1 빅데이터 기반 AI 응용 솔루션 개발자 전문 과정

1.0.1 교과목명 : 머신러닝알고리즘 이해 및 활용

- 평가일 : 10.14
- 성명 : 양 주희
- 점수 : 70

Q1. iris data를 불러와서 아래 사항을 수행하세요.

- 결정트리 모델을 시각화하고 주요한 인사이트를 기술하세요.(tree.plot_tree or tree.export_graphviz 이용)
- Feature importance를 추출하고 시각화하세요.

In [1]:
```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users
\user\appdata\local\programs\python\python39\lib\si
te-packages (3.3.3)Note: you may need to restart th
e kernel to use updated packages.

Requirement already satisfied: wheel in c:\users\us
er\appdata\local\programs\python\python39\lib\site-
packages (from lightgbm) (0.37.1)
Requirement already satisfied: scipy in c:\users\us
er\appdata\local\programs\python\python39\lib\site-
packages (from lightgbm) (1.8.0)
Requirement already satisfied: scikit-learn!=0.22.0
in c:\users\user\appdata\local\programs\python\pyth
on39\lib\site-packages (from lightgbm) (1.1.2)
Requirement already satisfied: numpy in c:\users\us
er\appdata\local\programs\python\python39\lib\site-
packages (from lightgbm) (1.22.3)
Requirement already satisfied: joblib>=1.0.0 in
c:\users\user\appdata\local\programs\python\python3
9\lib\site-packages (from scikit-learn!=0.22.0->lig
htgbm) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0
in c:\users\user\appdata\local\programs\python\pyth
on39\lib\site-packages (from scikit-learn!=0.22.0->
lightgbm) (3.1.0)
```

In [2]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings('ignore')

#DecisionTreeClassifier 생성
dt_clf = DecisionTreeClassifier(random_state=156)


iris_data = load_iris()

X_train,X_test, y_train, y_test = train_test_split(iris_data.data,

#학습
dt_clf.fit(X_train,y_train)
```

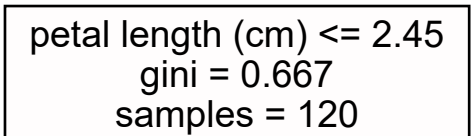
```
DecisionTreeClassifier(random_st
ate=156)
```

In [3]:

```
from sklearn.tree import export_graphviz

#export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을
export_graphviz(dt_clf, out_file='tree_test.dot', class_names=iris
```

In [4]:

```
import graphviz

#위에서 생성한 tree.dot 파일을 graphviz가 읽어서 주피터 노트북상에
with open('tree_test.dot') as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

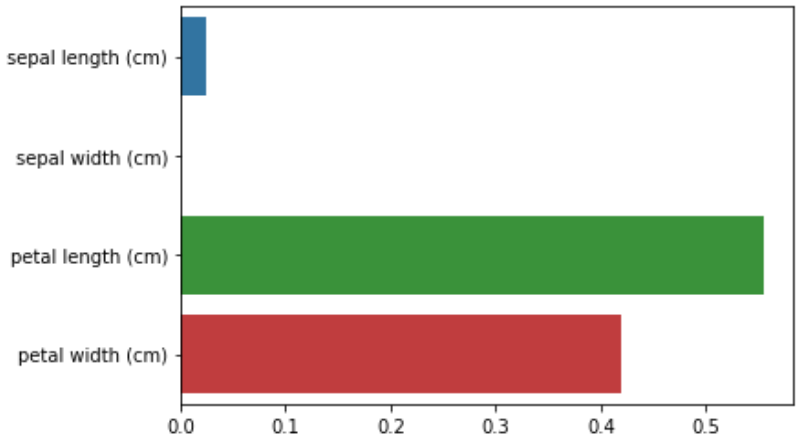


In [5]:

```
import seaborn as sns
import numpy as np

print(f'Feature Importances: ₩n {np.round(dt_clf.feature_importanc
for name, value in zip(iris_data.feature_names, dt_clf.feature_imp
    print(f'{name} : {value:.3f}')
print()
sns.barplot(x=dt_clf.feature_importances_, y=iris_data.feature_nam
```

```
Feature Importances:
 [0.025 0.    0.555 0.42 ]

sepal length (cm) : 0.025
sepal width (cm) : 0.000
petal length (cm) : 0.555
petal width (cm) : 0.420
```

```
<AxesSubplot:>
```



Q2~Q3. 'dataset/creditcard.csv'를 불러와서 신용카드 사기 검출 분류문제를 아래와 같이 수행하세요

- 로지스틱 리그레션을 적용한 모델 학습 및 사용자 함수를 이용하여 평가
  - 인자로 입력받은 DataFrame을 복사한 뒤 Time 칼럼만 삭제하고 복사된 df 반환하는 사용자 함수 생성
  - 사전 데이터 가공 후 학습과 테스트 데이터 세트를 반환하는 함수(테스트 사이즈 0.3)
  - 오차행렬, 정확도, 정밀도, 재현율, f1, AUC 평가 함수

- 인자로 사이킷런의 Estimator 객체와 학습/테스트 데이터 세트를 입력 받아서 학습/예측/평가 수행
  - 사용자 함수를 사용하여 LightGBM으로 모델을 학습한 뒤 별도의 테스트 데이터 세트에서 예측 평가를 수행. 단, n_estimators=1000, num_leaves=64 적용
    ※ 레이블 값이 극도로 불균형한 분포를 가지고 있는 경우 boost_from_average=False로 파라미터 설정(default=True). default 설정은 재현율, AUC 성능을 매우 크게 저하시킴
  - 넘파이의 np.log1p( )를 이용하여 Amount를 로그 변환하는 사용자 함수 생성
  - Amount를 로그 변환 후 로지스틱 회귀 및 LightGBM 수행.

In [6]:

```
import pandas as pd
card_df = pd.read_csv('creditcard.csv')
card_df.head()
```

	Time	V1	V2	V3	V4	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.3
1	0.0	1.191857	0.266151	0.166480	0.448154	0.0(
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.5
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.4

5 rows × 31 columns

1.0.2 Q2.

In [7]:

```
# 인자로 입력받은 DataFrame을 복사한 뒤 Time 칼럼만 삭제하고 복사된

from sklearn.model_selection import train_test_split
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    df_copy.drop('Time',axis=1,inplace=True)
    return df_copy
```

In [8]:

```
# 사전 데이터 가공 후 학습과 테스트 데이터 세트를 반환하는 함수(테

def get_train_test_dataset(df=None):
    df_copy = get_preprocessed_df(df)
    #맨 끝에 레이블 값 빼고 -> features
    X_features = df_copy.iloc[:,:-1]
    y_target = df_copy.iloc[:,-1]
    X_train,X_test,y_train,y_test = train_test_split(X_features,y_
    return X_train,X_test,y_train,y_test

X_train,X_test,y_train,y_test = get_train_test_dataset(card_df)
```

In [9]:

```
# 오차행렬, 정확도, 정밀도, 재현율, f1, AUC 평가 함수
from sklearn.metrics import confusion_matrix, accuracy_score, prec
from sklearn.metrics import roc_auc_score

def get_clf_eval(y_test, pred, pred_proba):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},W
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall,
```

1.0.3 Q3

In [10]:

```
# 인자로 사이킷런의 Estimator 객체와 학습/테스트 데이터 세트를 입
def get_model_train_eval(model, ftr_train=None,ftr_test=None,tgt_t
    model.fit(ftr_train,tgt_train)
    pred = model.predict(ftr_test)
    pred_proba = model.predict_proba(ftr_test)[:,1]
    get_clf_eval(tgt_test,pred,pred_proba)
```

In [11]:

```
# 사용자 함수를 사용하여 LightGBM으로 모델을 학습한 뒤 별도의 테스
# ※ 레이블 값이 극도로 불균형한 분포를 가지고 있는 경우 boost_fro

from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=1000,num_leaves=64,n_jobs=-
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test,
```

```
오차 행렬
[[85290     5]
 [   36   112]]
정확도: 0.9995, 정밀도: 0.9573, 재현율: 0.7568,
F1: 0.8453, AUC:0.9790
```

In [12]:

```
# 넘파이의 np.log1p( )를 이용하여 Amount를 로그 변환하는 사용자 함
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    # 넘파이의 log1p( )를 이용하여 Amount를 로그 변환
    amount_n = np.log1p(df_copy['Amount'])
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time','Amount'], axis=1, inplace=True)
    return df_copy
```

In [13]:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
# Amount를 로그 변환 후 로지스틱 회귀 및 LightGBM 수행
X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)

print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, t

print('### LightGBM 예측 성능 ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test,
```

```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[85283    12]
 [   59    89]]
정확도: 0.9992, 정밀도: 0.8812, 재현율: 0.6014,
F1: 0.7149, AUC:0.9727
### LightGBM 예측 성능 ###
오차 행렬
[[85290     5]
 [   35   113]]
정확도: 0.9995, 정밀도: 0.9576, 재현율: 0.7635,
F1: 0.8496, AUC:0.9796
```

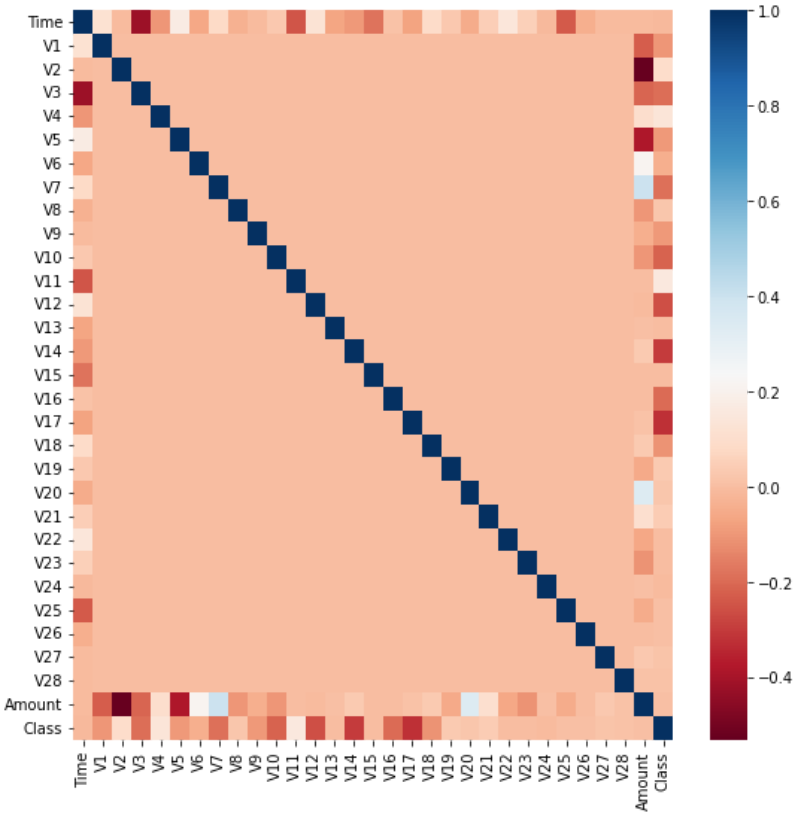
Q4. Q2 신용카드 사기 검출 분류문제에서 아래를 참고하여 이상치 데이터를 제거하고 모델 학습/예측/평가를 수행하세요

- 히트맵을 이용해 레이블과의 상관성을 시각화
- 레이블과 상관성이 높은 피처를 위주로 이상치 검출하는 사용자 함수 생성
- 사용자 함수를 이용하여 이상치 검출
- 이상치 제거 사용자 함수를 이용하여 이상치 제거 후 로지스틱 회귀 및 LightGBM 수행 및 평가

In [14]:

```
# 히트맵을 이용해 레이블과의 상관성을 시각화
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(9, 9))
corr = card_df.corr()
sns.heatmap(corr, cmap='RdBu')
```

```
<AxesSubplot:>
```



In [15]:

```
# 레이블과 상관성이 높은 피처를 위주로 이상치 검출하는 사용자 함수

def get_outlier(df=None, column=None, weight=1.5):
    # fraud에 해당하는 column 데이터만 추출, 1/4 분위와 3/4 분위 지
    fraud = df[df['Class']==1][column]
    quantile_25 = np.percentile(fraud.values, 25)
    quantile_75 = np.percentile(fraud.values, 75)
    # IQR을 구하고, IQR에 1.5를 곱하여 최대값과 최소값 지점 구함.
    iqr = quantile_75 - quantile_25
    iqr_weight = iqr * weight
    lowest_val = quantile_25 - iqr_weight
    highest_val = quantile_75 + iqr_weight
    # 최대값 보다 크거나, 최소값 보다 작은 값을 아웃라이어로 설정한
    outlier_index = fraud[(fraud < lowest_val) | (fraud > highest_
    return outlier_index
```

In [16]:

```
# 사용자 함수를 이용하여 이상치 검출
outlier_index = get_outlier(df=card_df, column='V14', weight=1.5)
print('이상치 데이터 인덱스:', outlier_index)
```

```
이상치 데이터 인덱스: Int64Index([8296, 8615, 9035,
9252], dtype='int64')
```

In [17]:

```
# 이상치 제거 사용자 함수를 이용하여 이상치 제거 후 로지스틱 회귀
# get_processed_df( )를 로그 변환 후 V14 피처의 이상치 데이터를 삭
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    amount_n = np.log1p(df_copy['Amount'])
    df_copy.insert(0, 'Amount_Scaled', amount_n)
    df_copy.drop(['Time','Amount'], axis=1, inplace=True)
    # 이상치 데이터 삭제하는 로직 추가
    outlier_index = get_outlier(df=df_copy, column='V14', weight=
    df_copy.drop(outlier_index, axis=0, inplace=True)
    return df_copy

X_train, X_test, y_train, y_test = get_train_test_dataset(card_df)
print('### 로지스틱 회귀 예측 성능 ###')
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, t
print('### LightGBM 예측 성능 ###')
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test,
```

```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[85281    14]
 [   48    98]]
정확도: 0.9993, 정밀도: 0.8750, 재현율: 0.6712,
F1: 0.7597, AUC:0.9743
### LightGBM 예측 성능 ###
오차 행렬
[[85290     5]
 [   25   121]]
정확도: 0.9996, 정밀도: 0.9603, 재현율: 0.8288,
F1: 0.8897, AUC:0.9780
```

Q5. SMOTE 오버 샘플링 적용 후 LightGBM 모델을 이용하여 학습, 예측, 평가를 수행하세요.

In [18]:

```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:Wu
sersWuserWappdataWlocalWprogramsWpythonWpython39Wli
bWsite-packages (1.1.2)
Requirement already satisfied: joblib>=1.0.0 in
c:WusersWuserWappdataWlocalWprogramsWpythonWpython3
9WlibWsite-packages (from scikit-learn) (1.1.0)
Requirement already satisfied: scipy>=1.3.2 in c:Wu
sersWuserWappdataWlocalWprogramsWpythonWpython39Wli
bWsite-packages (from scikit-learn) (1.8.0)
Requirement already satisfied: threadpoolctl>=2.0.0
in c:WusersWuserWappdataWlocalWprogramsWpythonWpyth
on39WlibWsite-packages (from scikit-learn) (3.1.0)
Requirement already satisfied: numpy>=1.17.3 in
c:WusersWuserWappdataWlocalWprogramsWpythonWpython3
9WlibWsite-packages (from scikit-learn) (1.22.3)
Note: you may need to restart the kernel to use upd
ated packages.
```

In [19]:

```
pip install -U imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
c:WusersWuserWappdataWlocalWprogramsWpythonWpython3
9WlibWsite-packages (0.9.1)Note: you may need to re
start the kernel to use updated packages.

Requirement already satisfied: scikit-learn>=1.1.0
in c:WusersWuserWappdataWlocalWprogramsWpythonWpyth
on39WlibWsite-packages (from imbalanced-learn) (1.
1.2)
Requirement already satisfied: joblib>=1.0.0 in
c:WusersWuserWappdataWlocalWprogramsWpythonWpython3
9WlibWsite-packages (from imbalanced-learn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0
in c:WusersWuserWappdataWlocalWprogramsWpythonWpyth
on39WlibWsite-packages (from imbalanced-learn) (3.
1.0)
Requirement already satisfied: numpy>=1.17.3 in
c:WusersWuserWappdataWlocalWprogramsWpythonWpython3
9WlibWsite-packages (from imbalanced-learn) (1.22.
3)
Requirement already satisfied: scipy>=1.3.2 in c:Wu
sersWuserWappdataWlocalWprogramsWpythonWpython39Wli
bWsite-packages (from imbalanced-learn) (1.8.0)
```

In [20]:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=0)
X_train_over, y_train_over = smote.fit_resample(X_train, y_train)
print('SMOTE 적용 전 학습용 피처/레이블 데이터 세트: ', X_train.sh
print('SMOTE 적용 후 학습용 피처/레이블 데이터 세트: ', X_train_ov
print('SMOTE 적용 후 레이블 값 분포: ₩n', pd.Series(y_train_over).
```

```
SMOTE 적용 전 학습용 피처/레이블 데이터 세트:  (199
362, 29) (199362,)
SMOTE 적용 후 학습용 피처/레이블 데이터 세트:  (398
040, 29) (398040,)
SMOTE 적용 후 레이블 값 분포:
 0    199020
1    199020
Name: Class, dtype: int64
```

In [21]:

```
lr_clf = LogisticRegression()
# ftr_train과 tgt_train 인자값이 SMOTE 증식된 X_train_over와 y_tra
get_model_train_eval(lr_clf, ftr_train=X_train_over, ftr_test=X_te
```

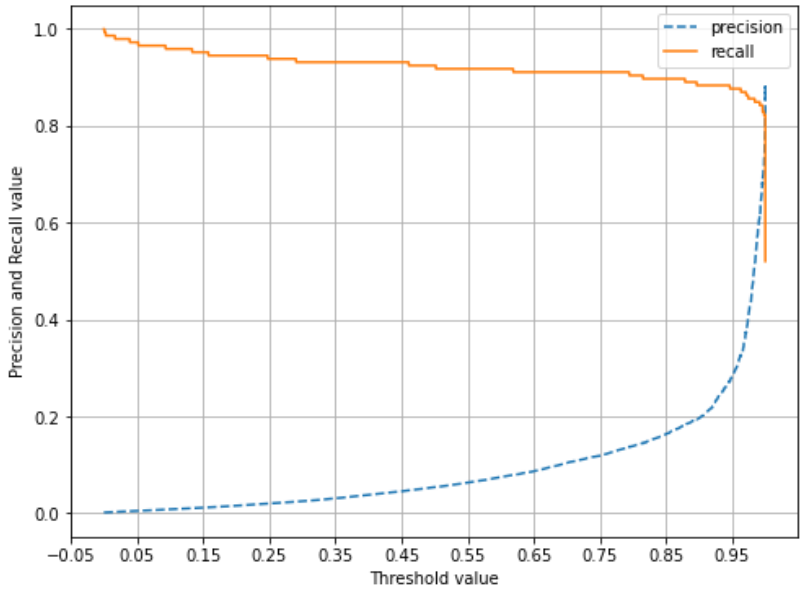
```
오차 행렬
[[82937  2358]
 [   11   135]]
정확도: 0.9723, 정밀도: 0.0542, 재현율: 0.9247,
F1: 0.1023, AUC:0.9737
```

In [22]:

```
from sklearn.metrics import precision_recall_curve
def precision_recall_curve_plot(y_test , pred_proba_c1):
    # threshold ndarray와 이 threshold에 따른 정밀도, 재현율 ndarr
    precisions, recalls, thresholds = precision_recall_curve( y_te

    # X축을 threshold값으로, Y축은 정밀도, 재현율 값으로 각각 Plot
    plt.figure(figsize=(8,6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linesty
    plt.plot(thresholds, recalls[0:threshold_boundary],label='reca

    # threshold 값 X 축의 Scale을 0.1 단위로 변경
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))

    # x축, y축 label과 legend, 그리고 grid 설정
    plt.xlabel('Threshold value'); plt.ylabel('Precision and Recal
    plt.legend(); plt.grid()
    plt.show()
```

In [23]:

```
precision_recall_curve_plot( y_test, lr_clf.predict_proba(X_test)[
```



In [24]:

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs
get_model_train_eval(lgbm_clf, ftr_train=X_train_over, ftr_test=X_
                     tgt_train=y_train_over, tgt_test=y_test)
```

```
오차 행렬
[[85283    12]
 [   22   124]]
정확도: 0.9996, 정밀도: 0.9118, 재현율: 0.8493,
F1: 0.8794, AUC:0.9814
```

Q6. 사이킷런에서 제공해주는 load_boston 데이터셋을 가져와서 아래 사항을 수행하세요.

- 데이터셋의 타겟 이름을 'PRICE'로 지정한 후 데이터프레임을 생성 pickle 파일로 저장 후 다시 불어오세요.
- 히트맵을 이용하여 타겟과 상관관계가 높은 독립 변수를 선택하세요.
- 종속변수를 로그 변환하세요
- 위의 사항을 반영하여 선회회귀 모델을 생성 후 평가하고 회귀계수를 출력하세요.

In [25]:

```
from sklearn.datasets import load_boston
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestRegressor
import numpy as np
import pandas as pd

boston = load_boston()
bostonDF = pd.DataFrame(boston.data, columns=boston.feature_names)

bostonDF['PRICE'] = boston.target
bostonDF.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.

In [26]:

```
import pickle

with open( "boston", "wb" ) as file:
    pickle.dump( bostonDF, file)
```

In [27]:

```
with open( "boston", "rb" ) as file:
    loaded_data = pickle.load(file)
```

In [28]:

```
loaded_data
```

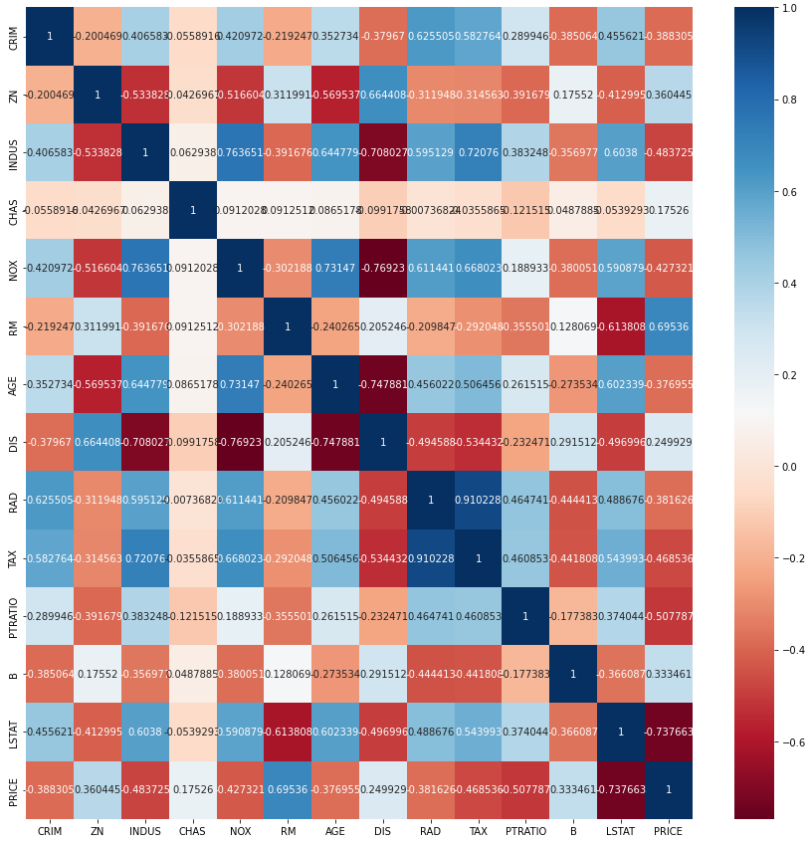
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2
...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8

506 rows × 14 columns

In [32]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 15))
corr = loaded_data.corr()
sns.heatmap(corr, cmap='RdBu', annot=True, fmt='g')
```

<AxesSubplot:>



In [33]:

```
corr
```

ZN	INDUS	CHAS	NOX	RM	AG
00469	0.406583	-0.055892	0.420972	-0.219247	0.352734
0000	-0.533828	-0.042697	-0.516604	0.311991	-0.56953
33828	1.000000	0.062938	0.763651	-0.391676	0.644779
42697	0.062938	1.000000	0.091203	0.091251	0.086518
16604	0.763651	0.091203	1.000000	-0.302188	0.731470
1991	-0.391676	0.091251	-0.302188	1.000000	-0.24026
69537	0.644779	0.086518	0.731470	-0.240265	1.000000
4408	-0.708027	-0.099176	-0.769230	0.205246	-0.74788
11948	0.595129	-0.007368	0.611441	-0.209847	0.456022
14563	0.720760	-0.035587	0.668023	-0.292048	0.506456
91679	0.383248	-0.121515	0.188933	-0.355501	0.261515
5520	-0.356977	0.048788	-0.380051	0.128069	-0.27353
12995	0.603800	-0.053929	0.590879	-0.613808	0.602339
0445	-0.483725	0.175260	-0.427321	0.695360	-0.37695

In [35]:

```
# 히트맵을 이용하여 타겟과 상관관계가 높은 독립 변수를 선택하세요.
# 종속변수를 로그 변환하세요
# 위의 사항을 반영하여 선회회귀 모델을 생성 후 평가하고 회귀계수를
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

X_feature = loaded_data.drop(['PRICE','CHAS','DIS'], axis=1, inpla
y_target = loaded_data['PRICE'].values.reshape(-1, 1)
y_target = np.log1p(y_target)

X_train, X_test, y_train, y_test = train_test_split(X_feature, y_t

lr = LinearRegression()

lr.fit(X_feature, y_target)
y_preds = lr.predict(X_test)

mse = mean_squared_error(y_test, y_preds)
rmse = np.sqrt(mse)

print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
print('Variance score : {0:.3f}'.format(r2_score(y_test, y_preds))
```

```
MSE : 0.036 , RMSE : 0.189
Variance score : 0.760
```

In [39]:

```
print('절편 값:',lr.intercept_)
print('회귀 계수값:', np.round(lr.coef_, 2))
```

```
절편 값: [3.65975286]
회귀 계수값: [[-0.01 -0.    0.01 -0.47  0.11  0.
0.01 -0.   -0.04  0.   -0.03]]
```

In [46]:

```
# 회귀 계수를 큰 값 순으로 정렬하기 위해 Series로 생성. index가 칼
coeff = pd.Series(data=np.round(lr.coef_, 2).reshape(-1), index=X_
coeff.sort_values(ascending=False)
```

```
RM          0.11
INDUS       0.01
RAD         0.01
ZN         -0.00
AGE         0.00
TAX        -0.00
B           0.00
CRIM       -0.01
LSTAT      -0.03
PTRATIO    -0.04
NOX        -0.47
dtype: float64
```

Q7. house_df.pkl 데이터셋을 불러와서 아래사항을 수행하세요.

- alphas = [0, 0.1, 1, 10, 100] 를 적용하여 Ridge 회귀 모델링 및 교차 검증 수행 후 5 폴드 평균 RMSE 출력
- lasso_alphas = [0.07,0.1,0.5,1,3] 를 적용, Lasso 회귀 모델링 및 교차 검증 수행 후 5 폴드 평균 RMSE 출력(def get_linear_reg_eval(model_name,params=None,X_data_n=None, y_target_n=None, verbose=True 사용자 함수 이용)
- elastic_alphas = [0.07,0.1,0.5,1,3] 를 적용, ElasticNet 회귀 모델링 및 교차검증 후 5 폴드 평균 RMSE를 출력(사용자 함수 이용)

In [49]:

```
house_df = loaded_data
house_df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2
...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8

506 rows × 14 columns

In [50]:

```
y_target = house_df['PRICE']
X_data = house_df.drop(['PRICE'],axis=1,inplace=False)
```

In [55]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso, ElasticNet

# alpha값에 따른 회귀 모델의 폴드 평균 RMSE를 출력하고 회귀 계수값
def get_linear_reg_eval(model_name, params=None, X_data_n=None, y_
                        verbose=True, return_coeff=True):
    coeff_df = pd.DataFrame()
    if verbose : print('####### ', model_name , '#######')
    for param in params:
        if model_name =='Ridge': model = Ridge(alpha=param)
        elif model_name =='Lasso': model = Lasso(alpha=param)
        elif model_name =='ElasticNet': model = ElasticNet(alpha=p
        neg_mse_scores = cross_val_score(model, X_data_n,
                                         y_target_n, scoring="
        avg_rmse = np.mean(np.sqrt(-1 * neg_mse_scores))
        print('alpha {0}일 때 5 폴드 세트의 평균 RMSE: {1:.3f} '.f
        # cross_val_score는 evaluation metric만 반환하므로 모델을

        model.fit(X_data_n , y_target_n)
        if return_coeff:
            # alpha에 따른 피처별 회귀 계수를 Series로 변환하고 이
            coeff = pd.Series(data=model.coef_ , index=X_data_n.co
            colname='alpha:'+str(param)
            coeff_df[colname] = coeff

    return coeff_df
```

In [56]:

```
ridge_alphas = [0 , 0.1 , 1 , 10 , 100]
coeff_ridge_df =get_linear_reg_eval('Ridge', params=ridge_alphas,
```

```
#######  Ridge #######
alpha 0일 때 5 폴드 세트의 평균 RMSE: 5.829
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.788
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.653
alpha 10일 때 5 폴드 세트의 평균 RMSE: 5.518
alpha 100일 때 5 폴드 세트의 평균 RMSE: 5.330
```

In [57]:

```
lasso_alphas = [ 0.07, 0.1, 0.5, 1, 3]
coeff_lasso_df =get_linear_reg_eval('Lasso', params=lasso_alphas,
```

```
#######  Lasso #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 5.612
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.615
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 5.669
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.776
alpha 3일 때 5 폴드 세트의 평균 RMSE: 6.189
```

In [58]:

```
elastic_alphas = [0.07,0.1,0.5,1,3]
coeff_elastic_df =get_linear_reg_eval('ElasticNet', params=elastic
```

```
#######  ElasticNet #######
alpha 0.07일 때 5 폴드 세트의 평균 RMSE: 5.542
alpha 0.1일 때 5 폴드 세트의 평균 RMSE: 5.526
alpha 0.5일 때 5 폴드 세트의 평균 RMSE: 5.467
alpha 1일 때 5 폴드 세트의 평균 RMSE: 5.597
alpha 3일 때 5 폴드 세트의 평균 RMSE: 6.068
```

Q8. load_boston 데이터셋을 불러와서 다음사항을 수행하세요.

- SVM 알고리즘을 활용한 주택가격 예측모델 생성 및 평가(MSE, RMSE, R2)
- 개발된 예측모델을 활용하여 아래 test_data가 주어졌은때의 주택가격 예측
  test_data = [3.7, 0, 18.4, 1, 0.87, 5.95, 91, 2.5052, 26, 666, 20.2, 351.34, 15.27]

In [ ]:

Q9. mtcars 데이터셋(mtcars.csv)의 qsec 컬럼을 최소최대 척도(Min-Max Scale)로 변환한 후 0.5보다 큰 값을 가지는 레코드 수를 구하시오

In [68]:

```
from sklearn.preprocessing import MinMaxScaler
mtcars_df = pd.read_csv('mtcars.csv',index_col=0)
mtcars_df
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0

	mpg	cyl	disp	hp	drat	wt	qsec	vs
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1

In [69]:

```
scaler = MinMaxScaler()

scaler.fit(mtcars_df)
mtcars_scaled = scaler.transform(mtcars_df)

mtcars_scaled_df = pd.DataFrame(data=mtcars_scaled, columns=mtcars
len(mtcars_scaled_df[mtcars_scaled_df.qsec > 0.5])

# mtcars_scaled_df
```

9

Q10. purdata.csv는 백화점 고객의 1년 간 구매 데이터이다. 아래사항을 수행하세요.

- 남성고객을 분류하는 모델을 생성(분류알고리즘 : dt,rf,lr)
- 모델 성능을 roc_auc로 평가

In [79]:

```
df = pd.read_csv('purdata.csv')
df.head()
```

	cust_id	총구매액	최대구매액	환불금액	주구매상품	주구매지점	내점일수
0	0	68282840	11264000	6860000.0	기타	강남점	19
1	1	*	2136000	300000.0	스포츠	잠실점	2
2	2	3197000	1639000	NaN	남성캐주얼	관악점	2
3	3	*	4935000	NaN	기타	광주점	18
4	4	29050000	24000000	NaN	보석	본점	2

In [95]:

```
# index 활용한 삭제
idx = df[df.총구매액 == '*'].index
df = df.drop(idx)
```

In [97]:

```
df['총구매액'] = df['총구매액'].astype(float)
```

In [99]:

```
df['환불금액'].fillna(0, inplace=True)
```

In [100]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3500 entries, 0 to 3499
Data columns (total 11 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   cust_id  3500 non-null   int64  
 1   총구매액     3500 non-null   float64
 2   최대구매액    3500 non-null   int64  
 3   환불금액     3500 non-null   float64
 4   주구매상품    3500 non-null   object 
 5   주구매지점    3500 non-null   object 
 6   내점일수     3500 non-null   int64  
 7   내점당구매건수  3500 non-null   float64
 8   주말방문비율   3500 non-null   float64
 9   구매주기     3500 non-null   int64  
 10  gender   3500 non-null   int64  
dtypes: float64(4), int64(5), object(2)
memory usage: 300.9+ KB
```

In [ ]:
```
df['주구매상품']
```

In [101]:
```
X_feature = df.drop(['gender'], axis=1, inplace=False)
y_target = df['gender']
```

In [102]:
```
X_train,X_test, y_train, y_test = train_test_split(X_feature, y_ta
```

In [103]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
lr = LinearRegression()
```

In [ ]:

```
dt_clf = DecisionTreeClassifier()
dt.fit(X_train , y_train)
pred = dt.predict(X_test)
accuracy = accuracy_score(y_test , pred)
print('결정 트리 예측 정확도: {0:.4f}'.format(accuracy))

# DecisionTreeClassifier의 하이퍼 파라미터 추출
print('DecisionTreeClassifier 기본 하이퍼 파라미터:₩n', dt.get_par
```

In [ ]: