Escuela de
Ciencia y Tecnología
ECyT_UNSAM

# MATEMÁTICA III

## *Redes Neuronales*

*Modelo de Predicción de Aprobación Estudiantil basado en Características y Hábitos.*

**Alumnas:** Aburto Melina y Balbi Melisa
**Docentes:** Bompensieri Josefina y Prudente Tomás
**Año: 2025**

# Análisis de la Base de Datos

## Selección de Base de Datos

La base de datos que elegimos explora cómo los hábitos de vida afectan el rendimiento académico de los estudiantes. Contiene 1000 registros sintéticos de estudiantes y diferentes categorías sobre el estilo de vida de los estudiantes y los compara con las calificaciones de los exámenes finales.

***Datos de Entrada:***

*student_id:* Variable categórica nominal. Indica la identificación del estudiante.

*age:* Variable numérica discreta. Indica la edad del estudiante.

*gender:* Variable categórica nominal. Indica el género del estudiante. Para la creación de la red neuronal la transformamos en una variable numérica discreta con los valores 0: Male, 1: Female, 2: Other.

*study_hours_per_day:* Variable numérica continua. Indica las horas que estudia por día.

*social_media_hours:* Variable numérica continua. Indica las horas que está en redes sociales por día.

*netflix_hours:* Variable numérica continua. Indica las horas que está viendo Netflix por día.

*part_time_job:* Variable categórica binaria. Indica si el estudiante tiene un trabajo de medio tiempo. Para la creación de la red neuronal la transformamos en una variable numérica discreta.

*attendance_percentage:* Variable numérica continua. Indica el porcentaje de asistencia a clase.

*sleep_hours:* Variable numérica continua. Indica las horas de sueño del estudiante.

*diet_quality:* Variable categórica ordinal. Indica la calidad de la dieta, se clasifican en Poor, Fair y Good.

*exercise_frequency:* Variable numérica discreta. Indica la frecuencia semanal que el estudiante hace ejercicio.

*parental_education_level:* Variable categórica ordinal. Indica el nivel de educación de los padres, se clasifica en HighSchool, Bachelor, Master, None.

*internet_quality:* Variable categórica ordinal. Indica la calidad del internet, se clasifica en Poor, Average, Good.

*mental_health_rating:* Variable numérica discreta. Indica la calificación de salud mental del 1 al 10.

*extracurricular_participation:* Variable categórica binaria. Indica si hace o no alguna actividad extracurricular.

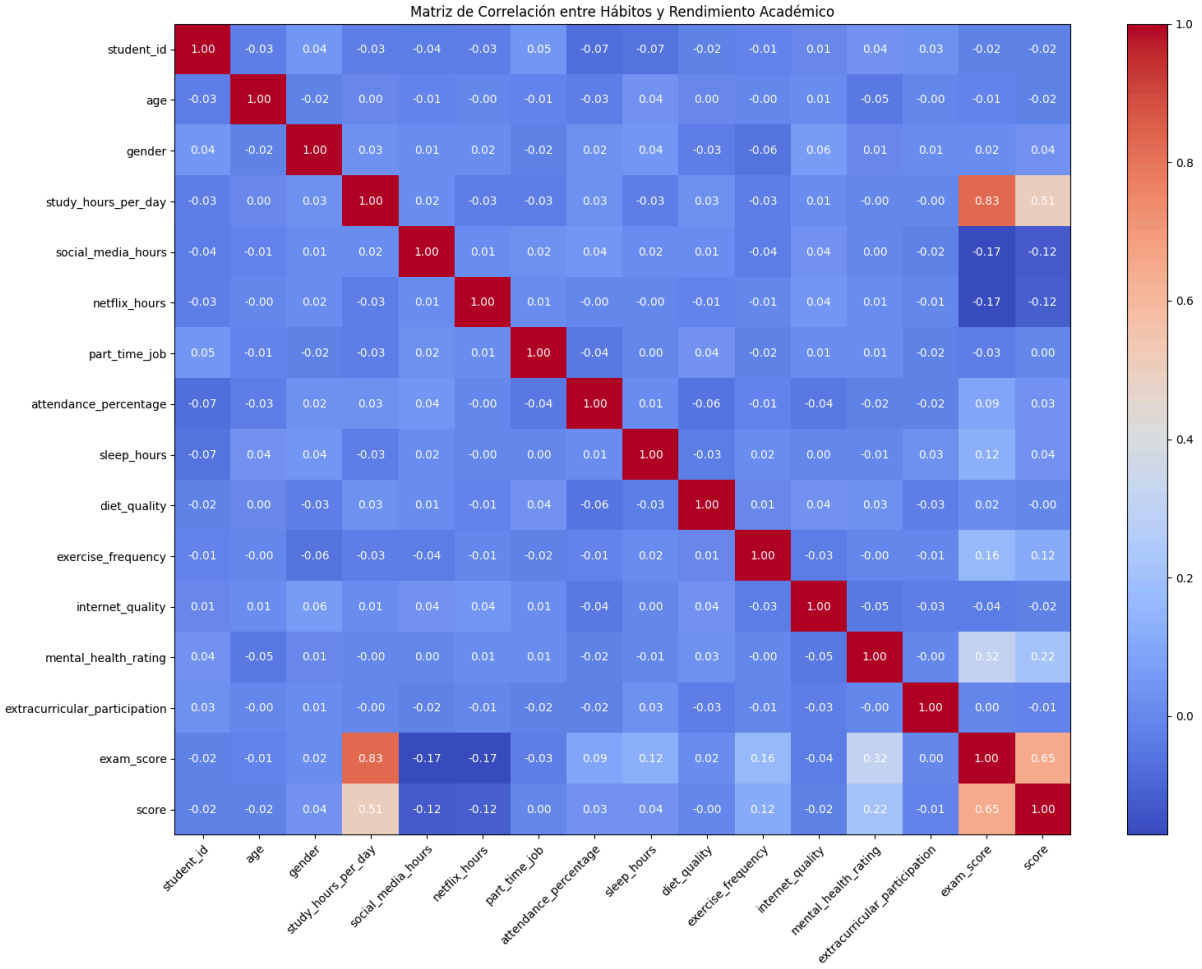
***Columna Objetivo:***

*exam_score:* Variable numérica continúa. Indica la calificación que obtuvo el alumno en el examen final.

*score:* Variable numérica discreta. Indica si el alumno aprobó o no el examen final, con valores 0: No aprobó, 1: Aprobó. Esta columna es una transformación de “exam_score” para el uso en la red neuronal.

## Análisis de Correlaciones

Antes de realizar el gráfico de correlaciones realizamos la normalización de los datos, dividiendo a cada uno de ellos por el número máximo de su columna.



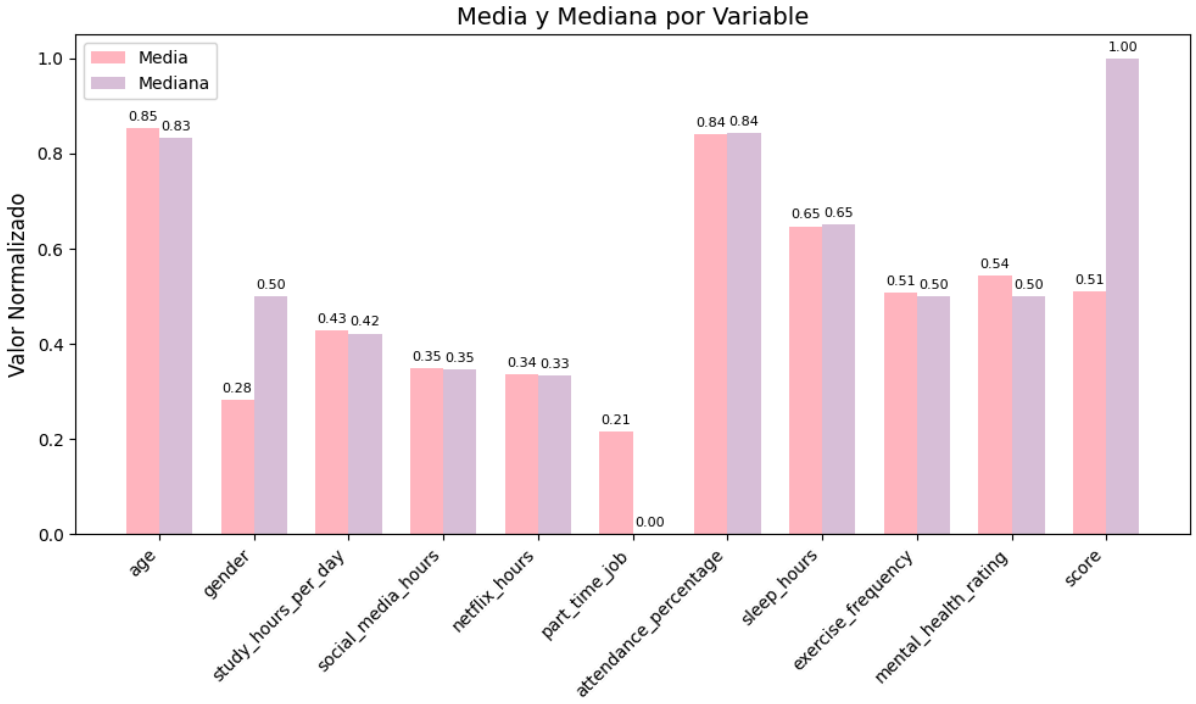

Analizando las correlaciones podemos ver que las entradas que son más influyentes sobre nuestra columna objetivo son: *study_hours_per_day* con una relación del 0.51, *mental_health_rating* con 0.22, *social_media_hours* y *netflix_hours* con una relación inversa del 0.12, y *exercise_frequency* con 0.12.

## Análisis de Factibilidad

Creemos que nuestra base de datos es adecuada porque contiene datos sintéticos que simulan estilos de vida reales de diferentes estudiantes y cómo estos pueden afectar en el desempeño académico.
El propósito de nuestra red es identificar cómo afecta, positiva o negativamente, cada hábito del estudiante sobre la nota del examen final. La clasificación final que obtendremos será si un alumno aprobó o no aprobó el examen.

## Datos Atípicos y Limpieza de Datos

La base de datos no posee valores atípicos, en el gráfico se puede ver como la media y la mediana tienen valores similares en cada categoría por lo que no fue necesario hacer la limpieza de datos.
En caso de que fuera necesaria la limpieza de datos, utilizando el *rango intercuartílico* $Q_1$, $Q_2$ y $Q_3$, calculamos { $(Q_3 - Q_1) * 1.5 + Q_3$ } y { $Q_1 - (Q_3 - Q_1) * 1.5$ } los valores que queden fuera de rango se consideran valores atípicos y serían eliminados.

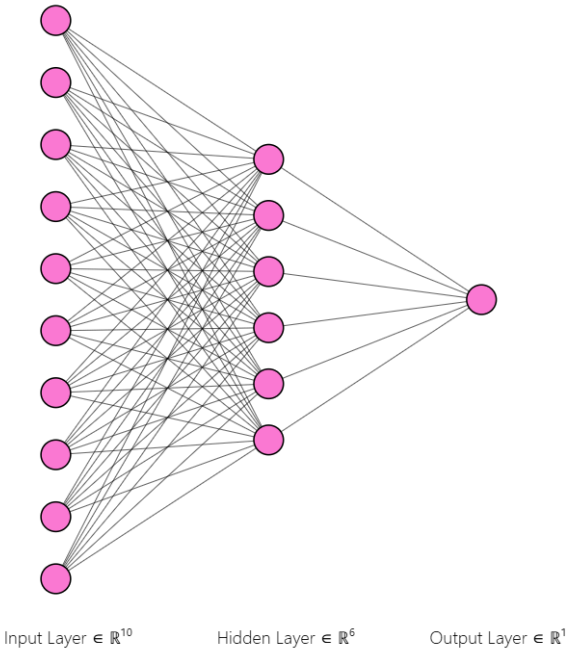



## Transformaciones Preliminares

Luego de cargar nuestro dataset vimos necesario hacer diferentes modificaciones sobre nuestros datos. Todas las columnas categóricas las transformamos a valores numéricos por la incapacidad de la red de entender las diferentes categorías y solo poder trabajar con entradas y salidas numéricas.
Se agregó una nueva columna objetivo llamada *score* que se obtuvo transformando la columna *exam_score* mediante una función que comparaba la nota del examen con el criterio de aprobación, nos retornaba 1 si el alumno aprobó y 0 si no aprobó, así logramos tener en nuestra columna de salida una variable numérica discreta.
En base a los resultados del gráfico de correlaciones vimos necesario reducir la cantidad de entradas, eliminando las columnas que no presentaban relación relevante con la columna objetivo para el aprendizaje de nuestra red neuronal. Las columnas eliminadas fueron: *diet_quality, internet_quality, parental_education_level y extracurricular_participation*.
Por último, decidimos normalizar la base de datos, cada columna la dividimos por su valor máximo para obtener números entre el 0 y 1 para optimizar el cálculo de la red, en nuestro caso no sucede pero en otro tipo de base de datos la normalización es necesaria para evitar el *overflow*.
Para guardar todas las transformaciones realizadas sobre el dataset se creó un nuevo archivo llamado *“student_habits_performanceNormalizado.csv”.*

# Desarrollo de la Red Neuronal

## Arquitectura de la Red




La red neuronal que creamos es de tres capas:

***Capa de Entrada (Input Layer)*** con 10 entradas.
***Capa Oculta (Hidden Layer)*** con 6 neuronas y con la función de activación ***ReLu***. Se eligió esta función de activación ya que ayuda a evitar que los gradientes se vuelvan demasiado pequeños durante el entrenamiento, lo que permite un entrenamiento más eficiente con muchas iteraciones. Además, *ReLu* aporta la no linealidad necesaria para que la red pueda modelar relaciones complejas entre las entradas y las salidas, mejorando la precisión. Al ser simple y rápido de calcular, contribuye a un entrenamiento más rápido y estable en nuestro modelo con la arquitectura que usamos.
***Capa de Salida (Output Layer)*** con 1 nodo de salida con una capa de función de activación ***Logistic*** (sigmoidea). Esta elección se basa en que al tratarse de una salida binaria,la función *logistic* permite obtener una probabilidad en el rango [0, 1], lo que facilita la interpretación del resultado final como una predicción binaria.

## Implementación de Numpy

Luego de la carga y limpieza del csv ya detallado anteriormente empezamos con la creación de nuestra red neuronal.

- Definimos los datos de entrada y salida.
- Separamos los datos ⅔ para entrenamiento y ⅓ para testear, mediante *sklearn.model_selection import train_test_split.*

```
xEntrenamiento, xTest, yEntrenamiento, yTest = train_test_split(allInputs, allOutputs,
    test_size=1/3)
```

- Inicializamos los pesos y sesgos aleatoriamente con una *seed* para asegurarnos de que cada ejecución tenga la misma inicialización y siempre obtener los mismos resultados. Cada peso se multiplica por 0.2 para facilitar que la red comience a aprender de manera eficiente.

```
np.random.seed(10)
hidden_size = 6
w_hidden = np.random.rand(hidden_size, 10)*0.2
w_output = np.random.rand(1, hidden_size)*0.2
b_hidden = np.random.rand(hidden_size, 1)*0.2
b_output = np.random.rand(1, 1)*0.2
```

- Se definen las funciones de activación *ReLu* para la capa oculta y *Logistic* para la capa de salida.

```
relu = lambda x: np.maximum(x, 0)
logistic = lambda x: 1 / (1 + np.exp(-x))
```

- Se define el *Forward Propagation* con la capa oculta y de salida.

```
def forward_prop(X):
    Z1 = w_hidden @ X + b_hidden
    A1 = relu(Z1)
    Z2 = w_output @ A1 + b_output
    A2 = logistic(Z2)
    return Z1, A1, Z2, A2
```

- Se define el *Backward Propagation,* calculando la derivada del costo.

```
dC_dA2 = 2 * A2 - 2 * Y
dA2_dZ2 = d_logistic(Z2)
dZ2_dA1 = w_output
dZ2_dW2 = A1
dZ2_dB2 = 1
dA1_dZ1 = d_relu(Z1)
dZ1_dW1 = X
dZ1_dB1 = 1
```

```
dC_dW2 = dC_dA2 @ dA2_dZ2 @ dZ2_dW2.T

dC_dB2 = dC_dA2 @ dA2_dZ2 * dZ2_dB2

dC_dA1 = dC_dA2 @ dA2_dZ2 @ dZ2_dA1

dC_dW1 = dC_dA1 @ dA1_dZ1 @ dZ1_dW1.T

dC_dB1 = dC_dA1 @ dA1_dZ1 * dZ1_dB1

return dC_dW1, dC_dB1, dC_dW2, dC_dB2
```

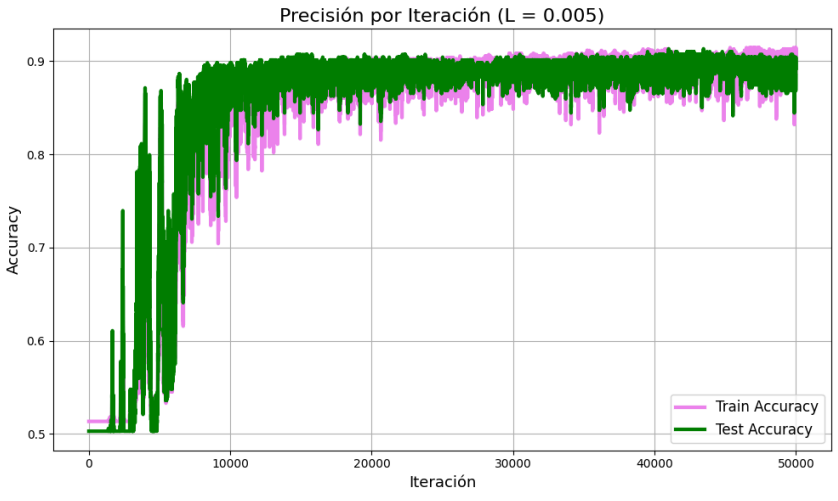
## Entrenamiento y Evaluación

Con el fin de evaluar el rendimiento de nuestra red neuronal, utilizamos diferentes valores para la tasa de aprendizaje e iteraciones.

- Se realiza el descenso de gradiente estocástico seleccionando aleatoriamente datos del conjunto de entrenamiento y los utilizamos en el *forward propagation* para que la red aprenda y nos produzca una predicción en la capa de salida.
- Los valores obtenidos de Z1, A1, Z2, A2 los usamos en el *backward propagation* buscando actualizar los sesgos y pesos de la red para obtener la mejor predicción posible.
- Se realiza el cálculo de precisión de entrenamiento y de prueba para su comparación.

Luego de ejecutar nuestra red con diferentes valores, obtuvimos que los mejores valores para el aprendizaje de la red son:

*Tasa de aprendizaje (L):* 0.005
*Número de Iteraciones:* 50 000



### Análisis de Overfitting

Durante el entrenamiento del modelo se realizaron múltiples pruebas con distintas combinaciones de cantidad de iteraciones y tasas de aprendizaje (L), con el objetivo de analizar el comportamiento del aprendizaje y detectar posibles señales de overfitting.

Se utilizaron:

**Iteraciones:** 1 000, 10 000, 50 000 y 100 000

**Tasas de aprendizaje:** 0.001, 0.005, 0.01, 0.05 y 0.1

En particular, al analizar la evolución de la precisión con L = 0.005, se observó que el modelo alcanzaba un buen desempeño en ambos conjuntos ya en las primeras 10.000 a 15.000 iteraciones, mostrando una mejora sostenida hasta alrededor de las 30.000 iteraciones. A partir de ese punto, aunque se mantiene de forma considerada el nivel de porcentajes de aciertos, la precisión sobre el conjunto de prueba comenzó a mostrar mayor variabilidad y leves caídas, mientras que la precisión de entrenamiento se mantuvo elevada y más estable.

Esto sugiere que el modelo comienza a sobre ajustarse a los datos de entrenamiento después de cierto punto, reduciendo su capacidad de generalización. En función de esto, se decidió limitar el número de iteraciones a 50.000 como máximo, ya que entrenar más allá de ese umbral no aporta mejoras significativas y puede generar un rendimiento menos estable en datos nuevos. (o hacerle en todo caso un early stopping entre las 30.000 a las 40.000 iteraciones para mayor eficacia)

Esta decisión permitió mantener un buen equilibrio entre rendimiento, tiempo de entrenamiento y capacidad de generalización del modelo, evitando el sobreentrenamiento.

## Comparación con scikit-learn

Para evaluar la efectividad y eficiencia de nuestra red neuronal, comparamos dos implementaciones: una realizada manualmente con *NumPy* y otra utilizando la librería de alto nivel *Scikit-Learn*.
Ambas redes fueron entrenadas sobre el mismo conjunto de datos, con la misma división de entrenamiento y validación y gracias a esto, se pudo observar que:

- Ambos modelos lograron resultados de precisión similares sobre el conjunto de prueba (alrededor de 90%), lo que indica que ambos enfoques fueron capaces de aprender patrones relevantes del conjunto de datos.
- Ambas versiones mostraron un rendimiento equilibrado entre los test y los entrenamientos.
- La red con scikit-learn fue notablemente más rápida en tiempos de ejecución (ya que está optimizada internamente).
- La versión con scikit-learn es mucho más sencilla en su implementación(abstrae detalles como el cálculo manual del backpropagation o la gestión de sesgos y pesos)
- Nuestra red hecha con NumPy, aunque más compleja, permite un mayor control (para modificaciones) sobre el proceso de aprendizaje de la red neuronal.

# Conclusión Final

Una de las principales ventajas de construir la red neuronal manualmente fue que nos permitió entender el funcionamiento y mecanismo de creación y entrenamiento de la misma. Además nos permitió experimentar con diferentes configuraciones sobre cantidad de neuronas en la capa oculta, tasa de aprendizaje y cantidad de épocas para el aprendizaje, lo cual fue útil para entender y evitar el overfitting de nuestra red.
Sin embargo, también se nos presentaron inconvenientes en el desarrollo manual ya que es más propenso a errores, más costoso en tiempo y menos eficiente.
El uso de una librería como scikit-learn simplifica enormemente el proceso, nos permite entrenar el modelo de forma rápida, con muy pocas líneas de código. Su uso resulta ideal para las tareas en las que se prioriza la eficiencia y la rapidez.
En resumen, trabajar desde cero fue clave para consolidar conocimientos teóricos, mientras que la comparación con scikit-learn permitió valorar la potencia de las herramientas disponibles en la práctica profesional actual.