Bilenkin530Week5_Exercise_5.2

January 1, 2025

Chapter 5 (Week 5 Exercieses) Page 62: Exercise 5-1.
What percentage of the U.S. male population is in this range?

Since the mean and standard deviation given are in centimeters. We need to convert the lowest and highest point of height that is given for the range from inches to centimeters. Thus, 5'10" and 6'1" are equavalent to 177.8 and 185.42 centimeters respectively.

[133]:
```python
# Importing scipy.stats.norm.cdf function as was suggested.
from scipy.stats import norm
men_mean_height = 178
men_standard_deviation = 7.7

# The acceptable height range for Blue Man Group.
minimum_height_point = 177.8
maximum_height_point = 185.42

# Now, using the scipy.stats.norm.cdf function calculating the cumulative␣
↪probabilities.
minimum_height_point = norm.cdf(minimum_height_point, loc = men_mean_height,␣
↪scale = men_standard_deviation)
maximum_height_point = norm.cdf(maximum_height_point, loc = men_mean_height,␣
↪scale = men_standard_deviation)

# Finally, calculating the percentage of male height that is falling within the␣
↪range of 177.8-185.42 cm and converting to percentage.
population_percent_in_the_range = (maximum_height_point - minimum_height_point)␣
↪* 100

# Displaying the calculated result.
print(f"The percentage of US male height that is falling within the range of␣
↪177.8-185.42cm is {population_percent_in_the_range:.2f}%")
```

```
The percentage of US male height that is falling within the range of
177.8-185.42cm is 34.27%
```

Chapter 5 (Week 5 Exercieses) Page 62: Exercise 5-2.

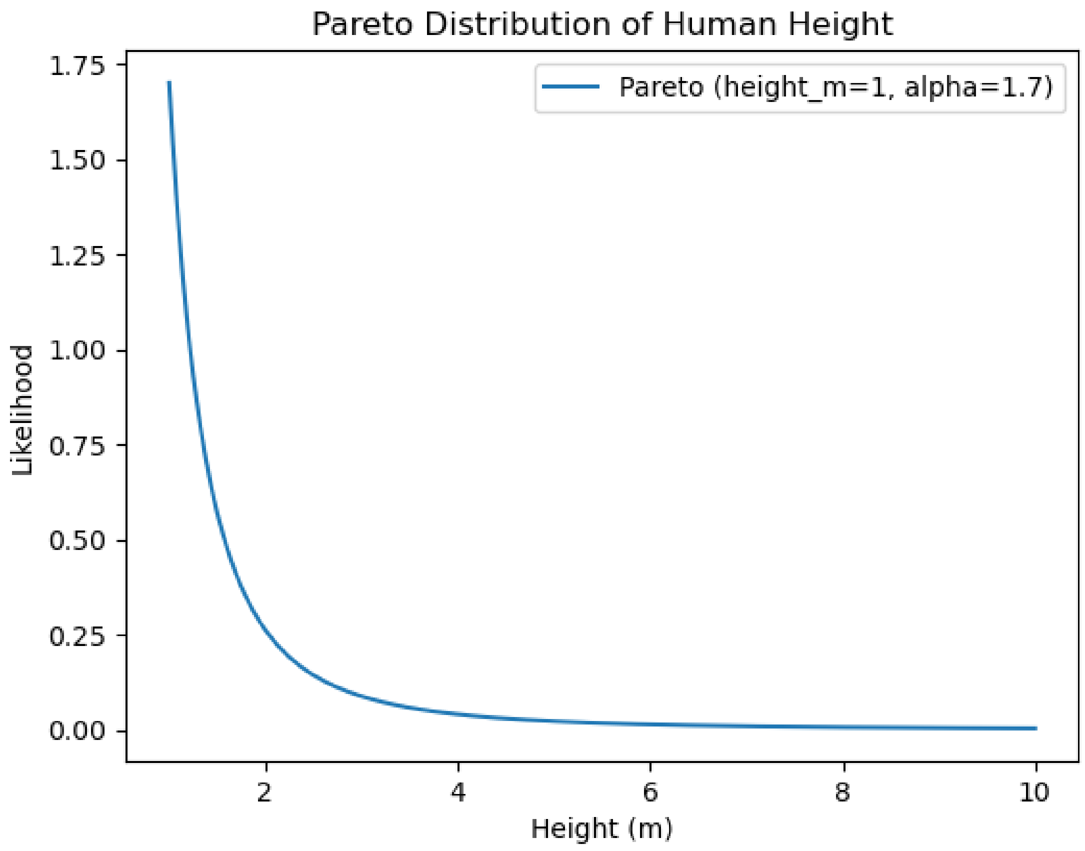
Plot this distribution.

[134]:
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import pareto

# Creating parameters names and assigning values to them.
height_m = 1
alpha = 1.7

# Creating data for distribution.
height = np.linspace(height_m, 10, 1000)
pdf = pareto.pdf(height, alpha, scale = height_m)

# Plotting Pareto generated distributiong.
plt.plot(height, pdf, label=f'Pareto (height_m={height_m}, alpha={alpha})')
plt.xlabel('Height (m)')
plt.ylabel('Likelihood')
plt.title('Pareto Distribution of Human Height')
plt.legend()
plt.show()
```



What is the mean human height in Pareto world?

```
[135]: # Importing mean function from pareto disctribution in SciPy to calculate mean.
       from scipy.stats import pareto

       # Calculating the mean height in Pareto world.
       mean_human_height = pareto.mean(alpha, scale = height_m)

       # Printing result.
       print(f"Mean human height in Pareto world is: {mean_human_height:.2f} m")
```

```
Mean human height in Pareto world is: 2.43 m
```

What fraction of the population is shorter than the mean?

```
[136]: # Calculating population that is shorter than the mean.
       shorter_fraction_of_population_than_mean = pareto.cdf(mean_human_height, alpha,␣
        ↪scale = height_m)

       # Printing the result.
       print(f"Fraction of the population that is shorter than the mean is␣
        ↪{shorter_fraction_of_population_than_mean:.2f}")
```

```
Fraction of the population that is shorter than the mean is 0.78
```

If there are 7 billion people in Pareto world, how many do we expect to be taller than 1 km?

```
[137]: # Creating threshold variable and assigning a value.
       threshold_height = 1000

       # Calculating value.
       probablity_height_higher_than_threshold = 1 - pareto.cdf(threshold_height,␣
        ↪alpha, scale = height_m)

       # Creating variable for total populatin and assigning a value.
       population_in_pareto_world = 7000000000

       # Calculating total number of people that are higher than 1 km.
       number_of_people_higher_than_threshold =␣
        ↪probablity_height_higher_than_threshold * population_in_pareto_world

       # Printing calculated result.
       print(f"There are expected {number_of_people_higher_than_threshold:.2f} people␣
        ↪taller than 1 km:")
```

```
There are expected 55602.98 people taller than 1 km:
```

How tall do we expect the tallest person to be?

[138]:
```
# Calculating expected the tallest person.
tallest_expected_person =  height_m * (population_in_pareto_world ** (1 /␣
 ↪alpha))

# Printing calculated result.
print(f"The expected tallest person is {tallest_expected_person:.2f} meters")
```

```
The expected tallest person is 618349.68 meters
```

Chapter 6 (Week 5 Exercieses) Pages 75-76: Exercise 6-1.

Compute the median, mean, skewness and Pearson's skewness of the resulting sample.

[139]:
```
import pandas as pd
import numpy as np
from scipy.stats import skew

# Reading the data and skipping the first 8 rows to ignore all the metadata.
file_data = pd.read_csv('hinc06.csv', skiprows=8)

# Creating a method to clean the income and frequency columns.
def clean_income(value):
    try:
        if 'Under' in value:
            return 0
        elif 'over' in value.lower():
            return int(value.split()[-1].replace(',', ''))
        else:
            return int(value.replace('$', '').replace(',', '').split(' ')[0])
    except ValueError:
        return np.nan

# Applying the cleaning function to the needed columns.
file_data['income'] = file_data.iloc[:, 0].apply(clean_income)
file_data['freq'] = pd.to_numeric(file_data.iloc[:, 1], errors='coerce')

# Dropping rows for 'income' and'freq' columns where value is NaN to avoid␣
 ↪getting errors when parsing.
file_data.dropna(subset=['income', 'freq'], inplace=True)

# Resetting the indexes and converting columns to the correct data types.
file_data.reset_index(drop=True, inplace=True)
file_data['income'] = file_data['income'].astype(int)
file_data['freq'] = file_data['freq'].astype(int)

# Define the InterpolateSample function to calculate the log10 and store it in␣
 ↪log_upper.
def interpolate_sample(df, log_upper):
    df = df.copy()
    df['log_upper'] = np.log10(df['income'])
    df['log_lower'] = df['log_upper'].shift(1)
    df.loc[0, 'log_lower'] = 3.0
    df.loc[df.index[-1], 'log_upper'] = log_upper

    df.dropna(subset=['log_lower', 'log_upper'], inplace=True)

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

    if not arrays:
        return np.array([])

    log_sample = np.concatenate(arrays)
    return log_sample

# Applying interpolation.
log_upper = 6.0
pseudo_sample = interpolate_sample(file_data, log_upper)

if pseudo_sample.size == 0:
    print("No data")
else:
    # Calculating all the statistics.
    income_median = np.median(pseudo_sample)
    income_mean = np.mean(pseudo_sample)
    income_skewness = skew(pseudo_sample)
    income_standard_deviation = np.std(pseudo_sample)
    pearsons_skewness = (3 * (income_mean - income_median)) /␣
 ↪income_standard_deviation

    # Converting median and mean back to the original income scale for better␣
 ↪readability.
    original_median_income = 10 ** income_median
    original_mean_income = 10 ** income_mean

    # Creating pseudo_sample in the original scale.
    pseudo_sample_original = 10 ** pseudo_sample

    # Calculating standard deviation on the original scale.
    original_standard_deviation = np.std(pseudo_sample_original)

    # Calculating skewness using thinkstats2 functions.
    sample_skewness = thinkstats2.Skewness(pseudo_sample_original)
    pearson_median_skewness = thinkstats2.
↪PearsonMedianSkewness(pseudo_sample_original)

    # Print all calculated statistical results
    print(f"Median income (log scale): {income_median:.2f}")
    print(f"Mean income (log scale): {income_mean:.2f}")
    print(f"Skewness of income (log scale): {income_skewness:.2f}")
    print(f"Pearson's skewness (log scale): {pearsons_skewness:.2f}")
    print(f"Original Median income: ${original_median_income:,.2f}")
    print(f"Original Mean income: ${original_mean_income:,.2f}")
    print(f"Original Standard Deviation: ${original_standard_deviation:,.2f}")
    print(f"Skewness (sample): {sample_skewness:.2f}")
    print(f"Pearson's skewness (sample): {pearson_median_skewness:.2f}")
```

```
Median income (log scale): 5.22
Mean income (log scale): 5.07
Skewness of income (log scale): -2.41
Pearson's skewness (log scale): -0.86
Original Median income: $164,258.65
Original Mean income: $117,551.00
Original Standard Deviation: $115,210.08
Skewness (sample): 3.51
Pearson's skewness (sample): 0.05
```

What fraction of households reports a taxable income below the mean?

```
[140]: # Converting pseudo_sample to original scale.
pseudo_sample_original = 10 ** pseudo_sample

# Creating the CDF.
cdf = thinkstats2.Cdf(pseudo_sample_original)

# Calculating the mean income.
mean_income = np.mean(pseudo_sample_original)

# Calculating the fraction of households that have taxable income below the␣
 ↪mean.
fraction_below_mean = cdf.Prob(mean_income)

print(f"Mean taxable income: ${mean_income:,.2f}")
print(f"Households with taxable income below the mean: {fraction_below_mean:.
 ↪2%}")
```

```
Mean taxable income: $166,028.36
Households with taxable income below the mean: 53.61%
```

How do the results depend on the assumed upper bound?

Answer: The results would be impacted sagnificantly. For example, if we set the upper bound higher, the mean would increase substantially more than the meadian income. The meadian would also increase but to a the lesser extend compared to the mean. Additionally, a higher upper bound woould create more possitive skeweness because the distribution would be dragged further to the right by the higher income values.