Examples and Exercises from Think Stats, 2nd Edition

http://thinkstats2.com

Exercises

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 $5000." The highest range includes respondents who made "$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 10^6, 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.

In [87]:

```
from __future__ import print_function, division

%matplotlib inline

import numpy as np

import nsfg
import first
import analytic
```

```
import thinkstats2
import thinkplot
```

In [88]:

```
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
```

In [89]:

```
def RawMoment(xs, k):
    return sum(x**k for x in xs) / len(xs)

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

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

def Median(xs):
    cdf = thinkstats2.Cdf(xs)
    return cdf.Value(0.5)

def Skewness(xs):
    return StandardizedMoment(xs, 3)

def Mean(xs):
    return RawMoment(xs, 1)
```

```
def CentralMoment(xs, k):
    mean = RawMoment(xs, 1)
    return sum((x - mean)**k for x in xs) / len(xs)
```

In [90]:
```
import hinc
income_df = hinc.ReadData()
```

In [91]:
```
log_sample = InterpolateSample(income_df, log_upper=6.0)
```

In [92]:
```
log_cdf = thinkstats2.Cdf(log_sample)
```

In [93]:
```
sample = np.power(10, log_sample)
```

In [94]:
```
cdf = thinkstats2.Cdf(sample)
```

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?

In [95]:
```
Mean(sample)
```

Out[95]: 74278.70753118733

In [96]:
```
Median(sample)
```

Out[96]: 51226.45447894046

In [97]:
```
Skewness(sample)
```

Out[97]: 4.949920244429583

In [98]:
```
PearsonMedianSkewness(sample)
```

Out[98]: 0.7361258019141782

In [99]:
```
cdf.Prob(Mean(sample))
```

Out[99]: 0.660005879566872

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.

Examples and Exercises from Think Stats, 2nd Edition

http://thinkstats2.com

Exercises

Exercise: In the BRFSS (see Section 5.4), the distribution of heights is roughly normal with parameters μ = 178 cm and σ = 7.7 cm for men, and μ = 163 cm and σ = 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` .

`scipy.stats` contains objects that represent analytic distributions

In [1]:
```
import scipy.stats
```

For example `scipy.stats.norm` represents a normal distribution.

In [2]:
```
mu = 178
sigma = 7.7
dist = scipy.stats.norm(loc=mu, scale=sigma)
type(dist)
```

Out[2]:
```
scipy.stats._distn_infrastructure.rv_frozen
```

A "frozen random variable" can compute its mean and standard deviation.

In [3]:
```
dist.mean(), dist.std()
```

Out[3]:
```
(178.0, 7.7)
```

It can also evaluate its CDF. How many people are more than one standard deviation below the mean? About 16%

In [4]:
```
dist.cdf(mu-sigma)
```

Out[4]:
```
0.1586552539314574
```

How many people are between 5'10" and 6'1"?

In [7]:
```
# Find 5'10

low = dist.cdf(177.8)

# Find 6'1
high = dist.cdf(185.4)
```

```
low, high, high-low
```

Out[7]: (0.48963902786483265, 0.8317337108107857, 0.3420946829459531)

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 α = 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.

In [11]:
```
alpha = 1.7
xmin = 1       # meter
dist = scipy.stats.pareto(b=alpha, scale=xmin)
dist.median()
```

Out[11]: 1.5034066538560549

What is the mean height in Pareto world?

In [12]:
```
dist.mean()
```

Out[12]: 2.428571428571429

What fraction of people are shorter than the mean?

In [13]:
```
(1 - dist.cdf(1000)) * 7e9, dist.sf(1000) * 7e9
```

Out[13]: (55602.976430479954, 55602.97643069972)

Out of 7 billion people, how many do we expect to be taller than 1 km? You could use `dist.cdf` or `dist.sf`.

In [14]:
```
# looking for a height out of 7 billion people

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

Out[14]: 1.0525455861201714

How tall do we expect the tallest person to be?

In [15]:
```
# compute the height that corresponds to the probabilitiy

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

Out[15]: 618349.6106759505