

# German Credit Risk

“독일 대출 데이터로 본  
신용등급 예측 모델 구축과 성능 비교”

# 목차

- 01 프로젝트 개요
  - 02 데이터 소개
  - 03 EDA 및 전처리
  - 04 신용등급 모델링
  - 05 정확도 개선 머신러닝
  - 06 대출 가이드라인
  - 07 결론 및 인사이트



“

은행 대출 고객의 인구학적, 사회적, 재정적, 대출 관련 특성을 기반으로  
신용등급을 분류해보고, 신용 리스크를 줄이기 위한 예측 모델을 만들어보자.

”

※ 신용평점은 신용평가회사와 은행 등 금융회사에서 산정합니다.

- 신용평가회사는 여러 금융회사의 금융거래정보를 수집하여 신용평점을 산정하고 이를 금융회사 등 정보이용자에게 제공하고 있습니다.
- 금융회사는 대출거래 취급 여부 및 거래조건(예:대출금리)을 정하기 위하여 신용평가회사의 신용평점, 해당 금융회사와의 금융거래정보(예금 평균잔액 등), 고객이 제공한 개인정보(소득, 직업 등) 등을 종합적으로 고려하여 자체적으로 신용평점을 산정합니다.

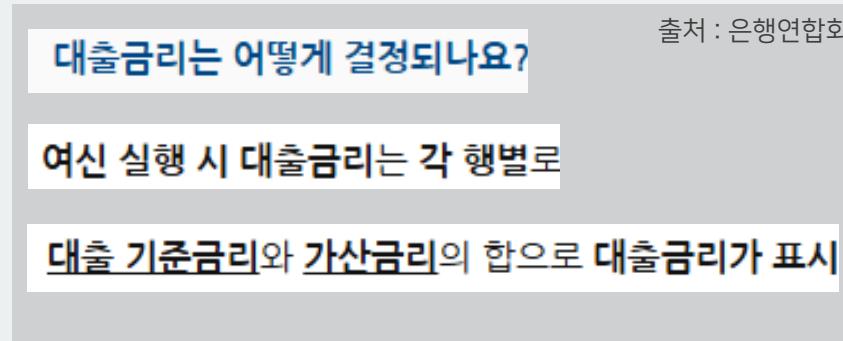
따라서 금융회사가 산정하는 신용평점은 신용평가회사의 신용평점과 다르며 금융회사별로도 상이 할 수 있습니다. 예를 들어 동일한 고객이라 할지라도 장기간 거래하면서 신용상태를 잘 아는 금융회사와 처음 거래하는 금융회사는 고객의 신용도를 달리 평가할 수 있습니다. 기존 대출거래가 있고 연체없이 성실하게 상환한 금융회사에서는 신용평점이 높게 산정되지만 과거에 거래가 전혀 없었던 금융회사에서는 신용평점이 낮게 산정될 수 있습니다.

창닫기

출처 : 금융감독원

## “신용점수란?”

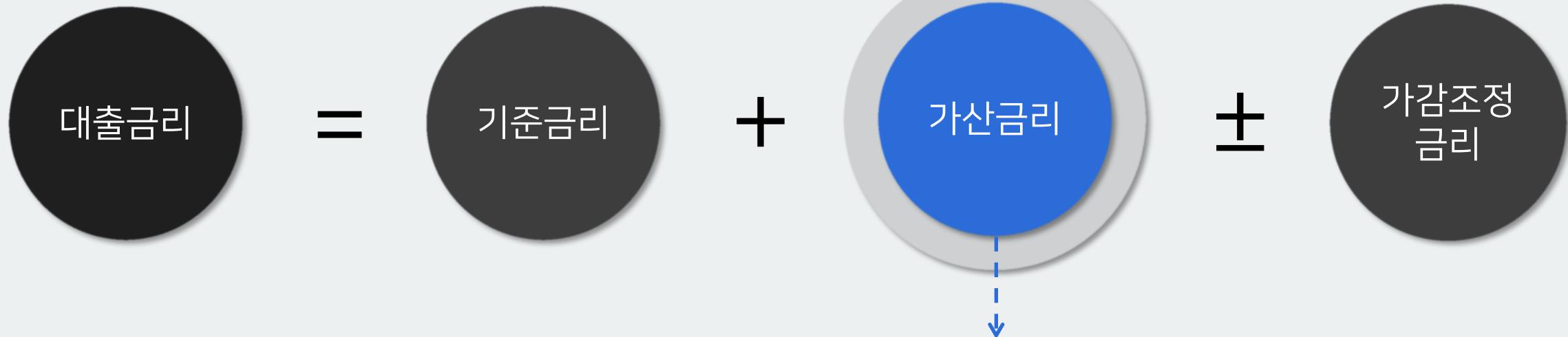
개인이 빌린 돈을 제때 갚을 수 있는 능력  
즉, 신용을 평가한 점수



〈대출금리 체계의 합리성 제고를 위한 모범규준에 따른 대출금리 결정 체계〉

출처 : 은행연합회

① 기준 금리	② 가산금리							
	리스크 관리비용			원가	법적비용	기타	가감조정 전결금리	
리스크 프리미엄	유동성 프리미엄	신용 프리미엄	자본 비용	업무 원가	법정 출연료 등	기대 이익률	부수거래 감면금리	본부·영업점 조정금리



고객의 신용점수 등에 따른 예상 손실 비용을 신용 프리미엄으로 반영

은행	구분	CB사 신용점수별 금리(%)										평균 신용점수	CB회사명	참고 사항
		1000~ 951점	950~ 901점	900~ 851점	850~ 801점	800~ 751점	750~ 701점	700~ 651점	650~ 601점	600점 이하	평균 금리			
NH농협은행	대출금리	4.69	4.80	4.97	5.10	5.08	5.26	5.74	5.89	6.76	4.80	936	KCB	
	기준금리	3.30	3.31	3.32	3.32	3.31	3.32	3.34	3.27	3.31	3.31			
	가산금리	3.08	3.09	3.17	3.22	3.25	3.34	3.75	4.06	4.42	3.12			
	가감조정금리	1.69	1.60	1.52	1.44	1.48	1.40	1.35	1.44	0.97	1.63			
신한은행	대출금리	4.95	5.07	5.08	5.55	6.01	6.83	8.13	8.64	7.29	5.16	927	KCB	
	기준금리	3.28	3.29	3.29	3.29	3.33	3.35	3.42	3.40	3.40	3.29			
	가산금리	2.61	2.70	2.67	3.14	3.52	4.32	5.68	6.14	4.80	2.79			
	가감조정금리	0.94	0.92	0.88	0.88	0.84	0.84	0.97	0.90	0.91	0.92			
우리은행	대출금리	4.87	5.01	5.23	5.47	5.87	6.21	6.64	7.46	8.87	5.06	935	KCB	
	기준금리	3.25	3.25	3.23	3.25	3.26	3.26	3.28	3.28	3.32	3.25			
	가산금리	2.83	2.92	3.07	3.24	3.52	3.79	4.10	4.80	6.10	2.96			
	가감조정금리	1.21	1.16	1.07	1.02	0.91	0.84	0.74	0.62	0.55	1.15			
하나은행	대출금리	4.67	4.71	4.81	4.82	4.98	5.03	5.19	5.66	7.26	4.72	939	KCB	
	기준금리	3.25	3.24	3.25	3.23	3.25	3.25	3.23	3.24	3.27	3.24			
	가산금리	3.32	3.33	3.40	3.41	3.60	3.66	3.80	4.28	6.30	3.35			
	가감조정금리	1.90	1.86	1.84	1.82	1.87	1.88	1.84	1.86	2.31	1.87			
KB국민은행	대출금리	4.57	4.69	4.91	5.15	5.46	5.91	6.79	7.79	7.83	4.75	930	KCB	
	기준금리	3.25	3.25	3.26	3.25	3.26	3.28	3.30	3.34	3.34	3.25			
	가산금리	3.74	3.73	3.91	4.03	4.21	4.42	4.83	5.39	5.38	3.81			
	가감조정금리	2.42	2.29	2.26	2.13	2.01	1.79	1.34	0.94	0.89	2.31			

출처 : 은행연합회

은행	구분	신용점수별 금리(%)		신용점수에 따른 차이값
		1000~951점	600점이하	
KB국민은행	대출금리	4.57	7.83	3.26
	기준금리	3.25	3.34	0.09
	가산금리	3.74	5.38	1.64
	가감조정금리	2.42	0.89	-1.53
NH농협은행	대출금리	4.69	6.76	2.07
	기준금리	3.30	3.31	0.01
	가산금리	3.08	4.42	1.34
	가감조정금리	1.69	0.97	-0.72
우리은행	대출금리	4.87	8.87	4
	기준금리	3.25	3.32	0.07
	가산금리	2.83	6.10	3.27
	가감조정금리	1.21	0.55	-0.66
신한은행	대출금리	4.95	7.29	2.34
	기준금리	3.28	3.40	0.12
	가산금리	2.61	4.80	2.19
	가감조정금리	0.94	0.91	-0.03
하나은행	대출금리	4.67	7.26	2.59
	기준금리	3.25	3.27	0.02
	가산금리	3.32	6.30	2.98
	가감조정금리	1.90	2.31	0.41

\* 신용점수에 따라 적게는 2.07%에서 많게는 4%까지 금리를 차등 책정

## 돈을 돌려받지 못할 위험을 예측하기 위한 지표로 작용되는 신용점수

신용점수가 높은 고객 : 갚을 가능성이 높은 안정적인 수익원

신용점수가 낮은 고객 : 채무 불이행 위험이 높아, 이를 보상하기 위해 더 높은 금리를 부과하거나 대출을 거절함으로써 손실을 최소화하고 수익을 도모

- 한국 시중 5대은행 가계대출 신용점수별 금리현황(2024년 11월 기준)
- 대출금리 = 기준금리 + 가산금리 - 가감조정금리

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	0	67	male	2	own	NaN	little	1169	6	radio/TV	good
1	1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
2	2	49	male	1	own	little	NaN	2096	12	education	good
3	3	45	male	2	free	little	little	7882	42	furniture/equipment	good
4	4	53	male	2	free	little	little	4870	24	car	bad
...	...	...	...	...	...	...	...	...	...	...	...
995	995	31	female	1	own	little	NaN	1736	12	furniture/equipment	good
996	996	40	male	3	own	little	little	3857	30	car	good
997	997	38	male	2	own	little	NaN	804	12	radio/TV	good
998	998	23	male	2	free	little	little	1845	45	radio/TV	bad
999	999	27	male	2	own	moderate	moderate	4576	45	car	good

- 행 : 1000개
- 열 : 11개 (나이, 성별, 일자리, 주택유형, 저축계좌, 당좌예금, 대출금액, 상환기간, 대출목적, 대출리스크 수준)

---

01**나이**

경제활동이 활발한 연령대(30~50세)일수록 신용등급이 높을 것이다.

---

02

**직업 안정성**

숙련직일수록 신용등급이 높을 것이다.

---

03

**저축 수준**

저축계좌와 당좌예금의 상태가 좋을수록 신용등급이 높을 것이다.

---

04

**주택 유형**

자가(주택을 소유)인 경우 신용등급이 높을 것이다.

---

05

**대출 금액 대비 기간 비율**

대출금액 대비 상환기간이 짧을수록 상환 능력이 뛰어나 신용등급이 높을 것이다.

---

06

**대출 목적**

대출 목적은 유형자산이 남는 것일수록 신용등급이 높을 것이다.

---

Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk
0	67	male	2	own	NaN	little	1169	6	radio/TV	good
1	22	female	2	own	little	moderate	5951	48	radio/TV	bad
2	49	male	1	own	little	NaN	2096	12	education	good
3	45	male	2	free	little	little	7882	42	furniture/equipment	good
4	53	male	2	free	little	little	4870	24	car	bad
...	...	...	...	...	...	...	...	...	...	...
995	31	female	1	own	little	NaN	1736	12	furniture/equipment	good
996	40	male	3	own	little	little	3857	30	car	good
997	38	male	2	own	little	NaN	804	12	radio/TV	good
998	23	male	2	free	little	little	1845	45	radio/TV	bad
999	27	male	2	own	moderate	moderate	4576	45	car	good

- 인덱스 열인 Unnamed: 0은 분석에 필요하지 않기 때문에 drop 함수를 사용해 삭제

## Isnull().sum() 결과

Age	0
Sex	0
Job	0
Housing	0
Saving accounts	183
Checking account	394
credit amount	0
Duration	0
Purpose	0
Risk	0
age_group	0
Credit_per_duration	0
dtype: int64	

- 두 컬럼의 .unique() 결과값  
→ little, moderate, (quite rich), rich

## • 결측치 삭제

- 결측치의 비율이 꽤 높지만 데이터상 금융에 관련된 중요한 정보이므로 **기각**

## • 결측치 대체

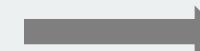
- 결측치를 제외한 나머지 데이터가 차지하는 비율별로 결측치 대체  
→ 열의 비율은 채울 수 있으나 행 각자의 데이터 통일성에 결함을  
줄 수 있어 **기각**
- 행 자체의 통일성을 유지하기 위해 Age group, Job, Housing을  
기준으로 그룹화하고 각 그룹별 최빈값을 구해 그 값으로 결측치를  
대체 → **선택**

## Value\_counts() 결과

```
(Saving accounts
 little      784
 moderate    103
 quite rich   64
 rich        48
 0           1
 Name: count, dtype: int64,
Checking account
moderate      473
little       463
rich         63
 0           1
Name: count, dtype: int64,
```

## Value\_counts(), isna().sum() 결과

```
(Saving accounts
 little      784
 moderate    104
 quite rich   64
 rich        48
Name: count, dtype: int64,
Checking account
moderate      474
little       463
rich         63
Name: count, dtype: int64)
```



- 1단계 처리 후에도 각 1개 행이 0으로 존재  
→ 데이터 전체 최빈값으로 대체

- 결측치 처리 완료

## Value\_counts() 결과

```
(Saving accounts
 little      784
 moderate    103
 quite rich   64
 rich        48
 0           1
 Name: count, dtype: int64,
 Checking account
 moderate    473
 little      463
 rich        63
 0           1
 Name: count, dtype: int64,
```



## Value\_counts(), isna().sum() 결과

```
(Age          0
 Sex          0
 Job          0
 Housing      0
 Saving accounts  0
 Checking account 0
 Credit amount   0
 Duration       0
 Purpose        0
 Risk           0
 Age_group      0
 dtype: int64,
```

- 1단계 처리 후에도 각 1개 행이 0으로 존재  
→ 데이터 전체 최빈값으로 대체

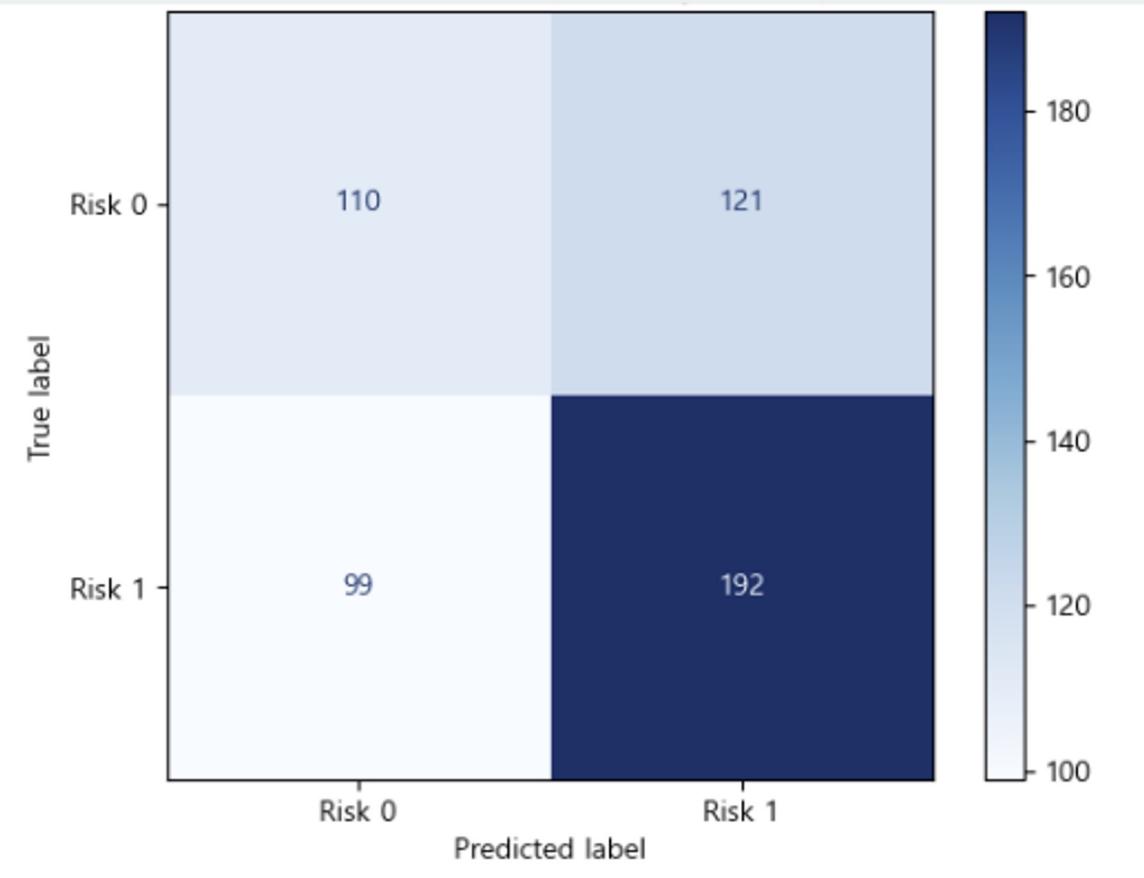
- 결측치 처리 완료

## 결측치 제거 후 정확도

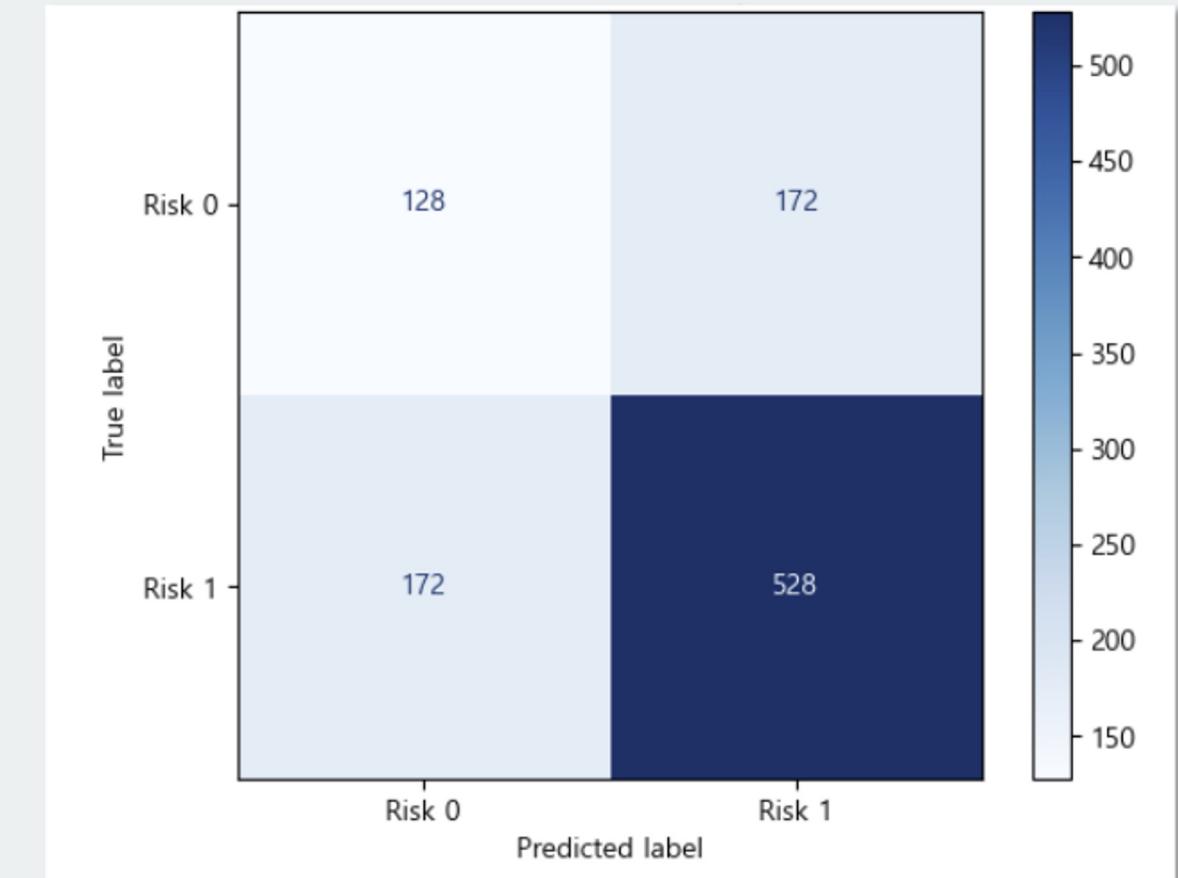
VS

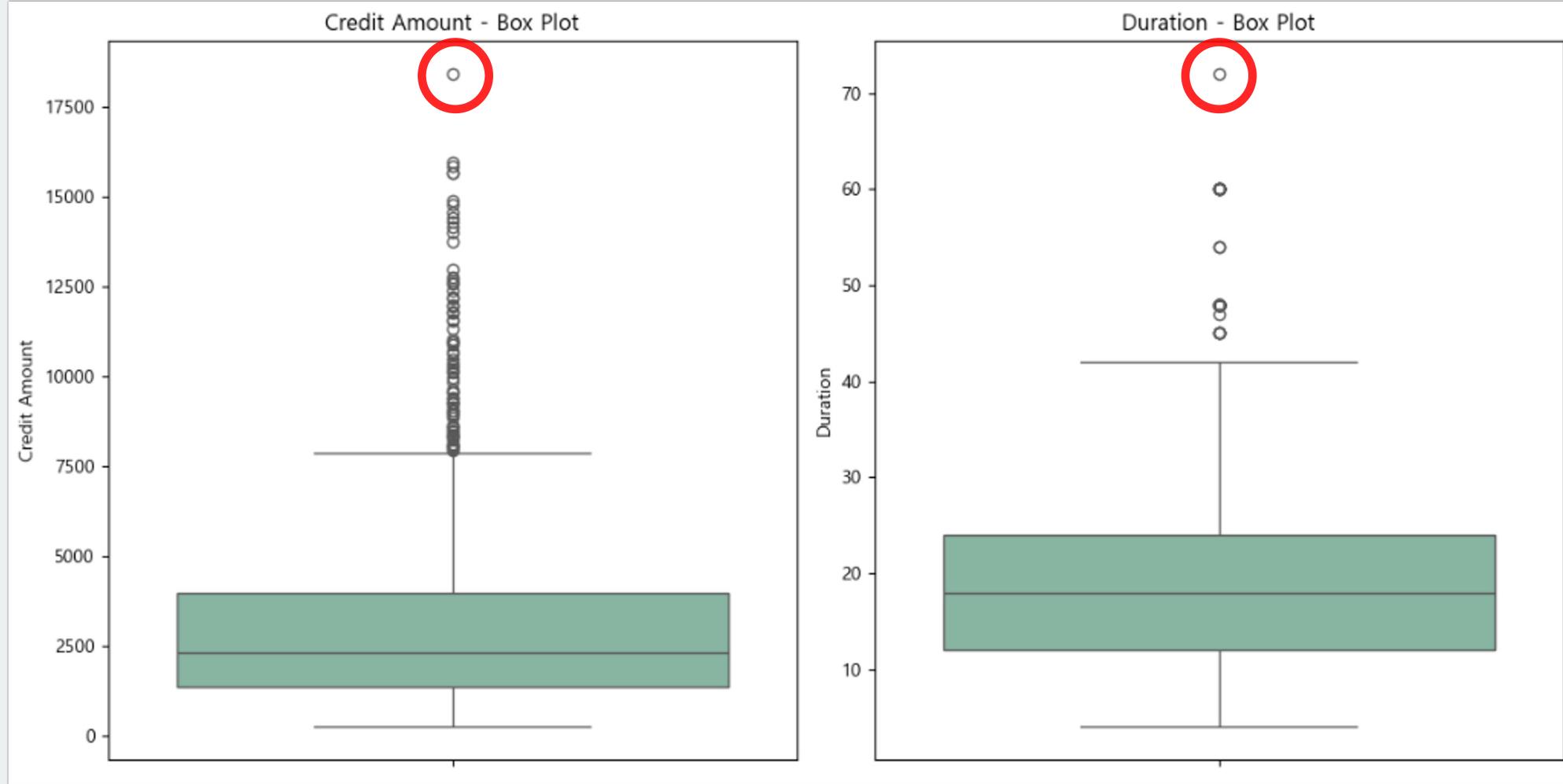
## 결측치 대체 후 정확도

Confusion Matrix (Accuracy: 0.5785)



Confusion Matrix (Accuracy: 0.6560)





	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose	Risk	Age_group	Credit_ratio
0	67	male	2	own	little	little	1169	6	radio/TV	good	Senior	194.833333
1	22	female	2	own	little	moderate	5951	48	radio/TV	bad	Student	123.979167
2	49	male	1	own	little	little	2096	12	education	good	Adult	174.666667
3	45	male	2	free	little	little	7882	42	furniture/equipment	good	Adult	202.916667
4	53	male	2	free	little	little	4870	24	car	bad	...	...
...	...	...	...	...	...	...	...	...	...	...	Young	144.666667
995	31	female	1	own	little	moderate	1736	12	furniture/equipment	good	Adult	128.566667
996	40	male	3	own	little	little	3857	30	car	good	Adult	67.000000
997	38	male	2	own	little	little	804	12	radio/TV	good	Student	41.000000
											Young	101.688889

- Age\_group : 18세, 25세, 35세, 60세 구간으로 나누어 'Student', 'Young', 'Adult', 'Senior' 라벨링
- Credit\_ratio : 대출 금액 대비 상환기간 비율 (Credit amount / Duration)

```
#범주형 데이터 수치화
sex_mapping = {'female': 2, 'male': 1}
saving_mapping = {'little': 1, 'moderate': 2, 'quite rich': 3, 'rich': 4}
housing_mapping = {'free': 1, 'rent': 2, 'own': 3}
purpose_mapping = {'radio/TV': 1, 'education': 0, 'furniture/equipment': 1,
                   'car': 1, 'business': 0, 'domestic appliances': 1,
                   'repairs': 0, 'vacation/others': 0}
risk_mapping = {'bad': 0, 'good': 1}

data['Sex'] = data['Sex'].map(sex_mapping)
data['Saving accounts'] = data['Saving accounts'].map(saving_mapping)
data['Checking account'] = data['Checking account'].map(saving_mapping)
data['Housing'] = data['Housing'].map(housing_mapping)
data['Purpose'] = data['Purpose'].map(purpose_mapping)
data['Risk'] = data['Risk'].map(risk_mapping)
```

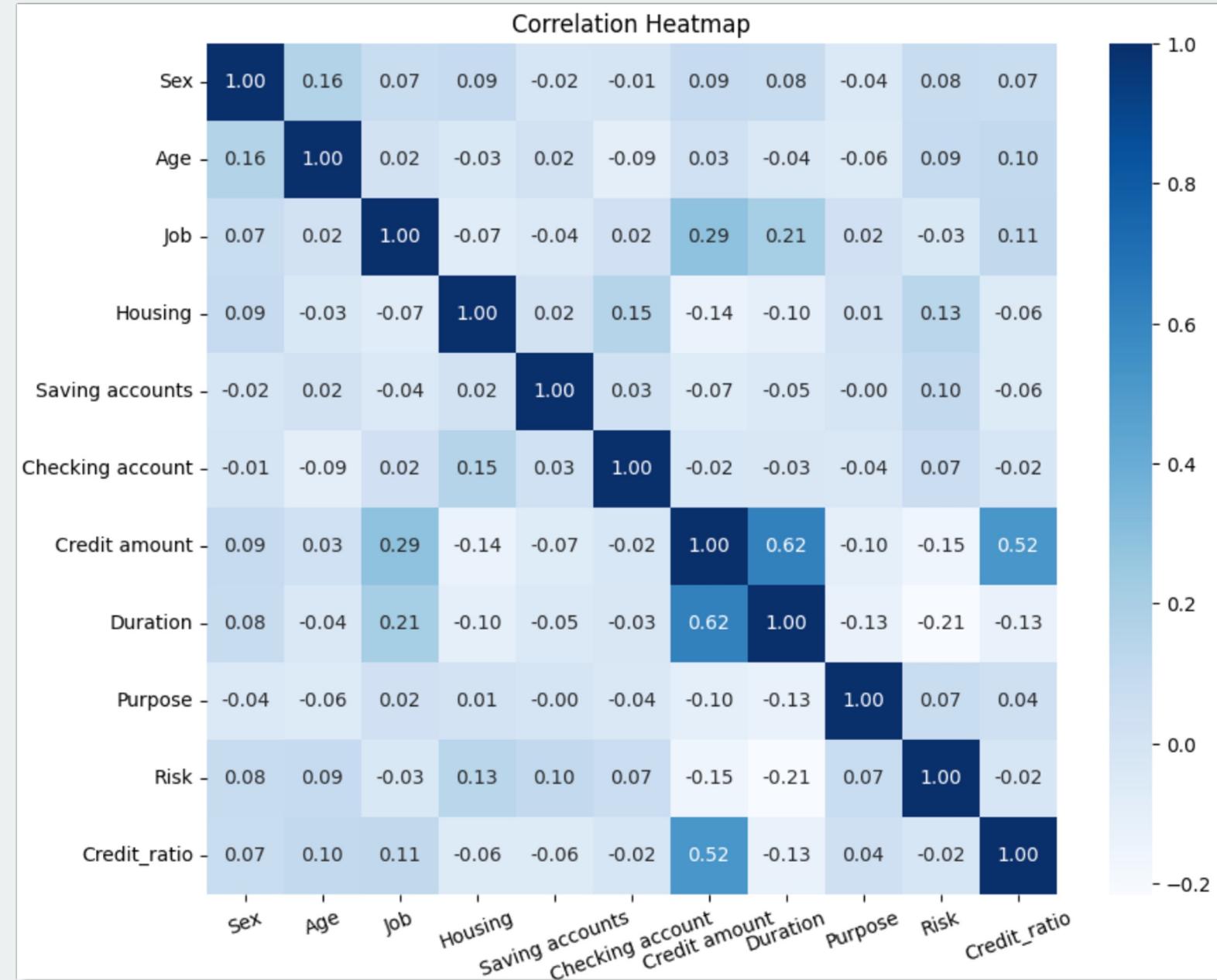
- 상관분석을 위해 범주형 데이터를 수치형으로 변환

원본 컬럼과 파생 컬럼의 관계를 제외하면

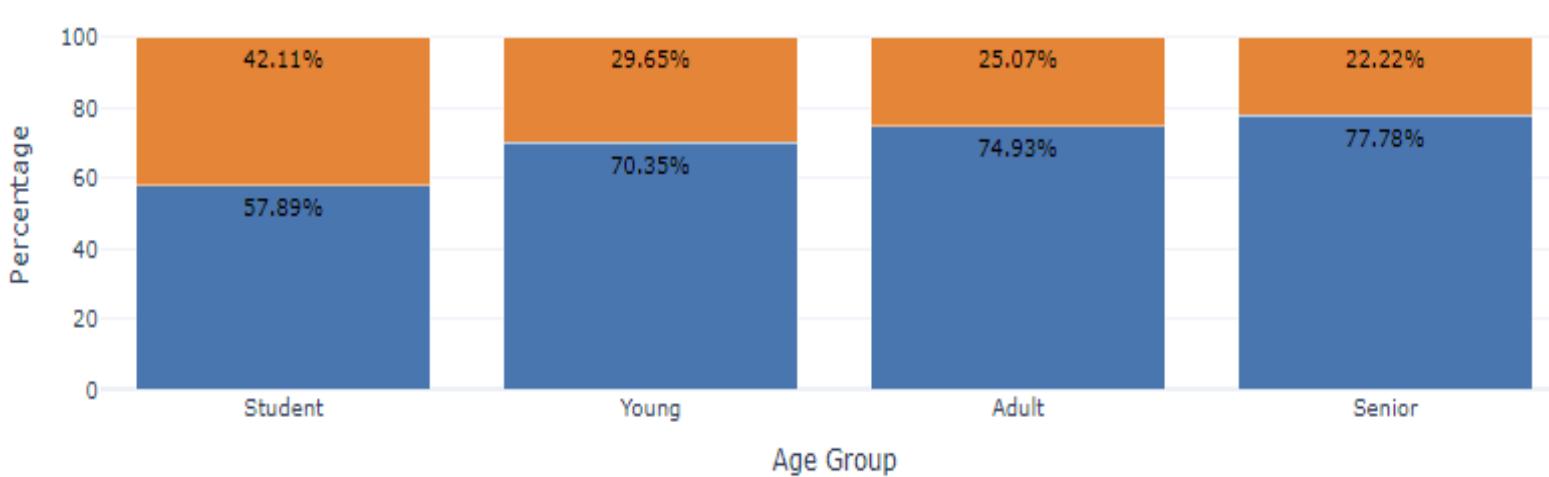
Duration과 Credit amount (0.62)  
Job과 Credit amount (0.29)

:

그 외 다른 변수들 간 큰 상관관계 없음



Age Group vs Risk (%)



- Student: 18~25세
- Young: 26~35세
- Adult: 36~60세
- Senior: 60세~

## 01 나이

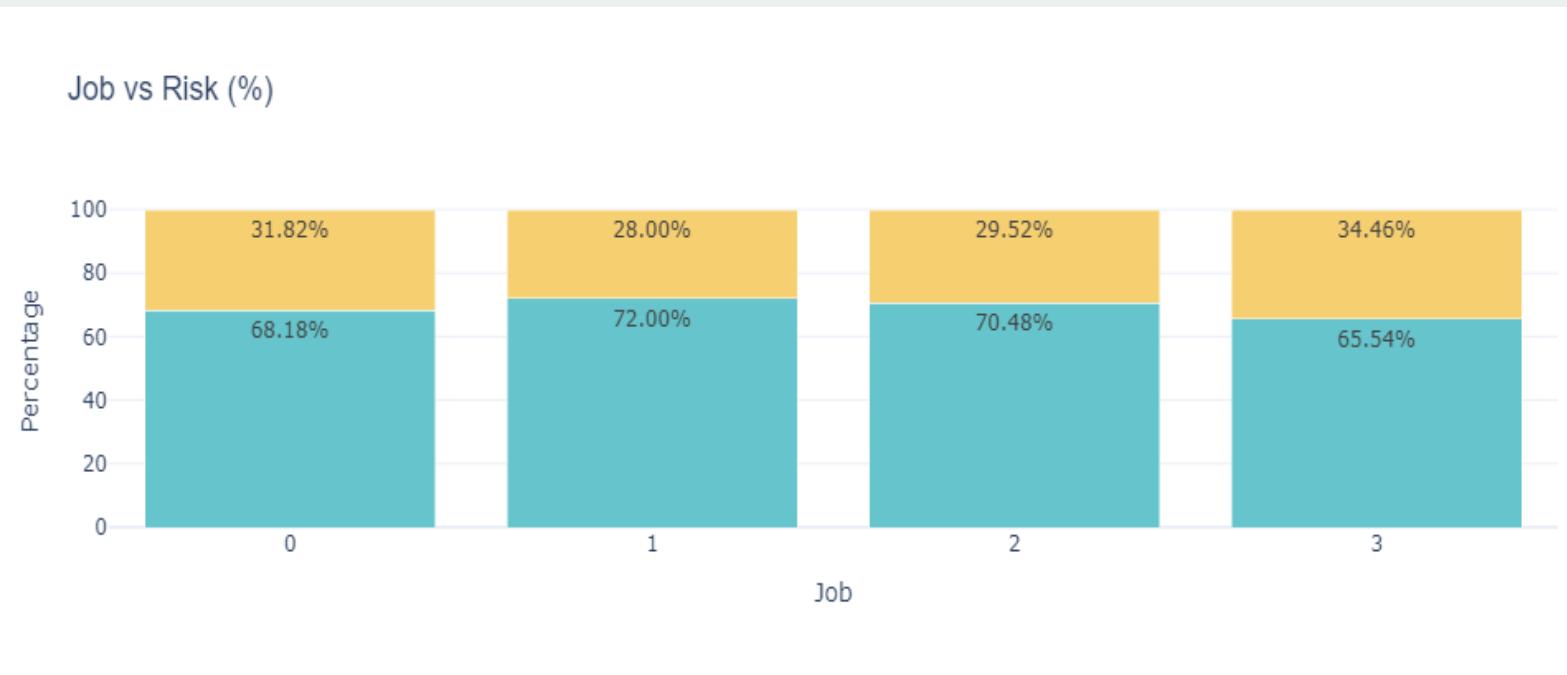
02 직업 안정성

03 저축 수준

04 주택 유형

05 대출 금액 대비 기간 비율

06 대출 목적



- 0: 비숙련 단기 계약직
- 1: 비숙련 정규직
- 2: 숙련된 기술직
- 3: 전문직

01 나이

## 02 직업 안정성

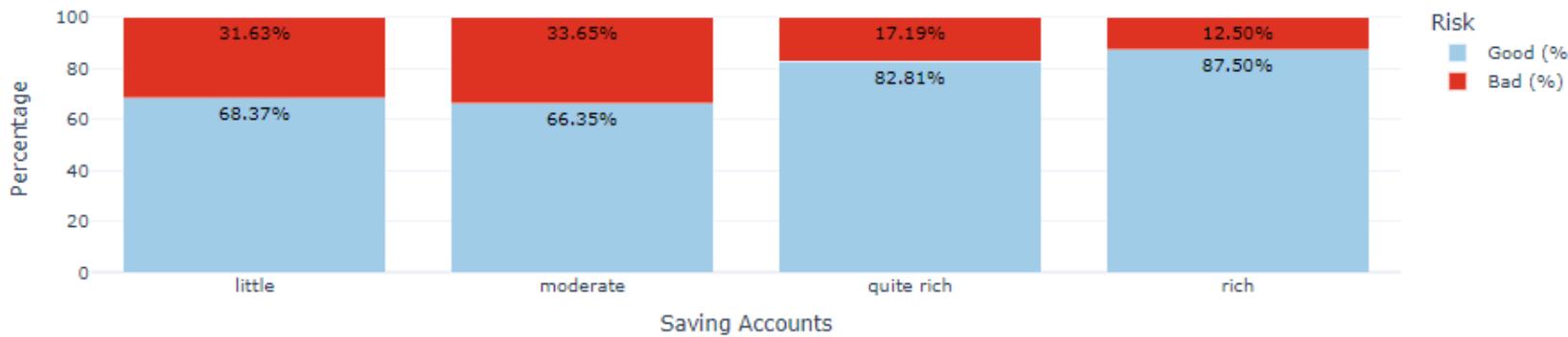
03 저축 수준

04 주택 유형

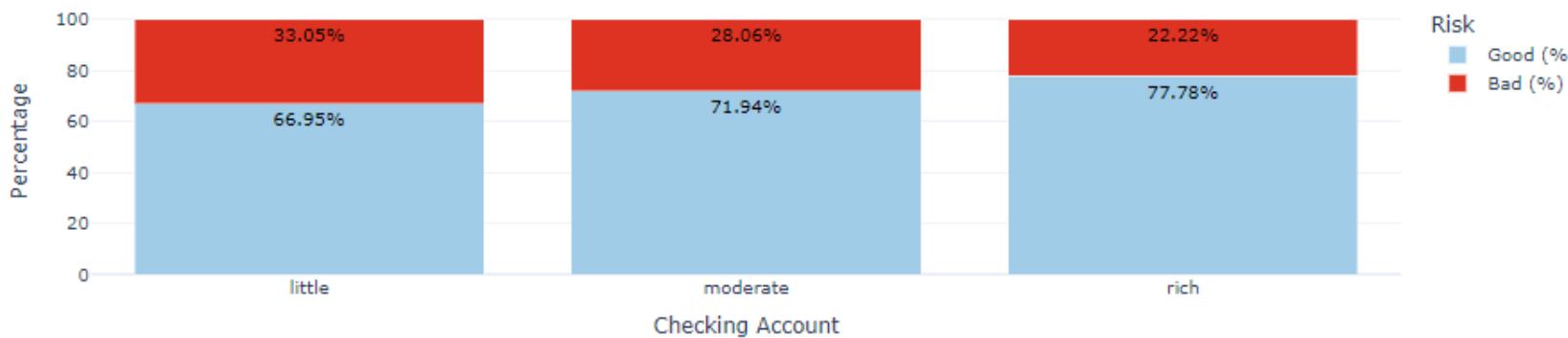
05 대출 금액 대비 기간 비율

06 대출 목적

Saving Accounts vs Risk (%)



Checking Account vs Risk (%)



01 나이

02 직업 안정성

## 03 저축 수준

04 주택 유형

05 대출 금액 대비 기간 비율

06 대출 목적

Housing vs Risk (%)



- free: 무료
- rent: 월세, 전세
- own: 자가

01 나이

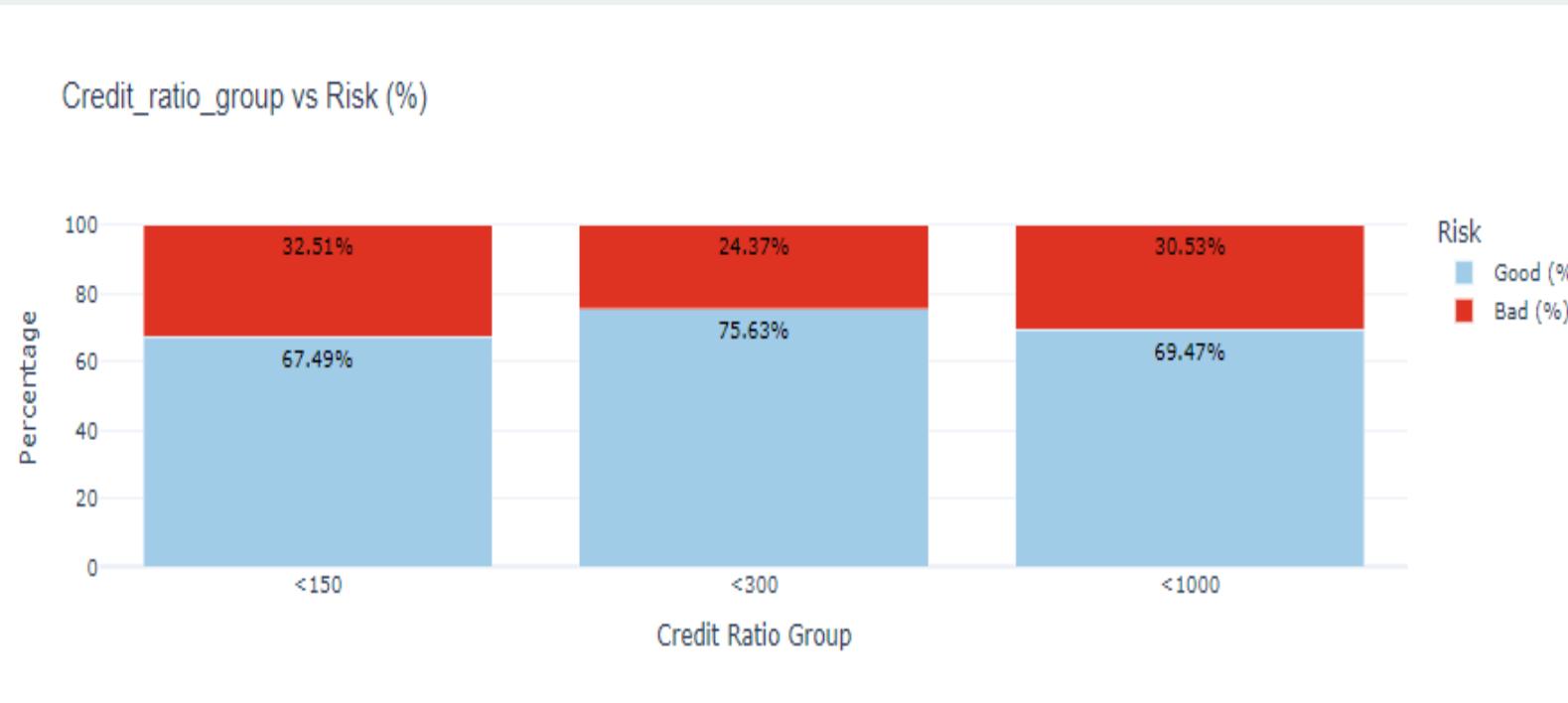
02 직업 안정성

03 저축 수준

## 04 주택 유형

05 대출 금액 대비 기간 비율

06 대출 목적



01 나이

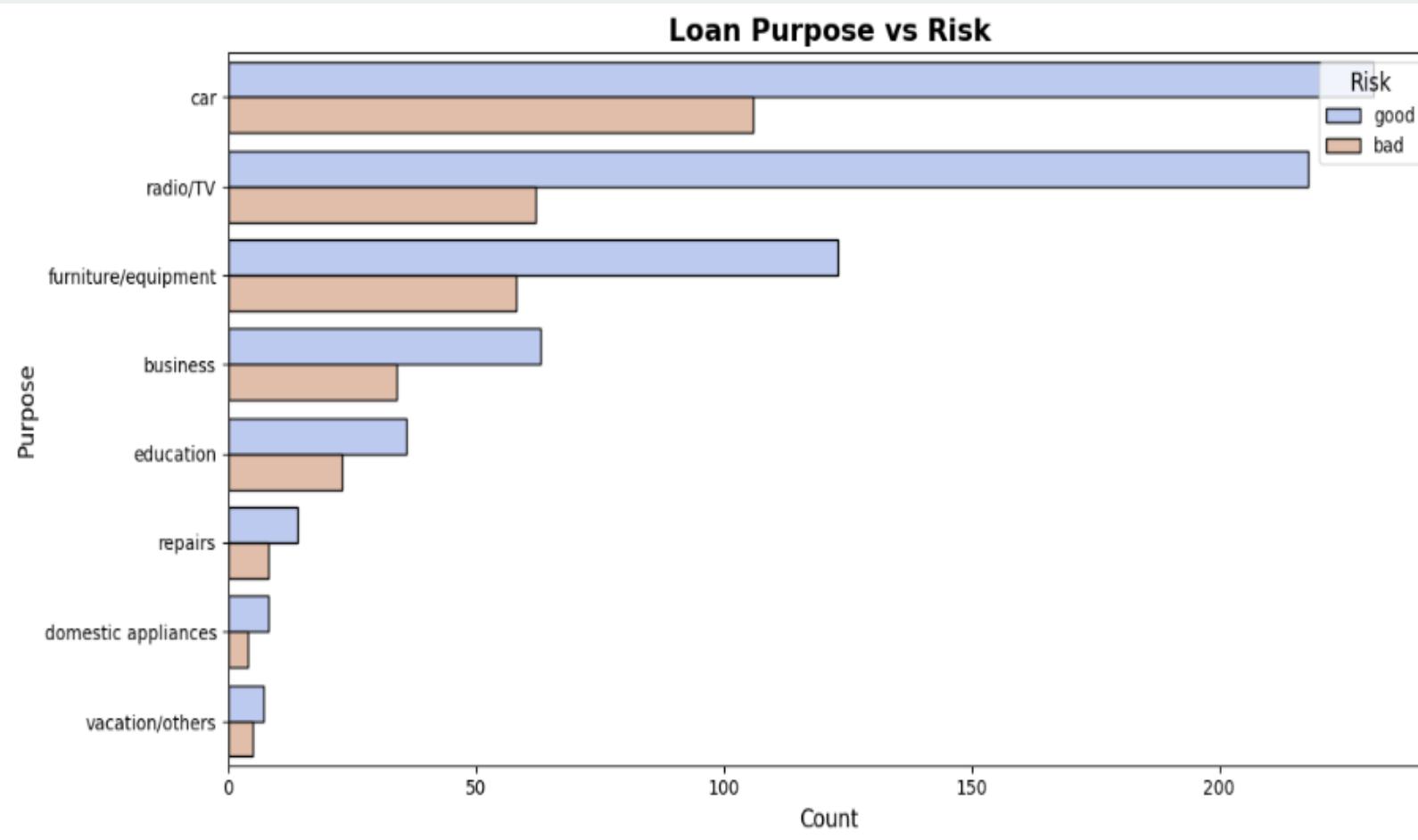
02 직업 안정성

03 소득과 저축 수준

04 주택 유형

## 05 금액 / 기간 비율

06 대출 목적



- Car: 차
- Radio/TV: 라디오/TV
- Furniture/equipment: 가구/장비
- Business: 사업
- Education: 교육
- Repairs: 수리, 정비
- Domestic appliances: 가전제품
- Vacation/others: 휴가

01 나이

02 직업 안정성

03 소득과 저축 수준

04 주택 유형

05 대출 금액 대비 상환기간

## 06 대출 목적

1. 결측치가 많은 데이터라 할지라도 도메인상 중요한 데이터에 해당된다면 컬럼을 삭제하는 방법보다 결측치를 **정밀하게 대체하는 방법을 우선적으로 시도해볼 필요가 있음.**
2. 이상치는 존재했지만 크게 문제되지 않아 데이터를 보존함.
3. 데이터들 간의 **상관관계는 전반적으로 낮게 파악됨.**
4. 분석 결과, 가설로 삼았던 항목들 중 'Housing, Saving accounts, Checking accounts, Purpose' 네 항목은 개인의 신용위험도(Risk)와 관련이 있어 **가설이 옳고**, 따라서 개인 신용등급 모델링 시 **가중치를 더 부여하는 방식을 선택.**
5. 반면, 'Age, Job, Credit\_ratio' 세 항목은 **가설이 틀렸음을 확인했고**, 따라서 개인 신용등급 모델링 시 **가중치를 덜 부여하는 방식을 선택.**

```

import pandas as pd
import numpy as np

# 기준별 점수화
def calculate_score(row):
    score = 0
    # Saving accounts와 Checking account 점수
    score += row["Saving accounts"] * 10
    score += row["Checking account"] * 10

    # Housing 점수
    score += row["Housing"] * 3

    # Job 점수
    score += row["Job"] * 3
    # Purpose 점수
    score += 10 if row["Purpose"] == 1 else 0

    # Age_group 점수
    age_score = { "Student": 5, "Young": 10, "Adult": 15, "Senior": 20 }
    score += age_score[row["Age_group"]]

    # Credit_ratio 점수 (정규화하여 반영)
    score += row["Credit_ratio"] / 100

    return score

# 점수 계산
df["Score"] = df.apply(calculate_score, axis=1)

# 가중치를 반영한 비율 설정 (1등급: 10%, 2등급: 40%, 3등급: 20%, 4등급: 30%)
bins = [0, 0.3, 0.5, 0.9, 1.0]

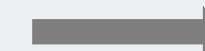
# qcut으로 신용등급 나누기
df['Credit_Rating'] = pd.qcut(df['Score'].rank(method='first') / len(df), q=bins, labels=["1", "2", "3", "4"])

# 결과 1. 신용도 good 인데 1등급, 2등급이 랭킹하지만 low도 많다..
df.groupby(['Risk', 'Credit_Rating'], observed=False).size().to_frame()

```

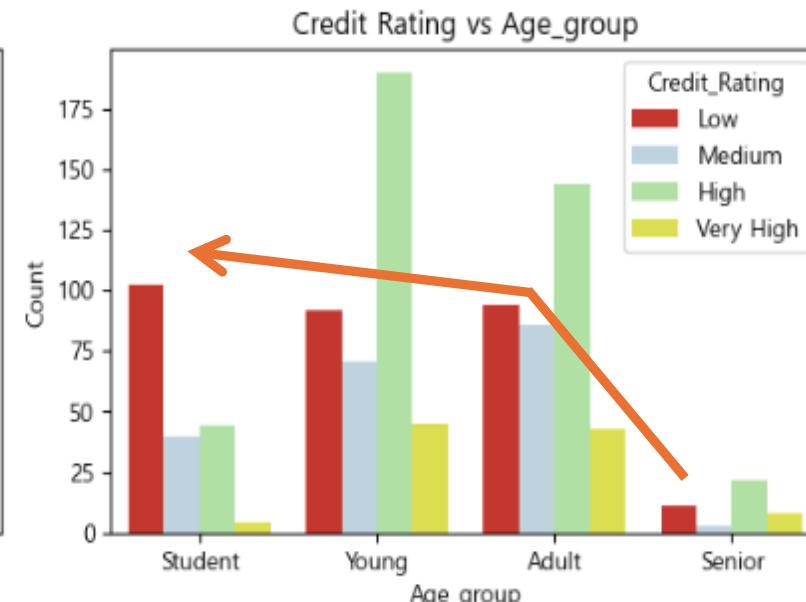
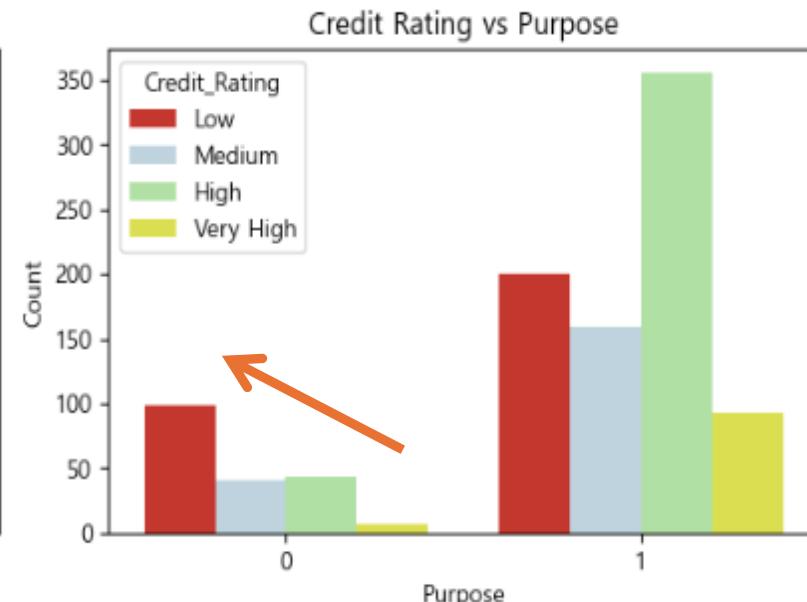
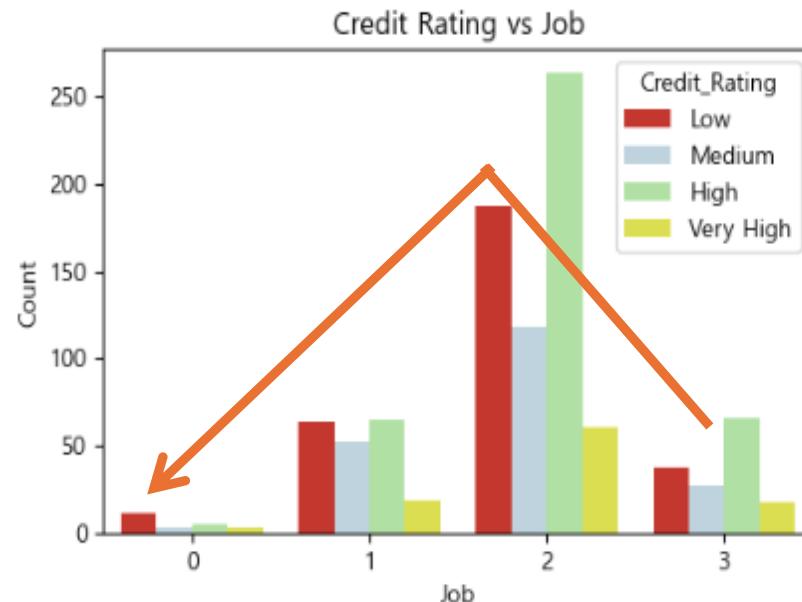
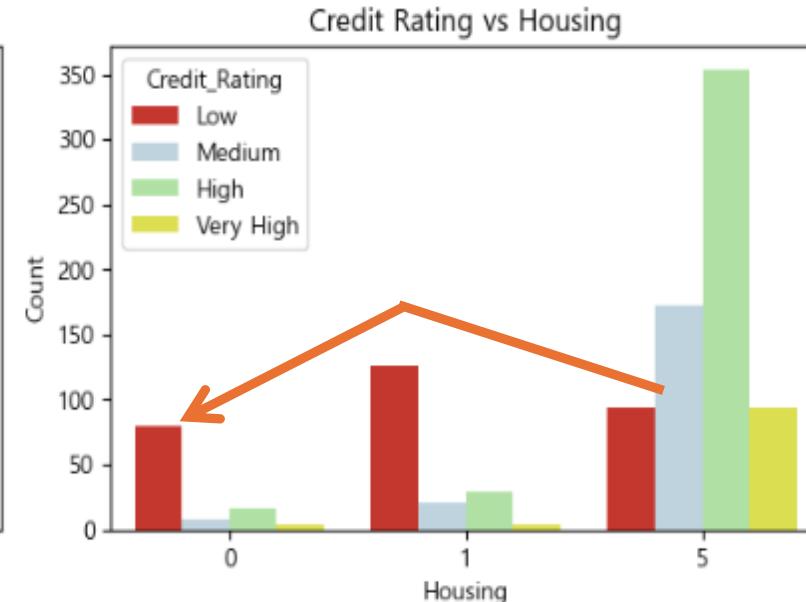
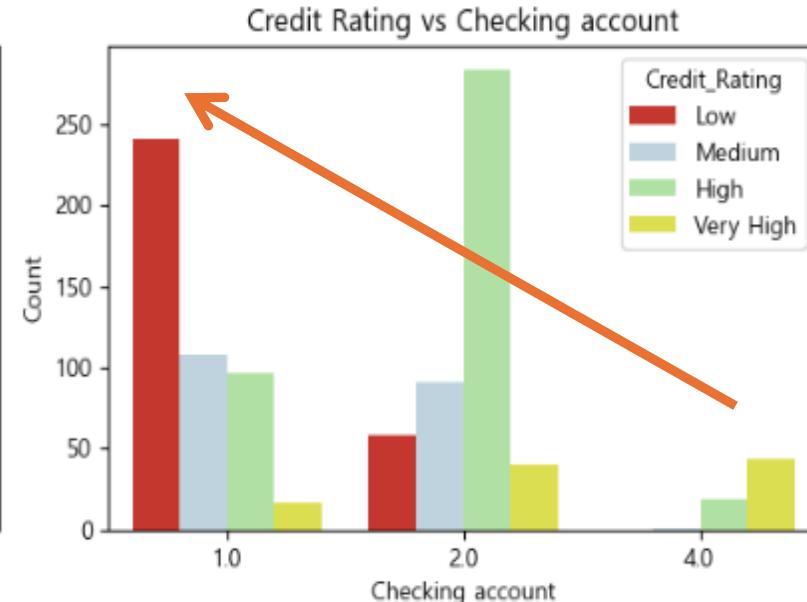
## 고객별 점수를 계산하기 위한 코드 작성

- 가설 검증 Good (점)
  1. Saving accounts: x 10
  2. Checking account: x 10
  3. Purpose: 0 or 10
  4. Age: 5~20
  
- 가설 검증 Bad (점)
  1. Housing: x 3
  2. Job: x 3
  3. Credit\_ratio: / 100



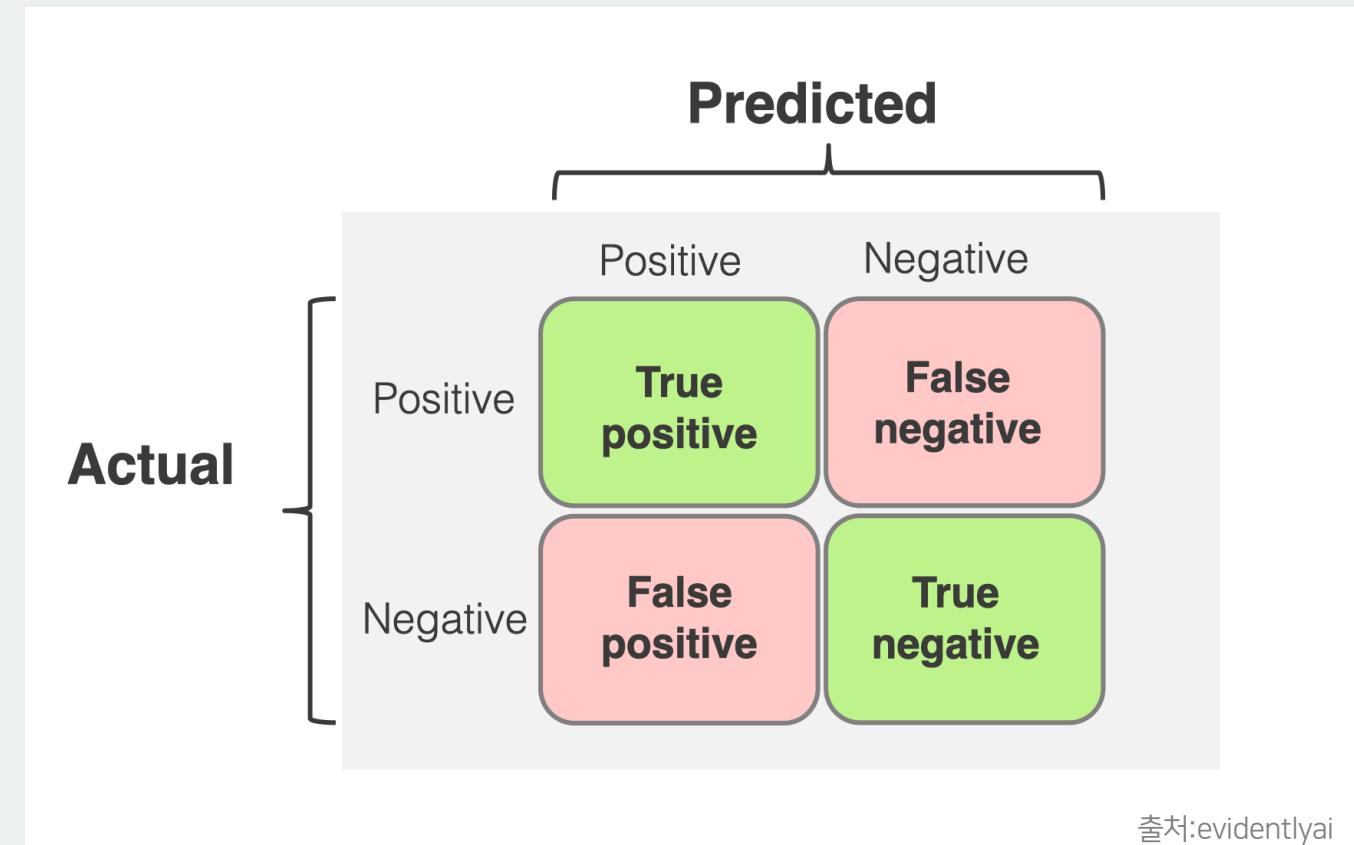
실행 결과 Risk별 신용등급 분포

Risk	Credit_Rating	(행)
bad	1	128
	2	62
	3	92
	4	18
good	1	172
	2	138
	3	308
	4	82



## 혼동행렬이란?

- 실제 값과 모델이 예측한 값을 비교하여 모델의 정확성과 기타 성능 지표를 평가
- 요소
  1. TP: 실제 True(타입)을 True로 예측한 case
  2. TN: 실제 False를 False로 예측한 case
  3. FN: 실제 True를 False로 예측한 case
  4. FP: 실제 False를 True로 예측한 case
- 성능 측정 지표
  1. 정확도:  $TP + TN / ALL$ : 전체 예측 성공 비율
  2. 정밀도:  $TP / (TP + FP)$ : 예측 True 중 실제 True 비율
  3. 재현율:  $TP / (TP + FN)$ : 실제 True 중 예측 True 비율
  4. 특이도:  $TN / (TN + FP)$ : 실제 False 중 예측 False 비율
  5. f-1 score: 전반적인 모델 성능 지표 (조화평균 가중치)



출처:evidentlyai

- 모델 전체 정확도

- Accuracy (정확도): **0.66**  $(528 + 128) / 1000$

- Case1 (Risk1 = Good)

- Precision (정밀도) : **0.75**  $(528 + 172) / 700$

- Recall (재현율) : **0.75**  $(528 + 172) / 700$

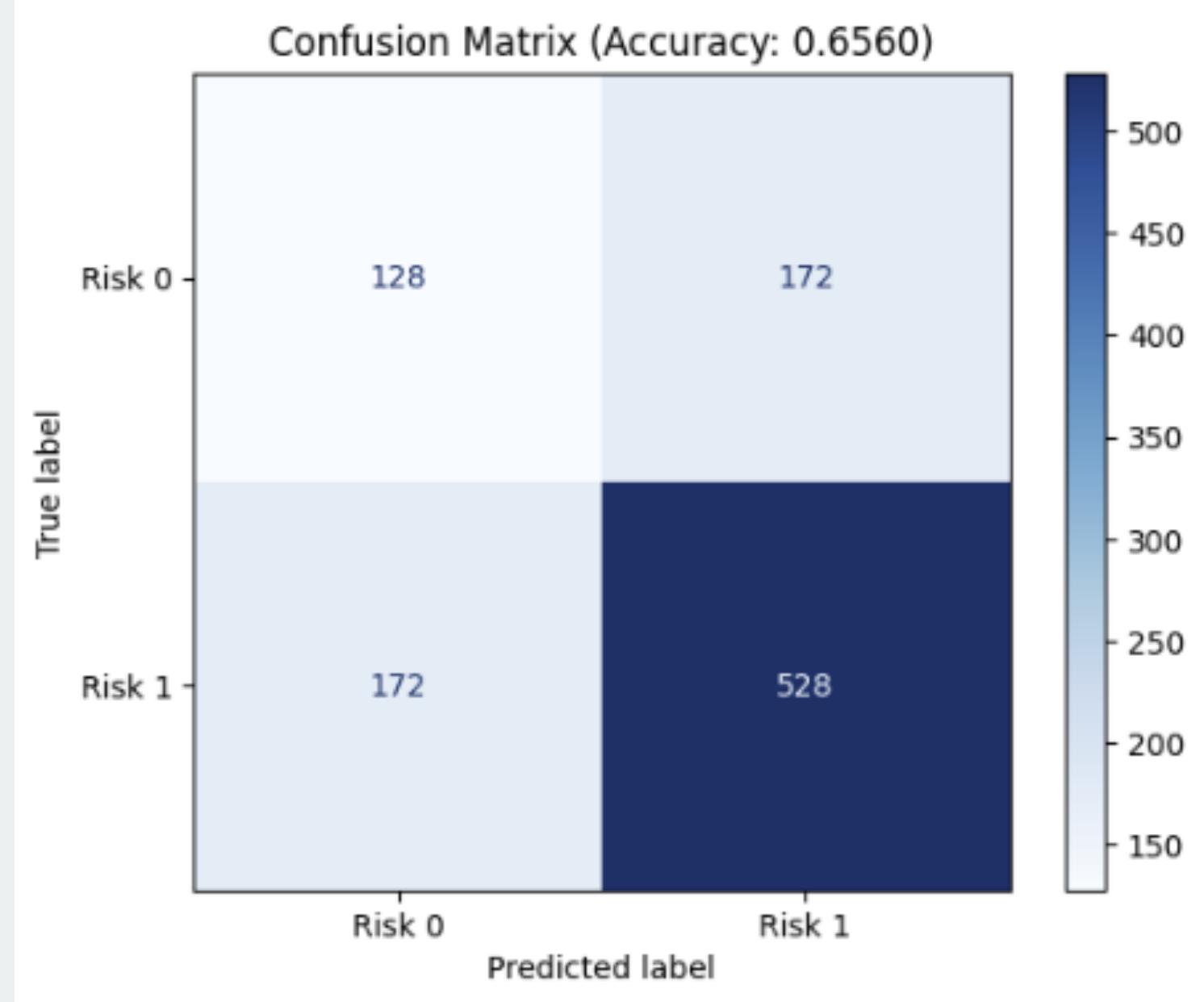
- F1-score : **0.75**

- Case2 (Risk0 = Bad)

- Precision (정밀도) : **0.43**  $(528 + 172) / 700$

- Recall(재현율) : **0.43**  $(528 + 172) / 700$

- F1-score : **0.43**



- 모델 전체 정확도

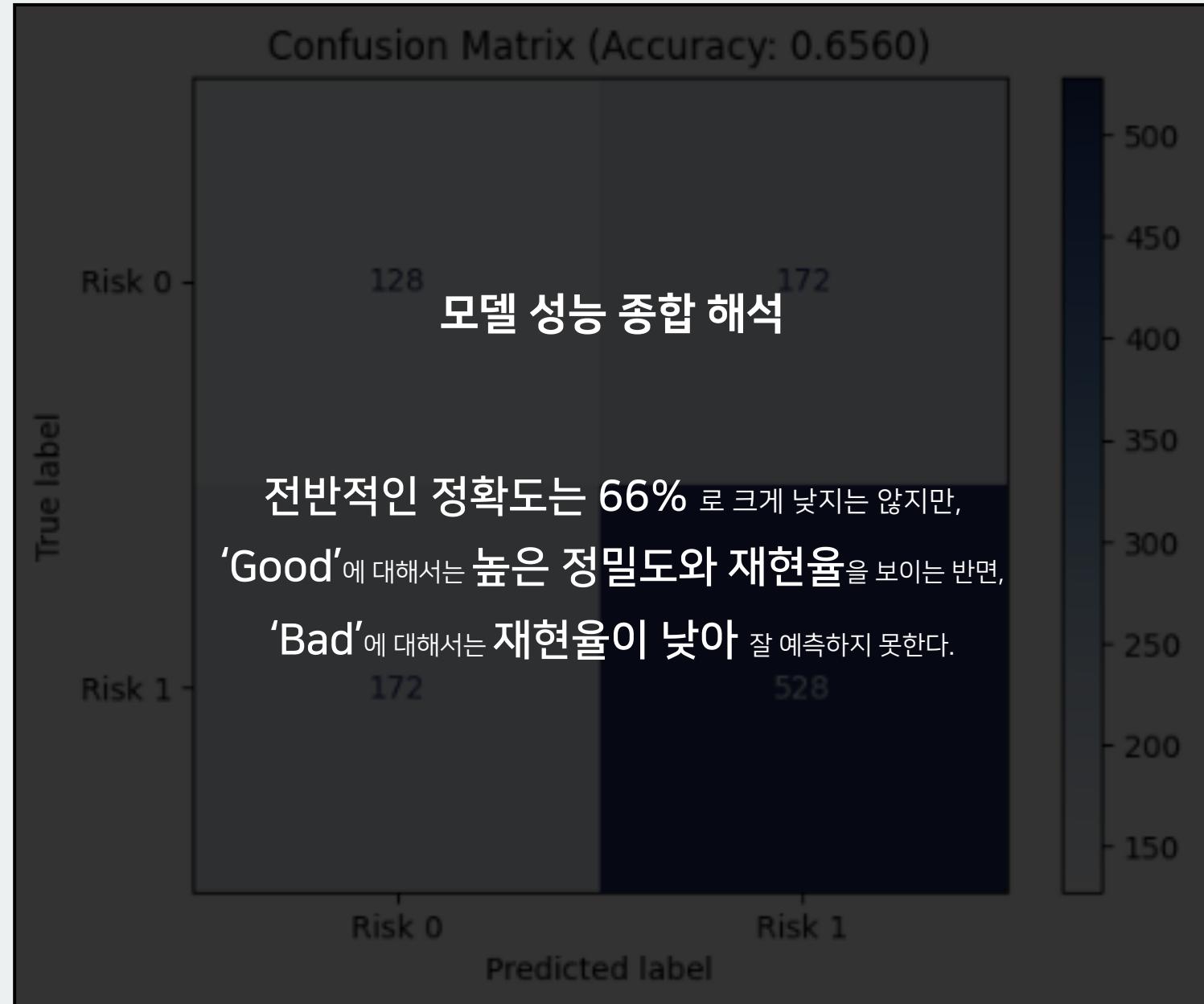
- Accuracy (정확도): **0.66**  $(528 + 128) / 1000$

- Case1 (Risk1 = Good)

- Precision (정밀도) : **0.75**  $(528 + 172) / 700$
- Recall (재현율) : **0.75**  $(528 + 172) / 700$
- F1-score : **0.75**

- Case2 (Risk0 = Bad)

- Precision (정밀도) : **0.43**  $(528 + 172) / 700$
- Recall(재현율) : **0.43**  $(528 + 172) / 700$
- F1-score : **0.43**



1. Risk가 good인 데이터를 예측한 경우의 재현율, 정밀도, f1-score의 지표는 **0.75로 준수한 결과**가 나왔음
2. Risk가 bad인 데이터를 예측한 경우의 재현율, 정밀도, f1-score의 지표는 **0.43으로 낮은 결과**가 나왔음
3. 하지만 모델링을 통한 궁극적인 목표는 신용리스크가 높은 고객들을 판별, 대출 형태를 다르게 제공하는 것을 통해  
기업의 위험도를 낮추는 것이기 때문에 본 모델이 갖는 **약 66%의 정확도가 높다고 판단하기 어려움**
4. 정확도 목표를 **70% 이상**으로 설정하였기 때문에 이를 개선할 방안이 필요
5. 모델 개선방안으로서 여러 머신러닝 기법을 모색할 필요가 있음

```
from sklearn.metrics import accuracy_score

# 실제 값과 예측 값 준비
y_true = df['Risk'] # 실제 값
y_pred = df['Credit_Rating'].map({
    "Low": 0,
    "Medium": 1,
    "High": 1,
    "Very High": 1
}) # 예측 값 (Risk 0 또는 1로 대체)

# 정확도 계산
accuracy = accuracy_score(y_true, y_pred)

# 결과 출력
print(f"Accuracy: {accuracy:.4f}")
```

Accuracy: 0.6560

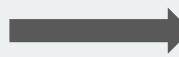
“

현재 예상치는 66%로, 모델이 조 자체 목표치인 70% 비해 낮음.  
개선을 통해 더 높은 정확도를 달성하여 리스크 평가 정확성을 극대화하고,  
신뢰할 수 있는 대출 가이드라인을 제공하고자 하는 목표 달성을 위해 여러  
머신러닝 방안을 탐색함.

”

‘알고리즘 모델’을 적용해보자!

How?



1. RandomForest 모델
2. XGBoost 모델
3. 로지스틱 선형회귀 모델
4. 그라디언트 부스팅 모델
5. SVM 모델

“

RandomForest란? → 데이터를 보고 결정을 도와주는 숲

여러 개의 Decision Tree를 생성하고 그 결과를 종합하여 더 나은 결과를 예측하는 방법

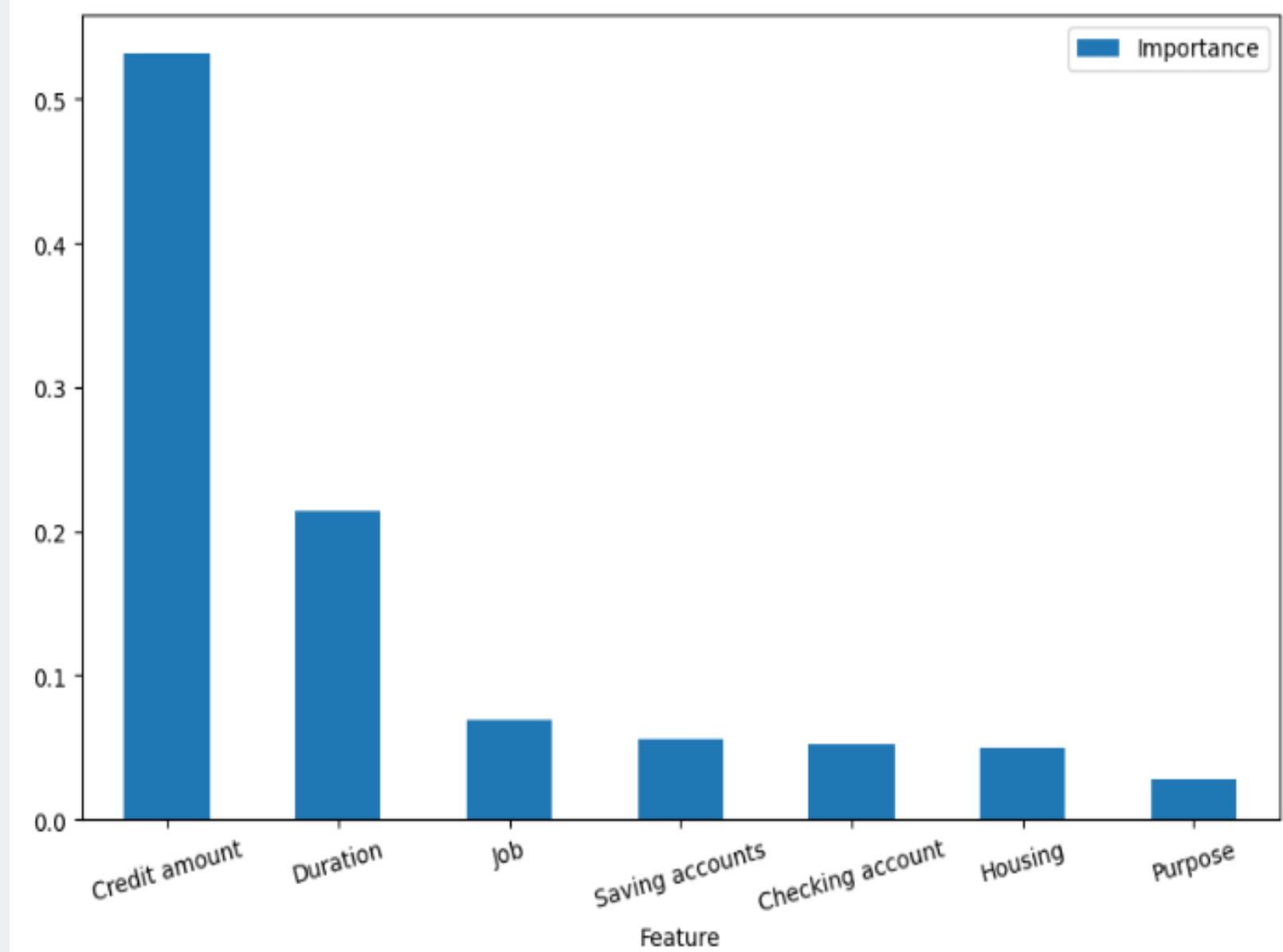
### 랜덤포레스트를 선택한 이유는?

- 랜덤 포레스트 모델은 데이터가 복잡하고 다양한 특성을 가지고 있어도 잘 처리할 수 있음.
- 또한 여러 특성 간의 관계를 고려해서 각 특성이 예측에 얼마나 중요한지 평가할 수 있음.
- 그리고 다양한 형태의 데이터에 사용이 가능하였고, 높은 정확도와 안정성을 제공함.
- 또한, 과적합(훈련 데이터에만 지나치게 잘 맞는 현상)이 잘 발생하지 않을 수 있는 결과를 제공함.

”

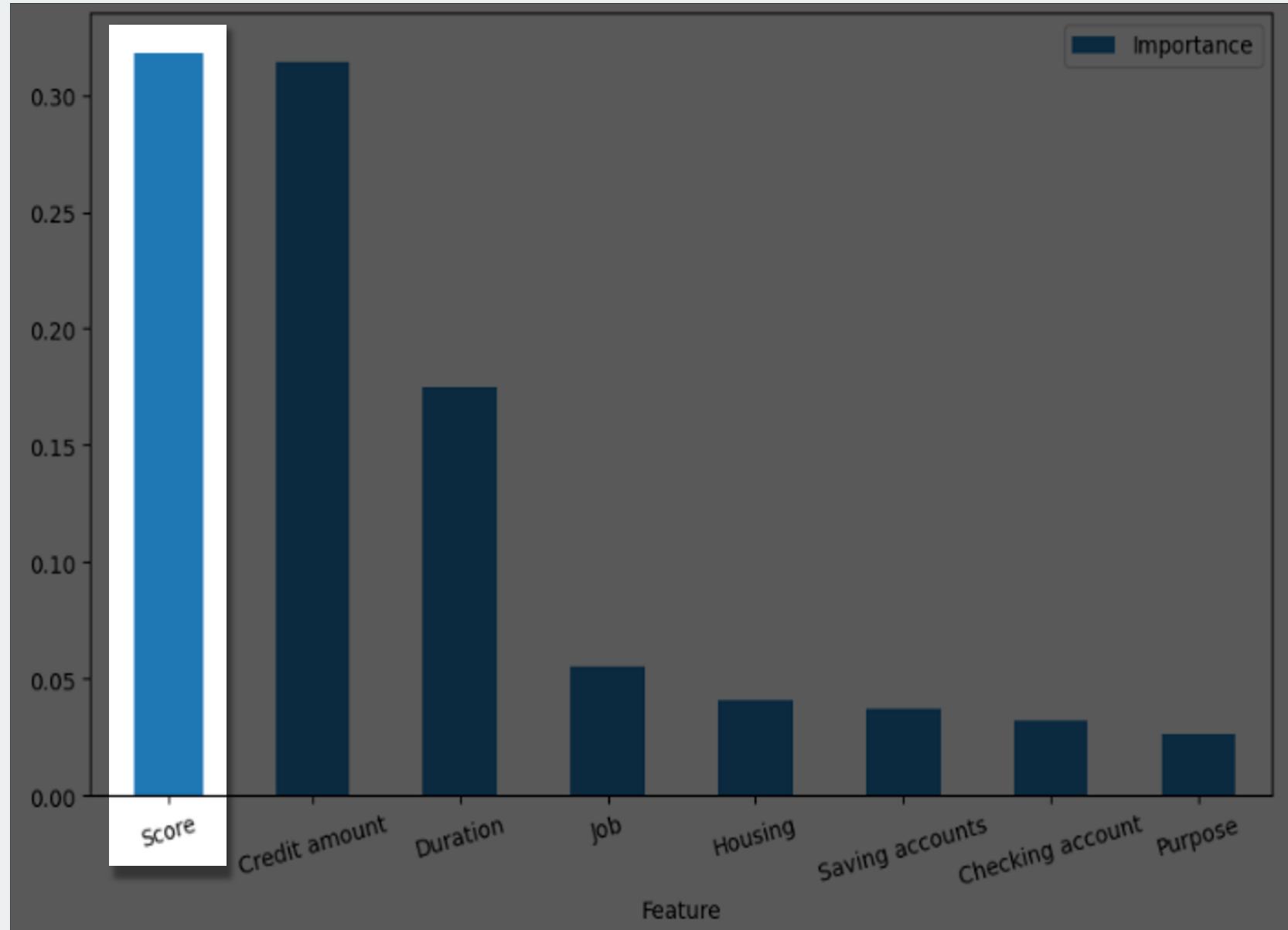
<Score 포함 전>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Credit amount'였음



<Score 포함 후>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Score'였음



## Score 포함여부에 따른 예측 결과

65%  
정확도

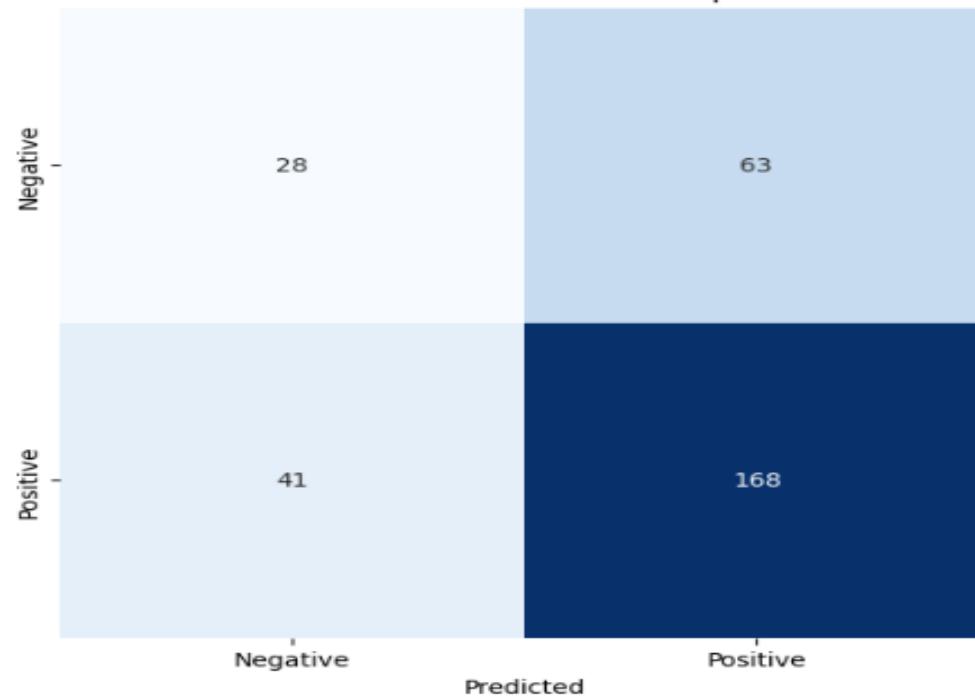
Score 컬럼을 포함하지 않은 모델의 정확도는 65%로 나타남

66%  
정확도

Score 컬럼을 포함한 모델의 정확도 외에도 risk 0에서 전반적인 향상을 보임

Classification Report:				
	precision	recall	f1-score	support
0	0.41	0.31	0.35	91
1	0.73	0.80	0.76	209
accuracy		0.65		300
macro avg	0.57	0.56	0.56	300
weighted avg	0.63	0.65	0.64	300

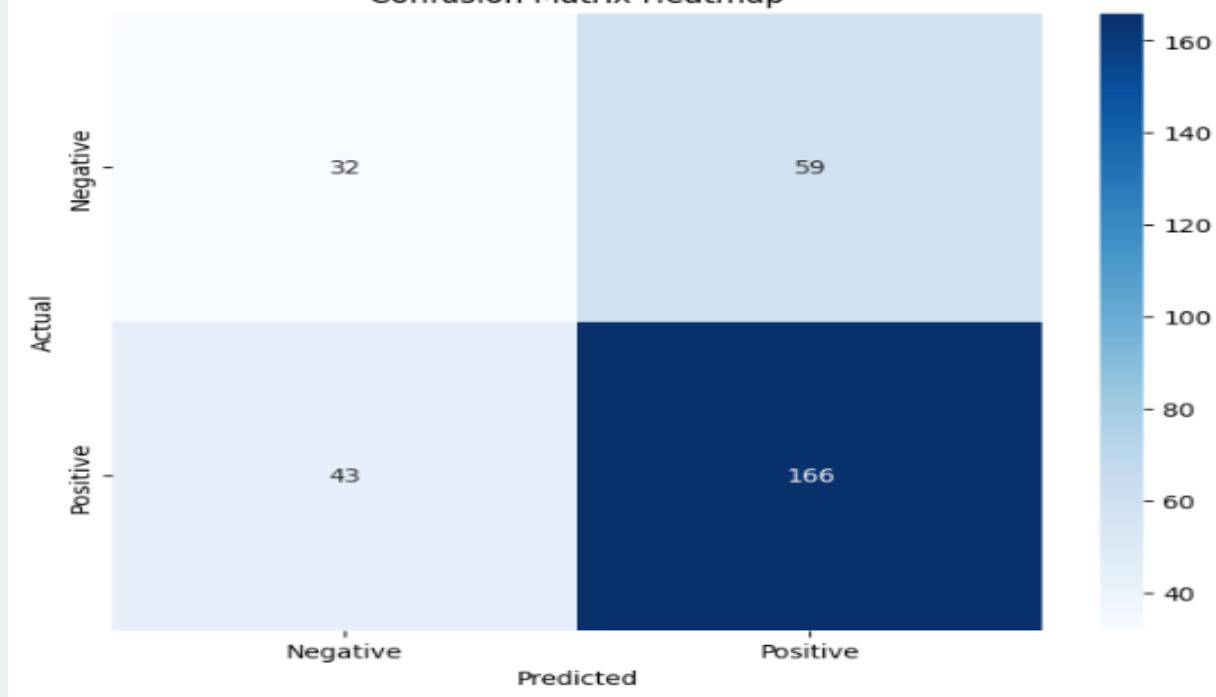
Confusion Matrix Heatmap



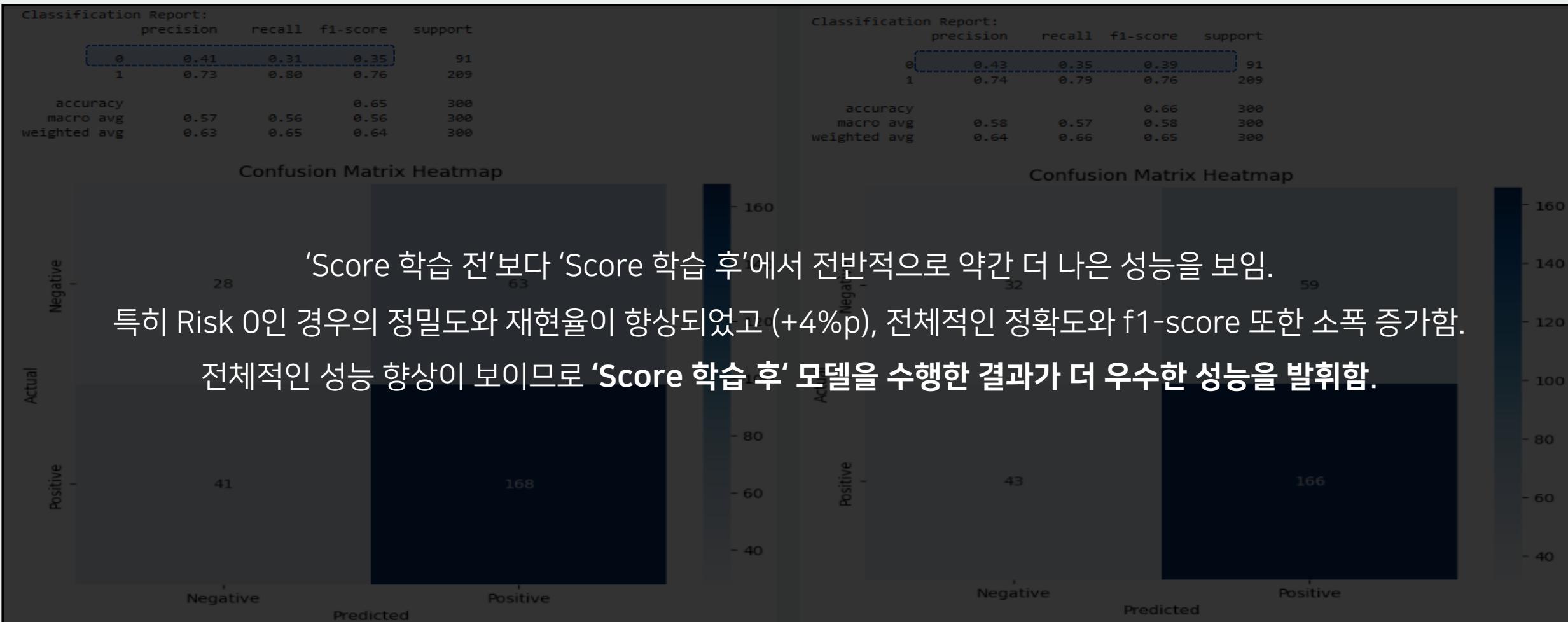
&lt;Score 학습 전&gt;

Classification Report:				
	precision	recall	f1-score	support
0	0.43	0.35	0.39	91
1	0.74	0.79	0.76	209
accuracy		0.66		300
macro avg	0.58	0.57	0.58	300
weighted avg	0.64	0.66	0.65	300

Confusion Matrix Heatmap



&lt;Score 학습 후&gt;



&lt;Score 학습 전&gt;

&lt;Score 학습 후&gt;

“

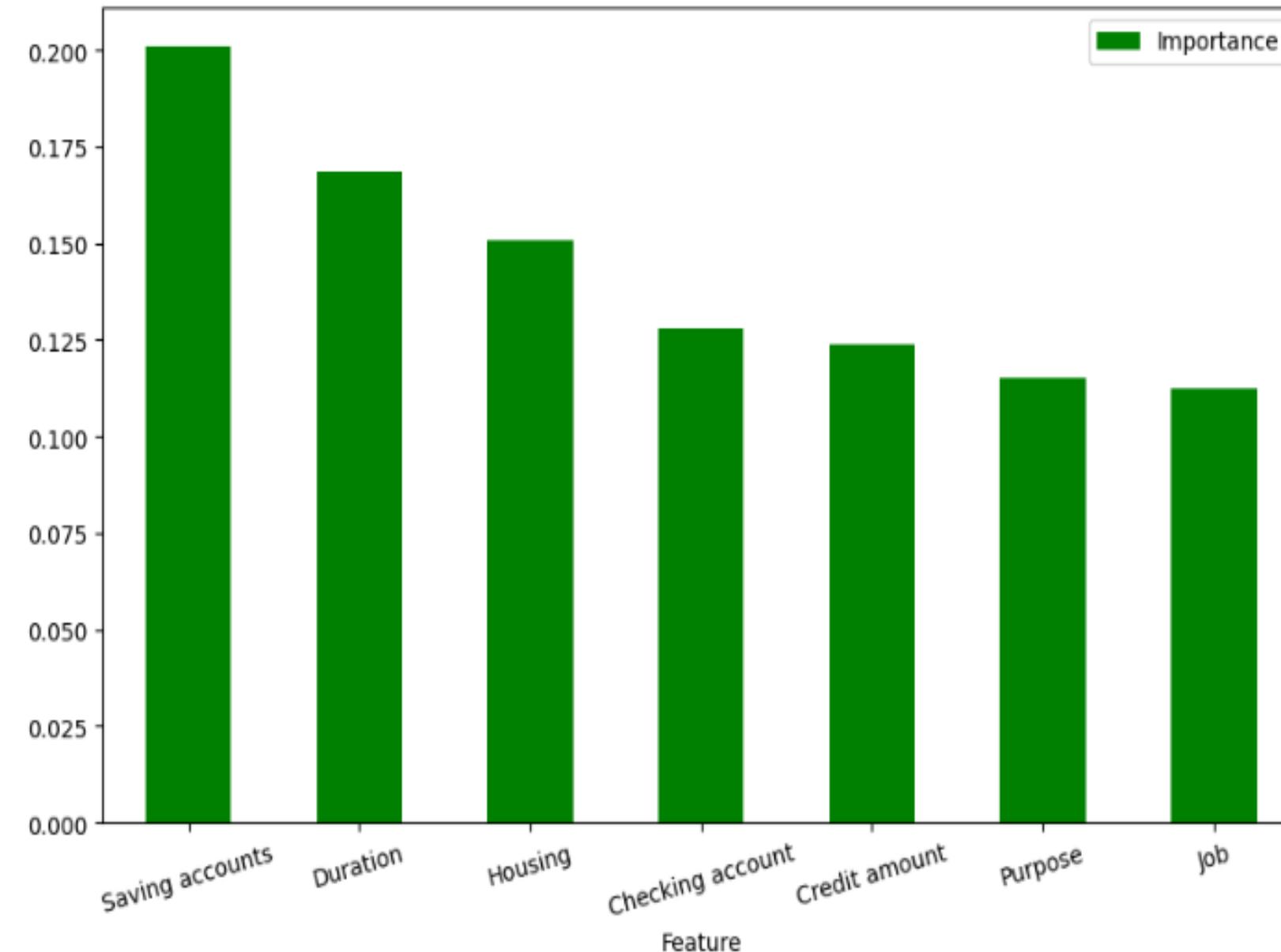
XGBoost란? → 데이터를 보고, 패턴을 찾아내는 똑똑한 시스템

여러 개의 약한 학습기(주로 의사결정나무)를 순차적으로 결합하여  
강력한 예측 모델을 만드는 알고리즘

XGBOOST를 선택한 이유는?

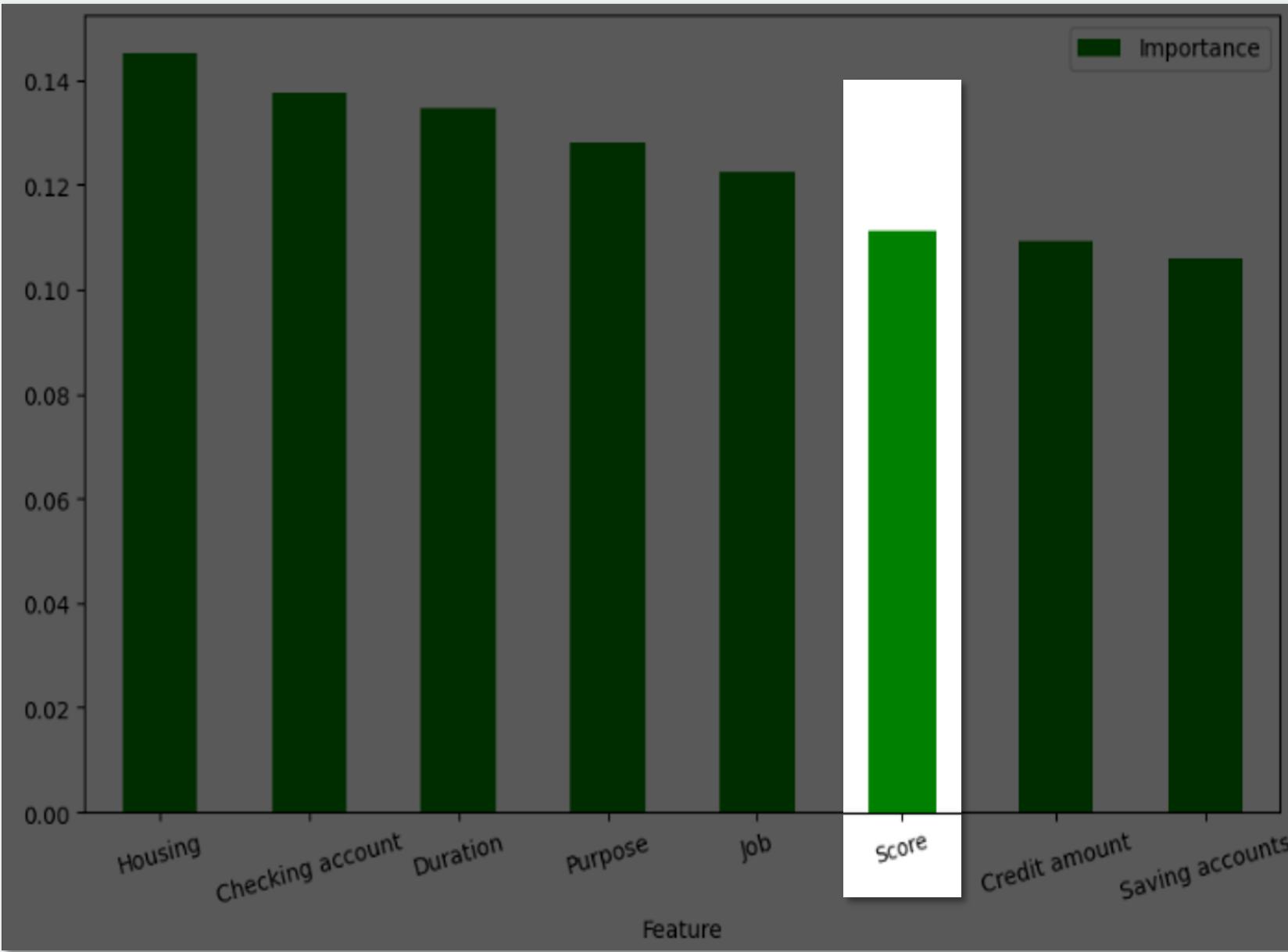
- 복잡한 데이터 구조처리에 강하여 신용등급예측 문제에서 여러 특성이 서로 영향을 미칠 가능성이 높은 요인을 처리할 수 있으리라 판단함.
- 결측치를 자동으로 처리할 수 있고 데이터 부족 상황에서 성능을 유지하여, 회귀, 분류, 예측 등 다양한 문제 상황에서 유연하게 적용 가능한 장점이 있음.

”



<Score 포함 전>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Saving accounts'였음



<Score 포함 후>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Housing'이었고, 'Score'는 후순위였음

## Score 포함여부에 따른 예측 결과

63%  
정확도

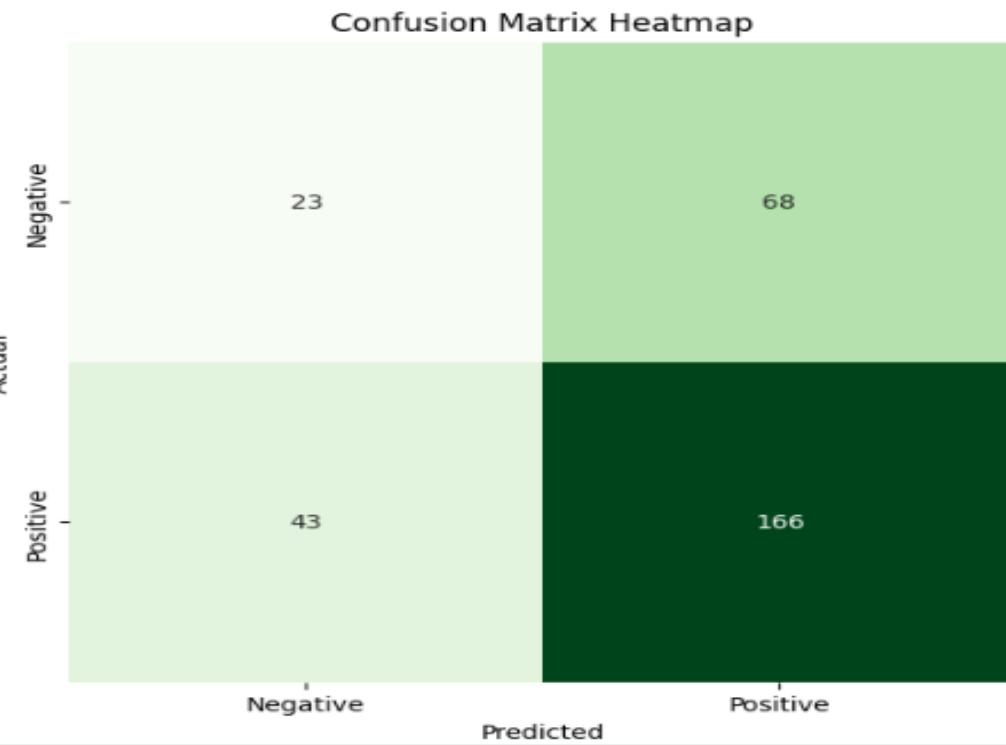
Score 컬럼을 포함하지 않은 모델의 정확도는 63%로 나타남.

66%  
정확도

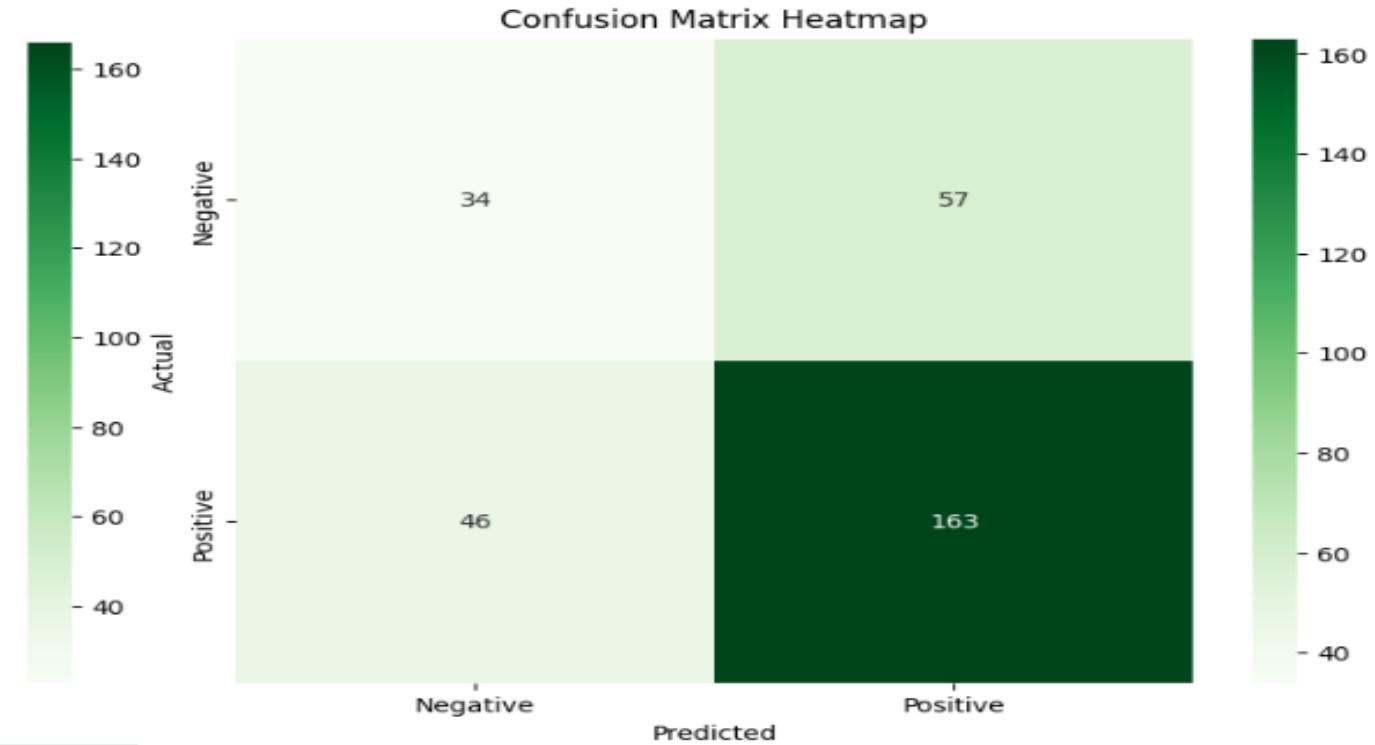
Score 컬럼을 포함한 모델에서 RandomForest 모델과 동일하게  
risk 0에서 전반적인 향상이 나타남.

Classification Report:				
	precision	recall	f1-score	support
0	0.35	0.25	0.29	91
1	0.71	0.79	0.75	209
accuracy		0.63	0.63	300
macro avg	0.53	0.52	0.52	300
weighted avg	0.60	0.63	0.61	300

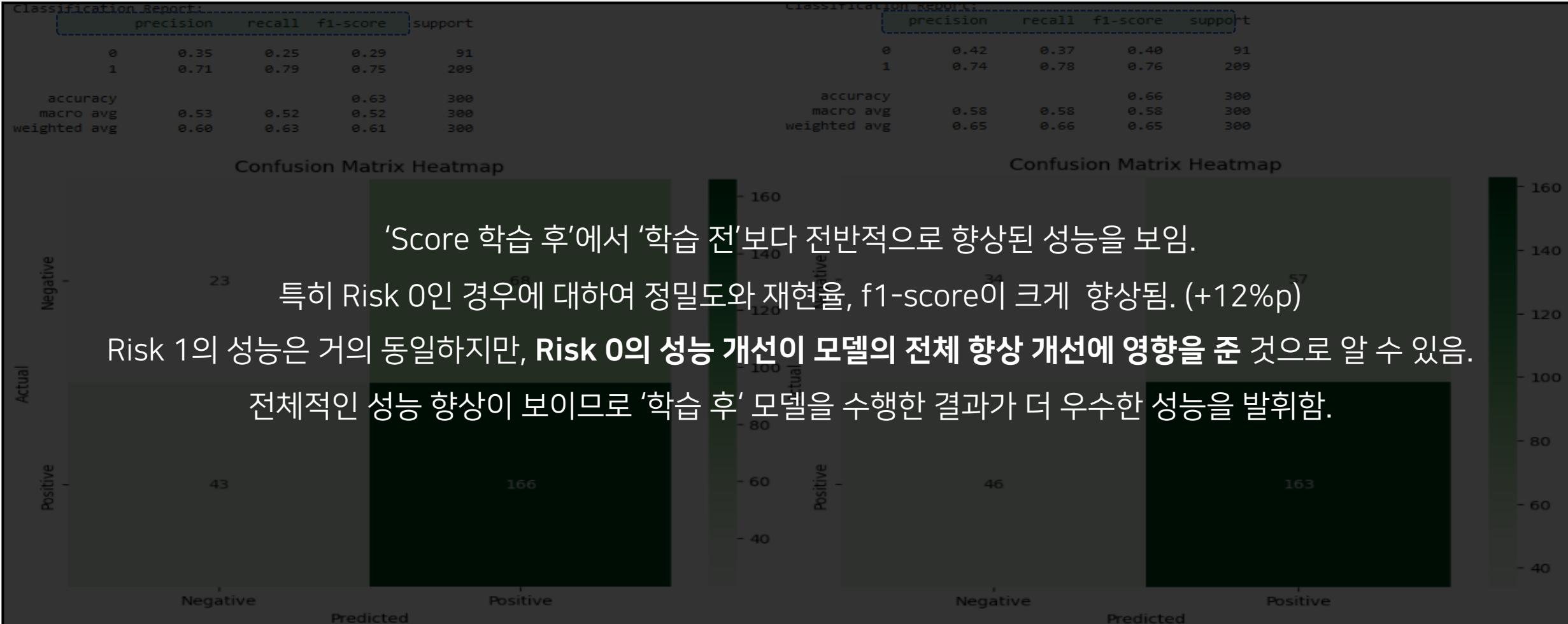
Classification Report:				
	precision	recall	f1-score	support
0	0.42	0.37	0.40	91
1	0.74	0.78	0.76	209
accuracy		0.66	0.66	300
macro avg	0.58	0.58	0.58	300
weighted avg	0.65	0.66	0.65	300



&lt;Score 학습 전&gt;



&lt;Score 학습 후&gt;



&lt;Score 학습 전&gt;

&lt;Score 학습 후&gt;

“ 로지스틱 선형회귀란?



모든 가능성을 파악해 그럴듯한 선택을  
예측하는 저울

데이터를 보고 결과가 특정 범주에 속할 확률을 계산해, 분류 문제를 해결하는 알고리즘

로지스틱 선형회귀를 선택한 이유는?

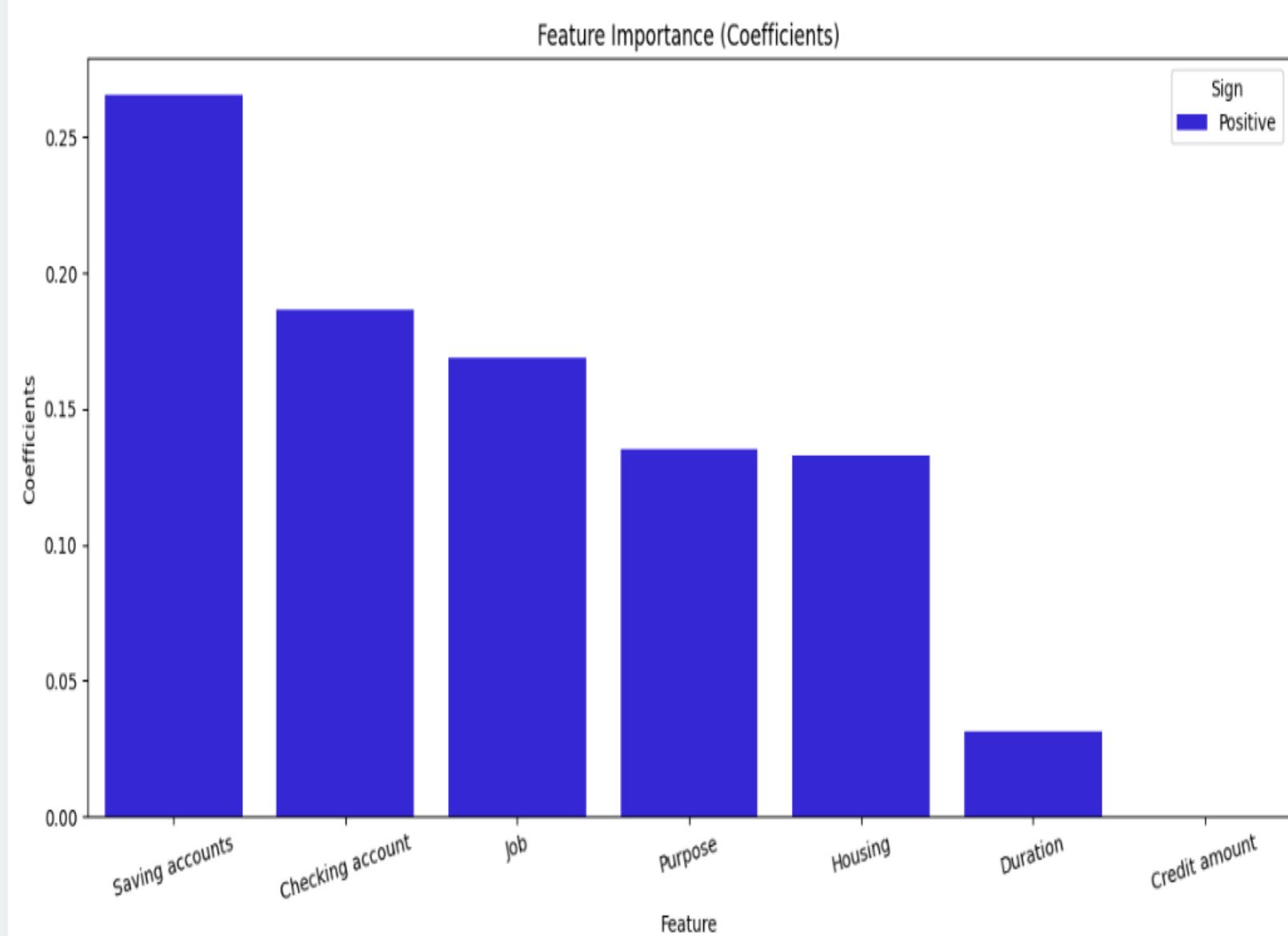
해석이 간단하고 직관적이며, 확률 기반의 예측을 제공하여 신용 점수에서 어떤 요인이 대출 승인이 되는지, 혹은 거절이 되는지 설명이 가능하여 결정기준에 대한 신뢰성을 제공할 수 있음.

빠르고 효율적인 계산이 특징이라 대규모 데이터에서도 사용이 적합함.

”

### <Score 포함 전>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Saving accounts'였음

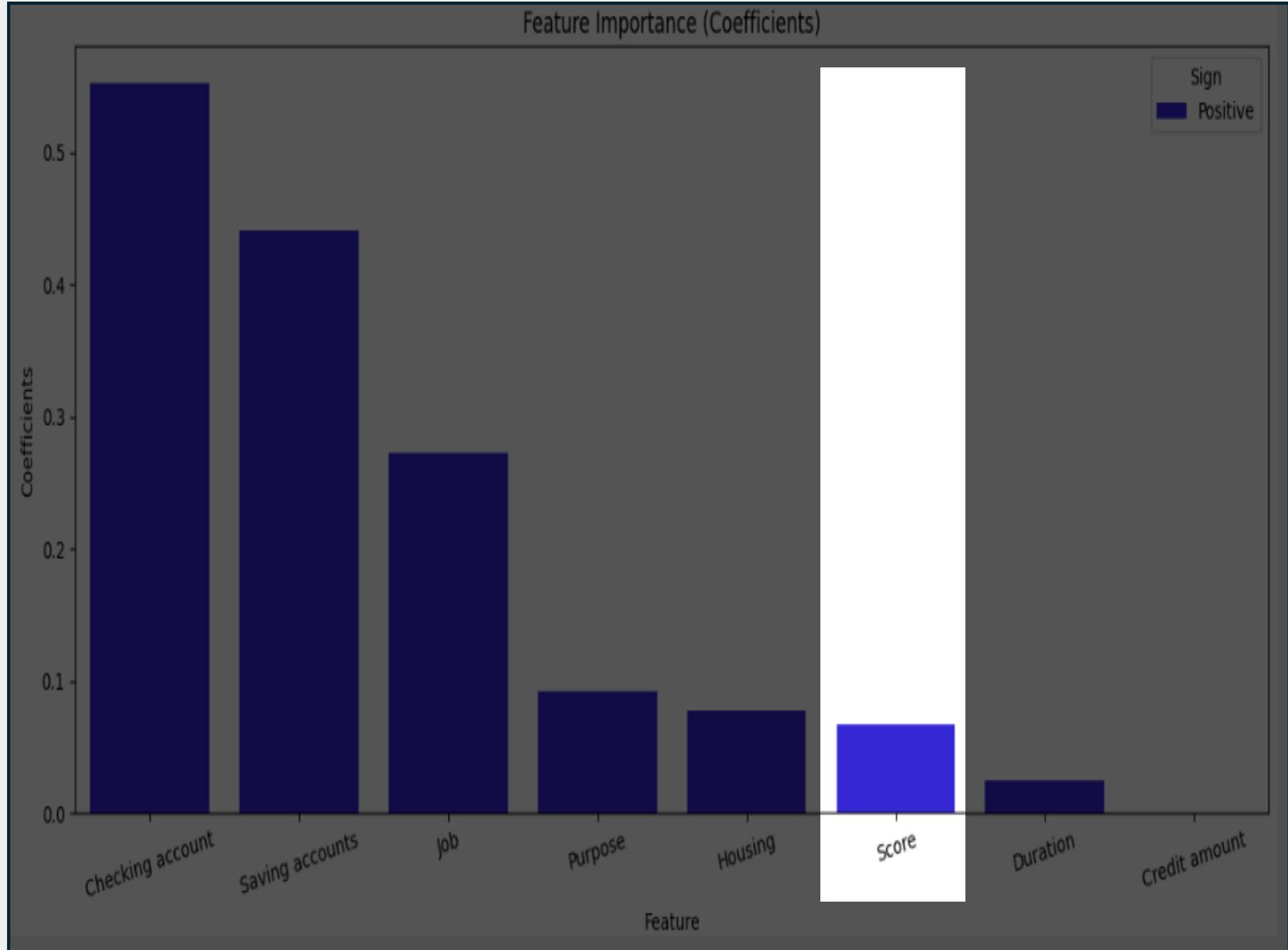


<Score 포함 후>

Risk와의 중요성을 비교했을 때

가장 높은 관련성이 있는 컬럼은

'Checking account'였고, 'Score'는 후순위였음



Class 0 (실제로 'Bad'인 경우)

Precision(정밀도) : 0.55

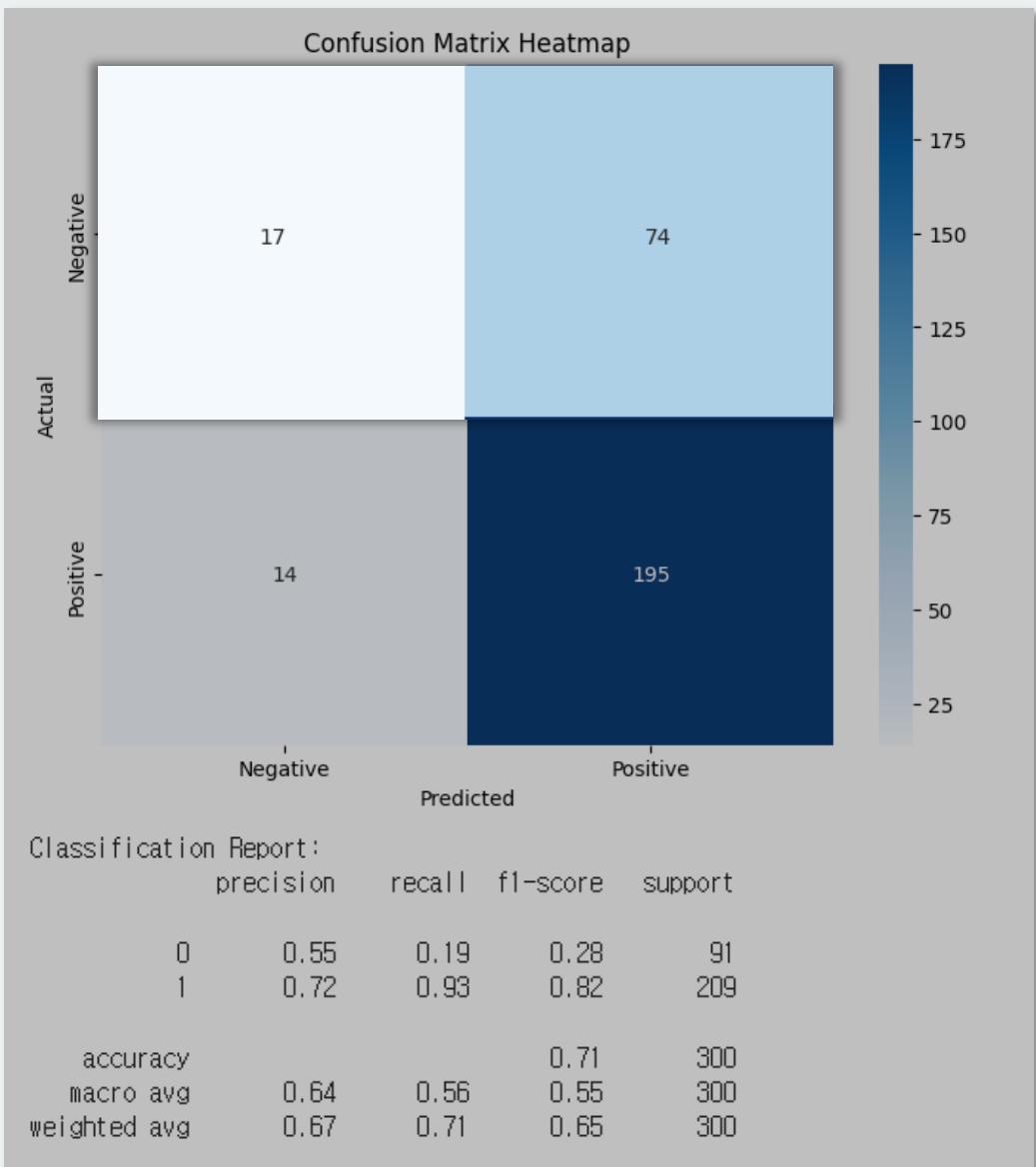
예측을 'Bad'로 한 것 중 실제로 'Bad'인 비율 → 55%가 실제로 'Bad'

Recall(재현율) : 0.19

실제 'Bad'인 것 중에서 모델이 'Bad'로 예측한 비율 → 19%만 'Bad'로 올바르게 예측

F1-score : 0.28 → 모델이 'Bad'를 예측할 때의 전반적인 성능

Support : 91 → 실제 'Bad'로 분류된 샘플수



### >>> Case3. 로지스틱 회귀

#### Class 1 (실제로 'Good'인 경우)

- Precision(정밀도) : 0.72

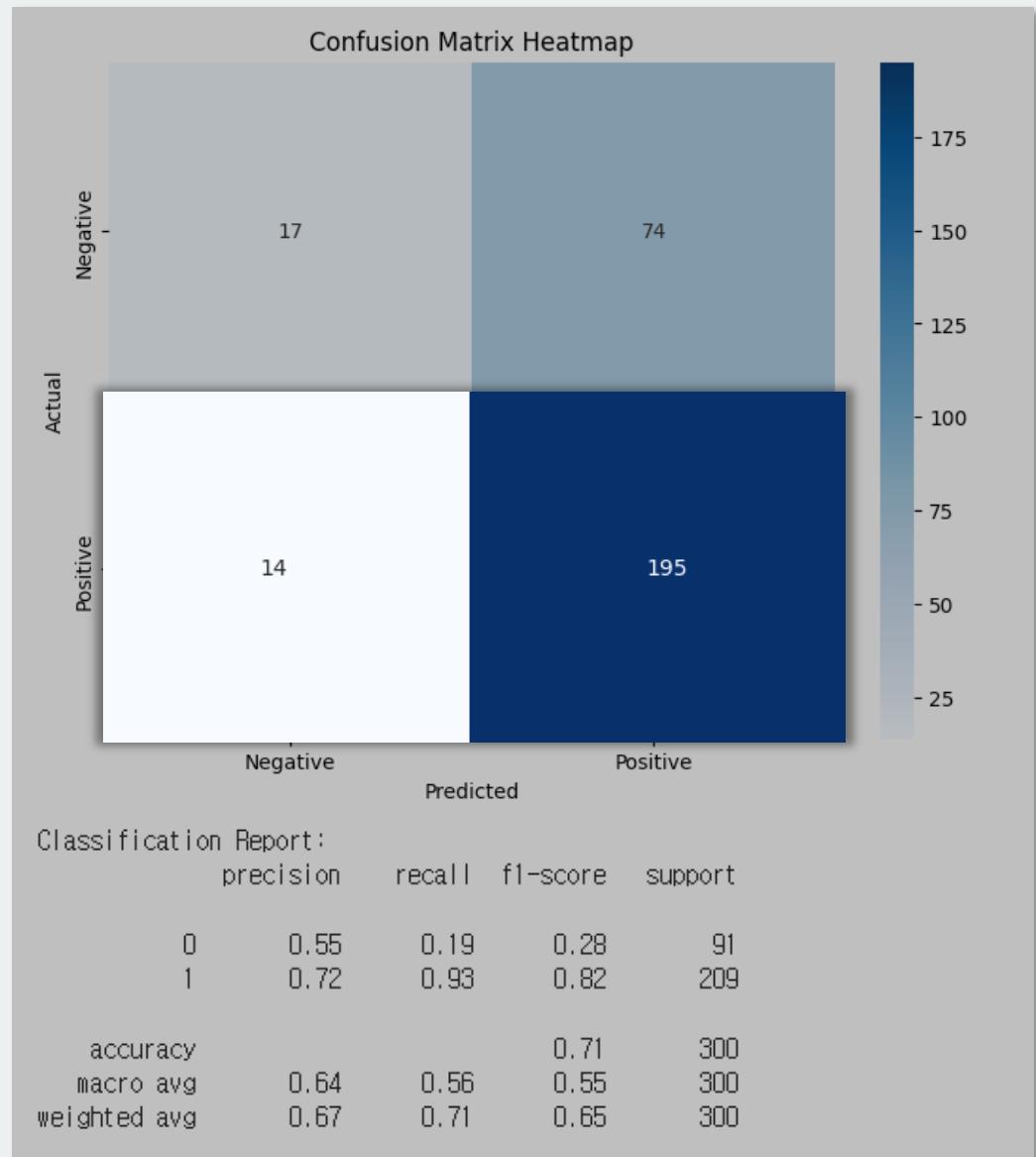
예측을 'Good'으로 한 것 중 실제로 'Good'인 비율 → 72%가 실제로 'Good'

- Recall(재현율) : 0.93

실제 'Good'인 것 중에서 'Good'으로 예측한 비율 → 93%가 'Good'으로 예측

- F1-score : 0.82 → 모델이 'Good'를 예측할 때의 전반적인 성능

- Support : 209 → 실제 'Good'으로 분류된 샘플수



## >>> Case3. 로지스틱 회귀

### 전체 지표

- Accuracy : 0.71

전체 샘플 중에서 올바르게 예측한 비율 → 71%의 정확도

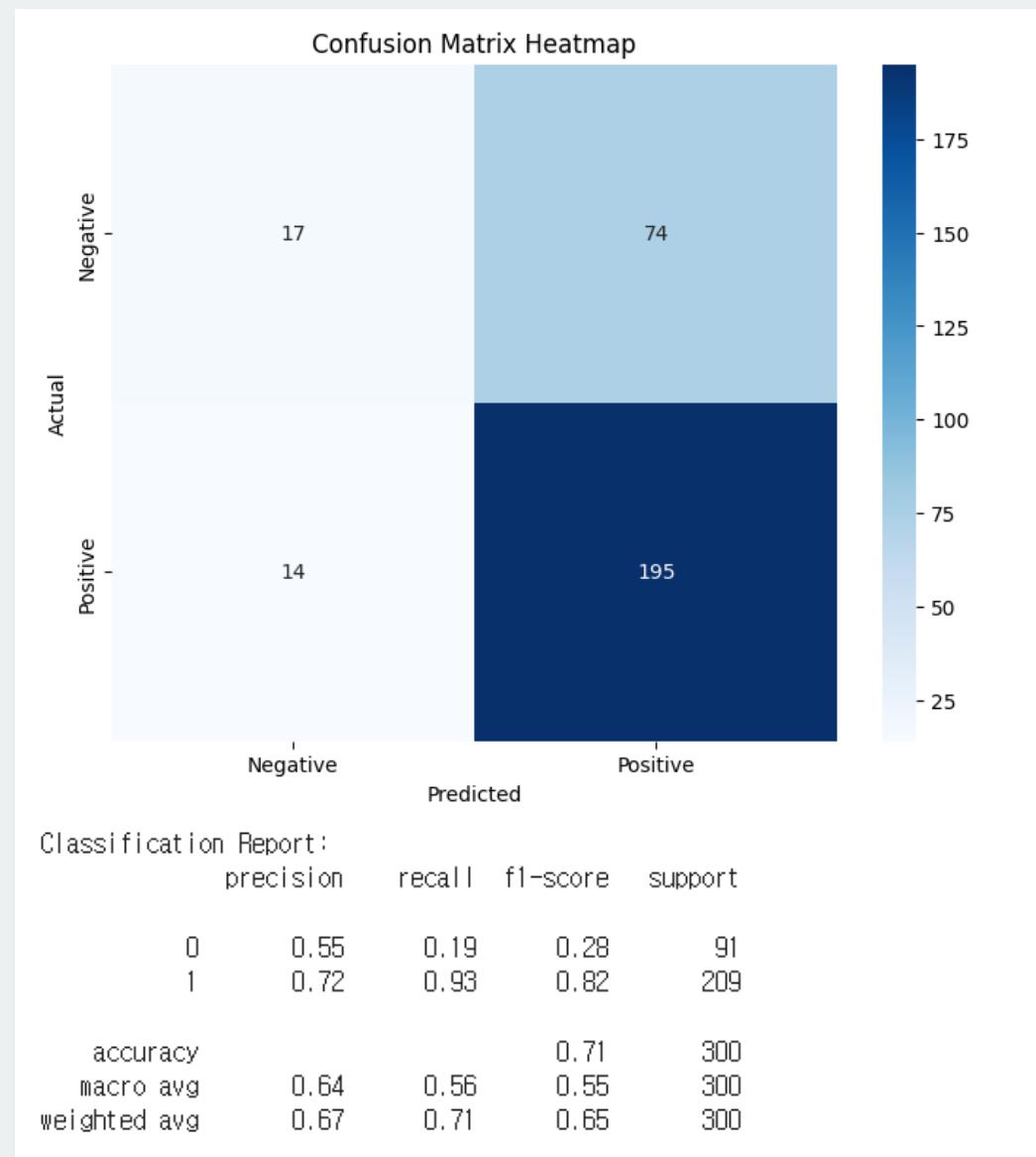
### 종합 해석

'Good'에 대해서는 높은 정밀도와 재현율을 보이지만

'Bad'에 대해서는 재현율이 낮아 잘 예측하지 못한다.

전반적인 정확도는 71%로 양호하지만,

두 클래스 간의 불균형 때문에 'Bad' 예측 성능이 상대적으로 낮다.



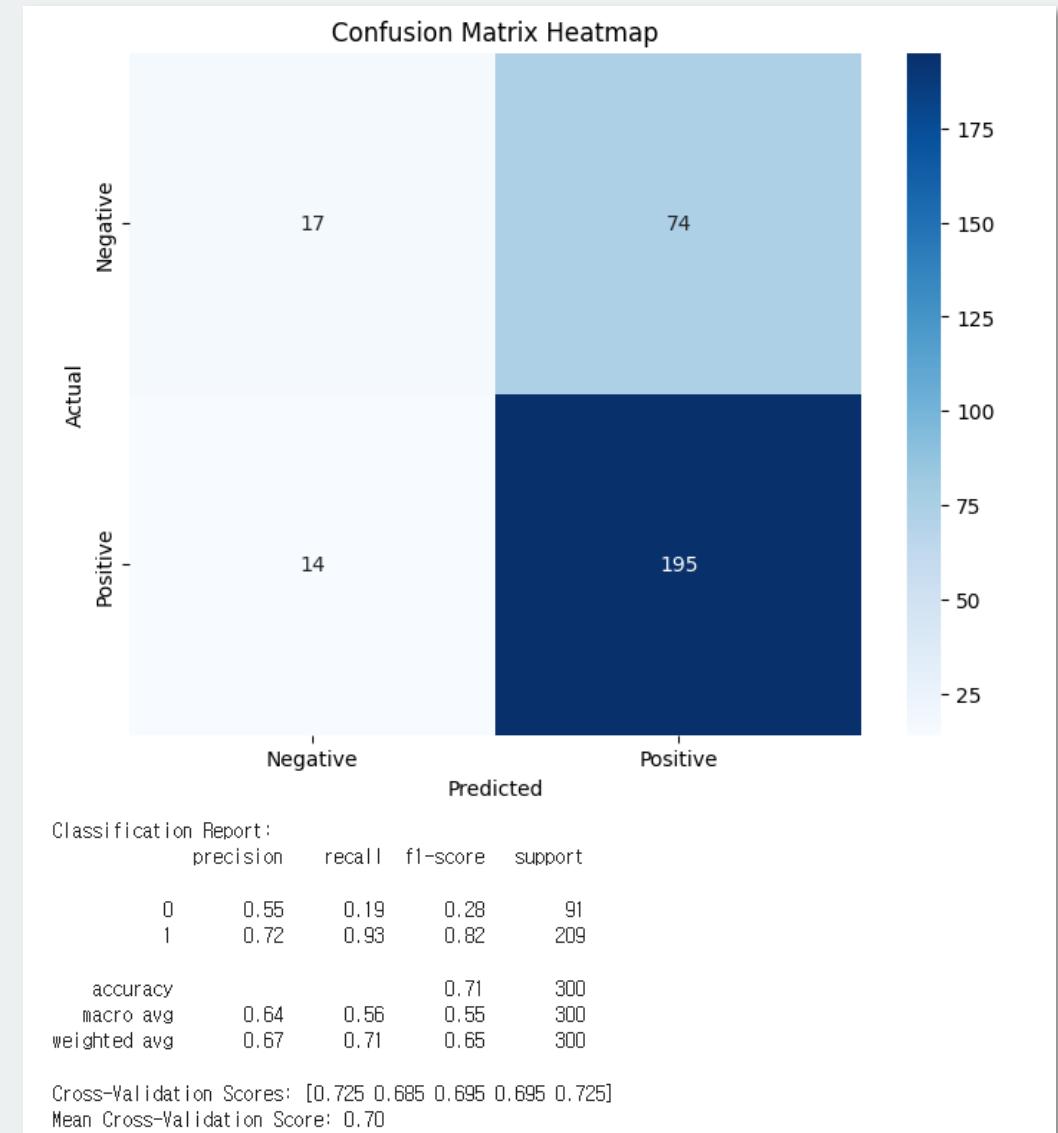
## >>> Case3. 로지스틱 회귀

Bad 데이터가 각 테스트 데이터에 적절히 포함되도록

**불균형한 데이터에 대한 안정적인 평가를 위해 K-겹 교차검증 활용**

### K-Fold Cross-Validation Scores :

- 첫 번째 Fold : 0.725
- 두 번째 Fold : 0.685
- 세 번째 Fold : 0.695
- 네 번째 Fold : 0.695
- 다섯 번째 Fold : 0.725
- 평균 정확도: 0.70



\* K-겹 교차검증 : 전체 데이터를 K개의 부분(Fold)로 나누어 각 Fold를 한 번씩 검증 세트로 사용하고, 나머지 K-1개의 Fold는 훈련 데이터로 번갈아 사용하여 모델을 훈련하고 평가하는 기법

“

그라디언트 부스팅이란? → 실수를 분석하며 점점 더 똑똑해지는 학습 팀

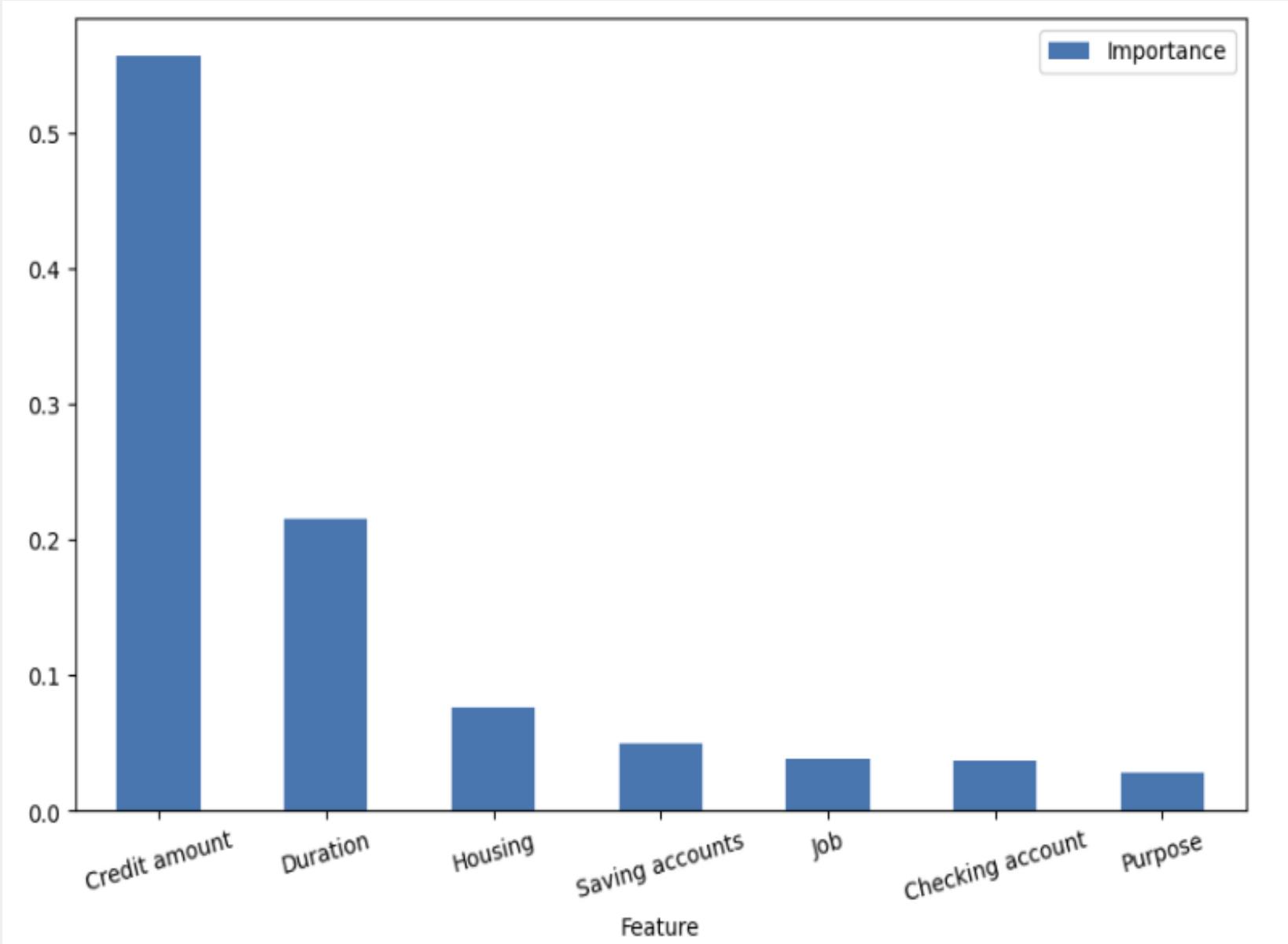
이전 모델의 오류를 순차적으로 보완해 최적의 예측 결과를 만드는 양상을 알고리즘

그라디언트 부스팅을 선택한 이유는?

데이터에 전처리가 까다로운 신용 데이터를 다룰 때 안정적인 성능을 보장하고, 커스텀마이징을 통해 하이퍼 파라미터를 조정하여 모델의 과적합 방지를 통해 신용 평가 모델의 유연성을 극대화함.

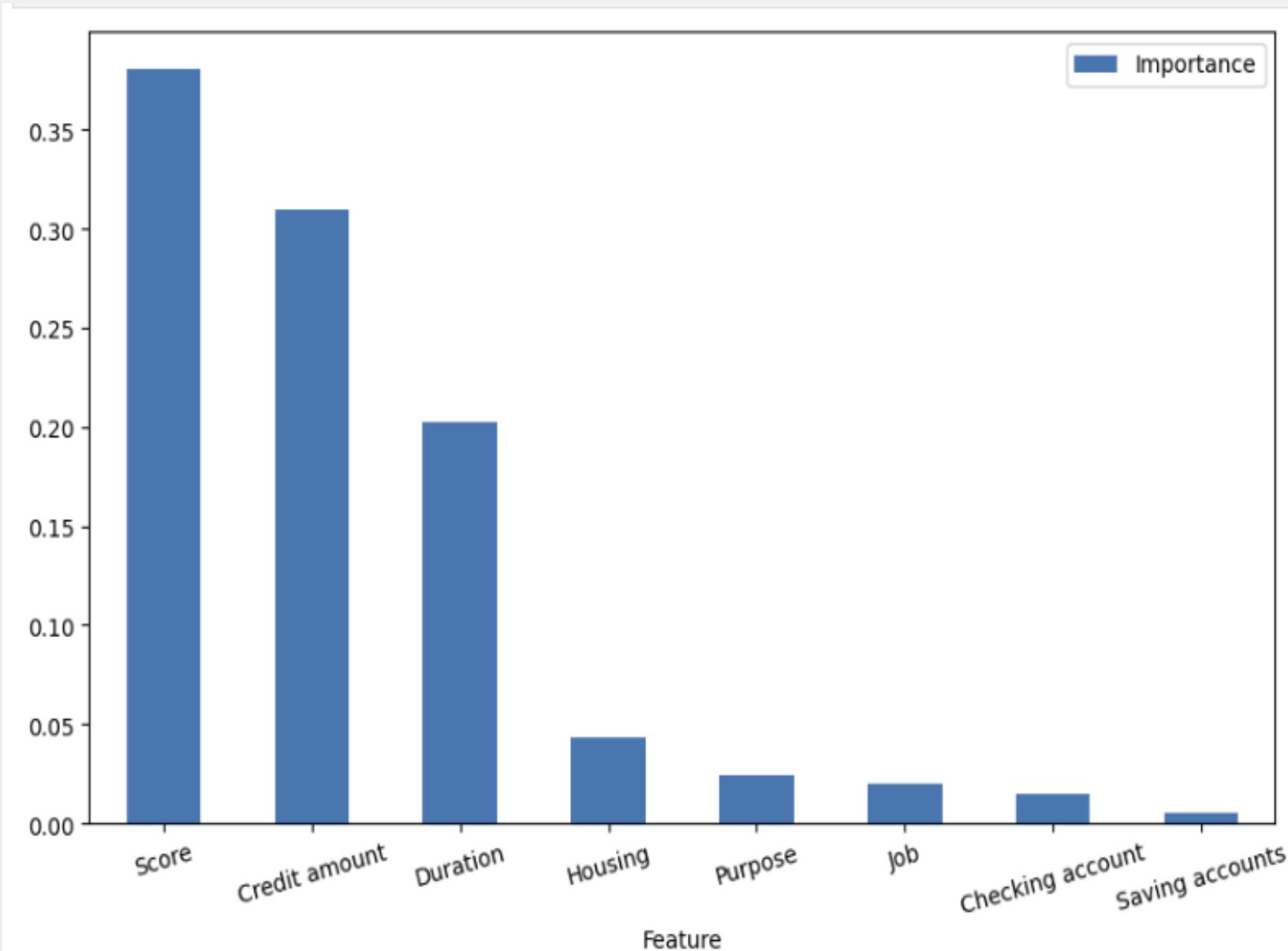
이러한 점에서 대출 승인 여부나 신용 점수 같은 중요한 의사결정에 필요한 정확한 결과를 도출할 수 있음.  
또한, 잔여 오차를 최소화하여 학습하여, 복잡한 패턴을 효과적으로 모델링이 가능하다는 장점이 있음.

”



<Score 포함 전>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Credit amount'였음.



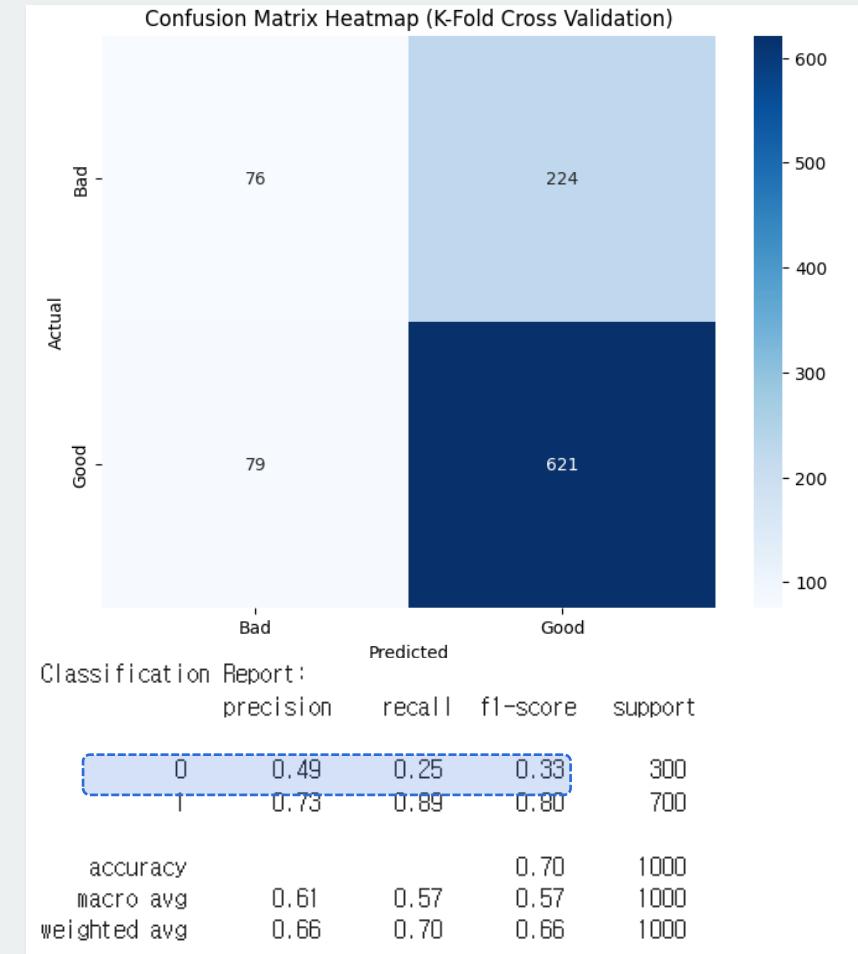
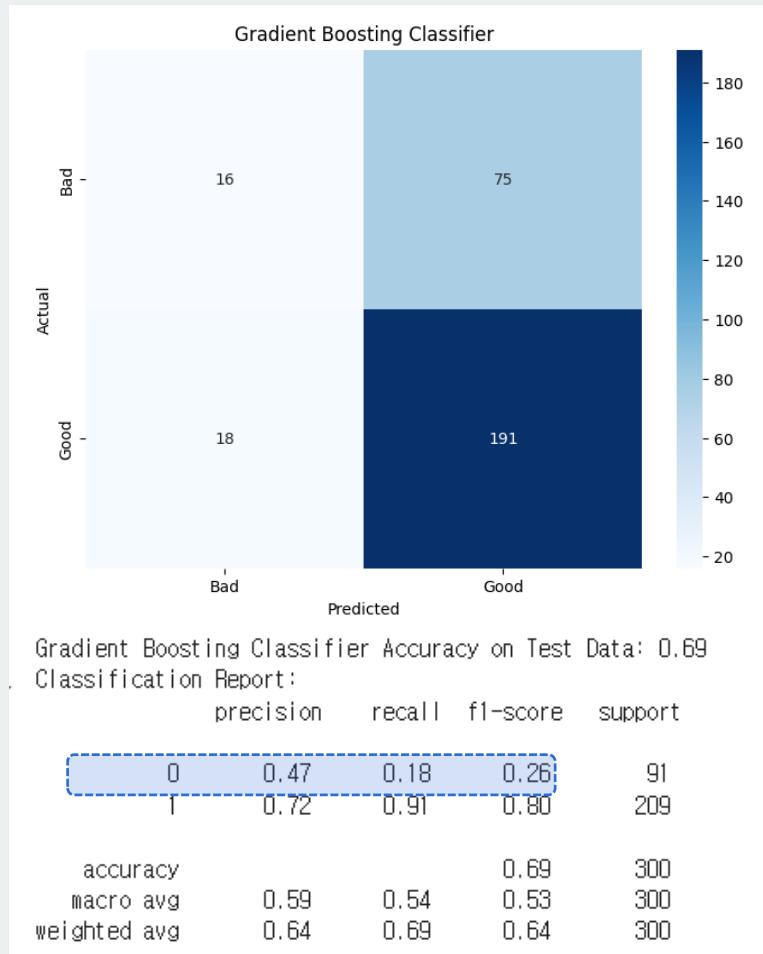
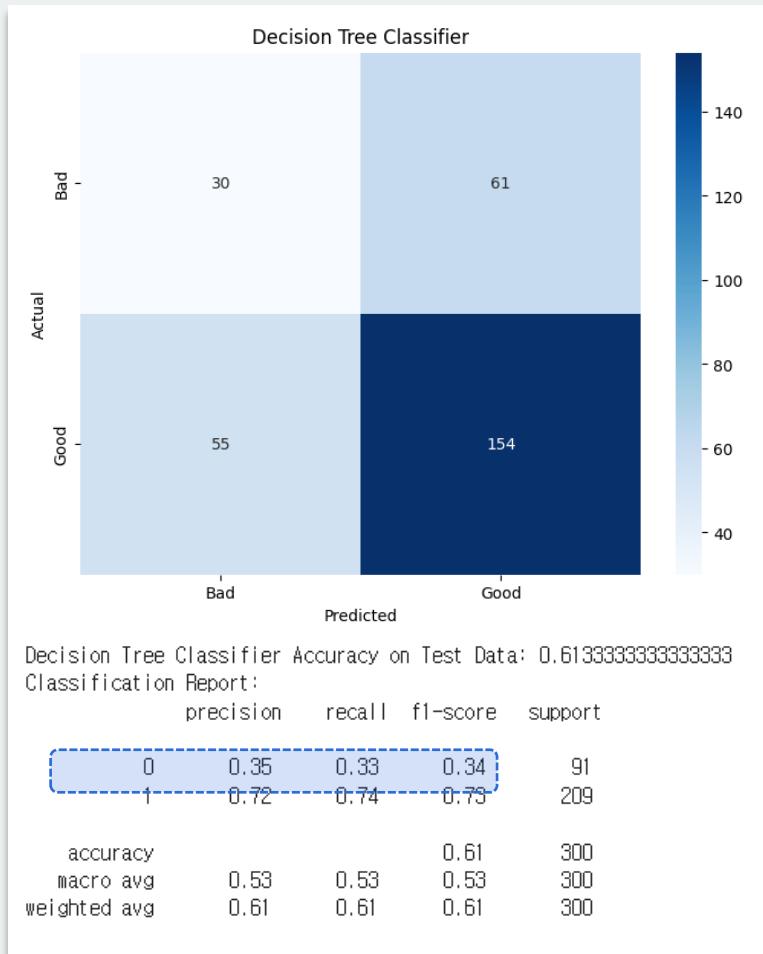
<Score 포함 후>

Risk와의 중요성을 비교했을 때  
가장 높은 관련성이 있는 컬럼은  
'Score'였음.

## >>> Case4. 결정트리(그라디언트부스팅)

질문을 차례대로 던져가며 최종적인 결정을 내리는 방식

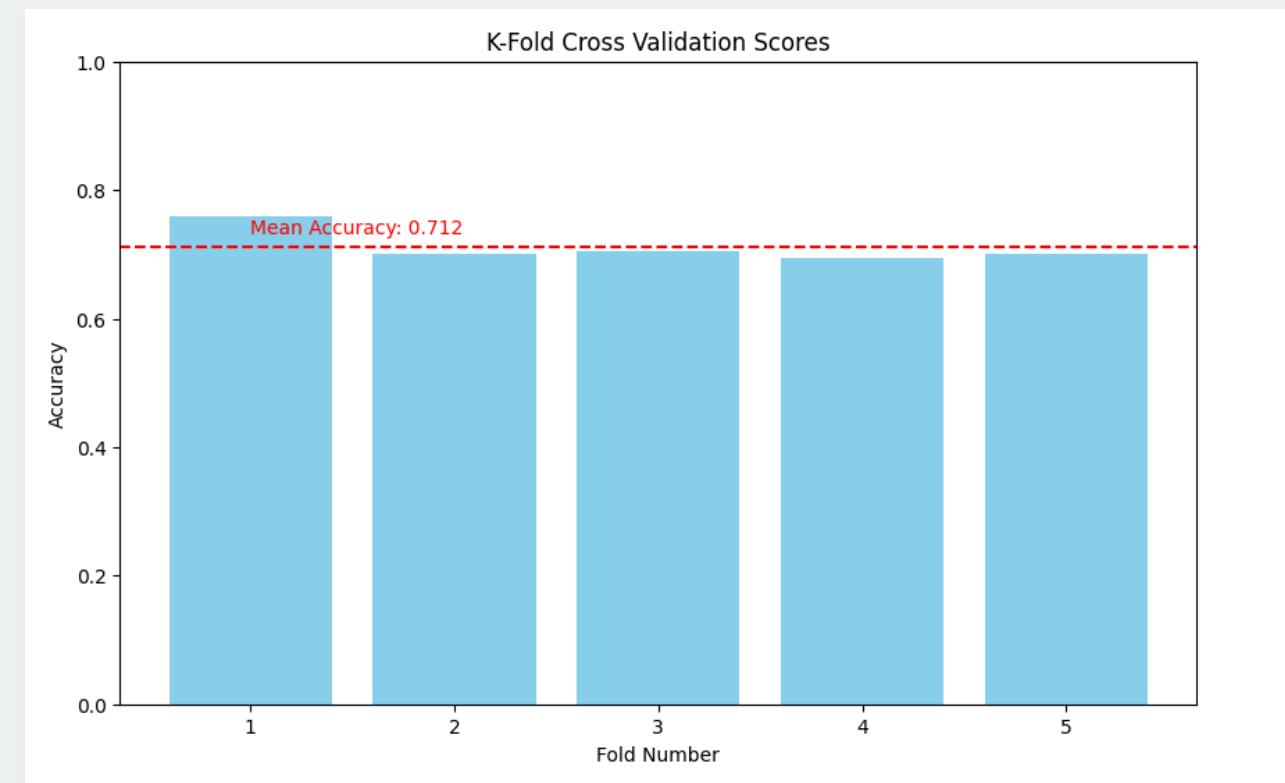
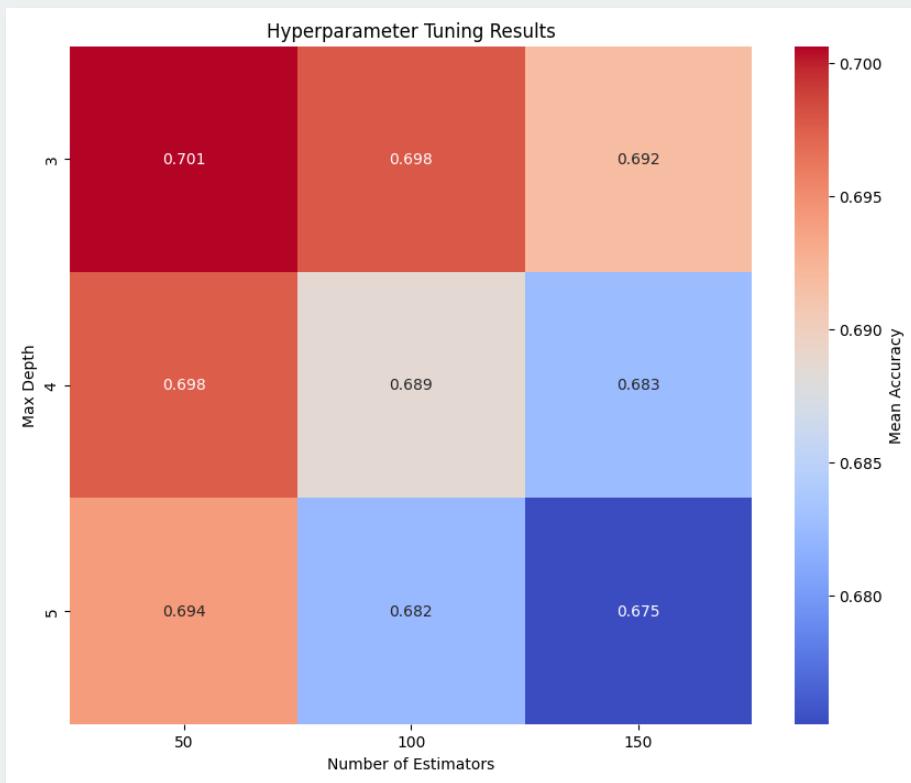
하나의 질문으로 결정을 이어가는 방식 -----> 여러가지 질문을 합쳐 예측을 보완 -----> K-Fold 교차 검증



## >>> Case4. 결정트리(그라디언트부스팅)

질문을 차례대로 던져가며 최종적인 결정을 내리는 방식

- 하이퍼 파라미터 튜닝 : 트리의 최대 깊이, 최소 샘플 수 등의 값 최적화



- Max Depth(개별 트리의 최대 깊이)가 3일 때, n\_estimators(트리의 수)가 50인 경우(결정 트리의 수) 가장 높은 정확도 (0.701)

[0.76, 0.7, 0.705, 0.695, 0.7]

“

SVM이란? ➡ 데이터를 잘 나눌 수 있는 최적의 경계선을 찾는 알고리즘

선형커널 - 선형적으로 구분 가능한 데이터를 처리하는 데 적합한 커널

다항식커널 - 특성 간의 곱셈이나 비선형적인 관계로 구분이 가능할 때 효과적

**RBF커널 - 비선형 데이터에 적합. 신용등급 예측 문제에서 다른 커널보다 더 나은 성능 보여줌**

시그모이드커널 - 보통 신경망 관련된 문제에서 사용

SVM – RBF커널을 선택한 이유는?

신용 평가 데이터는 여러 변수가 포함된 고차원 데이터로 이루어져 있음.

SVM은 차원이 높아질수록 성능을 유지하여 저희 가진 데이터를 효율적으로 분석할 수 있는 도구라 판단함.

특히, RBF커널은 데이터를 고차원으로 매핑한 후, 각 데이터 포인트 간의 관계를 비선형적으로 잘 구분할 수 있음.

이러한 특성이, 다른 커널에 비해 더 나은 성능과 예측의 정확도를 높이는 데 유리하다고 판단되어 적용함.

”

## >>> Case5. SVM (Support Vector Machine)

- 데이터를 두 그룹으로 나누는 최적의 결정 경계를 찾는 분류 알고리즘

### Class 0 (실제로 'Bad'인 경우)

- Precision(정밀도) : 0.83

예측을 'Bad'로 한 것 중 실제로 'Bad'인 비율 → 83%가 실제로 'Bad'

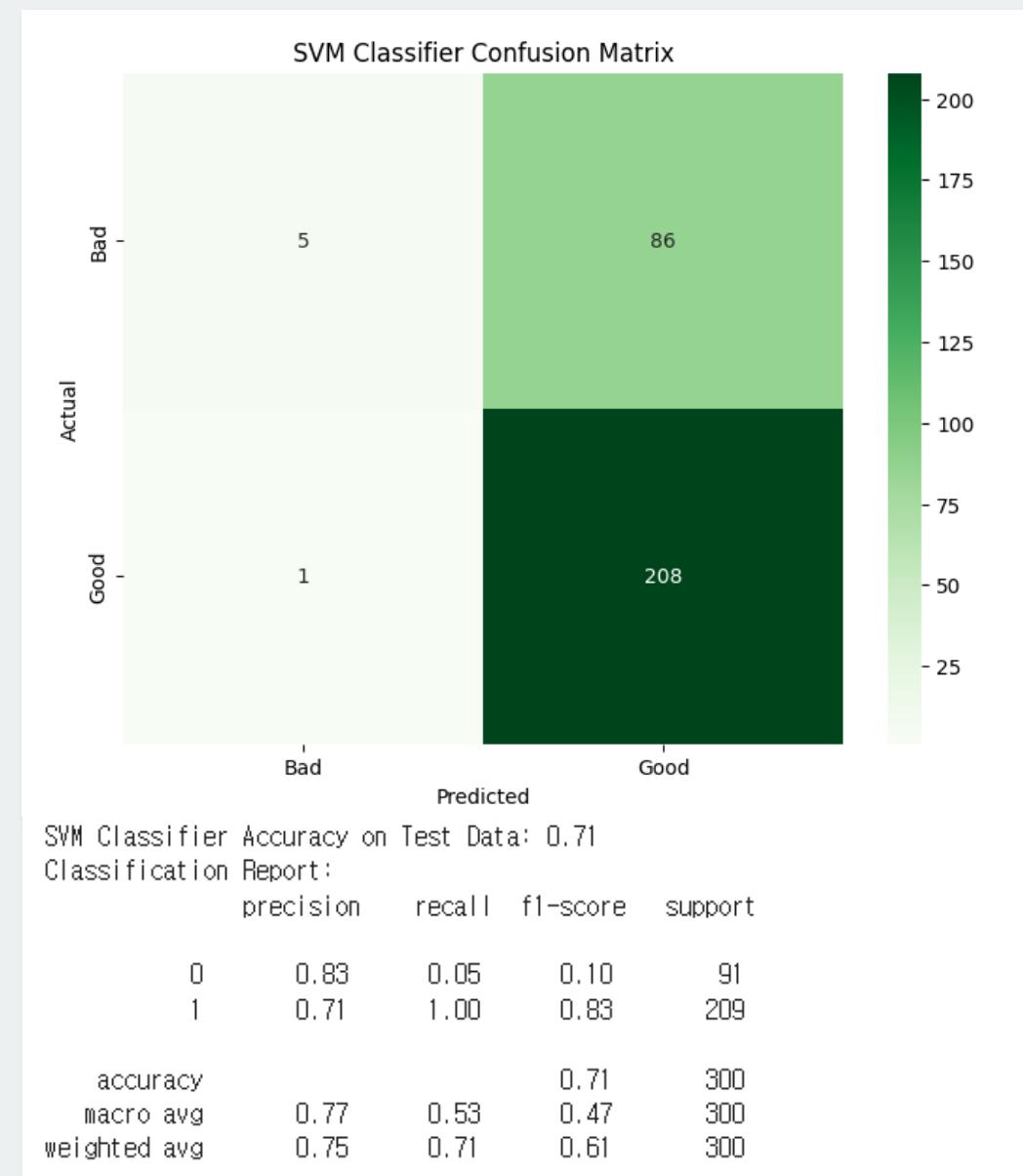
- Recall(재현율) : 0.05

실제 'Bad'인 것 중에서 모델이 'Bad'로 예측한 비율 → 5%만 'Bad'로 올바르게 예측

- F1-score : 0.10 → 모델이 'Bad'를 예측할 때의 전반적인 성능

- Support : 91 → 실제 'Bad'로 분류된 샘플수

\* 실제로 나쁨인 91개의 샘플 중 5개는 나쁨으로 정확하게 예측되었고, 86개는 좋음으로 잘못 예측



## >>> Case5. SVM (Support Vector Machine)

- 데이터를 두 그룹으로 나누는 최적의 결정 경계를 찾는 분류 알고리즘

### Class 1 (실제로 'Good'인 경우)

- Precision(정밀도) : 0.71

예측을 'Good'으로 한 것 중 실제로 'Good'인 비율 → 72%가 실제로 'Good'

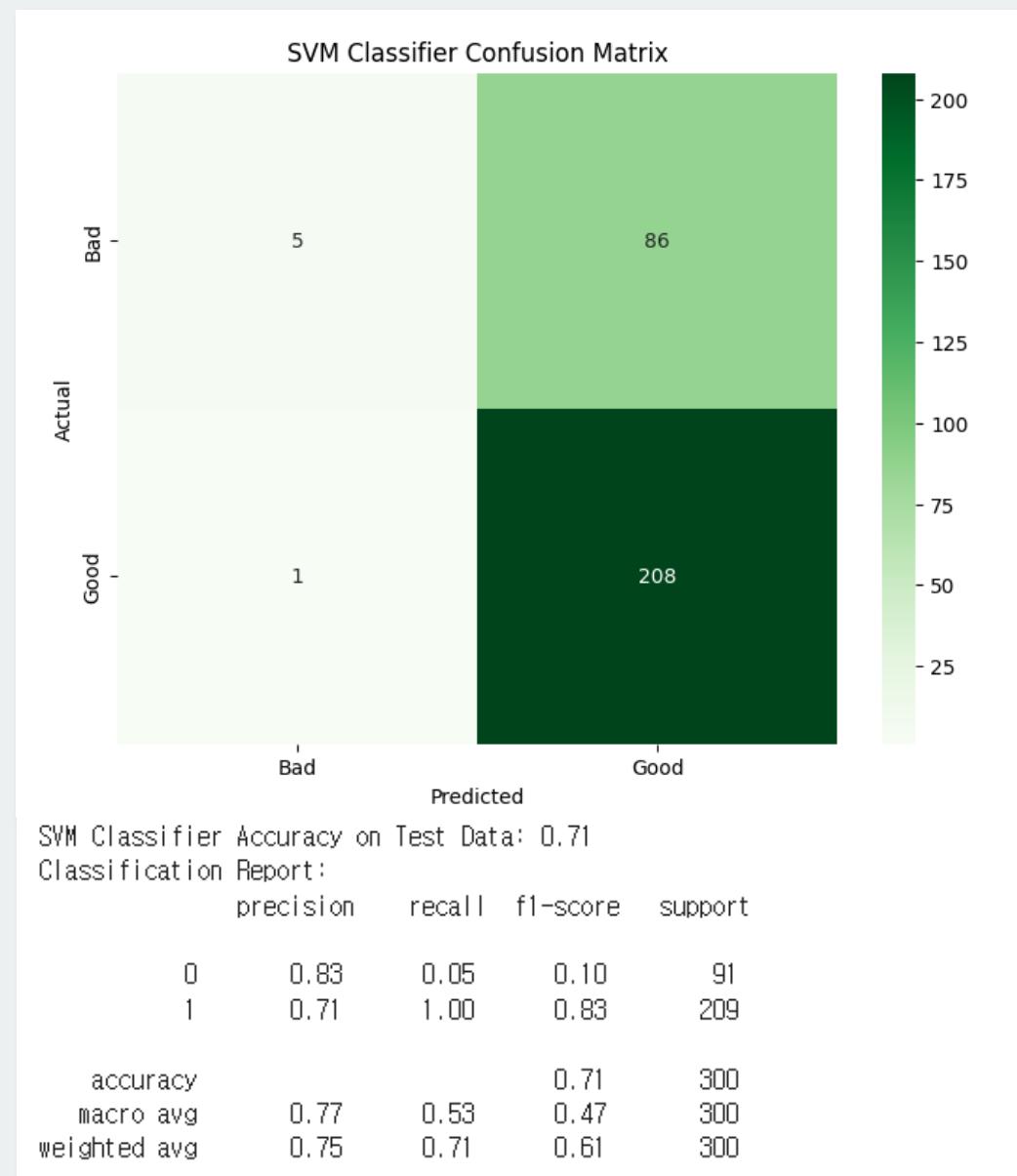
- Recall(재현율) : 1

실제 'Good'인 것 중에서 'Good'으로 예측한 비율 → 100%가 'Good'으로 예측

- F1-score : 0.83 → 모델이 'Good'를 예측할 때의 전반적인 성능

- Support : 209 → 실제 'Good'으로 분류된 샘플수

\* 실제로 좋은 209개의 샘플 중 1개는 나쁨으로 잘못 예측되었고, 208개는 정확히 좋음으로 예측



## &gt;&gt;&gt; Case5. 로지스틱 회귀

## 전체 지표

- Accuracy : 0.71

전체 샘플 중에서 올바르게 예측한 비율 → 71%의 정확도

## 종합 해석

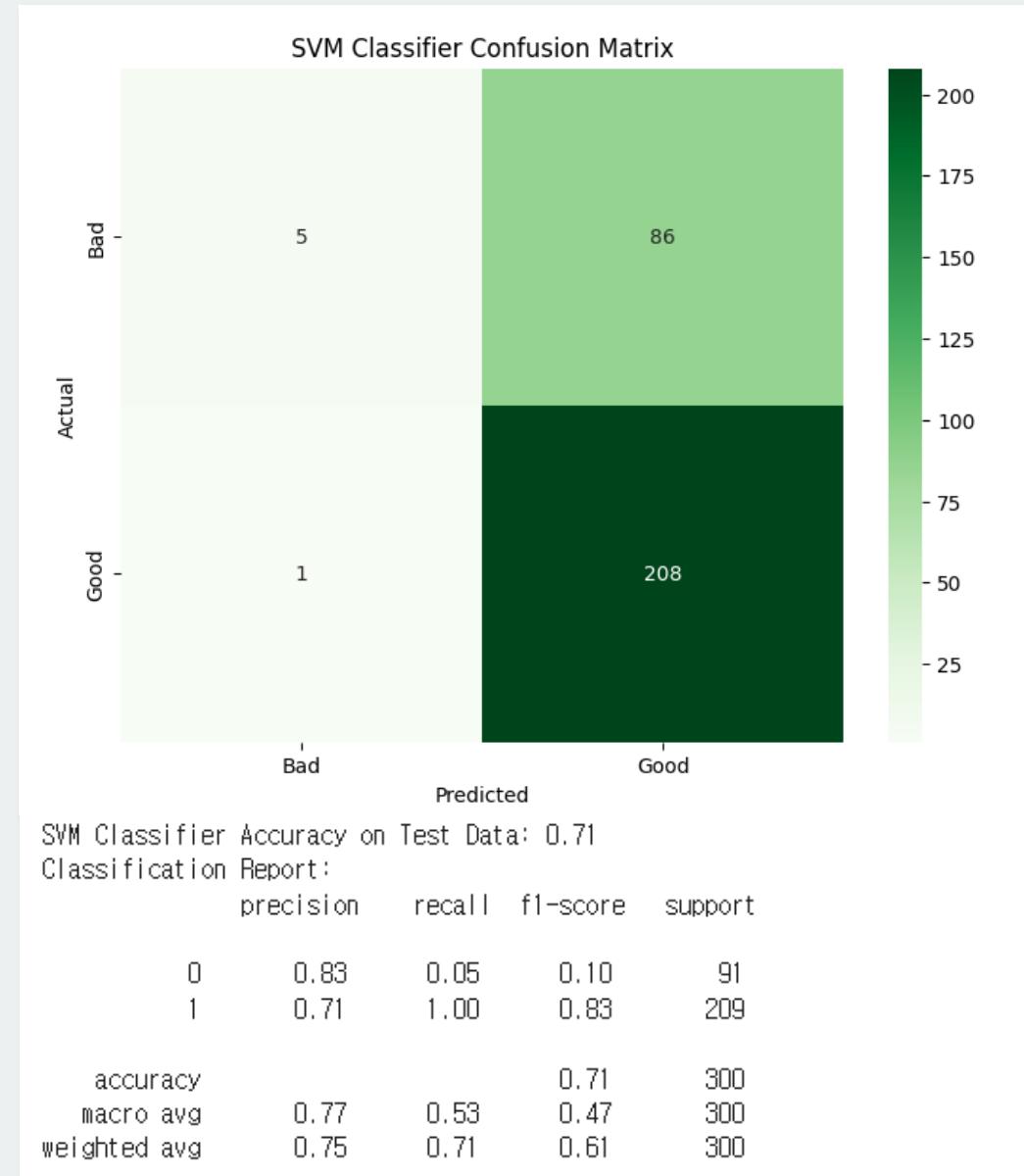
'Good'에 대해서는 잘 분류하는 반면,

'Bad'에 대해서는 정밀도는 높지만 **재현율이 낮다.**

**전체 정확도는 71%** 이지만,

데이터의 클래스 불균형(좋음이 많고 나쁨이 적음) 때문에

**정확도만 보고 평가하기에는 어렵다.**

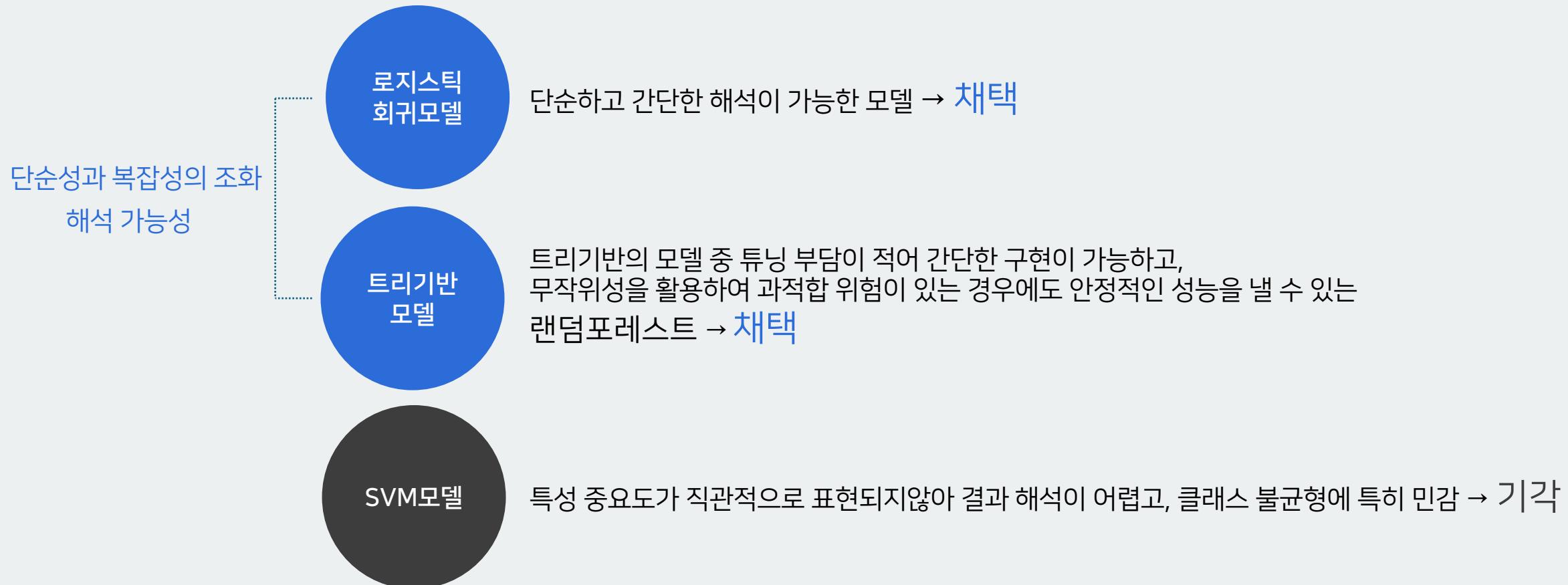


1. 본 프로젝트의 목표는 Risk가 0인 데이터들(채무 불이행 가능성 ↑)을 선별해 은행의 대출 리스크를 줄이는 것  
이기 때문에, 앞서 분석한 머신러닝 기법들 중 Risk = 0인 데이터의 f1-score가 높은 기법을 선택 → XGBoost  
의 0.4
2. XGBoost와 조에서 만든 모델을 비교했을 시, 모델 전체 정확도는 0.66으로 동일했으나, Risk = 0인 데이터의  
f1-score은 (XGBoost: 0.4)와 (본 모델: 0.43)으로 조에서 생성한 모델이 본 프로젝트의 목표 하에서는 더  
**좋은 모델임을 확인할 수 있었음.**
3. 정확도와는 별개로 모델들이 전반적으로 신용 위험도가 Good인 데이터들은 잘 분류하지만, Bad인 데이터들을  
분류하는 성능은 상대적으로 낮은 것으로 분석 (0.3~0.4)  
→ 데이터 자체의 **depth 문제(분석 칼럼 수의 제한)**, **데이터 불균형의 문제(good:bad = 7:3)** 등의 영향이 있  
으므로, 가이드라인 작성 시 다른 요소들을 추가적으로 고려해야 함

## 새로운 고객의 대출 심사를 가정한다면?

훈련 모델이 신용점수(Score)와 신용등급에 따라 대출 승인/거절을 어떻게 예측할까?

새로운 고객의 대출 심사를 예측하는 모델의 성능을 살펴보기 위해 두 가지 모델을 선정하여 테스트



```
# 새로운 고객 데이터를 입력받아 예측 # Credit 1

new_customer = {'Job': 1, 'Housing': 0, 'Saving accounts': 1, 'Checking account': 1, 'Credit amount': 2000, 'Duration': 48, 'Purpose': 1, 'Score': 40}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# X_train에 사용된 스케일러로 데이터를 변환
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.35
거절 확률: 0.65
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 2

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 6000, 'Duration': 48, 'Purpose': 1, 'Score': 67}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# X_train에 사용된 스케일러로 데이터를 변환
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.12
거절 확률: 0.88
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 3

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 1200, 'Duration': 6, 'Purpose': 1, 'Score': 72}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# X_train에 사용된 스케일러로 데이터를 변환
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.14
거절 확률: 0.86
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 4

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 4, 'Checking account': 2, 'Credit amount': 3000, 'Duration': 12, 'Purpose': 1, 'Score': 110}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# X_train에 사용된 스케일러로 데이터를 변환
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.04
거절 확률: 0.96
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 1

new_customer = {'Job': 1, 'Housing': 0, 'Saving accounts': 1, 'Checking account': 1, 'Credit amount': 2000, 'Duration': 48, 'Purpose': 1, 'Score': 40}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.35
거절 확률: 0.65
```

Credit\_rating 1인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 2

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 6000, 'Duration': 48, 'Purpose': 1, 'Score': 67}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.12
거절 확률: 0.88
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 3

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 1200, 'Duration': 6, 'Purpose': 1, 'Score': 72}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.14
거절 확률: 0.86
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 4

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 4, 'Checking account': 2, 'Credit amount': 3000, 'Duration': 12, 'Purpose': 1, 'Score': 110}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.04
거절 확률: 0.96
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 1

new_customer = {'Job': 1, 'Housing': 0, 'Saving accounts': 1, 'Checking account': 1, 'Credit amount': 2000, 'Duration': 48, 'Purpose': 1, 'Score': 40}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.35
거절 확률: 0.65
```

Credit\_rating 1인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 2

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 6000, 'Duration': 48, 'Purpose': 1, 'Score': 67}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.12
거절 확률: 0.88
```

Credit\_rating 2인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 3

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 1200, 'Duration': 6, 'Purpose': 1, 'Score': 72}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.14
거절 확률: 0.86
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 4

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 4, 'Checking account': 2, 'Credit amount': 3000, 'Duration': 12, 'Purpose': 1, 'Score': 110}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.04
거절 확률: 0.96
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 1

new_customer = {'Job': 1, 'Housing': 0, 'Saving accounts': 1, 'Checking account': 1, 'Credit amount': 2000, 'Duration': 48, 'Purpose': 1, 'Score': 40}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.35
거절 확률: 0.65
```

Credit\_rating 1인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 2

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 6000, 'Duration': 48, 'Purpose': 1, 'Score': 67}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.12
거절 확률: 0.88
```

Credit\_rating 2인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 3

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 1200, 'Duration': 6, 'Purpose': 1, 'Score': 72}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.14
거절 확률: 0.86
```

Credit\_rating 3인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 4

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 4, 'Checking account': 2, 'Credit amount': 3000, 'Duration': 12, 'Purpose': 1, 'Score': 110}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.04
거절 확률: 0.96
```

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 1

new_customer = {'Job': 1, 'Housing': 0, 'Saving accounts': 1, 'Checking account': 1, 'Credit amount': 2000, 'Duration': 48, 'Purpose': 1, 'Score': 40}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.35
거절 확률: 0.65
```

Credit\_rating 1인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 2

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 6000, 'Duration': 48, 'Purpose': 1, 'Score': 67}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.12
거절 확률: 0.88
```

Credit\_rating 2인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 3

new_customer = {'Job': 2, 'Housing': 5, 'Saving accounts': 1, 'Checking account': 2, 'Credit amount': 1200, 'Duration': 6, 'Purpose': 1, 'Score': 72}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.14
거절 확률: 0.86
```

Credit\_rating 3인 경우,

대출 승인 불가

```
# 새로운 고객 데이터를 입력받아 예측 # Credit 4

new_customer = {'Job': 1, 'Housing': 5, 'Saving accounts': 4, 'Checking account': 2, 'Credit amount': 3000, 'Duration': 12, 'Purpose': 1, 'Score': 110}

# 고객 데이터를 DataFrame으로 변환
import pandas as pd
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X) # 기존 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df) # 새로운 고객 데이터 스케일링

# 새로운 고객 데이터에 대한 예측
prediction = model.predict(new_customer_scaled)
prediction_proba = model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.04
거절 확률: 0.96
```

Credit\_rating 4인 경우,

대출 승인 불가

```
# Credit 1
# 데이터 분리
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 0, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 1, # 체크 계좌 상태
    'Credit amount': 1845, # 대출 금액
    'Duration': 45, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 41 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: " + str(prediction_proba[0][1]))
print("거절 확률: " + str(prediction_proba[0][0]))
```

**대출 거절:** 고객은 대출 승인이 불가능합니다.  
**승인 확률:** 0.26  
**거절 확률:** 0.74

```
# Credit 2
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 6000, # 대출 금액
    'Duration': 48, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 67 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: " + str(prediction_proba[0][1]))
print("거절 확률: " + str(prediction_proba[0][0]))
```

**대출 거절:** 고객은 대출 승인이 불가능합니다.  
**승인 확률:** 0.25  
**거절 확률:** 0.75

```
# Credit 3 # Credit_Rain 3부터는 랜덤포레스트 모델이 대출이 가능하다고 심사
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 72 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: " + str(prediction_proba[0][1]))
print("거절 확률: " + str(prediction_proba[0][0]))
```

**대출 승인:** 고객은 대출 승인이 가능합니다.  
**승인 확률:** 0.98  
**거절 확률:** 0.02

```
# Credit_Rating 4
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 1, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 110 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: " + str(prediction_proba[0][1]))
print("거절 확률: " + str(prediction_proba[0][0]))
```

**대출 승인:** 고객은 대출 승인이 가능합니다.  
**승인 확률:** 0.93  
**거절 확률:** 0.07

```
# Credit 1

# 데이터 분리
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 0, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 1, # 체크 계좌 상태
    'Credit amount': 6000, # 대출 금액
    'Duration': 48, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 67 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: {prediction_proba[0][1]:.2f}")
print("거절 확률: {prediction_proba[0][0]:.2f}")

# 대출 거절: 고객은 대출 승인이 불가능합니다.
# 승인 확률: 0.26
# 거절 확률: 0.74
```

Credit rating 1인 경우,  
대출 승인 불가

```
# Credit 2

# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 1100, # 대출 금액
    'Duration': 6, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 72 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: {prediction_proba[0][1]:.2f}")
print("거절 확률: {prediction_proba[0][0]:.2f}")

# 대출 거절: 고객은 대출 승인이 불가능합니다.
# 승인 확률: 0.25
# 거절 확률: 0.75
```

```
# Credit 3 # Credit_Rain 3부터는 랜덤포레스트 모델이 대출이 가능하다고 심사

# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 110 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: {prediction_proba[0][1]:.2f}")
print("거절 확률: {prediction_proba[0][0]:.2f}")

# 대출 승인: 고객은 대출 승인이 가능합니다.
# 승인 확률: 0.98
# 거절 확률: 0.02
```

```
# Credit_Rating 4

# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 탄생 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 1, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 110 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame([new_customer])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: {prediction_proba[0][1]:.2f}")
print("거절 확률: {prediction_proba[0][0]:.2f}")

# 대출 승인: 고객은 대출 승인이 가능합니다.
# 승인 확률: 0.93
# 거절 확률: 0.07
```

```
# Credit 1
# 데이터 분리
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 타겟 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 0, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 1, # 체크 계좌 상태
    'Credit amount': 6000, # 대출 금액
    'Duration': 48, # 대출 기간
    'Purpose': 3, # 대출 목적
    'Score': 50 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: (prediction_proba[0][1]:.2f)")
print("거절 확률: (prediction_proba[0][0]:.2f)")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.26
거절 확률: 0.74
```

```
# Credit 2
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 타겟 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 1100, # 대출 금액
    'Duration': 6, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 50 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: (prediction_proba[0][1]:.2f)")
print("거절 확률: (prediction_proba[0][0]:.2f)")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.25
거절 확률: 0.75
```

```
# Credit 3 # Credit_Rating 3부분은 랜덤포레스트 모델이 대출이 가능하다고 카운트
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 타겟 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 110 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: (prediction_proba[0][1]:.2f)")
print("거절 확률: (prediction_proba[0][0]:.2f)")

대출 승인: 고객은 대출 승인이 가능합니다.
승인 확률: 0.98
거절 확률: 0.02
```

```
# Credit_Rating 4
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 타겟 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 1, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 1, # 대출 목적
    'Score': 110 # 신용 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print("승인 확률: (prediction_proba[0][1]:.2f)")
print("거절 확률: (prediction_proba[0][0]:.2f)")

대출 승인: 고객은 대출 승인이 가능합니다.
승인 확률: 0.93
거절 확률: 0.07
```

```
# Credit 1
# 데이터 분리
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 단것 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 0, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 1, # 체크 계좌 상태
    'Credit amount': 6000, # 대출 금액
    'Duration': 48, # 대출 기간
    'Purpose': 3, # 대출 목적
    'Score': 3 # 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.26
거절 확률: 0.74
```

```
# Credit 2
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 단것 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 6000, # 대출 금액
    'Duration': 6, # 대출 기간
    'Purpose': 3, # 대출 목적
    'Score': 3 # 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 거절: 고객은 대출 승인이 불가능합니다.
승인 확률: 0.25
거절 확률: 0.75
```

Credit\_rating 1인 경우,  
대출 승인 불가

```
# Credit 3 # Credit_Rating 3부터는 랜덤포레스트 모델이 대출이 가능하다고 함
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 단것 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 2, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 1, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 1100, # 대출 금액
    'Duration': 6, # 대출 기간
    'Purpose': 3, # 대출 목적
    'Score': 3 # 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 승인: 고객은 대출 승인이 가능합니다.
승인 확률: 0.98
거절 확률: 0.02
```

Credit\_rating 3인 경우,  
대출 승인 가능

```
# Credit_Rating 4
# 데이터 분리 (예: 기존 데이터)
X = df[['Job', 'Housing', 'Saving accounts', 'Checking account', 'Credit amount', 'Duration', 'Purpose', 'Score']]
y = df['Risk'] # 단것 변수 (0: 거절, 1: 승인)

# 학습 데이터와 테스트 데이터로 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 데이터 스케일링
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 랜덤포레스트 모델 생성 및 학습
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train)

# 새로운 고객 데이터 입력
new_customer = {
    'Job': 1, # 직업 유형
    'Housing': 5, # 주택 상태
    'Saving accounts': 4, # 저축 계좌 상태
    'Checking account': 2, # 체크 계좌 상태
    'Credit amount': 3000, # 대출 금액
    'Duration': 12, # 대출 기간
    'Purpose': 3, # 대출 목적
    'Score': 3 # 점수
}

# 새로운 고객 데이터를 DataFrame으로 변환
new_customer_df = pd.DataFrame(new_customer, index=[0])

# 데이터 스케일링
new_customer_scaled = scaler.transform(new_customer_df)

# 예측 수행
prediction = rf_model.predict(new_customer_scaled)
prediction_proba = rf_model.predict_proba(new_customer_scaled)

# 결과 출력
if prediction[0] == 1:
    print("대출 승인: 고객은 대출 승인이 가능합니다.")
else:
    print("대출 거절: 고객은 대출 승인이 불가능합니다.")

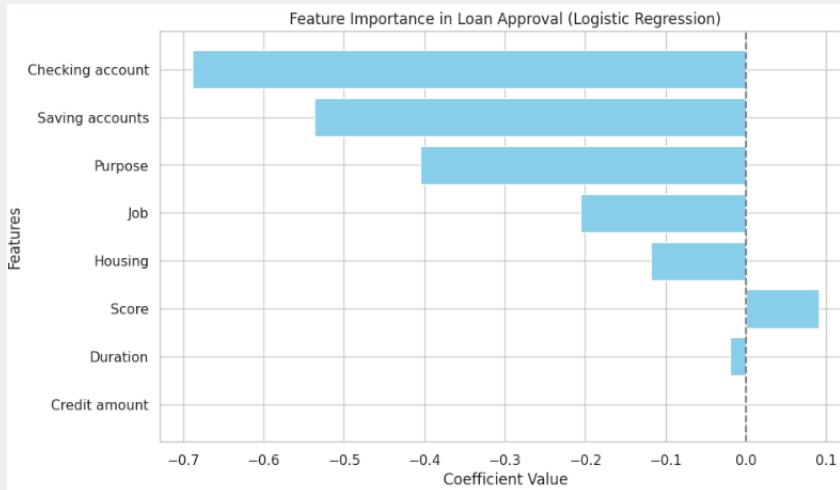
# 예측 확률 출력
print(f"승인 확률: {prediction_proba[0][1]:.2f}")
print(f"거절 확률: {prediction_proba[0][0]:.2f}")

대출 승인: 고객은 대출 승인이 가능합니다.
승인 확률: 0.93
거절 확률: 0.07
```

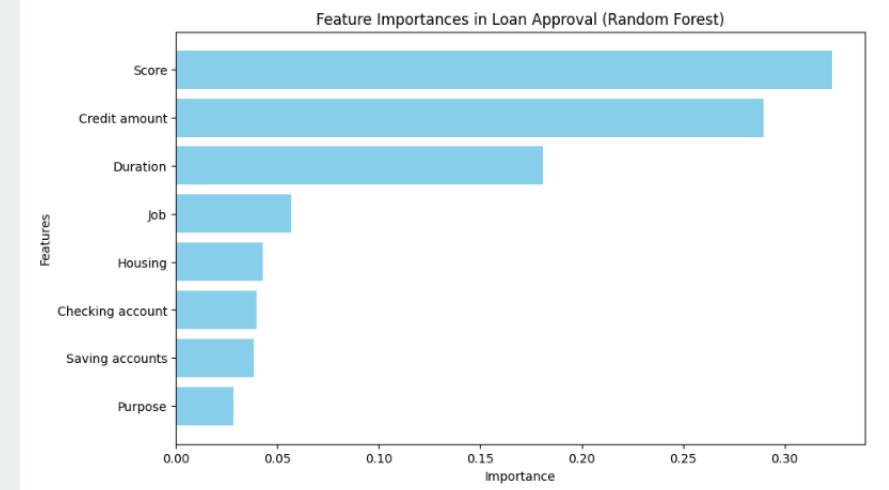
Credit\_rating 4인 경우,  
대출 승인 가능

## 왜 로지스틱 회귀모델은 예측을 잘하지 못했고, 랜덤포레스트 모델은 예측을 잘했을까?

### 로지스틱 회귀모델의 특성 중요도



### 랜덤포레스트 모델의 특성 중요도



### 특성 중요도 분석

가장 중요한 특성: Checking account, Saving accounts, Purpose  
 음수 계수가 대부분으로, 대출 거절 가능성을 높이는 요인으로 작용  
 Score(신용 점수)와 같은 핵심 변수의 기여도가 상대적으로 작아,  
 주요 변수의 중요도를 제대로 반영하지 못함

가장 중요한 특성: Score, Credit amount, Duration  
 데이터에 중요한 특성이 잘 반영되어 대출 승인 여부에 중요한 역할  
 Checking account, Saving accounts 등은 상대적으로 중요도가 낮게 평가됨

### 성능차이의 이유

로지스틱 회귀는 선형 모델로, 데이터가 직선관계에 있을 경우 적합하며,  
 데이터가 비선형적인 관계를 가지고 있을 경우, 복잡한 관계를 학습하지 못함

여러 개의 의사결정 트리를 결합하여 작동하기에 데이터에 복잡한 패턴이 있을 때 예측 성능이 좋음  
 비선형 데이터와 특성 간 상호작용을 잘 반영하여, 대출 심사 데이터를 더 정확히 예측

## 1. 예측 모델 기반 가이드라인 설정

모델이 예측한 결과에 따라 대출 승인 불가인 경우,

→ 대출거절

→ 추가심사 (조건부승인) :

- 추가 제출 서류 : 소득 증빙 자료, 부동산 등 담보물에 대한 서류
- 담보 요구 : 담보 가치가 대출 금액의 125% 이상이어야 함 (LTV 80%)
- 보증인 요구: 보증인의 신용 점수가 4등급 이상이어야 함

## 2. 신용등급 기반으로 대출 금리 차등 설정

1등급 ~ 2등급 : 높은 기본금리와 가산 금리를 적용하여, 신용점수에 따른 예상 손실률을 줄이는 리스크 관리

3등급 ~ 4등급 : 우대 금리를 제공하여 우수한 신용의 고객 확보를 확보하고, 고객 충성도를 높이는 효과

## 3. 신용점수(Score)의 사분위수에 따라 대출 한도 설정

신용점수의 사분위수에 따라 대출 한도를 다르게 설정하는 방안

신용 점수에 따른 한도 차등을 두어 과도한 대출을 방지하고, 고객의 상환 능력에 따라 적절한 대출 한도 설정

## 주요 인사이트

1. 단순히 정확도라는 단일 지표에 의존하지 않고, 정밀도(Precision), 재현율(Recall), F1 Score 등 다양한 성능 지표를 활용하여 모델의 성능을 다각도로 평가하는 것이 중요함을 확인함. 이러한 접근은 프로젝트의 목표와 데이터를 고려한 최적의 모델을 선택하는 데 도움을 줄 수 있음을 시사함.
2. 신용도가 낮은 고객 탐지에 어려움을 겪은 주요 원인 중 하나로 데이터 편향과 결측치 문제를 특정함. 특히 주요 특성에서의 결측치와 불균형 데이터로 인해 모델 학습이 제한되었으며, 이는 신용등급 분류의 성능 저하로 이어지는 모습을 확인 가능했음.

## 한계점

1. 제한된 데이터셋 크기와 주요 변수의 결측치는 모델의 예측력을 저하시킴. 이러한 데이터 제약은 신뢰할 수 있는 모델 개발에 걸림돌로 작용했음.
2. 비교 분석한 모델들의 성능이 실제 은행 환경에서 적용하기에는 미흡한 수준이었음. 특히, 신용도가 낮은 고객을 효과적으로 식별하지 못해, 추가 데이터 확보와 고도화된 전처리 과정이 필수적임을 확인했음.

# QnA

Thank you