Datamining Preprocessing:
The Human Activity Recognition 70+

Prepared by:

Youssef Mahmoud Anis - 192100029

Youssef Muhammed Gabr -192100069

Libraries Used IN This Project:

- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns
- from sklearn.preprocessing import MinMaxScaler
- from sklearn.model_selection import train_test_split

1. Explore the dataset's features and target variable.

[184]:
```
file_ids = range(501, 519)
patients = [pd.read_csv(f"{pid}.csv") for pid in file_ids]

# Combine all patients into one DataFrame
all_data = pd.concat(patients, ignore_index=True)
```

[212]:
```
print("First 10 rows of the dataset:\n")
display(all_data.head(10))
```

```
First 10 rows of the dataset:
```

	timestamp	back_x	back_y	back_z	thigh_x	thigh_y	thigh_z	label
0	2021-03-24 14:42:03.839	-0.999023	-0.063477	0.140625	-0.980469	-0.112061	-0.048096	6
1	2021-03-24 14:42:03.859	-0.980225	-0.079346	0.140625	-0.961182	-0.121582	-0.051758	6
2	2021-03-24 14:42:03.880	-0.950195	-0.076416	0.140625	-0.949463	-0.080566	-0.067139	6
3	2021-03-24 14:42:03.900	-0.954834	-0.059082	0.140381	-0.957520	-0.046143	-0.050781	6
4	2021-03-24 14:42:03.920	-0.972412	-0.042969	0.142822	-0.977051	-0.023682	-0.026611	6
5	2021-03-24 14:42:03.940	-0.988770	-0.026123	0.157227	-0.984863	-0.042725	-0.032715	6
6	2021-03-24 14:42:03.960	-1.001953	-0.016113	0.162109	-0.992920	-0.075439	-0.024170	6
7	2021-03-24 14:42:03.980	-1.000488	-0.035400	0.191406	-0.996338	-0.072754	-0.013428	6
8	2021-03-24 14:42:04.000	-0.996826	-0.056152	0.187500	-0.974609	-0.060303	-0.015625	6
9	2021-03-24 14:42:04.019	-0.978027	-0.083252	0.187500	-0.966797	-0.062500	-0.015625	6

[188]:
```
print(f"Dataset contains {all_data.shape[0]} rows and {all_data.shape[1]} columns.")
```

```
Dataset contains 2259597 rows and 8 columns.
```

We started by combining 18 different CSV files, each representing data from a different patient. These files were identified by a unique patient ID (PID), ranging from 501 to 518. Using a loop, we read each file using the read_csv function and stored them in a list called patients.

Next, we used pd.concat() to merge all these individual patient datasets into a single DataFrame called all_data, which makes it easier to work with the complete dataset as one unit.

To understand the structure of the dataset, we displayed the first 10 rows. This gave us a preview of the types of data we are working with. After that, we used .shape to check the total number of rows and columns in the dataset. It turns out the dataset contains approximately **2.25 million rows and 8 columns**.

The dataset includes:

- **timestamp**: the exact time of the data recording
- **back_x, back_y, back_z**: accelerometer readings from the lower back (in x, y, and z directions)
- **thigh_x, thigh_y, thigh_z**: accelerometer readings from the right front thigh
- **label**: the activity being performed, which is categorized as:

  1: walking
  3: shuffling
  4: stairs (ascending)
  5: stairs (descending)
  6: standing
  7: sitting
  8: lying

This setup allows us to explore how different body movements are represented in the sensor data and how we might use this for activity classification.

2. Handle missing values (if any) and outliers.

[190]:
```
#mean
bx_mean = all_data ['back_x'].mean()
by_mean = all_data ['back_y'].mean()
bz_mean = all_data ['back_z'].mean()
tx_mean = all_data ['thigh_x'].mean()
ty_mean = all_data ['thigh_y'].mean()
tz_mean = all_data ['thigh_z'].mean()

print("\nBack Mean scores: ")
print(f"back_x: {bx_mean}, back_y: {by_mean}, back_z: {bz_mean}")

print("\nThigh Mean scores: ")
print(f"thigh_x: {tx_mean}, thigh_y: {ty_mean}, thigh_z: {tz_mean}")
```

```
Back Mean scores:
back_x: -0.8699343930886793, back_y: -0.03316849906111574, back_z: 0.023424907512711337

Thigh Mean scores:
thigh_x: -0.6796213034173795, thigh_y: 0.0027747422088983133, thigh_z: -0.384121996420158
```

[129]:
```
#missing values
print("Missing values in each column:\n")
print(all_data.isnull().sum())
```

```
Missing values in each column:

timestamp    0
back_x       0
back_y       0
back_z       0
thigh_x      0
thigh_y      0
thigh_z      0
label        0
dtype: int64
```

We calculated the **mean values** for each of the sensor data columns:

- back_x, back_y, back_z (lower back sensors)
- thigh_x, thigh_y, thigh_z (thigh sensors)

This gives us a general idea of the average sensor readings, which can be useful later for normalization or handling missing data.

After calculating the means, we checked for any **missing values** in the dataset using **isnull().sum()**. The result showed that there are **no missing values** in any of the columns, which means the dataset is clean and we won't be using the mean here, so now we will go find outliers.

[10]:
```
# IQR for Back
Q1_back = all_data[['back_x', 'back_y', 'back_z']].quantile(0.25)
Q3_back = all_data[['back_x', 'back_y', 'back_z']].quantile(0.75)
IQR_back = Q3_back - Q1_back

print("Back IQR values:\n", IQR_back)

mask_back = ~((all_data[['back_x', 'back_y', 'back_z']] < (Q1_back - 1.5 * IQR_back)) |
              (all_data[['back_x', 'back_y', 'back_z']] > (Q3_back + 1.5 * IQR_back))).any(axis=1)

# IQR for Thigh
Q1_thigh = all_data[['thigh_x', 'thigh_y', 'thigh_z']].quantile(0.25)
Q3_thigh = all_data[['thigh_x', 'thigh_y', 'thigh_z']].quantile(0.75)
IQR_thigh = Q3_thigh - Q1_thigh

print("\nThigh IQR values:\n", IQR_thigh)

mask_thigh = ~((all_data[['thigh_x', 'thigh_y', 'thigh_z']] < (Q1_thigh - 1.5 * IQR_thigh)) |
               (all_data[['thigh_x', 'thigh_y', 'thigh_z']] > (Q3_thigh + 1.5 * IQR_thigh))).any(axis=1)

# Combine both masks
final_mask = mask_back & mask_thigh
all_data_clean = all_data[final_mask]

print("\nShape after IQR-based outlier removal:", all_data_clean.shape)
print(f"Total outliers removed: {all_data.shape[0] - all_data_clean.shape[0]}")
```

```
Back IQR values:
 back_x    0.164795
back_y    0.160400
back_z    0.587646
dtype: float64

Thigh IQR values:
 thigh_x    0.916748
thigh_y    0.229004
thigh_z    0.981934
dtype: float64

Shape after IQR-based outlier removal: (1821212, 8)
Total outliers removed: 438385
```

We applied the **Interquartile Range (IQR)** method to remove outliers from the dataset.
First, we calculated the 25th (**Q1**) and 75th (**Q3**) percentiles for the **back sensor** data (back_x, back_y, back_z). The IQR was computed as Q3 - Q1, and values outside the range Q1 - 1.5*IQR to Q3 + 1.5*IQR were marked as outliers. A mask was created to identify these outlier rows.

The same steps were repeated for the **thigh sensor** data (thigh_x, thigh_y, thigh_z).
Another mask was created for outliers in the thigh readings. We then combined both masks to filter out rows that had outliers in either group.

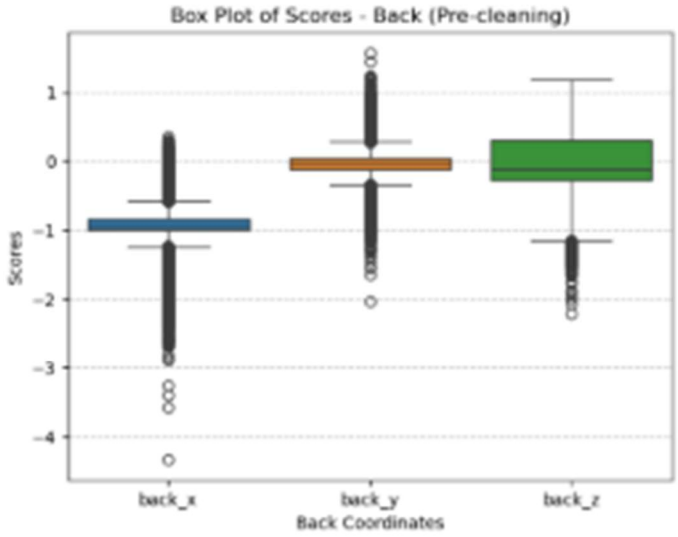
The cleaned dataset was saved in all_data_clean. This process reduced the dataset from **2,259,597** rows to **1,822,112**. A total of **438,885 outlier rows** were removed.

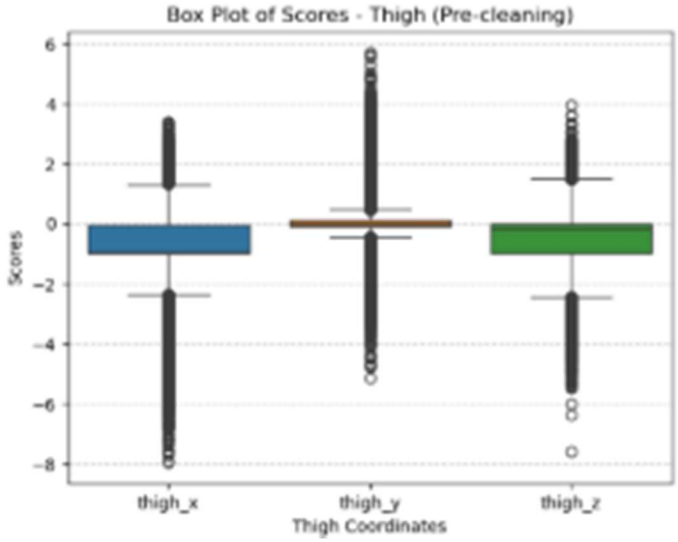
Removing these outliers helps improve the quality of the data. It ensures that extreme values won't distort analysis or model training. To see the difference between before and after of this effect we can use boxplot pre and post outliers removal.

- **Pre-Cleaning**

[9]:
```
# Boxplots (Pre-cleaning)
sns.boxplot(data=all_data[['back_x', 'back_y', 'back_z']])
plt.title("Box Plot of Scores - Back (Pre-cleaning)")
plt.xlabel('Back Coordinates')
plt.ylabel('Scores')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

sns.boxplot(data=all_data[['thigh_x', 'thigh_y', 'thigh_z']])
plt.title("Box Plot of Scores - Thigh (Pre-cleaning)")
plt.xlabel('Thigh Coordinates')
plt.ylabel('Scores')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

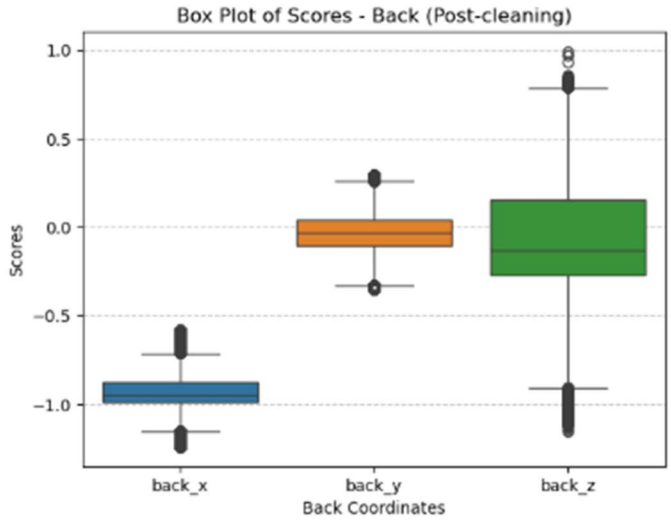


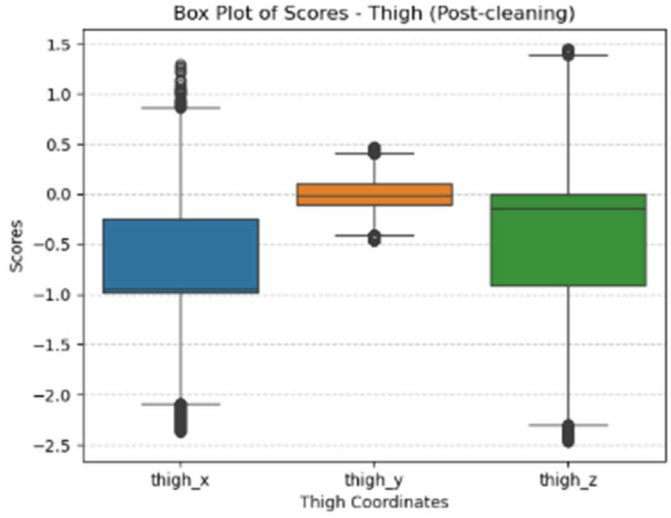


- **Post-Cleaning**

[222]:
```
# Boxplots (Post-cleaning)
sns.boxplot(data=all_data_clean[['back_x', 'back_y', 'back_z']])
plt.title("Box Plot of Scores - Back (Post-cleaning)")
plt.xlabel('Back Coordinates')
plt.ylabel('Scores')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

sns.boxplot(data=all_data_clean[['thigh_x', 'thigh_y', 'thigh_z']])
plt.title("Box Plot of Scores - Thigh (Post-cleaning)")
plt.xlabel('Thigh Coordinates')
plt.ylabel('Scores')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```





3. Perform feature scaling or normalization.

[238]:
```
# Select features and target
features_to_scale = ['back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y', 'thigh_z']
X = all_data_clean[features_to_scale]
y = all_data_clean['label']

print("\nShape before normalization:", X.shape)

# Apply Min-Max Normalization
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

print("Shape after normalization:", X_scaled.shape)
```

```
Shape before normalization: (1821212, 6)
Shape after normalization: (1821212, 6)
```

We use the Min-Max scaling technique. Six features—**'back_x', 'back_y', 'back_z', 'thigh_x', 'thigh_y'**, and **'thigh_z'**—were selected from the cleaned dataset and stored in the variable X, while the target labels were stored separately in y. Before applying normalization, the shape of X was printed to confirm the dimensionality of the dataset.

A **MinMaxScaler** from the **sklearn** library was then instantiated and applied to the feature set using the **fit_transform** method, which scales the features to a range between 0 and 1. The output was saved in X_scaled, and its shape was printed to verify that normalization did not alter the data structure, which remained at (1821212, 6).

This confirms that each of the six features across over 1.8 million samples was successfully normalized. Normalizing the data in this manner is an essential preprocessing step, as it ensures that all features contribute equally to the model.

Dataset Link: DATASET

The code GitHub Link: GitHub