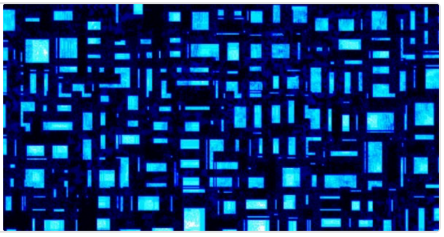
Bank Marketing - Term Deposite prediction

활용데이터 정보

Bank Marketing

source: https://archive.ics.uci.edu/ml/datasets/bank+marketing

k https://www.kaggle.com/datasets/henriqueyamahata/bank-marketing/data



1. 서론

1-1 배경

이 데이터셋은 포르투갈 은행의 정기예금 마케팅 캠페인과 관련된 정보를 포함하며, 주로 전화 통화를 통해 수집된 자료이다. 데이터에는 고객의 개인 정보, 경제적 지표, 캠페인 관련 정보가 포함되어 있다. 이를 통해 고객의 특성과 경제적 요인이 정기예금 가입 의사에 미치는 영향을 탐구하고, 효과적인 마케팅 전략을 제안할 수 있는 기초 자료로 활용한다.

1-2 분석목표

이 데이터는 포르투갈 은행 기관의 정기예금 마케팅 캠페인 관련된 데이터이다. 분류 목표는 고객이 정기예금에 가입할지 여부를 예측하는 것이다.

고객의 개인 정보(예: 나이, 직업, 결혼 상태, 대출 여부)뿐만 아니라 경제적 지표(고용 변화율, 소비자 물가 지수 등), 캠페인 관련 정보(연락 횟수, 이전 캠페인 결과 등)를 포함하고 있다.

이를 통해 고객의 특성 및 경제적 요인들이 정기예금 가입 의사에 미치는 영향을 탐구하며, 이를 바탕으로 보다 효과적인 마케팅 전략 제공하는 것을 목표로 한다. 데이터 분석을 통해 주요 변수들과 캠페인 성공률 간의 상관 관계를 밝히고, 이를 토대로 실질적인 비즈니스 개선 방안을 제안한다. 고객을 그룹화하여 개인적인 경제적 상황, 시대의 경제 상황 등 다양한 요소를 반영해 맞춤형 전략을 제시한다.

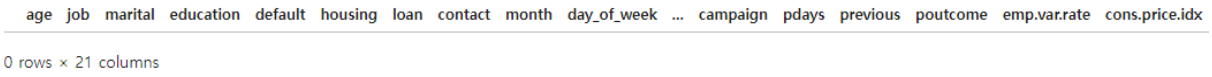
1-3 데이터 컬럼 설명

Attribute	Description
Age	나이 (숫자)
Job	직업 유형(범주형: '관리자', '사무직', '기업가', '가정부', '관리직', '은퇴', '자영업', '서비스직', '학생', '기술자', '무직', '불명')
Marital	결혼 여부(범주형: '이혼', '기혼', '미혼', '불명', '이혼'은 이혼 또는 사별을 의미함)
Education	교육 수준(범주형: '초졸', '중졸', '고졸', '문맹', '전문대졸', '대학졸업', '미상')
Default	신용불량 상태 여부(범주형: '아니요', '예', '알 수 없음')
Housing	주택 대출 여부(범주형: '아니요', '예', '알 수 없음')
Loan	개인 대출 여부(범주형: '아니요', '예', '알 수 없음')
Contact	연락처 커뮤니케이션 유형(범주형: '셀룰러', '전화')
Month	연도의 마지막 연락처 월(범주형: '1월', '2월', '3월', ..., '12월')
Day_of_week	마지막 연락 요일(범주형: '월', '화', '수', '목', '금')
Duration	마지막 연락 지속 시간(초)(숫자)
Campaign	이 캠페인 기간동안 연락된 수 (숫자, 마지막 연락처 포함)
Pdays	이전 캠페인에서 마지막으로 연락 후 경과한 일수(숫자, 999는 이전 연락 없음)
Previous	이전 캠페인에서 연락된 수 (숫자)
Poutcome	이전 마케팅 캠페인의 결과(범주형: '실패', '없음', '성공')
Emp.var.rate	고용 변동률 - 분기별 지표(숫자)
Cons.price.idx	소비자 물가 지수 - 월별 지표(숫자)
Cons.conf.idx	소비자 신뢰 지수 - 월별 지표(숫자)
Euribor3m	3개월 기간에 대한 금리 - 일간 지표(숫자)
Nr.employed	분기별 고용 수치 변화(숫자)
Y	정기예금 가입 여부(이진: '예', '아니오')

2. 필요한 데이터 정렬 및 정제 [EDA]

2 - 1. 결측치 확인 , 중복데이터 제거

```
# 각 컬럼별 결측치 개수 확인
missing_values_count = data.isnull().sum()

# 결측치가 있는 컬럼만 필터링
missing_columns = missing_values_count[missing_values_count > 0]

# 결측치가 있는 컬럼들과 그 개수 출력
print(missing_columns)

# 결측치가 있는 행만 필터링해서 출력
data_with_missing = data[data.isnull().any(axis=1)]

# 결측치가 있는 데이터 출력
data_with_missing
```



age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx

0 rows × 21 columns

이 데이터 셋에는 결측치가 있는 데이터는 존재하지 않았다.

```
# 중복 데이터가 있는지 확인하고, 중복 데이터를 제거하는 작업을 수행
# 중복 여부 확인
duplicate_rows = data[data.duplicated()]

# 중복 데이터 개수 출력
duplicate_count = duplicate_rows.shape[0]
duplicate_count

# 결과 : 12개

# 중복 데이터 제거
data = data.drop_duplicates()
```

데이터의 중복을 알아보기 위해 data.duplicated를 수행하여 12개의 중복데이터가 존재하는 것을 확인하고 제거했다.

2 - 2. ydata_profiling 를 활용한 EDA Overview

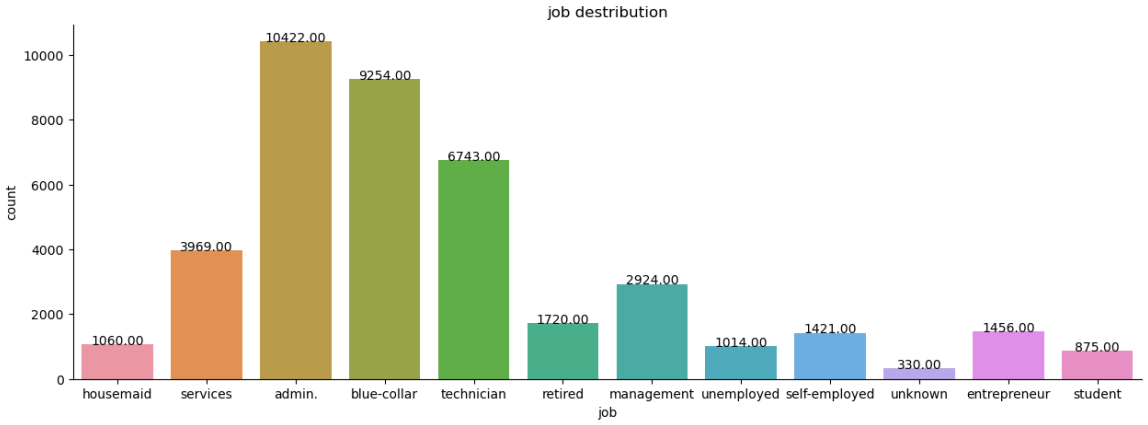
```
import pandas as pd
from ydata_profiling import ProfileReport

# CSV 파일을 불러와 데이터프레임으로 변환
data = pd.read_csv('bank-additional-full.csv', sep=';')

# 프로파일링 리포트 생성
profile = ProfileReport(data)

# 리포트를 HTML 파일로 저장
profile.to_file("report.html")

# 깃허브에 호스팅 하여 임베드
```

https://sunwoo209.github.io/Bank-Marketing/report.html

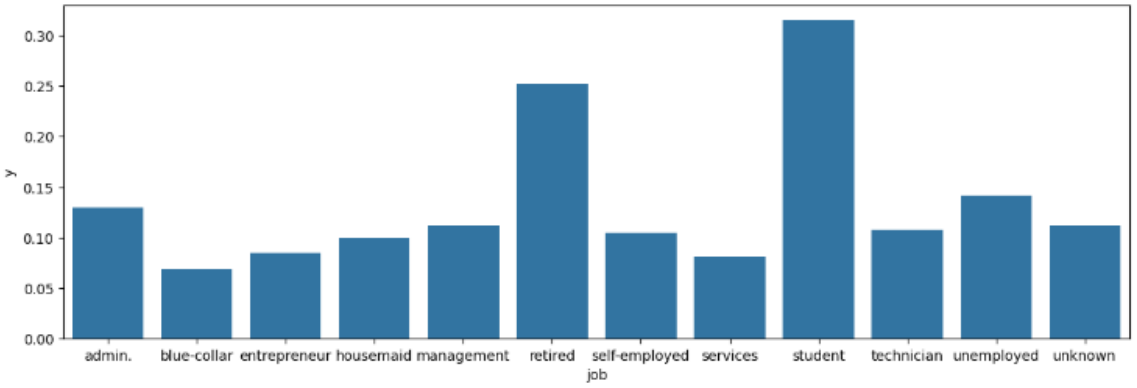
2 - 3. Column 분석 및 추론

- **직업:**

```
fig, ax = plt.subplots()
fig.set_size_inches(15, 5)
ax.set_title('job destribution')
sns.countplot(client, x = 'job')
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2),
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



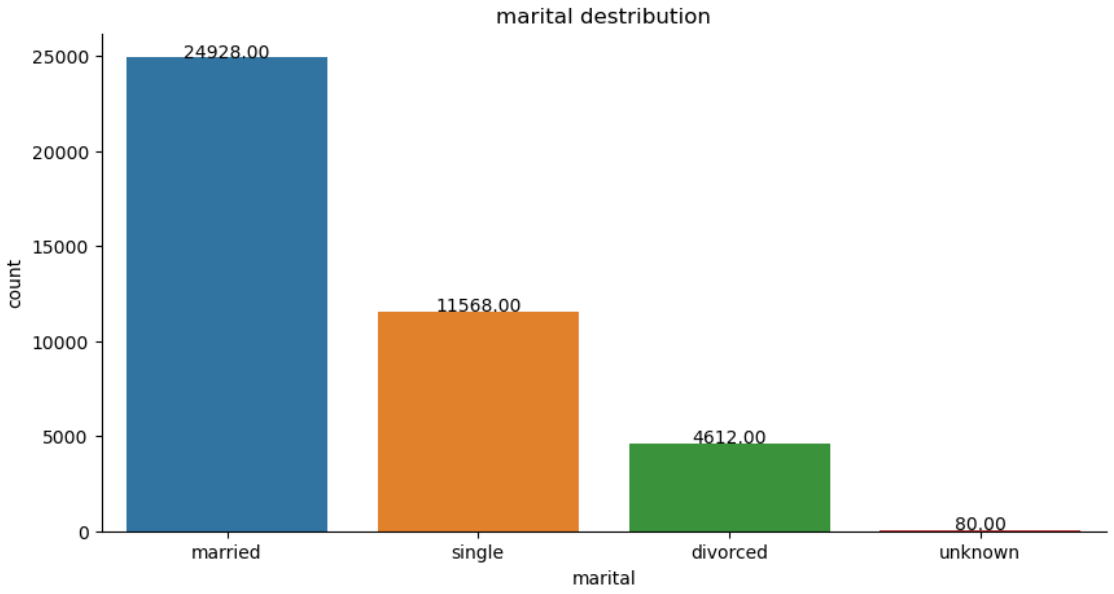
가장 많이 관측되는 직업군은 "admin"(관리자)가 10,422명, "blue-collar"(육체 노동자)가 9,254명, "technician"(기술자)가 6,743명인 것을 plot을 통해 확인해봤다.



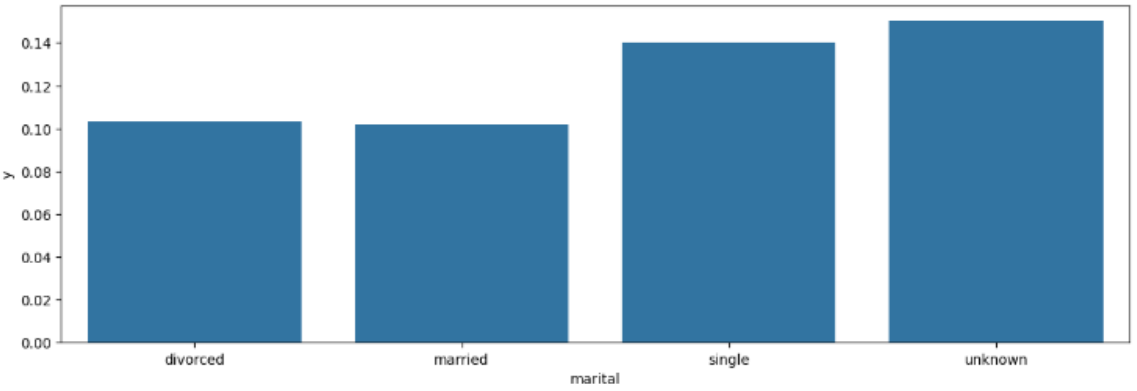
student(학생)과 retired(은퇴자)의 경우에 정기 예금을 가입할 확률이 가장 높은 것으로 보였다. 하지만 다른 데이터에 비해 student와 retired의 수가 적으므로 정기 예금에 가입할 확률이 다른 직업보다 높다고 단정하기는 어렵다. 하지만 육체 노동자들의 정기예금 가입률은 낮은 것으로 보여졌다.

- **결혼 상태:**

```
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
ax.set_title('marital destribution')
sns.countplot(client, x = 'marital')
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



대부분은 "married"(결혼)이 24,928명이였으며, "single"(미혼)이 11,568명이고, “divorced”(이혼 및 사별)이 4612명이 있는 것을 확인했다. 결혼 상태를 알 수 없는 데이터도 80개 존재함을 알 수 있었다.

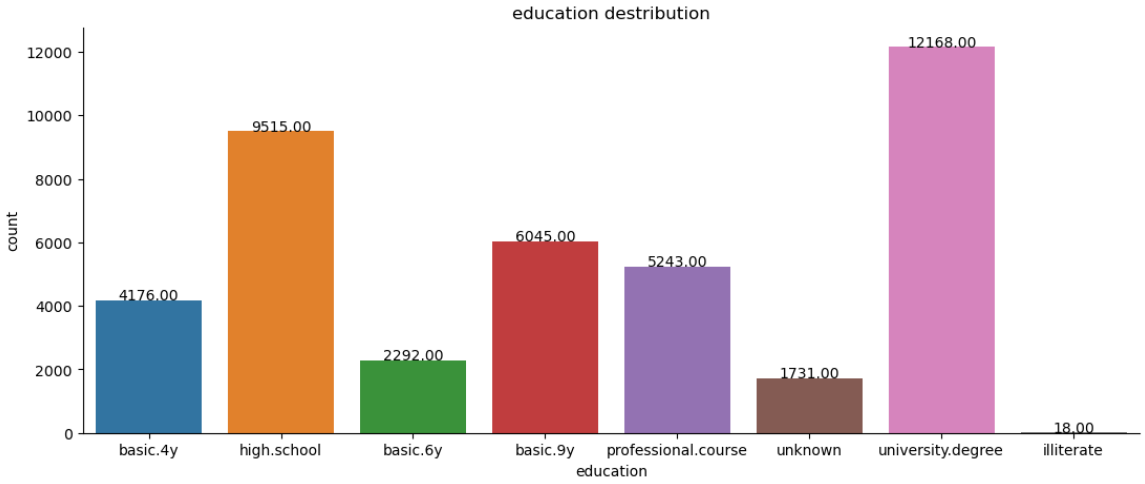


결혼 상태가 알려지지 않은 고객들의 정기예금 가입 확률이 가장 높다. 하지만 결혼 상태를 알 수 없는 사람의 데이터 수가 다른 상태에 비해 적기 때문에 결혼 상태를 알 수 없는 집단을 제외하고는 결혼을 하지 않은 사람들의 정기예금 가입 확률이 가장 높은 것을 확인할 수 있었다.

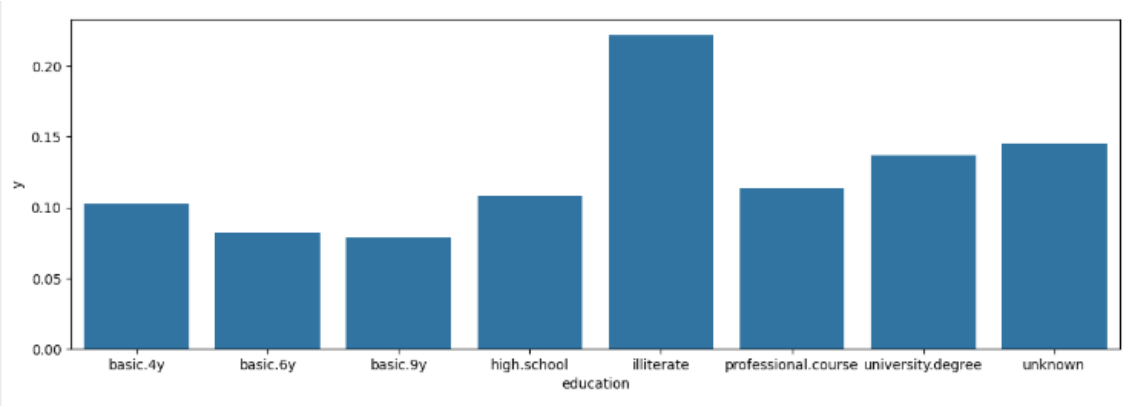
- **교육:**

```
fig, ax = plt.subplots()
fig.set_size_inches(13, 5)
ax.set_title('education destribution')
sns.countplot(client, x = 'education')
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
```

```
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



"university.degree"(대학 졸업)이 12,168명, "high.school"(고등 졸업)이 9,515명, “basic.9y”(중등 졸업)이 6,045명의 순서로 많은 것을 확인할 수 있었다.

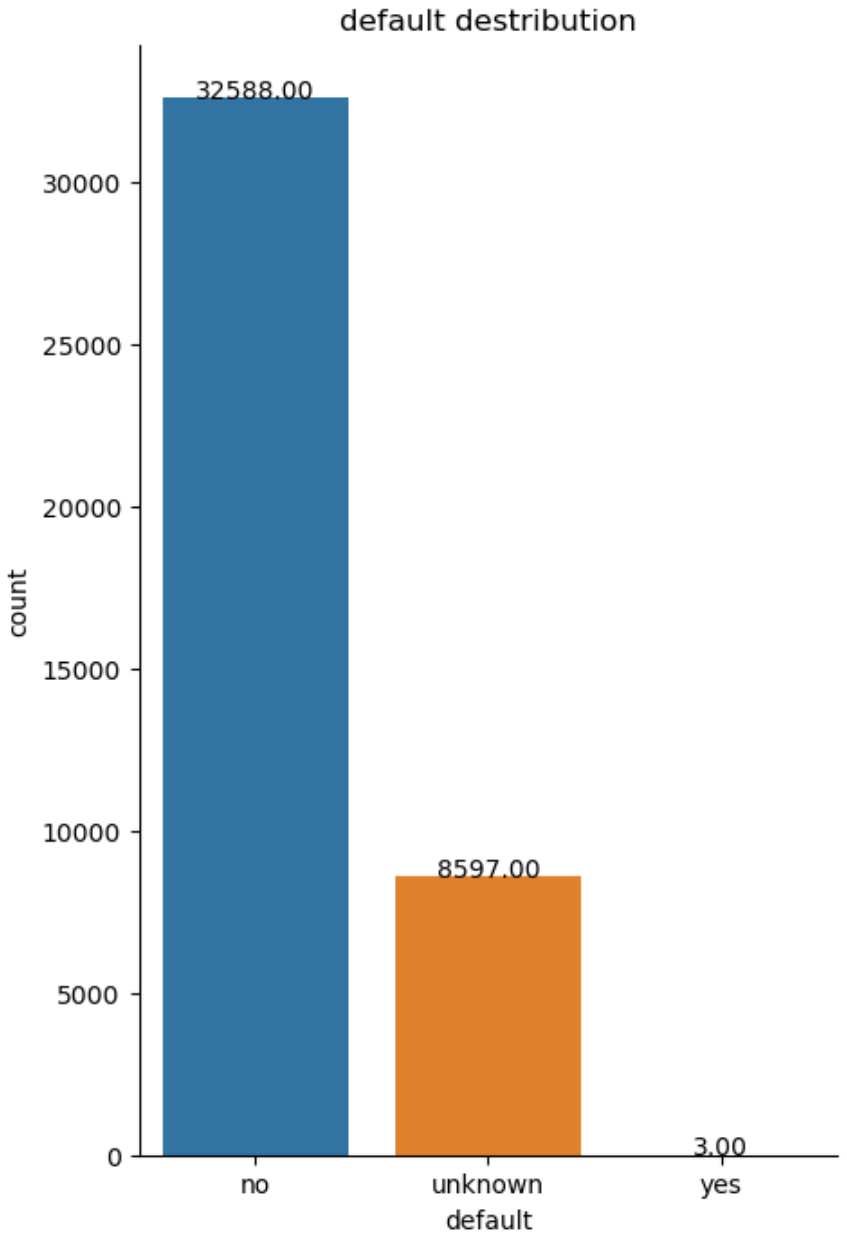


문맹(illiterate) 집단의 정기예금 가입 확률이 높지만 문맹 집단의 데이터 수가 적으므로 정기예금 가입확률이 크다고 단정할 수 없다. 문맹 집단과 학업 수준을 알 수 없는 집단을 제외하고는 대학을 졸업한 집단이 다른 교육 수준을 가진 집단에 비해 정기예금 가입 확률이 높은 것을 확인했다.

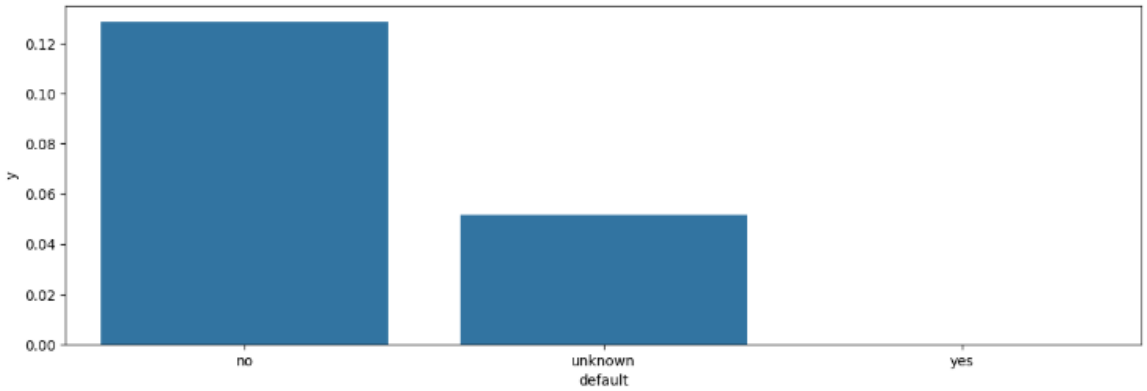
- **채무 불이행:**

```
fig, ax = plt.subplots()
fig.set_size_inches(5, 8)
ax.set_title('default destribution')
sns.countplot(client, x = 'default')
sns.despine()

for p in ax.patches:
```

```
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



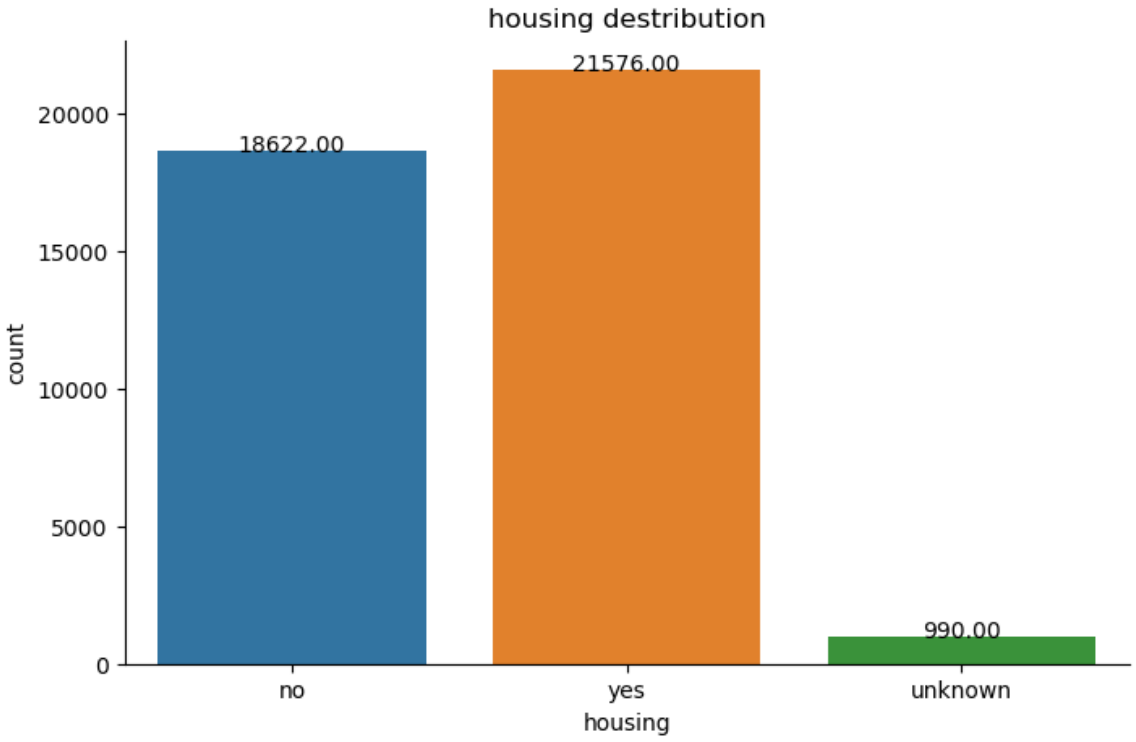
거의 모든 데이터가 "no"(채무 이행)으로 표시되고 "yes"(채무 불이행)는 매우 적다. 알려지지 않은 파산 상태가 8,597개로 많은 것을 볼 수 있다.



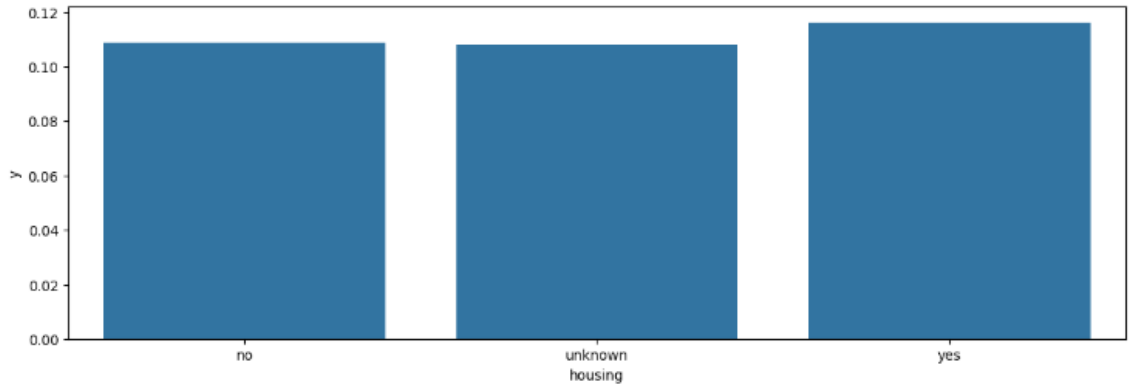
채무를 이행하는 집단이 정기예금을 가입할 확률이 가장 크며, 채무 불이행인 집단 중 정기 예금을 가입한 사람들은 단 한 명도 없는 것을 확인했다. 채무 이행상태를 알 수 없는 집단은 정기예금을 가입한 것을 확인할 수 있다. 그러므로 unknown 집단 또한 채무를 잘 이행하는 사람들로 추측할 수 있다.

- **주택 대출:**

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
ax.set_title('housing destribution')
sns.countplot(client, x = 'housing')
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



주택 대출이 있는 사람이 없는 사람보다 많이 있다는 것을 볼 수 있다. 주택 대출 여부를 알 수 없는 사람들 또한 990명 존재하는 것을 확인했다.

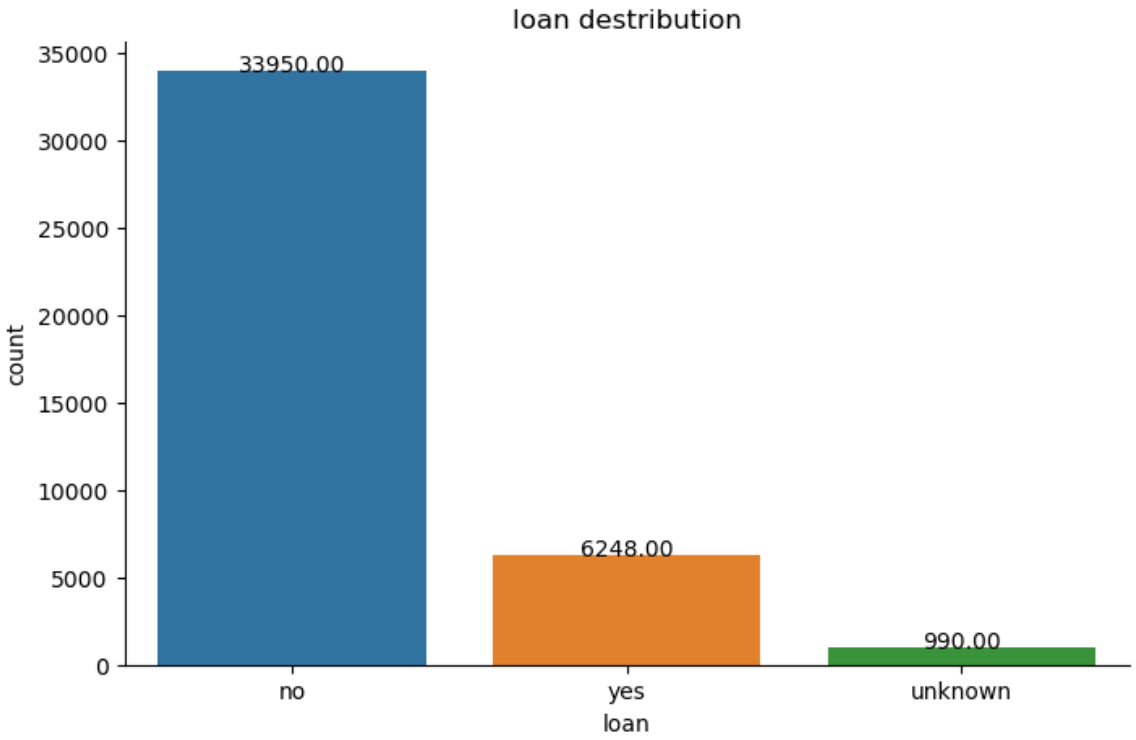


주택 대출의 유무에 대해서는 모두 비슷한 정기 예금 가입률을 보여주기 때문에 관련이 없어 보인다.

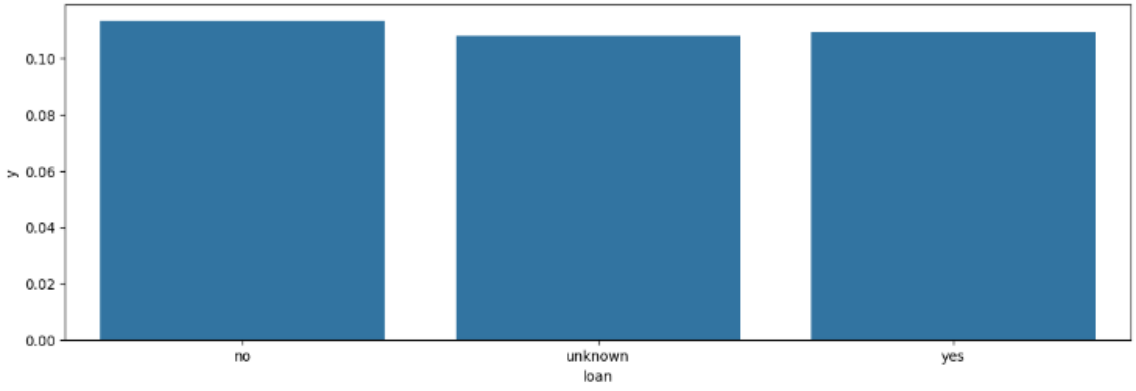
- **개인 대출:**

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
ax.set_title('loan destribution')
sns.countplot(client, x = 'loan')
sns.despine()
```

```
for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



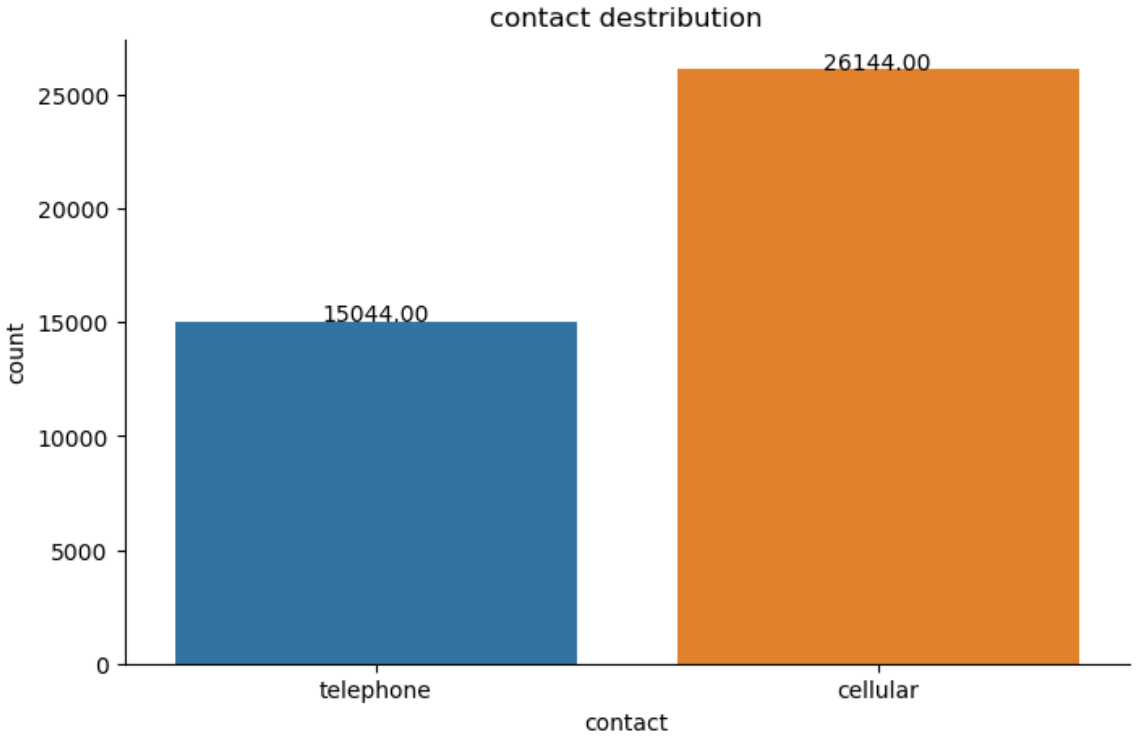
많은 사람이 개인 대출을 이용하지 않고 있다(33,950명)고 응답했으며 6,248명만 개인 대출을 이용하는 것을 확인했다. 개인 대출의 이용여부를 알 수 없는 사람의 수는 앞의 주택 대출과 같이 990명인 것을 확인했다.



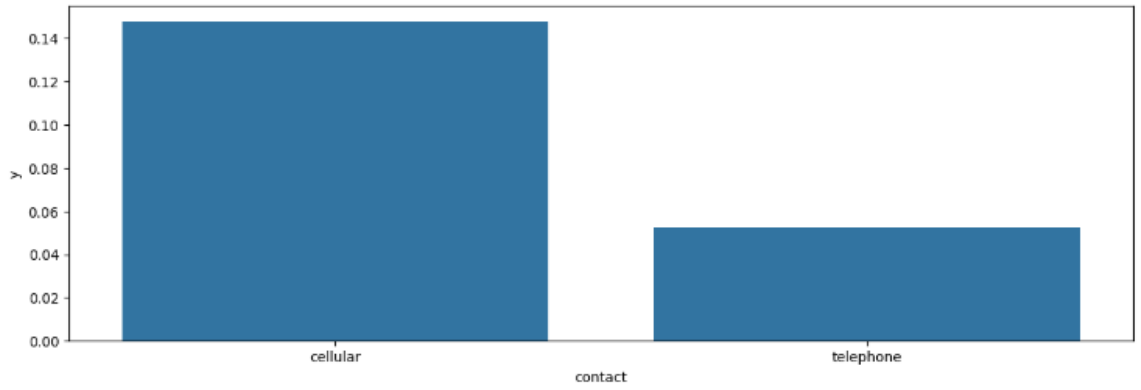
개인 대출 또한 주택 대출과 마찬가지로 정기예금 가입확률이 비슷하므로 관련이 있어 보이지 않았다.

- **연락 방법:**

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
ax.set_title('contact destribution')
sns.countplot(campaign, x = 'contact')
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



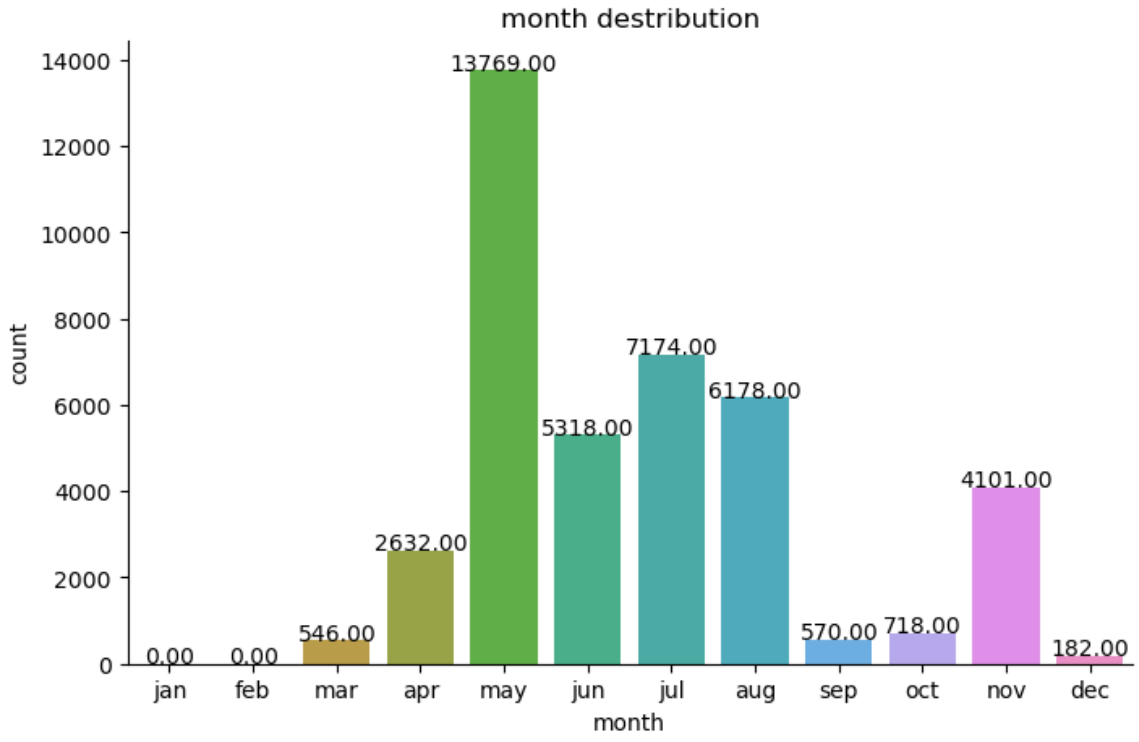
연락은 휴대폰으로 하는 빈도가 전화를 이용하는 것 보다 많은 것을 확인할 수 있었다.



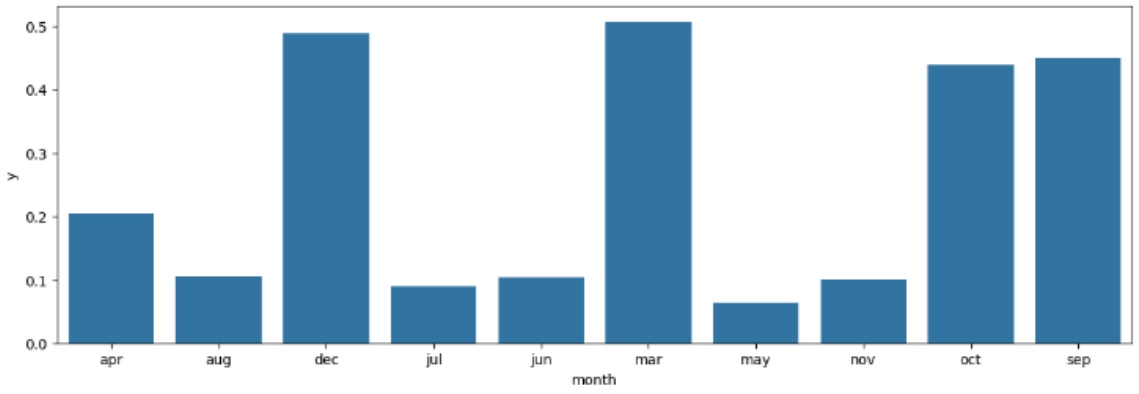
휴대전화로 연락이 이루어졌을 경우 전화를 이용해 연락을 한 경우보다 정기예금 가입 확률이 높은 것을 알 수 있었다.

- **월:**

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
ax.set_title('month destribution')
sns.countplot(campaign, x = 'month', order = ['jan', 'feb', 'mar'
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



5월에 가장 많은 연락을 했으며 7, 8, 6월이 그 뒤를 따른다. 대부분의 연락은 여름쯤 이뤄진 것을 확인할 수 있었다.

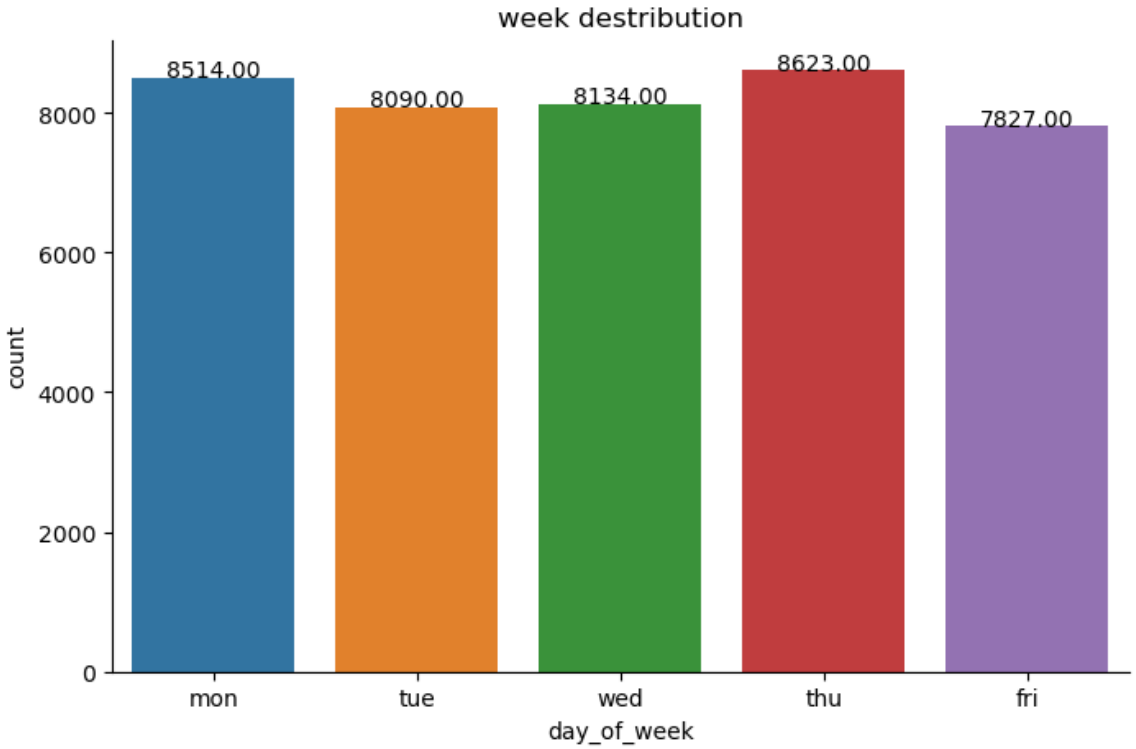


12월, 3월, 10월, 9월은 정기 예금 가입 확률이 다른 월에 비해 높지만, 데이터 수가 적기에 판단하기 어렵다. 반면에 5월에 가장 많은 연락을 했지만, 정기 예금 가입 확률은 가장 낮은 것을 볼 수 있다.

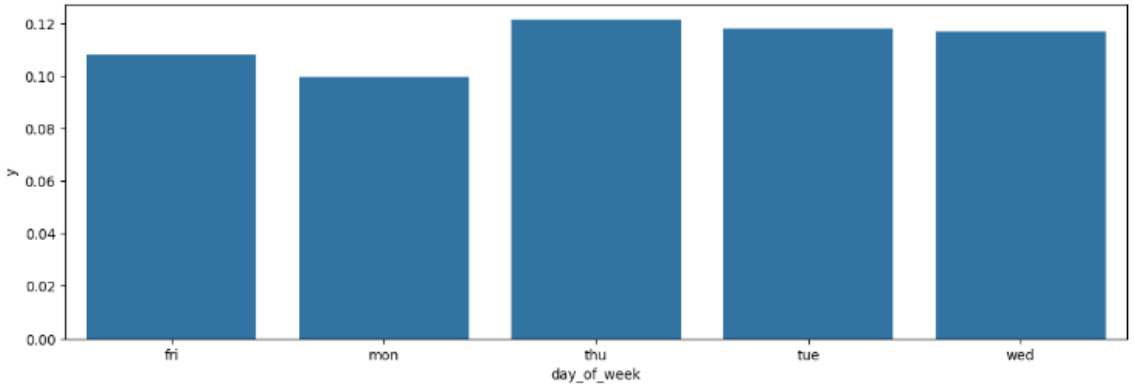
- **요일:**

```
fig, ax = plt.subplots()
fig.set_size_inches(8, 5)
ax.set_title('week destribution')
sns.countplot(campaign, x = 'day_of_week')
sns.despine()
```

```
for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```



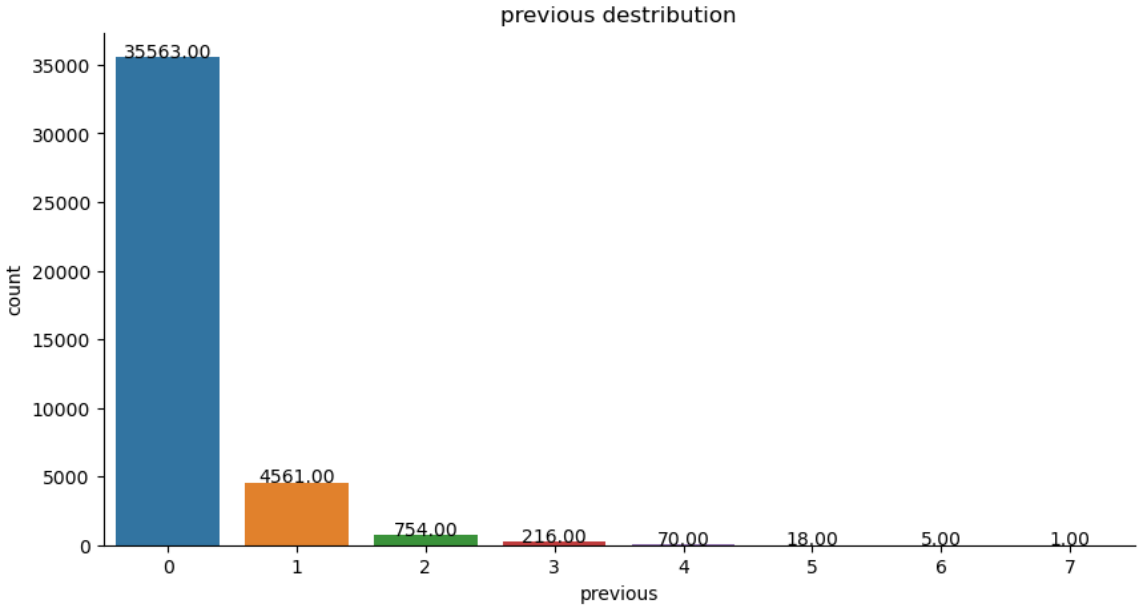
요일 별 연락 횟수는 크게 다르지 않은 것을 볼 수 있다.



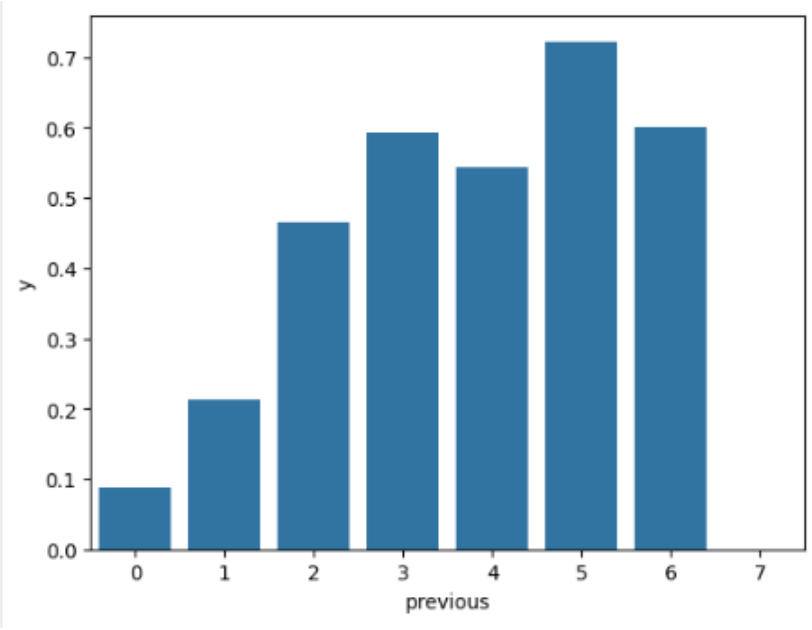
요일별로 정기예금 가입 확률이 비슷하게 분포된 것으로 보아 연락하는 요일은 정기예금 가입과 관련이 없는 것으로 보인다.

- **이전 캠페인에서 연락된 횟수 (previous):**

```
fig, ax = plt.subplots()
fig.set_size_inches(10, 5)
ax.set_title('previous destribution')
sns.countplot(other, x = 'previous')
sns.despine()

for p in ax.patches:
    ax.text(p.get_x() + (p.get_width()/2) ,
            p.get_y() + p.get_height(),
            f"{p.get_height():.2f}",
            ha = 'center' )
```

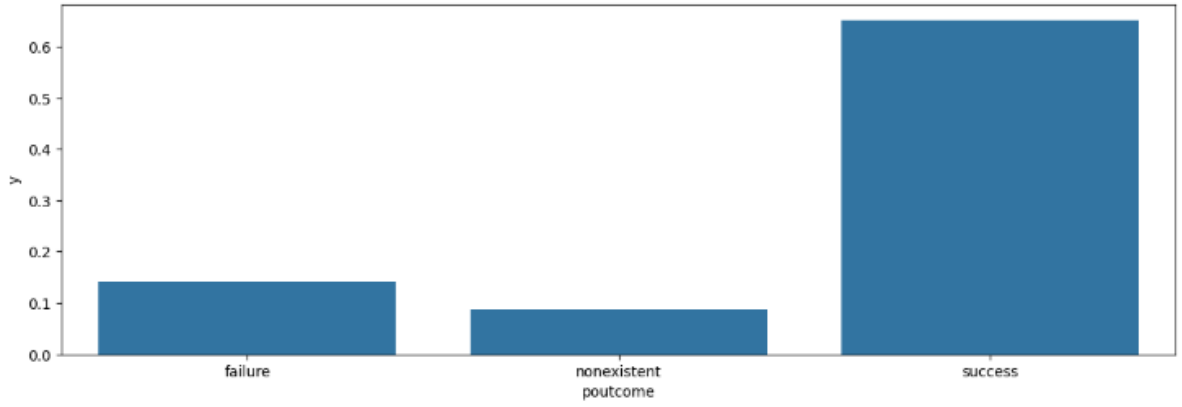


대부분의 데이터는 이전 마케팅의 캠페인에 응답하지 않은 것을 확인했다.



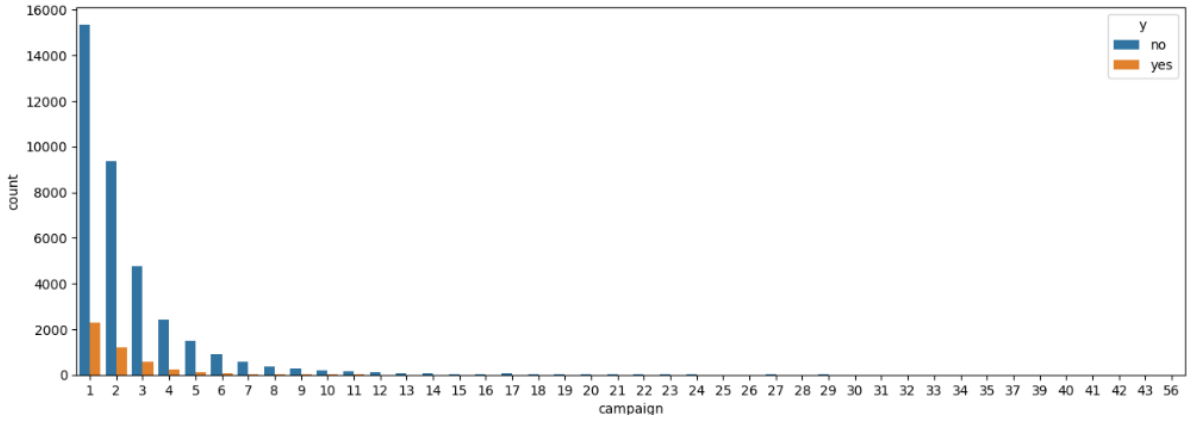
이전 캠페인에 응답하지 않은 사람들은 정기 예금 가입확률이 가장 낮았고, 연락 횟수가 많아질 수록 가입 확률이 높아지는 것을 확인했다. 하지만 이전 마케팅에서 연락된 사람들의 데이터 수가 적기 때문에 확정 짓기는 어렵다.

- **이전 결과 (poutcome)**



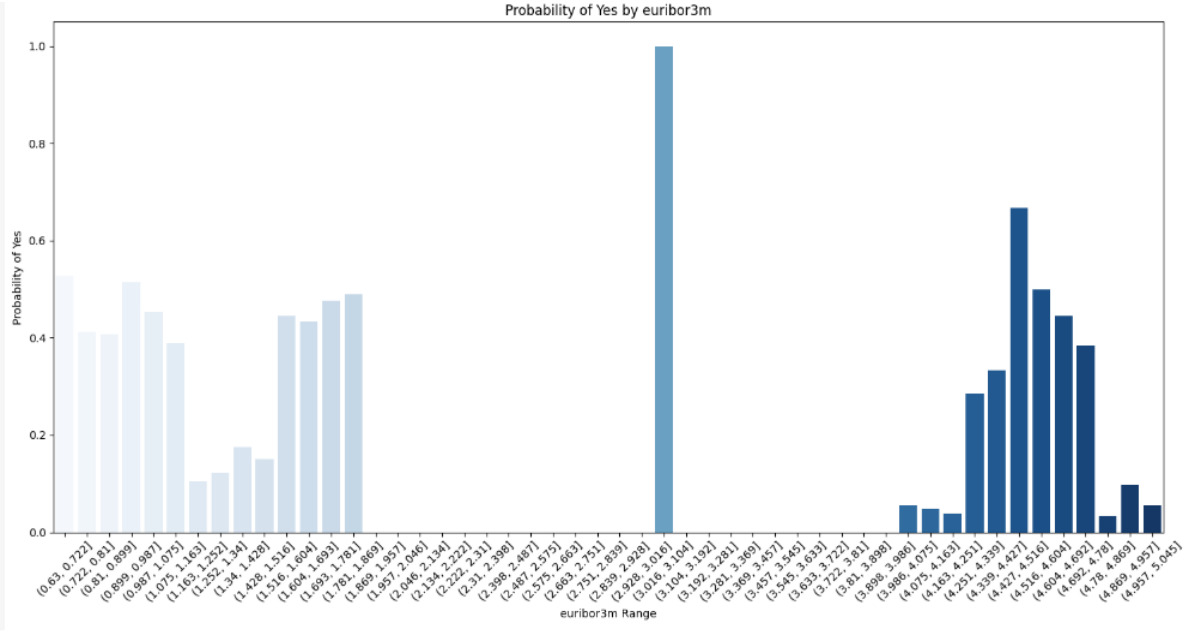
이전 캠페인 결과가 성공인 사람들은 이번에도 정기 예금에 가입할 확률이 높은 것을 알 수 있다.

- **이 캠페인 동안 연락된 횟수 (campaign)**



정기예금 가입 비율이 데이터의 수에 따라 비례하므로 관련 있어 보이지 않았다.

- 금리



금리에 따른 정기 예금 가입 확률 그래프로 금리가 낮거나 클 때만 정기 예금 가입이 이루어지는 것을 확인할 수 있다. (중간의 데이터는 적은 데이터에 의한 것이기 때문에 무시하도록 했다.)

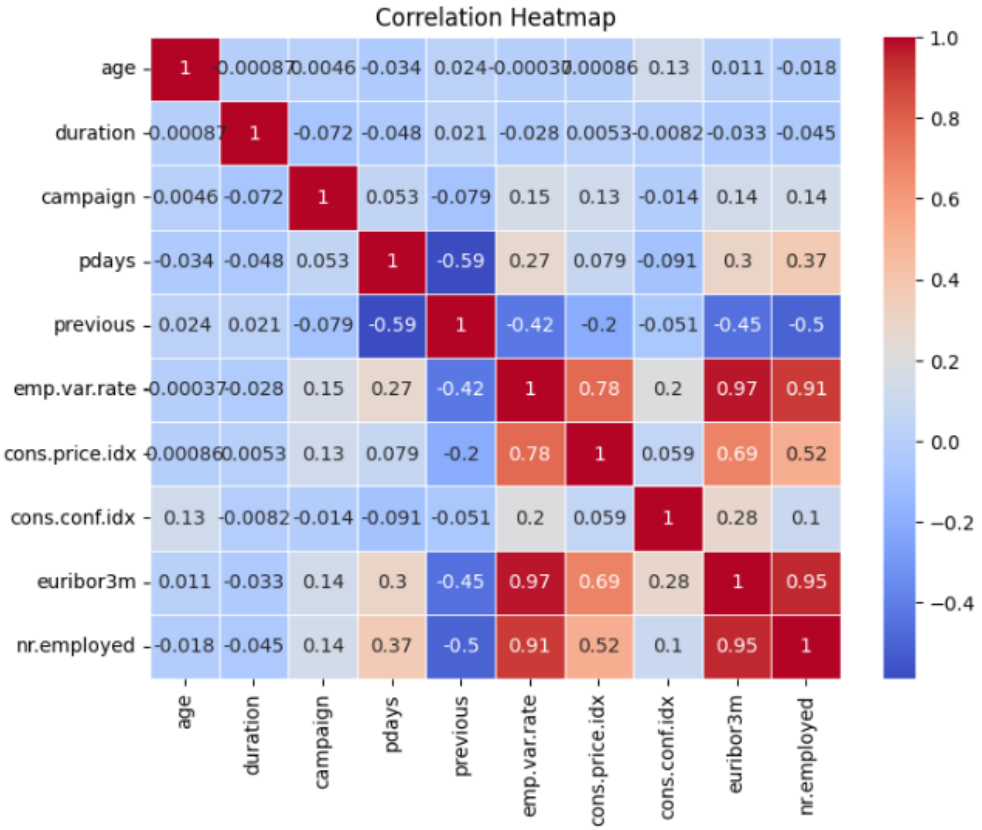
- **Y**

```
# 목표 비율 검색
target_distribution = data['y'].value_counts(normalize=True) * 100
target_distribution
```

target인 y는 정기 예금에 가입 했는지에 관한 정보를 담은 칼럼으로 가입 안한 사람의 비율이 88.73%으로 정기 예금에 가입 한 사람(11.27%)에 비해 매우 많은 것을 확인할 수 있었다.

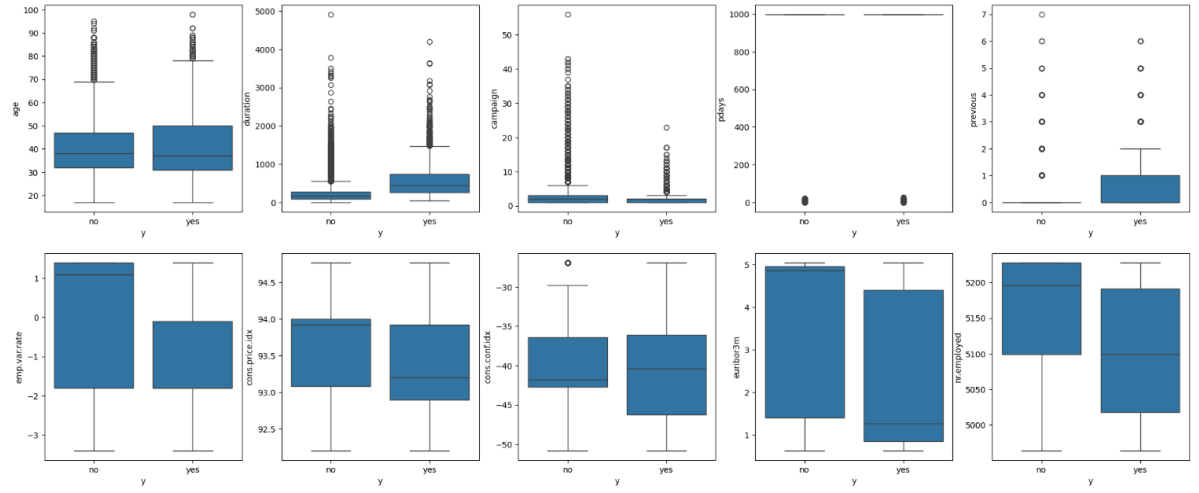
2 - 4. 컬럼간의 상관관계 파악 및 추론

컬럼 간의 상관관계를 알아보기위한 히트맵



- 강한 양의 상관관계 ($corr \geq 0.8$)
  - emp.var.rate ↔ euribor3m
  - emp.var.rate ↔ nr.employed
  - euribor3m ↔ nr.employed
- 양의 상관관계 ($corr \geq 0.4$)
  - emp.var.rate ↔ cons.price.idx
  - cons.price.idx ↔ euribor3m
  - cons.price.idx ↔ nr.employed
- 음의 상관관계 ($corr \leq -0.4$)
  - pdays ↔ previous
  - previous ↔ emp.var.rate
  - previous ↔ euribor3m
  - previous ↔ nr.employed
- y에 따른 수치형 데이터들의 분포

```
cols = bank_num.columns.drop('y')
fig, ax = plt.subplots(2, 5, figsize=(25, 10))
for i, v in enumerate(cols):
    row = i // 5
    col = i % 5
    sns.boxplot(x='y', y=v, data=bank_num, ax=ax[row, col])

plt.show()
```



연락 지속시간과 고객들의 신뢰도가 높을 수록 정기 예금을 가입하는 고객의 수가 많은 것을 확인했다.

3. 가설

EDA 분석 결과에 따라서 정기예금에 가입하게 하는 원인과 결과에 대해 가설들을 만들어 낼 수 있었다.

- 직업군에 따라 유의미한 차이가 존재하는가?
- 대출에 따라 유의미한 차이가 존재하는가?
- 결혼에 따라 유의미한 차이가 존재?
- 통화 지속시간에 따라 유의미한 차이가 존재?
- 연령에 따라 유의미한 차이가 존재?
- 고용 시장에 따라 유의미한 차이가 존재?
- 금리에 따라 유의미한 차이가 존재?

직업군에 따라 유의미한 차이가 존재?

```
import pandas as pd
from scipy.stats import chi2_contingency

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')
```

```
# 가입 여부 그룹 분리 (y를 0과 1로 변환)
df['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 직업별로 가입 여부를 교차표로 생성
contingency_table = pd.crosstab(df['job'], df['y'])

# 카이제곱 검정 수행
chi2, p, dof, expected = chi2_contingency(contingency_table)

# 결과 출력
print("직업별 정기예금 가입 여부 교차표:\n", contingency_table)
print(f"\n카이제곱 통계량: {chi2}")
print(f"p-value: {p}")

if p < 0.05:
    print("결론: 직업군과 정기예금 가입 여부 간에 유의미한 관계가 있습니다.")
else:
    print("결론: 직업군과 정기예금 가입 여부 간에 유의미한 관계가 없습니다.")

# 결과
카이제곱 통계량: 961.2424403289555
p-value: 4.189763287563623e-199
```

직업군과 정기예금 가입 여부에 유의미한 관계가 있는지 확인하기 위해 카이제곱검정을 수행했다. 검정 수행 결과 p-value가 0.05미만으로 직업군과 정기 예금의 가입 여부에는 유의미한 관계가 있는 것을 알 수 있었다.

```
import pandas as pd
from scipy.stats import chi2_contingency

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')

# 가입 여부 그룹 분리
df['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 'job'의 유니크한 직업별로 정기예금 가입 여부를 검정
job_types = df['job'].unique()

# 각 직업별 카이제곱 테스트 수행
for job in job_types:
```

```
    # 해당 직업인 경우와 아닌 경우로 구분
    df[f'is_{job}'] = df['job'].apply(lambda x: 1 if x == job else 0

    # 교차표 생성
    contingency_table = pd.crosstab(df[f'is_{job}'], df['y'])

    # 카이제곱 테스트 수행
    chi2, p, dof, expected = chi2_contingency(contingency_table)

    # 결과 출력
    print(f"\n직업: {job}")
    print("교차표:\n", contingency_table)
    print(f"카이제곱 통계량: {chi2}")
    print(f"p-value: {p}")

    if p < 0.05:
        print(f"결론: {job} 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다
    else:
        print(f"결론: {job} 직업군과 정기예금 가입 간에 유의미한 관계가 없습니다
```

어떤 직업군이 정기예금 가입과 유의미한 관계가 있는지 알아보기 위해 직업별로 카이제곱검정을 수행했다. 그 결과 services, admin, blue-collar, retired, unemployed, entrepreneur, studen의 경우 p-value가 0.05 미만으로 정기예금 가입과 유의미한 관계가 있는 것으로 확인할 수 있었다.

[결과]

직업: housemaid
교차표:
y 0 1
is_housemaid
0 35594 4534
1 954 106
카이제곱 통계량: 1.6153245040208968
p-value: 0.20374495312275925
결론: housemaid 직업군과 정기예금 가입 간에 유의미한 관계가 없습니다.

직업: services
교차표:
y 0 1
is_services
0 32902 4317
1 3646 323
카이제곱 통계량: 42.62782223999282

p-value: 6.621088624908477e-11
결론:
services 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

직업: admin.
교차표:
y 0 1
is_admin.
0 27478 3288
1 9070 1352
카이제곱 통계량: 40.44872983599433
p-value: 2.018436228668527e-10
결론:
admin. 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

직업: blue-collar
교차표:
y 0 1
is_blue-collar
0 27932 4002
1 8616 638
카이제곱 통계량: 227.56951474021812
p-value: 2.0202396474322607e-51
결론:
blue-collar 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

직업: technician
교차표:
y 0 1
is_technician
0 30535 3910
1 6013 730
카이제곱 통계량: 1.505029248720356
p-value: 0.2198991485090137
결론: technician 직업군과 정기예금 가입 간에 유의미한 관계가 없습니다.

직업: retired
교차표:
y 0 1
is_retired
0 35262 4206
1 1286 434

카이제곱 통계량: 348.8343180780503
p-value: 7.602992299255729e-78
결론:
retired 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

직업: management
교차표:
y 0 1
is_management
0 33952 4312
1 2596 328
카이제곱 통계량: 0.0029882437049918503
p-value: 0.95640548855985
결론: management 직업군과 정기예금 가입 간에 유의미한 관계가 없습니다.

직업: unemployed
교차표:
y 0 1
is_unemployed
0 35678 4496
1 870 144
카이제곱 통계량: 8.664698905456135
p-value: 0.003244336054193065
결론:
unemployed 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

직업: self-employed
교차표:
y 0 1
is_self-employed
0 35276 4491
1 1272 149
카이제곱 통계량: 0.8164214647052459
p-value: 0.36622852727762345
결론: self-employed 직업군과 정기예금 가입 간에 유의미한 관계가 없습니다.

직업: unknown
교차표:
y 0 1
is_unknown
0 36255 4603
1 293 37
카이제곱 통계량: 0.0

p-value: 1.0
결론: unknown 직업군과 정기예금 가입 간에 유의미한 관계가 없습니다.

직업: entrepreneur
교차표:
y 0 1
is_entrepreneur
0 35216 4516
1 1332 124
카이제곱 통계량: 11.1265594522361
p-value: 0.0008510029748307555
결론:
entrepreneur 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

직업: student
교차표:
y 0 1
is_student
0 35948 4365
1 600 275
카이제곱 통계량: 361.53079934101333
p-value: 1.3069493135096854e-80
결론:
student 직업군과 정기예금 가입 간에 유의미한 관계가 있습니다.

대출에 따라 유의미한 차이가 존재?

```
import pandas as pd
from scipy.stats import chi2_contingency

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')

# 가입 여부 그룹 분리 (y를 0과 1로 변환)
df['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 주택 대출 및 개인 대출 상태별로 그룹화하여 교차표 생성
housing_contingency_table = pd.crosstab(df['housing'], df['y'])
loan_contingency_table = pd.crosstab(df['loan'], df['y'])
```

```
# 카이제곱 검정 수행 (주택 대출)
chi2_housing, p_housing, dof_housing, expected_housing = chi2_contin

# 카이제곱 검정 수행 (개인 대출)
chi2_loan, p_loan, dof_loan, expected_loan = chi2_contingency(loan_c

# 결과 출력
print("주택 대출 상태별 정기예금 가입 여부 교차표:\n", housing_contingency_t
print(f"주택 대출과 정기예금 가입 간 카이제곱 통계량: {chi2_housing}, p-value

print("개인 대출 상태별 정기예금 가입 여부 교차표:\n", loan_contingency_tabl
print(f"개인 대출과 정기예금 가입 간 카이제곱 통계량: {chi2_loan}, p-value: {

# 결론
if p_housing < 0.05:
    print("주택 대출 상태와 정기예금 가입 간에 유의미한 관계가 있습니다.")
else:
    print("주택 대출 상태와 정기예금 가입 간에 유의미한 관계가 없습니다.")

if p_loan < 0.05:
    print("개인 대출 상태와 정기예금 가입 간에 유의미한 관계가 있습니다.")
else:
    print("개인 대출 상태와 정기예금 가입 간에 유의미한 관계가 없습니다.")
```

가설을 증명하기 위해 주택 대출과 개인 대출을 이용하여 카이제곱검정을 수행했다. 주택 대출의 경우 p-value가 0.0582로 유의미한 관계가 없는 것으로 나타났으나 임계값 근처의 값이기 때문에 유의해서 살펴볼 필요가 있다. 개인 대출의 경우 p-value가 0.5786로 유의미한 관계가 없는 것으로 나타났다.

```
import pandas as pd
from scipy.stats import chi2_contingency

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')

# 가입 여부 그룹 분리 (y를 0과 1로 변환)
df['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 주택 대출과 개인 대출을 합산한 대출 상태 생성
df['has_loan'] = df.apply(lambda row: 1 if row['housing'] == 'yes' o
```

```
# 교차표 생성 (대출 상태 vs 정기예금 가입 여부)
loan_contingency_table = pd.crosstab(df['has_loan'], df['y'])

# 카이제곱 검정 수행
chi2, p, dof, expected = chi2_contingency(loan_contingency_table)

# 결과 출력
print("대출 상태별 정기예금 가입 여부 교차표:\n", loan_contingency_table)
print(f"\n카이제곱 통계량: {chi2}")
print(f"p-value: {p}")

# 결론
if p < 0.05:
    print("대출 상태와 정기예금 가입 간에 유의미한 관계가 있습니다.")
    if loan_contingency_table.loc[0, 1] > loan_contingency_table.loc
        print("대출이 없는 고객이 대출이 있는 고객보다 정기예금에 더 많이 가입합
    else:
        print("대출이 있는 고객이 대출이 없는 고객보다 정기예금에 더 많이 가입합
else:
    print("대출 상태와 정기예금 가입 간에 유의미한 관계가 없습니다.")
```

대출 여부에 따라서도 차이가 있는지 알아보기 위해 대출이 있는지에 대한 정보를 담은 has_loan 칼럼을 신설하고 카이제곱 검정을 수행했다. 수행 결과 p-value가 0.0504로 대출 상태는 정기예금 가입과 유의미한 관계가 없다는 결과가 나왔다. 하지만 임계값 근처의 값이기 때문에 완전히 관계가 없다고는 할 수 없다.

```
import pandas as pd
from scipy.stats import chi2_contingency

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')

# 가입 여부 그룹 분리 (y를 0과 1로 변환)
df['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 대출 상태에 따른 그룹 분류
def classify_loan(row):
    if row['housing'] == 'no' and row['loan'] == 'no':
        return 'No Loan'
    elif row['housing'] == 'yes' and row['loan'] == 'yes':
        return 'Both Loans'
```

```
    else:
        return 'One Loan'

# 새로운 대출 상태 컬럼 생성
df['loan_status'] = df.apply(classify_loan, axis=1)

# 교차표 생성 (대출 상태 vs 정기예금 가입 여부)
loan_contingency_table = pd.crosstab(df['loan_status'], df['y'])

# 카이제곱 검정 수행
chi2, p, dof, expected = chi2_contingency(loan_contingency_table)

# 결과 출력
print("대출 상태별 정기예금 가입 여부 교차표:\n", loan_contingency_table)
print(f"\n카이제곱 통계량: {chi2}")
print(f"p-value: {p}")

# 결론
if p < 0.05:
    print("대출 상태와 정기예금 가입 간에 유의미한 관계가 있습니다.")
else:
    print("대출 상태와 정기예금 가입 간에 유의미한 관계가 없습니다.")
```

실행중인 대출의 종류 수가 정기 예금의 가입에 유의미한 관계가 있는지 알아보기 위해 주택, 개인 대출 모두 없는 경우, 모두 있는 경우, 둘 중 한 개만 가진 경우로 나눠 loan_status 칼럼을 만들고 카이제곱 검정을 수행했다. 수행 결과 p-value가 0.1253이기 때문에 실행중인 대출의 종류 수는 정기예금 가입에 유의미한 영향을 주지 않는다.

```
import pandas as pd
from scipy.stats import f_oneway

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')

# 가입 여부 그룹 분리 (y를 0과 1로 변환)
df['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 대출 상태에 따른 그룹 분류
def classify_loan(row):
    if row['housing'] == 'no' and row['loan'] == 'no':
        return 'No Loan'
```

```
    elif row['housing'] == 'yes' and row['loan'] == 'yes':
        return 'Both Loans'
    else:
        return 'One Loan'

# 새로운 대출 상태 컬럼 생성
df['loan_status'] = df.apply(classify_loan, axis=1)

# 각 그룹의 y 값 분리 (정기예금 가입 여부)
no_loan_group = df[df['loan_status'] == 'No Loan']['y']
one_loan_group = df[df['loan_status'] == 'One Loan']['y']
both_loans_group = df[df['loan_status'] == 'Both Loans']['y']

# ANOVA 검정 수행 (세 그룹 간 차이)
f_stat, p_value = f_oneway(no_loan_group, one_loan_group, both_loans

# 결과 출력
print(f"ANOVA F-통계량: {f_stat}")
print(f"p-value: {p_value}")

# 결론
if p_value < 0.05:
    print("세 그룹 간에 정기예금 가입 여부에 유의미한 차이가 있습니다.")
else:
    print("세 그룹 간에 정기예금 가입 여부에 유의미한 차이가 없습니다.")
```

위에서 나눈 세 집단간 차이가 있는지 알아보기 위해 ANOVA 분석을 진행했다. 분석 결과 p-value는 0.1253으로 세 집단간 차이가 없는 것으로 나타났다.

결혼에 따라 유의미한 차이가 존재?

```
import pandas as pd
from scipy.stats import chi2_contingency

# 데이터 불러오기
df = pd.read_csv('bank-additional-full.csv', sep=';')

# 결혼 상태와 y값의 빈도표 생성
contingency_table = pd.crosstab(df['marital'], df['y'])

# 카이제곱 검정 수행
chi2, p, dof, expected = chi2_contingency(contingency_table)
```

```
# 결과 출력
print(f"Chi-squared: {chi2}")
print(f"p-value: {p}")

# 결혼 상태와 y값 간의 관계가 유의미한지 여부
if p < 0.05:
    print("귀무가설을 기각. 결혼상태에 따라 정기 예금 가입의 차이가 존재")
else:
    print("귀무가설을 채택.")
```

가설을 증명하기 위해 결혼 상태를 기준으로 카이제곱검정을 수행했다. 수행 결과 p-value가 2.0680e-26으로 결혼상태에 따라 정기 예금의 가입에 차이가 있음을 알 수 있다.

통화 지속시간에 따라 유의미한 차이가 존재?

```
import pandas as pd
from scipy.stats import ttest_ind

# 데이터 불러오기
df = pd.read_csv('bank-additional-full.csv', sep=';')

# y값이 yes인 그룹과 no인 그룹의 통화 지속시간 추출
duration_yes = df[df['y'] == 'yes']['duration']
duration_no = df[df['y'] == 'no']['duration']

# 두 그룹 간의 평균 차이에 대한 t-검정 수행
t_stat, p_value = ttest_ind(duration_yes, duration_no, equal_var=Fal

# 결과 출력
print(f"T-statistic: {t_stat}")
print(f"p-value: {p_value}")

# 통화 지속시간에 따른 y값의 차이가 유의미한지 여부
if p_value < 0.05:
    print("통화 지속시간에 따라 y값에 유의미한 차이가 있다.")
else:
    print("통화 지속시간에 따라 y값에 유의미한 차이가 없다.")
```

가설을 증명하기 위해 통화 지속시간을 기준으로 t-test를 수행했다. 수행 결과 p-value가 0으로 매우 유의미한 차이가 있음을 알 수 있다.

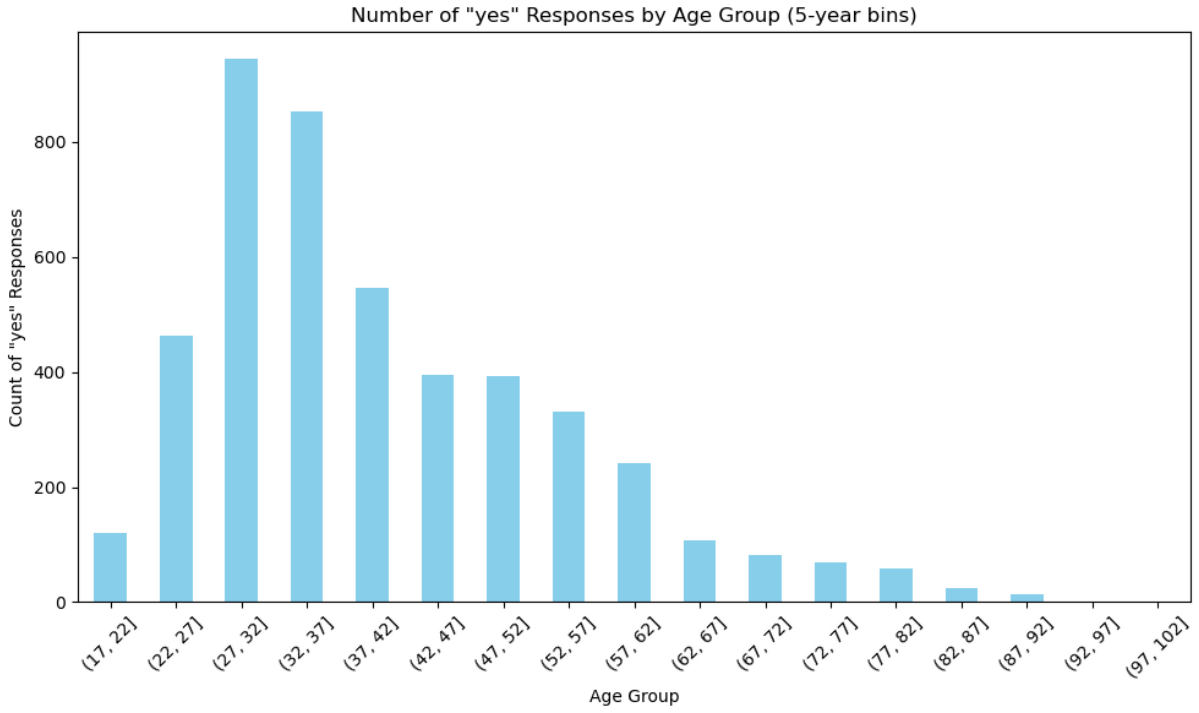
연령에 따라 유의미한 차이가 존재?

```
import pandas as pd
from scipy.stats import ttest_ind

# 데이터 불러오기
df = pd.read_csv('bank-additional-full.csv', sep=';')

# y값이 yes인 그룹과 no인 그룹의 연령 추출
age_yes = df[df['y'] == 'yes']['age']
age_no = df[df['y'] == 'no']['age']

# 두 그룹 간의 평균 차이에 대한 t-검정 수행
t_stat, p_value = ttest_ind(age_yes, age_no, equal_var=False)

# 결과 출력
print(f"T-statistic: {t_stat}")
print(f"p-value: {p_value}")

# 연령에 따른 y값의 차이가 유의미한지 여부
if p_value < 0.05:
    print("연령에 따라 y값에 유의미한 차이가 있다.")
else:
    print("연령에 따라 y값에 유의미한 차이가 없다.")
```

가설을 증명하기 위해 연령을 기준으로 t-test를 수행했다. 수행 결과 p-value가 1.8047e-06으로 연령에 따라 정기 예금 가입에 유의미한 차이가 있는 것을 알 수 있다.

```
import pandas as pd

# y값이 yes인 데이터만 필터링
yes_df = df[df['y'] == 'yes'].copy()  # .copy()로 경고 방지

# 나이를 5살 간격으로 구간화
age_bins = range(yes_df['age'].min(), yes_df['age'].max() + 5, 5)
yes_df.loc[:, 'age_group'] = pd.cut(yes_df['age'], bins=age_bins)

# 각 나이 구간별 빈도 계산
```

```
age_group_distribution = yes_df['age_group'].value_counts().sort_ind

# 히스토그램 그리기
plt.figure(figsize=(10,6))
age_group_distribution.plot(kind='bar', color='skyblue')
plt.title('Number of "yes" Responses by Age Group (5-year bins)')
plt.xlabel('Age Group')
plt.ylabel('Count of "yes" Responses')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



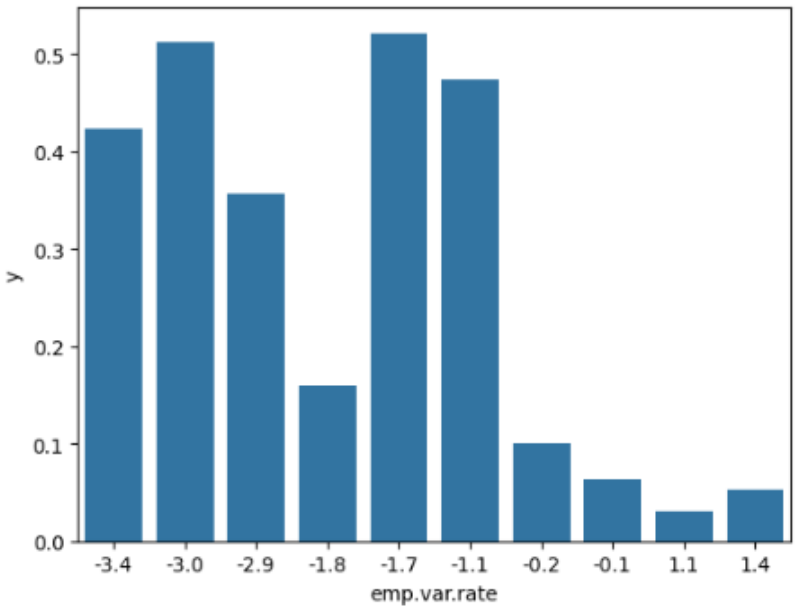
어떤 연령대가 가장 많이 가입하는지 알아보기 위해 나이를 5씩 끊어서 히스토그램으로 그려보았다.
그 결과 20-40대에서 가장 많이 정기 예금에 가입하는 것을 알 수 있었다.

고용 시장에 따라 유의미한 차이가 존재?

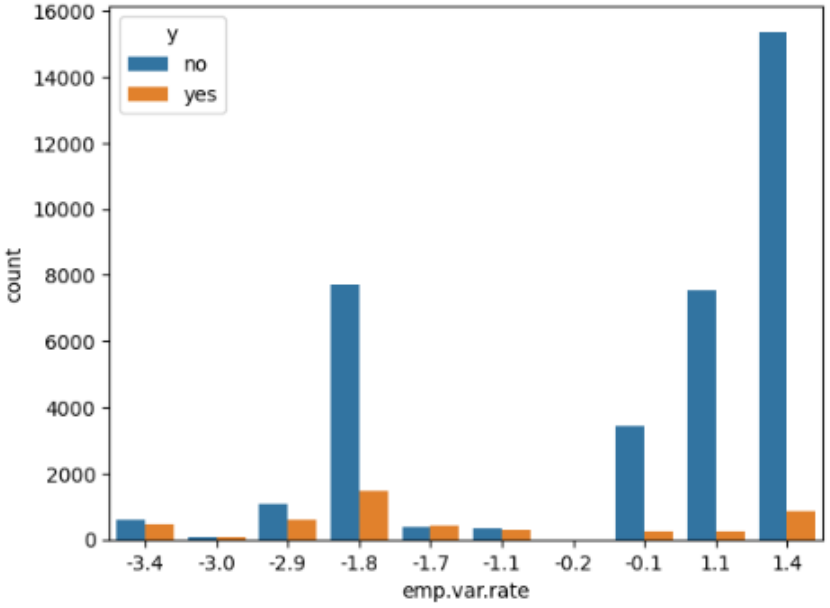
가설은 “고용시장이 활발하지 않을 때 안정적인 투자를 위해 정기 예금에 가입할 것이다”로 수립했다.

```
y_prob = bank.groupby('emp.var.rate')['y'].apply(lambda x : (x == 'y
```

```
sns.barplot(x='emp.var.rate', y='y', data = y_prob)
plt.show()
```

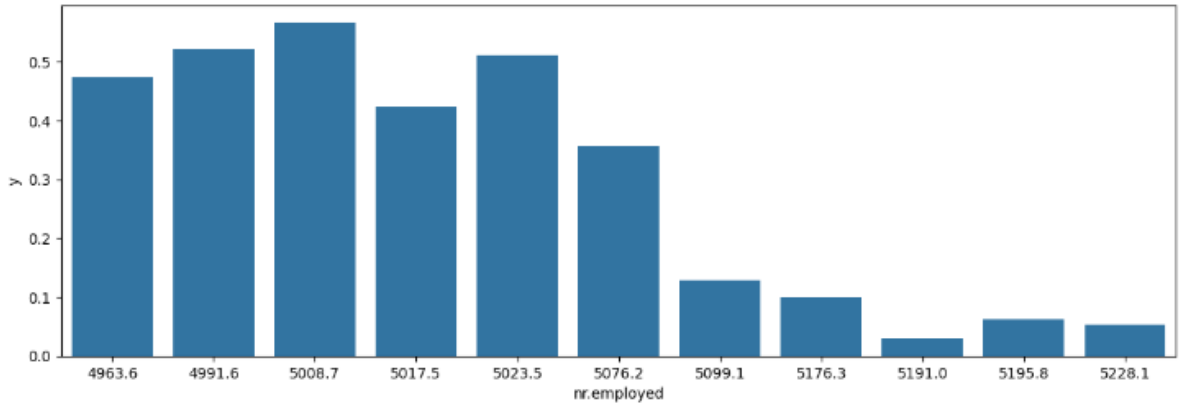



```
sns.countplot(x='emp.var.rate', data=bank, hue='y')
plt.show()
```



고용 시장이 활발할 때보다 활발하지 않을 때의 데이터 수가 적긴 하지만 정기 예금에 가입한 사람들의 수는 고용 시장이 활발할 때보다 많은 것을 확인할 수 있다. 즉, 고용 시장이 활발하지 않을 때, 고객들의 정기 예금 가입 확률이 높은 것을 볼 수 있다.

```
cols = bank_num.columns.drop('y')
fig, ax = plt.subplots(5,2, figsize=(30,25))
for i, v in enumerate(cols):
    y_prob = bank.groupby(v)['y'].apply(lambda x: (x == 'yes').mean(
    row = i // 2
    col = i % 2
    sns.barplot(x=v, y='y', data=y_prob, ax=ax[row, col])
plt.show()
```



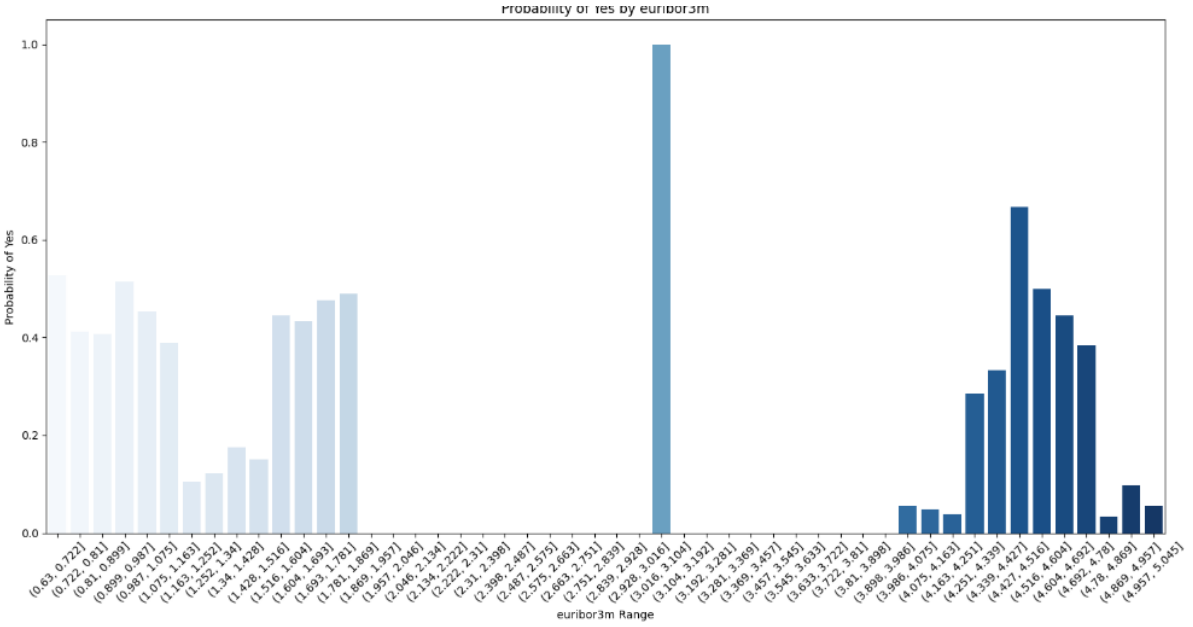
위의 내용을 근거하는 자료로, 고용 시장이 활발하지 않아 직원 수가 적을 수록 정기 예금 가입 확률이 높은 것을 보여준다.

금리에 따라 유의미한 차이가 존재?

가설은 “금리가 상승함에 따라 많은 이자를 받기 위해 정기예금 가입률이 올라갈 것이다”로 수립했다.

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# 원래 데이터프레임 bank 사용
# 'interest_rate'라는 칼럼이 금리라고 가정

# 금리 데이터로 y='yes'의 확률 계산
bins = 50  # 구간 수 설정
euribor3m_bins = pd.cut(bank['euribor3m'], bins=bins)

# 각 구간에 대해 y가 'yes'인 확률 계산
y_prob = bank.groupby(euribor3m_bins)['y'].apply(lambda x: (x == 'ye
```

```
# Barplot 생성
plt.figure(figsize=(15, 8))
sns.barplot(x='euribor3m', y='probability', data=y_prob, palette='Bl
plt.title('Probability of Yes by euribor3m')
plt.xlabel('euribor3m Range')
plt.ylabel('Probability of Yes')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



위의 그래프를 보았을 때 금리가 높을 때 정기 예금에 가입 하지만, 금리가 낮을 때 또한 정기 예금에 가입하는 것을 볼 수 있다. 그러면 금리가 낮을 때 정기 예금에 가입하는 고객들의 특징은 무엇인지 알아보았다.

[50]:
```
low_euri.describe(include='object')
```

[50]:

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	y
count	13453	13453	13453	13453	13453	13453	13453	13453	13453	13453	13453
unique	12	4	8	2	3	3	2	10	5	3	2
top	admin.	married	university.degree	no	yes	no	cellular	may	mon	nonexistent	no
freq	3580	7316	4164	12035	7580	11080	12129	6006	2994	8563	10174

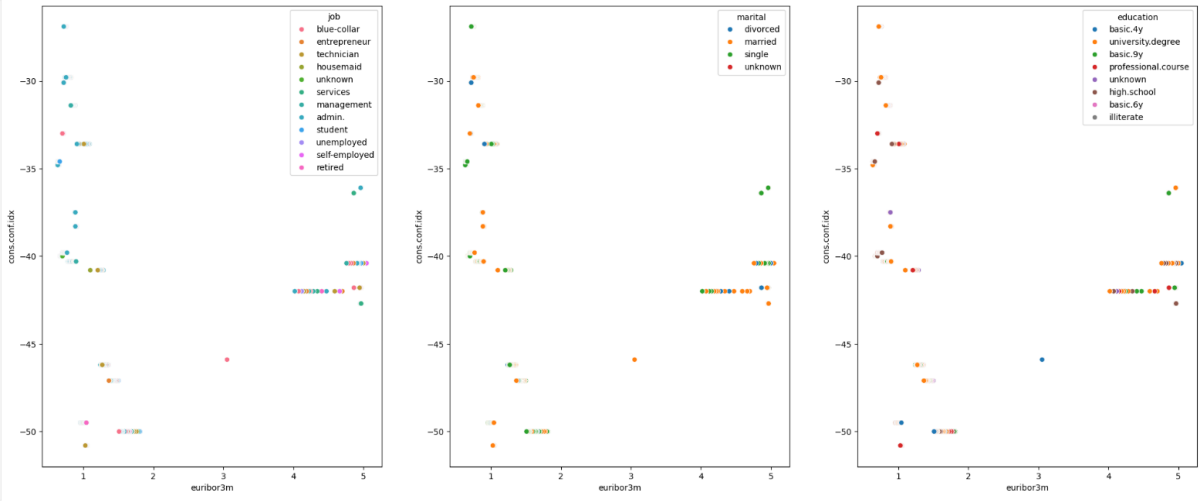
[49]:
```
high_euri.describe(include='object')
```

[49]:

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome	y
count	27681	27681	27681	27681	27681	27681	27681	27681	27681	27681	27681
unique	12	4	8	3	3	3	2	7	5	3	2
top	admin.	married	university.degree	no	yes	no	cellular	may	wed	nonexistent	no
freq	6826	17587	7983	20499	13959	22825	13974	7763	5713	26948	26343

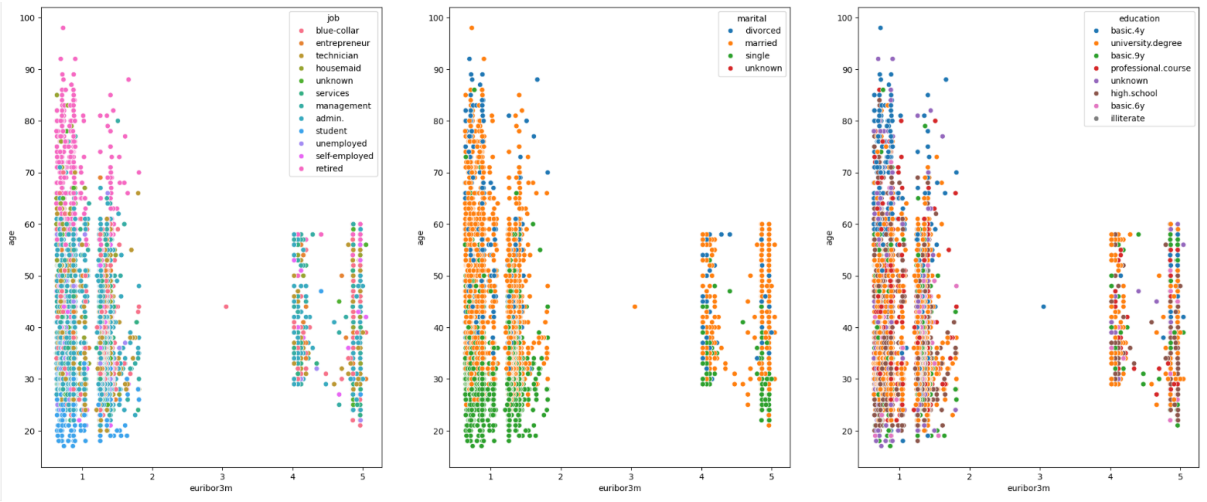
금리에 따라 분리해봤을 때 범주형 데이터에 대해선 별다른 차이가 없어보인다. 경제 전망에 대한 불안감을 가지는 사람들이 정기 예금에 가입하는 것은 아닐 수 있다는 생각을 했으며 이를 확인해보았다.

```
cols = ['job','marital','education']

yes_data = bank[bank['y'] == 'yes']

fig, ax = plt.subplots(1,3, figsize=(25,10))
for i, v in enumerate(cols):
    # row = i // 3
    col = i % 3
    sns.scatterplot(x='euribor3m', y='cons.conf.idx', hue=v, data=ye

plt.show()
```



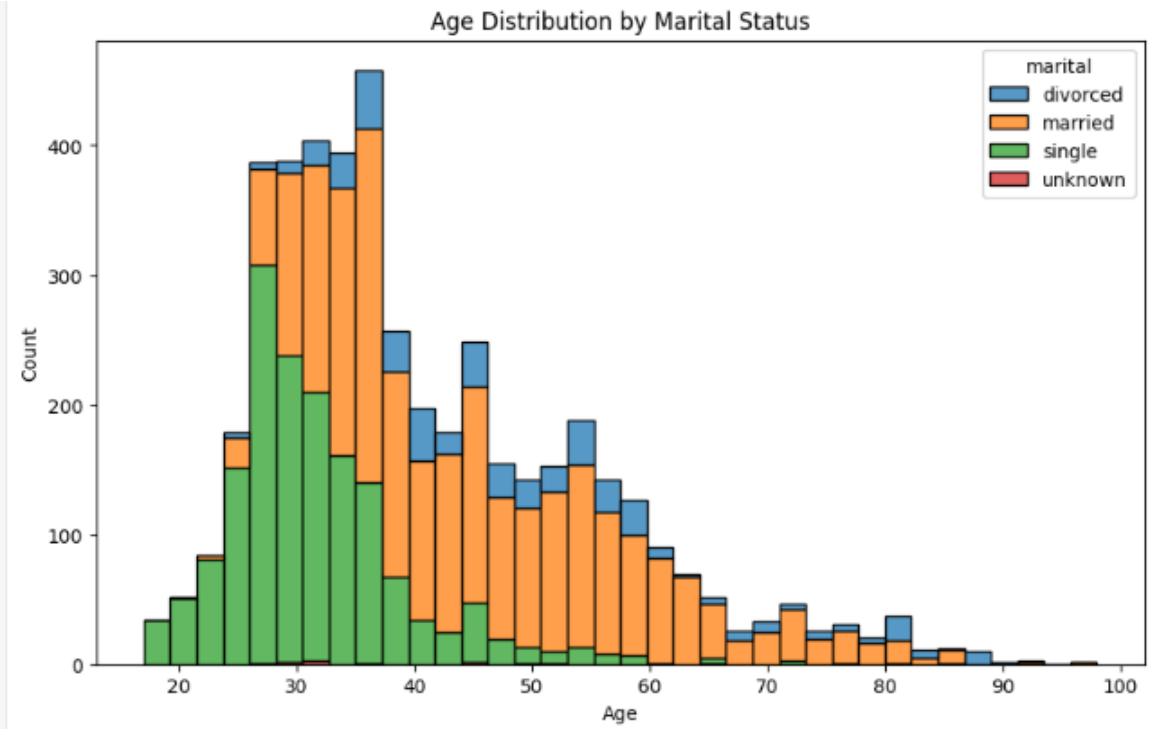
위는 정기 예금을 가입한 사람들에 대한 데이터로 그린 그래프로, 경제 전망에 대한 불안감을 가지는 사람들이 정기 예금에 가입하는 것은 아닌 것으로 보였다. 그렇다면 나이가 많은 사람들이 원금 보존(저축)을 목적으로 정기 예금에 가입하는 것이 아닐까라는 생각을 해보고 이것을 확인해 보았다.

```
cols = ['job','marital','education']

yes_data = bank[bank['y'] == 'yes']

fig, ax = plt.subplots(1,3, figsize=(25,10))
for i, v in enumerate(cols):
    # row = i // 3
    col = i % 3
    sns.scatterplot(x='euribor3m', y='age', hue=v, data=yes_data, ax

plt.show()
```



위의 그래프로 볼 때 금리가 낮을 때 나이가 많은 사람들이 저축을 목적으로 정기 예금에 가입한다는 사실을 볼 수 있다. 하지만 나이가 어린 사람들 또한 정기 예금에 가입하는 사실을 확인 할 수 있다. 나이가 어린 사람들이 정기 예금에 가입하는 것은, 결혼을 해서 저축을 하기 위함이 아닐까라는 생각을 했고 이를 확인했다.

```
yes_data = bank[bank['y']=='yes']

plt.figure(figsize=(10, 6))
sns.histplot(data=yes_data, x='age', hue='marital', multiple='stack'
plt.title('Age Distribution by Marital Status')
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```



위의 그래프로 보았을 때, 나이가 어린 사람들이 정기 예금에 가입하는 이유는 결혼 때문은 아닌 것으로 볼 수 있었다.

4. 잠재고객 예측을 위한 머신러닝 모델 활용

XG Boost

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV, 
from sklearn.metrics import accuracy_score, precision_score, recall_
from sklearn.metrics import f1_score, roc_auc_score
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# 데이터 로드
df = pd.read_csv('bank-additional-full.csv', sep=';')
df = df.drop_duplicates()

# 직업군을 역순으로 정의
job_order = [
    "management",
```

```
    "self-employed",
    "entrepreneur",
    "technician",
    "blue-collar",
    "admin.",
    "services",
    "housemaid",
    "retired",
    "student",
    "unemployed",
    "unknown"
]

# 직업군을 정수로 변환
df['job'] = df['job'].apply(lambda x: job_order.index(x))

# default 변수 인코딩
df['default'] = df['default'].apply(lambda x: -1 if x == 'no' else (

# 나머지 범주형 변수를 원-핫 인코딩
df_encoded = pd.get_dummies(df, columns=['marital', 'housing', 'loan

# 교육 인코딩
education_order = ["unknown", "basic.4y", "basic.6y", "basic.9y", "h
                   "professional.course", "university.degree"]
df_encoded['education'] = df['education'].apply(lambda x: education_

# 월과 요일 인코딩
month_order = ["jan", "feb", "mar", "apr", "may", "jun", "jul", "aug
df_encoded['month'] = df['month'].apply(lambda x: month_order.index(
day_of_week_order = ["mon", "tue", "wed", "thu", "fri"]
df_encoded['day_of_week'] = df['day_of_week'].apply(lambda x: day_of_

# 타겟 변수 인코딩
df_encoded['y'] = df['y'].apply(lambda x: 1 if x == 'yes' else 0)

# 나이를 구간으로 묶기
df_encoded['age_group'] = pd.cut(df['age'], bins=np.arange(0, 101, 1

# housing과 loan 변수를 하나로 통합
df_encoded['loan_combined'] = df_encoded['housing_yes'] + df_encoded

# 결혼 여부 변수를 0, 1, 2로 변환
```

```
df_encoded['marital_status'] = np.select(
    [df_encoded['marital_unknown'] == 1, df_encoded['marital_single'
    [0, 1, 2],
    default=0  # 나머지 경우 기본값으로 0 처리
)

# 기존 marital 관련 변수 삭제
X = df_encoded.drop(['y', 'duration', 'age', 'marital_unknown', 'mar
                     'housing_no', 'housing_yes', 'loan_no', 'loan_y
y = df_encoded['y']
```

우선 데이터 전처리를 진행했다.

- job: 유럽 평균 임금 기반으로 평균 임금이 낮을 수록 작은 값을 갖도록 정수로 인코딩했다.
- default: no인 경우 -1, yes인 경우 1, 이외는 0으로 인코딩했다.
- education: 학력이 높아질 수록 큰 값을 갖도록 순차적인 정수로 인코딩했다.
- month: 각 월에 맞는 숫자로 인코딩했다.
- day_of_week: 월요일부터 금요일까지 1부터 시작하는 순차적인 정수로 인코딩했다.
- y: yes인 경우 1로 이외의 경우는 0으로 인코딩했다.
- age: 0살부터 10살 단위로 구간화하여 age_group이라는 변수를 파생했다.
- housing, loan: 주택 대출과 개인 대출이 모두 없는 경우 0, 하나만 있는 경우 1, 둘 다 있는 경우 2를 갖는 loan_combined 변수를 파생했다.
- marital_status: 불분명하거나 이혼인 경우 0, 싱글인 경우 1, 기혼인 경우 2로 인코딩했다.
- 이외 기타 범주형 변수들은 원 핫 인코딩으로 진행 후 학습에 사용되지 않는 열을 drop 시키고 Feature와 Label로 분리한다.

```
# 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=

# 스케일링
scaler = StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X.colu
X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)

# SMOTE 적용
smote = SMOTE(random_state=156)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y
```

train set : test set = 8 : 2로 분할했으며 Standardscaler를 사용하여 데이터를 스케일링 했다. 데이터에서 y가 yes인 비율이 11%이기 때문에 SMOTE를 사용하여 데이터의 비율을 맞춰주었다.

```
# 하이퍼파라미터 튜닝
param_grid = {
    'max_depth': [5, 7, 9],
    'min_child_weight': [2, 3, 4],
    'subsample': [0.4, 0.6, 0.8],
    'eta': [0.01, 0.1]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=156)
grid_search = GridSearchCV(xgb.XGBClassifier(objective='binary:logis
                           param_grid, scoring='f1', cv=cv)
grid_search.fit(X_train_resampled, y_train_resampled)
print("Best parameters found: ", grid_search.best_params_)

# 최적 모델을 사용한 예측
best_model = grid_search.best_estimator_
pred_probs = best_model.predict_proba(X_test)[:, 1]

# 최적 임계값 찾기
thresholds = np.arange(0, 0.9, 0.01)
f1_scores = []

for threshold in thresholds:
    preds = (pred_probs > threshold).astype(int)
    f1 = f1_score(y_test, preds)
    f1_scores.append(f1)

best_threshold = thresholds[np.argmax(f1_scores)]
print(f'Best threshold: {best_threshold}')

# 최적 임계값으로 예측
preds = (pred_probs > best_threshold).astype(int)
```

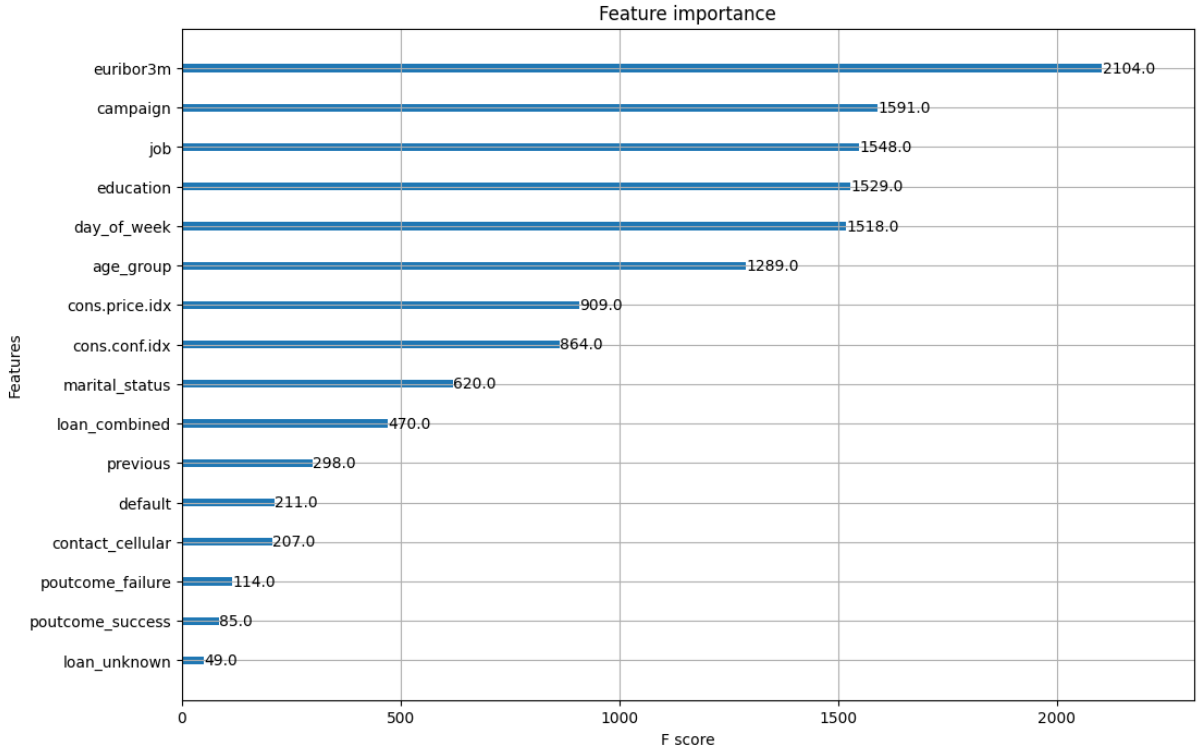
XGBoost를 사용할 때 GridSearchCV를 사용하여 여러 경우의 수를 용이하게 튜닝했다. 튜닝 과정에서 StratifiedKFold를 적용하여 속성값의 개수를 동일하게 맞춰 학습이 더 정확하게 진행되도록 했다.

```
# 분류기 평가 함수
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred)
    recall = recall_score(y_test, pred)
```

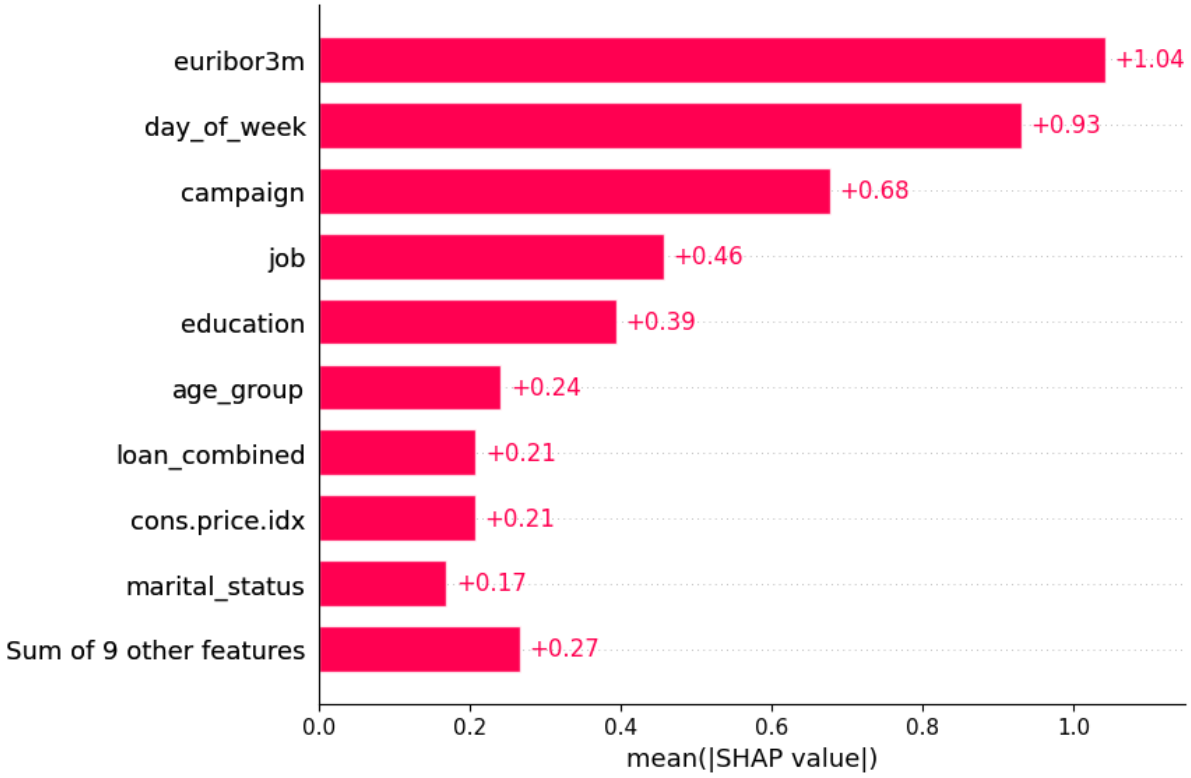
```
    f1 = f1_score(y_test, pred)
    auc = roc_auc_score(y_test, pred_proba)

    print("Confusion Matrix:")
    print(confusion)
    print(f"Accuracy: {accuracy:.4f}, Precision: {precision:.4f}, Re

# 모델 평가
get_clf_eval(y_test, pred=preds, pred_proba=pred_probs)
```

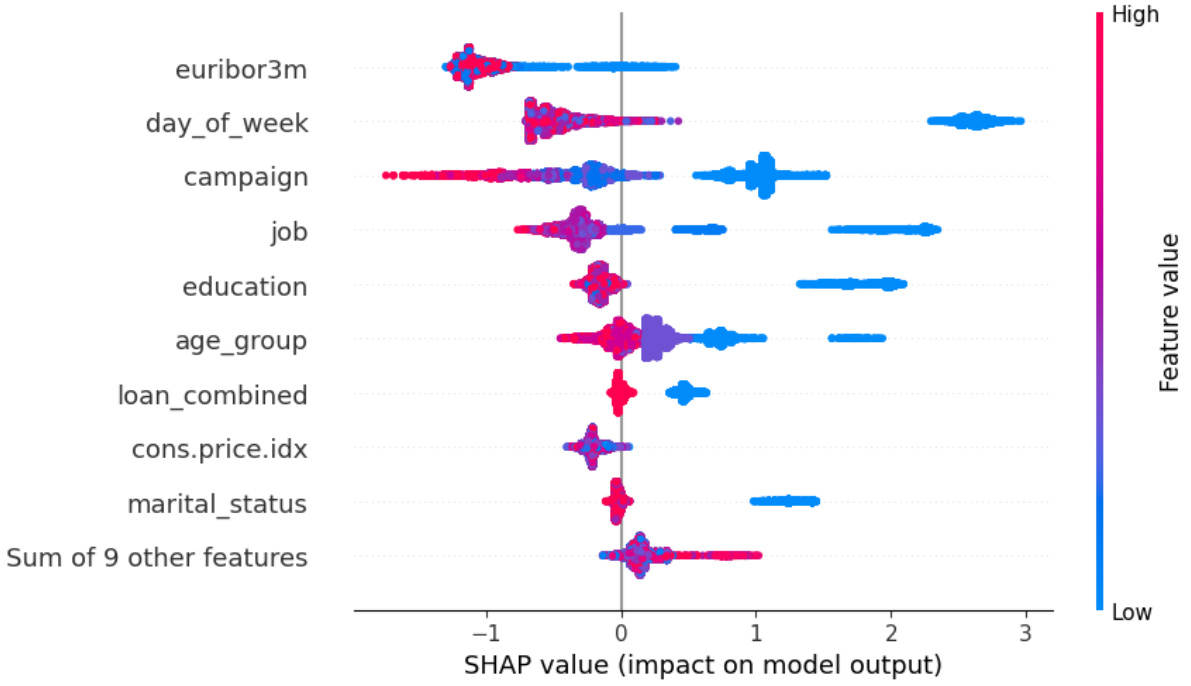
만들어진 모델을 평가하는 함수를 정의하고 test 데이터를 활용해 모델을 검증하는 과정을 거쳤다. 현재 가장 높은 스코어는 Accuracy: 0.8718, Precision: 0.4452, Recall: 0.5359, F1: 0.4864, AUC: 0.7788이며 이때 사용된 hyperparameter는 eta: 0.1, max_depth: 9, min_child_weight: 2, subsample: 0.8이다.

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(12, 8))
xgb.plot_importance(best_model, ax = ax)
plt.show()
```



위의 그림은 학습된 XGBoost model의 변수중요도 그래프다. 모델은 금리, 연락 횟수, 직업, 학업 순으로 변수를 중요하게 생각하고 있었다.





샤플리 밸류를 사용하여 변수의 중요도를 확인해본 결과 유럽 3개월 금리, 요일, 기간동안 연락된 횟수, 직업을 중요하게 보는 것으로 확인할 수 있다. 유럽의 금리는 낮은 경우, 요일은 월요일에 가까울수록, 연락된 횟수는 적을수록 정기 예금의 가입에 좋은 영향을 미쳤다.

랜덤 포레스트

```
# 데이터 불러오기
file_path = 'bank-additional-full.csv'
data = pd.read_csv(file_path, delimiter=';')

# 'duration' 컬럼 삭제
data = data.drop(columns=['duration'])

# 범주형 변수 처리
categorical_cols = ['job', 'marital', 'education', 'default', 'housi
                    'contact', 'month', 'day_of_week', 'poutcome']

# 범주형 변수들을 Label Encoding
le = LabelEncoder()
for col in categorical_cols:
    data[col] = le.fit_transform(data[col].astype(str))

# 'y' 타겟 변수 이진 처리
data['y'] = data['y'].map({'yes': 1, 'no': 0})

# Pdays 컬럼에서 999를 -1로 처리
data['pdays'] = data['pdays'].replace(999, -1)

# 입력 변수(X)와 타겟 변수(y) 나누기
X = data.drop(columns=['y'])
y = data['y']

# 데이터셋 분할 (Train 60%, Validation 20%, Test 20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_

# 최적 하이퍼파라미터로 랜덤포레스트 모델 초기화
rf_model = RandomForestClassifier(n_estimators=200, max_depth=10, mi
                                  min_samples_leaf=4, bootstrap=True

# 모델 학습
rf_model.fit(X_train, y_train)

# 검증 세트에 대한 예측
y_val_pred = rf_model.predict(X_val)

# 검증 세트의 정확도, 혼동 행렬, 분류 리포트
print(f"Validation Accuracy: {accuracy_score(y_val, y_val_pred)}")
```

```
print("Confusion Matrix:")
print(confusion_matrix(y_val, y_val_pred))
print("Classification Report:")
print(classification_report(y_val, y_val_pred))
```

```
Validation Accuracy: 0.9016751638747269
Confusion Matrix:
[[7181  129]
 [ 681  247]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.95      7310
           1       0.66      0.27      0.38       928

    accuracy                           0.90      8238
   macro avg       0.79      0.62      0.66      8238
weighted avg       0.88      0.90      0.88      8238
```

전처리

'duration' 컬럼은 데이터 분석에서 중요한 예측 변수가 아니기 때문에 모델에서 제거한다.

데이터셋에 있는 범주형 변수들(job, marital, education, default, housing, loan, contact, month, day_of_week, poutcome)을 숫자로 변환해야 한다.

목표 변수인 'y'는 'yes'와 'no'로 되어있는데, 이를 1(예금 가입)과 0(예금 미가입)으로 매핑한다. 즉, {'yes': 1, 'no': 0}으로 변환한다.

pdays는 이전 캠페인에서 마지막으로 연락한 후 경과한 일수를 나타내는데, 999는 이전에 연락한 적이 없다는 의미이다. 이를 -1로 변경하여 특수한 값을 처리한다.

데이터를 학습용(60%), 검증용(20%), 테스트용(20%)으로 나눈다.

먼저 학습용 데이터를 60%로 나누고, 나머지 40%는 검증용과 테스트용으로 나누어 각각 20%씩 사용한다.

- *클래스 0 (가입하지 않음)**에 대해서는 모델이 매우 높은 성능을 보인다. **정밀도(Precision)** 0.91, **재현율(Recall)** 0.98, **F1-스코어(F1-score)** 0.95로, 예금에 가입하지 않은 사람들을 거의 완벽하게 예측하고 있다.

- 반면, **클래스 1 (가입)**에 대한 성능은 다소 낮다. **정밀도**는 0.66으로 예금에 가입한 사람 중 66%를 정확하게 예측했으나, **재현율**이 0.27에 불과하여 예금에 가입한 사람들의 27%만을 제대로 잡아내고 있다. 이로 인해 **F1-스코어**는 0.38로 낮게 나왔는데, 이는 특히 재현율의 부족으로 인해 성능이 떨어진 결과이다.

yes 클래스(정기예금에 가입한 고객)에 대한 예측 성능이 떨어지고 있다. 높이는 방향으로 개선이 이루어져야 한다.

yes 부분의 모델 성능향상을 위한 처리 시도

1 . 하이퍼 파라미터 튜닝

GridSearchCV

```
from sklearn.model_selection import GridSearchCV

# 하이퍼파라미터 그리드 설정
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# 랜덤포레스트 모델을 사용한 GridSearch
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid

# 학습
grid_search.fit(X_train, y_train)

# 최적의 하이퍼파라미터
print(f"Best Hyperparameters: {grid_search.best_params_}")
```

bootstrap': True, 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 200

2. SMOTE(Synthetic Minority Over-sampling Technique)

소수 클래스 데이터 포인트들을 기반으로 인위적으로 새로운 데이터를 생성하여 데이터 균형을 맞춤.

```
from imblearn.over_sampling import SMOTE

# SMOTE 적용하여 오버샘플링
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# RandomForest 모델 학습
rf_model_smote = RandomForestClassifier(n_estimators=200, max_depth=
                                        min_samples_leaf=4, bootstra

rf_model_smote.fit(X_train_smote, y_train_smote)

# 검증 세트 평가
y_val_pred_smote = rf_model_smote.predict(X_val)

# 검증 세트 결과
print(f"Validation Accuracy (SMOTE): {accuracy_score(y_val, y_val_pr
print("Confusion Matrix (SMOTE):")
print(confusion_matrix(y_val, y_val_pred_smote))
print("Classification Report (SMOTE):")
print(classification_report(y_val, y_val_pred_smote))
```

```
Validation Accuracy (SMOTE): 0.85360524399126
Confusion Matrix (SMOTE):
[[6512  798]
 [ 408  520]]
Classification Report (SMOTE):
              precision    recall  f1-score   support

           0       0.94      0.89      0.92      7310
           1       0.39      0.56      0.46       928

    accuracy                           0.85      8238
   macro avg       0.67      0.73      0.69      8238
weighted avg       0.88      0.85      0.86      8238
```

3. 언더샘플링 적용하기

```
from imblearn.under_sampling import RandomUnderSampler

# 언더샘플링 적용
rus = RandomUnderSampler(random_state=42)
X_train_under, y_train_under = rus.fit_resample(X_train, y_train)

# RandomForest 모델 학습
rf_model_under = RandomForestClassifier(n_estimators=200, max_depth=
                                        min_samples_leaf=4, bootstra

rf_model_under.fit(X_train_under, y_train_under)

# 검증 세트 평가
y_val_pred_under = rf_model_under.predict(X_val)

# 검증 세트 결과
print(f"Validation Accuracy (Under-sampling): {accuracy_score(y_val,
print("Confusion Matrix (Under-sampling):")
print(confusion_matrix(y_val, y_val_pred_under))
print("Classification Report (Under-sampling):")
print(classification_report(y_val, y_val_pred_under))
```

```
Validation Accuracy (Under-sampling): 0.8477785870356883
Confusion Matrix (Under-sampling):
[[6409  901]
 [ 353  575]]
Classification Report (Under-sampling):
              precision    recall  f1-score   support

           0       0.95      0.88      0.91      7310
           1       0.39      0.62      0.48       928

    accuracy                           0.85      8238
   macro avg       0.67      0.75      0.69      8238
weighted avg       0.88      0.85      0.86      8238
```

4. 임계값(Threshold) 조정

기본적으로 분류 모델은 예측값이 0.5보다 크면 'yes', 그렇지 않으면 'no'로 예측합니다. 그러나 이 임계값을 변경하여 'yes' 클래스에 더 높은 민감도를 부여할 수 있습니다.

```
# 예측값의 확률을 반환하는 predict_proba 사용
y_val_probs = rf_model.predict_proba(X_val)[:, 1]

# 새로운 threshold 설정 (예: 0.4로 낮추기)
new_threshold = 0.4
y_val_pred_new_threshold = (y_val_probs >= new_threshold).astype(int

# 검증 세트 결과
print(f"Validation Accuracy (New Threshold): {accuracy_score(y_val, 
print("Confusion Matrix (New Threshold):")
print(confusion_matrix(y_val, y_val_pred_new_threshold))
print("Classification Report (New Threshold):")
print(classification_report(y_val, y_val_pred_new_threshold))
```

```
Validation Accuracy (New Threshold): 0.8964554503520272
Confusion Matrix (New Threshold):
[[7061  249]
 [ 604  324]]
Classification Report (New Threshold):
              precision    recall  f1-score   support

           0       0.92      0.97      0.94      7310
           1       0.57      0.35      0.43       928

    accuracy                           0.90      8238
   macro avg       0.74      0.66      0.69      8238
weighted avg       0.88      0.90      0.89      8238
```

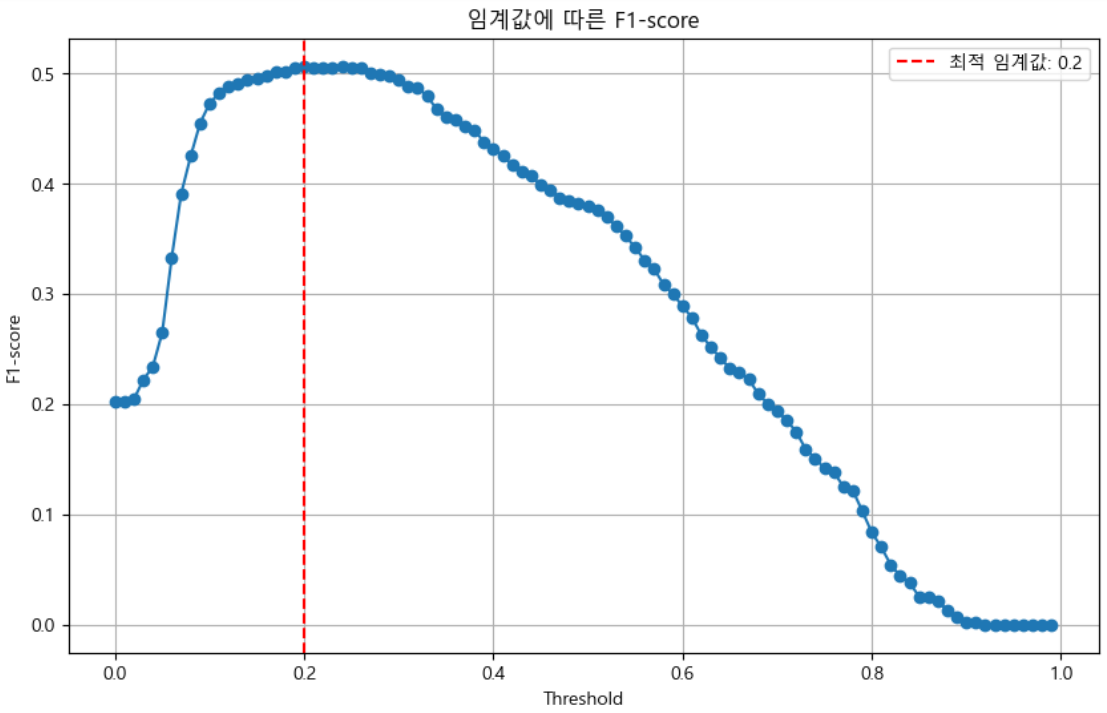
4.임계값을 재조정해 F1 - score 최적화 시도

#Precision-Recall Curve 계산 + 그리드 서치의 최적화 하이퍼파라미터 적용

```
from sklearn.metrics import f1_score
import numpy as np

# 예측 확률 얻기
y_val_probs = rf_model.predict_proba(X_val)[:, 1]  # 'yes' 클래스의 확

# 임계값 범위 설정
thresholds = np.arange(0.0, 1.0, 0.01)

# 각 임계값에 대해 F1-score 계산
f1_scores = []
for threshold in thresholds:
    y_val_pred_threshold = (y_val_probs >= threshold).astype(int)
    f1 = f1_score(y_val, y_val_pred_threshold)
    f1_scores.append(f1)

# F1-score가 최대가 되는 임계값 찾기
best_threshold = thresholds[np.argmax(f1_scores)]
best_f1_score = max(f1_scores)

print(f"최적의 임계값: {best_threshold}")
print(f"최고 F1-score: {best_f1_score}")
```

```
최적의 임계값: 0.2
최고 F1-score: 0.5062937062937063
```

```
import matplotlib.pyplot as plt

# 임계값에 따른 F1-score 그래프 그리기
plt.figure(figsize=(10, 6))
plt.plot(thresholds, f1_scores, marker='o')
plt.axvline(x=best_threshold, color='r', linestyle='--', label=f'최적
plt.title('임계값에 따른 F1-score')
plt.xlabel('Threshold')
plt.ylabel('F1-score')
plt.legend()
plt.grid(True)
plt.show()
```




```
# 최적 임계값 적용
y_val_pred_best_threshold = (y_val_probs >= best_threshold).astype(i

# 최종 평가 결과
accuracy = accuracy_score(y_val, y_val_pred_best_threshold)
print(f"최적 임계값에서의 정확도: {accuracy}")
print("Confusion Matrix:")
print(confusion_matrix(y_val, y_val_pred_best_threshold))
print("Classification Report:")
print(classification_report(y_val, y_val_pred_best_threshold))
```

```
최적 임계값에서의 정확도: 0.8714493809176985
Confusion Matrix:
[[6636  674]
 [ 385  543]]
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93      7310
           1       0.45      0.59      0.51       928

    accuracy                           0.87      8238
```

```
   macro avg       0.70      0.75      0.72      8238
weighted avg       0.89      0.87      0.88      8238
```

LightGBM

```
# 필요한 라이브러리 불러오기
import pandas as pd
import lightgbm as lgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classi
from sklearn.preprocessing import LabelEncoder

# 데이터 불러오기
file_path = 'bank-additional-full.csv'  # 경로 앞에 '/' 추가
data = pd.read_csv(file_path, delimiter=';')


# 전처리
data = data.replace('unknown', pd.NA)  # 'unknown' 값을 결측치로 처리

# 범주형 변수 목록
categorical_cols = ['job', 'marital', 'education', 'default', 'housi
                    'contact', 'month', 'day_of_week', 'poutcome']

# LabelEncoder를 사용하여 각 범주형 변수를 숫자로 변환
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col].astype(str))  # Label enc
    label_encoders[col] = le

# 'pdays'에서 999를 -1로 처리
data['pdays'] = data['pdays'].replace(999, -1)

# 'y' 변수도 Label Encoding으로 변환 (yes -> 1, no -> 0)
data['y'] = LabelEncoder().fit_transform(data['y'].astype(str))

# 모델 학습을 위한 피처(X)와 타겟(y) 나누기
```

```
X = data.drop(columns=['y'])  # 모든 피처 사용
y = data['y']

# 데이터셋 분할 (학습: 60%, 검증: 20%, 테스트: 20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_

# LightGBM 모델 초기화
lgbm = lgb.LGBMClassifier(boosting_type='gbdt', objective='binary', r

# 하이퍼파라미터 그리드 설정
param_grid = {
    'num_leaves': [31, 63, 127],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 200, 500],
    'max_depth': [5, 10, 15],
    'min_child_samples': [20, 50, 100],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}

# GridSearchCV 설정 (교차 검증 사용)
grid_search = GridSearchCV(estimator=lgbm, param_grid=param_grid,
                           scoring='accuracy', cv=3, verbose=1, n_jo

# 그리드 서치 실행
grid_search.fit(X_train, y_train)

# 최적의 파라미터와 성능 출력
print(f'최적의 하이퍼파라미터: {grid_search.best_params_}')
print(f'최고 정확도: {grid_search.best_score_}')

# 최적 파라미터로 모델을 학습 후 테스트 세트 평가
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

# 결과 출력
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f'Accuracy: {accuracy}')
```

```
print(f'Confusion Matrix:\n {conf_matrix}')
print(f'Classification Report:\n {class_report}')
```

```
최적의 하이퍼파라미터: {'colsample_bytree': 0.6, 'learning_rate': 0.0
5, 'max_depth': 15, 'min_child_samples': 20, 'n_estimators': 100,
'num_leaves': 31, 'subsample': 0.6}
최고 정확도: 0.918460707551571
Accuracy: 0.9150279193979121
Confusion Matrix:
 [[7079  234]
 [ 466  459]]
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      7313
           1       0.66      0.50      0.57       925

    accuracy                           0.92      8238
   macro avg       0.80      0.73      0.76      8238
weighted avg       0.91      0.92      0.91      8238
```

1.
클래스 가중치 조정 (Scale Pos Weight)

```
# 클래스 불균형 처리 - 클래스 가중치 계산
scale_pos_weight = len(y_train[y_train == 0]) / len(y_train[y_train

# LightGBM 파라미터 설정
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'boosting': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
```

```
    'scale_pos_weight': scale_pos_weight  # 클래스 불균형을 처리
}
```

```
Confusion Matrix:
 [[6566  747]
 [ 151  774]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.90      0.94      7313
           1       0.51      0.84      0.63       925

    accuracy                           0.89      8238
   macro avg       0.74      0.87      0.78      8238
weighted avg       0.92      0.89      0.90      8238
```

2. SMOTE

```
from imblearn.over_sampling import SMOTE

# SMOTE를 사용하여 클래스 불균형 해결
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y

# LightGBM 파라미터 설정 (scale_pos_weight 불필요)
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'boosting': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9
}

# LightGBM 전용 Dataset으로 변환 (Train, Validation)
train_data = lgb.Dataset(X_train_resampled, label=y_train_resampled)
valid_data = lgb.Dataset(X_val, label=y_val, reference=train_data)

# 콜백을 사용한 early stopping 구현
callbacks = [lgb.early_stopping(stopping_rounds=10)]

# 모델 학습
```

```
model = lgb.train(params, train_data, valid_sets=[valid_data],
                  num_boost_round=100, callbacks=callbacks)

# 모델 평가 (테스트 세트에서 예측)
y_pred = model.predict(X_test)
y_pred_binary = [1 if x > 0.5 else 0 for x in y_pred]

# 결과 출력
accuracy = accuracy_score(y_test, y_pred_binary)
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n {confusion_matrix(y_test, y_pred_binary)
print(f'Classification Report:\n {classification_report(y_test, y_pr
```

```
Accuracy: 0.8949987861131342
Confusion Matrix:
 [[6651  662]
 [ 203  722]]
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94      7313
           1       0.52      0.78      0.63       925

    accuracy                           0.89      8238
   macro avg       0.75      0.85      0.78      8238
weighted avg       0.92      0.89      0.90      8238
```

3. RandomUnderSampler

```
from imblearn.under_sampling import RandomUnderSampler

# RandomUnderSampler를 사용하여 클래스 불균형 해결
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_t

# LightGBM 파라미터 설정 (scale_pos_weight 불필요)
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'boosting': 'gbdt',
```

```
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9
}

# LightGBM 전용 Dataset으로 변환 (Train, Validation)
train_data = lgb.Dataset(X_train_resampled, label=y_train_resampled)
valid_data = lgb.Dataset(X_val, label=y_val, reference=train_data)

# 콜백을 사용한 early stopping 구현
callbacks = [lgb.early_stopping(stopping_rounds=10)]

# 모델 학습
model = lgb.train(params, train_data, valid_sets=[valid_data],
                  num_boost_round=100, callbacks=callbacks)

# 모델 평가 (테스트 세트에서 예측)
y_pred = model.predict(X_test)
y_pred_binary = [1 if x > 0.5 else 0 for x in y_pred]

# 결과 출력
accuracy = accuracy_score(y_test, y_pred_binary)
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n {confusion_matrix(y_test, y_pred_binary)
print(f'Classification Report:\n {classification_report(y_test, y_pr
```

```
Accuracy: 0.8505705268268997
Confusion Matrix:
 [[6145 1168]
 [  63  862]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.84      0.91      7313
           1       0.42      0.93      0.58       925

    accuracy                           0.85      8238
   macro avg       0.71      0.89      0.75      8238
weighted avg       0.93      0.85      0.87      8238
```

Catboost

- duration컬럼 제거

```
bank.drop('duration', axis=1, inplace=True)
```

- 파생변수 생성 및 결측치 처리

```
bank['low_euri_age'] = np.where(((bank['age'] < 25) | (bank['age'] >
bank['low_emp_var_rate'] = np.where(bank['emp.var.rate'] <= -1.1, 'L
bank['default'] = bank['default'].replace('unknown', 'no')
bank['cellular'] = np.where(bank['contact'] == 'cellular', 'Y', 'N')
```

- object형 데이터들의 데이터 타입을 category형으로 변환

```
object_cols = x_train.select_dtypes(include='object')

# object type 컬럼들을 category type으로 변환
def object2category(df):
    # object형 > category형 변환
    for i in df.columns:
        if i in object_cols:
            df[[i]] = df[[i]].astype('category')
        elif df[i].dtype == 'object':
            df[[i]] = df[[i]].astype('category')
    return df

x_train = object2category(x_train)
x_val = object2category(x_val)
```

- optuna적용하여 하이퍼파라미터 튜닝

```
from catboost import CatBoostClassifier, Pool
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score
import numpy as np
import optuna

sampler = optuna.samplers.TPESampler(seed=SEED)

def objective(trial):
    # CatBoost 모델 정의
    params = {
```

```
        'learning_rate': trial.suggest_uniform('learning_rate', 0.01
        'depth': trial.suggest_int('depth', 4, 16),
        'l2_leaf_reg': trial.suggest_loguniform('l2_leaf_reg', 1e-8,
        'border_count': trial.suggest_int('border_count', 32, 255),
        'bagging_temperature': trial.suggest_uniform('bagging_temper
        'random_strength': trial.suggest_uniform('random_strength',
        'bootstrap_type': 'Bayesian',
        'cat_features': categorical_features,
        'random_seed': SEED,
        'verbose': 0,
        'early_stopping_rounds': 100,
    }
    # Stratified K-Fold 교차 검증 수행
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEE
    f1_scores = []
    for train_index, valid_index in skf.split(x_train, y_train):
        X_train, X_valid = x_train.iloc[train_index], x_train.iloc[v
        Y_train, Y_valid = y_train.iloc[train_index], y_train.iloc[v

        train_pool = Pool(X_train, label=Y_train, cat_features=categ
        valid_pool = Pool(X_valid, label=Y_valid, cat_features=categ

        model = CatBoostClassifier(**params)
        model.fit(train_pool, eval_set=valid_pool, verbose=0)

        y_pred = model.predict(X_valid)
        f1_scores.append(f1_score(Y_valid, y_pred))

    return np.mean(f1_scores)

# Optuna 최적화 수행
study = optuna.create_study(direction='maximize', sampler=sampler)
study.optimize(objective, n_trials=100)

# 최적 하이퍼파라미터 출력
print("Best trial:")
trial = study.best_trial
print("Value: {}".format(trial.value))
print("Params: ")
for key, value in trial.params.items():
    print("    {}: {}".format(key, value))
```

```
# optuna best_param
ctb_param = {#'iterations': 5809,
    'learning_rate': 0.02365809705003199,
    'depth': 7,
    'l2_leaf_reg': 1.3306505602046657,
    'border_count': 53,
    'bagging_temperature': 0.4849747775486627,
    'random_strength': 0.10713621832055503
    # 'scale_pos_weight': 4.161991815142886
    }
```

- class_weights를 부여하여 클래스 불균형 문제 해결

```
# 클래스 불균형 해결을 위한 class weight 부여
classes = np.unique(y_train)
weights = compute_class_weight(class_weight='balanced', classes=clas
class_weights = dict(zip(classes, weights))
```

- 모델 훈련 (K-fold 교차검증을 통해 훈련 결과, 성능이 더 떨어짐.)

```
ctb_model.fit(x_train,y_train)
```

- 성능 평가

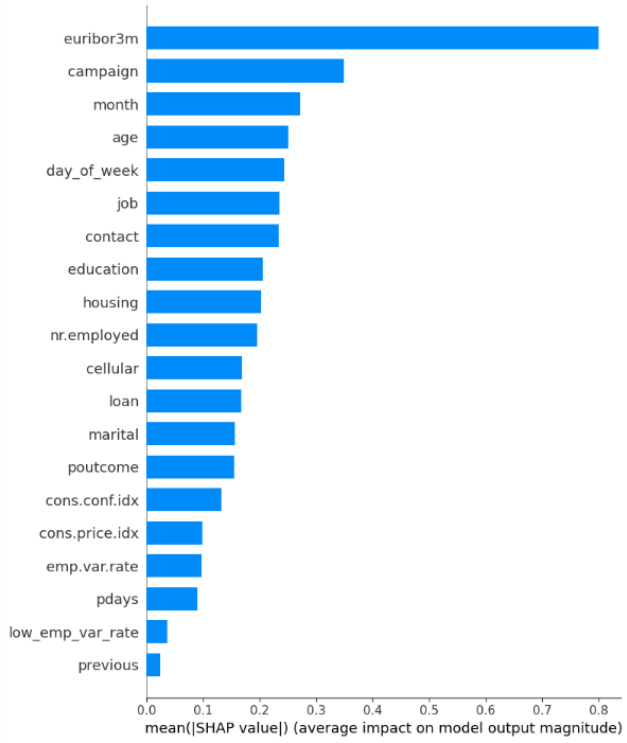
```
y_pred = ctb_model.predict(x_val)

# Catboost 모델 오차 행렬
y_val_pred = np.where(y_pred>=0.5, 1, 0)
get_clf_eval(y_val, y_val_pred)
```

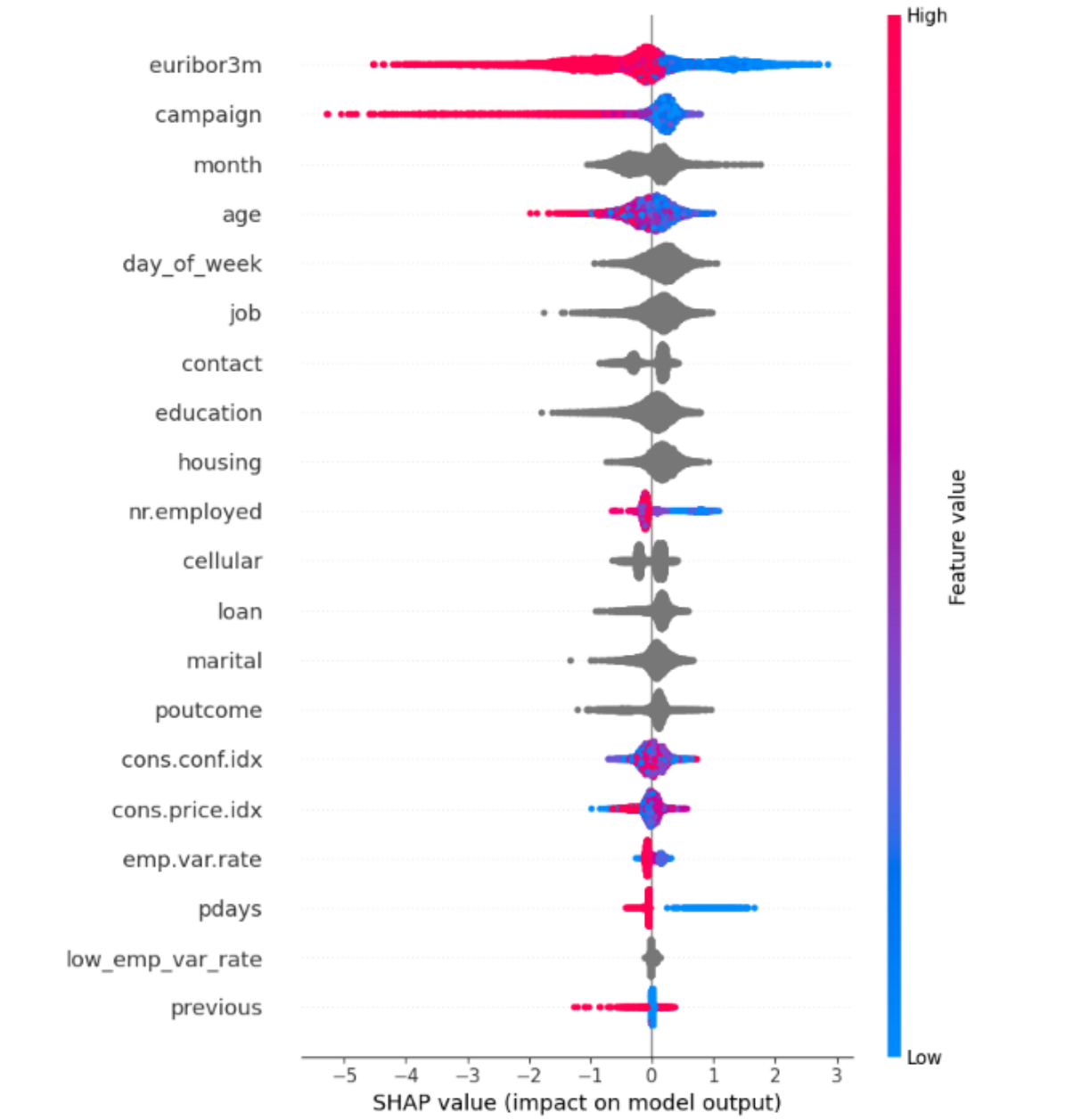
```
오차행렬:
 [[ 508  420]
 [ 561 6749]]

정확도: 0.8809
정밀도: 0.4752
재현율: 0.5474
F1: 0.5088
```

- 훈련데이터와 테스트데이터의 비율을 8:2로 나눴으며, 훈련결과 F1 score 0.5088이 현재 가장 높은 성능임.

- shapley value 실행

```
import shap

# SHAP Explainer 생성
explainer = shap.TreeExplainer(ctb_model)

# SHAP 값 계산 (테스트 데이터 사용)
shap_values = explainer.shap_values(x_val)

# SHAP Summary Plot (전체 피처의 중요도를 한 번에 시각화)
shap.summary_plot(shap_values, x_val, plot_type="bar")  # 막대형 그래프
shap.summary_plot(shap_values, x_val)  # 점 그래프 (상세한 시각화)

# SHAP Force Plot (특정 예측의 피처 영향 시각화)
# 첫 번째 샘플에 대한 Force Plot
shap.force_plot(explainer.expected_value, shap_values[0], x_val.iloc

# 피처에 따른 SHAP 값 분포 그래프 (개별 피처별로 확인)
shap.dependence_plot("특정 피처 이름", shap_values, x_val)
```





euribor3m컬럼에 대해서 Shapley value값이 가장 높게게 나왔으며 값이 음수일수록 정기예금 가입 예측에 좋은 영향을 미침.

또한 cons.price.idx, day_of_week, cellular, loan, month, contact, age, housing에서 정기예금 가입 예측에 긍정정인 영향을 끼쳤으며, euribor3m, campaign에서 부정적인 영향을 끼침.

5. 결론

5 - 1. 분석 결론

고객 데이터를 분석한 결과, 정기예금 가입률에 영향을 미치는 여러 중요한 요인들이 확인된다.

- 우선, **대학 학위를 보유한 고객들은 정기예금 가입률이 상대적으로 높다는 점**은 교육 수준이 금융 의사결정에 긍정적인 영향을 미친다는 것을 시사한다. 반면, **신용불량이 있는 고객들은 정기예금에 가입하지 않는 경향이 명확히 나타난다**. 이는 신용 상태가 정기예금 가입에 결정적인 영향을 미친다는 것을 보여준다.
- 또한, **개인 휴대전화로의 접촉이 이루어진 경우 정기예금 가입률이 높아지는 현상**은 개인화된 접근 방식이 고객의 결정을 유도하는 데 효과적임을 나타낸다. 이전 캠페인에서 성공적으로 가입한 고객들은 이번 캠페인에서도 정기예금에 재가입하는 경향이 높았으며, 이는 **고객의 이전 경험이 새로운 금융 상품 선택에 영향을 미친다는 사실을 강조한다**.
- 흥미롭게도, 금리가 높을 때와 낮을 때 주로 정기 예금 가입이 이루어지며, **금리가 낮을 때 정기예금에 가입한 고객들은 금리가 높을 때에 비교하여 20대 초반 또는 40대 이상의 연령층이 보였으며**, 이는 각 연령대의 재무적 필요와 투자 성향의 차이를 반영한다. 또한, **고객과의 연락 지속 시간이 길어질수록 정기예금 가입 고객 수가 증가하는 점**은 고객과의 신뢰 구축과 관계 유지를 통한 긍정적인 결과를 나타낸다.
- 마지막으로, 고용시장이 침체일 때 안정적인 투자를 선호하여 정기예금에 가입하는 경향이 있으며, 특히 고용 시장의 침체로 인해 회사 직원 수가 감소할수록 정기예금에 가입하는 비율이 높아진다.
- 종합적으로, 고객의 교육 수준, 신용 상태, 개인화된 접촉, 이전 경험, 금리, 연락 지속 시간, 고용 시장의 상황 등 다양한 요소들이 정기예금 가입률에 영향을 미친다는 것을 확인한다.

분석 결과를 바탕으로 정기 예금을 가입하는 집단 클러스터링 결과

- **고용 불안 및 금리 민감층**
  - 고용 시장이 불안정하거나, 금리가 낮을 때에 가입한 고객.
  - 금리가 낮을 때, 고용시장이 불안정할 때, 안정적인 투자를 위해 정기 예금에 가입하는 경향이 있는 고객들이다.
- **과거 경험이 중요한 충성 고객**
  - 과거에 성공적으로 정기예금을 가입한 이력이 있는 고객.
  - 이 집단은 과거 경험을 토대로 신뢰를 중요시하며, **재가입** 가능성이 높아 지속적인 관계 유지가 중요함.
- **높은 학력 및 안정 금융 상태의 고객**
  - 대학 학위를 보유하고 있으며, 신용 상태가 좋은 고객.

- 안정적인 재정 기반과 높은 교육 수준은 정기 예금 가입에서 신중하며, 위험을 회피하고, 안정적인 투자를 위해 정기 예금을 가입할 확률이 큼.

5 - 2. 모델 성능 종합

	XGBoost	Random Forest	LightGBM	CatBoost
Accuracy	0.8718	0.8714	0.8506	0.8809
Precision	0.4673	0.4462	0.4246	0.4752
Recall	0.4974	0.5851	0.9319	0.5474
F1 Score	0.4818	0.5063	0.5834	0.5088

5 - 3. 모델 해석

추가로 이번 분석에서는 고객의 정기예금 가입 예측을 위해 다양한 모델(Random Forest, XGBoost, CatBoost, LightGBM)을 비교하여 최적의 성능을 도출하고자 했다. 각 모델의 성능을 평가한 결과, 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1-score와 같은 다양한 지표가 도출되었고, 이 지표들을 기반으로 결론을 도출했다.

이 모델들을 분석하여 비즈니스 결정에 어떠한 방식으로 적용할 것인지 분석해야한다. 때문에, 수익 최대화 측면과 손실 최소화 측면에서 모두 고려하여, 현재 비즈니스가 가용가능한 전략이 무었인지, 상황에 맞게 개별 적용 할 필요가 있다. 다음은 지표들을 해석 하는 방법이다.

- **미탐(False Negative)**:
  정의: 실제로 고객이 정기예금에 가입할 의사가 있으나, 모델이 이를 '가입하지 않을 것'으로 잘못 예측하는 경우이다.

  결과: 미탐이 발생하면 실제로 정기예금에 가입할 가능성이 있는 고객을 놓치게 되어, 은행은 잠재적인 수익 기회를 상실하게 된다.

  중요성: 마케팅 캠페인의 주요 목표가 고객 확보이며, 고객당 예금의 가치가 높다면 미탐은 매우 중요한 문제로 대두된다. 은행이 더 많은 고객을 유치하려 한다면, 정기예금에 가입할 가능성이 있는 고객을 놓치지 않는 것이 중요하다.

- **오탐(False Positive)**:
  정의: 실제로 고객이 정기예금에 가입할 의사가 없음에도 불구하고, 모델이 이를 '가입할 것'으로 잘못 예측하는 경우이다.

  오탐이 발생하면 마케팅 팀은 가입 의사가 없는 고객에게 리소스를 낭비하게 된다. 예를 들어, 전화나 이메일과 같은 후속 마케팅에 비용이 발생하게 된다.

  리소스가 한정되어 있거나 마케팅 비용이 중요한 경우, 오탐을 줄이는 것이 필요하다. 그러나 마케팅 비용이 적거나 추가적인 마케팅 시도가 큰 비용을 수반하지 않는다면, 오탐의 중요성은 상대적으로 낮아질 수 있다.

은행이 정기예금 가입 고객을 최대한 확보하는 것이 최우선 목표라면, 미탐을 줄이고 재현율(Recall)을 높이는 것이 중요하다. 정기예금에 가입할 가능성이 있는 고객을 놓치지 않도록 하는 것이 핵심이다.
마케팅 비용이 제한적이거나 리소스 관리가 중요한 상황에서는, 오탐을 줄이고 정밀도(Precision)를 높이는 것이 중요하다. 실제로 가입할 가능성이 없는 고객에게 마케팅 리소스를 낭비하지 않도록 하는 것이 핵심이다.

모델별 성능을 비교하면, **CatBoost**가 F1-score(0.5088)와 Accuracy(0.8809)에서 가장 우수한 성능을 보였다. 특히 CatBoost는 **재현율(Recall)**이 0.5474로, 예금에 가입할 가능성이 있는 고객을 비교적 잘 예측하면서도, **정밀도(Precision)**가 0.4752로 무작정 많은 고객을 대상으로 하지 않고 정확하게 예측한 편이었다. 따라서 **CatBoost**는 미탐과 오탐 사이에서 적절한 균형을 유지한 모델로 볼 수 있다.
반면,
LightGBM은 매우 높은 재현율(0.9319)을 보였지만, 정밀도(0.4246)가 낮았다. 즉, 많은 잠재 고객을 발견하지만 실제로는 가입 의사가 없는 고객도 상당수 포함하고 있었다. 이는 마케팅 리소스를 많이 사용하는 상황에서 적합하지 않을 수 있지만, 고객 확보가 최우선인 상황에서는 적합한 모델이다.

Random Forest와 **XGBoost**는 상대적으로 균형 잡힌 성능을 보였다. **Random Forest**는 Accuracy가 0.8714, F1-score가 0.5063으로 평균적인 성능을 보였으며, **XGBoost**는 비슷한 정확도(0.8718)와 정밀도(0.4673)를 나타냈다. 두 모델 모두 비교적 안정적이었으나, 최적화가 더 필요해 보였다.

이러한 결과를 종합해보면, **회사의 전략이 무엇을 우선시 하는지**에 따라 모델 선택이 달라질 수 있다. 고객 확보가 우선이고 미탐(False Negative)을 줄이는 것이 중요하다면, 높은 재현율을 가진 **LightGBM**이 적합하다. 반면, 마케팅 리소스가 한정되어 있고, 불필요한 고객에게 리소스를 낭비하지 않으려면, **CatBoost**나 **XGBoost** 같은 모델이 유리하다. **CatBoost**는 두 지표에서 고루 좋은 성능을 보였으므로, 다양한 시나리오에서 균형 잡힌 성능을 제공할 수 있다. 균형의 상태를 추구한다면 , **F1-score를 최대화**하여 두 가지(미탐과 오탐) 사이에서 적절한 균형을 잡는 것이 현재로서 최적의 방안으로 보인다.

5 - 4. 분석과 모델을 종합한 결론

분석을 통해 우리는 대학 학위 , 마케터의 전화접촉, 현 금리 등 다양한 요소가 정기예금 가입에 영향을 미치는 것을 알았다. 또한 이러한 요인들은 그룹화가 가능하다는 사실 또한 알게되었다. 때문에, 고객의 특성에 맞춘 개별적인 금융 상품을 추천하고, 특히 휴대전화를 통한 맞춤형 접촉 방식과 장기적인 고객 관리를 강화해야 한다. 이를 통해 더 효과적으로 고객과 소통하며, 고객의 요구에 맞춘 서비스를 제공할 수 있다. 또한, 데이터 기반 의사결정을 통해 경제 지표와 고객의 개인적 상황을 반영한 마케팅 전략을 지속적으로 개선해야 한다.

모델 활용 측면에서는 최적화된 머신러닝 모델을 적용하여 높은 재현율을 유지하는 동시에 불필요한 비용을 줄일 수 있는 균형 잡힌 마케팅 캠페인을 실행해야 한다. 특히 F1-Score가 높은 CatBoost 모델을 활용하면 더 높은 성과를 기대할 수 있다. 이를 바탕으로 은행은 고객 맞춤형 전략을 통해 마케팅 효율성을 극대화하고, 더 많은 고객을 유치할 수 있는 확실한 근거를 확보할 수 있다. 이와 같은 데이터 활용을 기반으로 한 의사결정은 향후 캠페인의 성공에 있어 중요한 역할을 할 것이다.

5 - 5. 활용 가능한 마케팅 전략

고학력 고객 타겟팅 전략

- **대학 학위를 보유한 고객들이 정기예금 가입률이 높다는 점**을 활용하여, 고학력 고객을 대상으로 한 맞춤형 금융 상품을 제안한다. 특히, 고수익 또는 복잡한 금융 상품을 소개하고, 고학력 고객층이 선호할 만한 재무 관련 컨텐츠와 혜택을 제공하는 것이 효과적일 것이다.

신용 불량 고객을 위한 재무 회복 프로그램

- **신용 불량 고객들이 정기예금에 가입하지 않는 경향**을 보이므로, 이들을 대상으로 한 **신용 회복 프로그램**을 먼저 제공한다. 이를 통해 고객의 신용 상태가 개선되면, 후속 마케팅을 통해 정기예금 상품을 소개하고 가입을 유도한다.

개인화된 휴대전화 마케팅 강화

- **휴대전화로 접촉한 경우 정기예금 가입률이 높아진다는 점**을 활용해, **SMS 및 개인화된 푸시 알림**을 통해 고객 맞춤형 금융 상품을 제안한다. 개인의 금융 상황에 맞는 혜택을 강조한 메시지와 상담을 제공하여 정기예금 가입을 유도한다.
- **고객의 이전 금융 거래 데이터를 바탕으로 맞춤형 메시지**를 전송하고, **빠른 응답이 가능한 1:1 전화 상담**을 통해 신속한 가입을 촉진한다.

연령대 맞춤 상품 개발

- **금리가 낮을 때에도 20대 초반과 40대 이상의 연령층에서 정기예금 가입률이 높다는 점**을 반영해, **연령대별 맞춤 금융 상품**을 개발한다. 예를 들어, 20대 초반에게는 미래 자산 증식을 목표로 한 장기 금융 상품을 제공하고, 40대 이상에게는 안정적인 투자 수단으로서의 정기예금을 홍보한다.
- **연령대별로 적합한 금리 제안**을 통해, 각 고객층의 재무적 필요에 맞는 혜택을 제공하고 장기적인 자산 증식의 기회를 강조한다.

고용 시장 상황에 따른 마케팅 전략

- **고용시장이 불활발할 때 고객들이 정기예금 가입을 선호한다는 점**을 고려해, **경제 불확실성을 강조하는 캠페인**을 운영한다. 경기 침체기에는 안전한 자산 증식을 강조하며, 불확실성 속에서 안정성을 제공하는 금융 상품으로 정기예금을 제안한다.
- **고용시장이 불안정할 때 집중적인 마케팅**을 통해 고객에게 안정적인 자산 관리 방안을 제공하고, **고용 시장 회복기에 맞춘 재투자 상품**도 함께 제시한다.

연락 지속 시간 최적화

- **고객과의 연락 지속 시간이 길어질수록 정기예금 가입률이 높아진다는 점**을 활용해, 고객과의 대화 및 상담 시간을 늘리고 **장기적인 관계를 구축**하는 전략을 채택한다. 특히, 고객의 관심사와 재무적 목표를 파악한 후 맞춤형 상담을 제공함으로써 고객 신뢰도를 높인다.
- **CRM(Customer Relationship Management)** 시스템을 통해 고객과의 상호작용 기록을 관리하고, 이를 바탕으로 고객의 선호도에 맞춘 추가 상담과 금융 상품을 제안한다.

금리 변동에 따른 마케팅 메시지 차별화

- **금리가 높을 때뿐만 아니라 낮을 때도 정기예금에 가입하는 고객이 존재**하는 점을 활용해, **금리 변동에 따른 맞춤형 마케팅**을 진행한다. 금리가 높을 때는 더 높은 수익성을 강조하며, 금리가 낮을 때는 안전 자산으로서의 정기예금의 중요성을 부각시킨다.
- **금리 변동에 민감한 고객층**에게는 실시간으로 금리 정보를 제공하고, 최적의 타이밍에 정기예금 상품을 제안하는 방식으로 마케팅을 차별화한다.

고객 클러스터 별 제안 가능한 마케팅

앞에서 분류한 고객 계층에 대해 어떤 마케팅 전략을 사용할 수 있는지 제안할 수 있다.

1. **고용 불안 및 금리 민감층**

   앞에서 고객들은 고용시장이 불안할 때에 정기 예금에 활발하게 가입하는 경향이 있었으며, 20대 초반과 40대 이상의 연령층에서 금리가 낮을 때 정기 예금 가입률이 높은 것을 볼 수 있었다. 이러한 고객들의 특징을 살려 이 클러스터의 고객들에게는 연령대별 맞춤 금융 상품을 개발하고 경기 침체기에 정기 예금의 안전 자산으로서의 장점을 부각시켜 고객들로 하여금 정기 예금에 가입할 수 있도록 유도하는 전략을 사용할 수 있다.

2. **과거 경험이 중요한 충성 고객**

   이 분류의 고객들은 과거 경험을 토대로 신뢰를 중요시하며 재가입 가능성이 높아 지속적인 신뢰 관계의 구축이 필요하다. 또한 앞에서 고객과의 연락 지속시간이 길어지고 휴대전화를 사용하여 연락을 시도했을 때 정기 예금 가입률이 높아지는 경향을 보이는 것을 확인했다. 이러한 특징을 살려 고객과 연락을 할 때 SMS나 개인화된 푸시 알람을 통해 고객 개인에 맞는 혜택을 강조한 메시지를 전송하고 이러한 기록들을 CRM을 사용하여 상호작용 기록을 관리하고 이를 활용하여 고객의 선호도에 맞춘 추가 상담과 금융 상품의 제안을 하는 방법으로 접근하는 전략을 사용할 수 있다.

3. **높은 학력 및 안정 금융 상태의 고객**

   앞에서 고객들은 학력이 높을 수록 정기 예금 가입률이 높았고, 금융 상태가 안정됐을 때 정기 예금의 가입률이 높은 것을 볼 수 있었다. 이러한 고객들의 특징을 살려 이 분류의 고객들에게는 우선 신용 불량 고객들에 대한 신용 회복 프로그램을 지원해주고, 이를 통해 고객들의 신용 상태가 개선이 되면 후속 마케팅으로 정기 예금에 대한 홍보를 하여 가입을 유도하는 전략을 사용할 수 있다. 또한 고학력자 고객에게는 복잡하거나 고수익인 금융 상품을 추천하거나 선호할 수 있는 재무 관련 콘텐츠 등을 휴대전화를 통해 전송하여 가입을 유도하는 전략을 사용할 수 있다.

- 배경 및 분석 목표

  고객의 특성 및 경제적 요인들이 정기예금 가입 의사에 미치는 영향을 탐구

- EDA

  EDA 분석 결과를 바탕으로 정기예금 가입에 영향을 미치는 직업군, 대출, 결혼 여부, 통화 지속시간, 연령, 고용 시장, 금리 등의 요인에 대해 가설을 도출

- 가설검정

  가설을 바탕으로 t-test, 카이제곱 검정 및 심층적인 분석을 통한 검정

- 머신러닝 모델 학습

  고객의 정기예금 가입 여부를 예측하는 모델 생성

- 결론

  고객을 그룹화하여 다양한 요소를 반영해 맞춤형 전략을 제시