Reppeto530Week5

January 14, 2024

Chapter 5-1 Brian Reppeto 530 Prof. Jim Week 5 HW

[172]:
```
# import the scipy stat module from the Scipy library

import scipy.stats
```

[173]:
```
# The normal distribution of height for men

mu = 178             # the mean height for men,
sigma = 7.7          # Standard deviation of height
dist = scipy.stats.norm(loc=mu, scale=sigma)  # create a normal distribution␣
 ↪object
type(dist)                                    # print the type of the dist␣
 ↪object,
```

[173]:
```
scipy.stats._distn_infrastructure.rv_continuous_frozen
```

[174]:
```
# returns the mean & standard dev of the normal distribution

dist.mean(), dist.std()
```

[174]:
```
(178.0, 7.7)
```

[175]:
```
# calculate the CDF of the normal distribution

dist.cdf(mu - sigma)
```

[175]:
```
0.15865525393145741
```

[176]:
```
#



# calc the CDF value for the low point in the normal distribution
low = dist.cdf(177.8)

# calc the CDF value for the high point in the normal distribution
high = dist.cdf(185.4)
```

```
# calc the probability that a random variable is less than or equal to 177.8␣
↪for low
# calc the probability that a random variable is less than or equal to 185.4␣
↪for high
# high - low calculates the probability of the random variable falling between␣
↪177.8 and 185.4

low, high, high - low
```

[176]: (0.48963902786483265, 0.8317337108107857, 0.3420946829459531)

The percentage of the U.S. Male population that falls into the Blue Man Group range is 34.2%

0.0.1 Exercise 5-2

[177]:
```
# Calc pareto dist with (alpha and xmin) and then calc and print the median of␣
↪the distribution in meters

alpha = 1.7          #  Alph of the pareto dist to 1.7
xmin = 1             # min  of the pareto dist to 1
dist = scipy.stats.pareto(b=alpha, scale=xmin)  # pareto dist using the␣
↪parameters alpha and xmin
dist.median()  # calc the median
```

[177]: 1.5034066538560549

What is the mean height in Pareto world?

[178]:
```
# Calc the mean height

dist.mean()
```

[178]: 2.428571428571429

What fraction of people are shorter than the mean?

[179]:
```
# calc the CDF value at the mean of the pareto distribution

dist.cdf(dist.mean())

# 78 % of people are shorter than the mean
```

[179]: 0.778739697565288

Out of 7 billion people in Paretom, how many are taller than 1 km?

[180]:
```
# calc the prob that a random var from the pareto dist is > than 1000 and then␣
↪scales it by 7 billion

```

```
(1 - dist.cdf(1000)) * 7000000000  # Use the CDF value convert from 1 meter to␣
 ↪a KM * 7 billion
```

[180]: 55602.976430479954

How tall do we expect the tallest person to be?

[181]:
```
# calc the survival function value at 1000 in the pareto distribution scaled by␣
 ↪7 billion

dist.ppf(1 - 1 / 7000000000)
```

[181]: 618349.6106759505

0.0.2 Exercise 6-1

[182]:
```
# 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/hinc.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/hinc06.
 ↪csv")
```

[183]:
```
# import Numpy, thinkstats2, thinkplot, and the Hinc.py and read the csv file␣
 ↪as an income data frame


import numpy as np
import thinkstats2
import thinkplot
import hinc
income_df = hinc.ReadData()
```

[184]:
```
# create a function named interpolate sample with two inputs Df, log_upper as =␣
 ↪6.0

def InterpolateSample(df, log_upper=6.0):
    # 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
```

[185]:
```
# create a function to calc the raw moment of a set of values xs up to the␣
↪specified order k

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

[186]:
```
# create a function to calc the mean of a given list of values xs

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

[187]:
```
# create a function  to calc the median of a given list of values xs using the␣
↪thinkstats2.Cdf function

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

[188]:
```
# creat a function to calc the skewness of a given list of values xs

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

[ ]:
```
# function to calc the central moment of order k for a given list of values xs

def CentralMoment(xs, k):
```

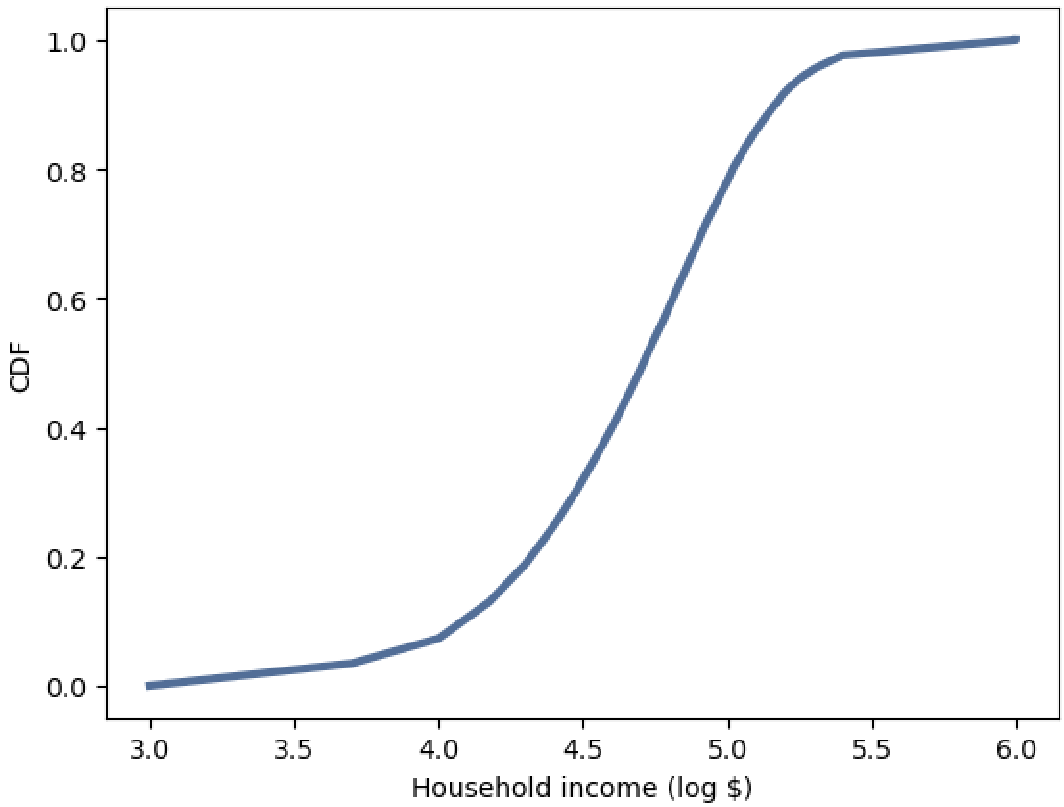
```
    mean = RawMoment(xs, 1)
    return sum((x - mean)**k for x in xs) / len(xs)
```

[199]:
```
# create a function to calc the Pearson's median skewness for a given list of␣
↪values xs

def PearsonMedianSkewness(xs):
    median = Median(xs)                  # calc the median of the input values xs
    mean = RawMoment(xs, 1)              # calc the mean of the input values xs
    var = CentralMoment(xs, 2)           # calc the variance of the input values
    std = np.sqrt(var)                   # calc the standard dev by taking the␣
↪square root of the calc variance
    gp = 3 * (mean - median) / std       # calc pearson median skewness
    return gp
```

[190]:
```
# calc the stand moment of order k for a given list of values xs

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

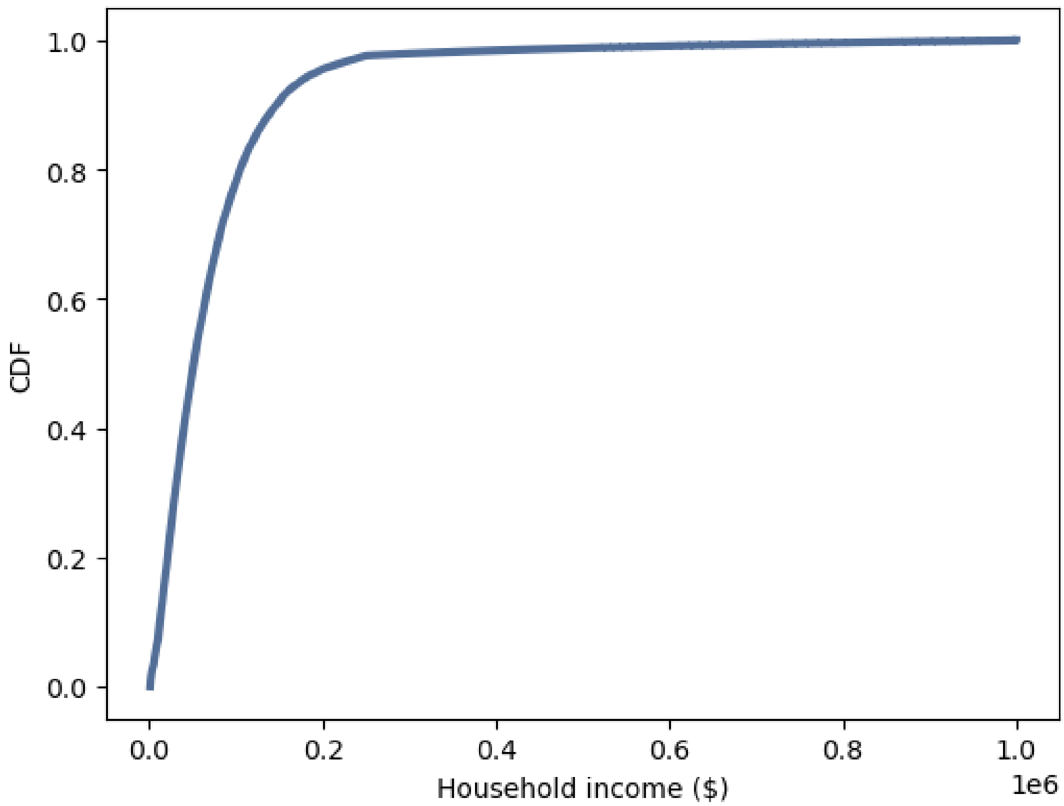
[192]:
```
# using the InterpolateSample function passing the income DF and the log_upper␣
↪to store the log sample var

log_sample = InterpolateSample(income_df, log_upper=6.0)
```

[193]:
```
# using the thinkstats2 library create a CDF) plot for the log-transformed␣
↪sample log_sample

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



[194]:
```
# use NumPy's power function to raise 10 to the power of each element in the
↪log_sample

sample = np.power(10, log_sample)
```

[195]:
```
# create a CDF plot of household income

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



[196]:
```
# Using the Mean and Median functions to calc the mean and median of the sample

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

[196]: (74278.70753118733, 51226.45447894046)

[197]:
```
#  Using the Skewness and PearsonMedianSkewness functions to calc those␣
 ↪functions agains the sample

Skewness(sample), PearsonMedianSkewness(sample)
```

[197]: (4.949920244429583, 0.7361258019141782)

[198]:
```
# use the cdf to calc the prob of a value being less than or equal to the mean␣
 ↪of the sample

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

[198]: 0.660005879566872

66% of the households makes less than the mean

[ ]:

```

```