The International Journal of Embedded Computer Engineering (IJEmCE), Vol. 2, No. 2, October 2026.
https://ejurnal.ung.ac.id/index.php/IJEmCE

Analysis and Preprocessing of Accelerometer Sensor Data

Revalina Dwi Putri Hasan[1],Rahmad Hidayat Dongka[2*],Ulfatun Nadifa[3], Ade Irawaty Tolago[4], Wildan[5]
Computer Engineering, Universitas Negeri Gorontalo, Indonesia[1,3]
Electrical Engineering, Universitas Negeri Gorontalo, Indonesia[2,4,5]
*Corresponding Author Email: rahmatdongka@ung.ac.id

***Abstract* —** Accelerometer sensor data obtained directly from a device is generally still raw data that may contain missing values, differences in feature scale, time-format errors, and extreme values (outliers). This study aims to analyze and preprocess an accelerometer dataset consisting of 27,937 observations of acceleration along the X, Y, and Z axes together with the recording date and time. The analysis was carried out using the Pipeline method from the Scikit-learn library to clean, process, and transform the data in a structured manner, starting from converting the date and time format, extracting time-based features, standardizing the numerical data with a StandardScaler, and detecting and handling outliers with the Interquartile Range (IQR) method and a capping technique. The inspection results show that the dataset has no missing values, so imputation is not required, and the transformation process produces 11 features. The Pearson correlation heatmap visualization shows that the three acceleration axes have a weak and relatively independent relationship with one another, so all three still need to be retained as separate features. As a result, the dataset becomes cleaner, more stable, and ready to be used for further analysis such as activity classification and sensor-based machine learning modeling.

Copyright © 2026 IJEmCE. All rights reserved.

Keywords:
Accelerometer;
Correlation Analysis;
Pipeline Method;
Outlier;
Data Preprocessing;

Article History:
Received: June 1, 2026
Revised: July 1, 2026
Accepted: Octo 1, 2026
Published: Octo 31, 2026

I. INTRODUCTION

The development of information technology and digital sensors today generates large volumes of data that are real-time and complex in nature. One type of data widely used in various fields such as health, sports, the Internet of Things (IoT), and activity-monitoring systems is data from accelerometer sensors [1]. An accelerometer sensor can measure acceleration along three main axes, namely the X, Y, and Z axes, so that it can represent the movement of an object in detail [2].

However, accelerometer data obtained directly from a sensor is generally still raw data. Raw data often contains various issues, such as missing values, differences in feature scale, time-format errors, noise, and the presence of extreme values (outliers) [3], [4]. If such data is used directly without going through a processing step, the resulting analysis or model may become inaccurate and biased.

Therefore, an important step called data preprocessing is required. Preprocessing aims to clean, transform, and prepare data so that it is suitable for further analysis [5]. One approach widely used in modern data analysis is the Pipeline method, an analytical workflow that integrates data cleaning, feature transformation, and modeling into a single organized system [6]. This approach allows every stage of the analysis to be carried out automatically and repeatedly, thereby improving the efficiency and consistency of the analysis results.

In this study, the accelerometer dataset is analyzed through several preprocessing stages using the Pipeline method, such as date and time transformation, numerical data standardization, correlation analysis between columns, and outlier handling using a capping technique. By carrying out these stages, the dataset is expected to become more stable, informative, and ready for the development of an analytical or machine learning model in a subsequent stage.

Based on this background, this study aims to analyze and preprocess the accelerometer dataset by making use of the Pipeline method. This study is expected to provide an overview of a systematic sensor-

data analysis procedure, from data cleaning through to the interpretation of the results, and to contribute to the development of more effective sensor data analysis methods to support IoT-based applications and activity-monitoring systems.

II. METHOD

This study uses a quantitative approach based on data analysis to examine the characteristics of and the relationships between variables in the accelerometer sensor dataset. The dataset used is sourced from Kaggle [7] and consists of 27,937 rows of data with 5 variables, namely Date, Time, *accel_x*, *accel_y*, and *accel_z*.

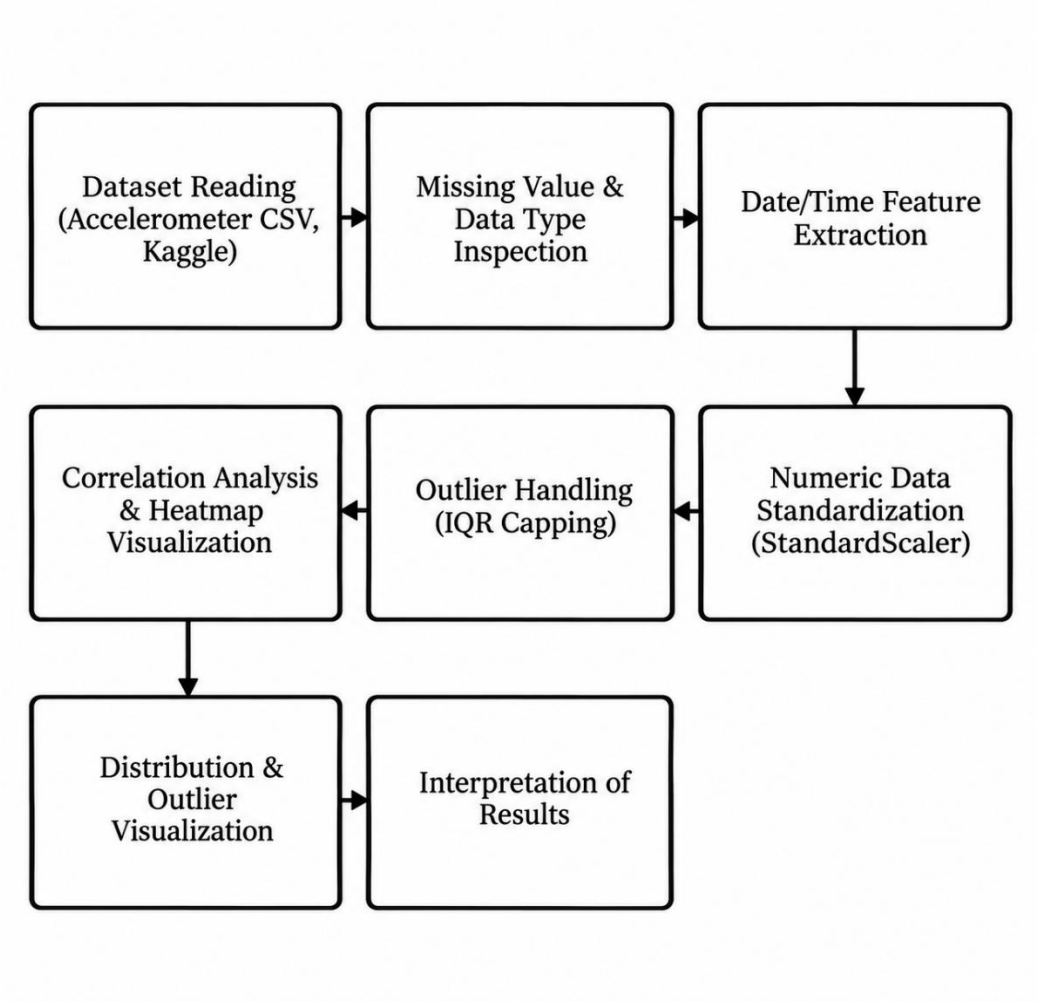


FIGURE 1.
Research Flow Diagram

The data analysis procedures used in this study are described as follows.

1) Dataset Reading and Exploration: The accelerometer dataset is read using the pd.read_csv() function from the pandas library, and then its first five rows are displayed with df.head() and its complete structure with df.info(). Based on these results, the dataset consists of 27,937 rows and 5 columns, with a float64 data type for *accel_x*, *accel_y*, and *accel_z*, and an object data type for Date and Time.

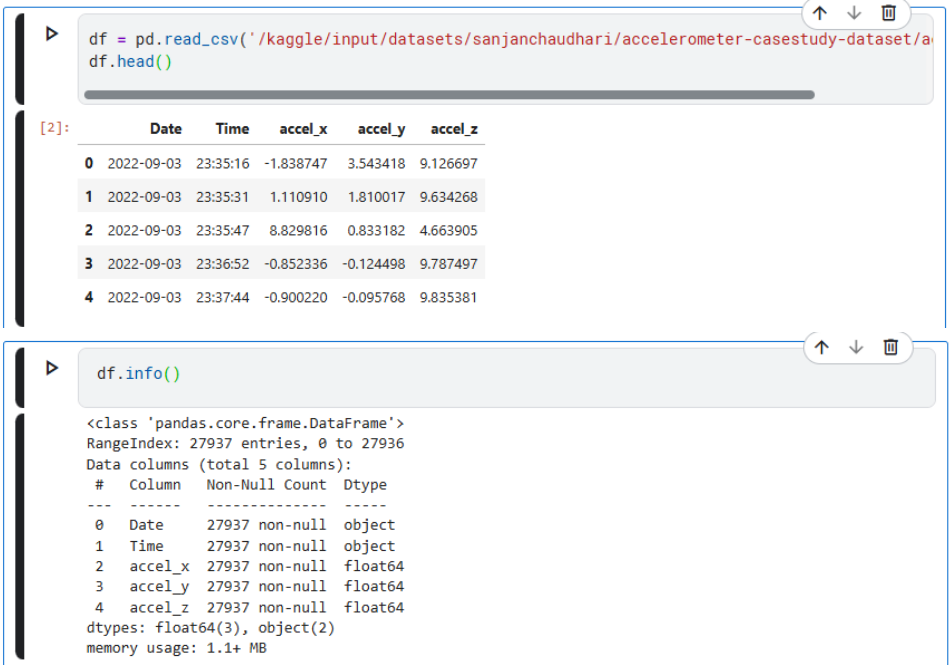


FIGURE 2.
Dataset Reading and Initial Structure

The following is a brief description of each column in the dataset:

- Date: the recording date of the sensor data.
- Time: the recording time of the sensor data.
- accel_x: the acceleration value along the X axis.
- accel_y: the acceleration value along the Y axis.
- accel_z: the acceleration value along the Z axis.

2) Missing Value Inspection: Each column is checked for missing values using the df.isna().sum() function. The check results show that every column has a value of 0, which means the dataset has no missing values, so no imputation process is required.

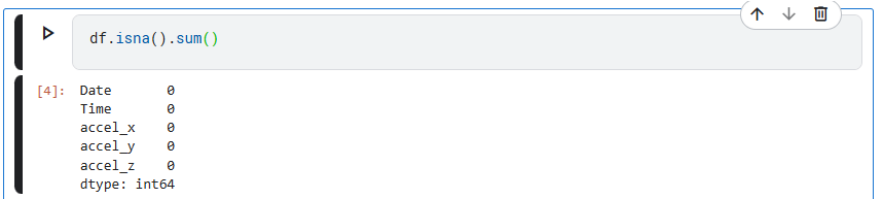


FIGURE 3.
Missing Value Inspection in the Dataset

3) Data Type Separation: The dataset columns are separated into numerical and categorical columns using the select_dtypes() function. The *accel_x*, *accel_y*, and *accel_z* columns are grouped as numerical columns (num_cols), while Date and Time are grouped as categorical columns (cat_cols). This separation is required because the two types of columns need different handling in the subsequent pipeline stage.

```
num_cols = df.select_dtypes(include=['int64','float64']).columns
cat_cols = df.select_dtypes(include=['object']).columns
print(f"num_col: {num_cols}")
print(f"cat_col: {cat_cols}")
```

```
num_col: Index(['accel_x', 'accel_y', 'accel_z'], dtype='object')
cat_col: Index(['Date', 'Time'], dtype='object')
```

+ Code + Markdown

```
import pandas as pd
import numpy as np

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTransformer
from sklearn.impute import SimpleImputer
```

FIGURE 4.
Data Type Separation and Library Imports

4) Date and Time Feature Extraction: The Date and Time columns are processed using the extract_date and extract_time functions. Both functions convert the values to datetime format with the errors='coerce' parameter so that values that do not match the expected format are turned into NaN, and then extract the year, month, day, day of week, and quarter from Date, as well as the hour, minute, and second from Time. Values that fail to convert are filled with 0.

```
num_cols = [
'accel_x', 'accel_y', 'accel_z'
]

cat_cols = [
'Date', 'Time'
]

date_cols = ['Date']
time_cols = ['Time']
```

```
def extract_date(X):
    df = pd.DataFrame(X, columns=['Date'])
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

    out = pd.DataFrame({
        'year': df['Date'].dt.year.fillna(0),
        'month': df['Date'].dt.month.fillna(0),
        'day': df['Date'].dt.day.fillna(0),
        'dayofweek': df['Date'].dt.dayofweek.fillna(0),
        'quarter': df['Date'].dt.quarter.fillna(0)
    })

    return out.values
```

```
def extract_time(X):
    df = pd.DataFrame(X, columns=['Time'])
    df['Time'] = pd.to_datetime(df['Time'], format='%H:%M:%S', errors='coerce')

    out = pd.DataFrame({
        'hour': df['Time'].dt.hour.fillna(0),
        'minute': df['Time'].dt.minute.fillna(0),
        'second': df['Time'].dt.second.fillna(0)
    })

    return out.values
```

FIGURE 5.
Date and Time Feature Extraction Functions

5) Pipeline Construction and Data Transformation: The entire data-cleaning and transformation process is combined into a single Pipeline object from the Scikit-learn library. The numerical columns are processed through a SimpleImputer with a median strategy and a StandardScaler, while the Date and Time columns are processed through a FunctionTransformer wrapping the extract_date and extract_time functions. The entire pipeline is combined using a ColumnTransformer into a single preprocessor object, which is then applied to the dataset with fit_transform() [6], [8], [9].

```
X_trans = preprocessor.fit_transform(X)

print("Shape asli transform:", X_trans.shape)

feature_names = []

# numeric
feature_names += num_cols

# date
feature_names += ['year','month','day','dayofweek','quarter']

# time
feature_names += ['hour','minute','second']

print("Jumlah nama fitur :", len(feature_names))
print("Jumlah kolom real :", X_trans.shape[1])

X_clean = pd.DataFrame(X_trans, columns=feature_names)

print("NaN tersisa:", X_clean.isna().sum().sum())

X_clean.head()
```

```
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

date_pipeline = Pipeline([
    ('date', FunctionTransformer(extract_date, validate=False))
])

time_pipeline = Pipeline([
    ('time', FunctionTransformer(extract_time, validate=False))
])
```

FIGURE 6.
Pipeline Construction and Data Transformation Code

```
Shape asli transform: (27937, 11)
Jumlah nama fitur : 11
Jumlah kolom real : 11
NaN tersisa: 0
```

	accel_x	accel_y	accel_z	year	month	day	dayofweek	quarter	hour	minute	second
0	-0.776988	0.540877	0.620091	2022.0	9.0	3.0	5.0	3.0	23.0	35.0	16.0
1	0.097895	0.177377	0.710573	2022.0	9.0	3.0	5.0	3.0	23.0	35.0	31.0
2	2.387362	-0.027469	-0.175468	2022.0	9.0	3.0	5.0	3.0	23.0	35.0	47.0
3	-0.484413	-0.228298	0.737889	2022.0	9.0	3.0	5.0	3.0	23.0	36.0	52.0
4	-0.498616	-0.222273	0.746425	2022.0	9.0	3.0	5.0	3.0	23.0	37.0	44.0

FIGURE 7.
Output of the Data Transformation

The transformation results in 11 features with a data shape of (27937, 11) and no remaining missing values, as shown in Figure 7.

6) Outlier Handling with Capping: An outlier is a data point that differs significantly from the other observations. Various outlier-detection strategies have been developed and classified according to data characteristics, such as high-dimensional data and streaming data, taking into account both the efficiency and the accuracy of detection [10]. In this study, outliers in the numerical data are detected and handled using the Interquartile Range (IQR) method [3], [11]. The IQR value, the lower bound, and the upper bound are determined according to (1), (2), and (3):

$$\text{IQR} = Q_3 - Q_1 \tag{1}$$

$$\text{Lower bound} = Q_1 - 1.5 \times \text{IQR} \tag{2}$$

$$\text{Upper bound} = Q_3 + 1.5 \times \text{IQR} \tag{3}$$

where Q_1 is the first quartile (the 25th percentile) and Q_3 is the third quartile (the 75th percentile). Values that fall outside these bounds are bounded (capped) using the clip() function so that they are not removed from the dataset, thereby preserving the number of observations.

```python
# ============================
# CAPPING OUTLIER
# ============================

def capping(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1

    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    return df.clip(lower, upper, axis=1)


X_final = capping(X_clean)

print("Setelah capping:")
print("Shape:", X_final.shape)
print("NaN:", X_final.isna().sum().sum())

X_final.head()
```

FIGURE 8.
Code and Output of Outlier Handling with Capping

7) Correlation Analysis and Heatmap Visualization: Correlation analysis is carried out using the Pearson correlation coefficient with a threshold of 0.9 to select the most correlated features, which are then visualized as a heatmap using the matplotlib library.

```python
# ============================
# HITUNG KORELASI
# ============================
corr = X_clean.corr().abs()

# threshold bisa diubah
threshold = 0.9

# fitur yang punya korelasi tinggi
high_corr_features = corr.columns[(corr > threshold).any()]

print("Jumlah fitur berkorelasi tinggi:", len(high_corr_features))
print(high_corr_features.tolist())

# ============================
# AMBIL KORELASI FITUR TERPILIH
# ============================
corr_high = X_clean[high_corr_features].corr().round(2)
```

```python
# ============================
# PLOT HEATMAP
# ============================
plt.figure(figsize=(10, 8))
plt.imshow(corr_high)
plt.colorbar()
plt.title("High Correlation Features")

plt.xticks(range(len(corr_high.columns)), corr_high.columns, rotation=90, fontsize=8)
plt.yticks(range(len(corr_high.columns)), corr_high.columns, fontsize=8)

for i in range(len(corr_high.columns)):
    for j in range(len(corr_high.columns)):
        plt.text(j, i,
                 corr_high.iloc[i, j],
                 ha="center",
                 va="center",
                 fontsize=7)

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

```
Jumlah fitur berkorelasi tinggi: 8
['accel_x', 'accel_y', 'accel_z', 'day', 'dayofweek', 'hour', 'minute', 'second']
```

FIGURE 9.
Correlation Computation and Heatmap Configuration

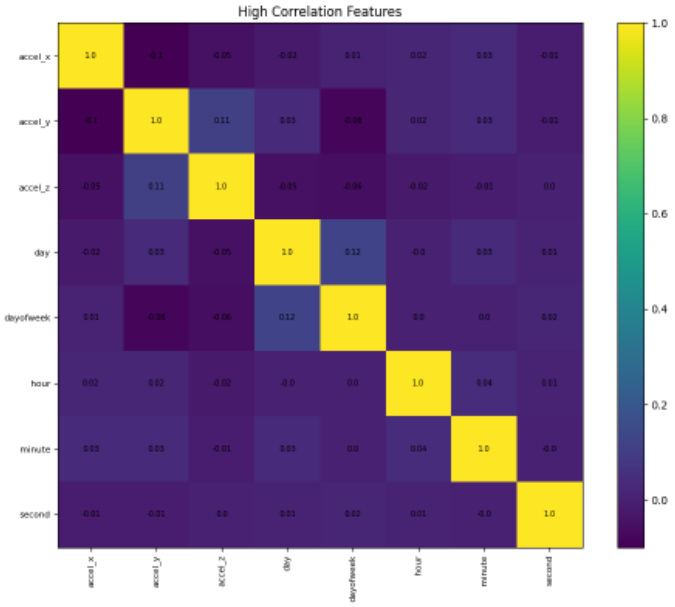


FIGURE 10.
Correlation Heatmap Between Features

The visualization results show that almost all pairs of features, including the three acceleration axes *accel_x*, *accel_y*, and *accel_z*, have correlation values that are weak and close to zero (ranging from about −0.05 to 0.13), which indicates that the three axes are relatively independent of one another. The pair of features with the highest correlation is day and dayofweek (about 0.12), which still falls into the weak-relationship category [12], [13].

8) Data Distribution Visualization: The distribution of each column is analyzed using histograms and a normal curve. For each column, a skewness value is computed, and the direction of the distribution is then classified as right-skewed (skewness > 0.5), left-skewed (skewness < −0.5), or normal.

```python
from scipy.stats import norm
import math

cols = X_final.columns
n = len(cols)

# jumlah kolom subplot (bisa ubah)
ncols = 8
nrows = math.ceil(n / ncols)

plt.figure(figsize=(4*ncols, 3*nrows))

for i, col in enumerate(cols):
    data = X_final[col].dropna()  # hindari error jika ada NaN

    ax = plt.subplot(nrows, ncols, i+1)

    # Histogram (density)
    ax.hist(data, bins=30, density=True)

    # Kurva normal pembanding
    mu = data.mean()
    sigma = data.std()

    if sigma > 0:  # hindari error jika std = 0
        x = np.linspace(data.min(), data.max(), 100)
        ax.plot(x, norm.pdf(x, mu, sigma))

    # Cek arah skewness
    skew = data.skew()

    if skew > 0.5:
        arah = "Skew Kanan"
    elif skew < -0.5:
        arah = "Skew Kiri"
    else:
        arah = "Normal"

    ax.set_title(f"{col}\n{arah}", fontsize=8)
    ax.set_xticks([])
    ax.set_yticks([])

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

FIGURE 11.
Program Code for Data Distribution Visualization

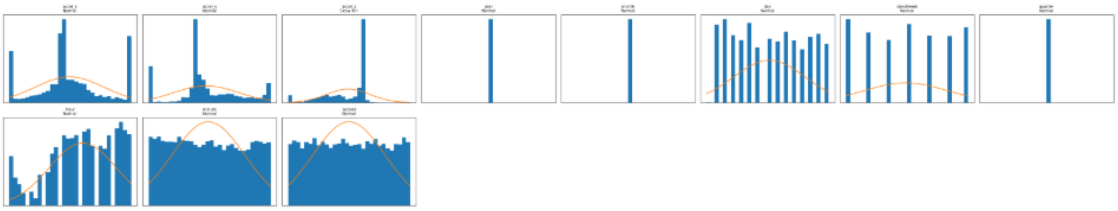


FIGURE 12.
Histograms and Normal Curve for Each Column

9) Outlier Visualization with Boxplots: The outlier values still present after the capping process are visualized using boxplots for each column, in order to confirm that the data distribution has become more stable compared with before the outlier handling was applied.

```python
cols = X_final.columns
n = len(cols)

# jumlah kolom per baris
ncols = 8
nrows = math.ceil(n / ncols)

plt.figure(figsize=(4*ncols, 3*nrows))

for i, col in enumerate(cols):
    ax = plt.subplot(nrows, ncols, i+1)

    data = X_final[col].dropna()

    ax.boxplot(data, vert=True)
    ax.set_title(col, fontsize=8)
    ax.set_xticks([])

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

FIGURE 13.
Program Code for Boxplot Visualization

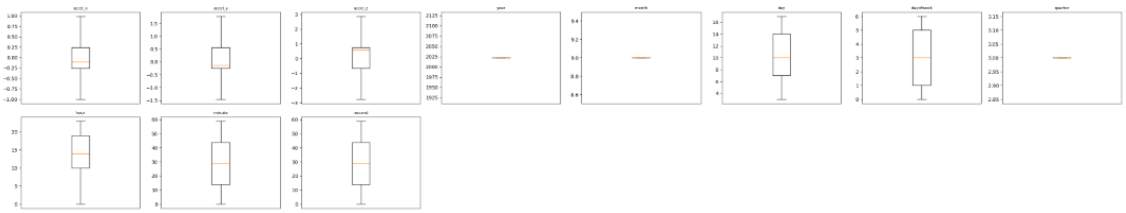


FIGURE 14.
Boxplots for Each Column

III. RESULTS AND DISCUSSION

A. Results

Based on the analysis carried out, the preprocessing stage using the Pipeline method successfully turned the accelerometer dataset, which originally had mixed data types (object and numeric), into 11 clean and consistent numerical features with no remaining missing values (Figure 7). Outlier handling with the capping technique successfully bounded extreme values in each column without reducing the number of observations, so that the data shape remained (27937, 11) after this process was applied (Figure 8).

The correlation heatmap visualization (Figure 10) shows that the relationships between features, including between the three acceleration axes, are generally weak. Meanwhile, the histogram (Figure 12) and boxplot (Figure 14) visualizations show that most columns have an asymmetric (skewed) distribution and still exhibit a small number of potential outliers even after the capping process.

B. Discussion

The weak correlation values between the acceleration axes indicate that *accel_x*, *accel_y*, and *accel_z* carry relatively independent motion information from one another, so all three need to be retained as separate features in further analysis or modeling, rather than being reduced into a single combined variable. This finding is consistent with a study on human activity recognition based on accelerometer and gyroscope sensors, which made use of all three acceleration axes simultaneously to distinguish movement patterns [1].

The use of an IQR-based capping technique in this study was chosen because it can control extreme values without sacrificing the amount of data, as shown by a comparative study of outlier-detection methods that highlighted how an IQR-based approach is relatively simple yet remains effective compared to more complex methods such as Isolation Forest [11]. Likewise, standardizing the numerical data using a StandardScaler in the Pipeline stage is consistent with findings showing that data-scaling has a significant effect on machine learning algorithm performance in a subsequent modeling stage [14].

Overall, the use of the Pipeline method proved advantageous in terms of efficiency and process consistency, since the entire cleaning, transformation, and scaling stages could be run automatically and repeatedly within a single structured workflow [6]. The Exploratory Data Analysis (EDA) approach used during the visualization stage also helped reveal distribution patterns and potential outliers that are not visible from summary statistics alone [15]. Consequently, the preprocessed dataset is considered sufficiently clean and representative for use in further analysis, such as activity classification or sensor-based machine learning modeling.

IV. CONCLUSION

Based on the analysis and preprocessing carried out on the accelerometer sensor dataset using the Pipeline method, several conclusions can be drawn as follows.

First, the preprocessing stages, which include data reading, missing-value inspection, data type separation, date and time feature extraction, and standardization using a StandardScaler, were successfully carried out in a structured manner through a single Pipeline object, producing a clean dataset with 11 features and no missing values.

Second, outlier handling with the Interquartile Range (IQR) method and a capping technique successfully bounded extreme values in each column without removing any data, so that the number of observations was preserved while the data distribution became more stable.

Third, the correlation heatmap visualization shows that the X, Y, and Z acceleration axes have a weak and relatively independent relationship with one another, so all three still need to be retained as separate features in further analysis or modeling.

Fourth, overall, the preprocessed dataset is cleaner, more stable, and ready to be used for further analysis, such as activity classification, movement pattern prediction, or the implementation of a sensor-based machine learning model, so that it can support the development of IoT-based applications and activity-monitoring systems in the future.

V. REFERENCES

[1] Arifien, Z., Bachtiar, F. A., & Yudistira, N. (2022). Pengenalan aktivitas manusia menggunakan sensor akselerometer dan giroskop pada smartphone dengan metode K-Nearest Neighbor [Human activity recognition using accelerometer and gyroscope sensors on a smartphone with the K-Nearest Neighbor method]. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK), 9*(1), 173–180. https://doi.org/10.25126/jtiik.202295593

[2] Arduino Indonesia. (2022, December). *Accelerometer: Pengertian, cara kerja, dan jenis-jenisnya* [Accelerometer: Definition, working principle, and types]. https://www.arduinoindonesia.id/2022/12/accelerometer-pengertian-cara-kerja-dan-jenis-jenisnya.html

[3] Han, J., Kamber, M., & Pei, J. (2012). *Data mining: Concepts and techniques* (3rd ed.). Morgan Kaufmann.

[4] Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data mining: Practical machine learning tools and techniques* (4th ed.). Morgan Kaufmann.

[5] Januaji, D. (2022, August 22). *Data preprocessing: Pengertian, manfaat, dan tahapan kerjanya* [Data preprocessing: Definition, benefits, and work stages]. Accurate. https://accurate.id/teknologi/data-preprocessing/

[6] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

[7] sanjanchaudhari. (n.d.). *Accelerometer case study dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/sanjanchaudhari/accelerometer-casestudy-dataset

[8] Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed.). O'Reilly Media.

[9] McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

[10] Boukerche, A., Zheng, L., & Alfandi, O. (2020). Outlier detection: Methods, models, and classification. *ACM Computing Surveys, 53*(3), 1–37. https://doi.org/10.1145/3381028

[11] Ramadhani, J. W., & Susanto. (2026). Perbandingan metode IQR vs Isolation Forest dalam deteksi outlier pada data klaim asuransi di PT Askrindo Cabang Semarang [Comparison of the IQR and Isolation Forest methods for outlier detection in insurance claim data at PT Askrindo Semarang Branch]. *Jurnal Transformatika, 23*(2), 268–277. https://doi.org/10.26623/transformatika.v23i2.13726

[12] Accurate. (n.d.). *Koefisien korelasi* [Correlation coefficient]. https://accurate.id/akuntansi/koefisien-korelasi/

[13] Sugiyono. (2016). *Statistika untuk penelitian* [Statistics for research] (Cet. 27). Alfabeta.

[14] Ambarwari, A., Adrian, Q. J., & Herdiyeni, Y. (2020). Analisis pengaruh data scaling terhadap performa algoritma machine learning untuk identifikasi tanaman [Analysis of the effect of data scaling on machine learning algorithm performance for plant identification]. *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi), 4*(1), 117–122. https://doi.org/10.29207/resti.v4i1.1517

[15] Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley.