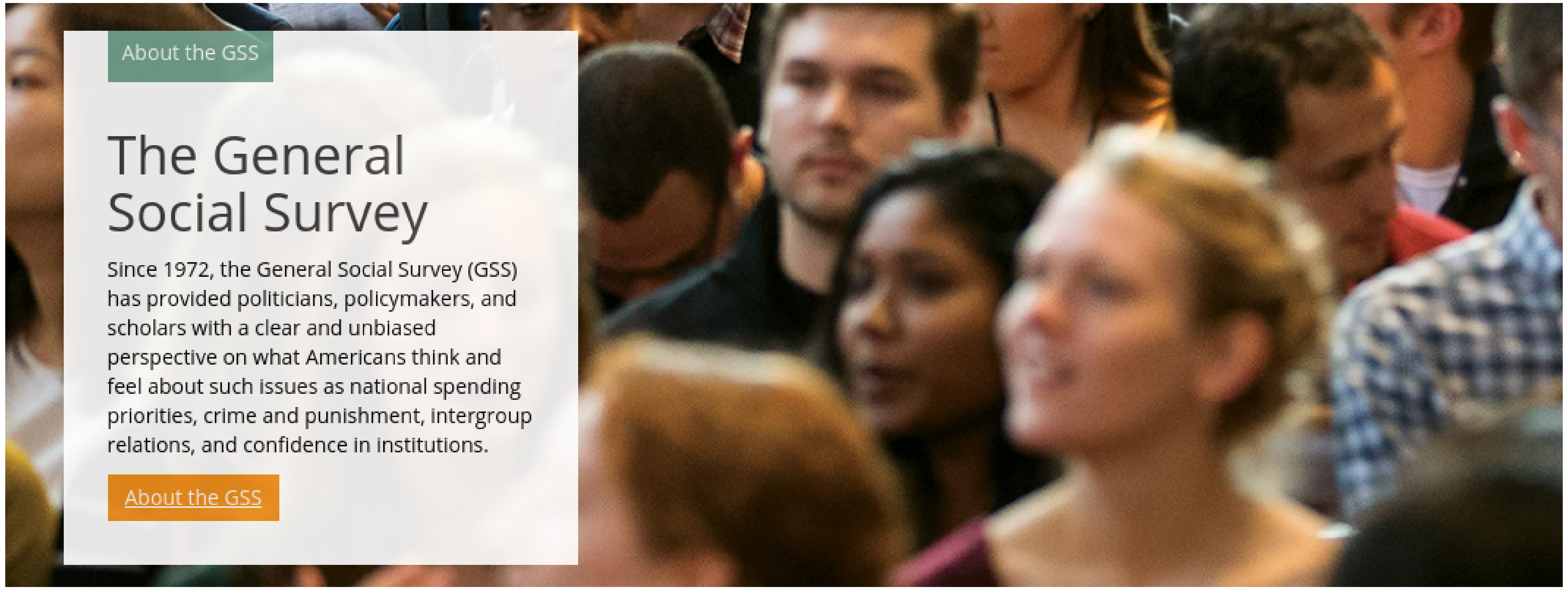
Probability mass functions

EXPLORATORY DATA ANALYSIS IN PYTHON

Allen Downey
Professor, Olin College

GSS

- Annual sample of U.S. population.
- Asks about demographics, social and political beliefs.
- Widely used by policy makers and researchers.



About the GSS

The General Social Survey

Since 1972, the General Social Survey (GSS) has provided politicians, policymakers, and scholars with a clear and unbiased perspective on what Americans think and feel about such issues as national spending priorities, crime and punishment, intergroup relations, and confidence in institutions.

About the GSS

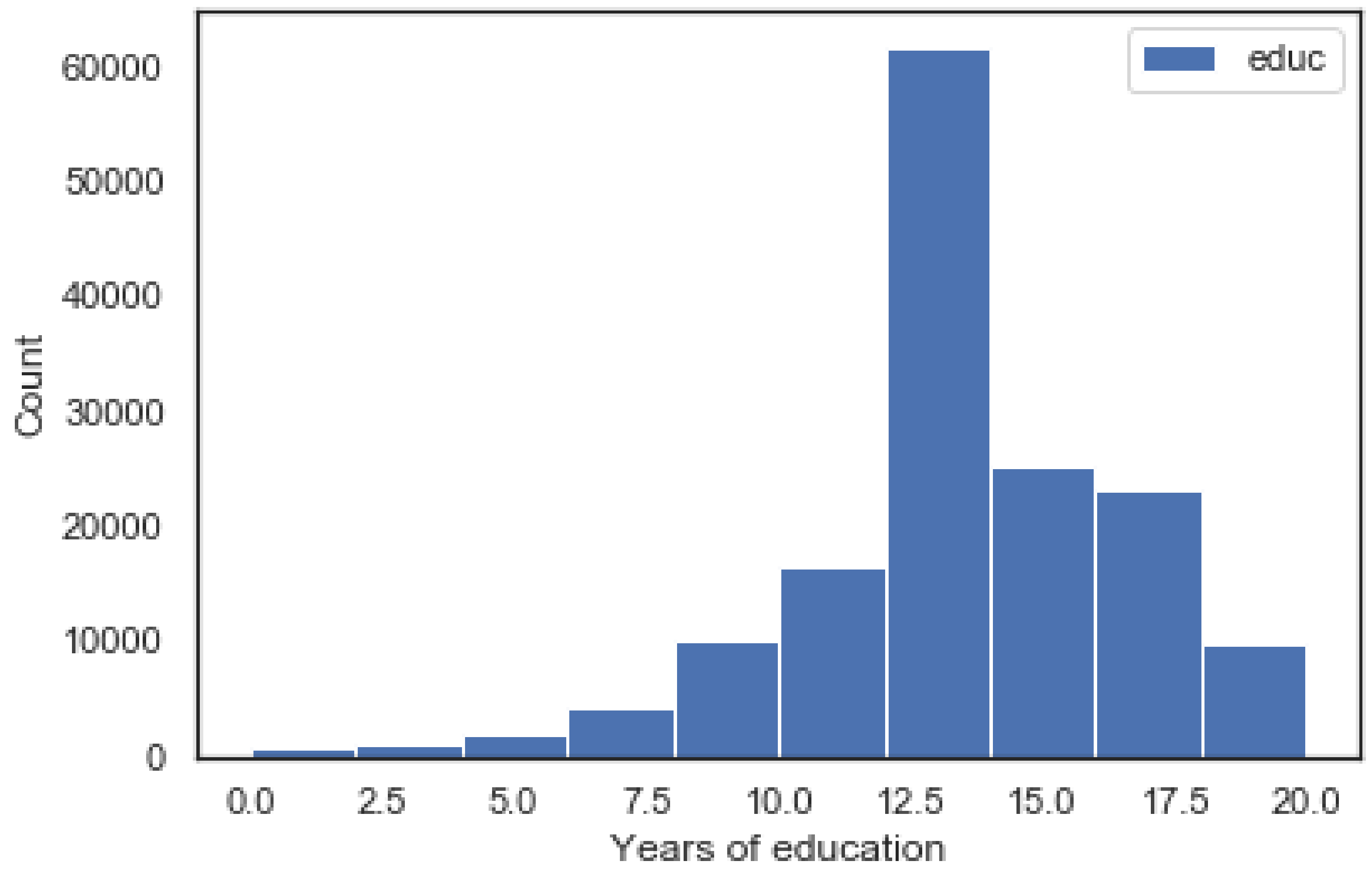
Read the data

```
gss = pd.read_hdf('gss.hdf5', 'gss')


gss.head()
```

```
   year  sex   age  cohort  race  educ  realinc  wtssall
0  1972    1  26.0  1946.0     1  18.0  13537.0   0.8893
1  1972    2  38.0  1934.0     1  12.0  18951.0   0.4446
2  1972    1  57.0  1915.0     1  12.0  30458.0   1.3339
3  1972    2  61.0  1911.0     1  14.0  37226.0   0.8893
4  1972    1  59.0  1913.0     1  12.0  30458.0   0.8893
```

```python
educ = gss['educ']
plt.hist(educ.dropna(), label='educ')
plt.show()
```



PMF

```
pmf_educ = Pmf(educ, normalize=False)
pmf_educ.head()
```

```
0.0     566
1.0     118
2.0     292
3.0     686
4.0     746
Name: educ, dtype: int64
```

5. PMF

An alternative is a probability mass function, or PMF, that contains the unique values in the dataset and how often each one appears. I've provided a class called Pmf that computes a probability mass function. This class is based on a Pandas Series and it provides some methods that aren't in Pandas. The first argument can be any kind of sequence; in this case, it's a Series object. The second argument indicates that we don't want to normalize this PMF. I'll explain what that means soon. The result is a Series that contains the values on the left and the counts on the right...

PMF

```
pmf_educ[12]
```

```
47689
```

6. PMF
You can use the bracket operator to look up a value in a Pmf and get a count. This example shows that there are 47689 respondents with 12 years of education, which is the most frequent value.

```
pmf_educ = Pmf(educ, normalize=True)


pmf_educ.head()
```

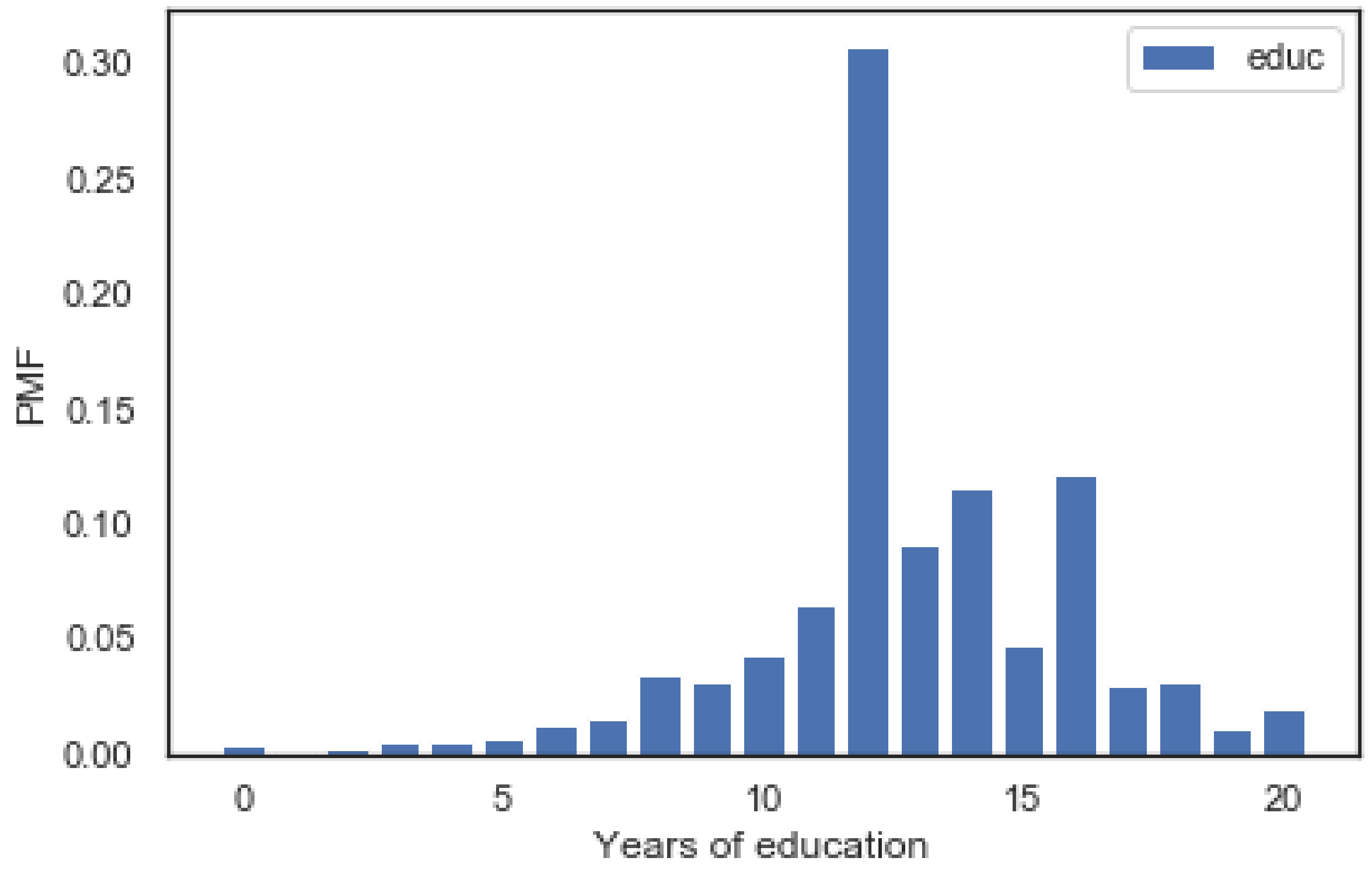
```
0.0     0.003663
1.0     0.000764
2.0     0.001890
3.0     0.004440
4.0     0.004828
Name: educ, dtype: int64
```

7. Normalize
Usually when we make a PMF, we want to know the fraction of respondents with each value, rather than the counts. We can do that by setting normalize=True; then we get a normalized PMF, that is, a PMF where the frequencies add up to 1. Now if we use the bracket operator, the result is a fraction. In this example, the fraction of people with 12 years of education is 0.3086, a little more than 30%.

```
pmf_educ[12]
```

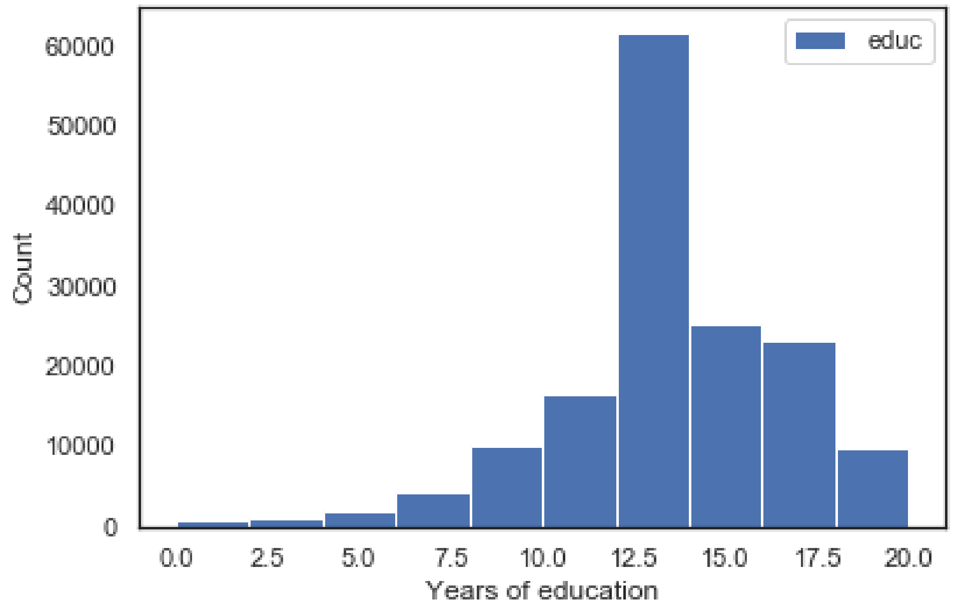
```
0.30863869940587907
```

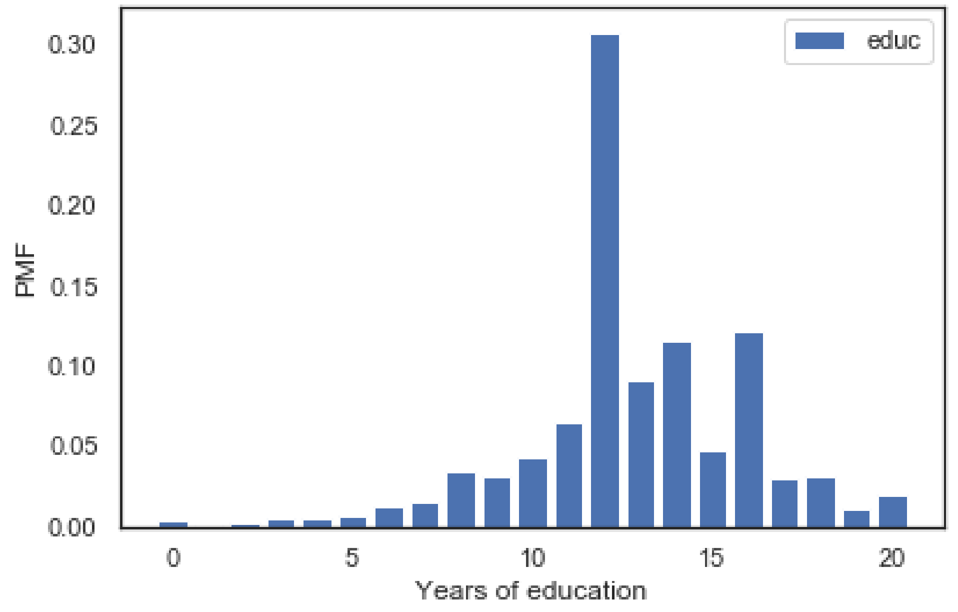
```
pmf_educ.bar(label='educ')
plt.xlabel('Years of education')
plt.ylabel('PMF')
plt.show()
```



8. Plot
The Pmf class provides a bar() method that plots a PMF as a bar chart. And here's what it looks like. In this figure, we can see more clearly that the most common value is 12 years, but there are also peaks at 14 and 16, which correspond to two and four years of college.

Histogram vs. PMF





9. Histogram vs. PMF
For this data, the PMF is probably a better choice than the histogram. The PMF shows all unique values, so we can see exactly where the peaks are. **Because the histogram puts values into bins, it obscures some details.** In this example, we can't see the peaks at 14 and 16 years.

10. Let's make some PMFs!
PMFs have limitations, too, as we'll see in the next lesson. But first, let's get some practice with PMFs.

Let's make some PMFs!

EXPLORATORY DATA ANALYSIS IN PYTHON

Cumulative distribution functions

EXPLORATORY DATA ANALYSIS IN PYTHON

Allen Downey
Professor, Olin College

From PMF to CDF

If you draw a random element from a distribution:

- PMF (Probability Mass Function) is the probability that you get exactly x
- CDF (Cumulative Distribution Function) is the probability that you get a value <= x

for a given value of x.

2. From PMF to CDF
You might remember that a PMF tells you - if you draw a random value from a distribution - what's the chance of getting x, for any given value of x. The CDF is similar; if you draw a random value from a distribution, it tells you the chance of getting a value less than or equal to x.

Example

PMF of {1, 2, 2, 3, 5}

PMF(1) = 1/5

PMF(2) = 2/5

PMF(3) = 1/5

PMF(5) = 1/5

CDF is the cumulative sum of the PMF.

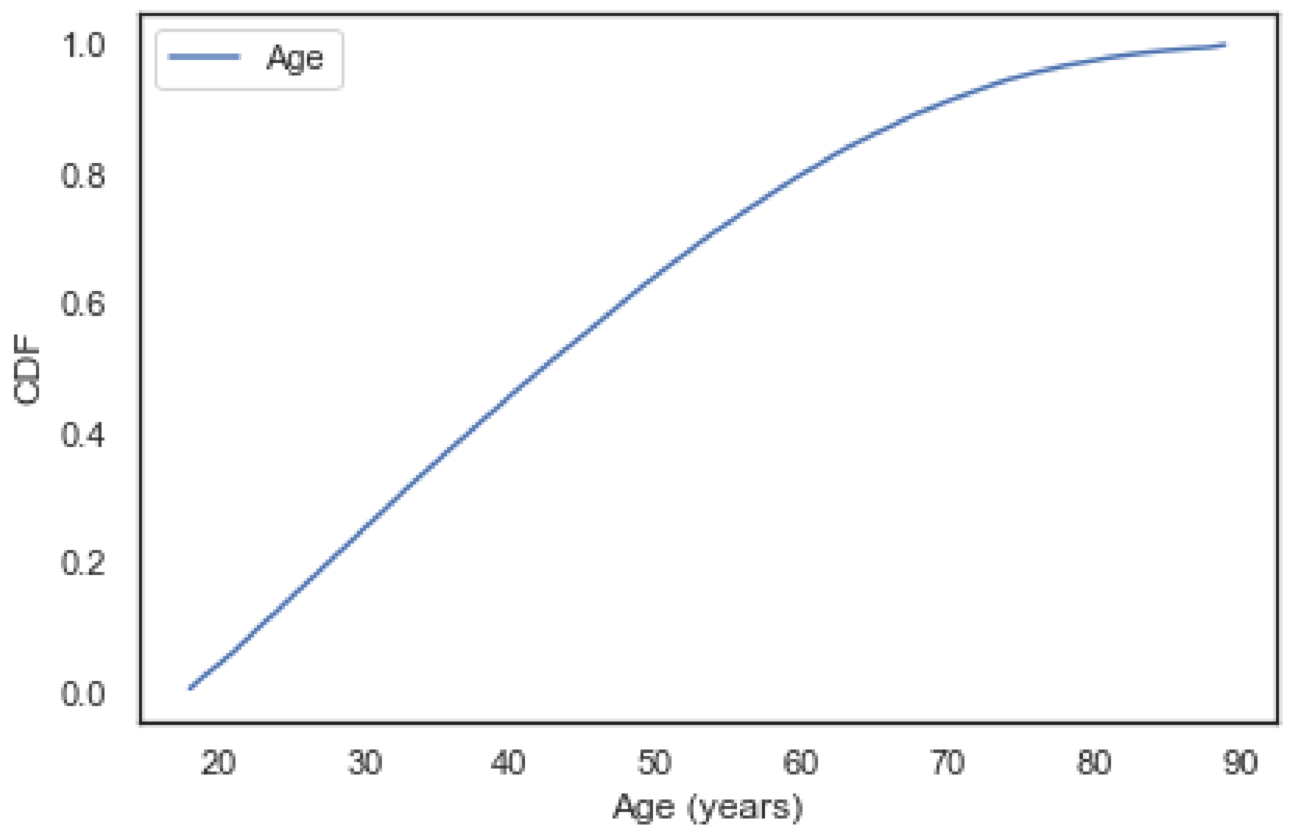
CDF(1) = 1/5

CDF(2) = 3/5

CDF(3) = 4/5

CDF(5) = 1

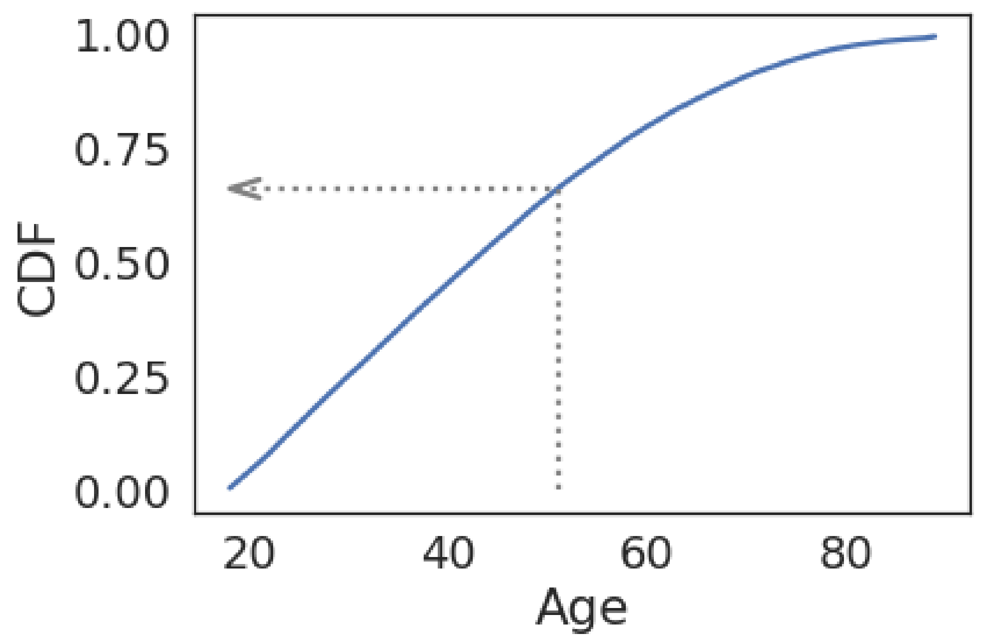
```
cdf = Cdf(gss['age'])
cdf.plot()
plt.xlabel('Age')
plt.ylabel('CDF')
plt.show()
```



Evaluating the CDF

```
q = 51
p = cdf(q)
print(p)
```

```
0.66
```



5. Evaluating the CDF
The Cdf object can be used as a function, so if you give it an age, it returns the corresponding probability. In this example, the age is the quantity, q, which is 51. The corresponding probability is p, which is 0.66. That means that about 66% of the respondents are 51 years old or younger. The arrow in the figure shows how you could read this value from the CDF, at least approximately.

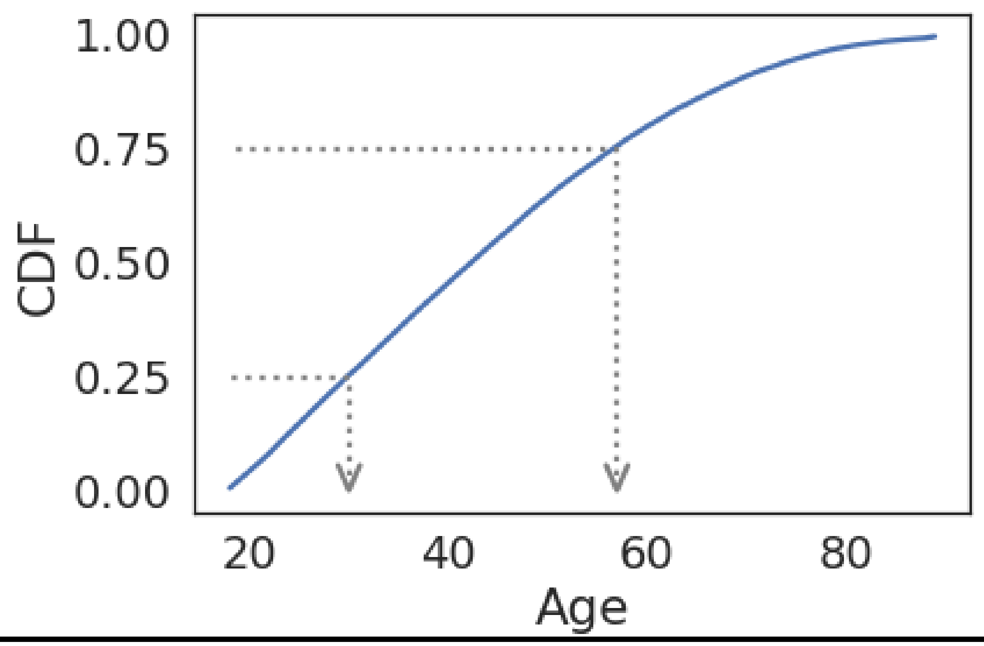
Evaluating the inverse CDF

```python
p = 0.25
q = cdf.inverse(p)
print(q)
```

```
30
```

```python
p = 0.75
q = cdf.inverse(p)
print(q)
```

```
57
```



6. Evaluating the inverse CDF
The CDF is an invertible function, which means that if you have a probability, p, you can look up the corresponding quantity, q. In this example, I look up the probability 0.25, which returns 30. That means that 25% of the respondents are age 30 or less. Another way to say the same thing is "age 30 is the 25th percentile of this distribution". I also look up probability 0.75, which returns 57, so 75% of the respondents are 57 or younger. Again, the arrows in the figure show how you could read these values from the CDF. By the way, the distance from the 25th to the 75th percentile is called the interquartile range, or IQR. It measures the spread of the distribution, so it is similar to standard deviation or variance. Because it is based on percentiles, it doesn't get thrown off by extreme values or outliers, the way variance does. So IQR can be more "robust" than variance, which means it works well even if there are errors in the data or extreme values.

Let's practice!

EXPLORATORY DATA ANALYSIS IN PYTHON

Comparing distributions

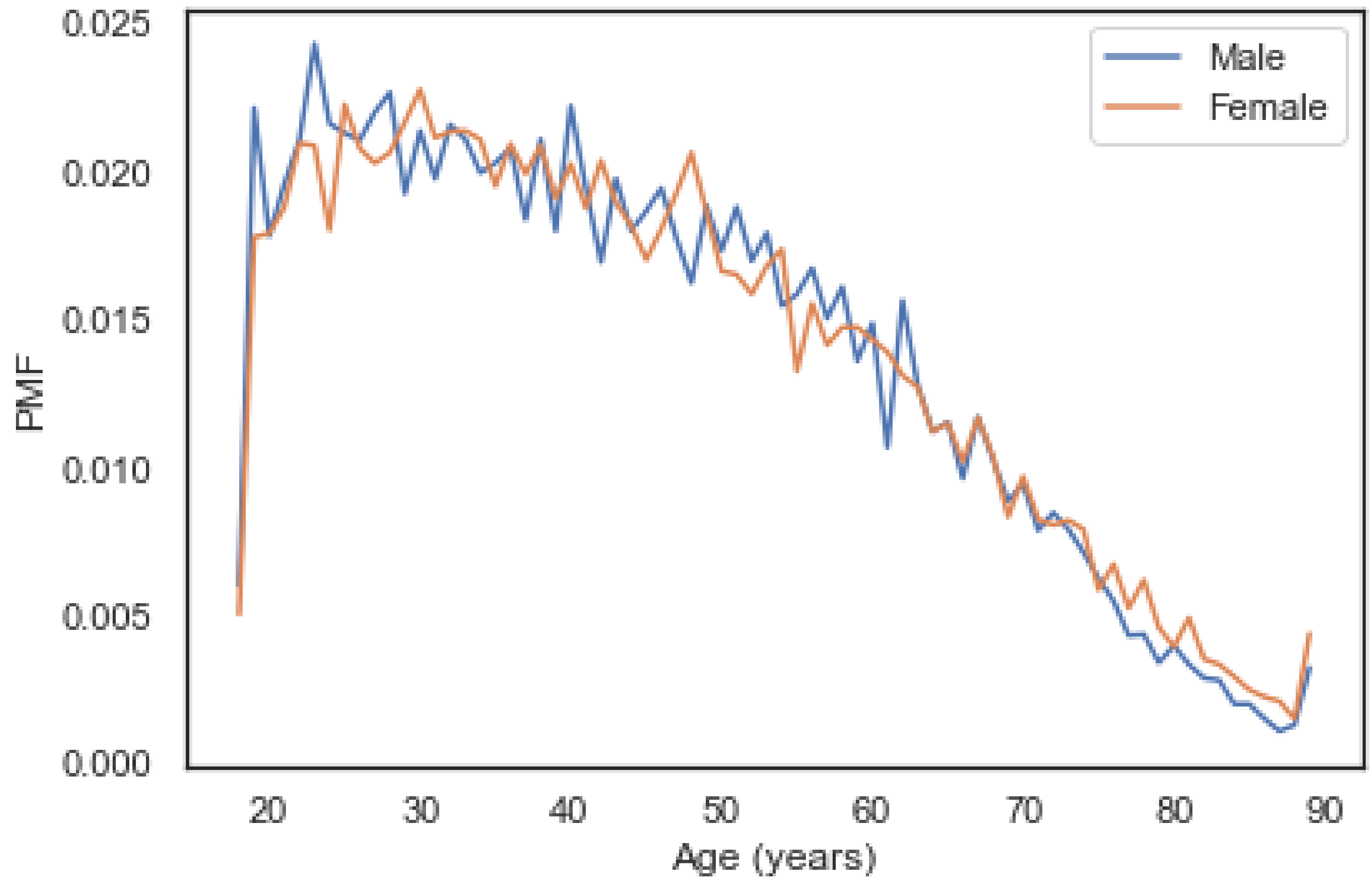
EXPLORATORY DATA ANALYSIS IN PYTHON

Allen Downey
Professor, Olin College

Multiple PMFs

```python
male = gss['sex'] == 1
age = gss['age']
male_age = age[male]
female_age = age[~male]
Pmf(male_age).plot(label='Male')
Pmf(female_age).plot(label='Female')
plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.show()
```

2. Multiple PMFs
One way to compare distributions is to plot multiple PMFs on the same axes. For example, suppose we want to compare the distribution of age for male and female respondents. First I'll create a boolean Series that's true for male respondents. And I'll extract the age column. Now I can select ages for the male and female respondents. And plot a Pmf for each. Of course I always remember to label the axes!



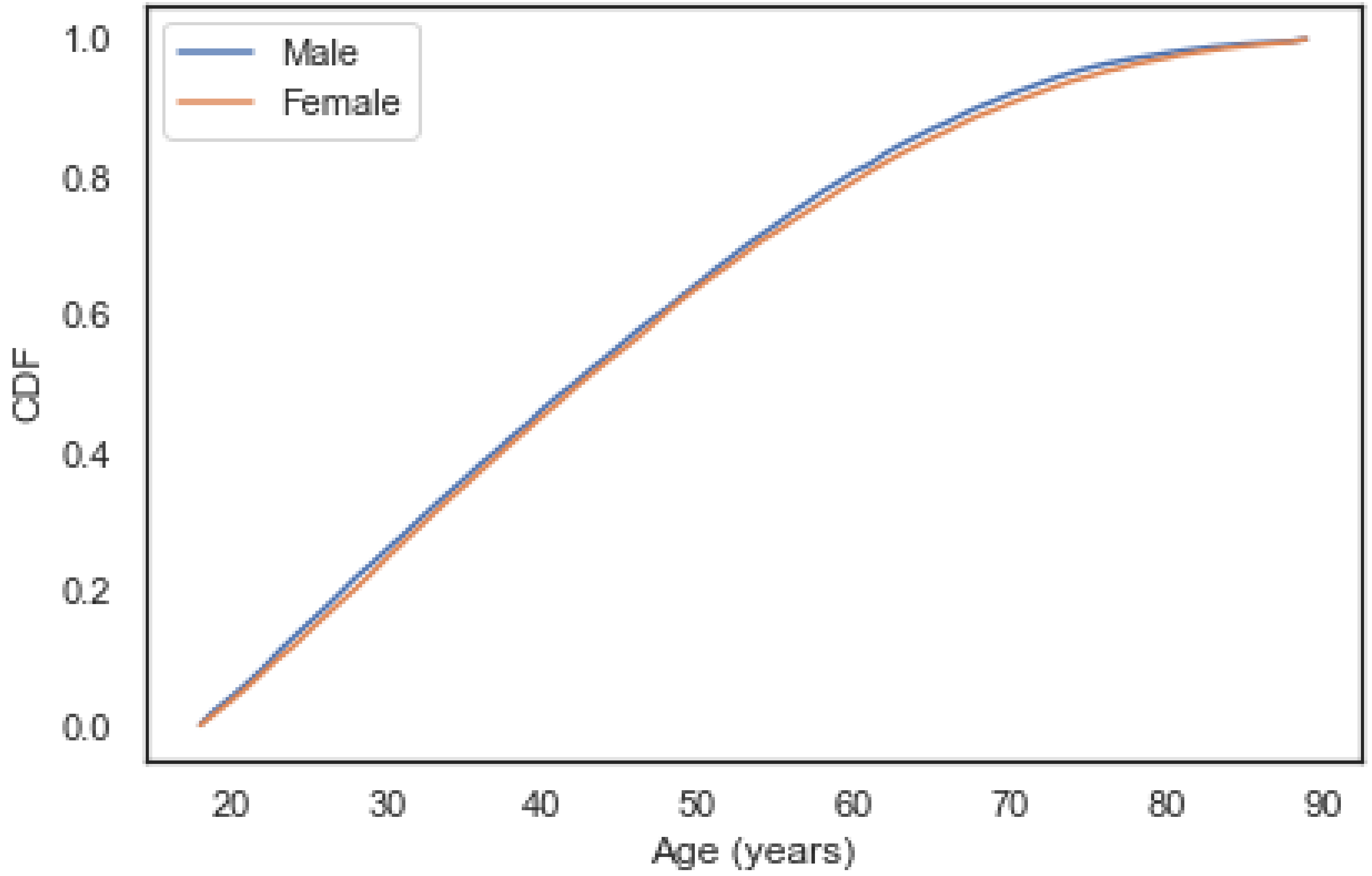
3. Age PMFs
Here's the result. It looks like there are more men in their twenties, maybe. And there are more women in their 70s and 80s. In between, the plot is pretty noisy; most of these differences are just random variations.

Multiple CDFs

```
Cdf(male_age).plot(label='Male')
Cdf(female_age).plot(label='Female')

plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.show()
```

4. Multiple CDFs
We can do the same thing with CDFs. Here's the code: everything is the same except I replaced Pmf with Cdf.

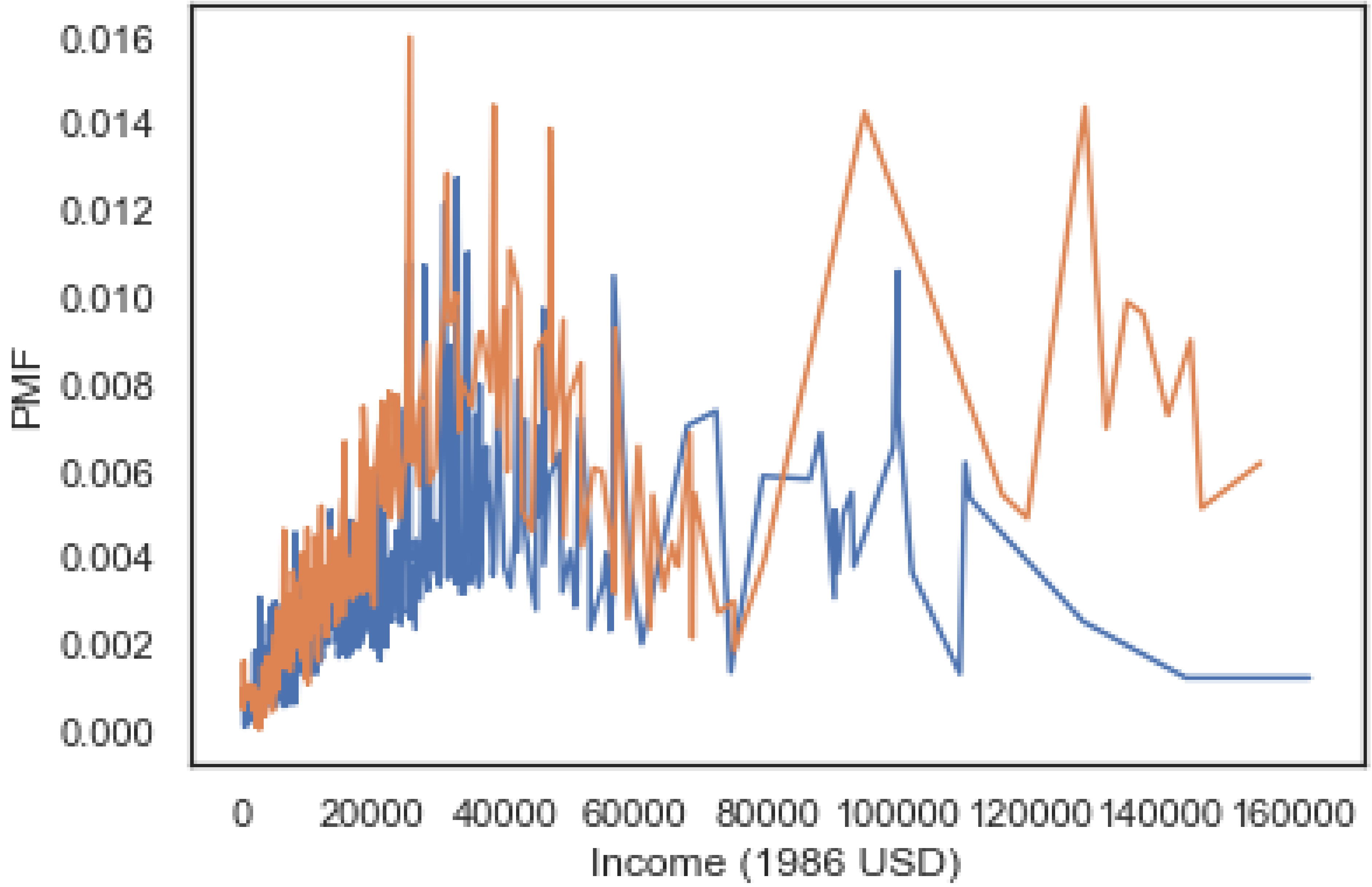


5. Age CDFs
And here is the result. In general, CDFs are smoother than PMFs. Because they smooth out randomness, we can often get a better view of real differences between distributions. In this case, the lines overlap over the whole range; that is, the distributions are nearly identical. But we can see the blue line to the left of the orange line across the distribution, which shows that men are younger at every percentile. Or, another way to think of it: for every age, the fraction of men below that age is more than the fraction of women below that age. But not by very much.

Income distribution

```
income = gss['realinc']
pre95 = gss['year'] < 1995
Pmf(income[pre95]).plot(label='Before 1995')
Pmf(income[~pre95]).plot(label='After 1995')
plt.xlabel('Income (1986 USD)')
plt.ylabel('PMF')
plt.show()
```

6. Income distribution
As another example, let's look at household income and compare the distribution before and after 1995 (I chose 1995 because it's roughly the midpoint of the survey). The variable realinc represents household income in 1986 dollars. I'll make a boolean Series to select respondents interviewed before 1995. Now I can plot the PMFs. And label the axes.



7. Income PMFs

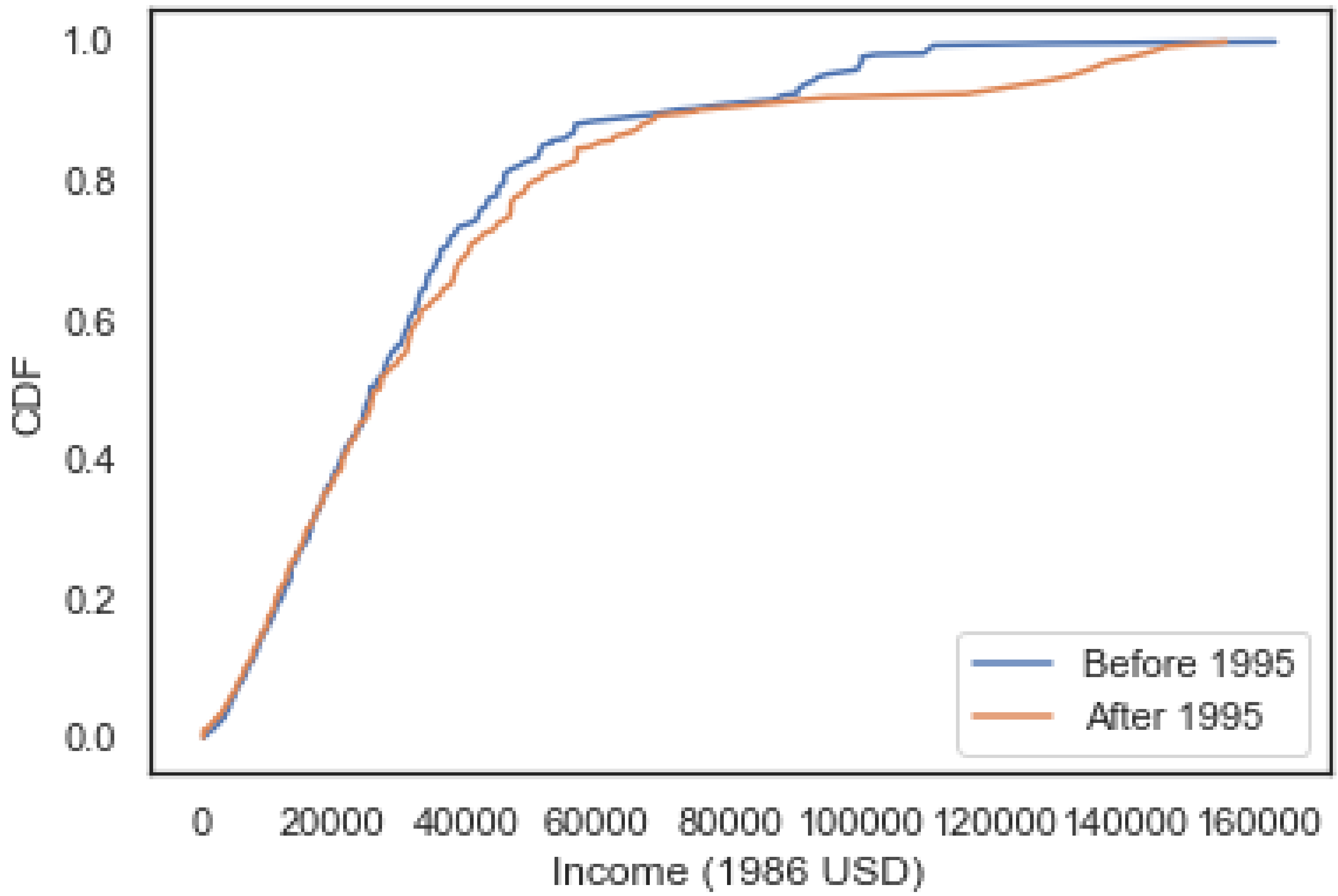
Here's what it looks like. There are a lot of unique values in this distribution, and none of them appear very often. **The PMF is so noisy, we can't really see the shape of the distribution.** It looks like there are more people with high incomes after 1995, but it's hard to tell. We can get a clearer picture with a CDF.

Income CDFs

```
Cdf(income[pre95]).plot(label='Before 1995')
Cdf(income[~pre95]).plot(label='After 1995')
```

8. Income CDFs
Here's the code to generate the CDFs.

9. Income CDFs
And here are the results. Below $30,000 the CDFs are almost identical; above that, we can see that the orange distribution is shifted to the right. In other words, the fraction of people with high incomes is about the same, but the income of high earners has increased.



10. Let's practice!
In general, I recommend CDFs for exploratory analysis. They give you a clear view of the distribution, without too much noise, and they are good for comparing distributions, especially if you have more than two. In the exercises for this lesson, you'll have a chance to compare incomes for respondents with different education levels.

Let's practice!

EXPLORATORY DATA ANALYSIS IN PYTHON

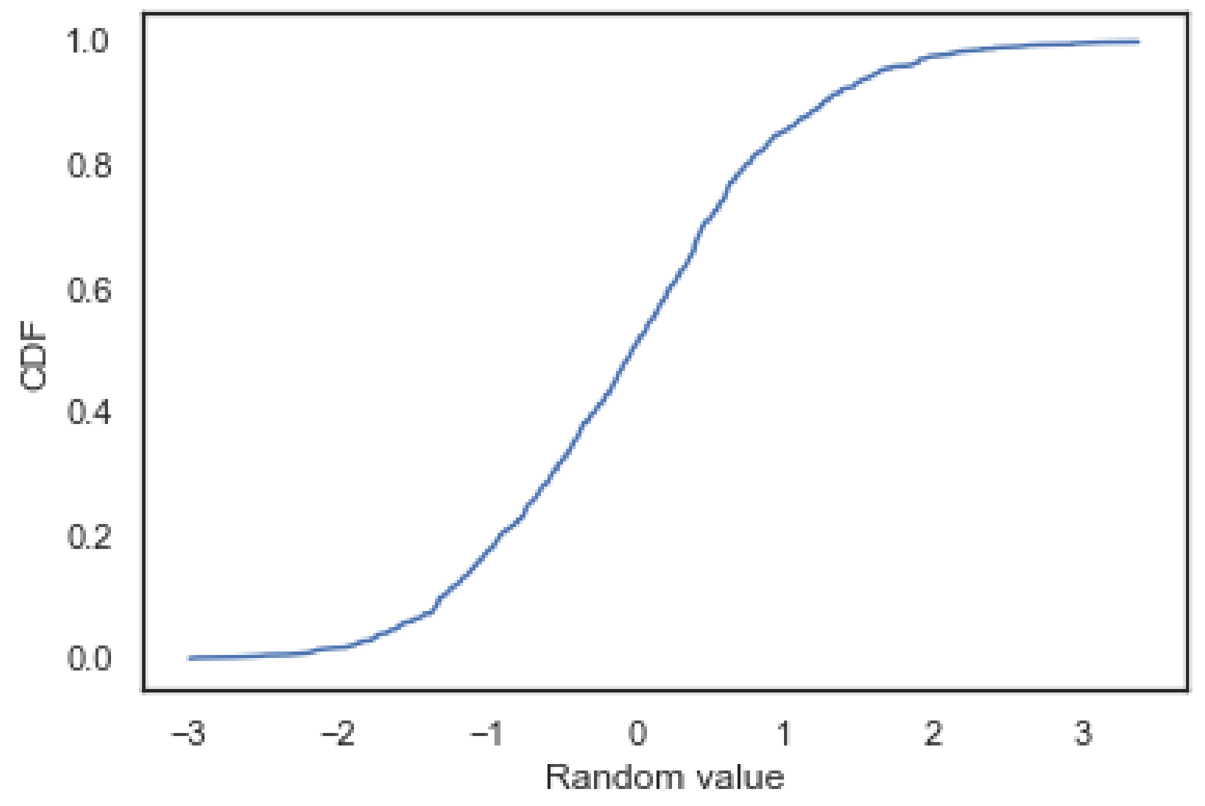
Modeling distributions

EXPLORATORY DATA ANALYSIS IN PYTHON

Allen Downey
Professor, Olin College

The normal distribution

```
sample = np.random.normal(size=1000)
Cdf(sample).plot()
```



2. The normal distribution
Let's start with the normal distribution, also called the Gaussian distribution. It turns out that many things we measure in the world have distributions that are well approximated by a normal distribution. To get a sense of what the normal distribution looks like, I'll use the NumPy random module to generate 1000 random values from a normal distribution. And plot their CDF. This sigmoid shape is what we expect to see with values from a normal distribution.

The normal CDF

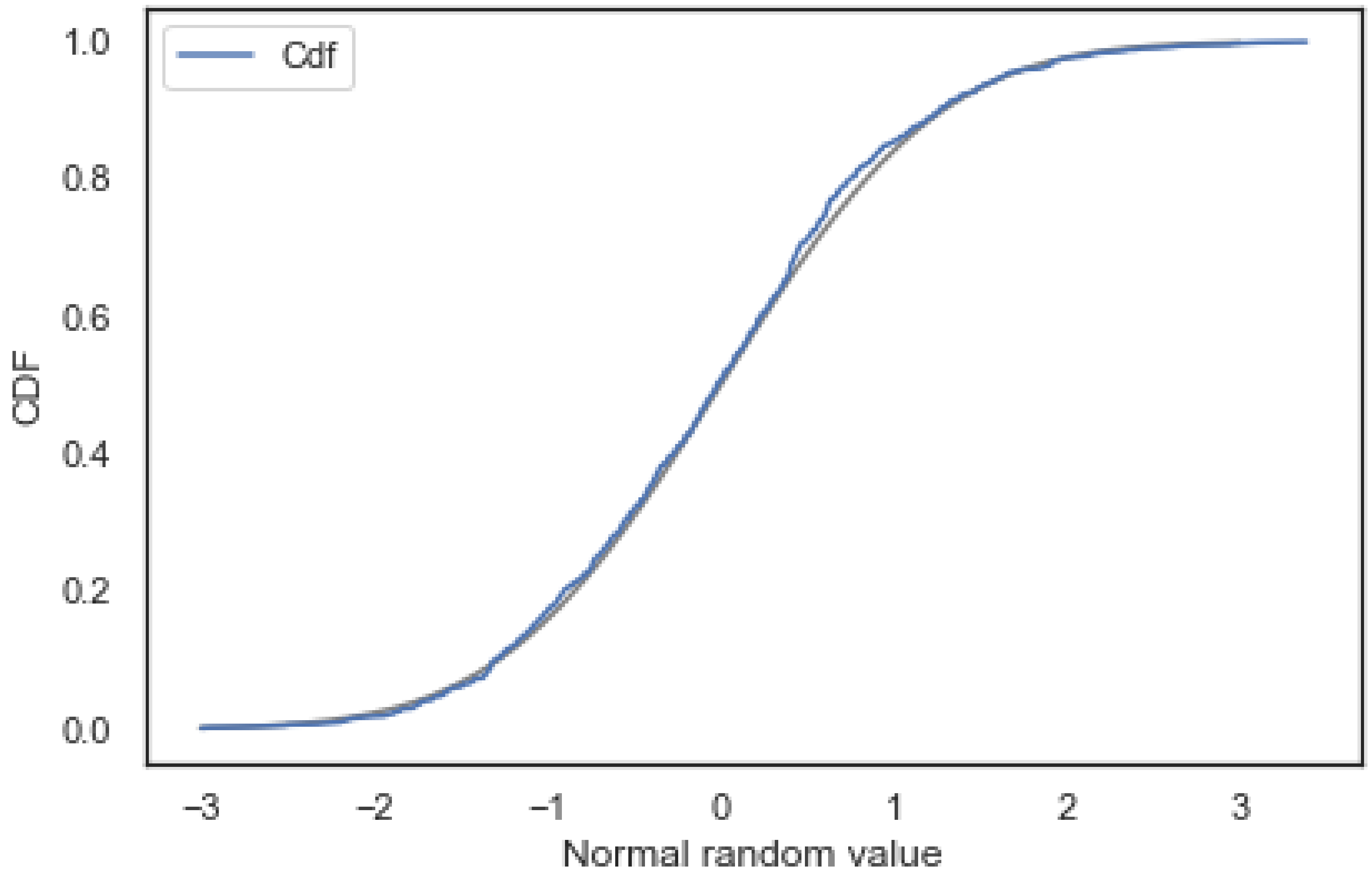
```
from scipy.stats import norm
```

```
xs = np.linspace(-3, 3)
ys = norm(0, 1).cdf(xs)
```

```
plt.plot(xs, ys, color='gray')
```

```
Cdf(sample).plot()
```

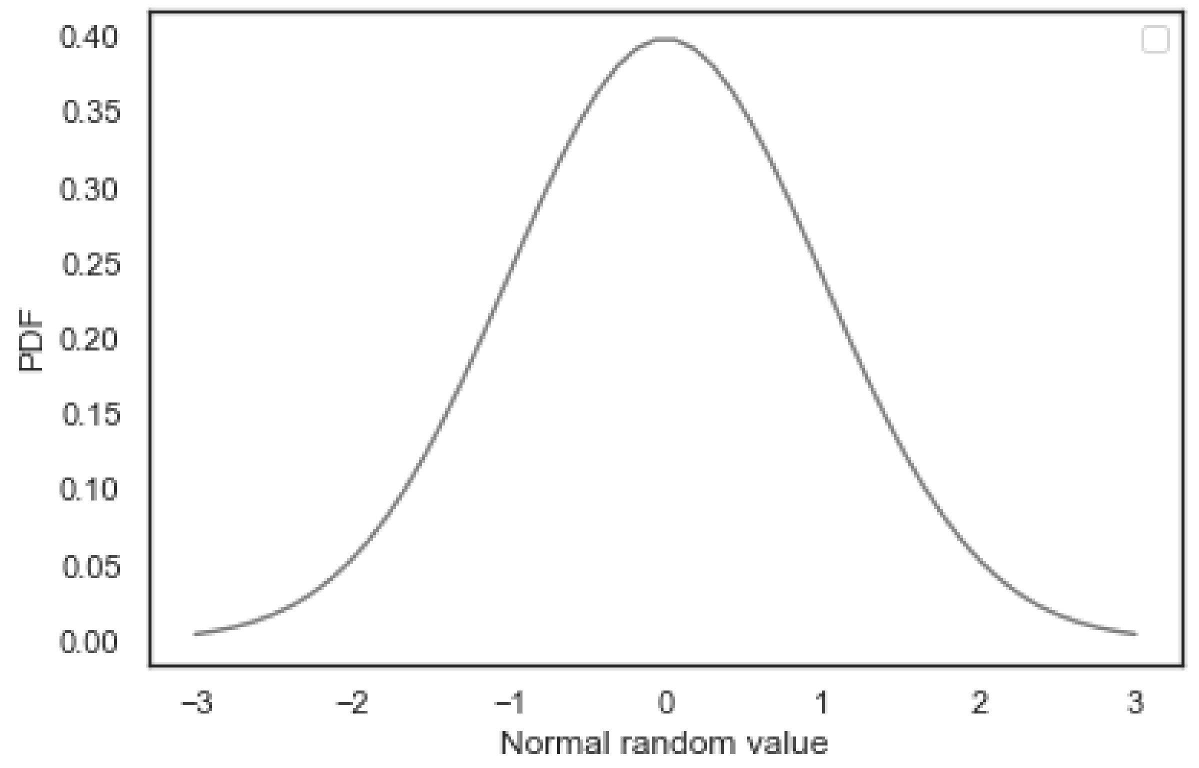
3. The normal CDF
SciPy provides an object called norm that represents the normal distribution. I'll use np dot linspace() to create an array of equally-spaced points from -3 to 3. norm(0, 1) creates an object that represents a normal distribution with mean 0 and standard deviation 1. Dot cdf() evaluates the CDF of the normal distribution. I'll plot the results with a gray line. And then plot the CDF of the sample again.



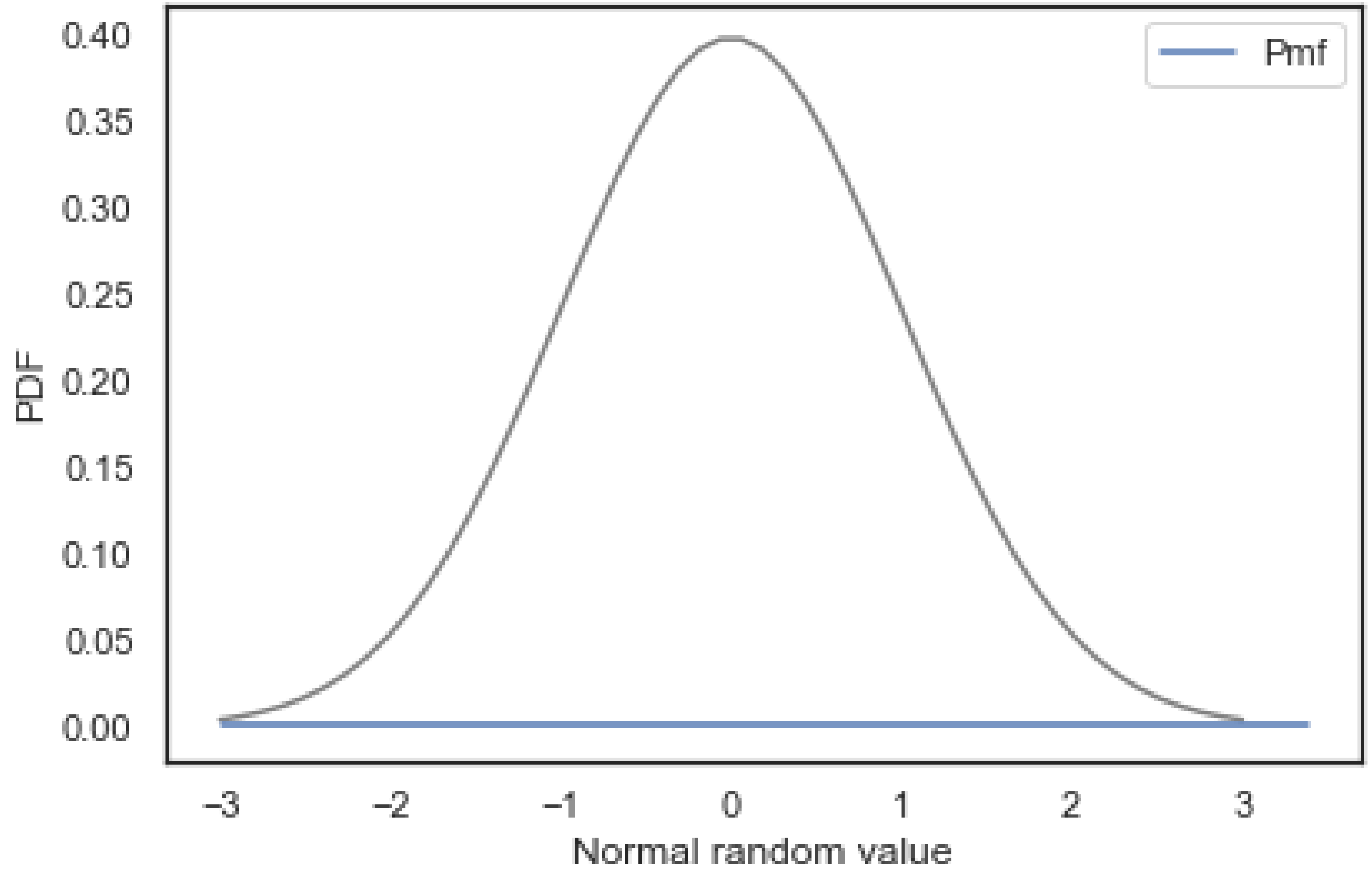
4. Comparing CDFs
Here's the result. The CDF from the sample is pretty close to the normal CDF over the whole range. If this were real data, we would conclude that the normal distribution is a good model for the data.

The bell curve

```
xs = np.linspace(-3, 3)
ys = norm(0,1).pdf(xs)
plt.plot(xs, ys, color='gray')
```



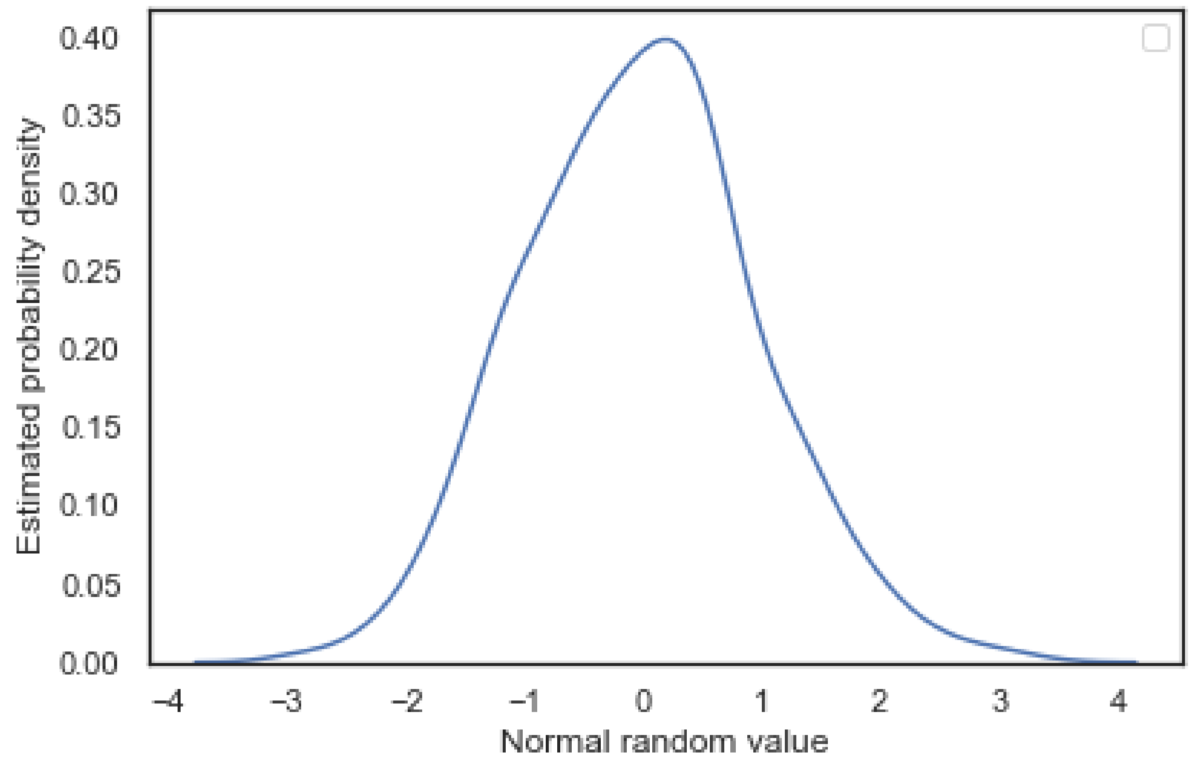
5. The bell curve
The norm object also provides dot pdf(), which evaluates the probability density function, or PDF. And here's what that looks like. It's the classic bell curve. **Unfortunately, if we compare this PDF to the PMF of the sample, it doesn't work very well.**



6. Sample PMF
Here's what it looks like. The PMF of the sample is a flat line across the bottom. In the random sample, every value is unique, so they all have the same probability, one in 1000. However, we can use the points in the sample to estimate the PDF of the distribution they came from. This process is called kernel density estimation, or KDE. It's a way of getting from a PMF, a probability mass function, to a PDF, a probability density function.

KDE plot

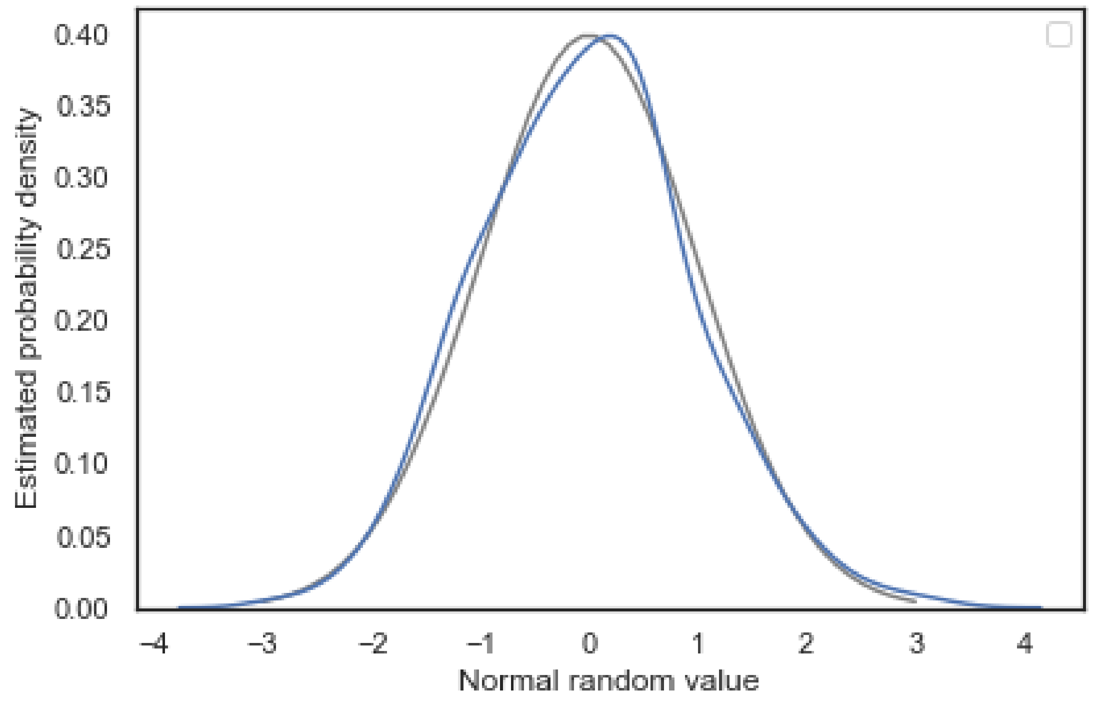
```python
import seaborn as sns
sns.kdeplot(sample)
```



7. KDE plot
To generate a KDE plot, we'll use the Seaborn library for data visualization, which I import as sns. Seaborn provides kdeplot, which takes the sample, estimates the PDF, and plots it. Here's what it looks like.

KDE and PDF

```
xs = np.linspace(-3, 3)
ys = norm.pdf(xs)
plt.plot(xs, ys, color='gray')
sns.kdeplot(sample)
```



8. KDE and PDF
Now we can compare the KDE plot and the normal PDF. Here's the code. There's nothing there we haven't already seen. And here's the result. The KDE plot matches the normal PDF pretty well, although the differences look bigger when we compare PDFs than they did with the CDFs. On one hand, that means that the PDF is a more sensitive way to look for differences, but often it is too sensitive. It's hard to tell whether apparent differences mean anything, or if they are just random, as in this case.

PMF, CDF, KDE

- Use CDFs for exploration.
- Use PMFs if there are a small number of unique values.
- Use KDE if there are a lot of values.

9. PMF, CDF, KDE
To summarize, we've seen three ways to visualize distributions, PMFs, CDFs, and KDE. In general, I use CDFs when I am exploring data. I think they give the best view of what's going on without getting distracted by noise. The biggest drawback of CDFs is that they are less well known. If I am presenting results to an audience unfamiliar with CDFs, I'll use a PMF for distributions with a small number of unique values and KDE if there are a lot of values.

Let's practice!

EXPLORATORY DATA ANALYSIS IN PYTHON