...	male
3	3	7882	42	2	guarantor	furniture or equipment	1	none	existing credits paid back duly till now	male
4	4	4870	24	3	none	new car	2	none	delay in paying off in the past	male

03 데이터 탐색(Exploratory Data Analysis, EDA)

목차로 이동하기

데이터 시각화

In [ ]:

```
print( train['payment'].value_counts() )
print( train['bankruptcy'].value_counts() )
print( train['marital_status'].value_counts() )
print( train['employment'].value_counts() )
print( train['reason'].value_counts() )
print( train['credits'].value_counts() )
```

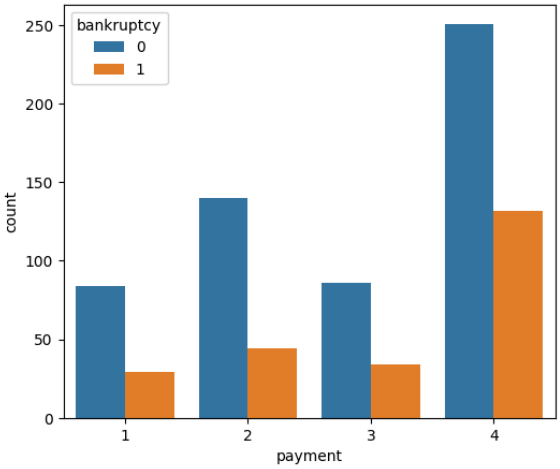
```
payment
4    383
2    184
3    120
1    113
Name: count, dtype: int64
bankruptcy
0    561
1    239
Name: count, dtype: int64
marital_status
male single                                437
female divorced or separated or married    255
male married or widowed                     70
male divorced or separated                  38
Name: count, dtype: int64
employment
1 to 4 y.          275
7+ y.              203
4 to 7 y.          141
less-than 1 y.     133
unemployed          48
Name: count, dtype: int64
reason
television or radio       223
new car                   184
furniture or equipment    144
used car                   81
business                   77
education                  45
repairs                    19
other                      10
household appliances        9
requalification             8
Name: count, dtype: int64
credits
1    513
2    261
3     22
4      4
Name: count, dtype: int64
```

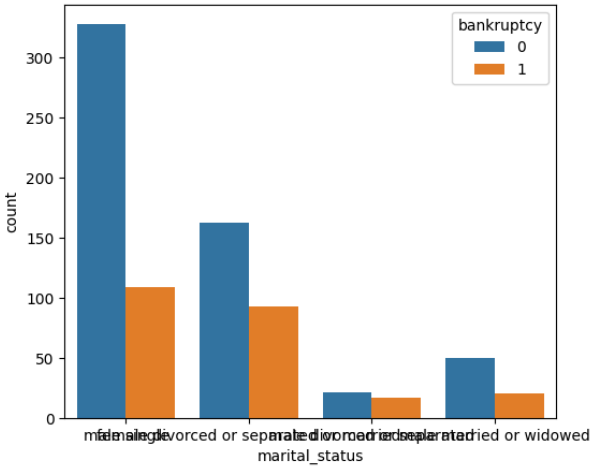
월상환액과 파산여부, 결혼상태와 파산여부는 어떤 관계가 있을까?

In [ ]:
```
plt.figure(figsize=(13,5) )
plt.subplot(1,2,1)
sns.countplot(x="payment", hue='bankruptcy',  data=train)

plt.subplot(1,2,2)
sns.countplot(x="marital_status", hue='bankruptcy',  data=train)
```

Out[ ]:
```
<Axes: xlabel='marital_status', ylabel='count'>
```



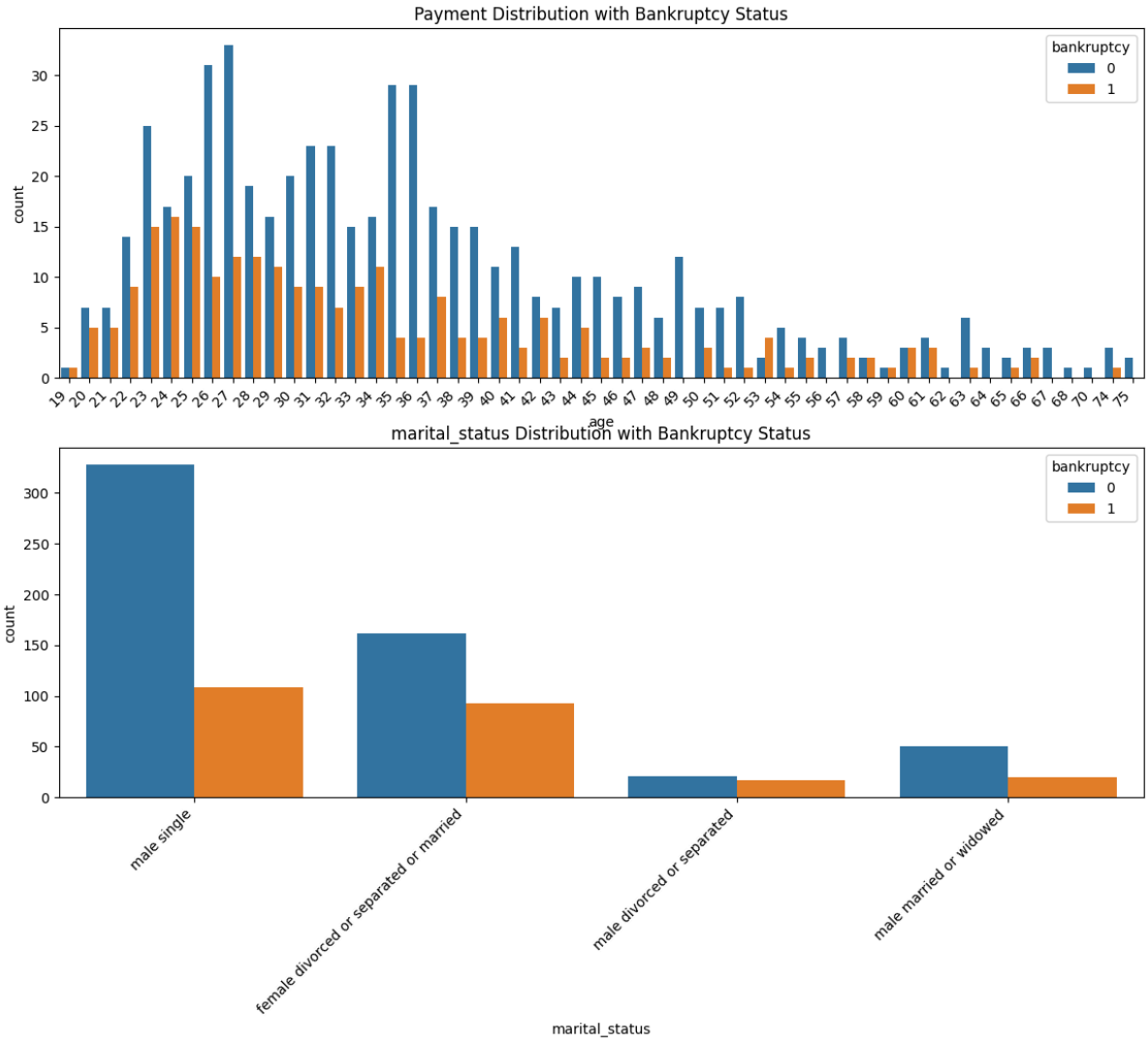


나이와 파산여부는 어떤 관계가 있을까?

In [ ]:
```
plt.figure(figsize=(14,10) )
plt.subplot(2,1,1)
sns.countplot(x="age", hue='bankruptcy',  data=train)
plt.title('Payment Distribution with Bankruptcy Status')
plt.xticks(rotation=45, ha="right")

plt.subplot(2,1,2)
sns.countplot(x="marital_status", hue='bankruptcy',  data=train)
plt.title('marital_status Distribution with Bankruptcy Status')
plt.xticks(rotation=45, ha="right")
```

Out[ ]:
```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'male single'),
  Text(1, 0, 'female divorced or separated or married'),
  Text(2, 0, 'male divorced or separated'),
  Text(3, 0, 'male married or widowed')])
```



In [ ]:
```
# 결혼 상태별 파산 여부 카운트
marital_status_counts = train.groupby('marital_status')['bankruptcy'].value_counts()
print(marital_status_counts)
```

```
bankruptcy                                0    1
marital_status
female divorced or separated or married  162   93
male divorced or separated                21   17
male married or widowed                   50   20
male single                              328  109
```

- payment가 1 또는 3일 때는 파산 여부에 큰 영향을 주지 않는 것으로 보입니다.
- 남성이고 이혼했거나 별거 중일때 파산의 비율의 높았다.
- 남성이 싱글일 때, 상대적으로 파산 확률이 낮았다.

plotly를 활용한 시각화

In [ ]:
```
reason_order = train['reason'].value_counts().index.tolist()   # 대출 사유
bankruptcy_order = train['bankruptcy'].value_counts().index.tolist()  # 파산 여부
```

어떠한 대출사유가 있을까?

In [ ]:
```
import plotly.express as px
```

```
fig = px.bar(train, x='reason', color='bankruptcy', barmode='group',
             category_orders={'reason': reason_order, 'bankruptcy': ['0', '1']},
             title='Reason Distribution with Bankruptcy Status')
fig.update_layout(xaxis={'categoryorder': 'total descending'}, xaxis_tickangle=-45)
fig.show()
```

In [ ]:
```
train.reason.value_counts()
```

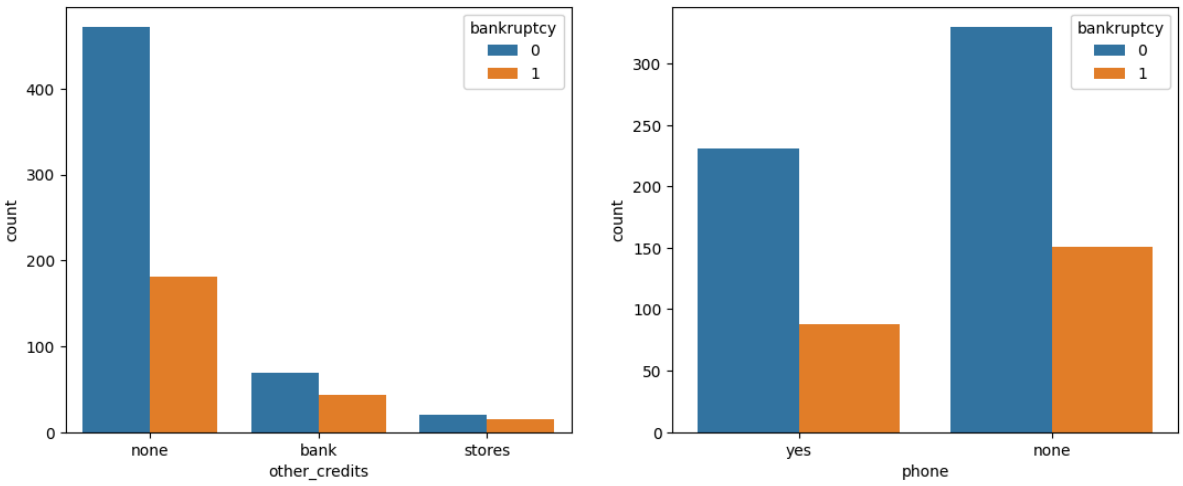
Out[ ]:
```
reason
television or radio       223
new car                   184
furniture or equipment    144
used car                   81
business                   77
education                  45
repairs                    19
other                      10
household appliances        9
requalification             8
Name: count, dtype: int64
```

다른 대출 상태와 전화소유여부는 파산여부는 어떤 관계가 있을까?

In [ ]:
```
plt.figure(figsize=(13,5) )
plt.subplot(1,2,1)
sns.countplot(x="other_credits", hue="bankruptcy",  data=train)
```

```
plt.subplot(1,2,2)
sns.countplot(x="phone", hue="bankruptcy", data=train)
```

Out[ ]:
```
<Axes: xlabel='phone', ylabel='count'>
```

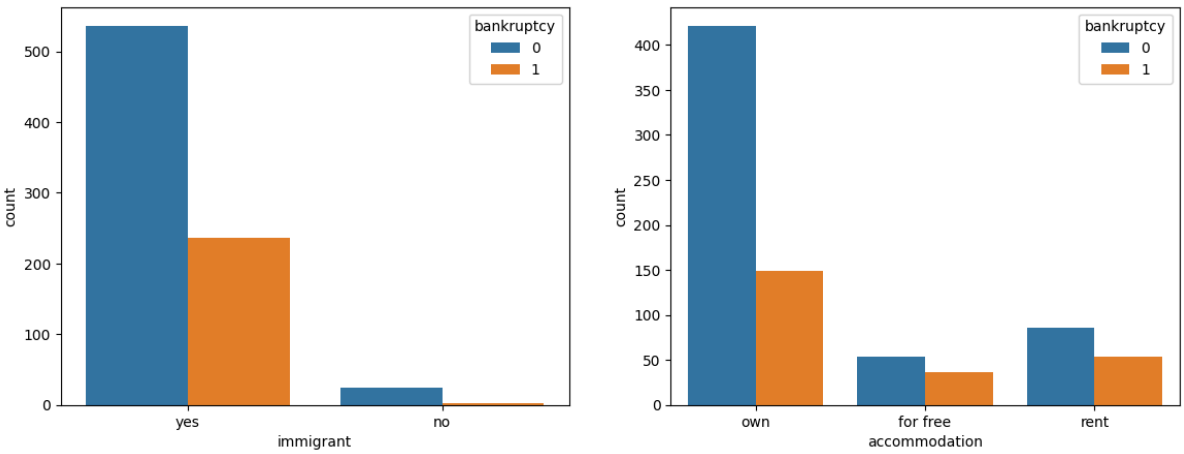


이민자여부, 주거형태는 파산여부는 어떤 관계가 있을까?

In [ ]:
```
plt.figure(figsize=(14,5) )
plt.subplot(1,2,1)
sns.countplot(x="immigrant", hue="bankruptcy",  data=train)


plt.subplot(1,2,2)
sns.countplot(x="accommodation", hue="bankruptcy", data=train)
```

Out[ ]:
```
<Axes: xlabel='accommodation', ylabel='count'>
```



In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['id', 'sum', 'term', 'payment', 'guarantees', 'reason', 'credits',
       'other_credits', 'credit_report', 'marital_status', 'age', 'employment',
       'qualification', 'immigrant', 'residence_since', 'accommodation',
       'estate', 'savings', 'dependents', 'phone', 'status', 'bankruptcy'],
      dtype='object')
```

In [ ]:
```
train.head()
```

Out[ ]:

	id	sum	term	payment	guarantees	reason	credits	other_credits	credit_report	marital_s
0	0	1169	6	4	none	television or radio	2	none	critical account or other credits existing(not...	male
1	1	5951	48	2	none	television or radio	1	none	existing credits paid back duly till now	f divor separa m
2	2	2096	12	2	none	education	1	none	critical account or other credits existing(not...	male
3	3	7882	42	2	guarantor	furniture or equipment	1	none	existing credits paid back duly till now	male
4	4	4870	24	3	none	new car	2	none	delay in paying off in the past	male

04 데이터 전처리 및 데이터 나누기

목차로 이동하기

In [ ]:
```
print( train.guarantees.unique() )
print( train.reason.unique()  )
print( train.other_credits.unique()  )
print( train.marital_status.unique()  )
print( train.qualification.unique()  )
print( train.immigrant.unique()  )
print( train.accommodation.unique()  )
print( train.estate.unique()  )
print( train.phone.unique()  )
```

```
['none' 'guarantor' 'co-applicant']
['television or radio' 'education' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'repairs' 'other'
 'requalification']
['none' 'bank' 'stores']
['male single' 'female divorced or separated or married'
 'male divorced or separated' 'male married or widowed']
['skilled employee' 'unskilled resident'
 'management or self-employed or highly qualified employee'
 'unemployed or unskilled non-resident']
['yes' 'no']
['own' 'for free' 'rent']
['real estate' 'building society savings agreement or life insurance'
 'unknown or no property' 'car or other']
['yes' 'none']
```

In [ ]:
```
### 데이터 전처리
sub = pd.read_csv("/kaggle/input/bankruptcy-risk-prediction/submission_example.csv")
train = pd.read_csv("/kaggle/input/bankruptcy-risk-prediction/train.csv")
test = pd.read_csv("/kaggle/input/bankruptcy-risk-prediction/test.csv")
```

```
train.shape, test.shape, sub.shape
```

Out[ ]:
```
((800, 22), (200, 21), (200, 2))
```

각 컬럼별 데이터 값들 확인하기

In [ ]:
```
### 전체 컬럼 unique() 확인하기
### 범주형 - 범주가 50개 이하만 출력하기
def columns_print(dataset):
    for one in dataset.columns:
        # print(one)
        col_length = len( dataset[one].unique() )
        if col_length <= 50:
            print("colname : ", one)
            print(dataset[one].unique())

columns_print(train)
```

```
colname :  term
[ 6 48 12 42 24 36 30 15  9 10  7 60 18 45 11 27  8 54 20 14 33 21 16  4
 47 13 22 39 28  5 26 72 40]
colname :  payment
[4 2 3 1]
colname :  guarantees
['none' 'guarantor' 'co-applicant']
colname :  reason
['television or radio' 'education' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'repairs' 'other'
 'requalification']
colname :  credits
[2 1 3 4]
colname :  other_credits
['none' 'bank' 'stores']
colname :  credit_report
['critical account or other credits existing(not at this bank)'
 'existing credits paid back duly till now'
 'delay in paying off in the past' 'no credits or all paid'
 'all credits at this bank paid back duly']
colname :  marital_status
['male single' 'female divorced or separated or married'
 'male divorced or separated' 'male married or widowed']
colname :  employment
['7+ y.' '1 to 4 y.' '4 to 7 y.' 'unemployed' 'less-than 1 y.']
colname :  qualification
['skilled employee' 'unskilled resident'
 'management or self-employed or highly qualified employee'
 'unemployed or unskilled non-resident']
colname :  immigrant
['yes' 'no']
colname :  residence_since
[4 2 3 1]
colname :  accommodation
['own' 'for free' 'rent']
colname :  estate
['real estate' 'building society savings agreement or life insurance'
 'unknown or no property' 'car or other']
colname :  savings
['unknown or no savings account' 'less-than 100 cu' '500 to 1000 cu'
 'greater-than 1000 cu' '100 to 500 cu']
colname :  dependents
[1 2]
colname :  phone
['yes' 'none']
colname :  status
['less-than 0 cu' '0 to 200 cu' 'no checking account'
 'greater-than 200 cu or salary assignments for at least 1 year']
colname :  bankruptcy
[0 1]
```

In [ ]:

```python
### 인코딩 map 내용 만들기
def map_con(dataset, col_names):
    dict_map = {}
    cnt = 0
    unique_value = dataset[col_names].unique()
    print( unique_value )

    for one in unique_value:
        # print(one)
        dict_map[one] = cnt
        cnt += 1
    # print(dict_map)
    return dict_map
```

```
map_con(train, "reason")
```

Out[ ]:
```
['television or radio' 'education' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'repairs' 'other'
 'requalification']
{'television or radio': 0,
 'education': 1,
 'furniture or equipment': 2,
 'new car': 3,
 'used car': 4,
 'business': 5,
 'household appliances': 6,
 'repairs': 7,
 'other': 8,
 'requalification': 9}
```

In [ ]:
```
columns_print(train)
```

```
colname :  term
[ 6 48 12 42 24 36 30 15  9 10  7 60 18 45 11 27  8 54 20 14 33 21 16  4
 47 13 22 39 28  5 26 72 40]
colname :  payment
[4 2 3 1]
colname :  guarantees
['none' 'guarantor' 'co-applicant']
colname :  reason
['television or radio' 'education' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'repairs' 'other'
 'requalification']
colname :  credits
[2 1 3 4]
colname :  other_credits
['none' 'bank' 'stores']
colname :  credit_report
['critical account or other credits existing(not at this bank)'
 'existing credits paid back duly till now'
 'delay in paying off in the past' 'no credits or all paid'
 'all credits at this bank paid back duly']
colname :  marital_status
['male single' 'female divorced or separated or married'
 'male divorced or separated' 'male married or widowed']
colname :  employment
['7+ y.' '1 to 4 y.' '4 to 7 y.' 'unemployed' 'less-than 1 y.']
colname :  qualification
['skilled employee' 'unskilled resident'
 'management or self-employed or highly qualified employee'
 'unemployed or unskilled non-resident']
colname :  immigrant
['yes' 'no']
colname :  residence_since
[4 2 3 1]
colname :  accommodation
['own' 'for free' 'rent']
colname :  estate
['real estate' 'building society savings agreement or life insurance'
 'unknown or no property' 'car or other']
colname :  savings
['unknown or no savings account' 'less-than 100 cu' '500 to 1000 cu'
 'greater-than 1000 cu' '100 to 500 cu']
colname :  dependents
[1 2]
colname :  phone
['yes' 'none']
colname :  status
['less-than 0 cu' '0 to 200 cu' 'no checking account'
 'greater-than 200 cu or salary assignments for at least 1 year']
colname :  bankruptcy
[0 1]
```

데이터 전처리 - 범주형 문자열을 숫자로 변경

- 같은 내용을 두 번 실행할 경우, 셀의 실행 결과는 우리가 예상하는 결과값이 보여지지 않을 수 있다.

In [ ]:
```
# 라벨 인코딩
train['guarantees'] = train.guarantees.map( {'none':0, 'guarantor':1, 'co-applicant'
train.guarantees.unique()
```

Out[ ]:
```
array([0, 1, 2])
```

In [ ]:
```
# 라벨 인코딩
# 문자를 숫자로 변경
print(train.reason.unique())
reason_dict = map_con(train, "reason")
print(reason_dict)
```

```
['television or radio' 'education' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'repairs' 'other'
 'requalification']
['television or radio' 'education' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'repairs' 'other'
 'requalification']
{'television or radio': 0, 'education': 1, 'furniture or equipment': 2, 'new car':
3, 'used car': 4, 'business': 5, 'household appliances': 6, 'repairs': 7, 'other':
8, 'requalification': 9}
```

In [ ]:
```
train['reason'] = train.reason.map( {'television or radio': 0,
                        'education': 1, 'furniture or equipment': 2, 'new car': 3,
                        'used car': 4, 'business': 5, 'household appliances': 6,
                        'repairs': 7, 'other': 8, 'requalification': 9} )
train.reason.unique()
```

Out[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
#print(train['other_credits'].unique())
dat_dict = map_con(train, "other_credits")
#print(reason_dict)
train['other_credits'] = train['other_credits'].map( dat_dict )
train['other_credits'].unique()
```

```
['none' 'bank' 'stores']
```

Out[ ]:
```
array([0, 1, 2])
```

test 데이터 셋 변환

- guarantees, reason, other_credits

In [ ]:
```
test['guarantees'] = test.guarantees.map( {'none':0, 'guarantor':1, 'co-applicant':2
test.guarantees.unique()

reason_dict = map_con(test, "reason")
print(reason_dict)
test['reason'] = test.reason.map( reason_dict )
test.reason.unique()
```

```
['education' 'television or radio' 'furniture or equipment' 'new car'
 'used car' 'business' 'household appliances' 'other' 'repairs'
 'requalification']
{'education': 0, 'television or radio': 1, 'furniture or equipment': 2, 'new car':
3, 'used car': 4, 'business': 5, 'household appliances': 6, 'other': 7, 'repairs':
8, 'requalification': 9}
```

Out[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:
```
dat_dict = map_con(test, "other_credits")
#print(reason_dict)
test['other_credits'] = test['other_credits'].map( dat_dict )
test['other_credits'].unique()
```

```
['none' 'bank' 'stores']
```

Out[ ]:
```
array([0, 1, 2])
```

In [ ]:
```
#  employment, qualification, immigrant, accommodation, savings, phone
```

In [ ]:
```
dat_dict = map_con(train, "credit_report")
train['credit_report'] = train['credit_report'].map( dat_dict )
train['credit_report'].unique()

dat_dict = map_con(train, "marital_status")
train['marital_status'] = train['marital_status'].map( dat_dict )
train['marital_status'].unique()

dat_dict = map_con(train, "employment")
train['employment'] = train['employment'].map( dat_dict )

dat_dict = map_con(train, "qualification")
train['qualification'] = train['qualification'].map( dat_dict )

dat_dict = map_con(train, "immigrant")
train['immigrant'] = train['immigrant'].map( dat_dict )

dat_dict = map_con(train, "accommodation")
train['accommodation'] = train['accommodation'].map( dat_dict )

dat_dict = map_con(train, "savings")
train['savings'] = train['savings'].map( dat_dict )

dat_dict = map_con(train, "phone")
train['phone'] = train['phone'].map( dat_dict )

dat_dict = map_con(train, "estate")
train['estate'] = train['estate'].map( dat_dict )

dat_dict = map_con(train, "credit_report")
train['credit_report'] = train['credit_report'].map( dat_dict )

dat_dict = map_con(train, "status")
train['status'] = train['status'].map( dat_dict )

train.head()
```

```
['critical account or other credits existing(not at this bank)'
 'existing credits paid back duly till now'
 'delay in paying off in the past' 'no credits or all paid'
 'all credits at this bank paid back duly']
['male single' 'female divorced or separated or married'
 'male divorced or separated' 'male married or widowed']
['7+ y.' '1 to 4 y.' '4 to 7 y.' 'unemployed' 'less-than 1 y.']
['skilled employee' 'unskilled resident'
 'management or self-employed or highly qualified employee'
 'unemployed or unskilled non-resident']
['yes' 'no']
['own' 'for free' 'rent']
['unknown or no savings account' 'less-than 100 cu' '500 to 1000 cu'
 'greater-than 1000 cu' '100 to 500 cu']
['yes' 'none']
['real estate' 'building society savings agreement or life insurance'
 'unknown or no property' 'car or other']
[0 1 2 3 4]
['less-than 0 cu' '0 to 200 cu' 'no checking account'
 'greater-than 200 cu or salary assignments for at least 1 year']
```

Out[ ]:

	id	sum	term	payment	guarantees	reason	credits	other_credits	credit_report	marital_stat
0	0	1169	6	4	0	0	2	0	0	
1	1	5951	48	2	0	0	1	0	1	
2	2	2096	12	2	0	1	1	0	0	
3	3	7882	42	2	1	2	1	0	1	
4	4	4870	24	3	0	3	2	0	2	

In [ ]:

```
dat_dict = map_con(test, "credit_report")
test['credit_report'] = test['credit_report'].map( dat_dict )
test['credit_report'].unique()

dat_dict = map_con(test, "marital_status")
test['marital_status'] = test['marital_status'].map( dat_dict )
test['marital_status'].unique()

dat_dict = map_con(test, "employment")
test['employment'] = test['employment'].map( dat_dict )

dat_dict = map_con(test, "qualification")
test['qualification'] = test['qualification'].map( dat_dict )

dat_dict = map_con(test, "immigrant")
test['immigrant'] = test['immigrant'].map( dat_dict )

dat_dict = map_con(test, "accommodation")
test['accommodation'] = test['accommodation'].map( dat_dict )

dat_dict = map_con(test, "savings")
test['savings'] = test['savings'].map( dat_dict )

dat_dict = map_con(test, "phone")
test['phone'] = test['phone'].map( dat_dict )

dat_dict = map_con(test, "estate")
test['estate'] = test['estate'].map( dat_dict )

dat_dict = map_con(test, "credit_report")
test['credit_report'] = test['credit_report'].map( dat_dict )

dat_dict = map_con(test, "status")
test['status'] = test['status'].map( dat_dict )

test.head()
```

```
['critical account or other credits existing(not at this bank)'
 'existing credits paid back duly till now'
 'all credits at this bank paid back duly' 'no credits or all paid'
 'delay in paying off in the past']
['male single' 'female divorced or separated or married'
 'male married or widowed' 'male divorced or separated']
['7+ y.' 'unemployed' '4 to 7 y.' 'less-than 1 y.' '1 to 4 y.']
['skilled employee' 'unskilled resident'
 'unemployed or unskilled non-resident'
 'management or self-employed or highly qualified employee']
['yes' 'no']
['for free' 'rent' 'own']
['less-than 100 cu' 'unknown or no savings account' '500 to 1000 cu'
 '100 to 500 cu' 'greater-than 1000 cu']
['none' 'yes']
['unknown or no property' 'real estate'
 'building society savings agreement or life insurance' 'car or other']
[0 1 2 3 4]
['no checking account' '0 to 200 cu' 'less-than 0 cu'
 'greater-than 200 cu or salary assignments for at least 1 year']
```

Out[ ]:

	id	sum	term	payment	guarantees	reason	credits	other_credits	credit_report	marital_sta
0	800	1597	24	4	0	0	2	0	0	
1	801	1795	18	3	1	1	2	1	0	
2	802	4272	20	1	0	2	2	0	0	
3	803	976	12	4	0	1	2	0	0	
4	804	7472	12	1	0	3	1	0	1	

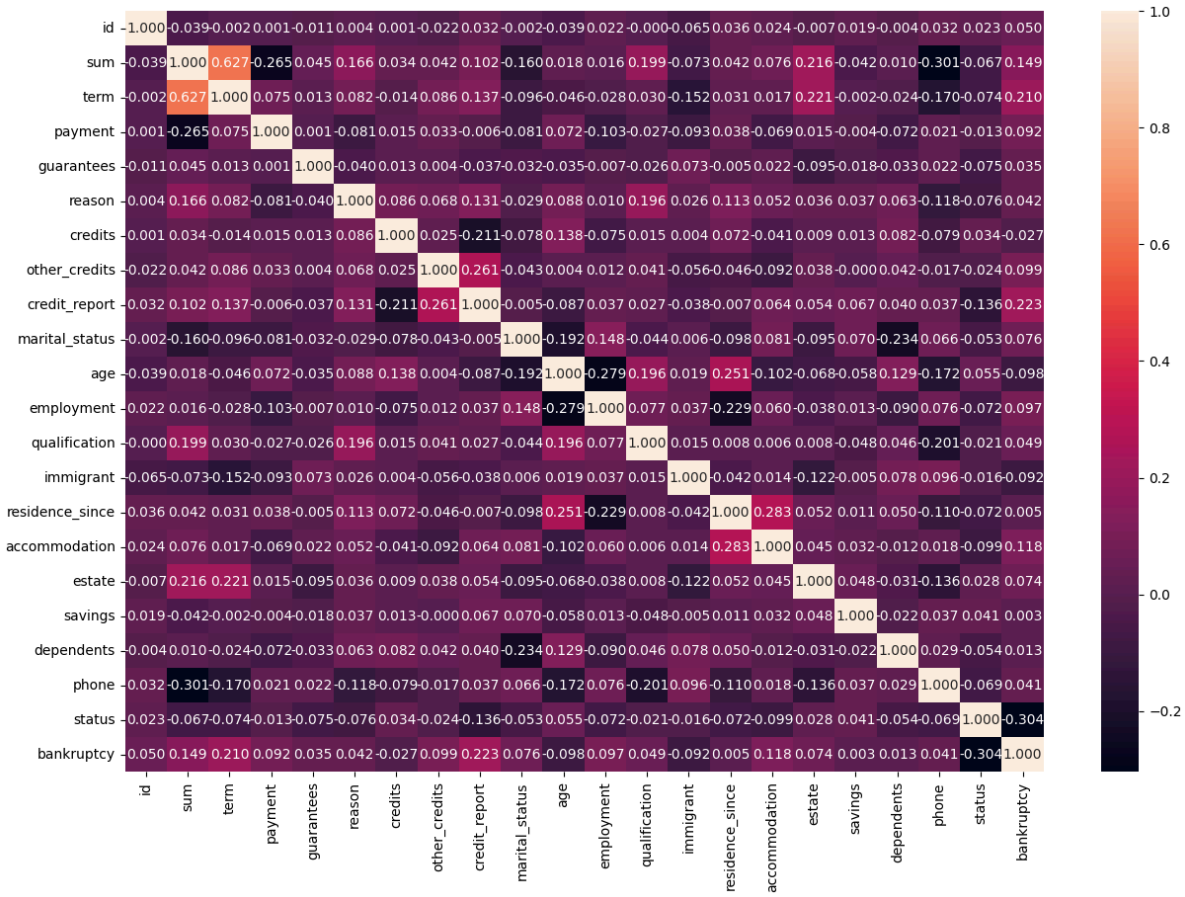
In [ ]:
```
train.columns
```

Out[ ]:
```
Index(['id', 'sum', 'term', 'payment', 'guarantees', 'reason', 'credits',
       'other_credits', 'credit_report', 'marital_status', 'age', 'employment',
       'qualification', 'immigrant', 'residence_since', 'accommodation',
       'estate', 'savings', 'dependents', 'phone', 'status', 'bankruptcy'],
      dtype='object')
```

변수별 상관관계를 히트맵과 상관계수를 통해 확인

In [ ]:
```
plt.figure(figsize=(15,10))
sns.heatmap(train.corr(), annot=True, fmt=".3f")
```

Out[ ]:
```
<Axes: >
```



데이터 나누기

- 예측을 위해 사용할 변수 선택하기
- 예측하고자 하는 y값을 선택하기
- 학습용, 테스트용 데이터 셋으로 나누기
- 입력(X)와 출력(y)로 나누기

In [ ]:
```
#  변경2 : 0.05 이하 제외 'guarantees', 'credits', 'qualification', 'residence_since
sel = ['sum', 'term', 'payment', 'reason', 'other_credits', 'credit_report',
       'marital_status', 'age', 'employment', 'immigrant',  'accommodation',
       'estate', 'status']

X = train[sel]
y = train['bankruptcy']

last_test = test[sel]

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state = 0)
```

05 머신러닝 모델 구축/학습/평가

목차로 이동하기

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
```

모델 구축 및 학습, 평가

- KNN
- DecisionTree(의사결정트리)
- RandonForest(랜덤포레스트)
- XGBoost

- model.score() 함수를 이용하여 모델을 평가할 수 있음.
- 여기에서 model.score()는 모델이 예측의 정확도가 어느정도 되는가를 나타낸다. 0.95의 경우 95%의 정확도

In [ ]:
```
model1 = KNeighborsClassifier()
model1.fit(X_train, y_train)
print("model1 학습용 정확도 : ", model1.score(X_train,y_train) )
print("model1 테스트용 정확도 : ", model1.score(X_test,y_test) )
print()

model2 = DecisionTreeClassifier(max_depth=3, random_state=0)
model2.fit(X_train, y_train)
print("model2 학습용 정확도 : ", model2.score(X_train,y_train) )
print("model2 테스트용 정확도 : ", model2.score(X_test,y_test) )
print()

model3 = RandomForestClassifier(max_depth=5, random_state=0)
model3.fit(X_train, y_train)
print("model3 학습용 정확도 : ", model3.score(X_train,y_train) )
print("model3 테스트용 정확도 : ", model3.score(X_test,y_test) )
print()

model4 = xgb.XGBClassifier(colsample_bytree = 0.2, # 각나무마다 사용하는 feature 비율
                           learning_rate = 0.05,
                           max_depth = 3,
                           alpha = 0.1,
                           n_estimators = 1000)

model4.fit(X_train, y_train)
print("model4 학습용 정확도 : ", model4.score(X_train,y_train) )
print("model5 테스트용 정확도 : ", model4.score(X_test,y_test) )
print()
```

```
model1 학습용 정확도 :  0.7517857142857143
model1 테스트용 정확도 :  0.6666666666666666

model2 학습용 정확도 :  0.7732142857142857
model2 테스트용 정확도 :  0.7083333333333334

model3 학습용 정확도 :  0.8285714285714286
model3 테스트용 정확도 :  0.725

model4 학습용 정확도 :  0.9375
model5 테스트용 정확도 :  0.7375
```

가장 좋은 모델을 선택

In [ ]:
```
model3 = RandomForestClassifier(max_depth=5, random_state=0)
model3.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=5, random_state=0)
```

In [ ]:
```
model4 = xgb.XGBClassifier(colsample_bytree = 0.2, # 각나무마다 사용하는 feature 비율
                           learning_rate = 0.05,
                           max_depth = 3,
                           alpha = 0.1,
                           n_estimators = 1000)
model4.fit(X_train, y_train)
```

Out[ ]:
```
XGBClassifier
XGBClassifier(alpha=0.1, base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.2, device=None, early_stopping_rounds=
None,
              enable_categorical=False, eval_metric=None, feature_types
=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin
=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
```

예측을 할때, 확률로 예측해야 한다면 **predict_proba()** 함수 이용하여 확률 예측 가능

In [ ]:
```
y_pred=model4.predict_proba(last_test)
y_pred
```

Out[ ]:
```
array([[0.72907174, 0.27092826],
       [0.6507627 , 0.34923732],
       [0.9398224 , 0.06017762],
       [0.43901914, 0.56098086],
       [0.7172885 , 0.28271154],
       [0.8583744 , 0.14162557],
       [0.8928971 , 0.10710287],
       [0.7128705 , 0.2871295 ],
       [0.79603755, 0.20396242],
       [0.45203513, 0.5479649 ],
       [0.43083948, 0.5691605 ],
       [0.75797135, 0.24202864],
       [0.9880464 , 0.01195362],
       [0.3486606 , 0.6513394 ],
       [0.9625053 , 0.03749474],
       [0.05601865, 0.94398135],
       [0.7575426 , 0.24245737],
       [0.9079186 , 0.09208142],
       [0.24266607, 0.75733393],
       [0.7496728 , 0.25032723],
       [0.25798196, 0.74201804],
       [0.7048456 , 0.2951544 ],
       [0.6376219 , 0.36237815],
       [0.892068  , 0.107932  ],
       [0.66136587, 0.33863416],
       [0.49605894, 0.50394106],
       [0.88621813, 0.11378188],
       [0.3379503 , 0.6620497 ],
       [0.98128074, 0.01871928],
       [0.1536482 , 0.8463518 ],
       [0.11175066, 0.88824934],
       [0.34582537, 0.6541746 ],
       [0.36688185, 0.63311815],
       [0.5192131 , 0.48078695],
       [0.95318204, 0.04681797],
       [0.34635592, 0.6536441 ],
       [0.02592844, 0.97407156],
       [0.8954129 , 0.10458711],
       [0.9968895 , 0.00311052],
       [0.4735453 , 0.5264547 ],
       [0.73592854, 0.26407146],
       [0.13258827, 0.86741173],
       [0.33758968, 0.6624103 ],
       [0.29107296, 0.70892704],
       [0.3528363 , 0.6471637 ],
       [0.86945844, 0.13054158],
       [0.66619796, 0.33380204],
       [0.01297778, 0.9870222 ],
       [0.937717  , 0.06228301],
       [0.81265205, 0.18734793],
       [0.71193945, 0.28806055],
       [0.9676989 , 0.03230112],
       [0.74158823, 0.2584118 ],
       [0.91811496, 0.08188505],
       [0.09169865, 0.90830135],
       [0.08296579, 0.9170342 ],
       [0.39358133, 0.60641867],
       [0.669458  , 0.33054203],
       [0.8680058 , 0.1319942 ],
       [0.94738835, 0.05261167],
       [0.9194691 , 0.08053087],
       [0.03507715, 0.96492285],
       [0.97059876, 0.02940124],
       [0.38138658, 0.6186134 ],
```

```
[0.952471  , 0.04752899],
[0.64097476, 0.3590252 ],
[0.88766634, 0.11233363],
[0.4936204 , 0.5063796 ],
[0.20077622, 0.7992238 ],
[0.789636  , 0.21036398],
[0.25821382, 0.7417862 ],
[0.9859377 , 0.01406229],
[0.9657875 , 0.03421251],
[0.15415412, 0.8458459 ],
[0.9070599 , 0.09294012],
[0.6580944 , 0.34190562],
[0.7960796 , 0.20392041],
[0.20582902, 0.794171  ],
[0.9353413 , 0.06465872],
[0.700299  , 0.29970095],
[0.94470257, 0.05529745],
[0.8360417 , 0.16395833],
[0.77886415, 0.22113587],
[0.81300646, 0.18699354],
[0.8266995 , 0.17330053],
[0.6589769 , 0.34102312],
[0.81855696, 0.18144304],
[0.05541855, 0.94458145],
[0.11782056, 0.88217944],
[0.80378544, 0.19621457],
[0.68483293, 0.31516707],
[0.53494   , 0.46506   ],
[0.89969426, 0.10030574],
[0.83172977, 0.16827026],
[0.2052508 , 0.7947492 ],
[0.8071377 , 0.1928623 ],
[0.6943245 , 0.30567554],
[0.71907884, 0.28092116],
[0.89508784, 0.10491216],
[0.92839575, 0.07160422],
[0.8480721 , 0.1519279 ],
[0.49016672, 0.5098333 ],
[0.9201378 , 0.07986216],
[0.40850484, 0.59149516],
[0.38981283, 0.6101872 ],
[0.9305857 , 0.06941432],
[0.9848031 , 0.01519689],
[0.2650345 , 0.7349655 ],
[0.6999742 , 0.3000258 ],
[0.50258416, 0.49741584],
[0.02247584, 0.97752416],
[0.6837262 , 0.31627384],
[0.5132478 , 0.4867522 ],
[0.84261876, 0.15738122],
[0.8636584 , 0.13634159],
[0.05712795, 0.94287205],
[0.97885853, 0.02114148],
[0.91101146, 0.08898856],
[0.8318104 , 0.16818957],
[0.9408888 , 0.05911118],
[0.14715874, 0.85284126],
[0.08242601, 0.917574  ],
[0.5888095 , 0.4111905 ],
[0.77593786, 0.22406214],
[0.7145891 , 0.28541088],
[0.44970983, 0.55029017],
[0.73702216, 0.2629778 ],
[0.9535413 , 0.04645873],
```

```
[0.13922262, 0.8607774 ],
[0.47087216, 0.52912784],
[0.954758  , 0.04524201],
[0.26798666, 0.73201334],
[0.08472341, 0.9152766 ],
[0.421075  , 0.578925  ],
[0.67019147, 0.32980853],
[0.34632832, 0.6536717 ],
[0.2245596 , 0.7754404 ],
[0.808923  , 0.19107702],
[0.61439514, 0.38560486],
[0.7986399 , 0.20136012],
[0.5844757 , 0.4155243 ],
[0.83445084, 0.16554916],
[0.1767109 , 0.8232891 ],
[0.93993384, 0.06006617],
[0.88446075, 0.11553926],
[0.2477715 , 0.7522285 ],
[0.9395333 , 0.06046671],
[0.58578426, 0.41421574],
[0.12525266, 0.87474734],
[0.8090607 , 0.19093928],
[0.61020255, 0.38979748],
[0.87464076, 0.12535924],
[0.96394384, 0.03605618],
[0.04060435, 0.95939565],
[0.61313593, 0.3868641 ],
[0.6499446 , 0.3500554 ],
[0.926943  , 0.07305699],
[0.80673885, 0.19326116],
[0.89410627, 0.10589376],
[0.9093168 , 0.09068321],
[0.92144144, 0.07855853],
[0.02443779, 0.9755622 ],
[0.39179456, 0.60820544],
[0.49877286, 0.50122714],
[0.7250537 , 0.27494627],
[0.23156619, 0.7684338 ],
[0.5832999 , 0.4167001 ],
[0.3043865 , 0.6956135 ],
[0.35223782, 0.6477622 ],
[0.98580927, 0.01419071],
[0.81918174, 0.18081826],
[0.31204224, 0.68795776],
[0.8624966 , 0.13750339],
[0.9198062 , 0.08019383],
[0.08887774, 0.91112226],
[0.46324188, 0.5367581 ],
[0.89727175, 0.10272825],
[0.25214905, 0.74785095],
[0.28715932, 0.7128407 ],
[0.15655035, 0.84344965],
[0.82878506, 0.17121497],
[0.1332137 , 0.8667863 ],
[0.2847364 , 0.7152636 ],
[0.94365156, 0.05634846],
[0.85416836, 0.14583164],
[0.8684711 , 0.13152891],
[0.3914917 , 0.6085083 ],
[0.8978692 , 0.10213074],
[0.98815304, 0.01184694],
[0.6575978 , 0.34240222],
[0.7557153 , 0.2442847 ],
[0.22057396, 0.77942604],
```

```
       [0.8961587 , 0.10384128],
       [0.88218445, 0.11781555],
       [0.62537444, 0.37462553],
       [0.6592953 , 0.3407047 ],
       [0.94973135, 0.05026868],
       [0.11658889, 0.8834111 ],
       [0.5839323 , 0.41606775],
       [0.8019027 , 0.19809727]], dtype=float32)
```

In [ ]:
```
y_pred[:, 1]  # 행별 (0일 확률, 1일 확률) 값이 나옴. 이중에 두번째 것을 가져와 제출
```

Out[ ]:
```
array([0.27092826, 0.34923732, 0.06017762, 0.56098086, 0.28271154,
       0.14162557, 0.10710287, 0.2871295 , 0.20396242, 0.5479649 ,
       0.5691605 , 0.24202864, 0.01195362, 0.6513394 , 0.03749474,
       0.94398135, 0.24245737, 0.09208142, 0.75733393, 0.25032723,
       0.74201804, 0.2951544 , 0.36237815, 0.107932  , 0.33863416,
       0.50394106, 0.11378188, 0.6620497 , 0.01871928, 0.8463518 ,
       0.88824934, 0.6541746 , 0.63311815, 0.48078695, 0.04681797,
       0.6536441 , 0.97407156, 0.10458711, 0.00311052, 0.5264547 ,
       0.26407146, 0.86741173, 0.6624103 , 0.70892704, 0.6471637 ,
       0.13054158, 0.33380204, 0.9870222 , 0.06228301, 0.18734793,
       0.28806055, 0.03230112, 0.2584118 , 0.08188505, 0.90830135,
       0.9170342 , 0.60641867, 0.33054203, 0.1319942 , 0.05261167,
       0.08053087, 0.96492285, 0.02940124, 0.6186134 , 0.04752899,
       0.3590252 , 0.11233363, 0.5063796 , 0.7992238 , 0.21036398,
       0.7417862 , 0.01406229, 0.03421251, 0.8458459 , 0.09294012,
       0.34190562, 0.20392041, 0.794171  , 0.06465872, 0.29970095,
       0.05529745, 0.16395833, 0.22113587, 0.18699354, 0.17330053,
       0.34102312, 0.18144304, 0.94458145, 0.88217944, 0.19621457,
       0.31516707, 0.46506   , 0.10030574, 0.16827026, 0.7947492 ,
       0.1928623 , 0.30567554, 0.28092116, 0.10491216, 0.07160422,
       0.1519279 , 0.5098333 , 0.07986216, 0.59149516, 0.6101872 ,
       0.06941432, 0.01519689, 0.7349655 , 0.3000258 , 0.49741584,
       0.97752416, 0.31627384, 0.4867522 , 0.15738122, 0.13634159,
       0.94287205, 0.02114148, 0.08898856, 0.16818957, 0.05911118,
       0.85284126, 0.917574  , 0.4111905 , 0.22406214, 0.28541088,
       0.55029017, 0.2629778 , 0.04645873, 0.8607774 , 0.52912784,
       0.04524201, 0.73201334, 0.9152766 , 0.578925  , 0.32980853,
       0.6536717 , 0.7754404 , 0.19107702, 0.38560486, 0.20136012,
       0.4155243 , 0.16554916, 0.8232891 , 0.06006617, 0.11553926,
       0.7522285 , 0.06046671, 0.41421574, 0.87474734, 0.19093928,
       0.38979748, 0.12535924, 0.03605618, 0.95939565, 0.3868641 ,
       0.3500554 , 0.07305699, 0.19326116, 0.10589376, 0.09068321,
       0.07855853, 0.9755622 , 0.60820544, 0.50122714, 0.27494627,
       0.7684338 , 0.4167001 , 0.6956135 , 0.6477622 , 0.01419071,
       0.18081826, 0.68795776, 0.13750339, 0.08019383, 0.91112226,
       0.5367581 , 0.10272825, 0.74785095, 0.7128407 , 0.84344965,
       0.17121497, 0.8667863 , 0.7152636 , 0.05634846, 0.14583164,
       0.13152891, 0.6085083 , 0.10213074, 0.01184694, 0.34240222,
       0.2442847 , 0.77942604, 0.10384128, 0.11781555, 0.37462553,
       0.3407047 , 0.05026868, 0.8834111 , 0.41606775, 0.19809727],
      dtype=float32)
```

In [ ]:
```
sub.columns
```

Out[ ]:
```
Index(['id', 'proba'], dtype='object')
```

In [ ]:
```
# 제출용 컬럼 proba에 예측 확률 업데이트
sub['proba'] = y_pred[:, 1]
```

In [ ]:
```
# 제출용 파일 생성
sub.to_csv("sub_xg_01.csv", index=False)
```

0.35983 - 앙상블 -RandomForest

```
sel = ['sum', 'term', 'payment', 'guarantees', 'reason', 'credits',
       'other_credits', 'credit_report', 'marital_status', 'age',
'employment',
       'qualification', 'immigrant', 'residence_since',
'accommodation',
       'estate', 'savings', 'dependents', 'phone', 'status']
```

Score: 0.44097, xgb - sub_xg_01.csv

```
model4 = xgb.XGBClassifier(colsample_bytree = 0.2, # 각나무마다 사용
하는 feature 비율
                           learning_rate = 0.05,
                           max_depth = 3,
                           alpha = 0.1,
                           n_estimators = 1000)

#  변경2 : 0.05 이하 제외 'guarantees', 'credits', 'qualification',
'residence_since', 'savings', 'dependents', 'phone',
sel = ['sum', 'term', 'payment', 'reason', 'other_credits',
'credit_report',
       'marital_status', 'age', 'employment', 'immigrant',
'accommodation',
       'estate', 'status']
```

In [ ]: