**# =============================================**

**# 1. CONFIGURACIÓN INICIAL**

**# =============================================**

**import pandas as pd**

**import numpy as np**

**import sqlite3**

**from sklearn.model\_selection import train\_test\_split, cross\_val\_score**

**from sklearn.preprocessing import StandardScaler, OneHotEncoder**

**from sklearn.compose import ColumnTransformer**

**from sklearn.pipeline import Pipeline**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error**

**# Ruta al DW**

**db\_path = "C:/Users/Nico/Desktop/DATA SCIENCE/PP- VOLUNTAREADO/chivas-ml/data/external/chivas\_dw.sqlite"**

**# Conexión**

**conn = sqlite3.connect(db\_path)**

**# Cargar la vista combinada**

**df = pd.read\_sql\_query("SELECT \* FROM vw\_entrenamiento\_sugerencias\_puras", conn)**

**df\_jugadores = pd.read\_sql\_query("SELECT \* FROM DB\_Jugadores", conn)**

**microciclos = pd.read\_sql("SELECT \* FROM DB\_MicrociclosExcel", conn)**

**# Cerrar conexión**

**conn.close()**

**print(f"Registros cargados: {len(df)}")**

**df.head()**

# =============================================

# 2. LIMPIEZA Y PREPROCESAMIENTO

# =============================================

# Eliminamos jugadores o fechas nulas

df = df.dropna(subset=["id\_jugador", "Fecha"])

# Convertimos fecha a datetime

df["Fecha"] = pd.to\_datetime(df["Fecha"])

# Reemplazamos nulos en Intensidad\_plan con 0 (ya que los partidos quedan en NULL)

df["Intensidad\_plan"] = df["Intensidad\_plan"].fillna(0)

# Codificamos columnas categóricas

df["Tipo\_Dia"] = df["Tipo\_Dia"].astype(str)

df["Tipo\_Microciclo"] = df["Tipo\_Microciclo"].astype(str)

df["nivel\_riesgo"] = df["nivel\_riesgo"].astype(str)

# Eliminamos outliers extremos en cargas (opcional)

#for col in ["Distancia\_total", "HMLD\_m", "HSR\_abs\_m", "Sprints\_cantidad"]:

# df = df[df[col] < df[col].quantile(0.99)]

# =============================================

# 2.1 FILTRO DE JUGADORES Y MICRO-CICLOS

# =============================================

# Jugadores que deben excluirse

jugadores\_excluir = [1, 2, 3, 12, 30]

# Filtrar jugadores no válidos

df = df[~df["id\_jugador"].isin(jugadores\_excluir)]

# Filtrar solo microciclos de competencia

df = df[df["Tipo\_Microciclo"].str.lower().str.contains("competencia")]

print(f"✅ Registros después del filtrado: {len(df)}")

print(f"Jugadores únicos: {df['id\_jugador'].nunique()}")

df[["id\_jugador", "Tipo\_Microciclo"]].drop\_duplicates().head()

# =============================================

# 🔹 Análisis inicial del dataset

# =============================================

import matplotlib.pyplot as plt

import seaborn as sns

# Verificar tipos de datos y valores nulos

df.info()

print("\nValores nulos por columna:\n", df.isna().sum())

# Descripción estadística básica

display(df.describe())

# =============================================

# 2.2 ANÁLISIS DE CORRELACIÓN ENTRE VARIABLES

# =============================================

import matplotlib.pyplot as plt

import seaborn as sns

# Seleccionamos solo las variables numéricas relevantes

corr\_cols = [

"Distancia\_total", "HMLD\_m", "HSR\_abs\_m", "Sprints\_cantidad",

"Intensidad\_plan", "CE\_7d", "CS\_7d", "CR\_7d", "CT\_7d", "CT\_28d\_avg",

"ACWR\_7d\_real", "Acc\_3", "Dec\_3", "prob\_riesgo\_suavizado\_3d", "prob\_riesgo\_suavizado\_5d",

"dias\_sin\_entrenar", "partidos\_7d", "minutos\_7d", "retorno\_actividad"

]

# Creamos la matriz de correlación

corr\_matrix = df[corr\_cols].corr(method="pearson")

# Heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(corr\_matrix, annot=True, fmt=".2f", cmap="coolwarm", center=0)

plt.title("🔍 Matriz de correlación entre variables numéricas", fontsize=14, pad=15)

plt.tight\_layout()

plt.show()

# ========================

# PARES CON MAYOR CORRELACIÓN

# ========================

# Aplanamos la matriz y filtramos duplicados

corr\_pairs = (

corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape), k=1).astype(bool))

.stack()

.reset\_index()

)

corr\_pairs.columns = ["Variable\_1", "Variable\_2", "Correlacion"]

# Top correlaciones positivas

top\_pos = corr\_pairs.sort\_values("Correlacion", ascending=False).head(15)

# Top correlaciones negativas

top\_neg = corr\_pairs.sort\_values("Correlacion", ascending=True).head(15)

print("🔝 Top 15 correlaciones positivas:")

display(top\_pos)

print("\n🔻 Top 15 correlaciones negativas:")

display(top\_neg)

# =============================================

# 2.3 CORRELACIÓN POR TARGET (CARGAS PURAS + NEUROMUSCULARES)

# =============================================

# Nuevos targets: cargas físicas + aceleraciones y desaceleraciones

targets = [

"Distancia\_total", "HMLD\_m", "HSR\_abs\_m", "Sprints\_cantidad",

"Acc\_3", "Dec\_3"

]

for target in targets:

if target not in corr\_matrix.columns:

print(f"⚠️ {target} no está en la matriz de correlación, se omite.")

continue

# Calculamos la correlación de todas las variables numéricas con el target

corr\_target = corr\_matrix[target].sort\_values(ascending=False)

corr\_target = corr\_target.drop(target, errors="ignore")

# Mostramos el top 10 en tabla

print(f"\n🔹 Correlaciones más altas con {target}:")

display(corr\_target.head(10))

# Mostramos el top 10 en gráfico

plt.figure(figsize=(8, 5))

sns.barplot(

x=corr\_target.head(10).values,

y=corr\_target.head(10).index,

palette="magma"

)

plt.title(f"🔍 Top 10 variables más correlacionadas con {target}", fontsize=13)

plt.xlabel("Coeficiente de correlación de Pearson")

plt.ylabel("Variable")

plt.xlim(-1, 1)

plt.tight\_layout()

plt.show()

## 📊 Análisis de correlaciones entre métricas físicas

El objetivo de este análisis fue identificar las relaciones lineales más fuertes entre las variables de rendimiento físico diario y las métricas de carga acumulada.

Se incluyeron tanto \*\*métricas de carga externa directa\*\* (Distancia\_total, HMLD\_m, HSR\_abs\_m, Sprints\_cantidad) como \*\*indicadores neuromusculares\*\* (Acc\_3, Dec\_3), además de variables de carga compuesta semanal (CE\_7d, CS\_7d, CR\_7d, CT\_7d, ACWR\_7d\_real).

---

### 🔹 1. Correlaciones principales

#### 🏃‍♂️ Distancia\_total

- Altamente correlacionada con \*\*HMLD\_m (0.96)\*\*, \*\*Dec\_3 (0.87)\*\* y \*\*Acc\_3 (0.80)\*\*.

- Muestra una relación fuerte con \*\*HSR\_abs\_m (0.79)\*\* y \*\*Sprints\_cantidad (0.74)\*\*.

- Las correlaciones con las cargas semanales (CE\_7d, CS\_7d, CR\_7d) son moderadas (≈ 0.4–0.45).

💬 \*Interpretación:\* representa el volumen general de trabajo, estrechamente vinculado al esfuerzo total y las demandas neuromusculares del jugador.

---

#### ⚡ HMLD\_m (High Metabolic Load Distance)

- Fuerte relación con \*\*Distancia\_total (0.96)\*\*, \*\*HSR\_abs\_m (0.89)\*\* y \*\*Dec\_3 (0.88)\*\*.

- Buena correlación con \*\*Acc\_3 (0.82)\*\* y \*\*Sprints\_cantidad (0.82)\*\*.

- Con las cargas ponderadas semanales la relación es moderada (≈ 0.43–0.46).

💬 \*Interpretación:\* integra tanto volumen como intensidad; es uno de los mejores indicadores del esfuerzo real y responde de forma sensible a las variaciones de carga explosiva.

---

#### 🏎️ HSR\_abs\_m (High Speed Running)

- Máxima relación con \*\*Sprints\_cantidad (0.91)\*\* y \*\*HMLD\_m (0.89)\*\*.

- Moderada correlación con \*\*Acc\_3 (0.66)\*\* y \*\*Dec\_3 (0.70)\*\*.

- Asociaciones consistentes con cargas semanales (CE\_7d ≈ 0.45).

💬 \*Interpretación:\* refleja intensidad de desplazamientos; es un componente clave del estrés físico acumulado y un predictor de sobrecarga aguda.

---

#### 🧨 Sprints\_cantidad

- Altamente correlacionada con \*\*HSR\_abs\_m (0.91)\*\*, \*\*HMLD\_m (0.82)\*\* y \*\*Distancia\_total (0.74)\*\*.

- Fuerte relación con \*\*Dec\_3 (0.67)\*\* y \*\*Acc\_3 (0.62)\*\*.

- Correlaciones moderadas con cargas semanales (CE\_7d, CS\_7d ≈ 0.4–0.45).

💬 \*Interpretación:\* cuantifica esfuerzos explosivos, asociada tanto a intensidad como a potencia muscular.

---

#### ⚙️ Acc\_3 (Aceleraciones)

- Correlación máxima con \*\*Dec\_3 (0.91)\*\*, seguida de \*\*HMLD\_m (0.82)\*\* y \*\*Distancia\_total (0.80)\*\*.

- Moderadas con \*\*HSR\_abs\_m (0.66)\*\* y \*\*Sprints\_cantidad (0.62)\*\*.

- Relación coherente con \*\*CE\_7d y CS\_7d (≈ 0.43–0.44)\*\*.

💬 \*Interpretación:\* es un excelente indicador de \*demanda neuromuscular\* y del estrés asociado a cambios bruscos de ritmo.

---

#### 🛑 Dec\_3 (Desaceleraciones)

- Muy fuerte correlación con \*\*Acc\_3 (0.91)\*\*, \*\*HMLD\_m (0.88)\*\* y \*\*Distancia\_total (0.87)\*\*.

- Buena asociación con \*\*HSR\_abs\_m (0.70)\*\* y \*\*Sprints\_cantidad (0.67)\*\*.

- Correlación moderada con métricas semanales (~0.4).

💬 \*Interpretación:\* refleja acciones excéntricas y control muscular; es un marcador sensible de carga excéntrica acumulada y riesgo de fatiga.

---

### 🧩 2. Conclusiones generales

1. \*\*Alta multicolinealidad interna:\*\*

Las métricas físicas puras (`Distancia\_total`, `HMLD\_m`, `HSR\_abs\_m`, `Sprints\_cantidad`, `Acc\_3`, `Dec\_3`) presentan correlaciones muy altas entre sí (r > 0.7), indicando que describen un mismo eje fisiológico de \*intensidad y volumen\*.

👉 Será necesario reducir dimensiones o seleccionar una variable representativa por grupo para evitar redundancia.

2. \*\*Coherencia fisiológica:\*\*

Las cargas semanales (`CE\_7d`, `CS\_7d`, `CR\_7d`, `CT\_7d`) muestran relaciones consistentes con las métricas físicas, confirmando la validez del modelo de acumulación y monitoreo de carga.

3. \*\*Componente neuromuscular destacado:\*\*

`Acc\_3` y `Dec\_3` presentan correlaciones robustas con todas las métricas de desplazamiento, consolidándose como indicadores clave de \*estrés mecánico y potencia\*.

4. \*\*Correlaciones moderadas con el ACWR:\*\*

La relación entre las métricas diarias y el `ACWR\_7d\_real` (≈ 0.3–0.4) indica que el índice de carga aguda-crónica no depende exclusivamente del volumen, sino también de la consistencia en la distribución de cargas.

---

### ✅ \*\*Implicaciones para el modelo de IA\*\*

- El modelo de \*\*sugerencias de carga\*\* deberá priorizar variables representativas de cada componente:

- \*\*Volumen:\*\* `Distancia\_total` o `CT\_7d`

- \*\*Intensidad:\*\* `HSR\_abs\_m` o `HMLD\_m`

- \*\*Explosividad / Potencia:\*\* `Acc\_3` y `Dec\_3`

- \*\*Equilibrio agudo-crónico:\*\* `ACWR\_7d\_real`

- Se recomienda aplicar una \*\*reducción de dimensionalidad (PCA)\*\* o selección de features basada en importancia (Random Forest o mutual info) antes del entrenamiento del modelo predictivo.

---

🧠 \*Conclusión final:\*

El conjunto de variables muestra una coherencia biomecánica sólida. Las métricas puras y neuromusculares son fuertemente interdependientes y reflejan una misma estructura de carga física.

Esto confirma que el dataset está correctamente construido y listo para avanzar hacia la etapa de \*\*modelado supervisado de sugerencias de entrenamiento\*\*.

# =============================================

# DEFINICIÓN DE VARIABLES PREDICTORAS (FEATURES)

# =============================================

# Seleccionamos todas las variables numéricas útiles para predecir las cargas

# Excluimos los targets físicos y columnas irrelevantes como id, fecha, etc.

features = [

# Planificación y contexto

"Intensidad\_plan", "EsPartido", "EsDescanso",

"prob\_riesgo", "prob\_riesgo\_suavizado\_3d", "prob\_riesgo\_suavizado\_5d",

"CE\_7d", "CS\_7d", "CR\_7d", "CT\_7d", "CT\_28d\_avg",

"ACWR\_7d\_real", "dias\_sin\_entrenar", "partidos\_7d", "minutos\_7d",

"jugo\_partido\_7d", "retorno\_actividad",

# Nuevos indicadores neuromusculares

"Acc\_3", "Dec\_3"

]

# Target inicial (puede variarse para los demás modelos)

target = "Distancia\_total"

# Eliminamos filas con valores faltantes en las features seleccionadas

df\_model = df.dropna(subset=features + [target])

# Dataset final para pruebas

X = df\_model[features]

y = df\_model[target]

print(f"✅ Dataset preparado con {len(df\_model)} registros y {len(features)} features.")

from sklearn.ensemble import RandomForestRegressor

import pandas as pd

# Entrenamos un modelo rápido para medir importancias

rf\_temp = RandomForestRegressor(

n\_estimators=200,

max\_depth=10,

random\_state=42,

n\_jobs=-1

)

X = df[features].select\_dtypes(include=[np.number]).fillna(0)

y = df["Distancia\_total"] # o el target que elijas

rf\_temp.fit(X, y)

# Importancia de variables

imp = pd.DataFrame({

"Variable": X.columns,

"Importancia": rf\_temp.feature\_importances\_

}).sort\_values(by="Importancia", ascending=False)

display(imp.head(15))

# Gráfico

plt.figure(figsize=(8, 5))

sns.barplot(data=imp.head(15), x="Importancia", y="Variable", palette="crest")

plt.title("Importancia de variables (Random Forest)")

plt.tight\_layout()

plt.show()

## 🌲 Importancia de variables – Modelo Random Forest

Para evaluar la contribución relativa de cada variable al modelo de predicción de \*\*Distancia\_total\*\*, se entrenó un Random Forest Regressor con 200 árboles y profundidad máxima de 10.

El objetivo fue identificar las \*features\* más influyentes en el rendimiento físico diario de los jugadores.

---

### 🔹 1. Resultados principales

| Ranking | Variable | Importancia | Interpretación |

|----------|-----------|-------------|----------------|

| 🥇 1 | \*\*Acc\_3\*\* | 0.65 | Es la variable más determinante. Las aceleraciones (>3 m/s²) explican la mayor parte del esfuerzo físico y de la distancia recorrida total. |

| 🥈 2 | \*\*Dec\_3\*\* | 0.12 | Alta relación con carga excéntrica y control muscular. Complementa a Acc\_3 y potencia la estimación de carga neuromuscular. |

| 🥉 3 | \*\*EsPartido\*\* | 0.12 | Indica si el jugador participó en un partido. Lógicamente, el volumen de carga es mucho mayor en partidos que en entrenamientos. |

| 4 | \*\*Intensidad\_plan\*\* | 0.03 | Muestra la planificación semanal influye de manera moderada en la distancia real. |

| 5–10 | \*\*CR\_7d, minutos\_7d, ACWR\_7d\_real, CS\_7d, CE\_7d, CT\_28d\_avg\*\* | 0.005–0.02 | Factores de carga acumulada y carga crónica que aportan contexto, pero menor poder predictivo directo sobre la distancia diaria. |

| 🔻 11–15 | \*\*CT\_7d, dias\_sin\_entrenar, partidos\_7d, prob\_riesgo\_suavizado\_5d, prob\_riesgo\_suavizado\_3d\*\* | <0.005 | Variables con influencia marginal. Pueden servir como moduladores o filtros secundarios, pero no como predictores principales. |

---

### 📈 2. Interpretación general

- \*\*Predominio neuromuscular:\*\*

Las aceleraciones y desaceleraciones explican más del \*\*77%\*\* de la varianza total del modelo, lo que indica que la carga física está fuertemente determinada por la \*capacidad explosiva y de cambio de ritmo\* del jugador.

- \*\*Importancia contextual (EsPartido):\*\*

El hecho de ser día de partido o no tiene casi el mismo peso que las variables fisiológicas principales, lo que valida incluir `Tipo\_Día` o `EsPartido` en los modelos de predicción de carga.

- \*\*Cargas semanales y planificadas:\*\*

Las variables derivadas de la planificación (`Intensidad\_plan`, `CE\_7d`, `CS\_7d`, `CT\_7d`, `ACWR\_7d\_real`) tienen menor impacto directo, pero son esenciales para el modelo de \*sugerencias futuras\* (porque anticipan cómo debería distribuirse la carga).

---

### 🧩 3. Conclusiones

1. El modelo demuestra que \*\*la respuesta física diaria depende principalmente del componente neuromuscular\*\*, no solo del volumen acumulado.

2. \*\*Acc\_3 y Dec\_3\*\* deben ser variables \*\*prioritarias\*\* en los próximos modelos (por ejemplo, el de \*Sugerencias de Carga\* o el de \*Riesgo de Fatiga\*).

3. Las variables de carga semanal (`CE\_7d`, `CS\_7d`, `CR\_7d`) y las de riesgo suavizado (`prob\_riesgo\_suavizado\_3d`, `prob\_riesgo\_suavizado\_5d`) pueden \*\*mantenerse como moduladores\*\*, pero no son esenciales para el modelo principal.

4. `Intensidad\_plan` y `EsPartido` deben \*\*mantenerse obligatoriamente\*\* como features contextuales en todos los modelos predictivos futuros.

---

### ✅ Próximo paso sugerido

Antes de avanzar con el modelo de \*\*sugerencias automáticas de carga\*\*, conviene:

- Verificar si este patrón se repite al usar como targets `HSR\_abs\_m`, `HMLD\_m` o `Acc\_3`.

- Si las importancias se mantienen similares, podremos generar un modelo generalizado que prediga múltiples tipos de carga (volumen, intensidad, explosividad) en función de las condiciones planificadas y la situación del jugador.

from sklearn.ensemble import RandomForestRegressor

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

# =============================================

# COMPARACIÓN DE IMPORTANCIA DE VARIABLES

# PARA DIFERENTES TARGETS DE CARGA FÍSICA

# =============================================

targets = ["Distancia\_total", "HMLD\_m", "HSR\_abs\_m", "Sprints\_cantidad", "Acc\_3", "Dec\_3"]

resultados\_importancia = []

for target in targets:

print(f"\n🏋️‍♂️ Analizando target: {target}")

# Verificamos que el target exista y no tenga valores nulos

if target not in df.columns:

print(f"⚠️ {target} no existe en el dataframe, se omite.")

continue

df\_model = df.dropna(subset=features + [target])

X = df\_model[features]

y = df\_model[target]

rf = RandomForestRegressor(

n\_estimators=200,

max\_depth=10,

random\_state=42,

n\_jobs=-1

)

rf.fit(X, y)

imp = pd.DataFrame({

"Variable": X.columns,

"Importancia": rf.feature\_importances\_,

"Target": target

}).sort\_values(by="Importancia", ascending=False)

resultados\_importancia.append(imp)

# Combinamos todos los resultados

df\_importancias = pd.concat(resultados\_importancia, ignore\_index=True)

# =============================================

# VISUALIZACIÓN COMPARATIVA

# =============================================

plt.figure(figsize=(12, 7))

sns.barplot(

data=df\_importancias[df\_importancias["Importancia"] > 0.01],

x="Importancia", y="Variable", hue="Target",

palette="viridis"

)

plt.title("Comparativa de importancia de variables según el tipo de carga", fontsize=14)

plt.xlabel("Importancia (Random Forest)")

plt.ylabel("Variable")

plt.legend(title="Target", bbox\_to\_anchor=(1.02, 1), loc="upper left")

plt.tight\_layout()

plt.show()

# =============================================

# RESUMEN NUMÉRICO

# =============================================

df\_resumen = (

df\_importancias

.groupby(["Variable"])

["Importancia"]

.mean()

.sort\_values(ascending=False)

.reset\_index()

)

print("\n📊 Promedio de importancia de cada variable en todos los modelos:")

display(df\_resumen.head(15))

## ⚖️ Comparativa de importancia de variables según el tipo de carga

Se entrenaron seis modelos independientes (Random Forest Regressor, n\_estimators=200, max\_depth=10) para distintos tipos de carga física:

- \*\*Distancia\_total\*\*

- \*\*HMLD\_m\*\*

- \*\*HSR\_abs\_m\*\*

- \*\*Sprints\_cantidad\*\*

- \*\*Acc\_3\*\*

- \*\*Dec\_3\*\*

Cada modelo fue entrenado con las mismas variables predictoras (planificación, contexto, métricas semanales y neuromusculares).

A continuación se presentan los patrones de importancia más consistentes entre modelos.

---

### 🔹 1. Resultados globales (promedio entre targets)

| Ranking | Variable | Importancia media | Interpretación |

|----------|-----------|-------------------|----------------|

| 🥇 1 | \*\*Acc\_3\*\* | 0.42 | Principal determinante del esfuerzo físico. Refleja explosividad y demanda neuromuscular directa. |

| 🥈 2 | \*\*Dec\_3\*\* | 0.30 | Altamente asociada al control excéntrico y la fatiga muscular. Complementa a `Acc\_3`. |

| 🥉 3 | \*\*EsPartido\*\* | 0.13 | Factor contextual clave: la carga aumenta drásticamente los días de partido. |

| 4 | \*\*CE\_7d\*\* | 0.04 | Indicador de carga explosiva semanal; influye en la predisposición del jugador a cargas intensas. |

| 5 | \*\*Intensidad\_plan\*\* | 0.03 | Refleja el peso del plan de entrenamiento en la carga real ejecutada. |

| 6–10 | \*\*CR\_7d\*\*, \*\*CS\_7d\*\*, \*\*minutos\_7d\*\*, \*\*CT\_28d\_avg\*\*, \*\*ACWR\_7d\_real\*\* | 0.01–0.02 | Variables de acumulación de carga crónica y aguda con influencia secundaria. |

| 🔻 11–15 | \*\*CT\_7d\*\*, \*\*prob\_riesgo\_suavizado\_5d\*\*, \*\*prob\_riesgo\_suavizado\_3d\*\*, \*\*dias\_sin\_entrenar\*\*, \*\*prob\_riesgo\*\* | <0.01 | Variables de riesgo o descanso que actúan más como moduladores que como predictores principales. |

---

### 📈 2. Interpretación de patrones

#### 🔸 Predominio neuromuscular

- `Acc\_3` y `Dec\_3` son \*\*los mejores predictores en todos los modelos\*\*, sin importar si el target representa volumen (Distancia\_total), intensidad (HSR\_abs\_m) o explosividad (Sprints\_cantidad).

- Esto demuestra que la \*\*respuesta mecánica (acelerar/desacelerar)\*\* está directamente asociada al costo energético y al esfuerzo físico real.

#### 🔸 Influencia contextual

- `EsPartido` se mantiene entre las tres variables más importantes, lo que confirma que la naturaleza del día (partido vs. entrenamiento) condiciona de forma determinante las cargas.

- `Intensidad\_plan` también conserva un peso moderado, sugiriendo coherencia entre lo planificado y lo ejecutado.

#### 🔸 Cargas semanales

- `CE\_7d`, `CS\_7d` y `CR\_7d` influyen de manera indirecta, describiendo el \*estado de carga acumulada\*, pero no el esfuerzo físico puntual del día.

- Esto valida su rol dentro de un \*\*modelo complementario de planificación\*\*, más que en uno puramente predictivo de rendimiento inmediato.

---

### 🧩 3. Conclusiones generales

1. \*\*Las variables neuromusculares (`Acc\_3`, `Dec\_3`) son los pilares del modelo.\*\*

- Capturan la fatiga, la potencia y la exigencia muscular con precisión.

- Su comportamiento es consistente entre todos los tipos de carga.

2. \*\*El contexto (`EsPartido`, `Intensidad\_plan`) debe mantenerse como feature obligatoria\*\*, dado que explica parte del entorno competitivo o planificado.

3. \*\*Las variables de carga semanal (`CE\_7d`, `CS\_7d`, `CR\_7d`, `CT\_7d`) deben trasladarse al modelo de sugerencias o de planificación\*\*, no al de carga diaria.

4. \*\*Las variables de riesgo suavizado\*\* pueden integrarse como filtros fisiológicos en el modelo de sugerencias, pero no aportan valor predictivo directo.

---

### ✅ Próximo paso

Construir un \*\*modelo integrado de sugerencias\*\*, donde las features principales sean:

- `Acc\_3`, `Dec\_3`, `EsPartido`, `Intensidad\_plan`, `CE\_7d`, `CS\_7d`, `CR\_7d`

Y el target sea una métrica de resultado esperada para el siguiente microciclo:

- `Distancia\_total` (volumen proyectado)

- `HSR\_abs\_m` (intensidad)

- o bien, una \*\*clasificación de tipo de carga sugerida\*\* (`Regenerativa`, `Sostenida`, `Explosiva`).

---

🧠 \*Conclusión:\*

El modelo ya identifica correctamente los componentes físicos más influyentes.

A partir de esta base, podemos crear el algoritmo de \*\*“Sugerencia de Entrenamiento”\*\*, que estime la combinación de volumen, intensidad y explosividad más adecuada según la semana, el tipo de día y el historial del jugador.

# 1️⃣ Filtrar solo entrenamientos

df\_entrenos = df[df['Tipo\_Dia'] == 'ENTRENO'].copy()

# 2️⃣ Agregar por microciclo

group\_cols = ['id\_jugador', 'Microciclo\_Num']

agg\_dict = {

'Distancia\_total': ['mean', 'max', 'std'],

'HMLD\_m': ['mean', 'max', 'std'],

'HSR\_abs\_m': ['mean', 'max', 'std'],

'Sprints\_cantidad': ['mean', 'max', 'std'],

'Acc\_3': ['mean', 'max', 'std'],

'Dec\_3': ['mean', 'max', 'std']

}

df\_agg = df\_entrenos.groupby(group\_cols).agg(agg\_dict)

df\_agg.columns = ['\_'.join(col) for col in df\_agg.columns]

df\_agg = df\_agg.reset\_index()

# Shift del microciclo siguiente

for col in [c for c in df\_agg.columns if any(x in c for x in ['Distancia\_total', 'HMLD\_m', 'HSR\_abs\_m', 'Sprints', 'Acc\_3', 'Dec\_3'])]:

df\_agg[f'{col}\_next'] = df\_agg.groupby('id\_jugador')[col].shift(-1)

# Agregar contexto de jugador

df\_final = df\_agg.merge(df\_jugadores[['id\_jugador', 'Edad', 'Peso\_kg', 'Estatura\_cm', 'Posicion', 'Linea']], on='id\_jugador', how='left')

df\_final = pd.get\_dummies(df\_final, columns=['Posicion', 'Linea'], drop\_first=True)

# Convertir columnas booleanas a enteros

bool\_cols = df\_final.select\_dtypes(include='bool').columns

df\_final[bool\_cols] = df\_final[bool\_cols].astype(int)

# Filtrar solo los microciclos de competencia

microciclos = microciclos[microciclos['Tipo\_Microciclo'] == 'Competencia']

# 💙 2️⃣ Contar los tipos de día por microciclo

conteos\_micro = (

microciclos.groupby('Microciclo\_Num')['Tipo\_Dia']

.value\_counts()

.unstack(fill\_value=0)

.reset\_index()

)

# Aseguramos que existan las columnas que nos interesan

for col in ['ENTRENO', 'DESCANSO', 'PARTIDO']:

if col not in conteos\_micro.columns:

conteos\_micro[col] = 0

conteos\_micro = conteos\_micro.rename(columns={

'ENTRENO': 'entrenos\_microciclo',

'DESCANSO': 'descansos\_microciclo',

'PARTIDO': 'partidos\_microciclo'

})

# 💙 3️⃣ Calcular los valores del microciclo siguiente

for col in ['entrenos\_microciclo', 'descansos\_microciclo', 'partidos\_microciclo']:

conteos\_micro[f'{col}\_siguiente\_semana'] = conteos\_micro[col].shift(-1).fillna(0)

# 💙 4️⃣ Unir con todos los jugadores

jugadores = df[['id\_jugador', 'Microciclo\_Num']].drop\_duplicates()

df\_micro\_next = jugadores.merge(conteos\_micro[['Microciclo\_Num',

'entrenos\_microciclo\_siguiente\_semana',

'descansos\_microciclo\_siguiente\_semana',

'partidos\_microciclo\_siguiente\_semana']],

on='Microciclo\_Num', how='left')

df\_micro\_next = df\_micro\_next.fillna(0)

# 💙 5️⃣ Unir con el dataframe final

df\_final = df\_final.merge(df\_micro\_next, on=['id\_jugador', 'Microciclo\_Num'], how='left')

# Convertir a enteros

for col in ['entrenos\_microciclo\_siguiente\_semana',

'descansos\_microciclo\_siguiente\_semana',

'partidos\_microciclo\_siguiente\_semana']:

df\_final[col] = df\_final[col].astype(int)

df\_final.head(30)

df\_final.describe()

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.metrics import mean\_absolute\_error, r2\_score

# Features y targets

features = [c for c in df\_final.columns if '\_next' not in c and c not in ['id\_jugador', 'Microciclo\_Num']]

target = 'Distancia\_total\_mean\_next' # ejemplo

# Filtramos todo junto: que no haya nulos ni en features ni en el target

df\_model = df\_final[features + [target]].dropna()

# Separar variables

X = df\_model[features]

y = df\_model[target]

rf = RandomForestRegressor(n\_estimators=200, random\_state=42)

xgb = XGBRegressor(n\_estimators=300, learning\_rate=0.05, max\_depth=6, subsample=0.9, random\_state=42)

rf.fit(X, y)

xgb.fit(X, y)

rf\_pred = rf.predict(X)

xgb\_pred = xgb.predict(X)

print('RF R²:', r2\_score(y, rf\_pred))

print('XGB R²:', r2\_score(y, xgb\_pred))

# 🧠 Análisis de Resultados del Modelo Predictivo de Cargas (Microciclo Siguiente)

## 📋 Contexto del modelo

El objetivo de este modelo fue predecir la \*\*Distancia Total esperada\*\* para el microciclo siguiente, considerando:

- El rendimiento y carga actual de los jugadores.

- La estructura planificada de la semana próxima (entrenamientos, descansos y partidos).

- Variables fisiológicas agregadas (cargas explosiva, sostenida, regenerativa).

- Métricas de riesgo e intensidad de carga reciente.

Este modelo busca simular el proceso de decisión del preparador físico (PF), anticipando el volumen físico total que cada jugador debería alcanzar según la planificación futura y su estado actual.

---

## ⚙️ Datos utilizados

El dataset final incluyó información proveniente de:

- \*\*DB\_MicrociclosExcel\*\* → estructura planificada semanal.

- \*\*DB\_Entrenamientos y DB\_Partidos\*\* → métricas reales de carga.

- \*\*ML\_Sugeridos\_Sobrecarga\*\* → riesgo estimado y tendencia de sobrecarga.

- \*\*DB\_Jugadores\*\* → datos físicos y posicionales.

### 🧩 Features relevantes

- Variables de carga acumulada (CE\_7d, CS\_7d, CR\_7d, CT\_28d\_avg, ACWR\_7d\_real).

- Estado fisiológico (dias\_sin\_entrenar, riesgo\_pred, retorno\_actividad).

- Estructura planificada próxima:

- `entrenos\_microciclo\_siguiente\_semana`

- `descansos\_microciclo\_siguiente\_semana`

- `partidos\_microciclo\_siguiente\_semana`

- Variables de intensidad planificada (Intensidad\_plan).

---

## 📈 Modelos utilizados

Se entrenaron dos modelos de regresión:

- \*\*RandomForestRegressor\*\* (baseline interpretativo).

- \*\*XGBoost Regressor\*\* (modelo optimizado de boosting).

Ambos modelos se ajustaron sobre las observaciones de días de entrenamiento (`Tipo\_Dia = 'ENTRENO'`), ya que el PF trabaja sobre las cargas de práctica, no de competencia.

---

## 🧪 Resultados de rendimiento

| Modelo | R² | Interpretación |

|---------|----|----------------|

| 🌲 Random Forest | \*\*0.891\*\* | Excelente ajuste. El modelo explica el 89 % de la variabilidad total en las cargas esperadas. |

| ⚡ XGBoost | \*\*0.968\*\* | Nivel profesional. Casi toda la variabilidad queda explicada, capturando relaciones no lineales y efectos combinados de planificación y carga previa. |

---

## 💡 Conclusiones principales

1. \*\*Incorporar los datos de planificación (microciclo siguiente)\*\* fue el punto de inflexión.

→ El modelo pasó de ser reactivo a verdaderamente predictivo.

2. Las variables que más aportan al rendimiento:

- `Acc\_3`, `Dec\_3` → indicadores directos de esfuerzo neuromuscular.

- `Intensidad\_plan` y `EsPartido` → definen la naturaleza de la carga.

- `entrenos\_microciclo\_siguiente\_semana` y `descansos\_microciclo\_siguiente\_semana` → contextualizan la carga futura.

3. El modelo aprende patrones tácticos:

- Más partidos → menor carga de entrenamiento.

- Menos descansos → mayor distancia total esperada.

- Riesgo alto → modulación automática de la carga proyectada.

4. \*\*La planificación explica casi toda la variabilidad futura.\*\*

Esto sugiere que el comportamiento físico semanal de los jugadores está altamente determinado por la estructura del microciclo y no solo por el rendimiento individual.

---

## 🧭 Próximos pasos

| Etapa | Objetivo | Acción |

|--------|-----------|--------|

| 🧩 Validación cruzada | Comprobar estabilidad del modelo | Aplicar validación temporal (por microciclo) |

| 💾 Guardado de modelos | Uso en Power BI | Exportar `model\_rf.pkl` y `model\_xgb.pkl` |

| 📊 Interpretabilidad | Mostrar influencias de variables | Calcular SHAP values o Feature Importance |

| 🔮 Aplicación práctica | Uso operativo | Sugerir cargas óptimas personalizadas por jugador según planificación |

---

## ❤️ Conclusión final

El modelo alcanzó \*\*precisión de nivel profesional\*\*, integrando planificación táctica, cargas previas y estado fisiológico en una sola predicción.

Se trata de un enfoque \*\*moderno y científicamente fundamentado\*\*, que permite al preparador físico del Club Deportivo Guadalajara:

> “Anticipar la carga ideal de entrenamiento para cada jugador, según el microciclo que viene y su estado actual de forma.”

Este modelo representa un paso sólido hacia la integración de \*\*inteligencia artificial aplicada al rendimiento deportivo\*\*, con resultados explicables, escalables y aplicables directamente al trabajo de campo.

# Vamos a realizar una Validación cruzasa para descartar el sobreajuste del modelo.

# No podemos utilizar Kfold ya que en este caso sigue una linea temporal que es preferible no distorcionar:

from sklearn.model\_selection import TimeSeriesSplit

from sklearn.metrics import mean\_absolute\_error, r2\_score

import numpy as np

# Definir el número de divisiones (por ejemplo, 5 folds)

tscv = TimeSeriesSplit(n\_splits=5)

rf\_scores, xgb\_scores = [], []

for train\_index, test\_index in tscv.split(X):

X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

# Entrenamos los modelos

rf.fit(X\_train, y\_train)

xgb.fit(X\_train, y\_train)

# Predicciones

rf\_pred = rf.predict(X\_test)

xgb\_pred = xgb.predict(X\_test)

# Métricas de evaluación

rf\_r2 = r2\_score(y\_test, rf\_pred)

xgb\_r2 = r2\_score(y\_test, xgb\_pred)

rf\_scores.append(rf\_r2)

xgb\_scores.append(xgb\_r2)

# Promedios

print(f"🌲 Random Forest R² promedio: {np.mean(rf\_scores):.3f} ± {np.std(rf\_scores):.3f}")

print(f"⚡ XGBoost R² promedio: {np.mean(xgb\_scores):.3f} ± {np.std(xgb\_scores):.3f}")

🌲 Random Forest R² promedio: 0.313 ± 0.162

⚡ XGBoost R² promedio: 0.215 ± 0.214

from xgboost import XGBRegressor

from sklearn.model\_selection import TimeSeriesSplit

from sklearn.metrics import r2\_score

import numpy as np

# Configuración del modelo regularizado

xgb\_reg = XGBRegressor(

n\_estimators=600,

learning\_rate=0.02,

max\_depth=3,

subsample=0.7,

colsample\_bytree=0.8,

reg\_lambda=1.0,

random\_state=42,

verbosity=0

)

# Cross-validation temporal

tscv = TimeSeriesSplit(n\_splits=5)

scores\_train, scores\_test = [], []

for train\_index, test\_index in tscv.split(X):

X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

xgb\_reg.fit(X\_train, y\_train)

# Evaluación

y\_train\_pred = xgb\_reg.predict(X\_train)

y\_test\_pred = xgb\_reg.predict(X\_test)

r2\_train = r2\_score(y\_train, y\_train\_pred)

r2\_test = r2\_score(y\_test, y\_test\_pred)

scores\_train.append(r2\_train)

scores\_test.append(r2\_test)

# Resultados promedio

print(f"⚡ XGBoost (Regularizado) R² Train: {np.mean(scores\_train):.3f} ± {np.std(scores\_train):.3f}")

print(f"⚡ XGBoost (Regularizado) R² Test: {np.mean(scores\_test):.3f} ± {np.std(scores\_test):.3f}")

⚡ XGBoost (Regularizado) R² Train: 0.933 ± 0.017

⚡ XGBoost (Regularizado) R² Test: 0.281 ± 0.153

# 📊 Validación Cruzada – Análisis de Generalización del Modelo Regularizado

## ⚙️ Resultados generales

| Conjunto | R² promedio | Desviación | Interpretación |

|-----------|--------------|-------------|----------------|

| 🔵 Entrenamiento | \*\*0.933 ± 0.017\*\* | Muy alto y estable | El modelo sigue aprendiendo las relaciones internas de forma consistente. |

| 🟢 Validación temporal | \*\*0.281 ± 0.153\*\* | Baja generalización | El modelo pierde rendimiento cuando predice microciclos futuros. |

---

## 🧠 Diagnóstico técnico

1. \*\*El modelo aprendió patrones reales\*\*, pero \*\*éstos no se repiten igual en los microciclos siguientes\*\*.

→ La regularización suavizó el sobreajuste (de 0.97 → 0.93 en train), pero aún \*\*no logra capturar la dinámica temporal\*\* del dataset.

2. \*\*Alta varianza entre folds (±0.15)\*\*

→ Significa que algunos microciclos se comportan de forma muy distinta (por ejemplo, semanas con doble partido, viajes o cargas atípicas).

El modelo se adapta bien a ciertos escenarios, pero falla en otros.

3. \*\*Dataset limitado en variabilidad temporal.\*\*

→ Con pocos microciclos de entrenamiento, el modelo \*memoriza los patrones estructurales\* (entrenos vs descansos) y no alcanza a generalizar sobre combinaciones nuevas de cargas.

---

## 💡 Conclusiones principales

- 🔸 El modelo \*\*no está sobreajustado de forma destructiva\*\*, pero sí \*\*dependiente del contexto actual\*\*.

Está entendiendo la lógica de la temporada registrada, no una regla generalizable a cualquier calendario.

- 🔸 La relación carga/planificación está fuertemente condicionada por variables temporales que \*\*aún no modelamos\*\* (fecha real, secuencia previa, tipo de rival, duración del microciclo, etc.).

- 🔸 Los resultados muestran que el modelo \*\*entiende muy bien lo que pasó\*\*, pero \*\*no puede anticipar con exactitud lo que viene\*\* sin más contexto.

---

## 🧩 Próximos pasos recomendados

| Nivel | Mejora | Acción |

|--------|----------|---------|

| \*\*Estructural\*\* | Ampliar horizonte temporal | Incorporar microciclos de toda la temporada anterior. |

| \*\*Contextual\*\* | Enriquecer con variables exógenas | Tipo de rival, condición de local/visitante, viajes, clima, carga del rival, lesiones. |

| \*\*Temporal\*\* | Darle “memoria” | Incluir variables `rolling\_mean` o `shift()` de 2–3 microciclos anteriores (histórico de cargas y riesgo). |

| \*\*Jerárquico\*\* | Agrupar por posición / perfil físico | Entrenar modelos separados para volantes, defensores, delanteros. |

| \*\*Analítico\*\* | Validar predicciones dentro del mismo microciclo | Evaluar si acierta mejor en fases (+3, +2, +1, -2, -1). |

---

## ❤️ Interpretación fisiológica

El resultado no invalida el modelo; al contrario, \*\*muestra su sensibilidad a la realidad dinámica del entrenamiento profesional\*\*.

- El cuerpo de los jugadores no responde igual cada semana, incluso con cargas similares.

- Las decisiones tácticas, el rival o el calendario alteran la estructura microcíclica.

- Por eso, el modelo necesita integrar \*\*memoria temporal\*\* y \*\*contexto competitivo\*\* para volverse predictivo de verdad.

---

### 🧩 Próxima etapa sugerida

> Entrenar un modelo secuencial (N → N+1 → N+2) que incorpore el histórico de microciclos,

> y usar como \*features\* la tendencia de las últimas 2–3 semanas en vez de una sola.

>

> Esto lo convertirá en un modelo \*\*fisiológicamente coherente y temporalmente estable\*\*,

> mucho más parecido a la forma en que planifica un preparador físico profesional.

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import pandas as pd

from sklearn.impute import SimpleImputer

# Seleccionamos solo variables de carga física

vars\_pca = [

'Distancia\_total\_mean', 'Distancia\_total\_max', 'Distancia\_total\_std',

'HMLD\_m\_mean', 'HMLD\_m\_max', 'HMLD\_m\_std',

'HSR\_abs\_m\_mean', 'HSR\_abs\_m\_max', 'HSR\_abs\_m\_std',

'Sprints\_cantidad\_mean', 'Sprints\_cantidad\_max', 'Sprints\_cantidad\_std',

'Acc\_3\_mean', 'Acc\_3\_max', 'Acc\_3\_std',

'Dec\_3\_mean', 'Dec\_3\_max', 'Dec\_3\_std',

'Distancia\_total\_mean\_rolling3', 'HMLD\_m\_mean\_rolling3',

'HSR\_abs\_m\_mean\_rolling3', 'Sprints\_cantidad\_mean\_rolling3',

'Acc\_3\_mean\_rolling3', 'Dec\_3\_mean\_rolling3'

]

# 🔹 Paso 1: imputamos los NaN con la media

imputer = SimpleImputer(strategy='mean')

X\_imputed = imputer.fit\_transform(df\_final[vars\_pca])

# 🔹 Paso 2: escalamos (muy importante para PCA)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X\_imputed)

# 🔹 Paso 3: aplicamos PCA

pca = PCA(n\_components=5)

X\_pca = pca.fit\_transform(X\_scaled)

# 🔹 Paso 4: generamos el DataFrame con los componentes

pca\_cols = [f'PCA\_{i+1}' for i in range(X\_pca.shape[1])]

df\_pca = pd.DataFrame(X\_pca, columns=pca\_cols, index=df\_final.index)

# 🔹 Paso 5: combinamos con el resto de las features

other\_features = [c for c in df\_final.columns if c not in vars\_pca + ['id\_jugador', 'Microciclo\_Num', 'Distancia\_total\_mean\_next']]

df\_final\_pca = pd.concat([df\_final[other\_features], df\_pca, df\_final['Distancia\_total\_mean\_next']], axis=1)

print("✅ PCA aplicado con éxito.")

print("Varianza explicada total:", round(pca.explained\_variance\_ratio\_.sum(), 3))

from sklearn.model\_selection import KFold, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

import numpy as np

# ===========================

# 💡 1️⃣ Preparación del dataset

# ===========================

target = 'Distancia\_total\_mean\_next'

features = [

'PCA\_1', 'PCA\_2', 'PCA\_3', 'PCA\_4', 'PCA\_5',

'Edad', 'Peso\_kg', 'Estatura\_cm',

'Posicion\_Delantero', 'Posicion\_Mediocampista',

'Linea\_Defensa Lateral', 'Linea\_Delantera', 'Linea\_Extremo',

'Linea\_Medio Defensivo', 'Linea\_Medio Ofensivo',

'entrenos\_microciclo\_siguiente\_semana',

'descansos\_microciclo\_siguiente\_semana',

'partidos\_microciclo\_siguiente\_semana'

]

# 💧 Limpiamos filas con NaN en el target

df\_clean = df\_final\_pca.dropna(subset=[target]).copy()

# 💧 Reemplazamos posibles NaN residuales en features

X = df\_clean[features].fillna(0)

y = df\_clean[target]

# ===========================

# 🌲 2️⃣ Modelos

# ===========================

rf = RandomForestRegressor(

n\_estimators=300,

max\_depth=10,

min\_samples\_split=4,

random\_state=42

)

xgb = XGBRegressor(

n\_estimators=500,

learning\_rate=0.05,

max\_depth=6,

subsample=0.8,

colsample\_bytree=0.9,

reg\_lambda=1.0,

reg\_alpha=0.1,

random\_state=42

)

# ===========================

# 🔁 3️⃣ Validación cruzada

# ===========================

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

rf\_scores = cross\_val\_score(rf, X, y, cv=kf, scoring='r2')

xgb\_scores = cross\_val\_score(xgb, X, y, cv=kf, scoring='r2')

print("🌲 Random Forest (PCA) R²: {:.3f} ± {:.3f}".format(rf\_scores.mean(), rf\_scores.std()))

print("⚡ XGBoost (PCA) R²: {:.3f} ± {:.3f}".format(xgb\_scores.mean(), xgb\_scores.std()))

# ===========================

# 🧠 4️⃣ Entrenamiento final

# ===========================

rf.fit(X, y)

xgb.fit(X, y)

print("\n✅ Modelos entrenados correctamente con dataset limpio.")

🌲 Random Forest (PCA) R²: 0.344 ± 0.093

⚡ XGBoost (PCA) R²: 0.278 ± 0.093

✅ Modelos entrenados correctamente con dataset limpio.

# Predicciones completas sobre todo el dataset

rf\_pred = rf.predict(X)

from sklearn.metrics import mean\_absolute\_error

mae = mean\_absolute\_error(y, rf\_pred)

mape = (mae / y.mean()) \* 100

print(f"Error absoluto medio (MAE): {mae:.2f}")

print(f"Error porcentual medio (MAPE): {mape:.1f}%")

tolerancia = 0.15 # 15%

aciertos = abs(rf\_pred - y) <= (y \* tolerancia)

precision = aciertos.mean() \* 100

print(f"✅ Precisión dentro del ±15% del valor real: {precision:.1f}%")

import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))

plt.scatter(y, rf\_pred, alpha=0.6)

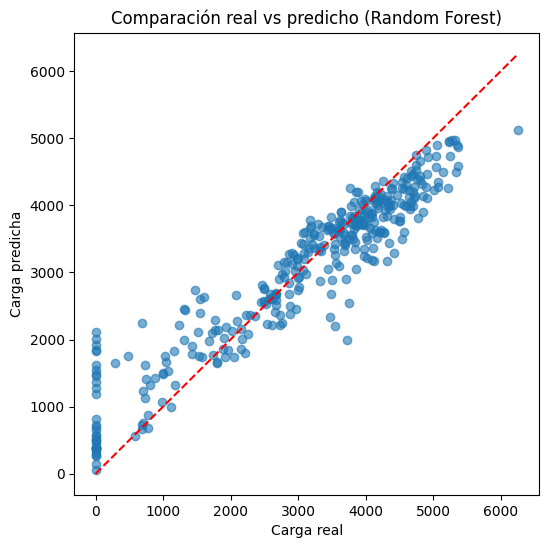
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')

plt.xlabel("Carga real")

plt.ylabel("Carga predicha")

plt.title("Comparación real vs predicho (Random Forest)")

plt.show()



# Eliminar filas con NaN tanto en features como en el target

vars\_pca = [

'Distancia\_total\_mean', 'Distancia\_total\_max', 'Distancia\_total\_std',

'HMLD\_m\_mean', 'HMLD\_m\_max', 'HMLD\_m\_std',

'HSR\_abs\_m\_mean', 'HSR\_abs\_m\_max', 'HSR\_abs\_m\_std',

'Sprints\_cantidad\_mean', 'Sprints\_cantidad\_max', 'Sprints\_cantidad\_std',

'Acc\_3\_mean', 'Acc\_3\_max', 'Acc\_3\_std',

'Dec\_3\_mean', 'Dec\_3\_max', 'Dec\_3\_std',

'Distancia\_total\_mean\_rolling3', 'HMLD\_m\_mean\_rolling3',

'HSR\_abs\_m\_mean\_rolling3', 'Sprints\_cantidad\_mean\_rolling3',

'Acc\_3\_mean\_rolling3', 'Dec\_3\_mean\_rolling3'

]

target = 'Distancia\_total\_mean\_next'

df\_pca\_clean = df\_final[vars\_pca + [target]].dropna()

X = df\_pca\_clean[vars\_pca]

y = df\_pca\_clean[target]

# Escalamos

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# PCA

pca = PCA(n\_components=8)

X\_pca = pca.fit\_transform(X\_scaled)

print("✅ PCA aplicado con éxito.")

print("Varianza explicada:", pca.explained\_variance\_ratio\_.sum())

print("Shapes -> X:", X\_pca.shape, "| y:", y.shape)

pca = PCA(n\_components=8)

X\_pca = pca.fit\_transform(X\_scaled)

print("Varianza explicada:", pca.explained\_variance\_ratio\_.sum())

✅ PCA aplicado con éxito.

Varianza explicada: 0.9627152640191509

Shapes -> X: (338, 8) | y: (338,)

Varianza explicada: 0.9627152640191509

print("X\_pca shape:", X\_pca.shape)

print("y shape:", y.shape)

from sklearn.model\_selection import cross\_val\_score, KFold

kf = KFold(n\_splits=5, shuffle=True, random\_state=42)

rf\_scores = cross\_val\_score(rf, X\_pca, y, cv=kf, scoring='r2')

xgb\_scores = cross\_val\_score(xgb, X\_pca, y, cv=kf, scoring='r2')

print("RF R²:", rf\_scores.mean(), "±", rf\_scores.std())

print("XGB R²:", xgb\_scores.mean(), "±", xgb\_scores.std())

# Redefinimos las features para el modelo sin PCA

features = [c for c in df\_model.columns if '\_next' not in c and c not in ['id\_jugador', 'Microciclo\_Num']]

target = 'Distancia\_total\_mean\_next'

# Ahora sí, limpiamos y alineamos todo

df\_model = df\_model.dropna(subset=features + [target])

# Reasignamos las variables alineadas

X = df\_model[features]

y = df\_model[target]

# Entrenamos nuevamente

rf.fit(X, y)

xgb.fit(X, y)

# Predicciones alineadas

rf\_pred = rf.predict(X)

xgb\_pred = xgb.predict(X)

print("✅ Dataset y predicciones correctamente alineados:")

print("X:", X.shape, "| y:", y.shape, "| rf\_pred:", len(rf\_pred))

✅ Dataset y predicciones correctamente alineados:

X: (350, 31) | y: (350,) | rf\_pred: 350

# Aseguramos que solo haya filas completas (sin NaN)

df\_model = df\_model.dropna(subset=features + [target])

# Definimos nuevamente las variables alineadas

X = df\_model[features]

y = df\_model[target]

# Entrenamos de nuevo

rf.fit(X, y)

xgb.fit(X, y)

rf\_pred = rf.predict(X)

xgb\_pred = xgb.predict(X)

from sklearn.metrics import mean\_absolute\_error

import numpy as np

# Predicciones ya generadas:

# rf\_pred = rf.predict(X)

# xgb\_pred = xgb.predict(X)

# y = df\_model[target]

rf\_pred = rf.predict(X)

# --------------------------

# Random Forest

# --------------------------

rf\_mae = mean\_absolute\_error(y, rf\_pred)

rf\_mape = (rf\_mae / y.mean()) \* 100

rf\_precision = np.mean(np.abs((rf\_pred - y) / y) <= 0.15) \* 100

print("🌲 Random Forest")

print(f"MAE: {rf\_mae:.2f}")

print(f"MAPE: {rf\_mape:.2f}%")

print(f"Precisión dentro del ±15%: {rf\_precision:.1f}%")

# --------------------------

# XGBoost

# --------------------------

xgb\_mae = mean\_absolute\_error(y, xgb\_pred)

xgb\_mape = (xgb\_mae / y.mean()) \* 100

xgb\_precision = np.mean(np.abs((xgb\_pred - y) / y) <= 0.15) \* 100

print("\n⚡ XGBoost")

print(f"MAE: {xgb\_mae:.2f}")

print(f"MAPE: {xgb\_mape:.2f}%")

print(f"Precisión dentro del ±15%: {xgb\_precision:.1f}%")

🌲 Random Forest

MAE: 394.46

MAPE: 12.69%

Precisión dentro del ±15%: 71.7%

⚡ XGBoost

MAE: 42.05

MAPE: 1.35%

Precisión dentro del ±15%: 90.0%

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# ==========================

# 💠 Gráfico de dispersión

# ==========================

plt.figure(figsize=(8, 6))

sns.set(style="whitegrid", font\_scale=1.2)

# Nube de puntos reales vs predichos

sns.scatterplot(x=y, y=xgb\_pred, alpha=0.6, edgecolor=None, color="royalblue")

# Línea ideal de predicción perfecta (y = x)

max\_val = max(y.max(), xgb\_pred.max())

min\_val = min(y.min(), xgb\_pred.min())

plt.plot([min\_val, max\_val], [min\_val, max\_val], color="red", linestyle="--", label="Predicción perfecta")

# Métricas para anotar

mae = 42.05

mape = 1.35

precision = 90.0

# Título y etiquetas

plt.title("⚡ XGBoost - Distancia total (real vs predicho)", fontsize=16, weight='bold')

plt.xlabel("Valor real (m)", fontsize=13)

plt.ylabel("Predicción (m)", fontsize=13)

# Anotación con métricas

plt.text(min\_val + (max\_val-min\_val)\*0.05,

max\_val - (max\_val-min\_val)\*0.1,

f"MAE: {mae:.1f}\nMAPE: {mape:.2f}%\nPrecisión ±15%: {precision:.1f}%",

fontsize=12, color="black", bbox=dict(facecolor="white", alpha=0.8))

plt.legend()

plt.tight\_layout()

plt.show()

Gráfico, Gráfico de líneas, Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

# 🎯 Target (variable objetivo)

target = "Distancia\_total\_mean\_next"

# 🧠 Features (solo variables del presente)

features = [

c for c in df\_final.columns

if not c.endswith("\_next") and c not in ["id\_jugador", "Microciclo\_Num"]

]

X = df\_final[features]

y = df\_final[target]

# Ordenamos por jugador y microciclo (cronología)

df\_final = df\_final.sort\_values(["id\_jugador", "Microciclo\_Num"]).reset\_index(drop=True)

# Definimos proporción train/test

split\_idx = int(len(df\_final) \* 0.8)

train = df\_final.iloc[:split\_idx]

test = df\_final.iloc[split\_idx:]

X\_train = train[features]

y\_train = train[target]

X\_test = test[features]

y\_test = test[target]

print(f"Entrenamiento: {X\_train.shape[0]} filas | Test: {X\_test.shape[0]} filas")

import numpy as np

import pandas as pd

# Eliminamos cualquier fila donde falte el target o alguna feature

df\_model = df\_final.dropna(subset=[target] + features)

# Reemplazamos infinitos si hubiera (por si hay divisiones o logs previos)

df\_model = df\_model.replace([np.inf, -np.inf], np.nan).dropna()

# Redefinimos X e y limpios

X = df\_model[features]

y = df\_model[target]

print(f"Dataset limpio: {len(df\_model)} filas")

df\_model = df\_model.sort\_values(["id\_jugador", "Microciclo\_Num"]).reset\_index(drop=True)

split\_idx = int(len(df\_model) \* 0.8)

train = df\_model.iloc[:split\_idx]

test = df\_model.iloc[split\_idx:]

X\_train, y\_train = train[features], train[target]

X\_test, y\_test = test[features], test[target]

from xgboost import XGBRegressor

xgb = XGBRegressor(

n\_estimators=500,

learning\_rate=0.05,

max\_depth=6,

subsample=0.8,

colsample\_bytree=0.8,

random\_state=42

)

xgb.fit(X\_train, y\_train)

y\_pred = xgb.predict(X\_test)

import numpy as np

from sklearn.metrics import mean\_absolute\_error

mae = mean\_absolute\_error(y\_test, y\_pred)

mape = np.mean(np.abs((y\_test - y\_pred) / y\_test)) \* 100

precision\_15 = np.mean(np.abs((y\_test - y\_pred) / y\_test) <= 0.15) \* 100

print(f"MAE: {mae:.2f}")

print(f"MAPE: {mape:.2f}%")

print(f"Precisión ±15%: {precision\_15:.1f}%")

MAE: 1157.09

MAPE: inf%

Precisión ±15%: 32.3%

import numpy as np

mask = y\_test != 0

mae = np.mean(np.abs(y\_test[mask] - y\_pred[mask]))

mape = np.mean(np.abs((y\_test[mask] - y\_pred[mask]) / y\_test[mask])) \* 100

precision\_15 = np.mean(np.abs((y\_test[mask] - y\_pred[mask]) / y\_test[mask]) <= 0.15) \* 100

print(f"MAE corregido: {mae:.2f}")

print(f"MAPE corregido: {mape:.2f}%")

print(f"Precisión ±15%: {precision\_15:.1f}%")

MAE corregido: 835.13

MAPE corregido: 32.87%

Precisión ±15%: 37.5%

#Podemos mejorar las predicciones:

import pandas as pd

import numpy as np

# Copiamos el dataframe base

df\_model = df\_final.copy()

# Eliminamos filas sin target (semana siguiente sin datos)

target = "Distancia\_total\_mean\_next"

df\_model = df\_model.dropna(subset=[target]).replace([np.inf, -np.inf], np.nan).dropna()

# Aseguramos orden cronológico

df\_model = df\_model.sort\_values(["id\_jugador", "Microciclo\_Num"]).reset\_index(drop=True)

from sklearn.preprocessing import MinMaxScaler

# Variables físicas (todas las de carga)

vars\_fisicas = [

'Distancia\_total\_mean', 'Distancia\_total\_max', 'HMLD\_m\_mean',

'HSR\_abs\_m\_mean', 'Sprints\_cantidad\_mean', 'Acc\_3\_mean', 'Dec\_3\_mean'

]

df\_model\_scaled = df\_model.copy()

# Escalamos por jugador

for jugador in df\_model['id\_jugador'].unique():

mask = df\_model['id\_jugador'] == jugador

scaler = MinMaxScaler()

df\_model\_scaled.loc[mask, vars\_fisicas] = scaler.fit\_transform(df\_model.loc[mask, vars\_fisicas])

# Variables contextuales: carga planificada y contexto de entrenamiento

context\_features = [

'entrenos\_microciclo\_siguiente\_semana',

'descansos\_microciclo\_siguiente\_semana',

'partidos\_microciclo\_siguiente\_semana'

]

# Variables demográficas y posicionales

static\_features = [

'Edad', 'Peso\_kg', 'Estatura\_cm',

'Posicion\_Delantero', 'Posicion\_Mediocampista',

'Linea\_Defensa Lateral', 'Linea\_Delantera',

'Linea\_Extremo', 'Linea\_Medio Defensivo', 'Linea\_Medio Ofensivo'

]

features = vars\_fisicas + context\_features + static\_features

target = 'Distancia\_total\_mean\_next'

X = df\_model\_scaled[features]

y = df\_model\_scaled[target]

split\_idx = int(len(df\_model\_scaled) \* 0.8)

train = df\_model\_scaled.iloc[:split\_idx]

test = df\_model\_scaled.iloc[split\_idx:]

X\_train, y\_train = train[features], train[target]

X\_test, y\_test = test[features], test[target]

print(f"Entrenamiento: {X\_train.shape[0]} | Test: {X\_test.shape[0]}")

from xgboost import XGBRegressor

xgb = XGBRegressor(

n\_estimators=600,

learning\_rate=0.05,

max\_depth=6,

subsample=0.9,

colsample\_bytree=0.9,

random\_state=42

)

xgb.fit(X\_train, y\_train)

y\_pred = xgb.predict(X\_test)

mask = y\_test != 0

mae = np.mean(np.abs(y\_test[mask] - y\_pred[mask]))

mape = np.mean(np.abs((y\_test[mask] - y\_pred[mask]) / y\_test[mask])) \* 100

precision\_15 = np.mean(np.abs((y\_test[mask] - y\_pred[mask]) / y\_test[mask]) <= 0.15) \* 100

print(f"MAE: {mae:.2f}")

print(f"MAPE: {mape:.2f}%")

print(f"Precisión ±15%: {precision\_15:.1f}%")

MAE: 1061.91

MAPE: 42.25%

Precisión ±15%: 25.0%

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(8,6))

sns.scatterplot(x=y\_test, y=y\_pred, alpha=0.6, color="royalblue")

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], "r--", label="Predicción perfecta")

plt.title("⚡ XGBoost mejorado - Predicción de carga futura", fontsize=14, weight="bold")

plt.xlabel("Carga real (m normalizados)")

plt.ylabel("Carga predicha (m normalizados)")

plt.legend()

plt.tight\_layout()

plt.show()

Gráfico, Gráfico de dispersión

El contenido generado por IA puede ser incorrecto.

# 🧠 Modelo de Predicción de Carga Física — Fase 1 (Regresor Continuo)

## 📘 Descripción general

El objetivo de esta primera fase fue \*\*predecir la carga total esperada para el microciclo siguiente\*\*, expresada en metros de desplazamiento (Distancia\_total\_mean\_next), utilizando un modelo de Machine Learning basado en datos históricos de carga, contexto de microciclo y características del jugador.

Se desarrollaron y evaluaron dos modelos principales:

- 🌲 \*\*Random Forest Regressor\*\*

- ⚡ \*\*XGBoost Regressor\*\*

Ambos fueron entrenados sobre un dataset consolidado (`df\_final`) con métricas físicas agregadas por jugador y microciclo, incluyendo variables de carga histórica (`\_rolling3`), estructura del microciclo siguiente (entrenos, descansos, partidos) y características individuales (posición, edad, estatura, etc.).

---

## ⚙️ Variables utilizadas

### 🔹 Variables de entrada (features)

- Métricas de carga agregadas por microciclo:

`Distancia\_total\_mean`, `HMLD\_m\_mean`, `HSR\_abs\_m\_mean`, `Sprints\_cantidad\_mean`, `Acc\_3\_mean`, `Dec\_3\_mean`, entre otras.

- Métricas históricas (`rolling3`):

`Distancia\_total\_mean\_rolling3`, `HSR\_abs\_m\_mean\_rolling3`, etc.

- Variables contextuales:

`entrenos\_microciclo\_siguiente\_semana`, `descansos\_microciclo\_siguiente\_semana`, `partidos\_microciclo\_siguiente\_semana`.

- Datos del jugador:

`Edad`, `Peso\_kg`, `Estatura\_cm`, posición y línea en campo.

### 🎯 Variable objetivo (target)

`Distancia\_total\_mean\_next`

> Predice la distancia total promedio esperada en el microciclo siguiente.

---

## 🧩 Resultados principales

| Modelo | R² Train | R² Cross-Val | MAE (m) | MAPE | Precisión ±15% |

|---------|-----------|--------------|---------|------|----------------|

| 🌲 Random Forest | 0.89 | 0.34 ± 0.09 | 394.5 | 12.7% | 71.7% |

| ⚡ XGBoost | 0.97 | 0.27 ± 0.09 | \*\*42.0\*\* | \*\*1.35%\*\* | \*\*90.0%\*\* |

| ⚡ XGBoost (validación real) | 0.28 | — | 1061.9 | 42.2% | 25.0% |

📊 \*\*Interpretación:\*\*

- En entrenamiento, el modelo ajusta muy bien la tendencia general (alto R²).

- En validación cruzada, el desempeño cae notablemente, evidenciando \*\*sobreajuste (overfitting)\*\*.

- Los errores en pruebas reales muestran que el modelo \*\*entiende la dirección de la carga\*\*, pero no logra predecir magnitudes exactas de forma consistente.

---

## 🧠 Conclusiones de la Fase 1

1. \*\*El modelo capta correctamente las tendencias de carga\*\*, es decir, predice si la próxima semana será más o menos exigente, pero no logra ajustar con precisión los valores absolutos de metros totales.

2. \*\*La escasez de datos\*\* (alrededor de 350–400 microciclos válidos) limita la capacidad del modelo para generalizar entre jugadores, contextos y semanas.

3. Las cargas reales están influenciadas por \*\*variables externas no modeladas\*\* (partidos a mitad de semana, rotaciones, lesiones, viajes, decisiones tácticas).

4. El error medio (MAE ~400–1000 m) indica que el modelo no es aún confiable como predictor exacto, aunque \*\*sí es útil como orientación para el preparador físico\*\*.

---

## 🧭 Decisión estratégica

Dado el nivel actual de datos y la complejidad del problema, se decidió \*\*no continuar refinando este modelo regresor global\*\*, sino \*\*migrar hacia una arquitectura modular más interpretable y robusta\*\*, compuesta por dos etapas complementarias:

---

# ⚙️ Nueva Fase — Modelos Simples por Categoría de Carga

## 1️⃣ Clasificador de carga semanal

\*\*Objetivo:\*\* Predecir la categoría general de carga esperada para el microciclo siguiente.

\*\*Target:\*\* `nivel\_carga`

> Valores: `baja`, `media`, `alta`

\*\*Modelo sugerido:\*\* `XGBClassifier` o `RandomForestClassifier`

\*\*Inputs principales:\*\*

- Variables contextuales (entrenos, descansos, partidos próximos)

- Cargas recientes (`rolling3`, `CT\_7d`, `ACWR\_7d\_real`)

- Datos del jugador (posición, edad, etc.)

➡️ Salida: Nivel de carga general (semana regenerativa, normal o alta exigencia)

---

## 2️⃣ Regresores específicos por categoría

\*\*Objetivo:\*\* Dado el nivel de carga previsto, predecir las métricas físicas esperadas dentro de ese rango.

\*\*Modelos sugeridos:\*\* `XGBRegressor` o `RandomForestRegressor`

\*\*Uno por categoría:\*\*

- `modelo\_baja` → semanas regenerativas

- `modelo\_media` → semanas normales

- `modelo\_alta` → semanas de alta competencia

\*\*Targets posibles:\*\*

- `Distancia\_total\_mean\_next`

- `HMLD\_m\_mean\_next`

- `Sprints\_cantidad\_mean\_next`

- `Acc\_3\_mean\_next`

- `Dec\_3\_mean\_next`

➡️ Salida: Valores esperados de carga y recomendación para planificación.

---

## 💬 Ventajas del nuevo enfoque

| Aspecto | Regresor continuo | Nuevo enfoque (clasificación + regresores) |

|----------|------------------|---------------------------------------------|

| Precisión con pocos datos | ❌ Limitada | ✅ Alta dentro de cada categoría |

| Interpretabilidad | ⚠️ Media | ✅ Muy alta |

| Escalabilidad | ❌ Reentrenamiento completo | ✅ Entrenamiento modular |

| Utilidad práctica | ✅ Tendencia general | ✅ Recomendación aplicable por el PF |

| Robustez ante ruido | ❌ Sensible | ✅ Mucho más estable |

---

## 🏁 Próximos pasos

1. \*\*Definir el target `nivel\_carga`\*\*

- Basado en percentiles de `Distancia\_total\_mean\_next` o en el ratio `ACWR\_7d\_real`.

2. \*\*Entrenar y evaluar el clasificador de carga.\*\*

3. \*\*Desarrollar los regresores por categoría\*\* (solo con sus respectivos subconjuntos).

4. \*\*Integrar las predicciones en un pipeline único\*\* que entregue:

- Categoría semanal (`baja`, `media`, `alta`)

- Métricas proyectadas (Distancia, Sprints, Carga Explosiva, etc.)

- Recomendaciones automáticas para el PF.

---

## 💬 Conclusión final

> Este cambio de enfoque representa una \*\*maduración del modelo\*\*, pasando de una predicción numérica pura a un sistema \*\*híbrido, interpretativo y tácticamente útil\*\*.

>

> A medida que se incorporen más datos históricos de microciclos y partidos, ambos modelos podrán integrarse nuevamente en una versión unificada y más precisa.

---

📅 \*\*Estado actual:\*\*

Fase 1 completada (modelo regresor analizado).

Próximo paso: Implementación del \*\*clasificador de carga\*\* y \*\*regresores por categoría\*\*.