Machine Learning: Examen 1

Clustering methods and principal component analysis

By: Enrique Mena Camilo

In [1]:
```
import importlib

from itertools import permutations

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler, MinMaxScaler

from utils import eda, dimensionality_reduction, clustering, visualization

pd.set_option('display.max_columns', None)
sns.set_theme(style="darkgrid")
```

Paths

In [2]:
```
DATA_PATH = "../data/"
FIGURES_PATH = "./figures/"
```

Data loading

In [3]:
```
data = pd.read_csv(DATA_PATH + "bodyPerformance.csv")
print(f"Total instances: {data.shape[0]}")
print(f"Total features: {data.shape[1]}")
data.head(10)
```

```
Total instances: 13393
Total features: 12
```

Out[3]:

	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce	sit and bend forward_cm
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	18.4
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	16.3
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	12.0
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	15.2
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	27.1
5	36.0	F	165.4	55.40	22.0	64.0	119.0	23.8	21.0
6	42.0	F	164.5	63.70	32.2	72.0	135.0	22.7	0.8
7	33.0	M	174.9	77.20	36.9	84.0	137.0	45.9	12.3
8	54.0	M	166.8	67.50	27.6	85.0	165.0	40.4	18.6
9	28.0	M	185.0	84.60	14.4	81.0	156.0	57.9	12.1

Rename some columns for easier access

In [4]:
```
data = data.rename(columns={
    "body fat_%": "body_fat_pct",
    "gripForce": "grip_force",
    "sit and bend forward_cm": "sit_and_bend_forward_cm",
    "sit-ups counts": "sit_ups_counts",
    "broad jump_cm": "broad_jump_cm",
})
data.head(10)
```

Out[4]:

	age	gender	height_cm	weight_kg	body_fat_pct	diastolic	systolic	grip_force	sit_
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	
5	36.0	F	165.4	55.40	22.0	64.0	119.0	23.8	
6	42.0	F	164.5	63.70	32.2	72.0	135.0	22.7	
7	33.0	M	174.9	77.20	36.9	84.0	137.0	45.9	
8	54.0	M	166.8	67.50	27.6	85.0	165.0	40.4	
9	28.0	M	185.0	84.60	14.4	81.0	156.0	57.9	

Exploratory Data Analysis

The dataset has NaN values?

In [5]:
```
eda.get_nan_count(data)
```

Out[5]:

	nan_count	nan_percentage
age	0	0.0
gender	0	0.0
height_cm	0	0.0
weight_kg	0	0.0
body_fat_pct	0	0.0
diastolic	0	0.0
systolic	0	0.0
grip_force	0	0.0
sit_and_bend_forward_cm	0	0.0
sit_ups_counts	0	0.0
broad_jump_cm	0	0.0
class	0	0.0

R: No, the dataset is complete

What about the data types?

In [6]:
```
data.dtypes
```

Out[6]:
```
age                         float64
gender                       object
height_cm                   float64
weight_kg                   float64
body_fat_pct                float64
diastolic                   float64
systolic                    float64
grip_force                  float64
sit_and_bend_forward_cm     float64
sit_ups_counts              float64
broad_jump_cm               float64
class                        object
dtype: object
```

R: It's needed to code the columns 'gender' and 'class'

In [7]:
```
eda.code_categorical(data, "gender")
eda.code_categorical(data, "class")
data.head(10)
```

Out[7]:

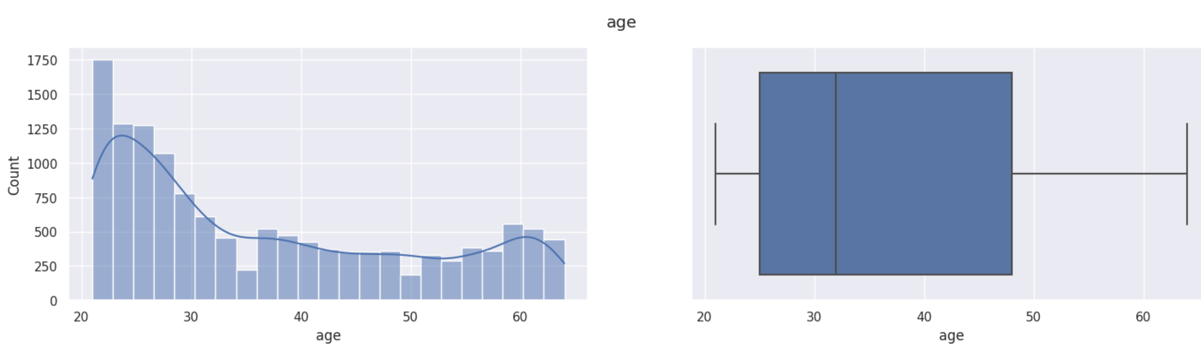
	age	gender	height_cm	weight_kg	body_fat_pct	diastolic	systolic	grip_force	sit_
0	27.0	1	172.3	75.24	21.3	80.0	130.0	54.9	
1	25.0	1	165.0	55.80	15.7	77.0	126.0	36.4	
2	31.0	1	179.6	78.00	20.1	92.0	152.0	44.8	
3	32.0	1	174.5	71.10	18.4	76.0	147.0	41.4	
4	28.0	1	173.8	67.70	17.1	70.0	127.0	43.5	
5	36.0	0	165.4	55.40	22.0	64.0	119.0	23.8	
6	42.0	0	164.5	63.70	32.2	72.0	135.0	22.7	
7	33.0	1	174.9	77.20	36.9	84.0	137.0	45.9	
8	54.0	1	166.8	67.50	27.6	85.0	165.0	40.4	
9	28.0	1	185.0	84.60	14.4	81.0	156.0	57.9	

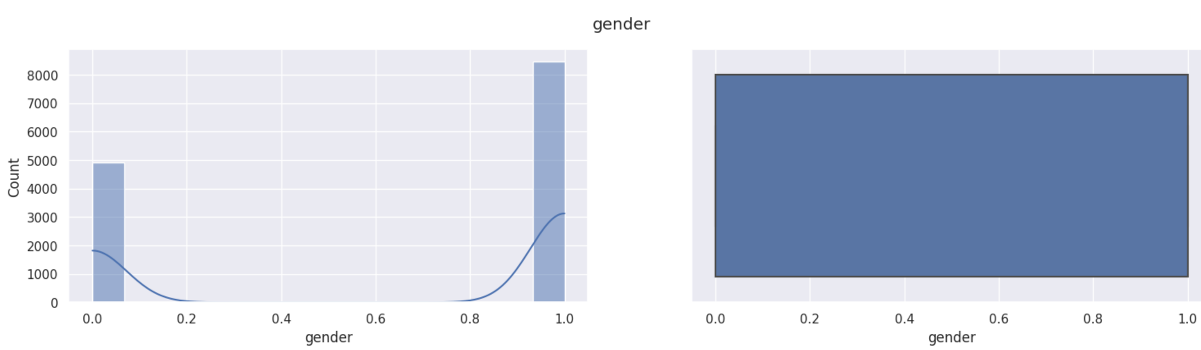
What about the distribution of the data?

Original distributions

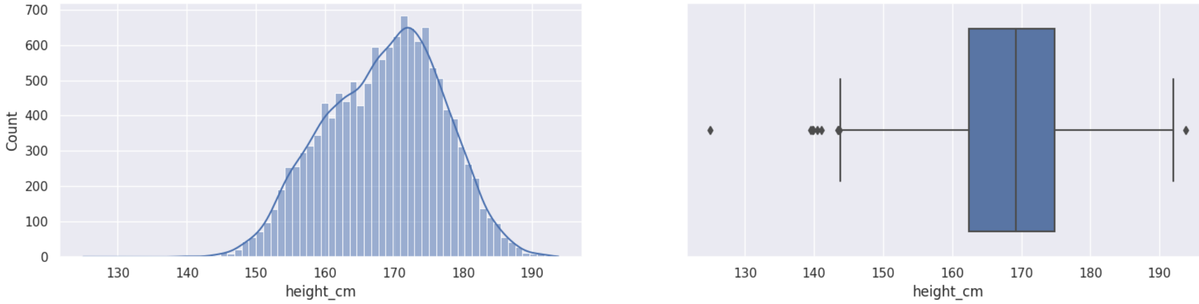
In [8]:

```
for column in data.columns.tolist():
    eda.plot_distribution(data, column, figures_path=FIGURES_PATH)
```

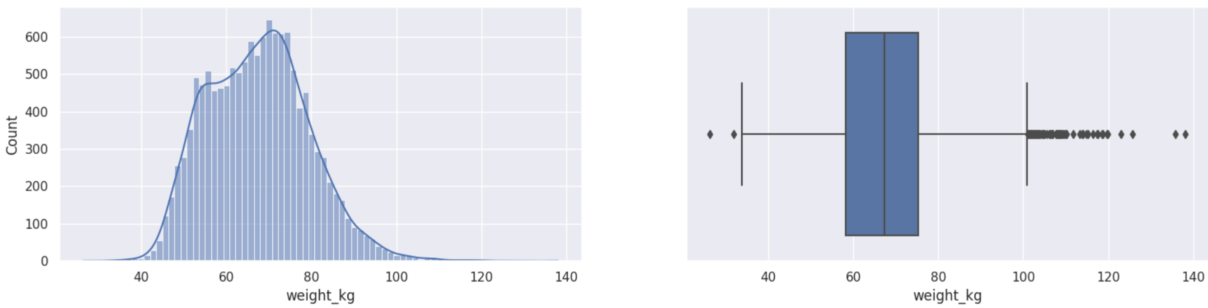




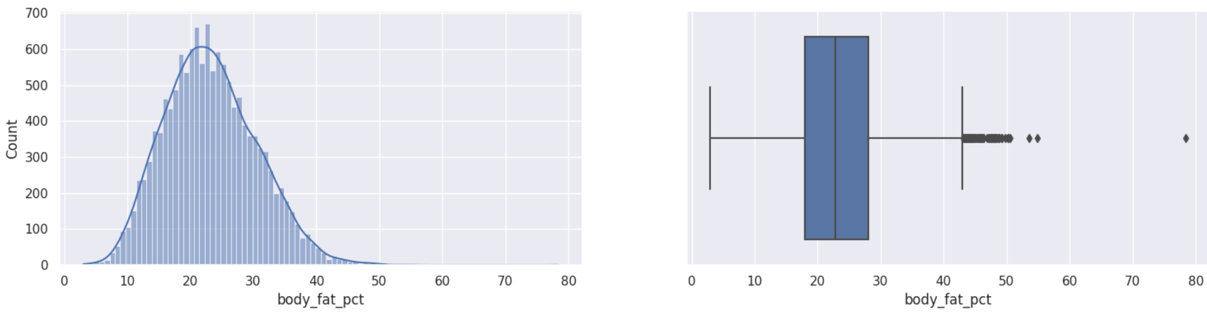
height_cm



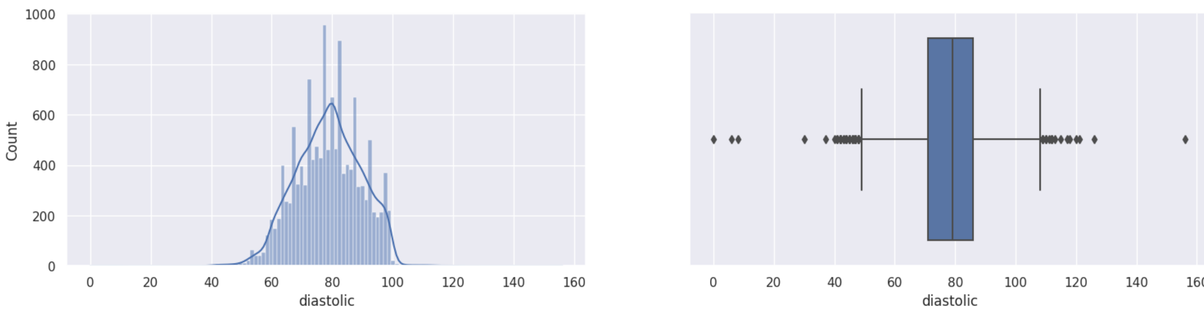
weight_kg



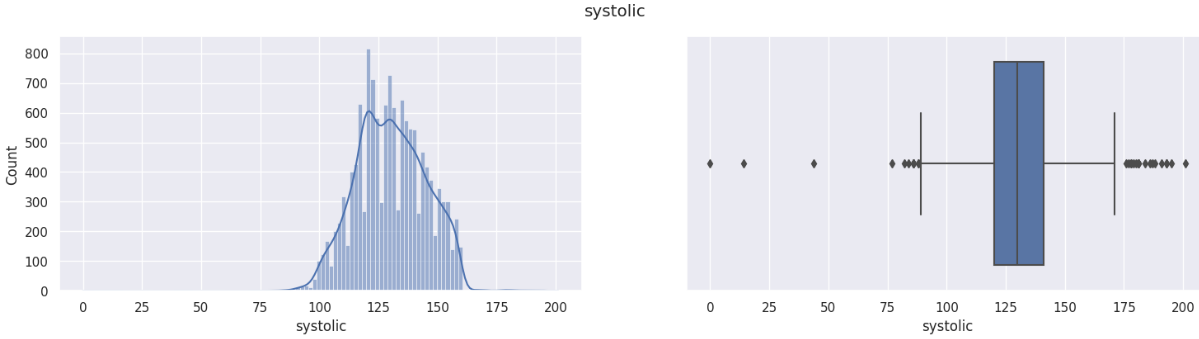
body_fat_pct



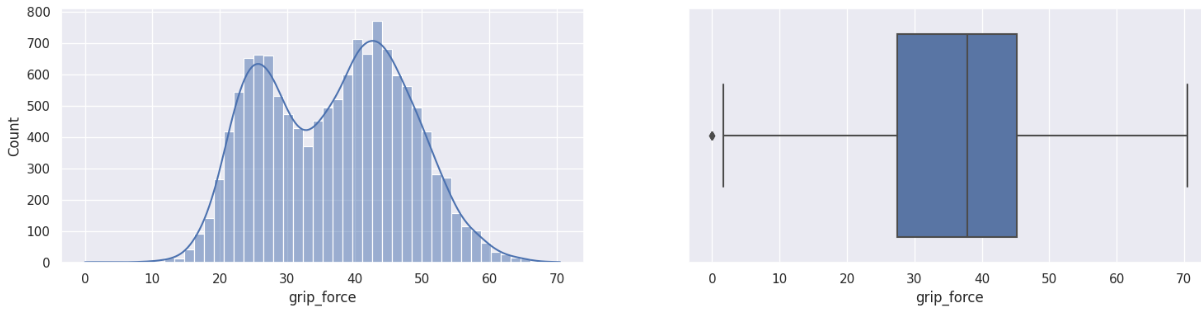
diastolic



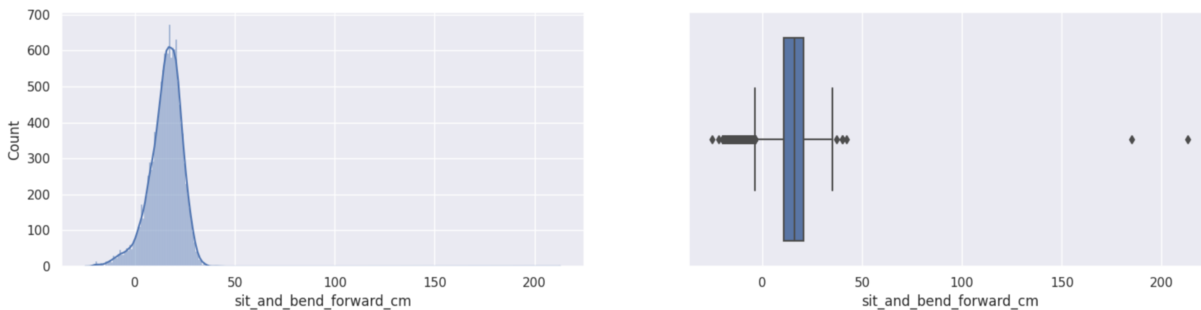
systolic



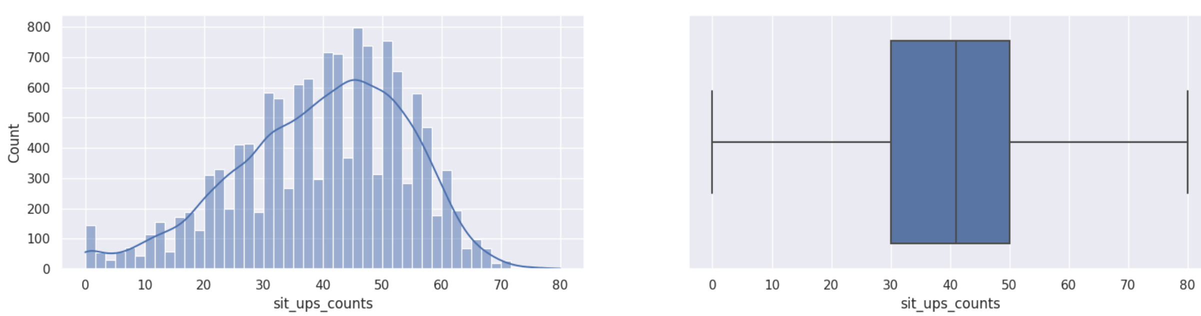
grip_force



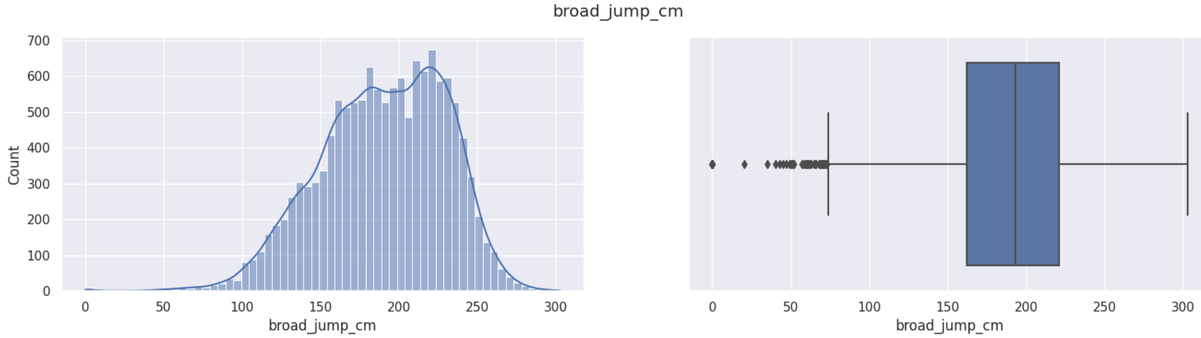
sit_and_bend_forward_cm



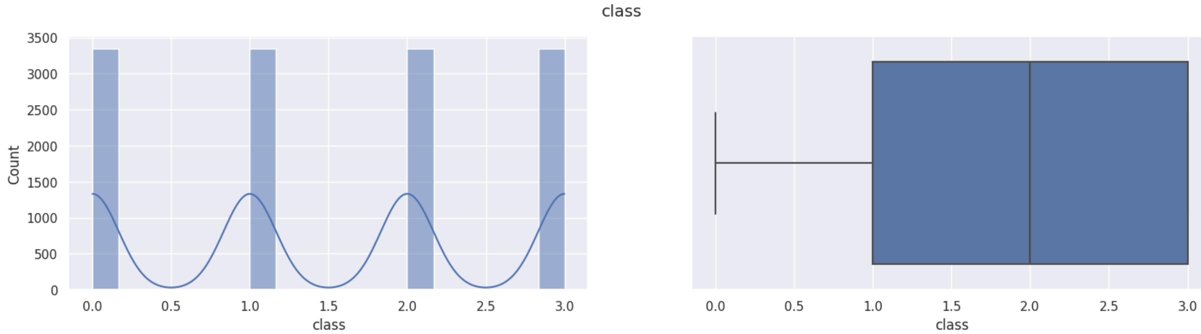
sit_ups_counts



broad_jump_cm



class



R: Most of the features present a distribution close to the normal distribution. Standard scaler will be used for these features

```
In [9]: data_norm = data.copy()
```

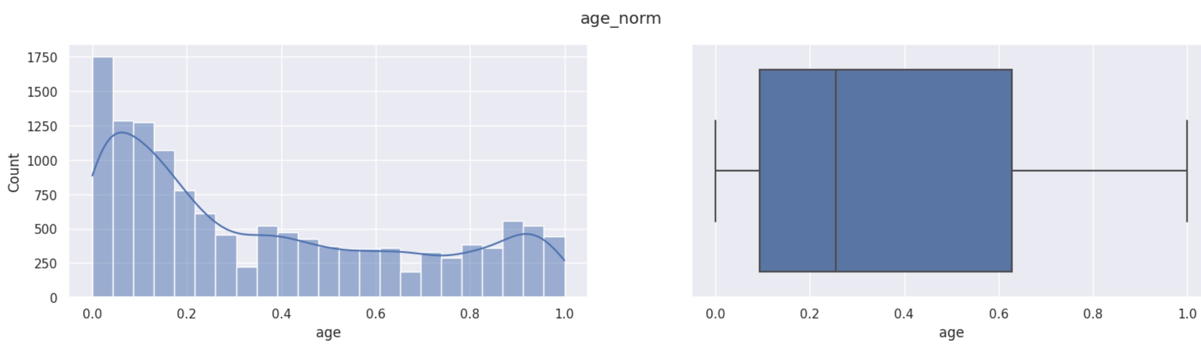
```
In [10]: features = ["height_cm", "weight_kg", "body_fat_pct", "diastolic", "systolic
                     "grip_force", "sit_and_bend_forward_cm", "sit_ups_counts","broac
         for feature in features:
             data_norm[feature] = StandardScaler().fit_transform(data_norm[[feature]]
```

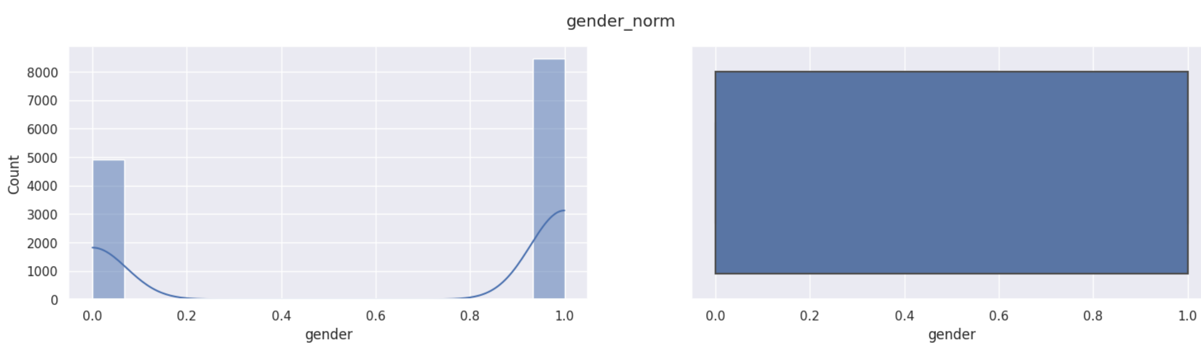
R: Age will be scaled using the min max method, this to ensure that the clustering algorithm has comparable ranges between all attributes

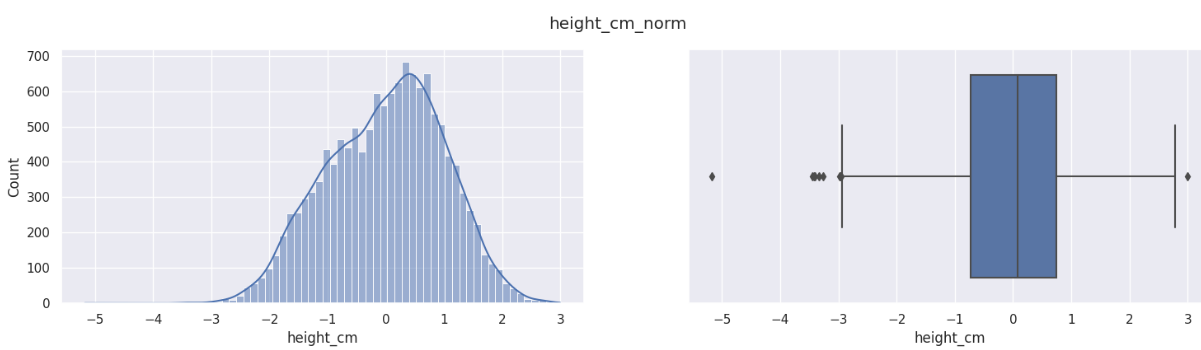
```
In [11]: data_norm["age"] = MinMaxScaler().fit_transform(data_norm[["age"]])
```

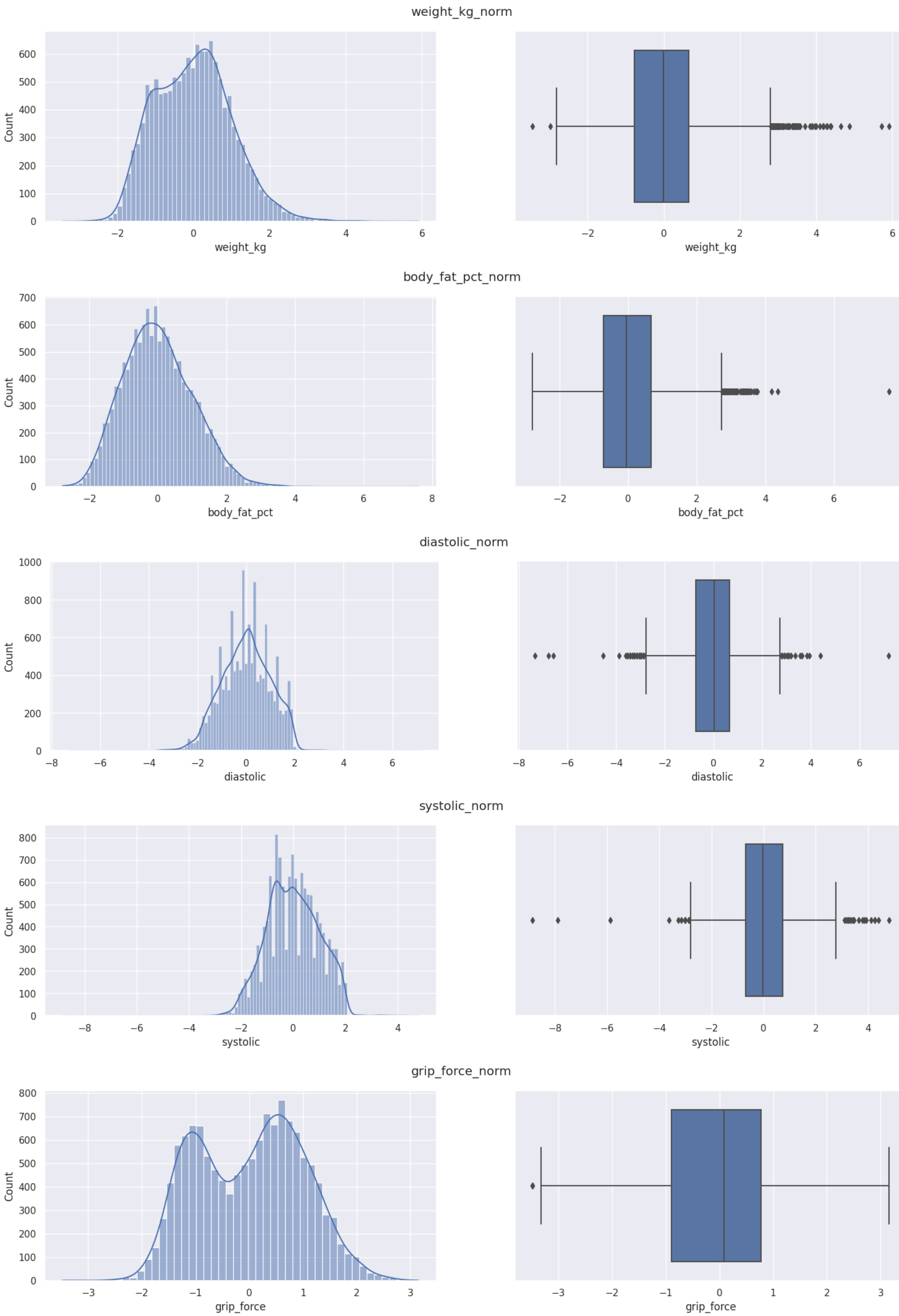
Scaled distributions

```
In [12]: for column in data_norm.columns.tolist():
             eda.plot_distribution(data_norm, column, figures_path=FIGURES_PATH, sufi
```

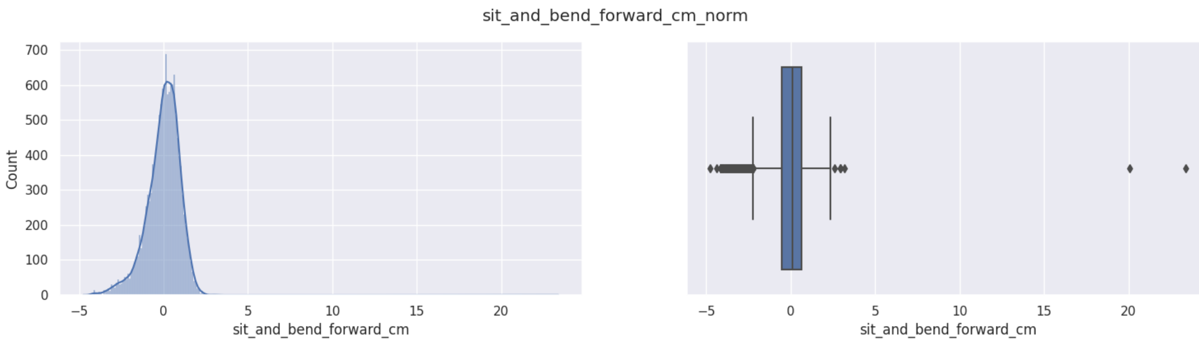


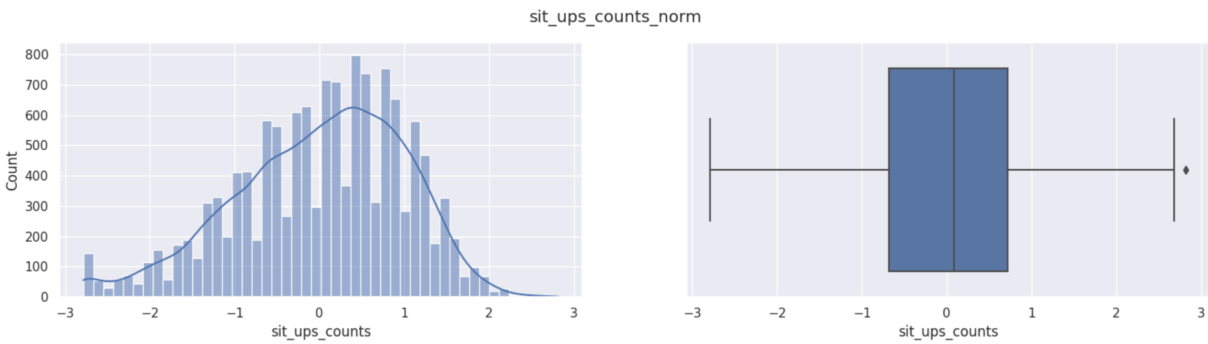


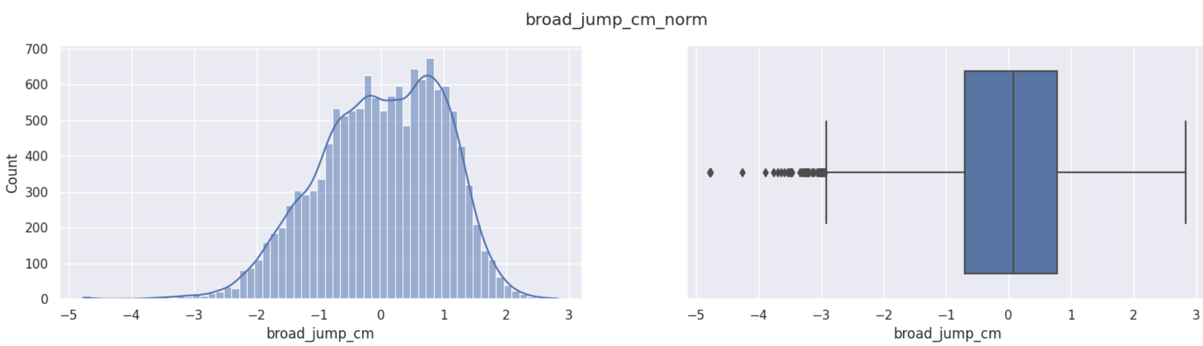


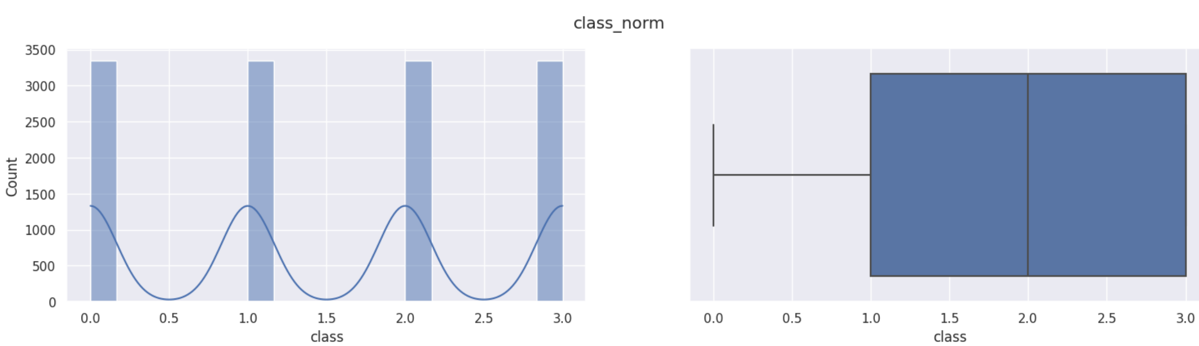


weight_kg_norm
Count
weight_kg
weight_kg
body_fat_pct_norm
Count
body_fat_pct
body_fat_pct
diastolic_norm
Count
diastolic
diastolic
systolic_norm
Count
systolic
systolic
grip_force_norm
Count
grip_force
grip_force



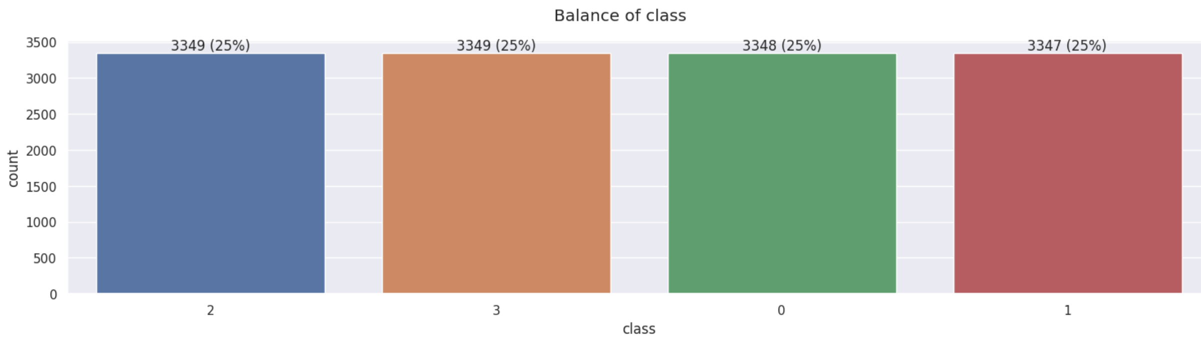






What do we know about the classes? are they balanced?

In [13]:
```
eda.plot_count(data, "class", figures_path=FIGURES_PATH)
```



R: The total number of instances per class varies by 1 or 2 instances, so it could be considered that they are balanced

Dimensionality reduction: Principal component analysis (PCA)

The attributes and the objective variable must be segmented

In [14]:
```
X = data_norm.drop(columns=["class"])
Y = data_norm['class'].to_numpy()
```

Obtaining the most relevant attributes

In [15]:
```
pca_features = dimensionality_reduction.pca_features(X, normalize_data=False
pca_features
```

Out[15]:

	relevance
age	45.201545
gender	19.945752
height_cm	13.057954
weight_kg	7.159068
body_fat_pct	4.281306
broad_jump_cm	3.398194
sit_ups_counts	2.788807
sit_and_bend_forward_cm	2.012700
grip_force	1.147991
systolic	0.639009
diastolic	0.367673

In [16]:
```
relevant_features = pca_features.iloc[:4].index.tolist()
X_relevant = X[relevant_features]
```

```
print(f"Total gain: {pca_features.loc[relevant_features].sum(axis=0)[0]:.2f}
X_relevant.head(10)
```

```
Total gain: 85.36%
```

Out[16]:

	age	gender	height_cm	weight_kg
0	0.139535	1	0.443873	0.652150
1	0.093023	1	-0.422465	-0.974734
2	0.232558	1	1.310211	0.883127
3	0.255814	1	0.704961	0.305684
4	0.162791	1	0.621888	0.021147
5	0.348837	0	-0.374995	-1.008209
6	0.488372	0	-0.481804	-0.313603
7	0.279070	1	0.752432	0.816177
8	0.767442	1	-0.208848	0.004409
9	0.162791	1	1.951065	1.435465

Using the 'age', 'gender', 'height_cm' and 'weight_cm' features yields a gain of 85%

Obtaining subspace of principal components

In [17]:

```
X_pca = dimensionality_reduction.pca(X, 2, normalize_data=False)
X_pca.head(10)
```

Out[17]:

	PC1	PC2
0	2.216666	0.437671
1	0.653010	1.348408
2	2.081523	-1.615139
3	1.790092	0.014114
4	1.191661	1.429019
5	-2.085863	1.242047
6	-2.259175	-1.384257
7	1.240391	-1.332304
8	-0.119634	-2.072823
9	3.752017	-0.844322

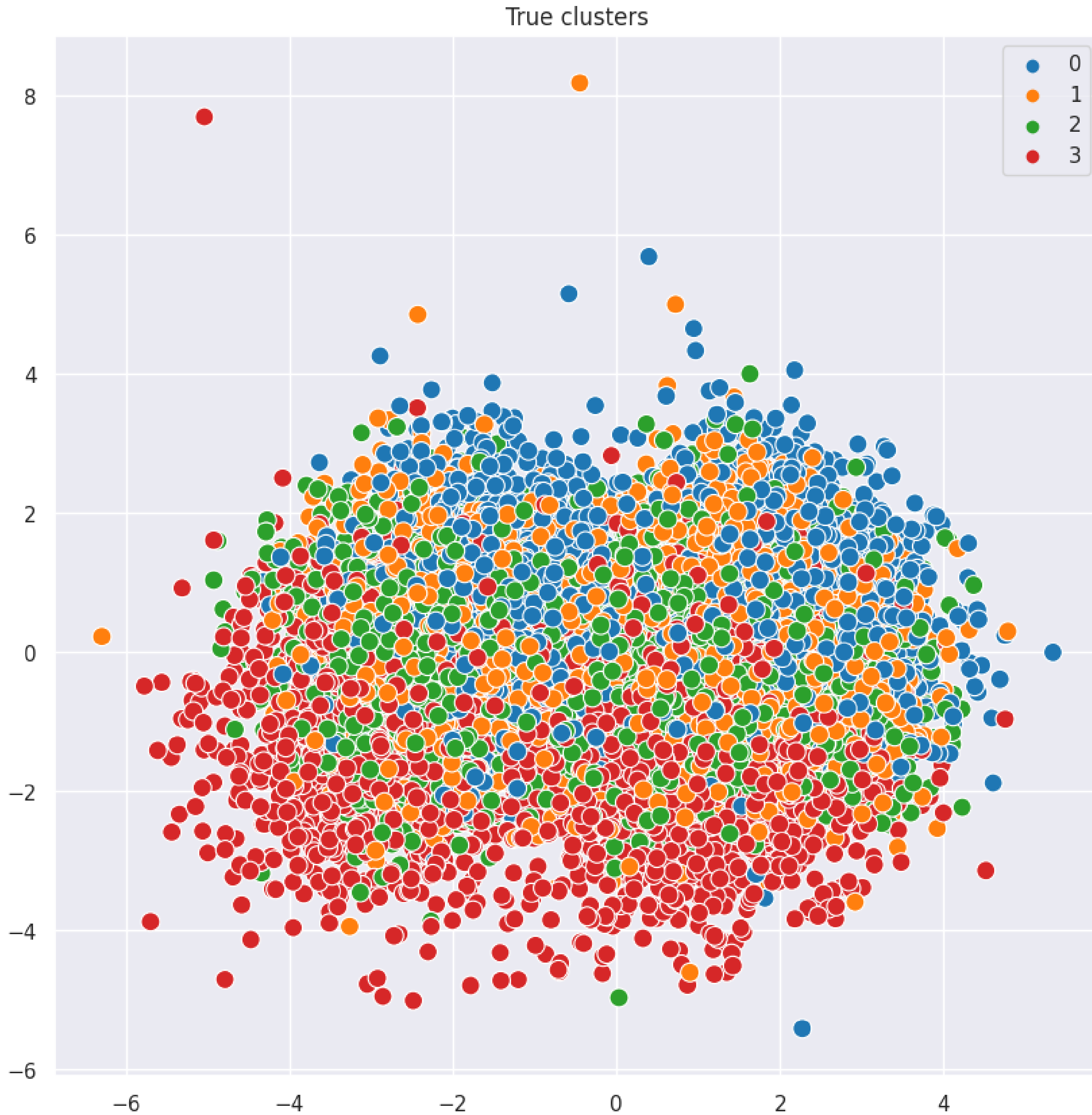
Convert X variables to numpy arrays

In [18]:

```
X = X.to_numpy()
X_relevant = X_relevant.to_numpy()
X_pca = X_pca.to_numpy()
```

Target variable visualization

In [19]:
```
visualization.plot_clustering_result(X_pca, Y, title="True clusters", filena
```

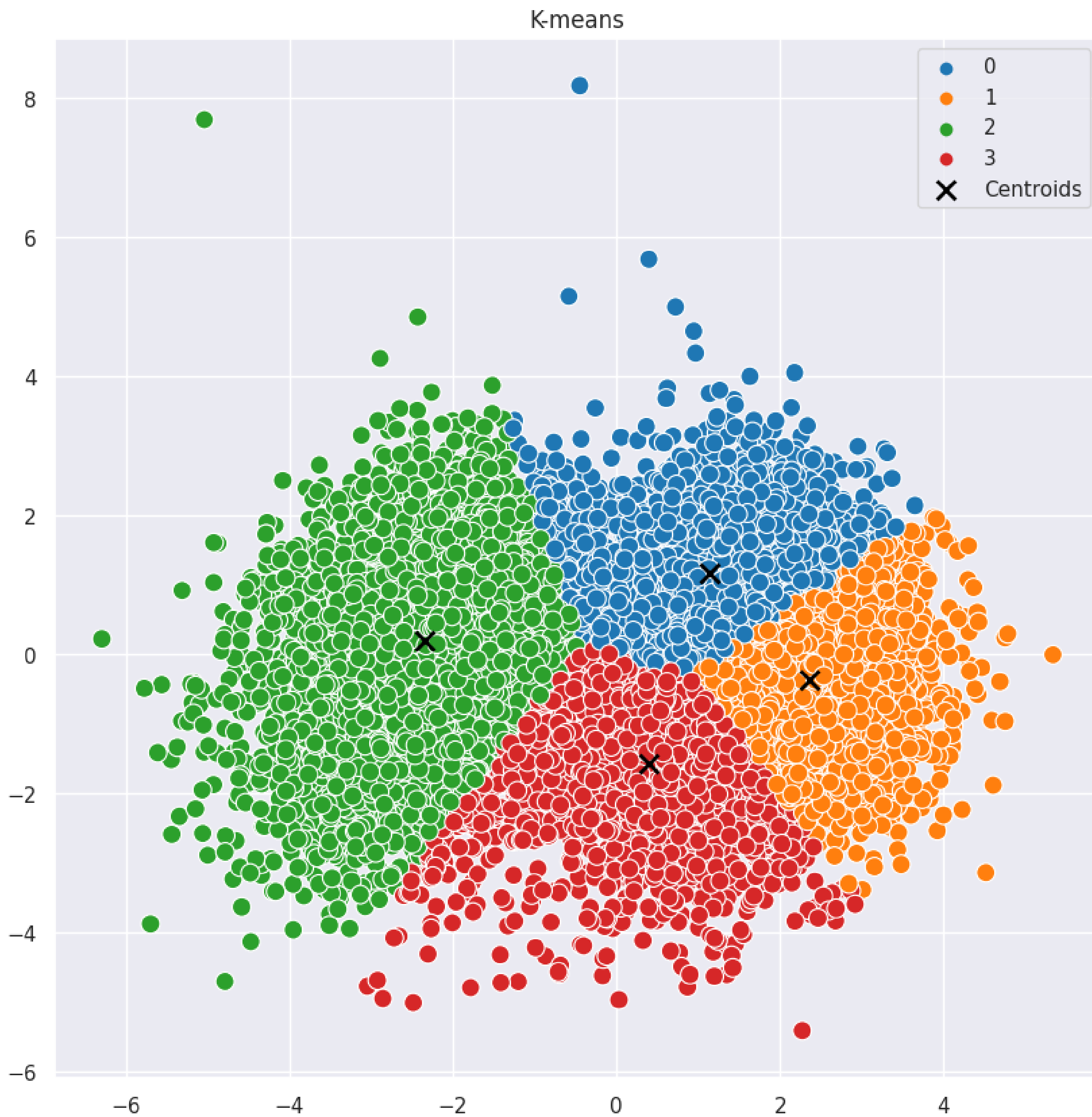


Clustering

K-means example

In [20]:
```
Y_predict, centroids = clustering.k_means(X_pca, 4, max_iter=50, tol=1e-3)
visualization.plot_clustering_result(X_pca, Y_predict, centroids=centroids,
```

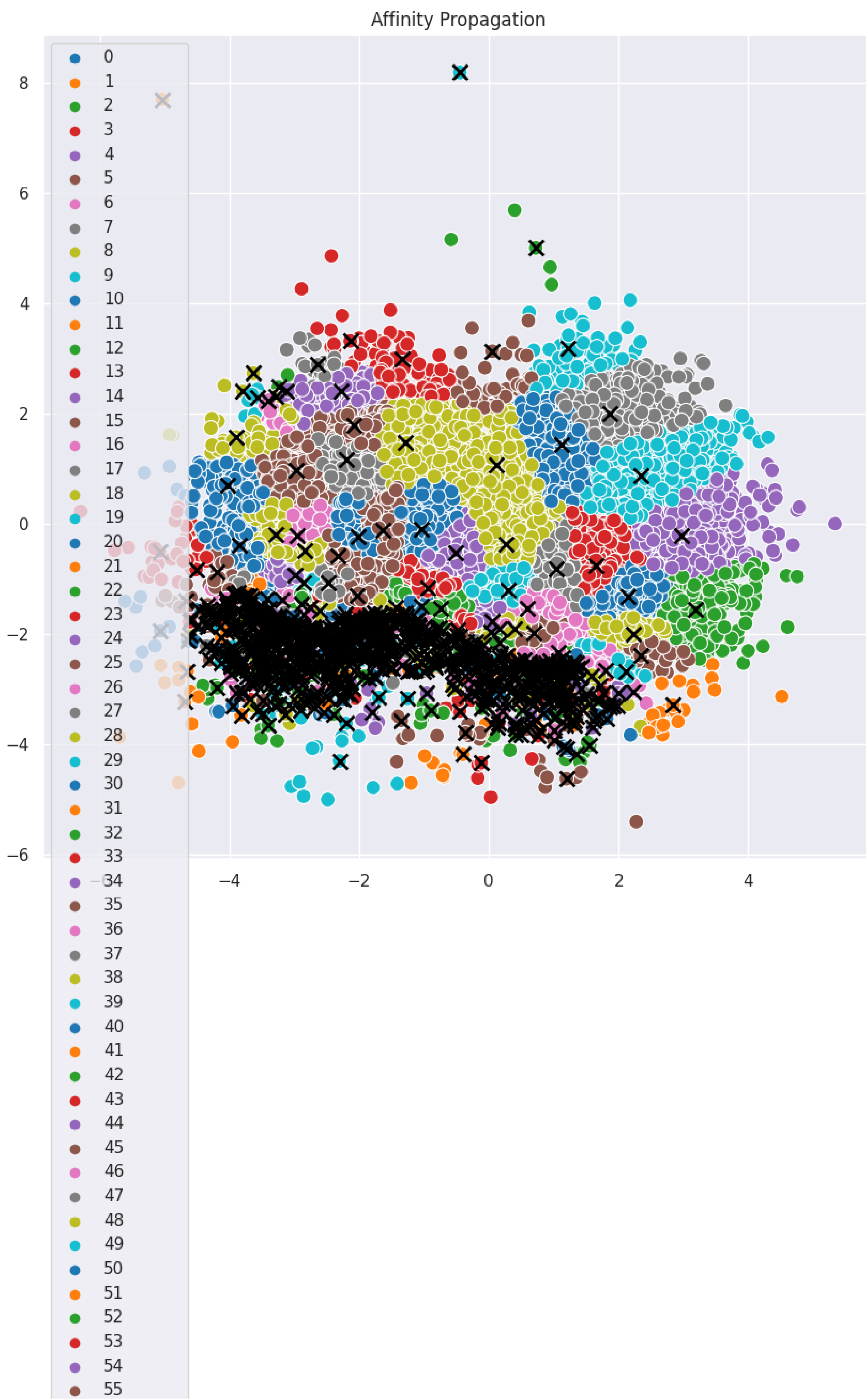
```
34%|███          | 17/50 [00:00<00:00, 477.41it/s]
```



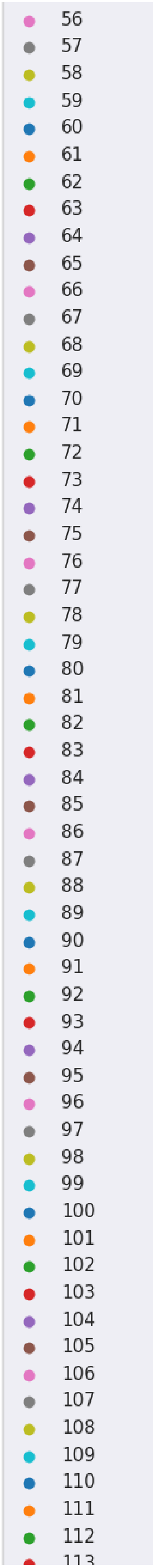
Affinity propagation example

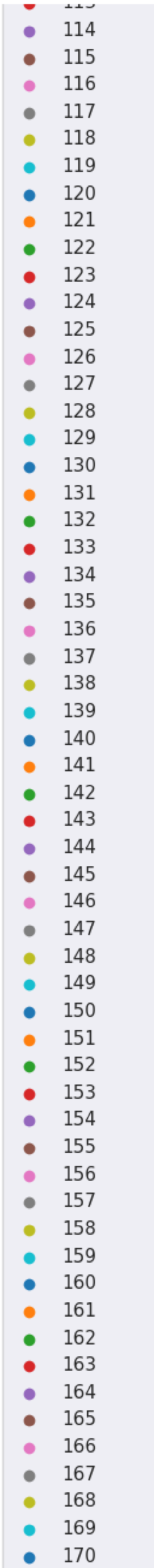
In [21]:
```
Y_predict, centroids = clustering.affinity_propagation(X_pca, damping=0.5, m
visualization.plot_clustering_result(X_pca, Y_predict, centroids=centroids,
```

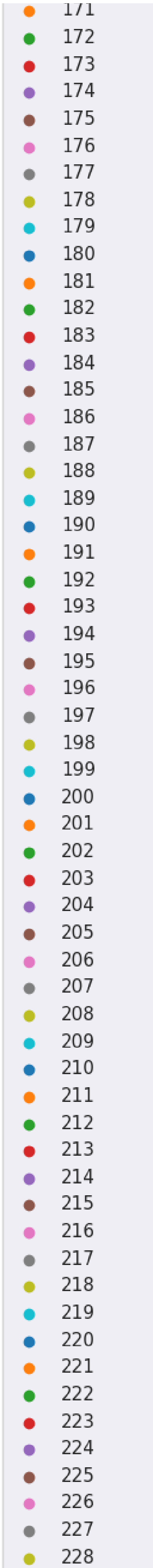
```
/mnt/Datos/.enviroments/ubu20/ML-E1/lib/python3.8/site-packages/sklearn/clust
er/_affinity_propagation.py:143: ConvergenceWarning: Affinity propagation did
not converge, this model may return degenerate cluster centers and labels.
  warnings.warn(
```



Affinity Propagation
8
6
4
2
0
−2
−4
−6
−4
−2
0
2
4
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

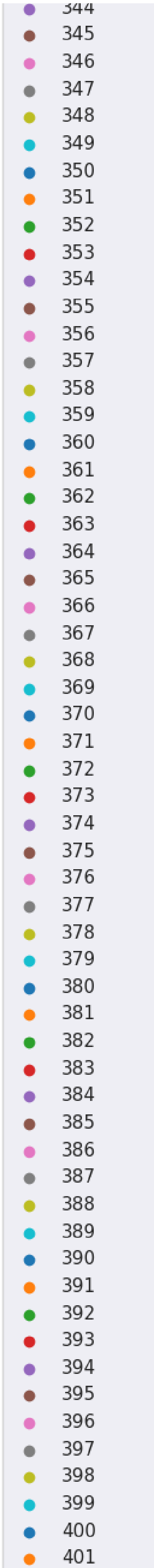

56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112


114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170


171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228


229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286

287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343


344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401


402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459

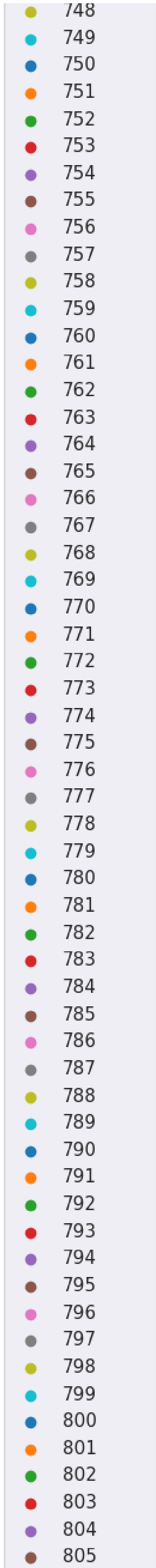

460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516


518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574


575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632


633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689


691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747


748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805


806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
Centroids

Evaluation

In [22]:
```
def get_accuracy(y, predictions):
    allPermutations=np.array(list(permutations(np.unique(y))))
    acc=[]
    for perm in allPermutations:
        classes=np.arange(0,y.shape[0])
        for index in range(classes.shape[0]):
            classes[index]=np.where(y[index]==perm)[0][0]
        acc.append(np.sum(classes==predictions))
    acc=np.array(acc)
    bestAccIndex=np.where(max(acc)==acc)[0][0]
    return dict(zip(np.arange(0,allPermutations.shape[1]),allPermutations[be
```

In [23]:
```
def k_means_k_fold(X, Y, k, max_iter=100, tol=1e-3, splits=5):
    """Implementation of the k-means algorithm with k-fold cross validation.

    :param np.ndarray X: Data to cluster.
    :param np.ndarray Y: Labels.
    :param int k: Number of clusters.
    :param int max_iter: Maximum number of iterations. Defaults to 100.
    :param float tol: Tolerance. Defaults to 1e-3.
    :param int splits: Number of splits for k-fold cross validation. Default
    """
    kf = KFold(n_splits=splits)
    accuracies = []

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        Y_train, Y_test = Y[train_index], Y[test_index]

        _, centroids = clustering.k_means(X_train, k, max_iter=max_iter, tol

        Y_predict = clustering.k_means_predict(X_test, centroids)
        _, accuracy = get_accuracy(Y_test, Y_predict)
        accuracies.append((accuracy/Y_test.shape[0])*100)

    return np.array(accuracies)
```

In [24]:
```
def boxplot(data: pd.DataFrame, title: str, filename: str):
    """Plots a nice boxplot.

    :param pd.DataFrame data: Data to plot.
    :param str title: Title of the plot.
    :param str filename: Filename to save the plot.
    """
    fig, ax = plt.subplots(figsize=(9, 4))
    ax.set_title(title)
    ax.set_ylabel("Accuracy [%]")
    sns.boxplot(data=data, ax=ax)

    decribed_data = data.describe().T
    for x, wt in enumerate(ax.get_xticklabels()):
        median_ = decribed_data.loc[wt.get_text(), '50%']
```

```
        min_ = decribed_data.loc[wt.get_text(), 'min']
        max_ = decribed_data.loc[wt.get_text(), 'max']
        q1_ = decribed_data.loc[wt.get_text(), '25%']
        q3_ = decribed_data.loc[wt.get_text(), '75%']

        ax.text(x, median_, f'{median_:.4f}', horizontalalignment='center',
                color='w', backgroundcolor='k', weight='semibold', fontsize=
        ax.text(x, min_, f'{min_:.4f}', horizontalalignment='center',
                color='w', backgroundcolor='k', weight='semibold', fontsize=
        ax.text(x, max_, f'{max_:.4f}', horizontalalignment='center',
                color='w', backgroundcolor='k', weight='semibold', fontsize=
        ax.text(x, q1_, f'{q1_:.4f}', horizontalalignment='center',
                color='w', backgroundcolor='k', weight='semibold', fontsize=
        ax.text(x, q3_, f'{q3_:.4f}', horizontalalignment='center',
                color='w', backgroundcolor='k', weight='semibold', fontsize=

    plt.savefig(f"{filename}.png")
    plt.show()
```

K-Means with all features

In [25]:
```
all_features_accuracies = k_means_k_fold(X, Y, 4, max_iter=100, tol=1e-3, sp
print(f"Mean accuracy: {all_features_accuracies.mean():.2f}%")
```

```
 46%|████▌     | 46/100 [00:00<00:00, 429.32it/s]
 36%|███▌      | 36/100 [00:00<00:00, 487.41it/s]
 42%|████▏     | 42/100 [00:00<00:00, 487.94it/s]
 30%|███       | 30/100 [00:00<00:00, 481.39it/s]
 70%|███████   | 70/100 [00:00<00:00, 481.42it/s]
 55%|█████▌    | 55/100 [00:00<00:00, 470.20it/s]
 46%|████▌     | 46/100 [00:00<00:00, 481.56it/s]
 32%|███▏      | 32/100 [00:00<00:00, 480.52it/s]
 34%|███▍      | 34/100 [00:00<00:00, 485.09it/s]
 23%|██▎       | 23/100 [00:00<00:00, 479.39it/s]
Mean accuracy: 36.42%
```

K-Means with relevant features

In [26]:
```
relevant_features_accuracies = k_means_k_fold(X_relevant, Y, 4, max_iter=100
print(f"Mean accuracy: {relevant_features_accuracies.mean():.2f}%")
```

```
 77%|███████▋  | 77/100 [00:00<00:00, 667.78it/s]
 13%|█▎        | 13/100 [00:00<00:00, 432.60it/s]
 68%|██████▊   | 68/100 [00:00<00:00, 654.50it/s]
100%|██████████| 100/100 [00:00<00:00, 675.35it/s]
 63%|██████▎   | 63/100 [00:00<00:00, 671.95it/s]
100%|██████████| 100/100 [00:00<00:00, 686.43it/s]
100%|██████████| 100/100 [00:00<00:00, 665.78it/s]
100%|██████████| 100/100 [00:00<00:00, 690.27it/s]
 74%|███████▍  | 74/100 [00:00<00:00, 685.27it/s]
 15%|█▌        | 15/100 [00:00<00:00, 638.85it/s]
Mean accuracy: 30.55%
```

K-Means with PCA features

In [27]:
```
pca_features_accuracies = k_means_k_fold(X_pca, Y, 4, max_iter=100, tol=1e-3
print(f"Mean accuracy: {pca_features_accuracies.mean():.2f}%")
```

```
 22%|██▏        | 22/100 [00:00<00:00, 602.22it/s]
 31%|███        | 31/100 [00:00<00:00, 681.11it/s]
 19%|█▉         | 19/100 [00:00<00:00, 666.91it/s]
 36%|███▌       | 36/100 [00:00<00:00, 687.26it/s]
 19%|█▉         | 19/100 [00:00<00:00, 664.97it/s]
 31%|███        | 31/100 [00:00<00:00, 684.78it/s]
 53%|█████▎     | 53/100 [00:00<00:00, 698.08it/s]
 34%|███▍       | 34/100 [00:00<00:00, 689.15it/s]
 22%|██▏        | 22/100 [00:00<00:00, 672.66it/s]
 24%|██▍        | 24/100 [00:00<00:00, 674.43it/s]
Mean accuracy: 34.35%
```

Comparsion

In [28]:
```
results = pd.DataFrame(
    np.array([all_features_accuracies, relevant_features_accuracies, pca_fea
    columns=["all_features", "relevant_features", "pca_features"]
)
results.head(10)

boxplot(results, "Accuracy of each K-Means method", FIGURES_PATH + "k_means_
```