Predict

if the client will subscribe a bank term deposit

홍수지

목차

데이터 정보

Input variables:

• Bank client data:

• 1 - id: client ID

• 2 - age (numeric)

• 3 - job : type of job (categorical: "admin.","blue-collar","entrepreneur","housemaid","management","retired","self-employed","services","student","technician","unemployed","unknown")

• 4 - marital : marital status (categorical: "divorced","married","single","unknown"; note: "divorced" means divorced or widowed)

• 5 - education (categorical: "basic.4y","basic.6y","basic.9y","high.school","illiterate","professional.course","university.degree","unknown")

• 6 - default: has credit in default? (categorical: "no","yes","unknown")

• 7 - housing: has housing loan? (categorical: "no","yes","unknown")

• 8 - loan: has personal loan? (categorical: "no","yes","unknown")

• Related with the last contact of the current campaign:

• 9 - contact: contact communication type (categorical: "cellular","telephone")

• 10 - month: last contact month of year (categorical: "jan", "feb", "mar", ..., "nov", "dec")

• 11 - day of week: last contact day of the week (categorical: "mon","tue","wed","thu","fri")

• Other attributes:

• 12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

• 13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

• 14 - previous: number of contacts performed before this campaign and for this client (numeric)

• 15 - poutcome: outcome of the previous marketing campaign (categorical: "failure","nonexistent","success")

• Social and economic context attributes

• 16 - emp.var.rate: employment variation rate - quarterly indicator (numeric)

• 17 - cons.price.idx: consumer price index - monthly indicator (numeric)

• 18 - cons.conf.idx: consumer confidence index - monthly indicator (numeric)

• 19 - euribor3m: euribor 3 month rate - daily indicator (numeric)

• 20 - nr.employed: number of employees - quarterly indicator (numeric)

• Output variable (desired target):

• 21 – y: has the client subscribed a term deposit? (binary: "yes","no")

데이터 탐색: 시각화 및 통계량 확인 등

- train 데이터와 test 데이터의 행과 열의 개수
    - train 데이터의 행과 열의 개수: (32950,21)
    - test 데이터의 행과 열의 개수:(8238,20)

- train 데이터의 columns의 data type

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32950 entries, 0 to 32949
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   id              32950 non-null  int64  
 1   age             32950 non-null  int64  
 2   job             32950 non-null  object 
 3   marital         32950 non-null  object 
 4   education       32950 non-null  object 
 5   default         32950 non-null  object 
 6   housing         32950 non-null  object 
 7   loan            32950 non-null  object 
 8   contact         32950 non-null  object 
 9   month           32950 non-null  object 
 10  day_of_week     32950 non-null  object 
 11  campaign        32950 non-null  int64  
 12  pdays           32950 non-null  int64  
 13  previous        32950 non-null  int64  
 14  poutcome        32950 non-null  object 
 15  emp.var.rate    32950 non-null  float64
 16  cons.price.idx  32950 non-null  float64
 17  cons.conf.idx   32950 non-null  float64
 18  euribor3m       32950 non-null  float64
 19  nr.employed     32950 non-null  float64
 20  y               32950 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 5.3+ MB
```

- train 데이터의 기초 통계량 확인

	id	age	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
count	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000	32950.000000
mean	20569.615569	40.023703	2.567830	962.415964	0.172838	0.083129	93.576610	-40.500091	3.622516	5167.036495
std	11895.520420	10.401749	2.766994	187.054556	0.498098	1.571951	0.578725	4.632363	1.734791	72.250873
min	1.000000	17.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600000
25%	10258.250000	32.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100000
50%	20571.000000	38.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000000
75%	30846.750000	47.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100000
max	41188.000000	98.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100000

위의 기초 통계량에 따르면 age가 음수인 이상치는 존재하지 않는다는 것을 알 수 있다.

- test 데이터의 columns의 data type

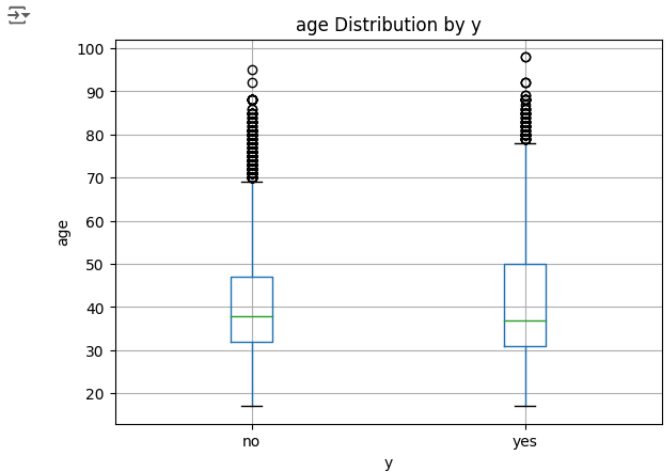
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8238 entries, 0 to 8237
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              8238 non-null   int64
 1   age             8238 non-null   int64
 2   job             8238 non-null   object
 3   marital         8238 non-null   object
 4   education       8238 non-null   object
 5   default         8238 non-null   object
 6   housing         8238 non-null   object
 7   loan            8238 non-null   object
 8   contact         8238 non-null   object
 9   month           8238 non-null   object
 10  day_of_week     8238 non-null   object
 11  campaign        8238 non-null   int64
 12  pdays           8238 non-null   int64
 13  previous        8238 non-null   int64
 14  poutcome        8238 non-null   object
 15  emp.var.rate    8238 non-null   float64
 16  cons.price.idx  8238 non-null   float64
 17  cons.conf.idx   8238 non-null   float64
 18  euribor3m       8238 non-null   float64
 19  nr.employed     8238 non-null   float64
dtypes: float64(5), int64(5), object(10)
memory usage: 1.3+ MB
```

- 각 변수들의 상관관계

	id	age	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
id	1.000000	-0.003518	-0.089736	-0.286184	0.423373	-0.832593	-0.737360	-0.263166	-0.848403	-0.754331
age	-0.003518	1.000000	0.017021	-0.036528	0.017812	0.012417	0.001636	0.124263	0.022828	-0.002547
campaign	-0.089736	0.017021	1.000000	0.048346	-0.074408	0.147741	0.124982	-0.007773	0.132583	0.141420
pdays	-0.286184	-0.036528	0.048346	1.000000	-0.599908	0.265586	0.057789	-0.084041	0.299288	0.383054
previous	0.423373	0.017812	-0.074408	-0.599908	1.000000	-0.410565	-0.184848	-0.051650	-0.448454	-0.502237
emp.var.rate	-0.832593	0.012417	0.147741	0.265586	-0.410565	1.000000	0.773736	0.199372	0.972037	0.905933
cons.price.idx	-0.737360	0.001636	0.124982	0.057789	-0.184848	0.773736	1.000000	0.053648	0.686035	0.517846
cons.conf.idx	-0.263166	0.124263	-0.007773	-0.084041	-0.051650	0.199372	0.053648	1.000000	0.282141	0.109111
euribor3m	-0.848403	0.022828	0.132583	0.299288	-0.448454	0.972037	0.686035	0.282141	1.000000	0.944738
nr.employed	-0.754331	-0.002547	0.141420	0.383054	-0.502237	0.905933	0.517846	0.109111	0.944738	1.000000

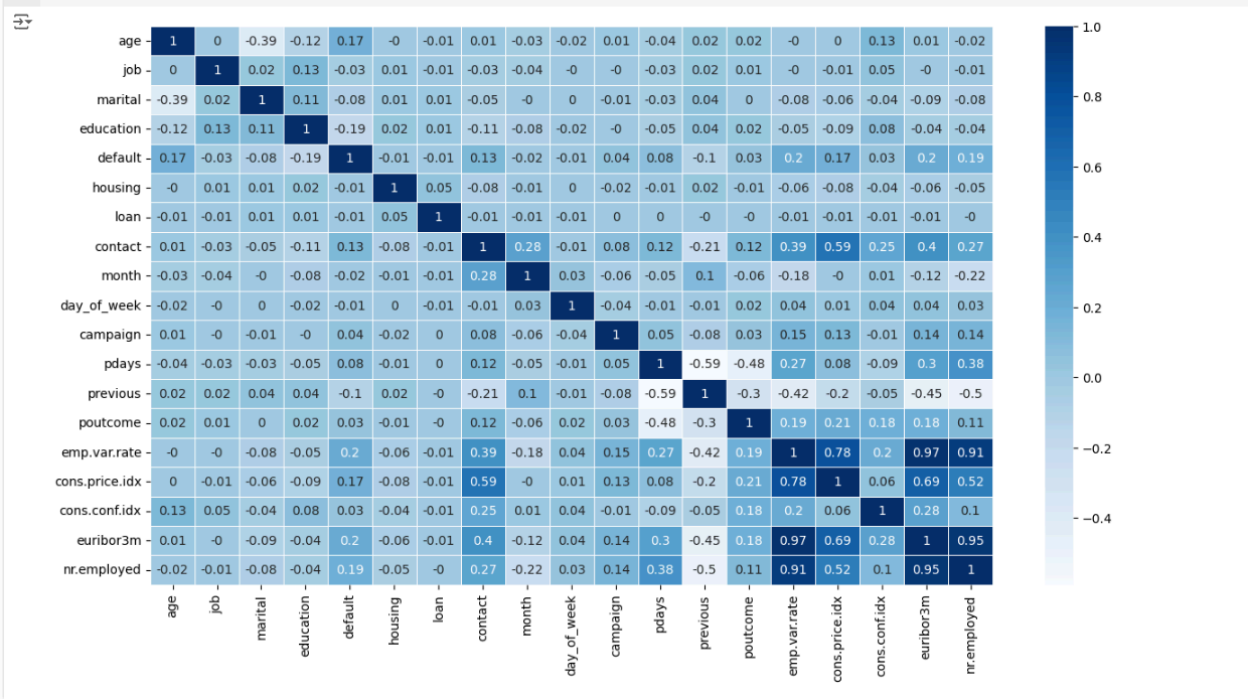
test데이터의 'age' 변수 또한 음수를 가진 이상치가 없는것을 알 수 있다.

- train 데이터의 age 값을 y축으로 target 값을 x축을 한 Boxplot



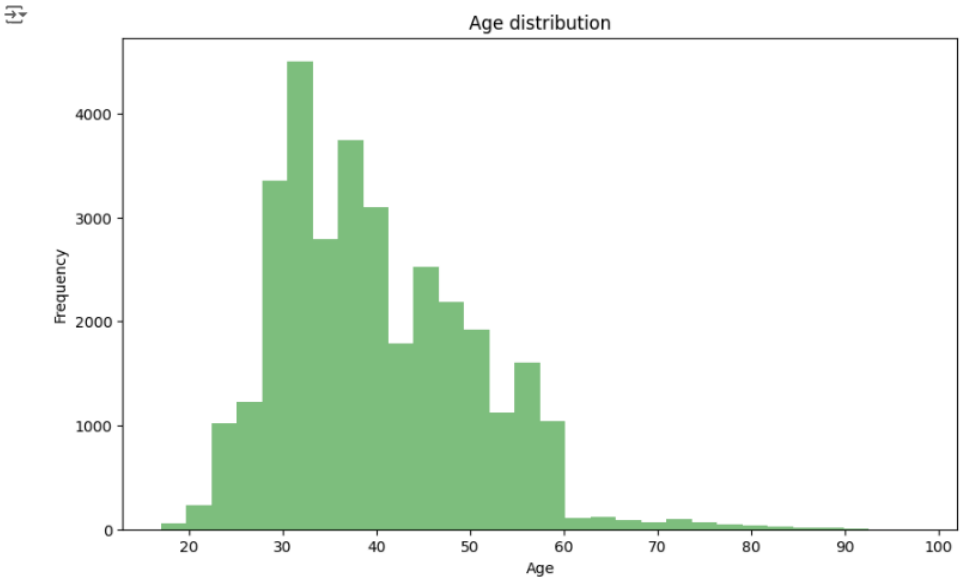
사분위수를 활용하여 확인했을 때 age의 이상치는 80이상으로 볼 수 있다.

- 각 변수들의 상관관계를 보여주는 heatmap

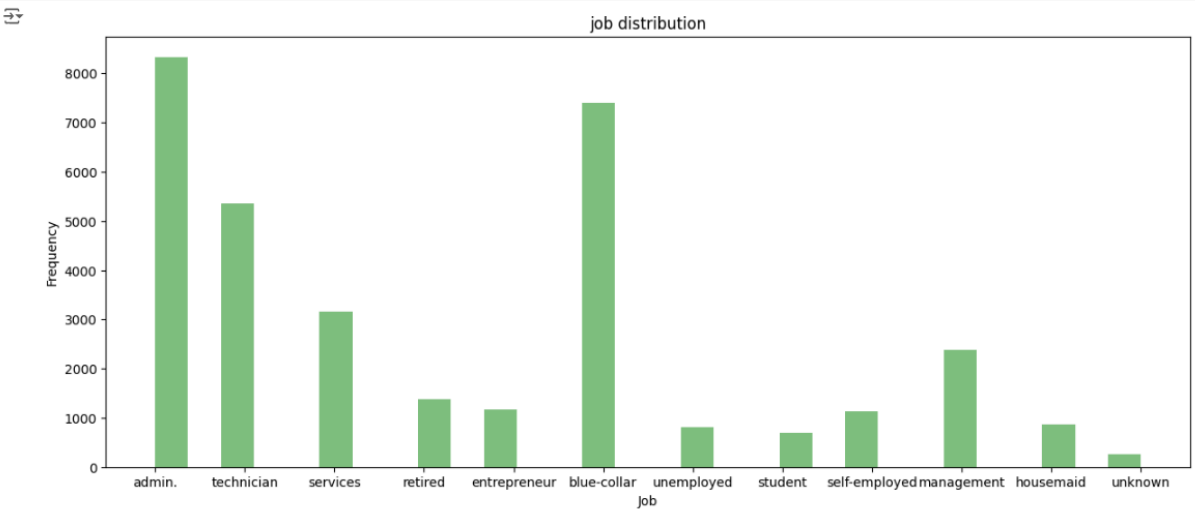


상관관계가 0.9가 넘는 관계인 emp.var.rate와 euribor3m, euribor3m 과 nr.employed, emp.var.rate와 nr.employed로 높아 중복 정보를 제공할 수 있으므로 데이터 전처리 과정에서 처리해보고 F1 Score를 전과 후 비교

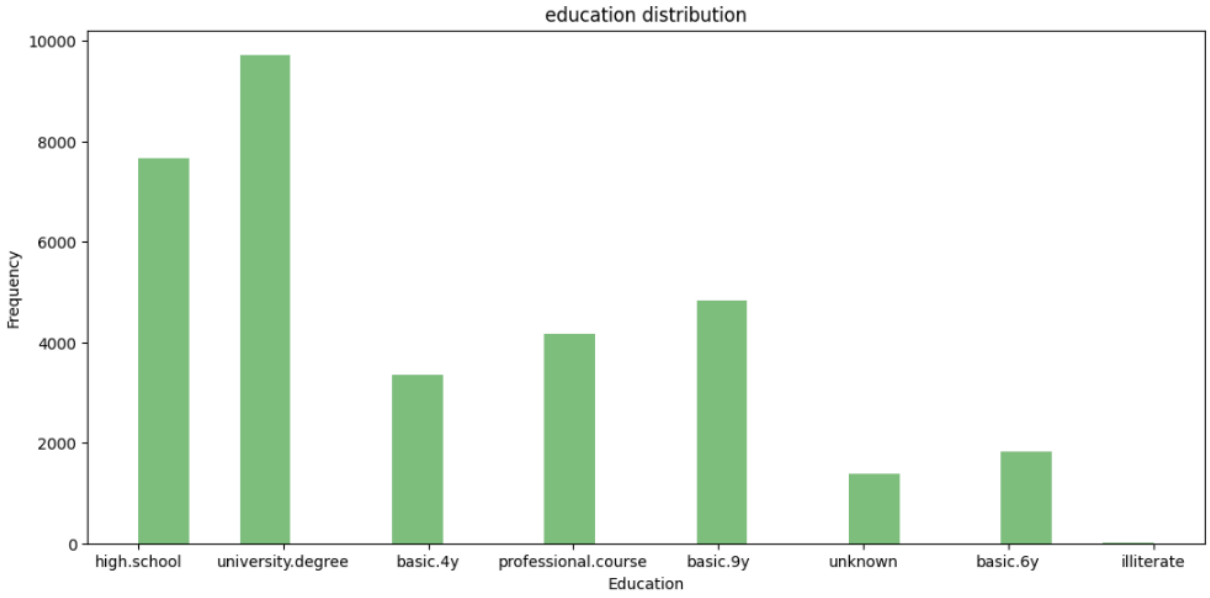
- ‘age’ 변수의 분포를 나타내는 히스토그램



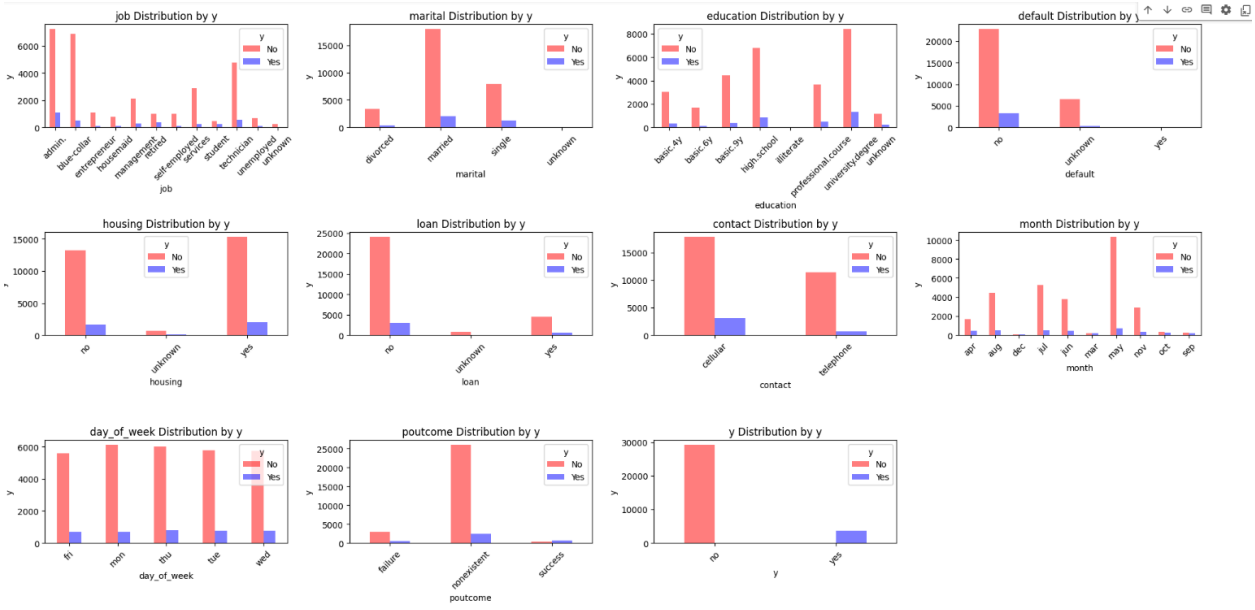
- ‘job’ 변수의 분포를 나타내는 히스토그램



- ‘education’변수의 분포를 나타내는 히스토그램



- data type이 object인 변수들의 분포가 ‘y’변수에 따라 어떻게 달라지는지 나타내는 바 그래프



데이터 전처리: 정규화, 인코딩, 결측치/이상치 처리

- train 데이터와 test데이터의 결측치확인

```
id              0
age             0
job             0
marital         0
education       0
default         0
housing         0
loan            0
contact         0
month           0
day_of_week     0
campaign        0
pdays           0
previous        0
poutcome        0
emp.var.rate    0
cons.price.idx  0
cons.conf.idx   0
euribor3m       0
nr.employed     0
y               0
dtype: int64
```

```
id              0
age             0
job             0
marital         0
education       0
default         0
housing         0
loan            0
contact         0
month           0
day_of_week     0
campaign        0
pdays           0
previous        0
poutcome        0
emp.var.rate    0
cons.price.idx  0
cons.conf.idx   0
euribor3m       0
nr.employed     0
dtype: int64
```

trian데이터와 test데이터의 결측치는 없으므로 따로 처리하지 않았다

- 변수 처리

'id' 변수는 unique한 값으로 train에 있는 'id'변수는 drop 시키고 test 데이터에 있는 'id' 값은 pop으로 따로 뽑아주었습니다

그리고 target값으로 train데이터에 있는 'y'변수 pop을 사용하여 따로 정의했습니다

- 이상치 처리

  Python의 scipy 라이브러리에서 winsorize 함수를 사용하여 특정 백분위수 이상의 값들을 해당 백분위수 값으로 대체하고 극단값의 영향을 줄이면서도 데이터 손실을 최소화하는 방법을 'age' 변수에 적용했습니다

- 인코딩

  인코딩 방법은 Label Encoding와 One-hot Encoding 두 방법 중 선택했습니다.

  각 인코딩을 실행후 데이터의 shape을 확인했습니다.

  - Label Encoding 실행 후 행과 열의 개수:
    - trian: (32950,19)
    - test: (8238,19)
  - One-Hot Encoding 실행 후 행과 열의 개수:
    - train: (32950, 115)
    - test: (8238,115)

  - Label Encoding
    - 장점: 간단하고 빠르며 메모리가 절약된다
    - 단점: 순서 관계에 문제가 생긴다
  - One-Hot Encoding
    - 장점: 순서 관계가 없고 일부 모델에서는 더 잘 작동한다
    - 단점: 메모리 소모가 크며 고차원 문제가 발생한다

  위의 이유로 LinearRegression 과 Logistic Regression에서는 One-Hot Encoding 방법으로 실행하고 나머지는 Label Encoding으로 실행할 예정입니다.

  (Linear Regression은 f1 score가 불가능하므로 참고용으로 실행해 보았습니다)

  - target값인 'y' 컬럼을 no:0, yes:1로 replace해줍니다

모형 학습 및 검증

1. Linear Regression (선형회귀)

    - f1_score: 0.30912863070539415
    - roc_auc_score: 0.7772312365440998
    - accuracy_score: 0.898937784522003

2. LogisticRegression (로지스틱 회귀)

    - f1_score: 0.29442691903259727
    - roc_auc_score: 0.7590580245564378
    - accuracy_score: 0.8981790591805766

3. Decision Tree(의사결정 나무)

    - f1_score:0.328732747804266
    - roc_auc_score: 0.6220088294303348
    - accuracy_score: 0.8333839150227618

- 튜닝: max_depth=5, min_samples_split=2, min_impurity_decrease=0.0

    - f1_score: 0.3730084348641049
    - roc_auc_score: 0.7704943798240483
    - accuracy_score: 0.8984825493171472

4. Bagging (배깅)

    - f1_score:0.328732747804266
    - roc_auc_score: 0.6220088294303348
    - accuracy_score: 0.8333839150227618

- 튜닝: max_depth=5, min_samples_split=2, min_impurity_decrease=0.0

    - f1_score: 0.3730084348641049
    - roc_auc_score: 0.7704943798240483
    - accuracy_score: 0.8984825493171472

5. RandomForest (랜덤포레스트)

    - f1_score: 0.36094158674803833
    - roc_auc_score: 0.7660195233555592
    - accuracy_score: 0.8887708649468892

- 튜닝: max_depth=9, min_samples_split=2, bootstrap=True,n_jobs=-1

    - f1_score: 0.36084452975047987
    - roc_auc_score: 0.7866949571347172
    - accuracy_score: 0.898937784522003

6. Ada Boost

    - f1_score: 0.30443548387096775
    - roc_auc_score: 0.7787547016178124
    - accuracy_score: 0.8952959028831563

- 튜닝: n_estimators=100

    - f1_score::0.33070866141732286
    - roc_auc_score: 0.7820928302175557
    - accuracy_score: 0.8968133535660091

7. GradientBoosting

    - f1_score: 0.3655502392344498
    - roc_auc_score: 0.7859298452295118
    - accuracy_score: 0.8993930197268589

- 튜닝: n_estimators=50,subsample=1.0

    - f1_score: 0.34604105571847504
    - roc_auc_score: 0.7815021853630506
    - accuracy_score: 0.8984825493171472

8. XGBoost

    - f1_score: 0.3599290780141844
    - roc_auc_score: 0.7684264355041249
    - accuracy_score: 0.8904400606980273

- 튜닝: n_estimators=50, subsample=1.0, learning_rate=0.1, max_depth=6, n_jobs=-1

    - f1_score: 0.3691148775894539
    - roc_auc_score: 0.7919623041197822
    - accuracy_score:0.8983308042488619

9. LightGBM

    - f1_score: 0.368271954674221
    - roc_auc_score: 0.7865253013881071
    - accuracy_score: 0.8984825493171472

- 튜닝: n_estimators=50,data_sample_strategy="goss", top_rate=0.2,other_rate=0.1,force_col_wise=True,verbosity=0,n_jobs=-1

    - f1_score: 0.3779232927970065
    - roc_auc_score: 0.7931400426281543
    - accuracy_score: 0.8990895295902883

학습된 모형 및 예측 결과에 대한 해석/분석

- F1: 모델이 불균형한 데이터에서 양성과 음성을 균형 있게 잘 예측하고, Precision과 Recall을 모두 높게 유지하기 때문에 F1 Score가 높을수록 모델의 성능이 좋다고 판단할 수 있습니다.
- ROC AUC: 모델이 다양한 임계값에서 일관되게 좋은 분류 성능을 제공하고, 특히 클래스 불균형 데이터에서도 강력한 성능을 나타나 ROC AUC Score가 1에 가까울수록 모델의 성능이 좋다고 판단할 수 있습니다.
- Accuracy:모델이 대부분의 예측을 정확하게 수행하고, 클래스가 균형 잡힌 데이터셋에서 신뢰할 수 있는 성능을 보여 일반적으로 높은 정확도는 모델의 성능이 좋다는 것을 의미합니다. 그러나 정확도가 항상 모델의 성능을 완벽하게 반영하지는 않습니다.

모델의 성능을 평가하고 비교하기 위해 train_test_split을 사용하여 train데이터를 학습용 데이터(x_tr, y_tr)와 검증용 데이터(x_val, y_val)로 분리하였습니다. 각각의 모델을 학습한 후, 검증 데이터로 예측값을 생성하여 성능 지표(F1 Score, Accuracy, ROC AUC Score)를 통해 평가하였습니다.

그 중 F1 Score가 높게 나온 모델인 lightGBM(n_estimators=50,data_sample_strategy="goss", top_rate=0.2,other_rate=0.1,force_col_wise=True,verbosity=0,n_jobs=-1)을 선택했습니다.

위 모델의 F1 Score는 0.3779232927970065, ROC AUC Score는 0.7931400426281543 그리고Accuracy는 0.8990895295902883 가 나왔습니다.

Test set에 대한 예측

위에서 선택한 모델 lightGBM(n_estimators=50,data_sample_strategy="goss", top_rate=0.2,other_rate=0.1,force_col_wise=True,verbosity=0,n_jobs=-1)로 test 데이터의 예측결과를 뽑았습니다.

예측결과의 0과 1의 비율을 비교해보니

```
0    0.955329
1    0.044671
```

로 0이 즉, No 결과가 더 많은 것을 알 수 있었습니다.

결과 해석 및 분석

Main metric인 F1 Score를 선택 했을 때 모델 대부분의 F1 Score가 0.5보다 작게 나온 것을 확인할 수 있습니다. F1 score는 정밀도(Precision)와 재현율(Recall)의 조화 평균으로, 특히 데이터셋이 불균형할 때 유용한 지표입니다. 본 프로젝트에서 F1 스코어는 0.5보다 작게 나왔습니다. 이는 모델이 양성과 음성을 구분하는 데 어려움을 겪고 있음을 나타냅니다. 특히, 양성 클래스의 예측 성능이 낮을 가능성이 큽니다.

모델 선택**: LightGBM**

LightGBM 모델이 다른 모델의 F1 Score를 비교하면 가장 높은 F1 Score 기록하여, 최종 모델로 선택되었습니다. F1 스코어가 0.5보다 낮지만, 이는 데이터셋의 불균형 문제와 관련이 있을 수 있습니다.

데이터 불균형 문제: F1 스코어가 낮게 나온 주된 원인은 데이터의 불균형일 가능성이 큽니다. 대부분의 샘플이 'No' 클래스로 분류되고, 'Yes' 클래스의 샘플이 상대적으로 적기 때문에,

모델이 'Yes' 클래스를 잘 예측하지 못하는 경향이 있습니다.

이 불균형 문제를 해결할 수있는 방안을 찾아보면 아래와 같다.

성능 개선 방안

1. 데이터 불균형 해결: 오버샘플링(SMOTE), 언더샘플링 등의 기법을 사용하여 데이터셋의 불균형을 해결할 수 있습니다.
2. 모델 튜닝: LightGBM 모델의 하이퍼파라미터를 최적화하여 성능을 향상시킬 수 있습니다.
3. 앙상블 기법 사용: 여러 모델을 앙상블하여 보다 강력한 예측 성능을 얻을 수 있습니다.
4. 다양한 평가 지표 사용: 단일 평가 지표에 의존하지 않고, F1 스코어, ROC AUC, 정확도 등 여러 지표를 함께 고려하여 모델의 성능을 평가할 필요가 있습니다.

이번 프로젝트를 통해 LightGBM 모델이 가장 우수한 성능을 보이는 것으로 나타났습니다. 그러나 F1 스코어가 낮게 나온 점을 감안할 때, 데이터셋의 불균형 문제를 해결하고 모델의 예측 성능을 종합적으로 평가하는 것이 중요합니다. 향후 연구에서는 데이터 전처리와 모델 튜닝을 통해 성능을 더욱 개선할 수 있다고하여 Standard Scale을 실행하고 오버 샘플링(SMOTE)를 적용하였다.

추가 전처리 결과

1. Linear Regression (선형회귀)

    - f1_score: 0.4057724957555178
    - roc_auc_score: 0.7713982176638553
    - accuracy: 0.787556904400607

2. LogisticRegression (로지스틱 회귀)

    - f1_score: 0.46739757645701097
    - roc_auc_score: 0.7533984417331601
    - accuracy_score: 0.8599393019726859

3. Decision Tree(의사결정 나무)

- 튜닝: max_depth=5, min_samples_split=2, min_impurity_decrease=0.0

    - f1_score: 0.46739757645701097
    - roc_auc_score: 0.7533984417331601
    - accuracy_score: 0.8599393019726859

4. Bagging (배깅)

- 튜닝: max_depth=5, min_samples_split=2, min_impurity_decrease=0.0

    - f1_score: 0.4114114114114114
    - roc_auc_score: 0.7559794772678482
    - accuracy_score: 0.88103186646434

5. RandomForest (랜덤포레스트)

- 튜닝: max_depth=9, min_samples_split=2, bootstrap=True,n_jobs=-1

    - f1_score: 0.4772087788407428
    - roc_auc_score: 0.7834703524005314
    - accuracy_score: 0.8590288315629742

6. Ada Boost

- 튜닝: n_estimators=100

    - f1_score:0.4382227632379793
    - roc_auc_score: 0.7609639883053234

- accuracy_score: 0.8599393019726859

7. GradientBoosting

- 튜닝: n_estimators=50,subsample=1.0

- f1_score: 0.47058823529411764
- roc_auc_score: 0.7816323196832053
- accuracy_score: 0.87298937784522

8. XGBoost

- 튜닝: n_estimators=50, subsample=1.0, learning_rate=0.1, max_depth=6, n_jobs=-1

- f1_score: 0.46923076923076923
- roc_auc_score: 0.7786835630501927
- accuracy_score:0.8743550834597875

9. LightGBM

- 튜닝: n_estimators=50,data_sample_strategy="goss", top_rate=0.2,other_rate=0.1,force_col_wise=True,verbosity=0,n_jobs=-1

- f1_score: 0.4638472032742156
- roc_auc_score: 0.7802780521366657
- accuracy_score: 0.8807283763277693

성능 개선

학습된 모형 및 예측 결과에 대한 해석/분석

모델 선택**: Random Forest Classifier**

Standard Scale과 오버 샘플링(SMOTE)을 적용한 후 F1 Score가 높게 나온 모델인 Random Forest Classifier(max_depth=9, min_samples_split=2, bootstrap=True,n_jobs=-1)을 선택했습니다.

위 모델의 F1 Score는 0.4772087788407428, ROC AUC Score는 0.7834703524005314 그리고Accuracy는 0.8590288315629742 가 나왔습니다.

Test set에 대한 예측

위에서 선택한 모델 Random Forest Classifier(max_depth=9, min_samples_split=2, bootstrap=True,n_jobs=-1)로 test 데이터의 예측결과를 뽑았습니다.

예측결과의 0과 1의 비율을 비교해보니

```
y_predict
1    0.630129
0    0.369871
Name: proportion, dtype: float64
```

위와 같이 'Yes' 전체의 0.630129, 'No'는 전체의 0.369871이 나왔습니다.

결과 해석 및 분석

F1 Score는 모델의 정밀도(Precision)와 재현율(Recall)의 조화 평균을 나타내며, 불균형 데이터에서 중요한 성능 지표입니다. 이번 분석에서 F1 Score는 0.4772로, 이는 모델이 긍정 클래스와 부정 클래스를 균형 있게 예측하지 못하고 있음을 의미합니다. 즉, 모델이 False Positive와 False Negative를 충분히 줄이지 못하고 있다는 것을 나타냅니다.

결과 분석

F1 Score가 상대적으로 낮다는 것은 모델이 긍정 클래스를 잘 잡아내지 못하고 있음을 나타냅니다. 이는 불균형 데이터에서 흔히 발생하는 문제로, 긍정 클래스가 충분히 학습되지 못했을 가능성이 있습니다. 반면, ROC AUC Score와 Accuracy는 비교적 높은 점수를 보이고 있어, 모델이 전반적으로 긍정 클래스와 부정 클래스를 잘 구분하고 있음을 나타냅니다.

성능 개선 방안

1. 교차 검증: 교차 검증을 통해 모델의 일반화 성능을 평가하고 과적합을 방지할 수 있습니다.
2. 앙상블 기법: 여러 모델을 결합하여 예측하는 앙상블 기법을 사용할 수 있습니다.

이번 분석을 통해 Random Forest 모델의 성능을 평가한 결과, F1 Score는 0.4772로 전처리 전보다는 성능이 개선되었지만, ROC AUC Score는 0.7835로, Accuracy는 0.8590로 추가 전처리 전 ROC AUC Score와 Accuracys 보다 낮아진 것으로 보입니다. 성능 개선을 위해 하이퍼파라미터 튜닝, 더 나은 데이터 전처리, 다른 모델 시도, 교차 검증, 앙상블 기법 등을 고려해볼 수 있습니다. 이러한 방법을 통해 모델의 F1 Score를 향상시키고, 궁극적으로 모델의 전반적인 예측 성능을 개선할 수 있을 것입니다.