



은행의 마케팅 데이터 분석

* 상태	완료
▣ 마감일	@10/28/2025
▣ 작업 유형	스프린트 미션
☰ 설명	스프린트 미션 #4
⌚ 업데이트 시간	@2025년 10월 27일 오후 7:50

스프린트 미션 4

은행의 마케팅 데이터 분석을 통한 정기 예금 가입 가능성 예측

- 이번 미션에서는 포르투갈 은행의 마케팅 데이터를 분석해볼 예정입니다. 이 실습에서는 결정 트리와 앙상블 기법을 사용하여 분류 모델을 구축하고, 마케팅 캠페인의 효율성을 높이는 전략을 도출해 보겠습니다.
- 사용 데이터셋: [bank-additional-full.csv](#)
- 해당 데이터는 UC Irvine Machine Learning Repository에서 제공하는 **Bank Marketing(링크)** 데이터입니다.

데이터 출처

Moro, S., Cortez, P., & Rita, P. (2014). A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, In press, <http://dx.doi.org/10.1016/j.dss.2014.03.001>

분석 목표

- 마케팅 데이터를 활용하여 고객의 직업군, 교육 수준, 나이, 과거 마케팅 캠페인의 성공여부 등의 요인에 따른 정기 예금 가입 가능성을 예측합니다.
- 예측한 결과를 바탕으로 한 마케팅 계획 수립하여 비즈니스 전략을 제시합니다.

데이터 확인 및 정리

```
df.info() # 데이터 확인  
df.shape # 데이터 모양 확인  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 41188 entries, 0 to 41187  
Data columns (total 1 columns):  
 # Column  
Non-Null Count Dtype  
---  
O age;"job";"marital";"education";"default";"housing";"loan";"contact";"month";"day_of_week";"duration";"camp  
aign";"pdays";"previous";"poutcome";"emp.var.rate";"cons.price.idx";"cons.conf.idx";"euribor3m";"nr.employe  
d";"y" 41188 non-null object  
dtypes: object(1)
```

```
memory usage: 321.9+ KB  
(41188, 1)
```

```
df = pd.read_csv('data/bank-additional-full.csv', sep=";")  
df = df.apply(lambda s: s.astype(str).str.strip('')) # 따옴표 제거  
  
df.info  
df.head()  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 41188 entries, 0 to 41187  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype  
---  
 0 age      41188 non-null object  
 1 job      41188 non-null object  
 2 marital   41188 non-null object  
 3 education 41188 non-null object  
 4 default   41188 non-null object  
 5 housing   41188 non-null object  
 6 loan      41188 non-null object  
 7 contact   41188 non-null object  
 8 month     41188 non-null object  
 9 day_of_week 41188 non-null object  
 10 duration  41188 non-null object  
 11 campaign  41188 non-null object  
 12 pdays    41188 non-null object  
 13 previous  41188 non-null object  
 14 poutcome  41188 non-null object  
 15 emp.var.rate 41188 non-null object  
 16 cons.price.idx 41188 non-null object  
 17 cons.conf.idx 41188 non-null object  
 18 euribor3m   41188 non-null object  
 19 nr.employed 41188 non-null object  
 20 y         41188 non-null object  
dtypes: object(21)  
memory usage: 6.6+ MB
```

▼ df.head()

	age	job	marital	education	default	housing	loan
0	56	housemaid	married	basic.4y	no	no	no
1	57	services	married	high.school	unknown	no	no
2	37	services	married	high.school	no	yes	no
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

▼ 데이터 개요

컬럼명	설명
age	나이 (숫자)
job	직업 (범주형)
marital	결혼 여부 (범주형)
education	교육 수준 (범주형)

컬럼명	설명
default	신용 불량 여부 (범주형)
housing	주택 대출 여부 (범주형)
loan	개인 대출 여부 (범주형)
contact	연락 유형 (범주형)
month	마지막 연락 월 (범주형)
day_of_week	마지막 연락 요일 (범주형)
duration	마지막 연락 지속 시간, 초 단위 (숫자)
campaign	캠페인 동안 연락 횟수 (숫자)
pdays	이전 캠페인 후 지난 일수 (숫자)
previous	이전 캠페인 동안 연락 횟수 (숫자)
poutcome	이전 캠페인의 결과 (범주형)
emp.var.rate	고용 변동률 (숫자)
cons.price.idx	소비자 물가지수 (숫자)
cons.conf.idx	소비자 신뢰지수 (숫자)
euribor3m	3개월 유리보 금리 (숫자)
nr.employed	고용자 수 (숫자)
y	정기 예금 가입 여부 ('yes' 또는 'no')

```
# 숫자형으로 변환할 컬럼들
num_cols = ["age", "duration", "campaign", "pdays", "previous", "emp.var.rate",
            "cons.price.idx", "cons.conf.idx", "euribor3m", "nr.employed"]

for col in num_cols:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce") # 숫자로 변환, 변환 불가는 NaN

# 정기 예금 가입 여부. yes = 1, no = 0 으로 변환
df["y"] = df["y"].map({"no": 0, "yes": 1})
```

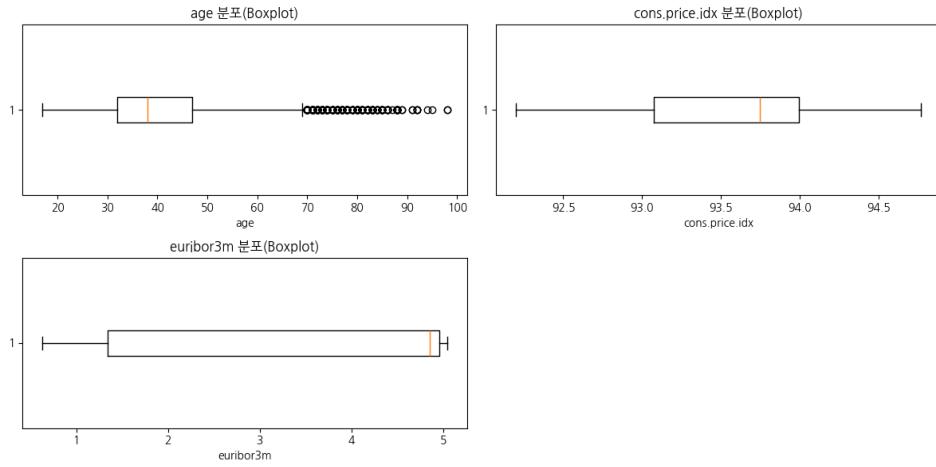
- 결측치 및 이상치 확인

```
df.isnull().sum() # 결측치 없음

num_cols = ["age", "cons.price.idx", "euribor3m"]
plt.figure(figsize=(12, 6))

for i, col in enumerate(num_cols, 1):
    plt.subplot(2, 2, i)
    plt.boxplot(df[col], vert=False)
    plt.title(f"{col} 분포(Boxplot)")
    plt.xlabel(col)

plt.tight_layout()
plt.show()
```



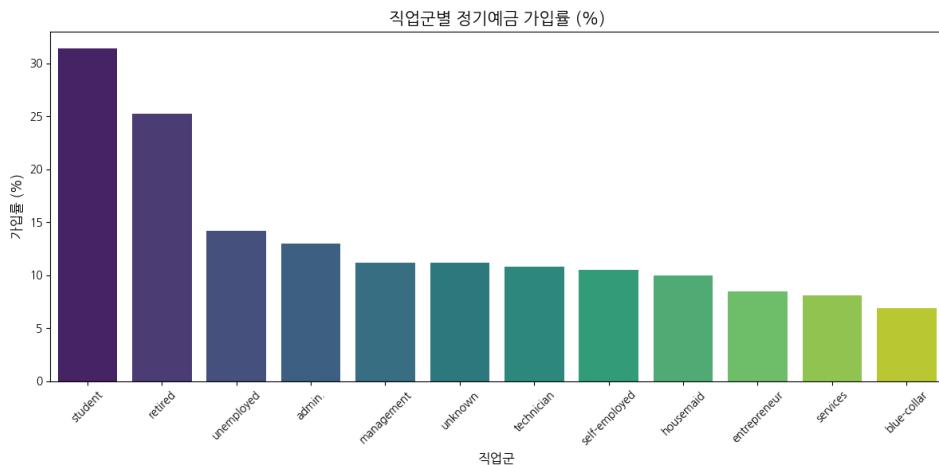
가입 여부에 영향을 미치는 주요인으로 생각되는 컬럼들에 대한 이상치 탐색.

나이(age)에 이상치 확인하였으나 수용할 수 있는 범위의 데이터이므로 데이터유지 결정.

EDA

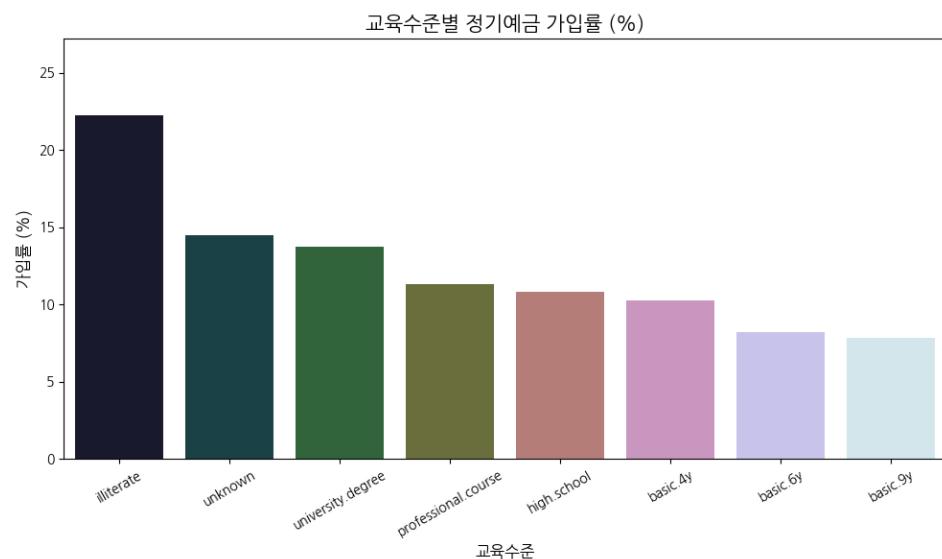
- 정기 예금 가입률에 영향을 미칠 것으로 판단되는 요인들을 대상으로 데이터 분석

1. 직업군 별 정기 예금 가입률



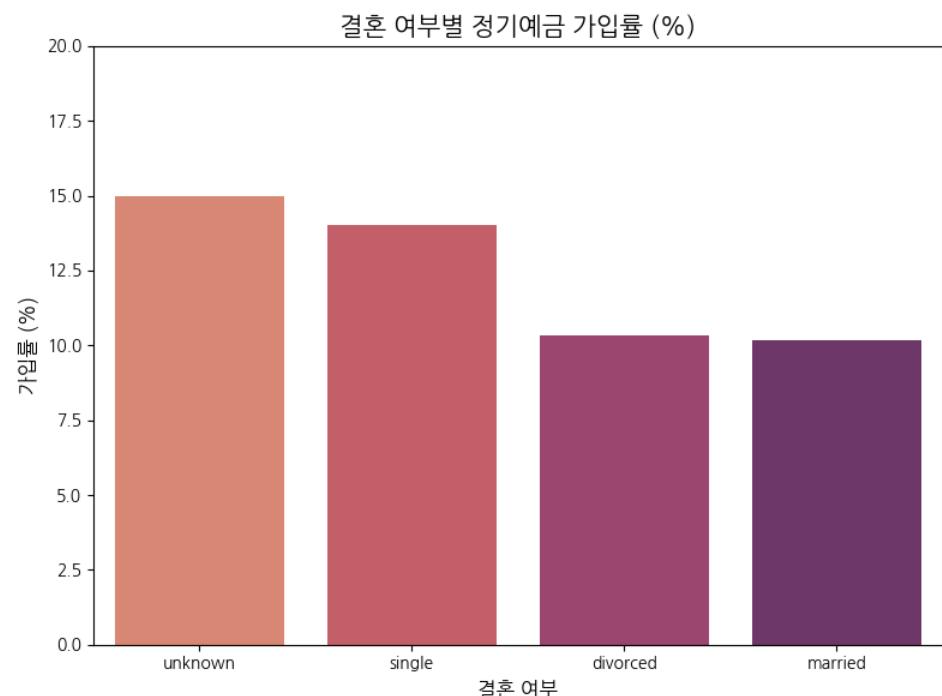
- 학생 혹은 은퇴한 고객에게서 정기 예금 가입률이 가장 높게 나타남.
- 서비스업 종사자, 육체 노동자가 정기 예금 가입률은 가장 낮게 나타나며 그 외 기업인과 주부, 자영업자 순으로 가입률이 낮게 나타나는 것을 확인.
- 특정 직업군에 정기 예금 가입률은 유의미한 차이를 나타냄.

2. 교육 수준 별 정기 예금 가입률

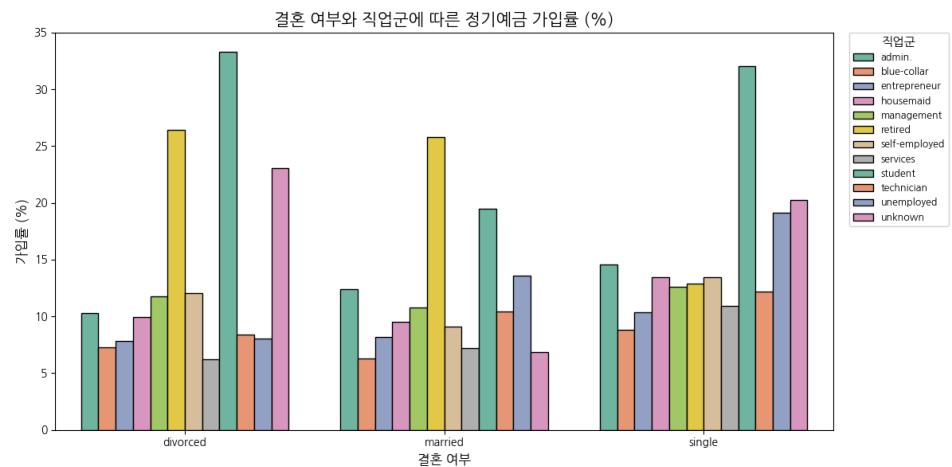


- 교육 수준별 정기 예금 가입률의 상관관계는 유의미 하지 않은 것으로 판단됨
- 초등교육 수료자와 고등교육 수료자와의 가입률의 차이는 크지 않으며 외려 문맹인 고객의 예금가입률이 높게 나타남.

3. 결혼 여부에 따른 정기 예금 가입률

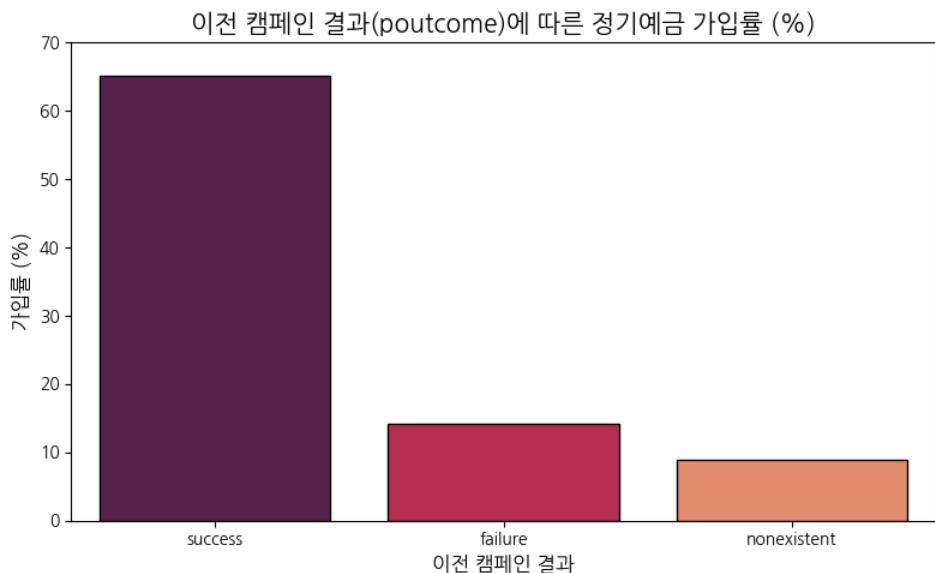


- 이혼, 기혼 고객보다 미혼의 고객에게서 정기 예금 가입률이 높게 나타남.



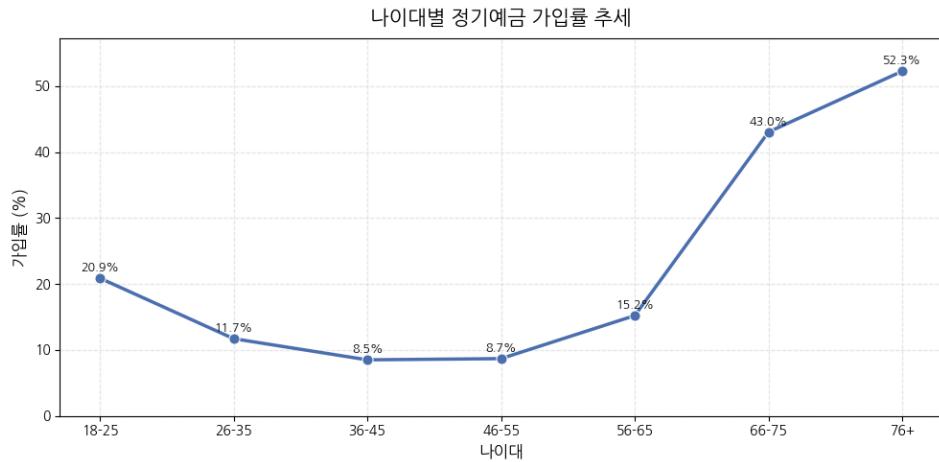
- 기혼자의 경우 은퇴하였거나 학생인 경우에 가입률이 높게 나타나며, 미혼자의 경우에는 학생인 경우 가입률이 높게 확인됨.
- 이혼자의 경우도 마찬가지로 학생이거나 은퇴한 경우 가입률이 높게 확인됨.

4. 과거 마케팅 캠페인 성공 여부에 따른 정기 예금 가입률



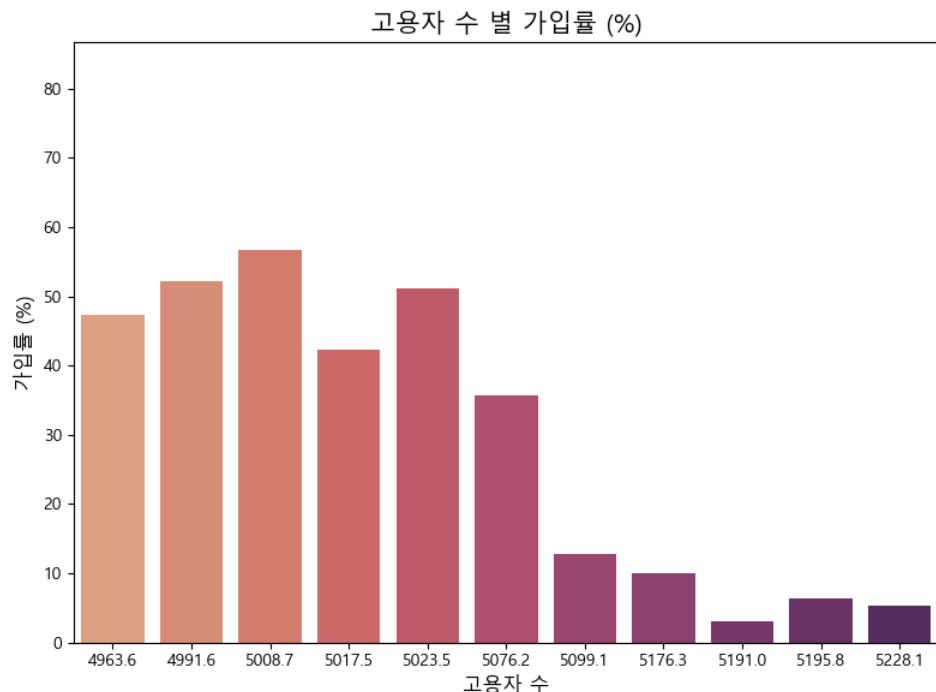
- 당연하게도 이전 마케팅 캠페인의 결과가 성공적으로 판단되는 고객들은 가입률이 매우 높게 나타남.

5. 나이대 별 정기 예금 가입률



- 25세 이하의 고객이나 56세 이상의 고객들에게서 가입률이 높게 확인됨.
- 직업군 별 가입률의 결과와 유의미한 상관관계를 보임
- 학생이 많은 25세 이하의 나이대나, 은퇴 이후 경제활동이 줄어든 고객들에게서 높은 가입률을 확인할 수 있음.

6. 경기 상황별 정기 예금 가입률



EDA를 바탕으로 정기 예금 가입 가능성 예측

데이터 전처리

```
X = df.drop("y",axis=1) # 종속변수 y 드랍
y = df["y"]
```

```
# 쓸모없는 컬럼 제거 (특정 X에서만)
```

```

drop_cols = ["duration", "contact", "month", "day_of_week", "pdays", "default"]
X = X.drop(columns=drop_cols, errors="ignore")

cat_cols = X.select_dtypes(include=["object", "category"]).columns

# 원-핫 인코더/컬럼트랜스포머 정의 (X만 변환)
ct = ColumnTransformer(
    transformers=[
        ("encoder", OneHotEncoder(handle_unknown="ignore", sparse_output=False), cat_cols)
    ],
    remainder="passthrough",
    verbose_feature_names_out=False
)

# 학습/테스트 분할 (먼저 나누고, 학습셋으로 fit, 테스트셋은 transform만!)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# 변환 적용
X_train = ct.fit_transform(X_train) # fit은 학습셋으로만
X_test = ct.transform(X_test) # 테스트셋에는 transform만

```

모델 별 학습 및 검증

```

# XGBoost
pos = y_train.sum()
neg = len(y_train) - pos
scale_pos_weight = neg / pos

xgb_model = XGBClassifier(
    n_estimators=500,
    learning_rate=0.05,
    max_depth=4,
    subsample=0.8,
    colsample_bytree=0.8,
    reg_lambda=1.0,
    random_state=42,
    n_jobs=-1,
    objective="binary:logistic",
    eval_metric="auc",
    scale_pos_weight=scale_pos_weight,
    tree_method="hist"
)
xgb_model.fit(X_train, y_train)

# RandomForest
rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)
rf_model.fit(X_train, y_train)

# LightGBM
lgb_model = LGBMClassifier(
    n_estimators=700,
    learning_rate=0.05,

```

```

        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42,
        n_jobs=-1,
        class_weight=None # 불균형 보정이 필요하면 "balanced" 고려
    )

lgb_model.fit(X_train, y_train)

# 예측
xgb_pred = xgb_model.predict(X_test)
rf_pred = rf_model.predict(X_test)
lgb_pred = lgb_model.predict(X_test)

# 확률 예측 (AUC 계산용)
xgb_proba = xgb_model.predict_proba(X_test)[:, 1]
rf_proba = rf_model.predict_proba(X_test)[:, 1]
lgb_proba = lgb_model.predict_proba(X_test)[:, 1]

# 성능 비교
print("==== XGBoost ===")
print("Accuracy :", accuracy_score(y_test, xgb_pred))
print("ROC-AUC :", roc_auc_score(y_test, xgb_proba))
print(classification_report(y_test, xgb_pred, digits=4))

print("\n==== RandomForest ===")
print("Accuracy :", accuracy_score(y_test, rf_pred))
print("ROC-AUC :", roc_auc_score(y_test, rf_proba))
print(classification_report(y_test, rf_pred, digits=4))

print("\n==== LightGBM ===")
print("Accuracy :", accuracy_score(y_test, lgb_pred))
print("ROC-AUC :", roc_auc_score(y_test, lgb_proba))
print(classification_report(y_test, lgb_pred, digits=4))

```

- XGBoost, RandomForest, LGBM 세가지 모델을 학습시켜 성능을 비교.

```

==== XGBoost ===
Accuracy : 0.8436513716921583
ROC-AUC : 0.8038333176093212
      precision  recall f1-score support
          0       0.9521   0.8674   0.9078    7310
          1       0.3859   0.6562   0.4860     928

      accuracy          0.8437   8238
   macro avg   0.6690   0.7618   0.6969    8238
weighted avg   0.8883   0.8437   0.8603    8238

```

```

==== RandomForest ===
Accuracy : 0.8936635105608157
ROC-AUC : 0.7800163038350866
      precision  recall f1-score support
          0       0.9135   0.9722   0.9419    7310

```

```

1 0.5568 0.2748 0.3680 928

accuracy          0.8937 8238
macro avg 0.7351 0.6235 0.6550 8238
weighted avg 0.8733 0.8937 0.8773 8238

==== LightGBM ====
Accuracy : 0.8975479485311969
ROC-AUC : 0.7967597970423133
precision recall f1-score support

0 0.9111 0.9802 0.9444 7310
1 0.6123 0.2468 0.3518 928

accuracy          0.8975 8238
macro avg 0.7617 0.6135 0.6481 8238
weighted avg 0.8775 0.8975 0.8776 8238

```

- XGBoost

장점: 재현율(0.65) 가장 높음 → 실제 가입자를 놓치지 않음

단점: 정밀도(0.38)가 낮음 → 가입 안 할 사람을 가입자로 착각함

활용 추천: “가입자를 최대한 많이 찾아야 하는” 상황 (예: 마케팅 캠페인)

“찾을 수 있는 한 많이 찾아라” 전략에 적합함.

- RandomForest

장점: 전체 정확도와 음성 클래스 예측은 뛰어남

단점: 가입자(1) 재현율이 0.27로 매우 낮음

활용 추천: “잘못된 예측을 최소화”해야 하는 상황 (보수적 분류)

즉, “가입하지 않을 사람은 거의 틀리지 않는다” 하지만 “가입할 사람은 대부분 놓친다”.

- LightGBM

장점: 정확도(0.898), 정밀도(0.61) 우수

단점: 재현율(0.25)이 낮음 → 실제 가입자 대부분을 놓침

활용 추천: “정확히 맞출 때만 행동하는 전략” (예: 고비용 프로모션 최소화)

“예측이 맞을 때만 연락해도 된다”는 전략이면 LightGBM이 유리합니다.

⇒ 가입자를 최대한 많이 확보할 수 있는 마케팅 캠페인에 적절한 모델은 XGBoost로 판단. 최종 예측 모델로 결정

- 예측 모델의 성능 향상을 위한 Threshold 조정.

```
# gpt에 정확도를 더 높일 수 있는 방안을 물어보니 Threshold 조정을 제안했다. 다른 모델은 아직 미정.
```

```

prec, rec, thr = precision_recall_curve(y_test, y_proba_tuned)
# precision_recall_curve가 반환하는 thresholds 길이는 n-1이므로 f1도 n-1에 맞춰 계산
f1 = 2 * prec[:-1] * rec[:-1] / (prec[:-1] + rec[:-1] + 1e-12)
best_idx = np.argmax(f1)
best_thr_f1 = thr[best_idx]

print(f"F1 최대 임계값: {best_thr_f1:.4f}, Precision={prec[best_idx]:.3f}, Recall={rec[best_idx]:.3f}, F1={f1[best_idx]:.3f}")

```

```

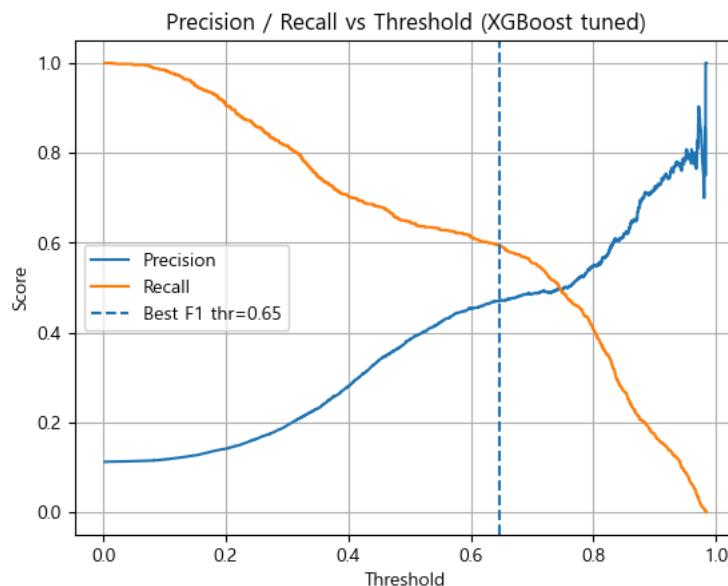
3f}")

# 적용
y_pred_best = (y_proba_tuned >= best_thr_f1).astype(int)
print(classification_report(y_test, y_pred_best, digits=4))

prec, rec, thr = precision_recall_curve(y_test, y_proba_tuned)

plt.plot(thr, prec[:-1], label="Precision")
plt.plot(thr, rec[:-1], label="Recall")
plt.axvline(best_thr_f1, linestyle="--", label=f"Best F1 thr={best_thr_f1:.2f}")
plt.xlabel("Threshold"); plt.ylabel("Score")
plt.title("Precision / Recall vs Threshold (XGBoost tuned)")
plt.legend(); plt.grid(True); plt.show()

```



- 정밀도(Precision)와 재현율(Recall)의 균형(F1-Score)이 가장 높은 지점인 곳의 Threshold 확인.

```

# threshold 조정 후 성능
y_pred_best = (y_proba_tuned >= best_thr_f1).astype(int)

print("\n==== 조정 후 성능 ===")
print(classification_report(y_test, y_pred_best, digits=4))

```

```

F1 최대 임계값: 0.6476, Precision=0.473, Recall=0.595, F1=0.527
precision    recall    f1-score   support
          0    0.9468    0.9159    0.9311     7310
          1    0.4730    0.5948    0.5270     928

accuracy              0.8797    8238
macro avg    0.7099    0.7553    0.7290     8238
weighted avg  0.8935    0.8797    0.8856     8238

```

- 최적화 완료 (사실 한계에 부딪힘)
- 예측에 영향을 크게 미친 변수들에 대해 알아보고 정기 예금 가입자 예측을 위한 인사이트를 도출해보자.

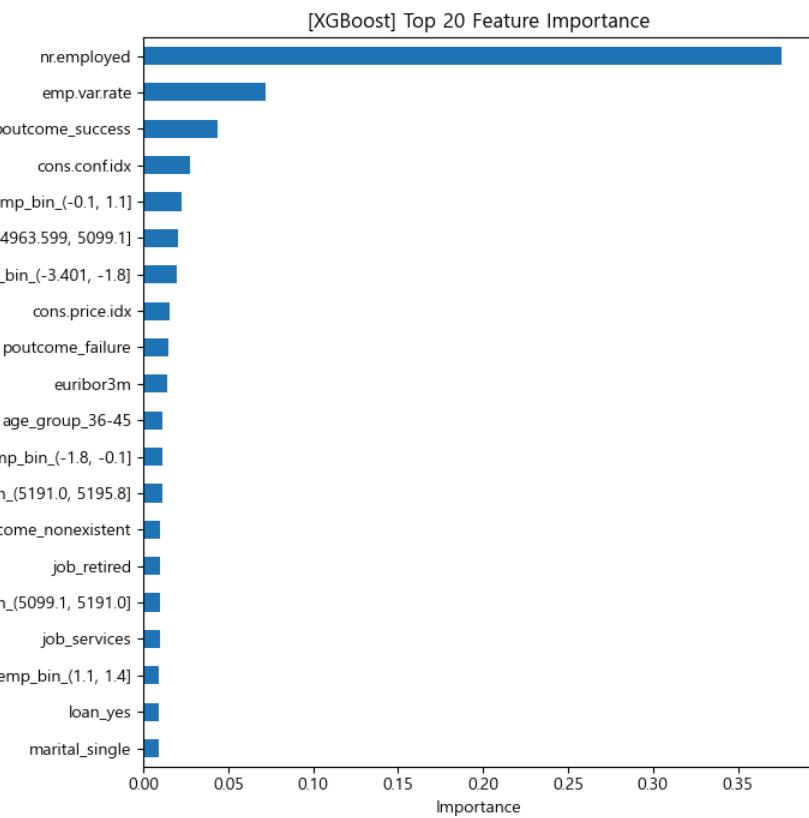
```

feature_names = ct.get_feature_names_out()

def plot_top_importances(model, feature_names, topn=20, title="Feature Importance (one-hot level)"):
    importances = model.feature_importances_
    s = pd.Series(importances, index=feature_names).sort_values(ascending=False).head(topn)
    plt.figure(figsize=(8, max(4, topn*0.35)))
    s.iloc[::-1].plot(kind="barh")
    plt.title(title)
    plt.xlabel("Importance")
    plt.tight_layout()
    plt.show()

plot_top_importances(xgb_tuned, feature_names, topn=20, title="[XGBoost] Top 20 Feature Importance")

```



- 모델이 정기 예금 가입 가능성을 예측할 때 가장 크게 참고하는 변수는 고양자 수 & 고용 변동률인 것으로 확인.
- 두 변수가 높을 수록 가입 가능성이 높은것을 확인.
- 이전 캠페인의 성공 여부가 두번째 요인 변수임을 확인.
- 해당 요인 변수들을 토대로 정기 예금 가능성을 높일 수 있는 마케팅 방법 파악.