의료비 예측 프로젝트

프레젠테이션

이예지

목 차

01 주제 선정 동기

경제적 부담 감소

예상치 못한 의료 비용은 부담이 될 수 있습니다.
따라서 더 효과적으로 재정을 계획하고 관리하고자 해당 프로젝트를 진행하게 되었습니다.

02 분석 주제 및 목표

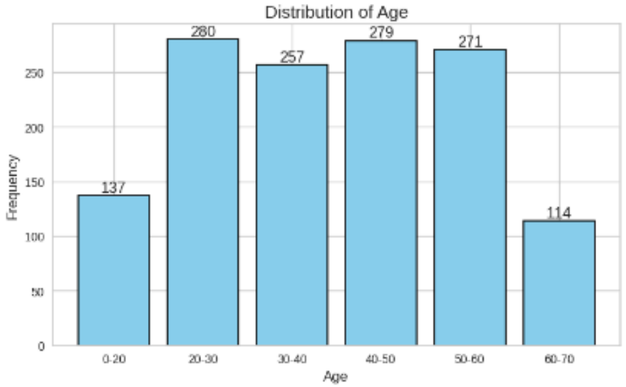
주제	의료비 예측 모델링
목표	의료비 예측을 위해 머신러닝, 딥러닝, 하이퍼파라미터 최적화, 스태킹 기법을 적용

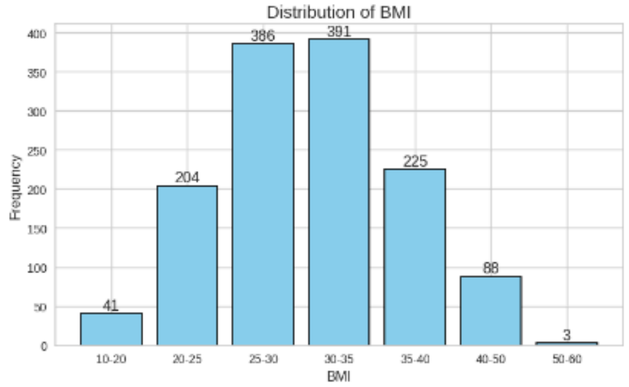
03 데이터 탐색 및 소개

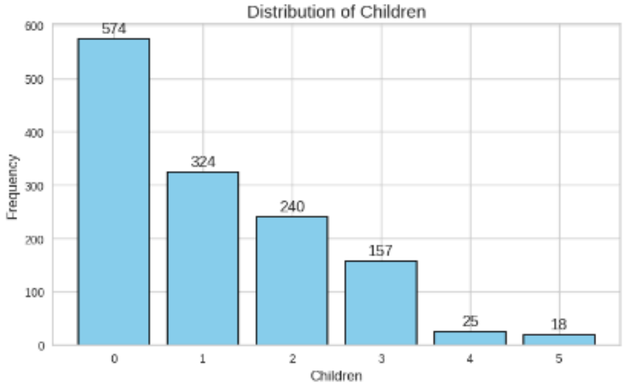
	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

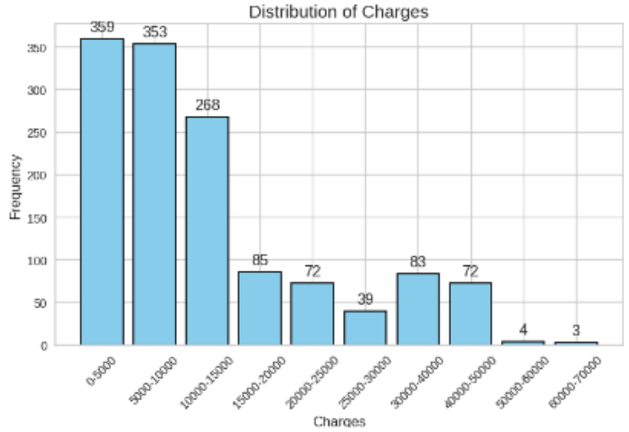
- 나이: 주요 수혜자의 나이
- 성별: 보험 계약자의 성별 (여성, 남성).
- bmi: 신체 질량 지수 (체중/키^2).
- 자녀: 건강보험 적용 자녀 수 / 부양가족 수
- 흡연자: 흡연 여부.
- 지역: 미국 내 수혜자의 거주 지역 (북동부, 남동부, 남서부, 북서부).
- 요금: 건강 보험에서 청구하는 개인 의료비

03 데이터 탐색 및 소개



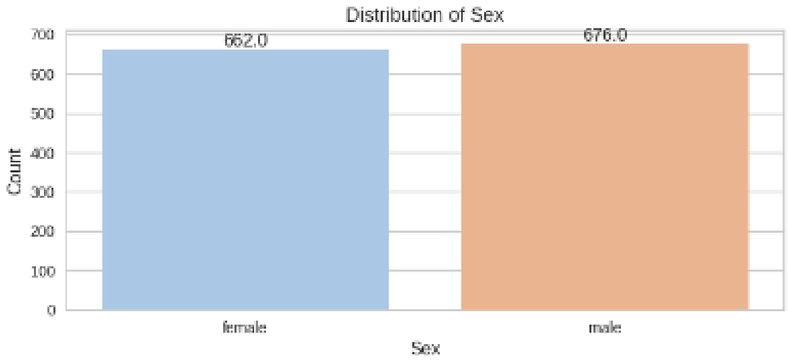


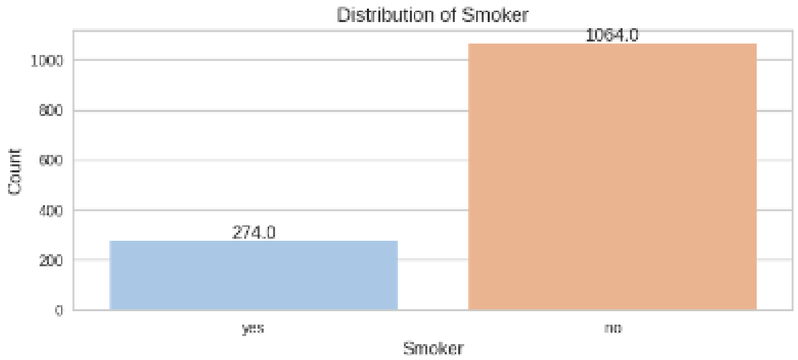


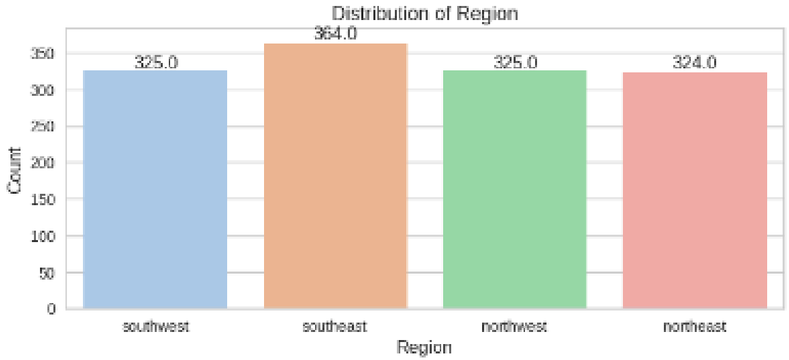


- 수치형 데이터 분포 시각화
- 구간으로 나눠 막대 그래프로 표현

03 데이터 탐색 및 소개







- 범주형 데이터 분포 시각화
- Countplot 활용
- 성별이나 지역에 따른 차이는 거의 없음
- 흡연자 : 비흡연자 = 1 : 3

04 데이터 전처리

1

결측값 처리 불필요
(결측값 없음)

2

이상치 제거 적용하지 않음
(이상치 제거 시 모델 성능 하락)

3

전처리 과정에
파이프라인 적용

4

명확한 식별을 위해
각 컬럼명 재구성

04 데이터 전처리

- Train-test split 적용
- 데이터 전처리 파이프라인 구성
  -> 수치형 변수 : MinMaxScaler 적용
  -> 범주형 변수 : OnehotEncoder 적용
- 각 컬럼명 재구성

```
# Pipeline 정의
# 수치형 변수의 전처리
numeric_transformer = Pipeline(steps=[
    ('scaler', MinMaxScaler())
])

# 범주형 변수의 전처리
categorical_transformer = Pipeline(steps=[

    # 만약 학습할 때는 있지만 테스트 데이터에서 새로운 범주가 발생할 경우
    # handle_unknown='ignore' 옵션을 통해 이를 무시하고 변환 작업을 진행
    # 이 옵션을 사용함으로써 모델이 학습할 때와 테스트할 때의 일관성을 유지하면서 처리할 수 있음.
    # (이름, 변환기)
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# 전체 전처리 파이프라인 정의
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, ['age', 'bmi', 'children']),  # 수치형 변수
        ('cat', categorical_transformer, ['sex', 'smoker', 'region'])  # 범주형 변수
    ])

# 최종 Pipeline 정의 (전처리 + 모델)
pipeline = Pipeline(steps=[('preprocessor', preprocessor)])

# Train 데이터에 대해서만 fit을 적용
pipeline.fit(X_train)

# Train 데이터와 Test 데이터 각각에 대해 transform 수행
X_train = pipeline.transform(X_train)
X_test = pipeline.transform(X_test)

# 컬럼명 재구성
# 수치형 변수와 범주형 변수의 컬럼명을 다시 가져오기
numeric_features = preprocessor.transformers_[0][2]
categorical_features = preprocessor.transformers_[1][1]['onehot'].get_feature_names(['sex', 'smoker', 'region'])

# 전체 특성의 컬럼명 재구성
feature_names = list(numeric_features) + list(categorical_features)

X_train = pd.DataFrame(X_train, columns=feature_names)
X_test = pd.DataFrame(X_test, columns=feature_names)
```

04 데이터 전처리

age	bmi	children	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	region_southwest
0.608696	0.107345	0.4	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0
0.630435	0.224913	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
0.739130	0.239440	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
0.456522	0.493947	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
0.782609	0.148238	0.6	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0

05 머신러닝 모델 분석

```
# PyCaret의 setup 함수 사용하여 머신 러닝 모델을 학습하기 위한 형태로 변환
model = setup(data=df,
              target='charges',
              categorical_features=['sex', 'smoker', 'region'],
              normalize=True, normalize_method='minmax',
              train_size=0.8,
              session_id=42) # Random seed
```

```
# 모델의 설명력을 평가하고, 그 성능을 비교하기 위한 목적에서
# 좀 더 직관적이고 일반적으로 사용되는 지표를 활용하기 위해서 R2 지표도 확인해보기.
top_3_models = compare_models(sort='RMSE', n_select = 3)
```

상위 3개 모델

GradientBoostingRegressor

LightGBMRegressor

RandomForestRegressor

05 머신러닝 모델 분석

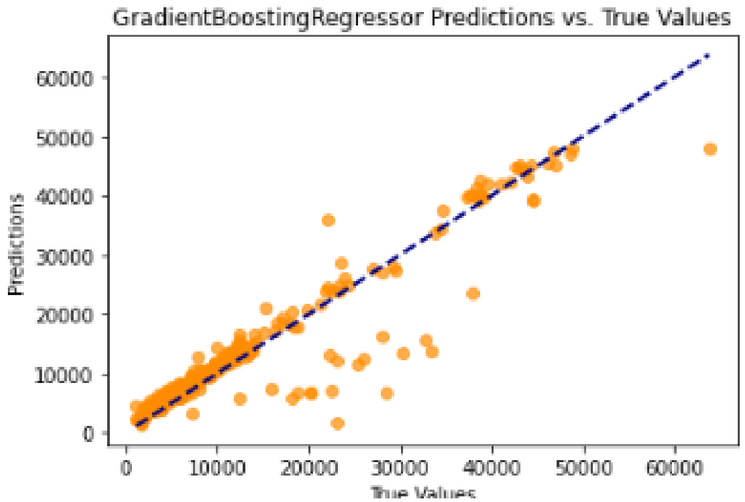
	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2609.8744	21714023.9797	4624.2374	0.8444	0.4325	0.3031	0.0620
lightgbm	Light Gradient Boosting Machine	2943.6636	24416570.6599	4910.8092	0.8248	0.5236	0.3583	97.3950
rf	Random Forest Regressor	2825.1307	24838293.4569	4959.9233	0.8230	0.4640	0.3314	0.1140
ada	AdaBoost Regressor	3981.0569	26747817.8312	5156.3222	0.8104	0.5917	0.6510	0.0340
et	Extra Trees Regressor	2768.2531	27165447.1879	5192.9926	0.8057	0.4720	0.3153	0.0970
xgboost	Extreme Gradient Boosting	3250.5256	30095093.6000	5467.6827	0.7873	0.6067	0.4278	0.0480
knn	K Neighbors Regressor	3649.9163	35550660.4000	5934.6120	0.7469	0.4883	0.3600	0.0310
lasso	Lasso Regression	4242.3928	37881920.2778	6132.7752	0.7298	0.5870	0.4237	0.0250
llar	Lasso Least Angle Regression	4242.3837	37882017.1413	6132.7813	0.7298	0.5870	0.4237	0.0250
lar	Least Angle Regression	4243.9235	37885866.4059	6133.0840	0.7297	0.5877	0.4241	0.0260
br	Bayesian Ridge	4240.6531	37885980.6834	6133.2662	0.7298	0.5897	0.4228	0.0250
ridge	Ridge Regression	4235.7688	37890233.7232	6133.9223	0.7299	0.5909	0.4209	0.0260
lr	Linear Regression	4247.5849	37953589.9000	6138.7322	0.7293	0.6280	0.4250	0.4640
dt	Decision Tree Regressor	3120.2748	43348469.6871	6568.9552	0.6941	0.5226	0.3527	0.0260
huber	Huber Regressor	3496.3265	49075381.3739	6961.7764	0.6476	0.4698	0.2222	0.0280
par	Passive Aggressive Regressor	3797.4028	51936051.9205	7173.6631	0.6283	0.4637	0.2107	0.0300
omp	Orthogonal Matching Pursuit	5712.8153	56825791.8833	7527.8548	0.5968	0.7195	0.8627	0.0240
en	Elastic Net	7405.0171	102227257.7758	10087.2033	0.2882	0.8715	1.2324	0.0250
dummy	Dummy Regressor	9004.8338	144530836.0000	11994.8297	-0.0067	0.9877	1.4919	0.0240

05 머신러닝 모델 분석

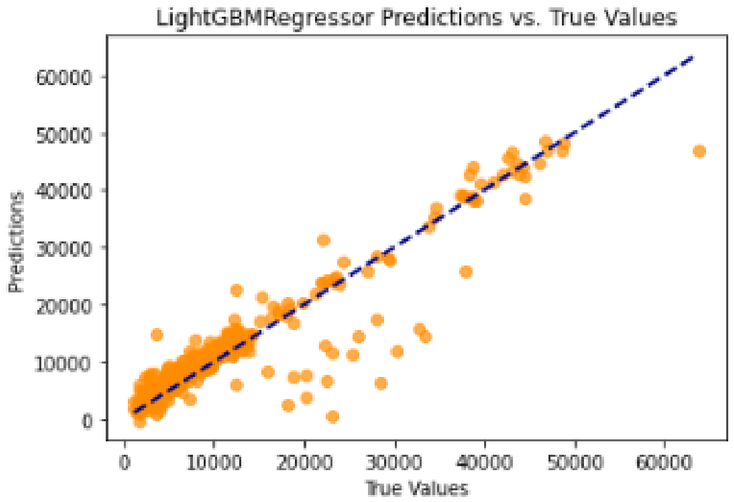
```
models = {
    'GradientBoostingRegressor': GradientBoostingRegressor(random_state=0),
    'LightGBMRegressor': LGBMRegressor(),
    'RandomForestRegressor': RandomForestRegressor(random_state=0),
}

for model_name, model in models.items():
    model.fit(X_train, y_train)

    pred = model.predict(X_test)

    # RMSE 계산
    rmse = np.sqrt(mean_squared_error(y_test, pred))
    print(f'{model_name} RMSE: {rmse:.4f}')  # RMSE 출력
    print(f"R2 Score on Test set for {model_name}: {model.score(X_test, y_test):.6f}")  # 테스트 세트 R2 Score
```

RMSE와 R2 스코어를 확인

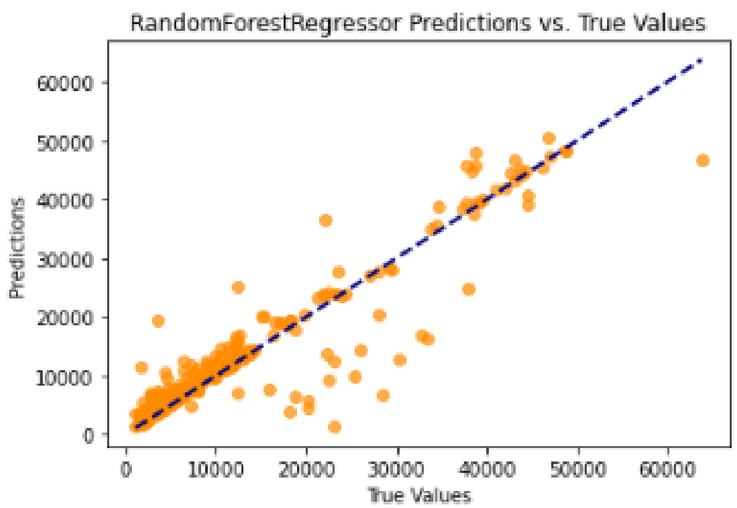
05 머신러닝 모델 분석



GradientBoostingRegressor
RMSE: 4329.2109
R2 Score: 0.879277



LightGBMRegressor
RMSE: 4571.5871
R2 Score: 0.865381



RandomForestRegressor
RMSE: 4652.8020
R2 Score: 0.860556

05 머신러닝 모델 분석

```
models = {
    'GradientBoostingRegressor': (GradientBoostingRegressor(random_state=0), {
        'n_estimators': Integer(100, 500),  # 트리 개수
        'max_depth': Integer(3, 12),          # 최대 깊이
        'learning_rate': Categorical([0.01, 0.1, 0.2, 0.3, 0.4]),  # 학습률
        'subsample': Categorical([0.5, 0.6, 0.7, 0.8, 0.9, 1.0])  # 샘플 비율
    }),
    'LightGBMRegressor': (LGBMRegressor(random_state=0), {
        'n_estimators': Integer(100, 500),  # 트리 개수
        'learning_rate': Categorical([0.01, 0.1, 0.2, 0.3, 0.4]),  # 학습률
        'subsample': Categorical([0.5, 0.6, 0.7, 0.8, 0.9, 1.0]),  # 샘플 비율
        'num_leaves': Integer(20, 100)  # 리프 노드의 수
    }),
    'RandomForestRegressor': (RandomForestRegressor(random_state=0), {
        'n_estimators': Integer(100, 500),  # 트리 개수
        'max_depth': Integer(3, 20),          # 최대 깊이
        'min_samples_split': Integer(2, 20),  # 최소 샘플 분할 수
        'min_samples_leaf': Integer(1, 20)  # 최소 리프 노드 샘플 수
    })
}
```

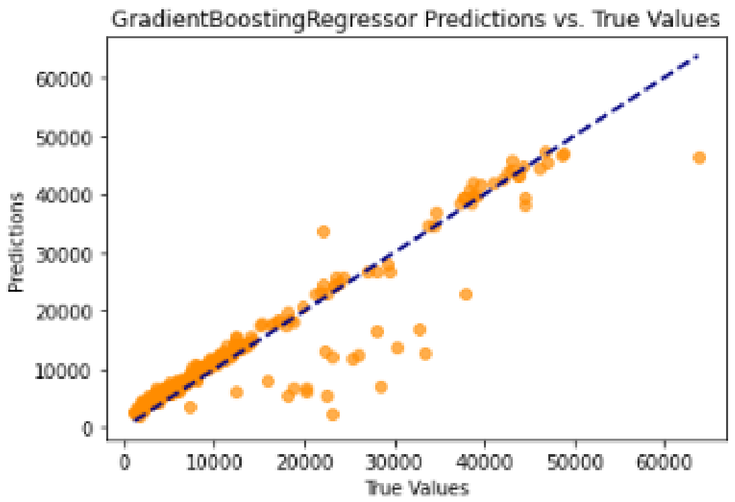
Bayesian HPO 적용을 위한 하이퍼파라미터 범위 설정

05 머신러닝 모델 분석

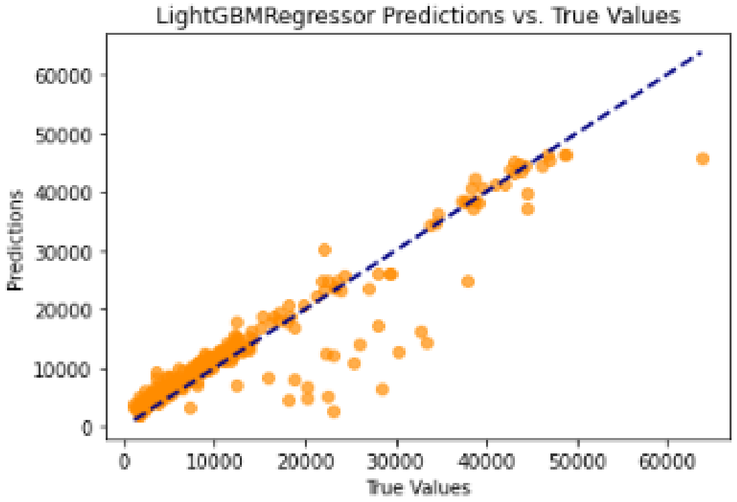
```
# 각 모델에 대해 베이지안 최적화를 수행하고 결과를 출력합니다.
for model_name, (estimator, param) in models.items():
    print(f"Optimizing {model_name}...")  # 최적화 모델 이름 출력

    # BayesSearchCV 객체를 생성합니다.
    hpo = BayesSearchCV(
        random_state=0,
        estimator=estimator,
        search_spaces=param,  # 하이퍼파라미터
        refit=True,  # 최적 파라미터로 다시 학습
        n_jobs=-1,  # 모든 CPU 코어 사용
        n_iter=72,  # 최적화 반복 횟수
        cv=5  # 교차 검증 folds 수
    )

    # 최적화 수행
    hpo.fit(X_train, y_train)

    # 최적 파라미터 출력
    print('The best parameters are ', hpo.best_params_, '\n')

    # 모델 인스턴스 생성
    best_params = hpo.best_params_
    if model_name == 'GradientBoostingRegressor':
        model_instance = GradientBoostingRegressor(
            random_state=0,
            n_estimators=best_params['n_estimators'],
            max_depth=best_params['max_depth'],
            learning_rate=best_params['learning_rate'],
            subsample=best_params['subsample']
        )
    elif model_name == 'LightGBMRegressor':
        model_instance = LGBMRegressor(
            random_state=0,
            n_estimators=best_params['n_estimators'],
            learning_rate=best_params['learning_rate'],
            subsample=best_params['subsample'],
            num_leaves=best_params['num_leaves']
        )
    elif model_name == 'RandomForestRegressor':
        model_instance = RandomForestRegressor(
            random_state=0,
            n_estimators=best_params['n_estimators'],
            max_depth=best_params['max_depth'],
            min_samples_split=best_params['min_samples_split'],
            min_samples_leaf=best_params['min_samples_leaf']
        )
```

Bayesian HPO

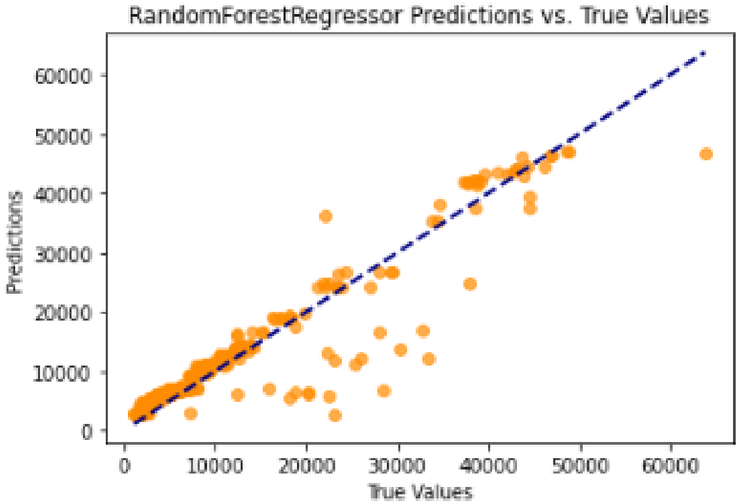
05 머신러닝 모델 분석



GradientBoostingRegressor
RMSE: 4311.7676
R2 Score: 0.880248



LightGBMRegressor
RMSE: 4387.5083
R2 Score: 0.876004



RandomForestRegressor
RMSE: 4408.5625
R2 Score: 0.874811

05 머신러닝 모델 분석

```
# StackingTransformer를 만든다.
stack = StackingTransformer(estimators,
                            regression=True,  # 회귀 문제를 위해 True로 설정
                            n_folds=4, stratified=False, shuffle=True,
                            random_state=0, verbose=2)

# StackingTransformer에 X_train, y_train을 이용해 학습한다.
# transform() 함수를 이용하여 변환한다.
stack = stack.fit(X_train, y_train)

S_train = stack.transform(X_train)
S_test = stack.transform(X_test)

# 2단계 모델을 GradientBoostingRegressor로 설정
model = GradientBoostingRegressor(random_state=0, n_estimators=100)

# 2단계 모델을 학습.
model = model.fit(S_train, y_train)

# 학습된 2단계 모델을 가지고 예측한다.
y_pred = model.predict(S_test)

# RMSE 계산
rmse = np.sqrt(mean_squared_error(y_test, y_pred))  # RMSE 계산
print('Final RMSE: %.4f' % rmse)  # RMSE 출력
print('Final R2 Score: %.4f' % r2_score(y_test, y_pred))

# 예측 결과와 실제 값을 비교하는 산점도 그리기
plt.figure()
plt.scatter(y_test, y_pred, color='darkorange', alpha=0.7)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='navy', lw=2, linestyle='--')
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('Stacking Regressor Predictions vs. True Values')
plt.show()
```

StackingTransformer로 예측을 통합한 후
GradientBoostingRegressor로 최종 예측을 수행

05 머신러닝 모델 분석

Stacking
RMSE: 4570.3875
R2 Score: 0.8655

GradientBoostingRegressor
RMSE: 4329.2109 -> 4311.7676
R2 Score: 0.879277 -> 0.880248

LightGBMRegressor
RMSE: 4571.5871 -> 4387.5083
R2 Score: 0.865381 -> 0.876004

RandomForestRegressor
RMSE: 4652.8020 -> 4408.5625
R2 Score: 0.860556 -> 0.874811

06 딥러닝 모델 분석

```python
# 랜덤 시드 설정
random.seed(42) # 랜덤 시드
np.random.seed(42)  # NumPy 랜덤 시드
tf.random.set_seed(42)  # TensorFlow 랜덤 시드

# 데이터 타입 변환
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Sequential 모델 초기화
model = Sequential()

# 입력층: 입력 데이터의 차원(input_dim)을 지정하고 첫 번째 Dense 레이어 추가
# units=32: 출력 차원, activation='relu': ReLU 활성화 함수 사용
model.add(layers.Dense(units=32, input_dim=X_train.shape[1], activation='relu'))

# 두 번째 Dense 레이어 추가
# units=32: 출력 차원, activation='relu': ReLU 활성화 함수 사용
model.add(layers.Dense(units=32, activation='relu'))

# 세 번째 Dense 레이어 추가
# units=480: 출력 차원, activation='relu': ReLU 활성화 함수 사용
model.add(layers.Dense(units=480, activation='relu'))

# 출력층: 최종 예측값을 하나의 스칼라 값으로 출력
# units=1: 출력 차원, activation='linear': 출력 값을 연속적인 값으로 반환
model.add(layers.Dense(units=1, activation='linear'))

# 모델 구조 요약 출력
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                384
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 480)               15840
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 481
=================================================================
Total params: 17,761
Trainable params: 17,761
Non-trainable params: 0
_________________________________________________________________
```

```python
# 모델 컴파일
model.compile(optimizer=optimizers.Adam(learning_rate=0.01),  # Adam 옵티마이저, 학습률 0.01
              loss='mean_squared_error',  # 손실 함수: MSE
              metrics=['mean_squared_error'])  # 평가 지표: MSE

# 모델 학습
model.fit(X_train, y_train, epochs=100, validation_split=0.2, verbose=0)

# 모델 평가
mse_result = model.evaluate(X_test, y_test)  # MSE 평가

# 예측 결과
y_pred = model.predict(X_test)

# MSE와 RMSE 계산
mse = mse_result[0]
rmse = np.sqrt(mse)

# R² 계산
r2 = r2_score(y_test, y_pred)

# 결과 출력
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R²: {r2:.4f}')
```

```
9/9 [==============================] - 0s 3ms/step - loss: 23122986.0000 - mean_squared_error: 23122986.0000
MSE: 23122986.0000
RMSE: 4808.6366
R²: 0.8511
```

4개의 Dense 레이어를 가진
신경망 모델 정의

MSE: 23122986.0000
RMSE: 4808.6366
R^2: 0.8511

06 딥러닝 모델 분석

```python
# 하이퍼파라미터 튜닝을 위한 모델과 튜너를 설정

def build_hyper_model(hp):
    # Sequential 모델 초기화
    model = models.Sequential()

    # 입력층 추가
    model.add(layers.Dense(
        units=hp.Int('units_input', min_value=32, max_value=512, step=32),
        activation=hp.Choice('activation_input', values=['relu', 'elu']),
        input_dim=X_train.shape[1]  # 입력 데이터의 특성 수
    ))

    # 은닉층 추가
    for i in range(hp.Int('num_layers', min_value=1, max_value=3)):
        model.add(layers.Dense(
            units=hp.Int(f'units_{i}', min_value=32, max_value=512, step=32),
            activation=hp.Choice(f'activation_{i}', values=['relu', 'elu'])
        ))

    # 출력층 추가
    model.add(layers.Dense(1, activation='linear'))

    # 학습률 설정
    learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

    # 옵티마이저 선택
    optimizer_choice = hp.Choice('optimizer', values=['adam', 'sgd', 'rmsprop'])
    if optimizer_choice == 'adam':
        optimizer = optimizers.Adam(learning_rate=learning_rate)
    elif optimizer_choice == 'sgd':
        optimizer = optimizers.SGD(learning_rate=learning_rate, momentum=0.9)
    else:
        optimizer = optimizers.RMSprop(learning_rate=learning_rate)

    # 모델 컴파일
    model.compile(
        optimizer=optimizer,
        loss=losses.MeanSquaredError(),
        metrics=['MeanSquaredError']
    )

    return model
```

```python
# 하이퍼파라미터 튜너 설정
tuner = kt.BayesianOptimization(
    build_hyper_model,  # 모델 생성 함수
    objective='mean_squared_error',  # 최소화할 목표: MSE
    max_trials=10,  # 시도할 하이퍼파라미터 조합의 수
    directory='test_prac_dir',  # 결과를 저장할 디렉토리
    project_name='Medical_hyper_1'  # 프로젝트 이름
)

# 탐색 공간 요약 출력
tuner.search_space_summary()
```

```
Default search space size: 10
units_input (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
activation_input (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'elu'], 'ordered': False}
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 1, 'max_value': 3, 'step': 1, 'sampling': None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
activation_0 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'elu'], 'ordered': False}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
activation_1 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'elu'], 'ordered': False}
units_2 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
activation_2 (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'elu'], 'ordered': False}
```

Keras Tuner의 Bayesian Optimization을 사용하여 모델 하이퍼파라미터를 효율적으로 탐색

06 딥러닝 모델 분석

```python
# 하이퍼파라미터 튜닝 및 상위 N개의 트라이얼 출력

# 훈련 데이터와 검증 데이터로 분할 (검증 데이터는 모델 튜닝에 사용)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

# 하이퍼파라미터 튜닝 시작
tuner.search(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# 상위 N개의 트라이얼 가져오기
top_trials = tuner.oracle.get_best_trials(num_trials=3)

# 각 트라이얼의 하이퍼파라미터 및 성능 출력
for i, trial in enumerate(top_trials):
    mse = trial.score
    rmse = np.sqrt(mse)

    print(f"Trial {i + 1}:")
    print(f"  하이퍼파라미터: {trial.hyperparameters.values}")
    print(f"  Score (RMSE): {rmse:.4f}")
    print(f"  디렉토리 (폴더 번호): {trial.trial_id}")
    print("-" * 40)

# 최적 하이퍼파라미터 선택
best_hps = tuner.get_best_hyperparameters(num_trials=3)[0]
```

```
INFO:tensorflow:Oracle triggered exit
Trial 1:
  하이퍼파라미터: {'units_input': 224, 'activation_input': 'relu', 'num_layers': 2, 'units_0': 96, 'activation_0': 'elu', 'learning_rate': 0.01, 'units
_1': 224, 'activation_1': 'elu'}
  Score (RMSE): 5483.6017
  디렉토리 (폴더 번호): 274307a55f05b51df7516039e82a9a94
----------------------------------------
Trial 2:
  하이퍼파라미터: {'units_input': 160, 'activation_input': 'elu', 'num_layers': 3, 'units_0': 256, 'activation_0': 'relu', 'learning_rate': 0.01, 'unit
s_1': 160, 'activation_1': 'relu', 'units_2': 32, 'activation_2': 'relu'}
  Score (RMSE): 5625.7542
  디렉토리 (폴더 번호): 29dc6aaa7ad37888e12b5d6c71855431
----------------------------------------
Trial 3:
  하이퍼파라미터: {'units_input': 32, 'activation_input': 'elu', 'num_layers': 3, 'units_0': 192, 'activation_0': 'relu', 'learning_rate': 0.01, 'units
_1': 320, 'activation_1': 'elu', 'units_2': 32, 'activation_2': 'relu'}
  Score (RMSE): 5671.3184
  디렉토리 (폴더 번호): d6f8608dda62c8f0efd50dce2dd0185b
----------------------------------------
```

최적 하이퍼파라미터 저장
'units_input': 224, 'activation_input': 'relu',
'num_layers': 2, 'units_0': 96, 'activation_0': 'elu',
'learning_rate': 0.01, 'units_1': 224, 'activation_1': 'elu'
Score (RMSE): 5483.6017

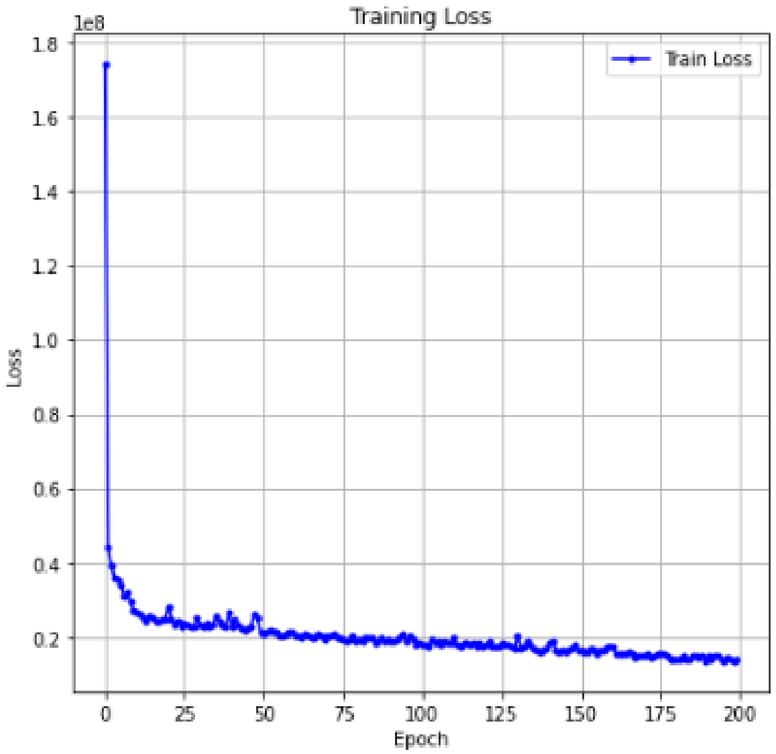
06 딥러닝 모델 분석

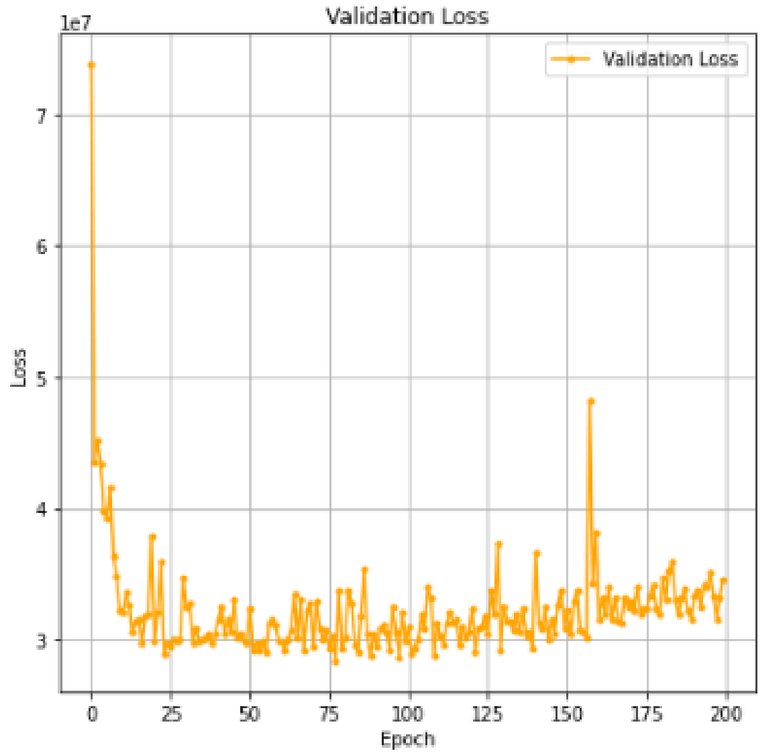
```
# 모델 빌드, 훈련, 체크포인트 저장

# 최적 하이퍼파라미터로 모델 빌드
model = tuner.hypermodel.build(best_hps)

# 체크포인트 콜백 객체 생성
checkpoint_path = 'best_medical_charge_model.h5'
callback_checkpoint = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    monitor='val_loss',
    save_best_only=True,
    verbose=0
)

# 모델 훈련
# 훈련 과정에서 가장 좋은 성능을 가진 모델이 checkpoint_path에 저장됨
history = model.fit(
    X_train,
    y_train,
    batch_size=16,
    epochs=200,
    validation_data=(X_val, y_val),
    verbose=0,
    callbacks=[callback_checkpoint]
)
```

```
# 모델 로드, 평가, 결과 출력

# 체크포인트에서 모델 로드
model = models.load_model(checkpoint_path)

# 모델 평가
results = model.evaluate(X_test, y_test, verbose=0)
y_pred = model.predict(X_test).flatten()

# 성능 지표 계산
mse = results[1]  # 손실과 MSE는 결과의 순서에 따라 다를 수 있음
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

최적 하이퍼파라미터로 모델 훈련
최적의 은닉층 수: 2
최적의 학습률: 0.01
층 1 - 유닛 수: 96
층 1 - 활성화 함수: elu
층 2 - 유닛 수: 224
층 2 - 활성화 함수: elu
MSE: 22013308.0000
RMSE: 4691.8342
R^2: 0.8582

06 딥러닝 모델 분석


Training Loss
1e8
Train Loss
Loss
Epoch
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
25
50
75
100
125
150
175
200


Validation Loss
1e7
Validation Loss
Loss
Epoch
7
6
5
4
3
0
25
50
75
100
125
150
175
200

06 딥러닝 모델 분석

```python
# TabNet 모델 정의
model = TabNetRegressor(verbose=0, seed=42, optimizer_fn=torch.optim.Adam)

# 모델 학습
model.fit(
    X_train=X_train.values,
    y_train=y_train,
    eval_set=[(X_test.values, y_test)], # 평가 세트

    # 조기 종료를 위한 patience 설정.
    # 이 값만큼의 에포크 동안 성능 개선이 없으면 학습을 조기 종료합니다.
    patience=100,
    max_epochs=1000,
    eval_metric=['rmse']
)

# 예측
y_pred = model.predict(X_test.values)

# 평가
r2 = r2_score(y_test, y_pred)
print(f'R²  Score: {r2:.4f}')

# 모델 학습이 max_epochs 설정값인 1000에 도달했지만, 가장 좋은 성능을 보인 에포크는 981임.
# TabNetRegressor는 학습 도중 최적의 성능을 보인 에포크(여기서는 981)의 가중치를 자동으로 사용하여 모델을 저장.
```

TabNetRegressor
RMSE: 6321.46994
R^2 Score: 0.7426

07 분석 결론 및 요약

ML Part

GradientBoostingRegressor
RMSE: 4329.2109 -> 4311.7676
R2 Score: 0.879277 -> 0.880248

LightGBMRegressor
RMSE: 4571.5871 -> 4387.5083
R2 Score: 0.865381 -> 0.876004

RandomForestRegressor
RMSE: 4652.8020 -> 4408.5625
R2 Score: 0.860556 -> 0.874811

Stacking
RMSE: 4570.3875
R2 Score: 0.8655

DL Part

Deep Neural Network
RMSE: 4808.6366
R^2: 0.8511

Keras-Tuner를 이용한 딥러닝 하이퍼파라미터 최적화
RMSE: 4691.8342
R^2: 0.8582

TabNetRegressor
RMSE: 6321.46994
R^2: 0.7426

07 분석 결론 및 요약

- GradientBoostingRegressor: 하이퍼파라미터 최적화 후 RMSE가 4329.21에서 4311.77로, R^2 Score가 0.8793에서 0.8802로 향상되며, 최종적으로 가장 높은 성능을 기록함.

- LightGBMRegressor, RandomForestRegressor, Stacking: GradientBoostingRegressor와 비교했을 때 상대적으로 낮은 성능을 보임.

- Keras-Tuner를 이용한 딥러닝 하이퍼파라미터 최적화: 여전히 GradientBoostingRegressor와 같은 머신러닝 모델에 비해 성능이 미치지 못함. 이는 정형 데이터에서 딥러닝 모델의 성능이 제한적일 수 있음.

- TabNetRegressor: RMSE가 6321.47로 가장 높고, R^2 Score는 0.7426으로 가장 낮음. 이는 정형 데이터에서 딥러닝 기반 모델의 성능이 기대 이하였으며, 데이터 수가 부족해 딥러닝 모델이 최적의 성능을 발휘하지 못했을 가능성이 있음.

08 추후 보완 및 개선점

데이터 수의 부족

의료 관련 feature의
수가 적고 단순함

미국 의료기반 시스템
으로 측정한 의료비

Data 보완점

08 추후 보완 및 개선점

새로운 특성 추가,
피처 간의 상호작용 고려

Feature Importance
분석을 통해 모델이 어떤
피처에 의존하는지 파악

더 다양한 하이퍼파라미터
검색 기법을 사용

한국 실정에 맞게 소득분위 및 보험료에 대한 분석을 할 경우 더욱 가계 재정에 효과적인 분석이 가능

감사합니다

이예지