Package 'edm1'

June 20, 2024

Title Set of statistical functions

Version 2.0.0.0

Description Provides functions to sort normal distribution data from an unordered dataframe, extract normal data from a random dataset, a function to calculate the probability of an event given a normal distribution...

License GPL (==3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports stringr,
stringi

Contents

`all_stat` *all_stat*

Description

Allow to see all the main statistics indicators (mean, median, variance, standard deviation, sum, max, min, quantile) of variables in a dataframe by the modality of a variable in a column of the input datarame. In addition to that, you can get the occurence of other qualitative variables by your chosen qualitative variable, you have just to precise it in the vector "stat_var" where all the statistics indicators are given with "occu-var_you_want/".

Usage

```
all_stat(inpt_v, var_add = c(), stat_var = c(), inpt_datf)
```

Arguments

inpt_v	is the modalities of the variables
var_add	is the variables you want to get the stats from
stat_var	is the stats indicators you want
inpt_datf	is the input dataframe

Examples

```
datf <- data.frame("mod"=c("first", "seco", "seco", "first", "first", "third", "first"),
                   "var1"=c(11, 22, 21, 22, 22, 11, 9),
                  "var2"=c("d", "d", "z", "z", "z", "d", "z"),
                  "var3"=c(45, 44, 43, 46, 45, 45, 42),
                 "var4"=c("A", "A", "A", "A", "B", "C", "C"))

print(all_stat(inpt_v=c("first", "seco"), var_add = c("var1", "var2", "var3", "var4"),
 stat_var=c("sum", "mean", "median", "sd", "occu-var2/", "occu-var4/", "variance",
"quantile-0.75/"),
 inpt_datf=datf))

#   modal_v var_vector occu sum mean  med standard_devaition         variance
#1    first
#2               var1       64   16 16.5   6.97614984548545 48.6666666666667
#3             var2-d    1
#4             var2-z    3
#5               var3      178 44.5   45   1.73205080756888                3
#6             var4-A    2
#7             var4-B    1
#8             var4-C    1
#9     seco
#10              var1       43 21.5 21.5  0.707106781186548              0.5
#11            var2-d    1
#12            var2-z    1
#13              var3       87 43.5 43.5  0.707106781186548              0.5
#14            var4-A    2
#15            var4-B    0
#16            var4-C    0
#   quantile-0.75
#1
#2             22
#3
#4
#5          45.25
#6
#7
#8
#9
#10         21.75
#11
#12
#13         43.75
#14
```

```
#15
#16
```

`extract_normal` *extract_normal*

Description

Allow to extract values that fits a normal distribution from any kind of dataset, see examples and parameters

Usage

```
extract_normal(
  inpt_datf,
  mean,
  sd,
  accuracy,
  round_value = 1,
  normalised = FALSE,
  n = NA,
  tries = 3
)
```

Arguments

`inpt_datf`	is the input dataset as a dataframe, values/modalities are in the first column and frequency (not normalised) is in the second column
`mean`	is the mean of the target normal distribution
`sd`	is the standard deviation of the target normal distribution
`accuracy`	is how much of a difference beetween the points of the targeted normal distribution and the actual points is tolerated
`round_value`	is the round value for the normal distribution used under the hood to compare the dataset and extract the best points, defaults to 1
`normalised`	is if the input frequency is divided by n, if TRUE the parameter `n` must be filled
`n`	is the number of points
`tries`	is how many normal distributions are used under the hood to compare their points to the those in the input dataset, defaults to 3. The higher it is, the higher the number of different points from the input dataset will be in accordance for the normal distribution the function tries to build from the dataset. It does not increase by a lot but can be non-negligible and note that the higher the number of tries is, the higher the execution time of the function will be.

Examples

```
sample_val <- round(rnorm(n = 72000, mean = 12, sd = 2), 1)
sample_freq <- unique_total(sample_val)
sample_qual <- infinite_char_seq(n = length(sample_freq))
datf_test <- data.frame(sample_qual, sample_freq)
n <- nrow(datf_test)
print(datf_test)
```

```
   sample_qual sample_freq
1            a          72
2            b        1155
3            c        1255
4            d         743
5            e         696
6            f        1028
7            g        1160
8            h        1219
9            i        1353
10           j        1336
11           k        1308
12           l         485
13           m        1306
14           n        1429
15           o         623
16           p        1172
17           q        1054
18           r         999
19           s         125
20           t        1461
21           u        1430
22           v         341
23           w        1453
24           x         427
25           y         869
26           z        1395
27          aa         841
28          ab         952
29          ac         246
30          ad         468
31          ae         237
32          af         555
33          ag        1297
34          ah         571
35          ai         349
36          aj         773
37          ak        1086
38          al        1281
39          am        1471
40          an        1236
41          ao         394
42          ap        1433
43          aq        1328
44          ar         976
45          as         640
46          at         308
47          au         698
```

48	av	864
49	aw	1346
50	ax	1349
51	ay	6
52	az	1071
53	ba	248
54	bb	929
55	bc	925
56	bd	452
57	be	207
58	bf	546
59	bg	62
60	bh	107
61	bi	1184
62	bj	739
63	bk	624
64	bl	850
65	bm	1408
66	bn	620
67	bo	202
68	bp	10
69	bq	700
70	br	397
71	bs	1291
72	bt	178
73	bu	397
74	bv	1089
75	bw	1301
76	bx	328
77	by	1348
78	bz	97
79	ca	1452
80	cb	4
81	cc	100
82	cd	593
83	ce	503
84	cf	164
85	cg	32
86	ch	259
87	ci	1089
88	cj	249
89	ck	165
90	cl	42
91	cm	143
92	cn	467
93	co	347
94	cp	143
95	cq	69
96	cr	18
97	cs	290
98	ct	55
99	cu	141
100	cv	86
101	cw	303
102	cx	88
103	cy	16
104	cz	213

```
105          da           3
106          db          75
107          dc          32
108          dd          66
109          de         105
110          df          34
111          dg          56
112          dh          17
113          di          22
114          dj         120
115          dk          54
116          dl           9
117          dm           8
118          dn          36
119          do          20
120          dp          26
121          dq          54
122          dr           8
123          ds          10
124          dt           4
125          du          53
126          dv          29
127          dw           1
128          dx           8
129          dy          10
130          dz           4
131          ea          22
132          eb           9
133          ec          17
134          ed          55
135          ee          21
136          ef           6
137          eg           4
138          eh           3
139          ei           7
140          ej           1
141          ek           4
142          el           2
143          em           5
144          en           4
145          eo           1
146          ep           2
147          eq           3
148          er           8
149          es           4
150          et           3
151          eu           3
152          ev           2
153          ew           2
154          ex           2
155          ey           1
156          ez           2
157          fa           2
158          fb           1

teste <- extract_normal(inpt_datf = datf_test,
                        mean = 10,
```

```
                      sd = 2,
                      accuracy = .1,
                      round_value = 1,
                      normalised = FALSE,
                      tries = 5)

print(length(unique(teste[, 1])) / n)

[1] 0.2848101 # so nearly 28.5 % of the different points were in
 #accordance with the construction of the target normal distribution

print(teste)

   values    frequency
1       dw 0.0001406866
2       dw 0.0001406866
3       dw 0.0001406866
4       el 0.0002813731
5       el 0.0002813731
6       el 0.0002813731
7       el 0.0002813731
8       da 0.0004220597
9       da 0.0004220597
10      cb 0.0005627462
11      cb 0.0005627462
12      em 0.0007034328
13      ay 0.0008441193
14      ay 0.0008441193
15      ei 0.0009848059
16      ei 0.0009848059
17      ei 0.0009848059
18      dm 0.0011254924
19      bp 0.0014068655
20      cy 0.0022509848
21      cy 0.0022509848
22      cy 0.0022509848
23      dh 0.0023916714
24      dh 0.0023916714
25      cr 0.0025323579
26      ee 0.0029544176
27      di 0.0030951041
28      dp 0.0036578503
29      dp 0.0036578503
30      cg 0.0045019696
31      cg 0.0045019696
32      df 0.0047833427
33      dn 0.0050647158
34      cl 0.0059088351
35      cl 0.0059088351
36      du 0.0074563872
37      du 0.0074563872
38      dg 0.0078784468
39      dg 0.0078784468
40      bg 0.0087225661
41      bg 0.0087225661
42      dd 0.0092853123
43      cq 0.0097073720
```

```
44      cq 0.0097073720
45       a 0.0101294316
46      cv 0.0120990433
47      cx 0.0123804164
48      cx 0.0123804164
49      bz 0.0136465954
50      cc 0.0140686550
51      bh 0.0150534609
52      bh 0.0150534609
53      dj 0.0168823860
54       s 0.0175858188
55       s 0.0175858188
56      cm 0.0201181767
57      cf 0.0230725943
58      ck 0.0232132808
59      bt 0.0250422060
60      bt 0.0250422060
61      be 0.0291221159
62      be 0.0291221159
63      cz 0.0299662352
64      cz 0.0299662352
65      be 0.0291221159
66      bo 0.0284186832
67      bt 0.0250422060
68      ck 0.0232132808
69      ck 0.0232132808
70      cm 0.0201181767
71      cu 0.0198368036
72       s 0.0175858188
73      dj 0.0168823860
74      bh 0.0150534609
75      bh 0.0150534609
76      de 0.0147720878
77      bz 0.0136465954
78      bz 0.0136465954
79      cx 0.0123804164
80      cv 0.0120990433
81      db 0.0105514913
82       a 0.0101294316
83      cq 0.0097073720
84      dd 0.0092853123
85      dd 0.0092853123
86      bg 0.0087225661
87      bg 0.0087225661
88      dg 0.0078784468
89      dk 0.0075970737
90      du 0.0074563872
91      cl 0.0059088351
92      cl 0.0059088351
93      dn 0.0050647158
94      df 0.0047833427
95      df 0.0047833427
96      cg 0.0045019696
97      dv 0.0040799100
98      dp 0.0036578503
99      di 0.0030951041
100     di 0.0030951041
```

```
101     ee 0.0029544176
102     cr 0.0025323579
103     dh 0.0023916714
104     cy 0.0022509848
105     cy 0.0022509848
106     cy 0.0022509848
107     cy 0.0022509848
108     dl 0.0012661790
109     dm 0.0011254924
110     ei 0.0009848059
111     ei 0.0009848059
112     ay 0.0008441193
113     ay 0.0008441193
114     em 0.0007034328
115     em 0.0007034328
116     cb 0.0005627462
117     cb 0.0005627462
118     da 0.0004220597
119     da 0.0004220597
120     el 0.0002813731
121     el 0.0002813731
122     el 0.0002813731
123     el 0.0002813731
124     dw 0.0001406866
125     dw 0.0001406866
126     dw 0.0001406866
```

`how_normal` *how_normal*

Description

Allow to get how much a sequence of numbers fit a normal distribution with chosen parameters, see examples

Usage

```
how_normal(inpt_datf, normalised = TRUE, mean = 0, sd = 1)
```

Arguments

`inpt_datf`	is the input dataframe containing all the values in the first column and their frequency (normalised or no), in the second column
`normalised`	is a boolean, takes TRUE if the frequency for each value is divided by n, FALSE if not
`mean`	is the mean of the normal distribution that the dataset tries to fit
`sd`	is the standard deviation of the normal distribution the dataset tries to fit

Examples

```
sample_val <- round(rnorm(n = 12000, mean = 6, sd = 1.25), 1)
sample_freq <- unique_total(sample_val)
datf_test <- data.frame(unique(sample_val), sample_freq)
print(datf_test)
```

```
   unique.sample_val. sample_freq
1                 6.9         306
2                 8.3          63
3                 7.7         148
4                 5.6         363
5                 6.5         349
6                 4.6         202
7                 6.6         324
8                 6.7         335
9                 6.0         406
10                5.7         365
11                7.9         109
12                6.2         420
13                5.9         386
14                4.5         185
15                5.1         326
16                6.1         360
17                5.5         346
18                6.3         375
19                7.4         207
20                7.6         162
21                4.2         129
22                3.9         102
23                5.2         325
24                2.3           7
25                5.8         387
26                6.4         319
27                9.1          21
28                7.0         280
29                8.8          27
30                4.9         218
31                8.1          98
32                3.0          25
33                8.4          66
34                4.3         160
35                7.2         267
36                8.7          40
37                5.3         313
38                4.1         127
39                5.0         275
40                4.0         119
41                9.3          13
42                4.4         196
43                6.8         313
44                7.1         247
45                3.5          57
46                7.8         139
47                3.6          57
48                7.5         189
49                7.3         215
```

```
50                4.7         230
51                3.2          36
52                9.5           8
53                3.8          79
54                8.2          62
55                5.4         343
56                8.5          55
57                4.8         207
58                3.7          79
59                8.6          33
60                3.3          38
61                3.4          43
62                8.9          21
63                8.0         105
64                3.1          23
65                9.0          27
66               10.0           5
67                2.5          10
68                2.9          16
69                9.7           7
70                2.7          11
71               10.5           1
72                9.4          13
73                9.2          16
74                2.6          16
75                9.9           3
76                2.8          10
77                2.4          10
78                1.9           2
79                2.0           6
80               10.2           2
81                9.6           3
82               11.3           1
83                1.8           1
84                2.2           3
85                2.1           2
86                1.6           1
87               10.6           1
88                9.8           1
89               10.4           1
90                1.7           1

print(how_normal(inpt_datf = datf_test,
                 normalised = FALSE,
                 mean = 6,
                 sd = 1))

[1] 9.003683

print(how_normal(inpt_datf = datf_test,
                 normalised = FALSE,
                 mean = 5,
                 sd = 1))

[1] 9.098484
```

how_unif *how_unif*

Description

Allow to see how much a sequence of numbers fit a uniform distribution, see examples

Usage

```
how_unif(inpt_v, normalised = TRUE)
```

Arguments

normalised	is a boolean, takes TRUE if the frequency for each value is divided by n, FALSE if not
inpt_datf	is the input dataframe containing all the values in the first column and their frequencyu at the second column

Examples

```
sample_val <- round(runif(n = 12000, min = 24, max = 27), 1)
sample_freq <- unique_total(sample_val)
datf_test <- data.frame(unique(sample_val), sample_freq)

print(datf_test)
```

```
   unique.sample_val. sample_freq
1                24.4         400
2                24.8         379
3                25.5         414
4                26.0         366
5                26.6         400
6                25.7         419
7                24.3         389
8                24.1         423
9                26.1         404
10               26.5         406
11               26.2         356
12               26.8         407
13               24.6         388
14               25.3         402
15               26.3         388
16               25.4         422
17               25.0         436
18               25.9         373
19               25.2         423
20               25.6         388
21               27.0         202
22               24.2         380
23               24.9         404
24               25.1         417
25               26.4         401
26               26.7         431
27               24.5         392
```

```
28              24.0         218
29              26.9         407
30              25.8         371
31              24.7         394

print(how_unif(inpt_datf = datf_test, normalised = FALSE))

[1] 0.0752957

sample_val <- round(rnorm(n = 12000, mean = 24, sd = 7), 1)
sample_freq <- unique_total(sample_val)
datf_test <- data.frame(unique(sample_val), sample_freq)

print(how_unif(inpt_datf = datf_test, normalised = FALSE))

[1] 0.7797352
```

`normal_dens` *normal_dens*

Description

Calculates the normal distribution probality, see examples

Usage

```
normal_dens(target_v = c(), mean, sd)
```

Arguments

`target_v`	is the target value(s) (one or bounded), see examples
`mean`	is the mean of the normal distribution
`sd`	is the standard deviation of the normal distribution

Examples

```
print(normal_dens(target_v = 13, mean = 12, sd = 2))

[1] 0.1760327

print(normal_dens(target_v = c(9, 11), mean = 12, sd = 1.5, step = 0.01))

[1] 0.2288579

print(normal_dens(target_v = c(1, 18), mean = 12, sd = 1.5, step = 0.01))

[1] 0.9999688
```

`sort_normal_qual` *sort_normal_qual*

Description

Sort qualitative modalities that have their frequency normally distributed from an unordered dataset, see examples. This function uses an another algorythm than choose_normal_qual2 which may be faster.

Usage

```
sort_normal_qual(inpt_datf)
```

Arguments

`inpt_datf`	is the input dataframe, containing the values in the first column and their frequency in the second

Examples

```
sample_val <- round(rnorm(n = 2000, mean = 12, sd = 2), 1)
sample_freq <- unique_total(sample_val)
sample_qual <- infinite_char_seq(n = length(sample_freq))
datf_test <- data.frame(sample_qual, sample_freq)
datf_test[, 2] <- datf_test[, 2] / sum(datf_test[, 2]) # optional

print(datf_test)
```

```
   sample_qual sample_freq
1            a 0.208695652
2            b 0.234782609
3            c 0.321739130
4            d 0.339130435
5            e 0.330434783
6            f 0.069565217
7            g 0.234782609
8            h 0.400000000
9            i 0.347826087
10           j 0.043478261
11           k 0.278260870
12           l 0.286956522
13           m 0.243478261
14           n 0.147826087
15           o 0.234782609
16           p 0.252173913
17           q 0.417391304
18           r 0.095652174
19           s 0.313043478
20           t 0.008695652
21           u 0.130434783
22           v 0.391304348
23           w 0.113043478
24           x 0.295652174
25           y 0.243478261
```

```
26           z 0.382608696
27          aa 0.008695652
28          ab 0.347826087
29          ac 0.330434783
30          ad 0.321739130
31          ae 0.347826087
32          af 0.321739130
33          ag 0.173913043
34          ah 0.278260870
35          ai 0.278260870
36          aj 0.347826087
37          ak 0.026086957
38          al 0.295652174
39          am 0.226086957
40          an 0.295652174
41          ao 0.234782609
42          ap 0.113043478
43          aq 0.234782609
44          ar 0.173913043
45          as 0.017391304
46          at 0.252173913
47          au 0.078260870
48          av 0.086956522
49          aw 0.278260870
50          ax 0.086956522
51          ay 0.200000000
52          az 0.295652174
53          ba 0.052173913
54          bb 0.165217391
55          bc 0.408695652
56          bd 0.269565217
57          be 0.104347826
58          bf 0.391304348
59          bg 0.104347826
60          bh 0.043478261
61          bi 0.200000000
62          bj 0.095652174
63          bk 0.191304348
64          bl 0.008695652
65          bm 0.165217391
66          bn 0.226086957
67          bo 0.086956522
68          bp 0.017391304
69          bq 0.121739130
70          br 0.234782609
71          bs 0.121739130
72          bt 0.078260870
73          bu 0.173913043
74          bv 0.104347826
75          bw 0.208695652
76          bx 0.017391304
77          by 0.243478261
78          bz 0.034782609
79          ca 0.017391304
80          cb 0.008695652
81          cc 0.173913043
82          cd 0.147826087
```

```
83           ce 0.060869565
84           cf 0.017391304
85           cg 0.060869565
86           ch 0.008695652
87           ci 0.208695652
88           cj 0.043478261
89           ck 0.052173913
90           cl 0.017391304
91           cm 0.017391304
92           cn 0.095652174
93           co 0.113043478
94           cp 0.017391304
95           cq 0.017391304
96           cr 0.026086957
97           cs 0.034782609
98           ct 0.017391304
99           cu 0.026086957
100          cv 0.026086957
101          cw 0.026086957
102          cx 0.017391304
103          cy 0.043478261
104          cz 0.008695652
105          da 0.034782609
106          db 0.017391304
107          dc 0.060869565
108          dd 0.008695652
109          de 0.008695652
110          df 0.017391304
111          dg 0.008695652
112          dh 0.008695652
113          di 0.017391304
114          dj 0.008695652
115          dk 0.008695652

print(sort_normal_qual(inpt_datf = datf_test))

0.00869565217391304 0.00869565217391304 0.00869565217391304 0.00869565217391304
              "aa"                "cb"                "cz"                "de"
0.00869565217391304 0.00869565217391304  0.0173913043478261  0.0173913043478261
              "dh"                "dk"                "bp"                "ca"
0.0173913043478261  0.0173913043478261  0.0173913043478261  0.0173913043478261
              "cl"                "cp"                "ct"                "db"
0.0173913043478261  0.0260869565217391  0.0260869565217391  0.0347826086956522
              "di"                "cr"                "cv"                "bz"
0.0347826086956522  0.0434782608695652  0.0434782608695652  0.0521739130434783
              "da"                "bh"                "cy"                "ck"
0.0608695652173913  0.0695652173913043  0.0782608695652174  0.0869565217391304
              "cg"                 "f"                "bt"                "ax"
0.0956521739130435  0.0956521739130435   0.104347826086957    0.11304347826087
               "r"                "cn"                "bg"                 "w"
  0.11304347826087   0.121739130434783   0.147826086956522   0.165217391304348
              "co"                "bs"                 "n"                "bb"
 0.173913043478261   0.173913043478261   0.191304347826087                 0.2
              "ag"                "bu"                "bk"                "bi"
 0.208695652173913   0.226086956521739   0.234782608695652   0.234782608695652
              "bw"                "am"                 "b"                 "o"
 0.234782608695652   0.243478260869565   0.243478260869565   0.252173913043478
```

```
              "aq"                "m"               "by"               "at"
 0.278260869565217  0.278260869565217  0.28695652173913  0.295652173913043
              "k"               "ai"                "l"               "al"
 0.295652173913043  0.321739130434783  0.321739130434783  0.330434782608696
              "az"                "c"               "af"               "ac"
 0.347826086956522  0.347826086956522  0.382608695652174  0.391304347826087
              "i"               "ae"                "z"               "bf"
 0.408695652173913  0.417391304347826                0.4  0.391304347826087
             "bc"                "q"                "h"                "v"
 0.347826086956522  0.347826086956522  0.339130434782609  0.330434782608696
             "aj"               "ab"                "d"                "e"
 0.321739130434783   0.31304347826087  0.295652173913043  0.295652173913043
             "ad"                "s"               "an"                "x"
 0.278260869565217  0.278260869565217  0.269565217391304  0.252173913043478
             "aw"               "ah"               "bd"                "p"
 0.243478260869565  0.234782608695652  0.234782608695652  0.234782608695652
              "y"               "br"               "ao"                "g"
 0.226086956521739  0.208695652173913  0.208695652173913                0.2
             "bn"               "ci"                "a"               "ay"
 0.173913043478261  0.173913043478261  0.165217391304348  0.147826086956522
             "cc"               "ar"               "bm"               "cd"
 0.130434782608696  0.121739130434783   0.11304347826087  0.104347826086957
              "u"               "bq"               "ap"               "bv"
 0.104347826086957 0.0956521739130435 0.0869565217391304 0.0869565217391304
             "be"               "bj"               "bo"               "av"
0.0782608695652174 0.0608695652173913 0.0608695652173913 0.0521739130434783
             "au"               "dc"               "ce"               "ba"
0.0434782608695652 0.0434782608695652 0.0347826086956522 0.0260869565217391
             "cj"                "j"               "cs"               "cw"
0.0260869565217391 0.0260869565217391 0.0173913043478261 0.0173913043478261
             "cu"               "ak"               "df"               "cx"
0.0173913043478261 0.0173913043478261 0.0173913043478261 0.0173913043478261
             "cq"               "cm"               "cf"               "bx"
0.0173913043478261 0.00869565217391304 0.00869565217391304 0.00869565217391304
             "as"               "dj"               "dg"               "dd"
0.00869565217391304 0.00869565217391304 0.00869565217391304
             "ch"               "bl"                "t"
```

`sort_normal_qual2` *sort_normal_qual2*

Description

Sort qualitative modalities that have their frequency normally distributed from an unordered dataset, see examples. This function uses an another algorythm than choose_normal_qual which may be faster.

Usage

```
sort_normal_qual2(inpt_datf)
```

Arguments

`inpt_datf`	is the input dataframe, containing the values in the first column and their frequency in the second

Examples

```
sample_val <- round(rnorm(n = 2000, mean = 12, sd = 2), 1)
sample_freq <- unique_total(sample_val)
sample_qual <- infinite_char_seq(n = length(sample_freq))
datf_test <- data.frame(sample_qual, sample_freq)
datf_test[, 2] <- datf_test[, 2] / sum(datf_test[, 2])

print(datf_test)
```

```
   sample_qual sample_freq
1            a 0.208695652
2            b 0.234782609
3            c 0.321739130
4            d 0.339130435
5            e 0.330434783
6            f 0.069565217
7            g 0.234782609
8            h 0.400000000
9            i 0.347826087
10           j 0.043478261
11           k 0.278260870
12           l 0.286956522
13           m 0.243478261
14           n 0.147826087
15           o 0.234782609
16           p 0.252173913
17           q 0.417391304
18           r 0.095652174
19           s 0.313043478
20           t 0.008695652
21           u 0.130434783
22           v 0.391304348
23           w 0.113043478
24           x 0.295652174
25           y 0.243478261
26           z 0.382608696
27          aa 0.008695652
28          ab 0.347826087
29          ac 0.330434783
30          ad 0.321739130
31          ae 0.347826087
32          af 0.321739130
33          ag 0.173913043
34          ah 0.278260870
35          ai 0.278260870
36          aj 0.347826087
37          ak 0.026086957
38          al 0.295652174
39          am 0.226086957
40          an 0.295652174
41          ao 0.234782609
42          ap 0.113043478
43          aq 0.234782609
44          ar 0.173913043
45          as 0.017391304
46          at 0.252173913
```

```
47          au 0.078260870
48          av 0.086956522
49          aw 0.278260870
50          ax 0.086956522
51          ay 0.200000000
52          az 0.295652174
53          ba 0.052173913
54          bb 0.165217391
55          bc 0.408695652
56          bd 0.269565217
57          be 0.104347826
58          bf 0.391304348
59          bg 0.104347826
60          bh 0.043478261
61          bi 0.200000000
62          bj 0.095652174
63          bk 0.191304348
64          bl 0.008695652
65          bm 0.165217391
66          bn 0.226086957
67          bo 0.086956522
68          bp 0.017391304
69          bq 0.121739130
70          br 0.234782609
71          bs 0.121739130
72          bt 0.078260870
73          bu 0.173913043
74          bv 0.104347826
75          bw 0.208695652
76          bx 0.017391304
77          by 0.243478261
78          bz 0.034782609
79          ca 0.017391304
80          cb 0.008695652
81          cc 0.173913043
82          cd 0.147826087
83          ce 0.060869565
84          cf 0.017391304
85          cg 0.060869565
86          ch 0.008695652
87          ci 0.208695652
88          cj 0.043478261
89          ck 0.052173913
90          cl 0.017391304
91          cm 0.017391304
92          cn 0.095652174
93          co 0.113043478
94          cp 0.017391304
95          cq 0.017391304
96          cr 0.026086957
97          cs 0.034782609
98          ct 0.017391304
99          cu 0.026086957
100         cv 0.026086957
101         cw 0.026086957
102         cx 0.017391304
103         cy 0.043478261
```

```
104          cz 0.008695652
105          da 0.034782609
106          db 0.017391304
107          dc 0.060869565
108          dd 0.008695652
109          de 0.008695652
110          df 0.017391304
111          dg 0.008695652
112          dh 0.008695652
113          di 0.017391304
114          dj 0.008695652
115          dk 0.008695652

print(sort_normal_qual2(inpt_datf = datf_test))

0.00869565217391304 0.00869565217391304 0.00869565217391304 0.00869565217391304
              "aa"                "cb"                "cz"                "de"
0.00869565217391304 0.00869565217391304  0.0173913043478261  0.0173913043478261
              "dh"                "dk"                "bp"                "ca"
0.0173913043478261  0.0173913043478261  0.0173913043478261  0.0173913043478261
              "cl"                "cp"                "ct"                "db"
0.0173913043478261  0.0260869565217391  0.0260869565217391  0.0347826086956522
              "di"                "cr"                "cv"                "bz"
0.0347826086956522  0.0434782608695652  0.0434782608695652  0.0521739130434783
              "da"                "bh"                "cy"                "ck"
0.0608695652173913  0.0695652173913043  0.0782608695652174  0.0869565217391304
              "cg"                 "f"                "bt"                "ax"
0.0956521739130435  0.0956521739130435   0.104347826086957    0.11304347826087
               "r"                "cn"                "bg"                 "w"
  0.11304347826087   0.121739130434783   0.147826086956522   0.165217391304348
              "co"                "bs"                 "n"                "bb"
 0.173913043478261   0.173913043478261   0.191304347826087                 0.2
              "ag"                "bu"                "bk"                "bi"
 0.208695652173913   0.226086956521739   0.234782608695652   0.234782608695652
              "bw"                "am"                 "b"                 "o"
 0.234782608695652   0.243478260869565   0.243478260869565   0.252173913043478
              "aq"                 "m"                "by"                "at"
 0.278260869565217   0.278260869565217    0.28695652173913   0.295652173913043
               "k"                "ai"                 "l"                "al"
 0.295652173913043   0.321739130434783   0.321739130434783   0.330434782608696
              "az"                 "c"                "af"                "ac"
 0.347826086956522   0.347826086956522   0.382608695652174   0.391304347826087
               "i"                "ae"                 "z"                "bf"
 0.408695652173913   0.417391304347826                 0.4   0.391304347826087
              "bc"                 "q"                 "h"                 "v"
 0.347826086956522   0.347826086956522   0.339130434782609   0.330434782608696
              "aj"                "ab"                 "d"                 "e"
 0.321739130434783    0.31304347826087   0.295652173913043   0.295652173913043
              "ad"                 "s"                "an"                 "x"
 0.278260869565217   0.278260869565217   0.269565217391304   0.252173913043478
              "aw"                "ah"                "bd"                 "p"
 0.243478260869565   0.234782608695652   0.234782608695652   0.234782608695652
               "y"                "br"                "ao"                 "g"
 0.226086956521739   0.208695652173913   0.208695652173913                 0.2
              "bn"                "ci"                 "a"                "ay"
 0.173913043478261   0.173913043478261   0.165217391304348   0.147826086956522
              "cc"                "ar"                "bm"                "cd"
```

```
 0.130434782608696   0.121739130434783    0.11304347826087   0.104347826086957
               "u"                 "bq"                 "ap"                 "bv"
 0.104347826086957  0.095652173913043  0.0869565217391304  0.0869565217391304
              "be"                 "bj"                 "bo"                 "av"
0.0782608695652174  0.0608695652173913  0.0608695652173913  0.0521739130434783
              "au"                 "dc"                 "ce"                 "ba"
0.0434782608695652  0.0434782608695652  0.0347826086956522  0.0260869565217391
              "cj"                  "j"                 "cs"                 "cw"
0.0260869565217391  0.0260869565217391  0.0173913043478261  0.0173913043478261
              "cu"                 "ak"                 "df"                 "cx"
0.0173913043478261  0.0173913043478261  0.0173913043478261  0.0173913043478261
              "cq"                 "cm"                 "cf"                 "bx"
0.0173913043478261 0.00869565217391304 0.00869565217391304 0.00869565217391304
              "as"                 "dj"                 "dg"                 "dd"
0.00869565217391304 0.00869565217391304 0.00869565217391304
              "ch"                 "bl"                  "t"
```

Index