Task 2: Z-Score-Based Outlier Detection Report

Objective: To implement Z-score-based outlier detection on the dataset and flag anomalies based on Z-score values.

1. Review of Z-Score Method:

The Z-score is a statistical measure that represents the number of standard deviations a data point is from the mean. It is useful for identifying outliers in a dataset by showing how far a value deviates from the norm. In outlier detection, Z-scores allow us to flag anomalies that are significantly higher or lower than the average.

Relevance to Outlier Detection:

- Z-scores standardize different data points, making it easier to identify points that are out of the ordinary.
- Data points with Z-scores greater than 3 or less than -3 are typically considered outliers, as they deviate more than three standard deviations from the mean.

2. Z-Score-Based Outlier Detection Implementation:

Step 1: Load the Dataset

- We begin by loading the dataset into a Pandas DataFrame.

```
python
Copy code
aug_df = pd.read_csv('augmented_dataset.csv')
```

Step 2: Data Inspection

- Check for missing values in the relevant columns (`Speed`, `Acceleration_Rate`, `Jerk`) and handle them accordingly.

```
python
print(aug_df[['Speed', 'Acceleration_Rate',
'Jerk']].isna().sum())
```

- **Handling NaN Values:**
  - We fill NaN values for `Acceleration_Rate` with 0 and drop rows with missing values.
  - For the `Jerk` column, missing values are replaced with 0.

```
python
Copy code
aug_df['Acceleration_Rate'] =
aug_df['Acceleration_Rate'].fillna(0)
aug_df['Jerk'] = aug_df['Jerk'].fillna(0)
```

Step 3: Handle Infinite Values

- Infinite values (`inf` or `-inf`) might exist in columns like `Jerk`. These need to be replaced before calculating Z-scores.

```
python
Copy code
# Replace infinite values with NaN and fill NaN
values with 0
aug_df.replace([np.inf, -np.inf], np.nan,
inplace=True)
aug_df.fillna(0, inplace=True)
```

Step 4: Calculate Z-Scores

- Z-scores are calculated for the relevant features: `Speed`, `Acceleration_Rate`, and `Jerk`.

```
python
Copy code
from scipy.stats import zscore

# Calculate Z-scores for the selected columns
z_scores = aug_df[['Speed', 'Acceleration_Rate',
'Jerk']].apply(zscore)
```

Step 5: Flag Anomalies Based on Z-Scores

- After calculating Z-scores, anomalies are flagged where the absolute value of the Z-score is greater than 3.

```
python
Copy code
# Flag anomalies where Z-score > 3 or Z-score < -3
aug_df['Anomaly'] = (np.abs(z_scores) >
3).any(axis=1).astype(int)
```

Step 6: Save the Updated Dataset

- The updated dataset, including flagged anomalies, is saved for further analysis.

```
python
Copy code
aug_df.to_csv('augmented_dataset.csv', index=False)
```

Step 7: Display Anomalies

- Anomalies (rows where the `Anomaly` column equals 1) are displayed for review.

```
python
Copy code
anomalies = aug_df[aug_df['Anomaly'] == 1]
print(anomalies)
```

3. Results and Observations

- **Initial Data Inspection:**
  - The dataset contained some NaN values in the `Acceleration_Rate` and `Jerk` columns, which were successfully handled by filling with 0.
- **Handling of Infinite Values:**
  - Infinite values in the `Jerk` column were replaced and handled properly before Z-score calculation to avoid errors.
- **Z-Score Calculation:**
  - Z-scores were calculated for `Speed`, `Acceleration_Rate`, and `Jerk`. The `RuntimeWarning` related to invalid values was resolved after handling NaN and infinite values.
- **Anomaly Detection:**
  - Anomalies were flagged using the Z-score method with a threshold of ±3. However, the initial detection resulted in **no anomalies**, indicating the dataset might be well within normal ranges or further tuning of thresholds might be required.

4. Next Steps and Recommendations

1. **Threshold Adjustment:**
   If no anomalies are detected, consider lowering the Z-score threshold from ±3 to a lower value (e.g., ±2.5) to identify more subtle anomalies.
2. **Feature Scaling:**
   Ensure that features like `Speed`, `Acceleration_Rate`, and `Jerk` are on a similar scale before calculating Z-scores. If necessary, apply normalization techniques.
3. **Visualization:**
   Visualizing the Z-scores using histograms or scatter plots could help understand the distribution of values and identify areas where anomalies may exist.

New Features Created

During the course of outlier detection and feature engineering, several new features were created to enhance the dataset and improve the model's ability to detect anomalies.

1. Acceleration_Rate

- **Description**: This feature measures the rate of acceleration over time, capturing how quickly the velocity changes.

- **Calculation**:

```
python
Copy code
aug_df['Acceleration_Rate'] =
aug_df['Speed_Change'] / aug_df['Time_Change']
```

NaN Handling: NaN values in this feature were filled with 0.

2. Jerk

- **Description**: The jerk is the rate of change of acceleration and provides a higher-order measure of movement dynamics.
- **Calculation**:

```
python
Copy code
aug_df['Jerk'] =
aug_df['Acceleration_Rate'].diff()
```

NaN Handling: NaN values were handled by filling with 0, and infinite values were replaced with NaN and then filled.

3. Braking_Intensity

- **Description**: This feature measures the intensity of braking events by identifying sudden decreases in speed.
- **Creation Logic**:

```
python
Copy code
aug_df['Braking_Intensity'] =
(aug_df['Acceleration_Rate'] < 0).astype(int)
```

4. Cumulative_Distance

- **Description**: This feature keeps a running total of the distance covered over time, helping to track the overall movement pattern.
- **Creation Logic**:

```
python
Copy code
aug_df['Cumulative_Distance'] =
aug_df['Distance'].cumsum()
```

5. Speed_Variance

- **Description**: Measures the variance in speed over a rolling window, capturing fluctuations in the speed pattern.
- **Creation Logic**:

```
python
Copy code
aug_df['Speed_Variance'] =
aug_df['Speed'].rolling(window=3).var()
```

6. Rolling_Mean_AccX

- **Description**: The rolling mean of the acceleration in the X direction, helping to smooth out short-term fluctuations.
- **Creation Logic**:

```
python
Copy code
aug_df['Rolling_Mean_AccX'] = aug_df['Acc
X'].rolling(window=5).mean()
```

7. Variance_GyroX

- **Description**: The variance of the gyroscope readings in the X direction over a rolling window, providing insight into rotational stability.
- **Creation Logic**:

```
python
Copy code
aug_df['Variance_GyroX'] =
aug_df['gyro_x'].rolling(window=3).var()
```

8. Total_Acc

- **Description**: Combines the magnitude of acceleration in all directions, providing a holistic view of overall movement.
- **Creation Logic**:

```
python
Copy code
aug_df['Total_Acc'] = np.sqrt(aug_df['Acc X']**2
+ aug_df['Acc Y']**2 + aug_df['Acc Z']**2)
```

These new features were designed to enrich the dataset and enhance its capacity for anomaly detection. They are integral in providing deeper insights into crowd behavior and movement patterns.