Kernel Density Estimates (KDEs) and generative models in Python

Interactive Lecture

Smith College

Dr Meridith Joyce

Marie Curie Widening Fellow: MATISSE
CSFK Konkoly Observatory, Budapest
MESA Developers

@MeridithJoyceGR
www.meridithjoyce.com
github.com/mjoyceGR

Big Picture:

Given an observed distribution of stellar ages in the center of our Galaxy, we would like to predict the distribution of stellar ages in other Galaxies

Big Picture:

Given an observed distribution of stellar ages in the center of our Galaxy, we would like to predict the distribution of stellar ages in other Galaxies

In this interactive lecture, we will apply a *kernel density estimate* to a measured stellar age distribution and use this to make synthetic data sets from a *generative model*

Step 0: Bookkeeping

There are several ways to access and run these exercises

Method 1: Github

(1) go to https://github.com/mjoyceGR/KDE_exercise

(2) download and save
  KDE_exercises.ipynb
  KDE_full_solutions.ipynb
  stellar_ages.dat
in the same folder on your computer

(3) Load the two ipynb files into your preferred Jupyter Notebook environment. You will want both accessible so you can consult the solutions if you get stuck

Step 0: Bookkeeping

Method 2: Google Drive

(1) Scan this QR code, save the link, then open it on the device you will use to code

You should see a Google Drive folder called Smith_KDE_Exercises

(2) download and save

- **KDE_exercises.ipynb**
- **KDE_full_solutions.ipynb**
- stellar_ages.dat

Step 0: Bookkeeping

Method 2: Google Drive

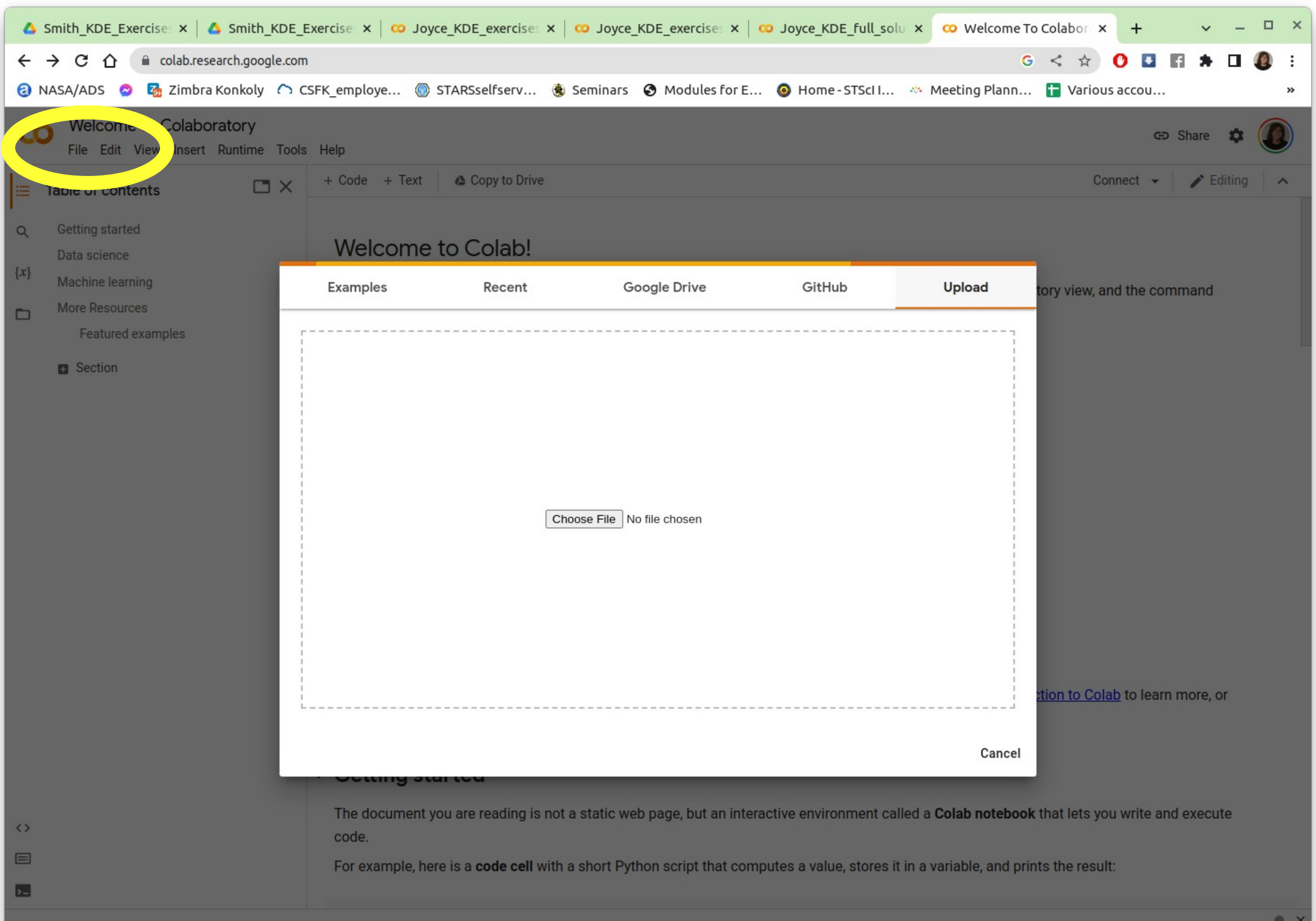
(2) Go to
https://colab.research.google.com/

(i) Under “File” in the top left, click “open notebook”
(ii) Click “Upload” on the far right of the menu that pops up
(iii) Upload your downloaded copy of **KDE_exercises.ipynb**

Do (i- iii) for **KDE_full_solutions.ipynb** in a different window

Step 0: Bookkeeping

https://colab.research.google.com/



KDE_exercises.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:03 PM

Step 1: Load the modules we will need

```
#!/usr/bin/env python3
############################
#
# template by M Joyce
# for use with Smith College students
#
############################

## import the modules
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy import stats
from scipy.stats import norm

print("modules imported")
```

Is everyone on this page?

Step 2: Define a function to make figures look nice

```
[ ] def set_fig(ax):
        ax.tick_params(axis = 'both',which='both', width=2)
        ax.tick_params(axis = 'both',which='major', length=12)
        ax.tick_params(axis = 'both',which='minor', length=8, color='black')
        ax.tick_params(axis='both', which='major', labelsize=24)
        ax.tick_params(axis='both', which='minor', labelsize=20)
        return

    print("plot settings function defined")
```

Step 3: Load the data

Pause

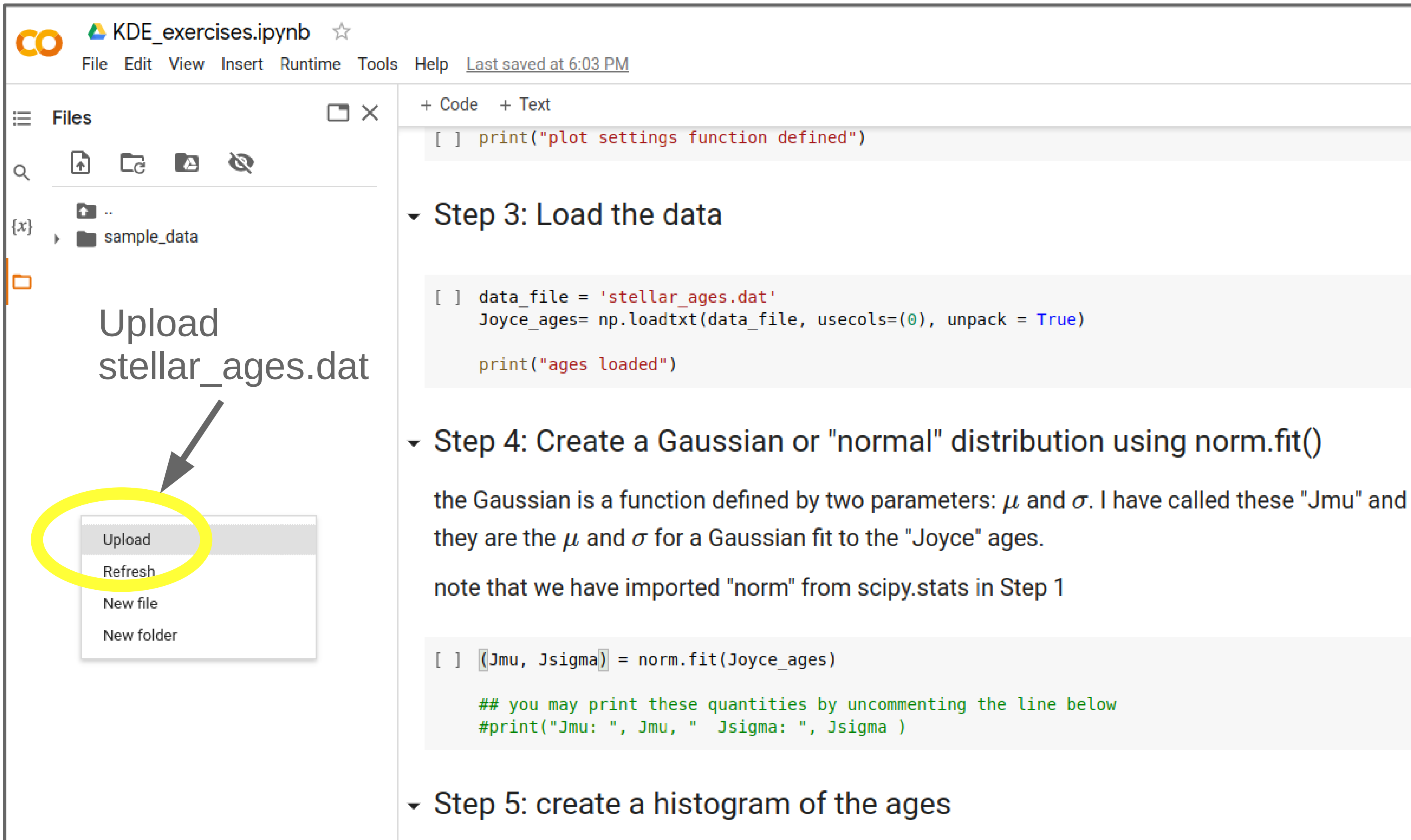
Step 1: Load the modules

Step 2: define a figure function set_fig()

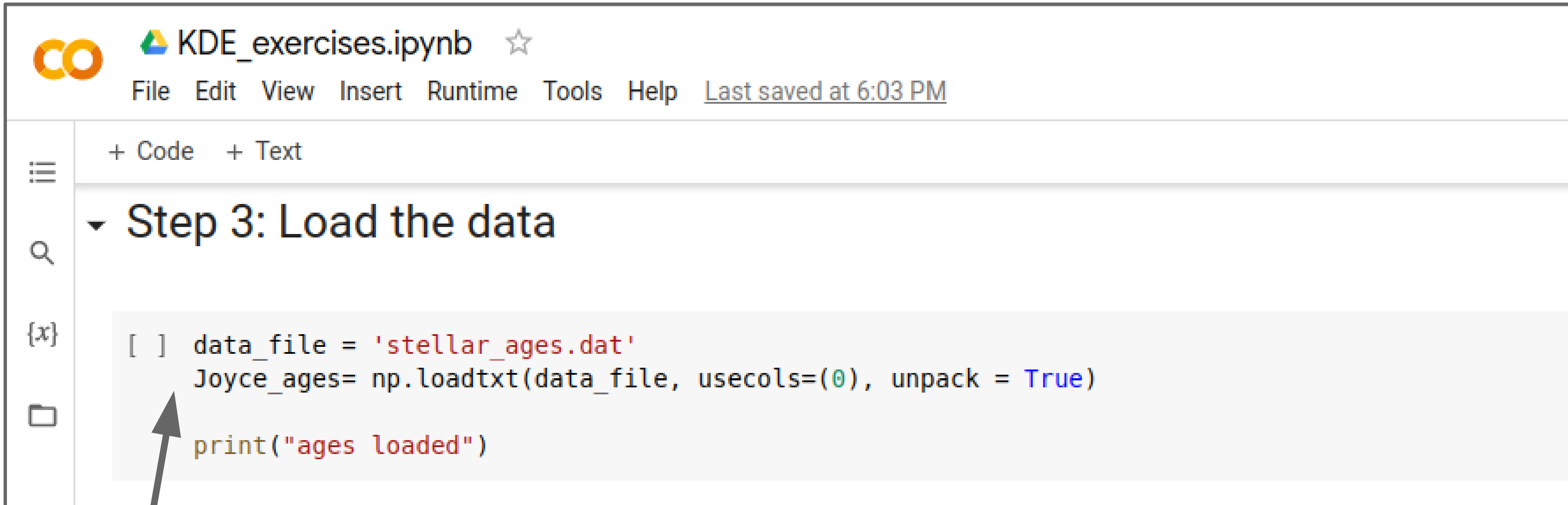
You do not need to understand how this works, just that it makes the figures look nice

Step 3: Load the data

The next cell won't work because we have not uploaded stellar_ages.dat to colab



Step 3: (now) Load the data



The data we are loading is a set of 91 stellar age determinations, measured in Gigayears (1 Gyr = 1 billion years = 10^9 years)

These are real data from my research, hence “Joyce ages”

Step 3: (now) Load the data

KDE_exercises.ipynb

File Edit View Insert Runtime Tools Help Last saved at 6:03 PM

+ Code + Text

Step 3: Load the data

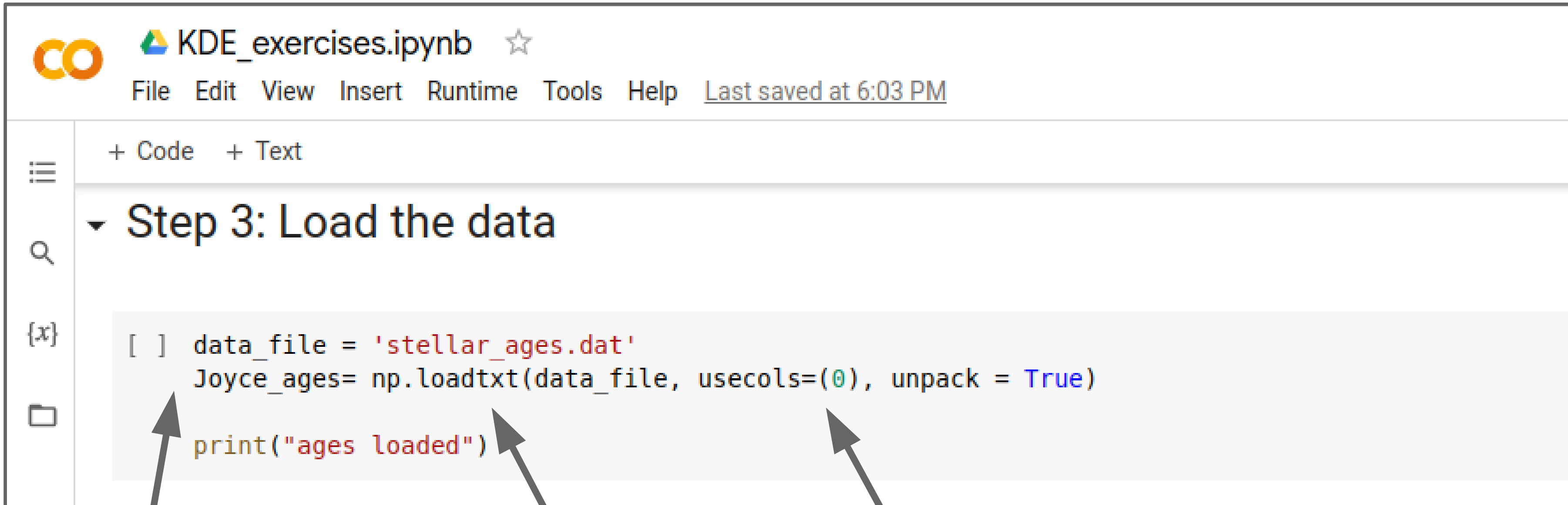
```
[ ] data_file = 'stellar_ages.dat'
    Joyce_ages= np.loadtxt(data_file, usecols=(0), unpack = True)

    print("ages loaded")
```

The data we are loading is a set of 91 stellar age determinations, measured in Gigayears (1 Gyr = 1 billion years = 10^9 years)

These are real data from my research, hence “Joyce ages”

We imported numpy as “np”
We are using a numpy function called “loadtxt” which automatically converts columns into np arrays

Step 3: (now) Load the data



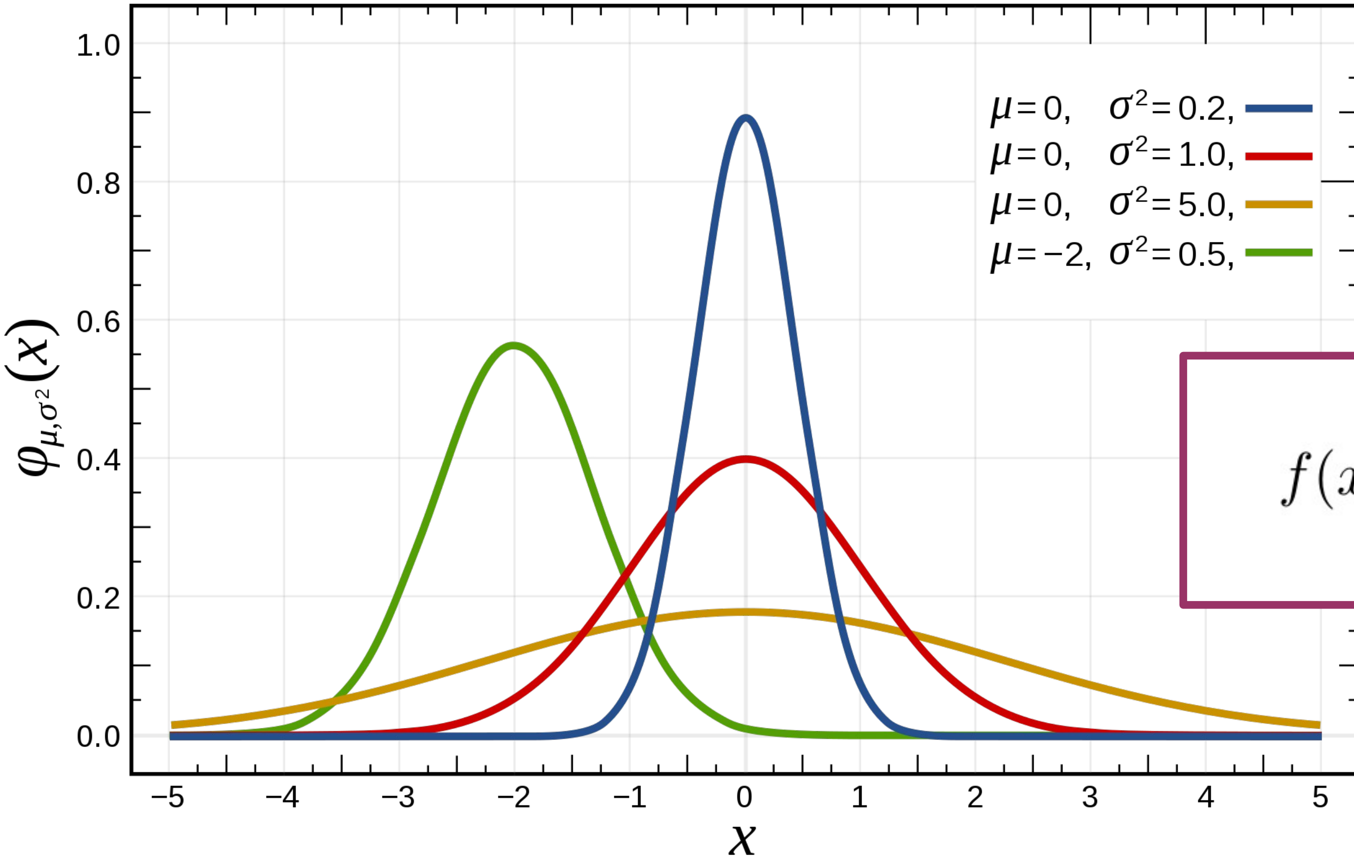
Python indexes from zero, so the first column of stellar_ages.dat is "column 0"

The data we are loading is a set of 91 stellar age determinations, measured in Gigayears (1 Gyr = 1 billion years = 10^9 years)

These are real data from my research, hence "Joyce ages"

We imported numpy as "np"
We are using a numpy function called "loadtxt" which automatically converts columns into np arrays

Step 4: Create a Gaussian



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

- the shape of the curve is captured by **mu**, the "expected value," or mean, and **sigma**, which is related to the width and represents one standard deviation (sigma^2 is the "variance," as shown in the legend)

- the type of distribution everyone (in astronomy) assumes discrete random variables follow

- also called a "normal distribution"

Step 4: Create a Gaussian

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Step 4: Create a Gaussian or "normal" distribution using norm.fit()

the Gaussian is a function defined by two parameters: μ and σ. I have called these "Jmu" and "Jsigma" because they are the μ and σ for a Gaussian fit to the "Joyce" ages.

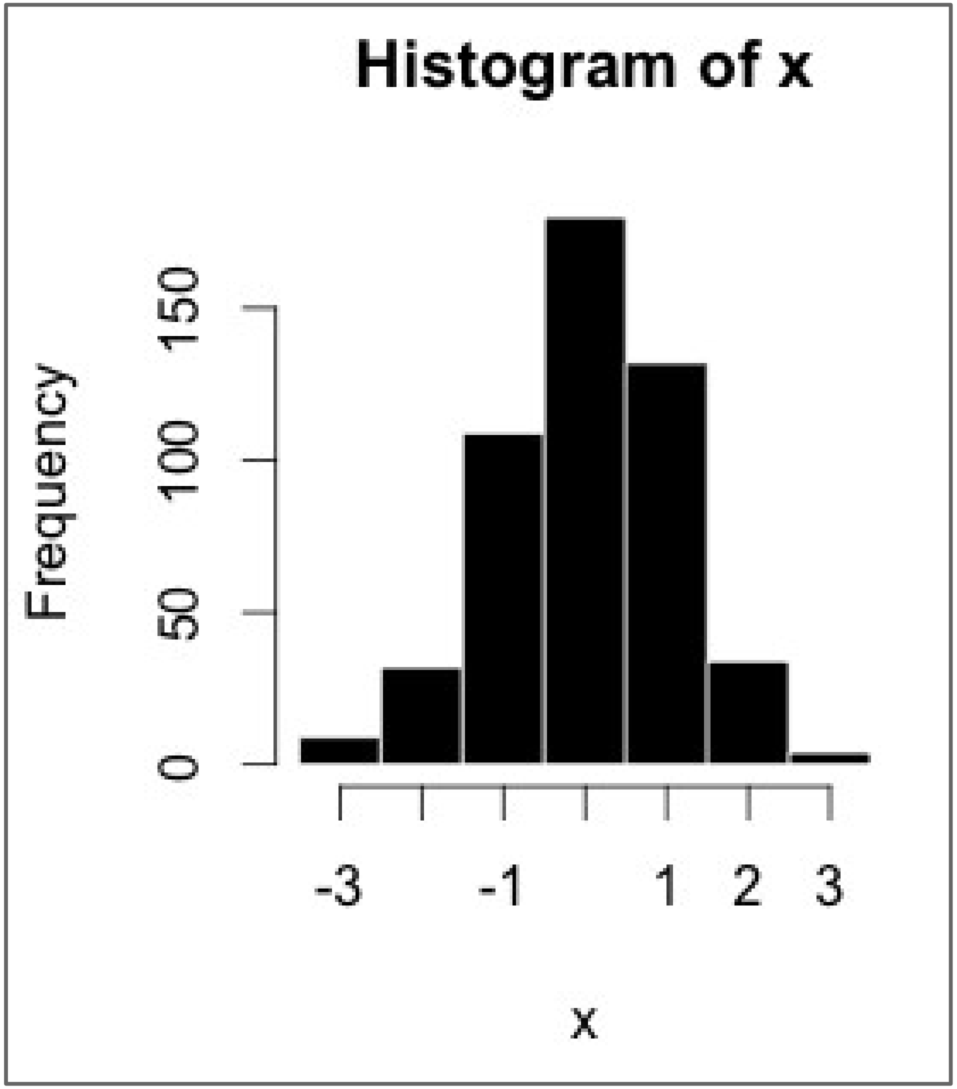
note that we have imported "norm" from scipy.stats in Step 1

```
[ ] (Jmu, Jsigma) = norm.fit(Joyce_ages)

    ## you may print these quantities by uncommenting the line below
    #print("Jmu: ", Jmu, "  Jsigma: ", Jsigma )
```

- what this piece of code does is find the “best” values of mu and sigma for a fit of ***f(x)*** to the distribution formed by Joyce_ages

- it names these fit parameters *Jmu, Jsigma* and we will use them later to make a function

Step 5: Histogram and bins

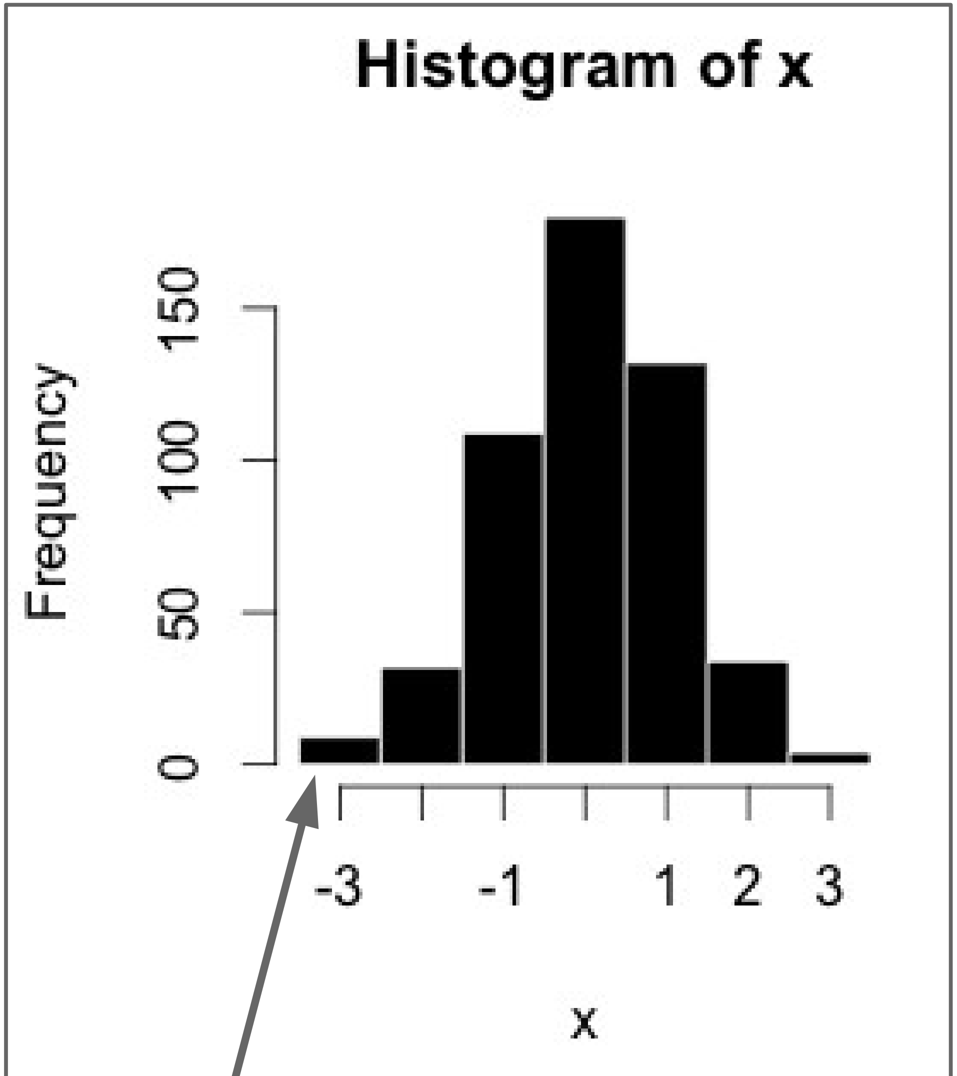


- on the x-axis, we have observations of some quantity
in our case, *x* = stellar ages

- the y-axis counts the number of occurrences of *x*

So, if out of 91 stars (*x*), we have 12 stars with an age of 10 Gyr, the *y* value for *x* = 10 will be 12.

Step 5: Histogram and bins



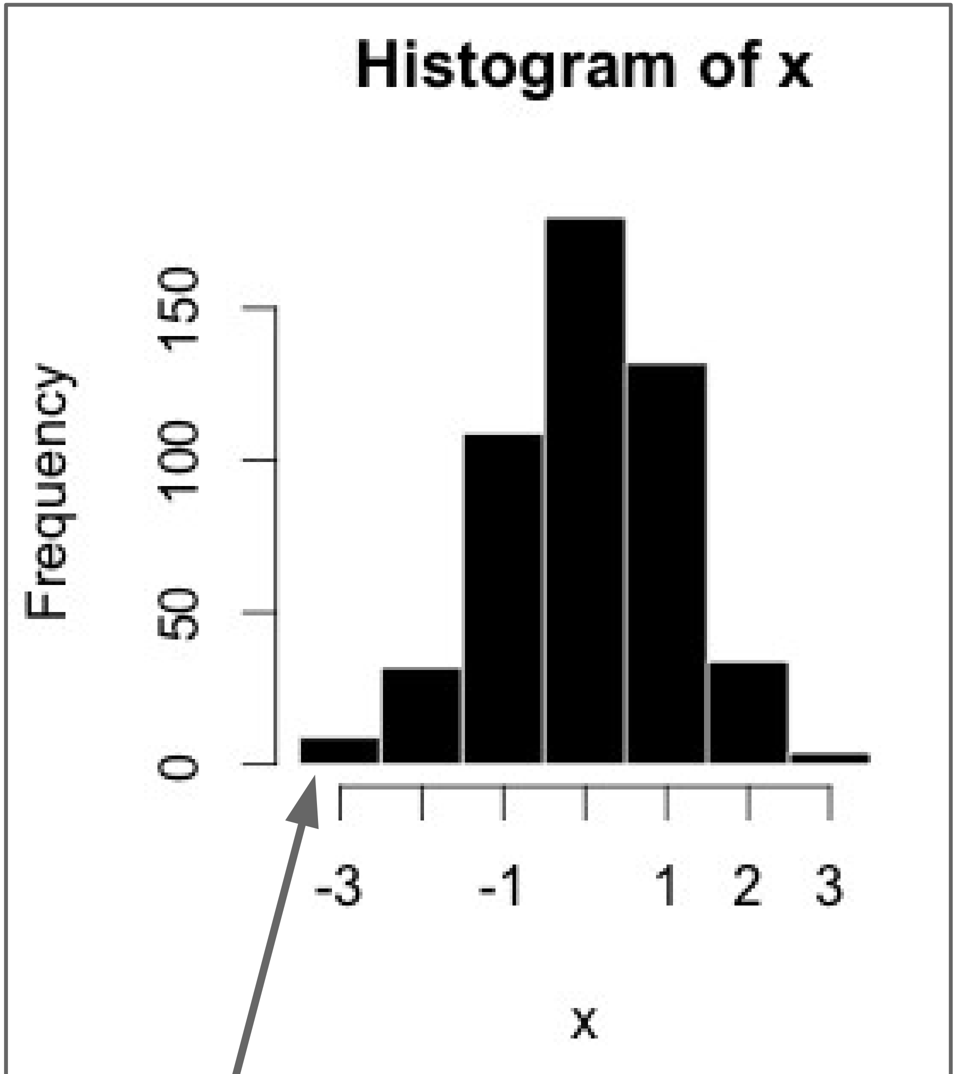
- on the x-axis, we have observations of some quantity
in our case, *x* = stellar ages

- the y-axis counts the number of occurrences of *x*

So, if out of 91 stars (*x*), we have 12 stars with an age of 10 Gyr, the *y* value for *x* = 10 will be 12.

The number of bars corresponds to the number of “bins,” in this case, 7.
The choice of bin number (or bin size) can have a noticeable effect on the shape of a distribution—this is notorious weakness of histograms.

Step 5: Histogram and bins



- on the x-axis, we have observations of some quantity
in our case, *x* = stellar ages

- the y-axis counts the number of occurrences of *x*

So, if out of 91 stars (*x*), we have 12 stars with an age of 10 Gyr, the *y* value for *x* = 10 will be 12.

Step 5: create a histogram of the ages

```
[ ] histogram = np.histogram(Joyce_ages)
```

Now, grab the bins from the histogram we have created

```
[ ] bins = histogram[1]
    ## you may print the bins by uncommenting the line below
    #print("bins: ", bins)
```

The number of bars corresponds to the number of “bins,” in this case, 7. The choice of bin number (or bin size) can have a noticeable effect on the shape of a distribution—this is notorious weakness of histograms.

Step 6: Creating a Gaussian model for our data

- we now have bins, mu, and sigma defined for our data (mu = Jmu, sigma = Jsigma)

- we can think of the bins as serving the role of *x* in the Gaussian function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Step 6: Creating a Gaussian model for our data

- we now have bins, mu, and sigma defined for our data (mu = Jmu, sigma = Jsigma)

- we use the scipy.stats function **norm.pdf()** to create our model with this information

Step 6: Create a curve defined by μ, σ

"pdf" in norm.pdf stands for "probability density function," and it is normalized such that its area is 1 by default

```
[ ]  normalized_gaussian_pdf = norm.pdf(bins, Jmu, Jsigma)
```

Step 6: Creating a Gaussian model for our data

- we now have bins, mu, and sigma defined for our data (mu = Jmu, sigma = Jsigma)

- we use the scipy.stats function **norm.pdf()** to create our model with this information

Step 6: Create a curve defined by μ, σ

"pdf" in norm.pdf stands for "probability density function," and it is normalized such that its area is 1 by default

```
[ ] normalized_gaussian_pdf = norm.pdf(bins, Jmu, Jsigma)
```

However (!!) this will generate a Gaussian whose integral is equal to 1, by definition: AKA “normalized”

Step 6: Creating a Gaussian model for our data

Step 6: Create a curve defined by μ, σ

"pdf" in norm.pdf stands for "probability density function," and it is normalized such that its area is 1 by default

```
[ ] normalized_gaussian_pdf = norm.pdf(bins, Jmu, Jsigma)
```

Now, rescale the curve so that it fits the size of our data. There are 91 age measurements, so len(Joyce_ages) = 91. We multiply our normalized Gaussian by this value

```
[ ] gaussian_pdf= normalized_gaussian_pdf*len(Joyce_ages)
```

So, we rescale the normalized Gaussian pdf so that its integral (~ sum over discrete bins) is equal to our data size (91 stellar ages)

Step 7: Graphically compare data and model

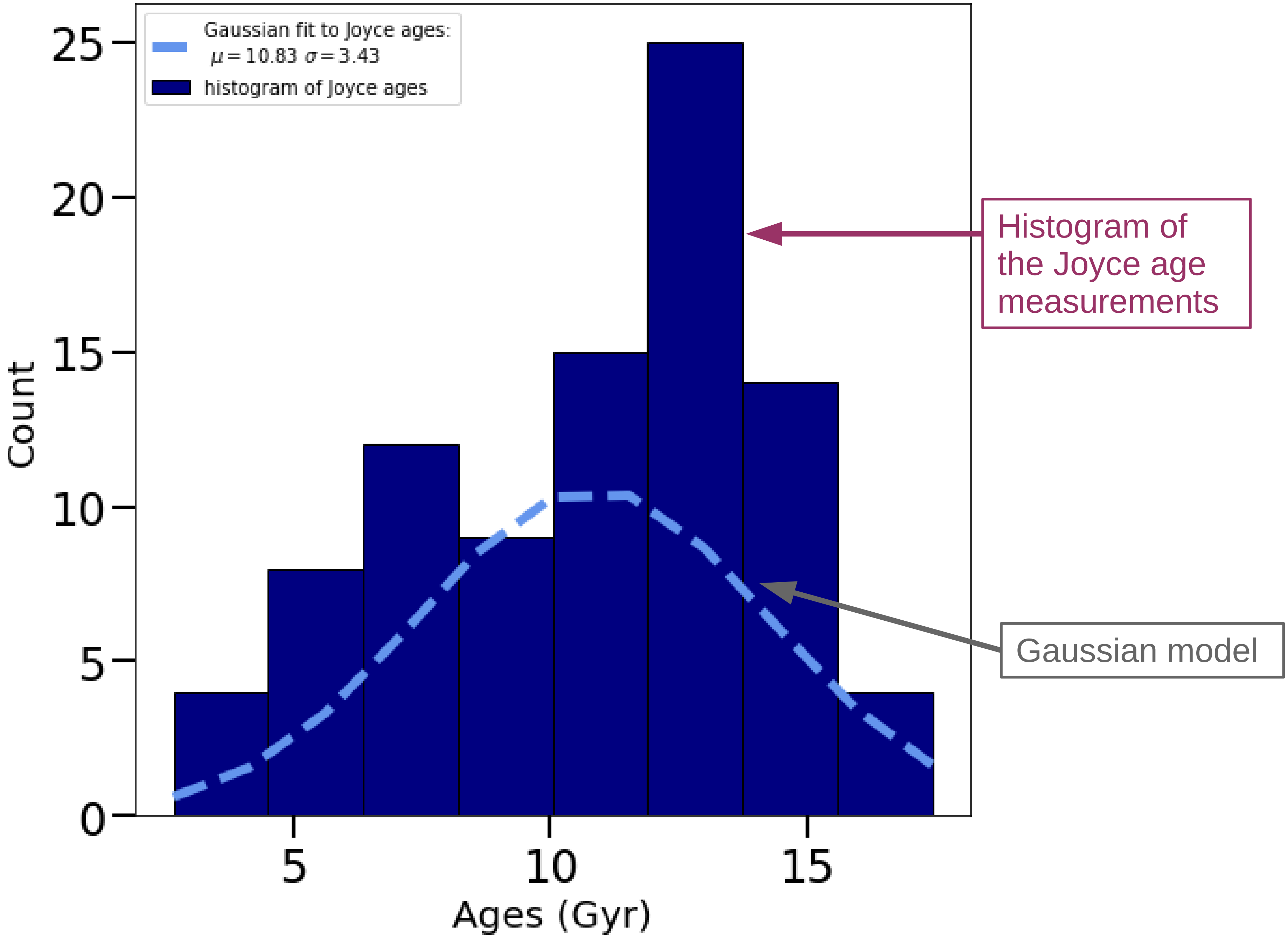
Step 7: Plot our histogram and the Gaussian curve we have fit to it

```
## initiate the figure
fig, ax = plt.subplots(figsize = (8,8))
set_fig(ax)


## this is the histogram
plt.hist(Joyce_ages,  bins="auto", color='navy', edgecolor='black', label='histogram of Joyce ages')

## this is the Gaussian curve
plt.plot(bins, gaussian_pdf,\
         '--', color='cornflowerblue', linewidth=5,\
         label='Gaussian fit to Joyce ages:\n $\mu=$'+ "%.2f"%Jmu + ' $\sigma=$'+ "%.2f"%Jsigma )


## these lines are plot bookkeeping
plt.xlabel('Ages (Gyr)', fontsize=20)
plt.ylabel('Count', fontsize=20)
plt.legend(loc=2)
plt.show()
plt.close()
```

Histogram of the Joyce age measurements

Gaussian model vs bins

Step 7: Graphically compare data and model



Pause – how good is this fit?

Pause – how good is this fit?

Not great!

Kernel Density Estimation

Briefly: Math of KDE

Step 8: KDE for Joyce ages

Step 8: Try a Kernel Density Estimate (KDE) instead

Create the kde model for the stellar ages

```
kde_model = stats.gaussian_kde(Joyce_ages)
```

to make the model smoother, we can increase the resolution of the x-axis

the line below subdivides the age range into 1000 equally spaced values. The age range is the minimum age measurement, min(Joyce_ages), to the maximum age measurement, max(Joyce_ages). These correspond to about 2 Gyr (billion years) and 17 Gyr, respectively

```
[ ] age_x_values = np.linspace(min(Joyce_ages), max(Joyce_ages), 1000)
```

the following line evaluates the kde_model function we made at the beginning of Step 8 over the smoother array of x values defined above.

```
[ ] kde = kde_model(age_x_values)
```

once again, the model is normalized to 1, so we must rescale it by the number of age measurements

```
[ ] ## scale the kde by the number of stellar ages in our sample (91)
    scaled_kde = kde*len(Joyce_ages)
```

Step 9: Compare KDE graphically

Python implementation