**DOCUMENTACIÓN GITHUB HATEMEDIA**

**Importación y uso del modelo para realizar predicciones:**

Los diferentes modelos empleados tienen como fin analizar datos extraídos de diferentes redes sociales como por ejemplo Twitter o Facebook de la cual se extraen los comentarios referentes a los periódicos (La Vanguardia, El Mundo, ABC, El País y 20 Minutos) y páginas web de los mismos para su posterior clasificación en función de su tipo de odio.

Para llevar a cabo esta clasificación, se ha realizado el ajuste fino de [Robertuito](https://huggingface.co/pysentimiento/robertuito-sentiment-analysis), un modelo derivado de [RoBERTa](https://huggingface.co/FacebookAI/roberta-base) y entrenando con un corpus en español de alrededor de 5.000 tweets.

Para el uso del modelo de predicción se adjunta el script Python “ejemplo.py”.

Antes de ejecutar estos scripts es necesario instalar las distintas librerías importadas en ellos.

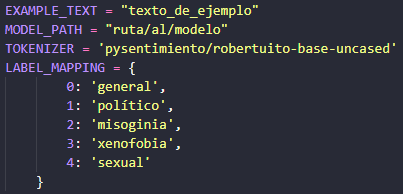
**EJEMPLO.PY:**

Se detalla a continuación mediante el código de este script el procedimiento a seguir para realizar las predicciones.

Realizamos los siguientes imports::

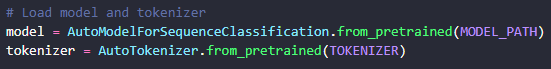


A continuación tenemos las siguientes variables:

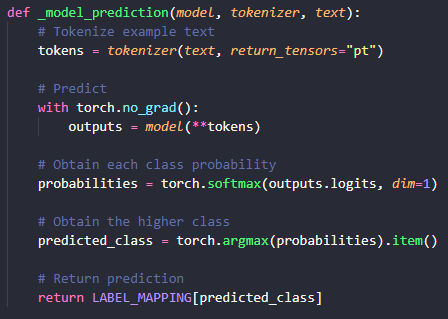


* **EXAMPLE\_TEXT:** contiene el texto sobre el que queremos realizar la predicción.
* **MODEL\_PATH:** contiene la ruta al modelo.
* **TOKENIZER:** tokenizador de Robertuito disponible en [Hugging Face](https://huggingface.co/).
* **LABEL\_MAPPING:** se utiliza para presentar el tipo de odio por pantalla.

Cargamos el modelo y el tokenizer:



Llamamos a la función que utiliza el modelo para realizar la predicción:

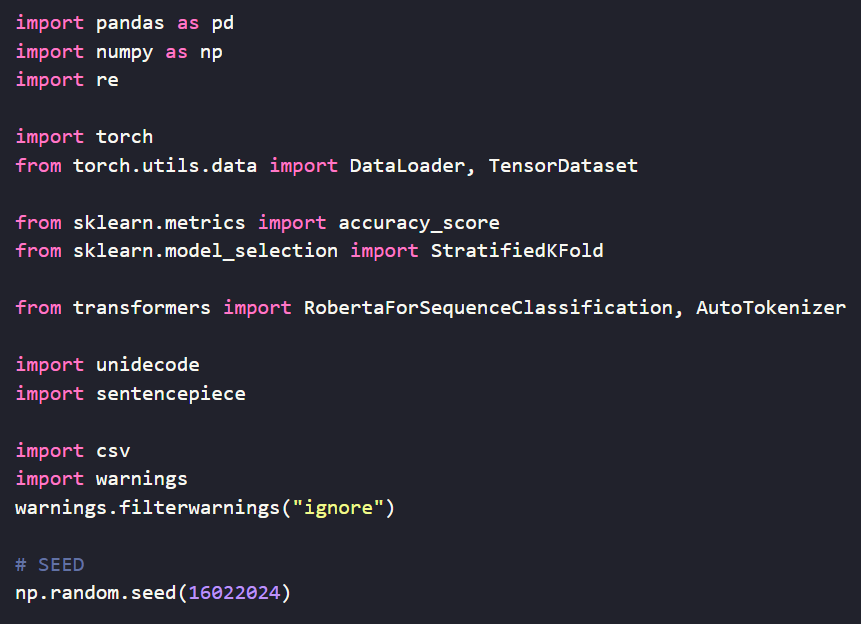




Y mostramos el resultado por pantalla:

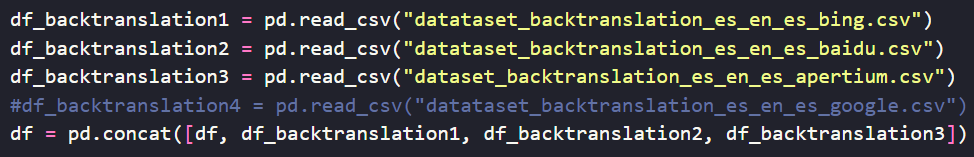


**Uso del código de entrenamiento:**

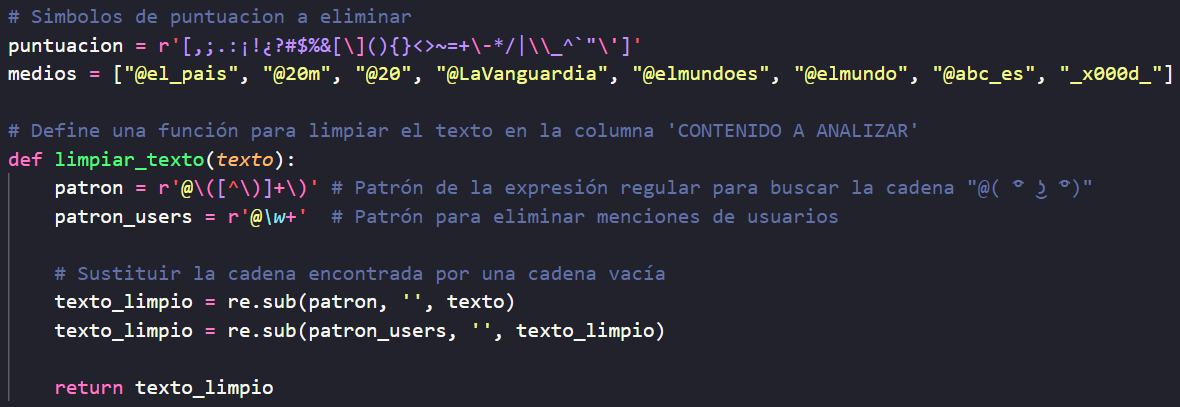
Importamos las librerías necesarias:  


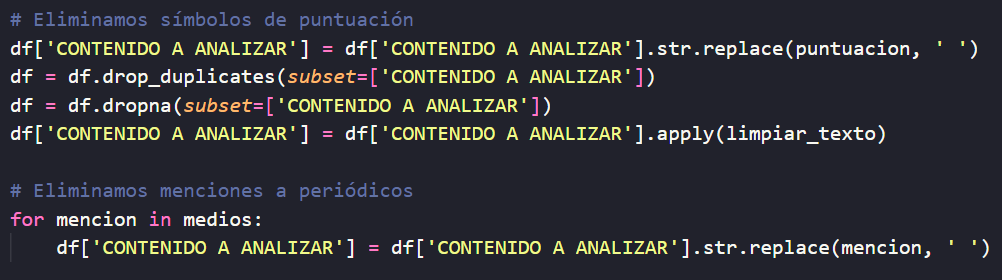
Leemos el dataset de entrenamiento a un dataframe de pandas:  


Leemos también los datasets con las entradas generadas mediante backtranslation para intentar balancear el dataset y los concatenamos al que habíamos leído anteriormente para conformar el dataset final:

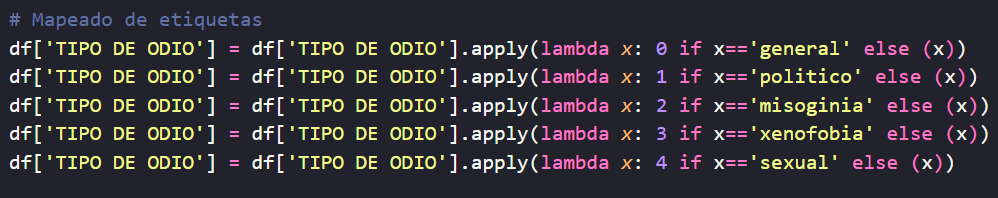


Aplicamos funciones de limpieza al texto:

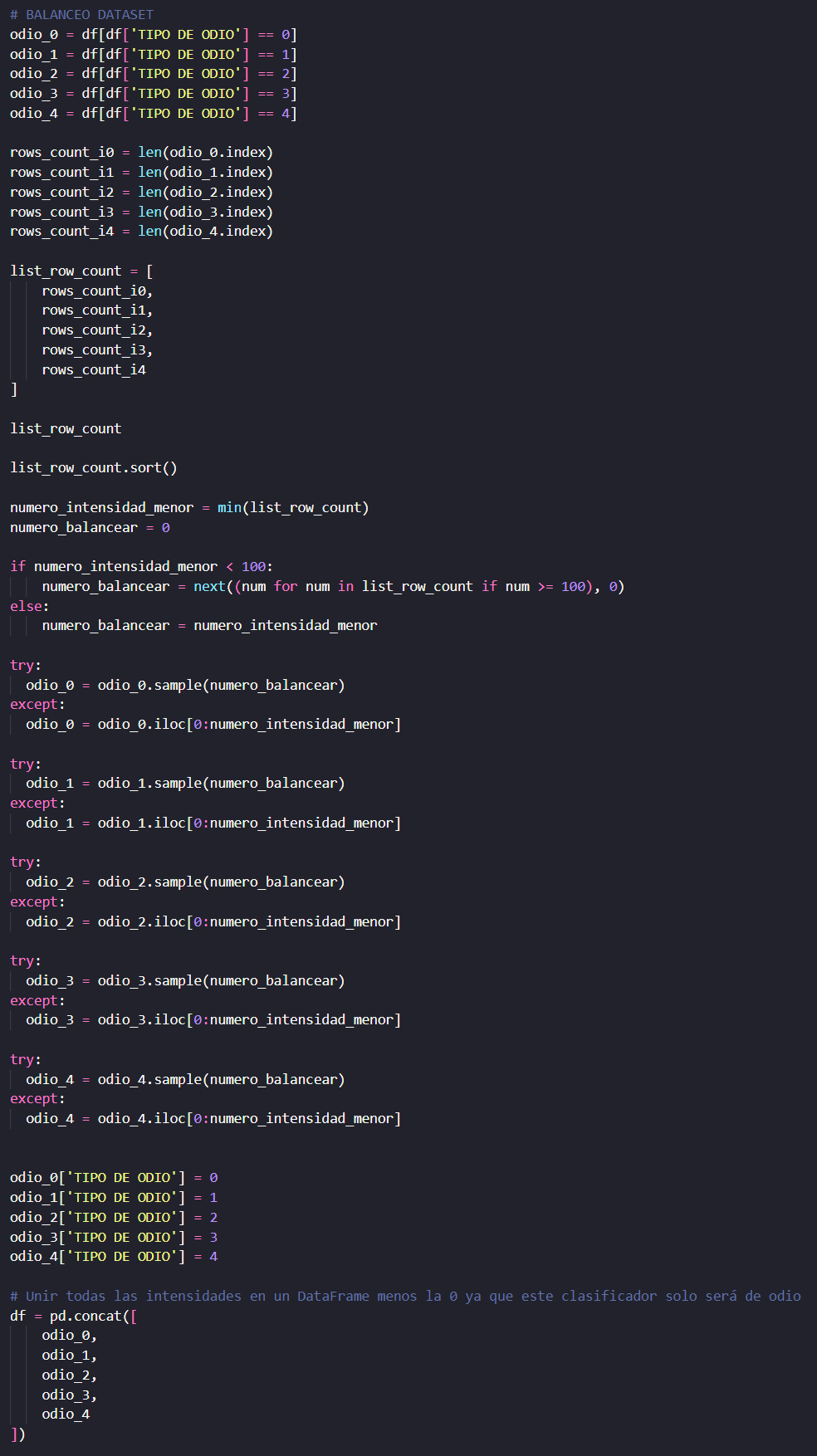




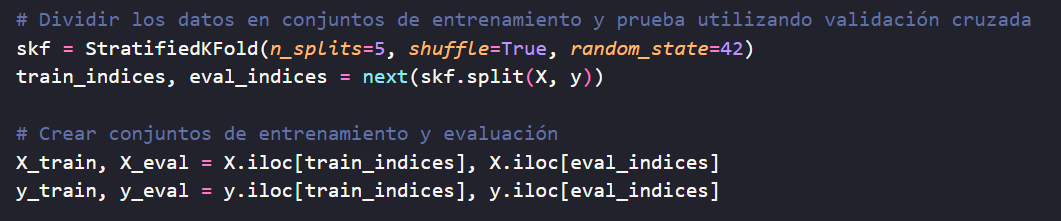
Mapeamos las etiquetas a valores numéricos:



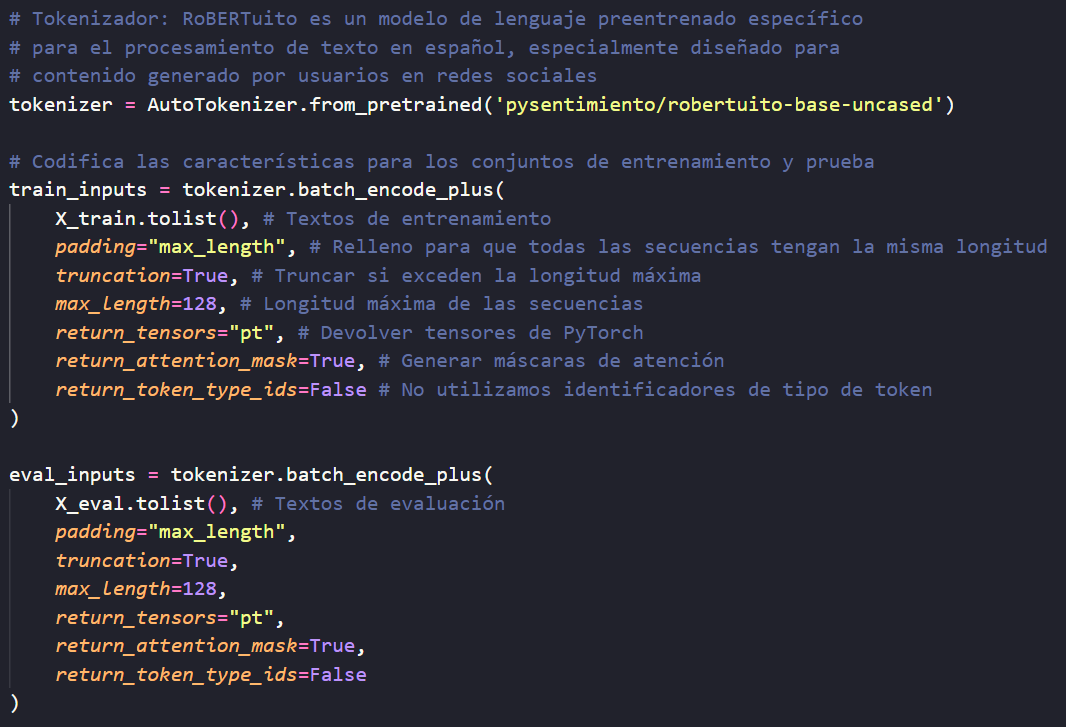
Como seguimos teniendo las clases desbalanceadas, nos quedamos con el mismo número de entradas que tenga la clase menos poblada:

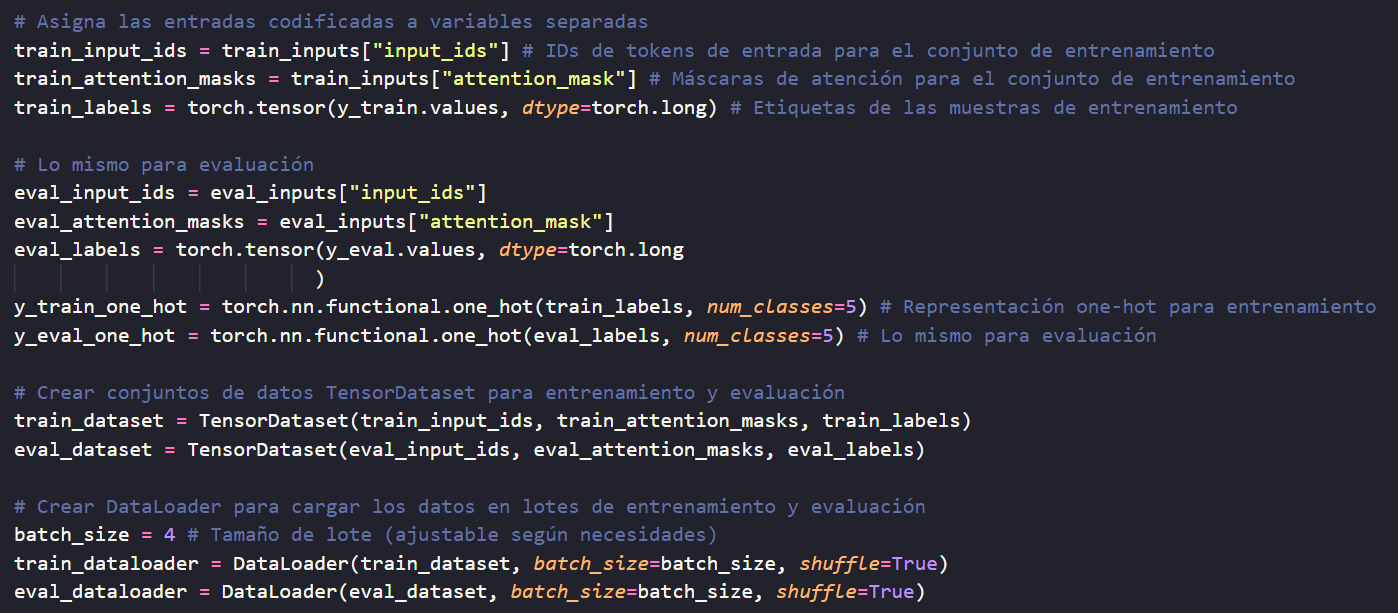


Dividimos los datos en el conjunto de entrenamiento y validación:

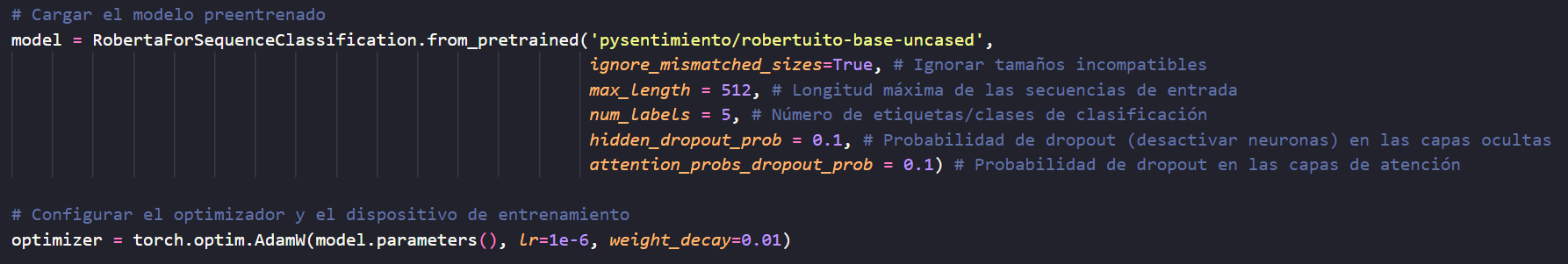


Inicializamos el tokenizador ya preentrenado de *Robertuito* y tokenizamos los cuerpos de entrenamiento y validación:

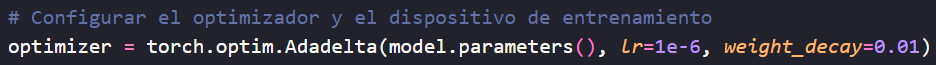


Preparamos los datos de entrada para el entrenamiento y validación del modelo:  


Inicializamos el modelo Robertuito y el optimizador:

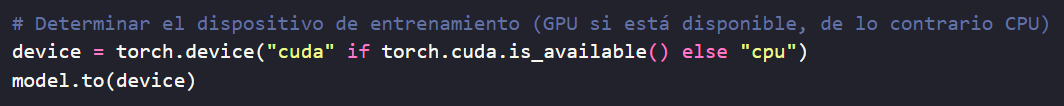


Con respecto al optimizador, se han realizado pruebas con AdamW, SGD, ASGD y Adadelta. Para utilizar cualquiera de los optimizadores mencionados solo hay que sustituir el nombre en el código de la siguiente forma (por ejemplo para utilizar Adadelta):

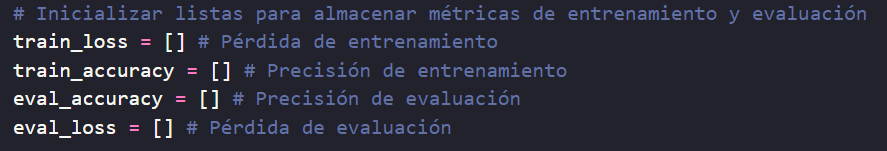


En caso de especificar un *weight\_decay* distinto de cero, estaremos aplicando regularización L2.

Con las siguientes líneas de código podremos hacer uso de la GPU en caso de que nuestro equipo cuente con ella:

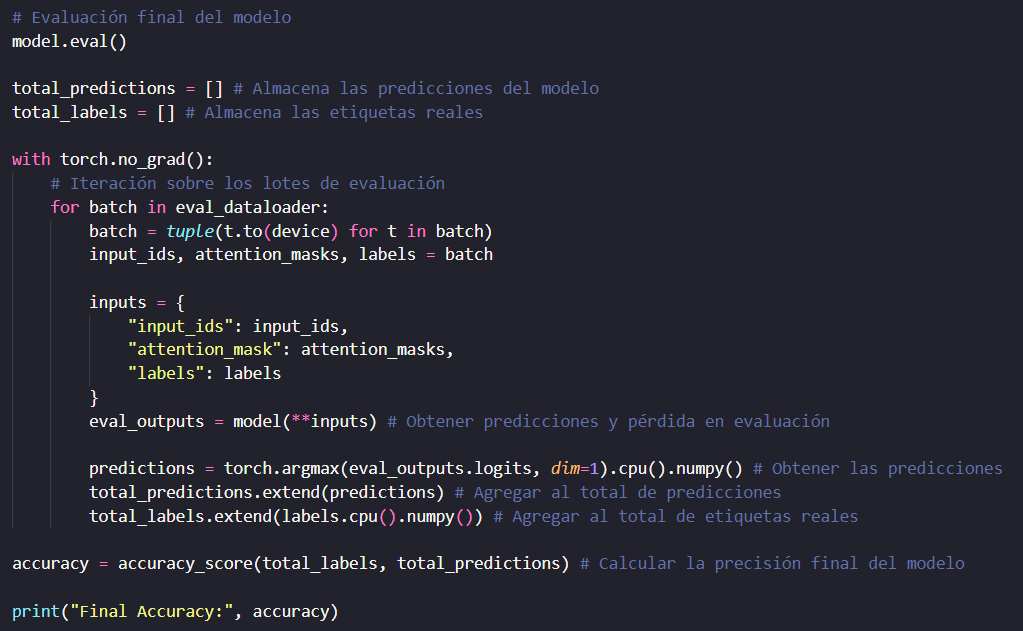


Inicializamos las variables donde almacenaremos la evolución del *accuracy* y *loss* durante el entrenamiento y la validación:



Entrenamiento y validación del modelo:

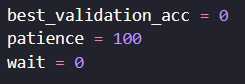




Podemos especificar un número distinto de épocas durantes las que realizar el entrenamiento y validación modificando la variable *num\_epochs*:

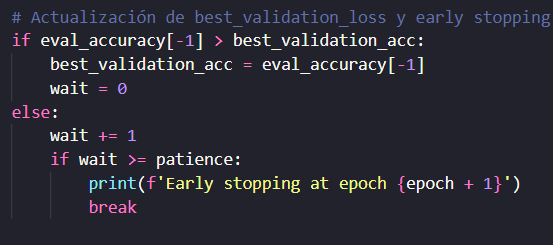


A continuación se muestra la implementación de un mecanismo de *early stopping* para que en caso de que el *accuracy* en validación no aumente en un número de épocas específico se detenga el entrenamiento. Para ello, comenzamos inicializando las siguientes variables:

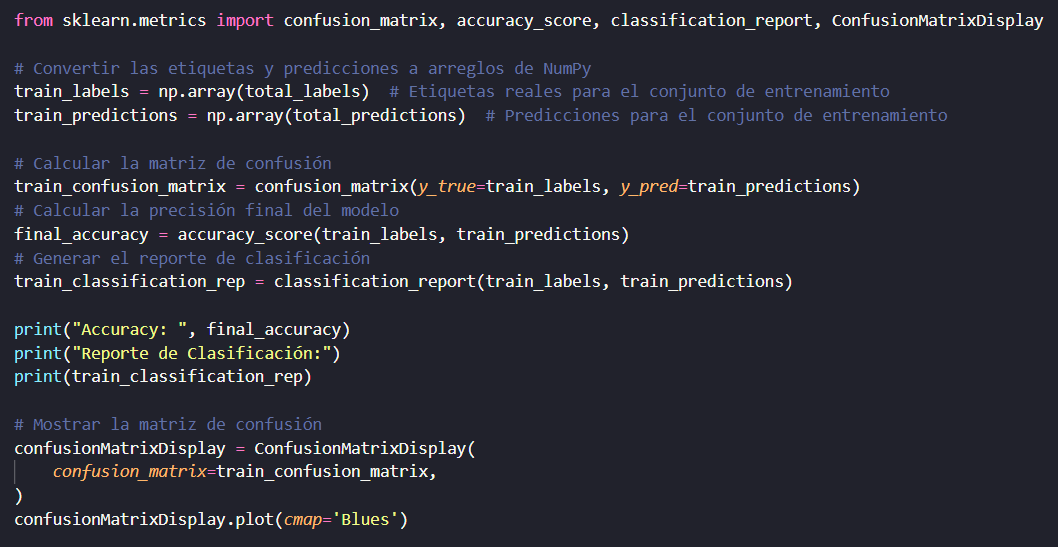
****

Siendo ***patience*** el número máximo de épocas en las que esperar a que se mejore el *accuracy* en validación.

Añadimos el mecanismo de *early stopping* tras la validación realizada en cada época:



Visualización del reporte de clasificación y matriz de confusión:



Podemos visualizar graficadas las evoluciones del *accuracy* y *loss* del modelo durante las épocas de entrenamiento y validación con:

