Project

Imports

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy
import os
import pandas as pd
import plotly.io as pio
import seaborn as sns
%matplotlib inline
```

In [2]:
```
from utils import *
from handle_dataset import *
from density import *
from zscore import *
from plots import *
from kmeans import *
from linear_regression import *
from distributions import *
from extract_features import *
```

In [3]:
```
DATA_PATH = 'data'
```

In [4]:
```
RESULTS_PATH = 'results'
```

In [5]:
```
activities_labels = {
    1 : 'STAND',
    2 : 'SIT',
    3 : 'SIT&TALK',
    4 : 'WALK',
    5 : 'WALK&TALK',
    6 : 'CLIMB_STAIRS',
    7 : 'CLIMB_STAIRS&TALK',
    8 : 'STAND_TO_SIT',
    9 : 'SIT_TO_STAND',
    10: 'STAND_TO_SIT&TALK',
    11: 'SIT&TALK_TO_STAND',
    12: 'STAND_TO_WALK',
    13: 'WALK_TO_STAND',
    14: 'STAND_TO_CLIMB_STAIRS',
    15: 'CLIMB_STAIRS_TO_WALK',
    16: 'CLIMB_STAIRS&TALK_TO_WALK&TALK',

}
```

In [6]:
```
labels = [
    'STAND',
    'SIT',
    'SIT&TALK',
    'WALK',
    'WALK&TALK',
    'CLIMB_STAIRS',
    'CLIMB_STAIRS&TALK',
    'STAND_TO_SIT',
    'SIT_TO_STAND',
    'STAND_TO_SIT&TALK',
```

```
    'SIT&TALK_TO_STAND',
    'STAND_TO_WALK',
    'WALK_TO_STAND',
    'STAND_TO_CLIMB_STAIRS',
    'CLIMB_STAIRS_TO_WALK',
    'CLIMB_STAIRS&TALK_TO_WALK&TALK',

]
```

In [7]:
```
import warnings
warnings.filterwarnings("ignore")
```

Exercise 2 - Import dataset

In [8]:
```
# data_user_0 = get_user_data(path=DATA_PATH, user_id=0)
```

In [9]:
```
dataset = get_all_data(DATA_PATH)
```

In [10]:
```
dataset.shape
```

Out[10]:
```
(3242728, 12)
```

In [11]:
```
dataset.shape
```

Out[11]:
```
(3242728, 12)
```

Explore data

In [12]:
```
dataset.describe()
```

Out[12]:

	device_id	accelerometer_x	accelerometer_y	accelerometer_z	gyroscope_x	gyroscope_y	gyrosco
count	3.242728e+06	3.242728e+06	3.242728e+06	3.242728e+06	3.242728e+06	3.242728e+06	3.242728e
mean	3.429246e+00	7.717793e-01	8.451621e+00	1.520613e+00	-1.691060e-01	1.890623e+00	3.556803
std	1.188858e+00	2.494840e+00	3.743438e+00	3.747458e+00	3.592437e+01	5.289265e+01	4.621969e
min	1.000000e+00	-2.489000e+01	-2.324600e+01	-2.502500e+01	-5.088600e+02	-5.071500e+02	-4.077900e
25%	2.000000e+00	-5.244600e-01	8.352300e+00	-1.328500e+00	-2.967200e+00	-2.829200e+00	-1.415125e
50%	3.000000e+00	3.722300e-01	9.388200e+00	1.323900e+00	1.709100e-01	-1.321600e-01	-1.696150
75%	4.000000e+00	2.006400e+00	9.801800e+00	3.299600e+00	5.930100e+00	8.948325e+00	5.541700e
max	5.000000e+00	2.538100e+01	2.598500e+01	2.529000e+01	5.061000e+02	5.035700e+02	4.741600e

In [13]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3242728 entries, 0 to 3242727
Data columns (total 12 columns):
 #   Column           Dtype
---  ------           -----
 0   device_id        int64
 1   accelerometer_x  float64
 2   accelerometer_y  float64
 3   accelerometer_z  float64
 4   gyroscope_x      float64
```

```
 5   gyroscope_y      float64
 6   gyroscope_z      float64
 7   magnetometer_x   float64
 8   magnetometer_y   float64
 9   magnetometer_z   float64
 10  timestamp        float64
 11  activity         int64
dtypes: float64(10), int64(2)
memory usage: 296.9 MB
```

Append metrics

In [14]:
```
data_with_features = append_metrics(dataset)
```

In [15]:
```
data_with_features
```

Out[15]:

	device_id	accelerometer_x	accelerometer_y	accelerometer_z	gyroscope_x	gyroscope_y	gyroscope_z	n
0	1	-1.8650	9.3890	2.58120	-1.141800	-1.18560	0.84998	
1	1	-1.7963	9.3742	2.44600	-1.561800	-0.66165	0.59730	
2	1	-1.8696	9.3000	2.35140	-1.187700	-1.28410	0.14212	
3	1	-1.7961	9.3624	2.45840	-0.583990	-2.03340	0.42912	
4	1	-1.6768	9.3506	2.46850	-0.370500	-1.36470	0.37194	
...	...	...	...	...	...	...	...	...
3242723	5	-1.0568	9.7161	0.37560	-0.158430	-0.88740	0.44327	
3242724	5	-1.0565	9.6921	0.38826	0.112400	-0.59353	0.39687	
3242725	5	-1.0566	9.6920	0.37607	-0.054114	-0.67227	0.36407	
3242726	5	-1.0563	9.6801	0.38850	0.096447	-0.51923	0.16742	
3242727	5	-1.0686	9.6920	0.38851	0.171850	-0.45506	0.36981	

3242728 rows × 15 columns

Select only right wrist data

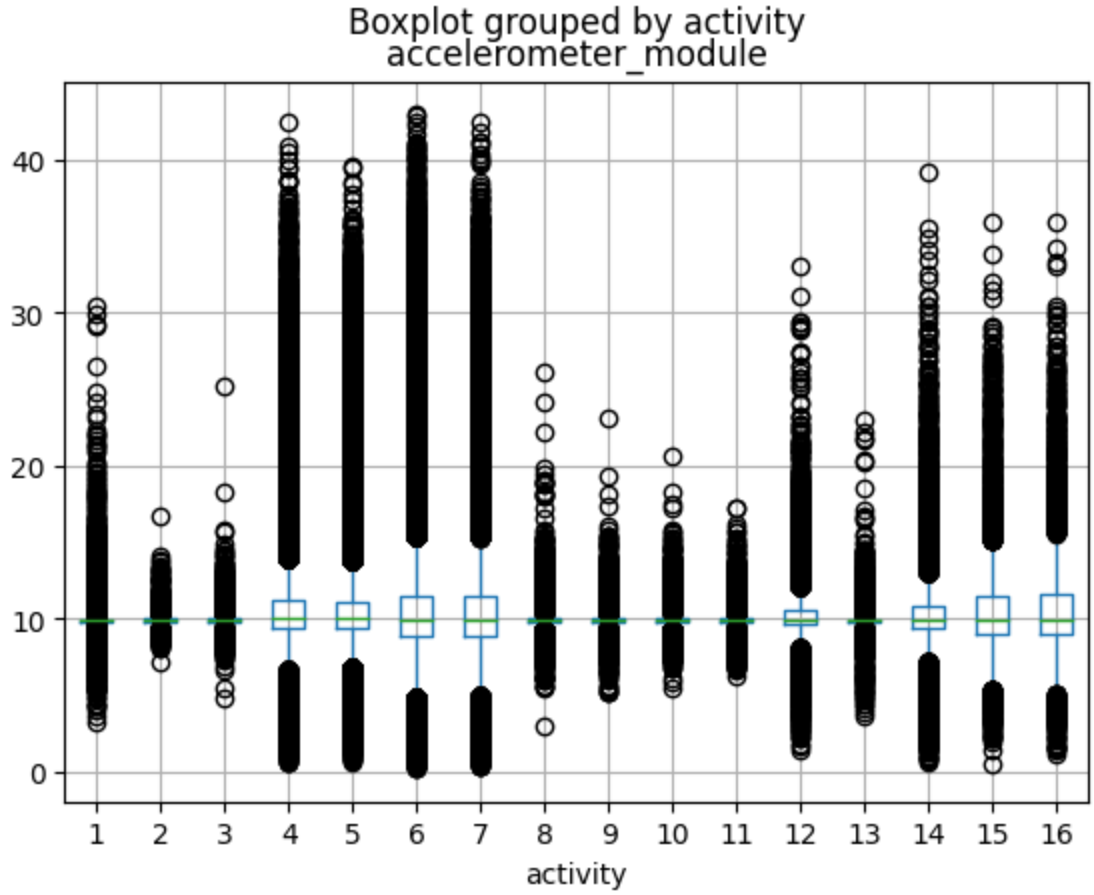
Nos seguintes exercícios, apenas serão usados os dados do pulso direito.

In [16]:
```
right_wrist_id = 2
data = get_device_data(data_with_features, right_wrist_id)
right_wrist_data = data[['accelerometer_module', 'gyroscope_module', 'magnetometer_modul
right_features = right_wrist_data[['accelerometer_module','gyroscope_module', 'magnetome
```

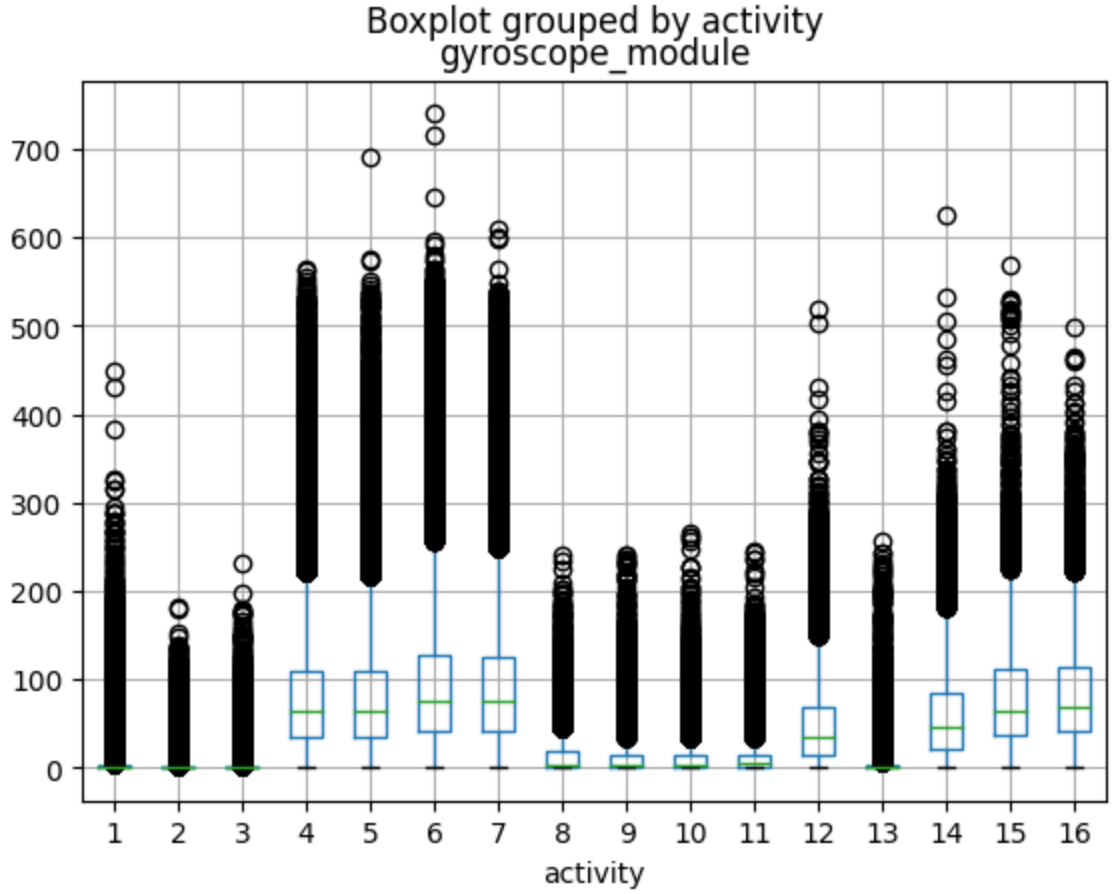
Exercise 3.1 - Boxplot of the vector modules grouped by activity

In [17]:
```
boxplot_features(data_with_features, 'accelerometer_module')
boxplot_features(data_with_features, 'gyroscope_module')
boxplot_features(data_with_features, 'magnetometer_module')
```

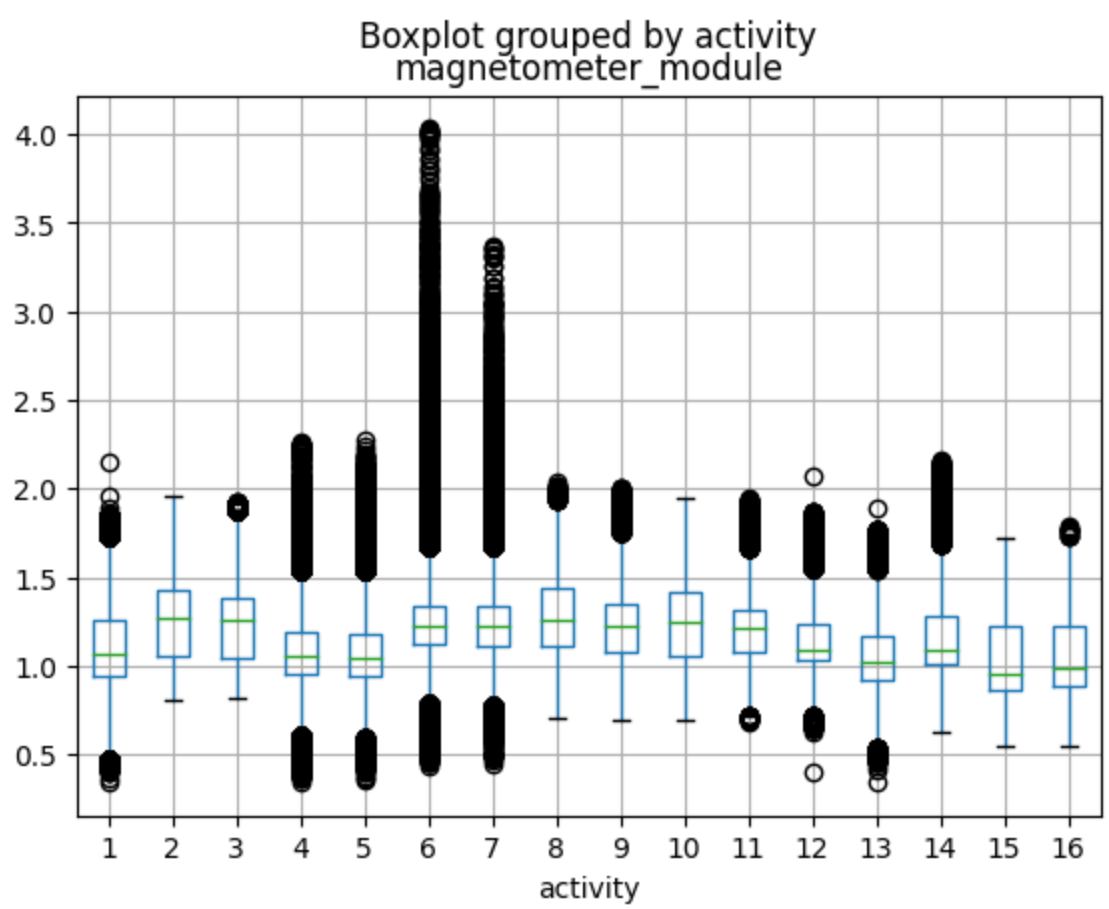
```
<Figure size 2100x1000 with 0 Axes>
```



<Figure size 2100x1000 with 0 Axes>



<Figure size 2100x1000 with 0 Axes>



Pelos boxplot dos módulos dos diferentes vetores podemos concluir que:

- os três módulos (e respetivos vetores) ordens de grandeza muito diferentes
- existe uma grande densidade de outliers. Isto pode dever-se a estarem a ser utilizados neste gráfico todos os dispositivos de sensores, o que poderá afetar os resultados.

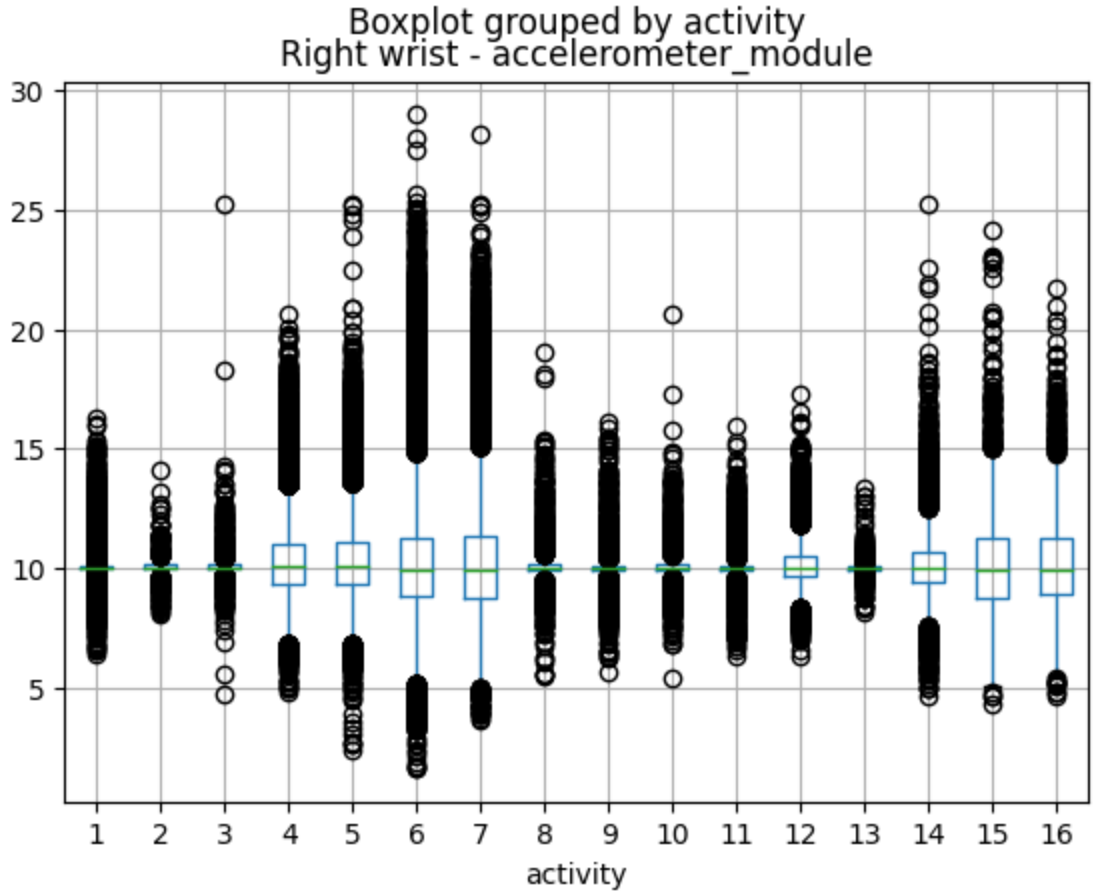
Plotting boxplot of the modules by activity separated by devices

Boxplot core

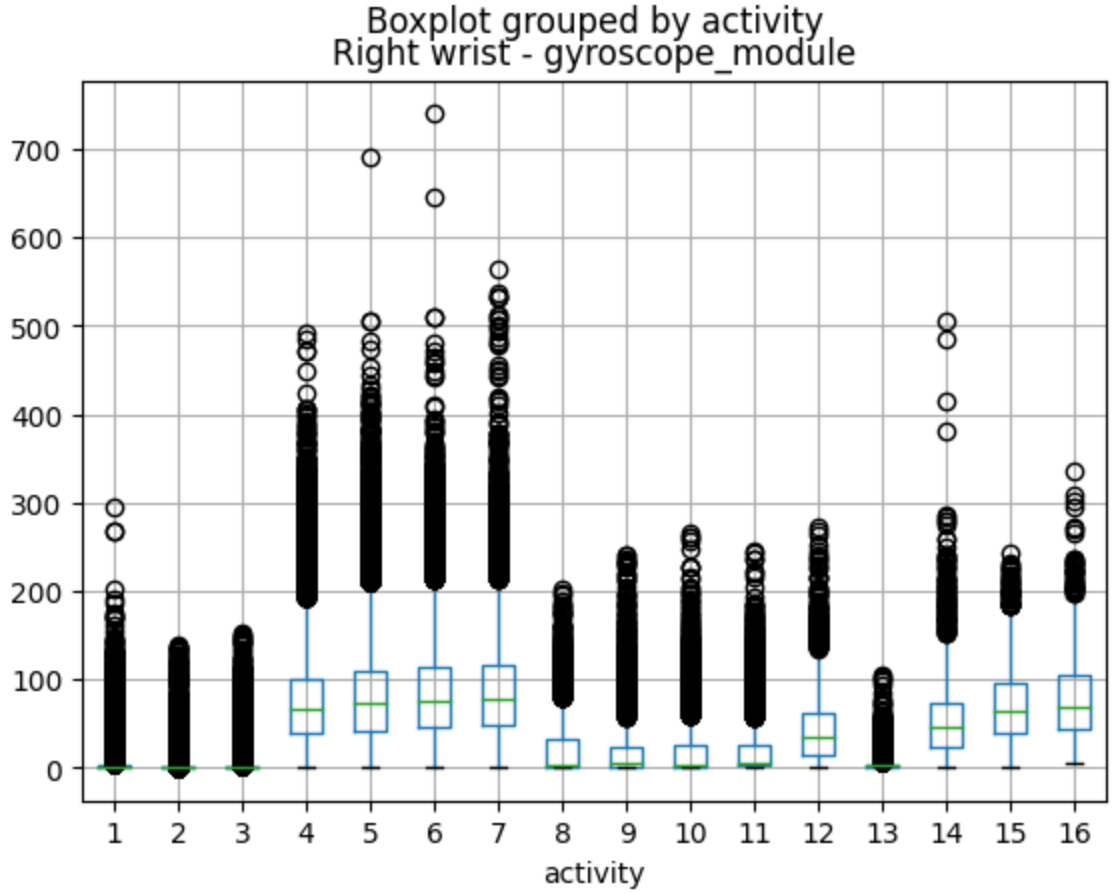
Plot of the right wrist (id = 2)

In [18]:
```
boxplot_features(right_wrist_data, 'accelerometer_module', title='Right wrist')
boxplot_features(right_wrist_data, 'gyroscope_module', title='Right wrist')
boxplot_features(right_wrist_data, 'magnetometer_module', title='Right wrist')
```

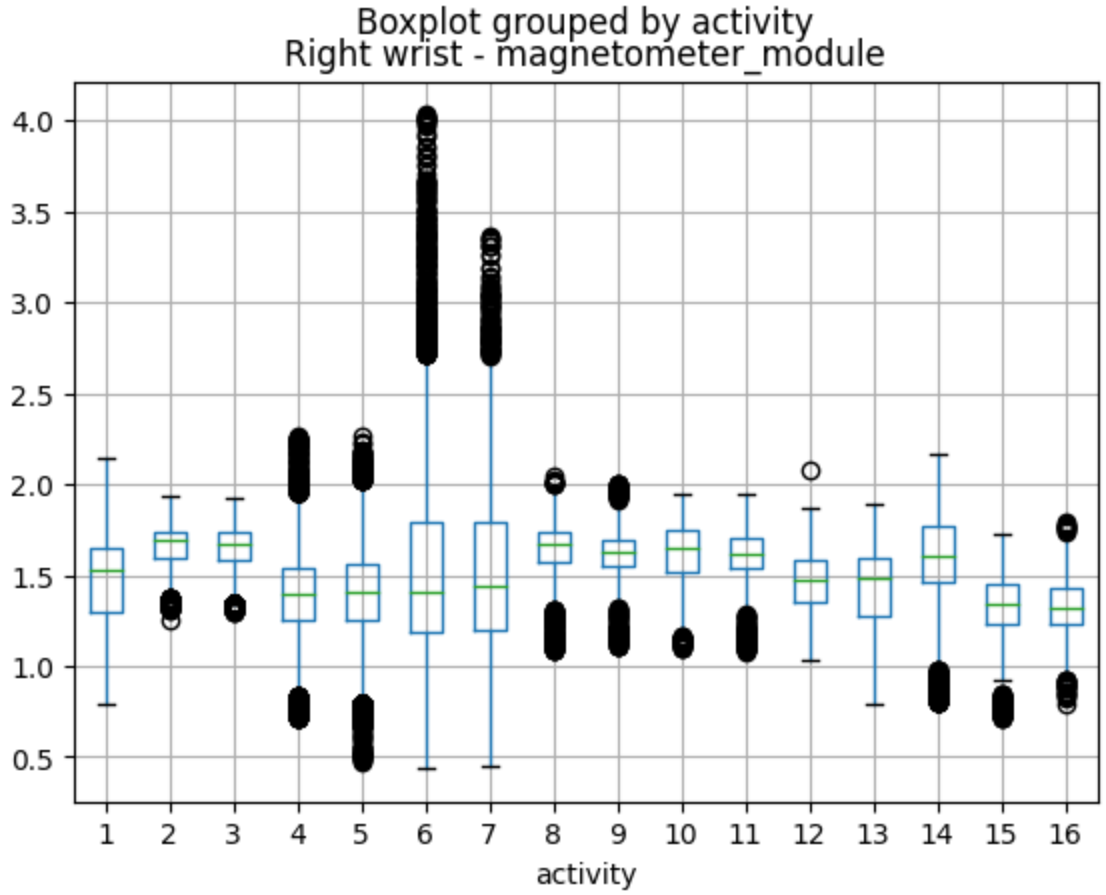
```
<Figure size 2100x1000 with 0 Axes>
```



<Figure size 2100x1000 with 0 Axes>



<Figure size 2100x1000 with 0 Axes>



Ao utilizar os dados de apenas um dispositivo (dados medido no pulso direito), é notório que a quantidade de outliers diminui consideravelmente.

In [19]:
```
right_wrist_data.describe()
```

Out[19]:

	accelerometer_module	gyroscope_module	magnetometer_module	activity
count	781822.000000	781822.000000	781822.000000	781822.000000
mean	10.179602	44.511468	1.484294	4.055064
std	1.342380	52.610313	0.260959	2.585601
min	1.614946	0.000000	0.435554	1.000000
25%	9.845890	1.142540	1.306523	2.000000
50%	10.014603	24.413977	1.519626	4.000000
75%	10.260752	77.157773	1.679282	5.000000
max	28.986407	740.443292	4.033180	16.000000

Exercise 3.2 - Outliers density

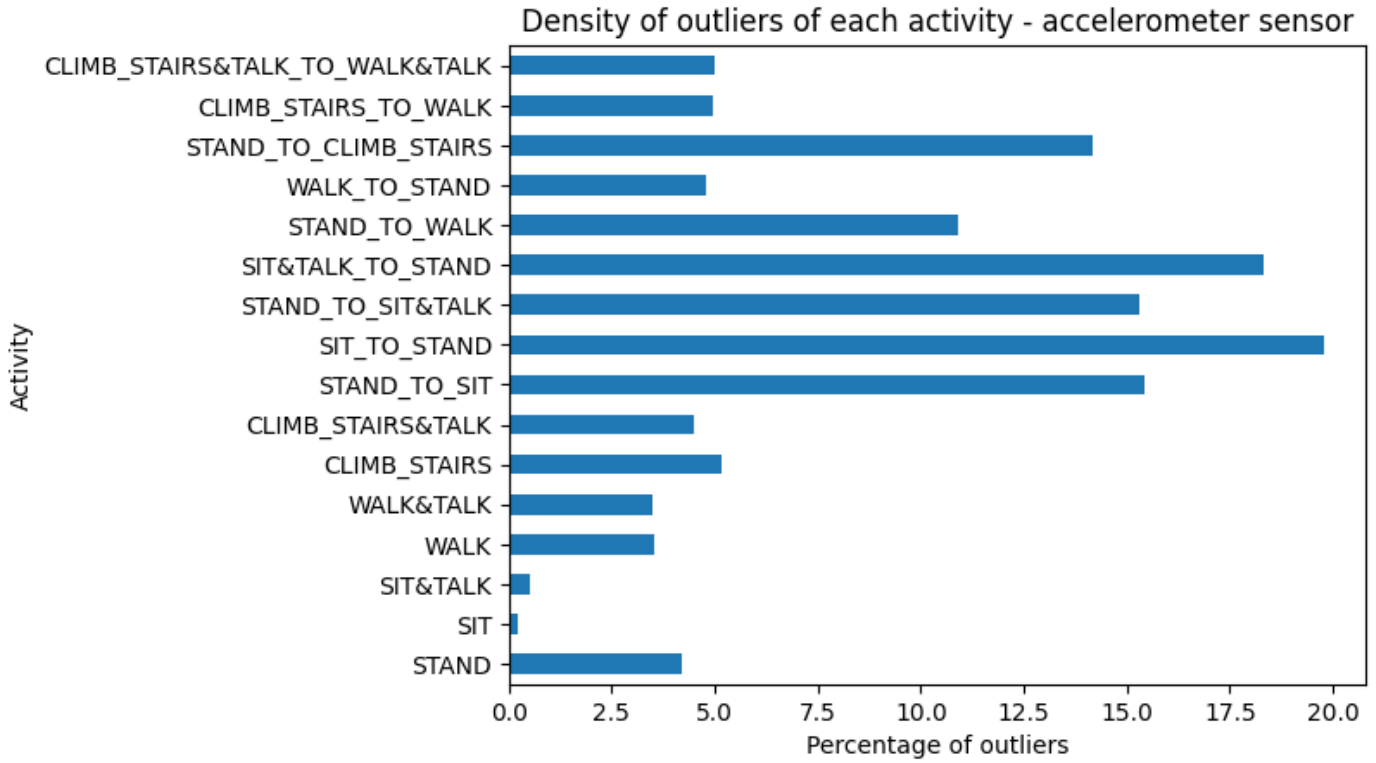
In [20]:
```
densities = calculate_density_by_activity(data, labels)
densities
```

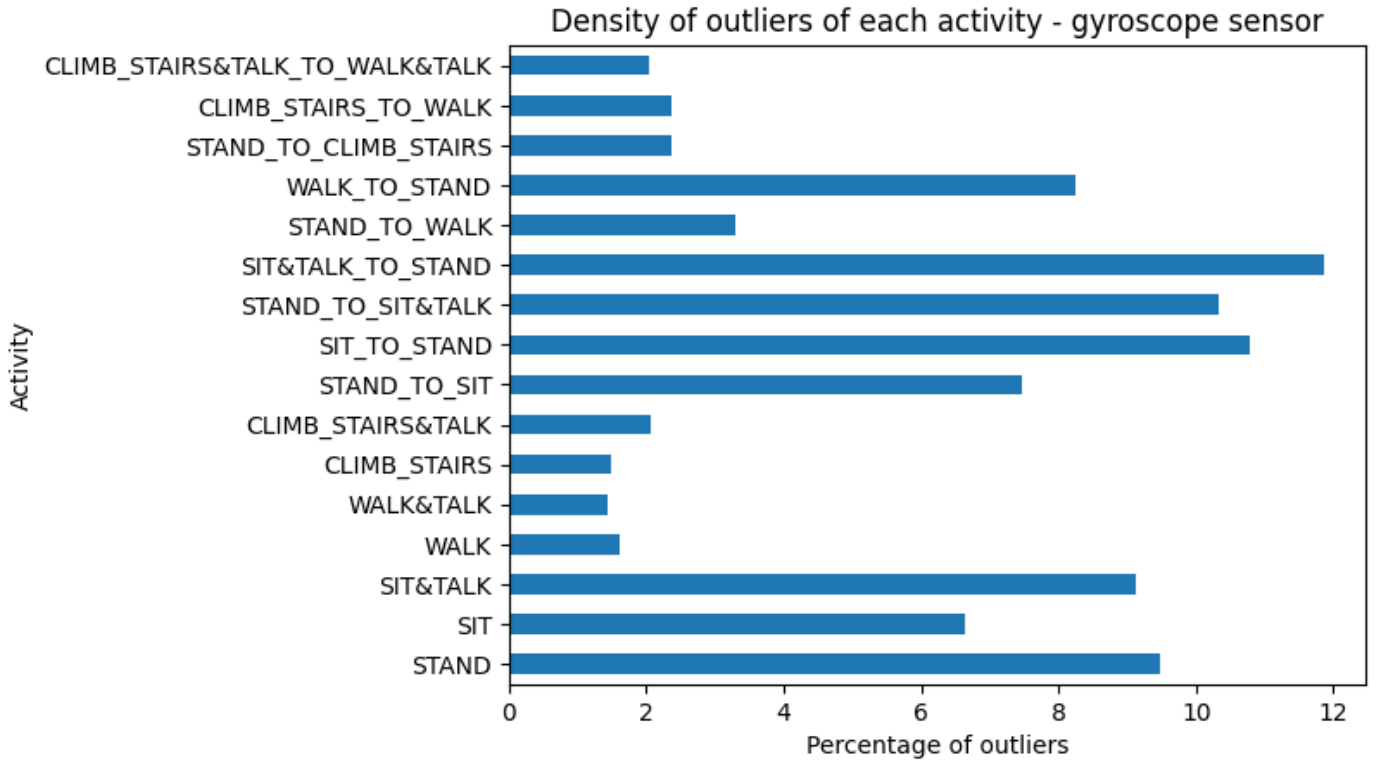
Out[20]:

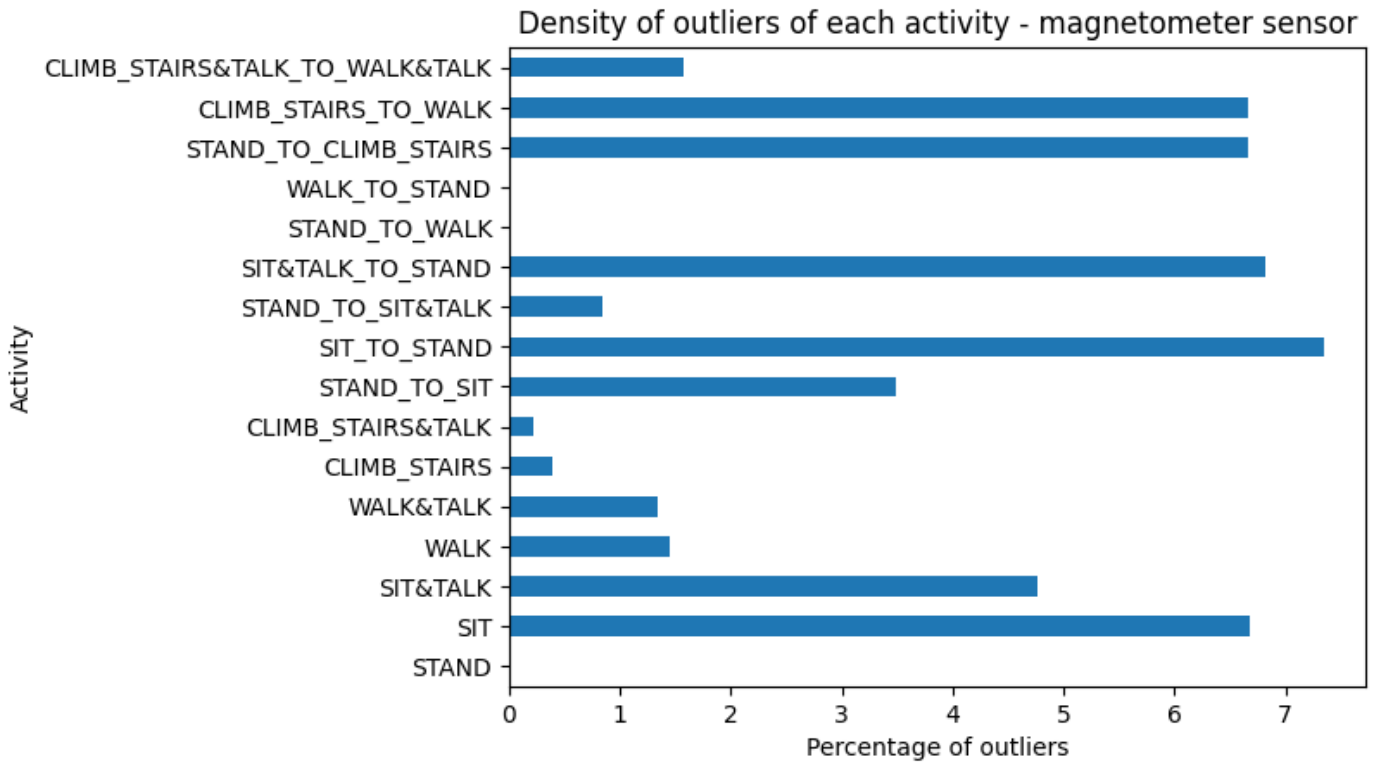
	STAND	SIT	SIT&TALK	WALK	WALK&TALK	CLIMB_STAIRS	CLIMB_STAIRS&TALI
accelerometer_module	4.195538	0.215558	0.513851	3.530950	3.477656	5.165680	4.51462
gyroscope_module	9.484932	6.646580	9.117459	1.620764	1.428009	1.498450	2.07348

magnetometer_module	0.000000	6.672780	4.766364	1.450684	1.338803	0.390559	0.22150

In [21]:
```
plot_densities(densities)
```



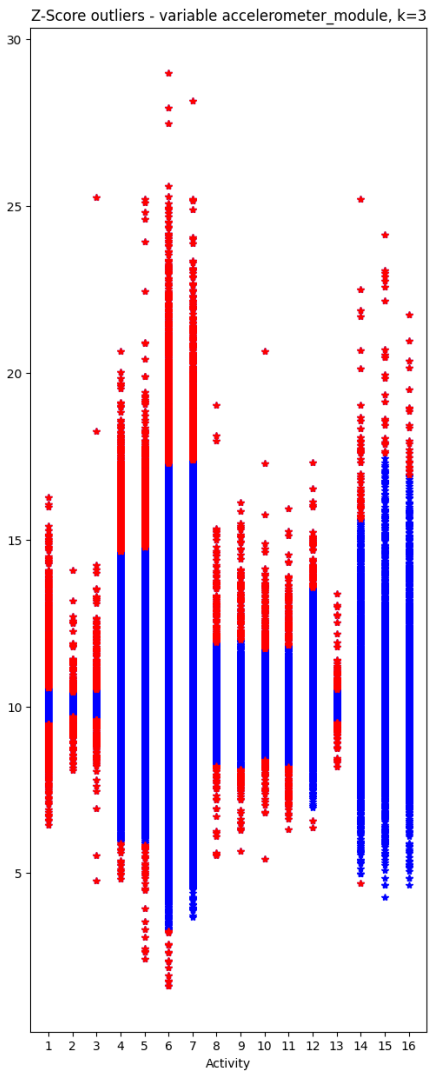


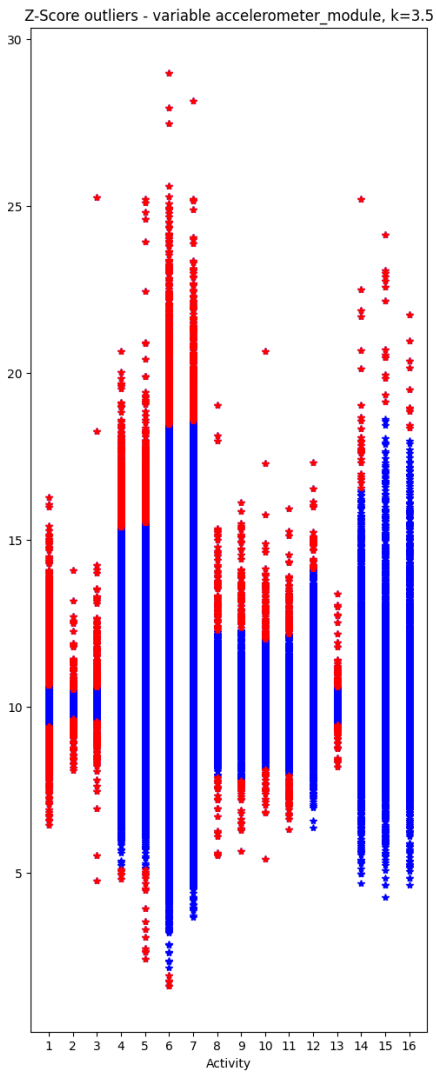


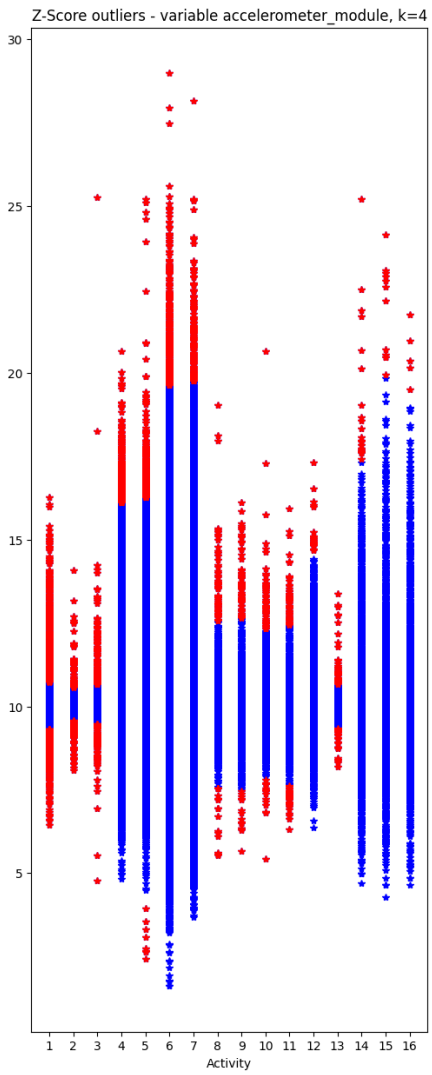
Como se pode observar pelas densidades de outliers obtidas, o vetor de aceleração é o que apresenta maior quantidade de outliers. Isto poderá dever-se ao facto de ser o mais sensível a movimentos da pessoa. Pelo contrário, o vetor de variação do campo magnético, em algumas situações nem apresenta outliers. Outra conclusão que podemos retirar do gráfico, é que as atividades *Sit to stand* e *Sit and talk to stand* são as que apresentam maior quantidade de outliers nos três vetores, mostrando que o movimento de levantar gera muitas oscilações e perturbações nos sensores.

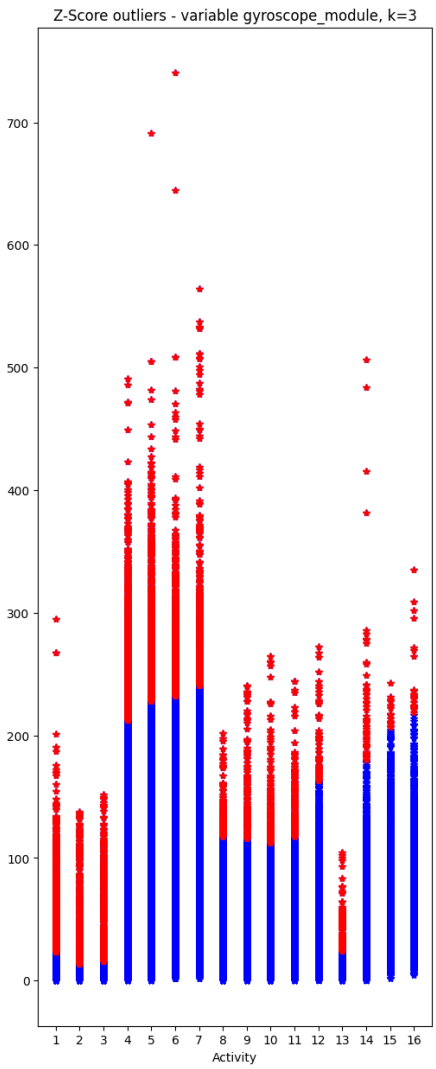
Exercise 3.3 - Z-score test

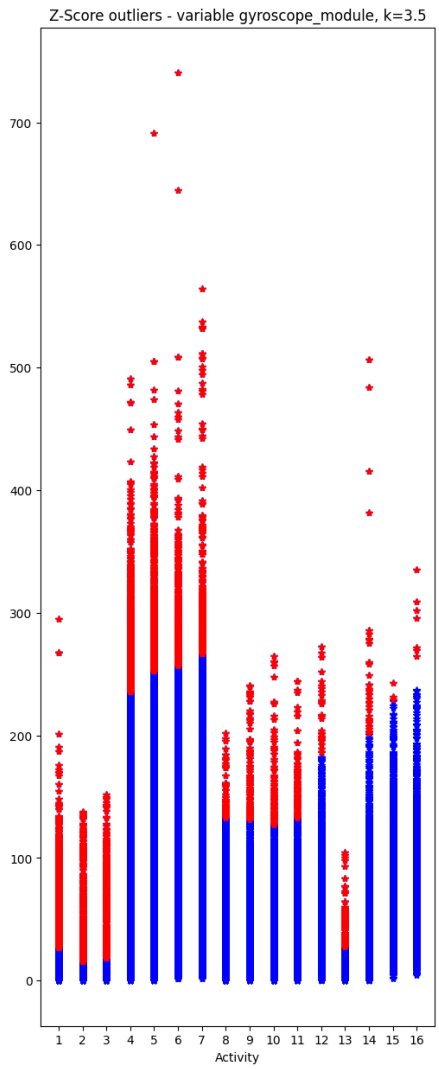
In [22]:
```
plot_zscore_outliers(data,'accelerometer_module')
plot_zscore_outliers(data,'gyroscope_module')
plot_zscore_outliers(data,'magnetometer_module')
```

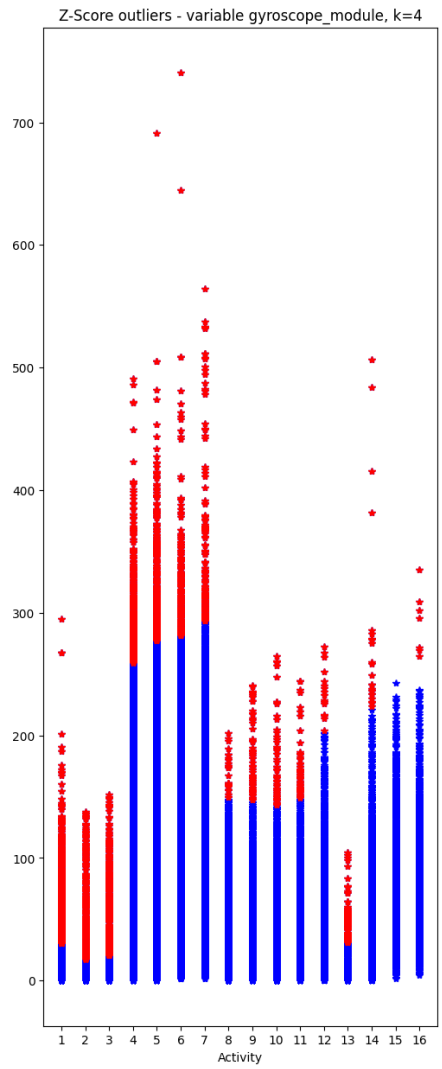

Z-Score outliers - variable accelerometer_module, k=3
30
25
20
15
10
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Activity

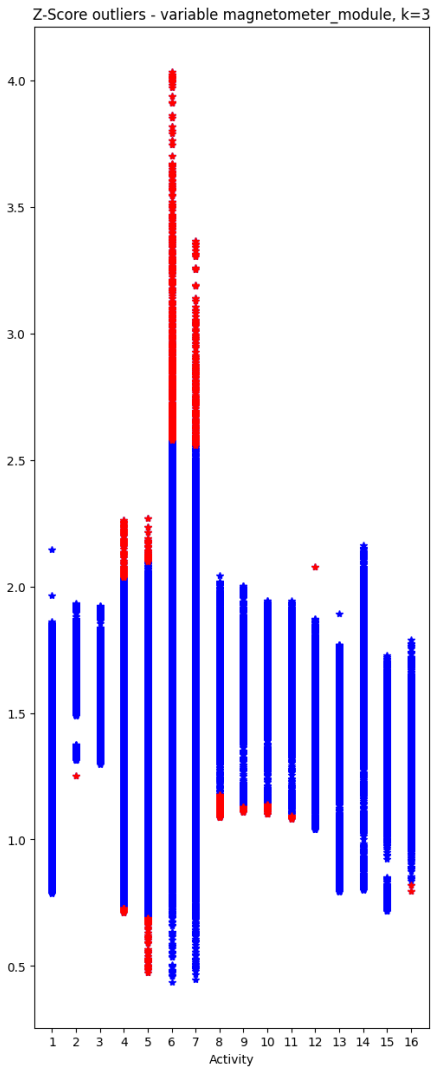

Z-Score outliers - variable accelerometer_module, k=3.5
30
25
20
15
10
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Activity

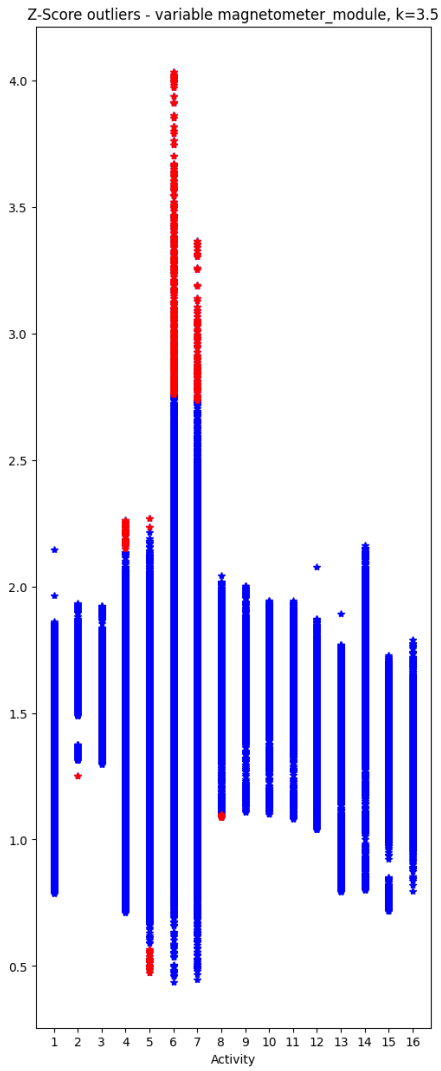

Z-Score outliers - variable accelerometer_module, k=4
30
25
20
15
10
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Activity

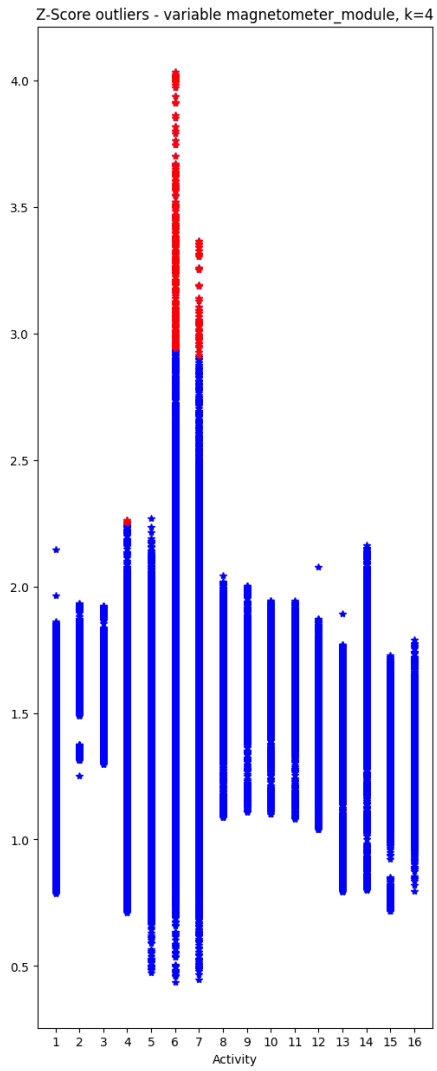

Z-Score outliers - variable gyroscope_module, k=3
Activity


Z-Score outliers - variable gyroscope_module, k=3.5
Activity

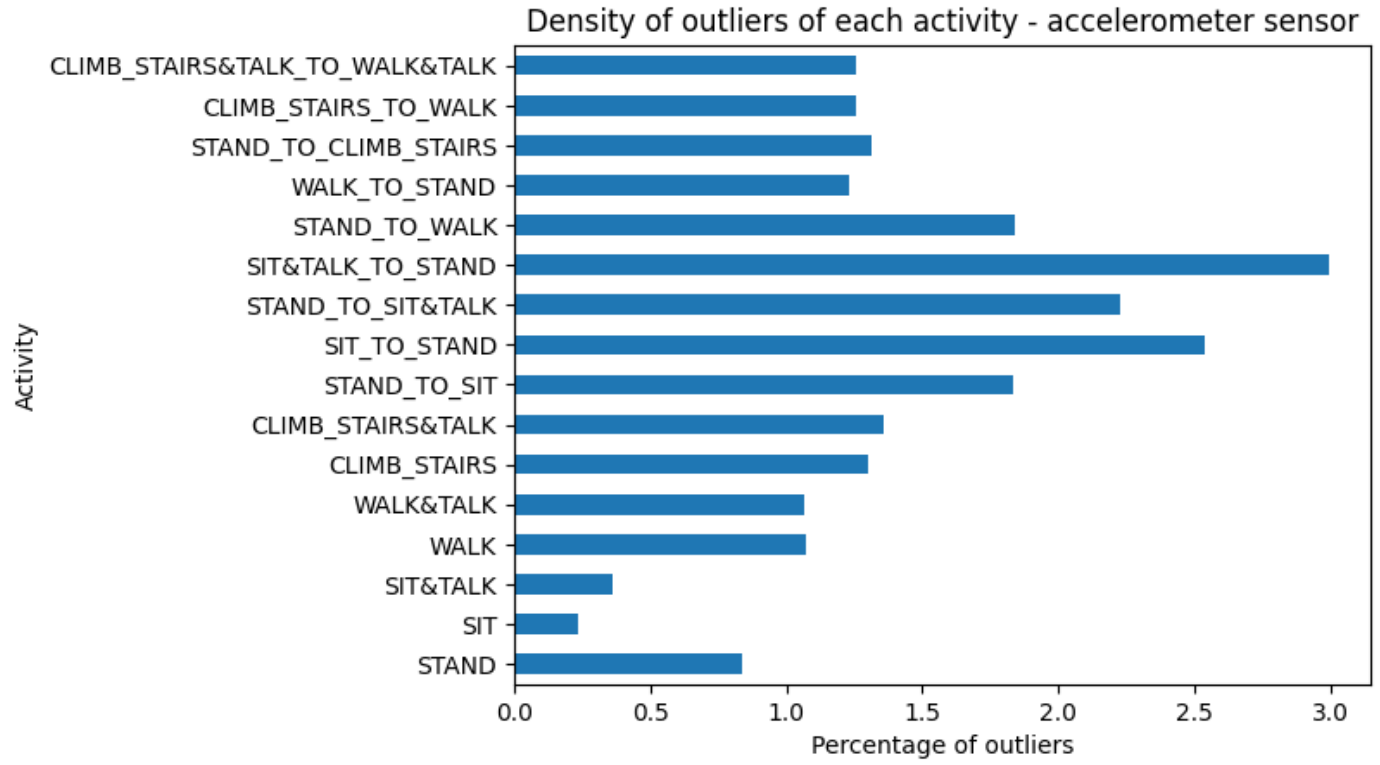

Z-Score outliers - variable gyroscope_module, k=4
Activity

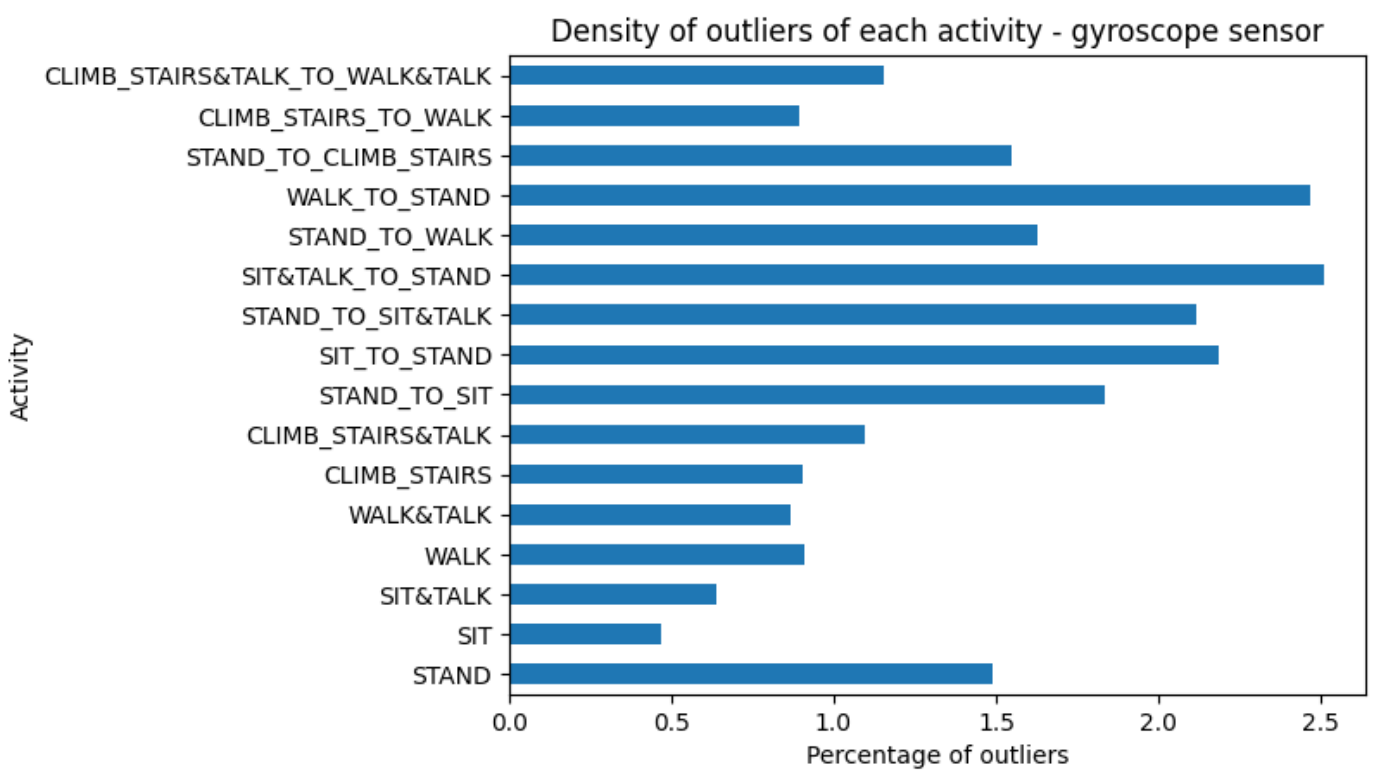


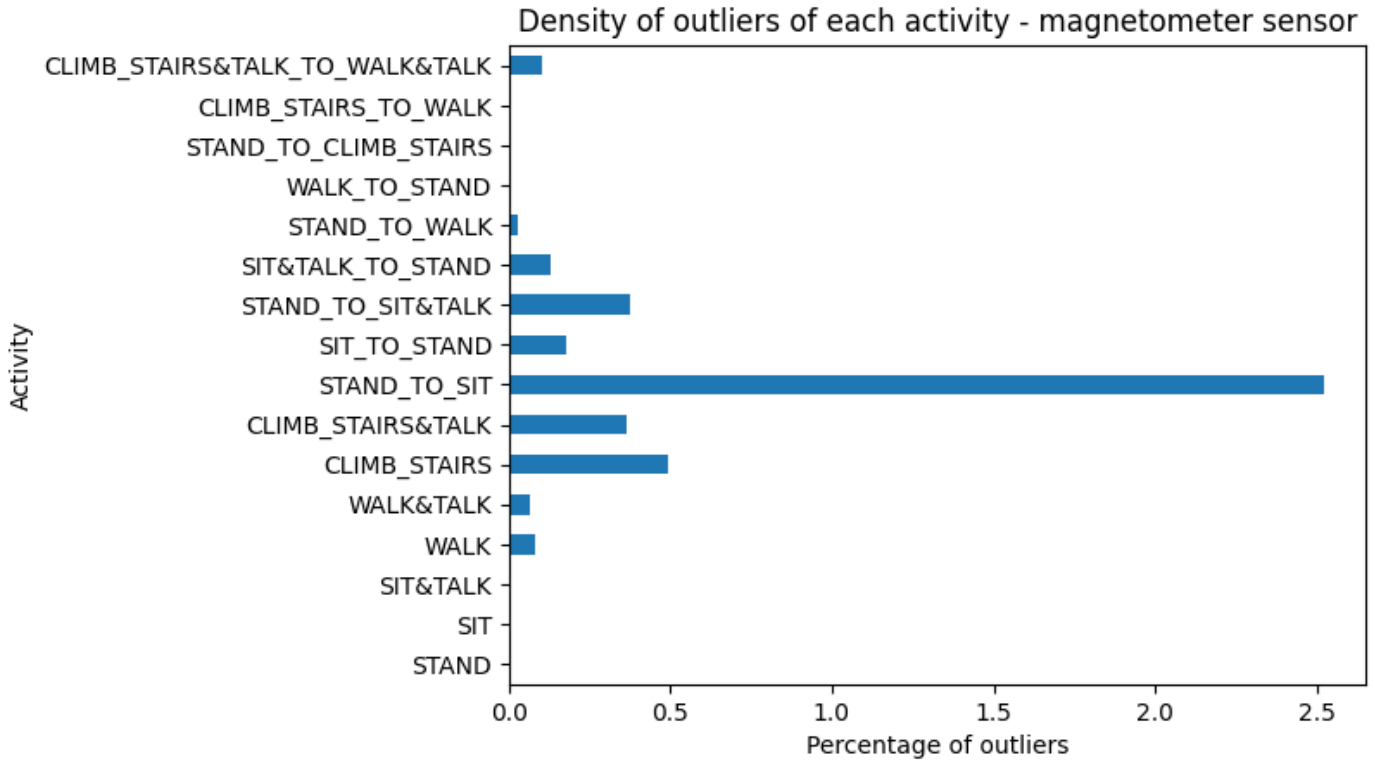




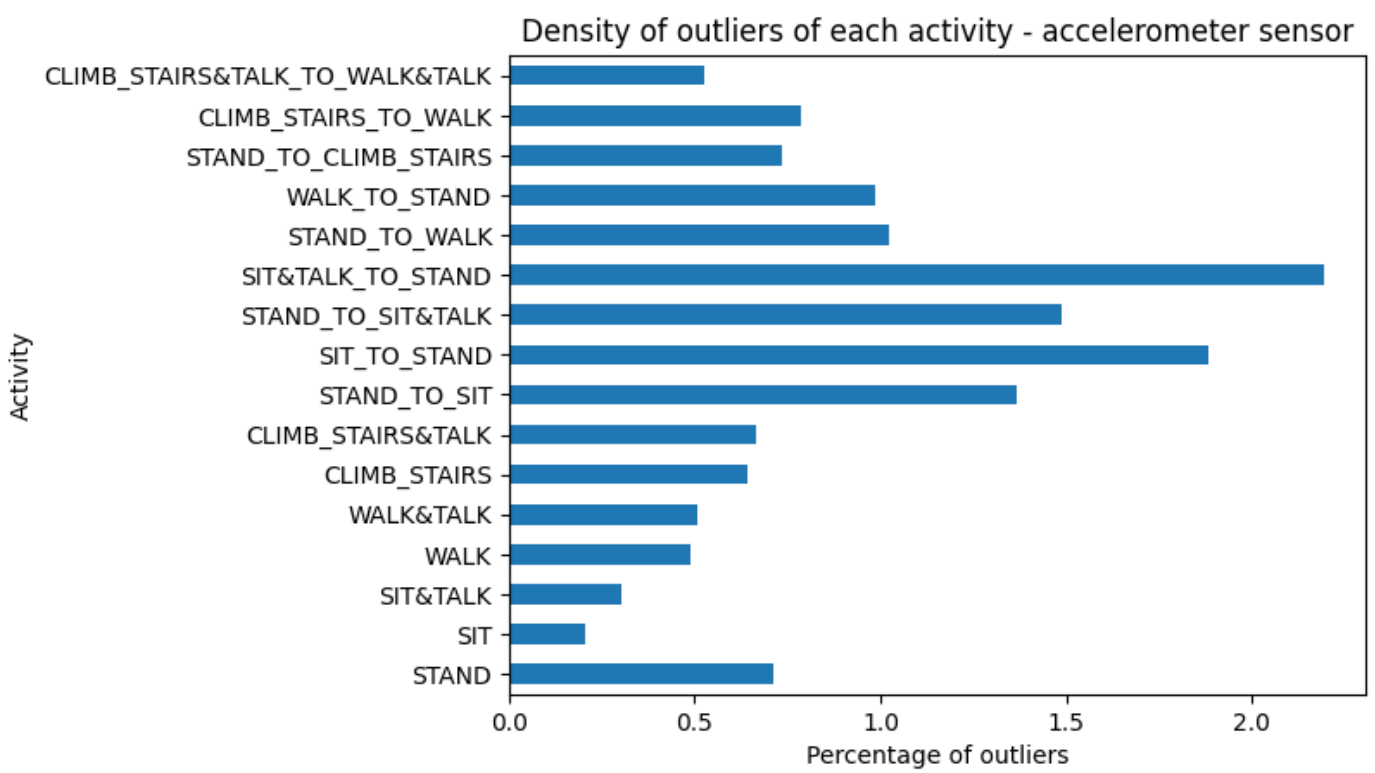
In [23]:
```
densities_zscore_3 = calculate_zscore_densities(data, labels, 3)
plot_densities(densities_zscore_3)
```

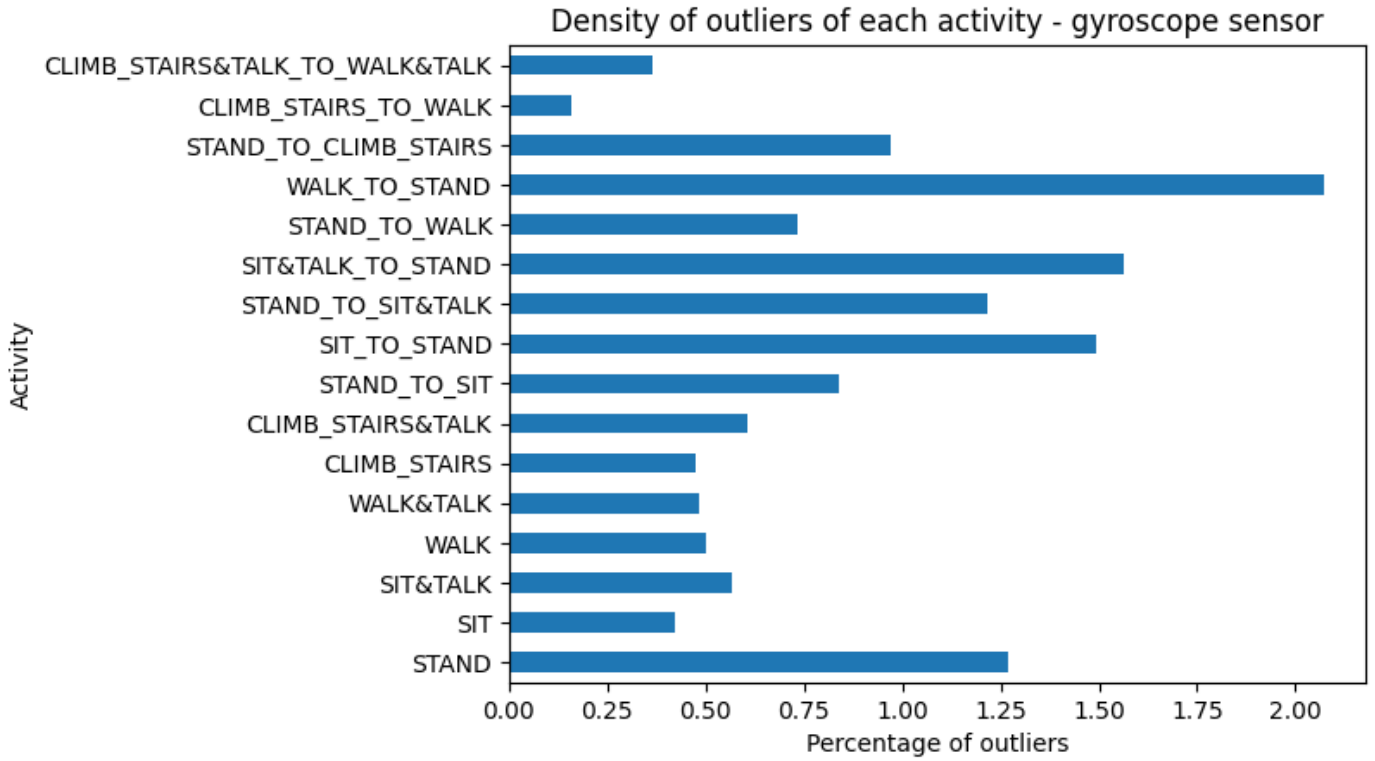


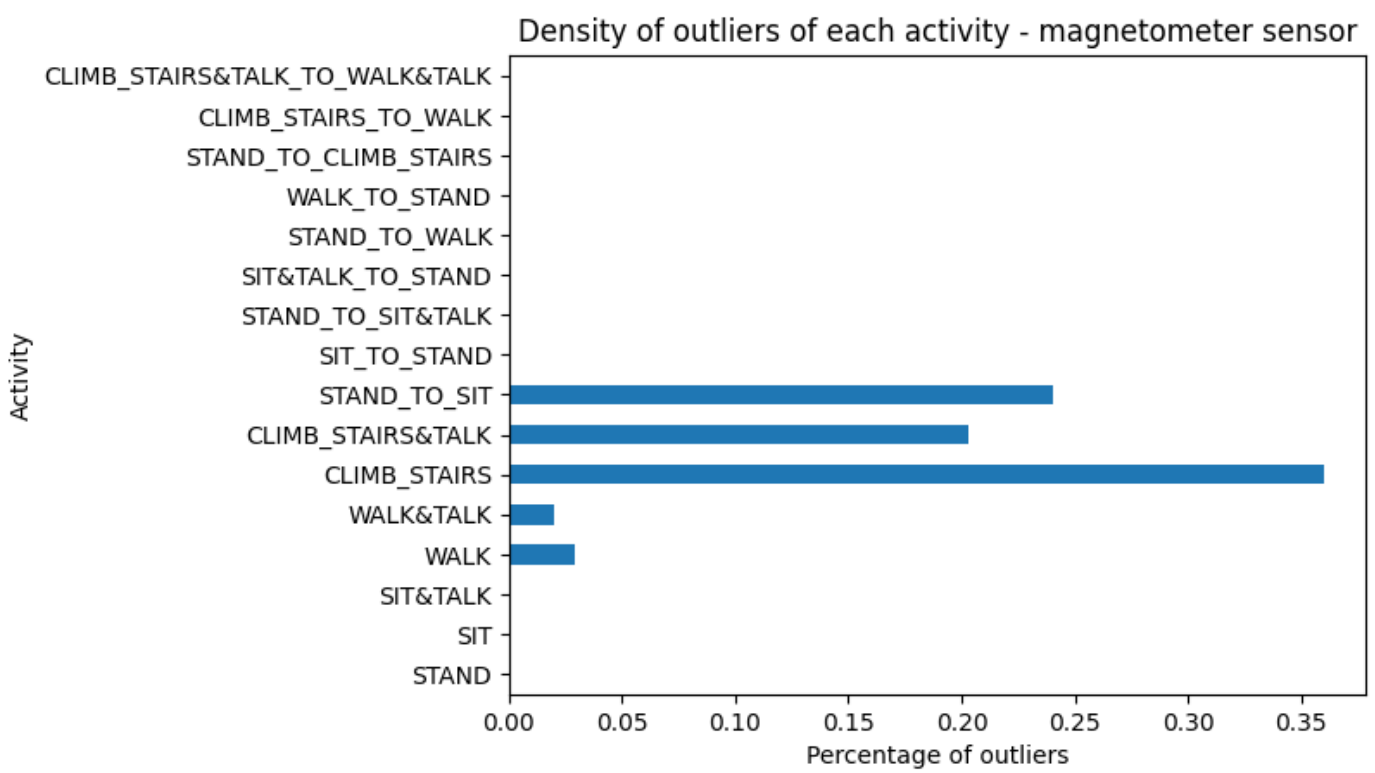




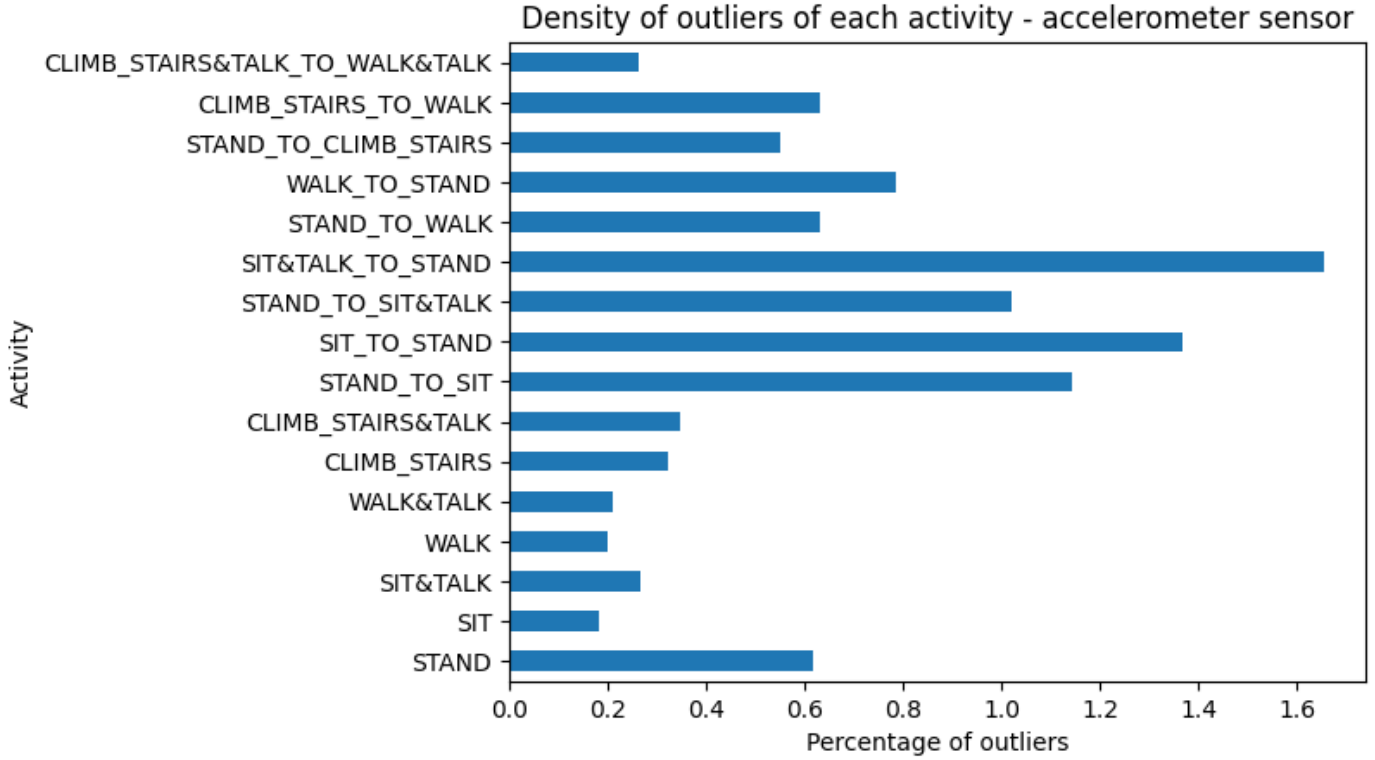
In [24]:
```
densities_zscore_35 = calculate_zscore_densities(data, labels, k=3.5)
plot_densities(densities_zscore_35)
```

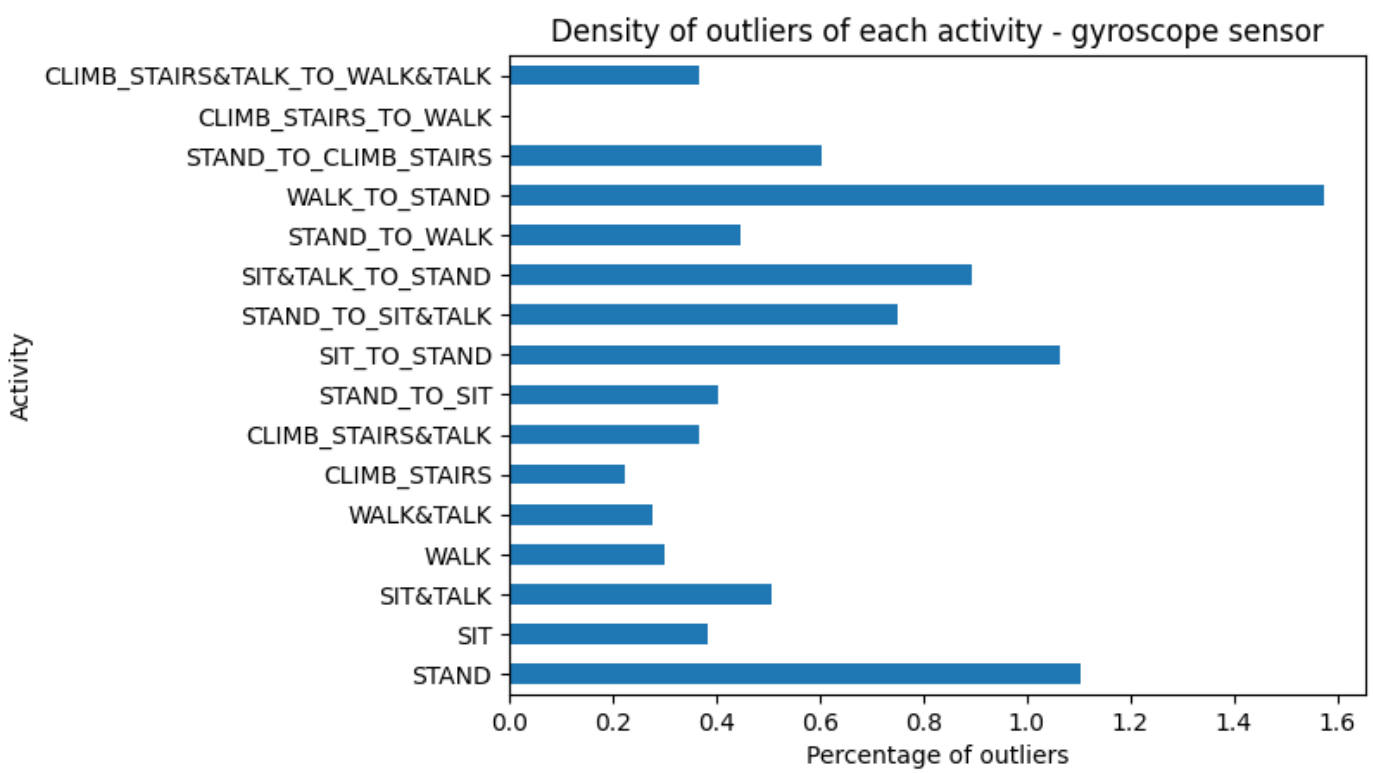

Density of outliers of each activity - accelerometer sensor
CLIMB_STAIRS&TALK_TO_WALK&TALK
CLIMB_STAIRS_TO_WALK
STAND_TO_CLIMB_STAIRS
WALK_TO_STAND
STAND_TO_WALK
SIT&TALK_TO_STAND
STAND_TO_SIT&TALK
SIT_TO_STAND
STAND_TO_SIT
CLIMB_STAIRS&TALK
CLIMB_STAIRS
WALK&TALK
WALK
SIT&TALK
SIT
STAND
Activity
0.0
0.5
1.0
1.5
2.0
Percentage of outliers

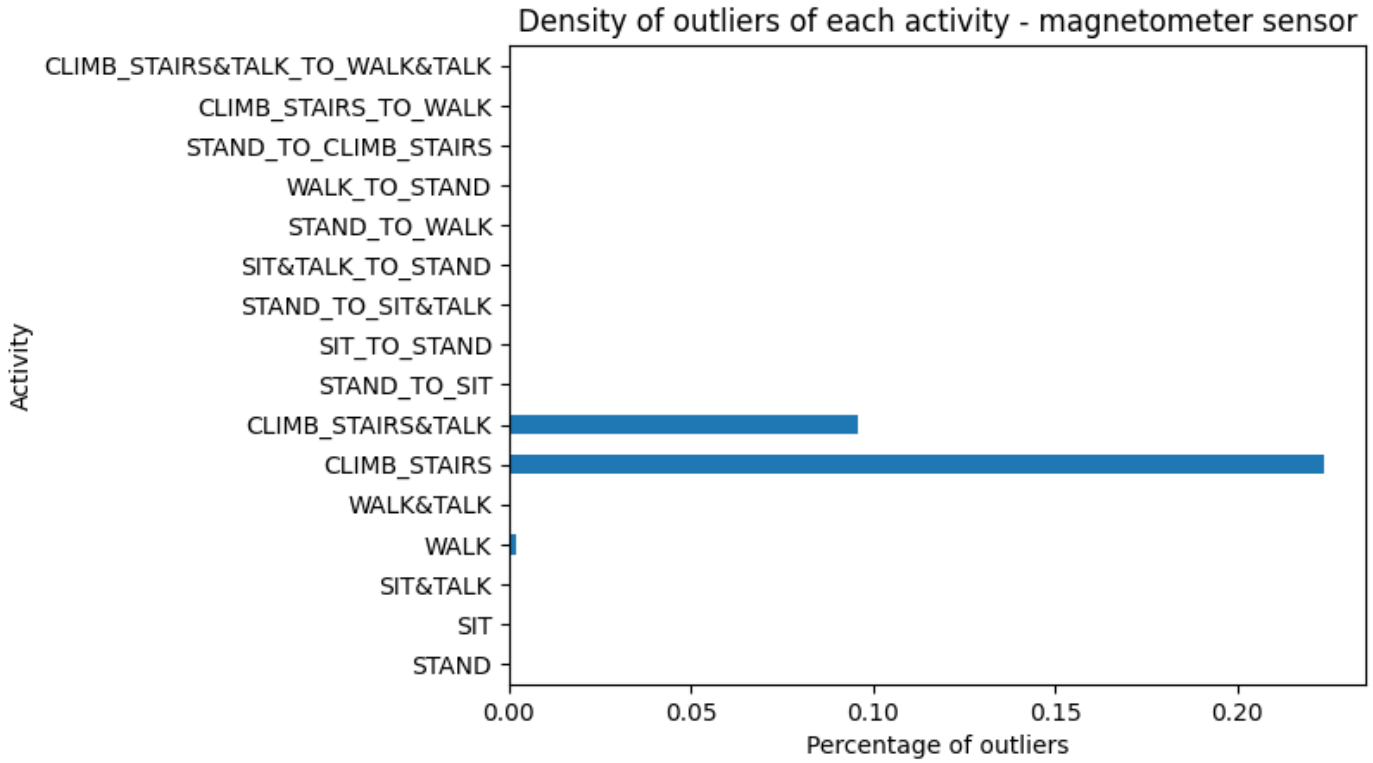

Density of outliers of each activity - gyroscope sensor
CLIMB_STAIRS&TALK_TO_WALK&TALK
CLIMB_STAIRS_TO_WALK
STAND_TO_CLIMB_STAIRS
WALK_TO_STAND
STAND_TO_WALK
SIT&TALK_TO_STAND
STAND_TO_SIT&TALK
SIT_TO_STAND
STAND_TO_SIT
CLIMB_STAIRS&TALK
CLIMB_STAIRS
WALK&TALK
WALK
SIT&TALK
SIT
STAND
Activity
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Percentage of outliers



In [25]:
```
densities_zscore_4 = calculate_zscore_densities(data, labels, k=4)
plot_densities(densities_zscore_4)
```







Pelos gráficos obtidos, podemos observar que à medida que o k aumenta, o número de pontos considerados outliers diminui. Mostra-se assim, que ao aumentar k, aumentamos a gama de valor válidos, sendo menos sensível a outlier.

Comparando os resultados com a alínea anterior, repara-se a densidade de outliers é muito menor. Isto poderá dizer que o Z-Score com os k que estamos a utilizar são menos sensíveis a outliers.

Exercise 3.6 - K-means

Try to divide by activities

Foi utilizado o KMeans com k igual ao número de atividades para tentar perceber se as atividades teriam valores distintos dos módulos dos vetores para as diferentes atividades

In [17]:
```
kmeans = KMeans(len(data.activity.unique()))
labels = kmeans.predict(right_features)
plot_3d(kmeans.data, labels, 'KMeans clusters for all activities in the rights_wrist wit
plot_3d(right_wrist_data, data.activity, 'Real activities cluster in dataset of the righ
```

```
Iteration: 0
Iteration: 10
Iteration: 20
Iteration: 30
Iteration: 40
Iteration: 50
Iteration: 60
Iteration: 70
Iteration: 80
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In [17], line 2
      1 kmeans = KMeans(len(data.activity.unique()))
----> 2 labels = kmeans.predict(right_features)
      3 plot_3d(kmeans.data, labels, 'KMeans clusters for all activities in the rights_w
rist with k=16')
      4 plot_3d(right_wrist_data, data.activity, 'Real activities cluster in dataset of
 the right wrist')

File c:\Users\joana\Desktop\mecd\TCD\clf-human-activities\src\kmeans.py:20, in KMeans.pr
edict(self, dataset)
     18 if i % 10 == 0:
     19     print(f'Iteration: {i}')
---> 20 new_centroids = self.update_clusters()
     21 if self.centroids.equals(new_centroids):
     22     break

File c:\Users\joana\Desktop\mecd\TCD\clf-human-activities\src\kmeans.py:31, in KMeans.up
date_clusters(self)
     30 def update_clusters(self):
---> 31     self.distances = self.centroids.apply(lambda x : np.sqrt(((self.data - x) **
2).sum(axis=1)), axis=1).T
     32     self.labels = self.distances.idxmin(axis=1)
     33     new_centroids = self.data.groupby(self.labels).mean()

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\fra
me.py:9558, in DataFrame.apply(self, func, axis, raw, result_type, args, **kwargs)
   9547 from pandas.core.apply import frame_apply
   9549 op = frame_apply(
   9550     self,
   9551     func=func,
   (...)
   9556     kwargs=kwargs,
   9557 )
-> 9558 return op.apply().__finalize__(self, method="apply")

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\app
ly.py:741, in FrameApply.apply(self)
    738 elif self.raw:
    739     return self.apply_raw()
--> 741 return self.apply_standard()

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\app
ly.py:868, in FrameApply.apply_standard(self)
    867 def apply_standard(self):
--> 868     results, res_index = self.apply_series_generator()
```

```
    870     # wrap results
    871     return self.wrap_results(results, res_index)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\app
ly.py:884, in FrameApply.apply_series_generator(self)
    881 with option_context("mode.chained_assignment", None):
    882     for i, v in enumerate(series_gen):
    883         # ignore SettingWithCopy here in case the user mutates
--> 884         results[i] = self.f(v)
    885         if isinstance(results[i], ABCSeries):
    886             # If we have a view on v, we need to make a copy because
    887             #  series_generator will swap out the underlying data
    888             results[i] = results[i].copy(deep=False)

File c:\Users\joana\Desktop\mecd\TCD\clf-human-activities\src\kmeans.py:31, in KMeans.up
date_clusters.<locals>.<lambda>(x)
     30 def update_clusters(self):
---> 31     self.distances = self.centroids.apply(lambda x : np.sqrt(((self.data - x) **
2).sum(axis=1)), axis=1).T
     32     self.labels = self.distances.idxmin(axis=1)
     33     new_centroids = self.data.groupby(self.labels).mean()

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\ops
\common.py:72, in _unpack_zerodim_and_defer.<locals>.new_method(self, other)
     68             return NotImplemented
     70 other = item_from_zerodim(other)
---> 72 return method(self, other)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\arr
aylike.py:111, in OpsMixin.__sub__(self, other)
    109 @unpack_zerodim_and_defer("__sub__")
    110 def __sub__(self, other):
--> 111     return self._arith_method(other, operator.sub)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\fra
me.py:7586, in DataFrame._arith_method(self, other, op)
   7582 other = ops.maybe_prepare_scalar_for_op(other, (self.shape[axis],))
   7584 self, other = ops.align_method_FRAME(self, other, axis, flex=True, level=None)
-> 7586 new_data = self._dispatch_frame_op(other, op, axis=axis)
   7587 return self._construct_result(new_data)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\fra
me.py:7625, in DataFrame._dispatch_frame_op(self, right, func, axis)
   7619     # TODO: The previous assertion `assert right._indexed_same(self)`
   7620     #  fails in cases with empty columns reached via
   7621     #  _frame_arith_method_with_reindex
   7622
   7623     # TODO operate_blockwise expects a manager of the same type
   7624     with np.errstate(all="ignore"):
-> 7625         bm = self._mgr.operate_blockwise(
   7626             # error: Argument 1 to "operate_blockwise" of "ArrayManager" has
   7627             # incompatible type "Union[ArrayManager, BlockManager]"; expected
   7628             # "ArrayManager"
   7629             # error: Argument 1 to "operate_blockwise" of "BlockManager" has
   7630             # incompatible type "Union[ArrayManager, BlockManager]"; expected
   7631             # "BlockManager"
   7632             right._mgr,  # type: ignore[arg-type]
   7633             array_op,
   7634         )
   7635     return self._constructor(bm)
   7637 elif isinstance(right, Series) and axis == 1:
   7638     # axis=1 means we want to operate row-by-row

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\int
ernals\managers.py:1566, in BlockManager.operate_blockwise(self, other, array_op)
   1562 def operate_blockwise(self, other: BlockManager, array_op) -> BlockManager:
```

```
   1563     """
   1564     Apply array_op blockwise with another (aligned) BlockManager.
   1565     """
-> 1566     return operate_blockwise(self, other, array_op)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\internals\ops.py:63, in operate_blockwise(left, right, array_op)
     61 res_blks: list[Block] = []
     62 for lvals, rvals, locs, left_ea, right_ea, rblk in _iter_block_pairs(left, right):
---> 63     res_values = array_op(lvals, rvals)
     64     if left_ea and not right_ea and hasattr(res_values, "reshape"):
     65         res_values = res_values.reshape(1, -1)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\ops\array_ops.py:226, in arithmetic_op(left, right, op)
    222     _bool_arith_check(op, left, right)
    224     # error: Argument 1 to "_na_arithmetic_op" has incompatible type
    225     # "Union[ExtensionArray, ndarray[Any, Any]]"; expected "ndarray[Any, Any]"
--> 226     res_values = _na_arithmetic_op(left, right, op)  # type: ignore[arg-type]
    228 return res_values

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\ops\array_ops.py:165, in _na_arithmetic_op(left, right, op, is_cmp)
    162     func = partial(expressions.evaluate, op)
    164 try:
--> 165     result = func(left, right)
    166 except TypeError:
    167     if not is_cmp and (is_object_dtype(left.dtype) or is_object_dtype(right)):
    168         # For object dtype, fallback to a masked operation (only operating
    169         #  on the non-missing values)
    170         # Don't do this for comparisons, as that will handle complex numbers
    171         #  incorrectly, see GH#32047

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\computation\expressions.py:242, in evaluate(op, a, b, use_numexpr)
    239 if op_str is not None:
    240     if use_numexpr:
    241         # error: "None" not callable
--> 242         return _evaluate(op, op_str, a, b)  # type: ignore[misc]
    243 return _evaluate_standard(op, op_str, a, b)

File c:\Users\joana\anaconda3\envs\cl_human_activities\lib\site-packages\pandas\core\computation\expressions.py:71, in _evaluate_standard(op, op_str, a, b)
     69 if _TEST_MODE:
     70     _store_test_result(False)
---> 71 return op(a, b)

KeyboardInterrupt:
```

Pelos resultados obtidos pelo KMeans ao tentar dividir as atividades em diferentes clusters, conclui-se que o modelo tenta dividir os clusters pelo módulo do Magnetómetro. No entando, ao visualizar os clusters reais, repara-se que os resultados previsto pelo modelo estavam errados.

Try to divide by devices

À semelhança do teste anterior, foi realizado um teste a todo o dataset para tentar perceber se os módulos dos três vetores para os diferentes dispositivos têm valores diferentes.

In [ ]:
```
kmeans_devices = KMeans(len(dataset.device_id.unique()))
labels_devices = kmeans_devices(dataset[['accelerometer_module','gyroscope_module', 'mag
```

```
plot_3d(kmeans_devices.data, labels_devices, 'KMeans cluster for all devices k=5')
plot_3d(kmeans_devices.data, dataset.device_id, 'Real devices used in dataset')
```

Select data

In [19]:
```
accelerometer_data = data[['accelerometer_x', 'accelerometer_y', 'accelerometer_z']]
gyroscope_data = data[['gyroscope_x', 'gyroscope_y', 'gyroscope_z']]
magnetometer_data = data[['magnetometer_x', 'magnetometer_y', 'magnetometer_z']]
```

Accelerometer

Como o cálculo do melhor k a utilizar é um processo muito demorado, as linhas de código que procedem o k-means para os diferentes k estão comentadas, estão apena apresentado o gráfico resultante do processo. Para executar, basta descomentar as linhas.

In [31]:
```
# optimal_k_acc = best_number_clusters(normalize_data(accelerometer_data), threshold=0.9
# print('optimal k:', optimal_k_acc)
```

In [32]:
```
# print('optimal k:', optimal_k_acc)
```

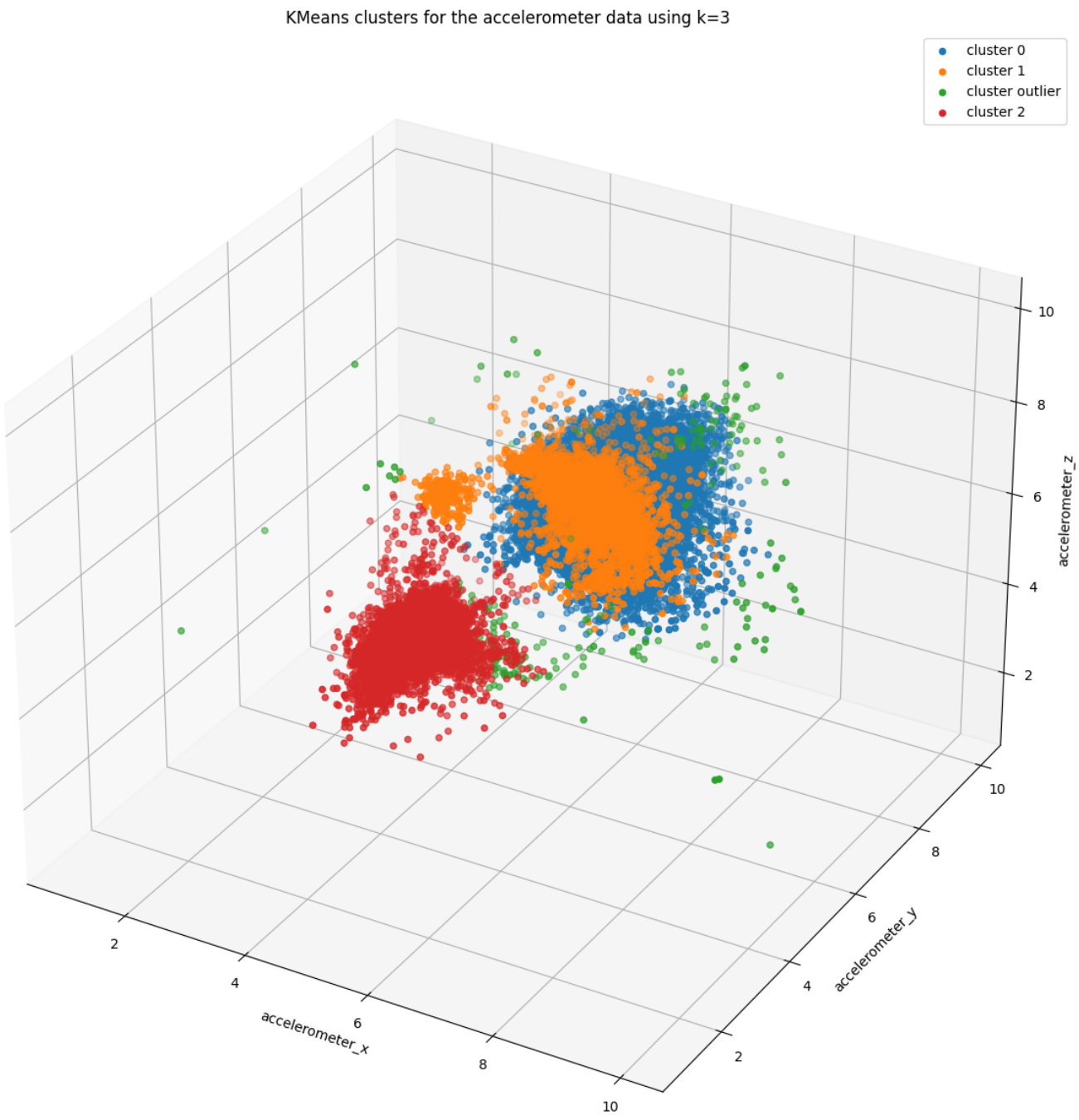
 Best number clusters - KMeans - accelerometer data

Como se pode observar, segundo o método Elbow, k=3 parece ser o número de clusters ideal.

In [20]:
```
%%time
kmeans_acc = KMeans(3, max_iterations=150)
labels_acc = kmeans_acc.predict(accelerometer_data)
```

```
Iteration: 0
Iteration: 10
Iteration: 20
CPU times: total: 1min 13s
Wall time: 1min 34s
```

In [21]:
```
labels_acc = kmeans_acc.get_labels_with_outliers(2.5)
plot_3d(kmeans_acc.data, labels_acc, 'KMeans clusters for the accelerometer data using k
```



Comparison with Z-score outliers

In [27]:
```
outliers_accelerometer_zscore = calculate_outliers_indexes(data['accelerometer_module'],
density_acc_zscore = calculate_density(outliers_accelerometer_zscore)
print(f'Density outliers - accelerometer - Z-Score: {density_acc_zscore*100}%')
```

```
Density outliers - accelerometer - Z-Score: 2.149338340440663%
```

In [28]:
```
outliers_accelerometer_kmeans = kmeans_acc.get_outliers(3)
density_acc_kmeans = calculate_density(outliers_accelerometer_kmeans)
print(f'Density outliers - accelerometer - KMeans: {density_acc_kmeans*100}%')
```

```
Density outliers - accelerometer - KMeans: 0.009209257350138523%
```

In [29]:
```
print(f'Percentage of common outliers: {calculate_common_outliers(outliers_accelerometer
```

```
Percentage of common outliers: 0.005950964056177101%
```

Ao comparar os outliers detetados por um modelo e outro, Z-Score e KMeans, mostra-se que identificam poucos outliers comuns.

Gyroscope

In [ ]:
```
# optimal_k_gyro = best_number_clusters(gyroscope_data, threshold=0.90)
# print('optimal k:', optimal_k_gyro)
```

Tal como para o acelerómetro, o processo de escolha do melhor K foi realizado utilizando o método o Elbow. O resultado final para os dados do giroscópio foi:

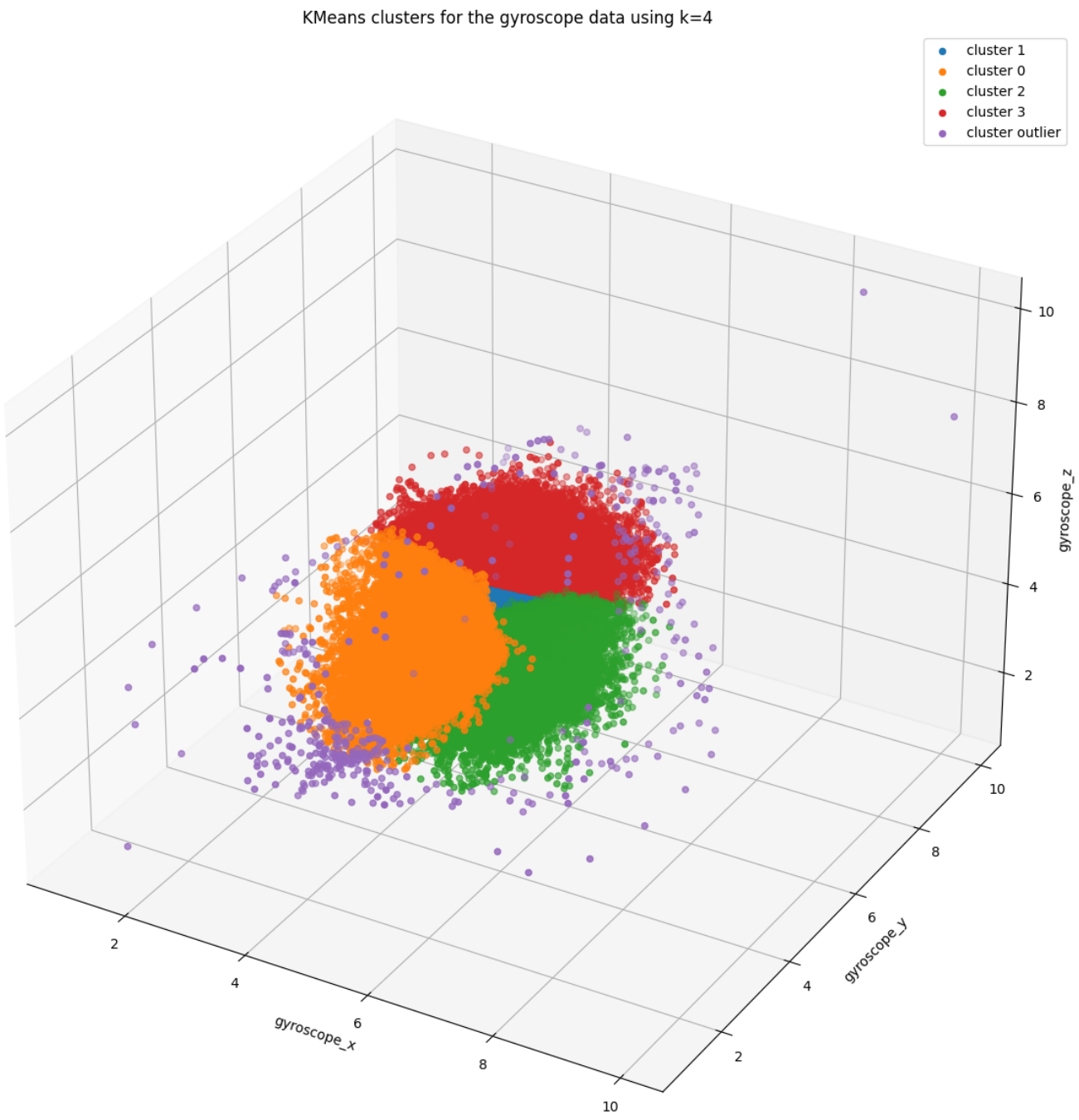
 Best number clusters - KMeans - gyroscope data

Como de pode observar, o número ideal de clusters a utilizar é quatro.

In [30]:
```
%%time
kmeans_gyro = KMeans(4)
labels_gyro = kmeans_gyro.predict(gyroscope_data)
```

```
Iteration: 0
Iteration: 10
Iteration: 20
Iteration: 30
CPU times: total: 1min 42s
Wall time: 2min 26s
```

In [31]:
```
labels_gyro = kmeans_gyro.get_labels_with_outliers(2.5)
plot_3d(kmeans_gyro.data, labels_gyro, 'KMeans clusters for the gyroscope data using k=4
```



Comparison with Z-score outliers

In [33]:
```
outliers_gyroscope_zscore = calculate_outliers_indexes(data['gyroscope_module'], 3)
density_gyro_zscore = calculate_density(outliers_gyroscope_zscore)
print(f'Density outliers - gyroscope - Z-Score: {density_gyro_zscore*100}%')
```

```
Density outliers - gyroscope - Z-Score: 0.9694022424541648%
```

In [34]:
```
outliers_gyroscope_kmeans = kmeans_gyro.get_outliers(3)
density_gyro_kmeans = calculate_density(outliers_gyroscope_kmeans)
print(f'Density outliers - gyroscope - KMeans: {density_gyro_kmeans*100}%')
```

```
Density outliers - gyroscope - KMeans: 0.018290608348191788%
```

In [35]:
```
print(f'Percentage of common outliers: {calculate_common_outliers(outliers_gyroscope_zsc
```

```
Percentage of common outliers: 0.0%
```

Tal como para o acelerómetro, a percentagem de outliers comuns, ou seja, identificados por ambos os modelos, é baixa

Magnetometer

In [ ]:
```
# optimal_k_mag = best_number_clusters(normalize_data(magnetometer_data), threshold=0.85
# print('optimal k:', optimal_k_mag)
```

À semelhança dos anteriores o processo de escolha do melhor K foi realizado utilizando o método o Elbow.
O resultado final para os dados do magnetómetro foi:

 Best number clusters - KMeans - magnetometer data

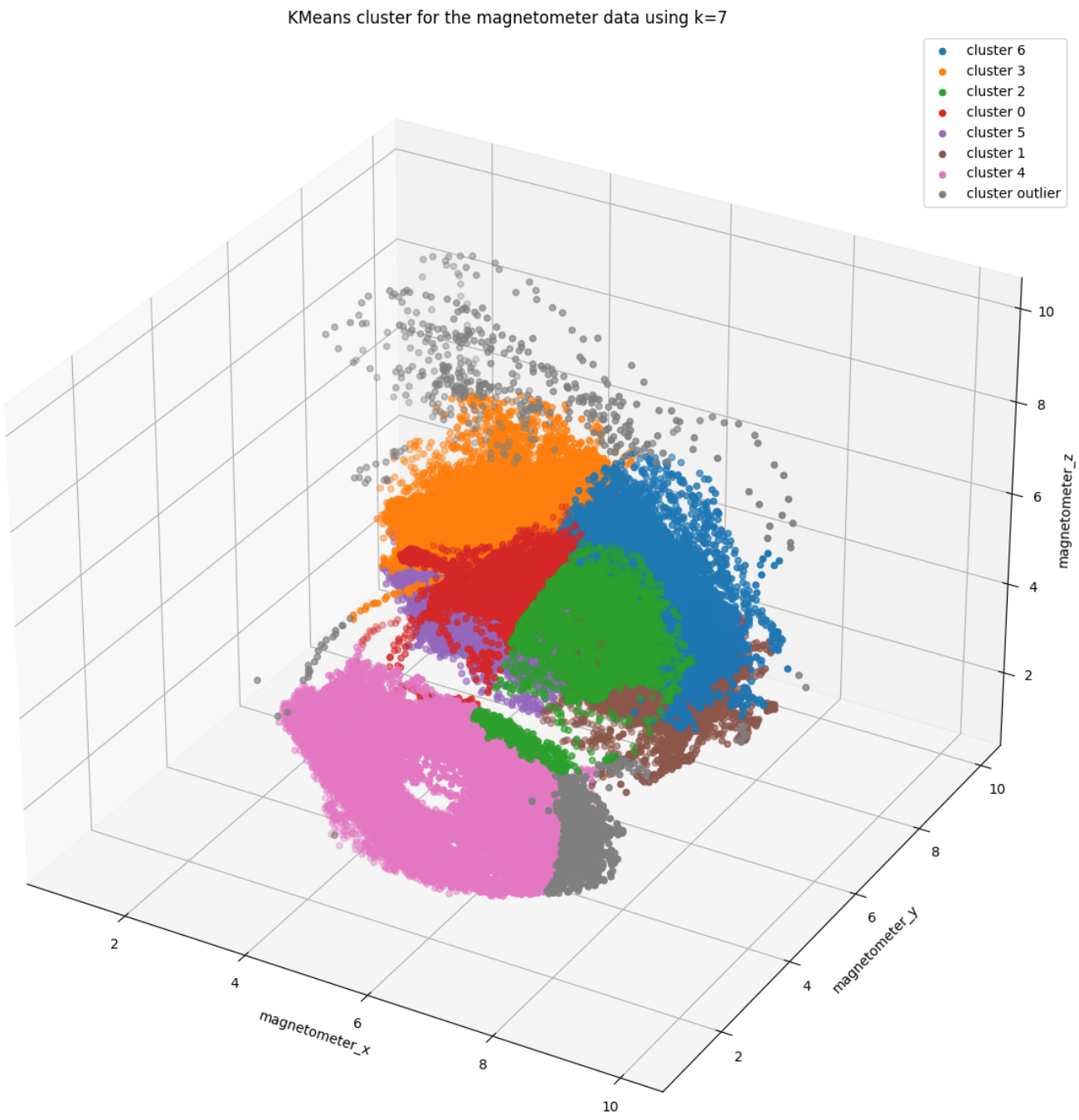
Como de pode observar, o número ideal de clusters a utilizar é sete.

In [36]:
```
%%time
kmeans_mag = KMeans(7)
labels_mag = kmeans_mag.predict(magnetometer_data)
```

```
Iteration: 0
Iteration: 10
Iteration: 20
Iteration: 30
Iteration: 40
Iteration: 50
Iteration: 60
CPU times: total: 3min 27s
Wall time: 4min 51s
```

In [37]:
```
labels_mag = kmeans_mag.get_labels_with_outliers(2.5)
```

In [38]:
```
plot_3d(kmeans_mag.data, labels_mag, 'KMeans cluster for the magnetometer data using k=7
```



Comparison with Z-score outliers

In [40]:
```
outliers_magnetometer_zscore = calculate_outliers_indexes(data['magnetometer_module'], 3
density_mag_zscore = calculate_density(outliers_magnetometer_zscore)
print(f'Density outliers - magnetometer - Z-Score: {density_mag_zscore*100}%')
```

```
Density outliers - magnetometer - Z-Score: 0.22690586859924639%
```

In [41]:
```
outliers_magnetometer_kmeans = kmeans_mag.get_outliers(3)
density_mag_kmeans = calculate_density(outliers_magnetometer_kmeans)
print(f'Density outliers - magnetometer - KMeans: {density_mag_kmeans*100}%')
```

```
Density outliers - magnetometer - KMeans: 0.03581377858387203%
```

In [43]:
```
print(f'Percentage of common outliers: {calculate_common_outliers(outliers_magnetometer_
```

```
Percentage of common outliers: 0.0%
```

Como se pode observar, a persentagem de outliers comuns entre Z-Score e KMeans é nula para o magnetómetro.

Z-Score vs KMeans

Para comparar os resultados entre o KMeans e o Z-Score, foram analisados os outliers que cada um deles detetou. Pelos resultados obtidos para os três vetores, reparamos que a percentagem de outliers comuns entre os dois é baixa. Isto deve-se a uma série de fatores, tais como:

- Os algoritmos são bastante diferentes, um escala os dados para ter uma média de zero e desvio padrão de 1 (Z-Score) e o outro é um método não supervisionado para calcular clusters.
- Os número de clusters dado ao KMeans pode não coicidir com os valores do Z-Score
- Como os dados não colocados numa escala de 1-10 para calcular os clusters, poderá estar a influenciar os valores que serám outliers ou não, ou seja, o threshold utilizado no KMeans não corresponde ao threshold utilizado no Z-Score.

Exercise 3.8 - Add outliers

In [61]:
```
k = 3
percentage = 0.1
```

In [114…
```
series_with_outliers, density = add_outliers(percentage, k, data['accelerometer_module'])
```

```
0.02149338340440663
Quantity: 61378
Count 0
inject outliers
0.022067682925269436
Count 1000
inject outliers
0.022618959302756893
Count 2000
inject outliers
0.023220119157557603
Count 3000
inject outliers
0.0237918605513787
Count 4000
inject outliers
0.024421159803638168
Count 5000
inject outliers
0.024954529291833692
Count 6000
inject outliers
0.02550964285988371
Count 7000
inject outliers
0.026005919505974504
Count 8000
inject outliers
0.026558474946982817
Count 9000
inject outliers
0.02715068135713756
Count 10000
inject outliers
0.027676376464207966
Count 11000
inject outliers
0.028214862206486897
```

Count 12000
inject outliers
0.028753347948765832
Count 13000
inject outliers
0.02926881054766942
Count 14000
inject outliers
0.029818807861636026
Count 15000
inject outliers
0.030327875142935348
Count 16000
inject outliers
0.030803686772692504
Count 17000
inject outliers
0.03127949840244966
Count 18000
inject outliers
0.03178089130262387
Count 19000
inject outliers
0.03230147015561087
Count 20000
inject outliers
0.032770886467763764
Count 21000
inject outliers
0.0332339074623124
Count 22000
inject outliers
0.03369948658390273
Count 23000
inject outliers
0.034244367643785925
Count 24000
inject outliers
0.03474064428987672
Count 25000
inject outliers
0.03521389779259217
Count 26000
inject outliers
0.03566412815193228
Count 27000
inject outliers
0.036116916638314095
Count 28000
inject outliers
0.03649935663104901
Count 29000
inject outliers
0.03680889000309533
Count 30000
inject outliers
0.0371683068524549
Count 31000
inject outliers
0.03745097989056333
Count 32000
inject outliers
0.03771446697585896
Count 33000
inject outliers
0.0380099306491759

```
Count 34000
inject outliers
0.038281092115596646
Count 35000
inject outliers
0.038603416122851494
Count 36000
inject outliers
0.03889504260560588
Count 37000
inject outliers
0.03909201838781717
Count 38000
inject outliers
0.03935806360015451
Count 39000
inject outliers
0.03957038814461604
Count 40000
inject outliers
0.03971620138599323
Count 41000
inject outliers
0.039867130881453834
Count 42000
inject outliers
0.03995538626439266
Count 43000
inject outliers
0.04010887388689497
Count 44000
inject outliers
0.040249570874188755
Count 45000
inject outliers
0.04036212846402378
Count 46000
inject outliers
0.04048108137146307
Count 47000
inject outliers
0.040490034816109036
Count 48000
inject outliers
0.04045550010104602
Count 49000
inject outliers
0.040452941974004314
Count 50000
inject outliers
0.040493872006671594
Count 51000
inject outliers
0.04041712819542044
Count 52000
inject outliers
0.04033526813008587
Count 53000
inject outliers
0.040270035890522395
Count 54000
inject outliers
0.04008585074351963
Count 55000
inject outliers
0.03993620031157987
```

```
Count 56000
inject outliers
0.03983387522991167
Count 57000
inject outliers
0.03961643443136673
Count 58000
inject outliers
0.039341435774383426
Count 59000
inject outliers
0.03909841370542144
Count 60000
inject outliers
0.038782485015770855
Count 61000
inject outliers
0.03850237010470414
Count 62000
inject outliers
0.03824399927349192
```

In [115…
```
outliers_data = data.copy()
outliers_data['accelerometer_data'] = series_with_outliers
```

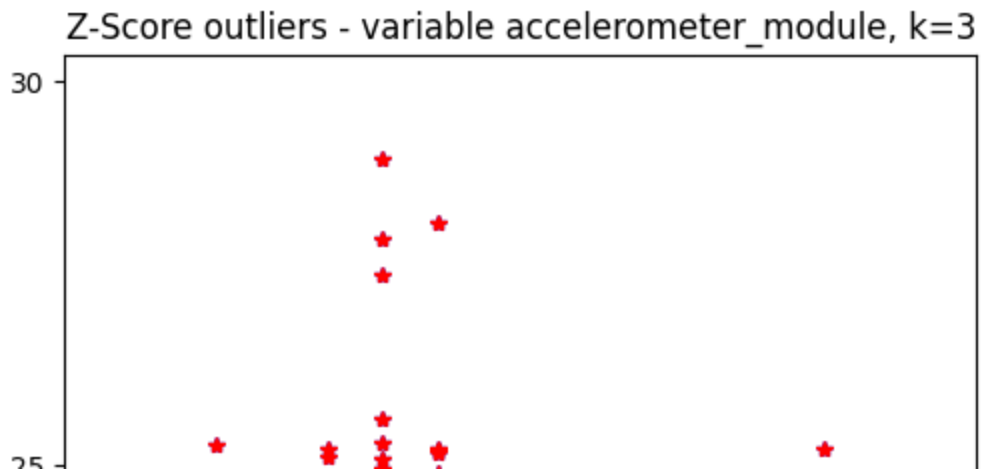
In [116…
```
outliers_data.to_csv(os.path.join(RESULTS_PATH, 'data_with_outliers.csv'), index=False)
```

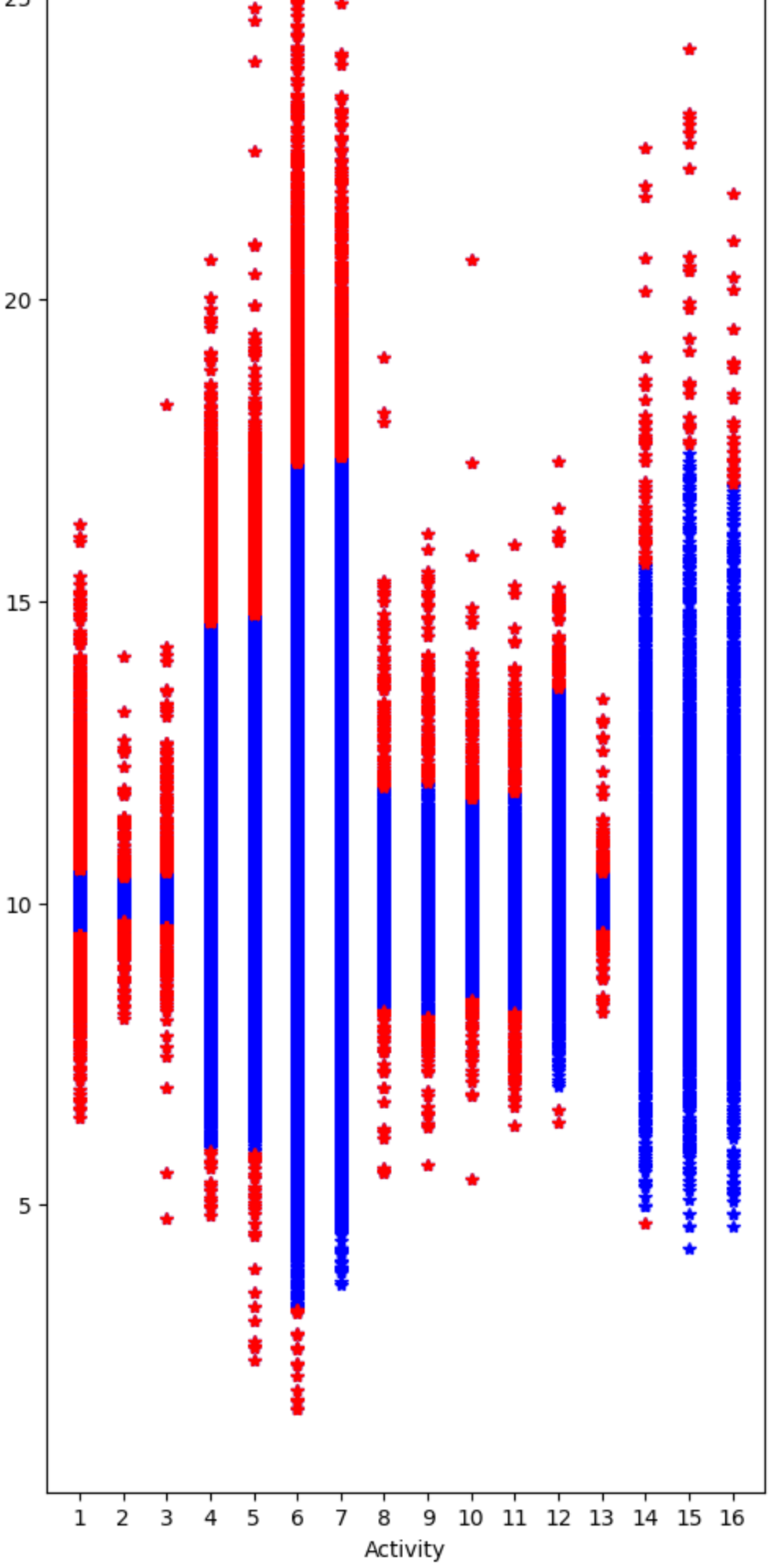
In [117…
```
data['accelerometer_module'].equals(series_with_outliers)
```

Out[117]:
```
False
```

In [118…
```
plot_zscore_outliers(outliers_data, 'accelerometer_module', k_value=k )
plot_zscore_outliers(data, 'accelerometer_module', k_value=k)
```

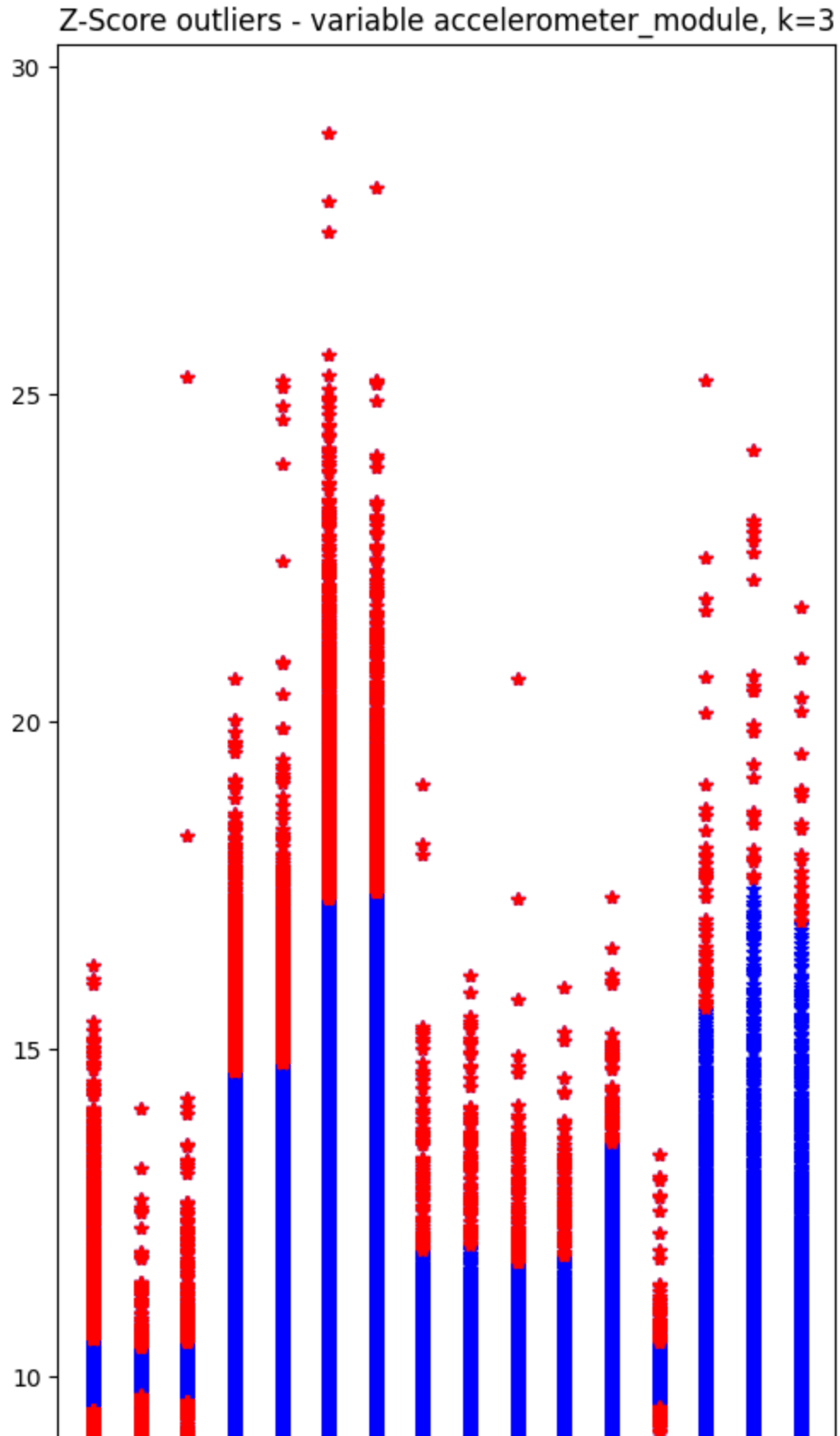
```
Density:  0.8411345181844668
Density:  1.8329326923076923
Density:  0.23699504573170732
Density:  2.5390625
Density:  2.2235576923076925
Density:  0.3635153785488959
Density:  2.994791666666667
Density:  1.837270341207349
Density:  1.069790043877828
Density:  1.2335958005249343
Density:  1.0690754815054155
Density:  1.3123359580052494
Density:  1.3014917355834332
Density:  1.2598425196850394
Density:  1.3587266789979675
Density:  1.2598425196850394
```

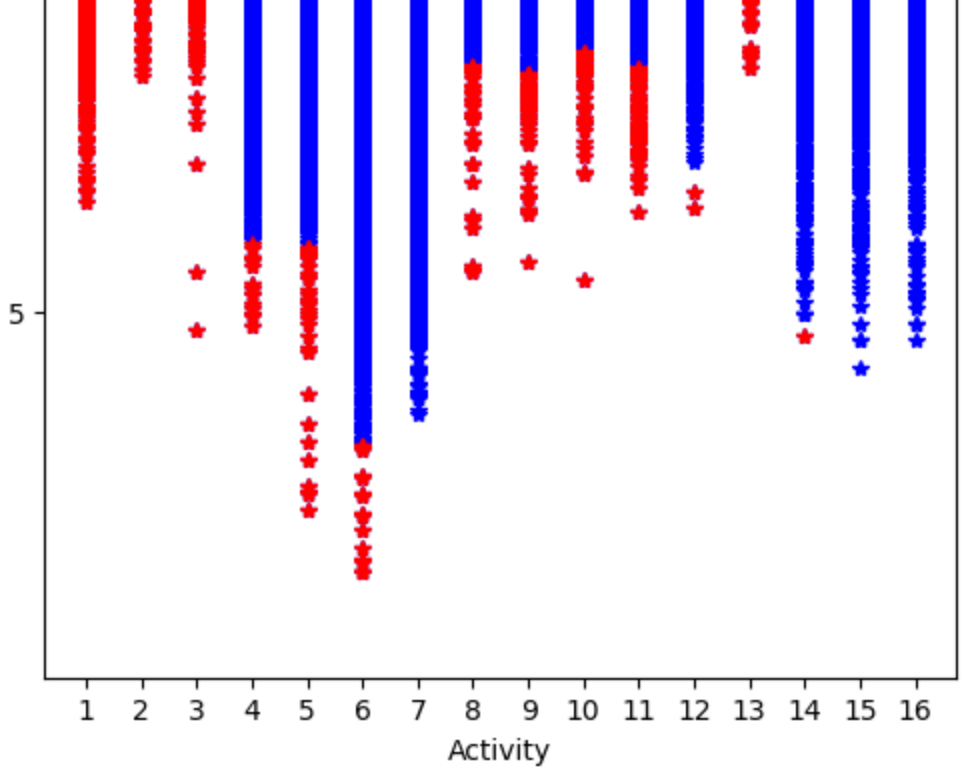





```
Density:  0.8411345181844668
Density:  1.8329326923076923
Density:  0.23699504573170732
```

```
Density:  2.5390625
Density:  2.2235576923076925
Density:  0.3635153785488959
Density:  2.994791666666667
Density:  1.837270341207349
Density:  1.069790043877828
Density:  1.2335958005249343
Density:  1.0690754815054155
Density:  1.3123359580052494
Density:  1.3014917355834332
Density:  1.2598425196850394
Density:  1.3587266789979675
Density:  1.2598425196850394
```





A densidade final não chega aos 10% dado como input à função. A razão deve-se a que, apesar de estarem a ser adicionada a qauntidade de outliers necessários, eles estão a alterar a média e desvio padrão de tal forma que deixam de ser considerados outliers.

Exercise 3.9 - Linear regression

Para calcular o melhor número de valores anteriores (*p*) em que basear as estimativas do modelo de regressão linear, utilizado foram usados os k primeiros valores (não outliers) do dataset, onde k é o índice do primeiro outlier calculado. Estes valores foram usados na técnica do *leave-one-out* para testar o melhor número de p em que basear o modelo. Este teste foi 5 vezes para prever valores diferentes, usando o *RMSE* para calcular a accuracy. O valor de p foi calculado a partir da média dos melhores p obtidos nos testes. Este valor será depois utilizado pelo modelo para estimar todos os outliers

In [62]:
```
k = 3
```

In [63]:
```
outliers_data = pd.read_csv(os.path.join(RESULTS_PATH, 'data_with_outliers.csv'))
regressed_acc = outliers_data['accelerometer_module'].copy()
outliers_indexes = calculate_outliers_indexes(regressed_acc, k)
```

In [64]:
```
density = calculate_density(outliers_indexes)
print('Initial density:', density * 100, '%')
```

```
Initial density: 2.149338340440663 %
```

In [65]:
```
non_outliers_data = outliers_data.loc[outliers_indexes == False]['accelerometer_module']
```

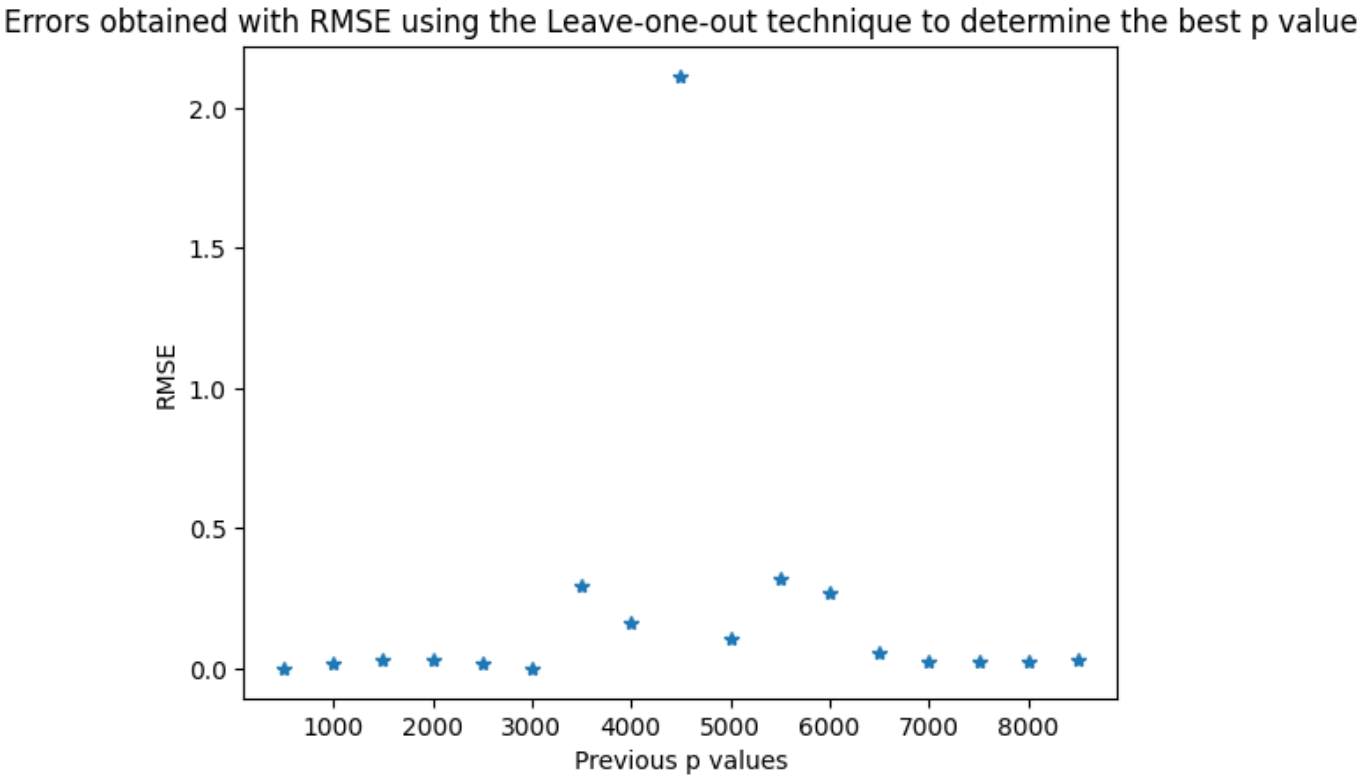
In [66]:
```
indexes_outliers = pd.Series(outliers_data[outliers_indexes].index)
```

Determine best number of previous values to use in the model

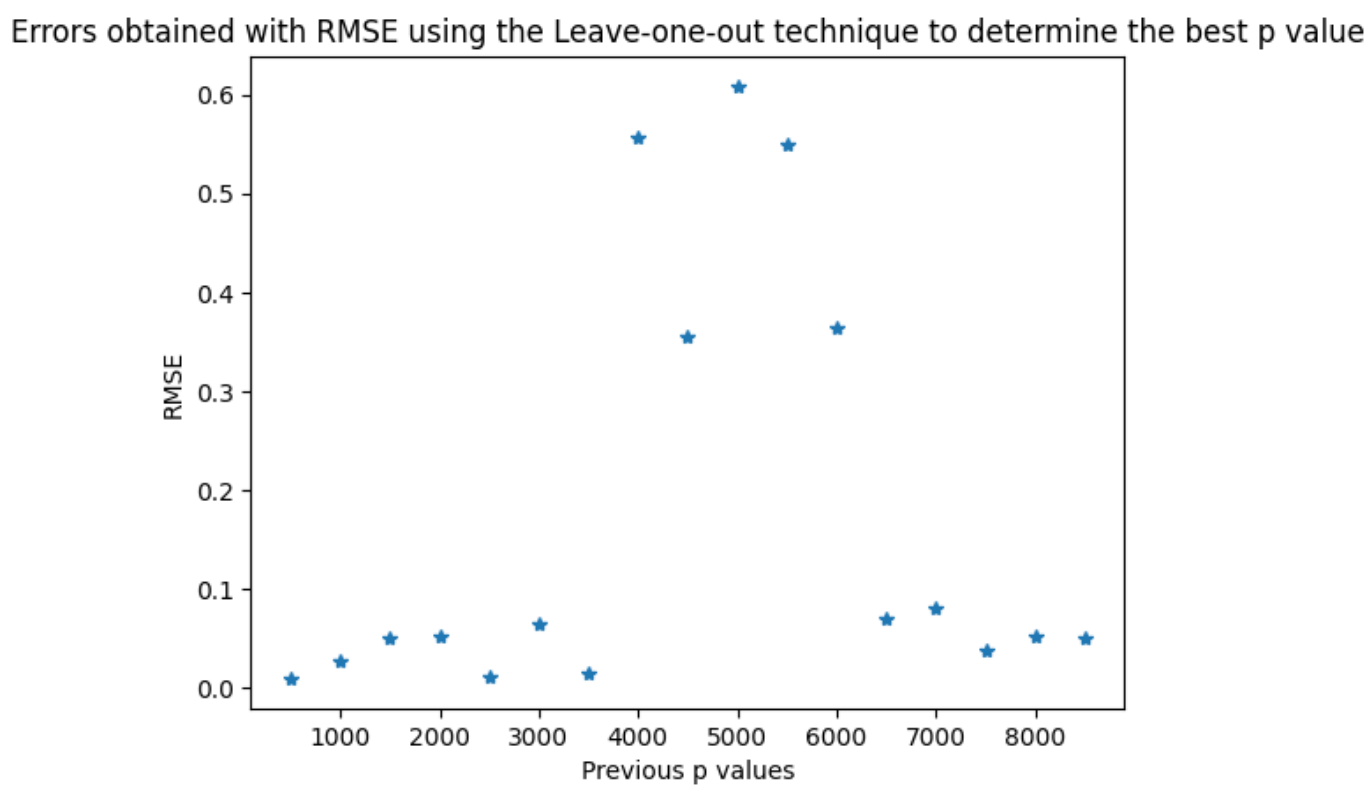
Para determinar o melhor p, foi utilizado como dados de input o maior intervalo sem outliers, para que o cálculo dos pesos não seja influenciado por valores de outliers. A gama de valores possíveis para p é entre

10 e 2000, havendo um step de 50 entre cada p.

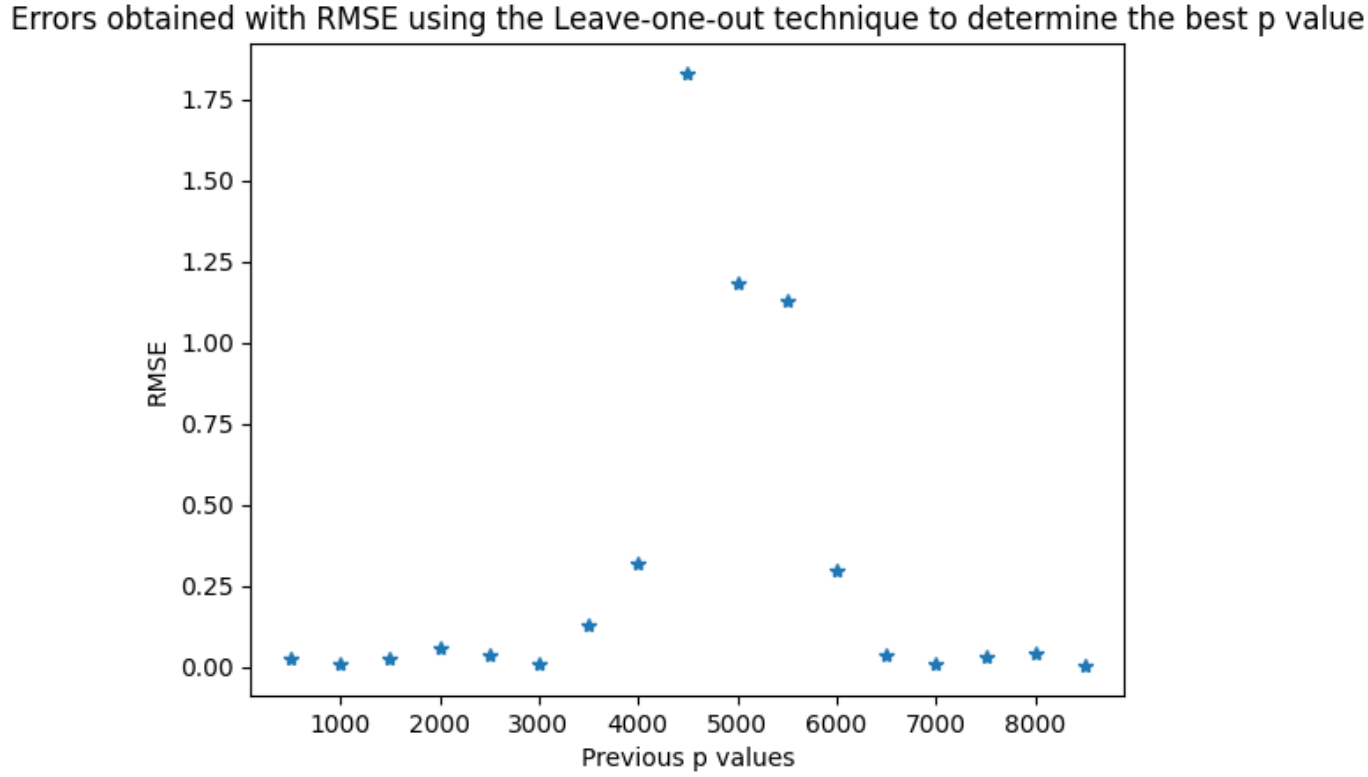
In [67]:
```
%%time
bests = []
errors_values = []
init=500
final = 9000
step= 500
for i in range(5):
    index = np.random.randint(final, indexes_outliers.min()-1)
    best_p_value, rmses = define_best_p_value(outliers_data['accelerometer_module'], ind
    print('Best:', best_p_value)
    bests.append((index, best_p_value))
    errors_values.append(rmses)
```



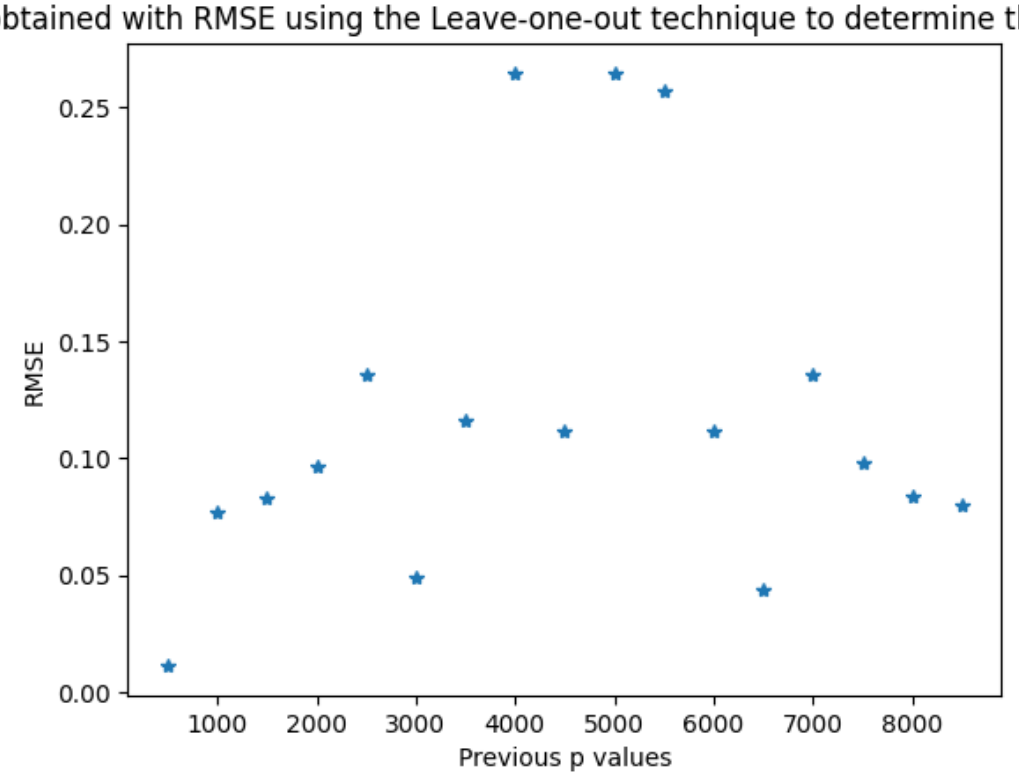

```
Best: 500
```



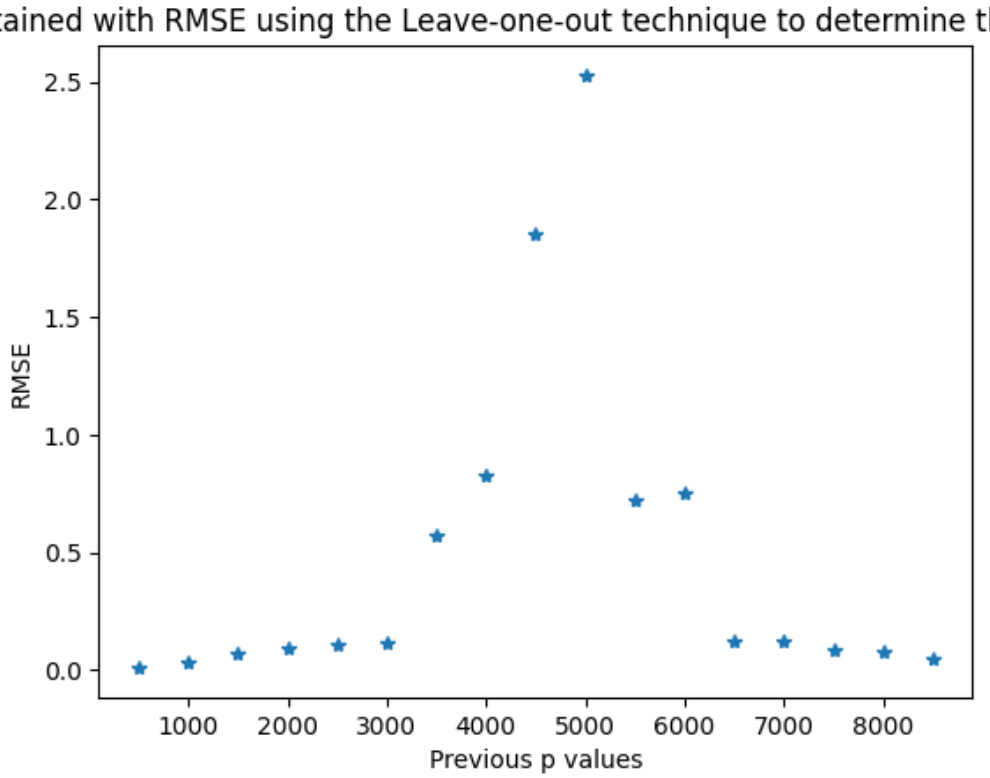

```
Best: 500
```




```
Best: 8500
```




```
Best: 500
```




```
Best: 500
CPU times: total: 44min 32s
Wall time: 38min 38s
```

Pelos resultados observados, ao testar diferentes valores de p nas previsões, observa-se que o p=500 é o que tem um menor RMSE em quase todos os testes. Assim, iremos usar o p=500 nos próximos testes.

In [68]:
```
p_value = 500
p_value_middle = int(p_value/2)
```

Calculate alpha matrices to use

Como o calculo das matrixes são muito demoradas, foi calculada inicialmente a matriz a utilizar como os valores não outliers

In [70]:
```
a_path = os.path.join(RESULTS_PATH, 'matrix_a.csv')
if os.path.exists(a_path):
    a = np.genfromtxt(a_path, delimiter=',')
else:
    a = calculate_alphas(non_outliers_data, p_value)
    a.tofile(a_path, sep=',')
```

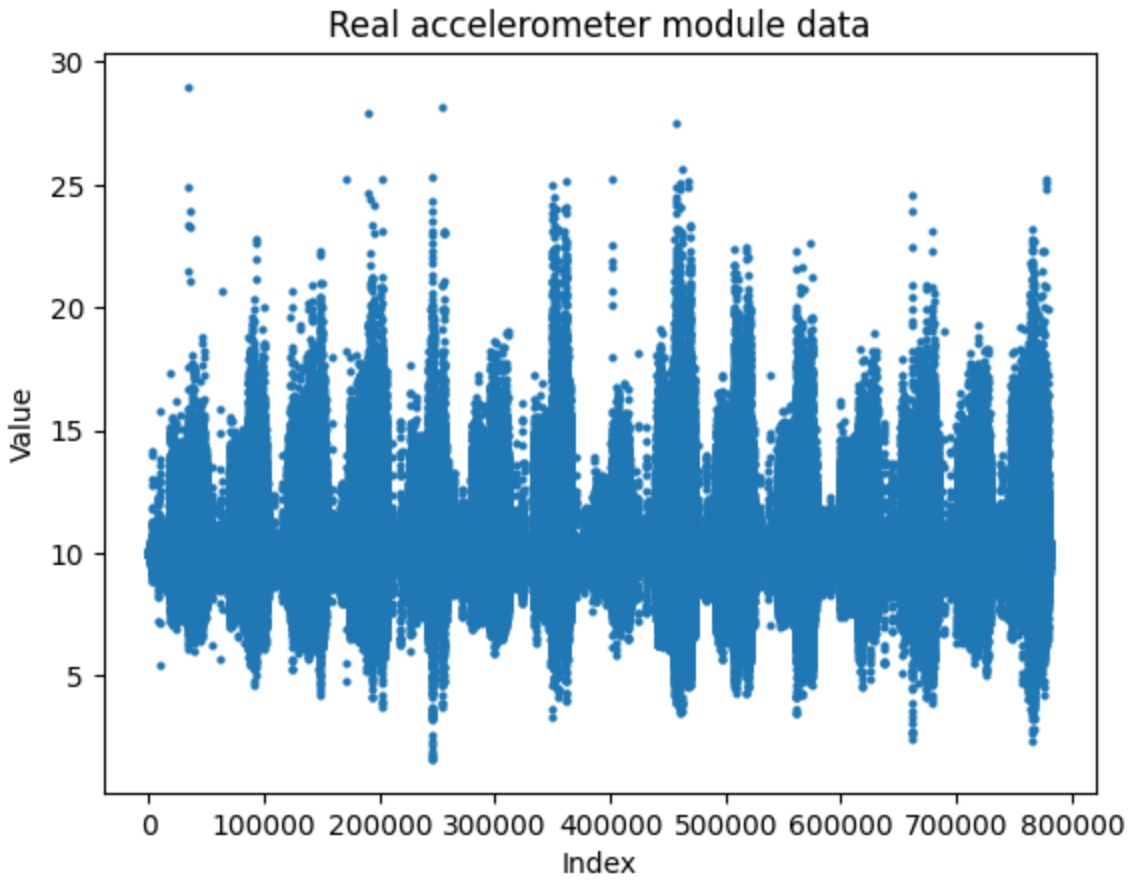
In [71]:
```
a_prev_path = os.path.join(RESULTS_PATH, 'matrix_a_prev.csv')
if os.path.exists(a_prev_path):
    a_prev = np.genfromtxt(a_prev_path, delimiter=',')
else:
    a_prev = calculate_alphas(non_outliers_data, p_value_middle)
    a_prev.tofile(a_prev_path, sep=',')
```

In [72]:
```
a_post_path = os.path.join(RESULTS_PATH, 'matrix_a_post.csv')
if os.path.exists(a_post_path):
    a_post = np.genfromtxt(a_post_path, delimiter=',')
else:
    a_post = calculate_alphas(non_outliers_data[::-1], p_value_middle)
    a_post.tofile(a_post_path, sep=',')
```

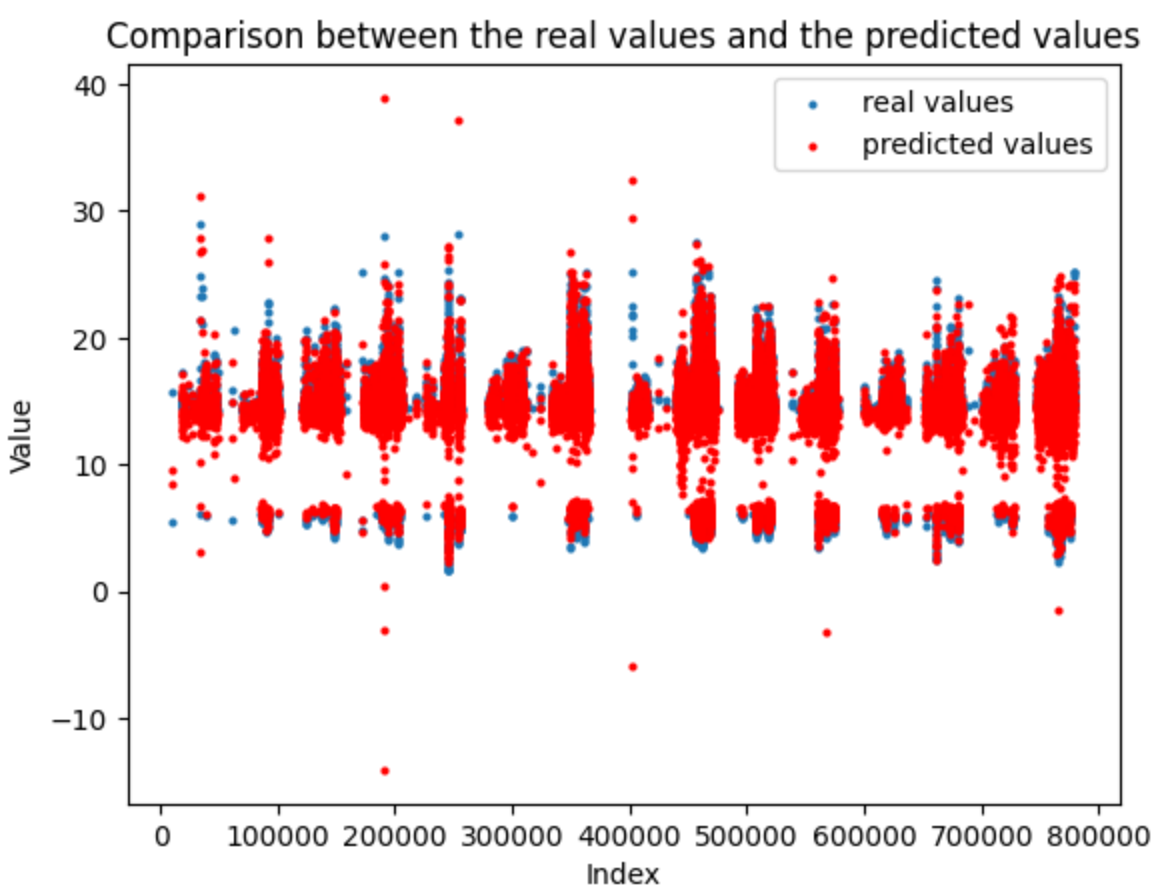
Exercise 3.10 Predict values based on p previous values

In [79]:
```
predicted_values = indexes_outliers.apply(lambda x: predict(regressed_acc[:x], p_value,
```

In [80]:
```
plot_original(outliers_data['accelerometer_module'], 'Real accelerometer module data')
```



In [75]:
```
plot_results(outliers_data['accelerometer_module'],predicted_values, indexes_outliers)
```



In [81]:
```
regressed_acc[indexes_outliers] = predicted_values
outliers_data['accelerometer_data'] = regressed_acc
outliers_data.to_csv(os.path.join(RESULTS_PATH, 'data_linear_regression.csv'), index=Fal
```

In [82]:
```
## calculate final density of outliers
outs = calculate_outliers_indexes(regressed_acc, k)
density = calculate_density(outs)
print('Final density:', density * 100, '%')
```

```
Final density: 1.514539115041531 %
```

Analisando os resultados, obtidos, pode-se constatar que continuam a existir alguns outliers. Isto pode-se dever ao facto de estar a ser utilizado um valor de p adequado em alguns casos. No entanto, pode-se observar que a densidade de outliers diminui, quando comparada com a densidade inicial.

Exercise 3.11 - Predict values based on previous and next values

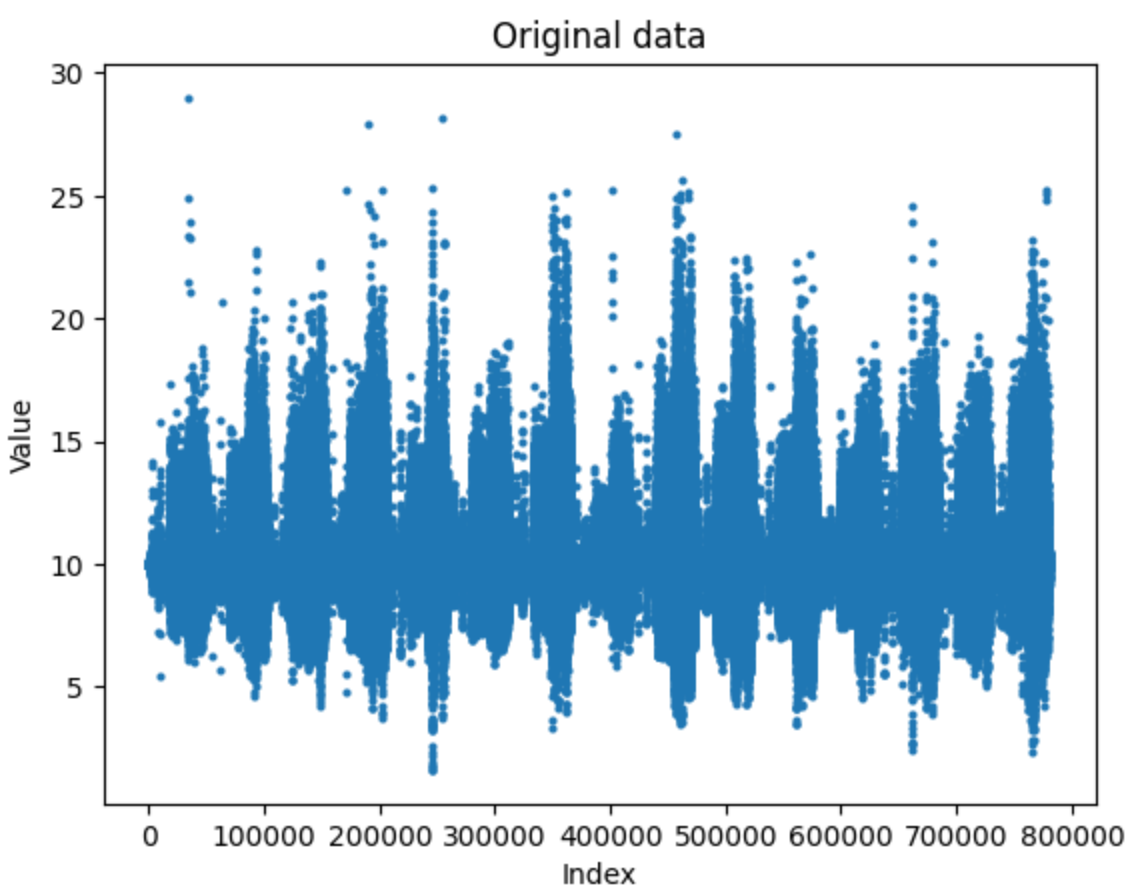
In [83]:
```
outliers_data = pd.read_csv(os.path.join(RESULTS_PATH, 'data_with_outliers.csv'))
regressed_acc = outliers_data['accelerometer_module'].copy()
outliers_indexes = calculate_outliers_indexes(regressed_acc, k)
```

In [84]:
```
## calculate final density of outliers
outs = calculate_outliers_indexes(regressed_acc, k)
density = calculate_density(outs)
print('Initial density:', density * 100, '%')
```

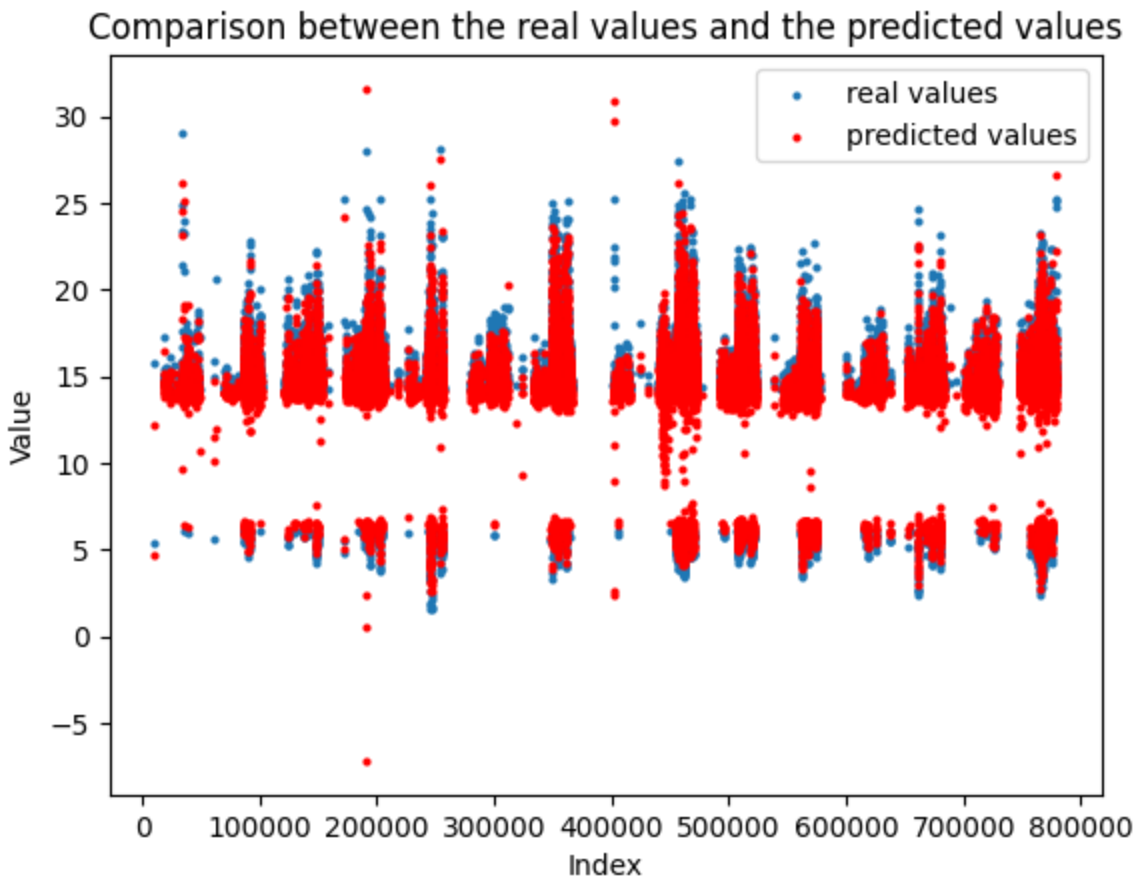
```
Initial density: 2.149338340440663 %
```

In [85]:
```
indexes_outliers = pd.Series(regressed_acc[outliers_indexes].index)
predicted_values = indexes_outliers.apply(lambda x: predict_in_the_middle(regressed_acc,
```

In [86]:
```
plot_original(outliers_data['accelerometer_module'], 'Original data')
```



In [87]:
```
plot_results(regressed_acc, predicted_values, indexes_outliers)
```



In [88]:
```
regressed_acc[indexes_outliers] = predicted_values
regressed_acc.equals(outliers_data['accelerometer_module'])
outliers_data['accelerometer_module']= regressed_acc
outliers_data.to_csv(os.path.join(RESULTS_PATH,'data_regressed_middle.csv'), index=False
```

In [89]:
```
## calculate final density of outliers
outs = calculate_outliers_indexes(regressed_acc, k)
density = calculate_density(outs)
print('Final density:', density * 100, '%')
```

```
Final density: 1.9856182097715336 %
```

Apesar de criar alguns outliers como o de valor negativo perto do índice 200000, podemos observar que a maioria dos valores previstos, têm uma gama menor que os valores anteriores (valores que eram outliers), ou seja, elimina os outliers que existiam. Nos casos em que, ao contrário de eliminar, cria um outlier ainda mais alto, poderá indicar que o valor de p utilizado não está ajustado para alguns casos.

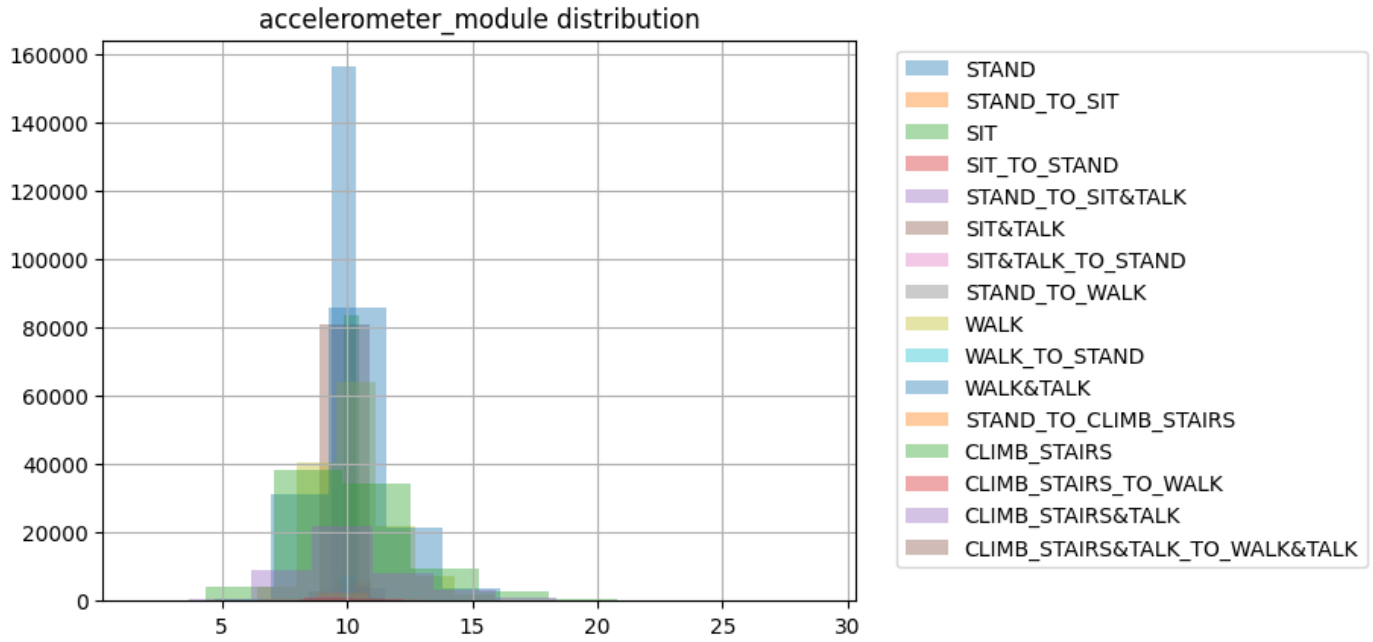
Outro ponto a ter em conta, é que visualmente parece que os valores previstos têm uma amplitude menor, quando comparado com os valores previsto baseados apenas nos anteriores, o que indicaria que existem menos outliers. No entanto, ao analisar a densidade de outliers resultante, constata-se que a percentagem de outliers não diminui tão significamente como na alínea anterior.

Exercise 4.1 - Statistical significance

Nos resultados apresentados a seguir, foi utilizado um *threshold* the 5% no teste de Kolgomorov-Smirov para que o p-value de para verificar se as variáveis seguem uma distribuição normal.

In [9]:
```
ks_test(data[['accelerometer_module', 'activity']], 'accelerometer_module', activities_l
```

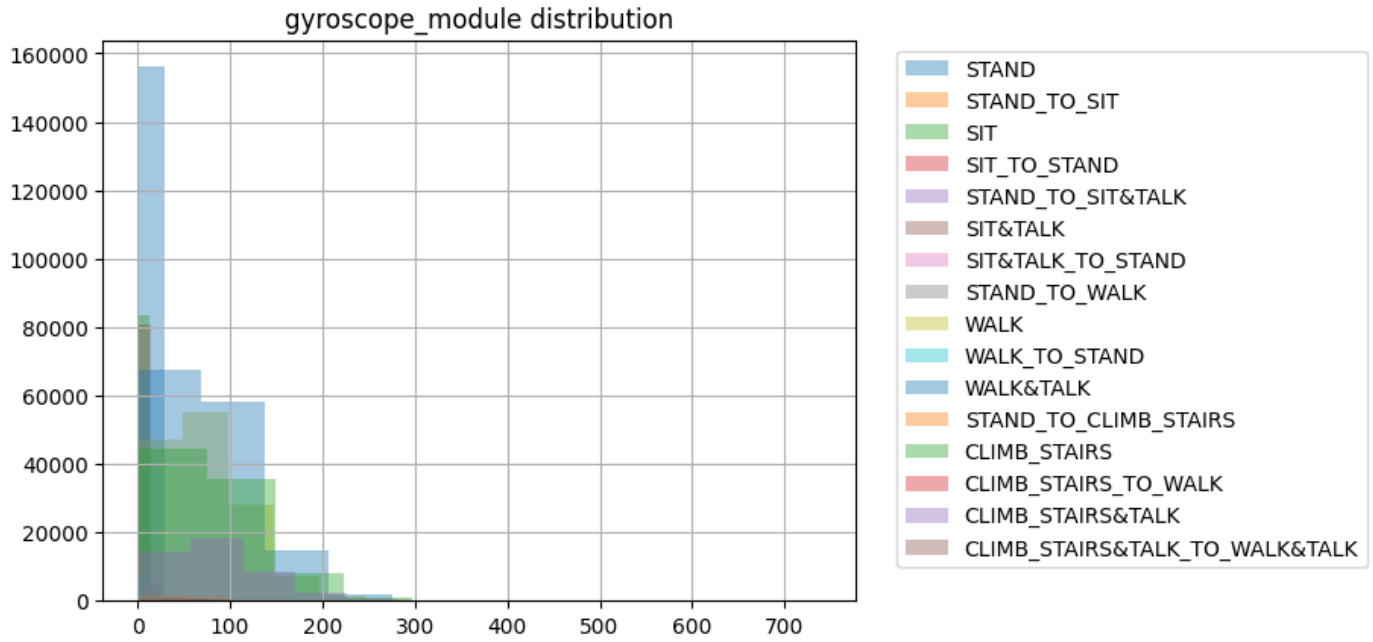
```
Variable accelerometer_module - activitiy STAND: Reject normal distributions | p-value=
0.0
Variable accelerometer_module - activitiy STAND_TO_SIT: Reject normal distributions | p-
value=0.0
Variable accelerometer_module - activitiy SIT: Reject normal distributions | p-value=0.0
Variable accelerometer_module - activitiy SIT_TO_STAND: Reject normal distributions | p-
value=0.0
Variable accelerometer_module - activitiy STAND_TO_SIT&TALK: Reject normal distributions
| p-value=0.0
Variable accelerometer_module - activitiy SIT&TALK: Reject normal distributions | p-valu
e=0.0
Variable accelerometer_module - activitiy SIT&TALK_TO_STAND: Reject normal distributions
| p-value=0.0
Variable accelerometer_module - activitiy STAND_TO_WALK: Reject normal distributions | p
-value=0.0
Variable accelerometer_module - activitiy WALK: Reject normal distributions | p-value=0.
0
Variable accelerometer_module - activitiy WALK_TO_STAND: Reject normal distributions | p
-value=0.0
Variable accelerometer_module - activitiy WALK&TALK: Reject normal distributions | p-val
ue=0.0
Variable accelerometer_module - activitiy STAND_TO_CLIMB_STAIRS: Reject normal distribut
ions | p-value=0.0
Variable accelerometer_module - activitiy CLIMB_STAIRS: Reject normal distributions | p-
value=0.0
Variable accelerometer_module - activitiy CLIMB_STAIRS_TO_WALK: Reject normal distributi
ons | p-value=0.0
Variable accelerometer_module - activitiy CLIMB_STAIRS&TALK: Reject normal distributions
| p-value=0.0
Variable accelerometer_module - activitiy CLIMB_STAIRS&TALK_TO_WALK&TALK: Reject normal
distributions | p-value=0.0
```



Como se pode observar, apesar de histogramas das diferentes actividades para o módulo do vetor de aceleração parecerem seguir uma distribuição normal, todos têm p-value inferior ao mínimo necessário para aceitar a hipótese nula, logo não segue uma distribuição normal.

In [10]:
```
ks_test(data[['gyroscope_module', 'activity']], 'gyroscope_module', activities_labels)
```

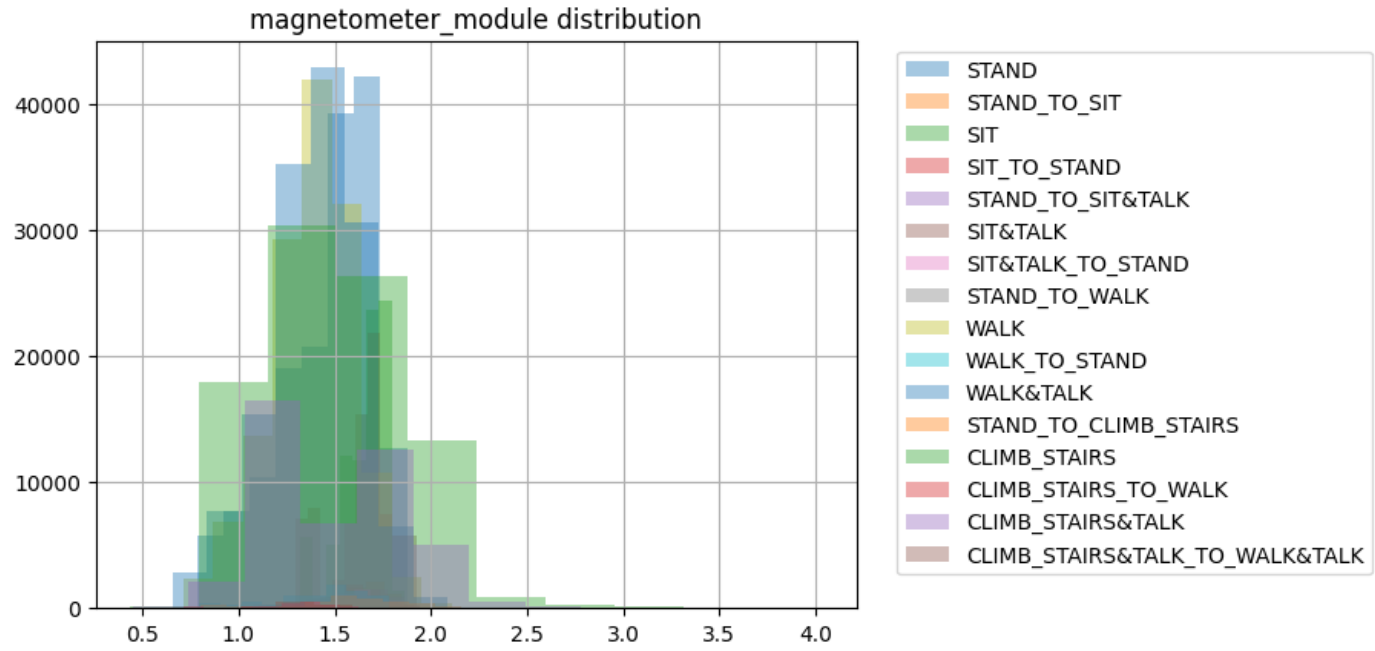
```
Variable gyroscope_module - activitiy STAND: Reject normal distributions | p-value=0.0
Variable gyroscope_module - activitiy STAND_TO_SIT: Reject normal distributions | p-valu
e=0.0
Variable gyroscope_module - activitiy SIT: Reject normal distributions | p-value=0.0
Variable gyroscope_module - activitiy SIT_TO_STAND: Reject normal distributions | p-valu
e=0.0
Variable gyroscope_module - activitiy STAND_TO_SIT&TALK: Reject normal distributions | p
-value=0.0
Variable gyroscope_module - activitiy SIT&TALK: Reject normal distributions | p-value=0.
0
Variable gyroscope_module - activitiy SIT&TALK_TO_STAND: Reject normal distributions | p
-value=0.0
Variable gyroscope_module - activitiy STAND_TO_WALK: Reject normal distributions | p-val
ue=0.0
Variable gyroscope_module - activitiy WALK: Reject normal distributions | p-value=0.0
Variable gyroscope_module - activitiy WALK_TO_STAND: Reject normal distributions | p-val
ue=0.0
Variable gyroscope_module - activitiy WALK&TALK: Reject normal distributions | p-value=
0.0
Variable gyroscope_module - activitiy STAND_TO_CLIMB_STAIRS: Reject normal distributions
| p-value=0.0
Variable gyroscope_module - activitiy CLIMB_STAIRS: Reject normal distributions | p-valu
e=0.0
Variable gyroscope_module - activitiy CLIMB_STAIRS_TO_WALK: Reject normal distributions
| p-value=0.0
Variable gyroscope_module - activitiy CLIMB_STAIRS&TALK: Reject normal distributions | p
-value=0.0
Variable gyroscope_module - activitiy CLIMB_STAIRS&TALK_TO_WALK&TALK: Reject normal dist
ributions | p-value=0.0
```



Como se pode observar pela distribuição e pelo p-value, o módulo do vetor giroscópio não segue uma distribuição gaussiana.

In [11]:
```
ks_test(data[['magnetometer_module', 'activity']], 'magnetometer_module', activities_lab
```

```
Variable magnetometer_module - activitiy STAND: Reject normal distributions | p-value=0.
0
Variable magnetometer_module - activitiy STAND_TO_SIT: Reject normal distributions | p-v
alue=0.0
Variable magnetometer_module - activitiy SIT: Reject normal distributions | p-value=0.0
Variable magnetometer_module - activitiy SIT_TO_STAND: Reject normal distributions | p-v
alue=0.0
Variable magnetometer_module - activitiy STAND_TO_SIT&TALK: Reject normal distributions
| p-value=0.0
Variable magnetometer_module - activitiy SIT&TALK: Reject normal distributions | p-value
=0.0
Variable magnetometer_module - activitiy SIT&TALK_TO_STAND: Reject normal distributions
| p-value=0.0
Variable magnetometer_module - activitiy STAND_TO_WALK: Reject normal distributions | p-
value=0.0
Variable magnetometer_module - activitiy WALK: Reject normal distributions | p-value=0.0
Variable magnetometer_module - activitiy WALK_TO_STAND: Reject normal distributions | p-
value=0.0
Variable magnetometer_module - activitiy WALK&TALK: Reject normal distributions | p-valu
e=0.0
Variable magnetometer_module - activitiy STAND_TO_CLIMB_STAIRS: Reject normal distributi
ons | p-value=0.0
Variable magnetometer_module - activitiy CLIMB_STAIRS: Reject normal distributions | p-v
alue=0.0
Variable magnetometer_module - activitiy CLIMB_STAIRS_TO_WALK: Reject normal distributio
ns | p-value=0.0
Variable magnetometer_module - activitiy CLIMB_STAIRS&TALK: Reject normal distributions
| p-value=0.0
Variable magnetometer_module - activitiy CLIMB_STAIRS&TALK_TO_WALK&TALK: Reject normal d
istributions | p-value=0.0
```



Pela observação so gráfico e pelo valor obtido no p-value concluí-mos que o módulo do vetor do magnetómetro não segue uma distribuição normal.

Pelos resultados obtidos, verifica-se que, em todas as variáveis, o que valor resultante do p-value usando o teste de Kolmogorov-Smirnov é nulo, ou perto disso. Assim, concluí-mos que nenhum dos vetores segue uma distribuição normal.

Em suma :

- os módulos dos vetores não seguem uma distribuição normal/gaussiana, pelo que temos de escolher um teste não paramétrico (apesar de termos uma dimensão bastante elevada de valores de input)
- temos mais de 2 grupos de dados de valores contínuos (no total existem 16 atividades) que não estão emparelhadas

Segundo o *Choosing Statistical Tests* (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2881615/), escolhemos o teste de Kruskal-Wallis para calcular a significância estatística das variáveis e verificar se os dados para as diferentes variáveis são semelhantes (mesma média) out não

In [12]:
```
kruskal_wallis_test(right_wrist_data)
```

```
Kruscal - Wallis test - accelerometer_module
p-value: 0.0
Reject!
Kruscal - Wallis test - gyroscope_module
p-value: 0.0
Reject!
Kruscal - Wallis test - magnetometer_module
p-value: 0.0
Reject!
```

Como se pode observar, o p-value resultante do teste de Kruska-Wallis para todos os vetores e actividades, é sempre nulo, pelo que rejeitamos a hipótese de que os dados seguem a mesma distribuição.

Exercise 4.2 - feature extraction

Após ler o artigo *A Feature Selection-Based Framework for Human Activity Recognition Using Wearable*

Multimodal Sensors (https://www.semanticscholar.org/paper/A-feature-selection-based-framework-for-human-using-Zhang-Sawchuk/d67fa24e9877b0773e76af259441278ee3cf8bc4?p2df), verificou-se que seria necessário recolher features estatísticas e físicas. Seguindo o que os autores fizeram, apenas serão utilizados os dados da perna direita (device_id = 4) na extração das features.

Features Estatísticas

As features calculadas serão:

- média
- mediana
- desvio padrão
- variância
- RMS - root mean square
- média das derivadas de primeira ordem
- intervalo inter-quartile
- zero crossing rate
- mean crossing rate
- spectral entropy
- correlação entre as diferentes features

Features Físicas

As features físicas utilizadas serão:

- média da intensidade de movimento (AI) - apenas para o vetor de aceleração
- variância da intensidade do movimento (VI) - apenas para o vetor de aceleração
- eigenvalues da direção dominante no vetor de aceleração (EVA)
- correlação entre direção da gravidade e as restantes direções no vetor de aceleração (CAGH)
- média da velocidade ao longo da direção - (AVH)
- média da velocidade ao longo da direção da gravidade (AVG)
- média da rotação dos ângulos dada a direção da gravidade (ARATG)
- frequência dominante em cada vetor direção (DF)
- energia de cada vetor direção (Energy)
- média da energia da aceleração (AAE)
- média da energia da rotação (ARE)

Seguindo a discussão dos resultados dos autores, não se irá utilizar a feature área normalizada da magnitude do sinal (SMA) uma vez que é redundante se já tivermos a média da intensidade do movimento (AI).

Assim, no total, termos 172 features que descrevem todo o dataset

In [11]:
```
sensors_columns = [
    'accelerometer_x', 'accelerometer_y', 'accelerometer_z',
    'gyroscope_x', 'gyroscope_y', 'gyroscope_z',
    'magnetometer_x', 'magnetometer_y', 'magnetometer_z',
]
```

In [91]:
```
features_path = os.path.join(RESULTS_PATH, 'extracted_features.csv')
```

In [16]:
```
# select rows
```

```
device_id = 4 # Right Thigh
sensors_data = get_device_data(data_with_features, device_id)[sensors_columns]
```

Frequency

Segundo o README do dataset, as amostas foram recolhidas a uma frequencia de amostraggem de 51.2 Hz

In [17]:
```
fs = 51.2
t = 1 / fs
```

Window size and overlap

De forma a evitar o aliasing, iremos calcular as features por janela, sendo que iremos usar um tamanho de janela de 3 periodos e uma sobreposição de 60%

In [18]:
```
window_periods = 3
window_size = int(window_periods / t)
overlap = 0.6
step = int(window_size * overlap)
```

In [19]:
```
%%time
extracted_data = extract_features(pd.DataFrame((sensors_data)), window_size, fs, step, w
```

```
8763
CPU times: total: 8min 10s
Wall time: 13min 58s
```

In [29]:
```
extracted_data.to_csv(features_path, index=False)
```

Exercise 4.3 - PCA

Para saber como utilizar o PCA do sklearn, foi utilizado *Müller Andreas C & Guido S. (2017). Introduction to machine learning with python : a guide for data scientists (First). O'Reilly Media*

Após verificar que se gerariam muitos principal components (PC), foi limitado este número para apenas 10, uma vez notou em testes anteriores que o PC1 e PC2 é que seriam mais significativos, sendo os restantes muito perto de zero.

In [92]:
```
features_data = pd.read_csv(features_path)
features_data
```

Out[92]:

	accelerometer_x_mean	accelerometer_x_median	accelerometer_x_std	accelerometer_x_variance	accelerometer
0	-0.642723	-0.64332	0.031736	0.001007	0
1	-0.617430	-0.61705	0.047394	0.002246	0
2	-0.562038	-0.56780	0.047931	0.002297	0
3	-0.522878	-0.51853	0.037411	0.001400	0
4	-0.537807	-0.53994	0.037037	0.001372	0
...	...	...	...	...	
8758	0.144423	0.14169	0.029543	0.000873	0
8759	0.130531	0.13001	0.025618	0.000656	0
8760	0.113802	0.10762	0.023492	0.000552	0

8761	0.115606	0.11737	0.021704	0.000471	0
8762	0.105449	0.10568	0.026225	0.000688	0

8763 rows × 172 columns

O dataset de features extraidas contém 119 features diferentes que representam o dataset original.

In [97]:
```
from sklearn.decomposition import PCA
```

In [100…
```
scaled_data = calculate_zscore(features_data)
```

In [48]:
```
pca = PCA()
pca.fit(scaled_data)
pca_data = pca.transform(scaled_data)
```

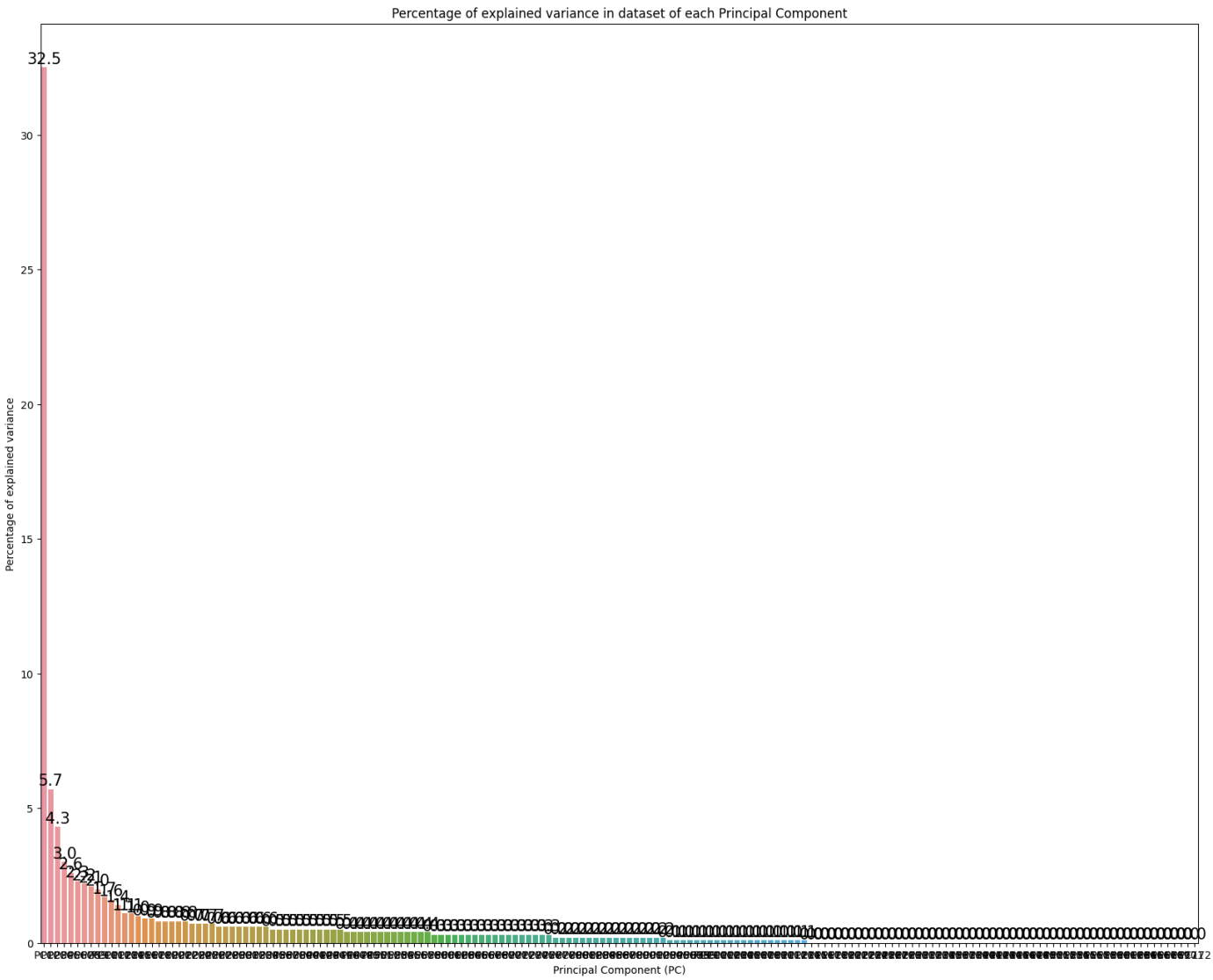
In [49]:
```
# calculate percentage variation of each principal component
percentage_var = np.round(pca.explained_variance_ratio_ * 100, decimals=1)
```

In [50]:
```
# labels of each principal component
labels_pca = [f'PC{x}' for x in range(1, len(percentage_var) + 1)]
```

In [51]:
```
def plot_pc(labels_pca, percentage_var, title='Percentage of explained variance in datas
    plt.figure(figsize=(20, 16))
    plots = sns.barplot(x='Principal Component', y='Percentage of explained varience',
                        data={'Principal Component' : labels_pca,
                              'Percentage of explained varience': percentage_var})
    for bar in plots.patches:
        plots.annotate(format(bar.get_height(), '.1f'),
                       (bar.get_x() + bar.get_width() / 2,
                           bar.get_height()), ha='center', va='center',
                       size=15, xytext=(0, 8),
                       textcoords='offset points')
    plt.xlabel('Principal Component (PC)')
    plt.ylabel('Percentage of explained variance')
    plt.title(title)
    plt.show()
```

In [ ]:
```
```

In [52]:
```
plot_pc(labels_pca, percentage_var)
```



In [53]:
```
print('Principal components to achieve at least 75% of explained variance:')
cumulative_sum = np.cumsum(percentage_var)
relevant_pc = np.argmax(cumulative_sum >= 75)
labels_pca[:relevant_pc]
```

```
Principal components to achieve at least 75% of explained variance:
```

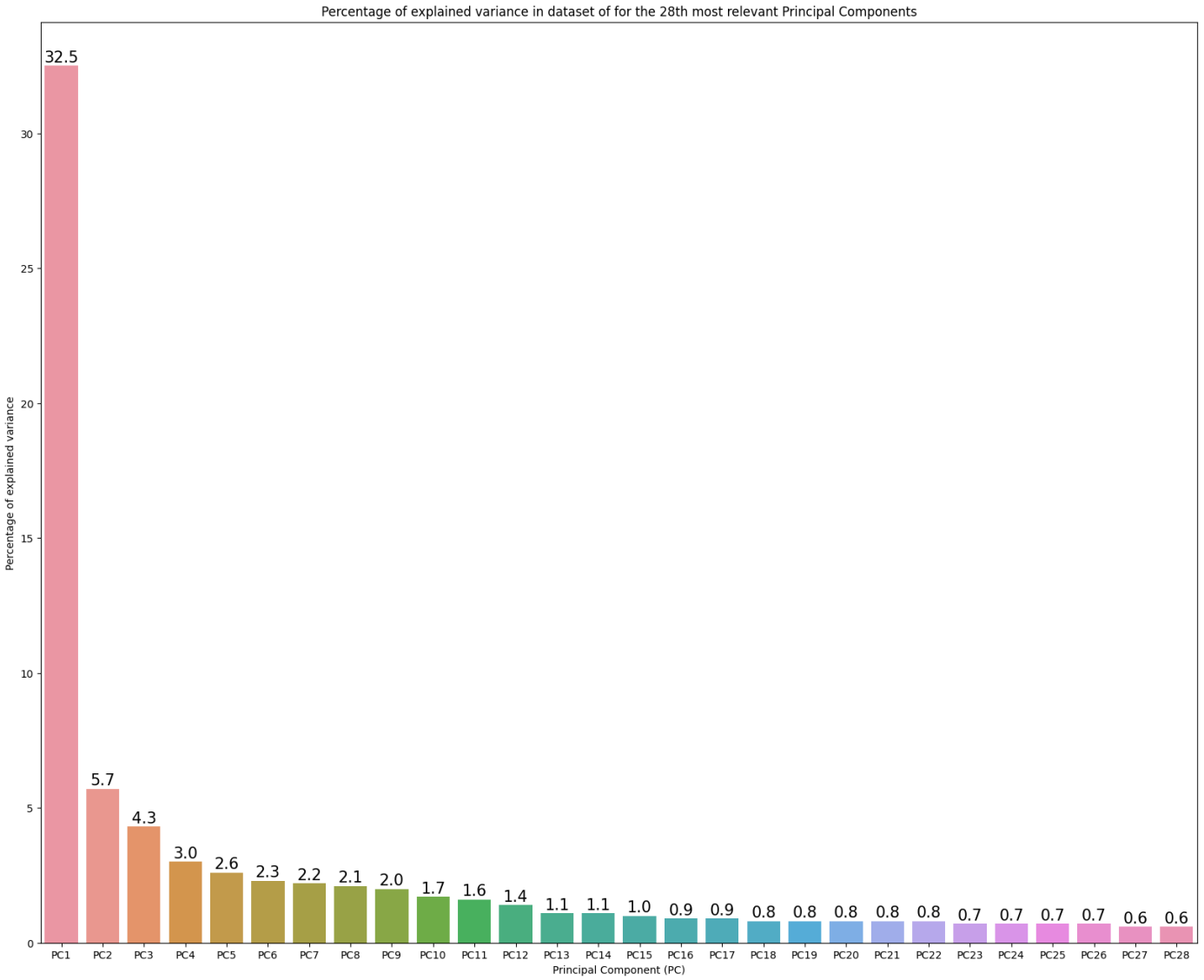
Out[53]:
```
['PC1',
 'PC2',
 'PC3',
 'PC4',
 'PC5',
 'PC6',
 'PC7',
 'PC8',
 'PC9',
 'PC10',
 'PC11',
 'PC12',
 'PC13',
 'PC14',
 'PC15',
 'PC16',
 'PC17',
 'PC18',
 'PC19',
 'PC20',
 'PC21',
 'PC22',
 'PC23',
 'PC24',
```

```
 'PC25',
 'PC26',
 'PC27',
 'PC28']
```

In [80]:
```
print(f'Como de pode observar, a componente que melhor descreve o dataset é a componente
```

```
Como de pode observar, a componente que melhor descreve o dataset é a componente PC1, de
screvendo 32.5% dos dados. Para conseguir descrever pelo menos 75% dos dados são necessá
rios os primeiros 28 principal components
```

In [82]:
```
plot_pc(labels_pca[:relevant_pc], percentage_var[:relevant_pc], title=f'Percentage of ex
```



In [55]:
```
pca_dataframe = pd.DataFrame(pca.components_.T, index=features_data.columns, columns=lab
pca_dataframe
```

Out[55]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	
accelerometer_x_mean	0.019390	0.027811	0.007095	-0.252328	0.155031	-0.201217	0.067290	-0.037596	0
accelerometer_x_median	0.024242	0.022822	0.018504	-0.255495	0.149311	-0.193023	0.067396	-0.049686	0
accelerometer_x_std	0.127368	0.035842	-0.031690	0.000278	0.012759	0.029632	0.021736	-0.050589	0
accelerometer_x_variance	0.114851	0.066705	-0.067376	0.043735	0.034695	0.043014	0.012989	-0.054069	0
accelerometer_x_rms	0.082407	0.115046	-0.014515	-0.102012	0.057561	-0.019891	0.043007	-0.060995	0

...	...	...	...	...	...	...	...	...	
avh	-0.033913	-0.000004	-0.176453	0.101571	-0.027193	-0.069785	0.070271	-0.189158	-0
avg	0.019404	0.027839	0.007080	-0.252271	0.155063	-0.201202	0.067314	-0.037611	0
aratg	-0.028112	-0.025165	0.006242	-0.030683	-0.081214	-0.166005	0.142237	0.141013	0
aae	0.125014	0.077351	-0.054727	0.049377	0.017308	-0.014512	-0.017366	0.005708	-0
are	0.125014	0.077351	-0.054727	0.049377	0.017308	-0.014512	-0.017366	0.005708	-0

172 rows × 172 columns

In [56]:
```
pca_dataframe_relevant = pca_dataframe.loc[:, pca_dataframe.columns[:relevant_pc]]
pca_dataframe_relevant
```

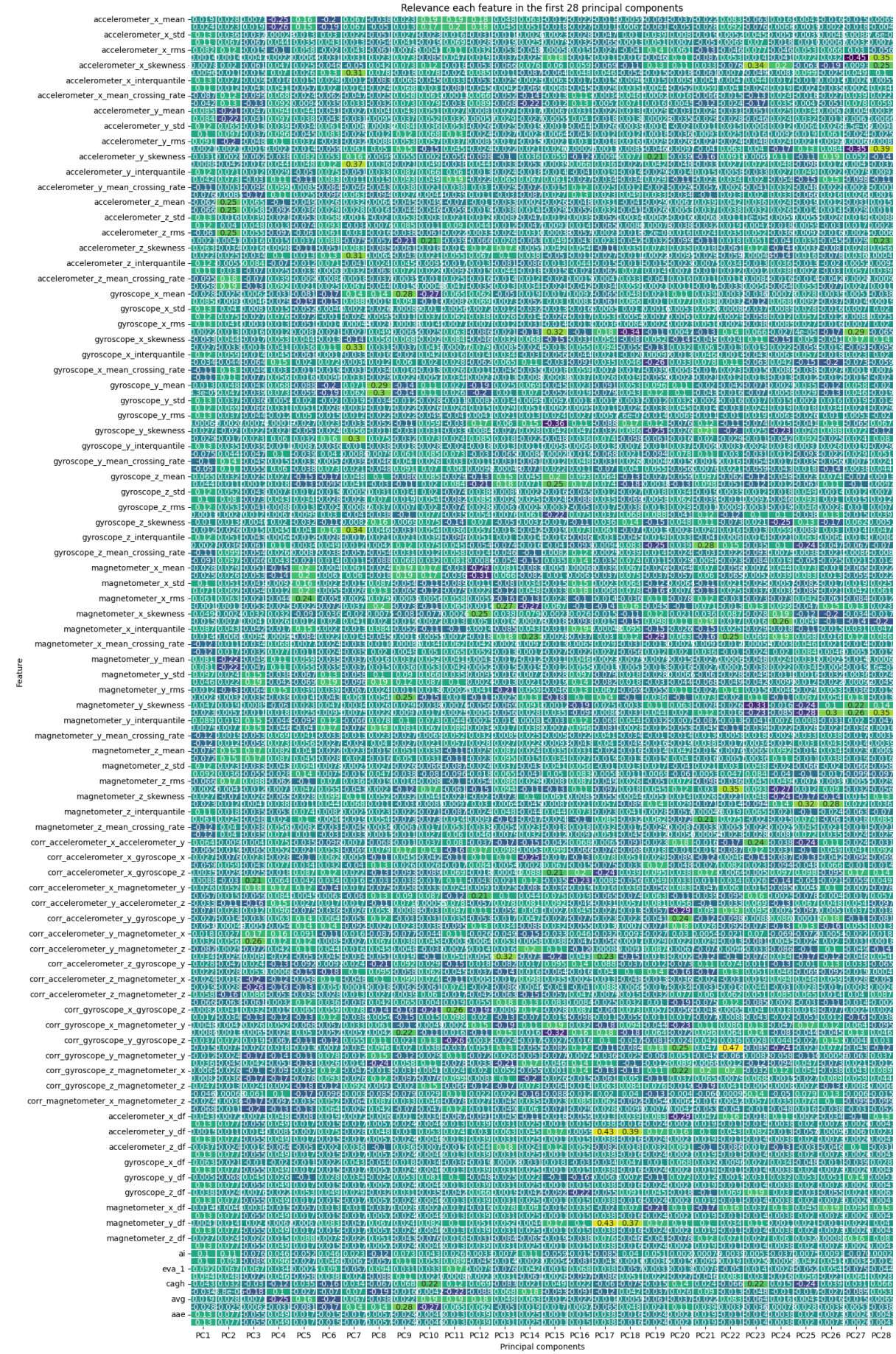
Out[56]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	
accelerometer_x_mean	0.019390	0.027811	0.007095	-0.252328	0.155031	-0.201217	0.067290	-0.037596	0
accelerometer_x_median	0.024242	0.022822	0.018504	-0.255495	0.149311	-0.193023	0.067396	-0.049686	0
accelerometer_x_std	0.127368	0.035842	-0.031690	0.000278	0.012759	0.029632	0.021736	-0.050589	0
accelerometer_x_variance	0.114851	0.066705	-0.067376	0.043735	0.034695	0.043014	0.012989	-0.054069	0
accelerometer_x_rms	0.082407	0.115046	-0.014515	-0.102012	0.057561	-0.019891	0.043007	-0.060995	0
...	...	...	...	...	...	...	...	...	
avh	-0.033913	-0.000004	-0.176453	0.101571	-0.027193	-0.069785	0.070271	-0.189158	-0
avg	0.019404	0.027839	0.007080	-0.252271	0.155063	-0.201202	0.067314	-0.037611	0
aratg	-0.028112	-0.025165	0.006242	-0.030683	-0.081214	-0.166005	0.142237	0.141013	0
aae	0.125014	0.077351	-0.054727	0.049377	0.017308	-0.014512	-0.017366	0.005708	-0
are	0.125014	0.077351	-0.054727	0.049377	0.017308	-0.014512	-0.017366	0.005708	-0

172 rows × 28 columns

In [57]:
```
def plot_relevance(dataframe, relevant=None):
    plt.figure(figsize=(20, 30))
    sns.heatmap(dataframe, cmap='viridis',linewidths=0.3, annot=True)
    plt.xlabel('Principal components')
    plt.ylabel('Feature')
    plt.title(f'Relevance each feature in the first {relevant} principal components')
    plt.show()
```

In [58]:
```
plot_relevance(pca_dataframe_relevant, relevant_pc)
```



In [76]:
```
print('Most relevant feature of each Principal Component')
pca_dataframe_relevant.idxmax()
```

Out[76]:
```
Most relevant feature of each Principal Component
PC1                          gyroscope_x_rms
PC2                    accelerometer_z_rms
PC3      corr_accelerometer_y_magnetometer_y
```

```
PC4     corr_accelerometer_x_magnetometer_y
PC5                    magnetometer_x_rms
PC6                magnetometer_y_variance
PC7               accelerometer_y_kurtosis
PC8                     gyroscope_y_median
PC9                       gyroscope_x_mean
PC10                                  cagh
PC11          corr_gyroscope_x_gyroscope_z
PC12               magnetometer_x_skewness
PC13      corr_accelerometer_z_gyroscope_x
PC14      magnetometer_x_non_crossing_rate
PC15           gyroscope_x_avg_derivatives
PC16      corr_accelerometer_x_gyroscope_z
PC17                  accelerometer_y_df
PC18                  accelerometer_y_df
PC19              accelerometer_y_skewness
PC20       corr_gyroscope_y_magnetometer_x
PC21         gyroscope_z_non_crossing_rate
PC22       corr_gyroscope_y_magnetometer_x
PC23              accelerometer_x_skewness
PC24               magnetometer_x_kurtosis
PC25               magnetometer_z_kurtosis
PC26               magnetometer_y_kurtosis
PC27           gyroscope_x_avg_derivatives
PC28        accelerometer_y_avg_derivatives
dtype: object
```

In [77]:
```
pca_features = pd.DataFrame(data=np.zeros(pca_dataframe_relevant.shape), columns=pca_dat
```

In [74]:
```
for column in pca_features.columns:
    pca_features[column] = pd.Series(pca_dataframe_relevant.sort_values(by=column, ascen
```

In [75]:
```
pca_features
```

Out[75]:

	PC1	PC2	PC3	
0	gyroscope_x_rms	accelerometer_z_rms	corr_accelerometer_y_magnetometer_y	cor
1	gyroscope_x_std	accelerometer_z_mean	corr_accelerometer_x_magnetometer_x	cor
2	gyroscope_y_std	accelerometer_z_median	magnetometer_y_variance	co
3	gyroscope_y_rms	accelerometer_z_spectral_entropy	corr_accelerometer_x_magnetometer_y	
4	accelerometer_x_std	accelerometer_z_mean_crossing_rate	corr_accelerometer_y_magnetometer_x	
...	...	...	...	
167	magnetometer_x_spectral_entropy	accelerometer_y_rms	corr_gyroscope_y_magnetometer_y	cor
168	magnetometer_z_spectral_entropy	accelerometer_y_mean	avh	
169	magnetometer_y_mean_crossing_rate	accelerometer_y_median	corr_magnetometer_y_magnetometer_z	
170	magnetometer_z_mean_crossing_rate	magnetometer_y_median	corr_accelerometer_z_magnetometer_x	
171	magnetometer_x_mean_crossing_rate	magnetometer_y_mean	corr_accelerometer_z_magnetometer_y	

172 rows × 28 columns

A tabela anterior mostra as features ordenadas, de forma descendente (mais relevante primeiro), por relevância para cada Principal Component.

Aa grande vantagem de utilizar o PCA nas features extraídas é que reduz a dimensionalidade do dataset. POr exemplo, para descrever 75% do dataset apenas precisamos de 28 principal component (variáveis), em

vez das 172 features que existiam inicialmente.