Velikadu_Krishnamoorthy_530Week4

April 9, 2023

1 Guruprasad Velikadu Krishnamoorthy

2 DSC530- Assignment Week 4

2.1 Initial Setup

[1]:
```python
# Download basename and exists from OS module which will be used in the␣
↪download function
from os.path import basename, exists

# Create a function named download_file, to download the scripts and files from␣
↪Github to local path
def download_files(url):
    """
    Downloads the scripts/ files from Github to local directory
    takes url as input.
    """
    filename = basename(url)
    # Checking if the file exists in the local directory and it downloads the␣
↪file if it doesn't exist already.
    if not exists(filename):
        from urllib.request import urlretrieve
        # Downloading the files to the local path
        local, _ = urlretrieve(url, filename)
        # Printing confirmation message
        print("Downloaded " + local)
```

[2]:
```python
#  Calling download functions to download .py files and data files used␣
↪throughtout this assignment
download_files("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
↪2002FemResp.dct")
download_files("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
↪2002FemResp.dat.gz")
download_files("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
↪thinkstats2.py")
download_files("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
↪thinkplot.py")
```

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

[3]:
```
# Importing the nsfg and other modules module from the author's code
import nsfg
import thinkstats2
import thinkplot

# importing the required libraries
import numpy as np
import sys
from collections import defaultdict
import math
import itertools
import pandas as pd
```

[4]:
```
# Setting the maximum number of rows and columns to display
pd.options.display.max_rows=20
pd.options.display.max_columns=20
pd.options.display.precision =3
```

[5]:
```
# Creating Respondents Dataframe
dct_file='2002FemResp.dct'
dat_file='2002FemResp.dat.gz'
nrows = None
# Reading the dictionary file using  thinkstats2 module
dictionary_file = thinkstats2.ReadStataDct(dct_file)
# creating dataframe for the respondents file read and print top 5 lines
respondents_df = dictionary_file.ReadFixedWidth(dat_file, compression='gzip',␣
 ↪nrows=nrows)
respondents_df.head()
```

[5]:
```
   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  3247.917         5123.760
1     42      718       42  …           0  2335.279         2846.799
2     43      708       43  …           0  2335.279         2846.799
```

```
3      15      1042        15  …            0  3783.152         5071.464
4      20       991        20  …            0  5341.330         6437.336

   finalwgt  secu_r  sest  cmintvw  cmlstyr  screentime  intvlngth
0  5556.717       2    18     1234     1222    18:26:36    110.493
1  4744.191       2    18     1233     1221    16:30:59     64.294
2  4744.191       2    18     1234     1222    18:19:09     75.149
3  5923.977       2    18     1234     1222    15:54:43     28.643
4  7229.128       2    18     1233     1221    14:19:44     69.503

[5 rows x 3087 columns]
```

[6]:
```
# Creating Pregnancy Dataframe using the nfsg module
pregnancy_df = nsfg.ReadFemPreg()
# Creating seperate dataframes for Live births, first and other births
live_births = pregnancy_df[pregnancy_df.outcome == 1]
first_births = live_births[live_births.birthord == 1]
other_births = live_births[live_births.birthord != 1]
```

3 Exercise 3.1

Question: 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.

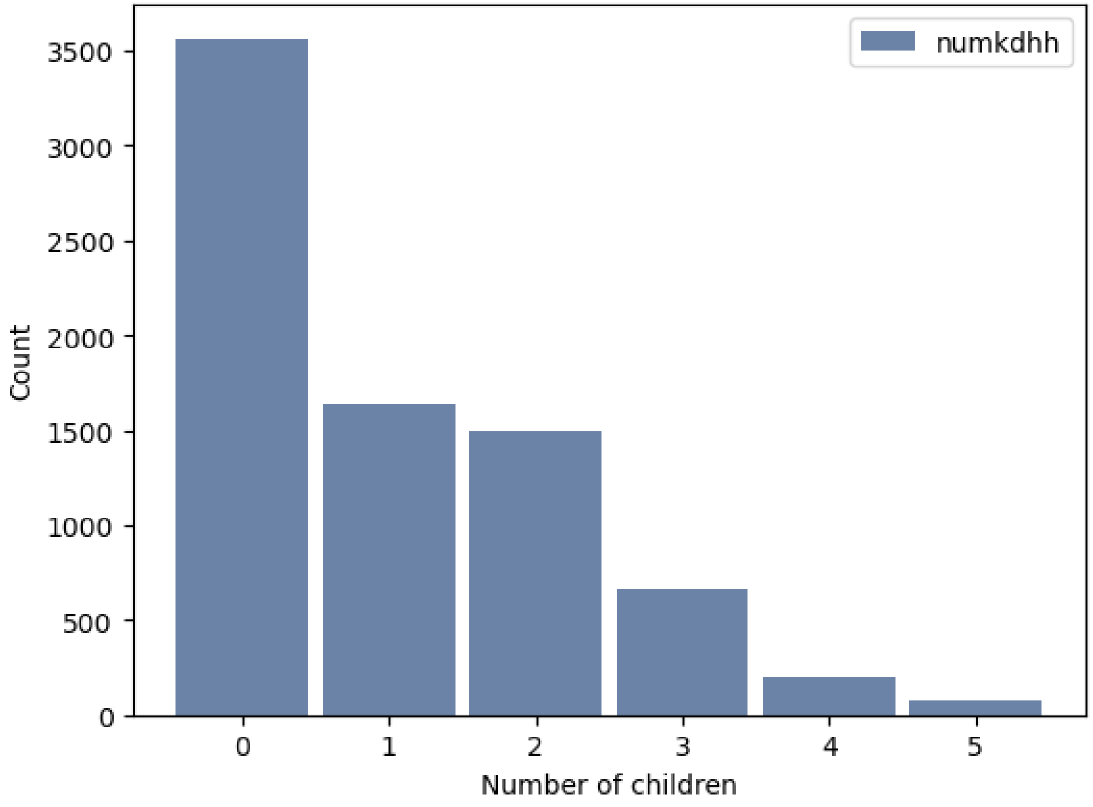
[7]:
```
# creating a histogram on the variable numkdhh from respondents dataframe
numkdhh_hist = thinkstats2.Hist(respondents_df.numkdhh, label="numkdhh")
numkdhh_hist
```

[7]:
```
Hist({0: 3563, 1: 1636, 2: 1500, 3: 666, 4: 196, 5: 82}, 'numkdhh')
```

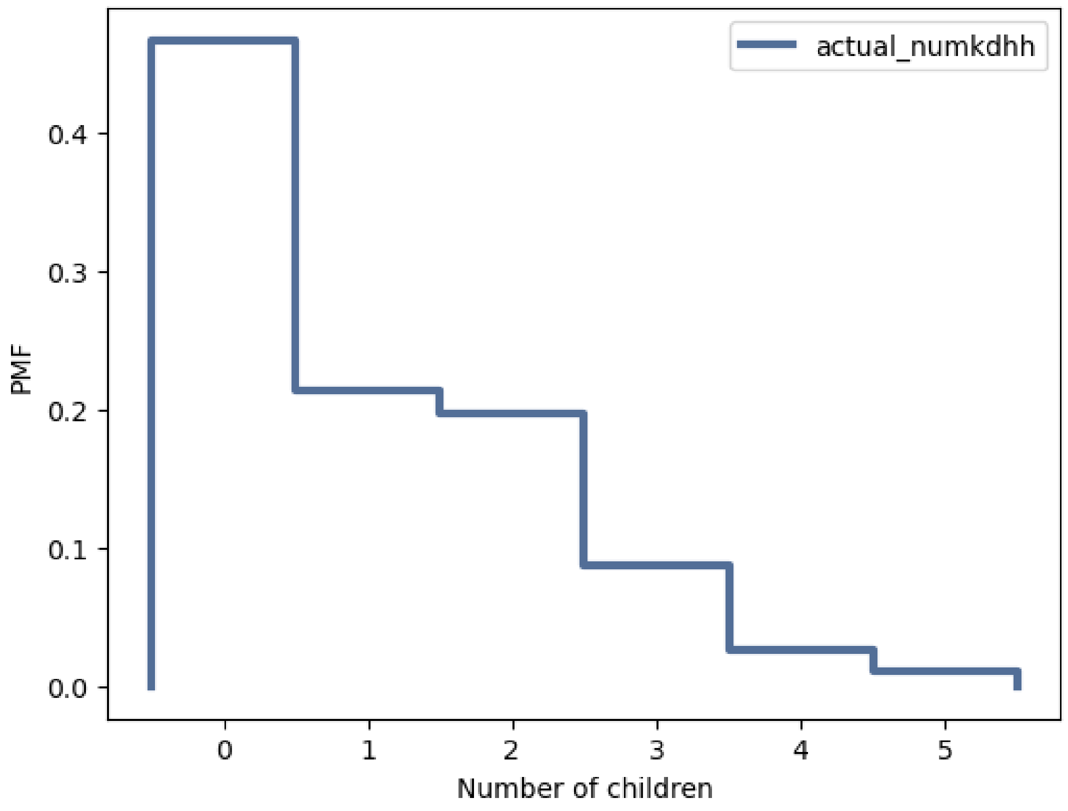
[8]:
```
# creating a Probablity Mass function on the variable numkdhh from respondents␣
↪dataframe
numkdhh_pmf = thinkstats2.Pmf(respondents_df.numkdhh, label="actual_numkdhh")
numkdhh_pmf
```

[8]:
```
Pmf({0: 0.466178202276593, 1: 0.21405207379301322, 2: 0.19625801386889966, 3:
0.08713855815779145, 4: 0.025644380478869556, 5: 0.01072877142483318},
'actual_numkdhh')
```

```
[9]: # Plotting the histogram on variable numkdhh from respondents dataframe
thinkplot.Hist(numkdhh_hist)
thinkplot.Config(xlabel="Number of children", ylabel="Count")
```

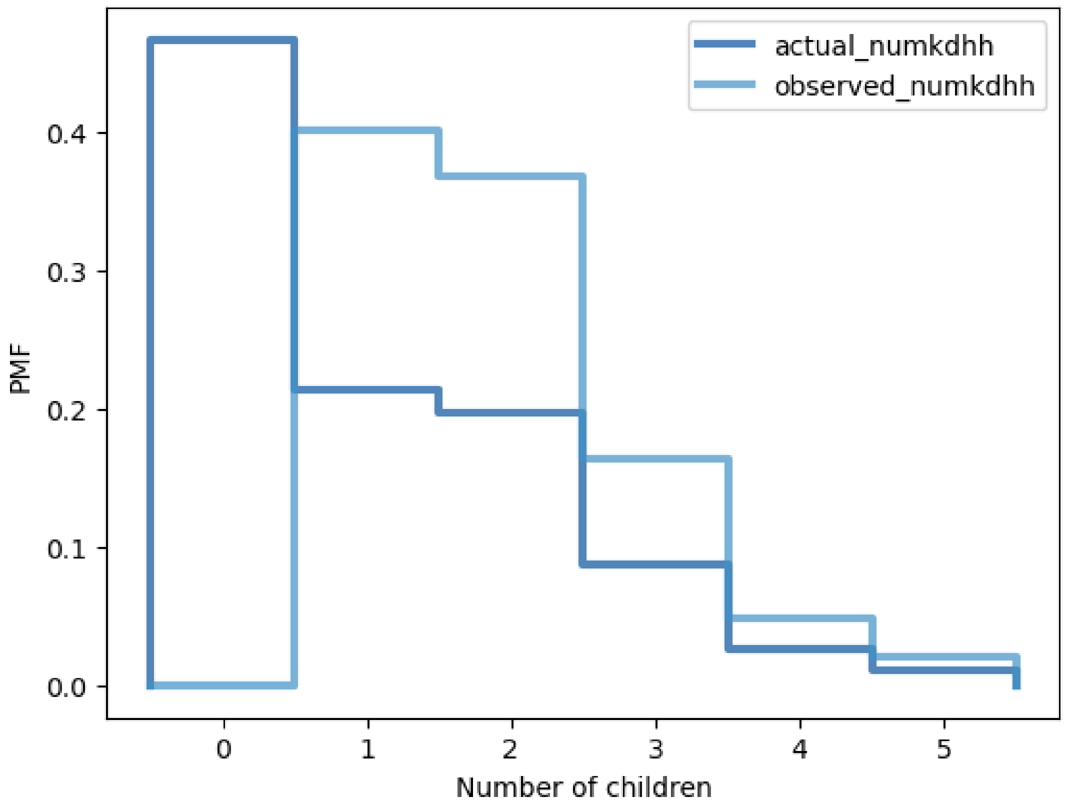


```
[10]: # Plotting the pmf on variable numkdhh from respondents dataframe. This Uses a␣
    ↪smooth plot.
thinkplot.Pmf(numkdhh_pmf)
thinkplot.Config(xlabel="Number of children", ylabel="PMF")
```



```
[11]: # Creating a function to calculate the Biased pmf
      def BiasedPmf(prob_mass_func, label):
              """
              Calculates Biased pmf for the given pmf.
              Takes the existing pmf name and the label for Biased pmf as inputs.
              """
              # Creates a copy of existing pmf and names it as new_prob_mass_func
              new_prob_mass_func = prob_mass_func.Copy(label=label)
              # Iterates through each item in pmf and takes the the item and its␣
        ↪frequency
              for item, prob in prob_mass_func.Items():
              # The probability of each item is multiplies by the value of the item.
              # for example if the item is 2 and if its probablity id 0.2, the new␣
        ↪probablity will be 2 * 0.2= 0.4
                      new_prob_mass_func.Mult(item, item)
              # This will standardize so the sum of all probablities in new_pmf will␣
        ↪be 1 and return the new pmf
                      new_prob_mass_func.Normalize()
                      return new_prob_mass_func
```

[12]:
```
# Calling the biased pmf function for numkdhh_pmf and pass the balel of biased␣
↪pmf as observed_numkdhh
biased_numkdhh_pmf = BiasedPmf(numkdhh_pmf, label="observed_numkdhh")
# This will create 2 graphs for 2 pmfs in the same plot
thinkplot.PrePlot(2)
# Note Pmfs function is used from thinkplot module and not Pmf to plot both␣
↪biased and unbiased pmf
thinkplot.Pmfs([numkdhh_pmf, biased_numkdhh_pmf])
# Adding the labels to the graph
thinkplot.Config(xlabel="Number of children", ylabel="PMF")
```



[13]:
```
# Calculating the mean and variance of Original Unbiased numkdhh_pmf
numkdhh_pmf.Mean(), numkdhh_pmf.Var()
```

[13]:
```
(1.024205155043831, 1.4128643263531195)
```

[14]:
```
# Calculating the mean and variance of biased biased_numkdhh_pmf
biased_numkdhh_pmf.Mean(),biased_numkdhh_pmf.Var()
```

[14]:
```
(1.9186274509803922, 0.9306334102268358)
```

Interpretaion of Class size paradox results: Children experience averages in terms of their experience and not by the averages that is calculated by the mathematics.Hence, to add up each child's experience, we multiply x number of experience for x number of children, we end up in a different mean for the biased pmf. The mean of unbiased and biased pmfs are 1.024 and 1.918 respectively. This means each child in reality believes that there are 1.918 children in their household under 18 on average, while in reality the mathematical mean suggests a different value of 1.024. This difference may be due to the difference in the distribution of children in each household.

4 Exercise 3.2

Question: 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`.

[15]:
```
# Calculate the Mean of Pmf
def PMF_Mean(sample_pmf):
    """Computes the mean of a PMF.
    Returns:
        float mean
    """
    # setting initial value of mean to 0
    mean=0.0
    # Looping through each item and calculating the product of probablity and␣
 ↪the item value
    for item,prob in sample_pmf.Items():
        # Calculating the sum of mean of all values
        mean+=(item*prob)
    return mean
```

[16]:
```
# Calculating the mean of numkdhh_pmf using the function PMF_Mean
PMF_Mean(numkdhh_pmf)
```

[16]:
```
1.024205155043831
```

[17]:
```
# Using assert function to validate the value of mean caculated
assert PMF_Mean(numkdhh_pmf) == numkdhh_pmf.Mean()
```

[18]:
```
def PMF_Variance(sample_pmf,mean=None):
    """Computes the variance of a PMF.
    mean: the point around which the variance is computed;
            if omitted, computes the mean
    returns: float variance
    """
    # setting initial value of variance to 0
    variance=0.0
    # If mean value is not provided it will calculate the mean using PMF_Mean␣
 ↪function
    if mean is None:
        mean=PMF_Mean(sample_pmf)
    # Looping through each item and calculating the variance using the formula␣
 ↪given in the question
    for item,prob in sample_pmf.Items():
        # Calculating the sum of variance of all values
        variance+=(prob * (item - mean) ** 2)
    return variance
```

[19]:
```
# Calculating the Variance of numkdhh_pmf using the function PMF_Variance
PMF_Variance(numkdhh_pmf)
```

[19]:
```
1.4128643263531195
```

[20]:
```
# Using assert function to validate the value of Variance caculated
assert PMF_Variance(numkdhh_pmf)==numkdhh_pmf.Var()
```

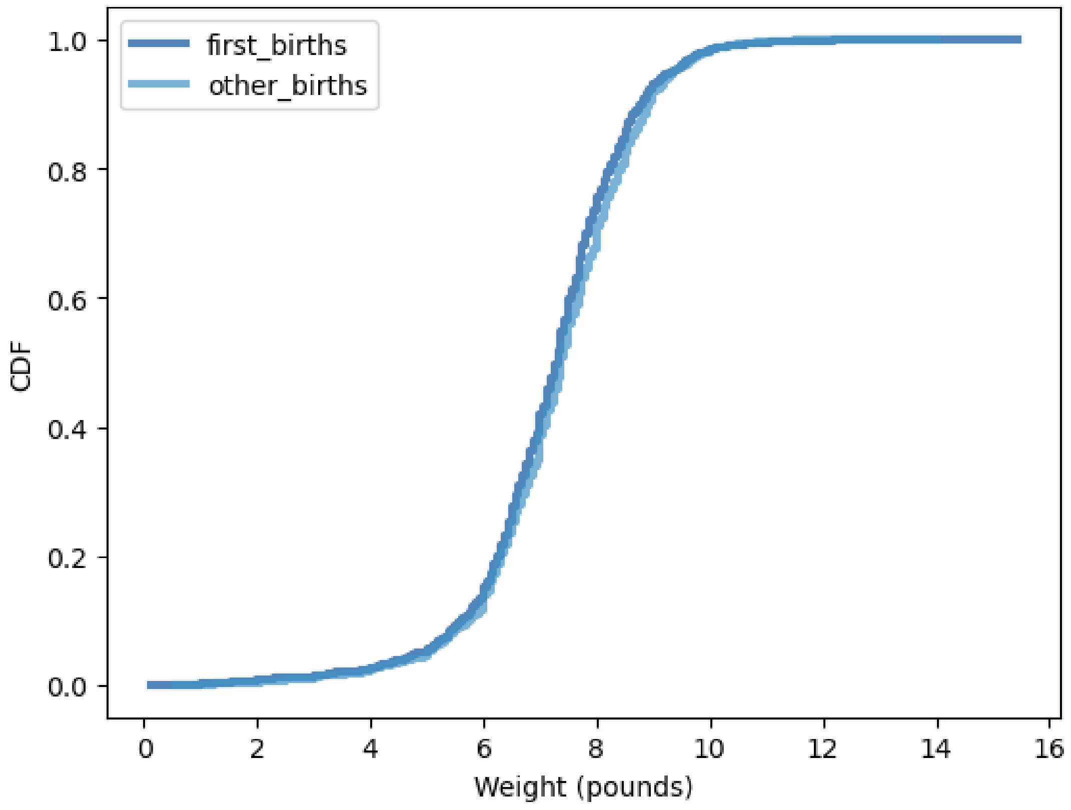
Interpretation of the results: The results of the mean computed by the PMF_Mean and PMF_Variance are consistent with the results from Mean and Var methods provided by Pmf.

5 Exercise 4.1

***Question:*: How much did you weigh at birth? If you don't 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 90th percentile or higher, call your mother back and apologize.**

[21]:
```
# Using CDF function in thinkstats2 module to calculate the Cumulative␣
 ↪distribution function.
# Using first births dataframe and other births dataframe
first_births_cdf = thinkstats2.Cdf(first_births.totalwgt_lb,␣
 ↪label='first_births')
other_births_cdf = thinkstats2.Cdf(other_births.totalwgt_lb,␣
 ↪label='other_births')
# Creating plot  with 2 graphs
```

```
thinkplot.PrePlot(2)
# using Cdfs function(not cdf) to plot graph with 2 arguments
thinkplot.Cdfs([first_births_cdf, other_births_cdf])
thinkplot.Config(xlabel='Weight (pounds)', ylabel='CDF')
```



[22]:
```
# Printing results from the graph for each percentile
print(
    f"First_births_cdf- Percentile(25) value is {first_births_cdf.
 ↪Percentile(25)}. "
    f"Other_births_cdf- Percentile(25) value is {other_births_cdf.
 ↪Percentile(25)}.")
print(f"First_births_cdf- Percentile(50) value is {first_births_cdf.
 ↪Percentile(50)}. "
    f"Other_births_cdf- Percentile(50) value is {other_births_cdf.
 ↪Percentile(50)}.")
print(f"First_births_cdf- Percentile(75) value is {first_births_cdf.
 ↪Percentile(75)}. "
    f"Other_births_cdf- Percentile(75) value is {other_births_cdf.
 ↪Percentile(75)}.")
```

```
print(f"First_births_cdf- Percentile(90) value is {first_births_cdf.
 ↪Percentile(90)}. "
    f"Other_births_cdf- Percentile(90) value is {other_births_cdf.
 ↪Percentile(90)}.")
```

```
First_births_cdf- Percentile(25) value is 6.4375. Other_births_cdf-
Percentile(25) value is 6.5.
First_births_cdf- Percentile(50) value is 7.3125. Other_births_cdf-
Percentile(50) value is 7.375.
First_births_cdf- Percentile(75) value is 8.0. Other_births_cdf- Percentile(75)
value is 8.1875.
First_births_cdf- Percentile(90) value is 8.8125. Other_births_cdf-
Percentile(90) value is 8.9375.
```

Interpretation of the graph: The above results indicate that the first babies weigh lesser than the other babies. At each percentile value, the weight of first babies vs other babies can be compared.

[23]:
```
# Below is my birth weight(I was the first born) which falls in the␣
 ↪percentileRank of 45.7 %. Luckily I didn't had to apologize to my mother :)
first_births_cdf.PercentileRank(7.14)
```

[23]:
```
45.70249828099931
```

[24]:
```
# As a fun exercise, I asked my mother about my brother's birth weight and␣
 ↪found that he falls in the PercentileRank of 72.36 %
other_births_cdf.PercentileRank(8.10)
```

[24]:
```
72.36363636363636
```

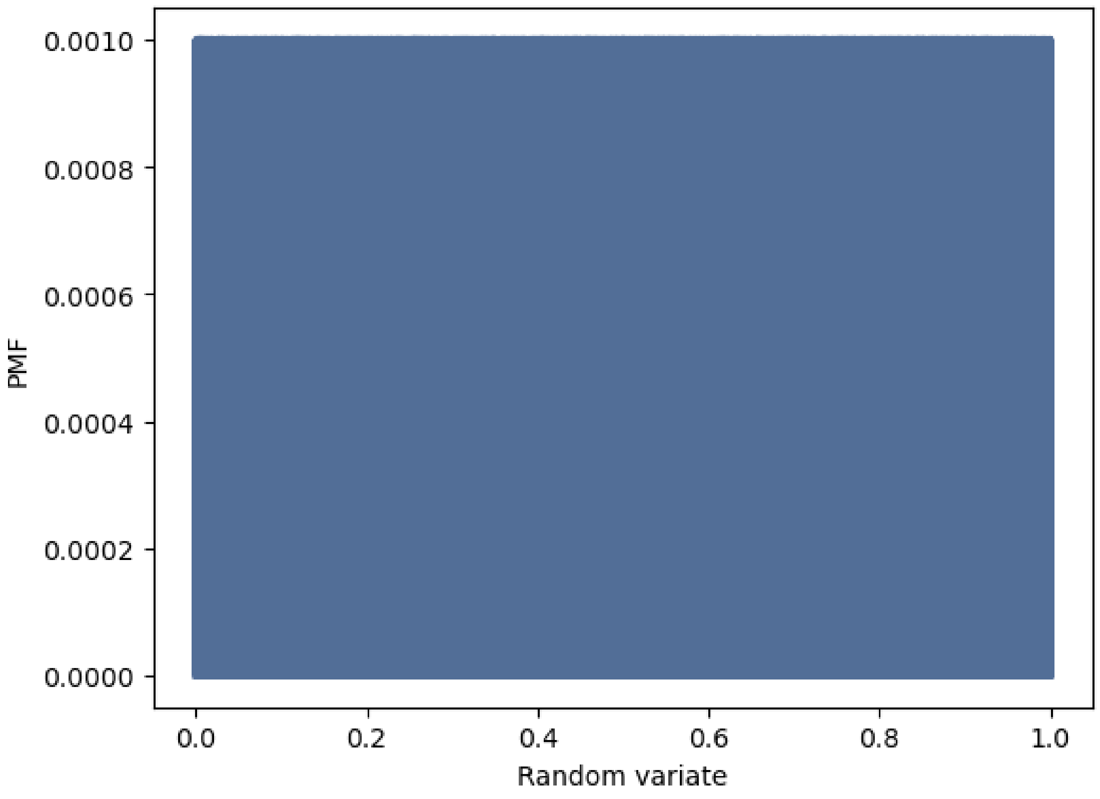
6 Exercise 4.2

***Question*: 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?**

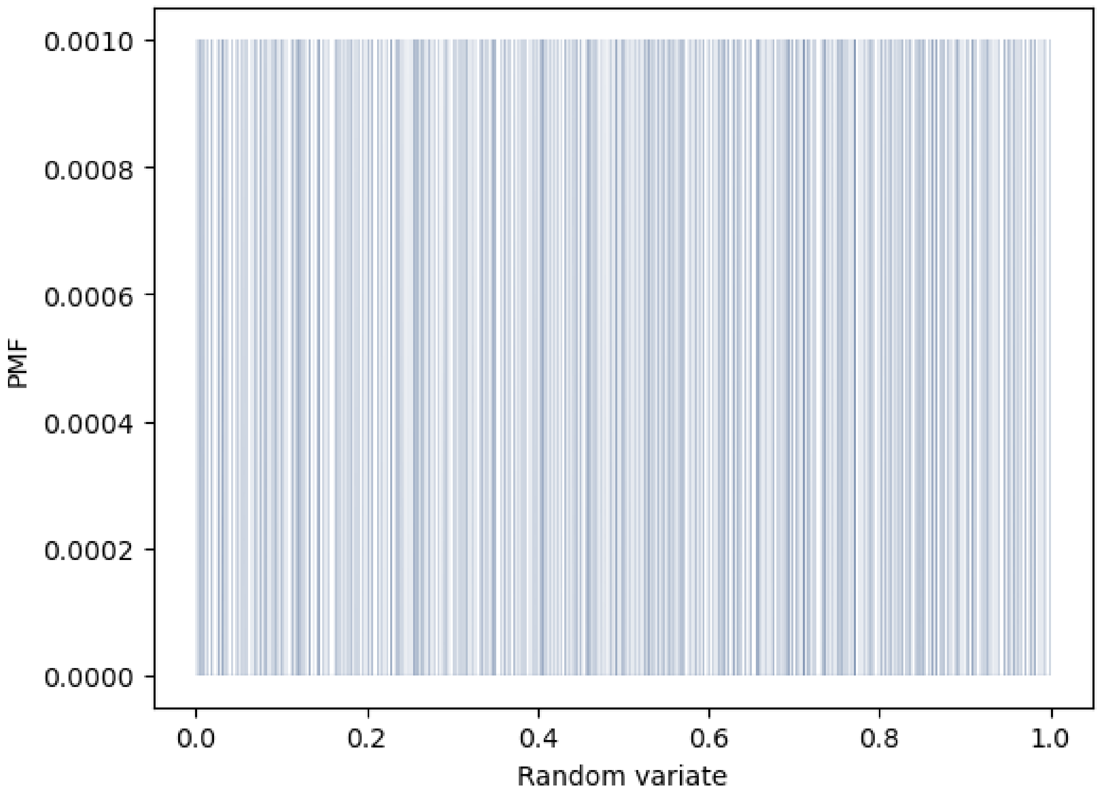
[25]:
```
# Using numpy series to generate 1000 random numbers and printing sample values
# Using set seed function to make sure same numbers are generated each time for␣
 ↪illustration purposes
np.random.seed(40)
series = np.random.random(1000)
series[0:20]
```

[25]:
```
array([0.40768703, 0.05536604, 0.78853488, 0.28730518, 0.45035059,
       0.30391231, 0.52639952, 0.62381221, 0.77677546, 0.68624165,
       0.98093886, 0.60081609, 0.81396852, 0.70864515, 0.02753468,
       0.90426722, 0.44990485, 0.11892465, 0.83530018, 0.20224823])
```

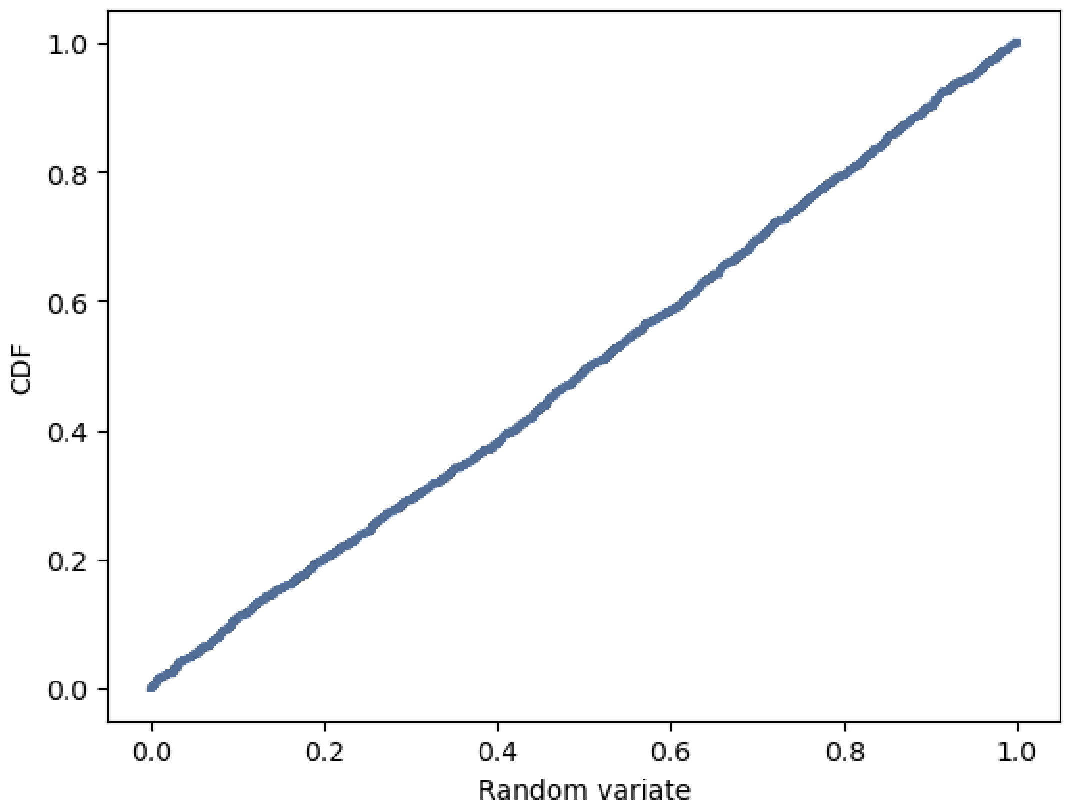
[26]:
```
# Creating a pmf on the 1000 random numbers generated
series_pmf = thinkstats2.Pmf(series)
# Plotting the series_pmf sing thinkplot function
thinkplot.Pmf(series_pmf)
thinkplot.Config(xlabel='Random variate', ylabel='PMF')
# The resulting graph is hard to read as the lines are placed continous, plot␣
↪looks like a blue rectangle.
```



[27]:
```
# To avoid the issue with above graph, we are using a linewidth iof 0.05 to see␣
↪the difference in lines.
thinkplot.Pmf(series_pmf, linewidth=0.05)
thinkplot.Config(xlabel='Random variate', ylabel='PMF')
```



[28]:
```
# Creating a CDF distribution of series
series_cdf = thinkstats2.Cdf(series)
# Plotting the CDF using thinkplot module
thinkplot.Cdf(series_cdf)
thinkplot.Config(xlabel='Random variate', ylabel='CDF')
```



[29]:
```
# Printing the values of Percentile Rank for different values from graph
series_cdf.PercentileRank(0.25),series_cdf.PercentileRank(0.5),series_cdf.
↪PercentileRank(0.75),series_cdf.PercentileRank(0.9)
```

[29]:
```
(24.2, 49.0, 74.5, 90.0)
```

[30]:
```
# Printing the values of Percentiles for different values from graph
series_cdf.Percentile(25),series_cdf.Percentile(50),series_cdf.
↪Percentile(75),series_cdf.Percentile(90)
```

[30]:
```
(0.25563487626961534, 0.5083099649576006, 0.7540292373444691, 0.89806400303675)
```

Interpretation of the Results: **The Results indicate that the distibution is almost uniform. This can be proved from the results of PercentileRank and Percentile functions above. the PercentileRank of 0.25 is 24.2% and PercentileRank of 0.75 is 74.5%. Similarly, the Percentile of Rank 50% is 0.508 and the Percentile of Rank 90% is 0.898. All these values indicate that the distribution is almost uniform**

Also please note that seed function is used so that the same results are generated each time. Else different set of random numbers will be generated for each execution. So in our example, Nothing can go wrong.