En esta práctica se ha tratado de predecir mediante regresión de la forma mejor aproximada posible el precio de los apartamentos del dataset de airbnb.

Tenemos dos fuentes de información:
- Numérica/categórica
- Imágenes

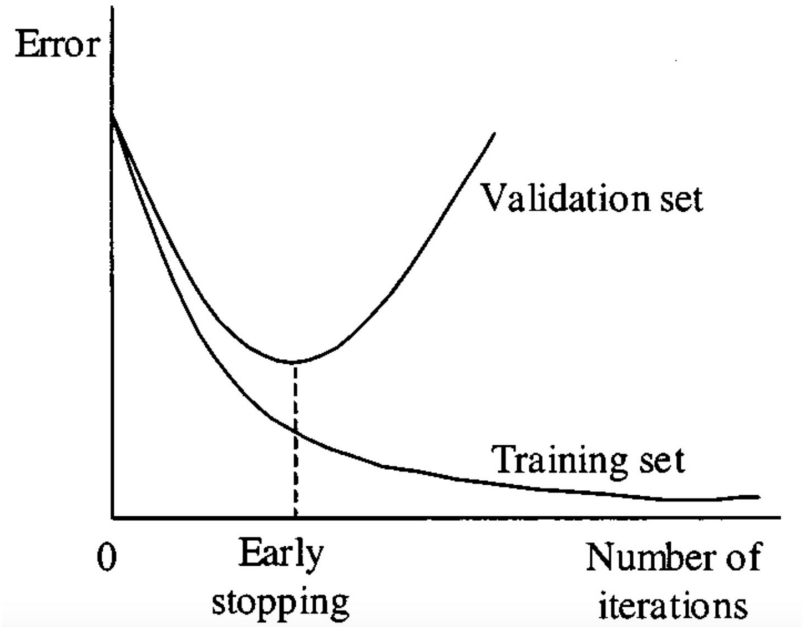
Por lo que tratamos los datos en dos ramas diferentes, guardando los modelos entrenados, para finalmente unirlos en el notebook "*final_project*".

Dentro de la rama: *numerical_branch* tenemos:

- **numerical_data_processing.ipynb**
  En este notebook se muestra todo el análisis de datos de la parte numérica, basándonos en el dataset de training.

- **MLP_model.ipynb**
  Aquí se trabaja con la data numérica (y se guarda para usar en la parte de imágenes las mismas instancias cogiendo los índices) para crear el modelo que nos predecirá los precios.
  La arquitectura que sigue es de 2 hidden layers con 64 y 32 neuronas respectivamente, y finalmente la output layer con una sola neurona y activación "linear" ya que se trata de una regresión.
  Además a las hidden layers se les aplica Dropout (para evitar overfitting) y BatchNormalization.

  Al hacer el fit del modelo usamos *EarlyStopping(monitor='val_loss')*.



  La intención de usar esto es evitar de nuevo el overfitting.

Tenemos también la rama: *images_branch*:

- **load_images.ipynb**
  Aquí tenemos el código que lee el url de la imagen en el dataset y busca la imagen y la convierte a matrix.

  Se puede encontrar el archivo .npy en el siguiente enlace del drive, ya que pesa demasiado como para subirlo al GitHub:
  https://drive.google.com/file/d/1F7BvHDE6Sb9WwIB3owT-q6or2In3PoSZ/view?usp=sharing

- **CNN_model.ipynb**
  En este caso hemos cogido los dataframes: *y_train*, *y_test.* Hemos cogido sus índices y buscado las imágenes con las que se corresponden, para que más adelante, fusionar las dos ramas para obtener un único valor predicho, tuviera coherencia.

  En este caso la arquitectura que se ha seguido es:

  - 3 capas convolucionales, con 16, 32 y 64 neuronas respectivamente. Añadiendo también BatchNormalization y MaxPooling2D.
  - Flatten(), para poder aplicar la capa FullyConnected
  - 1 capa dense de 16 neuronas. Con BatchNormalization y Dropuout.
  - Finalmente, la output layer con 1 neurona y activacion linear, ya que estamos en el caso de regresión.

  En este caso también hemos usado el earlyStopping para evitar el overfitting.

Finalmente, tenemos el main: **final_project.ipynb**

Aquí cargamos todos los datos y los dos modelos (mlp y cnn) y concatenamos sus outputs para tener una modelo que recibe tanto los datos numéricos como las imagenes para predecir el precio de un apartamento.

****NOTA:**

Inicialmente traté el problema desde un punto de vista de “clasificación”. He dejado lo que utilizaba comentado tanto en el notebook de MLP_model como en CNN_model.

También está incluido el **utils.py** donde está la función que limpia los datos. Aquí he dejado comentado los pasos que hacía para tener la data lista para un problema de clasificación. Sin embargo, los resultados eran bastante malos: en la parte numérica, la accuracy era más baja que usando un Random Forest de Machine Learning, y en la parte de imágenes la accuracy era de 50% (era un modelo totalmente aleatorio).

Entiendo que hay algún paso del modelo o del procesamiento de datos que me he dejado sin darme cuenta, por eso lo dejo comentado, por si se pudiera echar un vistazo y poder mejorarlo como next steps.