

# ANALÍSIS DEL RENDIMIENTO ACADEMICO



MMCD

EQUIPO 6:  
PÉREZ MARTÍNEZ ÁNGEL NOEL  
RIVERA HERNÁNDEZ MILENA FERNANDA  
CRUZ MENDOZA VALENTINA AYELEN  
MONROY VILLEGAS ISAAC  
ZAMORA ANTIGA ÁNGEL JAVIER



INSTITUTO DE  
INVESTIGACIONES  
EN MATEMÁTICAS  
APLICADAS Y  
EN SISTEMAS

# CONTEXTO

- Interés por parte de muchas **instituciones**.
  - **Riesgo**,
  - **SELECCIÓN**
- ¿Estudiantes con familias de bajos ingresos, bajo nivel de alfabetización?
- **Factores demográficos, sociales y educativos.**

# CONJUNTO DE DATOS

A través de Kaggle obtuvimos el DataSet **Student Performance Factors**, que ofrece un panorama general sobre diversos factores (Horas de estudio, asistencia, disponibilidad de recursos, horas de sueño, etcétera) que se piensa afecta el rendimiento estudiantil en exámenes

**6,607 registros, 20  
características**

**Valores numéricos y  
categóricos**

**Compleitud mayor al 98%  
en todas las características**

Hours_Studied
Attendance
Parental_Involvement
Access_to_Resources
Extracurricular_Activities
Sleep_Hours
Previous_Scores
Motivation_Level
Internet_Access
Tutoring_Sessions
Family_Income
Teacher_Quality
School_Type
Peer_Influence
Physical_Activity
Learning_Disabilities
Parental_Education_Level
Distance_from_Home
Gender
Exam_Score

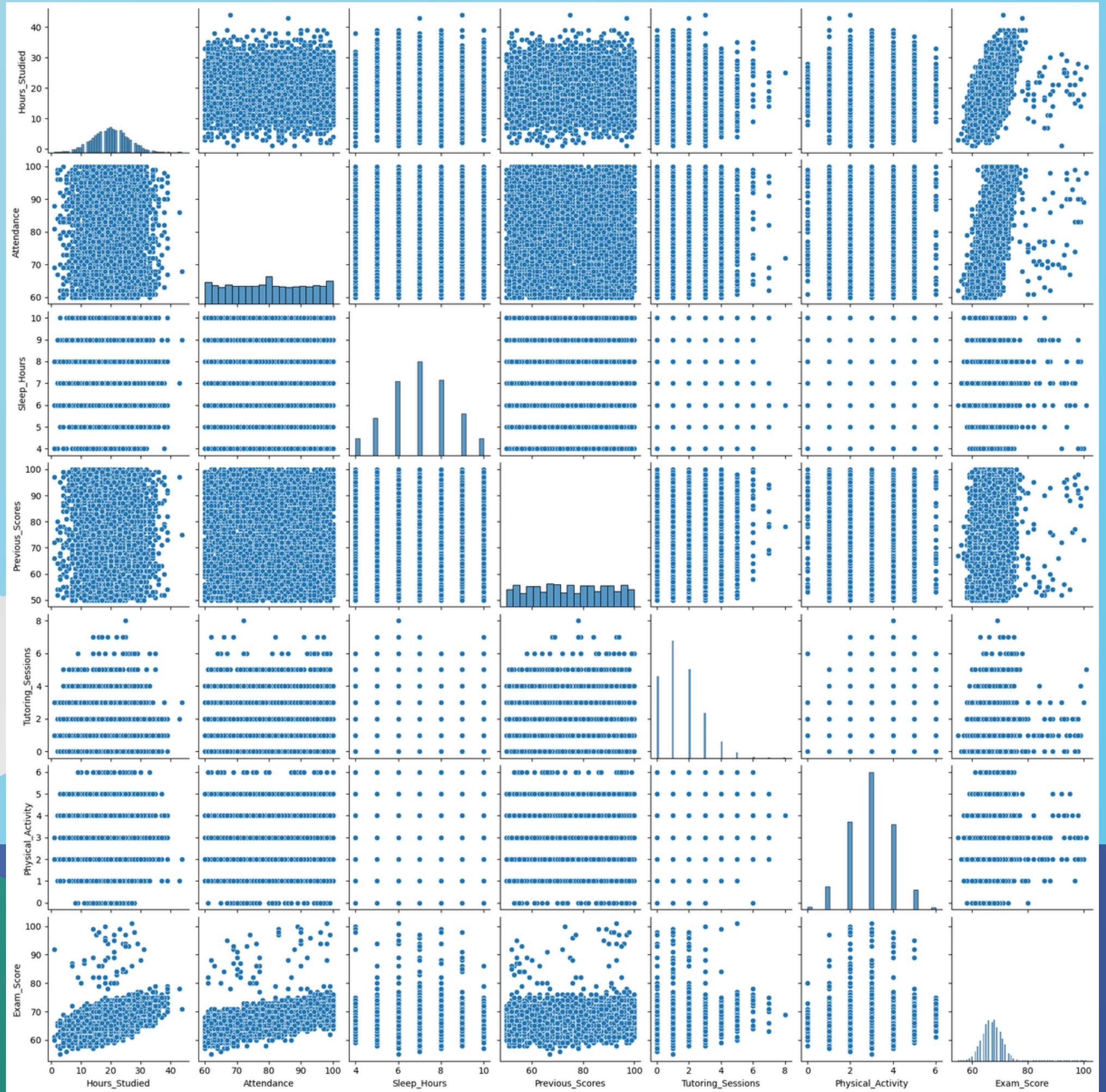
# OBJETIVOS

- Condensar las relaciones a través del **Análisis de Componentes Principales** minimizando la pérdida de información.
- Facilitar la futura predicción sobre el rendimiento del alumnado en evaluaciones.

# HIPÓTESIS

A través de PCA se podrán identificar los 3 factores más influyentes. Que empíricamente se asume serán:

- Horas de estudio
- Horas de sueño
- Asistencia



## DISTRIBUCIONES POR PARES

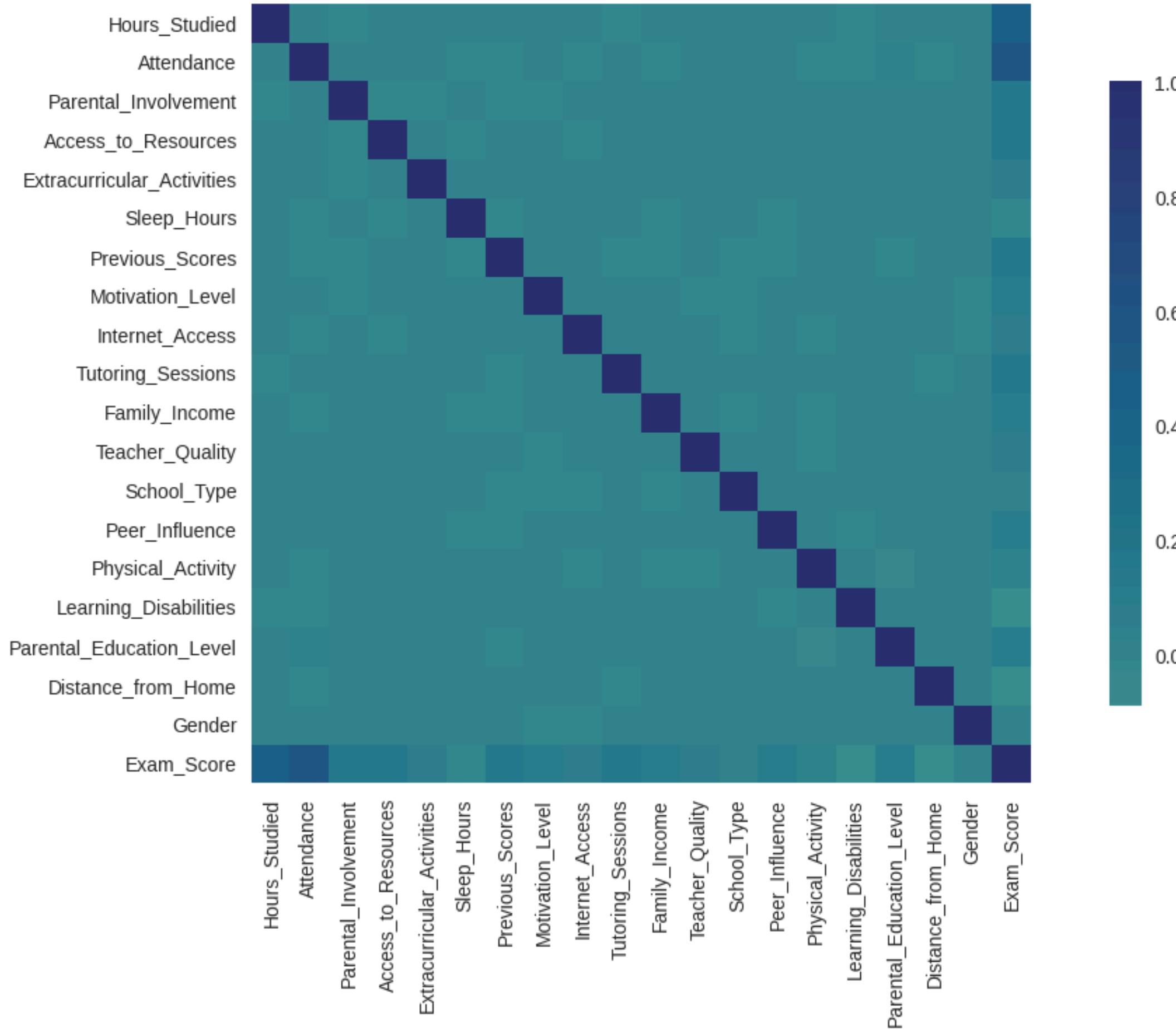
Destacamos las gráficas sobre la diagonal: Algunas podrían relacionarse con distribuciones conocidas. Asimismo, en otras es observable una gran correlación.

# IDENTIFICACIÓN DE CORRELACIONES



**Se tiene un alto índice de correlación del puntaje en el examen con las horas de estudio y asistencia a clases**

Matriz de Correlación - Variables Numéricas



# PREPROCESAMIENTO

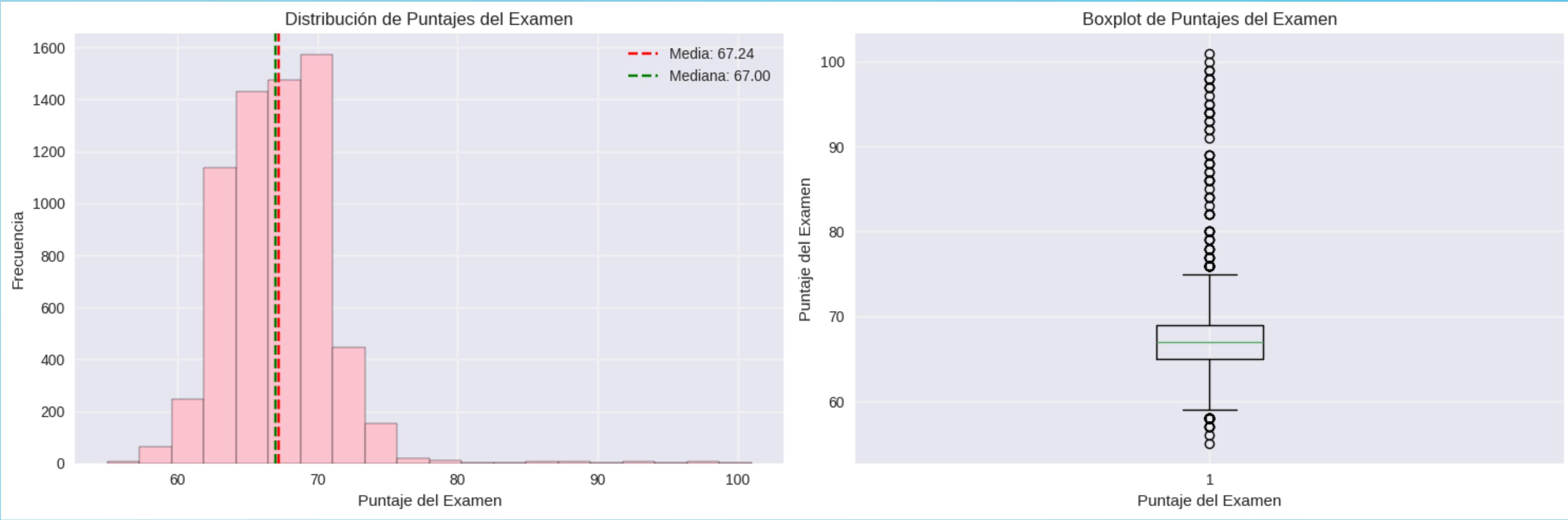
```
1 medida_map = {'Low': 1, 'Medium': 2, 'High': 3,  
2                 'Yes': 1, 'No': 0,  
3                 'Positive': 2, 'Negative': 0, 'Neutral': 1,  
4                 'Near': 1, 'Moderate': 2, 'Far': 3,  
5                 'Male': 0, 'Female': 1,  
6                 'Public': 0, 'Private': 1,  
7                 'High School': 1, 'College': 2, 'Postgraduate': 3}  
8 |  
9 for column in data.columns:  
10    if data[column].dtype == 'object':  
11        data[column] = data[column].map(medida_map)  
12  
13 display(data.tail())
```

## SUSTITUCIÓN NULOS

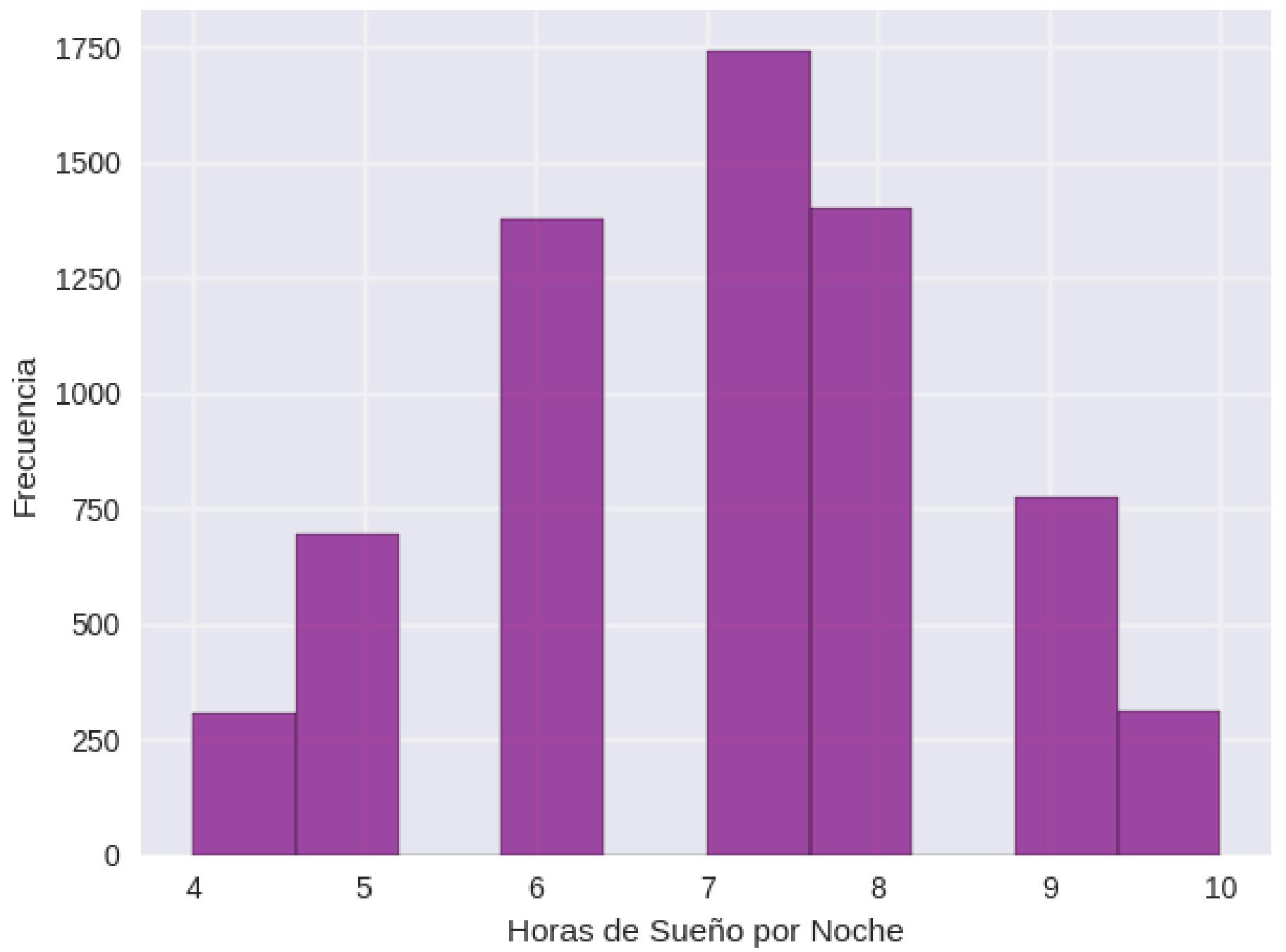
```
1 # Sustitución por columna  
2 for col in data.columns.values:  
3     falta = np.sum(data[col].isnull())  
4     if falta:  
5         print(f'Asignando {falta} valores en columna {col}')  
6         med = data[col].median()  
7         data[col] = data[col].fillna(med)
```

```
Asignando 78 valores en columna Teacher_Quality  
Asignando 90 valores en columna Parental_Education_Level  
Asignando 67 valores en columna Distance_from_Home
```

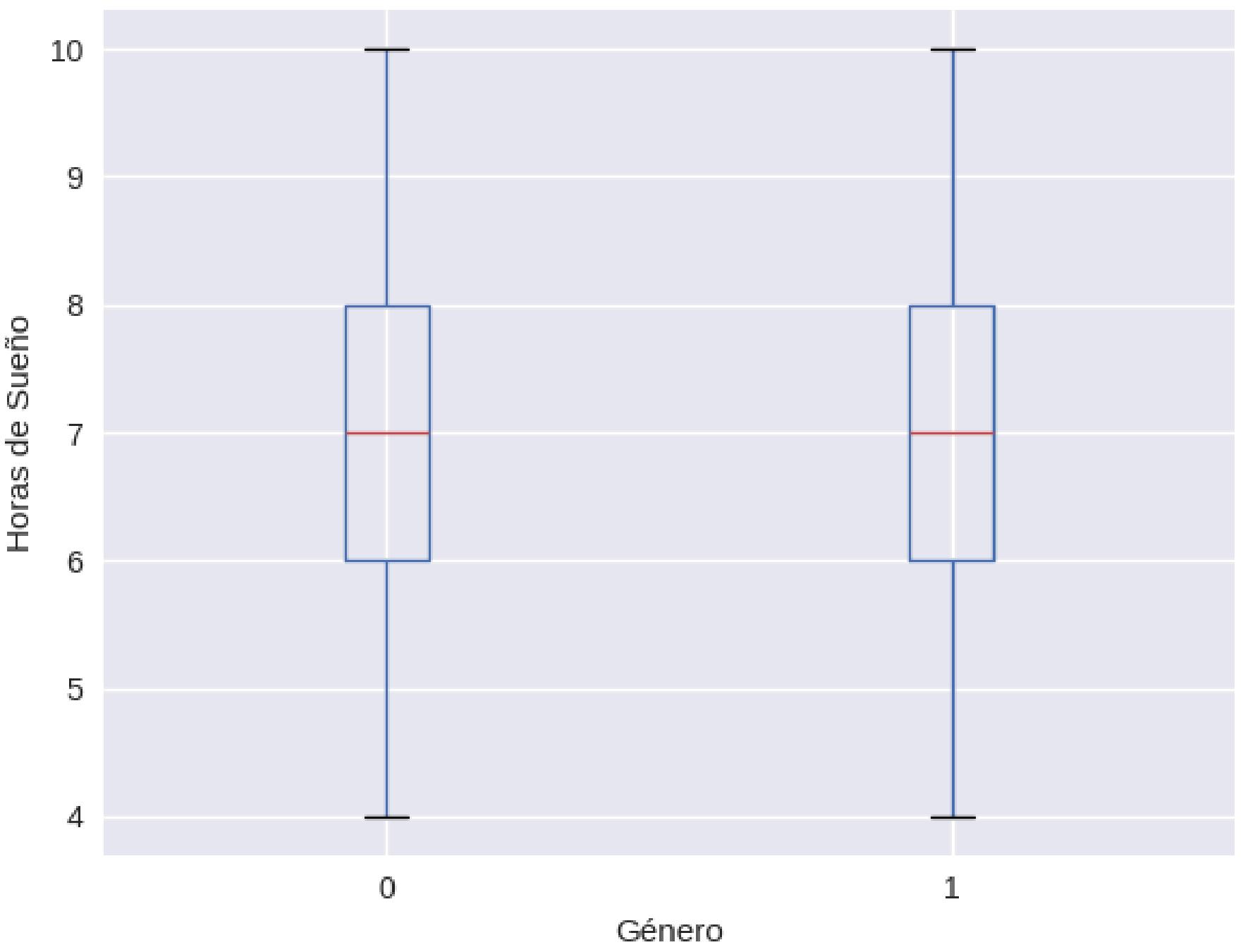
Transformarmos  
valores categóricos a  
numéricos y llenamos  
valores nulos con la  
**mediana**

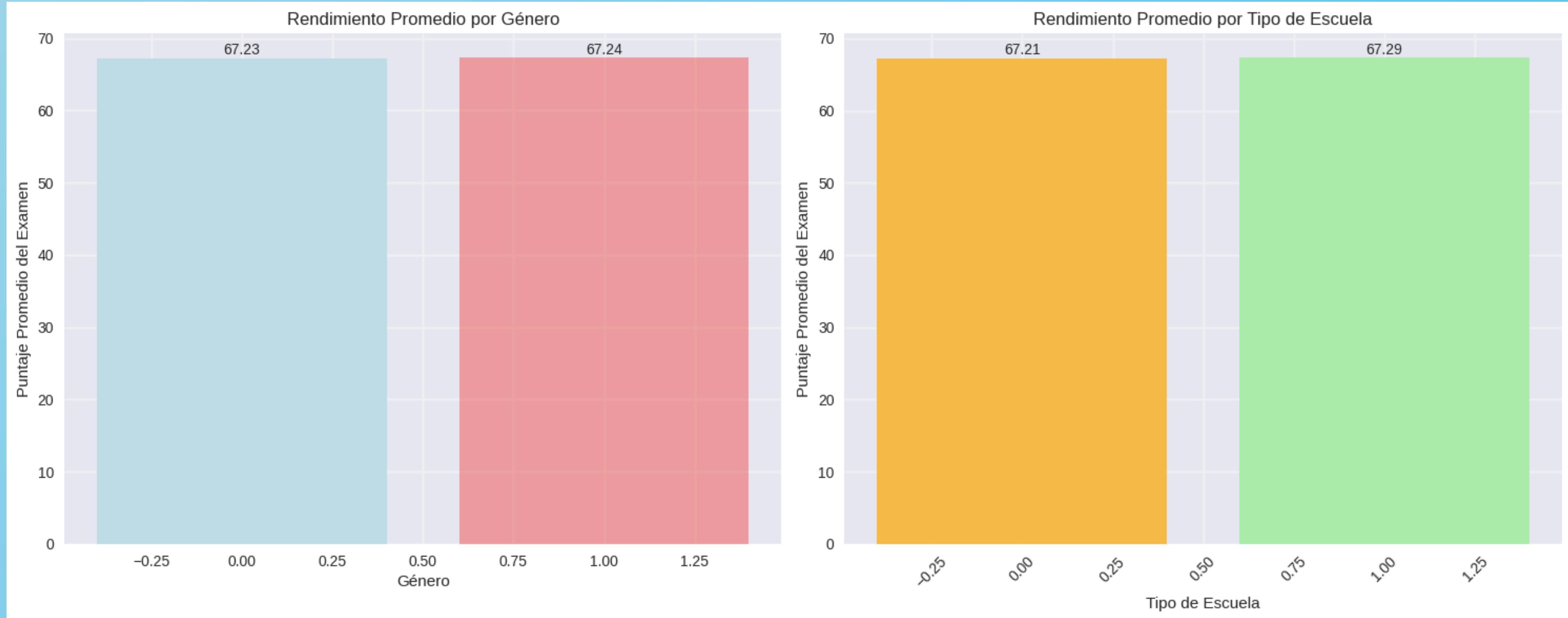


Distribución de Horas de Sueño

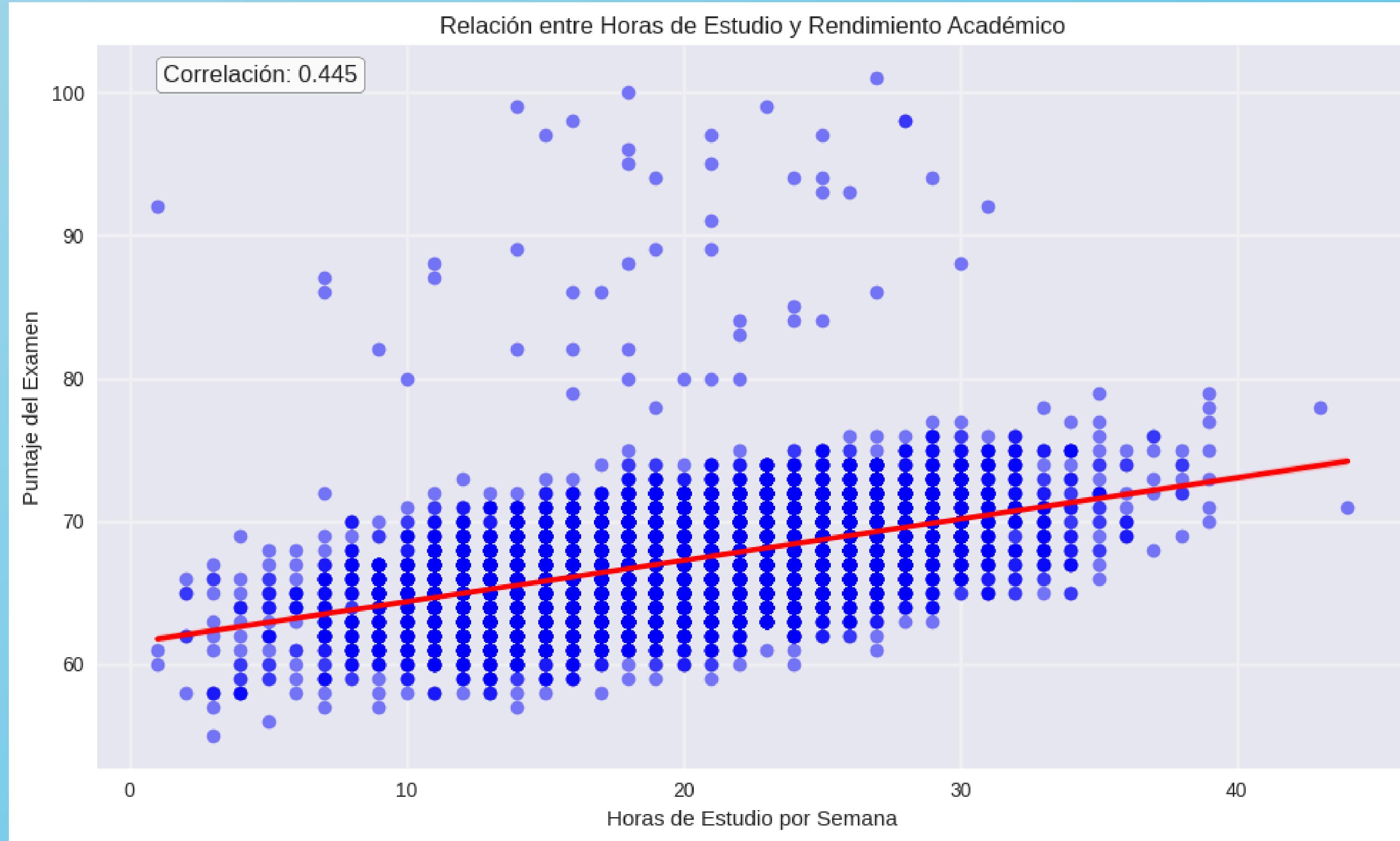


Horas de Sueño por Género



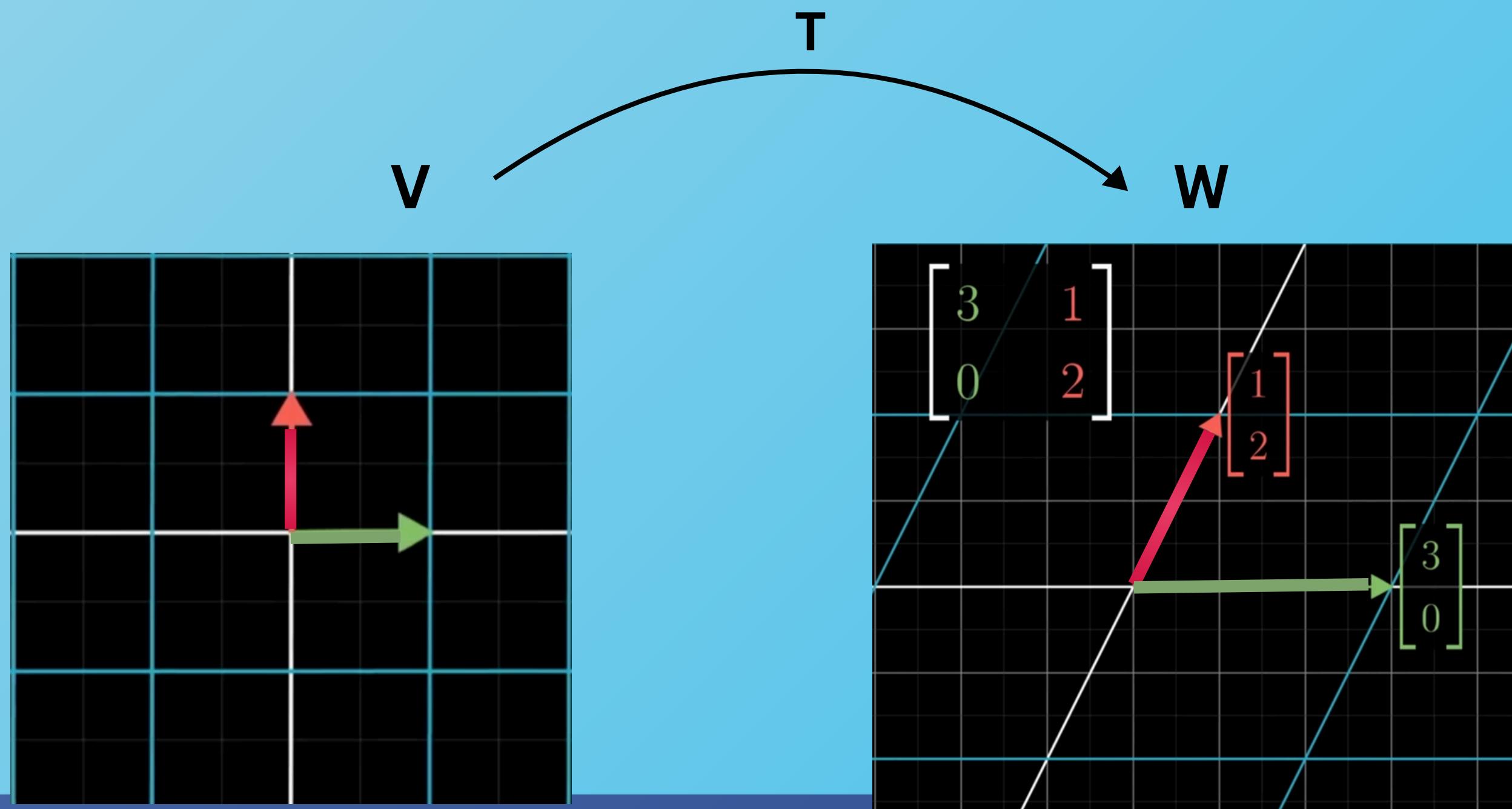


## Relación entre Horas de Estudio y Rendimiento Académico



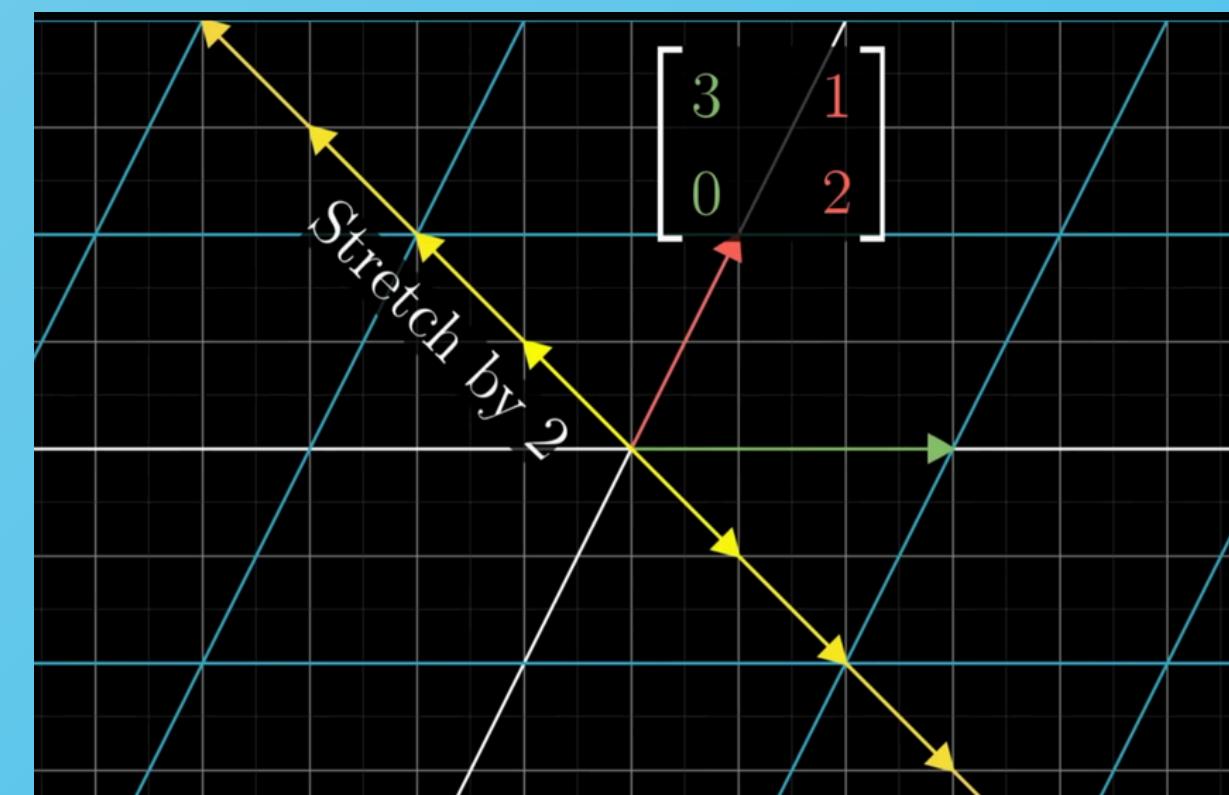
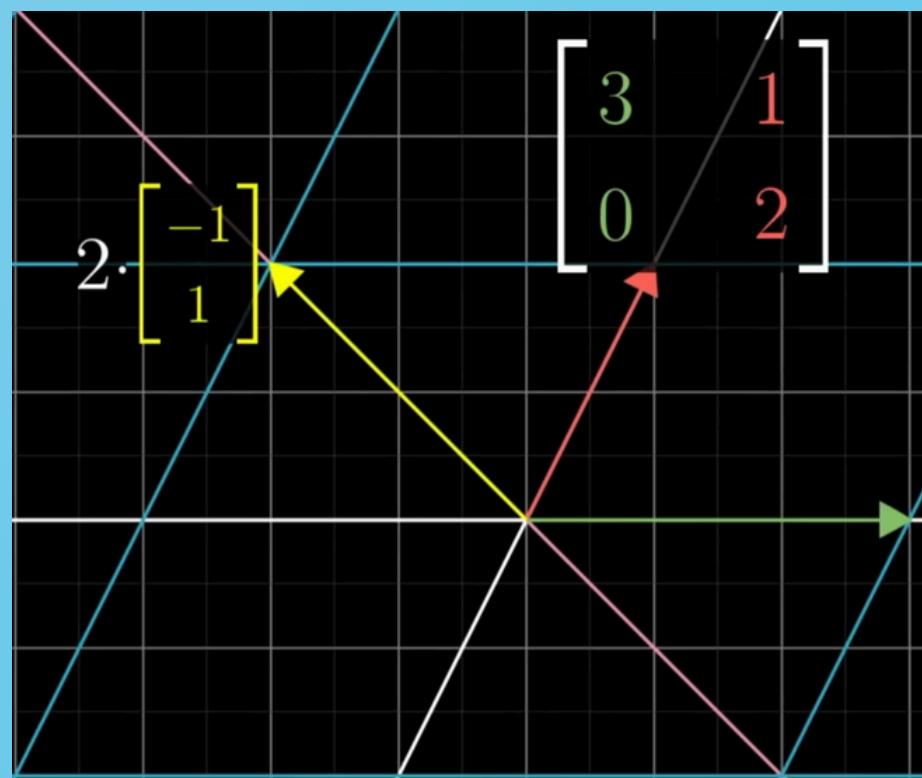
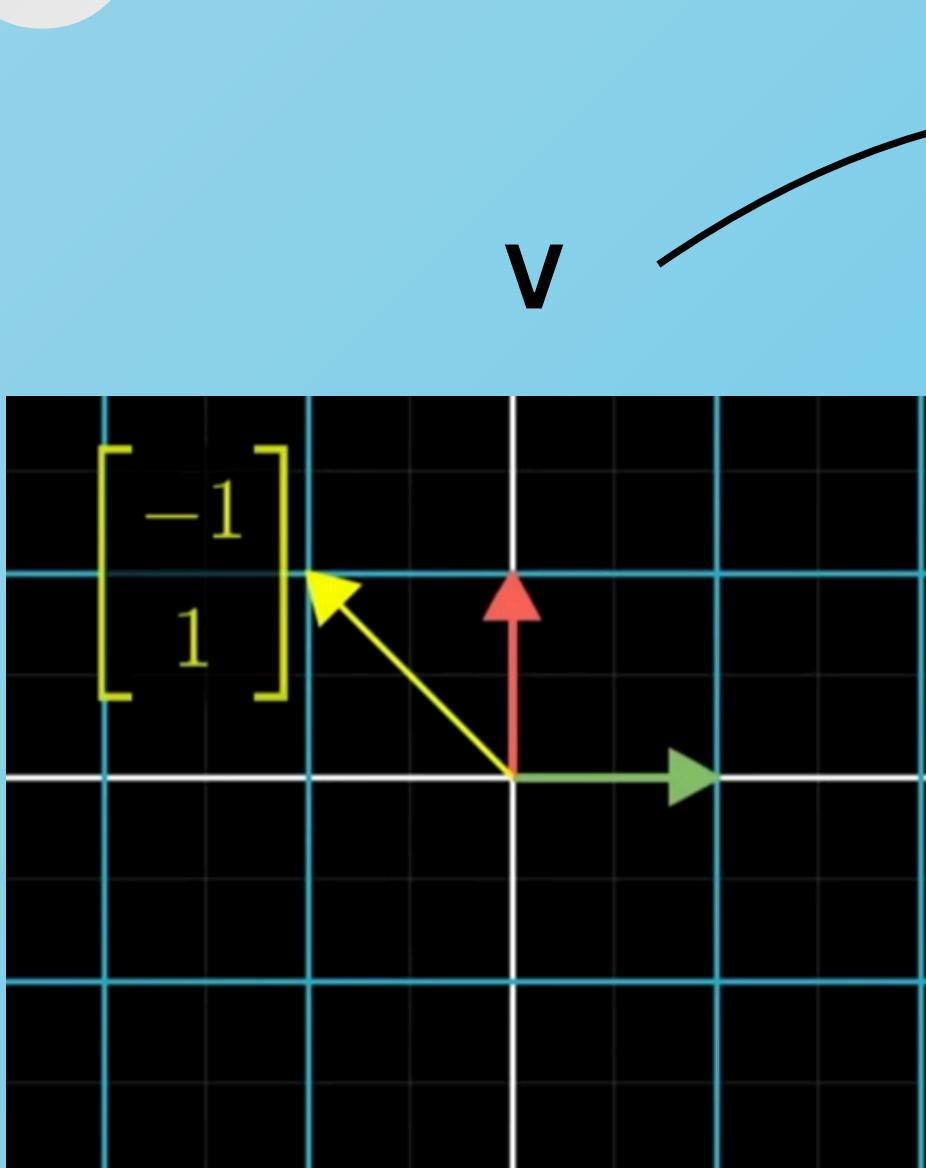
# TRANSFORMACIONES LINEALES

- Es una función que mapea vectores de un espacio vectorial a otro



# EIGENVALORES Y EIGENVECTORES

- Los **Eigenvectores** son vectores que cuando la matriz actúa sobre ellos, NO cambian de dirección; solo se estiran o encogen.
- Los **Eigenvalores** son los factores por los cuales los vectores propios son alargados o comprimidos durante una transformación lineal



$$\vec{Av} = \lambda \vec{v}$$

# RELACIÓN CON PCA

- Los Eigenvectores definen las direcciones fundamentales en las cuales ocurre el cambio de forma más simple.
- PCA consiste en encontrar las direcciones donde los datos se estiran más (mayor varianza). Y las direcciones de estiramiento de una matriz se obtienen resolviendo sus eigenvectores.
- Calcula la matriz de covarianza, esta matriz describe cómo se dispersan los datos en el espacio
- La matriz de covarianza tiene como eigenvectores las direcciones donde la varianza del dato es máxima o mínima, y sus eigenvalores indican cuánta varianza hay en esa dirección.

# ANALISIS DE COMPONENTES PRINCIPALES

1 Centrar los datos

2 Calcular matriz de covarianzas

3 Calcular eigenpares

4 Ordenarlos descentemente



## CLASE PCA\_propio:

# PSEUDOCÓDIGO

### ATRIBUTOS:

n\_pca: Entero (Número de componentes a mantener)  
scale\_data: Booleano (Si se debe estandarizar los datos)  
mean\_: Vector (Media de los datos originales)  
std\_: Vector (Desviación estándar de los datos originales)  
w\_: Matriz (Matriz de pesos/proyección con los eigenvectores seleccionados)  
var\_exp\_: Vector (Varianza explicada por cada componente)  
cum\_var\_exp\_: Vector (Varianza explicada acumulada)

### METODO CONSTRUCTOR(\_\_init\_\_):

ENTRADA: n\_pca (default=3), scale\_data (default=True)  
Guardar n\_pca en self.n\_pca  
Guardar scale\_data en self.scale\_data  
Inicializar resto de atributos como Nulo

### METODO FIT(X):

"""Calcula los parámetros del modelo (Eigenvectores y Eigenvalores)"""  
ENTRADA: Matriz X de dimensiones (muestras, características)

#### 1. PREPROCESAMIENTO:

Calcular la media de X por columna -> self.mean\_  
Centrar datos: X\_centrado = X - self.mean\_

##### SI self.scale\_data ES VERDADERO:

Calcular desviación estándar de X -> self.std\_  
Manejar división por cero (si std es 0, reemplazar por 1)  
X\_procesado = X\_centrado / self.std\_

##### SINO:

Establecer self.std\_ como vector de 1s  
X\_procesado = X\_centrado

#### 2. MATRIZ DE COVARIANZA:

Calcular matriz de covarianza de X\_procesado -> self.cov\_mat

**3. DESCOMPOSICIÓN EN EIGENVALORES (EIGENDECOMPOSITION):**

Obtener eigenvalores (valores propios) y eigenvectores (vectores propios) de self.cov\_mat

**4. ORDENAMIENTO:**

Emparejar cada eigenvalor con su eigenvector correspondiente

Ordenar los pares de mayor a menor basándose en el valor absoluto del eigenvalor

Separar los valores ordenados y vectores ordenados

**5. SELECCIÓN DE COMPONENTES (Matriz de Proyección):**

Determinar k = mínimo entre (self.n\_pca y número de columnas de X)

Seleccionar los primeros k eigenvectores ordenados

Guardar esta matriz reducida en self.w\_

**6. CÁLCULO DE VARIANZA EXPLICADA:**

Varianza total = Suma de todos los eigenvalores

self.var\_exp\_ = eigenvalores\_ordenados / Varianza total

self.cum\_var\_exp\_ = Suma acumulada de self.var\_exp\_

RETORNAR self (el objeto entrenado)

**METODO TRANSFORM(X):**

"""Proyecta los datos originales al nuevo espacio dimensional"""

ENTRADA: Matriz X (nuevos datos)

**SI self.w\_ ES NULO:**

Lanzar Error: "El modelo no ha sido entrenado (fit)"

**1. APLICAR PREPROCESAMIENTO GUARDADO:**

X\_procesado = (X - self.mean\_) / self.std\_

**2. PROYECCIÓN (Producto Punto):**

X\_transformado = ProductoPunto(X\_procesado, self.w\_)

RETORNAR X\_transformado

# MATRIZ DE COVARIANZA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1.000151	-0.009909	-0.016094	-0.005453	-0.006018	0.010978	0.024850	-0.010397	0.005039	-0.014284	0.000762	-0.003904	-0.000299	0.009700	0.004625	-0.014403	-0.007247	0.013565	0.004241	0.445522
1	-0.009909	1.000151	-0.009315	-0.011510	-0.002223	-0.015920	-0.020189	-0.005011	-0.018320	0.014326	-0.012776	-0.000965	0.018647	-0.007520	-0.022438	-0.022235	0.026240	-0.018745	-0.008042	0.581160
2	-0.016094	-0.009315	1.000151	-0.026561	-0.021885	-0.007946	-0.021077	-0.022397	0.012459	-0.002107	0.012385	0.012955	0.016738	0.021316	-0.005384	0.006340	-0.008209	-0.008252	0.016135	0.157138
3	-0.005453	-0.011510	-0.026561	1.000151	-0.008206	-0.014507	0.023868	0.006207	-0.013948	-0.011776	-0.004211	-0.010629	0.020596	-0.001247	-0.009998	-0.006103	-0.003865	-0.001958	0.000066	0.169796
4	-0.006018	-0.002223	-0.021885	-0.008206	1.000151	0.001379	0.004355	0.016118	-0.005346	0.003193	-0.009042	0.017639	-0.005656	0.009143	-0.001032	-0.008290	0.004454	0.013230	-0.007946	0.064391
5	0.010978	-0.015920	-0.007946	-0.014507	0.001379	1.000151	-0.021754	0.001031	0.011903	-0.012218	-0.018918	0.006829	-0.002776	-0.016869	-0.000378	0.015395	0.006216	-0.001739	0.007480	-0.017024
6	0.024850	-0.020189	-0.021077	0.023868	0.004355	-0.021754	1.000151	0.006461	0.004305	-0.013124	-0.013663	-0.003594	-0.013454	-0.017985	-0.011275	0.007058	-0.012757	-0.005941	-0.001021	0.175106
7	-0.010397	-0.005011	-0.022397	0.006207	0.016118	0.001031	0.006461	1.000151	0.019206	0.001259	0.011066	-0.012521	-0.013756	-0.000302	-0.007389	-0.000703	-0.006568	0.002267	-0.013984	0.086951
8	0.005039	-0.018320	0.012459	-0.013948	-0.005346	0.011903	0.004305	0.019206	1.000151	-0.010072	0.002343	0.000411	-0.012788	-0.010431	-0.012869	0.006518	-0.003590	0.010508	-0.015141	0.051483
9	-0.014284	0.014326	-0.002107	-0.011776	0.003193	-0.012218	-0.013124	0.001259	-0.010072	1.000151	0.005084	0.001464	-0.008259	-0.005140	0.017736	0.007565	0.002690	-0.017656	0.006735	0.156549
10	0.000762	-0.012776	0.012385	-0.004211	-0.009042	-0.018918	-0.013663	0.011066	0.002343	0.005084	1.000151	-0.005148	-0.014796	0.016662	-0.022823	0.016335	-0.001654	-0.010728	0.001277	0.094569
11	-0.003904	-0.000965	0.012955	-0.010629	0.017639	0.006829	-0.003594	-0.012521	0.000411	0.001464	-0.005148	1.000151	0.007582	-0.008461	-0.016046	-0.000576	-0.000169	0.006038	0.004968	0.076696
12	-0.000299	0.018647	0.016738	0.020596	-0.005656	-0.002776	-0.013454	-0.013756	-0.012788	-0.008259	-0.014796	0.007582	1.000151	-0.006720	0.009916	-0.000354	0.016642	-0.005741	-0.004178	0.008845
13	0.009700	-0.007520	0.021316	-0.001247	0.009143	-0.016869	-0.017985	-0.000302	-0.010431	-0.005140	0.016662	-0.008461	-0.006720	1.000151	-0.001973	-0.013478	0.001825	0.012401	-0.009352	0.100232
14	0.004625	-0.022438	-0.005384	-0.009998	-0.001032	-0.000378	-0.011275	-0.007389	-0.012869	0.017736	-0.022823	-0.016046	0.009916	-0.001973	1.000151	0.013164	-0.032214	-0.006059	-0.005211	0.027829
15	-0.014403	-0.022235	0.006340	-0.006103	-0.008290	0.015395	0.007058	-0.000703	0.006518	0.007565	0.016335	-0.000576	-0.000354	-0.013478	0.013164	1.000151	-0.012247	-0.001181	0.015182	-0.085079
16	-0.007247	0.026240	-0.008209	-0.003865	0.004454	0.006216	-0.012757	-0.006568	-0.003590	0.002690	-0.001654	-0.000169	0.016642	0.001825	-0.032214	-0.012247	1.000151	0.007006	0.002294	0.103496
17	0.013565	-0.018745	-0.008252	-0.001958	0.013230	-0.001739	-0.005941	0.002267	0.010508	-0.017656	-0.010728	0.006038	-0.005741	0.012401	-0.006059	-0.001181	0.007006	1.000151	-0.000886	-0.088948
18	0.004241	-0.008042	0.016135	0.000066	-0.007946	0.007480	-0.001021	-0.013984	-0.015141	0.006735	0.001277	0.004968	-0.004178	-0.009352	-0.005211	0.015182	0.002294	-0.000886	1.000151	0.002033
19	0.445522	0.581160	0.157138	0.169796	0.064391	-0.017024	0.175106	0.086951	0.051483	0.156549	0.094569	0.076696	0.008845	0.100232	0.027829	-0.085079	0.103496	-0.088948	0.002033	1.000151

# VECTORES PROPIOS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-0.370317	-0.490099	-0.114101	-0.133276	-0.050120	0.031172	-0.143185	-0.066019	-0.031292	-0.132096	-0.073785	-0.060612	-0.019608	-0.085788	-0.002237	0.091583	-0.096950	0.082797	0.002344	-0.709647
1	-0.318686	0.271479	0.398525	-0.260927	-0.146172	-0.010743	-0.484622	-0.293787	-0.178520	0.215883	0.066949	0.131625	0.241366	0.066356	-0.007804	0.020416	0.199060	-0.177246	0.143509	-0.011247
2	0.255401	-0.268795	0.428380	-0.224606	-0.208968	0.029807	0.037950	-0.007994	0.273865	0.065063	0.383308	-0.005577	-0.262452	0.184557	0.122269	0.306214	-0.332806	-0.046154	0.174544	0.035986
3	0.040599	-0.008054	-0.080174	0.230926	-0.226819	-0.057213	0.202916	-0.195208	-0.293837	0.213399	-0.204839	-0.159220	0.188979	-0.276600	0.477562	0.206024	-0.295213	-0.330261	0.162221	0.008387
4	0.354431	-0.063822	0.100664	-0.138793	-0.022481	0.475127	0.025559	-0.320470	0.137105	-0.271670	-0.378277	0.310050	0.191459	-0.213050	0.018381	0.038475	0.086144	0.212906	0.206645	0.012619
5	0.120266	-0.163153	0.213384	0.361631	-0.260578	-0.333892	0.102629	-0.302263	-0.202155	-0.401862	0.089894	-0.098801	0.273252	0.345708	-0.149218	-0.196016	0.012206	0.152641	0.050444	-0.016207
6	0.198991	-0.057168	-0.019580	-0.234079	0.293323	-0.050070	-0.241879	-0.074657	-0.175045	0.097238	-0.215702	-0.051581	-0.018249	0.421045	0.515821	-0.251271	-0.203625	0.241502	-0.231157	0.002819
7	0.025230	0.120506	0.067079	-0.023949	-0.417479	0.194866	-0.211453	0.160730	0.394699	-0.213879	-0.088127	-0.435823	0.221491	-0.056411	0.141988	-0.099048	0.006066	-0.135392	-0.444005	0.001251
8	0.247812	0.010692	-0.324159	-0.050417	-0.191127	0.285724	-0.202416	-0.067821	-0.282127	0.050632	0.154030	-0.497368	-0.257393	0.110934	-0.039435	0.022910	0.323866	0.095976	0.345050	-0.007556
9	0.016818	0.114220	-0.004711	-0.376844	-0.246323	-0.260460	0.218777	-0.345925	0.040556	0.001157	-0.209415	-0.054849	-0.460049	-0.181219	-0.149949	-0.450589	-0.085078	-0.143112	-0.041151	-0.025946
10	0.054135	-0.056250	-0.049174	0.060894	0.114778	0.409114	-0.119317	0.047886	-0.294488	-0.308558	0.174110	0.195702	-0.088688	0.118322	-0.114669	-0.258537	-0.275538	-0.597024	-0.076915	-0.004969
11	0.204945	-0.027933	-0.285451	-0.206055	-0.007823	-0.071068	0.083187	-0.402840	-0.126924	-0.007552	0.393351	0.144925	0.105902	-0.100917	0.005163	0.317585	0.178900	-0.031506	-0.558270	-0.010930
12	-0.113197	0.027348	0.220049	-0.282645	0.395691	0.000817	0.380474	-0.044509	-0.030089	-0.267752	-0.209571	-0.380940	0.047142	0.223647	0.002469	0.286169	0.256335	-0.304350	0.026589	0.000984
13	-0.040363	0.451883	-0.049036	-0.251136	-0.024391	-0.148219	-0.103356	0.181104	-0.269820	-0.485058	0.080094	-0.048749	-0.008634	-0.194174	-0.014961	0.244718	-0.398239	0.283626	0.093976	0.004422
14	0.245615	-0.221719	-0.016959	-0.316817	0.183620	-0.252884	-0.017695	0.224912	0.033478	-0.043511	0.304487	-0.087503	0.408219	-0.354402	0.118399	-0.365644	0.124646	-0.124004	0.258252	-0.018451
15	-0.066466	0.132699	-0.052555	0.282157	0.464223	-0.011557	-0.244887	-0.501908	0.383900	-0.067612	0.193946	-0.282724	-0.058894	-0.174854	-0.004175	-0.041771	-0.222561	-0.005086	0.114662	0.008526
16	-0.095976	0.062601	-0.223444	0.090890	-0.109756	-0.239946	-0.127022	0.015204	0.240942	-0.392504	0.031162	0.317390	-0.257131	0.110995	0.502579	0.061035	0.345204	-0.195533	0.191395	0.009859
17	-0.134365	-0.066357	0.480141	0.196089	0.087166	0.180183	0.094169	0.027976	-0.281568	-0.111418	0.198013	-0.064883	-0.312530	-0.393852	0.306417	-0.151136	0.246703	0.219198	-0.211080	0.027132
18	0.401560	-0.177716	0.178889	0.145244	0.111320	-0.345024	-0.463133	0.110994	-0.115251	-0.034321	-0.339285	-0.028556	-0.207005	-0.211955	-0.219455	0.253512	0.090939	-0.197646	-0.133691	-0.004738
19	0.373869	0.486395	0.149434	0.153302	0.058420	0.002160	0.149691	0.079012	0.053099	0.129448	0.084531	0.067786	-0.004763	0.081923	0.044309	-0.053100	0.079193	-0.065581	0.004759	-0.701650

Variable	Frecuencia	Componentes donde domina
Physical_Activity	3	4, 7, 17
Exam_scores	2	1, 20
Parental_Involvement	2	3, 18
Tutoring_Sessions	2	6, 14
Gender	2	8, 12
School_Type	2	10, 15
Previous_Scores	1	2
Sleep_Hours	1	5
Teacher_Quality	1	9
Distance_from_Home	1	11
Extracurricular_Activities	1	13
Motivation_Level	1	16
Hours_Studied	1	19

# COMPARACIÓN CON SKLEARN

Comparación de Varianza Explicada

Clase propia: [0.09176634 0.05444912 0.05331813]

Scikit-Learn: [0.09176634 0.05444912 0.05331813]

---

Comparación de la primera fila de datos

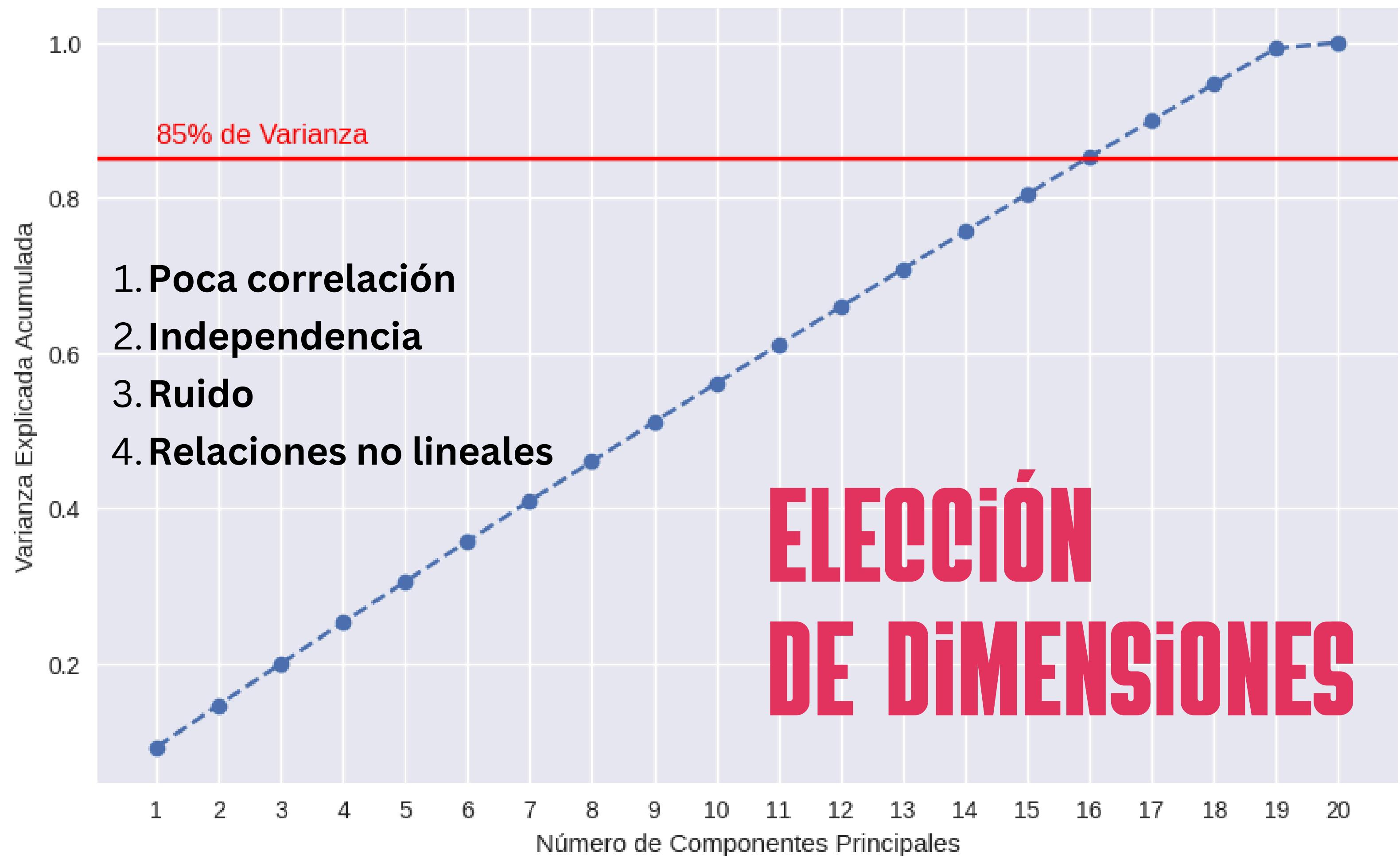
Clase propia: [ 0.02257582 -1.08952942 -0.61299543 0.77005361 0.13810861 1.29015085  
0.43234301 1.15913861 0.38820034 1.02555959]

Scikit-Learn: [-0.02257582 1.08952942 -0.61299543 0.77005361 0.13810861 -1.29015085  
0.43234301 -1.15913861 -0.38820034 -1.02555959]

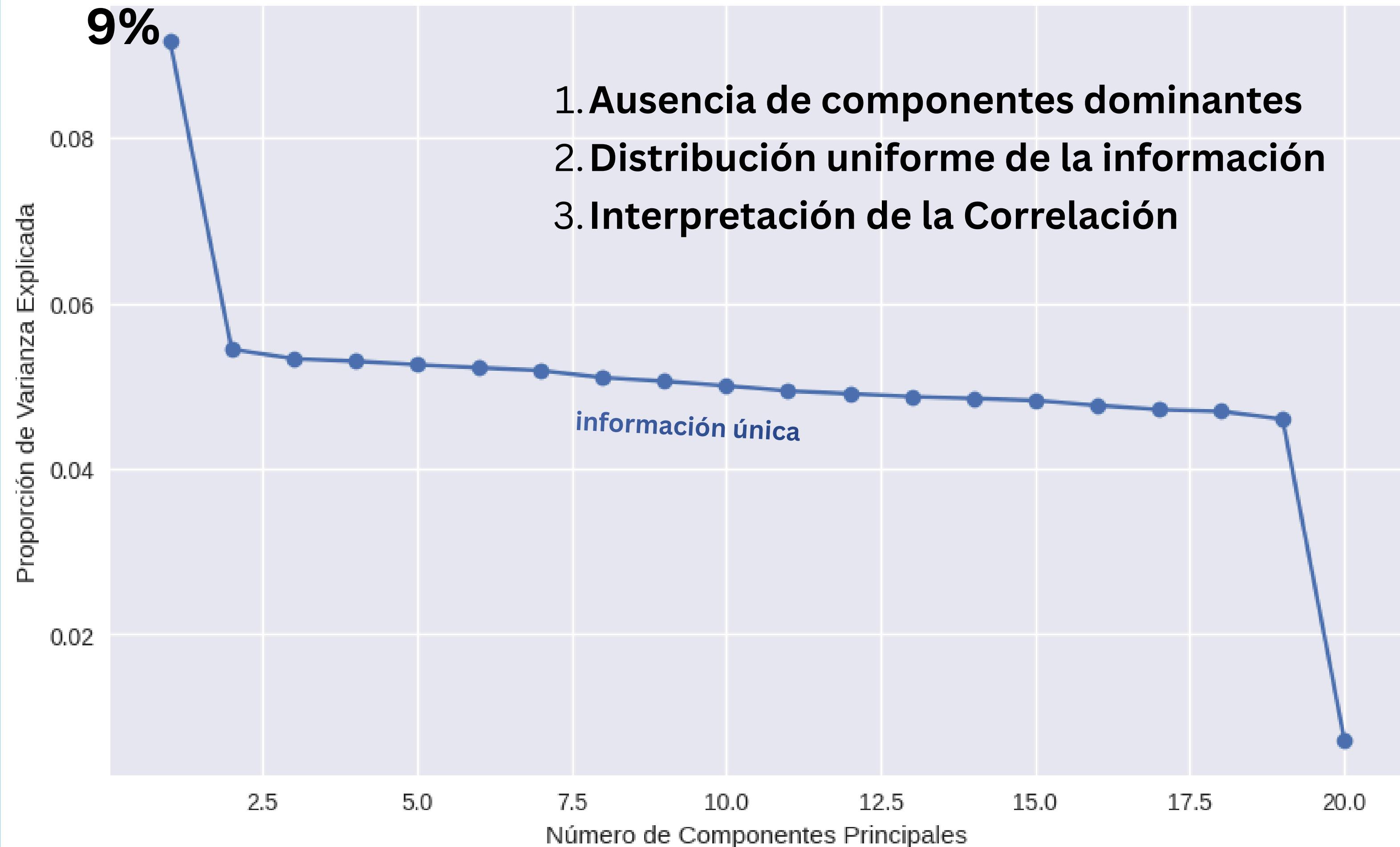
Los resultados coinciden

$$Av = \lambda v \Leftrightarrow A(-v) = \lambda(-v)$$

Gráfico de Codo de Varianza Acumulada (PCA)



## Gráfico de Varianza Explicada Individual



# INTERPOLACIÓN

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{pmatrix}$$

La interpolación parte del conjunto de datos discreto original. A partir de la matriz de vandermonde, definida como:

$$V_{ij} = x_i^{j-1}$$

Podemos plantear el siguiente sistema:

$$V\vec{a} = \vec{y}$$

Replantable y solucionable a través de la factorización de Cholesky como:

$$V'\vec{a} = V^t V\vec{a} = V^t \vec{y} = b$$



# ALGORÍTMOS PREVIOS

Matriz de Vandermonde

Factorización de Cholesky

Interpolación



```
def matriz_vandermonde(vector_x, grado):
    """matriz_vandermonde
    Funcion auxiliar que construye la matriz de Vandermonde manualmente.
    Filas: m (numero de datos)
    Columnas: n + 1 (grado del polinomio + termino independiente)
    ---
    params:
        vector_x: vector x con la que la construiremos
        grado: el grado de la matriz a contruir
    ---
    return:
        V: matriz de Vandermonde
    """
```

```
def Cholesky(A):
    '''Cholesky
    Funcion optimizada que simula el metodo de cholesky, donde A = L·L^T
    ---
    param:
        A: Matriz cuadrada y simetrica
    ---
    return:
        L: Matriz triangular inferior
    ...
```

```
def Ux_b(U, b):
    '''Ux_b
    Funcion que resuelve un sistema Ux=b, donde U es matriz triangular superior.
    ---
    params:
        U: matriz triangular superior
        b: vector columna b
    ---
    return:
        x: vector solucion al sistema Ux=b
    ...
```

```
def Lx_b(L, b):
    '''Lx_b
    Funcion que resuelve un sistema Lx=b, donde L es matriz triangular inferior.
    ---
    params:
        L: matriz triangular inferior
        b: vector columna b
    ---
    return:
        x: vector solucion al sistema Lx=b
    ...
```

```
def interpolacion(x, y, grado):
    """interpolacion
    Funcion que implementa la interpolacion.
    ---
    params:
        x: Vector de variables independientes. Dimension (m,).
        y: Vector de variables dependientes (valores a predecir). Dimension (m,).
        grado: Grado del polinomio deseado (n).
    ---
    return:
        a: Vector de coeficientes optimos del polinomio. Dimension (n+1,).
        norma_error: El valor escalar del error residual minimo: ||Va - y||.
    """
    # Construimos V
    V = matriz_vandermonde(x, grado)

    # Resolvemos la Ecuación Normal (V^T V)a = V^T
    A_sistema = V.T @ V # Esta matriz es simetrica y def. positiva
    b_sistema = V.T @ y

    # a son los coeficientes y es el vector que buscamos
    # resolvemos con cholesky
    a = Ax_b_cholesky(A_sistema, b_sistema)

    # Calculamos el error ||Va - y||
    # y calculamos la prediccion con los puntos originales
    y_prediccion_puntos = V @ a

    # El residuo es la diferencia vector real - vector estimado
    residuo = y_prediccion_puntos - y

    norma_error = np.sqrt(np.sum(residuo**2))

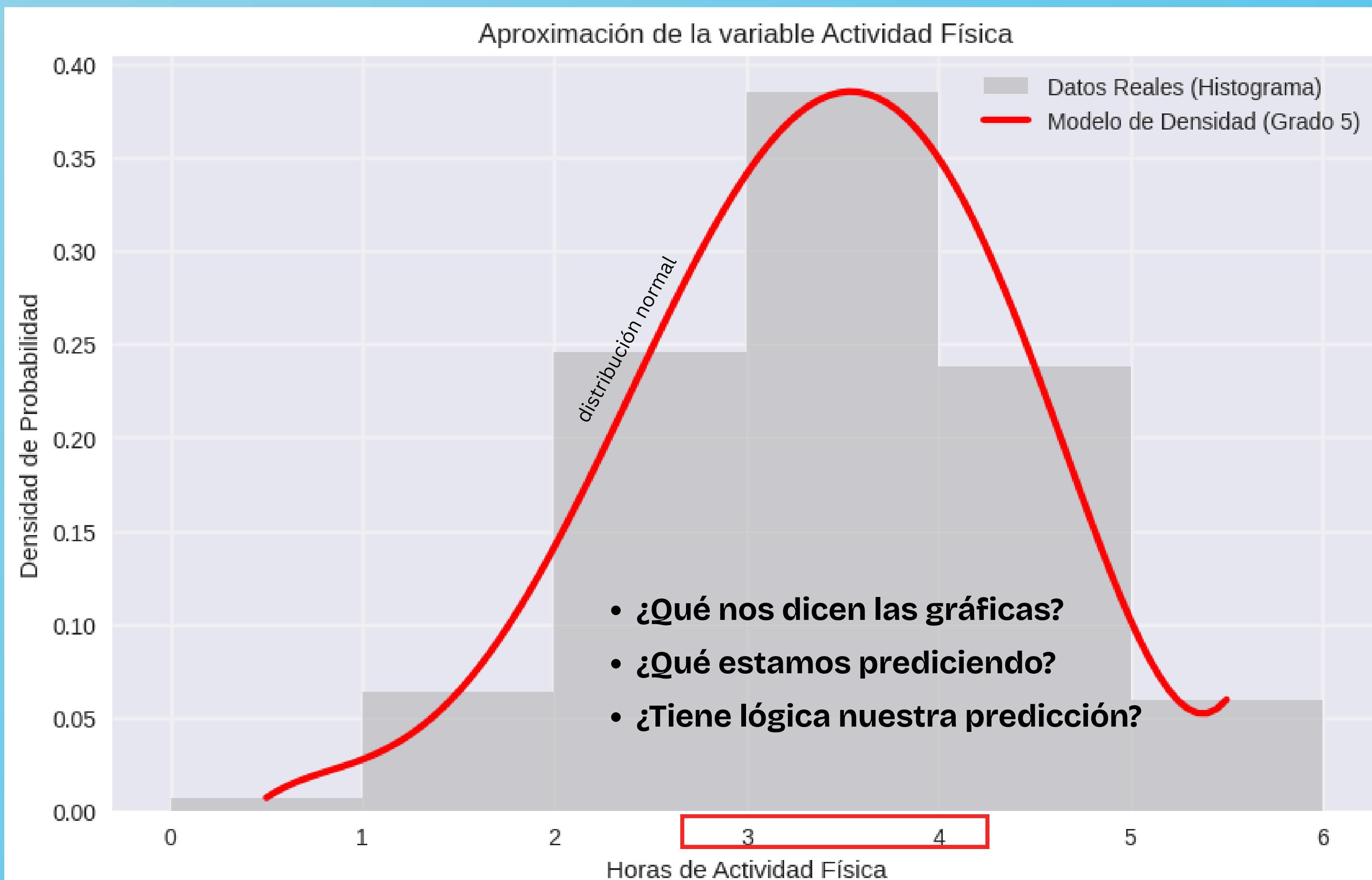
    return a, norma_error
```

# INTERPOLACIÓN

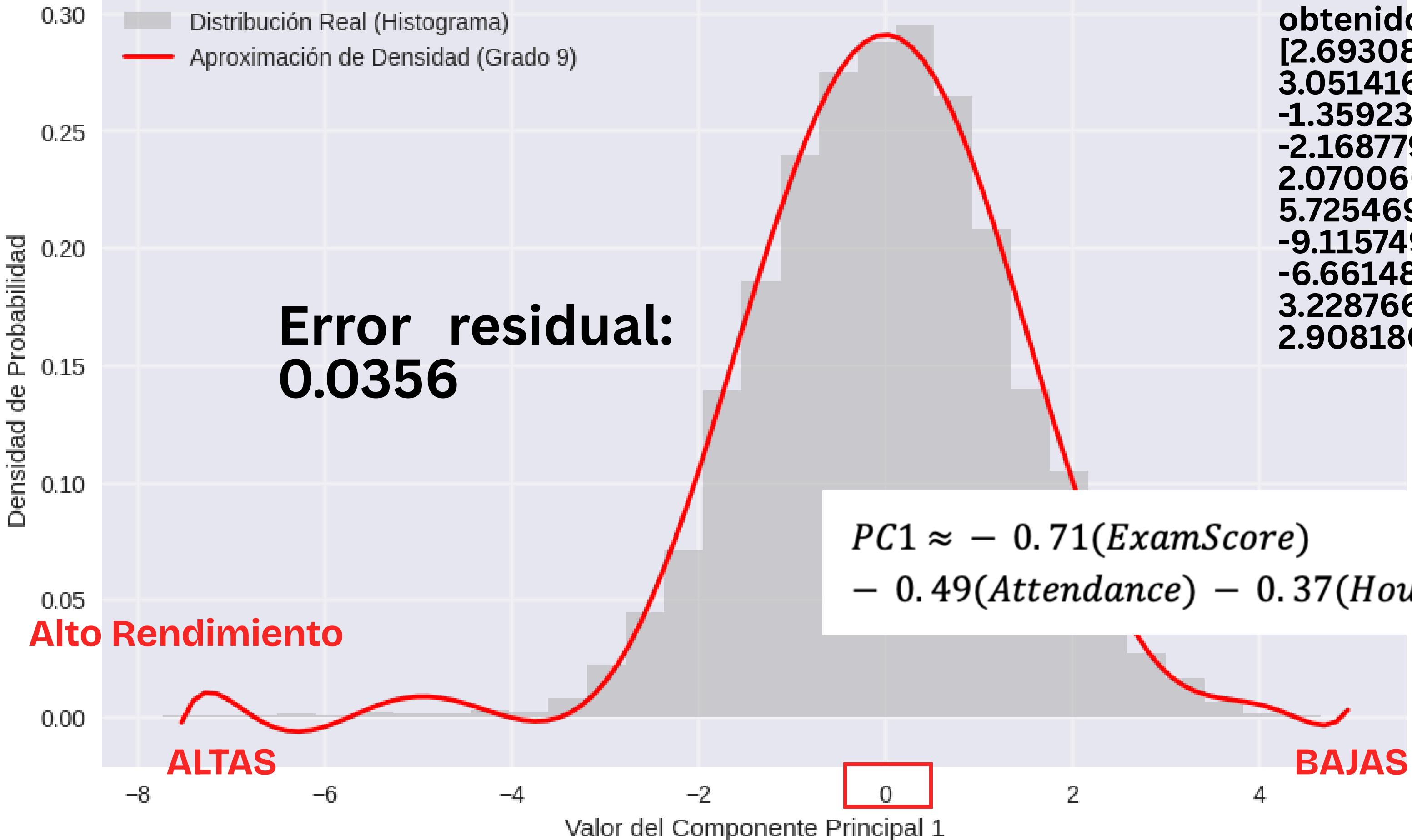
- Coeficientes obtenidos ( $a$ ):  
[31.30833333  
-411.39583329  
1774.06249975  
-2894.01041606  
2217.52760354  
-536.28515606]

Error residual:  
0.0000

**SOBREAJUSTE**



## Aproximación de la Densidad del Primer Componente Principal (PC1)



Coeficientes obtenidos (a):  
[2.69308072e-07  
3.05141696e-06  
-1.35923239e-05  
-2.16877964e-04  
2.07006021e-04  
5.72546978e-03  
-9.11574987e-04  
-6.66148998e-02  
3.22876655e-04  
2.90818606e-01]

# EXTRAPOLACIÓN

## DEF.

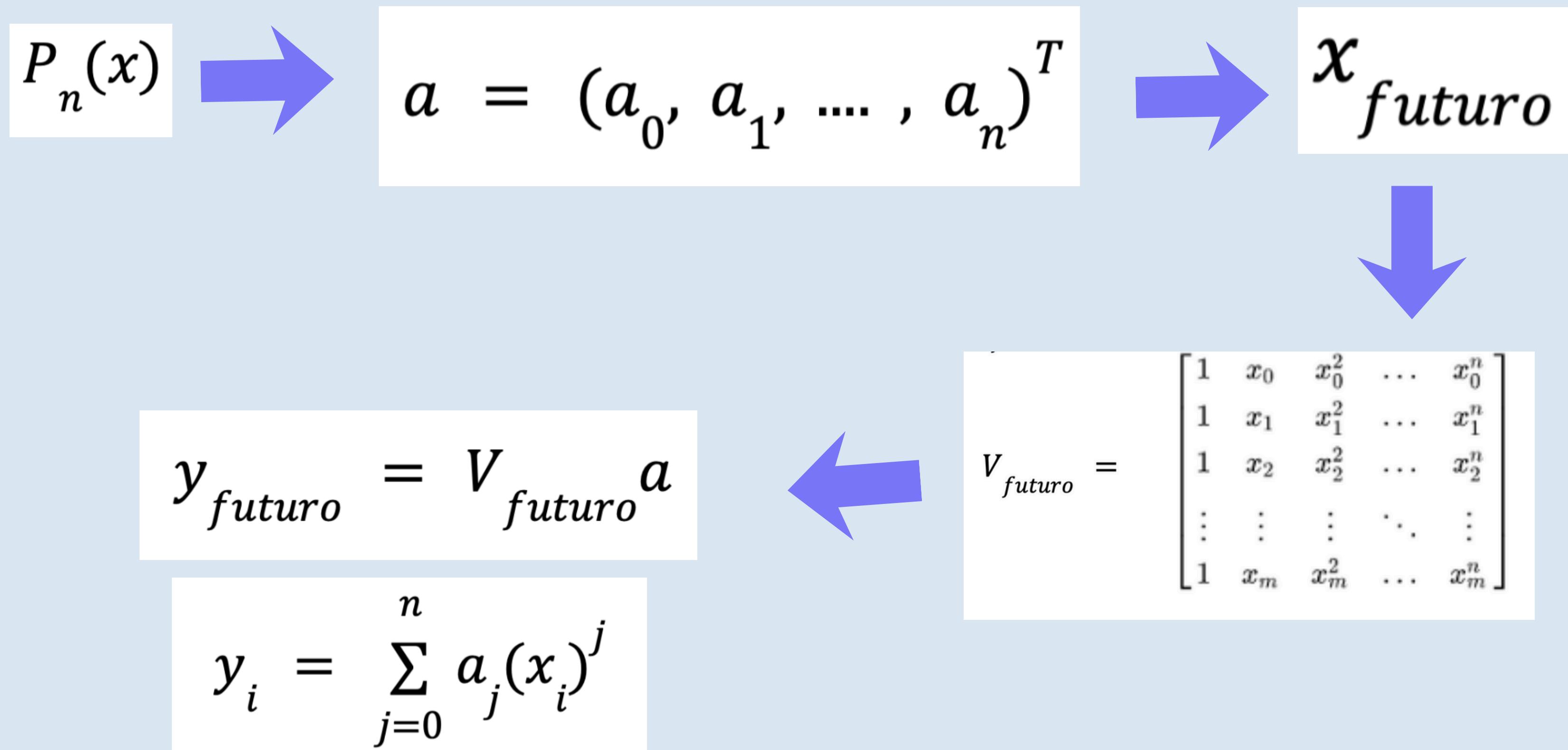
**Estimar** más allá del intervalo de observación original el valor de la variable, con base en su relación con otra variable

## Limitaciones

**Grados grandes → sobreajustes**

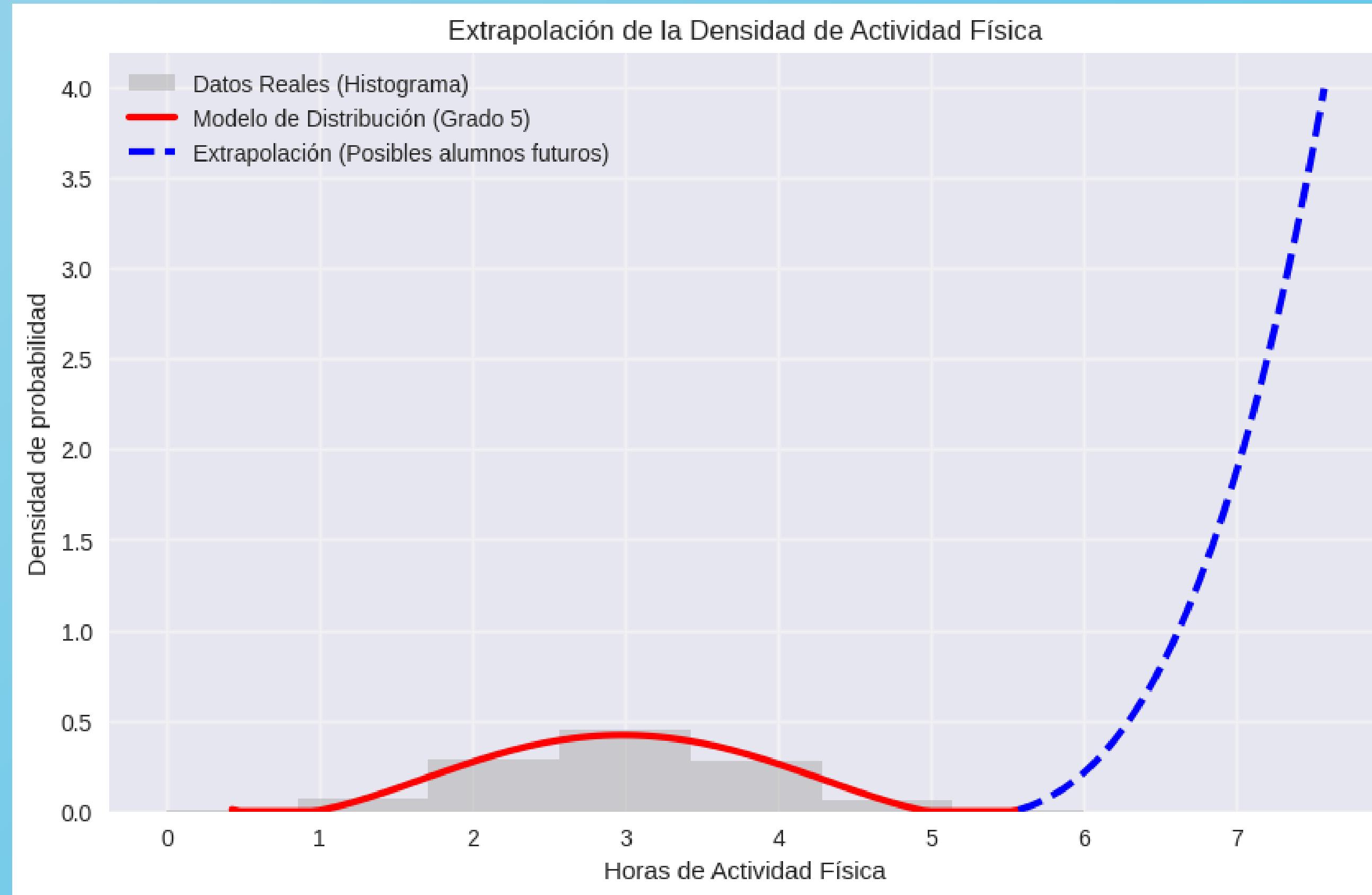
**Interpretaciones erróneas**

- ¿En qué consiste?

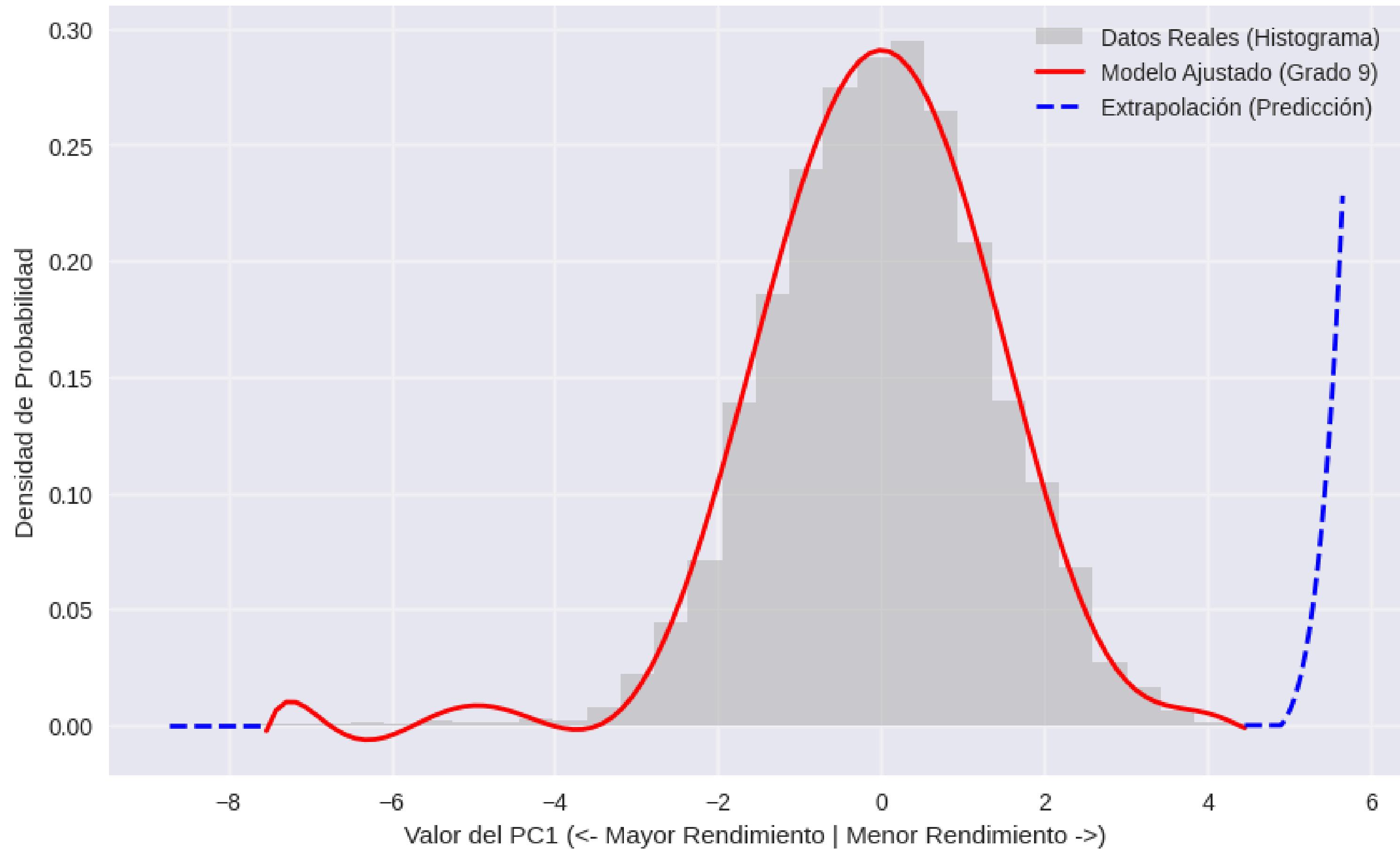


```
1 # primero tenemos que calcular la extrapolacion
2 def extrapolacion(x,coeficientes):
3     """extrapolacion
4     Funcion que implementa el metodo de extrapolacion, prediciendo
5     valores para un rango fuera del original.
6         y_predecidos = V_nuevo * a
7     ---
8     params:
9         x: vector de rango a predecir
10        coeficientes: vector de coeficientes encontrados con la interpolacion
11    return:
12        y_predecidos: vector de valores predecidos.
13    """
14    # Deducimos el grado del polinomio basado en cuantos coeficientes tenemos
15    # Si coeficientes tiene longitud n+1, el grado es n
16    grado = len(coeficientes) - 1
17
18    # Construimos la matriz de Vandermonde para los nuevos puntos
19    V_nuevos = matriz_vandermonde(x, grado)
20
21    # Evaluamos el polinomio usando multiplicacion matricial
22    y_predichos = V_nuevos @ coeficientes
23
24    return y_predichos
```

# EXTRAPOLACIÓN



## Extrapolación de la Densidad del Primer Componente Principal



# INTEGRAL NÚMÉRICA PARA OBTENER LA DENSIDAD Y LA CFD

Implementación: Sumas de Riemann

Método: Aproximación del área bajo la curva mediante particiones rectangulares

Aplicación: Integraremos la función de densidad interpolada  $f(x)$  para construir la Función de Distribución Acumulada (CDF).

Objetivo: Calcular probabilidades exactas  $P(X \leq x)$



## ALGORiTMOS UTILiZADOS

### Integral de Riemman

### Integral de Riemman acumulada

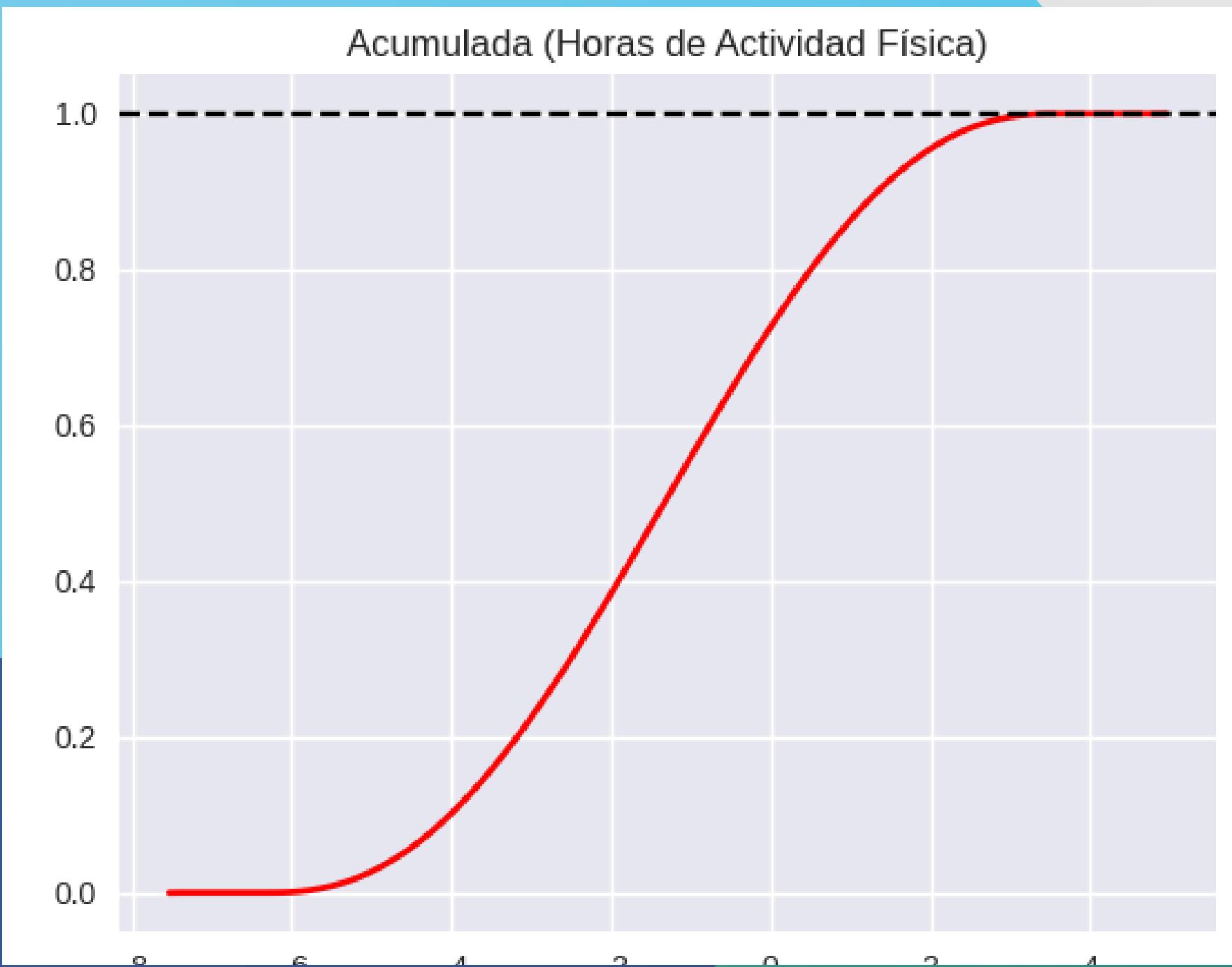
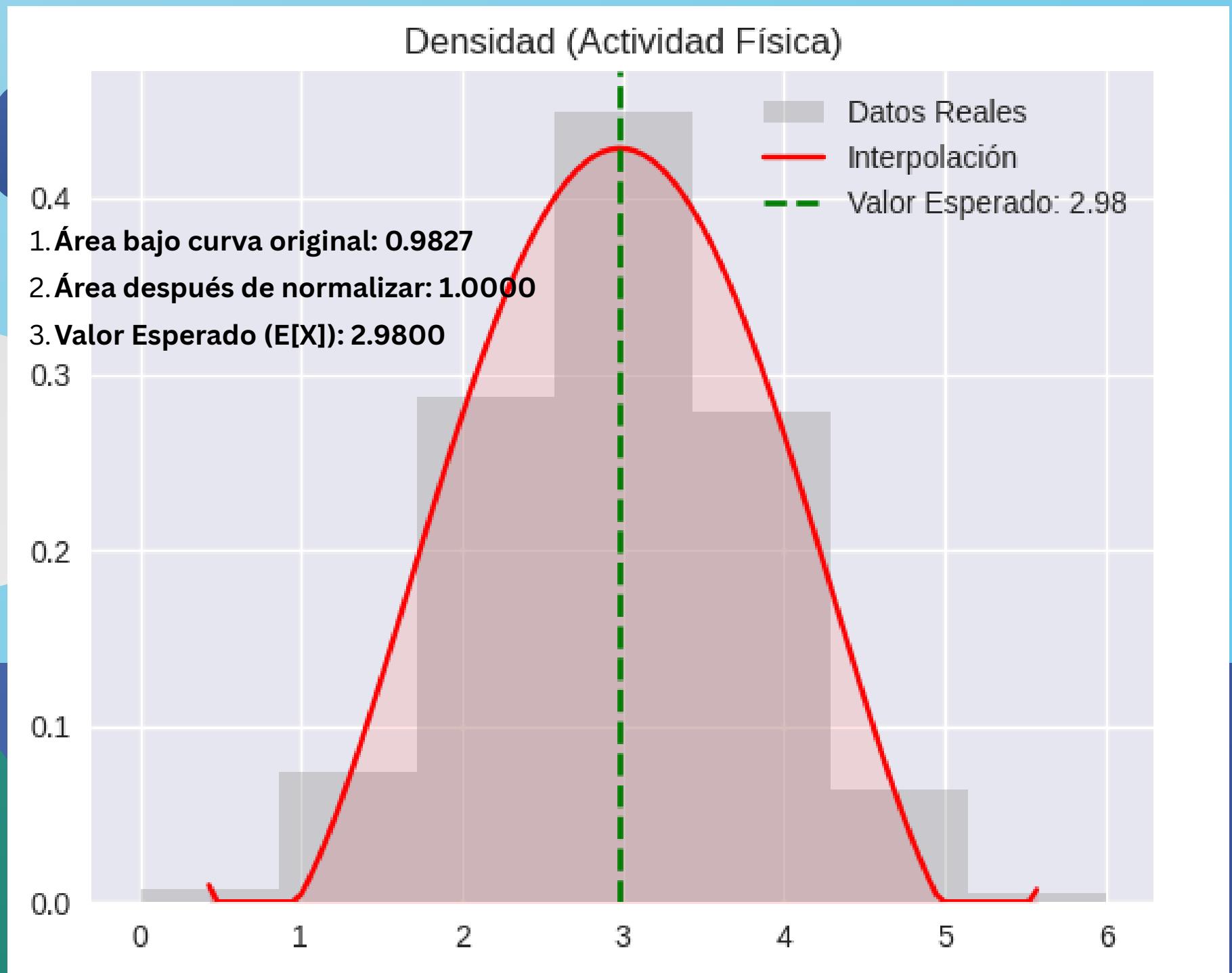
$$F(x_k) = \frac{b-a}{n} \left( \sum_{i=0}^{k-1} f\left(\frac{x_i + x_{i+1}}{2}\right) \right)$$

# ALGORITMO

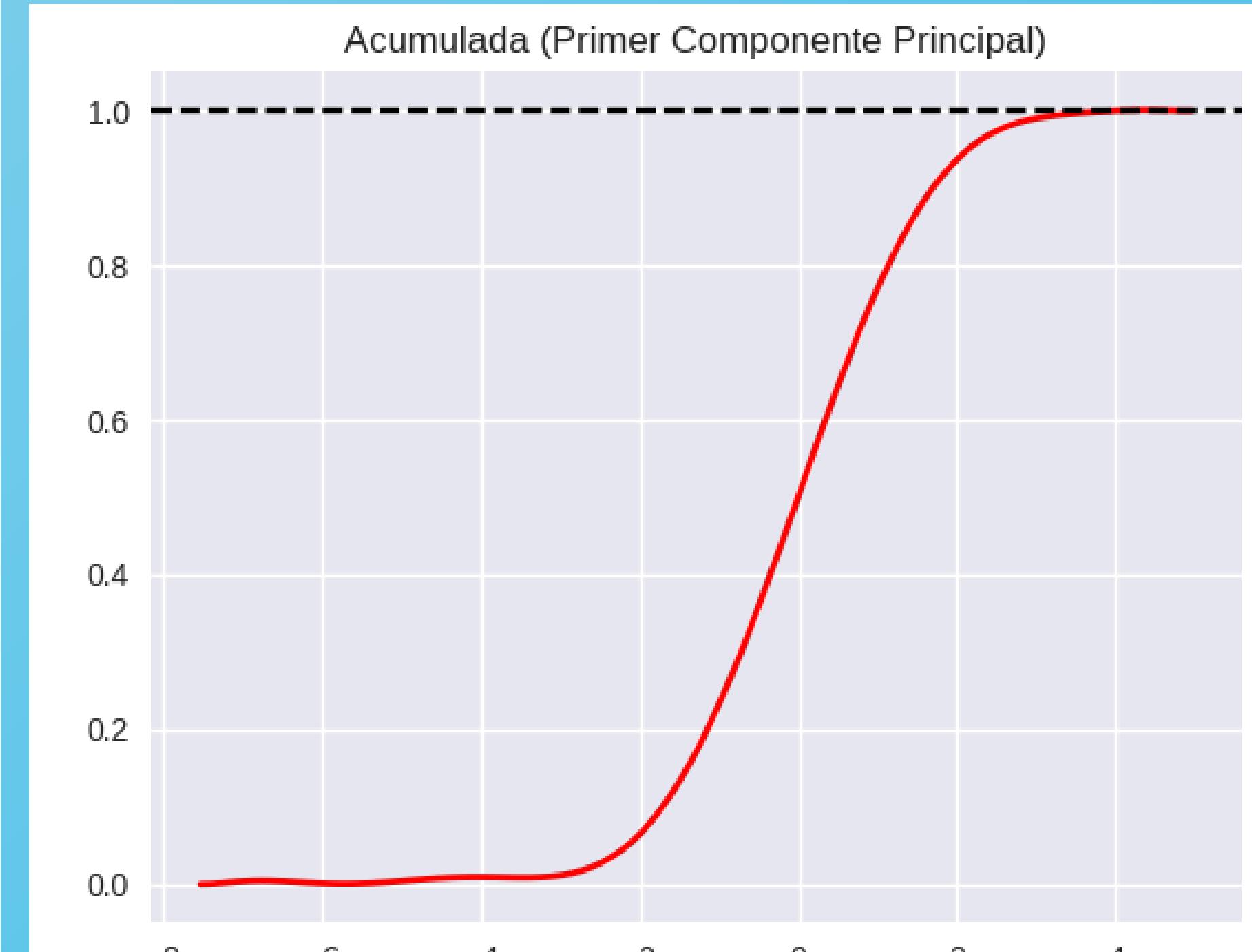
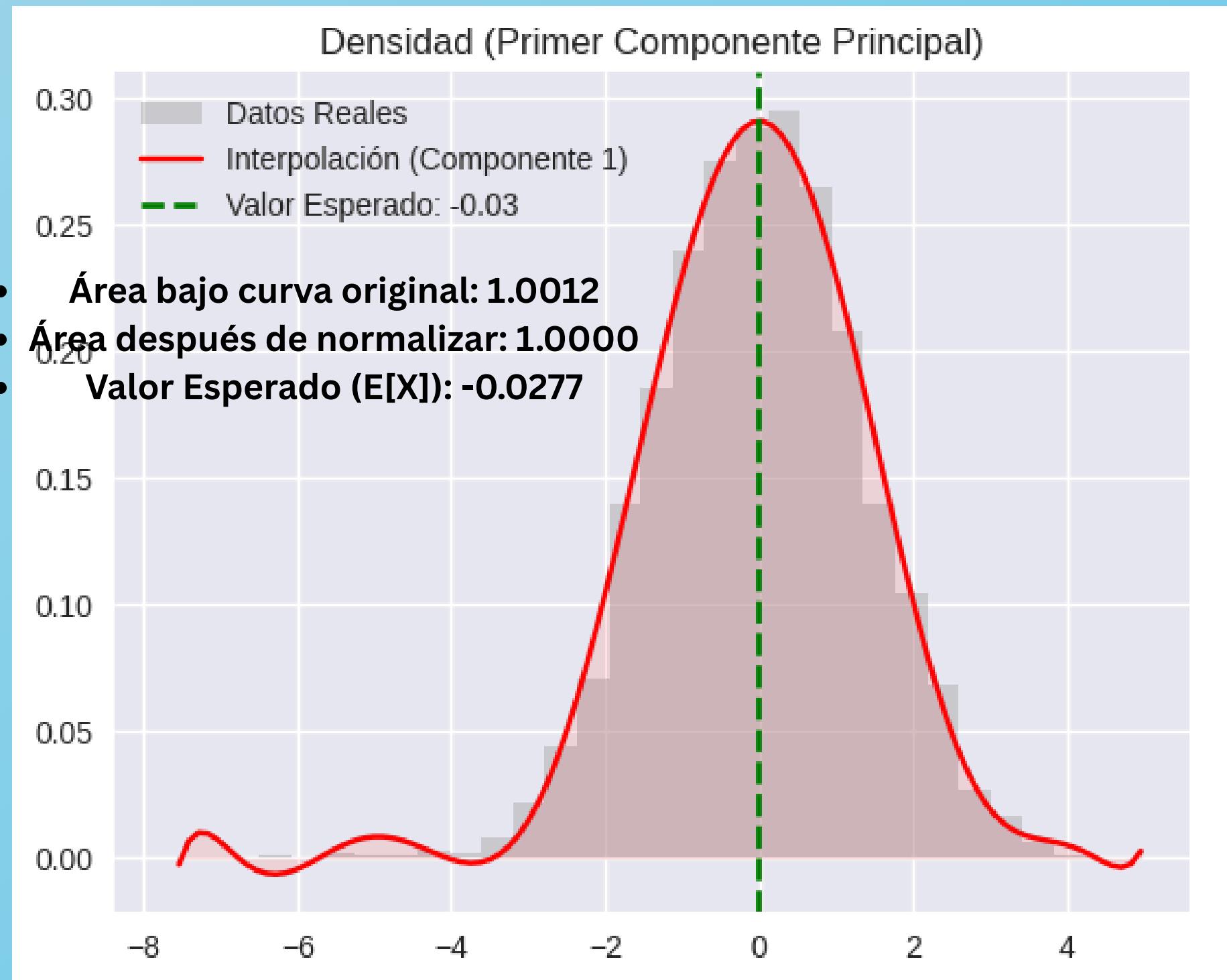
```
1 # usaremos la integral de riemann
2 def integral_riemann(x, y):
3     """integral_riemann
4     Calcula el area bajo la curva utilizando Sumas de Riemann con la
5     Regla del Punto Medio. Aproxima la altura del rectangulo promediando los
6     valores de y en los extremos del intervalo.
7     ---
8     params:
9         x: Vector de variables independientes (eje horizontal). Dimension (m,).
10        Se asume que estan ordenados ascendentemente.
11        y: Vector de variables dependientes (alturas de la funcion). Dimension (m,).
12    ---
13    return:
14        area: Valor flotante que representa el area total bajo la curva.
15    """
16    area = 0.0
17    # Iteramos sobre los intervalos
18    for i in range(len(x) - 1):
19        # Base del rectangulo (dx)
20        dx = x[i+1] - x[i]
21
22        # Altura del rectangulo (Regla del Punto Medio: promedio de y[i] y y[i+1])
23        altura_media = (y[i] + y[i+1]) / 2
24
25        # Area = base * altura
26        area += altura_media * dx
27
28    return area
```

```
1 def integral_acumulada_riemann(x, y):
2     """integral_acumulada_riemann
3     Calcula la funcion de distribucion acumulada (CDF) paso a paso utilizando
4     Sumas de Riemann con la Regla del Punto Medio.
5     ---
6     params:
7         x: Vector de variables independientes. Dimension (m,).
8         y: Vector de densidades de probabilidad (PDF). Dimension (m,).
9             Nota: 'y' debe estar normalizado.
10    ---
11    return:
12        y_cdf: Vector de probabilidades acumuladas. Dimension (m,).
13            y_cdf[i] representa la probabilidad acumulada hasta x[i].
14    """
15    y_cdf = np.zeros_like(y)
16    acumulado = 0.0
17
18    for i in range(1, len(x)):
19        dx = x[i] - x[i-1]
20
21        # Altura estimada en el punto medio del intervalo
22        altura_media = (y[i] + y[i-1]) / 2
23
24        area_rectangulo = altura_media * dx
25
26        acumulado += area_rectangulo
27        y_cdf[i] = acumulado
28
29    return y_cdf
```

# INTEGRAL NUMÉRICA [ACTIVIDAD FÍSICA]



# INTEGRAL NUMÉRICA [COMPONENTE PRINCIPAL]



- A partir de PCA concluimos que el rendimiento académico tiene causas multifactoriales.
- Se profundizó en la comprensión de los métodos matemáticos utilizados.
- Los métodos implementados nos permitieron transformar datos en modelos capaces de predecir tendencias.
- La extrapolación demostró ser útil, pero debe usarse con cautela.
- Proporcionamos un marco analítico para identificar patrones de éxito o riesgo para ayudar a fundamentar la toma de decisiones de instituciones académicas.



## REFLEXIÓN

# GRACIAS

