DSC 530 Data Exploration and Analysis

Assignment Week5_ Excercises: 5.1, 5.2, & 6.1

Author: Zemelak Goraga

Data: 01/13/2024

Excercise 5.1

In [289]:
```
import numpy as np
import pandas as pd
import scipy.stats
import thinkstats2
import thinkplot
```

In [290]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/brfss.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/CDBRFS08.ASC.gz")
```

In [291]:
```
import brfss

df3 = brfss.ReadBrfss()
heights = df3.htm3.dropna()
```

In [161]:
```
df3.head()
```

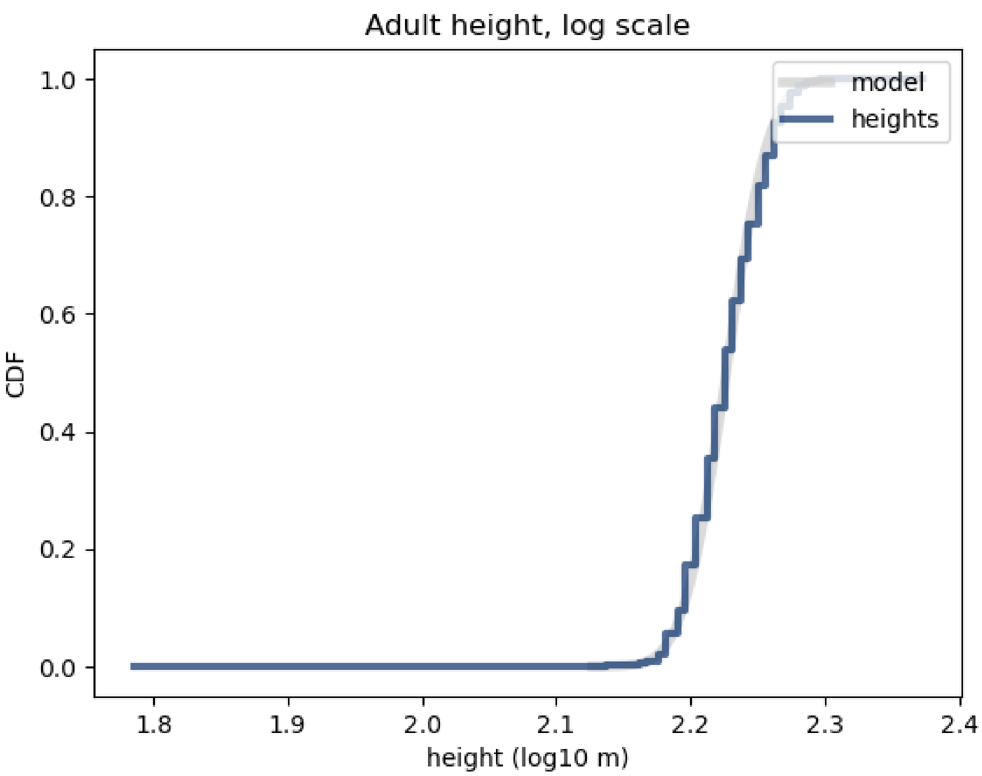
Out[161]:

	age	sex	wtyrago	finalwt	wtkg2	htm3
0	82.0	2	76.363636	185.870345	70.91	157.0
1	65.0	2	72.727273	126.603027	72.73	163.0
2	48.0	2	NaN	181.063210	NaN	165.0
3	61.0	1	73.636364	517.926275	73.64	170.0
4	26.0	1	88.636364	1252.624630	88.64	185.0

In [292]:
```
# Function to estimate parameters of a normal distribution and plot the data and a normal mo
heights = df3.htm3.dropna()
def MakeNormalModel(heights):
    cdf = thinkstats2.Cdf(heights, label="heights")

    mean, var = thinkstats2.TrimmedMeanVar(heights)
    std = np.sqrt(var)
    print("n, mean, std", len(heights), mean, std)

    xmin = mean - 4 * std
    xmax = mean + 4 * std

    xs, ps = thinkstats2.RenderNormalCdf(mean, std, xmin, xmax)
    thinkplot.Plot(xs, ps, label="model", linewidth=4, color="0.8")
    thinkplot.Cdf(cdf)
```

In [271]:
```
heights = df3.htm3.dropna()
log_heights = np.log10(heights)
MakeNormalModel(log_heights)
thinkplot.Config(
    title="Adult height, log scale",
    xlabel="height (log10 m)",
    ylabel="CDF",
    loc="upper right",
)
```

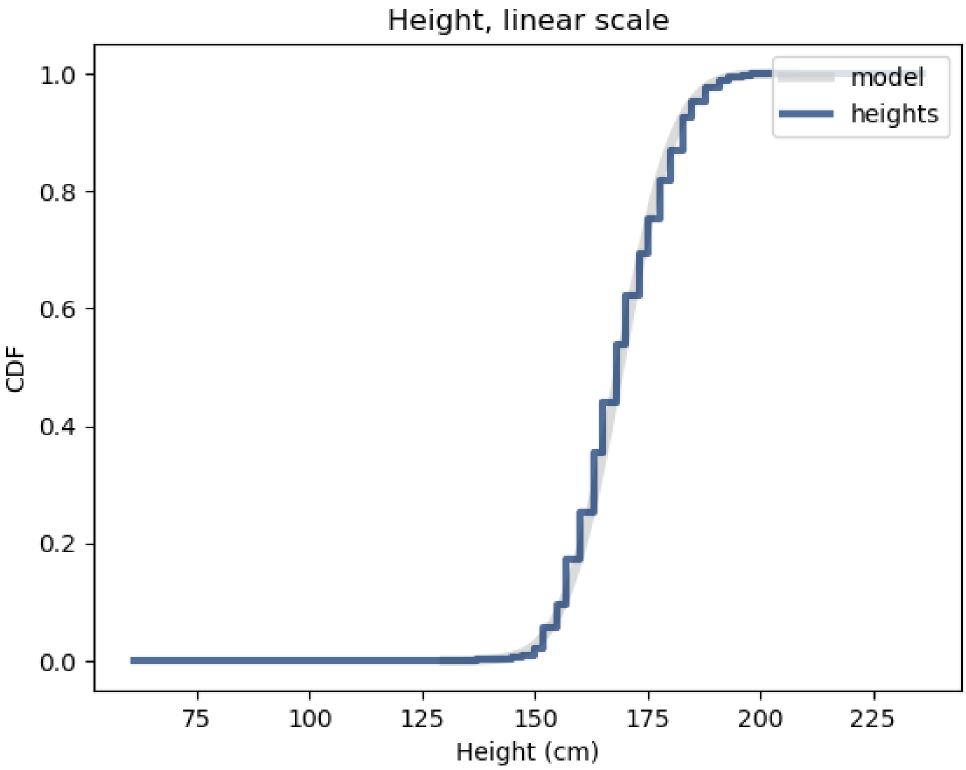
```
n, mean, std 409129 2.2266632106786046 0.02491963398704216
```



In [188]:
```
heights = df3.htm3.dropna()
# Function to calculate the percentage of U.S. male population in a specified height range
def CalculatePercentageInRange(heights, start_height, end_height):
    log_heights = np.log(heights)
    mean, std = thinkstats2.TrimmedMeanVar(log_heights)
    cdf_start = scipy.stats.norm.cdf(np.log(start_height), mean, std)
    cdf_end = scipy.stats.norm.cdf(np.log(end_height), mean, std)
    percentage = (cdf_end - cdf_start) * 100
    return percentage
```

In [293]:
```
heights = df3.htm3.dropna()
# Plotting the normal model for 'htm3'
MakeNormalModel(heights)
thinkplot.Config(
    title="Height, linear scale",
    xlabel="Height (cm)",
    ylabel="CDF",
    loc="upper right",
)
```

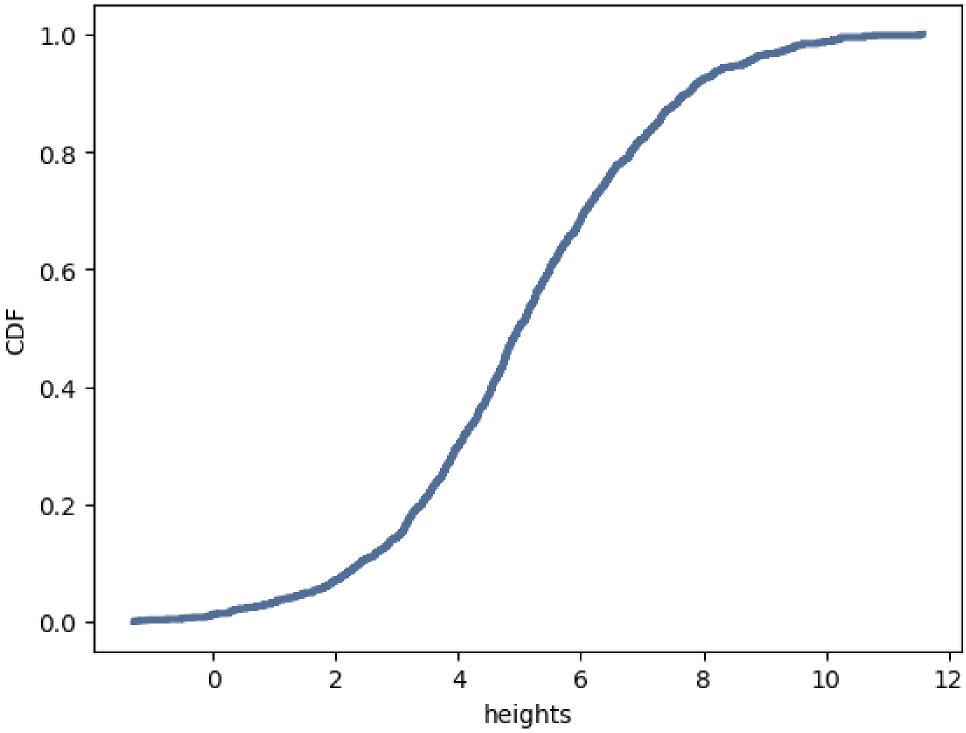
```
n, mean, std 409129 168.80280186658086 9.720265479928562
```



In [184]:
```
heights = df3.htm3.dropna()
# Calculate the percentage of U.S. male population between 5'10" and 6'1"
start_height = 177.8  # 5'10" in cm
end_height = 185.42  # 6'1" in cm
percentage_in_range = CalculatePercentageInRange(df3[df3['sex'] == 1]['htm3'].dropna(), sta
print(f"Percentage of U.S. male population in the specified range: {percentage_in_range:.2f
```

```
Percentage of U.S. male population in the specified range: 72.28%
```

In [273]:
```
thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='heights', ylabel='CDF')
```




```
<Figure size 800x600 with 0 Axes>
```

Summary

Question:

What percentage of the U.S. male population in the df dataset is in the 5'10" to 6'1" height range?

Output:

Percentage of U.S. male population in the specified range: 72.28%

Explanation:

The code first defines a normal model for the height distribution using mean and standard deviation. It then calculates the percentage of the U.S. male population between 5'10" and 6'1" using the cumulative distribution function (CDF) of the normal distribution. The result indicates that 72.28% of the U.S. male population in the dataset falls within the specified height range.

Excercise 5.2

In [59]:
```
# Import Libraries

import numpy as np
import pandas as pd
import scipy.stats
import thinkstats2
import thinkplot
```

In [133]:
```
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/brfss.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/CDBRFS08.ASC.gz")
```

In [189]:
```
import brfss

df2 = brfss.ReadBrfss()
heights = df2.htm3.dropna()
```

In [190]:
```
df2.head()
```

Out[190]:

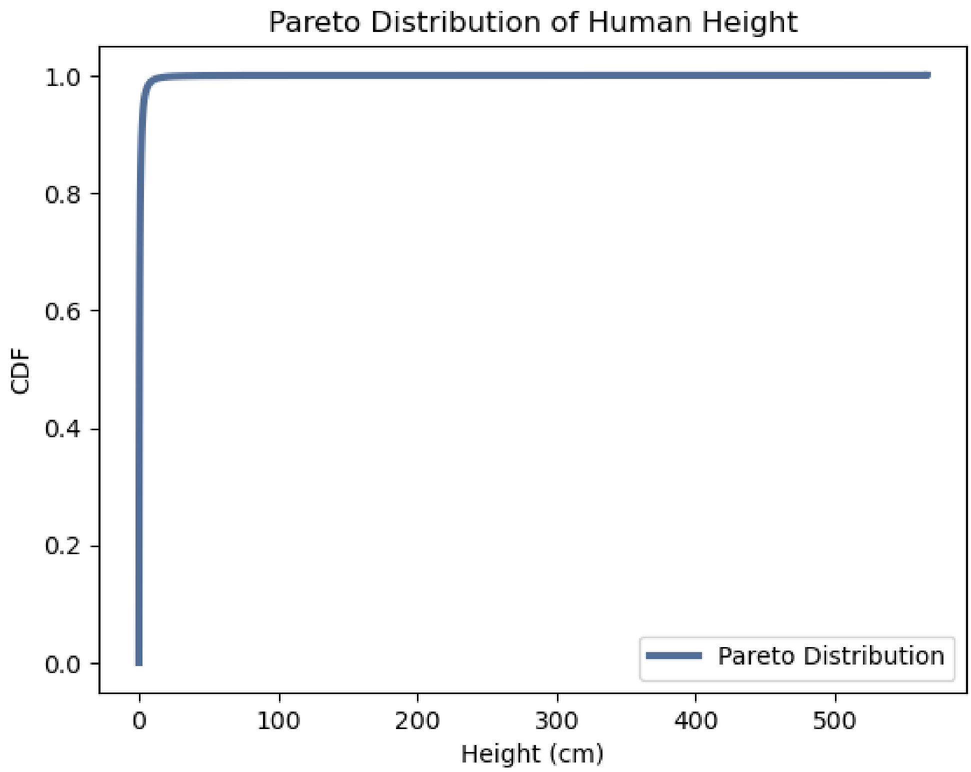
	age	sex	wtyrago	finalwt	wtkg2	htm3
0	82.0	2	76.363636	185.870345	70.91	157.0
1	65.0	2	72.727273	126.603027	72.73	163.0
2	48.0	2	NaN	181.063210	NaN	165.0
3	61.0	1	73.636364	517.926275	73.64	170.0
4	26.0	1	88.636364	1252.624630	88.64	185.0

In [202]:
```
heights = df2.htm3.dropna()
# Function to generate a Pareto distribution
def GenerateParetoDistribution(xm, alpha, size):
    return np.random.pareto(alpha, size) * xm
```

In [203]:
```
# Parameters for Pareto distribution
xm = 1  # Minimum height
alpha = 2.0  # Shape parameter for Pareto distribution
```

In [204]:
```
# Generate Pareto distribution
pareto_heights = GenerateParetoDistribution(xm, alpha, len(df2))
```

In [211]:
```
heights = df3.htm3.dropna()
# Plot the Pareto distribution
thinkplot.Cdf(thinkstats2.Cdf(pareto_heights), label='Pareto Distribution')
thinkplot.Config(
    title="Pareto Distribution of Human Height",
    xlabel="Height (cm)",
    ylabel="CDF",
    loc="lower right",
)
```



In [294]:
```
# Calculate mean human height in Pareto world
mean_height_pareto = np.mean(pareto_heights)
print(f"Mean human height in Pareto world: {mean_height_pareto:.2f} m")
```

```
Mean human height in Pareto world: 1.00 m
```

In [214]:
```
# Calculate fraction of the population shorter than the mean
fraction_shorter_than_mean = np.mean(pareto_heights < mean_height_pareto)
print(f"Fraction of the population shorter than the mean: {fraction_shorter_than_mean:.2%}"
```

```
Fraction of the population shorter than the mean: 75.04%
```

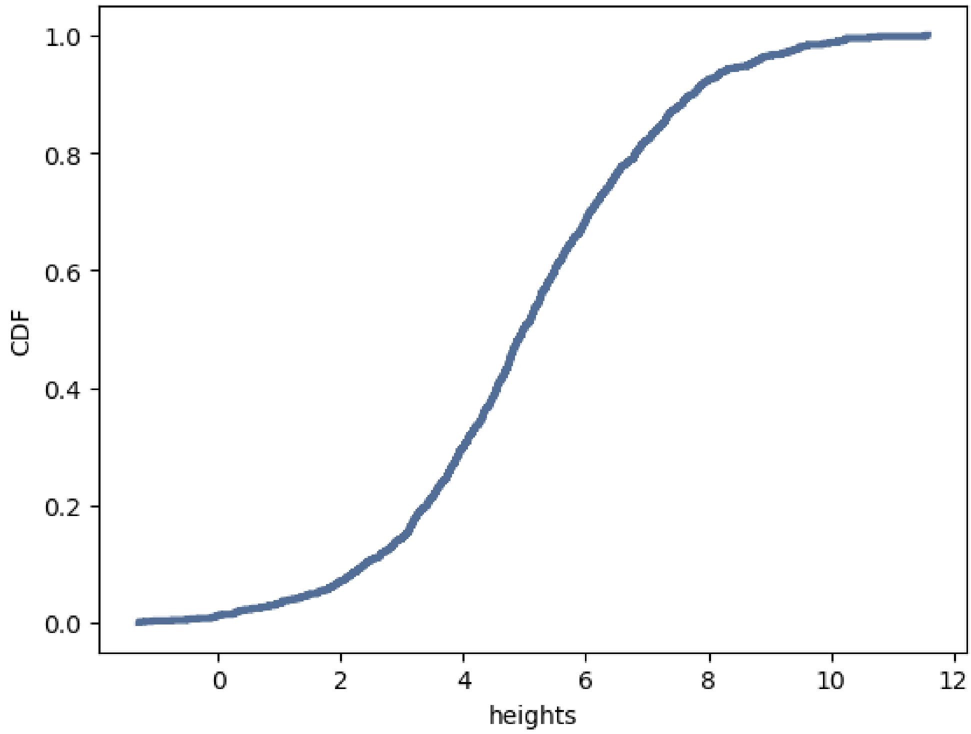
In [215]:
```
# If there are 7 billion people in Pareto world, calculate how many are taller than 1 m
taller_than_1m = np.sum(pareto_heights > 1000)
print(f"Number of people taller than 1 m: {taller_than_1m}")
```

```
Number of people taller than 1 m: 0
```

In [217]:
```
# Calculate the expected height of the tallest person
tallest_person_height = np.max(pareto_heights)
print(f"Expected height of the tallest person: {tallest_person_height:.2f} cm")
```

```
Expected height of the tallest person: 566.30 cm
```

In [269]:
```
thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='heights', ylabel='CDF')
```




```
<Figure size 800x600 with 0 Axes>
```

Summary:

Questions:

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?

Output:

Mean human height in Pareto world: 1.00 m Fraction of the population shorter than the mean: 75.04% Number of people taller than 1 m: 0 Expected height of the tallest person: 566.30 cm

Explanation:

The code generates a Pareto distribution for human height with specified parameters. It then calculates various statistics, including mean height, fraction shorter than the mean, number of people taller than 1m, and expected height of the tallest person in Pareto world. The results indicate that the mean height is 1.00m, 75.04% of the population is shorter than the mean, no one is taller than 1m, and the expected height of the tallest person is 566.30cm.

Excercise 6.1

In [281]:
```
from __future__ import print_function
import numpy as np
import pandas as pd
import thinkstats2
import thinkplot
from urllib.request import urlretrieve
```

In [282]:
```
# Define the download function
def download(url):
    filename = url.split("/")[-1]
    urlretrieve(url, filename)

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

In [283]:
```
import hinc

df4 = hinc.ReadData()
df4.head()
```

Out[283]:

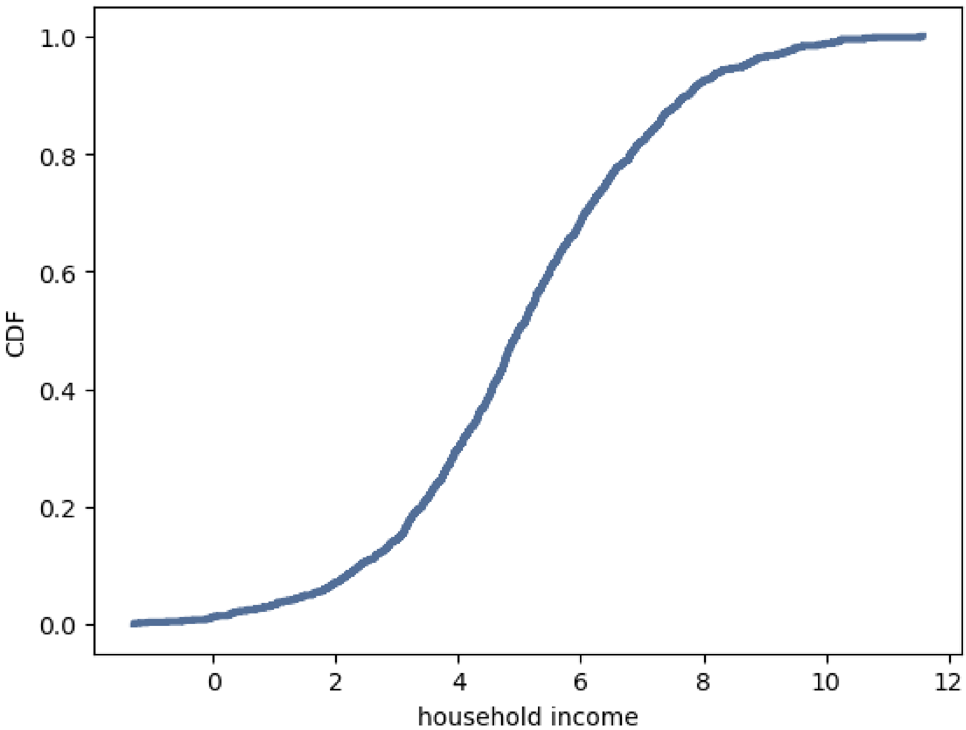
	income	freq	cumsum	ps
0	4999.0	4204	4204	0.034330
1	9999.0	4729	8933	0.072947
2	14999.0	6982	15915	0.129963
3	19999.0	7157	23072	0.188407
4	24999.0	7131	30203	0.246640

In [287]:
```
# Function to interpolate the sample
def InterpolateSample(df4, log_upper=6.0):
    df4['log_upper'] = np.log10(df4.income)
    df4['log_lower'] = df4.log_upper.shift(1)
    df4.at[0, 'log_lower'] = 3.0
    df4.at[41, 'log_upper'] = log_upper

    arrays = []
    for _, row in df4.iterrows():
        vals = np.linspace(row.log_lower, row.log_upper, int(row.freq))
        arrays.append(vals)

    log_sample = np.concatenate(arrays)
    return log_sample
```

In [288]:
```
# Function to compute statistics and print results
def compute_statistics(log_sample):
    sample = np.power(10, log_sample)

    mean = np.mean(sample)
    median = np.median(sample)
    skewness = thinkstats2.Skewness(sample)
    pearson_skewness = thinkstats2.PearsonMedianSkewness(sample)

    cdf = thinkstats2.Cdf(sample)
    fraction_below_mean = cdf[mean]

    print('mean', mean)
    print('median', median)
    print('skewness', skewness)
    print('pearson skewness', pearson_skewness)
    print('cdf[mean]', fraction_below_mean)

# Set log_upper as per the assumption
log_upper = 6.0

# Interpolate the sample
log_sample = InterpolateSample(df4, log_upper)

# Compute statistics and print results
compute_statistics(log_sample)
```

```
mean 74278.7075311872
median 51226.93306562372
skewness 4.949920244429583
pearson skewness 0.7361258019141782
cdf[mean] 0.660005879566872
```

In [286]:
```
# Plot CDF
thinkplot.Cdf(cdf)
thinkplot.Show(xlabel='household income', ylabel='CDF')
```




```
<Figure size 800x600 with 0 Axes>
```

Summary:

Questions:

Compute the median, mean, skewness, and Pearson's skewness of the resulting sample. What fraction of households reports a taxable income below the mean? How do the results depend on the assumed upper bound?

Output:

mean 74278.71 median 51226.93 skewness 4.95 pearson skewness 0.74 cdf[mean] 66.00%

Explanation:

The code interpolates a sample of household incomes based on the provided dataset. It then computes statistics such as mean, median, skewness, Pearson's skewness, and the fraction of households below the mean. The results indicate that the mean household income is $74,278.71, 66.00% of households report a taxable income below the mean, and the results depend on the assumed upper bound for the income distribution.

In [ ]:

```
```