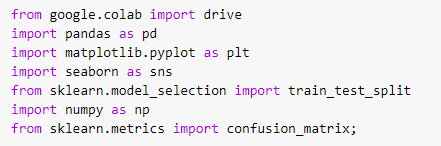
Documentación Notebook

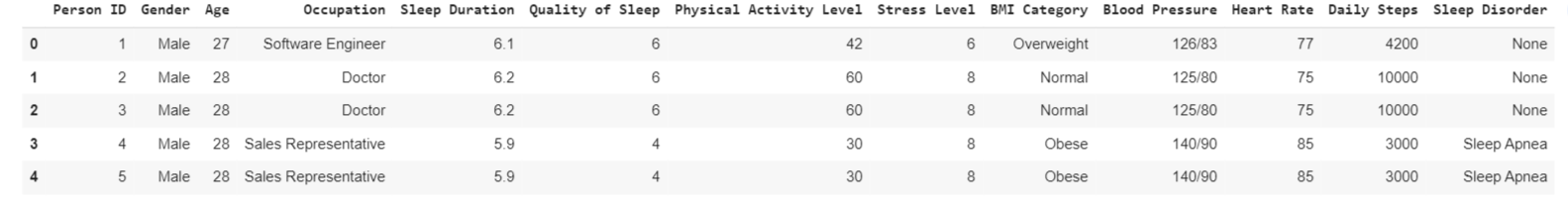
**Importes(Principales)**

Primero importamos las librerías necesarias para la primera parte del proyecto proyecto como la de drive para conectar el notebook con el drive y poder importar nuestro .csv, **matplotlib** y **sns** que son unas librerías de visualización para ver la distribución de los datos, **train\_test\_split** para dividir nuestro dataset, **numpy** para poder trabajar con arrays de numpy en nuestro proyecto y **confusion\_matrix** para crear nuestra matriz de confusión y ver correlaciones.



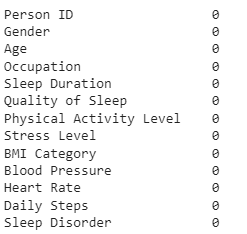
**Primeros pasos y visualizaciones de los datos**

Primero montamos nuestro drive y vemos cómo se divide, sus columnas, los nombres de las mismas, la cantidad de filas y columnas



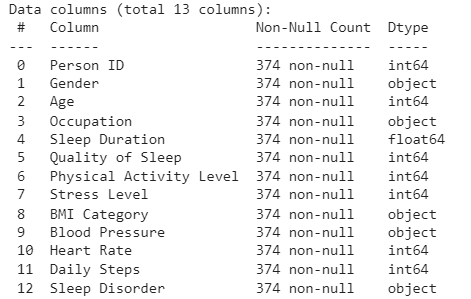
Vemos los tipos de cada columna y la cantidad de valores y valores estadísticos

Vemos si tiene nulos por columna



**Valores por columnas y primeros cambios**

Nos fijamos en las columnas que no son “object” para ver que valores contienen dentro de cada una:



Gender = Vemos que contiene solo ‘Male’ y ‘Female’ entonces lo dejamos así

Occupation = Vemos que se divide en diferentes ocupaciones

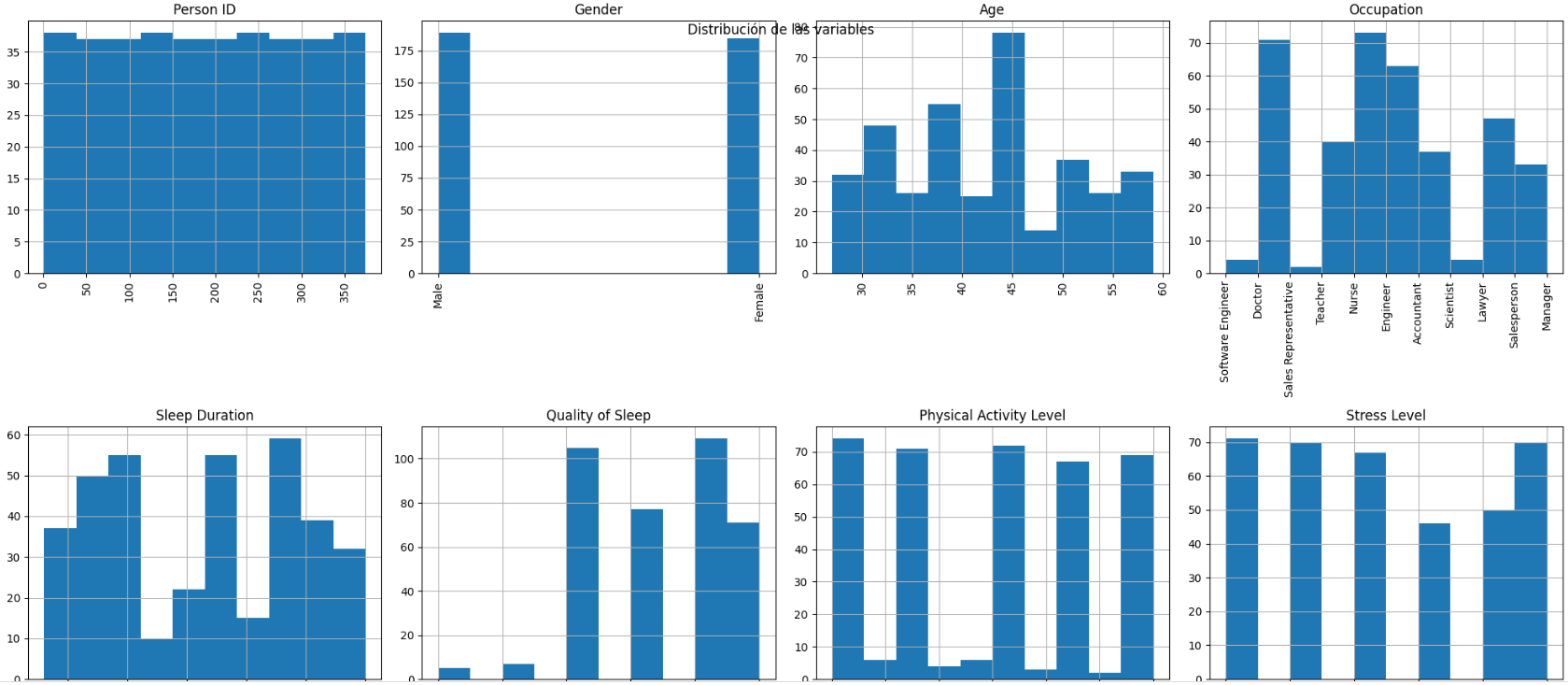
BMI Category = Vemos que dentro de ella hay dos variables que son iguales ‘Normal’ y ‘Normal Weight’ entonces lo que hacemos es unir las dos en ‘Normal Weight’

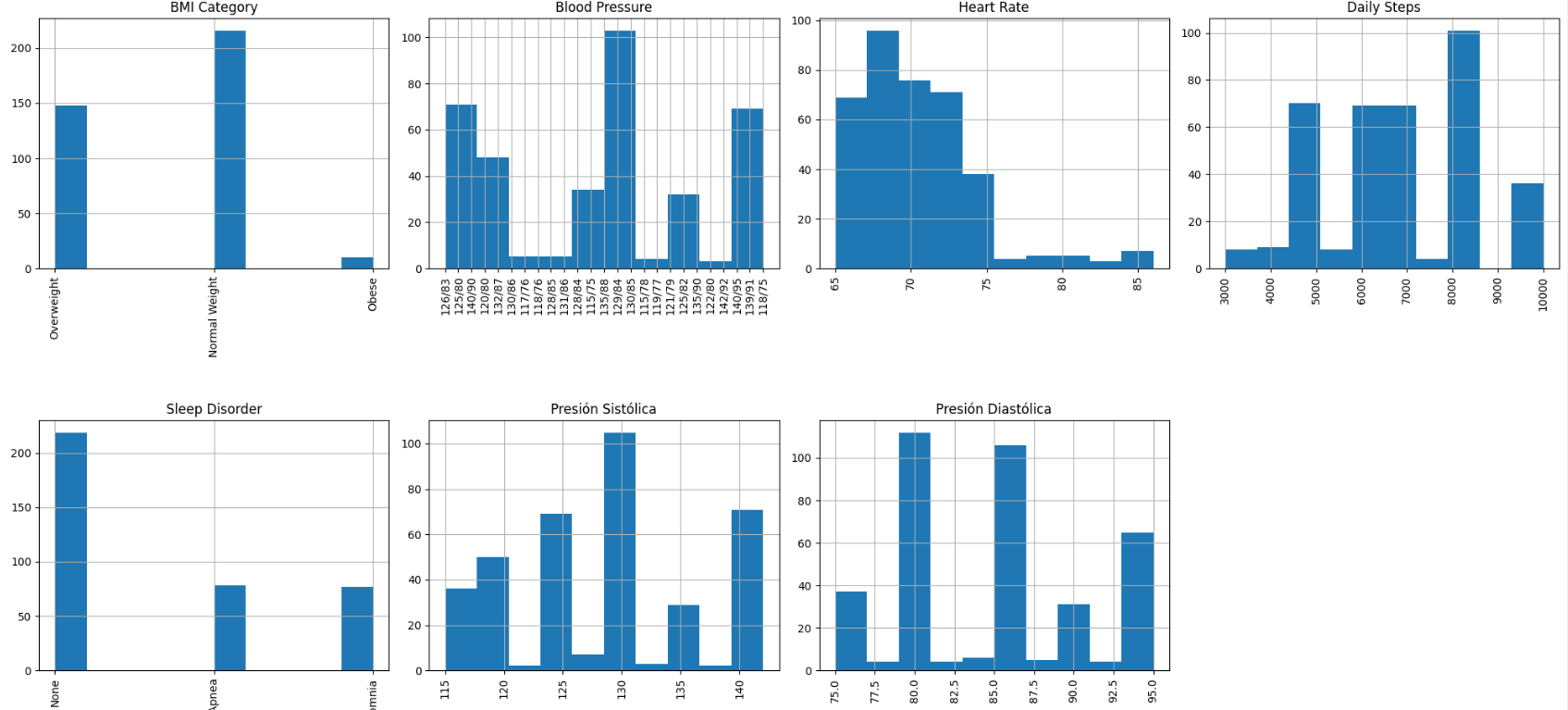
Sleep Disorder = Vemos que se divide en 3 enfermedades

Blood Pressure = Vemos que en la presión sanguínea hay dos valores uno por encima de 100 siempre y otro por debajo de 100 entonces vemos en la descripción del dataset que son la máxima y la mínima por lo que creamos dos columnas una con ‘Presión Sistólica’ o máxima y otra con ‘Presión Diastólica’ o mínima dividiéndolas por el /

**Distribución de variables**

Diistribución de todas nuestras variables:



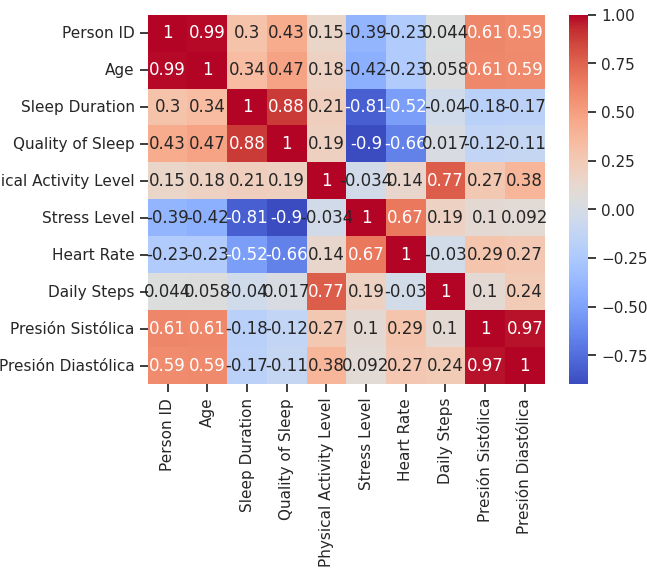


En calidad del sueño solo tenemos representación de 6 valores y del 4 y del 5 tenemos muy muy pocas.

Los demás valores están más o menos distribuidos como estarían en nuestra población.

Las mujeres son mayores que los hombres en nuestro dataset.

Vemos las correlaciones en una gráfica dividiéndolo por Género y podemos intuir las primeras correlaciones que después verificaremos con la matriz de correlación por ejemplo entre las horas de sueño y la calidad o entre el heart rate y el BMI o en los clústeres de edades.

Con la matriz de correlación vemos algunas correlaciones muy claras por ejemplo entre la duración y la calidad del sueño que es nuestra variable a predecir, entre los pasos diarios y nuestra actividad física, entre el heart rate y el estrés y entre la calidad del sueño y la edad

Las observamos por separado y vemos que por lo general cuanto más duerme la gente más calidad del sueño y que va bastante correlacionado la edad y la calidad como por clústeres de rangos de edad y que se intuye que hay menos calidad del sueño en edades intermedias y más calidad en jóvenes y mayores

Observamos que cuando las pulsaciones aumentan la duración del sueño desciende

Vemos también en cuánto a los desórdenes del sueño, que cuanta más edad y estrés, más desórdenes y que cuanta más actividad física menos desórdenes

Vemos que hay profesiones que claramente están muy afectadas por desórdenes y que el BMI es muy importante para ver el desorden del sueño

## 

## **Valoración de reducción de dimensionalidad.**

No haremos ninguna reducción de dimensionalidad porque hay 13 columnas y son todas bastante importantes para el dataset

**Creación de función de limpieza y entrenamiento de los modelos y red neuronal**

Creamos la función para que haga todo lo dicho arriba con la columna del BMI y Presión y a todas las columnas Object le pasamos un encoder para pasarlas a número.

Dividimos el train y el test dándole el 30% de los datos al test, el X\_train son todos los datos menos el Sleep Disorder y al y\_train le quitamos la columna Sleep Disorder que es sobre la que queremos predecir

#### **Implementamos los algoritmos más usados en clasificación y visualizamos sus medidas de rendimiento**

Vamos a implementar 3 modelos con diferentes algoritmos y una red neuronal y veremos cual es el que mejor funciona:

Primero, el (Support Vector Classifier, SVC) es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación. Se basa en el concepto de Máquinas de Vectores de Soporte (SVM, por sus siglas en inglés) y se utiliza para entrenar un modelo que pueda clasificar datos en diferentes categorías



Segundo, el (Stochastic Gradient Descent, SGD) El algoritmo SGD es una técnica de optimización iterativa utilizada en aprendizaje automático para entrenar modelos lineales, como clasificadores lineales o regresores lineales, en grandes conjuntos de datos. A diferencia de los métodos de descenso de gradiente tradicionales, que requieren que los datos se carguen en memoria por completo, el algoritmo SGD procesa los datos en mini lotes, lo que lo hace eficiente en términos de uso de memoria y velocidad de entrenamiento y por eso nos da tal mal porque no es lineal el nuestro:



Tercero, los Bosques Aleatorios son un tipo de modelo de aprendizaje automático que combina múltiples árboles de decisión individuales para realizar predicciones. Cada árbol de decisión se entrena con una muestra aleatoria de datos y produce una predicción. Luego, el resultado final se determina mediante una votación o promedio de las predicciones individuales de los árboles.

En el Random forest usaremos los siguientes hiperparámetros:



Aumentar el número de estimadores (n\_estimators) en un bosque aleatorio proporciona beneficios en términos de estabilidad y generalización del modelo, hemos seleccionado el valor 100 ya que al probar con más no mejoraba el modelo, pero si aumentaba el costo computacional.

Random state se utiliza para controlar la aleatoriedad durante el entrenamiento del modelo.

Cuarto, el (Extreme Gradient Boosting, XGBoost) este método se basa en árboles de decisión y supone una mejora sobre otros métodos, como el bosque aleatorio y refuerzo de gradientes. Los valores se calculan generalmente utilizando el registro de momios y probabilidades. Construye una secuencia de árboles, donde cada árbol se enfoca en corregir los errores del modelo anterior.

**Comprobamos cada modelo con las siguiente métricas:**

Cross Val Predict lo usaremos para generar predicciones utilizando validación cruzada, que nos proporciona una evaluación más precisa del rendimiento del modelo y reduce el riesgo de sobreajuste

La precisión mide la capacidad del modelo para no etiquetar como positivo una muestra que es negativa. Precisión = VP / (VP + FP)

La sensibilidad mide la proporción de verdaderos positivos sobre el total de muestras positivas. sensibilidad = VP / (VP +FN)

F1 score combina la precisión y el recall (recuperación) de un modelo en un solo valor que refleja su rendimiento general.

Después con la matriz de confusión y un mapa de calor vemos como ha predecido cada clase para los dos modelos que mejor valoración dieron que son el SVC y el RandomForest y los dos los testeamos con el test set para comprobar si no ha sobreaprendido y generaliza bien con los nuevos datos

**Conclusión de los modelos**

#### Vemos que tanto el RandomForestClassifier como el XGBClassifier dan muy buenos resultados tanto en la predicción como en la sensibilidad y para las tres clases, asique nos quedaremos con el **XGBoost** ya que es el más actual y tiende a sobreaprender menos que el RandomForest.

**Por último la Red neuronal:**

Para la red neuronal primero vamos a importar los datos otra vez, los limpiamos, los ponemos en número y los arreglamos

Después dividimos la data en X\_train, X\_test, y\_train e y\_test y los pasamos a tensores para trabajar con pytorch y ponemos en punto flotante o float las “X” y las “y” en .long porque los tensores que representan las etiquetas de clasificación se deben representar como enteros en lugar de números de punto flotante así se asegura de que las etiquetas de clasificación se representen correctamente como enteros en los tensores.

**Definición del modelo**

En la entrada tenemos 25 columnas por lo que el input va a ser 25 (26 - la columna a predecir) y ponemos 128 para la entrada de la capa oculta porque previamente probamos con 32 y con 64 y necesitaba más y con 128 no sobre aprende, después le pasamos la función de activación RELU (Rectified Linear Unit) después de la capa de entrada y la capa oculta. La función ReLU es una opción común y efectiva para introducir no linealidad en la red neuronal.

Tenemos una capa oculta de 128X128 porque es la más óptima después de haber probado y como queremos predecir entre 3 clases la salida son 3

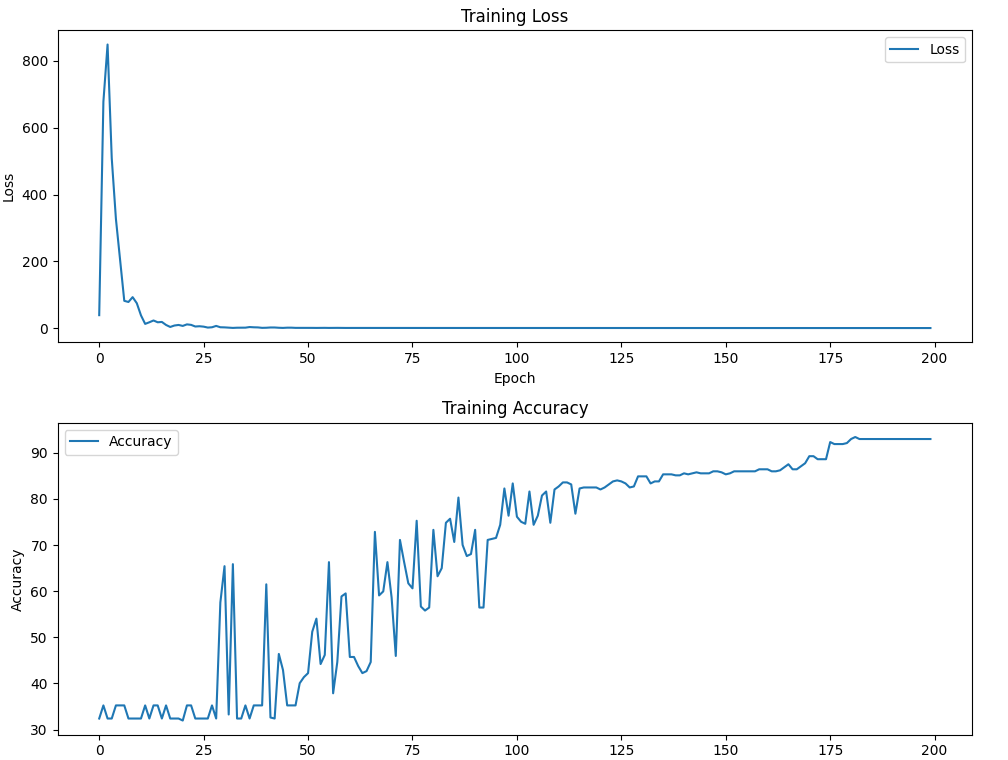
No le ponemos al final una función softmax porque nuestra lossfun es comúnmente utilizada en problemas de clasificación multiclase, combina una función de activación softmax y la función de pérdida de entropía cruzada, lo que hace que sea adecuada para problemas de clasificación con más de dos clases.

Utilizamos el optimizador Adam porque es el mejor

Y el learningrate probamos 0.0001, 0.001, 0.01 y el que mejor resultados nos da es el 0,01 define el tamaño del paso que el algoritmo de optimización da en cada iteración para ajustar los parámetros del modelo.

Creamos 2 listas una para guardarla accuracy y otra para la pérdida

A continuación entrenamos el modelo prediciendo con la red creada y comparando lo predicho con lo correcto y la “loss function” para calcular la pérdida y calculamos la accuracy obteniendo con torch.argmax índice del valor máximo a lo largo de un eje o dimensión específica de un tensor. El índice devuelto representa la posición del valor máximo en esa dimensión por eso el dim=1 que busca el valor máximo para cada muestra

Vemos ploteadas las curvas de las loss y de la accuracy

Por último para comprobar que nuestra red generalice y no haya sobreaprendido usamos el modelo en el X\_test y el y\_test y vemos la precisión calculada de la misma manera que arriba.



**Pasar la red neuronal a una clase**

Para ello creamos una clase llamamos al constructor creamos la red con nuestras capas, hacemos la función de entrenamiento, exactamente como antes pero añadiendo umbrales de pérdida y accuracy que si llega a ellos pare de entrenar y una función para hacer un plot de la loss y de la accuracy.

le damos todos los parámetros que necesita la clase y comprobamos con el test.