

## ▼ Permutation codes

```
1 initial_permutation = [2, 6, 3, 1, 4, 8, 5, 7]
2 expansion_permutation = [4, 1, 2, 3, 2, 3, 4, 1]
3 P4 = [2, 4, 3, 1]
4 inv_initial_permutation = [4, 1, 3, 5, 7, 2, 8, 6]
5 P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
6 P8 = [6, 3, 7, 4, 8, 5, 10, 9]
```

```
1 S0 = [[1,0,3,2],
2       [3,2,1,0],
3       [0,2,1,3],
4       [3,1,3,2]]
5
6 S1= [[0,1,2,3],
7       [2,0,1,3],
8       [3,0,1,0],
9       [2,1,0,3]]
```

## ▼ Utility functions

### ▼ Permutation

```
1 def permutation(x, p):
2     res = []
3     for ind in p:
4         res.append(x[ind-1])
5     return res
```

To undo cell deletion use ⌘/Ctrl+M Z or the 'Undo' option in the 'Edit' menu ✕

```
1 def left_shift(x, pos = 1):
2     pos = pos % len(x)
3     return x[pos:] + x[:pos]
```

### ▼ XOR

```
1 def xor(x, y):
2     res = []
3     for i in range(len(x)):
```

```

4     res.append(x[i]^y[i])
5     return res

```

## ▼ sbox

```

1 def sbox_op(x, s):
2     r = int(f'{x[0]}{x[3]}', 2)
3     c = int(f'{x[1]}{x[2]}', 2)
4     val = s[r][c]
5
6     if val == 0: return [0, 0]
7     elif val == 1: return [0, 1]
8     elif val == 2: return [1, 0]
9     else: return [1, 1]

```

## ▼ fk

- split into l and r (4bits each)
- expansion permutation on r (r\_ep = 8 bits)
- xor with key1
- split into r\_ep\_l, e\_ep\_r and s box them
- concatenate into r\_ep\_mod (4bits)
- P4 permutation r\_ep\_mod' (4 bits)
- res = xor l and r\_ep\_mod'
- pass res and r to next stage(swap)

```

1 def f_k(init, key):
2     l = init[:4]
3     r = init[4:]
4     r_ep = permutation(r, expansion_permutation)
5     r_ep_xor = xor(r_ep, key)

```

To undo cell deletion use ⌘/Ctrl+M Z or the 'Undo' option in the 'Edit' menu ✕

```

8     res = xor(l, r_ep_mod2)
9     return res + r

```

## ▼ Key Generation

- Permutation of K10
- Split K10
- Left Shift(1) K10\_1 and K10\_2
- Combine to form K10'

- Key1 = P8 permutation of K10'
- Split K10'
- Left shift(2) K10'\_1 and K10'\_2
- Combine to form K10"
- Key2 = P8 permutation of K10"

```

1 def key_generation(key):
2     K10 = permutation(key, P10)
3     l = K10[:5]
4     r = K10[5:]
5     l = left_shift(l)
6     r = left_shift(r)
7     K10_2 = l+r
8
9     key1 = permutation(K10_2, P8)
10
11     l = left_shift(l, 2)
12     r = left_shift(r, 2)
13     K10_3 = l+r
14
15     key2 = permutation(K10_3, P8)
16
17     return key1, key2

1 key = [1, 1, 0, 0, 0, 1, 1, 1, 1, 0]

1 key1, key2 = key_generation(key)
2 print("Key1: ", key1)
3 print("Key2: ", key2)
4

Key1:  [1, 1, 1, 0, 1, 0, 0, 1]
Key2:  [1, 0, 1, 0, 0, 1, 1, 1]

```

## ▼ Encryption

To undo cell deletion use ⌘/Ctrl+M Z or the 'Undo' option in the 'Edit' menu ✕

- fk
- swap
- fk
- initial permutation (inverse)

```

1 def encryption(plain_text, key1, key2):
2     IP = permutation(plain_text, initial_permutation)
3     pt_1 = f_k(IP, key1)
4     pt_swap = pt_1[4:] + pt_1[:4]
5     pt_2 = f_k(pt_swap, key2)

```

```
6 IP_inv = permutation(pt_2, inv_initial_permutation)
7
8 return IP_inv
```

```
1 plain_text = [0, 0, 1, 0, 1, 0, 0, 0]
```

```
1 sdes = encryption(plain_text, key1, key2)
```

```
1 print("Output ciphertext: ", sdes)
```

```
Output ciphertext: [1, 0, 0, 0, 1, 0, 1, 0]
```

## ▼ Decryption

```
1 def decryption(plain_text, key1, key2):
2     IP = permutation(plain_text, initial_permutation)
3     pt_1 = f_k(IP, key2)
4     pt_swap = pt_1[4:] + pt_1[:4]
5     pt_2 = f_k(pt_swap, key1)
6     IP_inv = permutation(pt_2, inv_initial_permutation)
7
8     return IP_inv
```

```
1 decryption(sdes, key1, key2)
```

```
[0, 0, 1, 0, 1, 0, 0, 0]
```

To undo cell deletion use ⌘/Ctrl+M Z or the 'Undo' option in the 'Edit' menu ✕



```
1 import numpy as np
```

## ▼ Utility functions

```
1 def nibble_to_hex(n):
2     assert len(n) <= 4, 'Invalid nibble provided.'
3     if len(n) < 4: n = (4-len(n))*'0' + n
4     return hex(int(n, 2))[2:]
```

```
1 def hex_to_nibble(h):
2     assert len(h) == 1, 'Invalid hex digit.'
3     n = bin(int(h,16))[2:]
4     return (4-len(n))*'0' + n
```

```
1 def block_to_state(b):
2     return [
3         [b[0], b[2]],
4         [b[1], b[3]]
5     ]
```

```
1 def state_to_block(s):
2     return [s[0][0], s[1][0], s[0][1], s[1][1]]
```

```
1 def sub_nibbles(s):
2     S = [
3         ['9', '4', 'a', 'b'],
4         ['d', '1', '8', '5'],
5         ['6', '2', '0', '3'],
6         ['c', 'e', 'f', '7']
7     ]
8     b = state_to_block(s)
9     b_new = []
10    for h in b:
11        n = hex_to_nibble(h)
12        n_new = S[int(n[:2], 2)][int(n[2:], 2)]
13        b_new.append(n_new)
14    return block_to_state(b_new)
```

```
1 def shift_rows(s):
2     return [
3         [s[0][0], s[0][1]],
4         [s[1][1], s[1][0]]
5     ]
```

```
1 def mul(x, y):
2     p1 = [int(c) for c in hex_to_nibble(x)]
```

```

3  p2 = [int(c) for c in hex_to_nibble(y)]
4  return np.polymul(p1, p2)
5
6  def add(x, y):
7      p = list(np.polyadd(x, y))
8      p = [c%2 for c in p]
9      _, r = np.polydiv(p, [1, 0, 0, 1, 1])
10     r = [str(int(c%2)) for c in r]
11     return nibble_to_hex(''.join(r))
12
13 def mix_columns(s):
14     C = [
15         ['1', '4'],
16         ['4', '1']
17     ]
18     s_new = [
19         [None, None],
20         [None, None]
21     ]
22     for i in range(2):
23         for j in range(2):
24             s_new[i][j] = add(mul(C[i][0], s[0][j]), mul(C[i][1], s[1][j]))
25     return s_new

```

```

1  def rot_word(w):
2      return [w[1], w[0]]
3
4  def sub_word(w):
5      S = [
6          ['9', '4', 'a', 'b'],
7          ['d', '1', '8', '5'],
8          ['6', '2', '0', '3'],
9          ['c', 'e', 'f', '7']
10     ]
11     w_new = []
12     for h in w:
13         n = hex_to_nibble(h)
14         n_new = S[int(n[:2], 2)][int(n[2:], 2)]
15         w_new.append(n_new)
16     return w_new
17
18 def xor(w1, w2):
19     w = []
20     for i in range(2):
21         x = int(hex_to_nibble(w1[i]), 2)
22         y = int(hex_to_nibble(w2[i]), 2)
23         w.append(nibble_to_hex(bin(x^y)[2:]))
24     return w
25
26 def key_expansion(k):
27     w0, w1 = k[:2], k[2:]
28     r1 = ['8', '0']
29     t2 = xor(sub_word(rot_word(w1)), r1)
30     w2 = xor(w0, t2)

```

```

31 w3 = xor(w1, w2)
32 r2 = ['3', '0']
33 t4 = xor(sub_word(rot_word(w3)), r2)
34 w4 = xor(w2, t4)
35 w5 = xor(w3, w4)
36 return w0 + w1, w2 + w3, w4 + w5

1 def add_round_key(k, s):
2     k_state = block_to_state(k)
3     w1 = xor([k_state[0][0], k_state[1][0]], [s[0][0], s[1][0]])
4     w2 = xor([k_state[0][1], k_state[1][1]], [s[0][1], s[1][1]])
5     return [
6         [w1[0], w2[0]],
7         [w1[1], w2[1]]
8     ]

```

## ▼ Encryption

```

1 def encrypt(plaintext, k):
2     k1, k2, k3 = key_expansion(k)
3     state = block_to_state(plaintext)
4     state = add_round_key(k1, state)
5
6     ### ROUND 1
7     state = sub_nibbles(state)
8     state = shift_rows(state)
9     state = mix_columns(state)
10    state = add_round_key(k2, state)
11
12    ### ROUND 2
13    state = sub_nibbles(state)
14    state = shift_rows(state)
15    #state = [['6', '4'], ['7', 'b']]
16    state = add_round_key(k3, state)
17
18    ciphertext = state_to_block(state)
19    return ciphertext

```

```

1 plaintext = ['1', 'a', '2', '3']
2 key = ['2', '4', '7', '5']

```

```

1 encrypt(plaintext, key)

['d', 'a', '4', '2']

```



---

✓ 0s completed at 19:42

● ✕

```
1 import math

1 class User:
2     def __init__(self, g, p, private_key):
3         self.private_key = private_key
4         self.g = g
5         self.p = p
6         self.public_key = math.pow(self.g, self.private_key) % self.p
7
8     def get_shared_key(self, public_key2):
9         shared_key = math.pow(public_key2, self.private_key) % self.p
10        print("Shared Key = ", shared_key)
11        return shared_key
12
13    def get_public_key(self):
14        return self.public_key
15

1 A = User(17, 23, 13)
2 B = User(17, 23, 9)

1 pk_A = A.get_public_key()
2 pk_B = B.get_public_key()
3 print("public key for A:", pk_A)
4 print("public key for B:", pk_B)

    public key for A: 9.0
    public key for B: 7.0

1 shared key A B = A.get_shared_key(pk_B)

    Shared Key = 20.0
```

---

✓ 0s completed at 15:52

● ✕

```
1 class RSA:
2     def __init__(self, p, q):
3         self.p = p
4         self.q = q
5         self.n = p * q
6         self.e = self.generate_e(p, q)
7         self.d = self.generate_d(p, q)
8         self.public_key = [self.n, self.e]
9         self.private_key = [self.n, self.d]
10
11     def __gcd(self, a, b):
12         if (a == 0 or b == 0): return 0
13         if (a == b): return a
14         if (a > b): return self.__gcd(a - b, b)
15         return self.__gcd(a, b - a)
16
17     def is_coprime(self, x, y):
18         if self.__gcd(x, y) == 1:
19             return 1
20         else:
21             return 0
22
23     def generate_e(self, p, q):
24         x = (p-1) * (q-1)
25         e = 0
26         for i in range(2, x):
27             if self.is_coprime(i, x):
28                 return i
29
30     def generate_d(self, p, q):
31         m = (p-1) * (q-1)
32         for x in range(1, m):
33             if (((self.e % m) * (x % m)) % m == 1):
34                 return x
35
36     def encrypt(self, M):
37         return (M ** self.e) % self.n
38
39     def decrypt(self, C):
40         return (C ** self.d) % self.n
41
42     def show_keys(self):
43         print("Private key:", self.private_key)
44         print("Public key:", self.public_key)
45
```

```
1 rsa = RSA(7, 17)
```

```
1 rsa.show_keys()
```

```
Private key: [119, 77]
Public key: [119, 5]
```

```
1 plain_text = 19
2 cipher_text = rsa.encrypt(19)
3 print("Plain text {} is encrypted as {}".format(plain_text, cipher_text))
4
5 decrypted_text = rsa.decrypt(cipher_text)
6 print("Cipher text {} is decrypted as {}".format(cipher_text, decrypted_text))
```

```
Plain text 19 is encrypted as 66
Cipher text 66 is decrypted as 19
```

---

✓ 0s completed at 16:25



```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import mpl_toolkits
6 from sklearn.linear_model import LinearRegression
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import StandardScaler
9 import datetime as dt
10 %matplotlib inline

```

```

1 data = [[10, 95], [9, 80], [2, 10], [15, 50], [10, 45], [16, 58], [11, 38], [16,
2 df = pd.DataFrame(data, columns = ['Hours', 'Risk'])

```

```

1 x = np.array([10, 9, 2, 15, 10, 16, 11, 16])
2 y = np.array([95, 80, 10, 50, 45, 98, 38, 93])

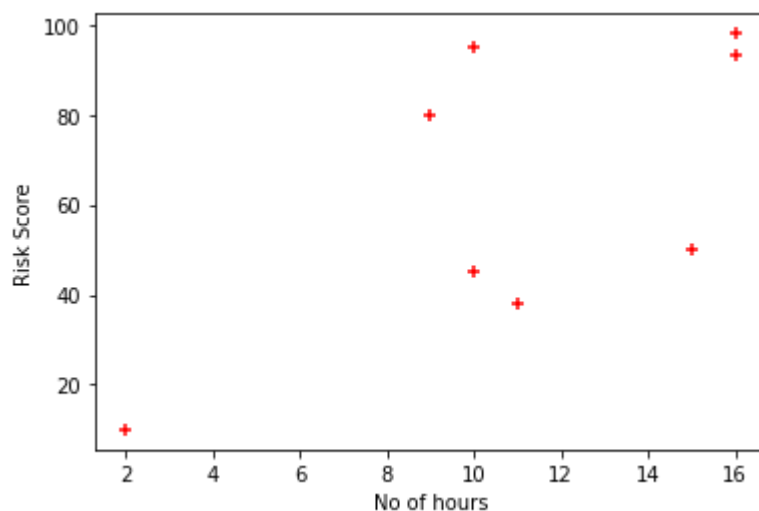
```

```

1 plt.xlabel('No of hours')
2 plt.ylabel('Risk Score')
3 plt.scatter(x,y,color='red',marker='+')

```

<matplotlib.collections.PathCollection at 0x7f445d88e2d0>



```

1 #principle of least squares
2 def getCoef(x,y):
3     mean_x = np.mean(x)
4     mean_y = np.mean(y)
5
6     n = len(x)
7
8     numer = 0
9     denom = 0
10    for i in range(n):
11        numer += (x[i] - mean_x) * (y[i] - mean_y)
12        denom += (x[i] - mean_x) ** 2
13    b1 = numer / denom
14    b0 = mean_y - (b1 * mean_x)

```

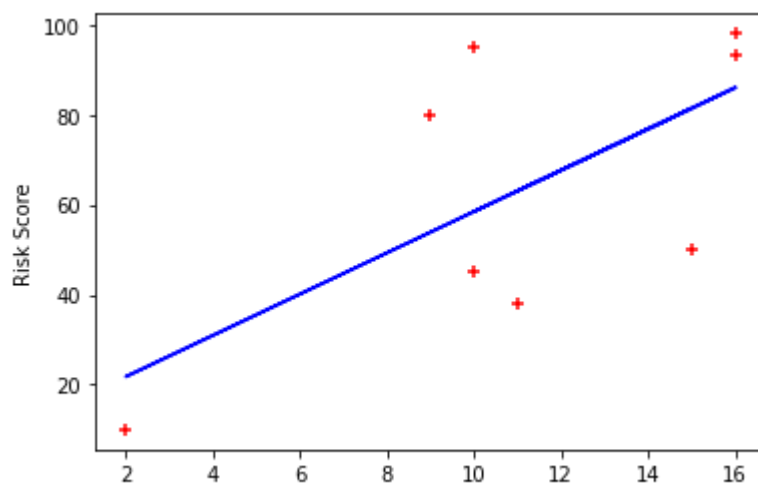
```
15
16     return(b0, b1)
```

```
1 #y = b0 + b1 * x
2 coefs_ = getCoef(x,y)
3 print("Coefficients")
4 print(coefs_[0])
5 print(coefs_[1])
```

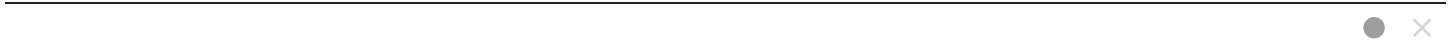
```
Coefficients
12.584627964022893
4.58789860997547
```

```
1 plt.xlabel('No of hours')
2 plt.ylabel('Risk Score')
3 plt.scatter(x,y,color='red',marker='+')
4 y_pred = coefs_[0] + coefs_[1]*x
5 plt.plot(x, y_pred, color = "b")
```

[<matplotlib.lines.Line2D at 0x7f445d3769d0>]



1





```
1 from google.colab import drive
2 drive.mount('/content/gdrive')
```

Mounted at /content/gdrive

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.preprocessing import LabelEncoder
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.tree import export_graphviz
6 from matplotlib import pyplot as plt
```

```
1 data = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/LP3/ML/ML_2_data.csv')
2 data.head()
```

	Age	Income	Gender	MaritalStatus	Buys
0	<21	High	Male	Single	No
1	<21	High	Male	Married	No
2	21-35	High	Male	Single	Yes
3	>35	Medium	Male	Single	Yes
4	>35	Low	Female	Single	Yes

```
1 le = LabelEncoder()
2 data = data.apply(le.fit_transform)
3 x = data.iloc[:, :-1]
4 #x = x.apply(le.fit_transform)
5 x.head()
```

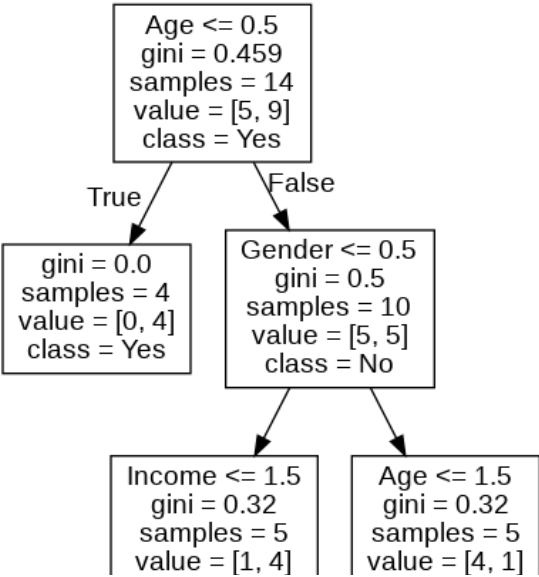
	Age	Income	Gender	MaritalStatus
0	1	0	1	1
1	1	0	1	0
2	2	2	1	1
3	2	2	1	1
4	2	1	0	1

A Google Drive error has occurred.

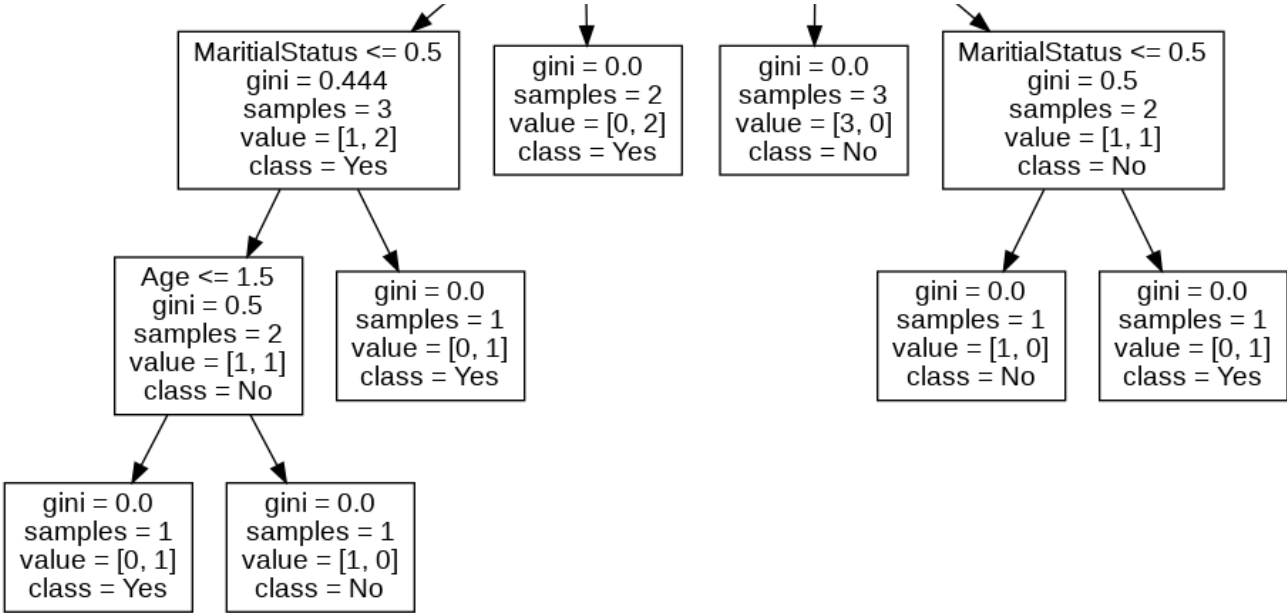
```
1 y = data.iloc[:, -1]
2 y.head()
```

```
0    0
1    0
2    1
3    1
```

```
4     1
5     2
6     3
7     4
8     5
9     6
10    7
11    8
12    9
13   10
14   11
15   12
16   13
17   14
18   15
19   16
20   17
21   18
22   19
23   20
24   21
25   22
26   23
27   24
28   25
29   26
30   27
31   28
32   29
33   30
34   31
35   32
36   33
37   34
38   35
39   36
40   37
41   38
42   39
43   40
44   41
45   42
46   43
47   44
48   45
49   46
50   47
51   48
52   49
53   50
54   51
55   52
56   53
57   54
58   55
59   56
60   57
61   58
62   59
63   60
64   61
65   62
66   63
67   64
68   65
69   66
70   67
71   68
72   69
73   70
74   71
75   72
76   73
77   74
78   75
79   76
80   77
81   78
82   79
83   80
84   81
85   82
86   83
87   84
88   85
89   86
90   87
91   88
92   89
93   90
94   91
95   92
96   93
97   94
98   95
99   96
100  97
101  98
102  99
103  100
104  101
105  102
106  103
107  104
108  105
109  106
110  107
111  108
112  109
113  110
114  111
115  112
116  113
117  114
118  115
119  116
120  117
121  118
122  119
123  120
124  121
125  122
126  123
127  124
128  125
129  126
130  127
131  128
132  129
133  130
134  131
135  132
136  133
137  134
138  135
139  136
140  137
141  138
142  139
143  140
144  141
145  142
146  143
147  144
148  145
149  146
150  147
151  148
152  149
153  150
154  151
155  152
156  153
157  154
158  155
159  156
160  157
161  158
162  159
163  160
164  161
165  162
166  163
167  164
168  165
169  166
170  167
171  168
172  169
173  170
174  171
175  172
176  173
177  174
178  175
179  176
180  177
181  178
182  179
183  180
184  181
185  182
186  183
187  184
188  185
189  186
190  187
191  188
192  189
193  190
194  191
195  192
196  193
197  194
198  195
199  196
200  197
201  198
202  199
203  200
204  201
205  202
206  203
207  204
208  205
209  206
210  207
211  208
212  209
213  210
214  211
215  212
216  213
217  214
218  215
219  216
220  217
221  218
222  219
223  220
224  221
225  222
226  223
227  224
228  225
229  226
230  227
231  228
232  229
233  230
234  231
235  232
236  233
237  234
238  235
239  236
240  237
241  238
242  239
243  240
244  241
245  242
246  243
247  244
248  245
249  246
250  247
251  248
252  249
253  250
254  251
255  252
256  253
257  254
258  255
259  256
260  257
261  258
262  259
263  260
264  261
265  262
266  263
267  264
268  265
269  266
270  267
271  268
272  269
273  270
274  271
275  272
276  273
277  274
278  275
279  276
280  277
281  278
282  279
283  280
284  281
285  282
286  283
287  284
288  285
289  286
290  287
291  288
292  289
293  290
294  291
295  292
296  293
297  294
298  295
299  296
300  297
301  298
302  299
303  300
304  301
305  302
306  303
307  304
308  305
309  306
310  307
311  308
312  309
313  310
314  311
315  312
316  313
317  314
318  315
319  316
320  317
321  318
322  319
323  320
324  321
325  322
326  323
327  324
328  325
329  326
330  327
331  328
332  329
333  330
334  331
335  332
336  333
337  334
338  335
339  336
340  337
341  338
342  339
343  340
344  341
345  342
346  343
347  344
348  345
349  346
350  347
351  348
352  349
353  350
354  351
355  352
356  353
357  354
358  355
359  356
360  357
361  358
362  359
363  360
364  361
365  362
366  363
367  364
368  365
369  366
370  367
371  368
372  369
373  370
374  371
375  372
376  373
377  374
378  375
379  376
380  377
381  378
382  379
383  380
384  381
385  382
386  383
387  384
388  385
389  386
390  387
391  388
392  389
393  390
394  391
395  392
396  393
397  394
398  395
399  396
400  397
401  398
402  399
403  400
404  401
405  402
406  403
407  404
408  405
409  406
410  407
411  408
412  409
413  410
414  411
415  412
416  413
417  414
418  415
419  416
420  417
421  418
422  419
423  420
424  421
425  422
426  423
427  424
428  425
429  426
430  427
431  428
432  429
433  430
434  431
435  432
436  433
437  434
438  435
439  436
440  437
441  438
442  439
443  440
444  441
445  442
446  443
447  444
448  445
449  446
450  447
451  448
452  449
453  450
454  451
455  452
456  453
457  454
458  455
459  456
460  457
461  458
462  459
463  460
464  461
465  462
466  463
467  464
468  465
469  466
470  467
471  468
472  469
473  470
474  471
475  472
476  473
477  474
478  475
479  476
480  477
481  478
482  479
483  480
484  481
485  482
486  483
487  484
488  485
489  486
490  487
491  488
492  489
493  490
494  491
495  492
496  493
497  494
498  495
499  496
500  497
501  498
502  499
503  500
504  501
505  502
506  503
507  504
508  505
509  506
510  507
511  508
512  509
513  510
514  511
515  512
516  513
517  514
518  515
519  516
520  517
521  518
522  519
523  520
524  521
525  522
526  523
527  524
528  525
529  526
530  527
531  528
532  529
533  530
534  531
535  532
536  533
537  534
538  535
539  536
540  537
541  538
542  539
543  540
544  541
545  542
546  543
547  544
548  545
549  546
550  547
551  548
552  549
553  550
554  551
555  552
556  553
557  554
558  555
559  556
560  557
561  558
562  559
563  560
564  561
565  562
566  563
567  564
568  565
569  566
570  567
571  568
572  569
573  570
574  571
575  572
576  573
577  574
578  575
579  576
580  577
581  578
582  579
583  580
584  581
585  582
586  583
587  584
588  585
589  586
590  587
591  588
592  589
593  590
594  591
595  592
596  593
597  594
598  595
599  596
600  597
601  598
602  599
603  600
604  601
605  602
606  603
607  604
608  605
609  606
610  607
611  608
612  609
613  610
614  611
615  612
616  613
617  614
618  615
619  616
620  617
621  618
622  619
623  620
624  621
625  622
626  623
627  624
628  625
629  626
630  627
631  628
632  629
633  630
634  631
635  632
636  633
637  634
638  635
639  636
640  637
641  638
642  639
643  640
644  641
645  642
646  643
647  644
648  645
649  646
650  647
651  648
652  649
653  650
654  651
655  652
656  653
657  654
658  655
659  656
660  657
661  658
662  659
663  660
664  661
665  662
666  663
667  664
668  665
669  666
670  667
671  668
672  669
673  670
674  671
675  672
676  673
677  674
678  675
679  676
680  677
681  678
682  679
683  680
684  681
685  682
686  683
687  684
688  685
689  686
690  687
691  688
692  689
693  690
694  691
695  692
696  693
697  694
698  695
699  696
700  697
701  698
702  699
703  700
704  701
705  702
706  703
707  704
708  705
709  706
710  707
711  708
712  709
713  710
714  711
715  712
716  713
717  714
718  715
719  716
720  717
721  718
722  719
723  720
724  721
725  722
726  723
727  724
728  725
729  726
730  727
731  728
732  729
733  730
734  731
735  732
736  733
737  734
738  735
739  736
740  737
741  738
742  739
743  740
744  741
745  742
746  743
747  744
748  745
749  746
750  747
751  748
752  749
753  750
754  751
755  752
756  753
757  754
758  755
759  756
760  757
761  758
762  759
763  760
764  761
765  762
766  763
767  764
768  765
769  766
770  767
771  768
772  769
773  770
774  771
775  772
776  773
777  774
778  775
779  776
780  777
781  778
782  779
783  780
784  781
785  782
786  783
787  784
788  785
789  786
790  787
791  788
792  789
793  790
794  791
795  792
796  793
797  794
798  795
799  796
800  797
801  798
802  799
803  800
804  801
805  802
806  803
807  804
808  805
809  806
810  807
811  808
812  809
813  810
814  811
815  812
816  813
817  814
818  815
819  816
820  817
821  818
822  819
823  820
824  821
825  822
826  823
827  824
828  825
829  826
830  827
831  828
832  829
833  830
834  831
835  832
836  833
837  834
838  835
839  836
840  837
841  838
842  839
843  840
844  841
845  842
846  843
847  844
848  845
849  846
850  847
851  848
852  849
853  850
854  851
855  852
856  853
857  854
858  855
859  856
860  857
861  858
862  859
863  860
864  861
865  862
866  863
867  864
868  865
869  866
870  867
871  868
872  869
873  870
874  871
875  872
876  873
877  874
878  875
879  876
880  877
881  878
882  879
883  880
884  881
885  882
886  883
887  884
888  885
889  886
890  887
891  888
892  889
893  890
894  891
895  892
896  893
897  894
898  895
899  896
900  897
901  898
902  899
903  900
904  901
905  902
906  903
907  904
908  905
909  906
910  907
911  908
912  909
913  910
914  911
915  912
916  913
917  914
918  915
919  916
920  917
921  918
922  919
923  920
924  921
925  922
926  923
927  924
928  925
929  926
930  927
931  928
932  929
933  930
934  931
935  932
936  933
937  934
938  935
939  936
940  937
941  938
942  939
943  940
944  941
945  942
946  943
947  944
948  945
949  946
950  947
951  948
952  949
953  950
954  951
955  952
956  953
957  954
958  955
959  956
960  957
961  958
962  959
963  960
964  961
965  962
966  963
967  964
968  965
969  966
970  967
971  968
972  969
973  970
974  971
975  972
976  973
977  974
978  975
979  976
980  977
981  978
982  979
983  980
984  981
985  982
986  983
987  984
988  985
989  986
990  987
991  988
992  989
993  990
994  991
995  992
996  993
997  994
998  995
999  996
1000 997
1001 998
1002 999
1003 1000
1004 1001
1005 1002
1006 1003
1007 1004
1008 1005
1009 1006
1010 1007
1011 1008
1012 1009
1013 1010
1014 1011
1015 1012
1016 1013
1017 1014
1018 1015
1019 1016
1020 1017
1021 1018
1022 1019
1023 1020
1024 1021
1025 1022
1026 1023
1027 1024
1028 1025
1029 1026
1030 1027
1031 1028
1032 1029
1033 1030
1034 1031
1035 1032
1036 1033
1037 1034
1038 1035
1039 1036
1040 1037
1041 1038
1042 1039
1043 1040
1044 1041
1045 1042
1046 1043
1047 1044
1048 1045
1049 1046
1050 1047
1051 1048
1052 1049
1053 1050
1054 1051
1055 1052
1056 1053
1057 1054
1058 1055
1059 1056
1060 1057
1061 1058
1062 1059
1063 1060
1064 1061
1065 1062
1066 1063
1067 1064
1068 1065
1069 1066
1070 1067
1071 1068
1072 1069
1073 1070
1074 1071
1075 1072
1076 1073
1077 1074
1078 1075
1079 1076
1080 1077
1081 1078
1082 1079
1083 1080
1084 1081
1085 1082
1086 1083
1087 1084
1088 1085
1089 1086
1090 1087
1091 1088
1092 1089
1093 1090
1094 1091
1095 1092
1096 1093
1097 1094
1098 1095
1099 1096
1100 1097
1101 1098
1102 1099
1103 1100
1104 1101
1105 1102
1106 1103
1107 1104
1108 1105
1109 1106
1110 1107
1111 1108
1112 1109
1113 1110
1114 1111
1115 1112
1116 1113
1117 1114
1118 1115
1119 1116
1120 1117
1121 1118
1122 1119
1123 1120
1124 1121
1125 1122
1126 1123
1127 1124
1128 1125
1129 1126
1130 1127
1131 1128
1132 1129
1133 1130
1134 1131
1135 1132
1136 1133
1137 1134
1138 1135
1139 1136
1140 1137
1141 1138
1142 1139
1143 1140
1144 1141
1145 1142
1146 1143
1147 1144
1148 1145
1149 1146
1150 1147
1151 1148
1152 1149
1153 1150
1154 1151
1155 1152
1156 1153
1157 1154
1158 1155
1159 1156
1160 1157
1161 1158
1162 1159
1163 1160
1164 1161
1165 1162
1166 1163
1167 1164
1168 1165
1169 1166
1170 1167
1171 1168
1172 1169
1173 1170
1174 1171
1175 1172
1176 1173
1177 1174
1178 1175
1179 1176
1180 1177
1181 1178
1182 1179
1183 1180
1184 1181
1185 1182
1186 1183
1187 1184
1188 1185
1189 1186
1190 1187
1191 1188
1192 1189
1193 1190
1194 1191
1195 1192
1196 1193
1197 1194
1198 1195
1199 1196
1200 1197
1201 1198
1202 1199
1203 1200
1204 1201
1205 1202
1206 1203
1207 1204
1208 1205
1209 1206
1210 1207
1211 1208
1212 1209
1213 1210
1214 1211
1215 1212
1216 1213
1217 1214
1218 1215
1219 1216
1220 1217
1221 1218
1222 1219
1223 1220
1224 1221
1225 1222
1226 1223
1227 1224
1228 1225
1229 1226
1230 1227
1231 1228
1232 1229
1233 1230
1234 1231
1235 1232
1236 1233
1237 1234
1238 1235
1239 1236
1240 1237
1241 1238
1242 1239
1243 1240
1244 1241
1245 1242
1246 1243
1247 1244
1248 1245
1249 1246
1250 1247
1251 1248
1252 1249
1253 1250
1254 1251
1255 1252
1256 1253
1257 1254
1258 1255
1259 1256
1260 1257
1261 1258
1262 1259
1263 1260
1264 1261
1265 1262
1266 1263
1267 1264
1268 1265
1269 1266
1270 1267
1271 1268
1272 1269
1273 1270
1274 1271
1275 1272
1276 1273
1277 1274
1278 1275
1279 1276
1280 1277
1281 1278
1282 1279
1283 1280
1284 1281
1285 1282
1286 1283
1287 1284
1288 1285
1289 1286
1290 1287
1291 1288
1292 1289
1293 1290
1294 1291
1295 1292
1296 1293
1297 1294
1298 1295
1299 1296
1300 1297
1301 1298
1302 1299
1303 1300
1304 1301
1305 1302
1306 1303
1307 1304
1308 1305
1309 1306
1310 1307
1311 1308
1312 1309
1313 1310
1314 1311
1315 1312
1316 1313
1317 1314
1318 1315
1319 1316
1320 1317
1321 1318
1322 1319
1323 1320
1324 1321
1325 1322
1326 1323
1327 1324
1328 1325
1329 1326
1330 1327
1331 1328
1332 1329
1333 1330
1334 1331
1335 1332
1336 1333
1337 1334
1338 1335
1339 1336
1340 1337
1341 1338
1342 1339
1343 1340
1344 1341
1345 1342
1346 1343
1347 1344
1348 1345
1349 1346
1350 1347
1351 1
```



1



```
1 train_data_X = [[2, 4], [4, 4], [4, 6], [4, 2], [6, 2], [6, 4]]
2 train_data_y = ["Orange", "Blue", "Orange", "Orange", "Blue", "Orange"]
3 test_data = [[6, 6]]
4

1 import math

1 class kNN:
2     def __init__(self, k=2, algorithm = 'auto'):
3         self.k = k
4         self.X = []
5         self.y = []
6         self.algorithm = algorithm
7
8     def get_distance(self, pt1, pt2):
9         return math.sqrt((pt1[0]-pt2[0])*(pt1[0]-pt2[0]) + (pt1[1]-pt2[1])*(pt1[1]-pt2[1]))
10
11    def fit(self, X, y):
12        self.X = X
13        self.y = y
14
15    def auto_knn(self, test, distances):
16        prediction = max((distances), key = lambda tup: tup[1])
17        return prediction[1]
18
19    def distance_weighted_knn(self, test, distances):
20        weights = {}
21        for d in distances:
22            try:
23                weights[d[1]]+=float(1/d[0])
24            except:
25                weights[d[1]] = float(1/d[0])
26        prediction = max((weights), key = lambda x: weights[x])
27        return prediction
28
29    def locally_weighted_averaging_knn(self, test, distances):
30        frequencies = {}
31        weights = {}
32        for d in distances:
33            try:
34                weights[d[1]]+=float(1/d[0])
35                frequencies[d[1]]+=1
36            except:
37                weights[d[1]] = float(1/d[0])
38                frequencies[d[1]]=1
39
40        for w in weights:
41            weights[w]/= frequencies[w]
42
43        prediction = max((weights), key = lambda x: weights[x])
44        return prediction
45
```

```

46 def predict(self, tests):
47     results = []
48     for test in tests:
49         distances = []
50
51         for i in range(len(self.X)):
52             distances.append([self.get_distance(self.X[i], test), self.y[i]])
53         distances.sort(key=lambda tup: tup[0])
54         distances = distances[:self.k]
55         print("Nearest Neighbours:", distances)
56
57         if self.algorithm == 'auto':
58             result = self.auto_knn(test, distances)
59             results.append(result)
60         elif self.algorithm == 'distance-weighted':
61             result = self.distance_weighted_knn(test, distances)
62             results.append(result)
63         elif self.algorithm == 'locally-weighted-averaging':
64             result = self.locally_weighted_averaging_knn(test, distances)
65             results.append(result)
66     return results
67

```

## ▼ Basic kNN

```

1 n = kNN(3)
2 n.fit(train_data_X, train_data_y)
3 y_test = n.predict(test_data)
4 print(y_test)

```

```

Nearest Neighbours: [[2.0, 'Orange'], [2.0, 'Orange'], [2.8284271247461903, 'Orange']]

```

## ▼ Distance Weighted kNN

```

1 n = kNN(3, 'distance-weighted')
2 n.fit(train_data_X, train_data_y)
3 y_test = n.predict(test_data)
4 print(y_test)

```

```

Nearest Neighbours: [[2.0, 'Orange'], [2.0, 'Orange'], [2.8284271247461903, 'Orange']]

```

## ▼ Locally Weighted Averaging kNN

```

1 n = kNN(3, 'locally-weighted-averaging')
2 n.fit(train_data_X, train_data_y)

```

```
3 y_test = n.predict(test_data)\n4 print(y_test)
```

```
Nearest Neighbours: [[2.0, 'Orange'], [2.0, 'Orange'], [2.8284271247461903, 'Orange']\n['Orange']
```

---

---

✓ 0s completed at 17:20





```
1 from google.colab import drive
2 drive.mount('/content/gdrive')
```

Mounted at /content/gdrive

```
1 from copy import deepcopy
2 import numpy as np
3 import pandas as pd
4 from matplotlib import pyplot as plt
5 import math
6 import seaborn as sns
```

```
1 data = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/LP3/ML/ML_4_data.csv')
2 data.head()
```

	x	y
0	0.10	0.60
1	0.15	0.71
2	0.08	0.90
3	0.16	0.85
4	0.20	0.30

```
1 X = np.array(data)

1 c_x = np.array([0.1,0.3])
2 c_y = np.array([0.6,0.2])
3
4 centroids = np.array(list(zip(c_x,c_y)))
5 centroids
```

```
array([[0.1, 0.6],
       [0.3, 0.2]])
```

```
1 class K_Means:
2     def __init__(self, k=2, tol=0.01, max_iter = 300):
3         self.k = k
4         self.tol = tol
5         self.max_iter = max_iter
6         self.cur_centroid = []
7
8     def get_cluster(self, data):
9         distances = []
10        x1 = data[0]
11        y1 = data[1]
12
13        for centroid in self.cur_centroid:
```

```

14     distances.append((y1-centroid[1])*(y1-centroid[1]) + (x1-centroid[0])*(x1-
15
16     classification = distances.index(min(distances))
17     return classification
18
19 def fit(self, data, centroid):
20     self.cur_centroid = centroid
21     new_centroids = []
22
23     iter = 0
24
25     while iter<self.max_iter:
26         i=0
27         cur_clusters = [[] for i in range(self.k)]
28
29         for pt in data:
30             clust = self.get_cluster(pt)
31             cur_clusters[clust].append(pt)
32
33         new_centroids = [np.average(cur_clusters[i], axis=0) for i in range(self.k)]
34
35         optimised = True
36         for i in range(len(self.cur_centroid)):
37             if (np.abs(np.sum((new_centroids[i]-self.cur_centroid[i])/self.cur_centroid[i])) > 0.01):
38                 optimised = False
39
40         if optimised == True:
41             break
42
43         self.cur_centroid = new_centroids
44         iter+=1
45
46     print("Final centroids", self.cur_centroid)
47     return cur_clusters, self.cur_centroid

```

```

1 km = K_Means(2, 0.01, 1)
2 clusters, centroids = km.fit(X, centroids)

```

```
Final centroids [array([0.148, 0.712]), array([0.24666667, 0.24666667])]
```

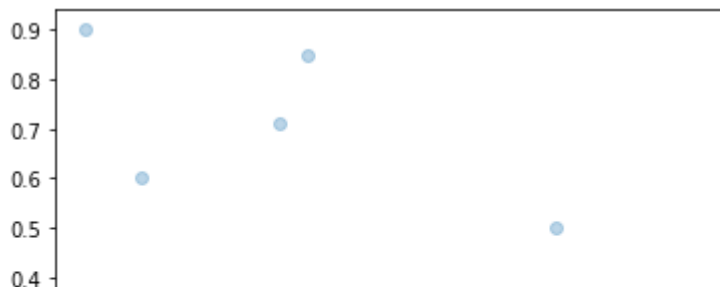
## ▼ Before Clustering

```

1 plt.figure()
2 plt.scatter(X[:,0],X[:,1],alpha=0.3)
3 plt.show()

```

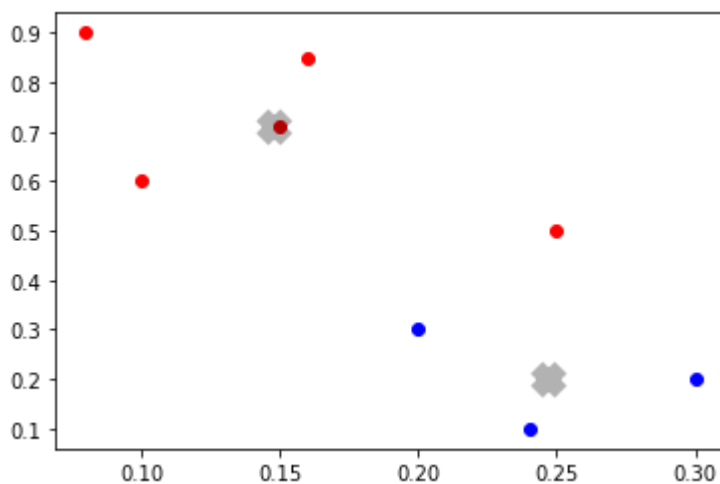




## ▼ After Clustering



```
1 colors = ['r', 'b']
2 plt.figure()
3
4 for i in range(len(clusters)):
5     for c in clusters[i]:
6         plt.scatter(c[0], c[1], color = colors[i])
7
8 for i in range(len(centroids)):
9     plt.scatter(centroids[i][0], centroids[i][1], marker = 'x', color = 'black',
10 plt.show()
```



1

---

✓ 0s completed at 22:54

