Dave530Week5

January 10, 2025

[109]:
```
# 5.2 Exercise: Modeling Distributions and PDFs
```

[110]:
```
import warnings
# Suppress all warnings
warnings.filterwarnings("ignore")
```

0.1 Page 62: 5-1

Exercise: In the BRFSS (In the BRFSS (see "The lognormal Distribution" on page 56), the distribution of heights is roughly normal with parameters µ = 178 cm and = 7.7 cm for men, and µ = 163 cm and = 7.3 cm for women.

In order to join Blue Man Group, you have to be male between 5'10" and 6'1" (see http://bluemancasting.com). What percentage of the U.S. male population is in this range? Hint: use `scipy.stats.norm.cdf`.

[112]:
```
# Download libraries to local
```

[113]:
```
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")
```

[114]:
```
import numpy as np

import thinkstats2
import thinkplot
```

[115]:
```
# `scipy.stats` contains objects that represent analytic distributions, it␣
↪provides a method cfd, that evaluates the standard normal CDF.
```

[116]:
```
import scipy.stats
```

[117]:
```
# For example <tt>scipy.stats.norm</tt> represents a normal distribution.
```

[118]:
```
# This indicates that dist is a "frozen" normal distribution object created␣
↪using scipy.stats.norm.
# A frozen distribution is a distribution object with fixed parameters (mu and␣
↪sigma in this case)
# that allows you to use various methods, such as:

# dist.pdf(x): Compute the probability density function at x.
# dist.cdf(x): Compute the cumulative distribution function at x.
# dist.rvs(size=n): Generate n random samples from the distribution.
```

[119]:
```
mu = 178
sigma = 7.7
dist = scipy.stats.norm(loc=mu, scale=sigma)
type(dist)
```

[119]:
```
scipy.stats._distn_infrastructure.rv_continuous_frozen
```

[120]:
```
# scipy.stats._distn_infrastructure.rv_continuous_frozen is a subclass of scipy.
↪stats._distn_infrastructure.rv_frozen,
# specifically used for frozen continuous probability distributions in the␣
↪scipy.stats module.

# What is rv_continuous_frozen?
# It is a frozen version of a continuous distribution, where the parameters (e.
↪g., loc, scale, and shape parameters)
# are fixed at instantiation. This allows for efficient and reusable␣
↪computation of distribution-specific values such as
# the PDF, CDF, or random sampling.
```

[121]:
```
# scipy.stats._distn_infrastructure.rv_frozen is a class in the scipy.stats␣
↪module that represents a frozen distribution.

# When a distribution is "frozen," it means that the parameters of the␣
↪distribution
# (like the mean loc and standard deviation scale for a normal distribution)␣
↪are fixed, allowing you to work with the
# distribution in a convenient way without repeatedly specifying those␣
↪parameters.
```

[122]:
```
scipy.stats._distn_infrastructure.rv_frozen
```

[122]:
```
scipy.stats._distn_infrastructure.rv_frozen
```

[123]:
```
#A "frozen random variable" can compute its mean and standard deviation.
# dist.mean() This calculates and returns the mean of the frozen distribution.
# dist.std() This calculates and returns the standard deviation of the frozen␣
 ↪distribution.
```

[124]:
```
dist.mean(), dist.std()
```

[124]:
```
(178.0, 7.7)
```

[125]:
```
# For a frozen normal distribution, dist.cdf(mu - sigma) calculates the␣
 ↪cumulative distribution function (CDF) at one standard deviation below the␣
 ↪mean.



# mu=178 (mean)

# sigma =7.7 (standard deviation)
# The CDF gives the probability that a random variable from the distribution is␣
 ↪less than or equal to a given value.
# So, dist.cdf(mu - sigma) is the probability that a random variable is less␣
 ↪than or equal to
```

[126]:
```
dist.cdf(mu - sigma)
```

[126]:
```
0.1586552539314574
```

[127]:
```
# This reflects that about 15.87% of the data lies below one standard deviation␣
 ↪below the mean in a normal distribution.
```

[128]:
```
# How many people are between 5'10" and 6'1"?
```

[129]:
```
# Solution

low = dist.cdf(177.8)  # 5'10" convert height to cm
high = dist.cdf(185.4)  # 6'1" convert height to cm
low, high, high - low
```

[129]:
```
(0.48963902786483265, 0.8317337108107857, 0.3420946829459531)
```

[130]:
```
# CDF at 177.8 cm: Around 0.4974 (close to 50%)
# CDF at 185.4 cm: Around 0.8317.

# This means that about 34.21% of the population's heights lie between 177.8␣
 ↪and 185.4
```

0.1.1 Page 63: 5-2

Exercise: To get a feel for the Pareto distribution, let's see how different the world would be if the distribution of human height were Pareto. With the parameters xm = 1 m and = 1.7, we get a distribution with a reasonable minimum, 1 m, and median, 1.5 m.

Plot this distribution. What is the mean human height in Pareto world? What fraction of the population is shorter than the mean? If there are 7 billion people in Pareto world, how many do we expect to be taller than 1 km? How tall do we expect the tallest person to be?

scipy.stats.pareto represents a pareto distribution. In Pareto world, the distribution of human heights has parameters alpha=1.7 and xmin=1 meter. So the shortest person is 100 cm and the median is 150.

```
[132]: # The median of a Pareto distribution can be calculated directly using its␣
         ↪mathematical properties or using the median()
       # method of a frozen Pareto distribution object in SciPy.
```

```
[133]: # Parameters for the Pareto distribution
       alpha = 1.7 # Shape parameter
       xmin = 1   # Scale parameter (minimum value)

       ## Create the frozen Pareto distribution
       dist = scipy.stats.pareto(b=alpha, scale=xmin)
       dist.median()
```

```
[133]: 1.5034066538560549
```

```
[134]: # What is the mean height in Pareto world?
```

```
[135]: # Solution
       dist.mean()
```

```
[135]: 2.428571428571429
```

```
[136]: # What fraction of people are shorter than the mean?
```

```
[137]: # Solution
       # To calculate you are determining the cumulative probability (CDF) of the mean␣
         ↪value for the Pareto distribution.
       # This gives the probability that a randomly chosen value from the distribution␣
         ↪is less than or equal to the mean.
       dist.cdf(dist.mean())
```

```
[137]: 0.778739697565288
```

```
[138]: # Thus, the cumulative probability meaning approximately 77.9% of the␣
         ↪distribution lies below the mean.
```

```
[139]: # Out of 7 billion people, how many do we expect to be taller than 1 km?  You␣
        ↪could use <tt>dist.cdf</tt> or <tt>dist.sf</tt>.
```

```
[140]: # Solution
       # Global population 7e9
       # Calculate SF Survival function at 1000 meters
       (1 - dist.cdf(1000)) * 7e9, dist.sf(1000) * 7e9
```

```
[140]: (55602.976430479954, 55602.97643069972)
```

```
[141]: # How tall do we expect the tallest person to be?
```

```
[142]: # Solution

       # One way to solve this is to search for a height that we
       # expect one person out of 7 billion to exceed.

       # It comes in at roughly 600 kilometers.

       dist.sf(600000) * 7e9
```

```
[142]: 1.0525455861201714
```

```
[143]: # Solution

       # Another way is to use `ppf`, which evaluates the "percent point function",␣
        ↪which
       # is the inverse CDF.  So we can compute the height in meters that corresponds␣
        ↪to
       # the probability (1 - 1/7e9).

       dist.ppf(1 - 1 / 7e9)
```

```
[143]: 618349.6106759505
```

0.2 Page 75-76: 6-1

Exercise: The distribution of income is famously skewed to the right. In this exercise, we'll measure how strong that skew is. The Current Population Survey (CPS) is a joint effort of the Bureau of Labor Statistics and the Census Bureau to study income and related variables. Data collected in 2013 is available from http://www.census.gov/hhes/www/cpstables/032013/hhinc/toc.htm. I downloaded hinc06.xls, which is an Excel spreadsheet with information about household income, and converted it to hinc06.csv, a CSV file you will find in the repository for this book. You will also find hinc2.py, which reads this file and transforms the data.

The dataset is in the form of a series of income ranges and the number of respondents who fell in each range. The lowest range includes respondents who reported annual household income "Under 250,000 or more."

To estimate mean and other statistics from these data, we have to make some assumptions about the lower and upper bounds, and how the values are distributed in each range. hinc2.py provides InterpolateSample, which shows one way to model this data. It takes a DataFrame with a column, income, that contains the upper bound of each range, and freq, which contains the number of respondents in each frame.

It also takes log_upper, which is an assumed upper bound on the highest range, expressed in log10 dollars. The default value, log_upper=6.0 represents the assumption that the largest income among the respondents is , or one million dollars.

InterpolateSample generates a pseudo-sample; that is, a sample of household incomes that yields the same number of respondents in each range as the actual data. It assumes that incomes in each range are equally spaced on a log10 scale.

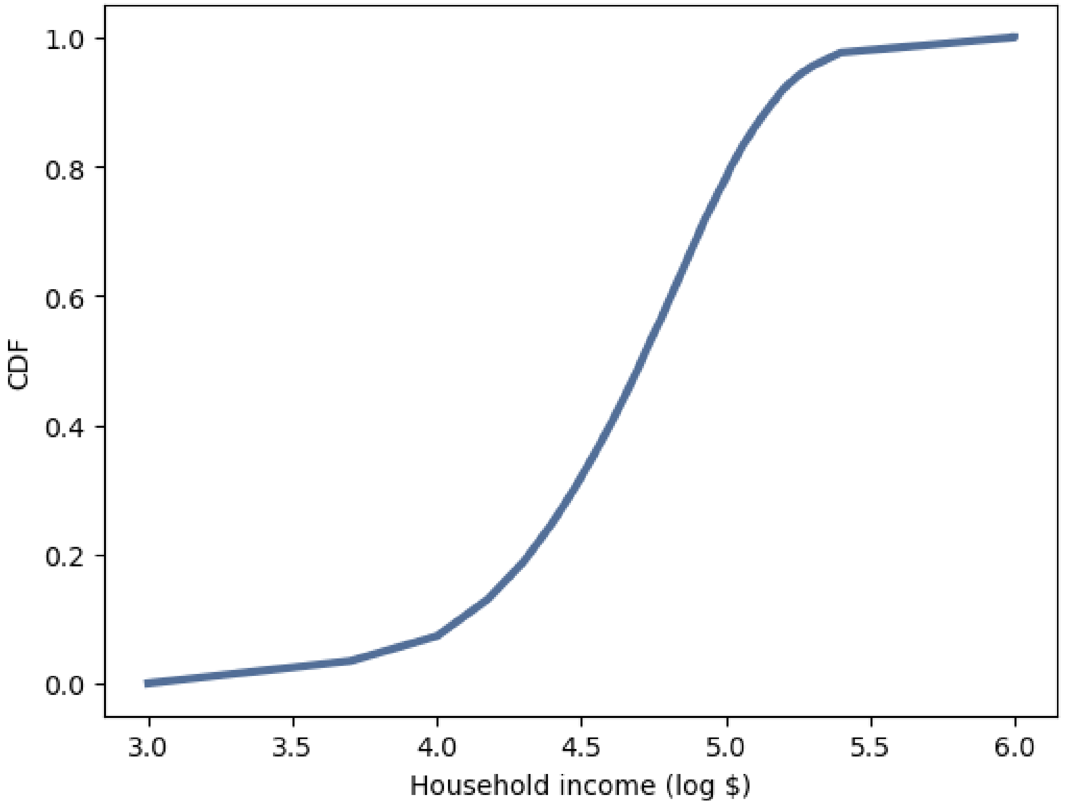
[145]:
```
def InterpolateSample(df, log_upper=6.0):
    """Makes a sample of log10 household income.

    Assumes that log10 income is uniform in each range.

    df: DataFrame with columns income and freq
    log_upper: log10 of the assumed upper bound for the highest range

    returns: NumPy array of log10 household income
    """
    # compute the log10 of the upper bound for each range
    df['log_upper'] = np.log10(df.income)

    # get the lower bounds by shifting the upper bound and filling in
    # the first element
    df['log_lower'] = df.log_upper.shift(1)
    df.loc[0, 'log_lower'] = 3.0

    # plug in a value for the unknown upper bound of the highest range
    df.loc[41, 'log_upper'] = log_upper

    # use the freq column to generate the right number of values in
    # each range
    arrays = []
    for _, row in df.iterrows():
        vals = np.linspace(row.log_lower, row.log_upper, int(row.freq))
        arrays.append(vals)

    # collect the arrays into a single sample
    log_sample = np.concatenate(arrays)
    return log_sample
```

[146]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/hinc.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/hinc06.
 ↪csv")
```

[147]:
```
import hinc
income_df = hinc.ReadData()
```

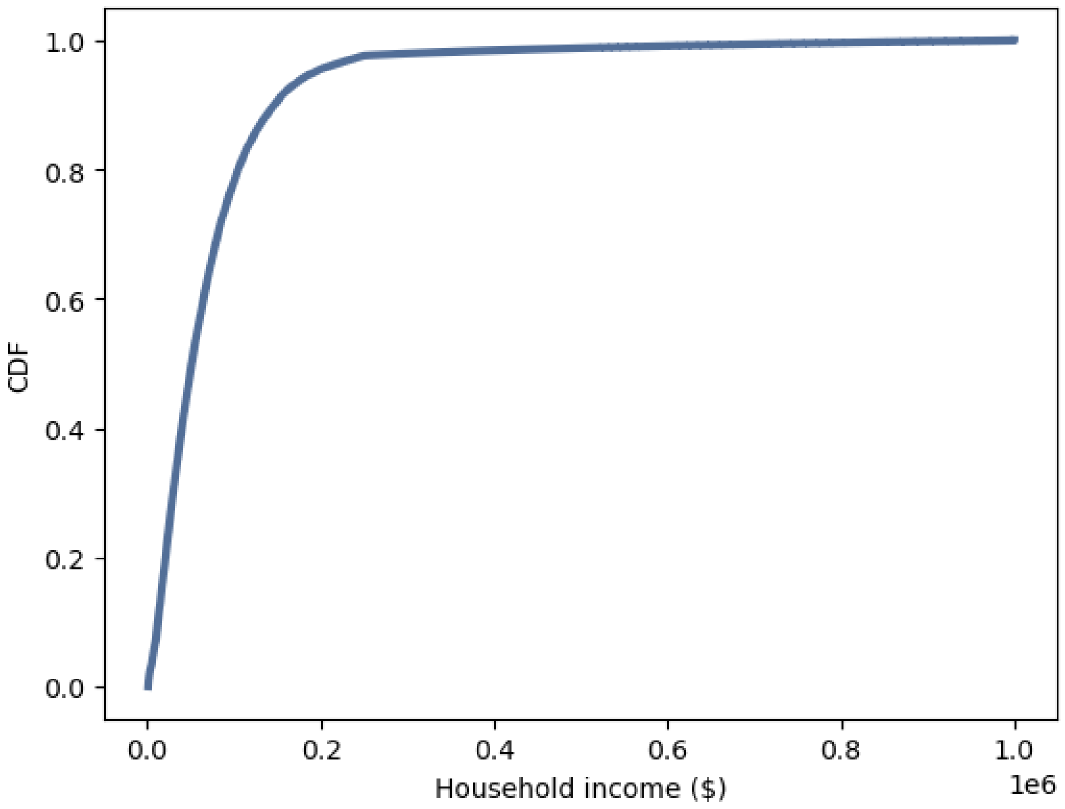
[148]:
```
log_sample = InterpolateSample(income_df, log_upper=6.0)
```

[149]:
```
log_cdf = thinkstats2.Cdf(log_sample)
thinkplot.Cdf(log_cdf)
thinkplot.Config(xlabel='Household income (log $)',
                 ylabel='CDF')
```



[150]:
```
sample = np.power(10, log_sample)
```

[151]:
```
cdf = thinkstats2.Cdf(sample)
thinkplot.Cdf(cdf)
thinkplot.Config(xlabel='Household income ($)',
                 ylabel='CDF')
```



[152]:
```
# Compute the median, mean, skewness and Pearson's skewness of the resulting␣
  ↪sample.
# What fraction of households report a taxable income below the mean? How do␣
  ↪the results depend on the assumed upper bound?
```

[153]:
```
# Raw moments are just sums of powers.
def RawMoment(xs, k):
    return sum(x**k for x in xs) / len(xs)
```

[154]:
```
def Mean(xs):
    return RawMoment(xs, 1)
```

[155]:
```
def StandardizedMoment(xs, k):
    var = CentralMoment(xs, 2)
    std = np.sqrt(var)
    return CentralMoment(xs, k) / std**k
```

[156]:
```
def Skewness(xs):
    return StandardizedMoment(xs, 3)
```

```
[157]: def Median(xs):
           cdf = thinkstats2.Cdf(xs)
           return cdf.Value(0.5)
```

```
[158]: # Solution

       Mean(sample), Median(sample)
```

```
[158]: (74278.70753118733, 51226.45447894046)
```

```
[159]: # The central moments are powers of distances from the mean.
       def CentralMoment(xs, k):
           mean = RawMoment(xs, 1)
           return sum((x - mean)**k for x in xs) / len(xs)
```

```
[160]: def PearsonMedianSkewness(xs):
           median = Median(xs)
           mean = RawMoment(xs, 1)
           var = CentralMoment(xs, 2)
           std = np.sqrt(var)
           gp = 3 * (mean - median) / std
           return gp
```

```
[161]: Skewness(sample), PearsonMedianSkewness(sample)
```

```
[161]: (4.949920244429583, 0.7361258019141782)
```

```
[162]: # Solution

       # About 66% of the population makes less than the mean

       cdf.Prob(Mean(sample))
```

```
[162]: 0.660005879566872
```

```
[163]: # All of this is based on an assumption that the highest income is one million␣
        ↪dollars, but that's certainly not correct. What happens to the skew if the␣
        ↪upper bound is 10 million?

       #Without better information about the top of this distribution, we can't say␣
        ↪much about the skewness of the distribution.
```

```
[ ]:
```