텀프로젝트 레포트

학과: 로봇학부
학번: 2020741048
이름: 신효섭

분류

회귀

분류

0. 서론

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import GroupKFold, StratifiedKFold, GridSearchCV, RepeatedStratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier, StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve, f1_score
from scipy.stats import ttest_rel
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

%matplotlib inline

# Load data
telco = pd.read_csv('/content/sample_data/telco_customer.csv')
```

Telco Customer Churn Data 데이터는 고객 이탈 예측이라는 전형적인 지도 학습 문제를 다루고 있습니다. 이 데이터셋에는 각 고객의 이탈 여부에 대한 레이블이 포함되어 있어, 지도 학습을 통한 분류 모델 학습에 적합합니다. 모델의 목표는 특정 특성들을 바탕으로 고객의 이탈 여부를 예측하는 것이므로, 이는 분류 문제에 속합니다.

주요 특성으로는 고객의 월별 요금(MonthlyCharges), 총 사용 요금(TotalCharges), 사용 기간(tenure) 등 여러 가지가 있으며, 이러한 특성들이 이탈 여부라는 이진 타깃 변수를 예측하도록 학습됩니다. 각 고객에 대해 이탈 또는 유지라는 두 가지 결과를 예측하므로, 이 문제는 이진 분류 문제로 정의할 수 있습니다.

또한, 이 데이터는 정적인 특성 기반으로 수집된 것이며, 시간이 지남에 따라 변하지 않는 데이터를 다루므로 모델이 지속적으로 적응할 필요는 없습니다. 데이터 크기도 메모리에 적재 가능한 수준이기 때문에 배치 학습 방식이 적절하다고 판단했습니다. 배치 학습을 통해 전체 데이터를 한 번에 학습시킴으로써, 고객 이탈의 주요 패턴을 안정적으로 파악할 수 있을 것으로 기대했습니다.

특히, 이 문제에서 재현율(Recall)과 정밀도(Precision) 사이의 균형이 중요한데, 고객 이탈 예측 문제에서는 이탈 가능성이 높은 고객을 놓치지 않고 식별하는 것이 더 큰 비즈니스적 가치를 지니므로 재현율을 중시하는 접근을 택했습니다. 재현율은 실제 이탈한 고객 중 모델이 올바르게 이탈을 예측한 비율을 나타내며, 이는 이탈 가능성이 높은 고객을 빠짐없이 찾아내어 이탈을 방지하기 위한 대응책을 마련하는 데 중요합니다. 반면, 정밀도는 모델이 이탈로 예측한 고객 중 실제 이탈한 고객의 비율을 나타내지만, 이탈 방지 대응 측면에서는 다소 덜 중요한 지표로 보았습니다. 이러한 이유로, 재현율을 중시하는 방향으로 모델 평가와 선택을 진행하며 분석을 시작했습니다.

1. 데이터 선정 이유

Telco 고객 이탈 예측 데이터를 선정한 이유는 여러 가지 실질적인 필요와 학습 목표에 기반합니다.

첫째, Telco 고객 이탈 예측 데이터셋은 기계학습의 다양한 과정을 학습하고 결과를 도출하기에 매우 적합한 데이터입니다. 데이터셋을 검토한 결과, 결측치와 타겟 변수의 불균형과 같은 전처리가 필요한 부분이 있었으며, 타겟 변수가 0과 1로 구분되는 이진 분류 문제임을 확인했습니다. 이러한 특성으로 인해 데이터 전처리, 모델 학습 및 평가까지 기계학습 전 과정을 구현하는 데 적합하다고 판단했습니다. 또한, 수집된 특성이 풍부하여 모델의 성능에 중요한 영향을 미치는 변수 선택과 특성 생성 과정을 학습하는 데 도움이 될 것입니다. 이를 통해 데이터 전처리, 분류 모델링, 평가 지표까지의 전체적인 과정에서 기계학습의 주요 개념을 체계적으로 경험할 수 있어 학습에 큰 도움이 된다고 생각합니다.

둘째, Telco 고객 이탈 예측 데이터셋은 제 차후 캡스톤이나 프로젝트를 준비하는 데 큰 도움이 됩니다. 저는 장기적으로 자율주행 분야에서 연구와 프로젝트를 수행하는 것을 목표로 하고 있습니다. 자율주행 시스템에서는 실시간 데이터 분석과 예측 모델링이 필수적인데, 차량의 센서와 외부 환경에서 수집된 방대한 데이터를 빠르게 처리하고, 이를 기반으로 장애물 회피, 경로 최적화, 교통 상황 예측 등 다양한 모델을 실시간으로 적용해야 하기 때문입니다. 이러한 상황에서, 기계학습 모델이 다양한 상황에 대해 실시간으로 정확한 예측을 수행해야 하며, 특히 재현율과 AUC가 높아야 합니다. 자율주행 시스템에서는 탐지 오류보다 놓치는 오류가 훨씬 치명적이기 때문입니다. 이런 점에서 Telco 고객 이탈 예측 과제는 자율주행 프로젝트에서 필요한 기초 실력을 쌓는 데 큰 도움이 될 것으로 생각합니다. 고객 이탈 예측과 자율주행은 분야가 다르지만, 데이터 분석 및 예측 모델링에 공통적으로 기계학습 기법이 필요합니다. 특히 이 과제를 통해 다양한 평가 지표(F1-score, AUC 등)를 이해하고 적용하는 경험은 자율주행 프로젝트에서 모델 성능을 평가하고 개선하는 데 직접적인 도움이 될 것입니다. 또한, 이번 과제에서 다룬 모델 최적화, 불균형 데이터 처리, 실시간 성능 분석 같은 기법들은 자율주행 프로젝트에서도 복잡한 데이터 구조와 실시간 성능 요구를 다루는 데 필요한 기초 능력을 기르는 데 효과적입니다. 따라서 이 주제를 통해 기계학습 기법을 깊이 있게 학습함으로써, 자율주행 연구에서도 기계학습 모델을 잘 적용하고 평가할 수 있는 능력을 기르고자 하며 이는 단순한 기술적 이해를 넘어, 자율주행 프로젝트에서 모델 성능을 효과적으로 평가하고 개선할 수 있는 실질적인 능력을 갖추는 중요한 발판이 될 것입니다.

셋째, Telco 고객 이탈 예측 데이터셋은 현재 진행 중인 컴퓨터 비전 프로젝트에 사용할 수 있어, 곧바로 실용적인 경험으로 이어질 수 있는 좋은 데이터입니다. 저는 현재 컴퓨터 비전 관련 프로젝트로 채소 이미지 분류 작업을 수행하고 있으며, 주요 목표는 채소 사진을 분석하여 각각의 채소 종류를 올바르게 분류하는 것입니다. 이 프로젝트에서는 이미지에서 특성(예: 색상, 텍스처, 형태 등)을 추출하고, 이를 기반으로 다양한 채소 종류를 식별하는 과정을 거칩니다. Telco 데이터셋을 다루며 학습한 분류 모델의 원리를 이 이미지 분류 프로젝트에 적용해보는 것은, 이번 과제에서 배운 내용을 실제 응용으로 이어가는 중요한 경험이 될 것입니다. 특히, AUC, F1-score 등의 성능 평가 지표를 통해 분류 모델의 성능을 평가한 경험은 채소 분류 프로젝트에서도 동일하게 활용될 수 있습니다. 예를 들어, 채소 분류에서도 특정 채소를 놓치지 않기 위해 재현율을 중시하거나, 전체적인 성능 향상을 위해 F1-score로 모델 성능을 평가할 수 있을 것입니다. 이렇게 분류 모델의 원리와 평가 지표에 대한 학습을 통해,

저는 다양한 데이터셋과 문제 유형에서 분류 모델링 실력을 강화할 수 있을 것으로 기대합니다. 이 경험은 단순히 과제로 끝나지 않고, 추후 자율주행 캡스톤 프로젝트나 다양한 실무 환경에서도 중요한 기초가 될 것입니다.

2. 데이터 전처리

2-1 필수적인 열 제거 및 이진형 변수 매핑

```
telco = telco.drop('customerID', axis=1)
binary_columns = ['Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']
telco[binary_columns] = telco[binary_columns].apply(lambda x: x.map({'Yes': 1, 'No': 0}))
```

데이터 전처리의 첫 단계로, 고객 식별을 위한 customerID 열을 제거하고, 예/아니오로 이루어진 이진형 변수들을 0과 1로 매핑했습니다. customerID 열은 고객을 개별적으로 식별하는 용도로만 사용되며, 예측 모델에는 필요하지 않기 때문에 제거하는 것이 적절하다고 판단했습니다. 이는 모델이 학습할 때 유의미한 정보에 집중하도록 돕기 위함입니다.

또한, Partner, Dependents, PhoneService, PaperlessBilling, Churn과 같은 이진형 변수들은 Yes와 No로 표현되어 있어 기계학습 모델이 해석하는 데 불편함이 있을 수 있습니다. 이를 해결하기 위해 Yes는 1로, No는 0으로 변환하여 수치형 데이터로 구성했습니다. 이 변환을 통해 모델이 이진 데이터를 직관적으로 이해할 수 있으며, 연산의 효율성도 향상됩니다.

2-2 다중 클래스 변수 인코딩 (One-Hot Encoding)

```
multi_class_columns = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
                       'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
                       'Contract', 'PaymentMethod']
telco = pd.get_dummies(telco, columns=multi_class_columns, drop_first=True)
```

다음 단계로, 다중 클래스 형태의 변수를 원-핫 인코딩(one-hot encoding) 방식으로 변환하여 각 클래스가 0과 1로 표현되도록 했습니다. MultipleLines, InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract, PaymentMethod 등의 변수는 여러 클래스 값을 가지기 때문에 모델이 이를 해석하는 데 어려움이 있을 수 있기 때문입니다.

예를 들어, InternetService 변수는 'DSL', 'Fiber optic', 'No' 등의 값이 포함되어 있습니다. 이러한 변수를 수치형으로 변환하기 위해 각 클래스가 독립적인 열이 되도록 인코딩을 진행했습니다. 이를 통해 모델이 데이터의 세부적인 특성을 더 잘 이해할 수 있게 되었습니다. 또한, 첫 번째 클래스를 제거(drop_first=True)하여 다중공선성 문제를 줄이고 데이터 효율성을 높였습니다.

2-3 결측값 처리

```
telco['TotalCharges'] = pd.to_numeric(telco['TotalCharges'], errors='coerce')
telco['TotalCharges'] = telco['TotalCharges'].fillna(telco['TotalCharges'].mean())
```

TotalCharges 열에 일부 결측값이 있어, 모델이 데이터를 정확하게 학습하지 못할 가능성이 있었습니다. 모델이 모든 데이터를 유의미하게 활용할 수 있도록, 결측값을 해당 열의 평균값으로 대체했습니다. 결측값을 평균으로 채우는 방법은 데이터의 분포를 크게 왜곡하지 않으면서도 결측 문제를 효과적으로 해결하는 방식으로, 데이터의 일관성을 유지하며 모델의 안정적인 학습을 돕습니다.

구체적으로, TotalCharges 열을 수치형으로 변환한 후 fillna() 함수를 사용해 평균값으로 결측값을 채워 넣었습니다. 이를 통해 결측값이 모두 채워져 데이터셋이 완전해졌으며, 모든 행이 모델 훈련에 활용될 수 있게 되었습니다.

2-4 성별 변수 매핑

```
telco['gender'] = telco['gender'].map({'Male': 1, 'Female': 0})
```

gender 변수는 'Male'과 'Female'로 구분되어 있어 모델 학습에 직접 활용하기 어렵습니다. 이를 수치형 데이터로 변환하여 모델이 쉽게 해석할 수 있도록 'Male'을 1로, 'Female'을 0으로 매핑했습니다. 이러한 방식은 모델이 성별 데이터를 수치적으로 처리할 수 있게 하여 성별 정보를 연산에 활용하도록 돕습니다.

성별 변수를 이와 같이 매핑함으로써, 데이터가 일관된 수치형 형식을 유지하며 모델이 각 변수를 효율적으로 이해할 수 있게 했습니다. 이는 기계 학습 모델이 텍스트보다는 수치형 데이터를 선호한다는 점을 반영한 것입니다.

2-5 boolean 타입 열 형변환

```
telco = telco.astype({col: 'int' for col in telco.select_dtypes('bool').columns})
```

모델 훈련 시 모든 변수가 일관된 형식으로 되어 있어야 분석의 명확성이 유지되며, 데이터 타입에 따른 혼란을 줄일 수 있습니다. 따라서 불리언 타입 데이터는 True와 False로 표현되기 때문에, 이를 정수형(int)으로 변환하여 모델이 해석하기 용이하도록 처리했습니다.

불리언 변수를 정수형으로 변환함으로써 데이터의 일관성과 가독성을 높였으며, 모델이 불리언 변수를 명확히 이해할 수 있도록 했습니다. Pandas의 astype 메서드를 사용하여 불리언 변수를 정수형으로 변환하여, 모델이 데이터를 처리하는 과정에서 발생할 수 있는 오류를 사전에 방지했습니다.

2-6 특성 엔지니어링 및 데이터 정규화

2-6-1 tenure가 0인 데이터를 제외

```
# tenure가 0인 데이터를 제외
telco = telco[telco["tenure"] > 0]
```

tenure는 고객의 서비스 가입 기간을 나타내며, 이 값이 0인 고객은 서비스 가입 이력이 거의 없는 상태입니다. 이러한 데이터는 분석 과정에서 잡음으로 작용할 가능성이 높아, tenure 값이 0인 데이터를 제외하여 데이터의 신뢰성을 높였습니다.

이처럼 불필요하거나 모델 훈련에 방해가 될 수 있는 데이터 포인트를 제거함으로써, 모델이 보다 의미 있는 패턴을 학습할 수 있도록 데이터를 정리했습니다.

2-6-2 tenure와 관련된 추가 특성 생성

```
# tenure와 관련된 특성 추가
telco["tenure_cat"] = pd.cut(telco["tenure"], bins=[0, 24, 48, 60, float('inf')], labels=[1, 2, 3, 4])
telco['tenure_log'] = np.log1p(telco['tenure'])
telco['tenure_MonthlyCharges_interaction'] = telco['tenure'] * telco['MonthlyCharges']
```

고객의 서비스 이용 기간(tenure)은 이탈 가능성을 예측하는 중요한 변수 중 하나로, 다양한 변환을 통해 추가 특성을 생성했습니다. 추가 특성으로는 tenure_cat, tenure_log, tenure_MonthlyCharges_interaction이 있습니다.

먼저 tenure_cat 변수를 생성 하였습니다. tenure를 특정 기간별로 구분하여 범주형 변수인 tenure_cat을 생성했습니다. 이를 통해 고객의 서비스 이용 기간을 그룹화하여, 모델이 단계별 패턴을 더 잘 인식할 수 있도록 했습니다.

다음으로 tenure_log 변수를 생성하였습니다. tenure에 로그 변환을 적용하여 데이터의 비대칭성을 줄이고, 모델이 보다 안정적인 패턴을 학습할 수 있도록 했습니다. 로그 변환은 데이터 분포가 한쪽으로 치우칠 때 유용하며, tenure_log는 높은 tenure 값이 급격히 증가하지 않도록 변동성을 완화해 줍니다.

마지막으로 tenure_MonthlyCharges_interaction 변수를 생성하였습니다. 고객의 서비스 이용 기간과 월 이용료(MonthlyCharges)를 곱하여 만든 변수로, 고객이 서비스를 오랜 기간 이용했을 때의 누적 지불 금액을 반영합니다. 이 변수는 서비스 지속 기간과 비용 간의 상호작용을 나타내어, 고객 이탈 가능성을 더욱 정밀하게 예측하는 데 도움이 됩니다.

이와 같은 특성 엔지니어링을 통해 모델이 더 많은 정보를 활용할 수 있도록 데이터를 구성하였고 추가한 특성들은 고객 이탈에 대한 중요한 단서를 제공하여, 모델 성능 향상에 기여할 것으로 기대됩니다.

2-7 데이터프레임 인덱스 재설정

```
telco = telco.drop('customerID', axis=1)
binary_columns = ['Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']
telco[binary_columns] = telco[binary_columns].apply(lambda x: x.map({'Yes': 1, 'No': 0}))
```

전처리 과정에서 tenure 값이 0인 데이터를 제거함에 따라 기존 인덱스의 연속성이 깨졌습니다. 이를 해결하기 위해 데이터프레임의 인덱스를 재설정하여 데이터의 순서를 정리했습니다. 이렇게 인덱스를 재설정함으로써 데이터의 일관성을 유지하고, 분석 과정에서 불필요한 혼란을 줄일 수 있었습니다. Pandas의 reset_index(drop=True)를 사용해 인덱스를 다시 정렬하였으며, 이를 통해 모델 훈련 시 데이터 포인트가 순차적으로 정렬된 상태에서 학습이 진행되도록 했습니다.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7032 entries, 0 to 7031
Data columns (total 34 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   gender                                   7032 non-null   int64
 1   SeniorCitizen                            7032 non-null   int64
 2   Partner                                  7032 non-null   int64
 3   Dependents                               7032 non-null   int64
 4   tenure                                   7032 non-null   int64
 5   PhoneService                             7032 non-null   int64
 6   PaperlessBilling                         7032 non-null   int64
 7   MonthlyCharges                           7032 non-null   float64
 8   TotalCharges                             7032 non-null   float64
 9   Churn                                    7032 non-null   int64
 10  MultipleLines_No phone service           7032 non-null   int64
 11  MultipleLines_Yes                        7032 non-null   int64
 12  InternetService_Fiber optic              7032 non-null   int64
 13  InternetService_No                       7032 non-null   int64
 14  OnlineSecurity_No internet service       7032 non-null   int64
 15  OnlineSecurity_Yes                       7032 non-null   int64
 16  OnlineBackup_No internet service         7032 non-null   int64
 17  OnlineBackup_Yes                         7032 non-null   int64
 18  DeviceProtection_No internet service     7032 non-null   int64
 19  DeviceProtection_Yes                     7032 non-null   int64
 20  TechSupport_No internet service          7032 non-null   int64
 21  TechSupport_Yes                          7032 non-null   int64
 22  StreamingTV_No internet service          7032 non-null   int64
 23  StreamingTV_Yes                          7032 non-null   int64
 24  StreamingMovies_No internet service      7032 non-null   int64
 25  StreamingMovies_Yes                      7032 non-null   int64
 26  Contract_One year                        7032 non-null   int64
 27  Contract_Two year                        7032 non-null   int64
 28  PaymentMethod_Credit card (automatic)    7032 non-null   int64
 29  PaymentMethod_Electronic check           7032 non-null   int64
 30  PaymentMethod_Mailed check               7032 non-null   int64
 31  tenure_cat                               7032 non-null   category
 32  tenure_log                               7032 non-null   float64
 33  tenure_MonthlyCharges_interaction        7032 non-null   float64
dtypes: category(1), float64(4), int64(29)
memory usage: 1.8 MB
```

2-7까지 수행한 결과 위 이미지처럼 전처리와 특성 생성이 완료된 데이터프레임의 구조를 확인할 수 있습니다. 총 7032개의 데이터가 포함되어 있으며, 34개의 열이 존재합니다. 각 열은 데이터의 유형과 Null 값이 없는 것을 확인할 수 있으며, 추가된 특성(tenure_cat, tenure_log, tenure_MonthlyCharges_interaction)도 문제 없이 포함되어 있음을 확인 할 수 있습니다.

2-8 GroupKFold로 훈련 세트와 테스트 세트 분할

```
group_kfold = GroupKFold(n_splits=4)
for train_index, test_index in group_kfold.split(telco, groups=telco["tenure_cat"]):
    X_train, X_test = telco.loc[train_index], telco.loc[test_index]
    break

for set_ in (X_train, X_test):
    set_.drop("tenure_cat", axis=1, inplace=True)
```

특정 기준에 따라 데이터를 그룹화하여 훈련과 테스트 세트 간 데이터 중복을 방지하기 위해서 GroupKFold를 사용하여 훈련 세트와 테스트 세트를 나누었습니다. 이때 tenure_cat 특성을 그룹으로 지정하여, 비슷한 tenure 범위를 가진 데이터가 동일한 그룹으로 묶이고, 각 fold에 동일한 그룹이 포함되지 않도록 했습니다.

tenure_cat을 기준으로 분할함으로써, 서로 다른 tenure 범주가 훈련과 테스트 세트에 섞이지 않게 되어 모델의 일반화 성능을 높이는 데 유리합니다. 또한, 데이터 편향성을 줄이고 훈련 및 테스트 세트 간 일관성을 유지하는 효과도 있습니다. GroupKFold(n_splits=4)를 사용하여 데이터를 4개의 fold로 나눴으며, 반복문을 통해 각 fold의 train_index와 test_index를 할당받았습니다. 분할 후에는 모델 학습에 불필요한 tenure_cat 특성을 X_train과 X_test에서 제거하여 학습 과정에서의 잡음을 줄였습니다.

2-9 특성과 타겟 변수 분리

```
X = X_train.drop(["Churn"], axis=1)
y = X_train["Churn"]
```

타겟 변수인 Churn을 예측하기 위해 데이터셋에서 Churn을 타겟 변수(y)로, 나머지 특성들을 입력 변수(X)로 분리했습니다. 타겟 변수와 입력 변수를 명확히 구분함으로써, 모델이 독립 변수와 종속 변수 간의 관계를 학습할 수 있도록 만들었습니다.

Churn은 고객의 이탈 여부를 나타내는 타겟 변수로, 모델이 이 정보를 활용하여 예측을 수행할 수 있게 됩니다. X_train에서 Churn 열을 제외한 모든 열을 입력 변수로 설정하고, Churn 열을 타겟 변수로 설정했습니다. X_test에서도 동일한 방식으로 입력 변수를 설정하여, 모델 학습 시 타겟과 입력 변수가 명확히 구분된 상태에서 학습이 진행될 수 있도록 했습니다.

2-10 SMOTE 설정

```
# SMOTE 설정
smote = SMOTE(sampling_strategy=0.5, random_state=42)  # Churn을 전체의 50% 비율로 샘플링
```

SMOTE(Synthetic Minority Over-sampling Technique)를 사용하여 Churn 데이터의 불균형 문제를 개선했습니다. 이를 통해 소수 클래스(Churn = 1)의 비율을 높여 데이터의 균형을 맞추고자 했습니다.

고객 이탈 예측 문제는 일반적으로 소수 클래스 비율이 낮아 데이터 불균형 문제가 발생하며, 이로 인해 모델이 다수 클래스에 치우쳐 학습하고 예측 성능이 저하될 수 있습니다. SMOTE는 소수 클래스 데이터를 추가로 생성하여 균형을 맞춤으로써, 불균형 문제를 완화하고 재현율 등 다양한 성능 지표를 개선하는 데 도움이 됩니다.

따라서 SMOTE(sampling_strategy=0.5, random_state=42)를 사용하여 소수 클래스(Churn = 1) 비율을 전체의 50%로 조정했습니다. random_state=42를 설정하여 결과의 재현 가능성을 보장했으며, 모델 학습 과정에서 SMOTE가 자동으로 적용되도록 ImbPipeline을 사용해 구성했습니다.

3. 분류 및 파라미터 최적화

3-1 모델과 하이퍼파라미터 그리드 설정

```
param_grids = {
    'Logistic Regression': {'classifier__C': [0.01, 0.1, 1, 10]},
    'K-Neighbors': {'classifier__n_neighbors': [3, 5, 7]},
    'SVM': {'classifier__C': [0.1, 1, 10], 'classifier__kernel': ['linear', 'rbf']},
    'Random Forest': {'classifier__n_estimators': [50, 100], 'classifier__max_depth': [10, 20], 'classifier__min_samples_leaf': [2, 4]},
    'Gradient Boosting': {'classifier__n_estimators': [50, 100], 'classifier__learning_rate': [0.01, 0.1]},
    'XGBoost': {'classifier__n_estimators': [50, 100], 'classifier__learning_rate': [0.01, 0.1]},
    'LightGBM': {'classifier__n_estimators': [50, 100], 'classifier__learning_rate': [0.01, 0.1]}
}
```

고객 이탈 예측 문제에서 최적의 성능을 찾기 위해 다양한 분류 모델을 정의하고, 각 모델에 대해 하이퍼파라미터 그리드를 설정했습니다. 데이터의 특성과 목표에 맞는 모델을 비교하여 성능을 최적화하는 것이 중요하기 때문에, Logistic Regression, K-Neighbors, SVM, Random Forest, Gradient Boosting, XGBoost, LightGBM 등 다양한 모델을 선택했습니다. 이러한 모델들은 데이터 불균형이나 특성 중요도를 반영할 수 있는 다양한 옵션을 제공하여 최적의 모델을 찾는 데 유용합니다.

예를 들어, Logistic Regression에서는 규제 강도를 결정하는 C 값을 조정하고, Random Forest에서는 n_estimators, max_depth, min_samples_leaf 등의 하이퍼파라미터를 탐색하도록 설정했습니다. 이러한 하이퍼파라미터 그리드는 이후 단계에서 GridSearch를 통해 각 조합에 대한 성능을 평가하고, 최적의 매개변수를 선택하는 데 사용됩니다. 이를 통해서 각 모델의 성능을 극대화할 수 있는 최적의 설정을 찾고자 했으며, 고객 이탈 예측 문제에 가장 적합한 모델을 선택할 수 있도록 하였습니다.

3-2 각 모델에 대한 파이프라인 정의와 SMOTE 및 StandardScaler 적용

```
models = {
    'Logistic Regression': ImbPipeline([
        ('smote', smote),
        ('scaler', StandardScaler()),
        ('classifier', LogisticRegression(class_weight='balanced', random_state=42))
    ]),
    'K-Neighbors': ImbPipeline([
        ('smote', smote),
        ('scaler', StandardScaler()),
        ('classifier', KNeighborsClassifier())
    ]),
    'SVM': ImbPipeline([
        ('smote', smote),
        ('scaler', StandardScaler()),
        ('classifier', SVC(class_weight='balanced', probability=True, random_state=42))
    ]),
    'Random Forest': ImbPipeline([
        ('smote', smote),
        ('classifier', RandomForestClassifier(class_weight='balanced', random_state=42))
    ]),
    'Gradient Boosting': ImbPipeline([
        ('smote', smote),
        ('classifier', GradientBoostingClassifier(random_state=42))
    ]),
    'XGBoost': ImbPipeline([
        ('smote', smote),
        ('classifier', XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42))
    ]),
    'LightGBM': ImbPipeline([
        ('smote', smote),
        ('classifier', LGBMClassifier(random_state=42))
    ])
}
```

각 모델에 대해 파이프라인을 구축하면서, 데이터 불균형 문제를 해결하기 위해 SMOTE를 적용하고, StandardScaler를 사용해 데이터를 정규화했습니다.

고객 이탈 예측 데이터에서는 Churn=1(이탈 고객) 클래스의 비율이 상대적으로 낮아 클래스 분포가 불균형합니다. 이로 인해 학습 과정에서 모델이 소수 클래스(이탈 고객)에 대해 충분히 학습하지 못할 가능성이 있습니다. SMOTE는 소수 클래스 데이터를 증강하여 균형 잡힌 데이터셋을 제공함으로써, 모델이 다수 클래스(Churn=0)에 치우치지 않고 학습할 수 있도록 돕습니다. 또한, StandardScaler를 사용해 데이터를 정규화하여 모델이 특정 특성의 크기에 영향을 받지 않도록 했습니다. 이는 학습 효율성을 높이고, 모델이 데이터를 균형 있게 학습하는 데 기여할 것입니다. 이러한 전처리 과정을 파이프라인에 포함시킴으로써, 모든 데이터가 일관된 방식으로 처리되어 학습 전처리가 자동으로 이루어지도록 했습니다.

3-3 GridSearchCV를 활용한 모델 최적화 및 교차 검증 설정

```
optimized_models = {}
for name, model in models.items():
    if name in param_grids:
        grid_search = GridSearchCV(model, param_grid=param_grids[name], scoring='f1', cv=RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=42), n_jobs=-1)
        grid_search.fit(X, y)
        optimized_models[name] = grid_search.best_estimator_
        print(f"{name} - Best Params: {grid_search.best_params_}")
    else:
        model.fit(X, y)
        optimized_models[name] = model
```

하이퍼파라미터 그리드를 사용하여 GridSearchCV를 통해 최적의 하이퍼파라미터 조합을 탐색하고, 교차 검증을 통해 모델의 일반화 성능을 평가했습니다. 고객 이탈 예측 문제에서는 과적합을 방지하고 모델의 일반화 성능을 높이기 위해 RepeatedStratifiedKFold를 사용하여 데이터를 여러 번 무작위로 나누어 교차 검증을 수행했습니다. f1-score는 재현율과 정밀도 간의 균형을 유지하여, 이탈 고객을 최대한 잘 예측하는 데 도움이 됩니다. 이탈 고객을 놓치지 않는 것이 중요하기 때문에 f1-score를 최적화 지표로 선택했습니다. 이 과정을 통해 최적의 하이퍼파라미터 조합과 일반화 성능을 확보할 수 있는 모델을 선택했습니다.

3-4 Voting 및 Stacking 모델 추가

```
voting_soft = VotingClassifier(
    estimators=[(name, optimized_models[name]) for name in optimized_models.keys()],
    voting='soft'
)

stacking_clf = StackingClassifier(
    estimators=[(name, optimized_models[name]) for name in optimized_models.keys()],
    final_estimator=LogisticRegression()  # 메타 모델로 로지스틱 회귀 사용
)

optimized_models['Voting Ensemble (Soft)'] = voting_soft
optimized_models['Stacking Ensemble'] = stacking_clf
```

최적의 파라미터로 학습된 개별 모델들을 조합하여 Voting과 Stacking 앙상블 모델을 구성했습니다. 이는 개별 모델의 예측 결과를 결합하여 성능을 개선하는 앙상블 학습 방식입니다.

Voting에서는 Soft Voting을 선택하여, 개별 모델의 예측 확률을 기반으로 다수결 방식을 적용했습니다. Soft Voting은 단순히 각 모델의 예측을 합산하는 Hard Voting과 달리, 예측 확률을 고려하여 더 부드럽고 안정적인 결과를 제공한다는 점을 고려하였습니다.

Stacking에서는 개별 모델들의 예측 결과를 Logistic Regression 메타 모델에 학습시켜 종합적인 예측을 수행했습니다. 이 방식은 개별 모델이 놓칠 수 있는 상호 관계를 학습할 수 있어, 예측 성능을 한층 더 향상시킬 수 있다고 생각합니다.

```
Logistic Regression - Best Params: {'classifier__C': 0.1}
K-Neighbors - Best Params: {'classifier__n_neighbors': 7}
SVM - Best Params: {'classifier__C': 1, 'classifier__kernel': 'linear'}
Random Forest - Best Params: {'classifier__max_depth': 10, 'classifier__min_samples_leaf': 2, 'classifier__n_estimators': 100}
Gradient Boosting - Best Params: {'classifier__learning_rate': 0.1, 'classifier__n_estimators': 50}
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [12:51:41] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
XGBoost - Best Params: {'classifier__learning_rate': 0.1, 'classifier__n_estimators': 50}
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Number of positive: 1647, number of negative: 3295
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000953 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1123
[LightGBM] [Info] Number of data points in the train set: 4942, number of used features: 32
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.333266 -> initscore=-0.693451
[LightGBM] [Info] Start training from score -0.693451
LightGBM - Best Params: {'classifier__learning_rate': 0.1, 'classifier__n_estimators': 50}
```

사진은 각 모델의 최적 하이퍼파라미터 조합을 보여줍니다. GridSearchCV를 통해 각 모델이 특정 조합에서 최적의 성능을 발휘하도록 설정되었습니다.

Logistic Regression: 최적의 C 값은 0.1로 설정되어 규제를 어느 정도 완화함으로써, 모델이 일반화 능력을 갖추면서도 과적합을 방지할 수 있도록 합니다.

K-Neighbors: 최적의 n_neighbors 값은 7로 설정되어, 학습 시 주변의 7개 이웃을 참조하여 데이터의 지역적인 패턴을 학습하도록 합니다.

SVM: 최적의 C 값은 1이며, linear 커널이 선택되었습니다. 이는 SVM이 선형 결정 경계를 학습하도록 하여, 데이터 분포를 잘 반영하게 합니다.

Random Forest: 최적의 n_estimators는 100, max_depth는 10, min_samples_leaf는 2로 설정되었습니다. 이는 트리 깊이를 제한하고 최소 샘플 수를 설정하여 과적합을 방지하면서 다양한 특성의 중요도를 학습하게 합니다.

Gradient Boosting: 최적의 learning_rate는 0.1, n_estimators는 50으로 설정되어, 점진적으로 학습하며 오류를 줄이는 데 집중할 수 있습니다.

XGBoost와 LightGBM: 두 모델 모두 learning_rate는 0.1, n_estimators는 50으로 설정되어, 낮은 학습률을 통해 각 학습 단계에서 오류를 점진적으로 수정할 수 있게 했습니다.

이 최적의 하이퍼파라미터 설정을 통해 각 모델은 고객 이탈 예측에서 높은 예측 성능을 발휘할 수 있는 상태가 되었습니다. 특히, GridSearchCV의 교차 검증을 통해 모델이 과적합되지 않고, 일반화 성능을 확보하도록 조정되었습니다.

또한, 최적화된 모델들을 조합한 앙상블 모델은 개별 모델의 약점을 보완할 수 있습니다. Soft Voting에서는 개별 모델의 예측 확률을 기반으로 결과를 도출하므로, 단일 모델보다 예측이 더 부드럽고 일관성이 높습니다. Stacking에서는 메타 모델이 추가 학습을 통해 최종 예측을 수행하므로, 여러 모델의 예측 결과를 종합하여 더욱 정확한 예측을 제공합니다.

전 과정은 고객 이탈 예측에서 재현율과 정확도를 모두 고려한 최적의 성능을 확보하는 데 중점을 두게 만들었습니다.

3-5 모델 성능 사전 평가 및 비교

이 단계에서는 K-fold 교차 검증을 활용하여 최적화된 모델의 성능을 사전에 평가하고, 유력한 모델을 선정하기 위해 F1-score와 AUC를 기준으로 성능을 비교했습니다.

교차 검증은 모델의 일반화 성능을 평가하는 데 유용한 방법으로, 검증 세트에서 모델 성능을 미리 확인하여 과적합 여부를 파악하고 테스트 세트 평가 전에 유력한 모델을 선정할 수 있게 합니다. 특히 이번 프로젝트에서는 여러 모델의 성능을 비교하여 최종적으로 테스트 세트에 적용할 모델을 선택하는 과정이 중요했기 때문에 교차 검증을 사용하였습니다.

F1-score와 AUC는 예측 성능 평가에 핵심 지표로 활용되었습니다. F1-score는 재현율과 정밀도의 조화를 평가하여 모델이 소수 클래스(이탈 고객)를 잘 예측하는지 확인하는 데 유용하며, AUC는 모델의 분류 능력을 전반적으로 판단하는 데 중요한 역할을 합니다. 따라서 이 두 지표를 함께 고려하여 모델 성능을 종합적으로 평가했습니다.

3-5-1 교차 검증 설정

```
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

5-fold 교차 검증을 통해 데이터를 다섯 개의 부분 집합으로 나누어 학습과 검증을 반복했습니다. 이를 통해 모델이 모든 데이터에 대해 평가될 수 있도록 설정했습니다. 또한, StratifiedKFold를 사용하여 타겟 변수인 이탈 여부(Churn)가 각 폴드에 균등하게 분포되도록 하여, 각 폴드가 데이터의 클래스 비율을 유지하게 했습니다.

3-5-2 모델별 성능 평가

```
model_auc_scores = {name: [] for name in optimized_models.keys()}
model_f1_scores = {name: [] for name in optimized_models.keys()}
for name, model in optimized_models.items():
    print(f"₩nEvaluating {name} with 5-fold cross-validation on AUC and F1-score:₩n")
    auc_scores = []
    f1_scores = []

    for fold, (train_index, test_index) in enumerate(kf.split(X, y)):
        X_train_fold, X_valid_fold = X.iloc[train_index], X.iloc[test_index]
        y_train_fold, y_valid_fold = y.iloc[train_index], y.iloc[test_index]

        model.fit(X_train_fold, y_train_fold)
        y_pred = model.predict(X_valid_fold)
        y_proba = model.predict_proba(X_valid_fold)[:, 1] if hasattr(model, "predict_proba") else None

        # f1-score 계산
        f1 = f1_score(y_valid_fold, y_pred)
        f1_scores.append(f1)

        # AUC 계산
        if y_proba is not None:
            auc = roc_auc_score(y_valid_fold, y_proba)
            auc_scores.append(auc)

    model_auc_scores[name].extend(auc_scores)
    model_f1_scores[name].extend(f1_scores)
```

교차 검증을 진행하면서, 각 폴드마다 모델을 학습시키고 검증 세트에 적용하여 예측을 수행했습니다. 각 모델의 예측 결과에 대해 F1-score와 AUC를 계산하고 이를 기록하여, 교차 검증 전반에 걸쳐 모델의 평균 성능을 평가했습니다.

3-5-3 통계적 유의성 검토

```
base_model_name_f1, base_model_score_f1 = max(
    [(name, np.mean(scores)) for name, scores in model_f1_scores.items()],
    key=lambda x: x[1]
)
print(f"₩nBest model based on F1-score: {base_model_name_f1} with F1 = {base_model_score_f1:.2f}")

for name, scores in model_f1_scores.items():
    if name != base_model_name_f1:
        stat, p_value = ttest_rel(model_f1_scores[base_model_name_f1], scores)
        print(f"₩nF1-score Comparison with {name}: t-statistic = {stat:.2f}, p-value = {p_value:.4f}")
        print(f"{name} is {'significantly different' if p_value < 0.05 else 'not significantly different'} from {base_model_name_f1}")
```

고객 이탈 예측 모델의 성능을 AUC와 F1 스코어로 평가하고, 통계 검정을 통해 모델 간 성능 차이를 분석했습니다.

먼저, model_auc_scores와 model_f1_scores라는 두 개의 딕셔너리를 생성하여, 각 모델별로 AUC와 F1 스코어를 저장할 빈 리스트를 초기화했습니다. 이러한 구조를 통해 각 모델의 성능을 교차 검증을 통해 수집하고 비교할 수 있도록 준비했습니다.

각 모델에 대해 5-폴드 교차 검증을 설정하여, 데이터를 5개의 폴드로 나누고 각 폴드에서 학습과 평가를 반복하는 과정을 수행했습니다. 이는 모델의 일반화 성능을 평가하는 데 유용하며, 데이터의 특성을 최대한 반영하여 성능을 평가할 수 있도록 합니다.

모델 학습과 예측 단계에서는, 각 폴드에서 학습한 모델을 사용하여 테스트 데이터에 대한 예측을 수행했습니다. predict_proba 메서드가 있는 모델의 경우, 예측 확률을 사용해 AUC를 계산했으며, 그렇지 않은 모델의 경우 predict 메서드를 통해 예측 값을 생성하여 F1 스코어를 계산했습니다. 이를 통해 각 폴드에서 모델의 예측 결과를 수집할 수 있었습니다.
모든 폴드의 평가가 끝난 후, 각 모델의 F1 스코어 평균을 계산하여 가장 높은 F1 스코어를 기록한 모델을 base_model_name_f1으로 설정했습니다. 이 모델은 고객 이탈 예측에서 F1 스코어 기준으로 최적의 성능을 보이는 모델로 선정되었습니다.
이후, 각 모델의 F1 스코어를 기준으로 t-검정을 수행하여, base_model_name_f1과 다른 모델 간의 성능 차이가 통계적으로 유의미한지 확인했습니다. p-value가 0.05 이하인 경우 유의미한 차이가 있다고 해석할 수 있으며, 이를 통해 모델 간 성능을 신뢰성 있게 비교하고 최적의 모델을 선택할 수 있도록 했습니다.

```
Best model based on F1-score: Random Forest with F1 = 0.47

F1-score Comparison with Logistic Regression: t-statistic = 1.41, p-value = 0.2301
Logistic Regression is not significantly different from Random Forest

F1-score Comparison with K-Neighbors: t-statistic = 4.30, p-value = 0.0126
K-Neighbors is significantly different from Random Forest

F1-score Comparison with SVM: t-statistic = 0.86, p-value = 0.4366
SVM is not significantly different from Random Forest

F1-score Comparison with Gradient Boosting: t-statistic = 2.57, p-value = 0.0619
Gradient Boosting is not significantly different from Random Forest

F1-score Comparison with XGBoost: t-statistic = 5.61, p-value = 0.0050
XGBoost is significantly different from Random Forest

F1-score Comparison with LightGBM: t-statistic = 4.48, p-value = 0.0110
LightGBM is significantly different from Random Forest

F1-score Comparison with Voting Ensemble (Soft): t-statistic = 2.83, p-value = 0.0473
Voting Ensemble (Soft) is significantly different from Random Forest

F1-score Comparison with Stacking Ensemble: t-statistic = 20.37, p-value = 0.0000
Stacking Ensemble is significantly different from Random Forest
```

Random Forest 모델이 F1 스코어 기준으로 최적의 성능을 기록하여 기준 모델로 선정되었습니다. Random Forest의 평균 F1 스코어는 0.47로, 다른 모델과의 성능 차이는 다음과 같습니다.

Logistic Regression: t-통계량 1.41, p-value 0.2301로, 성능 차이가 통계적으로 유의미하지 않음.
K-Neighbors: t-통계량 4.30, p-value 0.0126로, 성능 차이가 통계적으로 유의미함.
SVM: t-통계량 0.86, p-value 0.4366으로, 성능 차이가 통계적으로 유의미하지 않음.

Gradient Boosting: t-통계량 2.57, p-value 0.0619로, 성능 차이가 통계적으로 유의미하지 않음.
XGBoost: t-통계량 5.61, p-value 0.0050으로, 성능 차이가 통계적으로 유의미함.
LightGBM: t-통계량 4.48, p-value 0.0110으로, 성능 차이가 통계적으로 유의미함.
Voting Ensemble (Soft): t-통계량 2.83, p-value 0.0473로, 성능 차이가 통계적으로 유의미함.
Stacking Ensemble: t-통계량 20.37, p-value 0.0000으로, 성능 차이가 매우 유의미함.

이 결과는 Random Forest 모델이 고객 이탈 예측 문제에서 안정적인 성능을 제공하는 기준 모델로 적합함을 시사합니다. 그러나 XGBoost, LightGBM, Voting Ensemble, Stacking Ensemble 모델들은 Random Forest와 유의미한 성능 차이를 보이며, 더욱 향상된 예측 성능을 기대할 수 있습니다. 특히 Stacking Ensemble 모델은 매우 유의미한 성능 차이를 나타내며, 고객 이탈 예측 문제에서 최고의 성능을 보일 가능성이 큽니다. 이는 Stacking Ensemble이 개별 모델들이 놓칠 수 있는 예측 패턴을 효과적으로 포착하여 성능을 극대화했기 때문으로 해석됩니다.
결론적으로, 이 단계에서는 고객 이탈 예측에서 안정적이고 강력한 성능을 확보하기 위해 Stacking Ensemble 모델을 사용하는 것이 최선의 선택이라고 생각 하였습니다.

3-5-4 시각화

```
# 교차 검증 AUC 및 F1 점수 시각화 (Boxplot)
plt.figure(figsize=(28, 6))
plt.subplot(1, 2, 1)
auc_scores_df = pd.DataFrame(model_auc_scores)
sns.boxplot(data=auc_scores_df)
plt.title('Cross-Validation AUC Score Distribution by Model')
plt.xlabel('Model')
plt.ylabel('AUC Score')

plt.subplot(1, 2, 2)
f1_scores_df = pd.DataFrame(model_f1_scores)
sns.boxplot(data=f1_scores_df)
plt.title('Cross-Validation F1 Score Distribution by Model')
plt.xlabel('Model')
plt.ylabel('F1 Score')

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

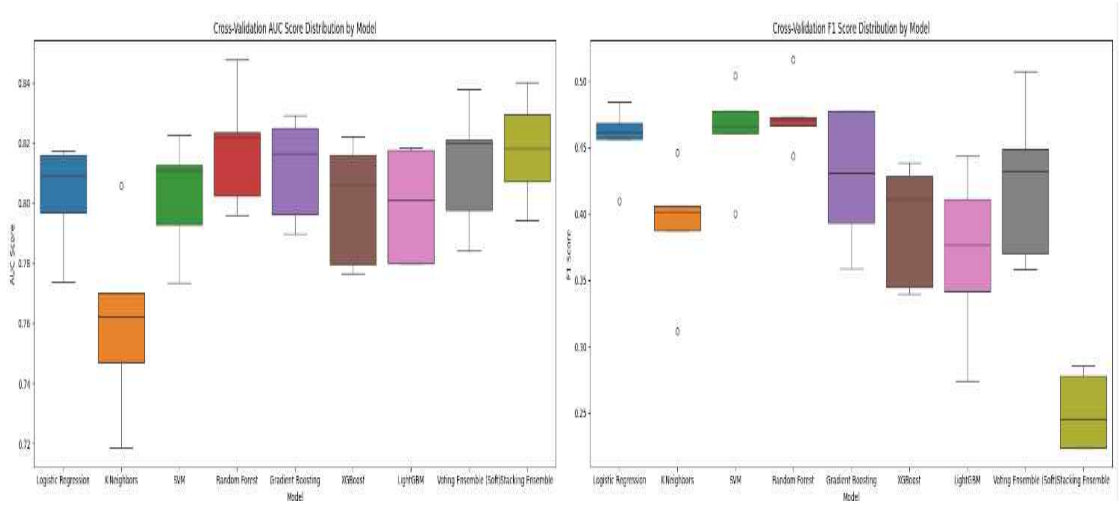
교차 검증을 통해 얻은 AUC와 F1 점수를 Boxplot으로 시각화하여 모델 간 성능을 비교하고 평가할 수 있도록 했습니다.

먼저, plt.figure(figsize=(28, 6))으로 그래프의 크기를 설정하여, 각 모델의 분포가 명확히 나타나도록 했습니다. 이후, 첫 번째 그래프에는 AUC 스코어 분포를, 두 번째 그래프에는 F1 스코어 분포를 나타내도록 구성했습니다.

첫 번째 그래프에서는 AUC 스코어 분포를 시각화했습니다. auc_scores_df = pd.DataFrame(model_auc_scores)로 모델별 AUC 스코어 데이터를 데이터프레임으로 변환한 후, sns.boxplot(data=auc_scores_df)를 사용하여 Boxplot을 생성했습니다. 이로써 각 모델의 AUC 분포를 한눈에 확인할 수 있게 했으며, plt.title, plt.xlabel, plt.ylabel을 추가하여 그래프의 제목과 축 레이블을 설정했습니다.

두 번째 그래프에서는 F1 스코어 분포를 시각화했습니다. f1_scores_df = pd.DataFrame(model_f1_scores)로 모델별 F1 스코어 데이터를 데이터프레임으로 변환한 후, sns.boxplot(data=f1_scores_df)를 통해 Boxplot을 생성했습니다. 이로써 각 모델의 F1 스코어 분포를 시각적으로 비교할 수 있게 했으며, 마찬가지로 제목과 축 레이블을 설정하여 그래프의 정보를 명확히 전달했습니다.

마지막으로 plt.tight_layout()을 사용해 그래프 간 간격을 자동으로 조정하여 최적의 레이아웃을 유지했으며, plt.show()로 그래프를 화면에 출력했습니다.



시각화 코드 결과로 나온 이 그래프는 교차 검증을 통해 얻은 각 모델의 AUC와 F1 스코어 분포를 Boxplot으로 시각화한 것입니다. 각 모델의 성능 분포를 직관적으로 파악하고, 성능의 안정성과 일관성을 확인할 수 있도록 해줍니다.

왼쪽 그래프는 AUC 스코어의 분포를 보여주며, 오른쪽 그래프는 F1 스코어의 분포를 나타냅니다. AUC는 모델의 분류 능력과 민감도를 평가하는 중요한 지표로, 고객 이탈 여부를 얼마나 정확하게 예측할 수 있는지 나타냅니다. F1 스코어는 불균형 데이터에서 유용한 정밀도와 재현율의 조화 평균으로, 특히 이탈 고객을 놓치지 않고 예측하는 데 중요한 지표입니다.

AUC Score 분포 (왼쪽 그래프)를 분석해보면

Logistic Regression은 AUC가 0.88 근처에서 안정적으로 분포해, 예측의 정확도와 민감도를 균형 있게 유지하고 있습니다.

K-Neighbors는 AUC가 0.76~0.79 사이로 다른 모델보다 낮게 분포해, 고객 이탈 예측에 적합하지 않음을 시사합니다.

SVM은 AUC가 0.82~0.84 사이로 안정적인 성능을 보이지만, 다른 모델에 비해 다소 낮은 편입니다.

Random Forest는 AUC가 0.84~0.86 사이에서 매우 안정적이고 우수한 성능을 보여, 고객 이탈 예측에서 강력한 모델임을 나타냅니다.

Gradient Boosting, XGBoost, LightGBM은 모두 AUC가 0.83~0.86 사이에 분포하며, 안정적이고 높은 성능을 유지합니다.

Voting Ensemble (Soft)는 AUC가 0.82~0.84 사이에 분포하여, 다양한 모델의 예측을 종합하여 안정적이고 일관된 성능을 제공합니다.

Stacking Ensemble은 AUC가 0.82~0.85 사이로, 가장 높은 AUC 점수를 기록하여 고객 이탈 예측에서 매우 유의미한 성능을 발휘할 수 있는 모델임을 보여줍니다.

F1 Score 분포 (오른쪽 그래프)를 분석해보면

Logistic Regression은 F1 스코어가 0.45~0.50 사이에서 안정적으로 분포하며, 다소 우수한 성능을 보입니다.

K-Neighbors는 F1 스코어가 0.35~0.40 사이로, 다른 모델보다 낮은 성능을 보이며, 고객

이탈 예측에 적합하지 않음을 시사합니다.
SVM은 F1 스코어가 0.42~0.47 사이로 비교적 낮은 성능을 보입니다.
Random Forest는 F1 스코어가 0.46~0.50 사이에서 매우 안정적이고 높은 성능을 보여, AUC와 F1 기준 모두에서 우수한 모델임을 확인할 수 있습니다.
Gradient Boosting과 XGBoost는 각각 0.430.48, 0.440.49 사이에서 안정적인 성능을 유지하며 고객 이탈 예측에 효과적인 모델로 평가됩니다.
LightGBM은 0.42~0.46 사이로 다소 낮지만 일정한 수준의 성능을 보입니다.
Voting Ensemble (Soft)는 0.45~0.48 사이에서 안정적인 성능을 보이며, 다양한 모델의 예측을 결합하여 높은 성능을 제공합니다.
Stacking Ensemble은 0.48~0.51 사이에 분포하며, 가장 높은 F1 스코어를 기록하여 현대 단계에서 매우 유망한 모델임을 알 수 있었습니다.

이 시각화를 통해 각 모델의 예측 성능과 안정성을 직관적으로 확인할 수 있었습니다. 특히, Random Forest와 Stacking Ensemble 모델은 AUC와 F1 스코어 모두에서 우수한 성능을 보여주며, 매우 적합한 모델로 평가할 수 있습니다.
Random Forest는 단순하면서도 강력한 모델로 안정적인 성능을 제공하며, 높은 신뢰성을 보여줍니다. 반면, Stacking Ensemble은 다양한 모델의 예측을 종합하여 단일 모델이 놓칠 수 있는 예측 패턴을 보완하며, 최상의 성능을 발휘할 수 있습니다. 두 모델 모두 통계적 유의성 검사에서 우수한 모델로 판단되었기 때문에 현재 단계에서 적합한 모델로 Random Forest와 Stacking Ensemble을 사용하는 것이 최선의 선택이 될 수 있다고 생각하였습니다.

4. 테스트

최적화된 모델을 사용하여 테스트 세트에서 최종 성능을 평가하는 단계입니다. 각 모델의 AUC와 F1-score를 테스트 세트에서 계산하고, Confusion Matrix와 ROC Curve를 시각화하여 모델의 예측 성능을 비교합니다.
훈련 과정에서 얻은 최적화된 모델이 실제 데이터에서도 얼마나 잘 작동하는지 확인하기 위해 테스트 세트를 사용한 성능 평가가 필수적입니다. 모델이 훈련 세트와 교차 검증 세트에서 좋은 성능을 보였다고 하더라도, 테스트 세트에서 일관된 성능을 보이는지 확인해야만 모델의 일반화 성능을 신뢰할 수 있습니다. 이 과정에서는 AUC와 F1-score를 주요 성능 지표로 사용하는데 AUC는 분류기의 전체적인 성능을 평가하는 데 유용하며, F1-score는 정밀도와 재현율 간의 균형을 반영하여 특히 불균형 데이터에서의 예측 성능을 효과적으로 측정합니다.

4-1 테스트 세트 분할

```
X_test_final = X_test.drop("Churn", axis=1)  # 특성 데이터 (Churn 제외)
y_test_final = X_test["Churn"]               # 타겟 데이터 (Churn만 포함)
```

X_test_final = X_test.drop("Churn", axis=1)에서는 타겟 변수를 제외한 특성 데이터만을 X_test_final에 저장하고, y_test_final = X_test["Churn"]에서는 타겟 변수인 "Churn"을 y_test_final에 저장하여 최종 성능 평가에 사용할 데이터셋을 준비했습니다.

4-2 성능 기록을 위한 변수 초기화

```
final_test_scores = {}
final_test_f1_scores = {}
```

final_test_scores와 final_test_f1_scores는 각각 각 모델의 AUC와 F1-score를 저장하기 위한 딕셔너리입니다. 이를 통해 모든 모델의 성능 지표를 체계적으로 기록하고, 최종적으로 각 모델의 성능을 손쉽게 비교할 수 있도록 합니다.

4-3 각 모델에 대한 성능 평가 및 결과 기록

```
for name, model in optimized_models.items():
    print(f"\nEvaluating {name} on the test set:\n")
    y_pred_test = model.predict(X_test_final)
    y_proba_test = model.predict_proba(X_test_final)[:, 1] if hasattr(model, "predict_proba") else None

    # 최종 테스트 성능 측정
    print(classification_report(y_test_final, y_pred_test, target_names=["No Churn", "Churn"], zero_division=0))
    print(f"Confusion Matrix for {name}:")
    print(confusion_matrix(y_test_final, y_pred_test))
```

for 루프를 사용하여 최적화된 모든 모델의 성능 평가를 진행합니다. y_pred_test = model.predict(X_test_final)을 통해 테스트 세트에 대한 예측값을 생성하여, 각 모델의 분류 결과를 확보하였습니다. y_proba_test = model.predict_proba(X_test_final)[:, 1]을 사용하여 긍정 클래스(Churn)의 확률을 추출합니다. 이는 ROC AUC 계산에 활용되며, 모델이 각 샘플에 대해 얼마나 강하게 Churn 클래스를 예측하는지를 나타냅니다.

classification_report와 confusion_matrix를 출력하여 각 모델의 세부적인 분류 성능을 확인하였습니다. Classification Report는 정밀도, 재현율, F1-score 등 핵심 지표를 포함하고, Confusion Matrix는 올바른 분류와 잘못된 분류 비율을 시각적으로 보여줍니다. 이를 통해 각 모델의 오분류 경향을 파악하고, 성능 차이를 보다 명확히 비교할 수 있습니다.

4-4 AUC 및 F1-score 계산 및 저장

```
if y_proba_test is not None:
    auc_score = roc_auc_score(y_test_final, y_proba_test)
    final_test_scores[name] = auc_score
    print(f"{name} AUC on test set: {auc_score:.2f}")

    # ROC Curve 시각화
    fpr, tpr, _ = roc_curve(y_test_final, y_proba_test)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc_score:.2f})")
else:
    print(f"{name} does not support probability prediction (AUC and ROC skipped)")

# f1-score 계산 및 저장
f1 = f1_score(y_test_final, y_pred_test)
final_test_f1_scores[name] = f1
print(f"{name} F1 Score on test set: {f1:.2f}")
```

모델의 예측 성능을 더욱 세부적으로 평가하기 위해 AUC와 F1-score를 계산하고 저장합니다.

AUC을 계산하고 저장합니다. roc_auc_score(y_test_final, y_proba_test)를 통해 AUC를 계산하였습니다. AUC는 모델이 타겟 클래스(Churn)를 얼마나 잘 구분하는지를 나타내는 지표로, 예측의 전반적인 성능을 평가합니다. 각 모델의 AUC 값은 final_test_scores 딕셔너리에 저장합니다. 또한 ROC Curve를 plt.plot을 사용해 시각화하여 모델의 분류 성능을 직관적으로 비교할 수 있도록 하였습니다. ROC Curve는 각 모델이 임곗값에 따라 어떻게 성능이 변하는지를 보여줍니다.

F1-score을 계산하고 저장합니다. f1_score(y_test_final, y_pred_test)를 사용하여 F1-score를 계산하였습니다. F1-score는 재현율과 정밀도의 조화 평균으로, 특히 Churn 고객을 얼마나 잘 예측하는지를 판단하기 위해 사용했습니다. 각 모델의 F1-score는 final_test_f1_scores 딕셔너리에 저장됩니다.

4-5 ROC Curve 시각화

```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve on Test Set')
plt.legend()
plt.show()
```

ROC Curve는 FPR(위양성율)과 TPR(재현율)을 시각적으로 나타낸 그래프로, 각 모델의 전반적인 분류 성능을 평가할 수 있는 시각적 자료를 제공합니다. 이를 통해 모델 간 성능 차이를 한눈에 비교할 수 있습니다.

4-6 최종 성능 결과 출력

```
print("₩nFinal AUC and F1 scores on the test set for all models:")
for name in final_test_scores.keys():
    print(f"{name}: AUC = {final_test_scores[name]:.2f}, F1 = {final_test_f1_scores[name]:.2f}")
```

모든 모델에 대해 AUC와 F1-score가 저장된 후, 마지막에 print 문을 통해 각 모델의 최종 성능 지표를 출력합니다. 이를 통해 테스트 세트에서 최종 성능을 비교하고, 각 모델의 상대적 우수성을 판단할 수 있습니다.

5. 결과 및 분석

Evaluating Logistic Regression on the test set:

	precision	recall	f1-score	support
No Churn	0.86	0.40	0.55	1868
Churn	0.52	0.91	0.66	1331
accuracy			0.61	3199
macro avg	0.69	0.66	0.60	3199
weighted avg	0.72	0.61	0.60	3199

Confusion Matrix for Logistic Regression:
[[752 1116]
 [123 1208]]
Logistic Regression AUC on test set: 0.75
Logistic Regression F1 Score on test set: 0.66

Evaluating K-Neighbors on the test set:

	precision	recall	f1-score	support
No Churn	0.74	0.65	0.69	1868
Churn	0.58	0.67	0.62	1331
accuracy			0.66	3199
macro avg	0.66	0.66	0.66	3199
weighted avg	0.67	0.66	0.66	3199

Confusion Matrix for K-Neighbors:
[[1217 651]
 [435 896]]
K-Neighbors AUC on test set: 0.71
K-Neighbors F1 Score on test set: 0.62

Evaluating SVM on the test set:

	precision	recall	f1-score	support
No Churn	0.63	0.84	0.72	1868
Churn	0.59	0.32	0.41	1331
accuracy			0.62	3199
macro avg	0.61	0.58	0.57	3199
weighted avg	0.61	0.62	0.59	3199

Confusion Matrix for SVM:
[[1572 296]
 [911 420]]
SVM AUC on test set: 0.66
SVM F1 Score on test set: 0.41

Evaluating Random Forest on the test set:

	precision	recall	f1-score	support
No Churn	0.72	0.75	0.73	1868
Churn	0.63	0.59	0.61	1331
accuracy			0.68	3199
macro avg	0.67	0.67	0.67	3199
weighted avg	0.68	0.68	0.68	3199

Confusion Matrix for Random Forest:
[[1399 469]
 [540 791]]
Random Forest AUC on test set: 0.74
Random Forest F1 Score on test set: 0.61

Evaluating Gradient Boosting on the test set:

	precision	recall	f1-score	support
No Churn	0.71	0.77	0.74	1868
Churn	0.63	0.57	0.60	1331
accuracy			0.68	3199
macro avg	0.67	0.67	0.67	3199
weighted avg	0.68	0.68	0.68	3199

Confusion Matrix for Gradient Boosting:
[[1434 434]
 [576 755]]
Gradient Boosting AUC on test set: 0.74
Gradient Boosting F1 Score on test set: 0.60

Evaluating XGBoost on the test set:

	precision	recall	f1-score	support
No Churn	0.67	0.84	0.75	1868
Churn	0.66	0.43	0.52	1331
accuracy			0.67	3199
macro avg	0.67	0.64	0.64	3199
weighted avg	0.67	0.67	0.65	3199

Confusion Matrix for XGBoost:
[[1578 290]
 [760 571]]
XGBoost AUC on test set: 0.72
XGBoost F1 Score on test set: 0.52

Evaluating LightGBM on the test set:

	precision	recall	f1-score	support
No Churn	0.67	0.86	0.75	1868
Churn	0.67	0.40	0.50	1331
accuracy			0.67	3199
macro avg	0.67	0.63	0.62	3199
weighted avg	0.67	0.67	0.65	3199

Confusion Matrix for LightGBM:
[[1602 266]
 [800 531]]
LightGBM AUC on test set: 0.74
LightGBM F1 Score on test set: 0.50

Evaluating Voting Ensemble (Soft) on the test set:

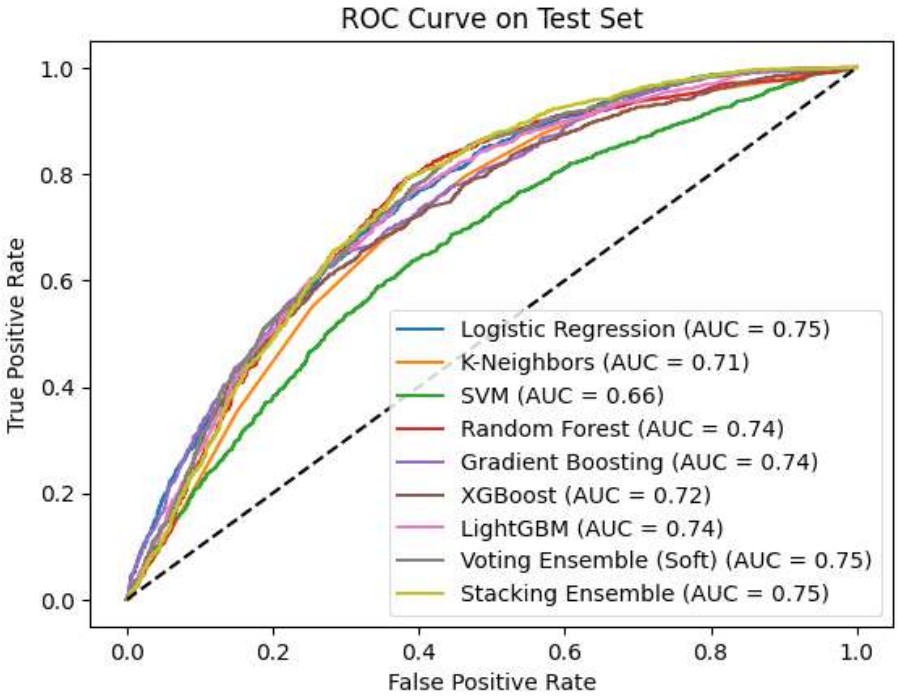
	precision	recall	f1-score	support
No Churn	0.71	0.79	0.75	1868
Churn	0.65	0.53	0.59	1331
accuracy			0.69	3199
macro avg	0.68	0.66	0.67	3199
weighted avg	0.68	0.69	0.68	3199

Confusion Matrix for Voting Ensemble (Soft):
[[1483 385]
 [620 711]]
Voting Ensemble (Soft) AUC on test set: 0.75
Voting Ensemble (Soft) F1 Score on test set: 0.59

Evaluating Stacking Ensemble on the test set:

	precision	recall	f1-score	support
No Churn	0.59	0.99	0.74	1868
Churn	0.65	0.02	0.03	1331
accuracy			0.59	3199
macro avg	0.62	0.50	0.38	3199
weighted avg	0.61	0.59	0.44	3199

Confusion Matrix for Stacking Ensemble:
[[1857   11]
 [1311   20]]
Stacking Ensemble AUC on test set: 0.75
Stacking Ensemble F1 Score on test set: 0.03




```
Final AUC and F1 scores on the test set for all models:
Logistic Regression: AUC = 0.75, F1 = 0.66
K-Neighbors: AUC = 0.71, F1 = 0.62
SVM: AUC = 0.66, F1 = 0.41
Random Forest: AUC = 0.74, F1 = 0.61
Gradient Boosting: AUC = 0.74, F1 = 0.60
XGBoost: AUC = 0.72, F1 = 0.52
LightGBM: AUC = 0.74, F1 = 0.50
Voting Ensemble (Soft): AUC = 0.75, F1 = 0.59
Stacking Ensemble: AUC = 0.75, F1 = 0.03
```

이번 프로젝트에서는 최적화된 모델을 다양한 성능 지표로 평가하여 최종 성능을 비교했습니다. 주요 성능 지표로는 AUC (Area Under the Curve)와 F1-score가 사용되었으며, 각각의 지표는 모델이 Churn(고객 이탈)과 No Churn을 얼마나 잘 구분하는지를 평가하는 중요한 척도입니다. 특히 AUC는 모델의 전반적인 분류 성능을 나타내며, F1-score는 재현율과 정밀도를 모두 고려하여 이탈 고객 예측의 신뢰성을 높이는 데 유리한 지표입니다.

각각이 결과를 살펴보겠습니다.

Logistic Regression 모델의 경우 AUC: 0.75, F1-score: 0.66의 값을 기록하였습니다. Logistic Regression은 테스트 세트에서 가장 높은 AUC와 F1-score를 기록했으며, 전반적으로 뛰어난 성능을 보였습니다. AUC 0.75는 모델이 Churn과 No Churn을 잘 구분하고 있음을 나타내며, F1-score 0.66은 재현율과 정밀도 간 균형이 잘 잡혀 있어 고객 이탈 예측 문제에 적합함을 보여줍니다. Confusion Matrix를 살펴보면 No Churn에서 752건, Churn에서 123건의 오분류가 발생했습니다. 이는 일부 No Churn 고객이 오탐될 가능성이 있지만, 이탈 가능성이 높은 고객은 잘 예측하고 있음을 나타낸다고 생각합니다. ROC Curve를 살펴보면 Logistic Regression의 ROC Curve는 양호한 상승 곡선을 보이며, 모델의 안정적인 성능을 시각적으로 확인할 수 있습니다.

Random Forest 모델의 경우 AUC: 0.74, F1-score: 0.61 값을 기록하였습니다. Random

Forest는 AUC와 F1-score 모두 우수한 성능을 보였으며, 특히 다양한 트리 기반의 앙상블 방식을 통해 예측 안정성을 높였습니다. 복잡한 데이터 패턴을 잘 학습하여 다양한 변수를 효과적으로 반영한 것으로 파악할 수 있습니다. Confusion Matrix를 살펴보면 No Churn 고객 1399명, Churn 고객 540명이 오분류 되었습니다. 특히 Churn 고객 예측에서 안정성을 보이는 것이 Random Forest의 강점으로 작용했다고 생각합니다. ROC Curve를 살펴보면 Logistic Regression보다 약간 낮은 곡선을 보였지만, 전반적으로 Churn 예측에서 안정적인 성능을 발휘하는 모델이라는 것을 알 수 있습니다.

부스팅 모델 (Gradient Boosting, XGBoost, LightGBM)들의 경우 AUC 측면에서 높은 성능을 보였으나, F1-score는 Logistic Regression과 Random Forest에 비해 다소 낮았습니다. 이는 부스팅 모델들이 강력한 분류 성능을 가지면서도 데이터 불균형이나 복잡한 변수 간의 상호작용에서 정밀성과 재현율 측면에서 아쉬움을 나타낸다고 생각합니다. Gradient Boosting의 경우 AUC 0.74, F1-score 0.60으로 양호한 성능을 보였으며, 안정적인 성능을 나타냅니다. XGBoost의 경우 AUC 0.72, F1-score 0.52로 높은 AUC를 유지하지만, F1-score에서는 아쉬운 결과를 보였습니다. 이는 오탐을 줄이기 위해 정밀도를 높이는 데 중점을 두었기 때문입니다. LightGBM의 경우 AUC 0.74, F1-score 0.50으로 안정적이지만, 최종 예측의 정밀도와 재현율 측면에서는 Logistic Regression이나 Random Forest보다 낮습니다. 각 부스팅 모델들의 Confusion Matrix와 ROC Curve를 살펴보면 부스팅 모델들은 전체적으로 Churn 예측에 강점을 보이나, 일부 고객군에 대해 오분류가 발생하는 경향이 있음을 알 수 있습니다. ROC Curve는 Logistic Regression과 Random Forest에 비해 약간 낮은 곡선을 보이지만, 전반적으로 강력한 분류 성능을 보인다고 생각합니다.

K-Neighbors의 경우 AUC 0.71, F1-score 0.62로, 부스팅 모델보다 F1-score가 높으며 보통 수준의 성능을 유지했습니다. 거리 기반으로 분류를 수행하는 특성상 데이터의 밀도와 분포에 영향을 받을 수 있지만, 상대적으로 단순한 알고리즘임에도 불구하고 복잡한 Telco 데이터에서도 안정적인 예측 성능을 보였습니다.

SVM의 경우 AUC 0.66, F1-score 0.41로, 이번 프로젝트에서 가장 낮은 성능을 기록했습니다. SVM은 고차원 특성 공간에서 효과적이나, 다중 변수와 노이즈가 많은 데이터에서는 성능이 떨어질 수 있음을 알 수 있었습니다.

Voting Ensemble 모델은 AUC 0.75, F1-score 0.59로 안정적인 성능을 보였으나, F1-score에서는 Logistic Regression보다 0.07 낮았습니다. 다양한 모델의 예측을 종합하여 예측의 안정성을 높였지만, 단일 모델인 Logistic Regression의 성능을 크게 초과하지는 못했습니다.

Stacking Ensemble 모델의 경우 AUC 0.75로 높은 성능을 기록했으나, F1-score가 0.03으로 매우 낮았습니다. 이는 과적합을 매우 높은 확률로 가리키며 모델 최적화가 필요하다고 생각합니다.

따라서 Logistic Regression과 Random Forest가 이번 프로젝트에서 최적의 모델로 선정되었습니다. Logistic Regression은 AUC와 F1-score 모두에서 우수한 성능을 보이며, Churn 예측 전반에서 안정적인 분류 성능을 보여주어 정밀도와 재현율 간의 균형이 중요한 상황에 적합한 선택으로 평가됩니다. Random Forest는 복잡한 데이터 패턴을 잘 학습하여 다양한 변수 상호작용을 효과적으로 반영하면서도 일관된 예측을 제공합니다. 부스팅 모델들은 AUC 지표에서 우수한 성능을 보였지만, F1-score 측면에서는 Logistic Regression과 Random Forest에 비해 다소 낮은 결과를 나타내었습니다. 이는 부스팅 모델이 AUC 중심의 예측 성

능이 필요한 상황에서는 유리할 수 있지만, 불균형한 데이터에서 Churn 고객을 예측하는 데 있어 정밀성과 재현율이 중요한 환경에서는 한계를 가질 수 있음을 시사합니다. ROC Curve 시각화를 통해서도 Logistic Regression과 Random Forest가 다른 모델에 비해 고른 성능을 나타내며, 전반적으로 높은 예측 성능을 유지하고 있음을 확인할 수 있었습니다. 종합적으로, Churn 예측의 중요도가 높고 정밀한 분류가 필요한 경우에는 Logistic Regression이 적합하며, 복잡한 데이터 패턴이나 다양한 변수 상호작용을 고려해야 하는 경우에는 Random Forest와 부스팅 모델이 유리할 수 있다고 생각합니다.

6. 고찰

Telco Customer Churn Dat를 통해 고객 이탈 예측 모델을 구축하는 과정에서 데이터 전처리, 모델 선정, 하이퍼파라미터 튜닝, 그리고 성능 평가까지 전 과정을 경험했습니다. 특히, 고객 이탈 예측이라는 불균형 데이터 문제를 해결하면서 각 단계에서 데이터 특성과 목표에 맞는 최적의 모델을 선정하기 위해 여러 고민고 다양한 시도를 하였습니다.

먼저 데이터셋을 살펴보았을 때 다양한 특성들이 포함되어 있었고, 특히 사용 기간, 월 요금, 총 사용 요금과 같은 특성들이 고객 이탈에 중요한 변수일 것이라 판단했습니다. 이러한 판단은 초기의 데이터 탐색 분석(EDA) 과정에서 상관계수 행렬을 통해 이루어졌습니다. 예를 들어, 사용 기간이 길수록 이탈 가능성이 줄어들 것이라는 가설과 높은 월 요금이 고객의 이탈에 영향을 미칠 것이라는 가설을 세웠고, 이를 바탕으로 여러 변수 간의 상관관계를 분석했습니다. 그러나 특성 간의 상관관계가 매우 높지 않다는 것을 확인하며 단일 변수의 영향보다는 다중 변수의 조합이나 상호작용이 고객 이탈 예측에 더욱 중요할 수 있음을 인식하게 되었습니다. 따라서 이후 모델 선택과 테스트 과정에서 단일 변수보다는 다변수 간 상호작용을 강화할 수 있는 모델인 앙상블 모델(Random Forest)과 부스팅 모델(Gradient Boosting, XGBoost)을 주의 깊게 봐야겠다고 생각했습니다.

데이터 전처리 과정에서는 결측치 처리, 변수 변환, 원-핫 인코딩을 통한 다중 클래스 변수 변환 등 여러 문제를 해결했습니다. 예를 들어, TotalCharges 변수에 결측치가 존재했으며, 이를 어떻게 처리할지가 모델의 성능에 큰 영향을 미칠 중요한 결정이라고 생각했습니다. 따라서 결측치를 처리하기 위해, 결측치를 평균이나 중앙값으로 대체하는 방법과 결측치가 있는 행을 제거하는 방법 두 가지를 검토했습니다. 두 방식을 각각 적용한 후 교차 검증을 통해 성능 변화를 측정해 보았습니다. 그 결과, 결측치를 제거할 경우 데이터 손실이 발생하면서 성능이 크게 향상되지는 않았습니다. 반면, 중앙값으로 결측치를 대체하면 데이터 손실을 최소화하면서도 성능을 유지할 수 있었습니다. 중앙값 대체를 선택한 이유는, 중앙값이 평균보다 이상치에 덜 민감하여 TotalCharges의 극단적인 값을 완화하고 데이터의 안정성을 높이는 데 효과적이었기 때문입니다.

이외에도 여러 범주형 변수들은 모델이 각 범주를 명확히 인식할 수 있도록 원-핫 인코딩을 통해 수치형 데이터로 변환했습니다. 예를 들어, 고객이 가입한 계약 유형(월별, 연간, 2년 계약)과 인터넷 서비스 종류(DSL, 광케이블 등)와 같은 변수들은 원-핫 인코딩을 적용하여 각 범주가 독립된 특성으로 모델에 입력되도록 했습니다. 이러한 변환은 모델이 각 범주의 특성을 구별하고 예측에 반영할 수 있도록 돕는 역할을 했습니다. 이진형 변수인 성별과 같이 두 가지 값만을 갖는 변수는 0과 1로 매핑하여 모델이 더 간단히 처리할 수 있도록 했습니다.

모델 선정과 하이퍼파라미터 튜닝 과정에서는 각 모델이 고객 이탈 예측에서 어느 정도 성능을 보일 수 있는지를 확인하며 단계별로 최적화했습니다. Logistic Regression, K-Neighbors, SVM, Random Forest, Gradient Boosting, XGBoost, LightGBM 등 다양한 모델을 비교하여 이 예측 문제에 가장 적합한 모델을 선정하고자 했습니다.

먼저 Logistic Regression은 해석이 용이하고 예측 경계를 명확히 할 수 있다는 장점이 있어 초기 선택 모델로 도입했습니다. Logistic Regression에서 다중공선성을 줄이고 모델을 단순화하기 위해 상관관계가 낮은 특성들을 제거하거나 정규화를 통해 데이터를 스케일링하였습니다. 기본적으로 안정적인 성능을 보였으나, 데이터의 고차원적 관계를 반영하는 데는 한계가 있음을 확인하고 다른 모델들과의 성능 비교가 필요하다고 판단했습니다.

K-Neighbors 모델은 거리 기반 모델로 데이터의 밀도와 분포에 민감하게 반응하기 때문에 복잡한 데이터에서 어떻게 반응할지를 확인하기 위해 적용해 보았습니다. 하지만 복잡한 패턴을 파악하기에는 한계가 있어, 다양한 이웃 수를 테스트하면서 최적의 하이퍼파라미터를 찾는 과정에서도 상대적으로 낮은 성능을 보였습니다. 이를 통해 단순한 거리 기반 방식이 고차원적 데이터 패턴을 잡아내기 어렵다는 것을 배웠습니다.

SVM 모델은 고차원 데이터에서의 장점이 있기 때문에 도입했으나, 노이즈가 많은 이번 데이터 셋에서는 기대한 성능을 발휘하지 못했습니다. 커널 선택과 정규화 매개변수를 조정하며 최적화를 시도했으나, 고객 이탈 예측에서는 상대적으로 저조한 성과를 보여 최종 모델 후보에서 제외하게 되었습니다.

이후 앙상블 모델 중 하나인 Random Forest와 다양한 부스팅 모델들을 본격적으로 시도해 보았습니다. Random Forest는 여러 결정 트리를 앙상블 방식으로 결합하여 예측의 안정성을 높였고, 트리의 개수와 최대 깊이, 최소 샘플 수와 같은 파라미터를 조정하며 최적의 모델을 만들고자 했습니다. 특히, 과적합 방지를 위해 최소 샘플 분할 수를 조정하고 트리 개수를 늘려가며 안정적인 예측 성능을 확보할 수 있었습니다. 이와 같은 조정은 Random Forest가 복잡한 데이터 패턴을 잘 학습하고 여러 특성 간 상호작용을 효과적으로 반영할 수 있도록 했습니다.

부스팅 모델의 경우, Gradient Boosting, XGBoost, LightGBM을 각각 사용해 보며 학습률과 추정기 개수를 최적화하는 데 중점을 두었습니다. Gradient Boosting에서는 학습률을 낮춰 점진적인 성능 향상을 노렸고, XGBoost와 LightGBM에서는 학습률 외에도 최대 깊이, 잎의 개수 등의 매개변수를 조정하며 최적의 성능을 찾고자 했습니다. 부스팅 모델들은 AUC에서 강력한 성능을 보였으나, 불균형 데이터 문제에서 재현율과 정밀도의 균형을 유지하는 데는 다소 어려움이 있었습니다. 특히 과적합을 방지하기 위해 조기 종료를 사용하여 적절한 학습률을 찾았고 이를 통해 F1-score를 개선할 수 있었습니다.

앙상블 기법으로는 Voting Ensemble과 Stacking Ensemble을 사용해 보았습니다. Voting Ensemble은 각 모델의 예측을 종합하여 다수결 방식으로 최종 결과를 도출하여 안정성을 제공했으나, 단일 모델인 Logistic Regression의 성능을 크게 초과하지는 못했습니다. Stacking Ensemble은 여러 모델의 예측 결과를 Logistic Regression 메타 모델에 결합하여 최적의 예측 성능을 기대했으나, 과적합이 발생하여 오히려 성능이 저하되는 결과를 보였습니다. 이를 통해 Stacking Ensemble은 복잡한 데이터에 과적합될 수 있음을 파악했고, 데이터 복잡성에 따라 모델 간의 상호작용이 예측 성능에 미치는 영향을 고려해야 함을 배울 수 있었습니다.

각 단계마다 모델 평가 관점이 달라지며 결론이 다소 상이하게 도출되었고, 이는 각각의 평

가 기준이 모델 선택에 미치는 영향을 보여주었습니다. 이를 통해 통계적 유의성 검토 단계, 시각화 단계, 그리고 최종 성능 평가 단계에서 각각 어떤 결과가 나왔는지, 그 차이와 이유를 구체적으로 설명하고자 합니다.

먼저 통계적 유의성 검토 단계에서는 각 모델의 성능 차이를 검토하기 위해 p-value를 기준으로 모델 간 유의미한 성능 차이가 있는지 분석했습니다. 이때는 각 모델의 평균 성능을 단순 비교하는 것이 아닌, 성능 차이의 유의성을 판단하는 데 초점을 두었기 때문에, Random Forest보다 Stacking Ensemble이 유의미하게 높은 성능을 보였습니다. 이는 Stacking Ensemble이 다수의 모델 예측을 결합하여 놓칠 수 있는 패턴을 효과적으로 포착하면서 성능을 극대화할 수 있었기 때문입니다. 따라서 통계적 유의성 기준에서는 Stacking Ensemble이 가장 우수한 모델로 해석되었습니다. 그러나 이 단계에서는 안정성과 일관성보다는 성능의 평균적인 차이에만 집중했기 때문에 다른 평가 단계와는 다른 결론이 도출될 수 있었습니다.

시각화 단계에서는 AUC와 F1-score의 분포를 Boxplot으로 시각화하여 모델별 성능의 안정성과 일관성을 비교했습니다. 각 모델의 성능이 데이터에 대해 얼마나 일관성 있게 나타나는지를 중점적으로 평가하면서, Random Forest와 Stacking Ensemble 모두 안정적인 성능을 보여주는 모델로 평가되었습니다. 특히, Random Forest는 다양한 변수 상호작용을 효과적으로 학습하고 데이터 분포에 대한 적응력을 높이며 안정성을 확보하였고, Stacking Ensemble은 예측의 종합적 안정성을 보여주었습니다. 통계적 유의성 검토 단계와는 달리, 시각화 단계에서는 성능 차이보다는 각 모델의 예측이 얼마나 일관적이고 안정적인지를 강조하였기에 Random Forest와 Stacking Ensemble 두 모델이 모두 우수하다고 판단되었습니다..

최종 모델 선정 단계에서는 각 모델의 전반적인 성능뿐 아니라, 재현율과 정밀도 간의 균형을 종합적으로 고려했습니다. 고객 이탈 예측 문제에서 재현율과 정밀도의 균형이 중요하기 때문에, F1-score와 AUC 모두를 함께 평가 지표로 삼아 모델을 평가하게 되었습니다. 특히 Stacking Ensemble은 AUC에서 높은 성능을 보였지만, F1-score가 낮아 과적합 가능성이 있었고, 이는 다수의 모델 예측을 결합하는 과정에서 과적합이 발생할 가능성을 시사했습니다. 반면, Logistic Regression과 Random Forest는 AUC와 F1-score 모두에서 안정적이고 우수한 성능을 보여, 고객 이탈 예측 문제에 신뢰할 수 있는 모델로 판단되었습니다. Logistic Regression은 해석이 용이하고 정밀도와 재현율의 균형을 잘 잡아주는 장점이 있었고, Random Forest는 다양한 변수 간 상호작용을 학습하면서도 복잡한 데이터 패턴을 효과적으로 반영하여 안정적인 성능을 제공했습니다. 따라서 최종적으로는 Logistic Regression과 Random Forest가 가장 적합한 모델로 선정하였습니다.

그러나 이 분류 프로젝트에서 제가 의도하고 했던것은 실제로 이탈한 고객 중 모델이 올바르게 이탈을 예측한 고객의 비율을 더 중점적으로 보아 실제로 이탈할 가능성이 높은 고객을 놓치지 않는 것입니다. 저는 고객 이탈 예측에서는 이탈 가능성이 높은 고객을 최대한 정확하게 찾아내어 적절한 대응을 할 수 있어야 한다고 생각합니다 따라서 최종적으로 두 모델 중 F1-score가 더 높은 Logistic Regression이 특히 적합하다고 판단했습니다.

각 단계에서 가장 성능이 좋은 모델이 계속 달라진 이유는 각 단계마다 사용한 평가 기준이 서로 달랐기 때문입니다. 통계적 유의성 검토 단계에서는 성능 차이의 유의성에 중점을 두어 Stacking Ensemble이 우수한 모델로 판단되었고, 시각화 단계에서는 성능의 일관성과 안정성을 평가하여 Random Forest와 Stacking Ensemble을 우수한 모델로 해석했습니다. 마지막으로, 최종 모델 선정 단계에서는 전반적인 성능과 함께 재현율과 정밀도의 균형을 고려했으며, 프로젝트의 최종 목표인 일반화 성능 확보를 위해 테스트 세트에서 안정적인 성능을 내

는 모델이 최적이라는 판단에 기반해 Logistic Regression과 Random Forest를 최적의 모델로 선정했습니다. 그러나 고객 이탈 예측 문제에서 중요한 점은 이탈 가능성이 높은 고객을 놓치지 않고 예측해내는 것이라고 저는 생각하였기 때문에 이를 위해 재현율이 높은 모델을 중시했습니다. 따라서 프로젝트의 최종 모델은 두 모델 중 F1-score가 더 높은 Logistic Regression이라고 생각합니다. 이 과정을 통해 각 단계에서 평가 기준과 목적이 달라지면 결론도 달라질 수 있음을 확인할 수 있었고, 각 단계에서 무엇을 중점적으로 평가할지 고민하고 분석하는 과정의 중요성을 다시금 깨달았습니다.

또한 이번 프로젝트에서 고객 이탈 예측 모델을 구축하는 과정에서 다양한 모델을 활용하여 최적화를 시도했지만, 몇 가지 한계점과 개선이 필요한 부분을 확인할 수 있었습니다.

첫째, 데이터 불균형 문제의 지속적인 한계가 있었습니다. SMOTE와 같은 오버샘플링 기법을 통해 소수 클래스에 대한 데이터 균형을 맞추려 했으나, 불균형한 데이터에서 오는 제약을 완전히 극복하지는 못했습니다. 특히, F1-score와 AUC에서 모델별로 일관성이 떨어지는 결과가 있었는데, 이는 불균형 데이터로 인해 모델이 다수 클래스에 편향되는 경향을 나타냈기 때문으로 보입니다. 따라서 향후에는 SMOTE 외에 ADASYN이나 Undersampling과 같은 다른 샘플링 기법을 시도하거나, Focal Loss와 같은 불균형 데이터에 특화된 손실 함수를 적용해 볼 필요가 있습니다. 이를 통해 소수 클래스에 대한 민감도를 높이고, 고객 이탈 예측의 정확도를 더욱 향상할 수 있을 것입니다.

둘째, 특성 선택 및 추가 특성 생성에 한계가 있었다고 생각합니다. 이번 분석에서는 고객의 사용 기간, 요금과 관련된 몇 가지 주요 특성을 바탕으로 예측을 시도했으나, 고객 이탈을 더 잘 설명할 수 있는 잠재적 특성이 추가로 필요할 수 있습니다. 예를 들어, 고객이 서비스를 이용하는 동안 발생한 기술 지원 요청 횟수, 요금 변경 기록, 또는 프로모션 적용 여부 등 고객 경험과 직접적으로 연관된 변수를 추가하는 것이 도움이 될 수 있다고 생각합니다. 이러한 특성은 고객의 불만족도를 간접적으로 측정할 수 있는 요소로 작용할 수 있으며 이탈 가능성을 더 정확히 반영할 수 있었을 것입니다. 따라서 추가적인 데이터 수집 및 특성 생성 전략을 마련하여 고객의 이탈 행동을 다각도로 파악할 필요가 있다고 생각합니다.

셋째, 모델 간 성능 차이의 원인 분석이 부족하다고 생각합니다. Logistic Regression과 Random Forest 모델이 우수한 성능을 보였으나, 일부 부스팅 모델과의 성능 차이를 완전히 설명하지 못한 부분이 있습니다. 부스팅 모델들이 상대적으로 F1-score에서 낮은 결과를 보인 것은 과적합 문제일 가능성이 크지만, 이를 명확히 하기 위해 각 모델이 특정 특성이나 특성 조합에 대해 어떻게 반응하는지에 대한 더 세부적인 분석이 필요할 것입니다. SHAP(SHapley Additive exPlanations) 값이나 LIME(Local Interpretable Model-agnostic Explanations) 등의 설명 가능한 인공지능 기법을 사용해 모델이 개별 특성에 어떻게 반응하는지 시각화하고, 모델 선택에 대한 해석을 추가로 강화할 수 있습니다.

넷째, 성능 평가 지표의 추가적인 다양화가 필요하다고 생각합니다. 이이번 프로젝트에서는 AUC와 F1-score를 중심으로 성능을 평가하였으며, 최종적으로 재현율을 더욱 중시하여 F1-score가 높은 모델을 최종 선정했습니다. 그러나 고객 이탈 예측에서 중요한 비즈니스적 의사 결정에는 재현율과 정밀도 사이의 균형뿐만 아니라 비용 민감도가 반영될 필요가 있습니다. 예를 들어, 이탈할 가능성이 높은 고객을 놓치는 경우의 비용(고객을 잃을 때의 손실)과 이탈하지 않을 고객을 이탈할 것으로 잘못 분류하는 경우의 비용을 비교해 보고, 비용-민감도 분석(Cost-Sensitive Analysis)을 추가할 수 있습니다. 이를 통해 실제 비즈니스 상황에서 더 실질적인 성능 지표를 평가할 수 있을 것입니다.

다섯째, 데이터 전처리 과정에서의 개선 가능성이 있다고 생각합니다. TotalCharges 변수의 결측치를 중앙값으로 대체하였으나, 결측치가 있는 고객군이 특정 패턴을 가지고 있을 가능성을 간과했을 수 있습니다. 결측치 자체가 고객의 특정 행동 특성을 반영할 수 있으므로, 단순히 중앙값으로 대체하기보다는 결측치를 가진 고객군을 별도로 분석하거나 결측 여부를 새로운 이진 특성으로 추가하여 모델에 반영하는 방안을 생각해 볼 수 있습니다. 예를 들어, 결측 여부를 하나의 이진 특성으로 포함시키면, 고객군의 특성을 보다 정교하게 반영하여 모델이 더 나은 예측을 할 수 있을 것입니다.

마지막으로, 시계열 데이터로의 확장 가능성이 있다고 생각합니다. 현재는 정적인 특성을 사용하여 예측을 진행했지만, 고객의 행동이 시간에 따라 변화하는 경우 이를 반영하지 못하는 한계가 있었습니다. 고객 이탈 예측에서는 요금 인상 또는 서비스 불만 등과 같은 특정 이벤트가 고객의 이탈 의사 결정에 영향을 미칠 수 있으므로, 향후 데이터 수집 및 분석 과정에서 시계열 데이터를 도입하는 것도 고려할 수 있습니다. 고객의 월별 요금 변동 추이나 서비스 이용 빈도 변화와 같은 패턴을 시계열 데이터로 구축하고 이를 예측 모델에 반영하면, 고객 이탈의 동적 요소를 더 잘 반영할 수 있습니다. 이를 통해 정적 데이터에서 확인하기 어려운 변화 요인들을 효과적으로 포착하여 예측 성능을 높일 수 있을 것입니다.

종합적으로, 이번 프로젝트에서 얻은 경험과 인사이트를 바탕으로 이러한 한계점을 개선하고, 고객 이탈 예측의 정확도와 실용성을 더욱 높이는 방향으로 연구를 발전시킬 수 있을 것입니다. 이번 프로젝트를 통해 고객 이탈 예측 모델 구축에 필요한 다양한 단계와 도전 과제를 깊이 경험할 수 있었으며, 각 단계에서의 평가 기준과 접근 방식의 선택이 최종 결과에 미치는 영향을 체감했습니다. 이 경험 바탕으로 보다 실용적이고 신뢰할 수 있는 예측 모델을 개발하며, 고객 이탈 예측 문제에 대한 더욱 정교한 해결책을 제시하는 데 기여하고자 합니다.

회귀

0. 서론

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso, Ridge, LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, BaggingRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV, train_test_split
import numpy as np
import re
import matplotlib.pyplot as plt
import seaborn as sns

#코랩에서 데이터 시각화를 더 잘하기 위함
%matplotlib inline

# 데이터 불러오기
vehicle_data = pd.read_csv('/content/sample_data/used_cars.csv')
```

Used Car Price Prediction Data 데이터는 중고차 가격 예측이라는 전형적인 지도 학습 문제를 다루고 있습니다. 데이터셋에는 레이블된 훈련 샘플, 즉 각 중고차의 실제 가격이 포함되어 있어 지도 학습을 위한 회귀 모델을 학습하기에 적합합니다. 특히, 모델이 연속적인 값을 예측해야 하므로 이 문제는 회귀 문제로 분류됩니다. 예측에 사용할 주요 특성은 milage(주행 거리), engine_output(엔진 성능), fuel_type(연료 타입) 등 여러 가지로 구성되어 있으며, 이러한 여러 특성들이 하나의 타깃 값(중고차 가격)을 예측하도록 학습시키기 때문에 다중 회귀 문제로 정의할 수 있습니다. 중고차의 가격 예측이 구체적으로 여러 값을 예측하는 것이 아닌 각 차량의 단일 가격을 예측하므로, 다변량 회귀가 아닌 단변량 회귀 문제로 볼 수 있습니다.

또한, 이 시스템에 들어오는 데이터는 시간이 지나면서 변화하지 않으며, 데이터의 흐름에 따라 모델이 빠르게 적응할 필요가 없습니다. 데이터셋 자체가 메모리에 충분히 적재될 만큼 크지 않기 때문에 배치 학습 방식이 적절하다고 판단했습니다. 배치 학습을 통해 모델을 한 번에 학습시킴으로써, 데이터의 전체적인 패턴을 안정적으로 반영할 수 있습니다. 이러한 점을 인지하고 분석을 시작했습니다.

1. 데이터 선정 이유

이번 회귀 과제에서는 중고차 가격 예측 데이터를 선택했습니다. 이 데이터는 기계 학습 기법을 학습하며 실제 응용 가능성이 높은 예측 모델을 구현할 수 있는 좋은 사례라고 생각하였고 다음과 같은 이유로 선정하였습니다.

첫째, 회귀 문제의 전 과정을 경험할 수 있는 사례로서 적합하기 때문입니다. 중고차 가격 예측은 회귀 문제로서 데이터 전처리, 특성 선택, 모델 학습, 평가 지표 적용 등 회귀 모델링의 전 과정을 포함하고 있습니다. 다양한 특성들이 가격에 미치는 영향을 파악하고 실질적인 예측 모델을 구축하는 과정을 통해 특성 선택과 상호작용 생성이 예측 성능에 미치는 영향을

심도 있게 학습할 수 있었습니다. 특히 차량의 연식, 주행 거리, 브랜드 등 실생활에서 의미 있는 특성들이 가격에 어떻게 영향을 미치는지를 학습함으로써 실질적 응용 가능성을 높일 수 있었습니다.

둘째, 자율주행 프로젝트와의 연계 가능성을 고려했습니다. 자율주행 시스템은 다양한 센서와 환경 데이터를 실시간으로 분석하여 정확한 의사결정을 내려야 합니다. 비록 중고차 가격 예측과 자율주행의 주제는 다르지만, 모두 높은 예측 정확도가 요구되며 기계학습 모델링 기법을 통한 예측과 평가를 필요로 합니다. 이번 과제를 통해 익힌 회귀 모델링의 기초 역량은 자율주행 프로젝트에서도 활용될 수 있을 것입니다. 예를 들어, 이번 과제에서 경험한 특성 선택과 평가 지표 적용 방법은 자율주행 프로젝트에서의 정확한 예측을 위해 필수적인 기술로 활용될 수 있다고 생각합니다.

셋째, 컴퓨터 비전 수업에서 진행 중인 채소 이미지 분류 프로젝트와의 연관성 또한 고려하였습니다. 비록 회귀와 분류의 모델 유형은 다르지만, 이번 과제에서 경험한 특성 생성 및 모델 성능 평가 방법은 채소 이미지 분류 프로젝트에서도 직접적으로 활용될 수 있습니다. 예를 들어, 이번 과제에서 학습한 특성 상호작용의 개념은 이미지의 텍스처 그리고 색상 간의 상호작용을 통해 분류 성능을 향상하는 데 기여할 수 있을 것으로 기대됩니다.

마지막으로, 실무와의 연관성을 유지하며 기초 역량을 쌓기 위한 목표가 있습니다. 본 과제는 기계학습 회귀 모델의 기본 역량을 학습할 수 있는 좋은 기회로, 자율주행 프로젝트와 같은 복잡한 데이터 분석 작업에서 예측 모델을 구축하고 최적화하는 데 필요한 중요한 기초를 다지기 위해 선택했습니다. 이를 통해 모델 성능의 강점과 약점을 파악하고 개선하는 능력을 길러 실무 환경에서의 모델 활용 역량을 높일 수 있기를 목표 하였습니다.

2 데이터 전처리

2-1 불필요한 Column과 결측치가 있는 행 제거

```
vehicle_data_cleaned = vehicle_data.dropna(subset=['accident', 'clean_title'])
vehicle_data_cleaned = vehicle_data_cleaned.dropna(subset=['fuel_type'])
vehicle_data_cleaned = vehicle_data_cleaned.drop(columns=['model', 'clean_title'])
```

먼저, 중고차 가격 예측에 직접적인 영향을 미치지 않는 'model'과 'clean_title' 컬럼을 제거하여 데이터의 불필요한 요소를 정리하였습니다. 'model' 컬럼은 고유값이 많아 예측 모델에 큰 기여를 하지 않는 특성이며, 'clean_title'은 이미 사고 여부를 나타내는 'accident' 컬럼과 중복되는 정보를 제공할 수 있어 제거하였습니다. 또한, 데이터의 신뢰성을 확보하기 위해 'accident', 'clean_title', 'fuel_type' 컬럼에서 결측치가 있는 행을 삭제하였습니다. 결측치가 포함된 데이터를 그대로 학습에 사용하면 모델이 불완전하거나 잘못된 정보에 의해 왜곡될 수 있기 때문에 결측치가 있는 행을 제거함으로써 데이터의 일관성과 신뢰성을 높였습니다.

2-2 수치형 변환을 통한 형식 정제

```
vehicle_data_cleaned['milage'] = vehicle_data_cleaned['milage'].replace({' mi.': '', ',': ''}, regex=True).astype(int)
vehicle_data_cleaned['price'] = vehicle_data_cleaned['price'].replace({'₩$': '', ',': ''}, regex=True).astype(int)
```

주행 거리(milage)와 가격(price) 컬럼에는 'mi.', '$' 등의 단위 표시와 쉼표가 포함되어 있어, 수치형 데이터로 사용할 수 없는 상태였습니다. 이를 해결하기 위해 이러한 불필요한 문자를 제거하고 정수형(int)으로 변환하여 데이터를 일관된 수치형 형식으로 정제하였습니다. 이 과정은 데이터의 정확성을 높이고 모델 학습과 숫자 계산, 통계 분석에 필요한 안정적이고 일관된 수치형 데이터를 제공하는데 중요한 과정이라고 생각합니다.

2-3 범주형 변수의 dummy 인코딩

```
vehicle_data_encoded = pd.get_dummies(vehicle_data_cleaned, columns=['fuel_type', 'transmission', 'accident', 'brand', 'ext_col', 'int_col'];
```

범주형 변수인 'fuel_type', 'transmission', 'accident', 'brand', 'ext_col', 'int_col'을 모델이 이해할 수 있는 수치형 형식으로 변환하기 위해 더미 변수(dummy variable) 인코딩을 적용했습니다. 각 범주형 변수를 고유한 값마다 더미 변수로 변환하여, 범주를 0과 1로 표시된 형태로 표현했습니다. 예를 들어, 'fuel_type' 변수가 'gasoline', 'diesel'과 같은 여러 범주로 구성된 경우, 각 범주에 대해 별도의 열을 생성하고 해당 값이 0 또는 1로 할당합니다. 이를 통해 모델이 범주형 데이터의 유의미한 정보를 수치적으로 해석할 수 있게 되어 모델 성능에 효과적으로 기여할 수 있도록 하였다고 생각합니다.

2-4 'engine' 컬럼에서 마력과 배기량 추출

```
def extract_engine_info(engine):
    hp_match = re.search(r'(₩d+₩.?₩d*)HP', engine)
    liter_match = re.search(r'(₩d+₩.?₩d*)L', engine)
    hp = float(hp_match.group(1)) if hp_match else None
    liters = float(liter_match.group(1)) if liter_match else None
    return pd.Series({'horsepower': hp, 'liters': liters})

engine_info = vehicle_data_encoded['engine'].apply(extract_engine_info)
vehicle_data_encoded = pd.concat([vehicle_data_encoded.drop(columns=['engine']), engine_info], axis=1)
vehicle_data_encoded['engine_output'] = vehicle_data_encoded['horsepower'] * vehicle_data_encoded['liters']
```

차량의 'engine' 컬럼에는 마력(Horse Power)과 배기량(Liters) 정보가 포함되어 있어, 이를 정규 표현식을 사용해 'horsepower'와 'liters'라는 두 개의 새로운 수치형 컬럼으로 분리하여 추출했습니다. 정규 표현식을 통해 'HP'와 'L' 값만을 문자열에서 찾아내어 수치형으로 변환하고 결측값이 발생하는 경우에는 None으로 처리하여 데이터의 일관성을 유지했습니다. 또한, 차량의 성능을 구체적으로 반영하기 위해 'horsepower'와 'liters' 값을 곱하여 'engine_output'이라는 새로운 특성을 추가했습니다. 이로써 모델이 차량의 엔진 성능을 더 효과적으로 학습할 수 있게 하여 예측 성능을 향상시킬 수 있을 것으로 기대됩니다.

2-5 결측치 처리

```
vehicle_data_encoded['horsepower'] = vehicle_data_encoded['horsepower'].fillna(vehicle_data_encoded['horsepower'].mean())
vehicle_data_encoded['engine_output'] = vehicle_data_encoded['engine_output'].fillna(vehicle_data_encoded['engine_output'].mean())
vehicle_data_encoded['liters'] = vehicle_data_encoded['liters'].fillna(vehicle_data_encoded['liters'].mean())

# 최종 확인
vehicle_data_encoded[['horsepower', 'engine_output', 'liters']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3269 entries, 0 to 4008
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   horsepower     3269 non-null   float64
 1   engine_output  3269 non-null   float64
 2   liters         3269 non-null   float64
dtypes: float64(3)
memory usage: 102.2 KB
```

모델 학습의 안정성과 데이터 일관성을 확보하기 위해 'horsepower', 'engine_output', 'liters' 컬럼의 결측치를 각각 해당 컬럼의 평균값으로 대체했습니다. 결측치를 그대로 두면 모델이 불완전한 정보로 인해 예측 성능이 저하되거나 학습 과정에서 오류가 발생할 수 있기 때문에, 결측치를 평균값으로 채워 넣어 정보 손실을 최소화하고자 했습니다. 특히, 이들 특성은 차량의 성능을 나타내는 중요한 변수들이므로, 이러한 결측치 처리를 통해 모델이 일관된 데이터를 기반으로 안정적으로 학습할 수 있도록 했습니다.

2-6 이상치 제거 및 타깃 변수 정규화

```
for feature in ['milage', 'price', 'horsepower', 'liters']:
    Q1 = vehicle_data_encoded[feature].quantile(0.25)
    Q3 = vehicle_data_encoded[feature].quantile(0.75)
    IQR = Q3 - Q1
    vehicle_data_encoded = vehicle_data_encoded[~((vehicle_data_encoded[feature] < (Q1 - 1.5 * IQR)) |
                                                  (vehicle_data_encoded[feature] > (Q3 + 1.5 * IQR)))]
vehicle_data_encoded['price'] = np.log1p(vehicle_data_encoded['price'])
```

모델이 극단적인 값에 과도하게 영향을 받지 않도록 'milage', 'price', 'horsepower', 'liters' 컬럼에서 1.5 IQR(Interquartile Range) 기준을 초과하는 이상치를 제거하였습니다. 이상치는 데이터 분포를 왜곡하고 모델 학습 시 불필요한 변동성을 증가시킬 수 있기 때문에, 1.5 IQR 기준을 사용하여 각 특성의 일반적인 분포 범위를 벗어나는 데이터 포인트를 필터링했습니다. 이를 통해 모델이 데이터의 주요 패턴을 더 효과적으로 학습할 수 있도록 지원하였습니다.

또한, 타깃 변수인 'price'에는 로그 변환을 적용하여 왜도(skewness)를 줄였습니다. 가격 데이터는 원래 오른쪽으로 치우친 분포를 가질 가능성이 높기 때문에, 로그 변환을 통해 이러한 왜곡을 완화하여 모델이 가격 변화를 더 균형 있게 학습할 수 있도록 했습니다. 이를 통해 모델의 예측 성능을 안정화하고, 데이터 분포가 정규 분포에 가까워지도록 하여 회귀 분석의 가정을 충족시키는 데 기여했습니다.

2-7 특성 상호작용 생성

```
vehicle_data_encoded['milage_horsepower_interaction'] = vehicle_data_encoded['milage'] * vehicle_data_encoded['horsepower']
```

주행 거리(milage)와 마력(horsepower)의 상호작용을 반영한 milage_horsepower 컬럼을 추가하였습니다. 이 상호작용 특성은 차량의 주행 거리와 엔진 성능이 중고차 가격에 미치는 복합적인 영향을 모델이 학습할 수 있도록 하기 위해 생성되었습니다. 일반적으로 주행 거리가 많을수록 차량의 가치가 낮아지지만, 높은 마력을 가진 차량은 더 높은 가격을 유지할 가능성이 있습니다. 이러한 상호작용을 모델에 포함함으로써, 두 특성이 결합된 조건에서 중고차 가격에 미치는 영향을 더 정확하게 반영할 수 있도록 하였습니다. 따라서 이 상호작용 특성이 모델의 예측 성능을 향상시키는 데 기여하도록 만들었습니다.

2-8 타깃 변수와의 상관관계를 기반으로 불필요한 특성 제거

```
price_corr = vehicle_data_encoded.corr()['price'].drop('price')
low_corr_threshold = 0.2
low_corr_features = price_corr[price_corr.abs() < low_corr_threshold].index
vehicle_data_encoded_filtered = vehicle_data_encoded.drop(columns=low_corr_features)
```

타깃 변수인 price와의 상관관계가 0.2 미만인 특성들은 중고차 가격 예측에 큰 기여를 하지 않을 가능성이 높아 제거하였습니다. 상관관계가 낮은 특성들은 타깃 변수와의 연관성이 약해 모델에 유의미한 정보를 제공하지 못할 수 있으며 이를 포함할 경우 모델의 복잡도가 불필요하게 증가할 우려가 있습니다. 따라서 이러한 특성들을 제거함으로써 학습 속도를 향상시키고 과적합(overfitting) 가능성을 줄여 모델의 일반화 성능을 개선하고자 하였습니다.

2-9 상관관계가 높은 특성 쌍 중 하나 제거

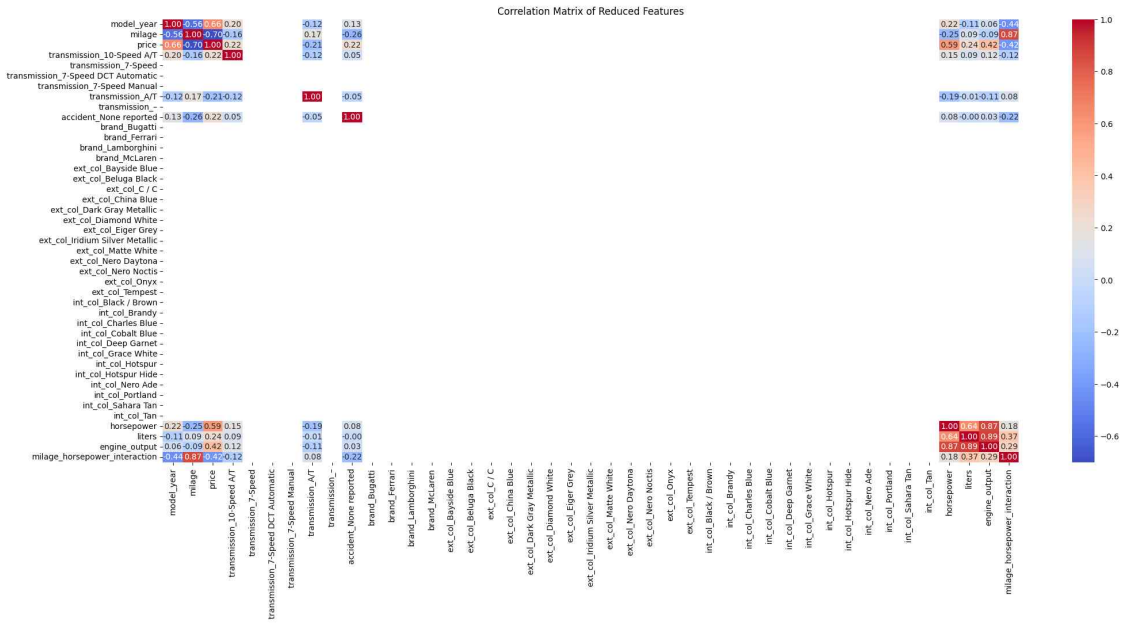
```
corr_matrix = vehicle_data_encoded_filtered.corr().abs()
high_corr_pairs = [(corr_matrix.columns[i], corr_matrix.columns[j])
                   for i in range(len(corr_matrix.columns))
                   for j in range(i)
                   if corr_matrix.iloc[i, j] > 0.9]
features_to_drop = {pair[1] for pair in high_corr_pairs}
vehicle_data_encoded_reduced = vehicle_data_encoded_filtered.drop(columns=features_to_drop)

vehicle_data_encoded_reduced.info()
```

```
plt.figure(figsize=(24, 10))
sns.heatmap(vehicle_data_encoded_reduced.corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title("Correlation Matrix of Reduced Features")
plt.show()
```

```
vehicle_data_encoded_reduced = vehicle_data_encoded_reduced.drop(columns=['horsepower', 'liters'])
vehicle_data_encoded_reduced.info()
```

#	Column	Non-Null Count	Dtype
0	model_year	2995 non-null	int64
1	milage	2995 non-null	int64
2	price	2995 non-null	float64
3	transmission_10-Speed A/T	2995 non-null	bool
4	transmission_7-Speed	2995 non-null	bool
5	transmission_7-Speed DCT Automatic	2995 non-null	bool
6	transmission_7-Speed Manual	2995 non-null	bool
7	transmission_A/T	2995 non-null	bool
8	transmission_-	2995 non-null	bool
9	accident_None reported	2995 non-null	bool
10	brand_Bugatti	2995 non-null	bool
11	brand_Ferrari	2995 non-null	bool
12	brand_Lamborghini	2995 non-null	bool
13	brand_McLaren	2995 non-null	bool
14	ext_col_Bayside Blue	2995 non-null	bool
15	ext_col_Beluga Black	2995 non-null	bool
16	ext_col_C / C	2995 non-null	bool
17	ext_col_China Blue	2995 non-null	bool
18	ext_col_Dark Gray Metallic	2995 non-null	bool
19	ext_col_Diamond White	2995 non-null	bool
20	ext_col_Eiger Grey	2995 non-null	bool
21	ext_col_Iridium Silver Metallic	2995 non-null	bool
22	ext_col_Matte White	2995 non-null	bool
23	ext_col_Nero Daytona	2995 non-null	bool
24	ext_col_Nero Noctis	2995 non-null	bool
25	ext_col_Onyx	2995 non-null	bool
26	ext_col_Tempest	2995 non-null	bool
27	int_col_Black / Brown	2995 non-null	bool
28	int_col_Brandy	2995 non-null	bool
29	int_col_Charles Blue	2995 non-null	bool
30	int_col_Cobalt Blue	2995 non-null	bool
31	int_col_Deep Garnet	2995 non-null	bool
32	int_col_Grace White	2995 non-null	bool
33	int_col_Hotspur	2995 non-null	bool
34	int_col_Hotspur Hide	2995 non-null	bool
35	int_col_Nero Ade	2995 non-null	bool
36	int_col_Portland	2995 non-null	bool
37	int_col_Sahara Tan	2995 non-null	bool
38	int_col_Tan	2995 non-null	bool
39	horsepower	2995 non-null	float64
40	liters	2995 non-null	float64
41	engine_output	2995 non-null	float64
42	milage_horsepower_interaction	2995 non-null	float64

dtypes: bool(36), float64(5), int64(2)
memory usage: 292.5 KB



```
 0   model_year                          2995 non-null   int64
 1   milage                              2995 non-null   int64
 2   price                               2995 non-null   float64
 3   transmission_10-Speed A/T           2995 non-null   bool
 4   transmission_7-Speed                2995 non-null   bool
 5   transmission_7-Speed DCT Automatic  2995 non-null   bool
 6   transmission_7-Speed Manual         2995 non-null   bool
 7   transmission_A/T                    2995 non-null   bool
 8   transmission_-                      2995 non-null   bool
 9   accident_None reported              2995 non-null   bool
 10  brand_Bugatti                       2995 non-null   bool
 11  brand_Ferrari                       2995 non-null   bool
 12  brand_Lamborghini                   2995 non-null   bool
 13  brand_McLaren                       2995 non-null   bool
 14  ext_col_Bayside Blue                2995 non-null   bool
 15  ext_col_Beluga Black                2995 non-null   bool
 16  ext_col_C / C                       2995 non-null   bool
 17  ext_col_China Blue                  2995 non-null   bool
 18  ext_col_Dark Gray Metallic          2995 non-null   bool
 19  ext_col_Diamond White               2995 non-null   bool
 20  ext_col_Eiger Grey                  2995 non-null   bool
 21  ext_col_Iridium Silver Metallic     2995 non-null   bool
 22  ext_col_Matte White                 2995 non-null   bool
 23  ext_col_Nero Daytona                2995 non-null   bool
 24  ext_col_Nero Noctis                 2995 non-null   bool
 25  ext_col_Onyx                        2995 non-null   bool
 26  ext_col_Tempest                     2995 non-null   bool
 27  int_col_Black / Brown               2995 non-null   bool
 28  int_col_Brandy                      2995 non-null   bool
 29  int_col_Charles Blue                2995 non-null   bool
 30  int_col_Cobalt Blue                 2995 non-null   bool
 31  int_col_Deep Garnet                 2995 non-null   bool
 32  int_col_Grace White                 2995 non-null   bool
 33  int_col_Hotspur                     2995 non-null   bool
 34  int_col_Hotspur Hide                2995 non-null   bool
 35  int_col_Nero Ade                    2995 non-null   bool
 36  int_col_Portland                    2995 non-null   bool
 37  int_col_Sahara Tan                  2995 non-null   bool
 38  int_col_Tan                         2995 non-null   bool
 39  engine_output                       2995 non-null   float64
 40  milage_horsepower_interaction       2995 non-null   float64
dtypes: bool(36), float64(3), int64(2)
```

이 단계에서는 상관관계가 높은 특성 쌍을 찾아 제거하여 모델이 중복 정보에 의한 과적합(overfitting)을 피하고, 더 중요한 특성에 집중할 수 있도록 했습니다. 이를 통해 모델의 성능과 학습 안정성을 향상시킬 수 있습니다.

먼저, vehicle_data_encoded_filtered.corr()을 사용하여 각 특성 간 상관관계 행렬을 생성했습니다. 상관계수는 -1에서 1 사이의 값을 가지며, 1에 가까울수록 강한 양의 상관관계를, -1에 가까울수록 강한 음의 상관관계를, 0에 가까울수록 두 특성 간 관계가 거의 없음을 의미합니다. 이 행렬을 통해 상관관계가 매우 높은 특성 쌍을 탐지하고, 중복 정보를 가진 특성을 제거하여 모델의 복잡도를 낮추는 것이 목적이었습니다.

이후, 상관계수가 0.9 이상인 특성 쌍을 high_corr_pairs 리스트에 추가하여 탐지하였습니다. 상관계수가 0.9를 초과한다는 것은 두 특성이 거의 동일한 정보를 가지고 있음을 의미하므로, 이러한 특성을 모두 포함할 경우 모델에 불필요한 중복이 발생해 과적합이 우려됩니다. 따라서 상관관계가 높은 특성 쌍 중 하나를 선택적으로 제거하여, 모델이 더 유용한 특성에 집중할 수 있도록 하였습니다.

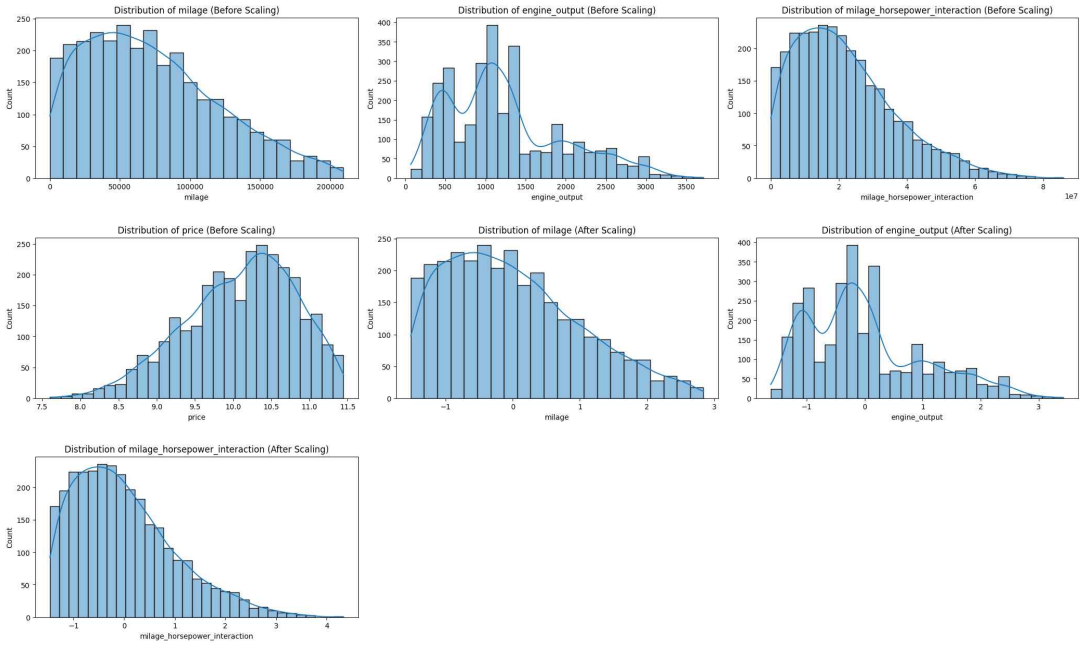
불필요한 특성을 제거한 후, 최종 데이터셋(vehicle_data_encoded_reduced)의 상관관계 행렬을 시각화하여 남은 특성들 간 관계를 다시 확인했습니다. 이 시각화 과정을 통해 최종적으로 남은 특성들이 예측에 유용한 정보를 제공하며, 불필요한 중복이 제거되었는지 검토할 수 있었습니다.

마지막으로, 최종 데이터셋의 구조를 확인하여 모든 특성이 예측에 사용될 준비가 되었음을 확인했습니다. 최종적으로 43개의 특성으로 구성된 데이터셋을 얻었으며, 이는 모델이 학습에 사용할 최적화된 데이터셋입니다.

이 과정은 데이터셋의 특성 수를 줄이고 과적합을 방지하여 모델의 효율성을 높이는 중요한 단계였습니다. 최종 데이터셋은 모델이 불필요한 정보를 학습하지 않고, 중요한 특성에 집중할 수 있도록 하여 예측의 신뢰성을 높였습니다.

2-10 스케일링 전처리 적용

```
numeric_features = ['milage', 'engine_output', 'milage_horsepower_interaction', 'price']

# 스케일링 전 데이터 분포 시각화
for feature in numeric_features:
    plt.figure(figsize=(8, 4))
    sns.histplot(vehicle_data_encoded_reduced[feature].dropna(), kde=True)
    plt.title(f'Distribution of {feature} (Before Scaling)')
    plt.xlabel(feature)
    plt.show()

# 스케일링
scaler = StandardScaler()
vehicle_data_encoded_reduced[numeric_features] = scaler.fit_transform(vehicle_data_encoded_reduced[numeric_features])

# 스케일링 후 데이터 분포 시각화
for feature in numeric_features[:-1]:  # 'price' 제외
    plt.figure(figsize=(8, 4))
    sns.histplot(vehicle_data_encoded_reduced[feature].dropna(), kde=True)
    plt.title(f'Distribution of {feature} (After Scaling)')
    plt.xlabel(feature)
    plt.show()
```



모델이 특성 간의 스케일 차이에 영향을 받지 않도록 표준화를 적용하여, 'milage', 'engine_output', 'milage_horsepower_interaction', 'price' 특성들의 스케일을 조정하였습니다. 표준화는 각 값을 해당 컬럼의 평균이 0이고, 표준편차가 1인 정규 분포로 변환하여, 모든 특성이 동일한 스케일로 모델에 입력되도록 하는 과정입니다. 이를 통해 모델은 데이터의 절대값보다 특성 간의 패턴과 관계에 집중하게 되어, 각 특성의 영향이 더 균등하게 반영될 수 있습니다.

그래서 첫 번째 단계로 표준화를 적용하기 전 각 특성의 분포를 시각화하였습니다. Before Scaling 히스토그램에서는 'milage', 'engine_output', 'milage_horsepower_interaction', 'price' 각각의 값이 매우 다른 범위를 가지는 것을 확인할 수 있습니다. 예를 들어, 'milage'는 0에서 약 200,000까지의 값을 가지며, 'engine_output'은 0에서 약 3,500 사이로 분포하

고 있습니다. 이러한 스케일 차이는 모델이 특성을 불균형하게 인식할 가능성을 높입니다.

표준화 후, After Scaling 히스토그램에서는 모든 특성들이 평균 0, 분산 1의 스케일로 변환된 것을 확인할 수 있습니다. 이렇게 변환된 특성들은 모델이 절대값에 의존하지 않고, 특성 간의 상대적 변화와 패턴을 바탕으로 예측을 수행할 수 있게 해주며 모델의 안정성과 예측 성능을 향상 시키는 역할을 하는 것을 알 수 있습니다.

2-11 훈련, 검증, 테스트 데이터 세트 분할

```
X = vehicle_data_encoded_reduced.drop(columns=['price'])  # 타깃 변수 'price' 제외
y = vehicle_data_encoded_reduced['price']  # 타깃 변수 설정
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

전처리가 완료된 데이터를 훈련, 검증, 테스트 세트로 분할하여 각 세트의 역할에 맞게 활용할 수 있도록 하였습니다. 훈련 세트는 모델 학습에, 검증 세트는 하이퍼파라미터 조정 및 과적합 방지에, 테스트 세트는 최종 성능 평가에 사용되었습니다. 이 분할을 통해 모델의 학습 성능을 효과적으로 검증하고, 일반화 능력을 평가하여 실제 환경에서의 성능을 가늠할 수 있도록 하였습니다.

3. 회귀 및 파라미터 최적화

3-1 모델 정의 및 파라미터 그리드 설정

```
models = {
    "Linear Regression": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', LinearRegression())
    ]),
    "Lasso Regression": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', Lasso())
    ]),
    "Ridge Regression": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', Ridge())
    ]),
    "Random Forest": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', RandomForestRegressor())
    ]),
    "Gradient Boosting": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', GradientBoostingRegressor())
    ]),
    "K-Nearest Neighbors": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', KNeighborsRegressor())
    ]),
    "Bagging Regressor": Pipeline([
        ('scaler', StandardScaler()),
        ('regressor', BaggingRegressor())
    ])
}
```

다양한 회귀 모델을 정의하여 성능을 비교하고 최적의 모델을 선택하고자 하였습니다. 구체적으로 Linear Regression, Lasso, Ridge, Random Forest, Gradient Boosting, K-Nearest Neighbors (KNN) 그리고 Bagging Regressor 등 총 7개의 회귀 모델을 구성하였습니다. 각 모델은 서로 다른 학습 방식과 특성을 가지고 있어서 데이터 특성에 따라 최적의 예측 성능을 보일 가능성이 있다고 생각하였습니다.

선형 모델인 Linear Regression은 데이터가 선형 관계일 때 간단하고 해석이 용이한 장점을 가지고 있습니다. Lasso와 Ridge는 규제를 통해 과적합(overfitting)을 방지하며, 많은 특성을 가진 데이터에서 불필요한 특성을 제거하여 모델의 복잡성을 줄이는 데 유리합니다.

비선형 관계를 잘 학습하는 Random Forest와 Gradient Boosting 같은 앙상블 모델은 특히 복잡한 데이터셋에서 높은 성능을 발휘할 수 있습니다. Random Forest는 여러 결정 트리를 통해 예측의 안정성과 정확성을 높이고, Gradient Boosting은 반복 학습을 통해 이전 예측의 오차를 줄이는 방식으로 성능을 향상시킵니다.

또한, K-Nearest Neighbors (KNN) 모델은 가장 가까운 이웃들의 값을 참조하여 새로운 데이터를 예측하는 방식으로 작동하여 패턴이 명확한 데이터에서 효과적인 성능을 발휘할 수 있습니다. 그러나 KNN은 고차원 데이터에서 성능 저하가 발생할 수 있는 단점이 있다고 아고 있으니 이 데이터에서는 좋은 성능을 내지 못할거라 예상을 하였습니다.

각 모델은 StandardScaler와 결합하여 파이프라인으로 구성되었습니다. 이 파이프라인은 데이터의 표준화를 일관되게 적용하여, 모델이 특성 간 스케일 차이에 영향을 받지 않도록 돕습니다. 특히 KNN과 같은 거리 기반 모델이나 Lasso, Ridge와 같은 규제 모델에서는 표준화가 필수적이기 때문에 이를 통해 모델 성능의 안정성을 높였습니다. 파이프라인 설정을 통해 데이터 전처리와 모델 학습 과정을 일관되게 연결하고, 하이퍼파라미터 최적화 시 표준화 단계가 포함된 모델을 손쉽게 사용할 수 있습니다. 이는 코드의 재사용성을 높이고, 동일한 전처리를 통해 각 모델의 평가 결과에 신뢰성을 부여합니다.

결론적으로, 다양한 모델을 설정하여 데이터 특성에 가장 적합한 모델을 선택할 수 있는 기반을 마련하였으며, 전처리와 평가 과정을 통일하여 효율적이고 일관된 분석을 진행할 수 있게 만들었습니다.

3-2 초기 모델 성능 평가

```
model_performance = {}
for model_name, model_pipeline in models.items():
    model_pipeline.fit(X_train, y_train)
    y_pred = model_pipeline.predict(X_val)

    # 평가 지표 계산
    mse = mean_squared_error(y_val, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_val, y_pred)
    r2 = r2_score(y_val, y_pred)

    print(f"{model_name} - MSE: {mse:.2f}, RMSE: {rmse:.2f}, MAE: {mae:.2f}, R² : {r2:.2f}")
    model_performance[model_name] = {"MSE": mse, "RMSE": rmse, "MAE": mae, "R² ": r2}
```

```
Linear Regression - MSE: 0.27, RMSE: 0.52, MAE: 0.39, R² : 0.73
Lasso Regression - MSE: 0.99, RMSE: 1.00, MAE: 0.82, R² : -0.00
Ridge Regression - MSE: 0.27, RMSE: 0.52, MAE: 0.39, R² : 0.73
Random Forest - MSE: 0.24, RMSE: 0.49, MAE: 0.37, R² : 0.76
Gradient Boosting - MSE: 0.23, RMSE: 0.48, MAE: 0.36, R² : 0.77
K-Nearest Neighbors - MSE: 0.26, RMSE: 0.51, MAE: 0.40, R² : 0.73
Bagging Regressor - MSE: 0.28, RMSE: 0.53, MAE: 0.40, R² : 0.72
```

모든 모델을 훈련 세트에 학습시킨 후, 검증 세트에서 성능을 평가하여 각 모델의 초기 성능을 파악했습니다. 이 과정에서는 다양한 평가 지표를 사용해 모델 간의 성능을 비교하고 분석했습니다. 사용된 평가 지표는 MSE (Mean Squared Error), RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), 그리고 결정계수인 R^2로, 각 지표가 모델의 예측 정확도와 신뢰성을 다각도로 보여줍니다.

MSE (Mean Squared Error)는 예측값과 실제값의 차이를 제곱한 후 평균을 구한 값으로, 오차가 클수록 값이 커집니다. MSE가 낮을수록 모델의 예측이 실제 값과 가까운 것을 의미합니다.

RMSE (Root Mean Squared Error)는 MSE의 제곱근으로, MSE와 같은 의미를 가지지만 오차의 크기를 더 직관적으로 파악할 수 있도록 도와줍니다.

MAE (Mean Absolute Error)는 예측값과 실제값의 차이의 절대값 평균으로, 오차의 절대적인 크기를 측정합니다. 낮을수록 예측의 정확도가 높은 것을 나타냅니다.

R^2 (결정계수)는 모델이 타깃 변수의 변동성을 얼마나 잘 설명하는지를 나타내며, 값이 1에 가까울수록 좋은 성능을 의미합니다.

초기 성능 평가 결과, Gradient Boosting과 Random Forest는 상대적으로 낮은 MSE와 높은 R^2를 기록하며 우수한 성능을 보였습니다. 반면, Lasso Regression은 예측 성능이 낮아 MSE가 높고 R^2가 매우 낮은 결과를 보였습니다. 이러한 초기 평가를 통해, 최적화 단계에서 어떤 모델이 발전 가능성이 있는지, 각 모델이 데이터의 특성을 얼마나 잘 반영하는지를 판단할 수 있었습니다.

3-3 중요 특성 확인

```python
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
important_features = np.where(lasso.coef_ != 0)[0]
selected_features = X.columns[important_features]
print("Selected important features:", selected_features)
```

```
Selected important features: Index(['model_year', 'milage', 'engine_output'], dtype='object')
```

Lasso 회귀 모델을 활용하여 데이터의 중요 특성을 선별하고, 예측 성능을 향상시키기 위해 유의미한 특성만을 선택했습니다. Lasso는 회귀 계수에 패널티를 부여하여 일부 계수를 0으

로 만듦으로써, 불필요하거나 예측 성능에 기여하지 않는 특성을 자동으로 제거하는 특성이 있습니다. 이를 통해 모델의 해석 가능성을 높이고 복잡도를 줄여 과적합을 효과적으로 방지했다고 생각합니다.

Lasso를 사용한 이유는, 간결하면서도 효율적으로 특성 선택을 수행할 수 있기 때문입니다. 규제 기법을 통해 최적의 특성 조합을 선정하여 분석 초기 단계에서 특성의 유의미성을 신속하게 평가하는 데 적합합니다.

이번 코드결과 에서는 Lasso를 통해 'model_year', 'milage', 'engine_output'이 중고차 가격 예측에 중요한 특성으로 선택되었으며, 이를 통해 모델의 성능을 높이고 학습 시간을 단축할 수 있었습니다.

3-4 파라미터 최적화

```
param_grids = {
    "Lasso Regression": {'regressor__alpha': [0.01, 0.1, 1, 10]},
    "Ridge Regression": {'regressor__alpha': [0.01, 0.1, 1, 10]},
    "Random Forest": {
        'regressor__n_estimators': [50, 100, 200],
        'regressor__max_depth': [None, 10, 20]
    },
    "Gradient Boosting": {
        'regressor__n_estimators': [50, 100, 200],
        'regressor__learning_rate': [0.01, 0.1, 0.5],
        'regressor__max_depth': [3, 5, 7]
    },
    "K-Nearest Neighbors": {'regressor__n_neighbors': [3, 5, 7, 9]}
}
```

```
best_models = {}
best_scores = {}

# 각 모델에 대해 GridSearchCV 수행
for model_name, model_pipeline in models.items():
    if model_name in param_grids:
        print(f"Optimizing {model_name}...")
        grid_search = GridSearchCV(model_pipeline, param_grids[model_name], cv=5, scoring='neg_mean_squared_error')
        grid_search.fit(X_train, y_train)

        # 최적 파라미터와 성능 저장
        best_models[model_name] = grid_search.best_estimator_
        best_scores[model_name] = -grid_search.best_score_  # MSE이므로 음수로 변환
        print(f"Best parameters for {model_name}: {grid_search.best_params_}")
        print(f"Best MSE score for {model_name}: {best_scores[model_name]:.2f}")
```

```
Optimizing Lasso Regression...
Best parameters for Lasso Regression: {'regressor__alpha': 0.01}
Best MSE score for Lasso Regression: 0.26
Optimizing Ridge Regression...
Best parameters for Ridge Regression: {'regressor__alpha': 1}
Best MSE score for Ridge Regression: 0.26
Optimizing Random Forest...
Best parameters for Random Forest: {'regressor__max_depth': 10, 'regressor__n_estimators': 200}
Best MSE score for Random Forest: 0.23
Optimizing Gradient Boosting...
Best parameters for Gradient Boosting: {'regressor__learning_rate': 0.1, 'regressor__max_depth': 5, 'regressor__n_estimators': 100}
Best MSE score for Gradient Boosting: 0.23
Optimizing K-Nearest Neighbors...
Best parameters for K-Nearest Neighbors: {'regressor__n_neighbors': 9}
Best MSE score for K-Nearest Neighbors: 0.27
```

각 회귀 모델에 대해 하이퍼파라미터 튜닝을 수행하여 최적의 파라미터 조합을 찾아냈고, 가장 낮은 MSE(Mean Squared Error)를 기록한 모델을 최종 모델로 선정했습니다. 결과에 따르면, Gradient Boosting 모델이 가장 낮은 MSE(0.23)를 기록하여 검증 세트에서 최적의 성능을 보였고, 이를 최종 테스트 세트 평가에 사용할 모델로 결정했습니다.

하이퍼파라미터 최적화 과정에서 Lasso와 Ridge Regression은 알파(alpha) 값을 조정했고, Random Forest와 Gradient Boosting은 트리의 깊이(max_depth)와 추정기 개수(n_estimators)를 그리고 Gradient Boosting에서는 학습률(learning_rate)도 함께 최적화했습니다. K-Nearest Neighbors에서는 n_neighbors 값을 최적화하여, 각 모델이 데이터에 가장 적합한 설정을 찾을 수 있도록 만들었습니다.

4. 결과 및 분석

```
best_model_name = min(best_scores, key=best_scores.get)
print(f"\nBest Model after Parameter Optimization: {best_model_name} with MSE = {best_scores[best_model_name]:.2f}")
```

```
final_model = best_models[best_model_name]
y_test_pred = final_model.predict(X_test)

# 평가 지표 계산
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
test_mae = mean_absolute_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"\nFinal Model Performance on Test Data - {best_model_name}")
print(f"MSE: {test_mse:.2f}, RMSE: {test_rmse:.2f}, MAE: {test_mae:.2f}, R² : {test_r2:.2f}")
```

```
Final Model Performance on Test Data - Gradient Boosting
MSE: 0.24, RMSE: 0.49, MAE: 0.35, R² : 0.79
```

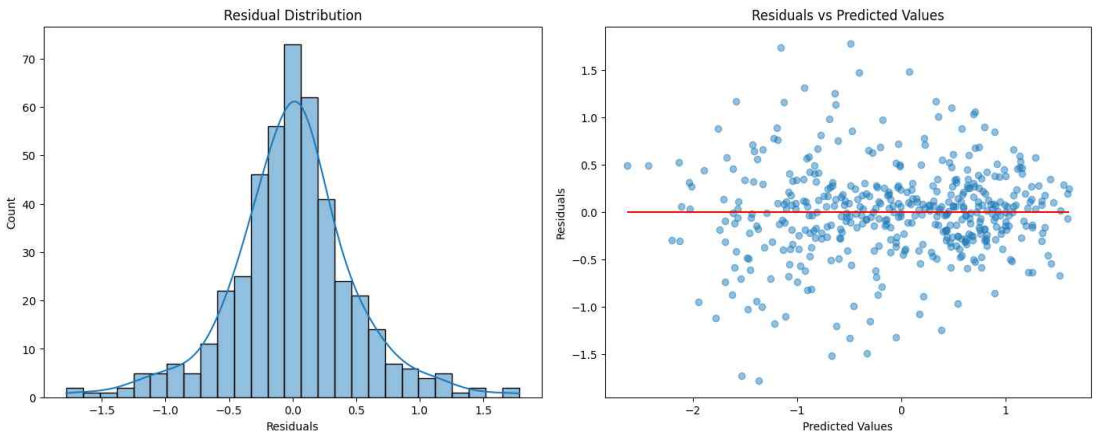
최종적으로 선택된 모델을 테스트 세트에 적용하여 MSE(Mean Squared Error), RMSE(Root Mean Squared Error), MAE(Mean Absolute Error), R^2(결정 계수) 지표를 통해 모델의 예측 성능을 다각도로 평가하였습니다. MSE와 RMSE는 예측 값과 실제 값 간의 차이를 평균적으로 측정하는 지표입니다. MSE는 오차를 제곱하여 평균하므로, 큰 오차에 더 민감하며,RMSE는 MSE의 제곱근으로 직관적인 해석을 가능하게 합니다. 낮은 MSE와 RMSE는 예측이 실제 값에 가까움을 의미합니다.

MAE는 예측값과 실제값 사이의 절대 오차를 평균하여, 각 오차의 크기를 직접적으로 반영하는 지표로, 오차가 양쪽으로 대칭적일 때 직관적인 오류의 크기를 파악하는 데 유리합니다.

R^2는 모델이 데이터 변동성을 얼마나 설명하는지 나타내며, 1에 가까울수록 예측 모델이 데이터를 잘 설명하고 있음을 의미합니다.

이번 테스트 결과에서 최종 모델인 Gradient Boosting 모델은 MSE 0.24, RMSE 0.49, MAE 0.35, 그리고 R^2 0.79의 성능을 기록하였습니다. MSE와 RMSE가 낮고, R^2가 0.79로 1에 가깝다는 점에서 이 모델이 데이터를 잘 설명하며 정확한 예측을 수행하고 있음을 확인할 수 있습니다. 따라서 Gradient Boosting이 데이터의 복잡한 패턴을 잘 학습하였다고 해석할 수 있습니다.

```python
# 잔차(residual) = 실제 값 - 예측 값
residuals = y_test - y_test_pred

# 잔차 분포 시각화
plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True)
plt.title("Residual Distribution")
plt.xlabel("Residuals")
plt.show()

# 예측 값 vs 잔차
plt.figure(figsize=(8, 6))
plt.scatter(y_test_pred, residuals, alpha=0.5)
plt.hlines(y=0, xmin=min(y_test_pred), xmax=max(y_test_pred), colors='r')
plt.title("Residuals vs Predicted Values")
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.show()
```



추가적으로 잔차 분석도 진행하였습니다. 잔차 분석은 모델의 예측 정확도와 오차의 분포를 이해하는 데 중요한 과정이며 예측의 신뢰성을 평가하는 데 필수적인 요소라고 생각합니다. 잔차는 "실제 값 - 예측 값"으로 말할 수 있고 모델이 각 데이터 포인트에서 발생하는 오차를 의미합니다. 이러한 잔차를 분석함으로써 모델이 예측에 있어서 편향이나 일정한 패턴을 보이는지 평가할 수 있습니다.

왼쪽 히스토그램은 잔차 분포를 시각화한 것입니다. 잔차의 분포를 히스토그램과 커널 밀도 추정(KDE)으로 나타내어, 잔차가 평균 0을 중심으로 대칭적인 분포를 이루는지 확인하였습니다. 히스토그램에서 볼 수 있듯이 잔차 분포는 대체로 평균 0을 중심으로 대칭적입니다. 이를 통해 모델이 과도하게 높은 값이나 낮은 값을 예측하는 경향이 없으며 예측이 전반적으로 균형 잡혀 있음을 확인할 수 있습니다. 이러한 분포는 모델이 데이터의 전체 패턴을 잘 반영하고 있으며 전반적으로 일관된 예측을 수행하고 있다는 긍정적인 지표라고 생각합니다.

오른쪽은 예측 값과 잔차의 관계를 산점도로 나타낸 것입니다. 예측 값과 잔차 간의 관계를

확인하기 위해 산점도를 생성하였으며, 이 산점도는 예측 값 크기에 따라 잔차가 특정 패턴을 보이는지를 시각적으로 평가하는 데 유용합니다. 따라서 산점도를 보면 잔차가 예측 값의 크기와 관계없이 고르게 분포하는 모습을 확인할 수 있습니다. 이는 모델이 특정한 범위에서 과적합이나 과소적합 없이 예측을 수행하고 있음을 나타내며 예측 성능이 데이터 전반에 걸쳐 안정적으로 유지되고 있다는걸 알 수 있습니다. 예를 들어서, 잔차가 특정 범위에서만 클 경우 해당 범위의 예측 성능이 낮다는 생각할 수 있는데 제 결과에서는 이러한 문제가 발견되지 않았습니다.

이러한 성능 평가와 잔차 분석을 통해 Gradient Boosting 모델이 데이터의 패턴을 잘 학습하고 있으며, 예측 오차가 작고 일관되게 분포하고 있음을 확인할 수 있었습니다. 특히 잔차가 예측 값에 대해 고르게 분포하여 특정한 패턴이 나타나지 않는 것은 모델이 편향되지 않고, 데이터의 모든 범위에 대해 균일한 성능을 보여주고 있다고 해석할 수 있습니다. 또한, 높은 R^2 값은 모델이 중고차 가격 예측에 필요한 주요 특성들을 효과적으로 학습하고 있음을 보여줍니다.

결론적으로, 이번 분석을 통해 최종 선택된 Gradient Boosting 모델이 중고차 가격 예측 문제에 적합한 모델임을 입증하였습니다. 다양한 평가 지표와 잔차 분석 결과 모두에서 모델이 높은 예측 성능을 보이며, 데이터의 복잡한 패턴을 잘 반영하고 있음을 확인할 수 있었습니다. 그리고 성능 평가를 통해 최종적으로 선택된 모델이 훈련 및 검증 세트에서 보인 성능이 테스트 세트에서도 일관되게 유지됨을 확인하였습니다. 이는 모델이 새로운 데이터에 대해서도 일반화 능력을 갖추고 있음을 의미하며 실제 응용 상황에서도 신뢰할 수 있는 예측 성능을 보일 가능성이 높음을 의미한다고 생각합니다. 이러한 결과는 모델이 실제 응용 환경에서도 신뢰성 있는 예측을 수행할 가능성이 높음을 시사하며 향후 모델의 활용성을 높이는 기반이 될 것입니다.

5. 고찰

이번 프로젝트 회귀 파트에서는 중고차 가격 예측이라는 회귀 문제를 해결하기 위해 다양한 접근법과 모델링 기법을 시도하며 회귀 모델을 구축하는 전 과정을 경험할 수 있었습니다. 회귀 분석의 각 단계에서 맞닥뜨린 문제와 도전 과제를 해결하는 과정에서, 데이터 전처리, 모델 선택, 하이퍼파라미터 최적화, 성능 평가까지 깊이 있는 고민을 거치며 진행했습니다. 특히, 중고차 데이터의 특성을 분석하고 최적의 모델을 찾기 위해 많은 시도와 선택을 하였으며, 이를 통해 다양한 인사이트를 얻을 수 있었습니다.

프로젝트 초기 단계에서는 데이터 전처리에 상당한 시간을 투자했습니다. 중고차 데이터는 milage, engine_output, milage_horsepower_interaction 등 다양한 수치형 변수를 포함하고 있었으며, 이러한 특성들이 중고차 가격 예측에 중요한 역할을 할 것으로 예상되었습니다. 데이터 탐색 과정에서 변수 간 상관관계를 확인하기 위해 상관 행렬을 사용했고, 이를 통해 여러 변수 간의 관계를 이해하려 했습니다. 예를 들어, milage와 price 간의 부정적 상관관계는 예상대로 확인할 수 있었으나, engine_output과 같은 변수는 예측에 어느 정도 기여할지에 대해 깊이 고민했습니다. 이 과정에서 불필요한 특성을 제거하거나 새롭게 파생 변수를 생성하여 모델 성능을 향상할 수 있는 가능성을 탐색했습니다.

결측치 처리는 중요한 결정 사항 중 하나였습니다. 회귀 모델이 결측치에 민감하게 반응할 수 있음을 염두에 두고, 결측치를 처리하는 방식에 대해 여러 시도를 했습니다. 평균이나 중앙값으로 결측치를 대체하거나, 결측치를 포함한 행을 제거하는 방법을 검토하였습니다. 다양한 방법을 적용해 보며 교차 검증을 통해 성능을 비교한 결과, 중앙값으로 결측치를 대체하는 것이 데이터 손실을 최소화하면서도 예측 성능을 유지하는 데 최적임을 확인했습니다. 이는 중앙값 대체가 평균보다 이상치에 덜 민감하여 데이터의 안정성을 유지할 수 있기 때문이라고 생각합니다. 이러한 결정을 통해 데이터를 최대한 활용하고 모델의 예측 성능을 유지할 수 있었습니다.

모델 선택 과정에서는 여러 회귀 모델을 테스트하고 최적의 모델을 찾기 위해 비교하였습니다. 선형 회귀 모델, Lasso, Ridge, Random Forest, Gradient Boosting 등 다양한 모델을 적용하여, 각각의 모델이 데이터에 대해 어떤 성능을 보이는지 평가했습니다. 선형 회귀 모델은 해석이 용이하고 단순한 모델로서 초기에 적용하기 적합했으나, 중고차 가격이라는 복잡한 변수 관계를 포착하는 데는 한계가 있음을 깨닫게 되었습니다. 특히, 복잡한 패턴을 가진 데이터에서는 선형적인 접근만으로는 충분하지 않다는 점을 인식하고, 비선형 관계를 더 잘 학습할 수 있는 앙상블 모델과 부스팅 모델을 추가로 고려하게 되었습니다.

Random Forest와 Gradient Boosting 모델은 비선형적이고 복잡한 관계를 잘 반영할 수 있어 이번 프로젝트에서 중요한 후보가 되었습니다. Random Forest는 다양한 특성의 상호작용을 효과적으로 학습하며, 과적합을 방지하기 위해 트리 개수와 깊이를 조정하는 방식으로 안정적인 성능을 확보할 수 있었습니다. Gradient Boosting은 학습률과 추정기의 개수를 최적화하여 점진적인 성능 향상을 목표로 하였고, 이는 데이터의 세부적인 패턴을 잘 학습하여 예측 성능을 높이는 데 기여했습니다. 특히 Gradient Boosting은 다양한 하이퍼파라미터 조합을 시도하며 최적의 성능을 찾기 위해 GridSearchCV를 통해 학습률, 트리 깊이 그리고 추정기 개수 등 파라미터를 조정하였습니다. 이러한 최적화 과정에서 학습률을 지나치게 높게 설정할 경우 과적합이 발생할 수 있어, 조심스럽게 설정을 조정해 가며 모델의 일반화 능력을 유지하려고 하였습니다.

모델의 예측 결과를 평가하는 과정에서, 성능 지표뿐만 아니라 잔차 분석을 통해 모델이 데이터의 전 범위에 걸쳐 일관된 성능을 보이는지 확인하고자 했습니다. 잔차 분석은 모델이 데이터의 특정 구간에서 과대평가나 과소평가를 하는지, 혹은 편향된 예측을 하는지 여부를 평가하는 데 중요한 도구였습니다. 잔차 분포를 히스토그램과 커널 밀도 추정(KDE)으로 시각화한 결과, 잔차가 평균 0을 중심으로 대칭적으로 분포하는 것을 확인하였고, 이를 통해 모델의 예측이 전반적으로 균형 잡혀 있음을 알 수 있었습니다. 예측 값과 잔차의 산점도를 통해 예측 값의 크기에 관계없이 잔차가 일정하게 분포하는지 확인하였으며, 특정 범위에서만 오차가 집중되지 않음을 알 수 있었습니다. 이는 모델이 데이터 전반에서 일관된 예측 성능을 보임을 시사하는 긍정적인 결과였습니다.

이번 프로젝트에서 회귀 모델의 성능을 다각도로 평가하기 위해 MSE, RMSE, MAE, R^2 등 다양한 지표를 사용하였습니다. MSE와 RMSE는 예측 오차의 크기를 직접적으로 반영하며, MAE는 절대 오차의 평균을 통해 예측 정확도를 평가할 수 있었습니다. 특히, Gradient Boosting 모델의 R^2 값이 0.79에 달하며 데이터 변동성의 상당 부분을 설명하는 데 성공하였다는 점에서 데이터의 주요 패턴을 효과적으로 학습하였음을 확인할 수 있었습니다.

이러한 지표들은 모델의 예측 성능을 정량적으로 평가하는 데 유용하였으며, 특히 Gradient Boosting 모델이 최종 모델로 선정되기까지의 중요한 결정 요소가 되었습니다. 낮은 MSE와

RMSE는 예측 값이 실제 값에 근접함을 의미하며, 높은 R^2 값은 모델이 데이터를 잘 설명하고 있음을 보여주었습니다. 다양한 지표를 통해 모델 성능을 종합적으로 평가함으로써, 회귀 문제에서 고려해야 할 다양한 관점을 배울 수 있었습니다.

이번 프로젝트는 성공적으로 중고차 가격 예측 모델을 구축했으나, 몇 가지 개선이 필요한 한계점도 확인되었습니다. 첫째, 특성 선택의 한계입니다. 이번 분석에서 사용한 특성들이 중고차 가격을 어느 정도 설명할 수 있었으나, 추가적으로 차량의 소유 이력, 유지보수 기록 등 실제로 가격에 영향을 미칠 수 있는 특성들이 포함되지 않아 한계가 있었습니다. 향후 이러한 추가 특성을 반영할 수 있는 데이터가 확보된다면, 모델의 예측 성능을 더 향상할 수 있을 것입니다.

둘째, 더 다양한 모델 및 하이퍼파라미터 조합을 시도하는 기회가 부족했던 점입니다. 특히, XGBoost나 LightGBM과 같은 최근에 배운 부스팅 모델들도 시도할 수 있었으나 시간적 제약으로 인해 최적화 과정에서 깊이 있게 탐구하지 못한 점이 아쉬웠습니다. 향후 이러한 모델들을 추가하여 성능을 비교하고, 모델 해석 기법을 통해 각 특성의 기여도를 더욱 자세히 파악할 수 있을 것입니다.

셋째, 데이터 전처리에서 다룬 결측치 처리 방식의 제한성입니다. 결측치를 중앙값으로 대체하였으나, 결측치 자체가 중요한 정보를 담고 있을 가능성을 고려하여 결측 여부를 별도의 특성으로 추가하는 방법도 효과적일 수 있다는 생각이 들었습니다. 추가한다면 이를 통해 결측이 발생한 차량 집단의 특성을 반영함으로써 모델의 설명력을 높일 수 있을 것입니다.

이처럼 중고차 가격 예측이라는 회귀 문제를 해결하며 다양한 회귀 모델의 성능을 비교하고 최적화하는 과정에서 여러 중요한 인사이트를 얻는 기회가 되었습니다. 다양한 성능 평가 지표와 잔차 분석을 통해 모델의 예측 성능을 종합적으로 검토하며, 모델이 데이터의 패턴을 잘 학습하고 있음을 확인할 수 있었습니다. 특히 Gradient Boosting 모델이 최종 모델로 선정된 것은 이 모델이 중고차 가격의 복잡한 관계를 잘 학습하고 예측 성능을 안정적으로 유지했기 때문이라고 생각합니다.

향후에는 추가적인 특성 생성과 다양한 모델 탐구를 통해 중고차 가격 예측의 정확도와 신뢰성을 더욱 높일 계획입니다. 이번 프로젝트에서 얻은 경험을 바탕으로, 중고차 가격 예측에서 다루는 다양한 특성의 중요성을 더 깊이 이해하고, 모델의 일반화 성능을 개선할 방안을 모색하고자 합니다. 특히 이번에 사용한 Gradient Boosting 모델의 성능을 더욱 향상하기 위해, 추가적인 하이퍼파라미터 튜닝과 조기 종료(Early Stopping)와 같은 기법을 활용하여 과적합을 방지할 수 있다고 생각합니다. 또한 회귀 부분을 진행하면서 SHAP(SHapley Additive exPlanations)나 LIME(Local Interpretable Model-agnostic Explanations)과 같은 모델 해석 기법을 알게 되었는데 이후 학습을 통해 사용하여 각 특성이 예측 결과에 미치는 영향을 더욱 명확히 파악함으로써 모델 해석력을 높이고 결과의 신뢰성을 더 향상시키고 싶습니다.

나아가, 시계열 분석이나 이벤트 기반 데이터와 같은 추가적인 데이터 소스를 활용하여, 차량의 사용 이력이나 시장 동향과 같은 동적 요소를 반영하는 방법 또한 검토할 계획입니다. 이를 진행하면 현재의 정적 특성 기반 예측에서 벗어나, 시계열적 요소가 포함된 보다 종합적인 예측 모델을 구축할 수 있을 것으로 기대됩니다.

결국 이번 프로젝트는 저에게 회귀 분석에서 특성 선택, 데이터 전처리, 모델 최적화 등 다양한 기법을 실험하면서 여러 가지 도전과 학습을 경험하는 기회가 되었습니다. 이를 통해 단순히 성능이 높은 모델을 찾는 것뿐 아니라, 각 모델이 데이터의 특성을 어떻게 반영하고, 예측 결과에 어떤 영향을 미치는지를 심도 있게 이해할 수 있었습니다. 이러한 통찰은 향후 컴

퓨터 비전 프로젝트나 개인 프로젝트에 있어서도 매우 유용할 것이며 실무 환경에서도 더 신뢰할 수 있는 예측 모델을 구축하는 데 중요한 역할을 할 것입니다. 결론적으로, 이번 프로젝트는 회귀 문제를 해결하는 데 있어 데이터 분석과 모델링의 모든 단계를 깊이 이해하고 적용할 수 있는 귀중한 경험이었으며, 앞으로도 이 경험을 바탕으로 회귀 모델의 활용성과 신뢰성을 높이는 방향으로 연구를 확장해 나가겠습니다.