AD_2_final

June 5, 2024

1 Activity Detection

Part 2

Data source: https://www.kaggle.com/datasets/luisomoreau/activity-detection

Our data consists of 12 folders, where each folder represents one activity. In each folder (except one), there are 12 CSV files with data. Each CSV file corresponds to one sensor that recorded the data. A description of the files with their values is provided below.

Acceleration (Accelerometer) - Accelerometer_z: Acceleration along the Z-axis. - Accelerometer_y: Acceleration along the Y-axis. - Accelerometer_x: Acceleration along the X-axis.

Annotation - empty

Gravity - Gravity_z: Gravity vector component along the Z-axis. - Gravity_y: Gravity vector component along the Y-axis. - Gravity_x: Gravity vector component along the X-axis.

Gyroscope - Gyroscope_z: Angular velocity around the Z-axis. - Gyroscope_y: Angular velocity around the Y-axis. - Gyroscope_x: Angular velocity around the X-axis.

Location - Location_bearingAccuracy: Bearing (azimuth) accuracy in location. - Location_speedAccuracy: Speed accuracy in location. - Location_verticalAccuracy: Altitude accuracy in location. - Location_horizontalAccuracy: Horizontal accuracy in location. - Location_speed: Speed in location. - Location_bearing: Bearing (azimuth) in location. - Location_altitude: Altitude in location. - Location_longitude: Longitude in location. - Location_latitude: Latitude in location.

Metadata - additional data

GPS (LocationGps) - LocationGps_bearingAccuracy: Bearing (azimuth) accuracy obtained from GPS. - LocationGps_speedAccuracy: Speed accuracy obtained from GPS. - LocationGps_verticalAccuracy: Altitude accuracy obtained from GPS. - LocationGps_horizontalAccuracy: Horizontal accuracy obtained from GPS. - LocationGps_speed: Speed obtained from GPS. - LocationGps_bearing: Bearing (azimuth) obtained from GPS. - LocationGps_altitude: Altitude obtained from GPS. - LocationGps_longitude: Longitude obtained from GPS. - LocationGps_latitude: Latitude obtained from GPS.

Network Location (LocationNetwork) - LocationNetwork_bearingAccuracy: Bearing (azimuth) accuracy obtained from the network. - LocationNetwork_speedAccuracy: Speed accuracy obtained from the network. - LocationNetwork_verticalAccuracy: Altitude accuracy obtained from the network. - LocationNetwork_horizontalAccuracy: Horizontal accuracy obtained from the network. - LocationNetwork_speed: Speed obtained from the network. - LocationNetwork_bearing: Bearing (azimuth) obtained from the network. - LocationNetwork_altitude: Altitude obtained from the network. - LocationNetwork_longitude: Longitude obtained from the network. - LocationNetwork_latitude: Latitude obtained from the network.

Magnetometer - Magnetometer_z: Magnetic field strength along the Z-axis. - Magnetometer_y: Magnetic field strength along the Y-axis. - Magnetometer_x: Magnetic field strength along the X-axis.

Orientation - Orientation_qz: Z component of the quaternion representing orientation. - Orientation_qy: Y component of the quaternion representing orientation. - Orientation_qx: X component of the quaternion representing orientation. - Orientation_qw: W component of the quaternion representing orientation. - Orientation_roll: Roll angle of the orientation. - Orientation_pitch: Pitch angle of the orientation. - Orientation_yaw: Yaw angle of the orientation.

Pedometer - Pedometer_steps: Number of steps recorded by the pedometer.

Total Acceleration - TotalAcceleration_z: Total acceleration along the Z-axis. - TotalAcceleration_y: Total acceleration along the Y-axis. - TotalAcceleration_x: Total acceleration along the X-axis.

1.1 BUSINESS GOAL

We work for a company that makes devices for athletes (like sports watches) that track physical activities. Using sensors, they collect data such as speed and location from each activity separately. The user doesn't select the type of activity - the smart system just knows when they start doing something. This way, we get a bunch of activities with different data points. We want to cluster these activities to figure out what kinds of activities our users prefer and when they do them. This can be used for more personalized ads or for classification problems.

1.2 EDA

1.2.1 Imports

[ ]:
```
import pandas as pd
import numpy as np
import datetime
import os
import seaborn as sns
import matplotlib.pyplot as plt
import math
import warnings
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import PowerTransformer
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.pipeline import make_pipeline
from sklearn.metrics import␣
↪silhouette_score,davies_bouldin_score,calinski_harabasz_score


warnings.filterwarnings("ignore")
```

1.2.2 Reading prepared in part 1 csv file

[ ]:
```
result = pd.read_csv("..//final_data//result_data.csv")
Y_train = pd.read_csv("..//final_data//Y_train.csv")

# Y_train = pd.read_csv("..//final_data//Y_valid.csv") # for validation
# result = pd.read_csv("..//final_data//result_data_valid.csv") # for␣
 ↪validation

# Y_train = pd.read_csv("..//final_data//Y_valid.csv") # for validation
# result = pd.read_csv("..//final_data//result_data_valid.csv") # for␣
 ↪validation
```

1.2.3 Some info about data frame

[ ]:
```
result
```

[ ]:
```
          id  total_time    mean_speed     max_speed     min_speed  \
0        7.0    2.516518  2.408345e+01  2.522632e+01  2.356563e+01
1        8.0    2.516498  2.172662e+01  2.387931e+01  1.837205e+01
2        9.0    2.516481  1.501819e+01  1.837205e+01  1.257880e+01
3       12.0    2.516464  2.259201e+01  2.355715e+01  1.923772e+01
4       13.0    2.516464  2.374844e+01  2.398566e+01  2.299106e+01
...      ...         ...           ...           ...           ...
1357  2774.0    2.516899  9.276274e-31  5.110263e-30  1.783098e-33
1358  2776.0    2.516883  2.389216e-36  1.065754e-35  2.291695e-39
1359  2777.0    2.516877  8.063703e-40  2.291695e-39  3.944935e-42
1360  2778.0    2.516874  9.794135e-43  3.944935e-42  0.000000e+00
1361  2780.0    0.859112  0.000000e+00  0.000000e+00  0.000000e+00

      total_distance  mean_acceleration  max_acceleration  min_acceleration  \
0           0.013338           9.919805         33.370849          2.437895
1           0.016061           9.841829         19.279124          4.524240
2           0.007480           9.842018         33.347219          3.645055
3           0.020190          10.071431         14.881125          3.962931
4           0.021156           9.505357         13.644567          4.475255
...              ...                ...               ...               ...
1357        0.000000           9.717226         10.134856          9.111182
1358        0.000000           9.713042         10.172372          9.214315
1359        0.000000           9.713457          9.876455          9.330433
1360        0.000000           9.747030         12.981812          7.583295
1361        0.000000          10.352763         13.589607          9.024704

      sd_acceleration  ...  average_pitch  median_pitch  min_pitch  max_pitch  \
0            4.656513  ...       0.983790      1.084632   0.555272   1.180261
1            2.392057  ...       1.074759      1.156837   0.672223   1.221615
2            3.281112  ...       0.920063      0.955256   0.625777   1.166046
```

```
3          2.263844  ...       0.889693      0.945513   0.531719   1.186114
4          2.545282  ...       0.843380      0.854464   0.531811   1.167627
...             ...  ...            ...           ...        ...        ...
1357       0.096787  ...      -0.265641     -0.260294  -0.298774  -0.245418
1358       0.111261  ...      -0.260877     -0.256462  -0.307705  -0.236793
1359       0.043896  ...      -0.250566     -0.250087  -0.258994  -0.245895
1360       0.482260  ...      -0.246482     -0.245037  -0.314204  -0.229960
1361       0.747762  ...      -0.604976     -0.651176  -0.708414  -0.281792

      sd_pitch  average_yaw  median_yaw   min_yaw   max_yaw    sd_yaw
0     0.206061     0.926043    0.940593  0.722495  1.136638  0.098998
1     0.162367     1.097486    1.082139  0.880115  1.298065  0.131101
2     0.184394     1.109633    1.101204  0.762258  1.389442  0.170913
3     0.203140     1.555590    1.521060  1.384156  1.799936  0.115415
4     0.201057     1.613284    1.568194  1.453768  1.920251  0.126165
...        ...          ...         ...       ...       ...       ...
1357  0.013399     2.783523    2.790204  2.725673  2.810498  0.017086
1358  0.013606     2.770419    2.778999  2.704930  2.806852  0.033515
1359  0.002698     2.737599    2.740781  2.725769  2.745026  0.006032
1360  0.008723     2.727584    2.732468  2.577463  2.743785  0.023602
1361  0.103262     2.227272    2.289268  1.984317  2.425556  0.132824

[1362 rows x 29 columns]
```

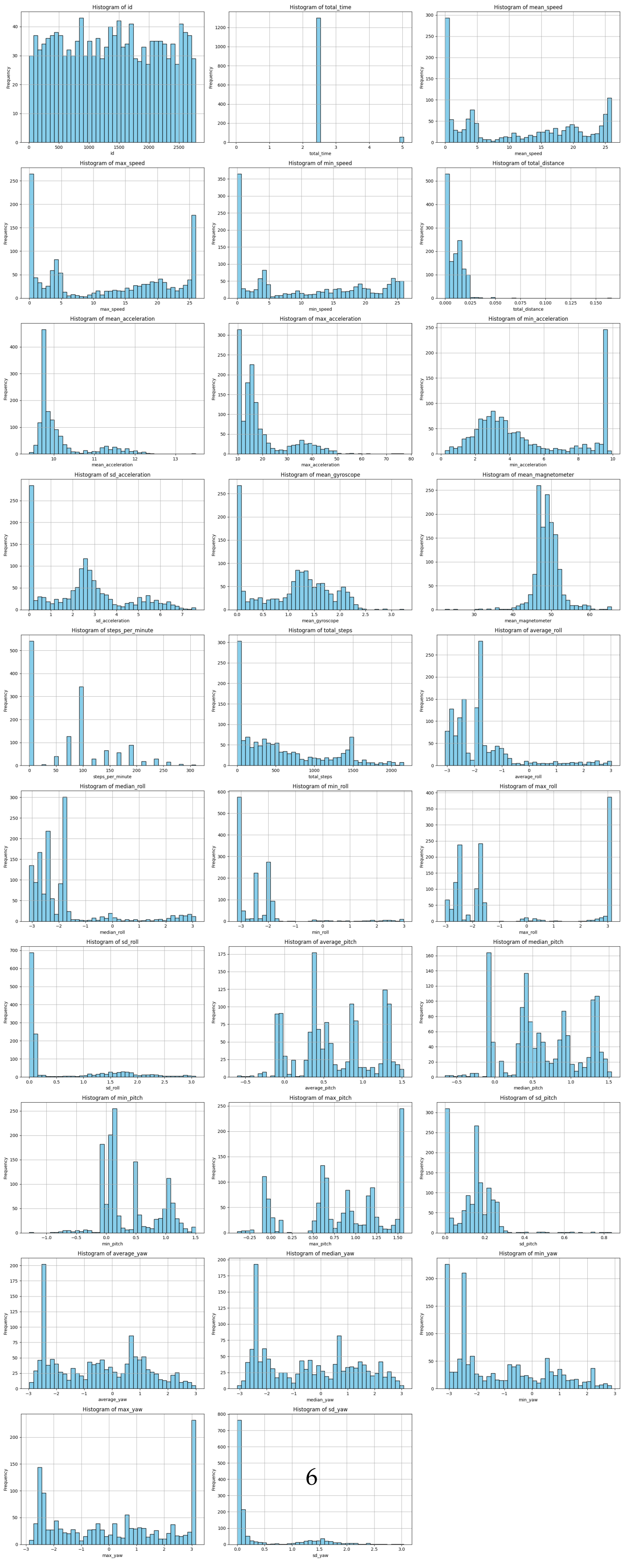
[ ]:
```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1362 entries, 0 to 1361
Data columns (total 29 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  1362 non-null   float64
 1   total_time          1362 non-null   float64
 2   mean_speed          1362 non-null   float64
 3   max_speed           1362 non-null   float64
 4   min_speed           1362 non-null   float64
 5   total_distance      1362 non-null   float64
 6   mean_acceleration   1362 non-null   float64
 7   max_acceleration    1362 non-null   float64
 8   min_acceleration    1362 non-null   float64
 9   sd_acceleration     1362 non-null   float64
 10  mean_gyroscope      1362 non-null   float64
 11  mean_magnetometer   1362 non-null   float64
 12  steps_per_minute    1362 non-null   float64
 13  total_steps         1362 non-null   float64
 14  average_roll        1362 non-null   float64
 15  median_roll         1362 non-null   float64
 16  min_roll            1362 non-null   float64
```

```
 17   max_roll           1362 non-null   float64
 18   sd_roll            1362 non-null   float64
 19   average_pitch      1362 non-null   float64
 20   median_pitch       1362 non-null   float64
 21   min_pitch          1362 non-null   float64
 22   max_pitch          1362 non-null   float64
 23   sd_pitch           1362 non-null   float64
 24   average_yaw        1362 non-null   float64
 25   median_yaw         1362 non-null   float64
 26   min_yaw            1362 non-null   float64
 27   max_yaw            1362 non-null   float64
 28   sd_yaw             1362 non-null   float64
dtypes: float64(29)
memory usage: 308.7 KB
```

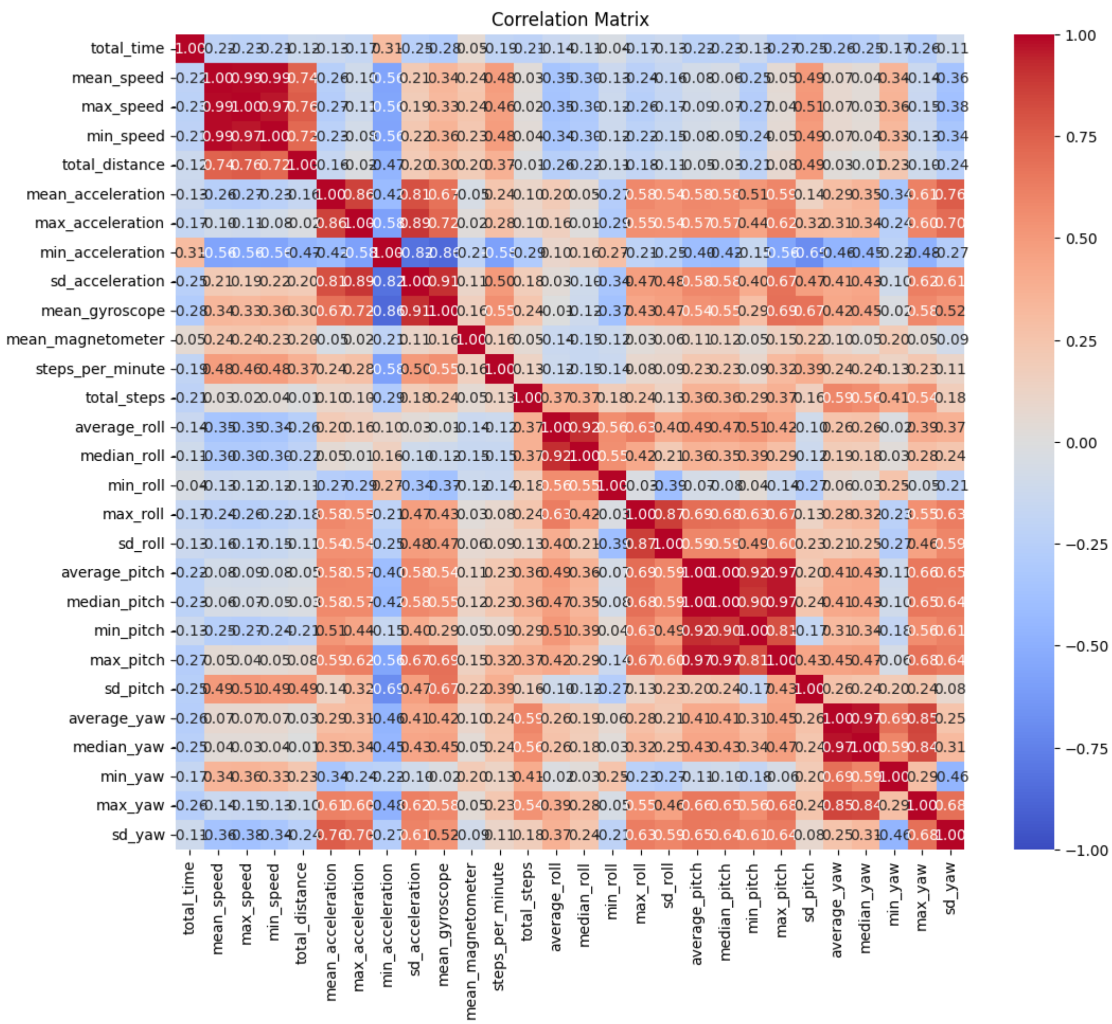
1.2.4 Histograms for every column

[ ]:
```
num_columns = len(result.columns)
num_rows = (num_columns + 2) // 3

fig, axes = plt.subplots(num_rows, 3, figsize=(20, 5 * num_rows))
axes = axes.flatten()

for i, column in enumerate(result.columns):
    axes[i].hist(result[column], bins=40, color='skyblue', edgecolor='black')
    axes[i].set_title(f'Histogram of {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')
    axes[i].grid(True)

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()

print()
print()
```


Histogram of id
Histogram of total_time
Histogram of mean_speed
Histogram of max_speed
Histogram of min_speed
Histogram of total_distance
Histogram of mean_acceleration
Histogram of max_acceleration
Histogram of min_acceleration
Histogram of sd_acceleration
Histogram of mean_gyroscope
Histogram of mean_magnetometer
Histogram of steps_per_minute
Histogram of total_steps
Histogram of average_roll
Histogram of median_roll
Histogram of min_roll
Histogram of max_roll
Histogram of sd_roll
Histogram of average_pitch
Histogram of median_pitch
Histogram of min_pitch
Histogram of max_pitch
Histogram of sd_pitch
Histogram of average_yaw
Histogram of median_yaw
Histogram of min_yaw
Histogram of max_yaw
Histogram of sd_yaw

1.2.5 Heatmap of correlation

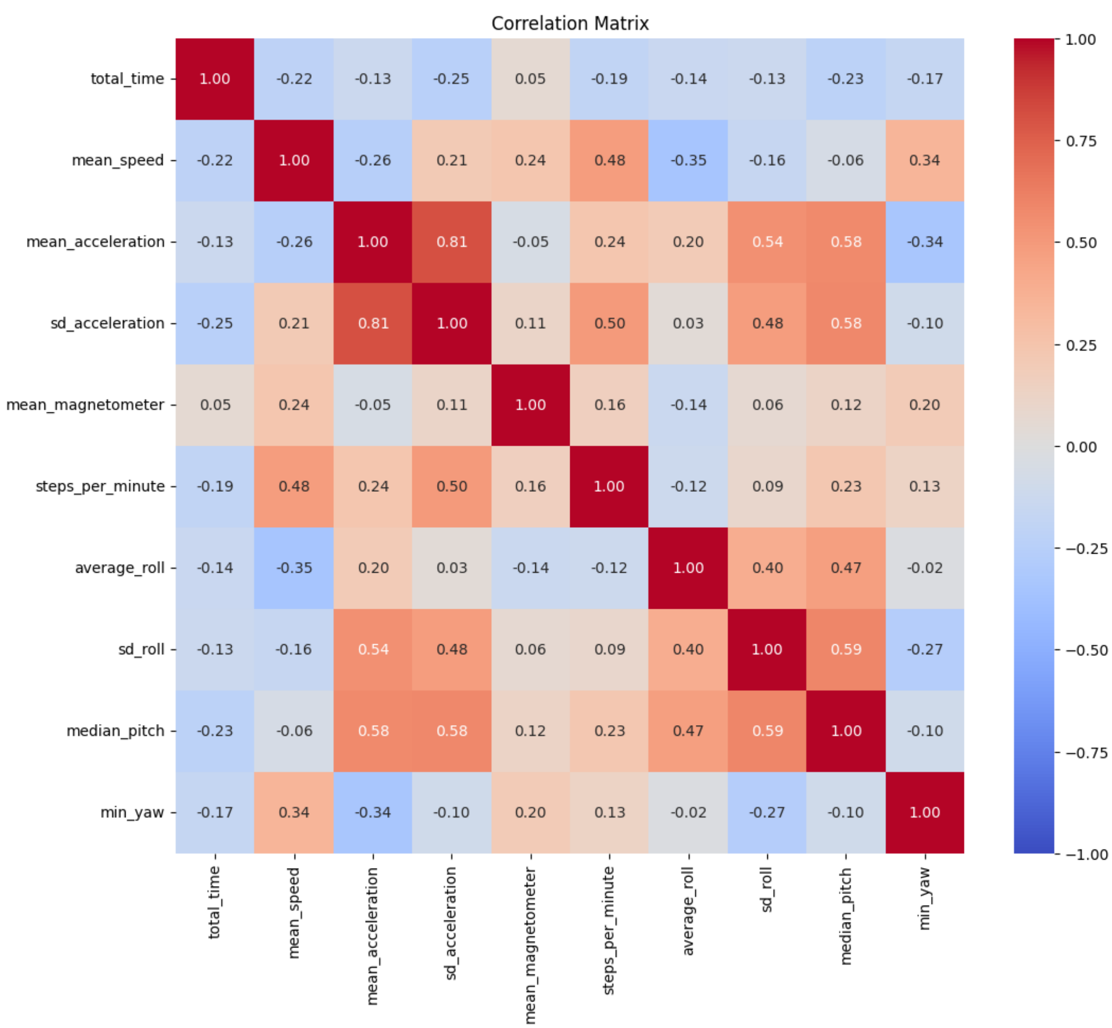
[ ]:
```
plt.figure(figsize=(12, 10))
sns.heatmap(result.loc[:,result.columns != 'id'].corr(), annot=True,␣
↪cmap='coolwarm', fmt=".2f", vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



1.2.6 Dropping correlated columns

[ ]:
```
reduced_result = result.drop(columns=['sd_yaw', 'min_speed', 'total_steps',␣
↪'max_speed', 'min_pitch', 'total_distance', 'median_roll', 'max_yaw',␣
↪'median_yaw', 'average_yaw', 'sd_pitch', 'max_pitch', 'average_pitch',␣
↪'max_acceleration', 'min_acceleration', 'mean_gyroscope', 'max_roll',␣
↪'min_roll'])
```

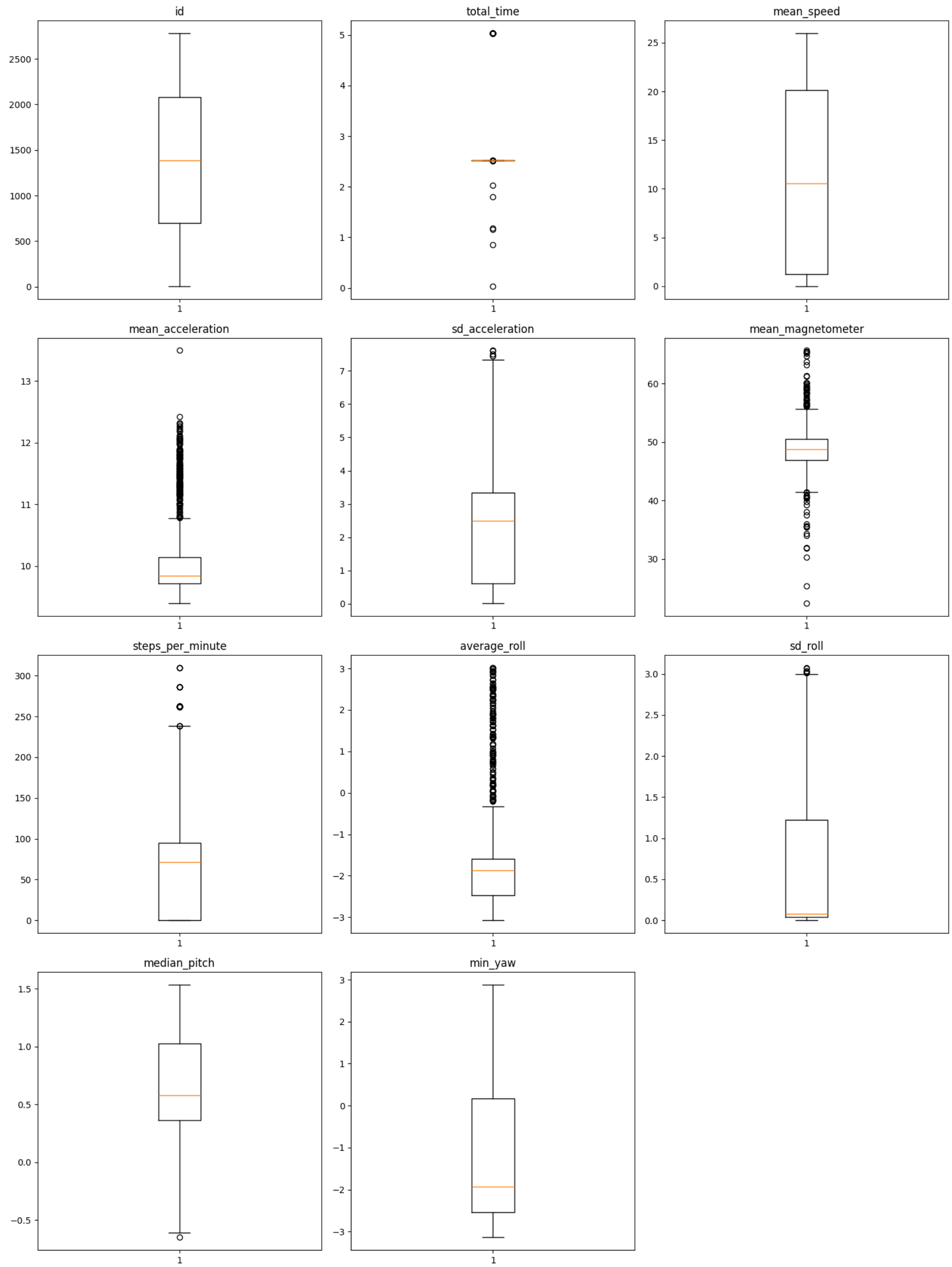
[ ]:
```
plt.figure(figsize=(12, 10))
sns.heatmap(reduced_result.loc[:, reduced_result.columns != 'id'].corr(),
            annot=True,
            cmap='coolwarm',
            fmt=".2f",
            center=0,
            vmin=-1,
            vmax=1)
plt.title('Correlation Matrix')
plt.show()
```



1.2.7 Boxplots for every column

[ ]:
```python
def plot_boxplots(df):
    num_cols = len(df.columns)

    num_rows = (num_cols + 2) // 3  # Round up to the nearest integer

    plt.figure(figsize=(15, 5 * num_rows))

    for i, col in enumerate(df.columns):
        plt.subplot(num_rows, 3, i + 1)
        plt.boxplot(df[col])
        plt.title(col)

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

[ ]:
```python
plot_boxplots(reduced_result)
```



We need to modify outliers in total_time and mean_magnetometer.

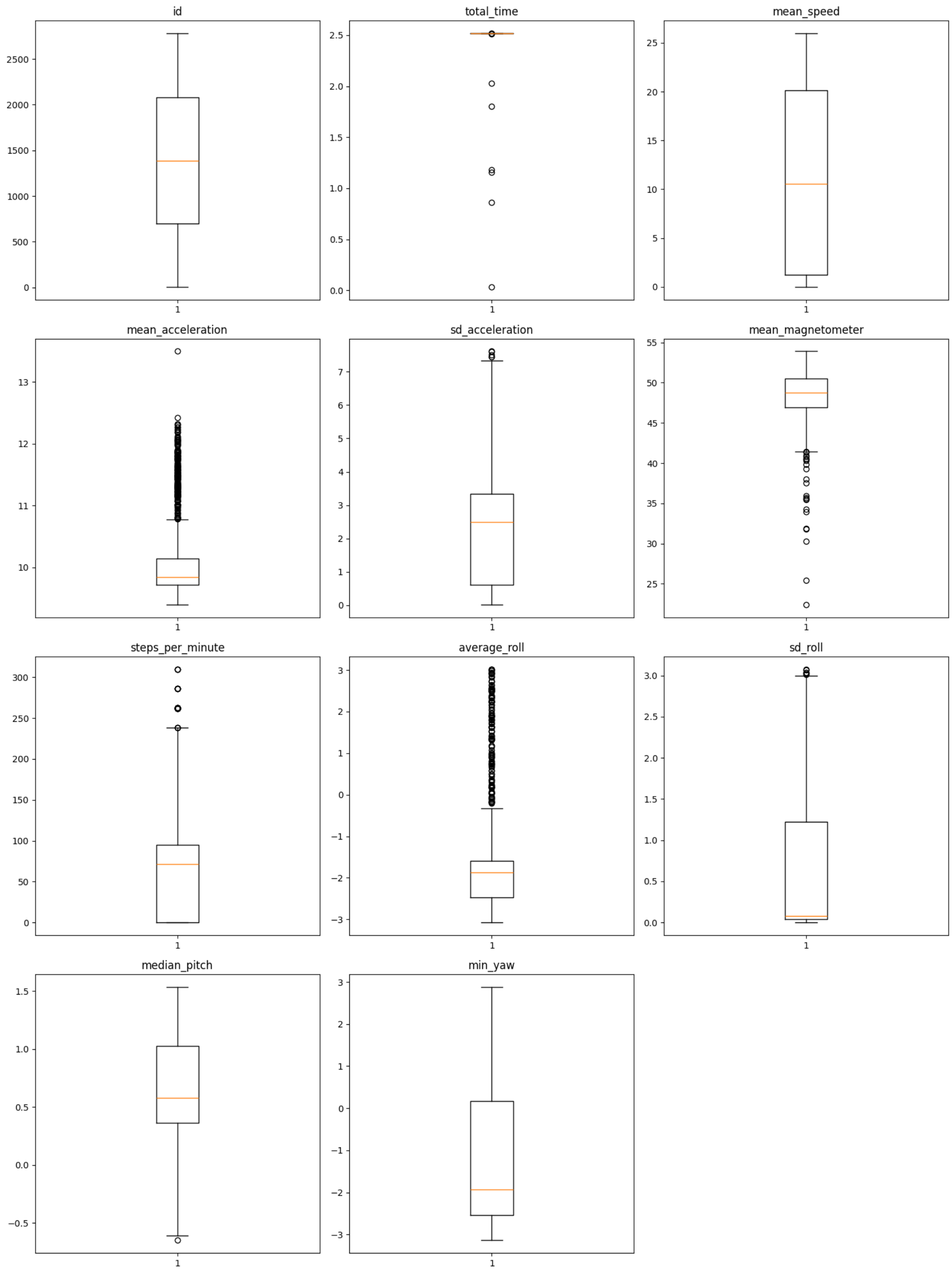
[ ]:
```python
def replace_outliers_with_quantile(df):
    time_quantile = df['total_time'].quantile(0.95)
    magneto_quantile = df['mean_magnetometer'].quantile(0.95)

    df.loc[df['total_time'] > time_quantile, 'total_time'] = time_quantile

    df.loc[df['mean_magnetometer'] > magneto_quantile, 'mean_magnetometer'] =␣
 ↪magneto_quantile

    return df
```

[ ]:
```python
replace_outliers_with_quantile(reduced_result)
```

[ ]:
```
          id  total_time    mean_speed  mean_acceleration  sd_acceleration  \
0        7.0    2.516518  2.408345e+01           9.919805         4.656513
1        8.0    2.516498  2.172662e+01           9.841829         2.392057
2        9.0    2.516481  1.501819e+01           9.842018         3.281112
3       12.0    2.516464  2.259201e+01          10.071431         2.263844
4       13.0    2.516464  2.374844e+01           9.505357         2.545282
...      ...         ...           ...                ...              ...
1357  2774.0    2.516899  9.276274e-31           9.717226         0.096787
1358  2776.0    2.516883  2.389216e-36           9.713042         0.111261
1359  2777.0    2.516877  8.063703e-40           9.713457         0.043896
1360  2778.0    2.516874  9.794135e-43           9.747030         0.482260
1361  2780.0    0.859112  0.000000e+00          10.352763         0.747762

      mean_magnetometer  steps_per_minute  average_roll   sd_roll  \
0             53.910455        119.212340     -2.900108  0.114005
1             53.910455          0.000000     -1.040673  2.811269
2             53.910455        286.113823     -1.493453  2.470934
3             53.910455         95.371919     -2.788037  0.140679
4             53.910455        143.057878     -2.597302  0.997407
...                 ...               ...           ...       ...
1357          47.345006          0.000000     -2.792382  0.016608
1358          47.278723          0.000000     -2.758431  0.035709
1359          46.980630          0.000000     -2.694629  0.003280
1360          47.180419          0.000000     -2.711286  0.013943
1361          47.325072          0.000000     -0.134196  0.051130

      median_pitch   min_yaw
0         1.084632  0.722495
1         1.156837  0.880115
2         0.955256  0.762258
3         0.945513  1.384156
4         0.854464  1.453768
...            ...       ...
1357     -0.260294  2.725673
1358     -0.256462  2.704930
```

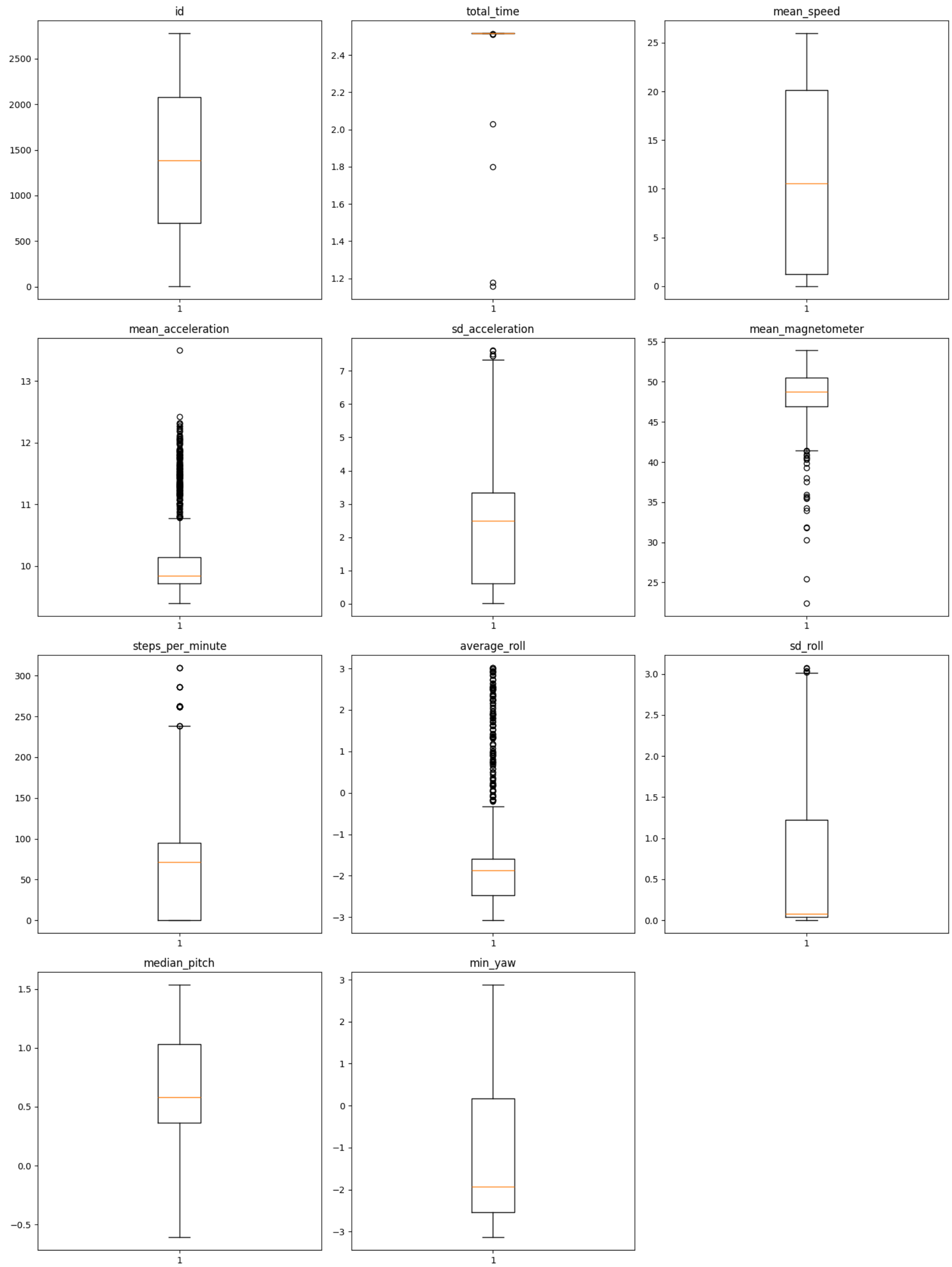
```
1359     -0.250087  2.725769
1360     -0.245037  2.577463
1361     -0.651176  1.984317

[1362 rows x 11 columns]
```

[ ]:
```
plot_boxplots(reduced_result)
```



We don't want to analyze activities shorter than 1 second, so we will remove them from the dataset.

[ ]:
```
time_min = 1
Y_train.drop(reduced_result[reduced_result['total_time'] < time_min].index,␣
↪inplace=True)
reduced_result.drop(reduced_result[reduced_result['total_time'] < time_min].
↪index, inplace=True)

plot_boxplots(reduced_result)
```



1.3 Models

Real labels from dataset.

[ ]:
```
real_labels = Y_train['act_type']
```

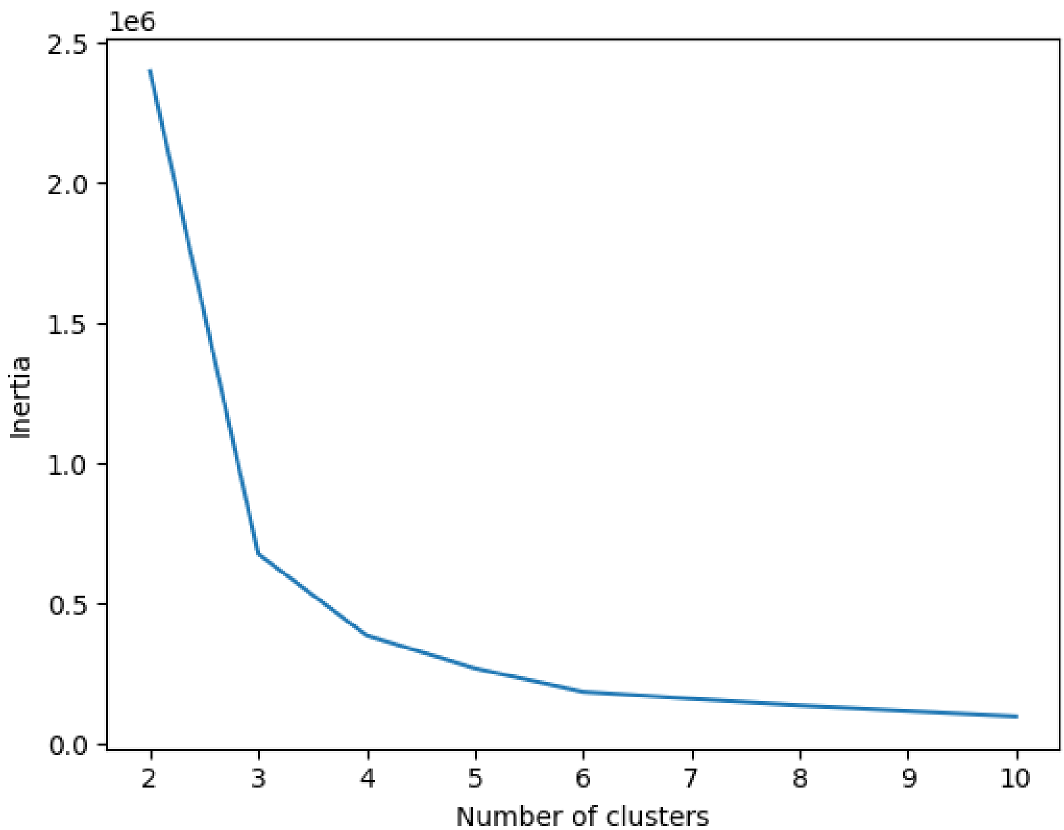
We can now drop id from our data frame.

[ ]:
```
reduced_result.drop(columns=['id'], inplace=True)
```

1.3.1 Elbow method

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

inertias = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(reduced_result)
    labels = kmeans.predict(reduced_result)
    print(f'Number of clusters: {i}, Inertia: {kmeans.inertia_}')
    inertias.append(kmeans.inertia_)

plt.plot(range(2, 11), inertias)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```

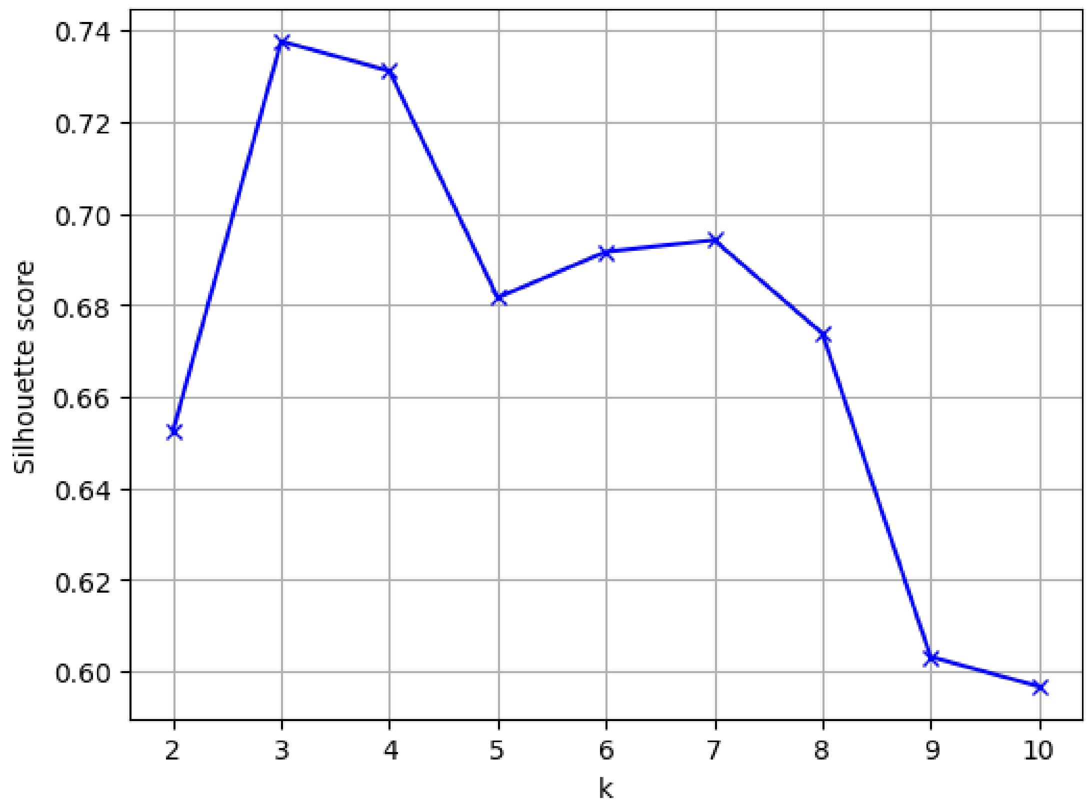
```
Number of clusters: 2, Inertia: 2398399.0172427148
Number of clusters: 3, Inertia: 676298.8075330282
Number of clusters: 4, Inertia: 386213.30186821765
Number of clusters: 5, Inertia: 269133.0470570539
Number of clusters: 6, Inertia: 184992.9019991527
Number of clusters: 7, Inertia: 160291.4550844947
Number of clusters: 8, Inertia: 136287.86009575395
Number of clusters: 9, Inertia: 116576.06052181394
Number of clusters: 10, Inertia: 97493.24676707006
```



1.3.2 Silhouette score

[ ]:
```
def count_clustering_scores(X, cluster_num, model, score_fun):
    if isinstance(cluster_num, int):
        cluster_num_iter = [cluster_num]
    else:
        cluster_num_iter = cluster_num

    scores = []
    for k in cluster_num_iter:
        model_instance = model(n_clusters=k,random_state=42)
        labels = model_instance.fit_predict(X)
        wcss = score_fun(X, labels)
        scores.append(wcss)

    if isinstance(cluster_num, int):
        return scores[0]
    else:
        return scores
```

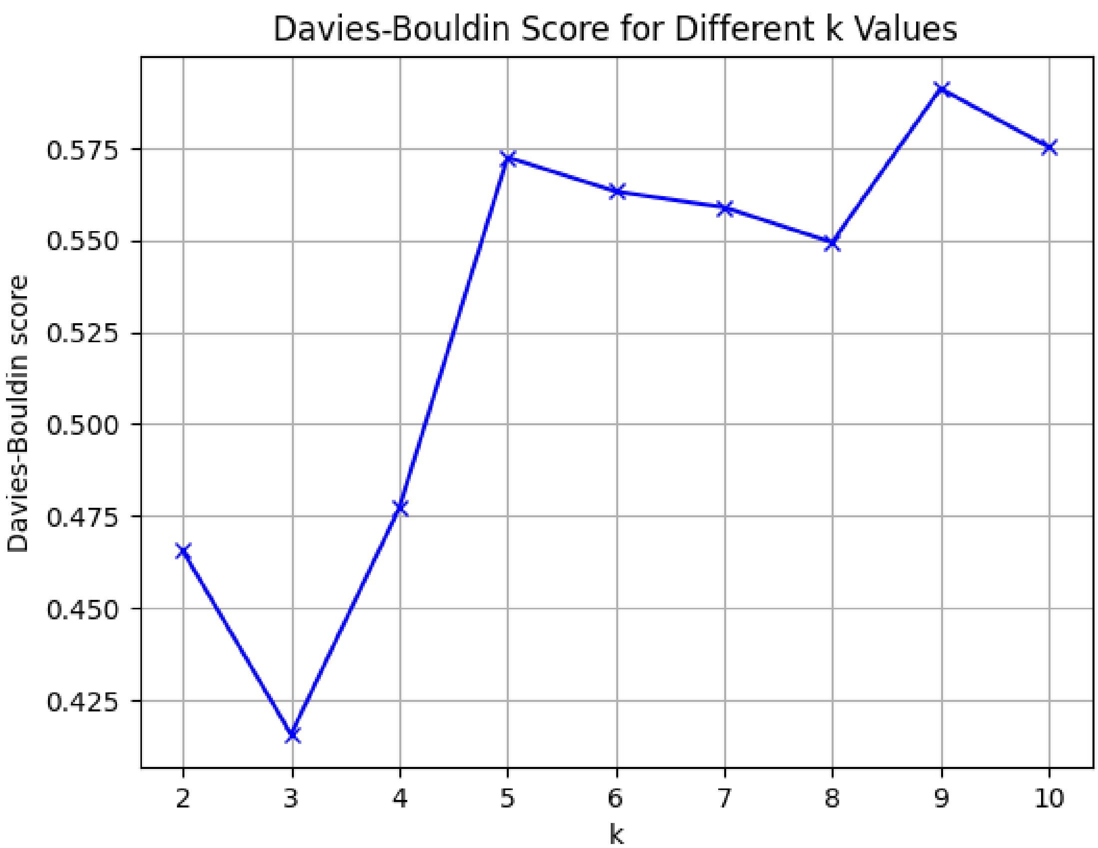
```
[ ]: cluster_num_seq = range(2, 11)
silhouette_vec = count_clustering_scores(reduced_result, cluster_num_seq,␣
 ↪KMeans, silhouette_score)
plt.plot(cluster_num_seq, silhouette_vec, 'bx-')
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.xticks(cluster_num_seq)
plt.grid(True)
plt.show()
```



1.3.3 Davies-Bouldin score

```
[ ]: cluster_num_seq = range(2, 11)
#davies_bouldin_vec = count_clustering_scores(df, cluster_num_seq, KMeans,␣
 ↪davies_bouldin_score)

davies_bouldin_vec = []
for k in cluster_num_seq:
    kmeans = KMeans(n_clusters=k,random_state=42)
    davies_bouldin_vec.append(davies_bouldin_score(reduced_result, kmeans.
 ↪fit_predict(reduced_result)))
```

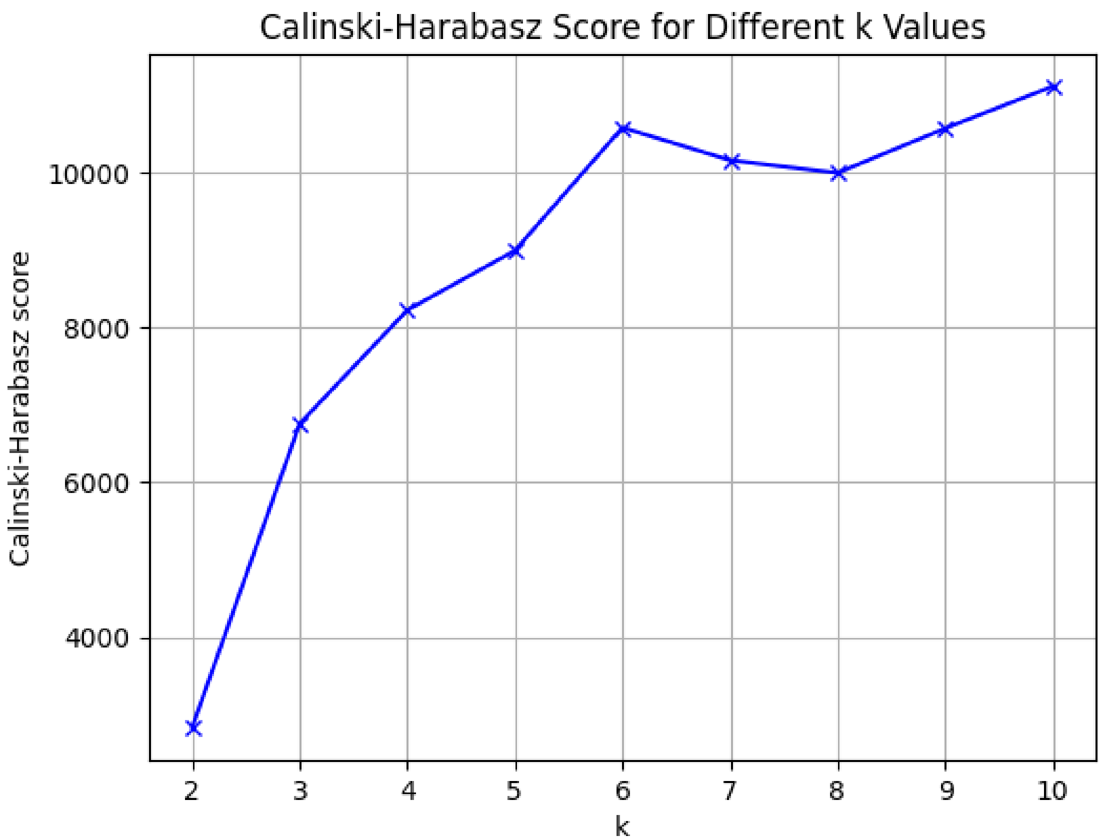
```
plt.plot(cluster_num_seq, davies_bouldin_vec, 'bx-')
plt.xlabel('k')
plt.ylabel('Davies-Bouldin score')
plt.title('Davies-Bouldin Score for Different k Values')
plt.xticks(cluster_num_seq)
plt.grid(True)
plt.show()
```



1.3.4 Caliski-Harabasz score

[ ]:
```
cluster_num_seq = range(2, 11)

calinski_harabasz_vec = count_clustering_scores(reduced_result,␣
 ↪cluster_num_seq, KMeans, calinski_harabasz_score)

plt.plot(cluster_num_seq, calinski_harabasz_vec, 'bx-')
plt.xlabel('k')
```

```
plt.ylabel('Calinski-Harabasz score')
plt.title('Calinski-Harabasz Score for Different k Values')
plt.xticks(cluster_num_seq)
plt.grid(True)
plt.show()
```



We set n_clusters = 3

[ ]:
```
n_clusters = 3
```

[ ]:
```
real_color_labels = ['yellow' if real_labels.iloc[i] == 0 else 'blue' if␣
 ↪real_labels.iloc[i] == 1 else 'green' for i in range(len(real_labels))]
real_color_labels
```

[ ]:
```
['yellow',
 'yellow',
 'yellow',
 'yellow',
 'yellow',
 'yellow',
 'yellow',
 'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
```

```
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'yellow',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
```

```
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
```

```
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
```

```
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
'blue',
```

```
'blue',
'blue',
'blue',
'blue',
'blue',
...]
```

1.3.5 Function to do KMeans clustering

[ ]:
```
from sklearn import metrics
def doKmeans(X, nclust=n_clusters, xaxis = 2, yaxis = 6, real = False, ␣
↪multidimensional = False, zaxis = 4):
    model = KMeans(nclust)
    model.fit(X)
    clust_labels = model.predict(X).flatten()
    centers = model.cluster_centers_

    color_labels = ['yellow' if clust_labels[i] == 0 else 'blue' if␣
↪clust_labels[i] == 1 else 'green' for i in range(len(clust_labels))]

    print(f"Model inertia: {model.inertia_}")
    print("Accuracy: ", np.mean(real_labels == clust_labels))
    print("Silhouette coefficient:" , silhouette_score(X, clust_labels))
    print("Davies Bouldin Score:" , davies_bouldin_score(X, clust_labels))
    print("Calinski Harabasz Score:" , calinski_harabasz_score(X, clust_labels))
    print()



    # i = 0
    # for label in real_labels:
    #     if label == 0:
    #         plt.scatter(X.iloc[i, 0], X.iloc[i, 1], marker='o',␣
↪c=color_labels[i], s=50)
    #     elif label == 1:
    #         plt.scatter(X.iloc[i, 0], X.iloc[i, 1], marker='s',␣
↪c=color_labels[i], s=50)
    #     elif label == 2:
    #         plt.scatter(X.iloc[i, 0], X.iloc[i, 1], marker='^',␣
↪c=color_labels[i], s=50)
    #     i += 1


    plt.scatter(X.iloc[:, xaxis], X.iloc[:, yaxis], marker='o', c=color_labels,␣
↪s=50, cmap='viridis')
```

```
    plt.scatter(centers[:, xaxis], centers[:, yaxis], c='red', s=200, alpha=0.
↪75, marker='X')

    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('K-means Clustering with Centroids')
    plt.show()

    if real:
        plt.scatter(X.iloc[:, xaxis], X.iloc[:, yaxis], marker='o',␣
↪c=real_color_labels, s=50, cmap='viridis')

        plt.scatter(centers[:, xaxis], centers[:, yaxis], c='red', s=200,␣
↪alpha=0.75, marker='X')

        plt.xlabel('Feature 1')
        plt.ylabel('Feature 2')
        plt.title('K-means Clustering with Centroids')
        plt.show()

    if multidimensional:
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')

        # Scatter plot for data points
        ax.scatter(X.iloc[:, xaxis], X.iloc[:, yaxis], X.iloc[:, zaxis],␣
↪c=color_labels, s=50, cmap='viridis')

        # Scatter plot for centroids
        ax.scatter(centers[:, xaxis], centers[:, yaxis], centers[:, zaxis],␣
↪c='red', s=200, alpha=0.75, marker='X')

        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_zlabel('Feature 3')
        ax.set_title('K-means Clustering with Centroids')

        plt.show()

    if multidimensional & real:
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')

        # Scatter plot for data points
        ax.scatter(X.iloc[:, xaxis], X.iloc[:, yaxis], X.iloc[:, zaxis],␣
↪c=real_color_labels, s=50, cmap='viridis')
```

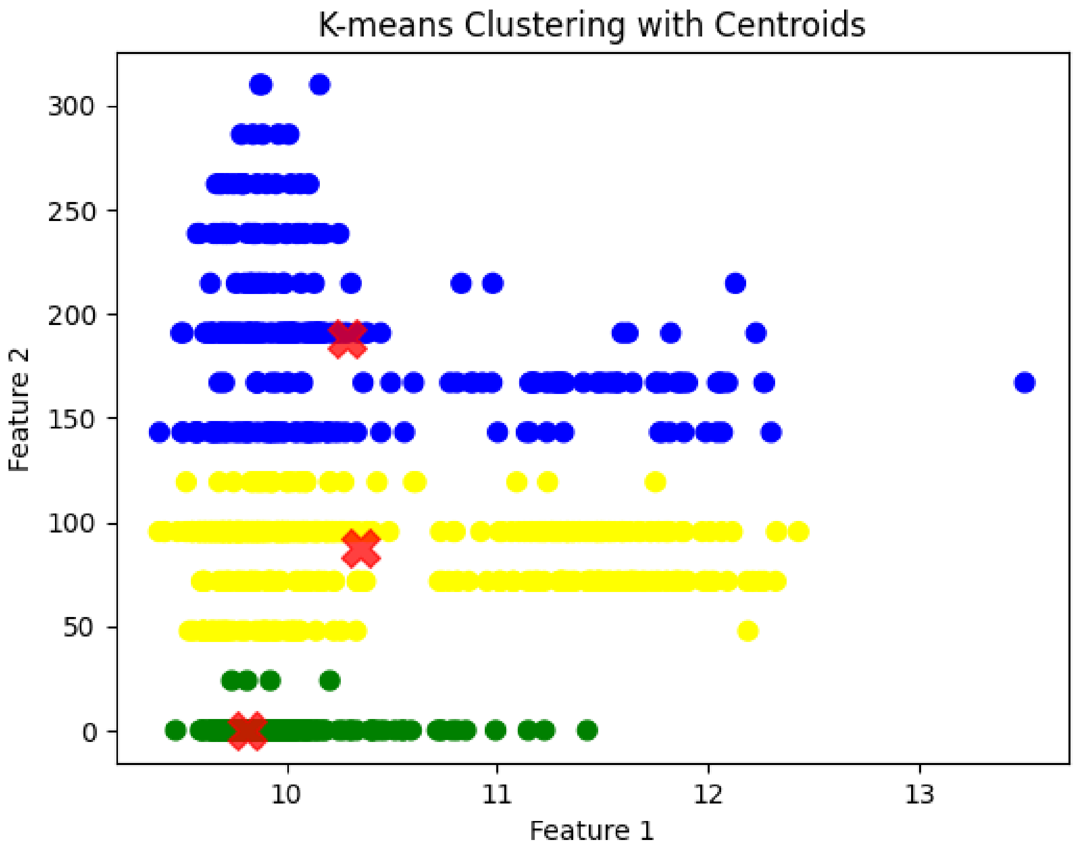
```
        # Scatter plot for centroids
        ax.scatter(centers[:, xaxis], centers[:, yaxis], centers[:, zaxis],␣
↪c='red', s=200, alpha=0.75, marker='X')

        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_zlabel('Feature 3')
        ax.set_title('K-means Clustering with Centroids')

        plt.show()

    return clust_labels
```

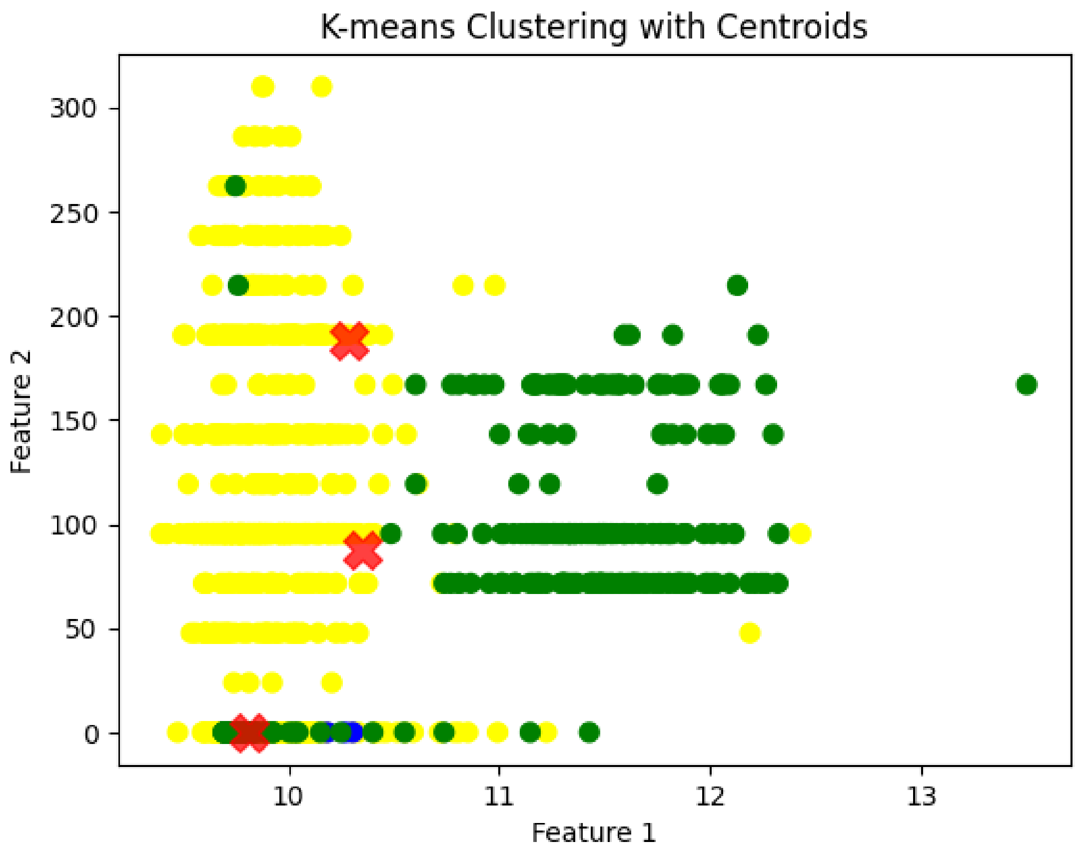
1.3.6 Tests

Basic

[ ]:
```
x = reduced_result.copy()
clust_labels = doKmeans(x, n_clusters, 2, 5, True)
```

```
Model inertia: 676298.8075330282
Accuracy:  0.3426470588235294
Silhouette coefficient: 0.7375605939938716
Davies Bouldin Score: 0.41555828381883003
Calinski Harabasz Score: 6756.265618212855
```

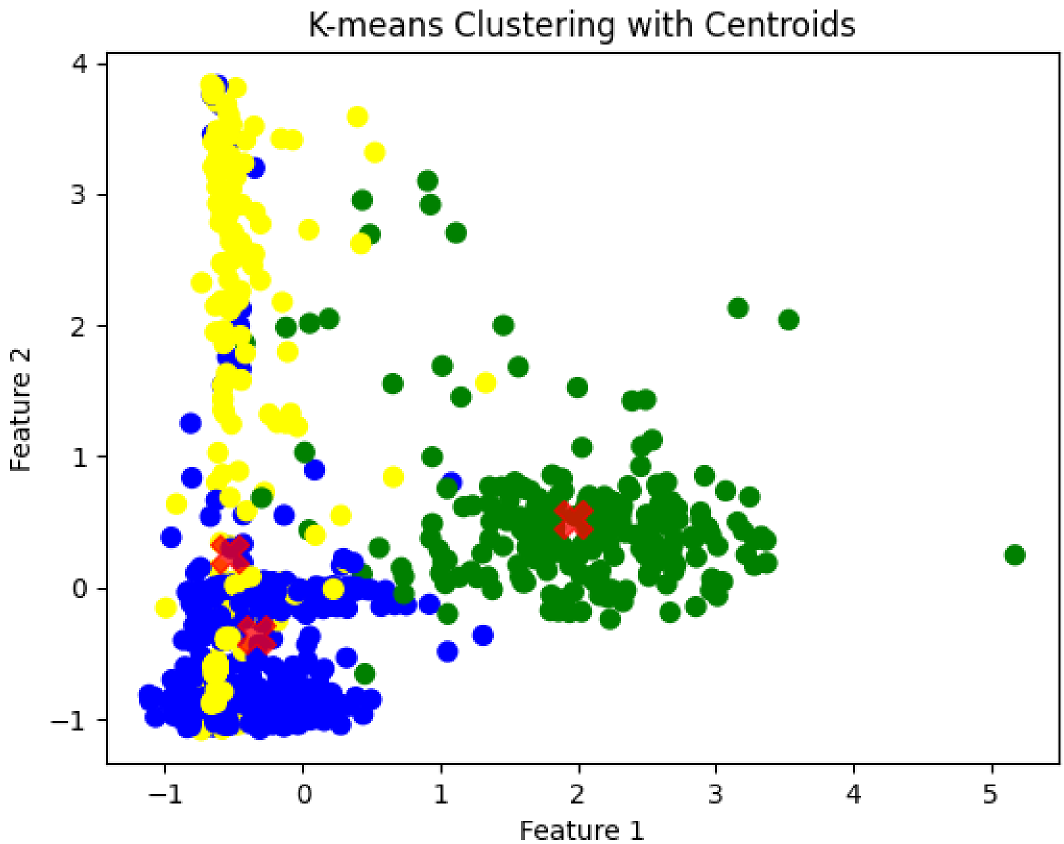

K-means Clustering with Centroids
Feature 2
300
250
200
150
100
50
0
10
11
12
13
Feature 1

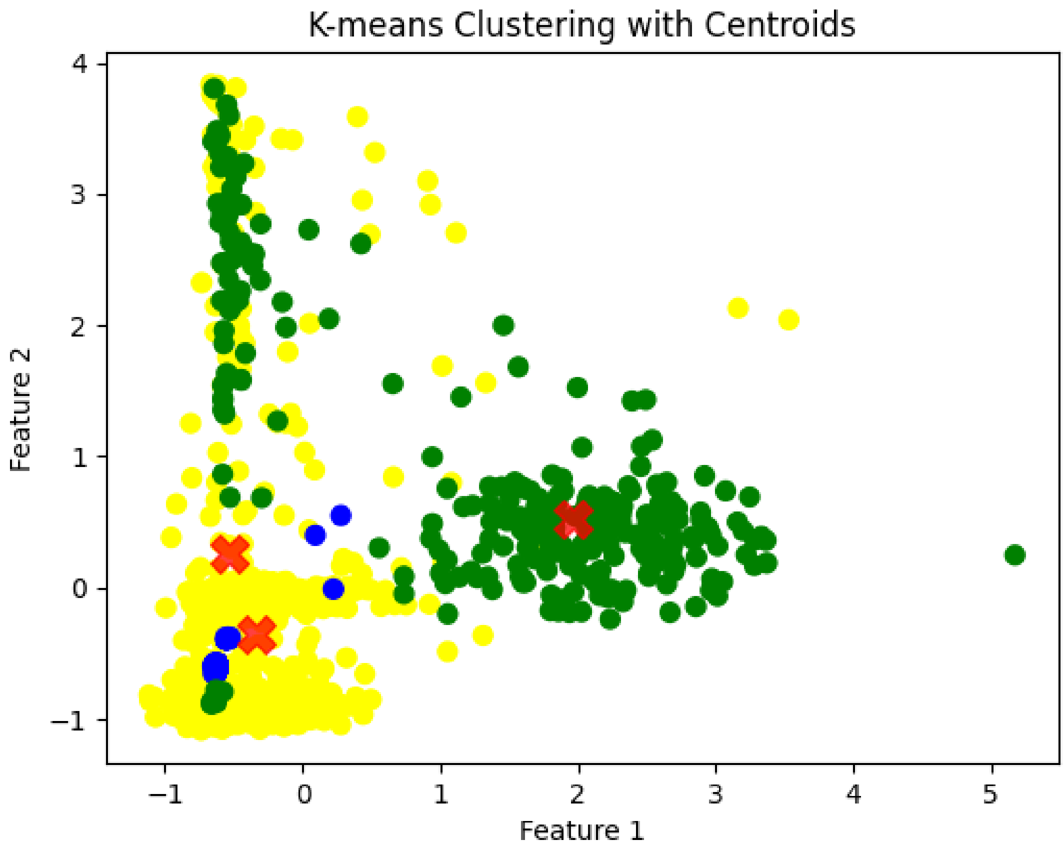


Standarization

[ ]:
```
x = reduced_result.copy()
scaler = StandardScaler()
x = scaler.fit_transform(x)
x = pd.DataFrame(x, columns=reduced_result.columns)
clust_labels = doKmeans(x, n_clusters, 2, 6, True)
```

```
Model inertia: 7907.732464791002
Accuracy:  0.2647058823529412
Silhouette coefficient: 0.3555755904929665
Davies Bouldin Score: 1.2709793457218654
Calinski Harabasz Score: 488.40847105484073
```

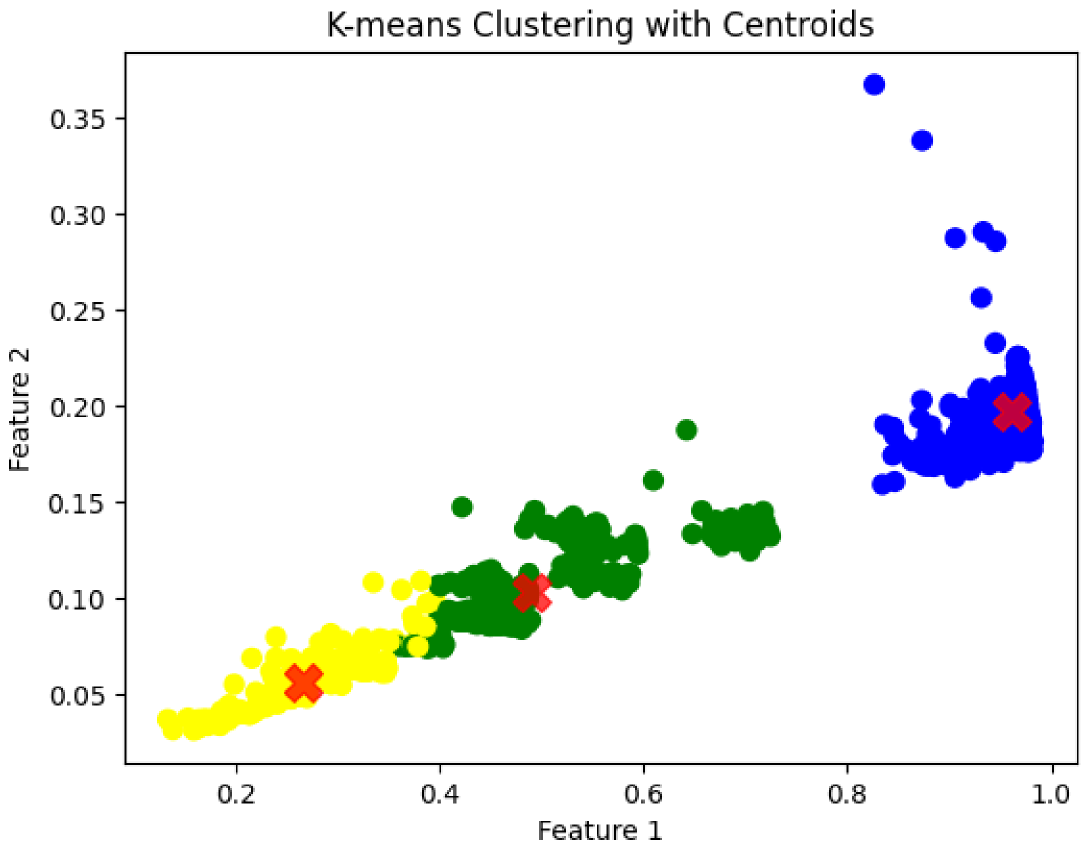

K-means Clustering with Centroids
Feature 2
Feature 1
4
3
2
1
0
−1
−1
0
1
2
3
4
5

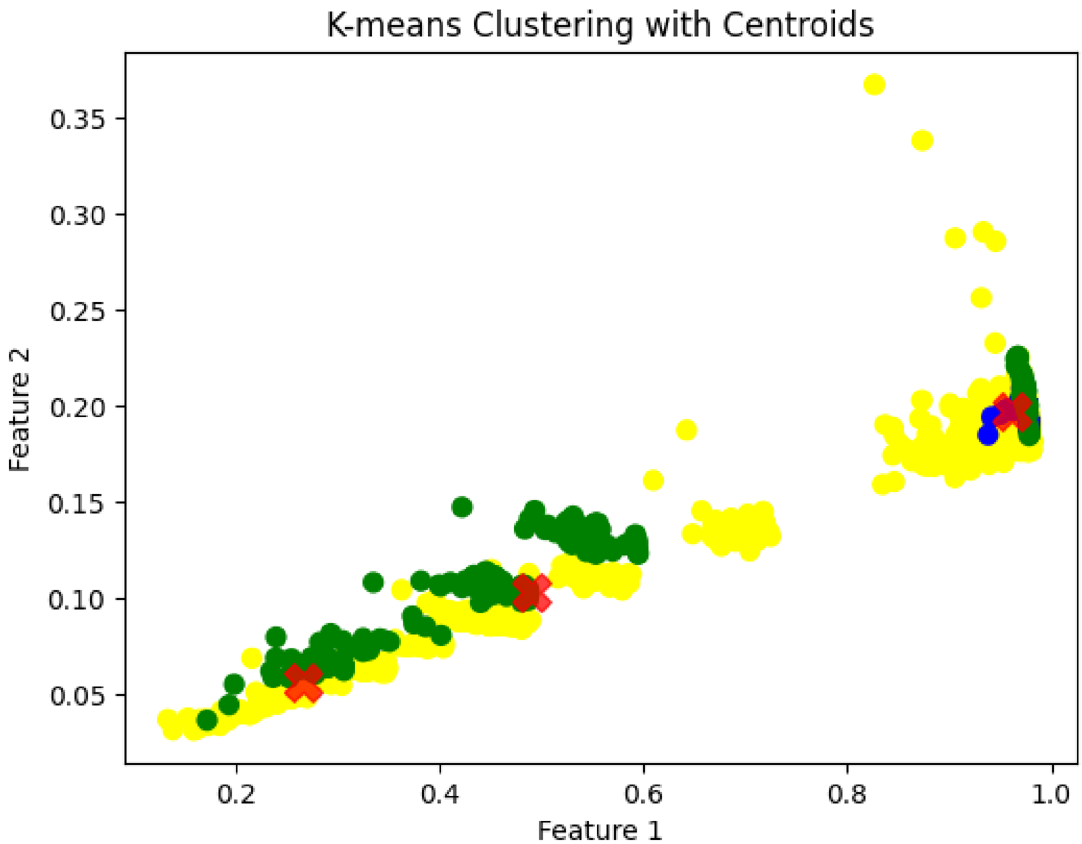


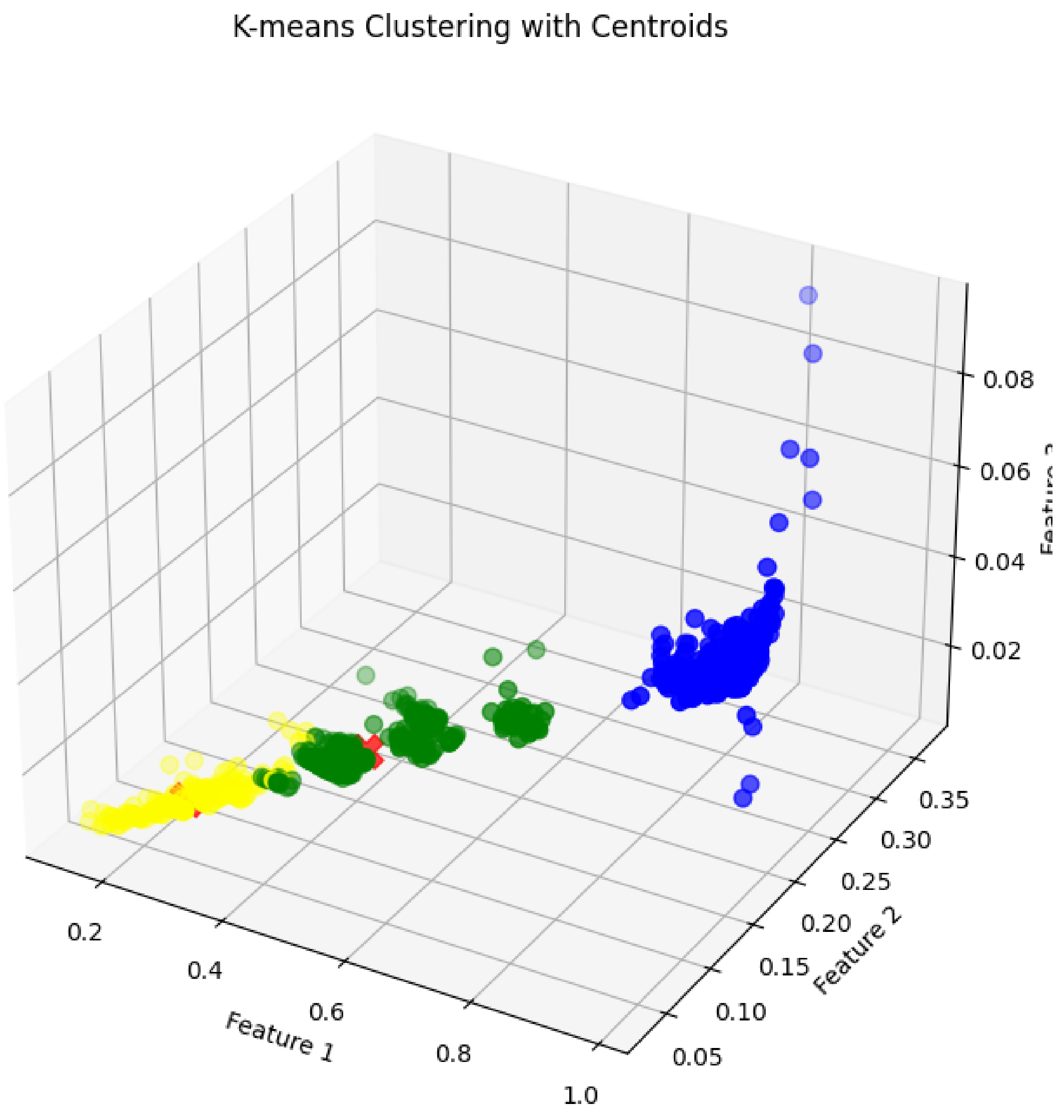
Normalization

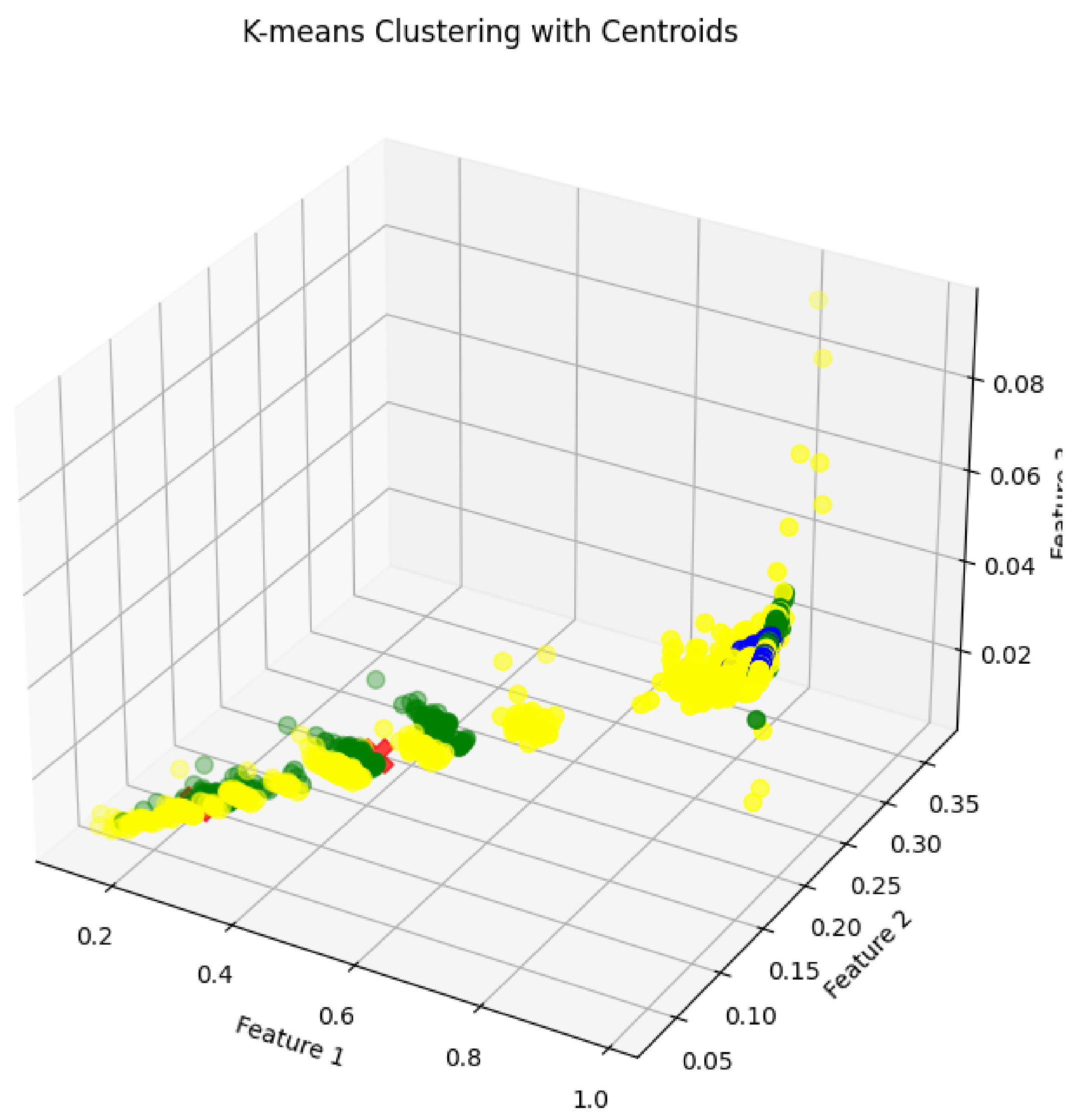
```
x = reduced_result.copy()
normalizer = Normalizer()
x = normalizer.fit_transform(x)
x = pd.DataFrame(x, columns=reduced_result.columns)
clust_labels_norm = doKmeans(x, n_clusters, 4, 2, True, True, 0)
```

```
Model inertia: 23.77406388527578
Accuracy:  0.4485294117647059
Silhouette coefficient: 0.6362474478652769
Davies Bouldin Score: 0.5470337447761109
Calinski Harabasz Score: 10553.589103896726
```


K-means Clustering with Centroids
Feature 2
Feature 1
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.2
0.4
0.6
0.8
1.0


K-means Clustering with Centroids
Feature 2
Feature 1
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.2
0.4
0.6
0.8
1.0


K-means Clustering with Centroids
0.02
0.04
0.06
0.08
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Feature 2
0.2
0.4
0.6
0.8
1.0
Feature 1

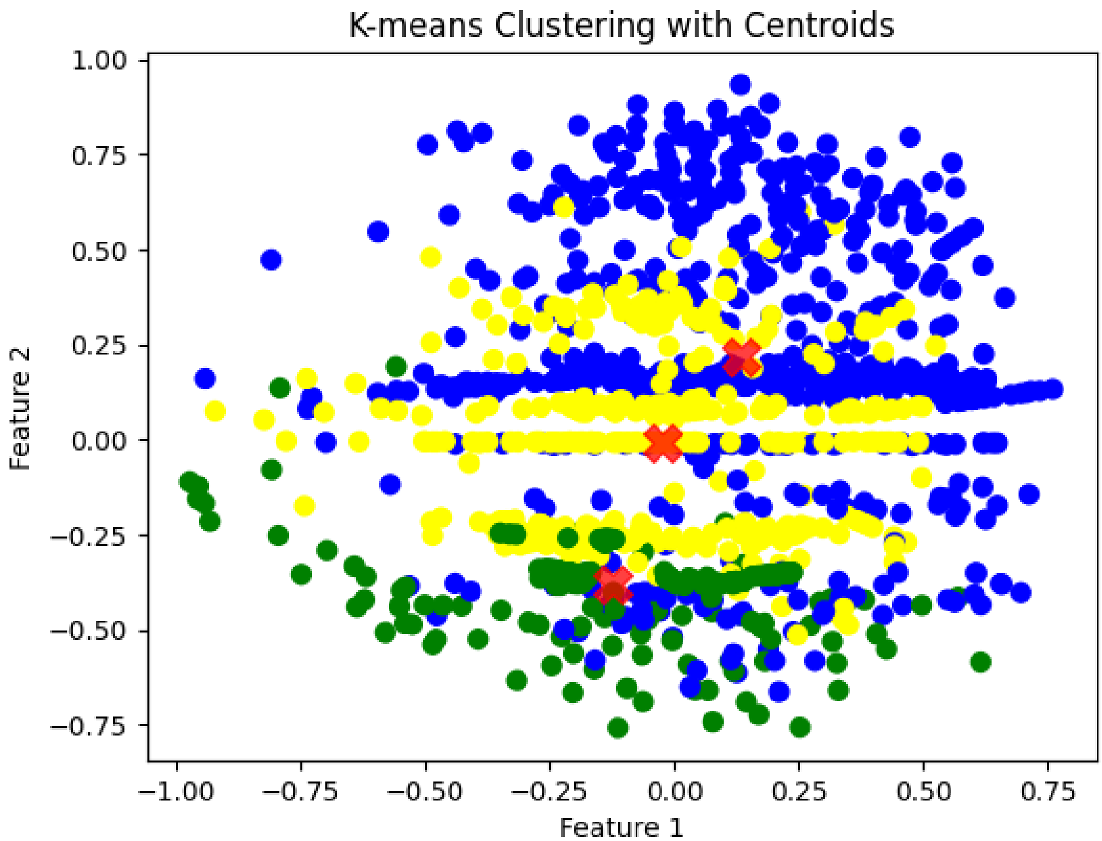


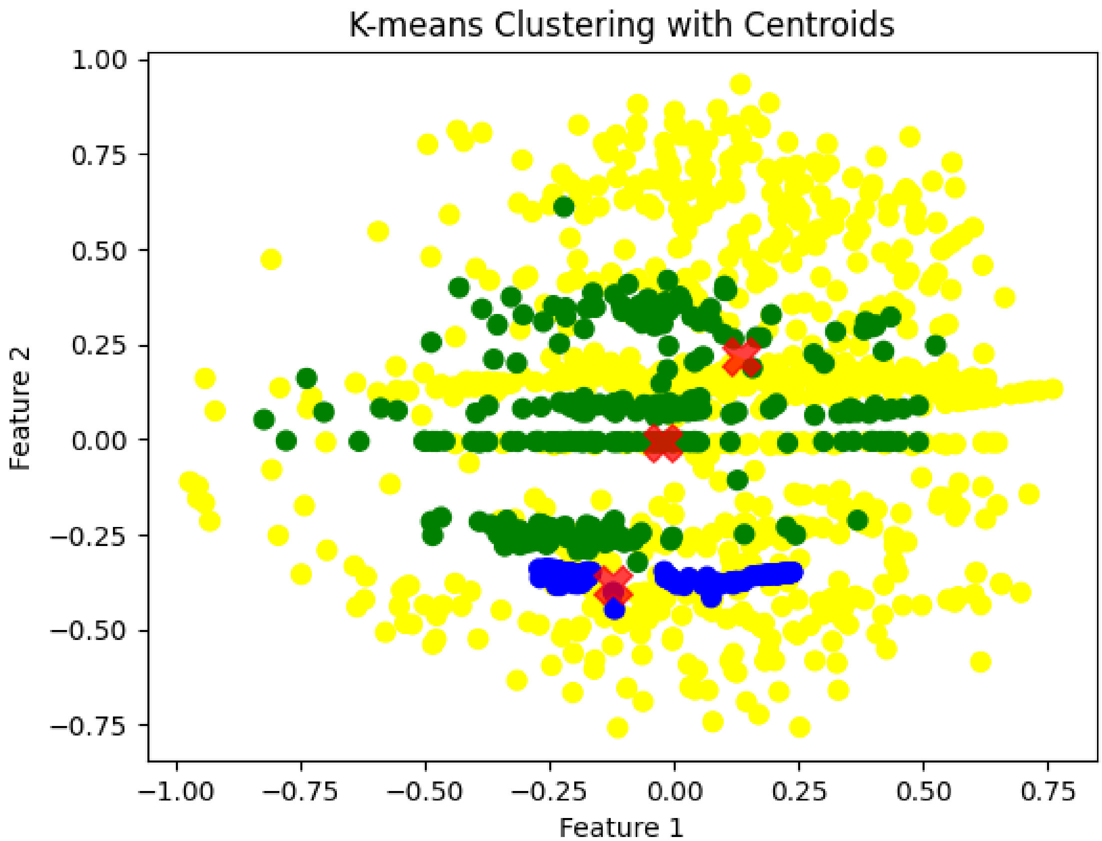
Normalization + standarization

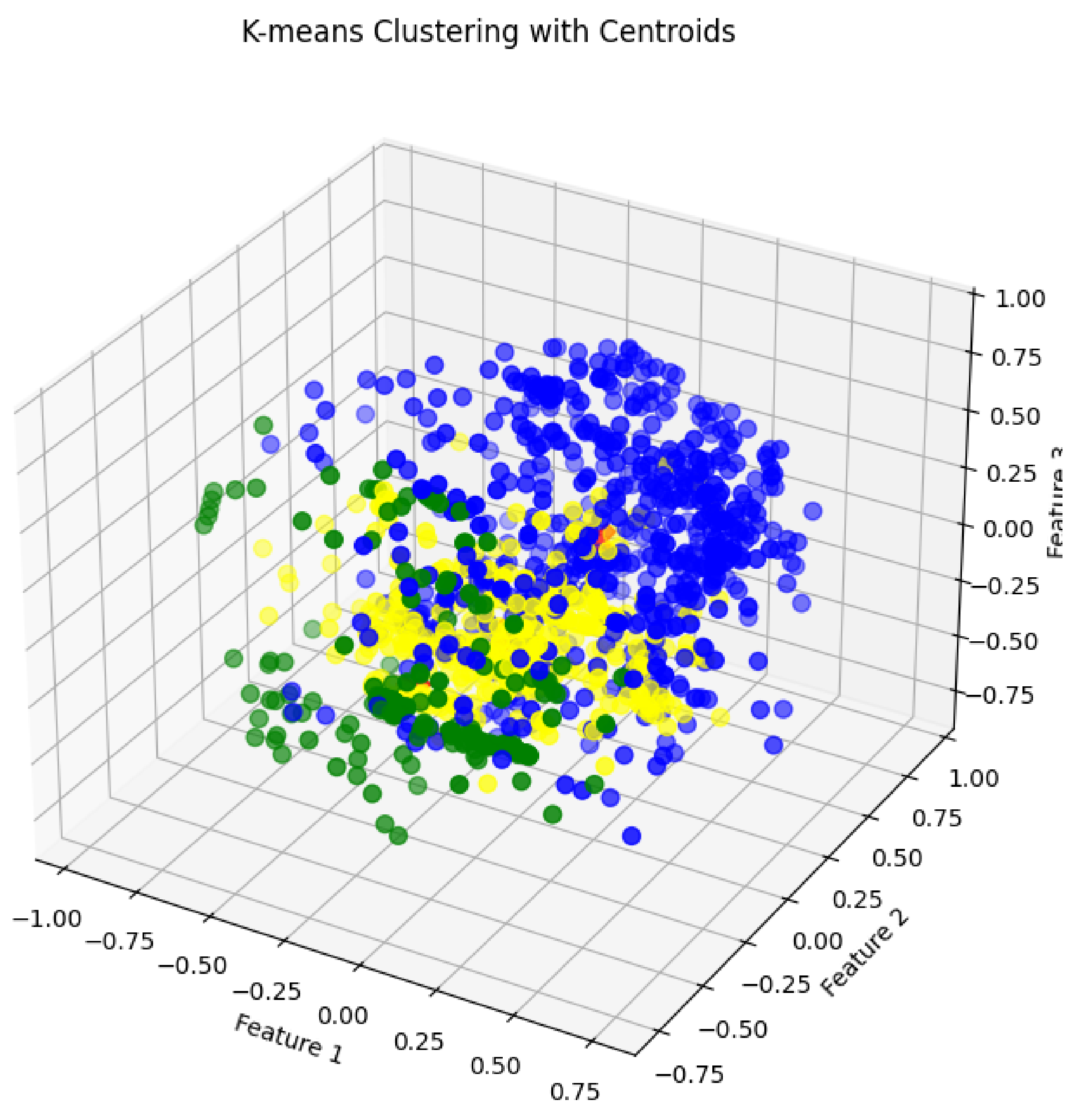
[ ]:
```
x = reduced_result.copy()
normalizer = Normalizer()
scaler = StandardScaler()
x = scaler.fit_transform(x)
x = normalizer.fit_transform(x)
x = pd.DataFrame(x, columns=reduced_result.columns)
clust_labels = doKmeans(x, n_clusters, 4, 5, True, True, 9)
```

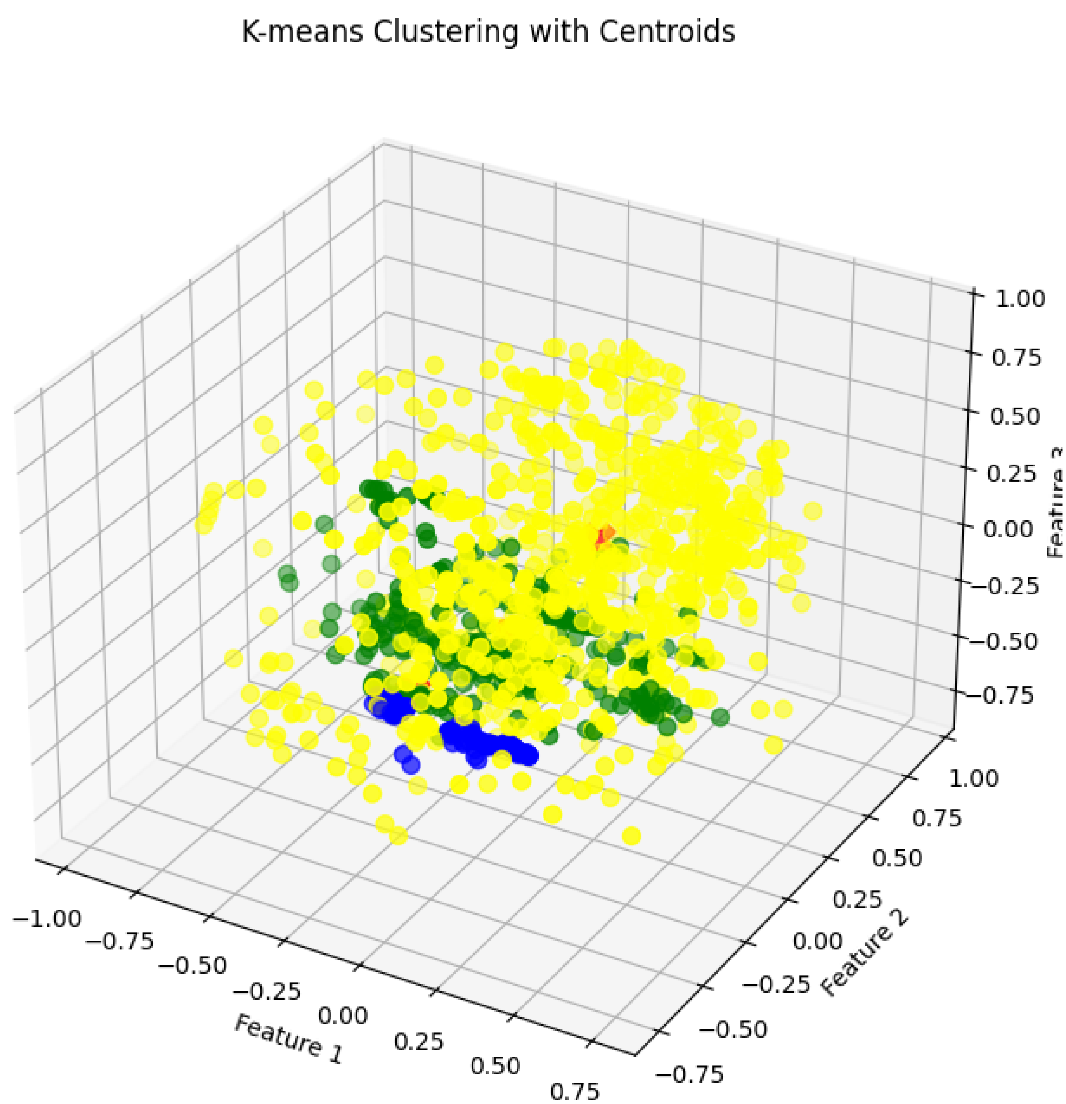
```
Model inertia: 711.9654535180476
Accuracy:  0.0838235294117647
Silhouette coefficient: 0.35894634377456575
Davies Bouldin Score: 1.2135614305355047
```

```
Calinski Harabasz Score: 597.543555340724
```




K-means Clustering with Centroids
Feature 2
Feature 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75

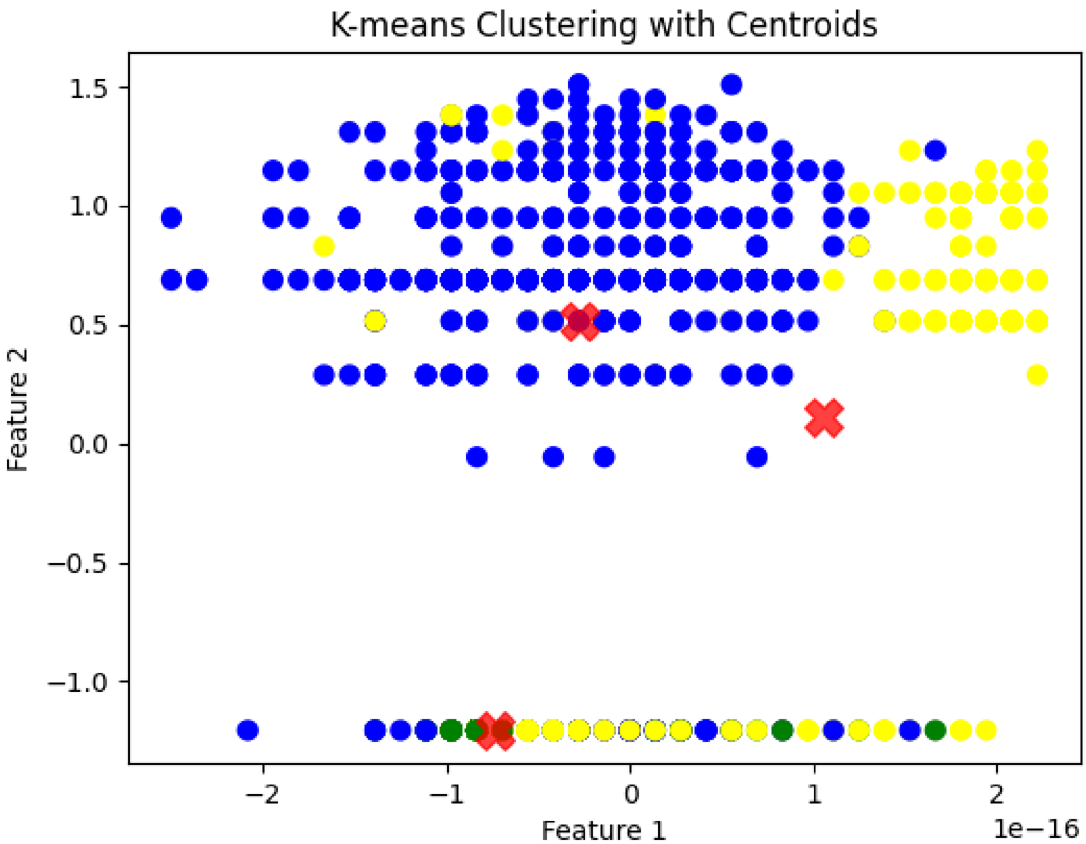

K-means Clustering with Centroids
Feature 1
Feature 2
Feature 3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

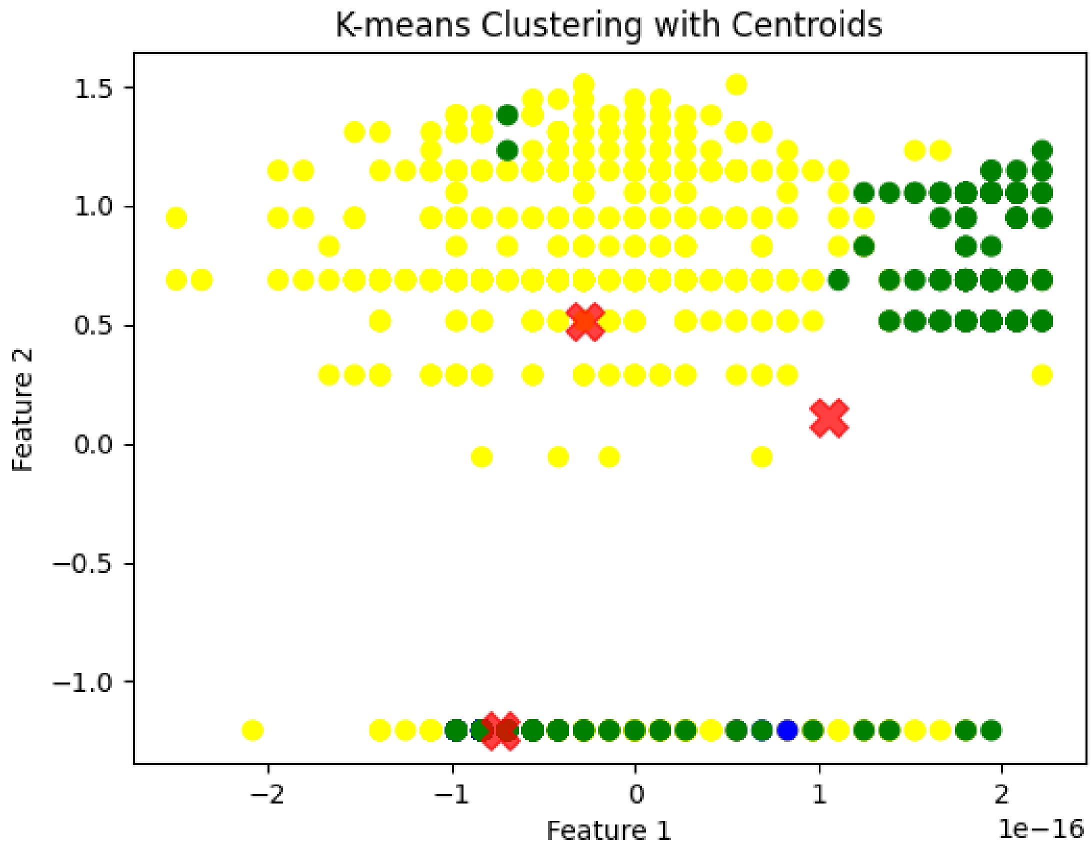


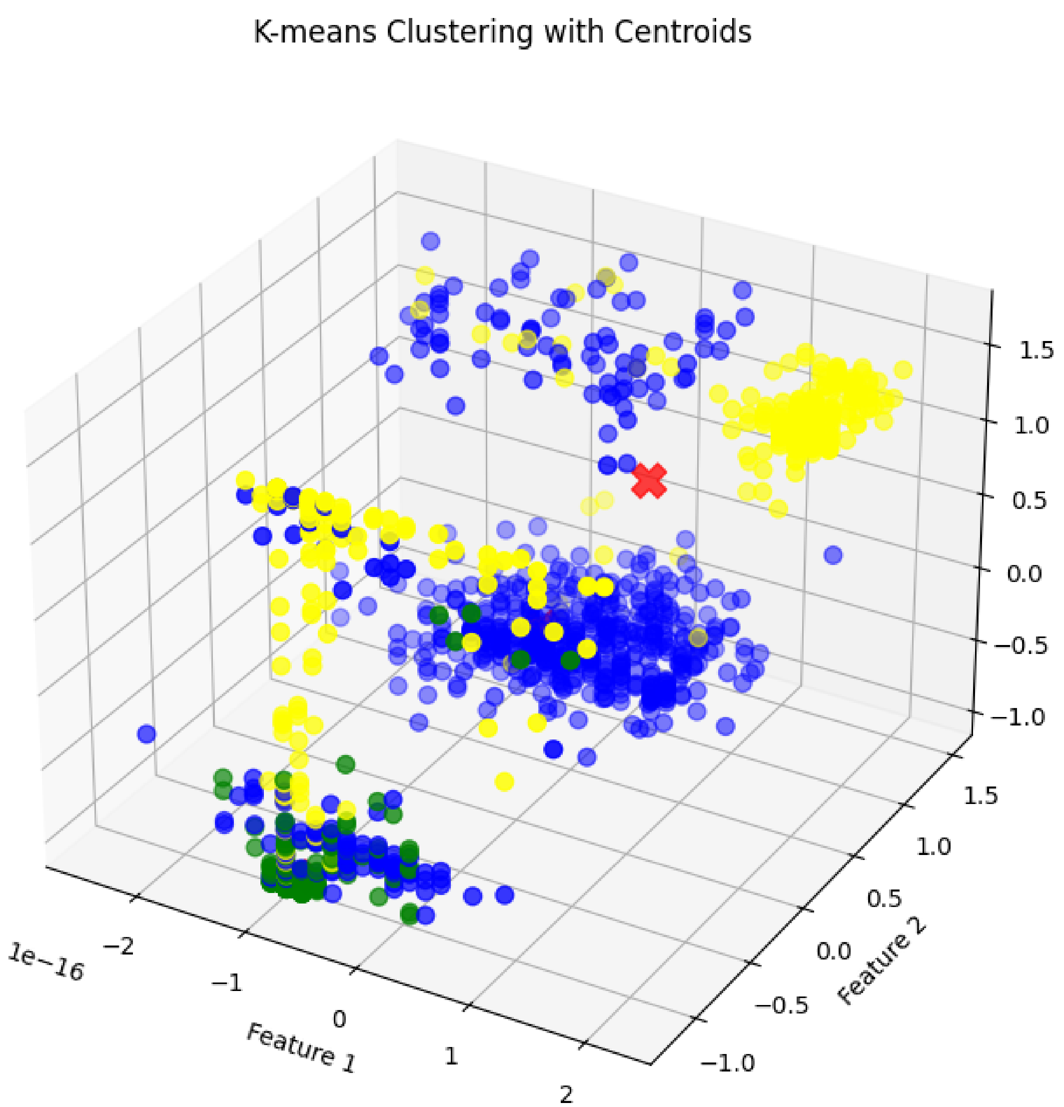
PowerTransformer

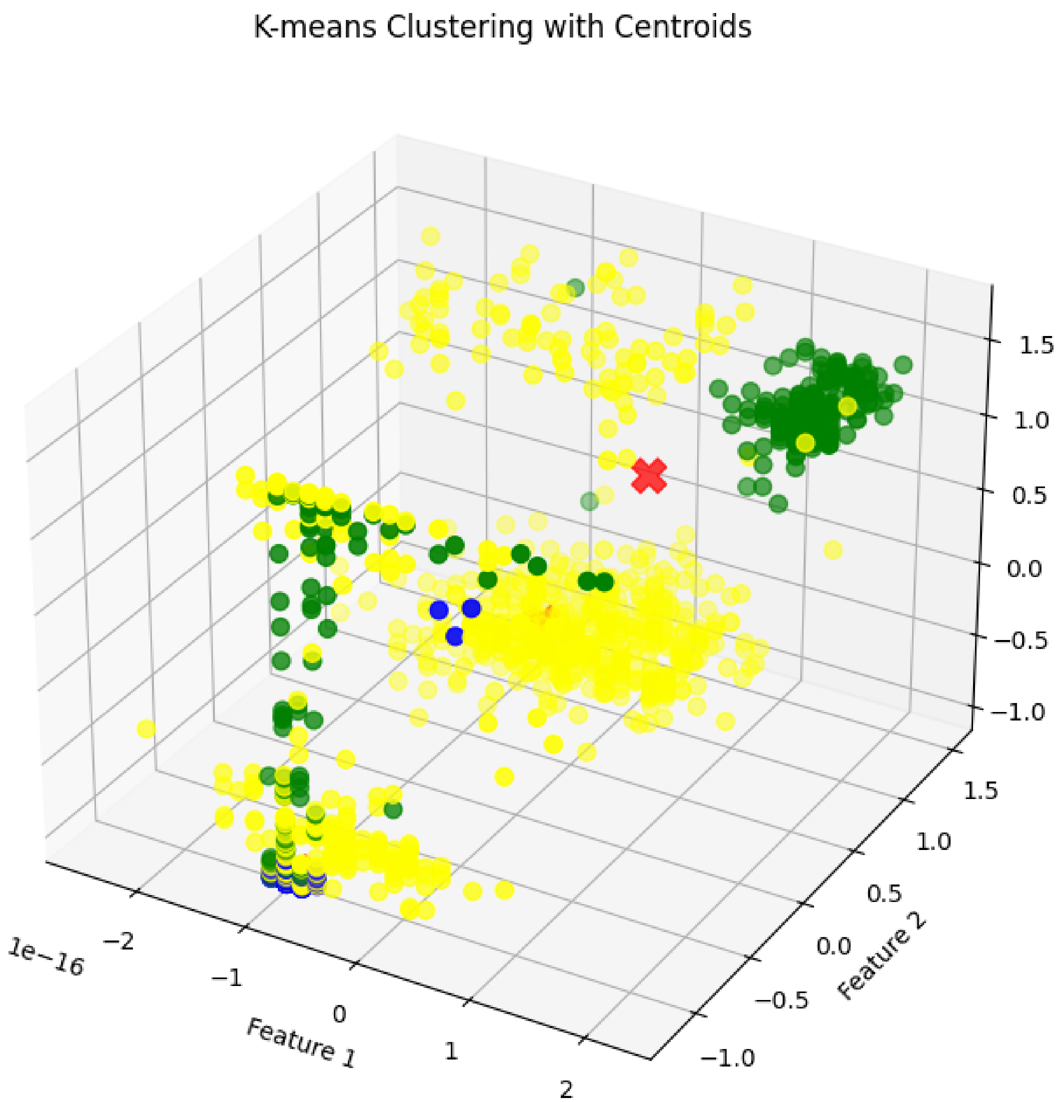
```
[ ]: x = reduced_result.copy()
powerTransformer = PowerTransformer()
x = powerTransformer.fit_transform(x)
x = pd.DataFrame(x, columns=reduced_result.columns)
clust_labels = doKmeans(x, n_clusters, 2, 5, True, True, 7)
```

```
Model inertia: 6723.133845380107
Accuracy:  0.061764705882352944
Silhouette coefficient: 0.3848607168256301
Davies Bouldin Score: 1.152506331790146
Calinski Harabasz Score: 556.7632256022667
```


K-means Clustering with Centroids
Feature 2
Feature 1
1e−16
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2


K-means Clustering with Centroids
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−2
−1
0
1
2
1e−16

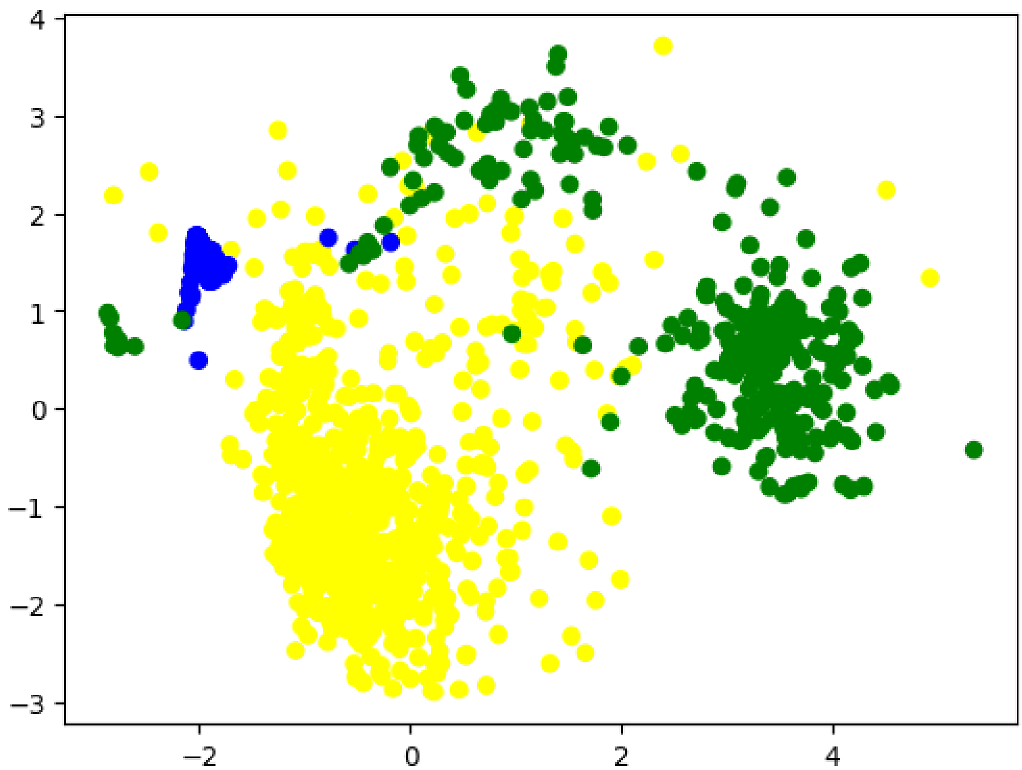

K-means Clustering with Centroids
1e−16
−2
−1
0
1
2
Feature 1
−1.0
−0.5
0.0
0.5
1.0
1.5
Feature 2
1.5
1.0
0.5
0.0
−0.5
−1.0



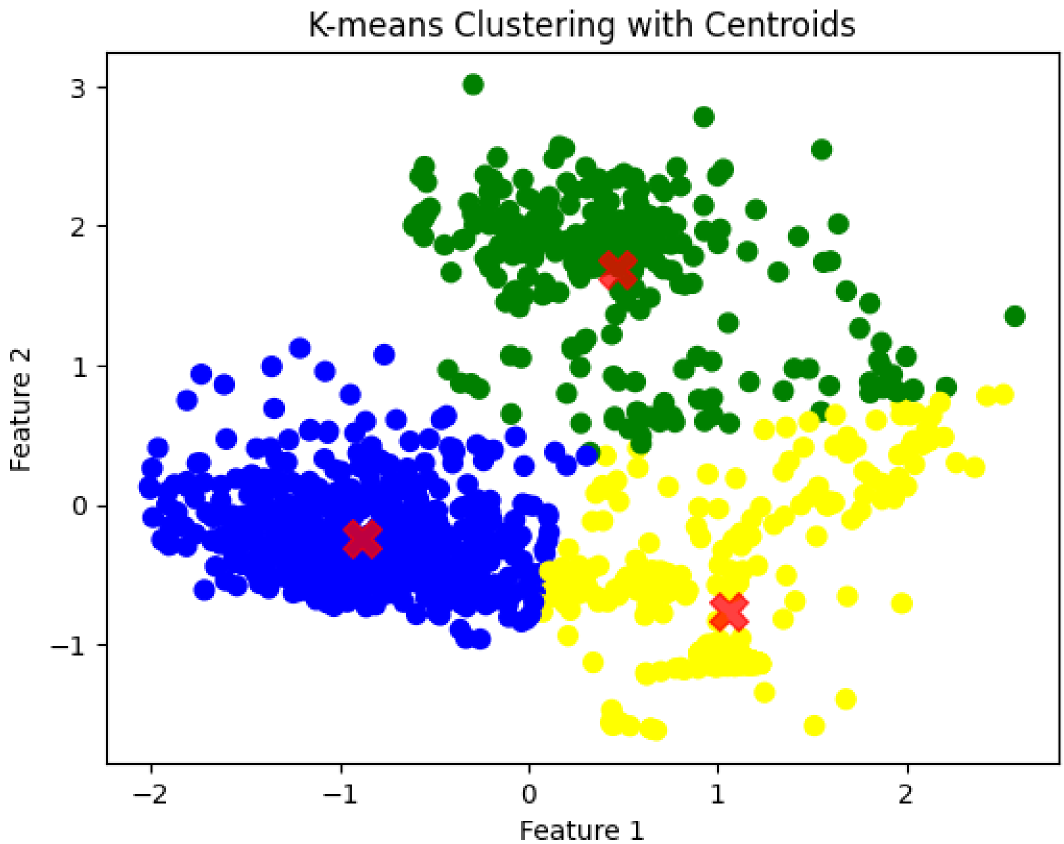
1.3.7 PCA

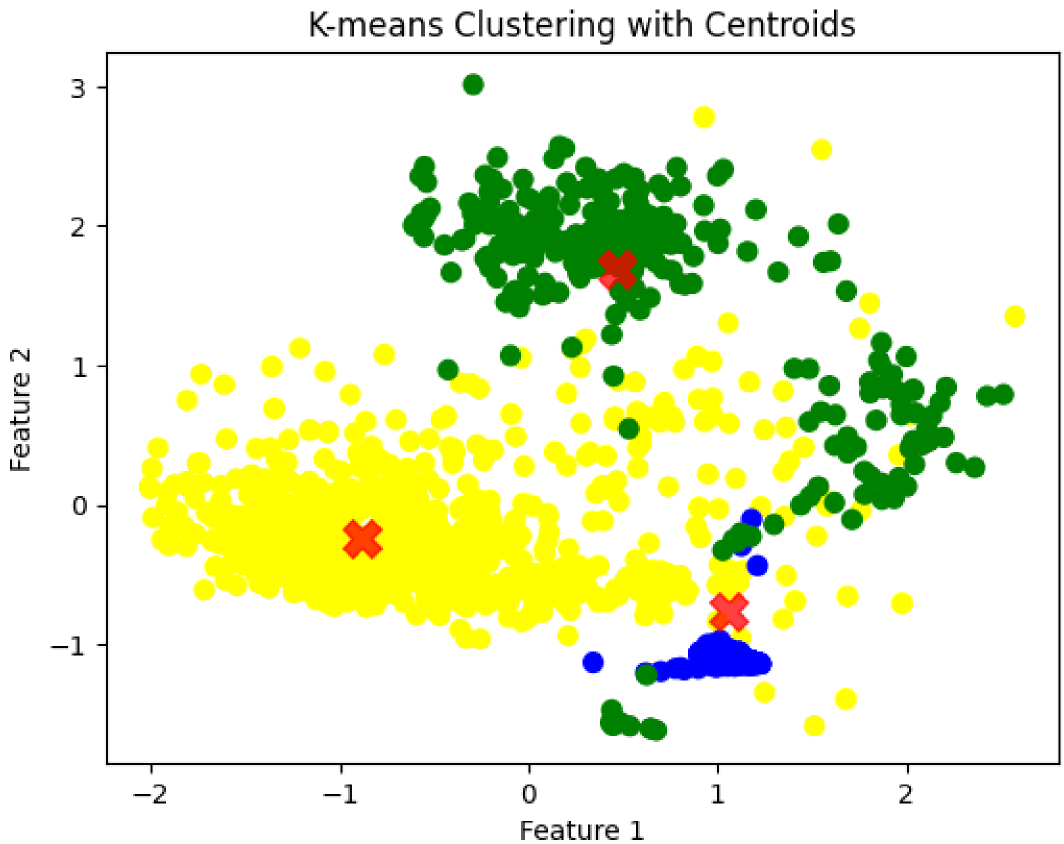
[ ]:
```
scaler = StandardScaler()
x = reduced_result.copy()
x = scaler.fit_transform(x)

model = PCA(n_components=2)
transformed = model.fit_transform(x)
xs = transformed[:,0]
ys = transformed[:,1]
x = pd.DataFrame(transformed)
scaler2 = StandardScaler()
```

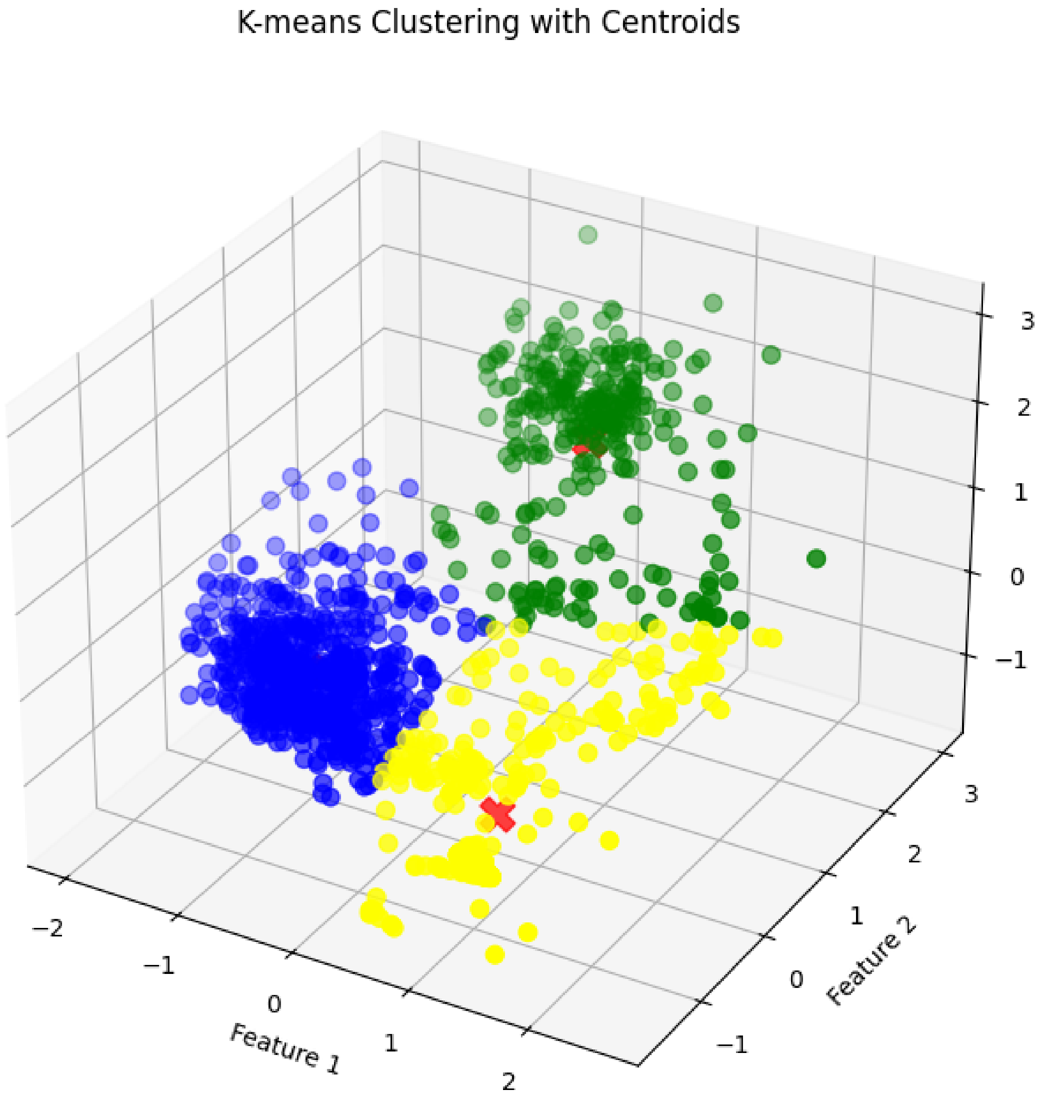
```python
x = scaler2.fit_transform(x)
x = pd.DataFrame(x)
plt.scatter(xs,ys,c=real_color_labels)
plt.show()
clust_labels = doKmeans(x, n_clusters, 1, 0, True, True, 0)
```

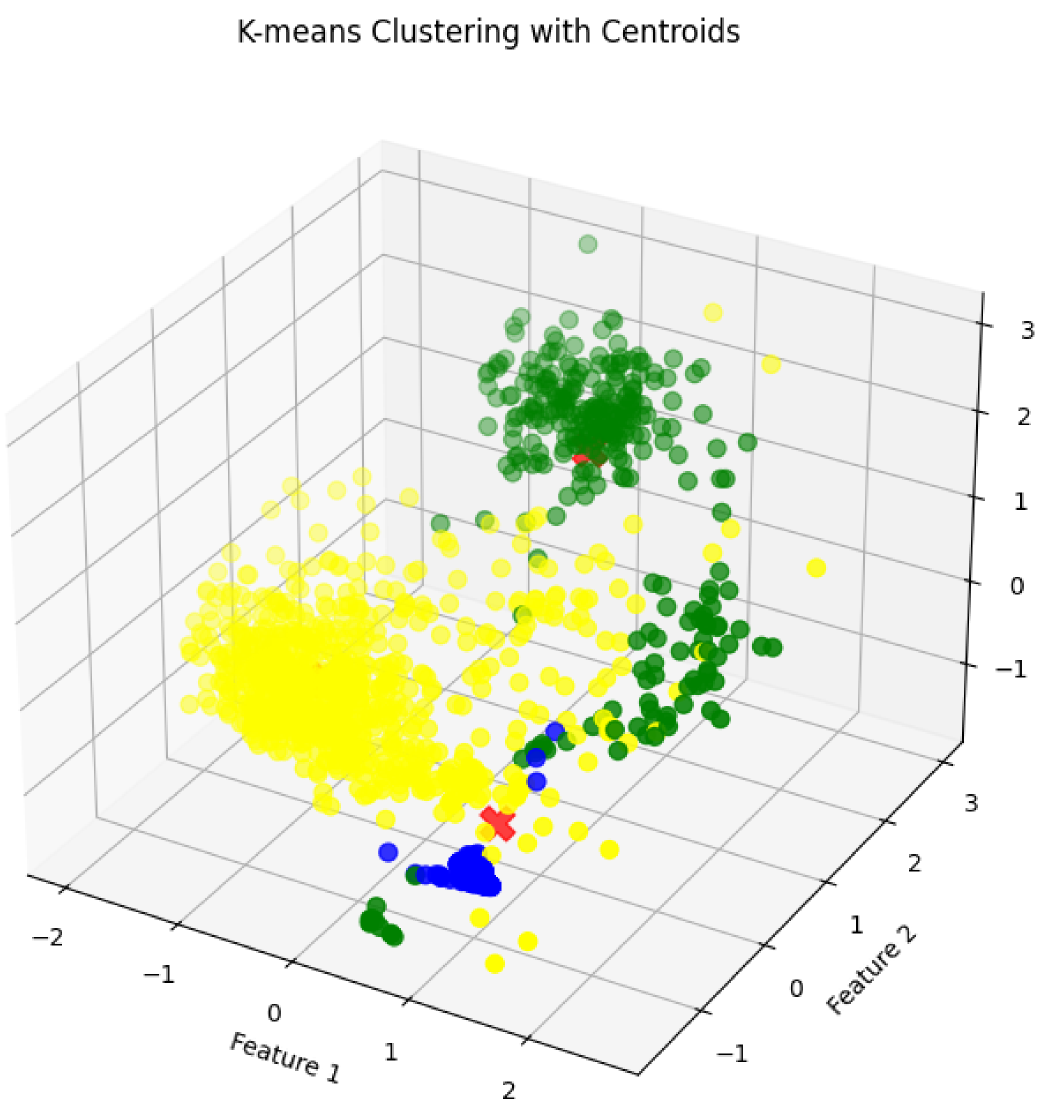



```
Model inertia: 587.3023512865127
Accuracy:  0.26544117647058824
Silhouette coefficient: 0.6103824186962272
Davies Bouldin Score: 0.5224255985188533
Calinski Harabasz Score: 2463.867804176675
```


K-means Clustering with Centroids
Feature 1
Feature 2
−2
−1
0
1
2
3


K-means Clustering with Centroids
Feature 2
Feature 1
3
2
1
0
−1
−2
−1
0
1
2

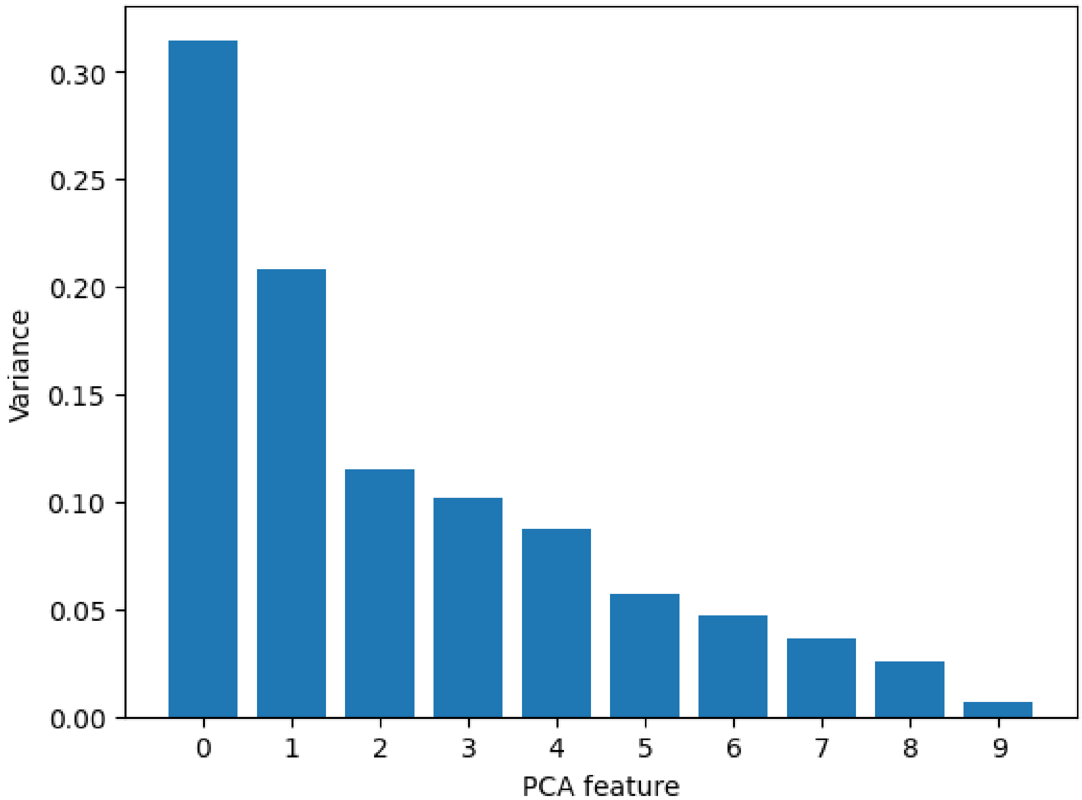

K-means Clustering with Centroids
Feature 1
Feature 2
−2
−1
0
1
2
3



1.3.8 Explained variance for PCA

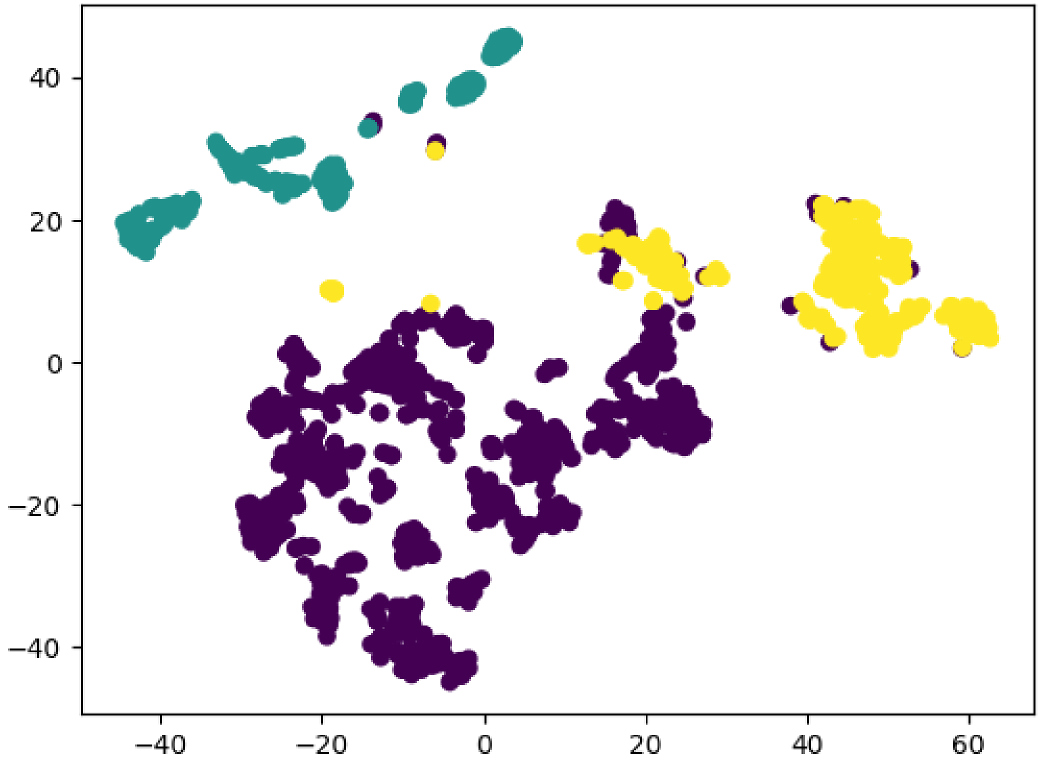
[ ]:
```
x = reduced_result.copy()
scaler = StandardScaler()

x = scaler.fit_transform(x)
model = PCA()
model.fit(x)

features = range(model.n_components_)
plt.bar(features, model.explained_variance_ratio_)
```

```
plt.xlabel('PCA feature')
plt.ylabel('Variance')
plt.xticks(features)
plt.show()
```



1.3.9 TSNE

[ ]:
```
scaler = StandardScaler()


x = reduced_result.copy()
x = scaler.fit_transform(x)

model = TSNE(learning_rate=100)
transformed = model.fit_transform(x)
xs = transformed[:,0]
ys = transformed[:,1]
plt.scatter(xs,ys,c=real_labels)
plt.show()
```



1.4 Other models

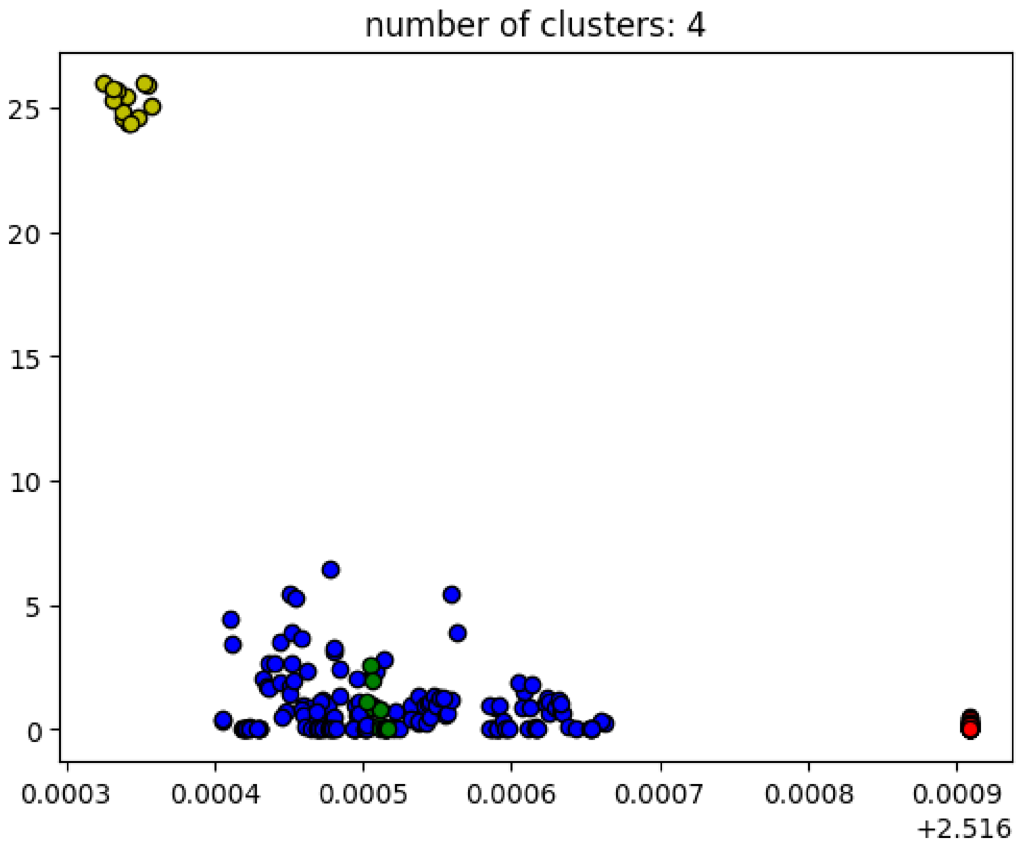
1.4.1 DBSCAN

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
```

[ ]:
```
X = reduced_result.copy()
normalizer = Normalizer()
scaler = StandardScaler()
x = normalizer.fit_transform(x)
x = scaler.fit_transform(X)
db = DBSCAN(eps=0.3, min_samples=10).fit(x)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
```

```python
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# Plot result

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = ['y', 'b', 'g', 'r']
print(colors)
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = 'k'

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy.iloc[:, 0], xy.iloc[:, 1], 'o', markerfacecolor=col,
             markeredgecolor='k',
             markersize=6)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy.iloc[:, 0], xy.iloc[:, 1], 'o', markerfacecolor=col,
             markeredgecolor='k',
             markersize=6)

plt.title('number of clusters: %d' % n_clusters_)
plt.show()
```

```
['y', 'b', 'g', 'r']
```



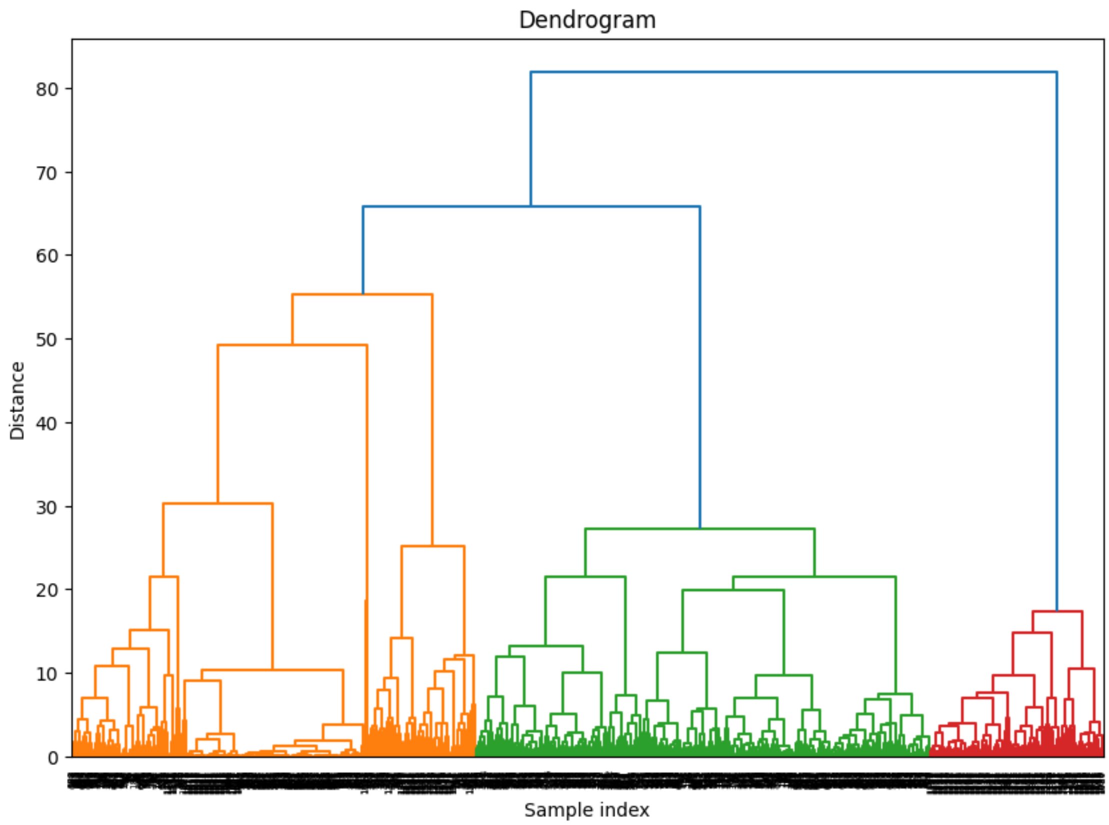
[ ]:
```
# evaluation metrics
sc = metrics.silhouette_score(X, labels)
print("Silhouette Coefficient:%0.2f" % sc)
```

```
Silhouette Coefficient:-0.42
```

1.4.2 Agglomerative Hierarchical Clustering

[ ]:
```
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
x = reduced_result.copy()
scaler = StandardScaler()
df_scaled = scaler.fit_transform(x)

# Przeprowadzenie hierarchicznej klasteryzacji aglomeracyjnej
linked = linkage(df_scaled, method='ward')

# Rysowanie dendrogramu
plt.figure(figsize=(10, 7))
dendrogram(linked, orientation='top', distance_sort='descending',␣
 ↪show_leaf_counts=True)
```

```
plt.title('Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
plt.show()
```



[ ]:
```
max_d = 40  # Próg odcicia, mona dostosowa
clusters = fcluster(linked, max_d, criterion='distance')
```

[ ]:
```
unique_labels = set(clusters)
unique_labels
```

[ ]: {1, 2, 3, 4, 5}

[ ]:
```
silhouette_avg = silhouette_score(df_scaled, clusters)
print(f'Silhouette Score: {silhouette_avg}')
```

```
Silhouette Score: 0.3779674441114403
```

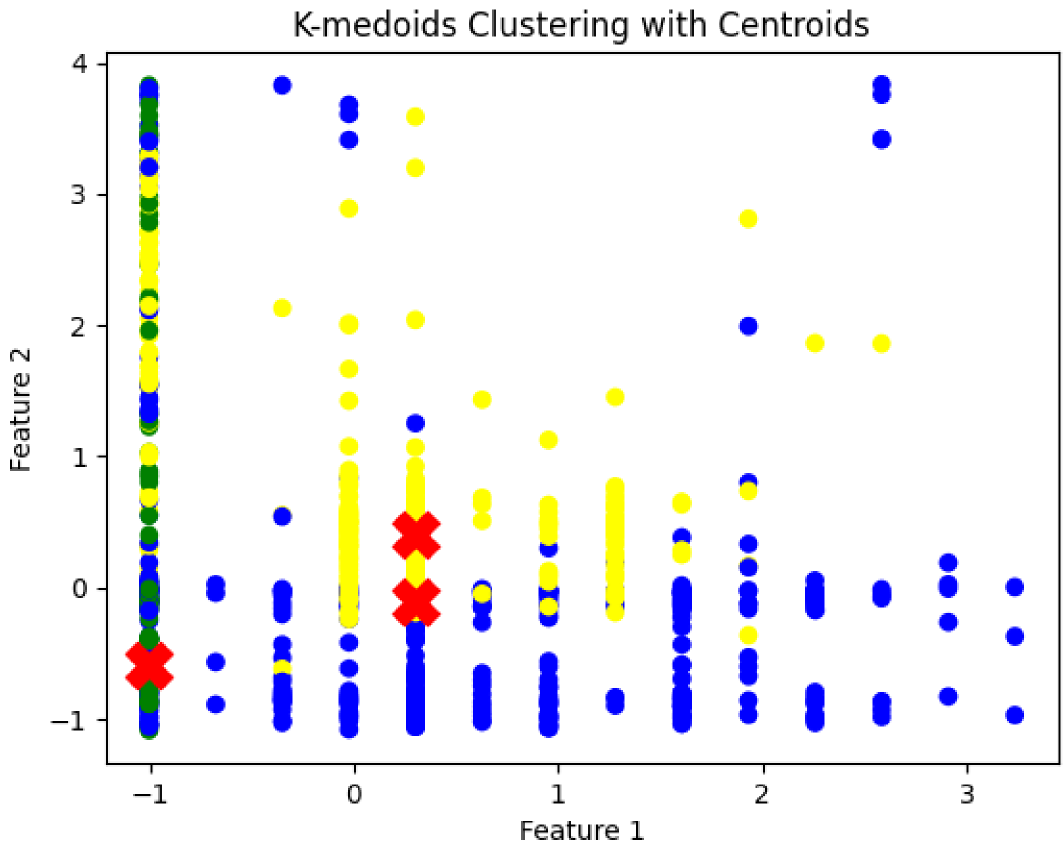
1.4.3 K-medoids

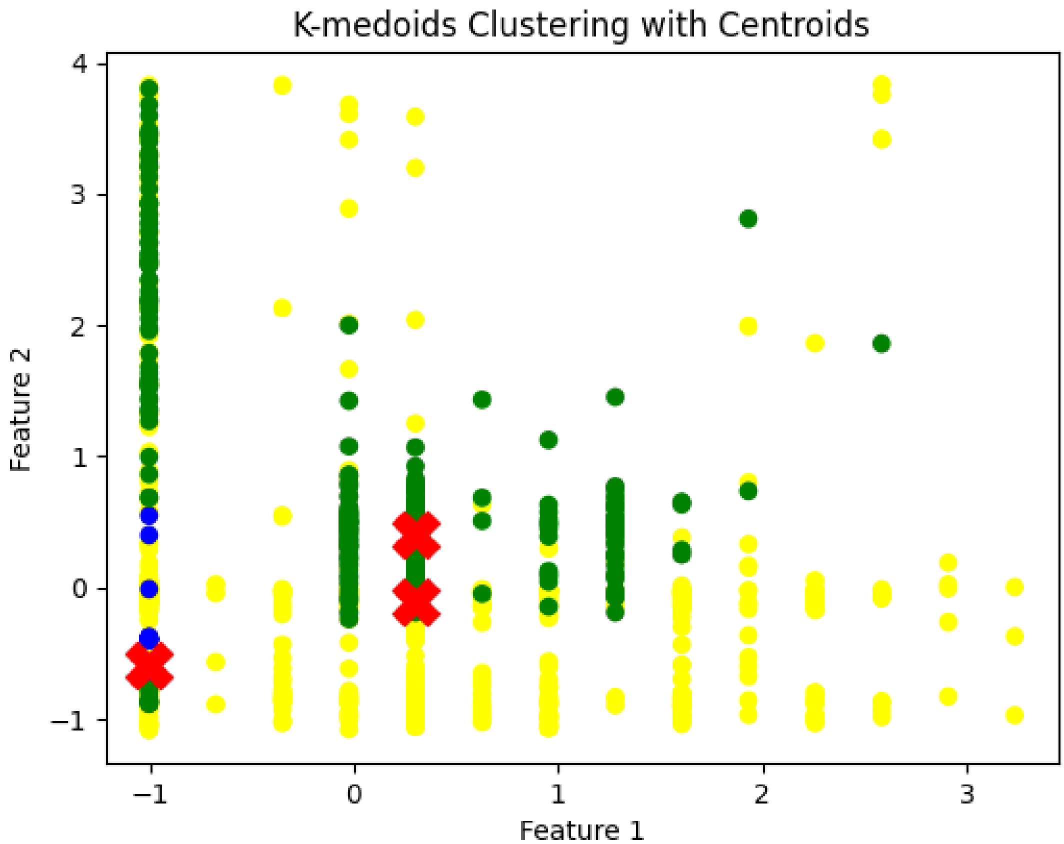
[ ]:
```
#%pip install scikit-learn-extra
from sklearn_extra.cluster import KMedoids
def doKMedoids(X, nclust=n_clusters, xaxis = 2, yaxis = 6, real = False, ␣
↪multidimensional = False, zaxis = 4):
    model = KMedoids(nclust, random_state=42)

    clust_labels = model.fit_predict(df_scaled)


    color_labels = ['yellow' if clust_labels[i] == 0 else 'blue' if␣
↪clust_labels[i] == 1 else 'green' for i in range(len(clust_labels))]


    print(f"Model inertia: {model.inertia_}")
    print("Accuracy: ", np.mean(real_labels == clust_labels))
    print("Silhouette coefficient:" , silhouette_score(X, clust_labels))
    print("Davies Bouldin Score:" , davies_bouldin_score(X, clust_labels))
    print("Calinski Harabasz Score:" , calinski_harabasz_score(X, clust_labels))
    print()


    plt.scatter(X.iloc[:,xaxis], X.iloc[:,yaxis], c=color_labels,␣
↪cmap='viridis')
    plt.scatter(model.cluster_centers_[:, xaxis], model.cluster_centers_[:,␣
↪yaxis], s=300, c='red', marker='X')
    plt.title('K-medoids Clustering with Centroids')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.show()

    if real:
        plt.scatter(X.iloc[:,xaxis], X.iloc[:,yaxis], c=real_color_labels,␣
↪cmap='viridis')
        plt.scatter(model.cluster_centers_[:, xaxis], model.cluster_centers_[:,␣
↪yaxis], s=300, c='red', marker='X')
        plt.title('K-medoids Clustering with Centroids')
        plt.xlabel('Feature 1')
        plt.ylabel('Feature 2')
        plt.show()

    if multidimensional:
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')
```

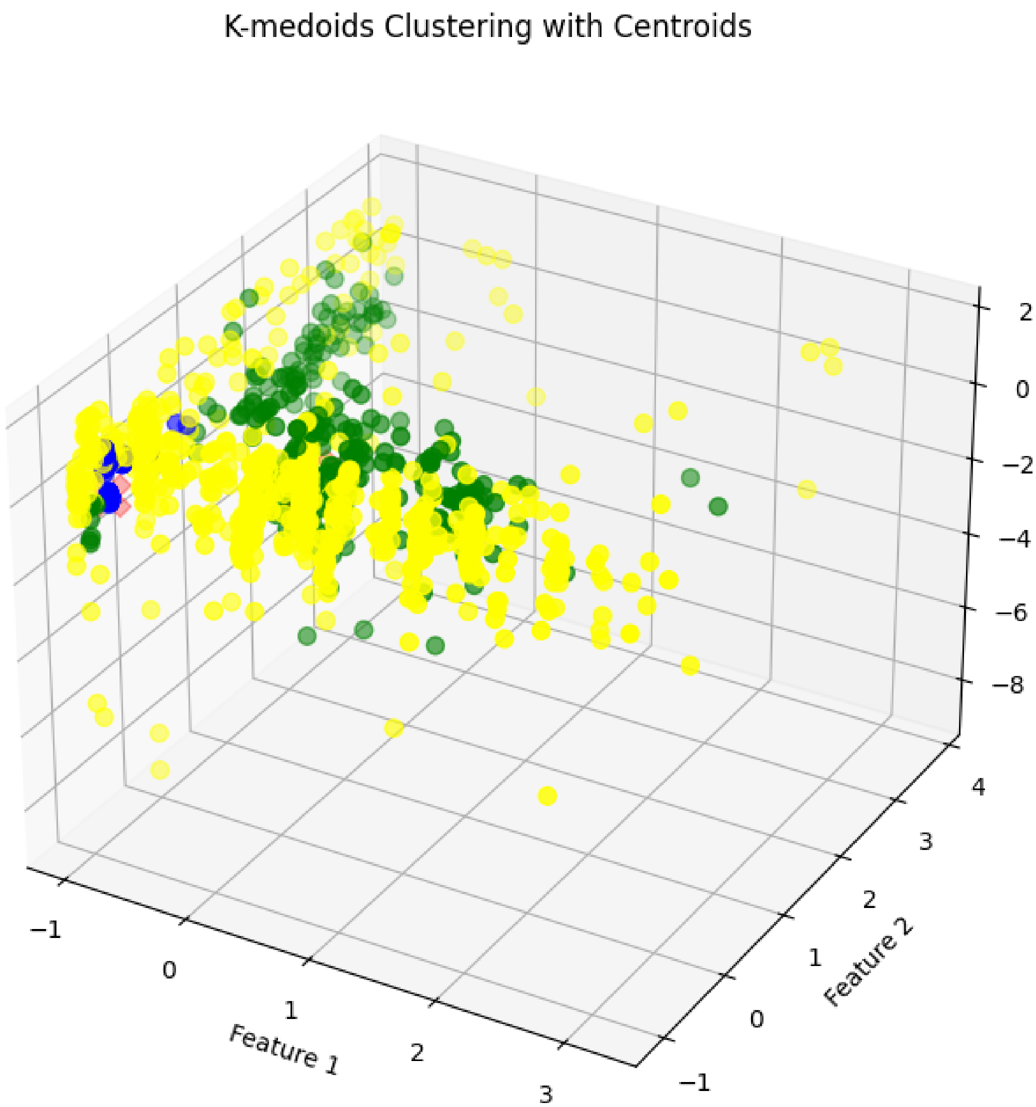
```
        ax.scatter(X.iloc[:,xaxis], X.iloc[:,yaxis], X.iloc[:,zaxis],␣
↪c=real_color_labels, s=50, cmap='viridis')
        ax.scatter(model.cluster_centers_[:, xaxis], model.cluster_centers_[:,␣
↪yaxis],model.cluster_centers_[:, zaxis], s=300, c='red', marker='X')
        ax.set_title('K-medoids Clustering with Centroids')
        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_zlabel('Feature 3')
        plt.show()
    if multidimensional & real:
        fig = plt.figure(figsize=(10, 8))
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(X.iloc[:,xaxis], X.iloc[:,yaxis], X.iloc[:,zaxis],␣
↪c=color_labels, s=50, cmap='viridis')
        ax.scatter(model.cluster_centers_[:, xaxis], model.cluster_centers_[:,␣
↪yaxis],model.cluster_centers_[:, zaxis], s=300, c='red', marker='X')
        ax.set_title('K-medoids Clustering with Centroids')
        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_zlabel('Feature 3')
        plt.show()
    return clust_labels
```

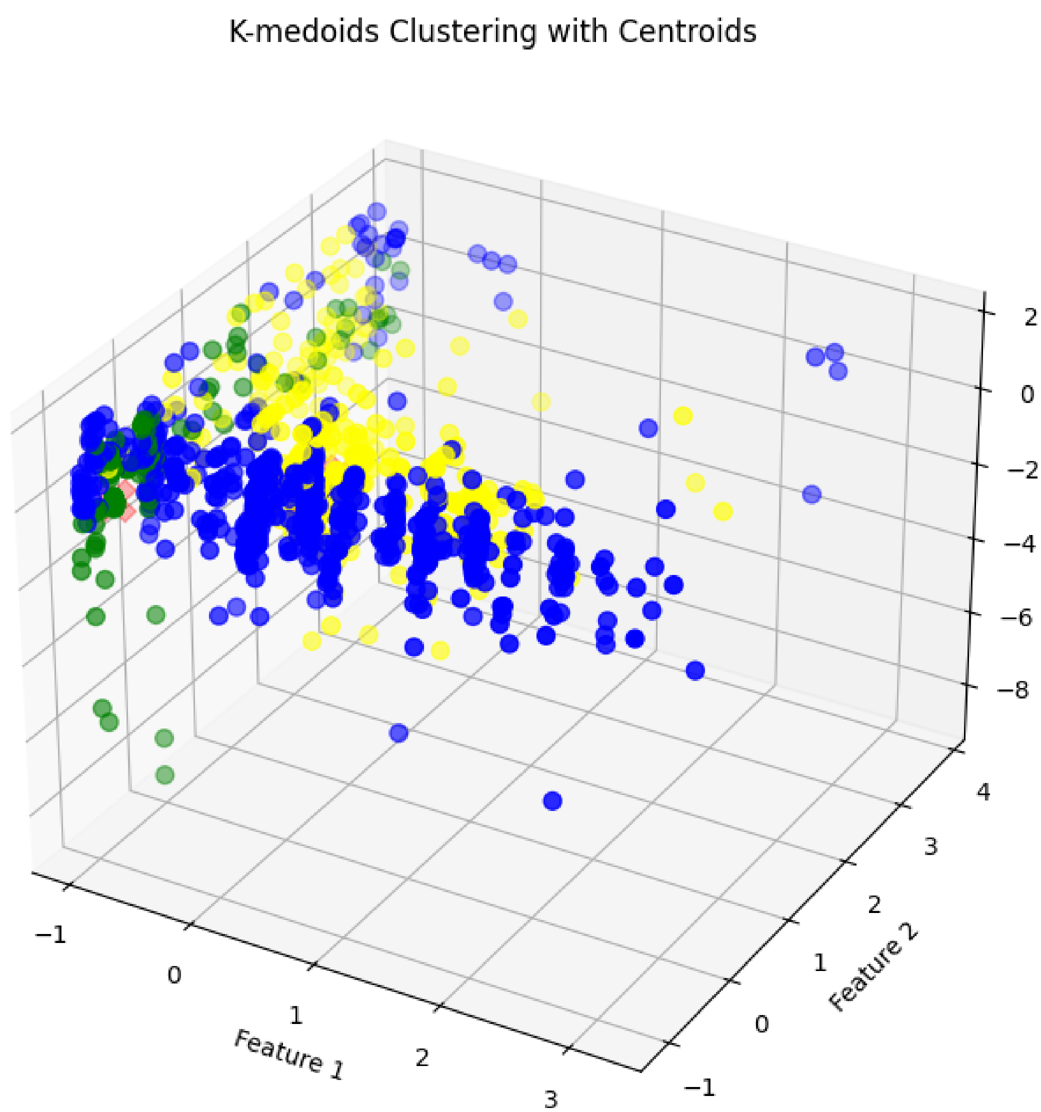
[ ]:
```
x = reduced_result.copy()
scaler = StandardScaler()
x = scaler.fit_transform(x)
x = pd.DataFrame(x, columns=reduced_result.columns)
clust_labels = doKMedoids(x, n_clusters, 5, 6, True, True, 4)
```

```
Model inertia: 2710.311988662948
Accuracy:  0.05073529411764706
Silhouette coefficient: 0.3550226318094353
Davies Bouldin Score: 1.1716027085049459
Calinski Harabasz Score: 470.79220093875347
```


K-medoids Clustering with Centroids
Feature 2
Feature 1
4
3
2
1
0
−1
−1
0
1
2
3


K-medoids Clustering with Centroids
Feature 2
Feature 1
4
3
2
1
0
−1
−1
0
1
2
3


K-medoids Clustering with Centroids
Feature 1
Feature 2
−1
0
1
2
3
−1
0
1
2
3
4
2
0
−2
−4
−6
−8



Normalization approach seems to be the best and now we are trying to get the meaning of clusters.

2 Clusters meaning

[ ]:
```
df = reduced_result.copy()

df
```

[ ]:
```
       total_time    mean_speed  mean_acceleration  sd_acceleration  \
0        2.516518  2.408345e+01           9.919805         4.656513
1        2.516498  2.172662e+01           9.841829         2.392057
```

```
2       2.516481  1.501819e+01           9.842018          3.281112
3       2.516464  2.259201e+01          10.071431          2.263844
4       2.516464  2.374844e+01           9.505357          2.545282
...          ...           ...                ...               ...
1356    2.516909  1.285305e-24           9.702890          0.082049
1357    2.516899  9.276274e-31           9.717226          0.096787
1358    2.516883  2.389216e-36           9.713042          0.111261
1359    2.516877  8.063703e-40           9.713457          0.043896
1360    2.516874  9.794135e-43           9.747030          0.482260

      mean_magnetometer  steps_per_minute  average_roll   sd_roll  \
0             53.910455        119.212340     -2.900108  0.114005
1             53.910455          0.000000     -1.040673  2.811269
2             53.910455        286.113823     -1.493453  2.470934
3             53.910455         95.371919     -2.788037  0.140679
4             53.910455        143.057878     -2.597302  0.997407
...                 ...               ...           ...       ...
1356          47.112348          0.000000     -2.808275  0.017303
1357          47.345006          0.000000     -2.792382  0.016608
1358          47.278723          0.000000     -2.758431  0.035709
1359          46.980630          0.000000     -2.694629  0.003280
1360          47.180419          0.000000     -2.711286  0.013943

      median_pitch   min_yaw
0         1.084632  0.722495
1         1.156837  0.880115
2         0.955256  0.762258
3         0.945513  1.384156
4         0.854464  1.453768
...            ...       ...
1356     -0.275511  2.757832
1357     -0.260294  2.725673
1358     -0.256462  2.704930
1359     -0.250087  2.725769
1360     -0.245037  2.577463

[1360 rows x 10 columns]
```

[ ]:
```
clust_labels_norm
```

[ ]:
```
array([2, 1, 0, ..., 1, 1, 1])
```

[ ]:
```
df = reduced_result.copy()
df['cluster'] = clust_labels_norm
```

[ ]:
```
unique_labels = np.unique(clust_labels_norm)

for label in unique_labels:
    cluster_data = df[clust_labels_norm == label]
```

```python
    cluster_mean = cluster_data.mean()
    cluster_median = cluster_data.median()
    cluster_std = cluster_data.std()

    print(f"Cluster {label} Summary:")
    print("Mean:")
    print(cluster_mean)
    print("Median:")
    print(cluster_median)
    print("Standard Deviation:")
    print(cluster_std)
    print("\n")
```

```
Cluster 0 Summary:
Mean:
total_time              2.516560
mean_speed             15.597247
mean_acceleration      10.297781
sd_acceleration         3.346301
mean_magnetometer      49.066893
steps_per_minute      184.471257
average_roll           -1.850365
sd_roll                 0.651305
median_pitch            0.766108
min_yaw                -1.006568
cluster                 0.000000
dtype: float64
Median:
total_time              2.516557
mean_speed             18.239145
mean_acceleration      10.012300
sd_acceleration         2.864155
mean_magnetometer      49.243845
steps_per_minute      190.719055
average_roll           -1.813175
sd_roll                 0.102560
median_pitch            0.637864
min_yaw                -1.089999
cluster                 0.000000
dtype: float64
Standard Deviation:
total_time             0.000227
mean_speed             8.609816
mean_acceleration      0.735036
sd_acceleration        1.460186
mean_magnetometer      3.226206
steps_per_minute      41.409303
```

```
average_roll          0.948921
sd_roll               0.836442
median_pitch          0.380721
min_yaw               1.704996
cluster               0.000000
dtype: float64


Cluster 1 Summary:
Mean:
total_time            2.509413
mean_speed            4.884155
mean_acceleration     9.814485
sd_acceleration       0.840378
mean_magnetometer    47.968278
steps_per_minute      0.175627
average_roll         -1.515021
sd_roll               0.380504
median_pitch          0.446015
min_yaw              -1.439981
cluster               1.000000
dtype: float64
Median:
total_time            2.516548
mean_speed            0.621910
mean_acceleration     9.720197
sd_acceleration       0.153119
mean_magnetometer    47.394126
steps_per_minute      0.000000
average_roll         -2.447100
sd_roll               0.019106
median_pitch          0.369083
min_yaw              -2.395597
cluster               1.000000
dtype: float64
Standard Deviation:
total_time           0.089622
mean_speed           7.368702
mean_acceleration    0.217518
sd_acceleration      1.148805
mean_magnetometer    3.020207
steps_per_minute     2.040594
average_roll         1.612504
sd_roll              0.799418
median_pitch         0.556529
min_yaw              1.613813
cluster              0.000000
dtype: float64
```

```
Cluster 2 Summary:
Mean:
total_time            2.516576
mean_speed           15.372104
mean_acceleration    10.342810
sd_acceleration       3.630984
mean_magnetometer    49.260025
steps_per_minute     86.950191
average_roll         -1.875503
sd_roll               0.722723
median_pitch          0.802143
min_yaw              -1.140467
cluster               2.000000
dtype: float64
Median:
total_time            2.516560
mean_speed           17.410559
mean_acceleration    10.002121
sd_acceleration       2.987783
mean_magnetometer    49.137282
steps_per_minute     95.359755
average_roll         -1.810046
sd_roll               0.109180
median_pitch          0.792527
min_yaw              -1.457354
cluster               2.000000
dtype: float64
Standard Deviation:
total_time            0.000215
mean_speed            8.492964
mean_acceleration     0.769381
sd_acceleration       1.480246
mean_magnetometer     2.683725
steps_per_minute     16.073218
average_roll          0.846735
sd_roll               0.852241
median_pitch          0.384508
min_yaw               1.754047
cluster               0.000000
dtype: float64
```

Based on the provided data, we can attempt to characterize the activity groups represented by the three clusters. Here is the analysis of each cluster:

2.0.1 Cluster 0

Mean Values: - Total time: 2.52 minutes - Mean speed: 15.31 m/s - Mean acceleration: 10.35 m/sš - Standard deviation of acceleration: 3.64 m/sš - Mean magnetometer value: 49.33 - Steps per minute: 86.58 - Average roll: -1.86 - Standard deviation of roll: 0.77 - Median pitch: 0.81 - Minimum yaw: -1.15

Median Values: - Total time: 2.52 minutes - Mean speed: 17.24 m/s - Mean acceleration: 10.01 m/sš - Standard deviation of acceleration: 3.08 m/sš - Mean magnetometer value: 49.22 - Steps per minute: 95.36 - Average roll: -1.79 - Standard deviation of roll: 0.12 - Median pitch: 0.80 - Minimum yaw: -1.39

Standard Deviation: - Total time: 0.00022 - Mean speed: 8.56 - Mean acceleration: 0.77 - Standard deviation of acceleration: 1.46 - Mean magnetometer value: 2.75 - Steps per minute: 16.13 - Average roll: 0.85 - Standard deviation of roll: 0.86 - Median pitch: 0.39 - Minimum yaw: 1.74

Characteristics: - High mean speed and acceleration, suggesting intense activity. - Relatively high number of steps per minute, indicating vigorous movement, such as running. - Moderate variability in acceleration and roll.

2.0.2 Cluster 1

Mean Values: - Total time: 2.51 minutes - Mean speed: 5.06 km/h - Mean acceleration: 9.80 m/sš - Standard deviation of acceleration: 0.83 m/sš - Mean magnetometer value: 48.00 - Steps per minute: 0.26 - Average roll: -1.64 - Standard deviation of roll: 0.34 - Median pitch: 0.43 - Minimum yaw: -1.44

Median Values: - Total time: 2.52 minutes - Mean speed: 0.66 km/h - Mean acceleration: 9.72 m/sš - Standard deviation of acceleration: 0.18 m/sš - Mean magnetometer value: 47.42 - Steps per minute: 0.00 - Average roll: -2.45 - Standard deviation of roll: 0.02 - Median pitch: 0.33 - Minimum yaw: -2.40

Standard Deviation: - Total time: 0.08 - Mean speed: 7.52 km/h - Mean acceleration: 0.18 - Standard deviation of acceleration: 1.08 - Mean magnetometer value: 3.36 - Steps per minute: 2.47 - Average roll: 1.52 - Standard deviation of roll: 0.74 - Median pitch: 0.54 - Minimum yaw: 1.57

Characteristics: - Very low number of steps per minute, indicating little to no movement, such as sitting or standing still. - Low mean speed and moderate mean acceleration. - High variability in roll and pitch, suggesting unstable posture or slight movements.

2.0.3 Cluster 2

Mean Values: - Total time: 2.52 minutes - Mean speed: 15.14 km/h - Mean acceleration: 10.28 m/sš - Standard deviation of acceleration: 3.31 m/sš - Mean magnetometer value: 48.99 - Steps per minute: 187.75 - Average roll: -1.86 - Standard deviation of roll: 0.64 - Median pitch: 0.78 - Minimum yaw: -1.06

Median Values: - Total time: 2.52 minutes - Mean speed: 17.33 km/h - Mean acceleration: 9.95 m/sš - Standard deviation of acceleration: 2.88 m/sš - Mean magnetometer value: 49.06 - Steps per minute: 190.72 - Average roll: -1.81 - Standard deviation of roll: 0.10 - Median pitch: 0.65 - Minimum yaw: -1.37

Standard Deviation: - Total time: 0.00021 - Mean speed: 8.64 - Mean acceleration: 0.73 - Standard deviation of acceleration: 1.47 - Mean magnetometer value: 3.36 - Steps per minute: 44.67 - Average roll: 0.90 - Standard deviation of roll: 0.82 - Median pitch: 0.38 - Minimum yaw: 1.74

Characteristics: - High mean speed and acceleration, similar to Cluster 0, suggesting intense activity. - Higher number of steps per minute than Cluster 0, indicating very vigorous movement, such as sprinting or fast running. - Moderate variability in acceleration and roll.

2.0.4 Summary

1. **Cluster 0:** High-intensity activity (likely running).
2. **Cluster 1:** Low to no physical activity (sitting or standing still).
3. **Cluster 2:** Very high-intensity activity (likely cycling).

Each cluster represents different levels of intensity and types of physical activity, ranging from high-intensity running, to no movement, to very high-intensity sprinting or fast running.