음악 추천 시스템(spotify)

Spotify_music_recommender

2018108263

Introduction

1. 노래의 여러가지 요소를 분석
2. 분석한 요소들의 상관 관계를 분석하고 사용자가 좋아요를 누른 노래를 기준으로 추천
3. 여러가지 머신러닝 알고리즘을 이용하여 점수를 확인 후 가장 적합한 것을 이용

Introduction

1. 사용자가 좋아요를 누른 노래를 바탕으로 추천
2. 사용자가 좋아할 만한 노래를 들을 수 있음

사용한 라이브러리

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

데이터를 읽어들이고 유지하고 관리할 수 있는 멋진 모듈

다양한 수치연산, 변환 기능 등을 갖는 멋진 모듈

데이터를 멋지게 표시하는 모듈

사용한 라이브러리

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score , roc_auc_score , confusion_matrix
```

```
from sklearn.tree import DecisionTreeClassifier , plot_tree
```

```
from sklearn.ensemble import RandomForestClassifier , ExtraTreesClassifier , BaggingClassifier , AdaBoostClassifier
```

머신러닝 모델을 만들 수 있는 멋진 모듈

Input Dataset

```
spotify_df = pd.read_csv('../input/spotify-recommendation/data.csv')
spotify_df.head()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	liked
0	0.803	0.6240	7	-6.764	0	0.0477	0.451	0.000734	0.1000	0.6280	95.968	304524	4	0
1	0.762	0.7030	10	-7.951	0	0.3060	0.206	0.000000	0.0912	0.5190	151.329	247178	4	1
2	0.261	0.0149	1	-27.528	1	0.0419	0.992	0.897000	0.1020	0.0382	75.296	286987	4	0
3	0.722	0.7360	3	-6.994	0	0.0585	0.431	0.000001	0.1230	0.5820	89.860	208920	4	1
4	0.787	0.5720	1	-7.516	1	0.2220	0.145	0.000000	0.0753	0.6470	155.117	179413	4	1

```
spotify_df.tail()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	liked
190	0.166	0.0551	9	-19.494	0	0.0520	0.9760	0.635000	0.1190	0.143	176.616	206520	3	0
191	0.862	0.6240	3	-11.630	1	0.0565	0.0192	0.000153	0.0465	0.882	124.896	254240	4	0
192	0.499	0.3510	9	-11.509	0	0.0448	0.9510	0.000099	0.1180	0.616	90.664	235947	4	0
193	0.574	0.7290	10	-5.838	0	0.0965	0.0406	0.000004	0.1940	0.413	110.547	190239	5	1
194	0.747	0.6660	11	-7.845	1	0.1970	0.1300	0.000000	0.3600	0.531	77.507	177213	4	1

Input Dataset

```
spotify_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   danceability      195 non-null    float64
 1   energy            195 non-null    float64
 2   key               195 non-null    int64  
 3   loudness          195 non-null    float64
 4   mode              195 non-null    int64  
 5   speechiness       195 non-null    float64
 6   acousticness      195 non-null    float64
 7   instrumentalness  195 non-null    float64
 8   liveness          195 non-null    float64
 9   valence           195 non-null    float64
 10  tempo             195 non-null    float64
 11  duration_ms       195 non-null    int64  
 12  time_signature    195 non-null    int64  
 13  liked             195 non-null    int64  
dtypes: float64(9), int64(5)
memory usage: 21.5 KB
```

데이터 전처리

첫번째

Like 항목을 지우고 like열만 포함한 객체를 만든다.

두번째

shape를 이용해서 행과 열을 숫자로 보여줌

세번째

데이터의 종류를 10종류로 한정

데이터 전처리

```
X = spotify_df.drop('liked' , axis = 1)
y = spotify_df['liked']
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
0	0.803	(0.599, 0.698]	7	-6.764	0	0.0477	0.4510	0.000734	0.1000	0.6280	95.968	304524	4
1	0.762	(0.698, 0.797]	10	-7.951	0	0.3060	0.2060	0.000000	0.0912	0.5190	151.329	247178	4
2	0.261	(0.00141, 0.102]	1	-27.528	1	0.0419	0.9920	0.897000	0.1020	0.0382	75.296	286987	4
3	0.722	(0.698, 0.797]	3	-6.994	0	0.0585	0.4310	0.000001	0.1230	0.5820	89.860	208920	4
4	0.787	(0.499, 0.599]	1	-7.516	1	0.2220	0.1450	0.000000	0.0753	0.6470	155.117	179413	4
5	0.778	(0.599, 0.698]	8	-6.415	1	0.1250	0.0404	0.000000	0.0912	0.8270	140.951	224029	4
6	0.666	(0.499, 0.599]	0	-8.405	0	0.3240	0.5550	0.000000	0.1140	0.7760	74.974	146053	4
7	0.922	(0.698, 0.797]	7	-6.024	1	0.1710	0.0779	0.000040	0.1750	0.9040	104.964	161800	4
8	0.794	(0.599, 0.698]	7	-7.063	0	0.0498	0.1430	0.002240	0.0944	0.3080	112.019	247460	4
9	0.853	(0.599, 0.698]	3	-6.995	1	0.4470	0.2630	0.000000	0.1040	0.7450	157.995	165363	4

```
0    0
1    1
2    0
3    1
4    1
5    1
6    1
7    1
8    0
9    1
Name: liked, dtype: int64
```

데이터 전처리

shape

데이터의 행과 열의 갯수를 나타내줌

```
spotify_df.tail()
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	liked
190	0.166	0.0551	9	-19.494	0	0.0520	0.9760	0.635000	0.1190	0.143	176.616	206520	3	0
191	0.862	0.6240	3	-11.630	1	0.0565	0.0192	0.000153	0.0465	0.882	124.896	254240	4	0
192	0.499	0.3510	9	-11.509	0	0.0448	0.9510	0.000099	0.1180	0.616	90.664	235947	4	0
193	0.574	0.7290	10	-5.838	0	0.0965	0.0406	0.000004	0.1940	0.413	110.547	190239	5	1
194	0.747	0.6660	11	-7.845	1	0.1970	0.1300	0.000000	0.3600	0.531	77.507	177213	4	1

```
spotify_df.shape
```

```
(195, 14)
```

데이터 전처리

train test split

싸이킷런 안에 있는 모듈로 학습 데이터와 테스트 데이터를 쉽게 나누어 주는 모듈

```
from sklearn.model_selection import train_test_split
```

```
X_train , X_test , y_train , y_test = train_test_split(X , y , test_size=0.3)
print("the shapes of x-train and x-test are : " , X_train.shape , X_test.shape)
print("the shapes of y-train and y-test are : " , y_train.shape , y_test.shape)
```

```
the shapes of x-train and x-test are :  (136, 13) (59, 13)
the shapes of y-train and y-test are :  (136,) (59,)
```

데이터 시각화

첫번째

heatmap을 이용한 상관 관계 분석

두번째

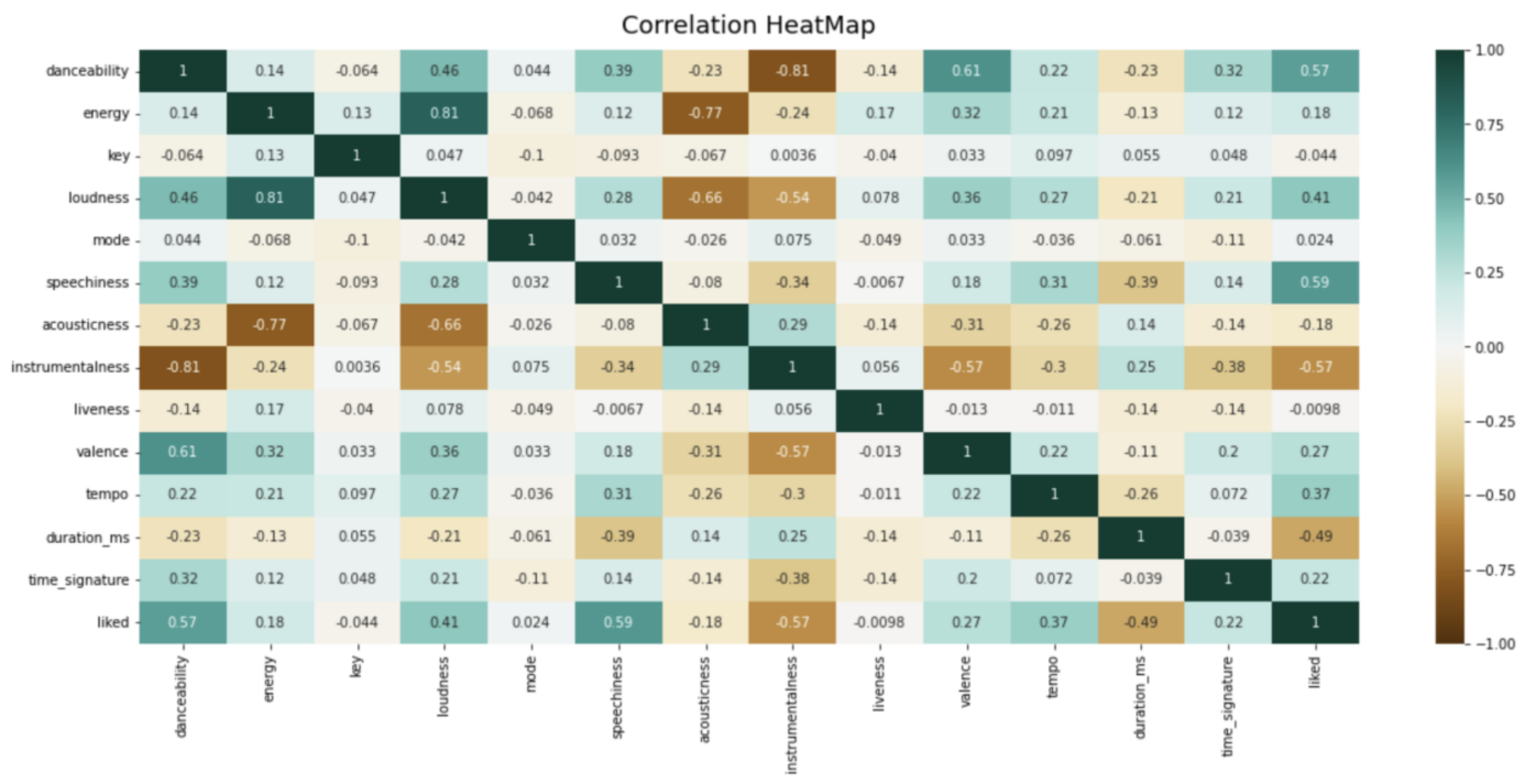
violinplot 을 이용한 데이터 시각화

세번째

그 외의 다양한 방법을 데이터를 시각화

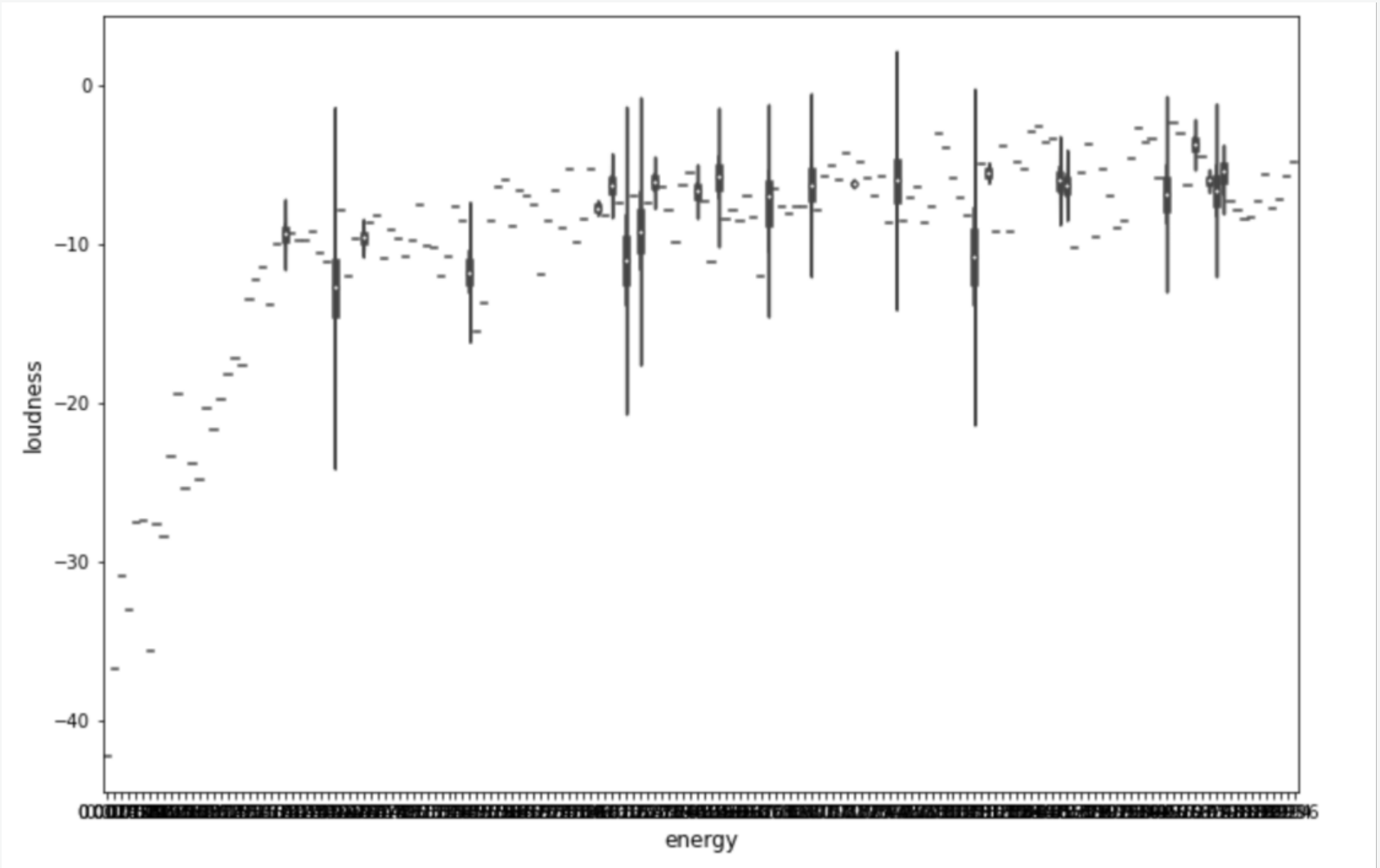
데이터 시각화

```
plt.figure(figsize=(20,8))
heatmap = sns.heatmap(spotify_df.corr() , vmin=-1 , vmax=1 ,annot=True , cmap='BrBG')
heatmap.set_title('Correlation HeatMap' , fontdict = {'fontsize' : 18} , pad=12)
```



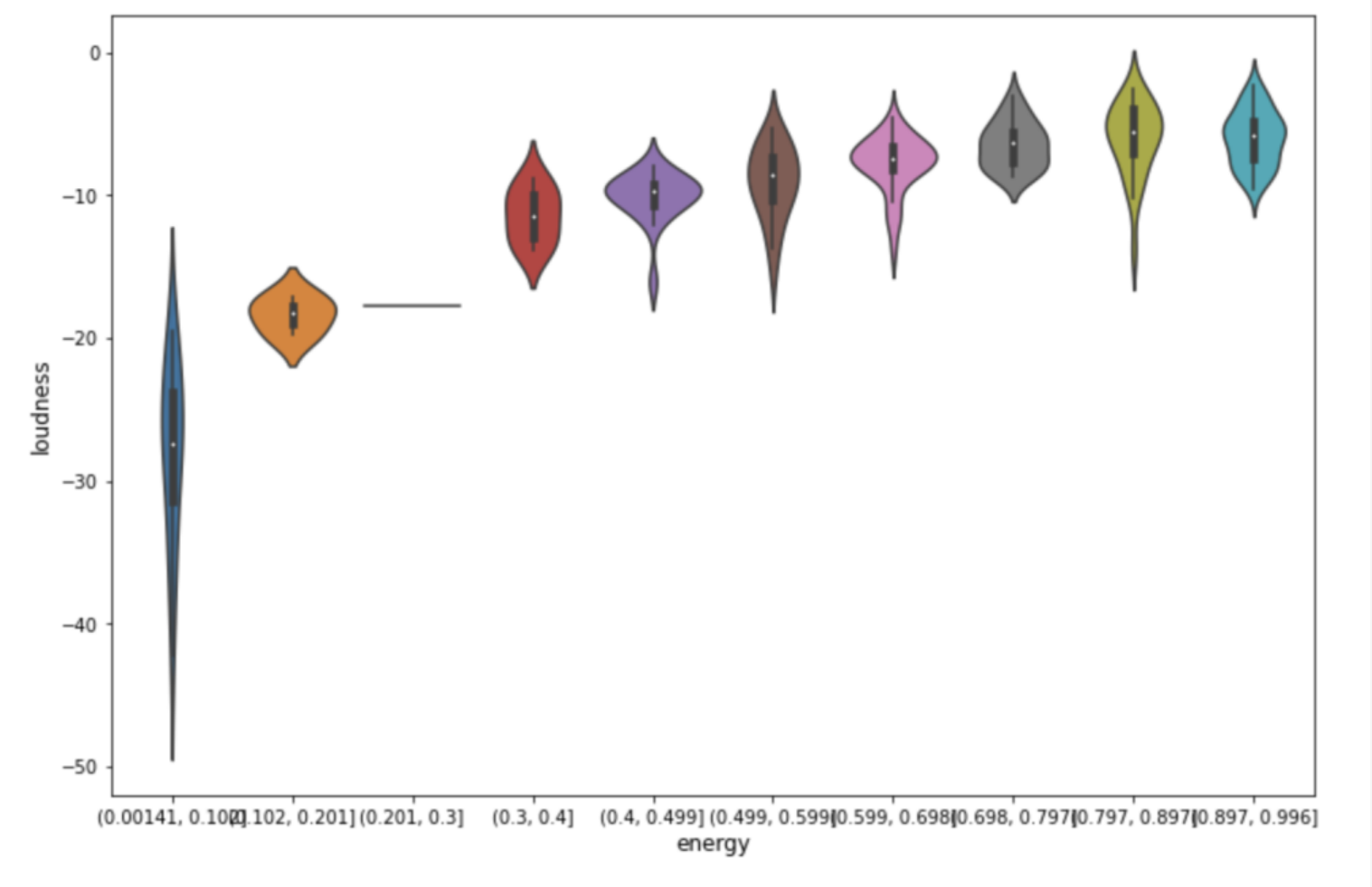
데이터 시각화

```
plt.figure(figsize=(12,8))
sns.violinplot(x='energy', y='loudness', data=spotify_df)
plt.xlabel('energy', fontsize=12)
plt.ylabel('loudness', fontsize=12)
plt.show()
```



데이터 시각화

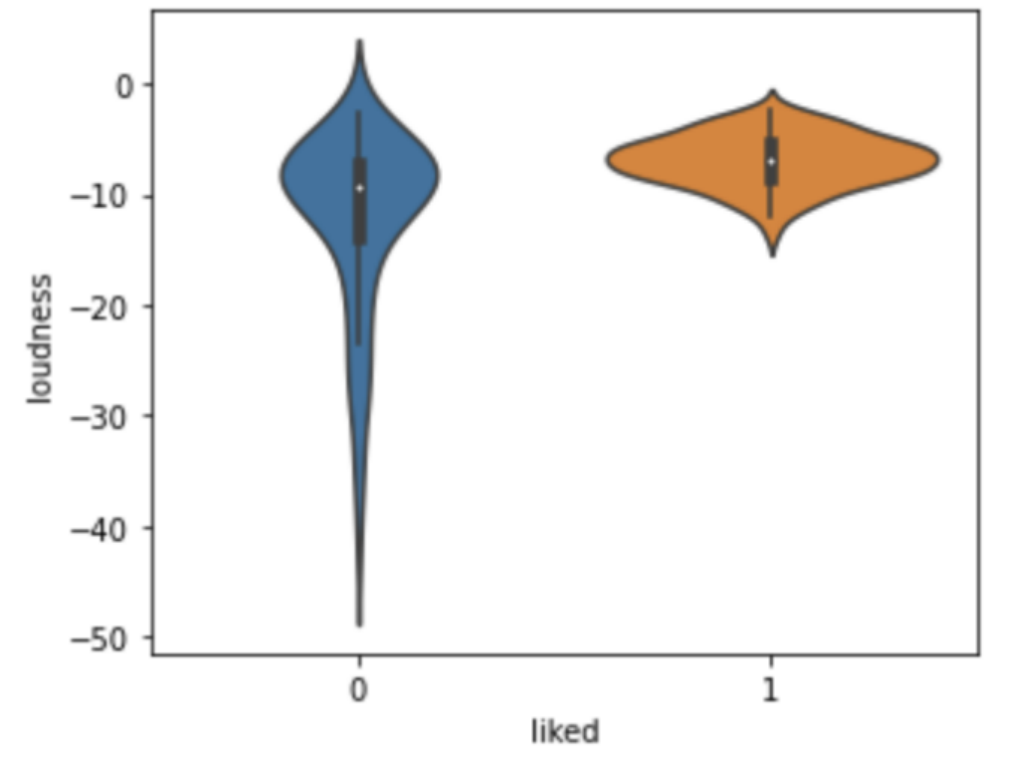
```
san['energy'] = pd.cut(spotify_df['energy'], bins=10)

plt.figure(figsize=(12,8))
sns.violinplot(x='energy', y='loudness', data=san)
plt.xlabel('energy', fontsize=12)
plt.ylabel('loudness', fontsize=12)
plt.show()
```



데이터 시각화

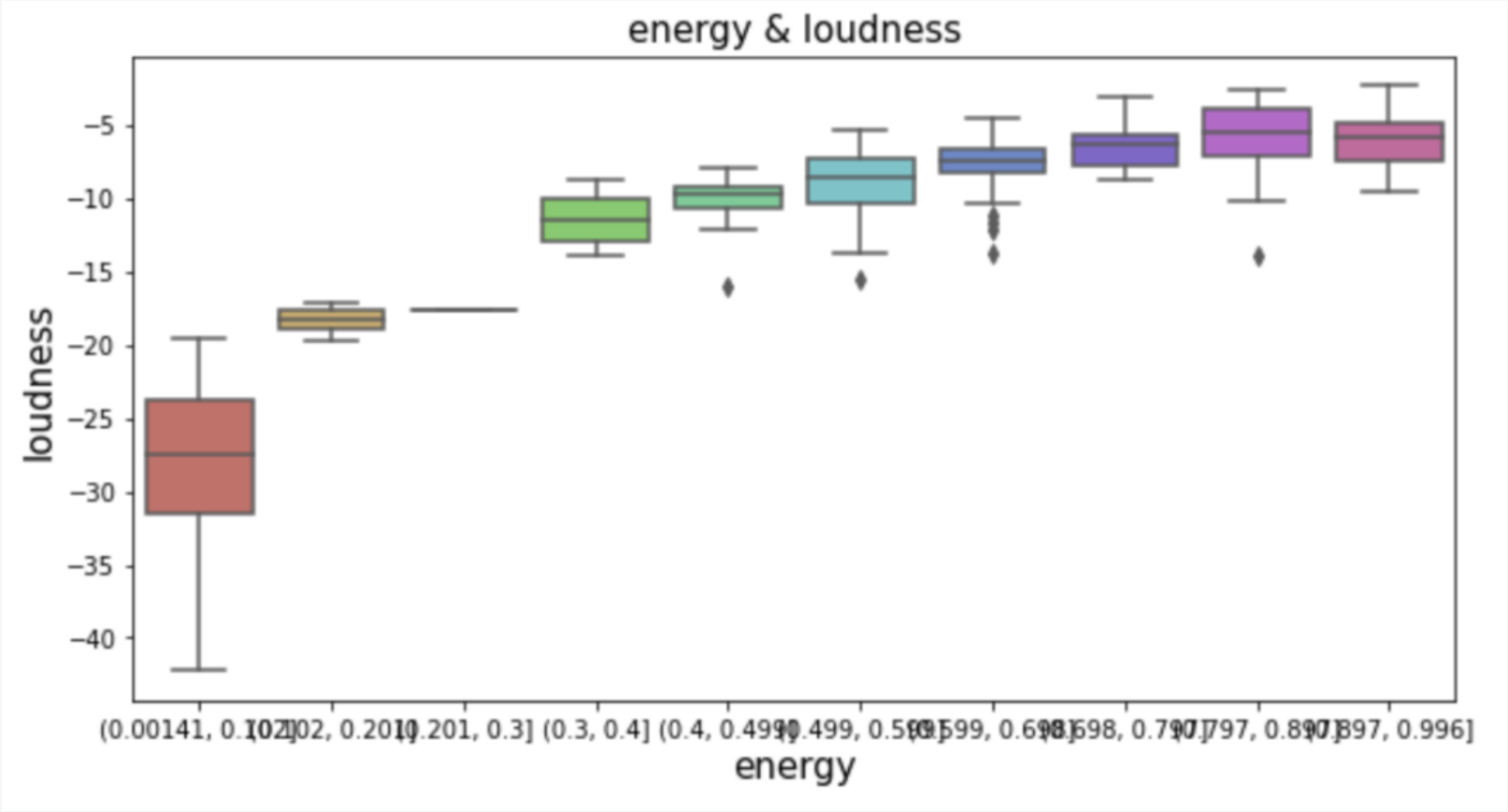
```
def violinplot(df, a, b):
    plt.figure(figsize=(5,4))
    plt.subplot(1,1,1)
    sns.violinplot(x=a,y=b,data=df)
```

```
violinplot(spotify_df, 'liked', 'loudness')
```



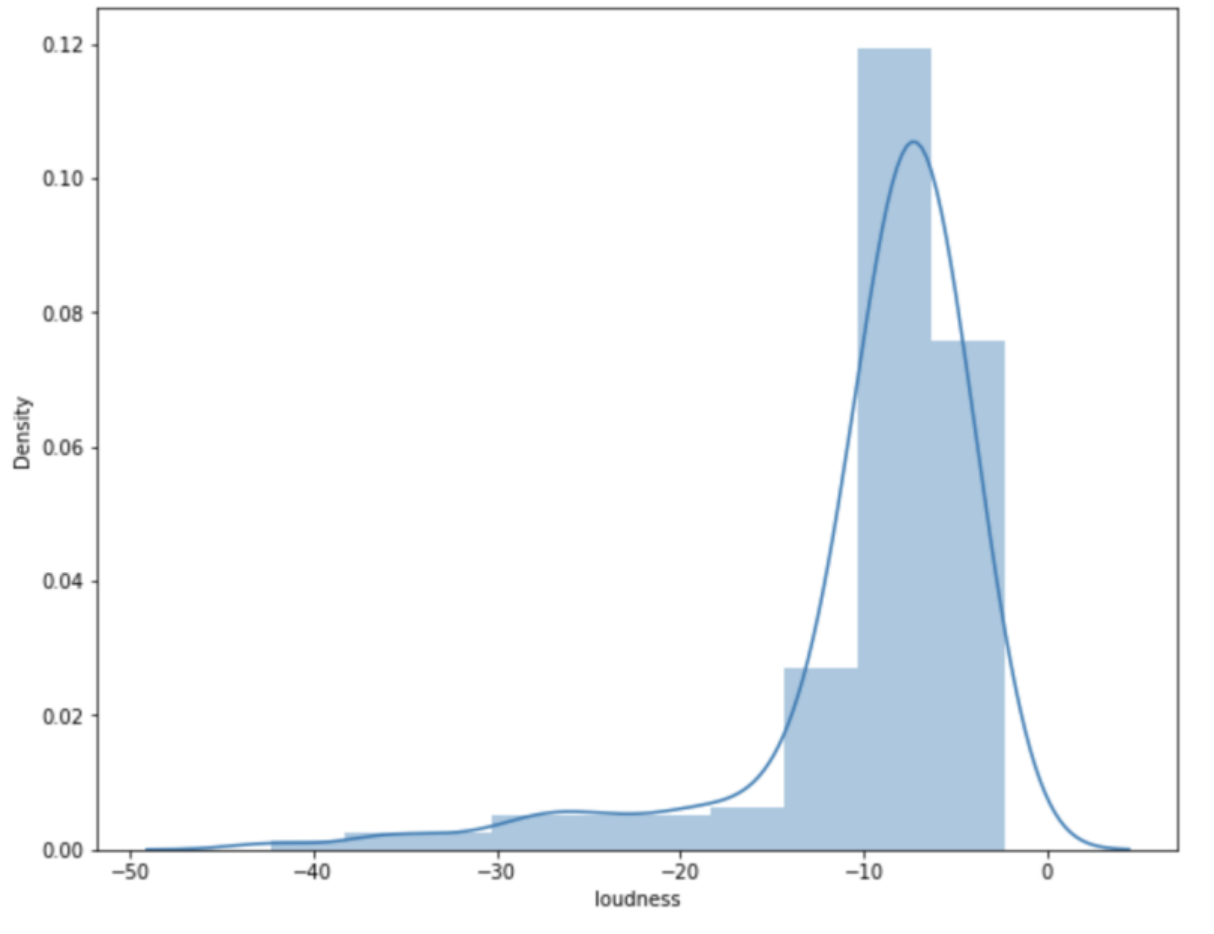
데이터 시각화

```
plt.figure(figsize=(10,5))
sns.boxplot(x='energy',y='loudness',data=san,palette='hls')
plt.title('energy & loudness',fontsize=15)
plt.xlabel('energy',fontsize=15)
plt.ylabel('loudness',fontsize=15)
plt.show()
```



데이터 시각화

```
f, ax = plt.subplots(figsize=(10,8))
x = spotify_df['loudness']
ax = sns.distplot(x, bins=10)
plt.show()
```



데이터 학습

첫번째

DesisionTree를 이용한 학습

두번째

머신러닝 모델과 모델별 성능 비교

세번째

LGBM 방식에 대한 설명과 성능 비교

데이터 학습

```
from sklearn.tree import DecisionTreeClassifier , plot_tree
dt_clf = DecisionTreeClassifier()
dt_clf.fit(X_train , y_train)
y_preds_dt = dt_clf.predict(X_test)
```

```
print("Accuracy Score of the Decision Tree Model" , accuracy_score(y_test , y_preds_dt))
print("ROC AUC score of the Decision Tree Model is : " , roc_auc_score(y_test , y_preds_dt))
```

```
Accuracy Score of the Decision Tree Model 0.8135593220338984
ROC AUC score of the Decision Tree Model is :  0.8136574074074076
```

데이터 학습

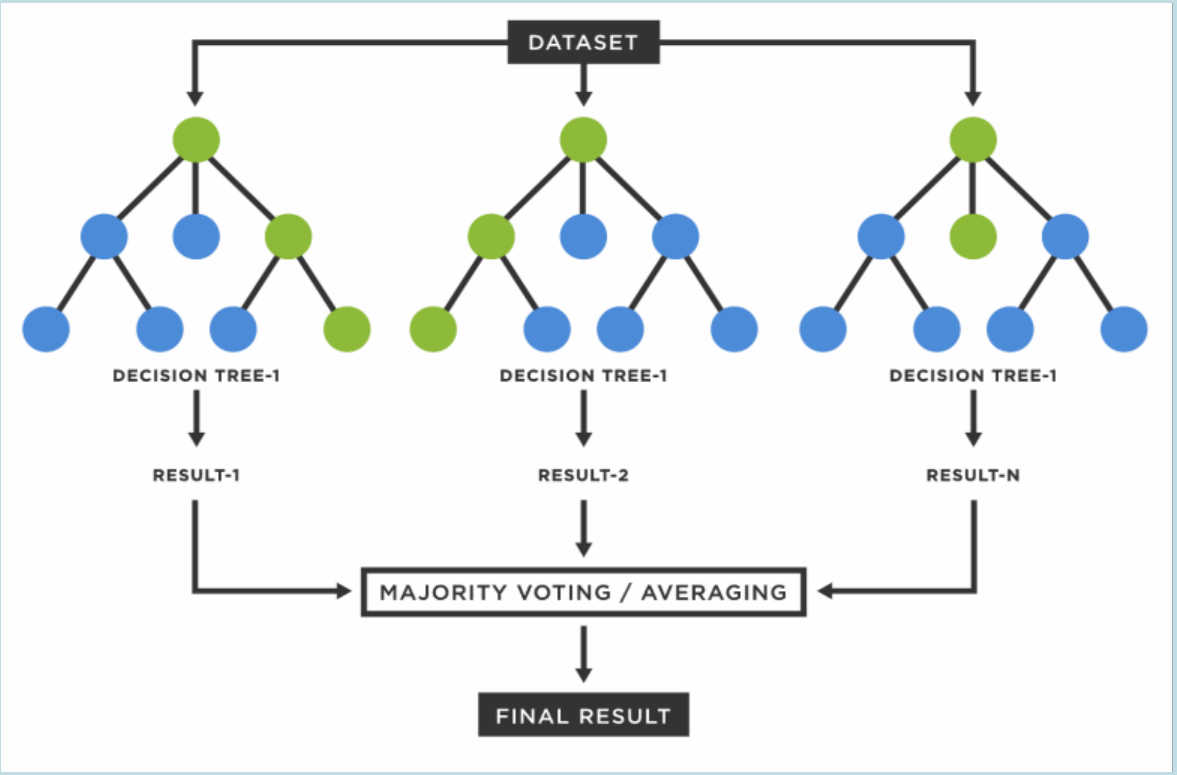

RandomForest Classifier
ExtraTrees Classifier
머신러닝 모델
Bagging Classifier
AdaBoost Classifier

데이터 학습

RandomForest Classifier

정의

-랜덤 포레스트는 다수의 결정 트리들을 학습하는 앙상블 방법이다. 랜덤 포레스트는 검출, 분류, 그리고 회귀 등 다양한 문제에 활용되고 있다.



데이터 학습

ExtraTrees Classifier

정의
- 엑스트라 트리는 랜덤 포레스트와 매우 비슷하게 동작하는데, 기본적으로 100개의 결정 트리를 훈련시키며, 전체 특정 중에 일부 특성을 랜덤하게 선택하여 노드를 분할하기 위해 사용한다.

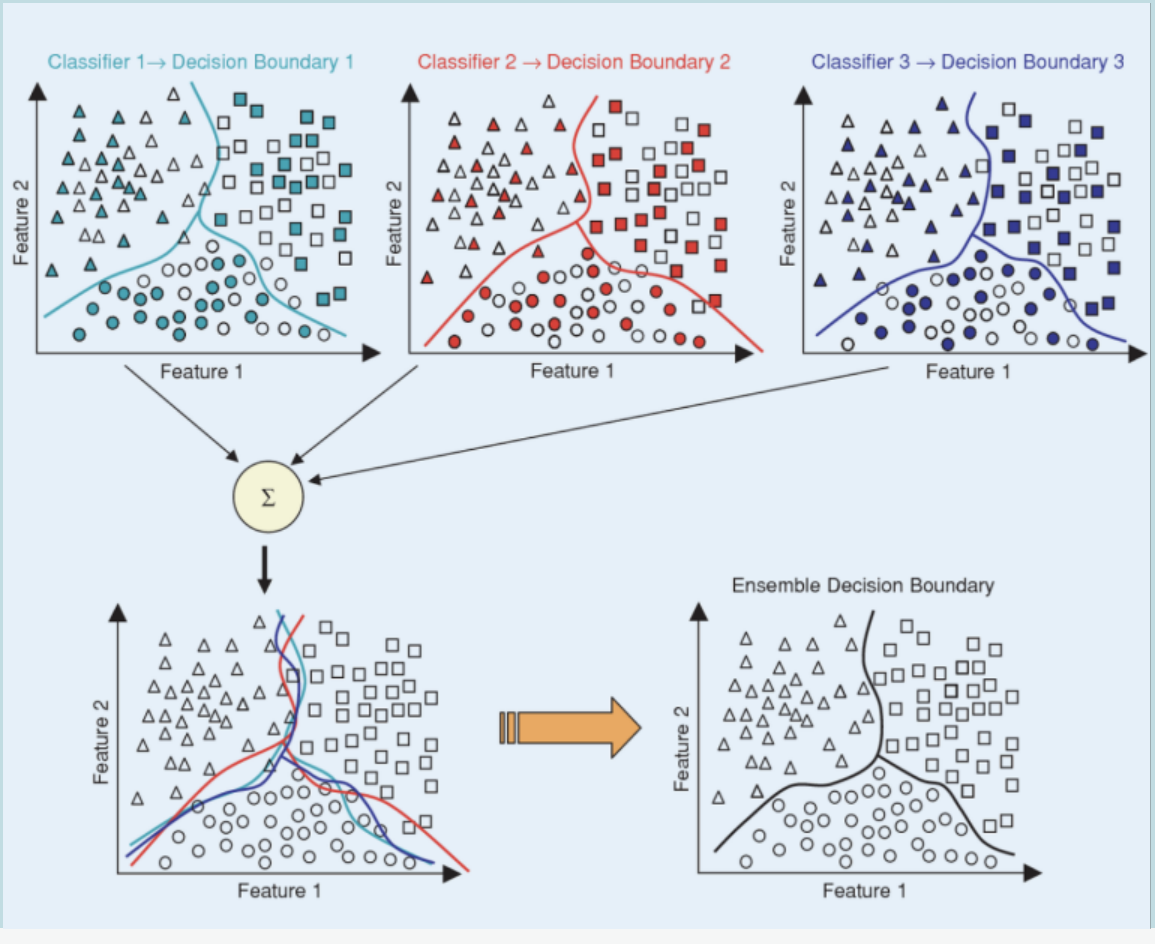
차이점
- 엑스트라 트리와 랜덤 포레스트의 차이는 부트스트랩 샘플을 사용하지 않는다는 점이다. 즉, 각 결정 트리를 만들어낼 때 전체 훈련 세트를 사용한다는 것인데 가장 좋은 분할을 찾는 것이 아닌 무작위로 분할 한다는 뜻이다 .

데이터 학습

Bagging Classifier

정의

-Bagging이란, 기존 학습 데이터(Original Data)로부터 랜덤하게 '복원추출'하여 동일한 사이즈의 데이터셋을 여러개 만들어 앙상블을 구성하는 여러 모델을 학습시키는 방법이다. 이러한 복원추출로 만들어진 새로운 데이터셋을 'Bootstrap'이라고 한다. 여기서 학습 모델은 정해진 것이 아니며, 어떠한 지도 학습 알고리즘이든 다 활용될 수 있다.



데이터 학습

AdaBoost
Classifier

정의
-에이다부스트는 Adaptive + Boosting 로 만들어진 단어입니다.
-예측 성능이 낮은 약한 분류기들을 조합하여 최종적으로 조금 더 성능이 좋은 강한 분류기 하나를 만드는 것이다. 약한 분류기들이 상호보완적(adaptive)으로 학습해나가고, 이러한 약한 분류기들을 조합하여 하나의 분류기를 만들기 때문에 boosting이 된다.

$$H(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + ... + \alpha_t h_t(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

H(x) : 최종 강한 분류기
h : 약한 분류기
α : 약한 분류기의 가중치
t : 반복 횟수

데이터 학습

```
from sklearn.ensemble import RandomForestClassifier , ExtraTreesClassifier , BaggingClassifier , AdaBoostClassifier
```

```
rf_clf = RandomForestClassifier(n_estimators=100)
extratree_clf = ExtraTreesClassifier(n_estimators=100)
bg_clf = BaggingClassifier(n_estimators=100)
ada_clf = AdaBoostClassifier(n_estimators=100)
```

```
rf_clf.fit(X_train , y_train)
extratree_clf.fit(X_train , y_train)
bg_clf.fit(X_train , y_train)
ada_clf.fit(X_train , y_train)
```

```
y_preds_rf = rf_clf.predict(X_test)
y_preds_et = extratree_clf.predict(X_test)
y_preds_bg = bg_clf.predict(X_test)
y_preds_ada = ada_clf.predict(X_test)
```

데이터 학습 - 혼동 행렬

혼동 행렬을 이용한 알고리즘 성능 시각화

혼동 행렬

지도 학습으로 훈련된 분류 알고리즘의 성능을 시각화
할 수 있는 표

		현실	
		긍정(Positive)	부정(Negative)
예측	긍정(Positive)	**참긍정** (TP; True Positive)	**거짓긍정** (FP; False Positive)
	부정(Negative)	**거짓부정** (FN; False Negative)	**참부정** (TN; True Negative)

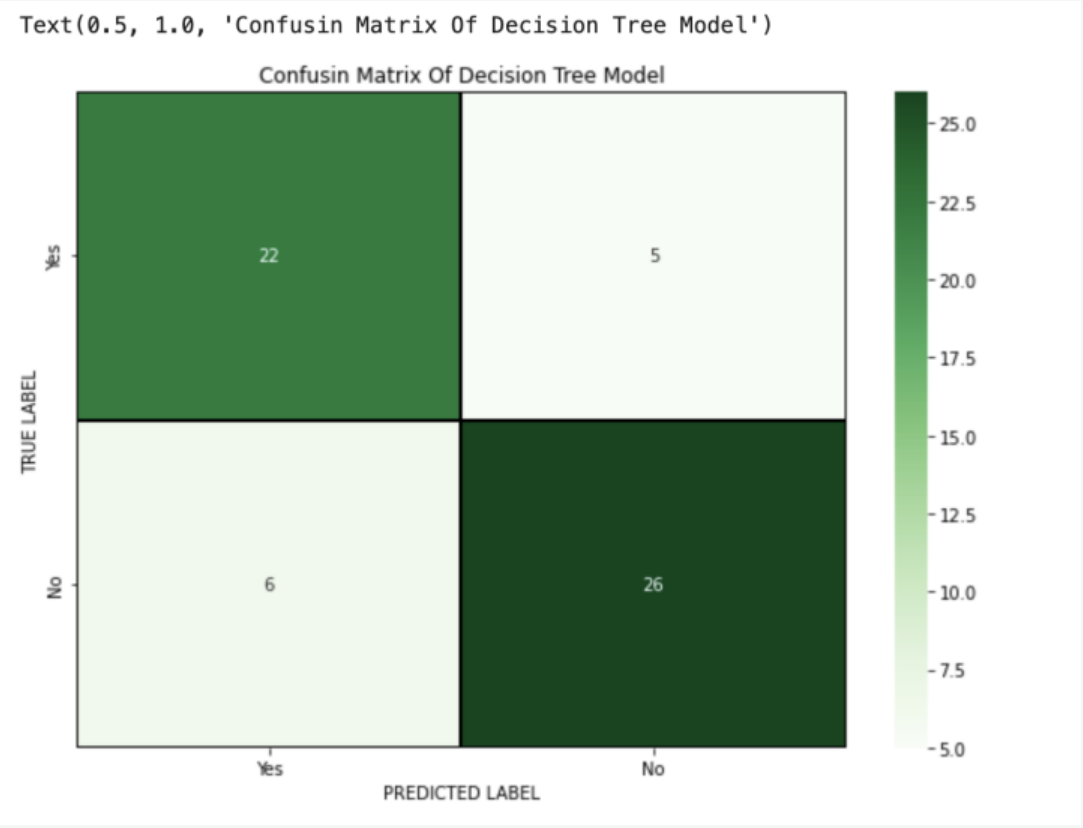
정확도 계산

$$\frac{(TP+TN)}{(TP+TN+FP+FN)}$$

혼동 행렬 - DesisionTree

```
#confusion Matrix of Decision Tree Model
cm  = confusion_matrix(y_test , y_preds_dt)

x_axis_labels = ["Yes" , "No"]
y_axis_labels = ["Yes" , "No"]

f , ax = plt.subplots(figsize=(10,7))
sns.heatmap(cm , annot=True, linewidths=0.2 , linecolor="black" , fmt=".0f" , ax=ax , cmap="Greens" ,
           xticklabels=x_axis_labels , yticklabels=y_axis_labels)
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title("Confusin Matrix Of Decision Tree Model")
```

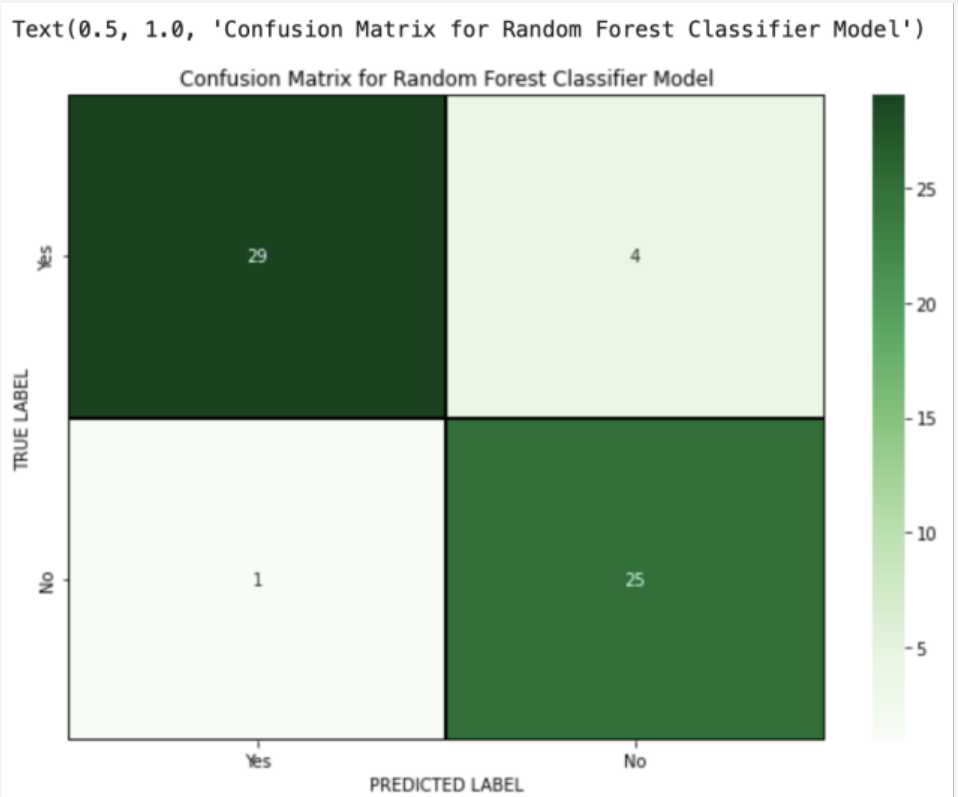
```
Text(0.5, 1.0, 'Confusin Matrix Of Decision Tree Model')
```



혼동 행렬 - RandomForest

```
#Confusion Matrix of Random Forest Classifier Model.
cm = confusion_matrix(y_test, y_preds_rf)

x_axis_labels = ["Yes", "No"]
y_axis_labels = ["Yes", "No"]

f, ax = plt.subplots(figsize =(10,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Greens",
            xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for Random Forest Classifier Model')
```

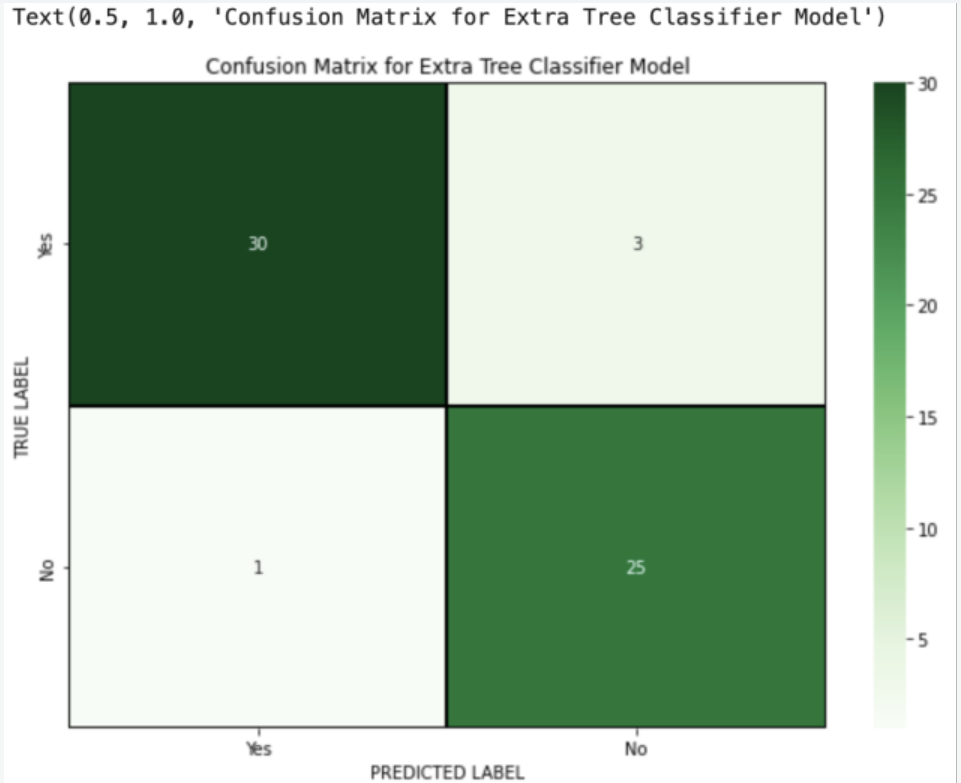
```
Text(0.5, 1.0, 'Confusion Matrix for Random Forest Classifier Model')
```



혼동 행렬 - ExtraTree

```
#Confusion Matrix of Extra Tree Classifier Model.
cm = confusion_matrix(y_test, y_preds_et)

x_axis_labels = ["Yes", "No"]
y_axis_labels = ["Yes", "No"]

f, ax = plt.subplots(figsize =(10,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Greens",
            xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for Extra Tree Classifier Model')
```

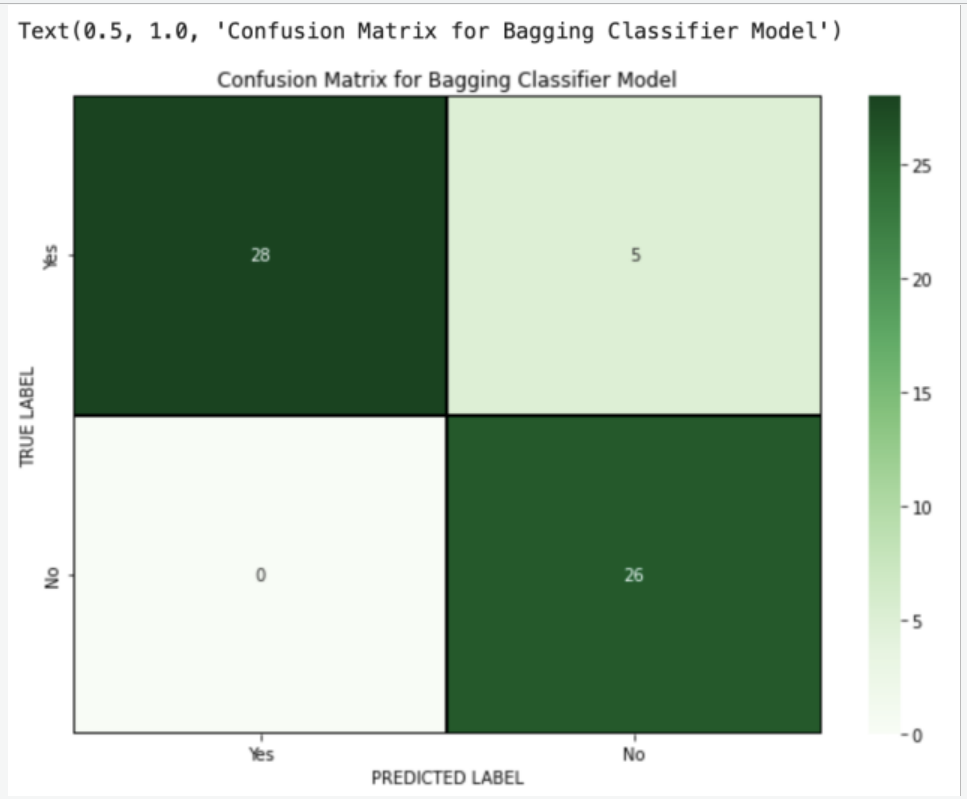
```
Text(0.5, 1.0, 'Confusion Matrix for Extra Tree Classifier Model')
```



혼동 행렬 - Bagging

```
#Confusion Matrix of Bagging Classifier Model.
cm = confusion_matrix(y_test, y_preds_bg)

x_axis_labels = ["Yes", "No"]
y_axis_labels = ["Yes", "No"]

f, ax = plt.subplots(figsize =(10,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Greens",
            xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for Bagging Classifier Model')
```

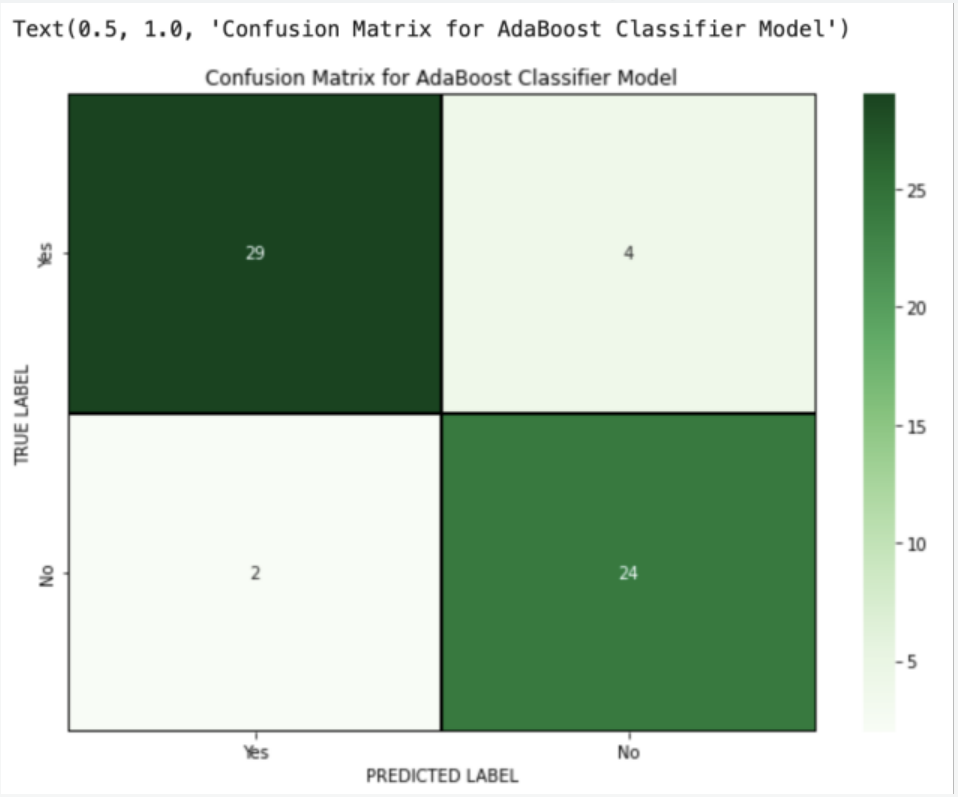
```
Text(0.5, 1.0, 'Confusion Matrix for Bagging Classifier Model')
```



혼동 행렬 - AdaBoost

```
#Confusion Matrix of AdaBoost Classifier Model.
cm = confusion_matrix(y_test, y_preds_ada)

x_axis_labels = ["Yes", "No"]
y_axis_labels = ["Yes", "No"]

f, ax = plt.subplots(figsize =(10,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Greens",
            xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for AdaBoost Classifier Model')
```

```
Text(0.5, 1.0, 'Confusion Matrix for AdaBoost Classifier Model')
```



데이터 학습

혼동 행렬을 통해서 정확도를 계산 하면
Extra Tree > Bagging = Random Forest > AdaBoost
다음과 같은 순서로 정확도가 높다

```
print("Accuracy Score of the Random Forest Model", accuracy_score(y_test, y_preds_rf))
print("Accuracy Score of the Extre Tree Classifier Model", accuracy_score(y_test, y_preds_et))
print("Accuracy Score of the Bagging Classifier Model", accuracy_score(y_test, y_preds_bg))
print("Accuracy Score of the AdaBoost Classifier Model", accuracy_score(y_test, y_preds_ada))
```

```
Accuracy Score of the Random Forest Model 0.9152542372881356
Accuracy Score of the Extre Tree Classifier Model 0.9322033898305084
Accuracy Score of the Bagging Classifier Model 0.9152542372881356
Accuracy Score of the AdaBoost Classifier Model 0.8983050847457628
```

실제 스코어 점수를 보았을 때
우리가 확인한 성능 순서가 정확하다는 사실을 알 수 있다.

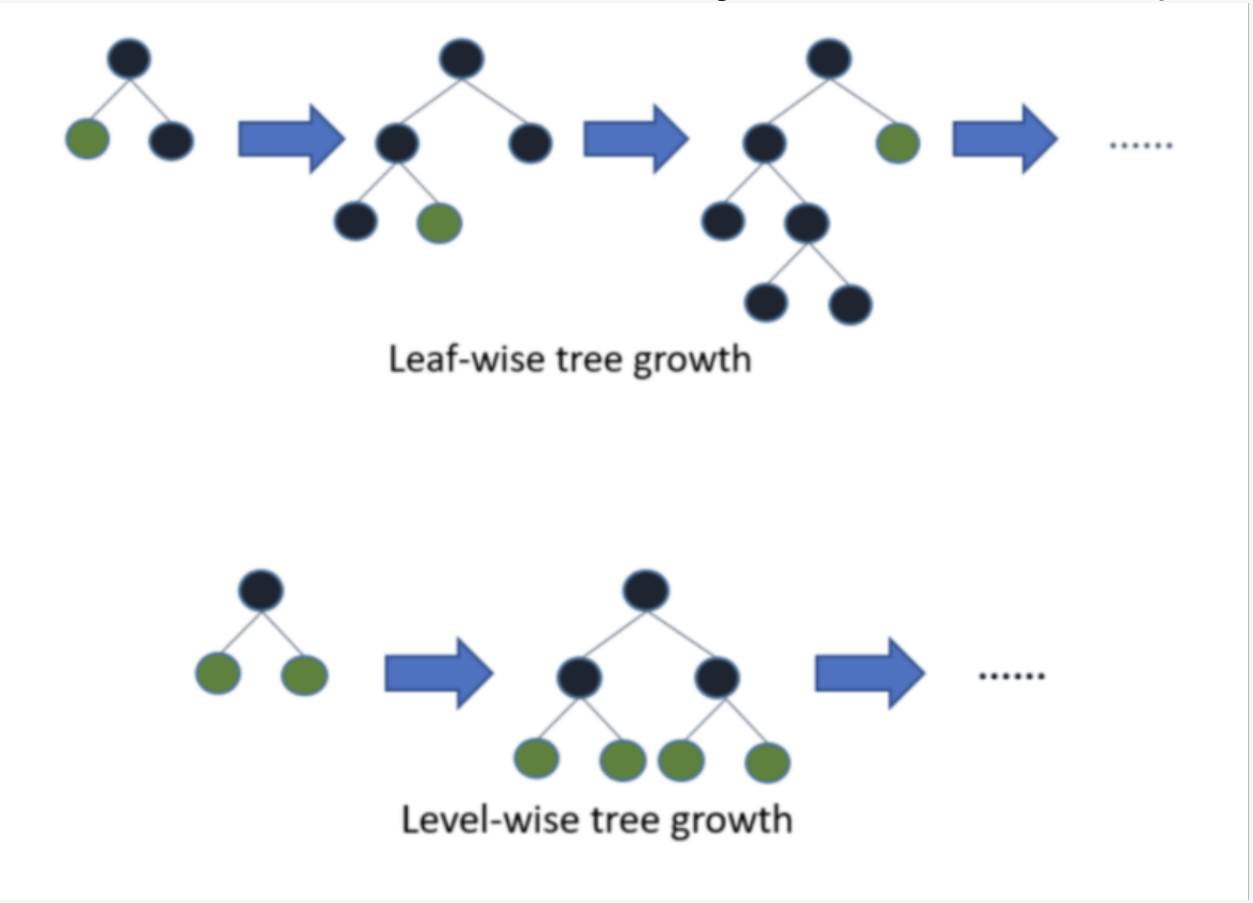
LGBM Classifier

정의

-Light GBM(Light Gradient Boosting Machine)은 트리 기반의 학습 알고리즘인 gradient boosting 방식의 프레임 워크이다.

차이점

- Light GBM은 다른 알고리즘이 트리를 수평으로 확장하는 것에 반해 트리를 수직으로 확장한다. 즉, 기존의 알고리즘은 수평으로 확장하여 포화 트리를 만들고(Level-wise tree growth), left-wise tree growth인 LGBM은 최대 delta loss가 증가하도록 잎의 개수를 정한다. 그렇기에 leaf-wise 알고리즘은 다른 level-wise 알고리즘보다 낮은 loss를 달성하는 경향이 있다. 데이터의 크기가 작은 경우 leaf-wise는 과적합(overfitting)이 되기 쉬우므로 max_depth를 줄여줘야 한다.



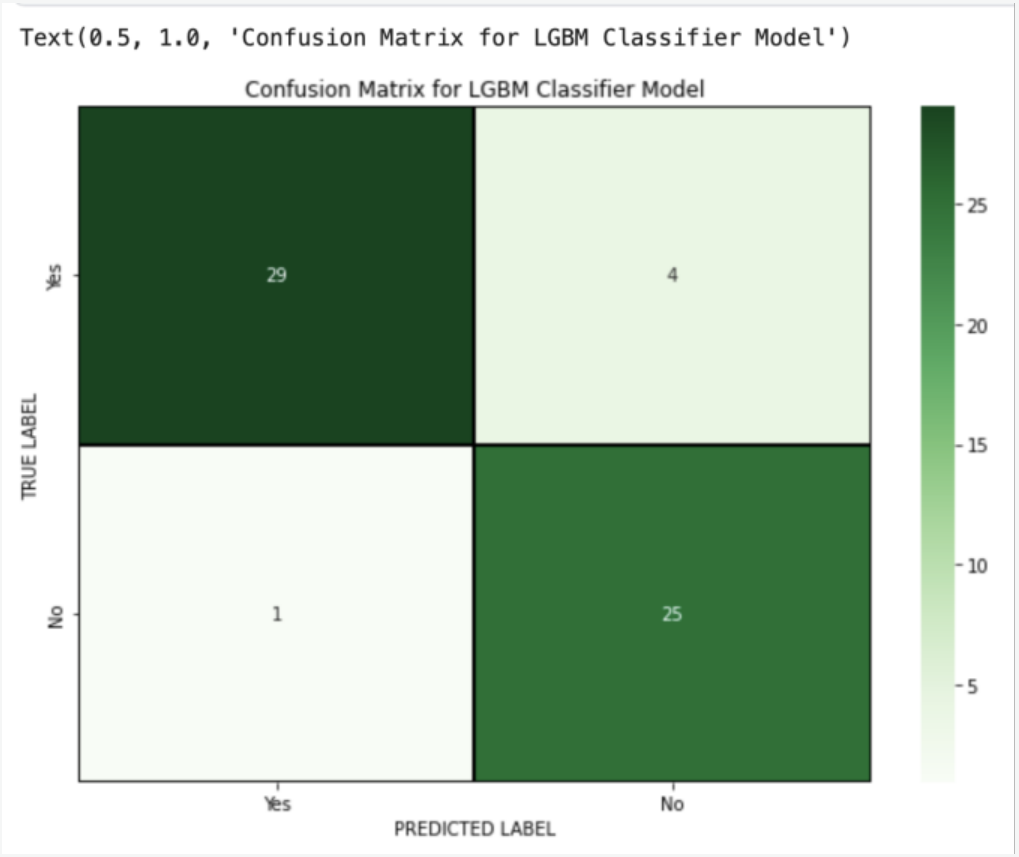
LGBM Classifier

```
from lightgbm import LGBMClassifier
lgbm_clf = LGBMClassifier()
lgbm_clf.fit(X_train , y_train)
y_preds_lgbm = lgbm_clf.predict(X_test)
```

```
#Confusion Matrix of LGBM Classifier Model.
cm = confusion_matrix(y_test, y_preds_lgbm)

x_axis_labels = ["Yes", "No"]
y_axis_labels = ["Yes", "No"]

f, ax = plt.subplots(figsize =(10,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Greens"
            xticklabels=x_axis_labels, yticklabels=y_axis_labels)
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for LGBM Classifier Model')
```

LGBM Classifier

```
Text(0.5, 1.0, 'Confusion Matrix for LGBM Classifier Model')
```




```
print("Accuracy Score of the LGBM Model", accuracy_score(y_test, y_preds_lgbm))
```

```
Accuracy Score of the LGBM Model 0.9152542372881356
```

소감

이번 발표를 준비하면서 머신러닝 알고리즘에 대해 더욱더 이해할 수 있었고 케글과 파이썬에 대해서도 더욱 알게 되는 시간이었습니다.

또한 여러 머신러닝 알고리즘을 사용하기 위해 캐글을 둘러보며 알고리즘 사용 방법을 찾는 과정에서 교수님께서 강조하신 Know where 의 중요성에 대해서도 함께 알아가는 시간이었습니다.