1. Where are the old left-handed people?

Barack Obama is left-handed. So are Bill Gates and Oprah Winfrey; so were Babe Ruth and Marie Curie. A 1991 study (https://www.nejm.org/doi/full/10.1056/NEJM199104043241418) reported that left-handed people die on average nine years earlier than right-handed people. Nine years! Could this really be true?

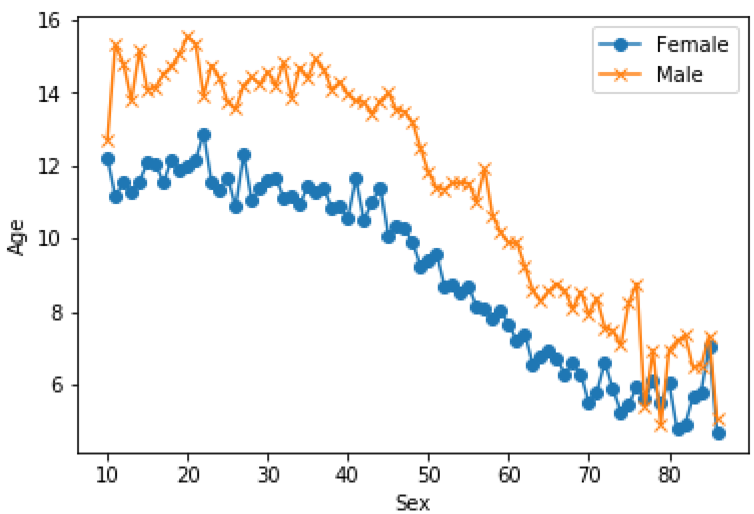
In this notebook, we will explore this phenomenon using age distribution data to see if we can reproduce a difference in average age at death purely from the changing rates of left-handedness over time, refuting the claim of early death for left-handers. This notebook uses `pandas` and Bayesian statistics to analyze the probability of being a certain age at death given that you are reported as left-handed or right-handed.

A National Geographic survey in 1986 resulted in over a million responses that included age, sex, and hand preference for throwing and writing. Researchers Avery Gilbert and Charles Wysocki analyzed this data and noticed that rates of left-handedness were around 13% for people younger than 40 but decreased with age to about 5% by the age of 80. They concluded based on analysis of a subgroup of people who throw left-handed but write right-handed that this age-dependence was primarily due to changing social acceptability of left-handedness. This means that the rates aren't a factor of *age* specifically but rather of the *year you were born*, and if the same study was done today, we should expect a shifted version of the same distribution as a function of age. Ultimately, we'll see what effect this changing rate has on the apparent mean age of death of left-handed people, but let's start by plotting the rates of left-handedness as a function of age.

This notebook uses two datasets: death distribution data (https://www.cdc.gov/nchs/data/statab/vs00199_table310.pdf) for the United States from the year 1999 (source website here (https://www.cdc.gov/nchs/nvss/mortality_tables.htm)) and rates of left-handedness digitized from a figure in this 1992

In [210]:
```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt

# load the data
data_url_1 = "https://gist.githubusercontent.com/mbonsma/8da0990b71ba9a09f7de395574e54df1/raw/aec88b30af87fad
8d45da7e774223f91dad09e88/lh_data.csv"
lefthanded_data = pd.read_csv(data_url_1)

# plot male and female left-handedness rates vs. age
%matplotlib inline
fig, ax = plt.subplots() # create figure and axis objects
ax.plot("Age", "Female",data=lefthanded_data, marker = 'o') # plot "Female" vs. "Age"
ax.plot("Age", "Male",data=lefthanded_data,marker = 'x') # plot "Male" vs. "Age"
ax.legend() # add a legend
ax.set_xlabel("Sex")
ax.set_ylabel("Age")
```

Out[210]:
```
Text(0,0.5,'Age')
```



In [211]:
```
%%nose

def test_data_shape():
    assert (lefthanded_data.shape == (77, 3)), \
    'The lefthanded_data you loaded is not the right shape. It should be 77, 3.'

def test_num_lines():
    assert (len(ax.lines) == 2), \
    'Did you plot lefthanded rates for both men and women?'

def test_plot_dimensions():
     assert ((ax.get_yticks()[-1] < 20) and (ax.get_xticks()[-1] > 20)), \
     'Did you plot "Female" vs. "Age" and "Male" vs. "Age"?'

def test_plot_labels():
    assert ax.get_xlabel() != '' and ax.get_ylabel() != '', \
    'Please add x and y labels to your plot.'
```

Out[211]:
```
4/4 tests passed
```

2. Rates of left-handedness over time

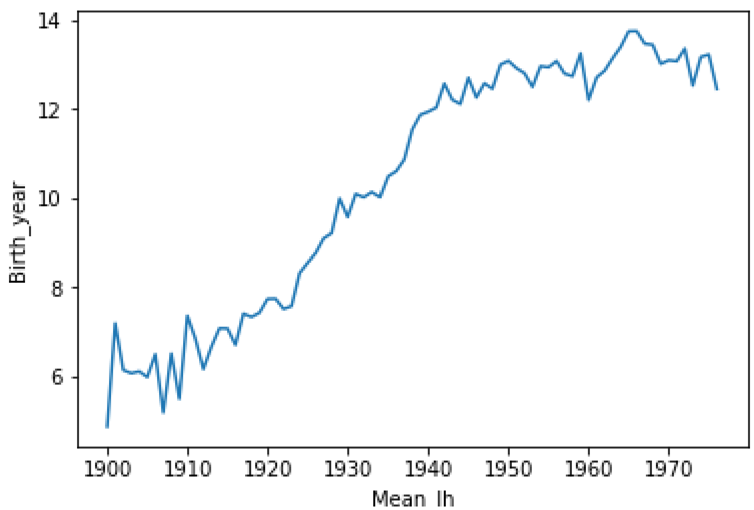
Let's convert this data into a plot of the rates of left-handedness as a function of the year of birth, and average over male and female to get a single rate for both sexes.

Since the study was done in 1986, the data after this conversion will be the percentage of people alive in 1986 who are left-handed as a function of the year they were born.

In [212]:
```
print(lefthanded_data.head())
# create a new column for birth year of each age
lefthanded_data['Birth_year']=1986-lefthanded_data['Age']

# create a new column for the average of male and female
lefthanded_data["Mean_lh"]=lefthanded_data[['Male','Female']].mean(axis=1)

print(lefthanded_data.head())
# create a plot of the 'Mean_lh' column vs. 'Birth_year'
fig, ax = plt.subplots()
ax.plot("Birth_year", "Mean_lh", data=lefthanded_data) # plot 'Mean_lh' vs. 'Birth_year'
ax.set_xlabel("Mean_lh") # set the x label for the plot
ax.set_ylabel("Birth_year") # set the y label for the plot
```

```
   Age       Male     Female
0   10  12.717558  12.198041
1   11  15.318830  11.144804
2   12  14.808281  11.549240
3   13  13.793744  11.276442
4   14  15.156304  11.572906
   Age       Male     Female  Birth_year    Mean_lh
0   10  12.717558  12.198041        1976  12.457800
1   11  15.318830  11.144804        1975  13.231817
2   12  14.808281  11.549240        1974  13.178760
3   13  13.793744  11.276442        1973  12.535093
4   14  15.156304  11.572906        1972  13.364605
```

```
Out[212]:  Text(0,0.5,'Birth_year')
```



In [213]:
```
%%nose

def test_new_columns():
    assert 'Birth_year' and 'Mean_lh' in lefthanded_data.columns, \
    'Did you create two new columns called "Birth_year" and "Mean_lh"?'

def test_birth_year_column():
    import numpy as np
    assert np.all(lefthanded_data["Birth_year"] >= 1900), \
    'The values in the "Birth_year" column should be years >= 1900.'

def test_mean_lh_column():
    import numpy as np
    assert not np.any(np.isnan(lefthanded_data["Mean_lh"])), \
    'Make sure you calculate the mean lefthandedness for each row of the DataFrame.'

def test_plot_contents():
    assert (len(ax.lines) == 1), \
    'Did you plot the mean lefthandedness data?'

def test_plot_labels():
    assert ax.get_xlabel() != '' and ax.get_ylabel() != '', \
    'Please add x and y labels to your plot.'
```

Out[213]: 5/5 tests passed

3. Applying Bayes' rule

The probability of dying at a certain age given that you're left-handed is **not** equal to the probability of being left-handed given that you died at a certain age. This inequality is why we need **Bayes' theorem**, a statement about conditional probability which allows us to update our beliefs after seeing evidence.

We want to calculate the probability of dying at age A given that you're left-handed. Let's write this in shorthand as P(A | LH). We also want the same quantity for right-handers: P(A | RH).

Here's Bayes' theorem for the two events we care about: left-handedness (LH) and dying at age A.

$$P(A|LH) = \frac{P(LH|A)P(A)}{P(LH)}$$

P(LH | A) is the probability that you are left-handed *given that* you died at age A. P(A) is the overall probability of dying at age A, and P(LH) is the overall probability of being left-handed. We will now calculate each of these three quantities, beginning with P(LH | A).

To calculate P(LH | A) for ages that might fall outside the original data, we will need to extrapolate the data to earlier and later years. Since the rates flatten out in the early 1900s and late 1900s, we'll use a few points at each end and take the mean to extrapolate the rates on each end. The number of points used for this is arbitrary, but we'll pick 10 since the data looks flat-ish until about 1910.

In [214]:

```
# ... YOUR CODE FOR TASK 3 ...
import numpy as np

# create a function for P(LH | A)
def P_lh_given_A(ages_of_death, study_year = 1990):
    """ P(Left-handed | ages of death), calculated based on the reported rates of left-handedness.
    Inputs: numpy array of ages of death, study_year
    Returns: probability of left-handedness given that subjects died in `study_year` at ages `ages_of_death`
 """
    # Use the mean of the 10 last and 10 first points for left-handedness rates before and after the start
    early_1900s_rate = lefthanded_data["Mean_lh"][-10:].mean()
    late_1900s_rate = lefthanded_data["Mean_lh"][:10].mean()
    middle_rates = lefthanded_data.loc[lefthanded_data['Birth_year'].isin(study_year - ages_of_death)]['Mean_
lh']
    youngest_age = study_year - 1986 + 10 # the youngest age is 10
    oldest_age = study_year - 1986 + 86 # the oldest age is 86

    P_return = np.zeros(ages_of_death.shape) # create an empty array to store the results
    # extract rate of left-handedness for people of ages 'ages_of_death'
    P_return[ages_of_death > oldest_age] = early_1900s_rate/100
    P_return[ages_of_death < youngest_age] = late_1900s_rate/100
    P_return[np.logical_and((ages_of_death <= oldest_age), (ages_of_death >= youngest_age))] = middle_rates/1
00

    return P_return
```

In [215]:
```
%%nose

def test_output_type():
    test_input = np.array([80])
    assert (type(P_lh_given_A(test_input)) == float or
            type(P_lh_given_A(test_input)) == np.float64 or
            type(P_lh_given_A(test_input)) == np.ndarray), \
    'Does the function P_lh_given_A return a number?'

def test_late_1900s_rate():
    assert round(float(P_lh_given_A(np.array([10]))), 2) == 0.13, \
    'Did you calculate a left-handedness probability for the late 1900s?'

def test_early_1900s_rate():
    assert round(float(P_lh_given_A(np.array([95]))), 2) == 0.06, \
    'Did you calculate a left-handedness probability for the early 1900s?'

def test_middle_rate():
    assert P_lh_given_A(np.array([80])) > 0.06 and P_lh_given_A(np.array([80])) < 0.13, \
    'Make sure that P_lh_given_A returns the correct left-handedness rate as a fraction (< 1).'

def test_middle_rates_are_different():
    assert np.any(P_lh_given_A(np.array([20])) != P_lh_given_A(np.array([50]))), \
    'P_lh_given_A should return different rates for different ages between youngest_age and oldest_age.'
```

Out[215]: 5/5 tests passed

4. When do people normally die?

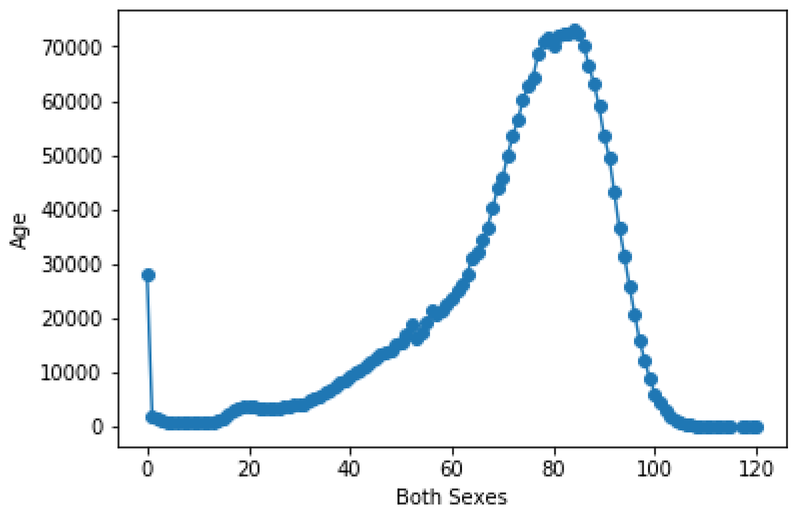
To estimate the probability of living to an age A, we can use data that gives the number of people who died in a given year and how old they were to create a distribution of ages of death. If we normalize the numbers to the total number of people who died, we can think of this data as a probability distribution that gives the probability of dying at age A. The data we'll use for this is from the entire US for the year 1999 - the closest I could find for the time range we're interested in.

In this block, we'll load in the death distribution data and plot it. The first column is the age, and the other columns are the number of people who died at that age.

In [216]:
```
# Death distribution data for the United States in 1999
data_url_2 = "https://gist.githubusercontent.com/mbonsma/2f4076aab6820ca1807f4e29f75f18ec/raw/62f3ec07514c7e3
1f5979beeca86f19991540796/cdc_vs00199_table310.tsv"

# load death distribution data
death_distribution_data=pd.read_csv(data_url_2,sep='\t',skiprows=[1])

print(death_distribution_data.head())
# drop NaN values from the `Both Sexes` column
death_distribution_data=death_distribution_data.dropna(subset=['Both Sexes'])

# plot number of people who died as a function of age
fig, ax = plt.subplots()
ax.plot("Age", "Both Sexes", data = death_distribution_data, marker='o') # plot 'Both Sexes' vs. 'Age'
ax.set_xlabel("Both Sexes")
ax.set_ylabel("Age")
```

```
   Age  Both Sexes     Male   Female
0    0     27937.0  15646.0  12291.0
1    1      1989.0   1103.0    886.0
2    2      1376.0    797.0    579.0
3    3      1046.0    601.0    445.0
4    4       838.0    474.0    364.0
```

Out[216]: Text(0,0.5,'Age')



In [217]:
```
%%nose

def test_skiprows():
    assert death_distribution_data.loc[0]["Age"] == 0, \
    'Make sure to include `skiprows=[1]` as an argument in `pd.read_csv`.'

def test_data_shape(): # this will also test if the dropna worked
    assert (death_distribution_data.shape == (120, 4)), \
    'Make sure you drop NaN values in the "Both Sexes" column only. The resulting DataFrame should have 120 r
ows.'

def test_plot_contents():
    assert (len(ax.lines) == 1), \
    'Did you plot the death distribution data?'

def test_plot_dimensions():
    assert ((ax.get_xticks()[-1] < 200) and (ax.get_yticks()[-1] > 200)), \
    'Did you plot "Both Sexes" vs. "Age"?'

def test_plot_labels():
    assert ax.get_xlabel() != '' and ax.get_ylabel() != '', \
    'Please add x and y labels to your plot.'
```

Out[217]: 5/5 tests passed

5. The overall probability of left-handedness

In the previous code block we loaded data to give us P(A), and now we need P(LH). P(LH) is the probability that a person who died in our particular study year is left-handed, assuming we know nothing else about them. This is the average left-handedness in the population of deceased people, and we can calculate it by summing up all of the left-handedness probabilities for each age, weighted with the number of deceased people at each age, then divided by the total number of deceased people to get a probability. In equation form, this is what we're calculating, where N(A) is the number of people who died at age A (given by the dataframe `death_distribution_data`):

$$P(LH) = \frac{\sum_{\text{A}} P(LH|A)N(A)}{\sum_{\text{A}} N(A)}$$

In [218]:
```
def P_lh(death_distribution_data, study_year = 1990): # sum over P_lh for each age group
    """ Overall probability of being left-handed if you died in the study year
    Input: dataframe of death distribution data, study year
    Output: P(LH), a single floating point number """
    p_list = death_distribution_data['Both Sexes'].mul(P_lh_given_A(death_distribution_data['Age'],study_year
)) # multiply number of dead people by P_lh_given_A
    p = np.sum(p_list) # calculate the sum of p_list
    return p/np.sum(death_distribution_data['Both Sexes']) # normalize to total number of people (sum of deat
h_distribution_data['Both Sexes'])

print(P_lh(death_distribution_data))
```

```
0.07766387615350638
```

In [219]:
```
%%nose

def test_output_type():
    assert type(P_lh(death_distribution_data)) == np.float64 or type(P_lh(death_distribution_data)) == float,
\
    'Have you defined a function called P_lh that returns the overall probability of left-handedness?'

def test_study_year():
    assert round(P_lh(death_distribution_data, 2018), 2) == 0.11, \
    'Make sure to include `study_year` as the second argument for the function P_lh()'

def test_P_lh():
    assert round(P_lh(death_distribution_data), 2) == 0.08, \
    'The overall probability of left-handedness should be approximately 0.08.'
```

Out[219]: 3/3 tests passed

6. Putting it all together: dying while left-handed (i)

Now we have the means of calculating all three quantities we need: P(A), P(LH), and P(LH | A). We can combine all three using Bayes' rule to get P(A | LH), the probability of being age A at death (in the study year) given that you're left-handed. To make this answer meaningful, though, we also want to compare it to P(A | RH), the probability of being age A at death given that you're right-handed.

We're calculating the following quantity twice, once for left-handers and once for right-handers.

$$P(A|LH) = \frac{P(LH|A)P(A)}{P(LH)}$$

First, for left-handers.

In [220]:
```
def P_A_given_lh(ages_of_death, death_distribution_data, study_year = 1990):
    """ The overall probability of being a particular `age_of_death` given that you're left-handed """
    P_A = death_distribution_data["Both Sexes"][ages_of_death]/np.sum(death_distribution_data["Both Sexes"])
    P_left = P_lh(death_distribution_data,study_year) # use P_lh function to get probability of left-handedne
ss overall
    P_lh_A = P_lh_given_A(ages_of_death,study_year) # use P_lh_given_A to get probability of left-handedness
 for a certain age
    return P_lh_A*P_A/P_left
```

In [221]:
```
%%nose

def test_output_type():
    test_input = np.array([60])
    assert (type(P_A_given_lh(test_input, death_distribution_data)) == pd.core.series.Series), \
    'Have you defined a function called P_A_given_lh that returns a pandas Series?'

def test_output_is_probability():
    test_input = np.array([60])
    assert (P_A_given_lh(test_input, death_distribution_data) < 1).all(), \
    'Make sure the function returns numbers that are less than 1.'

def test_output_sums_to_1():
    test_input = np.arange(0,115)
    assert (round(np.nansum(P_A_given_lh(test_input, death_distribution_data)), 2) == 1), \
    'P_A_given_lh(np.arange(0,115), death_distribtuion_data) should sum up to 1.'

def test_study_year():
    test_input = np.array([45])
    assert (P_A_given_lh(test_input, death_distribution_data) >
            P_A_given_lh(test_input, death_distribution_data, 2018)).all(), \
    'Make sure to include `study_year` as the third argument for the function P_A_given_lh()'
```

Out[221]: 4/4 tests passed

7. Putting it all together: dying while left-handed (ii)

And now for right-handers.

In [222]:
```
def P_A_given_rh(ages_of_death, death_distribution_data, study_year = 1990):
    """ The overall probability of being a particular `age_of_death` given that you're right-handed """
    P_A = P_A = death_distribution_data["Both Sexes"][ages_of_death]/np.sum(death_distribution_data["Both Sex
es"])
    P_right = 1-P_lh(death_distribution_data,study_year) # either you're left-handed or right-handed, so P_ri
ght = 1 - P_left
    P_rh_A = 1- P_lh_given_A(ages_of_death,study_year) # P_rh_A = 1 - P_lh_A
    return P_rh_A*P_A/P_right
```

In [223]:
```
%%nose

def test_output_type():
    test_input = np.array([60])
    assert (type(P_A_given_rh(test_input, death_distribution_data)) == pd.core.series.Series), \
    'Have you defined a function called P_A_given_rh that returns a pandas Series?'

def test_output_is_probability():
    test_input = np.array([60])
    assert (P_A_given_rh(test_input, death_distribution_data) < 1).all(), \
    'Make sure the function returns numbers that are less than 1.'

def test_output_sums_to_1():
    test_input = np.arange(0,115)
    assert (round(np.nansum(P_A_given_rh(test_input, death_distribution_data)), 2) == 1), \
    'P_A_given_rh(np.arange(0,115), death_distribtuion_data) should sum up to 1.'

def test_correct_trend():
    assert (P_A_given_lh(np.array([80]), death_distribution_data) <
    P_A_given_rh(np.array([80]), death_distribution_data)).all(), \
    'Did you mix up any components of P_A_given_lh and P_A_given_rh?'

def test_study_year():
    test_input = np.array([45])
    assert (P_A_given_rh(test_input, death_distribution_data) <
            P_A_given_rh(test_input, death_distribution_data, 2018)).all(), \
    'Make sure to include `study_year` as the third argument for the function P_A_given_rh()'
```

Out[223]: 5/5 tests passed

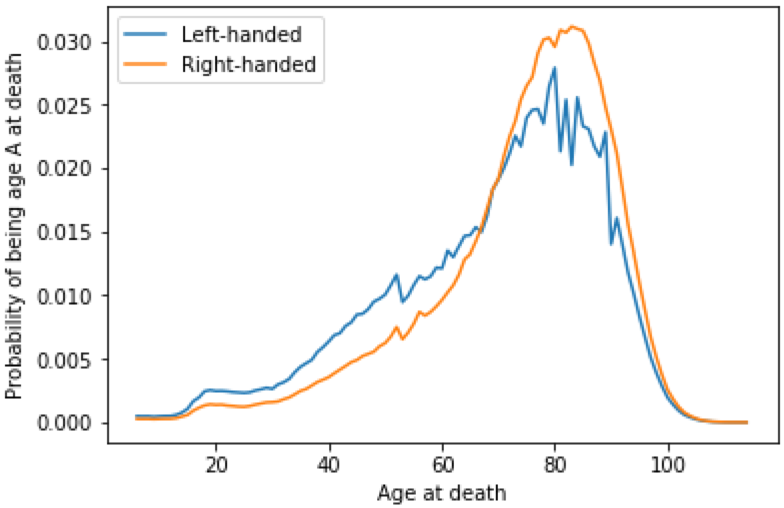
8. Plotting the distributions of conditional probabilities

Now that we have functions to calculate the probability of being age A at death given that you're left-handed or right-handed, let's plot these probabilities for a range of ages of death from 6 to 120.

Notice that the left-handed distribution has a bump below age 70: of the pool of deceased people, left-handed people are more likely to be younger.

In [224]:
```
ages = np.arange(6, 115, 1) # make a list of ages of death to plot

# calculate the probability of being left- or right-handed for each
left_handed_probability = P_A_given_lh(ages,death_distribution_data)
right_handed_probability = P_A_given_rh(ages,death_distribution_data)

# create a plot of the two probabilities vs. age
fig, ax = plt.subplots() # create figure and axis objects
ax.plot(ages, left_handed_probability, label = "Left-handed")
ax.plot(ages, right_handed_probability, label = "Right-handed")
ax.legend() # add a legend
ax.set_xlabel("Age at death")
ax.set_ylabel(r"Probability of being age A at death")
```

Out[224]:
```
Text(0,0.5,'Probability of being age A at death')
```



In [225]:
```
%%nose

def test_list_length():
    assert len(left_handed_probability) > 1 and len(right_handed_probability) > 1, \
    'Make sure you append values to each list for every age A in ages.'

def test_list_values():
    assert np.max(left_handed_probability) < 1 and np.max(right_handed_probability) < 1, \
    'All the values in each list should be probabilities (less than 1)'

def test_lists_are_different():
    assert np.max(left_handed_probability) < np.max(right_handed_probability), \
    'Did you calculate separate values for left- and right-handed probabilities?'

def test_num_lines():
    assert (len(ax.lines) == 2), \
    'Did you plot lefthanded rates for both men and women?'

def test_plot_labels():
    assert ax.get_xlabel() != '' and ax.get_ylabel() != '', \
    'Please add x and y labels to your plot.'
```

Out[225]: 5/5 tests passed

9. Moment of truth: age of left and right-handers at death

Finally, let's compare our results with the original study that found that left-handed people were nine years younger at death on average. We can do this by calculating the mean of these probability distributions in the same way we calculated P(LH) earlier, weighting the probability distribution by age and summing over the result.

$$\text{Average age of left-handed people at death} = \sum_A AP(A|LH)$$

$$\text{Average age of right-handed people at death} = \sum_A AP(A|RH)$$

In [226]:
```
# calculate average ages for left-handed and right-handed groups
# use np.array so that two arrays can be multiplied
average_lh_age =  np.nansum(ages*np.array(left_handed_probability))
average_rh_age =  np.nansum(ages*np.array(right_handed_probability))

# print the average ages for each group
print(average_lh_age)
print(average_rh_age)

# print the difference between the average ages
print("The difference in average ages is " + str(round(average_lh_age-average_rh_age, 1)) + " years.")
```

```
67.24503662801027
72.79171936526477
The difference in average ages is -5.5 years.
```

In [227]:
```
%%nose

def test_result_type():
    assert ((type(average_lh_age) == np.float64 or type(average_lh_age) == float) and
            (type(average_rh_age) == np.float64 or type(average_rh_age) == float)), \
    'average_lh_age and average_rh age should each be a single number.'

def test_average_ages():
    assert average_lh_age < average_rh_age, \
    'You should get a smaller number for average_lh_age than average_rh_age.'
```

Out[227]: 2/2 tests passed

10. Final comments

We got a pretty big age gap between left-handed and right-handed people purely as a result of the changing rates of left-handedness in the population, which is good news for left-handers: you probably won't die young because of your sinisterness. The reported rates of left-handedness have increased from just 3% in the early 1900s to about 11% today, which means that older people are much more likely to be reported as right-handed than left-handed, and so looking at a sample of recently deceased people will have more old right-handers.

Our number is still less than the 9-year gap measured in the study. It's possible that some of the approximations we made are the cause:

1. We used death distribution data from almost ten years after the study (1999 instead of 1991), and we used death data from the entire United States instead of California alone (which was the original study).
2. We extrapolated the left-handedness survey results to older and younger age groups, but it's possible our extrapolation wasn't close enough to the true rates for those ages.

One thing we could do next is figure out how much variability we would expect to encounter in the age difference purely because of random sampling: if you take a smaller sample of recently deceased people and assign handedness with the probabilities of the survey, what does that distribution look like? How often would we encounter an age gap of nine years using the same data and assumptions? We won't do that here, but it's possible with this data and the tools of random sampling.

To finish off, let's calculate the age gap we'd expect if we did the study in 2018 instead of in 1990. The gap turns out to be much smaller since rates of left-handedness haven't increased for people born after about 1960. Both the National Geographic study and the 1990 study happened at a unique time - the rates of left-handedness had been changing across the lifetimes of most people alive, and the difference in handedness between old and young was at its most striking.

In [228]:
```
# Calculate the probability of being left- or right-handed for all ages
left_handed_probability_2018 = P_A_given_lh(ages,death_distribution_data,study_year=2018)
right_handed_probability_2018 = P_A_given_rh(ages,death_distribution_data,study_year=2018)

# calculate average ages for left-handed and right-handed groups
average_lh_age_2018 = np.nansum(ages*np.array(left_handed_probability_2018))
average_rh_age_2018 = np.nansum(ages*np.array(right_handed_probability_2018))

print("The difference in average ages is " +
      str(round(average_rh_age_2018 - average_lh_age_2018, 1)) + " years.")
```

```
The difference in average ages is 2.3 years.
```

In [229]:
```
%%nose

# def test_current_year():
#     import datetime
#     now = datetime.datetime.now()
#     assert current_year >= now.year, \
#     'Did you set the "current_year" variable to the current year?'

def test_list_length():
    assert len(left_handed_probability_2018) > 1 and len(right_handed_probability_2018) > 1, \
    'Make sure you append values to each list for every age A in ages.'

def test_list_values():
    assert np.max(left_handed_probability_2018) < 1 and np.max(right_handed_probability_2018) < 1, \
    'All the values in each list should be probabilities (less than 1).'

def test_lists_are_different():
    assert (left_handed_probability_2018 != right_handed_probability_2018).all(), \
    'Did you calculate separate values for left and right-handed probabilities?'

def test_average_ages():
    assert (average_rh_age_2018 - average_lh_age_2018 < 4), \
    'The difference in ages between the left-handed and right-handed groups should be less than the value you
calculated for the 1990 study.'
```

Out[229]: 4/4 tests passed