1.Importing all the required Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import os
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scor
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import learning_curve
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
```

2.EDA

In [2]:
```
activity_codes_mapping = {
    'A': 'walking',
    'B': 'jogging',
    'C': 'stairs',
    'D': 'sitting',
    'E': 'standing',
    'F': 'typing',
    'G': 'brushing teeth',
    'H': 'eating soup',
    'I': 'eating chips',
    'J': 'eating pasta',
    'K': 'drinking from cup',
    'L': 'eating sandwich',
    'M': 'kicking soccer ball',
    'O': 'playing catch tennis ball',
    'P': 'dribbling basket ball',
    'Q': 'writing',
    'R': 'clapping',
    'S': 'folding clothes'
}

activity_color_map = {
    activity_codes_mapping['A']: 'lime',
    activity_codes_mapping['B']: 'red',
    activity_codes_mapping['C']: 'blue',
    activity_codes_mapping['D']: 'orange',
    activity_codes_mapping['E']: 'yellow',
    activity_codes_mapping['F']: 'lightgreen',
    activity_codes_mapping['G']: 'greenyellow',
    activity_codes_mapping['H']: 'magenta',
    activity_codes_mapping['I']: 'gold',
    activity_codes_mapping['J']: 'cyan',
    activity_codes_mapping['K']: 'purple',
    activity_codes_mapping['L']: 'lightgreen',
    activity_codes_mapping['M']: 'violet',
```

```
    activity_codes_mapping['O']: 'limegreen',
    activity_codes_mapping['P']: 'deepskyblue',
    activity_codes_mapping['Q']: 'mediumspringgreen',
    activity_codes_mapping['R']: 'plum',
    activity_codes_mapping['S']: 'olive'
}
```

In [3]:
```
def show_accel_per_activity(device, df, act, interval_in_sec=None):
    '''Plots acceleration time history per activity'''

    df1 = df.loc[df['Label'] == act].copy()
    df1.reset_index(drop=True, inplace=True)

    df1['Duration'] = (df1['Timestamp'] - df1['Timestamp'].iloc[0]) / 1000  # mill

    if interval_in_sec is None:
        ax = df1.plot(kind='line', x='Duration', y=['X', 'Y', 'Z'], figsize=(25, 7
    else:
        interval = int(interval_in_sec * 1000)  # seconds to milliseconds
        ax = df1[:interval].plot(kind='line', x='Duration', y=['X', 'Y', 'Z'], fig

    ax.set_xlabel('Duration (seconds)', fontsize=15)
    ax.set_ylabel('Acceleration (m/s^2)', fontsize=15)
    ax.set_title('Acceleration: Device: ' + device + '   Activity: ' + act, fontsi
```

In [4]:
```
#Reading the file
data_directory = r"\Users\haris\Downloads\wisdm-dataset\wisdm-dataset\raw\watch\5 p
files = os.listdir(data_directory)

dfs = []  # List to store the dataframes

for file in files:
    if file.endswith(".txt"):
        file_path = os.path.join(data_directory, file)

        df = pd.read_csv(file_path, names=['participant_id', 'activity_code', 'time

        df['z'] = df['z'].str.strip(';')
        df['z'] = pd.to_numeric(df['z'])

        df['activity'] = df['activity_code'].map(activity_codes_mapping)

        df = df[['participant_id', 'activity_code', 'activity', 'timestamp', 'x',

        dfs.append(df)

# Concatenate all the dataframes into a single dataframe
combined_df = pd.concat(dfs, ignore_index=True)

print(combined_df)
```

```
        participant_id activity_code         activity       timestamp  \
0                 1618             A          walking  15001753925717
1                 1618             A          walking  15001803425717
2                 1618             A          walking  15001852925717
3                 1618             A          walking  15001902425717
4                 1618             A          walking  15001951925717
...                ...           ...              ...             ...
193355            1620             S  folding clothes  42368439960621
193356            1620             S  folding clothes  42368489889911
193357            1620             S  folding clothes  42368539819201
193358            1620             S  folding clothes  42368589748491
193359            1620             S  folding clothes  42368639677781

                x         y         z
0        5.824045 -3.594146  2.519897
1        6.113744 -3.876661  2.517503
2        7.521534 -3.251775  3.003526
3        9.101707 -2.995595  3.367445
4       11.905317 -3.433734  3.547010
...           ...       ...       ...
193355   6.524798 -4.589834  0.573262
193356   9.100959 -4.168455 -0.602291
193357   7.590218 -5.736657 -1.854459
193358   8.681973 -6.284929 -5.180005
193359  10.161591 -3.840449 -3.152116

[193360 rows x 7 columns]
```

In [5]:
```
print(combined_df.columns)
```

```
Index(['participant_id', 'activity_code', 'activity', 'timestamp', 'x', 'y',
       'z'],
      dtype='object')
```

In [6]:
```
#  data of first participant
participant_id = 1618  # Specify the participant ID you want to extract
participant_data = combined_df[combined_df['participant_id'] == participant_id]

# Printing the first participant's data
print(participant_data)
```

```
       participant_id activity_code         activity        timestamp  \
0                1618             A          walking   15001753925717
1                1618             A          walking   15001803425717
2                1618             A          walking   15001852925717
3                1618             A          walking   15001902425717
4                1618             A          walking   15001951925717
...               ...           ...              ...              ...
61293            1618             S  folding clothes  110956475526577
61294            1618             S  folding clothes  110956525455517
61295            1618             S  folding clothes  110956575384457
61296            1618             S  folding clothes  110956625313397
61297            1618             S  folding clothes  110956675242337

              x         y         z
0      5.824045 -3.594146  2.519897
1      6.113744 -3.876661  2.517503
2      7.521534 -3.251775  3.003526
3      9.101707 -2.995595  3.367445
4     11.905317 -3.433734  3.547010
...         ...       ...       ...
61293 -0.974141 -4.424185  6.915951
61294 -1.237503 -8.549395  7.284658
61295 -1.163283 -6.222231  7.509713
61296 -0.686837 -6.011541  7.598299
61297 -0.892738 -5.147234  6.444293

[61298 rows x 7 columns]
```

In [7]:
```
# data of second participant
participant_id = 1619  # Specify the participant ID you want to extract
sec_participant_data = combined_df[combined_df['participant_id'] == participant_id
print(sec_participant_data)
```

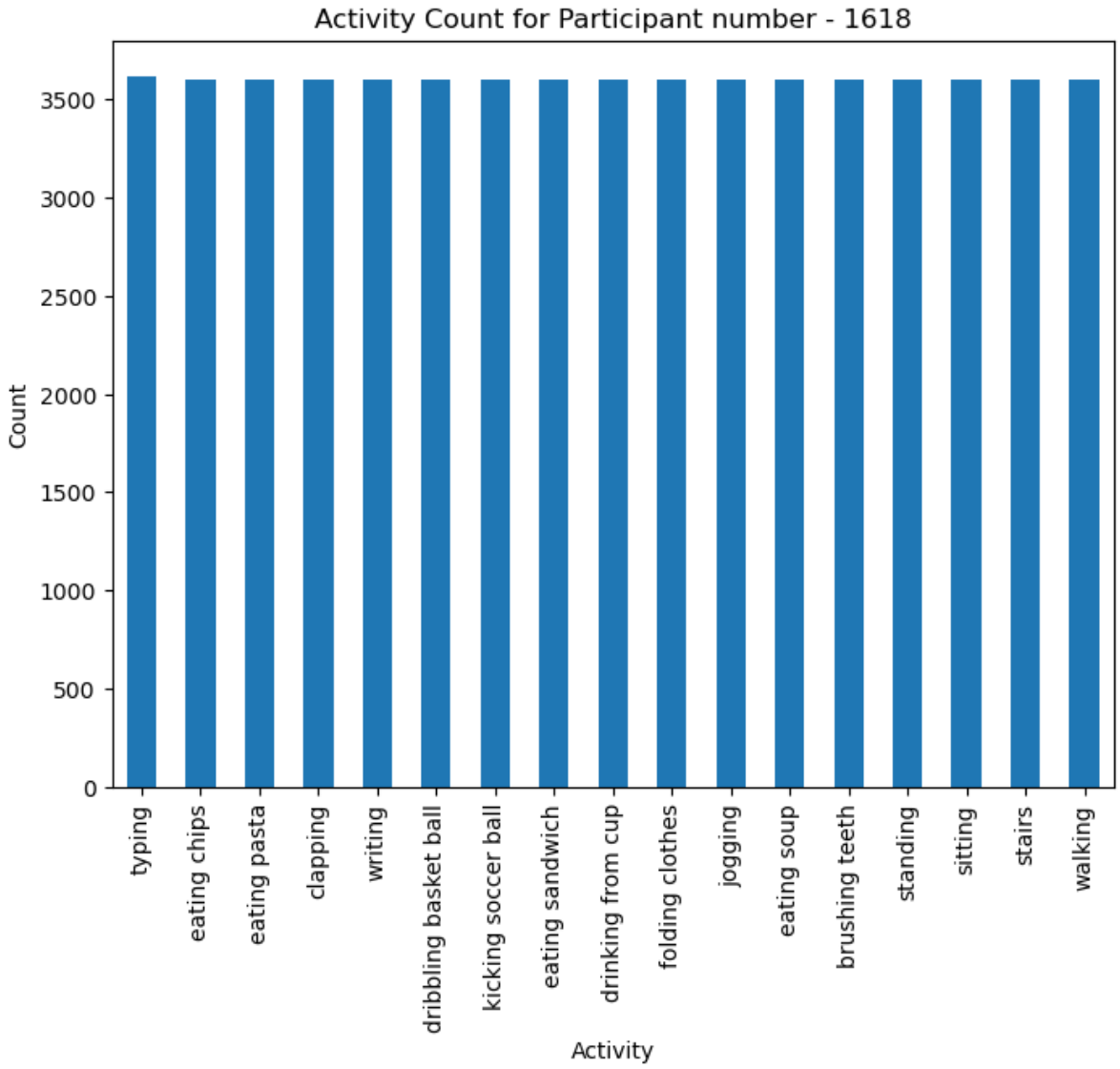
```
       participant_id activity_code         activity        timestamp  \
61298            1619             A          walking  351205245071760
61299            1619             A          walking  351205294571760
61300            1619             A          walking  351205344071760
61301            1619             A          walking  351205393571760
61302            1619             A          walking  351205443071760
...               ...           ...              ...              ...
126161           1619             S  folding clothes  350830196752313
126162           1619             S  folding clothes  350830246708223
126163           1619             S  folding clothes  350830296664133
126164           1619             S  folding clothes  350830346620043
126165           1619             S  folding clothes  350830396575953

              x         y         z
61298  9.306112 -1.640178 -2.385074
61299  8.958953 -1.609053 -2.310853
61300  8.044368 -0.943465 -2.282123
61301  8.848820 -0.177321 -2.598158
61302  8.173655 -0.117466 -2.459294
...         ...       ...       ...
126161 -1.434276 -9.809194  1.771260
126162 -1.817349 -9.758916  0.947655
126163 -1.606659 -9.086145  0.064195
126164 -2.023250 -8.930522 -0.302118
126165 -1.874810 -9.512313 -0.091429

[64868 rows x 7 columns]
```

In [8]:
```
#  data of third participant
participant_id = 1620  # Specify the participant ID you want to extract
```

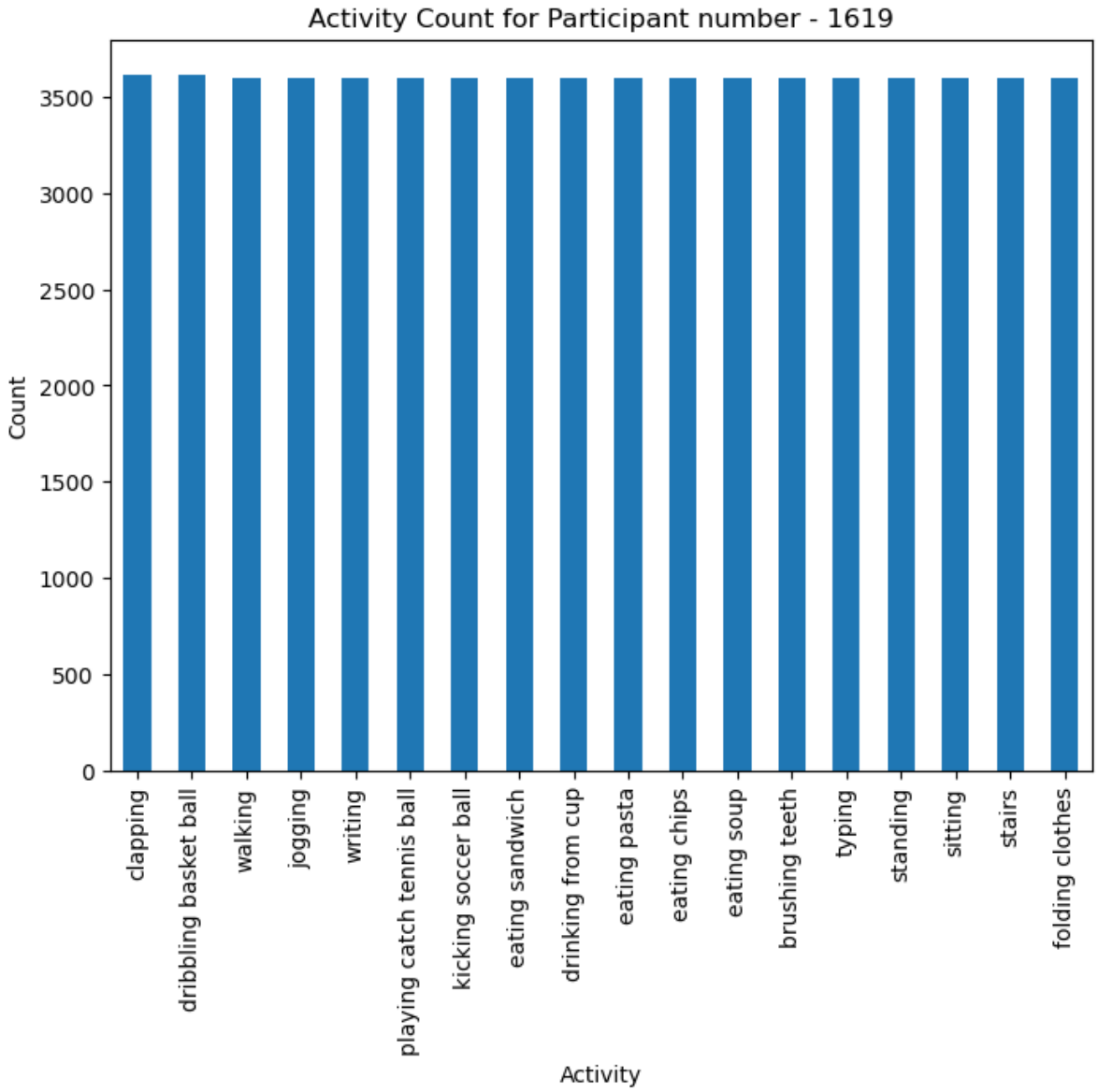
```
thi_participant_data = combined_df[combined_df['participant_id'] == participant_id
print(thi_participant_data)
```

```
        participant_id activity_code         activity       timestamp  \
126166            1620             A          walking  37945621841814
126167            1620             A          walking  37945671341814
126168            1620             A          walking  37945720841814
126169            1620             A          walking  37945770341814
126170            1620             A          walking  37945819841814
...                ...           ...              ...             ...
193355            1620             S  folding clothes  42368439960621
193356            1620             S  folding clothes  42368489889911
193357            1620             S  folding clothes  42368539819201
193358            1620             S  folding clothes  42368589748491
193359            1620             S  folding clothes  42368639677781

                x         y         z
126166   3.417424 -2.164957 -4.849306
126167   5.423765 -6.936601 -4.954651
126168   4.700716 -3.127426 -7.648128
126169   8.033444 -4.700417 -5.024083
126170   3.970484 -1.645415 -3.621080
...           ...       ...       ...
193355   6.524798 -4.589834  0.573262
193356   9.100959 -4.168455 -0.602291
193357   7.590218 -5.736657 -1.854459
193358   8.681973 -6.284929 -5.180005
193359  10.161591 -3.840449 -3.152116

[67194 rows x 7 columns]
```

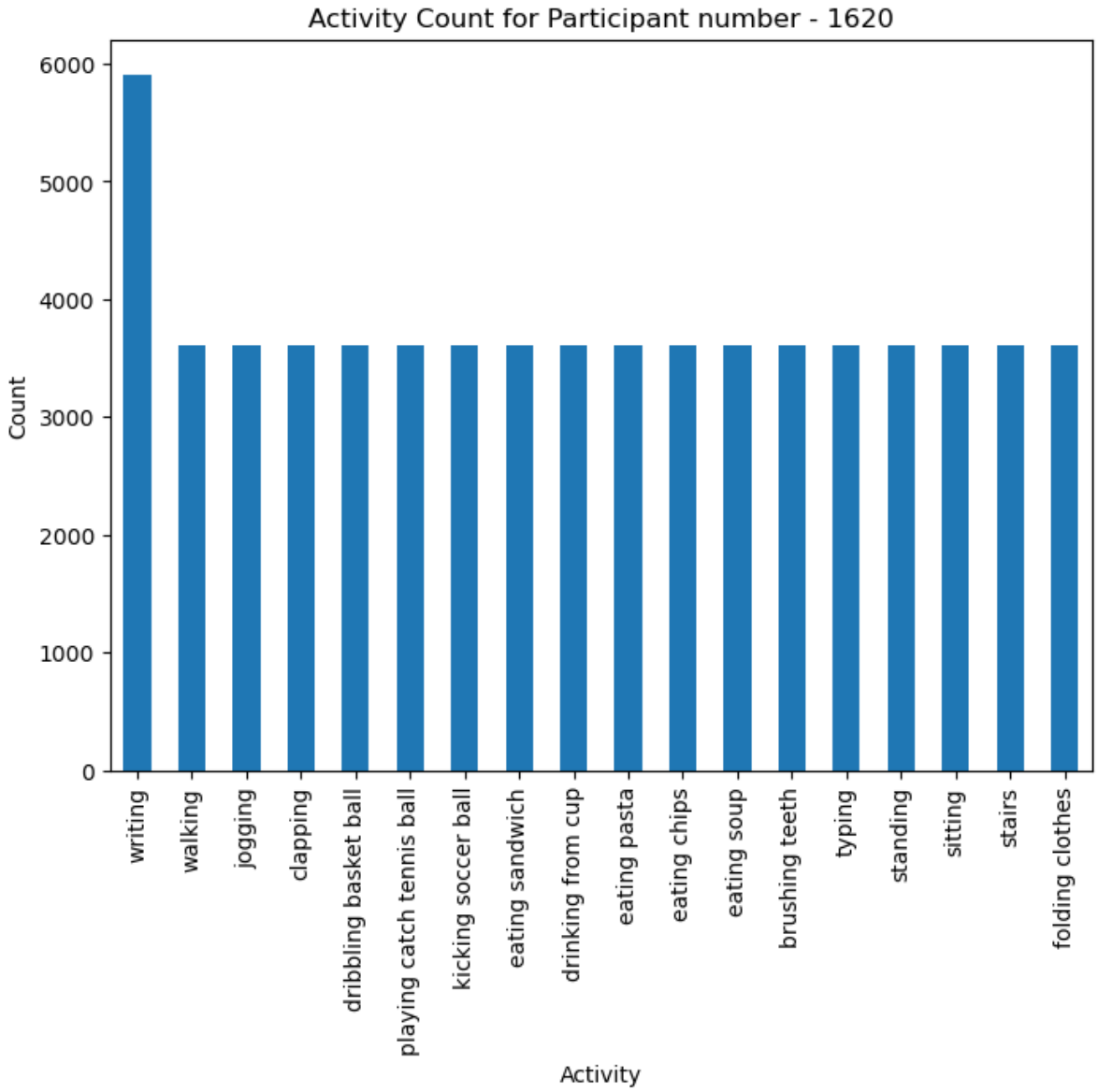
In [9]:
```
activity_counts = participant_data['activity'].value_counts()

# Plot the bar chart
plt.figure(figsize=(8, 6))
activity_counts.plot(kind='bar')
plt.title(f"Activity Count for Participant number - {1618}")
plt.xlabel('Activity')
plt.ylabel('Count')
plt.show()
```



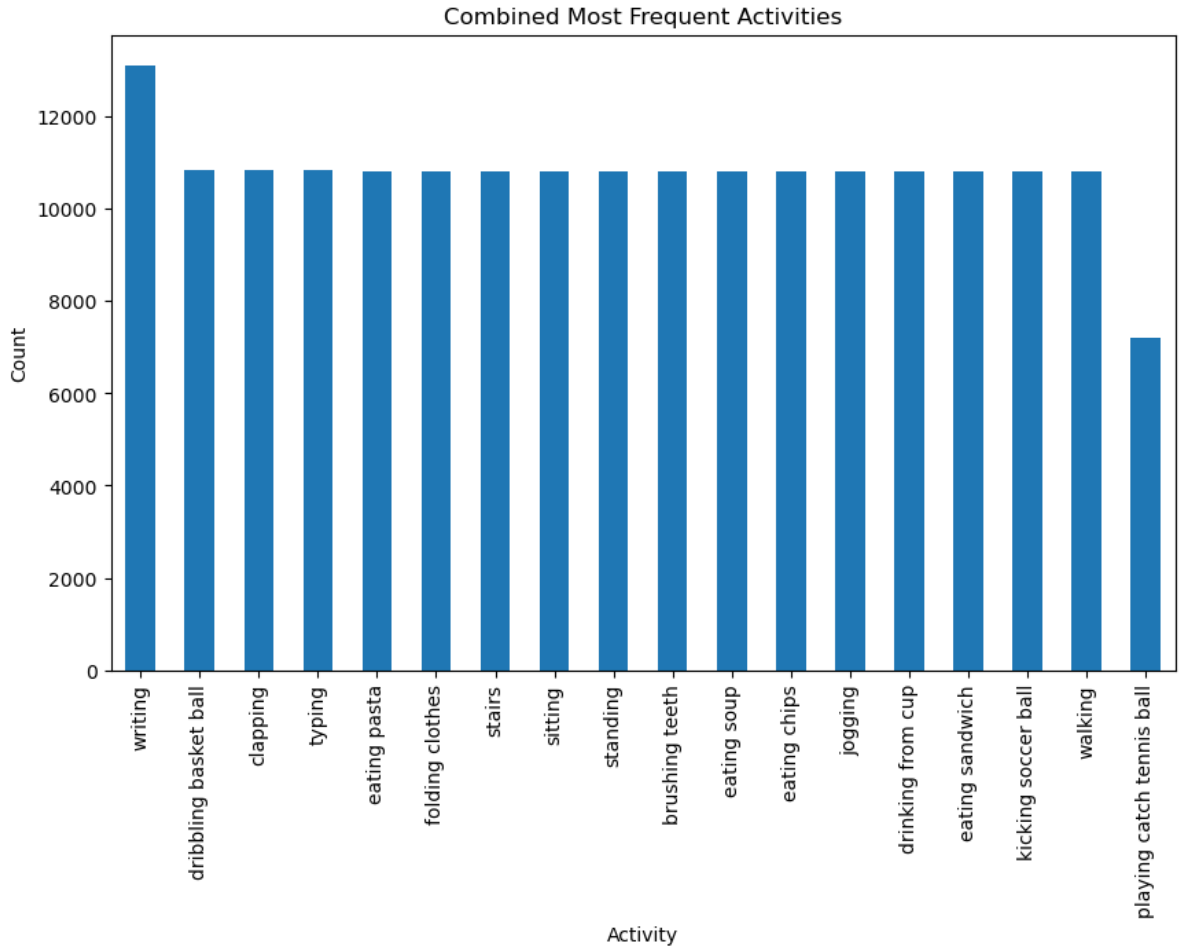
In [10]:
```
activity_counts = sec_participant_data['activity'].value_counts()

plt.figure(figsize=(8, 6))
activity_counts.plot(kind='bar')
plt.title(f"Activity Count for Participant number - {1619}")
plt.xlabel('Activity')
plt.ylabel('Count')
plt.show()
```



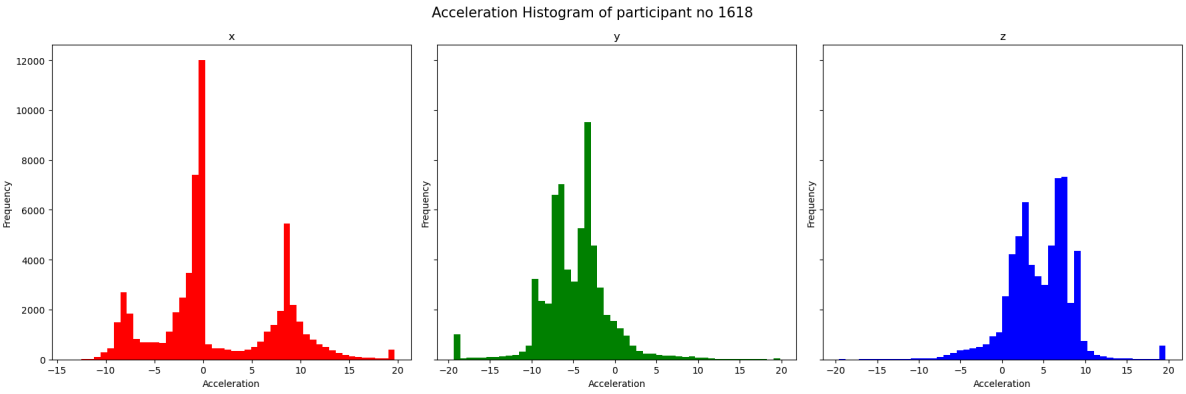
In [11]:
```
activity_counts = thi_participant_data['activity'].value_counts()

plt.figure(figsize=(8, 6))
activity_counts.plot(kind='bar')
plt.title(f"Activity Count for Participant number - {1620}")
plt.xlabel('Activity')
plt.ylabel('Count')
plt.show()
```



In [12]:
```
activity_counts = combined_df ['activity'].value_counts()

plt.figure(figsize=(10, 6))
activity_counts.plot(kind='bar')
plt.xlabel('Activity')
plt.ylabel('Count')
plt.title('Combined Most Frequent Activities')
plt.show()
```

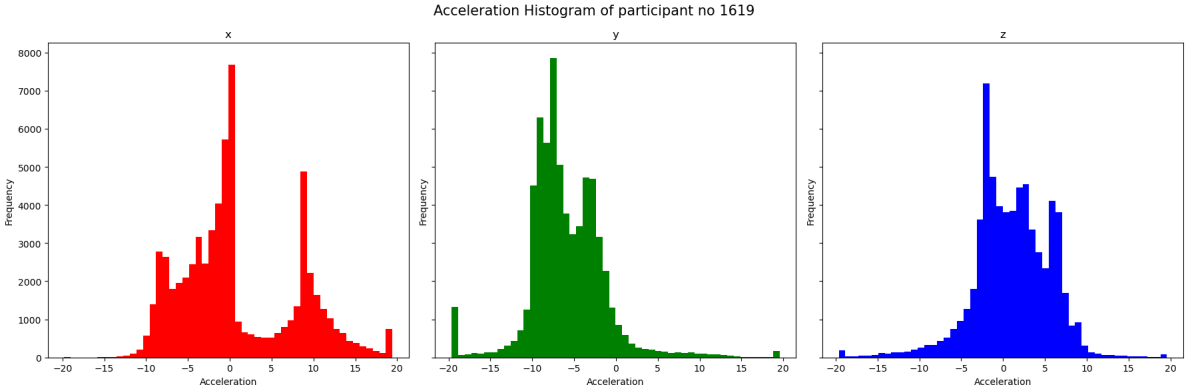


In [13]:
```
colors = ['red', 'green', 'blue']

# Create subplots with 3 histograms
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6),sharey=True)

# Iterate over the columns and plot histograms with different colors
for i, col in enumerate(['x', 'y', 'z']):
    axes[i].hist(participant_data[col].dropna().values, bins=50, color=colors[i])
    axes[i].set_title(col)
    axes[i].set_xlabel('Acceleration')
    axes[i].set_ylabel('Frequency')
plt.suptitle('Acceleration Histogram of participant no 1618', fontsize=15)
plt.tight_layout()
plt.show()
```

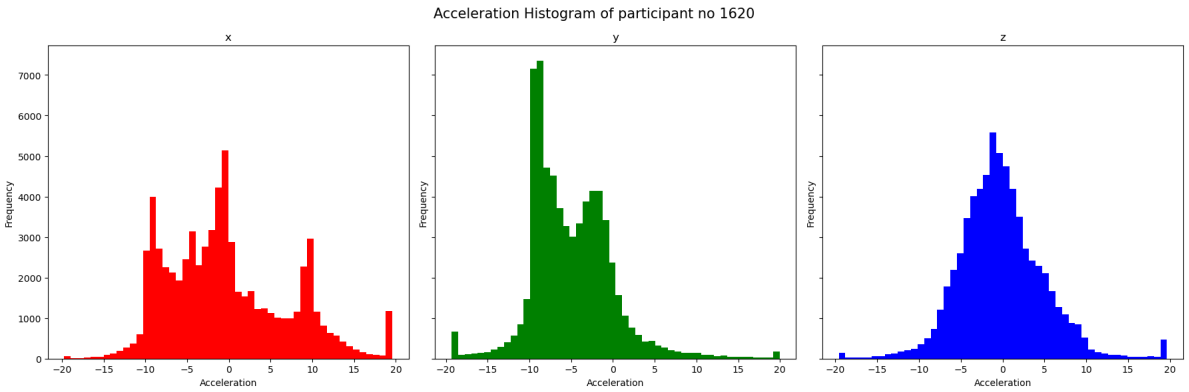


In [14]:
```
colors = ['red', 'green', 'blue']

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6),sharey=True)

for i, col in enumerate(['x', 'y', 'z']):
```

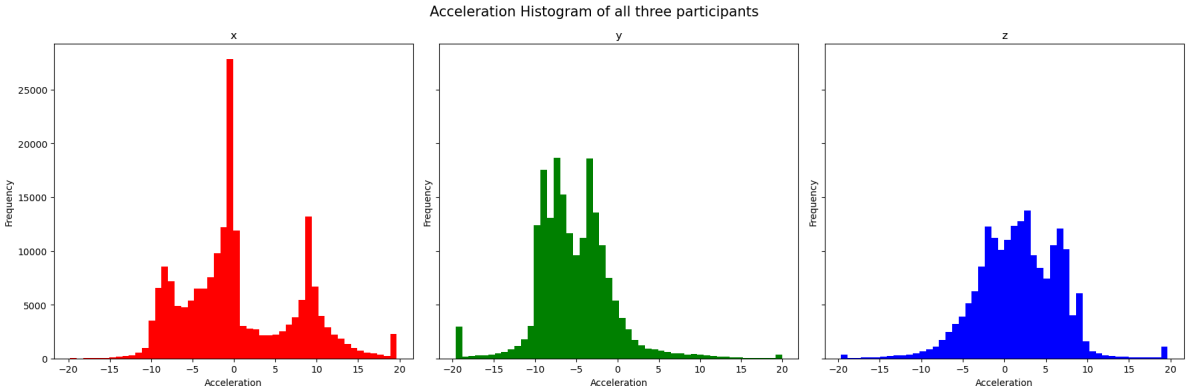
```
    axes[i].hist(sec_participant_data[col].dropna().values, bins=50, color=colors[
    axes[i].set_title(col)
    axes[i].set_xlabel('Acceleration')
    axes[i].set_ylabel('Frequency')

plt.suptitle('Acceleration Histogram of participant no 1619', fontsize=15)
plt.tight_layout()
plt.show()
```



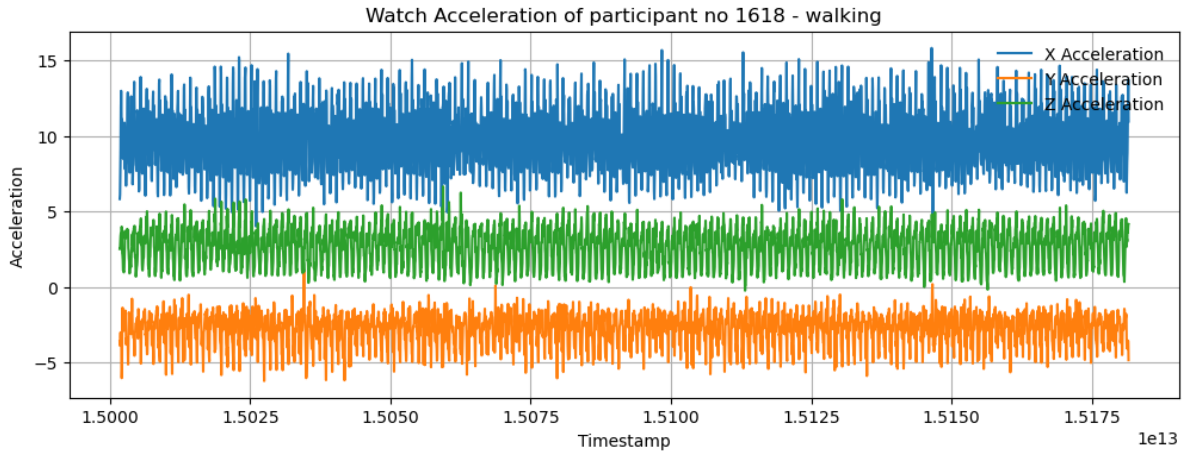
In [15]:
```
colors = ['red', 'green', 'blue']

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6),sharey=True)

for i, col in enumerate(['x', 'y', 'z']):
    axes[i].hist(thi_participant_data[col].dropna().values, bins=50, color=colors[
    axes[i].set_title(col)
    axes[i].set_xlabel('Acceleration')
    axes[i].set_ylabel('Frequency')

plt.suptitle('Acceleration Histogram of participant no 1620', fontsize=15)
plt.tight_layout()
plt.show()
```

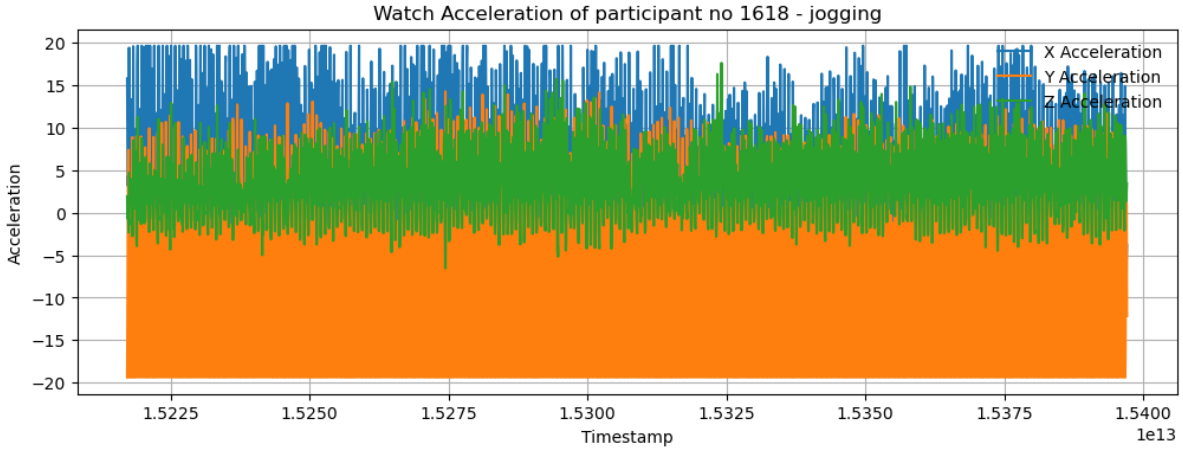


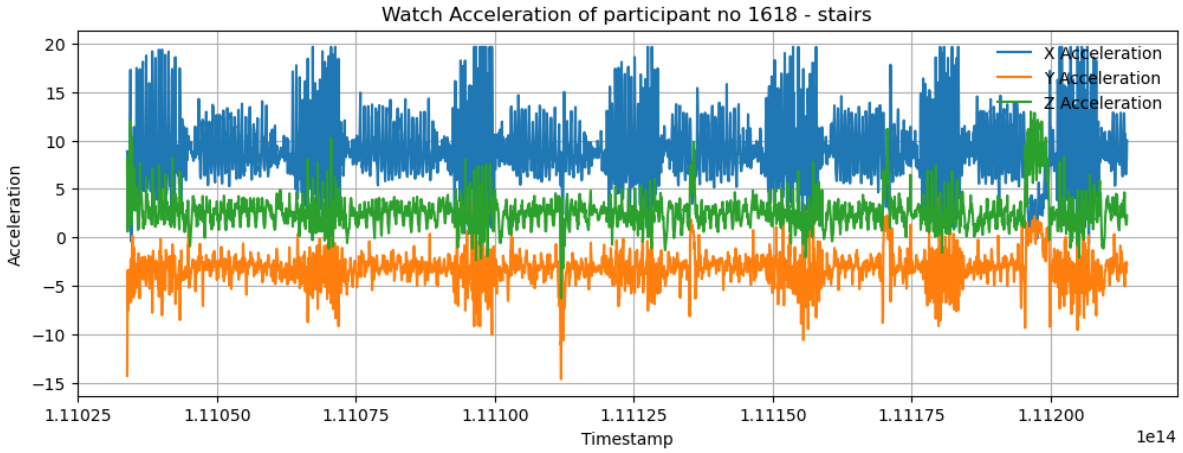
In [16]:
```
colors = ['red', 'green', 'blue']

fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(18, 6),sharey=True)

for i, col in enumerate(['x', 'y', 'z']):
    axes[i].hist(combined_df[col].dropna().values, bins=50, color=colors[i])
    axes[i].set_title(col)
    axes[i].set_xlabel('Acceleration')
    axes[i].set_ylabel('Frequency')

plt.suptitle('Acceleration Histogram of all three participants', fontsize=15)
plt.tight_layout()
plt.show()
```

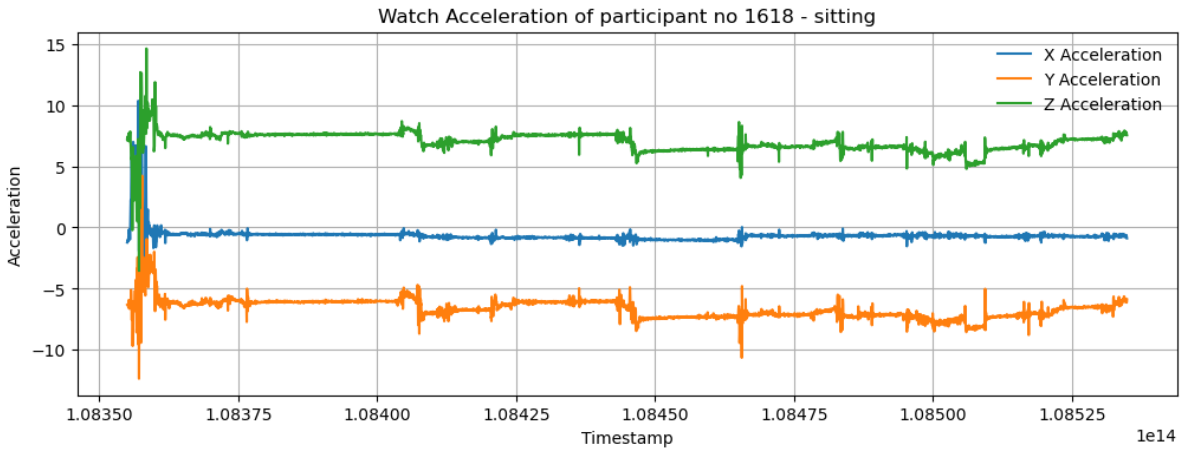


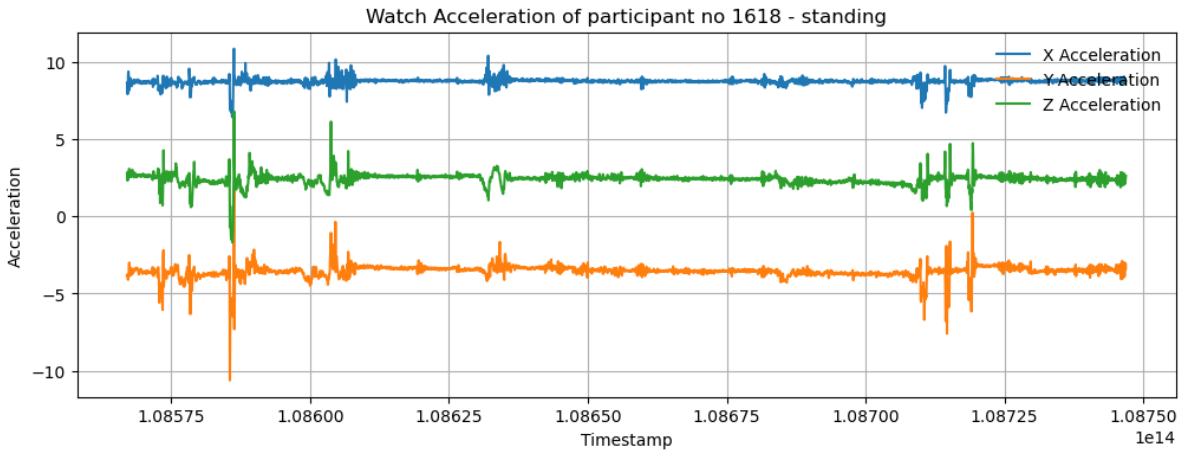
In [17]:
```
def show_accel_per_activity(device, data, activity, bins):
    # Select data for the specified activity
    activity_data = data[data['activity'] == activity]

    # Plot line graphs for X, Y, and Z acceleration
    plt.figure(figsize=(12,4))
    plt.plot(activity_data['timestamp'], activity_data['x'], label='X Acceleration
    plt.plot(activity_data['timestamp'], activity_data['y'], label='Y Acceleration
    plt.plot(activity_data['timestamp'], activity_data['z'], label='Z Acceleration

    # Set plot title and labels
    plt.title(f'{device} Acceleration of participant no 1618 - {activity}')
    plt.xlabel('Timestamp')
    plt.ylabel('Acceleration')

    # Customize the legend
    plt.legend(loc='upper right', frameon=False)

    # Display gridlines
    plt.grid(True)
    plt.show()

# Iterate over activity codes and plot acceleration data for each activity
for key in activity_codes_mapping:
    show_accel_per_activity('Watch', participant_data, activity_codes_mapping[key]
```

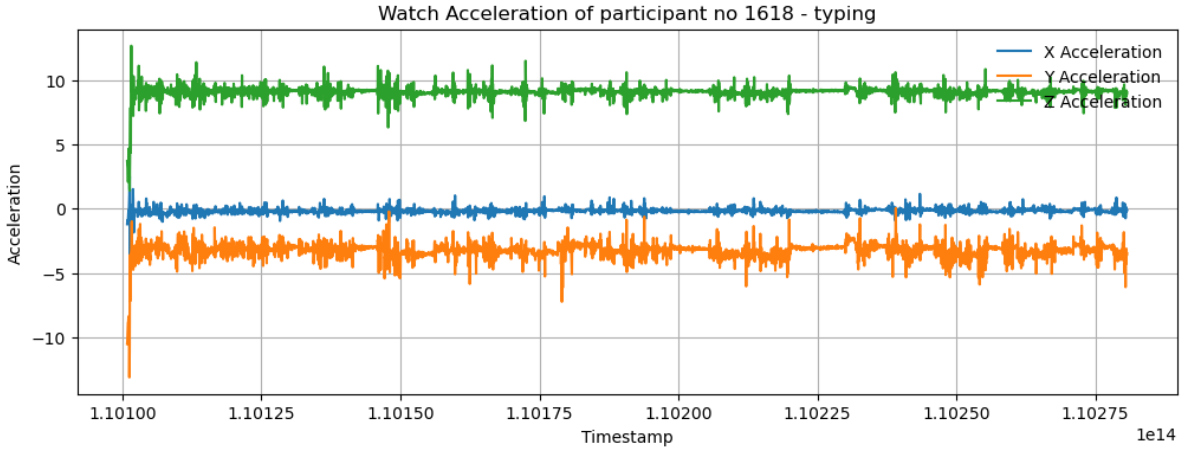


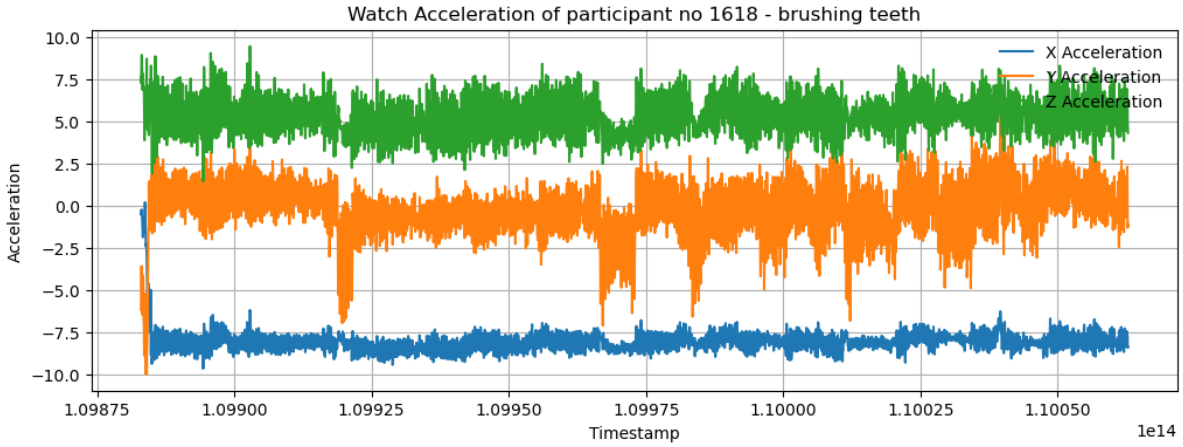

Watch Acceleration of participant no 1618 - jogging
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

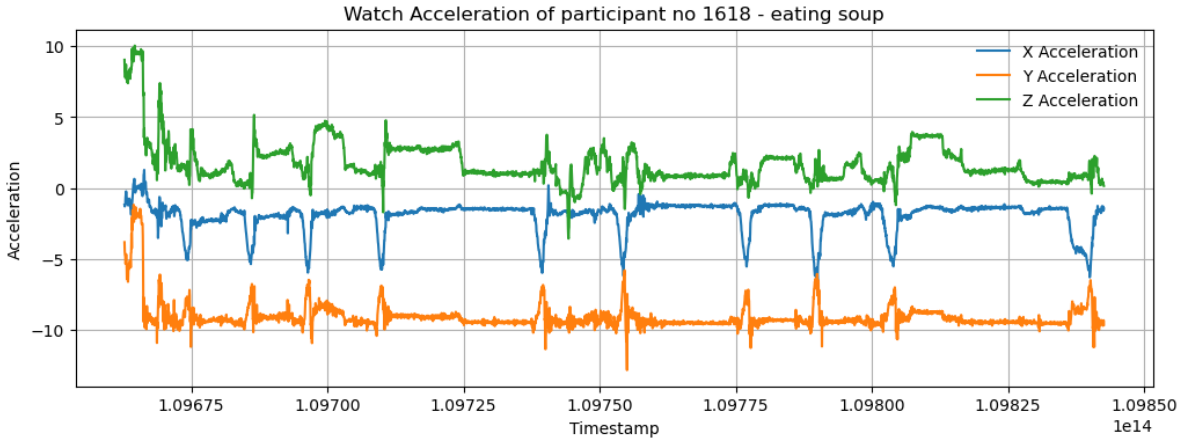

Watch Acceleration of participant no 1618 - stairs
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

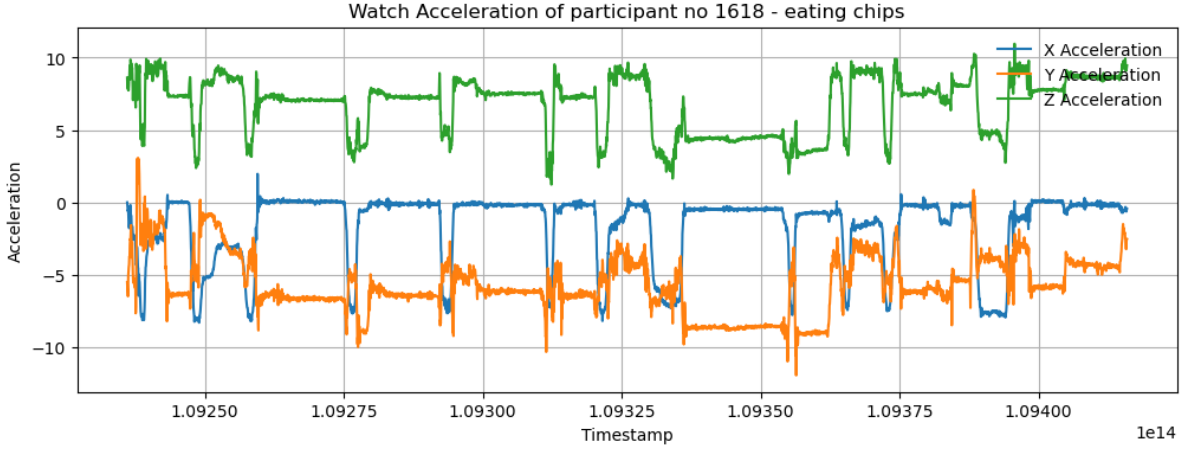

Watch Acceleration of participant no 1618 - sitting
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

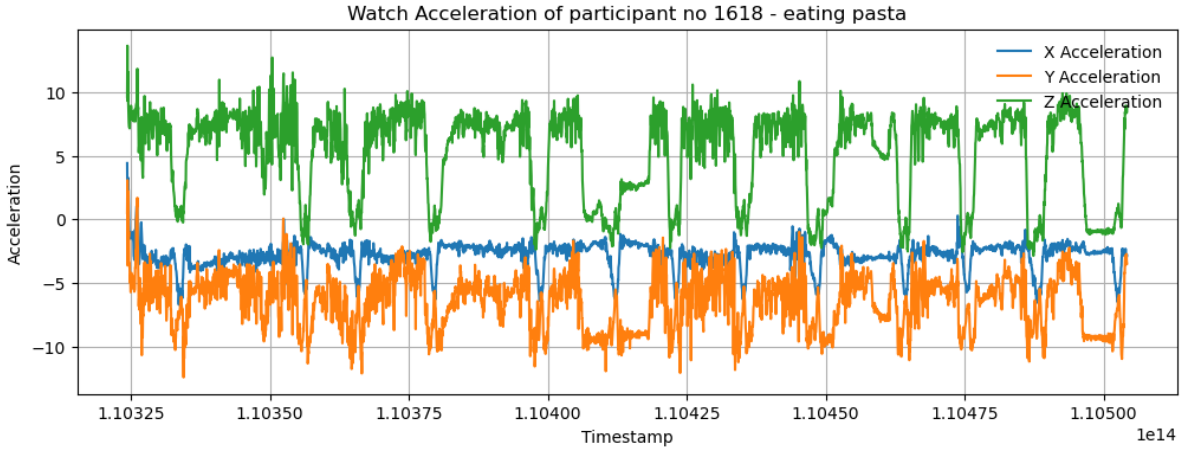

Watch Acceleration of participant no 1618 - standing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

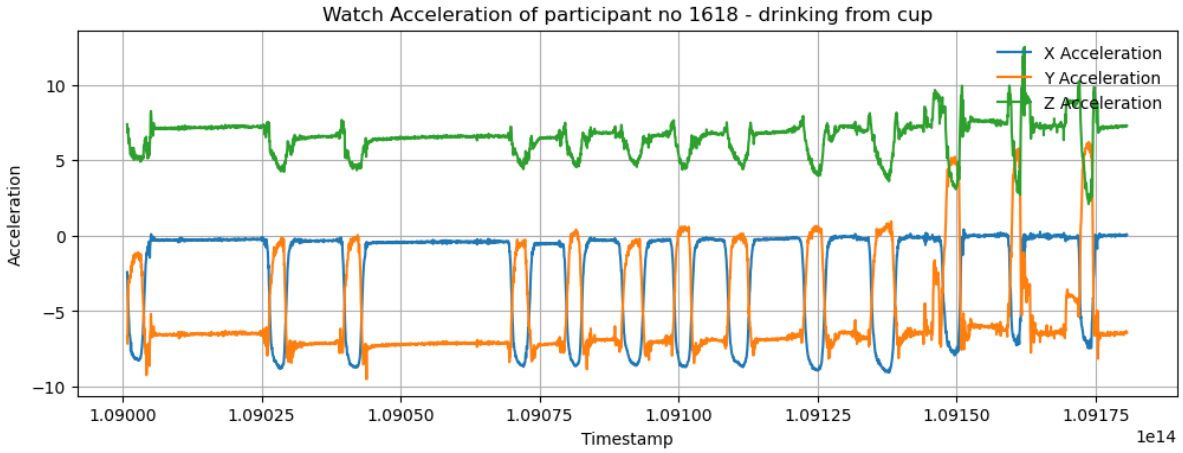

Watch Acceleration of participant no 1618 - typing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

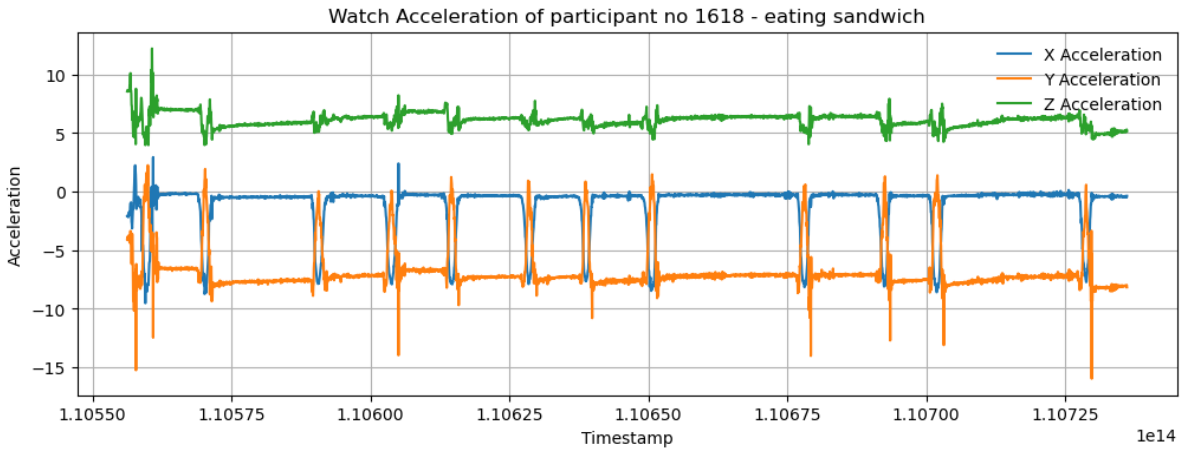

Watch Acceleration of participant no 1618 - brushing teeth
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

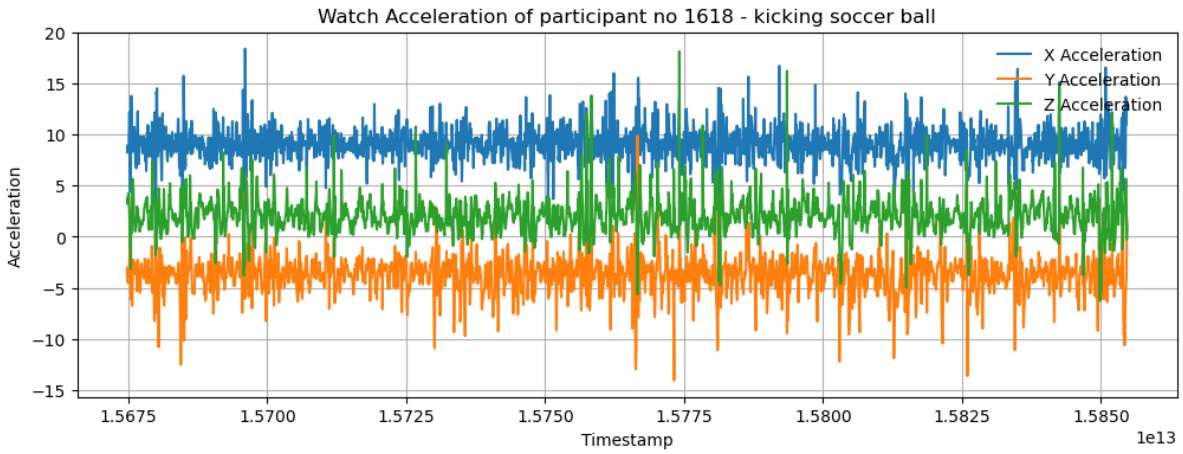

Watch Acceleration of participant no 1618 - eating soup
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

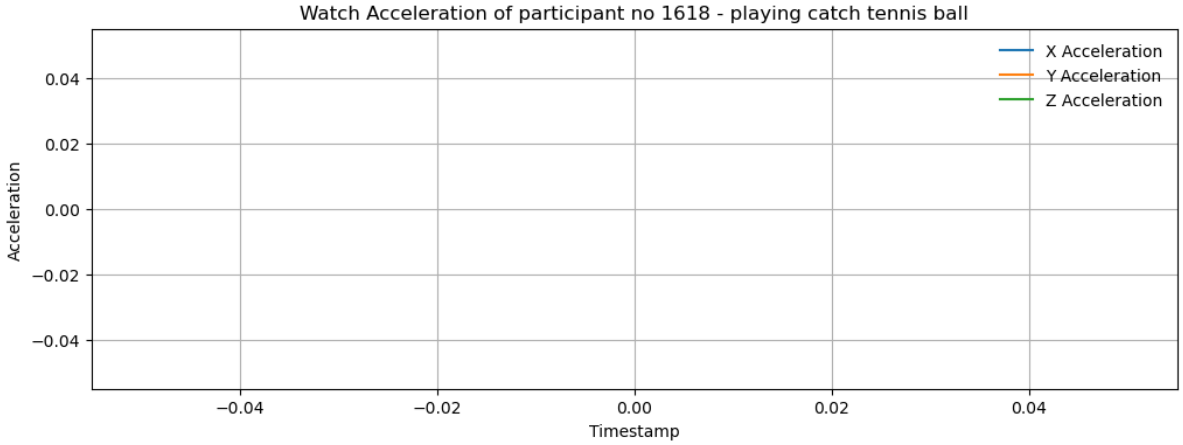

Watch Acceleration of participant no 1618 - eating chips
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

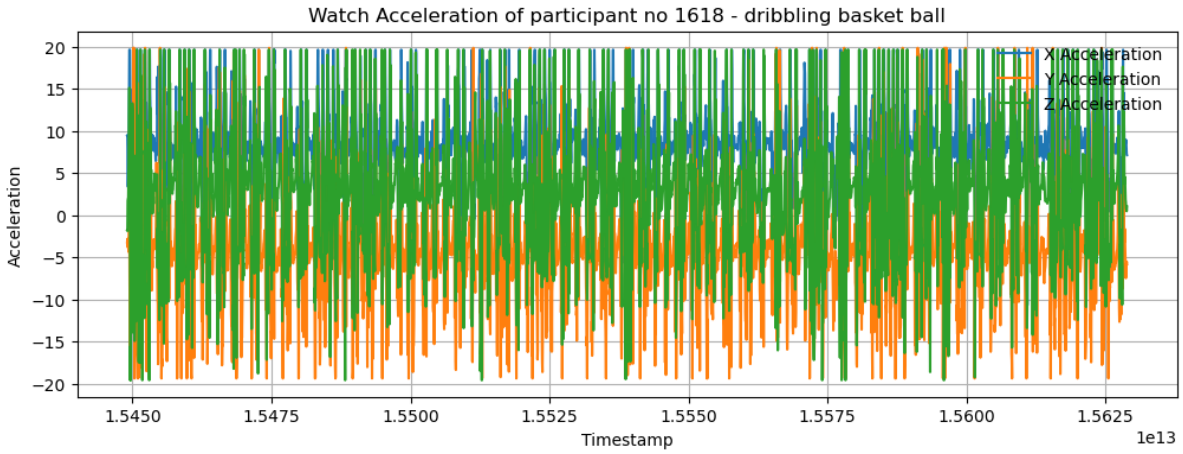

Watch Acceleration of participant no 1618 - eating pasta
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

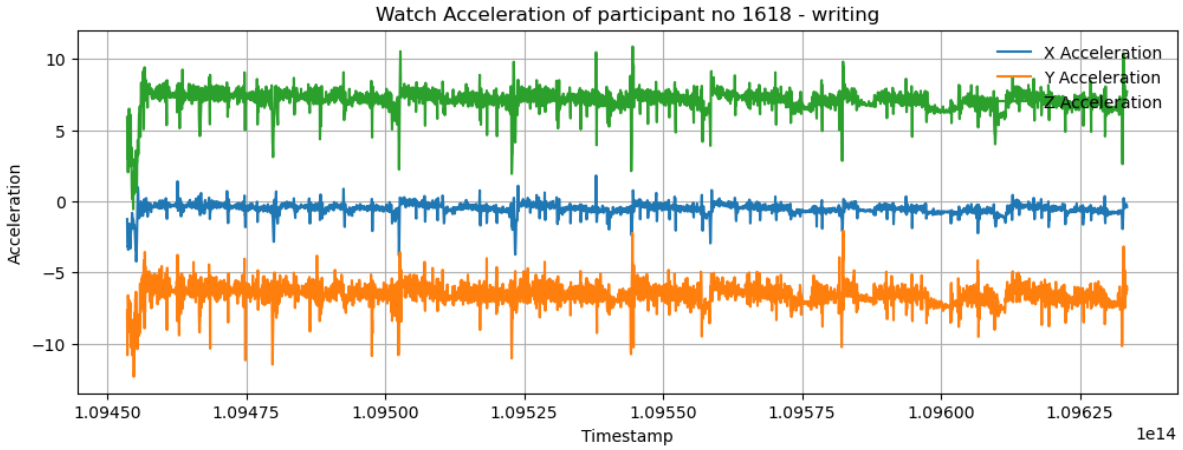

Watch Acceleration of participant no 1618 - drinking from cup
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

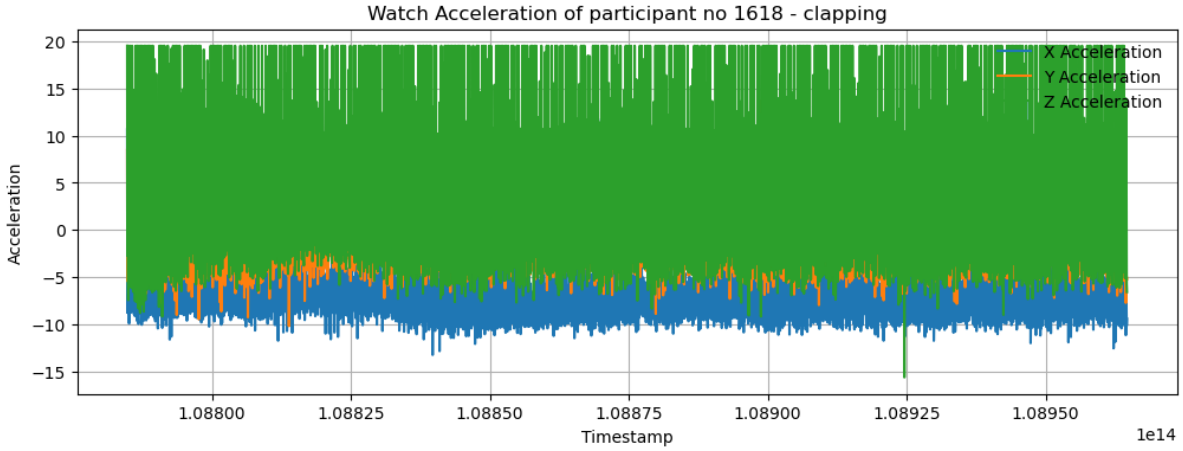

Watch Acceleration of participant no 1618 - eating sandwich
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

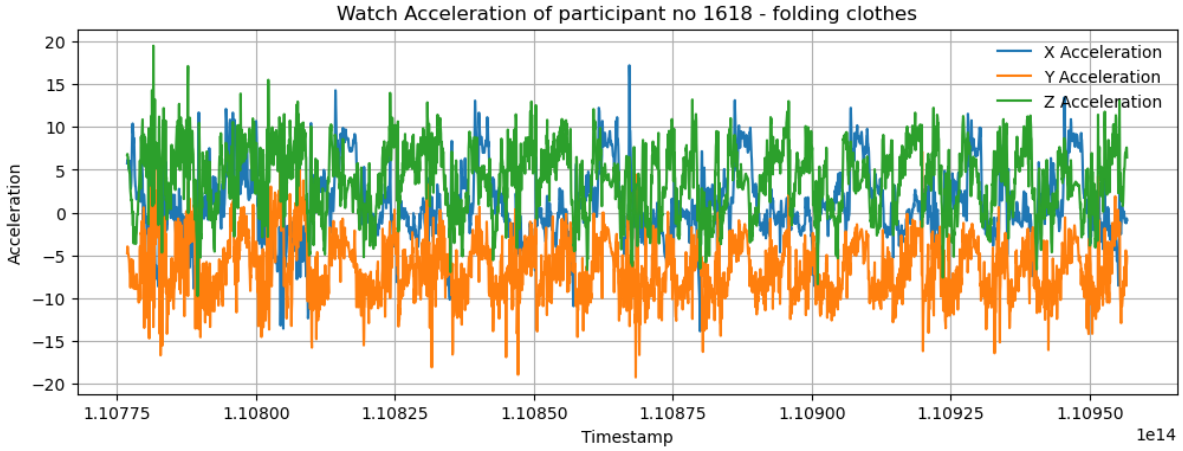

Watch Acceleration of participant no 1618 - kicking soccer ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13


Watch Acceleration of participant no 1618 - playing catch tennis ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp

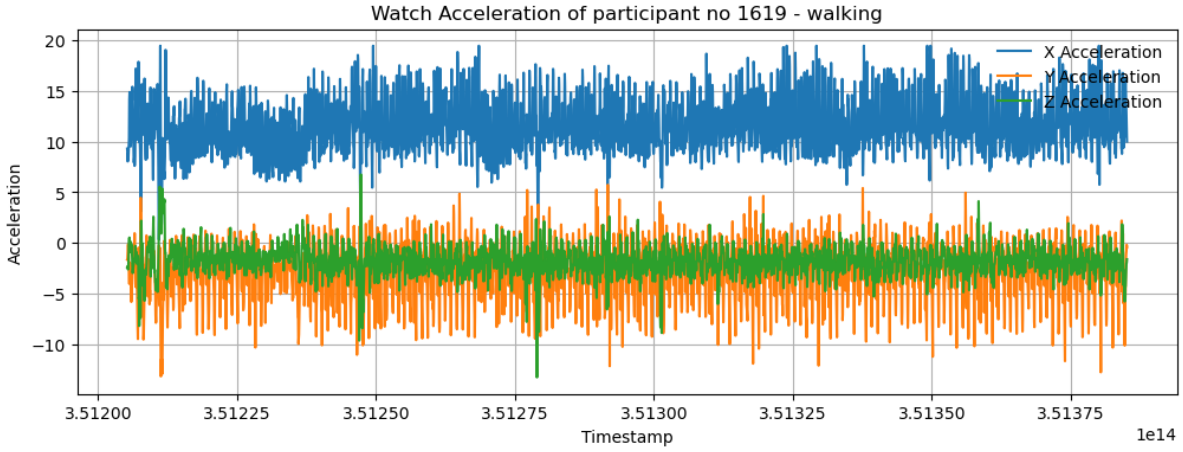

Watch Acceleration of participant no 1618 - dribbling basket ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

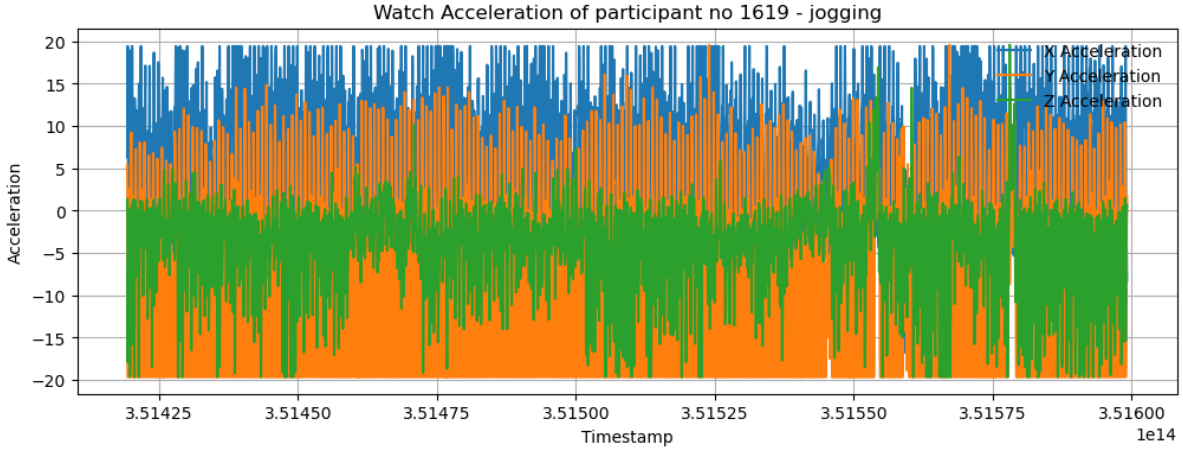

Watch Acceleration of participant no 1618 - writing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

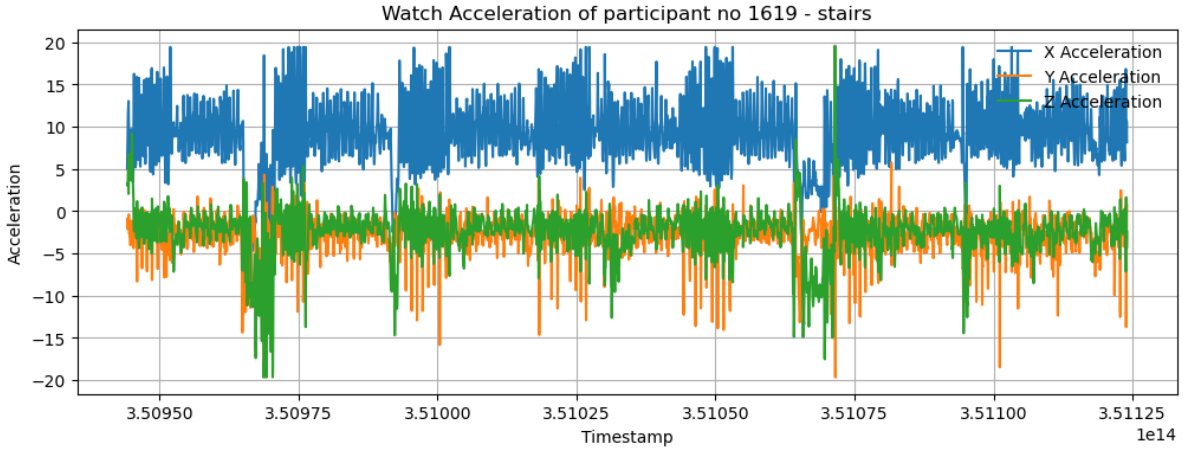

Watch Acceleration of participant no 1618 - clapping
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

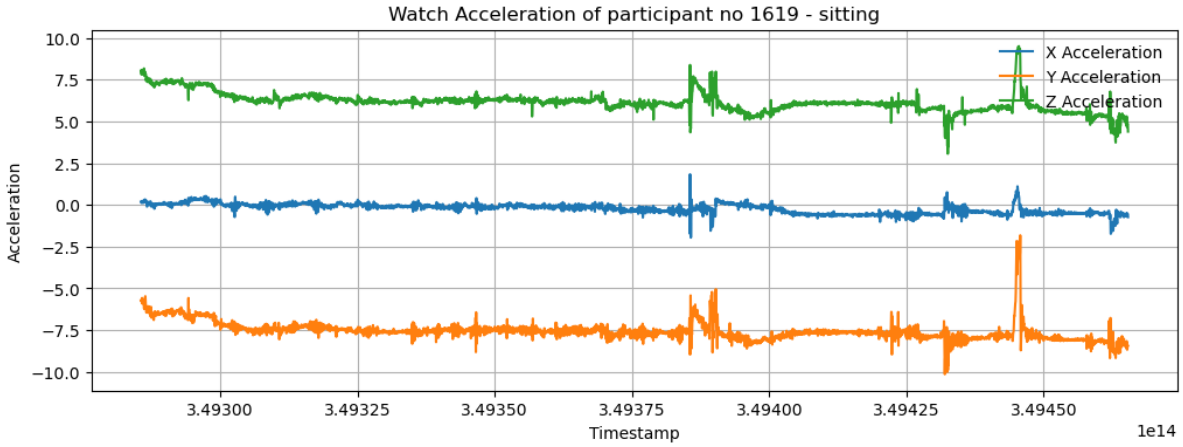


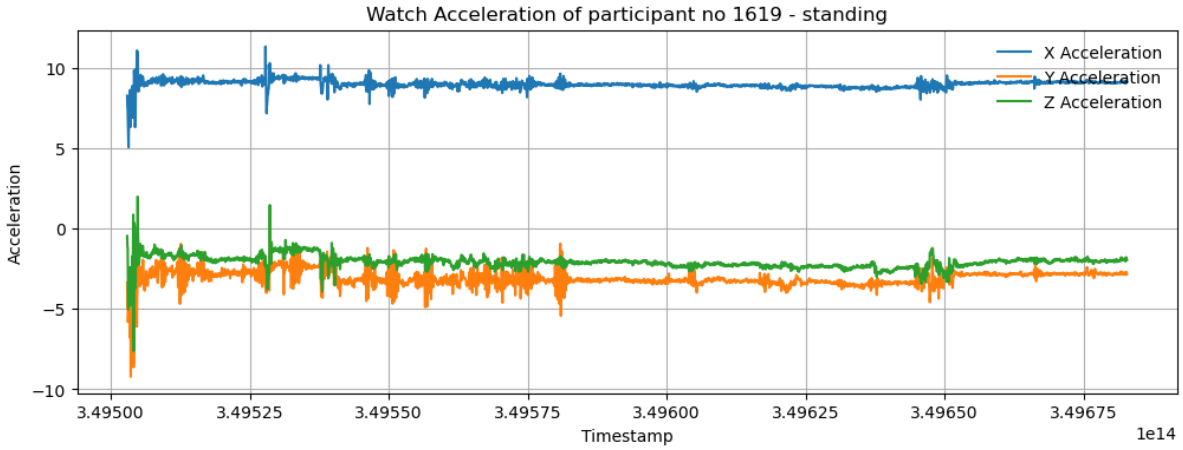
In [18]:
```
def show_accel_per_activity(device, data, activity, bins):

    activity_data = data[data['activity'] == activity]

    plt.figure(figsize=(12,4))
    plt.plot(activity_data['timestamp'], activity_data['x'], label='X Acceleration
    plt.plot(activity_data['timestamp'], activity_data['y'], label='Y Acceleration
    plt.plot(activity_data['timestamp'], activity_data['z'], label='Z Acceleration

    plt.title(f'{device} Acceleration of participant no 1619 - {activity}')
    plt.xlabel('Timestamp')
    plt.ylabel('Acceleration')

    plt.legend(loc='upper right', frameon=False)
    plt.grid(True)
    plt.show()

for key in activity_codes_mapping:
    show_accel_per_activity('Watch', sec_participant_data, activity_codes_mapping[
```

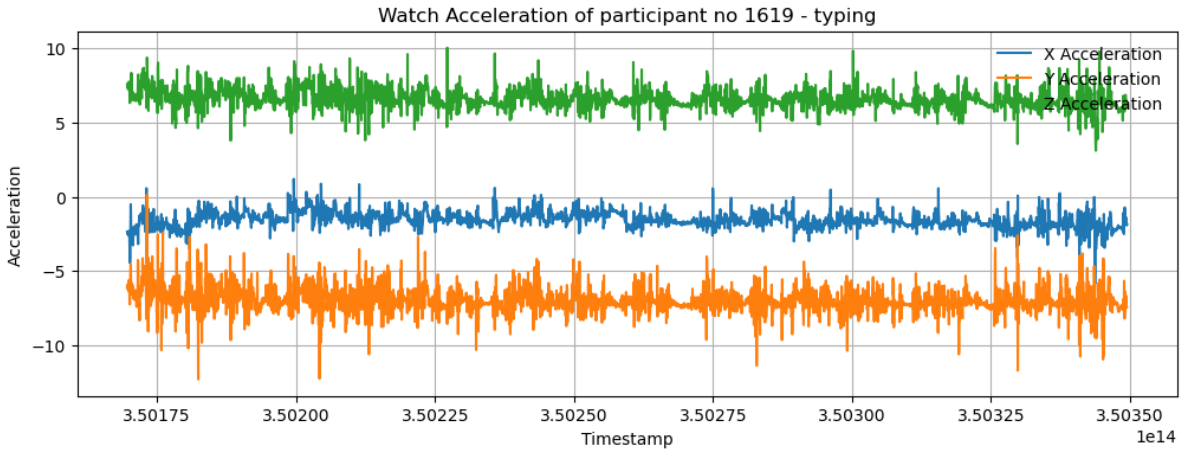


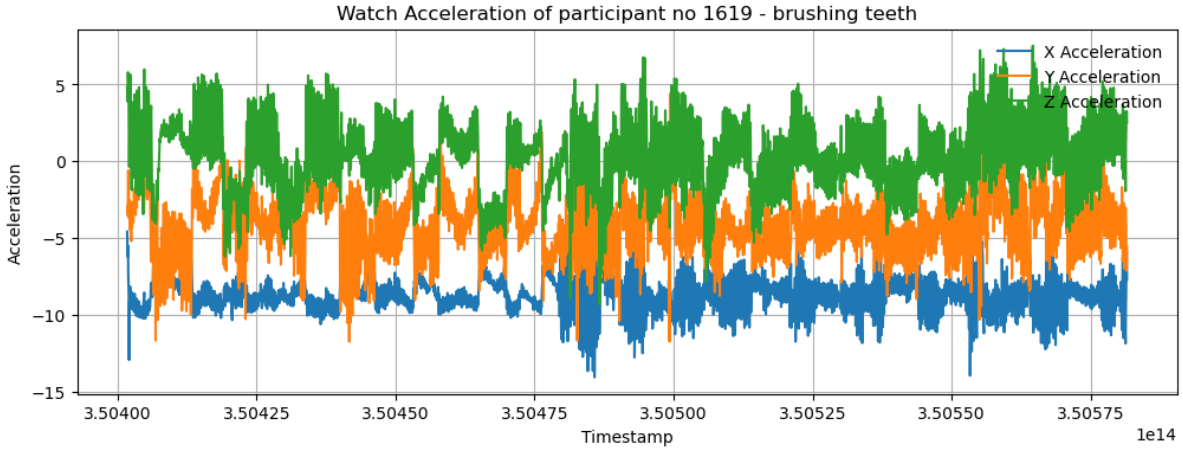


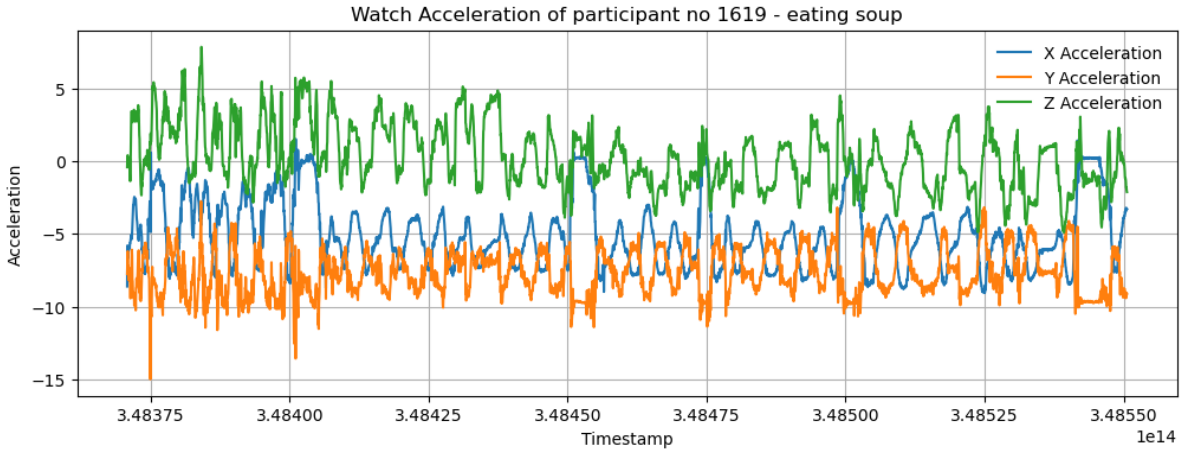

Watch Acceleration of participant no 1619 - stairs
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

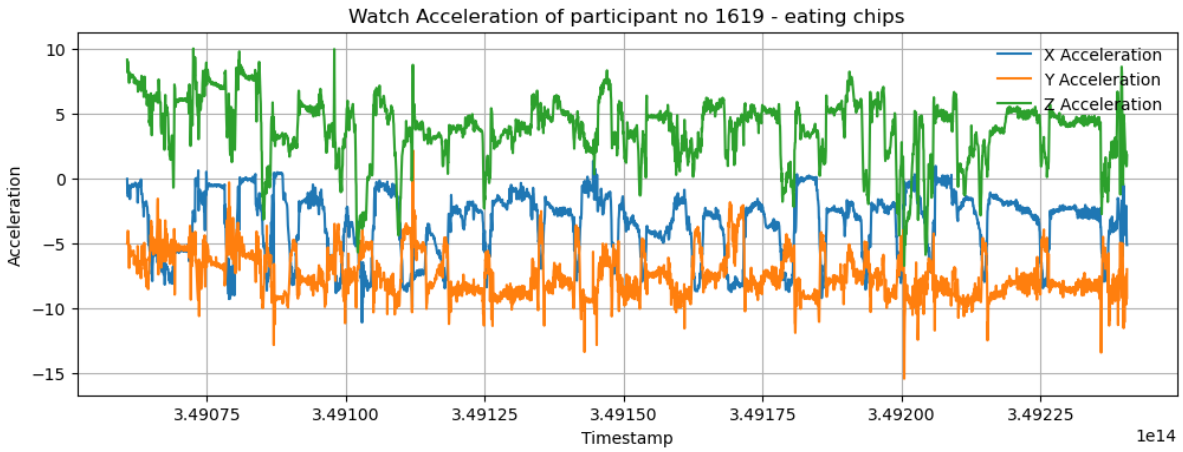

Watch Acceleration of participant no 1619 - sitting
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

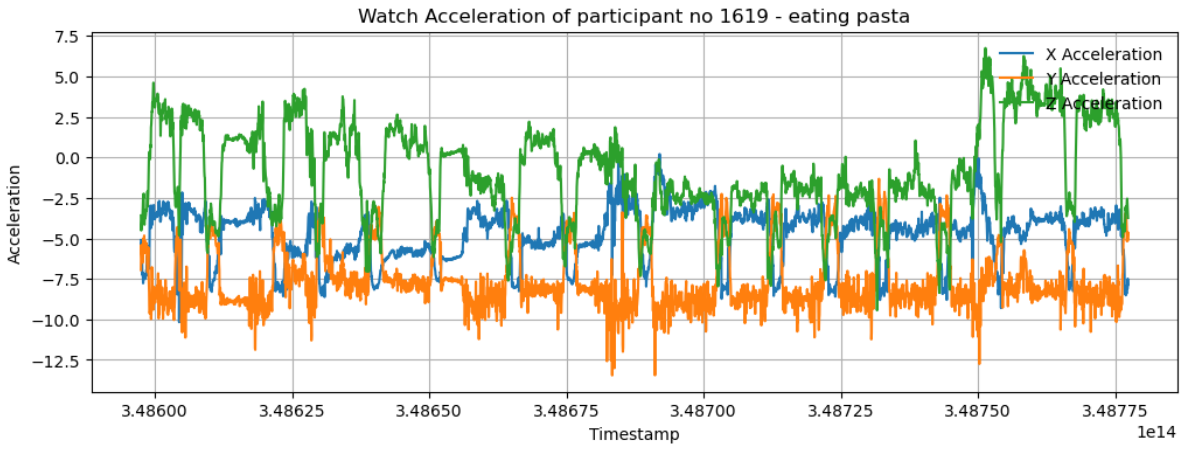

Watch Acceleration of participant no 1619 - standing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

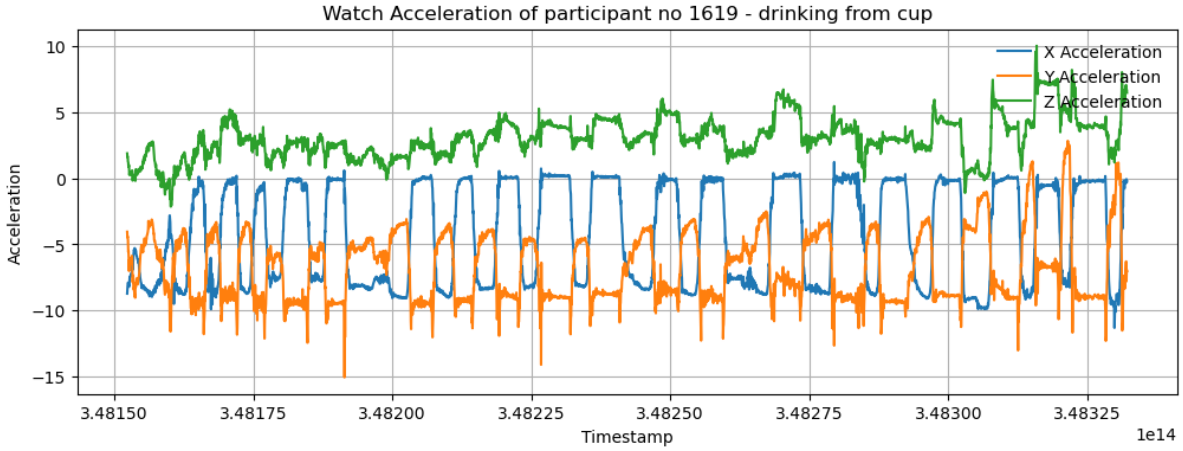

Watch Acceleration of participant no 1619 - typing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

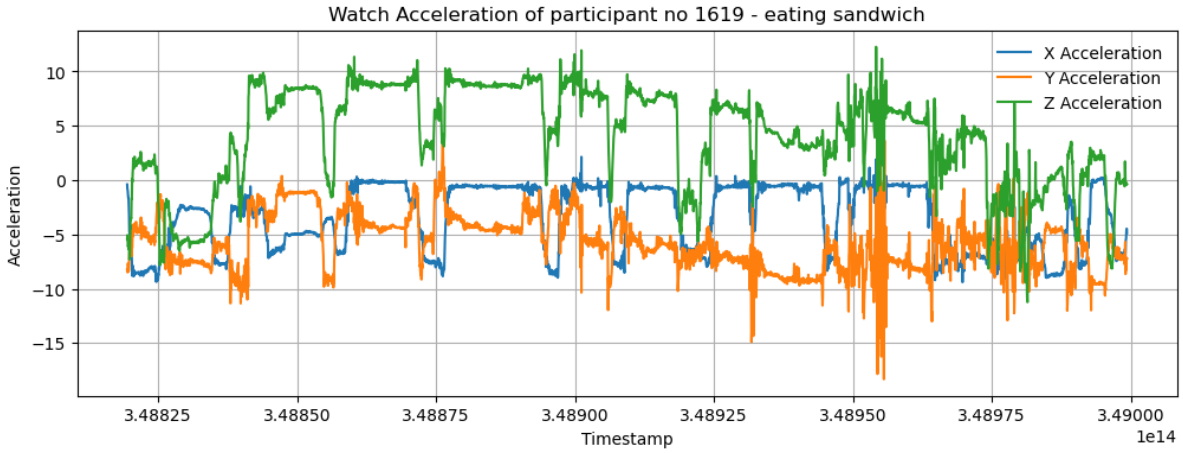

Watch Acceleration of participant no 1619 - brushing teeth
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

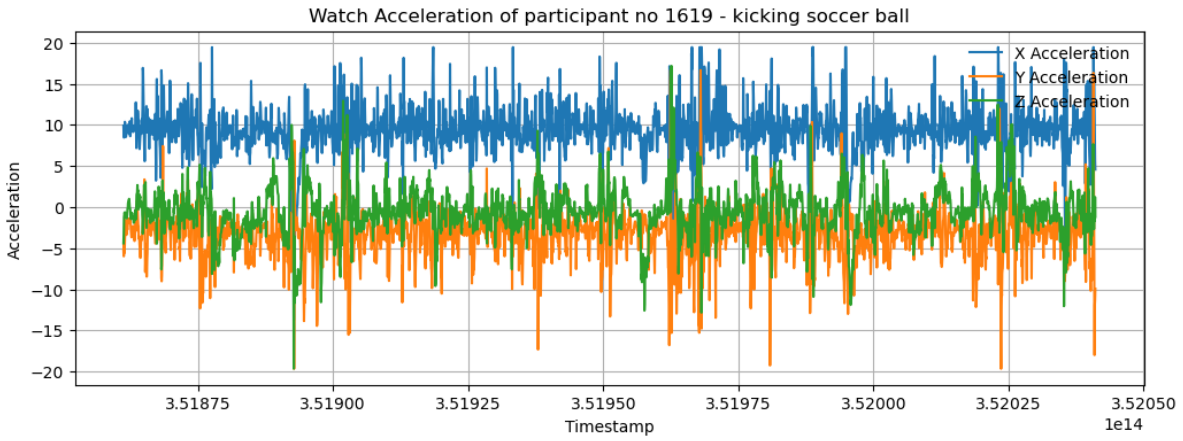

Watch Acceleration of participant no 1619 - eating soup
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

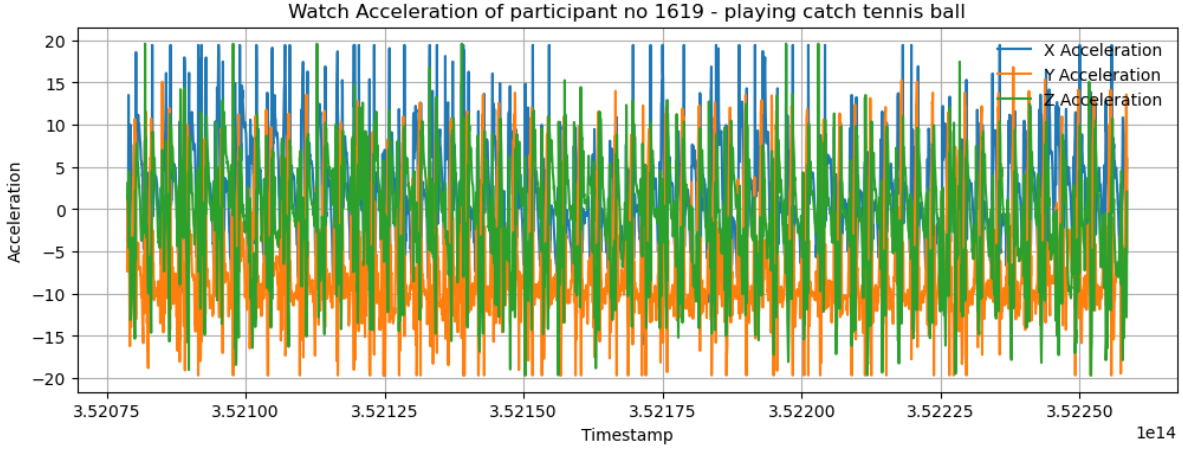

Watch Acceleration of participant no 1619 - eating chips
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

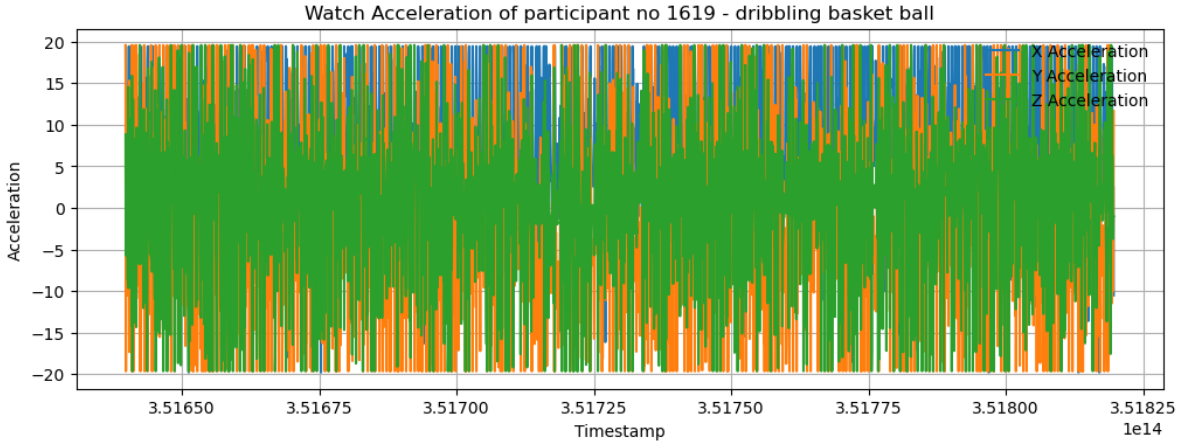

Watch Acceleration of participant no 1619 - eating pasta
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

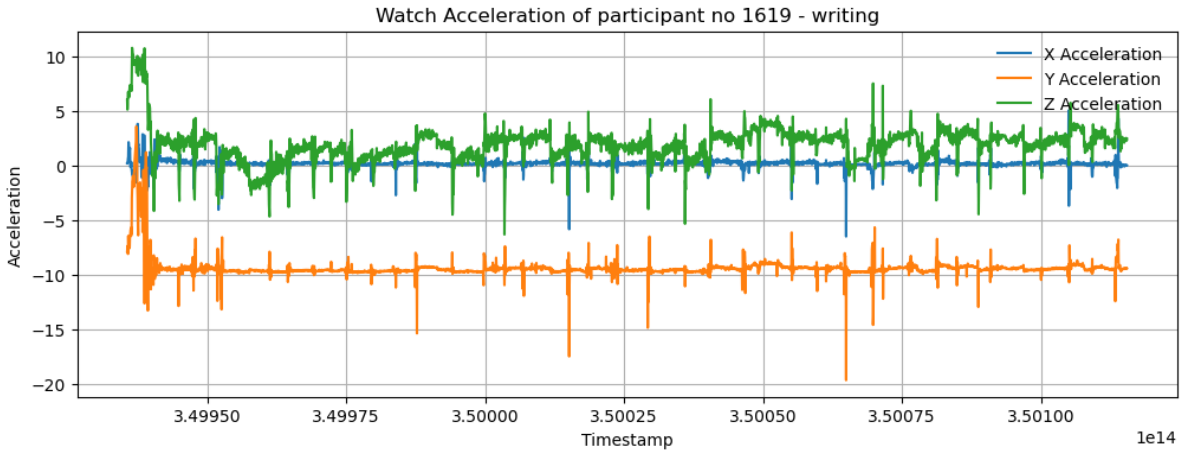

Watch Acceleration of participant no 1619 - drinking from cup
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

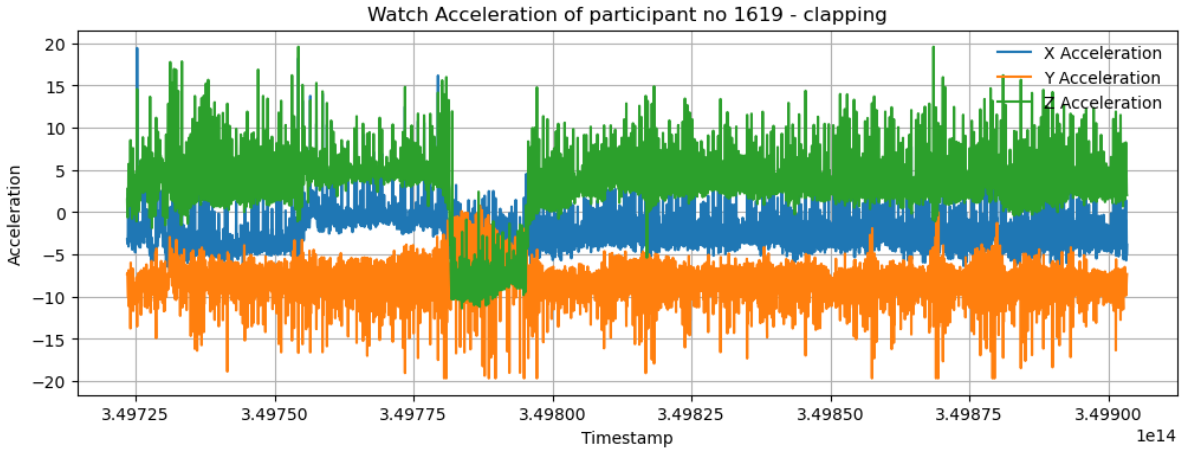

Watch Acceleration of participant no 1619 - eating sandwich
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14

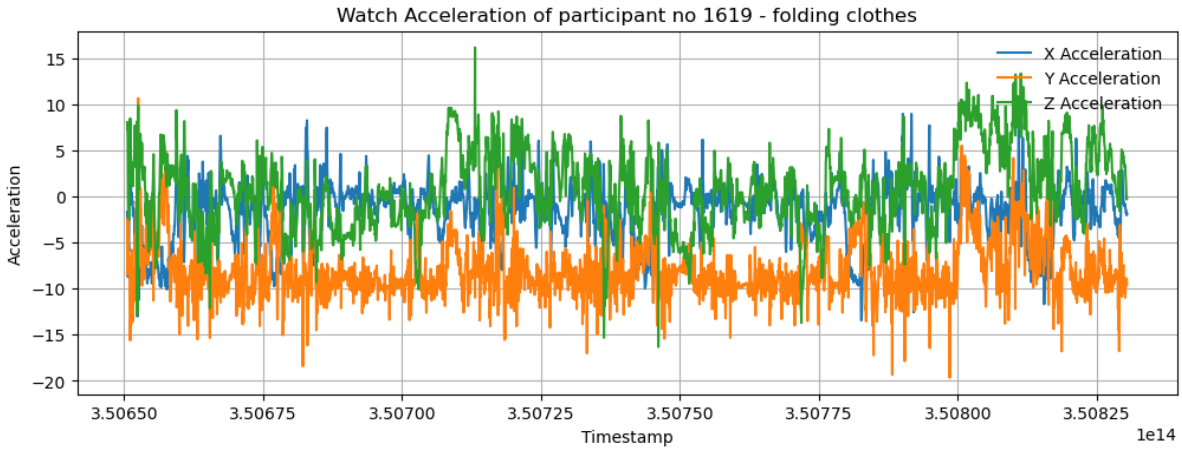

Watch Acceleration of participant no 1619 - kicking soccer ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14


Watch Acceleration of participant no 1619 - playing catch tennis ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e14



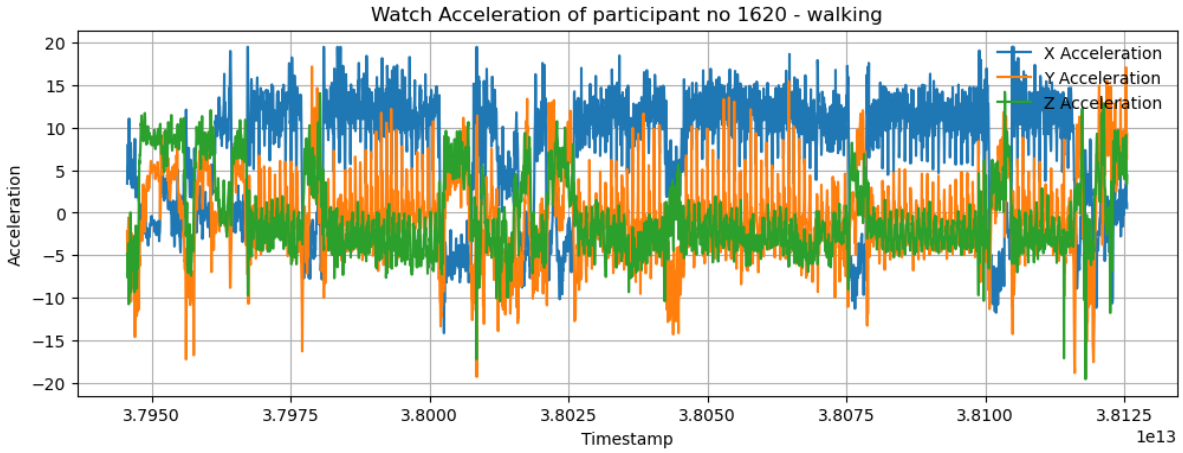


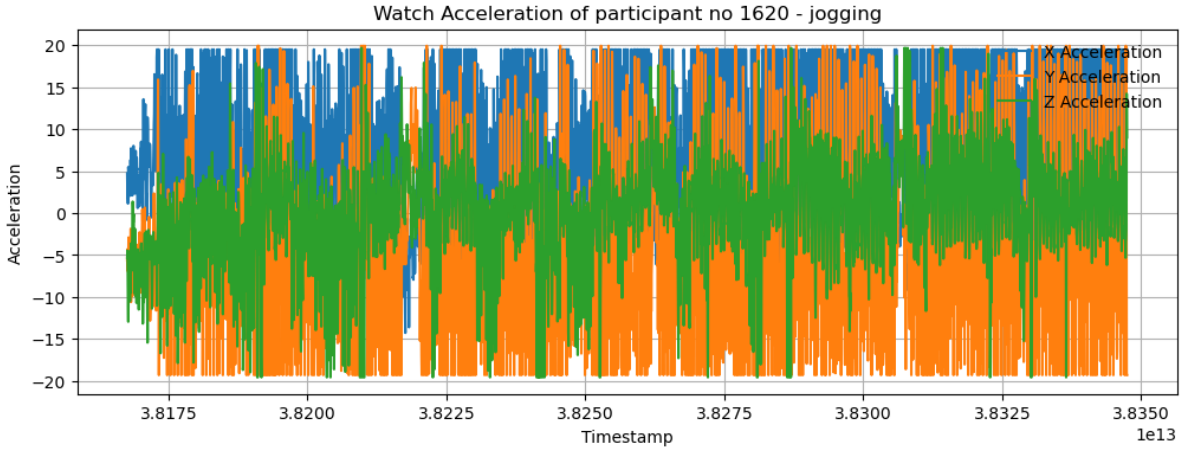


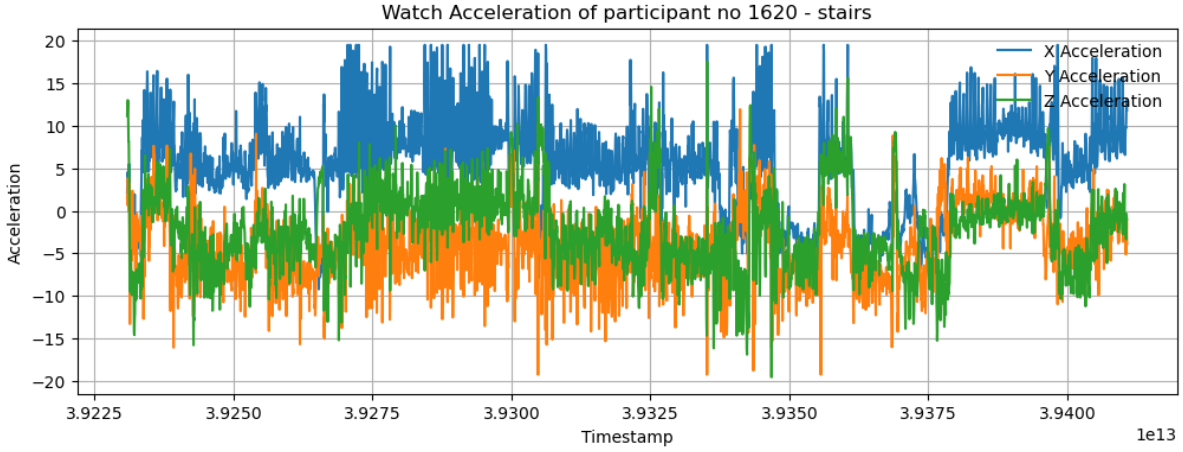


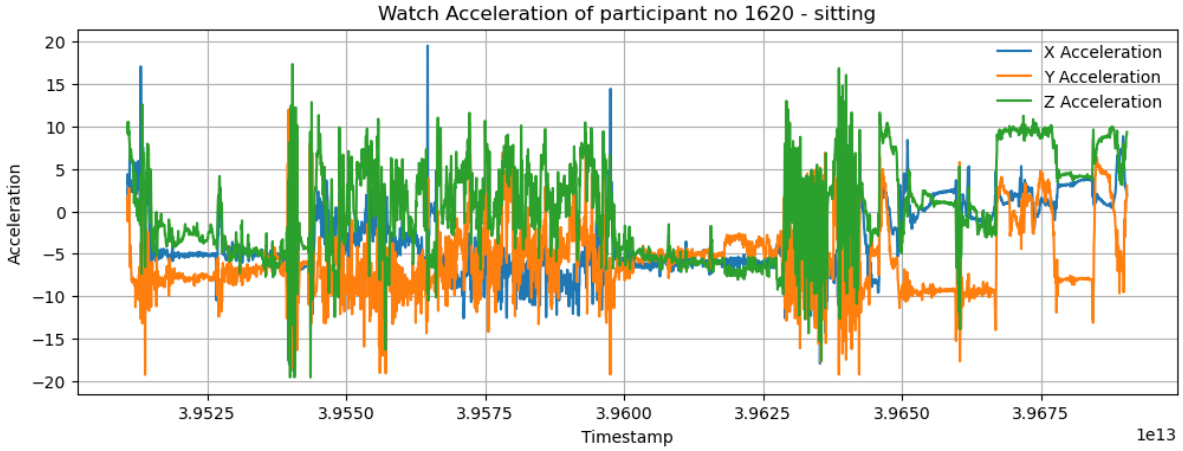
In [19]:
```
import matplotlib.pyplot as plt

def show_accel_per_activity(device, data, activity, bins):

    activity_data = data[data['activity'] == activity]
```

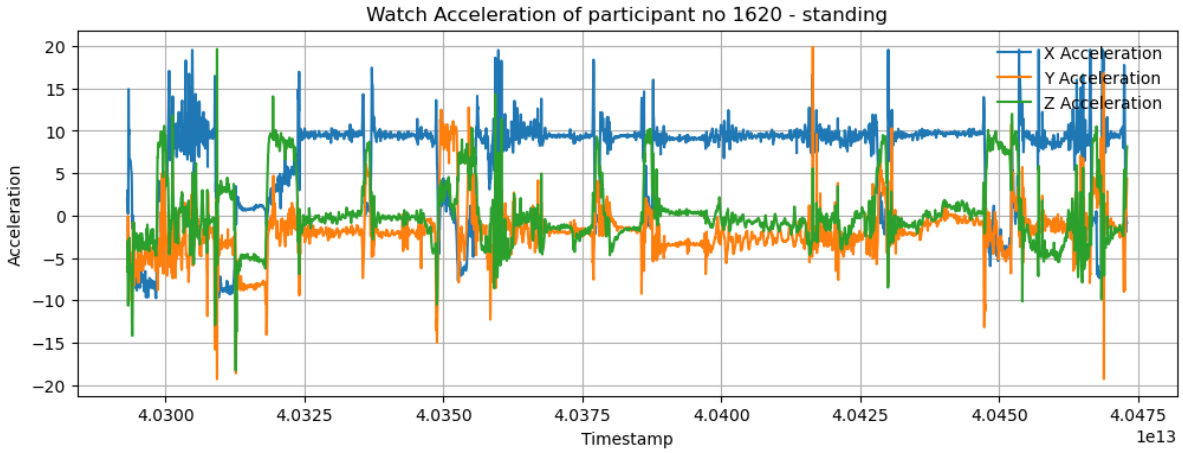
```
    plt.figure(figsize=(12,4))
    plt.plot(activity_data['timestamp'], activity_data['x'], label='X Acceleration
    plt.plot(activity_data['timestamp'], activity_data['y'], label='Y Acceleration
    plt.plot(activity_data['timestamp'], activity_data['z'], label='Z Acceleration

    plt.title(f'{device} Acceleration of participant no 1620 - {activity}')
    plt.xlabel('Timestamp')
    plt.ylabel('Acceleration')

    plt.legend(loc='upper right', frameon=False)
    plt.grid(True)
    plt.show()

for key in activity_codes_mapping:
    show_accel_per_activity('Watch', thi_participant_data, activity_codes_mapping[
```

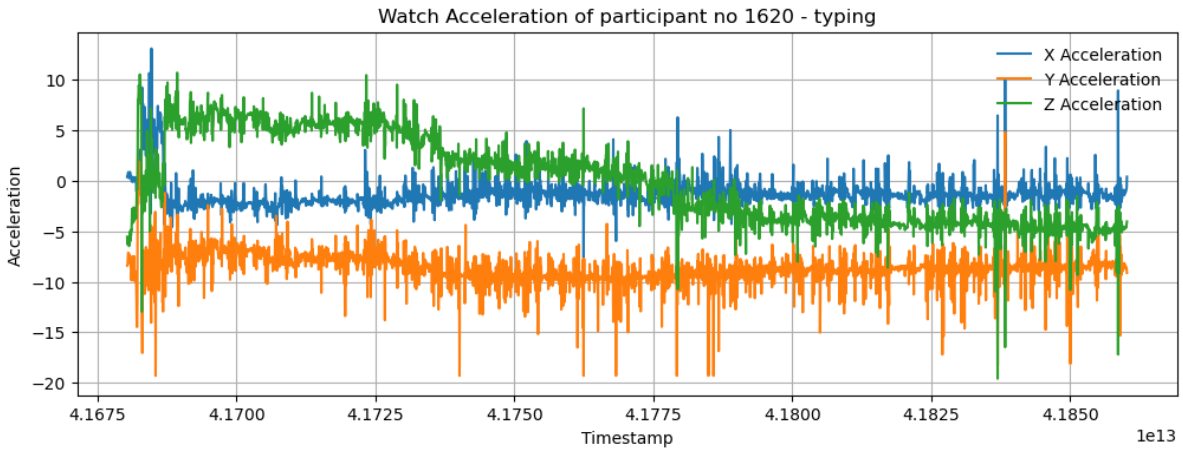


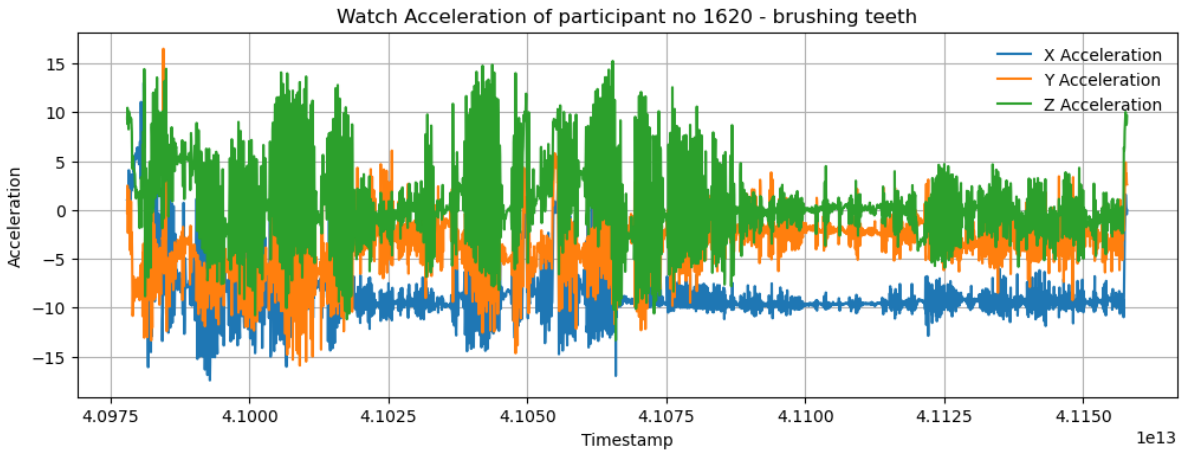


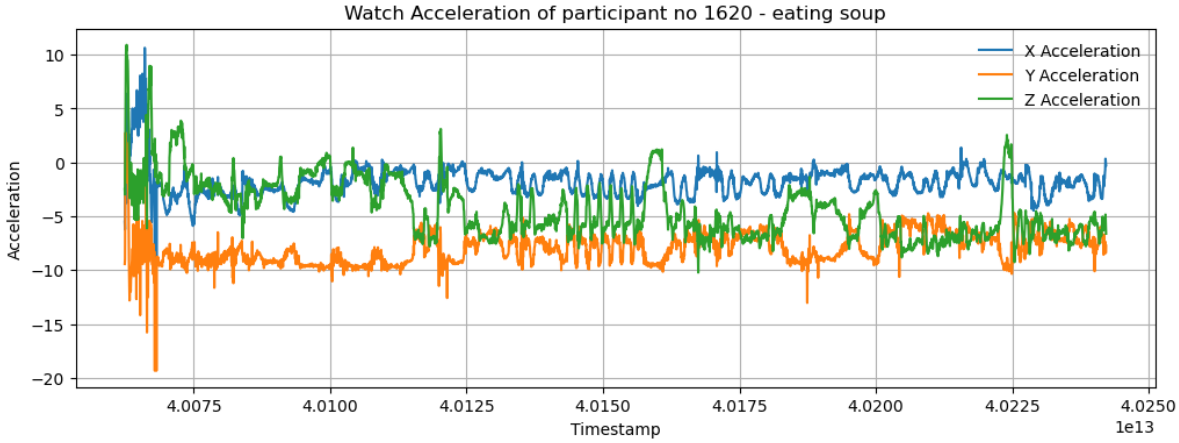


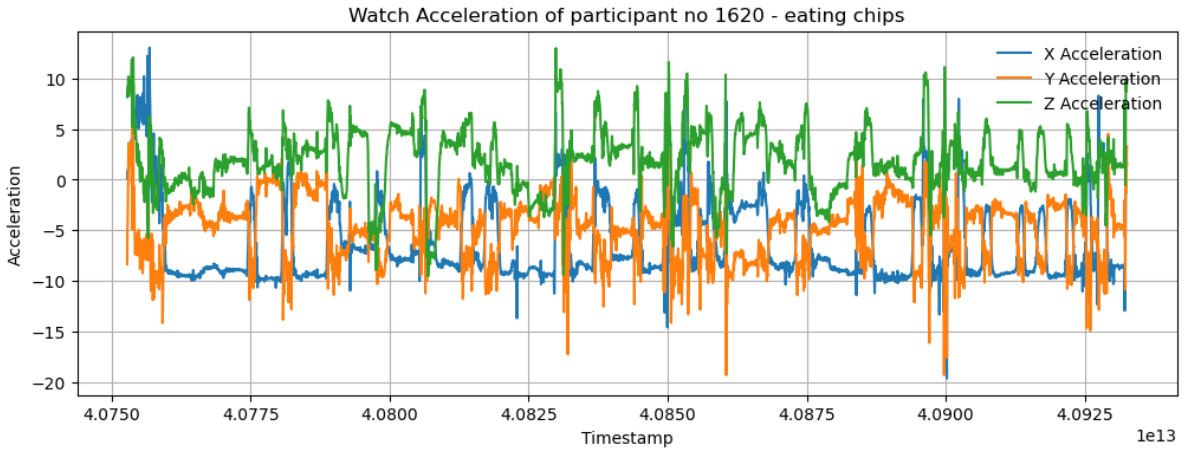

Watch Acceleration of participant no 1620 - sitting
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

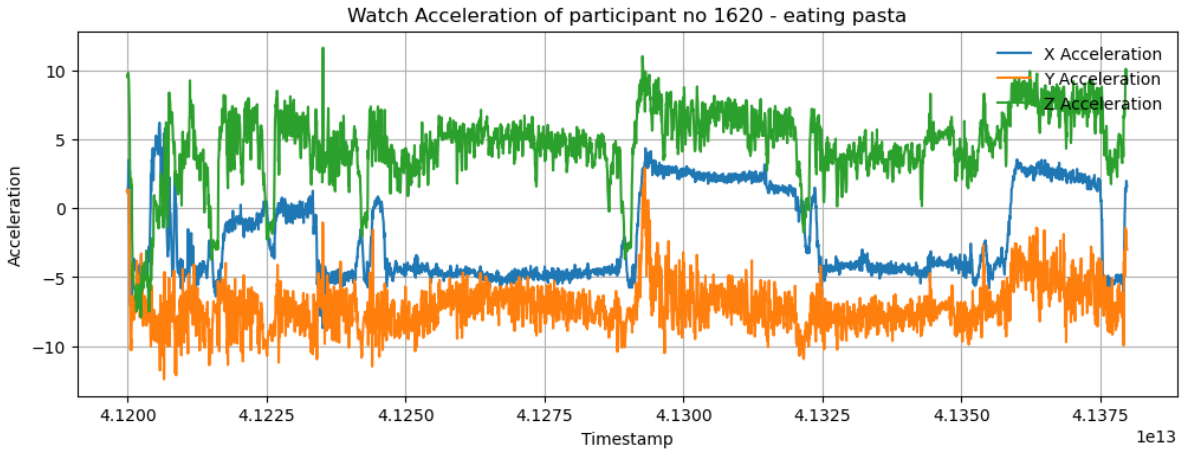

Watch Acceleration of participant no 1620 - standing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

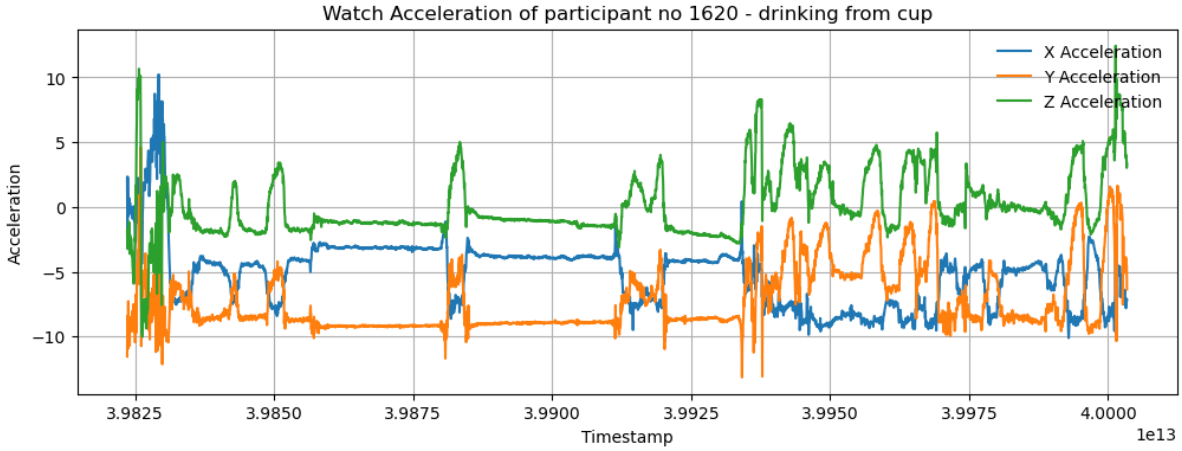

Watch Acceleration of participant no 1620 - typing
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

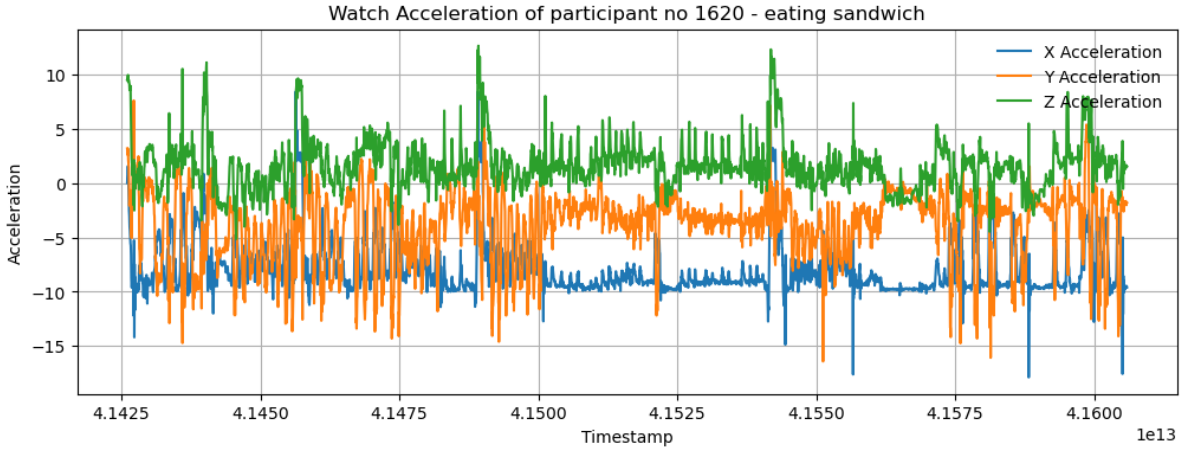

Watch Acceleration of participant no 1620 - brushing teeth
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

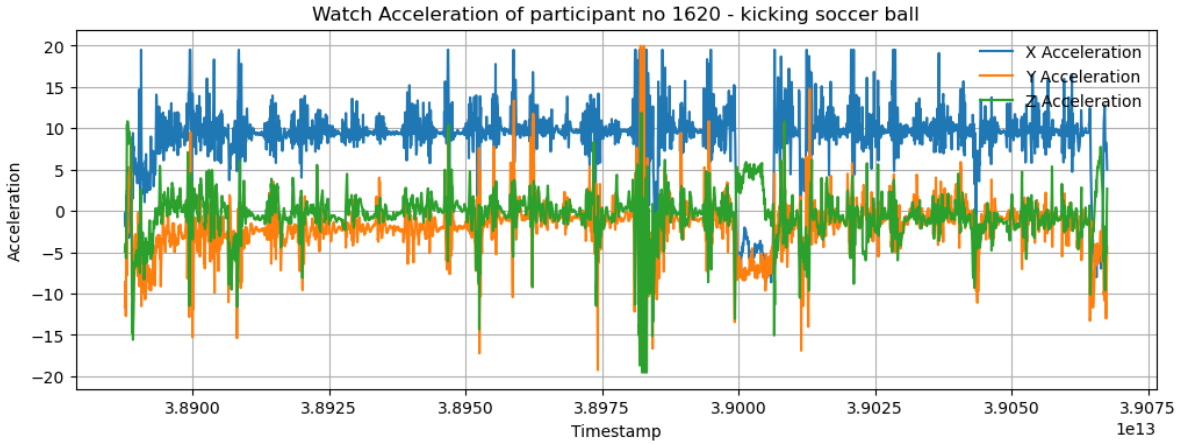

Watch Acceleration of participant no 1620 - eating soup
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

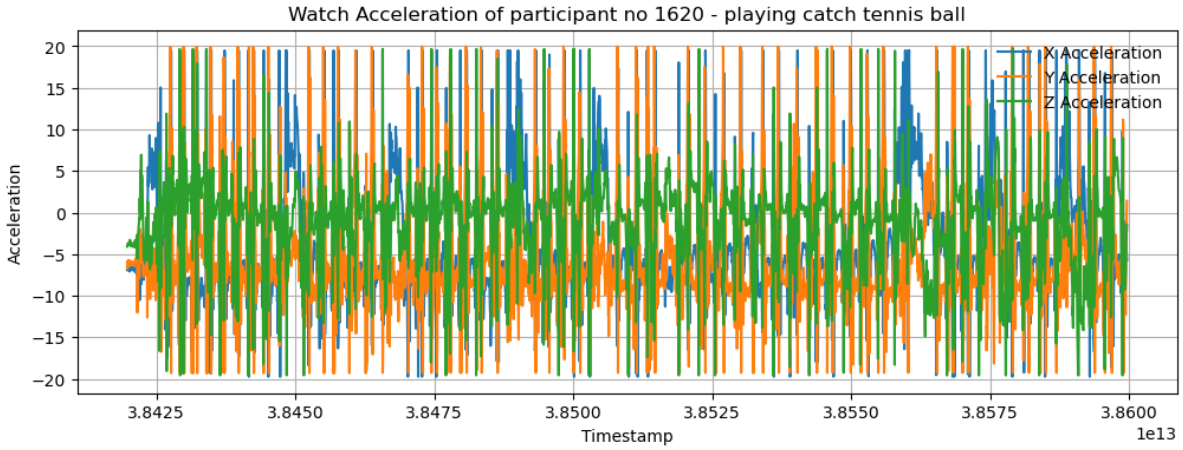

Watch Acceleration of participant no 1620 - eating chips
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

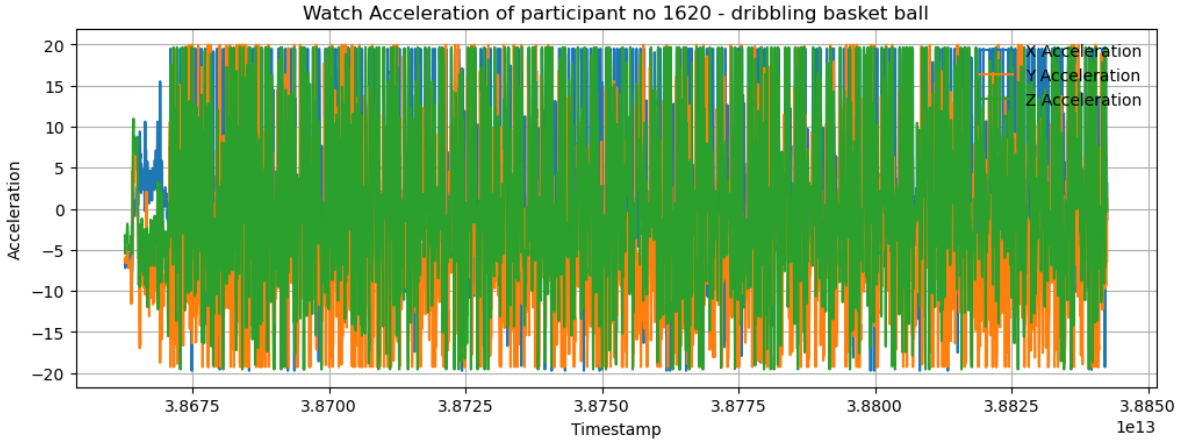

Watch Acceleration of participant no 1620 - eating pasta
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

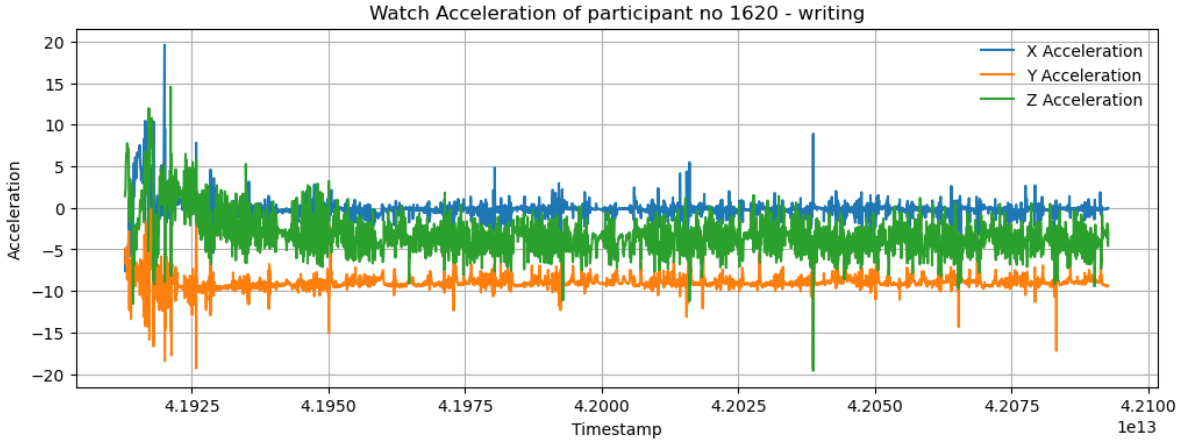

Watch Acceleration of participant no 1620 - drinking from cup
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

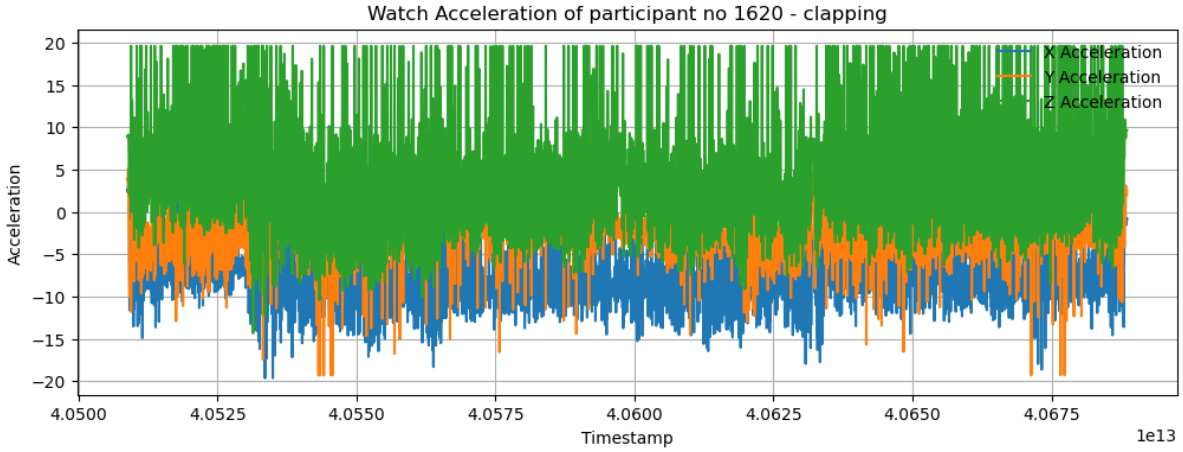

Watch Acceleration of participant no 1620 - eating sandwich
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13

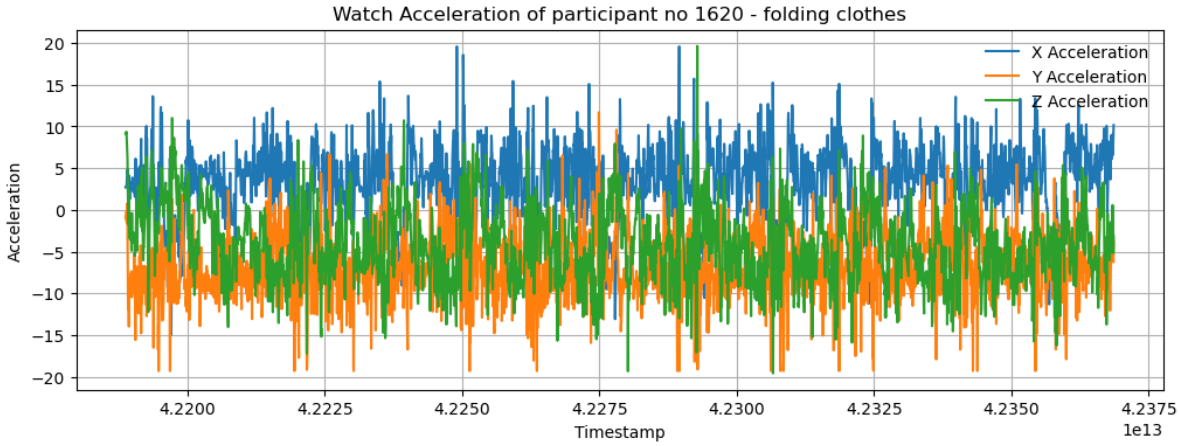

Watch Acceleration of participant no 1620 - kicking soccer ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13


Watch Acceleration of participant no 1620 - playing catch tennis ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13


Watch Acceleration of participant no 1620 - dribbling basket ball
X Acceleration
Y Acceleration
Z Acceleration
Acceleration
Timestamp
1e13





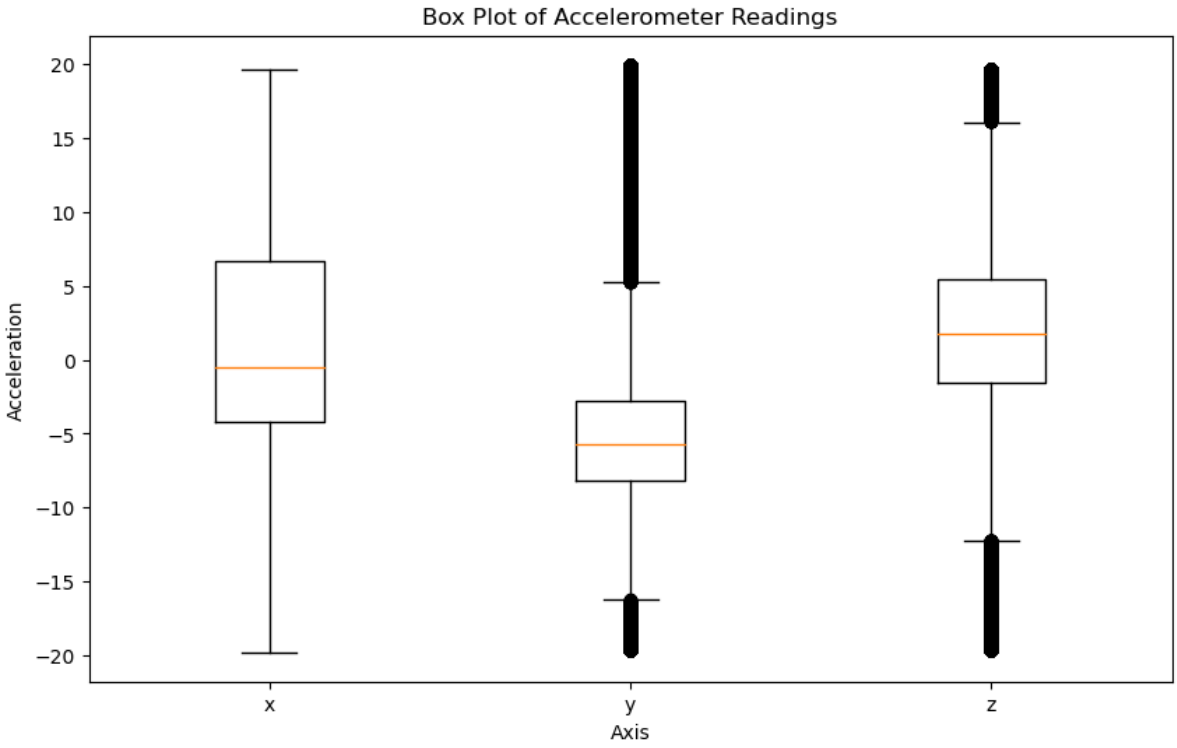


3.Data Preprocessing

In [20]:
```
# Check for missing values in data of all 3 participants
missing_values = combined_df.isnull().sum()
print("Missing Values:\n", missing_values)

# Drop rows with missing values
combined_df = combined_df.dropna()
```

```
Missing Values:
 participant_id    0
activity_code      0
activity           0
timestamp          0
x                  0
y                  0
z                  0
dtype: int64
```

In [21]:
```
#finding the outliers using boxplot method
import matplotlib.pyplot as plt

# Create a box plot of the accelerometer readings
plt.figure(figsize=(10, 6))
plt.boxplot([combined_df['x'], combined_df['y'], combined_df['z']])
plt.xticks([1, 2, 3], ['x', 'y', 'z'])
plt.xlabel('Axis')
plt.ylabel('Acceleration')
plt.title('Box Plot of Accelerometer Readings')
plt.show()
```



3.1 Feature Scaling

In [22]:
```
# Select the features to be scaled
features = ['x', 'y', 'z']

# Create a scaler object
scaler = StandardScaler()

# Fit the scaler to the selected features
scaler.fit(combined_df[features])

# Transform the selected features using the scaler
combined_df[features] = scaler.transform(combined_df[features])
```

3.2 Encoding Categorical Values

In [23]:
```
data_types = combined_df.dtypes

# Filter the columns with categorical variables
categorical_columns = data_types[data_types == 'object'].index.tolist()

print("Categorical Columns:")
print(categorical_columns)
```

```
Categorical Columns:
['activity_code', 'activity']
```

In [24]:
```
# Perform one-hot encoding
encoded_data = pd.get_dummies(combined_df, columns=['activity_code', 'activity'])

print("Encoded Data:")
print(encoded_data.head())
```

```
Encoded Data:
   participant_id       timestamp         x         y         z  \
0            1618  15001753925717  0.757946  0.355960  0.163999
1            1618  15001803425717  0.800070  0.295290  0.163517
2            1618  15001852925717  1.004771  0.429484  0.261296
3            1618  15001902425717  1.234537  0.484498  0.334510
4            1618  15001951925717  1.642198  0.390409  0.370636

   activity_code_A  activity_code_B  activity_code_C  activity_code_D  \
0                1                0                0                0
1                1                0                0                0
2                1                0                0                0
3                1                0                0                0
4                1                0                0                0

   activity_code_E  ...  activity_folding clothes  activity_jogging  \
0                0  ...                         0                 0
1                0  ...                         0                 0
2                0  ...                         0                 0
3                0  ...                         0                 0
4                0  ...                         0                 0

   activity_kicking soccer ball  activity_playing catch tennis ball  \
0                             0                                   0
1                             0                                   0
2                             0                                   0
3                             0                                   0
4                             0                                   0

   activity_sitting  activity_stairs  activity_standing  activity_typing  \
0                 0                0                 0                0
1                 0                0                 0                0
2                 0                0                 0                0
3                 0                0                 0                0
4                 0                0                 0                0

   activity_walking  activity_writing
0                 1                0
1                 1                0
2                 1                0
3                 1                0
4                 1                0

[5 rows x 41 columns]
```

In [25]:
```
X =encoded_data .drop(['activity_code_A', 'activity_code_B', 'activity_code_C', 'a
y = encoded_data[['activity_code_A', 'activity_code_B', 'activity_code_C', 'activi
```

3.3 Dimentionality reduction using PCA

In [26]:
```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Apply PCA
pca = PCA(n_components=4)  # Set the desired number of components
X_pca = pca.fit_transform(X_scaled)
```

3.4 Data Splitting

In [27]:
```
# Split the data in 80%-20% into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, rando
```

4.Building Models

4.1 Training RandomForestClassifier

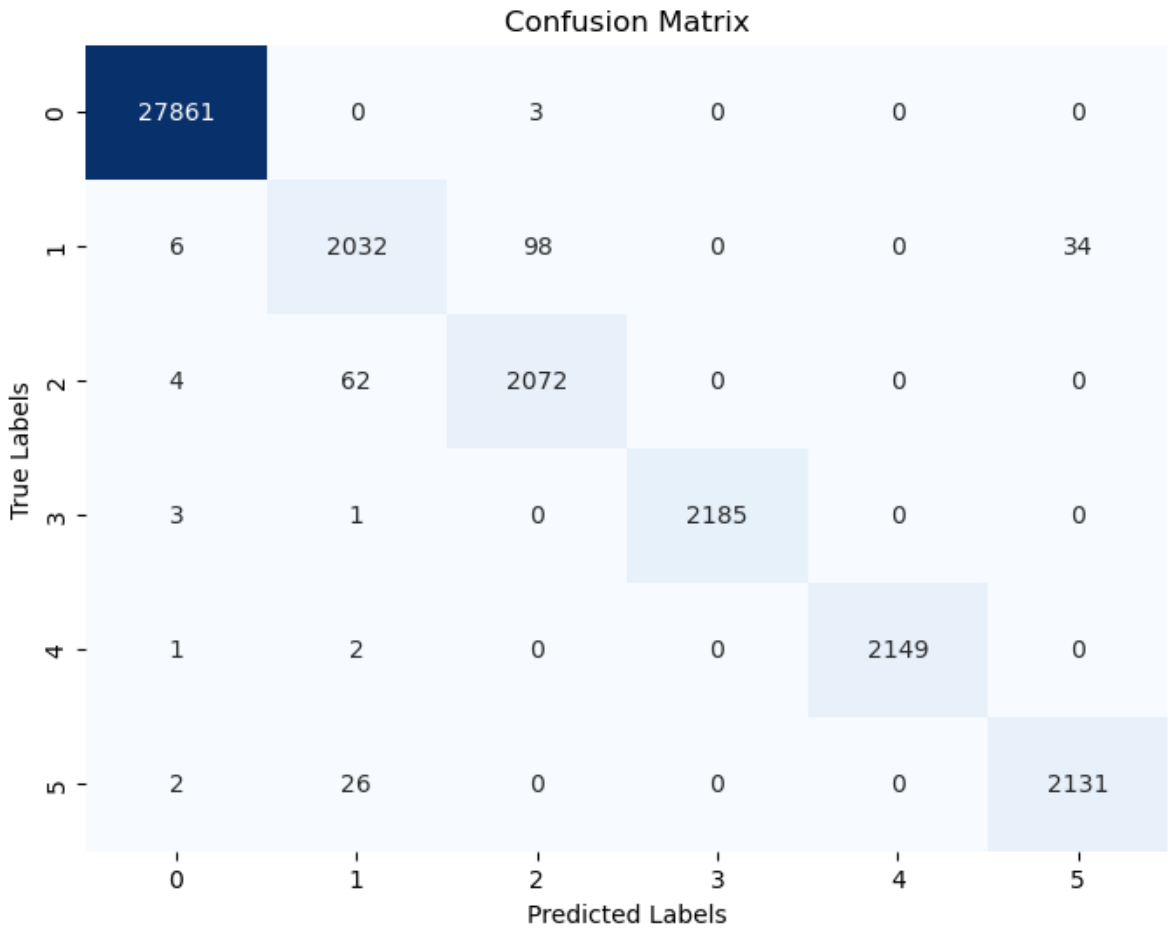
In [28]:
```
# Create a Random Forest Classifier object
rf_classifier = RandomForestClassifier()

# Train the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Calculate precision, recall, and F1-score
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.9937422424493173
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2122
           1       0.96      0.94      0.95      2170
           2       0.95      0.97      0.96      2138
           3       1.00      1.00      1.00      2189
           4       1.00      1.00      1.00      2152
           5       0.98      0.99      0.99      2159

   micro avg       0.98      0.98      0.98     12930
   macro avg       0.98      0.98      0.98     12930
weighted avg       0.98      0.98      0.98     12930
 samples avg       0.33      0.33      0.33     12930
```

```
C:\Users\haris\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\haris\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [29]:

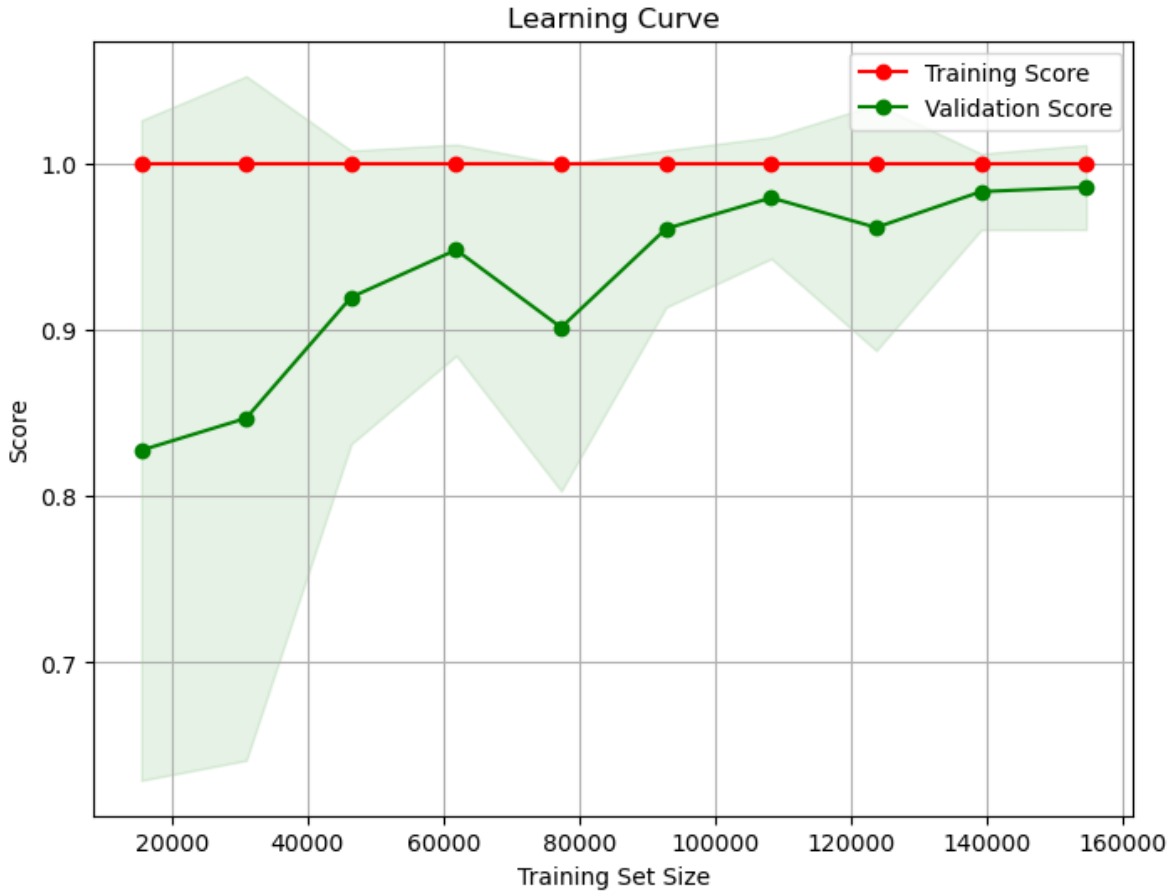
```
# Convert multilabel-indicator format to binary format
y_test_binary = label_binarize(y_test, classes=[0, 1, 2, 3, 4, 5])
y_pred_binary = label_binarize(y_pred, classes=[0, 1, 2, 3, 4, 5])

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test_binary.argmax(axis=1), y_pred_binary.argmax(

# Create a heatmap using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```



In [30]:

```
def plot_learning_curve(estimator, X, y):
    train_sizes, train_scores, val_scores = learning_curve(
        estimator, X, y, train_sizes=np.linspace(0.1, 1.0, 10), cv=5
    )

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
```

```
    val_mean = np.mean(val_scores, axis=1)
    val_std = np.std(val_scores, axis=1)

    plt.figure(figsize=(8, 6))
    plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training Score')
    plt.plot(train_sizes, val_mean, 'o-', color='g', label='Validation Score')
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, a
    plt.fill_between(train_sizes, val_mean - val_std, val_mean + val_std, alpha=0.1
    plt.xlabel('Training Set Size')
    plt.ylabel('Score')
    plt.title('Learning Curve')
    plt.legend(loc='best')
    plt.grid(True)
    plt.show()

model = RandomForestClassifier()

plot_learning_curve(model, X, y)
```



4.2 Training DecisionTreeClassifier

In [33]:

```
dt_classifier = DecisionTreeClassifier()

# Train the classifier
dt_classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = dt_classifier.predict(X_test)

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

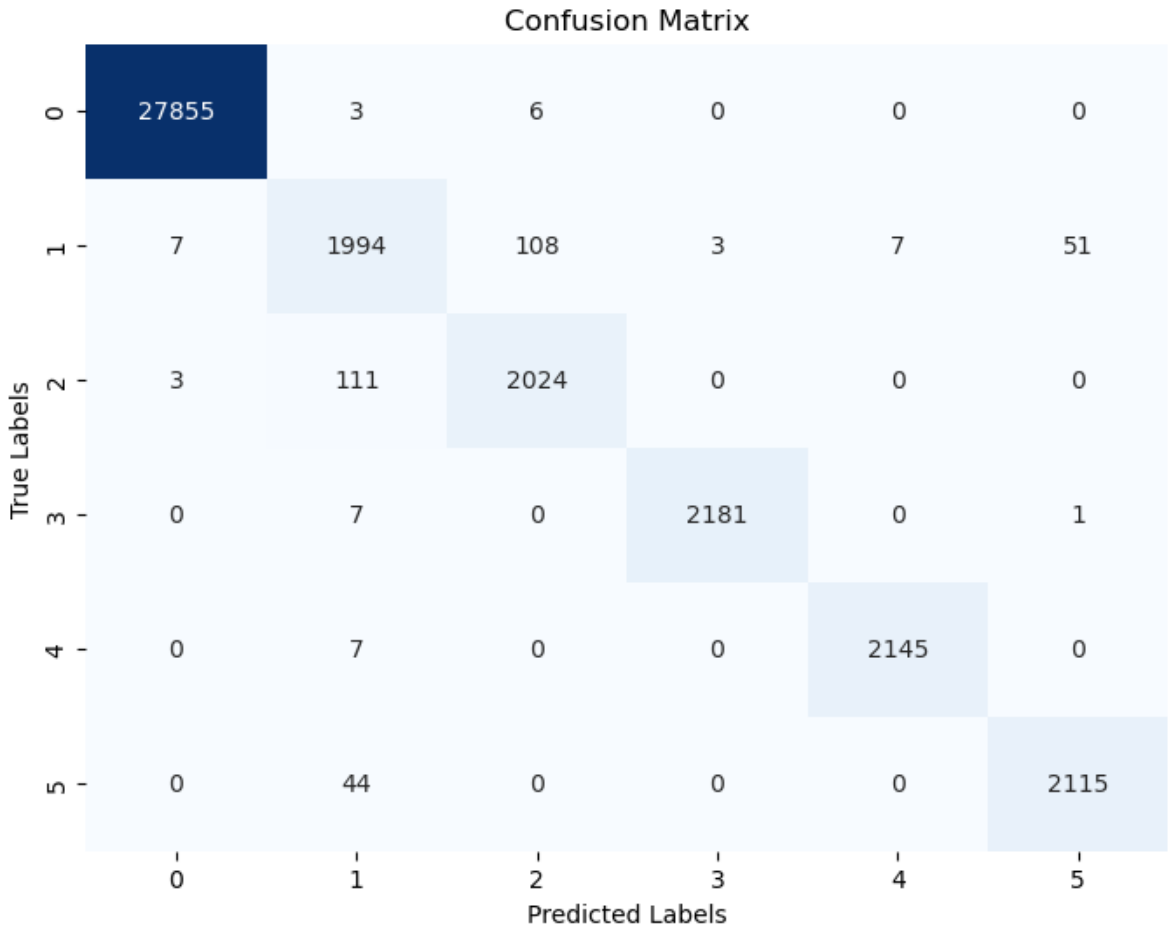
```
# Calculate and print precision, recall, and F1-score
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.9907685146876293
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2122
           1       0.93      0.91      0.92      2170
           2       0.94      0.95      0.95      2138
           3       1.00      1.00      1.00      2189
           4       1.00      1.00      1.00      2152
           5       0.97      0.98      0.98      2159

   micro avg       0.97      0.97      0.97     12930
   macro avg       0.97      0.97      0.97     12930
weighted avg       0.97      0.97      0.97     12930
 samples avg       0.33      0.33      0.33     12930
```

```
C:\Users\haris\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\haris\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [32]:

```
# Convert multilabel-indicator format to binary format
y_test_binary = label_binarize(y_test, classes=[0, 1, 2, 3, 4, 5])
y_pred_binary = label_binarize(y_pred, classes=[0, 1, 2, 3, 4, 5])

# Generate the confusion matrix
conf_matrix = confusion_matrix(y_test_binary.argmax(axis=1), y_pred_binary.argmax(a

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```


Confusion Matrix
True Labels
Predicted Labels
0
1
2
3
4
5
27855
3
6
0
0
0
7
1994
108
3
7
51
3
111
2024
0
0
0
0
7
0
2181
0
1
0
7
0
0
2145
0
0
44
0
0
0
2115

In [ ]:

In [ ]: