Estadística descriptiva aplicada: funciones de densidad de probabilidad.

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

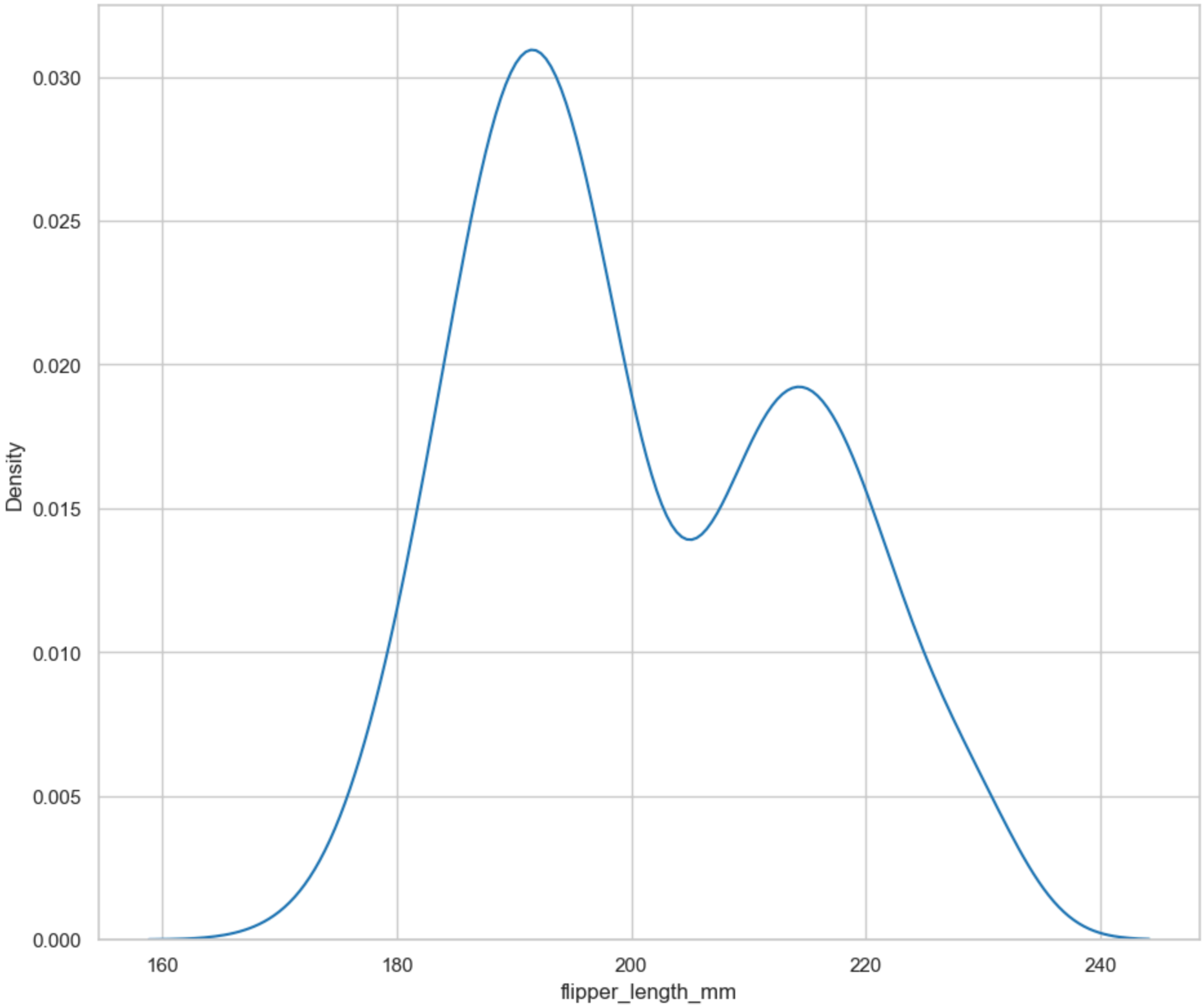
Datos Preprocesados

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

Función de densidad de probabilidad

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

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



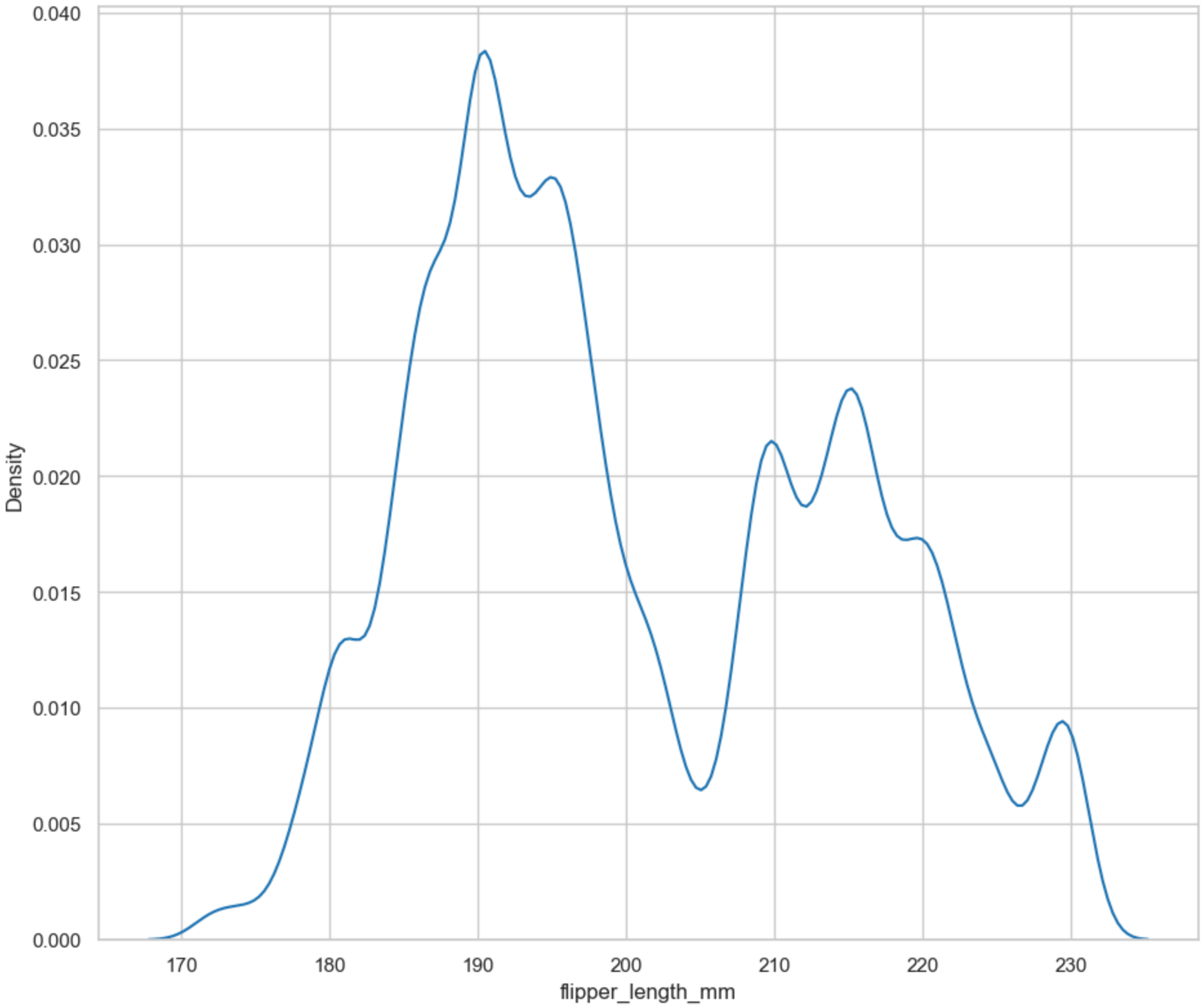
En el eje X tenemos la longitud de las aletas del pingüino y en el eje Y, tenemos **Densidad** que van entre 0 y 0.30, lo que obtenemos son las probabilidades de que obtengamos ciertos valores en aleta del pingüino, algo similar PMF pero de forma continua.

Recordemos que teníamos zonas en la cual algunos valores no existían, en este caso la **Densidad de probabilidad** nos ayudará a hacer estimaciones para encontrar la probabilidad entre un valor y el siguiente, es decir de una manera continua.

A veces tiene una gran aplicación para ver la forma de la distribución, pero no como interpretación. Al igual que el **Histograma**, nosotros podemos modificar el *bandwidth* y ver como se puede ajustar, es decir vamos a obtener curvas de manera diferente dependiendo de que tan bien las ajustemos.

In [ ]:
```
sns.kdeplot(
    data=preprocessed_penguins_df,
    x='flipper_length_mm',
    bw_method=0.1
)
```

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



Ahora tenemos un comportamiento diferente y como se puede ver obtenemos una curva menos suavizada, y esto es debido al ajuste del ancho de banda.

Este tipo de observaciones no las hubiéramos podido encontrar sino se hubiera ajustado el parámetro `bw_method=0.1` .

Entonces como consejo, hay que explorar en detalle los parámetros de la gráfica de densidad de probabilidad y del histograma.

Lo interesante de este tipo de gráfica es que la podemos usar para comparar otro tipo de distribuciones.

In [ ]:
```
#guardando estadísticos
stats = preprocessed_penguins_df.body_mass_g.describe()
stats
```

Out[ ]:
```
count     342.000000
mean     4201.754386
std       801.954536
min      2700.000000
25%      3550.000000
50%      4050.000000
75%      4750.000000
max      6300.000000
Name: body_mass_g, dtype: float64
```

Ahora podemos empezar a generar datos aleatorios y compararlos con los valores que tenga nuestra variable.

In [ ]:
```
#Muestras
xsam=np.linspace(stats['min'],stats['max'])
```

Out[ ]:
```
array([2700.        , 2773.46938776, 2846.93877551, 2920.40816327,
       2993.87755102, 3067.34693878, 3140.81632653, 3214.28571429,
       3287.75510204, 3361.2244898 , 3434.69387755, 3508.16326531,
       3581.63265306, 3655.10204082, 3728.57142857, 3802.04081633,
       3875.51020408, 3948.97959184, 4022.44897959, 4095.91836735,
       4169.3877551 , 4242.85714286, 4316.32653061, 4389.79591837,
       4463.26530612, 4536.73469388, 4610.20408163, 4683.67346939,
       4757.14285714, 4830.6122449 , 4904.08163265, 4977.55102041,
       5051.02040816, 5124.48979592, 5197.95918367, 5271.42857143,
       5344.89795918, 5418.36734694, 5491.83673469, 5565.30612245,
       5638.7755102 , 5712.24489796, 5785.71428571, 5859.18367347,
       5932.65306122, 6006.12244898, 6079.59183673, 6153.06122449,
       6226.53061224, 6300.        ])
```

Vamos a comparar con distribución normal, ya sabemos que los datos no siguen ese tipo de distribución. Pero vamos a generarla.

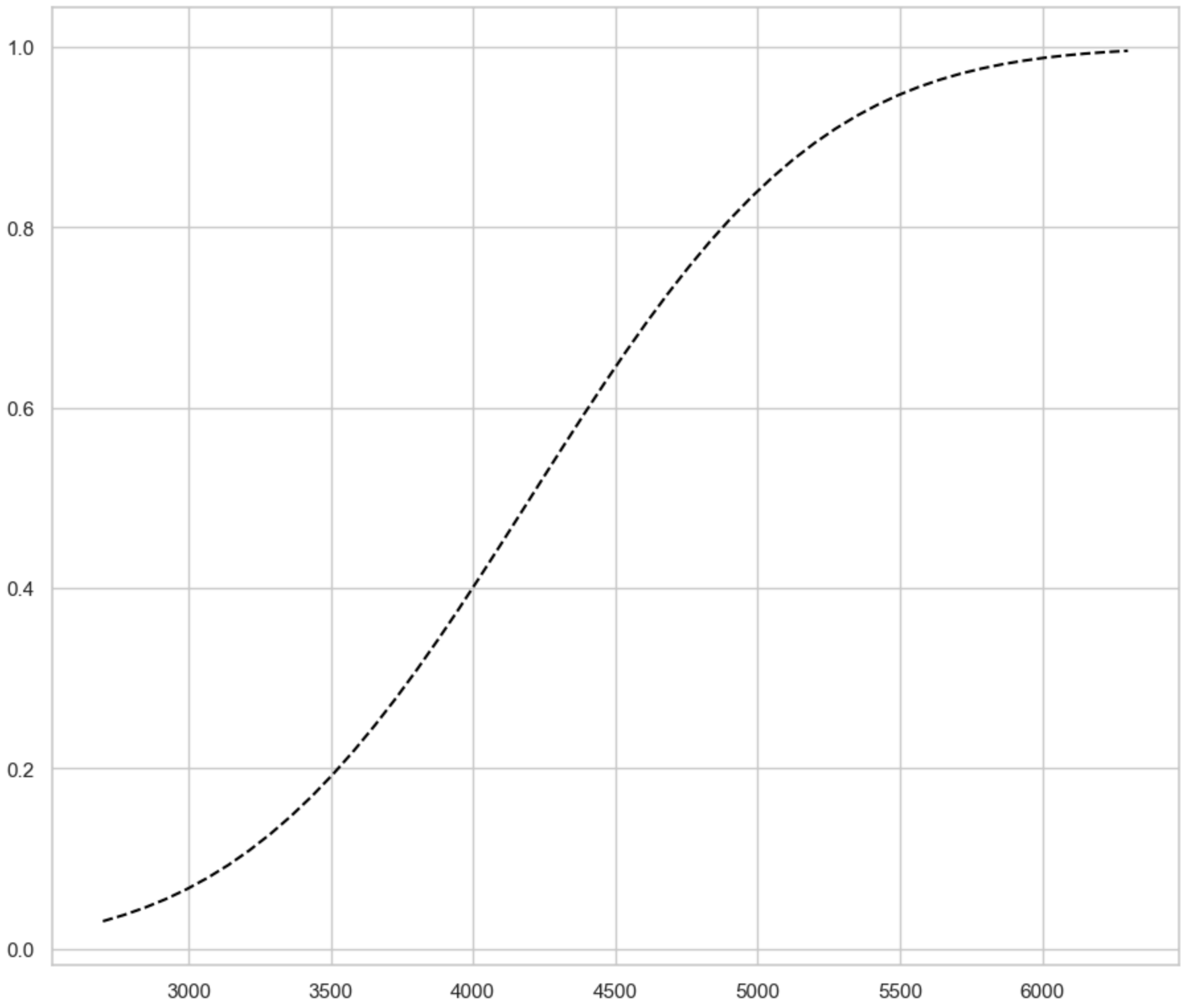
In [ ]:
```
y_sam=scipy.stats.norm(stats['mean'],stats['std']).cdf(xsam)
y_sam
```

Out[ ]:
```
array([0.03056059, 0.03745582, 0.04557216, 0.05504607, 0.06601225,
       0.07859975, 0.0929276 , 0.1091002 , 0.12720261, 0.14729588,
       0.16941268, 0.19355346, 0.21968339, 0.24773013, 0.27758288,
       0.30909253, 0.34207329, 0.37630551, 0.41153992, 0.44750299,
       0.48390319, 0.52043813, 0.55680205, 0.59269345, 0.62782261,
       0.66191854, 0.69473525, 0.72605692, 0.75570199, 0.78352586,
       0.8094223 , 0.83332353, 0.85519902, 0.87505326, 0.89292249,
       0.90887086, 0.92298596, 0.9353742 , 0.94615603, 0.95546139,
       0.96342539, 0.97018445, 0.97587298, 0.98062054, 0.98454969,
       0.98777435, 0.99039874, 0.99251675, 0.99421182, 0.99555707])
```

Con `cdf()` lo que le indicamos es que le estamos pasando unos valores de x que generamos. Ahora si me dará una probabilidad acumulada

In [ ]:
```
plt.plot(xsam,y_sam,color='black',linestyle='dashed')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f40e82c7430>]
```

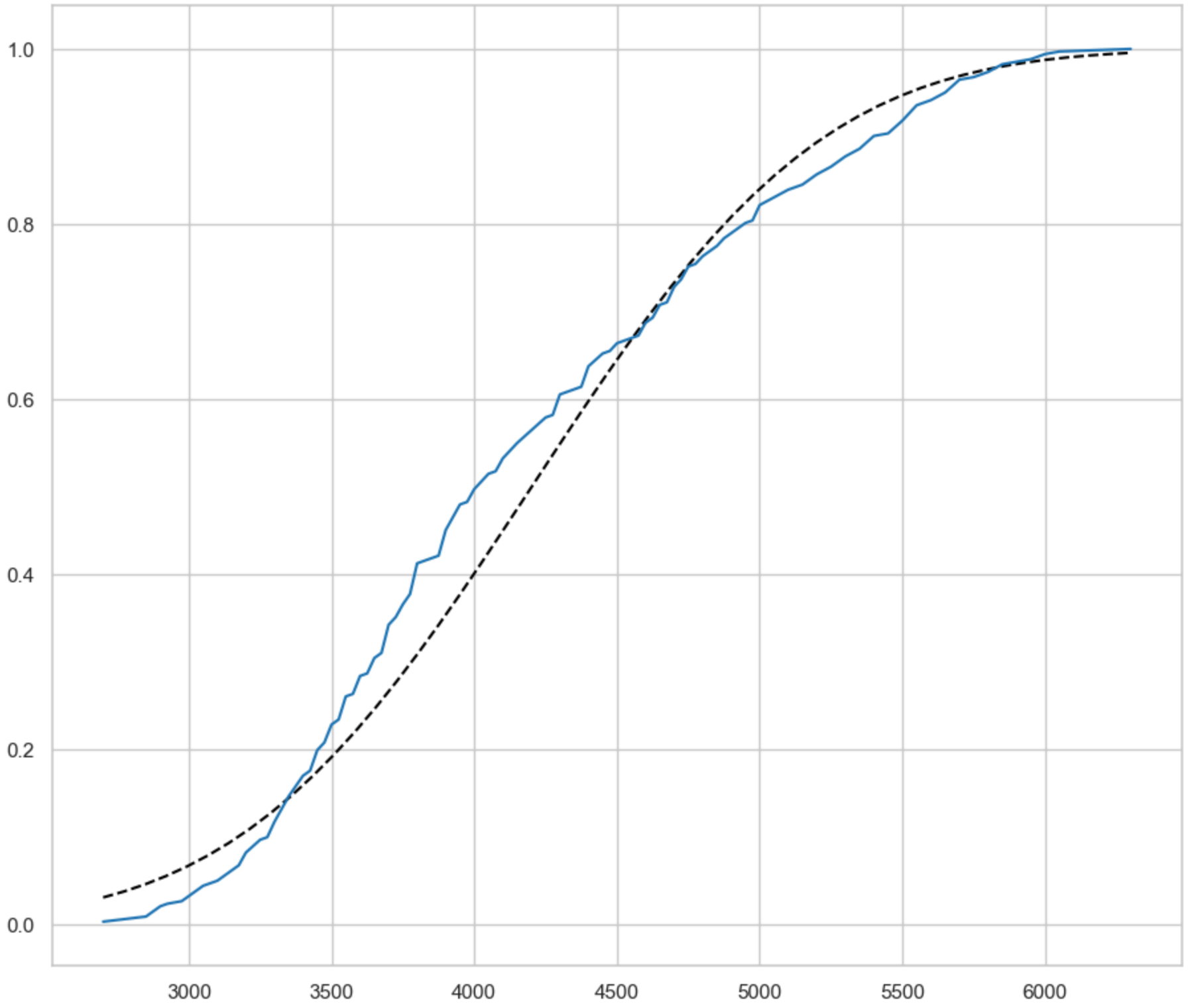


Aquí ya tenemos como lucirían nuestros datos, dado que fueran una distribución normal. Nosotros sabemos que no siguen una distribución normal, pero para no andar con suposiciones, lo comprobaremos:

In [ ]:
```
plt.plot(xsam,y_sam,color='black',linestyle='dashed')

empiricaldist.Cdf.from_seq(
    preprocessed_penguins_df.body_mass_g,
    normalize=True
).plot()
```

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

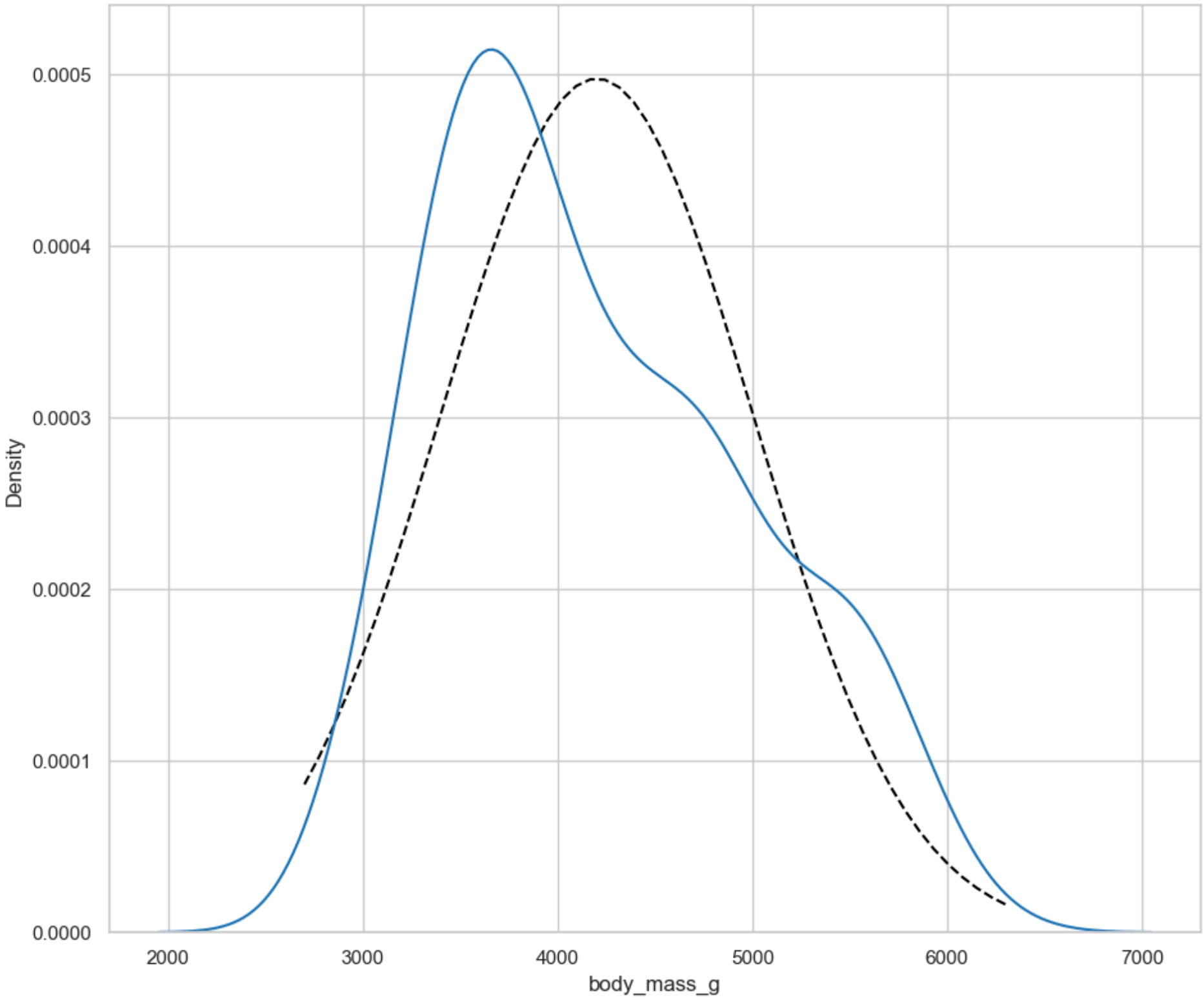


Aquí tenemos una comparación de lo que es la distribución teórica y la real, La distribución teórica esta en negro y la distribución real en azul. Si vemos que puede existir algunos valores en los cuales puede llegar a parecerse, hay otros en los que no. Entonces nuestros datos no se comportan de una manera normal. Esto es un caso de uso para las funciones acumuladas de probabilidad, pero podemos usar las funciones de **Densidad de probabilidad**.

In [ ]:
```
#Calculando la PDF
y_sam=scipy.stats.norm(stats['mean'],stats['std']).pdf(xsam)
#Graficandola
plt.plot(xsam,y_sam,color='black',linestyle='dashed')

sns.kdeplot(
    data=preprocessed_penguins_df,
    x='body_mass_g',
)
```

Out[ ]:
```
<AxesSubplot: xlabel='body_mass_g', ylabel='Density'>
```



Fijémonos como tenemos otra forma, la distribución teórica es la negra y la real (datos) es la azul.

Ahora podemos ver que no se parecen nada. Nuestra distribución real está sesgada a la izquierda, es decir nos dan una idea de que nuestros datos no siguen una distribución normal.

El hecho de que no sigan una distribución normal es bastante importante, porque en el mundo de los datos, muchos de los estadísticos, pruebas, análisis; van a asumir muchas cosas, entre ellas la de "Los datos van a seguir una distribución normal", tanto los datos como los residuos después de que ajustas un modelo.

Ya verificamos y estimamos los datos reales contra ciertas referencias normales, y efectivamente nos dimos cuenta que no se comportan de manera normal.