Data Cleaning and Merging

Here we are going to perform the initial operations of data acquisition from various text files, and merging them together. After that we are going to check for Nan values and if there are any Nan values, we are going to perform suitable operations on them such as imputation or removal. Then we are going to check for outliers (if any) and then we will hedge the outliers.

In [53]:

```
import pandas as pd
import numpy as np
import seaborn as sns
```

In [54]:

```
names = ["Time(ms)", "ankle_acc_hor_forward", "ankle_acc_ver", "ankle_acc_hor_latera
         "upper_leg_acc_hor_forward", "upper_leg_acc_ver", "upper_leg_acc_hor_latera
        "trunk_acc_hor_forward", "trunk_acc_ver", "trunk_acc_hor_lateral", "annotati
df1 = pd.read_csv("../dataset/S01R01.txt", delimiter=' ', header=None, names = names
df1
```

Out[54]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
0	15	70	39	-970	
1	31	70	39	-970	
2	46	60	49	-960	
3	62	60	49	-960	
4	78	50	39	-960	
...	...	...	...	...	
151982	2374734	80	39	-960	
151983	2374750	60	39	-950	
151984	2374765	60	39	-950	
151985	2374781	60	29	-950	
151986	2374796	70	49	-970	

151987 rows × 11 columns

In [55]:

```
names = ["Time(ms)", "ankle_acc_hor_forward", "ankle_acc_ver", "ankle_acc_hor_latera
         "upper_leg_acc_hor_forward", "upper_leg_acc_ver", "upper_leg_acc_hor_latera
        "trunk_acc_hor_forward", "trunk_acc_ver", "trunk_acc_hor_lateral", "annotati
df2 = pd.read_csv("../dataset/S01R02.txt", delimiter=' ', header=None, names = names
df2
```

Out[55]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forw
0	15	40	29	-960	
1	31	70	49	-960	
2	46	60	29	-970	
3	62	60	29	-970	

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forw
4	78	50	19	-960	
...	...	...	...	...	
52090	813921	70	39	-960	
52091	813937	60	49	-960	
52092	813953	60	39	-960	
52093	813968	80	9	-960	
52094	813984	70	49	-970	

52095 rows × 11 columns

In [56]:

```
names = ["Time(ms)", "ankle_acc_hor_forward", "ankle_acc_ver", "ankle_acc_hor_latera
         "upper_leg_acc_hor_forward", "upper_leg_acc_ver", "upper_leg_acc_hor_latera
        "trunk_acc_hor_forward", "trunk_acc_ver", "trunk_acc_hor_lateral", "annotati
df3 = pd.read_csv("../dataset/S02R01.txt", delimiter=' ', header=None, names = names
df4 = pd.read_csv("../dataset/S02R02.txt", delimiter=' ', header=None, names = names
df5 = pd.read_csv("../dataset/S03R01.txt", delimiter=' ', header=None, names = names
df6 = pd.read_csv("../dataset/S03R02.txt", delimiter=' ', header=None, names = names
df7 = pd.read_csv("../dataset/S03R03.txt", delimiter=' ', header=None, names = names
df8 = pd.read_csv("../dataset/S04R01.txt", delimiter=' ', header=None, names = names
df9 = pd.read_csv("../dataset/S05R01.txt", delimiter=' ', header=None, names = names
df10 = pd.read_csv("../dataset/S05R02.txt", delimiter=' ', header=None, names = name
df11 = pd.read_csv("../dataset/S06R01.txt", delimiter=' ', header=None, names = name
df12 = pd.read_csv("../dataset/S06R02.txt", delimiter=' ', header=None, names = name
df13 = pd.read_csv("../dataset/S07R01.txt", delimiter=' ', header=None, names = name
df14 = pd.read_csv("../dataset/S07R02.txt", delimiter=' ', header=None, names = name
df15 = pd.read_csv("../dataset/S08R01.txt", delimiter=' ', header=None, names = name
df16 = pd.read_csv("../dataset/S09R01.txt", delimiter=' ', header=None, names = name
df17 = pd.read_csv("../dataset/S10R01.txt", delimiter=' ', header=None, names = name
```

In [57]:

```
lst = [df1,df2,df3,df4,df5,df6,df7,df8,df9,df10,df11,df12,df13,df14,df15,df16,df17]
df_merged = pd.concat(lst, axis=0)
df_merged
```

Out[57]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
0	15	70	39	-970	
1	31	70	39	-970	
2	46	60	49	-960	
3	62	60	49	-960	
4	78	50	39	-960	
...	...	...	...	...	
193298	3020296	-131	107	-960	
193299	3020312	-121	127	-970	
193300	3020328	-141	117	-960	
193301	3020343	-131	127	-980	

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forw
193302	3020359	-141	0	0	

1917887 rows × 11 columns

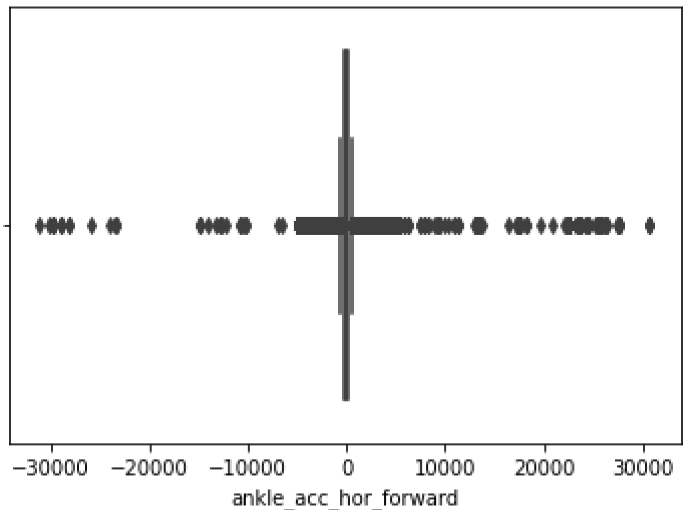
In [58]:
```
df_merged.isna().sum()
```

Out[58]:
```
Time(ms)                   0
ankle_acc_hor_forward      0
ankle_acc_ver              0
ankle_acc_hor_lateral      0
upper_leg_acc_hor_forward  0
upper_leg_acc_ver          0
upper_leg_acc_hor_lateral  0
trunk_acc_hor_forward      0
trunk_acc_ver              0
trunk_acc_hor_lateral      0
annotation                 0
dtype: int64
```

In [59]:
```
sns.boxplot(df_merged['ankle_acc_hor_forward'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

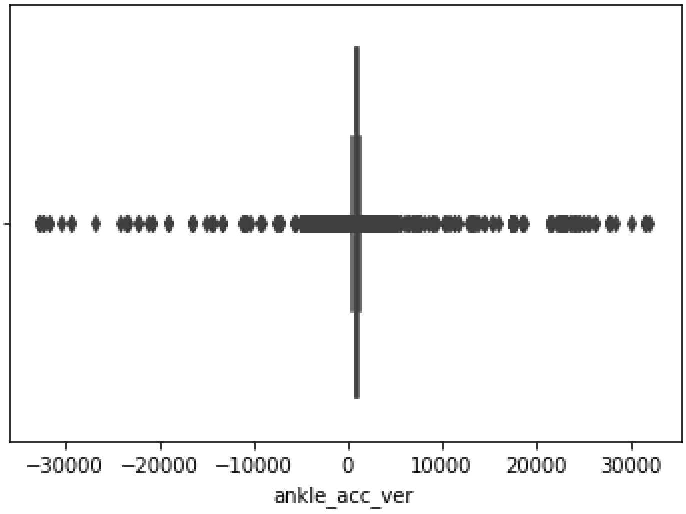
Out[59]: <AxesSubplot:xlabel='ankle_acc_hor_forward'>



In [60]:
```
sns.boxplot(df_merged['ankle_acc_ver'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[60]: <AxesSubplot:xlabel='ankle_acc_ver'>

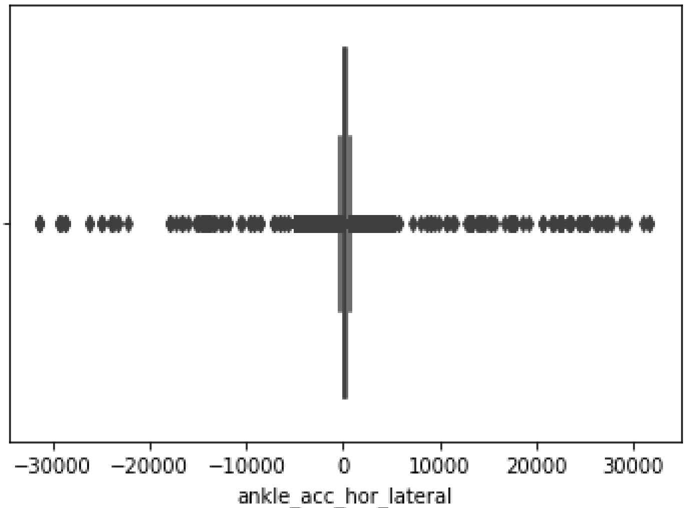


In [61]:

```
sns.boxplot(df_merged['ankle_acc_hor_lateral'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[61]: <AxesSubplot:xlabel='ankle_acc_hor_lateral'>

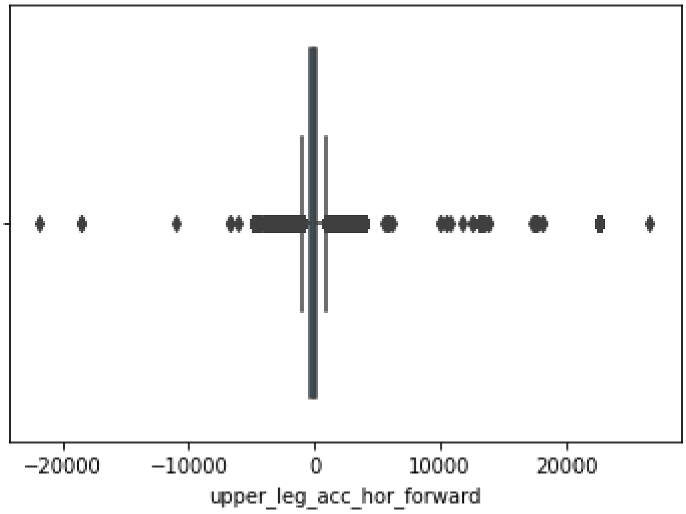


In [62]:

```
sns.boxplot(df_merged['upper_leg_acc_hor_forward'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

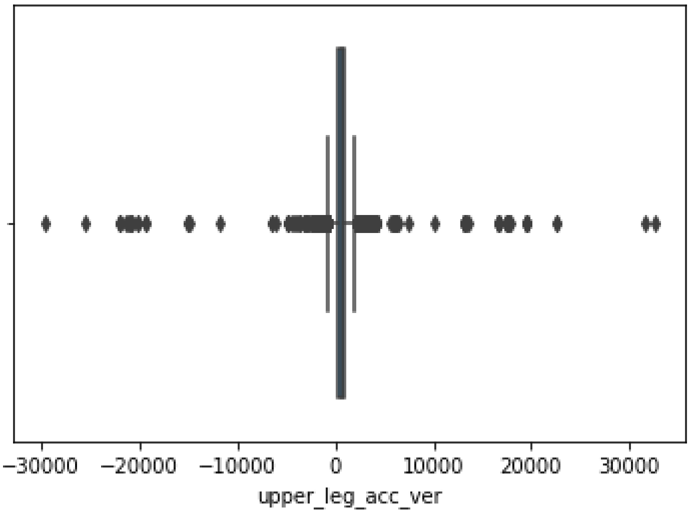
Out[62]: <AxesSubplot:xlabel='upper_leg_acc_hor_forward'>



In [63]:
```
sns.boxplot(df_merged['upper_leg_acc_ver'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

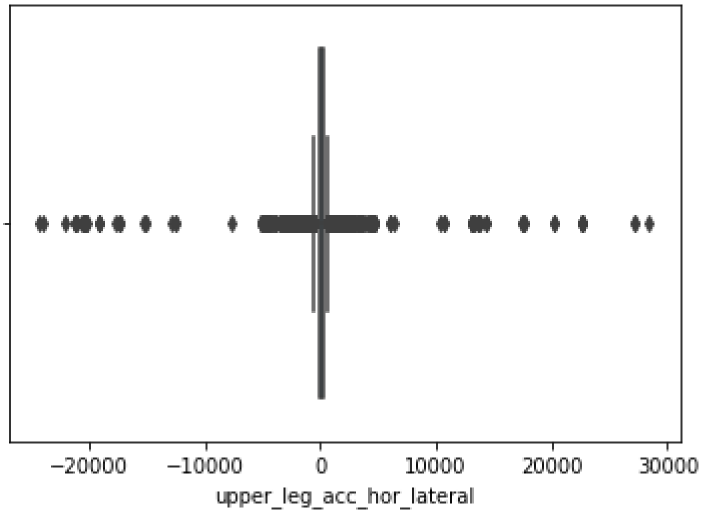
Out[63]: <AxesSubplot:xlabel='upper_leg_acc_ver'>



In [64]:
```
sns.boxplot(df_merged['upper_leg_acc_hor_lateral'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[64]: <AxesSubplot:xlabel='upper_leg_acc_hor_lateral'>

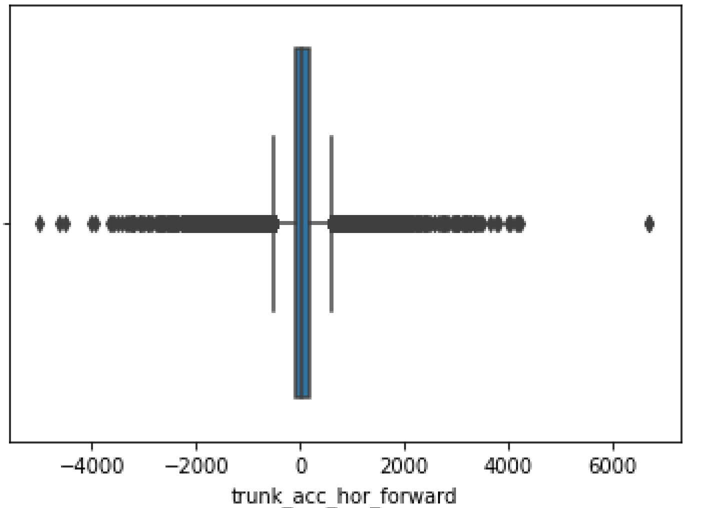


In [65]:

```
sns.boxplot(df_merged['trunk_acc_hor_forward'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[65]: <AxesSubplot:xlabel='trunk_acc_hor_forward'>

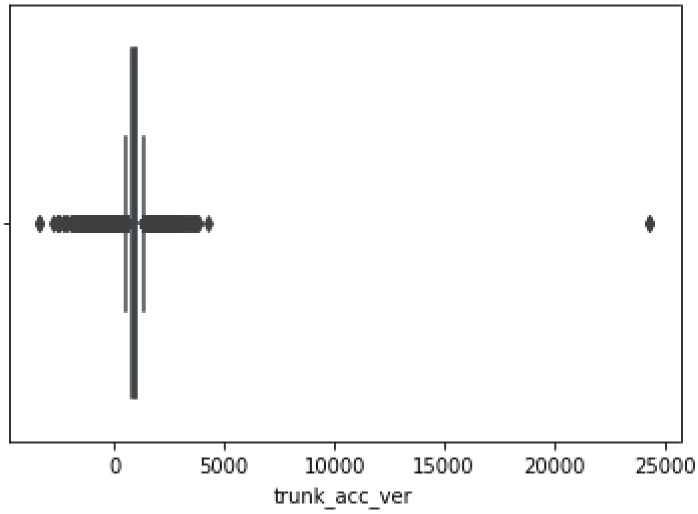


In [66]:

```
sns.boxplot(df_merged['trunk_acc_ver'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

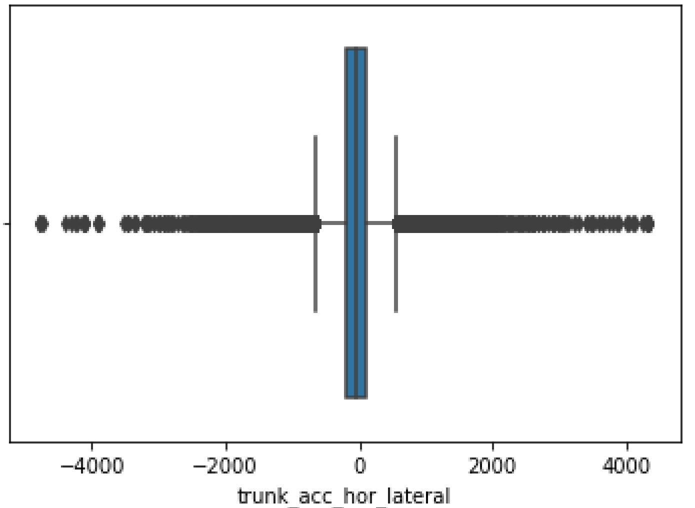
Out[66]: <AxesSubplot:xlabel='trunk_acc_ver'>



In [67]:
```
sns.boxplot(df_merged['trunk_acc_hor_lateral'])
```

```
C:\Users\Dev\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[67]: <AxesSubplot:xlabel='trunk_acc_hor_lateral'>



In [68]:
```
q1 = df_merged['trunk_acc_ver'].quantile(0.25)
q3 = df_merged['trunk_acc_ver'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged[(df_merged['trunk_acc_ver'] > lower) & (df_merged['trunk_acc_
df_merged1
```

Out[68]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
15865	247906	60	39	-950	
15866	247921	60	49	-970	
15867	247937	60	29	-960	
15868	247953	60	49	-960	
15869	247968	50	49	-980	

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
...	...	...	...	...	
161296	2520265	-555	882	425	
161297	2520281	-535	892	376	
161298	2520296	-585	823	346	
161299	2520312	-282	980	198	
161300	2520328	-252	1039	138	

1487248 rows × 11 columns

In [69]:

```
q1 = df_merged1['ankle_acc_hor_forward'].quantile(0.25)
q3 = df_merged1['ankle_acc_hor_forward'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['ankle_acc_hor_forward'] > lower) & (df_merged1[
df_merged1
```

Out[69]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
15865	247906	60	39	-950	
15866	247921	60	49	-970	
15867	247937	60	29	-960	
15868	247953	60	49	-960	
15869	247968	50	49	-980	
...	...	...	...	...	
161296	2520265	-555	882	425	
161297	2520281	-535	892	376	
161298	2520296	-585	823	346	
161299	2520312	-282	980	198	
161300	2520328	-252	1039	138	

1413952 rows × 11 columns

In [70]:

```
q1 = df_merged1['ankle_acc_ver'].quantile(0.25)
q3 = df_merged1['ankle_acc_ver'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['ankle_acc_ver'] > lower) & (df_merged1['ankle_a
df_merged1
```

Out[70]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
28160	440015	-141	1000	326	

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forw
29019	453437	-131	941	346	
29023	453500	-80	1009	306	
29024	453515	-50	980	346	
29025	453531	-90	970	346	
...	...	...	...	...	
161296	2520265	-555	882	425	
161297	2520281	-535	892	376	
161298	2520296	-585	823	346	
161299	2520312	-282	980	198	
161300	2520328	-252	1039	138	

1207193 rows × 11 columns

In [71]:

```
q1 = df_merged1['ankle_acc_hor_lateral'].quantile(0.25)
q3 = df_merged1['ankle_acc_hor_lateral'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['ankle_acc_hor_lateral'] > lower) & (df_merged1[
df_merged1
```

Out[71]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forw
28160	440015	-141	1000	326	
29019	453437	-131	941	346	
29023	453500	-80	1009	306	
29024	453515	-50	980	346	
29025	453531	-90	970	346	
...	...	...	...	...	
161296	2520265	-555	882	425	
161297	2520281	-535	892	376	
161298	2520296	-585	823	346	
161299	2520312	-282	980	198	
161300	2520328	-252	1039	138	

1186730 rows × 11 columns

In [72]:

```
q1 = df_merged1['upper_leg_acc_hor_forward'].quantile(0.25)
q3 = df_merged1['upper_leg_acc_hor_forward'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
```

```
df_merged1 = df_merged1[(df_merged1['upper_leg_acc_hor_forward'] > lower) & (df_merg
df_merged1
```

Out[72]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
28160	440015	-141	1000	326	
29019	453437	-131	941	346	
29023	453500	-80	1009	306	
29024	453515	-50	980	346	
29025	453531	-90	970	346	
...	...	...	...	...	
161296	2520265	-555	882	425	
161297	2520281	-535	892	376	
161298	2520296	-585	823	346	
161299	2520312	-282	980	198	
161300	2520328	-252	1039	138	

1185477 rows × 11 columns

In [73]:

```
q1 = df_merged1['upper_leg_acc_ver'].quantile(0.25)
q3 = df_merged1['upper_leg_acc_ver'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['upper_leg_acc_ver'] > lower) & (df_merged1['upp
df_merged1
```

Out[73]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
28160	440015	-141	1000	326	
29019	453437	-131	941	346	
29023	453500	-80	1009	306	
29024	453515	-50	980	346	
29025	453531	-90	970	346	
...	...	...	...	...	
161296	2520265	-555	882	425	
161297	2520281	-535	892	376	
161298	2520296	-585	823	346	
161299	2520312	-282	980	198	
161300	2520328	-252	1039	138	

1185177 rows × 11 columns

In [74]:
```
q1 = df_merged1['upper_leg_acc_hor_lateral'].quantile(0.25)
q3 = df_merged1['upper_leg_acc_hor_lateral'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['upper_leg_acc_hor_lateral'] > lower) & (df_merg
df_merged1
```

Out[74]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
28160	440015	-141	1000	326	
29019	453437	-131	941	346	
29023	453500	-80	1009	306	
29024	453515	-50	980	346	
29025	453531	-90	970	346	
...	...	...	...	...	
156599	2446875	-212	1000	376	
156600	2446890	-232	970	356	
156601	2446906	-232	970	356	
156602	2446921	-252	921	346	
156605	2446968	-232	862	366	

1109989 rows × 11 columns

In [75]:
```
q1 = df_merged1['trunk_acc_hor_forward'].quantile(0.25)
q3 = df_merged1['trunk_acc_hor_forward'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['trunk_acc_hor_forward'] > lower) & (df_merged1[
df_merged1
```

Out[75]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
29039	453750	-70	990	346	
29040	453765	-90	990	336	
29041	453781	-101	980	356	
29042	453796	-40	970	326	
29043	453812	-101	1000	316	
...	...	...	...	...	
156599	2446875	-212	1000	376	
156600	2446890	-232	970	356	
156601	2446906	-232	970	356	
156602	2446921	-252	921	346	

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
156605	2446968	-232	862	366	

1056469 rows × 11 columns

In [76]:

```
q1 = df_merged1['trunk_acc_hor_lateral'].quantile(0.25)
q3 = df_merged1['trunk_acc_hor_lateral'].quantile(0.75)
iqr = q3 - q1
upper = q3 + 1.5 * iqr
lower = q1 - 1.5 * iqr
df_merged1 = df_merged1[(df_merged1['trunk_acc_hor_lateral'] > lower) & (df_merged1[
df_merged1
```

Out[76]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forv
29042	453796	-40	970	326	
29051	453937	-60	990	316	
29056	454015	-111	980	346	
29057	454031	-111	980	346	
29058	454046	-60	1009	346	
...	...	...	...	...	
156599	2446875	-212	1000	376	
156600	2446890	-232	970	356	
156601	2446906	-232	970	356	
156602	2446921	-252	921	346	
156605	2446968	-232	862	366	

1021881 rows × 11 columns

In [80]:

```
df_merged1.to_csv("../dataset/merged.csv", index = False)
```

Data Visualization, Correlation Plots

Here we are going to perform data visualization in order to see if there is a significant difference amongst the groups with respect to each feature. After that we will perform correlation in order to check for multicollinearity

In [69]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [70]:
```
df = pd.read_csv("../dataset/merged.csv")
df
```

Out[70]:

	Time(ms)	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_fo
0	453796	-40	970	326	
1	453937	-60	990	316	
2	454015	-111	980	346	
3	454031	-111	980	346	
4	454046	-60	1009	346	
...	...	...	...	...	
1021876	2446875	-212	1000	376	
1021877	2446890	-232	970	356	
1021878	2446906	-232	970	356	
1021879	2446921	-252	921	346	
1021880	2446968	-232	862	366	

1021881 rows × 11 columns

In [71]:
```
df.drop(columns="Time(ms)", inplace=True)
df
```

Out[71]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

1021881 rows × 10 columns

In [72]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1021881 entries, 0 to 1021880
Data columns (total 10 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   ankle_acc_hor_forward      1021881 non-null  int64
 1   ankle_acc_ver              1021881 non-null  int64
 2   ankle_acc_hor_lateral      1021881 non-null  int64
 3   upper_leg_acc_hor_forward  1021881 non-null  int64
 4   upper_leg_acc_ver          1021881 non-null  int64
 5   upper_leg_acc_hor_lateral  1021881 non-null  int64
 6   trunk_acc_hor_forward      1021881 non-null  int64
 7   trunk_acc_ver              1021881 non-null  int64
 8   trunk_acc_hor_lateral      1021881 non-null  int64
 9   annotation                 1021881 non-null  int64
dtypes: int64(10)
memory usage: 78.0 MB
```

In [73]:

```
df_0 = df[df["annotation"] == 0]
df_0
```

Out[73]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	uppe
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
985130	-373	931	306	854	
985131	-363	911	306	872	
985132	-353	950	297	845	
985133	-363	931	316	863	
985134	-393	931	306	854	

289021 rows × 10 columns

In [74]:
```
df_1 = df[df["annotation"] == 1]
df_1
```

Out[74]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
17828	-30	990	326	-45	
17829	-30	1000	356	-18	
17830	-20	990	336	18	
17831	-20	1000	316	36	
17832	0	990	316	36	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

656131 rows × 10 columns

In [75]:
```
df_2 = df[df["annotation"] == 2]
df_2
```

Out[75]:

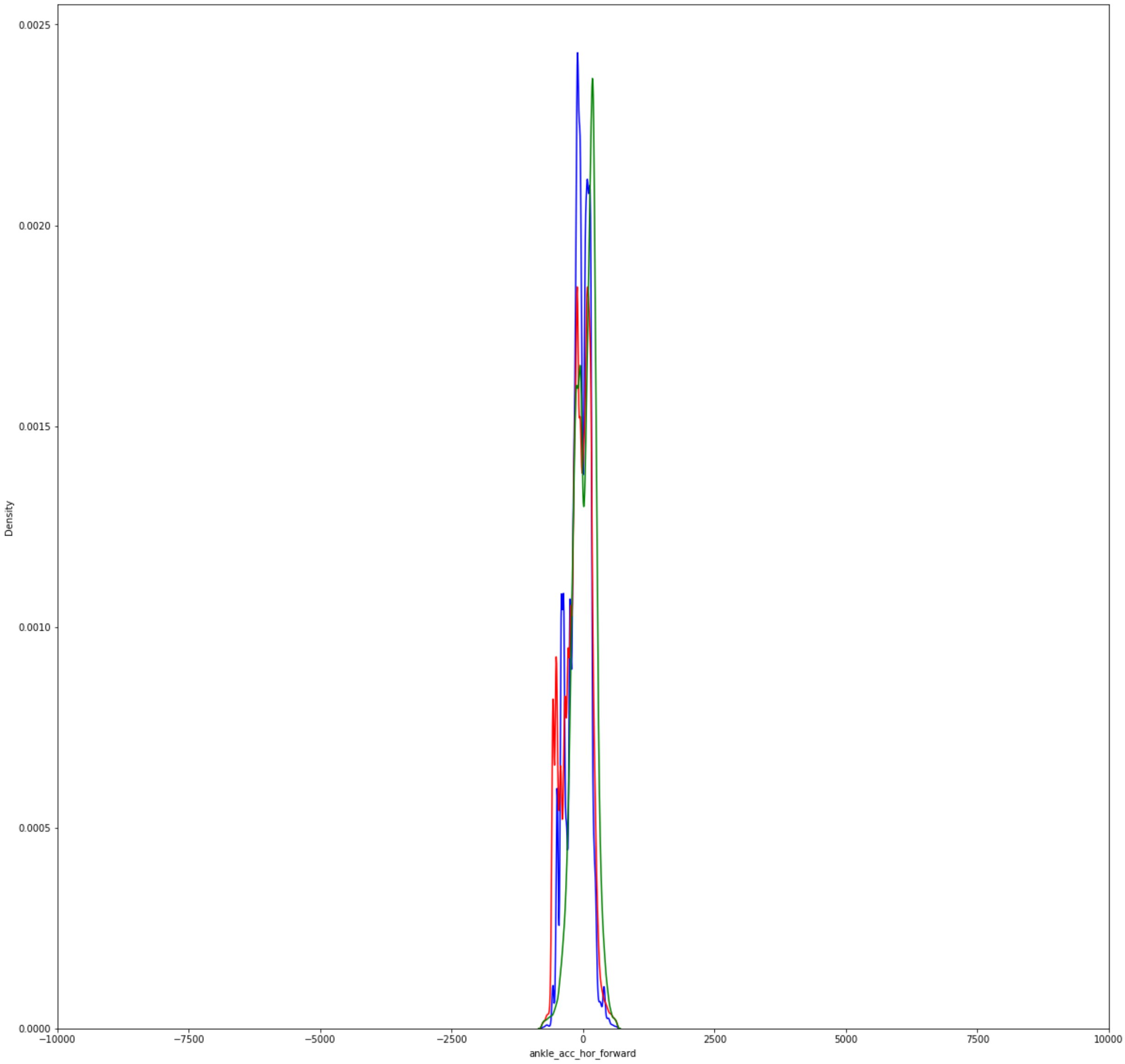
	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	uppe
35405	-202	960	425	445	
35406	-20	1127	386	536	
35407	-80	1058	475	672	
35408	-90	1156	534	836	
35409	-333	931	514	-290	
...	...	...	...	...	
929559	161	1029	207	-309	
929560	262	1000	257	-254	
929561	151	1019	188	-254	
929562	171	1029	178	-254	
929563	181	1058	178	-254	

76729 rows × 10 columns

In [76]:
```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['ankle_acc_hor_forward'],  color="blue", label="df_0_ankle_acc
ax = sns.kdeplot(df_1['ankle_acc_hor_forward'],  color="red", label="df_1_ankle_acc_
```

```
ax = sns.kdeplot(df_2['ankle_acc_hor_forward'],  color="green", label="df_2_ankle_ac
ax.set(xlim=(-10000, 10000))
```

Out[76]: [(-10000.0, 10000.0)]

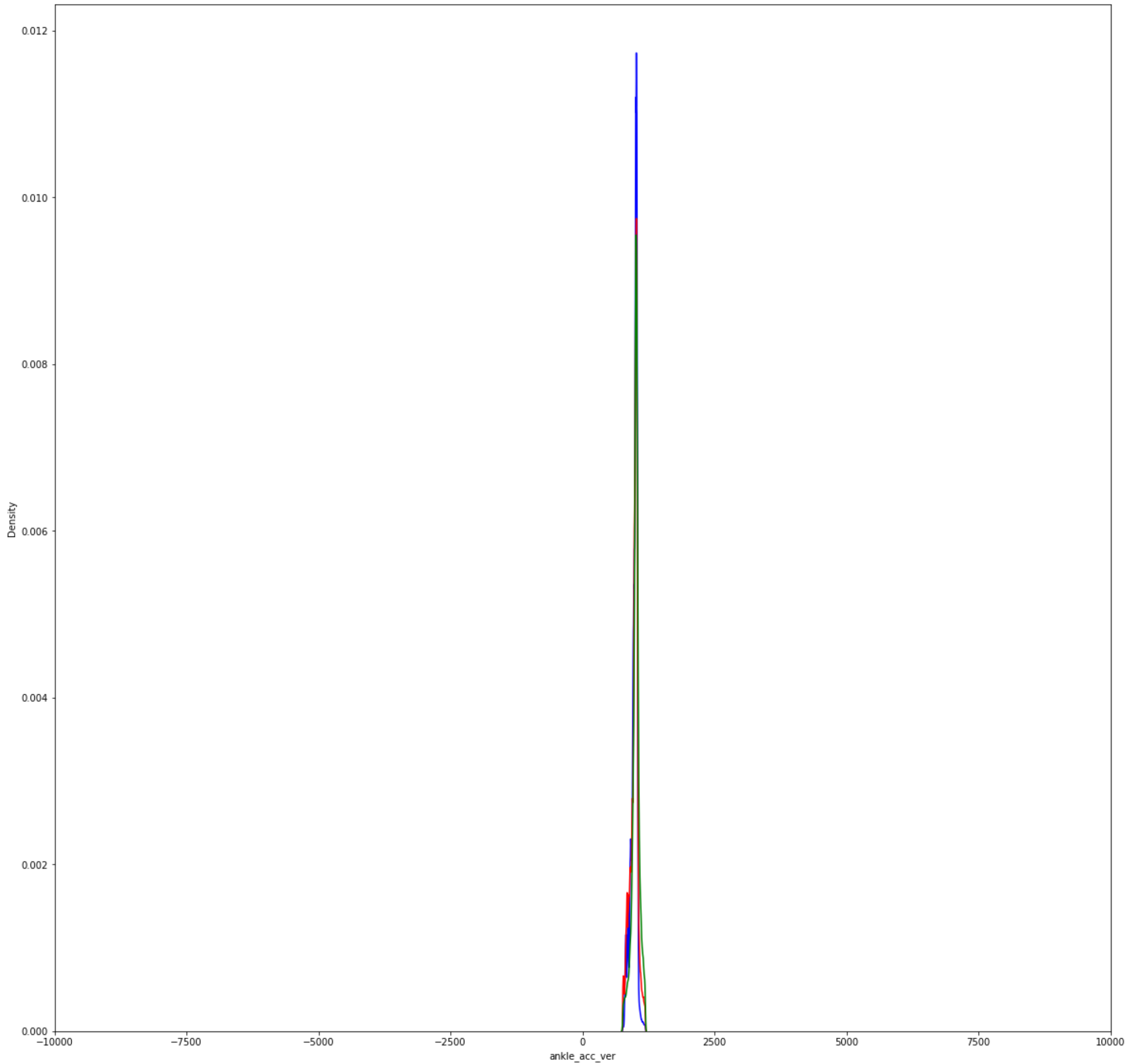


In [ ]:

In [77]:

```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['ankle_acc_ver'],  color="blue", label="df_0_ankle_acc_hor_for
ax = sns.kdeplot(df_1['ankle_acc_ver'],  color="red", label="df_1_ankle_acc_hor_forw
ax = sns.kdeplot(df_2['ankle_acc_ver'],  color="green", label="df_2_ankle_acc_hor_fo
ax.set(xlim=(-10000, 10000))
```

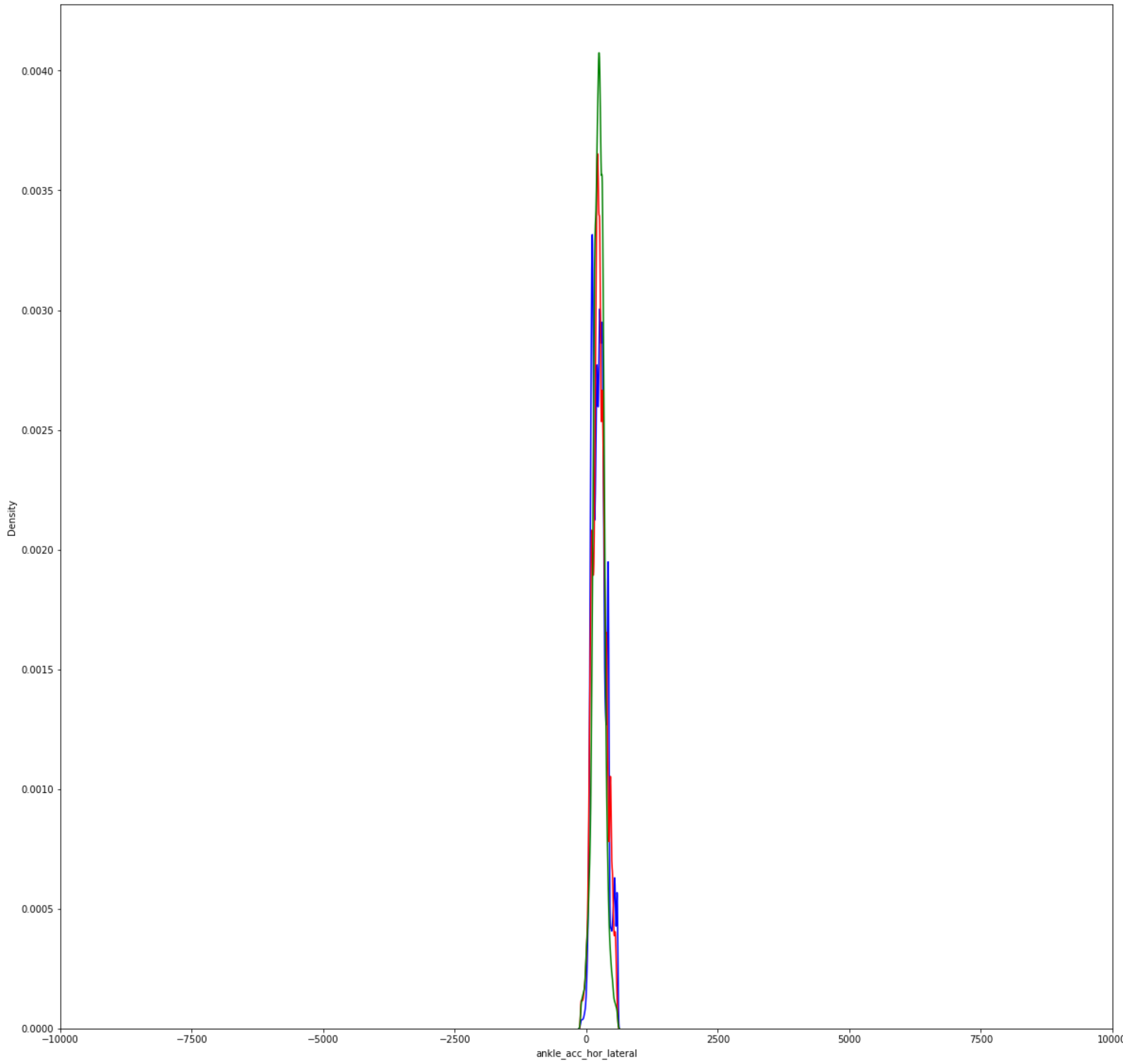
Out[77]: [(-10000.0, 10000.0)]



In [78]:

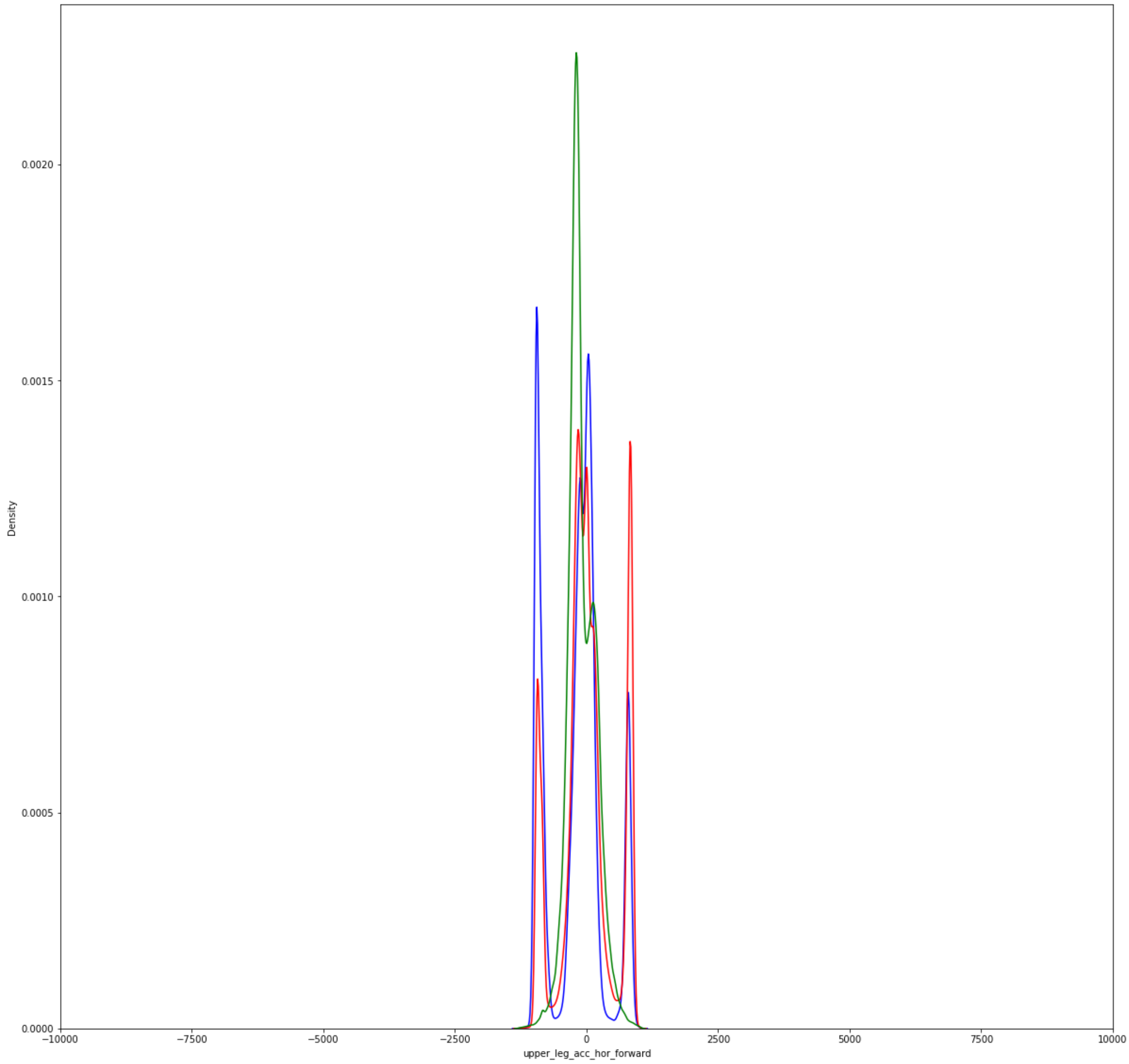
```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['ankle_acc_hor_lateral'],  color="blue", label="df_0_ankle_acc
ax = sns.kdeplot(df_1['ankle_acc_hor_lateral'],  color="red", label="df_1_ankle_acc_
ax = sns.kdeplot(df_2['ankle_acc_hor_lateral'],  color="green", label="df_2_ankle_ac
ax.set(xlim=(-10000, 10000))
```

Out[78]: [(-10000.0, 10000.0)]



In [79]:
```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['upper_leg_acc_hor_forward'],  color="blue", label="df_0_ankle
ax = sns.kdeplot(df_1['upper_leg_acc_hor_forward'],  color="red", label="df_1_ankle_
ax = sns.kdeplot(df_2['upper_leg_acc_hor_forward'],  color="green", label="df_2_ankl
ax.set(xlim=(-10000, 10000))
```

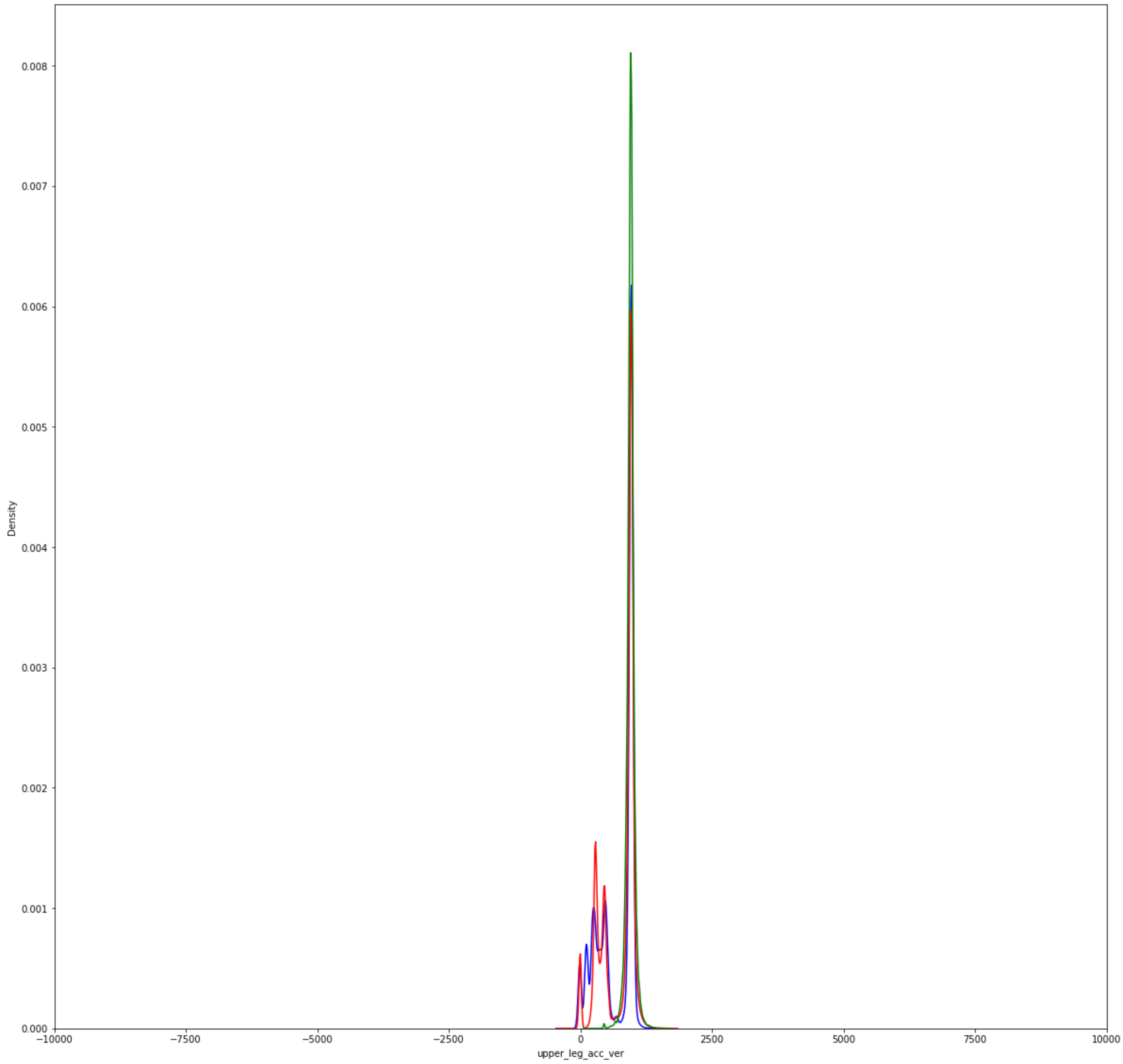
Out[79]: `[(-10000.0, 10000.0)]`



In [80]:

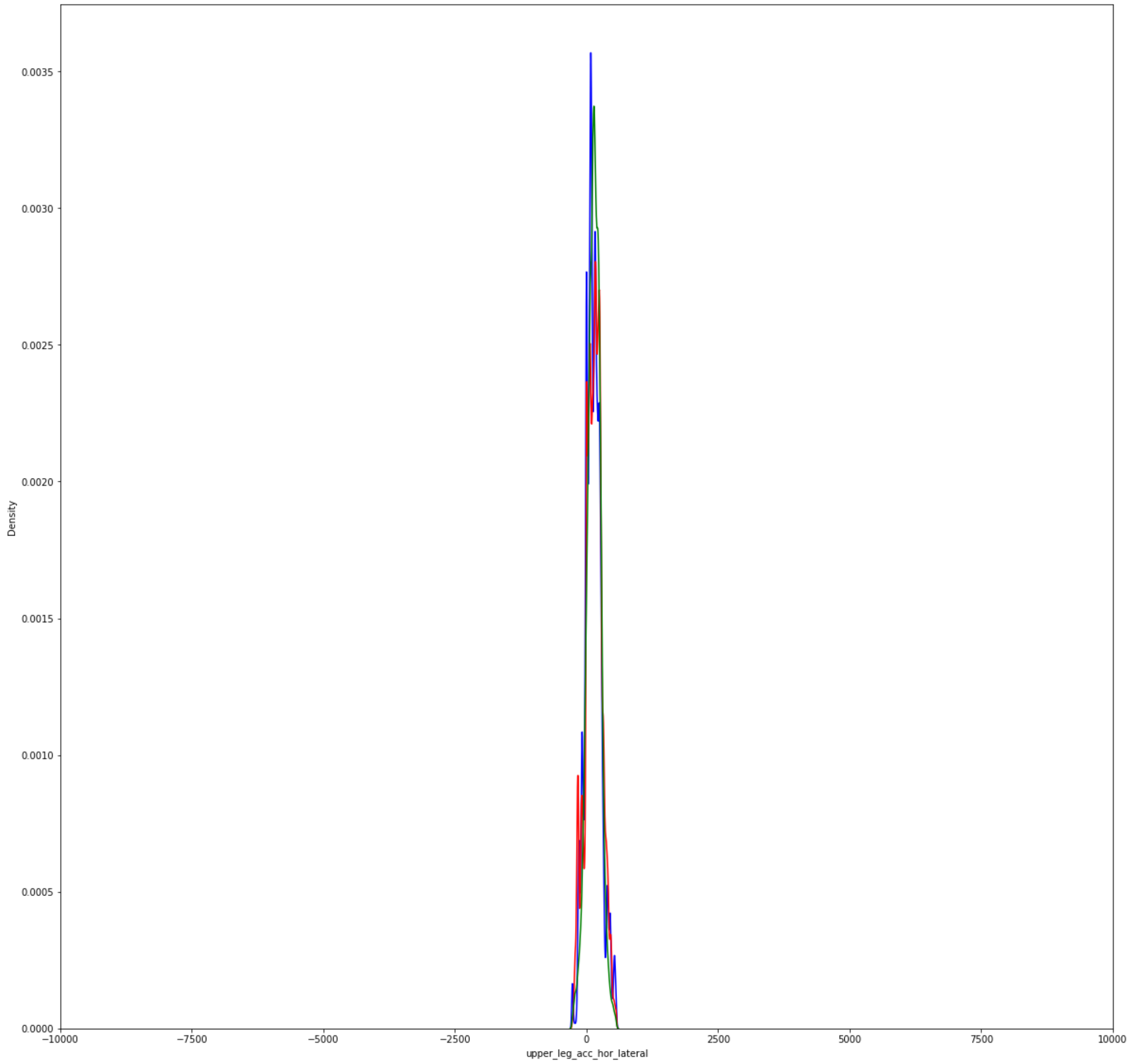
```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['upper_leg_acc_ver'],  color="blue", label="df_0_ankle_acc_hor
ax = sns.kdeplot(df_1['upper_leg_acc_ver'],  color="red", label="df_1_ankle_acc_hor_
ax = sns.kdeplot(df_2['upper_leg_acc_ver'],  color="green", label="df_2_ankle_acc_ho
ax.set(xlim=(-10000, 10000))
```

Out[80]: [(-10000.0, 10000.0)]



In [81]:
```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['upper_leg_acc_hor_lateral'],  color="blue", label="df_0_ankle
ax = sns.kdeplot(df_1['upper_leg_acc_hor_lateral'],  color="red", label="df_1_ankle_
ax = sns.kdeplot(df_2['upper_leg_acc_hor_lateral'],  color="green", label="df_2_ankl
ax.set(xlim=(-10000, 10000))
```

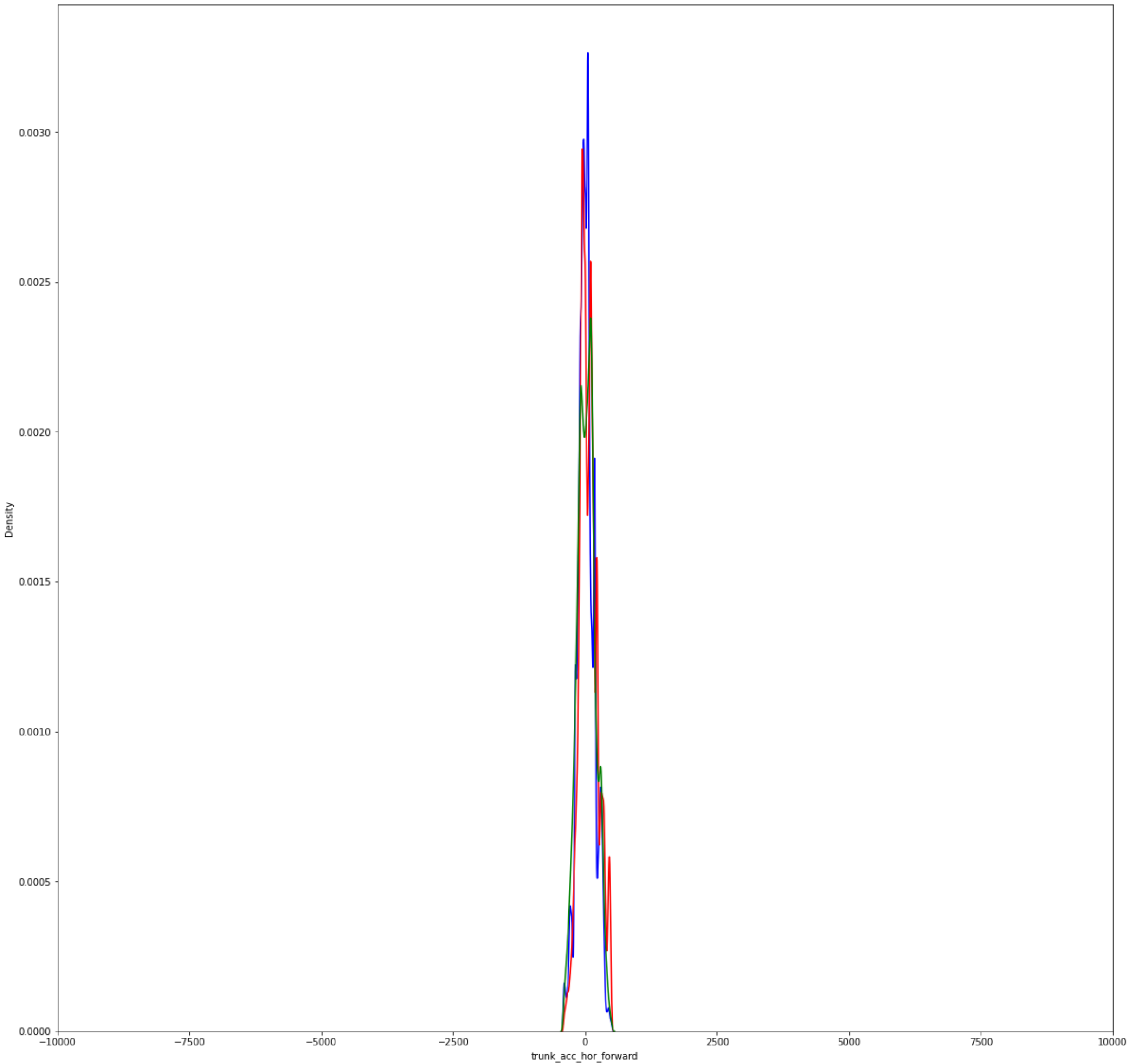
Out[81]: [(-10000.0, 10000.0)]



In [82]:

```
plt.figure(figsize= (20,20))
ax = sns.kdeplot(df_0['trunk_acc_hor_forward'],  color="blue", label="df_0_ankle_acc
ax = sns.kdeplot(df_1['trunk_acc_hor_forward'],  color="red", label="df_1_ankle_acc_
ax = sns.kdeplot(df_2['trunk_acc_hor_forward'],  color="green", label="df_2_ankle_ac
ax.set(xlim=(-10000, 10000))
```

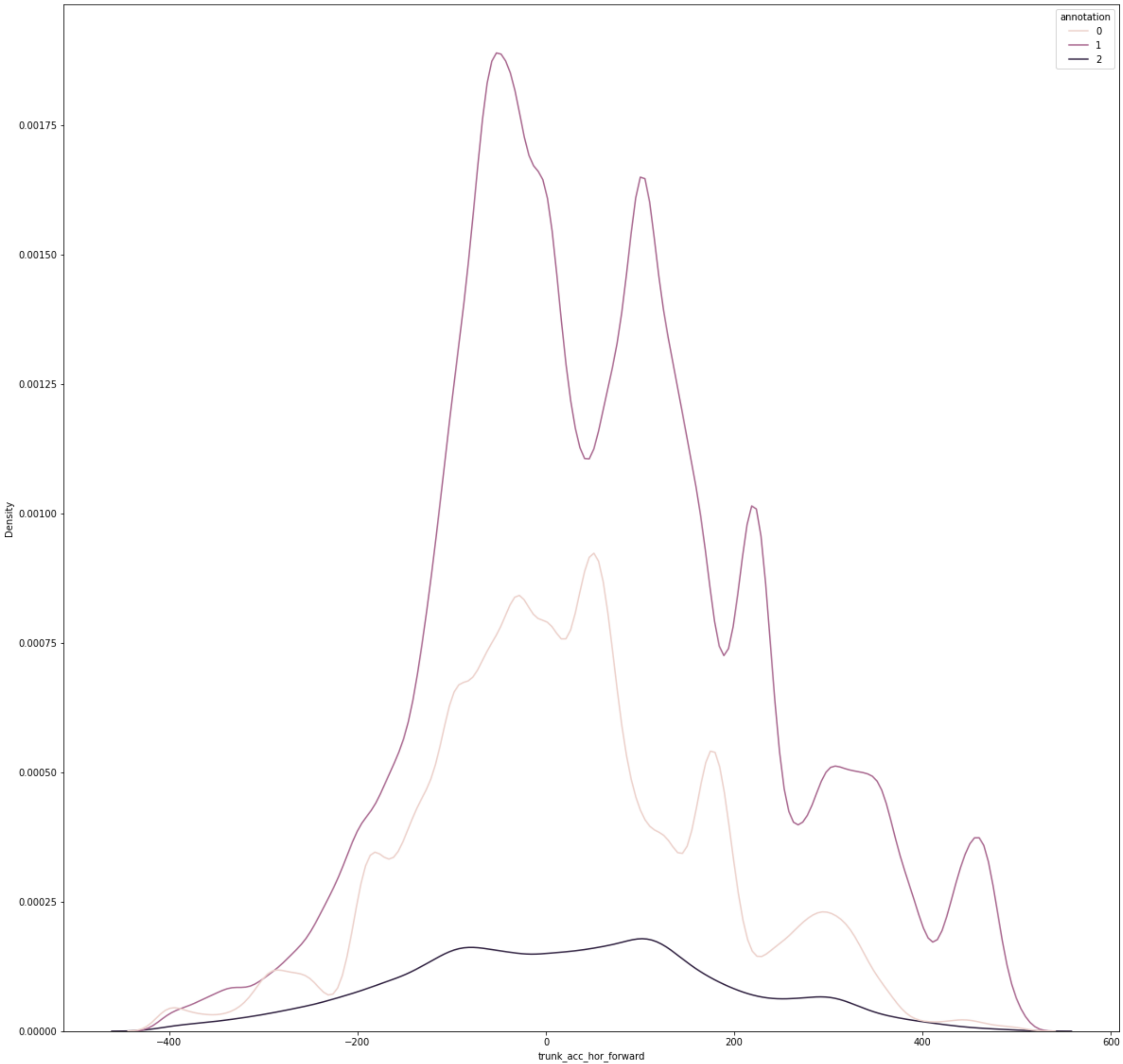
Out[82]: [(-10000.0, 10000.0)]



In [83]:

```
plt.figure(figsize= (20,20))
sns.kdeplot(df['trunk_acc_hor_forward'], hue= df['annotation'])
```

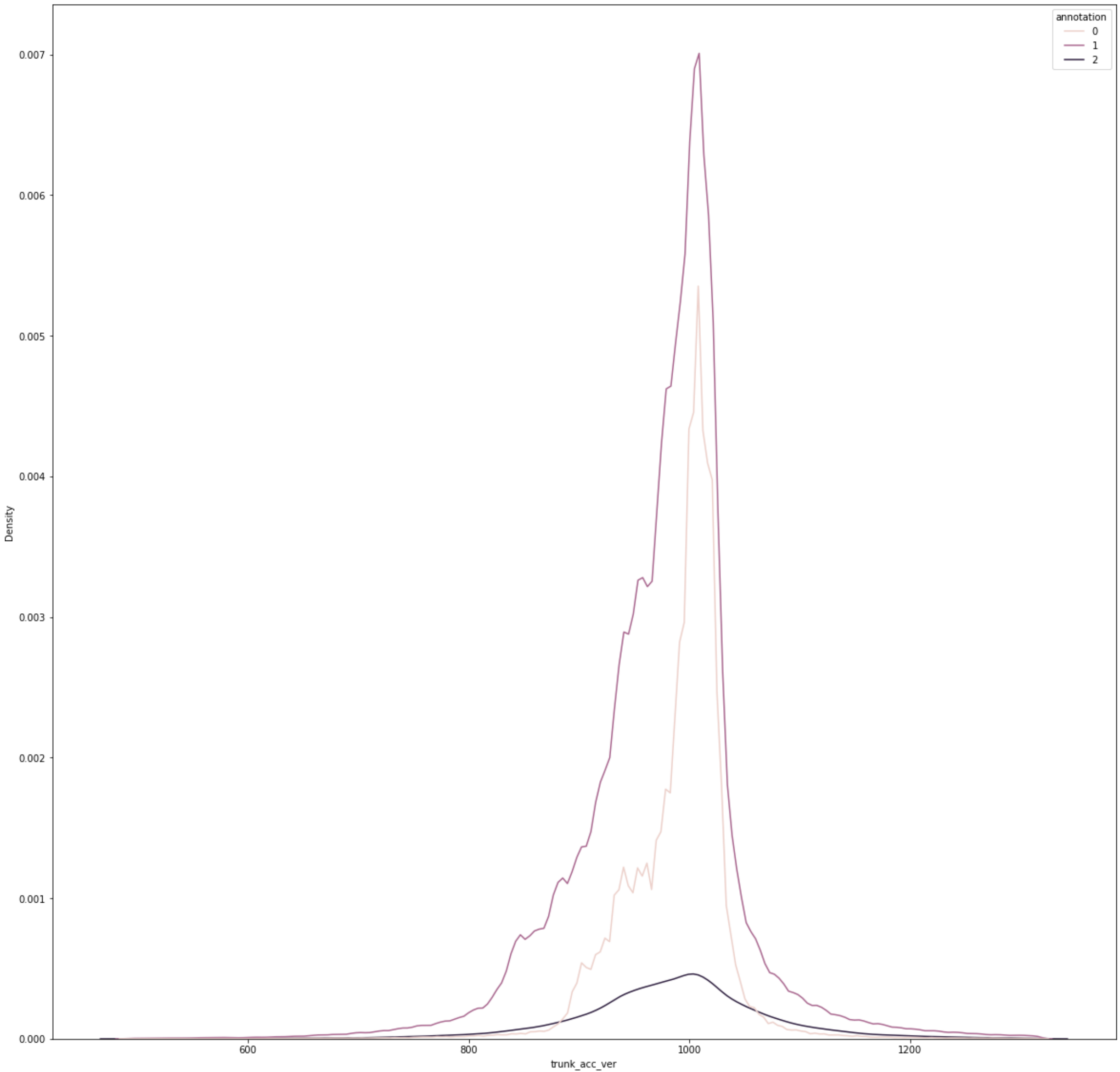
Out[83]: <AxesSubplot:xlabel='trunk_acc_hor_forward', ylabel='Density'>



In [84]:

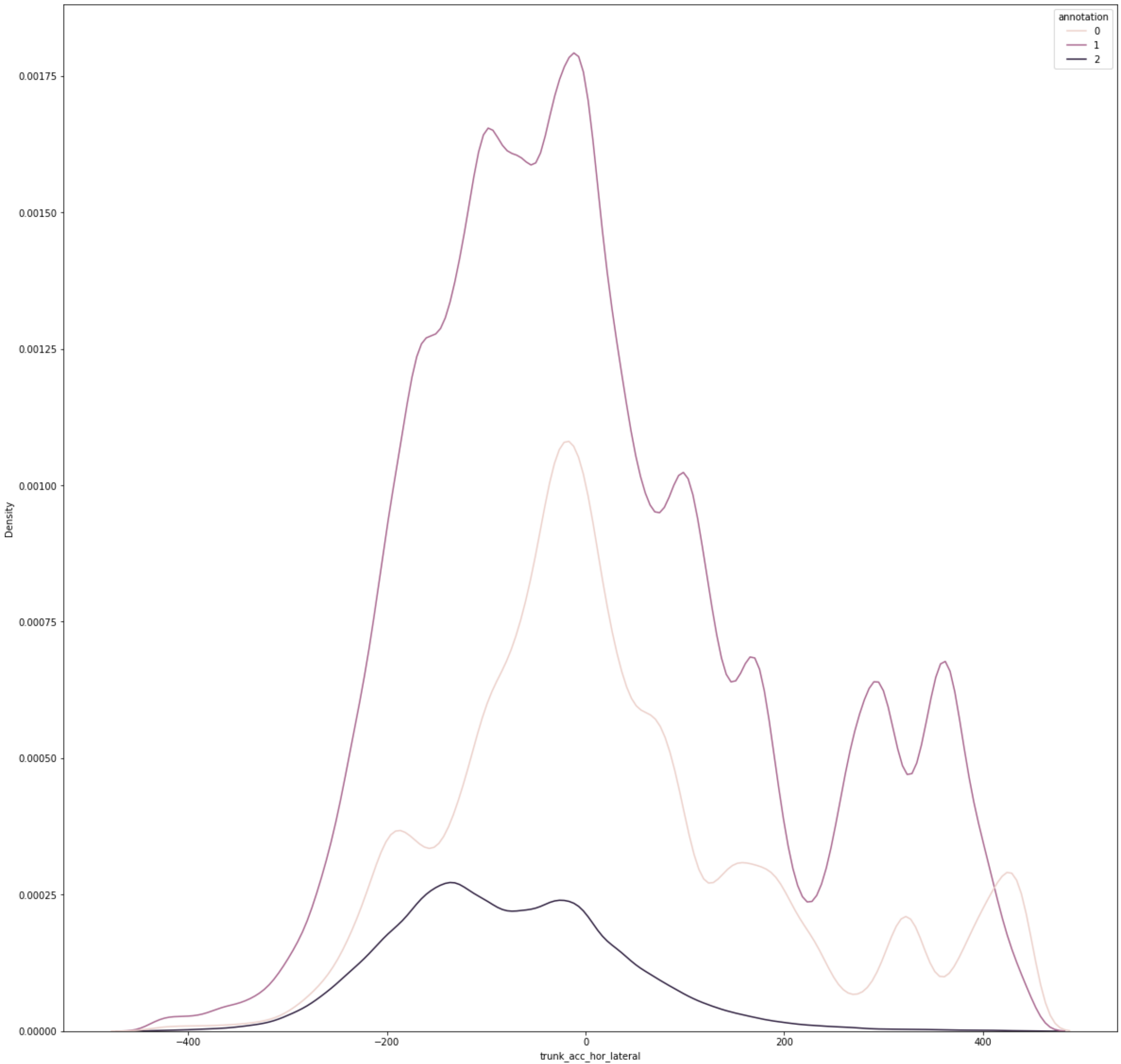
```
plt.figure(figsize= (20,20))
sns.kdeplot(df['trunk_acc_ver'], hue= df['annotation'])
```

Out[84]: <AxesSubplot:xlabel='trunk_acc_ver', ylabel='Density'>



In [85]:
```
plt.figure(figsize= (20,20))
sns.kdeplot(df['trunk_acc_hor_lateral'], hue= df['annotation'])
```

Out[85]: <AxesSubplot:xlabel='trunk_acc_hor_lateral', ylabel='Density'>



In [86]:
```
df_0
```

Out[86]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	uppe
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
985130	-373	931	306	854	
985131	-363	911	306	872	
985132	-353	950	297	845	
985133	-363	931	316	863	
985134	-393	931	306	854	

289021 rows × 10 columns

In [87]:
```
df_0.describe()
```

Out[87]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upper
count	289021.000000	289021.000000	289021.000000	289021.000000	28
mean	-79.730860	982.137554	249.569190	-195.575913	
std	194.767705	60.305208	131.883896	527.765718	
min	-787.000000	764.000000	-108.000000	-1281.000000	
25%	-191.000000	960.000000	138.000000	-827.000000	
50%	-70.000000	1000.000000	237.000000	-109.000000	
75%	70.000000	1019.000000	326.000000	72.000000	
max	646.000000	1186.000000	594.000000	1027.000000	

In [88]:
```
df_1.describe()
```

Out[88]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upper
count	656131.000000	656131.000000	656131.000000	656131.000000	65
mean	-107.535253	980.237061	247.035475	6.775365	
std	238.930753	75.199978	126.401102	506.031101	
min	-787.000000	764.000000	-108.000000	-1281.000000	
25%	-272.000000	941.000000	168.000000	-227.000000	
50%	-90.000000	1000.000000	237.000000	-27.000000	
75%	80.000000	1029.000000	326.000000	227.000000	
max	646.000000	1186.000000	594.000000	1027.000000	

In [89]:
```
df_2.describe()
```

Out[89]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upper_
count	76729.000000	76729.000000	76729.000000	76729.000000	76
mean	27.412491	1008.111913	235.311877	-91.326852	
std	197.420793	70.786855	103.266920	262.698298	
min	-787.000000	764.000000	-108.000000	-1281.000000	
25%	-121.000000	980.000000	168.000000	-254.000000	
50%	50.000000	1009.000000	237.000000	-145.000000	
75%	181.000000	1039.000000	306.000000	90.000000	

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upper_
max	646.000000	1186.000000	594.000000	1027.000000	1

Interpretation of the above Graphs and the description df's

From the above distribution plots, we could not find any feature which shows a clear distinction between the 0, 1 and 2 category. Also from the description of the 3 df's, the mean and the standard deviation also didnot show any clear distinction between the categories.

In [90]:

```
df_heatmap = df.drop(columns=['annotation'])
```

In [91]:

```
df_heatmap
```

Out[91]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

1021881 rows × 9 columns

In [92]:

```
df_heatmap.corr()
```

Out[92]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_
ankle_acc_hor_forward	1.000000	0.460587	-0.031685	
ankle_acc_ver	0.460587	1.000000	-0.438856	
ankle_acc_hor_lateral	-0.031685	-0.438856	1.000000	
upper_leg_acc_hor_forward	-0.253820	-0.120314	-0.078814	
upper_leg_acc_ver	0.311477	0.478124	-0.356738	
upper_leg_acc_hor_lateral	0.151415	-0.001425	0.235377	
trunk_acc_hor_forward	0.057021	-0.063498	0.040323	

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_
trunk_acc_ver	0.020151	0.069131	-0.113298	
trunk_acc_hor_lateral	-0.234631	-0.309249	0.371644	

From the correlation plots, we could see come medium level collinearity, but no significance collinearity can be seen between the features. Therefore we can conclude that there is no multicollinearity present in the dataset.

ANOVA test

Here we will perform ANOVA test followed by Turkey's HSD test to see whether there is a correlation between the independent and the dependent features.

In [16]:
```
import pandas as pd
from scipy.stats import f_oneway
import statsmodels.stats.multicomp as mc
```

In [17]:
```
df = pd.read_csv("../dataset/merged.csv")
df.drop(columns = ["Time(ms)"], inplace = True)
df
```

Out[17]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

1021881 rows × 10 columns

In [18]:
```
num_attr = df['ankle_acc_hor_forward']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 12818.10
p-value: 0.0000
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper   reject
-----------------------------------------------------
     0      1 -27.8044 0.001  -28.978 -26.6308   True
     0      2 107.1434 0.001 105.0084 109.2783   True
     1      2 134.9477 0.001  132.942 136.9535   True
-----------------------------------------------------
```

In [19]:

```python
num_attr = df['ankle_acc_ver']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 5321.06
p-value: 0.0000
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj  lower   upper  reject
---------------------------------------------------
     0      1  -1.9005 0.001 -2.2718 -1.5292   True
     0      2  25.9744 0.001 25.2989 26.6498   True
     1      2  27.8749 0.001 27.2403 28.5094   True
---------------------------------------------------
```

In [20]:

```python
num_attr = df['ankle_acc_hor_lateral']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 387.20
p-value: 0.0000
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper   reject
-----------------------------------------------------
     0      1  -2.5337 0.001  -3.1951  -1.8723   True
```

```
     0      2 -14.2573 0.001 -15.4605 -13.0542   True
     1      2 -11.7236 0.001 -12.8539 -10.5933   True
-----------------------------------------------------
```

In [21]:

```
num_attr = df['upper_leg_acc_hor_forward']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 16725.67
p-value: 0.0000
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
======================================================
group1 group2 meandiff p-adj   lower    upper   reject
------------------------------------------------------
     0      1 202.3513 0.001  199.7436 204.9589   True
     0      2 104.2491 0.001   99.5055 108.9927   True
     1      2 -98.1022 0.001 -102.5587 -93.6457   True
------------------------------------------------------
```

In [22]:

```
num_attr = df['upper_leg_acc_ver']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 21270.65
p-value: 0.0000
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper   reject
-----------------------------------------------------
     0      1  59.9089 0.001  58.2793  61.5385   True
     0      2 260.6735 0.001 257.7091 263.6379   True
     1      2 200.7646 0.001 197.9796 203.5496   True
-----------------------------------------------------
```

In [23]:

```
num_attr = df['upper_leg_acc_hor_lateral']
cat_attr = df['annotation']
```

```
# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 455.30
p-value: 0.0000
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj  lower   upper  reject
---------------------------------------------------
     0      1   9.0612 0.001  8.2941  9.8282   True
     0      2  12.8964 0.001 11.5011 14.2918   True
     1      2   3.8353 0.001  2.5244  5.1462   True
---------------------------------------------------
```

In [24]:

```
num_attr = df['trunk_acc_hor_forward']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 7810.88
p-value: 0.0000
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper   reject
-----------------------------------------------------
     0      1  43.2715 0.001   42.411   44.132   True
     0      2   4.3558 0.001   2.7905   5.9211   True
     1      2 -38.9157 0.001 -40.3863 -37.4451   True
-----------------------------------------------------
```

In [25]:

```
num_attr = df['trunk_acc_ver']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)
```

```
# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 4737.98
p-value: 0.0000
 Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower    upper   reject
-----------------------------------------------------
     0      1 -14.3593 0.001 -14.7137 -14.0048   True
     0      2  -4.5125 0.001  -5.1572  -3.8677   True
     1      2   9.8468 0.001   9.2411  10.4526   True
-----------------------------------------------------
```

In [26]:

```
num_attr = df['trunk_acc_hor_lateral']
cat_attr = df['annotation']

# Group the numerical attribute by the categorical attribute
groups = [num_attr[cat_attr == category] for category in set(cat_attr)]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(*groups)

# Print the results
print("F-statistic: {:.2f}".format(f_statistic))
print("p-value: {:.4f}".format(p_value))

tukey = mc.MultiComparison(num_attr, cat_attr)
tukey_result = tukey.tukeyhsd()

print(tukey_result)
```

```
F-statistic: 12467.38
p-value: 0.0000
  Multiple Comparison of Means - Tukey HSD, FWER=0.05
========================================================
group1 group2  meandiff p-adj   lower     upper   reject
--------------------------------------------------------
     0      1  -14.3138 0.001  -15.2011  -13.4265   True
     0      2 -107.5353 0.001 -109.1495 -105.9212   True
     1      2  -93.2215 0.001   -94.738  -91.7051   True
--------------------------------------------------------
```

Description of the above test

The above test was ANOVA test followed by Turkey's HSD test

ANOVA test is done to see if there is any significance difference in the means of each categories for all the attributes.

NULL Hypothesis : There is no significant difference in the means of the categories. ALTERNATE Hypothesis : There is significant difference in the means of the categories.

Tukey's HSD (honestly significant difference) test is a post-hoc test that can be used to determine which groups are significantly different from each other after performing an ANOVA test.

Now from the above results we can conclude that for all the ANOVA tests, F-statistic value is fairly greater than the p-value and the p-value is smaller than the significance value (0.05). So we can reject the null hypothesis. Also the Turkey's HSD test has shown that the groups are significantly different from each other as the p-values of all the above HSD test table has value greater than significant value(0.05)

Overall Conclusion regarding the Data Analysis

1. We visualized the data using the distribution plots and could not see any significant difference.
2. We plotted the correlation plot to find out multicollinearity, but could not find any multicollinearity.
3. We performed ANOVA test followed by Turkey's HSD test, and concluded that all the features are showing group wise significant difference.

PCA

Here we will be using the Principal Component Analysis method of dimensionality reduction for selecting important features for our model.

In [16]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
```

In [5]:
```
df = pd.read_csv("../dataset/merged.csv")
df.drop(columns=['Time(ms)'], inplace=True)
df
```

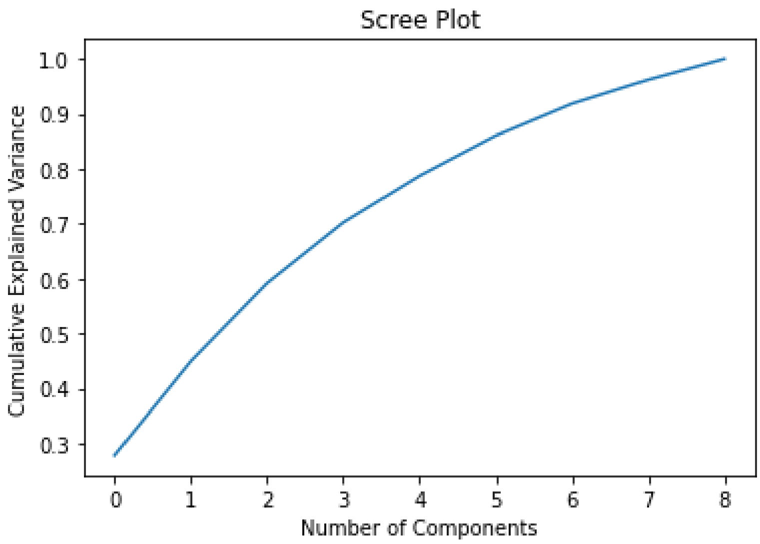
Out[5]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

1021881 rows × 10 columns

In [6]:
```
x = df.drop('annotation', axis=1)
y = df['annotation']
```

In [9]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
pca = PCA()
pca.fit_transform(x_scaled)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Scree Plot')
plt.show()
```



In [11]:

```
pca = PCA(n_components=6)
x_pca = pca.fit_transform(x_scaled)
print('Explained Variance Ratio:', pca.explained_variance_ratio_)
```

```
Explained Variance Ratio: [0.27855194 0.17133403 0.14202023 0.110601   0.08502709 0.
07302196]
```

In [14]:

```
pca_df = pd.DataFrame(data = x_pca, columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'P
pca_df['annotation'] = y
pca_df
```

Out[14]:

	PC1	PC2	PC3	PC4	PC5	PC6	annotation
0	-2.040942	3.389135	-0.697435	2.427390	2.321785	-0.222592	0
1	-1.235390	1.358520	-0.656503	0.102055	0.388039	-1.847160	0
2	-1.501984	1.165822	-0.824926	0.141681	1.516479	-1.859010	0
3	-1.638330	1.481119	-0.705295	0.630510	2.080654	-1.730703	0
4	-1.268800	1.588218	-0.809263	0.511559	0.913394	-1.778680	0
...	...	...	...	...	...	...	...
1021876	-3.062620	0.818409	-0.151781	0.252098	-0.139664	-1.649739	1
1021877	-2.871607	-0.420508	0.406236	0.927885	-0.445085	-1.281501	1
1021878	-2.641319	-0.747211	0.250917	1.298763	-0.566486	-1.162638	1
1021879	-3.087607	-1.209168	0.641357	1.558142	-0.524909	-0.810112	1
1021880	-3.406090	-1.275729	0.701268	1.277839	-0.523718	-0.440292	1

1021881 rows × 7 columns

In [15]:

```
pca_df.to_csv("../dataset/pca_df.csv", index=False)
```

Modeling Part 1

Here we shall be modeling the dataset based on Logistic Regression, Decision Tree and Random Forest and then will be calculating the results in form of confusion matrix and classification report.

In [57]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestClassifier
```

In [4]:

```
df = pd.read_csv("../dataset/merged.csv")
df.drop(columns=["Time(ms)"], inplace=True)
df
```

Out[4]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

1021881 rows × 10 columns

In [13]:

```
df_pca = pd.read_csv("../dataset/pca_df.csv")
df_pca
```

Out[13]:

	PC1	PC2	PC3	PC4	PC5	PC6	annotation
0	-2.040942	3.389135	-0.697435	2.427390	2.321785	-0.222592	0
1	-1.235390	1.358520	-0.656503	0.102055	0.388039	-1.847160	0
2	-1.501984	1.165822	-0.824926	0.141681	1.516479	-1.859010	0

	PC1	PC2	PC3	PC4	PC5	PC6	annotation
3	-1.638330	1.481119	-0.705295	0.630510	2.080654	-1.730703	0
4	-1.268800	1.588218	-0.809263	0.511559	0.913394	-1.778680	0
...	...	...	...	...	...	...	...
1021876	-3.062620	0.818409	-0.151781	0.252098	-0.139664	-1.649739	1
1021877	-2.871607	-0.420508	0.406236	0.927885	-0.445085	-1.281501	1
1021878	-2.641319	-0.747211	0.250917	1.298763	-0.566486	-1.162638	1
1021879	-3.087607	-1.209168	0.641357	1.558142	-0.524909	-0.810112	1
1021880	-3.406090	-1.275729	0.701268	1.277839	-0.523718	-0.440292	1

1021881 rows × 7 columns

In [9]:

```
x = df.drop(columns=["annotation"])
y = df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[  7249  50664     50]
 [  3291 127355    495]
 [   101  15048    124]]
              precision    recall  f1-score   support

           0       0.68      0.13      0.21     57963
           1       0.66      0.97      0.79    131141
           2       0.19      0.01      0.02     15273

    accuracy                           0.66    204377
   macro avg       0.51      0.37      0.34    204377
weighted avg       0.63      0.66      0.57    204377
```

In [12]:

```
for i in range(2,10):
    rfe = RFE(model, n_features_to_select=i)
    rfe.fit(x_train, y_train)
    x_train_rfe = rfe.transform(x_train)
    x_test_rfe = rfe.transform(x_test)
    model.fit(x_train_rfe, y_train)
    y_pred = model.predict(x_test_rfe)
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```
[[    19  57915     29]
 [   230 130560    351]
 [     0  15242     31]]
              precision    recall  f1-score   support

           0       0.08      0.00      0.00     57963
           1       0.64      1.00      0.78    131141
           2       0.08      0.00      0.00     15273
```

```
    accuracy                           0.64    204377
   macro avg       0.26      0.33      0.26    204377
weighted avg       0.44      0.64      0.50    204377

[[    20  57915     28]
 [   186 130673    282]
 [     0  15234     39]]
              precision    recall  f1-score   support

           0       0.10      0.00      0.00     57963
           1       0.64      1.00      0.78    131141
           2       0.11      0.00      0.00     15273

    accuracy                           0.64    204377
   macro avg       0.28      0.33      0.26    204377
weighted avg       0.45      0.64      0.50    204377

[[  1790  56151     22]
 [  6364 124552    225]
 [     0  15246     27]]
              precision    recall  f1-score   support

           0       0.22      0.03      0.05     57963
           1       0.64      0.95      0.76    131141
           2       0.10      0.00      0.00     15273

    accuracy                           0.62    204377
   macro avg       0.32      0.33      0.27    204377
weighted avg       0.48      0.62      0.50    204377

[[  8251  49682     30]
 [  3446 127231    464]
 [    28  15102    143]]
              precision    recall  f1-score   support

           0       0.70      0.14      0.24     57963
           1       0.66      0.97      0.79    131141
           2       0.22      0.01      0.02     15273

    accuracy                           0.66    204377
   macro avg       0.53      0.37      0.35    204377
weighted avg       0.64      0.66      0.57    204377

[[  9746  48186     31]
 [  5925 124757    459]
 [   104  15026    143]]
              precision    recall  f1-score   support

           0       0.62      0.17      0.26     57963
           1       0.66      0.95      0.78    131141
           2       0.23      0.01      0.02     15273

    accuracy                           0.66    204377
   macro avg       0.50      0.38      0.35    204377
weighted avg       0.62      0.66      0.58    204377

[[  7512  50416     35]
 [  5153 125530    458]
 [   105  15052    116]]
              precision    recall  f1-score   support

           0       0.59      0.13      0.21     57963
           1       0.66      0.96      0.78    131141
           2       0.19      0.01      0.01     15273

    accuracy                           0.65    204377
   macro avg       0.48      0.36      0.34    204377
weighted avg       0.60      0.65      0.56    204377
```

```
[[  7254  50660    49]
 [  3303 127341   497]
 [   101  15045   127]]
              precision    recall  f1-score   support

           0       0.68      0.13      0.21     57963
           1       0.66      0.97      0.79    131141
           2       0.19      0.01      0.02     15273

    accuracy                           0.66    204377
   macro avg       0.51      0.37      0.34    204377
weighted avg       0.63      0.66      0.57    204377

[[  7249  50664    50]
 [  3291 127355   495]
 [   101  15048   124]]
              precision    recall  f1-score   support

           0       0.68      0.13      0.21     57963
           1       0.66      0.97      0.79    131141
           2       0.19      0.01      0.02     15273

    accuracy                           0.66    204377
   macro avg       0.51      0.37      0.34    204377
weighted avg       0.63      0.66      0.57    204377
```

In [14]:

```
x = df_pca.drop(columns=["annotation"])
y = df_pca["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = LogisticRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[  4356  53599     8]
 [  4884 126093   164]
 [   132  15076    65]]
              precision    recall  f1-score   support

           0       0.46      0.08      0.13     57963
           1       0.65      0.96      0.77    131141
           2       0.27      0.00      0.01     15273

    accuracy                           0.64    204377
   macro avg       0.46      0.35      0.30    204377
weighted avg       0.57      0.64      0.53    204377
```

In [15]:

```
for i in range(2,6):
    rfe = RFE(model, n_features_to_select=i)
    rfe.fit(x_train, y_train)
    x_train_rfe = rfe.transform(x_train)
    x_test_rfe = rfe.transform(x_test)
    model.fit(x_train_rfe, y_train)
    y_pred = model.predict(x_test_rfe)
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```
[[     0  57958     5]
 [     0 131122    19]
 [     0  15265     8]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00     57963
```

```
           1       0.64      1.00      0.78    131141
           2       0.25      0.00      0.00     15273

    accuracy                           0.64    204377
   macro avg       0.30      0.33      0.26    204377
weighted avg       0.43      0.64      0.50    204377

C:\Users\Dev\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1308: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1308: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1308: Un
definedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
[[   205  57749      9]
 [   166 130890     85]
 [     8  15235     30]]
              precision    recall  f1-score   support

           0       0.54      0.00      0.01     57963
           1       0.64      1.00      0.78    131141
           2       0.24      0.00      0.00     15273

    accuracy                           0.64    204377
   macro avg       0.47      0.33      0.26    204377
weighted avg       0.58      0.64      0.50    204377

[[  5006  52949      8]
 [  4108 126969     64]
 [    73  15176     24]]
              precision    recall  f1-score   support

           0       0.54      0.09      0.15     57963
           1       0.65      0.97      0.78    131141
           2       0.25      0.00      0.00     15273

    accuracy                           0.65    204377
   macro avg       0.48      0.35      0.31    204377
weighted avg       0.59      0.65      0.54    204377

[[  4627  53330      6]
 [  5249 125765    127]
 [   130  15095     48]]
              precision    recall  f1-score   support

           0       0.46      0.08      0.14     57963
           1       0.65      0.96      0.77    131141
           2       0.27      0.00      0.01     15273

    accuracy                           0.64    204377
   macro avg       0.46      0.35      0.31    204377
weighted avg       0.57      0.64      0.54    204377
```

In [17]:

```
x = df.drop(columns=["annotation"])
y = df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

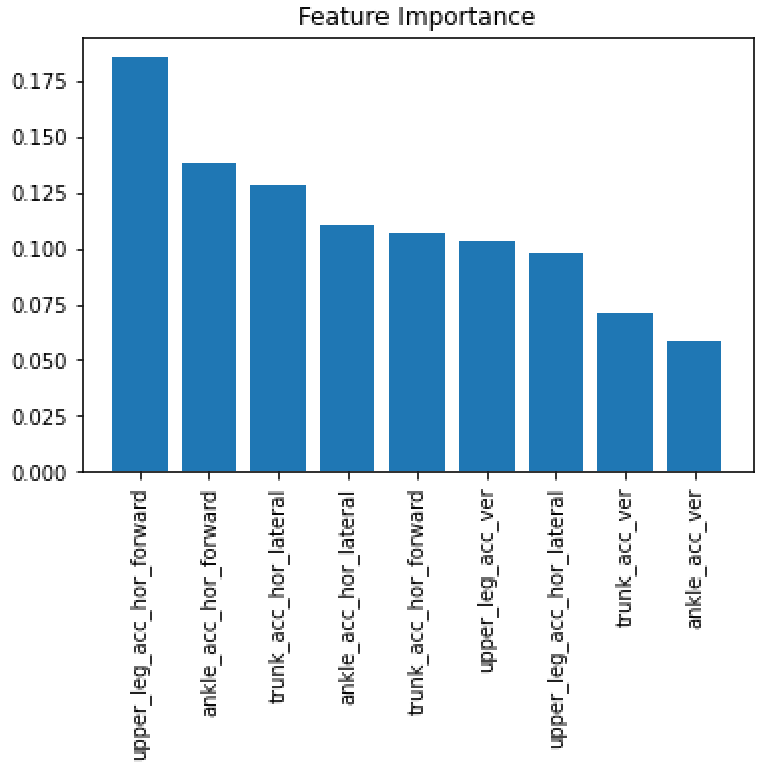
```
[[ 47030   9531   1402]
 [ 10213 113009   7919]
 [  1506   7263   6504]]
              precision    recall  f1-score   support

           0       0.80      0.81      0.81     57963
           1       0.87      0.86      0.87    131141
           2       0.41      0.43      0.42     15273

    accuracy                           0.81    204377
   macro avg       0.69      0.70      0.70    204377
weighted avg       0.82      0.81      0.82    204377
```

In [22]:

```
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]
names = [x_train.columns[i] for i in indices]
plt.figure()
plt.title("Feature Importance")
plt.bar(range(x_train.shape[1]), importances[indices])
plt.xticks(range(x_train.shape[1]), names, rotation=90)
plt.show()
```



In [45]:

```
lst = ['upper_leg_acc_hor_forward', 'ankle_acc_hor_forward', 'trunk_acc_hor_lateral'
       'ankle_acc_hor_lateral', 'trunk_acc_hor_forward', 'upper_leg_acc_ver']
x_latest = df[lst]
new_df = pd.concat([x_latest, df['annotation']], axis=1)
new_df
```

Out[45]:

	upper_leg_acc_hor_forward	ankle_acc_hor_forward	trunk_acc_hor_lateral	ankle_acc_hor_later
0	-36	-40	349	32
1	54	-60	446	31

	upper_leg_acc_hor_forward	ankle_acc_hor_forward	trunk_acc_hor_lateral	ankle_acc_hor_later
2	-27	-111	446	34
3	36	-111	446	34
4	18	-60	436	34
...	...	...	...	
1021876	690	-212	349	37
1021877	572	-232	339	35
1021878	272	-232	359	35
1021879	354	-252	359	34
1021880	390	-232	368	36

1021881 rows × 7 columns

In [46]:

```
new_df.isna().sum()
```

Out[46]:

```
upper_leg_acc_hor_forward    0
ankle_acc_hor_forward        0
trunk_acc_hor_lateral        0
ankle_acc_hor_lateral        0
trunk_acc_hor_forward        0
upper_leg_acc_ver            0
annotation                   0
dtype: int64
```

In [47]:

```
x = new_df.drop(columns=["annotation"])
y = new_df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 46124  10285   1554]
 [ 11574 110810   8757]
 [  1576   7797   5900]]
              precision    recall  f1-score   support

           0       0.78      0.80      0.79     57963
           1       0.86      0.84      0.85    131141
           2       0.36      0.39      0.37     15273

    accuracy                           0.80    204377
   macro avg       0.67      0.68      0.67    204377
weighted avg       0.80      0.80      0.80    204377
```

In [48]:

```
x = df_pca.drop(columns=["annotation"])
y = df_pca["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 42873  13129   1961]
 [ 13558 108370   9213]
 [  1826   8525   4922]]
              precision    recall  f1-score   support

           0       0.74      0.74      0.74     57963
           1       0.83      0.83      0.83    131141
           2       0.31      0.32      0.31     15273

    accuracy                           0.76    204377
   macro avg       0.63      0.63      0.63    204377
weighted avg       0.77      0.76      0.77    204377
```

In [52]:

```
x = df.drop(columns=["annotation"])
y = df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = RandomForestClassifier(n_estimators=500, random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

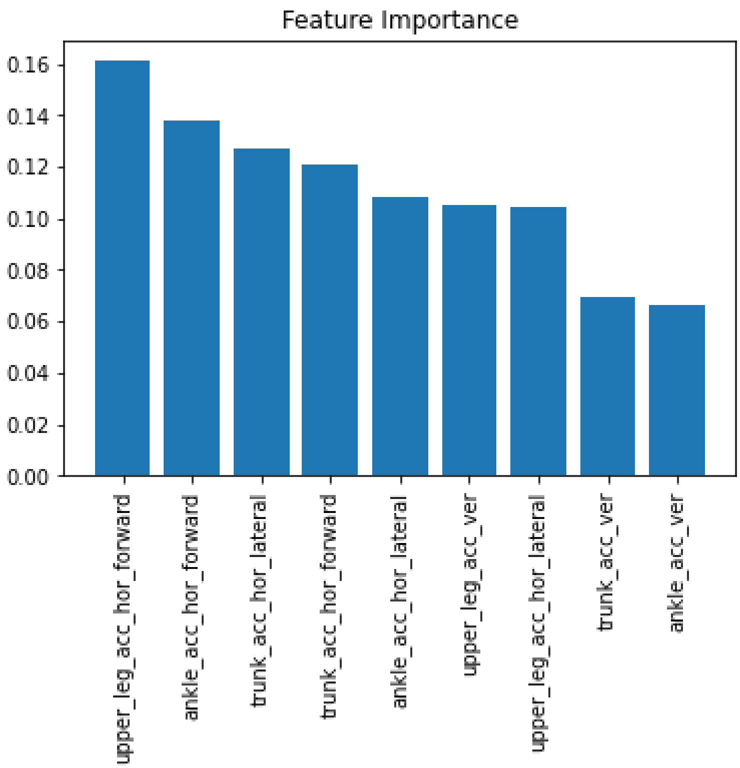
```
[[ 48384   9280    299]
 [  4083 125296   1762]
 [   294   9633   5346]]
              precision    recall  f1-score   support

           0       0.92      0.83      0.87     57963
           1       0.87      0.96      0.91    131141
           2       0.72      0.35      0.47     15273

    accuracy                           0.88    204377
   macro avg       0.84      0.71      0.75    204377
weighted avg       0.87      0.88      0.87    204377
```

In [53]:

```
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]
names = [x_train.columns[i] for i in indices]
plt.figure()
plt.title("Feature Importance")
plt.bar(range(x_train.shape[1]), importances[indices])
plt.xticks(range(x_train.shape[1]), names, rotation=90)
plt.show()
```



In [54]:

```
lst = ['upper_leg_acc_hor_forward', 'ankle_acc_hor_forward', 'trunk_acc_hor_lateral'
       'ankle_acc_hor_lateral', 'trunk_acc_hor_forward', 'upper_leg_acc_ver']
x_latest = df[lst]
new_df = pd.concat([x_latest, df['annotation']], axis=1)
new_df
```

Out[54]:

	upper_leg_acc_hor_forward	ankle_acc_hor_forward	trunk_acc_hor_lateral	ankle_acc_hor_later
0	-36	-40	349	32
1	54	-60	446	31
2	-27	-111	446	34
3	36	-111	446	34
4	18	-60	436	34
...	...	...	...	
1021876	690	-212	349	37
1021877	572	-232	339	35
1021878	272	-232	359	35
1021879	354	-252	359	34
1021880	390	-232	368	36

1021881 rows × 7 columns

In [55]:

```
x = new_df.drop(columns=["annotation"])
y = new_df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = RandomForestClassifier(n_estimators=500, random_state=42)
model.fit(x_train, y_train)
```

```
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 46923  10614    426]
 [  5538 123205   2398]
 [   471   9994   4808]]
              precision    recall  f1-score   support

           0       0.89      0.81      0.85     57963
           1       0.86      0.94      0.90    131141
           2       0.63      0.31      0.42     15273

    accuracy                           0.86    204377
   macro avg       0.79      0.69      0.72    204377
weighted avg       0.85      0.86      0.85    204377
```

In [56]:

```
x = df_pca.drop(columns=["annotation"])
y = df_pca["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
model = RandomForestClassifier(n_estimators=250, random_state=42)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 44529  13050    384]
 [  6158 123423   1560]
 [   563  11705   3005]]
              precision    recall  f1-score   support

           0       0.87      0.77      0.82     57963
           1       0.83      0.94      0.88    131141
           2       0.61      0.20      0.30     15273

    accuracy                           0.84    204377
   macro avg       0.77      0.64      0.67    204377
weighted avg       0.83      0.84      0.82    204377
```

Modeling Part 2

Here we shall be modeling the dataset based on KNN, XGBoost and then will be calculating the results in form of confusion matrix and classification report.

In [3]:

```
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import xgboost as xgb
import seaborn as sns
```

In [2]:

```
df = pd.read_csv("../dataset/merged.csv")
df.drop(columns=["Time(ms)"], inplace=True)
df
```

Out[2]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

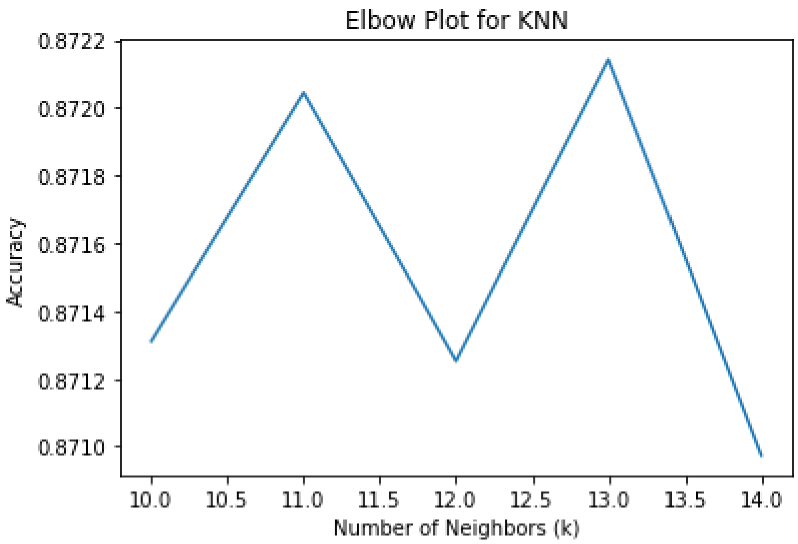
1021881 rows × 10 columns

In [6]:

```
x = df.drop(columns=["annotation"])
y = df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
k_range = range(10,15)
accuracy_scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
    print(k)
plt.plot(k_range, accuracy_scores)
plt.xlabel('Number of Neighbors (k)')
```

```
plt.ylabel('Accuracy')
plt.title('Elbow Plot for KNN')
plt.show()
```

```
10
11
12
13
14
```



In [7]:

```
knn = KNeighborsClassifier(n_neighbors=13)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 49715   7837    411]
 [  5987 122332   2822]
 [   779   8295   6199]]
              precision    recall  f1-score   support

           0       0.88      0.86      0.87     57963
           1       0.88      0.93      0.91    131141
           2       0.66      0.41      0.50     15273

    accuracy                           0.87    204377
   macro avg       0.81      0.73      0.76    204377
weighted avg       0.87      0.87      0.87    204377
```

In [12]:

```
dtrain = xgb.DMatrix(x_train, label=y_train)
dtest = xgb.DMatrix(x_test, label=y_test)
params = {
    'max_depth': 500,
    'learning_rate': 0.01,
    'objective': 'multi:softmax',
    'num_class': 3
}
num_rounds = 50
model = xgb.train(params, dtrain, num_rounds)
y_pred = model.predict(dtest)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[13:30:05] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/sr
c/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'multi:softmax' was changed from 'merror' to 'mlogloss'. Explicitly
```

```
set eval_metric if you'd like to restore the old behavior.
[[ 48134   9134    695]
 [  6036 121120   3985]
 [   766   8417   6090]]
              precision    recall  f1-score   support

           0       0.88      0.83      0.85     57963
           1       0.87      0.92      0.90    131141
           2       0.57      0.40      0.47     15273

    accuracy                           0.86    204377
   macro avg       0.77      0.72      0.74    204377
weighted avg       0.85      0.86      0.85    204377
```

Interpretation of the model results

We have worked with various algorithms such as :

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. KNN
5. XGBoost

Best Results from each model experimentation are as follows :

1. Logistic Regression

```
      precision    recall  f1-score   support

   0       0.62      0.17      0.26     57963
   1       0.66      0.95      0.78    131141
   2       0.23      0.01      0.02     15273
```

   accuracy 0.66

2. Decision Tree

```
          precision    recall  f1-score   support

       0       0.80      0.81      0.81     57963
       1       0.87      0.86      0.87    131141
       2       0.41      0.43      0.42     15273

accuracy                           0.81
```

1. Random Forest

```
        precision    recall  f1-score   support

   0       0.92      0.83      0.87     57963
   1       0.87      0.96      0.91    131141
   2       0.72      0.35      0.47     15273
```

   accuracy 0.88

2. KNN

	precision	recall	f1-score	support
0	0.88	0.86	0.87	57963
1	0.88	0.93	0.91	131141
2	0.66	0.41	0.50	15273

accuracy 0.87

3. XGBoost

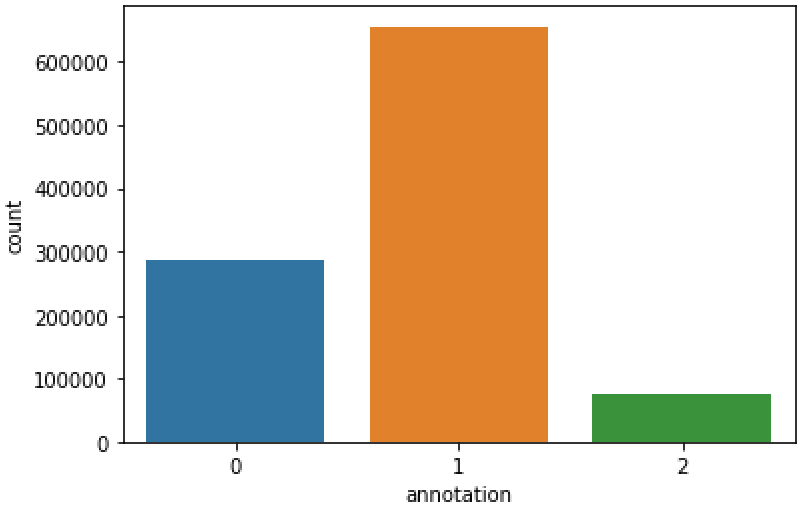
	precision	recall	f1-score	support
0	0.88	0.83	0.85	57963
1	0.87	0.92	0.90	131141
2	0.57	0.40	0.47	15273

accuracy 0.86

From the above results we can clearly see that KNN is giving us the most stable model with the category 2 recall and precision values of 41% and 66% respectively. But since KNN is not productionable as it takes a lot of time to produce results, we wont be using this algorithm further. Followed by XGBoost, Decision Tree and Random Forest, though the recall value is still not up to mark. It is due to the class imbalance problem, i.e. category 2 has very less data as compared to other two categories. Therefore in order to solve this problem, we will be using Oversampling technique such as Borderline SMOTE and shall retrain the models with the new dataset obtained.

In [4]:
```
sns.countplot(x = 'annotation', data=df)
```

Out[4]: <AxesSubplot:xlabel='annotation', ylabel='count'>



Borderline SMOTE

Here we will use Borderline SMOTE which is an extension of SMOTE to balance the classes and then will move forward with the modeling section.

In [33]:
```
from imblearn.over_sampling import BorderlineSMOTE
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import joblib
```

In [8]:
```
df = pd.read_csv("../dataset/merged.csv")
df.drop(columns=["Time(ms)"], inplace=True)
df
```

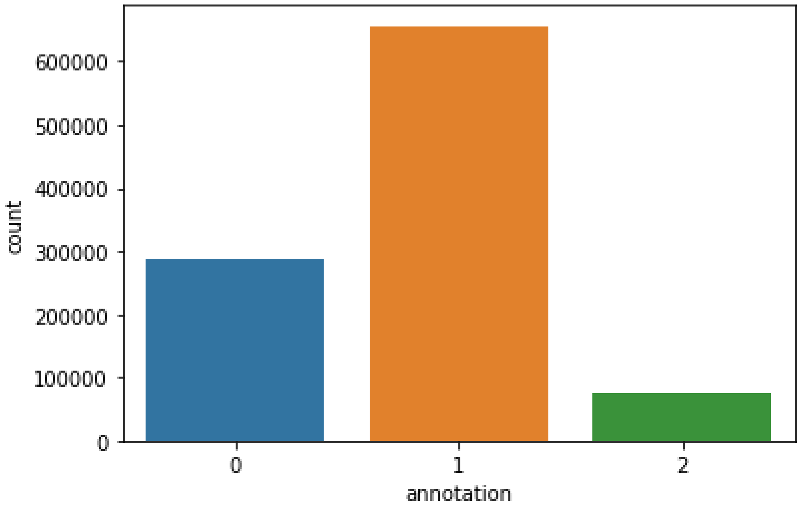
Out[8]:

	ankle_acc_hor_forward	ankle_acc_ver	ankle_acc_hor_lateral	upper_leg_acc_hor_forward	upp
0	-40	970	326	-36	
1	-60	990	316	54	
2	-111	980	346	-27	
3	-111	980	346	36	
4	-60	1009	346	18	
...	...	...	...	...	
1021876	-212	1000	376	690	
1021877	-232	970	356	572	
1021878	-232	970	356	272	
1021879	-252	921	346	354	
1021880	-232	862	366	390	

1021881 rows × 10 columns

In [9]:
```
sns.countplot(x = 'annotation', data=df)
```

Out[9]: <AxesSubplot:xlabel='annotation', ylabel='count'>



In [10]:

```
x = df.drop(columns=["annotation"])
y = df["annotation"]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=
bsmote = BorderlineSMOTE(random_state=42)
x_train_resampled, y_train_resampled = bsmote.fit_resample(x_train, y_train)
```

In [32]:

```
y_train_resampled.value_counts()
```

Out[32]:

```
2    524990
1    524990
0    524990
Name: annotation, dtype: int64
```

In [17]:

```
params = {
    'max_depth': 500,
    'learning_rate': 0.01,
    'objective': 'multi:softmax',
    'num_class': 3
}
xgb = XGBClassifier(params)
xgb.fit(x_train_resampled, y_train_resampled)
y_pred = xgb.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
C:\Users\Dev\anaconda3\lib\site-packages\xgboost\core.py:430: FutureWarning: Pass `o
bjective` as keyword args.  Passing these as positional arguments will be considered
as error in future releases.
  warnings.warn(
C:\Users\Dev\anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarning: The u
se of label encoder in XGBClassifier is deprecated and will be removed in a future r
elease. To remove this warning, do the following: 1) Pass option use_label_encoder=F
alse when constructing XGBClassifier object; and 2) Encode your labels (y) as intege
rs starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[15:35:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/sr
c/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used wit
h the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly
set eval_metric if you'd like to restore the old behavior.
[[ 46115  11094    754]
 [ 11957 112930   6254]
 [   458   8754   6061]]
              precision    recall  f1-score   support

           0       0.79      0.80      0.79     57963
```

```
           1       0.85      0.86      0.86    131141
           2       0.46      0.40      0.43     15273

    accuracy                           0.81    204377
   macro avg       0.70      0.68      0.69    204377
weighted avg       0.80      0.81      0.81    204377
```

In [18]:

```python
model = DecisionTreeClassifier(random_state=42)
model.fit(x_train_resampled, y_train_resampled)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 47353   8797   1813]
 [ 11797 108429  10915]
 [  1568   6132   7573]]
              precision    recall  f1-score   support

           0       0.78      0.82      0.80     57963
           1       0.88      0.83      0.85    131141
           2       0.37      0.50      0.43     15273

    accuracy                           0.80    204377
   macro avg       0.68      0.71      0.69    204377
weighted avg       0.81      0.80      0.80    204377
```

In [21]:

```python
model = RandomForestClassifier(n_estimators=10, random_state=42)
model.fit(x_train_resampled, y_train_resampled)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 50809   6004   1150]
 [  9474 113296   8371]
 [   885   5620   8768]]
              precision    recall  f1-score   support

           0       0.83      0.88      0.85     57963
           1       0.91      0.86      0.88    131141
           2       0.48      0.57      0.52     15273

    accuracy                           0.85    204377
   macro avg       0.74      0.77      0.75    204377
weighted avg       0.85      0.85      0.85    204377
```

In [22]:

```python
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(x_train_resampled, y_train_resampled)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 50706   6084   1173]
 [  6774 115856   8511]
 [   522   4708  10043]]
              precision    recall  f1-score   support

           0       0.87      0.87      0.87     57963
           1       0.91      0.88      0.90    131141
           2       0.51      0.66      0.57     15273

    accuracy                           0.86    204377
   macro avg       0.77      0.81      0.78    204377
```

```
weighted avg       0.87      0.86      0.87    204377
```

In [23]:

```
model = RandomForestClassifier(n_estimators=200, random_state=42)
model.fit(x_train_resampled, y_train_resampled)
y_pred = model.predict(x_test)
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 50700   6087   1176]
 [  6603 116079   8459]
 [   504   4677  10092]]
              precision    recall  f1-score   support

           0       0.88      0.87      0.88     57963
           1       0.92      0.89      0.90    131141
           2       0.51      0.66      0.58     15273

    accuracy                           0.87    204377
   macro avg       0.77      0.81      0.78    204377
weighted avg       0.87      0.87      0.87    204377
```

In [34]:

```
joblib.dump(model, '../model/model_rf.joblib')
```

Out[34]:

```
['../model/model_rf.joblib']
```

Interpretation of the above experiment

We used Borderline SMOTE to oversample the training dataset so that the classes become balanced. After that we experimented with XGBoost, Decision Tree and Random Forest. From the experiments we found that the random forest model is the most stable one with precision and recall of the annotation 2 class being 51% and 66% respectively and the overall accuracy being 87%.

Now from here we can fine tune the model hyperparameters using GridSearchCV so as to increase the performance and the stability of the model. Due to time constraint here we will not be going forward with the GridSearchCV technique.

After that we will move to the deployment section where we will just display a basic deployment scenario showing the probability of freeze.

Basic Deployment Demostration

Here we are going to provide a basic deployment demonstration showing the probability of freeze at every time interval

In [32]:
```
import joblib
```

In [14]:
```
model = joblib.load("../model/model_rf.joblib")
```

In [31]:
```
input_list = [
    [-40,970,326,-36,962,242,320,657,349],
    [-60,990,316,54,953,262,77,914,446],
    [-111,980,346,-27,953,262,-48,857,446],
    [-212,1000,376,690,-166,282,77,942,349],
    [-60,1009,346,18,972,242,77,866,436],
    [-232,862,366,390,111,-191,87,952,368],
]

for i in range(len(input_list)):
    print("Time unit : ", i+1 , "  " , "Freeze probability : ", model.predict_proba(
```

```
Time unit :  1    Freeze probability :  0.085
Time unit :  2    Freeze probability :  0.005
Time unit :  3    Freeze probability :  0.0
Time unit :  4    Freeze probability :  0.0
Time unit :  5    Freeze probability :  0.01
Time unit :  6    Freeze probability :  0.005
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does no
t have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does no
t have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does no
t have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does no
t have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does no
t have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\Dev\anaconda3\lib\site-packages\sklearn\base.py:439: UserWarning: X does no
t have valid feature names, but RandomForestClassifier was fitted with feature names
  warnings.warn(
```

As we can see from the above demonstration that the system is able to provide the freezing probability at every time interval

In [ ]:
```

```