Reppeto530Week3

December 17, 2023

1 Chapter 1

Brian Reppeto 530 Prof. Jim Week 3 HW

[35]:
```
# Import and download the code from the github repo

from os.path import basename, exists


def download(url):
    filename = basename(url)
    if not exists(filename):
        from urllib.request import urlretrieve

        local, _ = urlretrieve(url, filename)
        print("Downloaded " + local)


download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪thinkstats2.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/thinkplot.
 ↪py")
```

[36]:
```
# This part is to download the specific files needed for the assignment

download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/nsfg.py")

download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemPreg.dct")
download(
    "https://github.com/AllenDowney/ThinkStats2/raw/master/code/2002FemPreg.dat.
 ↪gz"
)
```

[37]:
```
# Importing the .py file to work with in the assignment

import nsfg
```

[79]:
```
# Reading the .py file and setting it = to preg

preg = nsfg.ReadFemPreg()
```

1.1 Exercise 1-1

Select the `birthord` column, print the value counts, and compare to results published in the codebook

[80]:
```
# Using the preg DF, select the "birthord" column and count the number of times␣
 ↪each occur.
# Then sort the output based on the cloumn
preg.birthord.value_counts().sort_index()
```

[80]:
```
birthord
1.0     4413
2.0     2874
3.0     1234
4.0      421
5.0      126
6.0       50
7.0       20
8.0        7
9.0        2
10.0       1
Name: count, dtype: int64
```

We can also use `isnull` to count the number of nans.

[50]:
```
# Using the preg DF, select the "birthord" column and sum the number of times␣
 ↪each the column isnull.

preg.birthord.isnull().sum()
```

[50]:
```
4445
```

Select the `prglngth` column, print the value counts, and compare to results published in the codebook

[51]:
```
# Using the preg DF, select the "prglngth" column and count the number of times␣
 ↪each occur.
# Then sort the output based on the cloumn
preg.prglngth.value_counts().sort_index()
```

[51]:
```
prglngth
0       15
1        9
2       78
3      151
```

```
4      412
5      181
6      543
7      175
8      409
9      594
10     137
11     202
12     170
13     446
14      29
15      39
16      44
17     253
18      17
19      34
20      18
21      37
22     147
23      12
24      31
25      15
26     117
27       8
28      38
29      23
30     198
31      29
32     122
33      50
34      60
35     357
36     329
37     457
38     609
39    4744
40    1120
41     591
42     328
43     148
44      46
45      10
46       1
47       1
48       7
50       2
Name: count, dtype: int64
```

To compute the mean of a column, you can invoke the `mean` method on a Series. For example, here is the mean birthweight in pounds:

[52]:
```
preg.totalwgt_lb.mean()
```

[52]:
```
7.265628457623368
```

Create a new column named totalwgt_kg that contains birth weight in kilograms. Compute its mean. Remember that when you create a new column, you have to use dictionary syntax, not dot notation.

[57]:
```
# Create a column named "totalwgt_kg" there are 2.2 Kg in a pound so convert␣
 ↪the pound column to get the Kg.
# Then set the new column = to the calc
# print the mean of the new kg column

preg['totalwgt_kg'] = preg.totalwgt_lb / 2.2
# print (preg.head())
preg.totalwgt_kg.mean()
```

[57]:
```
3.302558389828803
```

`nsfg.py` also provides `ReadFemResp`, which reads the female respondents file and returns a `DataFrame`:

[58]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemResp.dct")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪2002FemResp.dat.gz")
```

```
Downloaded 2002FemResp.dct
Downloaded 2002FemResp.dat.gz
```

[59]:
```
resp = nsfg.ReadFemResp()
```

`DataFrame` provides a method `head` that displays the first five rows:

[60]:
```
resp.head()
```

[60]:

	caseid	rscrinf	rdormres	rostscrn	rscreenhisp	rscreenrace	age_a	\
0	2298	1	5	5	1	5.0	27	
1	5012	1	5	1	5	5.0	42	
2	11586	1	5	1	5	5.0	43	
3	6794	5	5	4	1	5.0	15	
4	616	1	5	4	1	5.0	20	

	age_r	cmbirth	agescrn	…	pubassis_i	basewgt	adj_mod_basewgt	\
0	27	902	27	…	0	3247.916977	5123.759559	
1	42	718	42	…	0	2335.279149	2846.799490	
2	43	708	43	…	0	2335.279149	2846.799490	

```
3     15     1042       15  …           0  3783.152221      5071.464231
4     20      991       20  …           0  5341.329968      6437.335772

      finalwgt  secu_r  sest  cmintvw  cmlstyr  screentime   intvlngth
0  5556.717241       2    18     1234     1222    18:26:36  110.492667
1  4744.191350       2    18     1233     1221    16:30:59   64.294000
2  4744.191350       2    18     1234     1222    18:19:09   75.149167
3  5923.977368       2    18     1234     1222    15:54:43   28.642833
4  7229.128072       2    18     1233     1221    14:19:44   69.502667

[5 rows x 3087 columns]
```

Select the `age_r` column from `resp` and print the value counts. How old are the youngest and oldest respondents?

[62]:
```
# Using the resp DF, select the "age_r" column and count the number of times␣
  ↪each occur.
# Then sort the output based on the cloumn

resp.age_r.value_counts().sort_index()
```

[62]:
```
age_r
15    217
16    223
17    234
18    235
19    241
20    258
21    267
22    287
23    282
24    269
25    267
26    260
27    255
28    252
29    262
30    292
31    278
32    273
33    257
34    255
35    262
36    266
37    271
38    256
39    215
40    256
```

```
41    250
42    215
43    253
44    235
Name: count, dtype: int64
```

We can use the `caseid` to match up rows from `resp` and `preg`. For example, we can select the row from `resp` for `caseid` 2298 like this:

[63]:
```
resp[resp.caseid==2298]
```

[63]:
```
   caseid  rscrinf  rdormres  rostscrn  rscreenhisp  rscreenrace  age_a  \
0    2298        1         5         5            1          5.0     27

   age_r  cmbirth  agescrn  …  pubassis_i      basewgt  adj_mod_basewgt  \
0     27      902       27  …           0  3247.916977      5123.759559

      finalwgt  secu_r  sest  cmintvw  cmlstyr  screentime   intvlngth
0  5556.717241       2    18     1234     1222    18:26:36  110.492667

[1 rows x 3087 columns]
```

And we can get the corresponding rows from `preg` like this:

[64]:
```
preg[preg.caseid==2298]
```

[64]:
```
      caseid  pregordr  howpreg_n  howpreg_p  moscurrp  nowprgdk  pregend1  \
2610    2298         1        NaN        NaN       NaN       NaN       6.0
2611    2298         2        NaN        NaN       NaN       NaN       6.0
2612    2298         3        NaN        NaN       NaN       NaN       6.0
2613    2298         4        NaN        NaN       NaN       NaN       6.0

      pregend2  nbrnaliv  multbrth  …  religion_i  metro_i      basewgt  \
2610       NaN       1.0       NaN  …           0        0  3247.916977
2611       NaN       1.0       NaN  …           0        0  3247.916977
2612       NaN       1.0       NaN  …           0        0  3247.916977
2613       NaN       1.0       NaN  …           0        0  3247.916977

      adj_mod_basewgt     finalwgt  secu_p  sest  cmintvw  totalwgt_lb  \
2610      5123.759559  5556.717241       2    18      NaN       6.8750
2611      5123.759559  5556.717241       2    18      NaN       5.5000
2612      5123.759559  5556.717241       2    18      NaN       4.1875
2613      5123.759559  5556.717241       2    18      NaN       6.8750

      totalwgt_kg
2610     3.125000
2611     2.500000
2612     1.903409
```

```
2613     3.125000

[4 rows x 245 columns]
```

How old is the respondent with `caseid` 1?

[67]:
```
# Use the DF resp and then select the DF resp with column caseid that =1 then␣
 ↪return the column age_r to return the age of
# the pregnancy with caseid =1

resp[resp.caseid==1].age_r
```

[67]:
```
1069    44
Name: age_r, dtype: int64
```

What are the pregnancy lengths for the respondent with `caseid` 2298?

[77]:
```
# Use the DF preg and then select the DF preg with column caseid that = 2298␣
 ↪then return the column prglngth to return the age of
# the pregnancy with caseid = 2298


preg[preg.caseid==2298].prglngth
```

[77]:
```
2610    40
2611    36
2612    30
2613    40
Name: prglngth, dtype: int64
```

What was the birthweight of the first baby born to the respondent with `caseid` 5012?

[78]:
```
# Use the DF preg and then select the DF preg with column caseid that = 5012␣
 ↪then return the column birthwgt_lb
# to return the birth weight.  This respondent only had one pregnacy.


preg[preg.caseid==5012].birthwgt_lb
```

[78]:
```
5515    6.0
Name: birthwgt_lb, dtype: float64
```

1.2 Exercise 1-2

[81]:
```
# Reading the .py file and setting it = to resp

resp=nsfg.ReadFemResp()
```

[84]:
```
# Print the value count for this variable

resp.pregnum.value_counts()
```

[84]:
```
pregnum
0     2610
2     1432
1     1267
3     1110
4      611
5      305
6      150
7       80
8       40
9       21
10       9
11       3
12       2
14       2
19       1
Name: count, dtype: int64
```

[87]:
```
# Print the value count to compare to the resp DF.  The counts do not align.

preg.pregnum.value_counts()
```

[87]:
```
pregnum
3     3330
2     2864
4     2444
5     1525
1     1267
6      900
7      560
8      320
9      189
10      90
11      33
14      28
12      24
19      19
Name: count, dtype: int64
```

[100]:
```
# Test the cross-validation via caseid.

# Print the resp df caseid ==2 identifying the caseid and the pregnum fields to␣
 ↪compare cross-val.
```

```
resp_rows = resp[resp.caseid == 2]
print(resp_rows[['caseid', 'pregnum']])

# Print the preg df caseid ==2 identifying the caseid and the pregnum fields to␣
 ↪compare cross-val
preg_rows = preg[preg.caseid == 2]
print(preg_rows[['caseid', 'pregnum']])


# both df cross-valid for pregnancy number
```

```
      caseid  pregnum
7010       2        3
   caseid  pregnum
2       2        3
3       2        3
4       2        3
```

[104]:
```
# This maps the caseid to index

nsfg.MakePregMap(resp)
```

[104]:
```
defaultdict(list,
            {2298: [0],
             5012: [1],
             11586: [2],
             6794: [3],
             616: [4],
             845: [5],
             10333: [6],
             855: [7],
             8656: [8],
             3566: [9],
             5917: [10],
             9200: [11],
             6320: [12],
             11700: [13],
             7354: [14],
             3697: [15],
             4881: [16],
             5862: [17],
             8542: [18],
             2054: [19],
             3719: [20],
             11740: [21],
             11343: [22],
             7075: [23],
```

```
5422: [24],
2178: [25],
8358: [26],
5083: [27],
1545: [28],
5656: [29],
9334: [30],
5507: [31],
611: [32],
4260: [33],
11767: [34],
5573: [35],
11901: [36],
8975: [37],
5267: [38],
910: [39],
4463: [40],
8954: [41],
1814: [42],
7011: [43],
4057: [44],
7081: [45],
5499: [46],
6551: [47],
9242: [48],
11408: [49],
7168: [50],
2339: [51],
4138: [52],
8785: [53],
1511: [54],
2240: [55],
8190: [56],
5278: [57],
469: [58],
5351: [59],
10633: [60],
11058: [61],
3329: [62],
3814: [63],
4966: [64],
4327: [65],
6908: [66],
9302: [67],
4627: [68],
8085: [69],
9555: [70],
```

```
1821: [71],
319: [72],
10295: [73],
8269: [74],
4693: [75],
6370: [76],
8628: [77],
2171: [78],
581: [79],
8065: [80],
8081: [81],
2286: [82],
7758: [83],
2401: [84],
4795: [85],
6145: [86],
10463: [87],
547: [88],
2522: [89],
997: [90],
1786: [91],
4308: [92],
3800: [93],
7599: [94],
9866: [95],
5338: [96],
10289: [97],
2281: [98],
4150: [99],
3679: [100],
1810: [101],
7369: [102],
5243: [103],
1014: [104],
5455: [105],
8417: [106],
5296: [107],
7483: [108],
6397: [109],
7430: [110],
545: [111],
8661: [112],
3700: [113],
12031: [114],
5288: [115],
9846: [116],
2137: [117],
```

```
11470: [118],
3349: [119],
12014: [120],
1703: [121],
3391: [122],
6953: [123],
8270: [124],
175: [125],
9680: [126],
7598: [127],
6716: [128],
251: [129],
12109: [130],
7497: [131],
12219: [132],
1877: [133],
8366: [134],
785: [135],
5321: [136],
8062: [137],
3821: [138],
2749: [139],
6963: [140],
7498: [141],
435: [142],
10069: [143],
9849: [144],
3372: [145],
9527: [146],
5630: [147],
5078: [148],
9810: [149],
11421: [150],
9523: [151],
6281: [152],
5365: [153],
12434: [154],
7114: [155],
6441: [156],
12044: [157],
6032: [158],
2885: [159],
11333: [160],
5343: [161],
1798: [162],
2788: [163],
9726: [164],
```

```
5652: [165],
3508: [166],
2460: [167],
505: [168],
8617: [169],
11603: [170],
5479: [171],
2963: [172],
8389: [173],
10242: [174],
2336: [175],
11484: [176],
5539: [177],
8181: [178],
10860: [179],
8195: [180],
6832: [181],
2678: [182],
4790: [183],
3570: [184],
8817: [185],
9927: [186],
10844: [187],
11185: [188],
12267: [189],
9133: [190],
9516: [191],
1944: [192],
2587: [193],
6896: [194],
12205: [195],
7069: [196],
2257: [197],
2456: [198],
4500: [199],
5270: [200],
118: [201],
2077: [202],
7156: [203],
11823: [204],
10367: [205],
9227: [206],
5644: [207],
4200: [208],
12286: [209],
6151: [210],
5068: [211],
```

```
61: [212],
116: [213],
2196: [214],
8004: [215],
12400: [216],
1182: [217],
5666: [218],
10016: [219],
2218: [220],
9587: [221],
2200: [222],
4527: [223],
7770: [224],
11777: [225],
8036: [226],
1349: [227],
9983: [228],
2808: [229],
8100: [230],
3026: [231],
10803: [232],
781: [233],
2899: [234],
1561: [235],
5547: [236],
112: [237],
5378: [238],
9974: [239],
1910: [240],
1089: [241],
8196: [242],
1531: [243],
37: [244],
1045: [245],
3598: [246],
9658: [247],
5924: [248],
1518: [249],
11030: [250],
9376: [251],
7543: [252],
6178: [253],
12428: [254],
10113: [255],
5720: [256],
7053: [257],
5166: [258],
```

```
10261: [259],
12494: [260],
3660: [261],
1774: [262],
5731: [263],
8413: [264],
10201: [265],
12482: [266],
12215: [267],
11371: [268],
5176: [269],
8544: [270],
4921: [271],
11874: [272],
7799: [273],
5774: [274],
2838: [275],
9190: [276],
5540: [277],
2399: [278],
671: [279],
3319: [280],
2027: [281],
901: [282],
6295: [283],
9969: [284],
7848: [285],
12382: [286],
4233: [287],
1424: [288],
12550: [289],
1612: [290],
3953: [291],
765: [292],
1100: [293],
9667: [294],
4080: [295],
5915: [296],
9254: [297],
6718: [298],
9955: [299],
197: [300],
5323: [301],
9179: [302],
227: [303],
10456: [304],
7965: [305],
```

```
11919: [306],
3478: [307],
7926: [308],
6428: [309],
5480: [310],
10535: [311],
12504: [312],
9972: [313],
11747: [314],
1611: [315],
5688: [316],
4858: [317],
6140: [318],
11834: [319],
7628: [320],
4174: [321],
8674: [322],
10140: [323],
3402: [324],
385: [325],
1440: [326],
4832: [327],
8079: [328],
12422: [329],
8561: [330],
4999: [331],
6332: [332],
5478: [333],
10885: [334],
7837: [335],
10029: [336],
4006: [337],
7984: [338],
657: [339],
11958: [340],
1932: [341],
11613: [342],
11212: [343],
1017: [344],
1481: [345],
7046: [346],
6434: [347],
9859: [348],
943: [349],
3728: [350],
9207: [351],
8050: [352],
```

```
3447: [353],
11785: [354],
7008: [355],
2297: [356],
1654: [357],
8737: [358],
6594: [359],
2951: [360],
4153: [361],
6456: [362],
1752: [363],
12107: [364],
1707: [365],
6715: [366],
7883: [367],
954: [368],
7569: [369],
4292: [370],
10652: [371],
10247: [372],
12495: [373],
6329: [374],
10105: [375],
4372: [376],
7513: [377],
6782: [378],
7627: [379],
10205: [380],
12193: [381],
8080: [382],
11132: [383],
4885: [384],
10364: [385],
7567: [386],
11774: [387],
11152: [388],
12389: [389],
1315: [390],
7732: [391],
5898: [392],
5022: [393],
11382: [394],
8082: [395],
11858: [396],
11681: [397],
12113: [398],
458: [399],
```

```
3934: [400],
5234: [401],
9829: [402],
8743: [403],
10058: [404],
1833: [405],
8465: [406],
9518: [407],
8918: [408],
12344: [409],
393: [410],
2798: [411],
4266: [412],
1147: [413],
541: [414],
11898: [415],
8148: [416],
3463: [417],
5306: [418],
418: [419],
8675: [420],
12448: [421],
5850: [422],
8252: [423],
5842: [424],
7915: [425],
10189: [426],
4069: [427],
1853: [428],
258: [429],
8840: [430],
9842: [431],
7694: [432],
9861: [433],
4258: [434],
1418: [435],
6236: [436],
585: [437],
8334: [438],
6205: [439],
4177: [440],
2873: [441],
10694: [442],
3457: [443],
1499: [444],
5295: [445],
163: [446],
```

```
3835: [447],
5160: [448],
12468: [449],
10084: [450],
2306: [451],
6074: [452],
3039: [453],
5107: [454],
7931: [455],
399: [456],
1122: [457],
6546: [458],
2915: [459],
7066: [460],
2213: [461],
2059: [462],
8602: [463],
11975: [464],
3933: [465],
8966: [466],
6859: [467],
12162: [468],
8909: [469],
8613: [470],
10309: [471],
10874: [472],
217: [473],
12072: [474],
11936: [475],
7767: [476],
4923: [477],
11229: [478],
8819: [479],
5405: [480],
8117: [481],
2404: [482],
1963: [483],
5350: [484],
11611: [485],
9449: [486],
10945: [487],
5514: [488],
77: [489],
3811: [490],
22: [491],
10564: [492],
1305: [493],
```

```
2415: [494],
4946: [495],
11814: [496],
11445: [497],
5205: [498],
8478: [499],
3054: [500],
10595: [501],
10717: [502],
9696: [503],
7535: [504],
3682: [505],
6791: [506],
1491: [507],
135: [508],
218: [509],
5126: [510],
9283: [511],
10144: [512],
11985: [513],
9963: [514],
9585: [515],
5025: [516],
11710: [517],
832: [518],
8538: [519],
8802: [520],
2104: [521],
8138: [522],
522: [523],
11399: [524],
2959: [525],
374: [526],
6753: [527],
9506: [528],
2176: [529],
6897: [530],
10259: [531],
12190: [532],
11672: [533],
1228: [534],
4606: [535],
8458: [536],
12067: [537],
5048: [538],
2700: [539],
8307: [540],
```

```
1734: [541],
10561: [542],
10335: [543],
1422: [544],
1532: [545],
7085: [546],
3269: [547],
3251: [548],
3904: [549],
744: [550],
6640: [551],
8887: [552],
7059: [553],
4096: [554],
12276: [555],
140: [556],
8077: [557],
2406: [558],
5485: [559],
11839: [560],
5030: [561],
10492: [562],
8054: [563],
11144: [564],
4855: [565],
5329: [566],
3966: [567],
9192: [568],
2673: [569],
9675: [570],
941: [571],
6159: [572],
2795: [573],
745: [574],
4346: [575],
219: [576],
1629: [577],
9476: [578],
1597: [579],
95: [580],
8920: [581],
410: [582],
8671: [583],
4289: [584],
7741: [585],
6393: [586],
5403: [587],
```

```
751: [588],
4743: [589],
6093: [590],
2983: [591],
12156: [592],
11450: [593],
2605: [594],
2667: [595],
7343: [596],
4889: [597],
11935: [598],
6699: [599],
7509: [600],
497: [601],
11029: [602],
12089: [603],
3281: [604],
9113: [605],
10025: [606],
9348: [607],
12023: [608],
3865: [609],
5665: [610],
2422: [611],
3458: [612],
12507: [613],
10043: [614],
11161: [615],
3545: [616],
3146: [617],
2394: [618],
5996: [619],
3541: [620],
3979: [621],
5327: [622],
4810: [623],
859: [624],
12568: [625],
8139: [626],
9282: [627],
8941: [628],
4241: [629],
5801: [630],
10723: [631],
9394: [632],
546: [633],
7294: [634],
```

```
1082: [635],
11285: [636],
9515: [637],
8184: [638],
7698: [639],
4744: [640],
11896: [641],
3094: [642],
10664: [643],
2164: [644],
8242: [645],
1266: [646],
5523: [647],
7969: [648],
1773: [649],
11197: [650],
10777: [651],
6810: [652],
9271: [653],
9223: [654],
7715: [655],
12378: [656],
5538: [657],
8193: [658],
5951: [659],
4093: [660],
2521: [661],
3907: [662],
4565: [663],
3828: [664],
1892: [665],
7882: [666],
10560: [667],
10067: [668],
991: [669],
3738: [670],
10812: [671],
6027: [672],
4231: [673],
1894: [674],
9429: [675],
6694: [676],
11578: [677],
2128: [678],
8066: [679],
627: [680],
2535: [681],
```

```
1169: [682],
1171: [683],
734: [684],
8276: [685],
632: [686],
9624: [687],
9177: [688],
2439: [689],
12180: [690],
12564: [691],
7333: [692],
12294: [693],
738: [694],
9355: [695],
11471: [696],
723: [697],
11941: [698],
12493: [699],
544: [700],
10708: [701],
12074: [702],
4562: [703],
4160: [704],
8350: [705],
173: [706],
9797: [707],
6339: [708],
8532: [709],
9062: [710],
9953: [711],
11994: [712],
12556: [713],
225: [714],
7263: [715],
72: [716],
7703: [717],
365: [718],
8906: [719],
8089: [720],
4154: [721],
10103: [722],
5268: [723],
7861: [724],
1366: [725],
535: [726],
7331: [727],
8715: [728],
```

```
8279: [729],
12284: [730],
9: [731],
786: [732],
2280: [733],
12008: [734],
9714: [735],
6398: [736],
6083: [737],
3742: [738],
12356: [739],
7588: [740],
11215: [741],
10006: [742],
10152: [743],
7633: [744],
9526: [745],
6705: [746],
8702: [747],
3678: [748],
6840: [749],
7083: [750],
7507: [751],
4184: [752],
1556: [753],
6889: [754],
4540: [755],
4619: [756],
8325: [757],
2076: [758],
11665: [759],
1465: [760],
5144: [761],
9887: [762],
6196: [763],
12018: [764],
9059: [765],
39: [766],
6417: [767],
9882: [768],
5747: [769],
5948: [770],
10974: [771],
3555: [772],
10836: [773],
2609: [774],
11296: [775],
```

```
10530: [776],
9350: [777],
1313: [778],
11897: [779],
11857: [780],
890: [781],
5636: [782],
5597: [783],
9478: [784],
11601: [785],
3115: [786],
10193: [787],
12181: [788],
7992: [789],
5501: [790],
9612: [791],
2601: [792],
4631: [793],
12238: [794],
144: [795],
7544: [796],
2479: [797],
5167: [798],
10041: [799],
12451: [800],
7889: [801],
3941: [802],
12324: [803],
4123: [804],
11413: [805],
2825: [806],
238: [807],
548: [808],
11021: [809],
10083: [810],
6299: [811],
3523: [812],
6977: [813],
9139: [814],
7643: [815],
6051: [816],
7153: [817],
907: [818],
10454: [819],
237: [820],
9063: [821],
5607: [822],
```

```
3283: [823],
7921: [824],
6348: [825],
6303: [826],
7142: [827],
3862: [828],
11092: [829],
2048: [830],
1476: [831],
2239: [832],
2900: [833],
2629: [834],
9613: [835],
10439: [836],
2474: [837],
1768: [838],
7219: [839],
5712: [840],
8442: [841],
463: [842],
4558: [843],
3871: [844],
11532: [845],
21: [846],
8240: [847],
3743: [848],
1560: [849],
7254: [850],
2641: [851],
6916: [852],
4086: [853],
7695: [854],
11753: [855],
2842: [856],
2818: [857],
7802: [858],
918: [859],
717: [860],
2807: [861],
11910: [862],
9998: [863],
945: [864],
1701: [865],
11953: [866],
4238: [867],
2130: [868],
1099: [869],
```

```
2593: [870],
6125: [871],
9255: [872],
5064: [873],
2534: [874],
3188: [875],
1829: [876],
8239: [877],
4986: [878],
4820: [879],
10528: [880],
1939: [881],
6489: [882],
7854: [883],
2596: [884],
1138: [885],
10404: [886],
3656: [887],
12252: [888],
8756: [889],
1645: [890],
6856: [891],
8907: [892],
10918: [893],
4317: [894],
7386: [895],
12114: [896],
6161: [897],
44: [898],
10795: [899],
1995: [900],
10683: [901],
1790: [902],
10690: [903],
3178: [904],
10534: [905],
8923: [906],
10079: [907],
7952: [908],
10906: [909],
3140: [910],
5832: [911],
11068: [912],
4303: [913],
555: [914],
7123: [915],
867: [916],
```

```
11102: [917],
1699: [918],
6553: [919],
1705: [920],
10150: [921],
3116: [922],
11372: [923],
7804: [924],
2966: [925],
10380: [926],
8639: [927],
10729: [928],
6637: [929],
3159: [930],
3218: [931],
4027: [932],
9466: [933],
132: [934],
1153: [935],
9634: [936],
6964: [937],
6268: [938],
710: [939],
3003: [940],
12010: [941],
8934: [942],
10926: [943],
8865: [944],
11776: [945],
1575: [946],
2033: [947],
6266: [948],
10168: [949],
1906: [950],
7504: [951],
494: [952],
6403: [953],
3111: [954],
7001: [955],
9668: [956],
9900: [957],
10821: [958],
1581: [959],
8255: [960],
1570: [961],
8783: [962],
4628: [963],
```

```
3249: [964],
7756: [965],
1062: [966],
10072: [967],
8075: [968],
5513: [969],
11105: [970],
11078: [971],
8650: [972],
7934: [973],
11032: [974],
12240: [975],
952: [976],
4893: [977],
897: [978],
3597: [979],
2155: [980],
3770: [981],
3529: [982],
5535: [983],
6764: [984],
6967: [985],
5703: [986],
9447: [987],
12226: [988],
2165: [989],
3486: [990],
10160: [991],
11988: [992],
3803: [993],
2985: [994],
2005: [995],
8841: [996],
11377: [997],
4910: [998],
4689: [999],
…})
```

1.3 Exercise 2-1

Evening News

If I were to summarizes on the evening news whether first babies arrive late, I would use charts and graphs as they are easier for people to get the overall meaning of the data. Specifically, I would use a Histogram that shows the pregnancie in weeks with the first child being highlight compared to all other births. I would also use summarizing distributions to show the details between the first and other births.

1.4 Anxious Patient

Reassuring an anxious patient would require using stats that emphasize differnces in the situations or scenario to put the differences into context. A few examples of these would be Effect Size to understand the difference in the scenarios, Odds Ratios to compare the odds between the scenarios, also I think Summarizing Distributions would help explain variablity and probabilites between scenarios that patients would be able to understand in laymans terms.

1.5 Straight Dope question

Based on the statistical measurements in chapter 2, it is apparent that the belief in first babies arriving late is not statistically supported. The histogram (figure 2-5) depicting pregnancy lengths between first born and all others does not support this hypoeses and actual shows a mean where other born were at a much higher clip than that of first born. The summarization of distributions further underscores the subtle variations between first-time mothers and those with subsequent pregnancies, suggesting that the expectation of first babies arriving late may not hold true across the board. While individual experiences may vary, the comprehensive analysis of pregnancy data does not decisively support the widespread notion that first babies are predisposed to tardy arrivals. The intricacies of childbirth, influenced by numerous factors, warrant a nuanced understanding that goes beyond generalized assumptions.

1.6 Exercise 2-4

[108]:
```
# Create a DF for the live births from preg

live=preg[preg.outcome==1]

# create a division of the live df to subset first born from other born

first_born=live[live.birthord==1]
other_born=live[live.birthord!=1]
```

[116]:
```
# subselect each fb mean and other mean then print the mean of each in lb's to␣
 ↪determine if first babies are
# lighter or heavier.  The calc shows FB as lighter than other born in terms of␣
 ↪the mean.

fb_mean=first_born.totalwgt_lb.mean()
other_mean=other_born.totalwgt_lb.mean()
print(f"first born: {fb_mean} mean lb's")
print(f"other born: {other_mean} mean lb's")
```

```
first born: 7.201094430437772 mean lb's
other born: 7.325855614973262 mean lb's
```

[117]:
```
# define Cohen's d effect

#import library

```

```
import numpy as np

def CohenEffectSize(group1, group2):
    """Computes Cohen's effect size for two groups.

    group1: Series or DataFrame
    group2: Series or DataFrame

    returns: float if the arguments are Series;
             Series if the arguments are DataFrames
    """
    diff = group1.mean() - group2.mean()

    var1 = group1.var()
    var2 = group2.var()
    n1, n2 = len(group1), len(group2)

    pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
    d = diff / np.sqrt(pooled_var)
    return d
```

[118]:
```
# Calculate Cohen's d to compare the first born birth weight in lb's to other␣
 ↪born birth weights in lb's.

CohenEffectSize(first_born.totalwgt_lb,other_born.totalwgt_lb)
```

[118]: -0.088672927072602

[120]:
```
# subselect each fb length mean and other mean then print the mean of each in␣
 ↪length to determine if first babies are
# longer or shorter.  The calc shows FB as shorter than other born in terms of␣
 ↪the mean.

fb_ln_mean=first_born.prglngth.mean()
other_ln_mean=other_born.prglngth.mean()
print(f"first born length: {fb_mean} mean")
print(f"other born length: {other_mean} mean")
```

```
first born length: 7.201094430437772 mean
other born length: 7.325855614973262 mean
```

[119]:
```
# Calculate Cohen's d to compare the first born birth length to other born␣
 ↪birth length.

CohenEffectSize(first_born.prglngth,other_born.prglngth)
```

[119]: 0.02887904465444988

1.7 Response

The primary difference between the two Cohen's d values is their magnitude and the direction of the effect. The first value (−0.08867) (Birth weight) suggests a slightly larger effect size in the negative direction, while the second value (0.028879) (Birth length) suggests a smaller effect size in the positive direction. The negative Birth weight effect suggests that first born babies are lighter that of other born the low negative effect suggests only a small differnce in the mean of first born, The positive, yet smaller effect suggests there is less of a mean difference in the first born length compared to the other born length.

[ ]: