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

@MeriditḥJoyceGR
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:

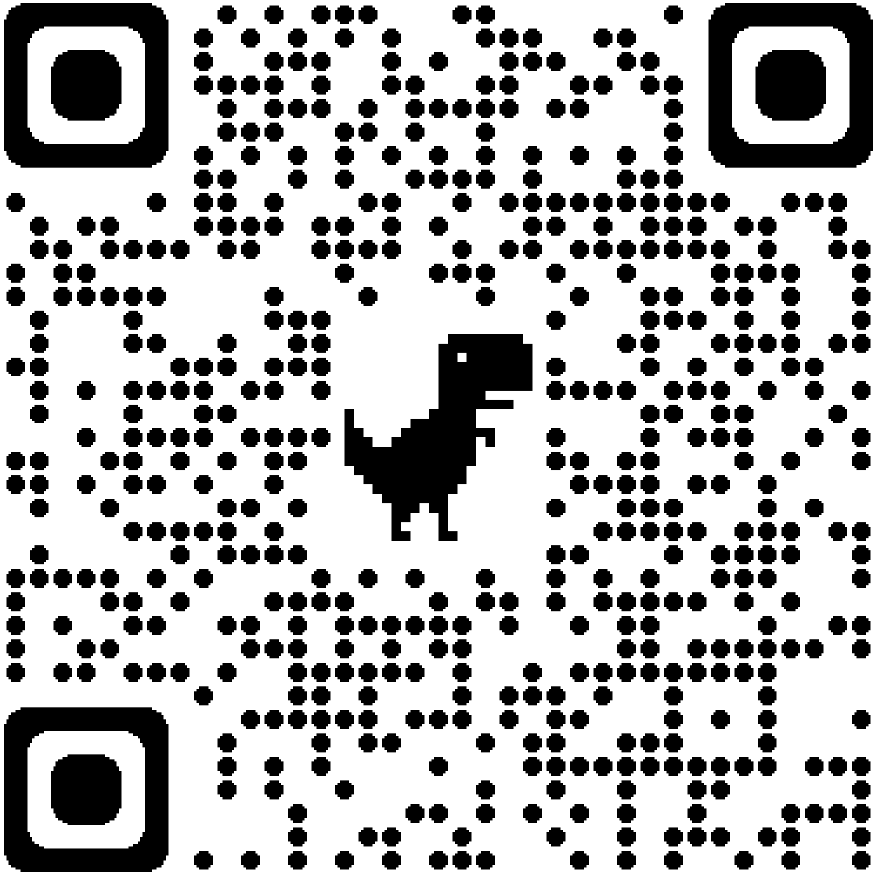
Given an observed distribution of stellar ages in the center of our Galaxy, we would like to predict the distribution of stellar ages in other Galaxies

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 -- 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

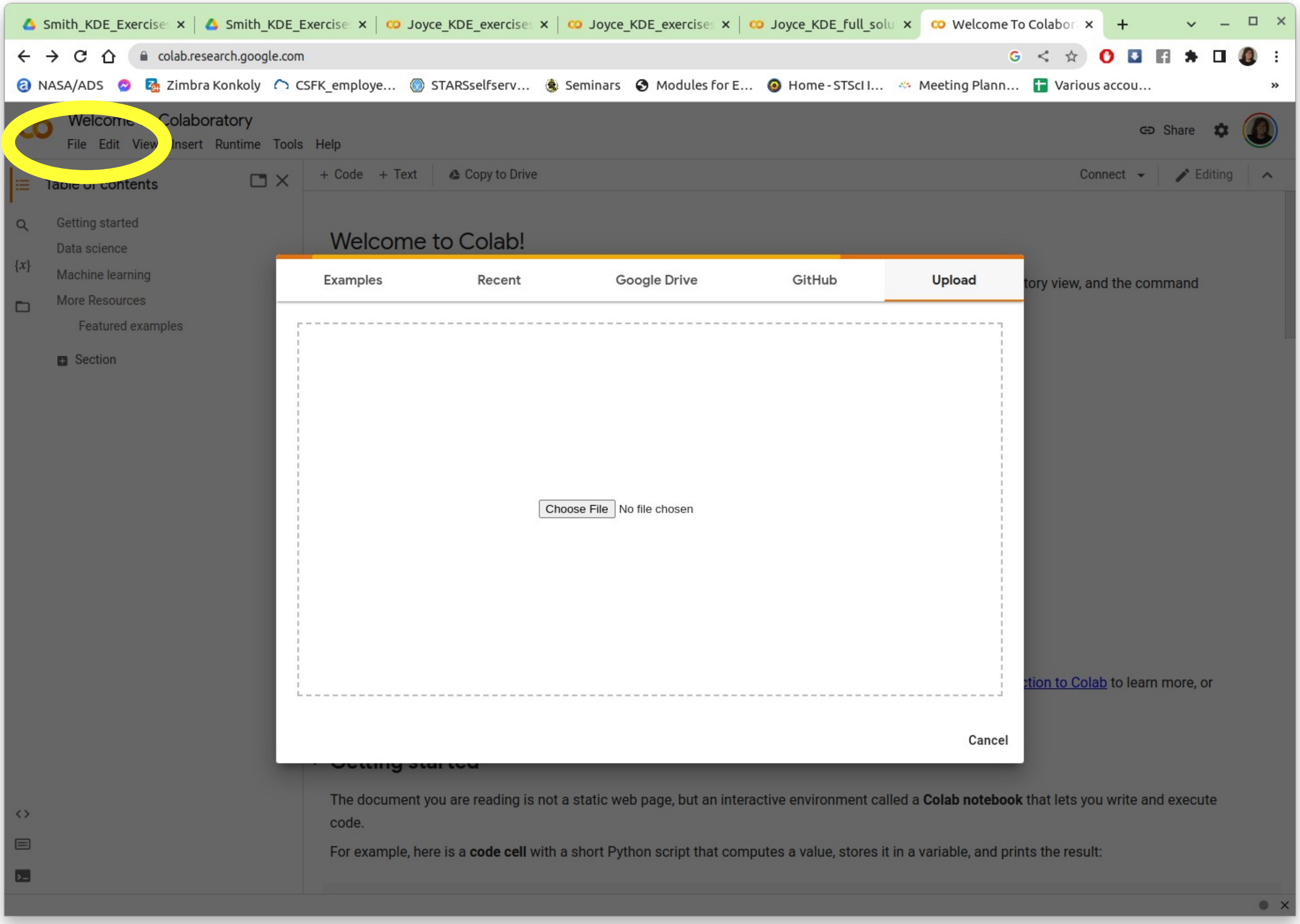
a) **KDE_exercises.ipynb**
b) stellar_ages.dat

(3) 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 files **a** and **b**

Step 0: Bookkeeping

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



Welcome to Colab!

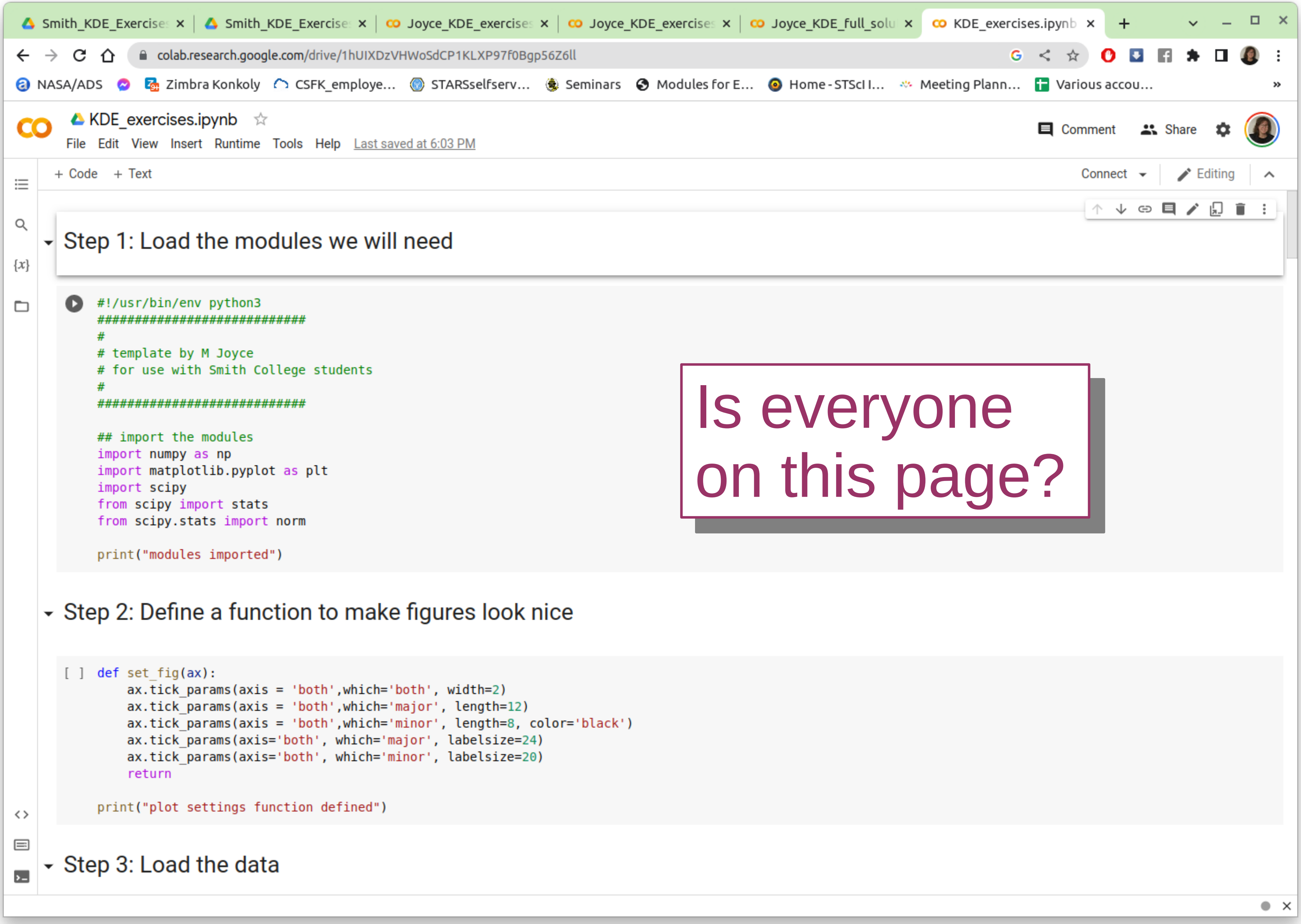
tory view, and the command

ction to Colab to learn more, or

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:



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

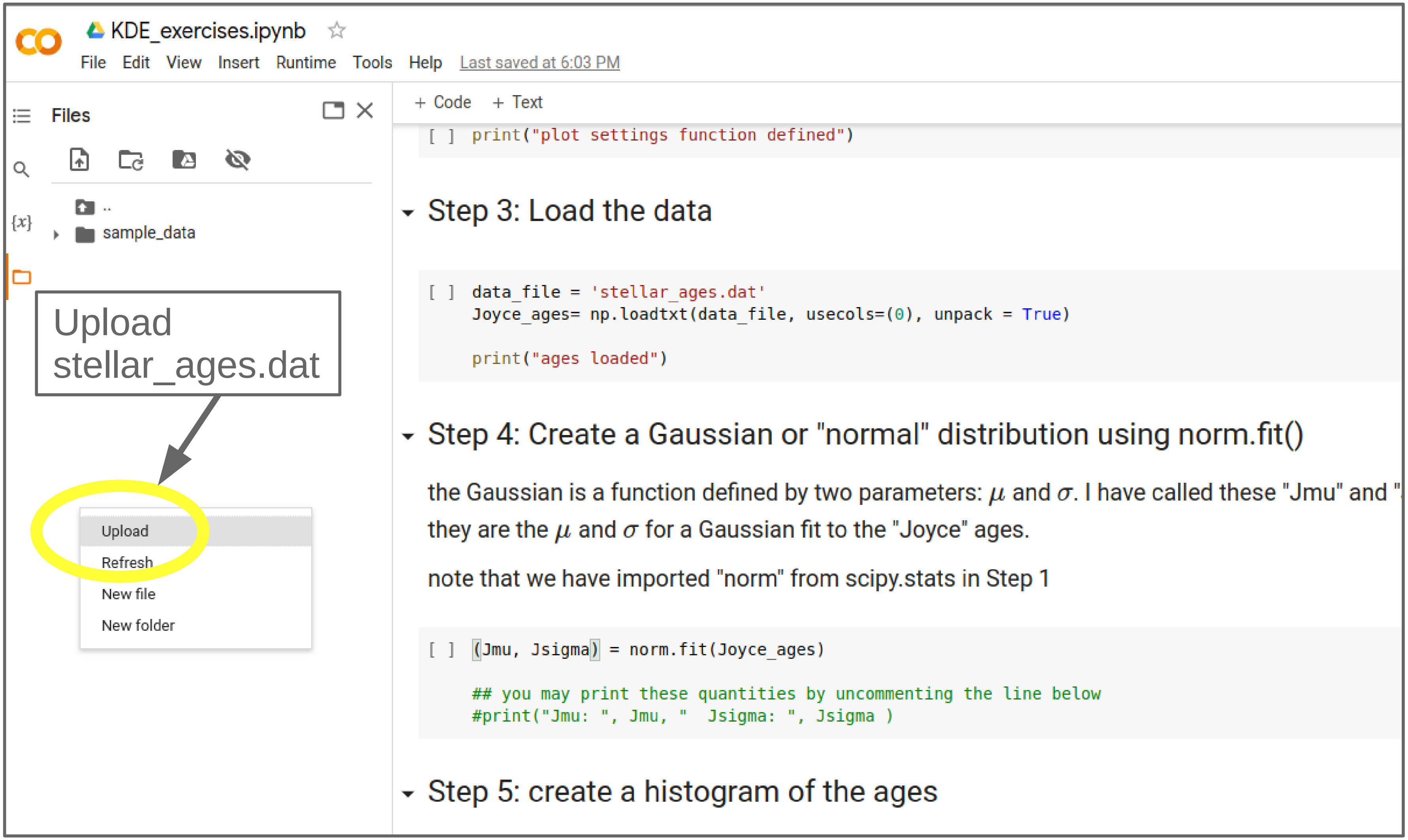
Step 1: Load the modules

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

You must upload stellar_ages.dat in colab, *in* the notebook you have loaded (KDE_exercises.ipynb)



Upload stellar_ages.dat

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

Step 3: Load the data

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

print("ages loaded")
```

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 "

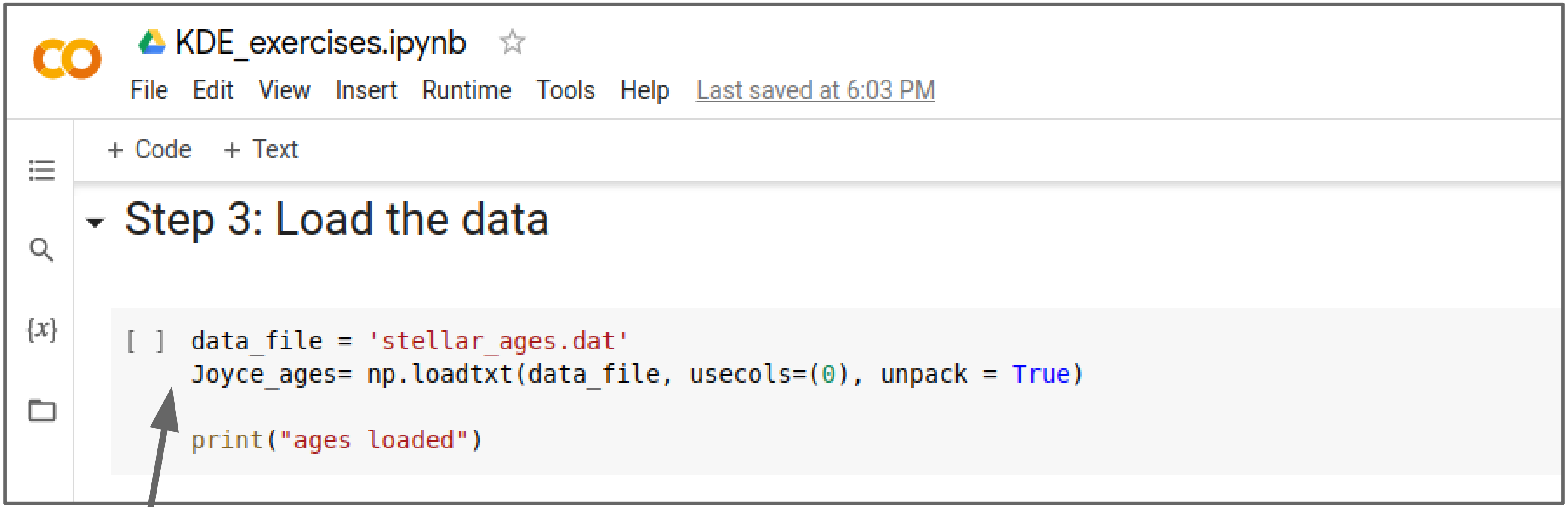
they are the μ and σ for a Gaussian fit to the "Joyce" ages.

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 )
```

Step 5: create a histogram of the 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

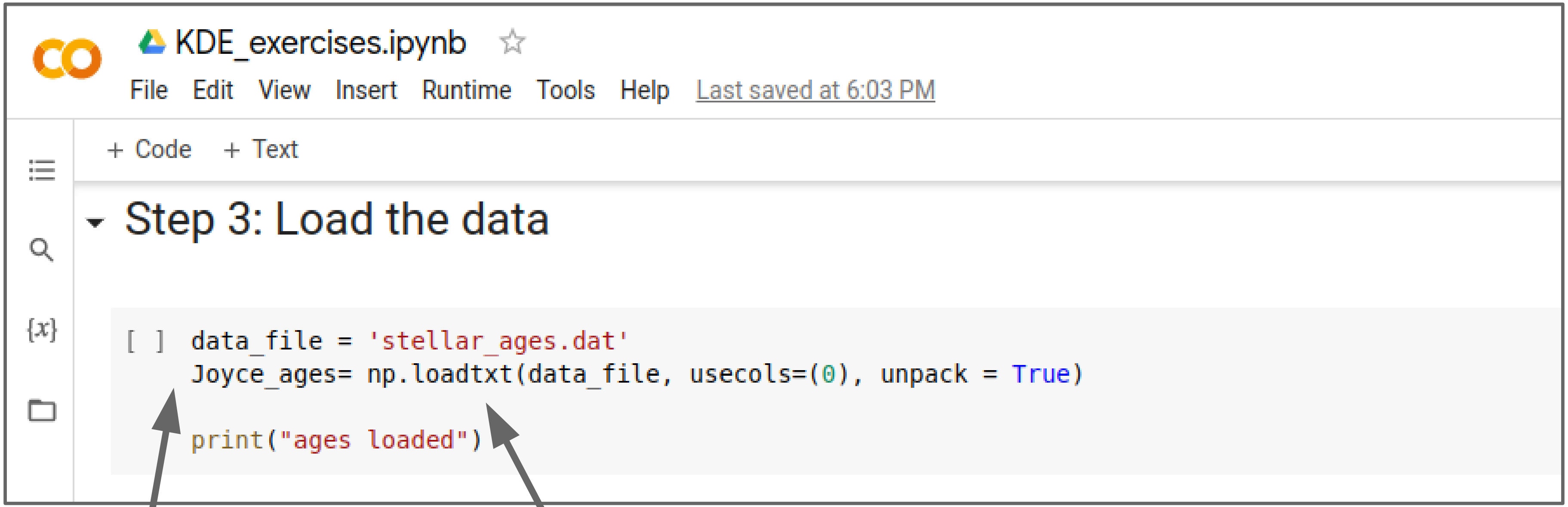
Step 3: Load the data

```
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”

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

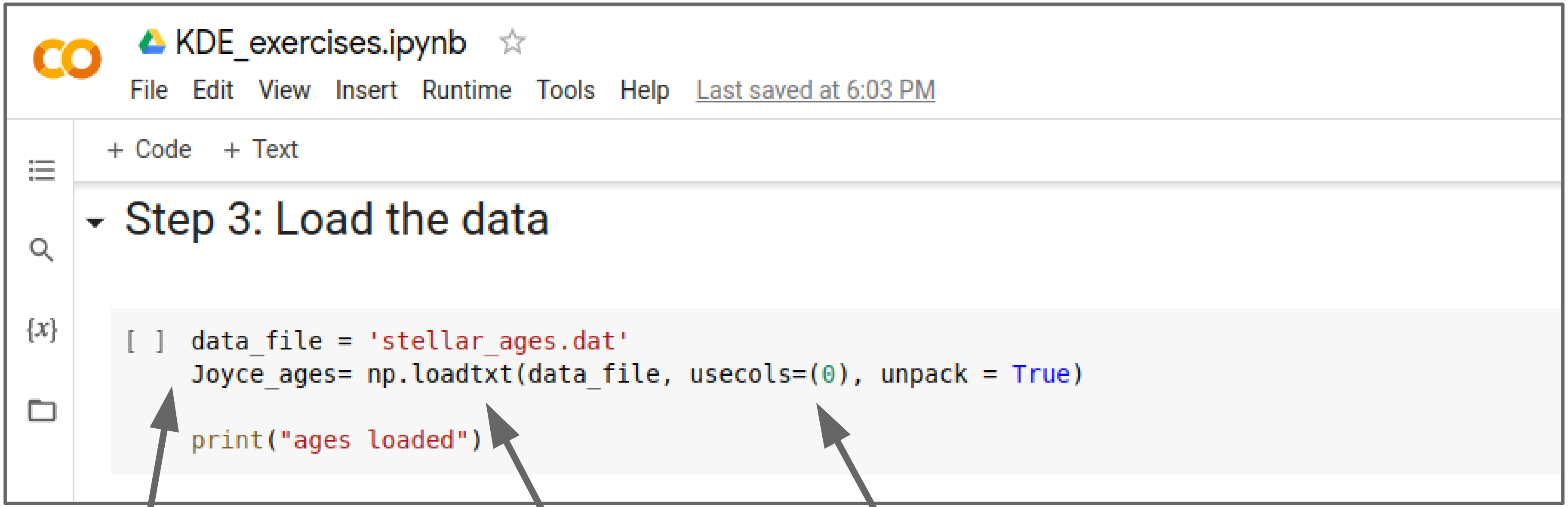
```
[ ] data_file = 'stellar_ages.dat'
    Joyce_ages= np.loadtxt(data_file, usecols=(0), unpack = True)

    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



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")
```

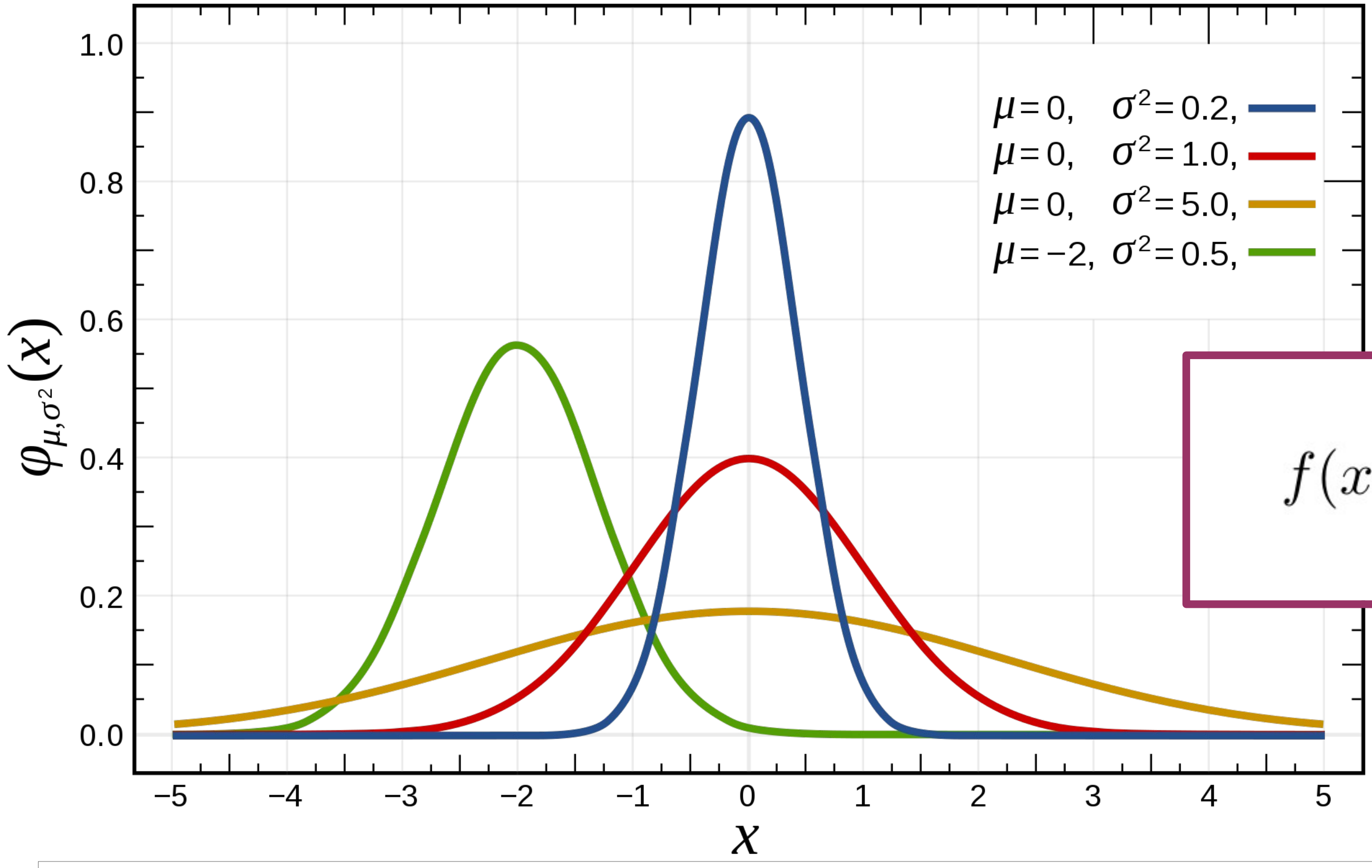
Python indexes from zero, so the first column of stellar_ages.dat is “column 0”

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



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

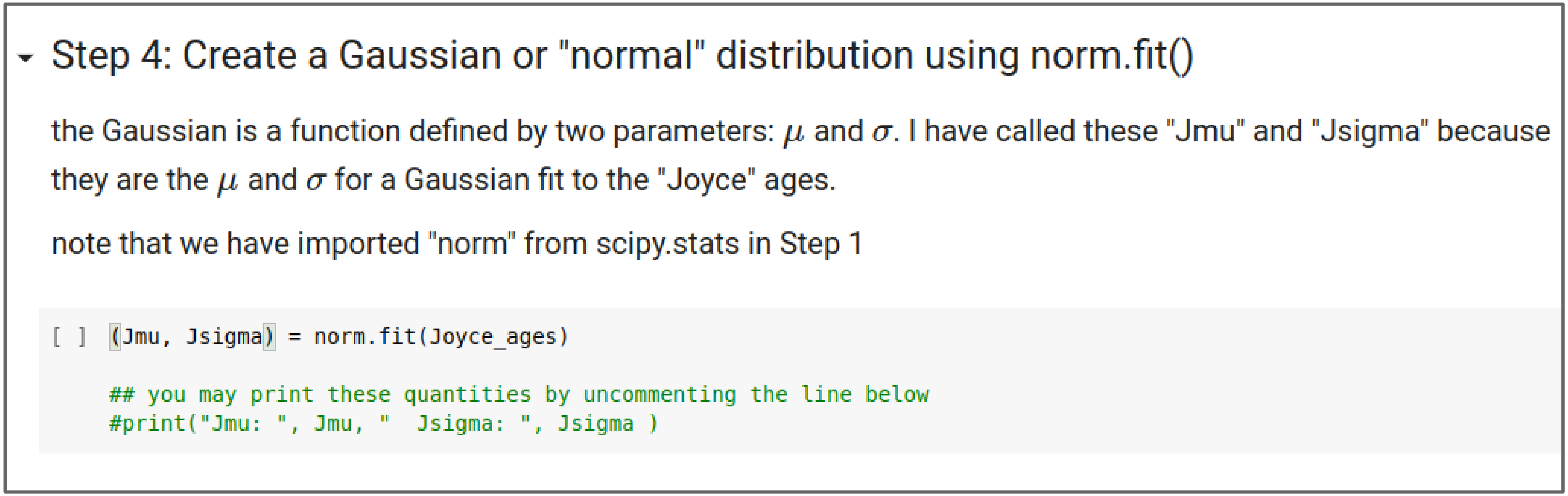
- the shape of the curve is captured by **mu**, the “expected value,” or mean, and **sigma**, which is related to the width and represents one standard deviation (sigma^2 is the “variance,” as shown in the legend)

- the type of distribution everyone (in astronomy) assumes their data 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.

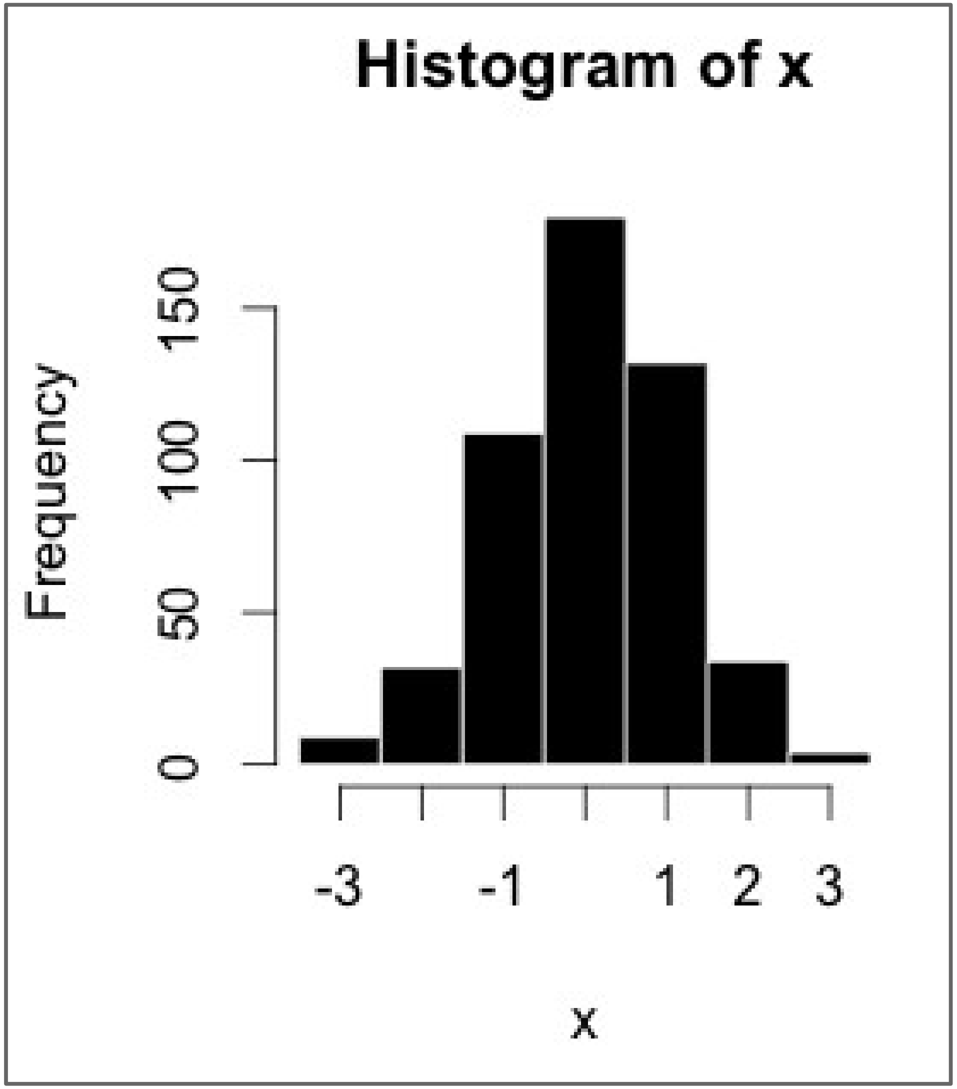
note that we have imported "norm" from scipy.stats in Step 1

```
[ ]  (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

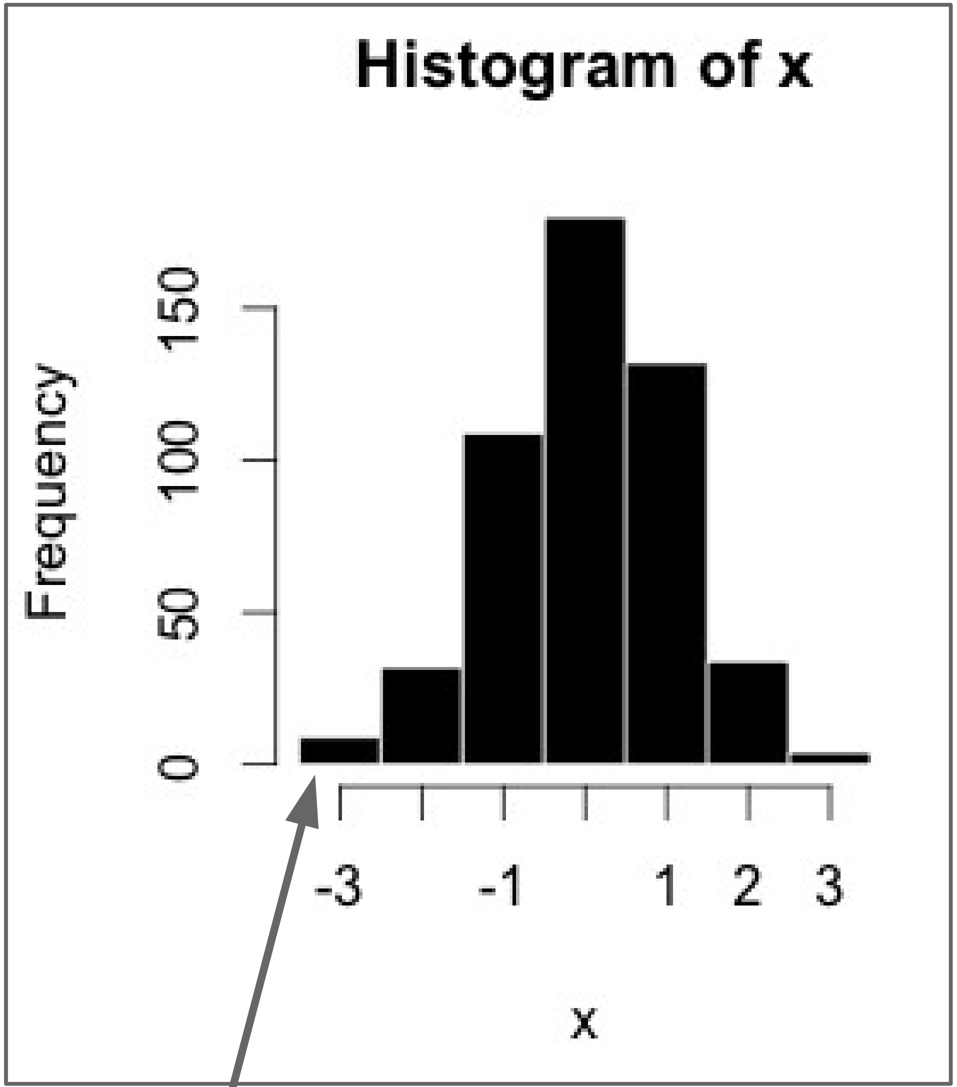


- on the x-axis, we have observations of some quantity
in our case, *x* = stellar ages

- 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



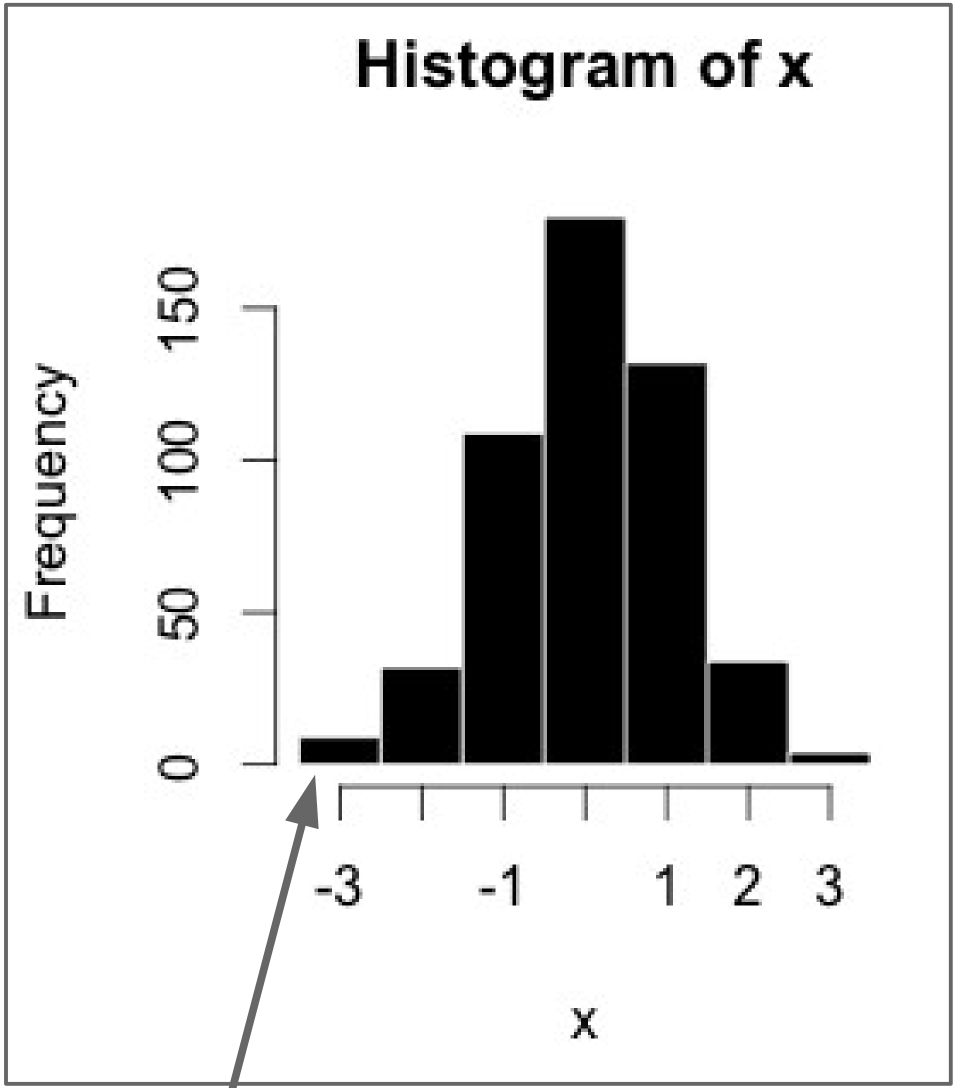
- on the x-axis, we have observations of some quantity
in our case, *x* = stellar ages

- 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.

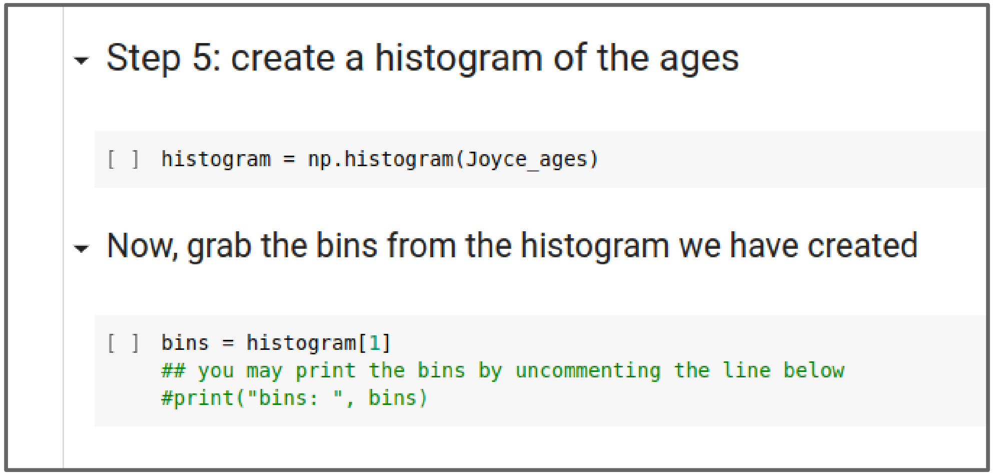
Step 5: Histogram and bins



- 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)

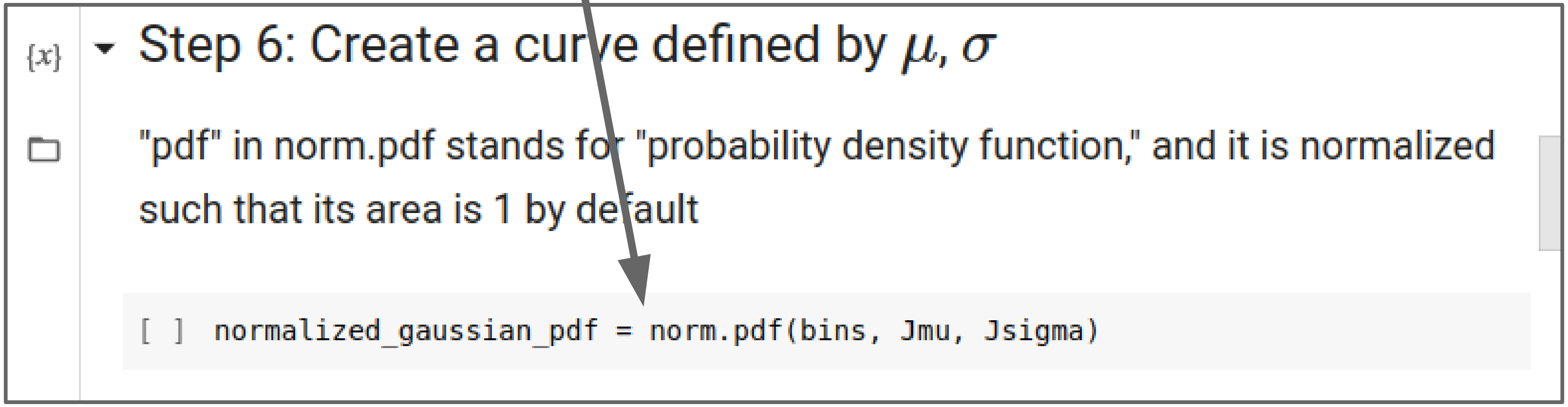
- we can think of the bins as serving the role of *x* in the Gaussian function

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

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

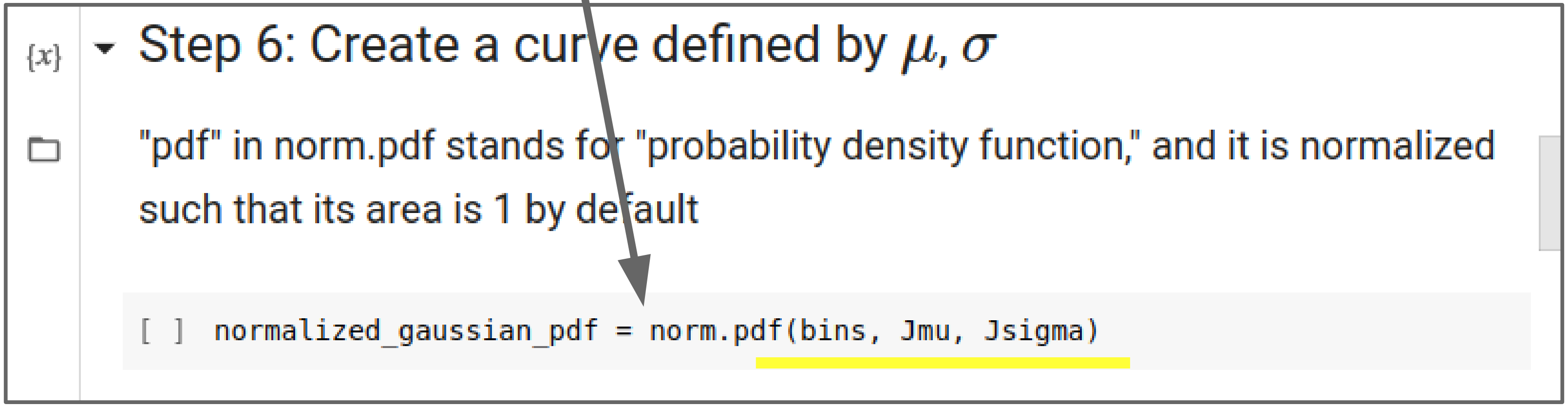
"pdf" in norm.pdf stands for "probability density function," and it is normalized such that its area is 1 by default

```
[ ]  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



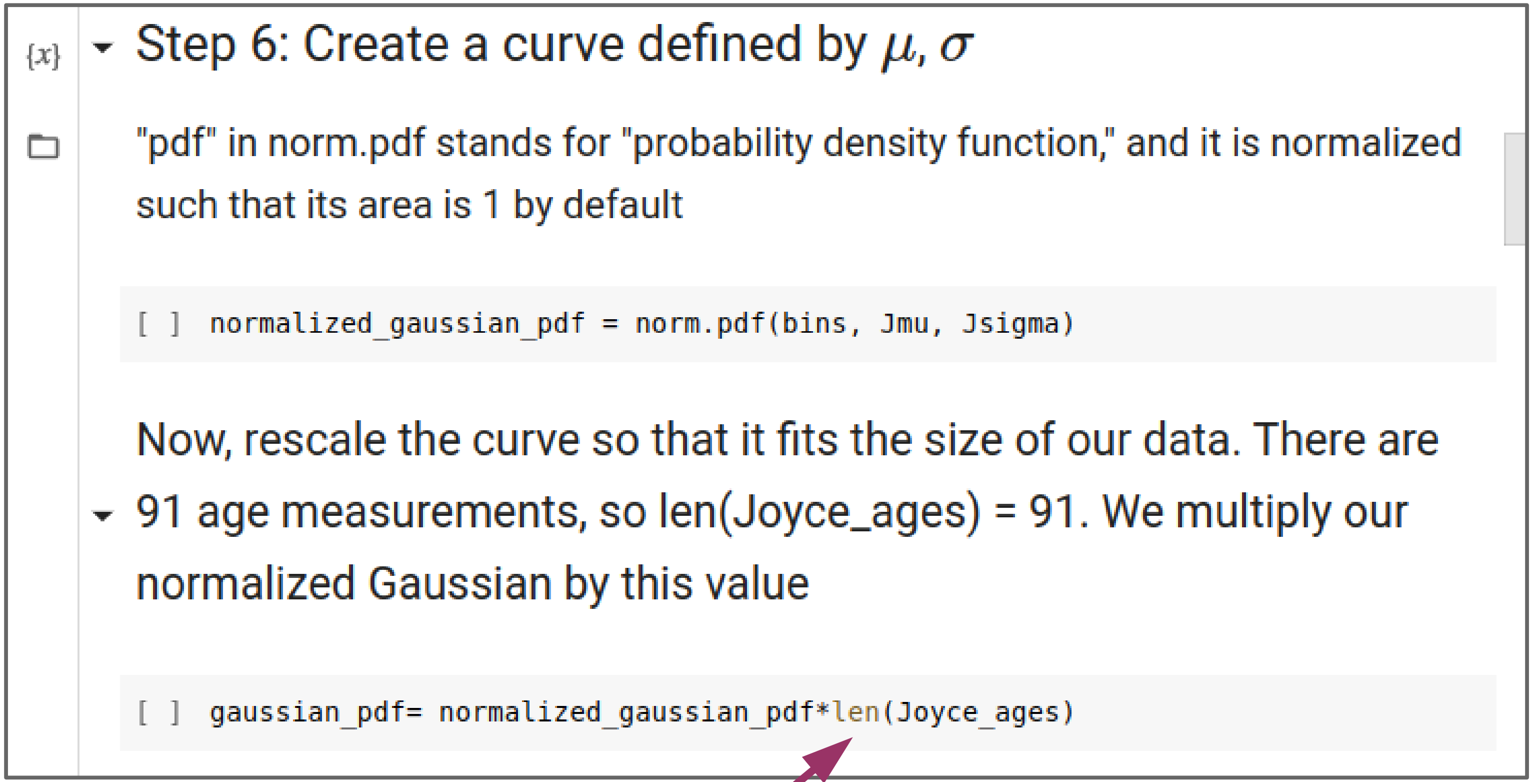
Step 6: Create a curve defined by μ, σ

"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

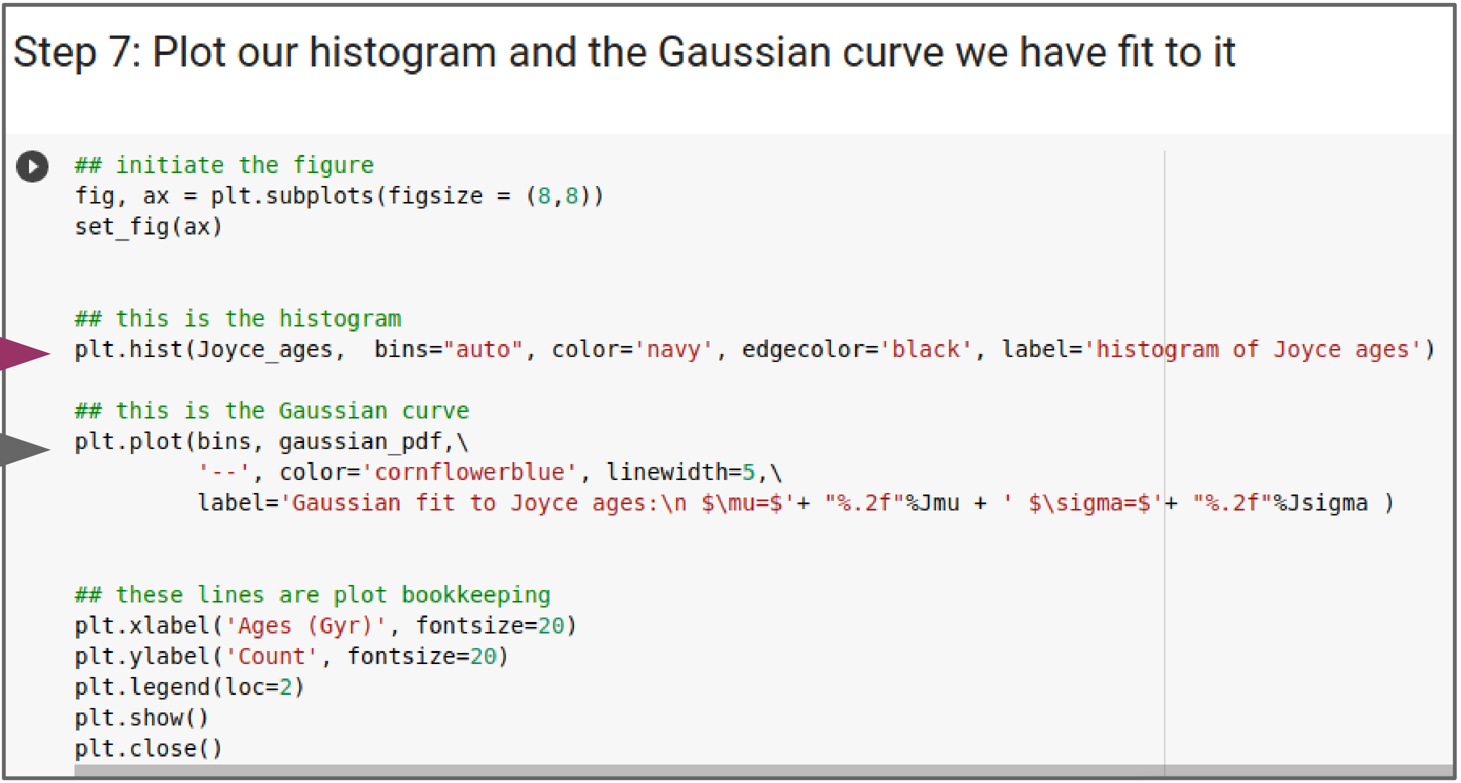
```
[ ]  gaussian_pdf= normalized_gaussian_pdf*len(Joyce_ages)
```

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

Histogram of the Joyce age measurements

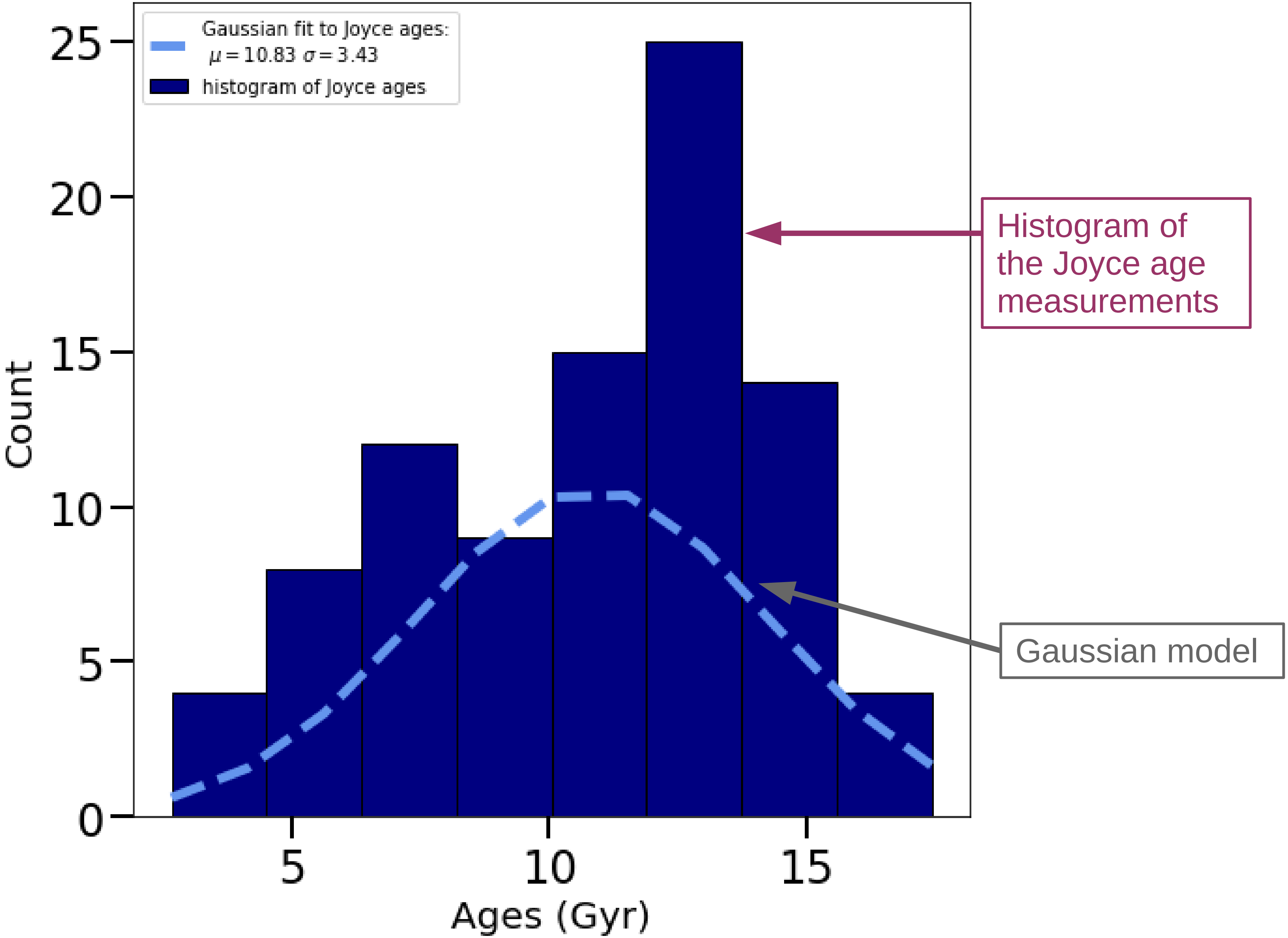
Gaussian model vs bins



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()
```

Step 7: Graphically compare data and model



Histogram of the Joyce age measurements

Gaussian model

Pause – how good is this fit?

Pause – how good is this fit?

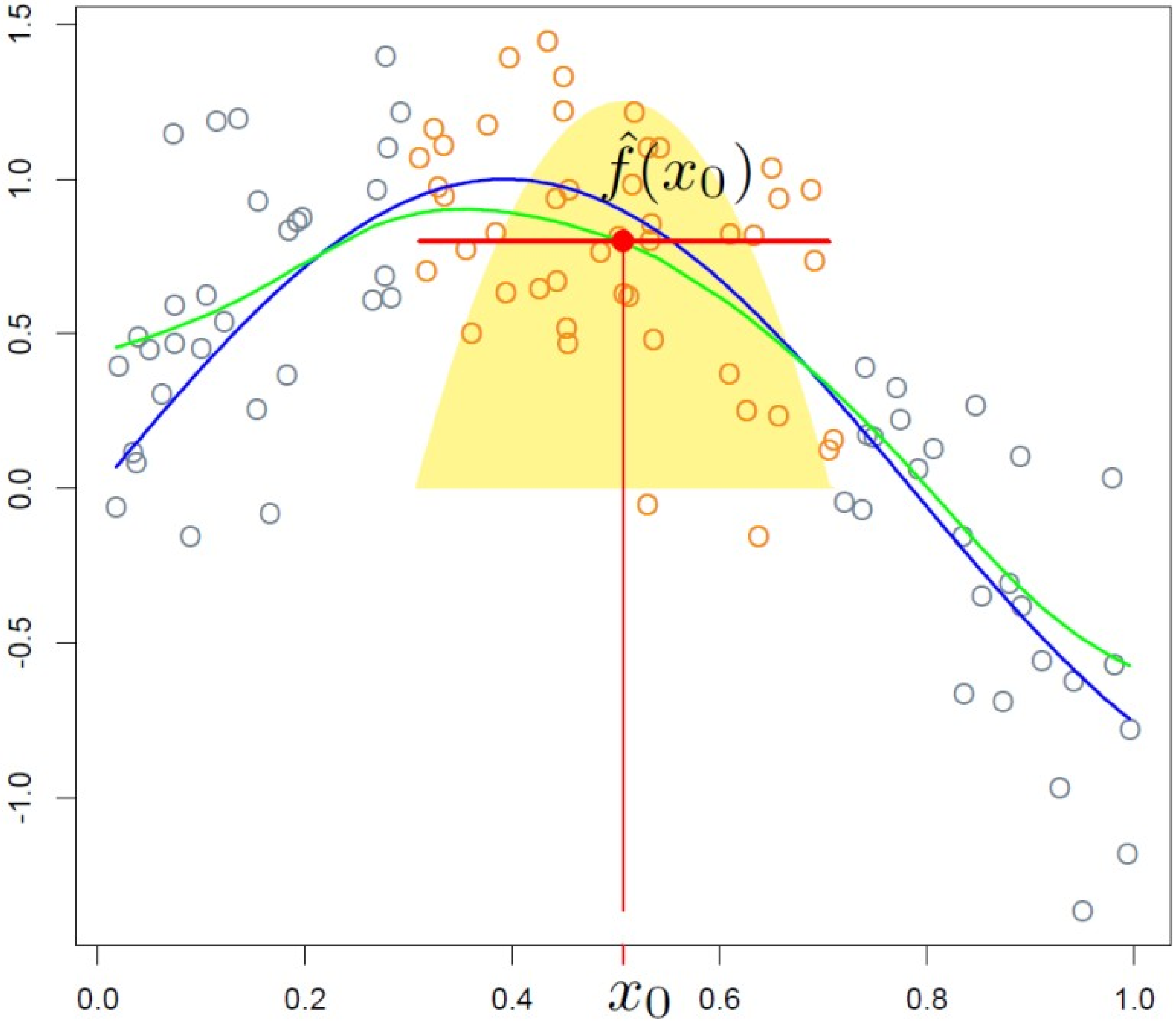
Not great!

Kernel Density Estimation

Kernel Density Estimation

- *kernel smoothing* is a statistical technique used to infer a function based on the local clustering (or density) of observed data

Kernel Density Estimation



We may think of “sliding” the yellow region—the kernel—across the data to generate local weighted averages of the data that combine to form the smoothed curve

Kernel Density Estimation

- *kernel smoothing* is a statistical technique used to infer a function based on the local clustering (or density) of observed data

- the scale over which the data are “grouped” and averaged determines the “smoothness” of the kernel. This scale is set by the *bandwidth***,** sometimes denoted with *h*

Kernel Density Estimation

- *kernel smoothing* is a statistical technique used to infer a function based on the local clustering (or density) of observed data

- the scale over which the data are “grouped” and averaged determines the “smoothness” of the kernel. This scale is set by the *bandwidth*, sometimes denoted with *h*

- Kernel methods are *non-parametric* estimators, meaning they require no constraints from theory. Neural networks are another type of non-parametric technique

Kernel Density Estimation

- *kernel smoothing* is a statistical technique used to infer a function based on the local clustering (or density) of observed data

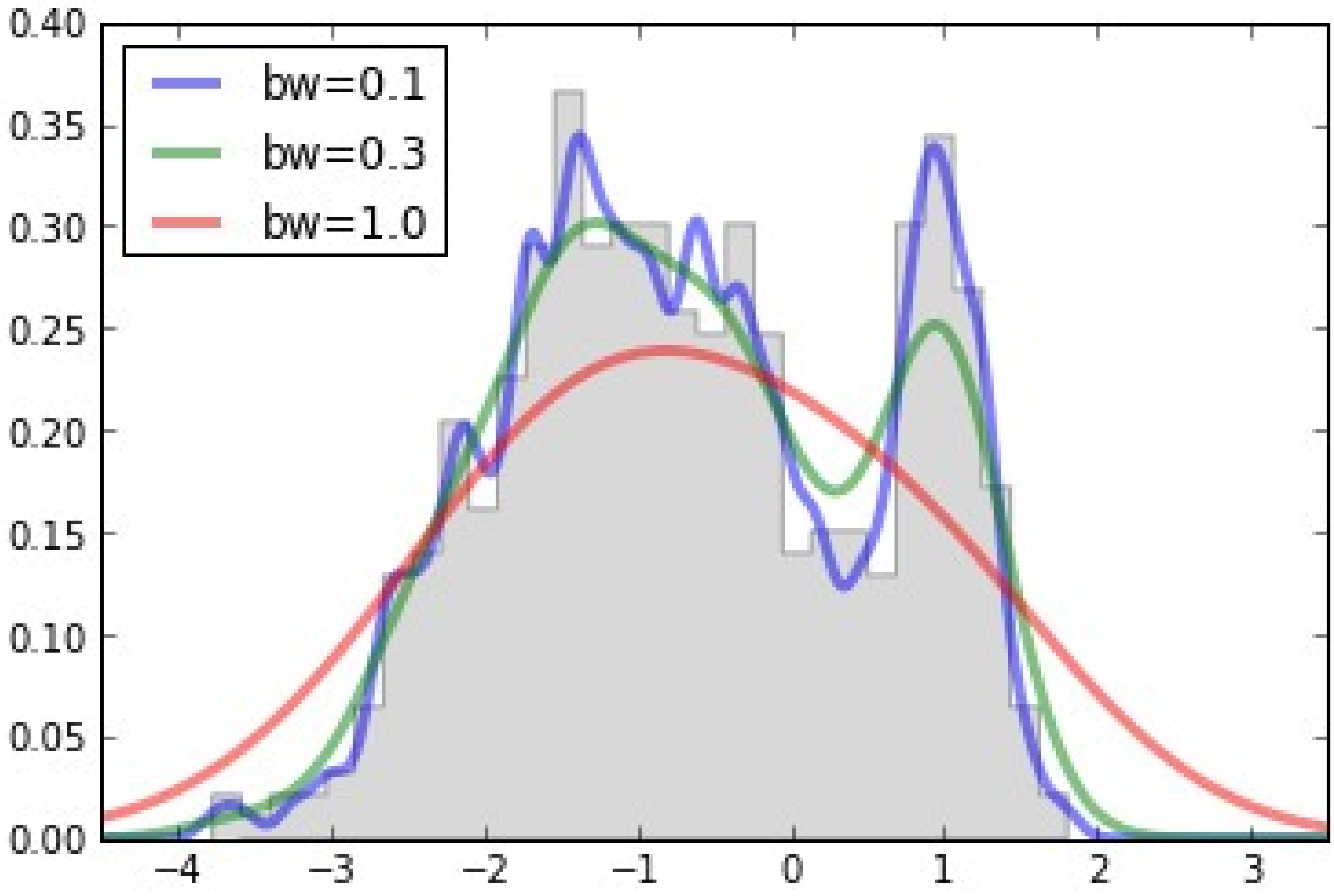
- the scale over which the data are “grouped” and averaged determines the “smoothness” of the kernel. This scale is set by the *bandwidth*, sometimes denoted with *h*

- Kernel methods are *non-parametric* estimators, meaning they require no constraints from theory. Neural networks are another type of non-parametric technique

- ***Kernel Density Estimation*** is the application of kernel smoothing to estimate the probability density function of a (continuous) random variable (e.g. stellar age) based on finite observed data (91 specific measurements of stellar age)

Kernel Density Estimation

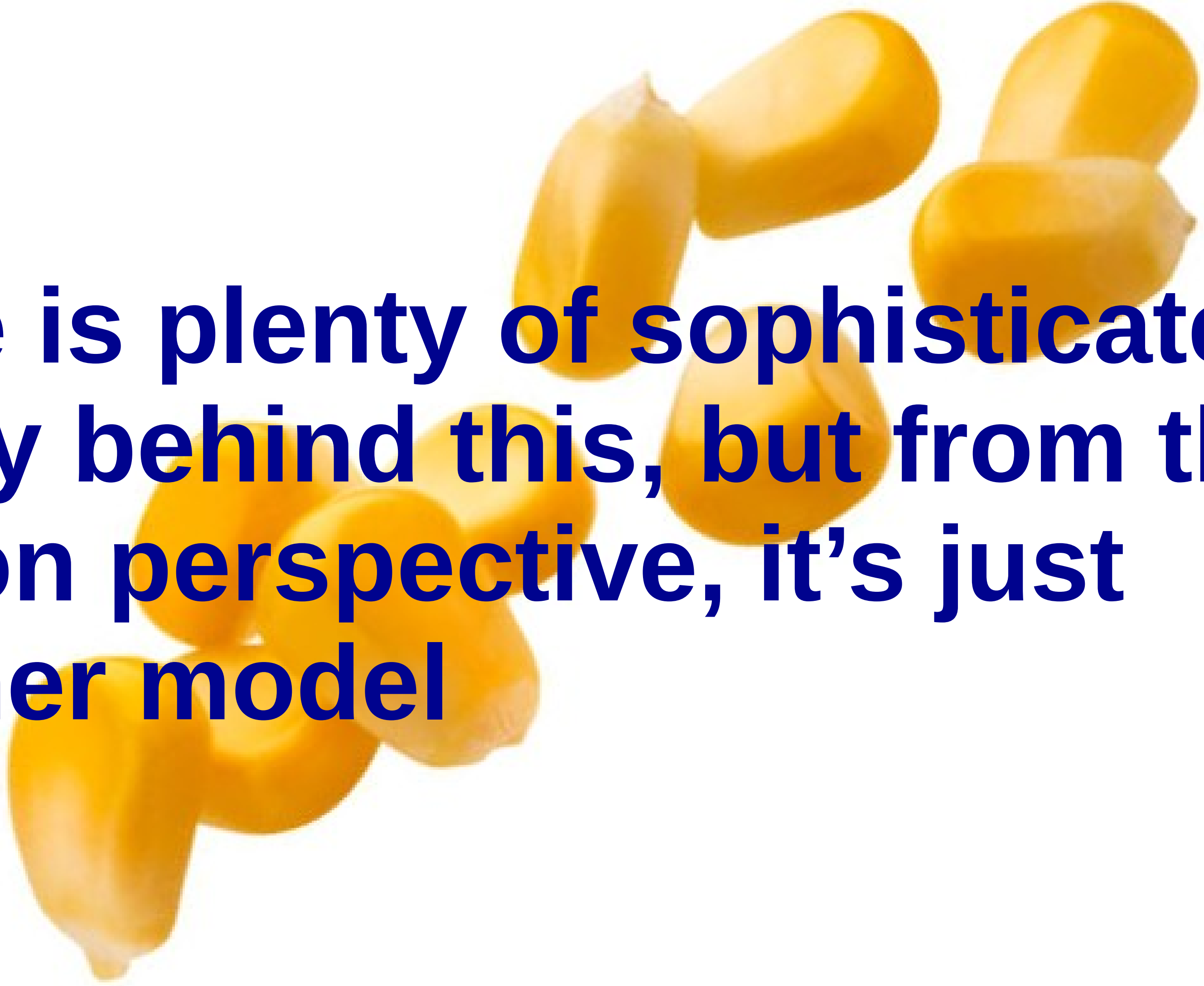
KDEs of some observed data (grey histogram) with three different bandwidths (bw)



Note that using a bandwidth of 1.0 results in a function that fails to capture some features of the observed distribution (**underfit**), whereas use of h = 0.1 **overfits** the data

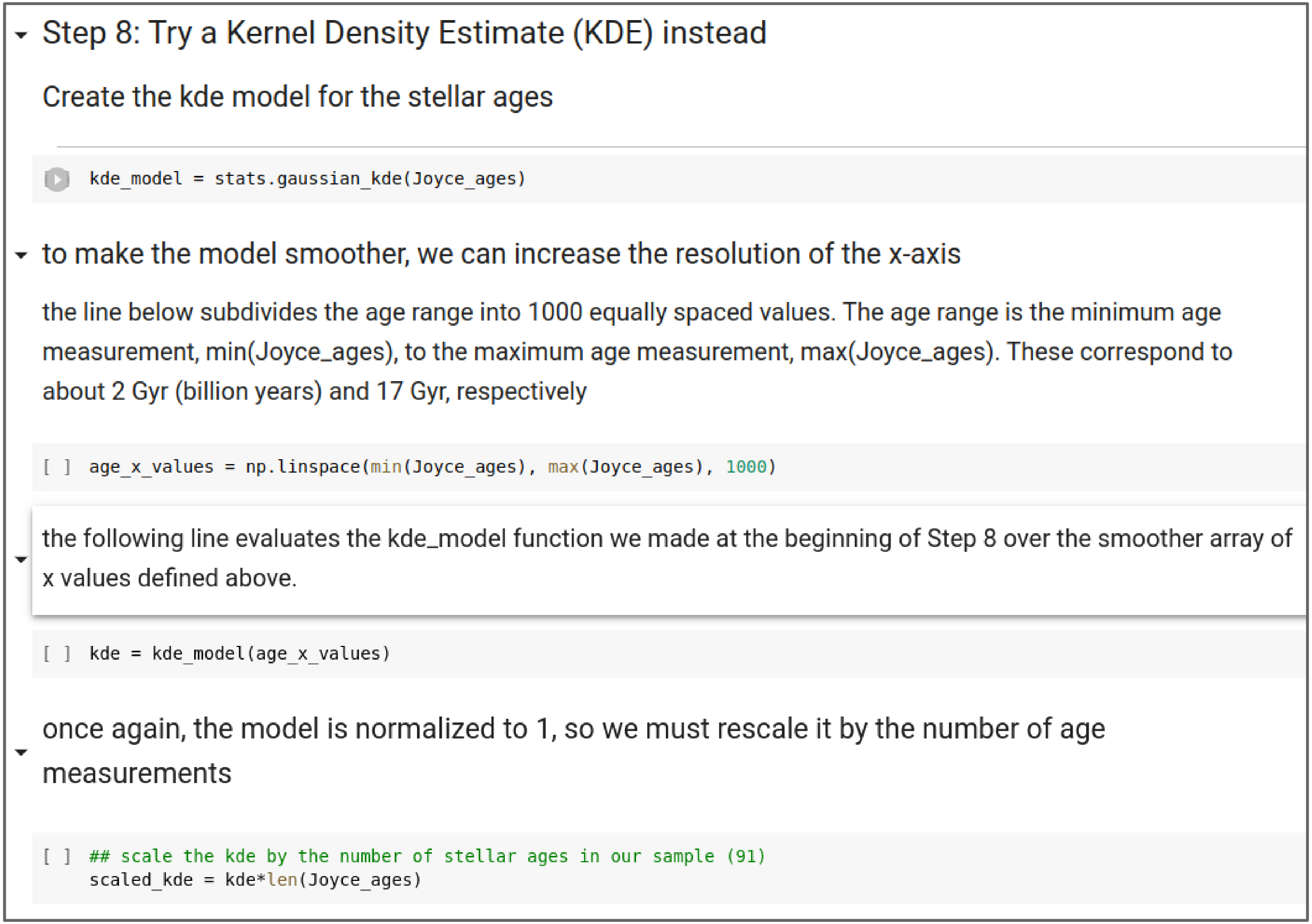
There is an optimal way to pick a bandwidth, but we won’t get into that here

Kernel Density Estimation



There is plenty of sophisticated theory behind this, but from the Python perspective, it’s just another model

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)
```

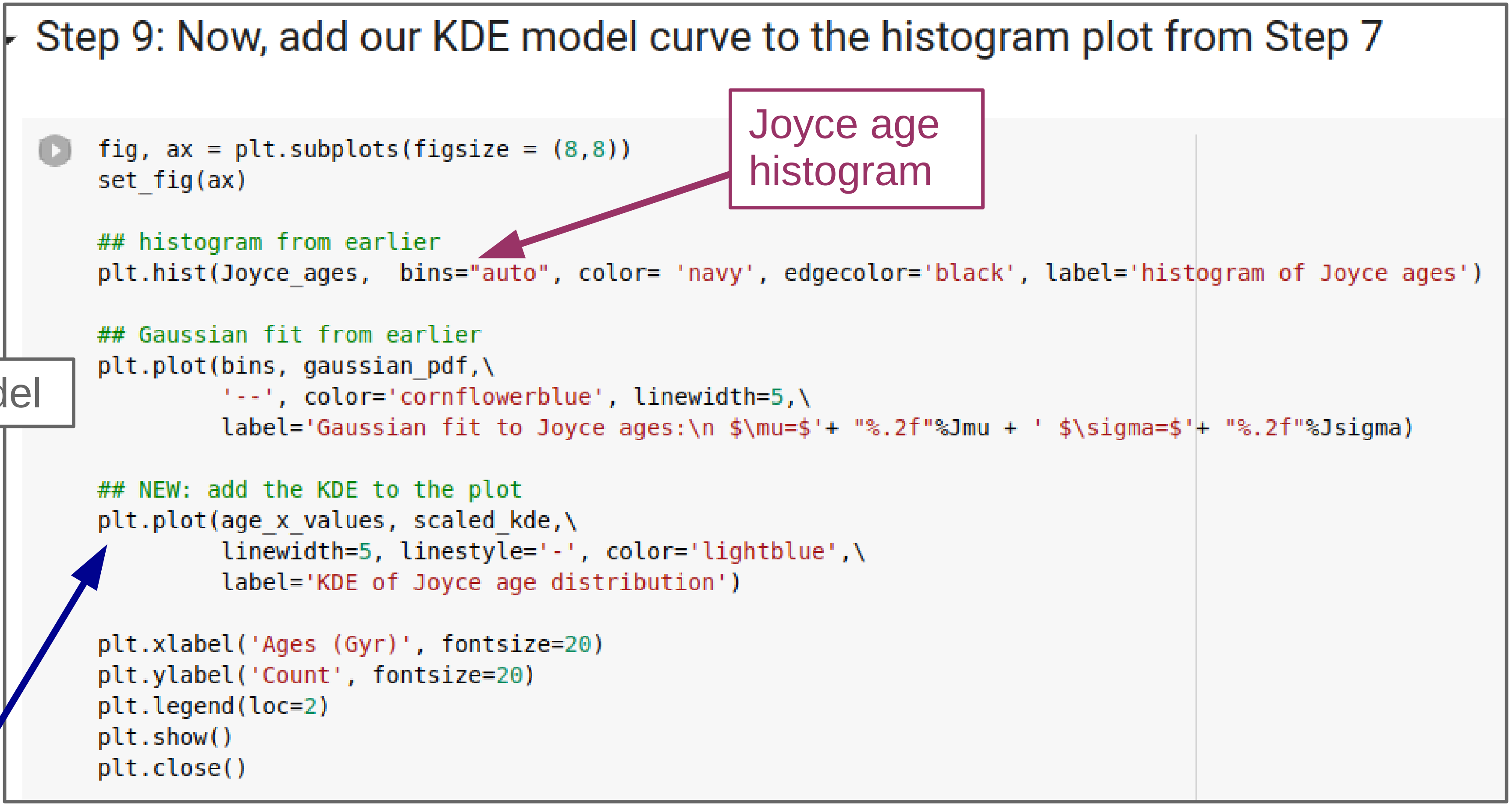
the following line evaluates the kde_model function we made at the beginning of Step 8 over the smoother array of x values defined above.

```
[ ] 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



Step 9: Now, add our KDE model curve to the histogram plot from Step 7

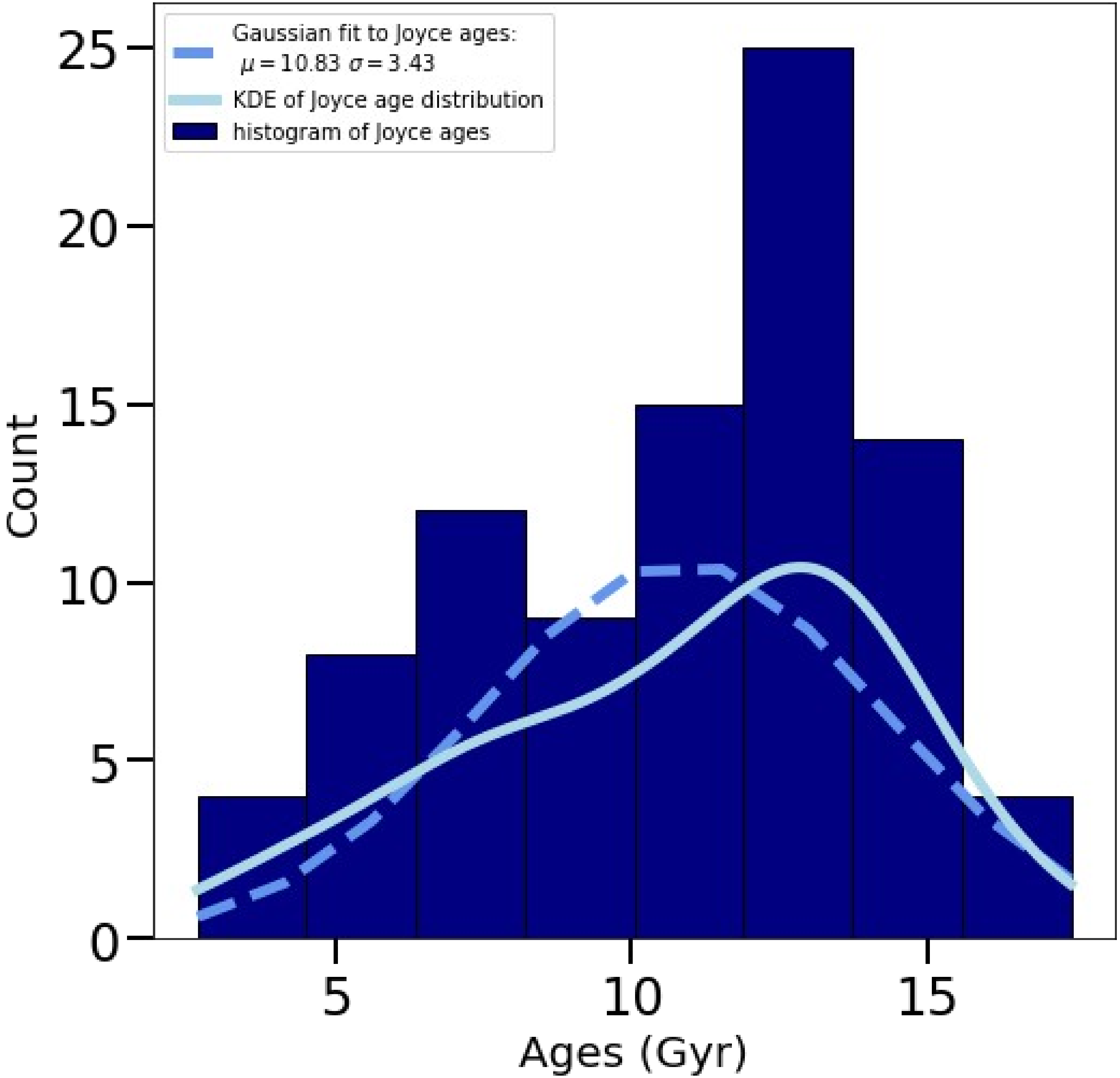
```
fig, ax = plt.subplots(figsize = (8,8))
set_fig(ax)

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

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

## NEW: add the KDE to the plot
plt.plot(age_x_values, scaled_kde,\
         linewidth=5, linestyle='-', color='lightblue',\
         label='KDE of Joyce age distribution')

plt.xlabel('Ages (Gyr)', fontsize=20)
plt.ylabel('Count', fontsize=20)
plt.legend(loc=2)
plt.show()
plt.close()
```

Joyce age histogram

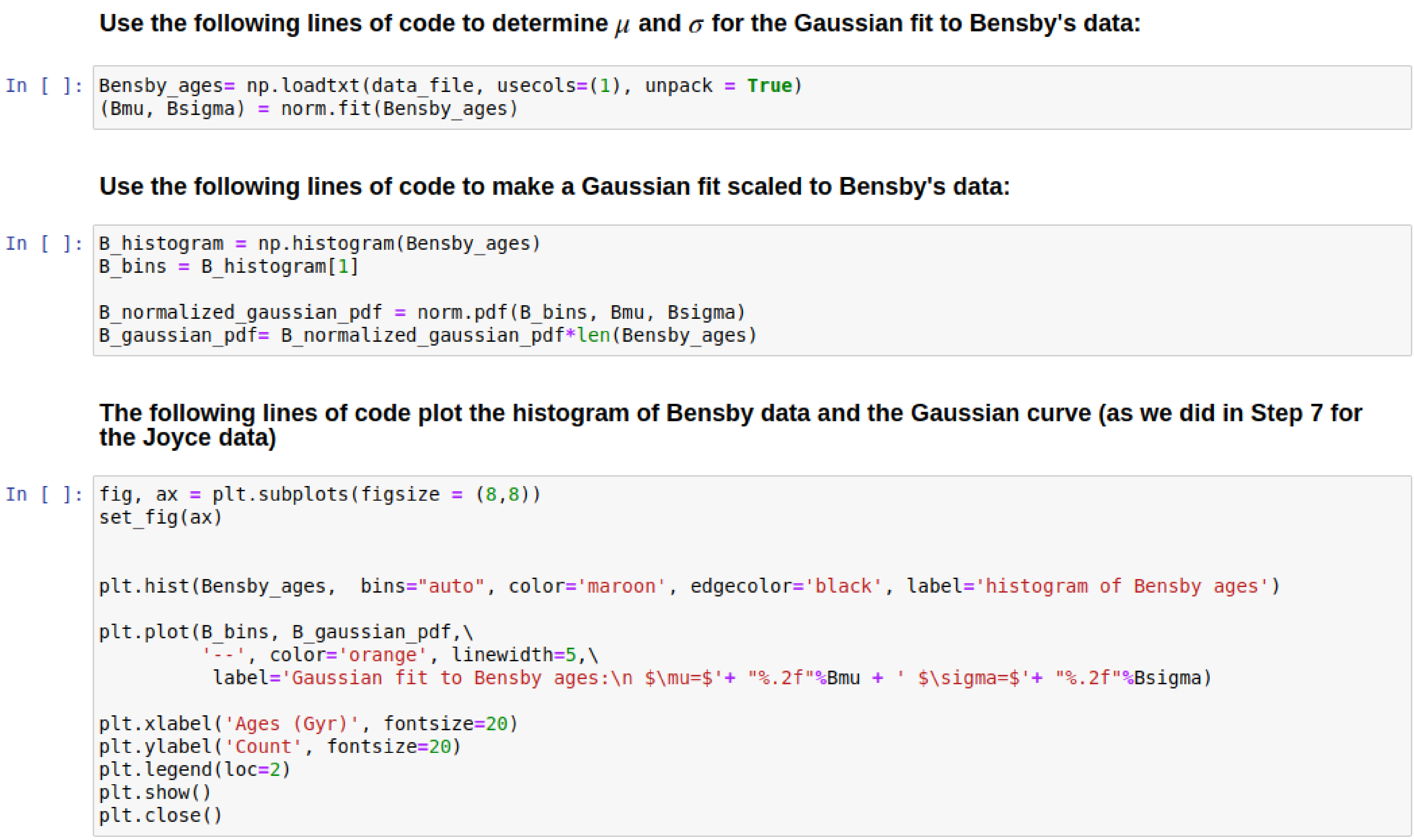
Gaussian model

NEW: KDE model

Step 9: Compare KDE graphically



Exercise 1: Make a Gaussian model for the Bensby age measurements



Use the following lines of code to determine μ and σ for the Gaussian fit to Bensby's data:

In [ ]:
```
Bensby_ages= np.loadtxt(data_file, usecols=(1), unpack = True)
(Bmu, Bsigma) = norm.fit(Bensby_ages)
```

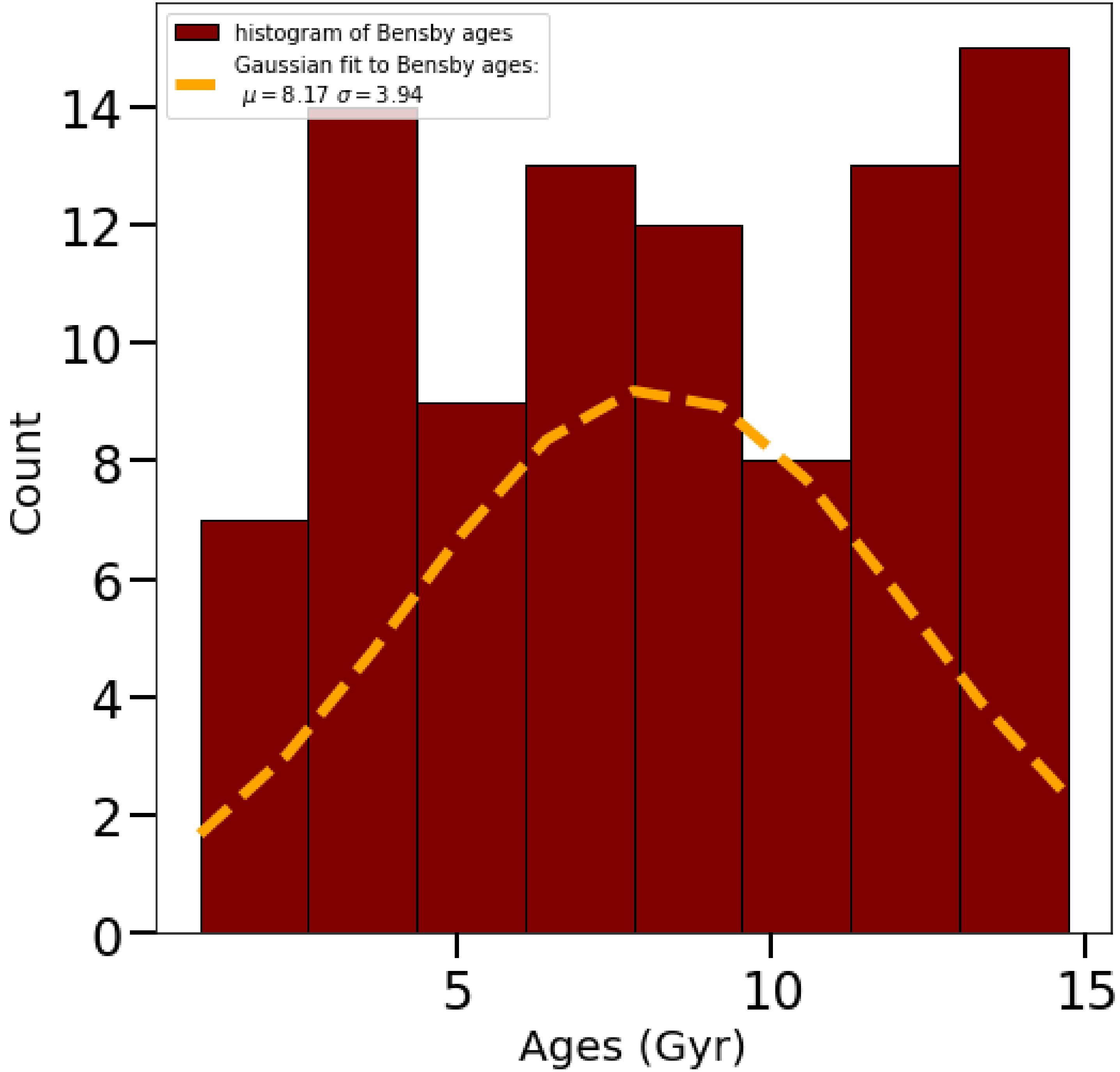
Use the following lines of code to make a Gaussian fit scaled to Bensby's data:

In [ ]:
```
B_histogram = np.histogram(Bensby_ages)
B_bins = B_histogram[1]

B_normalized_gaussian_pdf = norm.pdf(B_bins, Bmu, Bsigma)
B_gaussian_pdf= B_normalized_gaussian_pdf*len(Bensby_ages)
```

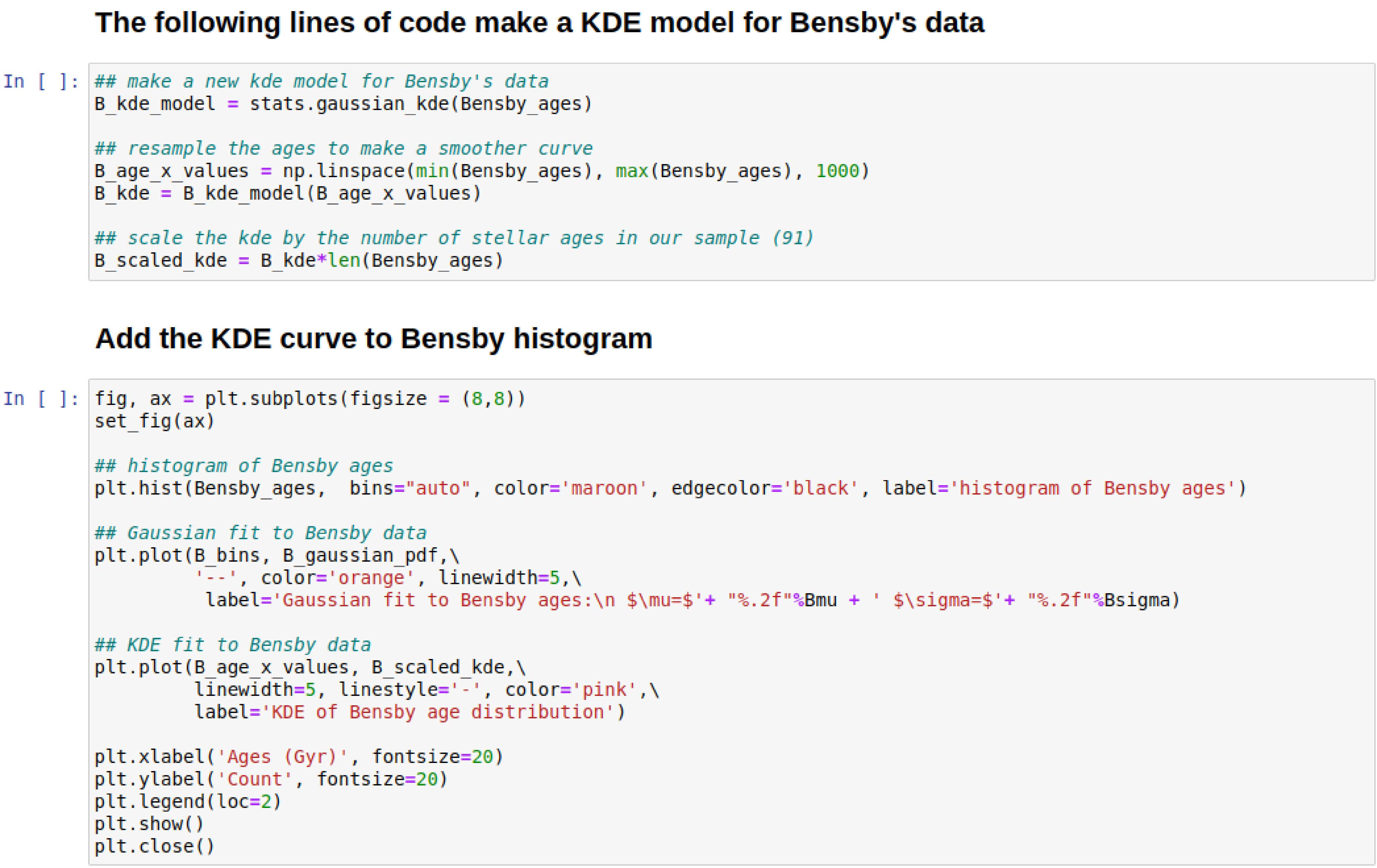
The following lines of code plot the histogram of Bensby data and the Gaussian curve (as we did in Step 7 for the Joyce data)

In [ ]:
```
fig, ax = plt.subplots(figsize = (8,8))
set_fig(ax)


plt.hist(Bensby_ages,  bins="auto", color='maroon', edgecolor='black', label='histogram of Bensby ages')

plt.plot(B_bins, B_gaussian_pdf,\
         '--', color='orange', linewidth=5,\
          label='Gaussian fit to Bensby ages:\n $\mu=$'+ "%.2f"%Bmu + ' $\sigma=$'+ "%.2f"%Bsigma)

plt.xlabel('Ages (Gyr)', fontsize=20)
plt.ylabel('Count', fontsize=20)
plt.legend(loc=2)
plt.show()
plt.close()
```

SOLUTION to Exercise 1:



Exercise 2: Make a KDE for Bensby’s data



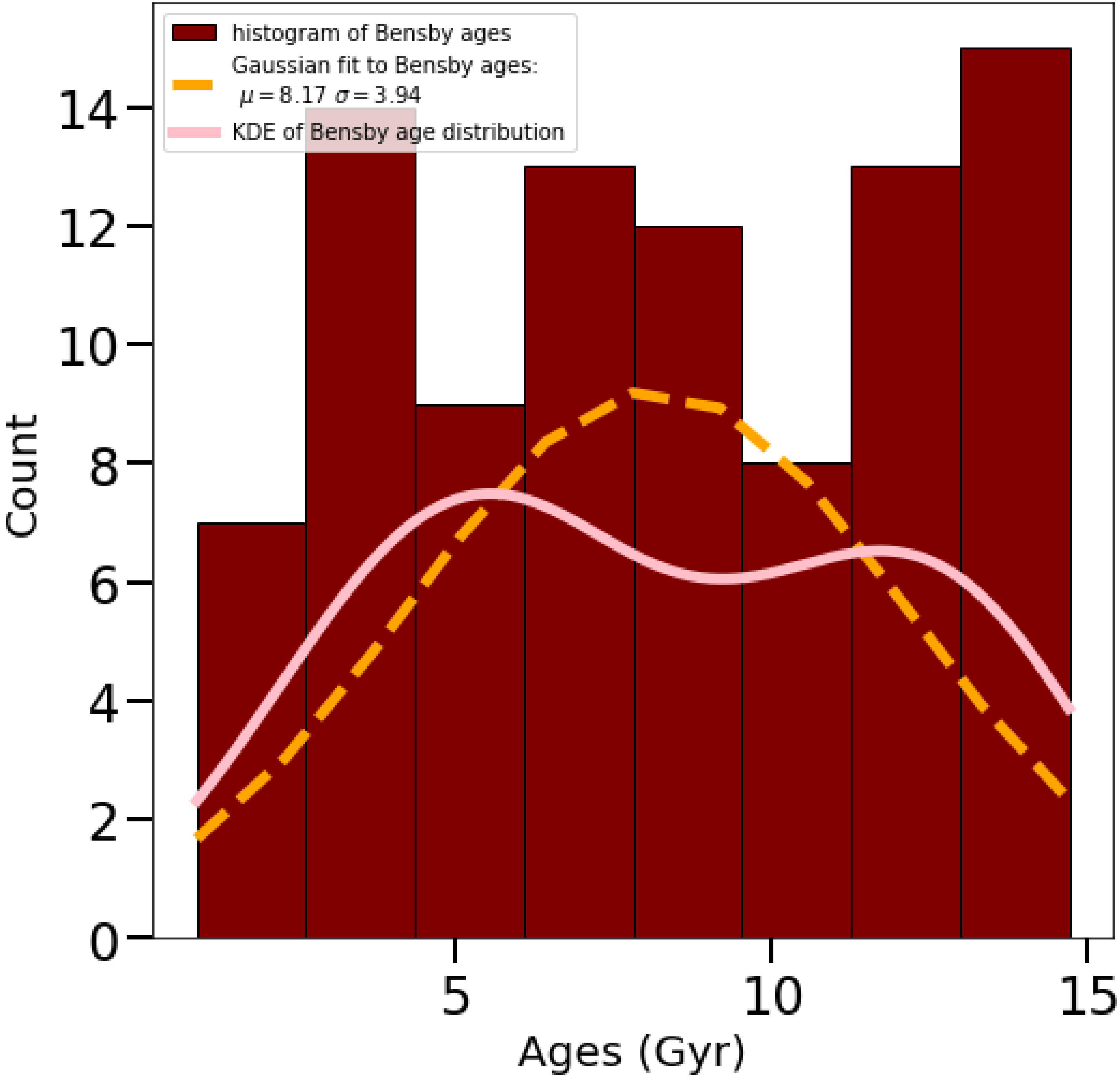
The following lines of code make a KDE model for Bensby's data

```
In [ ]: ## make a new kde model for Bensby's data
        B_kde_model = stats.gaussian_kde(Bensby_ages)

        ## resample the ages to make a smoother curve
        B_age_x_values = np.linspace(min(Bensby_ages), max(Bensby_ages), 1000)
        B_kde = B_kde_model(B_age_x_values)

        ## scale the kde by the number of stellar ages in our sample (91)
        B_scaled_kde = B_kde*len(Bensby_ages)
```

Add the KDE curve to Bensby histogram

```
In [ ]: fig, ax = plt.subplots(figsize = (8,8))
        set_fig(ax)

        ## histogram of Bensby ages
        plt.hist(Bensby_ages,  bins="auto", color='maroon', edgecolor='black', label='histogram of Bensby ages')

        ## Gaussian fit to Bensby data
        plt.plot(B_bins, B_gaussian_pdf,\
                 '--', color='orange', linewidth=5,\
                  label='Gaussian fit to Bensby ages:\n $\mu=$'+ "%.2f"%Bmu + ' $\sigma=$'+ "%.2f"%Bsigma)

        ## KDE fit to Bensby data
        plt.plot(B_age_x_values, B_scaled_kde,\
                 linewidth=5, linestyle='-', color='pink',\
                 label='KDE of Bensby age distribution')

        plt.xlabel('Ages (Gyr)', fontsize=20)
        plt.ylabel('Count', fontsize=20)
        plt.legend(loc=2)
        plt.show()
        plt.close()
```

SOLUTION to Exercise 2:



Generative Models

Here’s where the *machine learning* aspect really comes in:

Generative Models

Here's where the *machine learning* aspect really comes in:

(1) Given an *observed distribution* (the stellar age data), we have inferred the pdf – probability density function – describing this distribution using **kernel density estimation**

Generative Models

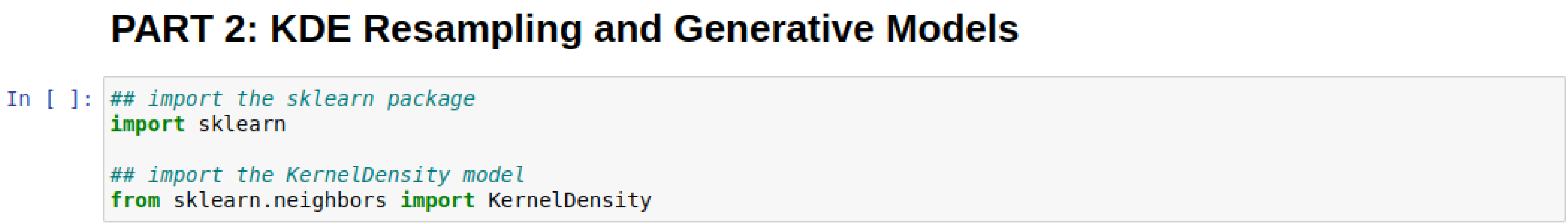
Here's where the *machine learning* aspect really comes in:

(1) Given an *observed distribution* (the stellar age data), we have inferred the pdf – probability density function – describing this distribution using **kernel density estimation**

(2) We can now use the KDE model to **make predictions** about similar situations *that we cannot observe*

sklearn: extremely powerful Python module for ML and data science applications

sklearn: extremely powerful Python module for ML and data science applications

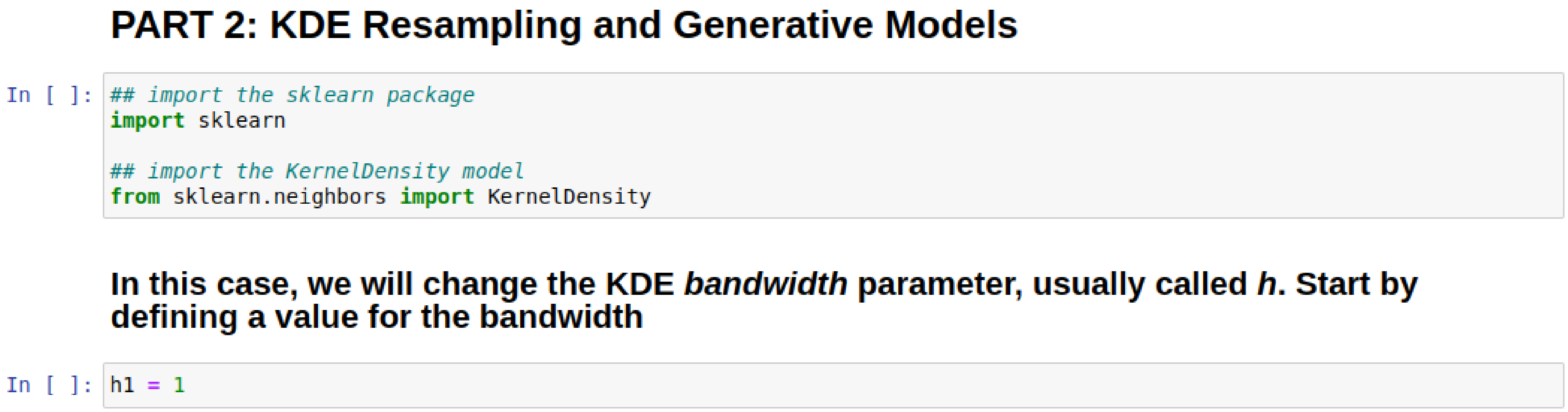


PART 2: KDE Resampling and Generative Models

In [ ]:

```
## import the sklearn package
import sklearn

## import the KernelDensity model
from sklearn.neighbors import KernelDensity
```

sklearn: extremely powerful Python module for ML and data science applications



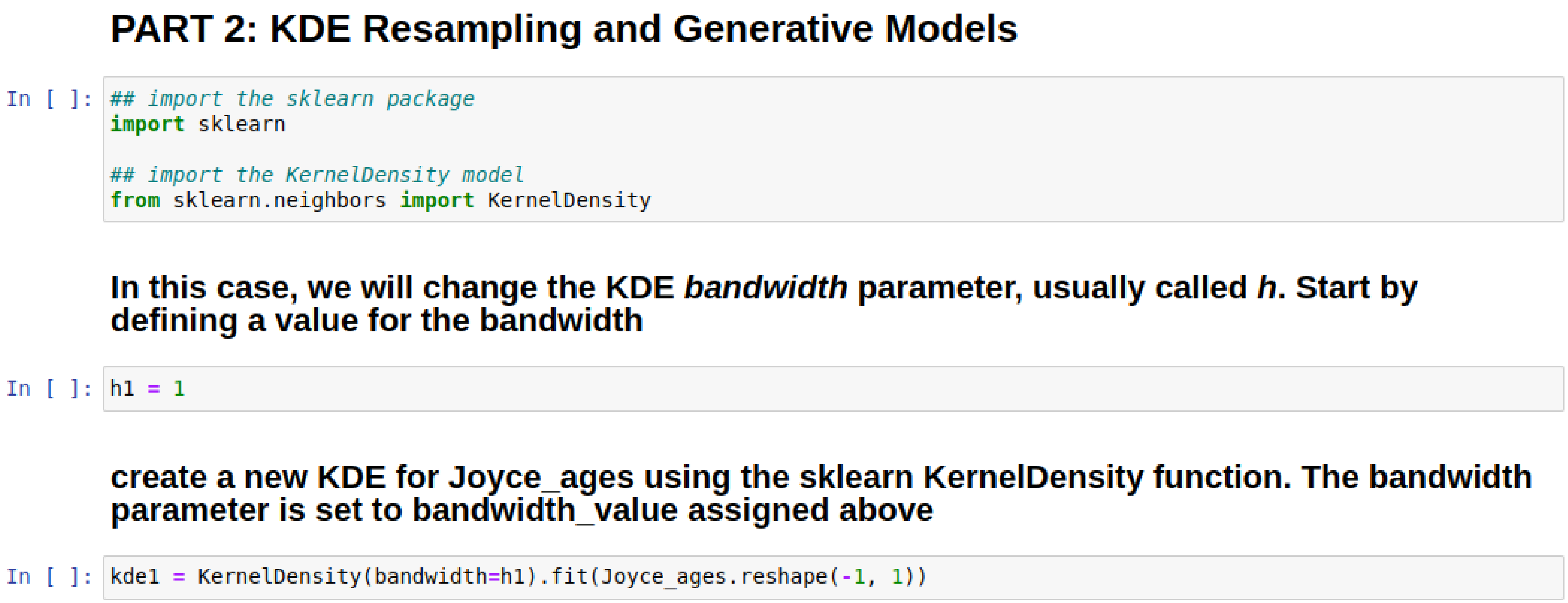
PART 2: KDE Resampling and Generative Models

```
In [ ]: ## import the sklearn package
        import sklearn

        ## import the KernelDensity model
        from sklearn.neighbors import KernelDensity
```

In this case, we will change the KDE *bandwidth* parameter, usually called *h*. Start by defining a value for the bandwidth

```
In [ ]: h1 = 1
```

sklearn: extremely powerful Python module for ML and data science applications



PART 2: KDE Resampling and Generative Models

In [ ]:

```
## import the sklearn package
import sklearn

## import the KernelDensity model
from sklearn.neighbors import KernelDensity
```

In this case, we will change the KDE *bandwidth* parameter, usually called *h*. Start by defining a value for the bandwidth

In [ ]:

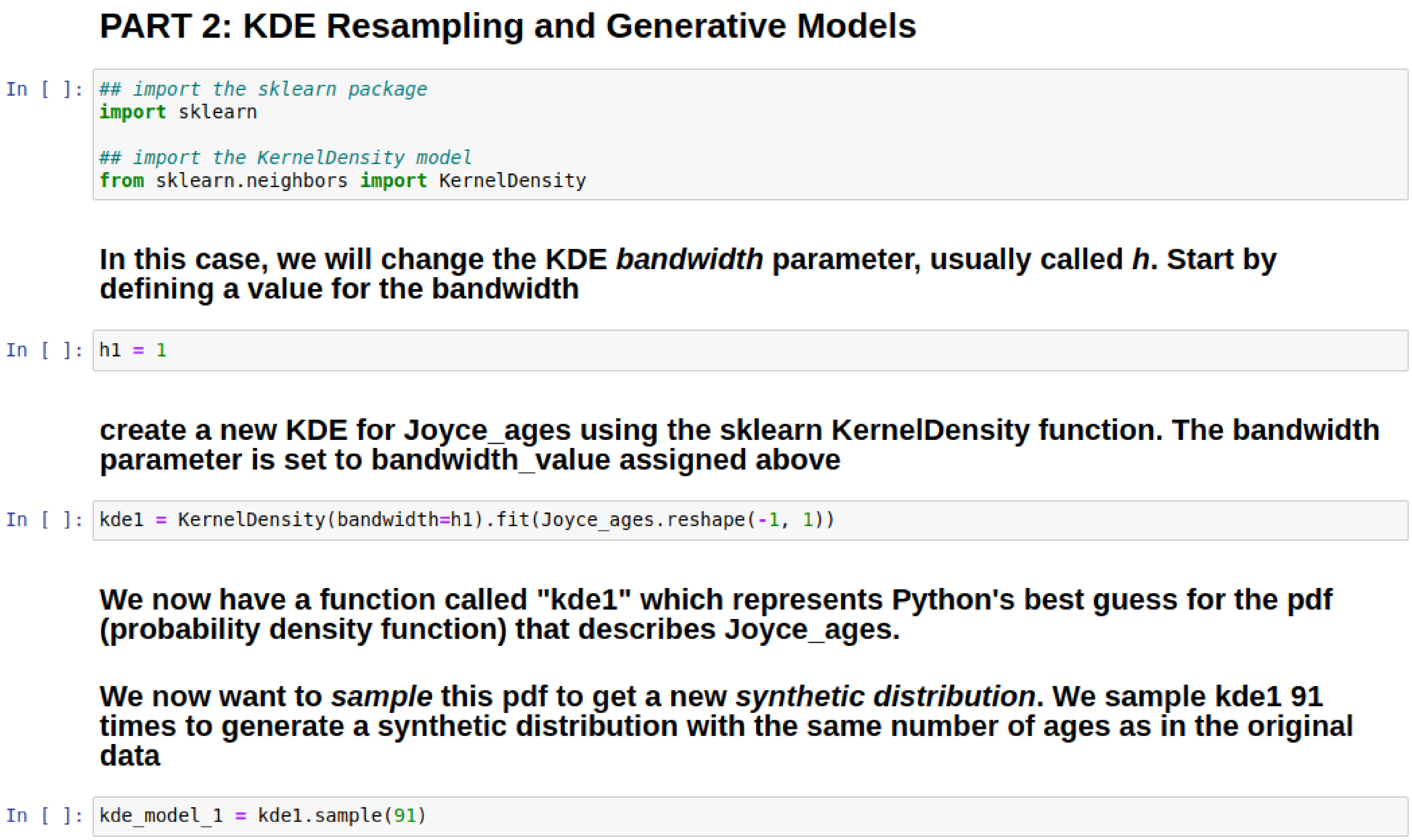
```
h1 = 1
```

create a new KDE for Joyce_ages using the sklearn KernelDensity function. The bandwidth parameter is set to bandwidth_value assigned above

In [ ]:

```
kde1 = KernelDensity(bandwidth=h1).fit(Joyce_ages.reshape(-1, 1))
```

sklearn: extremely powerful Python module for ML and data science applications



PART 2: KDE Resampling and Generative Models

```
In [ ]: ## import the sklearn package
        import sklearn

        ## import the KernelDensity model
        from sklearn.neighbors import KernelDensity
```

In this case, we will change the KDE *bandwidth* parameter, usually called *h*. Start by defining a value for the bandwidth

```
In [ ]: h1 = 1
```

create a new KDE for Joyce_ages using the sklearn KernelDensity function. The bandwidth parameter is set to bandwidth_value assigned above

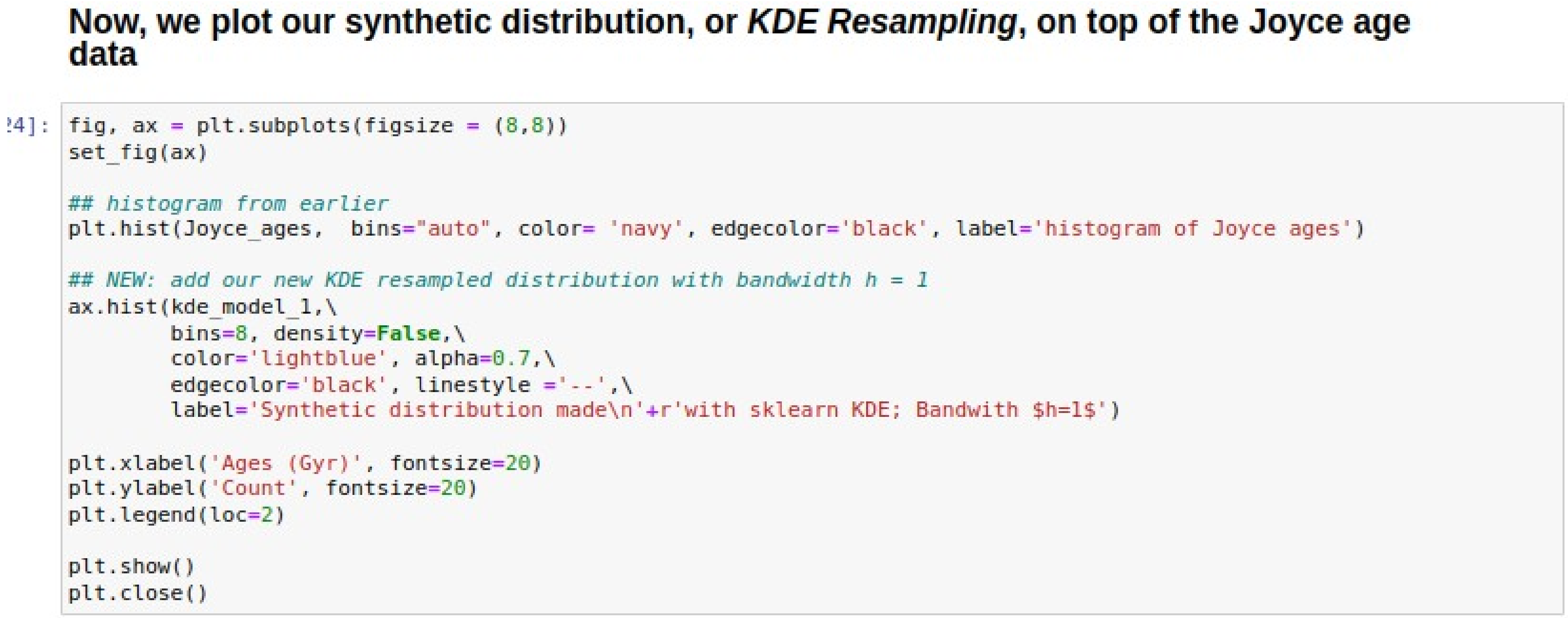
```
In [ ]: kde1 = KernelDensity(bandwidth=h1).fit(Joyce_ages.reshape(-1, 1))
```

We now have a function called "kde1" which represents Python's best guess for the pdf (probability density function) that describes Joyce_ages.

We now want to *sample* this pdf to get a new *synthetic distribution*. We sample kde1 91 times to generate a synthetic distribution with the same number of ages as in the original data

```
In [ ]: kde_model_1 = kde1.sample(91)
```

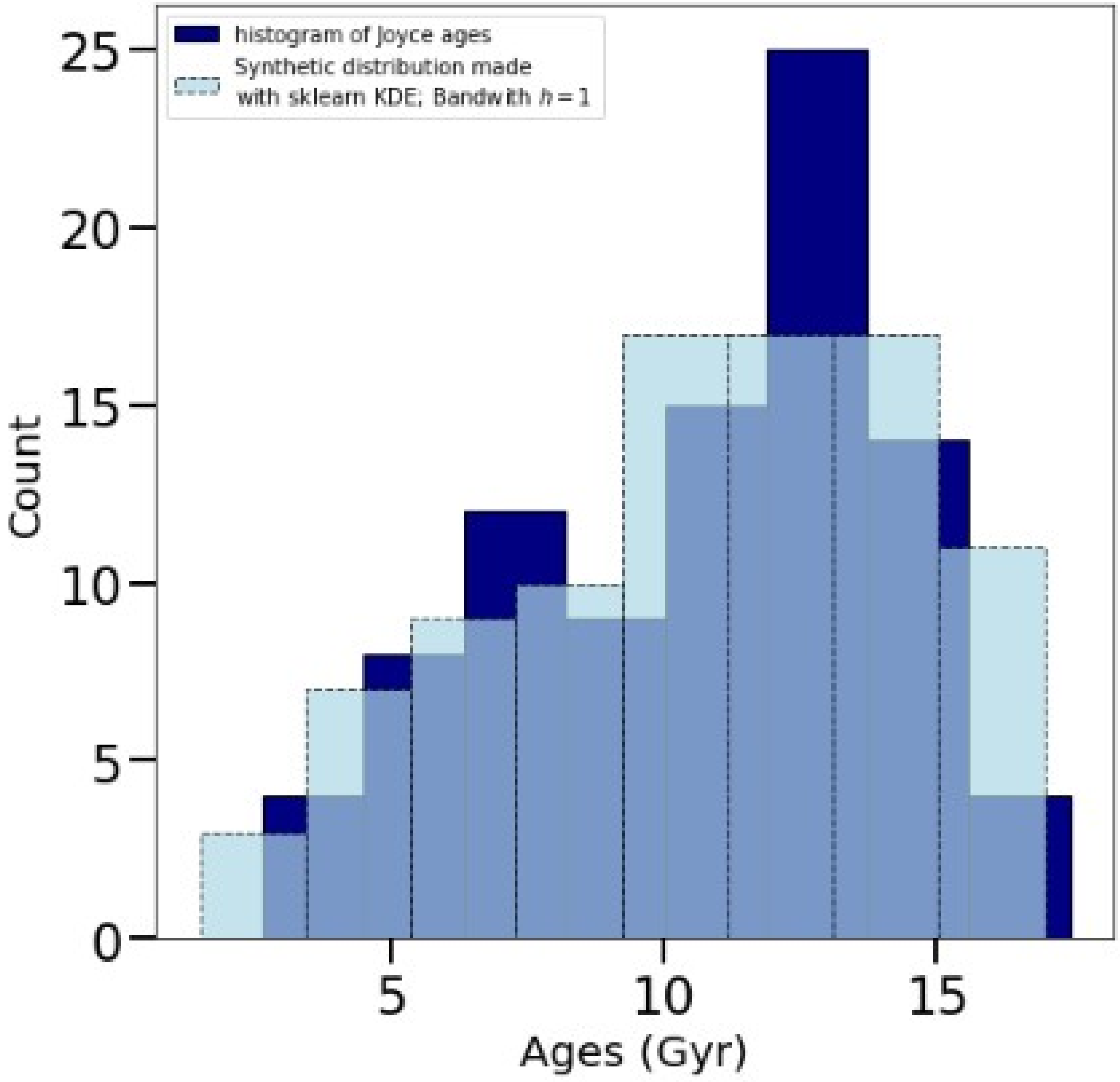
Plot your synthetic distribution, generated by sampling the KDE



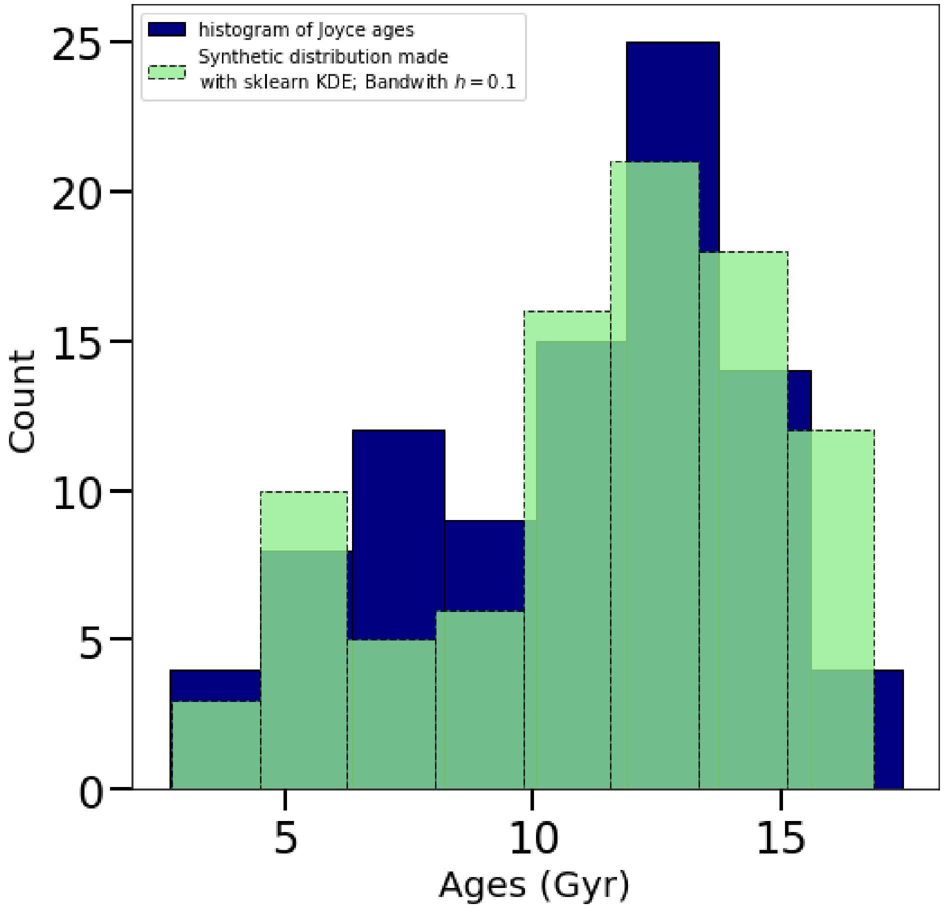
Now, we plot our synthetic distribution, or *KDE Resampling*, on top of the Joyce age data

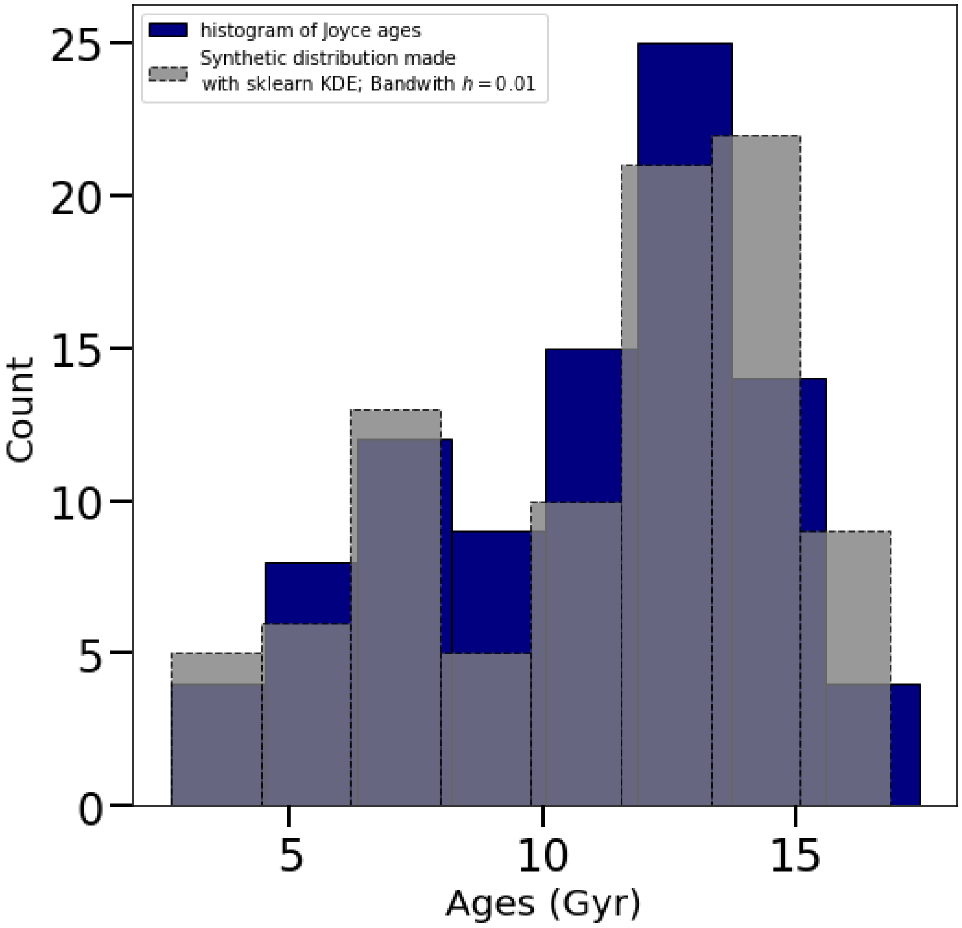
```
fig, ax = plt.subplots(figsize = (8,8))
set_fig(ax)

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

## NEW: add our new KDE resampled distribution with bandwidth h = 1
ax.hist(kde_model_1,\
        bins=8, density=False,\
        color='lightblue', alpha=0.7,\
        edgecolor='black', linestyle ='--',\
        label='Synthetic distribution made\n'+r'with sklearn KDE; Bandwith $h=1$')

plt.xlabel('Ages (Gyr)', fontsize=20)
plt.ylabel('Count', fontsize=20)
plt.legend(loc=2)

plt.show()
plt.close()
```

Plot your synthetic distribution, generated by sampling the KDE

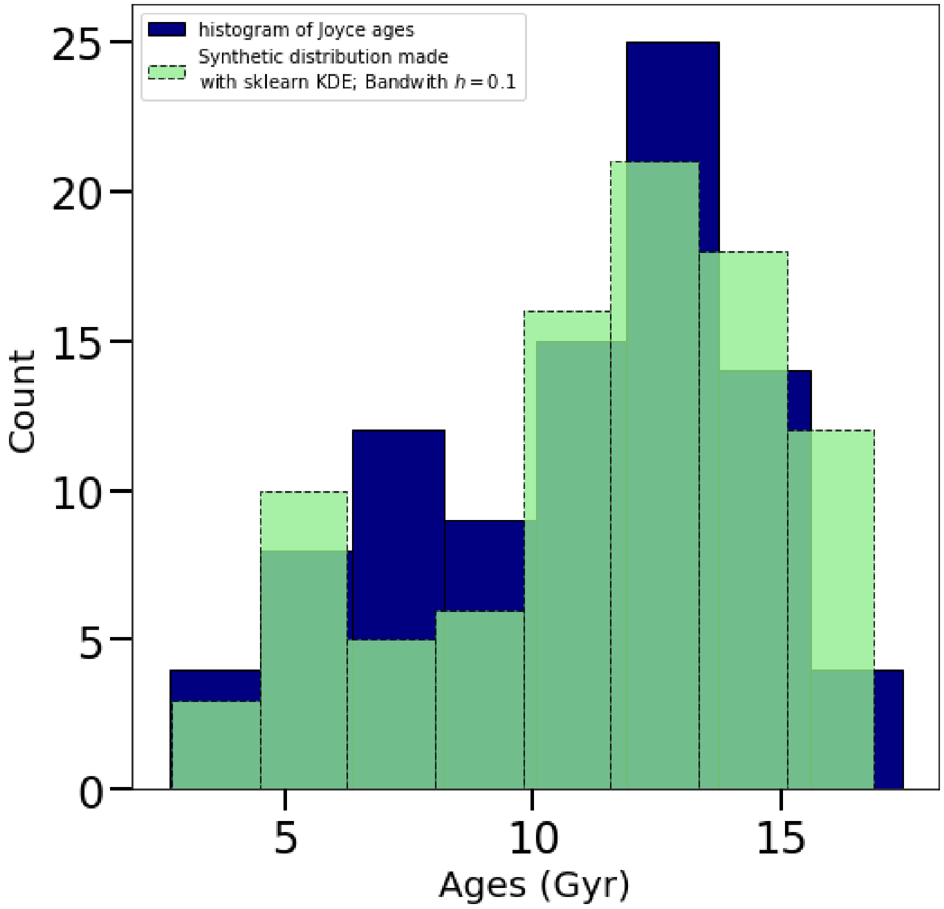


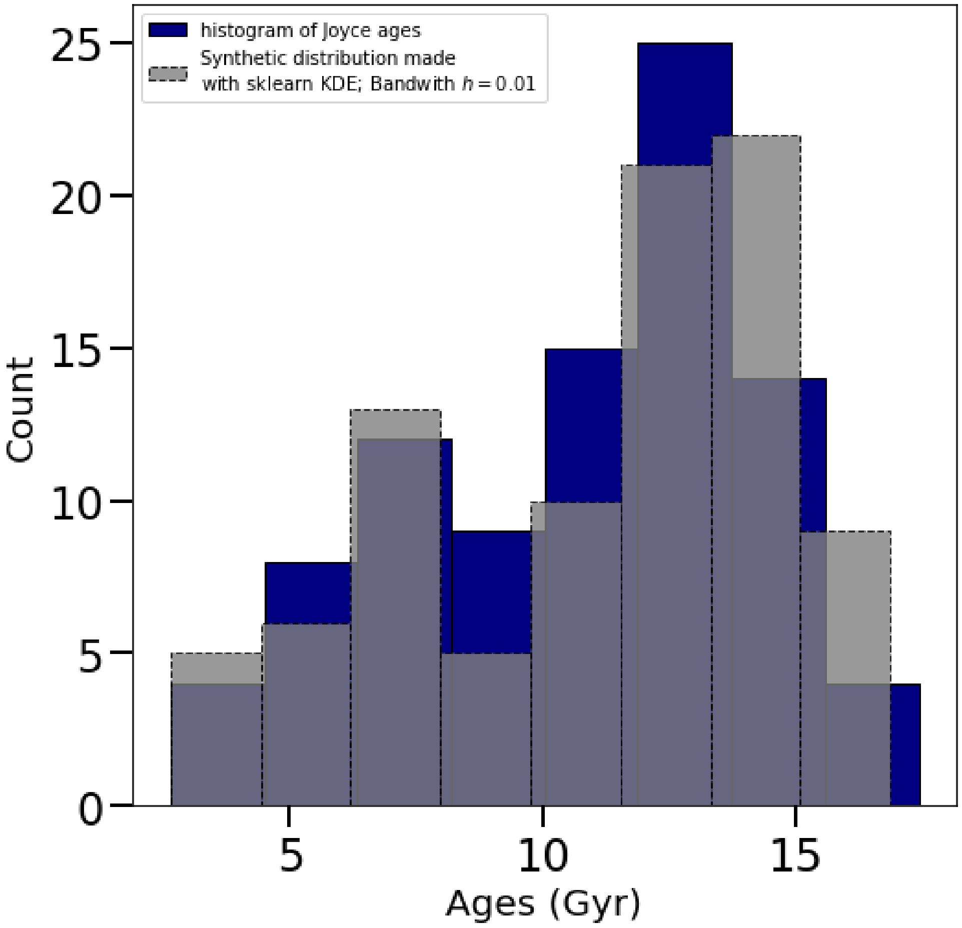
Generative Models – what's going on?





Generative Models – what's going on?





We have given the machine real data (Joyce stellar ages) and asked it to generate new information: the synthetic age distributions (green and grey), made by sampling our KDE

Questions to think about:

Questions to think about:

- What is the impact of changing the bandwidth parameter?

Questions to think about:

- What is the impact of changing the bandwidth parameter?

- How would predictions made by models trained on my age measurements differ from predictions made by models trained on Bensby’s age measurements?

Questions to think about:

- What is the impact of changing the bandwidth parameter?

- How would predictions made by models trained on my age measurements differ from predictions made by models trained on Bensby's age measurements?

- How might we decide which training set is better?

Questions to think about:

- What is the impact of changing the bandwidth parameter?

- How would predictions made by models trained on my age measurements differ from predictions made by models trained on Bensby's age measurements?

- How might we decide which training set is better?

- An important cautionary tale: machine learning is incredibly powerful, but if the training set is biased, unrepresentative, or otherwise poor, the predictions based on it will be, too

Questions to think about:

- What is the impact of changing the bandwidth parameter?

- How would predictions made by models trained on my age measurements differ from predictions made by models trained on Bensby's age measurements?

- How might we decide which training set is better?

- An important cautionary tale: machine learning is incredibly powerful, but if the training set is biased, unrepresentative, or otherwise poor, the predictions based on it will be, too

Thank you for your attention!