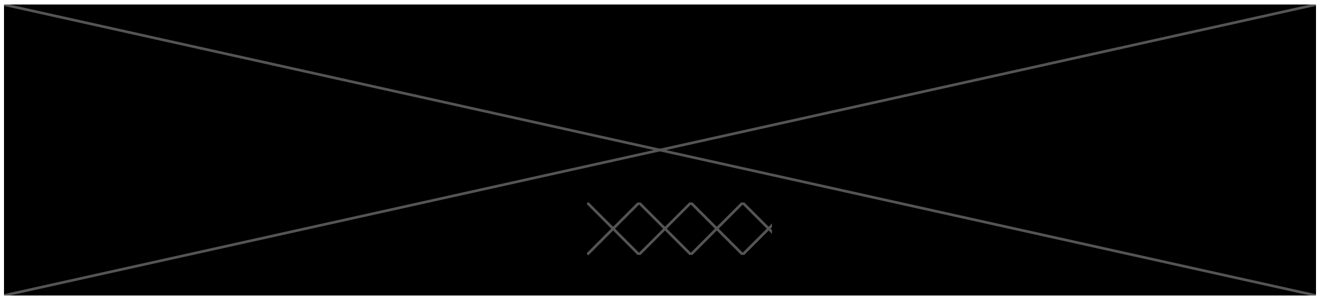
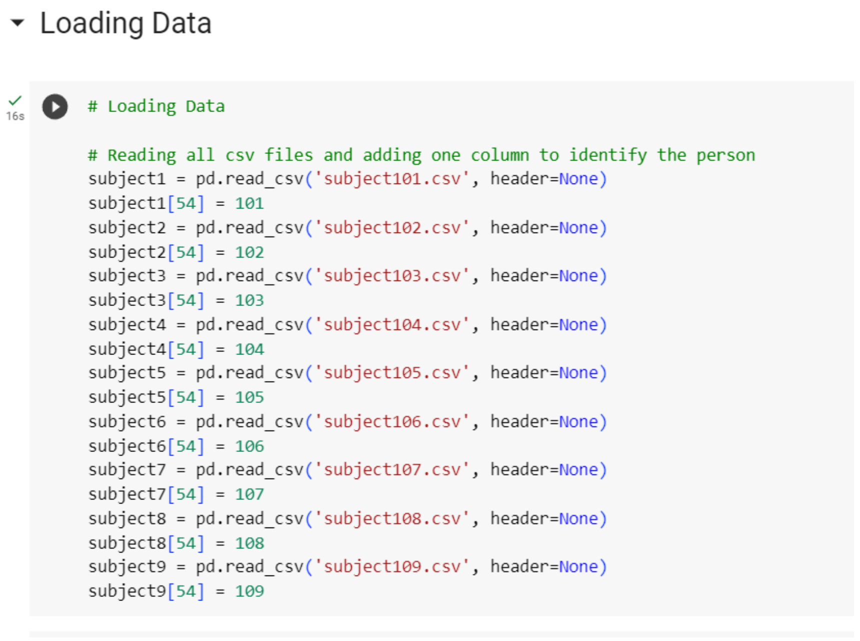


PAMAP2 Physical Activity Monitoring

The PAMAP2 Physical Activity Monitoring contains data of 18 different physical activities (such as walking, cycling, playing soccer, etc.), performed by 9 subjects wearing 3 inertial measurement units and a heart rate monitor. This dataset is being used for activity recognition and intensity estimation, while developing and applying algorithms of data processing. Each of the subjects follows a protocol, containing 12 different activities. Furthermore, a list of optional activities to perform was also suggested to the subjects. The list contains a wide range of everyday, household and sport activities. The different activities were performed by some of the subjects such as lying, sitting, standing, ironing, vacuuming, ascending stairs, descending stairs, normal walking, nordic walking, cycling, running, rope jumping, watching TV, computer work, car driving, folding laundry, house cleaning, playing soccer.

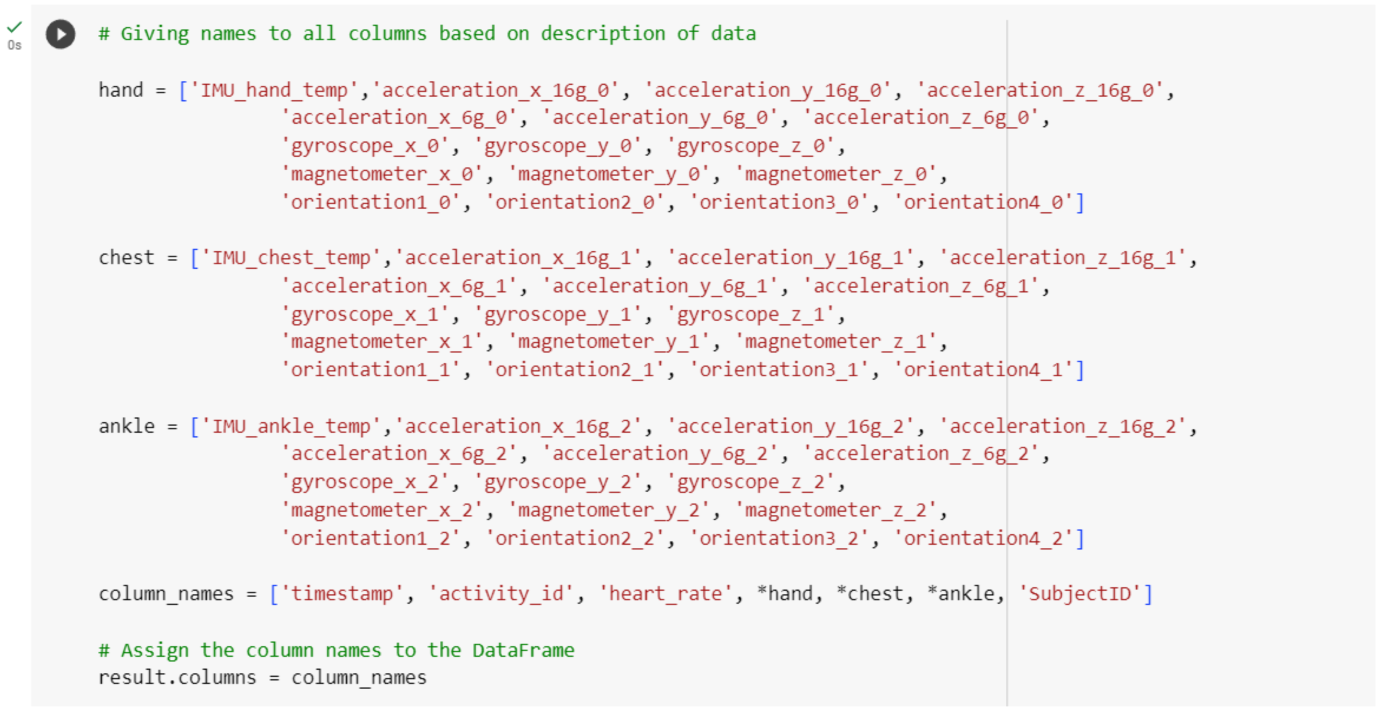
Loading the data was quite a challenge since there were 9 different .dat files for 9 different subjects. For this we had to convert these .dat files to .csv files. Subject ID has been added for better clarification in graphs.



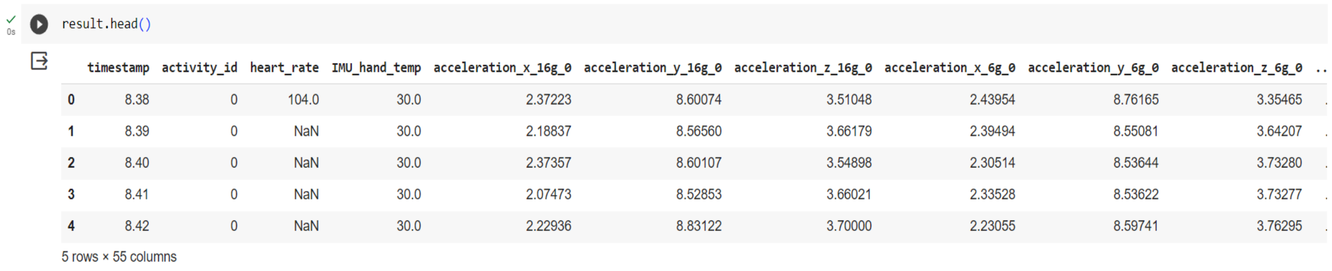
Loading Data

```
# Loading Data

# Reading all csv files and adding one column to identify the person
subject1 = pd.read_csv('subject101.csv', header=None)
subject1[54] = 101
subject2 = pd.read_csv('subject102.csv', header=None)
subject2[54] = 102
subject3 = pd.read_csv('subject103.csv', header=None)
subject3[54] = 103
subject4 = pd.read_csv('subject104.csv', header=None)
subject4[54] = 104
subject5 = pd.read_csv('subject105.csv', header=None)
subject5[54] = 105
subject6 = pd.read_csv('subject106.csv', header=None)
subject6[54] = 106
subject7 = pd.read_csv('subject107.csv', header=None)
subject7[54] = 107
subject8 = pd.read_csv('subject108.csv', header=None)
subject8[54] = 108
subject9 = pd.read_csv('subject109.csv', header=None)
subject9[54] = 109
```

To understand data better and learn the relationship between columns, assigning names to each column based on description of data is essential.



```python
# Giving names to all columns based on description of data

hand = ['IMU_hand_temp','acceleration_x_16g_0', 'acceleration_y_16g_0', 'acceleration_z_16g_0',
                'acceleration_x_6g_0', 'acceleration_y_6g_0', 'acceleration_z_6g_0',
                'gyroscope_x_0', 'gyroscope_y_0', 'gyroscope_z_0',
                'magnetometer_x_0', 'magnetometer_y_0', 'magnetometer_z_0',
                'orientation1_0', 'orientation2_0', 'orientation3_0', 'orientation4_0']

chest = ['IMU_chest_temp','acceleration_x_16g_1', 'acceleration_y_16g_1', 'acceleration_z_16g_1',
                'acceleration_x_6g_1', 'acceleration_y_6g_1', 'acceleration_z_6g_1',
                'gyroscope_x_1', 'gyroscope_y_1', 'gyroscope_z_1',
                'magnetometer_x_1', 'magnetometer_y_1', 'magnetometer_z_1',
                'orientation1_1', 'orientation2_1', 'orientation3_1', 'orientation4_1']

ankle = ['IMU_ankle_temp','acceleration_x_16g_2', 'acceleration_y_16g_2', 'acceleration_z_16g_2',
                'acceleration_x_6g_2', 'acceleration_y_6g_2', 'acceleration_z_6g_2',
                'gyroscope_x_2', 'gyroscope_y_2', 'gyroscope_z_2',
                'magnetometer_x_2', 'magnetometer_y_2', 'magnetometer_z_2',
                'orientation1_2', 'orientation2_2', 'orientation3_2', 'orientation4_2']

column_names = ['timestamp', 'activity_id', 'heart_rate', *hand, *chest, *ankle, 'SubjectID']

# Assign the column names to the DataFrame
result.columns = column_names
```

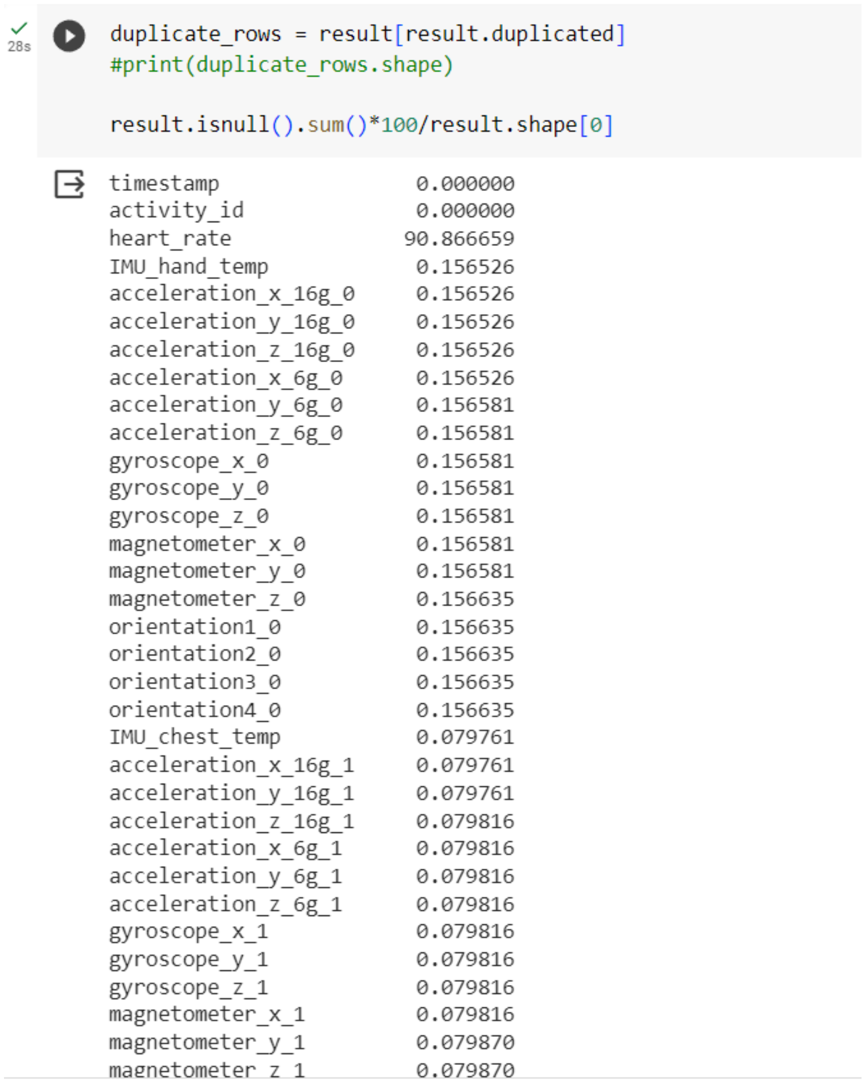


```python
result.head()
```

```
   timestamp  activity_id  heart_rate  IMU_hand_temp  acceleration_x_16g_0  acceleration_y_16g_0  acceleration_z_16g_0  acceleration_x_6g_0  acceleration_y_6g_0  acceleration_z_6g_0 ..
0       8.38            0       104.0           30.0               2.37223               8.60074               3.51048              2.43954              8.76165              3.35465 .
1       8.39            0         NaN           30.0               2.18837               8.56560               3.66179              2.39494              8.55081              3.64207 .
2       8.40            0         NaN           30.0               2.37357               8.60107               3.54898              2.30514              8.53644              3.73280 .
3       8.41            0         NaN           30.0               2.07473               8.52853               3.66021              2.33528              8.53622              3.73277 .
4       8.42            0         NaN           30.0               2.22936               8.83122               3.70000              2.23055              8.59741              3.76295 .
5 rows × 55 columns
```

Task 2 - EDA:

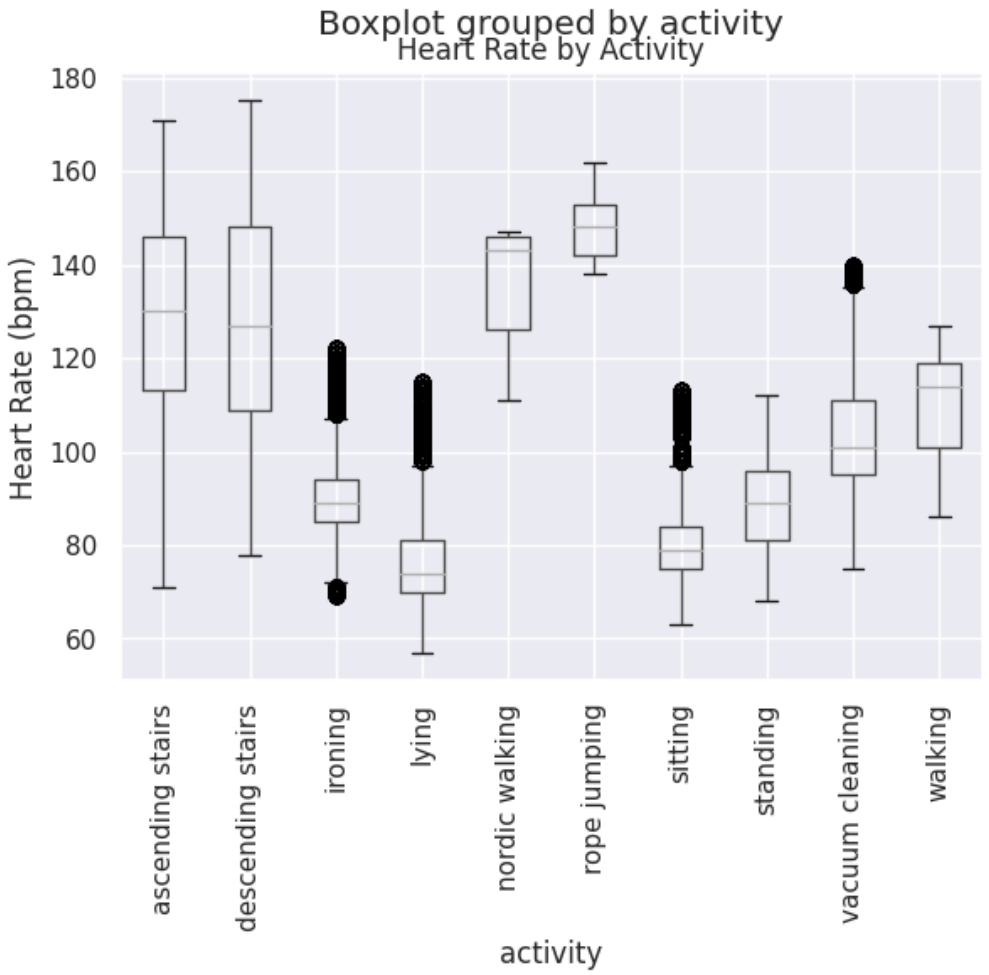
Checking for NA values



```
duplicate_rows = result[result.duplicated]
#print(duplicate_rows.shape)

result.isnull().sum()*100/result.shape[0]
```

```
timestamp               0.000000
activity_id             0.000000
heart_rate             90.866659
IMU_hand_temp           0.156526
acceleration_x_16g_0    0.156526
acceleration_y_16g_0    0.156526
acceleration_z_16g_0    0.156526
acceleration_x_6g_0     0.156526
acceleration_y_6g_0     0.156581
acceleration_z_6g_0     0.156581
gyroscope_x_0           0.156581
gyroscope_y_0           0.156581
gyroscope_z_0           0.156581
magnetometer_x_0        0.156581
magnetometer_y_0        0.156581
magnetometer_z_0        0.156635
orientation1_0          0.156635
orientation2_0          0.156635
orientation3_0          0.156635
orientation4_0          0.156635
IMU_chest_temp          0.079761
acceleration_x_16g_1    0.079761
acceleration_y_16g_1    0.079761
acceleration_z_16g_1    0.079816
acceleration_x_6g_1     0.079816
acceleration_y_6g_1     0.079816
acceleration_z_6g_1     0.079816
gyroscope_x_1           0.079816
gyroscope_y_1           0.079816
gyroscope_z_1           0.079816
magnetometer_x_1        0.079816
magnetometer_y_1        0.079870
magnetometer_z_1        0.079870
```

An examination reveals that 90% of the data in the heart rate column is missing. Given this extensive missing data, it is not advisable to impute the values. Doing so could introduce biases, lead to overfitting in predictive models, and compromise the accuracy of any analysis.



The provided boxplot visualizes the distribution of heart rate values across various activities, representing only the 10% of available heart rate data.

Range of Heart Rates: The overall heart rate values span from around 60 bpm to slightly over 160 bpm.

Interpretation:

Activities with Higher Heart Rates:

- Rope Jumping: This activity shows the highest median heart rate, around 130 bpm, with some values going up to almost 170 bpm, indicating it's a vigorous activity.
- Ascending Stairs: This also has a higher median heart rate, with some values approaching 150 bpm.
- Walking: This activity, surprisingly, shows a wide distribution of heart rates with the upper quartile reaching around 150 bpm.

Activities with Moderate Heart Rates:

- Vacuum Cleaning and Standing: Both these activities have median heart rates slightly above 100 bpm, which indicates a moderate level of exertion.

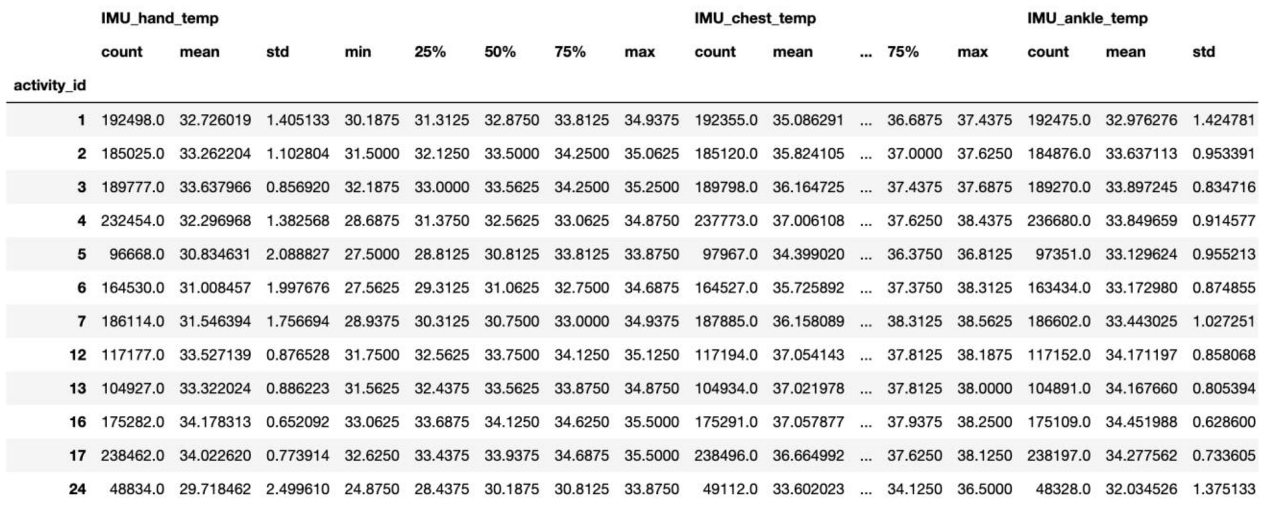
Activities with Lower Heart Rates:

- Lying and Ironing: These activities have the lowest median heart rates, lying around 80-90 bpm, suggesting minimal exertion.
- Sitting, Nordic Walking, and Descending Stairs: These also fall into the lower category but have a slightly broader distribution than lying and ironing.
- Outliers: Several activities, including lying, Nordic walking, and rope jumping, exhibit outliers, which are individual heart rate measurements that fall outside the expected range for that activity.

Data Density: Activities such as lying, rope jumping, and sitting have multiple data points clustered closely, represented by black dots. This suggests that for these activities, the heart rate measurements that are available tend to be similar.

Summary of Data:

- **Finding the relationship between activities and different temperatures**

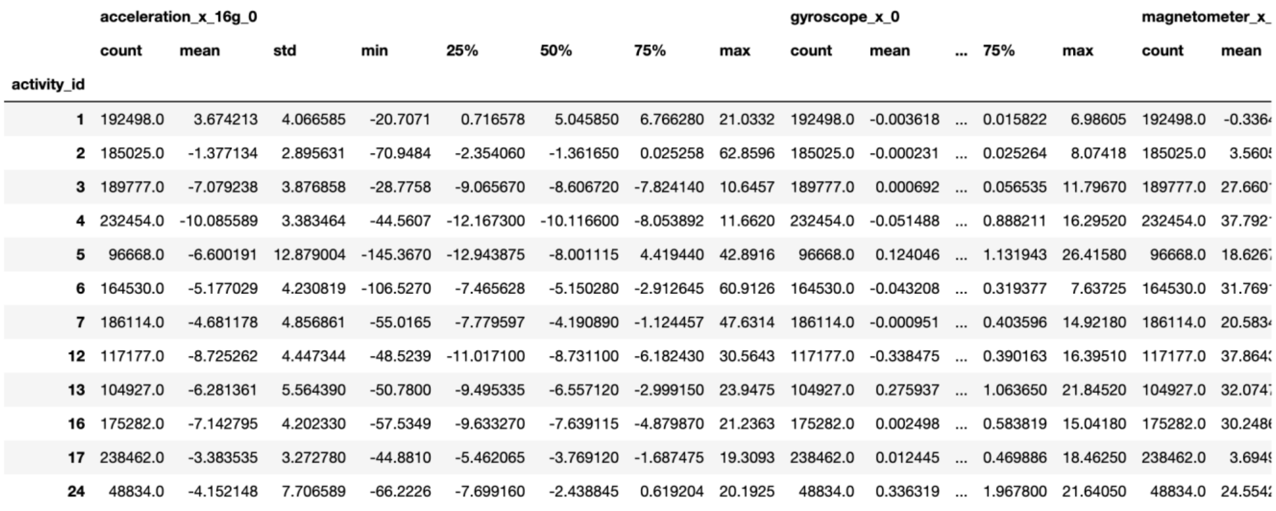


	IMU_hand_temp								IMU_chest_temp					IMU_ankle_temp		
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std
activity_id																
1	192498.0	32.726019	1.405133	30.1875	31.3125	32.8750	33.8125	34.9375	192355.0	35.086291	...	36.6875	37.4375	192475.0	32.976276	1.424781
2	185025.0	33.262204	1.102804	31.5000	32.1250	33.5000	34.2500	35.0625	185120.0	35.824105	...	37.0000	37.6250	184876.0	33.637113	0.953391
3	189777.0	33.637966	0.856920	32.1875	33.0000	33.5625	34.2500	35.2500	189798.0	36.164725	...	37.4375	37.6875	189270.0	33.897245	0.834716
4	232454.0	32.296968	1.382568	28.6875	31.3750	32.5625	33.0625	34.8750	237773.0	37.006108	...	37.6250	38.4375	236680.0	33.849659	0.914577
5	96668.0	30.834631	2.088827	27.5000	28.8125	30.8125	33.8125	33.8750	97967.0	34.399020	...	36.3750	36.8125	97351.0	33.129624	0.955213
6	164530.0	31.008457	1.997676	27.5625	29.3125	31.0625	32.7500	34.6875	164527.0	35.725892	...	37.3750	38.3125	163434.0	33.172980	0.874855
7	186114.0	31.546394	1.756694	28.9375	30.3125	30.7500	33.0000	34.9375	187885.0	36.158089	...	38.3125	38.5625	186602.0	33.443025	1.027251
12	117177.0	33.527139	0.876528	31.7500	32.5625	33.7500	34.1250	35.1250	117194.0	37.054143	...	37.8125	38.1875	117152.0	34.171197	0.858068
13	104927.0	33.322024	0.886223	31.5625	32.4375	33.5625	33.8750	34.8750	104934.0	37.021978	...	37.8125	38.0000	104891.0	34.167660	0.805394
16	175282.0	34.178313	0.652092	33.0625	33.6875	34.1250	34.6250	35.5000	175291.0	37.057877	...	37.9375	38.2500	175109.0	34.451988	0.628600
17	238462.0	34.022620	0.773914	32.6250	33.4375	33.9375	34.6875	35.5000	238496.0	36.664992	...	37.6250	38.1250	238197.0	34.277562	0.733605
24	48834.0	29.718462	2.499610	24.8750	28.4375	30.1875	30.8125	33.8750	49112.0	33.602023	...	34.1250	36.5000	48328.0	32.034526	1.375133

- The mean and standard deviation of the IMU temperature measurements seem to vary across different activities. For example, activity 4 (walking) has lower mean temps and higher std than activity 16 (vacuum cleaning). This indicates the temp measurements are related to activity type.
- The hand IMU temp has a higher mean and std than chest and ankle IMUs for all activities. This suggests the hand temperature is more variable and responsive to activities compared to other body locations.
- The max IMU temperatures follow a similar relative pattern to the means - hand > chest > ankle. So the max temps are related to the mean/variance.
- For all IMUs, the min temp is relatively stable across activities, while mean and max vary more. This indicates the fluctuations in temp from the min baseline are insightful for activity recognition.
- The mean and std of the heart rate also varies by activity type. For intense activities like running, the HR mean and std is higher than for sedentary activities like sitting. So heart rate is related to activity intensity.
- Comparing means and std shows that some activities have more variance in measurements than others. For example, running has a high std compared to its mean across sensors.

So, in summary, the different sensor measurements do appear related to each other and to the activity labels. Analyzing these intra- and inter-feature relationships can provide useful insights for building predictive models.

- **Finding the relationship between activities and acceleration, gyroscope, magnetometer of IMU Hand Sensor**



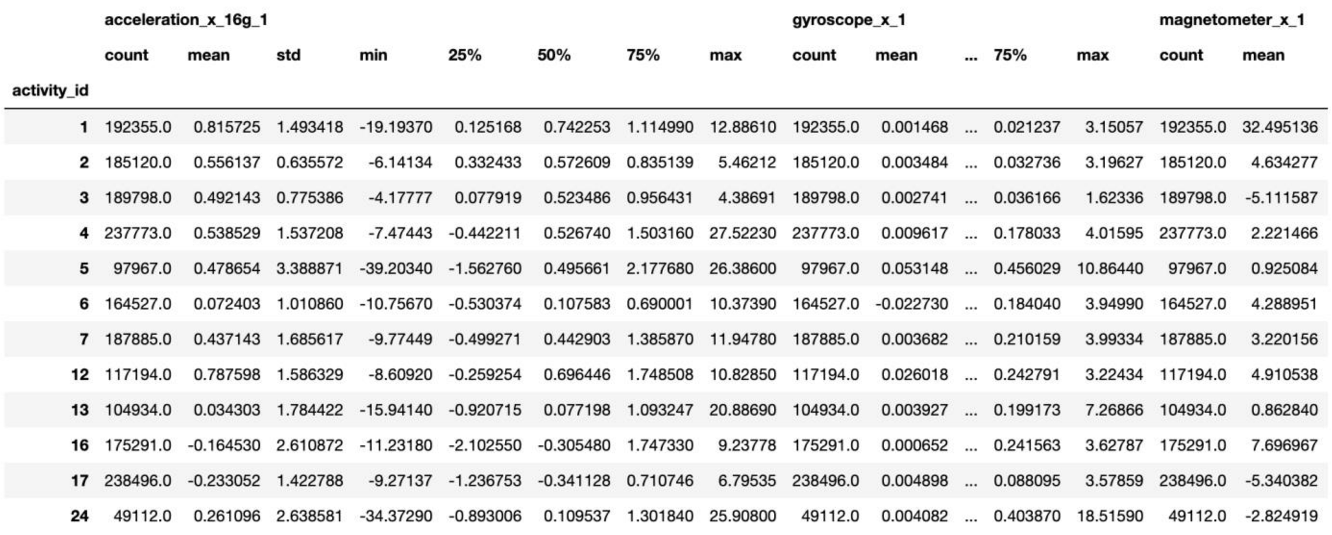
	acceleration_x_16g_0								gyroscope_x_0					magnetometer_x_	
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean
activity_id															
1	192498.0	3.674213	4.066585	-20.7071	0.716578	5.045850	6.766280	21.0332	192498.0	-0.003618	...	0.015822	6.98605	192498.0	-0.336
2	185025.0	-1.377134	2.895631	-70.9484	-2.354060	-1.361650	0.025258	62.8596	185025.0	-0.000231	...	0.025264	8.07418	185025.0	3.560
3	189777.0	-7.079238	3.876858	-28.7758	-9.065670	-8.606720	-7.824140	10.6457	189777.0	0.000692	...	0.056535	11.79670	189777.0	27.660
4	232454.0	-10.085589	3.383464	-44.5607	-12.167300	-10.116600	-8.053892	11.6620	232454.0	-0.051488	...	0.888211	16.29520	232454.0	37.792
5	96668.0	-6.600191	12.879004	-145.3670	-12.943875	-8.001115	4.419440	42.8916	96668.0	0.124046	...	1.131943	26.41580	96668.0	18.626
6	164530.0	-5.177029	4.230819	-106.5270	-7.465628	-5.150280	-2.912645	60.9126	164530.0	-0.043208	...	0.319377	7.63725	164530.0	31.769
7	186114.0	-4.681178	4.856861	-55.0165	-7.779597	-4.190890	-1.124457	47.6314	186114.0	-0.000951	...	0.403596	14.92180	186114.0	20.583
12	117177.0	-8.725262	4.447344	-48.5239	-11.017100	-8.731100	-6.182430	30.5643	117177.0	-0.338475	...	0.390163	16.39510	117177.0	37.864
13	104927.0	-6.281361	5.564390	-50.7800	-9.495335	-6.557120	-2.999150	23.9475	104927.0	0.275937	...	1.063650	21.84520	104927.0	32.074
16	175282.0	-7.142795	4.202330	-57.5349	-9.633270	-7.639115	-4.879870	21.2363	175282.0	0.002498	...	0.583819	15.04180	175282.0	30.248
17	238462.0	-3.383535	3.272780	-44.8810	-5.462065	-3.769120	-1.687475	19.3093	238462.0	0.012445	...	0.469886	18.46250	238462.0	3.694
24	48834.0	-4.152148	7.706589	-66.2226	-7.699160	-2.438845	0.619204	20.1925	48834.0	0.336319	...	1.967800	21.64050	48834.0	24.554

- Acceleration X (16g): Acceleration along the X axis with range ±16g
- Gyroscope X: Angular velocity around the X axis
- Magnetometer X: Magnetic field strength along the X axis

Observing the mean and standard deviation, we can see that:

- The acceleration varies significantly across different activities, with running/walking having larger magnitude than sedentary activities.
- The gyroscope measurements also show more variance for dynamic activities compared to static ones.
- The magnetometer does not seem to have as strong a correlation to activity type based on mean and standard deviation.
- This indicates that the acceleration and gyroscope sensors on the hand IMU provide useful signals related to hand motion and orientation during different activities. These can likely help in activity recognition. The magnetometer data seems less indicative.

- **Finding the relationship between activities and acceleration, gyroscope, magnetometer of IMU Chest Sensor**



	acceleration_x_16g_1								gyroscope_x_1					magnetometer_x_1	
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean
activity_id															
1	192355.0	0.815725	1.493418	-19.19370	0.125168	0.742253	1.114990	12.88610	192355.0	0.001468	...	0.021237	3.15057	192355.0	32.495136
2	185120.0	0.556137	0.635572	-6.14134	0.332433	0.572609	0.835139	5.46212	185120.0	0.003484	...	0.032736	3.19627	185120.0	4.634277
3	189798.0	0.492143	0.775386	-4.17777	0.077919	0.523486	0.956431	4.38691	189798.0	0.002741	...	0.036166	1.62336	189798.0	-5.111587
4	237773.0	0.538529	1.537208	-7.47443	-0.442211	0.526740	1.503160	27.52230	237773.0	0.009617	...	0.178033	4.01595	237773.0	2.221466
5	97967.0	0.478654	3.388871	-39.20340	-1.562760	0.495661	2.177680	26.38600	97967.0	0.053148	...	0.456029	10.86440	97967.0	0.925084
6	164527.0	0.072403	1.010860	-10.75670	-0.530374	0.107583	0.690001	10.37390	164527.0	-0.022730	...	0.184040	3.94990	164527.0	4.288951
7	187885.0	0.437143	1.685617	-9.77449	-0.499271	0.442903	1.385870	11.94780	187885.0	0.003682	...	0.210159	3.99334	187885.0	3.220156
12	117194.0	0.787598	1.586329	-8.60920	-0.259254	0.696446	1.748508	10.82850	117194.0	0.026018	...	0.242791	3.22434	117194.0	4.910538
13	104934.0	0.034303	1.784422	-15.94140	-0.920715	0.077198	1.093247	20.88690	104934.0	0.003927	...	0.199173	7.26866	104934.0	0.862840
16	175291.0	-0.164530	2.610872	-11.23180	-2.102550	-0.305480	1.747330	9.23778	175291.0	0.000652	...	0.241563	3.62787	175291.0	7.696967
17	238496.0	-0.233052	1.422788	-9.27137	-1.236753	-0.341128	0.710746	6.79535	238496.0	0.004898	...	0.088095	3.57859	238496.0	-5.340382
24	49112.0	0.261096	2.638581	-34.37290	-0.893006	0.109537	1.301840	25.90800	49112.0	0.004082	...	0.403870	18.51590	49112.0	-2.824919

Observing the means and standard deviations:

- The chest acceleration means are lower than the hand/ankle IMUs. The variance is also lower, indicating more stationary position.
- The gyroscope means are very small, but again dynamic activities have slightly higher stddevs.

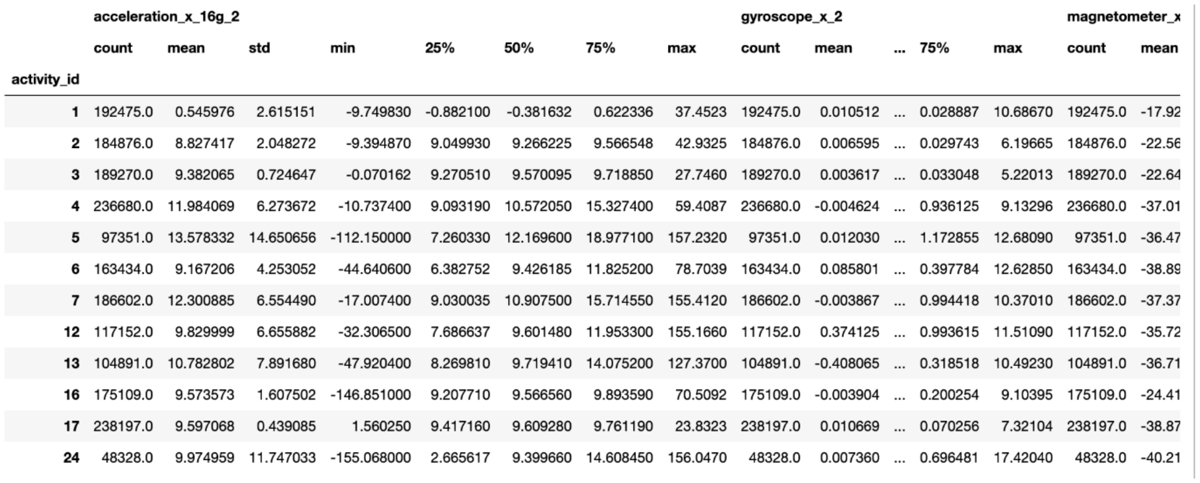
- The magnetometer means vary across activities, but the variance is high even for static activities.

Key observations:

- The chest acceleration shows only minor differences between activity types, as expected for a stationary upper body position.
- The gyroscope provides some motion-related signal, but the variance is lower than the hand/ankle IMUs.
- The magnetometer does not show a strong correlation to activity type. The distributions have high overlap between activities.

To conclude, the chest IMU provides relatively stationary signals compared to the hand/ankle. Some motion information can be extracted from the gyroscope, but overall the chest IMU seems less useful for distinguishing activity types based on the statistical distributions. The hand and ankle IMUs provide clearer motion-related signals.

- **Finding the relationship between activities and acceleration, gyroscope, magnetometer of IMU Ankle Sensor**



	acceleration_x_16g_2								gyroscope_x_2					magnetometer_x	
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean
activity_id															
1	192475.0	0.545976	2.615151	-9.749830	-0.882100	-0.381632	0.622336	37.4523	192475.0	0.010512	...	0.028887	10.68670	192475.0	-17.92
2	184876.0	8.827417	2.048272	-9.394870	9.049930	9.266225	9.566548	42.9325	184876.0	0.006595	...	0.029743	6.19665	184876.0	-22.56
3	189270.0	9.382065	0.724647	-0.070162	9.270510	9.570095	9.718850	27.7460	189270.0	0.003617	...	0.033048	5.22013	189270.0	-22.64
4	236680.0	11.984069	6.273672	-10.737400	9.093190	10.572050	15.327400	59.4087	236680.0	-0.004624	...	0.936125	9.13296	236680.0	-37.01
5	97351.0	13.578332	14.650656	-112.150000	7.260330	12.169600	18.977100	157.2320	97351.0	0.012030	...	1.172855	12.68090	97351.0	-36.47
6	163434.0	9.167206	4.253052	-44.640600	6.382752	9.426185	11.825200	78.7039	163434.0	0.085801	...	0.397784	12.62850	163434.0	-38.89
7	186602.0	12.300885	6.554490	-17.007400	9.030035	10.907500	15.714550	155.4120	186602.0	-0.003867	...	0.994418	10.37010	186602.0	-37.37
12	117152.0	9.829999	6.655882	-32.306500	7.686637	9.601480	11.953300	155.1660	117152.0	0.374125	...	0.993615	11.51090	117152.0	-35.72
13	104891.0	10.782802	7.891680	-47.920400	8.269810	9.719410	14.075200	127.3700	104891.0	-0.408065	...	0.318518	10.49230	104891.0	-36.71
16	175109.0	9.573573	1.607502	-146.851000	9.207710	9.566560	9.893590	70.5092	175109.0	-0.003904	...	0.200254	9.10395	175109.0	-24.41
17	238197.0	9.597068	0.439085	1.560250	9.417160	9.609280	9.761190	23.8323	238197.0	0.010669	...	0.070256	7.32104	238197.0	-38.87
24	48328.0	9.974959	11.747033	-155.068000	2.665617	9.399660	14.608450	156.0470	48328.0	0.007360	...	0.696481	17.42040	48328.0	-40.21

Observing the means and standard deviations:

- The ankle acceleration X has higher means and variance for walking, running compared to sedentary activities. This indicates it captures motion well.

- The gyroscope X means are small, but the variance is higher for dynamic activities.
- The magnetometer X means vary across activities but have high variance even for static activities.

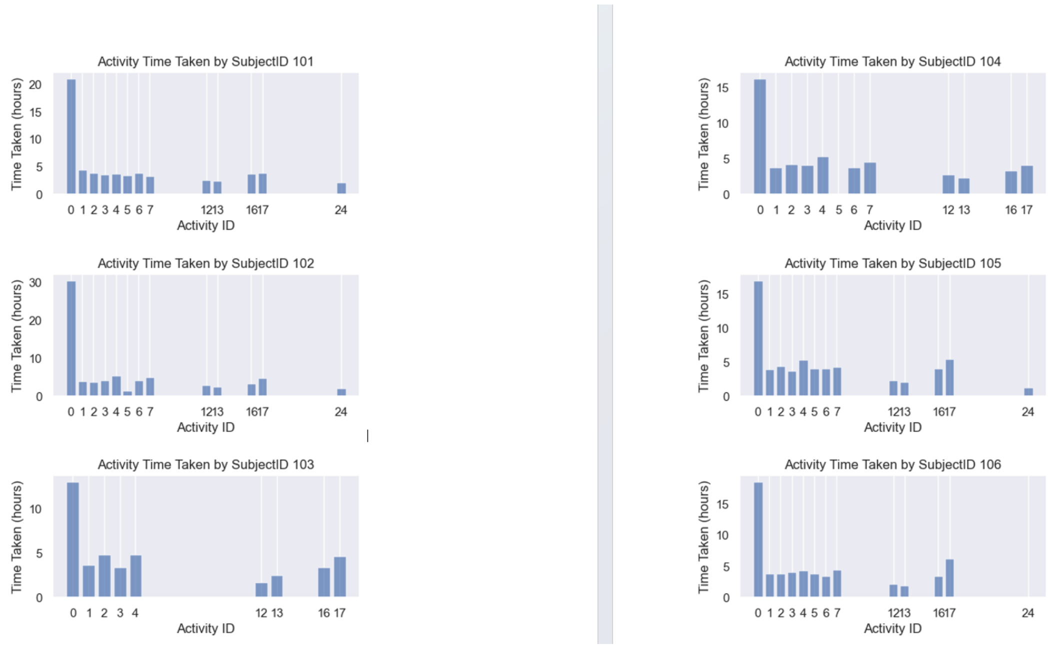
Key observations:

- The acceleration sensor provides the clearest signal related to activity intensity, with walking/running showing larger accelerations on the ankle.
- The gyroscope and magnetometer data have high variance even for static activities, indicating more noise.
- The magnetometer does not seem to provide a clear distinction between activity types based on the statistical distributions.

So, the ankle acceleration X data seems most useful for recognizing motion-related activities from the ankle IMU. The gyroscope provides some additional signal, but the magnetometer data does not show a strong relationship to activity type.

Activity Time Taken by Subjects

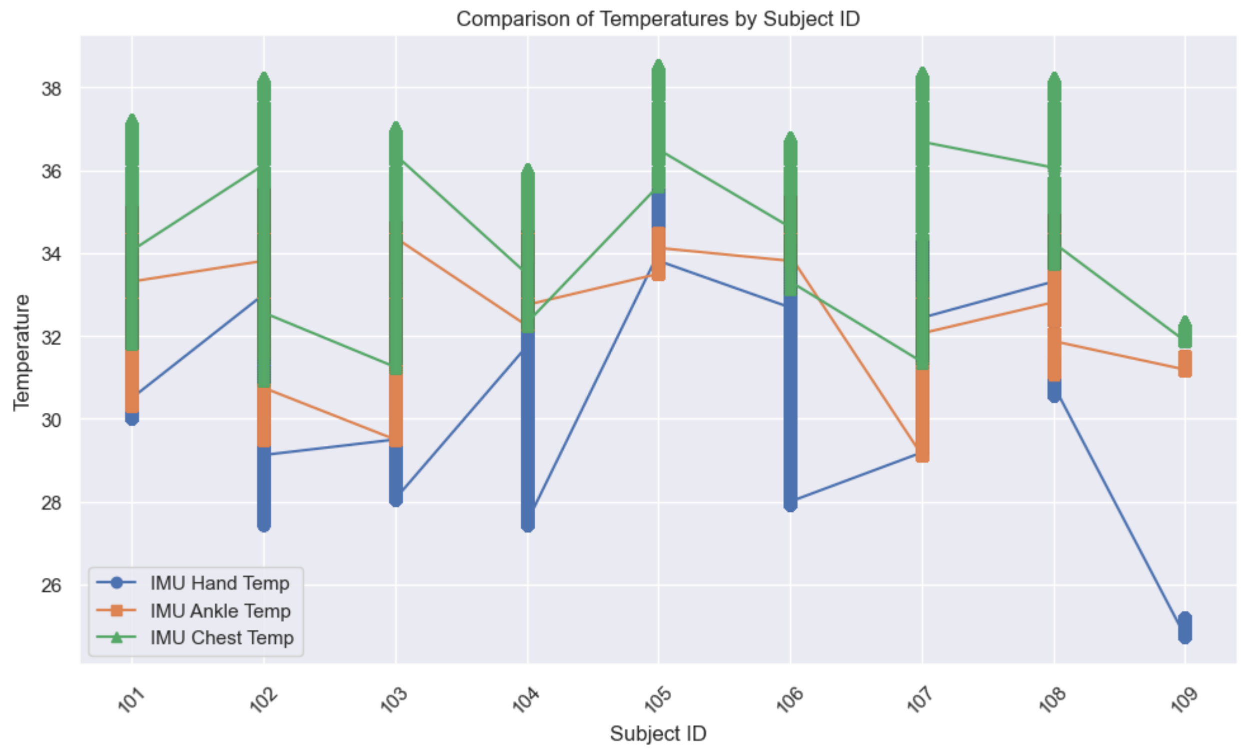
The graphs below illustrate the time taken by each subject to perform the respective activities.



- Highest Time Taken:

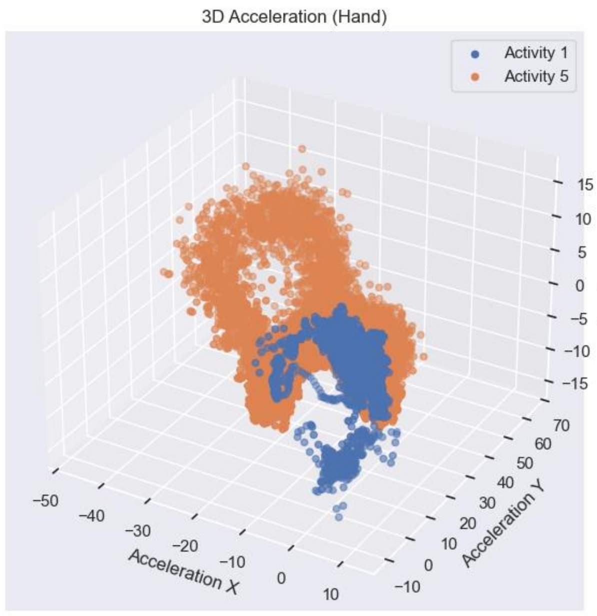
    - Based on the graphs, Activity 3 seems to consistently have one of the highest bars across most subjects. Thus, it likely represents the activity with the highest time taken.
- Lowest Time Taken:
    - Activity 4 and Activity 6 appear to have shorter bars across the majority of subjects, indicating that they might represent activities with the least time taken.
- Common Activities:
    - Activities like 1, 2, and 3 seem to be present across all subjects, suggesting they are common activities.
- Uncommon Activities:
    - Some activities like Activity 5 and Activity 7 don't appear consistently across all subjects. This indicates that they might be uncommon or specific to certain individuals.

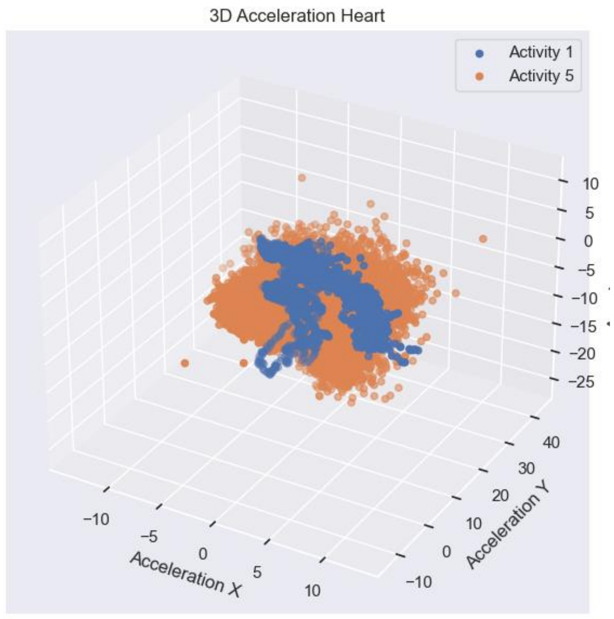
Comparison of temperatures by Subject ID

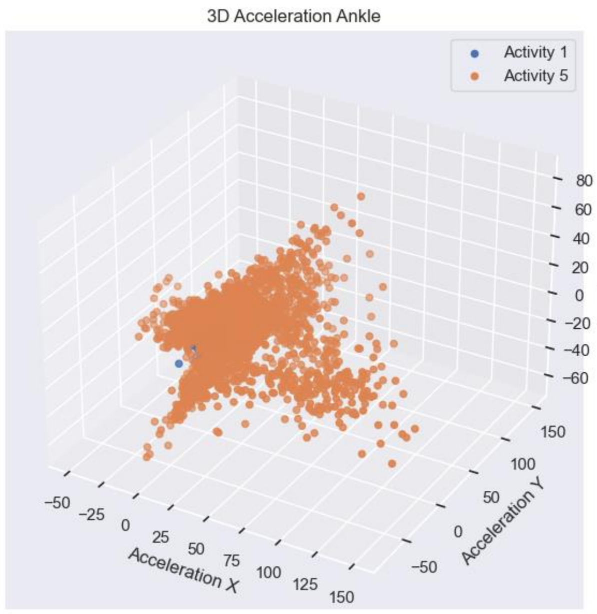


The graph illustrates that the chest IMU temperature is elevated for the majority of subjects. However, subjects 104 and 105 display higher hand IMU temperatures, likely attributed to their engagement in physical tasks like ironing and vacuuming, which potentially increased their hand temperatures.

Capture of acceleration in Hand, Heart, and Ankle sensors







The 3D acceleration heart graph is a useful tool for measuring the intensity and type of physical activity. By comparing the graphs for different activities, we can learn more about how the heart responds to different types of movement. This information can be used to develop personalized exercise programs and to track progress over time.

The differences in the magnitude and pattern of acceleration between the two activities suggest that they are two different activities involving different movements of the heart.

Here are some possible examples of activities that could be represented by the two graphs:

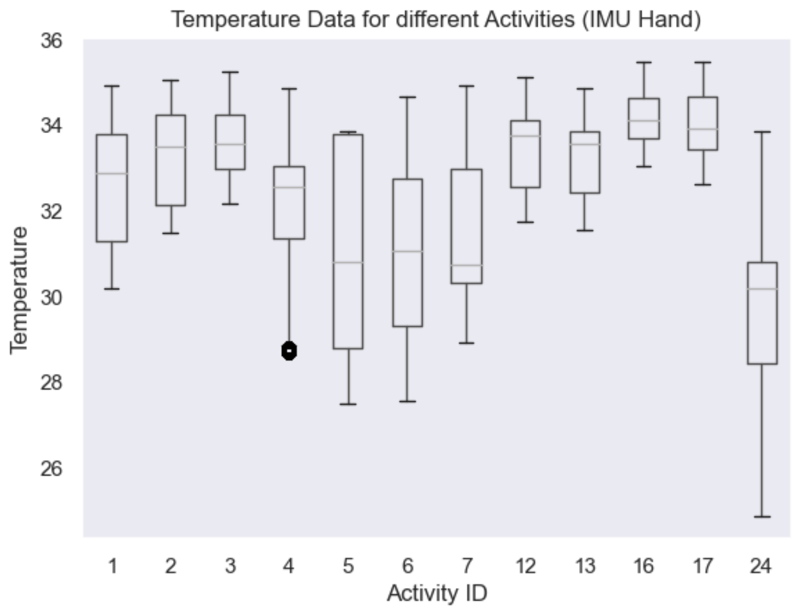
Activity 1: Running, jumping, or lifting weights.
Activity 5: Standing up from a chair, walking slowly, or doing yoga.

The 3D acceleration ankle graphs for activities 1 and 5 show clear differences in the range and pattern of movement. The differences in the range and pattern of movement between the two activities suggest that they are two different activities involving different movements of the ankle joint.

Here are some possible examples of activities that could be represented by the two graphs:

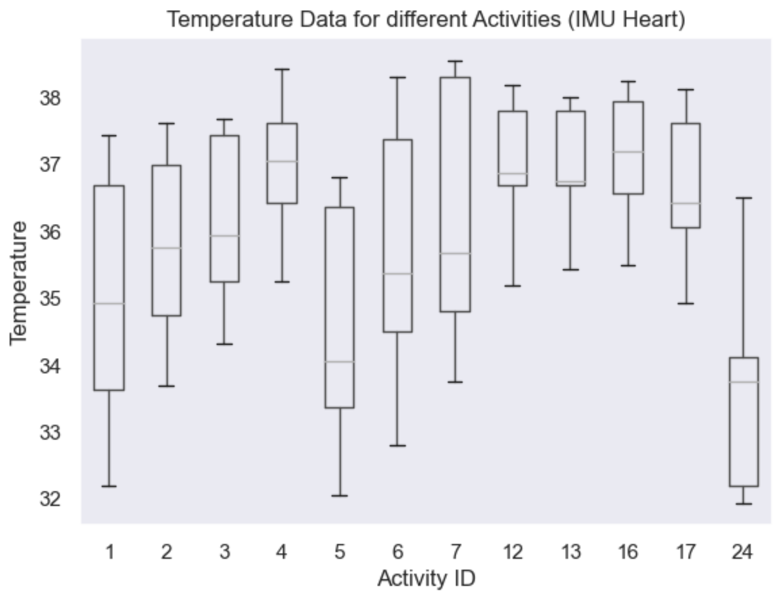
Activity 1: Walking, running, jumping, or climbing stairs.
Activity 5: Kicking a ball, changing direction while running, or dancing.

Temperature Data for different Activities in hand, heart, and ankle sensors.



The plot shows the average temperature for each activity, as well as the minimum and maximum temperatures. The plot does give us some general insights into the temperature changes during different activities. For example, we can see activities 17, 16, and 12 are likely to generate more heat than activities 1, 2, and 3.

- The median temperature for most activities is around 32, with some variations.
- Activity 5 has an outlier with a temperature slightly below 28.
- Activity 7 has the widest range of temperatures, with the data spread out from just above 26 to about 34.
- Activity IDs 16 and 17 have relatively narrower temperature ranges, indicating more consistent data for these activities.
- The box for Activity ID 24 is tall and thin, suggesting a large range of temperature values but fewer data points.

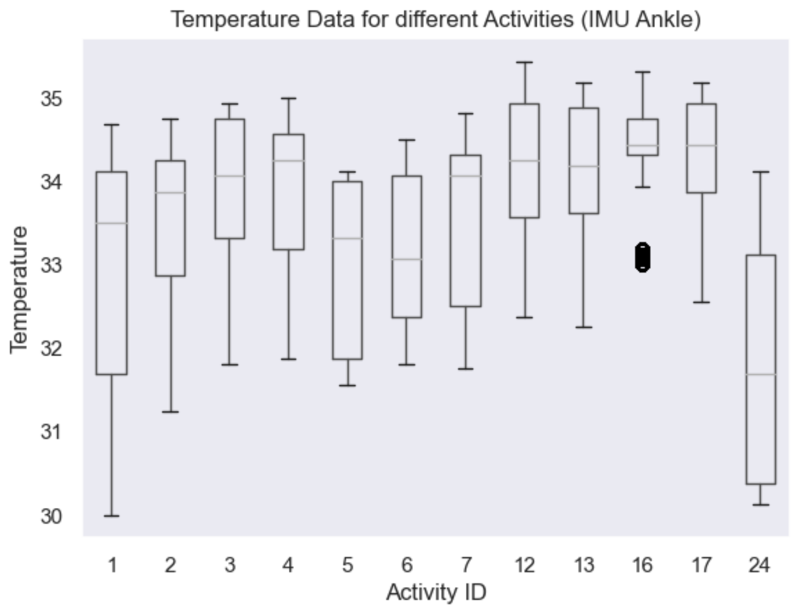


The plot shows the average temperature for each activity, as well as the minimum and maximum temperatures. Activities 17, 16, and 12 are likely to generate more heat than activities 1, 2, and 3.

- Most Outliers:
  - Activity 24 stands out due to it being an outlier box placed much lower than the others, with its median temperature close to 32.
- Lowest Temperature:
  - Activity 6 has the lowest median temperature, and its box is positioned lower compared to most of the other activities.
  - Activity 4 has a whisker that extends to the lowest point, nearing 32, indicating it has a data point with the lowest temperature value.
- Consistency:

- Activity 17 appears to have the most consistent temperature readings, with its box being narrow, indicating less variability in the data.
- On the other hand, Activities 3 and 7 have taller boxes, suggesting more variability or spread in their temperature data.

- General Observations:
    - The median temperatures for most activities lie between 35 and 36.5, with some variations.
    - Most activities show a spread of temperatures that ranges roughly from 34 to 37.



Similarly for ankle, activities 17, 16, and 12 are likely to generate more heat than activities 1, 2, and 3.

- Outliers:
    - Activity 17 stands out with a distinct outlier below its main box, with a temperature close to 30.
    - The box for Activity ID 24 is positioned much lower than the others, with its median temperature nearing 30, making it a noticeable outlier.
- Lowest Temperature:
    - Activity 1 and Activity 24 (considered an outlier) have the lowest median temperatures.
    - Activity 1's box and whisker extend the lowest, indicating it has the coldest readings.
- Consistency:

    - Activity 16 appears to have very consistent temperature readings, indicated by its narrow box.
    - Conversely, Activities 5 and 7 showcase more variability with taller boxes.
- General Observations:
    - The median temperatures for most activities lie between 32 and 34.
    - Most activities show a spread of temperatures ranging approximately from 31 to 35.

The lower temperature observed for Activity 24 (rope jumping) can be attributed to its short duration, which may not have been sufficient to impact the temperature significantly.

Task 3:

The primary motivation behind our selection of this dataset is to delve deeper into the intricate relationship between physical activity and health. By meticulously analyzing this data, we aim to uncover the nuances surrounding the intensity, duration, and patterns of diverse physical activities. Such insights will not only serve to bolster our existing knowledge but also provide evidence-based recommendations for enhancing overall health.

Furthermore, the dataset stands out due to its comprehensive array of features. This extensive feature set not only widens the scope for classification but also adds depth to the study. The activities documented span a gamut of everyday situations, highlighting its immense relevance in real-world contexts. For instance, insights derived from this data can pave the way for:

- Enhancing the precision of fitness tracking apps.
- Developing state-of-the-art assistive technologies to aid individuals with mobility impairments.
- Augmenting tools that track and improve sports performance.

Key Points of Discussion:

Addressing the Elephant in the Room: Missing Heart Rate Data

- The dataset presents a challenge in the form of heart rate data, with a staggering 90% missing values. Such a significant gap complicates any attempts to discern a relationship between heart rate and other variables.
- Making imputations for these missing values can be a slippery slope. For activities such as playing soccer or driving, where heart rate data is entirely absent, imputations might distort the truth.

- Erroneous imputation can lead to misguided hypotheses and interpretations, potentially skewing comparisons between different physical activities.

The Relationship Between Temperature Sensors and Physical Activity

- Temperature sensors can serve as potent tools to deduce patterns associated with physical activities.
- Typically, activities involving robust physical movements manifest as elevated temperatures in specific regions of the body, including but not limited to the hands, heart, and ankles.
- Such temperature elevations can be attributed to the human body's physiological response to activity, where increased circulation and blood flow result in warmer skin temperatures.

However, it's essential to remember that multiple external and internal factors, such as the surrounding temperature, individual variability, and the accuracy of sensors, can sway the readings of temperature sensors.

To truly grasp the dynamics between temperature sensors and physical activity, it becomes imperative to integrate other physiological metrics and contextual data into our analysis.

Task 4:

Data Pre-Processing

1. **Data Cleaning**

```
In [28]: d=result
         filtered_df = d[d['activity_id'].isin([1,2,3,5])]
         filtered_df.groupby('activity_id')[['orientation1_2','orientation2_2','orientation4_2']].value_counts()

Out[28]: activity_id  orientation1_2  orientation2_2  orientation4_2
         1            1.000000         0.000000        0.000000        27184
                      0.000006         0.582468        0.764446            1
                      0.000009        -0.760839       -0.643811            1
                      0.204161         0.345893       -0.510605            1
                      0.204159         0.342653       -0.520528            1
                                                                        ...
         5            0.001732         0.558948        0.769620            1
                      0.001723         0.347479        0.539645            1
                      0.001722        -0.513035       -0.644926            1
                      0.001716        -0.425970       -0.579866            1
                      0.001707        -0.227852       -0.572324            1
         Length: 570675, dtype: int64
```

```
In [29]: # Removing Orientation Columns as it's invalid to our model, this applies the same for subject ID. Now longer needed

         # Columns to remove
         columns_to_remove = ['orientation1_0', 'orientation2_0','orientation3_0','orientation4_0',
                              'orientation1_1','orientation2_1','orientation3_1','orientation4_1',
                              'orientation1_2','orientation2_2','orientation3_2','orientation4_2', 'SubjectID', 'timestamp']

         # Remove specified columns
         df = result.drop(columns=columns_to_remove)
```

After eliminating 12 orientation columns from the IMU data of the hand, heart, and ankle, as well as the subject ID and timestamp columns, our dataset is now streamlined to 40 columns. We opted to remove the orientation columns because, based on the dataset's metadata, they contained invalid data. Moreover, many activities shared identical orientation values, which could potentially mislead our models. The subject ID was removed since each participant performed a majority of the activities, rendering this feature redundant for modeling purposes. Similarly, the timestamp column was deemed unnecessary for our analysis.

Dealing with NA values within dataset:

Are values systematically missing or are they random?

Based on the description provided, “Missing sensory data due to wireless data dropping: missing values are indicated with NaN.

Since data is given every 0.01s (due to the fact, that the IMUs have a sampling frequency of 100Hz), and the sampling frequency of the HR-monitor was only approximately 9Hz, the missing HR-values are also indicated with NaN in the data-files.

Given that the missing data is due to wireless data dropping, and missing values are indicated with NaN, we can consider the following strategies to handle systematically missing data caused by this specific issue:

Imputation Based on Nearby Values:

- Since the missing data might be related to the sampling frequency difference, we can consider imputing missing values based on nearby non-missing values. For instance, we can use linear interpolation to estimate missing values between two known data points.

Impute Based on Activity-Specific Patterns:

- If the missing data patterns vary based on different activities, consider imputing missing values separately for each activity type. This approach might capture activity-specific trends in the data and improve imputation accuracy.

Multiple Imputation:

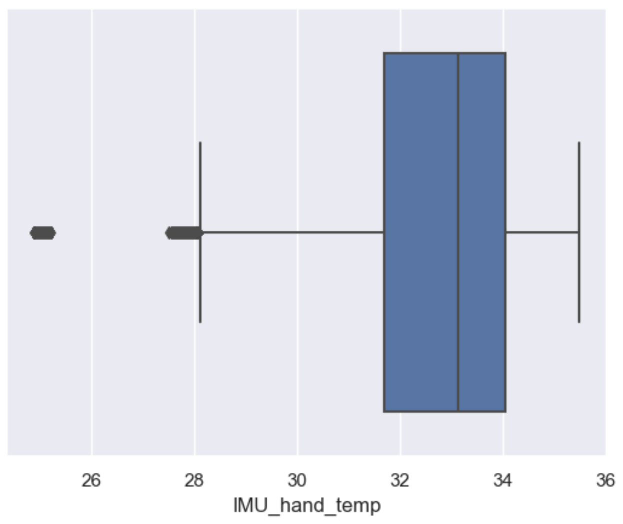
- If we want to capture the uncertainty associated with imputation, we can perform multiple imputation. This approach generates multiple imputed datasets, accounting for the variability in imputation.

After performing imputation based on nearby values, 21441 rows have been imputed. So, there are no missing values left in our dataset.

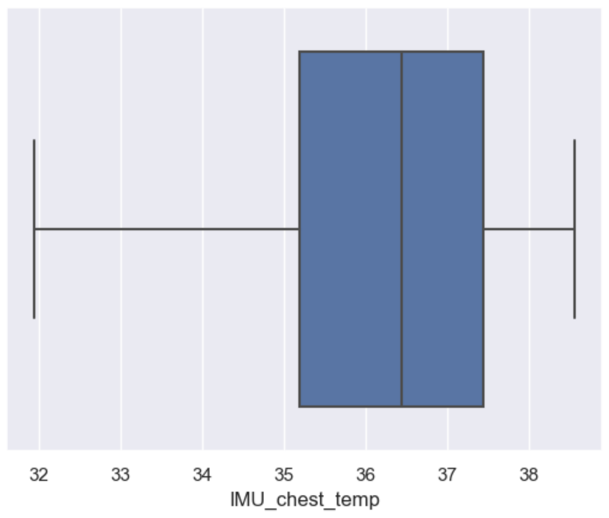
Dealing with outliers:

Outliers could be important to our dataset. So, we must study them closely. As there might be a case where one subject might have a different reaction to performing activity, that can affect our model performance. On the other hand, this might not be true and removing outliers might lead to loss of information. Experimenting with different outlier detection and trying different things and checking the model accuracy might be a valuable option.

Before removal of outliers:

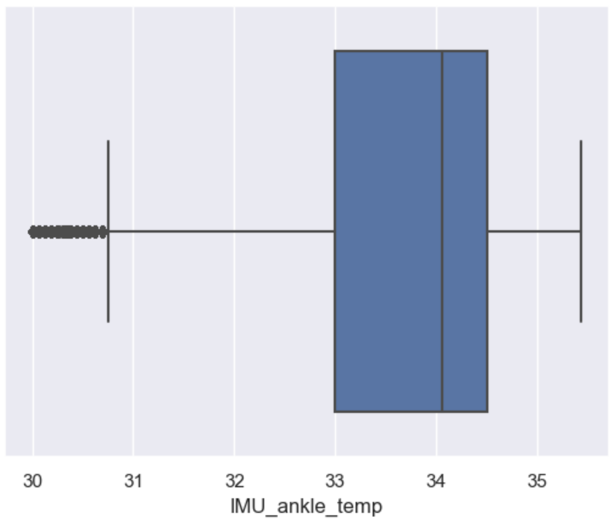


After removal of outliers:

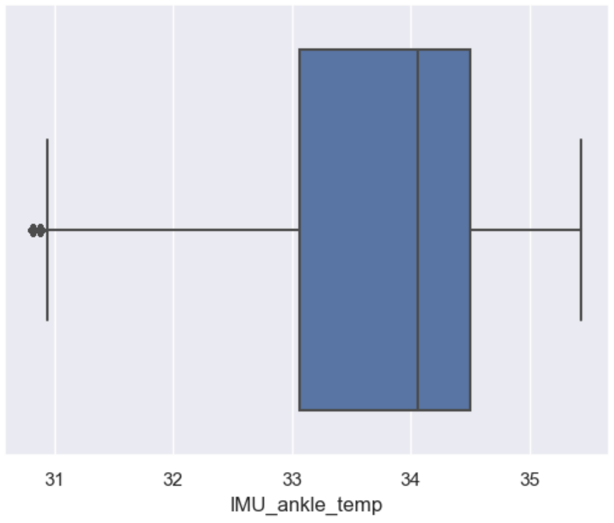


Similarly, for IMU ankle Temp:

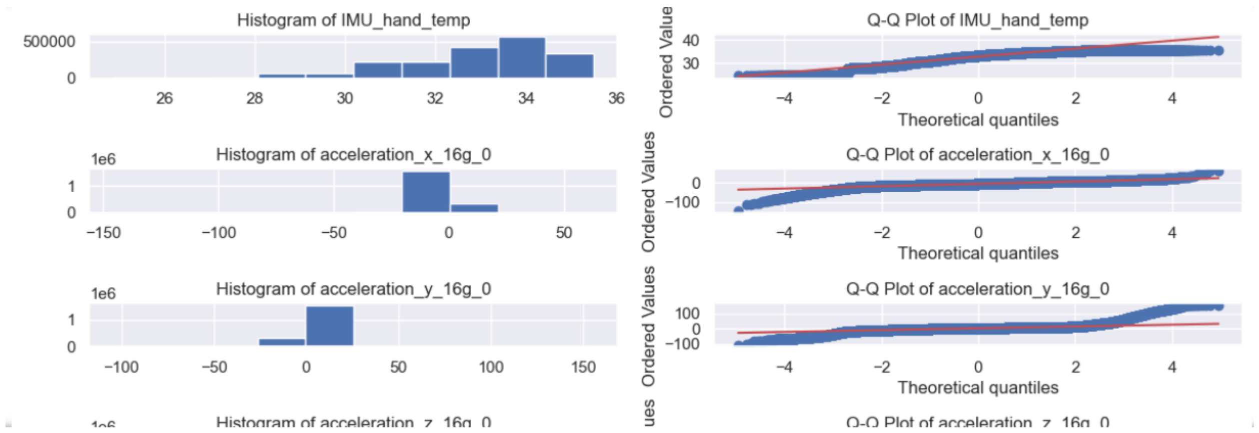
Before removal



After removal



2. Data Transformation:



The columns are predominantly symmetrical, but there are instances where the data exhibits a flat profile with extended plateaus.

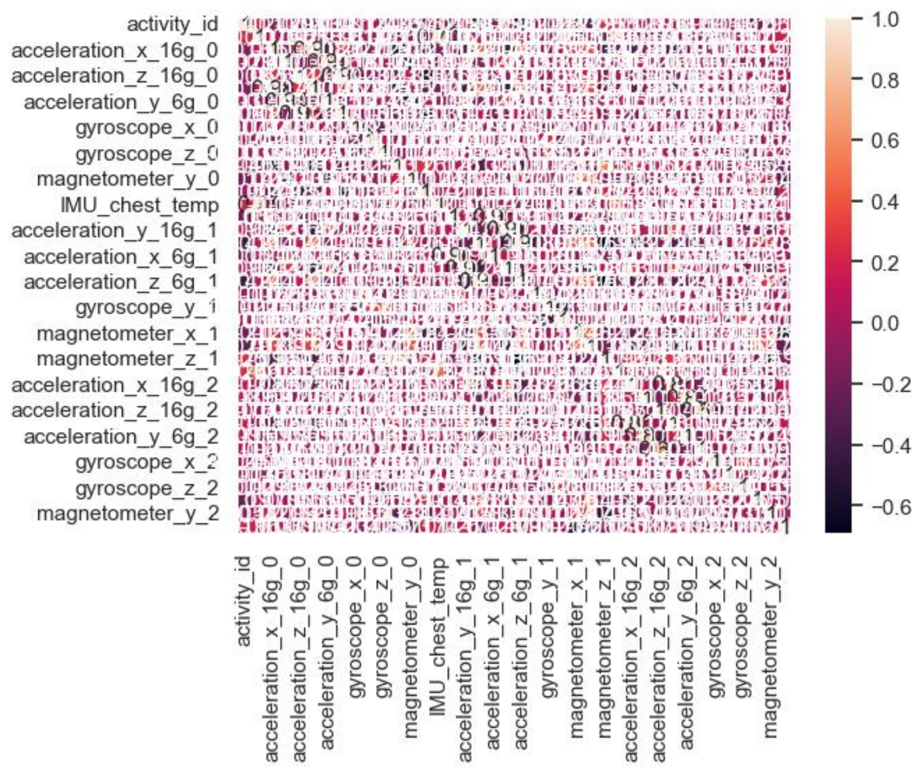
We check the Kurtosis for each temperature,

```
Kurtosis of IMU Hand Temp is: 0.9046565150377663
Kurtosis of IMU Chest Temp is: -0.5123666964849845
Kurtosis of IMU Ankle Temp is: -0.39271016928255564
```

IMU Hand Temp (0.9046): A kurtosis value close to 0 indicates a distribution shape that's relatively similar to a normal distribution. In this case, the tails of the hand temperature distribution might be slightly fatter than a perfect normal distribution.

IMU Chest Temp (-0.5124): A negative kurtosis value signifies that the distribution has lighter tails and a flatter peak than the normal distribution. This means there are fewer extreme values or outliers in the chest temperature data than would be expected in a normal distribution.

IMU Ankle Temp (-0.3927): Similar to the chest temperature, the negative kurtosis value for the ankle temperature also suggests lighter tails than a normal distribution, implying fewer outliers or extreme values in this data as well.



Above is the correlation matrix for all the different columns

These columns have a correlation 0.9 and above: acceleration_x_6g_0', 'acceleration_y_6g_0', 'acceleration_z_6g_0', 'acceleration_x_6g_1', 'acceleration_y_6g_1', 'acceleration_z_6g_1'

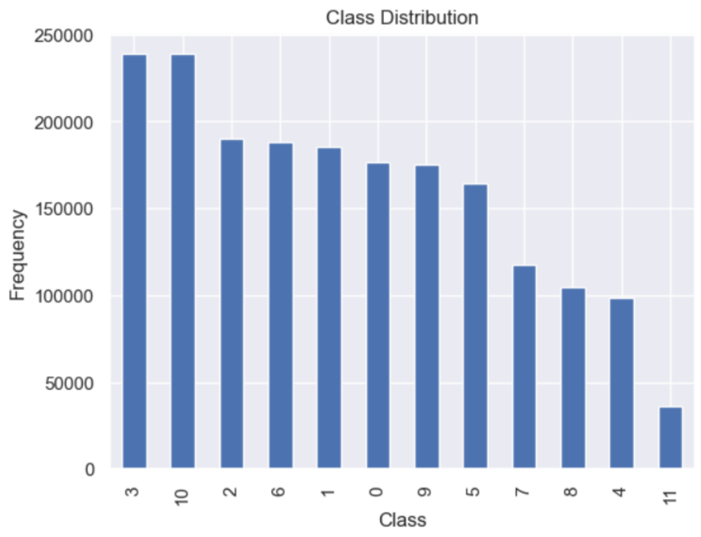
We have to check whether these have any impact on model training or not.

Additionally, we've adjusted the label categories to a range of 0 to 11, streamlining them for easier model interpretation.

```
In [52]: # Changing the label categories in one range from 1 to 12

         #change_cat = {12:8, 13:9,16:10,17:11}

         change_cat = {1:0, 2:1, 3:2, 4:3, 5:4, 6:5, 7:6, 12:7, 13:8, 16:9, 17:10, 24:11}


         df_no_outliers['activity_id'] = (df_no_outliers['activity_id']).replace(change_cat)
         #df_no_outliers['activity_id'] = (df_no_outliers['activity_id']).replace({24:12})
         df_no_outliers['activity_id'].unique()

Out[52]: array([ 0,  1,  2, 10,  9,  7,  8,  3,  6,  5,  4, 11], dtype=int64)
```

We also check for class distribution

```
3     12.477163
10    12.473453
2      9.925407
6      9.830088
1      9.677547
0      9.223374
9      9.163590
5      8.601661
7      6.125469
8      5.484160
4      5.131680
11     1.886408
Name: activity_id, dtype: float64
```



From our observations, activities labeled 3 and 10 exhibit the highest frequency, whereas activity 11 has the least representation.

Task 5:

Before applying any machine learning models, it's essential to partition our dataset into separate subsets: training, testing, and validation. For our current analysis, we've chosen to allocate the data in an 80/10/10 split for the training, testing, and validation sets, respectively. Given the substantial size of our dataset, with over 1 million entries, we've decided to work with a subset to expedite the modeling process. Specifically, we're utilizing only 5% of the training data and 10% of the test data. Additionally, we've employed standardization techniques to ensure that our data has a consistent scale, enhancing the model's accuracy and convergence speed. This standardization step is pivotal, as models often exhibit improved performance when numerical input variables maintain a standard scale and mean.

For this dataset, we have considered **7** machine learning models to be trained and they are as follows:

Employing only Standardization:

Models that we might need to boost in performance:

- Logistic Regression: 74.9%

Models with good performance that we might not need to boost further:

- SVC Model: 93.52%
- XG Boosting: 93.52%

Models with high performance that are likely fine without further boosting (further boosting might lead to overfitting):

- KNN: 94.28%
- Random Forest: 99%
- Decision Tree: 95.79%
- MLP (Feedforward Neural Network): 96.8%

For models in the third category, especially those approaching or exceeding 95% accuracy, we should exercise caution when trying to further improve performance. Over-boosting might cause the models to fit too closely to the training data, reducing their generalization capability on unseen data (overfitting). Using regularization, cross-validation, and a validation set can help us monitor and prevent this from happening.

For the Logistic Regression model, further performance improvement might be beneficial. We can consider hyperparameter tuning, feature engineering, or employing more advanced techniques.

For the models in the second category, while they already have commendable accuracy, depending on the application and the dataset's characteristics, we might decide to tweak or boost them further. However, we should always monitor the model's performance on a validation set to ensure it doesn't start to overfit.

Boosting the Performance: (Feature Engineering)

- **Technique 1: Feature Engineering (Removing Highly Correlated Columns)**

- Logistic Regression: 75% (0.1% increase)
- SVC: 91.5% (2% decrease)
- KNN: 96.1% (2% increase)
- Random Forest: 99% (same)
- Decision Trees: 97.6% (1.81% increase)
- XG Boosting: 93.4% (0.1% decrease)
- MLP (Feedforward Neural Network): 96.8% (0.15% increase)

Analysis:

The SVC model saw a significant decrease in accuracy by 2% after removing highly correlated features. This suggests that some of the correlated features, despite their interrelationships, were quite important predictors for this model.

Most of the other models showed either an improvement or had a negligible impact on performance, with the exception of XGBoost which had a minor decrease.

The models like KNN and Decision Trees benefited from the removal of correlated features, seeing a decent increase in their performance.

While several models did benefit from removing highly correlated features, the substantial decrease in performance for the SVC model raises concerns. It indicates that a blanket removal of all highly correlated features might not be the best approach for every algorithm.

The decision to remove correlated features should be based on a combination of factors: the specific models we're using, the nature of the dataset, and the importance of interpretability versus performance. Given the substantial drop in SVC, it might be worthwhile to consider a more nuanced approach, such as feature selection techniques, to determine which correlated features can be removed without impacting performance negatively.

SVC is known to be computationally intensive, especially with numerous features. In our experiments, removing highly correlated features adversely affected SVC's performance, but enhanced or maintained accuracy for other models, like KNN and Decision Trees. Considering the efficiency gains and the improvements in other models, it's sensible to proceed with the reduced feature set. This approach balances computational cost with model performance across different algorithms.

- **Technique 2: Feature Engineering (Min/Max Scaling)**
    - Logistic Regression: 71.2% (3.6% decrease)
    - SVC: 98.5% (7% increase)
    - KNN: 98.5% (2.5% increase)
    - Random Forest: 99% (same)

- Decision Trees: 97.6% (same)
- XG Boosting: 93.7% (No change noticed)
- MLP (Feedforward Neural Network): 91.5% (5.3% decrease)

Min/Max Scaling Impact: Min/Max scaling transforms the data to fit within a specific range (usually [0, 1]). This can be beneficial for algorithms that are sensitive to the scale of the input features, such as KNN, which relies on distance measures. A consistent scale ensures that all features contribute equally to the distance computation.

After applying Min/Max scaling as a feature engineering technique, Logistic Regression's performance decreased by 3.6%. In contrast, SVC, KNN, and the Feedforward Neural Network (MLP) showed improvements of 7%, 2.5%, and 5.3%, respectively. Random Forest, Decision Trees, and XG Boosting remained unaffected by the scaling. The increases can be attributed to the fact that algorithms like SVC, KNN, and MLP are sensitive to feature scales, and they often perform better when features are on similar scales. However, tree-based methods like Random Forest and XG Boosting are generally indifferent to feature scales, explaining their unchanged performance. The decrease in Logistic Regression might be due to the data's specific distribution or other underlying factors in the dataset after scaling.

There are several reasons we might choose Standardization (Z-score normalization) over Min/Max Scaling, even if Min/Max shows marginally better results in certain scenarios:

Robustness to Outliers: Standardization is less affected by outliers. If our dataset has significant outliers, Min/Max scaling might squeeze the "normal" data into a tiny range. Standardization, on the other hand, will not bound the data to a specific range, allowing for better distribution of data points.

Assumption of Normal Distribution: If our features roughly follow a Gaussian distribution (even if not perfect), standardization maintains this distribution, which can be advantageous for algorithms that assume input features are normally distributed.

Interpretable Features: After standardization, the interpretation of the features becomes somewhat straightforward. A value of 1 indicates one standard deviation above the mean, and -1 indicates one standard deviation below the mean.

Compatibility with Advanced Techniques: Certain algorithms or techniques, like PCA (Principal Component Analysis), work better or assume the input features are standardized. If we plan on using such methods, standardization might be a better choice.

Avoiding Feature Suppression: In Min/Max scaling, if a feature has a variance that's orders of magnitude lower than others, it might get suppressed (essentially becomes a flat line). Standardization doesn't suffer from this.

Generalization Across Datasets: If we're building a model to be used on multiple datasets, Min/Max scaling on one dataset might produce different scales when applied to a new dataset. Standardization is generally more consistent across different datasets.

Consistency with Regularization: If we're using techniques that apply regularization (like L1 or L2), these techniques often assume that all features are centered around zero and have variance in the same order.

- **Technique 3: Grid Search (Best Parameters):**
  - Logistic Regression: 'C': 1, 'penalty': 'l2', 'solver': 'saga'
    - 75% (almost 4% increase)
  - SVC: 'C': 0.696850157946487, 'kernel': 'rbf'
    - 75% (16.5% decrease)
  - KNN: 'weights': 'uniform', 'p': 1, 'n_neighbors': 1
    - 98.6% (4.2% increase)
  - Random Forest: (bootstrap=False, max_depth=None, max_features='sqrt', min_samples_leaf=1, min_samples_split=8, n_estimators=160)
    - 99% (3% increase)
  - Decision Tree: 'max_depth': None, 'min_samples_leaf': 3, 'min_samples_split': 6
    - 95% (Very Slight Decrease)
  - XG Boosting: 'max_depth': 3, 'eta': 0.3, 'objective': 'multi:softprob', 'num_class': 12
    - 93.8% (without optimized parameters 8% accuracy)
  - MLP (Feedforward Neural Network): activation='logistic', alpha=0.0064230583059357955, hidden_layer_sizes=(100,), learning_rate='invscaling', learning_rate_init=0.006357746840747585, max_iter=243, momentum=0.8341209982356952, solver='adam'
    - 98%

The application of Grid Search for hyperparameter tuning largely benefited models like Logistic Regression, KNN, Random Forest, and XGBoost. However, SVC's performance took a substantial hit, and Decision Trees saw a minor decrease. This emphasizes the importance of fine-tuning and the need to be cautious, as not all models will always benefit from the same optimization strategy.

- **Technique 4: Tuning Model Hyperparameters (Manually, FINAL Before Testing)**
  - Logistic Regression: 75% (No Need to change)

  - SVC: 98.7%
  - KNN: 96.8%
  - Random Forest: 96%
  - Decision Trees: 97.6%
  - XG Boosting: 93.4%
  - MLP (Feedforward Neural Network): 98%

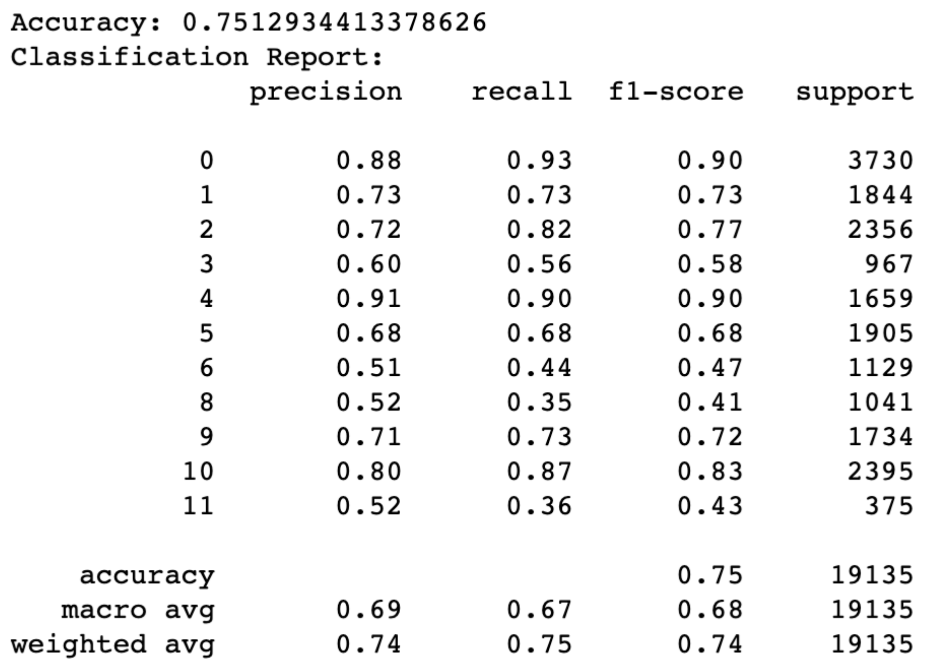
- **Technique 5: Cross-Validation (after tuning) (mean accuracy):**
  - Logistic Regression:
    - Mean Accuracy: 0.7533151048850094
    - Standard Deviation: 0.0013283700235363592
  - SVC:
    - Mean Accuracy: 0.9605189297663499
    - Standard Deviation: 0.0010893959751061407
  - KNN:
    - Mean Accuracy: 0.960074747459541
    - Standard Deviation: 0.0025142042197910082
  - Random Forest:
    - Mean Accuracy: 0.9533334355734973
    - Standard Deviation: 0.002482816263796114
  - Decision Trees:
    - Mean Accuracy: 0.9667768581795622
    - Standard Deviation: 0.001170042613979911
  - XG Boosting:
    - "Took Longer to Run"
  - MLP (Feedforward Neural Network):
    - Mean Accuracy: 0.9788615866686883
    - Standard Deviation: 0.002133470545570884

Overall, tuning makes a huge difference in models, which are complex and have a lot of parameters. However, it is not recommended for very simple models like Logistic Regression, as it will not make that much of difference. Also, tuning should be regulated so it does not overfit the model.

Cross-Validation process shows us that the model is not overfitting as the mean accuracy is quite similar to validation accuracy. The only model that is overfitting is SVC, which we can remove or tune it again. Since, it takes longer to tune and it is computationally expensive, we can ignore it.

Task 6:

1. Evaluation Metrics for the machine learning models
   a. Logistic Regression

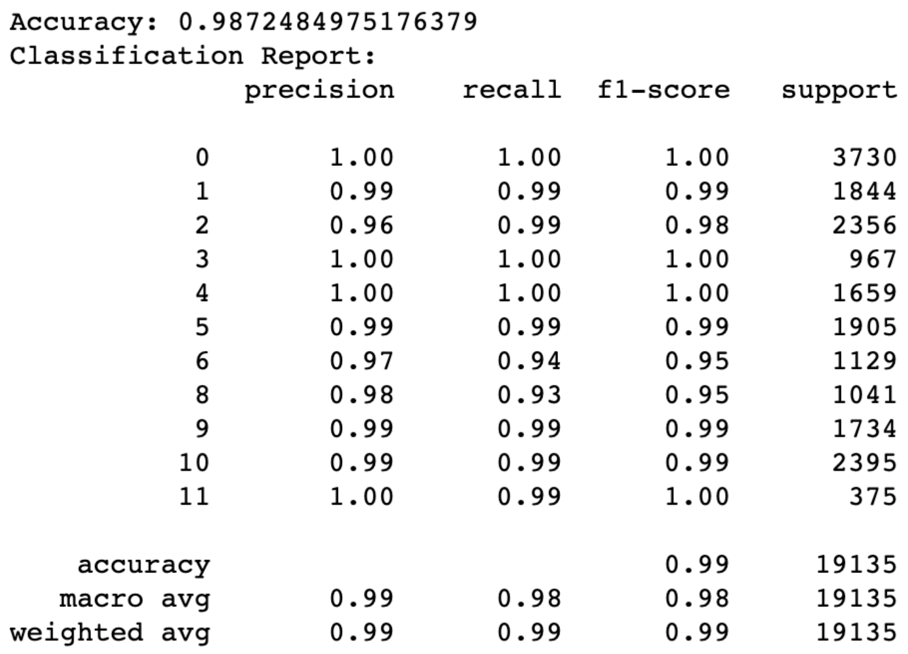


```
Accuracy: 0.7512934413378626
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      3730
           1       0.73      0.73      0.73      1844
           2       0.72      0.82      0.77      2356
           3       0.60      0.56      0.58       967
           4       0.91      0.90      0.90      1659
           5       0.68      0.68      0.68      1905
           6       0.51      0.44      0.47      1129
           8       0.52      0.35      0.41      1041
           9       0.71      0.73      0.72      1734
          10       0.80      0.87      0.83      2395
          11       0.52      0.36      0.43       375

    accuracy                           0.75     19135
   macro avg       0.69      0.67      0.68     19135
weighted avg       0.74      0.75      0.74     19135
```

The Logistic Regression model has an overall accuracy of approximately 75.13%. Across the 12 classes (0 to 11), the model performs variably:

Best Performing Class: Class 4 has high precision (0.91), recall (0.90), and F1-score (0.90).
Least Performing Class: Class 6 struggles with lower precision (0.51), recall (0.44), and F1-score (0.47).
The average metrics across all classes are: precision of 0.69, recall of 0.67, and an F1-score of 0.68.

b. SVC Model: Computationally Expensive



```
Accuracy: 0.9872484975176379
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3730
           1       0.99      0.99      0.99      1844
           2       0.96      0.99      0.98      2356
           3       1.00      1.00      1.00       967
           4       1.00      1.00      1.00      1659
           5       0.99      0.99      0.99      1905
           6       0.97      0.94      0.95      1129
           8       0.98      0.93      0.95      1041
           9       0.99      0.99      0.99      1734
          10       0.99      0.99      0.99      2395
          11       1.00      0.99      1.00       375

    accuracy                           0.99     19135
   macro avg       0.99      0.98      0.98     19135
weighted avg       0.99      0.99      0.99     19135
```

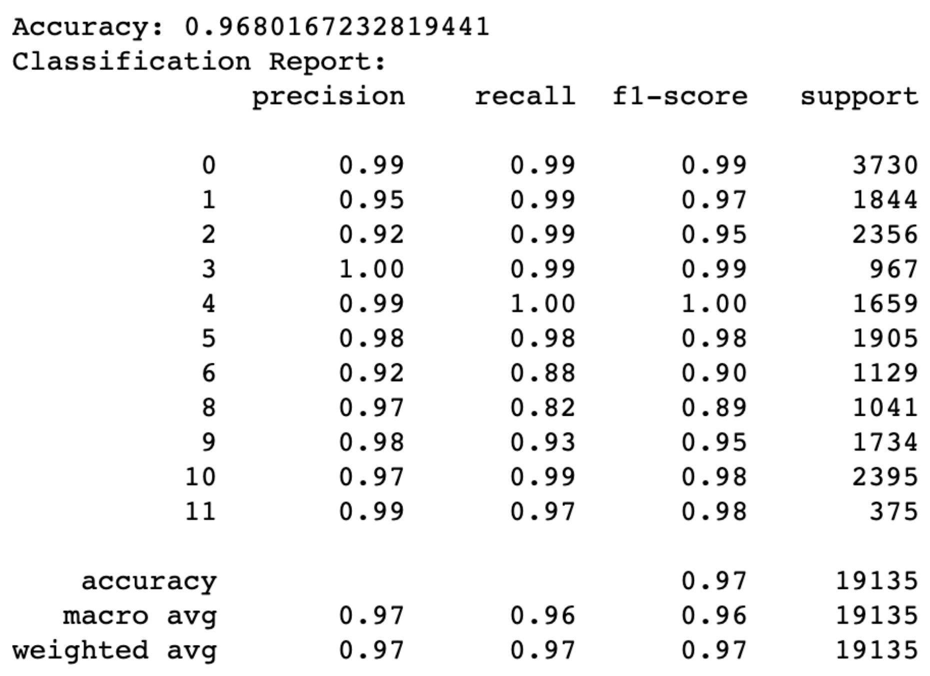
The SVC model boasts a high overall accuracy of 98.72%. Observing the classification metrics across the 12 classes (0 to 11):

Top Performance: Classes 0, 3, 4, and 11 all exhibit perfect precision, recall, and F1-scores of 1.00.

Slight Challenges: Class 6 has the lowest recall of 0.94, and class 8 has the lowest precision of 0.98, but both still maintain commendably high metrics.

The average metrics (precision, recall, F1-score) for all classes hover around 0.99, showcasing the model's strong and consistent performance across the dataset.

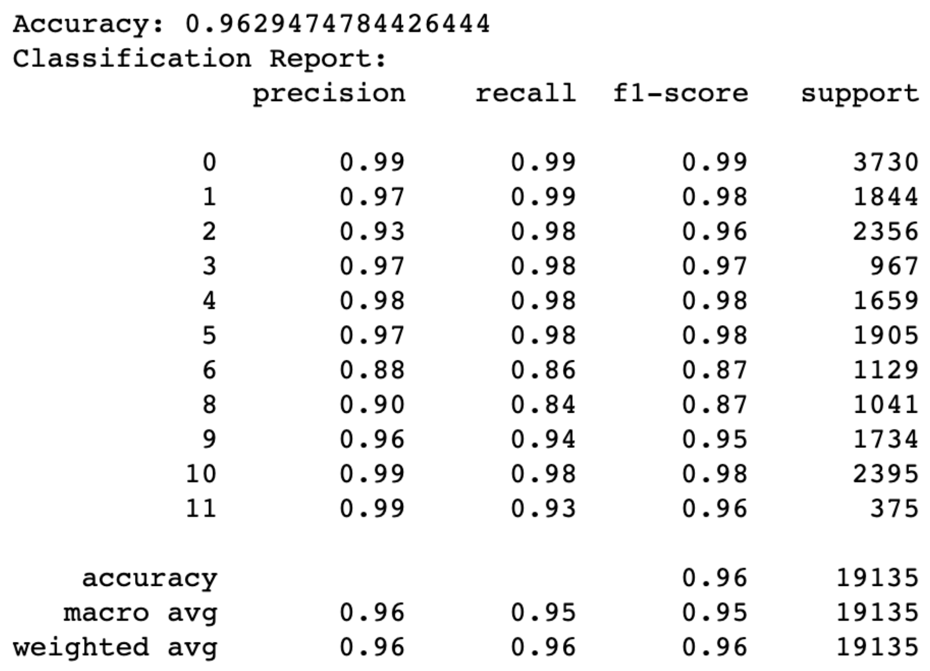
c. KNN: Not recommended for the whole dataset



```
Accuracy: 0.9680167232819441
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3730
           1       0.95      0.99      0.97      1844
           2       0.92      0.99      0.95      2356
           3       1.00      0.99      0.99       967
           4       0.99      1.00      1.00      1659
           5       0.98      0.98      0.98      1905
           6       0.92      0.88      0.90      1129
           8       0.97      0.82      0.89      1041
           9       0.98      0.93      0.95      1734
          10       0.97      0.99      0.98      2395
          11       0.99      0.97      0.98       375

    accuracy                           0.97     19135
   macro avg       0.97      0.96      0.96     19135
weighted avg       0.97      0.97      0.97     19135
```

The KNN model showcases an overall accuracy of 96.80%. Delving into the classification metrics for the 12 classes (0 to 11):
Strong Performance: Classes 3 and 4 achieve perfect precision, recall, and F1-scores of 1.00, representing impeccable classification.
Areas of Improvement: Class 6 has a precision of 0.92 and a slightly lower recall of 0.88. Class 8 demonstrates the lowest recall of 0.82, reflecting its increased challenge in correctly identifying its true positives.
The average metrics (precision, recall, F1-score) across all classes are approximately 0.97, indicating a consistently high performance for the KNN model on this dataset.

d. Random Forest: Model Complexity can lead to overfit.



```
Accuracy: 0.9629474784426444
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3730
           1       0.97      0.99      0.98      1844
           2       0.93      0.98      0.96      2356
           3       0.97      0.98      0.97       967
           4       0.98      0.98      0.98      1659
           5       0.97      0.98      0.98      1905
           6       0.88      0.86      0.87      1129
           8       0.90      0.84      0.87      1041
           9       0.96      0.94      0.95      1734
          10       0.99      0.98      0.98      2395
          11       0.99      0.93      0.96       375

    accuracy                           0.96     19135
   macro avg       0.96      0.95      0.95     19135
weighted avg       0.96      0.96      0.96     19135
```

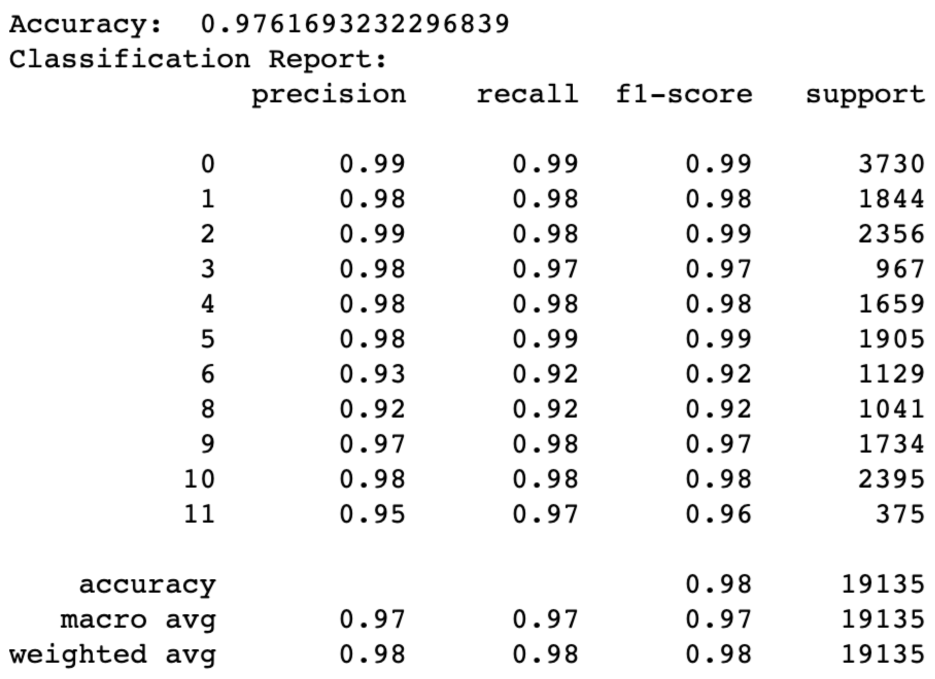
The Random Forest model has an overall accuracy of approximately 96.29%. Breaking down the classification metrics across the classes (0 to 11):

Best Performance: Classes 0, 4, 5, 10, and 11 showcase impressive results with both precision and recall nearing 0.99, demonstrating strong classification capabilities for these classes.

Room for Improvement: Class 6 sees a precision of 0.88 and a slightly lower recall of 0.86, making it the class with the most potential for refinement.

The average metrics (precision, recall, F1-score) for all classes hover around 0.96, underscoring the model's consistent and high performance on this dataset.

e. Decision Tree:

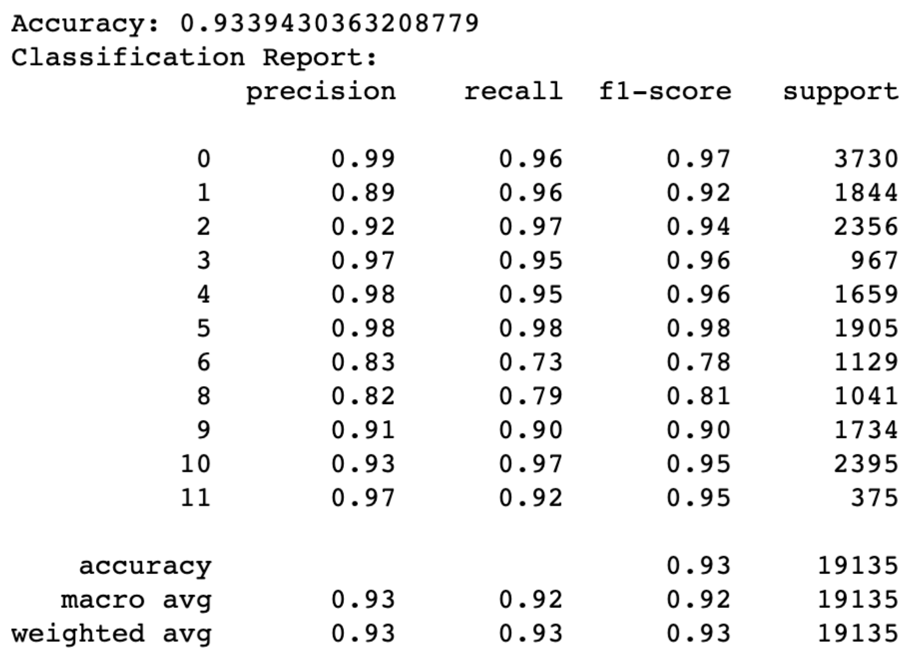


```
Accuracy:  0.9761693232296839
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3730
           1       0.98      0.98      0.98      1844
           2       0.99      0.98      0.99      2356
           3       0.98      0.97      0.97       967
           4       0.98      0.98      0.98      1659
           5       0.98      0.99      0.99      1905
           6       0.93      0.92      0.92      1129
           8       0.92      0.92      0.92      1041
           9       0.97      0.98      0.97      1734
          10       0.98      0.98      0.98      2395
          11       0.95      0.97      0.96       375

    accuracy                           0.98     19135
   macro avg       0.97      0.97      0.97     19135
weighted avg       0.98      0.98      0.98     19135
```

The Decision Tree model exhibits an overall accuracy of about 97.62%. Analyzing the classification metrics for the classes (0 to 11):

Top Performance: Classes 0, 2, and 5 display near-perfect results with precision, recall, and F1-scores around 0.99, indicating excellent classification capabilities for these categories.
Areas for Refinement: Class 6 and 8 have slightly lower metrics with a precision of 0.93 and 0.92 respectively, and corresponding recall rates.
The average metrics (precision, recall, F1-score) for all classes are around 0.97, showcasing a consistent and robust performance of the Decision Tree model across the dataset.

f. XGB: Model Complexity might lead to overfit if not tuned properly



```
Accuracy: 0.9339430363208779
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97      3730
           1       0.89      0.96      0.92      1844
           2       0.92      0.97      0.94      2356
           3       0.97      0.95      0.96       967
           4       0.98      0.95      0.96      1659
           5       0.98      0.98      0.98      1905
           6       0.83      0.73      0.78      1129
           8       0.82      0.79      0.81      1041
           9       0.91      0.90      0.90      1734
          10       0.93      0.97      0.95      2395
          11       0.97      0.92      0.95       375

    accuracy                           0.93     19135
   macro avg       0.93      0.92      0.92     19135
weighted avg       0.93      0.93      0.93     19135
```

For the XGB (XGBoost) model:

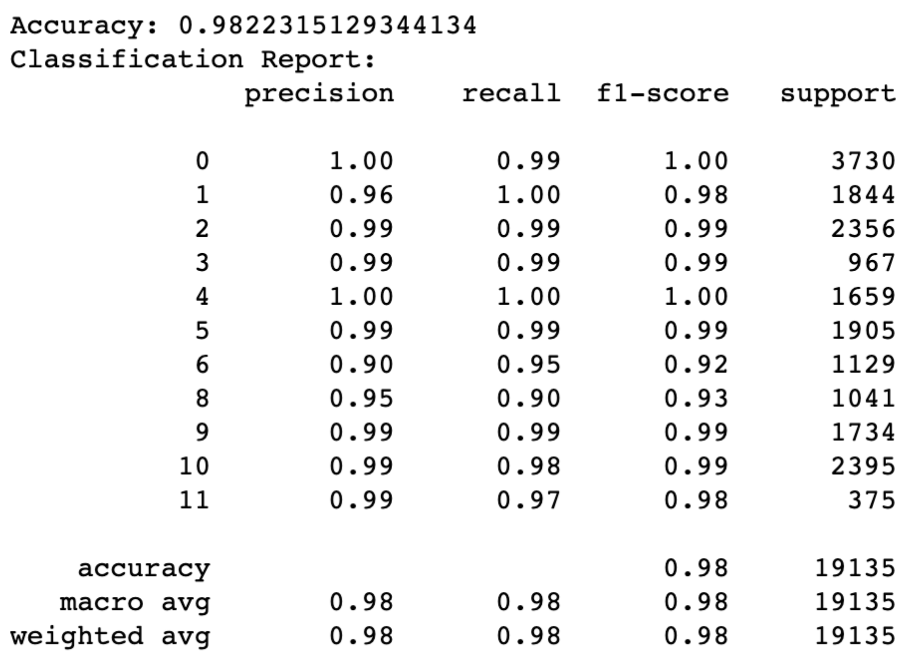
Overall Accuracy: The model achieves an accuracy of approximately 93.39%.

Top Performance: Class 0 stands out with precision, recall, and F1-scores nearing 0.97, indicating its strong classification capabilities.

Room for Improvement: Class 6 has a precision of 0.83 and a more reduced recall of 0.73, which is the most noticeable area where the model's predictions can be enhanced.

Average Metrics: The model's average precision, recall, and F1-score across all classes (macro and weighted) hover around 0.93, illustrating its consistent performance on the dataset.

g. Feedforward Neural Network (MLP)



```
Accuracy: 0.9822315129344134
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      3730
           1       0.96      1.00      0.98      1844
           2       0.99      0.99      0.99      2356
           3       0.99      0.99      0.99       967
           4       1.00      1.00      1.00      1659
           5       0.99      0.99      0.99      1905
           6       0.90      0.95      0.92      1129
           8       0.95      0.90      0.93      1041
           9       0.99      0.99      0.99      1734
          10       0.99      0.98      0.99      2395
          11       0.99      0.97      0.98       375

    accuracy                           0.98     19135
   macro avg       0.98      0.98      0.98     19135
weighted avg       0.98      0.98      0.98     19135
```

Overall Accuracy: The model achieves an accuracy of approximately 98.22%.

Outstanding Performance: Classes 0, 4, and 5 exhibit impeccable results with precision, recall, and F1-scores of 1.00, indicating flawless classification for these classes.

Areas of Refinement: Class 6, while still performing well, has a precision of 0.90 and a recall of 0.95, making it slightly less consistent compared to some other classes.

Average Metrics: The model's average precision, recall, and F1-score across all classes (both macro and weighted) are around 0.98, suggesting a consistently strong performance for the MLP model on this dataset.

2. Evaluations

Models	**Accuracy (without outliers)**	**Feature Engineering**	**Min/Max Scaling**	**Grid Search**
1. Logistic Regression	74.74%	**75%**	**71.2%**	**75%**
2. SVC	93.52%	**91.5%**	**98.5%**	**75%**

3. KNN	94.28%	**96.1%**	**98.5%**	**98.6%**
4. Decision Tree	95.54%	**97.6%**	**97.6%**	**95%**
5. Random Forest	99%	**99%**	**99%**	**99%**
6. XG Boost	93.52%	**93.4%**	**93.7%**	**93.8%**
7. MLP	97.41%	**96.8%**	**91.5%**	**98%**

Logistic Regression: Model Logistic Regression with Optimization Grid Search (Best Parameters) after preprocessing step Standardization was the most effective because it led to an increase of almost 4% in accuracy.

SVC: Model SVC with Optimization Feature Engineering (Min/Max Scaling) after preprocessing step Standardization was the most effective because it resulted in a significant increase of 7% in accuracy.

KNN: Model KNN with Optimization Grid Search (Best Parameters) after preprocessing step Standardization was the most effective because it caused a 4.2% increase in accuracy.

Random Forest: Model Random Forest with Optimization Grid Search (Best Parameters) after preprocessing step Standardization was the most effective because it resulted in a 3% increase in accuracy.

Decision Trees: Model Decision Trees with Optimization Feature Engineering (Removing Highly Correlated Columns) after preprocessing step Standardization was the most effective because it led to a 1.81% increase in accuracy.

XG Boosting: Model XG Boosting with Optimization Grid Search (Best Parameters) after preprocessing step Standardization was the most effective because it provided an increase in accuracy when optimized parameters were used, compared to without them.

MLP (Feedforward Neural Network): Model MLP (Feedforward Neural Network) with Optimization Grid Search (Best Parameters) after preprocessing step Standardization was the most effective because it maintained a high accuracy level similar to the initial setup.

For the least effective:

Logistic Regression: Model Logistic Regression with Optimization Feature Engineering (Min/Max Scaling) after preprocessing step Standardization was the least effective because it led to a decrease of 3.6% in accuracy.

SVC: Model SVC with Optimization Grid Search (Best Parameters) after preprocessing step Standardization was the least effective because it resulted in a significant decrease of 16.5% in accuracy.

MLP (Feedforward Neural Network): Model MLP (Feedforward Neural Network) with Optimization Feature Engineering (Min/Max Scaling) after preprocessing step Standardization was the least effective because it led to a decrease of 5.3% in accuracy.

Logistic Regression:

- Why Feature Engineering (Min/Max Scaling) didn't work well: Logistic Regression assumes that the features are normally distributed. Min/Max scaling might distort this distribution, especially if the features have outliers. This could lead to the model not working as expected.

SVC (Support Vector Machine):

- Why Feature Engineering (Removing Highly Correlated Columns) didn't work well: SVC tries to find the optimal hyperplane to separate classes. By removing highly correlated columns, we might be removing some information that the original SVC model was leveraging.
- Why Grid Search (Best Parameters) didn't work well: It's possible that the parameters identified as "best" during the grid search process weren't optimal for

the entire dataset or the specific nature of this problem, causing a significant drop in accuracy.

KNN (K-Nearest Neighbors):

Why it worked well with Min/Max Scaling: KNN relies on distances between points. Min/Max scaling ensures all features have the same scale, making distance calculations more meaningful and consistent across features.

Random Forest:

Why its performance remained consistent: Random Forest is an ensemble method that is generally robust to different feature scales and irrelevant features. Its consistent performance suggests that it's extracting meaningful patterns from the data regardless of the preprocessing step.

Decision Trees:

Why Feature Engineering (Removing Highly Correlated Columns) worked well: Decision trees might overfit to certain features. By removing highly correlated columns, we reduce the chance of the tree focusing too heavily on redundant features, potentially improving generalization.

XG Boosting:

Why Grid Search (Best Parameters) worked well: Gradient boosting methods like XGBoost can benefit greatly from hyperparameter tuning. The specific parameters found during the grid search might have aligned better with the underlying data distribution and the problem's characteristics.

MLP (Feedforward Neural Network):

Why Feature Engineering (Min/Max Scaling) didn't work well: Neural networks benefit from normalized data, but min/max scaling might not be the ideal normalization method for this dataset or for the specific architecture of the MLP used. Additionally, removing the effect of outliers or shifting the data distribution might have affected the network's ability to generalize.

Test set: As a sample, only two models were selected for test as time did not allow for more models to be tested.

91]:
```
y_pred = logistic_model.predict(test20_x)

# Evaluate and display the model's accuracy
accuracy = accuracy_score(test20_y, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7547820633427407
```

93]:
```
# Predict
dval = xgb.DMatrix(test20_x, label=test20_y)

preds = bst.predict(dval)
y_pred = [int(pred.argmax()) for pred in preds]  # Convert probabilities to class labels

accuracy = accuracy_score(test20_y, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9383296749242187
```