Rodriguez_Felipe_DSC530_Exercise5.2

January 15, 2023

Exercise 5.1

[199]:
```
# Copied from book to download scripts
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")
```

[200]:
```
import scipy.stats
import thinkstats2
import thinkplot
```

[201]:
```
# Set up values given
men_mu = 178
men_sigma = 7.7
# Creates data set
dist = scipy.stats.norm(loc=mu, scale=sigma)
```

[202]:
```
# Min Length converted to cm
length_feet_min = 5
length_inch_min = 10
total_cm_min = (length_feet_min * 30.48 ) + (length_inch_min * 2.54)
total_cm_min
```

[202]:
```
177.8
```

[203]:
```python
# Max length converted to cm
length_feet_max = 6
length_inch_max = 1
total_cm_max = (length_feet_max * 30.48 ) + (length_inch_max * 2.54)
total_cm_max
```

[203]: 185.42

[204]:
```python
# Calculates amount of people in the height range
min_amount = dist.cdf(total_cm_min)  # Lowest height 5'10" amount
max_amount = dist.cdf(total_cm_max)  # Highest height 6'1" amount
# Calculates percentage of people between height range
print('''Percentage of people that are between 5'10" and 6'1":''', ((max_amount␣
 ↪- min_amount)*100))
```

```
Percentage of people that are between 5'10" and 6'1": 34.274683763147365
```

Exercise 5.2

[205]:
```python
import scipy.stats
```

[206]:
```python
# Set up values given
alpha = 1.7
xmin = 1  # meter
# Creates Data
human_height = scipy.stats.pareto(b=alpha, scale=xmin)
```

[207]:
```python
# Calculates Mean
human_height.mean()
```

[207]: 2.428571428571429

[208]:
```python
# Calculation for people taller 1 km out of 7 billion
(1 - human_height.cdf(1000)) * 7e9
```

[208]: 55602.976430479954

[209]:
```python
human_height.ppf(1 - 1 / 7e9)
```

[209]: 618349.6106759505

[210]:
```python
human_height.sf(600000) * 7e9
```

[210]: 1.0525455861201714

Exercise 6.1

[211]:
```python
# Copied from book to download scripts
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)
```

[212]:
```python
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/hinc.py")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/hinc06.
 ↪csv")
download("https://github.com/AllenDowney/ThinkStats2/raw/master/code/
 ↪thinkstats2.py")
```

[213]:
```python
import hinc
import numpy as np
```

[214]:
```python
# Reads Data
income_df = hinc.ReadData()
```

[215]:
```python
# Copied from book to use function
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
```

[216]:
```
# Converts data
sample_data_log = InterpolateSample(income_df)
```

[217]:
```
data = np.power(10, sample_data_log)
```

[218]:
```
# Calculates Mean
means = thinkstats2.Mean(data)
means
```

[218]: 74278.7075311872

[219]:
```
# Calculates Median
thinkstats2.Median(data)
```

[219]: 51226.45447894046

[220]:
```
# Calculates Skewness
thinkstats2.Skewness(data)
```

[220]: 4.949920244429583

[221]:
```
# Calculates Pearson Median Skewness
thinkstats2.PearsonMedianSkewness(data)
```

[221]: 0.7361258019141782

[222]:
```
# Creates CDF of Data
cdf = thinkstats2.Cdf(data)
```

[223]:
```
# Calculatation of what people make below mean, close to 66%
cdf.Prob(means)
```

[223]: 0.660005879566872

How do the results depend on the assumed upper bound?
The Mean, Skewness, Pearson Median Skewness, and people who make below the mean all change

because the upper bound is increased. The data set shifts to the right. The value that remains consistent is the median of the data, which does not change based on upper bound.