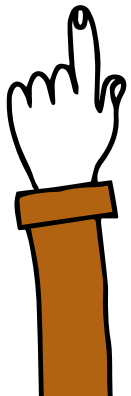
Telco 통신사 탈퇴 데이터

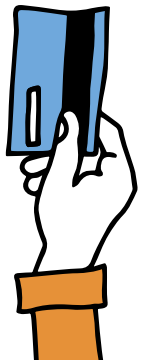
🔼 High_five 🖐

INDEX

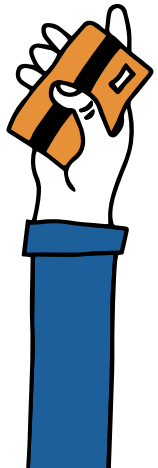
주제 선정
&
데이터 소개



데이터 전처리
&
시각화



Machine Learning Model

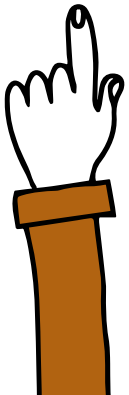


Deep Learning Model

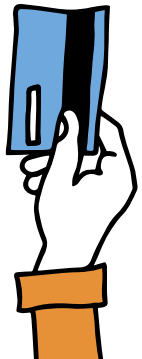
결과
&
추가 분석 과제

팀원 소개

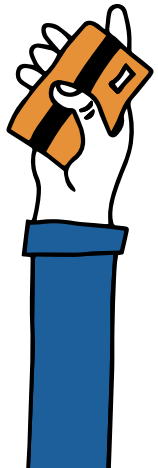
안태용



임은형

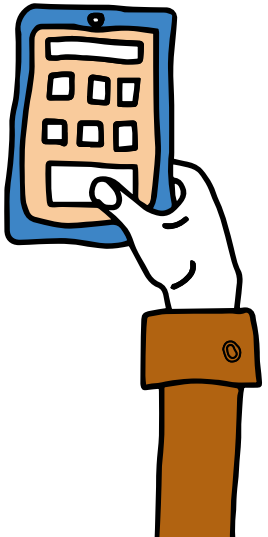


안지현



송다현

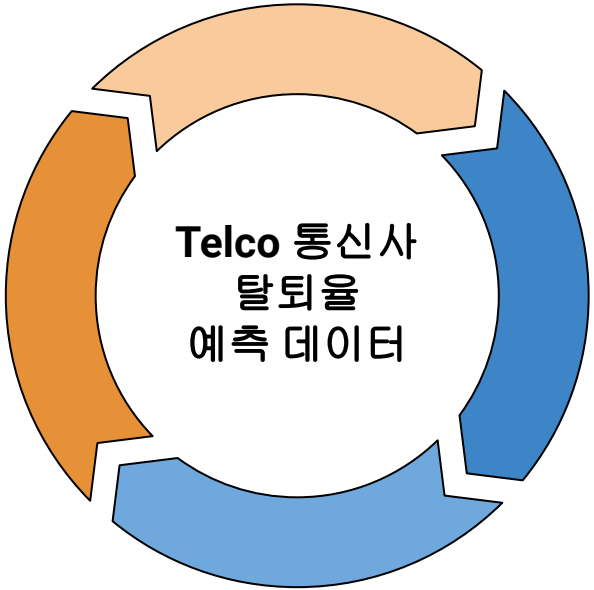
김나연



주제 선정 및 데이터 소개

주제 선정

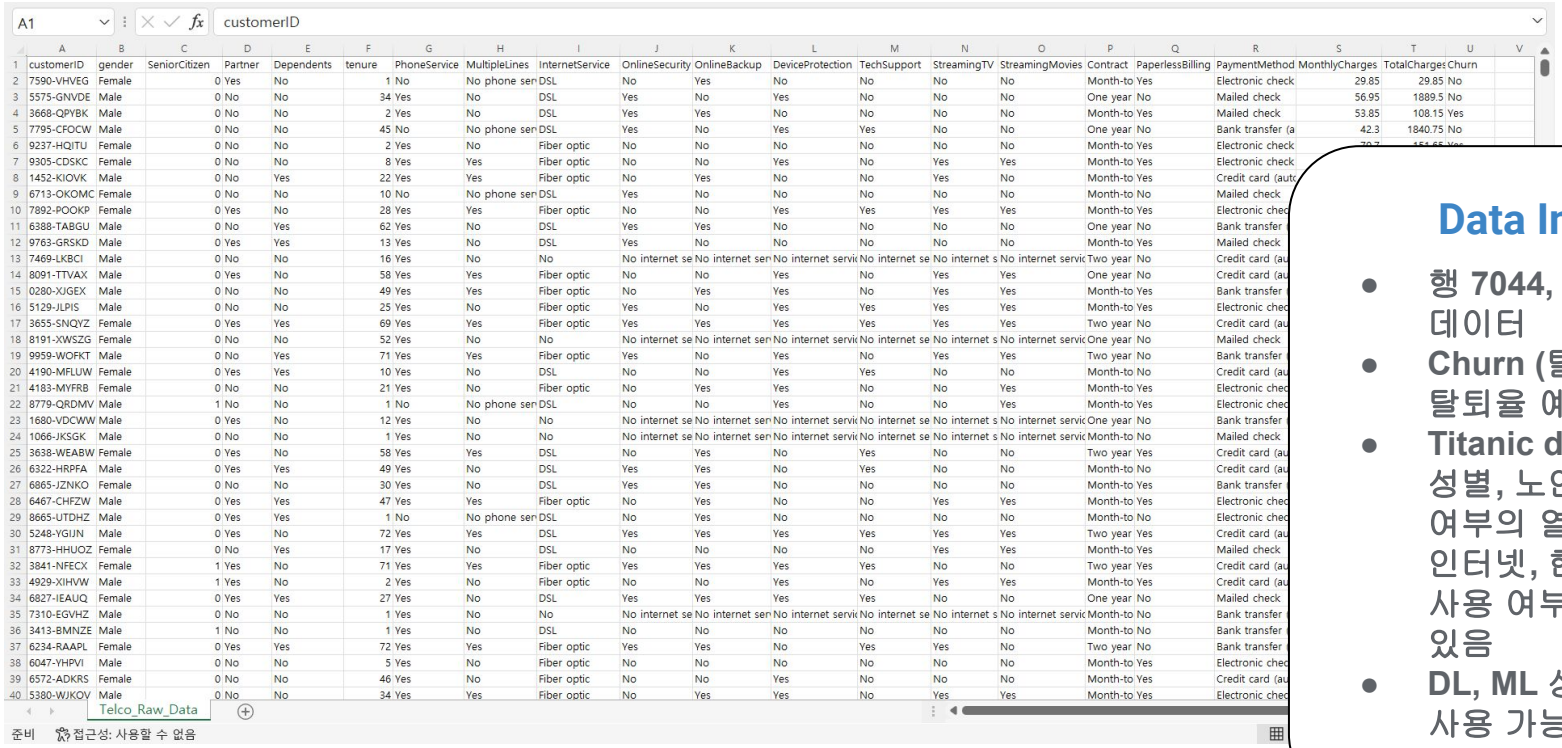
- **Kaggle**에서 여러 모델을 사용한 코드가 있는 데이터 위주로 선정
- 많은 행이 있는 데이터를 선정함
- 결측치를 채우는 것보다 행을 선별하는 데이터 위주로 선정함
- 예측 데이터



Data Information

- 행 **7044**, 열 **21**의 통신사 데이터
- **Churn** (탈퇴) 열을 통한 탈퇴율 예측 데이터
- **Titanic data**와 비슷하게 성별, 노인, 동거인, 자녀 여부의 열이 있으며, 인터넷, 핸드폰, 티비선 사용 여부 등 자세한 열이 있음
- **DL, ML** 상관없이 여러 모델 사용 가능

주제 선정 및 데이터 소개



A1 customerID

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
2	7590-VHVEG	Female	0	Yes	No	1	No	No phone ser	DSL	No	Yes	No	No	No	No	Month-to	Yes	Electronic check	29.85	29.85	No
3	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	No	No	One year	No	Mailed check	56.95	1889.5	No
4	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	No	No	Month-to	Yes	Mailed check	53.85	108.15	Yes
5	7795-CFOCW	Male	0	No	No	45	No	No phone ser	DSL	Yes	No	Yes	Yes	No	No	One year	No	Bank transfer (a	42.3	1840.75	No
6	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	No	No	Month-to	Yes	Electronic check			
7	9305-CDSKC	Female	0	No	No	8	Yes	Yes	Fiber optic	No	No	Yes	No	Yes	Yes	Month-to	Yes	Electronic check			
8	1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	Fiber optic	No	Yes	No	No	Yes	No	Month-to	Yes	Credit card (auto			
9	6713-OKOMC	Female	0	No	No	10	No	No phone ser	DSL	Yes	No	No	No	No	No	Month-to	No	Mailed check			
10	7892-POOKP	Female	0	Yes	No	28	Yes	Yes	Fiber optic	No	No	Yes	Yes	Yes	Yes	Month-to	Yes	Electronic chec			
11	6388-TABGU	Male	0	No	Yes	62	Yes	No	DSL	Yes	Yes	No	No	No	No	One year	No	Bank transfer (			
12	9763-GRSKD	Male	0	Yes	Yes	13	Yes	No	DSL	Yes	No	No	No	No	No	Month-to	Yes	Mailed check			
13	7469-LKBCI	Male	0	No	No	16	Yes	No	No	No internet se	No internet ser	No internet servi	No internet se	No internet s	No internet servic	Two year	No	Credit card (au			
14	8091-TTVAX	Male	0	Yes	No	58	Yes	Yes	Fiber optic	No	No	Yes	No	Yes	Yes	One year	No	Credit card (au			
15	0280-XJGEX	Male	0	No	No	49	Yes	Yes	Fiber optic	No	Yes	Yes	No	Yes	Yes	Month-to	Yes	Bank transfer (			
16	5129-JLPIS	Male	0	No	No	25	Yes	No	Fiber optic	Yes	No	Yes	Yes	Yes	Yes	Month-to	Yes	Electronic chec			
17	3655-SNQYZ	Female	0	Yes	Yes	69	Yes	Yes	Fiber optic	Yes	Yes	Yes	Yes	Yes	Yes	Two year	No	Credit card (au			
18	8191-XWSZG	Female	0	No	No	52	Yes	No	No	No internet se	No internet ser	No internet servi	No internet se	No internet s	No internet servic	One year	No	Mailed check			
19	9959-WOFKT	Male	0	No	Yes	71	Yes	Yes	Fiber optic	Yes	No	Yes	No	Yes	Yes	Two year	No	Bank transfer (			
20	4190-MFLUW	Female	0	Yes	Yes	10	Yes	No	DSL	No	No	Yes	Yes	No	No	Month-to	No	Credit card (au			
21	4183-MYFRB	Female	0	No	No	21	Yes	No	Fiber optic	No	Yes	Yes	No	No	Yes	Month-to	Yes	Electronic chec			
22	8779-QRDMV	Male	1	No	No	1	No	No phone ser	DSL	No	No	Yes	No	No	No	Month-to	Yes	Electronic chec			
23	1680-VDCWW	Male	0	Yes	No	12	Yes	No	No	No internet se	No internet ser	No internet servi	No internet se	No internet s	No internet servic	One year	No	Bank transfer (			
24	1066-JKSGK	Male	0	No	No	1	Yes	No	No	No internet se	No internet ser	No internet servi	No internet se	No internet s	No internet servic	Month-to	No	Mailed check			
25	3638-WEABW	Female	0	Yes	No	58	Yes	Yes	DSL	No	Yes	No	Yes	No	No	Two year	Yes	Credit card (au			
26	6322-HRPFA	Male	0	Yes	Yes	49	Yes	No	DSL	Yes	Yes	No	Yes	No	No	Month-to	No	Credit card (au			
27	6865-JZNKO	Female	0	No	No	30	Yes	No	DSL	Yes	Yes	No	No	No	No	Month-to	Yes	Bank transfer (			
28	6467-CHFZW	Male	0	Yes	Yes	47	Yes	Yes	Fiber optic	No	Yes	No	No	Yes	Yes	Month-to	Yes	Electronic chec			
29	8665-UTDHZ	Male	0	Yes	Yes	1	No	No phone ser	DSL	No	Yes	No	No	No	No	Month-to	No	Electronic chec			
30	5248-YGIJN	Male	0	Yes	No	72	Yes	Yes	DSL	Yes	Yes	Yes	Yes	Yes	Yes	Two year	Yes	Credit card (au			
31	8773-HHUOZ	Female	0	No	Yes	17	Yes	No	DSL	No	No	No	No	Yes	Yes	Month-to	Yes	Mailed check			
32	3841-NFECX	Female	1	Yes	No	71	Yes	Yes	Fiber optic	Yes	Yes	Yes	No	No	No	Two year	Yes	Credit card (au			
33	4929-XIHVW	Male	1	Yes	No	2	Yes	No	Fiber optic	No	No	Yes	No	Yes	Yes	Month-to	Yes	Credit card (au			
34	6827-IEAUQ	Female	0	Yes	Yes	27	Yes	No	DSL	Yes	Yes	Yes	No	No	No	One year	No	Mailed check			
35	7310-EGVHZ	Male	0	No	No	1	Yes	No	No	No internet se	No internet ser	No internet servi	No internet se	No internet s	No internet servic	Month-to	No	Bank transfer (			
36	3413-BMNZE	Male	1	No	No	1	Yes	No	DSL	No	No	No	No	No	No	Month-to	No	Bank transfer (			
37	6234-RAAPL	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	Yes	Yes	No	Yes	Yes	No	Two year	No	Bank transfer (			
38	6047-YHPVI	Male	0	No	No	5	Yes	No	Fiber optic	No	No	No	No	No	No	Month-to	Yes	Electronic chec			
39	6572-ADKRS	Female	0	No	No	46	Yes	No	Fiber optic	No	No	Yes	No	No	No	Month-to	Yes	Credit card (au			
40	5380-WJKOV	Male	0	No	No	34	Yes	Yes	Fiber optic	No	Yes	Yes	No	Yes	Yes	Month-to	Yes	Electronic chec			

Telco_Raw_Data

준비 접근성: 사용할 수 없음

Data Information

- 행 **7044**, 열 **21**의 통신사 데이터
- **Churn** (탈퇴) 열을 통한 탈퇴율 예측 데이터
- **Titanic data**와 비슷하게 성별, 노인, 동거인, 자녀 여부의 열이 있으며, 인터넷, 핸드폰, 티비선 사용 여부 등 자세한 열이 있음
- **DL, ML** 상관없이 여러 모델 사용 가능

DATA EDA 및 시각화

```
raw_data.isnull().sum()
```

```
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

Null이 아니기에 결측치 찾는 코드에서는 보이지 않음

```
raw_data.sort_values(by='TotalCharges').head(10)
```

...eSecurity	...	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
Yes	...	Yes	No	Yes	Yes	Two year	No	Mailed check	80.85		No
No internet service	...	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	25.35		No
No internet service	...	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	20.00		No
No internet service	...	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	20.25		No
No internet service	...	No internet service	No internet service	No internet service	No internet service	One year	Yes	Mailed check	19.70		No
No internet service	...	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	19.85		No
							Yes	Bank transfer (automatic)	61.90		No
							No	Mailed check	73.35		No
							No	Credit card (automatic)	56.05		No
							Yes	Bank transfer (automatic)	52.55		No

Sorting 했을 때, **Total Charges** 열에 결측치가 있는 것을 확인

DATA EDA 및 시각화

```
raw_data['TotalCharges'] = pd.to_numeric(raw_data['TotalCharges'], errors='coerce')
raw_data.dropna(inplace = True)
data_df = raw_data.iloc[:,1:]
data_df
```

OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
Yes	No	No	No	No	Month-to-month	Yes	Electronic check	29.85	29.85	No
No	Yes	No	No	No	One year	No	Mailed check	56.95	1889.50	No
Yes	No	No	No	No	Month-to-month	Yes	Mailed check	53.85	108.15	Yes
No	Yes	Yes	No	No	One year	No	Bank transfer (automatic)	42.30	1840.75	No
No	No	No	No	No	Month-to-month	Yes	Electronic check	70.70	151.65	Yes

숫자형 데이터로 변환 + “coerce”로 변환이 안 되는 건 결측치로 변환
Dropna로 결측치 제거
iloc 로 customerID 열 제거

DATA EDA 및 시각화

In [24]:

```
info_ko={'customerID':'Customer ID',
      'gender':'성별(Male, Female)',
      'SeniorCitizen':'고령자인 고객 (1, 0) ',
      'Partner':'결혼 여부 (Yes, No)',
      'Dependents':'부양가족 여부 (Yes, No)',
      'tenure':'서비스 가입 개월 수',
      'PhoneService':'고객의 전화 서비스 여부 (Yes, No)',
      'MultipleLines':'여러 전화 회선을 보유하고 있는지 여부 (Yes, No, No phone service)',
      'InternetService':' 인터넷 서비스 공급자 (DSL, Fiber optic, No)',
      'OnlineSecurity':'온라인 보안 여부 (Yes, No, No internet service)',
      'OnlineBackup':'온라인 백업 여부 (Yes, No, No internet service)',
      'DeviceProtection':'기기 보호 여부 (Yes, No, No internet service)',
      'TechSupport':'인터넷 기술 지원 여부 (Yes, No, No internet service)',
      'StreamingTV':'스트리밍 TV를 가지고 있는지 여부 (Yes, No, No internet service)',
      'StreamingMovies':'스트리밍 영화가 있는지 여부 (Yes, No, No internet service)',
      'Contract':'계약기간 옵션 (월간, One year, Two year)',
      'PaperlessBilling':'온라인 청구서 사용 여부 (Yes, No)',
      'PaymentMethod':'결제수단 (전자수표, 우편수표, Bank transfer (automatic), Credit card (automatic))',
      'MonthlyCharges':'매월 청구된 금액',
      'TotalCharges':'청구된 총 금액',
      'Churn':'이탈여부 (Yes or No)'}

info_ko_df=pd.DataFrame([info_ko])
info_ko_df
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection
0	Customer ID	성별 (Male, Female)	고령자인 고객 (1, 0)	결혼 여부 (Yes, No)	부양가족 여부 (Yes, No)	서비스 가입 개월 수	고객의 전화 서비스 여부 (Yes, No)	여러 전화 회선을 보유하고 있는지 여부 (Yes, No, No phone service)	인터넷 서비스 공급자 (DSL, Fiber optic, No)	온라인 보안 여부 (Yes, No, No internet service)	...	기기 보호 여부 (Yes, No, No internet service)

1 rows × 21 columns

Column별 의미 & 값 후보군 분석

Feature engineering을 위해 꼭 필요
데이터의 이해를 돕는 과정
셀 데이터의 타입을 **str**에서 **int**로 변환하기 위해
어떤 값들을 포함하고 있는지 알아야 함.
추가 보완을 위해 더 필요한 데이터가 무엇인지,
혹은 없어도 괜찮은 데이터는 무엇인지 알 수
있음

DATA EDA 및 시각화

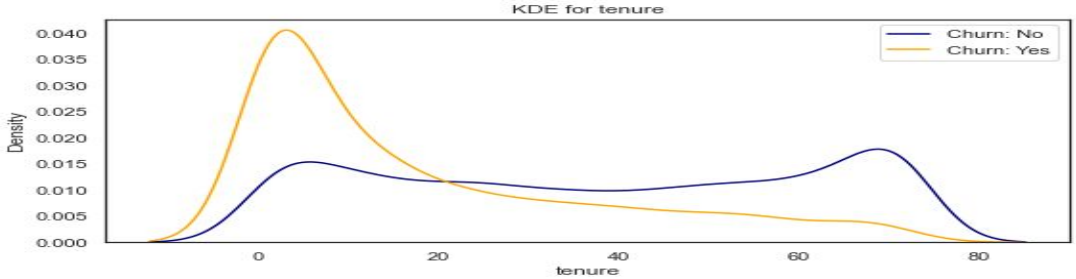
```
#Convertion the predictor variable in a binary numeric variable
data_df.replace(to_replace='Yes', value=1, inplace=True)
data_df.replace(to_replace='No',  value=0, inplace=True)
#gender
data_df.replace(to_replace='Male',  value=0, inplace=True)
data_df.replace(to_replace='Female',  value=1, inplace=True)
#MultipleLines
data_df.replace(to_replace='No phone service',  value=2, inplace=True)
#InternetService
data_df['InternetService'].replace(to_replace='Fiber optic',  value=1, inplace=True)
data_df['InternetService'].replace(to_replace='DSL',  value=2, inplace=True)
#OnlineSecurity,OnlineBackup,DeviceProtection,TechSupport,StreamingTV,StreamingMovies
data_df.replace(to_replace='No internet service',  value=2, inplace=True)
#Contract
data_df['Contract'].replace(to_replace='Month-to-month',  value=0, inplace=True)
data_df['Contract'].replace(to_replace='One year',  value=1, inplace=True)
data_df['Contract'].replace(to_replace='Two year',  value=2, inplace=True)

#PaymentMethod
data_df['PaymentMethod'].replace(to_replace='Electronic check',  value=0, inplace=True)
data_df['PaymentMethod'].replace(to_replace='Mailed check',  value=1, inplace=True)
data_df['PaymentMethod'].replace(to_replace='Bank transfer (automatic)',  value=2, inplace=True)
data_df['PaymentMethod'].replace(to_replace='Credit card (automatic)',  value=3, inplace=True)

data_df.head(10)
```

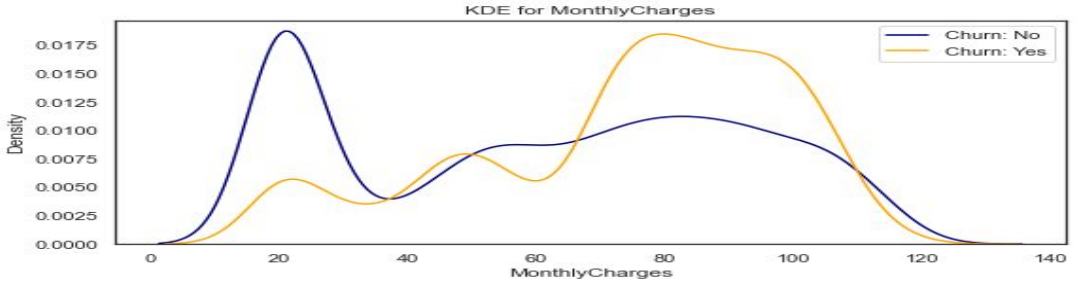
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupp
0	1	0	1	0	1	0	2	2	0	1	0	
1	0	0	0	0	34	1	0	2	1	0	1	
2	0	0	0	0	2	1	0	2	1	1	0	
3	0	0	0	0	45	0	2	2	1	0	1	
4	1	0	0	0	2	1	0	1	0	0	0	
5	1	0	0	0	8	1	1	1	0	0	1	
6	0	0	0	1	22	1	1	1	0	1	0	
7	1	0	0	0	10	0	2	2	1	0	0	
8	1	0	1	0	28	1	1	1	0	0	1	
9	0	0	0	1	62	1	0	2	1	1	0	

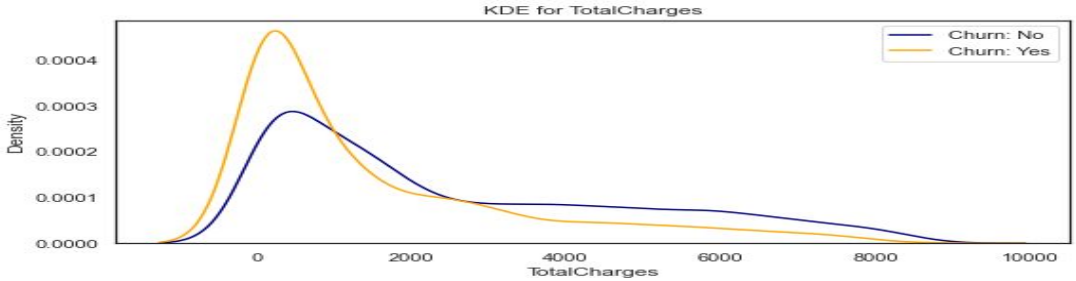
셀 데이터 타입 -> 숫자로 변환

모든 셀 : Yes, No -> 1, 0
Gender : Male, Female -> 0, 1
Multiple Lines : No phone service -> 2
Internet Service : Fiber optic, DSL -> 1, 2
Online Security, Online Backup, Device Protection, TechSupport, StreamingTV, StreamingMovies : No internet service -> 2
Contract : Month to Month, One Year, Two Year -> 0, 1, 2
Payment Method : Electronic Check, Mailed Check, Bank transfer, Credit Card -> 0, 1, 2, 3

DATA EDA 및 시각화



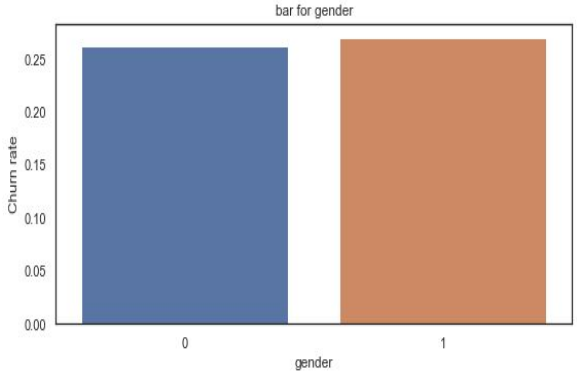


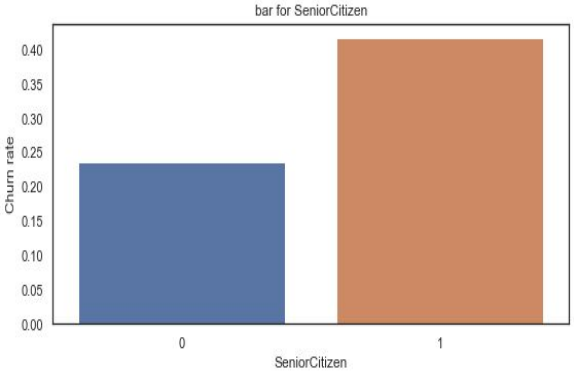


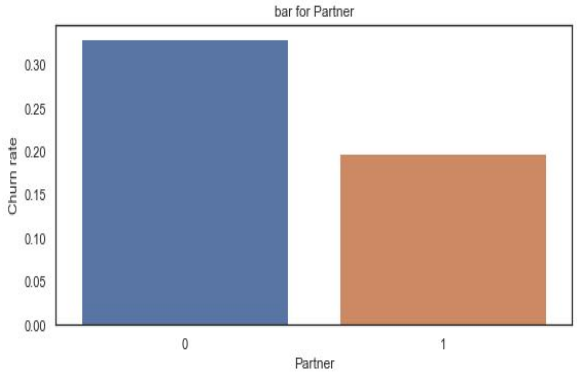
Numerical features

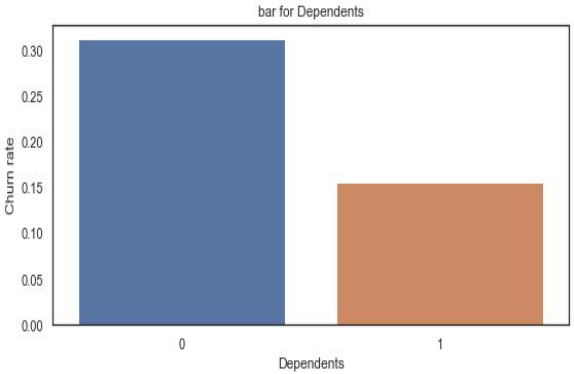
* 신규 고객 이탈률이 높음.
* MonthlyCharges가 높은 고객의 이탈률이 높음
* Tenure 와 MonthlyCharges는 중요도가 높은 항목

DATA EDA 및 시각화









Categorical features

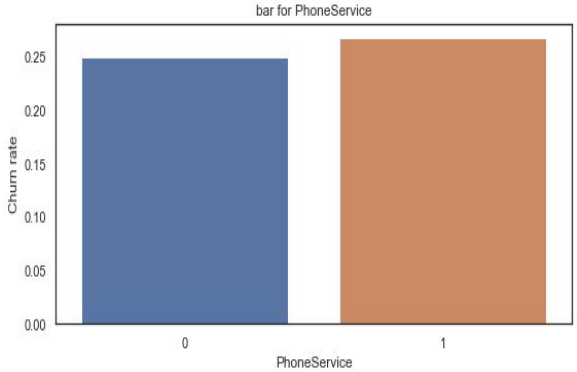
'gender':'성별(Male, Female)'

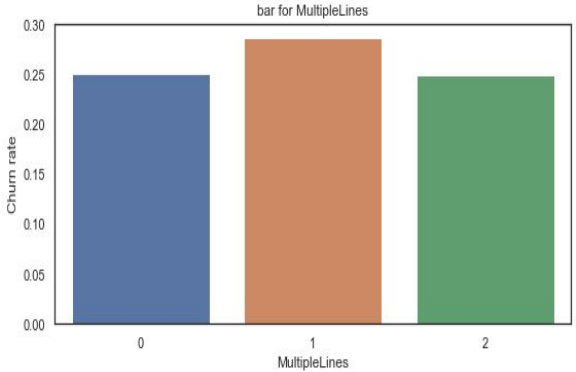
'SeniorCitizen':'고령자인 고객 (1, 0) '

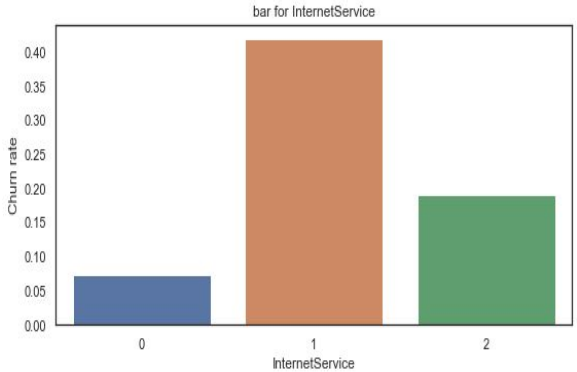
'Partner':'결혼 여부 (No, Yes)'

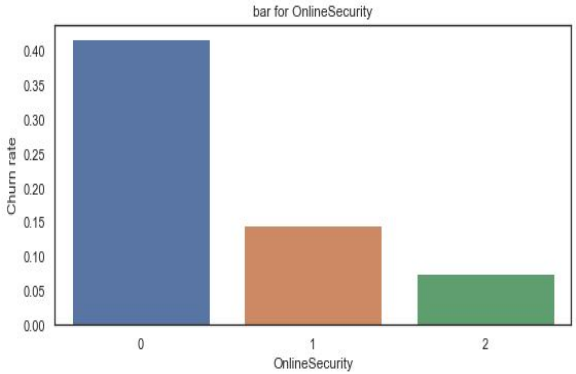
'Dependents':'부양가족 여부 (No, Yes)'

DATA EDA 및 시각화





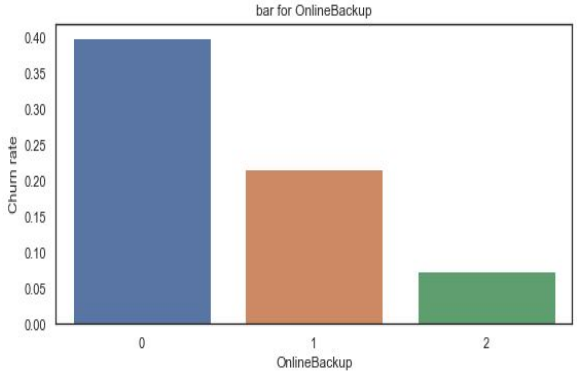


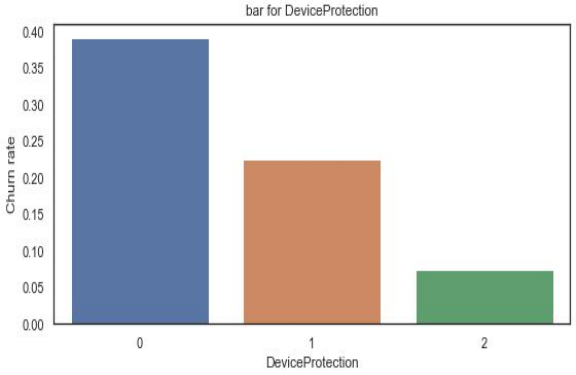


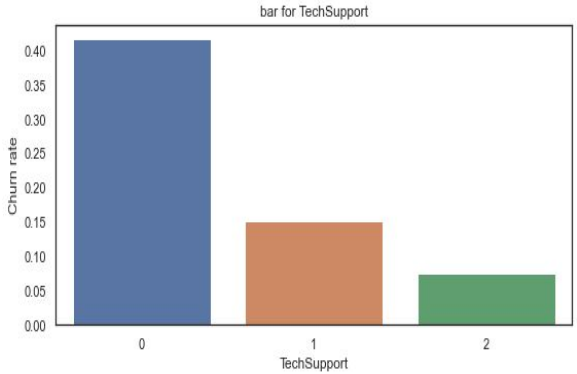
Categorical features

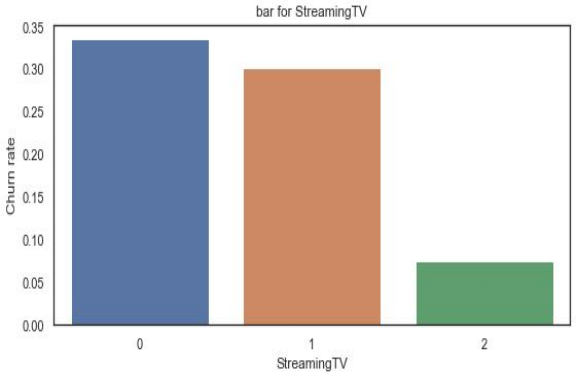
'PhoneService':'고객의 전화 서비스 여부 (No, Yes)'
'MultipleLines':'여러 전화 회선을 보유하고 있는지 여부 (No, Yes, No phone service)'
'InternetService':' 인터넷 서비스 공급자 (No, Fiber optic, DSL)'
'OnlineSecurity':'온라인 보안 여부 (No, Yes, No internet service)'

DATA EDA 및 시각화





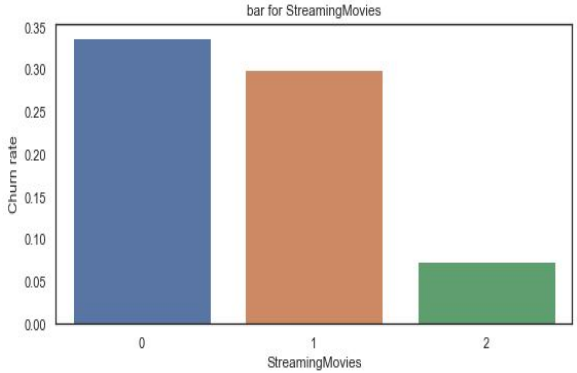


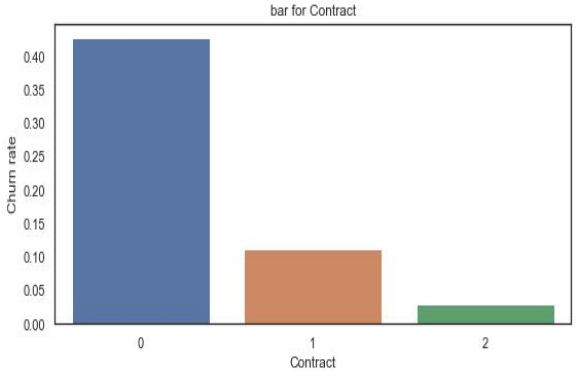


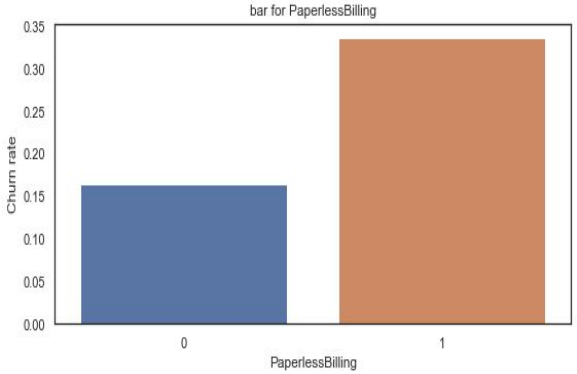
Categorical features

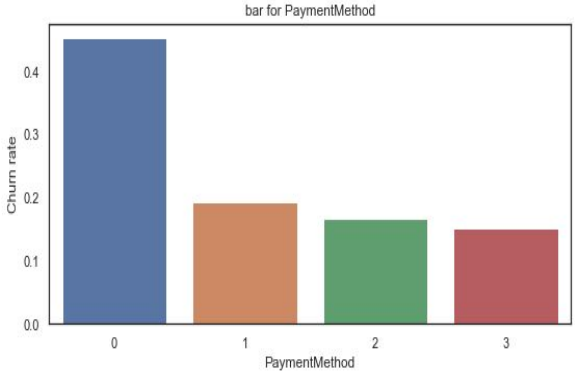
'OnlineBackup':'온라인 백업 여부 (No, Yes, No internet service)'
'DeviceProtection':'기기 보호 여부 (No, Yes, No internet service)'
'TechSupport':'인터넷 기술 지원 여부 (No, Yes, No internet service)'
'StreamingTV':'스트리밍 TV를 가지고 있는지 여부 (No, Yes, No internet service)'

DATA EDA 및 시각화





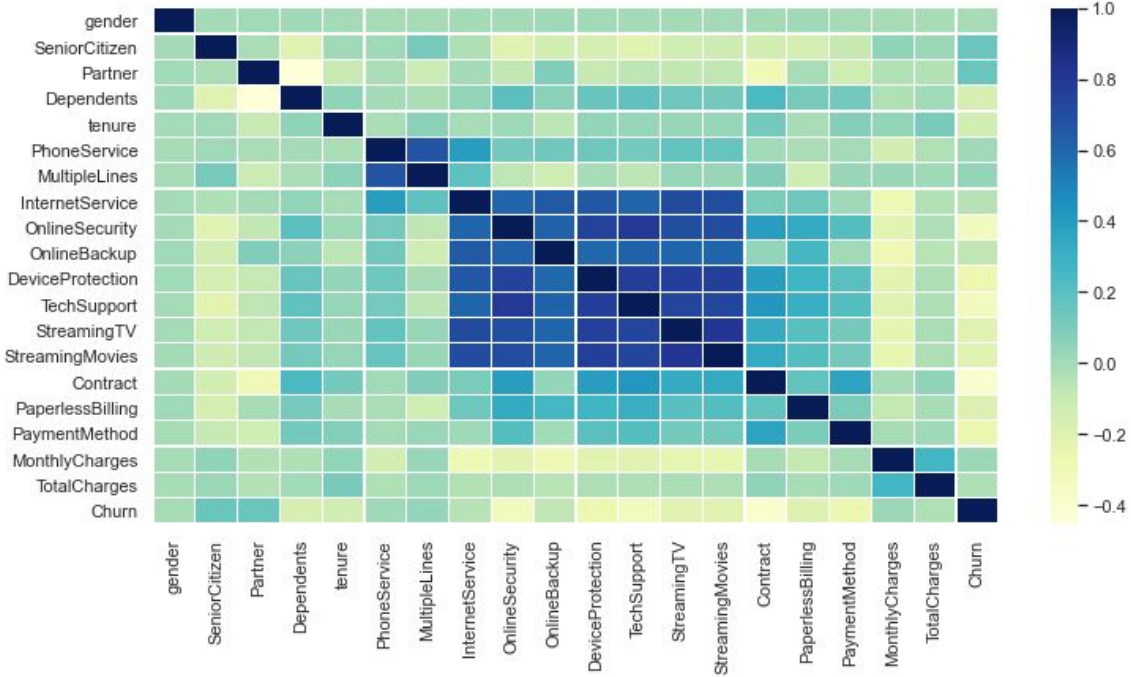




Categorical features

'StreamingMovies':'스트리밍 영화가 있는지 여부 (No, Yes, No internet service)'
'Contract':'계약기간 옵션 (월간, One year, Two year)'
'PaperlessBilling':'온라인 청구서 사용 여부 (No, Yes)'
'PaymentMethod':'결제수단 (전자수표, 우편수표, Bank transfer (automatic), Credit card (automatic))'

DATA EDA 및 시각화

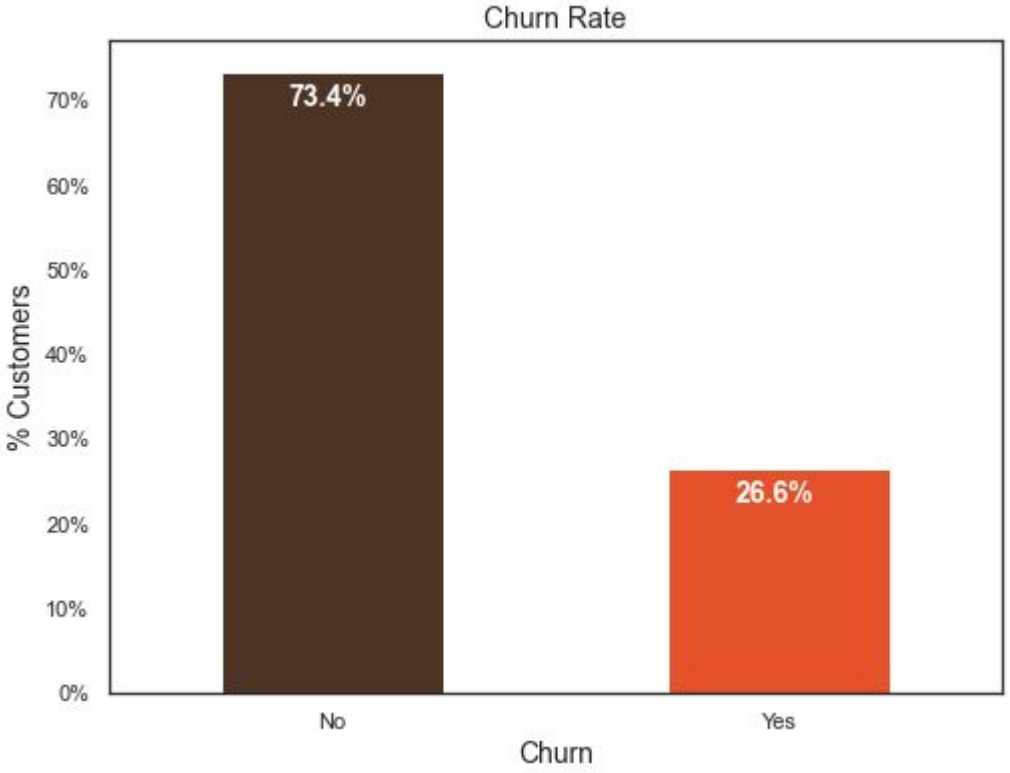


Correlation between features

앞에서 봤던대로 Churn데이터와 직접적인 연관성이 나타나는 항목은 적음.

ML & DL을 이용해서 항목간의 숨겨진 관계 파악이 중요

DATA EDA 및 시각화



Class imbalance between Churn Status

Churn No 데이터가 Churn Yes 데이터에 비해 많음.

Undersampling or oversampling 등을 통해 클래스 불균형을 해결 가능

DATA EDA 및 시각화

```
from imblearn.over_sampling import SMOTENC
col_list = list(range(45))
numerical_col = [1,2,3]
categorical_col = [x for x in col_list if x not in numerical_col]
smote_sample = SMOTENC(categorical_features= categorical_col,random_state=0)
X_train_over, y_train_over = smote_sample.fit_resample(X_train, y_train.ravel())
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', X.shape, y.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ', X_train_over.shape, y_train_over.shape)
print('SMOTE 적용 후 레이블 값 분포: ₩n', pd.Series(y_train_over).value_counts())
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트:  (7032, 45) (7032,)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트:  (7216, 45) (7216,)
SMOTE 적용 후 레이블 값 분포:
 0    3608
1    3608
dtype: int64
```

Class imbalance between Churn Status

SMOTENC는 categorical과 numerical feature가 함께 있을 때 둘을 구분하여 data를 oversampling 해줌
sampling_stratagy=1 등을 이용하여 oversampling 할 data의 양 조절 도 가능

약 3% 정도의 accuracy 향상 효과가 나옴.

데이터의 양이 많은 것이 아니기 때문에 undersampling은 사용하지 않음

Machine Learning Model

AutoML-Pycaret

-Accuracy 기반 Top 3 model 비교

-전처리 전 : AdaBoost, GradientBoosting, Logistic Regression

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ada	Ada Boost Classifier	0.8087	0.8493	0.5412	0.6652	0.5957	0.4724	0.4774	5.357
gbc	Gradient Boosting Classifier	0.8069	0.8503	0.5093	0.6728	0.5787	0.4568	0.4648	17.205
lr	Logistic Regression	0.8049	0.8465	0.5357	0.6567	0.5892	0.4631	0.4678	38.001
rf	Random Forest Classifier	0.7988	0.8303	0.4596	0.6687	0.5440	0.4206	0.4332	6.893
ridge	Ridge Classifier	0.7980	0.0000	0.4845	0.6549	0.5555	0.4288	0.4378	2.895

-최종 Accuracy : 0.7941

Machine Learning Model

AutoML-Pycaret

전처리 전: Blending

```
blended = blend_models(estimator_list=top_3_models,
                       fold=10, # default
                       optimize='Accuracy',
                       method = 'hard')
```

-Pycaret 기반 Top 3 model
-Hard Voting(Majority Voting)

-최종 Accuracy : 0.8031

```
VotingClassifier(estimators=[('ada',
                              AdaBoostClassifier(algorithm='SAMME.R',
                                                 base_estimator=None,
                                                 learning_rate=1.0,
                                                 n_estimators=50,
                                                 random_state=9)),
                             ('gbc',
                              GradientBoostingClassifier(ccp_alpha=0.0,
                                                         criterion='friedman_mse',
                                                         init=None,
                                                         learning_rate=0.1,
                                                         loss='deviance',
                                                         max_depth=3,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurit...
                                                         validation_fraction=0.1,
                                                         verbose=0,
                                                         warm_start=False)),
                             ('lr',
                              LogisticRegression(C=1.0, class_weight=None,
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=1000,
                                                 multi_class='auto',
                                                 n_jobs=None, penalty='l2',
                                                 random_state=9, solver='lbfgs',
                                                 tol=0.0001, verbose=0,
                                                 warm_start=False))],
                 flatten_transform=True, n_jobs=-1, verbose=False,
                 voting='hard', weights=None)
```

Machine Learning Model

AutoML-Pycaret

-Categorical 변수에 대해 전처리 진행
-전처리 후 : GradientBoosting, AdaBoost, LightGBM

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.8011	0.8453	0.5351	0.6616	0.5915	0.4620	0.4667	0.628
ada	Ada Boost Classifier	0.7991	0.8411	0.5344	0.6569	0.5890	0.4579	0.4624	0.257
lightgbm	Light Gradient Boosting Machine	0.7991	0.8343	0.5494	0.6506	0.5949	0.4628	0.4662	0.172
lr	Logistic Regression	0.7974	0.8414	0.5457	0.6477	0.5921	0.4587	0.4619	0.128
ridge	Ridge Classifier	0.7962	0.0000	0.5216	0.6522	0.5794	0.4472	0.4522	0.020

-최종 Accuracy:0.8038

Machine Learning Model

AutoML-Pycaret

전처리 후: Blending

```
blended = blend_models(estimator_list=top_3_models,
                       fold=10, # default
                       optimize='Accuracy',
                       method = 'hard')
```

-전처리 전과 동일하게 진행

-최종 Accuracy : 0.8336

```
VotingClassifier(estimators=[('gbc',
                              GradientBoostingClassifier(ccp_alpha=0.0,
                                                         criterion='friedman_mse',
                                                         init=None,
                                                         learning_rate=0.1,
                                                         loss='deviance',
                                                         max_depth=3,
                                                         max_features=None,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=1,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         n_estimators=100,
                                                         n_iter_no_change=No...
                                                  importance_type='split',
                                                  learning_rate=0.1, max_depth=-1,
                                                  min_child_samples=20,
                                                  min_child_weight=0.001,
                                                  min_split_gain=0.0,
                                                  n_estimators=100, n_jobs=-1,
                                                  num_leaves=31, objective=None,
                                                  random_state=9, reg_alpha=0.0,
                                                  reg_lambda=0.0, silent='warn',
                                                  subsample=1.0,
                                                  subsample_for_bin=200000,
                                                  subsample_freq=0))],
                 flatten_transform=True, n_jobs=-1, verbose=False,
                 voting='hard', weights=None)
```

Machine Learning Model

AutoML-H2O

```
# data_df을 8:2로 나눈다
train, valid = train_test_split(data_df,
                                test_size=0.2,
                                shuffle=True)
h2o_train = h2o.H2OFrame(train)
h2o_valid = h2o.H2OFrame(valid)
# For binary classification, response should be a factor
h2o_train[y] = h2o_train[y].asfactor()
h2o_valid[y] = h2o_valid[y].asfactor()
## Run AutoML for 120 seconds
aml = H2OAutoML(max_runtime_secs=max_runtime_secs,exclude_algos=['StackedEnsemble'])
aml.train(x = x, y = y, training_frame=h2o_train, leaderboard_frame=h2o_valid)
#결과 출력
leaderboard = aml.leaderboard
performance = aml.leader.model_performance(h2o_valid)  # (Optional) Evaluate performance on a test set
model_id   =aml.leader.model_id  # 최고 모델 명
accuracy   =performance.accuracy()  # 정확도
precision  =performance.precision()  # precision
recall     =performance.recall()  # recall
F1         =performance.F1()  # f1
auc        =performance.auc()  # auc
variable_importance=aml.leader.varimp()  # 중요한 입력 변수
```

-train/test split 구분:
성능 결과에 큰 영향x.
-전처리 여부 구분
-Metric 기반
-tns,tps

Machine Learning Model

AutoML-H2O

model_id	auc	logloss	aucpr	mean_per_class_error	rmse	mse	training_time_ms	predict_time_per_row_ms	algo
GLM_1_AutoML_1_20220504_12047	0.845241	0.424869	0.672631	0.229945	0.370317	0.137135	629	0.014395	GLM
GBM_2_AutoML_1_20220504_12047	0.843179	0.424666	0.677268	0.233073	0.370726	0.137438	695	0.01683	GBM
GBM_5_AutoML_1_20220504_12047	0.842968	0.425319	0.669887	0.235041	0.370883	0.137555	606	0.016922	GBM
XGBoost_grid_1_AutoML_1_20220504_12047_model_1	0.841317	0.426934	0.680737	0.242416	0.371809	0.138242	877	0.008211	XGBoost
GBM_1_AutoML_1_20220504_12047	0.841108	0.427031	0.668891	0.237068	0.371543	0.138045	2191	0.019753	GBM
GBM_grid_1_AutoML_1_20220504_12047_model_2	0.837918	0.451886	0.665559	0.23399	0.381456	0.145509	238	0.009322	GBM
XGBoost_grid_1_AutoML_1_20220504_12047_model_3	0.836956	0.431559	0.669878	0.242336	0.373662	0.139623	301	0.007649	XGBoost
DeepLearning_1_AutoML_1_20220504_12047	0.83654	0.43574	0.661341	0.238773	0.37532	0.140865	966	0.010124	DeepLearnin
XRT_1_AutoML_1_20220504_12047	0.833048	0.438078	0.654734	0.242639	0.378187	0.143025	1476	0.027443	DRF
GBM_grid_1_AutoML_1_20220504_12047_model_1	0.831981	0.438044	0.664382	0.245211	0.377738	0.142686	931	0.036974	GBM

-ML model>DL model

Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.3471938319963487:

		0	1	Error	Rate
0	0	813.0	206.0	0.2022	(206.0/1019.0)
1	1	100.0	288.0	0.2577	(100.0/388.0)
2	Total	913.0	494.0	0.2175	(306.0/1407.0)

-Confusion Matrix(전처리 후)

-전처리 전/후 변화

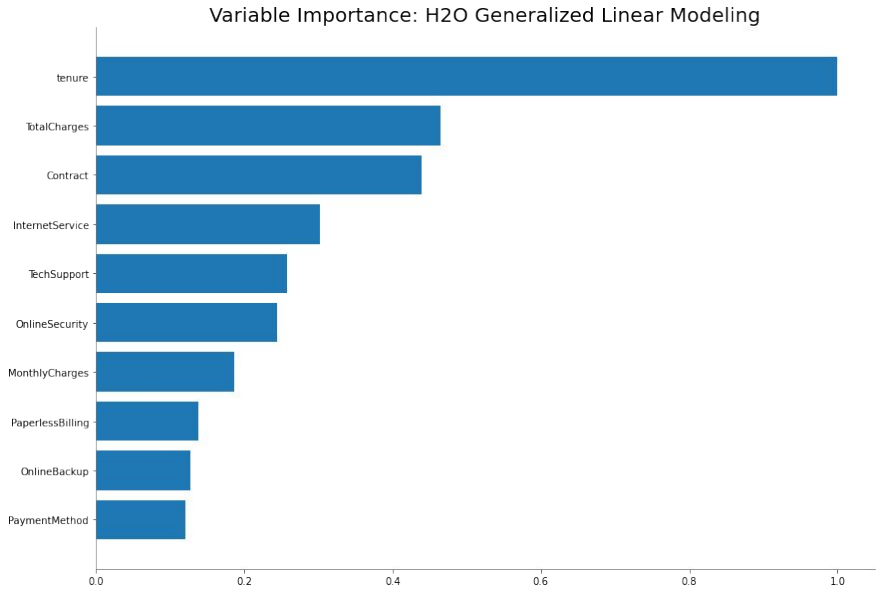
Accuracy: 0.75->0.78

AUC: 0.85->0.85

Tps,tms에서는 큰 차이x.

Machine Learning Model

AutoML-H2O

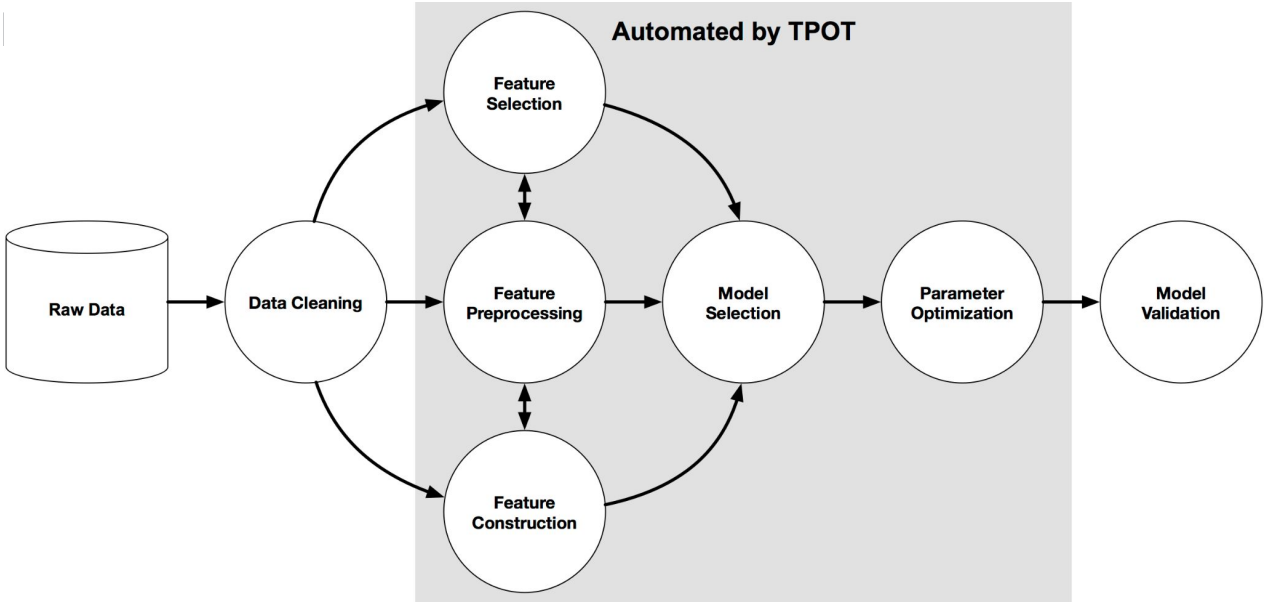


-Feature Importance

(전처리 전): Contract, tenure, TotalCharges ↔ (전처리 후) tenure,TotalCharges, Contract
Split 비율 변화는 feature importance에 영향을 끼치지 않음

Machine Learning Model

AutoML-TPOT



Machine Learning Model

AutoML-TPOT

```
tpot =  TPOTClassifier(generations=4, population_size=10, verbosity=3)
tpot.fit(X_train, y_train)
print(tpot.score(X_test, y_test))
```

```
32 operators have been imported by TPOT.
Optimization Progress: 100% 50/50 [01:49<00:00, 2.35s/pipeline]
```

-TPOT 기준 Best Model(Pipeline)

```
tpot.fitted_pipeline_
```

```
Pipeline(steps=[('extratreesclassifier',
                 ExtraTreesClassifier(criterion='entropy', max_features=0.3,
                                      min_samples_leaf=12, min_samples_split=13,
                                      random_state=50))])
```

-최종 Accuracy: 0.8003

Machine Learning Model

AutoML-Autogluon

```
    x_train, x_test = features.iloc[train_index], features.iloc[test_index]
    y_train, y_test = label.iloc[train_index], label.iloc[test_index]
```

```
label_column = 'Churn'
predictor = TabularPredictor(label=label_column).fit(x_train)
y_pred = predictor.predict(x_test)
perf = predictor.evaluate_predictions(y_true=y_test, y_pred=y_pred, auxiliary_metrics=True)
```

```
Evaluation: accuracy on test data: 0.8044096728307255
Evaluations on test data:
{
    "accuracy": 0.8044096728307255,
    "balanced_accuracy": 0.709921547900344,
    "mcc": 0.46390947666257626,
    "f1": 0.5801526717557252,
    "precision": 0.6761565836298933,
    "recall": 0.5080213903743316
}
```

-최종 Accuracy: 0.8044

Machine Learning Model

AutoML-Autogluon

	model	score_val	pred_time_val	fit_time	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order
0	CatBoost	0.806394	0.003057	1.063194	0.003057	1.063194	1	True	7
1	WeightedEnsemble_L2	0.806394	0.004520	1.627586	0.001463	0.564392	2	True	14
2	NeuralNetTorch	0.793961	0.011151	4.928169	0.011151	4.928169	1	True	12
3	NeuralNetFastAI	0.786856	0.026681	5.626101	0.026681	5.626101	1	True	10
4	LightGBMXT	0.785080	0.006076	0.661889	0.006076	0.661889	1	True	3
5	XGBoost	0.783304	0.008326	0.684035	0.008326	0.684035	1	True	11
6	LightGBM	0.781528	0.005372	0.608989	0.005372	0.608989	1	True	4
7	LightGBMLarge	0.781528	0.006864	1.014153	0.006864	1.014153	1	True	13
8	RandomForestGini	0.776199	0.204711	1.540009	0.204711	1.540009	1	True	5
9	RandomForestEntr	0.772647	0.204262	1.929509	0.204262	1.929509	1	True	6
10	ExtraTreesEntr	0.769094	0.204852	1.398498	0.204852	1.398498	1	True	9
11	ExtraTreesGini	0.761989	0.204616	1.230557	0.204616	1.230557	1	True	8
12	KNeighborsUnif	0.740675	0.105725	0.020509	0.105725	0.020509	1	True	1
13	KNeighborsDist	0.724689	0.105317	0.021532	0.105317	0.021532	1	True	2

-최종 Accuracy: 0.8044

Machine Learning Model

Autogluon-xfeat

-xFeat feature engineering : Count를 기반으로 Scaling, 관련성이 있는 변수 결합, Churn과 상관관계가 낮은 열 제거, 상관관계가 너무 높은 열 제거

```
target = 'Churn'
categorical_columns = ['gender','SeniorCitizen','Partner','Dependents','PhoneService','MultipleLines','InternetService','OnlineS
numerical_columns = ['tenure','MonthlyCharges','TotalCharges']
```

```
from xfeat import CountEncoder
encoder = CountEncoder(input_cols=categorical_columns)
x_train = encoder.fit_transform(x_train)
x_test = encoder.transform(x_test)
```

```
from xfeat import ConcatCombination
encoder = ConcatCombination(input_cols=categorical_columns,
                            drop_origin=False,
                            r=2)
x_train = encoder.fit_transform(x_train)
x_test = encoder.transform(x_test) #유관변수 결합
```

StreamingTV-Contract, StreamingMovie-PaperlessBilling 등 결합 열 생성

Machine Learning Model

Autogluon-xfeat

```
from xfeat import TargetEncoder
encoder = TargetEncoder(input_cols=categorical_columns,
                        target_col=target)
x_train = encoder.fit_transform(x_train)
x_test = encoder.transform(x_test)
#타겟 기반 인코딩#타겟 기반으로 중요도 출력
```

```
x_train.head()
```

InternetService_te	OnlineSecurity_te	OnlineBackup_te	DeviceProtection_te	TechSupport_te	StreamingTV_te	StreamingMovies_te	Contract_te
0.194622	0.415154	0.218608	0.393114	0.413793	0.327414	0.333333	0.419497
0.185771	0.142412	0.405996	0.226268	0.423782	0.340155	0.347238	0.108021
0.186062	0.134213	0.207707	0.394201	0.415740	0.330337	0.340486	0.424740
0.185771	0.142412	0.405996	0.226268	0.147516	0.340155	0.347238	0.108021
0.415742	0.422991	0.403357	0.396769	0.425011	0.339326	0.341709	0.434216

결합,

-중요도 확률 출력

Machine Learning Model

Autogluon-xfeat

```
#중복 열 등 추가 feature engineering
#상관 관계가 너무 높은 열 제거
from xfeat.pipeline import Pipeline
from xfeat.selector import DuplicatedFeatureEliminator
from xfeat.selector import ConstantFeatureEliminator
from xfeat.selector import SpearmanCorrelationEliminator
def get_feature_selector():
    return Pipeline(
        [
            DuplicatedFeatureEliminator(),
            ConstantFeatureEliminator(),
            SpearmanCorrelationEliminator(threshold=0.9),
        ]
    )

input_columns = [x for x in x_train.columns if x != target]
selector = get_feature_selector()
df_reduced = selector.fit_transform(x_train[input_columns])
print("Selected columns: {}".format(df_reduced.columns.tolist()))
x_train = x_train[df_reduced.columns.tolist()]
x_test = x_test[df_reduced.columns.tolist()]
```

-Selected columns: ['gender','SeniorCitizen','genderPartner_combi'] 등 총 102개의 열 선택

Machine Learning Model

AutoML-Autogluon(xFeat)

	model	score_val	pred_time_val	fit_time	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order
0	WeightedEnsemble_L2	0.815275	0.499355	21.819319	0.001615	0.570437	2	True	14
1	CatBoost	0.795737	0.004778	2.044024	0.004778	2.044024	1	True	7
2	NeuralNetFastAI	0.792185	0.060185	5.615905	0.060185	5.615905	1	True	10
3	LightGBM	0.788632	0.006621	1.025185	0.006621	1.025185	1	True	4
4	NeuralNetTorch	0.788632	0.022705	7.364331	0.022705	7.364331	1	True	12
5	XGBoost	0.786856	0.011350	1.455034	0.011350	1.455034	1	True	11
6	LightGBMXT	0.779751	0.006861	1.054260	0.006861	1.054260	1	True	3
7	LightGBMLarge	0.774423	0.006724	2.111131	0.006724	2.111131	1	True	13
8	RandomForestEntr	0.770870	0.204861	2.663498	0.204861	2.663498	1	True	6
9	ExtraTreesGini	0.763766	0.204873	1.766218	0.204873	1.766218	1	True	8
10	RandomForestGini	0.760213	0.204894	2.269772	0.204894	2.269772	1	True	5
11	ExtraTreesEntr	0.760213	0.204896	2.064662	0.204896	2.064662	1	True	9
12	KNeighborsUnif	0.737123	0.180379	0.026644	0.180379	0.026644	1	True	1
13	KNeighborsDist	0.733570	0.180230	0.026787	0.180230	0.026787	1	True	2

-최종 Accuracy: 0.8022

Machine Learning Model

신경망 아키텍쳐 탐색(NAS)

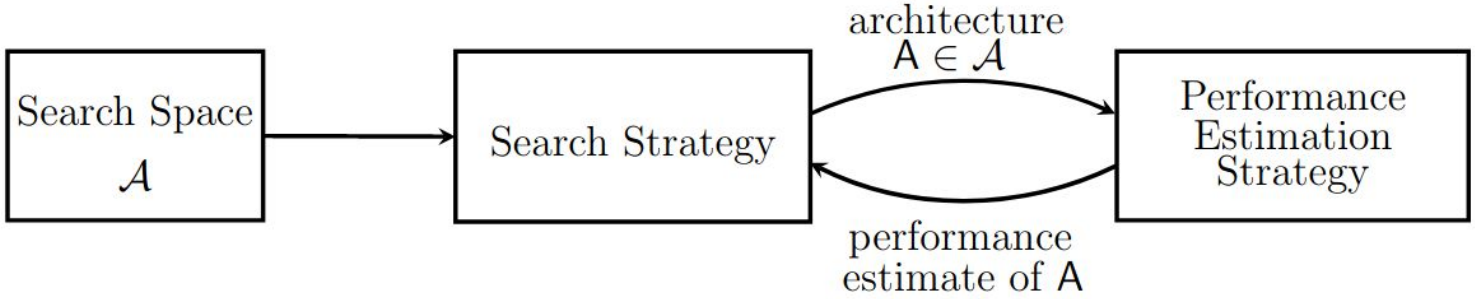


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

Machine Learning Model

AutoML-AutoKeras(NAS)

```
clf=ak.StructuredDataClassifier(overwrite=True,max_trials=5)
clf.fit(
    x_train,
    y_train,
    # Split the training data and use the last 15% as validation data.
    validation_split=0.15,
    epochs=10,
)
# Predict with the best model.
predicted_y = clf.predict(x_test)
# Evaluate the best model with testing data.
print(clf.evaluate(x_test, y_test))
```

```
Trial 5 Complete [00h 00m 13s]
val_accuracy: 0.805084764957428
```

-최종 Accuracy: 0.8050

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 19)]              0

 multi_category_encoding (Mu  (None, 19)               0
 ltiCategoryEncoding)

 normalization (Normalizatio  (None, 19)               39
 n)

 dense (Dense)               (None, 32)                640

 re_lu (ReLU)                (None, 32)                0

 dense_1 (Dense)             (None, 16)                528

 re_lu_1 (ReLU)              (None, 16)                0

 dense_2 (Dense)             (None, 1)                 17

 classification_head_1 (Acti  (None, 1)                0
 vation)

=================================================================
Total params: 1,224
Trainable params: 1,185
Non-trainable params: 39
_________________________________________________________________
```

Machine Learning Model

Hyperparameter Tuning - Hyperopt vs Optuna

-Hyperopt

자동화 튜닝 프레임워크로, 베이지안 최적화 모델 기반

Objective Function, Domain Space, Optimization Algorithm

: 손실 함수, 하이퍼파라미터의 통계 분포, 최적화 알고리즘 파라미터 존재

-Optuna

하이퍼파라미터 자동 최적화 프레임워크

목적 함수,병렬 처리,시각화 존재

Study, Trial 과정 :

최적화 과정, 목적함수 시행 과정

Machine Learning Model

AutoML-Hyperopt

```
from hpsklearn import any_preprocessing
```

```
if __name__ == '__main__':
    estim=HyperoptEstimator(classifier=any_classifier("my_clf"),preprocessing=any_preprocessing("my_pre"),algo=tpe.suggest,
                            max_evals=100,trial_timeout=120)
    estim.fit(x_train,y_train)
    print(estim.score(x_test,y_test))
    print(estim.best_model())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid fe
  "X does not have valid feature names, but"
0.8108108108108109
{'learner': XGBClassifier(colsample_bylevel=0.7543036061136814,
              colsample_bytree=0.6481436619788232, gamma=0.002663488945763503,
              learning_rate=0.0004614315263122706, max_depth=8, missing=nan,
              n_estimators=5600, reg_alpha=0.00042121112275681703,
              reg_lambda=1.292371623637659, seed=3, subsample=0.865106726751074,
              use_label_encoder=False), 'preprocs': (Normalizer(norm='l1'),), 'ex_preprocs': ()}
```

-최종 Accuracy: 0.8108

Machine Learning Model

MLJAR

-Mljar-Supervised:

Table data(structured data) 기반의 기계학습 패키지.
Optuna,explain,perform,compete 방식이 존재
시각화에 용이한 패키지
-Optuna: optuna hyperparameter tuning 방식 적용
-explain: Default data setting
-perform: 5 fold CV, 활용가능성이 높은 모델 위주 선별
-compete: 10 fold CV, 모든 모델에 대해 비교

Machine Learning Model

AutoML-MLJAR(Optuna)

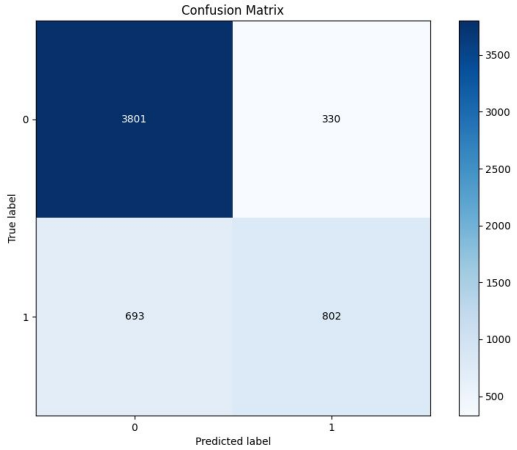
```
from supervised.automl import AutoML
automl = AutoML(mode="Optuna", optuna_time_budget=600,eval_metric='accuracy')
automl.fit(x_train, y_train)
```

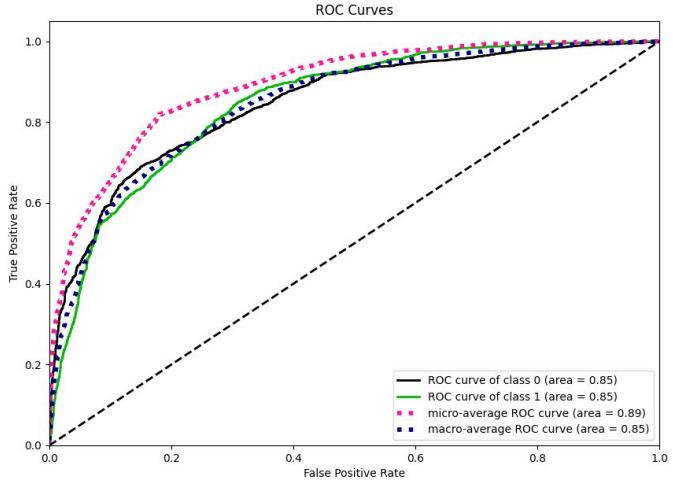
```
[I 2022-05-09 10:26:45,058] Trial 75 finished with value: 0.8046181172291297 and parameters: {
[I 2022-05-09 10:26:50,913] Trial 76 finished with value: 0.8028419182948491 and parameters: {
[I 2022-05-09 10:26:56,977] Trial 77 finished with value: 0.8046181172291297 and parameters: {
[I 2022-05-09 10:27:02,856] Trial 78 finished with value: 0.8081705150976909 and parameters: {
[I 2022-05-09 10:27:08,713] Trial 79 finished with value: 0.7992895204262878 and parameters: {
```

-Best Model: Ensemble_Stacked Model
-Accuracy: 0.8195

Machine Learning Model

AutoML-MLJAR(Optuna)





-Optuna(0.8195)>Hyperopt(0.8108)

Machine Learning Model

AutoML-MLJAR(explain)

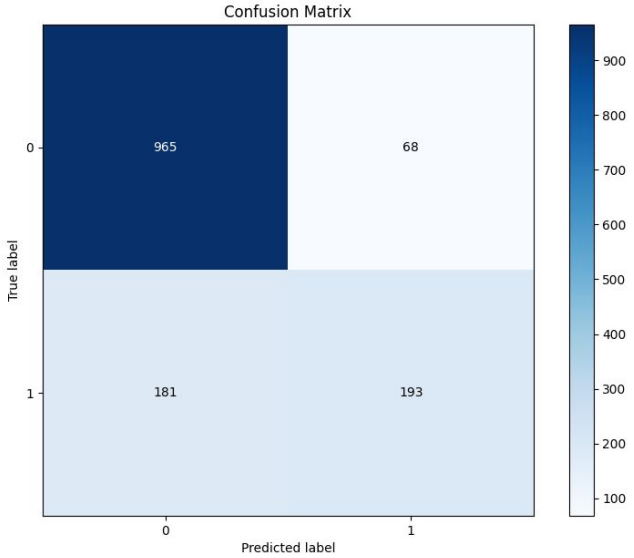
```
automl = AutoML(mode="Explain",eval_metric='accuracy')
automl.fit(x_train, y_train)
predictions = automl.predict(x_test)
```

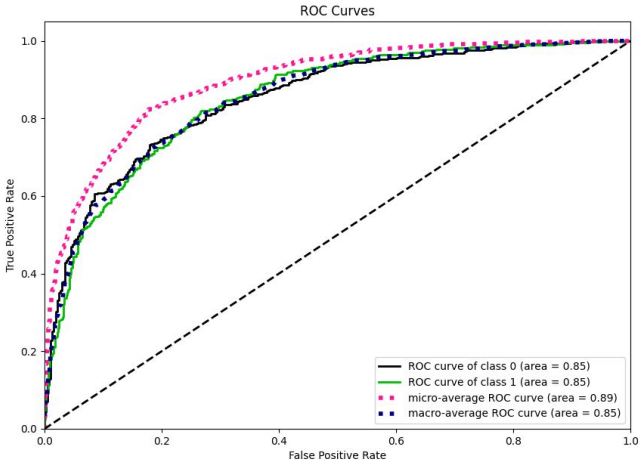
```
AutoML directory: AutoML_6
The task is binary_classification with evaluation metric accuracy
AutoML will use algorithms: ['Baseline', 'Linear', 'Decision Tree', 'Random Forest', 'Xgboost', 'Neural Network']
AutoML will ensemble available models
AutoML steps: ['simple_algorithms', 'default_algorithms', 'ensemble']
* Step simple_algorithms will try to check up to 3 models
1_Baseline accuracy 0.734186 trained in 5.37 seconds
2_DecisionTree accuracy 0.777541 trained in 13.6 seconds
3_Linear accuracy 0.816631 trained in 6.08 seconds
* Step default_algorithms will try to check up to 3 models
`feval` is deprecated, use `custom_metric` instead.  They have different behavior when custom objective is also used.
4_Default_Xgboost accuracy 0.808102 trained in 10.85 seconds
5_Default_NeuralNetwork accuracy 0.800284 trained in 3.35 seconds
6_Default_RandomForest accuracy 0.801706 trained in 13.44 seconds
* Step ensemble will try to check up to 1 model
Ensemble accuracy 0.823028 trained in 0.89 seconds
AutoML fit time: 71.48 seconds
AutoML best model: Ensemble
```

-Best Model: Ensemble
-Accuracy: 0.8238

Machine Learning Model

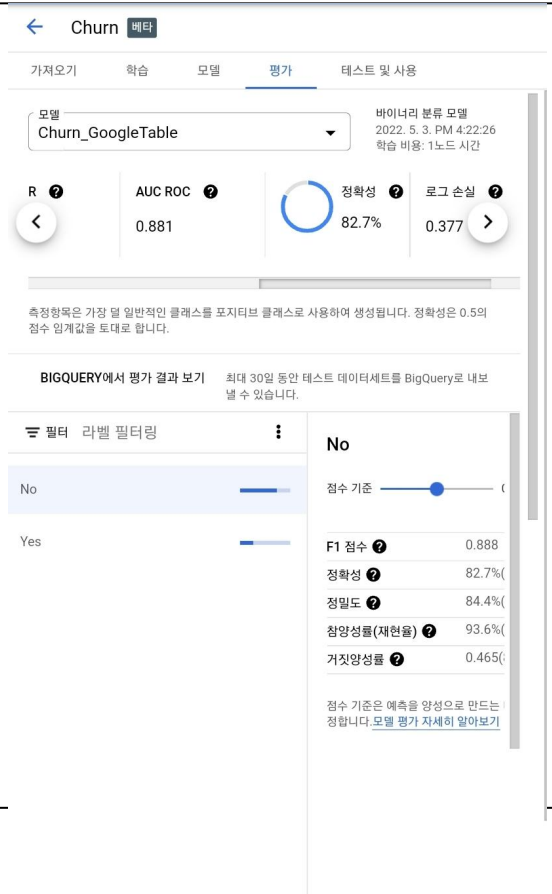
AutoML-MLJAR(Explain)





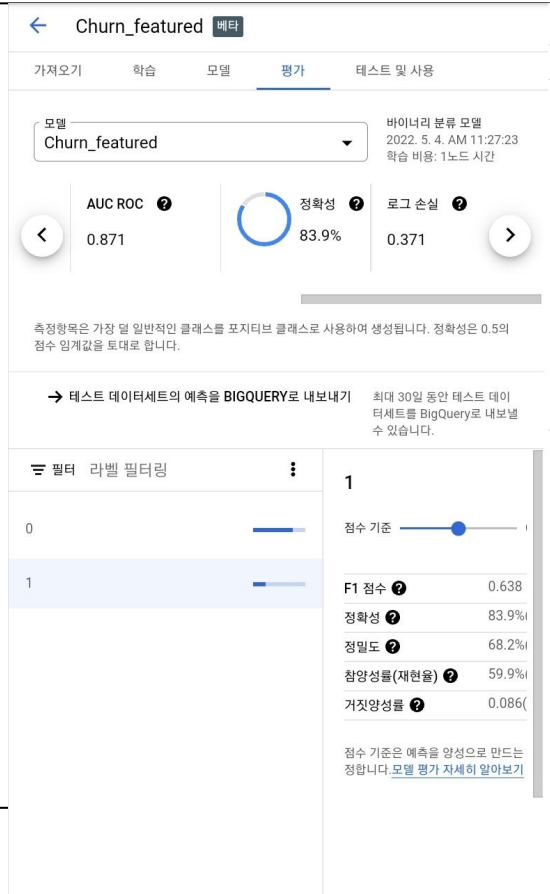
Machine Learning Model

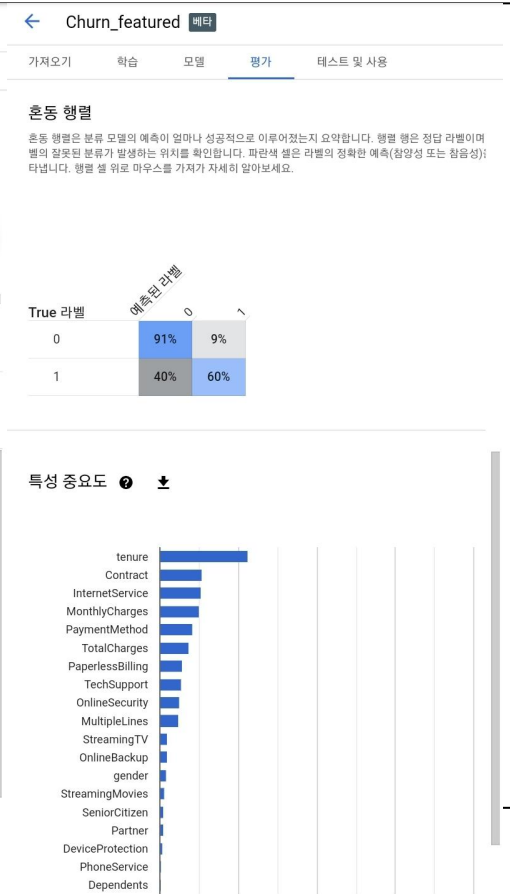
GoogleTable(전처리 전)



Machine Learning Model

GoogleTable(전처리 후)





Deep Learning Model (1) - 1

Basic Artificial Neural Network

```
ann = Sequential([
    Dense(units=12, activation='elu'),
    Dense(units=6, activation='elu'),
    Dense(units=3, activation='elu'),
    Dense(units=1, activation='sigmoid')
])
```

> Units, activation, initializer, dropout 적용해보며 accuracy 비교

```
1  ann_pred

array([[0.20155725],
       [0.11010385],
       [0.3502329 ],
       ...,
       [0.6141219 ],
       [0.16656831],
       [0.83087456]], dtype=float32)
```

1일 확률 -> 약 35%

< Sigmoid를 선택한 이유 >

(1) 이진 분류

(2) 예측 결과 뿐만 아니라 이진수 출력 값이 1일 확률을 알려줌

Deep Learning Model (1) - 1

Basic Artificial Neural Network

```
ann = Sequential([
    Dense(units=12, activation='elu'),
    Dense(units=6, activation='elu'),
    Dense(units=3, activation='elu'),
    Dense(units=1, activation='sigmoid')
])
```

```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Deep Learning Model (1) - 1

Basic Artificial Neural Network

```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
result = ann.evaluate(X_test, y_test)

print('loss (cross-entropy) :', result[0])
print('test accuracy :', result[1])
```

```
44/44 [==============================] - 0s 1ms/step - loss: 0.4333 - accuracy: 0.8045
loss (cross-entropy) : 0.43327051401138306
test accuracy : 0.8045486807823181
```

ANN : 0.8045

Deep Learning Model (1) - 2

Keras Tuner

```
tuner = kt.BayesianOptimization(build_hyper_model,
                                objective = 'val_accuracy',
                                max_trials = 10,
                                directory = 'test_prac_dir',
                                project_name = 'Basic')
```

< Build_hyper_model 수정 사항 >

Input_shape = (19,,)
model.add(layers.Dense(1, activation='sigmoid')
Loss = 'binary_corssentropy'

```
tuner.search(X_train, y_train, epochs=10, validation_data = (X_test, y_test))
```

```
Trial 10 Complete [00h 00m 05s]
val_accuracy: 0.8045486807823181

Best val_accuracy So Far: 0.8095238208770752
Total elapsed time: 00h 01m 26s
```

ANN : 0.8045 | Using_Keras_Tuner : 0.8095

Deep Learning Model (1) - 2

Keras Tuner

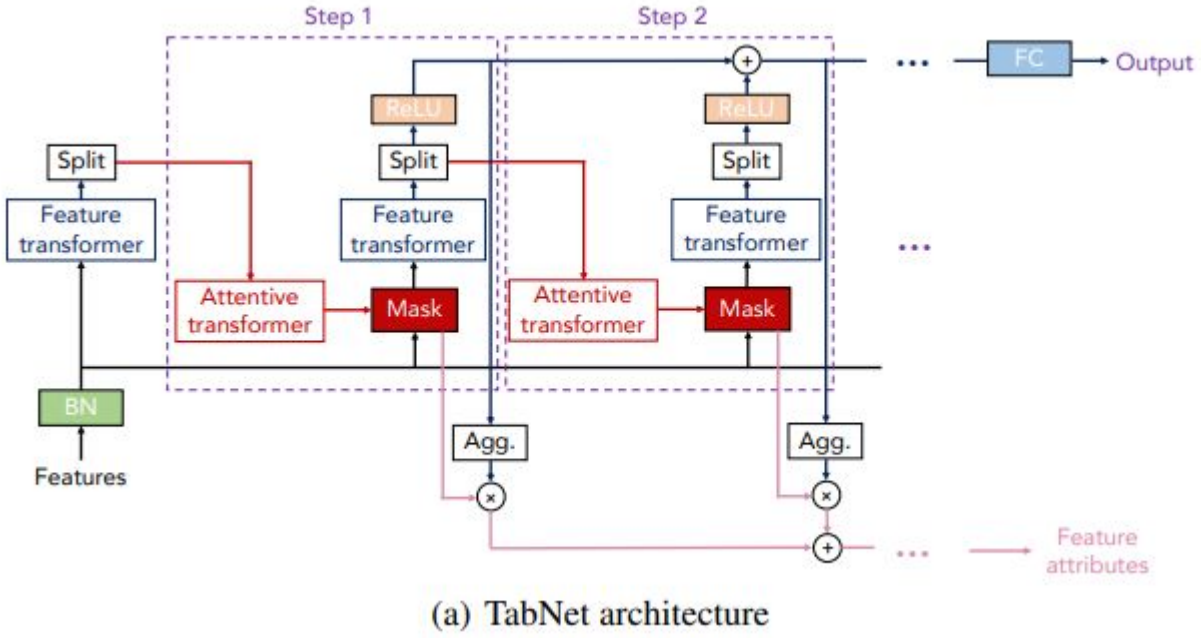
> Tabular data의 경우
Layer를 깊게 쌓는다고
성능이 좋아지지는 않음

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 19)                0

 dense (Dense)               (None, 32)                640

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 673
Trainable params: 673
Non-trainable params: 0
_________________________________________________________________

44/44 [==============================] - 0s 2ms/step - loss: 0.4278 - accuracy: 0.8095
Cross-entropy : 0.427834153175354
Accuracy : 0.8095238208770752
```

ANN : 0.8045 | Using_Keras_Tuner : 0.8095

Deep Learning Model (2) - 1

TabNet



(a) TabNet architecture

참고 : https://wsshin.tistory.com/5

Deep Learning Model (2) - 1

TabNet

> pip install pytorch-tabnet

```
import torch
import torch.nn as nn
from pytorch_tabnet.tab_model import TabNetClassifier
```

```
train_indices = train[train.Set=="train"].index
valid_indices = train[train.Set=="valid"].index
test_indices = train[train.Set=="test"].index
```

> Train / Valid / Test set으로 구성하기 위해 80%, 10%, 10%로 나눔.

참고 : https://wsshin.tistory.com/5

Deep Learning Model (2) - 1

TabNet

```
unused_feat = ['Set']
features = [ col for col in train.columns if col not in unused_feat+[target]]
cat_idxs = [ i for i, f in enumerate(features) if f in categorical_columns]
cat_dims = [ categorical_dims[f] for i, f in enumerate(features) if f in categorical_columns]
```

> TabNet은 입력으로 Categorical변수를 Embedding 하기 때문에,
Categorical 변수라는 것을 지정해야한다.
(cat_idxs와 cat_dims를 저장)

```
X_train = train[features].values[train_indices]
y_train = train[target].values[train_indices]

X_valid = train[features].values[valid_indices]
y_valid = train[target].values[valid_indices]

X_test = train[features].values[test_indices]
y_test = train[target].values[test_indices]
```

참고 : https://wsshin.tistory.com/5

Deep Learning Model (2) - 1

TabNet

< TabNet의 Classifier를 정의 >

```
clf = TabNetClassifier(cat_idxs=cat_idxs,                    -> 카테고리 변수의 위치
                       cat_dims=cat_dims,                    -> 각 카테고리 변수의 Cardinality ( 한 릴레이션을 구성하는 튜플의 수)
                       cat_emb_dim=10,                       -> 카테고리 변수의 임베딩 사이즈
                       optimizer_fn=torch.optim.Adam,
                       optimizer_params=dict(lr=1e-2),
                       scheduler_params={"step_size":50,
                                         "gamma":0.9},
                       scheduler_fn=torch.optim.lr_scheduler.StepLR,
                       mask_type='sparsemax' # "sparsemax", entmax
                                             활성화함수
                      )
```

Deep Learning Model (2) - 1

TabNet

```
max_epochs = 100

clf.fit(
    X_train=X_train, y_train=y_train,
    eval_set=[(X_train, y_train), (X_valid, y_valid)],
    eval_name=['train', 'valid'],
    eval_metric=['auc'],
    max_epochs=max_epochs , patience=20,
    batch_size=32, virtual_batch_size=16,
    num_workers=0,
    weights=1,
    drop_last=False,
)
```

```
epoch 0  | loss: 0.44674 | train_auc: 0.87773 | valid_auc: 0.8051  |  0:00:04s
epoch 1  | loss: 0.43964 | train_auc: 0.88263 | valid_auc: 0.80267 |  0:00:09s
epoch 2  | loss: 0.44847 | train_auc: 0.88486 | valid_auc: 0.80484 |  0:00:14s
epoch 3  | loss: 0.43611 | train_auc: 0.881   | valid_auc: 0.77732 |  0:00:19s
epoch 4  | loss: 0.43903 | train_auc: 0.88257 | valid_auc: 0.80559 |  0:00:23s
epoch 5  | loss: 0.44798 | train_auc: 0.88528 | valid_auc: 0.80854 |  0:00:28s
epoch 6  | loss: 0.44857 | train_auc: 0.88763 | valid_auc: 0.8027  |  0:00:33s
epoch 7  | loss: 0.43669 | train_auc: 0.88944 | valid_auc: 0.80392 |  0:00:38s
epoch 8  | loss: 0.43666 | train_auc: 0.89192 | valid_auc: 0.80793 |  0:00:43s
epoch 9  | loss: 0.43153 | train_auc: 0.89299 | valid_auc: 0.78451 |  0:00:48s
epoch 10 | loss: 0.42353 | train_auc: 0.89225 | valid_auc: 0.80161 |  0:00:53s
```

참고 : https://wsshin.tistory.com/5

Deep Learning Model (2) - 1

TabNet

```
preds = clf.predict_proba(X_test)
```

```
test_auc = roc_auc_score(y_score=preds[1:,1], y_true=y_test[1:])
```

```
test_auc
```

```
0.8308724584784973
```

ANN : 0.8045 | **Using_Keras_Tuner : 0.8095** | **TabNet : 0.8308**

참고 : https://wsshin.tistory.com/5

Deep Learning Model (2) - 2

TabGAN

```
from tabgan.sampler import OriginalGenerator, GANGenerator
```

```
1 new_train1, new_target1 = OriginalGenerator().generate_data_pipe(X_train, y_train, X_valid, )
```

```
1 new_train1.shape #5632->16559
```

```
(16559, 19)
```

```
1 preds = clf.predict_proba(X_test)
2 test_auc = roc_auc_score(y_score=preds[1:,1], y_true=y_test[1:])
3 test_auc
```

```
0.806815881952494
```

ANN : 0.8045 | Using_Keras_Tuner : 0.8095 | TabNet : 0.8308 | TabGAN : 0.8068

참고 : https://wsshin.tistory.com/5

Deep Learning Model (2) - 3

TabNet SMOTE

```
col_list = list(range(19))
numerical_col = [4,17,18]
categorical_col = [x for x in col_list if x not in numerical_col]
smote_sample = SMOTENC(categorical_features= categorical_col,random_state=0,sampling_strategy=1)
X_train_over, y_train_over = smote_sample.fit_resample(X_train, y_train)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', X_train.shape, y_train.shape)
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ', X_train_over.shape, y_train_over.shape) #5618->8268
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트:  (5623, 19) (5623, 1)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트:  (8268, 19) (8268, 1)
```

```
preds = clf.predict_proba(X_test)
```

```
test_auc = roc_auc_score(y_score=preds[1:,1], y_true=y_test[1:])
```

```
test_auc
```

```
0.8333198568800663
```

ANN : 0.8045 | Using_Keras_Tuner : 0.8095 | TabNet : 0.8308 | TabGAN : 0.8068 | TabNet SMOTE : 0.8333

참고 : https://wsshin.tistory.com/5

Deep Learning Model (3) - 1

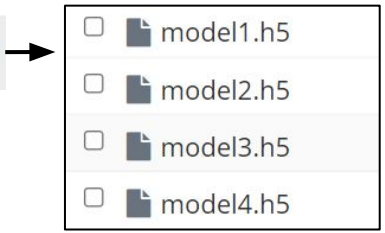
Stacking - NNs

> 여러 model을 이용하여 Stacking을 진행

```
model1 = Sequential()
model1.add(Dense(12,activation = 'elu',input_dim = 19))
model1.add(Dense(6,activation = 'elu'))
model1.add(Dense(1,activation = 'sigmoid'))
```

```
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model1.fit(X_train,y_train,validation_data = (X_test,y_test),epochs = 100)
```

```
model1.save('model1.h5')
```



참고 : https://github.com/YashK07/Stacking-Ensembling/blob/main/Ensemble_Stacking_in_Neural_Networks.ipynb

Deep Learning Model (3) - 1

Stacking - NNs

> 여러 model을 이용하여 Stacking을 진행

```
def load_all_models(n_models):
    all_models = list()
    for i in range(n_models):
        filename = 'model' + str(i + 1) + '.h5'
        model = load_model(filename)
        all_models.append(model)
        print('>loaded %s' % filename)
    return all_models
```

```
n_members = 4
members = load_all_models(n_members)
print('Loaded %d models' % len(members))
```

```
>loaded model1.h5
>loaded model2.h5
>loaded model3.h5
>loaded model4.h5
Loaded 4 models
```

참고 : https://github.com/YashK07/Stacking-Ensembling/blob/main/Ensemble_Stacking_in_Neural_Networks.ipynb

Deep Learning Model (3) - 1

Stacking - NNs

> 여러 model을 이용하여 Stacking을 진행

```
model = fit_stacked_model(members, X_test,y_test)
```

```
def fit_stacked_model(members, inputX, inputy):
    stackedX = stacked_dataset(members, inputX)
    model = LogisticRegression()
    model.fit(stackedX, inputy)
    return model
```

```
def stacked_dataset(members, inputX):
    stackX = None
    for model in members:
        yhat = model.predict(inputX, verbose=0)

        if stackX is None:
            stackX = yhat #
        else:
            stackX = dstack((stackX, yhat))

    stackX = stackX.reshape((stackX.shape[0], stackX.shape[1]*stackX.shape[2]))
    return stackX
```

참고 : https://github.com/YashK07/Stacking-Ensembling/blob/main/Ensemble_Stacking_in_Neural_Networks.ipynb

Deep Learning Model (3) - 1

Stacking - NNs

> 여러 model을 이용하여 Stacking을 진행

```
yhat = stacked_prediction(members, model, X_test)
```

```
def stacked_prediction(members, model, inputX):

    stackedX = stacked_dataset(members, inputX)
    yhat = model.predict(stackedX)
    return yhat
```

```
loss (cross-entropy) : 0.424976110458374
test accuracy : 0.8081023693084717
loss (cross-entropy) : 0.42592695355415344
test accuracy : 0.7988628149032593
loss (cross-entropy) : 0.42030492424964905
test accuracy : 0.8045486807823181
loss (cross-entropy) : 0.4239329397678375
test accuracy : 0.8031272292137146
```

```
1  np.mean(yhat==y_test)
```

```
0.8038379530916845
```

ANN : 0.8045 | Using_Keras_Tuner : 0.8095 | TabNet : 0.8308 | TabGAN : 0.8068 | TabNet SMOTE : 0.8333 | Stacking(NN) : 0.8038

Deep Learning Model (3) - 2

Stacking - NN + ML

```
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
```

> ML 파트에서 ACC가 높았던 3가지 모델

```
xgboost_model = XGBClassifier(n_estimators = 100, learning_rate = 0.01, max_depth=3)
rf_model = RandomForestClassifier(n_estimators=100, random_state=30)
lg_model = LogisticRegression(C=10)
```

참고 : https://github.com/YashK07/Stacking-Ensembling/blob/main/Ensemble_Stacking_in_Neural_Networks.ipynb

Deep Learning Model (3) - 2

Stacking - NN + ML

```
xgboost_model = XGBClassifier(n_estimators = 100, learning_rate = 0.01, max_depth=3)
xgboost_model.fit(X_train, y_train)
xgboost_pred = xgboost_model.predict(X_test)

rf_model = RandomForestClassifier(n_estimators=100, random_state=30)
lg_model = LogisticRegression(C=10)
```

```
model1 = Sequential()
model1.add(Dense(12,activation = 'elu',input_dim = 19))
model1.add(Dense(6,activation = 'elu'))
model1.add(Dense(1,activation = 'sigmoid'))
```

```
model3 = Sequential()
model3.add(Dense(12,activation = 'elu',input_dim = 19))
model3.add(Dense(6,activation = 'elu'))
model3.add(Dense(6,activation = 'elu'))
model3.add(Dropout(0.1))
model3.add(Dense(3,activation = 'elu'))
model3.add(Dense(1,activation = 'sigmoid'))
```

참고 : https://github.com/YashK07/Stacking-Ensembling/blob/main/Ensemble_Stacking_in_Neural_Networks.ipynb

Deep Learning Model (3) - 2

Stacking - NN + ML

```
pred = np.array([rf_pred,xgboost_pred, model1_pred, model3_pred])
```

```
pred = np.transpose(pred)
```

```
lr_final = LogisticRegression(C=10)
```

```
lr_final.fit(pred, y_test)
final = lr_final.predict(pred)
```

```
print('최종 정확도: {0:.4f}'.format(accuracy_score(y_test , final)))
```

```
최종 정확도: 0.8102
```

ANN : 0.8045 | Using_Keras_Tuner : 0.8095 | TabNet : 0.8308 | TabGAN : 0.8068 | TabNet SMOTE : 0.8333 | Stacking(NN) : 0.8038 | Stacking(NN+ML) : 0.8102

결론

ML

가장 잘 나온 모델 : **GradientBoostingModel(acc:약 0.8983)**
LogisticRegression,GradientBoosting,RandomForestClassifier(acc: 약 0.7924)
AutoML : MLJAR(acc: 약 0.8238)/ Google(acc: 0.839)

DL

가장 잘 나온 모델 : **TabNet SMOTE (acc : 약 0.8333)**

추가 분석 과제

- 전체 **accuracy**가 **0.89**을 넘지 못 함
- **Machine Learning** 모델들이 **Deep Learning** 모델들보다 **Acc**가 높음
- 통신사에 대한 정보 및 열과 다른 열들의 관계성을 잘 알 수 있다면, 전처리를 통해 더 높은 **acc**를 얻을 수 있을 것으로 예측