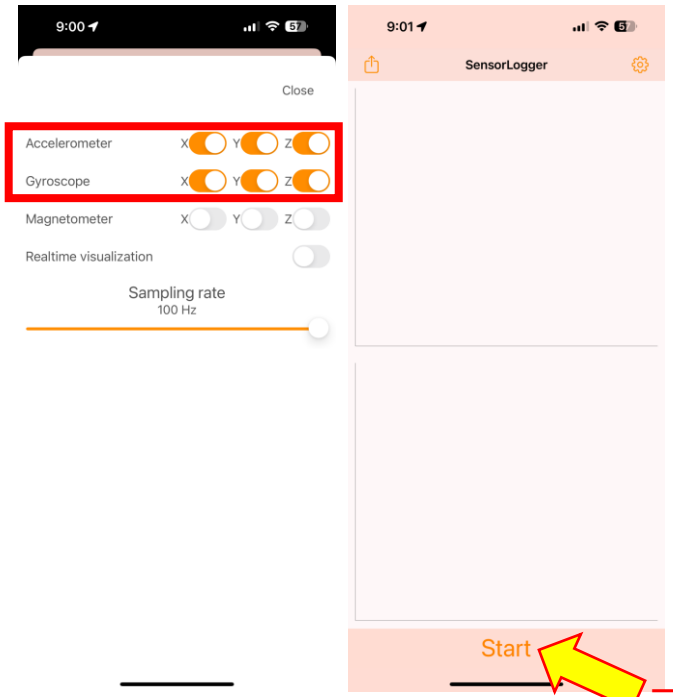
응원봉 센서 데이터 EDA 분석

팀 123
202215084 박지현
202110834 김정우
202115064 김동주
202410773 손주희
202410948 구채린

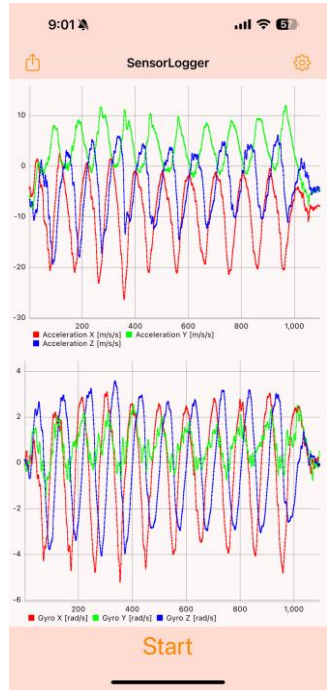
데이터 출처

Sensor Logger 앱(유료)



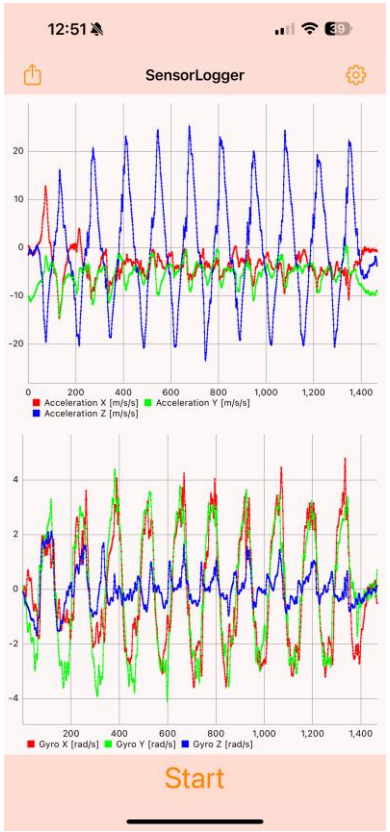
클릭!

Accelerometer 센서와 Gyroscope 센서 이용
Sampling rate로 100Hz를 사용

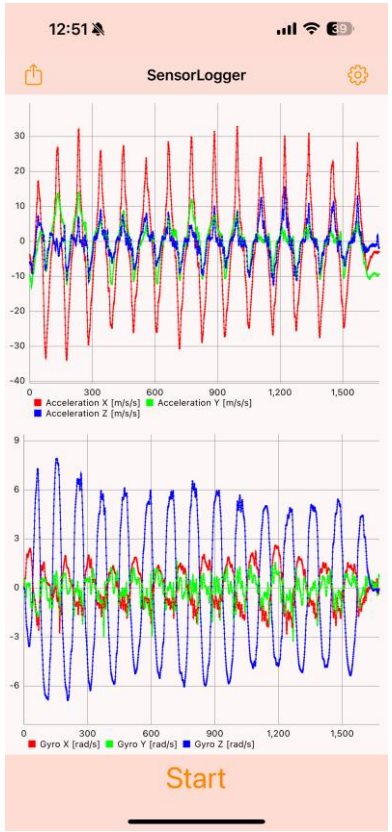


csv 파일로 저장

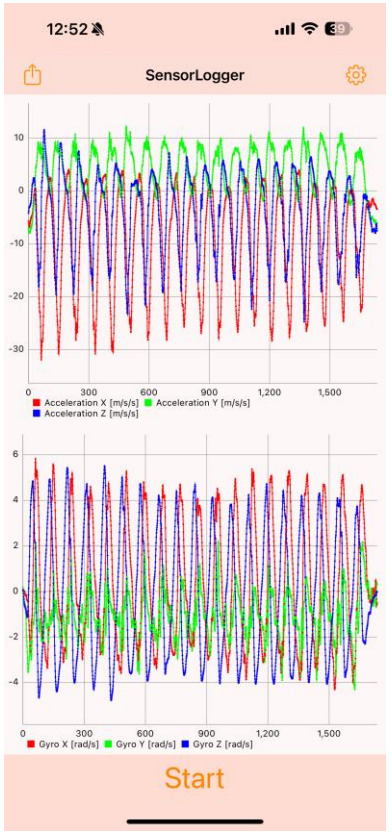
데이터 출처



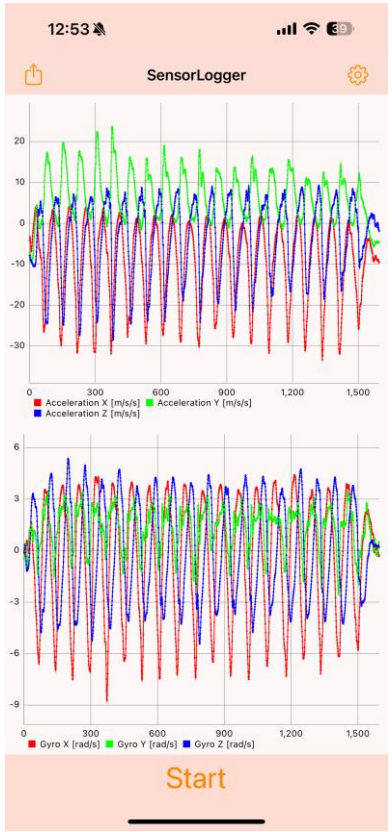
좌우로 흔들기
shaking_lr



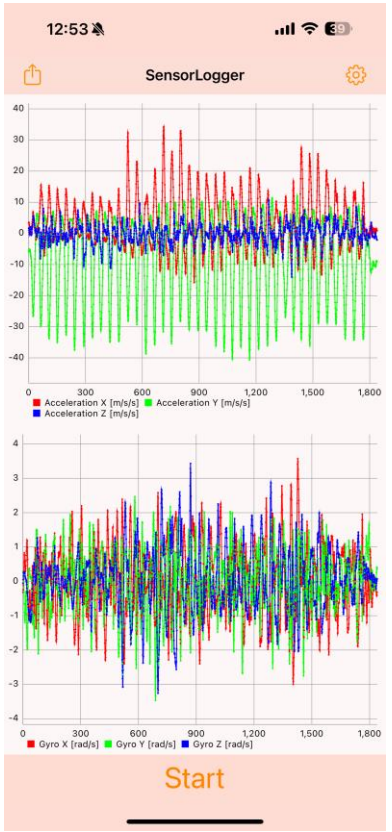
앞뒤로 흔들기
shaking_fb



시계방향으로 돌리기
rotate_cw



반시계 방향으로 돌리기
rotate_ccw



점프
jumping

과연 이 행동들을 구분할 수 있을까?

데이터 출처

	A	B	C	D	E	F	G
1	SamplingTime	AccelerationX	AccelerationY	AccelerationZ	GyroX	GyroY	GyroZ
2	1733469942	0.554557	-12.349442	-2.350059	-0.1491	0.392214	-0.47238
3	1733469942	1.289128	-12.496536	-2.169745	-0.10758	0.324546	-0.54511
4	1733469942	1.954267	-12.594698	-2.248904	0.01221	0.194641	-0.57276
5	1733469942	2.25444	-13.143718	-2.817227	0.25541	-0.05867	-0.55939
6	1733469942	2.094926	-13.35755	-3.113809	0.489295	-0.25822	-0.56065
7	1733469942	0.943615	-13.753342	-2.82441	0.697881	-0.52266	-0.67257
8	1733469942	0.369306	-14.308198	-2.530821	0.789302	-0.77861	-0.83765
9	1733469942	-0.024989	-14.136414	-2.398092	0.800005	-0.91444	-1.00888
10	1733469942	-0.306907	-14.15063	-2.571672	0.759906	-0.96066	-1.19076
11	1733469942	-0.861763	-13.998747	-2.971803	0.693831	-0.86453	-1.38632
12	1733469942	-1.404499	-13.734637	-3.972878	0.670484	-0.61332	-1.59596
13	1733469942	-1.757943	-13.32448	-5.080047	0.758419	-0.30005	-1.7921
14	1733469942	-2.002151	-13.034782	-5.697901	0.938567	-0.03326	-2.01755
15	1733469942	-1.959804	-12.584971	-5.664382	1.171339	0.089768	-2.2547
16	1733469942	-1.695544	-11.741913	-5.190629	1.446607	0.042056	-2.47881
17	1733469942	-1.355268	-10.978163	-4.439448	1.70935	-0.14588	-2.71296
18	1733469942	-0.902764	-10.153959	-3.64607	1.921022	-0.39167	-2.92244
19	1733469942	-0.456993	-9.056517	-3.096601	2.097569	-0.65388	-3.09465
20	1733469942	-0.32217	-8.244583	-2.605191	2.242822	-0.9407	-3.26551
21	1733469942	-0.105495	-7.47784	-2.268057	2.347796	-1.18663	-3.42287

추출된 csv 파일 예시

이 데이터를 전처리 시 **윈도우 처리**를 해야 한다.

윈도우 처리는 연속적인 센서 데이터를 일정 구간으로 나누어 **시간적 특징 유지**, **노이즈 감소**, **행동 패턴 추출** 등 다양한 이점을 제공한다. 이를 통해 머신러닝 모델이 센서 데이터를 효과적으로 학습하고, 특정 행동을 정확히 구분하는 데 도움이 된다.

raw 데이터 전처리

```python
import pandas as pd
import numpy as np
from scipy.stats import skew, kurtosis

def sma(window, columns): # Signal Magnitude Area(주로 움직임의 크기를 나타냄)
    return np.sum(np.abs(window[columns]), axis=1).mean()

def energy(window, column): # 에너지 계산(신호의 강도를 나타냄)
    return np.sum(window[column]**2) / len(window[column])

def iqr(window, column): # 데이터의 75%와 25% 사이의 범위를 계산하여 데이터의 분산 정도를 나타냄
    return np.percentile(window[column], 75) - np.percentile(window[column], 25)

def mad(window, column): # Median Absolute Deviation(데이터 값이 중앙값에서 얼마나 벗어나는지 나타냄)
    return np.median(np.abs(window[column] - np.median(window[column])))

def meanFreq(window, column):  # 푸리에 변환 계산 후 주파수 가중평균 반환(주파수 도메인에서 데이터의 주요 변화를 나타냄)
    # 주어진 데이터에 푸리에 변환을 적용한 뒤, 주파수와 진폭의 가중평균을 계산
    fft_vals = np.fft.fft(window[column])
    freqs = np.fft.fftfreq(len(fft_vals))
    mag = np.abs(fft_vals)
    return np.sum(freqs * mag) / np.sum(mag)

def bandsEnergy(window, column, num_bins=64): # 주파수 대역 에너지 계산(푸리에 변환 후, 상위 num_bins 내의 에너지를 계산)
    fft_vals = np.fft.fft(window[column])
    mag = np.abs(fft_vals)[:num_bins]
    return np.sum(mag**2) / num_bins

def angle(vec1, vec2): # 두 벡터 사이의 각도 계산
    dot_product = np.dot(vec1, vec2)
    magnitude = np.linalg.norm(vec1) * np.linalg.norm(vec2)
    return np.arccos(dot_product / magnitude) if magnitude != 0 else 0

```

raw 데이터 전처리

```python
def process_motion(file_path, window_size=100, stride=50): # 윈도우 크기: 100, 윈도우 이동 간격: 50
    data = pd.read_csv(file_path)
    data.columns = data.columns.str.strip() # 열 이름의 공백 제거

    windows = []
    for i in range(0, len(data) - window_size, stride): # stride만큼 건너뛰면서 윈도우를 만듦
        window = data.iloc[i:i + window_size] # 데이터에서 window_size 만큼의 데이터를 슬라이딩 방식으로 잘라 윈도우 생성.

        features = {}

        for sensor in ['Acceleration', 'Gyro']:
            for axis in ['X', 'Y', 'Z']: # 가속도(Acceleration)와 자이로(Gyro)의 X, Y, Z 축 데이터를 기준으로 다양한 특징(mean, std 등) 계산
                col = f"{sensor}{axis}"
                features[f'mean_{sensor.lower()}_{axis.lower()}'] = window[col].mean()
                features[f'std_{sensor.lower()}_{axis.lower()}'] = window[col].std()
                features[f'mad_{sensor.lower()}_{axis.lower()}'] = mad(window, col)
                features[f'max_{sensor.lower()}_{axis.lower()}'] = window[col].max()
                features[f'min_{sensor.lower()}_{axis.lower()}'] = window[col].min()
                features[f'energy_{sensor.lower()}_{axis.lower()}'] = energy(window, col)
                features[f'bandsEnergy_{sensor.lower()}_{axis.lower()}'] = bandsEnergy(window, col)
                features[f'iqr_{sensor.lower()}_{axis.lower()}'] = iqr(window, col)
                features[f'meanFreq_{sensor.lower()}_{axis.lower()}'] = meanFreq(window, col)
                features[f'skewness_{sensor.lower()}_{axis.lower()}'] = skew(window[col])
                features[f'kurtosis_{sensor.lower()}_{axis.lower()}'] = kurtosis(window[col])

        features['sma_acc'] = sma(window, ['AccelerationX', 'AccelerationY', 'AccelerationZ']) # 현재 구간(window) 동안의 전체 가속도 크기
        features['sma_gyro'] = sma(window, ['GyroX', 'GyroY', 'GyroZ']) # 현재 구간(window) 동안의 전체 회전 운동 크기

        acc_vector = [window['AccelerationX'].mean(), window['AccelerationY'].mean(), window['AccelerationZ'].mean()] # 가속도 데이터의 평균으로 구성된 벡터
        gyro_vector = [window['GyroX'].mean(), window['GyroY'].mean(), window['GyroZ'].mean()] # 자이로스코프 데이터의 평균으로 구성된 벡터
        features['angle_acc_gyro'] = angle(acc_vector, gyro_vector) # 가속도와 자이로스코프 벡터 간의 각도

        features['label'] = 'shaking_lr' # 각 데이터셋에 해당하는 label 입력

        windows.append(features)

    return pd.DataFrame(windows)
```

raw 데이터 전처리

```
# 6개의 dataset을 사용. 아래의 6개의 data는 모두 같은 행동의 data
# 각각의 dataset은 15초 동안 특정 행동을 수행하여 추출한 data.
file_paths = [
    'sensordata1.csv',
    'sensordata2.csv',
    'sensordata3.csv',
    'sensordata4.csv',
    'sensordata5.csv',
    'sensordata6.csv'
]

window_size = 100  # 윈도우 크기: 100
stride = 50  # 윈도우 이동 간격: 50

all_motion_features = []

for file_path in file_paths:
    motion_features = process_motion(file_path, window_size, stride)
    all_motion_features.append(motion_features)

motion_features_df = pd.concat(all_motion_features, ignore_index=True)  # 각 파일별로 처리한 결과를 하나의 데이터프레임으로 결합.

output_file_path = "shaking_lr_dataset.csv"
motion_features_df.to_csv(output_file_path, index=False)

output_file_path
```

```
'shaking_lr_dataset.csv'
```

전처리한 데이터 결합

[3]
```python
import pandas as pd
import numpy as np

file_paths = {
    "jumping": "jumping_dataset.csv",   # 점프
    "shaking_lr": "shaking_lr_dataset.csv",  # 좌우로 흔들기
    "shaking_fb": "shaking_fb_dataset.csv",  # 앞뒤로 흔들기
    "rotate_cw": "rotate_cw_dataset.csv",    # 시계 방향으로 돌리기
    "rotate_ccw": "rotate_ccw_dataset.csv",  # 반시계 방향으로 돌리기
}

dataframes = {key: pd.read_csv(path) for key, path in file_paths.items()}

combined_df = pd.concat(dataframes.values(), ignore_index=True)
combined_df.head()

combined_df.to_csv('sensor_data.csv', index=False)
```

[4]
```python
data = pd.read_csv('sensor_data.csv')
```

전체 데이터 정보 확인

- 전체 데이터의 행과 열 개수

```
[6]  1 #전체 데이터의 행,열 개수 확인
     2 data.shape

     (944, 70)
```

944개의 행과 70개의 열로 구성된 데이터 프레임이다.

- 전체 데이터 상위 5개 행(행의 개수가 많아서 간략히 상위 5개 행만 살펴보았다.)

```
[5]  1 # 전체 데이터의 상위 5개 행 확인
     2 data.head()
```

	mean_acceleration_x	std_acceleration_x	mad_acceleration_x	max_acceleration_x	min_acceleration_x	energy_acceleration_x
0	5.116406	17.380434	10.424653	46.241012	-34.851047	325.236293
1	6.348106	23.097543	16.827497	46.241012	-34.851047	568.459991
2	6.933795	21.229315	18.827329	44.736705	-27.075578	494.254495
3	4.681370	19.115007	14.301913	39.048531	-32.370804	383.644882
4	4.166894	16.854734	8.024915	39.048531	-32.370804	298.604246

5 rows × 70 columns

전체 데이터 정보 확인

- 전체 데이터의 모든 변수(컬럼)

```
[7]  1 #전체 데이터의 모든 변수 확인
     2 data.columns
```

```
Index(['mean_acceleration_x', 'std_acceleration_x', 'mad_acceleration_x',
       'max_acceleration_x', 'min_acceleration_x', 'energy_acceleration_x',
       'bandsEnergy_acceleration_x', 'iqr_acceleration_x',
       'meanFreq_acceleration_x', 'skewness_acceleration_x',
       'kurtosis_acceleration_x', 'mean_acceleration_y', 'std_acceleration_y',
       'mad_acceleration_y', 'max_acceleration_y', 'min_acceleration_y',
       'energy_acceleration_y', 'bandsEnergy_acceleration_y',
       'iqr_acceleration_y', 'meanFreq_acceleration_y',
       'skewness_acceleration_y', 'kurtosis_acceleration_y',
       'mean_acceleration_z', 'std_acceleration_z', 'mad_acceleration_z',
       'max_acceleration_z', 'min_acceleration_z', 'energy_acceleration_z',
       'bandsEnergy_acceleration_z', 'iqr_acceleration_z',
       'meanFreq_acceleration_z', 'skewness_acceleration_z',
       'kurtosis_acceleration_z', 'mean_gyro_x', 'std_gyro_x', 'mad_gyro_x',
       'max_gyro_x', 'min_gyro_x', 'energy_gyro_x', 'bandsEnergy_gyro_x',
       'iqr_gyro_x', 'meanFreq_gyro_x', 'skewness_gyro_x', 'kurtosis_gyro_x',
       'mean_gyro_y', 'std_gyro_y', 'mad_gyro_y', 'max_gyro_y', 'min_gyro_y',
       'energy_gyro_y', 'bandsEnergy_gyro_y', 'iqr_gyro_y', 'meanFreq_gyro_y',
       'skewness_gyro_y', 'kurtosis_gyro_y', 'mean_gyro_z', 'std_gyro_z',
       'mad_gyro_z', 'max_gyro_z', 'min_gyro_z', 'energy_gyro_z',
       'bandsEnergy_gyro_z', 'iqr_gyro_z', 'meanFreq_gyro_z',
       'skewness_gyro_z', 'kurtosis_gyro_z', 'sma_acc', 'sma_gyro',
       'angle_acc_gyro', 'label'],
      dtype='object')
```

총 70개의 컬럼이 있으며, 그중 'label'을 제외한 나머지 컬럼들은 모두 feature에 속한다.
즉, 69개의 feature와 1개의 label로 구성되어 있다.

전체 데이터 정보 확인

- 전체 데이터의 수치형 변수 분포

```
[8]  1 #전체 데이터의 수치형 변수 분포 확인
     2 data.describe()
```

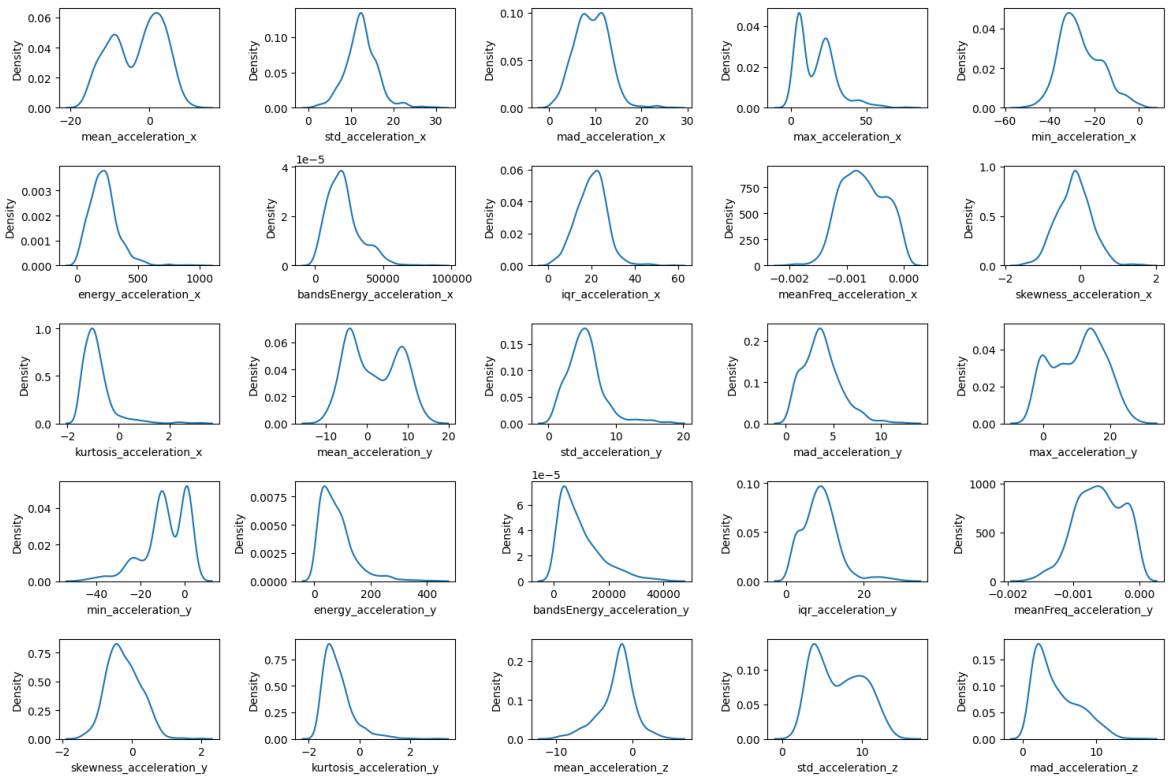
	mean_acceleration_x	std_acceleration_x	mad_acceleration_x	max_acceleration_x	min_acceleration_x	energy_acceleration
count	944.000000	944.000000	944.000000	944.000000	944.000000	944.000
mean	-3.025421	12.541333	9.249932	17.031736	-26.287119	220.761
std	6.478564	3.758346	3.615499	12.641306	9.214438	120.593
min	-15.986049	1.540763	0.488118	-1.028609	-50.330758	6.282
25%	-8.748598	10.390350	6.712986	5.854796	-32.926558	142.887
50%	-1.660457	12.462589	9.188348	16.983793	-27.887511	210.982
75%	2.420686	14.594264	11.829844	24.428451	-19.759720	276.253
max	10.940450	30.042704	26.520722	76.318320	0.863858	1009.779

8 rows × 69 columns

전체 데이터 정보 확인

- 데이터의 단변량 분석(전체 변수 중 25개의 변수만을 이용하여 kdeplot으로 표현)

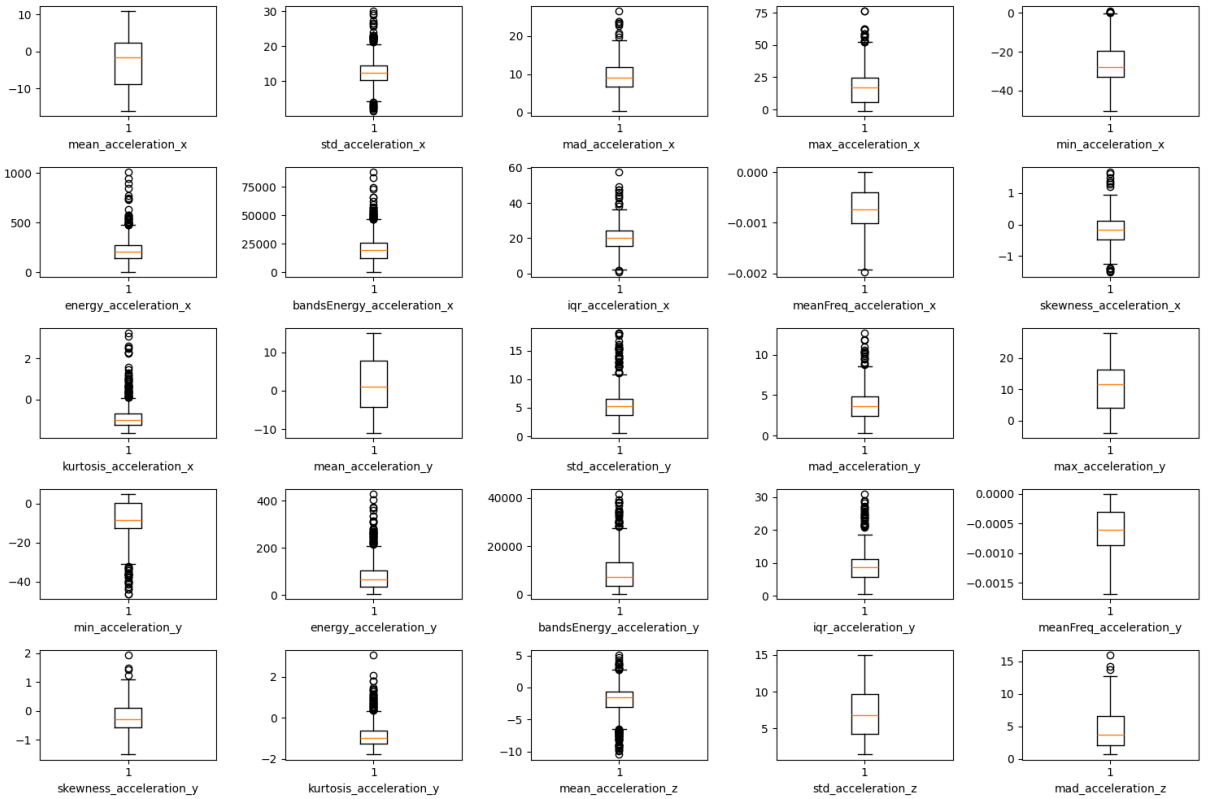
```python
# kde로 feature 분포 확인
fig, axes = plt.subplots(5, 5, figsize=(15,10))
idx = 0
for i in range(5):
    for j in range(5):
        sns.kdeplot(data[data.columns[idx]], ax=axes[i,j])
        idx += 1
plt.tight_layout()
plt.show()
```



전체 데이터 정보 확인

- 데이터의 단변량 분석(전체 변수 중 25개의 변수만을 이용하여 boxplot으로 표현)

```
# boxplot()으로 feature 분포 확인
fig, axes = plt.subplots(5, 5, figsize=(15,10))
idx = 0
for i in range(5):
    for j in range(5):
        axes[i,j].boxplot(data[data.columns[idx]])
        axes[i,j].set_xlabel(data.columns[idx])
        idx += 1
plt.tight_layout()
plt.show()
```



전체 데이터 정보 확인

- 데이터의 이변량 분석(상관관계(corr)로 분석을 진행)

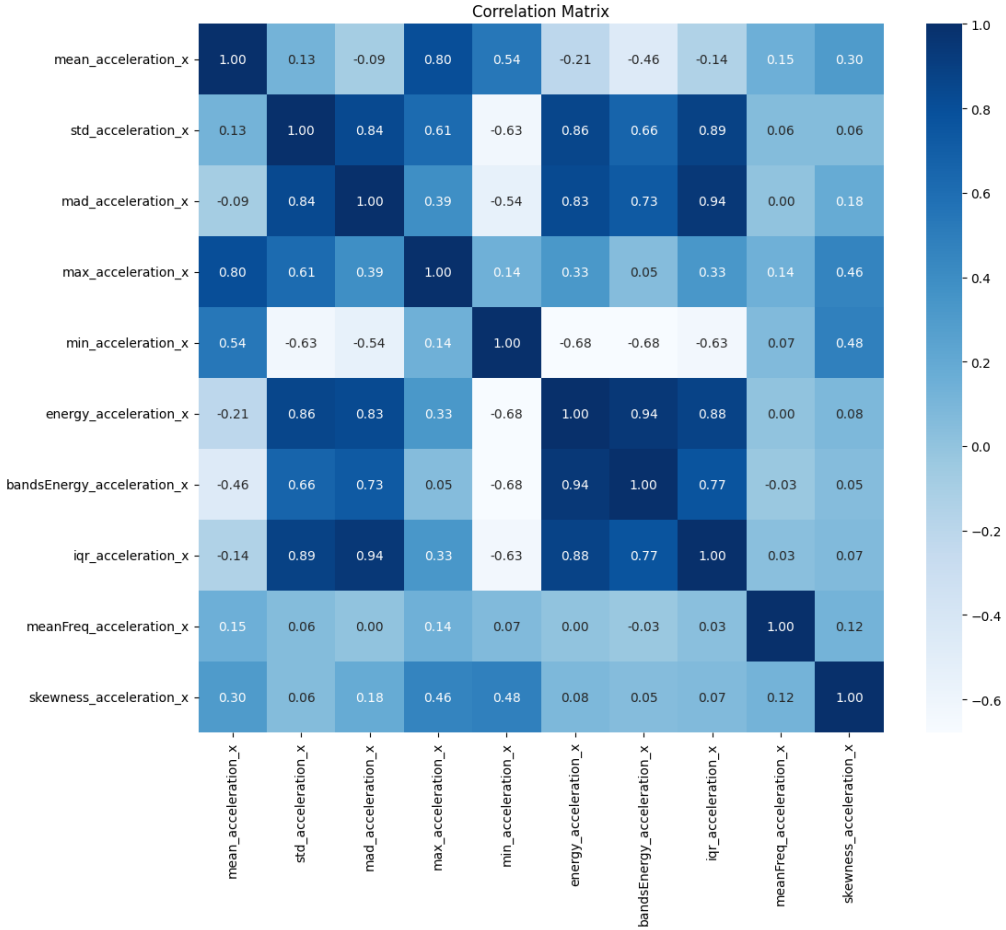
```
[11]  1 # 전체 데이터의 이변량 분석 (corr)
      2 data.iloc[:, :-1].corr()
```

	mean_acceleration_x	std_acceleration_x	mad_acceleration_x	max_acceleration_x	min_acceleration_x	energy_acceleration_x	bandsEnergy_acceleration_x	iqr_acceleration
mean_acceleration_x	1.000000	0.125866	-0.085787	0.803550	0.538748	-0.208910	-0.457397	-0.1442
std_acceleration_x	0.125866	1.000000	0.841800	0.613859	-0.626146	0.856133	0.661303	0.8853
mad_acceleration_x	-0.085787	0.841800	1.000000	0.388145	-0.542380	0.829674	0.725255	0.9411
max_acceleration_x	0.803550	0.613859	0.388145	1.000000	0.141299	0.328894	0.053847	0.3334
min_acceleration_x	0.538748	-0.626146	-0.542380	0.141299	1.000000	-0.675124	-0.679046	-0.6296
...	...	...	...	...	...	...	...	
skewness_gyro_z	0.127570	0.312263	0.178055	0.272484	-0.214724	0.224858	0.159039	0.1964
kurtosis_gyro_z	0.358974	-0.041160	-0.210958	0.278992	0.210772	-0.078004	-0.128866	-0.1959
sma_acc	-0.637670	0.414230	0.500060	-0.244386	-0.665263	0.710737	0.835107	0.5761
sma_gyro	-0.774708	0.279696	0.340400	-0.488513	-0.786666	0.481163	0.610718	0.4359
angle_acc_gyro	0.014438	0.035701	0.027854	0.015454	-0.011633	-0.019344	-0.044234	0.0616

69 rows × 69 columns

전체 데이터 정보 확인

- 데이터의 이변량 분석(변수의 개수가 매우 많으므로 변수10개만 선택하여 히트맵으로 표현)



변수들 사이의 상관관계가 큰 변수들이 있는 것을 파악할 수 있다.

상관관계가 큰 변수들을 이용해 모델을 학습하면, 모델이 불필요한 중복 정보를 학습하게 되어 과적합 될 가능성이 있다.

만약 랜덤 포레스트 모델을 사용한다면 변수의 개수가 많아도 각 트리에서 랜덤으로 일부 변수만 선택하여 분류하기 때문에, 높은 차원의 데이터에서도 비교적 강력한 성능을 보인다. 하지만, 상관관계가 큰 변수가 많을 경우 랜덤 포레스트도 중요하지 않은 정보를 선택할 가능성이 있으므로, 데이터 전처리 과정에서 변수 선택이나 차원 축소를 고려하는 것이 유용할 수 있다.

```python
# 이변량 시각화(변수 10개만 선택하여 표현).
correlation_matrix = data.iloc[:, :10].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='Blues', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```

전체 데이터 정보 확인

- 데이터의 범주 종류 및 범주별 빈도(막대 그래프로 표현)

```
[54]  1 # 범주의 종류
      2 target.unique()
```

```
array(['jumping', 'shaking_lr', 'shaking_fb', 'rotate_cw', 'rotate_ccw'],
      dtype=object)
```

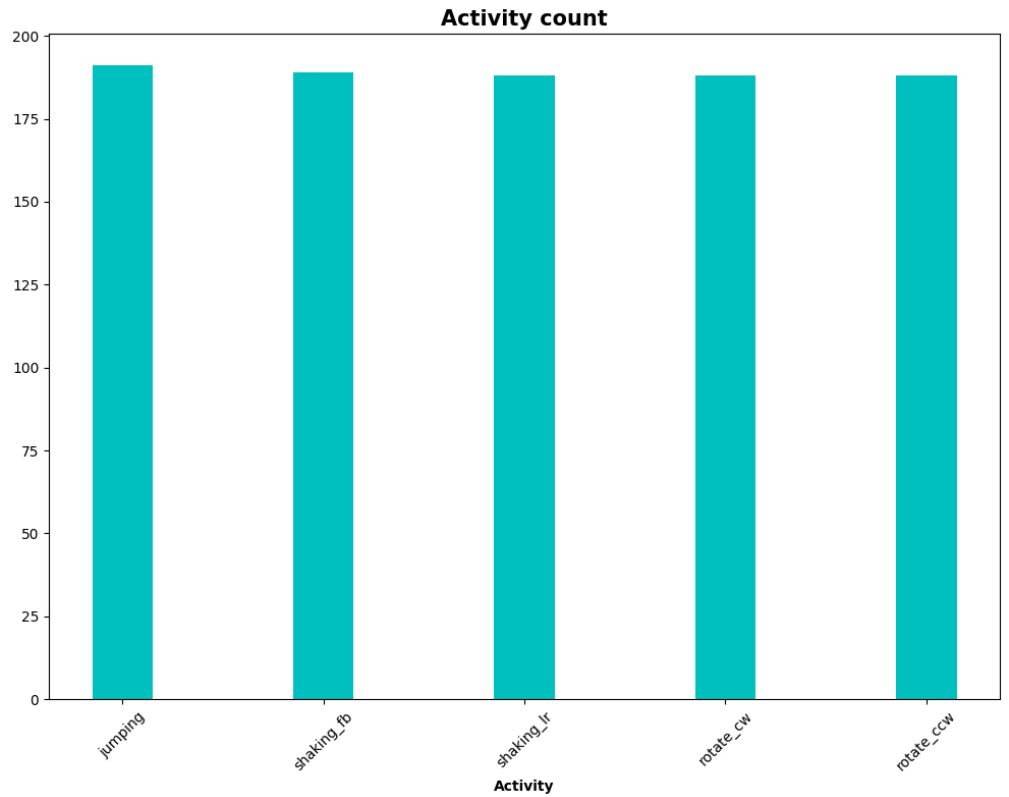
```
1 # 범주별 빈도수
2 target.value_counts()
```

label	count
jumping	191
shaking_fb	189
shaking_lr	188
rotate_cw	188
rotate_ccw	188

dtype: int64

jumping -> 점프
shaking_fb -> 앞뒤로 흔들기
shaking_lr -> 좌우로 흔들기
rotate_cw -> 시계방향으로 돌리기
rotate_ccw -> 반시계방향으로 돌리기

```
1 fig, ax = plt.subplots(figsize=(10, 8))
2 ax.bar(target.value_counts().index, target.value_counts(), width=0.3, color='c')
3 ax.set_title('Activity count', fontsize=15, fontweight='bold')
4 ax.set_xlabel('Activity', fontweight='bold')
5 plt.xticks(rotation=45)
6 plt.tight_layout()
7 plt.show()
```



전체 데이터 정보 확인

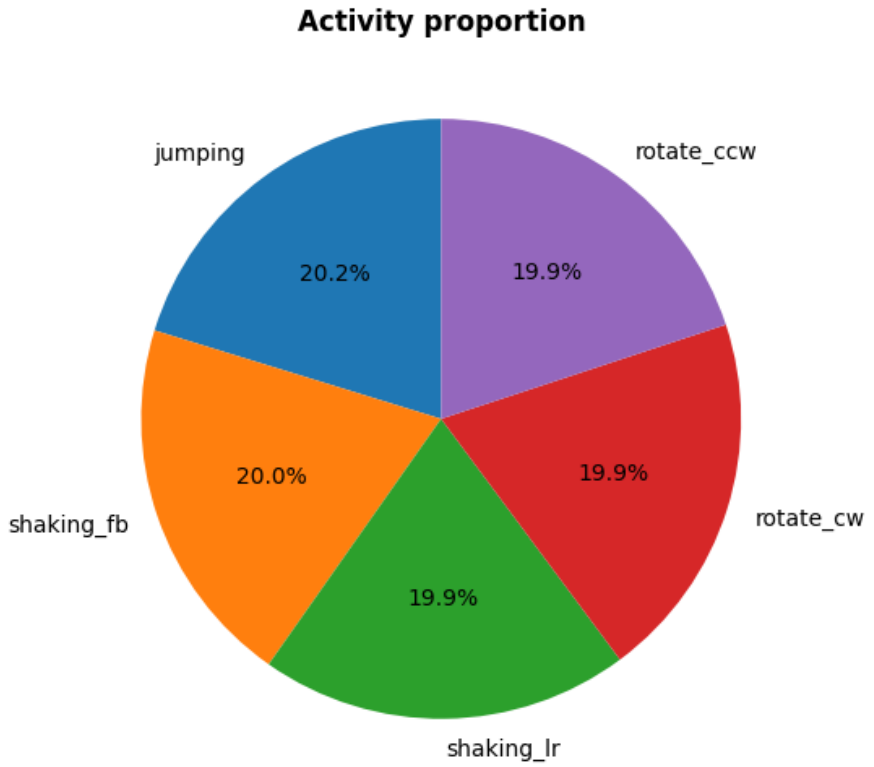
- 데이터의 범주별 비율(파이 그래프로 표현)

```
1 # 범주별 비율
2 target_proportion = target.value_counts(normalize=True)*100
3 target_proportion
```

label	proportion
jumping	20.233051
shaking_fb	20.021186
shaking_lr	19.915254
rotate_cw	19.915254
rotate_ccw	19.915254

dtype: float64

```
1 fig, ax = plt.subplots(figsize=(8, 6))
2 ax.pie(target_proportion, labels=target_proportion.index, startangle=90, autopct='%.1f%%')
3 ax.set_title('Activity proportion', fontweight='bold')
4 plt.show()
```



랜덤 포레스트 모델 훈련

- 데이터 전처리 및 모델링(EDA분석에서 사용된 모든 모델은 분석의 일관성을 위해 random_state=42로 설정하였다)

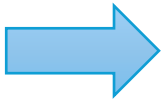
Splitting the dataset into the Training set and Test set

```
# 데이터 분할을 위한 전처리

target = 'label'

x = data.drop(target, axis = 1)
y = data.loc[:, target]

x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.2, random_state=42)
```

Feature Scaling

[60]
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train_scaled = sc.fit_transform(x_train)
x_val_scaled = sc.transform(x_val)
```

[61]
```
# 넘파이 배열을 다시 데이터프레임으로 변환
x_train = pd.DataFrame(x_train_scaled, columns=x_train.columns, index=x_train.index)
x_val = pd.DataFrame(x_val_scaled, columns=x_val.columns, index=x_val.index)
```



[62]
```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(random_state=42)
classifier.fit(x_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Predicting the Test set results

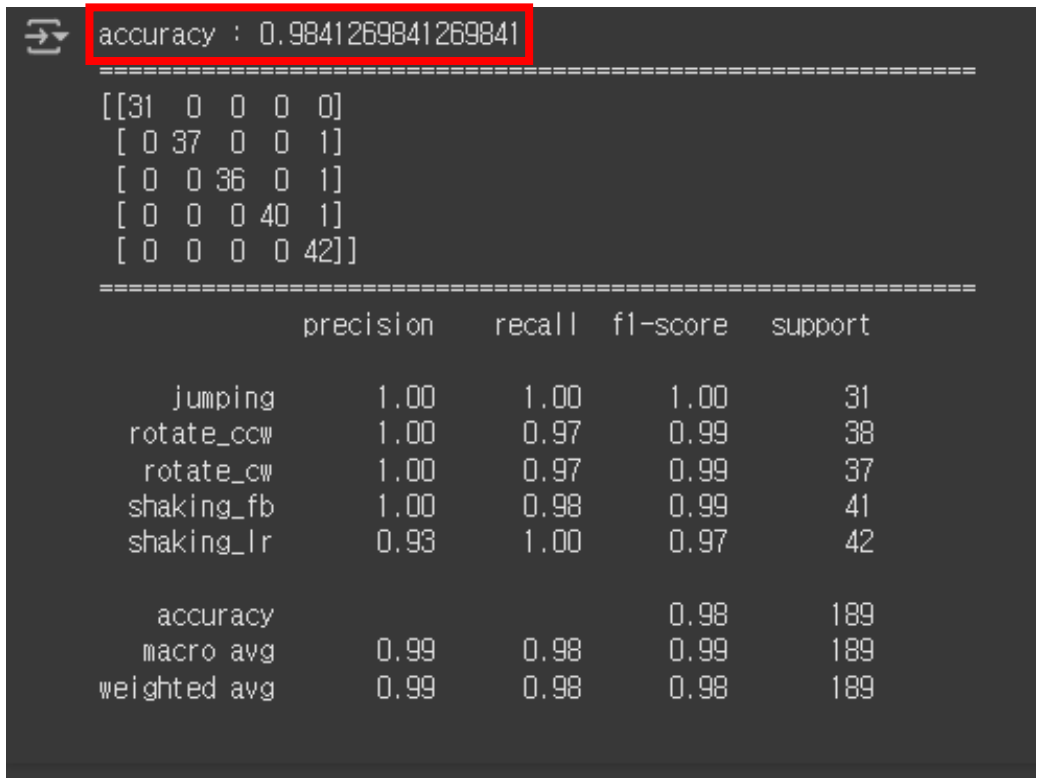
[63]
```
y_pred = classifier.predict(x_val)
```

Evaluate

[64]
```
#평가
from sklearn.metrics import *
print('accuracy :',accuracy_score(y_val, y_pred))
print('='*60)
print(confusion_matrix(y_val, y_pred))
print('='*60)
print(classification_report(y_val, y_pred))
```

랜덤 포레스트 모델 훈련

- 모델의 정확도 평가



```
accuracy : 0.9841269841269841
============================================================
[[31  0  0  0  0]
 [ 0 37  0  0  1]
 [ 0  0 36  0  1]
 [ 0  0  0 40  1]
 [ 0  0  0  0 42]]
============================================================
              precision    recall  f1-score   support

     jumping       1.00      1.00      1.00        31
  rotate_ccw       1.00      0.97      0.99        38
   rotate_cw       1.00      0.97      0.99        37
  shaking_fb       1.00      0.98      0.99        41
  shaking_lr       0.93      1.00      0.97        42

    accuracy                           0.98       189
   macro avg       0.99      0.98      0.99       189
weighted avg       0.99      0.98      0.98       189
```

-> 랜덤 포레스트 모델 훈련 결과: 정확도 약 98.4%

변수 중요도

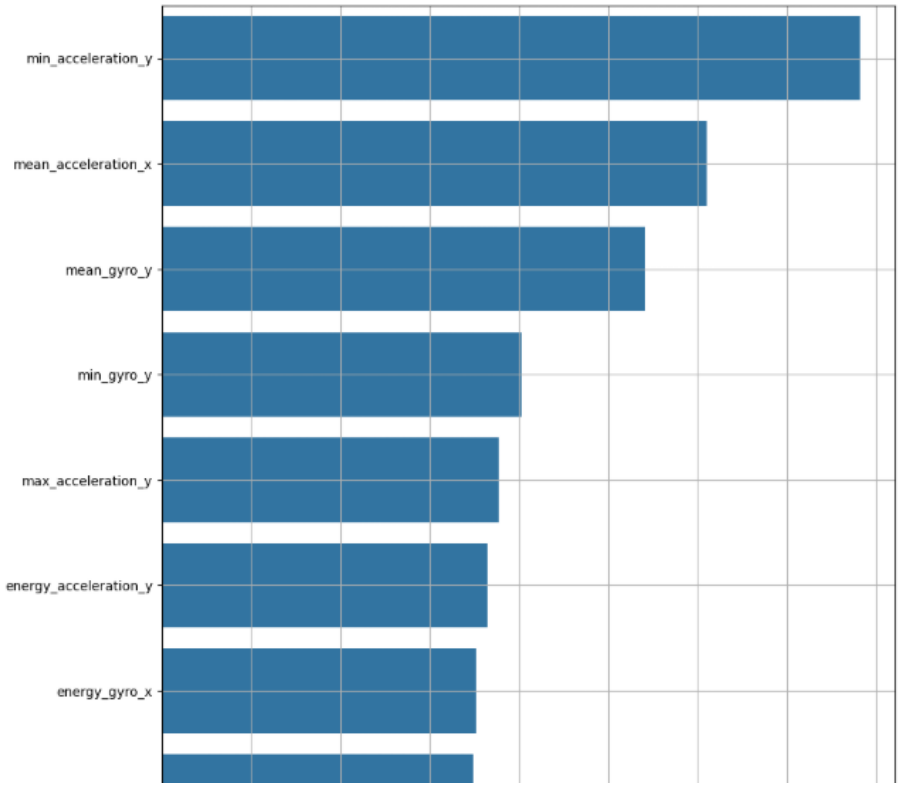
- 변수 중요도 추출

```
[25]  1 # 변수 중요도 추출
      2 r = plot_feature_importance(classifier.feature_importances_, list(x_train), False)
      3 r
```

	feature_name	feature_importance
0	min_acceleration_y	0.078234
1	mean_acceleration_x	0.061049
2	mean_gyro_y	0.054099
3	min_gyro_y	0.040231
4	max_acceleration_y	0.037728
...	...	...
64	meanFreq_acceleration_y	0.000580
65	meanFreq_gyro_z	0.000563
66	meanFreq_gyro_x	0.000347
67	meanFreq_acceleration_x	0.000331
68	meanFreq_gyro_y	0.000243

69 rows × 2 columns

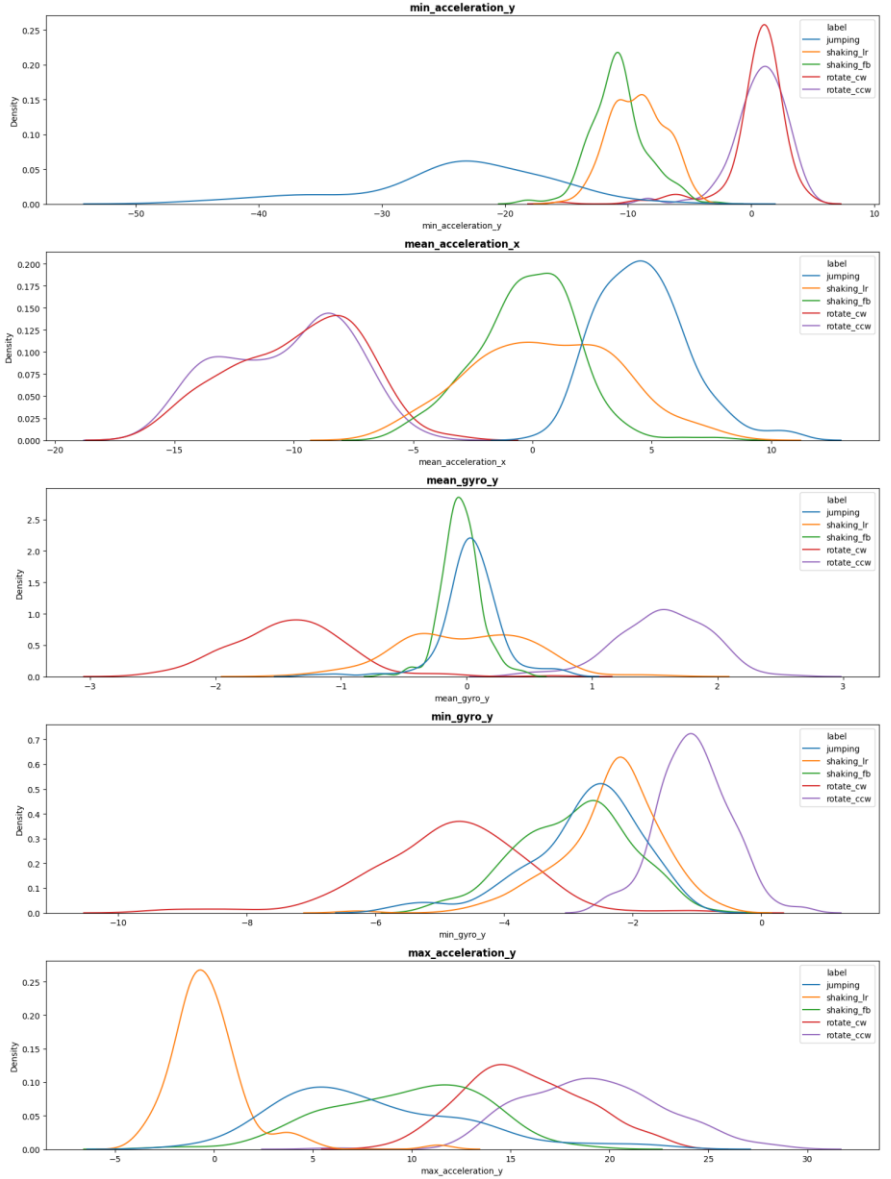
- 변수 중요도를 그래프로 표현



변수 중요도

- 중요도 기반 feature 분석

```python
# 중요도 상위 5개 정보에 대한 분석
fig, ax = plt.subplots(5, figsize=(15,20))
for i in range(5):
    var = r.iloc[i, 0]
    sns.kdeplot(x=var, data=data, hue=target, common_norm=False, ax=ax[i])
    ax[i].set_title(f'{var}', fontweight='bold')
plt.tight_layout()
plt.show()
```



각 행동들을 구분하기 위한 변수 중요도(상위 top5)

```python
[26] # 중요도 상위 top5
r.head(5)
```

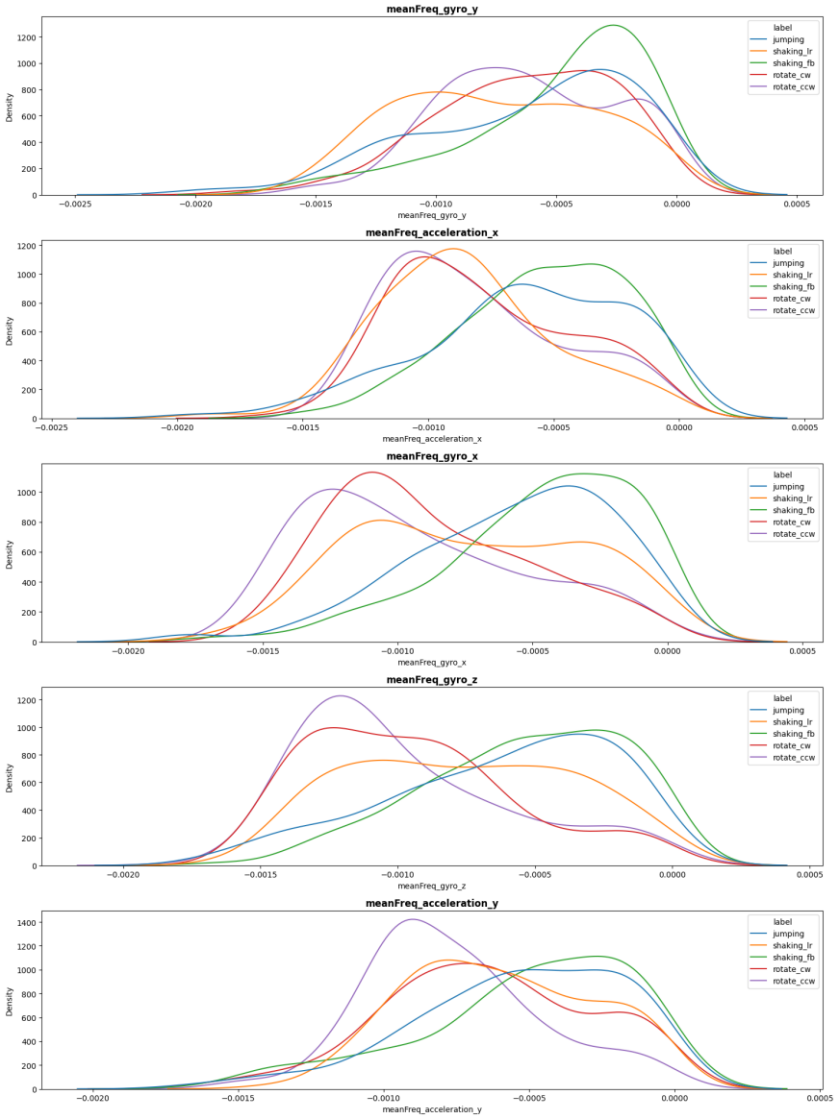
	feature_name	feature_importance
0	min_acceleration_y	0.078234
1	mean_acceleration_x	0.061049
2	mean_gyro_y	0.054099
3	min_gyro_y	0.040231
4	max_acceleration_y	0.037728

그래프를 보면 max_acceleration_y로 shaking_lr(좌우 흔들기)를 다른 행동들과 구분할 수 있을 것으로 보인다. 다른 feature들도 그래프를 볼 때 눈에 보일 정도로 밀집된 부분이 서로 다른 부분이 있으므로 특정 행동들을 구분할 수 있을 것으로 보인다. 다만, 밀집된 부분이 비슷할 경우, feature를 단독으로 사용하기 보다는 여러 feature들을 결합하여 행동을 구분한다면 더 잘 구분할 수 있을 것이다.

변수 중요도

- 중요도 기반 feature 분석

```python
# 중요도 하위 5개 정보에 대한 분석
fig, ax = plt.subplots(5, figsize=(15,20))
for i in range(1,6):
    var = r.iloc[-i, 0]
    sns.kdeplot(x=var, data=data, hue=target, common_norm=False, ax=ax[i-1])
    ax[i-1].set_title(f'{var}', fontweight='bold')

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



각 행동들을 구분하기 위한 변수 중요도(하위 top5)

```python
# 중요도 하위 top5
r.tail(5)
```

	feature_name	feature_importance
64	meanFreq_acceleration_y	0.000580
65	meanFreq_gyro_z	0.000563
66	meanFreq_gyro_x	0.000347
67	meanFreq_acceleration_x	0.000331
68	meanFreq_gyro_y	0.000243

변수 중요도 하위 top5를 보면 대체로 모든 타깃들에 대한 분포도가 매우 비슷하다.
따라서 타깃들을 서로 잘 구분하지 못할 가능성이 크다.

중요도 변수 top 10으로 모델 훈련

- 중요도 변수 top 10으로 모델 훈련을 다시 시도

- 데이터 전처리 및 모델링

```
[70]  1 # 중요도 상위 변수 몇개만 가지고 다시 모델링
      2
      3 # 데이터 분할을 위한 전처리
      4 target = 'label'
      5
      6 x = data.drop(target, axis = 1)
      7 x = x[r.iloc[:10, 0]]
      8 y = data.loc[:, target]
      9
     10 x_train, x_val, y_train, y_val = train_test_split(x, y, test_size = 0.2, random_state=42)
     11
     12 #생성
     13 model = RandomForestClassifier(random_state=42)
     14
     15 #학습
     16 model.fit(x_train, y_train)
     17 pred = model.predict(x_val)
     18
     19 #평가
     20 print('accuracy :',accuracy_score(y_val, pred))
     21 print('='*60)
     22 print(confusion_matrix(y_val, pred))
     23 print('='*60)
     24 print(classification_report(y_val, pred))
```

기존 모델 훈련 결과: 정확도 약 98.4%

10개의 feature로 모델 훈련 결과: 정확도 약 95.2%

```
accuracy : 0.9523809523809523
============================================================
[[26  0  0  3  2]
 [ 0 38  0  0  0]
 [ 0  0 36  1  0]
 [ 1  0  0 39  1]
 [ 0  0  0  1 41]]
============================================================
              precision    recall  f1-score   support

     jumping       0.96      0.84      0.90        31
  rotate_ccw       1.00      1.00      1.00        38
   rotate_cw       1.00      0.97      0.99        37
  shaking_fb       0.89      0.95      0.92        41
  shaking_lr       0.93      0.98      0.95        42

    accuracy                           0.95       189
   macro avg       0.96      0.95      0.95       189
weighted avg       0.95      0.95      0.95       189
```

10개의 feature만으로도 정확도가 높은 것을 알 수 있다.