

Dustin Van Tate Testa
CS484 Assignment 1

1. See attachment 'q1.py' for full python program
 - a. $A = 26$; $B = 36$

```
cs484 / assignment1 / q1.py
1  # imports
2  import pandas as pd;
3  import numpy as np;
4  import matplotlib.pyplot as plt;
5  import math
6
7  # Load data
8  data = pd.read_csv('NormalSample.csv')
9
10 #####
11 # Q1.a
12 #####
13 xs = data['x']
14 a = math.ceil(min(xs))
15 b = math.floor(max(xs))
16 print('\nQ1.a')
17 print('\tmin x: ', min(xs))
18 print('\ttherefore, a = ', a)
19
20 print('\tmax x: ', max(xs))
21 print('\ttherefore, b = ', b)
22
23 #####
24 # Q1.b-e
25 #####
```

Terminal - tate@archbook

TypeError: object of type 'filter' has no attribute 'x'

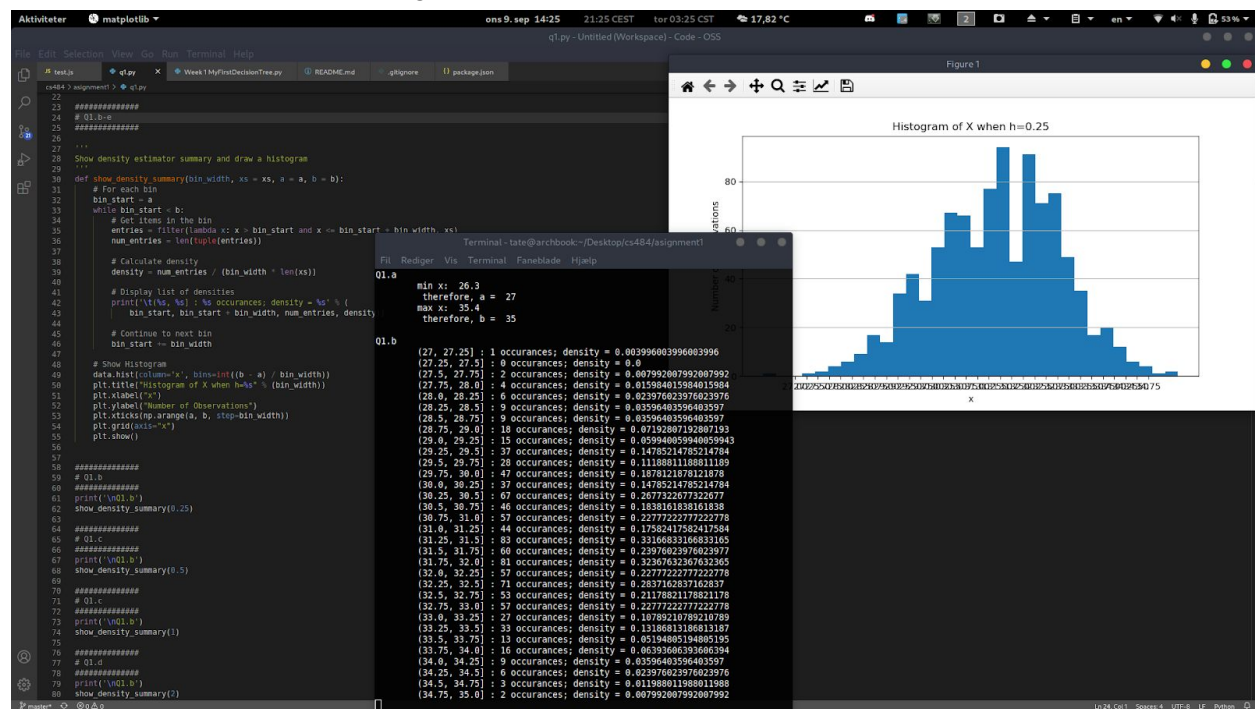
Q1.a

min x: 26.3
therefore, a = 27
max x: 35.4
therefore, b = 35

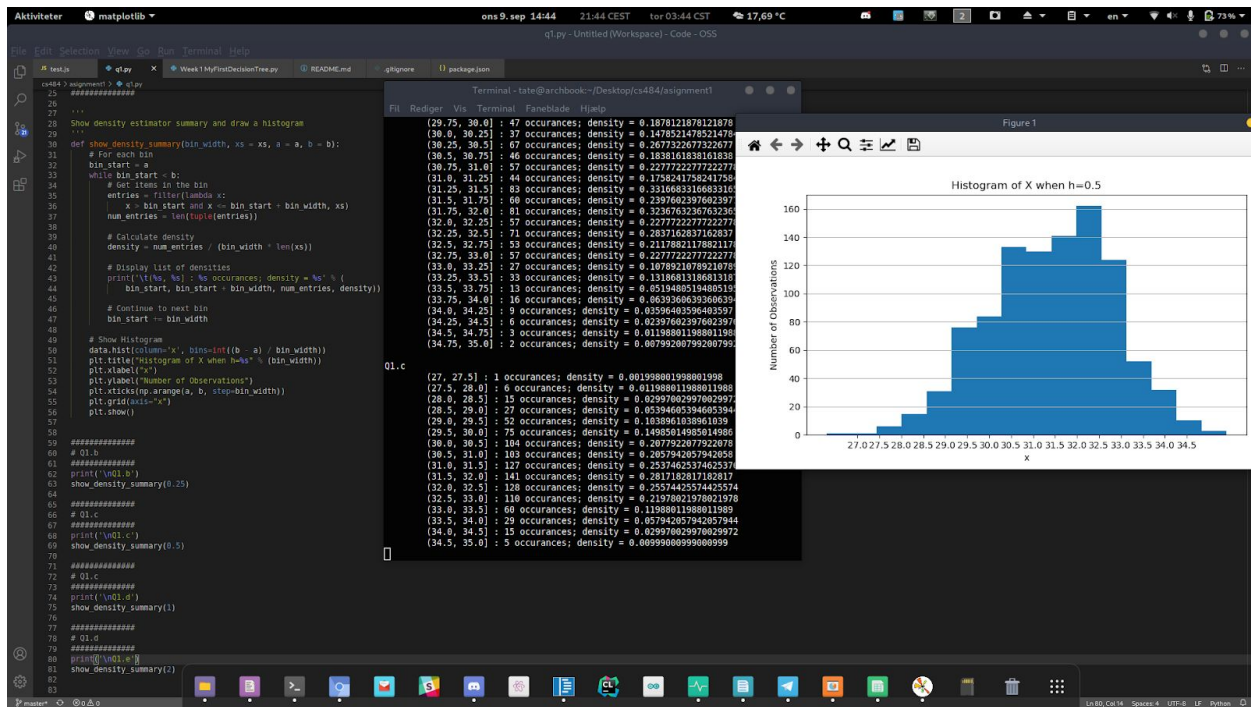
Q1.b

(27, 27.25] : 1 occurrences; density = 0.003996603996603996
(27.25, 27.5] : 0 occurrences; density = 0.0
(27.5, 27.75] : 2 occurrences; density = 0.007993207993207992
(27.75, 28.0] : 4 occurrences; density = 0.015986415986415984
(28.0, 28.25] : 6 occurrences; density = 0.023976623976623976
(28.25, 28.5] : 9 occurrences; density = 0.03596493596493597
(28.5, 28.75] : 18 occurrences; density = 0.03596493596493597
(28.75, 29.0] : 15 occurrences; density = 0.07192871928719287
(29.0, 29.25] : 15 occurrences; density = 0.03596493596493597
(29.25, 29.5] : 37 occurrences; density = 0.14785214785214784
(29.5, 29.75] : 28 occurrences; density = 0.11888118881188811
(29.75, 30.0] : 47 occurrences; density = 0.187321217812178
(30.0, 30.25] : 37 occurrences; density = 0.14785214785214784
(30.25, 30.5] : 67 occurrences; density = 0.2677322773227732
(30.5, 30.75] : 46 occurrences; density = 0.1838161381613816
(30.75, 31.0] : 57 occurrences; density = 0.22772227722722778
(31.0, 31.25] : 44 occurrences; density = 0.17582417582417584
(31.25, 31.5] : 83 occurrences; density = 0.3316833168331683
(31.5, 31.75] : 66 occurrences; density = 0.25976023976023977
(31.75, 32.0] : 81 occurrences; density = 0.32367632367632365
(32.0, 32.25] : 57 occurrences; density = 0.22772227722722778
(32.25, 32.5] : 71 occurrences; density = 0.2837162371623716
(32.5, 32.75] : 93 occurrences; density = 0.21178821178821178
(32.75, 33.0] : 57 occurrences; density = 0.22772227722722778
(33.0, 33.25] : 27 occurrences; density = 0.10789210789210789
(33.25, 33.5] : 33 occurrences; density = 0.13186813186813187
(33.5, 33.75] : 13 occurrences; density = 0.05194805194805195
(33.75, 34.0] : 18 occurrences; density = 0.0639360393603936
(34.0, 34.25] : 9 occurrences; density = 0.03596493596493597
(34.25, 34.5] : 6 occurrences; density = 0.023976623976623976
(34.5, 34.75] : 3 occurrences; density = 0.011988011988011988
(34.75, 35.0] : 2 occurrences; density = 0.007993207993207992

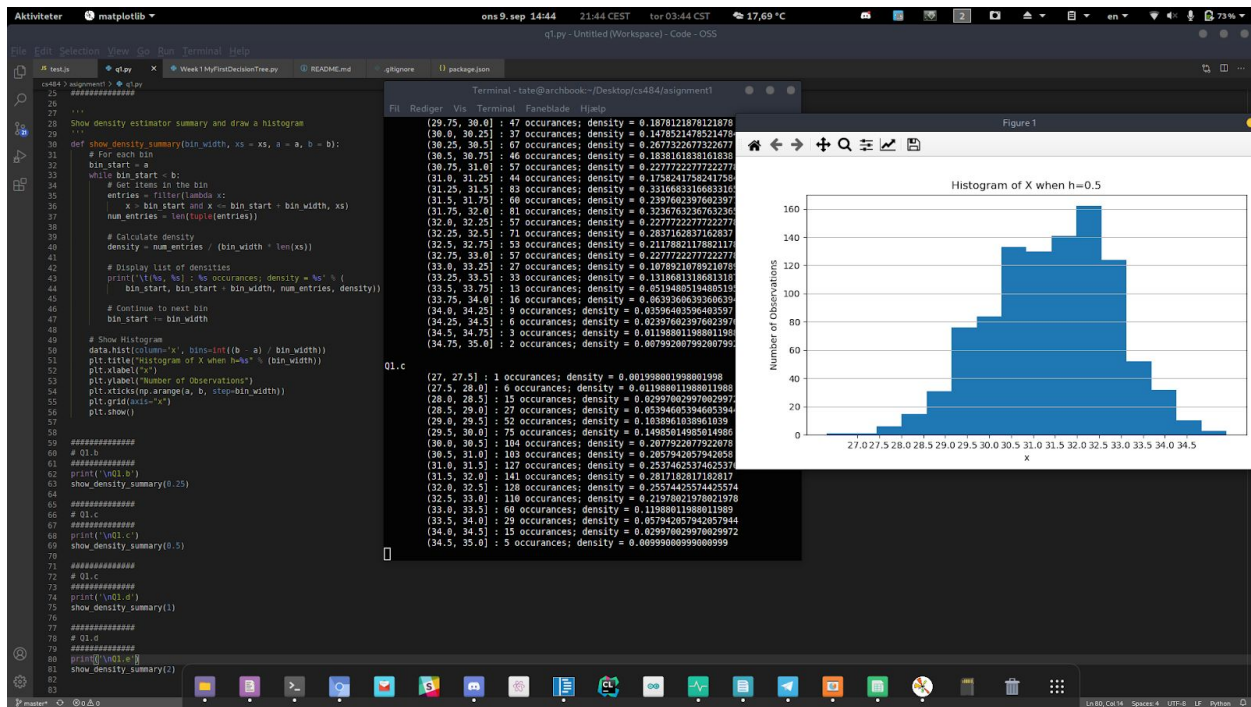
- b. List: see terminal window, histogram: see plot



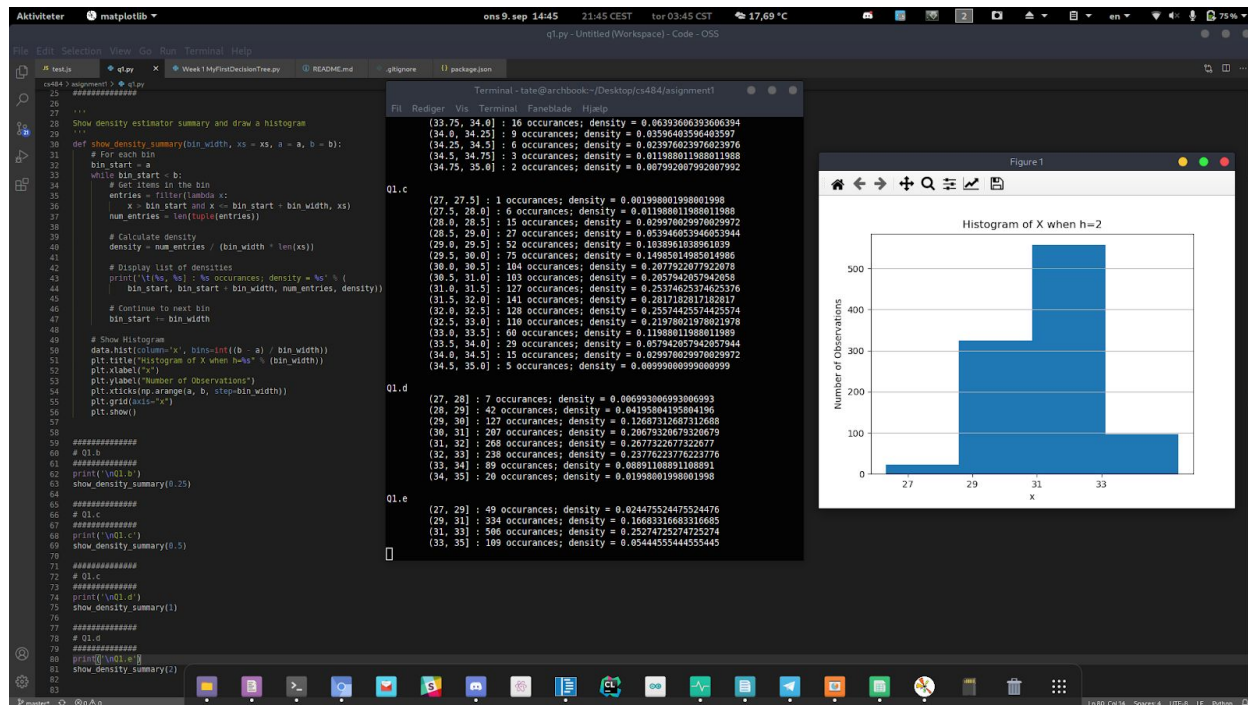
c. List: see terminal, histogram see plot



d. List: see terminal, histogram: see plot



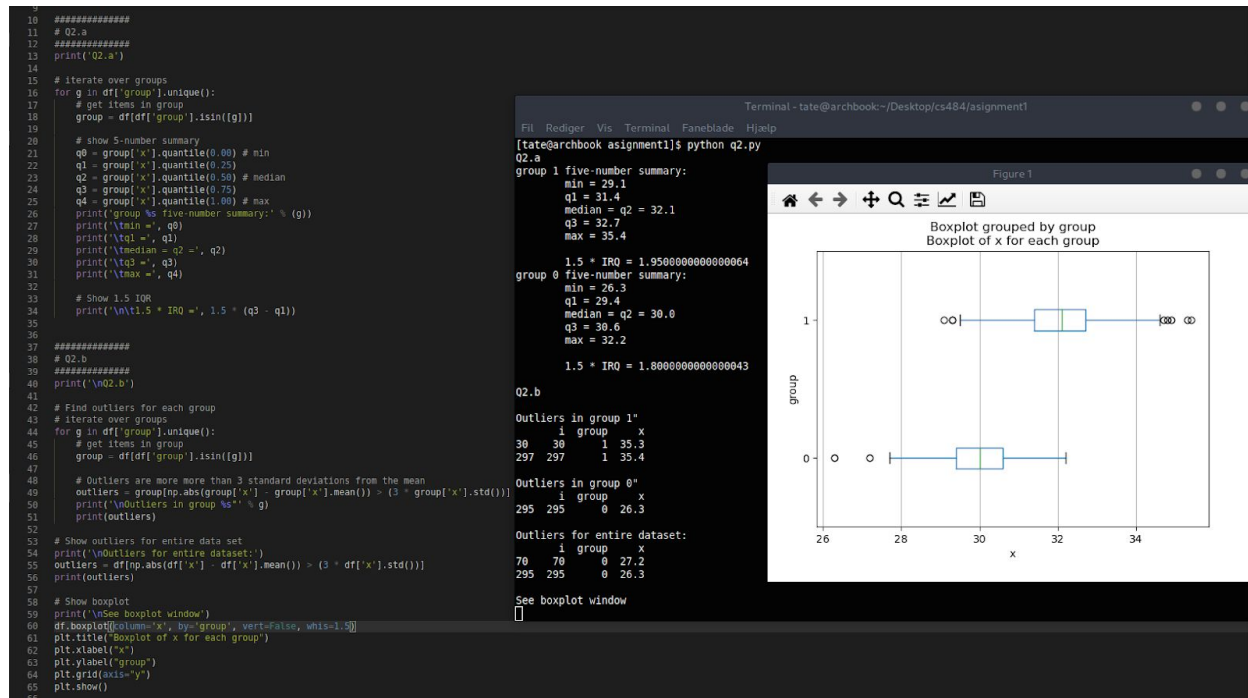
- e. List: see terminal, plot see chart



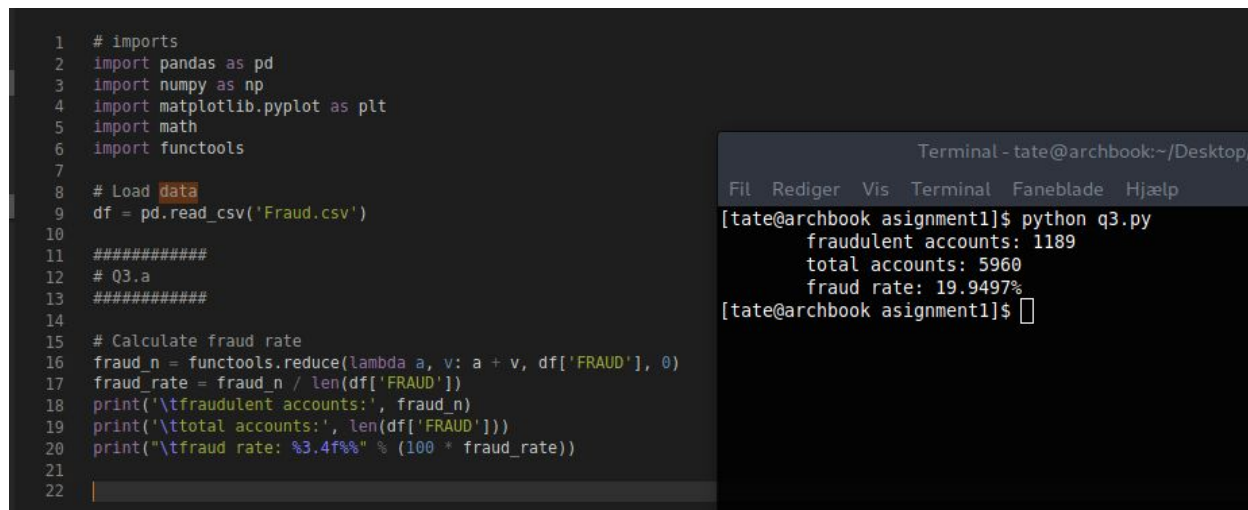
- f. According to the strategy suggested by izenman, A. J. (1991) in the american statistical association journal, the ideal bin width (h) is ~ 0.4 , among the 4 we tested here the 0.5 one is closest to this value. Without context however, this value might not be the most appropriate. And there are some minor things you can see in the $h=0.25$ histogram that aren't present in the $h=0.5$ data set. For example, the $x=(31.5, 31.75]$ interval has fewer members than its neighbors, which although unlikely, could potentially be statistically significant. For example, if these are ages of humans in years there may have been an approximately 3 month period with world events that caused fertility rates to decrease during that period.

```

4 #####
5 # Q1.f
6 #####
7 print('\nQ1.f')
8
9 # Print Interquartile-range
10 q1 = data.quantile(0.25)['x']
11 q3 = data.quantile(0.75)['x']
12 iqr = q3 - q1
13 print('\tinterquartile-range:', iqr)
14
15 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
16 print('\tIdeal bin-width: h =', ideal_h)
17
18 #####
19 # Q1.e
20 #####
21 print('\nQ1.e')
22
23 # Print Interquartile-range
24 q1 = data.quantile(0.25)['x']
25 q3 = data.quantile(0.75)['x']
26 iqr = q3 - q1
27 print('\tinterquartile-range:', iqr)
28
29 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
30 print('\tIdeal bin-width: h =', ideal_h)
31
32 #####
33 # Q1.d
34 #####
35 print('\nQ1.d')
36
37 # Print Interquartile-range
38 q1 = data.quantile(0.25)['x']
39 q3 = data.quantile(0.75)['x']
40 iqr = q3 - q1
41 print('\tinterquartile-range:', iqr)
42
43 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
44 print('\tIdeal bin-width: h =', ideal_h)
45
46 #####
47 # Q1.c
48 #####
49 print('\nQ1.c')
50
51 # Print Interquartile-range
52 q1 = data.quantile(0.25)['x']
53 q3 = data.quantile(0.75)['x']
54 iqr = q3 - q1
55 print('\tinterquartile-range:', iqr)
56
57 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
58 print('\tIdeal bin-width: h =', ideal_h)
59
60 #####
61 # Q1.b
62 #####
63 print('\nQ1.b')
64
65 # Print Interquartile-range
66 q1 = data.quantile(0.25)['x']
67 q3 = data.quantile(0.75)['x']
68 iqr = q3 - q1
69 print('\tinterquartile-range:', iqr)
70
71 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
72 print('\tIdeal bin-width: h =', ideal_h)
73
74 #####
75 # Q1.a
76 #####
77 print('\nQ1.a')
78
79 # Print Interquartile-range
80 q1 = data.quantile(0.25)['x']
81 q3 = data.quantile(0.75)['x']
82 iqr = q3 - q1
83 print('\tinterquartile-range:', iqr)
84
85 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
86 print('\tIdeal bin-width: h =', ideal_h)
87
88 #####
89 # Q1.g
90 #####
91 print('\nQ1.g')
92
93 # Print Interquartile-range
94 q1 = data.quantile(0.25)['x']
95 q3 = data.quantile(0.75)['x']
96 iqr = q3 - q1
97 print('\tinterquartile-range:', iqr)
98
99 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
100 print('\tIdeal bin-width: h =', ideal_h)
101
102 #####
103 # Q1.h
104 #####
105 print('\nQ1.h')
106
107 # Print Interquartile-range
108 q1 = data.quantile(0.25)['x']
109 q3 = data.quantile(0.75)['x']
110 iqr = q3 - q1
111 print('\tinterquartile-range:', iqr)
112
113 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
114 print('\tIdeal bin-width: h =', ideal_h)
115
116 #####
117 # Q1.i
118 #####
119 print('\nQ1.i')
120
121 # Print Interquartile-range
122 q1 = data.quantile(0.25)['x']
123 q3 = data.quantile(0.75)['x']
124 iqr = q3 - q1
125 print('\tinterquartile-range:', iqr)
126
127 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
128 print('\tIdeal bin-width: h =', ideal_h)
129
130 #####
131 # Q1.j
132 #####
133 print('\nQ1.j')
134
135 # Print Interquartile-range
136 q1 = data.quantile(0.25)['x']
137 q3 = data.quantile(0.75)['x']
138 iqr = q3 - q1
139 print('\tinterquartile-range:', iqr)
140
141 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
142 print('\tIdeal bin-width: h =', ideal_h)
143
144 #####
145 # Q1.k
146 #####
147 print('\nQ1.k')
148
149 # Print Interquartile-range
150 q1 = data.quantile(0.25)['x']
151 q3 = data.quantile(0.75)['x']
152 iqr = q3 - q1
153 print('\tinterquartile-range:', iqr)
154
155 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
156 print('\tIdeal bin-width: h =', ideal_h)
157
158 #####
159 # Q1.l
160 #####
161 print('\nQ1.l')
162
163 # Print Interquartile-range
164 q1 = data.quantile(0.25)['x']
165 q3 = data.quantile(0.75)['x']
166 iqr = q3 - q1
167 print('\tinterquartile-range:', iqr)
168
169 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
170 print('\tIdeal bin-width: h =', ideal_h)
171
172 #####
173 # Q1.m
174 #####
175 print('\nQ1.m')
176
177 # Print Interquartile-range
178 q1 = data.quantile(0.25)['x']
179 q3 = data.quantile(0.75)['x']
180 iqr = q3 - q1
181 print('\tinterquartile-range:', iqr)
182
183 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
184 print('\tIdeal bin-width: h =', ideal_h)
185
186 #####
187 # Q1.n
188 #####
189 print('\nQ1.n')
190
191 # Print Interquartile-range
192 q1 = data.quantile(0.25)['x']
193 q3 = data.quantile(0.75)['x']
194 iqr = q3 - q1
195 print('\tinterquartile-range:', iqr)
196
197 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
198 print('\tIdeal bin-width: h =', ideal_h)
199
200 #####
201 # Q1.o
202 #####
203 print('\nQ1.o')
204
205 # Print Interquartile-range
206 q1 = data.quantile(0.25)['x']
207 q3 = data.quantile(0.75)['x']
208 iqr = q3 - q1
209 print('\tinterquartile-range:', iqr)
210
211 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
212 print('\tIdeal bin-width: h =', ideal_h)
213
214 #####
215 # Q1.p
216 #####
217 print('\nQ1.p')
218
219 # Print Interquartile-range
220 q1 = data.quantile(0.25)['x']
221 q3 = data.quantile(0.75)['x']
222 iqr = q3 - q1
223 print('\tinterquartile-range:', iqr)
224
225 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
226 print('\tIdeal bin-width: h =', ideal_h)
227
228 #####
229 # Q1.q
230 #####
231 print('\nQ1.q')
232
233 # Print Interquartile-range
234 q1 = data.quantile(0.25)['x']
235 q3 = data.quantile(0.75)['x']
236 iqr = q3 - q1
237 print('\tinterquartile-range:', iqr)
238
239 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
240 print('\tIdeal bin-width: h =', ideal_h)
241
242 #####
243 # Q1.r
244 #####
245 print('\nQ1.r')
246
247 # Print Interquartile-range
248 q1 = data.quantile(0.25)['x']
249 q3 = data.quantile(0.75)['x']
250 iqr = q3 - q1
251 print('\tinterquartile-range:', iqr)
252
253 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
254 print('\tIdeal bin-width: h =', ideal_h)
255
256 #####
257 # Q1.s
258 #####
259 print('\nQ1.s')
260
261 # Print Interquartile-range
262 q1 = data.quantile(0.25)['x']
263 q3 = data.quantile(0.75)['x']
264 iqr = q3 - q1
265 print('\tinterquartile-range:', iqr)
266
267 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
268 print('\tIdeal bin-width: h =', ideal_h)
269
270 #####
271 # Q1.t
272 #####
273 print('\nQ1.t')
274
275 # Print Interquartile-range
276 q1 = data.quantile(0.25)['x']
277 q3 = data.quantile(0.75)['x']
278 iqr = q3 - q1
279 print('\tinterquartile-range:', iqr)
280
281 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
282 print('\tIdeal bin-width: h =', ideal_h)
283
284 #####
285 # Q1.u
286 #####
287 print('\nQ1.u')
288
289 # Print Interquartile-range
290 q1 = data.quantile(0.25)['x']
291 q3 = data.quantile(0.75)['x']
292 iqr = q3 - q1
293 print('\tinterquartile-range:', iqr)
294
295 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
296 print('\tIdeal bin-width: h =', ideal_h)
297
298 #####
299 # Q1.v
300 #####
301 print('\nQ1.v')
302
303 # Print Interquartile-range
304 q1 = data.quantile(0.25)['x']
305 q3 = data.quantile(0.75)['x']
306 iqr = q3 - q1
307 print('\tinterquartile-range:', iqr)
308
309 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
310 print('\tIdeal bin-width: h =', ideal_h)
311
312 #####
313 # Q1.w
314 #####
315 print('\nQ1.w')
316
317 # Print Interquartile-range
318 q1 = data.quantile(0.25)['x']
319 q3 = data.quantile(0.75)['x']
320 iqr = q3 - q1
321 print('\tinterquartile-range:', iqr)
322
323 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
324 print('\tIdeal bin-width: h =', ideal_h)
325
326 #####
327 # Q1.x
328 #####
329 print('\nQ1.x')
330
331 # Print Interquartile-range
332 q1 = data.quantile(0.25)['x']
333 q3 = data.quantile(0.75)['x']
334 iqr = q3 - q1
335 print('\tinterquartile-range:', iqr)
336
337 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
338 print('\tIdeal bin-width: h =', ideal_h)
339
340 #####
341 # Q1.y
342 #####
343 print('\nQ1.y')
344
345 # Print Interquartile-range
346 q1 = data.quantile(0.25)['x']
347 q3 = data.quantile(0.75)['x']
348 iqr = q3 - q1
349 print('\tinterquartile-range:', iqr)
350
351 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
352 print('\tIdeal bin-width: h =', ideal_h)
353
354 #####
355 # Q1.z
356 #####
357 print('\nQ1.z')
358
359 # Print Interquartile-range
360 q1 = data.quantile(0.25)['x']
361 q3 = data.quantile(0.75)['x']
362 iqr = q3 - q1
363 print('\tinterquartile-range:', iqr)
364
365 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
366 print('\tIdeal bin-width: h =', ideal_h)
367
368 #####
369 # Q1.aa
370 #####
371 print('\nQ1.aa')
372
373 # Print Interquartile-range
374 q1 = data.quantile(0.25)['x']
375 q3 = data.quantile(0.75)['x']
376 iqr = q3 - q1
377 print('\tinterquartile-range:', iqr)
378
379 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
380 print('\tIdeal bin-width: h =', ideal_h)
381
382 #####
383 # Q1.ab
384 #####
385 print('\nQ1.ab')
386
387 # Print Interquartile-range
388 q1 = data.quantile(0.25)['x']
389 q3 = data.quantile(0.75)['x']
390 iqr = q3 - q1
391 print('\tinterquartile-range:', iqr)
392
393 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
394 print('\tIdeal bin-width: h =', ideal_h)
395
396 #####
397 # Q1.ac
398 #####
399 print('\nQ1.ac')
400
401 # Print Interquartile-range
402 q1 = data.quantile(0.25)['x']
403 q3 = data.quantile(0.75)['x']
404 iqr = q3 - q1
405 print('\tinterquartile-range:', iqr)
406
407 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
408 print('\tIdeal bin-width: h =', ideal_h)
409
410 #####
411 # Q1.ad
412 #####
413 print('\nQ1.ad')
414
415 # Print Interquartile-range
416 q1 = data.quantile(0.25)['x']
417 q3 = data.quantile(0.75)['x']
418 iqr = q3 - q1
419 print('\tinterquartile-range:', iqr)
420
421 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
422 print('\tIdeal bin-width: h =', ideal_h)
423
424 #####
425 # Q1.ae
426 #####
427 print('\nQ1.ae')
428
429 # Print Interquartile-range
430 q1 = data.quantile(0.25)['x']
431 q3 = data.quantile(0.75)['x']
432 iqr = q3 - q1
433 print('\tinterquartile-range:', iqr)
434
435 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
436 print('\tIdeal bin-width: h =', ideal_h)
437
438 #####
439 # Q1.af
440 #####
441 print('\nQ1.af')
442
443 # Print Interquartile-range
444 q1 = data.quantile(0.25)['x']
445 q3 = data.quantile(0.75)['x']
446 iqr = q3 - q1
447 print('\tinterquartile-range:', iqr)
448
449 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
450 print('\tIdeal bin-width: h =', ideal_h)
451
452 #####
453 # Q1.ag
454 #####
455 print('\nQ1.ag')
456
457 # Print Interquartile-range
458 q1 = data.quantile(0.25)['x']
459 q3 = data.quantile(0.75)['x']
460 iqr = q3 - q1
461 print('\tinterquartile-range:', iqr)
462
463 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
464 print('\tIdeal bin-width: h =', ideal_h)
465
466 #####
467 # Q1.ah
468 #####
469 print('\nQ1.ah')
470
471 # Print Interquartile-range
472 q1 = data.quantile(0.25)['x']
473 q3 = data.quantile(0.75)['x']
474 iqr = q3 - q1
475 print('\tinterquartile-range:', iqr)
476
477 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
478 print('\tIdeal bin-width: h =', ideal_h)
479
480 #####
481 # Q1.ai
482 #####
483 print('\nQ1.ai')
484
485 # Print Interquartile-range
486 q1 = data.quantile(0.25)['x']
487 q3 = data.quantile(0.75)['x']
488 iqr = q3 - q1
489 print('\tinterquartile-range:', iqr)
490
491 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
492 print('\tIdeal bin-width: h =', ideal_h)
493
494 #####
495 # Q1.aj
496 #####
497 print('\nQ1.aj')
498
499 # Print Interquartile-range
500 q1 = data.quantile(0.25)['x']
501 q3 = data.quantile(0.75)['x']
502 iqr = q3 - q1
503 print('\tinterquartile-range:', iqr)
504
505 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
506 print('\tIdeal bin-width: h =', ideal_h)
507
508 #####
509 # Q1.ak
510 #####
511 print('\nQ1.ak')
512
513 # Print Interquartile-range
514 q1 = data.quantile(0.25)['x']
515 q3 = data.quantile(0.75)['x']
516 iqr = q3 - q1
517 print('\tinterquartile-range:', iqr)
518
519 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
520 print('\tIdeal bin-width: h =', ideal_h)
521
522 #####
523 # Q1.al
524 #####
525 print('\nQ1.al')
526
527 # Print Interquartile-range
528 q1 = data.quantile(0.25)['x']
529 q3 = data.quantile(0.75)['x']
530 iqr = q3 - q1
531 print('\tinterquartile-range:', iqr)
532
533 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
534 print('\tIdeal bin-width: h =', ideal_h)
535
536 #####
537 # Q1.am
538 #####
539 print('\nQ1.am')
540
541 # Print Interquartile-range
542 q1 = data.quantile(0.25)['x']
543 q3 = data.quantile(0.75)['x']
544 iqr = q3 - q1
545 print('\tinterquartile-range:', iqr)
546
547 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
548 print('\tIdeal bin-width: h =', ideal_h)
549
550 #####
551 # Q1.an
552 #####
553 print('\nQ1.an')
554
555 # Print Interquartile-range
556 q1 = data.quantile(0.25)['x']
557 q3 = data.quantile(0.75)['x']
558 iqr = q3 - q1
559 print('\tinterquartile-range:', iqr)
560
561 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
562 print('\tIdeal bin-width: h =', ideal_h)
563
564 #####
565 # Q1.ao
566 #####
567 print('\nQ1.ao')
568
569 # Print Interquartile-range
570 q1 = data.quantile(0.25)['x']
571 q3 = data.quantile(0.75)['x']
572 iqr = q3 - q1
573 print('\tinterquartile-range:', iqr)
574
575 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
576 print('\tIdeal bin-width: h =', ideal_h)
577
578 #####
579 # Q1.ap
580 #####
581 print('\nQ1.ap')
582
583 # Print Interquartile-range
584 q1 = data.quantile(0.25)['x']
585 q3 = data.quantile(0.75)['x']
586 iqr = q3 - q1
587 print('\tinterquartile-range:', iqr)
588
589 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
590 print('\tIdeal bin-width: h =', ideal_h)
591
592 #####
593 # Q1.aq
594 #####
595 print('\nQ1.aq')
596
597 # Print Interquartile-range
598 q1 = data.quantile(0.25)['x']
599 q3 = data.quantile(0.75)['x']
600 iqr = q3 - q1
601 print('\tinterquartile-range:', iqr)
602
603 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
604 print('\tIdeal bin-width: h =', ideal_h)
605
606 #####
607 # Q1.ar
608 #####
609 print('\nQ1.ar')
610
611 # Print Interquartile-range
612 q1 = data.quantile(0.25)['x']
613 q3 = data.quantile(0.75)['x']
614 iqr = q3 - q1
615 print('\tinterquartile-range:', iqr)
616
617 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
618 print('\tIdeal bin-width: h =', ideal_h)
619
620 #####
621 # Q1.as
622 #####
623 print('\nQ1.as')
624
625 # Print Interquartile-range
626 q1 = data.quantile(0.25)['x']
627 q3 = data.quantile(0.75)['x']
628 iqr = q3 - q1
629 print('\tinterquartile-range:', iqr)
630
631 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
632 print('\tIdeal bin-width: h =', ideal_h)
633
634 #####
635 # Q1.at
636 #####
637 print('\nQ1.at')
638
639 # Print Interquartile-range
640 q1 = data.quantile(0.25)['x']
641 q3 = data.quantile(0.75)['x']
642 iqr = q3 - q1
643 print('\tinterquartile-range:', iqr)
644
645 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
646 print('\tIdeal bin-width: h =', ideal_h)
647
648 #####
649 # Q1.au
650 #####
651 print('\nQ1.au')
652
653 # Print Interquartile-range
654 q1 = data.quantile(0.25)['x']
655 q3 = data.quantile(0.75)['x']
656 iqr = q3 - q1
657 print('\tinterquartile-range:', iqr)
658
659 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
660 print('\tIdeal bin-width: h =', ideal_h)
661
662 #####
663 # Q1.av
664 #####
665 print('\nQ1.av')
666
667 # Print Interquartile-range
668 q1 = data.quantile(0.25)['x']
669 q3 = data.quantile(0.75)['x']
670 iqr = q3 - q1
671 print('\tinterquartile-range:', iqr)
672
673 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
674 print('\tIdeal bin-width: h =', ideal_h)
675
676 #####
677 # Q1.aw
678 #####
679 print('\nQ1.aw')
680
681 # Print Interquartile-range
682 q1 = data.quantile(0.25)['x']
683 q3 = data.quantile(0.75)['x']
684 iqr = q3 - q1
685 print('\tinterquartile-range:', iqr)
686
687 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
688 print('\tIdeal bin-width: h =', ideal_h)
689
690 #####
691 # Q1.ax
692 #####
693 print('\nQ1.ax')
694
695 # Print Interquartile-range
696 q1 = data.quantile(0.25)['x']
697 q3 = data.quantile(0.75)['x']
698 iqr = q3 - q1
699 print('\tinterquartile-range:', iqr)
700
701 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
702 print('\tIdeal bin-width: h =', ideal_h)
703
704 #####
705 # Q1.ay
706 #####
707 print('\nQ1.ay')
708
709 # Print Interquartile-range
710 q1 = data.quantile(0.25)['x']
711 q3 = data.quantile(0.75)['x']
712 iqr = q3 - q1
713 print('\tinterquartile-range:', iqr)
714
715 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
716 print('\tIdeal bin-width: h =', ideal_h)
717
718 #####
719 # Q1.az
720 #####
721 print('\nQ1.az')
722
723 # Print Interquartile-range
724 q1 = data.quantile(0.25)['x']
725 q3 = data.quantile(0.75)['x']
726 iqr = q3 - q1
727 print('\tinterquartile-range:', iqr)
728
729 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
730 print('\tIdeal bin-width: h =', ideal_h)
731
732 #####
733 # Q1.ba
734 #####
735 print('\nQ1.ba')
736
737 # Print Interquartile-range
738 q1 = data.quantile(0.25)['x']
739 q3 = data.quantile(0.75)['x']
740 iqr = q3 - q1
741 print('\tinterquartile-range:', iqr)
742
743 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
744 print('\tIdeal bin-width: h =', ideal_h)
745
746 #####
747 # Q1.bb
748 #####
749 print('\nQ1.bb')
750
751 # Print Interquartile-range
752 q1 = data.quantile(0.25)['x']
753 q3 = data.quantile(0.75)['x']
754 iqr = q3 - q1
755 print('\tinterquartile-range:', iqr)
756
757 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
758 print('\tIdeal bin-width: h =', ideal_h)
759
760 #####
761 # Q1.bc
762 #####
763 print('\nQ1.bc')
764
765 # Print Interquartile-range
766 q1 = data.quantile(0.25)['x']
767 q3 = data.quantile(0.75)['x']
768 iqr = q3 - q1
769 print('\tinterquartile-range:', iqr)
770
771 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
772 print('\tIdeal bin-width: h =', ideal_h)
773
774 #####
775 # Q1.bd
776 #####
777 print('\nQ1.bd')
778
779 # Print Interquartile-range
780 q1 = data.quantile(0.25)['x']
781 q3 = data.quantile(0.75)['x']
782 iqr = q3 - q1
783 print('\tinterquartile-range:', iqr)
784
785 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
786 print('\tIdeal bin-width: h =', ideal_h)
787
788 #####
789 # Q1.be
790 #####
791 print('\nQ1.be')
792
793 # Print Interquartile-range
794 q1 = data.quantile(0.25)['x']
795 q3 = data.quantile(0.75)['x']
796 iqr = q3 - q1
797 print('\tinterquartile-range:', iqr)
798
799 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
800 print('\tIdeal bin-width: h =', ideal_h)
801
802 #####
803 # Q1 bf
804 #####
805 print('\nQ1.bf')
806
807 # Print Interquartile-range
808 q1 = data.quantile(0.25)['x']
809 q3 = data.quantile(0.75)['x']
810 iqr = q3 - q1
811 print('\tinterquartile-range:', iqr)
812
813 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
814 print('\tIdeal bin-width: h =', ideal_h)
815
816 #####
817 # Q1.bg
818 #####
819 print('\nQ1.bg')
820
821 # Print Interquartile-range
822 q1 = data.quantile(0.25)['x']
823 q3 = data.quantile(0.75)['x']
824 iqr = q3 - q1
825 print('\tinterquartile-range:', iqr)
826
827 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
828 print('\tIdeal bin-width: h =', ideal_h)
829
830 #####
831 # Q1.bh
832 #####
833 print('\nQ1.bh')
834
835 # Print Interquartile-range
836 q1 = data.quantile(0.25)['x']
837 q3 = data.quantile(0.75)['x']
838 iqr = q3 - q1
839 print('\tinterquartile-range:', iqr)
840
841 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
842 print('\tIdeal bin-width: h =', ideal_h)
843
844 #####
845 # Q1.bi
846 #####
847 print('\nQ1.bi')
848
849 # Print Interquartile-range
850 q1 = data.quantile(0.25)['x']
851 q3 = data.quantile(0.75)['x']
852 iqr = q3 - q1
853 print('\tinterquartile-range:', iqr)
854
855 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
856 print('\tIdeal bin-width: h =', ideal_h)
857
858 #####
859 # Q1.bj
860 #####
861 print('\nQ1.bj')
862
863 # Print Interquartile-range
864 q1 = data.quantile(0.25)['x']
865 q3 = data.quantile(0.75)['x']
866 iqr = q3 - q1
867 print('\tinterquartile-range:', iqr)
868
869 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
870 print('\tIdeal bin-width: h =', ideal_h)
871
872 #####
873 # Q1.bk
874 #####
875 print('\nQ1.bk')
876
877 # Print Interquartile-range
878 q1 = data.quantile(0.25)['x']
879 q3 = data.quantile(0.75)['x']
880 iqr = q3 - q1
881 print('\tinterquartile-range:', iqr)
882
883 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
884 print('\tIdeal bin-width: h =', ideal_h)
885
886 #####
887 # Q1.bl
888 #####
889 print('\nQ1.bl')
890
891 # Print Interquartile-range
892 q1 = data.quantile(0.25)['x']
893 q3 = data.quantile(0.75)['x']
894 iqr = q3 - q1
895 print('\tinterquartile-range:', iqr)
896
897 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
898 print('\tIdeal bin-width: h =', ideal_h)
899
900 #####
901 # Q1 bm
902 #####
903 print('\nQ1.bm')
904
905 # Print Interquartile-range
906 q1 = data.quantile(0.25)['x']
907 q3 = data.quantile(0.75)['x']
908 iqr = q3 - q1
909 print('\tinterquartile-range:', iqr)
910
911 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
912 print('\tIdeal bin-width: h =', ideal_h)
913
914 #####
915 # Q1.bn
916 #####
917 print('\nQ1.bn')
918
919 # Print Interquartile-range
920 q1 = data.quantile(0.25)['x']
921 q3 = data.quantile(0.75)['x']
922 iqr = q3 - q1
923 print('\tinterquartile-range:', iqr)
924
925 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
926 print('\tIdeal bin-width: h =', ideal_h)
927
928 #####
929 # Q1.bo
930 #####
931 print('\nQ1.bo')
932
933 # Print Interquartile-range
934 q1 = data.quantile(0.25)['x']
935 q3 = data.quantile(0.75)['x']
936 iqr = q3 - q1
937 print('\tinterquartile-range:', iqr)
938
939 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
940 print('\tIdeal bin-width: h =', ideal_h)
941
942 #####
943 # Q1.bp
944 #####
945 print('\nQ1.bp')
946
947 # Print Interquartile-range
948 q1 = data.quantile(0.25)['x']
949 q3 = data.quantile(0.75)['x']
950 iqr = q3 - q1
951 print('\tinterquartile-range:', iqr)
952
953 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
954 print('\tIdeal bin-width: h =', ideal_h)
955
956 #####
957 # Q1.bq
958 #####
959 print('\nQ1.bq')
960
961 # Print Interquartile-range
962 q1 = data.quantile(0.25)['x']
963 q3 = data.quantile(0.75)['x']
964 iqr = q3 - q1
965 print('\tinterquartile-range:', iqr)
966
967 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
968 print('\tIdeal bin-width: h =', ideal_h)
969
970 #####
971 # Q1.br
972 #####
973 print('\nQ1.br')
974
975 # Print Interquartile-range
976 q1 = data.quantile(0.25)['x']
977 q3 = data.quantile(0.75)['x']
978 iqr = q3 - q1
979 print('\tinterquartile-range:', iqr)
980
981 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
982 print('\tIdeal bin-width: h =', ideal_h)
983
984 #####
985 # Q1.bs
986 #####
987 print('\nQ1.bs')
988
989 # Print Interquartile-range
990 q1 = data.quantile(0.25)['x']
991 q3 = data.quantile(0.75)['x']
992 iqr = q3 - q1
993 print('\tinterquartile-range:', iqr)
994
995 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
996 print('\tIdeal bin-width: h =', ideal_h)
997
998 #####
999 # Q1.bt
1000 #####
1001 print('\nQ1.bt')
1002
1003 # Print Interquartile-range
1004 q1 = data.quantile(0.25)['x']
1005 q3 = data.quantile(0.75)['x']
1006 iqr = q3 - q1
1007 print('\tinterquartile-range:', iqr)
1008
1009 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
1010 print('\tIdeal bin-width: h =', ideal_h)
1011
1012 #####
1013 # Q1.bu
1014 #####
1015 print('\nQ1.bu')
1016
1017 # Print Interquartile-range
1018 q1 = data.quantile(0.25)['x']
1019 q3 = data.quantile(0.75)['x']
1020 iqr = q3 - q1
1021 print('\tinterquartile-range:', iqr)
1022
1023 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
1024 print('\tIdeal bin-width: h =', ideal_h)
1025
1026 #####
1027 # Q1.bv
1028 #####
1029 print('\nQ1.bv')
1030
1031 # Print Interquartile-range
1032 q1 = data.quantile(0.25)['x']
1033 q3 = data.quantile(0.75)['x']
1034 iqr = q3 - q1
1035 print('\tinterquartile-range:', iqr)
1036
1037 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
1038 print('\tIdeal bin-width: h =', ideal_h)
1039
1040 #####
1041 # Q1.bw
1042 #####
1043 print('\nQ1.bw')
1044
1045 # Print Interquartile-range
1046 q1 = data.quantile(0.25)['x']
1047 q3 = data.quantile(0.75)['x']
1048 iqr = q3 - q1
1049 print('\tinterquartile-range:', iqr)
1050
1051 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
1052 print('\tIdeal bin-width: h =', ideal_h)
1053
1054 #####
1055 # Q1.bx
1056 #####
1057 print('\nQ1.bx')
1058
1059 # Print Interquartile-range
1060 q1 = data.quantile(0.25)['x']
1061 q3 = data.quantile(0.75)['x']
1062 iqr = q3 - q1
1063 print('\tinterquartile-range:', iqr)
1064
1065 ideal h = 2 * iqr * len(xs) ** (-1 / 3)
1066 print('\tIdeal bin-width: h =', ideal_h)
1067
1068 #####
1069 # Q1.by
1070 #####
10
```

- The five number summary is as shown in the terminal above, the 1.5 iqr is also shown as 1.5 times the difference between q3 and q1 as shown in the terminal above
 - Note that there are only two unique groups for the dataset specified. See the terminal above for outliers, see the 'Figure 1' window above for the box-plot
3. See attached q3.py for full python program
- 19.9497% of investigations were found to be fraudulent



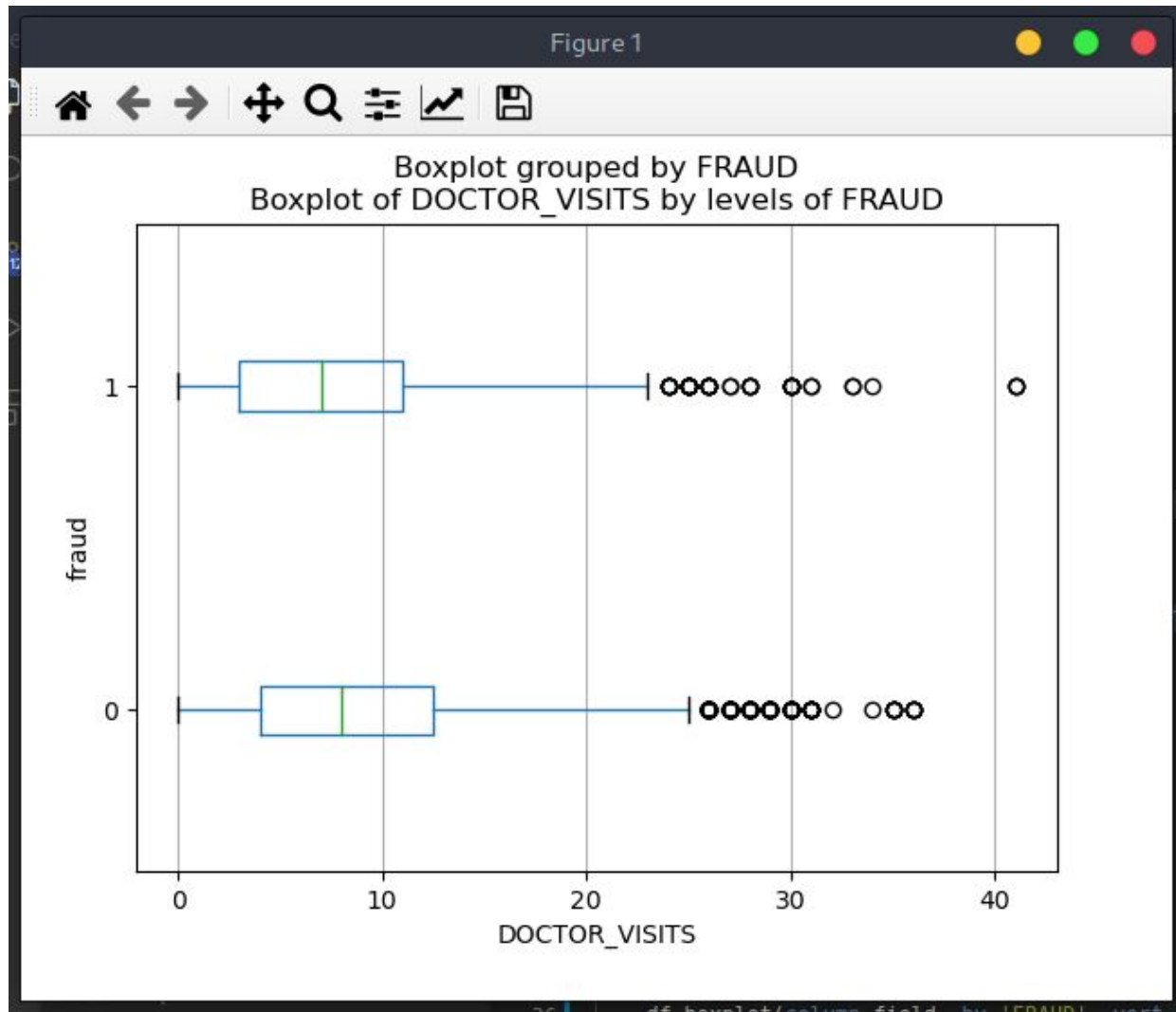
b. Code used to generate plots:

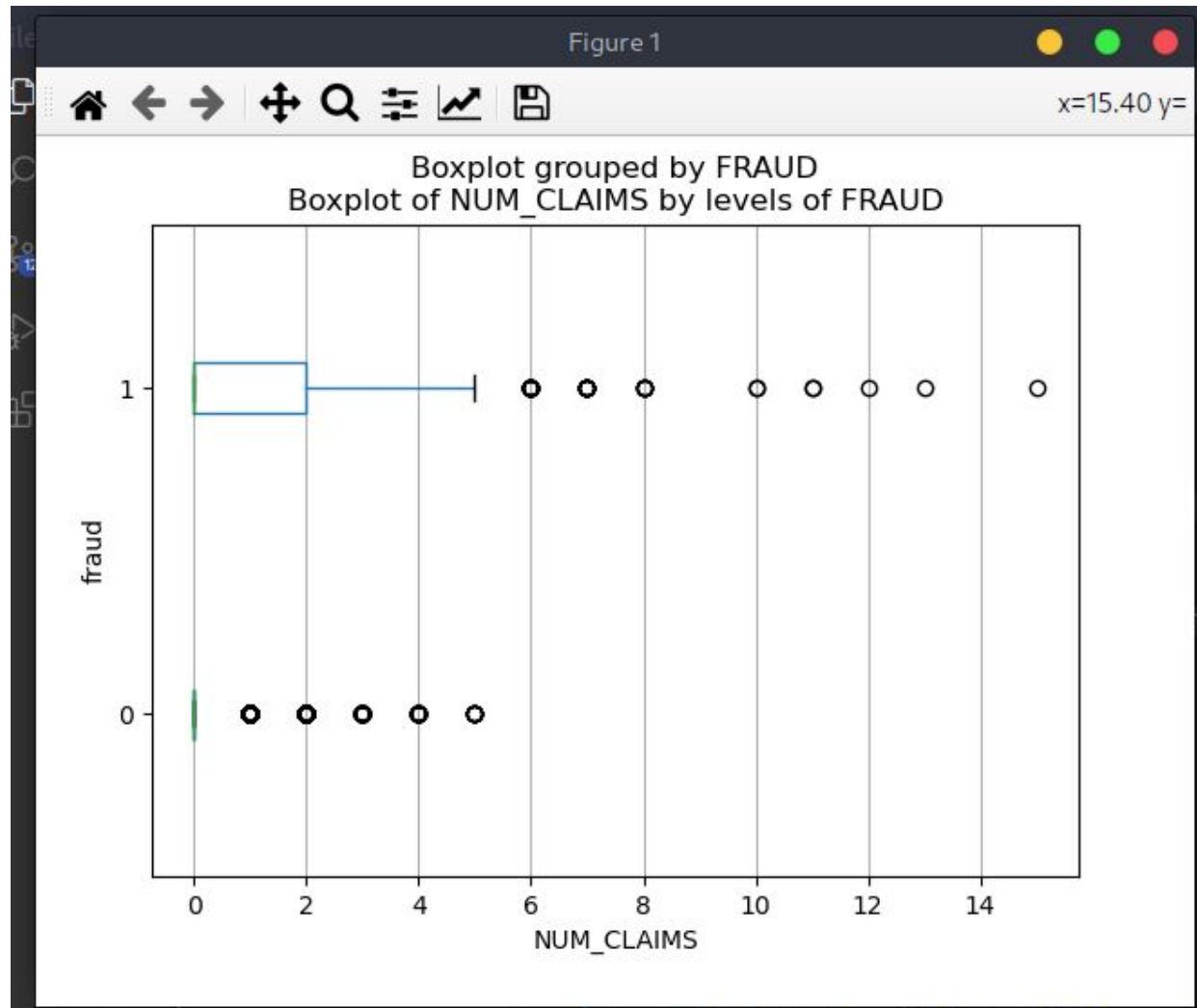
```
22
23 #####
24 # Q3.b
25 #####
26 print('Q3.b')
27
28 # Box plot for each field
29 for field in df:
30     # Skip case_id lol
31     if field in ('CASE_ID', 'FRAUD'):
32         continue
33
34     # Draw boxplot
35     df.boxplot(column=field, by='FRAUD', vert=False, whis=1.5, sym='')
36     plt.title("Boxplot of %s in relation to FRAUD" % field)
37     plt.xlabel(field)
38     plt.ylabel('fraud')
39     plt.grid(axis="y")
40     plt.show()
41
```

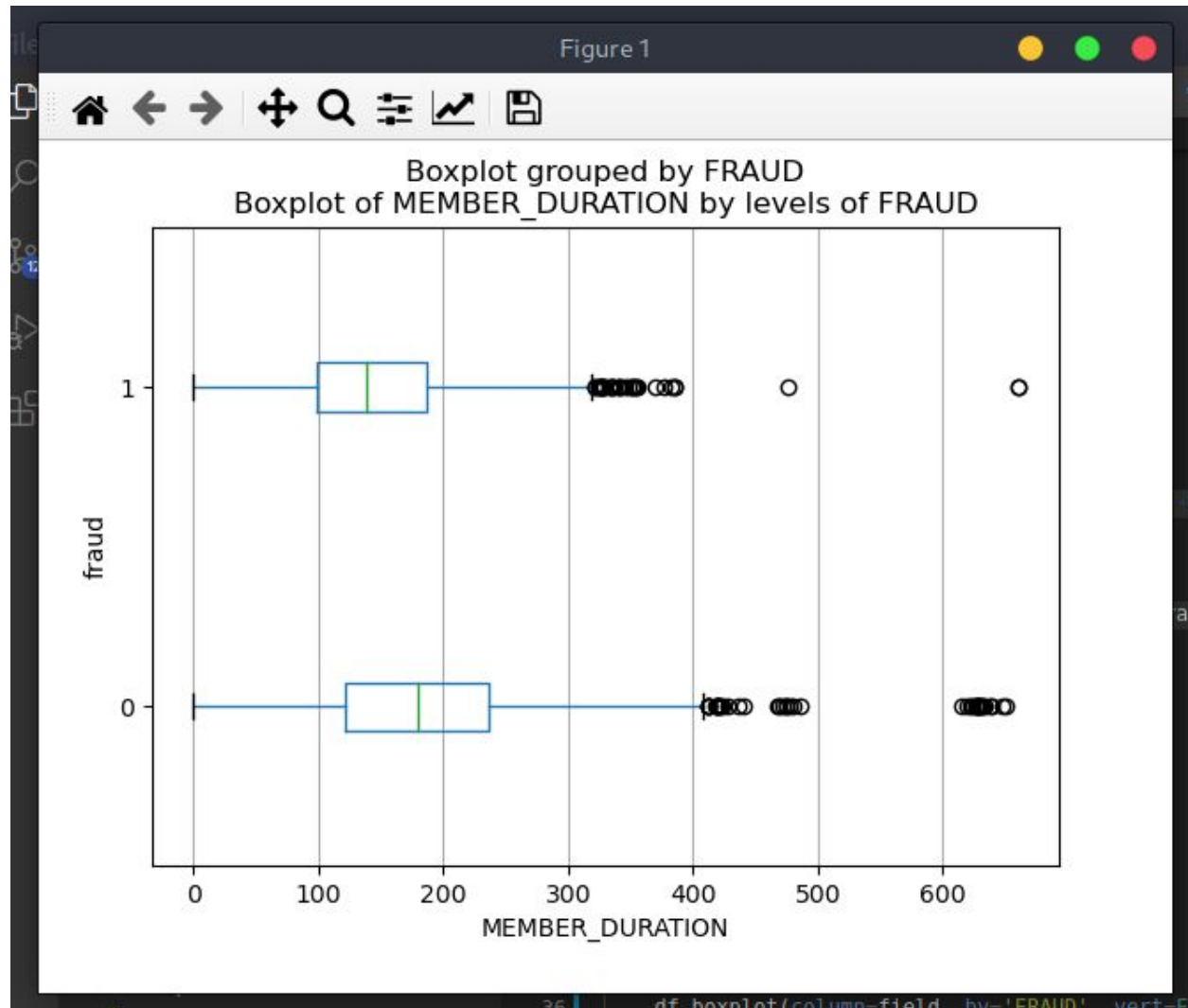
Plots:

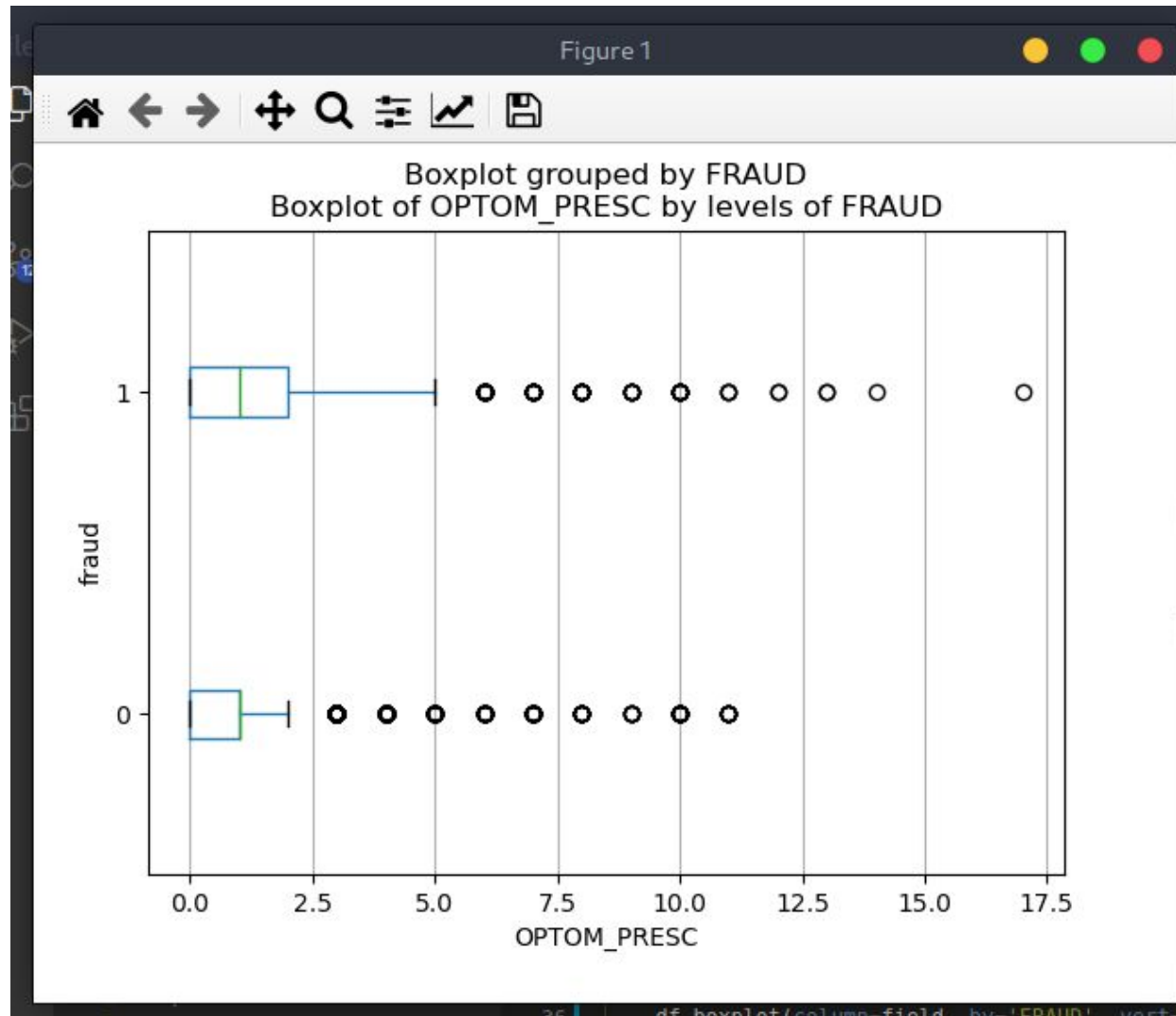


```
36 | df.boxplot(column='field', by='FRAUD', vert=False)
```

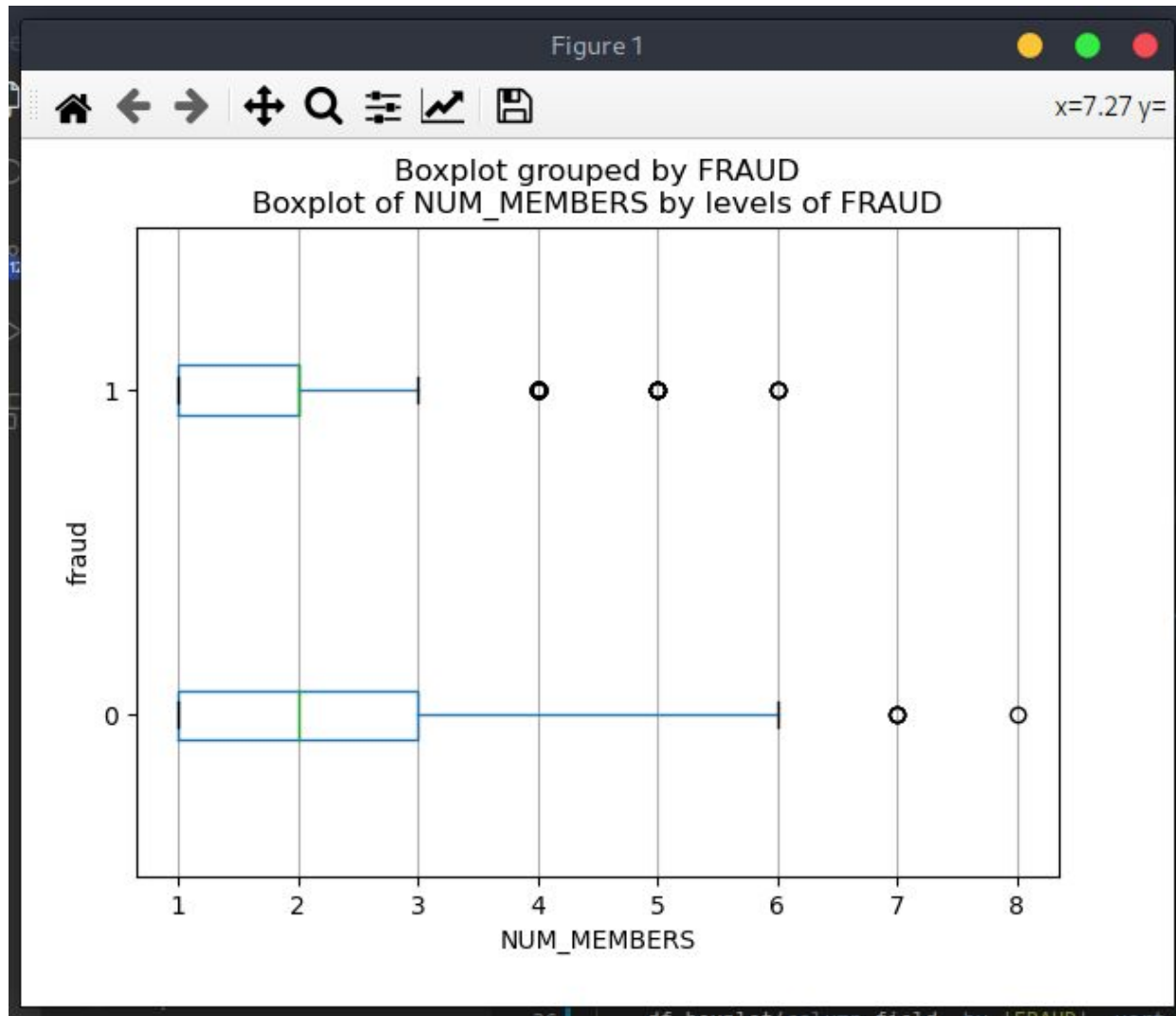








```
36 | df.boxplot(column='field', by='FRAUD', vert=
```



- C.
- i. There were some math concepts that I haven't yet covered in any of my classes (I'm just now taking linear algebra), however I did find the eigenvectors and eigenvalues using the same methods as used in the examples done in class. As you can see from the eigenvalues, all of them are above 1 so they should all be usable according to the criteria set forth in the assignment. (see 'usable dimensions' in screenshot)

```
38 # plt.xlabel(field)
39 # plt.ylabel('fraud')
40 # plt.grid(axis='y')
41 # plt.show()
42
43 #####
44 # Q3.c
45 #####
46 print('\nQ3.c')
47
48 # Orthonormalization
49
50 # get interval variable fields
51 fields = [field for field in df if field not in ('CASE_ID', 'FRAUD')]
52
53 # transpose
54 # mattx = mat.transpose() * mat
55 data = df.set_index('CASE_ID')
56 mat = numpy.matrix(df.values)
57 mattx = mat.transpose() * mat
58 print('t(data) * data:\n', mattx)
59
60 # Eigenvalue decomposition
61 evals, evecs = numpy.linalg.eigh(mattx)
62 usable_field_pairs = list(filter(lambda p: p[1] > 1, zip(fields, evals)))
63 usable_fields = list(map(lambda p: p[0], usable_field_pairs))
64 print("eigenvalues:\n", evals)
65 print('usable dimensions:')
66 for f, ev in usable_field_pairs:
67     print('\t', f, ': ', ev)
68
69 # Here is the transformation matrix
70 transf = evecs * numpy.linalg.inv(numpy.sqrt(numpy.diagflat(evals)))
71 print("Transformation Matrix = \n", transf)
72
73 # Here is the transformed X
74 transf_df = mat * transf
75 print("The Transformed x = \n", transf_df)
76
77 # Check columns of transformed X
78 print("Expect an Identity Matrix = \n", numpy.round(transf_df.transpose(), 10))
79
80 #####
81 # Q3.d
82 #####
83 print('\nQ3.d')
```

```
2121127      13581      29423]]
eigenvalues:
[8.08503761e+02 6.95330743e+03 8.39876804e+03 1.80629816e+04
 3.15629087e+05 7.86083187e+07 4.45764449e+09 2.87846878e+11]
usable dimensions:
TOTAL SPEND : 808.5037614522325
DOCTOR VISITS : 6953.307429707188
NUM CLAIMS : 8398.7680352244
MEMBER DURATION : 18062.98163764361
OPTOM PRESC : 315629.08739040286
NUM MEMBERS : 78608318.7360505
Transformation Matrix =
[[-7.45819084e-07 3.88904761e-07 6.01031836e-07 1.0442
-4.00557476e-07 4.07326752e-06 -1.47945552e-05 8.94112
-3.47131561e-02 1.57761749e-03 -6.44914431e-04 -5.22827
6.56579529e-06 -4.78204437e-08 -4.53125156e-11 4.16875
1.29133203e-07 2.22494408e-08 1.55793948e-07 1.25837
-1.86815711e-07 1.94660186e-07 2.27446499e-06 5.82575
1.86893445e-05 -6.26666958e-05 2.96172133e-04 9.34454
1.77823869e-03 -3.59325309e-06 -1.99664674e-08 2.15680
5.19634597e-03 1.15636121e-02 -2.07850562e-03 -8.14195
2.04058545e-05 -1.82663595e-07 -8.56584050e-10 8.89834
-6.42734920e-06 8.82918543e-08 5.25675438e-05 2.13868
-5.70803127e-05 -1.12655282e-04 -5.30323977e-07 4.23225
1.62916254e-03 -9.86865990e-04 2.36019519e-03 -7.23471
1.04459047e-05 -2.19383825e-07 -2.91639966e-09 2.79478
1.48082665e-03 -2.60809373e-03 -1.04243934e-02 -1.44056
4.81246437e-05 -6.80234847e-07 -4.54727906e-09 4.56273
The Transformed x =
[[-0.03037963 -0.00519161 -0.01076229 ... -0.01041255 0.
0.00064132]
[-0.02332657 0.02169855 -0.00653614 ... -0.01350899 0.0
0.00075805]
[-0.02933297 -0.0073465 -0.02030515 ... -0.01649811 0.0
0.00087476]
...
[0.00895449 -0.00183573 0.01221588 ... 0.01769404 0.1
0.05249938]
[0.00899968 -0.00175914 0.01211892 ... 0.01759319 0.1
0.05284902]
[0.00751114 0.00078943 0.02317107 ... 0.01693788 0.1
0.05290739]]
Expect an Identity Matrix =
[[ 1. -0. 0. 0. -0. 0. -0. 0.]
[-0. 1. -0. -0. 0. -0. 0. 0.]
[ 0. -0. 1. 0. -0. 0. -0. 0.]
[ 0. -0. 0. 1. -0. 0. -0. -0.]
[-0. 0. -0. -0. 1. -0. 0. 0.]
[ 0. -0. 0. 0. -0. 1. -0. 0.]
[-0. 0. -0. -0. 0. 0. -1. 0.]
[ 0. 0. 0. -0. 0. 0. 0. 1.]
```

ii. By multiplying the orthonormalized matrix by the original transpose I got an identity matrix as expected. I'm not sure if this is the right test, I tried a few other methods for normalizing the data but the checks for them were less clear (and probably wrong), so I decided to use the method from the book.

d.

- i. As you can see in the image below the score function returned 0.8395973154362416
- ii. This means that when I ran the classifier on our training data it was able to successfully correctly predict approximately 83.96% of them. However, because only approximately 19.9497% of the input data was fraudulent, our classifier is only about 3.91% better than always predicting fraud=0. However from the perspective of a company with millions of customers

this could be worth it.

```

89 | |
90 | #####
91 | # Q3.d
92 | #####
93 | print('\nQ3.d')
94 |
95 | from sklearn.neighbors import KNeighborsClassifier
96 |
97 | # Perform classification
98 | trainData = data[usable_fields]
99 | target = data['FRAUD']
100 | neigh = KNeighborsClassifier(n_neighbors=4, algorithm='brute', metric='euclidean')
101 | nhrs = neigh.fit(trainData, target)
102 |
103 | # See the classification probabilities
104 | class_prob = nhrs.predict_proba(trainData)
105 | print('classification probabilities:\n', class_prob)
106 |
107 | # Test performance with our training dataset
108 | test_data = data[[field for field in data if field not in ('FRAUD')]]
109 | print('score:', neigh.score(test_data, target))
110 | print('This is the ratio of correct predictions over total number of tests for our training data')
111 | #####

```

```

17 | target = data['FRAUD']
18 | neigh = KNeighborsClassifier(n_neighbors=4, algorithm='brute', metric='euclidean')
19 | nhrs = neigh.fit(trainData, target)
20 |
21 | # See the classification probabilities
22 | class_prob = nhrs.predict_proba(trainData)
23 | print('classification probabilities:\n', class_prob)
24 |
25 | # Test performance with our training dataset
26 | test_data = data[[field for field in data if field not in ('FRAUD')]]
27 | print('score:', neigh.score(test_data, target))
28 | print('This is the ratio of correct predictions over total number of tests for our training data')
29 | #####

```

```

Terminal - tate@archbook:~/Desktop/cs484/assignment1
[ 4.44979074e-04 -5.00985828e-01 8.12752890e+01 -1.31826801e-02
-5.71740902e+00 -1.44453484e+01]]
Q3.d
classification probabilities:
[[0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
score: 0.8395973154362416
This is the ratio of correct predictions over total number of tests for our t

```

e. The output of kneighbors is as follows:

- [0, 11.61895004, 12.489996 , 13.52774926, 14.07124728]
- [588, 577, 582, 573, 575]

```

111 | #####
112 | # Q3.e
113 | #####
114 | print('\nQ3.e')
115 |
116 | # Find nearest neighbor
117 | investigation = [[
118 |     7500,
119 |     15,
120 |     3,
121 |     127,
122 |     2,
123 |     2,
124 | ]]
125 |
126 | print('kneighbors:\n', nhrs.kneighbors(investigation, n_neighbors=5))
127 | #####
128 |

```

```

17 | target = data['FRAUD']
18 | neigh = KNeighborsClassifier(n_neighbors=4, algorithm='brute', metric='euclidean')
19 | nhrs = neigh.fit(trainData, target)
20 |
21 | # See the classification probabilities
22 | class_prob = nhrs.predict_proba(trainData)
23 | print('classification probabilities:\n', class_prob)
24 |
25 | # Test performance with our training dataset
26 | test_data = data[[field for field in data if field not in ('FRAUD')]]
27 | print('score:', neigh.score(test_data, target))
28 | print('This is the ratio of correct predictions over total number of tests for our training data')
29 | #####

```

```

Terminal - tate@archbook:~/Desktop/cs484/assignment1
[ 4.44979074e-04 -5.00985828e-01 8.12752890e+01 -1.31826801e-02
-5.71740902e+00 -1.44453484e+01]]
Q3.d
classification probabilities:
[[0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
score: 0.8395973154362416
This is the ratio of correct predictions over total number of tests for our training data
Q3.e
kneighbors:
(array([[ 0.         , 11.61895004, 12.489996 , 13.52774926, 14.07124728]]), array([[588, 577, 582, 573, 575]]))
Q3.f

```

f. The knn classifier gave a 25% probability that the case is non-fraudulent and a 75% chance that the case is a fraud. This will result in the case being classified as a fraud as the probability is greater than the average of 20%. Because it was not given whether or not this case is actually fraudulent or not we cannot determine if it's been misclassified until an investigation has been done

```

127 | print('kneighbors:\n', nhrs.kneighbors(investigation, n_neighbors=5))
128 | #####
129 | # Q3.f
130 | #####
131 | print('\nQ3.f')
132 |
133 | # Make prediction
134 | print('prediction:', nhrs.predict(investigation))
135 | print('prediction probability:', nhrs.predict_proba(investigation))
136 |

```

```

17 | target = data['FRAUD']
18 | neigh = KNeighborsClassifier(n_neighbors=4, algorithm='brute', metric='euclidean')
19 | nhrs = neigh.fit(trainData, target)
20 |
21 | # See the classification probabilities
22 | class_prob = nhrs.predict_proba(trainData)
23 | print('classification probabilities:\n', class_prob)
24 |
25 | # Test performance with our training dataset
26 | test_data = data[[field for field in data if field not in ('FRAUD')]]
27 | print('score:', neigh.score(test_data, target))
28 | print('This is the ratio of correct predictions over total number of tests for our training data')
29 | #####

```

```

Terminal - tate@archbook:~/Desktop/cs484/assignment1
[ 4.44979074e-04 -5.00985828e-01 8.12752890e+01 -1.31826801e-02
-5.71740902e+00 -1.44453484e+01]]
Q3.d
classification probabilities:
[[0. 1.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
score: 0.8395973154362416
This is the ratio of correct predictions over total number of tests for our training data
Q3.e
kneighbors:
(array([[ 0.         , 11.61895004, 12.489996 , 13.52774926, 14.07124728]]), array([[588, 577, 582, 573, 575]]))
Q3.f
prediction: [1]
prediction probability: [[0.25 0.75]]
[tate@archbook assignment1]$

```


4. I'm pretty confident I'm doing this wrong... Book doesn't do a good job of clarifying how to find the error-rate...

```
1
2
3 import pandas
4 import numpy
5 import sklearn.neighbors
6
7 # dataset: CASE_ID,x1,x2,y
8 X = numpy.matrix([
9     [1, 7.7, -37, 4],
10    [2, 9.5, -38, 1],
11    [3, 3.0, -34, 1],
12    [4, 9.1, -75, 1],
13    [5, 2.2, -31, 2],
14    [6, 4.8, -7, 4],
15    [7, 5.5, -6, 3],
16    [8, 10, -61, 1],
17    [9, 4.2, -23, 2],
18    [10, 1.6, -54, 1],
19 ])
20
21 for m in ('euclidean', 'manhattan', 'chebyshev'):
22     print('Scores for', m)
23     for i in range(1, 10):
24         nbrs = sklearn.neighbors.NearestNeighbors(
25             n_neighbors=i, algorithm='brute', metric=m).fit(X)
26         distances, indices = nbrs.kneighbors(X)
27         print('\twhen neighbors =', i,
28               'distance = ', sum(map(sum, distances)))
```

```
[tate@archbook assignment1]$ python q4.py
Scores for euclidean
  when neighbors = 1 distance = 0.0
  when neighbors = 2 distance = 64.92061316677129
  when neighbors = 3 distance = 198.53353326109823
  when neighbors = 4 distance = 382.60595253929273
  when neighbors = 5 distance = 597.1177596508309
  when neighbors = 6 distance = 854.1840532765247
  when neighbors = 7 distance = 1136.017541804338
  when neighbors = 8 distance = 1500.8658555051784
  when neighbors = 9 distance = 1942.8909385537931
Scores for manhattan
  when neighbors = 1 distance = 0.0
  when neighbors = 2 distance = 102.30000000000001
  when neighbors = 3 distance = 295.90000000000003
  when neighbors = 4 distance = 542.6
  when neighbors = 5 distance = 842.4000000000001
  when neighbors = 6 distance = 1192.1999999999998
  when neighbors = 7 distance = 1568.6
  when neighbors = 8 distance = 2007.7
  when neighbors = 9 distance = 2540.6
Scores for chebyshev
  when neighbors = 1 distance = 0.0
  when neighbors = 2 distance = 52.8
  when neighbors = 3 distance = 169.70000000000002
  when neighbors = 4 distance = 336.79999999999995
  when neighbors = 5 distance = 536.8
  when neighbors = 6 distance = 778.8
  when neighbors = 7 distance = 1053.8
  when neighbors = 8 distance = 1413.8000000000002
  when neighbors = 9 distance = 1850.8000000000002
[tate@archbook assignment1]$
```

5. Yes