Rubén Lázaro Jiménez

Víctor Angulo De Castro

Andrés Flores Giménez

DIAGNÓSTICO DE ENFERMEDADES

Índice

1. INTRODUCCIÓN

La inteligencia artificial ha revolucionado el campo de la medicina, especialmente en la predicción de enfermedades. Gracias a su capacidad para analizar grandes volúmenes de datos clínicos, genéticos e incluso imágenes médicas, la IA permite identificar patrones que pueden anticipar el desarrollo de diversas afecciones antes de que se manifiesten clínicamente, lo que es esencial para salvar vidas, como es en el caso del cáncer de tiroides y las distintas enfermedades en la piel.

Para hacerlo posible, se han recopilado datos clínicos de distintos pacientes de todo el mundo. Esto nos permite encontrar patrones en personas de distintos países y etnias para poder sacar conclusiones basadas en resultados.

2. BUSINESS UNDERSTANDING

2.1 OBJETIVOS

A. OBJETIVO GENERAL

Predecir si un paciente tiene cáncer o no mediante varios modelos de aprendizaje automático basados en datos obtenidos a través de datasets públicos y repositorios.
Habrá dos tipos de datos: dataset e imágenes.

- Dataset: información clínica de pacientes.
- Imágenes: imágenes de enfermedades cutáneas.

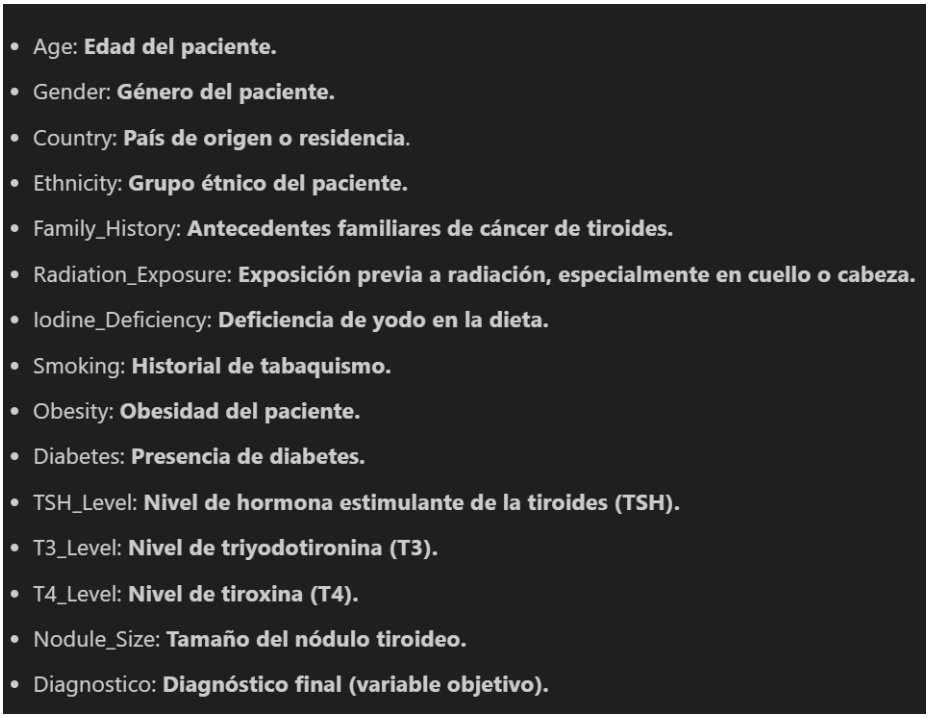
B. OBJETIVOS ESPECÍFICOS

- Extraer información relevante de los datos clínicos que ayuden a la predicción de la enfermedad.
- Estandarizar y depurar los datos recopilados para asegurar su calidad y consistencia, preparándolos para el análisis posterior.
- Diseñar y entrenar varios modelos analíticos o de aprendizaje automático capaz de predecir con alta precisión, validar y optimizar los modelos desarrollados mediante técnicas de evaluación, asegurando su fiabilidad en escenarios reales.

3. DATA UNDERSTANDING

Proporciona la base para tomar decisiones informadas sobre cómo procesar, transformar y analizar los datos en las etapas posteriores del proyecto. Una comprensión sólida de los datos asegura un análisis más efectivo y resultados de mayor calidad.

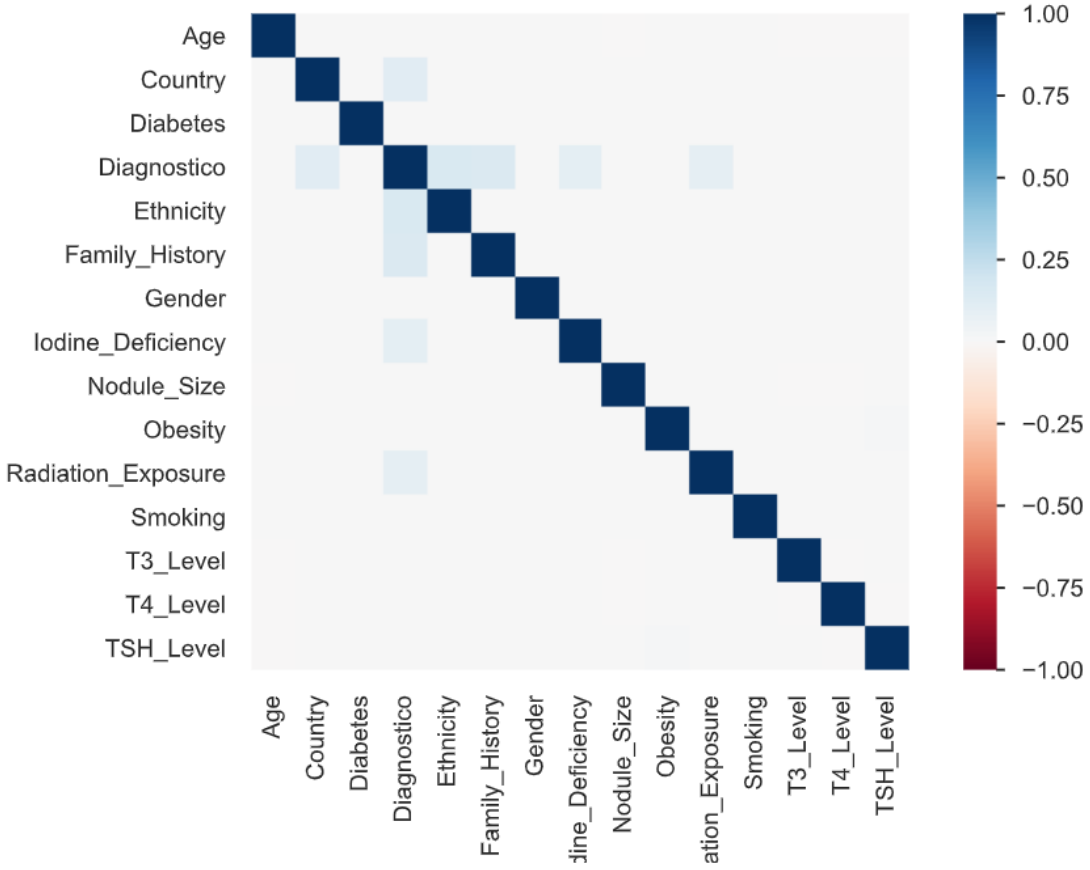
En este proyecto, se han recopilado datos de una página web, Kaggle. En el caso de los datos de tiroides se ha conseguido un dataset de 212.691 registros con la siguiente información:



Son en total 15 columnas de las cuales, 9 son categóricas y 6 son numéricas además de la columna objetivo.

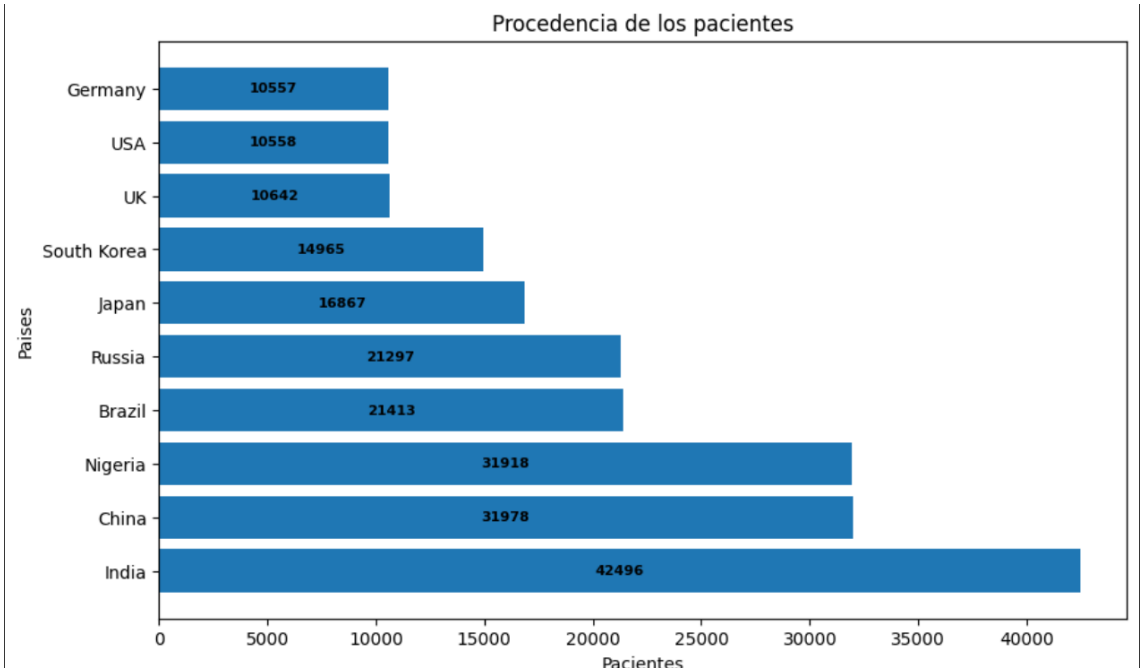
Antes de mostrar gráficas comprobamos que los datos estaban correctamente (sin nulos ni outliers) mediante métodos de pandas con **isna()** o **describe()** para ver las principales estadísticas de las columnas numéricas. Además, con ayuda de la librería **ydata_profiling** podemos ver por cada columna si hay nulos, cuantos registros de cada columna son distintos, es decir, que no se repiten, también te dice en caso de las numéricas el número más alto y el más pequeño para ver outliers y por último nos muestra un mapa de calor de cómo se relacionan entre sí las

variables numéricas:

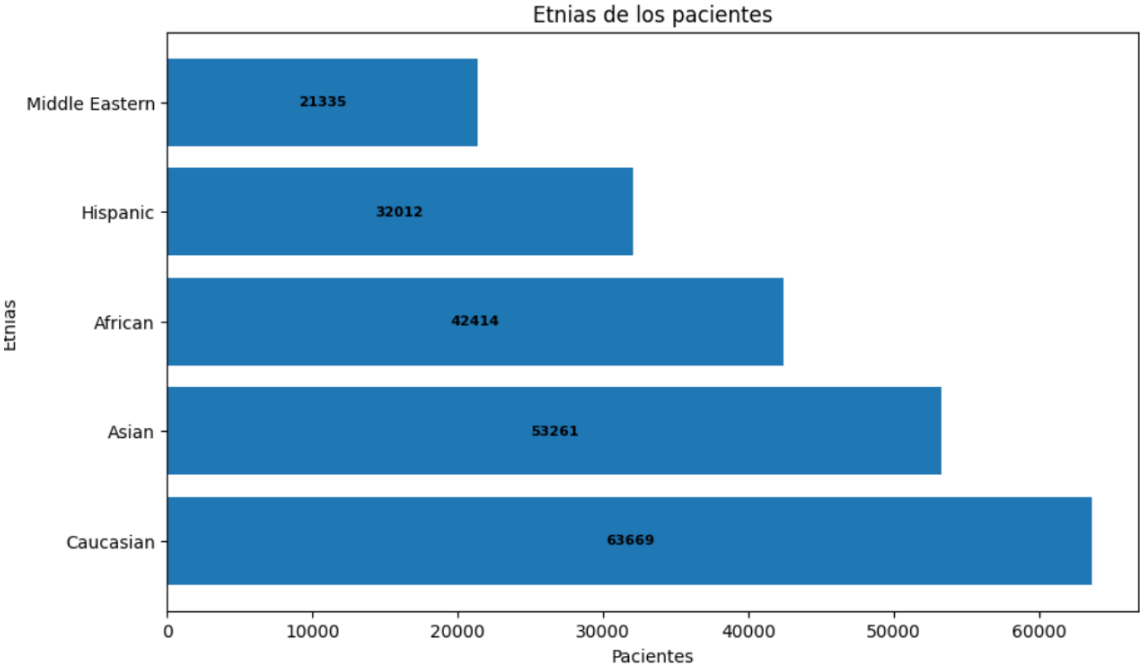


Se observa que las variables más relacionadas con diagnóstico son **ethnicity, family_history, Iodine_Deficiency y radiation_exposure**.

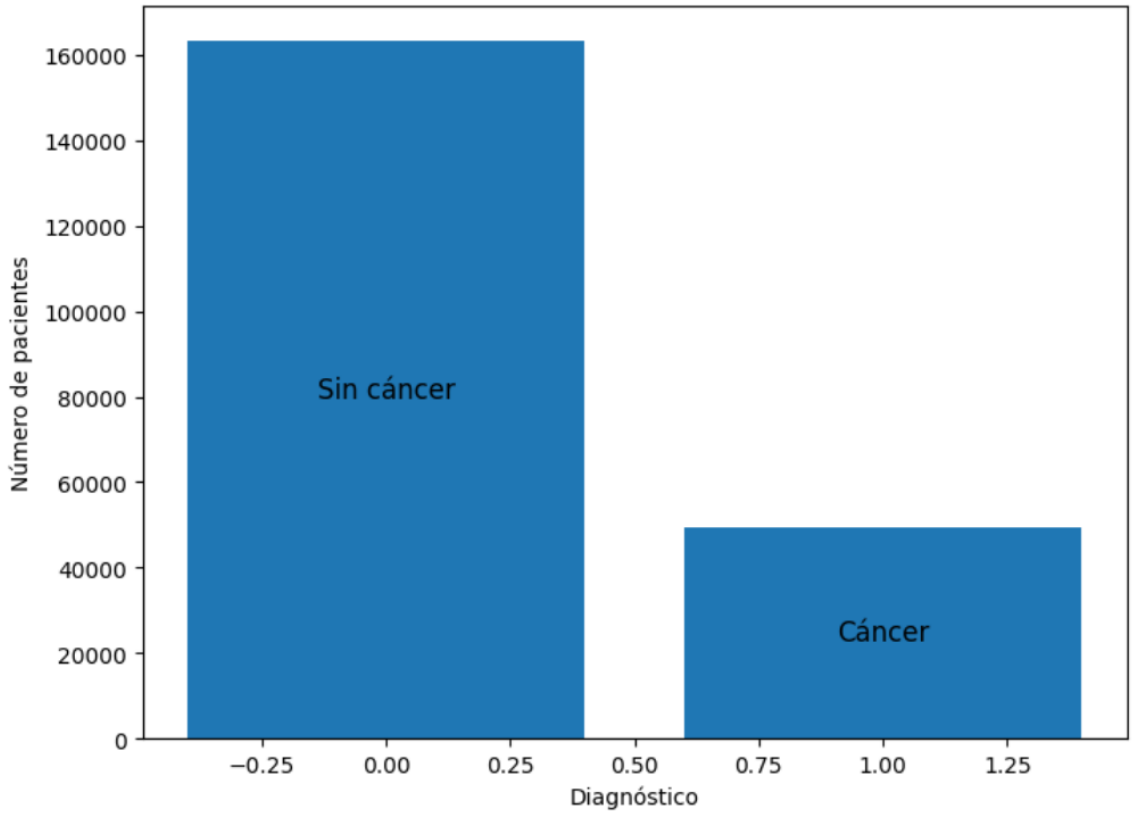
Al observar que no hay outliers, procedemos a graficar los datos para una mayor comprensión empezando por la procedencia de los pacientes en el que vemos que la mayoría de los pacientes son de la India seguido de China que puede ser debido a que ambos países son de los más poblados:



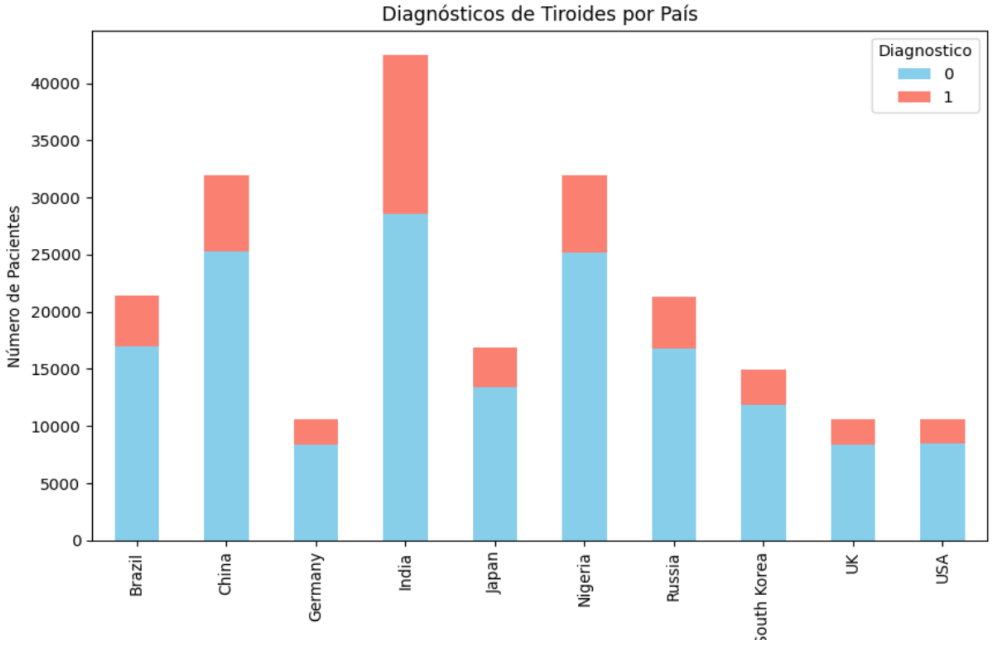
En cambio, cuando graficamos los datos de etnias vemos que la mayoría de pacientes son Caucásicos porque la mayoría de países de los pacientes son Caucásicos (Rusia, Alemania, Reino Unido, Estados Unidos):



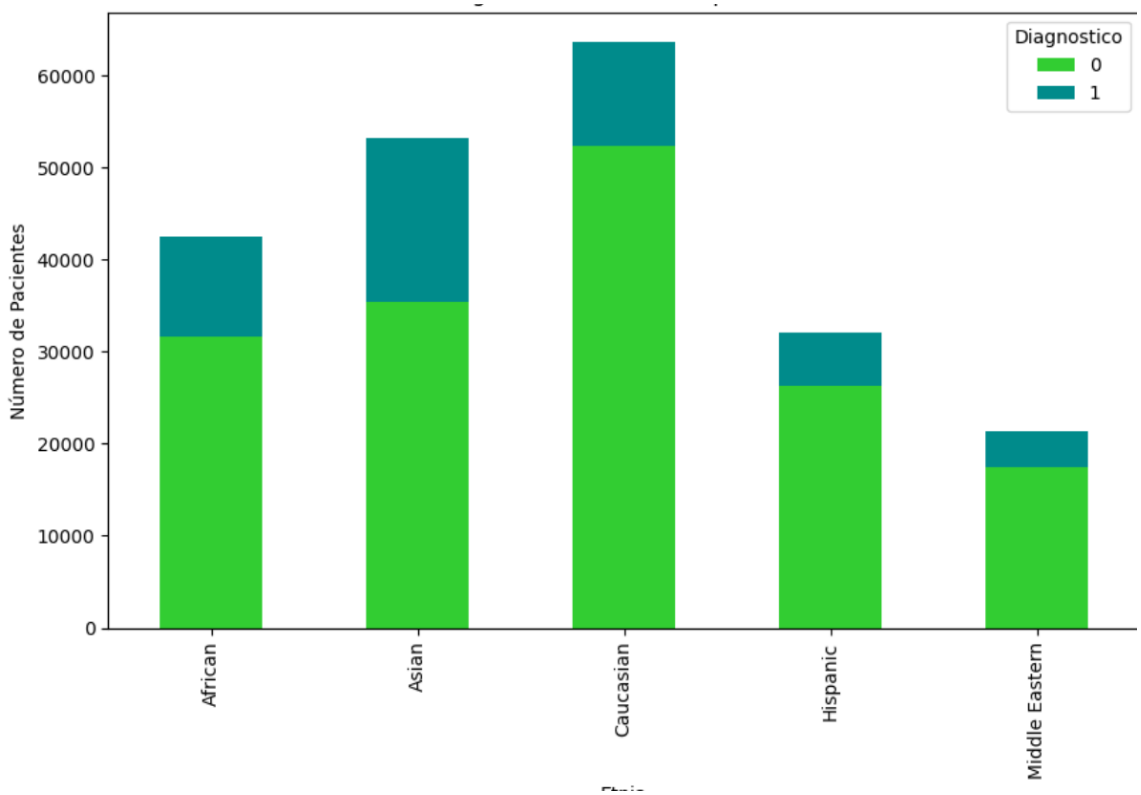
En relación con los pacientes que son diagnosticados vemos que en nuestros datos hay más personas que no tienen cáncer y como muestra la gráfica los datos no están balanceados:



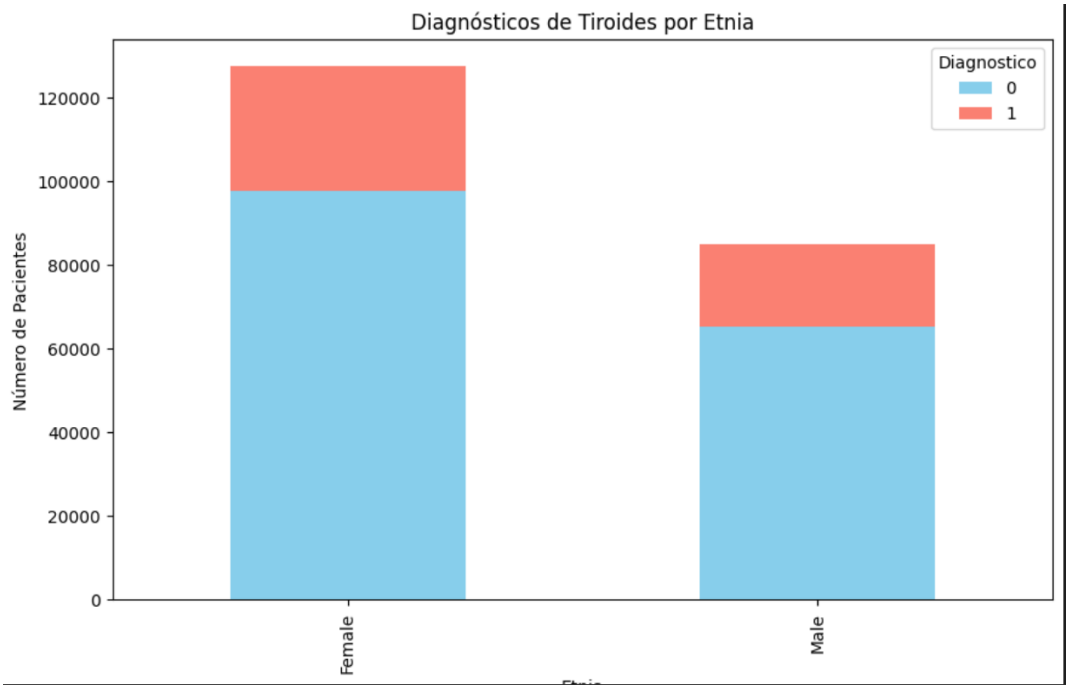
En la siguiente gráfica podemos ver cuántos diagnósticos son positivos o negativos en los pacientes pertenecientes a las distintas países en el que vemos que la India tiene más diagnósticos de cáncer:



En la siguiente gráfica podemos ver cuántos diagnósticos son positivos o negativos en los pacientes pertenecientes a las distintas etnias:



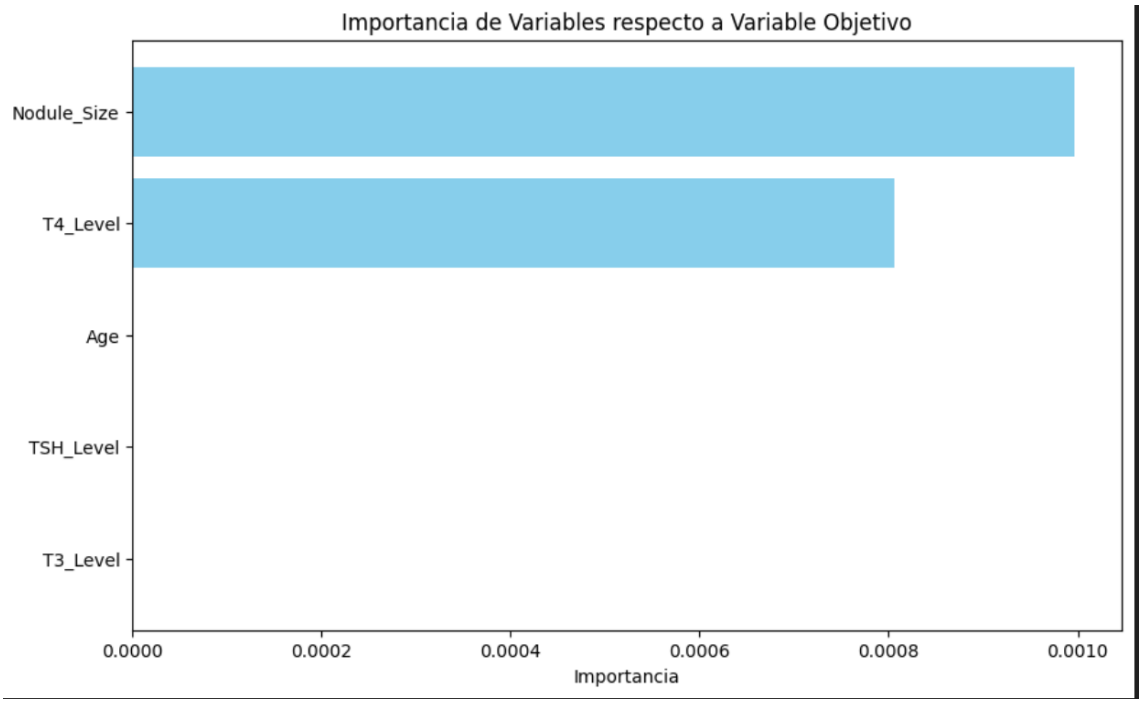
Para observar que género tiene más diagnósticos mostramos la siguiente gráfica en el que se ve que hay más mujeres que hombres pero el diagnóstico de cáncer en ambos es distinto:



En relación hay más hombres que son diagnosticados con cáncer:

	Gender	Diagnostico	Conteo
0	Female	0	97787
1	Female	1	29740
2	Male	0	65409
3	Male	1	19755

Por último, se muestran las variables discretas que son más relevantes con la librería de scikit learn **mutual_info_classif:**



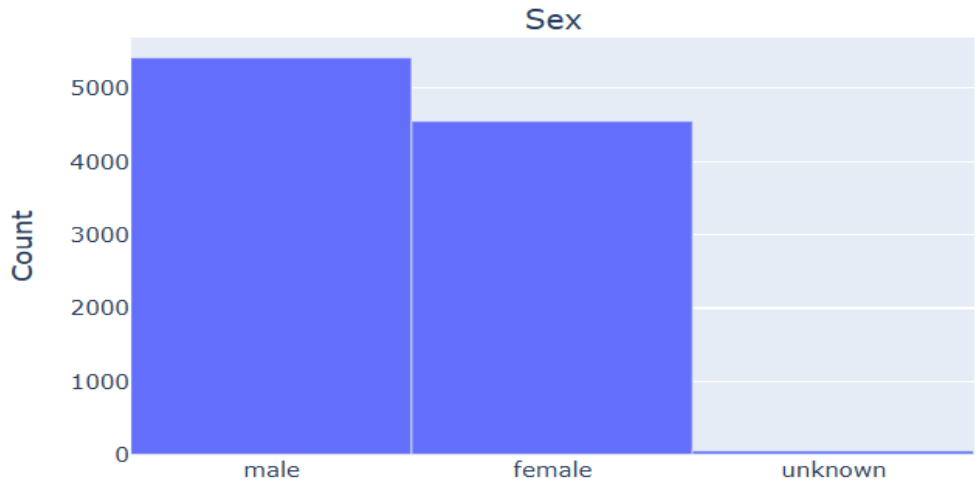
En el apartado de imágenes hemos obtenido los datos del dataset de MNIST: HAM10000, el cual nos proporciona 10015 imágenes de enfermedades cutáneas y unos datos asociados a esas imágenes.

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

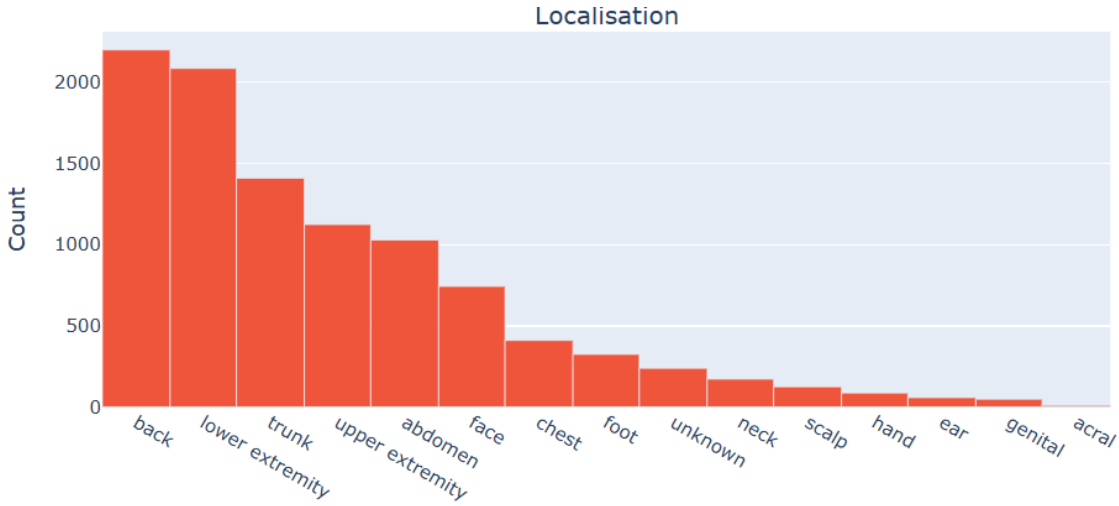
- **Lesion_id** e **image_id** son las columnas que nos dan el path a las imágenes.
- **dx** es el tipo de lesión o enfermedad, hay 7 tipos: bkl, nv, mel, bcc, akiek, vasc y df. (Más adelante se pondrá el nombre completo de la lesión).
- **dx_type** es la manera en la que se confirmó la enfermedad.
- **age** es la edad del paciente.
- **sex** es el sexo del paciente.
- **localization** muestra en que parte del cuerpo se encuentra la enfermedad.

Después de comprobar que no hay datos nulos, procedemos a visualizar los datos en gráficas para que sean más fáciles de analizar.

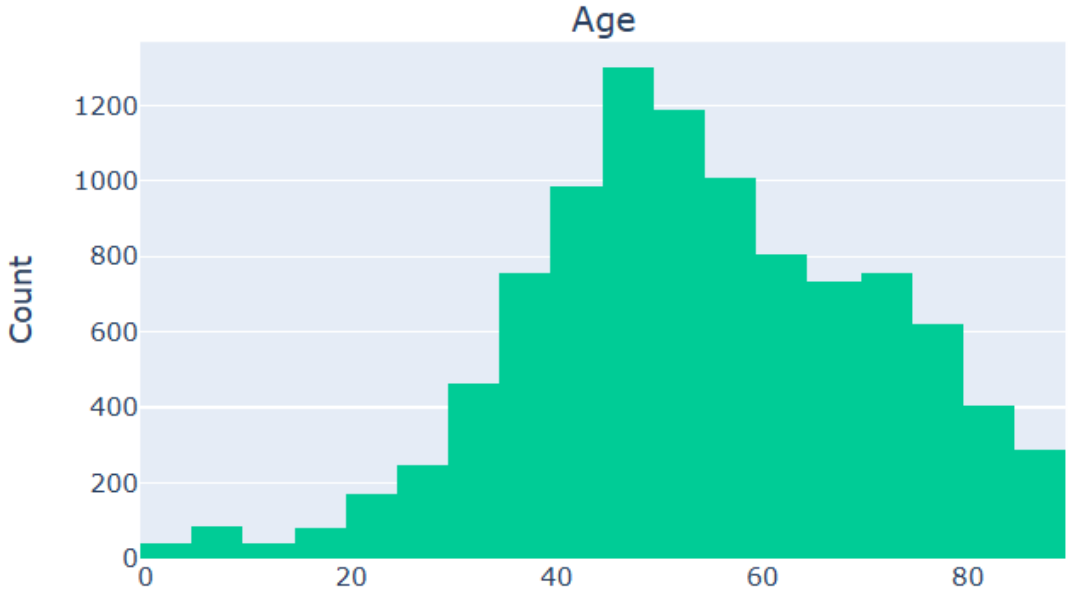
En cuanto al género de los pacientes, la gráfica está bastante balanceada, con una ínfima parte de los datos como **unknown**.



La localización de las enfermedades en el cuerpo sigue una gráfica bastante esperable, ya que los sitios con más lesiones suelen ser las partes que más abarcan del cuerpo como la espalda o las extremidades.

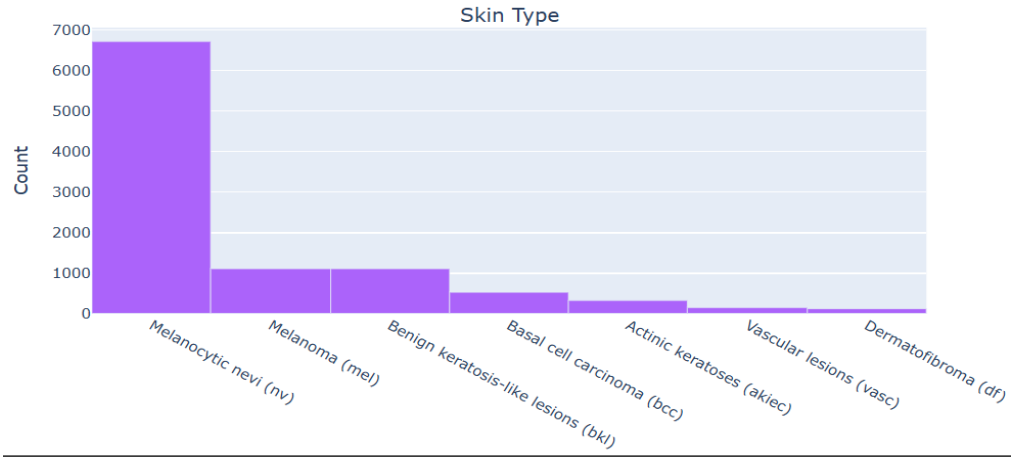


La gráfica de edad tiene sigue una distribución casi gaussiana.



En cuanto al conteo de cuantos datos hay de cada enfermedad, podemos observar que es un dataset bastante poco balanceado, esto se tratará más

adelante.



Por último, podemos comprobar que las imágenes se han cargado bien y se pueden visualizar, en la siguiente imagen se muestran 2 imágenes de cada tipo de enfermedad.

4. DATA PREPARATION

En cuanto a la transformación de los datos, como se ha dicho previamente, hay varias columnas cuyo valor es True o False o son categóricas (Gender, Country, Ethnicity, Family_History,

Radiation_Exposure, Iodine_Deficiency, Smoking, Obesity, Diabetes) por lo que hay que convertir dichas columnas a numéricas.

En nuestro caso, hemos utilizado la librería de pandas get_dummies que, por ejemplo, en caso de género en el que solo hay dos tipos (masculino o femenino), se crea una columna de género masculino con valores de 0 y 1.

Aplicando dicha librería, el dataset ahora pasa a tener 27 columnas.

5. APLICACIÓN DE MODELOS

En el caso de los datos de Kaggle, se van a aplicar dos algoritmos: **LightGBM** y **Redes Neuronales.**

5.1 LIGHTGBM

Es un algoritmo de machine learning basado de **gradient boosting** (es un método de ensamble que construye múltiples árboles de decisión secuenciales) en el que cada árbol aprende del anterior para minimizar errores minimizando la función de perdida mediante gradiente. Además, es más rápido que otros algoritmos de boosting como XGBoost.

En nuestro caso se utilizará la versión de clasificación, por lo que se procede a dividir los datos en entrenamiento (80%) y prueba (20%).

Para garantizar un mejor resultado se aplicará un **gridSearch** para la búsqueda de los mejores hiperparámetros además de dividir los datos en distintas folds para que no siempre aplique al mismo lote de datos, entre los hiperparámetros buscados están:

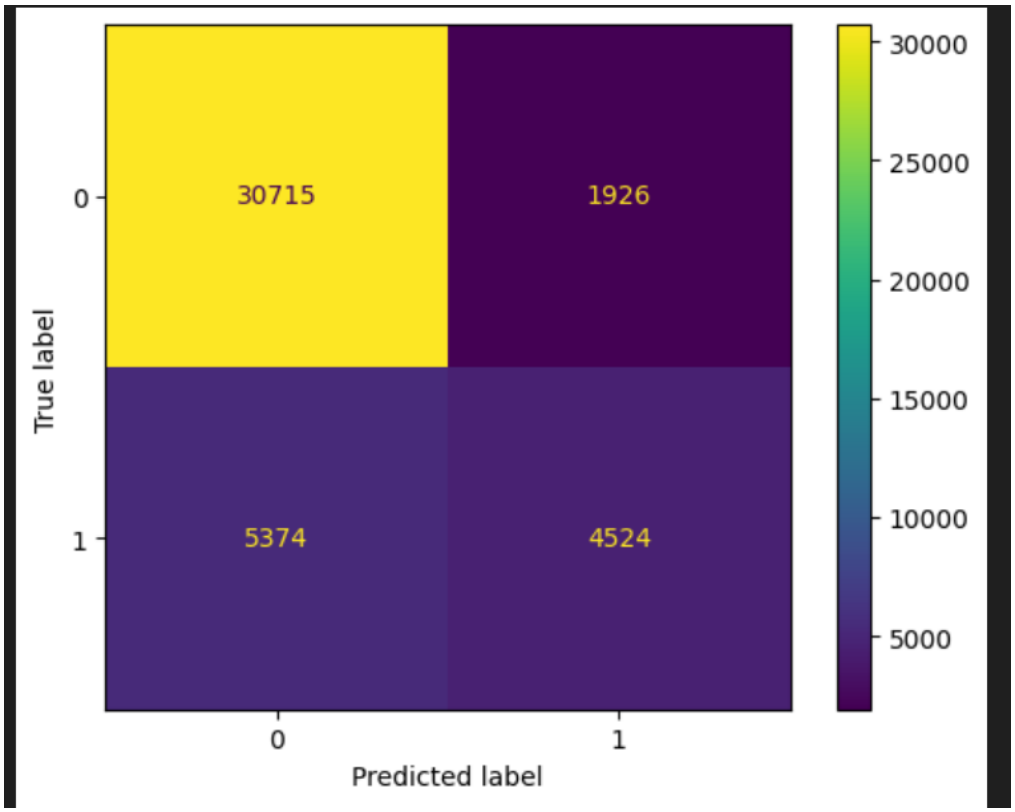
- **num_leaves**: número máximo de hojas en cada árbol.
- **max_depth**: profundidad máxima de los árboles.
- **learning_rate**: controla cuánto se ajusta el modelo en cada iteración.
- **lambda_l2**: regularización L2. Ayuda a prevenir sobreajuste penalizando pesos grandes en las hojas del árbol.
- **n_estimators**: número de árboles.

Ya aplicado el grid search y entrenado el modelo, se procede a aplicar métricas de evaluación para ver como de bueno es nuestro modelo. En nuestro caso se aplicará **accuracy** (cuantos registros ha acertado el modelo tanto para diagnósticos positivos y negativos, es decir, los verdaderos positivos más los verdaderos negativos entre todos) y **precisión** (como de bien a predicho nuestro modelo, es decir, los verdaderos positivos entre los verdaderos positivos más los falsos negativos).

El **accuracy** nos ha salido de un 82,83%.

La **precisión** de nuestro modelo es de un 70%.

Para entender los resultados hemos aplicado una matriz de confusión para ver, dentro de las predicciones, cuales son los verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos:



Aquí podemos ver que en total a predicho **42.539** registros de lo que **30.715** son verdaderos negativos, **1926** son falsos positivos, **5374** son falsos negativos y **4524** son verdaderos positivos.

Los falsos negativos son importantes porque son positivos que de manera errónea se han predicho como negativos, lo cual, es un problema porque en la predicción de cáncer no es lo mismo decir a un paciente que tiene cáncer, pero después ver que no lo tiene que decirle que no tiene cáncer y después detectar que si tiene.

En este caso de **42.539**, **5374** representa un **12%**, lo cual es bastante alto. Esto se puede mitigar añadiendo más datos de positivos para balancear los datos o añadir datos sintéticos si no existen datos clínicos verdaderos.

En este gráfico se ve mejor el resultado obtenido:

```
Reporte de Clasificación:
              precision    recall  f1-score   support

   No Cáncer       0.85      0.94      0.89     32641
      Cáncer       0.70      0.46      0.55      9898

    accuracy                           0.83     42539
   macro avg       0.78      0.70      0.72     42539
weighted avg       0.82      0.83      0.81     42539
```

- **Precision** es el porcentaje de las predicciones fueron positivas.
- **Recall** muestra el porcentaje de positivos reales.
- **F1-score** es la media entre la precisión y recall.
- **Support** es el número de muestras reales de cada clase.

Las conclusiones que se logran sacar son que, al haber mayor número de negativos en el dataset, los predice bien, pero pasa lo contrario cuando buscamos predicciones de cáncer ya que hay muchos menos registros.

5.2 Redes Neuronales

Es un algoritmo de Deep Learning que está compuesto de nodos que procesan información para reconocer patrones y hacer predicciones.

Está compuesta por:

- **Neuronas**: reciben datos, los transforman con una fórmula matemática y envían el resultado a otras neuronas.

- **Capas**: la capa de entrada recibe los datos originales, las capas ocultas procesan los datos internamente y la capa de salida entrega el resultado final.
- **Pesos y sesgos**: parámetros que la red ajusta durante el entrenamiento para aprender.
- **Función de activación**: decide si una neurona se activa con funciones sigmoide o RELU entre otras.
- **Función de perdida:** mide el error entre lo que predice la red y el valor real.
- **Algoritmo de optimización**: ajusta los pesos para reducir el error (como el descenso del gradiente).

En nuestro caso hemos aplicado la librería PyTorch, pero antes de nada hay que escalar los datos para una mejor aplicación del modelo.

Después metemos los datos en tensores que son arrays de una dimensión para luego dividir los datos en entrenamiento y test.

Para aplicar una red neuronal en PyTorch hay que definir la estructura de nuestra red la cual es la siguiente:

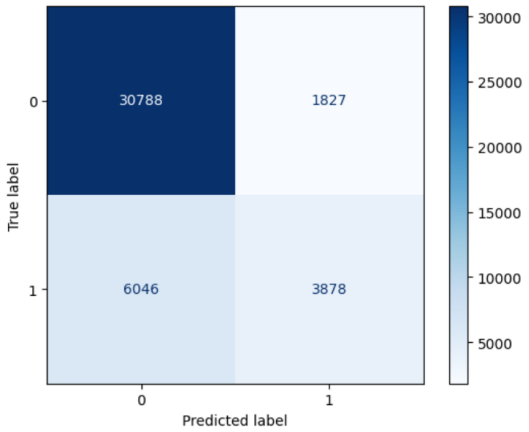
```python
# Definir la red neuronal
class TyroidNN(nn.Module):
    def __init__(self):
        super(TyroidNN, self).__init__()
        self.fc1 = nn.Linear(25, 66)
        self.fc2 = nn.Linear(66, 33)
        self.fc25 = nn.Linear(33, 20)
        self.fc3 = nn.Linear(20, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.relu(self.fc25(x))
        x = self.sigmoid(self.fc3(x))
        return x
```

Nuestra red tiene **5 capas**, una de entrada con **25** neuronas, **tres capas ocultas:** una con 66 neuronas, otra de 33 y otra de 20 y una de salida con una neurona porque es de clasificación. A todas las neuronas aplicamos la

función de activación **RELU** menos a la última que es **sigmoide** porque queremos un resultado entre 0 y 1.

Por otra parte, le aplicamos los hiperparámetros de optimizador (Adam) y learning rate para que los pesos y sesgos se vayan ajustando en cada época de la red neuronal.

Después de aplicar nuestro algoritmo y de utilizar la librería de accuracy sale que predice en un 81,49 %, lo que es ligeramente peor que el LightGBM pero hay que aplicar la matriz de confusión para ver los resultados:



Aquí podemos ver que en total a predicho **42.539** registros de lo que **30.788** son verdaderos negativos, **1827** son falsos positivos, **6046** son falsos negativos y **3878** son verdaderos positivos.

Por lo que el modelo tiene menor accuracy además de que tiene más falsos negativos, esto quiero decir que modelo **lightGBM** es mejor.

En este gráfico se ve más claro los resultados obtenidos:

Reporte de Clasificación:

	precision	recall	f1-score	support
No Cáncer	0.84	0.94	0.89	32615
Cáncer	0.68	0.39	0.50	9924
accuracy			0.81	42539
macro avg	0.76	0.67	0.69	42539
weighted avg	0.80	0.81	0.80	42539

- La **precisión** es el porcentaje de las predicciones fueron positivas.
- El **recall** muestra el porcentaje de positivos reales.
- **F1-score** es la media entre la precisión y recall.
- **Support** es el número de muestras reales de cada clase.

Las conclusiones que se logran sacar son que, al haber **mayor número de negativos** en el dataset, los predice bien, pero pasa lo contrario cuando buscamos predicciones de cáncer ya que hay muchos menos registros.

Por lo que **LightGBM** es mejor modelo que la rede neuronal de **PyTorch**.

5.3 CLASIFICACIÓN POR IMÁGENES

En la parte de clasificación por imágenes no ha habido que cambiar mucho los datos que venían en el archivo csv. Ya que la mayoría estaban con su tipo correcto.

Había 57 datos de edad con valor **NA** los cuales se han rellenado usando la función de **fillna()** y con la media de todos los datos de edad que si tenían número.

Lo primero que se ha hecho es crear un diccionario que añada al dataset el nombre completo de la enfermedad.

```
lesion_type_dict = {
    'nv': 'Melanocytic nevi (nv)',
    'mel': 'Melanoma (mel)',
    'bkl': 'Benign keratosis-like lesions (bkl)',
    'bcc': 'Basal cell carcinoma (bcc)',
    'akiec': 'Actinic keratoses (akiec)',
    'vasc': 'Vascular lesions (vasc)',
    'df': 'Dermatofibroma (df)'
}
```

Lo siguiente ha sido cambiar el tamaño a todas las imágenes a un tamaño de 28x28, se ha probado a usar otras escalas como 64x64 y 128x128, pero no mejoraban el resultado final y a cambio hacia que la red neuronal tardase varias veces más en ejecutarse.

También se ha añadido al dataset una columna con los colores de los pixeles (rgb) de cada foto.

Con estos cambios ya podemos hacer el split en **train**, **validation** y **test**. Lo hacemos así para poder medir la accuracy con los datos de validación mientras se ejecuta la red neuronal. Una vez divididos los datos vamos a pasar a la parte importante de este problema.

Data Augmentation

Ya que los datos estaban tan desbalanceados tenemos que recurrir a esta técnica para que la red neuronal no aprenda solo de un tipo de enfermedad.
Para ello hemos igualado el número de todas las clases al de la que más tuviese, la función **make_oversampled_df()** repite imágenes de las clases que tienen menos hasta que lleguen al número de imágenes requeridas.

Más adelante en la función **prepare_generators()** se cogerán todas las imágenes que estén en **train** y se pasarán por un ImageDataGenerator. Aquí es donde cada imagen será modificada un poquito para que sea diferente a las demás. Se van a hacer hasta 6 modificaciones diferentes:

- **rotation_range**: Rotaciones a las imágenes.
- **width_shift_range**: Desplazamiento horizontal.
- **height_shift_range**: Desplazamiento vertical.
- **shear_range**: Corte o Inclinación.
- **horizontal_flip**: Volteo horizontal.
- **fill_mode**: Relleno de huecos que se han generado.

Red neuronal (Tensorflow.keras)

La red neuronal se ha decidido hacerla con **Tensorflow.keras** ya que es la que mejor resultados me devolvió cuando empecé a trabajar con el proyecto.

```
model = Sequential()
model.add(Conv2D(16, kernel_size = (3,3), input_shape = (28, 28, 3),
activation = 'relu', padding = 'same'))
model.add(MaxPool2D(pool_size = (2,2)))

model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu', padding = 'same')
model.add(MaxPool2D(pool_size = (2,2), padding = 'same'))

model.add(Conv2D(64, kernel_size = (3,3), activation = 'relu', padding = 'same')
model.add(MaxPool2D(pool_size = (2,2), padding = 'same'))
model.add(Conv2D(128, kernel_size = (3,3), activation = 'relu', padding = 'same'
model.add(MaxPool2D(pool_size = (2,2), padding = 'same'))

model.add(Flatten())
model.add(Dense(64, activation = 'relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(7, activation='softmax'))

optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
```

El input que recibe la red neuronal son imágenes de 28x28 con 3 capas.

La red neuronal está compuesta de 4 capas de filtros **Conv2D()** en las que se usan 16, 32, 64 y 128 filtros respectivamente. Cada vez que se ejecuta esta capa se pasará por una función de activación de **ReLU.**

Después de cada capa de filtros se hace un **MaxPool2D()** para reducir el tamaño de las imágenes.

Finalmente se usa **Flatten()** para convertir lo que salga de las convolucionales y convertirlo en un vector que se pasará como entrada a la red neuronal estándar.

Esta red neuronal está compuesta de **3** capas, la primera de **64** neuronas y función **ReLU**, la segunda con **32** neuronas y también **ReLU**, y por último la capa de salida que tendrá **7** neuronas ya que tenemos 7 enfermedades distintas y para ello usaremos la función de activación **softmax.**

La métrica de medida será por **accuracy**.

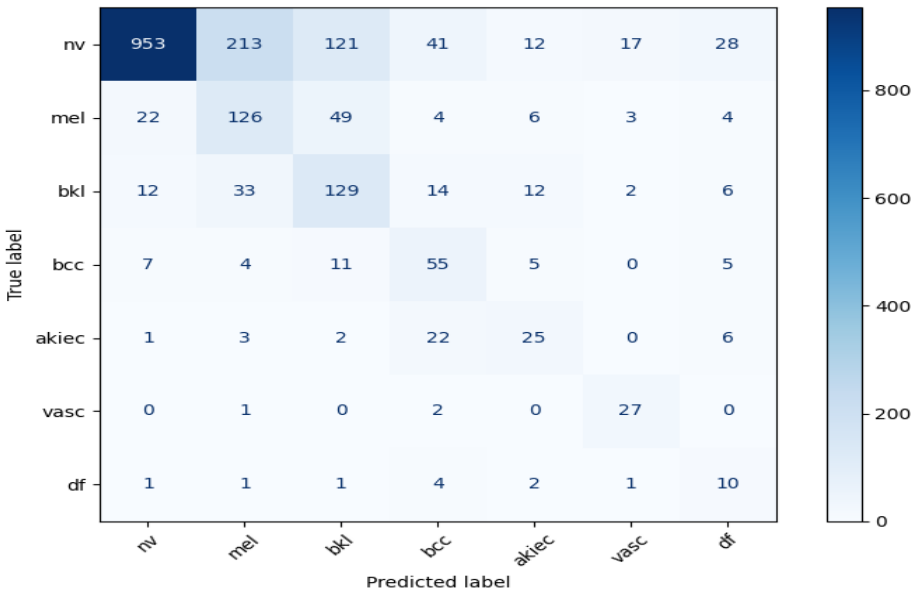
Los resultados finales son los siguientes:

Test Accuracy: 66.151%

	precision	recall	f1-score	support
nv	0.96	0.69	0.80	1385
mel	0.33	0.59	0.42	214
bkl	0.41	0.62	0.50	208
bcc	0.39	0.63	0.48	87
akiec	0.40	0.42	0.41	59
vasc	0.54	0.90	0.68	30
df	0.17	0.50	0.25	20
accuracy			0.66	2003
macro avg	0.46	0.62	0.51	2003
weighted avg	0.78	0.66	0.70	2003

- La **precisión** es el porcentaje de las predicciones fueron positivas.
- El **recall** muestra el porcentaje de positivos reales.
- **F1-score** es la media entre la precisión y recall.
- **Support** es el número de muestras reales de cada clase.

Comparado con los resultados antes de hacer data augmentation se nota una gran subida en el recall de todas las clases que no sean **nv** ya que antes solo hacia bien las predicciones de consiguiendo mejor accuracy pero el recall de las otras 6 era muy malo.



Con este modelo hay bastantes fallos al predecir **nv**, sobretodo con **mel** y **bkl**. Las imágenes entre estas 3 enfermedades son bastante similares por lo cual era probable que pasase.

6. GOOGLE CLOUD PLATFORM

Una vez finalizado el trabajo de aplicación de modelos, se ha procedido a almacenar de forma organizada todos los datos, notebooks, modelos entrenados y el simulador de predicción en **buckets** de **Google Cloud Platform** en formato **standard** para que sean accesibles todas las veces que queramos, garantizando su disponibilidad y seguridad. Además, se ha verificado que tanto los notebooks como los modelos funcionan correctamente utilizando **colab enterprise**, cambiando las rutas de lectura de ficheros asegurando su ejecución sin errores y su capacidad de reproducir los resultados esperados en futuros análisis o despliegues.