Dave530Week4

December 19, 2024

[1]:

```
# 4.2 Exercise: Probability Mass Functions and Cumulative Distribution Functions
```

[2]:

```
import warnings
# Suppress all warnings
warnings.filterwarnings("ignore")
```

0.1 Exercises Page 35-36: 3-1

0.2 Exercises

Exercise: Something like the class size paradox appears if you survey children and ask how many children are in their family. Families with many children are more likely to appear in your sample, and families with no children have no chance to be in the sample.

Use the NSFG respondent variable `numkdhh` to construct the actual distribution for the number of children under 18 in the respondents' households.

Now compute the biased distribution we would see if we surveyed the children and asked them how many children under 18 (including themselves) are in their household.

Plot the actual and biased distributions, and compute their means.

[6]:

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

[7]:
```
import nsfg
import first
import thinkstats2
import thinkplot
```

[8]:
```
resp = nsfg.ReadFemResp()
```

[9]:
```
# Read first 5 records
resp.head()
```

[9]:
```
   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]
```

[10]:
```
# Inspect column names
resp.columns
```

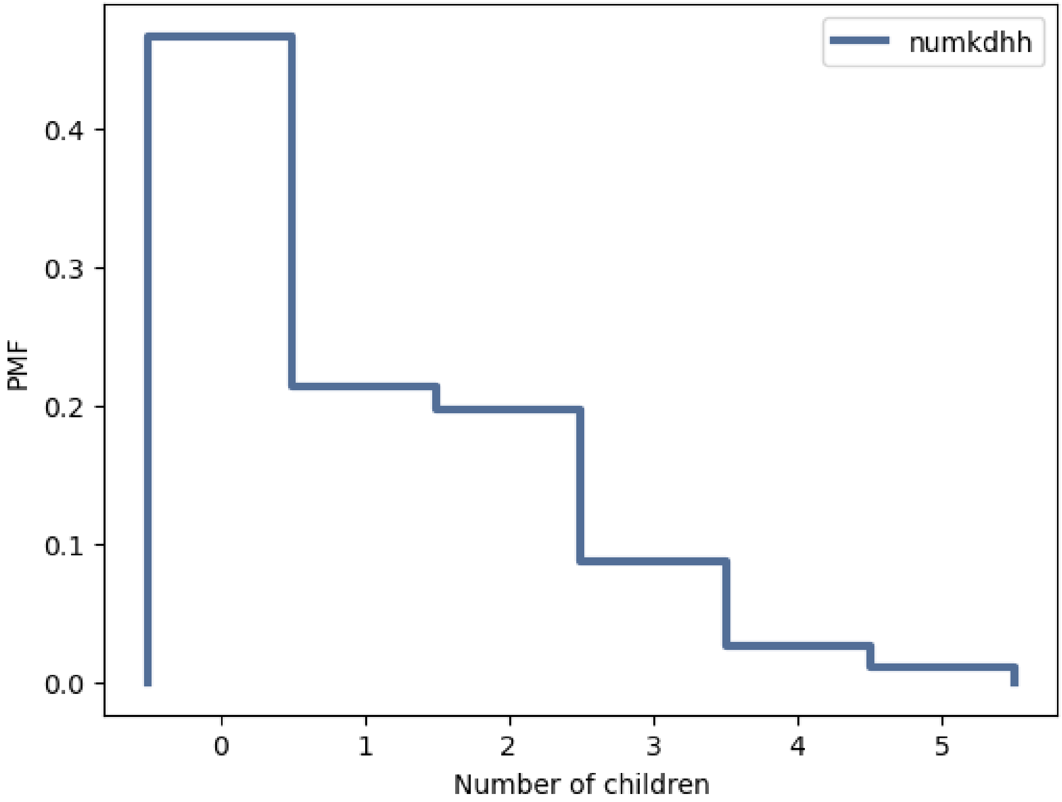
[10]:
```
Index(['caseid', 'rscrinf', 'rdormres', 'rostscrn', 'rscreenhisp',
       'rscreenrace', 'age_a', 'age_r', 'cmbirth', 'agescrn',
       …
       'pubassis_i', 'basewgt', 'adj_mod_basewgt', 'finalwgt', 'secu_r',
       'sest', 'cmintvw', 'cmlstyr', 'screentime', 'intvlngth'],
      dtype='object', length=3087)
```

[11]:

```
# Use the NSFG respondent variable numkdhh to construct the actual distribution␣
 ↪for the number of children under 18 in the respondents' households.

pmf = thinkstats2.Pmf(resp.numkdhh, label="numkdhh")
```

[12]:

```
# Solution

thinkplot.Pmf(pmf)
thinkplot.Config(xlabel="Number of children", ylabel="PMF")
```



[13]:

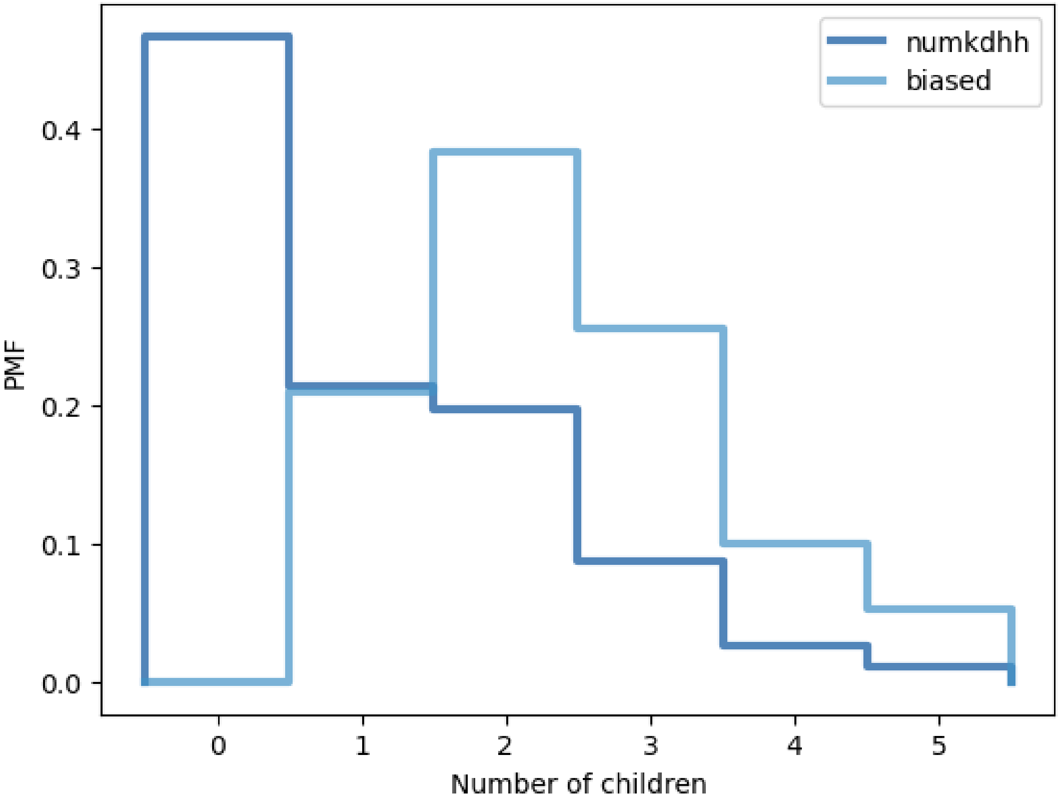
```
#This function computes the biased PMF we would get if we surveyed students and␣
 ↪asked about the size of the classes they are in.
```

[14]:

```
def BiasPmf(pmf, label):
    new_pmf = pmf.Copy(label=label)
    for x, p in pmf.Items():
        new_pmf.Mult(x, x)

    new_pmf.Normalize()
    return new_pmf
```

[15]:
```
# Solution

biased = BiasPmf(pmf, label="biased")
```

[16]:
```
# Plot the actual and biased distributions.

thinkplot.PrePlot(2)
thinkplot.Pmfs([pmf, biased])
thinkplot.Config(xlabel="Number of children", ylabel="PMF")
```



[17]:
```
# compute means for actual number of children

pmf.Mean()
```

[17]:
```
1.024205155043831
```

[18]:

```
pmf.Var()
```

[18]:

```
1.4128643263531195
```

[19]:

```
# compute the mean for biased number of children

biased.Mean()
```

[19]:

```
2.403679100664282
```

0.3 Exercise Page 36: 3-2

**In Chapter 3 we computed the mean of a sample by adding up the elements and dividing by n. If you are given a PMF, you can still compute the mean, but the process is slightly different: %

$$\bar{x} = \sum_i p_i \; x_i$$

% where the x_i are the unique values in the PMF and $p_i = PMF(x_i)$. Similarly, you can compute variance like this: %

$$S^2 = \sum_i p_i \; (x_i - \bar{x})^2$$

% Write functions called `PmfMean` and `PmfVar` that take a Pmf object and compute the mean and variance. To test these methods, check that they are consistent with the methods `Mean` and `Var` provided by `Pmf`.

[22]:

```
def PmfMean(pmf):
    """Computes the mean of a PMF.
    Returns:
        float mean
    """
    return sum(p * x for x, p in pmf.Items())
```

[23]:

```
def PmfVar(pmf, mu=None):
    """Computes the variance of a PMF.
    mu: the point around which the variance is computed;
            if omitted, computes the mean
    returns: float variance
    """
    if mu is None:
        mu = PmfMean(pmf)

    return sum(p * (x - mu) ** 2 for x, p in pmf.Items())
```

[24]:

```
PmfMean(pmf)
```

[24]:

```
1.024205155043831
```

[25]:

```
PmfVar(pmf, mu=None)
```

[25]:

```
1.4128643263531195
```

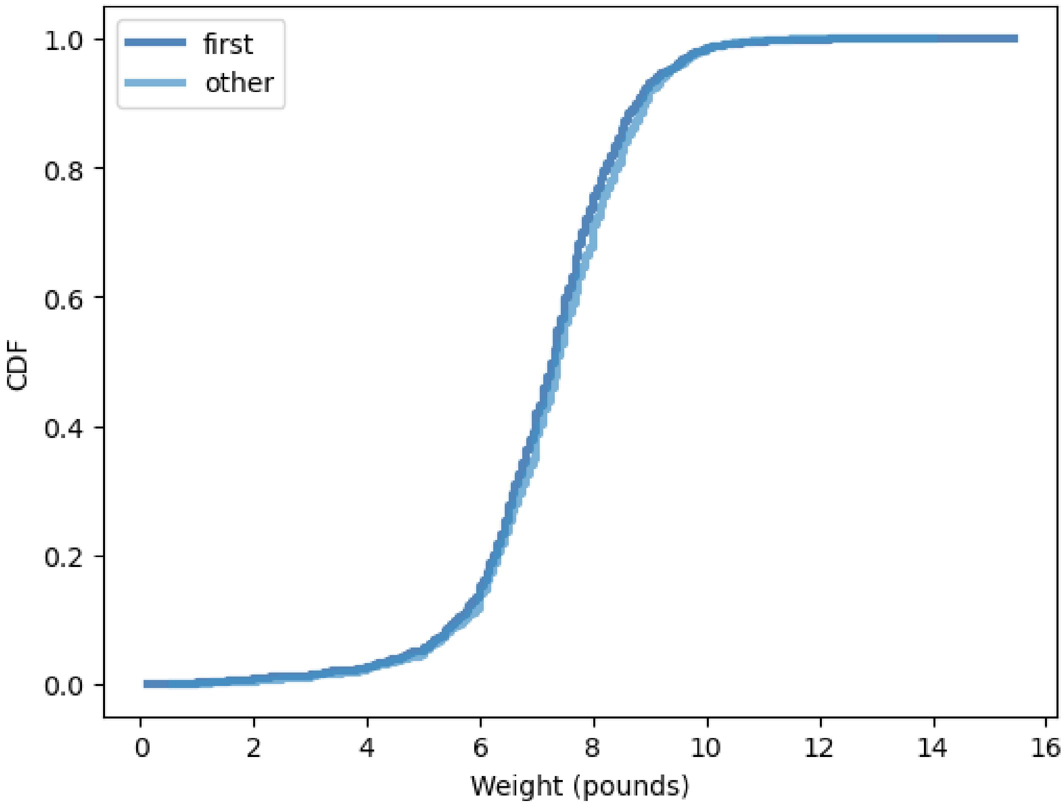
0.4 Exercise Page 47: 4-1

[27]:

```
# Exercise: Page 47: 4-1
# How much did you weigh at birth? If you do not know, call your mother or␣
 ↪someone else who knows.
# Using the NSFG data (all live births), compute the distribution of birth␣
 ↪weights and use it to find your percentile rank.
# If you were a first baby, find your percentile rank in the distribution for␣
 ↪first babies. Otherwise use the distribution for others.
# If you are in the 90 th percentile or higher, call your mother back and␣
 ↪apologize.
```

[28]:

```
import first
import warnings
# Suppress all warnings
warnings.filterwarnings("ignore")
live, firsts, others = first.MakeFrames()
first_cdf = thinkstats2.Cdf(firsts.totalwgt_lb, label='first')
other_cdf = thinkstats2.Cdf(others.totalwgt_lb, label='other')


# Plot the distribution of the birth weights
thinkplot.PrePlot(2)
thinkplot.Cdfs([first_cdf, other_cdf])
thinkplot.Config(xlabel='Weight (pounds)', ylabel='CDF')
```



[29]:
```
# In this example, we can see that first babies are slightly, but consistently,␣
↪lighter than others.

# We can use the CDF of birth weight to compute percentile-based statistics.
```

[30]:
```
# Calculate the percentile rank from cdf of first babies on birth weight␣
↪criteria

first_cdf.PercentileRank(8.5)
```

[30]:
```
85.90419436167774
```

[31]:
```
# Calculate the percentile rank from cdf of other babies on birth weight␣
↪criteria

other_cdf.PercentileRank(8.5)
```

[31]:
```
82.35294117647058
```

[32]:
```
# First babies are slightly lighter throuout the distribution
```

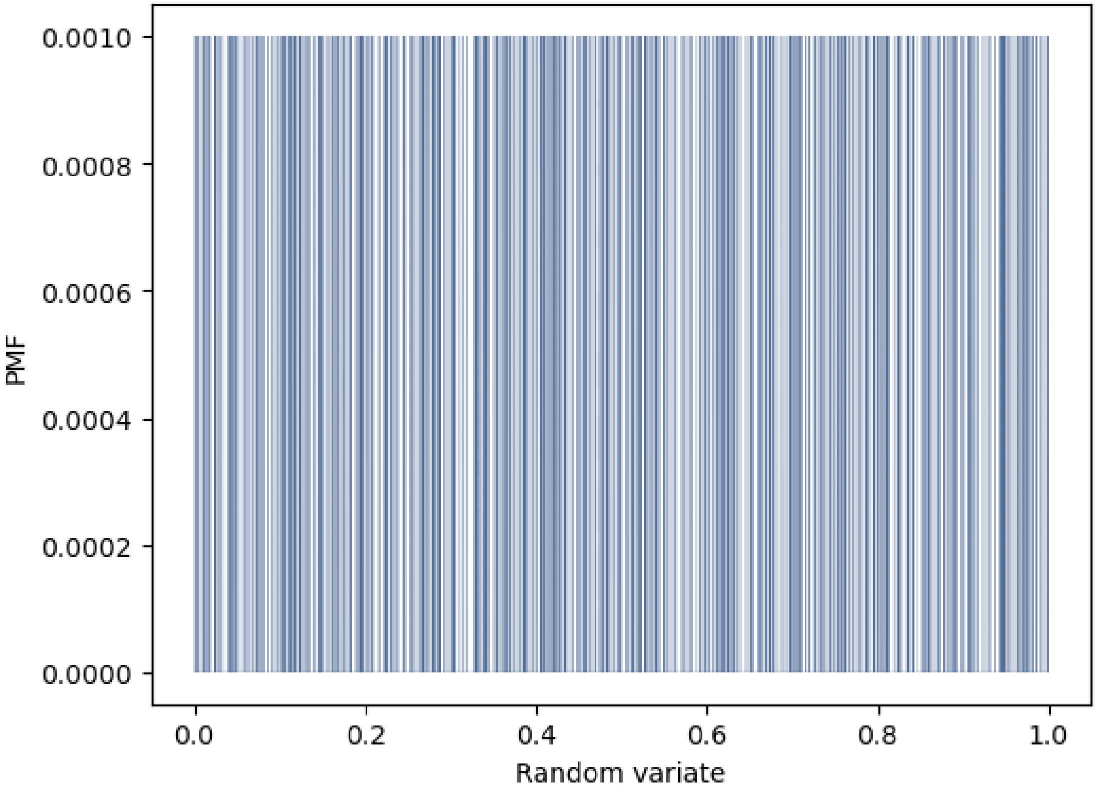
0.5 Exercise Page 48: 4-2

```
[34]: ## Exercise: Page 48: 4-2** The numbers generated by `numpy.random.random` are supposed to be uniform between 0 and 1; that is, every value in the range should have the same probability.

      # Generate 1000 numbers from `numpy.random.random` and plot their PMF.  What goes wrong?

      # Now plot the CDF. Is the distribution uniform?
```

```
[35]: import numpy as np
```

```
[36]: # Generating 1000 random numbers from numpy.random.random

      t = np.random.random(1000)
```

```
[37]: # plot PMF

      pmf = thinkstats2.Pmf(t)
      thinkplot.Pmf(pmf, linewidth=0.1)
      thinkplot.Config(xlabel='Random variate', ylabel='PMF')
```

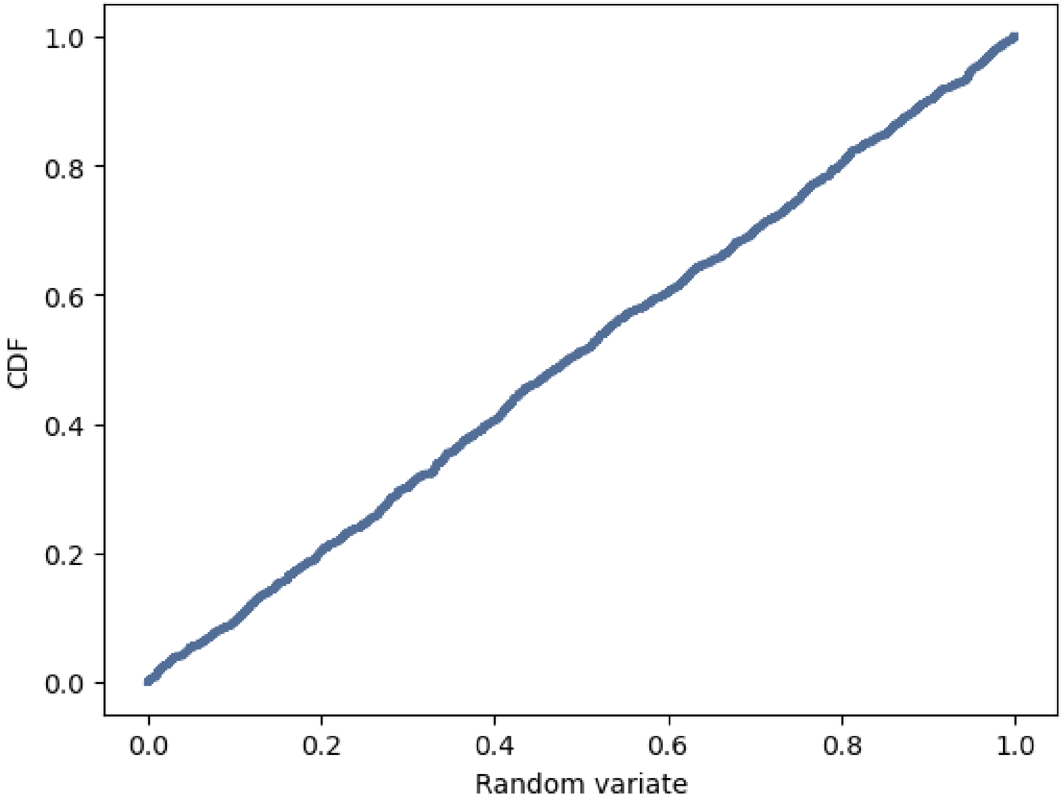


[38]:

```
# PMF plot is hard to interpret from the visualization.
```

[39]:

```
# plot the CDF

cdf = thinkstats2.Cdf(t)
thinkplot.Cdf(cdf)
thinkplot.Config(xlabel='Random variate', ylabel='CDF')
```



[40]:

```
# The CDF approximately is a straight line , which means that the distribution␣
 ↪is uniform.
```