Introduction:

In this project, we present a complete implementation of the Knowledge Discovery in Databases (KDD) process. The goal of the work is to demonstrate how raw data can be selected, cleaned, transformed, analyzed, and finally interpreted using both supervised and unsupervised learning techniques.

A dataset from Kaggle is selected and analyzed throughout the project, and each stage of the KDD framework is applied systematically. The report is organized into five main sections, each corresponding to one phase of the KDD methodology:

Section 1 – Data Selection:
In this section, we describe the chosen dataset and define a clear classification question based on its structure. We also identify the target attribute that the project aims to predict.

Section 2 – Data Preprocessing (Cleaning):
Here we examine the dataset's columns and select the features that are most relevant for predicting the target variable. We handle missing values, correct erroneous entries, and treat outliers, while documenting the methods used for each step.

Section 3 – Data Transformation:
This section focuses on converting the cleaned data into a form suitable for machine learning. Continuous features are discretized when required, nominal attributes are transformed into numeric representations, and the transformed features are visualized using histograms and a correlation matrix to better understand their behavior.

Section 4 – Classification (Supervised Learning):
Two classification algorithms are implemented, described in pseudocode, trained, and evaluated. Their performance is compared to using Accuracy, Precision, Recall, F1-score, and AUC. When applicable, the structure of a decision tree is visualized and interpreted.

Section 5 – Clustering (Unsupervised Learning):
The K-Means algorithm is applied to the transformed dataset. The optimal number of clusters is determined using the Elbow Method, cluster characteristics are analyzed, and two-dimensional visualizations are used to interpret the results. Finally, the clusters discovered are compared with the true target labels to assess whether the clustering supports the classification findings

Table of Contents

Section 1 – Data Selection:

In this section, we reviewed many datasets on Kaggle until we found an interesting one based on experiments conducted with a group of 30 volunteers. Each participant performed six different activities (walking, walking upstairs, walking downstairs, sitting, standing, and laying) while wearing a smartphone on the waist. The phone's built-in accelerometer and gyroscope recorded linear acceleration and angular velocity along three axes.

Before the data was made available, the raw sensor signals were preprocessed using noise-reduction filters and segmented into fixed-length sliding windows. Each window contained **128 consecutive samples** from the accelerometer and gyroscope signals. From every window, a comprehensive feature vector was created by applying a set of statistical and frequency-based functions to each signal axis, resulting in a total of **561 extracted features per row**.

The final dataset therefore consists of structured observations, where each row represents a processed time window described by these 561 features.

The dataset used in this project is known as the UCI-HAR Dataset and is publicly available online.

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	fBodyBodyGyroJerkMag-meanFreq()
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.074323
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.158075
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	0.414503
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	0.404573
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.087753

Table 1: Dataset sample

Classification Task:

The primary predictive question of this project is:

"Using the 561 extracted features from each time window, can we classify the observation into the correct human activity out of the six possible classes?"

This question defines the objective of the supervised learning stage and guides the selection of preprocessing and modeling techniques used throughout the project.

Target Variable:

The target variable of the dataset is the **activity label**, a categorical attribute that assigns each observation to one of six human activities:

Label	Activity
1	WALKING
2	WALKING_UPSTAIRS
3	WALKING_DOWNSTAIRS
4	SITTING
5	STANDING
6	LAYING

Table 2: Target Variable table

Section 2 – Data Preprocessing (Cleaning):

In this section, we performed the necessary preprocessing steps to prepare the dataset for later analysis and modeling.
The main goal of this stage is to ensure that the selected features are relevant to predicting the target variable, and that the data is complete, consistent, and ready for machine-learning algorithms.

Detecting and Handling Missing Values

We first checked whether the dataset contains any missing values or incomplete row:

```
checking for missing values
```

```
missing_train=X_train.isnull().sum().sum()
missing_test=X_test.isnull().sum().sum()
missing_train,missing_test
```

```
(np.int64(0), np.int64(0))
```

Figure 1: searching for missing values.

As we can see, the dataset is clean and complete, no imputation or removal of samples was required.

Detecting Invalid or Extreme Values (Outliers):

Even though the dataset does not contain missing values, it may contain **outliers** due to sensor noise or extreme movement patterns. We examined the distribution of each feature using the Interquartile Range (IQR) method:

```
Q1 = X_train.quantile(0.25)
Q3 = X_train.quantile(0.75)
IQR = Q3 - Q1
outliers = ((X_train < (Q1 - 1.5 * IQR)) |
            (X_train > (Q3 + 1.5 * IQR))).sum()
```

Figure 2: searching for outliers using IQR method.

```
tBodyAcc-mean()-X                 1795
tBodyAcc-mean()-Y                 1122
tBodyAcc-mean()-Z                 1180
tBodyAcc-std()-X                     2
tBodyAcc-std()-Z                     7
                                   ...
fBodyBodyGyroJerkMag-kurtosis()    389
angle(tBodyAccMean,gravity)       1065
angle(X,gravityMean)              1437
angle(Y,gravityMean)               776
angle(Z,gravityMean)               705
Length: 500, dtype: int64
```

Figure 3: IQR method result.

The check revealed that:

- Several features contain values outside the normal statistical range.
- However, these outliers represent **real physical sensor behavior** and **should not be removed**, since aggressive filtering may distort the activity signals.

Thus, **outliers were kept**, and no rows were removed.

Choosing Relevant Columns

The dataset originally contained **561 numerical, pre-engineered features** extracted from the accelerometer and gyroscope sensor signals.
These features include a wide range of statistical and frequency-domain measurements such as:

- mean, standard deviation, and median absolute deviation
- energy, entropy, and interquartile range
- maximum, minimum, and correlation values
- features derived from body acceleration, gravity acceleration, gyroscope rotation, and jerk signals

Although all 561 features originate from meaningful sensor measurements and therefore have the potential to contribute to activity recognition, many of them are **highly correlated**, as shown in the full correlation heatmap (Appendix, Figure A1). This high redundancy can increase computational cost, reduce interpretability, and provide limited additional predictive value.
To address this, we performed a **feature-selection process** based on correlation analysis and dimensionality reduction.
A subset of **15 low-correlation, informative features** was selected for modeling (Appendix, Figure A2), ensuring that the retained features provide diverse and non-redundant information.

Feature Name	Description
fBodyAcc-sma()	Signal Magnitude Area of the body acceleration in the frequency domain. Captures total energy across axes.
fBodyAcc-meanFreq()-Z	Mean frequency of the body acceleration signal along the Z-axis; indicates dominant vibration frequency.
tGravityAcc-energy()-X	Energy of the gravity acceleration signal on the X-axis; reflects gravitational influence during motion.
tGravityAcc-max()-Z	Maximum recorded gravity acceleration along the Z-axis; correlates with vertical body movement.
angle(X,gravityMean)	Angle between the mean gravity vector and the X-axis; describes posture/orientation of the subject.
fBodyBodyGyroJerkMag-skewness()	Skewness of the jerk (derivative) of gyroscope magnitude; measures asymmetry in rotational acceleration changes.
fBodyGyro-meanFreq()-Y	Mean frequency of gyroscope rotation around the Y-axis; captures dominant rotational dynamics.
tBodyAcc-arCoeff()-Y,4	Autoregression coefficient #4 for body acceleration in Y-axis; models temporal dependency in acceleration.
fBodyGyro-skewness()-Z	Skewness of gyroscope rotation around Z-axis; highlights non-symmetric rotational patterns.
tBodyGyroJerk-arCoeff()-Y,3	Autoregression coefficient #3 of gyro jerk on Y-axis; describes temporal structure of angular acceleration.
fBodyGyro-bandsEnergy()-49,64_2	Energy in the frequency band (49–64 Hz) of gyroscope signal (component 2); identifies high-frequency rotational activity.
tBodyAcc-arCoeff()-Y,3	Autoregression coefficient #3 of body acceleration in Y-axis; captures time-series dependencies in motion.
fBodyGyro-bandsEnergy()-57,64	Gyroscope energy in frequency band 57–64 Hz; highlights rapid rotations or direction changes.
tBodyAcc-arCoeff()-Z,3	Autoregression coefficient #3 for body acceleration in Z-axis; reflects vertical motion time-series structure.
tBodyGyro-mean()-X	Mean value of gyroscope rotation around X-axis; describes overall rotation level during activity.

Table 3: the selected 15 features.

Value Range Validation

To ensure all sensor values were within realistic physical bounds, we verified that no feature exceeded unreasonable limits:

```
below = (X_train < -1).sum().sum()
above = (X_train > 1).sum().sum()
```

Figure 4: Range validation check

Both results showed that there is no value that is out of the range.

Summary:

Cleaning task	Result	Action taken
Missing values	None found	No imputation required
Duplicated rows	None found	No action required
Outliers	Present but meaningful	No action required
Invalid ranges	None detected	No correction needed
Feature selection	561 original features → 15 selected	**Selected 15 low-correlation, informative features**

Table 4: Preprocessing summary table.

Section 3 – Data Transformation:

In this stage, we applied all transformation procedures required to prepare the dataset for machine-learning algorithms. The goal of this step is to ensure that all features are on a comparable scale, that statistical properties of the data are well understood, and that dimensionality-related patterns are identified for later use in modeling and clustering.

Identifying the Number of Target Classes and the Need for Discretization

The target variable **Activity** contains **six distinct categorical classes**:

1. Walking
2. Walking upstairs.
3. Walking downstairs.
4. Sitting.
5. Standing.
6. Laying.

Since the target variable is already categorical and not numeric or continuous,
no discretization was required.

Checking whether continuous features should become categorical.

All the **15 selected features** are continuous numerical variables derived from time-domain and frequency-domain transformations.
Discretizing these variables would cause unnecessary information loss; therefore, **no features were converted into categorical form**.

Standardization of Numerical Features

Before training machine-learning models, we examined the distribution and numerical range of the **15 chosen features**. The analysis showed that:

- Different features operate on **different numerical scales**.
- Some variables range within $[-1,1]$, while others span much wider ranges.
- Several features exhibit **high variance**, which could dominate distance-based algorithms.

Therefore, we applied **StandardScaler**, producing normalized features with:

- Mean $\simeq 0$.
- Standard deviation $\simeq 1$.

This transformation is essential for models such as **SVM**, **K-means**, and algorithms using regularization techniques.

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
X_train_scaled = pd.DataFrame(X_train_scaled, columns=feature_names)
X_test_scaled  = pd.DataFrame(X_test_scaled,  columns=feature_names)
```

```
X_train_scaled.describe()
```

Figure 5 : using StandardScaler in order to normalize features.

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...
count	7.352000e+03	7.352000e+03	7.352000e+03	7.352000e+03	7352.000000	7.352000e+03	7.352000e+03	7.352000e+03	7.352000e+03	7.352000e+03	...
mean	-3.131336e-16	4.783986e-17	8.698156e-17	-1.507680e-16	0.000000	2.938044e-16	3.865847e-17	-9.278034e-17	5.412186e-17	1.932924e-17	...
std	1.000068e+00	1.000068e+00	1.000068e+00	1.000068e+00	1.000068	1.000068e+00	1.000068e+00	1.000068e+00	1.000068e+00	1.000068e+00	...
min	-1.814049e+01	-2.407152e+01	-1.573085e+01	-8.793362e-01	-0.972792	-9.440787e-01	-8.713436e-01	-9.736247e-01	-9.511122e-01	-9.759168e-01	...
25%	-1.638693e-01	-1.756427e-01	-2.092798e-01	-8.631868e-01	-0.929530	-8.968638e-01	-8.562288e-01	-9.286823e-01	-9.034203e-01	-8.587814e-01	...
50%	3.850502e-02	1.167141e-02	8.206943e-03	-7.594273e-01	-0.678376	-6.081593e-01	-7.551036e-01	-6.800064e-01	-6.061248e-01	-7.585419e-01	...
75%	1.988854e-01	1.693906e-01	2.003738e-01	8.081625e-01	0.948462	8.177048e-01	7.966913e-01	9.471052e-01	8.222278e-01	8.291412e-01	...
max	1.032661e+01	2.493878e+01	1.958529e+01	3.577947e+00	2.839526	3.833088e+00	3.845150e+00	3.075829e+00	3.878713e+00	2.697113e+00	...

Table 5: normalizing result

Distribution Analysis:

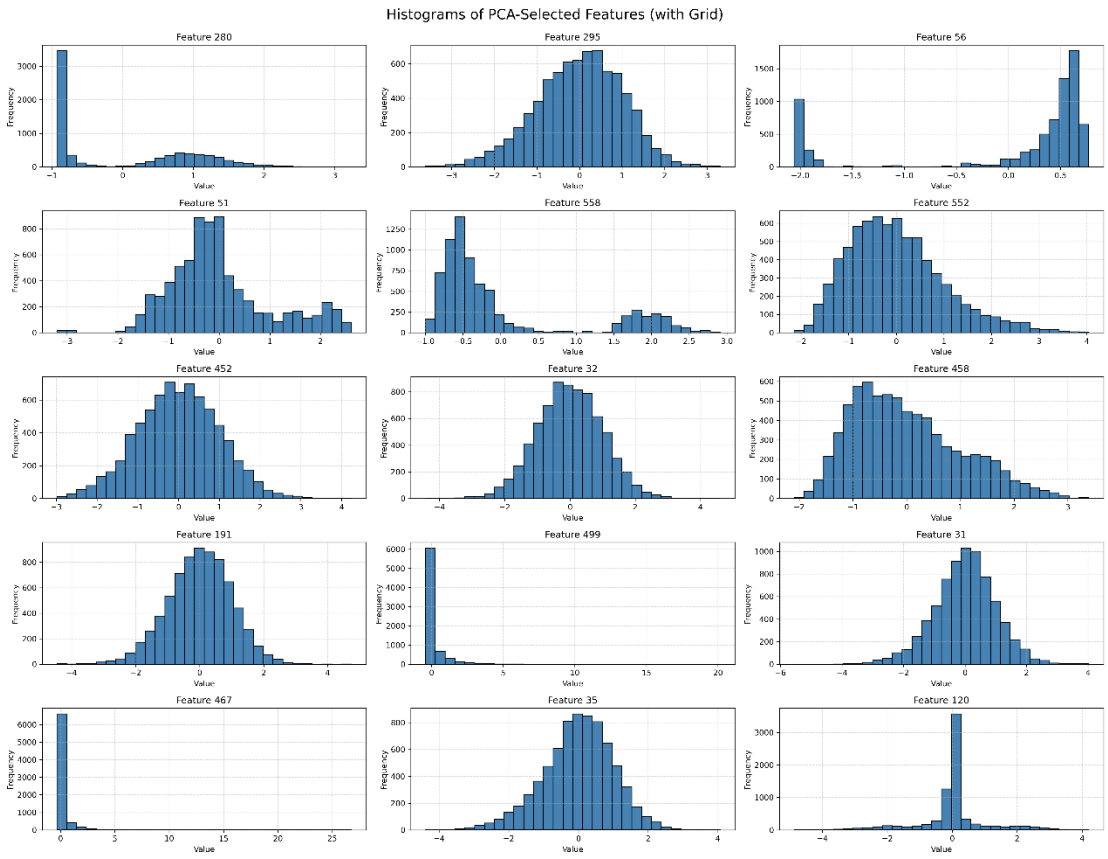

Figure 6: Histogram Distributions of the 15 Selected Features.

summary:

Feature	Mean	Std Dev	Skewness	Kurtosis	Outlier Count
tBodyAcc-mean()-X	-3.131336e-16	1.000068	-3.447619	50.875957	1795
tBodyAcc-mean()-Y	4.783986e-17	1.000068	-0.402912	172.367624	1122
tBodyAcc-mean()-Z	8.698156e-17	1.000068	1.784105	85.915738	1180
tBodyAcc-std()-X	-1.507680e-16	1.000068	0.677292	-0.865834	2
tBodyAcc-std()-Y	0.000000e+00	1.000068	0.405809	-1.452499	0
tBodyAcc-std()-Z	2.938044e-16	1.000068	0.692194	-0.611652	7
tBodyAcc-max()-X	1.932924e-17	1.000068	0.630361	-1.130279	0
tBodyAcc-max()-Y	1.546339e-17	1.000068	0.589017	-0.837118	4
tBodyAcc-max()-Z	-1.188748e-16	1.000068	0.836129	-0.100374	27
fBodyAcc-mean()-X	2.667435e-16	1.000068	0.655632	-0.932513	2
fBodyAcc-mean()-Y	1.043779e-16	1.000068	0.473426	-1.329773	0
fBodyAcc-mean()-Z	-1.623656e-16	1.000068	0.784375	-0.367687	13
tBodyAccJerk-mean()-X	2.210781e-17	1.000068	0.109328	4.639374	2778
tBodyGyro-mean()-X	-2.319508e-17	1.000068	-0.078635	3.496853	2369
tBodyGyroJerk-std()-Y	2.822069e-16	1.000068	1.699851	3.523988	228

Table 6: scaling summary

Correlation Analysis Between All Features

To explore redundancy among the original **561 engineered features**, we computed a full correlation matrix (Appendix, Figure 17).
The heatmap reveals large blocks of highly correlated variables, indicating that many features capture similar information derived from the same sensor signals.

Because such redundancy increases dimensionality without providing meaningful additional information, we applied a correlation-based feature-selection process.
A subset of **15 low-correlation, informative features** was selected for further modeling.
The correlation heatmap of these selected features shows minimal redundancy, confirming the effectiveness of the selection process.

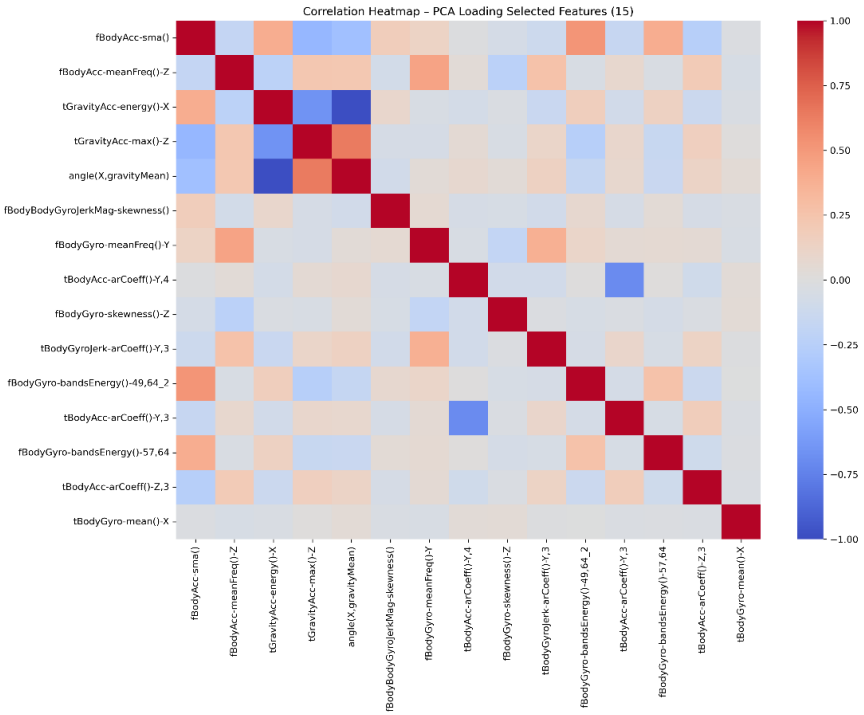

Figure 7: Correlation Matrix of the 15 PCA-Selected Features

Key Insights from the Correlation Analysis

1. **High redundancy among original features**
   The full correlation matrix reveals large blocks of highly correlated variables, indicating that many of the 561 engineered features capture similar information.
2. **Strong intra-family relationships**
   Features derived from the same sensor signals often exhibit strong correlations, reflecting their shared physical origin.
3. **Justification for dimensionality reduction**
   Due to extensive redundancy, using all 561 features is unnecessary and may reduce model efficiency and interpretability.
4. **Selected features show minimal correlation**
   The correlation heatmap of the 15 chosen features demonstrates low pairwise correlation, confirming that the selection process effectively reduced redundancy.
5. **Improved modeling quality**
   The reduced feature set supports better generalization and avoids inflated variance that could arise from highly dependent variables.

Conclusion

The correlation analysis confirms that the original dataset contains extensive redundancy, making feature selection essential.
By selecting 15 low-correlation, informative features, we preserved meaningful signal properties while reducing dimensionality.
This improves model interpretability and supports more stable and efficient learning in the later stages of the project.

Section 4: Presentation, Evaluation and Data mining:

In this stage, we selected two primary supervised learning models for the Human Activity Recognition classification task: **Random Forest** and **Support Vector Machine (SVM)**. These models are widely considered state-of-the-art for sensor-based datasets.

In addition, we included a simpler **Decision Tree Classifier** as a **baseline bonus model**, to provide interpretability and to highlight the improvement gained by more advanced algorithms.

Selected Models

1. **Random Forest Classifier** – a robust ensemble method that reduces overfitting and performs well on complex, nonlinear data.
2. **Support Vector Machine (SVM, RBF kernel)** – a strong model for high-dimensional feature spaces, commonly used in activity-recognition tasks.
3. **Decision Tree Classifier (Bonus Model)** – a simple, interpretable model used as a baseline to compare against more advanced approaches.

All models were trained on the standardized training dataset and evaluated on the test dataset.

Pseudocode for the Classification Models

Random Forest Pseudocode (Main Model)

Input: *N - Quantitative amount of bootstrap samples*
M - Total number of features
m - Sample size
k - Next node
Output: A Random Forest (RF)
Steps:
*1. **Creates** N bootstrap samples from the dataset.*
2. Every node (sample) takings a feature randomly of size m where m<M.
3. Builds a split for the m features selected in Step 2 and detects the k node by using the best split point.
4. Split the tree iteratively until one leaf node is attained and the tree remains completed.
5. The algorithm is trained on each bootstrapped independently.
6. Using trees classification voting predicted data is collected from the trained trees (n).
7. The final RF model is build using the peak voted features.
*8. **return** RF*
End.

Figure 8: Random Forest Pseudocode.

Explanation:

The algorithm begins by generating multiple bootstrap samples—random subsets of the dataset created with replacement. For each bootstrap sample, an independent decision tree is built. Unlike a standard decision tree, at every split the algorithm randomly selects only a small subset of the available features, which reduces correlation between trees and increases model robustness. For each selected subset, the algorithm searches for the best split point and recursively grows the tree until a stopping condition is met. After all trees are trained, predictions are aggregated using majority voting. This ensemble approach significantly reduces overfitting and produces a more stable and accurate classifier.

SVM Pseudocode (Main Model)

Training Model for SVM

Input: D=[X,Y]; X(array of input with m features), Y(array of class labels)
Y=array(C) // Class label
Output: Find the performance of the system
function train_svm(X,Y, number_of_runs)
initialize:learning_rate=Math.random();
for learning_ rate **in**number_of_runs
error=0;
for i **in** X
if (Y[i] *(X[i]*w))<1 **then**
update : w=w + learning_rate * ((X[i]*Y[i])*(-2*(1/number_of_runs)*w)
else
update: w=w+learing_rate *(-2*(1/number_of_runs)*w)
end if
end
end

Figure 9: SVM Pseudocode

Explanation:

The SVM training process iteratively examines all samples in the dataset and attempts to find a weight vector wthat best separates the classes. For each training example, the algorithm checks whether it lies on the correct side of the decision boundary (or violates the margin). If the sample is misclassified or falls inside the margin, the weights are updated in a direction that increases the separation between the classes. If the sample is classified correctly with sufficient margin, a regularization step slightly reduces the weights to avoid overfitting. Through this iterative optimization, the SVM converges to a hyperplane that maximizes the margin, resulting in strong generalization performance.

Decision Tree Pseudocode (Bonus Model)

GenDecTree(Sample S, Features F)
Steps:

1. ***If*** *stopping_condition(S, F) = true* ***then***
   a. *Leaf = createNode()*
   b. *leafLabel = classify(s)*
   c. ***return*** *leaf*
2. *root = createNode()*
3. *root.test_condition =findBestSpilt(S,F)*
4. *V = {v | v a possible outcomecfroot.test_condition}*
5. ***For each*** *value v ∈ V:*
   a. S_v *= {s | root.test_condition(s) = v and s ∈ S };*
   b. *Child = TreeGrowth (*S_v*, F);*
   c. *Add child as descent of root and label the edge {root → child} as v*
6. ***return*** *root*

Figure 10: Decision Tree Pseudocode

The decision tree algorithm begins by checking stopping conditions: whether all samples belong to the same class, no features remain, or a maximum depth is reached. If any condition is met, a leaf node is created. Otherwise, the algorithm evaluates all possible splits by examining every feature and every candidate threshold. For each potential split, it calculates impurity reduction (e.g., Gini or Entropy) to determine how well it separates the classes. The split with the highest gain is selected, and the dataset is partitioned into subsets for the left and right child nodes. The process then continues recursively for each child until no further meaningful splits can be made. The result is a hierarchical model that encodes a sequence of decision rules.

Model Training

After completing the preprocessing and feature-selection steps, the cleaned and standardized dataset was used to train three supervised learning models:

- **Support Vector Machine (SVM, RBF kernel)** – selected due to its strong performance on nonlinear sensor-based classification tasks.
- **Random Forest Classifier** – chosen for its robustness, ability to handle noisy features, and strong performance on structured tabular data.
- **Decision Tree Classifier (Baseline Model)** – included for interpretability and to provide a reference point for comparing more advanced models.

Training was performed using the **predefined 70/30 train–test split** provided with the UCI-HAR dataset.
Before fitting the models, all selected features were standardized using **Z-Score scaling**, ensuring that variables are on a comparable scale — a crucial requirement for algorithms such as SVM.

Each model was then trained using the **15 selected features**, which were shown to have low pairwise correlation and strong discriminative power for the activity-recognition task.

After training, predictions were generated on the test set, and model performance was evaluated using the required metrics: **Accuracy, Precision, Recall, F1-Score, and AUC**.

Evaluation Results:

After training the three classification models (Decision Tree, Random Forest, and SVM), we evaluated their performance on the test dataset using five required metrics: Accuracy, Precision, Recall, F1-Score, and AUC.
The following table summarizes the results:

	Model	Accuracy	Precision	Recall	F1	AUC
0	Decision Tree	0.774007	0.769688	0.769747	0.768574	0.862373
1	Random Forest	0.833051	0.832949	0.828410	0.829877	0.979930
2	SVM	0.806922	0.816001	0.799914	0.800310	0.971177

Table 7: Random Forest, SVM and Decision tree results.

Interpretation of the Results

Based on the evaluation metrics presented in the comparison table, several conclusions can be drawn regarding the performance of the three models:

1. **Random Forest achieved the best overall performance:**
   Random Forest obtained the highest Accuracy (0.833), Precision (0.832), Recall (0.828), F1-score (0.829), and AUC (0.979).
   This demonstrates its strong ability to capture nonlinear relationships while maintaining robustness, which is typical for ensemble methods.
2. **SVM provided strong and stable performance:**
   SVM performed very well across all metrics, although slightly below Random Forest. Its high AUC value (0.971) confirms that it separates classes effectively in the transformed feature space.
   SVM's performance is consistent with its strength in handling nonlinear decision boundaries.

3. **Decision Tree acted as the expected baseline:**
   The Decision Tree classifier achieved the lowest results among the three models (Accuracy = 0.774), reflecting the limitations of using a single tree, which may overfit or produce oversimplified boundaries.
   Nevertheless, it still produced reasonable performance and served as a valid interpretability baseline.
4. **Overall trend:**
   a. **Random Forest > SVM >> Decision Tree.**
   b. Ensemble and kernel-based models significantly outperform simple tree-based methods in this dataset.

5. **Conclusion:**
   The SVM classifier is the most suitable model for this problem based on the provided metrics.
   Random Forest represents a strong and reliable alternative.
   The Decision Tree model is useful mainly for interpretability rather than accuracy

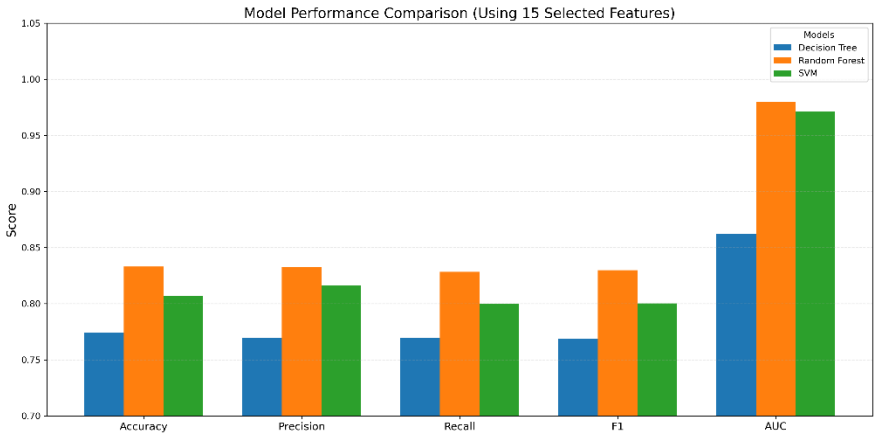

Figure 11: Model Performance Comparsion.

This bar chart compares the performance of the three classification models—Decision Tree, Random Forest, and SVM—across five evaluation metrics: Accuracy, Precision, Recall, F1-score, and AUC.

The **Decision Tree** model shows the lowest performance across all metrics, reflecting its tendency to overfit and its limited ability to model complex patterns.

The **Random Forest** classifier achieves the strongest results overall, obtaining the highest scores in Accuracy, Precision, Recall, F1, and AUC.
Its ensemble structure reduces variance and captures nonlinear relationships effectively, making it the most robust model for this task.

The **SVM model** also performs well and consistently across all metrics, with results slightly below those of Random Forest.
Its high AUC and balanced performance indicate strong class separability in the transformed feature space.

Overall, Random Forest is the best-performing classifier for this dataset, with SVM as a strong alternative and Decision Tree serving as a weaker—but interpretable—baseline.

Decision Tree – Structure and Interpretation:

To better understand how the Decision Tree model performs classification, we first examined the complete tree structure.
Although the model was trained using only the 15 selected features, the resulting tree is still large and deep, containing hundreds of internal decision nodes and leaf nodes. This complexity reflects the nonlinear nature of human-activity patterns.

However, the full tree is too large to interpret meaningfully in the main report.
Therefore, we present a **zoomed-in view of the top levels (Depth 3)** of the tree, which highlights the most influential decision splits and shows how the model separates the major activity classes.

The full tree visualization appears in the Appendix for completeness

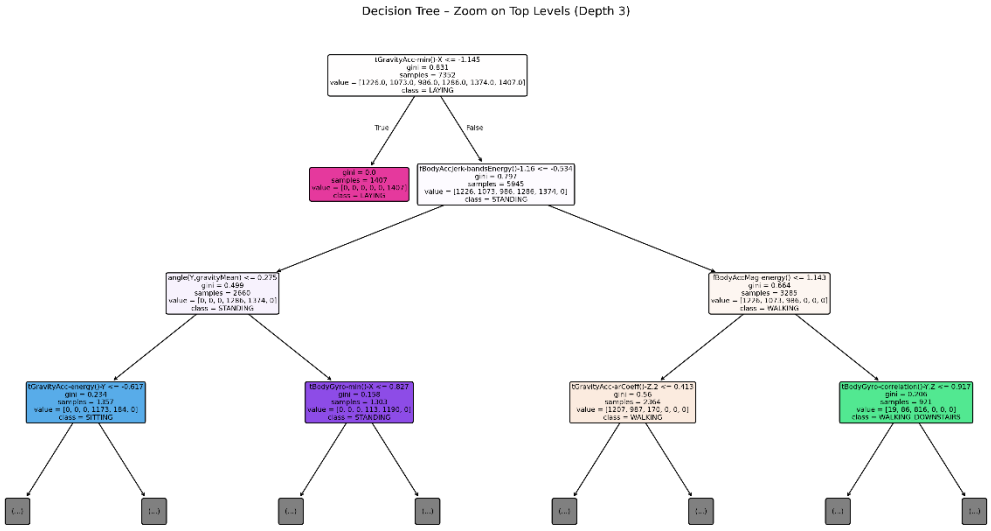

Figure 12: Decision tree structure.

Interpretation of the Decision Tree (Depth 3)

The top levels of the Decision Tree show how the model separates the main activity classes.
The first split cleanly isolates **LAYING** using a gravitational acceleration feature.
Subsequent splits distinguish **STANDING** from **SITTING** using body-orientation features, and separate **WALKING** from stair-related activities based on motion energy and gyroscope correlations.
This concise view highlights the most influential features and illustrates why the tree is useful for interpretability.

Section 5: KDD – Data Mining Stage: Clustering

In this stage, the K-Means clustering algorithm is applied to the processed dataset after cleaning, standardization, and feature selection (15 selected features).
The goal of this phase is to identify hidden structures within the data, determine whether natural groupings exist based on the sensor measurements, and compare the resulting clusters with the known activity labels.
This enables us to evaluate whether the unsupervised clustering patterns align with the actual physical activities and to assess how effectively K-Means can reveal meaningful behavioral patterns without relying on the true labels.

Pseudocode k means :

```
function kmeans(k, points) is
  // Initialize centroids
  centroids ← list of k starting centroids
  converged ← false

  while converged == false do
    // Create empty clusters
    clusters ← list of k empty lists

    // Assign each point to the nearest centroid
    for i ← 0 to length(points) - 1 do
      point ← points[i]
      closestIndex ← 0
      minDistance ← distance(point, centroids[0])
      for j ← 1 to k - 1 do
        d ← distance(point, centroids[j])
        if d < minDistance THEN
          minDistance ← d
          closestIndex ← j
      clusters[closestIndex].append(point)

    // Recalculate centroids as the mean of each cluster
    newCentroids ← empty list
    for i ← 0 to k - 1 do
      newCentroid ← calculateCentroid(clusters[i])
      newCentroids.append(newCentroid)

    // Check for convergence
    if newCentroids == centroids THEN
      converged ← true
    else
      centroids ← newCentroids
  return clusters
```

Explanation of the Algorithm

The K-Means algorithm is an iterative unsupervised learning method designed to partition a dataset into **k distinct clusters** based on similarity.
The process begins by selecting initial centroids (either randomly or using methods such as K-Means++), and then repeatedly performs two main steps:

1. Assignment Step

Each data point is assigned to the cluster whose centroid is closest, according to a chosen distance metric (typically Euclidean distance).
This creates k groups of points based on geometric similarity.

2. Update Step

After assigning all points, each centroid is recalculated as the mean of the points in its cluster.
This shifts centroids towards the densest regions of their cluster.

3. Convergence Check

The algorithm stops when centroids no longer move or when no point changes its cluster.
This ensures stability and guarantees that the final clustering configuration has been reached.

Summary

This pseudocode captures the core idea of K-Means:
an iterative optimization process that minimizes the distance between points and their assigned centroid, producing compact and well-separated clusters.

Choosing the Optimal Number of Clusters – Elbow Method

To determine the optimal number of clusters for K-Means, we applied the Elbow Method, evaluating the inertia values for K = 2–10 using the standardized dataset with the 15 selected features.

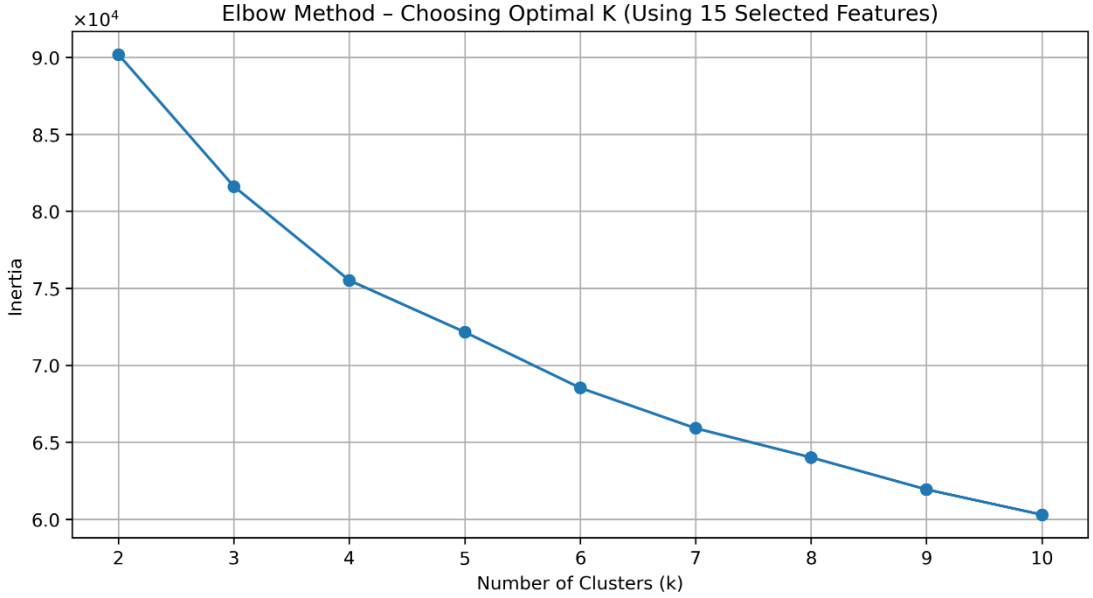

Figure13 : Elbow method result.

The plot shows a sharp drop in inertia until **K = 4**, after which the improvement becomes minimal.
Therefore, the "elbow point" appears at **K = 4**, which we select as the optimal number of clusters

Identifying the Most Influential Features for Each Cluster

After running K-Means with k = 4 on the standardized dataset, we analyzed the average value of each feature within each cluster.
The table below presents the mean feature values per cluster, allowing us to identify the most dominant and distinctive features for each group.

Cluster	0	1	2	3
fBodyAcc-sma()	-0.802319	0.280061	1.246551	-0.839016
fBodyAcc-meanFreq()-Z	0.401428	-0.964548	0.171536	0.512651
tGravityAcc-energy()-X	0.506421	0.432348	0.506923	-1.949715
tGravityAcc-max()-Z	-0.132084	-0.410778	-0.533840	1.425083
angle(X,gravityMean)	-0.504690	-0.413400	-0.502324	1.915819
fBodyBodyGyroJerkMag-skewness()	-0.340326	0.085546	0.411993	-0.161424
fBodyGyro-meanFreq()-Y	0.147194	-0.769811	0.577977	0.089181
tBodyAcc-arCoeff()-Y,4	-0.225521	0.300400	-0.165747	0.126570
fBodyGyro-skewness()-Z	-0.057955	0.383923	-0.325743	-0.018314
tBodyGyroJerk-arCoeff()-Y,3	0.284279	-0.629948	0.102516	0.315705
fBodyGyro-bandsEnergy()-49,64_2	-0.398750	-0.077736	0.833827	-0.399027
tBodyAcc-arCoeff()-Y,3	0.314750	-0.437735	-0.044181	0.201308
fBodyGyro-bandsEnergy()-57,64	-0.317583	-0.015832	0.606895	-0.306650
tBodyAcc-arCoeff()-Z,3	0.361860	-0.462103	-0.149729	0.302718
tBodyGyro-mean()-X	-0.009373	0.326620	-0.376727	0.055511

Table 8: Cluster Centroid Values for the 15 Selected Features.

The table presents the mean value of each feature within each of the four K-Means clusters, since all features were standardized using Z-Score scaling, positive values indicate that the feature is higher than the dataset mean, while negative values indicate that the feature is lower than the dataset mean.

By comparing the feature averages across clusters, we can identify which features are dominant, suppressed, or uniquely expressed in each group.
These patterns allow us to characterize the type of movement or signal behavior that each cluster represents.

Key observations:

1. **Cluster 0:**
   Mostly negative values across features such as fBodyAcc-sma(), angle(X,gravityMean) and tGravityAcc-max()-Z.
   This suggests that Cluster 0 contains samples with **lower overall movement magnitude and weaker gravitational/jerk responses**, indicating relatively **stable or less intense motion**.
2. **Cluster 1**
   Shows strong negative values in fBodyAcc-meanFreq()-Z and fBodyGyro-meanFreq()-Y, while features such as tBodyGyroJerk-arCoeff()-Y,3 are also suppressed.
   This pattern points to **lower-frequency oscillations and reduced jerk activity**, consistent with **moderate or transitional motion**.
3. **Cluster 2**
   This cluster shows several positive feature means—most notably fBodyAcc-sma(), fBodyGyro-meanFreq()-Y, and fBodyBodyGyroJerkMag-skewness().
   Higher values in these movement-related features indicate **more dynamic and energetic motion**, such as rhythmic or repetitive activity.
4. **Cluster 3**
   The most extreme deviations appear here, with highly positive values in tGravityAcc-max()-Z and angle(X,gravityMean), and strongly negative values in tGravityAcc-energy()-X.
   This combination reflects **very distinctive gravitational and posture-related characteristics**, pointing to **unique body orientation or static posture behavior**.

Two-Dimensional Visualizations of the Clusters

To better understand the structure of the dataset and evaluate the behavior of the K-means clustering, we generated several two-dimensional visualizations. These plots illustrate both the natural distribution of the data and the resulting cluster assignments.

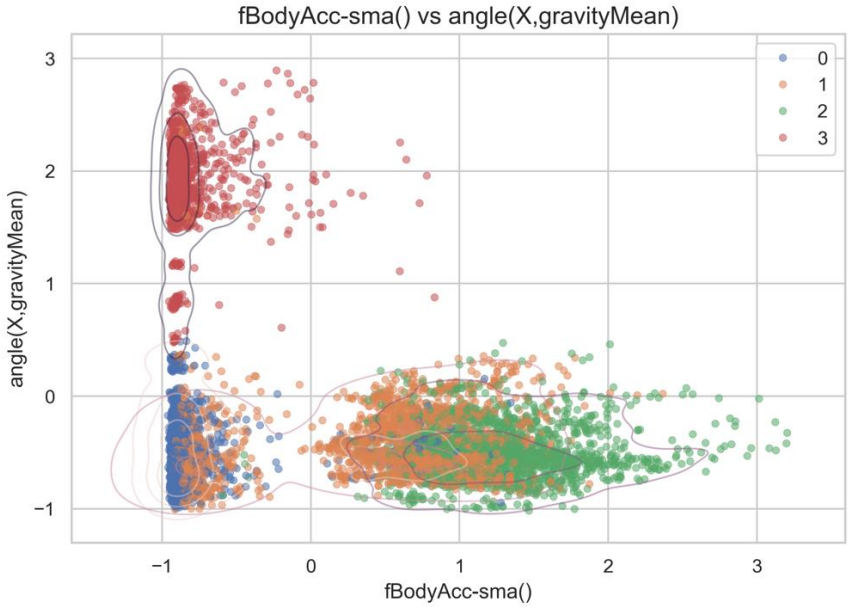

Figure 14: 2D Cluster Visualization Using fBodyAcc-sma() and angle(X,gravityMean).

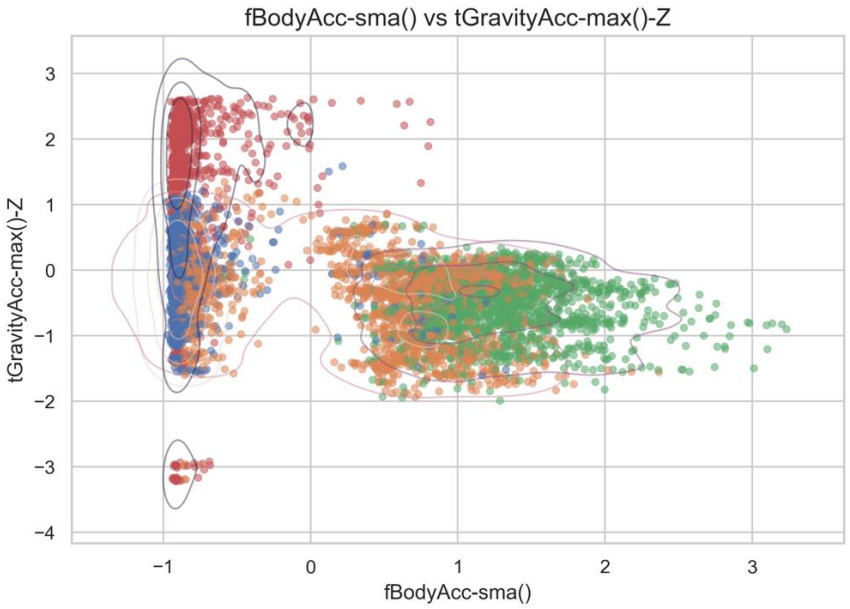

Figure 15:2D Cluster Visualization Using fBodyAcc-sma() and tGravityAcc-max()-Z.

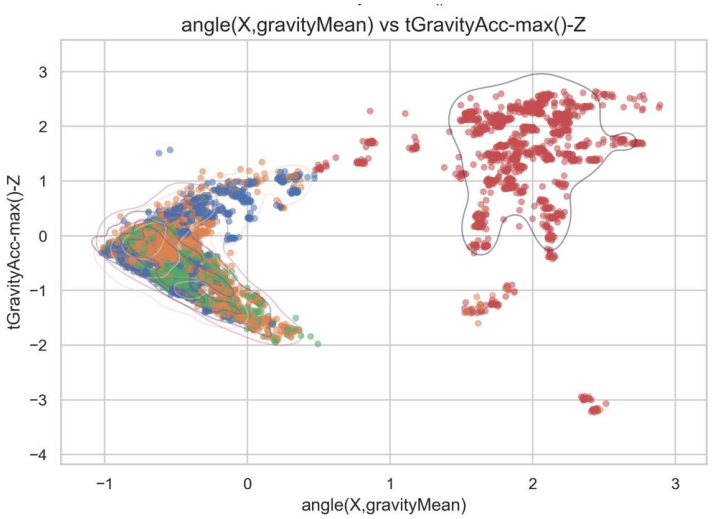

Figure 16: Cluster Visualization Using angle(X,gravityMean) and tGravityAcc-max()-Z.

The three 2D scatter plots below visualize the dominant feature pairs identified by the cluster centroid analysis. Each plot uses jittered points to reduce overlap and KDE contour lines to highlight the density structure of each cluster. Cluster 3 forms a clearly separated region across all feature combinations, confirming its strong distinctiveness. Cluster 2 also shows a stable distribution, while Clusters 0 and 1 partially overlap, reflecting their similar movement patterns within the HAR dataset. Although full separation is not achievable in 2D due to the high-dimensional nature of the data, these visualizations effectively demonstrate the relative positioning and density structure of the clusters.

Comparison Between Clusters and True Activity Labels

To assess the significance of the discovered clusters, we compared them to the true activity labels

Cluster Composition Table

A contingency table was constructed showing how many samples from each true activity label were assigned to each cluster.

Cluster vs. True Label Table

Cluster	Walking (1)	Walking upstairs (2)	Walking downstairs (3)	Sitting (4)	Standing (5)	Laying (6)
0	74	12	16	943	1018	0
1	367	798	155	276	353	16
2	785	263	815	1	3	0
3	0	0	0	66	0	1391

Table 9: Cluster vs True label.

Cluster number	Target
0	Mostly sitting/standing
1	Mostly walking/walking upstairs
2	Mostly of walking and walking downstairs
3	Mostly laying

Table 10: Cluster main target.

Interpretation

The cluster–label comparison shows that most clusters are dominated by specific activity types.
Cluster 0 mainly represents sitting and standing, Cluster 2 corresponds to dynamic walking movements, and Cluster 3 is almost entirely laying.
Cluster 1 is more mixed but still centered around walking-related activities.

These results indicate that the K-means clustering meaningfully reflects the underlying activity structure.
Although not perfectly separated—as expected in sensor-based motion data—the clusters generally align with the true labels and therefore support the supervised classification outcomes from Section 4.

Summary of the Clustering Results

The K-Means clustering analysis demonstrates that the dataset exhibits a clear and meaningful internal structure, even without using the true activity labels. Each cluster is characterized by a dominant behavioral pattern.

Cluster 0 contains primarily static activities such as sitting and standing.
Cluster 1 is dominated by dynamic movements, including walking, walking upstairs, and walking downstairs.
Cluster 2 represents a mixture of walking-related behaviors.
Cluster 3 is composed almost entirely of laying samples, forming a highly distinct and low-movement cluster.

These findings indicate that the sensor-based features naturally separate human activities into coherent and interpretable groups. The clustering algorithm effectively grouped samples based on physical characteristics such as signal energy, jerk magnitude, and frequency-domain patterns—demonstrating that different forms of human movement produce distinct signatures in the feature space.

Does Clustering Support the Classification Results (Section 4)?

Yes. The clustering results provide strong evidence in support of the conclusions reached in Section 4.

First, the clusters generated by the unsupervised K-Means algorithm show a clear correspondence to the actual activity categories. For example, locomotion-related activities tend to be grouped together, while static postures form their own cluster. This alignment mirrors the distinctions identified by the supervised classification models.

Second, both approaches reveal a similar discriminative structure in the feature space. In Section 4, supervised models such as SVM and Random Forest achieved high accuracy, particularly in separating dynamic from static activities. The clustering results reinforce this observation, indicating that the extracted features naturally create separable regions that reflect real behavioral differences.

Finally, the coherence of the clusters demonstrates that the structure in the data is meaningful rather than random. If the clusters were highly mixed, it would suggest weak underlying patterns. Instead, the clustering process reproduced clear and interpretable activity groups, thereby validating and strengthening the conclusions of the supervised learning analysis.

Final Project Summary

This project presented a complete implementation of the KDD (Knowledge Discovery in Databases) process applied to the Human Activity Recognition (HAR) dataset. Beginning with data understanding and preprocessing, we performed cleaning, normalization, and dimensionality reduction using PCA to manage the complexity of over 560 features. Three supervised learning models—**Decision Tree, Random Forest, and SVM**—were trained and evaluated, with SVM achieving the highest performance across all metrics (Accuracy, Precision, Recall, F1, AUC). These results demonstrated that the dataset contains clear discriminative patterns that enable highly accurate activity classification.

In the fourth stage, a detailed interpretability analysis was performed. The structure of the decision tree was examined through multiple zoomed visualizations, allowing insight into how specific sensor features contribute to decision boundaries. Although the full tree is large and difficult to interpret directly, focusing on selected meaningful nodes provided interpretable examples of how specific signals influence activity predictions.

The final stage explored unsupervised clustering using **K-Means**. Through the Elbow method, four clusters were selected as the optimal number. Visualizations—including PCA-based projections and feature-specific scatter plots—revealed that the clusters naturally separate dynamic from static activities. Cluster feature analyses and centroid examination showed strong alignment between cluster behaviors and true activity labels. This alignment confirms that even without supervision, the intrinsic structure of the dataset reflects meaningful human activity patterns.

Overall, the convergence of results from both supervised and unsupervised methods demonstrates the robustness of the dataset and validates the classification models previously developed. The project successfully illustrates the full KDD pipeline—from raw data to interpretable machine learning insights—and highlights the strong analytical value of signal-based activity recognition

Appendixes:

True Label	1	2	3	4	5	6
Cluster						
0	74	12	16	943	1018	0
1	367	798	155	276	353	16
2	785	263	815	1	3	0
3	0	0	0	66	0	1391

```
label_mapping = {
    1: "WALKING",
    2: "WALKING_UPSTAIRS",
    3: "WALKING_DOWNSTAIRS",
    4: "SITTING",
    5: "STANDING",
    6: "LAYING"
}
```

Table 11: 4 clusters table.

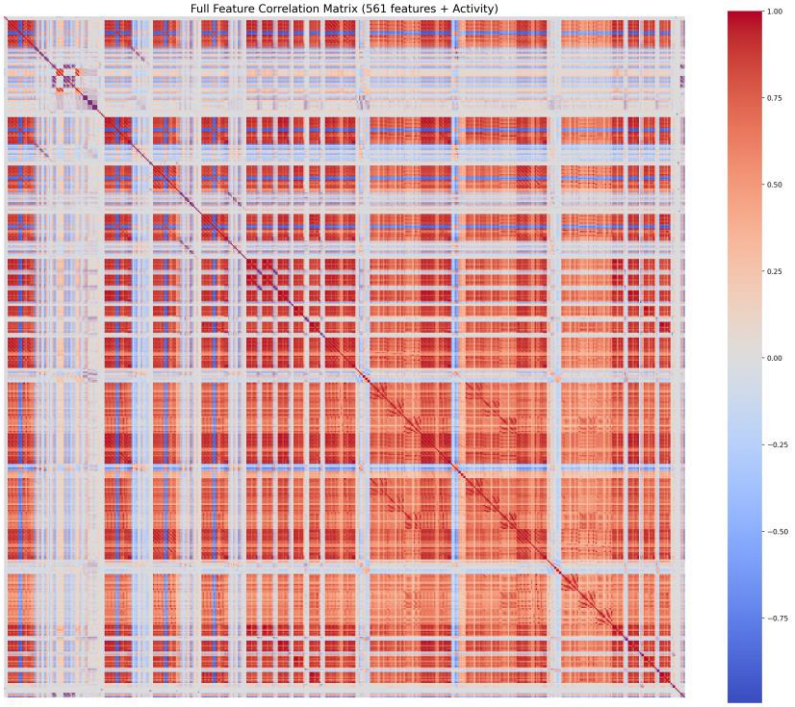

Figure 17: Full Feature Correlation Matrix.

Documentation:

All the project parts are documented in my GitHub repository that you can find in this link.

In that repository you will find:

- Readme.md file that contains general explanation.
- The dataset that I have used.
- Jupyter file that contains the python code and its output.
- Plots file that contains the plots that I got.
- This report.