Estadística descriptiva aplicada: distribuciones.

Ahora vamos a encontrar la probabilidad de encontrar un determinado valor.

¿Cómo visualizar una distribución?

- Histograma
- Función de probabilidad de masas (PMF)
- Función de distribución acumulada (CDF)
- Función de probabilidad de densidad (PDF)

Lo que nos importa ahora es encontrar la probabilidad de encontrar un cierto número o la probabilidad de encontrar un número menor a el, o también cual es la probabilidad de encuentre un valor entre 2 sitios. Estas son las funciones de probabilidad.

Función de probabilidad de masas (PMFs)

Nos dice la probabilidad de que una variable aleatoria discreta tome un valor determinado.

Por ejemplo la edad. ¿Cuál es la probabilidad de que en mi salón haya personas que tengan 23 años?

Función de distribución acumulada (CDFs)

Devuelve la probabilidad de que una variable sea igual o menor que un valor determinado.

Siguiendo el ejemplo del salón. ¿Cuál es la probabilidad de que haya personas que tengan 23 años o menos años?

Función de probabilidad de densidad (PDFs)

Determina la probabilidad de que una variable continua tome un valor determinado.

Por ejemplo la estatura. ¿Cuál es la probabilidad de que encuentre un pingüinos que mida 1.72 m?

Al ser números continuos podremos hacernos esas preguntas y ademas nos va a servir para realizarnos preguntas como por ejemplo:

¿Cuál es la probabilidad de que encuentre un pingüino que mida entre un valor A a B?

In [ ]:
```
# Importando librerías
import empiricaldist
import janitor
import matplotlib.pyplot as plt
import numpy as np
import palmerpenguins
import pandas as pd
import scipy.stats
import seaborn as sns
import sklearn.metrics
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats as ss
import session_info
```

Establecer apariencia general de las gráficas

In [ ]:
```
%matplotlib inline
sns.set_style(style='whitegrid')
sns.set_context(context='notebook')
plt.rcParams['figure.figsize'] = (11, 9.4)

penguin_color = {
    'Adelie': '#ff6602ff',
    'Gentoo': '#0f7175ff',
    'Chinstrap': '#c65dc9ff'
}
```

Cargar los datos

Datos Preprocesados

In [ ]:
```
preprocessed_penguins_df = pd.read_csv('dataset/penguins.csv')
```

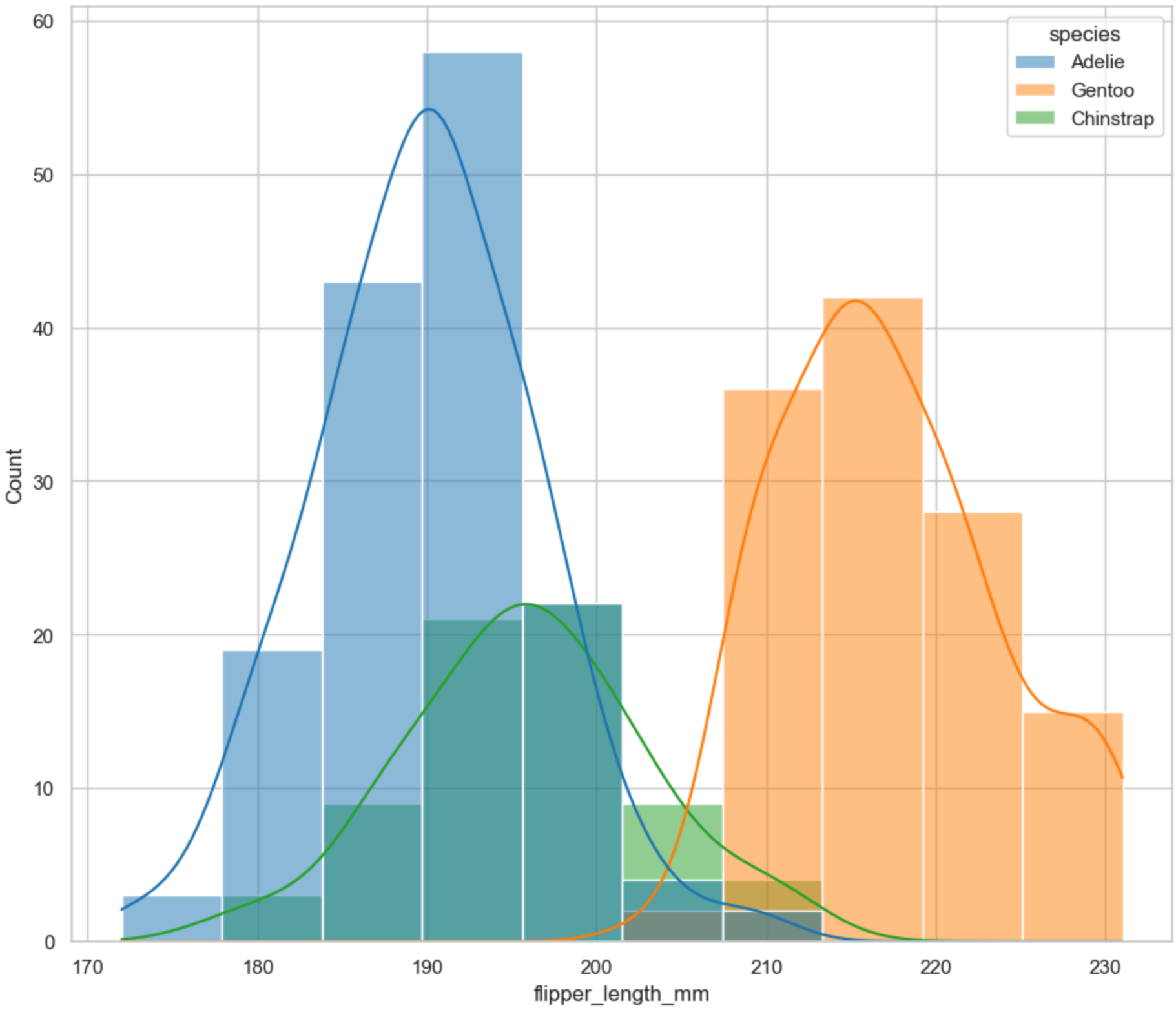
Distribuciones: PMF, CDF y PDF

Función de probabilidad de masas (PFM)

Utilizando Seaborn

In [ ]:
```
#Usando un histograma
sns.histplot(
    data=preprocessed_penguins_df,
    x='flipper_length_mm',
    hue='species',
    kde='True'
)
```

Out[ ]: <AxesSubplot: xlabel='flipper_length_mm', ylabel='Count'>



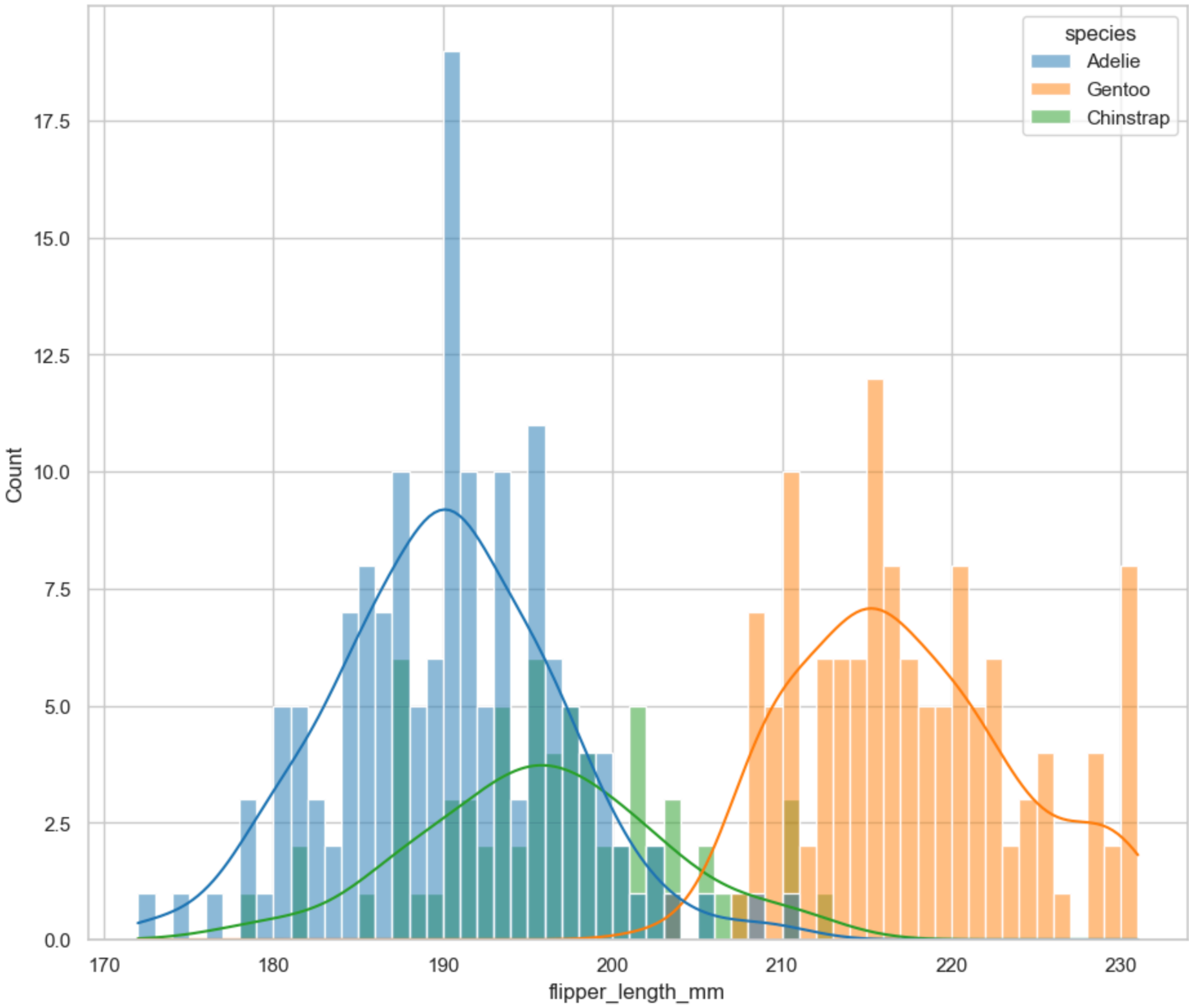
Lo que tenemos en la gráfica anterior es la longitud de aletas de nuestros pingüinos. En el cual tenemos un número de intervalos y calcula que cantidad de números cae en ciertos intervalos.

Lo que nos interesa en la **función de probabilidad de masas (PMF)**, es encontrar para cada valor único **cuanta frecuencia tiene**.

Como tenemos valores discretos y enteros de longitud, es decir 20, 31, etc. Nos interesa tener estaturas enteras y que no estén dentro de ciertos rangos. Le asignaremos un ancho al `bin` de 1.

In [ ]:
```
#Usando un histograma
sns.histplot(
    data=preprocessed_penguins_df,
    x='flipper_length_mm',
    hue='species',
    kde='True',
    binwidth=1
)
```

Out[ ]: <AxesSubplot: xlabel='flipper_length_mm', ylabel='Count'>



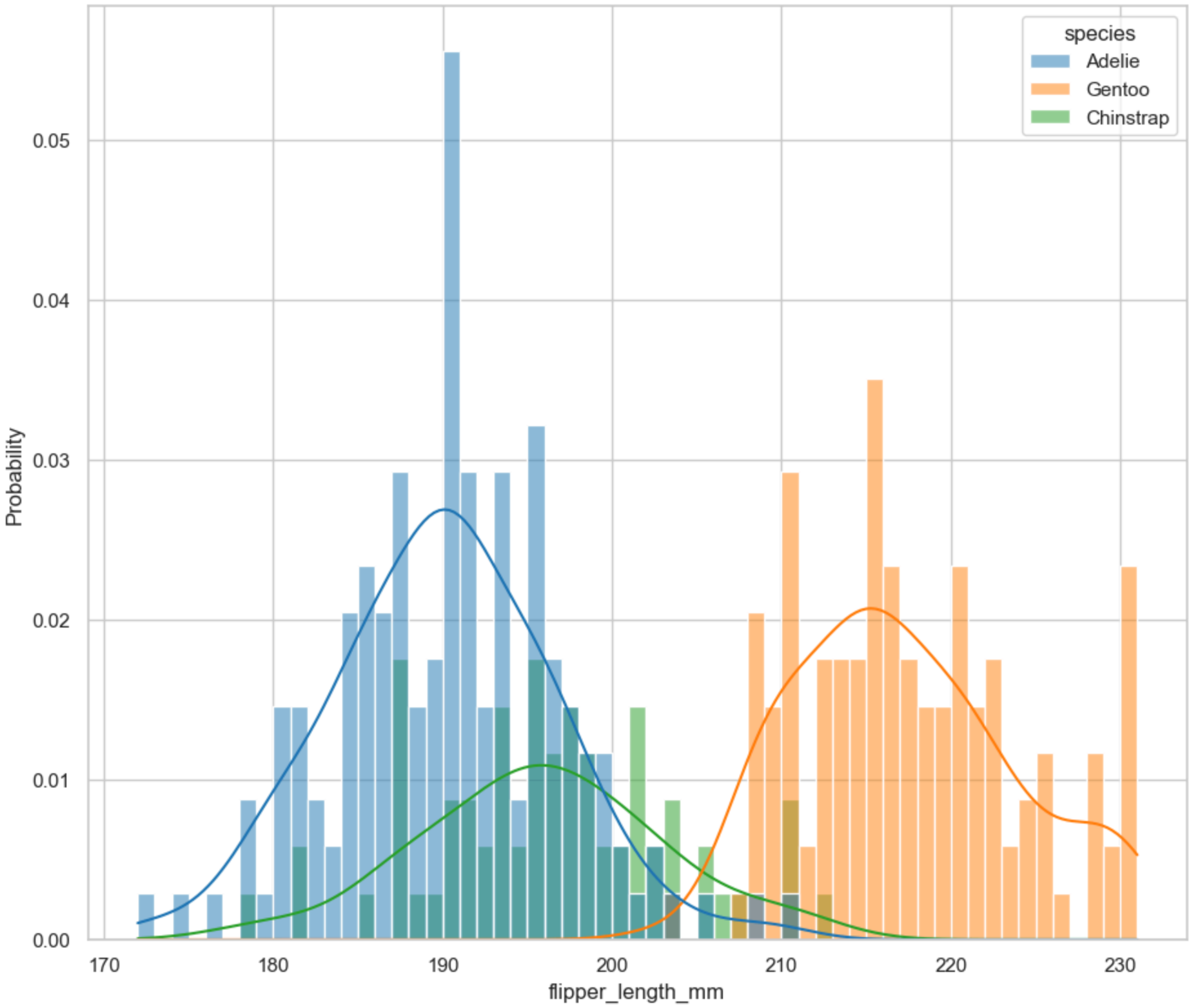
Aquí nos damos cuenta de que tenemos ciertos valores y que no tenemos pingüinos que caigan dentro de ese valor. Es decir no existen observaciones con ese número.

Así de esta manera vemos que existen zonas con huecos y otras con altas frecuencias de coincidencia.

Observamos los valores en el eje Y (conteo), pero la función de densidad de probabilidad es un valor diferente, así que ajustemos.

In [ ]:
```
#Usando un histograma
sns.histplot(
    data=preprocessed_penguins_df,
    x='flipper_length_mm',
    hue='species',
    kde='True',
    binwidth=1,
    stat='probability'
)
```

Out[ ]:
```
<AxesSubplot: xlabel='flipper_length_mm', ylabel='Probability'>
```



Ahora en lugar de que las barras representen su frecuencia, representan su probabilidad. Listo ya tenemos un PMF.

Utilizando `empiricaldist()`

Existen otras librerías que me pueden entregar los valores. Solo tenemos que hacer uso de ellas.

In [ ]:
```
empiricaldist.Pmf.from_seq(
    preprocessed_penguins_df.flipper_length_mm
)
```

Out[ ]:

	probs
172.0	0.002924
174.0	0.002924
176.0	0.002924
178.0	0.011696
179.0	0.002924
180.0	0.014620
181.0	0.020468
182.0	0.008772
183.0	0.005848
184.0	0.020468
185.0	0.026316
186.0	0.020468
187.0	0.046784
188.0	0.017544
189.0	0.020468
190.0	0.064327
191.0	0.038012
192.0	0.020468
193.0	0.043860
194.0	0.014620
195.0	0.049708
196.0	0.029240
197.0	0.029240
198.0	0.023392
199.0	0.017544
200.0	0.011696
201.0	0.017544
202.0	0.011696
203.0	0.014620
205.0	0.008772
206.0	0.002924
207.0	0.005848
208.0	0.023392
209.0	0.014620
210.0	0.040936
211.0	0.005848
212.0	0.020468
213.0	0.017544
214.0	0.017544
215.0	0.035088
216.0	0.023392
217.0	0.017544
218.0	0.014620
219.0	0.014620
220.0	0.023392
221.0	0.014620
222.0	0.017544
223.0	0.005848
224.0	0.008772
225.0	0.011696
226.0	0.002924

	probs
228.0	0.011696
229.0	0.005848
230.0	0.020468
231.0	0.002924

Como se puede observar me entrega una lista con sus respectivos valores. Si queremos convertir la lista a frecuencias, solo tenemos que ajustar el parámetro `Normalize=False`.

In [ ]:
```
empiricaldist.Pmf.from_seq(
    preprocessed_penguins_df.flipper_length_mm,
    normalize=False
)
```

Out[ ]:

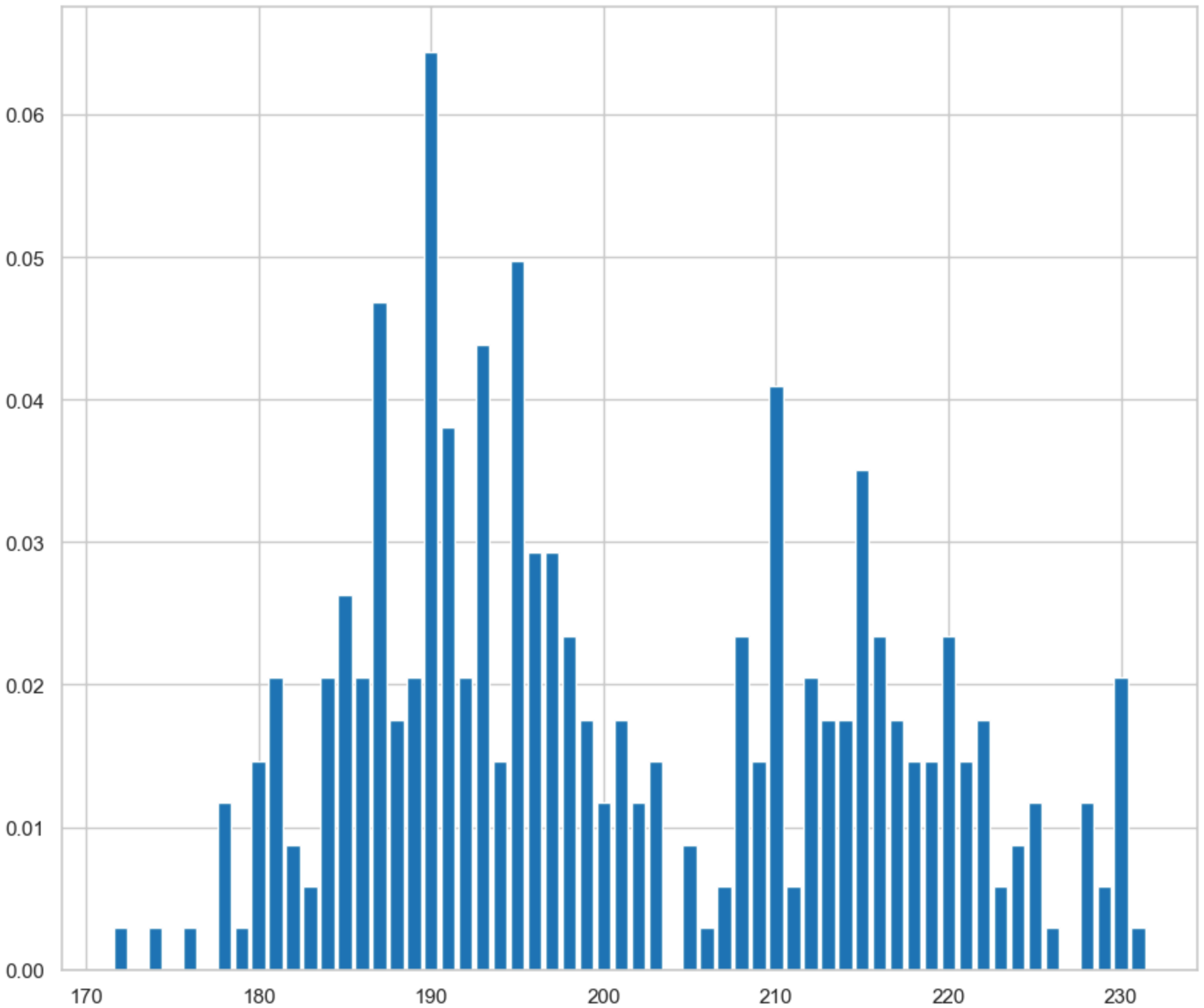
	probs
172.0	1
174.0	1
176.0	1
178.0	4
179.0	1
180.0	5
181.0	7
182.0	3
183.0	2
184.0	7
185.0	9
186.0	7
187.0	16
188.0	6
189.0	7
190.0	22
191.0	13
192.0	7
193.0	15
194.0	5
195.0	17
196.0	10
197.0	10
198.0	8
199.0	6
200.0	4
201.0	6
202.0	4
203.0	5
205.0	3
206.0	1
207.0	2
208.0	8
209.0	5
210.0	14
211.0	2
212.0	7
213.0	6
214.0	6
215.0	12
216.0	8
217.0	6
218.0	5
219.0	5
220.0	8
221.0	5
222.0	6
223.0	2
224.0	3
225.0	4
226.0	1

	probs
228.0	4
229.0	2
230.0	7
231.0	1

Ahora lo que haremos es guardar el conjunto de datos en una variable para ver sus posibles operaciones con este objeto.

In [ ]:
```
pmf_flipper_len_mm = empiricaldist.Pmf.from_seq(
        preprocessed_penguins_df.flipper_length_mm
    )
```

In [ ]:
```
pmf_flipper_len_mm.bar()
```



Como se puede observar nos va a generar una gráfica como en Seaborn, por cada valor discreto que tengamos.

La librería nos permite hacer otras cosas. Por ejemplo calcular la probabilidad de que un pingüino tenga una longitud de ala de 190.

In [ ]:
```
pmf_flipper_len_mm(190)
```

Out[ ]:
```
0.06432748538011696
```

Cómo se puede ver tenemos un parámetro que podemos obtenerlo fácilmente.

In [ ]:
```
pmf_flipper_len_mm(175)
```

Out[ ]:
```
0
```

Ademas obtuvimos la probabilidad de otros valores, pero debido a que no están en el set de datos, nos arroja una probabilidad de 0, como se puede observar.

También podemos obtener parámetros específicos:

In [ ]:
```
preprocessed_penguins_df.flipper_length_mm.max()
```

Out[ ]:
```
231.0
```

Si vemos los valores de la gráfica aparecen con `empiricaldist` y en `Seaborn` no aparecer, esto es debido a la forma en que hacen el cálculo. Entonces puede ser una desventaja en el sentido que en ocasiones te puede colapsar tus datos extremos, pero no es muy grave si queremos una

vista global.

En cambio si queremos preguntas especificas, es mejor usar `empiricaldist`

Función de Probabilidad Acumulada (CDF)

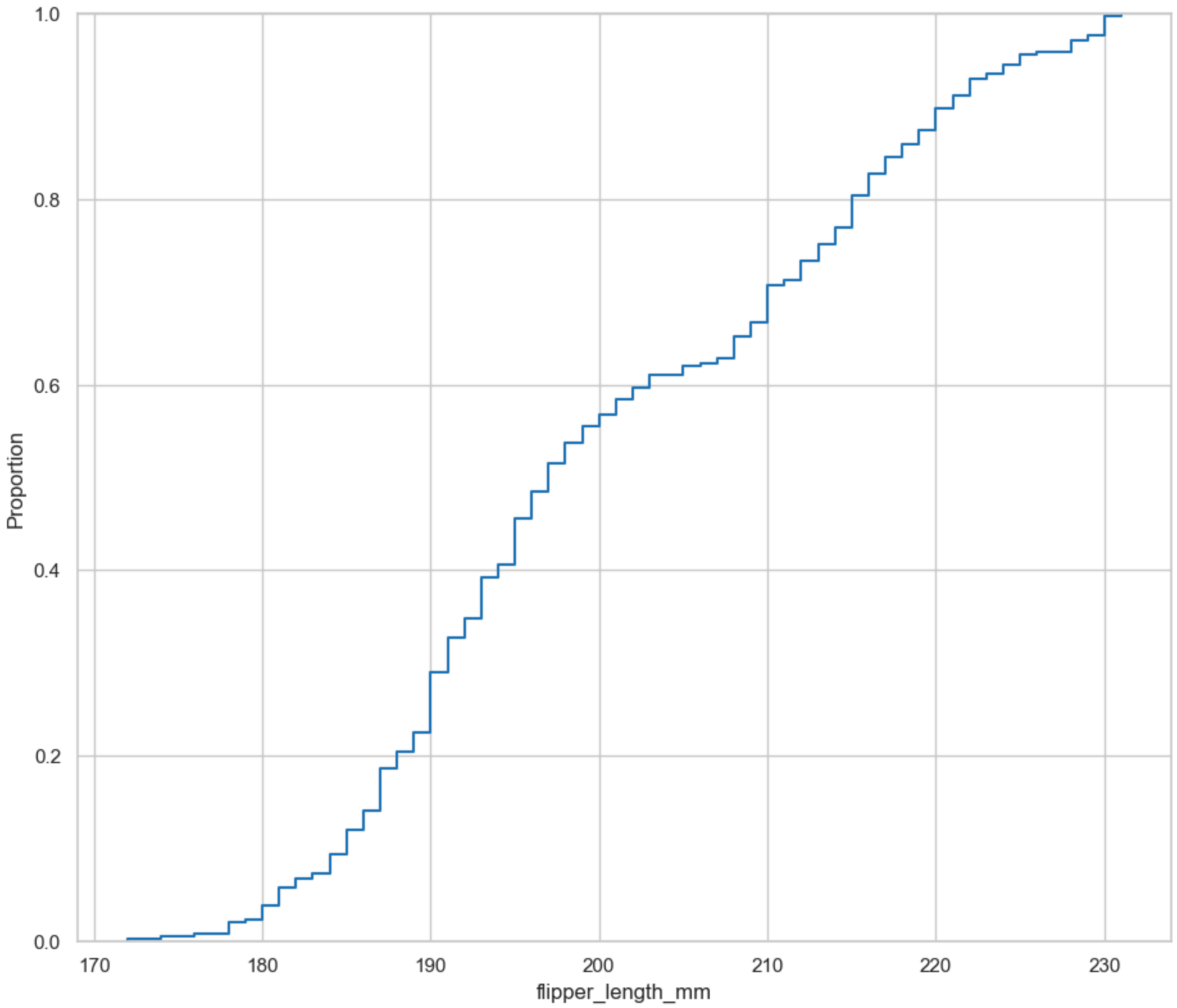
Podemos encontrar probabilidades de un valor o menor a el.

Ejemplo: ¿Cual es la probabilidad de que encuentre un pingüino con una longitud de ala de 180 mm o menos?

Utilizando `Seaborn`

In [ ]:
```
sns.ecdfplot(
    data=preprocessed_penguins_df,
    x='flipper_length_mm'
)
```

Out[ ]:
```
<AxesSubplot: xlabel='flipper_length_mm', ylabel='Proportion'>
```



Listo, como se puede ver es **acumulada**, es decir al inicio es 0 y al final termina en 1. Porque la suma de las probabilidades de todos los eventos siempre será igual a 1.

Entonces como podría preguntar, tendría que hacer una estimación basada en la gráfica.

Por ejemplo:

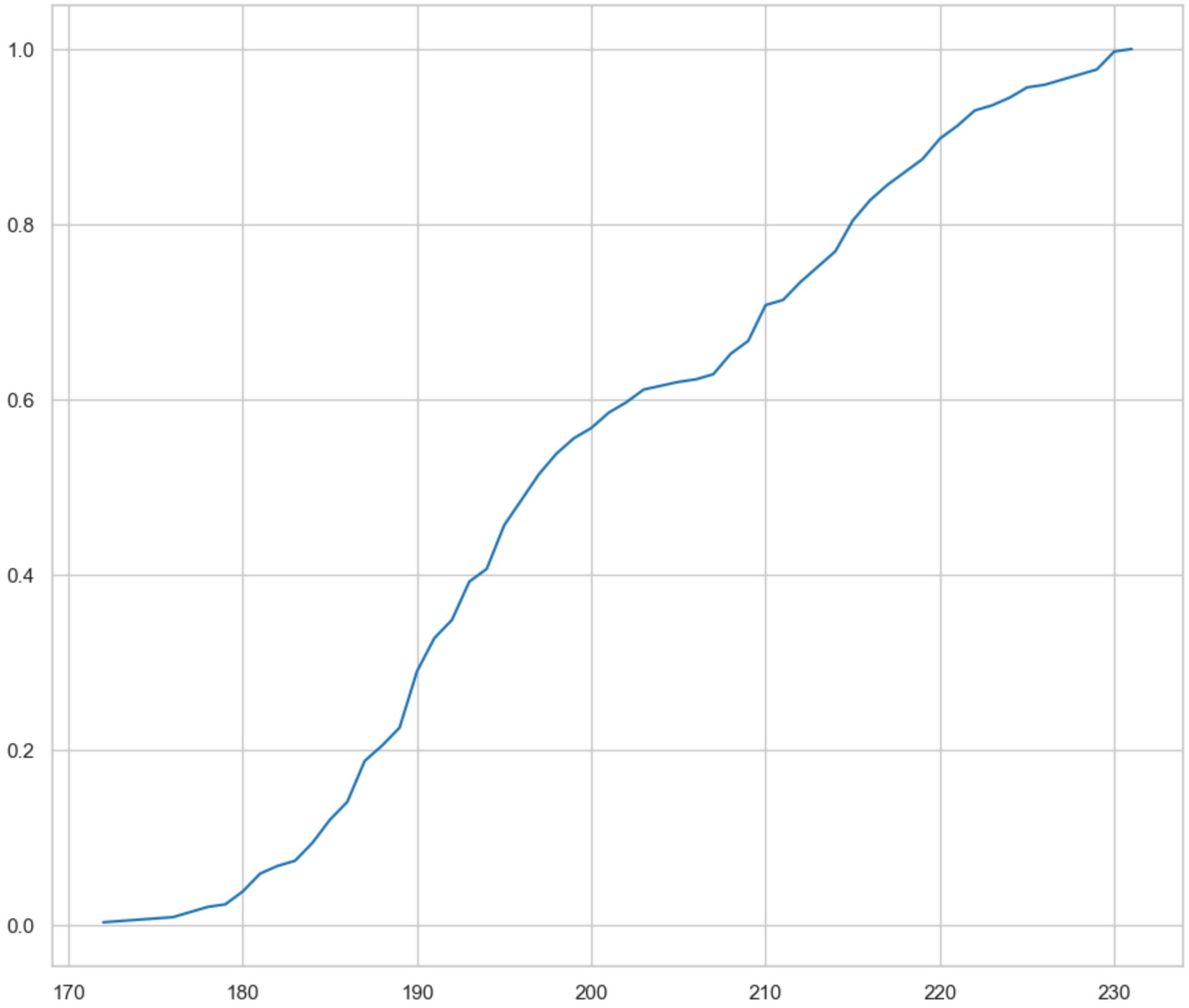
- ¿Cuál es la probabilidad de que tenga 200 mm de longitud de ala? Como se puede ver en el eje Y corta aproximadamente en 0.59

La desventaja de Seaborn, es que solo nos entrega la gráfica y no nos permite ser más detallado. Para ello necesitamos `empiricaldist`.

In [ ]:
```
#Guardamos el resultado en una variable
cdf_flipper_len_mm = empiricaldist.Cdf.from_seq(
    preprocessed_penguins_df.flipper_length_mm,
    normalize=True
)
```

In [ ]:
```
#Graficando
cdf_flipper_len_mm.plot()
```

Out[ ]:
```
<AxesSubplot: >
```



Como se puede observar, tenemos nuestra función de probabilidad acumulada y es suavizada, aunque se puede cambiar para que sea en forma escalonada.

Y si queremos encontrar un valor determinado hacemos lo siguiente:

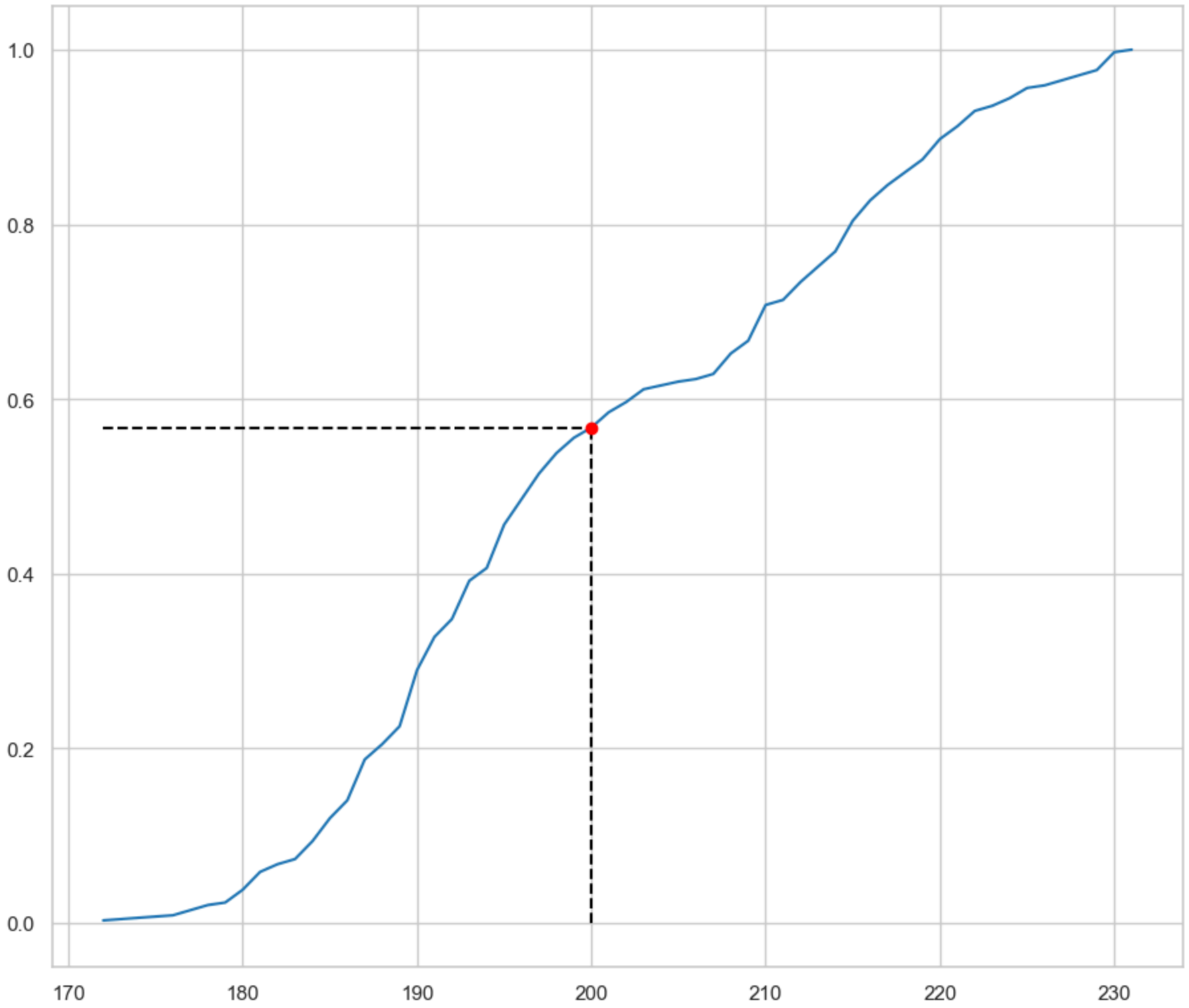
In [ ]:
```
#Encontrando probabilidad acumulada
#para el valor de 200
q=200
p =cdf_flipper_len_mm.forward(q)
p
```

Out[ ]:
```
array(0.56725146)
```

Aquí tenemos un valor más preciso y exacto. También podemos añadir guías para poder encontrar los valores.

In [ ]:
```
#Graficando
cdf_flipper_len_mm.plot()
#Colocando lineas
#vertical
plt.vlines(
    x=q,
    ymin=0,
    ymax=p,
    color='black',
    linestyles='dashed'
)
#horizontal
plt.hlines(
    y=p,
    xmin=pmf_flipper_len_mm.qs[0],
    xmax=q,
    color='black',
    linestyles='dashed'
)
#Punto rojo
plt.plot(q,p,'ro')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f1b2f5c5db0>]
```

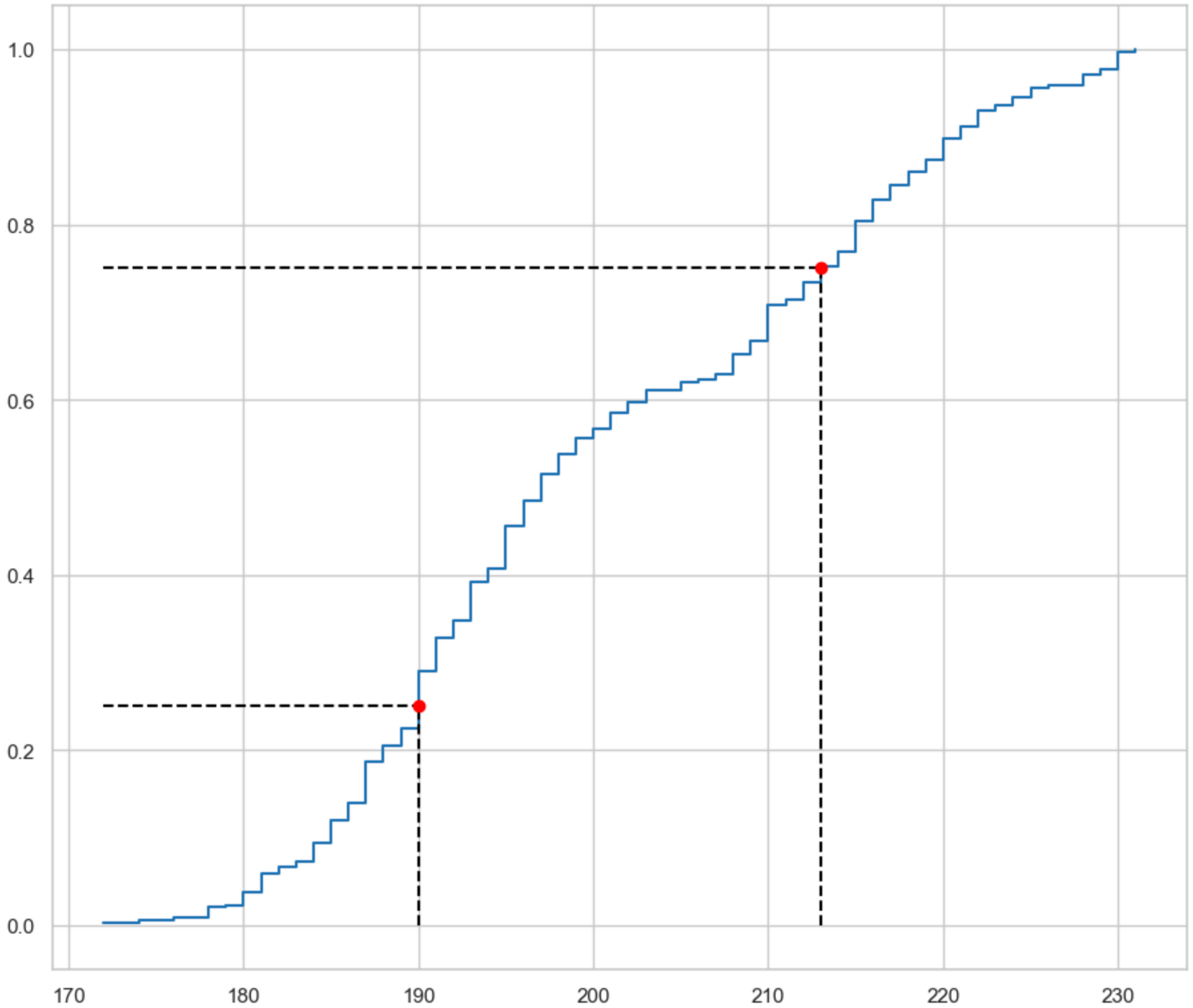


Esto nos ayuda a encontrar apoyándonos de guías visuales.

Ahora, exi|ste una relación estrecha entre los **cuantiles** y este tipo de gráfica

In [ ]:
```
cdf_flipper_len_mm.step()

p_1 = 0.25 # Specify probability
p_2 = 0.75

ps = (0.25, 0.75) # IQR
qs = cdf_flipper_len_mm.inverse(ps)

plt.vlines(
    x=qs,
    ymin=0,
    ymax=ps,
    color = 'black',
    linestyle='dashed'
)

plt.hlines(
    y=ps,
    xmin=pmf_flipper_len_mm.qs[0],
    xmax=qs,
    color='black',
    linestyle='dashed'
)

plt.scatter(
    x=qs,
    y=ps,
    color='red',
    zorder=2
)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f1b2f455510>
```



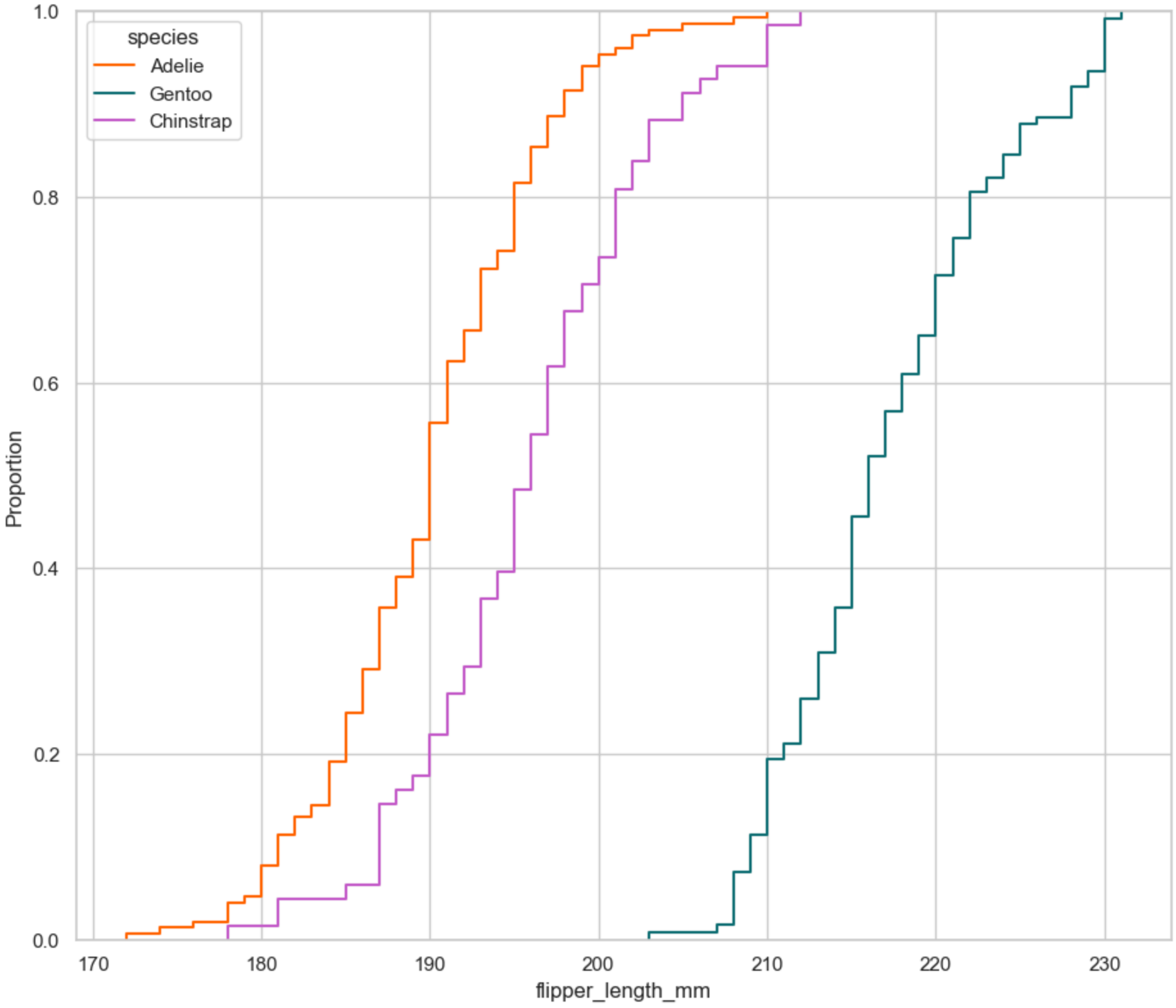
Ahora hay que poner atención en la función o parámetro `inverse`, ya que le damos una probabilidad y me calcula el valor inverso, en este caso el valor de la variable que tiene esa probabilidad.

Comparando distribuciones

Para este caso es mejor usar `Seaborn`, porque nos provee una manera sencilla de comparar distribuciones.

In [ ]:
```
sns.ecdfplot(
    data=preprocessed_penguins_df,
    x='flipper_length_mm',
    hue='species',
    palette=penguin_color
)
```

Out[ ]:
```
<AxesSubplot: xlabel='flipper_length_mm', ylabel='Proportion'>
```



Ahora yo puedo ver la distribución acumulada de probabilidad por cada una de las especies de pingüinos.

Ya se que los Gentoo, tienen aletas mucho más grande que los **Adelie y Chinstrap**.